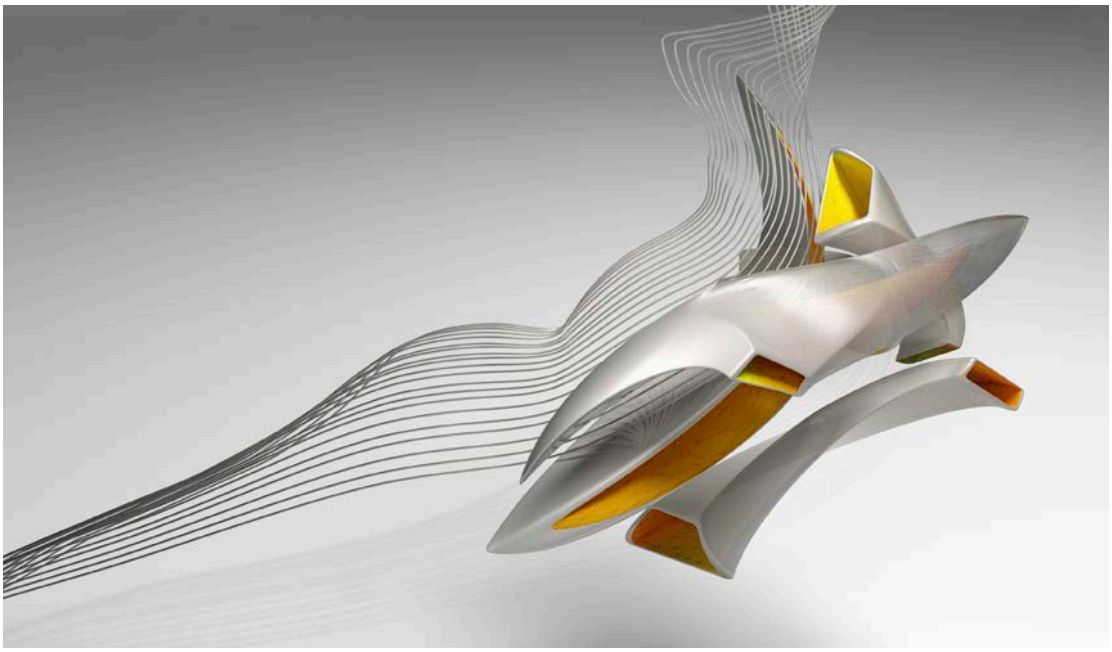
AutoCAD .NET Developer's Guide

AutoCAD .NET 开发指南 2012 版



孙成波 翻译、整理

前言

本开发指南根据 AutoDes 公司网站上的 AutoCAD .NET Developer s uide 2012 版翻译整理。

原始文档网址为
http //exchan e.autodes .com/autocad/enu/online help/browse#WS73099cc142f48755a52158612bd434e551 7fd5.htm。

主要内容包括 AutoCAD .NET API 介绍、使用 AutoCAD .NET API 控制 AutoCAD 工作环境、创建及编辑 AutoCAD 实体、创建和编辑尺寸标注、3D 图形处理、定义打印布局、打印输出、处理 AutoCAD 事件，以及使用 Microsoft Visual Studio 的错误处理、应用程序发布等。涵盖了 AutoCAD .NET 二次开发所涉及的所有基本任务。

在文档翻译后期，Autodes 公司发行了 AutoCAD2014 版。因此译者将指南中的示例代码在 Microsoft Visual Studio2010SP1 AutoCAD2014 环境下进行了测试（只测试了 C#版代码），并将测试过程中发现的 AutoCAD .NET API 2014 版与 AutoCAD .NET API 2012 版的不同之处在本指南内作了标注。

由于译者英语水平及 AUtoCAD 二次开发水平和经验有限，肯定存在描述不准确的地方，
大 指正并提出 意见。

本文档是 **free** 的，你可以自由地下载、打印、分享。

示例代码（C#）下载地址：http //download.csdn.net/detail/sunchen bo2007/6903599

成

2014 2 8

目录

第 0 章 AutoCAD .NET API 介绍

本章描述了通过托管.NET 应用程序编程接口（API）公开的 AutoCAD®对象的概念。AutoCAD .NET API 可以让我们的一些操作任务自动化，如创建和修改保存在图形文件数据库里的对象，或者修改自定义文件的内容等。

本指南涵盖了 Microsoft® Visual Studio® 2010，以及 AutoCAD .NET API 开发使用的编程语言 Microsoft® Visual Basic® .NET（简称为 VB.NET） 和 Microsoft® Visual C#®。

本章主要内容：

0.1 本指南主要内容

本指南提供了如何使用 AutoCAD.NET API 及如何使用 Microsoft Visual Studio 和 VB.NET、C#等编程语言进行 AutoCAD 二次开发的知识。关于使用 Microsoft Visual Studio 开发应用程序的专门知识，参见（附录§A *Microsoft Visual Studio 使用入门*）和（§8 *使用 VB.NET 和 C#开发应用程序*）。如果你没有使用 Microsoft Visual Studio 进行.NET 框架下的应用开发，可以跳过这两部分内容。不过，本指南里的所有示例代码使用的都是 VB.NET 和 C#这两种语言。

0.2 AutoCAD .NET API 概览

使用 AutoCAD.NET API 提供的程序集，我们可以通过编程对 AutoCAD 和图形文件进行操作。并且可以使用许多不同的编程语言和开发环境。

在 AutoCAD 中实现 .NET API 的几大优点:

- 更多的编程环境可以编程访问 AutoCAD 图形。在 .NET API 出现之前，开发人员只能局限于使用 ActiveX® 自动操作及支持 COM 的语言、AutoLISP®和 ObjectARX 的 C++。
- 通过使用应用程序本地化的 .NET API 或公开的 ActiveX/COM 库与其它 Windows® 应用程序(例如 Microsoft Excel 和 Word)共享数据比以前更方便了。
- .NET 框架是为 32 位及 64 位操作系统而设计，而 VBA 只为 32 位操作系统设计。
- 在使用高级编程接口方面，和传统编程语言如 C++相比，.NET 具有较低的学习曲线。

对象是 AutoCAD .NET API 的主要构造块。每一个公开的对象均精确代表一个 AutoCAD 组件，它们之间又组成了不同的程序集和命名空间。AutoCAD .NET API 有许多不同类型的对象。例如：

- 直线、圆弧、文字和标注等图形对象都是对象。
- 文字与标注样式等样式设置都是对象。
- 图层、组合和块等组织结构都是对象。
- 视图和视口等图形显示都是对象。
- 图形、AutoCAD 应用程序本身也是对象。

0.3 AutoCAD .NET API 的组件

AutoCAD .NET API 由不同的 DLL 文件组成，这些 DLL 文件包含有大量的类、结构、方法及事件，用于访问图形文件对象或 AutoCAD 程序对象。每个 DLL 文件定义了不同的命名空间，这些命名空间按功能组织 API 库组件。

常用的 AutoCAD .NET API 的三个主要 DLL 文件是：

- **AcDbMgd.dll** 当处理图形文件中存储的对象时引用；
- **AcMgd.dll** 当处理 AutoCAD 应用程序和用户接口时引用；
- **AcCui.dll** 当处理自定义文件时引用；

（**注：**AutoCAD 2014 版拆分出来一个 **AcCoreMgd.dll**，这样主要 DLL 文件变成了 4 个。）

- **AcCoreMgd.dll** 当处理编辑器、发布与打印、定义 AutoLISP 命令和函数时引用。

使用 AutoCAD .NET API 的 DLL 文件

使用 AutoCAD.NET API 相关 DLL 文件里提供的类、结构、方法及事件之前，必须在你的 Visual Studio 工程中引用相应的 DLL 文件。完成引用后，你就可以在工程中使用该 DLL 文件里定义的命名空间和 API 组件。

一旦引用了 AutoCAD.NET API DLL 文件，应将该引用的“复制本地”属性设置为 False（设置方法：解决方案资源管理器->右键该引用->属性->复制本地->选 False）。“复制本地”属性确定当 Microsoft Visual Studio 编译工程时是否创建一个引用文件的副本并将它放在与工程的程序集文件相同的目录下。 由于 AutoCAD 安装目录里包含有相同文件名的 DLL 文件， 因此，如果创建了引用文件副本，当加载程序集文件到 AutoCAD 中时可能会引发意想不到的结果。

AutoCAD .NET API DLL 文件的存放位置

AutoCAD .NET API 的 DLL 文件位于 AutoCAD 安装目录里（*<盘符>:\program Files\AutoCAD 2012*），同时，作为 *AutoCAD 2012 ObjectARX SDK* 开发包的一部分，可以从 http://www.objectarx.com 下载，或从 AutoCAD 开发者网站下载（http://www.autodesk.com/adn）。

安装 ObjectARX SDK 开发包后，可以在主安装目录下的 *inc* 目录下找到 AutoCAD .NET API 的 DLL 文件。

注： ObjectARX SDK 开发包里的.NET API DLL 文件是 AutoCAD 软件包里相应 DLL 文件的简化版本，不包含依赖 AutoCAD 用户界面的相关内容。推荐下载并安装 ObjectARX SDK 开发包，在工程中引用开发包里的 DLL 文件，而不引用 AutoCAD2012 安装目录里的文件。

操作步骤

◆ 下载并安装 AutoCAD 2012 ObjectARX SDK 开发包

1. 打开浏览器，进入 http://www.objectarx.com.
2. 在打开的网页上单击 License & Download.
3. 填写需要的表格并选择 **ObjectARX for AutoCAD 2012**， 单击 Submit.
4. 在下载页面，单击 Download Now 使用下载管理器下载，或单击 Standard Download Method 使用浏览器的默认下载方法下载。
5. 单击 Save 或用来保存文件到本地硬盘的选项。
6. 指定保存位置，下载 ObjectARX SDK 开发包文件。
7. 下载完后，打开下载文件所在目录，双击下载的文件。

出现安装向导。

8. 在 ObjectARX *〈Release〉*对话框里，指定一个新安装位置，或使用默认安装位置，单击 Install。

   如果没有遇到问题，安装完成后，会关闭安装向导。

◆ 安装 Managed .NET 工程向导

1. 打开浏览器，进入 http://www.autodesk.com/developautocad。
2. 下载并解压缩 AutoCAD 2012 .NET Wizards.zip。
3. 双击其中的安装文件 AutoCAD 2012 dotNET Wizards.msi。
4. 在 AutoCAD .NET Wizards 对话框里单击 Next。
5. 在 Select Installation Folder 页，单击 Browse 给向导指定一个新安装位置或保留默认位置，单击 Next。
6. 再单击 Next 确认向导安装。
7. 单击 Close 完成安装。

◆ 引用 AutoCAD .NET API 的 DLL 库文件

1. 进入 Microsoft Visual Studio，打开或新建一个项目。如果界面没有打开解决方案资源管理器的话，单击**视图** 菜单 ➤ **解决方案资源管理器** ，显示解决方案资源管理器。
2. 单击解决方案资源管理器上方工具条中的**显示所有文件**。
3. 右键单击**引用**节点选择**添加引用**。
4. 在添加引用对话框的**浏览**选项卡中，浏览并选择包含你要用的库的 DLL 文件，单击**确定**。
5. 在解决方案资源管理器，单击**引用**节点前的加号，展开节点。
6. 找到你刚添加的引用文件，右键单击它，选择**属性**。
7. 在属性窗口，单击**复制本地**字段，从后边的下拉列表中选择 False。

0.4 Microsoft Visual Studio 概览

Microsoft Visual Studio 是不依赖 AutoCAD 运行的面向对象程序开发环境。虽然 Microsoft Visual Studio 是 AutoCAD 及其他应用程序的外部环境，但它能够与那些公开了自己的本地.NET API 库或 ActiveX/COM 库的应用程序很好地进行交互。

0.4.1 Microsoft Visual Studio 的版本选用

Microsoft Visual Studio 有多个版本可用。开发 AutoCAD 2012 .NET API 项目，我们需要使用：

- Microsoft Visual Studio 2010
- Microsoft .NET Framework 4.0

注：由 Microsoft Visual Studio 2008 编译的、目标为 Microsoft .NET Framework 3.5 的项目可以加载到 AutoCAD2012，运行没有问题。但 Microsoft Visual Studio 2008 不能用来调试 AutoCAD 2012 加载的项目，调试项目时必须使用 Microsoft Visual Studio 2010。

如果使用的是 AutoCAD 2010 或 AutoCAD 2011，则应选择：

- Microsoft Visual Studio 2008 with Service Pack 1
- Microsoft .NET Framework 3.5 with Service Pack 1

如果使用的是 AutoCAD 2007 到 AutoCAD 2009，则应选择：

- Microsoft Visual Studio 2005
- Microsoft .NET Framework 2.0 or later （或 2.0 以上）

注：译者了解到，AutoCAD 2014 仍然是使用 vs2010sp1 编译的，所以
如果使用的是 AutoCAD 2013 和 AutoCAD 2014，则应选择：

- Microsoft Visual Studio 2010 with Service Pack 1
- Microsoft .NET Framework 4.0

期待 AutoCAD 2015 有一个进步。

Microsoft Visual Studio 有两种版本供我们选择：免费版本和付费版本。免费版本即众所周知的 Microsoft Visual Studio 2010 Express 版，付费版本的名称和价格因整合的开发工具的不同而异。Microsoft Visual Studio 2010 专业版提供了比 Microsoft Visual Studio 2010 Express 版改进了的调试性能及许多其他特性。开发人员最常用的 Microsoft Visual Studio 版本就是 Microsoft Visual Studio 2010 专业版。

注：可能有人使用 Microsoft Visual Studio Express 版进行 AutoCAD .NET API 开发，但本指南假设你使用的是 Microsoft Visual Studio 2010 专业版或 Microsoft Visual Studio 2010 高级版。

使用 Microsoft Visual Studio 有四大优势：

- 强大易用的开发环境，适度的学习曲线；
- VBA 和 VB.NET 语法相近，是现有 VBA 开发人员理想的环境；
- 直观、丰富的对话框创建工具；
- 可以将项目生成为独立的可执行程序或 DLL 程序集，加载到 AutoCAD 执行；

注：不像 VBA 项目，.NET 应用程序加载并运行在 64 位 AutoCAD 环境时性能不会下降。

更多关于 Microsoft Visual Studio 不同版本的信息，见 http://www.microsoft.com/vstudio 及 http://www.microsoft.com/express。

0.4.2 与.NET 一起使用 COM 互操作

Microsoft Visual Studio 可以在同一个项目中使用本地.NET 和 COM 两种接口，这样我们就可以移植现有的那些用 VB6 或 VBA 写的代码而不必完全重写。要想在 Microsoft Visual Studio 创建的项目中访问 AutoCAD 自动化对象，需要建立对下列文件的引用：

- AutoCAD 2012 类库文件 acax18enu.tlb，位于 C:\ *Program Files\Common Files\Autodesk Shared*;
- AutoCAD/ObjectDBX18.0 通用类库文件 axdb18enu.tlb，位于 C:\ *Program Files\Common Files\Autodesk Shared*;

注：上述类库文件也可以从 ObjectARX SDK 中得到。

引用这些类库可以获得对下列基本互操作程序集的使用：

- Autodesk.AutoCAD.Interop.dll (AutoCAD 专有类型)
- Autodesk.AutoCAD.Interop.Common.dll (ObjectDBX™宿主应用程序共享类型)

互操作程序集位于全局程序集缓存中，他们将自动化对象映射为相应的.NET 对等对象。

引用了这些类库，我们就可以在 Microsoft Visual Studio 中声明基于 AutoCAD 的变量，像下面的例子：

VB.NET

```
Dim objAcApp As Autodesk.AutoCAD.Interop.AcadApplication
Dim objLine As Autodesk.AutoCAD.Interop.Common.AcadLine
```

C#

```
Autodesk.AutoCAD.Interop.AcadApplication objAcApp;
Autodesk.AutoCAD.Interop.Common.AcadLine objLine;
```

应用互操作程序集使得将 VBA 项目转换为 VB.NET 项目更容易。不过，要想充分利用.NET 和 AutoCAD .NET API 提供的全部功能，还是需要重写现有 VBA 代码。

创建并引用 AutoCAD 应用程序

AutoCAD 2012 .NET 应用程序可以和 AutoCAD 自动化项目一样利用相同的类库（*acax18enu.tlb*）。该类库位于 c:*Program Files\Common Files\Autodesk Shared*。

AutoCAD 2012 .NET 应用程序的 CreateObject、GetObject 和 GetInterfaceObject 函数也

使用同样的依赖版本的 ProgID 值。例如，CreateObject("AutoCAD.Application.18")允许我们创建一个 AutoCAD 的实例，并获取代表应用程序新示例的对象。

0.4.3 依赖和限制

不像 ActiveX 自动化项目，.NET 项目在其他程序安装、重装及卸载时很少出现库文件冲突的情形。很少出现兼容问题的原因在于，.NET Framework 是一个标准化的平台。不过，还是会遇到依赖性问题。要避免.NET Framework 依赖性问题，应确保使用与 AutoCAD2012 所用的.NET Framework 版本相同或更早的版本来开发 VB.NET 或 C#项目。

更多信息参见（§0.4.1 *Microsoft Visual Studio 的版本选用*）。

0.5 更多内容

本指南假设你会使用 VB.NET 或 C#编程语言，因此，本指南不想将这两种语言的丰富的文档资料复制过来放在这里。如果你需要关于 VB.NET 或 C#编程语言的更多内容，以及关于使用 Microsoft Visual Studio 的更多内容，可参见 Microsoft Visual Studio 的帮助系统。

0.6 示例代码

本指南及 AutoCAD 2012 ObjectARX SDK 开发包里，提供了大量的示例项目、子程序和函数，用以演示 AutoCAD .NET API 中提供的类、结构、方法、属性和事件。

你还可以从 AutoDesk 公司网站（http://www.autodesk.com/developautocad）上找到一些演示 AutoCAD .NET API 各种功能的示例项目。 这些示例项目，从提取 AutoCAD 图形数据到 Microsoft Excel 电子表格，到绘制电力传输塔并对其进行应力分析，展示了广泛的功能。

许多这些示例都展示了如何将 VB.NET 和 C#编程语言的各个方面功能与给力的 AutoCAD .NET API 结合起来创建用户应用程序。

另外，你还可以拷贝本指南里的示例代码，直接粘贴到 Microsoft Visual Studio 的代码编辑窗口，然后编译并加载到 AutoCAD。这些代码的开头都有所需的命名空间语句，拷贝时也应将所需的这些代码添加到代码窗口的开头处。

注：为了保持概念简单、易于阅读，指南里的示例代码只包含有限的错误处理。将这些示例代码应用于你的项目时，应检查并适当添加额外的错误处理。关于错误处理的更多内容，参见（§8.1 *处理错误*）。

◆ 运行本指南里的示例代码

1. 打开 Microsoft Visual Studio，新建一个类库项目，按前面（§0.3）介绍的方法添加引用。
2. 在打开的代码编辑窗口添加适当的命名空间，将示例代码拷贝到类定义里。
3. 单击 Microsoft Visual Studio 的**生成**菜单，单击**生成**<*项目名称*>来生成项目（或直接按 Shift+F6）。
4. 进入 AutoCAD，在命令提示栏输入 **netload** 并回车。
5. 在选择 .NET 程序集对话框里，选择生成的程序集文件，单击**打开**。
6. 在 AutoCAD 命令提示栏，输入在所加载的程序集里定义的命令或 AutoLISP 函数的名称及所需参数。

0.7 ActiveX Automation 到 .NET 的转换

AutoCAD 2012 继续支持 VBA，不过必须单独下载并安装 VBA 组件。下载地址为 http://www.autodesk.com/vba-download。

虽然到目前为止还没有发行过不支持 VBA 的 AutoCAD 版本，AutoDesk 公司还是建议开始对现有的 VBA 项目进行移植，确保一旦对 VBA 的支持“下架”，你已做好准备。并且，AutoDesk 公司建议，对于新的应用项目，请使用 Microsoft Visual Studio 和 AutoCAD .NET API、AutoLISP 来开发，或者使用 C++ 和 ObjectARX 来开发。

你可以继续使用.NET 中的 AutoCAD COM Automation 库，帮助你更容易地从 VBA 转换到 VB.NET，参见（§0.4.2 *与.NET 一起使用 COM 互操作*）。

如果你是.NET 方面的新手， 参见（附录 A *Microsoft Visual Studio 使用入门*），那里有关于使用 Microsoft Visual Studio 和 AutoCAD .NET API 的基础知识。

第 1 章 AutoCAD .NET API 基础

要想有效利用 AutoCAD® .NET API 编程接口，必须熟悉与 AutoCAD 自动化任务相关的 AutoCAD 实体、对象和特性等概念。对于这些对象的图形属性和非图形属性了解的越多，使用 AutoCAD® .NET API 对他们进行操作也就越容易。AutoCAD .NET API 对象各成员（方法、函数、属性等）的详细信息，请参见《AutoCAD .NET 参考指南》。

本章主要内容：

- 了解 AutoCAD 对象层次
- 访问对象层次
- 集合对象
- 了解属性和方法
- 进程外与进程内对比
- 定义命令和 AutoLISP 函数

1.1 了解 AutoCAD 对象层次

对象是 AutoCAD .NET API 的主要构成成分。每个公开的对象都准确地代表 AutoCAD 的一个部件。AutoCAD .NET API 拥有许多不同类型的对象，例如：

- 直线、圆弧、文字和标注等图形对象；
- 图层、线型、标注样式等样式设置对象；
- 图层、组、块等图形组织结构对象；
- 视图、视口等图形显示对象；
- 乃至图形和 AutoCAD 应用程序本身也是对象；

所有对象以 AutoCAD 的 Application 对象为根对象，按层次结构方式组织，通常称层次结构为对象模型。下图所示表示了 Application 对象与 BlockTableRecord 模型空间内实体（Entity）的基本关系。这里只列出了 AutoCAD .NET API 中的部分对象。

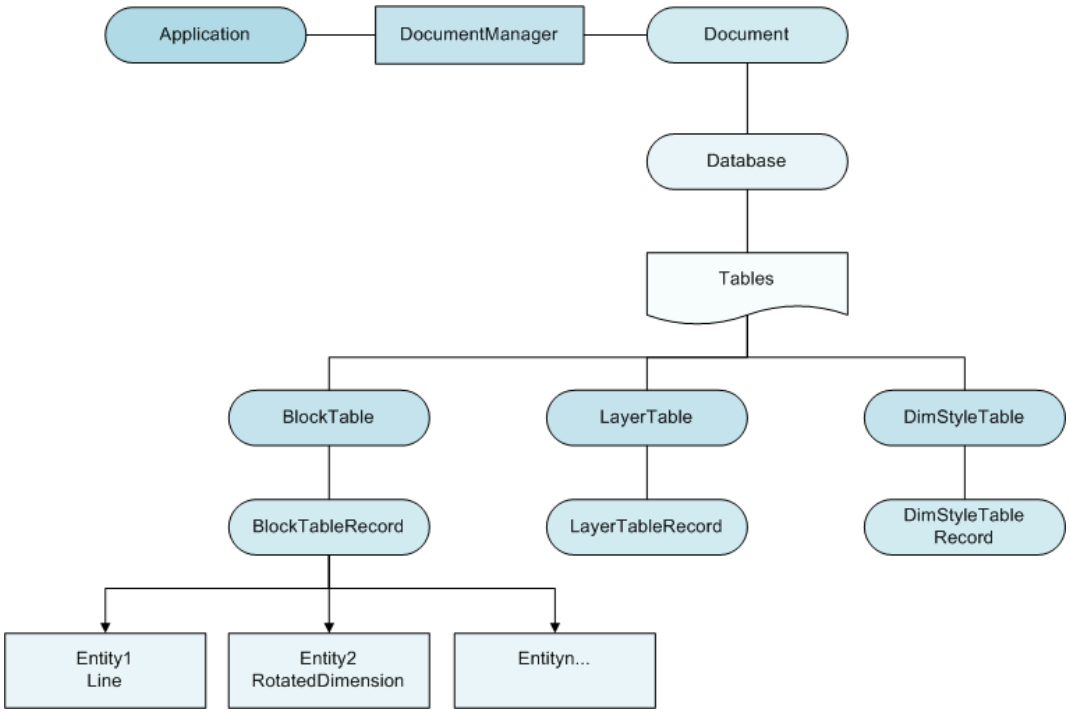


图 1-1 AutoCAD 对象层次示意

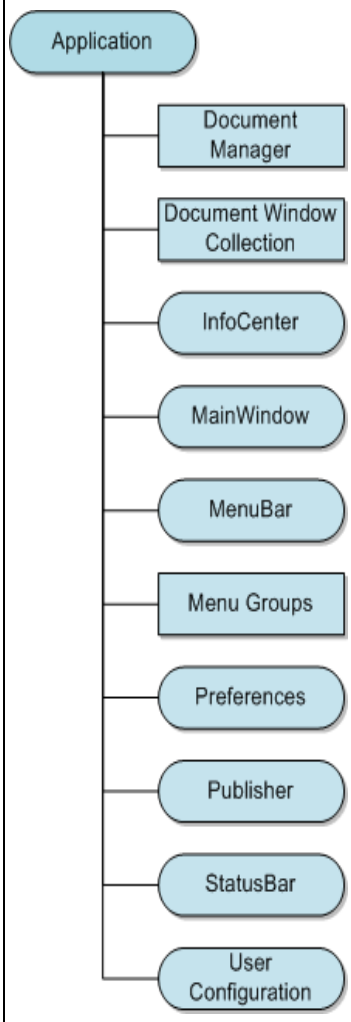
1.1.1 Application 对象

Application 对象是 AutoCAD .NET API 的根对象，从 Application 对象，可以访问 AutoCAD 主窗口，以及任何打开的图形，并进而访问图形里的各个对象。有关处理打开图形文档的信息，参见（§1.1.2 *Document 对象*）。

例如，Application 对象有一个 DocumentManager 属性，用来返回 DocumentCollection 对象，该对象提供了访问当前打开的 AutoCAD 图形的功能，并允许你创建、保存、打开图形文件。Application 对象的其他属性提供了访问应用程序特有数据的功能，像信息中心 InfoCenter、主窗口、状态栏等。MainWindow 属性允许访问应用程序的名称、主窗口大小、位置及可见性等。

Application 对象的大多数属性允许对 AutoCAD .NET API 里的对象进行访问，同时也有一些属性是对 AutoCAD ActiveX® Automation 里的对象（COM 对象）的引用，这些属性包括应用程序对象的 COM 版本(AcadApplication)、菜单栏(MenuBar)、加载的菜单组(MenuGroups)、以及选项设置（Preferences）等。

图 1-2 Application 对象层次示意



DocumentManager

所有文档对象的容器（每个打开的图形对应一个文档对象）

DocumentWindowCollection

所有文档窗口对象的容器（DocumentManager 里每个文档对象有一个文档窗口对象）

InfoCenter

包含对信息中心工具栏的引用

MainWindow

包含对 AutoCAD 应用程序窗口对象的引用

MenuBar

包含对 AutoCAD 菜单栏对应的 MenuBar COM 对象的引用

MenuGroups

包含对 MenuGroups COM 对象的引用，菜单组对象包含每个已加载 CUIx 文件的自定义组名

Preferences

包含对 Preferences COM 对象的引用，该对象允许你在选项（Options）对话框里修改许多设置

Publisher

包含对 Publisher 对象的引用，该对象用于发布图形

StatusBar

包含对应用程序窗口状态栏对象(StatusBar 对象)的引用

UserConfiguration

包含对 UserConfiguration 对象的引用，该对象允许你处理用户保存的配置信息

1.1.2 Document 对象

Document 对象，实际上就是一个 AutoCAD 图形，是 DocumentCollection 对象的一部分，提供了访问与 Document 对象相关联的 Database 对象的功能。Database 对象包含 AutoCAD 的全部图形对象和大部分非图形对象。

关于 Database 对象的更多信息参见（§1.1.3 *Database 对象*）。

Document 对象连同 Database 对象一起，提供了对图形状态栏、图形窗口、编辑器（Editor）及事务管理器（TransactionManager）对象的访问。Editor 对象提供了获取用户输入的功能，用户输入的形式可以是一个点、键入的一个字串或数值等。有关获取用户输入方面的更多内容，参见（§2.8 *提示用户输入*）。

事务管理器对象在被称为事务（transaction）的单一操作中，用来管理多个数据库对象。事务可以嵌套，当与事务打交道时，你可以提交或终止所做的修改。关于事务和事务管理器对象的更多内容，请参见（§3.1.2 *使用事务管理器管理事务*）。

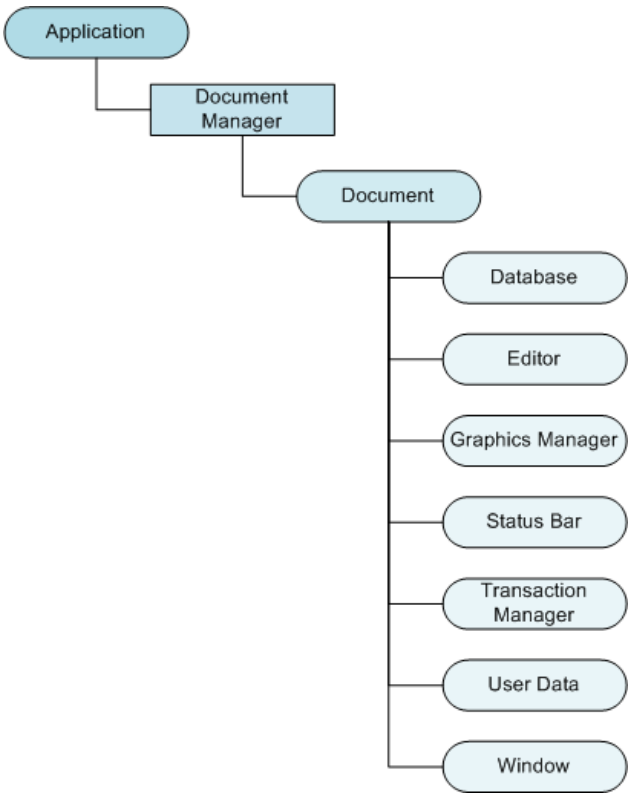


图 1-3 Document 对象层次示意

1.1.3 Database 对象

Database 对象包含 AutoCAD 所有的图形对象和决大部分非图形对象，其中包括实体(图元)、符号表、命名字典等。实体（图元）表示图形里的图形对象，直线、圆、弧线、文字、填充和多义线等都是实体。用户能在屏幕上看到实体并可以对其进行操作。

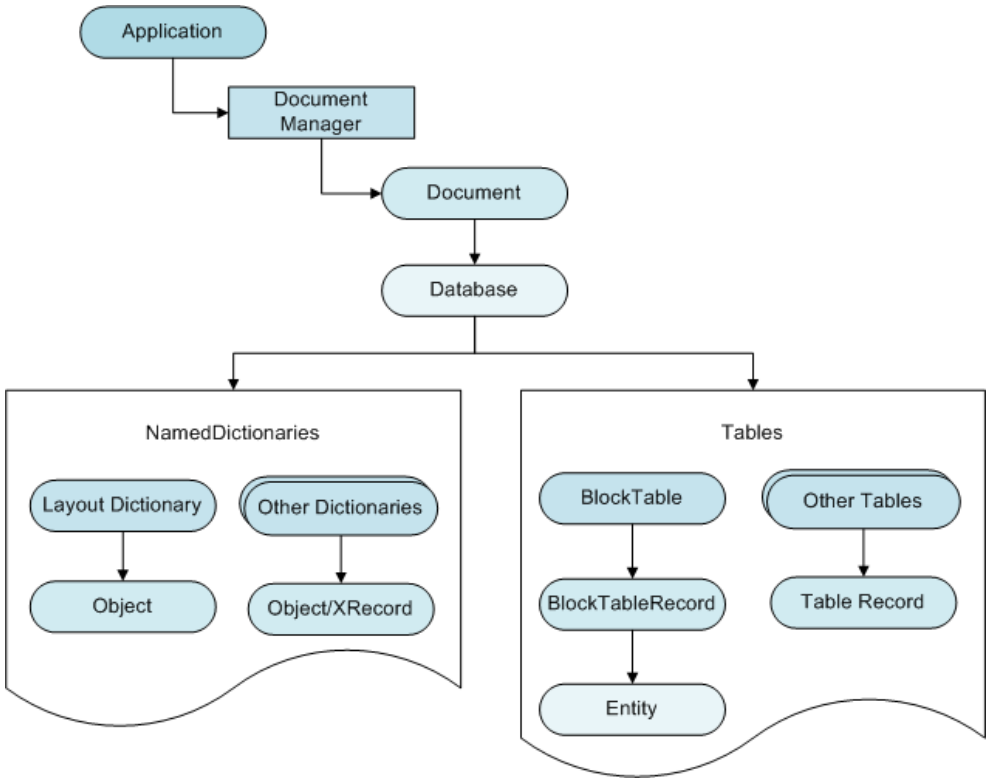


图 1-4 Database 对象层次示意

可通过 Document 对象的 Database 成员属性来访问当前文档的 Database 对象：

```
Application.DocumentManager.MdiActiveDocument.Database
```

符号表和字典

符号表和字典用来访问非图形对象（块、图层、线型、布局，等等）。图形文件中字典的个数会因 AutoCAD 应用程序的特点和类型而变化，而每个图形文件都包含有 9 个固定的符号表。不能往数据库里添加新的符号表。

符号表的例子，如图层表（*LayerTable*），其中包含图层表记录，还有块表（*BlockTable*），其中包含块表记录，等。所有的图形实体（线、圆、弧等等）都属于一个块表记录。缺省情

况下，任何图形文件都包含为模型空间和图纸空间预定义的块表记录。每个图纸空间布局拥有自己的块表记录。更多关于符号表的内容，见（§1.3 *集合对象*）。

字典是可以包含任何 AutoCAD 对象或 XRecord 的容器对象。字典要么保存在命名字典对象数据库里，要么作为一个表记录或图形实体的扩展字典来保存。命名字典对象是与数据库关联的所有字典的主表。与符号表不同，新的字典可以被创建并添加到命名字典对象里。更多关于字典的内容，见（§1.3 *集合对象*）。

注：字典对象不包含图形实体。

VBA/ActiveX 交叉参考

AutoCAD .NET API 中的 Database 对象与 ActiveX Automation 库中的 Document 对象类似。必须使用.NET API 中的 Document 对象和 Database 对象，才能访问 ActiveX Automation 库中 Document 对象的大多数可用属性。
关于.NET API 的 Document 对象的更多内容，参见（§1.1.2 *Document 对象*）。

1.1.4 图形对象和非图形对象

图形对象，又称为实体，是组成图形的可见对象（线、圆、光栅图像，等等）。向当前图形添加图形对象的方法是，通过引用正确的块表记录，使用 AppendEntity 方法将要添加的新对象添加到图形中。

要修改或查询对象，先从相应的块表记录里获得对该对象的引用，然后调用该对象自己的方法或属性。每个图形对象都拥有一些方法，这些方法实现了与大部分 AutoCAD 编辑命令相同的功能，像复制、删除、移动、镜像，等等。

这些对象还有一些方法，用于检索扩展数据（xdata）、突出显示和取消突出显示、从别的实体设置属性等。大多数图形对象都拥有一些彼此共有的属性，比如 LayerId、LinetypeId、Color、Handle 等。另外每个图形对象还拥有自己特有的属性，比如 Center、StartPoint、Radius，以及 FitTolerance 等。

非图形对象是图形中那部分不可见的（信息性质的）对象，像图层、线型、标注样式、文字样式，等等。要新建一个符号表记录，需调用该符号表的 Add 方法。要向命名对象字典添加一个字典，需调用 SetAt 方法。要修改或查询这些对象，调用这些对象自己相应的方法或属性。每个非图形对象都拥有特定功能的方法和属性，所有非图形对象都拥有检索扩展数据、删除自己的方法。

1.1.5 集合对象

AutoCAD 使用集合或容器对象对大部分图形对象和非图形对象进行了分组。尽管集合包含不同的数据类型，仍可用相似的技术对它们进行处理。每个集合都拥有把一个对象添加到集合

里的方法和从集合里获取一个集合项的方法。大多数集合调用 Add 方法或 SetAt 方法将一个对象添加到集合里。

大多数集合都提供类似的方法和属性，以便于使用和学习。Count 属性返回集合里从 0 开始的对象计数，Item 函数从集合里返回一个对象。AutoCAD .NET API 中的集合成员举例如下：

- 图层符号表中的图层表记录；
- ACAD_LAYOUT 字典中的布局；
- 文档集合中的文档；
- 块参考中的属性；

1.1.6 非本地的图形对象和非图形对象

AutoCAD .NET API 编程接口是 ObjectARX 和 ActiveX Automation 两种编程接口交叉实现的。当然你可以从 ObjectARX 访问 ActiveX Automation，但通过.NET API，你可以几乎无缝的使用这两种编程技术。正像你使用本地.NET API 处理对象一样，你可以从属性访问相同的 COM 对象。有些情况下，使用 COM 对象是编程访问 AutoCAD 功能的唯一途径。COM 对象通过.NET API 公开的属性的例子如：Preferences、Menubar、MenuGroups、AcadObject 及 AcadApplication 等。

注：当与 COM 对象打交道时，请确认你引用了 AutoCAD2012 的类型库。关于 COM 互操作的信息见（§0.4.2 *与 .NET 一起使用 COM 互操作*）。

Application 对象的 Preferences 属性用来访问一组 COM 对象，其中的每个 COM 对象对应选项对话框里的一个选项页，这些对象一起用来访问选项对话框里呈现出来的存储在注册表里的全部选项设置。你可以使用 Application 对象的 SetSystemVariable 方法和 GetSystemVariable 方法来设置和修改选项（当然也可以设置和修改哪些不在选项对话框里的系统变量）。使用 Preferences 对象的更多信息，见 *ActiveX 和 VBA 开发指南*。

当你与最初可能是用 VB 或 VBA 开发的代码打交道时，或者当你与那些使用了 AutoCAD ActiveX Automation 库和 AutoCAD .NET API 的第三方库打交道时，访问 COM 对象技术很有用。像 Preferences 对象那样，你还可以使用 Utility 对象的诸如转换坐标、基于角和距离定义新点等实用功能，Utility 对象可通过 COM 对象 AcadApplication 访问，同等地，也可以通过.NET API 中的 Application 对象访问。

注意：当使用了 AutoCAD .NET API 和 ActiveX Automation 两种技术时，如果你创建的自定义函数需要返回一个对象，建议返回 ObjectId 而不是对象本身，见（§3.1.1 *使用对象 ID*）。

1.2 访问 AutoCAD 对象层次

AutoCAD .NET API 中，Application 对象是根对象，我们一般会与当前图形文档的数据库打交道。Application 对象的 DocumentManager 属性允许我们使用它的 MdiActiveDocument 属

性来访问当前文档。从 MdiActiveDocument 属性返回 Document 对象，我们可以用 Document 对象的 Database 属性访问文档的数据库。

VB.NET

```
Application.DocumentManager.MdiActiveDocument.Database.Clayer
```

C#

```
Application.DocumentManager.MdiActiveDocument.Database.Clayer;
```

VBA/ActiveX 代码参考

```
ThisDrawing.ActiveLayer
```

1.2.1 引用对象层次中的对象

我们使用.NET API 中的对象时，可以直接引用某些对象，或者通过基于所用对象的用户自定义变量引用该对象。要直接引用一个对象，只要在调用层次中包含该对象即可。例如，下面的代码将一个图形文件附加到当前图形的数据库中。注意其调用层次是从 Application 对象开始一直到 Database 对象，最后在 Database 对象上调用 AttachXref()方法：

VB.NET

```
Dim strFName As String, strBlkName As String
Dim objId As Autodesk.AutoCAD.DatabaseServices.ObjectId

strFName = "c:\clients\Proj 123\grid.dwg"
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName)

objId =
Application.DocumentManager.MdiActiveDocument.Database.AttachXref(strFName,
strBlkName)
```

C#

```
string strFName, strBlkName;
Autodesk.AutoCAD.DatabaseServices.ObjectId objId;

strFName = "c:/clients/Proj 123/grid.dwg";
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName);

objId =
Application.DocumentManager.MdiActiveDocument.Database.AttachXref(strFName,
strBlkName);
```

要通过用户自定义变量引用对象，首先定义一个所需类型的变量，然后将相应对象赋值给该变量。例如下列代码定义了一个 Autodesk.AutoCAD.DatabaseServices.Database 类型的变量 acCurDb，并将当前数据库赋值给该变量：

VB.NET

```
Dim acCurDb As Autodesk.AutoCAD.DatabaseServices.Database
acCurDb = Application.DocumentManager.MdiActiveDocument.Database
```

C#

```
Autodesk.AutoCAD.DatabaseServices.Database acCurDb;
acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
```

接下来下面的代码使用自定义变量 acCurDb 将一个图形文件添加到数据库：

VB.NET

```
Dim strFName As String, strBlkName As String
Dim objId As Autodesk.AutoCAD.DatabaseServices.ObjectId

strFName = "c:\clients\Proj 123\grid.dwg"
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName)

objId = acCurDb.AttachXref(strFName, strBlkName)
```

C#

```
string strFName, strBlkName;
Autodesk.AutoCAD.DatabaseServices.ObjectId objId;

strFName = "c:/clients/Proj 123/grid.dwg";
strBlkName = System.IO.Path.GetFileNameWithoutExtension(strFName);

objId = acCurDb.AttachXref(strFName, strBlkName);
```

从模型空间检索实体对象

下面示例代码返回模型空间中所有实体对象的列表。类似代码还可以对图纸空间的实体做相同的事情。注意所有图形对象都是实体对象：

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
```

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ListEntities")> _
Public Sub ListEntities()
  '' 获取当前数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以读模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForRead)

      Dim nCnt As Integer = 0
      acDoc.Editor.WriteMessage(vbLf & "Model space objects: ")

      '' 遍历模型空间里的每个对象，并显示找到的对象的类型
      For Each acObjId As ObjectId In acBlkTblRec
          acDoc.Editor.WriteMessage(vbLf & acObjId.ObjectClass().DxfName)

          nCnt = nCnt + 1
      Next

      '' 如果没发现对象则显示提示信息
      If nCnt = 0 Then
          acDoc.Editor.WriteMessage(vbLf & "  No objects found")
      End If

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```
[CommandMethod("ListEntities")]
public static void ListEntities()
{
  // 获取当前数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以读模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForRead) as BlockTableRecord;

      int nCnt = 0;
      acDoc.Editor.WriteMessage("\nModel space objects: ");

      // 遍历模型空间里的每个对象，并显示找到的对象的类型
      foreach (ObjectId acObjId in acBlkTblRec)
      {
          acDoc.Editor.WriteMessage("\n" + acObjId.ObjectClass.DxfName);

          nCnt = nCnt + 1;
      }

      // 如果没发现对象则显示提示信息
      if (nCnt == 0)
      {
          acDoc.Editor.WriteMessage("\n No objects found");
      }

      // 关闭事务
  }
}
```

⊟ VBA/ActiveX 代码参考

```
Sub ListEntities()
    ’ 本例返回模型空间里的第一个实体；
```

```
    On Error Resume Next
    Dim entity As AcadEntity
    If ThisDrawing.ModelSpace.count <> 0 Then
        Set entity = ThisDrawing.ModelSpace.Item(0)
        MsgBox entity.ObjectName + _
               " is the first entity in model space."
    Else
        MsgBox "There are no objects in model space."
    End If
End Sub
```

1.2.2 访问 Application 对象

Application 对象位于对象层次结构的根部，它提供了访问 AutoCAD 的主窗口的功能。例如，下列代码行更新 AutoCAD 应用程序的主窗口：

VB.NET

```
Autodesk.AutoCAD.ApplicationServices.Application.UpdateScreen()
```

C#

```
Autodesk.AutoCAD.ApplicationServices.Application.UpdateScreen();
```

VBA/ActiveX 代码参考

```
ThisDrawing.Application.Update
```

1.3 集合对象

集合是一种包含了许多相似对象实例的对象类型。下面列出了 AutoCAD .NET API 里定义的部分集合对象：

Block Table Record 块表记录

含指定块定义里的全部实体；

Block Table 块表

含图形文件里的全部块；

Named Objects Dictionary 命名字典

含图形文件里的全部字典；

Dimension Style Table 标注样式表

含图形文件里的全部标注样式；

Document Collection 文档集合

含当前对话任务里的全部打开的图形文件；

File Dependency Collection 文件依赖性集合

含文件依赖性列表里的全部项；

Group Dictionary 组字典

含图形文件里的全部组；

Hyperlink Collection 超链接集合

含给定实体的全部超链接；

Layer Table 图层表

含图形文件里的全部图层；

Layout Dictionary 布局字典

含图形文件里的全部布局；

Linetype Table 线型表

含图形文件里的全部线型；

MenuBar Collection 菜单栏集合

含 AutoCAD 当前显示的全部菜单项；

MenuGroup Collection 菜单组集合

含 AutoCAD 当前加载的全部自定义组。自定义组代表一个加载的 CUIx 文件，该文件可包含菜单、工具条、ribbon 选项卡以及定义用户界面的其他元素。

Plot Configuration Dictionary 绘图配置字典

含图形文件里的命名绘图设置；

Registered Application Table 已注册应用程序表

含图形文件里全部已注册的应用程序；

Text Style Table 文字样式表

含图形文件里的全部文字样式；

UCS Table UCS 表

含图形文件里的全部用户坐标系；

View Table 视图表

含图形文件里的全部视图；

Viewport Table 视口表

含图形文件里的全部视口；

1.3.1 访问集合

多数集合对象和容器对象都是通过 Document 对象或 Database 对象来访问的。Document 对象和 Database 对象包含一个属性，用来访问大多数可用集合对象中的对象或对象 ID。例如，下面的代码定义了一个变量并检索 LayersTable 对象，LayersTable 对象表示当前图形文件的图层集合：

VB.NET

```
''获取当前文档，启动事务管理器
Dim acCurDb As Database = Application.DocumentManager.MdiActiveDocument.Database
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' 本例返回当前数据库中的图层表
    Dim acLyrTbl As LayerTable
    acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                 OpenMode.ForRead)

    '' 关闭事务
End Using
```

C#

```
// 获取当前文档，启动事务管理器
Database acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // 本例返回当前数据库中的图层表
    LayerTable acLyrTbl;
    acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                 OpenMode.ForRead) as LayerTable;

     // 关闭事务
}
```

VBA/ActiveX 代码参考

```
Dim layerCollection as AcadLayers
Set layerCollection = ThisDrawing.Layers
```

1.3.2 向集合对象添加新成员

向集合中添加新成员，使用 Add()方法。下面的示例代码新建一个图层并将其添加到 Layer 表：

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("AddMyLayer")> _
Public Sub AddMyLayer()
  '' 获取当前数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 返回当前数据库的图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' 检查图层表里是否有图层 MyLayer
      If Not acLyrTbl.Has("MyLayer") Then
          '' 以写模式打开图层表
          acLyrTbl.UpgradeOpen()

          '' 新创建一个图层表记录，并命名为”MyLayer”
          Dim acLyrTblRec As LayerTableRecord = New LayerTableRecord
          acLyrTblRec.Name = "MyLayer"

          ''添加新的图层表记录到图层表，添加事务
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)

          '' 提交修改
          acTrans.Commit()
      End If
```

```
    '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("AddMyLayer")]
public static void AddMyLayer()
{
  // 获取当前文档和数据库，并启动事务；
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 返回当前数据库的图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // 检查图层表里是否有图层 MyLayer
      if (acLyrTbl.Has("MyLayer") != true)
      {
          // 以写模式打开图层表
          acLyrTbl.UpgradeOpen();

          // 新创建一个图层表记录，并命名为”MyLayer”
          LayerTableRecord acLyrTblRec = new LayerTableRecord();
          acLyrTblRec.Name = "MyLayer";

          // 添加新的图层表记录到图层表，添加事务
          acLyrTbl.Add(acLyrTblRec);
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);

          //提交修改
          acTrans.Commit();
      }

      // 关闭事务，回收内存；
```

```
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddMyLayer()
    Dim newLayer as AcadLayer
    Set newLayer = ThisDrawing.Layers.Add("MyLayer")
End Sub
```

1.3.3 迭代集合对象

选择集合对象的指定成员，用 Item()方法或 GetAt()方法。Item()方法和 GetAt()方法需要一个代表成员名称的字符串形式参数值。对大多数集合来说，Item()方法是隐含的，这意味着我们其实不需要使用该方法。

对有些集合对象来说，我们还可以用一个索引值来指定一个成员在要检索的集合里的位置。我们所用的这些方法，会根据我们所用的编程语言的不同，以及我们是使用符号表还是使用字典，而有不同的格式。

下列代码演示如何访问图层符号表里的 MyLayer 表记录：

VB.NET

```
acObjId = acLyrTbl.Item("MyLayer")

acObjId = acLyrTbl("MyLayer")
```

C#

```
acObjId = acLyrTbl["MyLayer"];
```

VBA/ActiveX 代码参考

```
acLayer = ThisDrawing.Layers.Item("MyLayer")

acLayer = ThisDrawing.Layers("MyLayer")
```

迭代 LayerTable 对象

下面的例子遍历 LayerTable 对象并显示全部图层表记录的名称。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
```

```
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("IterateLayers")> _
Public Sub IterateLayers()
  '' 获取当前数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      ''返回当前数据库的图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' 遍历图层表并打印每个图层的名字
      For Each acObjId As ObjectId In acLyrTbl
          Dim acLyrTblRec As LayerTableRecord
          acLyrTblRec = acTrans.GetObject(acObjId, OpenMode.ForRead)

          acDoc.Editor.WriteMessage(vbLf & acLyrTblRec.Name)
      Next

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("IterateLayers")]
public static void IterateLayers()
{
  // 获取当前数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // This example 返回当前数据库的图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;
```

```
      // 遍历图层表并打印每个图层的名字

      foreach (ObjectId acObjId in acLyrTbl)
      {
          LayerTableRecord acLyrTblRec;
          acLyrTblRec = acTrans.GetObject(acObjId,
                                          OpenMode.ForRead) as LayerTableRecord;

          acDoc.Editor.WriteMessage("\n" + acLyrTblRec.Name);
      }

      // 关闭事务
  }
}
```

VBA/ActiveX 代码参考

```
Sub IterateLayers()
    ' 遍历集合
    On Error Resume Next

    Dim lay As AcadLayer
    Dim msg As String
    msg = ""
    For Each lay In ThisDrawing.Layers
        msg = msg + lay.Name + vbCrLf
    Next

    ThisDrawing.Utility.prompt msg
End Sub
```

从 LayerTable 对象中查找名为 MyLayer 的图层表记录

下面的例子检查 LayerTable 对象，确定名为 MyLayer 的图层是否存在，并显示相应消息。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("FindMyLayer")> _
```

```
Public Sub FindMyLayer()
  '' 获取当前数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 返回当前数据库的图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' 检查图层表里是否有图层 MyLayer
      If Not acLyrTbl.Has("MyLayer") Then
          acDoc.Editor.WriteMessage(vbLf & "'MyLayer' does not exist")
      Else
          acDoc.Editor.WriteMessage(vbLf & "'MyLayer' exists")
      End If

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("FindMyLayer")]
public static void FindMyLayer()
{
  // 获取当前数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 返回当前数据库的图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // 检查图层表里是否有图层 MyLayer
      if (acLyrTbl.Has("MyLayer") != true)
      {
```

```
            acDoc.Editor.WriteMessage("\n'MyLayer' does not exist");
        }
        else
        {
            acDoc.Editor.WriteMessage("\n'MyLayer' exists");
        }

        // 关闭事务
    }
}
```

VBA/ActiveX 代码参考

```
Sub FindMyLayer()
    ' 使用 Item 方法查找名为 MyLayer 的图层
    On Error Resume Next
    Dim ABCLayer As AcadLayer
    Set ABCLayer = ThisDrawing.Layers("MyLayer")
    If Err <> 0 Then
        ThisDrawing.Utility.prompt "'MyLayer' does not exist"
    Else
        ThisDrawing.Utility.prompt "'MyLayer' exists"
    End If
End Sub
```

1.3.4 从集合对象中删除成员

可以使用集合对象成员的 Erase()方法删除集合对象成员。下面的示例代码演示如何从 LayerTable 对象中删除图层 MyLayer。

从图形中删除一个图层前，应先确定该图层能否被安全删除。我们用 Purge()方法确定是否可以删除图层、块、文字样式等这样的命名对象。

关于 Purge()方法的内容，见（§3.4.1.1 *清理未被引用的命名对象*）。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("RemoveMyLayer")> _
Public Sub RemoveMyLayer()
```

```
  '' 获取当前数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 返回当前数据库的图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      '' 检查图层表里是否有图层 MyLayer
      If acLyrTbl.Has("MyLayer") = True Then
          Dim acLyrTblRec As LayerTableRecord
          acLyrTblRec = acTrans.GetObject(acLyrTbl("MyLayer"), _
                                          OpenMode.ForWrite)

          Try
              acLyrTblRec.Erase()
              acDoc.Editor.WriteMessage(vbLf & "'MyLayer' was erased")

              '' 提交修改
              acTrans.Commit()
          Catch
              acDoc.Editor.WriteMessage(vbLf & "'MyLayer' could not be erased")
          End Try
      Else
          acDoc.Editor.WriteMessage(vbLf & "'MyLayer' does not exist")
      End If

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RemoveMyLayer")]
public static void RemoveMyLayer()
{
  // 获取当前数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
  Database acCurDb = acDoc.Database;
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 返回当前数据库的图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;
      // 检查图层表里是否有图层 MyLayer
      if (acLyrTbl.Has("MyLayer") == true)
      {
          LayerTableRecord acLyrTblRec;
          acLyrTblRec = acTrans.GetObject(acLyrTbl["MyLayer"],
                                          OpenMode.ForWrite) as
LayerTableRecord;

          try
          {
              acLyrTblRec.Erase();
              acDoc.Editor.WriteMessage("\n'MyLayer' was erased");

              // 提交修改
              acTrans.Commit();
          }
          catch
          {
              acDoc.Editor.WriteMessage("\n'MyLayer' could not be erased");
          }
      }
      else
      {
          acDoc.Editor.WriteMessage("\n'MyLayer' does not exist");
      }

      // 关闭事务
  }
}
```

VBA/ActiveX 代码参考

```
Sub RemoveMyLayer()
  On Error Resume Next

  '' 从 Layers 集合里获取图层"MyLayer"
  Dim ABCLayer As AcadLayer
```

```
  Set ABCLayer = ThisDrawing.Layers.Item("MyLayer")

  '' 错误检查，没发生错误说明图层存在
  If Err = 0 Then
    '' 删除图层
    ABCLayer.Delete
    '' 清空错误对象
    Err.Clear

    '' 再获取图层，如果能找到说明没被删除
    Set ABCLayer = ThisDrawing.Layers.Item("MyLayer")

    ''错误检查，发生错误说明图层已被删除
    If Err <> 0 Then
      ThisDrawing.Utility.prompt "'MyLayer' was removed"
    Else
      ThisDrawing.Utility.prompt "'MyLayer' could not be removed"
    End If
  Else
    ThisDrawing.Utility.prompt "'MyLayer' does not exist"
  End If
End Sub
```

一旦删除了对象，我们就不能试图在随后的程序中再访问该对象，否则就会出错。上面的示例代码演示了在访问对象前检查对象是否存在。试图删除对象时，应该用 Has()方法检查其是否存在，或用 Try 语句捕捉可能发生的任何异常。有关处理异常的更多内容见（§8.1 ***处理错误***）。

1.4 了解属性和方法

每个对象都有与之关联的属性和方法。属性用来描述对象的状态，方法是在对象上能执行的动作。对象一旦创建，我们就可以通过其属性和方法对其进行查询和编辑。

例如，Circle 对象有 Center 属性，该属性代表圆心的坐标。要改变圆心，只需简单将 Center 属性设置为新点即可。Circle 对象还有一个 GetOffsetCurves()方法，该方法创建一个从已存在的圆偏离指定距离的新对象。

查看 Circle 对象的全部属性和方法列表，参考 *AutoCAD .NET 参考手册*中的 Circle 对象，或在 Microsoft Visual Studio 中使用对象浏览器。参见（附录§A.7 ***访问和查找引用库（对象浏览器）***）

1.5 进程外与进程内

当我们开发新的应用程序时，即可以运行在进程内，又可以运行在进程外。AutoCAD .NET API 只设计用来运行于进程内，这与能运行于进程外的 ActiveX Automation 库是不同的。

- 进程内应用程序运行在与宿主应用程序相同的进程空间。这种情况下，由宿主程序 AutoCAD 来加载 DLL 程序集。
- 进程外应用程序不和宿主应用程序运行在相同的进程空间，这样的应用程序通常生成为独立的可执行程序。

如果需要创建一个独立应用程序来驾驭 AutoCAD，最好是在应用程序中用 CreateObject()方法和 GetObject()方法新创建一个 AutoCAD 程序的实例，或者返回正在运行的 AutoCAD 程序的一个实例。一旦返回对 AcadApplication 的引用，我们就可以使用 SendCommand()方法将我们的进程内.NET 应用程序加载到 AutoCAD。 SendCommand()方法是 AcadApplication 的 ActiveDocument 属性的一个成员。

除了在进程内运行你的.NET 应用程序外，作为一个替代，还可以在你的应用程序使用 COM 互操作。

注： 访问 AutoCAD 2012 的 COM 应用程序要用到的 ProgID 值是 ***AutoCAD.Application.18***。
访问 AutoCAD 2014 的 COM 应用程序要用到的 ProgID 值是 ***AutoCAD.Application.19***。

VB.NET

```
Imports System
Imports System.Runtime.InteropServices

Imports Autodesk.AutoCAD.Interop
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("ConnectToAcad")> _
Public Sub ConnectToAcad()
  Dim acAppComObj As AcadApplication
  Dim strProgId As String = "AutoCAD.Application.18"

  On Error Resume Next

  '' 获取正在运行的 AutoCAD 实例
  acAppComObj = GetObject(, strProgId)

  '' 如果没有实例运行就出错
  If Err.Number > 0 Then
      Err.Clear()
```

```
      '' 创建新的 AutoCAD 实例
      acAppComObj = CreateObject("AutoCAD.Application.18")

      '' 检查 AutoCAD 实例是否创建了
      If Err.Number > 0 Then
         Err.Clear()

         '' 创建新实例不成功就显示信息并退出
         MsgBox("Instance of 'AutoCAD.Application' could not be created.")

         Exit Sub
      End If
  End If

  '' 显示获得的应用程序实例并返回名称、版本
  acAppComObj.Visible = True
  MsgBox("Now running " & acAppComObj.Name & " version " & acAppComObj.Version)

  '' 获取当前文档
  Dim acDocComObj As AcadDocument
  acDocComObj = acAppComObj.ActiveDocument

  '' 可选的，加载程序集并启动命令，如果进程内程序集
  '' 已被加载，直接启动命令即可
 acDocComObj.SendCommand("(command " & Chr(34) & "NETLOAD" & Chr(34) & " " & _
                         Chr(34) & "c:/myapps/mycommands.dll" & Chr(34) & ") ")

  acDocComObj.SendCommand("MyCommand ")
End Sub
```

C#

```
using System;
using System.Runtime.InteropServices;
using Autodesk.AutoCAD.Interop;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("ConnectToAcad")]
public static void ConnectToAcad()
{

  AcadApplication acAppComObj = null;
  const string strProgId = "AutoCAD.Application.18";//19 for AutoCAD2014
```

```
  // 获取正在运行的 AutoCAD 实例
  try
  {
      acAppComObj = (AcadApplication)Marshal.GetActiveObject(strProgId);
  }
  catch //如果没有实例运行就出错
  {
      try
      {
          // 创建新的 AutoCAD 实例
          acAppComObj =
(AcadApplication)Activator.CreateInstance(Type.GetTypeFromProgID(strProgId),
true);
      }
      catch
      {
          // 创建新实例不成功就显示信息并退出
          System.Windows.Forms.MessageBox.Show("Instance of
'AutoCAD.Application'" +
                                               " could not be created.");

          return;
      }
  }

  // 显示获得的应用程序实例并返回名称、版本
  acAppComObj.Visible = true;
  System.Windows.Forms.MessageBox.Show("Now running " + acAppComObj.Name +
                                       " version " + acAppComObj.Version);

  // 获取当前文档
  AcadDocument acDocComObj;
  acDocComObj = acAppComObj.ActiveDocument;

  // 可选的，加载程序集并启动命令，如果进程内程序集
  // 已被加载，直接启动命令即可
  acDocComObj.SendCommand("(command " + (char)34 + "NETLOAD" + (char)34 + " " +
                          (char)34 + "c:/myapps/mycommands.dll" + (char)34 + ")
");

  acDocComObj.SendCommand("MyCommand ");
}
```

注： 需要引用 Autodesk.AutoCAD.Interop.dll 和 Autodesk.AutoCAD.Interop.Common.dll 两个程序集，在 C:\ObjectARX_2014\inc-win32\目录内。（译者）

VBA/ActiveX 代码参考

```
Sub ConnectToAcad()
    Dim acadApp As AcadApplication
    On Error Resume Next
    Set acadApp = GetObject(, "AutoCAD.Application.18")
    If Err Then
        Err.Clear
        Set acadApp = CreateObject("AutoCAD.Application.18")
        If Err Then
            MsgBox Err.Description
            Exit Sub
        End If
    End If
    acadApp.Visible = True
    MsgBox "Now running " + acadApp.Name + _
           " version " + acadApp.Version
    Dim acadDoc as AcadDocument
    Set acadDoc = acadApp.ActiveDocument
    '' 可选的，加载程序集并启动命令，如果进程内程序集
    '' 已被加载，直接启动命令即可
    acadDoc.SendCommand("(command " & Chr(34) & "NETLOAD" & Chr(34) & " " & _
                        Chr(34) & "c:/myapps/mycommands.dll" & Chr(34) & ") ")
    acadDoc.SendCommand("MyCommand ")
End Sub
```

1.6 定义命令和 AutoLISP 函数

可以通过 AutoCAD .NET API 编程接口提供的的两个属性来定义命令和 AutoLISP 函数，这两个属性是 CommandMethod 和 LispFunction。 只要将这两个属性中的一个放在 AutoCAD 命令或 AutoLISP 函数要调用的方法前就可以了。

用来定义命令的方法不能定义带参数的命令，相反，用来定义 AutoLISP 函数的方法所定义的函数必须带一个 ResultBuffer 类型的参数。

1.6.1 定义命令

定义命令使用 CommandMethod 属性。CommandMethod 属性需要一个字符串值作为要定义的命令的全局名字。除了全局命令名外，CommandMethod 属性还可以接受下列参数值：

- **Command Flags 命令标志** – 定义命令的行为；
- **Group Name 组名** – 命令编组名称；
- **Local Name 本地名** – 指定语言的本地命令名称；
- **Help Topic Name 帮助主题名** – 按下 F1 键时将要显示的帮助主题名；
- **Context Menu Type Flags 上下文菜单类型标志** – 定义命令处于活动状态时的上下文菜单行为；
- **Help File Name 帮助文件名** – 帮助文件，含有命令活动状态下按下 F1 时要显示的帮助主题；

下表列出了定义命令行为用到的标志：

表 1-1 **Command Flags 命令标志**

枚举值	描述
ActionMacro	可以用动作录制器录制命令动作；
DocReadLock	命令执行时将被只读锁定；
Interruptible	提示用户输入时可以中断命令；
Modal	别的命令运行时不能运行此命令；
NoActionRecording	不能用动作录制器录制命令动作；
NoBlockEditor	不能从块编辑器使用该命令；
NoHistory	不能将命令添加到 repeat-last-command（重复上一个命令）历史列表；
NoPaperSpace	不能从图纸空间使用该命令；
NoTileMode	当 TILEMODE 置 1 时不能使用该命令；
NoUndoMarker	命令不支持撤销标记。用于不修改数据库因而也就无需出现在撤销记录中的那些命令；
Redraw	不清空取回的先选择后执行设置及对象捕捉设置；
Session	命令运行于应用程序上下文，而不是当前图形文档上下文；
Transparent	别的命令运行时可以运行此命令；
Undefined	只能通过全局名使用命令；
UsePickSet	清空取回的先选择后执行设置；

定义命令的语法

下面演示使用 CommandMethod 属性创建一个名为 CheckForPickfirstSelection 命令，该属性还用 UsePickSet 命令标志，表示允许该命令使用命令启动前已经选择的对象。

VB.NET

```
<CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)> _
Public Sub CheckForPickfirstSelection()
 . . .
End Sub
```

C#

```
[CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)]
public static void CheckForPickfirstSelection()
{
 . . .
}
```

用 VB.NET 里的+操作符和 C#里的&操作符，可以指定使用多个命令标志。

VB.NET

```
<CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet + _
                                             CommandFlags.NoBlockEditor)> _
Public Sub CheckForPickfirstSelection()
 . . .
End Sub
```

C#

```
[CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet &
                                             CommandFlags.NoBlockEditor)]
public static void CheckForPickfirstSelection()
{
 . . .
}
```

1.6.2 定义 AutoLISP 函数

定义 AutoLISP 函数使用 LispFunction 属性。LispFunction 属性需要一个字符串值作为要定义的 AutoLISP 函数的全局名字。除了全局函数名，LispFunction 结构还可接受下列值：

- **Local Name 本地名** - 指定语言的本地函数名；
- **Help Topic Name 帮助主题名** - 与 AutoLISP 函数关联的帮助主题名称；
- **Help File Name 帮助文件名** - 帮助文件，含有命令活动状态下按下 F1 时要显示的帮助主题；

定义 AutoLISP 函数的语法

下面示例演示用 LispFunction 属性创建一个名为 InsertDynamicBlock 的 AutoLISP 函数。

VB.NET

```
<LispFunction("InsertDynamicBlock")> _
Public Sub InsertDynamicBlock(ByVal rbArgs As ResultBuffer)
 . . .
End Sub
```

C#

```
[LispFunction("InsertDynamicBlock ")]
public static void InsertDynamicBlock (ResultBuffer rbArgs)
{
 . . .
}
```

检索传入 AutoLISP 函数的值

使用 Foreach 循环遍历 AutoLISP 函数返回的 ResultBuffer 中的值。ResultBuffer 是 TypedValue 对象的集合。TypedValue 对象的 TypeCode 属性用来确定传入 AutoLISP 函数的每个值的类型，Value 属性则用来返回 TypedValue 对象的值。

定义一个 AutoLISP 函数

本示例代码定义一个名为 DisplayFullName 的 AutoLISP 函数。在.NET 项目里定义的方法接受一个参数值，而所定义的 AutoLISP 函数需要两个字符串值来产生正确的输出结果。

将.NET 项目加载到 AutoCAD，在命令提示行输入下列 lisp 指令：

```
(displayfullname "First" "Last")
```

下面是 AutoLISP 函数执行后显示的输出结果：

```
Name: First Last
```

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices;
```

```
<LispFunction("DisplayFullName")> _
Public Sub DisplayFullName(ByVal rbArgs As ResultBuffer)
  If Not rbArgs = Nothing Then
      Dim strVal1 As String = "", strVal2 As String = ""

      Dim nCnt As Integer = 0
      For Each rb As TypedValue In rbArgs
          If (rb.TypeCode = Autodesk.AutoCAD.Runtime.LispDataType.Text) Then
              Select Case nCnt
                  Case 0
                      strVal1 = rb.Value.ToString()
                  Case 1
                      strVal2 = rb.Value.ToString()
              End Select

              nCnt = nCnt + 1
          End If
      Next

      Application.DocumentManager.MdiActiveDocument.Editor. _
         WriteMessage(vbLf & "Name: " & strVal1 & " " & strVal2)
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;


[LispFunction("DisplayFullName")]
public static void DisplayFullName(ResultBuffer rbArgs)
{
  if (rbArgs != null)
  {
      string strVal1 = "";
      string strVal2 = "";

      int nCnt = 0;
      foreach (TypedValue rb in rbArgs)
      {
          if (rb.TypeCode == (int)Autodesk.AutoCAD.Runtime.LispDataType.Text)
          {
              switch(nCnt)
```

```
                {
                    case 0:
                        strVal1 = rb.Value.ToString();
                        break;
                    case 1:
                        strVal2 = rb.Value.ToString();
                        break;
                }

                nCnt = nCnt + 1;
            }
        }

        Application.DocumentManager.MdiActiveDocument.Editor.
            WriteMessage("\nName: " + strVal1 + " " + strVal2);
    }
}
```

第 2 章 控制 AutoCAD 环境

本章是开发 AutoCAD 进程内应用程序的基础，将学习到许多控制和有效使用 AutoCAD 环境的概念和方法。

本章主要内容：

- 控制应用程序窗口
- 控制图形窗口
- 创建、打开、保存和关闭图形
- 锁定和解锁文档
- 设置 AutoCAD 选项
- 设置和返回系统变量
- 精确制图
- 提示用户输入
- 访问 AutoCAD 命令行

2.1 控制应用程序窗口

控制 AutoCAD 应用程序窗口的能力让开发人员可以灵活地创建高效智能的应用程序。比如有时我们需要在程序中适时地最小化 AutoCAD 窗口，也许此时我们的代码正在使用其他应用程序如 Excel 处理任务。又比如，我们在执行像提示用户输入这样的任务前，经常需要确认 AutoCAD 窗口的状态。

使用 Application 对象的方法和属性，我们可以改变 AutoCAD 应用程序窗口的位置、大小及可见性，还可以用 WindowState 属性来最小化、最大化 Application 窗口，以及检查 Application 窗口的当前状态等。

设置应用程序窗口位置和大小

本例使用 Location 属性和 Size 属性将 AutoCAD 应用程序窗口定位于屏幕左上角，并将窗口大小设置为 400×400 像素。

（在 AutoCAD 2013 版和 AutoCAD 2014 版里，没有提供 Location 属性和 Size 属性。 - 译者注）

注：下列示例需要在项目中引用 PresentationCore 库（*PresentationCore.dll*）。从添加引用对话框的.NET 选项页中选 PresentationCore 即可。

VB.NET

```
Imports System.Drawing
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("PositionApplicationWindow")> _
Public Sub PositionApplicationWindow()
  '' 设置应用程序窗口位置
  Dim ptApp As Point = New Point(0, 0)
  Application.MainWindow.Location = ptApp

  '' 设置应用程序窗口大小
  Dim szApp As Size = New Size(400, 400)
  Application.MainWindow.Size = szApp
End Sub
```

C#

```
using System.Drawing;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("PositionApplicationWindow")]
public static void PositionApplicationWindow()
{
  // 设置应用程序窗口位置
  Point ptApp = new Point(0, 0);
  Application.MainWindow.Location = ptApp;

  // 设置应用程序窗口大小
  Size szApp = new Size(400, 400);
  Application.MainWindow.Size = szApp;
}
```

VBA/ActiveX 代码参考

```
Sub PositionApplicationWindow()
    '' 设置应用程序窗口位置
    ThisDrawing.Application.WindowTop = 0
    ThisDrawing.Application.WindowLeft = 0
```

```
    '' 设置应用程序窗口大小
    ThisDrawing.Application.width = 400
    ThisDrawing.Application.height = 400
End Sub
```

最小化和最大化应用程序窗口

注： 下列示例需要在项目中引用 PresentationCore 库（*PresentationCore.dll*）。从添加引用对话框的.NET 选项页中选 PresentationCore 即可。

VB.NET

```
Imports System.Drawing
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("MinMaxApplicationWindow")> _
Sub MinMaxApplicationWindow()
  '' 最小化应用程序窗口
  Application.MainWindow.WindowState = Window.State.Minimized
  MsgBox("Minimized", MsgBoxStyle.SystemModal, "MinMax")

  '' 最大化应用程序窗口
  Application.MainWindow.WindowState = Window.State.Maximized
  MsgBox("Maximized", MsgBoxStyle.SystemModal, "MinMax")
End Sub
```

C#

```
using System.Drawing;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("MinMaxApplicationWindow")]
public static void MinMaxApplicationWindow()
{
  // 最小化应用程序窗口
  Application.MainWindow.WindowState = Window.State.Minimized;
  System.Windows.Forms.MessageBox.Show("Minimized", "MinMax",
              System.Windows.Forms.MessageBoxButtons.OK,
              System.Windows.Forms.MessageBoxIcon.None,
              System.Windows.Forms.MessageBoxDefaultButton.Button1,
              System.Windows.Forms.MessageBoxOptions.ServiceNotification);
```

```
  // 最大化应用程序窗口
  Application.MainWindow.WindowState = Window.State.Maximized;
  System.Windows.Forms.MessageBox.Show("Maximized", "MinMax");
}
```

VBA/ActiveX 代码参考

```
Sub MinMaxApplicationWindow()
    '' 最小化应用程序窗口
    ThisDrawing.Application.WindowState = acMin
    MsgBox "Minimized"

    ''最大化应用程序窗口
    ThisDrawing.Application.WindowState = acMax
    MsgBox "Maximized"
End Sub
```

获取应用程序窗口当前状态

本示例查询应用程序窗口的状态并将其显示出来。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("CurrentWindowState")> _
Public Sub CurrentWindowState()
  System.Windows.Forms.MessageBox.Show("The application window is " + _

Application.MainWindow.WindowState.ToString(), _
                                      "Window State")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("CurrentWindowState")]
public static void CurrentWindowState()
{
  System.Windows.Forms.MessageBox.Show("The application window is " +

Application.MainWindow.WindowState.ToString(),
```

```
                                        "Window State");
}
```

VBA/ActiveX 代码参考

```
Sub CurrentWindowState()
    Dim CurrWindowState As Integer
    Dim msg As String
    CurrWindowState = ThisDrawing.Application.WindowState
    msg = Choose(CurrWindowState, "Normal", _
                 "Minimized", "Maximized")
    MsgBox "The application window is " + msg
End Sub
```

使应用程序窗口不可见和可见

下列代码用 Visible 属性让 AutoCAD 应用程序窗口不可见，然后再让它可见。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("HideWindowState")> _
Public Sub HideWindowState()
  '' 隐藏应用程序窗口
  Application.MainWindow.Visible = False
  MsgBox("Invisible", MsgBoxStyle.SystemModal, "Show/Hide")

  '' 显示应用程序窗口
  Application.MainWindow.Visible = True
  MsgBox("Visible", MsgBoxStyle.SystemModal, "Show/Hide")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("HideWindowState")]
public static void HideWindowState()
{
  // 隐藏应用程序窗口
  Application.MainWindow.Visible = false;
  System.Windows.Forms.MessageBox.Show("Invisible", "Show/Hide");
```

```
  // 显示应用程序窗口
  Application.MainWindow.Visible = true;
  System.Windows.Forms.MessageBox.Show("Visible", "Show/Hide");
}
```

VBA/ActiveX 代码参考

```
Sub HideWindowState()
    ''隐藏应用程序窗口
    ThisDrawing.Application.Visible = False
    MsgBox "Invisible"

    ''显示应用程序窗口
    ThisDrawing.Application.Visible = True
    MsgBox "Visible"
End Sub
```

2.2 控制图形窗口

AutoCAD 应用程序是多文档界面软件，英文为 Multiple Document Interface，简称 MDI。在 AutoCAD 主窗口内可以打开多个图形文档，而每个图形文档都拥有一个窗口。

和 AutoCAD 应用程序窗口一样，我们同样可以对任一图形文档窗口进行最小化、最大化、改变位置、改变大小以及检查状态等操作。我们还可以通过使用视图、视口及缩放方法改变图形在窗口内的显示方式。

AutoCAD .NET API 编程接口提供了许多显示图形的手段。当浏览所绘图形的整体效果时，我们可以通过各种手段控制图形显示：快速移动到图形的不同区域；缩放或平移视图；保存命名视图，并在需要打印或引用特定细节时恢复命名视图；或者通过将屏幕分割为多个平铺视口的方式来一次显示多个视图，等等。

2.2.1 改变文档窗口的位置和大小

可以使用 Document 对象来调整图形文档窗口的位置和大小。可以用 WindowState 属性最大化和最小化文档窗口，还可以用 WindowState 属性获取文档窗口当前状态。

设置当前文档窗口的位置和大小

本例使用 Location 属性和 Size 属性设置文档窗口位置并将窗口大小设置为 400 像素宽 400 像素高。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("SizeDocumentWindow")> _
Public Sub SizeDocumentWindow()
  '' 获取当前文档窗口
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  acDoc.Window.WindowState = Window.State.Normal

  '' 设置文档窗口位置
  Dim ptDoc As Point = New Point(0, 0)
  acDoc.Window.Location = ptDoc

  '' 设置文档窗口大小
  Dim szDoc As Size = New Size(400, 400)
  acDoc.Window.Size = szDoc
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("SizeDocumentWindow")]
public static void SizeDocumentWindow()
{
  // 获取当前文档窗口
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  acDoc.Window.WindowState = Window.State.Normal;

  // 设置文档窗口位置
  Point ptDoc  = new Point(0, 0);
  acDoc.Window.Location = ptDoc;

  // 设置文档窗口大小
  Size szDoc = new Size(400, 400);
  acDoc.Window.Size = szDoc;
}
```

⊟ VBA/ActiveX 代码参考

```
Sub SizeDocumentWindow()
    ThisDrawing.Width = 400
    ThisDrawing.Height = 400
```

```
End Sub
```

最小化和最大化活动文档窗口

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("MinMaxDocumentWindow")> _
Public Sub MinMaxDocumentWindow()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' 最小化文档窗口
  acDoc.Window.WindowState = Window.State.Minimized
  MsgBox("Minimized", MsgBoxStyle.SystemModal, "MinMax")

  '' 最大化文档窗口
  acDoc.Window.WindowState = Window.State.Maximized
  MsgBox("Maximized", MsgBoxStyle.SystemModal, "MinMax")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("MinMaxDocumentWindow")]
public static void MinMaxDocumentWindow()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // 最小化文档窗口
  acDoc.Window.WindowState = Window.State.Minimized;
  System.Windows.Forms.MessageBox.Show("Minimized" , "MinMax");

  // 最大化文档窗口
  acDoc.Window.WindowState = Window.State.Maximized;
  System.Windows.Forms.MessageBox.Show("Maximized" , "MinMax");
}
```

VBA/ActiveX 代码参考

```
Sub MinMaxDocumentWindow()
    '' 最小化文档窗口
```

```
    ThisDrawing.WindowState = acMin
    MsgBox "Minimized"

    '' 最大化文档窗口
    ThisDrawing.WindowState = acMax
    MsgBox "Maximized"
End Sub
```

获取活动文档窗口的当前状态

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

<CommandMethod("CurrentDocWindowState")> _
Public Sub CurrentDocWindowState()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  System.Windows.Forms.MessageBox.Show("The document window is " & _
  acDoc.Window.WindowState.ToString(), "Window State")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Windows;

[CommandMethod("CurrentDocWindowState")]
public static void CurrentDocWindowState()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  System.Windows.Forms.MessageBox.Show("The document window is " +
  acDoc.Window.WindowState.ToString(), "Window State");
}
```

VBA/ActiveX 代码参考

```
Sub CurrentDocWindowState()
    Dim CurrWindowState As Integer
    Dim msg As String
    CurrWindowState = ThisDrawing.WindowState
    msg = Choose(CurrWindowState, "Normal", _
                 "Minimized", "Maximized")
```

```
    MsgBox "The document window is " + msg
End Sub
```

2.2.2 缩放和平移当前视图

视图是图形窗口中具有指定比例、位置、方向的图形。通过改变当前视图的高、宽及中心点位置，可以改变图形的视图。增大或减小视图的宽度或高度会影响图形显示的大小。通过调整当前视图的中心位置可以平移视图。

2.2.2.1 操作当前视图

我们通过调用 Editor 对象的 GetCurrentView()方法来访问模型空间或图纸空间中视口的当前视图。GetCurrentView 方法返回一个 ViewTableRecord 对象。我们就用 ViewTableRecord 对象来操作活动视口中视图的缩放、位置及方向。一旦修改了 ViewTableRecord 对象，我们就可以调用 SetCurrentView()方法来更新活动视口中的当前视图。

用来操作当前视图的常用属性：

- **CenterPoint** - DCS（显示坐标系）坐标系中视图的中心点；
- **Height** - DCS 坐标系中视图的高度，高度增加视图拉远，高度减小视图拉近；
- **Target** - WCS 坐标系中视图的目标；
- **ViewDirection** - WCS 坐标系中视图目标到视图观察点的矢量；
- **ViewTwist** - 视图扭转角（弧度）；
- **Width** - DCS 坐标系中视图的宽度，宽度增加视图拉远，宽度减小视图拉近；

⊟ VBA 交叉参考

AutoCAD .NET API 并没有提供像在 ActiveX Automation 库里那样的直接操作图形当前视图的方法。例如，如果想要对准图形中对象的范围或对准图形界限，我们必须操作当前视图的 Width、Height 及 CenterPoint 属性。要获取图形界限范围，我们用 Database 对象的 Extmin、Extmax、 Limmin 及 Limmax 属性。

用来操作当前视图的函数

本例代码是一个常用子程序，后面的示例中将用到。Zoom()函数接受 4 个参数，实现了缩放视图到边界、平移视图、视图居中以及按给定系数放大视图等功能。Zoom()函数要求所有坐标值为 WCS 坐标。

Zoom()函数的参数：

- **Minimum point** - 用来定义显示区域左下角的 3D 点；
- **Maximum point** - 用来定义显示区域右上角的 3D 点；

- Center point - 用来定义视图中心的 3D 点；
- Scale factor - 用来指定视图放大缩小比例的实数；

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

Public Sub Zoom(ByVal pMin As Point3d, ByVal pMax As Point3d, _
                ByVal pCenter As Point3d, ByVal dFactor As Double)
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Dim nCurVport As Integer =
System.Convert.ToInt32(Application.GetSystemVariable("CVPORT"))

  '' 没提供点或只提供了一个中心点时，获取当前空间的范围
  '' 检查当前空间是否为模型空间
  If acCurDb.TileMode = True Then
      If pMin.Equals(New Point3d()) = True And _
          pMax.Equals(New Point3d()) = True Then

          pMin = acCurDb.Extmin
          pMax = acCurDb.Extmax
      End If
  Else
      ''检查当前空间是否为图纸空间
      If nCurVport = 1 Then
          If pMin.Equals(New Point3d()) = True And _
             pMax.Equals(New Point3d()) = True Then

              pMin = acCurDb.Pextmin
              pMax = acCurDb.Pextmax
          End If
      Else
          '' 获取模型空间的范围
          If pMin.Equals(New Point3d()) = True And _
             pMax.Equals(New Point3d()) = True Then

              pMin = acCurDb.Extmin
              pMax = acCurDb.Extmax
          End If
```

```
    End If
End If

'' 启动事务
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
    '' Get the current view
    Using acView As ViewTableRecord = acDoc.Editor.GetCurrentView()
        Dim eExtents As Extents3d

        '' 将 WCS 坐标转换为 DCS 坐标
        Dim matWCS2DCS As Matrix3d
        matWCS2DCS = Matrix3d.PlaneToWorld(acView.ViewDirection)
        matWCS2DCS = Matrix3d.Displacement(acView.Target - Point3d.Origin) *
matWCS2DCS
        matWCS2DCS = Matrix3d.Rotation(-acView.ViewTwist, _
                                       acView.ViewDirection, _
                                       acView.Target) * matWCS2DCS

        '' 如果指定了中心点，为居中模式和放大模式定义
        '' 显示范围的 min 点和 max 点
        If pCenter.DistanceTo(Point3d.Origin) <> 0 Then
            pMin = New Point3d(pCenter.X - (acView.Width / 2), _
                               pCenter.Y - (acView.Height / 2), 0)

            pMax = New Point3d((acView.Width / 2) + pCenter.X, _
                               (acView.Height / 2) + pCenter.Y, 0)
        End If

        '' 使用一条线创建范围（Extents3d）对象
        Using acLine As Line = New Line(pMin, pMax)
            eExtents = New Extents3d(acLine.Bounds.Value.MinPoint, _
                                     acLine.Bounds.Value.MaxPoint)
        End Using

        '' 计算当前视图的高宽比
        Dim dViewRatio As Double
        dViewRatio = (acView.Width / acView.Height)

        '' 变换视图范围
        matWCS2DCS = matWCS2DCS.Inverse()
        eExtents.TransformBy(matWCS2DCS)

        Dim dWidth As Double
        Dim dHeight As Double
```

```
          Dim pNewCentPt As Point2d

          '' 检查是否提供了中心点 (Center 模式和 Scale 模式)
          If pCenter.DistanceTo(Point3d.Origin) <> 0 Then
              dWidth = acView.Width
              dHeight = acView.Height

              If dFactor = 0 Then
                  pCenter = pCenter.TransformBy(matWCS2DCS)
              End If

              pNewCentPt = New Point2d(pCenter.X, pCenter.Y)
          Else ''窗口、范围和界限模式下
              ''计算当前视图的新的高宽比
              dWidth = eExtents.MaxPoint.X - eExtents.MinPoint.X
              dHeight = eExtents.MaxPoint.Y - eExtents.MinPoint.Y

              '' 获取视图中心点
              pNewCentPt = New Point2d(((eExtents.MaxPoint.X +
eExtents.MinPoint.X) * 0.5), _
                                       ((eExtents.MaxPoint.Y +
eExtents.MinPoint.Y) * 0.5))
          End If

          '' 检查新宽度值是否适合当前窗口
          If dWidth > (dHeight * dViewRatio) Then dHeight = dWidth / dViewRatio

          '' 调整视图大小
          If dFactor <> 0 Then
              acView.Height = dHeight * dFactor
              acView.Width = dWidth * dFactor
          End If

          '' 设置视图中心点
          acView.CenterPoint = pNewCentPt

          '' 设置当前视图
          acDoc.Editor.SetCurrentView(acView)
      End Using

      '' 提交修改
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

static void Zoom(Point3d pMin, Point3d pMax, Point3d pCenter, double dFactor)
{
  //获取当前文档及数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  int nCurVport =
System.Convert.ToInt32(Application.GetSystemVariable("CVPORT"));

  // 没提供点或只提供了一个中心点时，获取当前空间的范围
  // 检查当前空间是否为模型空间
  if (acCurDb.TileMode == true)
  {
      if (pMin.Equals(new Point3d()) == true &&
          pMax.Equals(new Point3d()) == true)
      {
          pMin = acCurDb.Extmin;
          pMax = acCurDb.Extmax;
      }
  }
  else
  {
      // 检查当前空间是否为图纸空间
      if (nCurVport == 1)
      {
          // 获取图纸空间范围
          if (pMin.Equals(new Point3d()) == true &&
              pMax.Equals(new Point3d()) == true)
          {
              pMin = acCurDb.Pextmin;
              pMax = acCurDb.Pextmax;
          }
      }
      else
      {
          // 获取模型空间范围
          if (pMin.Equals(new Point3d()) == true &&
              pMax.Equals(new Point3d()) == true)
```

```
            {
                pMin = acCurDb.Extmin;
                pMax = acCurDb.Extmax;
            }
        }
    }
    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 获取当前视图
        using (ViewTableRecord acView = acDoc.Editor.GetCurrentView())
        {
            Extents3d eExtents;

            // 将 WCS 坐标变换为 DCS 坐标
            Matrix3d matWCS2DCS;
            matWCS2DCS = Matrix3d.PlaneToWorld(acView.ViewDirection);
            matWCS2DCS = Matrix3d.Displacement(acView.Target - Point3d.Origin) *
matWCS2DCS;
            matWCS2DCS = Matrix3d.Rotation(-acView.ViewTwist,
                                           acView.ViewDirection,
                                           acView.Target) * matWCS2DCS;

            //如果指定了中心点，就为中心模式和比例模式
            //设置显示范围的最小点和最大点；
            if (pCenter.DistanceTo(Point3d.Origin) != 0)
            {
                pMin = new Point3d(pCenter.X - (acView.Width / 2),
                                   pCenter.Y - (acView.Height / 2), 0);

                pMax = new Point3d((acView.Width / 2) + pCenter.X,
                                   (acView.Height / 2) + pCenter.Y, 0);
            }

            // 用直线创建范围对象；
            using (Line acLine = new Line(pMin, pMax))
            {
                eExtents = new Extents3d(acLine.Bounds.Value.MinPoint,
                                         acLine.Bounds.Value.MaxPoint);
            }

            // 计算当前视图的宽高比
            double dViewRatio;
            dViewRatio = (acView.Width / acView.Height);
```

```
            // 变换视图范围
            matWCS2DCS = matWCS2DCS.Inverse();
            eExtents.TransformBy(matWCS2DCS);

            double dWidth;
            double dHeight;
            Point2d pNewCentPt;

            //检查是否提供了中心点（中心模式和比例模式）
            if (pCenter.DistanceTo(Point3d.Origin) != 0)
            {
                dWidth = acView.Width;
                dHeight = acView.Height;

                if (dFactor == 0)
                {
                    pCenter = pCenter.TransformBy(matWCS2DCS);
                }

                pNewCentPt = new Point2d(pCenter.X, pCenter.Y);
            }
            else // 窗口、范围和界限模式下
            {
                // 计算当前视图的宽高新值；
                dWidth = eExtents.MaxPoint.X - eExtents.MinPoint.X;
                dHeight = eExtents.MaxPoint.Y - eExtents.MinPoint.Y;

                // 获取视图中心点
                pNewCentPt = new Point2d(((eExtents.MaxPoint.X +
eExtents.MinPoint.X) * 0.5),
                                         ((eExtents.MaxPoint.Y +
eExtents.MinPoint.Y) * 0.5));
            }

            // 检查宽度新值是否适于当前窗口
            if (dWidth > (dHeight * dViewRatio)) dHeight = dWidth / dViewRatio;

            // 调整视图大小；
            if (dFactor != 0)
            {
                acView.Height = dHeight * dFactor;
                acView.Width = dWidth * dFactor;
            }
```

```
            // 设置视图中心；
            acView.CenterPoint = pNewCentPt;

            // 更新当前视图；
            acDoc.Editor.SetCurrentView(acView);
        }

        // 提交更改；
        acTrans.Commit();
    }
}
```

2.2.2.2 定义缩放窗口

在 AutoCAD 中，我们用 ZOOM 命令的 Window 选项来定义要显示在图形窗口的图形区域。当我们通过指定两个点定义这样的显示区域时，当前视图的 Width 属性和 Height 属性会调整至与该区域相匹配，同时视图的 CenterPoint 属性也会移动到基于指定点的新位置上。

缩放到两点定义的区域

本例代码使用上一小节（§2.2.2.1 *操作当前视图*）给出的 Zoom()函数定义一个区域，并演示如何缩放到该区域。显示区域通过将坐标(1.3,7.8,0)和(13.7,-2.6,0)传递给 Zoom()方法的前两个参数来定义。

不需要新的中心点，所以将传递给 Zoom()函数第三个参数声明为一个新的 Point3d 对象。Zoom()函数的最后一个参数用来按比例缩放新视图。按比例缩放视图可以用来在显示区域和图形窗口间形成间隔。

VB.NET

```
<CommandMethod("ZoomWindow")> _
Public Sub ZoomWindow()
  '' 缩放到由点(1.3, 7.8)和点(13.7, -2.6)定义的窗口
  Dim pMin As Point3d = New Point3d(1.3, 7.8, 0)
  Dim pMax As Point3d = New Point3d(13.7, -2.6, 0)

  Zoom(pMin, pMax, New Point3d(), 1)
End Sub
```

C#

```
[CommandMethod("ZoomWindow")]
static public void ZoomWindow()
{
```

```
  // Zoom to a window boundary defined by 1.3,7.8 and 13.7,-2.6
  // 缩放到由点(1.3, 7.8)和点(13.7, -2.6)定义的窗口
  Point3d pMin = new Point3d(1.3, 7.8, 0);
  Point3d pMax = new Point3d(13.7, -2.6, 0);

  Zoom(pMin, pMax, new Point3d(), 1);
}
```

VBA/ActiveX 代码参考

```
Sub ZoomWindow()
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    point1(0) = 1.3: point1(1) = 7.8: point1(2) = 0
    point2(0) = 13.7: point2(1) = -2.6: point2(2) = 0

    ThisDrawing.Application.ZoomWindow point1, point2
End Sub
```

2.2.2.3 按比例缩放视图

如果需要增加或减少绘图窗口中图像的放大倍率，可以更改当前视图的 Width 和 Height 属性。改变视图大小时，应确保用相同的比例系数修改 Width 属性和 Height 属性。改变当前视图大小时计算的比例系数通常基于下列情形：

- 相对于图形界限；
- 相对于当前视图
- 相对于图纸空间的图形单位；

用指定比例放大活动图形

本示例代码演示如何使用（§2.2.2.1 *操作当前视图*）一节给出的 Zoom()函数将当前视图缩小 50%。

Zoom()函数总共需要 4 个参数，头两个为新声明的 3D 点，本例没用，第三个值为用来调整视图大小的中心点，最后一个为用来调整视图大小的比例系数。

VB.NET

```
<CommandMethod("ZoomScale")> _
Public Sub ZoomScale()
  '' 获取当前文档
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```
  '' 获取当前视图
  Using acView As ViewTableRecord = acDoc.Editor.GetCurrentView()
      '' 获取当前视图中心
      Dim pCenter As Point3d = New Point3d(acView.CenterPoint.X, _
                                           acView.CenterPoint.Y, 0)

      '' 设置比例系数
      Dim dScale As Double = 0.5

      '' 基于当前视图中心按比例缩放视图
      Zoom(New Point3d(), New Point3d(), pCenter, 1 / dScale)
  End Using
End Sub
```

C#

```
[CommandMethod("ZoomScale")]
static public void ZoomScale()
{
  //  获取当前文档;
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  //  获取当前视图;
  using (ViewTableRecord acView = acDoc.Editor.GetCurrentView())
  {
      //  获取当前视图中心;
      Point3d pCenter = new Point3d(acView.CenterPoint.X,
                                    acView.CenterPoint.Y, 0);

      //  设置比例系数;
      double dScale = 0.5;

      //  基于当前视图中心按比例缩放视图;
      Zoom(new Point3d(), new Point3d(), pCenter, 1 / dScale);
  }
}
```

VBA/ActiveX 代码参考

```
Sub ZoomScale()
    Dim scalefactor As Double
    Dim scaletype As Integer
    scalefactor = 0.5
    scaletype = acZoomScaledRelative
    ThisDrawing.Application.ZoomScaled scalefactor, scaletype
```

```
End Sub
```

2.2.2.4 居中显示对象

我们可以通过用 CenterPoint 属性修改视图中心点来调整图形窗口中图像的位置。当修改视图中心点而不修改视图大小时，我们看到的是视图在屏幕上平移的效果。

移动当前图形到指定中心点

本例代码演示怎样使用（§2.2.2.1 *操作当前视图*）一节给出的 Zoom()函数修改当前视图的中心点。

Zoom()函数总共需要 4 个值，前两个为新声明的 3D 点并忽略掉，第三个值点(5, 5, 0)为新定义的视图中心点，最后一个值传入 1 表示当前视图大小不变。

VB.NET

```
<CommandMethod("ZoomCenter")> _
Public Sub ZoomCenter()
  '' 设置视图的中心点为（5,5,0）
  Zoom(New Point3d(), New Point3d(), New Point3d(5, 5, 0), 1)
End Sub
```

C#

```
[CommandMethod("ZoomCenter")]
static public void ZoomCenter()
{
  // 设置视图的中心点为（5,5,0）
  Zoom(new Point3d(), new Point3d(), new Point3d(5, 5, 0), 1);
}
```

⊟ VBA/ActiveX 代码参考

```
Sub ZoomCenter()
    Dim Center(0 To 2) As Double
    Dim magnification As Double
    Center(0) = 5: Center(1) = 5: Center(2) = 0
    magnification = 1
    ThisDrawing.Application.ZoomCenter Center, magnification
End Sub
```

2.2.2.5 显示图形范围和界限

图形的范围或界限，是用来定义让最外边的对象都出现在里边的边界，或者由当前空间界限定义的区域。

计算当前空间的范围

当前空间的范围可以通过下列属性从数据库中获得：

- **Extmin and Extmax** - 返回模型空间的范围；
- **Pextmin and Pextmax** - 返回当前图纸空间布局的范围；

一旦获取当前空间的范围，我们就可以计算当前视图的 Width 属性值和 Height 属性值。视图新的宽度值用下面公式计算：

```
dWidth = MaxPoint.X - MinPoint.X
```

视图新的高度值用下面公式计算：

```
dHeight = MaxPoint.Y - MinPoint.Y
```

计算出视图的宽度和高度后，就可以计算视图的中心点。视图中心点坐标可以用下面的公式获得：

```
dCenterX = (MaxPoint.X + MinPoint.X) * 0.5
dCenterY = (MaxPoint.Y + MinPoint.Y) * 0.5
```

计算当前空间的界限

要基于当前空间界限修改图形显示，我们使用数据库对象的 Limmin 和 Limmax、 Plimmin 和 Plimmax 这两对属性。定义当前空间界限的点返回后，我们就可以用前面提到的公式计算新视图的宽、高和中心点了。

示例：缩放到当前空间的范围和界限

本例代码演示如何使用（§2.2.2.1 *操作当前视图*）一节给出的 Zoom()函数显示当前空间的范围和界限。

Zoom()函数的 4 个参数中，前两个为定义显示区域的最小点和最大点，第三个为新声明的 3D 点并被忽略，最后一个参数在图形没有完整填充整个图形窗口时用来调整图像的大小。

VB.NET

```
<CommandMethod("ZoomExtents")> _
Public Sub ZoomExtents()
  '' 缩放到当前空间的范围
  Zoom(New Point3d(), New Point3d(), New Point3d(), 1.01075)
End Sub

<CommandMethod("ZoomLimits")> _
Public Sub ZoomLimits()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```
  '' 缩放到模型空间的界限
  Zoom(New Point3d(acCurDb.Limmin.X, acCurDb.Limmin.Y, 0), _
       New Point3d(acCurDb.Limmax.X, acCurDb.Limmax.Y, 0), _
       New Point3d(), 1)
End Sub
```

C#

```
[CommandMethod("ZoomExtents")]
static public void ZoomExtents()
{
  // 缩放到当前空间的范围;
  Zoom(new Point3d(), new Point3d(), new Point3d(), 1.01075);
}

[CommandMethod("ZoomLimits")]
static public void ZoomLimits()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
  // 放到模型空间的界限;
  Zoom(new Point3d(acCurDb.Limmin.X, acCurDb.Limmin.Y, 0),
       new Point3d(acCurDb.Limmax.X, acCurDb.Limmax.Y, 0),
       new Point3d(), 1);
}
```

VBA/ActiveX 代码参考

```
Sub ZoomExtents()
    ThisDrawing.Application.ZoomExtents
End Sub

Sub ZoomLimits()
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    point1(0) = ThisDrawing.GetVariable("LIMMIN")(0)
    point1(1) = ThisDrawing.GetVariable("LIMMIN")(1)
    point1(2) = 0#
    point2(0) = ThisDrawing.GetVariable("LIMMAX")(0)
    point2(1) = ThisDrawing.GetVariable("LIMMAX")(1)
    point2(2) = 0#
    ThisDrawing.Application.ZoomWindow point1, point2
End Sub
```

2.2.3 使用命名视图

我们可以对需要反复使用的视图进行命名并保存，当不再需要这个视图时，可以将其删除。

命名视图存储在 View 表里，View 表是图形数据库里的一个命名符号表。通过 Add()方法将新视图添加到 View 表，就可以创建一个命名视图。当我们往 View 表添加新的命名视图时，会生成一个默认的模型空间视图。

在创建视图时为其命名。视图的名称可以至多含 255 个字符，包括字母、数字以及美元符号（$）、连字符（-）、下划线（_）等特殊字符。

通过调用 ViewTableRecord 对象的 Erase()方法，可以将命名视图从 View 表中删除。

添加命名视图并将其设置为当前视图

下面的例子为图形添加一个命名视图并将其设置为当前视图。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("CreateNamedView")> _
Public Sub CreateNamedView()
  ''获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  ''启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      ''以读模式打开 View 表
      Dim acViewTbl As ViewTable
      acViewTbl = acTrans.GetObject(acCurDb.ViewTableId, OpenMode.ForRead)

      ''检查名为 View1 的视图是否存在
      If (acViewTbl.Has("View1") = False) Then
          ''以写模式打开 View 表
          acViewTbl.UpgradeOpen()

          ''新建一个 View 表记录并命名为 View1
          Dim acViewTblRec As ViewTableRecord = New ViewTableRecord()
          acViewTblRec.Name = "View1"

          ''添加到 View 表及事务
```

```
            acViewTbl.Add(acViewTblRec)
            acTrans.AddNewlyCreatedDBObject(acViewTblRec, True)

            '' 置 View1 为当前视图
            acDoc.Editor.SetCurrentView(acViewTblRec)

            '' 提交修改
            acTrans.Commit()
        End If

        '' 关闭事务
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("CreateNamedView")]
public static void CreateNamedView()
{
  //获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  //启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      //以读模式打开 View 表
      ViewTable acViewTbl;
      acViewTbl = acTrans.GetObject(acCurDb.ViewTableId,
                                    OpenMode.ForRead) as ViewTable;

      //检查名为 View1 的视图是否存在
      if (acViewTbl.Has("View1") == false)
      {
          //以写模式打开 View 表
          acViewTbl.UpgradeOpen();
          // 新建一个 View 表记录并命名为 View1
          ViewTableRecord acViewTblRec = new ViewTableRecord();
          acViewTblRec.Name = "View1";

          //添加到 View 表及事务
```

```
            acViewTbl.Add(acViewTblRec);
            acTrans.AddNewlyCreatedDBObject(acViewTblRec, true);

            //置 View1 为当前视图
            acDoc.Editor.SetCurrentView(acViewTblRec);

            //提交修改
            acTrans.Commit();
        }

        //关闭事务
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateNamedView()
    ' 往视图集合里添加一个命名视图
    Dim viewObj As AcadView
    Set viewObj = ThisDrawing.Views.Add("View1")

    ThisDrawing.ActiveViewport.SetView viewObj
End Sub
```

删除一个命名视图

下面的例子从图形中删除一个命名视图。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("EraseNamedView")> _
Public Sub EraseNamedView()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 View 表
      Dim acViewTbl As ViewTable
      acViewTbl = acTrans.GetObject(acCurDb.ViewTableId, OpenMode.ForRead)
```

```
      '' 检查名为‘View1’的视图是否存在
      If (acViewTbl.Has("View1") = True) Then
          '' 以写模式打开 View 表
          acViewTbl.UpgradeOpen()

          '' 获取命名视图的记录
          Dim acViewTblRec As ViewTableRecord
          acViewTblRec = acTrans.GetObject(acViewTbl("View1"),
OpenMode.ForWrite)

          ''从 View 表删除命名视图
          acViewTblRec.Erase()

          '' 提交修改
          acTrans.Commit()
      End If

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("EraseNamedView")]
public static void EraseNamedView()
{
  //获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  //启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 View 表
      ViewTable acViewTbl;
      acViewTbl = acTrans.GetObject(acCurDb.ViewTableId,
                                    OpenMode.ForRead) as ViewTable;

      // 检查名为‘View1’的视图是否存在
      if (acViewTbl.Has("View1") == true)
      {
```

```
            // 以写模式打开 View 表
            acViewTbl.UpgradeOpen();

            //获取命名视图的记录
            ViewTableRecord acViewTblRec;
            acViewTblRec = acTrans.GetObject(acViewTbl["View1"],
                                    OpenMode.ForWrite) as ViewTableRecord;

            //从 View 表删除命名视图
            acViewTblRec.Erase();

            //提交修改
            acTrans.Commit();
        }

        // 关闭事务
    }
}
```

VBA/ActiveX 代码参考

```
Sub EraseNamedView()
    On Error Resume Next
    Dim viewObj As AcadView
    Set viewObj = ThisDrawing.Views("View1")

    If Err = 0 Then
      ' 删除视图
      viewObj.Delete
    End If
End Sub
```

2.2.4 使用平铺视口

AutoCAD 通常是用一个充满整个图形区域的单一平铺视口来开始新的绘图。我们可以分割模型空间的绘图区域让它同时显示多个视口。例如，如果我们让完整视图和局部细节视图同时可见，那么，我们就能看到对局部细节视图的修改反映在整个图形上的效果。对每个平铺视口，我们都可以进行下列操作：

- 缩放，设置捕捉、栅格及 UCS 图标模式，在本视口恢复命名视图；

- 一个视口正在执行命令时到另一个视口去绘图；
- 命名视口配置以便重复使用；

我们可以用不同的配置显示平铺视口，怎样显示取决于我们想要看到的视图的数量和大小。模型空间的平铺视口存储在 Viewport 表里。

描述视口的更多内容和示例，参见《AutoCAD 用户手册》的“设置模型空间视口”。

平铺视口存储于 Viewport 表，每条 Viewport 表记录代表一个视口，而且，不像别的表记录，在 Viewport 表里，可以有多条同名的表记录。同名的每条表记录都用来控制视口的显示。

例如，名为“*Active”的 Viewport 表记录代表当前显示在模型空间上的平铺视口。

2.2.4.1 辨别和操作活动视口

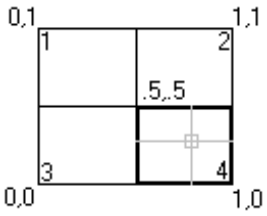
活动视口用 Viewport 表中名为“*Active”的记录表示，这个名字不是唯一的，因为当前在模型空间里显示的所有平铺视口的名字都是“*Active”。显示的每个视口会被赋予一个编码。活动视口的编码可通过下列方式获得：

- 检索系统变量 CVPORT 的值；
- 用 Editor 对象的 ActiveViewportId 属性获取活动视口的对象 ID，然后打开 Viewport 对象访问它的 Number 属性；

一旦取得活动视口，我们就可以控制它的显示属性，为视口启用栅格、对象捕捉等绘图辅助功能，甚至改变视口本身的大小等。平铺视口由对角两个点定义：左下角和右上角。LowerLeftCorner 属性和 UpperRightCorner 属性代表显示屏上视口的图形位置。

单个视口的配置为左下角（0，0）右上角（1，1）。不管模型空间里有多少个平铺视口，绘图窗口的左下角总是（0，0），右上角总是（1，1）。当显示的平铺视口多于一个时，每个视口的左下角和右上角会发生变化，但总会有这么两个视口，其中一个视口的左下角为（0，0）而另一个的右上角为（1，1）。

这些特性详细说明如下（以分割为四个视口为例）：



这个例子中：

- 视口 1 -左下角 = (0, .5)，右上角 = (.5, 1)
- 视口 2 -左下角 = (.5, .5)，右上角 = (1, 1)
- 视口 3 -左下角 = (0, 0)，右上角 = (.5, .5)
- 视口 4 -左下角 = (.5, 0)，右上角 = (1, .5)

创建一个有两个水平窗口的平铺视口配置

下例创建一个有两个水平视口的命名视口配置，并重新定义活动显示。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateModelViewport")> _
Public Sub CreateModelViewport()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      ''以读模式打开 Viewport 表
      Dim acVportTbl As ViewportTable
      acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId, OpenMode.ForRead)

      ''检查名为‘TEST_VIEWPORT’的视图是否存在
      If (acVportTbl.Has("TEST_VIEWPORT") = False) Then
          '' 以写模式打开 Viewport 表
          acVportTbl.UpgradeOpen()

          ''添加新视口到 Viewport 表并添加事务记录
          Dim acVportTblRecLwr As ViewportTableRecord = New ViewportTableRecord()
          acVportTbl.Add(acVportTblRecLwr)
          acTrans.AddNewlyCreatedDBObject(acVportTblRecLwr, True)

          '' 新视口命名为‘TEST_VIEWPORT’并将绘图窗口的下半部分赋给它
          acVportTblRecLwr.Name = "TEST_VIEWPORT"
          acVportTblRecLwr.LowerLeftCorner = New Point2d(0, 0)
          acVportTblRecLwr.UpperRightCorner = New Point2d(1, 0.5)

          '' 添加新视口到 Viewport 表并添加事务记录
          Dim acVportTblRecUpr As ViewportTableRecord = New ViewportTableRecord()
          acVportTbl.Add(acVportTblRecUpr)
          acTrans.AddNewlyCreatedDBObject(acVportTblRecUpr, True)

          '' 新视口命名为‘TEST_VIEWPORT’并将绘图窗口的上半部分赋给它
          acVportTblRecUpr.Name = "TEST_VIEWPORT"
```

```
        acVportTblRecUpr.LowerLeftCorner = New Point2d(0, 0.5)
        acVportTblRecUpr.UpperRightCorner = New Point2d(1, 1)

        '' 将新视口设为活动视口，需要删除名为'*Active'的视
        '' 口并基于‘TEST_VIEWPORT’重建
        '' 遍历符号表里的每个对象
        For Each acObjId As ObjectId In acVportTbl
            '' 以读模式打开对象
            Dim acVportTblRec As ViewportTableRecord
            acVportTblRec = acTrans.GetObject(acObjId, _
                                              OpenMode.ForRead)

            '' 检查是否为活动视口，是的话就删除
            If (acVportTblRec.Name = "*Active") Then
                acVportTblRec.UpgradeOpen()
                acVportTblRec.Erase()
            End If
        Next

        '' 复制新视口为活动视口
        For Each acObjId As ObjectId In acVportTbl
            '' 以读模式打开对象
            Dim acVportTblRec As ViewportTableRecord
            acVportTblRec = acTrans.GetObject(acObjId, _
                                              OpenMode.ForRead)

            '' 检查是否为活动视口，是的话就删除
            If (acVportTblRec.Name = "TEST_VIEWPORT") Then
                Dim acVportTblRecClone As ViewportTableRecord
                acVportTblRecClone = acVportTblRec.Clone()

                '' 添加新视口到 Viewport 表并添加事务记录
                acVportTbl.Add(acVportTblRecClone)
                acVportTblRecClone.Name = "*Active"
                acTrans.AddNewlyCreatedDBObject(acVportTblRecClone, True)
            End If
        Next

        '' 用新的平铺视口排列更新显示
        acDoc.Editor.UpdateTiledViewportsFromDatabase()

        '' 提交修改
        acTrans.Commit()
    End If
```

```
    '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateModelViewport")]
public static void CreateModelViewport()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Viewport 表
      ViewportTable acVportTbl;
      acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId,
                                     OpenMode.ForRead) as ViewportTable;

      //检查视图‘TEST_VIEWPORT’是否存在
      if (acVportTbl.Has("TEST_VIEWPORT") == false)
      {
          // 以写模式打开 Viewport 表
          acVportTbl.UpgradeOpen();

          //添加新视口到 Viewport 表并添加事务记录
          ViewportTableRecord acVportTblRecLwr = new ViewportTableRecord();
          acVportTbl.Add(acVportTblRecLwr);
          acTrans.AddNewlyCreatedDBObject(acVportTblRecLwr, true);

          // 新视口命名为‘TEST_VIEWPORT’并将绘图窗口的下半部分赋给它
          acVportTblRecLwr.Name = "TEST_VIEWPORT";
          acVportTblRecLwr.LowerLeftCorner = new Point2d(0, 0);
          acVportTblRecLwr.UpperRightCorner = new Point2d(1, 0.5);

          // 添加新视口到 Viewport 表并添加事务记录
          ViewportTableRecord acVportTblRecUpr = new ViewportTableRecord();
```

```
            acVportTbl.Add(acVportTblRecUpr);
            acTrans.AddNewlyCreatedDBObject(acVportTblRecUpr, true);

            // 新视口命名为‘TEST_VIEWPORT’并将绘图窗口的上半部分赋给它
            acVportTblRecUpr.Name = "TEST_VIEWPORT";
            acVportTblRecUpr.LowerLeftCorner = new Point2d(0, 0.5);
            acVportTblRecUpr.UpperRightCorner = new Point2d(1, 1);

            // 将新视口设为活动视口，需要删除名为'*Active'的视
            // 口并基于‘TEST_VIEWPORT’重建
            //遍历符号表里的每个对象
            foreach (ObjectId acObjId in acVportTbl)
            {
                // 以读模式打开对象
                ViewportTableRecord acVportTblRec;
                acVportTblRec = acTrans.GetObject(acObjId,
                                                  OpenMode.ForRead) as
ViewportTableRecord;

                // 检查是否为活动视口，是就删除
                if (acVportTblRec.Name == "*Active")
                {
                    acVportTblRec.UpgradeOpen();
                    acVportTblRec.Erase();
                }
            }

            // 复制新视口为活动视口
            foreach (ObjectId acObjId in acVportTbl)
            {
                // 以读模式打开对象
                ViewportTableRecord acVportTblRec;
                acVportTblRec = acTrans.GetObject(acObjId,
                                                  OpenMode.ForRead) as
ViewportTableRecord;

                //  检查是否为活动视口，是就删除
                if (acVportTblRec.Name == "TEST_VIEWPORT")
                {
                    ViewportTableRecord acVportTblRecClone;
                    acVportTblRecClone = acVportTblRec.Clone() as
ViewportTableRecord;

                    // 添加新视口到 Viewport 表并添加事务记录
```

```
                    acVportTbl.Add(acVportTblRecClone);
                    acVportTblRecClone.Name = "*Active";
                    acTrans.AddNewlyCreatedDBObject(acVportTblRecClone, true);
                }
            }

            // 用新的平铺视口排列更新显示
            acDoc.Editor.UpdateTiledViewportsFromDatabase();

            // 提交修改
            acTrans.Commit();
        }

        // 关闭事务
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateModelViewport()
    ' 新建一个视口
    Dim vportObj As AcadViewport
    Set vportObj = ThisDrawing.Viewports.Add("TEST_VIEWPORT")

    ' 将视口 vportObj 分割成 2 个水平窗口
    vportObj.Split acViewport2Horizontal

    ' 现在将 vportObj 设置为活动视口
    ThisDrawing.ActiveViewport = vportObj
End Sub
```

2.2.4.2 使平铺视口为当前视口

我们在当前视口进行输入点和选择对象的操作。要想使视口成为当前视口，需要使用系统变量 CVPORT 并将相应视口的编码传给该系统变量。

我们可以遍历已存在的视口来查找某个特定的视口。方法是，用视口的 Name 属性来识别 Viewport 表中名为“*Active”的表记录。

分割视口并遍历每个窗口

本例将活动视口分割为两个水平窗口，然后遍历图形中所有平铺视口并显示每个视口的名字、左下角和右上角。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SplitAndIterateModelViewports")> _
Public Sub SplitAndIterateModelViewports()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以写模式打开 Viewport 表
      Dim acVportTbl As ViewportTable
      acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId, _
                                     OpenMode.ForWrite)

      '' 以写模式打开当前视口
      Dim acVportTblRec As ViewportTableRecord
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                        OpenMode.ForWrite)

      Dim acVportTblRecNew As ViewportTableRecord = New ViewportTableRecord()

      '' 添加新视口到 Viewport 表，记录事务
      acVportTbl.Add(acVportTblRecNew)
      acTrans.AddNewlyCreatedDBObject(acVportTblRecNew, True)

      '' 新视口的 Name 设置为”*Active”
      acVportTblRecNew.Name = "*Active"

      '' 用当前左下角作为新视口的左下角
      acVportTblRecNew.LowerLeftCorner = acVportTblRec.LowerLeftCorner
      '' 获取当前右上角 X 值的一半
      acVportTblRecNew.UpperRightCorner = New
Point2d(acVportTblRec.UpperRightCorner.X, _

acVportTblRec.LowerLeftCorner.Y + _

((acVportTblRec.UpperRightCorner.Y - _
```

```
acVportTblRec.LowerLeftCorner.Y) / 2))
      '' 重新计算活动视口的两个角
      acVportTblRec.LowerLeftCorner = New
Point2d(acVportTblRec.LowerLeftCorner.X, _

acVportTblRecNew.UpperRightCorner.Y)

      '' 用新平铺视口布局更新显示
      acDoc.Editor.UpdateTiledViewportsFromDatabase()

      '' 遍历视口表中的每个对象
      For Each acObjId As ObjectId In acVportTbl
          '' 以读打开对象
          Dim acVportTblRecCur As ViewportTableRecord
          acVportTblRecCur = acTrans.GetObject(acObjId, _
                                               OpenMode.ForRead)

          If (acVportTblRecCur.Name = "*Active") Then
              Application.SetSystemVariable("CVPORT", acVportTblRecCur.Number)

              Application.ShowAlertDialog("Viewport: " & acVportTblRecCur.Number
& _
                                          " is now active." & _
                                          vbLf & "Lower left corner: " & _
                                          acVportTblRecCur.LowerLeftCorner.X & ",
" & _
                                          acVportTblRecCur.LowerLeftCorner.Y & _
                                          vbLf & "Upper right corner: " & _
                                          acVportTblRecCur.UpperRightCorner.X &
", " & _
                                          acVportTblRecCur.UpperRightCorner.Y)
          End If
      Next
      '' 提交修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
```

```
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SplitAndIterateModelViewports")]
public static void SplitAndIterateModelViewports()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以写模式打开 Viewport 表
      ViewportTable acVportTbl;
      acVportTbl = acTrans.GetObject(acCurDb.ViewportTableId,
                                     OpenMode.ForWrite) as ViewportTable;

      // 以写模式打开当前视口
      ViewportTableRecord acVportTblRec;
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                        OpenMode.ForWrite) as
ViewportTableRecord;

      ViewportTableRecord acVportTblRecNew = new ViewportTableRecord();

      // 添加新视口到 Viewport 表，记录事务
      acVportTbl.Add(acVportTblRecNew);
      acTrans.AddNewlyCreatedDBObject(acVportTblRecNew, true);

      // 新视口的 Name 设置为”*Active”
      acVportTblRecNew.Name = "*Active";

      // 用当前左下角作为新视口的左下角
      acVportTblRecNew.LowerLeftCorner = acVportTblRec.LowerLeftCorner;

      // 获取当前右上角 X 值的一半
      acVportTblRecNew.UpperRightCorner = new
Point2d(acVportTblRec.UpperRightCorner.X,

acVportTblRec.LowerLeftCorner.Y +

((acVportTblRec.UpperRightCorner.Y -

acVportTblRec.LowerLeftCorner.Y) / 2));
```

```
      // 重新计算活动视口的两个角
      acVportTblRec.LowerLeftCorner = new
Point2d(acVportTblRec.LowerLeftCorner.X,

acVportTblRecNew.UpperRightCorner.Y);

      // 用新平铺视口布局更新显示
      acDoc.Editor.UpdateTiledViewportsFromDatabase();

      // 遍历视口表中的每个对象
      foreach (ObjectId acObjId in acVportTbl)
      {
          // 以读打开对象
          ViewportTableRecord acVportTblRecCur;
          acVportTblRecCur = acTrans.GetObject(acObjId,
                                               OpenMode.ForRead) as
ViewportTableRecord;

          if (acVportTblRecCur.Name == "*Active")
          {
              Application.SetSystemVariable("CVPORT", acVportTblRecCur.Number);

              Application.ShowAlertDialog("Viewport: " + acVportTblRecCur.Number
                                  + " is now active." +
                                    "\nLower left corner: " +
                                    acVportTblRecCur.LowerLeftCorner.X + ", " +
                                    acVportTblRecCur.LowerLeftCorner.Y +
                                    "\nUpper right corner: " +
                                   acVportTblRecCur.UpperRightCorner.X + ", " +
                                   acVportTblRecCur.UpperRightCorner.Y);
          }
      }

      // 提交修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub SplitandInterateModelViewports()
    ' 获取当前视口
    Dim vportObj As AcadViewport
    Set vportObj = ThisDrawing.ActiveViewport
```

```
    ’ 将视口分割成 2 个窗口
    vportObj.Split acViewport2Horizontal

    ’ 遍历视口，
    ’ 突出显示每个视口并显示
    ’ 每个视口的右上角和左下角

    Dim vport As AcadViewport
    Dim LLCorner As Variant
    Dim URCorner As Variant

    For Each vport In ThisDrawing.Viewports
        ThisDrawing.ActiveViewport = vport
        LLCorner = vport.LowerLeftCorner
        URCorner = vport.UpperRightCorner
        MsgBox "Viewport: " & vport.Name & " is now active." & _
               vbCrLf & "Lower left corner: " & _
               LLCorner(0) & ", " & LLCorner(1) & vbCrLf & _
               "Upper right corner: " & _
               URCorner(0) & ", " & URCorner(1)
    Next vport
End Sub
```

2.2.5 更新文档窗口的几何信息

通过 AutoCAD .NET API 执行的许多操作都会修改绘图区域显示的内容，但不是所有的动作都立即更新图形显示。有这样的设计，我们就可以对图形进行多次修改而不必等待每次修改完都更新显示，相反我们可以将全部修改动作绑定在一起并在所有修改动作都完成后只执行一次更新显示的操作。

更新显示的方法，有 Application 对象和 Editor 对象的 UpdateScreen()方法，以及 Editor 对象的 Regen()方法。

UpdateScreen 方法重画应用程序窗口或文档窗口。Regen 方法重新生成绘图窗口中的图形对象，并重新计算所有对象的屏幕坐标和视图分解。Regen 方法还会重新索引图形数据库，以便优化图形显示性能和对象选择性能。

VB.NET

```
’’  重画图形
Application.UpdateScreen()
Application.DocumentManager.MdiActiveDocument.Editor.UpdateScreen()
```

```
''  重新生成图形
Application.DocumentManager.MdiActiveDocument.Editor.Regen()
```

C#

```
//  重画图形
Application.UpdateScreen();
Application.DocumentManager.MdiActiveDocument.Editor.UpdateScreen();

//  重新生成图形
Application.DocumentManager.MdiActiveDocument.Editor.Regen();
```

VBA/ActiveX 代码参考

```
''  重画图形
ThisDrawing.Application.Update

''  重新生成图形
ThisDrawing.Regen
```

2.3 新建、打开、保存和关闭图形

DocumentCollection 对象、Document 对象和 Database 对象提供了访问 AutoCAD®图形文件的方法。

VBA/ActiveX 交叉参考

VBA/ActiveX 类	.NET API 类
Documents collection	DocumentCollection
Document	Document and Database
Document.Saved	System.Convert.ToInt16(Application.GetSystemVariable("DBMOD"))

2.3.1 新建和打开图形文件

使用 DocumentCollection 对象提供的方法来创建新图形或打开已存在的图形。Add()方法基于图形样板创建一个新图形文件并将图形添加到 DocumentCollection 集合中。Open()方法打开一个已存在的图形文件。

创建新图形

本例使用 Add()方法基于图形样板文件 acad.dwt 创建一个新图形。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("NewDrawing", CommandFlags.Session)> _
Public Sub NewDrawing()
  '' 指定使用的样板，如果这个样板没找到，
  '' 就使用默认设置
  Dim strTemplatePath As String = "acad.dwt"

  Dim acDocMgr As DocumentCollection = Application.DocumentManager
  Dim acDoc As Document = acDocMgr.Add(strTemplatePath)
  acDocMgr.MdiActiveDocument = acDoc
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("NewDrawing", CommandFlags.Session)]
public static void NewDrawing()
{
  // 指定使用的样板，如果这个样板没找到，
  // 就使用默认设置
  string strTemplatePath = "acad.dwt";

  DocumentCollection acDocMgr = Application.DocumentManager;
  Document acDoc = acDocMgr.Add(strTemplatePath);
  acDocMgr.MdiActiveDocument = acDoc;
}
```

⊟ VBA/ActiveX 代码参考

```
Sub NewDrawing()
    Dim strTemplatePath As String
    strTemplatePath = "acad.dwt"

    Dim docObj As AcadDocument
    Set docObj = ThisDrawing.Application.Documents.Add(strTemplatePath)
End Sub
```

打开现有图形

本例使用 Open()方法打开一个已存在的图形。打开图形之前，程序会在试图打开文件前检查文件是否存在。

VB.NET

```
Imports System.IO
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("OpenDrawing", CommandFlags.Session)> _
Public Sub OpenDrawing()
  Dim strFileName As String = "C:\campus.dwg"
  Dim acDocMgr As DocumentCollection = Application.DocumentManager

  If (File.Exists(strFileName)) Then
      acDocMgr.Open(strFileName, False)
  Else
      acDocMgr.MdiActiveDocument.Editor.WriteMessage("File " & strFileName & _
                                                     " does not exist.")
  End If
End Sub
```

C#

```
using System.IO;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("OpenDrawing", CommandFlags.Session)]
public static void OpenDrawing()
{
  string strFileName = "C:\\campus.dwg";
  DocumentCollection acDocMgr = Application.DocumentManager;
```

```
  if (File.Exists(strFileName))
  {
      acDocMgr.Open(strFileName, false);
  }
  else
  {
      acDocMgr.MdiActiveDocument.Editor.WriteMessage("File " + strFileName +
                                                     " does not exist.");
  }
}
```

VBA/ActiveX 代码参考

```
Sub OpenDrawing()
    Dim dwgName As String
    dwgName = "c:\campus.dwg"
    If Dir(dwgName) <> "" Then
        ThisDrawing.Application.Documents.Open dwgName
    Else
        MsgBox "File " & dwgName & " does not exist."
    End If
End Sub
```

2.3.2 保存和关闭图形文件

保存数据库对象的内容使用 Database 对象的 SaveAs()方法。使用 SaveAs()方法时，可以指定是否需要对数据库重新命名，以及是否需要将硬盘上的图形备份重命名为备份文件（.bak 文件，通过将参数 bBakAndRename 设置为 True 实现），我们还可以通过检查系统变量 DWGTITLED 来确定数据库是否正在使用像 Drawing1、Drawing2 这样的默认文件名。如果系统变量 DWGTITLED 为 0，说明图形还没有被重新命名。

有时我们想要检查当前图形是否有尚未保存的修改。这项检查最好在退出 AutoCAD 或开始新图形之前进行。判断一个图形文件是否已经被修改，需要检查系统变量 DBMOD 的值。

关闭图形文件

关闭一个打开的图形使用Document 对象的CloseAndDiscard()方法和CloseAndSave()方法，其中 CloseAndDiscard()方法忽略所作的修改，CloseAndSave()方法保存所作的修改。还可以使用 DocumentCollection 的 CloseAll()方法关闭 AutoCAD 中打开的所有图形。

保存当前图形

本例将当前图形保存为“d:\MyDrawing.dwg”（要是还没保存或不是这个文件名的话）。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("SaveActiveDrawing")> _
Public Sub SaveActiveDrawing()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim strDWGName As String = acDoc.Name

  Dim obj As Object = Application.GetSystemVariable("DWGTITLED")

  '' 图形命名了吗? 0-没呢
  If System.Convert.ToInt16(obj) = 0 Then
      '' 如果图形使用了默认名 (Drawing1、Drawing2 等),
      ' '就提供一个新文件名

      strDWGName = "d:\MyDrawing.dwg"
  End If

  '' 保存图形
  acDoc.Database.SaveAs(strDWGName, True, DwgVersion.Current, _
                        acDoc.Database.SecurityParameters)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using  Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("SaveActiveDrawing")]
public static void SaveActiveDrawing()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  string strDWGName = acDoc.Name;

  object obj = Application.GetSystemVariable("DWGTITLED");

  //图形命名了吗? 0-没呢
  if (System.Convert.ToInt16(obj) == 0)
  {
      // 如果图形使用了默认名 (Drawing1、Drawing2 等),
      // 就提供一个新文件名
      strDWGName = "d:\\MyDrawing.dwg";
  }
```

```
  // 保存图形
  acDoc.Database.SaveAs(strDWGName, true, DwgVersion.Current,
                        acDoc.Database.SecurityParameters);
}
```

VBA/ActiveX 代码参考

```
Sub SaveActiveDrawing()
    ' 保存当前图形到新文件名下
    ThisDrawing.SaveAs "MyDrawing.dwg"
End Sub
```

判断图形是否有尚未保存的修改

本例检查图形，看看是否有尚未保存的修改，并由用户确认是否保存图形（如果不保存，则跳至结束）。如果用户确认保存，就使用 SaveAs()方法保存当前图形。代码如下：（注意在工程中添加引用 System.Windows.Forms.dll 。 -译者）

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("DrawingSaved")> _
Public Sub DrawingSaved()
  Dim obj As Object = Application.GetSystemVariable("DBMOD")

  ''检查系统变量 DBMOD 的值，0 表示没有未保存修改
  If Not (System.Convert.ToInt16(obj) = 0) Then
      If MsgBox("Do you wish to save this drawing?", _
                MsgBoxStyle.YesNo, _
                "Save Drawing") = MsgBoxResult.Yes Then
          Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
          acDoc.Database.SaveAs(acDoc.Name, True, DwgVersion.Current, _
                                acDoc.Database.SecurityParameters)
      End If
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
```

```
[CommandMethod("DrawingSaved")]
public static void DrawingSaved()
{
  object obj = Application.GetSystemVariable("DBMOD");

  // 检查系统变量 DBMOD 的值，0 表示没有未保存修改
  if (System.Convert.ToInt16(obj) != 0)
  {
      if (System.Windows.Forms.MessageBox.Show("Do you wish to save this
drawing?",
                                  "Save Drawing",
                                  System.Windows.Forms.MessageBoxButtons.YesNo,
                                  System.Windows.Forms.MessageBoxIcon.Question)
                                  == System.Windows.Forms.DialogResult.Yes)
      {
          Document acDoc = Application.DocumentManager.MdiActiveDocument;
          acDoc.Database.SaveAs(acDoc.Name, true, DwgVersion.Current,
                                  acDoc.Database.SecurityParameters);
      }
  }
}
```

VBA/ActiveX 代码参考

```
Sub DrawingSaved()
    If Not (ThisDrawing.Saved) Then
        If MsgBox("Do you wish to save this drawing?", _
                  vbYesNo) = vbYes Then
            ThisDrawing.Save
        End If
    End If
End Sub
```

2.3.3 没有文档打开时

AutoCAD 总是启动于新建一个文档或打开一个已存在的文档。也有可能，在当前运行过程中所有的文档都关闭了。

如果在 AutoCAD 用户界面下的所有文档都关闭了，我们会注意到应用程序窗口发生了一些变化。快速访问工具条和应用程序菜单只提供了有限的选项，这些有限的选项是有关创建和打开图形、显示图纸集管理器及恢复图形的。如果显示菜单条的话，也只出现简化了的文件、视图、窗口和帮助等菜单项。我们还会注意到这时候没有命令行栏。

工作在零文档状态时，我们可以进行下列操作：

- 新建文档或打开已存在文档；

- 自定义应用程序菜单和菜单条的零文档状态；
- 关掉 AutoCAD；

要想在 AutoCAD 进入零文档状态时对它作出反应，应使用 DocumentDestroyed 事件。DocumentDestroyed 事件在关闭文档时被触发。当关闭最后一个文档时，文档计数为 1。可以使用 DocumentManager 的 Count 属性来确定 DocumentDestroyed 事件被触发时打开的文档的个数。

更多的关于 AutoCAD 中使用事件的内容，参见（§7 *使用事件*）。

自定义应用程序菜单

本示例代码使用 DocumentDestroyed 事件监控最后一个文档何时关闭及何时进入零文档状态。一旦进入零文档状态，就将 Opening 事件注册到应用程序菜单。单击应用程序菜单就触发 Opening 事件。Opening 事件中向应用程序菜单添加一个新菜单项，单击新菜单项将显示一个消息框。

注：下例需要在工程中引用程序集 *AdWindows.dll*。程序集 AdWindows.dll 包含用来自定义应用程序菜单的命名空间，可以在 AutoCAD 的安装目录或 ObjectARX SDK 中找到。还需要引用程序集 WindowsBase，可以在“*添加引用*”对话框的.*NET*选项卡上找到。

VB.NET

```
Imports System.Windows.Input

Imports Autodesk.Windows
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

'' 为自定义应用程序菜单项创建命令处理器
Public Class MyCommandHandler
  Implements ICommand

  Event CanExecuteChanged(ByVal sender As Object, ByVal e As EventArgs) _
                                      Implements ICommand.CanExecuteChanged

  Function CanExecute(ByVal parameter As Object) As Boolean _
                                     Implements ICommand.CanExecute
      Return True
  End Function

  Sub Execute(ByVal parameter As Object) Implements ICommand.Execute
      Application.ShowAlertDialog("MyMenuItem has been clicked")
  End Sub
End Class

Public Class Chp02_3_3
```

```
  ''Global var for ZeroDocState
  Dim acApMenuItem As ApplicationMenuItem = Nothing

  <CommandMethod("AddZeroDocEvent")> _
  Public Sub AddZeroDocEvent()
      '' 获取 DocumentCollection 并注册 DocumentDestroyed 事件
      Dim acDocMgr As DocumentCollection = Application.DocumentManager
      AddHandler acDocMgr.DocumentDestroyed, AddressOf docDestroyed
  End Sub

  Public Sub docDestroyed(ByVal obj As Object, _
                          ByVal acDocDesEvtArgs As DocumentDestroyedEventArgs)

      '' 确定菜单项是否已存在
      '' 确定打开的文档数
      If Application.DocumentManager.Count = 1 And IsNothing(acApMenuItem) Then
          ''添加事件处理器来守候应用程序菜单
          ''记着添加引用 AdWindows.dll 啊~
          AddHandler ComponentManager.ApplicationMenu.Opening, _
                     AddressOf ApplicationMenu_Opening
      End If
  End Sub

  Public Sub ApplicationMenu_Opening(ByVal sender As Object, _
                                     ByVal e As EventArgs)
      '' 检查菜单项，看看之前添加过吗
      If IsNothing(acApMenuItem) Then
          '' 获取应用程序菜单组件
          Dim acApMenu As ApplicationMenu = ComponentManager.ApplicationMenu

          ''创建新菜单项
          acApMenuItem = New ApplicationMenuItem()
          acApMenuItem.Text = "MyMenuItem"
          acApMenuItem.CommandHandler = New MyCommandHandler()

          '' 添加新菜单项
          acApMenu.MenuContent.Items.Add(acApMenuItem)

          ''移除应用程序菜单项 Opening 事件的处理器
          RemoveHandler ComponentManager.ApplicationMenu.Opening, _
                        AddressOf ApplicationMenu_Opening
      End If
  End Sub
End Class
```

C#

```
using Autodesk.Windows;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

// 为自定义应用程序菜单项创建命令处理器
public class MyCommandHandler : System.Windows.Input.ICommand
{
  public bool CanExecute(object parameter)
  {
      return true;
  }

  public event EventHandler CanExecuteChanged;

  public void Execute(object parameter)
  {
      Application.ShowAlertDialog("MyMenuItem has been clicked");
  }
}

class Chp02_3_3
{
  //Global var for ZeroDocState 全局变量
  ApplicationMenuItem acApMenuItem = null;

  [CommandMethod("AddZeroDocEvent")]
  public void AddZeroDocEvent()
  {
      // 获取 DocumentCollection 并注册 DocumentDestroyed 事件
      DocumentCollection acDocMgr = Application.DocumentManager;
      acDocMgr.DocumentDestroyed +=
          new DocumentDestroyedEventHandler(docDestroyed);
  }

  public void docDestroyed(object obj,
                           DocumentDestroyedEventArgs acDocDesEvtArgs)
  {
      // 确定菜单项是否已存在
      // 确定打开的文档数
      if (Application.DocumentManager.Count == 1 && acApMenuItem == null)
      {
          // 添加事件处理器来守候应用程序菜单
          // 记着添加引用 AdWindows.dll 啊~
```

```
            ComponentManager.ApplicationMenu.Opening +=
                new EventHandler<EventArgs>(ApplicationMenu_Opening);
        }
    }

    void ApplicationMenu_Opening(object sender, EventArgs e)
    {
        // 检查菜单项，看看之前添加过吗
        if (acApMenuItem == null)
        {
            // 获取应用程序菜单组件
            ApplicationMenu acApMenu = ComponentManager.ApplicationMenu;

            // 创建新菜单项
            acApMenuItem = new ApplicationMenuItem();
            acApMenuItem.Text = "MyMenuItem";
            acApMenuItem.CommandHandler = new MyCommandHandler();

            // 追加新菜单项
            acApMenu.MenuContent.Items.Add(acApMenuItem);

            // 移除事件处理器
            ComponentManager.ApplicationMenu.Opening -=
                new EventHandler<EventArgs>(ApplicationMenu_Opening);
        }
    }
}
```

2.4 锁定和解锁文档

修改对象或访问 AutoCAD 的请求随时随地都会发生，为避免与其他请求冲突，我们有责任在修改前锁定文档。在某些情形下，未锁定文档会导致数据库更新过程出现锁冲突。当我们的应用程序进行下列操作时，需要锁定文档：

- 从无模式对话框与 AutoCAD 交互时；
- 访问已调入的文档而不是当前文档时；
- 应用程序作为 COM 服务器时；
- 用会话命令标志注册命令时

例如，向非当前文档的模型空间或图纸空间添加实体时，就需要锁定相应的文档。我们使用要锁定的数据库对象的 LockDocument()方法来锁定文档。调用 LockDocument()方法会返回一个 DocumentLock 对象。

一旦修改完已锁定数据库，就要将数据库解锁。解锁数据库，我们调用 DocumentLock 对象的 Dispose()方法。我们还可以使用 Using 语句，Using 语句运行结束，数据库也就解锁了。（DocumentLock 对象实现了 IDisposable 接口，因而可以使用 Using 语句。有关 Using 语句的内容请参考 C#语言中垃圾回收相关内容 - 译者）

注：运行没有使用会话命令标志的命令时，不需要在修改前锁定当前文档的数据库。

修改对象前锁定数据库

本例新建一个文档然后绘制一个圆。文档创建后，新文档的数据库被锁定，然后添加一个圆到文档，添加完圆后数据库解锁，相应文档窗口置为当前。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("LockDoc", CommandFlags.Session)> _
Public Sub LockDoc()
  '' 新建一个图形
  Dim acDocMgr As DocumentCollection = Application.DocumentManager
  Dim acNewDoc As Document = acDocMgr.Add("acad.dwt")
  Dim acDbNewDoc As Database = acNewDoc.Database

  '' 锁定新建文档
  Using acLckDoc As DocumentLock = acNewDoc.LockDocument()

      '' 启动新数据库的事务
      Using acTrans As Transaction =
acDbNewDoc.TransactionManager.StartTransaction()

          '' 以读模式打开块表
          Dim acBlkTbl As BlockTable
          acBlkTbl = acTrans.GetObject(acDbNewDoc.BlockTableId, _
                                       OpenMode.ForRead)

          '' 以写模式打开块表记录模型空间
          Dim acBlkTblRec As BlockTableRecord
          acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                          OpenMode.ForWrite)
```

```
            '' 在(5,5)创建一个半径为 3 的圆
            Dim acCirc As Circle = New Circle()
            acCirc.Center = New Point3d(5, 5, 0)
            acCirc.Radius = 3

            '' 添加新对象到模型空间和事务
            acBlkTblRec.AppendEntity(acCirc)
            acTrans.AddNewlyCreatedDBObject(acCirc, True)

            '' 提交修改
            acTrans.Commit()
        End Using

        '' 解锁文档
    End Using

    '' 设置为当前文档
    acDocMgr.MdiActiveDocument = acNewDoc
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("LockDoc", CommandFlags.Session)]
public static void LockDoc()
{
    // 新建图形
    DocumentCollection acDocMgr = Application.DocumentManager;
    Document acNewDoc = acDocMgr.Add("acad.dwt");
    Database acDbNewDoc = acNewDoc.Database;

    //  锁定新文档
    using (DocumentLock acLckDoc = acNewDoc.LockDocument())
    {
        // 启动新数据库事务
        using (Transaction acTrans =
acDbNewDoc.TransactionManager.StartTransaction())
        {
            // 以读模式打开块表
            BlockTable acBlkTbl;
```

```
            acBlkTbl = acTrans.GetObject(acDbNewDoc.BlockTableId,
                                         OpenMode.ForRead) as BlockTable;

            // 以写模式打开块表记录模型空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                            OpenMode.ForWrite) as
BlockTableRecord;

            // 在(5,5)创建一个半径为 3 的圆
            Circle acCirc = new Circle();
            acCirc.Center = new Point3d(5, 5, 0);
            acCirc.Radius = 3;

            // 添加新对象到模型空间和事务
            acBlkTblRec.AppendEntity(acCirc);
            acTrans.AddNewlyCreatedDBObject(acCirc, true);

            //提交修改
            acTrans.Commit();
        }

        //解锁文档（using 语句到此结束）
    }

    //将新文档置为当前
    acDocMgr.MdiActiveDocument = acNewDoc;
}
```

2.5 设置 AutoCAD 选项

AutoCAD .NET API 没有提供通过 AutoCAD 选项对话框访问选项的类和方法。要访问这些选项，必须使用 ActiveX® Automation 库。我们使用从 Application 对象的 Preferences 属性返回的 COM 对象来访问这些选项。

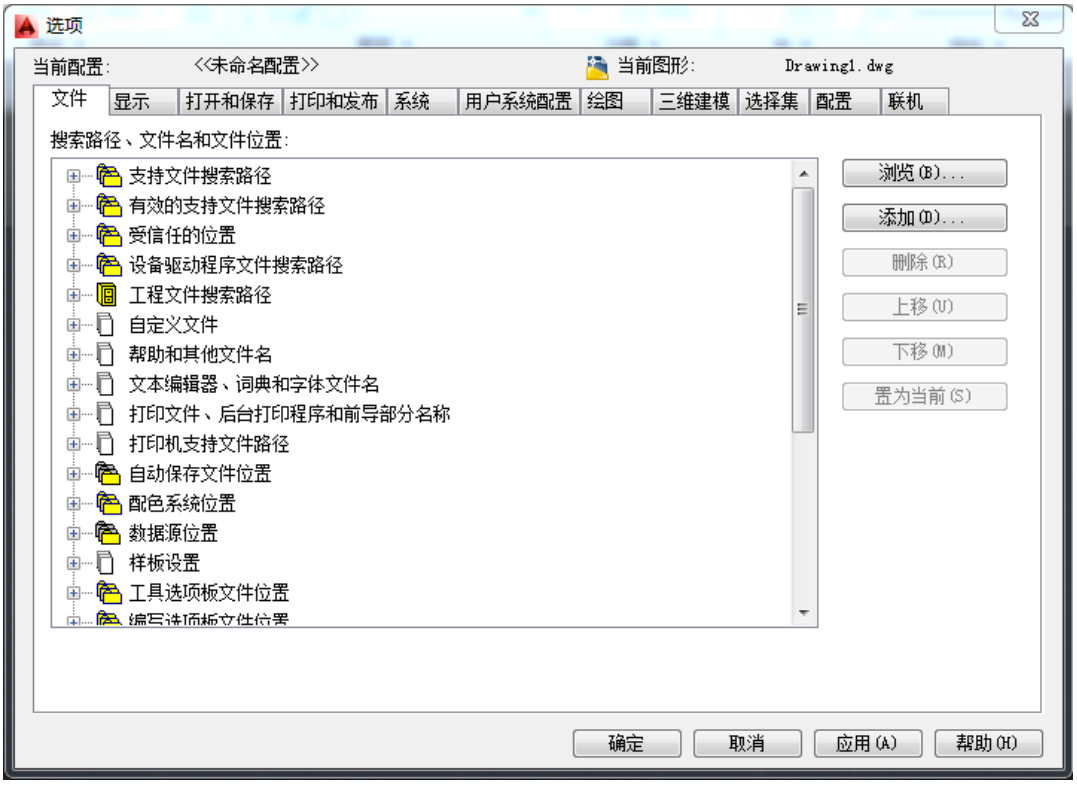


图 2-1 AutoCAD2014 的选项对话框

一旦获得 Preferences 对象，我们就可以访问附属于选项的九个对象，每个对象代表选项对话框的一个选项页。这些对象提供了访问选项对话框中存储在注册表里的全部选项的功能。通过使用这些对象的属性，我们可以自定义 AutoCAD 的许多设置。这些对象是：

（这九个对象统称为 Perferences 对象 - 译者注）

- PreferencesDisplay
- PreferencesDrafting
- PreferencesFiles
- PreferencesOpenSave
- PreferencesOutput
- PreferencesProfiles
- PreferencesSelection
- PreferencesSystem
- PreferencesUser

访问 Perferences 对象

下面示例演示如何通过 COM 交互操作访问 Perferences 对象。

VB.NET

```
Dim acPrefComObj As AcadPreferences = Application.Preferences
```

C#

```
AcadPreferences acPrefComObj = (AcadPreferences)Application.Preferences;
```

VBA/ActiveX 代码参考

```
Dim acadPref as AcadPreferences
Set acadPref = ThisDrawing.Application.Preferences
```

获得对 Perferences 对象的引用后，我们就可以使用 Display 属性、Drafting 属性、Files 属性、 OpenSave 属性、 Output 属性、 Profile 属性、 Selection 属性、 System 属性及 User 属性来访问指定的 Perferences 对象选项。

设置十字光标为全屏幕

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Interop

<CommandMethod("PrefsSetCursor")> _
Public Sub PrefsSetCursor()
  '' 本示例设置绘图窗口的十字光标为全屏
  '' 获得 Preferences 对象
  Dim acPrefComObj As AcadPreferences = Application.Preferences

  '' 使用 CursorSize 属性设置十字光标的大小
  acPrefComObj.Display.CursorSize = 100
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Interop;

[CommandMethod("PrefsSetCursor")]
public static void PrefsSetCursor()
{
  // 本示例设置绘图窗口的十字光标为全屏
  // 获得 Preferences 对象
```

```
  AcadPreferences acPrefComObj = (AcadPreferences)Application.Preferences;

  // 使用 CursorSize 属性设置十字光标的大小
  acPrefComObj.Display.CursorSize = 100;
}
```

VBA/ActiveX 代码参考

```
Sub PrefsSetCursor()
    ' 本示例设置绘图窗口的十字光标为全屏
    ' 获得 Preferences 对象
    Dim acadPref As AcadPreferences
    Set acadPref = ThisDrawing.Application.Preferences

    ' 使用 CursorSize 属性设置十字光标的大小
    acadPref.Display.CursorSize = 100
End Sub
```

隐藏滚动条

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Interop

<CommandMethod("PrefsSetDisplay")> _
Public Sub PrefsSetDisplay()
  '' 本例使滚动条失效

  '' 获得 Preferences 对象
  Dim acPrefComObj As AcadPreferences = Application.Preferences

  '' 不显示滚动条
  acPrefComObj.Display.DisplayScrollBars = False
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Interop;

[CommandMethod("PrefsSetDisplay")]
public static void PrefsSetDisplay()
{
  //本例使滚动条失效
```

```
  // 获得 Preferences 对象
  AcadPreferences acPrefComObj = (AcadPreferences)Application.Preferences;

  // 不显示滚动条
  acPrefComObj.Display.DisplayScrollBars = false;
}
```

VBA/ActiveX 代码参考

```
Sub PrefsSetDisplay()
    ' 本例使滚动条失效

    ' 获得 Preferences 对象
    Dim acadPref As AcadPreferences
    Set acadPref = ThisDrawing.Application.Preferences

    ' 不显示滚动条
    acadPref.Display.DisplayScrollBars = False
End Sub
```

2.5.1 数据库选项

除了应用程序级的选项设置外，还有基于图形的选项设置，这些选项设置存储在图形文件里，同样可以使用选项对话框来访问。要编程访问这些存储设置，使用 Database 对象的相应属性，或使用 Application 对象的 GetSystemVariable()方法和 SetSystemVariable()方法。

VBA/ActiveX 交叉参考

VBA/ActiveX 类	.NET API 类
DatabasePreferences	基于数据库和图形的系统变量

2.6 设置和返回系统变量

Application 对象提供了 SetSystemVariable()方法和 GetSystemVariable()方法来设置和提取 AutoCAD 系统变量的值。例如，要赋给系统变量 MAXSORT 一个整数值，使用下面的代码：

VB.NET

```
'' 获取系统变量的当前值
Dim nMaxSort as Integer = Application.GetSystemVariable("MAXSORT")

'' 给系统变量设置新值
Application.SetSystemVariable("MAXSORT", 100)
```

C#

```
// 获取系统变量的当前值
int nMaxSort = System.Convert.ToInt32(Application.GetSystemVariable("MAXSORT"));

// 给系统变量设置新值
Application.SetSystemVariable("MAXSORT", 100);
```

VBA/ActiveX 代码参考

```
'' 获取系统变量的当前值
Dim nMaxSort as Integer
nMaxSort = ThisDrawing.GetVariable("MAXSORT")

'' 给系统变量设置新值
ThisDrawing.SetVariable "MAXSORT", 100
```

2.7 精确绘图

使用 AutoCAD，我们可以进行精确的几何制图而无需进行繁琐的计算。通常我们无需知道坐标就能精确指定点。不用离开制图屏幕，我们就可以对图形执行计算并显示各种不同的状态信息。

2.7.1 调整捕捉和栅格对齐

栅格是可以度量距离的视觉化导线，捕捉模式用来限制光标移动。除了设置栅格间距和捕捉模式，我们还可以调整捕捉旋转角度和捕捉类型。

如果需要沿某一基线或角度绘图，我们可以旋转捕捉角度。捕捉角度旋转的中心点就是捕捉的基点。

注：修改了活动视口的捕捉和栅格设置后，应调用 Editor 对象的 UpdateTiledViewportsFromDatabase()方法更新一下绘图区域的显示。

捕捉和栅格不会影响通过.NET API 指定的点，但是，当使用 GetPoint()方法或 GetEntity()方法要求用户输入点时，用户在绘图区域指定的点会受到影响。关于使用和设置捕捉和栅格的更多内容，见《AutoCAD 用户指南》中的“调整栅格及栅格捕捉”一节。

修改栅格和捕捉设置

本例修改捕捉基点到(1,1)，并修改捕捉旋转角为 30 度。打开栅格并调整间距，使修改可见。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry
<CommandMethod("ChangeGridAndSnap")> _
Public Sub ChangeGridAndSnap()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
   '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 打开当前视口
      Dim acVportTblRec As ViewportTableRecord
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                        OpenMode.ForWrite)
      '' 打开栅格
      acVportTblRec.GridEnabled = True
      '' 调整栅格间距为 1,1
      acVportTblRec.GridIncrements = New Point2d(1, 1)
      '' 打开当前视口的捕捉模式
      acVportTblRec.SnapEnabled = True
      '' 调整捕捉间距为 0.5，0.5
      acVportTblRec.SnapIncrements = New Point2d(0.5, 0.5)
      '' 修改捕捉基点为 1，1
      acVportTblRec.SnapBase = New Point2d(1, 1)
      '' 修改捕捉旋转角为 30 度（0.524 弧度）
      acVportTblRec.SnapAngle = 0.524
      '' 更新平铺视口的显示
      acDoc.Editor.UpdateTiledViewportsFromDatabase()
      '' 提交修改，关闭事务
      acTrans.Commit()
```

```
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Runtime;

namespace FunGridSnap
{
    public class Class1
    {
        [CommandMethod("ChangeGridAndSnap")]
        public static void ChangeGridAndSnap()
        {
            // 获取当前数据库
            Document acDoc = Application.DocumentManager.MdiActiveDocument;
            Database acCurDb = acDoc.Database;
            // 启动事务
            using (Transaction acTrans =
acCurDb.TransactionManager.StartTransaction())
            {
                // 打开当前视口
                ViewportTableRecord acVportTblRec;
                acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                                  OpenMode.ForWrite) as
ViewportTableRecord;
                // 打开栅格
                acVportTblRec.GridEnabled = true;
                // 调整栅格间距为 1，1
                acVportTblRec.GridIncrements = new Point2d(1, 1);
                // 打开当前视口的捕捉模式
                acVportTblRec.SnapEnabled = true;
                // 调整捕捉间距为 0.5，0.5
                acVportTblRec.SnapIncrements = new Point2d(0.5, 0.5);
                // 修改捕捉基点为 1，1
                acVportTblRec.SnapBase = new Point2d(1, 1);
                // 修改捕捉旋转角为 30 度（0.524 弧度）
                acVportTblRec.SnapAngle = 0.524;
                // 更新平铺视口的显示
                acDoc.Editor.UpdateTiledViewportsFromDatabase();
                //提交修改，关闭事务
                acTrans.Commit();
```

```
            }
        }
    }
}
```

VBA/ActiveX 代码参考

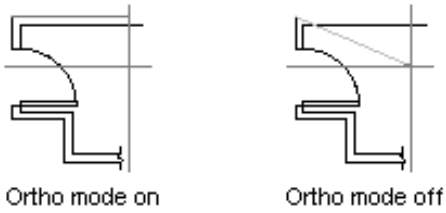
```
Sub ChangeGridAndSnap()
    ' 打开栅格
    ThisDrawing.ActiveViewport.GridOn = True
    ' 调整栅格间距为 1，1
    ThisDrawing.ActiveViewport.SetGridSpacing 1, 1
    ' 打开当前视口的捕捉模式
    ThisDrawing.ActiveViewport.SnapOn = True
    ' 调整捕捉间距为 0.5，0.5
    ThisDrawing.ActiveViewport.SetSnapSpacing 0.5, 0.5
    ' 修改捕捉基点为 1，1
    Dim newBasePoint(0 To 1) As Double
    newBasePoint(0) = 1: newBasePoint(1) = 1
    ThisDrawing.ActiveViewport.SnapBasePoint = newBasePoint
    ' 修改捕捉旋转角为 30 度（0.524 弧度）
    Dim rotationAngle As Double
    rotationAngle = 0.524
    ThisDrawing.ActiveViewport.SnapRotationAngle = rotationAngle
    ' 重置视口
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
End Sub
```

2.7.2 使用正交模式

绘制直线或移动对象时，我们可以使用正交模式将光标限制在水平方向或垂直方向。正交对齐依赖于当前捕捉角和用户坐标系。正交模式应用于需要指定第 2 个点的情形下，比如当使用 GetDistance()方法或 GetAngle()方法时。使用正交，不仅可以建立垂直对齐或水平对齐，还可以强制平行，或创建有规律的偏移。

通过让 AutoCAD 使用正交约束，我们可以更快捷地绘制图形。例如，通过在开始绘图前打开正交模式，我们可以创建一系列垂直直线。由于直线被限制在水平或垂直方向上，我们知道绘出来的线肯定是水平或垂直的，所以可以绘得更快。

下图分别演示了正交模式打开（Ortho mode on）与关闭（Ortho mode off）时绘图的情形：



下面的代码行的作用是打开正交模式。与栅格设置及捕捉设置不一样，正交模式在 Database 对象中维护，而不是在活动视口维护。

VB.NET

```
Application.DocumentManager.MdiActiveDocument.Database.Orthomode = True
```

C#

```
Application.DocumentManager.MdiActiveDocument.Database.Orthomode = true;
```

⊟ VBA/ActiveX 代码参考

```
ThisDrawing.ActiveViewport.OrthoOn = True
```

2.7.3 计算点和值

通过使用 Editor 对象提供的方法，以及 Geometry 命名空间和 Runtime 命名空间提供的方法，我们可以快速解决数学问题，还可以在图形中快速定位点。这些方法有：

- 使用 GetDistanceTo()方法和 DistanceTo()方法获取两个 2D 点或 3D 点间的距离；
- 使用 GetVectorTo()方法的返回值的 Angle 属性获取两个 2D 点对 x 轴的夹角；
- 使用 StringToAngle()方法将字串型角度值转换为实数值（双精度）；
- 使用 AngleToString()方法将角度从实数值（双精度）转换为字串值；
- 使用 StringToDistance()方法将距离从字串转换为实数值（双精度）；
- 使用 GetDistance()方法求出用户输入的两点之间的距离；

注：

.NET API 没有提供根据距离和角（极轴点）计算点的方法，以及在不同坐标系间转换坐标的方法。如果需要这些工具方法，要从 ActiveX Automation 库调用 PolarPoint()和 TranslateCoordinates()方法。

获取相对于 x 轴的角度

本例计算两点间的矢量，以及矢量相对于 x 轴的角度。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AngleFromXAxis")> _
Public Sub AngleFromXAxis()
  Dim pt1 As Point2d = New Point2d(2, 5)
  Dim pt2 As Point2d = New Point2d(5, 2)

  Application.ShowAlertDialog("Angle from XAxis: " & _
                              pt1.GetVectorTo(pt2).Angle.ToString())
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AngleFromXAxis")]
public static void AngleFromXAxis()
{
  Point2d pt1 = new Point2d(2, 5);
  Point2d pt2 = new Point2d(5, 2);

  Application.ShowAlertDialog("Angle from XAxis: " +
                              pt1.GetVectorTo(pt2).Angle.ToString());
}
```

VBA/ActiveX 代码参考

```
Sub AngleFromXAxis()
    ' 本例已知两点 pt1 和 pt2，求两点构成的矢量与 X 轴的夹角，单位弧度

    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double
    Dim retAngle As Double

    pt1(0) = 2: pt1(1) = 5: pt1(2) = 0
    pt2(0) = 5: pt2(1) = 2: pt2(2) = 0

    ' 计算并返回角
    retAngle = ThisDrawing.Utility.AngleFromXAxis(pt1, pt2)
```

```
    ' 显示计算结果
    MsgBox "The angle in radians between the X axis is " & retAngle
End Sub
```

计算极轴点

本例已知基点、角度、距离，求点的坐标。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.Geometry

Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

  Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                     pPt.Y + dDist * Math.Sin(dAng))
End Function

Public Shared Function PolarPoints(ByVal pPt As Point3d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

  Return New Point3d(pPt.X + dDist * Math.Cos(dAng), _
                     pPt.Y + dDist * Math.Sin(dAng), _
                     pPt.Z)
End Function

<CommandMethod("PolarPoints")> _
Public Sub PolarPoints()
  Dim pt1 As Point2d
  pt1 = PolarPoints(New Point2d(5, 2), 0.785398, 12)

  Application.ShowAlertDialog(vbLf & "PolarPoint: " & _
                              vbLf & "X = " & pt1.X & _
                              vbLf & "Y = " & pt1.Y)

  Dim pt2 As Point3d
  pt2 = PolarPoints(New Point3d(5, 2, 0), 0.785398, 12)
```

```
  Application.ShowAlertDialog(vbLf & "PolarPoint: " & _
                              vbLf & "X = " & pt2.X & _
                              vbLf & "Y = " & pt2.Y & _
                              vbLf & "Z = " & pt2.Z)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
{
  return new Point2d(pPt.X + dDist * Math.Cos(dAng),
                     pPt.Y + dDist * Math.Sin(dAng));
}

static Point3d PolarPoints(Point3d pPt, double dAng, double dDist)
{
  return new Point3d(pPt.X + dDist * Math.Cos(dAng),
                     pPt.Y + dDist * Math.Sin(dAng),
                     pPt.Z);
}

[CommandMethod("PolarPoints")]
public static void PolarPoints()
{
  Point2d pt1 = PolarPoints(new Point2d(5, 2), 0.785398, 12);

  Application.ShowAlertDialog("\nPolarPoint: " +
                              "\nX = " + pt1.X +
                              "\nY = " + pt1.Y);

  Point3d pt2 = PolarPoints(new Point3d(5, 2, 0), 0.785398, 12);

  Application.ShowAlertDialog("\nPolarPoint: " +
                              "\nX = " + pt2.X +
                              "\nY = " + pt2.Y +
                              "\nZ = " + pt2.Z);
}
```

⊟ VBA/ActiveX 代码参考

```
Sub PolarPoints()
    '本例已知到基点的距离和角度，求这个点的坐标
```

```
    Dim polarPnt As Variant
    Dim basePnt(0 To 2) As Double
    Dim angle As Double
    Dim distance As Double

    basePnt(0) = 2#: basePnt(1) = 2#: basePnt(2) = 0#
    angle = 0.785398
    distance = 6
    polarPnt = ThisDrawing.Utility.PolarPoint(basePnt, angle, distance)

    MsgBox vbLf + "PolarPoint: " + _
           vbLf + "X = " + CStr(polarPnt(0)) + _
           vbLf + "Y = " + CStr(polarPnt(1)) + _
           vbLf + "Z = " + CStr(polarPnt(2))
End Sub
```

用 GetDistance 方法计算两点间距离

本例使用 GetDistance()方法，先获得两个点，然后计算出两点间距离并显示出来。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetDistanceBetweenTwoPoints")> _
Public Sub GetDistanceBetweenTwoPoints()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pDblRes As PromptDoubleResult
  pDblRes = acDoc.Editor.GetDistance(vbLf & "Pick two points: ")

  Application.ShowAlertDialog(vbLf & "Distance between points: " & _
                              pDblRes.Value.ToString())
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetDistanceBetweenTwoPoints")]
```

```
public static void GetDistanceBetweenTwoPoints()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptDoubleResult pDblRes;
  pDblRes = acDoc.Editor.GetDistance("\nPick two points: ");

  Application.ShowAlertDialog("\nDistance between points: " +
                              pDblRes.Value.ToString());
}
```

VBA/ActiveX 代码参考

```
Sub GetDistanceBetweenTwoPoints()
    Dim returnDist As Double

    ' 提示用户输入两个点，返回两点间的距离
    returnDist = ThisDrawing.Utility.GetDistance(, "Pick two points.")

    MsgBox "The distance between the two points is: " & returnDist
End Sub
```

2.7.4 计算面积

我们可以使用 Area 属性计算下面这些实体的面积：圆弧、圆、椭圆、优化多段线、多段线、面域、填充、平面闭合样条曲线、及其他从基类 Curve 派生的实体。

如果需要计算多个对象的组合面积，我们可以求出每个对象的面积再汇总，或者，对一系列面域使用 Boolean()方法获得代表所求面积的单一面域。对这个单一面域，再使用 Area 属性求出其面积。

计算所得的面积因所查询对象的类型不同而有所不同。有关每种类型对象面积的计算方法的解释，见《AutoCAD 用户指南》“获取 Area 属性及 Mass 属性信息”一节。

2.7.4.1 计算给定面积

如果需要计算的面积是基于用户指定的几个点构成的，我们应考虑创建一个内存对象，比如轻量多段线等，然后在放弃这个对象前查询其面积。下列步骤解释实现的过程：

1. 使用 GetPoint()方法循环获取用户输入的点；
2. 用这些点创建轻量多段线。新建一个 Polyline 对象，指定顶点数及各点位置；
3. 使用 Area 属性获取新建的多段线的面积；

4. 使用多段线的 Dispose()方法销毁多段线（释放内存）。

计算由用户输入点定义的面积

本例提示用户输入 5 个点，然后用这 5 个点创建一个多段线。多段线是闭合的，查询多段线面积并显示在消息框内。因为无需将多段线添加到块中，因此命令结束前要将其销毁以释放内存。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("CalculateDefinedArea")> _
Public Sub CalculateDefinedArea()
  '' 提示用户输入 5 个点
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pPtRes As PromptPointResult
  Dim colPt As Point2dCollection = New Point2dCollection
  Dim pPtOpts As PromptPointOptions = New PromptPointOptions("")

  ''提示输入第 1 个点
  pPtOpts.Message = vbLf & "Specify first point: "
  pPtRes = acDoc.Editor.GetPoint(pPtOpts)
  colPt.Add(New Point2d(pPtRes.Value.X, pPtRes.Value.Y))

  '' 如果用户按 ESC 键，或取消命令，就退出
  If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

  Dim nCounter As Integer = 1

  While (nCounter <= 4)
      ''  提示下一个点
      Select Case nCounter
          Case 1
              pPtOpts.Message = vbLf & "Specify second point: "
          Case 2
              pPtOpts.Message = vbLf & "Specify third point: "
          Case 3
              pPtOpts.Message = vbLf & "Specify fourth point: "
          Case 4
```

```
                pPtOpts.Message = vbLf & "Specify fifth point: "
        End Select

        '' 用前一个点作基点
        pPtOpts.UseBasePoint = True
        pPtOpts.BasePoint = pPtRes.Value

        pPtRes = acDoc.Editor.GetPoint(pPtOpts)
        colPt.Add(New Point2d(pPtRes.Value.X, pPtRes.Value.Y))

        If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

        nCounter = nCounter + 1
    End While

    '' 用 5 个点创建多段线
    '' 所有的 2D 实体对象和 3D 实体对象都实现了 IDisposable，故可以使用 using 语句
    Using acPoly As Polyline = New Polyline()
        acPoly.AddVertexAt(0, colPt(0), 0, 0, 0)
        acPoly.AddVertexAt(1, colPt(1), 0, 0, 0)
        acPoly.AddVertexAt(2, colPt(2), 0, 0, 0)
        acPoly.AddVertexAt(3, colPt(3), 0, 0, 0)
        acPoly.AddVertexAt(4, colPt(4), 0, 0, 0)

        '' 闭合多段线
        acPoly.Closed = True

        '' 查询闭合多段线的面积
        Application.ShowAlertDialog("Area of polyline: " & _
                                    acPoly.Area.ToString())

        '' 销毁多段线
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("CalculateDefinedArea")]
```

```
public static void CalculateDefinedArea()
{
  // 提示用户输入 5 个点
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptPointResult pPtRes;
  Point2dCollection colPt = new Point2dCollection();
  PromptPointOptions pPtOpts = new PromptPointOptions("");

  // 提示输入第 1 个点
  pPtOpts.Message = "\nSpecify first point: ";
  pPtRes = acDoc.Editor.GetPoint(pPtOpts);
  colPt.Add(new Point2d(pPtRes.Value.X, pPtRes.Value.Y));

  // 如果用户按 ESC 键或取消命令就退出
  if (pPtRes.Status == PromptStatus.Cancel) return;

  int nCounter = 1;

  while (nCounter <= 4)
  {
      //  提示下一个点
      switch(nCounter)
      {
          case 1:
              pPtOpts.Message = "\nSpecify second point: ";
              break;
          case 2:
              pPtOpts.Message = "\nSpecify third point: ";
              break;
          case 3:
              pPtOpts.Message = "\nSpecify fourth point: ";
              break;
          case 4:
              pPtOpts.Message = "\nSpecify fifth point: ";
              break;
      }

      // 用前一个点作基点
      pPtOpts.UseBasePoint = true;
      pPtOpts.BasePoint = pPtRes.Value;

      pPtRes = acDoc.Editor.GetPoint(pPtOpts);
      colPt.Add(new Point2d(pPtRes.Value.X, pPtRes.Value.Y));
```

```
        if (pPtRes.Status == PromptStatus.Cancel) return;

        // 计数加 1
        nCounter = nCounter + 1;
    }

    // 用 5 个点创建多段线
    // 所有的 2D 实体对象和 3D 实体对象都实现了 IDisposable，故可以使用 using 语句
    using (Polyline acPoly = new Polyline())
    {
        acPoly.AddVertexAt(0, colPt[0], 0, 0, 0);
        acPoly.AddVertexAt(1, colPt[1], 0, 0, 0);
        acPoly.AddVertexAt(2, colPt[2], 0, 0, 0);
        acPoly.AddVertexAt(3, colPt[3], 0, 0, 0);
        acPoly.AddVertexAt(4, colPt[4], 0, 0, 0);

        // 闭合多段线
        acPoly.Closed = true;

        // 查询多段线面积
        Application.ShowAlertDialog("Area of polyline: " +
                                    acPoly.Area.ToString());

        // 销毁多段线
    }
}
```

VBA/ActiveX 代码参考

本例与上面.NET API 例子的过程相同，所不同的是，本例里多段线不是创建在内存中，而是作为数据库驻留对象添加到模型空间，因此获取多段线面积后，将其删除。

```
Sub CalculateDefinedArea()
    Dim p1 As Variant
    Dim p2 As Variant
    Dim p3 As Variant
    Dim p4 As Variant
    Dim p5 As Variant

    ' 获取 5 个点
    p1 = ThisDrawing.Utility.GetPoint(, vbCrLf & "Specify first point: ")
```

```
    p2 = ThisDrawing.Utility.GetPoint(p1, vbCrLf & "Specify second point: ")
    p3 = ThisDrawing.Utility.GetPoint(p2, vbCrLf & "Specify third point: ")
    p4 = ThisDrawing.Utility.GetPoint(p3, vbCrLf & "Specify fourth point: ")
    p5 = ThisDrawing.Utility.GetPoint(p4, vbCrLf & "Specify fifth point: ")

    ' 创建 2D 多段线
    Dim polyObj As AcadLWPolyline
    Dim vertices(0 To 9) As Double
    vertices(0) = p1(0): vertices(1) = p1(1)
    vertices(2) = p2(0): vertices(3) = p2(1)
    vertices(4) = p3(0): vertices(5) = p3(1)
    vertices(6) = p4(0): vertices(7) = p4(1)
    vertices(8) = p5(0): vertices(9) = p5(1)
    Set polyObj = ThisDrawing.ModelSpace.AddLightWeightPolyline _
                      (vertices)
    polyObj.Closed = True

    ' 显示闭合多段线的面积
    MsgBox "The area defined by the points is " & _
           polyObj.Area, , "Calculate Defined Area"

    ' 删除多段线
    polyObj.Delete
End Sub
```

2.8 提示用户输入

用户输入方法由 Document 对象的派生类 Editor 对象定义。用户输入方法在 AutoCAD 命令行或动态输入提示框里显示一个提示，请求各种不同类型的输入。这种用户输入在交互输入屏幕坐标、选择实体、输入短字串及数值时特别有用。如果程序需要输入多个选项或值时，用 Windows 窗体比单个的提示更合适。

每个用户输入方法都会在命令行显示一个可选提示，并返回一个和所需类型相符的值。例如，GetString()方法返回一个 PromptResult 类型的值，该值允许我们确定 GetString()方法的状态并取回用户所输入的值。每个用户输入方法都有与之相应的返回值。

输入方法接受一个字符串用来显示提示信息，或接受一个指定对象类型用以控制来自用户的输入。这些对象类型用来控制诸如 NULL 输入（按了 Enter 键）、基点、输入了 0 或负数、乱七八糟的文本输入等情况。

要强制提示本身显示在一行上，使用 VB.NET 时可以在提示字符串开头加上回车/换行常量（vbCrLf）或换行常量（vbLf），用 C#的话就在字符串前加上“\n”。

2.8.1 GetString()方法

GetString()方法提示用户在 Command 提示光标处输入一个字符串。PromptStringOptions 对象用来控制用户的输入及提示信息的显示方式。PromptStringOptions 对象的 AllowSpaces 属性控制是否可以输入空格，如果设置为 false，按空格键就终止输入。

从 AutoCAD 命令行获取用户输入的字符串

下面例子显示“Enter Your Name”的提示，并要求用户按 Enter 键完成输入（允许输入空格）。最后在消息框内显示输入的字符串。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetStringFromUser")> _
Public Sub GetStringFromUser()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pStrOpts As PromptStringOptions = New PromptStringOptions(vbLf & _
                                                                "Enter your name:
")
  pStrOpts.AllowSpaces = True
  Dim pStrRes As PromptResult = acDoc.Editor.GetString(pStrOpts)

  Application.ShowAlertDialog("The name entered was: " & _
                              pStrRes.StringResult)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetStringFromUser")]
```

```
public static void GetStringFromUser()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptStringOptions pStrOpts = new PromptStringOptions("\nEnter your name: ");
  pStrOpts.AllowSpaces = true;
  PromptResult pStrRes = acDoc.Editor.GetString(pStrOpts);

  Application.ShowAlertDialog("The name entered was: " +
                              pStrRes.StringResult);
}
```

VBA/ActiveX 代码参考

```
Sub GetStringFromUser()
    Dim retVal As String
    retVal = ThisDrawing.Utility.GetString _
                                    (1, vbCrLf & "Enter your name: ")
    MsgBox "The name entered was: " & retVal
End Sub
```

2.8.2 GetPoint()方法

GetPoint()方法提示用户在 Command 提示时指定一个点。PromptPointOptions 对象用来控制用户的输入及提示信息的显示方式。PromptPointOptions 对象的 UseBasePoint 属性和 BasePoint 属性控制是否从基点绘制一条橡皮筋儿线。PromptPointOptions 对象的 Keywords 属性用来定义除了指定点外还可以在 Command 提示光标处输入的关键字。

获取用户选取的点

下例提示用户选取两个点，然后用这两个点作为起止点画一条线。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetPointsFromUser")> _
Public Sub GetPointsFromUser()
  '' 获取当前数据库，启动事务管理器
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```
Dim pPtRes As PromptPointResult
Dim pPtOpts As PromptPointOptions = New PromptPointOptions("")

'' 提示起点
pPtOpts.Message = vbLf & "Enter the start point of the line: "
pPtRes = acDoc.Editor.GetPoint(pPtOpts)
Dim ptStart As Point3d = pPtRes.Value

'' 如果用户按 ESC 键或取消命令，就退出
If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

'' 提示终点
pPtOpts.Message = vbLf & "Enter the end point of the line: "
pPtOpts.UseBasePoint = True
pPtOpts.BasePoint = ptStart
pPtRes = acDoc.Editor.GetPoint(pPtOpts)
Dim ptEnd As Point3d = pPtRes.Value

If pPtRes.Status = PromptStatus.Cancel Then Exit Sub

'' 启动事务
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    Dim acBlkTbl As BlockTable
    Dim acBlkTblRec As BlockTableRecord

    '' 以写模式打开模型空间
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                 OpenMode.ForRead)

    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)

    '' 创建直线
    Dim acLine As Line = New Line(ptStart, ptEnd)

    '' 添加直线
    acBlkTblRec.AppendEntity(acLine)
    acTrans.AddNewlyCreatedDBObject(acLine, True)

    '' 缩放图形，全部显示
    acDoc.SendStringToExecute("._zoom _all ", True, False, False)
```

```
      '' 提交修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetPointsFromUser")]
public static void GetPointsFromUser()
{
  // 获取当前数据库，启动事务管理器
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  PromptPointResult pPtRes;
  PromptPointOptions pPtOpts = new PromptPointOptions("");

  // 提示起点
  pPtOpts.Message = "\nEnter the start point of the line: ";
  pPtRes = acDoc.Editor.GetPoint(pPtOpts);
  Point3d ptStart = pPtRes.Value;

  // 如果用户按 ESC 键或取消命令，就退出
  if (pPtRes.Status == PromptStatus.Cancel) return;

  // 提示终点
  pPtOpts.Message = "\nEnter the end point of the line: ";
  pPtOpts.UseBasePoint = true;
  pPtOpts.BasePoint = ptStart;
  pPtRes = acDoc.Editor.GetPoint(pPtOpts);
  Point3d ptEnd = pPtRes.Value;

  if (pPtRes.Status == PromptStatus.Cancel) return;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      BlockTable acBlkTbl;
      BlockTableRecord acBlkTblRec;
```

```
      // 以写模式打开模型空间
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建直线
      Line acLine = new Line(ptStart, ptEnd);

      // 添加直线
      acBlkTblRec.AppendEntity(acLine);
      acTrans.AddNewlyCreatedDBObject(acLine, true);

      // 缩放图形到全部显示
      acDoc.SendStringToExecute("._zoom _all ", true, false, false);

      // 提交修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub GetPointsFromUser()
    Dim startPnt As Variant
    Dim endPnt As Variant
    Dim prompt1 As String
    Dim prompt2 As String

    prompt1 = vbCrLf & "Enter the start point of the line: "
    prompt2 = vbCrLf & "Enter the end point of the line: "

    ' 获取第 1 个点
    startPnt = ThisDrawing.Utility.GetPoint(, prompt1)

    ' 用上一个点做基点
    endPnt = ThisDrawing.Utility.GetPoint(startPnt, prompt2)

    ' 用输入的两个点建立一条直线
    ThisDrawing.ModelSpace.AddLine startPnt, endPnt
    ThisDrawing.Application.ZoomAll
End Sub
```

2.8.3 GetKeywords()方法

GetKeywords()方法提示用户在 Command 提示光标处输入一个关键字。
PromptKeywordOptions 对象用来控制用户的输入及提示信息的显示方式。
PromptKeywordOptions 对象的 Keywords 属性用来定义可以在 Command 提示光标处键入的关键字。

从 AutoCAD 命令行获取用户输入的关键字

下例将 PromptKeywordOptions 对象的 AllowNone 属性设置为 false（不允许直接回车），这样使用户必须输入一个关键字。Keywords 属性用于添加允许的有效关键字。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetKeywordFromUser")> _
Public Sub GetKeywordFromUser()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
  pKeyOpts.Message = vbLf & "Enter an option "
  pKeyOpts.Keywords.Add("Line")
  pKeyOpts.Keywords.Add("Circle")
  pKeyOpts.Keywords.Add("Arc")
  pKeyOpts.AllowNone = False

  Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

  Application.ShowAlertDialog("Entered keyword: " & _
                              pKeyRes.StringResult)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetKeywordFromUser")]
public static void GetKeywordFromUser()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
  PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
  pKeyOpts.Message = "\nEnter an option ";
  pKeyOpts.Keywords.Add("Line");
  pKeyOpts.Keywords.Add("Circle");
  pKeyOpts.Keywords.Add("Arc");
  pKeyOpts.AllowNone = false;

  PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);

  Application.ShowAlertDialog("Entered keyword: " +
                              pKeyRes.StringResult);
}
```

VBA/ActiveX 代码参考

```
Sub GetKeywordFromUser()
    Dim keyWord As String
    ThisDrawing.Utility.InitializeUserInput 1, "Line Circle Arc"
    keyWord = ThisDrawing.Utility.GetKeyword _
              (vbCrLf & "Enter an option [Line/Circle/Arc]: ")
    MsgBox keyWord, , "GetKeyword Example"
End Sub
```

一个更加用户友好的关键字提示方式是,如果用户按了 Enter 键（没有输入），程序提供一个默认值。注意下例的这个小小改动。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetKeywordFromUser2")> _
Public Sub GetKeywordFromUser2()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
  pKeyOpts.Message = vbLf & "Enter an option "
  pKeyOpts.Keywords.Add("Line")
  pKeyOpts.Keywords.Add("Circle")
  pKeyOpts.Keywords.Add("Arc")
  pKeyOpts.Keywords.Default = "Arc"
  pKeyOpts.AllowNone = True

  Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

  Application.ShowAlertDialog("Entered keyword: " & _
```

```
                                pKeyRes.StringResult)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetKeywordFromUser2")]
public static void GetKeywordFromUser2()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
  pKeyOpts.Message = "\nEnter an option ";
  pKeyOpts.Keywords.Add("Line");
  pKeyOpts.Keywords.Add("Circle");
  pKeyOpts.Keywords.Add("Arc");
  pKeyOpts.Keywords.Default = "Arc";
  pKeyOpts.AllowNone = true;

  PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);

  Application.ShowAlertDialog("Entered keyword: " +
                              pKeyRes.StringResult);
}
```

VBA/ActiveX 代码参考

```
Sub GetKeywordFromUser2()
    Dim keyWord As String
    ThisDrawing.Utility.InitializeUserInput 0, "Line Circle Arc"
    keyWord = ThisDrawing.Utility.GetKeyword _
              (vbCrLf & "Enter an option [Line/Circle/Arc] <Arc>: ")
    If keyWord = "" Then keyWord = "Arc"
    MsgBox keyWord, , "GetKeyword Example"
End Sub
```

2.8.4 控制用户输入

当收集用户输入时，我们要确保能够限制用户输入的信息的类型，这样我们就可以得到我们想要的结果。使用各种不同的提示选项对象，不仅可以定义 Command 提示上显示的提示信息，而且可以限制用户提供的输入。有些输入方法，不仅可以返回方法所要求类型的值，还可以返回关键字。

例如，我们可以使用 GetPoint()方法让用户指定一个点，或用一个关键字回应，就像 LINE、CIRCLE 及 PLINE 这样的命令那样。

获取一个整数或一个关键字

下例提示用户输入一个非零的正整数或一个关键字。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("GetIntegerOrKeywordFromUser")> _
Public Sub GetIntegerOrKeywordFromUser()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim pIntOpts As PromptIntegerOptions = New PromptIntegerOptions("")
  pIntOpts.Message = vbCrLf & "Enter the size or "

  '' 限制输入必须大于 0
  pIntOpts.AllowZero = False
  pIntOpts.AllowNegative = False

  '' 定义合法关键字并允许直接按 Enter 键
  pIntOpts.Keywords.Add("Big")
  pIntOpts.Keywords.Add("Small")
  pIntOpts.Keywords.Add("Regular")
  pIntOpts.Keywords.Default = "Regular"
  pIntOpts.AllowNone = True

  '' 获取用户键入的值
  Dim pIntRes As PromptIntegerResult = acDoc.Editor.GetInteger(pIntOpts)

  If pIntRes.Status = PromptStatus.Keyword Then
      Application.ShowAlertDialog("Entered keyword: " & _
                                  pIntRes.StringResult)
  Else
      Application.ShowAlertDialog("Entered value: " & _
                                  pIntRes.Value.ToString())
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("GetIntegerOrKeywordFromUser")]
public static void GetIntegerOrKeywordFromUser()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  PromptIntegerOptions pIntOpts = new PromptIntegerOptions("");
  pIntOpts.Message = "\nEnter the size or ";

  // 限制输入必须大于 0
  pIntOpts.AllowZero = false;
  pIntOpts.AllowNegative = false;

  // 定义合法关键字并允许直接按 Enter 键
  pIntOpts.Keywords.Add("Big");
  pIntOpts.Keywords.Add("Small");
  pIntOpts.Keywords.Add("Regular");
  pIntOpts.Keywords.Default = "Regular";
  pIntOpts.AllowNone = true;

  // 获取用户键入的值
  PromptIntegerResult pIntRes = acDoc.Editor.GetInteger(pIntOpts);

  if (pIntRes.Status == PromptStatus.Keyword)
  {
      Application.ShowAlertDialog("Entered keyword: " +
                                  pIntRes.StringResult);
  }
  else
  {
      Application.ShowAlertDialog("Entered value: " +
                                  pIntRes.Value.ToString());
  }
}
```

VBA/ActiveX 代码参考

```
Sub GetIntegerOrKeywordFromUser()

    ' InitializeUserInput 方法的第 1 个参数 (6)用来限制输入必须是正数,
    ' 第 2 个参数是一个有效关键字列表
    ThisDrawing.Utility.InitializeUserInput 6, "Big Small Regular"
```

```
    ’ 设置提示字符串变量
    Dim promptStr As String
    promptStr = vbCrLf & "Enter the size or [Big/Small/Regular] <Regular>:"

    ’ 在 GetInteger 提示符下，输入关键字或
    ’ 不输入值直接按“ENTER”键，会导致错误。
    ’ 为了让应用程序继续并检查错误描述，您必须
    ’ 设置错误处理程序来恢复错误。
    On Error Resume Next

    ’ 获取用户键入的值
    Dim returnInteger As Integer
    returnInteger = ThisDrawing.Utility.GetInteger(promptStr)

    ’ 检查错误。如果错误号与下面那个匹配，就用 GetInput 获取返回的字符串，
    ’ 不匹配的话，就是要 returnInteger 的值。
    If Err.Number = -2145320928 Then
        Dim returnString As String
        Debug.Print Err.Description
        returnString = ThisDrawing.Utility.GetInput()
        If returnString = "" Then        ’ 直接按回车了
           returnString = "Regular"     ’ 默认
        End If
        Err.Clear
    Else ’ 否则
        returnString = returnInteger      ’使用键入的值
    End If

    ’ 显示结果
    MsgBox returnString, , "InitializeUserInput Example"
End Sub
```

2.9 访问 AutoCAD 命令行

我们可以使用 SendStringToExecute()方法直接向 AutoCAD 命令行发送命令。SendStringToExecute()方法将一个字符串发送给命令行。该字符串必须包含按照所执行的命令的一系列提示所期望的顺序排列的参数。

字符串中的空格，或代表回车符的 ASCII 码，等同于按下了键盘上的 Enter 键。和 AutoLISP 环境不同，调用不带参数的 SendStringToExecute()方法是无效的。

使用 SendStringToExecute()方法执行命令是异步的，直到.NET 命令结束，所调用的 AutoCAD 命令才被执行。如果需要立即执行命令（同步），我们应该：

- 使用 COM Automation 库提供的 SendCommand()方法。放心，.NET COM 互操作程序集可以访问 COM Automation 库；
- 对于 AutoCAD 本地命令，以及由 ObjectARX 或.NET API 定义的命令，调用(P/Invoke)非托管的 acedCommand()方法或 acedCmd()方法；
- 对于由 AutoLISP 定义的命令，调用（P/Invoke）非托管的 acedInvoke()方法；

发送一个命令到 AutoCAD 命令行

下面的例子用圆心（2，2，0）和半径 4 创建一个圆，然后将图形缩放到全部图形都可见。注意字符串结尾有一个空格，代表最后按 Enter 键开始执行命令。

VB.NET

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime

<CommandMethod("SendACommandToAutoCAD")> _
Public Sub SendACommandToAutoCAD()
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' 画圆并缩放到图形界限
  acDoc.SendStringToExecute("._circle 2,2,0 4 ", True, False, False)
  acDoc.SendStringToExecute("._zoom _all ", True, False, False)
End Sub
```

C#

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("SendACommandToAutoCAD")]
public static void SendACommandToAutoCAD()
{
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // 画圆并缩放到图形界限
  acDoc.SendStringToExecute("._circle 2,2,0 4 ", true, false, false);
  acDoc.SendStringToExecute("._zoom _all ", true, false, false);
}
```

VBA/ActiveX 代码参考

```
Sub SendACommandToAutoCAD()
   ’ 画圆并缩放到图形界限
   ThisDrawing.SendCommand "_Circle 2,2,0 4 "
   ThisDrawing.SendCommand "_zoom a "
End Sub
```

第 3 章 创建和编辑 AutoCAD 实体

我们可以创建各种 AutoCAD 实体对象，从简单的直线和圆到样条曲线、椭圆和关联填充区域等。通常使用 AppendEntity()函数将对象添加到 BlockTableRecord 对象。创建了一个对象后，我们就可以修改其图层、颜色、线型等属性。

图形数据库与其他数据库程序类似，我们可以将模型空间的直线对象想像成表记录，而将模型空间想像成数据库表。使用数据库时，必须在进行操作前打开和关闭记录。存储在图形数据库里的对象也是一样。我们使用 GetObject()函数从数据库中获取一个对象，并确定怎样使用该对象。

本章主要内容：

- 打开和关闭对象
- 创建对象
- 使用选择集
- 编辑命名对象和 2D 对象
- 使用图层、颜色和线型
- 保存和恢复图层状态
- 向图形添加文字

3.1 打开和关闭对象

不论是使用直线、圆和多段线这样的对象还是使用符号表及符号表记录这样的对象，我们都需要以读或写模式打开这些对象。当查询一个对象时要以读模式打开对象；如果要对一个对象进行修改，就要以写模式打开对象。

3.1.1 使用 ObjectId

图形数据库 Database 对象中包含的每个对象都被赋予了几种独特的 ID 标志，以方便访问对象。这些独特的标志有：

- 实体句柄 Handle；
- 对象 ObjectId；
- 实例指针 Instance pointer；

最常用方法是通过 ObjectId 访问对象。在一个开发项目里同时使用 COM 互操作和托管.NET API 的情况下，对象 ObjectId 都能很好地工作。如果创建自定义 AutoLISP 函数，就需要使用实体句柄来访问对象。

句柄存在于 AutoCAD 会话之间，因此，如果我们需要将图形信息导出到一个外部文件，以便随后用于更新图形，这时使用实体句柄访问对象是最好的方式。数据库中对象的 ObjectId 只在数据库加载到内存里时才存在。一旦数据库关闭，赋予对象的 ObjectId 就不存在了，并且下次打开数据库时可能为该对象赋予了一个不同的 ObjectId。

获取对象的 ObjectId

使用对象时，在打开对象对其进行查询或编辑前，需要先获得该对象的 ObjectId。图形文件打开时，数据库中已存在的对象会被赋予一个 ObjectId，新对象在第一次创建时会被赋予 ObjectId。数据库中已存在对象的 ObjectId 通常通过下列方式获得：

- 使用 Database 对象的成员属性，如 Clayer 属性用来提取当前图层的 ObjectId；
- 遍历符号表，如遍历图层符号表可以获得每个图层的 ObjectId；

打开一个对象

一旦得到对象的 ObjectId，就可以使用 GetObject()函数打开该对象。打开对象的方式（打开模式）有下列三种：

- **Read.** 以读模式打开对象；
- **Write.** 对还没打开的对象以写模式打开；
- **Notify.** 以通知方式打开对象，用于当对象已经关闭、已经以读模式打开或已经以写模式打开时。更多关于通知的内容，见（§7 *使用事件*）。

应以最适合访问的方式打开对象。以写模式打开对象可能会产生比你的需要还多的额外开销，原因是这时会创建撤销记录。如果不能确定正要打开的对象就是你要使用的，应该以读模式打开，然后使用 UpgrateOpen()方法将打开模式由读模式升级为写模式。关于使用 UpgrateOpen()方法的更多内容，参见（§3.1.4 *升级打开对象与降级打开对象*）。

GetObject()函数和 Open()函数都返回一个对象。使用某些编程语言时，我们可能需要对变量的返回值进行强制类型转换。如果使用 VB.NET，我们就不必考虑强制转换返回值的类型的问题，因为 VB.NET 已经为我们做了这件事。下面的例子演示如何获取当前数据库中图层 LayerZero 的 LayerTableRecord 记录：

VB.NET

下面的示例代码演示不再需要事务时手工将其关闭（释放内存）。

```
Dim acCurDb As Document = Application.DocumentManager.MdiActiveDocument.Database
Dim acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

Dim acLyrTblRec As LayerTableRecord
acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero, OpenMode.ForRead)

acTrans.Dispose()
```

下面的示例代码演示不再需要事务时，使用 Using 语句关闭事务（释放内存）。Using 语句是优先选用的编码方式。

```
Dim acCurDb As Document = Application.DocumentManager.MdiActiveDocument.Database
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
    Dim acLyrTblRec As LayerTableRecord
    acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero, OpenMode.ForRead)
End Using
```

C#

下面的示例代码演示不再需要事务时手工将其关闭（释放内存）。

```
Document acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
Transaction acTrans = acCurDb.TransactionManager.StartTransaction();

LayerTableRecord acLyrTblRec;
acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero,
                                OpenMode.ForRead) as LayerTableRecord;

acTrans.Dispose();
```

下面的示例代码演示不再需要事务时使用 Using 语句关闭事务（释放内存）。Using 语句是优先选用的编码方式。

```
Document acCurDb = Application.DocumentManager.MdiActiveDocument.Database;
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    LayerTableRecord acLyrTblRec;
    acLyrTblRec = acTrans.GetObject(acCurDb.LayerZero,
                                    OpenMode.ForRead) as LayerTableRecord;
}
```

3.1.2 使用事务管理器管理事务

事务用来将对多个对象的多个操作打包成单个操作组，以便于提交或回滚。事务通过事务管理器启动和管理。启动事务之后，我们就可以用 GetObject()方法打开对象。

当我们操作用 GetObject 函数打开的对象时，事务管理器 TransactionManager 会跟踪对该对象的修改。创建及添加到数据库的任何新对象也应同时调用 AddNewlyCreatedDBObject()方法将其添加到事务。对象编辑完并添加到数据库后，可以使用事务对象的 Commit()方法保存对数据库的修改并关闭所有打开的对象。事务处理完毕后，调用 Dispose()方法关闭事务。

3.1.2.1 启动新事务并打开对象

事务管理器从当前数据库的 TransactionManager 属性访问。一旦建立对事务管理器的引用，我们就可以使用 StartTransaction()方法启动一个新事务。StartTransaction()方法创建一个 Transaction 对象实例，并允许我们使用 GetObject()方法打开对象。

在事务期间打开的所有对象在事务结束时都会被关闭。要结束事务，需调用事务对象的 Dispose()方法。如果使用了.net 语言中的 Using 和 End Using 关键字来表示事务的开始和结束，就不需要调用 Dispose()方法。

在关闭事务之前，应使用 Commit()方法提交所作的任何修改。在关闭事务前，如果没有提交修改，那么可以将任何修改回滚到事务启动之前的状态。更多内容见（§3.1.2.2 ***提交修改与回滚修改***）一节。

可以启动不止一个事务。活动事务的个数可以用 TransactionManager 对象的 NumberOfActiveTransactions 属性获得，最顶层的事务或者最近的事务，可以用 TopTransaction 属性获得。

在程序执行过程中，为了回滚所作的部分修改，可以将一个事务嵌套在另一个事务中。更多关于使用多个事务及嵌套事务的内容，参见（§3.1.2.3 ***嵌套事务***）。

查询对象

下面的示例演示如何使用事务打开并读取对象。该示例中，使用 GetObject()方法先打开了块表 BlockTable，然后打开了块表中的一条记录模型空间 ModelSpace。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("OpenTransactionManager")> _
Public Sub OpenTransactionManager()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以读模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForRead)
```

```
      '' 遍历 Block 表记录
      For Each acObjId As ObjectId In acBlkTblRec
          acDoc.Editor.WriteMessage(vbLf & "DXF name: " &
acObjId.ObjectClass().DxfName)
          acDoc.Editor.WriteMessage(vbLf & "ObjectID: " & acObjId.ToString())
          acDoc.Editor.WriteMessage(vbLf & "Handle: " &
acObjId.Handle.ToString())
          acDoc.Editor.WriteMessage(vbLf)
      Next

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("OpenTransactionManager")]
public static void OpenTransactionManager()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以读模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForRead) as BlockTableRecord;

      // 遍历块表记录
      foreach (ObjectId asObjId in acBlkTblRec)
      {
          acDoc.Editor.WriteMessage("\nDXF name: " +
asObjId.ObjectClass.DxfName);
```

```
            acDoc.Editor.WriteMessage("\nObjectID: " + asObjId.ToString());
            acDoc.Editor.WriteMessage("\nHandle: " + asObjId.Handle.ToString());
            acDoc.Editor.WriteMessage("\n");
        }

        // 关闭事务
    }
}
```

向数据库添加新对象

下例演示使用事务向数据库添加一个圆。用 GetObject()方法先以读模式打开 BlockTable 块表，然后以写模式打开模型空间记录。以写模式打开模型空间后，就可以使用 AppendEntity()方法和 AddNewlyCreatedDBObject()方法向模型空间添加一个新的圆对象并同时将操作添加到事务。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddNewCircleTransaction")> _
Public Sub AddNewCircleTransaction()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 以半径 3 圆心(5,5)画圆
      Dim acCirc As Circle = New Circle()
      acCirc.Center = New Point3d(5, 5, 0)
      acCirc.Radius = 3
```

```
        '' 将新对象添加到 Model 空间并进行事务登记
        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' 提交修改并关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddNewCircleTransaction")]
public static void AddNewCircleTransaction()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 以半径 3 圆心 5,5 画圆
      Circle acCirc = new Circle();
      acCirc.Center = new Point3d(5, 5, 0);
      acCirc.Radius = 3;

      // 将新对象添加到 Model 空间并进行事务登记
      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);
```

```
        // 提交修改并关闭事务
        acTrans.Commit();
    }
}
```

3.1.2.2 提交修改与回滚修改

使用事务时，我们能够决定什么时候将对对象所作的修改保存到图形数据库去。我们使用事务的 Commit()方法保存对打开的对象所作的修改。如果程序出错，我们可以使用 Abort()方法回滚事务中所作的修改。

如果调用 Dispose()函数前没有调用 Commit()函数，将会回滚事务期间所作的全部修改。不论调用了 Commit()方法抑或 Abort()方法，我们都要调用 Dispose()方法发出结束事务的信号。如果启动事务对象使用了 Using 语句，就不必调用 Dispose()方法了。

VB.NET

```
''提交事务内所作修改
<transaction>.Commit()

''终止事务并回滚到之前的状态
<transaction>.Abort()
```

C#

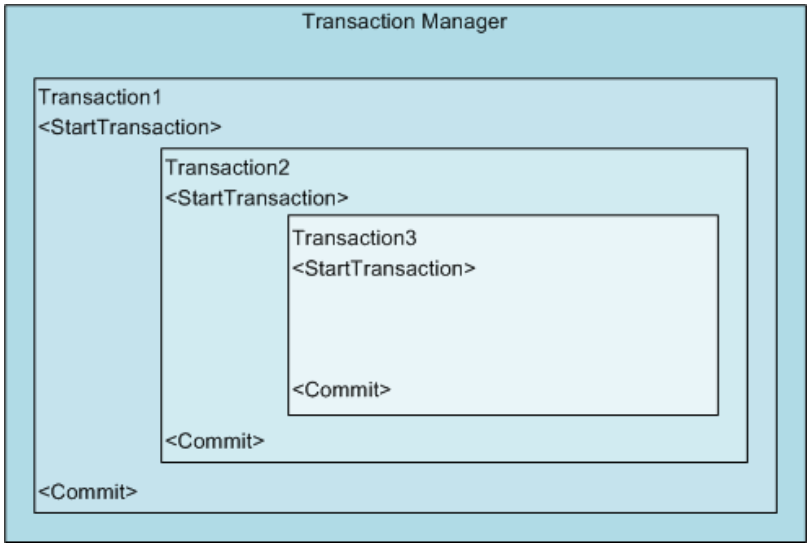
```
//提交事务内所作修改
<transaction>.Commit();

//终止事务并回滚到之前的状态
<transaction>.Abort();
```

3.1.2.3 嵌套事务

一个事务可以嵌套在另一个事务里。我们可能用外层事务撤销程序所作的所有修改，而用内层事务只撤销部分修改。使用嵌套事务时，我们启动一个顶层事务，也称之为最外层事务。

启动的新事务会被添加进前一个事务里。必须按与创建相反的顺序提交或终止嵌套事务。因此，如果有三个事务，我们必须先关闭第三个或最内层那个，然后是第二个，最后是第一个。如果终止第一个事务，那么三个事务所作的所有修改就都被撤销了。

下图显示了事务嵌套的情形。



使用嵌套事务创建及修改对象

下面这个例子演示使用三个事务创建一个圆和一条直线，然后改变它们的颜色。圆的颜色在第二个和第三个事务中修改，不过，由于终止了第三个事务，因此只有第一个和第二个事务中所作的修改被保存到数据库了。另外，创建和关闭事务时，在命令行窗口显示了活动事务的个数。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("NestedTransactions")> _
Public Sub NestedTransactions()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 引用 TransactionManager
  Dim acTransMgr As Autodesk.AutoCAD.DatabaseServices.TransactionManager
  acTransMgr = acCurDb.TransactionManager

  '' 新建一个事务
  Using acTrans1 As Transaction = acTransMgr.StartTransaction()

      '' 打印当前活动事务个数
      acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
```

```
                    acTransMgr.NumberOfActiveTransactions.ToString())

        '' 以读模式打开 Block 表
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans1.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

        '' 以写模式打开 Block 表记录 Model 空间
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans1.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                         OpenMode.ForWrite)

        '' 创建圆
        Dim acCirc As Circle = New Circle()
        acCirc.Center = New Point3d(5, 5, 0)
        acCirc.Radius = 3

        '' 将新对象添加到 Model 空间并进行事务登记
        acBlkTblRec.AppendEntity(acCirc)
        acTrans1.AddNewlyCreatedDBObject(acCirc, True)

        '' 创建第 2 个事务
        Using acTrans2 As Transaction = acTransMgr.StartTransaction()

            acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
                  acTransMgr.NumberOfActiveTransactions.ToString())

            '' 修改圆的颜色
            acCirc.ColorIndex = 5

            ''新建一条直线
            Dim acLine As Line = New Line(New Point3d(2, 5, 0), New Point3d(10, 7,
0))
            acLine.ColorIndex = 3

            '' 将新对象添加到 Model 空间并进行事务登记
            acBlkTblRec.AppendEntity(acLine)
            acTrans2.AddNewlyCreatedDBObject(acLine, True)

            ''创建第 3 个事务
            Using acTrans3 As Transaction = acTransMgr.StartTransaction()

                acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: "
& _
                acTransMgr.NumberOfActiveTransactions.ToString())
```

```
            '' 修改圆的颜色
            acCirc.ColorIndex = 3

            '' 更新图形显示
            acDoc.Editor.WriteMessage(vbLf)
            acDoc.Editor.Regen()

            '' 询问保留还是取消第三个事务中的修改
            Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
            pKeyOpts.Message = vbLf & "Keep color change "
            pKeyOpts.Keywords.Add("Yes")
            pKeyOpts.Keywords.Add("No")
            pKeyOpts.Keywords.Default = "No"
            pKeyOpts.AllowNone = True

            Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

            If pKeyRes.StringResult = "No" Then
                '' 取消事务 3 中的修改
                acTrans3.Abort()
            Else
                '' 保存事务 3 中的修改
                acTrans3.Commit()
            End If

            '' 关闭事务 3
        End Using

        acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
              acTransMgr.NumberOfActiveTransactions.ToString())

        '' 提交事务 2 中的修改
        acTrans2.Commit()
    End Using

    acDoc.Editor.WriteMessage(vbLf & "Number of transactions active: " & _
                  acTransMgr.NumberOfActiveTransactions.ToString())

    ''提交事务 1 中的修改
    acTrans1.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("NestedTransactions")]
public static void NestedTransactions()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 创建对事务管理器的引用
  Autodesk.AutoCAD.DatabaseServices.TransactionManager acTransMgr;
  acTransMgr = acCurDb.TransactionManager;

  // 新建事务
  using (Transaction acTrans1 = acTransMgr.StartTransaction())
  {
      // 打印当前活动事务的个数
      acDoc.Editor.WriteMessage("\nNumber of transactions active: " +

acTransMgr.NumberOfActiveTransactions.ToString());

      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans1.GetObject(acCurDb.BlockTableId,
                                    OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans1.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

      // 创建半径 2 圆心 5,5 的圆
      Circle acCirc = new Circle();
      acCirc.Center = new Point3d(5, 5, 0);
      acCirc.Radius = 3;

      // 添加新对象到模型空间并添加事务
      acBlkTblRec.AppendEntity(acCirc);
      acTrans1.AddNewlyCreatedDBObject(acCirc, true);
```

```
//创建第 2 个事务
using (Transaction acTrans2 = acTransMgr.StartTransaction())
{
    acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
          acTransMgr.NumberOfActiveTransactions.ToString());

    // 修改圆的颜色
    acCirc.ColorIndex = 5;

    // 新建一条直线
    Line acLine = new Line(new Point3d(2, 5, 0), new Point3d(10, 7, 0));
    acLine.ColorIndex = 3;

    // 将直线对象添加到 Model 空间并进行事务登记（事务 2）
    acBlkTblRec.AppendEntity(acLine);
    acTrans2.AddNewlyCreatedDBObject(acLine, true);

    //创建第 3 个事务
    using (Transaction acTrans3 = acTransMgr.StartTransaction())
    {
        acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                acTransMgr.NumberOfActiveTransactions.ToString());

        //  修改圆的颜色
        acCirc.ColorIndex = 3;

        //  更新图形显示
        acDoc.Editor.WriteMessage("\n");
        acDoc.Editor.Regen();

        // 询问保留还是取消第 3 个事务中的修改
        PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
        pKeyOpts.Message = "\nKeep color change ";
        pKeyOpts.Keywords.Add("Yes");
        pKeyOpts.Keywords.Add("No");
        pKeyOpts.Keywords.Default = "No";
        pKeyOpts.AllowNone = true;

        PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);

        if (pKeyRes.StringResult == "No")
        {
            //取消事务 3 中的修改
            acTrans3.Abort();
```

```
                }
                else
                {
                    //保存事务 3 中的修改
                    acTrans3.Commit();
                }
                //关闭事务 3
            }

            acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                      acTransMgr.NumberOfActiveTransactions.ToString());

            // 保留事务 2 中的修改
            acTrans2.Commit();
        }

        acDoc.Editor.WriteMessage("\nNumber of transactions active: " +
                    acTransMgr.NumberOfActiveTransactions.ToString());

        //保留事务 1 中的修改
        acTrans1.Commit();
    }
}
```

3.1.3 不使用事务管理器打开和关闭对象

使用事务使得打开和操作多个对象变得容易多了，不过这并不是打开和编辑对象的唯一方式。除了使用事务外，我们还可以使用 Open()方法和 Close()方法打开和关闭对象。使用 Open()方法仍然需要先获取对象的 ObjectId，就像使用事务时的 GetObject()方法一样，我们需要指定打开模式并返回一个对象。使用 Open()方法打开对象后如果对这个对象进行了修改，我们可以使用 Cancel()方法回滚对象打开以来的所有修改。当要回滚时，必须对每个对象都调用 Cancel()方法。

注：打开对象和关闭对象的操作必须成对出现。如果我们对一个对象使用了 Open()方法，就必须使用 Close()方法或 Cancel()方法关闭它。不关闭对象会导致读取访问冲突，并会导致 AutoCAD 不稳定。

如果我们只使用一个对象，和使用事务管理器相比，使用 Open()方法及 Close()方法能减少编写代码的行数。不过，还是推荐使用事务来打开和关闭对象。

警告：使用事务时不能使用 Open()和 Close()方法，因为不仅不能正常打开和关闭对象，还会导致 AutoCAD 崩溃。

查询对象

下面的示例演示不使用事务及 GetObject()方法怎样打开和关闭对象。可以和上一节使用事务管理器的例子比较一下。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("OpenCloseObjectId")> _
Public Sub OpenCloseObjectId()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 以读模式打开 Block 表
  Dim acBlkTbl As BlockTable
  acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead)

  ''以读模式打开块表记录模型空间
  Dim acBlkTblRec As BlockTableRecord
  acBlkTblRec = acBlkTbl(BlockTableRecord.ModelSpace).Open(OpenMode.ForRead)

  '' 遍历块表记录
  For Each acObjId As ObjectId In acBlkTblRec
      acDoc.Editor.WriteMessage(vbLf & "DXF name: " &
acObjId.ObjectClass().DxfName)
      acDoc.Editor.WriteMessage(vbLf & "ObjectID: " & acObjId.ToString())
      acDoc.Editor.WriteMessage(vbLf & "Handle: " & acObjId.Handle.ToString())
      acDoc.Editor.WriteMessage(vbLf)
  Next

  '' 关闭块表记录
  acBlkTblRec.Close()
  acBlkTblRec.Dispose()

  '' 关闭块表
  acBlkTbl.Close()
  acBlkTbl.Dispose()
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```
[CommandMethod("OpenCloseObjectId")]
public static void OpenCloseObjectId()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 以读模式打开 Block 表
  BlockTable acBlkTbl;
  acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead) as BlockTable;

  // 以读模式打开模型空间记录
  BlockTableRecord acBlkTblRec;
  acBlkTblRec = acBlkTbl[BlockTableRecord.ModelSpace].Open(OpenMode.ForRead) as
BlockTableRecord;

  // 遍历块表记录
  foreach (ObjectId acObjId in acBlkTblRec)
  {
      acDoc.Editor.WriteMessage("\nDXF name: " + acObjId.ObjectClass.DxfName);
      acDoc.Editor.WriteMessage("\nObjectID: " + acObjId.ToString());
      acDoc.Editor.WriteMessage("\nHandle: " + acObjId.Handle.ToString());
      acDoc.Editor.WriteMessage("\n");
  }

  // 关闭块表记录
  acBlkTblRec.Close();
  acBlkTblRec.Dispose();

  // 关闭块表
  acBlkTbl.Close();
  acBlkTbl.Dispose();
}
```

向数据库添加新对象

本例演示不使用事务管理器怎样创建新对象并添加到模型空间。可以和上一节使用事务管理器的例子比较一下。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
```

```
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddNewCircleOpenClose")> _
Public Sub AddNewCircleOpenClose()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 以读模式打开 Block 表
  Dim acBlkTbl As BlockTable
  acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead)

  '' 以写模式打开 Block 表记录 Model 空间
  Dim acBlkTblRec As BlockTableRecord
  acBlkTblRec = acBlkTbl(BlockTableRecord.ModelSpace).Open(OpenMode.ForWrite)

  '' 创建园，半径 3、圆心 5,5
  Dim acCirc As Circle = New Circle()
  acCirc.Center = New Point3d(5, 5, 0)
  acCirc.Radius = 3

  '' 将新对象添加到 Model 空间
  acBlkTblRec.AppendEntity(acCirc)

  '' 关闭圆
  acCirc.Close()
  acCirc.Dispose()

  '' 关闭块表记录
  acBlkTblRec.Close()
  acBlkTblRec.Dispose()

  '' 关闭块表
  acBlkTbl.Close()
  acBlkTbl.Dispose()
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
```

```
[CommandMethod("AddNewCircleOpenClose")]
public static void AddNewCircleOpenClose()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 以读模式打开 Block 表
  BlockTable acBlkTbl;
  acBlkTbl = acCurDb.BlockTableId.Open(OpenMode.ForRead) as BlockTable;

  // 以写模式打开 Block 表记录 Model 空间
  BlockTableRecord acBlkTblRec;
  acBlkTblRec = acBlkTbl[BlockTableRecord.ModelSpace].Open(OpenMode.ForWrite)
                  as BlockTableRecord;
  //创建园，半径 3、圆心 5,5
  Circle acCirc = new Circle();
  acCirc.Center = new Point3d(5, 5, 0);
  acCirc.Radius = 3;

  // 将新对象添加到 Model 空间
  acBlkTblRec.AppendEntity(acCirc);
  // 关闭圆
  acCirc.Close();
  acCirc.Dispose();

  // 关闭块表记录
  acBlkTblRec.Close();
  acBlkTblRec.Dispose();

  // 关闭块表
  acBlkTbl.Close();
  acBlkTbl.Dispose();
}
```

译者注：编译本小节的示例时提示如下警告信息：

```
警告 1 “Autodesk.AutoCAD.DatabaseServices.ObjectId.Open(Autodesk.AutoCAD.DatabaseServices.OpenMode)”已过时:
“"For advanced use only. Use GetObject instead""
警告 2 “Autodesk.AutoCAD.DatabaseServices.DBObject.Close()”已过时:“"Use Transaction instead""
```

因此还是推荐使用**事务**及 GetObject()方法。

3.1.4 升级打开对象与降级打开对象

使用 GetObject()方法或 Open()方法打开一个对象后，我们可以使用 UpgradeOpen()方法与 DowngradeOpen()方法改变对象当前的打开模式。UpgradeOpen()方法将对象的打开模式由读升级为写，DowngradeOpen()方法将对象的打开模式由写降级为读。不必为每个 UpgradeOpen()调用都配上一个 DowngradeOpen()调用，因为对象或事务的关闭能将实体的打开状态充分地清理干净。

试图打开一个对象时，想以什么模式使用对象就以什么模式打开。当只是查询一个对象时，就不要以写模式打开。以读模式打开对象并查询其属性比以写模式打开对象并查询效率高得多。

以写模式打开对象会启动该对象的撤销登记。撤销登记用来跟踪对象的修改，这样，所做的任何修改都能被回滚。如果不能确定是否需要修改对象，最好以读模式打开对象，然后在需要时再升级为写模式。这样做能帮助减少程序开销。

何时使用 UpgradeOpen()方法的一个例子是，当想查看对象是否满足某个指定条件时，就先以读模式打开对象，如果满足条件就将对象的打开模式由读升级为写模式，然后进行修改操作。

以通知方式打开 Open Notifications

类似的情形，如果是以通知模式打开的对象，如果接受到一个通知，可以使用 UpgradeFromNotify()方法将对象的打开模式升级为写模式，然后还可以用 DowngradeFromNotify()方法将对象的打开模式降级为通知模式。

UpgradeFromNotify()方法和 DowngradeFromNotify()方法是为试图修改对象自己的打开状态以便安全地修改自己的那些方法而保留的。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("FreezeDoorLayer")> _
Public Sub FreezeDoorLayer()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, OpenMode.ForRead)
```

```
    '' 遍历图层，将图层名以‘Door’开头的图层升级为写打开
    For Each acObjId As ObjectId In acLyrTbl
        '' 以读模式打开图层表记录
        Dim acLyrTblRec As LayerTableRecord
        acLyrTblRec = acTrans.GetObject(acObjId, OpenMode.ForRead)

        '' 检查图层名是否以‘Door’开头
        If (acLyrTblRec.Name.StartsWith("Door", _
                            StringComparison.OrdinalIgnoreCase) = True) Then
            '' 检查是否为当前图层，不冻结当前图层
            If acLyrTblRec.ObjectId <> acCurDb.Clayer Then
                '' 升级打开模式
                acLyrTblRec.UpgradeOpen()

                '' 冻结图层
                acLyrTblRec.IsFrozen = True
            End If
        End If
    Next

    '' 提交修改并关闭事务
    acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("FreezeDoorLayer")]
public static void FreezeDoorLayer()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
```

```
                                    OpenMode.ForRead) as LayerTable;

        // 遍历图层，将图层名以‘Door’开头的图层升级为写打开;
        foreach (ObjectId acObjId in acLyrTbl)
        {
            // 以读模式打开图层表记录以读打开图层表记录
            LayerTableRecord acLyrTblRec;
            acLyrTblRec = acTrans.GetObject(acObjId,
                                            OpenMode.ForRead) as LayerTableRecord;

            // 检查图层名是否以‘Door’开头
            if (acLyrTblRec.Name.StartsWith("Door",
                                StringComparison.OrdinalIgnoreCase) == true)
            {
                // 检查是否为当前层
                if (acLyrTblRec.ObjectId != acCurDb.Clayer)
                {
                    //  升级打开模式
                    acLyrTblRec.UpgradeOpen();

                    // 冻结图层
                    acLyrTblRec.IsFrozen = true;
                }
            }
        }

        // 提交修改并关闭事务
        acTrans.Commit();
    }
}
```

3.2 创建对象

对于创建相同的图形对象，AutoCAD 常常提供几个不同的方法。.NET API 虽然没有提供相同的创建方法组合，但为每个对象类型都提供了基本对象构造函数，并且对许多对象的构造函数还提供了重载。

例如，AutoCAD 中绘制一个圆有 4 个不同的方法：(1)通过指定圆心和半径；(2)通过两点定义的直径；(3)通过三个点定义的圆周；(4)通过两条切线和半径。可是，在.NETAPI 中，创建一个圆有两个方法，一个方法是无参数方法，另一个方法需要圆心、方向和半径。

注：新对象通过使用 New 关键字创建，然后使用 Add()或 AppendEntity()方法追加到其父对象中。具体是使用 Add()方法还是使用 AppendEntity()方法，取决于父对象是一个容器对象（符号表或字典）还是 BlockTableRecord 对象。

设置对象的默认属性值

一个新对象创建时，下列实体属性值被设置成当前文档数据库所定义的实体值：

- Color 颜色
- Layer 图层
- Linetype 线型
- Linetype scale 线形比例
- Lineweight 线宽
- Plot style name 打印样式名字
- Visibility 可见性
- Transparency 透明度

注：如果需要将对象的属性设置为当前数据库的默认值，可以调用要修改对象的 SetDatabaseDefaults()方法。

3.2.1 确定父对象

图形对象被添加到模型空间或图纸空间的 BlockTableRecord 对象中。我们通过 BlockTable 对象引用代表 Model 空间或 Paper 空间的块。如果要使用当前空间而不是指定某空间，可以使用 CurrentSpaceId 属性从当前数据库获取当前空间的 ObjectId。

块表记录 Model 空间和 Paper 空间的 ObjectId 可以使用属性从 BlockTable 对象取得，或者使用 DatabaseServices 命名空间下 SymbolUtilitySErvices 类的 GetBlockModelSpaceId()方法和 GetBlockPaperSpaceId()方法取得。

访问模型空间、图纸空间或当前空间

下面这个例子演示如何访问与 Model 空间、Paper 空间或当前空间关联的块表记录。获得对块表记录的引用后，添加一条新直线到块表记录。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("AccessSpace")> _
Public Sub AccessSpace()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以读模式打开块表记录
      Dim acBlkTblRec As BlockTableRecord

      '' 打开的是哪个块表记录?
      Dim pKeyOpts As PromptKeywordOptions = New PromptKeywordOptions("")
      pKeyOpts.Message = vbLf & "Enter which space to create the line in "
      pKeyOpts.Keywords.Add("Model")
      pKeyOpts.Keywords.Add("Paper")
      pKeyOpts.Keywords.Add("Current")
      pKeyOpts.AllowNone = False
      pKeyOpts.AppendKeywordsToMessage = True

      Dim pKeyRes As PromptResult = acDoc.Editor.GetKeywords(pKeyOpts)

      If pKeyRes.StringResult = "Model" Then
          '' 从块表获取 Model 空间的 ObjectID
          acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                          OpenMode.ForWrite)
      ElseIf pKeyRes.StringResult = "Paper" Then
          ''从块表获取 Paper 空间的 ObjectID
          acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.PaperSpace),
_
                                          OpenMode.ForWrite)
      Else
```

```
            ''从块表获取当前空间的 ObjectID
            acBlkTblRec = acTrans.GetObject(acCurDb.CurrentSpaceId, _
                                            OpenMode.ForWrite)
        End If

        '' 从(2,5)到(10,7)画一条直线
        Dim acLine As Line = New Line(New Point3d(2, 5, 0), _
                                      New Point3d(10, 7, 0))

        '' 添加新对象到块表记录并添加事务
        acBlkTblRec.AppendEntity(acLine)
        acTrans.AddNewlyCreatedDBObject(acLine, True)

        '' 保存新直线到数据库
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("AccessSpace")]
public static void AccessSpace()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以读模式打开块表表记录
      BlockTableRecord acBlkTblRec;

      // 打开的是哪个块表记录?
```

```
        PromptKeywordOptions pKeyOpts = new PromptKeywordOptions("");
        pKeyOpts.Message = "\nEnter which space to create the line in ";
        pKeyOpts.Keywords.Add("Model");
        pKeyOpts.Keywords.Add("Paper");
        pKeyOpts.Keywords.Add("Current");
        pKeyOpts.AllowNone = false;
        pKeyOpts.AppendKeywordsToMessage = true;

        PromptResult pKeyRes = acDoc.Editor.GetKeywords(pKeyOpts);

        if (pKeyRes.StringResult == "Model")
        {
            //从 Block 表获取 Model 空间的 ObjectID
            acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;
        }
        else if (pKeyRes.StringResult == "Paper")
        {
            //从 Block 表获取 Paper 空间的 ObjectID
            acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.PaperSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;
        }
        else
        {
            //从数据库获取当前空间的 ObjectID
            acBlkTblRec = acTrans.GetObject(acCurDb.CurrentSpaceId,
                                        OpenMode.ForWrite) as BlockTableRecord;
        }

        // 从(2,5)到(10,7)画一条直线
        Line acLine = new Line(new Point3d(2, 5, 0),
                               new Point3d(10, 7, 0));

        // 添加新对象到块表记录并添加事务
        acBlkTblRec.AppendEntity(acLine);
        acTrans.AddNewlyCreatedDBObject(acLine, true);

        //  保存新直线到数据库
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Public Sub AccessSpace()
```

```
    ' 定义合法关键字列表
    Dim keywordList As String
    keywordList = "Model Paper Current"

    ' 调用 InitializeUserInput 建立关键字列表
    ThisDrawing.Utility.InitializeUserInput 1, keywordList

    ' 获取用户输入
    Dim retVal As Variant
    retVal = ThisDrawing.Utility.GetKeyword(vbLf & _
                          "Enter which space to create the line in " & _
                          "[Model/Paper/Current]: ")

    ' 获取输入的关键字
    Dim strVal As String
    strVal = ThisDrawing.Utility.GetInput

    Dim acSpaceObj As Object

    If strVal = "Model" Or _
      (strVal = "Current" And ThisDrawing.ActiveSpace = acModelSpace) Then
        '' Get the Model space object
        Set acSpaceObj = ThisDrawing.ModelSpace
    Else
        '' Get the Paper space object
        Set acSpaceObj = ThisDrawing.PaperSpace
    End If

    '' 从(2,5)到(10,7)画一条直线
    Dim acLine As AcadLine
    Dim dPtStr(0 To 2) As Double
    dPtStr(0) = 2: dPtStr(1) = 5: dPtStr(2) = 0#

    Dim dPtEnd(0 To 2) As Double
    dPtEnd(0) = 10: dPtEnd(1) = 7: dPtEnd(2) = 0#

    Set acLine = acSpaceObj.AddLine(dPtStr, dPtEnd)
End Sub
```

3.2.2 创建线

线是 AutoCAD 中最基本的对象。我们可以创建各种不同的线——单一的直线、带圆弧或不带圆弧的复合线段。通常是通过指定坐标点来绘制线。创建的线从当前数据库继承当前的图层、线型及颜色等设置。

创建一条线，就是创建下列对象的一个新实例：

Line

创建一条直线；

Polyline

创建二维轻量级多段线；

MLine

创建多线；

Polyline2D

创建二维多段线；

Polyline3D

创建三维多段线

注：Polyline2D 对象是 Release14 之前版本 AutoCAD 中的传统多段线对象，Polyline 对象代表 AutoCAD Release 14 版新引入的、优化了的多段线。

创建一个 Line 对象

本例在模型空间添加一条直线，起点（5,5,0），终点（12,3,0）。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddLine")> _
Public Sub AddLine()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```
  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建直线
      Dim acLine As Line = New Line(New Point3d(5, 5, 0), _
                                    New Point3d(12, 3, 0))

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acLine)
      acTrans.AddNewlyCreatedDBObject(acLine, True)

      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddLine")]
public static void AddLine()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
```

```
                                    OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a line
      Line acLine = new Line(new Point3d(5, 5, 0),
                             new Point3d(12, 3, 0));

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acLine);
      acTrans.AddNewlyCreatedDBObject(acLine, true);

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddLine()
    ' 定义起点
    Dim ptStr(0 To 2) As Double
    ptStr(0) = 5: ptStr(1) = 5: ptStr(2) = 0#

    ' 定义终点
    Dim ptEnd(0 To 2) As Double
    ptEnd(0) = 12: ptEnd(1) = 3: ptEnd(2) = 0#

    ' 在模型空间建 Line 对象
    Dim lineObj As AcadLine
    Set lineObj = ThisDrawing.ModelSpace.AddLine(ptStr, ptEnd)

    ThisDrawing.Application.ZoomAll
End Sub
```

创建一个 Polyline 对象

本例往模型空间添加一条轻量级多段线，多段线有两段线段，二维坐标为（2,4）、（4,2）和（6,4）。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
```

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddLightweightPolyline")> _
Public Sub AddLightweightPolyline()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建 3 个顶点的多段线
      Dim acPoly As Polyline = New Polyline()
      acPoly.AddVertexAt(0, New Point2d(2, 4), 0, 0, 0)
      acPoly.AddVertexAt(1, New Point2d(4, 2), 0, 0, 0)
      acPoly.AddVertexAt(2, New Point2d(6, 4), 0, 0, 0)

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPoly)
      acTrans.AddNewlyCreatedDBObject(acPoly, True)

      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddLightweightPolyline")]
```

```
public static void AddLightweightPolyline()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create a polyline with two segments (3 points)
      Polyline acPoly = new Polyline();
      acPoly.AddVertexAt(0, new Point2d(2, 4), 0, 0, 0);
      acPoly.AddVertexAt(1, new Point2d(4, 2), 0, 0, 0);
      acPoly.AddVertexAt(2, new Point2d(6, 4), 0, 0, 0);

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPoly);
      acTrans.AddNewlyCreatedDBObject(acPoly, true);

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddLightWeightPolyline()
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 5) As Double

    ' 定义多段线用点
    points(0) = 2: points(1) = 4
    points(2) = 4: points(3) = 2
    points(4) = 6: points(5) = 4

    '在模型空间创建 Polyline 对象
```

```
    Set plineObj = ThisDrawing.ModelSpace. _
                   AddLightWeightPolyline(points)
    ThisDrawing.Application.ZoomAll
End Sub
```

3.2.3 创建曲线类对象

用 AutoCAD 可以创建各种不同的曲线类对象，包括样条曲线、螺旋线、圆、圆弧以及椭圆等。所有这些曲线都创建在当前用户坐标系的 XY 平面上。

创建曲线，就是新建下列对象的一个实例：

Arc

已知圆心、半径、起止角，建圆弧；

Circle

已知圆心、半径，建圆；

Ellipse

已知圆心、主轴上一点、半径比，建椭圆；

Spline

创建一条二阶或三阶不均匀有理 B 样条曲线；

Helix

创建二维或三维螺旋线对象；

创建圆对象

本例在模型空间创建一个圆，圆心为（2,3,0），半径为 4.25。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddCircle")> _
Public Sub AddCircle()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建圆，圆心(2,3)，半径 4.25
      Dim acCirc As Circle = New Circle()
      acCirc.Center = New Point3d(2, 3, 0)
      acCirc.Radius = 4.25

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddCircle")]
public static void AddCircle()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
```

```
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建圆，圆心(2,3)，半径 4.25
      Circle acCirc = new Circle();
      acCirc.Center = new Point3d(2, 3, 0);
      acCirc.Radius = 4.25;

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddCircle()
    ' 定义圆心
    Dim ptCen(0 To 2) As Double
    ptCen(0) = 2: ptCen(1) = 3: ptCen(2) = 0#

    ' 在模型空间创建 Circle 对象
    Dim circObj As AcadCircle
    Set circObj = ThisDrawing.ModelSpace.AddCircle(ptCen, 4.25)

    ThisDrawing.Application.ZoomAll
End Sub
```

创建圆弧对象

本例在模型空间创建一个圆弧，圆弧的圆心为（6.25,9.125,0），半径为 6，圆弧的起始角为 1.117（64°），终止角为 3.5605（204°）。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
```

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddArc")> _
Public Sub AddArc()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' Create an arc
      Dim acArc As Arc = New Arc(New Point3d(6.25, 9.125, 0), _
                                 6, 1.117, 3.5605)

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acArc)
      acTrans.AddNewlyCreatedDBObject(acArc, True)

      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddArc")]
public static void AddArc()
{
```

```
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 已知圆心、半径、起止角，创建圆弧
      Arc acArc = new Arc(new Point3d(6.25, 9.125, 0),
                          6, 1.117, 3.5605);

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acArc);
      acTrans.AddNewlyCreatedDBObject(acArc, true);

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddArc()
    ' 定义圆心
    Dim ptCen(0 To 2) As Double
    ptCen(0) = 6.25: ptCen(1) = 9.125: ptCen(2) = 0#

    ' 在模型空间创建 Arc 对象
    Dim arcObj As AcadArc
    Set arcObj = ThisDrawing.ModelSpace.AddArc(ptCen, 6#, 1.117, 3.5605)

    ThisDrawing.Application.ZoomAll
End Sub
```

创建样条曲线对象

本例用三个点(0, 0, 0)、 (5, 5, 0)和 (10, 0, 0)在模型空间创建一条样条曲线。该样条曲线的起止点的切线方向为(0.5, 0.5, 0.0)。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddSpline")> _
Public Sub AddSpline()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 定义样条曲线的拟合点
      Dim ptColl As Point3dCollection = New Point3dCollection()
      ptColl.Add(New Point3d(0, 0, 0))
      ptColl.Add(New Point3d(5, 5, 0))
      ptColl.Add(New Point3d(10, 0, 0))

      '' 获取点(0.5,0.5,0)的 3D 矢量
      Dim vecTan As Vector3d = New Point3d(0.5, 0.5, 0).GetAsVector

      '' 创建通过 3 个点的样条曲线，且起止点的切线方向为(0.5, 0.5, 0.0);
      '' 注：在公差值设置为 0.0 时（第 5 个参数），样条曲线直接通过拟合点。
      Dim acSpline As Spline = New Spline(ptColl, vecTan, vecTan, 4, 0.0)

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSpline)
      acTrans.AddNewlyCreatedDBObject(acSpline, True)
```

```
      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddSpline")]
public static void AddSpline()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 定义样条曲线的拟合点
      Point3dCollection ptColl = new Point3dCollection();
      ptColl.Add(new Point3d(0, 0, 0));
      ptColl.Add(new Point3d(5, 5, 0));
      ptColl.Add(new Point3d(10, 0, 0));

      // 获取点(0.5,0.5,0)的 3D 矢量
      Vector3d vecTan = new Point3d(0.5, 0.5, 0).GetAsVector();

      // 创建通过 3 个点的样条曲线，且起止点的切线方向为(0.5, 0.5, 0.0);
      // 注：在公差值设置为 0.0 时（第 5 个参数），样条曲线直接通过拟合点。
      Spline acSpline = new Spline(ptColl, vecTan, vecTan, 4, 0.0);
```

```
        // 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSpline);
        acTrans.AddNewlyCreatedDBObject(acSpline, true);

        // 保存新对象到数据库
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

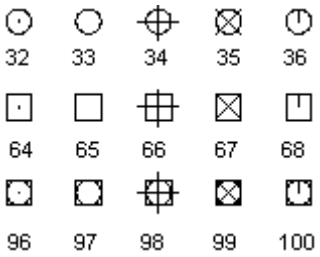
```
Sub AddSpline()
    ' 本例在模型空间创建一条样条曲线
    Dim splineObj As AcadSpline
    Dim startTan(0 To 2) As Double
    Dim endTan(0 To 2) As Double
    Dim fitPoints(0 To 8) As Double

    ' 定义变量
    startTan(0) = 0.5: startTan(1) = 0.5: startTan(2) = 0
    endTan(0) = 0.5: endTan(1) = 0.5: endTan(2) = 0
    fitPoints(0) = 1: fitPoints(1) = 1: fitPoints(2) = 0
    fitPoints(3) = 5: fitPoints(4) = 5: fitPoints(5) = 0
    fitPoints(6) = 10: fitPoints(7) = 0: fitPoints(8) = 0

    ' 创建 Spline 对象
    Set splineObj = ThisDrawing.ModelSpace.AddSpline _
                    (fitPoints, startTan, endTan)
    ZoomAll
End Sub
```

3.2.4 创建点对象

点对象 Point 有时会很有用，比如用作捕捉的节点或偏移对象的参考点。我们可以设置点的样式以及相对屏幕的大小或以绝对单位表示的大小。

Database 对象的 Pdmode 属性和 Pdsize 属性用来控制 Point 对象的外观样式。Pdmode 取值为 0、2、3 和 4 指定点画的外观，取值为 1 表示什么都不显示。如下图：

上述 Pdmode 值分别加上 32、64、96 表示分别在上述点的外形周围加画上不同的形状，如下图：



Pdsize 控制点形状的大小（Pdmode 取值为 0 和 1 时除外）。Pdsize 为 0 时生成的点是图形区域高度的 5% 。Pdsize 正值表示点形状的绝对大小，Pdsize 为负值解释为相对视口大小的百分比。图形重新生成时，会重新计算所有点的大小。

修改 Pdmode 和 Pdsize 的值后，现有点的形状会在下次重新生成图形时改变。

创建一个点对象并修改其外观

下面的实例代码在 Model 空间创建一个 Point 对象，坐标为(4, 3, 0)，然后更新 Pdmode 和 Pdzise 属性。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddPointAndSetPointStyle")> _
Public Sub AddPointAndSetPointStyle()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 在模型空间创建点(4, 3, 0)
      Dim acPoint As DBPoint = New DBPoint(New Point3d(4, 3, 0))
```

```
        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acPoint)
        acTrans.AddNewlyCreatedDBObject(acPoint, True)

        '' 设置图形中所有点的样式
        acCurDb.Pdmode = 34
        acCurDb.Pdsize = 1

        '' 将新对象保存到数据库
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddPointAndSetPointStyle")]
public static void AddPointAndSetPointStyle()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 在 Model 空间创建点(4, 3, 0)
      DBPoint acPoint = new DBPoint(new Point3d(4, 3, 0));

      // 将新对象添加到块表记录和事务
```

```
        acBlkTblRec.AppendEntity(acPoint);
        acTrans.AddNewlyCreatedDBObject(acPoint, true);

        //设置图形中所有点的样式
        acCurDb.Pdmode = 34;
        acCurDb.Pdsize = 1;

        // 将新对象保存到数据库
        acTrans.Commit();
    }
}
```

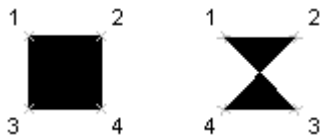
VBA/ActiveX 代码参考

```
Sub AddPointAndSetPointStyle()
    Dim pointObj As AcadPoint
    Dim location(0 To 2) As Double

    ' 定义点的位置
    location(0) = 4#: location(1) = 3#: location(2) = 0#

    Set pointObj = ThisDrawing.ModelSpace.AddPoint(location)
    ThisDrawing.SetVariable "PDMODE", 34
    ThisDrawing.SetVariable "PDSIZE", 1

    ZoomAll
End Sub
```

3.2.5 创建实体填充区域

我们可以创建具有色彩填充的三角形区域或四边形区域。创建填充区域时，将系统变量 FILLMODE 设置为关闭（off）可以提高性能，创建完成后可以再设置回打开（on）。

创建四边形实体填充区域时，第 3、4 点的顺序决定了区域的形状。比较下面的图例：



第 1、2 两点定义多边形的一个边，第 3 个点定义在第 2 个点的对角线另一端。如果第 4 个点设置成和第 3 个点相等，创建的就是一个三角形填充区域。

创建一个实体填充对象

下面示例用坐标(0,0,0)、 (5,0,0)、 (5,8,0)和 (0,8,0)在 Model 空间创建一个四边形实体（蝴蝶结），还用坐标(10,0,0)、 (15,0,0)、 (10,8,0)和 (15,8,0)创建了一个长方形的四边实体。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Add2DSolid")> _
Public Sub Add2DSolid()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 在模型空间创建一个蝴蝶结
      Dim ac2DSolidBow As Solid = New Solid(New Point3d(0, 0, 0), _
                                            New Point3d(5, 0, 0), _
                                            New Point3d(5, 8, 0), _
                                            New Point3d(0, 8, 0))

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(ac2DSolidBow)
      acTrans.AddNewlyCreatedDBObject(ac2DSolidBow, True)

      ''在模型空间创建一个四边形实体
      Dim ac2DSolidSqr As Solid = New Solid(New Point3d(10, 0, 0), _
                                            New Point3d(15, 0, 0), _
                                            New Point3d(10, 8, 0), _
                                            New Point3d(15, 8, 0))
```

```
      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(ac2DSolidSqr)
      acTrans.AddNewlyCreatedDBObject(ac2DSolidSqr, True)

      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("Add2DSolid")]
public static void Add2DSolid()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 在 Model 空间创建四边形实体（蝴蝶结）
      Solid ac2DSolidBow = new Solid(new Point3d(0, 0, 0),
                                     new Point3d(5, 0, 0),
                                     new Point3d(5, 8, 0),
                                     new Point3d(0, 8, 0));

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(ac2DSolidBow);
      acTrans.AddNewlyCreatedDBObject(ac2DSolidBow, true);
```

```
      // 在 Model 空间创建矩形实体
      Solid ac2DSolidSqr = new Solid(new Point3d(10, 0, 0),
                                     new Point3d(15, 0, 0),
                                     new Point3d(10, 8, 0),
                                     new Point3d(15, 8, 0));
      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(ac2DSolidSqr);
      acTrans.AddNewlyCreatedDBObject(ac2DSolidSqr, true);

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub Add2DSolid()
    Dim solidObj As AcadSolid
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim point3(0 To 2) As Double
    Dim point4(0 To 2) As Double

    ' 定义实体
    point1(0) = 0#: point1(1) = 0#: point1(2) = 0#
    point2(0) = 5#: point2(1) = 0#: point2(2) = 0#
    point3(0) = 5#: point3(1) = 8#: point3(2) = 0#
    point4(0) = 0#: point4(1) = 8#: point4(2) = 0#
    ' 在模型空间创建 Solid 对象
    Set solidObj = ThisDrawing.ModelSpace.AddSolid _
                                    (point1, point2, point3, point4)

    ' 定义实体
    point1(0) = 10#: point1(1) = 0#: point1(2) = 0#
    point2(0) = 15#: point2(1) = 0#: point2(2) = 0#
    point3(0) = 10#: point3(1) = 8#: point3(2) = 0#
    point4(0) = 15#: point4(1) = 8#: point4(2) = 0#
    ' 在模型空间创建 Solid 对象
    Set solidObj = ThisDrawing.ModelSpace.AddSolid _
                                    (point1, point2, point3, point4)

    ZoomAll
End Sub
```

3.2.6 使用面域

面域（Region）是由称之为环的闭合形状构成的二维闭合区域。环是由互不相交的直的和弯曲的对象组成的闭合边界。环可以是直线、轻量级多段线、2D 多段线、3D 多段线、圆、圆弧、椭圆、椭圆弧、样条曲线、3D 面、宽线（Trace）和填充实体等的组合。（译者注：Trace 绘制宽线，该命令在 AutoCAD2012 版已废弃。）

形成环的对象必须是闭合的，或与其他对象共用端点以形成闭合区域。这些对象还必须是共面的（在同一平面上）。构成面域的环必须定义为对象数组。

关于使用面域的更多内容，见《AutoCAD 用户指南》的“创建并合并区域（面域）”一节。

3.2.6.1 创建面域

通过创建一个 Region 对象实例并将该实例添加到 BlockTableRecord 上，来实现将面域添加到 BlockTableRecord 对象。在添加到 BlockTableRecord 对象之前，需要基于形成闭环的对象对面域进行计算。CreateFromCurves()函数使用输入的对象数组构成的每个闭环来创建面域。CreateFromCurves()方法请求并返回一个 DBObjectCollection 对象。

AutoCAD 将闭合的二维多段线及平面上的三维多段线转变为单独的面域，然后将形成平面闭环的多段线、直线、曲线转变为面域。如果有两条以上的曲线共用一个端点，那最后得到的面域可能是不确定的。这种情况下，CreateFromCurves()方法实际上会创建多个面域。要逐个将得到的面域添加到 BlockTableRecord 对象中去。

创建一个简单面域

下例用一个圆创建一个面域。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddRegion")> _
Public Sub AddRegion()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)
```

```
      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 在内存中创建一个圆
      Using acCirc As Circle = New Circle()
          acCirc.Center = New Point3d(2, 2, 0)
          acCirc.Radius = 5

          '' 添加到对象数组
          Dim acDBObjColl As DBObjectCollection = New DBObjectCollection()
          acDBObjColl.Add(acCirc)

          '' 基于每个闭环计算面域
          Dim myRegionColl As DBObjectCollection = New DBObjectCollection()
          myRegionColl = Region.CreateFromCurves(acDBObjColl)
          Dim acRegion As Region = myRegionColl(0)

          '' 将新对象添加到块表记录和事务
          acBlkTblRec.AppendEntity(acRegion)
          acTrans.AddNewlyCreatedDBObject(acRegion, True)

          '' Dispose of the in memory object not appended to the database
      End Using

      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddRegion")]
public static void AddRegion()
{
  // 获取当前文档和数据库
  //获取当前文档及数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
  Database acCurDb = acDoc.Database;
  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;
      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;
      // 在内存创建一个圆
      using (Circle acCirc = new Circle())
      {
          acCirc.Center = new Point3d(2, 2, 0);
          acCirc.Radius = 5;

          // 将圆添加到对象数组
          DBObjectCollection acDBObjColl = new DBObjectCollection();
          acDBObjColl.Add(acCirc);

          // 基于每个闭环计算面域
          DBObjectCollection myRegionColl = new DBObjectCollection();
          myRegionColl = Region.CreateFromCurves(acDBObjColl);
          Region acRegion = myRegionColl[0] as Region;

          // 将新对象添加到块表记录和事务
          acBlkTblRec.AppendEntity(acRegion);
          acTrans.AddNewlyCreatedDBObject(acRegion, true);

          //处置内存中的圆，不添加到数据库;
      }

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddRegion()
    ’ 定义拥有面域边界的数组
    Dim curves(0 To 0) As AcadCircle
```

```
    ' 创建一个圆作为面域的边界
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2
    center(1) = 2
    center(2) = 0
    radius = 5#
    Set curves(0) = ThisDrawing.ModelSpace.AddCircle _
                                    (center, radius)

    ' 创建面域
    Dim regionObj As Variant
    regionObj = ThisDrawing.ModelSpace.AddRegion(curves)

    ZoomAll
End Sub
```

3.2.6.2 创建组合面域

可以通过面域之间或 3D 实体之间的差集、并集、交集来创建组合面域，然后通过拉伸或旋转复合面域来创建复杂的实体。要创建一个组合面域，使用 BooleanOperation()方法。调用格式如下（C#）：

```
thisRegin.BooleanOperation(
    BooleanOperationType operation,
    Region otherRegion
);
```

其中的 operation 表示组合操作的模式，由下面的 enum 定义：

```
public enum BooleanOperationType {
  BoolUnite,          // 求并集
  BoolIntersect,      // 求交集
  BoolSubtract        // 求差集

}
```

差集面域

要从一个面域 thisRegin 中减去另一个面域 otherRegion，使用常量 BooleanOperationType.BoolSubtract 调用面域 thisRegin 的BooleanOperation()方法即可。

比如，要计算铺地板需要多少地毯，就调用整个地板空间外边界的 BooleanOperation()方法，并使用不需地毯的区域，如柱子、柜子等，作为参数列表中的对象 otherRegion。

并集面域

要合并面域，使用常量 BooleanOperationType.BoolUnite 来调用 BooleanOperation()方法即可。可以以任意顺序对各面域进行合并来获得组合面域。

求两个面域的交集

求两个面域的交集，使用常量 BooleanOperationType.BoolIntersect 来调用 BooleanOperation()方法。可以按任意顺序对各面域求交集来获得组合面域。

创建一个组合面域

下面这个示例用两个圆创建两个面域,然后从较大的一个中减去较小的一个来创建一个轮形面域。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateCompositeRegions")> _
Public Sub CreateCompositeRegions()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 在内存建两个圆
```

```
      Dim acCirc1 As Circle = New Circle()
      acCirc1.Center = New Point3d(4, 4, 0)
      acCirc1.Radius = 2

      Dim acCirc2 As Circle = New Circle()
      acCirc2.Center = New Point3d(4, 4, 0)
      acCirc2.Radius = 1

      '' 将圆添加到对象数组
      Dim acDBObjColl As DBObjectCollection = New DBObjectCollection()
      acDBObjColl.Add(acCirc1)
      acDBObjColl.Add(acCirc2)

      '' 基于每个闭环计算面域
      Dim myRegionColl As DBObjectCollection = New DBObjectCollection()
      myRegionColl = Region.CreateFromCurves(acDBObjColl)
      Dim acRegion1 As Region = myRegionColl(0)
      Dim acRegion2 As Region = myRegionColl(1)

      '' 从面域 2 减去面域 1
      If acRegion1.Area > acRegion2.Area Then
          '' 从较大面域中减去较小面域
          acRegion1.BooleanOperation(BooleanOperationType.BoolSubtract,
acRegion2)
          acRegion2.Dispose()

          '' 将最终的面域添加到数据库
          acBlkTblRec.AppendEntity(acRegion1)
          acTrans.AddNewlyCreatedDBObject(acRegion1, True)
      Else
          '' 从较大面域中减去较小面域
          acRegion2.BooleanOperation(BooleanOperationType.BoolSubtract,
acRegion1)
          acRegion1.Dispose()

          '' 将最终的面域添加到数据库
          acBlkTblRec.AppendEntity(acRegion2)
          acTrans.AddNewlyCreatedDBObject(acRegion2, True)
      End If

      '' 销毁内存中的两个圆对象
      acCirc1.Dispose()
      acCirc2.Dispose()
```

```
      '' 将新对象保存到数据库
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateCompositeRegions")]
public static void CreateCompositeRegions()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      //  在内存建两个圆
      Circle acCirc1 = new Circle();
      acCirc1.Center = new Point3d(4, 4, 0);
      acCirc1.Radius = 2;

      Circle acCirc2 = new Circle();
      acCirc2.Center = new Point3d(4, 4, 0);
      acCirc2.Radius = 1;

      // 将圆添加到对象数组
      DBObjectCollection acDBObjColl = new DBObjectCollection();
      acDBObjColl.Add(acCirc1);
```

```
        acDBObjColl.Add(acCirc2);

        // 基于每个闭环计算面域
        DBObjectCollection myRegionColl = new DBObjectCollection();
        myRegionColl = Region.CreateFromCurves(acDBObjColl);
        Region acRegion1 = myRegionColl[0] as Region;
        Region acRegion2 = myRegionColl[1] as Region;

        // 从面域 2 减去面域 1
        if (acRegion1.Area > acRegion2.Area)
        {
            // 从较大面域中减去较小面域
            acRegion1.BooleanOperation(BooleanOperationType.BoolSubtract,
acRegion2);
            acRegion2.Dispose();

            // 将最终的面域添加到数据库
            acBlkTblRec.AppendEntity(acRegion1);
            acTrans.AddNewlyCreatedDBObject(acRegion1, true);
        }
        else
        {
            // 从较大面域中减去较小面域
            acRegion2.BooleanOperation(BooleanOperationType.BoolSubtract,
acRegion1);
            acRegion1.Dispose();

            // 将最终的面域添加到数据库
            acBlkTblRec.AppendEntity(acRegion2);
            acTrans.AddNewlyCreatedDBObject(acRegion2, true);
        }

        // 销毁内存中的两个圆对象
        acCirc1.Dispose();
        acCirc2.Dispose();

        // 将新对象保存到数据库
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateCompositeRegions()
```

```
    ’ 创建 2 个圆，大圆表示车轮的外缘，小圆表示车轮的内缘
    Dim WheelParts(0 To 1) As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 4
    center(1) = 4
    center(2) = 0
    radius = 2#
    Set WheelParts(0) = ThisDrawing.ModelSpace. _
                            AddCircle(center, radius)
    radius = 1#
    Set WheelParts(1) = ThisDrawing.ModelSpace. _
                            AddCircle(center, radius)

    ’ 用 2 个圆创建面域
    Dim regions As Variant
    regions = ThisDrawing.ModelSpace.AddRegion(WheelParts)

    ’ 将面域复制给变量
    Dim WheelOuter As AcadRegion
    Dim WheelInner As AcadRegion

    If regions(0).Area > regions(1).Area Then
        ’ 第 1 个面域是轮子外缘
        Set WheelOuter = regions(0)
        Set WheelInner = regions(1)
    Else
        ’第 1 个面域是轮子内缘
        Set WheelInner = regions(0)
        Set WheelOuter = regions(1)
    End If

    ’ 从大面域里抠掉小面域
    WheelOuter.Boolean acSubtraction, WheelInner
End Sub
```

3.2.7 创建图案填充

具有闭合边界的图形可以使用填充图案进行填充。

创建图案填充时，不必一上来就指定要填充的区域，应首先创建 Hatch 对象，然后可以指定外环，也就是要填充的区域的最外边界，然后可以继续指定可能存在的所有填充内环。

关于使用图案填充的更多内容，见《AutoCAD 用户指南》的“填充图案与填充概览”。

3.2.7.1 创建填充（Hatch）对象

创建 Hatch 对象时，需指定填充图案类型、填充图案名称及关联性。一旦创建好 Hatch 对象，就不能修改关联性。

所谓创建 Hatch 对象，就是创建 Hatch 对象的一个新的实例，然后用 AppendEntity()方法将该实例添加到 BlockTableRecord 对象中。

3.2.7.2 关联图案填充

可以创建关联图案填充或非关联图案填充。关联图案填充就是图案填充对象与其边界是关联的，当边界改变时图案填充随之更新。而非关联图案填充是不依赖其边界的。

要想创建关联图案填充，需将所创建图案填充对象的 Associative 属性设置为 TRUE。创建非关联图案填充，就将 Associative 属性设置为 FALSE。

图案填充的关联属性必须在将其添加到填充环之前设置。如果图案填充对象是非关联的，可以通过设置 Associative 属性值为 TRUE 将其重新设为关联的，然后再重新将其添加到填充环内。

3.2.7.3 指定填充图案的类型和名称

AutoCAD 提供了一个实体填充和多至 50 种工业标准的填充图案。填充图案用来强调图形中的特殊形状或区域。比如，图案可以帮助区分三维对象的部件、表示构成对象的材质等。

我们可以使用 AutoCAD 提供的填充图案，或使用外部填充库提供的填充图案。AutoCAD 提供的填充图案的列表，见 *AutoCAD 命令参考指南*。

要指定某一特定的填充图案，必须同时指定 Hatch 对象的填充图案类型和名称。填充图案类型指明到哪里去查找填充图案的名称。输入填充图案类型时，使用下列常量之一：

HatchPatternType.PreDefined

从 acad.pat 文件中定义的图案名称中选择；

HatchPatternType.UserDefined

使用当前线型定义一个直线图案；

HatchPatternType.CustomDefined

从 acad.pat 以外的 PAT 文件中选择图案名称

输入图案名称时，使用的名称必须是图案类型指定的 PAT 文件里的有效名称。

3.2.7.4 定义填充边界

创建 Hatch 对象之后，就可以添加填充边界。边界可以是直线、圆弧、圆、二维多段线、椭圆、样条曲线以及面域等对象的任意组合。

首先添加的必须是外侧边界，它定义了图案填充的最外界限。添加外侧边界，使用 HatchLoopTypes.Outermost 常量调用 AppendLoop()方法，该常量定义了封闭边界的类型。

定义好外侧边界后，可以继续添加其他边界。使用 HatchLoopTypes.Default 常量调用 AppendLoop()方法可以添加内边界。

内边界定义填充区域内的孤岛区域。Hatch 对象如何处理这些孤岛区域依赖于 HatchStyle 属性的设置。HatchStyle 属性值及描述见下表：

填充样式定义		
填充样式	**HatchStyle 属性值**	**描述**
1 	Normal (HatchStyle.Normal)	表示标准样式或常规样式，是 HatchStyle 属性的默认设置。AutoCAD 从最外边往里填充，当遇到一个内边界时关闭填充直到遇到另一个内边界。
2 	Outer (HatchStyle.Outer)	只填充最外区域。本样式也是从最外边往里填充，但当遇到一个内边界时关闭填充就不再打开。
3 	Ignore (HatchStyle.Ignore)	忽略内部结构。本选项填充全部内部对象。

完成对图案填充的定义后，必须在显示前对其进行评估。使用 EvaluateHatch()方法来作这件事。

创建 Hatch 对象

下面的示例在模型空间创建一个关联图案填充。图案填充创建完后，可以改变填充所关联的圆的大小，填充将改变自己来适应圆的大小。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddHatch")> _
Public Sub AddHatch()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建一个圆作为填充的封闭边界
      Dim acCirc As Circle = New Circle()
      acCirc.Center = New Point3d(3, 3, 0)
      acCirc.Radius = 1

      '' 将圆添加到块表记录和事务
      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      '' 将圆的 ObjectId 添加到对象数组
      Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
      acObjIdColl.Add(acCirc.ObjectId)

      '' 创建填充对象并添加到块表记录
      Dim acHatch As Hatch = New Hatch()
      acBlkTblRec.AppendEntity(acHatch)
      acTrans.AddNewlyCreatedDBObject(acHatch, True)

      '' 设置填充对象的属性
      '' 在调用 AppendLoop 之前设置填充对象的关联属性
      acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31")
```

```
        acHatch.Associative = True
        acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl)
        acHatch.EvaluateHatch(True)

        '' 将新对象保存到数据库
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddHatch")]
public static void AddHatch()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建一个圆作为填充的封闭边界
      Circle acCirc = new Circle();
      acCirc.Center = new Point3d(3, 3, 0);
      acCirc.Radius = 1;

      // 将圆添加到块表记录和事务
      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);
```

```
        // 将圆的 ObjectId 添加到对象数组
        ObjectIdCollection acObjIdColl = new ObjectIdCollection();
        acObjIdColl.Add(acCirc.ObjectId);

        // 创建填充对象并添加到块表记录
        Hatch acHatch = new Hatch();
        acBlkTblRec.AppendEntity(acHatch);
        acTrans.AddNewlyCreatedDBObject(acHatch, true);

        // 设置填充对象的属性
        // 在调用 AppendLoop 之前设置填充对象的关联属性
        acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31");
        acHatch.Associative = true;
        acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl);
        acHatch.EvaluateHatch(true);

        // 将新对象保存到数据库
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub AddHatch()
    Dim hatchObj As AcadHatch
    Dim patternName As String
    Dim PatternType As Long
    Dim bAssociativity As Boolean

    ' 定义填充图案
    patternName = "ANSI31"
    PatternType = 0
    bAssociativity = True

    ' 创建关联 Hatch 对象
    Set hatchObj = ThisDrawing.ModelSpace.AddHatch _
                        (PatternType, patternName, bAssociativity)

    ' 创建圆作为填充的外边界
    Dim outerLoop(0 To 0) As AcadEntity
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 3: center(1) = 3: center(2) = 0
    radius = 1
    Set outerLoop(0) = ThisDrawing.ModelSpace. _
                            AddCircle(center, radius)
```

```
    ’添加填充对象边界，显示填充
    hatchObj.AppendOuterLoop (outerLoop)
    hatchObj.Evaluate
    ThisDrawing.Regen True
End Sub
```

3.3 使用选择集

选择集可以由单个对象组成，或者是一组复杂的对象组：比如某图层上的一组对象。选择集的典型创建过程，是在通过先选择后执行（Pickfirst）方式启动命令前，或在命令行出现

选择对象：

提示时，用户按要求选择图形中的对象。

选择集不是持久生存的对象，如果需要在多个命令间维持选择集，或为以后的使用保留选择集，需要创建一个自定义字典并记录下选择集中的每个对象的 ObjectId。

3.3.1 获得先选择后执行（PickFirst）选择集

在启动命令之前选择对象就创建了 PickFirst 选择集。获得 PickFirst 选择集对象必须具备下列几个条件：

- 系统变量 PICKFIRST 必须设置为 1；
- 要使用 PickFirst 选择集的命令必须定义好 UsePickSet 命令标志；
- 调用 SelectImplied()方法获得 PickFirst 选择集；

SetImpliedSelection()方法用来清空当前 PickFirst 选择集。

获得 PickFirst 选择集

本例显示 PickFirst 选择集中对象的数量，然后请求用户选择其他的对象。在请求用户选择对象之前，使用 SetImpliedSelection()方法清空了当前 PIckFirst 选择集。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)> _
Public Sub CheckForPickfirstSelection()
  '' 获取当前文档
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 获取 PickFirst 选择集
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.SelectImplied()

  Dim acSSet As SelectionSet

  '' 如果提示状态 OK，说明启动命令前选择了对象
  If acSSPrompt.Status = PromptStatus.OK Then
      acSSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects in Pickfirst selection: " &
_
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects in Pickfirst selection: 0")
  End If

  '' 清空 PickFirst 选择集
  Dim idarrayEmpty() As ObjectId
  acDocEd.SetImpliedSelection(idarrayEmpty)

  '' 求从图形区域选择对象
  acSSPrompt = acDocEd.GetSelection()

  '' 如果提示状态 OK，表示已选择对象
  If acSSPrompt.Status = PromptStatus.OK Then
      acSSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
```

```
                                    acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("CheckForPickfirstSelection", CommandFlags.UsePickSet)]
public static void CheckForPickfirstSelection()
{
  // 获取当前文档获
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 获取 PickFirst 选择集
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.SelectImplied();

  SelectionSet acSSet;

  // 如果提示状态 OK，说明启动命令前选择了对象;
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects in Pickfirst selection: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects in Pickfirst selection: 0");
  }

  // 清空 PickFirst 选择集
  ObjectId[] idarrayEmpty = new ObjectId[0];
  acDocEd.SetImpliedSelection(idarrayEmpty);

  // 请求从图形区域选择对象
  acSSPrompt = acDocEd.GetSelection();
```

```
  // 如果提示状态 OK，表示已选择对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

VBA/ActiveX 代码参考

```
Sub CheckForPickfirstSelection()
    ' 获取 PickFirst 选择集
    Dim acSSet As AcadSelectionSet
    Set acSSet = ThisDrawing.PickfirstSelectionSet

    ' 显示已选对象的个数
    MsgBox "Number of objects in Pickfirst selection set: " & acSSet.Count

    ' 创建新选择集
    Dim acSSetUser As AcadSelectionSet
    Set acSSetUser = ThisDrawing.SelectionSets.Add("User")

    ' 选择图形中的对象
    acSSetUser.SelectOnScreen

    ' 显示已选对象的个数
    MsgBox "Number of objects selected: " & acSSetUser.Count
    ' 删除选择集
    acSSetUser.Delete
End Sub
```

3.3.2 在绘图区域选择对象

可以通过与用户交互来选择对象，或者，通过.NET API 模拟各种不同的对象选择选项。如果程序执行多个选择集，要么需要跟踪返回的每个选择集，要么需要创建一个 ObjectIdCollection 对象来跟踪所有已选择的对象。下列函数用来从图形中选择对象：

GetSelection

提示用户从屏幕拾取对象。

SelectAll

选择当前空间内所有未锁定及未冻结的对象。

SelectCrossingPolygon

选择由给定点定义的多边形内的所有对象以及与多边形相交的对象。多边形可以是任意形状，但不能与自己交叉或接触。

SelectCrossingWindow

选择由两个点定义的窗口内的对象以及与窗口相交的对象。

SelectFence

选择与选择围栏相交的所有对象。围栏选择与多边形选择类似，所不同的是围栏不是封闭的，围栏同样不能与自己相交。

SelectLast

选择当前空间中最后创建的那个对象。

SelectPrevious

选择前一个“选择对象：”提示符期间已选定的所有对象。

SelectWindow

选择完全框入由两个点定义的矩形内的所有对象。

SelectWindowPolygon

选择完全框入由点定义的多边形内的对象。多边形可以是任意形状，但不能与自己交叉或接触。

SelectAtPoint

选择通过给定点的对象，并将其放入活动选择集。

SelectByPolygon

选择围栏里面的对象，并将其添加到活动选择集。

提示选择屏幕上的对象并遍历选择集

本示例代码提示用户选择对象，然后将选定的每个对象的颜色改为绿色（AutoCAD 颜色索引号 3）。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
```

```
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("SelectObjectsOnscreen")> _
Public Sub SelectObjectsOnscreen()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 请求在图形区域选择对象
      Dim acSSPrompt As PromptSelectionResult = acDoc.Editor.GetSelection()

      '' 如果提示状态 OK，表示已选择对象
      If acSSPrompt.Status = PromptStatus.OK Then
          Dim acSSet As SelectionSet = acSSPrompt.Value

          '' 遍历选择集中的对象
          For Each acSSObj As SelectedObject In acSSet
              '' 确认返回的是合法的 SelectedObject 对象
              If Not IsDBNull(acSSObj) Then
                  '' 以写模式打开所选对象
                  Dim acEnt As Entity = acTrans.GetObject(acSSObj.ObjectId, _
                                                          OpenMode.ForWrite)

                  If Not IsDBNull(acEnt) Then
                      '' 将对象颜色修改为绿色
                      acEnt.ColorIndex = 3
                  End If
              End If
          Next

          '' 将新对象保存到数据库
          acTrans.Commit()
      End If

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
```

```
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("SelectObjectsOnscreen")]
public static void SelectObjectsOnscreen()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 请求在图形区域选择对象
      PromptSelectionResult acSSPrompt = acDoc.Editor.GetSelection();

      // 如果提示状态 OK，表示已选择对象
      if (acSSPrompt.Status == PromptStatus.OK)
      {
          SelectionSet acSSet = acSSPrompt.Value;

          // 遍历选择集内的对象
          foreach (SelectedObject acSSObj in acSSet)
          {
              // 确认返回的是合法的 SelectedObject 对象
              if (acSSObj != null)
              {
                  // 以写模式打开所选对象
                  Entity acEnt = acTrans.GetObject(acSSObj.ObjectId,
                                                   OpenMode.ForWrite) as Entity;

                  if (acEnt != null)
                  {
                      // 将对象颜色修改为绿色
                      acEnt.ColorIndex = 3;
                  }
              }
          }

          // 保存新对象到数据库
          acTrans.Commit();
      }
      // 关闭事务
  }
```

```
}
```

VBA/ActiveX 代码参考

```
Sub SelectObjectsOnscreen()
    ' 新建一个选择集
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' 提示用户选择对象
    ' 并添加到选择集
    sset.SelectOnScreen

    Dim acEnt As AcadEntity

    ' 遍历所选对象并将每个对象的颜色改为绿色
    For Each acEnt In sset
        ' 使用 Color 属性设置对象的颜色
        acEnt.color = acGreen
    Next acEnt

    ' 最后删除选择集
    sset.Delete
End Sub
```

选择与窗口相交的对象

本示例代码演示选择窗口内的以及与窗口相交的对象。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("SelectObjectsByCrossingWindow")> _
Public Sub SelectObjectsByCrossingWindow()
  '' 获取当前文档编辑器
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' Create a crossing window from (2,2,0) to (10,8,0)
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.SelectCrossingWindow(New Point3d(2, 2, 0), _
```

```
                                New Point3d(10, 8, 0))

  '' 如果提示状态 OK，表示已选择对象
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("SelectObjectsByCrossingWindow")]
public static void SelectObjectsByCrossingWindow()
{
  // 获取当前文档编辑器
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 从(2,2,0)到(10,8,0)创建一个交叉窗口
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.SelectCrossingWindow(new Point3d(2, 2, 0),
                                            new Point3d(10, 8, 0));

  // 如果提示状态 OK，表示已选择对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
```

```
}
```

VBA/ActiveX 代码参考

```
Sub SelectObjectsByCrossingWindow()
    ' 新建选择集
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' 定义交叉窗口的 2 个点
    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double

    pt1(0) = 2#: pt1(1) = 2#: pt1(2) = 0#:
    pt2(0) = 10#: pt2(1) = 8#: pt2(2) = 0#:

    ' 创建窗口
    sset.Select acSelectionSetCrossing, pt1, pt2

    MsgBox "Number of objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

3.3.3 添加或合并多个选择集

可以合并多个选择集，方法是先创建一个 ObjectIdCollection 集合对象，然后将多个选择集中的对象 ObjectId 都添加到集合中。除了向 ObjectIdCollection 对象添加对象 ObjectId，还可以从中删除对象 ObjectId。所有对象 ObjectId 都添加到 ObjectIdCollection 集合对象后，可以遍历该集合，并根据需要操作其中的每个对象。

将所选对象添加到选择集

本例两次提示用户选择对象，然后将两个选择集合并为一个选择集。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
```

```
<CommandMethod("MergeSelectionSets")> _
Public Sub MergeSelectionSets()
  '' 获取当前文档编辑器
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 请求在图形区域选择对象
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection()

  Dim acSSet1 As SelectionSet
  Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()

  '' 如果提示状态 OK，表示已选择对象
  If acSSPrompt.Status = PromptStatus.OK Then
      '' 获取所选对象
      acSSet1 = acSSPrompt.Value

      '' 将选择集 1 的对象添加到集合里
      acObjIdColl = New ObjectIdCollection(acSSet1.GetObjectIds())
  End If

  '' 请求在图形区域选择对象
  acSSPrompt = acDocEd.GetSelection()

  Dim acSSet2 As SelectionSet

  '' 如果提示状态 OK，表示已选择对象
  If acSSPrompt.Status = PromptStatus.OK Then
      acSSet2 = acSSPrompt.Value

      '' 检查集合的长度，如果是 0，就直接用选择集 2 初始化集合
      If acObjIdColl.Count = 0 Then
          acObjIdColl = New ObjectIdCollection(acSSet2.GetObjectIds())
      Else
          Dim acObjId As ObjectId

          '' 遍历选择集 2
          For Each acObjId In acSSet2.GetObjectIds()
              '' 将选择集 2 中的对象添加到集合里
              acObjIdColl.Add(acObjId)
          Next
      End If
  End If
```

```
  Application.ShowAlertDialog("Number of objects selected: " & _
                              acObjIdColl.Count.ToString())
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("MergeSelectionSets")]
public static void MergeSelectionSets()
{
  // 获取当前文档编辑器
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 请求在图形区域选择对象
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection();

  SelectionSet acSSet1;
  ObjectIdCollection acObjIdColl = new ObjectIdCollection();

  // 如果提示状态 OK，表示已选择对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      // 获取所选对象
      acSSet1 = acSSPrompt.Value;

      // 向 ObjectIdCollection 中追加选择集 1 的对象
      acObjIdColl = new ObjectIdCollection(acSSet1.GetObjectIds());
  }

  // 请求在图形区域选择对象
  acSSPrompt = acDocEd.GetSelection();

  SelectionSet acSSet2;

  // 如果提示状态 OK，表示已选择对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      acSSet2 = acSSPrompt.Value;

      // 检查 ObjectIdCollection 集合大小，如果为 0 就用选择集 2 对其初始化
```

```
      if (acObjIdColl.Count == 0)
      {
          acObjIdColl = new ObjectIdCollection(acSSet2.GetObjectIds());
      }
      else
      {
          // 遍历选择集 2
          foreach (ObjectId acObjId in acSSet2.GetObjectIds())
          {
              // 将第二个选择集中的每个对象添加到集合内
              acObjIdColl.Add(acObjId);
          }
      }
  }

  Application.ShowAlertDialog("Number of objects selected: " +
                              acObjIdColl.Count.ToString());
}
```

VBA/ActiveX 代码参考

```
Sub MergeSelectionSets()
    ' 创建新选择集
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' 提示用户选择对象
    ' 并添加到选择集.
    sset.SelectOnScreen

    ' 再次提示用户选择对象
    ' 并添加到相同的选择集里
    sset.SelectOnScreen

    MsgBox "Number of total objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

3.3.4 定义选择集过滤器规则

我们可以通过使用选择过滤器来限制哪些对象被选中并添加到选择集。选择过滤器列表通过属性或类型过滤所选对象，例如，我们可能想只选择蓝色的对象或某一图层上的对象。我们还可以使用选择条件组合，例如，我们可以创建一个选择过滤器将选择对象限定于 Pattern 图层上的蓝色的圆。可以为（§3.3.2 *在图形区域选择对象*）一节中的各种不同选择方法指定选择过滤器参数。

注：使用过滤只能识别显式赋给对象的值，而不能识别继承自图层的那些值。比如，如果对象的线型属性设置为随图层（ByLayer）而该图层线型为 Hidden，那么要过滤线型为 Hidden 的对象将不会选择那些线型属性为随图层（ByLayer）的对象。

3.3.4.1 使用选择过滤器定义选择集规则

选择过滤器由一对 TypedValue 参数构成。TypedValue 的第一个参数表明过滤器的类型（例如对象），第二个参数为要过滤的值（例如圆）。过滤器类型是一个 DXF 组码，用来指定使用哪种过滤器。一些常用过滤器类型列表如下。

常见过滤器 DXF 组码	
DXF 组码	**过滤器类型**
0（或 DxfCode.Start）	对象类型（字符串格式），例如“Line”、“Circle”、“Arc”等等
2（或 DxfCode.BlockName）	块名（字符串格式），插入引用的块名
8（或 DxfCode.LayerName）	图层名（字符串格式），例如“Layer 0”
60（或 DxfCode.Visibility）	对象可见性（整型），0 = 可见，1 = 不可见.
62（或 DxfCode.Color）	颜色号（整型），0~256 数字索引值。0 代表随块 BYBLOCK，256 代表随层 BYLAYER，负值表示图层关闭了。
67	模型空间/图纸空间指示符（整型），0 或忽略 = 模型空间；1 = 图纸空间

DXF 组码的完整列表，见《DXF 参考手册》中*组码值类型*一节

为选择集指定单个选择条件

下面程序提示用户选择对象放到选择集内，然后过滤掉圆以外的其他所有对象。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterSelectionSet")> _
Public Sub FilterSelectionSet()
  '' 获取当前文档编辑器
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 使用 TypedValue 数组定义过滤条件
  Dim acTypValAr(0) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "CIRCLE"), 0)

  ''将过滤器条件赋值给 SelectionFilter 对象
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' 请求用户在图形区域选择对象
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' 提示状态 OK，表示已选择对象
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
```

```
[CommandMethod("FilterSelectionSet")]
public static void FilterSelectionSet()
{
  // 获取当前文档编辑器
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 创建一个 TypedValue 数组来定义过滤器条件
  TypedValue[] acTypValAr = new TypedValue[1];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "CIRCLE"), 0);

  // 将过滤器条件赋值给 SelectionFilter 对象
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // 请求用户在图形区域选择对象
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // 提示状态 OK，表示已选择对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

VBA/ActiveX 代码参考

```
Sub FilterSelectionSet()
    ' 创建新选择集
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    '定义过滤器列表，只选择圆
    Dim FilterType(0) As Integer
    Dim FilterData(0) As Variant
    FilterType(0) = 0
    FilterData(0) = "Circle"
```

```
    ' 提示用户选择对象
    ' 并添加到选择集
    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

3.3.4.2 指定多个过滤条件

选择过滤器可以包含过滤多个属性或对象的条件。可以通过声明一个包含足够大的数组来定义全部过滤条件，数组的每个元素代表一个过滤条件。

选择满足两个条件的对象

下面示例指定两个条件过滤所选的对象：对象为圆并且在 0 层上。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterBlueCircleOnLayer0")> _
Public Sub FilterBlueCircleOnLayer0()
  '' 获取当前文档 editor
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 创建 TypedValue 数组，定义过滤条件
  Dim acTypValAr(2) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Color, 5), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "CIRCLE"), 1)
  acTypValAr.SetValue(New TypedValue(DxfCode.LayerName, "0"), 2)

  '' 将过滤条件赋值给 SelectionFilter 对象
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' 请求在图形区域选择对象
  Dim acSSPrompt As PromptSelectionResult
```

```
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  ''如果提示状态 OK，表示对象已选
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterBlueCircleOnLayer0")]
public static void FilterBlueCircleOnLayer0()
{
  //获取当前文档编辑器
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 创建 TypedValue 数组定义过滤条件
  TypedValue[] acTypValAr = new TypedValue[3];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Color, 5), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "CIRCLE"), 1);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.LayerName, "0"), 2);

  // 将过滤条件赋值给 SelectionFilter 对象
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // 请求在图形区域选择对象
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // 如果提示状态 OK，表示对象已选
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
```

```
                                    acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

VBA/ActiveX 代码参考

```
Sub FilterBlueCircleOnLayer0()
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    ' 定义过滤器列表，只选择图层 0 上的蓝色的圆
    Dim FilterType(2) As Integer
    Dim FilterData(2) As Variant

    FilterType(0) = 62: FilterData(0) = 3
    FilterType(1) = 0: FilterData(1) = "Circle"
    FilterType(2) = 8: FilterData(2) = "0"

    ' 提示用户选择对象
    ' 并添加到选择集
    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

3.3.4.3 复杂的过滤条件

当指定多个选择条件时，AutoCAD 假设所选对象必须满足每个条件。我们还可以用另外一种方式定义过滤条件。对于数值项，可以使用关系运算（比如，圆的半径必须大于等于 5.0）。对于所有项，可以使用逻辑运算（比如单行文字或多行文字）。

使用 DXF 组码-4 或常量 DxfCode.Operator 表示选择过滤器中的关系预算符类型。运算符本身用字符串表示。可用的关系运算符列表如下：

关系运算符列表

运算符	描述
"*"	任何情况（总为 True）
"="	等于
"!="	不等于
"/="	不等于
"<>"	不等于
"<"	小于
"<="	小于等于
">"	大于
">="	大于等于
"&"	位与（仅限整数组）
"&="	位屏蔽等于（仅限整数组）

选择过滤器中的逻辑操作符同样用-4 组码或常量 DxfCode.Operator 表示，逻辑操作符为字符串，且必须成对出现。操作符开始于小于号（<），结束于大于号（>）。下表列出了用于选择集过滤器的逻辑操作符。

用于选择集过滤器的逻辑操作符列表		
起始操作符	包括：	结束操作符
"<AND"	一个以上操作数	"AND>"
"<OR"	一个以上操作数	"OR>"
"<XOR"	两个操作数	"XOR>"
"<NOT"	一个操作数	"NOT>"

选择半径大于等于 5.0 的圆

下面例子选择半径大于等于 5.0 的圆。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
```

```
<CommandMethod("FilterRelational")> _
Public Sub FilterRelational()
  '' 获取当前文档编辑器
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 创建 TypedValue 数组，定义过滤条件
  Dim acTypValAr(2) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "CIRCLE"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Operator, ">="), 1)
  acTypValAr.SetValue(New TypedValue(40, 5), 2)

  '' 将过滤条件赋给 SelectionFilter 对象
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' 请求在图形区域选择对象
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' 如果提示状态 OK，表示对象已选
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterRelational")]
public static void FilterRelational()
{
  // 获取当前文档编辑器
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 创建 TypedValue 数组，定义过滤条件
  TypedValue[] acTypValAr = new TypedValue[3];
```

```
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "CIRCLE"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Operator, ">="), 1);
  acTypValAr.SetValue(new TypedValue(40, 5), 2);

  // 将过滤条件赋给 SelectionFilter 对象
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // 请求在图形区域选择对象
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // 如果提示状态 OK，表示对象已选
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;
      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

VBA/ActiveX 代码参考

```
Sub FilterRelational()
    Dim sset As AcadSelectionSet
    Dim FilterType(2) As Integer
    Dim FilterData(2) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")
    FilterType(0) = 0: FilterData(0) = "Circle"
    FilterType(1) = -4: FilterData(1) = ">="
    FilterType(2) = 40: FilterData(2) = 5#

    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

选择单行文字或者多行文字

下面例子演示可以选择 Text（单行文字）或者 MText（多行文字）。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterForText")> _
Public Sub FilterForText()
  '' 获取当前文档编辑器
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 创建 TypedValue 数组，定义过滤条件
  Dim acTypValAr(3) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Operator, "<or"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "TEXT"), 1)
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "MTEXT"), 2)
  acTypValAr.SetValue(New TypedValue(DxfCode.Operator, "or>"), 3)

  '' 将过滤条件赋给 SelectionFilter 对象
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' 请求在图形区域选择对象
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' 如果提示状态 OK，说明已选对象
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value
      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterForText")]
public static void FilterForText()
{
  // 获取当前文档 editor
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 创建 TypedValue 数组，定义过滤条件
  TypedValue[] acTypValAr = new TypedValue[4];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Operator, "<or"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "TEXT"), 1);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "MTEXT"), 2);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Operator, "or>"), 3);

  // 将过滤条件赋给 SelectionFilter 对象
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // 请求在图形区域选择对象
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // 如果提示状态 OK，说明已选对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;
      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

VBA/ActiveX 代码参考

```
Sub FilterForText()
    Dim sset As AcadSelectionSet
    Dim FilterType(3) As Integer
    Dim FilterData(3) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    FilterType(0) = -4: FilterData(0) = "<or"
    FilterType(1) = 0: FilterData(1) = "TEXT"
    FilterType(2) = 0: FilterData(2) = "MTEXT"
```

```
    FilterType(3) = -4: FilterData(3) = "or>"

    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

3.3.4.4 在过滤条件里使用通配符

选择过滤器中的符号名字和字符串可以包含通配符。

下表为 AutoCAD 能够识别的通配字符，及其在字符串上下文的作用：

通配符	
字符	**定义**
# （井号）	匹配任意单个数字
@ （at）	匹配任意单个字母
. （句号）	匹配任意单个非字母字符
* （星号）	匹配任意字符序列，包括空串，可用在搜索样本的任何位置：开始、中间或末尾
? （问号）	匹配任意单个字符
~ （否定号）	如果是样本串的第一个字符，表示匹配样本串以外的任意字符（串）
[...]	匹配方括号内的任一字符
[~...]	匹配方括号内字符以外的任一字符
- （连字号）	用在括号内表示单个字符的范围
, （逗号）	分隔两个样本串
` （转义引号）	忽略特殊字符（逐个读取接下来的字符）

使用转义引号（’）表示一个字符不是通配符，应逐个字符使用。例如，要表示选择集里只包含名为“*U2”的匿名块，应使用值“`*U2”。

图 - 转义引号在键盘上的位置

选择包含指定文字的 MText

下面的示例代码定义一个选择过滤器，选择包含文字串“The”的 MText 对象。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterMtextWildcard")> _
Public Sub FilterMtextWildcard()
  '' 获取当前文档编辑器
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 创建 TypedValue 数组，定义过滤条件
  Dim acTypValAr(1) As TypedValue
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "MTEXT"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.Text, "*The*"), 1)

  '' 将过滤条件赋给 SelectionFilter 对象
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' 请求在图形区域选择对象
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' 如果提示状态 OK，说明已选对象
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value

      Application.ShowAlertDialog("Number of objects selected: " & _
```

```
                                    acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterMtextWildcard")]
public static void FilterMtextWildcard()
{
  // 获取当前文档编辑器
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 创建 TypedValue 数组，定义过滤条件
  TypedValue[] acTypValAr = new TypedValue[2];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "MTEXT"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Text, "*The*"), 1);

  // 将过滤条件赋给 SelectionFilter 对象
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // 请求在图形区域选择对象
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);

  // 如果提示状态 OK，说明已选对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;

      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

VBA/ActiveX 代码参考

```
Sub FilterMtextWildcard()
    Dim sset As AcadSelectionSet
    Dim FilterType(1) As Integer
    Dim FilterData(1) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    FilterType(0) = 0
    FilterData(0) = "MTEXT"
    FilterType(1) = 1
    FilterData(1) = "*The*"

    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

3.3.4.5 过滤扩展数据

外部应用程序可以向 AutoCAD 对象提供诸如文本串、数值、3D 点、距离、图层名等数据。这些数据我们称之为扩展数据，或叫 xdata。我们可以过滤含有指定外部程序扩展数据的实体。

选择含有扩展数据的圆

下面的示例代码过滤包含由 “MY_APP” 程序添加了扩展数据的圆。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput

<CommandMethod("FilterXdata")> _
Public Sub FilterXdata()
  '' 获取当前文档编辑器
  Dim acDocEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

  '' 创建 TypedValue 数组，定义过滤条件
  Dim acTypValAr(1) As TypedValue
```

```
  acTypValAr.SetValue(New TypedValue(DxfCode.Start, "Circle"), 0)
  acTypValAr.SetValue(New TypedValue(DxfCode.ExtendedDataRegAppName, _
                                     "MY_APP"), 1)

  '' 将过滤条件赋给 SelectionFilter 对象
  Dim acSelFtr As SelectionFilter = New SelectionFilter(acTypValAr)

  '' 请求在图形区域选择对象
  Dim acSSPrompt As PromptSelectionResult
  acSSPrompt = acDocEd.GetSelection(acSelFtr)

  '' 如果提示状态 OK，说明已选对象
  If acSSPrompt.Status = PromptStatus.OK Then
      Dim acSSet As SelectionSet = acSSPrompt.Value
      Application.ShowAlertDialog("Number of objects selected: " & _
                                  acSSet.Count.ToString())
  Else
      Application.ShowAlertDialog("Number of objects selected: 0")
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

[CommandMethod("FilterXdata")]
public static void FilterXdata()
{
  // 获取当前文档编辑器
  Editor acDocEd = Application.DocumentManager.MdiActiveDocument.Editor;

  // 创建 TypedValue 数组，定义过滤条件
  TypedValue[] acTypValAr = new TypedValue[2];
  acTypValAr.SetValue(new TypedValue((int)DxfCode.Start, "Circle"), 0);
  acTypValAr.SetValue(new TypedValue((int)DxfCode.ExtendedDataRegAppName,
                                     "MY_APP"), 1);
  // 将过滤条件赋给 SelectionFilter 对象
  SelectionFilter acSelFtr = new SelectionFilter(acTypValAr);

  // 请求在图形区域选择对象
  PromptSelectionResult acSSPrompt;
  acSSPrompt = acDocEd.GetSelection(acSelFtr);
```

```
  // 如果提示状态 OK，说明已选对象
  if (acSSPrompt.Status == PromptStatus.OK)
  {
      SelectionSet acSSet = acSSPrompt.Value;
      Application.ShowAlertDialog("Number of objects selected: " +
                                  acSSet.Count.ToString());
  }
  else
  {
      Application.ShowAlertDialog("Number of objects selected: 0");
  }
}
```

VBA/ActiveX 代码参考

```
Sub FilterXdata()
    Dim sset As AcadSelectionSet
    Dim FilterType(1) As Integer
    Dim FilterData(1) As Variant
    Set sset = ThisDrawing.SelectionSets.Add("SS1")

    FilterType(0) = 0: FilterData(0) = "Circle"
    FilterType(1) = 1001: FilterData(1) = "MY_APP"

    sset.SelectOnScreen FilterType, FilterData

    MsgBox "Number of objects selected: " & sset.Count

    ' 最后删除选择集
    sset.Delete
End Sub
```

3.3.5 从选择集删除对象

创建选择集后，接下来就可以使用所选对象的 ObjectId。选择集不允许向其中添加从中添加对象 ObjectId，也不允许从中删除对象 ObjectId，不过我们可以用 ObjectIdCollection 对象将多个选择集合并为一个集合对象使用。我们可以从 ObjectIdCollection 对象中添加或删除对象 ObjectId。从 ObjectIdCollection 对象中删除一个对象 ObjectId，使用 Remove()方法或 RemoveAt()方法。

关于合并多个选择集和使用 ObjectIdCollection 对象的更多内容，参见（§3.3.3 *添加或合并多个选择集*）。

3.4 编辑命名对象和二维对象

可以使用与对象关联的方法和属性对现有对象进行修改。如果修改了图形对象的可见性属性，记着用 Regen()方法重画屏幕上的对象。Regen()方法是 Editor 对象的一个成员。

本节主要内容有：

- 使用命名对象
- 删除对象
- 复制对象
- 偏移对象
- 变换对象
- 阵列对象
- 延伸和修剪对象
- 分解对象
- 编辑多段线
- 编辑样条曲线
- 编辑图案填充

3.4.1 使用命名对象

AutoCAD 除了使用图形对象外，还有许多非图形对象类型存储在图形数据库中。这些对象都有与之相关联的描述性名称。例如，块、图层、编组以及标注样式等，所有这些对象都被赋予一个名称，并且多数情况下可以对其重新命名。符号表记录的名称直接显示在 AutoCAD 用户界面上，多数情况下通过.NET API 编程引用对象时使用的是对象的 ObjectId。

例如，给一个图形对象的图层属性赋值使用的是 LayerTableRecord 对象的标识 ObjectId，而不是 LayerTableRecord 对象的实际名称。不过，LayerTableRecord 对象的 ObjectId 可以用所访问图层的名称从 Layer 表获得。

3.4.1.1 清理未引用的命名对象

可以随时将未引用的对象从数据库中清理出去。不可以清理被其他对象引用的对象。例如，某字体文件可能正由文字样式引用，或者，某图层正由该层上的对象引用。清理工作可以减小图形文件存盘时的大小。

从图形数据库中清除未被引用的对象使用 Puege 方法。Purge()方法需要一个包含要清除的对象的列表，列表为 ObjectIdCollection 对象或 ObjectIdGraph 对象形式。传递给 Purge()方法的 ObjectIdCollection 对象或 ObjectIdGraph 对象列表会随着可以从数据库中删除的对象更新。调用 Purge()方法后，必须显式地删除返回的每个对象。

VBA/ActiveX 交叉参考

在 ActiveX Automation 库中，我们使用 PurgeAll 方法删除所有未被引用的命名对象，而且该方法能够识别哪个对象可以删除。可是，使用.NET API 我们需提供要清除的对象给 Purge 方法，然后 Purge 方法会返回实际可以删除的对象。可见，使用.NET API 从数据库中清除所有未引用的命名对象时，相对繁琐一些。

```
ThisDrawing.PurgeAll
```

清除所有未引用的图层

下面这个例子演示怎样从数据库清除所有未引用的图层。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("PurgeUnreferencedLayers")> _
Public Sub PurgeUnreferencedLayers()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开图层表
            Dim acLyrTbl As LayerTable
            acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                                    OpenMode.ForRead)
```

```
            '' 创建一个 ObjectIdCollection 对象来保存每个图层表记录的 objectid
            Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()

            '' 遍历图层表并将每个图层添加到 ObjectIdCollection
            For Each acObjId As ObjectId In acLyrTbl
                    acObjIdColl.Add(acObjId)
            Next

            '' 从集合中删除还在使用的图层，返回可以删除的对象的集合
            acCurDb.Purge(acObjIdColl)

            '' 遍历经 Purge 方法处理过的 ObjectIdCollection 集合
            For Each acObjId As ObjectId In acObjIdColl
                    Dim acSymTblRec As SymbolTableRecord
                    acSymTblRec = acTrans.GetObject(acObjId, _
                                        OpenMode.ForWrite)

                    Try
                            '' 删除未引用的图层
                            acSymTblRec.Erase(True)
                    Catch Ex As Autodesk.AutoCAD.Runtime.Exception
                            '' 异常：图层不能删除
                            Application.ShowAlertDialog("Error:" & vbLf &
Ex.Message)
                    End Try
            Next

            '' 提交修改并关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("PurgeUnreferencedLayers")]
public static void PurgeUnreferencedLayers()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;
```

```
// 启动事务
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
        // 以读模式打开图层表
        LayerTable acLyrTbl;
        acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                     OpenMode.ForRead) as LayerTable;

        // 创建一个 ObjectIdCollection 对象来保存每个图层表记录的 objectid
        ObjectIdCollection acObjIdColl = new ObjectIdCollection();

        // 遍历图层表并将每个图层添加到 ObjectIdCollection
        foreach (ObjectId acObjId in acLyrTbl)
        {
                acObjIdColl.Add(acObjId);
        }

        // 从集合中删除还在使用的图层，返回可以删除的对象的集合
        // 调用 Purge 方法：传入参数 - ObjectIdCollection 对象；
        // 返回 - 更新了的 ObjectIdCollection 对象，包含可以删除的图层;
        acCurDb.Purge(acObjIdColl);

        // 遍历返回的 ObjectIdCollection 对象，并删除未引用的图层;
        foreach (ObjectId acObjId in acObjIdColl)
        {
                SymbolTableRecord acSymTblRec;
                acSymTblRec = acTrans.GetObject(acObjId,
                                        OpenMode.ForWrite) as SymbolTableRecord;

                try
                {
                        // 删除未引用的图层
                        acSymTblRec.Erase(true);
                }
                catch (Autodesk.AutoCAD.Runtime.Exception Ex)
                {
                        // 不能删除图层
                        Application.ShowAlertDialog("Error:\n" + Ex.Message);
                }
        }

        // 提交修改，关闭事务
        acTrans.Commit();
}
```

```
}
```

3.4.1.2 重命名对象

随着图形越来越复杂，我们可以给对象重新命名，以使对象名称有意义，或避免与所插入或附加的其他图形中的对象名发生冲突。Name 属性用来获取命名对象的当前名字，也可用来修改命名对象的名字。

我们可以对任何命名对象重新命名，AutoCAD 保留的除外，如图层 0 或 CONTINUOUS 线型。

名字最长可以有 255 个字符，除字母和数字外，还可以包含空格（尽管 AutoCAD 会删除直接出现在名称前后的空格），以及 Microsoft® Windows® 或 AutoCAD 没有使用的其他特殊字符。不能用于对象名字的特殊字符包括大于号和小于号（< >）、前后斜杠（/ \）、引号（”）、冒号（:）、分号（;）、问号（?）、逗号（,）、星号（*）、竖杠（|）、等号（=）及单引号（’）。Unicode 字体创建的特殊字符也不能用。

重命名图层

本例创建图层 0 的一个副本并命名为 MyLayer。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("RenameLayer")> _
Public Sub RenameLayer()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

           '' 返回当前数据库的图层表
           Dim acLyrTbl As LayerTable
           acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                        OpenMode.ForWrite)

           '' 复制图层 0（包括复制属性）到新图层
           Dim acLyrTblRec As LayerTableRecord
           acLyrTblRec = acTrans.GetObject(acLyrTbl("0"), _
                                OpenMode.ForRead).Clone()
```

```
            '' 修改新图层的名字
            acLyrTblRec.Name = "MyLayer"

            '' 添加新图层到图层表和事务
            acLyrTbl.Add(acLyrTblRec)
            acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)

            '' 提交修改，关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("RenameLayer")]
public static void RenameLayer()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 返回当前数据库的图层表
            LayerTable acLyrTbl;
            acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                                                    OpenMode.F
orWrite) as LayerTable;

            // 克隆图层 0（复制其及其属性）
            LayerTableRecord acLyrTblRec;
            acLyrTblRec = acTrans.GetObject(acLyrTbl["0"],
                                OpenMode.ForRead).Clone() as LayerTableRecord;

            // 修改克隆得到的图层的名称
            acLyrTblRec.Name = "MyLayer";

            // 使用图层 MyLayer 可用
            acLyrTbl.Add(acLyrTblRec);
```

```
            acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);

            // 提交修改，关闭事务
            acTrans.Commit();
    }
}
```

3.4.2 删除对象

我们可以使用 Erase()方法删除图形对象及非图形对象。

警告：尽管许多非图形对象，如 Layer 表记录及模型空间块表记录等，都拥有 Erase()方法，但不能对这些对象调用该方法，否则会发生错误。

创建一个多段线并删除它

本例创建一个轻量级多段线，然后将它删除。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EraseObject")> _
Public Sub EraseObject()
    '' 获取当前文档和数据库
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' 以读模式打开 Block 表
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

        '' 以写模式打开 Block 表记录 Model 空间
        Dim acBlkTblRec As BlockTableRecord
```

```
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_    OpenMode.ForWrite)

            '' 创建轻量多段线
            Dim acPoly As Polyline = New Polyline()
            acPoly.AddVertexAt(0, New Point2d(2, 4), 0, 0, 0)
            acPoly.AddVertexAt(1, New Point2d(4, 2), 0, 0, 0)
            acPoly.AddVertexAt(2, New Point2d(6, 4), 0, 0, 0)

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly)
            acTrans.AddNewlyCreatedDBObject(acPoly, True)

            '' 更新显示，显示告警信息
            acDoc.Editor.Regen()
            Application.ShowAlertDialog("Erase the newly added polyline.")

            '' 从图形中删除多段线
            acPoly.Erase(True)

            '' 将新对象保存到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("EraseObject")]
public static void EraseObject()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
```

```
                                                                    OpenMode.F
orRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                              OpenMode.ForWrite) as BlockTableRecord;

            // 创建轻量多段线
            Polyline acPoly = new Polyline();
            acPoly.AddVertexAt(0, new Point2d(2, 4), 0, 0, 0);
            acPoly.AddVertexAt(1, new Point2d(4, 2), 0, 0, 0);
            acPoly.AddVertexAt(2, new Point2d(6, 4), 0, 0, 0);

            // 添加新对象到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly);
            acTrans.AddNewlyCreatedDBObject(acPoly, true);

            // 更新显示并显示一条告警消息
            acDoc.Editor.Regen();
            Application.ShowAlertDialog("Erase the newly added polyline.");

            // 从图形中删除多段线
            acPoly.Erase(true);

            // 提交修改
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub EraseObject()
       ' 创建多段线
       Dim lwpolyObj As AcadLWPolyline
       Dim vertices(0 To 5) As Double
       vertices(0) = 2: vertices(1) = 4
       vertices(2) = 4: vertices(3) = 2
       vertices(4) = 6: vertices(5) = 4
       Set lwpolyObj = ThisDrawing.ModelSpace. _
                                                                    AddLig
htWeightPolyline(vertices)
       ZoomAll
       ' 删除多段线
```

```
        lwpolyObj.Delete
        ThisDrawing.Regen acActiveViewport
End Sub
```

3.4.3 复制对象

可以为数据库中的大多数图形对象和非图形对象创建一个拷贝。创建对象拷贝用 Clone()方法。我们可以对 Clone()方法返回的对象进行修改，然后将其添加到数据库内。通过使用 Clone()方法和 TransformBy()方法，我们可以模仿 AutoCAD 的许多修改命令。关于 TransformBy()方法的更多内容，见（§3.4.5 *变换对象*）。

除了直接创建对象的拷贝外，我们还可以使用 Clone()方法和 TransformBy()方法来偏移对象、镜像对象及阵列对象。关于复制对象的更多内容，见《AutoCAD 用户指南》中“复制、偏移或镜像对象”一节。

3.4.3.1 复制一个对象

要复制一个对象，对该对象应用 Clone()方法即可。Clone()方法创建一个新的对象，它是原始对象的副本。创建对象复本后，我们可以在将其添加到数据库前对其进行修改。如果没有变换这个新对象也没有修改它的位置，那么新对象就与原对象在相同位置上。

如果需要复制大量对象，可以将每个对象的 objectid 添加到 ObjectIdCollection 对象，然后遍历其中的每个对象。遍历每个对象时，就可以为每个对象调用 Clone()函数，然后将新对象添加或追加到数据库。

复制单个对象

下面示例新建一个圆，然后用直接复制该圆的方法创建第二个圆。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("SingleCopy")>_
Public Sub SingleCopy()
   '' 获取当前文档和数据库
   Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database
```

```
   '' 启动事务
   Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,OpenMode.ForRead)

            '' 以写模式打开块表记录模型空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec =acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                    OpenMode.ForWrite)

            ''创建圆，圆心(2,3)半径 4.25
            Dim acCirc As Circle = New Circle()
            acCirc.Center = New Point3d(2, 3, 0)
            acCirc.Radius = 4.25

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc)
            acTrans.AddNewlyCreatedDBObject(acCirc,True)

            ''创建圆的拷贝，修改其半径
            Dim acCircClone As Circle =acCirc.Clone()
            acCircClone.Radius = 1

            ''将拷贝的圆添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCircClone)
           acTrans.AddNewlyCreatedDBObject(acCircClone, True)

            '' 将新对象保存到数据库
            acTrans.Commit()
   End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
usingAutodesk.AutoCAD.Geometry;

[CommandMethod("SingleCopy")]
public static void SingleCopy()
```

```
{
   // 获取当前文档和数据库
   Document acDoc = Application.DocumentManager.MdiActiveDocument;
   Database acCurDb = acDoc.Database;

   // 启动事务
   using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
   {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                                                    OpenMode.Fo
rRead) as BlockTable;

            // 以写模式打开块表记录模型空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                                                    OpenM
ode.ForWrite) as BlockTableRecord;

            // 创建圆，圆心(2,3)半径 4.25
            Circle acCirc = new Circle();
            acCirc.Center = new Point3d(2, 3, 0);
            acCirc.Radius = 4.25;

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc);
            acTrans.AddNewlyCreatedDBObject(acCirc,true);

            // 创建圆的拷贝，修改拷贝的半径
            Circle acCircClone = acCirc.Clone() as Circle;
            acCircClone.Radius = 1;

            // 将拷贝的圆添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCircClone);
           acTrans.AddNewlyCreatedDBObject(acCircClone, true);

            // 保存新对象到数据库
            acTrans.Commit();
   }
}
```

VBA/ActiveX 代码参考

```
Sub SingleCopy()
       ’ 定义 Circle 对象
       Dim centerPoint(0 To 2) As Double
       centerPoint(0) = 2: centerPoint(1) = 3:centerPoint(2) = 0

       ’ 定义初始圆的半径
       Dim radius As Double
       radius = 4.25

       ’ 定义复制得到的圆的半径
       Dim radiusCopy As Double
       radiusCopy = 1#

       ’ 向模型空间添加圆
       Dim circleObj As AcadCircle
       Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPoint,radius)

       ’ 创建圆的拷贝
       Dim circleObjCopy As AcadCircle
       Set circleObjCopy = circleObj.Copy()
       circleObjCopy.radius = radiusCopy
End Sub
```

复制多个对象

下例使用ObjectIdCollection对象来跟踪要复制的对象。将对象的objectid添加到集合后，遍历该集合，然后用 Clone()方法创建新对象并添加到模型空间。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("MultipleCopy")>_
Public Sub MultipleCopy()
   ’’ 获取当前文档和数据库
   Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   ’’ 启动事务
```

```
   Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

            '' 以写模式打开块表记录模型空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec =acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                                        OpenM
ode.ForWrite)

            '' 创建圆，圆心(0,0,0)半径 5
            Dim acCirc1 As Circle = New Circle()
            acCirc1.Center = New Point3d(0, 0, 0)
            acCirc1.Radius = 5

            '' 添加新对象到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc1)
            acTrans.AddNewlyCreatedDBObject(acCirc1,True)

            ''创建圆，圆心(0,0,0)半径 7
            Dim acCirc2 As Circle = New Circle()
            acCirc2.Center = New Point3d(0, 0, 0)
            acCirc2.Radius = 7

            '' 添加新对象到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc2)
            acTrans.AddNewlyCreatedDBObject(acCirc2,True)

            '' 将所有要复制的对象添加到集合
            Dim acDBObjColl As DBObjectCollection =New DBObjectCollection()
            acDBObjColl.Add(acCirc1)
            acDBObjColl.Add(acCirc2)

            For Each acEnt As Entity In acDBObjColl
                    Dim acEntClone As Entity
                    acEntClone = acEnt.Clone()
                    acEntClone.ColorIndex = 1

                    '' 创建一个变换矩阵，每个复本实体向右移动 15 个单位
                   acEntClone.TransformBy(Matrix3d.Displacement(New Vector3d(15,
0, 0)))
```

```
                    '' 添加到块表记录和事务
                     acBlkTblRec.AppendEntity(acEntClone)
                    acTrans.AddNewlyCreatedDBObject(acEntClone, True)
            Next

            '' 将新对象保存到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
usingAutodesk.AutoCAD.Geometry;

[CommandMethod("MultipleCopy")]
public static void MultipleCopy()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                OpenMode.ForRead) as BlockTable;

            // 以写模式打开块表记录模型空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                    OpenMode.ForWrite) as BlockTableRecord;

            // 创建圆，圆心 0,0,0 半径 5
            Circle acCirc1 = new Circle();
            acCirc1.Center = new Point3d(0, 0, 0);
            acCirc1.Radius = 5;

            // 添加新对象到块表记录和事务
```

```
            acBlkTblRec.AppendEntity(acCirc1);
            acTrans.AddNewlyCreatedDBObject(acCirc1,true);

            // 创建圆，圆心 0,0,0 半径 7
            Circle acCirc2 = new Circle();
            acCirc2.Center = new Point3d(0, 0, 0);
            acCirc2.Radius = 7;

            // 添加新对象到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc2);
            acTrans.AddNewlyCreatedDBObject(acCirc2,true);

            // 将所有要复制的对象添加到集合
            DBObjectCollection acDBObjColl = newDBObjectCollection();
            acDBObjColl.Add(acCirc1);
            acDBObjColl.Add(acCirc2);

            foreach (Entity acEnt in acDBObjColl)
            {
                    Entity acEntClone;
                    acEntClone = acEnt.Clone() as Entity;
                    acEntClone.ColorIndex = 1;

                    // 创建一个变换矩阵，每个复本实体向右移动 15 个单位
                   acEntClone.TransformBy(Matrix3d.Displacement(new Vector3d(15,
0, 0)));

                    // 将克隆对象添加到块表记录和事务
                    acBlkTblRec.AppendEntity(acEntClone);
                   acTrans.AddNewlyCreatedDBObject(acEntClone, true);
            }

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub MultipleCopy()
       ’ 定义 Circle 对象
       Dim centerPoint(0 To 2) As Double
       centerPoint(0) = 0: centerPoint(1) = 0:centerPoint(2) = 0
```

```
        Dim radius1 As Double, radius2 As Double
        radius1 = 5#: radius2 = 7#

        ’ 在当前图形中添加 2 个圆
        Dim circleObj1 As AcadCircle, circleObj2 AsAcadCircle
        Set circleObj1 =ThisDrawing.ModelSpace.AddCircle _
                                (centerPoint, radius1)

        Set circleObj2 = ThisDrawing.ModelSpace.AddCircle_
                                (centerPoint, radius2)

        ’ 先将要复制的对象数组里
        Dim objCollection(0 To 1) As Object
        Set objCollection(0) = circleObj1
        SetobjCollection(1) = circleObj2

        ’ 在模型空间拷贝数组里的对象，返回新对象集合
        Dim retObjects As Variant
        retObjects =ThisDrawing.CopyObjects(objCollection)

        Dim ptFrom(0 To 2) As Double
        ptFrom(0) = 0: ptFrom(1) = 0: ptFrom(2) = 0

        Dim ptTo(0 To 2) As Double
        ptTo(0) = 15: ptTo(1) = 0: ptTo(2) = 0

        Dim nCount As Integer
        For nCount = 0 To UBound(retObjects)
                  Dim entObj As AcadEntity
                  Set entObj = retObjects(nCount)

                  entObj.color = acRed
                  entObj.Move ptFrom, ptTo
        Next
End Sub
```

3.4.3.2 在数据库之间复制对象

可以在两个数据库之间复制对象。Clone()方法用于在一个数据库内复制对象，而 WblockCloneObjects()方法用于从一个数据库复制对象到另一个数据库。WblockCloneObjects()方法是 Database 对象的成员。WblockCloneObjects()方法的 C#原型如下：

```
public void WblockCloneObjects(
    ObjectIdCollection identifiers,
    ObjectId id,
    IdMapping mapping,
    Autodesk.AutoCAD.DatabaseServices.DuplicateRecordCloning cloning,
    [MarshalAs(UnmanagedType.U1)] bool DeferTranslation
);
```

WblockCloneObjects()方法需要下列参数：

- ObjectIdCollection - 要复制的对象列表；
- ObjectId - 容纳复本的新父对象的 objectid；
- IdMapping - 要复制对象的当前 objectid 和新 objectid 的数据结构；
- DuplicateRecordCloning - 定义出现相同对象时如何处理；
- DeferTranslation - 控制是否需要翻译 objectid；

从一个数据库向另一个数据库复制对象

本示例先创建两个圆，然后使用 WblockCloneObjects()方法复制到新图形中。本示例还演示了在复制圆之前如何用 *acad.dwt* 创建新图形。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("CopyObjectsBetweenDatabases",CommandFlags.Session)> _
Public Sub CopyObjectsBetweenDatabases()
   Dim acObjIdColl As ObjectIdCollection = NewObjectIdCollection()

   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 锁定当前文档
   Using acLckDocCur As DocumentLock =acDoc.LockDocument()

           '' 启动事务
           Using acTrans As Transaction
=acCurDb.TransactionManager.StartTransaction()
```

```
                    '' 以读模式打开 Block 表
                    Dim acBlkTbl As BlockTable
                    acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
OpenMode.ForRead)

                    '' 以写模式打开 Block 表记录 ModelSpace
                    Dim acBlkTblRec As BlockTableRecord
                    acBlkTblRec
=acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)

                    ''创建圆，圆心(0,0,0)，半径 5
                    Dim acCirc1 As Circle = New Circle()
                    acCirc1.Center = New Point3d(0, 0, 0)
                    acCirc1.Radius = 5

                    '' 添加新对象到块表记录和事务
                    acBlkTblRec.AppendEntity(acCirc1)
                    acTrans.AddNewlyCreatedDBObject(acCirc1,True)

                    '' 创建圆，圆心(0,0,0)，半径 7
                    Dim acCirc2 As Circle = New Circle()
                    acCirc2.Center = New Point3d(0, 0, 0)
                    acCirc2.Radius = 7

                    '' 添加新对象到块表记录和事务
                    acBlkTblRec.AppendEntity(acCirc2)
                   acTrans.AddNewlyCreatedDBObject(acCirc2, True)

                    '' 将要复制的对象添加到对象集合里
                    acObjIdColl = New ObjectIdCollection()
                    acObjIdColl.Add(acCirc1.ObjectId)
                    acObjIdColl.Add(acCirc2.ObjectId)

                    '' 保存新对象到数据库
                    acTrans.Commit()
            End Using

            '' 解锁文档
    End Using

    '' 获取文档模板文件的路径
    Dim sLocalRoot As String =Application.GetSystemVariable("LOCALROOTPREFIX")
```

```
    Dim sTemplatePath As String = sLocalRoot +"Template\acad.dwt"

    '' 新建一个文档
    Dim acDocMgr As DocumentCollection =Application.DocumentManager
    Dim acNewDoc As Document =acDocMgr.Add(sTemplatePath)
    Dim acDbNewDoc As Database =acNewDoc.Database

    '' 锁定新文档
    Using acLckDoc As DocumentLock = acNewDoc.LockDocument()

            '' 启动新数据库的事务
            Using acTrans =acDbNewDoc.TransactionManager.StartTransaction()

                    '' 以读模式打开 Block 表
                    Dim acBlkTblNewDoc As BlockTable
                    acBlkTblNewDoc = acTrans.GetObject(acDbNewDoc.BlockTableId,_
                                                       OpenMode.ForRead)

                    '' 以读模式打开 Block 表记录 Modelspace
                    Dim acBlkTblRecNewDoc As BlockTableRecord
                    acBlkTblRecNewDoc =
acTrans.GetObject(acBlkTblNewDoc(BlockTableRecord.ModelSpace),_
                                          OpenMode.ForRead)

                    '' 复制对象到新数据库
                    Dim acIdMap As IdMapping = NewIdMapping()
                    acCurDb.WblockCloneObjects(acObjIdColl,acBlkTblRecNewDoc.Obj
ectId, acIdMap, DuplicateRecordCloning.Ignore, False)

                    '' 保存对象到数据库
                    acTrans.Commit()
            End Using

            '' 解锁文档
    End Using

    '' 新文档设置为当前文档
    acDocMgr.MdiActiveDocument = acNewDoc
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
```

```
usingAutodesk.AutoCAD.Geometry;

[CommandMethod("CopyObjectsBetweenDatabases",CommandFlags.Session)]
public static void CopyObjectsBetweenDatabases()
{
   ObjectIdCollection acObjIdColl = new ObjectIdCollection();

   // 获取当前文档和数据库
   Document acDoc = Application.DocumentManager.MdiActiveDocument;
   Database acCurDb = acDoc.Database;

   // 锁定当前文档
   using (DocumentLock acLckDocCur =acDoc.LockDocument())
   {
           // 启动事务
           using (Transaction acTrans
=acCurDb.TransactionManager.StartTransaction())
           {
                   //以读模式打开块表
                   BlockTable acBlkTbl;
                   acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

                   // 以写模式打开块表记录模型空间
                   BlockTableRecord acBlkTblRec;
                   acBlkTblRec
=acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                   OpenMode.ForWrite) as BlockTableRecord;

                   // 创建圆，圆心(0,0,0) 半径 5
                   Circle acCirc1 = new Circle();
                   acCirc1.Center = new Point3d(0, 0,0);
                   acCirc1.Radius = 5;

                   // 添加到块表记录和事务
                   acBlkTblRec.AppendEntity(acCirc1);
                   acTrans.AddNewlyCreatedDBObject(acCirc1, true);

                   // 创建圆，圆心(0,0,0)，半径 7
                   Circle acCirc2 = new Circle();
                   acCirc2.Center = new Point3d(0, 0,0);
                   acCirc2.Radius = 7;

                   // 添加到块表记录和事务
```

```
                    acBlkTblRec.AppendEntity(acCirc2);
                    acTrans.AddNewlyCreatedDBObject(acCirc2, true);

                    // 添加到要复制对象集合内
                    acObjIdColl = new ObjectIdCollection();
                    acObjIdColl.Add(acCirc1.ObjectId);
                    acObjIdColl.Add(acCirc2.ObjectId);

                    // 保存到数据库
                    acTrans.Commit();
            }

            // 解锁文档
    }

    // 获取图形模板路径和文件
    string sLocalRoot =Application.GetSystemVariable("LOCALROOTPREFIX") as string;
    string sTemplatePath = sLocalRoot + "Template\\acad.dwt";

    // 新建一个图形，我们将两个圆复制到这个新图形里
    DocumentCollection acDocMgr =Application.DocumentManager;
    Document acNewDoc =acDocMgr.Add(sTemplatePath);
    Database acDbNewDoc = acNewDoc.Database;

    // 锁定新文档
    using (DocumentLock acLckDoc =acNewDoc.LockDocument())
    {
            // 启动新文档的事务
            using (Transaction acTrans
=acDbNewDoc.TransactionManager.StartTransaction())
            {
                    // 以读模式打开 Block 表
                    BlockTable acBlkTblNewDoc;
                    acBlkTblNewDoc =acTrans.GetObject(acDbNewDoc.BlockTableId,
                                                    OpenMode.ForRead) as BlockTable;

                    // 以写打开块表记录模型空间
                    BlockTableRecord acBlkTblRecNewDoc;
                    acBlkTblRecNewDoc
=acTrans.GetObject(acBlkTblNewDoc[BlockTableRecord.ModelSpace],
                                    OpenMode.ForRead) as BlockTableRecord;

                    // 克隆对象到新数据库
                    IdMapping acIdMap = new IdMapping();
```

```
                    acCurDb.WblockCloneObjects(acObjIdColl,
acBlkTblRecNewDoc.ObjectId,acIdMap,
                          DuplicateRecordCloning.Ignore, false);

                    // 保存复制的对象到数据库
                    acTrans.Commit();
            }

            //解锁文档
    }

    // 将新文档设置为当前文档
    acDocMgr.MdiActiveDocument = acNewDoc;
}
```

VBA/ActiveX 代码参考

```
Sub CopyObjectsBetweenDatabases()
        Dim DOC0 As AcadDocument
        DimcircleObj1 As AcadCircle, circleObj2 As AcadCircle
        Dim centerPoint(0 To 2) As Double
        Dim radius1 As Double, radius2 As Double
        Dim objCollection(0 To 1) As Object
        Dim retObjects As Variant

        ' 定义 Circle 对象
        centerPoint(0) = 0: centerPoint(1) = 0:centerPoint(2) = 0
        radius1 = 5#: radius2 = 7#

        ' 当前图形添加 2 个圆
        Set circleObj1 =ThisDrawing.ModelSpace.AddCircle _
                                        (centerPoint, radius1)
        Set circleObj2 = ThisDrawing.ModelSpace.AddCircle_
                                        (centerPoint, radius2)

        ' 保存当前图形的指针
        Set DOC0 =ThisDrawing.Application.ActiveDocument

        ' 复制对象
        '
        ' 首先将要复制的对象保存在对象集合里
        Set objCollection(0) = circleObj1
        Set objCollection(1) = circleObj2

        ' 新建一个图形，打开模型空间
```

```
        Dim Doc1MSpace As AcadModelSpace
        Dim DOC1 As AcadDocument

        Set DOC1 = Documents.Add
        Set Doc1MSpace = DOC1.ModelSpace

        ’ 复制对象到新图形的模型空间，返回新对象集合
        retObjects =DOC0.CopyObjects(objCollection, Doc1MSpace)
End Sub
```

3.4.4 偏移对象

偏移对象就是在距原对象指定距离处创建一个新对象。可以偏移圆弧、圆、椭圆、直线、轻量级多段线、多段线、样条曲线以及构造线等。

要偏移一个对象，使用该对象的 GetOffsetCurves()方法即可。该方法需要一个表示偏移距离的正数或负数值。如果距离为负值，AutoCAD 理解为偏移产生一个“更小”的曲线（对于圆弧来说,就是偏移产生的圆弧半径比原来的圆弧半径缩短了给定偏移长度)。如果“更小”没有意义，AutoCAD 就向坐标值减小的方向偏移（WCS 坐标）。

对于多数对象，偏移操作的结果就是一个新的曲线（可能与原对象的类型不同）。例如，偏移椭圆得到的将是一个样条曲线，因为结果确实满足椭圆方程。有些情况下，偏移结果是几条曲线。正因为如此，GetOffsetCurves()方法返回的是一个 DBObjectCollection 对象，其中包含有偏移曲线创建的所有对象。需要先遍历返回的 DBObjectCollection 对象，获得创建的每个对象，然后逐个添加到图形数据库中。

偏移多段线

本例创建一个轻量级多段线，然后对其进行偏移操作。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("OffsetObject")>_
   Public Sub OffsetObject()
   ’’ 获取当前文档和数据库
```

```
    Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, _
                                        OpenMode.ForRead)

            '' 以写模式打开块表记录模型空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec =acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                        OpenMode.ForWrite)

            '' 创建多段线
            Dim acPoly As Polyline = New Polyline()
            acPoly.AddVertexAt(0, New Point2d(1, 1),0, 0, 0)
            acPoly.AddVertexAt(1, New Point2d(1, 2),0, 0, 0)
            acPoly.AddVertexAt(2, New Point2d(2, 2),0, 0, 0)
            acPoly.AddVertexAt(3, New Point2d(3, 2),0, 0, 0)
            acPoly.AddVertexAt(4, New Point2d(4, 4),0, 0, 0)
            acPoly.AddVertexAt(5, New Point2d(4, 1),0, 0, 0)

            '' 添加新对象到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly)
            acTrans.AddNewlyCreatedDBObject(acPoly,True)

            '' 偏移多段线给定距离
            Dim acDbObjColl As DBObjectCollection =acPoly.GetOffsetCurves(0.25)

            '' 遍历得到的新对象
            For Each acEnt As Entity In acDbObjColl
                    '' 添加每个对象
                    acBlkTblRec.AppendEntity(acEnt)
                   acTrans.AddNewlyCreatedDBObject(acEnt, True)
            Next

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("OffsetObject")]
public static void OffsetObject()
{
   // 获取当前文档和数据库
   Document acDoc =Application.DocumentManager.MdiActiveDocument;
   Database acCurDb = acDoc.Database;

   // 启动事务
   using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
   {
           // 以读模式打开 Block 表
           BlockTable acBlkTbl;
           acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                       OpenMode.ForRead) as BlockTable;

           // 以写模式打开块表记录模型空间
           BlockTableRecord acBlkTblRec;
           acBlkTblRec
=acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                              OpenMode.ForWrite) as BlockTableRecord;

           // 创建多段线
           Polyline acPoly = new Polyline();
           acPoly.AddVertexAt(0, new Point2d(1, 1),0, 0, 0);
           acPoly.AddVertexAt(1, new Point2d(1, 2),0, 0, 0);
           acPoly.AddVertexAt(2, new Point2d(2, 2),0, 0, 0);
           acPoly.AddVertexAt(3, new Point2d(3, 2),0, 0, 0);
           acPoly.AddVertexAt(4, new Point2d(4, 4),0, 0, 0);
           acPoly.AddVertexAt(5, new Point2d(4, 1),0, 0, 0);

           // 添加新对象到块表记录和事务
           acBlkTblRec.AppendEntity(acPoly);
           acTrans.AddNewlyCreatedDBObject(acPoly,true);

           // 偏移距离 0.25
           DBObjectCollection acDbObjColl =acPoly.GetOffsetCurves(0.25);

           // 遍历得到的新对象
           foreach (Entity acEnt in acDbObjColl)
```

```
            {
                    // 添加每个对象
                    acBlkTblRec.AppendEntity(acEnt);
                   acTrans.AddNewlyCreatedDBObject(acEnt, true);
            }

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub OffsetObject()
       ' 创建多段线
       Dim plineObj As AcadLWPolyline
       Dim points(0 To 11) As Double
       points(0) = 1: points(1) = 1
       points(2) = 1: points(3) = 2
       points(4) = 2: points(5) = 2
       points(6) = 3: points(7) = 2
       points(8) = 4: points(9) = 4
       points(10) = 4: points(11) = 1
       Set plineObj = ThisDrawing.ModelSpace. _
                                 AddLightWeightPolyline(points)
       plineObj.Closed = True
       ZoomAll

       ' 偏移多段线
       Dim offsetObj As Variant
       offsetObj = plineObj.Offset(0.25)

       ZoomAll
End Sub
```

3.4.5 变换对象

我们使用 Matrix3d 对象表示的 4×4 变换矩阵和 TransformBy()方法来对图形中的对象进行移动、缩放、旋转和镜像等变换操作。我们还可以使用 GetTransformedCopy()方法创建实体复本，然后对复本进行变换操作。Matrix3d 对象是在 Geometry 命名空间定义的。

矩阵的前三列确定缩放和旋转，矩阵的第四列是个平移矢量。下表为变换矩阵的配置示范，其中 R 表示旋转，T 表示平移：

变换矩阵配置			
R00	R01	R02	T0
R10	R11	R12	T1
R20	R21	R22	T2
0	0	0	1

变换一个对象，首先初始化 Matrix3d 对象。我们可以使用一个双精度实数数组或者一个代表世界坐标系或用户坐标系的矩阵来初始化变换矩阵。完成初始化后，我们就可以使用 Matrix3d 对象的函数修改矩阵的缩放变换、旋转变换或平移变换。

完成变换矩阵后，使用 TransformBy()方法将变换矩阵应用在对象上就可以了。下面这行代码演示在对象 acObj 上应用矩阵 dMatrix：

VB.NET

```
acObj.TransformBy(dMatrix)
```

C#

```
acObj.TransformBy(dMatrix);
```

旋转矩阵示例

下面演示用一维数组定义变换矩阵，赋值给变量 dMatrix，让实体绕点（0, 0, 0）旋转 90 度。

旋转矩阵： 绕点（0, 0, 0）旋转 90 度			
0	-1	0	0
1	0	0	0
0	0	1	0
0	0	0	1

VB.NET

使用数据数组初始化变换矩阵，数组包含旋转对象 90 度的数据。

```
Dim dMatrix(0 To 15) As Double

dMatrix(0) = 0.0
dMatrix(1) = -1.0
```

```
dMatrix(2) = 0.0
dMatrix(3) = 0.0
dMatrix(4) = 1.0
dMatrix(5) = 0.0
dMatrix(6) = 0.0
dMatrix(7) = 0.0
dMatrix(8) = 0.0
dMatrix(9) = 0.0
dMatrix(10) = 1.0
dMatrix(11) = 0.0
dMatrix(12) = 0.0
dMatrix(13) = 0.0
dMatrix(14) = 0.0
dMatrix(15) = 1.0
Dim acMat3d As Matrix3d = New Matrix3d(dMatrix)
```

不用数据数组初始化变换矩阵，而是用 Rotation()函数返回一个实现对象旋转 90 度的变换矩阵。

```
Dim acMat3d As Matrix3d = New Matrix3d()
Matrix3d.Rotation(Math.PI/2, curUCS.Zaxis, New Point3d(0, 0, 0))
```

C#

使用数据数组初始化变换矩阵，数组包含旋转对象 90 度的数据。

```
double[] dMatrix = new double[16];

dMatrix[0] = 0.0;
dMatrix[1] = -1.0;
dMatrix[2] = 0.0;
dMatrix[3] = 0.0;
dMatrix[4] = 1.0;
dMatrix[5] = 0.0;
dMatrix[6] = 0.0;
dMatrix[7] = 0.0;
dMatrix[8] = 0.0;
dMatrix[9] = 0.0;
dMatrix[10] = 1.0;
dMatrix[11] = 0.0;
dMatrix[12] = 0.0;
dMatrix[13] = 0.0;
dMatrix[14] = 0.0;
dMatrix[15] = 1.0;
```

```
Matrix3d acMat3d = new Matrix3d(dMatrix);
```

不用数据数组初始化变换矩阵，而是用 Rotation 函数返回一个实现对象旋转 90 度的变换矩阵。

```
Matrix3d acMat3d = new Matrix3d();
acMat3d = Matrix3d.Rotation(Math.PI/2, curUCS.Zaxis, new Point3d(0, 0, 0));
```

变换矩阵的其他示例

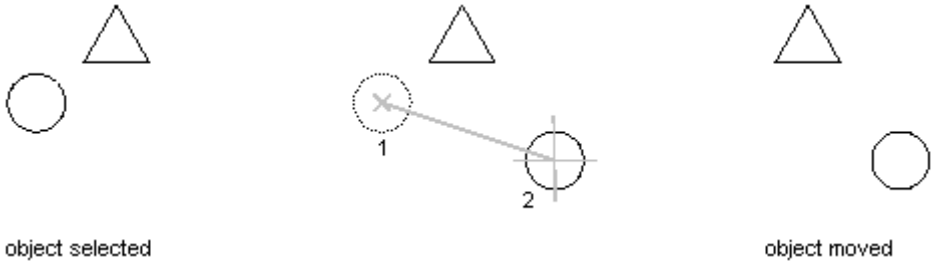
下面为更多的变换矩阵示例

旋转矩阵：围绕点(5, 5, 0)旋转 45 度			
0.707107	-0.707107	0	5
0.707107	0.707107	0	-2.071068
0	0	1	0
0	0	0	1
平移矩阵：沿矢量(10, 10, 0)移动实体			
1	0	0	10
0	1	0	10
0	0	1	0
0	0	0	1
放大矩阵：以(0,0,0)为基点 x、y、z 方向均放大 10 倍			
10	0	0	0
0	10	0	0
0	0	10	0
0	0	0	1
放大矩阵：以(2,2,0)为基点 x、y、z 方向均放大 10 倍			
10	0	0	-18
0	10	0	-18
0	0	10	0
0	0	0	1

3.4.5.1 移动对象

我们可以沿给定矢量移动所有的图像对象和属性参照对象。

移动对象使用变换矩阵的 Displacement()函数，该函数要求输入一个 Vector3d 对象。如果不知道所需的矢量，可以创建一个 Point3d 对象然后使用 GetVectorTo()方法返回两点间的矢量。位移矢量表示将一个对象移动多远及往哪个方向移动。



关于移动对象的更多内容见《AutoCAD 用户指南》中“移动对象”一节。

沿矢量移动一个圆

本例创建一个圆，然后将这个圆沿 X 轴移动 2 个单位。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("MoveObject")> _
Public Sub MoveObject()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

           '' 以读模式打开 Block 表
           Dim acBlkTbl As BlockTable
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                        OpenMode.ForRead)

           '' 以写模式打开 Block 表记录 Model 空间
```

```
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                    OpenMode.ForWrite)

            '' 创建圆
            Dim acCirc As Circle = New Circle()
            acCirc.Center = New Point3d(2, 2, 0)
            acCirc.Radius = 0.5

            '' 创建一个矩阵，沿(0,0,0)到(2,0,0)的矢量移动圆
            Dim acPt3d As Point3d = New Point3d(0, 0, 0)
            Dim acVec3d As Vector3d = acPt3d.GetVectorTo(New Point3d(2, 0, 0))

            acCirc.TransformBy(Matrix3d.Displacement(acVec3d))

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc)
            acTrans.AddNewlyCreatedDBObject(acCirc, True)

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("MoveObject")]
public static void MoveObject()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表以读打开块表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
```

```
                                        OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

            // 创建圆，圆心 2,2 半径 0.5
            Circle acCirc = new Circle();
            acCirc.Center = new Point3d(2, 2, 0);
            acCirc.Radius = 0.5;

            // 创建一个矩阵，使用(0,0,0)到(2,0,0)的矢量移动圆
            Point3d acPt3d = new Point3d(0, 0, 0);
            Vector3d acVec3d = acPt3d.GetVectorTo(new Point3d(2, 0, 0));

            acCirc.TransformBy(Matrix3d.Displacement(acVec3d));

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc);
            acTrans.AddNewlyCreatedDBObject(acCirc, true);

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

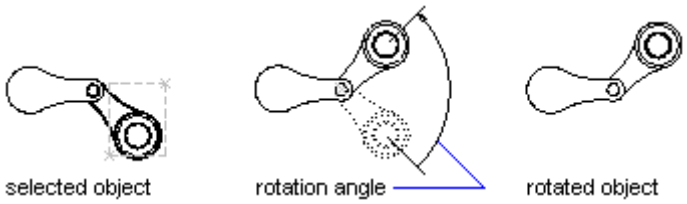
```
Sub MoveObject()
        ' 创建圆
        Dim circleObj As AcadCircle
        Dim center(0 To 2) As Double
        Dim radius As Double
        center(0) = 2#: center(1) = 2#: center(2) = 0#
        radius = 0.5
        Set circleObj = ThisDrawing.ModelSpace. _
                                    AddCircle(center, radius)
        ZoomAll

        ' 定义构成移动矢量的点
        ' 移动矢量将圆沿 X 轴移动 2 个单位
        Dim point1(0 To 2) As Double
        Dim point2(0 To 2) As Double
```

```
        point1(0) = 0: point1(1) = 0: point1(2) = 0
        point2(0) = 2: point2(1) = 0: point2(2) = 0

        ' 移动圆
        circleObj.Move point1, point2
        circleObj.Update
End Sub
```

3.4.5.2 旋转对象

我们可以旋转所有的图像对象和属性参照对象。

旋转对象使用变换矩阵的 Rotation()函数，该函数要求输入用弧度表示的旋转角度、旋转轴和旋转基点。旋转轴用 Vector3d 对象表示，旋转基点用 Point3d 对象表示。旋转角度表示相对于当前位置将对象围绕基点旋转多远。



关于旋转对象的更多内容见《AutoCAD 用户指南》中“旋转对象”一节。

绕基点旋转多段线

本例创建一个闭合轻量级多段线，然后将其绕点(4, 4.25, 0)旋转 45 度。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("RotateObject")> _
Public Sub RotateObject()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database
```

```
    '' 启动事务
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,OpenMode.ForRead)

            '' 以写模式打开 Block 表记录 Model 空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                            OpenMode.ForWrite)

            '' 创建轻量级多段线
            Dim acPoly As Polyline = New Polyline()
            acPoly.AddVertexAt(0, New Point2d(1, 2), 0, 0, 0)
            acPoly.AddVertexAt(1, New Point2d(1, 3), 0, 0, 0)
            acPoly.AddVertexAt(2, New Point2d(2, 3), 0, 0, 0)
            acPoly.AddVertexAt(3, New Point2d(3, 3), 0, 0, 0)
            acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
            acPoly.AddVertexAt(5, New Point2d(4, 2), 0, 0, 0)

            '' 闭合多段线
            acPoly.Closed = True

            Dim curUCSMatrix As Matrix3d =
acDoc.Editor.CurrentUserCoordinateSystem
            Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d

            '' 绕当前 UCS 的 Z 轴旋转多段线 45 度
            '' 旋转基点为(4,4.25,0)
            acPoly.TransformBy(Matrix3d.Rotation(0.7854, _
                               curUCS.Zaxis, New Point3d(4, 4.25, 0)))

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly)
            acTrans.AddNewlyCreatedDBObject(acPoly, True)

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("RotateObject")]
public static void RotateObject()
{
   // 获取当前文档和数据库
   Document acDoc = Application.DocumentManager.MdiActiveDocument;
   Database acCurDb = acDoc.Database;

   // 启动事务
   using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
   {
           // 以读模式打开 Block 表
           BlockTable acBlkTbl;
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                        OpenMode.ForRead) as BlockTable;

           // 以写模式打开 Block 表记录 Model 空间
           BlockTableRecord acBlkTblRec;
           acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                       OpenMode.ForWrite) as BlockTableRecord;

           // 创建轻量级多段线
           Polyline acPoly = new Polyline();
           acPoly.AddVertexAt(0, new Point2d(1, 2), 0, 0, 0);
           acPoly.AddVertexAt(1, new Point2d(1, 3), 0, 0, 0);
           acPoly.AddVertexAt(2, new Point2d(2, 3), 0, 0, 0);
           acPoly.AddVertexAt(3, new Point2d(3, 3), 0, 0, 0);
           acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
           acPoly.AddVertexAt(5, new Point2d(4, 2), 0, 0, 0);

           // 闭合多段线
           acPoly.Closed = true;

           Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
           CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;

           //绕当前 UCS 的 Z 轴将多段线旋转 45 度，基点(4,4.25,0)
           acPoly.TransformBy(Matrix3d.Rotation(0.7854,
                              curUCS.Zaxis, new Point3d(4, 4.25, 0)));
```

```
            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly);
            acTrans.AddNewlyCreatedDBObject(acPoly, true);

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

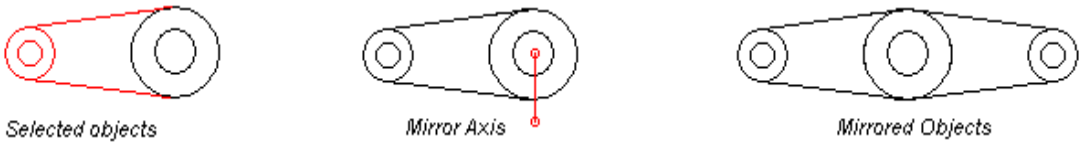
VBA/ActiveX 代码参考

```
Sub RotateObject()
        '创建多段线
        Dim plineObj As AcadLWPolyline
        Dim points(0 To 11) As Double
        points(0) = 1: points(1) = 2
        points(2) = 1: points(3) = 3
        points(4) = 2: points(5) = 3
        points(6) = 3: points(7) = 3
        points(8) = 4: points(9) = 4
        points(10) = 4: points(11) = 2
        Set plineObj = ThisDrawing.ModelSpace. _
                                        AddLightWeightPolyline(points)
        plineObj.Closed = True
        ZoomAll

        ' 定义旋转基点和旋转角度
        Dim basePoint(0 To 2) As Double
        Dim rotationAngle As Double
        basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
        rotationAngle = 0.7853981     ' 45 degrees

        ' 旋转多段线
        plineObj.Rotate basePoint, rotationAngle
        plineObj.Update
End Sub
```

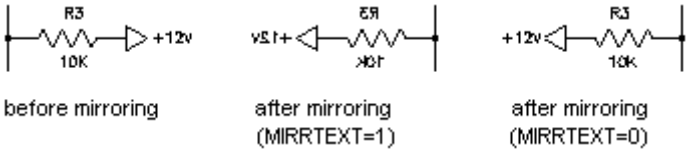
3.4.5.3 镜像对象

镜像就是沿着坐标轴或镜像线将对象翻过来。我们可以镜像所有图像对象。

镜像对象使用变换矩阵的 Mirroring()函数，该函数要求输入 Point3d 对象、Plane 对象或 Line3d 对象来确定镜像线。由于镜像是通过变换矩阵实现的，因此并没有创建新对象。如果需要保留原来对象，需要先创建对象的复本然后再镜像。



使用系统变量 MIRRTEXT 来管理文字对象的翻转属性。MIRRTEXT 系统变量的默认设置为开(1)，结果是文字对象和其他任何对象一样被翻转过来。MIRRTEXT 设置为关（0）时，文字不被翻转。可以使用 GetSystemVariable()方法和 SetSystemVariable()方法查询和设置 MIRRTEXT 变量。



镜像前　　镜像后（MIRRTEXT=1）镜像后（MIRRTEXT=0）

我们可以镜像图纸空间的视口对象，虽然这样做对模型空间的视图和对象没有作用。

关于镜像对象的更多内容见《AutoCAD 用户指南》中“复制、偏移或镜像对象”一节。

沿轴线镜像一个多段线

本例创建一个轻量级多段线，然后沿轴线对其进行镜像。得到的新对象改为蓝色。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("MirrorObject")> _
Public Sub MirrorObject()
```

```
'' 获取当前文档和数据库
Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
Dim acCurDb As Database = acDoc.Database

'' 启动事务
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' 以读模式打开 Block 表
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                             OpenMode.ForRead)

        '' 以写模式打开 Block 表记录 Model 空间
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                             OpenMode.ForWrite)

        '' 创建一个轻量级多段线
        Dim acPoly As Polyline = New Polyline()
        acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
        acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
        acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
        acPoly.AddVertexAt(3, New Point2d(3, 2), 0, 0, 0)
        acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
        acPoly.AddVertexAt(5, New Point2d(4, 1), 0, 0, 0)

        '' 设置顶点 1 到顶点 2 间的多段线段的凸度为-2
        acPoly.SetBulgeAt(1, -2)

        '' 闭合多段线
        acPoly.Closed = True

        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acPoly)
        acTrans.AddNewlyCreatedDBObject(acPoly, True)

        '' 创建原多段线的复本
        Dim acPolyMirCopy As Polyline = acPoly.Clone()
        acPolyMirCopy.ColorIndex = 5

        '' 定义镜像线
        Dim acPtFrom As Point3d = New Point3d(0, 4.25, 0)
        Dim acPtTo As Point3d = New Point3d(4, 4.25, 0)
```

```
            Dim acLine3d As Line3d = New Line3d(acPtFrom, acPtTo)

            ''沿 X 轴方向镜像多段线
            acPolyMirCopy.TransformBy(Matrix3d.Mirroring(acLine3d))

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPolyMirCopy)
            acTrans.AddNewlyCreatedDBObject(acPolyMirCopy, True)

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("MirrorObject")]
public static void MirrorObject()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                         OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

            // 创建一个轻量级多段线
            Polyline acPoly = new Polyline();
            acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
```

```
            acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
            acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
            acPoly.AddVertexAt(3, new Point2d(3, 2), 0, 0, 0);
            acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
            acPoly.AddVertexAt(5, new Point2d(4, 1), 0, 0, 0);

            //设置顶点 1 到顶点 2 间的多段线段的凸度为-2
            acPoly.SetBulgeAt(1, -2);

            // 闭合多段线
            acPoly.Closed = true;

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly);
            acTrans.AddNewlyCreatedDBObject(acPoly, true);

            // 创建原多段线的复本
            Polyline acPolyMirCopy = acPoly.Clone() as Polyline;
            acPolyMirCopy.ColorIndex = 5;

            // 定义镜像线
            Point3d acPtFrom = new Point3d(0, 4.25, 0);
            Point3d acPtTo = new Point3d(4, 4.25, 0);
            Line3d acLine3d = new Line3d(acPtFrom, acPtTo);

            // 沿 X 轴方向翻转多段线
            acPolyMirCopy.TransformBy(Matrix3d.Mirroring(acLine3d));

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPolyMirCopy);
            acTrans.AddNewlyCreatedDBObject(acPolyMirCopy, true);

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

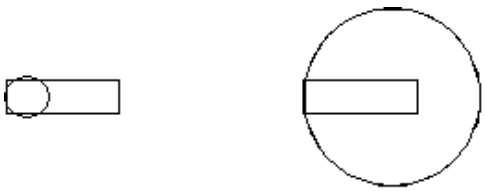
```
Sub MirrorObject()
        ' 创建多段线
        Dim plineObj As AcadLWPolyline
        Dim points(0 To 11) As Double
        points(0) = 1: points(1) = 1
        points(2) = 1: points(3) = 2
```

```
        points(4) = 2: points(5) = 2
        points(6) = 3: points(7) = 2
        points(8) = 4: points(9) = 4
        points(10) = 4: points(11) = 1
        Set plineObj = ThisDrawing.ModelSpace. _
                                AddLightWeightPolyline(points)
        plineObj.SetBulge 1, -2
        plineObj.Closed = True
        ZoomAll
        ' 定义镜像轴
        Dim point1(0 To 2) As Double
        Dim point2(0 To 2) As Double
        point1(0) = 0: point1(1) = 4.25: point1(2) = 0
        point2(0) = 4: point2(1) = 4.25: point2(2) = 0
        ' 镜像多段线
        Dim mirrorObj As AcadLWPolyline
        Set mirrorObj = plineObj.Mirror(point1, point2)
        mirrorObj.color = acBlue
        ZoomAll
End Sub
```

3.4.5.4 缩放对象

我们通过指定一个基准点和基于当前绘图单位的缩放系数来缩放对象。我们可以缩放任何对象，包括属性参照对象。

缩放对象使用变换矩阵的 Scaling()函数，该函数需要输入一个数值作为对象的缩放系数，还需要输入一个 Point3d 对象作为缩放操作的基点。Scaling()函数在 *X*、*Y*和 *Z*轴向上将对象缩放相同的倍数。对象的尺寸乘以缩放系数，缩放系数大于 1 则放大对象，缩放系数小于 1 则缩小对象。



注： 如果要采用不一致的缩放系数缩放对象，需使用相应的数据数组初始化变换矩阵，然后调用对象的 TransformBy()方法。详见前面（§3.4.5 *变换对象*）的内容

关于缩放对象的更多内容，见《AutoCAD 用户指南》中“更改对象的形状和大小”一节。

修改多段线的大小

本例创建一个闭合多段线，然后以（4, 4.25, 0）为基点将多段线缩小 0.5 倍。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("ScaleObject")> _
Public Sub ScaleObject()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

           '' 以读模式打开 Block 表
           Dim acBlkTbl As BlockTable
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                OpenMode.ForRead)

           '' 以写模式打开 Block 表记录 Model 空间
           Dim acBlkTblRec As BlockTableRecord
           acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                   OpenMode.ForWrite)

           '' 创建多段线
           Dim acPoly As Polyline = New Polyline()
           acPoly.AddVertexAt(0, New Point2d(1, 2), 0, 0, 0)
           acPoly.AddVertexAt(1, New Point2d(1, 3), 0, 0, 0)
           acPoly.AddVertexAt(2, New Point2d(2, 3), 0, 0, 0)
```

```
            acPoly.AddVertexAt(3, New Point2d(3, 3), 0, 0, 0)
            acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
            acPoly.AddVertexAt(5, New Point2d(4, 2), 0, 0, 0)

            '' 闭合多段线
            acPoly.Closed = True

            '' 缩小对象，系数 0.5，基点(4,4.25,0)
            acPoly.TransformBy(Matrix3d.Scaling(0.5, New Point3d(4, 4.25, 0)))

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly)
            acTrans.AddNewlyCreatedDBObject(acPoly, True)
            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ScaleObject")]
public static void ScaleObject()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                         OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
```

```
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                              OpenMode.ForWrite) as BlockTableRecord;

            // 创建多段线
            Polyline acPoly = new Polyline();
            acPoly.AddVertexAt(0, new Point2d(1, 2), 0, 0, 0);
            acPoly.AddVertexAt(1, new Point2d(1, 3), 0, 0, 0);
            acPoly.AddVertexAt(2, new Point2d(2, 3), 0, 0, 0);
            acPoly.AddVertexAt(3, new Point2d(3, 3), 0, 0, 0);
            acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
            acPoly.AddVertexAt(5, new Point2d(4, 2), 0, 0, 0);

            // 闭合多段线
            acPoly.Closed = true;

            // 缩小对象，系数 0.5，基点(4,4.25,0)
            acPoly.TransformBy(Matrix3d.Scaling(0.5, new Point3d(4, 4.25, 0)));

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly);
            acTrans.AddNewlyCreatedDBObject(acPoly, true);

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

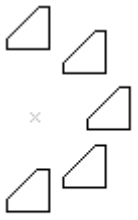
```
Sub ScaleObject()
       ' 创建多段线
       Dim plineObj As AcadLWPolyline
       Dim points(0 To 11) As Double
       points(0) = 1: points(1) = 2
       points(2) = 1: points(3) = 3
       points(4) = 2: points(5) = 3
       points(6) = 3: points(7) = 3
       points(8) = 4: points(9) = 4
       points(10) = 4: points(11) = 2
       Set plineObj = ThisDrawing.ModelSpace. _
                                       AddLightWeightPolyline(points)
       plineObj.Closed = True
       ZoomAll
```

```
        ’ 定义缩放参数
        Dim basePoint(0 To 2) As Double
        Dim scalefactor As Double
        basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
        scalefactor = 0.5
        ’ 缩小多段线
        plineObj.ScaleEntity basePoint, scalefactor
        plineObj.Update
End Sub
```

3.4.6 阵列对象

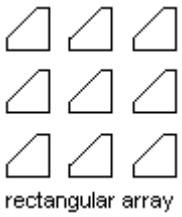
我们可以创建对象的环形阵列或矩形阵列。对象阵列不是使用一组专门的函数创建的，而是通过复制对象然后使用变换矩阵旋转和移动对象复本等组合动作创建的。下面简述一下每种阵列类型的基本逻辑：

- **环形阵列** 围绕基点按一个角度复制并移动对象得到的阵列是环形阵列。对象到阵列基点的距离用来计算所创建的每个复本的位置。移动复本对象之后，还可以绕基点旋转对象。每创建一个复本，就需将其追加到块表记录。



图示 - 180 度环形阵列，不旋转对象

- **矩形阵列** 基于所需行数和列数复制对象得到的阵列是矩形阵列。复本对象间的距离基于给定的行列间距。首先复制一定数量的原件创建第一行或列，然后基于第一行或列创建其他的行或列。每创建一个复本，就需将其追加到块表记录。



图示 - 3×3 矩形阵列

创建环形阵列

本示例代码先创建一个圆，然后实现该圆的环形阵列，创建 4 个圆绕点(4, 4, 0) 围成 180 度。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

   Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                                  pPt.Y + dDist * Math.Sin(dAng))
End Function

<CommandMethod("PolarArrayObject")> _
Public Sub PolarArrayObject()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

           '' 以读模式打开块表表记录
           Dim acBlkTbl As BlockTable
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                OpenMode.ForRead)

           '' 以写模式打开 Block 表记录 Model 空间
```

```
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                    OpenMode.ForWrite)

            '' 创建圆，圆心(2,2)，半径 1
            Dim acCirc As Circle = New Circle()
            acCirc.Center = New Point3d(2, 2, 0)
            acCirc.Radius = 1

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc)
            acTrans.AddNewlyCreatedDBObject(acCirc, True)

            '' 创建 4 个对象的 180 度环形阵列
            Dim nCount As Integer = 1

            '' 60 度的弧度值
            Dim dAng As Double = 1.0472

            '' 阵列基点(4,4,0)
            Dim acPt2dArrayBase As Point2d = New Point2d(4, 4)

            While (nCount < 4)
                    Dim acEntClone As Entity = acCirc.Clone()

                    Dim acExts As Extents3d
                    Dim acPtObjBase As Point2d

                    '' 通常情况下，使用对象范围的左上角作为要阵列对象上的点，除非
                    '' 要阵列对象是类似圆这样的对象（使用圆心）
                    Dim acCircArrObj As Circle = acEntClone

                    If IsDBNull(acCircArrObj) = False Then
                            acPtObjBase = New Point2d(acCircArrObj.Center.X, _
                                                    acCircArrObj.Center.Y)
                    Else
                            acExts = acEntClone.Bounds.GetValueOrDefault()
                            acPtObjBase = New Point2d(acExts.MinPoint.X, _
                                                    acExts.MaxPoint.Y)
                    End If

                    Dim dDist As Double =
acPt2dArrayBase.GetDistanceTo(acPtObjBase)
```

```
                    Dim dAngFromX As Double =
acPt2dArrayBase.GetVectorTo(acPtObjBase).Angle

                    Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                          (nCount * dAng) + dAngFromX, dDist)

                    Dim acVec2d As Vector2d = acPtObjBase.GetVectorTo(acPt2dTo)
                    Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
                    acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

                    ''下列代码演示怎样旋转每个对象，就像 Array 命令所做的那样
                    'acExts = acEntClone.Bounds.GetValueOrDefault()
                    'acPtObjBase = New Point2d(acExts.MinPoint.X, _
                    ' acExts.MaxPoint.Y)
                    '
                    ''围绕左上范围点旋转对象
                    'Dim curUCSMatrix As Matrix3d =
acDoc.Editor.CurrentUserCoordinateSystem
                    'Dim curUCS As CoordinateSystem3d =
curUCSMatrix.CoordinateSystem3d
                    'acEntClone.TransformBy(Matrix3d.Rotation(nCount * dAng, _
                    ' curUCS.Zaxis, _
                    ' New Point3d(acPtObjBase.X, _
                    ' acPtObjBase.Y, 0)))

                    acBlkTblRec.AppendEntity(acEntClone)
                    acTrans.AddNewlyCreatedDBObject(acEntClone, True)

                    nCount = nCount + 1
            End While

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
```

```
{
   return new Point2d(pPt.X + dDist * Math.Cos(dAng),
                          pPt.Y + dDist * Math.Sin(dAng));
}

[CommandMethod("PolarArrayObject")]
public static void PolarArrayObject()
{
   // 获取当前文档和数据库
   Document acDoc = Application.DocumentManager.MdiActiveDocument;
   Database acCurDb = acDoc.Database;

   // 启动事务
   using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
   {
            // 以读模式打开块表表记录
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                                                    OpenMode.F
orRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                 OpenMode.ForWrite) as BlockTableRecord;

            // 创建圆，圆心(2,2)半径 1
            Circle acCirc = new Circle();
            acCirc.Center = new Point3d(2, 2, 0);
            acCirc.Radius = 1;

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc);
            acTrans.AddNewlyCreatedDBObject(acCirc, true);

            //创建 4 个对象的 180 度环形阵列
            int nCount = 1;

            // 定义 60 度弧度值
            double dAng = 1.0472;

            // 定义阵列基点(4,4,0)
            Point2d acPt2dArrayBase = new Point2d(4, 4);
```

```
            while (nCount < 4)
            {
                    Entity acEntClone = acCirc.Clone() as Entity;

                    Extents3d acExts;
                    Point2d acPtObjBase;

                    // 通常情况下，使用对象范围的左上角作为要阵列对象上的点，除非
                    // 要阵列对象是类似圆这样的对象（使用圆心）
                    Circle acCircArrObj = acEntClone as Circle;

                    if (acCircArrObj != null)//是圆
                    {
                            acPtObjBase = new Point2d(acCircArrObj.Center.X,
                                                      acCircArrObj.Center.Y);
                    }
                    Else//不是圆
                    {
                            acExts = acEntClone.Bounds.GetValueOrDefault();
                            acPtObjBase = new Point2d(acExts.MinPoint.X,
                                                       acExts.MaxPoint.Y);
                    }

                    double dDist = acPt2dArrayBase.GetDistanceTo(acPtObjBase);
                    double dAngFromX =
acPt2dArrayBase.GetVectorTo(acPtObjBase).Angle;

                    Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                           (nCount * dAng) + dAngFromX, dDist);

                    Vector2d acVec2d = acPtObjBase.GetVectorTo(acPt2dTo);
                    Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
                    acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

                    /*
                    // 下列代码演示怎样旋转每个对象，就像 Array 命令所做的那样
                    acExts = acEntClone.Bounds.GetValueOrDefault();
                    acPtObjBase = new Point2d(acExts.MinPoint.X,
                                              acExts.MaxPoint.Y);

                    // 围绕左上范围点旋转对象
                    Matrix3d curUCSMatrix =
acDoc.Editor.CurrentUserCoordinateSystem;
```

```
                    CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;
                    acEntClone.TransformBy(Matrix3d.Rotation(nCount * dAng,
                                                curUCS.Zaxis,
                                          new Point3d(acPtObjBase.X, acPtObjBase.Y, 0)));
                    */

  // 添加新对象到块表记录和事务
                    acBlkTblRec.AppendEntity(acEntClone);
                    acTrans.AddNewlyCreatedDBObject(acEntClone, true);

                    nCount = nCount + 1;
            }

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub PolarArrayObject()
        ' 创建圆
        Dim circleObj As AcadCircle
        Dim center(0 To 2) As Double
        Dim radius As Double
        center(0) = 2#: center(1) = 2#: center(2) = 0#
        radius = 1
        Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)
        ZoomAll

        ' 定义环形阵列参数
        Dim noOfObjects As Integer
        Dim angleToFill As Double
        Dim basePnt(0 To 2) As Double
        noOfObjects = 4
        angleToFill = 3.14  ' 180°
        basePnt(0) = 4#: basePnt(1) = 4#: basePnt(2) = 0#

        ' 创建环形阵列
        Dim retObj As Variant
        retObj = circleObj.ArrayPolar (noOfObjects, angleToFill, basePnt)

        ZoomAll
End Sub
```

创建矩形阵列

本示例代码创建一个圆并实现该圆的五行五列矩形阵列。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                     ByVal dDist As Double)

   Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                                        pPt.Y + dDist * Math.Sin(dAng))
End Function


<CommandMethod("RectangularArrayObject")> _
Public Sub RectangularArrayObject()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

           '' 以读模式打开块表表记录
           Dim acBlkTbl As BlockTable
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                OpenMode.ForRead)

           '' 以写模式打开 Block 表记录 Model 空间
           Dim acBlkTblRec As BlockTableRecord
           acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                 OpenMode.ForWrite)

           '' 创建 Circle 对象
           Dim acCirc As Circle = New Circle()
```

```
            acCirc.Center = New Point3d(2, 2, 0)
            acCirc.Radius = 0.5

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acCirc)
            acTrans.AddNewlyCreatedDBObject(acCirc, True)

            '' 创建 5×5 矩形阵列
            Dim nRows As Integer = 5
            Dim nColumns As Integer = 5

            '' 设置行列间距及阵列角度
            Dim dRowOffset As Double = 1
            Dim dColumnOffset As Double = 1
            Dim dArrayAng As Double = 0

            '' 获取当前 UCS 坐标系的 X 轴的角度
            Dim curUCSMatrix As Matrix3d =
acDoc.Editor.CurrentUserCoordinateSystem
            Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d
            Dim acVec2dAng As Vector2d = New Vector2d(curUCS.Xaxis.X, _
                                                      curUCS.Xaxis.Y)

            ''如果 UCS 坐标系是旋转的，相应地调整阵列的角度
            dArrayAng = dArrayAng + acVec2dAng.Angle

            '' 使用对象范围的左上角作为阵列的基点
            Dim acExts As Extents3d = acCirc.Bounds.GetValueOrDefault()
            Dim acPt2dArrayBase As Point2d = New Point2d(acExts.MinPoint.X, _
                                                 acExts.MaxPoint.Y)

            '' 跟踪每列创建的对象
            Dim acDBObjCollCols As DBObjectCollection = New DBObjectCollection()
            acDBObjCollCols.Add(acCirc)

            ''  创建第一行的对象（个数等于列数）
            Dim nColumnsCount As Integer = 1
            While (nColumns > nColumnsCount)
                    Dim acEntClone As Entity = acCirc.Clone()
                    acDBObjCollCols.Add(acEntClone)

                    '' 计算新位置
                    Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                  dArrayAng, dColumnOffset * nColumnsCount)
```

```
                    Dim acVec2d As Vector2d =
acPt2dArrayBase.GetVectorTo(acPt2dTo)
                    Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
                    acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

                    acBlkTblRec.AppendEntity(acEntClone)
                    acTrans.AddNewlyCreatedDBObject(acEntClone, True)

                    nColumnsCount = nColumnsCount + 1
            End While

            '' 设置 90° 的弧度值
            Dim dAng As Double = Math.PI / 2

            '' 记录为每行列创建的对象
            Dim acDBObjCollLvls As DBObjectCollection = New DBObjectCollection()

            For Each acObj As DBObject In acDBObjCollCols
                    acDBObjCollLvls.Add(acObj)
            Next

            '' 创建各行
            For Each acEnt As Entity In acDBObjCollCols
                    Dim nRowsCount As Integer = 1

                    While (nRows > nRowsCount)
                            Dim acEntClone As Entity = acEnt.Clone()
                            acDBObjCollLvls.Add(acEntClone)

                            '' 计算新位置
                            Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase,
_
                                 dArrayAng + dAng, dRowOffset * nRowsCount)

                            Dim acVec2d As Vector2d =
acPt2dArrayBase.GetVectorTo(acPt2dTo)
                            Dim acVec3d As Vector3d = New Vector3d(acVec2d.X,
acVec2d.Y, 0)
                            acEntClone.TransformBy(Matrix3d.Displacement(acVec3d
))

                            acBlkTblRec.AppendEntity(acEntClone)
                            acTrans.AddNewlyCreatedDBObject(acEntClone, True)
```

```
                        nRowsCount = nRowsCount + 1
                    End While
            Next

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
{
    return new Point2d(pPt.X + dDist * Math.Cos(dAng),
                                        pPt.Y + dDist * Math.Sin(dAng));
}

[CommandMethod("RectangularArrayObject")]
public static void RectangularArrayObject()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                        OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

            // 创建 Circle 对象
```

```
        Circle acCirc = new Circle();
        acCirc.Center = new Point3d(200, 200, 0);
        acCirc.Radius = 60;

        // 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // 创建 5×5 矩形阵列
        int nRows = 5;
        int nColumns = 5;

        // 设置行列间距及阵列基角
        double dRowOffset = 200;
        double dColumnOffset = 300;
        double dArrayAng = 0;

        // 获取到当前 UCS 坐标系 X 轴的角度
        Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
        CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;
        Vector2d acVec2dAng = new Vector2d(curUCS.Xaxis.X,
                                           curUCS.Xaxis.Y);

        // 如果 UCS 坐标系是旋转的，相应地调整阵列的角度
        dArrayAng = dArrayAng + acVec2dAng.Angle;

        // 使用对象范围的左上角作为阵列的基点
        Extents3d acExts = acCirc.Bounds.GetValueOrDefault();
        Point2d acPt2dArrayBase = new Point2d(acExts.MinPoint.X,

    acExts.MaxPoint.Y);

        // 跟踪为每列创建的对象
        DBObjectCollection acDBObjCollCols = new DBObjectCollection();
        acDBObjCollCols.Add(acCirc);

        // 创建第一行的对象（个数等于列数）
        int nColumnsCount = 1;
        while (nColumns > nColumnsCount)
        {
                Entity acEntClone = acCirc.Clone() as Entity;
                acDBObjCollCols.Add(acEntClone);

                // 计算新位置
```

```
                Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                               dArrayAng,
                                               dColumnOffset * nColumnsCount);

                Vector2d acVec2d = acPt2dArrayBase.GetVectorTo(acPt2dTo);
                Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
                acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

                acBlkTblRec.AppendEntity(acEntClone);
                acTrans.AddNewlyCreatedDBObject(acEntClone, true);

                nColumnsCount = nColumnsCount + 1;
            }

            // 设置 90° 的弧度值
            double dAng = Math.PI / 2;

            // 记录为每行列创建的对象
            DBObjectCollection acDBObjCollLvls = new DBObjectCollection();

            foreach (DBObject acObj in acDBObjCollCols)
            {
                acDBObjCollLvls.Add(acObj);
            }

            // 创建其余各行
            foreach (Entity acEnt in acDBObjCollCols)
            {
                int nRowsCount = 1;

                while (nRows > nRowsCount)
                {
                    Entity acEntClone = acEnt.Clone() as Entity;
                    acDBObjCollLvls.Add(acEntClone);

                    // 计算新位置
                    Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                                   dArrayAng + dAng,
                                                   dRowOffset * nRowsCount);

                    Vector2d acVec2d =
acPt2dArrayBase.GetVectorTo(acPt2dTo);
                    Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y,
0);
```

```
                        acEntClone.TransformBy(Matrix3d.Displacement(acVec3d
));

                        acBlkTblRec.AppendEntity(acEntClone);
                        acTrans.AddNewlyCreatedDBObject(acEntClone, true);

                        nRowsCount = nRowsCount + 1;
                    }
                }

                // 保存新对象到数据库
                acTrans.Commit();
            }
        }
```

VBA/ActiveX 代码参考

```
Sub RectangularArrayObject()
    ' 创建圆
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2#: center(1) = 2#: center(2) = 0#
    radius = 0.5
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)
    ZoomAll

    ' 定义矩形阵列参数
    Dim numberOfRows As Long
    Dim numberOfColumns As Long
    Dim numberOfLevels As Long
    Dim distanceBwtnRows As Double
    Dim distanceBwtnColumns As Double
    Dim distanceBwtnLevels As Double
    numberOfRows = 5
    numberOfColumns = 5
    numberOfLevels = 0
    distanceBwtnRows = 1
    distanceBwtnColumns = 1
    distanceBwtnLevels = 0

    ' 创建对象阵列
    Dim retObj As Variant
    retObj = circleObj.ArrayRectangular _
                (numberOfRows, numberOfColumns, numberOfLevels, _
```

```
                    distanceBwtnRows, distanceBwtnColumns, distanceBwtnLevels)

        ZoomAll
End Sub
```

3.4.7 延伸和修剪对象

我们可以改变圆弧的角度，还可以改变直线、开放多段线、椭圆弧及开放样条曲线的长度。其结果类似于延伸对象和修剪对象。

我们可以编辑对象属性来延伸或修剪对象。例如，延长一条直线，只需简单修改直线的 StartPoint 属性或 EndPoint 属性的坐标。修改圆弧的角度，只需简单修改圆弧的 StartAngle 属性或 EndAngle 属性。改变了对象的一个或多个属性后，我们需要重新生成图形显示才能看到所作的修改。

更多关于延伸和修剪对象的内容，见《*AutoCAD 用户指南*》中的“修改对象的大小和形状”内容。

延长一条直线

本例创建一条直线，然后修改其 EndPoint 属性来使直线变长。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("ExtendObject")> _
Public Sub ExtendObject()
    '' 获取当前文档和数据库
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
```

```
            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                         OpenMode.ForRead)

            '' 以写模式打开 Block 表记录 Model 空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                            OpenMode.ForWrite)

            '' 从(4,4,0)到(7,7,0)创建 1 条直线段
            Dim acLine As Line = New Line(New Point3d(4, 4, 0), _
                                          New Point3d(7, 7, 0))

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acLine)
            acTrans.AddNewlyCreatedDBObject(acLine, True)

            '' 更新图形显示，显示一个信息框
            acDoc.Editor.Regen()
            Application.ShowAlertDialog("Before extend")

            '' 直线延长一倍
            acLine.EndPoint = acLine.EndPoint + acLine.Delta

            '' 将新对象保存到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ExtendObject")]
public static void ExtendObject()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;
```

```
    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                         OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

            // 从(4,4,0)到(7,7,0)创建 1 条直线段
            Line acLine = new Line(new Point3d(4, 4, 0), new Point3d(7, 7, 0));

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acLine);
            acTrans.AddNewlyCreatedDBObject(acLine, true);

            //更新显示
            acDoc.Editor.Regen();
            Application.ShowAlertDialog("Before extend");

            // 直线延长一倍
            acLine.EndPoint = acLine.EndPoint + acLine.Delta;

            // 将新对象保存到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub ExtendObject()
       ’ 定义并创建 1 条直线段
       Dim lineObj As AcadLine
       Dim startPoint(0 To 2) As Double
       Dim endPoint(0 To 2)   As Double
       startPoint(0) = 4
       startPoint(1) = 4
       startPoint(2) = 0
       endPoint(0) = 7
       endPoint(1) = 7
```

```
        endPoint(2) = 0
        Set lineObj = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)
        lineObj.Update

        ' 直线延长一倍
        endPoint(0) = lineObj.endPoint(0) + lineObj.Delta(0)
        endPoint(1) = lineObj.endPoint(1) + lineObj.Delta(1)
        endPoint(2) = lineObj.endPoint(2) + lineObj.Delta(2)
        lineObj.endPoint = endPoint
        lineObj.Update
End Sub
```

3.4.8 分解对象

分解对象就是将单个对象转变为它的组件。我们使用 Explode()函数分解对象，该函数需要一个 DBObjectCollection 类型的参数用来返回分解后的对象集合。例如，分解多段线会产生一个包含多条直线和圆弧的对象集合。

如果分解的是图块，则返回的对象集合包含定义图块的各个图形对象。完成分解后，原对象保留不变。如果要用返回的对象集合代替原对象，需删除原对象然后将返回的对象集合添加到块表记录。

关于分解对象的更多内容，见《AutoCAD 用户指南》中的“解除关联合成对象（分解）”。

分解多段线

本例创建一个轻量级多段线对象，然后将其分解为最简单的对象。多段线分解后将其从内存清除，将分解返回的对象添加到模型空间。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("ExplodeObject")> _
```

```
Public Sub ExplodeObject()
    '' 获取当前文档和数据库
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        '' 以读模式打开 Block 表
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                                OpenMode.F
orRead)

        '' 以写模式打开 Block 表记录 Model 空间
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                OpenMode.ForWrite)

        '' 创建一个轻量级多段线
        Using acPoly As Polyline = New Polyline()
            acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
            acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
            acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
            acPoly.AddVertexAt(3, New Point2d(3, 2), 0, 0, 0)
            acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)
            acPoly.AddVertexAt(5, New Point2d(4, 1), 0, 0, 0)

            '' 在顶点 3 和顶点 4 之间设置隆起
            acPoly.SetBulgeAt(3, -0.5)

            '' 分解多段线
            Dim acDBObjColl As DBObjectCollection = New
DBObjectCollection()
            acPoly.Explode(acDBObjColl)

            For Each acEnt As Entity In acDBObjColl
                '' 将新对象添加到块表记录和事务
                acBlkTblRec.AppendEntity(acEnt)
                acTrans.AddNewlyCreatedDBObject(acEnt, True)
            Next

            '' Dispose of the in memory polyline
```

```
            End Using

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("ExplodeObject")]
public static void ExplodeObject()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                         OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

            // 创建一个轻量级多段线
            using (Polyline acPoly = new Polyline())
            {
                    acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
                    acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
                    acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
                    acPoly.AddVertexAt(3, new Point2d(3, 2), 0, 0, 0);
                    acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);
                    acPoly.AddVertexAt(5, new Point2d(4, 1), 0, 0, 0);
```

```
                // 在顶点 3 和顶点 4 之间设置隆起
                acPoly.SetBulgeAt(3, -0.5);

                // 分解多段线
                DBObjectCollection acDBObjColl = new DBObjectCollection();
                acPoly.Explode(acDBObjColl);

                foreach (Entity acEnt in acDBObjColl)
                {
                        // 将新对象添加到块表记录和事务
                        acBlkTblRec.AppendEntity(acEnt);
                        acTrans.AddNewlyCreatedDBObject(acEnt, true);
                }

                // 清除内存中的多段线
            }

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub ExplodeObject()
        Dim plineObj As AcadLWPolyline
        Dim points(0 To 11) As Double

        ' 定义多段线顶点
        points(0) = 1: points(1) = 1
        points(2) = 1: points(3) = 2
        points(4) = 2: points(5) = 2
        points(6) = 3: points(7) = 2
        points(8) = 4: points(9) = 4
        points(10) = 4: points(11) = 1

        ' 创建 Polyline 对象
        Set plineObj = ThisDrawing.ModelSpace. _
                                AddLightWeightPolyline(points)

        ' 在顶点 3 和顶点 4 之间设置隆起
        ' 加点儿小变化～
        plineObj.SetBulge 3, -0.5
```

```
        ' 分解多段线
        Dim explodedObjects As Variant
        explodedObjects = plineObj.Explode

        ' 删除多段线
        plineObj.Erase
End Sub
```

3.4.9 编辑多段线

二维多段线和三维多段线、矩形、多边形、圆环以及三维多边形网格都是多段线的变体，可以使用相同的方法进行编辑。

AutoCAD 能够识别拟合多段线和样条曲线拟合多段线。样条曲线拟合多段线使用曲线拟合而成，类似于 B 样条曲线。有两种样条曲线拟合多段线：二次的和三次的，两种多段线均由系统变量 SPLINETYPE 控制。拟合多段线使用标准曲线利用在给定顶点设置切线方向拟合而成。

使用 Polyline 对象、 Polyline2d 对象或 Polyline3d 对象的属性和方法来编辑多段线。下列属性和方法用于打开或闭合一条多段线、改变多段线顶点坐标、增加顶点等：

Closed 属性

打开或闭合多段线；

ConstantWidth 属性

设置轻量级和 2D 多段线宽度常量；

AppendVertex 方法

给 2D 或 3D 多段线增加顶点；

AddVertexAt 方法

给轻量级多段线增加顶点；

ReverseCurve 方法

反转多段线的方向；

下列方法用于修改多段线的凸度或宽度：

SetBulgeAt 方法

给定线段索引，设置多段线的凸度（将直线段变成曲线段）；

SetStartWidthAt 方法

给定线段索引，设置轻量级多段线的起始宽度；

Straighten 方法

拉直 2D 或 3D 多段线

关于编辑多段线的更多内容，见《AutoCAD 用户指南》中的“修改或连结多段线”。

编辑多段线

本示例代码先创建一条轻量级多段线，然后将第三条线段变成曲线（添加凸度）、追加一个顶点、修改最后那条线段的宽度，最后闭合该多段线。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EditPolyline")> _
Public Sub EditPolyline()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
```

```
            Dim acBlkTbl As BlockTable
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                    OpenMode.ForRead)

            '' 以写模式打开 Block 表记录 Model 空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                    OpenMode.ForWrite)
            '' 创建一个轻量级多段线
            Dim acPoly As Polyline = New Polyline()
            acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
            acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
            acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
            acPoly.AddVertexAt(3, New Point2d(3, 2), 0, 0, 0)
            acPoly.AddVertexAt(4, New Point2d(4, 4), 0, 0, 0)

            '' 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acPoly)
            acTrans.AddNewlyCreatedDBObject(acPoly, True)

            '' 在顶点 3 和顶点 4 之间设置隆起
            acPoly.SetBulgeAt(3, -0.5)

            '' 添加一个新顶点
            acPoly.AddVertexAt(5, New Point2d(4, 1), 0, 0, 0)

            '' 设置线段 4 的起止宽度
            acPoly.SetStartWidthAt(4, 0.1)
            acPoly.SetEndWidthAt(4, 0.5)

            '' 闭合多段线
            acPoly.Closed = True

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```
using Autodesk.AutoCAD.Geometry;

[CommandMethod("EditPolyline")]
public static void EditPolyline()
{
   // 获取当前文档和数据库
   Document acDoc = Application.DocumentManager.MdiActiveDocument;
   Database acCurDb = acDoc.Database;

   // 启动事务
   using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
   {
           // 以读模式打开 Block 表
           BlockTable acBlkTbl;
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                        OpenMode.ForRead) as BlockTable;

           // 以写模式打开 Block 表记录 Model 空间
           BlockTableRecord acBlkTblRec;
           acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

           // 创建一个轻量级多段线
           Polyline acPoly = new Polyline();
           acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
           acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
           acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
           acPoly.AddVertexAt(3, new Point2d(3, 2), 0, 0, 0);
           acPoly.AddVertexAt(4, new Point2d(4, 4), 0, 0, 0);

           // 将新对象添加到块表记录和事务
           acBlkTblRec.AppendEntity(acPoly);
           acTrans.AddNewlyCreatedDBObject(acPoly, true);

           // 在顶点 3 和顶点 4 之间设置隆起设置线段 3 的凸度
           acPoly.SetBulgeAt(3, -0.5);

           // 添加一个新顶点
           acPoly.AddVertexAt(5, new Point2d(4, 1), 0, 0, 0);

           // 设置线段 4 的起止宽度
           acPoly.SetStartWidthAt(4, 0.1);
           acPoly.SetEndWidthAt(4, 0.5);
```

```
            // 闭合多段线
            acPoly.Closed = true;

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub EditPolyline()
        Dim plineObj As AcadLWPolyline
        Dim points(0 To 9) As Double

        ' 定义 2D 多段线顶点
        points(0) = 1: points(1) = 1
        points(2) = 1: points(3) = 2
        points(4) = 2: points(5) = 2
        points(6) = 3: points(7) = 2
        points(8) = 4: points(9) = 4

        ' 创建 Polyline 对象
        Set plineObj = ThisDrawing.ModelSpace. _
                                        AddLightWeightPolyline(points)

        ' 线段 3 添加凸度
        plineObj.SetBulge 3, -0.5

        ' 定义新顶点
        Dim newVertex(0 To 1) As Double
        newVertex(0) = 4: newVertex(1) = 1

        ' 添加顶点
        plineObj.AddVertex 5, newVertex

        ' 设置新线段的宽度
        plineObj.SetWidth 4, 0.1, 0.5

        ' 闭合多段线
        plineObj.Closed = True
        plineObj.Update
End Sub
```

3.4.10 编辑样条曲线

我们可以编辑开放样条曲线或闭合样条曲线的属性，甚至可以将其转换为多段线。下列属性用于打开或闭合样条曲线、修改控制点、转变样条曲线方向等：

Degree

返回样条曲线的多项表达式的阶数；

EndFitTangent

返回样条线的终点切线的方向矢量；

FitTolerance

使用新的拟合误差值将现有点重新拟合成样条曲线；

NumControlPoints

返回样条曲线的控制点的个数；

NumFitPoints

返回样条曲线的拟合点的个数；

StartFitTangent

返回样条曲线的起点切线的方向矢量；

另外，还可以使用下列方法编辑样条曲线：

InsertFitPointAt

在给定索引位置添加一个拟合点；

ElevateDegree

增加样条曲线的阶数；

GetControlPointAt

获取样条曲线给定索引位置的控制点（只一个控制点）。NumControlPoints 属性包含有样条曲线控制点的个数。

GetFitPointAt

获取样条曲线给定索引位置的拟合点（只一个拟合点。要查询样条曲线的所有拟合点，应使用Spline对象的FitData属性,并调用FitData结构的GetFitPoints()成员函数。该函数返回一个 Point3dCollection 对象，即所有拟合点的集合。）。NumFitPoints 属性包含有样条曲线拟合点的个数。

RemoveFitPointAt

删除样条曲线给定索引位置的拟合点；

ReverseCurve

反转样条曲线的方向；

SetControlPointAt

设置样条曲线给定索引位置的控制点

SetFitPointAt

设置样条曲线给定索引位置的拟合点（只设置一个拟合点）；

SetWeightAt

设置样条曲线给定索引位置的权重；

下列只读属性用于查询样条曲线：

Area

获取样条曲线闭合区域的面积；

Closed

表示样条曲线是开放的还是闭合的；

Degree

获取样条曲线多项表达式的阶数；

IsPeriodic

表示样条曲线是否为周期性的；

IsPlanar

表示样条曲线是否为平面的（二维的）；

IsRational

表示样条曲线是否为有理的；

NumControlPoints

获取样条曲线控制点的个数；

NumfFitPoints

获取样条曲线拟合点的个数；

更多关于编辑样条曲线的内容，见《*AutoCAD 用户指南*》里“修改样条曲线”一节。

修改样条曲线的控制点

本例创建一条样条曲线，然后修改它的第一个控制点。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
```

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EditSpline")> _
Public Sub EditSpline()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

           '' 以读模式打开 Block 表
           Dim acBlkTbl As BlockTable
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                   OpenMode.ForRead)

           '' 以写模式打开 Block 表记录 Model 空间
           Dim acBlkTblRec As BlockTableRecord
           acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                   OpenMode.ForWrite)

           '' 创建 Point3d 对象集合
           Dim acPt3dColl As Point3dCollection = New Point3dCollection()
           acPt3dColl.Add(New Point3d(1, 1, 0))
           acPt3dColl.Add(New Point3d(5, 5, 0))
           acPt3dColl.Add(New Point3d(10, 0, 0))

           '' 设置样条曲线起止点的切线矢量
           Dim acStartTan As Vector3d = New Vector3d(0.5, 0.5, 0)
           Dim acEndTan As Vector3d = New Vector3d(0.5, 0.5, 0)

           '' 创建样条曲线
           Dim acSpline As Spline = New Spline(acPt3dColl, _
                                               acStartTan, _
                                               acEndTan, 4, 0)

           '' 修改控制点
           acSpline.SetControlPointAt(0, New Point3d(0, 3, 0))

           '' 将新对象添加到块表记录和事务
           acBlkTblRec.AppendEntity(acSpline)
```

```
            acTrans.AddNewlyCreatedDBObject(acSpline, True)

            '' 保存新对象到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("EditSpline")]
public static void EditSpline()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                         OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;
            // 创建 Point3d 对象集合
            Point3dCollection acPt3dColl = new Point3dCollection();
            acPt3dColl.Add(new Point3d(1, 1, 0));
            acPt3dColl.Add(new Point3d(5, 5, 0));
            acPt3dColl.Add(new Point3d(10, 0, 0));

            // 设置样条曲线起止点的切线矢量
            Vector3d acStartTan = new Vector3d(0.5, 0.5, 0);
            Vector3d acEndTan = new Vector3d(0.5, 0.5, 0);
```

```
            // 创建样条曲线
            Spline acSpline = new Spline(acPt3dColl,
                                         acStartTan,
                                          acEndTan, 4, 0);
            // 修改控制点
            acSpline.SetControlPointAt(0, new Point3d(0, 3, 0));

            // 将新对象添加到块表记录和事务
            acBlkTblRec.AppendEntity(acSpline);
            acTrans.AddNewlyCreatedDBObject(acSpline, true);

            // 保存新对象到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub EditSpline()
        ' 创建样条曲线
        Dim splineObj As AcadSpline
        Dim startTan(0 To 2) As Double
        Dim endTan(0 To 2) As Double
        Dim fitPoints(0 To 8) As Double

        startTan(0) = 0.5: startTan(1) = 0.5: startTan(2) = 0
        endTan(0) = 0.5: endTan(1) = 0.5: endTan(2) = 0
        fitPoints(0) = 1: fitPoints(1) = 1: fitPoints(2) = 0
        fitPoints(3) = 5: fitPoints(4) = 5: fitPoints(5) = 0
        fitPoints(6) = 10: fitPoints(7) = 0: fitPoints(8) = 0
        Set splineObj = ThisDrawing.ModelSpace. _
                                    AddSpline(fitPoints, startTan, endTan)
        splineObj.Update

        ' 修改第 1 个拟合点的坐标
        Dim controlPoint(0 To 2) As Double
        controlPoint(0) = 0
        controlPoint(1) = 3
        controlPoint(2) = 0
        splineObj.SetControlPoint 0, controlPoint
        splineObj.Update
End Sub
```

3.4.11 编辑图案填充

我们可以编辑图案填充的填充边界和填充图案。如果编辑关联填充的边界，且编辑后边界合法，填充图案就会随之更新，即便所在的图层处于关闭状态，关联填充也会更新。我们可以修改填充图案，或为其选择新的填充图案，但是其关联属性只在填充创建时才能设置。我们可以使用 Associative 属性检查 Hatch 对象是否为关联的。

要想看到编辑后的效果，必须使用 EvaluateHatch()方法对所编辑的图案填充重新进行评估。

3.4.11.1 编辑填充边界

我们可以给 Hatch 对象的边界追加、插入及删除边界环。关联性图案填充会更新以适应边界变化，非关联性图案填充不更新。

下列方法用于编辑图案填充的边界：

AppendLoop()

给图案填充追加一个边界环。AppendLoop()方法的第一个参数是一个 HatchLoopTypes 枚举类型的常量，该参数定义了所追加的边界环的类型。

GetLoopAt()

获取图案填充给定索引位置的边界环。

InsertLoopAt()

在图案填充的给定索引位置插入一个边界环。

RemoveLoopAt()

在图案填充的给定索引位置删除一个边界环。

HatchLoopTypes 枚举类型的定义如下：

VB.NET

```
Public Enum HatchLoopTypes
```

```
  Default = 0
  Derived = 4
  Duplicate = &H100
  External = 1
  NotClosed = &H20
  Outermost = &H10
  Polyline = 2
  SelfIntersecting = &H40
  Textbox = 8
  TextIsland = &H80
End Enum
```

C#

```
public enum HatchLoopTypes {
  Default = 0,
  Derived = 4,
  Duplicate = 0x100,
  External = 1,
  NotClosed = 0x20,
  Outermost = 0x10,
  Polyline = 2,
  SelfIntersecting = 0x40,
  Textbox = 8,
  TextIsland = 0x80
}
```

下列方法用于查询图案填充的边界：

LoopTypeAt()

获取图案填充给定索引位置边界环的类型。

NumberOfLoops()

获取图案填充的边界环的个数。

给图案填充追加一个内部边界环

本示例代码先创建一个关联图案填充，然后创建一个圆并追加该圆作为填充的内部边界环。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EditHatchAppendLoop")> _
Public Sub EditHatchAppendLoop()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database
   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                    OpenMode.ForRead)

            '' 以写模式打开 Block 表记录 Model 空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                   OpenMode.ForWrite)

            '' 创建一个 Arc 对象作为填充的闭合边界
            Dim acArc As Arc = New Arc(New Point3d(5, 3, 0), 3, 0, 3.141592)

            acBlkTblRec.AppendEntity(acArc)
            acTrans.AddNewlyCreatedDBObject(acArc, True)

            '' 创建一个 Line 对象作为填充的闭合边界
            Dim acLine As Line = New Line(acArc.StartPoint, acArc.EndPoint)

            acBlkTblRec.AppendEntity(acLine)
            acTrans.AddNewlyCreatedDBObject(acLine, True)

            ''将圆弧和直线添加到 ObjectIdCollection 集合
            Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
            acObjIdColl.Add(acArc.ObjectId)
            acObjIdColl.Add(acLine.ObjectId)
```

```
            '' 创建 Hatch 对象并添加到块表记录
            Dim acHatch As Hatch = New Hatch()
            acBlkTblRec.AppendEntity(acHatch)
            acTrans.AddNewlyCreatedDBObject(acHatch, True)

            '' 设置填充对象的属性
            '' 关联属性必须在将填充对象添加到
            '' 块表记录之后、执行 AppendLoop 之前设置
            acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31")
            acHatch.Associative = True
            acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl)

            '' 创建一个 Circle 对象作为填充的内部边界
            Dim acCirc As Circle = New Circle()
            acCirc.Center = New Point3d(5, 4.5, 0)
            acCirc.Radius = 1

            acBlkTblRec.AppendEntity(acCirc)
            acTrans.AddNewlyCreatedDBObject(acCirc, True)

            '' 将圆添加到 ObjectIdCollection
            acObjIdColl.Clear()
            acObjIdColl.Add(acCirc.ObjectId)

            '' 追加圆为内部边界环并评估填充对象
            acHatch.AppendLoop(HatchLoopTypes.Default, acObjIdColl)
            acHatch.EvaluateHatch(True)

            '' 将新对象保存到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("EditHatchAppendLoop")]
public static void EditHatchAppendLoop()
```

```
{
   // 获取当前文档和数据库
   Document acDoc = Application.DocumentManager.MdiActiveDocument;
   Database acCurDb = acDoc.Database;

   // 启动事务
   using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
   {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                         OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                  OpenMode.ForWrite) as BlockTableRecord;

            // 创建一个 Arc 对象作为填充的闭合边界
            Arc acArc = new Arc(new Point3d(5, 3, 0), 3, 0, 3.141592);

            acBlkTblRec.AppendEntity(acArc);
            acTrans.AddNewlyCreatedDBObject(acArc, true);

            // 创建一个 Line 对象作为填充的闭合边界
            Line acLine = new Line(acArc.StartPoint, acArc.EndPoint);

            acBlkTblRec.AppendEntity(acLine);
            acTrans.AddNewlyCreatedDBObject(acLine, true);

            // 将圆弧和直线添加到 ObjectIdCollection
            ObjectIdCollection acObjIdColl = new ObjectIdCollection();
            acObjIdColl.Add(acArc.ObjectId);
            acObjIdColl.Add(acLine.ObjectId);

            // 创建 Hatch 对象并添加到块表记录
            Hatch acHatch = new Hatch();
            acBlkTblRec.AppendEntity(acHatch);
            acTrans.AddNewlyCreatedDBObject(acHatch, true);

            // 设置填充对象的属性
            // 关联属性必须在将填充对象添加到块表记录
            // 之后、执行 AppendLoop 之前设置
```

```
            acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31");
            acHatch.Associative = true;
            acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl);

            //  创建一个圆对象作为填充的内部边界
            Circle acCirc = new Circle();
            acCirc.Center = new Point3d(5, 4.5, 0);
            acCirc.Radius = 1;

            acBlkTblRec.AppendEntity(acCirc);
            acTrans.AddNewlyCreatedDBObject(acCirc, true);

            // 将圆添加到 ObjectIdCollection
            acObjIdColl.Clear();
            acObjIdColl.Add(acCirc.ObjectId);

            // 追加圆为内部边界环并评估填充对象
            acHatch.AppendLoop(HatchLoopTypes.Default, acObjIdColl);
            acHatch.EvaluateHatch(true);

            // 将新对象保存到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub EditHatchAppendLoop()
       Dim hatchObj As AcadHatch
       Dim patternName As String
       Dim PatternType As Long
       Dim bAssociativity As Boolean

       ' 定义、创建图案填充
       patternName = "ANSI31"
       PatternType = 0
       bAssociativity = True
       Set hatchObj = ThisDrawing.ModelSpace. _
                      AddHatch(PatternType, patternName, bAssociativity)

       ' 创建外边界环
       Dim outerLoop(0 To 1) As AcadEntity
       Dim center(0 To 2) As Double
       Dim radius As Double
```

```
        Dim startAngle As Double
        Dim endAngle As Double
        center(0) = 5: center(1) = 3: center(2) = 0
        radius = 3
        startAngle = 0
        endAngle = 3.141592
        Set outerLoop(0) = ThisDrawing.ModelSpace. _
                                AddArc(center, radius, startAngle, endAngle)
        Set outerLoop(1) = ThisDrawing.ModelSpace. _
                                AddLine(outerLoop(0).startPoint,_
                                                    outerLoop(0).endPoint)

        ' 给填充对象添加外边界
        hatchObj.AppendOuterLoop (outerLoop)

        ' 创建 1 个圆作为填充的内边界环
        Dim innerLoop(0) As AcadEntity
        center(0) = 5: center(1) = 4.5: center(2) = 0
        radius = 1
        Set innerLoop(0) = ThisDrawing.ModelSpace.AddCircle(center, radius)
        ' 给填充追加内边界环
        hatchObj.AppendInnerLoop (innerLoop)
        ' 评估并显示填充
        hatchObj.Evaluate
        ThisDrawing.Regen True
End Sub
```

3.4.11.2 编辑填充图案

我们可以修改现有填充图案的角度或间隔，或者用实体填充、渐变色填充以及 AutoCAD 提供的其他预定义填充图案来替换现有填充图案。图案填充对话框中的图案选项显示了这些图案的列表。为减小文件大小，在图形中将填充图案定义为单独的图像对象。

下列属性和方法用于编辑填充图案：

GradientAngle

指定填充的渐变色角度

GradientName

返回填充的渐变色名称

GradientShift

指定填充的渐变插值

GradientType

返回填充的渐变色类型

PatternAngle

指定填充图案的角度

PatternDouble

指定用户自定义填充是否为双填充（double-hatched）

PatternName

返回填充图案的名称（使用 SetHatchPattern 方法设置填充图案的名称和类型）

PatternScale

指定填充图案比例

PatternSpace

指定用户自定义填充图案的间隔

PatternType

返回填充图案的类型（使用 SetHatchPattern 方法设置填充图案的名称和类型）

SetGradient()

设置渐变填充的名称和类型

SetHatchPattern()

设置填充图案的名称和类型

修改填充图案的间距

本例创建一个图案填充，然后将该填充图案的间距加 2。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("EditHatchPatternScale")> _
Public Sub EditHatchPatternScale()
   '' 获取当前文档和数据库
   Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

   '' 启动事务
   Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

           '' 以读模式打开 Block 表
           Dim acBlkTbl As BlockTable
           acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                                OpenMode.ForRead)

           '' 以写模式打开 Block 表记录 Model 空间
           Dim acBlkTblRec As BlockTableRecord
           acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                                   OpenMode.ForWrite)

           '' 创建一个圆作为填充的边界
           Dim acCirc As Circle = New Circle()
           acCirc.Center = New Point3d(5, 3, 0)
           acCirc.Radius = 3

           acBlkTblRec.AppendEntity(acCirc)
           acTrans.AddNewlyCreatedDBObject(acCirc, True)

           '' 将圆添加到 ObjectIdCollection 集合
           Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
           acObjIdColl.Add(acCirc.ObjectId)
```

```
            '' 创建 Hatch 对象并添加到块表记录
            Dim acHatch As Hatch = New Hatch()
            acBlkTblRec.AppendEntity(acHatch)
            acTrans.AddNewlyCreatedDBObject(acHatch, True)
            '' 设置属性
            acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31")
            acHatch.Associative = True
            '' 添加边界环
            acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl)

            '' 评估填充
            acHatch.EvaluateHatch(True)

            '' 填充图案的比例加 2，重新评估填充
            acHatch.PatternScale = acHatch.PatternScale + 2
            acHatch.SetHatchPattern(acHatch.PatternType, acHatch.PatternName)
            acHatch.EvaluateHatch(True)

            '' 将新对象保存到数据库
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("EditHatchPatternScale")]
public static void EditHatchPatternScale()
{
    // 获取当前文档和数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
```

```
                                                OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

            // 创建一个圆作为填充的边界
            Circle acCirc = new Circle();
            acCirc.Center = new Point3d(5, 3, 0);
            acCirc.Radius = 3;

            acBlkTblRec.AppendEntity(acCirc);
            acTrans.AddNewlyCreatedDBObject(acCirc, true);

            // 将圆添加到 ObjectIdCollection 集合
            ObjectIdCollection acObjIdColl = new ObjectIdCollection();
            acObjIdColl.Add(acCirc.ObjectId);

            // 创建 Hatch 对象并添加到块表记录
            Hatch acHatch = new Hatch();
            acBlkTblRec.AppendEntity(acHatch);
            acTrans.AddNewlyCreatedDBObject(acHatch, true);

            // 设置填充对象属性
            // 关联属性必须在将填充对象添加到
            // 块表记录之后、执行 AppendLoop 之前设置
            acHatch.SetHatchPattern(HatchPatternType.PreDefined, "ANSI31");
            acHatch.Associative = true;
            acHatch.AppendLoop(HatchLoopTypes.Outermost, acObjIdColl);

            // 评估填充
            acHatch.EvaluateHatch(true);

            // 填充图案的比例加 2，重新评估填充
            acHatch.PatternScale = acHatch.PatternScale + 2;
            acHatch.SetHatchPattern(acHatch.PatternType, acHatch.PatternName);
            acHatch.EvaluateHatch(true);

            // 将新对象保存到数据库
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub EditHatchPatternScale()
       Dim hatchObj As AcadHatch
       Dim patternName As String
       Dim PatternType As Long
       Dim bAssociativity As Boolean

       ' 定义图案填充
       patternName = "ANSI31"
       PatternType = 0
       bAssociativity = True

       ' 创建关联 Hatch 对象
       Set hatchObj = ThisDrawing.ModelSpace. _
                           AddHatch(PatternType, patternName, bAssociativity)

       ' 创建外边界环
       Dim outerLoop(0 To 0) As AcadEntity
       Dim center(0 To 2) As Double
       Dim radius As Double
       center(0) = 5
       center(1) = 3
       center(2) = 0
       radius = 3
       Set outerLoop(0) = ThisDrawing.ModelSpace.AddCircle(center, radius)
       hatchObj.AppendOuterLoop (outerLoop)
       hatchObj.Evaluate

       ' 修改填充图案比例：当前比例+2
       hatchObj.patternScale = hatchObj.patternScale + 2
       hatchObj.Evaluate
       ThisDrawing.Regen True
End Sub
```

3.5 使用图层、颜色和线型

图层就像透明覆层，在上面我们可以组织各种不同的图形信息。我们创建的对象均拥有图层、颜色和线型等属性。颜色用于区分图形中的相似元素；线型，比如中心线或隐藏线，用于区分不同的绘图元素。组织好图层及图层上的对象能使管理图形信息更容易。

本主题的更多内容，见《AutoCAD 用户指南》中的“控制对象属性”。

3.5.1 使用图层

我们总是在某个图层上绘图，可能是在默认图层上，或者是在自己创建并命名的图层上。每个图层都有与之关联的颜色和线型，以及其他属性。 例如，我们可以创建一个图层并将其颜色设置为蓝色、将其线型设置为 CENTER，这样我们就只能在该图层上画中心线。之后，每当要绘制中心线时，我们切换到这个图层并直接绘制就可以了。

所有的图层和线型存储在单独的符号表里。图层保存在 Layers 表里，线型保存在 Linetypes 表里。

3.5.1.1 检索图层和线型

我们可以遍历 Layers 表和 Linetypes 表，来找到图形中所有的图层和线型。

遍历 Layers 表

下面的代码遍历 Layers 表来收集图形中所有图层的名称，然后将名称显示在信息框内。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("DisplayLayerNames")> _
Public Sub DisplayLayerNames()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)
```

```
    Dim sLayerNames As String = ""

    For Each acObjId As ObjectId In acLyrTbl
        Dim acLyrTblRec As LayerTableRecord
        acLyrTblRec = acTrans.GetObject(acObjId, _
                                        OpenMode.ForRead)

        sLayerNames = sLayerNames & vbLf & acLyrTblRec.Name
    Next

    Application.ShowAlertDialog("The layers in this drawing are: " & _
                                sLayerNames)

    '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("DisplayLayerNames")]
public static void DisplayLayerNames()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerNames = "";

      foreach (ObjectId acObjId in acLyrTbl)
      {
          LayerTableRecord acLyrTblRec;
          acLyrTblRec = acTrans.GetObject(acObjId,
```

```
                                        OpenMode.ForRead) as LayerTableRecord;

            sLayerNames = sLayerNames + "\n" + acLyrTblRec.Name;
        }

        Application.ShowAlertDialog("The layers in this drawing are: " +
                                    sLayerNames);

        // 关闭事务
    }
}
```

VBA/ActiveX 代码参考

```
Sub DisplayLayerNames()
    Dim layerNames As String
    Dim entry As AcadLayer
    layerNames = ""

    For Each entry In ThisDrawing.Layers
        layerNames = layerNames + entry.Name + vbCrLf
    Next

    MsgBox "The layers in this drawing are: " + _
           vbCrLf + layerNames
End Sub
```

3.5.1.2 创建并命名图层

我们可以创建新图层，并给新图层的颜色属性和线型属性赋值。每个图层均为 Layers 表的一条记录。创建新图层并将其添加到 Layers 表中，使用 Add 函数。

创建图层时可以为其命名，修改图层名称用 Name 属性。图层名称可以含最多 255 个字符，可以包括字母、数字，以及美元符号（$）、连字符（-）、下划线（_）等特殊字符。

新建一个图层，颜色为红色，然后添加到图层表里

下面的代码新创建一个图层和一个圆对象，新图层的颜色设为红色。将圆指定到新图层时，圆的颜色会相应地变成红色。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.Colors
```

```
<CommandMethod("CreateAndAssignALayer")> _
Public Sub CreateAndAssignALayer()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "Center"

      If acLyrTbl.Has(sLayerName) = False Then
          Dim acLyrTblRec As LayerTableRecord = New LayerTableRecord()

          '' 赋予图层颜色和名称（AutoCADColorIndex 为 1 表示红色）
          acLyrTblRec.Color = Color.FromColorIndex(ColorMethod.ByAci, 1)
          acLyrTblRec.Name = sLayerName

          '' 以写模式升级打开图层表
          acLyrTbl.UpgradeOpen()

          '' 添加新图层到图层表，记录事务
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
      End If

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建一个圆
      Dim acCirc As Circle = New Circle()
```

```
        acCirc.Center = New Point3d(2, 2, 0)
        acCirc.Radius = 1
        acCirc.Layer = sLayerName

        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' 保存修改，关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Colors;

[CommandMethod("CreateAndAssignALayer")]
public static void CreateAndAssignALayer()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "Center";

      if (acLyrTbl.Has(sLayerName) == false)
      {
          LayerTableRecord acLyrTblRec = new LayerTableRecord();

          // 赋予图层颜色和名称（AutoCADColorIndex 为 1 表示红色）
          acLyrTblRec.Color = Color.FromColorIndex(ColorMethod.ByAci, 1);
          acLyrTblRec.Name = sLayerName;
```

```
            // 以写模式升级打开图层表
            acLyrTbl.UpgradeOpen();

            // 添加新图层到图层表，记录事务
            acLyrTbl.Add(acLyrTblRec);
            acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
        }

        // 以读模式打开 Block 表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开 Block 表记录 Model 空间
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // 创建一个圆
        Circle acCirc = new Circle();
        acCirc.Center = new Point3d(2, 2, 0);
        acCirc.Radius = 1;

        // 设置圆所归属的图层
        acCirc.Layer = sLayerName;

        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // 保存修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateAssignALayer()
    ' 新建一个图层 Center，设置其颜色为红色
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("Center")
    layerObj.color = acRed

    ' 新建一个圆
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
```

```
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace. _
                    AddCircle(center, radius)

    ' 将圆放在 Center 图层上
    circleObj.Layer = "Center"

    circleObj.Update
End Sub
```

3.5.1.3 将图层设为当前图层

我们总是在活动图层绘制图形。当将某个图层设为活动图层后，新创建的对象就在这个图层上。如果又将别的图层设为活动图层，之后创建的新对象就在这个新活动图层上并使用该图层的颜色和线型。不能将冻结的图层设为活动图层。

将图层设为活动图层，使用 Database 对象的 Clayer 属性，或者使用系统变量 CLAYER。详见下面的示例代码：

通过数据库将图层设为当前图层

本例使用 Database 对象的 Clayer 属性将图层设为当前图层。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("SetLayerCurrent")> _
Public Sub SetLayerCurrent()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "Center"
```

```
    If acLyrTbl.Has(sLayerName) = True Then
        ''设置图层 Center 为当前图层
        acCurDb.Clayer = acLyrTbl(sLayerName)

        '' 保存修改
        acTrans.Commit()
    End If

    '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("SetLayerCurrent")]
public static void SetLayerCurrent()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "Center";

      if (acLyrTbl.Has(sLayerName) == true)
      {
          // 设置图层 Center 为当前图层
          acCurDb.Clayer = acLyrTbl[sLayerName];

          // 保存修改
          acTrans.Commit();
      }
```

```
        // 关闭事务
    }
}
```

VBA/ActiveX 代码参考

```
ThisDrawing.ActiveLayer = ThisDrawing.Layers("Center")
```

系统变量 CLAYER 设置当前图层

本例使用系统变量 CLAYER 设置当前图层。

VB.NET

```
Application.SetSystemVariable("CLAYER", "Center")
```

C#

```
Application.SetSystemVariable("CLAYER", "Center");
```

VBA/ActiveX 代码参考

```
ThisDrawing.SetVariable "CLAYER", "Center"
```

3.5.1.4 打开和关闭图层

我们重新生成图形时，已关闭的图层也会一起重新生成，只是不能显示或打印已关闭的图层而已。通过关闭图层，可以避免每次解冻图层时都重新生成图形。当打开已关闭的图层时，AutoCAD 会重新绘制该图层上的对象。

使用代表图层的图层表记录对象的 IsOff 属性，来实现打开或关闭图层。 IsOff 属性值为 TRUE 表示关闭图层；IsOff 属性值为 FALSE 表示打开图层。

关闭图层

本例新创建一个图层并将其关闭，然后往该图层添加一个圆，不过我们看不见这个圆。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("TurnLayerOff")> _
Public Sub TurnLayerOff()
    '' 获取当前文档和数据库
```

```
Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
Dim acCurDb As Database = acDoc.Database

'' 启动事务
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' 以读模式打开图层表
    Dim acLyrTbl As LayerTable
    acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                 OpenMode.ForRead)

    Dim sLayerName As String = "ABC"
    Dim acLyrTblRec As LayerTableRecord

    If acLyrTbl.Has(sLayerName) = False Then
        acLyrTblRec = New LayerTableRecord()

        '' 给图层名赋值
        acLyrTblRec.Name = sLayerName

        '' 以写模式升级打开图层表
        acLyrTbl.UpgradeOpen()

        '' 添加新图层到图层表，记录事务
        acLyrTbl.Add(acLyrTblRec)
        acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
    Else
        acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                        OpenMode.ForWrite)
    End If

    '' 关闭图层
    acLyrTblRec.IsOff = True

    '' 以读模式打开 Block 表
    Dim acBlkTbl As BlockTable
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                 OpenMode.ForRead)

    '' 以写模式打开 Block 表记录 Model 空间
    Dim acBlkTblRec As BlockTableRecord
    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)
```

```
      '' 创建一个圆
      Dim acCirc As Circle = New Circle()
      acCirc.Center = New Point3d(2, 2, 0)
      acCirc.Radius = 1
      acCirc.Layer = sLayerName

      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      '' 保存修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("TurnLayerOff")]
public static void TurnLayerOff()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "ABC";
      LayerTableRecord acLyrTblRec;

      if (acLyrTbl.Has(sLayerName) == false)
      {
          acLyrTblRec = new LayerTableRecord();

          // 给图层名赋值
          acLyrTblRec.Name = sLayerName;
```

```
            // 以写模式升级打开图层表
            acLyrTbl.UpgradeOpen();

            // 添加新图层到图层表，记录事务
            acLyrTbl.Add(acLyrTblRec);
            acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
        }
        else
        {
            acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                            OpenMode.ForWrite) as
LayerTableRecord;
        }

        //关闭图层
        acLyrTblRec.IsOff = true;

        // 以读模式打开 Block 表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开 Block 表记录 Model 空间
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // 创建一个圆
        Circle acCirc = new Circle();
        acCirc.Center = new Point3d(2, 2, 0);
        acCirc.Radius = 1;
        acCirc.Layer = sLayerName;

        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // 提交修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub TurnLayerOff()
    ' 新建名为"ABC"的图层
```

```
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("ABC")

    ' 关闭图层"ABC"
    layerObj.LayerOn = False

    ' 创建一个圆
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace. _
                                AddCircle(center, radius)

    ' 将圆的图层属性设置为图层"ABC"
    circleObj.Layer = "ABC"
    circleObj.Update

    ThisDrawing.Regen acActiveViewport
End Sub
```

3.5.1.5 冻结和解冻图层

我们可以通过冻结图层来加快显示对图形的修改，改进对象选择性能，以及减少复杂图像的重新生成时间。对于已经冻结的图层上的对象，AutoCAD 不显示，不打印，也不重新生成。如果长时间不用某个图层，可以将其冻结。当解冻某个已冻结图层时，AutoCAD 会重新生成并显示该图层上的对象。

使用图层表记录的 IsFrozen 属性来冻结或解冻一个图层。IsFrozen 属性值为 TRUE 则冻结图层，IsFrozen 属性值为 FALSE 则解冻图层。

冻结图层

本例创建一个名为“ABC”的新图层，然后将其冻结。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("FreezeLayer")> _
Public Sub FreezeLayer()
  '' 获取当前文档和数据库
```

```
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "ABC"
      Dim acLyrTblRec As LayerTableRecord

      If acLyrTbl.Has(sLayerName) = False Then
          acLyrTblRec = New LayerTableRecord()

          '' 给图层名赋值
          acLyrTblRec.Name = sLayerName

          '' 以写模式升级打开图层表
          acLyrTbl.UpgradeOpen()

          '' 添加新图层到图层表，记录事务
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
      Else
          acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                          OpenMode.ForWrite)
      End If

      '' 冻结图层
      acLyrTblRec.IsFrozen = True

      '' 保存修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```
[CommandMethod("FreezeLayer")]
public static void FreezeLayer()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "ABC";
      LayerTableRecord acLyrTblRec;

      if (acLyrTbl.Has(sLayerName) == false)
      {
          acLyrTblRec = new LayerTableRecord();

          // 给图层名赋值
          acLyrTblRec.Name = sLayerName;

          // 以写模式升级打开图层表
          acLyrTbl.UpgradeOpen();

          // 添加新图层到图层表，记录事务
          acLyrTbl.Add(acLyrTblRec);
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
      }
      else
      {
          acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                          OpenMode.ForWrite) as
LayerTableRecord;
      }

      //  冻结图层
      acLyrTblRec.IsFrozen = true;

      // 提交，关闭事务
      acTrans.Commit();
```

```
  }
}
```

VBA/ActiveX 代码参考

```
Sub FreezeLayer()
    ' 新建名为"ABC"的图层
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("ABC")

    ' 冻结图层"ABC"
    layerObj.Freeze = True
End Sub
```

3.5.1.6 锁定和解锁图层

我们不能编辑已锁定图层上的对象；不过，如果已锁定图层是打开的并且是解冻的，那么图层上的对象仍然是可见的。我们可以将已锁定图层设为当前图层并往上添加对象，我们还可以冻结、关闭已锁定图层，以及修改其关联的颜色和线型等。

使用图层表记录的 IsLocked 属性来锁定或解锁图层。IsLocked 属性值为 TRUE 则锁定图层，IsLocked 属性值为 FALSE 则解锁图层。

锁定图层

本例创建一个名为“ABC”的新图层，然后将其锁定。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("LockLayer")> _
Public Sub LockLayer()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)
```

```
      Dim sLayerName As String = "ABC"
      Dim acLyrTblRec As LayerTableRecord

      If acLyrTbl.Has(sLayerName) = False Then
          acLyrTblRec = New LayerTableRecord()

          '' 给图层名赋值
          acLyrTblRec.Name = sLayerName

          '' 以写模式升级打开图层表
          acLyrTbl.UpgradeOpen()

          '' 添加新图层到图层表，记录事务
          acLyrTbl.Add(acLyrTblRec)
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
      Else
          acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                          OpenMode.ForWrite)
      End If

      '' 锁定图层
      acLyrTblRec.IsLocked = True

      '' 保存修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("LockLayer")]
public static void LockLayer()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
```

```
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "ABC";
      LayerTableRecord acLyrTblRec;

      if (acLyrTbl.Has(sLayerName) == false)
      {
          acLyrTblRec = new LayerTableRecord();

          //给图层名称赋值
          acLyrTblRec.Name = sLayerName;

          // 以写模式升级打开图层表
          acLyrTbl.UpgradeOpen();

          // 添加新图层到图层表，记录事务
          acLyrTbl.Add(acLyrTblRec);
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
      }
      else
      {
          acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                          OpenMode.ForWrite) as
LayerTableRecord;
      }

      //  锁定图层
      acLyrTblRec.IsLocked = true;

      // 提交修改、关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub LockLayer()
    ' 新建图层 "ABC"
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers.Add("ABC")

    '锁定图层"ABC"
    layerObj.Lock = True
```

```
End Sub
```

3.5.1.7 指定图层颜色

每个图层都可以拥有自己的颜色。一个图层的颜色由**Colors**命名空间的**Color**对象来确定。Color 对象可以容纳一个 RGB 值、一个 ACI 数字（从 1 到 255 之间的整数）、或配色系统的一个颜色值。

为图层设置颜色，用 Color 属性。

注： 像直线和圆这样的对象支持两种不同的控制当前颜色的属性：Color 属性支持 RGB 值、ACI 索引号和配色系统颜色，ColorIndex 属性只支持 ACI 索引号。

如果使用 ACI 颜色 0 或 ByBlock，AutoCAD 就用默认颜色绘制新对象（白色或黑色，依你的设置不同而异），直到将这些对象编入图块里。当插入图块时，图块中的对象继承图块当前的属性设置。

如果使用 ACI 颜色 256 或 ByLayer，则新对象继承所在图层的颜色。

更多关于使用颜色的内容，参见（§3.5.2 *使用颜色*）。

设置图层颜色

下列实例创建三个新图层，并使用三种颜色设置方法，为每个图层赋予设置不同的颜色。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Colors

<CommandMethod("SetLayerColor")> _
Public Sub SetLayerColor()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)
```

```
'' 定义一个图层名数组
Dim sLayerNames(2) As String
sLayerNames(0) = "ACIRed"
sLayerNames(1) = "TrueBlue"
sLayerNames(2) = "ColorBookYellow"

'' 定义一个图层颜色数组
Dim acColors(2) As Color
acColors(0) = Color.FromColorIndex(ColorMethod.ByAci, 1)
acColors(1) = Color.FromRgb(23, 54, 232)
acColors(2) = Color.FromNames("PANTONE Yellow 0131 C", _
                              "PANTONE(R) pastel coated")

Dim nCnt As Integer = 0

'' 添加或修改图形中的每个图层
For Each sLayerName As String In sLayerNames
    Dim acLyrTblRec As LayerTableRecord

    If acLyrTbl.Has(sLayerName) = False Then
        acLyrTblRec = New LayerTableRecord()

        '' 设置图层名
        acLyrTblRec.Name = sLayerName

        '' 以写模式升级打开图层表
        If acLyrTbl.IsWriteEnabled = False Then acLyrTbl.UpgradeOpen()

        '' 添加新图层到图层表，记录事务
        acLyrTbl.Add(acLyrTblRec)
        acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
    Else
        '' 如果图层已存在，则以写模式打开
        acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                        OpenMode.ForWrite)
    End If

    '' 设置图层颜色
    acLyrTblRec.Color = acColors(nCnt)

    nCnt = nCnt + 1
Next

'' 保存修改，关闭事务
```

```
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Colors;

[CommandMethod("SetLayerColor")]
public static void SetLayerColor()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      // 定义一个图层名数组
      string[] sLayerNames = new string[3];
      sLayerNames[0] = "ACIRed";
      sLayerNames[1] = "TrueBlue";
      sLayerNames[2] = "ColorBookYellow";

      // 定义一个图层颜色数组
      Color[] acColors = new Color[3];
      acColors[0] = Color.FromColorIndex(ColorMethod.ByAci, 1);
      acColors[1] = Color.FromRgb(23, 54, 232);
      acColors[2] = Color.FromNames("PANTONE Yellow 0131 C",
                                    "PANTONE(R) pastel coated");

      int nCnt = 0;

      // 添加或修改图形中的每个图层
      foreach (string sLayerName in sLayerNames)
      {
          LayerTableRecord acLyrTblRec;
```

```
            if (acLyrTbl.Has(sLayerName) == false)
            {
                acLyrTblRec = new LayerTableRecord();

                // 设置图层名
                acLyrTblRec.Name = sLayerName;

                // 以写模式升级打开图层表升级打开图层表
                if (acLyrTbl.IsWriteEnabled == false) acLyrTbl.UpgradeOpen();

                // 添加新图层到图层表，记录事务
                acLyrTbl.Add(acLyrTblRec);
                acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
            }
            else
            {
                // 如果图层已存在，则以写模式打开
                acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                                OpenMode.ForWrite) as
LayerTableRecord;
            }

            // 设置图层颜色
            acLyrTblRec.Color = acColors[nCnt];

            nCnt = nCnt + 1;
        }

        // 提交修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub SetLayerColor()
    Dim layerObj As AcadLayer

    ' 定义一个图层名数组
    Dim sLayerNames(2) As String
    sLayerNames(0) = "ACIRed"
    sLayerNames(1) = "TrueBlue"
    sLayerNames(2) = "ColorBookYelow"
```

```
    ' 定义一个颜色数组
    Dim colorObj(2) As New AcadAcCmColor

    colorObj(0).ColorMethod = acColorMethodByACI
    colorObj(0).ColorIndex = acRed
    Call colorObj(1).SetRGB(23, 54, 232)
    Call colorObj(2).SetColorBookColor("PANTONE(R) pastel coated", _
                                       "PANTONE Yellow 0131 C")

    Dim nCnt As Integer

    ' 遍历图层名，创建新图层
    For Each sLayerName In sLayerNames
       Set layerObj = ThisDrawing.Layers.Add(sLayerName)
       layerObj.TrueColor = colorObj(nCnt)

       nCnt = nCnt + 1
    Next
End Sub
```

3.5.1.8 指定图层线型

定义图层时，线型提供了传达视觉信息的另一个途径。线型是划线、点及空格的重复图案，可以使用线型来区分不同线条的作用。

线型名称和定义描述了不同线型的点划序列、划线或空白的相对长度、包含的文字或形状的特性等。

使用 Linetype 属性来设置图层的线型，该属性以线型名作为输入参数。

注：将一个线型赋给图层前，必须先在图形中定义这个线型。更多关于使用线型的内容，参见（§3.5.3 *使用线型*）。

设置图层线型

下面示例创建一个名为“ABC”的新图层，并设置该图层线型为“Center”。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("SetLayerLinetype")> _
Public Sub SetLayerLinetype()
```

```
'' 获取当前文档和数据库
Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
Dim acCurDb As Database = acDoc.Database

'' 启动事务
Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

    '' 以读模式打开图层表
    Dim acLyrTbl As LayerTable
    acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                 OpenMode.ForRead)

    Dim sLayerName As String = "ABC"
    Dim acLyrTblRec As LayerTableRecord

    If acLyrTbl.Has(sLayerName) = False Then
        acLyrTblRec = New LayerTableRecord()

        acLyrTblRec.Name = sLayerName

        '' 以写模式升级打开图层表
        acLyrTbl.UpgradeOpen()

        '' 添加新图层到图层表，记录事务
        acLyrTbl.Add(acLyrTblRec)
        acTrans.AddNewlyCreatedDBObject(acLyrTblRec, True)
    Else
        acLyrTblRec = acTrans.GetObject(acLyrTbl(sLayerName), _
                                        OpenMode.ForRead)
    End If

    '' 以读模式打开图层表
    Dim acLinTbl As LinetypeTable
    acLinTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                 OpenMode.ForRead)

    If acLinTbl.Has("Center") = True Then
        '' 以写模式升级打开图层表记录
        acLyrTblRec.UpgradeOpen()

        '' 设置图层线型
        acLyrTblRec.LinetypeObjectId = acLinTbl("Center")
    End If
```

```
      '' 保存修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("SetLayerLinetype")]
public static void SetLayerLinetype()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "ABC";
      LayerTableRecord acLyrTblRec;

      if (acLyrTbl.Has(sLayerName) == false)
      {
          acLyrTblRec = new LayerTableRecord();

          // 给图层名称赋值
          acLyrTblRec.Name = sLayerName;

          // 以写模式升级打开图层表
          acLyrTbl.UpgradeOpen();

          // 添加新图层到图层表，记录事务
          acLyrTbl.Add(acLyrTblRec);
          acTrans.AddNewlyCreatedDBObject(acLyrTblRec, true);
      }
      else
      {
```

```
            acLyrTblRec = acTrans.GetObject(acLyrTbl[sLayerName],
                                            OpenMode.ForRead) as LayerTableRecord;
        }

        // 以读模式打开图层表
        LinetypeTable acLinTbl;
        acLinTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                     OpenMode.ForRead) as LinetypeTable;

        if (acLinTbl.Has("Center") == true)
        {
            // 以写模式升级打开图层表记录
            acLyrTblRec.UpgradeOpen();

            // 设置图层线型
            acLyrTblRec.LinetypeObjectId = acLinTbl["Center"];
        }

        // 保存修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub SetLayerLinetype()
    On Error Resume Next
    Dim layerObj As AcadLayer

    Set layerObj = ThisDrawing.Layers.Add("ABC")
    layerObj.Linetype = "Center"
End Sub
```

3.5.1.9 删除图层

可以在绘图过程中随时删除图层。不能删除当前图层、图层 0、依赖外部参考的图层，以及包含有对象的图层。

删除图层使用 Erase()方法。建议先使用 Purge()函数检查要删除的图层是否可以被清除，该函数同时还确认要删除的图层不是图层 0、Defpoints 图层或当前图层。

注：即使不含可见对象，也不能删除块定义所引用的图层、以及名为 Defpoints 的特殊图层。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("EraseLayer")> _
Public Sub EraseLayer()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开图层表
      Dim acLyrTbl As LayerTable
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId, _
                                   OpenMode.ForRead)

      Dim sLayerName As String = "ABC"

      If acLyrTbl.Has(sLayerName) = True Then
          '' 检查删除图层是否安全
          Dim acObjIdColl As ObjectIdCollection = New ObjectIdCollection()
          acObjIdColl.Add(acLyrTbl(sLayerName))
          acCurDb.Purge(acObjIdColl)

          If acObjIdColl.Count > 0 Then
              Dim acLyrTblRec As LayerTableRecord
              acLyrTblRec = acTrans.GetObject(acObjIdColl(0), OpenMode.ForWrite)

              Try
                  '' 删除未使用的图层
                  acLyrTblRec.Erase(True)

                  '' 保存修改，关闭事务
                  acTrans.Commit()
              Catch Ex As Autodesk.AutoCAD.Runtime.Exception
                  '' 不能删除
                  Application.ShowAlertDialog("Error:\n" + Ex.Message)
              End Try
          End If
      End If
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("EraseLayer")]
public static void EraseLayer()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开图层表
      LayerTable acLyrTbl;
      acLyrTbl = acTrans.GetObject(acCurDb.LayerTableId,
                                   OpenMode.ForRead) as LayerTable;

      string sLayerName = "ABC";

      if (acLyrTbl.Has(sLayerName) == true)
      {
          // 检查删除图层是否安全
          ObjectIdCollection acObjIdColl = new ObjectIdCollection();
          acObjIdColl.Add(acLyrTbl[sLayerName]);
          acCurDb.Purge(acObjIdColl);

          if (acObjIdColl.Count > 0)
          {
              LayerTableRecord acLyrTblRec;
              acLyrTblRec = acTrans.GetObject(acObjIdColl[0],
                                              OpenMode.ForWrite) as
LayerTableRecord;

              try
              {
                  // 删除未引用图层
                  acLyrTblRec.Erase(true);

                  // 保存修改，关闭事务
                  acTrans.Commit();
              }
```

```
                catch (Autodesk.AutoCAD.Runtime.Exception Ex)
                {
                    //不能删除
                    Application.ShowAlertDialog("Error:\n" + Ex.Message);
                }
            }
        }
    }
}
```

VBA/ActiveX 代码参考

```
Sub EraseLayer()
    On Error Resume Next
    Dim layerObj As AcadLayer
    Set layerObj = ThisDrawing.Layers("ABC")
    layerObj.Delete
End Sub
```

3.5.2 使用颜色

我们可以使用 Color 属性或 ColorIndex 属性给图形中的每个对象设置颜色。ColorIndex 属性接受 0-256 的 AutoCAD 颜色索引值(ACI)。Color 属性则可以使用 ACI 值、真彩色值、或配色系统值来给对象设置颜色。我们使用 Colors 命名空间的 Color 对象来修改 Color 属性的值。

Color 对象提供了一个 SetRGB()方法，允许我们从数百万个红绿蓝颜色组合中选择颜色。Color 对象还包含指定颜色名称、配色系统、颜色索引、及颜色值的方法和属性。

我们还可以给图层设置颜色。如果要让对象继承所在图层的颜色，只需将对象的颜色设置为 ByLayer 即可（通过将 ACI 值设置为 256 实现）。不同的对象和图层可以拥有相同的颜色编码。我们可以给彩色绘图仪的不同彩笔分别赋予一个颜色编码，或者在图形中使用颜色编码来识别特定的对象，即使我们在屏幕上看不到所设的颜色。

3.5.2.1 给对象指定颜色值

下面的示例代码创建 4 个圆，并使用四种不同方法为每个圆指定不同的颜色。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.Colors
```

```
<CommandMethod("SetObjectColor")> _
Public Sub SetObjectColor()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 为图层定义颜色数组
      Dim acColors(2) As Color
      acColors(0) = Color.FromColorIndex(ColorMethod.ByAci, 1)
      acColors(1) = Color.FromRgb(23, 54, 232)
      acColors(2) = Color.FromNames("PANTONE Yellow 0131 C", _
                                    "PANTONE(R) pastel coated")

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建一个圆并将其 ACI 值设为 4
      Dim acPt As Point3d = New Point3d(0, 3, 0)
      Dim acCirc As Circle = New Circle()
      acCirc.Center = acPt
      acCirc.Radius = 1
      acCirc.ColorIndex = 4

      acBlkTblRec.AppendEntity(acCirc)
      acTrans.AddNewlyCreatedDBObject(acCirc, True)

      Dim nCnt As Integer = 0

      While (nCnt < 3)
          '' 克隆圆
          Dim acCircCopy As Circle
          acCircCopy = acCirc.Clone()
```

```
            '' 将拷贝沿 Y 轴移动（Y 坐标+3）
            acPt = New Point3d(acPt.X, acPt.Y + 3, acPt.Z)
            acCircCopy.Center = acPt

            '' 新颜色
            acCircCopy.Color = acColors(nCnt)

            acBlkTblRec.AppendEntity(acCircCopy)
            acTrans.AddNewlyCreatedDBObject(acCircCopy, True)

            nCnt = nCnt + 1
        End While

        '' 保存修改，关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Colors;

[CommandMethod("SetObjectColor")]
public static void SetObjectColor()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 为图层定义颜色数组
      Color[] acColors = new Color[3];
      acColors[0] = Color.FromColorIndex(ColorMethod.ByAci, 1);
      acColors[1] = Color.FromRgb(23, 54, 232);
      acColors[2] = Color.FromNames("PANTONE Yellow 0131 C",
                                    "PANTONE(R) pastel coated");

      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
```

```
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开 Block 表记录 Model 空间
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // 创建一个圆并将其 ACI 值设为 4
        Point3d acPt = new Point3d(0, 3, 0);
        Circle acCirc = new Circle();
        acCirc.Center = acPt;
        acCirc.Radius = 1;
        acCirc.ColorIndex = 4;

        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        int nCnt = 0;
        while (nCnt < 3)
        {
            // 创建圆的拷贝
            Circle acCircCopy;
            acCircCopy = acCirc.Clone() as Circle;

            // 沿 Y 轴移动复本
            acPt = new Point3d(acPt.X, acPt.Y + 3, acPt.Z);
            acCircCopy.Center = acPt;

            // 给复本设置新颜色
            acCircCopy.Color = acColors[nCnt];

            acBlkTblRec.AppendEntity(acCircCopy);
            acTrans.AddNewlyCreatedDBObject(acCircCopy, true);

            nCnt = nCnt + 1;
        }

        // 提交修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub SetObjectColor()
    ' 定义颜色数组
    Dim colorObj(2) As New AcadAcCmColor

    colorObj(0).ColorMethod = acColorMethodByACI
    colorObj(0).ColorIndex = acRed
    Call colorObj(1).SetRGB(23, 54, 232)
    Call colorObj(2).SetColorBookColor("PANTONE(R) pastel coated", _
                                       "PANTONE Yellow 0131 C")

    ' 定义圆心
    Dim centerPt(0 To 2) As Double
    centerPt(0) = 0: centerPt(1) = 3: centerPt(2) = 0

    ' 创建一个圆，颜色 ACI 值为 4
    Dim circleObj As AcadCircle
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPt, 1)
    circleObj.color = acCyan

    Dim nCnt As Integer
    ' 再创建 3 个圆
    While (nCnt < 3)
       ' 创建圆的复本
       Dim circleObjCopy As AcadCircle
       Set circleObjCopy = circleObj.Copy

       ' 沿 Y 轴移动复本
       centerPt(1) = centerPt(1) + 3
       circleObjCopy.Center = centerPt

       ' 赋新颜色
       circleObjCopy.TrueColor = colorObj(nCnt)

       nCnt = nCnt + 1
    Wend
End Sub
```

3.5.2.2 通过数据库设置当前颜色

本例通过 Database 对象的 Cecolor 属性设置当前颜色。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Colors

<CommandMethod("SetColorCurrent")> _
Public Sub SetColorCurrent()
  '' 获取当前文档
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' 设置当前颜色
  acDoc.Database.Cecolor = Color.FromColorIndex(ColorMethod.ByLayer, 256)
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Colors;

[CommandMethod("SetColorCurrent")]
public static void SetColorCurrent()
{
  // 获取当前文档
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // 设置当前颜色
  acDoc.Database.Cecolor = Color.FromColorIndex(ColorMethod.ByLayer, 256);
}
```

3.5.2.3 使用系统变量 CECOLOR 设置当前颜色

本例使用系统变量 CECOLOR 将当前颜色设置为红色（Red）。

VB.NET

```
Application.SetSystemVariable("CECOLOR", "1")
```

C#

```
Application.SetSystemVariable("CECOLOR", "1");
```

⊟ VBA/ActiveX 代码参考

```
ThisDrawing.SetVariable "CECOLOR", "1"
```

附：关于系统变量 CECOLOR （摘自《AutoCAD 用户指南》）

类型： 字符串

保存位置： 图形

初始值： BYLAYER

功能：设置新对象的颜色。

有效值包括：

1、BYLAYER 或 BYBLOCK ；

2、AutoCAD 颜色索引 (ACI)：介于 1 和 255 之间的整数值，或前七种颜色中的颜色名称 （1 红、2 黄、3 绿、4 青、5 蓝、6 洋红、7 白/黑）；

3、真彩色：介于 000 和 255 之间的 RGB 或 HSL 值，格式为“RGB:130,200,240”；

4、配色系统：标准 PANTONE 配色系统或自定义配色系统、DIC 颜色手册或 RAL 颜色集里的定义，例如“DIC COLOR GUIDE(R)$DIC 43”。

3.5.3 使用线型

线型是点、划和空格的重复图案。复杂的线型是各种符号的重复图案。要使用线型，需先将该线型加载到图形里。加载前，线型库文件（.LIN 文件）里必须有该线型的定义。向图形加载线型使用 Database 对象的 LoadLineTypeFile 成员函数。

注：不要将 AutoCAD 内部使用的线型与某些绘图仪提供的硬件线型相混淆。这两种类型的虚线产生的效果相似。不要同时使用这两种类型，否则将产生不可预知的结果。

将线型加载到 AutoCAD

本例试图从线型文件 acad.lin 加载线型“CENTER”，如果该线型已存在，或线型文件不存在，则显示消息。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("LoadLinetype")> _
Public Sub LoadLinetype()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开线型表
      Dim acLineTypTbl As LinetypeTable
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                       OpenMode.ForRead)

      Dim sLineTypName As String = "Center"

      If acLineTypTbl.Has(sLineTypName) = False Then
          '' 装载 Center 线型
          acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin")
      End If

      '' 保存修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("LoadLinetype")]
public static void LoadLinetype()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开线型表
      LinetypeTable acLineTypTbl;
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                       OpenMode.ForRead) as LinetypeTable;

      string sLineTypName = "Center";
```

```
        if (acLineTypTbl.Has(sLineTypName) == false)
        {
            // 装载 Center 线型
            acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin");
        }

        // 提交修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub LoadLinetype()
    On Error GoTo ERRORHANDLER

    Dim linetypeName As String
    linetypeName = "CENTER"

    '从线型文件 acad.lin 装载 Center 线型
    ThisDrawing.Linetypes.Load linetypeName, "acad.lin"
    Exit Sub

ERRORHANDLER:
    MsgBox Err.Description

End Sub
```

3.5.3.1 设置活动线型

要使用线型，必须先将其置为活动线型。所有新创建的对象均使用活动线型绘制。对象的线型可由两种不同方法获得：直接指定或继承。可以直接给对象指定一个线型，同时覆盖掉对象所在图层的线型。另外，对象可以继承其所在图层的线型，方法是将对象的 Linetype 属性或 LinetypeId 属性设置成 ByLayer 线型。

注：不能将依赖外部参照的线型设置为活动线型。

每个图形里都存在三个线型，他们是 BYBLOCK、 BYLAYER 和 CONTINUOUS。这三个线型均可以通过线型表对象访问，或使用 SymbolUtilityServices 对象提供的方法访问。下列方法允许我们获取这些默认线型的对象 ID：

- **GetLinetypeByBlockId** - 返回 BYBLOCK 线型的对象 ID；
- **GetLinetypeByLayerId** - 返回 BYLAYER 线型的对象 ID；
- **GetLinetypeContinuousId** - 返回 CONTINUOUS 线型的对象 ID；

关于激活线型的更多内容，见《AutoCAD 用户指南》中的“设置当前线型”。

为对象指定线型

下面的例子创建一个圆并将圆的线型指定为“Center”。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SetObjectLinetype")> _
Public Sub SetObjectLinetype()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开线型表
      Dim acLineTypTbl As LinetypeTable
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                       OpenMode.ForRead)

      Dim sLineTypName As String = "Center"

      If acLineTypTbl.Has(sLineTypName) = False Then
          acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin")
      End If

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建一个圆
      Dim acCirc As Circle = New Circle()
      acCirc.Center = New Point3d(2, 2, 0)
      acCirc.Radius = 1
      acCirc.Linetype = sLineTypName
```

```
        acBlkTblRec.AppendEntity(acCirc)
        acTrans.AddNewlyCreatedDBObject(acCirc, True)

        '' 保存修改，关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SetObjectLinetype")]
public static void SetObjectLinetype()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开线型表
      LinetypeTable acLineTypTbl;
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                       OpenMode.ForRead) as LinetypeTable;

      string sLineTypName = "Center";

      if (acLineTypTbl.Has(sLineTypName) == false)
      {
          acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin");
      }

      // 以读模式打开 Block 表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 Block 表记录 Model 空间
      BlockTableRecord acBlkTblRec;
```

```
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // 创建一个圆
        Circle acCirc = new Circle();
        acCirc.Center = new Point3d(2, 2, 0);
        acCirc.Radius = 1;
        // 设置线型
        acCirc.Linetype = sLineTypName;

        acBlkTblRec.AppendEntity(acCirc);
        acTrans.AddNewlyCreatedDBObject(acCirc, true);

        // 保存修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub SetObjectLinetype()
    ' 装载 Center 线型
    Dim linetypeName As String
    linetypeName = "Center"

    On Error Resume Next
    ThisDrawing.Linetypes.Load linetypeName, "acad.lin"

    Dim centerPt(0 To 2) As Double
    centerPt(0) = 0: centerPt(1) = 3: centerPt(2) = 0

    Dim circleObj As AcadCircle
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPt, 1)
    ' 设置线型
    circleObj.Linetype = linetypeName

    circleObj.Update
End Sub
```

通过数据库设置当前线型

本例通过 Database 对象的 Celtype 属性将一个线型设置为当前线型。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("SetLinetypeCurrent")> _
Public Sub SetLinetypeCurrent()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开线型表
      Dim acLineTypTbl As LinetypeTable
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                       OpenMode.ForRead)

      Dim sLineTypName As String = "Center"

      If acLineTypTbl.Has(sLineTypName) = True Then
          '' 将 Center 线型设置为当前线型
          acCurDb.Celtype = acLineTypTbl(sLineTypName)

          '' 保存修改
          acTrans.Commit()
      End If

      '' 关闭事务
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("SetLinetypeCurrent")]
public static void SetLinetypeCurrent()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 以读模式打开线型表
        LinetypeTable acLineTypTbl;
        acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                         OpenMode.ForRead) as LinetypeTable;

        string sLineTypName = "Center";

        if (acLineTypTbl.Has(sLineTypName) == true)
        {
            // 将 Center 线型设置为当前线型
            acCurDb.Celtype = acLineTypTbl[sLineTypName];

            // 保存修改
            acTrans.Commit();
        }

        // 关闭事务
    }
}
```

VBA/ActiveX 代码参考

```
ThisDrawing.ActiveLinetype = ThisDrawing.Linetypes.Item("Center")
```

通过系统变量 CELTYPE 设置当前线型

本例通过系统变量 CELTYPE 将一个线型设置为当前线型。

VB.NET

```
Application.SetSystemVariable("CELTYPE", "Center")
```

C#

```
Application.SetSystemVariable("CELTYPE", "Center");
```

VBA/ActiveX 代码参考

```
ThisDrawing.SetVariable "CELTYPE", "Center"
```

3.5.3.2 重命名线型

给一个线型重新命名，使用 Name 属性。重命名一个线型时，我们只是重新命名了图形中的线型定义。线型库文件（.LIN 文件）中的线型名称不会随着更新。

3.5.3.3 删除线型

删除一个线型，使用 Erase()方法。可以在绘图过程的任何时候删除一个线型；不过，不能删除线型 BYLAYER、BYBLOCK、CONTINUOUS，以及当前线型和依赖外部参考的线型。还有，块定义所引用的线型也不能删除，不管有没有对象使用。

3.5.3.4 修改线型说明

线型可以有与之关联的说明文字。该说明文字提供了该线型的 ASCII 描述。我们可以使用 AsciiDescription 属性来设置或修改线型说明。

线型说明最多 47 个字符。说明可以是一个注释，也可以使用一系列下划线、点、虚线和空格来简单地表示出线型图案。

修改线型说明

下面的示例演示如何修改当前线型的说明。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ChangeLinetypeDescription")> _
Public Sub ChangeLinetypeDescription()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以写模式打开当前线型的线型表记录
      Dim acLineTypTblRec As LinetypeTableRecord
      acLineTypTblRec = acTrans.GetObject(acCurDb.Celtype, _
                                          OpenMode.ForWrite)

      '' 修改当前线型的说明
      acLineTypTblRec.AsciiDescription = "Exterior Wall"
      '' 保存修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ChangeLinetypeDescription")]
public static void ChangeLinetypeDescription()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Linetype table record of the current linetype for write
      // 以写模式打开当前线型的线型表记录
      LinetypeTableRecord acLineTypTblRec;
      acLineTypTblRec = acTrans.GetObject(acCurDb.Celtype,
                              OpenMode.ForWrite) as LinetypeTableRecord;

      //修改当前线型的说明
      acLineTypTblRec.AsciiDescription = "Exterior Wall";

      // 保存修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
ThisDrawing.ActiveLinetype.Description = "Exterior Wall"
```

3.5.3.5 指定线型比例

我们可以指定所创建对象的线型的比例。比例越小，每图形单位内生成的线型图案就越多。默认情况下，AutoCAD 使用全局线型比例 1.0，等于一个图形单位。我们可以为所有图形对象和属性参考修改线型比例。

系统变量 CELTSCALE 用来给新创建的对象设置线型比例。系统变量 LTSCALE 用来修改现有对象以及新对象的全局线型比例。每个对象的 LinetypeScale 属性用来修改该对象的线型比例。对象显示的线型比例基于对象自己的线型比例乘以全局线型比例的结果。

更多关于线型比例的内容，见《AutoCAD 用户指南》中的“控制线型比例”。

修改对象的线型比例

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SetObjectLinetypeScale")> _
Public Sub SetObjectLinetypeScale()
  '' 获取当前文档和数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 保存当前线型
      Dim acObjId As ObjectId = acCurDb.Celtype

      '' 设置全局线型比例
      acCurDb.Ltscale = 3

      '' 以读模式打开线型表
      Dim acLineTypTbl As LinetypeTable
      acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId, _
                                       OpenMode.ForRead)

      Dim sLineTypName As String = "Border"

      If acLineTypTbl.Has(sLineTypName) = False Then
          acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin")
      End If

      '' 将 Border 线型设置为当前线型
      acCurDb.Celtype = acLineTypTbl(sLineTypName)

      '' 以读模式打开 Block 表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开 Block 表记录 Model 空间
```

```
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      ''创建一个圆并将其线型比例设置为全尺寸的一半
      Dim acCirc1 As Circle = New Circle()
      acCirc1.Center = New Point3d(2, 2, 0)
      acCirc1.Radius = 4
      acCirc1.LinetypeScale = 0.5

      acBlkTblRec.AppendEntity(acCirc1)
      acTrans.AddNewlyCreatedDBObject(acCirc1, True)
      '' 再创建一个圆
      Dim acCirc2 As Circle = New Circle()
      acCirc2.Center = New Point3d(12, 2, 0)
      acCirc2.Radius = 4
      acBlkTblRec.AppendEntity(acCirc2)
      acTrans.AddNewlyCreatedDBObject(acCirc2, True)

      '' 恢复原来的活动线型
      acCurDb.Celtype = acObjId
      '' 保存修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SetObjectLinetypeScale")]
public static void SetObjectLinetypeScale()
{
  // 获取当前文档和数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 保存当前线型
      ObjectId acObjId = acCurDb.Celtype;
```

```
// 设置全局线型比例
acCurDb.Ltscale = 3;

// 以读模式打开线型表
LinetypeTable acLineTypTbl;
acLineTypTbl = acTrans.GetObject(acCurDb.LinetypeTableId,
                                 OpenMode.ForRead) as LinetypeTable;

string sLineTypName = "Border";

if (acLineTypTbl.Has(sLineTypName) == false)
{
    acCurDb.LoadLineTypeFile(sLineTypName, "acad.lin");
}

//将 Border 线型设置为当前线型;
acCurDb.Celtype = acLineTypTbl[sLineTypName];

// 以读模式打开 Block 表
BlockTable acBlkTbl;
acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                             OpenMode.ForRead) as BlockTable;

// 以写模式打开 Block 表记录 Model 空间
BlockTableRecord acBlkTblRec;
acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                             OpenMode.ForWrite) as BlockTableRecord;

// 创建一个圆并将其线型比例设置为全尺寸的一半
Circle acCirc1 = new Circle();
acCirc1.Center = new Point3d(2, 2, 0);
acCirc1.Radius = 4;
acCirc1.LinetypeScale = 0.5;

acBlkTblRec.AppendEntity(acCirc1);
acTrans.AddNewlyCreatedDBObject(acCirc1, true);

// 再创建一个圆
Circle acCirc2 = new Circle();
acCirc2.Center = new Point3d(12, 2, 0);
acCirc2.Radius = 4;

acBlkTblRec.AppendEntity(acCirc2);
```

```
      acTrans.AddNewlyCreatedDBObject(acCirc2, true);

      // 恢复原来的活动线型
      acCurDb.Celtype = acObjId;

      // 保存修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub SetObjectLinetypeScale()
    ' 保存当前线型
    Dim currLineType As AcadLineType
    Set currLineType = ThisDrawing.ActiveLinetype

    ' 设置全局线型比例
    ThisDrawing.SetVariable "LTSCALE", 3

    ' 加载 Border 线型
    On Error Resume Next
    If Not IsObject(ThisDrawing.Linetypes.Item("Border")) Then
      ThisDrawing.Linetypes.Load "Border", "acad.lin"
    End If

    ThisDrawing.ActiveLinetype = ThisDrawing.Linetypes.Item("BORDER")

    ' 在模型空间创建一个圆
    Dim center(0 To 2) As Double
    center(0) = 2: center(1) = 2: center(2) = 0

    Dim circleObj1 As AcadCircle
    Set circleObj1 = ThisDrawing.ModelSpace.AddCircle(center, 4)

    ' 设置圆的线型比例为全尺寸的一半
    circleObj1.LinetypeScale = 0.5
    circleObj1.Update

    Dim circleObj2 As AcadCircle
    center(0) = center(0) + 10
    Set circleObj2 = ThisDrawing.ModelSpace.AddCircle(center, 4)
    circleObj2.Update

    ' 恢复原来的活动线型
```

```
    ThisDrawing.ActiveLinetype = currLineType
End Sub
```

3.6 保存和恢复图层状态

我们可以保存图形中的图层状态，并在随后将其恢复。这使得我们能够很容易在完成绘图过程中，或在打印图形时，返回到不同阶段所有图层的指定设置状态。

图层状态包括图层是否打开、冻结、锁定、可打印、在新视口是否自动冻结，以及图层的颜色、线型、线宽、打印样式等（状态信息）。我们可以指定要保存哪些设置，并且可以给一个图形保存几组不同的图层设置。

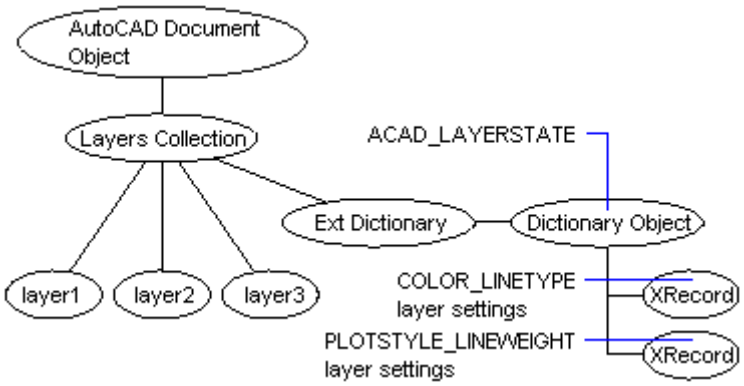
LayerStateManager 对象用来保存和恢复图层状态。

3.6.1 了解 AutoCAD 如何保存图层状态

AutoCAD 将图层设置信息保存在 Layers 表对象的扩展字典里。我们第一次保存图层状态时，AutoCAD 会进行下列操作：

- 在 Layers 表上创建一个扩展字典(Ext Dictionary)；
- 在扩展字典里创建一个名为 ACAD_LAYERSTATE 的 Dictionary 对象；
- 将图形中每个图层的属性存储在 ACAD_LAYERSTATE 字典的 XRecord 对象中。AutoCAD 将所有的图层设置存储在 XRecord 里，但可以识别出你选择保存的那些设置。当恢复图层设置时，AutoCAD 只恢复你选择保存的那些设置。

ACAD_LAYERSTATE 字典里的每个 XRecord 对象对应一个图层设置。当我们保存图形中的另一个图层设置时，AutoCAD 就会新创建一个 XRecord 对象来描述所保存的设置，并将该 XRecord 对象存储到 ACAD_LAYERSTATE 字典里。下图为这一过程的图解。



在使用图层状态时，我们不需要（也不要试图）去直接操作那些实体，使用 LayerStateManager 对象的方法来访问字典就可以了。我们一旦获得了对字典的引用，就可以遍历代表 DBDictionaryEntry 对象的每一个实体。

列出图形中保存的图层状态

如果图层状态已经保存在当前图形内，下面的代码会列出保存的所有图层状态的名称：

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ListLayerStates")>_
Public Sub ListLayerStates()
  ''获取当前文档及数据库
  Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
   Dim acCurDb As Database = acDoc.Database

  ''启动事务
  Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

          Dim acLyrStMan As LayerStateManager
          acLyrStMan = acCurDb.LayerStateManager

          Dim acDbDict As DBDictionary
          acDbDict =acTrans.GetObject(acLyrStMan.LayerStatesDictionaryId(True),
_
                                      OpenMode.ForRead)
```

```
        Dim sLayerStateNames As String =""

        For Each acDbDictEnt As DBDictionaryEntryIn acDbDict
                sLayerStateNames = sLayerStateNames& vbLf & acDbDictEnt.Key
        Next

        Application.ShowAlertDialog("Thesaved layer settings in this drawing
are:" & _
                                                sLayerStateNames)

  '' 关闭事务
  End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("ListLayerStates")]
public static void ListLayerStates()
{
  //获取当前文档及数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  //启动事务
  using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
  {
        LayerStateManager acLyrStMan;
        acLyrStMan = acCurDb.LayerStateManager;

        DBDictionary acDbDict;
        acDbDict =acTrans.GetObject(acLyrStMan.LayerStatesDictionaryId(true),
                                    OpenMode.ForRead) as DBDictionary;

        string sLayerStateNames = "";

        //遍历
        foreach (DBDictionaryEntry acDbDictEnt in acDbDict)
        {
                sLayerStateNames = sLayerStateNames +"\n" + acDbDictEnt.Key;
```

```
            }

            Application.ShowAlertDialog("The saved layer settings in this drawing
are:" + sLayerStateNames);

   //关闭事务
   }
}
```

VBA/ActiveX 代码参考

```
Sub ListLayerStates()
       On Error Resume Next
       Dim oLSMDict As AcadDictionary
       Dim XRec As Object
       Dim layerstateNames As String

       layerstateNames = ""
       ' 获取 ACAD_LAYERSTATES 字典（Layers 对象的扩展字典）
       Set oLSMDict = ThisDrawing.Layers. _
                     GetExtensionDictionary.Item("ACAD_LAYERSTATES")

       '列出保存的每个图层设置的名称（图层设置保存在字典的 Xrecords 记录里）

       For Each XRec In oLSMDict
             layerstateNames = layerstateNames + XRec.Name + vbCrLf
       Next XRec

       MsgBox "The saved layer settings in this drawing are: " + _
                     vbCrLf + layerstateNames
End Sub
```

3.6.2 用 LayerStateManager 管理图层状态

LayerStateManager 对象提供了一组创建和操作保存的图层状态的方法。这些方法有：

DeleteLayerState()

删除一个保存的图层状态；

ExportLayerState()

将制定的图层状态导出为 LAS 文件；

ImportLayerState()

从 LAS 文件导入图层状态；

ImportLayerStateFromDb()

从别的数据库导入图层状态；

RenameLayerState()

重命名保存的图层状态；

RestoreLayerState()

恢复当前图形中指定的图层状态；

SaveLayerState()

保存指定的图层状态及其属性；

数据库的 LayerStateManager 对象可以通过使用 Database 对象的 LayerStateManager 属性来访问。

VB.NET

```
Dim acDoc As Document= Application.DocumentManager.MdiActiveDocument
Dim acCurDb As Database = acDoc.Database

Dim acLyrStMan As LayerStateManager
acLyrStMan =acCurDb.LayerStateManager
```

C#

```
Document acDoc =Application.DocumentManager.MdiActiveDocument;
Database acCurDb =acDoc.Database;

LayerStateManager acLyrStMan;
acLyrStMan =acCurDb.LayerStateManager;
```

VBA/ActiveX 代码参考

声明 LayerStateManager 对象之后，在访问对象的方法之前必须将一个数据库与之关联。将数据库与 LayerStateManager 对象关联用 SetDatabase 方法。

```
Dim oLSM As AcadLayerStateManager
Set oLSM =ThisDrawing.Application. _
              GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

oLSM.SetDatabase ThisDrawing.Database
```

3.6.2.1 保存图层状态

使用 SaveLayerState()方法来保存图形中的一组图层设置。SaveLayerState()方法需要 3 个参数：第一个参数为一个字符串，用于命名要保存的图层状态；第二个参数表示要保存的图层属性。使用枚举 LayerStateMasks 定义的常量来识别要保存的图层设置。

下表列出了 LayerStateMasks 枚举定义的部分常量：

LayerStateMasks(图层状态掩码)常量	
枚举常量	**图层属性**
Color	图层颜色
CurrentViewport	当前视口图层冻结或解冻
Frozen	图层冻结或解冻
LastRestored	最近恢复的图层
LineType	图层线型
LineWeight	图层线宽
Locked	图层锁定或解锁
NewViewport	新视口图层冻结或解冻
None	无图层设置
On	图层打开或关闭

Plot	图层打印打开或关闭
PlotStyle	图层的打印样式

若要指定多个属性，只需将这些枚举常量加在一起即可。

SaveLayerState()方法的第三个参数是要保存的图层设置所在的视口的 ObjectId。使用 ObjectId.Null 表示不指定视口。

如果试图将图层状态保存在一个已经存在的名字下，会返回一个错误。必须将现有的图层状态改名，或干脆删除掉，才能重新使用这个名字。

保存图层的颜色和线型设置

下列代码将图形中当前图层的颜色和线型设置保存到名为 ColorLinetype 的图层状态中。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("SaveLayerColorAndLinetype")>_
Public Sub SaveLayerColorAndLinetype()
  '' 获取当前文档
  Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String ="ColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) =False Then
          acLyrStMan.SaveLayerState(sLyrStName, _
                                    LayerStateMasks.Color + _
                                    LayerStateMasks.LineType, _
                                     ObjectId.Null)
  End If
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("SaveLayerColorAndLinetype")]
public static void SaveLayerColorAndLinetype()
{
  // 获取当前文档
  Document acDoc =Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan =acDoc.Database.LayerStateManager;

  string sLyrStName ="ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) ==false)
  {
        acLyrStMan.SaveLayerState(sLyrStName,
                    LayerStateMasks.Color | LayerStateMasks.LineType,
                    ObjectId.Null);
  }
}
```

VBA/ActiveX 代码参考

```
Sub SaveLayerColorAndLinetype()
      Dim oLSM As AcadLayerStateManager

      ' 访问 LayerStateManager 对象
      Set oLSM = ThisDrawing.Application. _
                  GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

      ' 关联 LayerStateManager 与当前数据库
      oLSM.SetDatabase ThisDrawing.Database
      oLSM.Save "ColorLinetype",acLsColor + acLsLineType
End Sub
```

3.6.2.2 重命名图层状态

RenameLayerState()方法用来给图形中已保存的图层状态改名。下列代码将图层设置 ColorLinetype 重命名为 OldColorLinetype。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("RenameLayerState")>_
Public Sub RenameLayerState()
  '' 获取当前文档
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String ="ColorLinetype"
  Dim sLyrStNewName As String ="OldColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) =True And _
        acLyrStMan.HasLayerState(sLyrStNewName) =False Then
          acLyrStMan.RenameLayerState(sLyrStName,sLyrStNewName)
  End If
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("RenameLayerState")]
public static void RenameLayerState()
{
  // 获取当前文档
  Document acDoc =Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStName ="ColorLinetype";
  string sLyrStNewName ="OldColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) ==true &&
          acLyrStMan.HasLayerState(sLyrStNewName)== false)
  {
```

```
            acLyrStMan.RenameLayerState(sLyrStName,sLyrStNewName);
    }
}
```

VBA/ActiveX 代码参考

```
Sub RenameLayerState()
       Dim oLSM As AcadLayerStateManager
       Set oLSM = ThisDrawing.Application. _
                     GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

       oLSM.SetDatabase ThisDrawing.Database
       oLSM.Rename "ColorLinetype","OldColorLinetype"
End Sub
```

3.6.2.3 删除图层状态

DeleteLayerState()方法用来从图形中删除已保存的图层状态。下列代码删除保存的名为 ColorLinetype 的图层状态。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("RemoveLayerState")>_
Public Sub RemoveLayerState()
   '' 获取当前文档
   Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument

   Dim acLyrStMan As LayerStateManager
   acLyrStMan = acDoc.Database.LayerStateManager

   Dim sLyrStName As String = "ColorLinetype"

   If acLyrStMan.HasLayerState(sLyrStName) =True Then
           acLyrStMan.DeleteLayerState(sLyrStName)
   End If
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("RemoveLayerState")]
public static void RemoveLayerState()
{
  // 获取当前文档
  Document acDoc =Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan =acDoc.Database.LayerStateManager;

  string sLyrStName ="ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) ==true)
  {
          acLyrStMan.DeleteLayerState(sLyrStName);
  }
}
```

VBA/ActiveX 代码参考

```
Sub RemoveLayerState()
      Dim oLSM As AcadLayerStateManager
      Set oLSM = ThisDrawing.Application. _
                  GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

      oLSM.SetDatabase ThisDrawing.Database
      oLSM.Delete "ColorLinetype"
End Sub
```

3.6.2.4 恢复图层状态

RestoreLayerState()方法用来恢复图层状态中的图层设置，该方法需要 4 个参数：

第一个参数 - 要恢复的图层状态的名称；

第二个参数 - 要恢复的图层状态所在的视口的 ObjectId；

第三个参数 - 是一个整数，定义如何处理图层状态中没有的图层；

第四个参数 - 定义恢复哪些图层属性设置（LayerStateMasks 枚举常量）；

下列值确定如何处理图层状态中没有的图层：

- 0 - 图层状态中没有的图层保持不变；
- 1 - 关闭图层状态中没有的图层；
- 2 - 冻结图层状态中没有的当前视口中的图层；
- 4 - 以覆盖视口的方式恢复图层设置；

注：恢复图层状态时，可以使用上述的多个值的合计来定义不在图层状态中的图层的行为。例如，可以关闭并冻结图层状态中没有的图层。

例如，如果将颜色和线型设置保存到名为“ColorLinetype”的图层状态下，随后修改了这些设置，那么，恢复“ColorLinetype”就会将图层的颜色和线型重新设置为保存“ColorLinetype”时的状态。如果保存“ColorLinetype”后在图形中添加了新图层，当恢复“ColorLinetype”时新图层将不受影响。

恢复图形中图层的颜色和线型设置

假设前面已经将当前图形中图层的颜色和线型设置保存到“ColorLinetype”名下，下列代码会将图形中每个图层的颜色和线型设置恢复到保存“ColorLinetype”时的状态。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("RestoreLayerState")>_
Public Sub RestoreLayerState()
  '' 获取当前文档
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```
  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String ="ColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) =True Then
          acLyrStMan.RestoreLayerState(sLyrStName,_
                                  ObjectId.Null,  1, _
                                  LayerStateMasks.Color + _
                                  LayerStateMasks.LineType)
  End If
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("RestoreLayerState")]
public static void RestoreLayerState()
{
  // 获取当前文档
  Document acDoc =Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan =acDoc.Database.LayerStateManager;

  string sLyrStName ="ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) ==true)
  {
          acLyrStMan.RestoreLayerState(sLyrStName,
                          ObjectId.Null, 1,
                          LayerStateMasks.Color | LayerStateMasks.LineType);
  }
}
```

VBA/ActiveX 代码参考

```
Sub RestoreLayerState()
      Dim oLSM As AcadLayerStateManager
      Set oLSM = ThisDrawing.Application. _
                  GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")
```

```
        oLSM.SetDatabase ThisDrawing.Database
        oLSM.Restore "ColorLinetype"
End Sub
```

3.6.2.5 导出导入保存的图层状态

我们可以导出导入保存的图层状态，以便在别的图形文件中使用相同的图层设置。将保存的图层状态导出到 LAS 文件用 ExportLayerState()方法；将 LAS 文件导入到图形中用 ImportLayerState()方法。

注：导入图层状态并不恢复这些图层；必须在导入后使用 RestoreLayerState()方法来恢复图层状态。

ExportLayerState()方法有两个参数，第一个参数是个字符串，代表要导出的图层状态的名称，第二个参数是一个文件名，用来保存导出的图层状态。如不指定文件路径，文件将保存到与打开的图形文件相同的目录中。如指定的文件名已经存在，将被覆盖。文件名应使用.las 扩展名，AutoCAD 识别该扩展名为导出的图层状态文件。

ImportLayerState()方法的参数是一个字符串，表示要导入的图层状态的文件名。如果要导入的图层状态不在 LAS 文件里，而是在图形文件里，我们可以先打开图形文件，然后使用 ImportLayerStateFromDb()方法从该图形的 Database 对象中导入图层状态。

当导入图层状态时，如果所保存设置里引用的某个属性在要导入的图形中不可用，就会出现错误情况，不过导入操作会使用默认属性继续进行至到结束。例如，如果导出的图层设置的线型在要导入的图形中没有被加载，就会出现错误情况，并且线型会由图形的默认线型代替。我们编程应化解这个错误情况，并在出错的情况下能继续运行。

如果导入文件中定义的图层设置在当前图形中不存在，就会在当前图形中创建这些图层。当使用 RestoreLayerState()方法时，保存图层状态时指定的属性会赋给新图层；而新图层的所有其他属性则使用默认设置。

导出保存的图层设置

下面的示例代码将保存的图层状态导出到文件 *ColorLinetype.las*。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("ExportLayerState")>_
Public Sub ExportLayerState()
  '' 获取当前文档
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStName As String ="ColorLinetype"

  If acLyrStMan.HasLayerState(sLyrStName) =True Then
          acLyrStMan.ExportLayerState(sLyrStName,"c:\my documents\" & _
            sLyrStName & ".las")
  End If
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("ExportLayerState")]
public static void ExportLayerState()
{
  // 获取当前文档
  Document acDoc =Application.DocumentManager.MdiActiveDocument;
  LayerStateManager acLyrStMan;
  acLyrStMan = acDoc.Database.LayerStateManager;

  string sLyrStName ="ColorLinetype";

  if (acLyrStMan.HasLayerState(sLyrStName) ==true)
  {
          acLyrStMan.ExportLayerState(sLyrStName,"c:\\my documents\\" +
                                  sLyrStName + ".las");
  }
}
```

VBA/ActiveX 代码参考

```
Sub ExportLayerStates()
       Dim oLSM As AcadLayerStateManager
       Set oLSM = ThisDrawing.Application. _
             GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")

       oLSM.SetDatabase ThisDrawing.Database
       oLSM.Export "ColorLinetype","c:\my documents\ColorLinetype.las"
End Sub
```

导入保存的图层设置

下面的示例代码从文件 *ColorLinetype.las* 导入图层状态。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("ImportLayerState")>_
Public Sub ImportLayerState()
  '' 获取当前文档
  Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument

  Dim acLyrStMan As LayerStateManager
  acLyrStMan = acDoc.Database.LayerStateManager

  Dim sLyrStFileName As String = "c:\mydocuments\ColorLinetype.las"

  If System.IO.File.Exists(sLyrStFileName) Then
          Try
                  acLyrStMan.ImportLayerState(sLyrStFileName)
          Catch ex AsAutodesk.AutoCAD.Runtime.Exception
                 Application.ShowAlertDialog(ex.Message)
          End Try
  End If
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
```

```
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ImportLayerState")]
public static void ImportLayerState()
{
  // 获取当前文档
  Document acDoc =Application.DocumentManager.MdiActiveDocument;

  LayerStateManager acLyrStMan;
  acLyrStMan =acDoc.Database.LayerStateManager;

  string sLyrStFileName = "c:\\mydocuments\\ColorLinetype.las";

  if (System.IO.File.Exists(sLyrStFileName))
  {
          try
          {
                  acLyrStMan.ImportLayerState(sLyrStFileName);
          }
          catch (Autodesk.AutoCAD.Runtime.Exception ex)
          {
                  Application.ShowAlertDialog(ex.Message);
          }
  }
}
```

VBA/ActiveX 代码参考

```
Sub ImportLayerStates()
      Dim oLSM As AcadLayerStateManager
      Set oLSM = ThisDrawing.Application. _
               GetInterfaceObject("AutoCAD.AcadLayerStateManager.18")
      oLSM.SetDatabase ThisDrawing.Database

      '如果要导入的图形中不包含已保存的设置中引用的所有线型，则返回一个错误。
      '但会使用默认线型，并继续完成导入。
      On Error Resume Next
      oLSM.Import "c:\mydocuments\ColorLType.las"

      If Err.Number = -2145386359 Then
            ' 错误号表示线型没有定义
            MsgBox ("One or more linetypes specified in the imported " + _
                           "settings is not defined in your drawing")
      End If
```

```
        On Error GoTo 0
End Sub
```

3.7 向图形中添加文字

图形中的文字可以表达重要信息。文字对象可以用于标题栏、标注图形部件、技术要求，或者注释等。

AutoCAD 提供了多种创建文字的方式。对短小简单的实体，可以使用单行文字。对于具有内部格式的较长的实体，可以使用多行文字（MText）。尽管所有键入的文字都使用当前文字式样的默认字体和格式，我们仍可以用好几种方法来定制文字的外观。

3.7.1 使用文字样式

AutoCAD 图形中的所有文字都具有与之关联的样式。我们输入文字时，AutoCAD 使用当前文字样式，该样式设置了文字的字体、大小、角度、方向，及其他文字特性。我们可以对默认样式进行修改，或创建并加载新文字样式。一旦创建了一个文字样式，我们就可以修改其属性，并在不再需要时将其删除。

3.7.1.1 创建和修改文字样式

新建的文字对象从当前文字样式继承高度、宽度系数、倾斜角度及文字生成（text generation）等属性。创建文字样式的步骤，首先新建 TextStyleTableRecord 对象的一个实例，用 Name 属性给新文字样式命名，然后以写方式打开 TextStyleTable 对象，用 Add 方法将新创建文字样式添加到文字样式表中。

样式名可以包含字母、数字和美元符号（$）、下划线（_）、连字符（-）等特殊字符。AutoCAD 会将字符均转换成大写。要是不输入样式名，新样式将没有名字。

我们可以通过修改 TextStyleTableRecord 对象的属性来修改现有样式。如果要使用当前文字样式，引用 Database 对象的 TextStyle 属性即可，该属性拥有当前文字样式的 ObjectId。

我们还可以更新现有文字以反映对样式的修改。

下列属性用来修改 TextStyleTableRecord 对象：

BigFontFileName

指定用于非 ASCII 字符集的特殊形定义文件；

FileName

指定与字体关联的文件；

FlagBits

指定反向文字、颠倒文字，或两者；

Font

指定字样、粗细、倾斜、字符集、间距，及文字样式的公共设置；

IsVertical

指定文字水平或垂直方向；

ObliquingAngle

指定字符的倾斜角度；

TextSize

指定字符高度；

XScale

指定字符放大或缩小的比例；

如果修改了现有样式的字体或方向，使用该样式的所有文字都会更改为新的字体或方向。修改文字高度、宽度系数、倾角等不会更新现有文字，但会在随后创建的文字上体现所作的样式修改。

注： 重新生成图形才会看到对上述属性所做的修改。

3.7.1.2 指定字体

字体定义了构成每个字符集的字符形状。一个字体可以用于多个样式。给文字样式设置字体文件用 FileName 属性。我们可以给文字样式指定 TrueType 字体或 AutoCAD 编译字形(SHX)字体。

设置文字字体

下面的例子使用当前文字样式的 Font 属性获取当前的字体值，然后将字体修改为“PlayBill”。想看看修改效果，可在运行示例代码前往当前图形中添加一些多行文字或单行文字。注意如果你的系统没有 PlayBill 字体，需要替换一个别的字体以确保代码能正确运行。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("UpdateTextFont")>_
Public Sub UpdateTextFont()
    '' 获取当前文档及数据库
    Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

            '' 以写模式打开当前文字样式
            Dim acTextStyleTblRec AsTextStyleTableRecord
            acTextStyleTblRec =acTrans.GetObject(acCurDb.Textstyle, _

          OpenMode.ForWrite)
              '' 获取当前字体设置
            Dim acFont As Autodesk.AutoCAD.GraphicsInterface.FontDescriptor
            acFont = acTextStyleTblRec.Font

            '' 将文字样式中的字体改为”PlayBill”
            Dim acNewFont As Autodesk.AutoCAD.GraphicsInterface.FontDescriptor
            acNewFont = New   _
              Autodesk.AutoCAD.GraphicsInterface.FontDescriptor("PlayBill",_

                                    acFont.Bold, _

                                      acFont.Italic,_
```

```
                                    acFont.CharacterSet, _

                                    acFont.PitchAndFamily)

            acTextStyleTblRec.Font = acNewFont

            acDoc.Editor.Regen()

            '' 保存修改，关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
usingAutodesk.AutoCAD.GraphicsInterface;

[CommandMethod("UpdateTextFont")]
public static void UpdateTextFont()
{
    // 获取当前文档及数据库
    Document acDoc =Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
    {
            // 以写模式打开当前文字样式
            TextStyleTableRecord acTextStyleTblRec;
            acTextStyleTblRec =acTrans.GetObject(acCurDb.Textstyle,

        OpenMode.ForWrite) as TextStyleTableRecord;

            // 获取当前字体设置
            FontDescriptor acFont;
            acFont = acTextStyleTblRec.Font;

            // 将文字样式中的字体改为”PlayBill”
            FontDescriptor acNewFont;
            acNewFont = new FontDescriptor("PlayBill",
```

```
                                    acFont.Bold,

                                    acFont.Italic,

                                    acFont.CharacterSet,

                                    acFont.PitchAndFamily);

            acTextStyleTblRec.Font = acNewFont;

            acDoc.Editor.Regen();

            // 保存修改，关闭事务
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub UpdateTextFont()

        MsgBox ("Look at the text now...")

        Dim typeFace As String
        Dim SavetypeFace As String
        Dim Bold As Boolean
        Dim Italic As Boolean
        Dim charSet As Long
        Dim PitchandFamily As Long

        ' 获取当前文字设置
        ThisDrawing.ActiveTextStyle.GetFont typeFace, _
                                    Bold, Italic, charSet,PitchandFamily

        ' 在 SetFont 方法中，只修改 typeFace 属性，其他局使用默认值
        SavetypeFace = typeFace
        typeFace = "PlayBill"
        ThisDrawing.ActiveTextStyle.SetFont typeFace, _
                                    Bold, Italic, charSet,PitchandFamily
        ThisDrawing.Regen acActiveViewport
End Sub
```

3.7.1.3 使用 TrueType 字体

TrueType 字体在图形中总是填充的，不过打印时系统变量 TEXTFILL 控制字体是否填充。默认情况下 TEXTFILL 变量置为 1，表示打印填充字体。当我们将图形导出为 PostScript®格式并在 PostScript®设备上打印时，打印出的字体就会如设计所愿。

3.7.1.4 使用 Unicode 字体和大字体

AutoCAD 支持 Unicode 字符编码标准。Unicode 字体包含 65535 个字符，可以囊括许多语言的字形。随 AutoCAD 软件安装的所有 AutoCAD SHX 字形字体均支持 Unicode 字体。

有些语言的文本文件含有数量非常多的非 ASCII 字符。为了顺应这些文字，AutoCAD 支持一种称为大字体(Big Font)文件的特殊字形定义类型。我们在设置文字样式时，既可以使用常规字体文件，也可以使用大字体文件。指定常规字体文件使用 FileName 属性，指定大字体文件使用 BigFontFileName 属性。

注：字体文件名不能包含逗号。

当找不到指定的字体文件时，AutoCAD 允许指定一个替换字体。替换字体使用系统变量 FONTALT 或 Application 对象的 SetSystemVariable 成员方法来设置。

改变字体文件

下面的示例代码修改 FileName 属性和 BigFontFileName 属性。你需要将代码中的路径信息替换成你的系统里相应的路径和文件名。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices

<CommandMethod("ChangeFontFiles")>_
Public Sub ChangeFontFiles()
    '' 获取当前文档及数据库
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

            '' 以写模式打开当前文字样式
            Dim acTextStyleTblRec As TextStyleTableRecord
            acTextStyleTblRec =acTrans.GetObject(acCurDb.Textstyle, _

        OpenMode.ForWrite)
```

```
            '' 改用 bothBig 字体文件和常规字体
            acTextStyleTblRec.BigFontFileName ="C:\AutoCAD\Fonts\bigfont.shx"
            acTextStyleTblRec.FileName ="C:\AutoCAD\Fonts\italic.shx"

            '' 保存修改，关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;

[CommandMethod("ChangeFontFiles")]
public static void ChangeFontFiles()
{
    // 获取当前文档及数据库
    Document acDoc =Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
    {
            // 以写模式打开当前文字样式
            TextStyleTableRecord acTextStyleTblRec;
            acTextStyleTblRec =acTrans.GetObject(acCurDb.Textstyle,

        OpenMode.ForWrite) as TextStyleTableRecord;

            // 改用 bothBig 字体文件和常规字体
            acTextStyleTblRec.BigFontFileName ="C:/AutoCAD/Fonts/bigfont.shx";
            acTextStyleTblRec.FileName ="C:/AutoCAD/Fonts/italic.shx";

            // 保存修改，关闭事务
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub ChangeFontFiles()
        ThisDrawing.ActiveTextStyle.BigFontFile = _
                                    "C:/AutoCAD/Fonts/bigfont.shx"
```

```
        ThisDrawing.ActiveTextStyle.fontFile = _
                                    "C:/AutoCAD/Fonts/italic.shx"
End Sub
```

3.7.1.5 设置文字高度

文字高度确定了所使用字体字符的大小，字符大小用图形单位表示。该值通常表示大写字母的大小，但 TrueType 字体除外。

对于 TrueType 字体，文字高度值可能不表示大写字母的高度，而是大写字母的高度加上留给重音符号及用于非英语语言的其他标记的标音区域。留给大写字母区域和重音符号区域的相对大小是由字体设计者在设计字体时确定的，因而会因不同的字体而异。

指定的文字高度除了由大写字母高度和重音区域构成外，TrueType 字体字符还有一个扩展到文本插入线以下的下降区域，这样的字符有 y、j、p、g 及 q 等。

设置文字高度使用文字样式的 TextSize 属性或者文字对象的 Height 属性。该属性只接受正数值。

3.7.1.6 设置文字倾角

文字倾角用来确定文字向前倾斜或向后倾斜。倾斜角度为相对于垂直轴线（90 度）的偏移。设置文字倾角使用文字样式的 ObliquingAngle 属性或者文字对象的 Oblique 属性。倾角单位必须是弧度，角度为正数表示向右倾斜，负数角度会被加上 2π（2*PI）将其转换成相应的正数角度。

创建倾斜文字

本例创建一个单行文字对象，然后将它倾斜 45 度角。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("ObliqueText")>_
```

```
Public Sub ObliqueText()
    '' 获取当前文档及数据库
    Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, _
                                                                    OpenMode.Fo
rRead)

            '' 以写模式打开 Block 表记录 Model 空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec
=acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                            OpenMode.ForWrite)

            '' 创建一个单行文字对象
            Dim acText As DBText = New DBText()
            acText.Position = New Point3d(3, 3, 0)
            acText.Height = 0.5
            acText.TextString = "Hello,World."

            '' 修改文字对象的倾斜角度为 45 度（弧度值 0.707）
            acText.Oblique = 0.707

            acBlkTblRec.AppendEntity(acText)
            acTrans.AddNewlyCreatedDBObject(acText,True)

            '' 保存修改，关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
usingAutodesk.AutoCAD.Geometry;

[CommandMethod("ObliqueText")]
```

```
public static void ObliqueText()
{
    // 获取当前文档及数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec
=acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                        OpenMode.ForWrite) as BlockTableRecord;

            // 创建一个单行文字对象
            DBText acText = new DBText();
            acText.Position = new Point3d(3, 3, 0);
            acText.Height = 0.5;
            acText.TextString = "Hello, World! ";

            // 修改文字对象的倾斜角度为 45 度（弧度值 0.707）
            acText.Oblique = 0.707;

            acBlkTblRec.AppendEntity(acText);
            acTrans.AddNewlyCreatedDBObject(acText,true);

            // 保存修改，关闭事务
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub ObliqueText()
        Dim textObj As AcadText
        Dim textString As String
        Dim insertionPoint(0 To 2) As Double
        Dim height As Double
```

```
        ' 定义文字对象
        textString = "Hello, World."
        insertionPoint(0) = 3
        insertionPoint(1) = 3
        insertionPoint(2) = 0
        height = 0.5

        ' 在 Model 空间创建一个文字对象
        Set textObj = ThisDrawing.ModelSpace. _
                            AddText(textString,insertionPoint, height)

        ' 修改 ObliqueAngle 值
        ' to 45 degrees (.707 radians)
        textObj.ObliqueAngle = 0.707
        textObj.Update
End Sub
```

3.7.1.7 设置文字生成标志

文字生成标志用来表示文字是否反向显示或倒置显示。可以使用文字样式的 FlagBits 属性设置文字样式是否控制文字的反向显示或倒置显示，或者使用文字对象的 IsMirroredInX 属性和 IsMirroredInY 属性来单独对某个文字对象进行控制。

如果要反向显示文字，设置 FlagBits 属性值为 2；如果要倒置显示文字，设置 FlagBits 属性值为 4。FlagBits 属性值为 6 表示同时反向和倒置显示文字。要修改单个文字对象时，将其 IsMirroredInX 属性置为 TRUE 就表示反向显示文字，将其 IsMirroredInY 属性置为 TRUE 就表示倒置显示文字。

反向显示文字

下面的代码先创建一个单行文字对象，然后使用 IsMirroredInX 属性将其设置为反向显示。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("BackwardsText")>_
Public Sub BackwardsText()
    '' 获取当前文档及数据库
    Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
```

```
    Dim acCurDb As Database = acDoc.Database

    '' 启动事务
    Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

            '' 以读模式打开 Block 表
            Dim acBlkTbl As BlockTable
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, _
                                        OpenMode.ForRead)

            '' 以写模式打开 Block 表记录 Model 空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec
=acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                       OpenMode.ForWrite)

            '' 创建一个单行文字对象
            Dim acText As DBText = New DBText()
            acText.Position = New Point3d(3, 3, 0)
            acText.Height = 0.5
            acText.TextString = "Hello,World."

            '' 反向显示文字
            acText.IsMirroredInX = True

            acBlkTblRec.AppendEntity(acText)
            acTrans.AddNewlyCreatedDBObject(acText,True)

            '' 保存修改，关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
usingAutodesk.AutoCAD.Geometry;

[CommandMethod("BackwardsText")]
public static void BackwardsText()
{
    // 获取当前文档及数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                        OpenMode.ForRead) as BlockTable;

            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec
=acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                               OpenMode.ForWrite) as BlockTableRecord;

            // 创建一个单行文字对象
            DBText acText = new DBText();
            acText.Position = new Point3d(3, 3, 0);
            acText.Height = 0.5;
            acText.TextString = "Hello,World.";

            // 反向显示文字
            acText.IsMirroredInX = true;

            acBlkTblRec.AppendEntity(acText);
            acTrans.AddNewlyCreatedDBObject(acText,true);

            // 保存修改，关闭事务
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub BackwardsText()
        Dim textObj As AcadText
        Dim textString As String
        Dim insertionPoint(0 To 2) As Double
        Dim height As Double

        ' 创建一个文字对象
        textString = "Hello, World."
        insertionPoint(0) = 3
        insertionPoint(1) = 3
```

```
        insertionPoint(2) = 0
        height = 0.5
        Set textObj = ThisDrawing.ModelSpace. _
                                        AddText(textString,insertionPoint,
height)

        ' 修改 TextGenerationFlag 值
        textObj.TextGenerationFlag =acTextFlagBackward
        textObj.Update
End Sub
```

3.7.2 使用单行文字（Text 命令）

添加到图形中的文字可以传达许多信息，可以是复杂的技术说明、标题栏信息、标签，甚至图形的部件。对于不需要多种字体和多行文本的较短文字，可以通过创建一个 DBText 对象的实例来创建一个单行文字，用于标签时尤其方便。

3.7.2.1 创建单行文字

使用单行文字时，每一行文字都是一个单独的对象。创建单行文字的方法是，首先创建 DBText 对象的实例，然后将其添加到代表模型空间或图纸空间的块表记录中。创建 DBText 对象的实例时，不用给构造函数传递任何参数。

创建单行文字

下面这个例子在模型空间创建一个单行文字对象，起点坐标为（2,2,0），文字高度为 0.5，文字内容为“Hello, World.”。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("CreateText")>_
Public Sub CreateText()
  '' 获取当前文档及数据库
```

```
  Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

          '' 以读模式打开 Block 表
          Dim acBlkTbl As BlockTable
          acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, _
                                      OpenMode.ForRead)

          ''以写模式打开 Block 表记录 Model 空间
          Dim acBlkTblRec As BlockTableRecord
          acBlkTblRec =acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
 OpenMode.ForWrite)

          '' 创建一个单行文字对象
          Dim acText As DBText = New DBText()
          acText.Position = New Point3d(2, 2, 0)
          acText.Height = 0.5
          acText.TextString = "Hello,World."

          acBlkTblRec.AppendEntity(acText)
          acTrans.AddNewlyCreatedDBObject(acText,True)

          '' 保存修改，关闭事务
          acTrans.Commit()
  End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateText")]
public static void CreateText()
{
  // 获取当前文档及数据库
  Document acDoc =Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
  // 启动事务
  using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
  {
          // 以读模式打开 Block 表
          BlockTable acBlkTbl;
          acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                                                  OpenMode.Fo
rRead) as BlockTable;

          // 以写模式打开 Block 表记录 Model 空间
          BlockTableRecord acBlkTblRec;
          acBlkTblRec =acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                        OpenMode.ForWrite) as BlockTableRecord;

          // 创建一个单行文字对象
          DBText acText = new DBText();
          acText.Position = new Point3d(2, 2, 0);
          acText.Height = 0.5;
          acText.TextString = "Hello,World.";

           //添加到模型空间
          acBlkTblRec.AppendEntity(acText);
          acTrans.AddNewlyCreatedDBObject(acText,true);

          // 保存修改，关闭事务
          acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub CreateText()
      Dim textObj As AcadText
      Dim textString As String
      Dim insertionPoint(0 To 2) As Double
      Dim height As Double

      ' 创建文字对象
      textString = "Hello, World."
      insertionPoint(0) = 2
      insertionPoint(1) = 2
      insertionPoint(2) = 0
      height = 0.5

      Set textObj = ThisDrawing.ModelSpace. _
```

```
                        AddText(textString,insertionPoint, height)
    textObj.Update
End Sub
```

3.7.2.2 格式化单行文字

单行文字是使用当前文字样式创建的。我们可以通过修改与之关联的文字样式来修改单行文字的格式，或者通过直接编辑单行文字对象的属性来修改其格式。不能对单行文字对象里的单个文字和字符进行格式修改。

要修改单行文字对象的文字样式，将其 TextStyleId 属性设置为新文字样式即可。修改文字样式后，要想看到其效果，需重新生成图形或更新一下该文字对象。

除了标准的实体可编辑属性（颜色、图层、线型，等等）外，单行文字还另外拥有一些我们可以修改的属性，如下：

HorizontalMode

表示文字水平对齐；

VerticalMode

表示文字垂直对齐；

Position

表示文字的插入点；

Oblique

表示文字对象的倾角；

Rotation

表示文字的旋转角度（弧度）；

WidthFactor

表示文字的放大系数；

AlignmentPoint

表示文字的对齐点；

IsMirroredInX

表示文字是否反向显示；

IsMirroredInY

表示文字是否倒置显示；

TextString

表示实际显示的文字内容；

每次修改文字的一个属性后，都要重新生成图形或对该文字对象进行更新，这样才能看到所作的修改。

3.7.2.3 对齐单行文字

我们可以对单行文字进行水平方向和垂直方向的调整。单行文字默认情况下是左对齐的。设置水平对齐和垂直对齐选项，使用 DBText 对象的 HorizontalMode 属性和 VerticalMode 属性。

通常创建完文字对象时，文字对象的位置和对齐点会依据排版和文字样式作相应调整。不过内存里的文字对象的对齐不会自动更新。可以调用 AdjustAlignment()方法基于当前属性值对文字对象的对齐特性进行更新。

重新调整文字

下面的例子创建一个单行文字对象（DBText）和一个点对象（DBPoint）。将点对象设置成文字的对齐点，并将点的形状改成红色十字，以便观察。修改文字的排列对齐，并查看修改效果。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("TextAlignment")>_
Public Sub TextAlignment()
  '' 获取当前文档及数据库
  Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()

          '' 以读模式打开 Block 表
          Dim acBlkTbl As BlockTable
          acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, _
                                      OpenMode.ForRead)

          '' 以写模式打开 Block 表记录 Model 空间
          Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec =acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                       OpenMode.ForWrite)

          Dim textString(0 To 2) As String
          textString(0) = "Left"
          textString(1) = "Center"
          textString(2) = "Right"

          Dim textAlign(0 To 2) As Integer
          textAlign(0) = TextHorizontalMode.TextLeft
          textAlign(1) =TextHorizontalMode.TextCenter
          textAlign(2) =TextHorizontalMode.TextRight

          Dim acPtIns As Point3d = New Point3d(3,3, 0)
          Dim acPtAlign As Point3d = New Point3d(3,3, 0)

          Dim nCnt As Integer = 0
          For Each strVal As String In textString
                  '' 创建一个单行文字对象
                  Dim acText As DBText = New DBText()
                  acText.Position = acPtIns
```

```
                acText.Height = 0.5
                acText.TextString = strVal

                '' 设置文字对齐模式
                acText.HorizontalMode =textAlign(nCnt)

                If acText.HorizontalMode <>TextHorizontalMode.TextLeft Then
                        acText.AlignmentPoint = acPtAlign
                End If

                acBlkTblRec.AppendEntity(acText)
                acTrans.AddNewlyCreatedDBObject(acText, True)

                '' 在文字对齐点创建一个 DBPoint 对象（点）
                Dim acPoint As DBPoint = NewDBPoint(acPtAlign)
                acPoint.ColorIndex = 1

                acBlkTblRec.AppendEntity(acPoint)
                acTrans.AddNewlyCreatedDBObject(acPoint, True)

                '' 调整插入点和对齐点
                acPtIns = New Point3d(acPtIns.X,acPtIns.Y + 3, 0)
                acPtAlign = acPtIns

                nCnt = nCnt + 1
            Next
            '' 设置点的样式为十字
            Application.SetSystemVariable("PDMODE", 2)
            '' 保存修改，关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
usingAutodesk.AutoCAD.Geometry;

[CommandMethod("TextAlignment")]
public static void TextAlignment()
{
  // 获取当前文档及数据库
```

```
  Document acDoc =Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
  {
          // 以读模式打开 Block 表
          BlockTable acBlkTbl;
          acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                  OpenMode.ForRead) as BlockTable;

          // 以写模式打开 Block 表记录 Model 空间
          BlockTableRecord acBlkTblRec;
          acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                        OpenMode.ForWrite) as BlockTableRecord;

          string[] textString = new string[3];
          textString[0] = "Left";
          textString[1] = "Center";
          textString[2] = "Right";

          int[] textAlign = new int[3];
          textAlign[0] =(int)TextHorizontalMode.TextLeft;
          textAlign[1] =(int)TextHorizontalMode.TextCenter;
          textAlign[2] =(int)TextHorizontalMode.TextRight;

          Point3d acPtIns = new Point3d(3, 3, 0);
          Point3d acPtAlign = new Point3d(3, 3, 0);

          int nCnt = 0;

          foreach (string strVal in textString)
          {
                  // 创建一个单行文字对象
                  DBText acText = new DBText();
                  acText.Position = acPtIns;
                  acText.Height = 0.5;
                  acText.TextString = strVal;

                  // 设置文字的排列模式
                  acText.HorizontalMode =(TextHorizontalMode)textAlign[nCnt];

                  if (acText.HorizontalMode !=TextHorizontalMode.TextLeft)
```

```
                    {
                            acText.AlignmentPoint =acPtAlign;
                    }

                    acBlkTblRec.AppendEntity(acText);
                    acTrans.AddNewlyCreatedDBObject(acText, true);

                    // 在文字的对齐点上创建一个点对象
                    DBPoint acPoint = new DBPoint(acPtAlign);
                    acPoint.ColorIndex = 1; // 红色

                    acBlkTblRec.AppendEntity(acPoint);
                    acTrans.AddNewlyCreatedDBObject(acPoint, true);

                    // 调整插入点和对齐点
                    acPtIns = new Point3d(acPtIns.X,acPtIns.Y + 3, 0);
                    acPtAlign = acPtIns;

                    nCnt = nCnt + 1;
            }

            // 设置点的样式为十字
            Application.SetSystemVariable("PDMODE", 2);

            // 保存修改，关闭事务
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub TextAlignment()
      ’ 设置点的样式为十字
      ThisDrawing.SetVariable "PDMODE",2

      Dim textObj As AcadText
      Dim pointObj As AcadPoint

      ’ 定义文字串级文字对象的插入点
      Dim textString(0 To 2) As String
      textString(0) = "Left"
      textString(1) = "Center"
      textString(2) = "Right"

      Dim textAlign(0 To 2) As Integer
```

```
        textAlign(0) = acAlignmentLeft
        textAlign(1) = acAlignmentCenter
        textAlign(2) = acAlignmentRight

        Dim insertionPoint(0 To 2) As Double
        insertionPoint(0) = 3: insertionPoint(1) =0: insertionPoint(2) = 0

        Dim alignmentPoint(0 To 2) As Double
        alignmentPoint(0) = 3: alignmentPoint(1) =0: alignmentPoint(2) = 0

        Dim nCnt As Integer
        For Each strVal In textString
                ’在 Model 空间创建一个文字对象
                Set textObj = ThisDrawing.ModelSpace. _
                                        AddText(strVal,insertionPoint, 0.5)

                ’设置文字对齐模式
                textObj.Alignment = textAlign(nCnt)

                On Error Resume Next
                textObj.TextAlignmentPoint =alignmentPoint

                ’在文字的对齐点上创建一个点对象
                Set pointObj = ThisDrawing.ModelSpace.AddPoint(alignmentPoint)
                pointObj.color = acRed

                ’调整插入点和对齐点
                insertionPoint(1) = insertionPoint(1) +3
                alignmentPoint(1) = insertionPoint(1)

                nCnt = nCnt + 1
        Next
End Sub
```

3.7.2.4 修改单行文字

就像操作其他对象一样，我们也可以对文字对象进行移动、旋转、删除和复制等修改操作。还可以对文字进行镜像操作。镜像文字时，如果不想文字反向，可以将系统变量 MIRRTEXT 设置为 0。移动、旋转和复制对象使用 TransformBy()方法和 Clone()方法。

关于 TransformBy()方法和 Clone()方法的详细内容，见(§3.4 *编辑命名对象和二维对象*)。

3.7.3 使用多行文字（MText 命令）

对于字数较多的复杂的文字实体，就要创建多行文字对象（MText）。多行文字在指定宽度范围下可以垂直方向无限延展。我们可以对 MText 中的单个字或字符单独进行格式设置。

3.7.3.1 创建多行文字

创建多行文字的步骤：首先创建一个 Mtext 类的对象实例，然后将它添加到代表模型空间或图纸空间的块表记录中。MText 对象的构造函数没有参数。创建了 MText 对象实例后，我们可以使用对象属性给这个实例赋予一个文字串、设置插入点、宽度及其他属性值。我们还可以修改文字对象的高度、排版、旋转角度、文字样式等属性，或者设置选定字符的格式等。

创建一个多行文字对象

下面这个例子在模型空间创建一个 MText 对象，起点坐标为（2,2,0）。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry

<CommandMethod("CreateMText")>_
Public Sub CreateMText()
  '' 获取当前文档及数据库
  Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

          '' 以读模式打开 Block 表
          Dim acBlkTbl As BlockTable
```

```
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, _
                                        OpenMode.ForRead)

            '' 以写模式打开 Block 表记录 Model 空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec =acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                    OpenMode.ForWrite)

            '' 创建一个多行文字对象
            Dim acMText As MText = New MText()
            acMText.Location = New Point3d(2, 2, 0)
            acMText.Width = 4
            acMText.Contents = "This is a text string for the MText object."

            acBlkTblRec.AppendEntity(acMText)
            acTrans.AddNewlyCreatedDBObject(acMText,True)

            '' 保存修改，关闭事务
            acTrans.Commit()
    End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateMText")]
public static void CreateMText()
{
    // 获取当前文档及数据库
    Document acDoc =Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
    {
            // 以读模式打开 Block 表
            BlockTable acBlkTbl;
            acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                        OpenMode.ForRead) as BlockTable;
```

```
            // 以写模式打开 Block 表记录 Model 空间
            BlockTableRecord acBlkTblRec;
            acBlkTblRec =acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                            OpenMode.ForWrite) as BlockTableRecord;

            // 创建多行文字对象
            MText acMText = new MText();
            acMText.Location = new Point3d(2, 2, 0);
            //属性赋值
            acMText.Width = 4;
            acMText.Contents = "This is a text string for the MText object.";

            acBlkTblRec.AppendEntity(acMText);
            acTrans.AddNewlyCreatedDBObject(acMText,true);

            // 保存修改，关闭事务
            acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub Ch4_CreateMText()
      Dim mtextObj As AcadMText
      Dim insertPoint(0 To 2) As Double
      Dim width As Double
      Dim textString As String
      insertPoint(0) = 2
      insertPoint(1) = 2
      insertPoint(2) = 0
      width = 4
      textString = "This is a text string for the mtext object."
      ' 在模型空间创建文字对象
      Set mtextObj = ThisDrawing.ModelSpace. _
                              AddMText(insertPoint,width, textString)
      ZoomAll
End Sub
```

3.7.3.2 格式化多行文字

新创建的多行文字会自动呈现为当前文字样式的特性。默认文字样式为 STANDARD。我们可以通过格式化个别字符及设置多行文字属性来覆盖默认文字样式。我们还可以使用本节介绍的方法来格式化字符或指定特殊字符。

像文字样式、排版、宽度以及旋转等这些方向性选项，会影响多行文字框内的所有文字，而不是只影响指定的单词或字符。修改多行文字对象的排版使用 Attachment 属性，控制多行文字对象的旋转角度使用 Rotation 属性。

TextStyleId 属性用来设置多行文字对象的字体和格式化特性。创建多行文字时，我们可以从现有样式列表中选择我们想用的样式。如果多行文字的的部分字符已经格式化，这时要修改多行文字的样式的话，新样式将应用于整个多行文字对象，已做的字符格式化将不会保留。例如，将 TrueType 样式修改成使用 SHX 字体的样式或另外一种 TrueType 字体样式时，其结果是新样式应用于整个多行文字对象，任何字符格式化都会被丢弃。

像下划线、上下标文字、字体这些格式化选项可以应用于文字段落内的单个单词或字符。我们还可以修改单个单词或字符的颜色、字体、文字高度等，修改字符与字符之间的间距，或者扩大字符的宽度等。

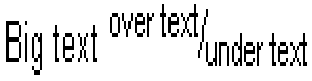
使用花括号（{}）可以只对括号内的文字进行格式修改。花括号的嵌套深度最多为 8 层。

我们还可以在行内或段落内键入 ASCII 值控制码来表示格式或特殊字符，像公差符号、尺寸标注符号等。

下面的控制字符可以用来创建图示那样的文字。

```
{{\H1.5x; Big text} \A2; overtext\A1;/\A0; under text}
```

效果图：



用控制字符格式化文字

下面的示例代码创建并格式化一个多行文字对象。

VB.NET

```
ImportsAutodesk.AutoCAD.Runtime
```

```
ImportsAutodesk.AutoCAD.ApplicationServices
ImportsAutodesk.AutoCAD.DatabaseServices
ImportsAutodesk.AutoCAD.Geometry
<CommandMethod("FormatMText")>_
Public Sub FormatMText()
  '' 获取当前文档及数据库
  Dim acDoc As Document =Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
  '' 启动事务
  Using acTrans As Transaction =acCurDb.TransactionManager.StartTransaction()
          '' 以读模式打开 Block 表
          Dim acBlkTbl As BlockTable
          acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId, _
                                      OpenMode.ForRead)
          '' 以写模式打开 Block 表记录 Model 空间
          Dim acBlkTblRec As BlockTableRecord
          acBlkTblRec =acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace),
_
                                 OpenMode.ForWrite)
          '' 创建多行文字对象
          Dim acMText As MText = New MText()
          acMText.Location = New Point3d(2, 2, 0)
          acMText.Width = 4.5
          acMText.Contents = "{{\H1.5x; Bigtext}\A2; over text\A1;/\A0;under
text}"

          acBlkTblRec.AppendEntity(acMText)
          acTrans.AddNewlyCreatedDBObject(acMText,True)
          '' 保存修改，关闭事务
          acTrans.Commit()
  End Using
End Sub
```

C#

```
usingAutodesk.AutoCAD.Runtime;
usingAutodesk.AutoCAD.ApplicationServices;
usingAutodesk.AutoCAD.DatabaseServices;
usingAutodesk.AutoCAD.Geometry;

[CommandMethod("FormatMText")]
public static void FormatMText()
{
  // 获取当前文档及数据库
```

```
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans =acCurDb.TransactionManager.StartTransaction())
  {
          // 以读模式打开 Block 表
          BlockTable acBlkTbl;
          acBlkTbl =acTrans.GetObject(acCurDb.BlockTableId,
                                      OpenMode.ForRead) as BlockTable;

          // 以写模式打开 Block 表记录 Model 空间
          BlockTableRecord acBlkTblRec;
          acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                OpenMode.ForWrite) as BlockTableRecord;

          // 创建多行文字对象
          MText acMText = new MText();
          acMText.Location = new Point3d(2, 2, 0);
          acMText.Width = 4.5;
          acMText.Contents = "{{\\H1.5x; Bigtext}\\A2; over text\\A1;/\\A0;under
text}";

          acBlkTblRec.AppendEntity(acMText);
          acTrans.AddNewlyCreatedDBObject(acMText,true);

          // 保存修改，关闭事务
          acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub FormatMText()
      Dim mtextObj As AcadMText
      Dim insertPoint(0 To 2) As Double
      Dim width As Double
      Dim textString As String

      insertPoint(0) = 2
      insertPoint(1) = 2
      insertPoint(2) = 0
      width = 4.5
```

```
        ' 定义控制字符的 ASCII 字符
        Dim OB As Long   ' 左花括号 {
        Dim CB As Long   ' 右花括号 }
        Dim BS As Long   ' 反斜杠  \
        Dim FS As Long   ' 斜杠 /
        Dim SC As Long   ' 分号 ;
        OB = Asc("{")
        CB = Asc("}")
        BS = Asc("\")
        FS = Asc("/")
        SC = Asc(";")

        ' 定义控制字符串:
        ' {{\H1.5x; Big text}\A2; over text\A1;/\A0;undertext}

        textString = Chr(OB) + Chr(OB) + Chr(BS) +"H1.5x" _
        + Chr(SC) + "Big text" + Chr(CB)+ Chr(BS) + "A2" _
        + Chr(SC) + "over text" + Chr(BS)+ "A1" + Chr(SC) _
        + Chr(FS) + Chr(BS) + "A0" +Chr(SC) + "under text" _
        + Chr(CB)

        ' 在模型空间创建文字对象
        Set mtextObj = ThisDrawing.ModelSpace. _
                              AddMText(insertPoint,width, textString)
        ZoomAll
End Sub
```

3.7.4 使用 Unicode 字符、控制码、特殊字符

我们可以在文本串中使用 Unicode 字符、控制码、及特殊字符来表示各种符号。（所有非文本字符都必须键入其相应的 ASCII 码。）

我们可以通过键入下列 Unicode 字符串来创建特殊符号：

Unicode 字符描述	
Unicode 字符	**描述**
\U+00B0	（° ）度符号

\U+00B1	（±）公差符号
\U+2205	（ϕ）直径符号

除了使用 Unicode 字符来产生特殊符号外，我们还可以通过在文本串中包含控制信息的方式来指定特殊符号。每个控制指令前有两个百分号（%%）。例如，下列控制码使用标准 AutoCAD 文字和 PostScript 字体绘制数字字符 *nnn*。

%%nnn

下面这些控制码只使用标准 AutoCAD 文字字体：

控制码描述	
控制码	**描述**
%%o	上划线模式开/关
%%u	下划线模式开/关
%%d	绘制（° ）度符号
%%p	绘制（±）公差符号
%%c	绘制（ϕ）直径符号
%%%	绘制单个百分号%

3.7.5 替换字体

当 AutoCAD 找不到图形中指定的字体时，我们可以为其选择替换字体，或者选择默认字体。

图形中文字的字体是由所应用的文字样式确定的，对多行文字，则是由文字各部分的不同字体格式确定的。

我们可以使用字体映射表，以确保绘图只使用特定的字体，或者将使用的字体转换成其他字体。我们可以使用这些字体映射表来执行团队字体标准，以方便离线打印。AutoCAD 含有默

认字体映射表，我们可以使用任何 ASCII 文本编辑器编辑这个表文件，我们还可以通过使用 FONTMAP 系统变量来指定另外一个字体映射表文件。

关于字体映射表及替换字体的更多内容，见《AutoCAD 用户指南》中的“替换字体”。

指定默认替换字体

如果图形中指定的字体在当前系统中不存在，AutoCAD 会自动替换成预先选定的替换字体。默认情况下，AutoCAD 使用 *simplex.shx* 文件作为替换字体。不过，如果必要，我们可以另外指定一个字体。设置替换字体文件使用 FONTALT 系统变量。

如果所用的文字样式使用了 Big FONT 字体，我们也可以用 FONTALT 系统变量将其映射为别的字体。字体映射必须使用成对的字体文件：*txt.shx*， *bigfont.shx*。

打开图形时，如果 AutoCAD 找不到某个字体文件，会使用字体替换规则的默认设置。

3.7.6 拼写检查

在拼写检查过程中，AutoCAD 拿图形中的单词与当前主词典中的单词相匹配。我们添加的任何单词都会储存在自定义词典里，并作为当前拼写检查的主词典。例如，我们可以往自定义词典里添加适当的名字，这样 AutoCAD 就不再将它当作有拼写错误的单词来识别。

其他语言的拼写检查，可以换不同的主词典进行。

AutoCAD .NET API 没有提供有关拼写检查的方法，不过，我们可以使用系统变量 DCTMAIN 来指定主词典，使用系统变量 DCTCUST 来指定自定义词典。

第 4 章 标注与公差

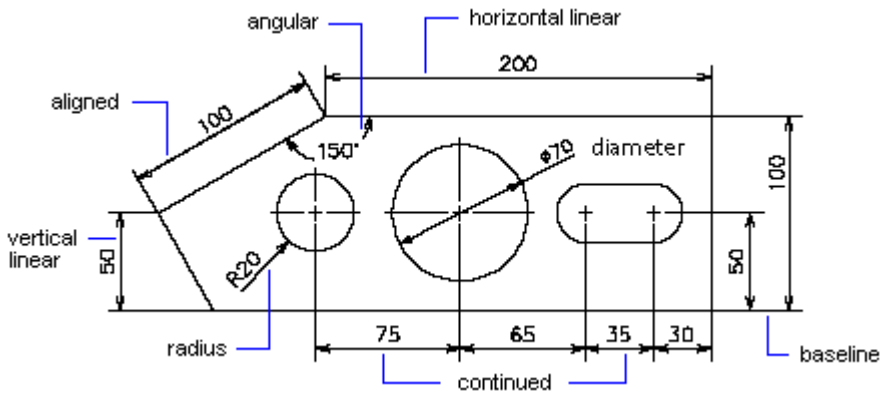
尺寸标注就是给图形添加几何测量值。公差表明一个几何尺寸的变化范围。通过 AutoCAD 提供的.NET API 编程接口，我们可以用标注样式和修改属性设置来管理尺寸标注。

本章主要内容：

- 尺寸标注的概念
- 创建尺寸标注
- 编辑标注
- 使用标注样式
- 模型空间和图纸空间的尺寸标注
- 创建引线和注释
- 使用形位公差

4.1 尺寸标注的概念

尺寸表示的是对象的几何测量、对象间距离或角度，或一个功能属性的 x、y 坐标等。AutoCAD 提供了三种基本尺寸标注类型：线性标注 linear、径向标注 radial 和角度标注 angular，线性标注又分对齐标注 aligned、旋转标注 rotated 和坐标标注 ordinate 三种。

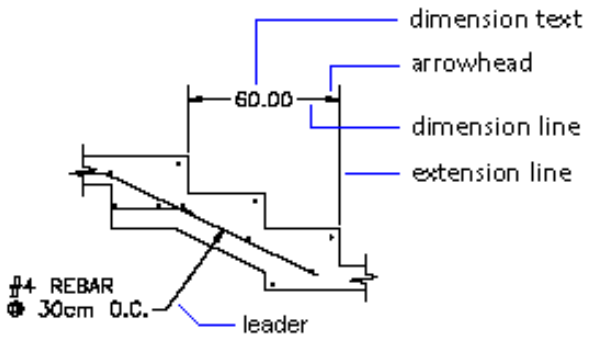


我们可以为直线、多线、圆弧、圆以及多段线线段等对象创建尺寸标注，或者创建单独的尺寸标注对象。

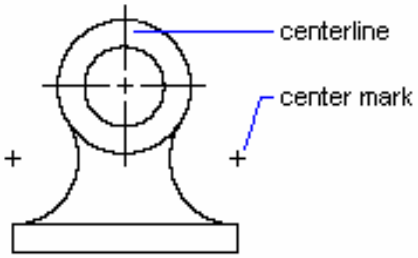
AutoCAD 在当前层绘制尺寸标注，每个尺寸标注均有与其相关联的标注样式，要么是默认样式，要么是你定义的样式。标注样式控制每个尺寸标注的颜色、文字样式、箭头及各元素的比例等特征。尺寸标注不支持对象厚度（thickness）。 样式族允许对一个基本标注样式进行细微的修改，以便用于不同的标注类型。对于一个特定的尺寸标注，允许对其标注样式进行修改。

4.1.1 尺寸的组成部分

尺寸标注可以应用于直线、文字、实体填充和块等许多不同的对象。虽然看上去每种标注类型彼此都不一样， 但它们都有常见的部件。



- **尺寸线 Dimension line.** 一条表示尺寸标注方向和内容的直线。对于角度尺寸，尺寸线是一个圆弧。
- **尺寸界线 Extension line.** 从标注点延伸到尺寸线的一条直线，又被称为投影线或界线。
- **箭头 Arrowhead.** 表示尺寸线端点的符号，又称终止符号或终止。
- **标注文字 Dimension text.** 表示所测量距离或角的实际测量值的文字串，标注文字还包含前缀、后缀及公差等。
- **引线 Leader.** 从注释文字到所参考功能点引出的一条实线。



- **圆心标记 Center mark.** 表示圆或圆弧的中心的小十字符号。
- **中心线 Centerline.** 标记圆或圆弧的中心的一组虚线

4.1.2 定义尺寸标注系统变量

尺寸标注系统变量控制着尺寸的外观，这些系统变量包括 DIMAUNIT、 DIMUPT、 DIMTOFL、DIMFIT、 DIMTIH、 DIMTOH、 DIMJUST 和 DIMTAD 等。我们可以用 Application 对象的 SetSystemVariable 方法来设置这些变量。例如，下面的代码行将系统变量 DIMAUNIT （控制角度的单位）设置为弧度（3）

VB.NET

```
Application.SetSystemVariable("DIMAUNIT", 3)
```

C#

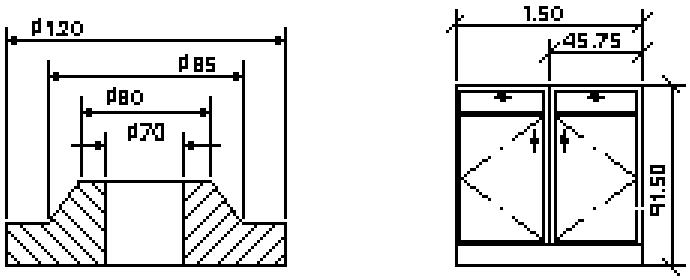
```
Application.SetSystemVariable("DIMAUNIT", 3);
```

VBA/ActiveX 代码参考

```
ThisDrawing.SetVariable "DIMAUNIT", 3
```

4.1.3 设置尺寸的文字样式

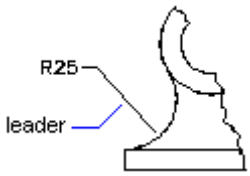
尺寸标注文字是指与尺寸相关的任何文字，包括尺寸、公差（位置的和形状的）、前缀、后缀，以及单行或段落形式的文字说明。你可以用 AutoCAD 计算出来的默认测量值作为标注文字，或提供你自己的文字，或完全禁止文字。你可以用标注文字来添加信息，比如特殊制造流程或装配指导等。标注文字使用由系统变量 DIMTXSTY 设定的文字样式。



4.1.4 了解引线

默认引线是一条带有一个指向图形特征的箭头的直线。通常，引线的功能是将注释与图形特征连接起来，在这种情况下，是指段落文本、块或特征控制框。这样的引线不同于 AutoCAD

为半径、直径及线性尺寸自动创建的简单引线，AutoCAD 自动创建的简单引线的文字不是正好在尺寸界线之间。



引线对象与注释是关联的，当对注释进行编辑时，引线会作相应的变化。我们可以将图形别处的注释复制过来附着给一个引线，或新创建一个注释。我们还可以创建没有注释的引线。

4.1.5 了解关联尺寸

当修改所关联的几何对象时，关联尺寸会自动调整位置、方向和尺寸值。DIMASSOC 系统变量用来控制关联尺寸标注行为。将设置为 2 就是打开关联尺寸标注。

4.2 创建尺寸标注

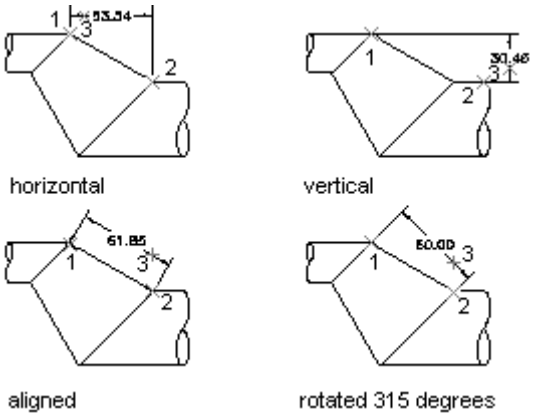
我们可以创建线性、径向、角度和坐标等尺寸标注。

创建尺寸标注时，会使用当前标注样式。标注创建完成后，你可以修改尺寸界线原点、尺寸文字位置、文字内容及其相对于尺寸线的角度等，还可以更改所使用的标注样式。

4.2.1 创建线性标注

线性标注可以是对齐的或旋转（或叫倾斜标注）的。对齐标注的尺寸线平行于尺寸界线原点所在的线，旋转标注的尺寸界线相对于尺寸界线原点所在的线有一个倾斜角度。

我们通过创建AlignedDimension对象和RotatedDimension对象的实例来创建线性标注。创建线性标注的实例后，可以修改其文字、文字倾斜角度、或者尺寸线的角度。下面的示例表明了线性标注的类型和尺寸界线原点的位置是如何影响尺寸线和文字的角度的。



在创建一个 AlignedDimension 对象的实例时，可以通过选项来设定尺寸界线原点、尺寸线位置、标注文字以及应用的标注样式等属性。如果没有给 AlignedDimension 对象的构造函数传递任何参数，对象就会使用默认属性值。

RotatedDimension 对象的构造函数提供了和 AlignedDimension 对象的构造函数一样的选项，只有一个例外，RotatedDimension 对象的构造函数额外还有一个参数，用来指定尺寸线旋转的角度。

折弯尺寸线 Dimension joglines

线性标注上的折弯尺寸线不是通过属性设置添加的，而是通过扩展数据（Xdata）添加的。负责折弯尺寸线标注任务的应用程序名为 ACAD_DSTYLE_DIMJAG_POSITION。下面的示例代码演示了如何将 Xdata 结构添加到线性尺寸标注对象上。

VB.NET

```
'' 以读模式打开已注册应用程序表
Dim acRegAppTbl As RegAppTable
acRegAppTbl = <transaction>.GetObject(<current_database>.RegAppTableId, _
                                      OpenMode.ForRead)

'' 检查看看应用"ACAD_DSTYLE_DIMJAG_POSITION"是否已经注册，
'' 如果没注册的话就将它添加进去
If acRegAppTbl.Has("ACAD_DSTYLE_DIMJAG_POSITION") = False Then
    Dim acRegAppTblRec As RegAppTableRecord = New RegAppTableRecord()

    acRegAppTblRec.Name = "ACAD_DSTYLE_DIMJAG_POSITION"

    acRegAppTbl.UpgradeOpen()

    acRegAppTbl.Add(acRegAppTblRec)
```

```
    <transaction>.AddNewlyCreatedDBObject(acRegAppTblRec, True)
End If

'' 创建一个结果缓冲区来定义 Xdata
Dim acResBuf As ResultBuffer = New ResultBuffer()
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataRegAppName, _
                                    "ACAD_DSTYLE_DIMJAG_POSITION"))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataInteger16, 387))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataInteger16, 3))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataInteger16, 389))
acResBuf.Add(New TypedValue(DxfCode.ExtendedDataXCoordinate, _
                                    New Point3d(-1.26985, 3.91514, 0)))

'' 将 Xdata 添加到尺寸标注对象
<dimension_object>.XData = acResBuf
```

C#

```
// 以读模式打开已注册应用程序表
RegAppTable acRegAppTbl;
acRegAppTbl = <transaction>.GetObject(<current_database>.RegAppTableId,
                                      OpenMode.ForRead) as RegAppTable;

// 检查看看应用"ACAD_DSTYLE_DIMJAG_POSITION"是否已经注册，
// 如果没注册的话就将它添加进去
if (acRegAppTbl.Has("ACAD_DSTYLE_DIMJAG_POSITION") == false)
{
    RegAppTableRecord acRegAppTblRec = new RegAppTableRecord();

    acRegAppTblRec.Name = "ACAD_DSTYLE_DIMJAG_POSITION";

    acRegAppTbl.UpgradeOpen();

    acRegAppTbl.Add(acRegAppTblRec);
    <transaction>.AddNewlyCreatedDBObject(acRegAppTblRec, true);
}

// 创建一个结果缓冲区来定义 Xdata
ResultBuffer acResBuf = new ResultBuffer();
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataRegAppName,
                                        "ACAD_DSTYLE_DIMJAG_POSITION"));
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataInteger16, 387));
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataInteger16, 3));
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataInteger16, 389));
acResBuf.Add(new TypedValue((int)DxfCode.ExtendedDataXCoordinate,
```

```
                                        new Point3d(-1.26985, 3.91514, 0)));

//将 Xdata 添加到尺寸标注对象
<dimension_object>.XData = acResBuf;
```

VBA/ActiveX 代码参考

```
Dim DataType(0 To 4) As Integer
Dim Data(0 To 4) As Variant
Dim jogPoint(0 To 2) As Double

DataType(0) = 1001: Data(0) = "ACAD_DSTYLE_DIMJAG_POSITION"
DataType(1) = 1070: Data(1) = 387
DataType(2) = 1070: Data(2) = 3
DataType(3) = 1070: Data(3) = 389

jogPoint(0) = -1.26985: jogPoint(1) = 3.91514: jogPoint(2) = 0#
DataType(4) = 1010: Data(4) = jogPoint

' 将 Xdata 添加到尺寸标注对象
<dimension_object>.SetXData DataType, Data
```

创建旋转线性标注 rotated linear dimension

本例在模型空间创建一个旋转尺寸标注。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateRotatedDimension")> _
Public Sub CreateRotatedDimension()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
```

```
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)

        '' 以写模式打开块表记录 ModelSpace（模型空间）
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' 创建旋转尺寸标注
        Dim acRotDim As RotatedDimension = New RotatedDimension()
        acRotDim.XLine1Point = New Point3d(0, 0, 0)
        acRotDim.XLine2Point = New Point3d(6, 3, 0)
        acRotDim.Rotation = 0.707
        acRotDim.DimLinePoint = New Point3d(0, 5, 0)
        acRotDim.DimensionStyle = acCurDb.Dimstyle

        '' 将新对象添加到块表记录 ModelSpace 及事务
        acBlkTblRec.AppendEntity(acRotDim)
        acTrans.AddNewlyCreatedDBObject(acRotDim, True)

        '' 提交修改，关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateRotatedDimension")]
public static void CreateRotatedDimension()
{
    // 获取当前数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 以读模式打开块表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
```

```
                                    OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录 ModelSpace（模型空间）
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建旋转尺寸标注
      RotatedDimension acRotDim = new RotatedDimension();
      acRotDim.XLine1Point = new Point3d(0, 0, 0);
      acRotDim.XLine2Point = new Point3d(6, 3, 0);
      acRotDim.Rotation = 0.707;
      acRotDim.DimLinePoint = new Point3d(0, 5, 0);
      acRotDim.DimensionStyle = acCurDb.Dimstyle;

      // 将新对象添加到块表记录 ModelSpace 及事务
      acBlkTblRec.AppendEntity(acRotDim);
      acTrans.AddNewlyCreatedDBObject(acRotDim, true);

      // 提交修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

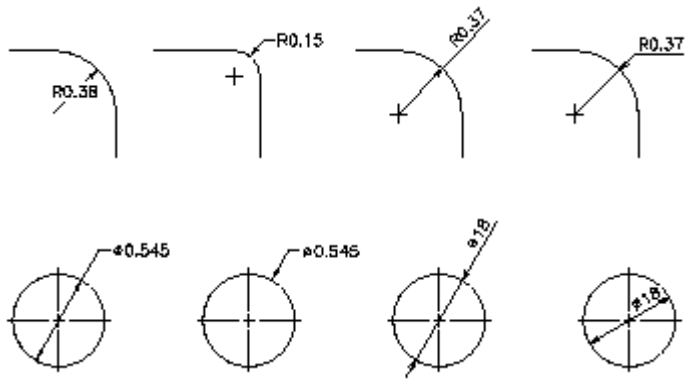
```
Sub CreateRotatedDimension()
    Dim dimObj As AcadDimRotated
    Dim rotationAngle As Double
    Dim startExtPoint(0 To 2) As Double
    Dim endExtPoint(0 To 2) As Double
    Dim dimLinePoint(0 To 2) As Double
    ' 定义一个尺寸标注
    rotationAngle = 0.707
    startExtPoint(0) = 0: startExtPoint(1) = 0: startExtPoint(2) = 0
    endExtPoint(0) = 6: endExtPoint(1) = 3: endExtPoint(2) = 0
    dimLinePoint(0) = 0: dimLinePoint(1) = 5: dimLinePoint(2) = 0

    ' 在模型空间创建旋转标注
    Set dimObj = ThisDrawing.ModelSpace. _
                                AddDimRotated(startExtPoint, endExtPoint, _
                                              dimLinePoint, rotationAngle)
    ZoomAll
End Sub
```

4.2.2 建径向标注

径向标志用来标注圆弧和圆的半径和直径。半径标注通过创建 RadialDimension 对象的实例来实现，直径标注通过创建 DiametricDimension 对象的实例来实现。

依据圆或圆弧的大小、标注文字位置、以及 DIMUPT、DIMTOFL、 DIMFIT、 DIMTIH、 DIMTOH、DIMJUST、DIMTAD 等标注相关系统变量取值等不同情况，我们可以创建出各种不同类型的半径标注。（系统变量可以通过 GetSystemVariable()方法和 SetSystemVariable()方法来设置和访问。）

对于水平标注文字，如果尺寸线与水平线的夹角大于 15 度，并且是标注在圆或圆弧之外，AutoCAD 会将尺寸线绘制成一个勾线，也称为着陆或“狗腿儿”。勾线紧挨着标注文字或位于标注文字下边，如下列插图所示：



创建 RadialDimension 对象实例时，得指定圆心和弦点、引线长度、标注文字及所用的标注样式。创建 DiametricDimension 对象 与创建 RadialDimension 对象类似，不同的是需指定圆上两个点（直径的近端和远端两个点），而不是圆心和弧上一个点。LeaderLength 属性指定了从 ChordPoint（弦点，尺寸引线在圆弧上的起/止点）到注释文字或引线结束端（不需要勾线时）的引线长度。

创建径向标注 Create a radial dimension

本例在模型空间创建一个径向标注。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateRadialDimension")> _
Public Sub CreateRadialDimension()
  '' 获取当前数据库
```

```
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 ModelSpace
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建半径标注
      Dim acRadDim As RadialDimension = New RadialDimension()
      acRadDim.Center = New Point3d(0, 0, 0)
      acRadDim.ChordPoint = New Point3d(5, 5, 0)
      acRadDim.LeaderLength = 5
      acRadDim.DimensionStyle = acCurDb.Dimstyle

      '' 添加新对象到模型空间和事务
      acBlkTblRec.AppendEntity(acRadDim)
      acTrans.AddNewlyCreatedDBObject(acRadDim, True)

      ''提交修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateRadialDimension")]
public static void CreateRadialDimension()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
  // Start a transaction 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // Open the Block table for read 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // Open the Block table record Model space for write
      // 以写模式打开块表记录 ModelSpace
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // Create the radial dimension 创建半径标注
      RadialDimension acRadDim = new RadialDimension();
      acRadDim.Center = new Point3d(0, 0, 0);
      acRadDim.ChordPoint = new Point3d(5, 5, 0);
      acRadDim.LeaderLength = 5;
      acRadDim.DimensionStyle = acCurDb.Dimstyle;

      // 添加新对象到模型空间和事务
      acBlkTblRec.AppendEntity(acRadDim);
      acTrans.AddNewlyCreatedDBObject(acRadDim, true);

      // 提交修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub CreateRadialDimension()
    Dim dimObj As AcadDimRadial
    Dim center(0 To 2) As Double
    Dim chordPoint(0 To 2) As Double
    Dim leaderLen As Integer

    ' 定义标注
    center(0) = 0
    center(1) = 0
    center(2) = 0
    chordPoint(0) = 5
    chordPoint(1) = 5
```

```
    chordPoint(2) = 0
    leaderLen = 5

    ' 在模型空间创建径向标注
    Set dimObj = ThisDrawing.ModelSpace. _
                                AddDimRadial(center, chordPoint, leaderLen)
    ZoomAll
End Sub
```

4.2.3 创建角度标注

角度标注用来测量和标注两条直线间或三个点间的夹角。比如，我们可以用角度标注来测量一个圆的两条半径线的夹角。角度标注的尺寸线是一条弧线。角度标注通过创建 LineAngularDimension2 对象和 Point3AngularDimension 对象的实例来建立。

- **LineAngularDimension2.** 表示由两条线定义的角度标注；
- **Point3AngularDimension.** 表示由三个点定义的角度标注；

创建 LineAngularDimension2 对象和 Point3AngularDimension 对象的实例时，构造函数会接受一组可选参数。创建新的 LineAngularDimension2 对象时需提供下列参数：

- 尺寸界线 1 的起点（XLine1Start 属性）
- 尺寸界线 1 的终点（XLine1End 属性）
- 尺寸界线 2 的起点（XLine2Start 属性）
- 尺寸界线 2 的终点（XLine2End 属性）
- 圆弧点（代表尺寸线位置）（ArcPoint 属性）
- 标注文字（DimensionText 属性）
- 标注样式（DimensionStyleName 属性或 DimensionStyle 属性）

创建新的 Point3AngularDimension 对象时需提供下列参数：

- 中心点（CenterPoint 属性）
- 尺寸界线 1 的点（XLine1Point 属性）
- 尺寸界线 2 的点(XLine2Point 属性）
- 圆弧点（代表尺寸线位置）（ArcPoint 属性）
- 标注文字（DimensionText 属性）
- 标注样式（DimensionStyleName 属性或 DimensionStyle 属性）

创建角度标注 Create an angular dimension

本例在模型空间创建一个角度标注。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateAngularDimension")> _
Public Sub CreateAngularDimension()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 ModelSpace
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 建立角度标注
      Dim acLinAngDim As LineAngularDimension2 = New LineAngularDimension2()
      acLinAngDim.XLine1Start = New Point3d(0, 5, 0)
      acLinAngDim.XLine1End = New Point3d(1, 7, 0)
      acLinAngDim.XLine2Start = New Point3d(0, 5, 0)
      acLinAngDim.XLine2End = New Point3d(1, 3, 0)
      acLinAngDim.ArcPoint = New Point3d(3, 5, 0)
      acLinAngDim.DimensionStyle = acCurDb.Dimstyle

      '' 添加新对象到模型空间和事务中
      acBlkTblRec.AppendEntity(acLinAngDim)
      acTrans.AddNewlyCreatedDBObject(acLinAngDim, True)

      '' 提交修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateAngularDimension")]
public static void CreateAngularDimension()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录 ModelSpace
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 建立角度标注
      LineAngularDimension2 acLinAngDim = new LineAngularDimension2();
      acLinAngDim.XLine1Start = new Point3d(0, 5, 0);
      acLinAngDim.XLine1End = new Point3d(1, 7, 0);
      acLinAngDim.XLine2Start = new Point3d(0, 5, 0);
      acLinAngDim.XLine2End = new Point3d(1, 3, 0);
      acLinAngDim.ArcPoint = new Point3d(3, 5, 0);
      acLinAngDim.DimensionStyle = acCurDb.Dimstyle;

      // 添加新对象到模型空间和事务中
      acBlkTblRec.AppendEntity(acLinAngDim);
      acTrans.AddNewlyCreatedDBObject(acLinAngDim, true);

      // 提交修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub CreateAngularDimension()
    Dim dimObj As AcadDimAngular
    Dim angVert(0 To 2) As Double
    Dim FirstPoint(0 To 2) As Double
    Dim SecondPoint(0 To 2) As Double
    Dim TextPoint(0 To 2) As Double

    ' 定义标注
    angVert(0) = 0
    angVert(1) = 5
    angVert(2) = 0
    FirstPoint(0) = 1
    FirstPoint(1) = 7
    FirstPoint(2) = 0
    SecondPoint(0) = 1
    SecondPoint(1) = 3
    SecondPoint(2) = 0
    TextPoint(0) = 3
    TextPoint(1) = 5
    TextPoint(2) = 0

    ' 在模型空间创建角度标注
    Set dimObj = ThisDrawing.ModelSpace. _
                 AddDimAngular(angVert, FirstPoint, SecondPoint, TextPoint)
    ZoomAll
End Sub
```

4.2.4 创建折弯的半径标注

折弯半径标注测量对象的半径并显示带有半径符号前缀的标注文字。出现下列情况时你可能需要使用折弯半径标注：

- 对象的中心点位于布局外，或超出了模型区域，没有足够空间容纳半径标注；
- 对象的半径非常大；

通过创建 RadialDimensionLarge 对象实例来建立折弯半径标注，RadialDimensionLarge 对象的构造函数可以接受一组可选参数。

创建新 RadialDimensionLarge 对象实例时应提供下列参数：

- 中心点 (Center 属性)
- 弦点 (ChordPoint 属性)
- 覆盖中心点 (OverrideCenter 属性)

- 折弯线位置 (JogPoint 属性)
- 折弯线角度 (JogAngle 属性)
- 标注文字 (DimensionText 属性)
- 标注样式 (DimensionStyleName 属性或 DimensionStyle 属性)

创建折弯半径标注 Create a jogged radius dimension

本示例在模型空间建立一个折弯半径标注。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateJoggedDimension")> _
Public Sub CreateJoggedDimension()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 ModelSpace
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建一个大半径标注
      Dim acRadDimLrg As RadialDimensionLarge = New RadialDimensionLarge()
      acRadDimLrg.Center = New Point3d(-3, -4, 0)
      acRadDimLrg.ChordPoint = New Point3d(2, 7, 0)
      acRadDimLrg.OverrideCenter = New Point3d(0, 2, 0)
      acRadDimLrg.JogPoint = New Point3d(1, 4.5, 0)
      acRadDimLrg.JogAngle = 0.707
```

```
        acRadDimLrg.DimensionStyle = acCurDb.Dimstyle

        '' 将新对象添加到模型空间和事务
        acBlkTblRec.AppendEntity(acRadDimLrg)
        acTrans.AddNewlyCreatedDBObject(acRadDimLrg, True)

        '' 提交修改，关闭事务
        acTrans.Commit()
    End Using

End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateJoggedDimension")]
public static void CreateJoggedDimension()
{
    // 获取当前数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 以读模式打开块表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开块表记录 ModelSpace
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // 创建一个大半径标注
        RadialDimensionLarge acRadDimLrg = new RadialDimensionLarge();
        acRadDimLrg.Center = new Point3d(-3, -4, 0);
        acRadDimLrg.ChordPoint = new Point3d(2, 7, 0);
        acRadDimLrg.OverrideCenter = new Point3d(0, 2, 0);
```

```
        acRadDimLrg.JogPoint = new Point3d(1, 4.5, 0);
        acRadDimLrg.JogAngle = 0.707;
        acRadDimLrg.DimensionStyle = acCurDb.Dimstyle;

        // 将新对象添加到模型空间和事务
        acBlkTblRec.AppendEntity(acRadDimLrg);
        acTrans.AddNewlyCreatedDBObject(acRadDimLrg, true);

        // 提交修改，关闭事务
        acTrans.Commit();
    }

}
```

VBA/ActiveX 代码参考

```
Sub CreateJoggedDimension()
    Dim dimObj As AcadDimRadialLarge
    Dim centerPoint(0 To 2) As Double
    Dim chordPoint(0 To 2) As Double
    Dim centerOverPoint(0 To 2) As Double
    Dim jogPoint(0 To 2) As Double
    Dim jogAngle As Double

    ' 定义标注
    centerPoint(0) = -3: centerPoint(1) = -4: centerPoint(2) = 0
    chordPoint(0) = 2: chordPoint(1) = 7: chordPoint(2) = 0
    centerOverPoint(0) = 0: centerOverPoint(1) = 2: centerOverPoint(2) = 0
    jogPoint(0) = 1: jogPoint(1) = 4.5: jogPoint(2) = 0
    jogAngle = 0.707

    ' 在模型空间创建折弯标注
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimRadialLarge(centerPoint, chordPoint, _
                                      centerOverPoint, jogPoint, _
                                      jogAngle)

    ZoomAll
End Sub
```

4.2.5 创建弧长标注

弧长标注就是沿一条圆弧线测量其长度，并显示带有圆弧符号的标注文字，圆弧符号要么在标注文字的上方，要么在标注文字前缀位置。当需要标注圆弧的实际长度而不是圆弧起止点之间的距离时，就需要用到弧长标注。

通过创建 ArcDimension 对象实例来建立弧长标注。 创建 ArcDimension 对象实例时，需要提供如下一组参数来定义标注：

- 中心点(Center 属性)
- 尺寸界线 1 原点 (XLine1Point 属性)
- 尺寸界线 2 原点 (XLine2Point 属性)
- 圆弧点 (ArcPoint 属性)
- 标注文字 (DimensionText 属性)
- 标注样式(DimensionStyleName 属性或 DimensionStyle 属性)

注　系统变量 DIMARCSYM 控制圆弧符号显示与否以及相对于标注文字的显示位置。

创建弧长标注 Create an arc length dimension

本例在模型空间创建一个圆弧长度标注。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateArcLengthDimension")> _
Public Sub CreateArcLengthDimension()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)
```

```
        '' 以写模式打开块表记录 ModelSpace
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' 创建一个弧长标注
        Dim acArcDim As ArcDimension = New ArcDimension(New Point3d(4.5, 1.5, 0), _
                                                        New Point3d(8, 4.25, 0), _
                                                        New Point3d(0, 2, 0), _
                                                        New Point3d(5, 7, 0), _
                                                        "<>", _
                                                        acCurDb.Dimstyle)

        '' 将新对象添加到模型空间和事务
        acBlkTblRec.AppendEntity(acArcDim)
        acTrans.AddNewlyCreatedDBObject(acArcDim, True)

        '' 提交修改，关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateArcLengthDimension")]
public static void CreateArcLengthDimension()
{
    // 获取当前数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 以读模式打开块表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开块表记录 ModelSpace
```

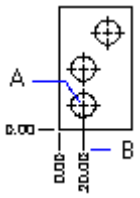
```
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建一个弧长标注
      ArcDimension acArcDim = new ArcDimension(new Point3d(4.5, 1.5, 0),
                                               new Point3d(8, 4.25, 0),
                                               new Point3d(0, 2, 0),
                                               new Point3d(5, 7, 0),
                                               "<>",
                                               acCurDb.Dimstyle);

      // 将新对象添加到模型空间和事务
      acBlkTblRec.AppendEntity(acArcDim);
      acTrans.AddNewlyCreatedDBObject(acArcDim, true);

      // 提交修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub CreateArcLengthDimension()
    Dim dimObj As AcadDimArcLength
    Dim arcCenterPoint(0 To 2) As Double
    Dim firstPoint(0 To 2) As Double
    Dim secondPoint(0 To 2) As Double
    Dim arcPoint(0 To 2) As Double

    ' 定义标注
    arcCenterPoint(0) = 4.5: arcCenterPoint(1) = 1.5: arcCenterPoint(2) = 0
    firstPoint(0) = 8: firstPoint(1) = 4.25: firstPoint(2) = 0
    secondPoint(0) = 0: secondPoint(1) = 2: secondPoint(2) = 0
    arcPoint(0) = 5: arcPoint(1) = 7: arcPoint(2) = 0

    ' 创建弧长标注
    Set dimObj = ThisDrawing.ModelSpace. _
                 AddDimArc(arcCenterPoint, firstPoint, _
                 secondPoint, arcPoint)

    ZoomAll
End Sub
```

4.2.6 创建坐标标注

坐标（或称基准）标注，就是测量从原点（又称基准）到标注功能点（比如零件上的一个孔）的垂直距离。这些标注能够通过保持从基准到功能点的精确偏移来避免尺寸累积错误。



坐标标注由带引线的 *X* 坐标或 *Y* 坐标构成。*X*-基准坐标标注沿 *X* 轴测量从基准点到功能点的距离，*Y*-基准坐标标注沿 *Y* 轴测量从基准点到功能点的距离。AutoCAD 使用当前用户坐标系统（UCS）的原点来确定要测量的坐标。标注使用的是坐标的绝对值。

不管当前标注样式的标注方向是如何定义的，标注文字只与坐标引线对齐。你可以接受默认测量值，或用你自己的数值替代。

我们通过创建 OrdinateDimension 对象的实例来创建坐标标注。 OrdinateDimension 对象的构造函数需要我们提供下列参数：

- 使用 *X* 轴？ (UsingXAxis 属性)
- 定义标注点 (DefiningPoint 属性)
- 引线终点 (LeaderEndPoint 属性)
- 标注文字(DimensionText 属性)
- 标注样式(DimensionStyleName 属性或 DimensionStyle 属性)

向 OrdinateDimension 对象的构造函数传递参数值时，第一个参数（UsingXAxis）是个布尔（boolean）标志，表示所创建的标注是 *X*-基准坐标标注还是 *Y*-基准坐标标注。该值为 TRUE 表示创建 *X*-基准坐标标注；该值为 FALSE 表示创建 *Y*-基准坐标标注。

创建一个坐标标注 Create an ordinate dimension

本例在模型空间创建一个坐标标注。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateOrdinateDimension")> _
Public Sub CreateOrdinateDimension()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database
```

```
  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 ModelSpace
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建坐标标注
      Dim acOrdDim As OrdinateDimension = New OrdinateDimension()
      acOrdDim.UsingXAxis = True
      acOrdDim.DefiningPoint = New Point3d(5, 5, 0)
      acOrdDim.LeaderEndPoint = New Point3d(10, 5, 0)
      acOrdDim.DimensionStyle = acCurDb.Dimstyle

      '' 将新对象添加到模型空间和事务
      acBlkTblRec.AppendEntity(acOrdDim)
      acTrans.AddNewlyCreatedDBObject(acOrdDim, True)

      '' 提交修改，关闭事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateOrdinateDimension")]
public static void CreateOrdinateDimension()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
```

```
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录 ModelSpace
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建坐标标注
      OrdinateDimension acOrdDim = new OrdinateDimension();
      acOrdDim.UsingXAxis = true;
      acOrdDim.DefiningPoint = new Point3d(5, 5, 0);
      acOrdDim.LeaderEndPoint = new Point3d(10, 5, 0);
      acOrdDim.DimensionStyle = acCurDb.Dimstyle;

      // 将新对象添加到模型空间和事务
      acBlkTblRec.AppendEntity(acOrdDim);
      acTrans.AddNewlyCreatedDBObject(acOrdDim, true);

      // 提交修改，关闭事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub CreateOrdinateDimension()
    Dim dimObj As AcadDimOrdinate
    Dim definingPoint(0 To 2) As Double
    Dim leaderEndPoint(0 To 2) As Double
    Dim useXAxis As Boolean

    ' 定义标注
    definingPoint(0) = 5
    definingPoint(1) = 5
    definingPoint(2) = 0
    leaderEndPoint(0) = 10
    leaderEndPoint(1) = 5
    leaderEndPoint(2) = 0
    useXAxis = True

    ' 在模型空间创建坐标标注
```

```
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimOrdinate(definingPoint, _
                    leaderEndPoint, useXAxis)

    ZoomAll
End Sub
```

4.3 编辑标注

像使用 AutoCAD 中的其他图形对象一样，我们可以用标注对象提供的方法和属性来对标注进行编辑。

大多数标注对象都提供了下述属性：

DimensionStyle

表示标注样式的对象 ID。

DimensionStyleName

表示标注样式的名称。

DimensionText

表示用户定义的标注文字。

HorizontalRotation

表示标注旋转的角度（用弧度表示）。

Measurement

表示标注的实际测量值。

TextPosition

表示标注文字的位置。

TextRotation

表示标注文字的旋转角度。

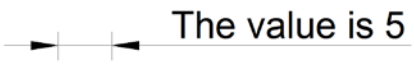
另外，除了使用指定属性和方法修改标注对象外，我们还可以对标注对象进行复制和转换。关于复制和转换对象的内容，见（§3.4 编辑命名对象和 2D 对象）。

4.3.1 替换标注文字

标注显示的尺寸值可以用 DimensionText 属性来替换。使用这一属性我们可以将显示的尺寸值整个替换掉，或给尺寸值添加文字。在替换文字里，使用字符串”<>”代表原测量值。

修改标注文字 Modify dimension text

本例给原有标注值添加些文字，使两者均显示出来，结果如下图。



VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("OverrideDimensionText")> _
Public Sub OverrideDimensionText()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 ModelSpace
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建一个对齐标注
      Dim acAliDim As AlignedDimension = New AlignedDimension()
```

```
        acAliDim.XLine1Point = New Point3d(5, 3, 0)
        acAliDim.XLine2Point = New Point3d(10, 3, 0)
        acAliDim.DimLinePoint = New Point3d(7.5, 5, 0)
        acAliDim.DimensionStyle = acCurDb.Dimstyle

        '' 改写标注文字
        acAliDim.DimensionText = "The value is <>"

        '' 将新对象添加到模型空间和事务
        acBlkTblRec.AppendEntity(acAliDim)
        acTrans.AddNewlyCreatedDBObject(acAliDim, True)

        '' 提交修改，关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("OverrideDimensionText")]
public static void OverrideDimensionText()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录 ModelSpace
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建一个对齐标注
```

```
        AlignedDimension acAliDim = new AlignedDimension();
        acAliDim.XLine1Point = new Point3d(5, 3, 0);
        acAliDim.XLine2Point = new Point3d(10, 3, 0);
        acAliDim.DimLinePoint = new Point3d(7.5, 5, 0);
        acAliDim.DimensionStyle = acCurDb.Dimstyle;

        // 改写标注文字
        acAliDim.DimensionText = "The value is <>";

        // 将新对象添加到模型空间和事务
        acBlkTblRec.AppendEntity(acAliDim);
        acTrans.AddNewlyCreatedDBObject(acAliDim, true);

        // 提交修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub OverrideDimensionText()
    Dim dimObj As AcadDimAligned
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim location(0 To 2) As Double

    ' 定义标注
    point1(0) = 5#: point1(1) = 3#: point1(2) = 0#
    point2(0) = 10#: point2(1) = 3#: point2(2) = 0#
    location(0) = 7.5: location(1) = 5#: location(2) = 0#

    ' 在模型空间创建对齐标注对象
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimAligned(point1, point2, location)

    ' 修改标注文本串
    dimObj.TextOverride = "The value is <>"
    dimObj.Update
End Sub
```

4.4 使用标注样式

一个命名了的标注样式就是确定标注外观的一组设置。使用命名标注样式，我们可以给绘图中的尺寸标注建立和执行一个标注。

所有的尺寸标注都是使用当前的编著样式创建的。在创建尺寸标注之前，如果你没有定义和应用样式，AutoCAD 会为你应用名为 STANDARD 的默认样式。可以使用当前数据库的 Dimstyle 属性来设置当前标注样式。

建立一个标注样式从命名并保存样式开始，新样式是在当前样式的基础上建立，并包含所有的设置，这些设置定义了标注的组件、文字位置以及注释的外观等。这里的注释指基本单位、换算单位、公差及文字等。

4.4.1 创建、修改、拷贝标注样式

创建新标注样式的方法是，先创建一个 DimStyleTableRecord 对象的实例，然后用 Add 方法将其添加到 DimStyleTable 标注样式表里。添加到样式表之前，使用 Name 属性给新样式命名。

还可以拷贝现有样式，或改写现有样式。从源对象拷贝标注样式用 CopyFrom()方法。源对象可以是另一个 DimStyleTableRecord 对象， 一个 Dimension 对象、 Tolerance 对象或 Leader 对象， 甚或一个 Database 对象。如果是从别的标注样式拷贝样式设置，则会完全复制当前样式。如果是从一个 Dimension 对象、 Tolerance 对象或 Leader 对象拷贝样式设置，则当前设置，包括任何被修改过的对象，都会复制到新样式。如果是拷贝 Database 对象的当前样式，则标注样式连同所有修改过的绘图都会复制到新样式。

拷贝并修改标注样式

本例新创建 3 个标注样式，并分别从当前数据库、给定的标注样式、给定的尺寸标注拷贝当前设置到 3 个新样式。按照下面的步骤进行相应设置，然后再运行代码示例，你会发现生成的标注样式是不一样的。

1. 新建一个绘图并使之为当前图形；
2. 在新图形中创建一个线性标注，该标注应为图形中唯一的一个对象；
3. 将尺寸线的颜色改为黄色；
4. 将系统变量 DIMCLRD 的值改为 5（蓝色）；
5. 运行下面的示例代码：

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
```

```
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CopyDimStyles")> _
Public Sub CopyDimStyles()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以读模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForRead)

      Dim acObj As Object = Nothing
      For Each acObjId As ObjectId In acBlkTblRec
          '' 获取模型空间里的第一个对象
          acObj = acTrans.GetObject(acObjId, _
                                    OpenMode.ForRead)

          Exit For
      Next

      '' 以读模式打开 DimStyle 表
      Dim acDimStyleTbl As DimStyleTable
      acDimStyleTbl = acTrans.GetObject(acCurDb.DimStyleTableId, _
                                        OpenMode.ForRead)

      Dim strDimStyleNames(2) As String
      strDimStyleNames(0) = "Style 1 copied from a dim"
      strDimStyleNames(1) = "Style 2 copied from Style 1"
      strDimStyleNames(2) = "Style 3 copied from the running drawing values"

      Dim nCnt As Integer = 0

      '' 定义一个第一个标注样式的引用，一会儿要用到
```

```
      Dim acDimStyleTblRec1 As DimStyleTableRecord = Nothing

      '' 遍历标注样式名称数组
      For Each strDimStyleName As String In strDimStyleNames
          Dim acDimStyleTblRec As DimStyleTableRecord
          Dim acDimStyleTblRecCopy As DimStyleTableRecord = Nothing

          '' 检查标注样式是否存在
          If acDimStyleTbl.Has(strDimStyleName) = False Then
              If acDimStyleTbl.IsWriteEnabled = False
ThenacDimStyleTbl.UpgradeOpen()

              acDimStyleTblRec = New DimStyleTableRecord()
              acDimStyleTblRec.Name = strDimStyleName

              acDimStyleTbl.Add(acDimStyleTblRec)
              acTrans.AddNewlyCreatedDBObject(acDimStyleTblRec, True)
          Else
              acDimStyleTblRec =
acTrans.GetObject(acDimStyleTbl(strDimStyleName), _
                                                  OpenMode.ForWrite)
          End If

          '' 确定新的标注样式是如何填充的
          Select Case nCnt
              '' 将标注对象的值赋给新标注样式
              Case 0
                  Try
                      '' 强制将对象类型转换为标注类型
                      Dim acDim As RotatedDimension = acObj

                      '' 从标注复制标注样式数据并
                      '' 设置标注样式的名称。
                      '' （复制的设置是未命名的）
                      acDimStyleTblRecCopy = acDim.GetDimstyleData()
                      acDimStyleTblRec1 = acDimStyleTblRec
                  Catch
                      '' 对象不是个标注
                  End Try

                  '' 将标注样式的值赋给新样式
              Case 1
                  acDimStyleTblRecCopy = acDimStyleTblRec1
```

```
                '' 将当前图形的值赋给标注样式
            Case 2
                acDimStyleTblRecCopy = acCurDb.GetDimstyleData()
        End Select

        '' 复制标注设置，设标注样式名称
        acDimStyleTblRec.CopyFrom(acDimStyleTblRecCopy)
        acDimStyleTblRec.Name = strDimStyleName

        '' 注销复制的标注样式，回收内存
        acDimStyleTblRecCopy.Dispose()

        nCnt = nCnt + 1
    Next

    '' 提交修改，注销事务，回收内存
    acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CopyDimStyles")]
public static void CopyDimStyles()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以读模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
```

```
                                        OpenMode.ForRead) as BlockTableRecord;

      object acObj = null;
      foreach (ObjectId acObjId in acBlkTblRec)
      {
          // 获取模型空间里的第一个对象
          acObj = acTrans.GetObject(acObjId,
                                    OpenMode.ForRead);

          break;
      }

      // 以读模式打开 DimStyle 表
      DimStyleTable acDimStyleTbl;
      acDimStyleTbl = acTrans.GetObject(acCurDb.DimStyleTableId,
                                        OpenMode.ForRead) as DimStyleTable;

      string[] strDimStyleNames = new string[3];
      strDimStyleNames[0] = "Style 1 copied from a dim";
      strDimStyleNames[1] = "Style 2 copied from Style 1";
      strDimStyleNames[2] = "Style 3 copied from the running drawing values";

      int nCnt = 0;

      // 定义一个第一个标注样式的引用，一会儿要用到
      DimStyleTableRecord acDimStyleTblRec1 = null;

      // 遍历标注样式名称数组
      foreach (string strDimStyleName in strDimStyleNames)
      {
          DimStyleTableRecord acDimStyleTblRec;
          DimStyleTableRecord acDimStyleTblRecCopy = null;

          // 检查标注样式是否存在
          if (acDimStyleTbl.Has(strDimStyleName) == false)
          {
              if (acDimStyleTbl.IsWriteEnabled == false)
acDimStyleTbl.UpgradeOpen();

              acDimStyleTblRec = new DimStyleTableRecord();
              acDimStyleTblRec.Name = strDimStyleName;

              acDimStyleTbl.Add(acDimStyleTblRec);
              acTrans.AddNewlyCreatedDBObject(acDimStyleTblRec, true);
```

```
            }
            else
            {
                acDimStyleTblRec =
acTrans.GetObject(acDimStyleTbl[strDimStyleName],
                                                    OpenMode.ForWrite) as
DimStyleTableRecord;
            }

            // 确定新的标注样式是如何填充的
            switch ((int)nCnt)
            {
                // 将标注对象的值赋给新标注样式
                case 0:
                    try
                    {
                        // 强制将对象类型转换为标注类型
                        Dimension acDim = acObj as Dimension;

                        // 从标注复制标注样式数据并
                        // 设置标注样式的名称
                        // （复制的设置是未命名的）
                        acDimStyleTblRecCopy = acDim.GetDimstyleData();
                        acDimStyleTblRec1 = acDimStyleTblRec;
                    }
                    catch
                    {
                        // 对象不是个标注
                    }

                    break;
                // 将标注样式的值赋给新样式
                case 1:
                    acDimStyleTblRecCopy = acDimStyleTblRec1;
                    break;
                // 将当前图形的值赋给标注样式
                case 2:
                    acDimStyleTblRecCopy = acCurDb.GetDimstyleData();
                    break;
            }

            // 复制标注设置，设标注样式名称
            acDimStyleTblRec.CopyFrom(acDimStyleTblRecCopy);
            acDimStyleTblRec.Name = strDimStyleName;
```

```
            // 注销复制的标注样式，回收内存
            acDimStyleTblRecCopy.Dispose();

            nCnt = nCnt + 1;
        }

        // 提交修改，注销事务，回收内存
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CopyDimStyles()
    Dim newStyle1 As AcadDimStyle
    Dim newStyle2 As AcadDimStyle
    Dim newStyle3 As AcadDimStyle

    Set newStyle1 = ThisDrawing.DimStyles. _
                        Add("Style 1 copied from a dim")
    Call newStyle1.CopyFrom(ThisDrawing.ModelSpace(0))

    Set newStyle2 = ThisDrawing.DimStyles. _
                        Add("Style 2 copied from Style 1")
    Call newStyle2.CopyFrom(ThisDrawing.DimStyles. _
                                Item("Style 1 copied from a dim"))

    Set newStyle2 = ThisDrawing.DimStyles. _
                        Add("Style 3 copied from the running drawing values")
    Call newStyle2.CopyFrom(ThisDrawing)
End Sub
```

使用 DIMSTYLE 命令打开标注样式管理器，你会看到有三个新标注样式在里面，样式 1 的尺寸线应该是黄色的，样式 2 应该与样式 1 一样，样式 3 的尺寸线应该是蓝色的。

4.4.2 修改标注的样式

每个尺寸标注都有能力修改由标注样式赋予它的设置。以下属性(/系统变量)可用于大多数标注对象：

Dimatfit

表示尺寸线只显示在尺寸界线的内测，并强制标注文字和箭头显示在尺寸界线的内测或外侧。

Dimaltrnd

表示换算单位的舍入。

Dimasz

表示尺寸线箭头、引线箭头和勾线的大小。

Dimaunit

表示角度单位格式。

Dimblk1, Dimblk2

表示用于尺寸线箭头的块。

Dimcen

表示径向标注中圆心标记的类型和大小。

Dimclrd

表示标注、引线、公差等对象的尺寸线的颜色。

Dimclre

表示尺寸界线的颜色。

Dimclrt

表示标注对象和公差对象的文字的颜色。

Dimdec

表示尺寸或公差的基本单元要显示的小数位数。

Dimdsep

表示用于十进制尺寸和公差值的小数点分隔符的字符。

Dimexe

表示尺寸界线伸出尺寸线部分的长度。

Dimexo

表示尺寸界线偏离标注原点的距离。

Dimfrac

表示尺寸和公差中分数值的格式。

Dimgap

表示当你打断尺寸线以适应标注文字时标注文字与尺寸线间的距离。

Dimlfac

表示线性标注测量时用的全局比例系数。

Dimltex1, Dimltex2

表示尺寸界线的线型。

Dimlwd

表示尺寸线的线宽。

Dimlwe

表示尺寸界线的线宽。

Dimjust

表示标注文字水平对齐。

Dimrnd

表示标注舍入。

Dimsd1, Dimsd2

表示尺寸线 1、尺寸线 2 的开关（显示或隐藏）。

Dimse1, Dimse2

表示尺寸界线 1、尺寸线界 2 的开关（显示或隐藏）。

Dimtad

表示标注文字相对于尺寸线的垂直位置（上方、下方、居中、外部等）。

Dimtdec

表示基本尺寸公差值的精度。

Dimtfac

表示公差值的字高相对于标注文字字高的比例系数。

Dimlunit

表示除角度外的所有尺寸的单位格式。

Dimtih

表示标注文字是否绘在尺寸界线内测。

Dimtm

表示尺寸文字的最小公差限制。

Dimtmove

表示当文字移动时如何绘制标注文字。

Dimtofl

表示即使标注文字放在尺寸界线外侧时，尺寸线是否绘制在尺寸界线之间。

Dimtoh

表示对于坐标标注以外的所有标注类型，放在尺寸界线外侧的标注文字的位置。（尺寸界线外侧的什么位置？）

Dimtol

表示公差是否和尺寸文字一起显示。

Dimtolj

表示公差值相对于公称尺寸文字的垂直对齐方式。

Dimtp

表示尺寸文字的最小公差限制。

Dimtxt

表示尺寸或公差文字的字高。

Dimzin

表示是否消隐测量的尺寸值中的前导零（如“0.09”显示为“.09”）、尾零、及零英尺和零英寸。

Prefix

表示尺寸值前缀。

Suffix

表示尺寸值后缀。

TextPrecision

表示角度尺寸的精度。

TextPosition

表示标注文字的位置。

TextRotation

表示标注文字的旋转角度。

为对齐标注输入一个用户自定义后缀

本例在模型空间创建一个对齐标注，然后使用 Suffix 属性让用户修改标注文字的后缀。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddDimensionTextSuffix")> _
Public Sub AddDimensionTextSuffix()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 ModelSpace
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建对齐标注
      Dim acAliDim As AlignedDimension = New AlignedDimension()
      acAliDim.XLine1Point = New Point3d(0, 5, 0)
      acAliDim.XLine2Point = New Point3d(5, 5, 0)
      acAliDim.DimLinePoint = New Point3d(5, 7, 0)
      acAliDim.DimensionStyle = acCurDb.Dimstyle

      '' 将新对象添加到模型空间并进行事务记录
      acBlkTblRec.AppendEntity(acAliDim)
      acTrans.AddNewlyCreatedDBObject(acAliDim, True)

      '' 给标注文字添加后缀
      Dim pStrOpts As PromptStringOptions = New PromptStringOptions("")

      pStrOpts.Message = vbLf & "Enter a new text suffix for the dimension: "
```

```
        pStrOpts.AllowSpaces = True
        Dim pStrRes As PromptResult = acDoc.Editor.GetString(pStrOpts)

        If pStrRes.Status = PromptStatus.OK Then
            acAliDim.Suffix = pStrRes.StringResult
        End If

        '' 提交修改，注销事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("AddDimensionTextSuffix")]
public static void AddDimensionTextSuffix()
{
    // 获取当前数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 以读模式打开块表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开块表记录 ModelSpace
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        // 创建对齐标注
        AlignedDimension acAliDim = new AlignedDimension();
        acAliDim.XLine1Point = new Point3d(0, 5, 0);
        acAliDim.XLine2Point = new Point3d(5, 5, 0);
        acAliDim.DimLinePoint = new Point3d(5, 7, 0);
```

```
      acAliDim.DimensionStyle = acCurDb.Dimstyle;

      // 将新对象添加到模型空间并进行事务记录
      acBlkTblRec.AppendEntity(acAliDim);
      acTrans.AddNewlyCreatedDBObject(acAliDim, true);

      // 给标注文字添加后缀
      PromptStringOptions pStrOpts = new PromptStringOptions("");

      pStrOpts.Message = "\nEnter a new text suffix for the dimension: ";
      pStrOpts.AllowSpaces = true;
      PromptResult pStrRes = acDoc.Editor.GetString(pStrOpts);

      if (pStrRes.Status == PromptStatus.OK)
      {
          acAliDim.Suffix = pStrRes.StringResult;
      }

      // 提交修改，注销事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddDimensionTextSuffix()
    Dim dimObj As AcadDimAligned
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim location(0 To 2) As Double
    Dim suffix As String

    ' 定义标注
    point1(0) = 0: point1(1) = 5: point1(2) = 0
    point2(0) = 5: point2(1) = 5: point2(2) = 0
    location(0) = 5: location(1) = 7: location(2) = 0

    '在模型空间创建对齐标注
    Set dimObj = ThisDrawing.ModelSpace. _
                    AddDimAligned(point1, point2, location)

    ThisDrawing.Application.ZoomAll
    ' 允许使用者修改标注文字后缀
    suffix = ThisDrawing.Utility. _
                GetString(True, vbLf & "Enter a new text " & _
```

```
                                        "suffix for the dimension: ")

    ' 提交修改
    dimObj.TextSuffix = suffix
    ThisDrawing.Regen acAllViewports
End Sub
```

4.5 模型空间和图纸空间的尺寸标注

我们可以在模型空间绘制尺寸标注，同样也可以在图纸空间绘制尺寸标注。不过，如果要标注的几何形状是在模型空间，最好还是在模型空间绘制尺寸标注，因为几何形状是在哪个空间绘制的，AutoCAD 就将定义点放在哪个空间。

如果几何形状是在模型空间绘制的，而你在图纸空间绘制其尺寸标注，那么，当你在模型空间视口使用编辑命令或改变显示的放大倍数时，图纸空间的标注将不会随着改变。当你从模型空间切换到图纸空间时，会发现图纸空间的尺寸标注还在原处没变。

如果你在图纸空间标注，并且将线性标注的全局比例系数（系统变量 DIMLFAC）设置成小于 0，那么测量到的距离是乘了 DIMLFAC 的绝对值的。而如果在模型空间标注，当 DIMLFAC 小于 0 时，会使用值 1.0。如果在“标注”提示符下修改变量值并选择“视口”选项，AutoCAD 会为 DIMLFAC 计算一个值。AutoCAD 计算模型空间到图纸空间的缩放比例，并将该值取负再赋给系统变量 DIMLFAC。

4.6 创建引线和注释

引线就是将文字注释与图形中的功能点连结起来的一条线。引线和注释是相关联的，就是说，如果修改注释，引线会跟着发生变化。不要把引线（Leader）对象同 AutoCAD 自动生成的作为尺寸线组成部分的引线混淆了。

4.6.1 创建引线

我们可以从图形的任一点或任一功能区创建引线（leader line），并控制引线的外观。引线可以是直线线段，或者光滑的样条曲线，引线的颜色由当前尺寸线颜色控制，引线的比例由当前标注样式确定的整体标注比例控制，引线箭头（有的话）的类型和大小也由当前标注样式定义的箭头控制。

通常有一小段称之为“勾线（hook line）”的水平基线将注释与引线连结起来。如果引线最后一段线段与水平线夹角大于 15 度，勾线会与多行文字及特征控制框一起出现。勾线的长度与单个箭头的长度相当。如果引线没有注释，就没有勾线。

通过创建 Leader 对象的实例来创建引线，Leader 对象的构造函数没有参数。AppendVertex() 方法用于定义所创建的引线的位置和长度。

创建一条引线 Create a leader line

本例在模型空间创建一条引线，该引线没有注释与之关联。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateLeader")> _
Public Sub CreateLeader()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建引线
      Dim acLdr As Leader = New Leader()
      acLdr.AppendVertex(New Point3d(0, 0, 0))
      acLdr.AppendVertex(New Point3d(4, 4, 0))
      acLdr.AppendVertex(New Point3d(4, 5, 0))
      acLdr.HasArrowHead = True

      '' 添加新对象到模型空间，记录事务
```

```
      acBlkTblRec.AppendEntity(acLdr)
      acTrans.AddNewlyCreatedDBObject(acLdr, True)

      '' 提交修改，回收内存
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

//在模型空间创建一条引线，该引线没有注释与之关联
[CommandMethod("CreateLeader")]
public static void CreateLeader()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建引线
      Leader acLdr = new Leader();
      acLdr.AppendVertex(new Point3d(0, 0, 0));
      acLdr.AppendVertex(new Point3d(4, 4, 0));
      acLdr.AppendVertex(new Point3d(4, 5, 0));
      acLdr.HasArrowHead = true;

      // 添加新对象到模型空间，记录事务
      acBlkTblRec.AppendEntity(acLdr);
```

```
        acTrans.AddNewlyCreatedDBObject(acLdr, true);

        // 提交修改，回收内存
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateLeader()
    Dim leaderObj As AcadLeader
    Dim points(0 To 8) As Double
    Dim leaderType As Integer
    Dim annotationObject As AcadObject

    points(0) = 0: points(1) = 0: points(2) = 0
    points(3) = 4: points(4) = 4: points(5) = 0
    points(6) = 4: points(7) = 5: points(8) = 0
    leaderType = acLineWithArrow
    Set annotationObject = Nothing

    ' 在模型空间创建引线对象
    Set leaderObj = ThisDrawing.ModelSpace. _
                        AddLeader(points, annotationObject, leaderType)

    ZoomAll
End Sub
```

4.6.2 给引线添加注释

引线注释可以是一个公差对象、Mtext（多行文字）对象或块参考对象。我们可以创建新注释，也可以添加现有注释的一个拷贝。将注释添加给引线的方法是，将注释对象的 ID 传给引线对象的 Annotation 属性。

4.6.3 引线关联

引线与其注释是相关联的，这样，当注释移动时，引线的末端点也会随之移动。当你移动文本注释和特征控制框注释时，根据注释与引线倒数第二个点的相对位置关系，最终引线线段会附着在注释的左边或右边交替变化。如果注释的中间点在引线倒数第二点的右侧，则引线附着在右侧，否则引线附着在左侧。

使用擦除、添加（添加块）或用 Wblock()方法等从图形中删除任何一个对象都会打破关联关系。如果在操作中一起复制引线和注释，新的备份也是关联的；如果是单独复制的，备份

就不关联了。如果关联关系因某个原因被打破了，如只复制引线或擦除了注释等，勾线也会从引线中删除。

关联引线与注释 Associate a leader to the annotation

本例先创建一个 MText 对象，然后使用 MText 对象作为注释创建一条引线。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("AddLeaderAnnotation")> _
Public Sub AddLeaderAnnotation()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建多行文字（MText）注释
      Dim acMText As MText = New MText()
      acMText.Contents = "Hello, World."
      acMText.Location = New Point3d(5, 5, 0)
      acMText.Width = 2

      '' 添加新对象到模型空间，记录事务
      acBlkTblRec.AppendEntity(acMText)
      acTrans.AddNewlyCreatedDBObject(acMText, True)

      '' 创建带注释的引线
      Dim acLdr As Leader = New Leader()
```

```
        acLdr.AppendVertex(New Point3d(0, 0, 0))
        acLdr.AppendVertex(New Point3d(4, 4, 0))
        acLdr.AppendVertex(New Point3d(4, 5, 0))
        acLdr.HasArrowHead = True

        '' 添加新对象到模型空间，记录事务
        acBlkTblRec.AppendEntity(acLdr)
        acTrans.AddNewlyCreatedDBObject(acLdr, True)

        ''给引线对象附加注释
        acLdr.Annotation = acMText.ObjectId
        acLdr.EvaluateLeader()

        '' 提交修改，回收内存
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;


//关联引线和注释
[CommandMethod("AddLeaderAnnotation")]
public static void AddLeaderAnnotation()
{
  // 获取当前数据库
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  // 启动事务
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
```

```
                                        OpenMode.ForWrite) as BlockTableRecord;

      // 创建多行文字（MText）注释
      MText acMText = new MText();
      acMText.Contents = "Hello, World.";
      acMText.Location = new Point3d(5, 5, 0);
      acMText.Width = 2;

      // 添加新对象到模型空间，记录事务
      acBlkTblRec.AppendEntity(acMText);
      acTrans.AddNewlyCreatedDBObject(acMText, true);

      // 创建带注释的引线
      Leader acLdr = new Leader();
      acLdr.AppendVertex(new Point3d(0, 0, 0));
      acLdr.AppendVertex(new Point3d(4, 4, 0));
      acLdr.AppendVertex(new Point3d(4, 5, 0));
      acLdr.HasArrowHead = true;

      // 添加新对象到模型空间，记录事务
      acBlkTblRec.AppendEntity(acLdr);
      acTrans.AddNewlyCreatedDBObject(acLdr, true);

      // 给引线对象附加注释
      acLdr.Annotation = acMText.ObjectId;
      acLdr.EvaluateLeader();

      // 提交修改，回收内存
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub AddLeaderAnnotation()
    Dim leaderObj As AcadLeader
    Dim mtextObj As AcadMText
    Dim points(0 To 8) As Double
    Dim insertionPoint(0 To 2) As Double
    Dim width As Double
    Dim leaderType As Integer
    Dim annotationObject As Object
    Dim textString As String, msg As String

    ' 在模型空间创建 MText 对象
```

```
    textString = "Hello, World."
    insertionPoint(0) = 5
    insertionPoint(1) = 5
    insertionPoint(2) = 0
    width = 2
    Set mtextObj = ThisDrawing.ModelSpace. _
                        AddMText(insertionPoint, width, textString)

    ' 引线数据
    points(0) = 0: points(1) = 0: points(2) = 0
    points(3) = 4: points(4) = 4: points(5) = 0
    points(6) = 4: points(7) = 5: points(8) = 0
    leaderType = acLineWithArrow

    ' 在模型空间创建引线对象并将 MText 对象与之关联
    Set annotationObject = mtextObj
    Set leaderObj = ThisDrawing.ModelSpace. _
                         AddLeader(points, annotationObject, leaderType)

    ZoomAll
End Sub
```

4.6.4 编辑引线关联

除了引线和注释之间的关联关系，引线及其注释是图形中的完全独立的对象。编辑引线不会影响注释，编辑注释也不会影响引线。

虽然文本注释创建时是使用 DIMCLRT、DIMTXT 和 DIMTXSTY 系统变量来定义它的颜色、高度和风格的，但它不能通过这些系统变量来修改，因为它不是一个真正的标注对象。文本注释必须用和任何其他 Mtext 对象相同的编辑方式进行编辑。

使用 Evaluate()方法来评估引线与所关联注释的关系，必要时该方法会调整引线的几何形状。

4.6.5 编辑引线

改变引线注释位置的任何修改都会影响引线末端点的位置，此外，旋转注释也会造成引线勾线（如果有的话）的旋转。

可以通过缩放其大小来调整引线。缩放引线时，注释会停留在相对于引线末端点相同的位置，但不进行缩放。除了缩放，你也可以移动、镜像、旋转引线。编辑引线使用 TransformBy()方法，修改相关的注释使用引线对象的成员属性和方法。

4.7 使用形位公差

形位公差显示的是一个功能特征的形式、轮廓、方向、位置和跳动等的偏差。我们在特征控制框里添加形位公差，这些控制框包含单个尺寸标注的全部公差信息。

4.7.1 创建形位公差

通过创建 FeatureControlFrame 对象实例来创建形位公差。FeatureControlFrame 对象的构造函数需要一组可选参数。创建新 FeatureControlFrame 对象时需要提供下列参数：

- 包括公差符号的文本串 (Text 属性)
- 插入点(Location 属性)
- 法向矢量 (Normal 属性)
- 方向矢量 (Direction 属性)

创建形位公差 Create a geometric tolerance

本例在模型空间创建一个简单的形位公差。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateGeometricTolerance")> _
Public Sub CreateGeometricTolerance()
  '' 获取当前数据库
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  '' 启动事务
  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)
```

```
        '' 以写模式打开块表记录模型空间
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForWrite)

        '' 创建形位公差（特征控制框）
        Dim acFcf As FeatureControlFrame = New FeatureControlFrame()
        acFcf.Text = "{\Fgdt;j}%%v{\Fgdt;n}0.001%%v%%v%%v%%v"
        acFcf.Location = New Point3d(5, 5, 0)

        '' 添加新对象到模型空间，作事务记录
        acBlkTblRec.AppendEntity(acFcf)
        acTrans.AddNewlyCreatedDBObject(acFcf, True)

        '' 提交修改，关闭事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateGeometricTolerance")]
public static void CreateGeometricTolerance()
{
    // 获取当前数据库
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    // 启动事务
    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 以读模式打开块表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开块表记录模型空间
        BlockTableRecord acBlkTblRec;
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;
```

```
        // 创建形位公差（特征控制框）
        FeatureControlFrame acFcf = new FeatureControlFrame();
        acFcf.Text = "{\\Fgdt;j}%%v{\\Fgdt;n}0.001%%v%%v%%v%%v";
        acFcf.Location = new Point3d(5, 5, 0);

        // 添加新对象到模型空间，作事务记录
        acBlkTblRec.AppendEntity(acFcf);
        acTrans.AddNewlyCreatedDBObject(acFcf, true);

        // 提交修改，关闭事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateGeometricTolerance()
    Dim toleranceObj As AcadTolerance
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim direction(0 To 2) As Double

    ' 定义公差对象
    textString = "{\Fgdt;j}%%v{\Fgdt;n}0.001%%v%%v%%v%%v"
    insertionPoint(0) = 5
    insertionPoint(1) = 5
    insertionPoint(2) = 0
    direction(0) = 1
    direction(1) = 0
    direction(2) = 0

    ' 在模型空间创建公差对象
    Set toleranceObj = ThisDrawing.ModelSpace. _
                           AddTolerance(textString, insertionPoint, direction)

    ZoomAll
End Sub
```

4.7.2 编辑形位公差

形位公差受多个系统变量和属性的影响。 下列系统变量和属性影响形位公差的外观：

DIMCLRD

控制特征控制框的颜色。

DIMCLRT

控制公差文字的颜色。

DIMGAP

控制特征控制框与文本之间的间距。

DIMTXT

控制公差文字的字号大小

DIMTXTSTY

控制公差文字的文字样式。

第 5 章 三维空间作业

多数 AutoCAD 图形都是由三维对象的二维视图构成的。尽管这种制图方法被广泛应用于建筑和工程领域，但还是有其局限性：图形都是 3D 对象的 2D 展示，必须进行目视解释（读图）。而且，由于每个视图都是单独绘制的，特别容易出现错误和歧义。 因此，你可能想创建真正的三维模型来代替二维视图展示。利用 AutoCAD 的绘图工具，我们可以创建出细致逼真的 3D 对象，并且可以使用各种方法进行编辑修改。

本章主要内容：

- 指定 3D 坐标
- 定义用户坐标系 UCS
- 坐标变换
- 创建 3D 对象
- 编辑 3D 对象
- 编辑 3D 实体

5.1 指定 3D 坐标

输入三维（3D）WCS 坐标，和输入二维（2D）WCS 坐标类似，除了指定 X 坐标值和 Y 坐标值外，再指定一个 Z 坐标值即可。2D 维坐标由一个 Point2D 对象表示，同样使用一个 Point3d 对象来表示 3D 坐标。AutoCAD .Net API 中的多数属性和方法都用到 3D 坐标。

定义、查询 2D/3D 多段线的坐标

本例创建两条多段线，每条有 3 个顶点。第一条是 2D 多段线，第二条是 3D 多段线。注意在创建 3D 多段线时，包含顶点的数组的长度被扩展为包括 Z 坐标了。程序最后查询多段线的坐标，并在消息框中显示出来。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Polyline_2D_3D")> _
```

```
Public Sub Polyline_2D_3D()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 注意创建 3D 多段线和创建 2D 多段线时
      '' 调用 AddVertexAt()函数和 AppendEntity()函数的顺序不一样！

      '' 用 2 条线段（3 个点）创建一条多段线
      Dim acPoly As Polyline = New Polyline()
      acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
      acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
      acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
      acPoly.ColorIndex = 1

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPoly)
      acTrans.AddNewlyCreatedDBObject(acPoly, True)

      '' 用 2 条线段（3 个点）创建一条 3D 多段线
      Dim acPoly3d As Polyline3d = New Polyline3d()
      acPoly3d.ColorIndex = 5

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPoly3d)
      acTrans.AddNewlyCreatedDBObject(acPoly3d, True)

      '' 给 3D 多段线添加顶点前，3D 多段线必须已在数据库中了
      Dim acPts3dPoly As Point3dCollection = New Point3dCollection()
      acPts3dPoly.Add(New Point3d(1, 1, 0))
      acPts3dPoly.Add(New Point3d(2, 1, 0))
      acPts3dPoly.Add(New Point3d(2, 2, 0))
```

```
        For Each acPt3d As Point3d In acPts3dPoly
            Dim acPolVer3d As PolylineVertex3d = New PolylineVertex3d(acPt3d)
            acPoly3d.AppendVertex(acPolVer3d)
            acTrans.AddNewlyCreatedDBObject(acPolVer3d, True)
        Next

        '' 获取多段线的顶点坐标
        Dim acPts2d As Point2dCollection = New Point2dCollection()
        For nCnt As Integer = 0 To acPoly.NumberOfVertices - 1
            acPts2d.Add(acPoly.GetPoint2dAt(nCnt))
        Next

        '' 获取 3D 多段线的顶点坐标
        Dim acPts3d As Point3dCollection = New Point3dCollection()
        For Each acObjIdVert As ObjectId In acPoly3d
            Dim acPolVer3d As PolylineVertex3d
            acPolVer3d = acTrans.GetObject(acObjIdVert, _
                                           OpenMode.ForRead)

            acPts3d.Add(acPolVer3d.Position)
        Next

        '' 显示坐标
        Application.ShowAlertDialog("2D polyline (red): " & vbLf & _
                                    acPts2d(0).ToString() & vbLf & _
                                    acPts2d(1).ToString() & vbLf & _
                                    acPts2d(2).ToString())

        Application.ShowAlertDialog("3D polyline (blue): " & vbLf & _
                                    acPts3d(0).ToString() & vbLf & _
                                    acPts3d(1).ToString() & vbLf & _
                                    acPts3d(2).ToString())

        '' 提交事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
```

```
[CommandMethod("Polyline_2D_3D")]
public static void Polyline_2D_3D()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 注意创建 3D 多段线和创建 2D 多段线时
      // 调用 AddVertexAt()函数和 AppendEntity()函数的顺序不一样！

      // 用 2 条线段（3 个点）创建一条多段线
      Polyline acPoly = new Polyline();
      acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
      acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
      acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
      acPoly.ColorIndex = 1;

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPoly);
      acTrans.AddNewlyCreatedDBObject(acPoly, true);

      // 用 2 条线段（3 个点）创建一条 3D 多段线
      Polyline3d acPoly3d = new Polyline3d();
      acPoly3d.ColorIndex = 5;

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPoly3d);
      acTrans.AddNewlyCreatedDBObject(acPoly3d, true);

      // 给 3D 多段线添加顶点前，3D 多段线必须已在数据库中了
      Point3dCollection acPts3dPoly = new Point3dCollection();
      acPts3dPoly.Add(new Point3d(1, 1, 0));
```

```
        acPts3dPoly.Add(new Point3d(2, 1, 0));
        acPts3dPoly.Add(new Point3d(2, 2, 0));

        foreach (Point3d acPt3d in acPts3dPoly)
        {
            PolylineVertex3d acPolVer3d = new PolylineVertex3d(acPt3d);
            acPoly3d.AppendVertex(acPolVer3d);
            acTrans.AddNewlyCreatedDBObject(acPolVer3d, true);
        }

        // 获取多段线的顶点坐标
        Point2dCollection acPts2d = new Point2dCollection();
        for (int nCnt = 0; nCnt < acPoly.NumberOfVertices; nCnt++)
        {
            acPts2d.Add(acPoly.GetPoint2dAt(nCnt));
        }

        // 获取 3D 多段线的顶点坐标
        Point3dCollection acPts3d = new Point3dCollection();
        foreach (ObjectId acObjIdVert in acPoly3d)
        {
            PolylineVertex3d acPolVer3d;
            acPolVer3d = acTrans.GetObject(acObjIdVert,
                                           OpenMode.ForRead) as PolylineVertex3d;
             acPts3d.Add(acPolVer3d.Position);
        }

        // 显示坐标
        Application.ShowAlertDialog("2D polyline (red): \n" +
                                    acPts2d[0].ToString() + "\n" +
                                    acPts2d[1].ToString() + "\n" +
                                    acPts2d[2].ToString());

        Application.ShowAlertDialog("3D polyline (blue): \n" +
                                    acPts3d[0].ToString() + "\n" +
                                    acPts3d[1].ToString() + "\n" +
                                    acPts3d[2].ToString());

        //提交事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub Polyline_2D_3D()
```

```
Dim pline2DObj As AcadLWPolyline
Dim pline3DObj As AcadPolyline

Dim points2D(0 To 5) As Double
Dim points3D(0 To 8) As Double

' 定义 3 个 2D 多段线点
points2D(0) = 1: points2D(1) = 1
points2D(2) = 1: points2D(3) = 2
points2D(4) = 2: points2D(5) = 2

'定义 3 个 3D 多段线点
points3D(0) = 1: points3D(1) = 1: points3D(2) = 0
points3D(3) = 2: points3D(4) = 1: points3D(5) = 0
points3D(6) = 2: points3D(7) = 2: points3D(8) = 0

' 创建 2D 轻量级多段线
Set pline2DObj = ThisDrawing.ModelSpace. _
                                AddLightWeightPolyline(points2D)
pline2DObj.Color = acRed
pline2DObj.Update

'创建 3D 多段线
Set pline3DObj = ThisDrawing.ModelSpace. _
                                AddPolyline(points3D)
pline3DObj.Color = acBlue
pline3DObj.Update

' 查询多段线坐标
Dim get2Dpts As Variant
Dim get3Dpts As Variant

get2Dpts = pline2DObj.Coordinates
get3Dpts = pline3DObj.Coordinates

' 显示坐标

MsgBox ("2D polyline (red): " & vbCrLf & "(" & _
        get2Dpts(0) & ", " & get2Dpts(1) & ")" & vbCrLf & "(" & _
        get2Dpts(2) & ", " & get2Dpts(3) & ")"& vbCrLf & "(" & _
        get2Dpts(4) & ", " & get2Dpts(5) & ")")

MsgBox ("3D polyline (blue): " & vbCrLf & "(" & _
        get3Dpts(0) & ", " & get3Dpts(1) & ", " & _
```

```
            get3Dpts(2) & ")" & vbCrLf & "(" & _
            get3Dpts(3) & ", " & get3Dpts(4) & ", " & _
            get3Dpts(5) & ")" & vbCrLf & "(" & _
            get3Dpts(6) & ", " & get3Dpts(7) & ", " & _
            get3Dpts(8) & ")")
End Sub
```

5.2 定义用户坐标系 UCS

我们通过定义一个用户坐标系（UCS）对象来改变原点(0, 0, 0)的位置以及 XY 平面和 Z 轴的方向。可以在 3D 空间的任何位置和任何方向上设置一个用户坐标系 UCS，而且需要多少个用户坐标系，就可以定义多少个用户坐标系，并且可以保存起来在需要的时候调入。坐标的输入和显示都是相对于当前用户坐标系的。

为了表示 UCS 的原点和方向，可以使用 Viewport 对象的 `IconAtOrigin` 属性或者使用 UCSICON 系统变量在 UCS 原点处显示 UCS 图标。如果 UCS 图标点亮了（`IconVisible` 属性）却没在原点显示出来，它就会显示在由 UCSORG 系统变量定义的 WCS 坐标处。

可以使用 UCSTable 对象的 Add()方法创建一个新的用户坐标系，该方法需要 4 个参数：原点坐标、*X* 轴和 *Y* 轴上的坐标、UCS 的名称。

在 AutoCAD®ActiveX Automation 中输入的所有坐标都是世界坐标系的。可以使用 `GetUCSMatrix` 方法返回已知 UCS 坐标系的变换矩阵。使用此变换矩阵可以计算出相应的 WCS 坐标。

可以使用 Document 对象的 ActiveUCS 属性来激活一个 UCS 坐标系使其成为当前坐标系。如果对当前 UCS 坐标系进行了修改，为使修改生效，必须将其重新设置为当前坐标系。重置的方法很简单，设置已修改了的 UCS 对象的 ActiveUCS 属性即可。

新创建一个 UCS 并设置为当前坐标系，然后将一个点的坐标转变为 UCS 坐标

下面的代码为图形新创建一个 UCS 并设置为当前坐标系，接着要求用户在图形中选取一个点，然后返回该点的 WCS 坐标和 UCS 坐标。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("NewUCS")> _
```

```
Public Sub NewUCS()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开 UCSTable
      Dim acUCSTbl As UcsTable
      acUCSTbl = acTrans.GetObject(acCurDb.UcsTableId, _
                                   OpenMode.ForRead)

      Dim acUCSTblRec As UcsTableRecord

      '' 检查 UCS 表中是否有“New_UCS”这条记录
      If acUCSTbl.Has("New_UCS") = False Then
          acUCSTblRec = New UcsTableRecord()
          acUCSTblRec.Name = "New_UCS"

          '' 以写模式打开 UCSTable
          acUCSTbl.UpgradeOpen()

          '' 往 UCSTable 添加新记录
          acUCSTbl.Add(acUCSTblRec)
          acTrans.AddNewlyCreatedDBObject(acUCSTblRec, True)
      Else
          acUCSTblRec = acTrans.GetObject(acUCSTbl("New_UCS"), _
                                          OpenMode.ForWrite)
      End If

      acUCSTblRec.Origin = New Point3d(4, 5, 3)
      acUCSTblRec.XAxis = New Vector3d(1, 0, 0)
      acUCSTblRec.YAxis = New Vector3d(0, 1, 0)

      '' 打开当前视口
      Dim acVportTblRec As ViewportTableRecord
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                        OpenMode.ForWrite)

      '' 在当前视口的原点显示 UCS 图标
      acVportTblRec.IconAtOrigin = True
      acVportTblRec.IconEnabled = True

      '' 设置 UCS 为当前坐标系
      acVportTblRec.SetUcs(acUCSTblRec.ObjectId)
```

```
acDoc.Editor.UpdateTiledViewportsFromDatabase()

'' 显示当前 UCS 坐标系的名称
Dim acUCSTblRecActive As UcsTableRecord
acUCSTblRecActive = acTrans.GetObject(acVportTblRec.UcsName, _
                                      OpenMode.ForRead)

Application.ShowAlertDialog("The current UCS is: " & _
                            acUCSTblRecActive.Name)

Dim pPtRes As PromptPointResult
Dim pPtOpts As PromptPointOptions = New PromptPointOptions("")

'' 提示选取一个点
pPtOpts.Message = vbLf & "Enter a point: "
pPtRes = acDoc.Editor.GetPoint(pPtOpts)

Dim pPt3dWCS As Point3d
Dim pPt3dUCS As Point3d

'' 将获得的点的坐标转变为当前 UCS 坐标
If pPtRes.Status = PromptStatus.OK Then
    pPt3dWCS = pPtRes.Value
    pPt3dUCS = pPtRes.Value

    '' 将该点的当前 UCS 坐标转变为 WCS 坐标
    Dim newMatrix As Matrix3d = New Matrix3d()
    newMatrix = Matrix3d.AlignCoordinateSystem(Point3d.Origin, _
                                               Vector3d.XAxis, _
                                               Vector3d.YAxis, _
                                               Vector3d.ZAxis, _
                                               acVportTblRec.Ucs.Origin, _
                                               acVportTblRec.Ucs.Xaxis, _
                                               acVportTblRec.Ucs.Yaxis, _
                                               acVportTblRec.Ucs.Zaxis)

    pPt3dWCS = pPt3dWCS.TransformBy(newMatrix)

    Application.ShowAlertDialog("The WCS coordinates are: " & vbLf & _
                                pPt3dWCS.ToString() & vbLf & _
                                "The UCS coordinates are: " & vbLf & _
                                pPt3dUCS.ToString())
End If
```

```
      '' 提交事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("NewUCS")]
public static void NewUCS()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 UCSTable
      UcsTable acUCSTbl;
      acUCSTbl = acTrans.GetObject(acCurDb.UcsTableId,
                                   OpenMode.ForRead) as UcsTable;
      UcsTableRecord acUCSTblRec;

      // 检查 UCS 表中是否有“New_UCS”这条记录
      if (acUCSTbl.Has("New_UCS") == false)
      {
          acUCSTblRec = new UcsTableRecord();
          acUCSTblRec.Name = "New_UCS";

          // 以写模式打开 UCSTable
          acUCSTbl.UpgradeOpen();

          // 往 UCSTable 添加新记录
          acUCSTbl.Add(acUCSTblRec);
          acTrans.AddNewlyCreatedDBObject(acUCSTblRec, true);
      }
      else
      {
          acUCSTblRec = acTrans.GetObject(acUCSTbl["New_UCS"],
                                          OpenMode.ForWrite) as UcsTableRecord;
```

```
    }

    acUCSTblRec.Origin = new Point3d(4, 5, 3);
    acUCSTblRec.XAxis = new Vector3d(1, 0, 0);
    acUCSTblRec.YAxis = new Vector3d(0, 1, 0);

    // 打开当前视口
    ViewportTableRecord acVportTblRec;
    acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                OpenMode.ForWrite) as ViewportTableRecord;

    // 在当前视口的原点显示 UCS 图标
    acVportTblRec.IconAtOrigin = true;
    acVportTblRec.IconEnabled = true;

    // 设置 UCS 为当前坐标系
    acVportTblRec.SetUcs(acUCSTblRec.ObjectId);
    acDoc.Editor.UpdateTiledViewportsFromDatabase();

    // 显示当前 UCS 坐标系的名称
    UcsTableRecord acUCSTblRecActive;
    acUCSTblRecActive = acTrans.GetObject(acVportTblRec.UcsName,
                                          OpenMode.ForRead) as UcsTableRecord;

    Application.ShowAlertDialog("The current UCS is: " +
                                acUCSTblRecActive.Name);

    PromptPointResult pPtRes;
    PromptPointOptions pPtOpts = new PromptPointOptions("");

    // 提示选取一个点
    pPtOpts.Message = "\nEnter a point: ";
    pPtRes = acDoc.Editor.GetPoint(pPtOpts);

    Point3d pPt3dWCS;
    Point3d pPt3dUCS;

    // 将获得的点的坐标转变为当前 UCS 坐标
    if (pPtRes.Status == PromptStatus.OK)
    {
        pPt3dWCS = pPtRes.Value;
        pPt3dUCS = pPtRes.Value;

        // 将该点的当前 UCS 坐标转变为 WCS 坐标
```

```
            Matrix3d newMatrix = new Matrix3d();
            newMatrix = Matrix3d.AlignCoordinateSystem(Point3d.Origin,
                                                       Vector3d.XAxis,
                                                       Vector3d.YAxis,
                                                       Vector3d.ZAxis,
                                                       acVportTblRec.Ucs.Origin,
                                                       acVportTblRec.Ucs.Xaxis,
                                                       acVportTblRec.Ucs.Yaxis,
                                                       acVportTblRec.Ucs.Zaxis);

            pPt3dWCS = pPt3dWCS.TransformBy(newMatrix);

            Application.ShowAlertDialog("The WCS coordinates are: \n" +
                                        pPt3dWCS.ToString() + "\n" +
                                        "The UCS coordinates are: \n" +
                                        pPt3dUCS.ToString());
        }

        // 提交事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub NewUCS()
    ' 定义所需变量
    Dim ucsObj As AcadUCS
    Dim origin(0 To 2) As Double
    Dim xAxisPnt(0 To 2) As Double
    Dim yAxisPnt(0 To 2) As Double

    ' 定义 UCS 需要的点
    origin(0) = 4: origin(1) = 5: origin(2) = 3
    xAxisPnt(0) = 5: xAxisPnt(1) = 5: xAxisPnt(2) = 3
    yAxisPnt(0) = 4: yAxisPnt(1) = 6: yAxisPnt(2) = 3

    ' 将 UCS 添加到 UserCoordinatesSystems 集合
    Set ucsObj = ThisDrawing.UserCoordinateSystems. _
                    Add(origin, xAxisPnt, yAxisPnt, "New_UCS")

    ' 显示 UCS 图标
    ThisDrawing.ActiveViewport.UCSIconAtOrigin = True
    ThisDrawing.ActiveViewport.UCSIconOn = True

    ' 将新 UCS 设置为当前 UCS
```

```
    ThisDrawing.ActiveUCS = ucsObj
    MsgBox "The current UCS is : " & ThisDrawing.ActiveUCS.Name _
           & vbCrLf & " Pick a point in the drawing."

    ' 计算点的 WCS 坐标和 UCS 坐标
    Dim WCSPnt As Variant
    Dim UCSPnt As Variant

    WCSPnt = ThisDrawing.Utility.GetPoint(, "Enter a point: ")
    UCSPnt = ThisDrawing.Utility.TranslateCoordinates _
                 (WCSPnt, acWorld, acUCS, False)

    MsgBox "The WCS coordinates are: " & WCSPnt(0) & ", " _
           & WCSPnt(1) & ", " & WCSPnt(2) & vbCrLf & _
           "The UCS coordinates are: " & UCSPnt(0) & ", " _
           & UCSPnt(1) & ", " & UCSPnt(2)
End Sub
```

5.3 坐标变换

`TransformBy()`方法可以将一个坐标系的点或位移转换成另一个坐标系的点或位移。`AlignCoordinateSystem()`方法用来指定从哪个坐标系（源坐标系）转换以及转换到哪个坐标系（目标坐标系）。需要为`AlignCoordinateSystem()`方法提供下列参数：

- 源坐标系的原点；
- 代表源坐标系的 X、Y、Z 轴的 3 个 3D 矢量；
- 目标坐标系的原点；
- 代表目标坐标系的 X、Y、Z 轴的 3 个 3D 矢量；

WCS

世界坐标系：为参照坐标系，永远不会改变，所有其他坐标系均是相对于世界坐标系来定义。不管变换到哪个坐标系，相对于 WCS 的测量值都是固定的。除非特别说明，.Net API 中所有方法和属性输入输出的点的坐标都是世界坐标系的。

UCS

用户坐标系(UCS)：为工作坐标系。用户指定一个 UCS 会使绘图工作更容易。传给 AutoCAD 命令的所有点坐标，包括从 AutoLISP 过程及外部函数返回的点的坐标，均是当前 UCS 坐标（除非用户在命令提示行输入坐标时使用了*前缀）。如果需要应用程序将 WCS 坐标、OCS 坐标或 DCS 坐标传给 AutoCAD 命令，必须先调用转换方法将其转换为 UCS 坐标，然后使用 TransformBy()方法变换表示坐标值的 Point3d 对象或 Point2d 对象。

OCS

对象坐标系（又称实体坐标系或 ECS）：Polyline2d 对象和 Polyline 对象的一些方法和属性指定的点的坐标值使用这种相对于对象的坐标系。根据对象的不同使用意图，这些点的坐标通常会被转换成 WCS 坐标、当前 UCS 坐标或当前 DCS 坐标。反之，使用 WCS 坐标、UCS 坐标或 DCS 坐标的点在通过相同的属性写入数据库之前，也必须转换成 OCS 坐标。关于使用这种坐标系的方法和属性的内容，参见《***AutoCAD .NET 托管类指南***》 。

将坐标转换为 OCS 坐标或将 OCS 坐标转换为其他坐标时，必须考虑 OCS 坐标系的法线。

DCS

显示坐标系：对象显示出来之前所要转换的坐标系。DCS 坐标系的原点就是存储在 AutoCAD 系统变量 TARGET 中的点，其 Z 轴就是观察方向。就是说，一个视口始终是它的 DCS 坐标系的一个平面视图。这些坐标可以用来确定图形的哪部分显示给用户。

PSDCS

图纸空间 DCS：该坐标系只能与模型空间视口的 DCS 坐标系进行相互转换，并且基本上只是进行 X、Y 坐标缩放的 2D 转换。因而可以用来求两个坐标系之间的缩放系数。只能将 PSDCS 转变到一个模型空间视口。

将 OCS 坐标转变为 WCS 坐标

本示例在模型空间创建一个多段线，然后用 OCS 坐标和 WCS 坐标显示多段线的第 1 个顶点。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("TranslateCoordinates")> _
Public Sub TranslateCoordinates()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
```

```
acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                OpenMode.ForWrite)

'' 创建 2D 多段线
Dim acPoly2d As Polyline2d = New Polyline2d()

'' 将新对象添加到块表记录和事务
acBlkTblRec.AppendEntity(acPoly2d)
acTrans.AddNewlyCreatedDBObject(acPoly2d, True)

'' 先将多段线添加到块表记录，然后才能给它添加顶点
Dim acPts2dPoly As Point3dCollection = New Point3dCollection()
acPts2dPoly.Add(New Point3d(1, 1, 0))
acPts2dPoly.Add(New Point3d(1, 2, 0))
acPts2dPoly.Add(New Point3d(2, 2, 0))
acPts2dPoly.Add(New Point3d(3, 2, 0))
acPts2dPoly.Add(New Point3d(4, 4, 0))

For Each acPt3d As Point3d In acPts2dPoly
    Dim acVer2d As Vertex2d = New Vertex2d(acPt3d, 0, 0, 0, 0)
    acPoly2d.AppendVertex(acVer2d)
    acTrans.AddNewlyCreatedDBObject(acVer2d, True)
Next

'' 设置 2D 多段线的法线
acPoly2d.Normal = New Vector3d(0, 1, 2)

'' 获取 2D 多段线的第 1 个坐标
Dim acPts3d As Point3dCollection = New Point3dCollection()
Dim acFirstVer As Vertex2d = Nothing
For Each acObjIdVert As ObjectId In acPoly2d
    acFirstVer = acTrans.GetObject(acObjIdVert, OpenMode.ForRead)

    acPts3d.Add(acFirstVer.Position)

    Exit For
Next

'' 获取 2D 多段线的第 1 个顶点，
'' Z 坐标值用 Elevation 属性值
Dim pFirstVer As Point3d = New Point3d(acFirstVer.Position.X, _
                                       acFirstVer.Position.Y, _
                                       acPoly2d.Elevation)
```

```
      '' OCS 到 WCS 转换
      Dim mWPlane As Matrix3d = Matrix3d.WorldToPlane(acPoly2d.Normal)
      Dim pWCSPt As Point3d = pFirstVer.TransformBy(mWPlane)

      Application.ShowAlertDialog("The first vertex has the following " & _
                                  "coordinates:" & _
                                  vbLf & "OCS: " + pFirstVer.ToString() & _
                                  vbLf & "WCS: " + pWCSPt.ToString())

      '' 提交事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("TranslateCoordinates")]
public static void TranslateCoordinates()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建 2D 多段线
      Polyline2d acPoly2d = new Polyline2d();

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPoly2d);
```

```
acTrans.AddNewlyCreatedDBObject(acPoly2d, true);

// 先将多段线添加到块表记录，然后才能给它添加顶点
Point3dCollection acPts2dPoly = new Point3dCollection();
acPts2dPoly.Add(new Point3d(1, 1, 0));
acPts2dPoly.Add(new Point3d(1, 2, 0));
acPts2dPoly.Add(new Point3d(2, 2, 0));
acPts2dPoly.Add(new Point3d(3, 2, 0));
acPts2dPoly.Add(new Point3d(4, 4, 0));

foreach (Point3d acPt3d in acPts2dPoly)
{
    Vertex2d acVer2d = new Vertex2d(acPt3d, 0, 0, 0, 0);
    acPoly2d.AppendVertex(acVer2d);
    acTrans.AddNewlyCreatedDBObject(acVer2d, true);
}

// 设置 2D 多段线的法线
acPoly2d.Normal = new Vector3d(0, 1, 2);

// 获取 2D 多段线的第 1 个坐标
Point3dCollection acPts3d = new Point3dCollection();
Vertex2d acFirstVer = null;
foreach (ObjectId acObjIdVert in acPoly2d)
{
    acFirstVer = acTrans.GetObject(acObjIdVert,
                                   OpenMode.ForRead) as Vertex2d;
    acPts3d.Add(acFirstVer.Position);

    break;
}
// 获取 2D 多段线的第 1 个顶点
// Z 坐标值用 Elevation 属性值
Point3d pFirstVer = new Point3d(acFirstVer.Position.X,
                                acFirstVer.Position.Y,
                                acPoly2d.Elevation);

// OCS 到 WCS 转换
Matrix3d mWPlane = Matrix3d.WorldToPlane(acPoly2d.Normal);
Point3d pWCSPt = pFirstVer.TransformBy(mWPlane);

Application.ShowAlertDialog("The first vertex has the following " +
                            "coordinates:" +
                            "\nOCS: " + pFirstVer.ToString() +
```

```
                                    "\nWCS: " + pWCSPt.ToString());

      // 提交事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub TranslateCoordinates()
    ' 在模型空间创建一条多段线
    Dim plineObj As AcadPolyline
    Dim points(0 To 14) As Double

    ' 定义多段线顶点
    points(0) = 1: points(1) = 1: points(2) = 0
    points(3) = 1: points(4) = 2: points(5) = 0
    points(6) = 2: points(7) = 2: points(8) = 0
    points(9) = 3: points(10) = 2: points(11) = 0
    points(12) = 4: points(13) = 4: points(14) = 0

    '在模型空间创建一条多段线
    Set plineObj = ThisDrawing.ModelSpace.AddPolyline(points)

    ' 求第 1 个顶点的坐标
    Dim firstVertex As Variant
    firstVertex = plineObj.Coordinate(0)

    ' 使用 Elevation 属性求 Z 坐标
    firstVertex(2) = plineObj.Elevation

    ' 修改多段线的法线，使两坐标系明显不一样
    Dim plineNormal(0 To 2) As Double
    plineNormal(0) = 0#
    plineNormal(1) = 1#
    plineNormal(2) = 2#
    plineObj.Normal = plineNormal

    ' 将 OCS 坐标转变为 WCS 坐标
    Dim coordinateWCS As Variant
    coordinateWCS = ThisDrawing.Utility.TranslateCoordinates _
                        (firstVertex, acOCS, acWorld, False, plineNormal)

    ' 显示点的坐标
    MsgBox "The first vertex has the following coordinates:" _
          & vbCrLf & "OCS: (" & firstVertex(0) & "," & _
```

```
            firstVertex(1) & ",," & firstVertex(2) & ")" & vbCrLf & _
            "WCS: (" & coordinateWCS(0) & "," & _
            coordinateWCS(1) & "," & coordinateWCS(2) & ")"
End Sub
```

5.4 创建 3D 对象

AutoCAD 支持 3 种类型的 3D 模型：线框 wireframe、曲面 surface 和实体 solid。每种模型均有自己的创建和编辑技巧。

5.4.1 创建线框 Wireframes

使用 AutoCAD，我们可以通过在三维空间任意位置放置任意的二维平面对象来创建线框模型。可以使用下列方法来实现在三维空间放置二维对象：

- 通过输入 3D 点来创建对象。输入定义了点的 *X*、*Y*、*Z* *位置的坐标*。
- 设置默认构建平面（XY 平面），通过定义一个 UCS 在该平面上绘制对象。
- 创建对象后，在三维空间内移动对象到适合的方向。

此外，我们可以创建一些可以存在于所有三维空间的线框对象，如线和 3D 多段线等。

更多关于创建线框的内容， 见《AutoCAD 用户指南》中的“创建线框模型”。

5.4.2 创建网格 Meshes

矩形网格（PolygonMesh 对象）使用平面镶嵌面表示一个对象的表面。网格中镶嵌面的密度或个数由 M 方向和 N 方向顶点数矩阵来定义，类似于行列构成的表格。M 和 N 分别表示给定顶点的行列位置。可以在 2D 平面和 3D 空间创建网格，不过主要还是用在 3D 空间。

创建网格的方法是，先创建 PolygonMesh 对象的一个实例，然后指定顶点的密度和位置。这个创建过程需要 6 个可选参数：要创建的多边形网格的类型；定义 M 方向和 N 方向顶点数的 2 个整数值；包含网格中所有顶点坐标的点的集合； 定义网格在 M 方向和 N 方向是否闭合的 2 个布尔值。

完成 PolygonMesh 的创建后，可以使用 `IsMClosed` 属性和 `NClosed` 属性来闭合网格。

更多关于创建网格的内容，参见《AutoCAD 用户指南》中“创建曲面”。

创建多边形网格

本示例创建一个 4×4 的多边形网格，然后调整当前视口的方向以方便观察网格的三维特性。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Create3DMesh")> _
Public Sub Create3DMesh()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建多边形网格
      Dim acPolyMesh As PolygonMesh = New PolygonMesh()
      acPolyMesh.MSize = 4
      acPolyMesh.NSize = 4

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPolyMesh)
      acTrans.AddNewlyCreatedDBObject(acPolyMesh, True)

      '' 添加顶点前，必须先将多边形网格添加到块表记录
      Dim acPts3dPMesh As Point3dCollection = New Point3dCollection()
      acPts3dPMesh.Add(New Point3d(0, 0, 0))
      acPts3dPMesh.Add(New Point3d(2, 0, 1))
      acPts3dPMesh.Add(New Point3d(4, 0, 0))
      acPts3dPMesh.Add(New Point3d(6, 0, 1))

      acPts3dPMesh.Add(New Point3d(0, 2, 0))
      acPts3dPMesh.Add(New Point3d(2, 2, 1))
      acPts3dPMesh.Add(New Point3d(4, 2, 0))
```

```
        acPts3dPMesh.Add(New Point3d(6, 2, 1))

        acPts3dPMesh.Add(New Point3d(0, 4, 0))
        acPts3dPMesh.Add(New Point3d(2, 4, 1))
        acPts3dPMesh.Add(New Point3d(4, 4, 0))
        acPts3dPMesh.Add(New Point3d(6, 4, 0))

        acPts3dPMesh.Add(New Point3d(0, 6, 0))
        acPts3dPMesh.Add(New Point3d(2, 6, 1))
        acPts3dPMesh.Add(New Point3d(4, 6, 0))
        acPts3dPMesh.Add(New Point3d(6, 6, 0))

        For Each acPt3d As Point3d In acPts3dPMesh
            Dim acPMeshVer As PolygonMeshVertex = New PolygonMeshVertex(acPt3d)
            acPolyMesh.AppendVertex(acPMeshVer)
            acTrans.AddNewlyCreatedDBObject(acPMeshVer, True)
        Next

        '' 打开当前视口
        Dim acVportTblRec As ViewportTableRecord
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                          OpenMode.ForWrite)

        '' 旋转当前视口的观察方向
        acVportTblRec.ViewDirection = New Vector3d(-1, -1, 1)
        acDoc.Editor.UpdateTiledViewportsFromDatabase()

        '' 提交事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("Create3DMesh")]
public static void Create3DMesh()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;
```

```
using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // 以读模式打开块表
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // 以写模式打开块表记录模型空间
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // 创建多边形网格
    PolygonMesh acPolyMesh = new PolygonMesh();
    acPolyMesh.MSize = 4;
    acPolyMesh.NSize = 4;

    // 将新对象添加到块表记录和事务
    acBlkTblRec.AppendEntity(acPolyMesh);
    acTrans.AddNewlyCreatedDBObject(acPolyMesh, true);

    // 添加顶点前，必须先将多边形网格添加到块表记录
    Point3dCollection acPts3dPMesh = new Point3dCollection();
    acPts3dPMesh.Add(new Point3d(0, 0, 0));
    acPts3dPMesh.Add(new Point3d(2, 0, 1));
    acPts3dPMesh.Add(new Point3d(4, 0, 0));
    acPts3dPMesh.Add(new Point3d(6, 0, 1));

    acPts3dPMesh.Add(new Point3d(0, 2, 0));
    acPts3dPMesh.Add(new Point3d(2, 2, 1));
    acPts3dPMesh.Add(new Point3d(4, 2, 0));
    acPts3dPMesh.Add(new Point3d(6, 2, 1));

    acPts3dPMesh.Add(new Point3d(0, 4, 0));
    acPts3dPMesh.Add(new Point3d(2, 4, 1));
    acPts3dPMesh.Add(new Point3d(4, 4, 0));
    acPts3dPMesh.Add(new Point3d(6, 4, 0));

    acPts3dPMesh.Add(new Point3d(0, 6, 0));
    acPts3dPMesh.Add(new Point3d(2, 6, 1));
    acPts3dPMesh.Add(new Point3d(4, 6, 0));
    acPts3dPMesh.Add(new Point3d(6, 6, 0));
```

```
        foreach (Point3d acPt3d in acPts3dPMesh)
        {
            PolygonMeshVertex acPMeshVer = new PolygonMeshVertex(acPt3d);
            acPolyMesh.AppendVertex(acPMeshVer);
            acTrans.AddNewlyCreatedDBObject(acPMeshVer, true);
        }

        // 打开当前视口
        ViewportTableRecord acVportTblRec;
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                    OpenMode.ForWrite) as ViewportTableRecord;

        // 旋转当前视口的观察方向
        acVportTblRec.ViewDirection = new Vector3d(-1, -1, 1);
        acDoc.Editor.UpdateTiledViewportsFromDatabase();

        // 提交事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub Create3DMesh()
    Dim meshObj As AcadPolygonMesh
    Dim mSize, nSize, Count As Integer

    ' 创建点矩阵
    Dim points(0 To 47) As Double
    points(0) = 0: points(1) = 0: points(2) = 0
    points(3) = 2: points(4) = 0: points(5) = 1
    points(6) = 4: points(7) = 0: points(8) = 0
    points(9) = 6: points(10) = 0: points(11) = 1
    points(12) = 0: points(13) = 2: points(14) = 0
    points(15) = 2: points(16) = 2: points(17) = 1
    points(18) = 4: points(19) = 2: points(20) = 0
    points(21) = 6: points(22) = 2: points(23) = 1
    points(24) = 0: points(25) = 4: points(26) = 0
    points(27) = 2: points(28) = 4: points(29) = 1
    points(30) = 4: points(31) = 4: points(32) = 0
    points(33) = 6: points(34) = 4: points(35) = 0
    points(36) = 0: points(37) = 6: points(38) = 0
    points(39) = 2: points(40) = 6: points(41) = 1
    points(42) = 4: points(43) = 6: points(44) = 0
    points(45) = 6: points(46) = 6: points(47) = 0
```

```
    mSize = 4: nSize = 4

    ' 在模型空间创建 3Dmesh
    Set meshObj = ThisDrawing.ModelSpace. _
                    Add3DMesh(mSize, nSize, points)

    ' 修改视口的观察方向，以便获得较好的观察效果
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport

    ZoomAll
End Sub
```

5.4.3 创建多面网格 Polyface Meshes

多面网格是指由拥有多个顶点的面定义的对象的表面。创建多面网格与创建矩形网格类似，首先创建 PolyFaceMesh 对象实例，PolyFaceMesh 对象的构造函数不需要任何参数。然后给多面网格添加顶点。添加顶点的方法是，创建 PolyFaceMeshVertex 对象，然后使用 AppendVertex()方法将 PolyFaceMeshVertex 对象添加到 PolyFaceMesh 对象。

完成多面网格的创建后，可以设置其指定边线不可见、设置图层、设置颜色等。

使多面网格的某个边线不可见的方法是，首先创建一个 FaceRecord 对象实例，使用 MakeEdgeInvisibleAt()方法设置好 FaceRecord 对象的哪个边不可见，然后使用 AppendFaceRecord()方法将 FaceRecord 对象添加到 PolyFaceMesh 对象即可。

Create a polyface mesh

本示例创建一个多面网格，然后调整当前视口的方向以方便观察网格的三维特性。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreatePolyfaceMesh")> _
Public Sub CreatePolyfaceMesh()
  '' 获取当前文档和数据库，启动事务
```

```
Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
Dim acCurDb As Database = acDoc.Database

Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
    '' 以读模式打开块表
    Dim acBlkTbl As BlockTable
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

    '' 以写模式打开块表记录模型空间
    Dim acBlkTblRec As BlockTableRecord
    acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                    OpenMode.ForWrite)

    '' 创建多面网格
    Dim acPFaceMesh As PolyFaceMesh = New PolyFaceMesh()

    '' 将新对象添加到块表记录和事务
    acBlkTblRec.AppendEntity(acPFaceMesh)
    acTrans.AddNewlyCreatedDBObject(acPFaceMesh, True)

    '' 添加顶点前，必须先将多边形网格添加到块表记录
    Dim acPts3dPFMesh As Point3dCollection = New Point3dCollection()
    acPts3dPFMesh.Add(New Point3d(4, 7, 0))
    acPts3dPFMesh.Add(New Point3d(5, 7, 0))
    acPts3dPFMesh.Add(New Point3d(6, 7, 0))
    acPts3dPFMesh.Add(New Point3d(4, 6, 0))
    acPts3dPFMesh.Add(New Point3d(5, 6, 0))
    acPts3dPFMesh.Add(New Point3d(6, 6, 1))

    For Each acPt3d As Point3d In acPts3dPFMesh
        Dim acPMeshVer As PolyFaceMeshVertex = New PolyFaceMeshVertex(acPt3d)
        acPFaceMesh.AppendVertex(acPMeshVer)
        acTrans.AddNewlyCreatedDBObject(acPMeshVer, True)
    Next

    Dim acFaceRec1 As FaceRecord = New FaceRecord(1, 2, 5, 4)
    acPFaceMesh.AppendFaceRecord(acFaceRec1)
    acTrans.AddNewlyCreatedDBObject(acFaceRec1, True)

    Dim acFaceRec2 As FaceRecord = New FaceRecord(2, 3, 6, 5)
    acPFaceMesh.AppendFaceRecord(acFaceRec2)
    acTrans.AddNewlyCreatedDBObject(acFaceRec2, True)

    '' 打开当前视口
```

```
      Dim acVportTblRec As ViewportTableRecord
      acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                        OpenMode.ForWrite)

      '' 旋转当前视口的观察方向
      acVportTblRec.ViewDirection = New Vector3d(-1, -1, 1)
      acDoc.Editor.UpdateTiledViewportsFromDatabase()

      '' 提交事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreatePolyfaceMesh")]
public static void CreatePolyfaceMesh()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建多面网格
      PolyFaceMesh acPFaceMesh = new PolyFaceMesh();

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acPFaceMesh);
      acTrans.AddNewlyCreatedDBObject(acPFaceMesh, true);
```

```
        // 添加顶点前，必须先将多边形网格添加到块表记录
        Point3dCollection acPts3dPFMesh = new Point3dCollection();
        acPts3dPFMesh.Add(new Point3d(4, 7, 0));
        acPts3dPFMesh.Add(new Point3d(5, 7, 0));
        acPts3dPFMesh.Add(new Point3d(6, 7, 0));

        acPts3dPFMesh.Add(new Point3d(4, 6, 0));
        acPts3dPFMesh.Add(new Point3d(5, 6, 0));
        acPts3dPFMesh.Add(new Point3d(6, 6, 1));

        foreach (Point3d acPt3d in acPts3dPFMesh)
        {
            PolyFaceMeshVertex acPMeshVer = new PolyFaceMeshVertex(acPt3d);
            acPFaceMesh.AppendVertex(acPMeshVer);
            acTrans.AddNewlyCreatedDBObject(acPMeshVer, true);
        }

        FaceRecord acFaceRec1 = new FaceRecord(1, 2, 5, 4);
        acPFaceMesh.AppendFaceRecord(acFaceRec1);
        acTrans.AddNewlyCreatedDBObject(acFaceRec1, true);

        FaceRecord acFaceRec2 = new FaceRecord(2, 3, 6, 5);
        acFaceRec2.MakeEdgeInvisibleAt(1); // 使第 2 个顶点开始的边线不可见
        acPFaceMesh.AppendFaceRecord(acFaceRec2);
        acTrans.AddNewlyCreatedDBObject(acFaceRec2, true);

        // 打开当前视口
        ViewportTableRecord acVportTblRec;
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                          OpenMode.ForWrite) as ViewportTableRecord;

        // 旋转当前视口的观察方向
        acVportTblRec.ViewDirection = new Vector3d(-1, -1, 1);
        acDoc.Editor.UpdateTiledViewportsFromDatabase();

        // 提交事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreatePolyfaceMesh()
    '定义网格顶点
    Dim vertex(0 To 17) As Double
```

```
    vertex(0) = 4: vertex(1) = 7: vertex(2) = 0
    vertex(3) = 5: vertex(4) = 7: vertex(5) = 0
    vertex(6) = 6: vertex(7) = 7: vertex(8) = 0
    vertex(9) = 4: vertex(10) = 6: vertex(11) = 0
    vertex(12) = 5: vertex(13) = 6: vertex(14) = 0
    vertex(15) = 6: vertex(16) = 6: vertex(17) = 1

    ’定义面列表
    Dim FaceList(0 To 7) As Integer
    FaceList(0) = 1
    FaceList(1) = 2
    FaceList(2) = 5
    FaceList(3) = 4
    FaceList(4) = 2
    FaceList(5) = 3
    FaceList(6) = 6
    FaceList(7) = 5

    ’创建多面网格
    Dim polyfaceMeshObj As AcadPolyfaceMesh
    Set polyfaceMeshObj = ThisDrawing.ModelSpace. _
                              AddPolyfaceMesh(vertex, FaceList)

    ’修改视口的观察方向，以便获得较好的观察效果
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport

    ZoomAll
End Sub
```

5.4.4 创建实体 Solids

实体对象（Solid3d 对象）表示一个对象的整个体积。实体是最能完整表达信息、最不模棱两可的 3D 建模类型。而且立体形状虽然复杂，却比线框和网格更容易构建和编辑。

我们可以使用 Solid3d 对象的成员方法和属性创建各种基本实体形状，比如箱体、球体、楔形以及其它形状的实体。我们还可以沿某一路径拉伸面域对象，或绕坐标轴旋转一个 2D 对象。

像网格一样，在对一个实体进行消隐、着色、渲染操作之前，实体以线框的形式显示。另外，我们还可以分析实体的质量属性（体积、惯性矩、重心，等等）。使用下列 MassProperties 属性，我们可以查询与 Solid3d 对象相关联的 Solid3dMassProperties 对象。Solid3dMassProperties 对象包含下列可以用来分析实体的属性：MomentOfInertia（惯性矩）、PrincipalAxess（主轴）、PrincipalMoments（主力矩）、ProductOfInertia（惯性积）、RadiiOfGyration（回转半径）和 Volume（体积）。

实体的显示受当前视觉样式及 3D 建模相关系统变量的影响。影响实体显示的系统变量比如 ISOLINES 和 FACETRES，ISOLINES 控制用于可视化线框的弯曲部分的镶嵌线的数目，而 FACETRES 用于调整阴影和隐藏线对象的平滑度。

更多关于创建实体的内容，参见《AutoCAD 用户指南》中的“创建 3D 对象”。

创建一个楔形实体

下面的示例创建一个楔形实体，然后调整当前视口的方向以方便观察网格的三维特性。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateWedge")> _
Public Sub CreateWedge()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建 3D 楔形实体 wedge
      Dim acSol3D As Solid3d = New Solid3d()
      acSol3D.CreateWedge(10, 15, 20)
```

```
        '' 3D 实体的中心点放在(5,5,0)
        acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                  Point3d.Origin))

        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3D)
        acTrans.AddNewlyCreatedDBObject(acSol3D, True)

        '' 打开当前视口
        Dim acVportTblRec As ViewportTableRecord
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId, _
                                          OpenMode.ForWrite)

        '' 旋转当前视口的观察方向
        acVportTblRec.ViewDirection = New Vector3d(-1, -1, 1)
        acDoc.Editor.UpdateTiledViewportsFromDatabase()

        '' 提交事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("CreateWedge")]
public static void CreateWedge()
{
    // 获取当前文档和数据库，启动事务
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;

    using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
    {
        // 以读模式打开块表
        BlockTable acBlkTbl;
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                     OpenMode.ForRead) as BlockTable;

        // 以写模式打开块表记录模型空间
        BlockTableRecord acBlkTblRec;
```

```
        acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                        OpenMode.ForWrite) as BlockTableRecord;

        //创建 3D 楔形实体 wedge
        Solid3d acSol3D = new Solid3d();
        acSol3D.CreateWedge(10, 15, 20);

        // 3D 实体的中心点放在(5,5,0)
        acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                  Point3d.Origin));

        // 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3D);
        acTrans.AddNewlyCreatedDBObject(acSol3D, true);

        // 打开当前视口
        ViewportTableRecord acVportTblRec;
        acVportTblRec = acTrans.GetObject(acDoc.Editor.ActiveViewportId,
                                    OpenMode.ForWrite) as ViewportTableRecord;

        // 旋转当前视口的观察方向
        acVportTblRec.ViewDirection = new Vector3d(-1, -1, 1);
        acDoc.Editor.UpdateTiledViewportsFromDatabase();

        // 提交事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateWedge()
    Dim wedgeObj As Acad3DSolid
    Dim center(0 To 2) As Double
    Dim length As Double
    Dim width As Double
    Dim height As Double

    ' 定义楔形
    center(0) = 5#: center(1) = 5#: center(2) = 0
    length = 10#: width = 15#: height = 20#

    ' 在模型空间创建楔形
    Set wedgeObj = ThisDrawing.ModelSpace. _
                       AddWedge(center, length, width, height)
```

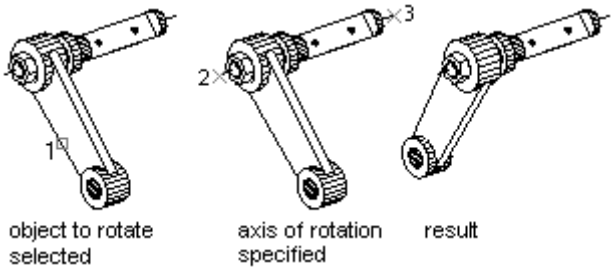
```
    ’修改视口的观察方向
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport

    ZoomAll
End Sub
```

5.5 编辑 3D 对象

本节讲解如何通过旋转、阵列、镜像等方式编辑 3D 对象。

5.5.1 在 3D 空间旋转对象

使用对象的 TransformBy()方法和矩阵的 Rotation()方法，我们可以绕一个给定点在 2D 空间旋转对象，2D 对象的旋转方向是绕 Z 轴旋转。对于 3D 对象，旋转轴不限于 Z 轴。当使用 Rotation()方法而不是绕 Z 轴旋转时，必须为 Rotation()方法指定一个 3D 矢量作为旋转轴。



① 选择要旋转的对象；

② 指定旋转轴；

③ 旋转后的结果

创建一个 3D 箱体并绕轴旋转

本示例先创建一个 3D 箱体，然后定义一个旋转轴，最后将箱体绕该轴旋转 30 度。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("Rotate_3DBox")> _
Public Sub Rotate_3DBox()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建一个 3D 实体箱子
      Dim acSol3D As Solid3d = New Solid3d()
      acSol3D.CreateBox(5, 7, 10)

      '' 3D 实体的中心点放在(5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                Point3d.Origin))

      Dim curUCSMatrix As Matrix3d = acDoc.Editor.CurrentUserCoordinateSystem
      Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d

      '' 将 3D 箱体绕点(-3,4,0)和点(-3,-4,0)定义的轴旋转 30 度
      Dim vRot As Vector3d = New Point3d(-3, 4, 0). _
                                     GetVectorTo(New Point3d(-3, -4, 0))

      acSol3D.TransformBy(Matrix3d.Rotation(0.5236, vRot, _
                                            New Point3d(-3, 4, 0)))

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3D)
      acTrans.AddNewlyCreatedDBObject(acSol3D, True)
```

```
      '' 提交事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("Rotate_3DBox")]
public static void Rotate_3DBox()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建一个 3D 实体箱子
      Solid3d acSol3D = new Solid3d();
      acSol3D.CreateBox(5, 7, 10);

      // 3D 实体的中心点放在(5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                Point3d.Origin));

      Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
      CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;

      // 将 3D 箱体绕点(-3,4,0)和点(-3,-4,0)定义的轴旋转 30 度
      Vector3d vRot = new Point3d(-3, 4, 0).
                                   GetVectorTo(new Point3d(-3, -4, 0));
```

```
      acSol3D.TransformBy(Matrix3d.Rotation(0.5236,
                                            vRot,
                                            new Point3d(-3, 4, 0)));

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3D);
      acTrans.AddNewlyCreatedDBObject(acSol3D, true);

      // 提交事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub Rotate_3DBox()
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double

    ' 定义箱体
    center(0) = 5: center(1) = 5: center(2) = 0
    length = 5
    width = 7
    height = 10

    '在模型空间创建一个 3D 箱体
    Set boxObj = ThisDrawing.ModelSpace. _
                    AddBox(center, length, width, height)

    ' 用 2 个点定义旋转轴
    Dim rotatePt1(0 To 2) As Double
    Dim rotatePt2(0 To 2) As Double
    Dim rotateAngle As Double
    rotatePt1(0) = -3: rotatePt1(1) = 4: rotatePt1(2) = 0
    rotatePt2(0) = -3: rotatePt2(1) = -4: rotatePt2(2) = 0
    rotateAngle = 30
    rotateAngle = rotateAngle * 3.141592 / 180#

    ' 旋转箱体
    boxObj.Rotate3D rotatePt1, rotatePt2, rotateAngle

    ZoomAll
```

```
End Sub
```

5.5.2 在 3D 空间阵列对象

使用对象的 TransformBy()方法和 Clone()方法，我们可以创建一个 3D 矩形阵列。除了像创建 2D 矩形阵列那样指定列数（*X* 方向）和行数（*Y* 方向）外，还需要指定层数（*Z* 方向）。

创建 3D 矩形阵列

本示例先创建一个圆，然后使用这个圆创建一个 4×4×3 的矩形阵列。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

Public Shared Function PolarPoints(ByVal pPt As Point2d, _
                                   ByVal dAng As Double, _
                                   ByVal dDist As Double)

    Return New Point2d(pPt.X + dDist * Math.Cos(dAng), _
                       pPt.Y + dDist * Math.Sin(dAng))
End Function

<CommandMethod("CreateRectangular3DArray")> _
Public Sub CreateRectangular3DArray()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForRead)

      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;
```

```
// 以写模式打开块表记录模型空间
BlockTableRecord acBlkTblRec;
acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                OpenMode.ForWrite) as BlockTableRecord;

'' 创建圆，圆心(2,2,0)，半径 0.5
Dim acCirc As Circle = New Circle()
acCirc.Center = New Point3d(2, 2, 0)
acCirc.Radius = 0.5

'' 将新对象添加到块表记录和事务
acBlkTblRec.AppendEntity(acCirc)
acTrans.AddNewlyCreatedDBObject(acCirc, True)

'' 创建 4 行 4 列 3 层的矩形阵列
Dim nRows As Integer = 4
Dim nColumns As Integer = 4
Dim nLevels As Integer = 3

'' 设置行、列、层 3 个方向的偏移量及阵列基角
Dim dRowOffset As Double = 1
Dim dColumnOffset As Double = 1
Dim dLevelsOffset As Double = 4
Dim dArrayAng As Double = 0

'' 获取当前 UCS 坐标系 X 轴的角度
Dim curUCSMatrix As Matrix3d = acDoc.Editor.CurrentUserCoordinateSystem
Dim curUCS As CoordinateSystem3d = curUCSMatrix.CoordinateSystem3d
Dim acVec2dAng As Vector2d = New Vector2d(curUCS.Xaxis.X, _
                                          curUCS.Xaxis.Y)

'' 如果 UCS 被旋转了，相应地调整阵列角度
dArrayAng = dArrayAng + acVec2dAng.Angle

'' 使用对象界限的左上角作为阵列基点
Dim acExts As Extents3d = acCirc.Bounds.GetValueOrDefault()
Dim acPt2dArrayBase As Point2d = New Point2d(acExts.MinPoint.X, _
                                             acExts.MaxPoint.Y)

'' 跟踪为每列创建的对象
Dim acDBObjCollCols As DBObjectCollection = New DBObjectCollection()
acDBObjCollCols.Add(acCirc)

'' 创建首列对象
```

```
    Dim nColumnsCount As Integer = 1
    While (nColumns > nColumnsCount)
        Dim acEntClone As Entity = acCirc.Clone()
        acDBObjCollCols.Add(acEntClone)

        '' 给复制的对象计算新位置
        Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                              dArrayAng, _
                                              dColumnOffset * nColumnsCount)

        Dim acVec2d As Vector2d = acPt2dArrayBase.GetVectorTo(acPt2dTo)
        Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
        acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

        acBlkTblRec.AppendEntity(acEntClone)
        acTrans.AddNewlyCreatedDBObject(acEntClone, True)

        nColumnsCount = nColumnsCount + 1
    End While

    '' 90 度的弧度值
    Dim dAng As Double = 1.5708

    '' 跟踪每行和列中创建的对象
    Dim acDBObjCollLvls As DBObjectCollection = New DBObjectCollection()

    For Each acObj As DBObject In acDBObjCollCols
        acDBObjCollLvls.Add(acObj)
    Next

    '' 创建每行对象
    For Each acEnt As Entity In acDBObjCollCols
        Dim nRowsCount As Integer = 1

        While (nRows > nRowsCount)
            Dim acEntClone As Entity = acEnt.Clone()
            acDBObjCollLvls.Add(acEntClone)

            '' 给复制的对象计算新位置
            Dim acPt2dTo As Point2d = PolarPoints(acPt2dArrayBase, _
                                                  dArrayAng + dAng, _
                                                  dRowOffset * nRowsCount)

            Dim acVec2d As Vector2d = acPt2dArrayBase.GetVectorTo(acPt2dTo)
```

```
                Dim acVec3d As Vector3d = New Vector3d(acVec2d.X, acVec2d.Y, 0)
                acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

                acBlkTblRec.AppendEntity(acEntClone)
                acTrans.AddNewlyCreatedDBObject(acEntClone, True)

                nRowsCount = nRowsCount + 1
            End While
        Next

        '' 创建 3D 阵列的层
        For Each acEnt As Entity In acDBObjCollLvls
            Dim nLvlsCount As Integer = 1

            While (nLevels > nLvlsCount)
                Dim acEntClone As Entity = acEnt.Clone()

                Dim acVec3d As Vector3d = New Vector3d(0, 0, dLevelsOffset *
nLvlsCount)
                acEntClone.TransformBy(Matrix3d.Displacement(acVec3d))

                acBlkTblRec.AppendEntity(acEntClone)
                acTrans.AddNewlyCreatedDBObject(acEntClone, True)

                nLvlsCount = nLvlsCount + 1
            End While
        Next

        '' 提交事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

static Point2d PolarPoints(Point2d pPt, double dAng, double dDist)
{
    return new Point2d(pPt.X + dDist * Math.Cos(dAng),
 pPt.Y + dDist * Math.Sin(dAng));
}
```

```
[CommandMethod("CreateRectangular3DArray")]
public static void CreateRectangular3DArray()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建圆，圆心(2,2,0)，半径 0.5
      Circle acCirc = new Circle();
      acCirc.Center = new Point3d(2, 2, 0);
      acCirc.Radius = 0.5;

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acCirc);
      acTrans.AddNewlyCreatedDBObject(acCirc, true);

      // 创建 4 行 4 列 3 层的矩形阵列
      int nRows = 4;
      int nColumns = 4;
      int nLevels = 3;

      // 设置行、列、层 3 个方向的偏移量及阵列基角
      double dRowOffset = 1;
      double dColumnOffset = 1;
      double dLevelsOffset = 4;
      double dArrayAng = 0;

      // 获取当前 UCS 坐标系 X 轴的角度
      Matrix3d curUCSMatrix = acDoc.Editor.CurrentUserCoordinateSystem;
      CoordinateSystem3d curUCS = curUCSMatrix.CoordinateSystem3d;
      Vector2d acVec2dAng = new Vector2d(curUCS.Xaxis.X,
```

```
                                        curUCS.Xaxis.Y);

// 如果 UCS 被旋转了，相应地调整阵列角度
dArrayAng = dArrayAng + acVec2dAng.Angle;

// 使用对象界限的左上角作为阵列基点
Extents3d acExts = acCirc.Bounds.GetValueOrDefault();
Point2d acPt2dArrayBase = new Point2d(acExts.MinPoint.X,
                                      acExts.MaxPoint.Y);

// 跟踪为每列创建的对象
DBObjectCollection acDBObjCollCols = new DBObjectCollection();
acDBObjCollCols.Add(acCirc);

// 创建首列对象
int nColumnsCount = 1;
while (nColumns > nColumnsCount)
{
    Entity acEntClone = acCirc.Clone() as Entity;
    acDBObjCollCols.Add(acEntClone);

    // 给复制的对象计算新位置
    Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                   dArrayAng,
                                   dColumnOffset * nColumnsCount);

    Vector2d acVec2d = acPt2dArrayBase.GetVectorTo(acPt2dTo);
    Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
    acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

    acBlkTblRec.AppendEntity(acEntClone);
    acTrans.AddNewlyCreatedDBObject(acEntClone, true);

    nColumnsCount = nColumnsCount + 1;
}

// 90 度的弧度值
double dAng = 1.5708;

// 跟踪每行和列中创建的对象
DBObjectCollection acDBObjCollLvls = new DBObjectCollection();

foreach (DBObject acObj in acDBObjCollCols)
{
```

```
        acDBObjCollLvls.Add(acObj);
    }

    // 创建每行对象
    foreach (Entity acEnt in acDBObjCollCols)
    {
        int nRowsCount = 1;

        while (nRows > nRowsCount)
        {
            Entity acEntClone = acEnt.Clone() as Entity;
            acDBObjCollLvls.Add(acEntClone);

            // 给复制的对象计算新位置
            Point2d acPt2dTo = PolarPoints(acPt2dArrayBase,
                                           dArrayAng + dAng,
                                           dRowOffset * nRowsCount);

            Vector2d acVec2d = acPt2dArrayBase.GetVectorTo(acPt2dTo);
            Vector3d acVec3d = new Vector3d(acVec2d.X, acVec2d.Y, 0);
            acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

            acBlkTblRec.AppendEntity(acEntClone);
            acTrans.AddNewlyCreatedDBObject(acEntClone, true);

            nRowsCount = nRowsCount + 1;
        }
    }

    // 创建 3D 阵列的层
    foreach (Entity acEnt in acDBObjCollLvls)
    {
        int nLvlsCount = 1;

        while (nLevels > nLvlsCount)
        {
            Entity acEntClone = acEnt.Clone() as Entity;

            Vector3d acVec3d = new Vector3d(0, 0, dLevelsOffset * nLvlsCount);
            acEntClone.TransformBy(Matrix3d.Displacement(acVec3d));

            acBlkTblRec.AppendEntity(acEntClone);
            acTrans.AddNewlyCreatedDBObject(acEntClone, true);
```

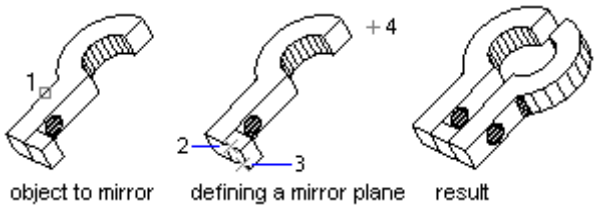
```
                nLvlsCount = nLvlsCount + 1;
            }
        }
        // 提交事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub CreateRectangular3DArray()
    ' 创建圆
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 0.5
    Set circleObj = ThisDrawing.ModelSpace. _
                        AddCircle(center, radius)
    ' 定义矩形阵列
    Dim numberOfRows As Long
    Dim numberOfColumns As Long
    Dim numberOfLevels As Long
    Dim distanceBwtnRows As Double
    Dim distanceBwtnColumns As Double
    Dim distanceBwtnLevels As Double
    numberOfRows = 4
    numberOfColumns = 4
    numberOfLevels = 3
    distanceBwtnRows = 1
    distanceBwtnColumns = 1
    distanceBwtnLevels = 4
    ' 创建阵列对象
    Dim retObj As Variant
    retObj = circleObj.ArrayRectangular _
                (numberOfRows, numberOfColumns, _
                 numberOfLevels, distanceBwtnRows, _
                 distanceBwtnColumns, distanceBwtnLevels)

    ZoomAll
End Sub
```

5.5.3 在 3D 空间沿平面镜像对象

使用对象的 TransformBy()方法和矩阵的 Mirroring ()方法，我们可以沿 3 个点定义的镜像平面来镜像对象。



① 选择要镜像的对象；

② 定义镜像平面；

③ 镜像的结果

在 3D 空间镜像对象

本示例先在模型空间创建一个箱体，然后沿一个平面镜像该箱体，并将镜像后得到的箱体涂成红色。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("MirrorABox3D")> _
Public Sub MirrorABox3D()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)
```

```
      '' 创建 3D 箱体
      Dim acSol3D As Solid3d = New Solid3d()
      acSol3D.CreateBox(5, 7, 10)

      '' 3D 实体的中心点放在(5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                Point3d.Origin))

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3D)
      acTrans.AddNewlyCreatedDBObject(acSol3D, True)

      '' 创建原 3D 箱体的拷贝并修改颜色
      Dim acSol3DCopy As Solid3d = acSol3D.Clone()
      acSol3DCopy.ColorIndex = 1

      '' 定义镜像平面
      Dim acPlane As Plane = New Plane(New Point3d(1.25, 0, 0), _
                                       New Point3d(1.25, 2, 0), _
                                       New Point3d(1.25, 2, 2))

      '' 沿平面镜像 3D 实体
      acSol3DCopy.TransformBy(Matrix3d.Mirroring(acPlane))

      '' 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3DCopy)
      acTrans.AddNewlyCreatedDBObject(acSol3DCopy, True)

      '' 提交事务
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("MirrorABox3D")]
public static void MirrorABox3D()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
Database acCurDb = acDoc.Database;

using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
{
    // 以读模式打开块表
    BlockTable acBlkTbl;
    acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                 OpenMode.ForRead) as BlockTable;

    // 以写模式打开块表记录模型空间
    BlockTableRecord acBlkTblRec;
    acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                    OpenMode.ForWrite) as BlockTableRecord;

    // 创建 3D 箱体
    Solid3d acSol3D = new Solid3d();
    acSol3D.CreateBox(5, 7, 10);

    // 3D 实体的中心点放在(5,5,0)
    acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                              Point3d.Origin));

    // 将新对象添加到块表记录和事务
    acBlkTblRec.AppendEntity(acSol3D);
    acTrans.AddNewlyCreatedDBObject(acSol3D, true);

    // 创建原 3D 箱体的拷贝并修改颜色
    Solid3d acSol3DCopy = acSol3D.Clone() as Solid3d;
    acSol3DCopy.ColorIndex = 1;

    // 定义镜像平面
    Plane acPlane = new Plane(new Point3d(1.25, 0, 0),
                              new Point3d(1.25, 2, 0),
                              new Point3d(1.25, 2, 2));

    // 沿平面镜像 3D 实体
    acSol3DCopy.TransformBy(Matrix3d.Mirroring(acPlane));

    // 将新对象添加到块表记录和事务
    acBlkTblRec.AppendEntity(acSol3DCopy);
    acTrans.AddNewlyCreatedDBObject(acSol3DCopy, true);

    // 提交事务
    acTrans.Commit();
```

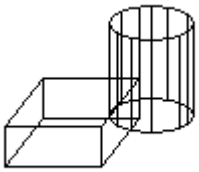
```
  }
}
```

VBA/ActiveX 代码参考

```
Sub MirrorABox3D()
    ' 创建箱体对象
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double
    center(0) = 5#: center(1) = 5#: center(2) = 0
    length = 5#: width = 7: height = 10#

    ' 在模型空间创建箱体对象
    Set boxObj = ThisDrawing.ModelSpace. _
                    AddBox(center, length, width, height)
    ' 用 3 个点定义镜像平面
    Dim mirrorPt1(0 To 2) As Double
    Dim mirrorPt2(0 To 2) As Double
    Dim mirrorPt3(0 To 2) As Double
    mirrorPt1(0) = 1.25: mirrorPt1(1) = 0: mirrorPt1(2) = 0
    mirrorPt2(0) = 1.25: mirrorPt2(1) = 2: mirrorPt2(2) = 0
    mirrorPt3(0) = 1.25: mirrorPt3(1) = 2: mirrorPt3(2) = 2
    ' 镜像箱体
    Dim mirrorBoxObj As Acad3DSolid
    Set mirrorBoxObj = boxObj.Mirror3D(mirrorPt1, mirrorPt2, mirrorPt3)
    mirrorBoxObj.Color = acRed

    ZoomAll
End Sub
```

5.6 编辑 3D 实体

掌握了创建实体的技能，接下来我们就可以通过实体与实体的加加减减来创建更加复杂的形状。我们可以让一个实体加入另一个实体（求并集），从一个实体中减去另一个实体（求差集），或者求出几个实体的公共体积（重叠部分）（求交集）。`BooleanOperation()`方法用来执行这些组合，`CheckInterference()`方法用来确定两个实体是否有重叠部分。



我们还可以通过获得实体的2D横截面或将一个实体切割成两部分等方法对实体作进一步的修改。GetSection()方法用于求实体的横截面，Slice()方法将实体切割成两部分。

查找两实体间的干涉（interference）

本示例先创建一个箱体和一个圆柱体，然后找到两个实体间的干涉（重叠部分）并使用重叠部分创建一个新实体。为便于观察，箱体设置成白色，圆柱体设置成青色，干涉体设置成红色。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("FindInterferenceBetweenSolids")> _
Public Sub FindInterferenceBetweenSolids()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建 3D 箱体
      Dim acSol3DBox As Solid3d = New Solid3d()
      acSol3DBox.CreateBox(5, 7, 10)
      acSol3DBox.ColorIndex = 7

      '' 3D 实体的中心点放在(5,5,0)
```

```
        acSol3DBox.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                     Point3d.Origin))

        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3DBox)
        acTrans.AddNewlyCreatedDBObject(acSol3DBox, True)

        '' 创建 3D 圆柱体
        '' 默认构造函数的中心点为(0,0,0),这样就不用再移动了
        Dim acSol3DCyl As Solid3d = New Solid3d()
        acSol3DCyl.CreateFrustum(20, 5, 5, 5)
        acSol3DCyl.ColorIndex = 4

        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3DCyl)
        acTrans.AddNewlyCreatedDBObject(acSol3DCyl, True)

        '' 用箱体和圆柱体的干涉创建一个 3D 实体
        Dim acSol3DCopy As Solid3d = acSol3DCyl.Clone()

        '' 检查箱体和圆柱体是否有重叠部分
        If acSol3DCopy.CheckInterference(acSol3DBox) = True Then
            acSol3DCopy.BooleanOperation(BooleanOperationType.BoolIntersect, _
                                         acSol3DBox.Clone())

            acSol3DCopy.ColorIndex = 1
        End If

        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3DCopy)
        acTrans.AddNewlyCreatedDBObject(acSol3DCopy, True)

        '' 提交事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("FindInterferenceBetweenSolids")]
```

```
public static void FindInterferenceBetweenSolids()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建 3D 箱体
      Solid3d acSol3DBox = new Solid3d();
      acSol3DBox.CreateBox(5, 7, 10);
      acSol3DBox.ColorIndex = 7;

      // 3D 实体的中心点放在(5,5,0)
      acSol3DBox.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                   Point3d.Origin));

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3DBox);
      acTrans.AddNewlyCreatedDBObject(acSol3DBox, true);

      // 创建 3D 圆柱体
      // 默认构造函数的中心点为(0,0,0),这样就不用再移动了
      Solid3d acSol3DCyl = new Solid3d();
      acSol3DCyl.CreateFrustum(20, 5, 5, 5);
      acSol3DCyl.ColorIndex = 4;

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3DCyl);
      acTrans.AddNewlyCreatedDBObject(acSol3DCyl, true);

      // 用箱体和圆柱体的干涉创建一个 3D 实体
      Solid3d acSol3DCopy = acSol3DCyl.Clone() as Solid3d;
```

```
        // 检查箱体和圆柱体是否有重叠部分
        if (acSol3DCopy.CheckInterference(acSol3DBox) == true)
        {
            acSol3DCopy.BooleanOperation(BooleanOperationType.BoolIntersect,
                                         acSol3DBox.Clone() as Solid3d);
            acSol3DCopy.ColorIndex = 1;
        }

        // 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3DCopy);
        acTrans.AddNewlyCreatedDBObject(acSol3DCopy, true);

        // 提交事务
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub FindInterferenceBetweenSolids()
    ' 定义箱体
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double
    center(0) = 5: center(1) = 5: center(2) = 0
    length = 5
    width = 7
    height = 10

    ' 在模型空间创建箱体对象
    ' 颜色设成白色
    Set boxObj = ThisDrawing.ModelSpace.AddBox(center, length, width, height)
    boxObj.Color = acWhite

    ' 定义圆柱体
    Dim cylinderObj As Acad3DSolid
    Dim cylinderRadius As Double
    Dim cylinderHeight As Double
    center(0) = 0: center(1) = 0: center(2) = 0
    cylinderRadius = 5
    cylinderHeight = 20

    ' 创建圆柱体并设置成青色
```

```
    Set cylinderObj = ThisDrawing.ModelSpace. _
                          AddCylinder(center, cylinderRadius, cylinderHeight)
    cylinderObj.Color = acCyan

    ' 求两实体的干涉并用它创建一个新实体，颜色设置成红色
    Dim solidObj As Acad3DSolid
    Set solidObj = boxObj.CheckInterference(cylinderObj, True)
    solidObj.Color = acRed

    ZoomAll
End Sub
```

将一个实体切割成两个实体

本示例先在模型空间创建一个箱体，然后用 3 点定义的平面切割箱体，切割操作返回一个 3D 实体对象。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("SliceABox")> _
Public Sub SliceABox()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录模型空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建 3D 箱体
      Dim acSol3D As Solid3d = New Solid3d()
      acSol3D.CreateBox(5, 7, 10)
```

```
        acSol3D.ColorIndex = 7

        '' 3D 实体的中心点放在(5,5,0)
        acSol3D.TransformBy(Matrix3d.Displacement(New Point3d(5, 5, 0) - _
                                                  Point3d.Origin))

        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3D)
        acTrans.AddNewlyCreatedDBObject(acSol3D, True)

        '' 定义镜像平面
        Dim acPlane As Plane = New Plane(New Point3d(1.5, 7.5, 0), _
                                 New Point3d(1.5, 7.5, 10), _
                                 New Point3d(8.5, 2.5, 10))

        Dim acSol3DSlice As Solid3d = acSol3D.Slice(acPlane, True)
        acSol3DSlice.ColorIndex = 1

        '' 将新对象添加到块表记录和事务
        acBlkTblRec.AppendEntity(acSol3DSlice)
        acTrans.AddNewlyCreatedDBObject(acSol3DSlice, True)

        '' 提交事务
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

[CommandMethod("SliceABox")]
public static void SliceABox()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
```

```
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录模型空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建 3D 箱体
      Solid3d acSol3D = new Solid3d();
      acSol3D.CreateBox(5, 7, 10);
      acSol3D.ColorIndex = 7;

      // 3D 实体的中心点放在(5,5,0)
      acSol3D.TransformBy(Matrix3d.Displacement(new Point3d(5, 5, 0) -
                                                Point3d.Origin));

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3D);
      acTrans.AddNewlyCreatedDBObject(acSol3D, true);

      // 定义镜像平面
      Plane acPlane = new Plane(new Point3d(1.5, 7.5, 0),
                                new Point3d(1.5, 7.5, 10),
                                new Point3d(8.5, 2.5, 10));

      Solid3d acSol3DSlice = acSol3D.Slice(acPlane, true);
      acSol3DSlice.ColorIndex = 1;

      // 将新对象添加到块表记录和事务
      acBlkTblRec.AppendEntity(acSol3DSlice);
      acTrans.AddNewlyCreatedDBObject(acSol3DSlice, true);

      // 提交事务
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub SliceABox()
    ' 创建箱体对象
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
```

```
    Dim height As Double
    Dim center(0 To 2) As Double
    center(0) = 5#: center(1) = 5#: center(2) = 0
    length = 5#: width = 7: height = 10#

    ' 在模型空间创建箱体对象
    Set boxObj = ThisDrawing.ModelSpace. _
                    AddBox(center, length, width, height)
    boxObj.Color = acWhite

    ' 用 3 个点定义截面
    Dim slicePt1(0 To 2) As Double
    Dim slicePt2(0 To 2) As Double
    Dim slicePt3(0 To 2) As Double

    slicePt1(0) = 1.5: slicePt1(1) = 7.5: slicePt1(2) = 0
    slicePt2(0) = 1.5: slicePt2(1) = 7.5: slicePt2(2) = 10
    slicePt3(0) = 8.5: slicePt3(1) = 2.5: slicePt3(2) = 10

    ' 切割箱体并将新实体设置成红色
    Dim sliceObj As Acad3DSolid
    Set sliceObj = boxObj.SliceSolid _
                        (slicePt1, slicePt2, slicePt3, True)
    sliceObj.Color = acRed

    ZoomAll
End Sub
```

第 6 章 定义布局和打印

使用 AutoCAD 创建了一个图形后，通常我们会将图形打印出来或发布出去。打印出来的图像可能包含图形的单个视图，或者几个视图的稍复杂点儿的排列。我们可以在图纸空间创建称之为“浮动视口”的窗口，来显示图形的各种视图。可以根据需要打印一个或多个视口，或者设置选项来确定打印什么、图像如何适合最终的输出格式等。

本章主要内容：

- 模型空间和图纸空间
- 布局
- 视口
- 打印图形

6.1 模型空间和图纸空间

模型空间是为你的设计模型创建几何图形的绘图环境。通常情况下，当你开始在模型空间中绘图时，你会设定绘图环境的图形界限，并且会使用现实世界中的单位来绘制。

图纸空间代表你的模型打印出来时在图纸上的表现。在图纸空间，你可以编排图形的不同视图、按比例单独缩放某一视图、按照你的打印要求排列各个视图。根据需要，你的图形可以用多个不同的图纸空间表示。

6.2 布局

图形的全部几何形状都包含在布局里。模型空间的几何形状均包含在一个叫“模型”的单一布局内。不能更改模型空间布局的名称，也不能创建另一个模型空间布局，因为一个图形（对应一个图形数据库）只能有一个模型空间布局。

图纸空间的几何形状也都包含在布局内。一个图形可以有许多不同的图纸空间布局，每一个布局代表一个不同的打印配置。可以修改图纸空间布局的名称。

块表 BlockTable 中的“*MODEL_SPACE”块表记录（BlockTableRecord）包含模型空间布局 Model 中的全部几何图形。由于图形中可以有许多个图纸空间布局，因此，块表 BlockTable 中的“*PAPER_SPACE”块表记录（BlockTableRecord）指向的是最近的那个活动布局。

更多关于使用图纸空间布局的内容，参见《AutoCAD 用户指南》中的“创建多视图图形布局（图纸空间）”。

6.2.1 布局和块

布局的内容由两个不同的对象构成：Layout 对象和 BlockTableRecord 对象。Layout 对象包含打印设置和布局在 AutoCAD 用户界面中的视觉属性。BlockTableRecord 对象包含显示在布局中的几何图形，比如注释、浮动视口、标题栏等。BlockTableRecord 对象还包括 Viewport 对象，可以控制用于布局的绘图辅助属性和图层属性的显示。

每个 Layout 对象都与一个，且只与一个 BlockTableRecord 对象关联。若要访问与指定布局关联的 BlockTableRecord 对象，使用 Layout 对象的 `BlockTableRecordId` 属性。

反过来，每个 BlockTableRecord 对象都与一个，且只与一个 Layout 对象关联。若要访问与指定块表记录 BlockTableRecord 关联的 Layout 对象，使用该块表记录的 LayoutId 属性。BlockTableRecord 对象的 `IsLayout` 属性可以用来确定其是否与 `Layout` 对象关联，如果 `IsLayout` 属性为 `TRUE`，则表示 BlockTableRecord 对象关联了一个 Layout 对象。

6.2.2 打印设置

PlotSettings 对象与 Layout 对象类似，两者都包含相同的打印信息，因为 Layout 类是从 PlotSettings 类派生的。两者的主要区别是， Layout 对象关联了一个包含要打印的几何图形的 BlockTableRecord 对象，而 PlotSettings 对象不与特定的 BlockTableRecord 对象关联，它只是打印设置的一个简单的命名集合，这些打印设置任何几何图形都可以使用。PlotSettings 对象在 AutoCAD 用户界面中称之为页面设置，可以从页面设置管理器访问。

6.2.3 布局设置

布局设置控制着打印或发布图形时最终的输出结果。这些设置会影响到纸张大小、打印比例、打印区域、打印原点，以及打印设备的名称等。了解如何使用布局设置可以确保按我们的期望打印布局。与输出布局相关的大多数设置都是只读的，修改布局的打印设置需要使用 PlotSettings 对象和 PlotSettingsValidator 对象（见 § 6.2.3.7 的示例）。

6.2.3.1 图纸大小和单位

纸张大小的选择依赖于所配置的绘图仪或输出设备。每个绘图仪或输出设备都有一个可用的输出尺寸的标准列表。为布局分配的输出尺寸可以使用 CanonicalMediaName 只读属性来查询。

还可以使用 PlotPaperUnits 属性查询布局的单位，该属性返回枚举 PlotPaperUnit 定义的三个值中的一个：Inches（英寸）、Millimeters（毫米）、Pixels（像素）。如果你的绘图仪配置为光栅输出，返回的输出单位是 Pixels（像素）。

6.2.3.2 打印原点

打印原点就是指定打印区域的左下角，可以使用 PlotOrigin 属性查询。通常情况下，打印原点设为(0, 0)。不过，可以将打印居中在纸张上。PlotCentered 属性返回当前是否为居中打印，如果 PlotCentered 属性为 TRUE 则为居中打印。居中打印改变了打印原点。

6.2.3.3 打印区域

打印布局时，打印区域由 PlotType 属性确定。PlotType 属性中保存的值是 PlotType 枚举定义的枚举值中的一个。PlotType 枚举定义了下列值：

```
public enum PlotType {
  Display,
  Extents,
  Limits,
  View,
  Window,
  Layout
}
```

Display 显示

打印当前模型空间显示的所有图形。当从图纸空间布局打印时此选项不可用。

Extents 范围

打印所有落在当前选定的工作空间的边界内的图形。

Limits 图形界限

打印当前空间图形界限内的所有图形。

View 视图

打印 PlotViewName 属性命名的视图。

Window 窗口

打印 PlotWindowArea 属性指定的窗口内的所有图形。

Layout 布局

打印落在指定纸张尺寸的边界内的所有图形。当从模型空间打印时此选项不可用。

当新创建一个图纸空间布局时，默认选项为 Layout 布局。

6.2.3.4 打印比例

一般来说，我们使用实际尺寸绘制对象。打印图形时，要么指定一个精确的比例，要么让图形适合输出尺寸。我们可以指定一个标准打印比例，也可以指定一个自定义打印比例。

当 UseStandardScale 设置为 TRUE 时使用标准比例。打印的实际比例值可以用 StdScale 属性查询到。

当 UseStandardScale 设置为 FALSE 时使用自定义比例。打印的实际比例值可以用 CustomPrintScale 属性查询到。

在审查早期的视图草稿时，精确的比例并不总是很重要。我们可以将 StdScaleType 属性设置为 StdScaleType 枚举定义的 ScaleToFit 值， 使布局的打印结果最大可能适合输出的大小。

6.2.3.5 线宽比例

我们可以使用布局的打印比例按比例对线宽进行缩放。通常情况下，线宽指定要打印对象的线的宽度，并且按照该线宽尺寸打印对象而不考虑打印比例。大多数情况下，我们使用默认打印比例来 1:1 打印一个布局。不过有的时候，比如想要将适合 A#图纸的布局打印到 E#图纸布局上，这时可以让线宽按新的打印比例进行缩放。

ScaleLineweights 属性（布尔类型）返回线宽是否按打印比例缩放，返回 TRUE 表示打印布局时线宽按比例缩放。

6.2.3.6 打印设备

打印设备名存储在 PlotConfigurationName 属性里，该设备名应和你的系统里的一个设备相匹配，否则使用默认设备。

6.2.3.7 示例：查询和设置布局

下面的示例演示如何查询和修改当前布局的打印设备。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.PlottingServices

<CommandMethod("ChangePlotSetting")> _
Public Sub ChangePlotSetting()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 引用布局管理器 LayoutManager
      Dim acLayoutMgr As LayoutManager
      acLayoutMgr = LayoutManager.Current

      '' 读取当前布局，在命令行窗口显示布局名字
      Dim acLayout As Layout
      acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
                                    OpenMode.ForRead)

      '' 输出当前布局名和设备名
      acDoc.Editor.WriteMessage(vbLf & "Current layout: " & _
                                acLayout.LayoutName)

      acDoc.Editor.WriteMessage(vbLf & "Current device name: " & _
                                acLayout.PlotConfigurationName)

      '' 从布局中获取 PlotInfo
      Dim acPlInfo As PlotInfo = New PlotInfo()
      acPlInfo.Layout = acLayout.ObjectId

      '' 复制布局中的 PlotSettings
      Dim acPlSet As PlotSettings = New PlotSettings(acLayout.ModelType)
      acPlSet.CopyFrom(acLayout)

      '' 更新 PlotSettings 对象的 PlotConfigurationName 属性
      Dim acPlSetVdr As PlotSettingsValidator = PlotSettingsValidator.Current
      acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3", _
                                          "ANSI_A_(8.50_x_11.00_Inches)")
```

```
        '' 更新布局
        acLayout.UpgradeOpen()
        acLayout.CopyFrom(acPlSet)

        '' 输出已更新的布局设备名
        acDoc.Editor.WriteMessage(vbLf & "New device name: " & _
                                  acLayout.PlotConfigurationName)

        '' 将新对象保存到数据库
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.PlottingServices;

[CommandMethod("ChangePlotSetting")]
public static void ChangePlotSetting()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 引用布局管理器 LayoutManager
      LayoutManager acLayoutMgr;
      acLayoutMgr = LayoutManager.Current;

      // 读取当前布局，在命令行窗口显示布局名
      Layout acLayout;
      acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout),
                                    OpenMode.ForRead) as Layout;

      // 输出当前布局名和设备名
      acDoc.Editor.WriteMessage("\nCurrent layout: " +
                                acLayout.LayoutName);

      acDoc.Editor.WriteMessage("\nCurrent device name: " +
```

```
                                acLayout.PlotConfigurationName);

      // 从布局中获取 PlotInfo
      PlotInfo acPlInfo = new PlotInfo();
      acPlInfo.Layout = acLayout.ObjectId;

      // 复制布局中的 PlotSettings
      PlotSettings acPlSet = new PlotSettings(acLayout.ModelType);
      acPlSet.CopyFrom(acLayout);

      // 更新 PlotSettings 对象的 PlotConfigurationName 属性
      PlotSettingsValidator acPlSetVdr = PlotSettingsValidator.Current;
      acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3",
                                          "ANSI_A_(8.50_x_11.00_Inches)");

      // 更新布局
      acLayout.UpgradeOpen();
      acLayout.CopyFrom(acPlSet);

      // 输出已更新的布局设备名
      acDoc.Editor.WriteMessage("\nNew device name: " +
                                acLayout.PlotConfigurationName);

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

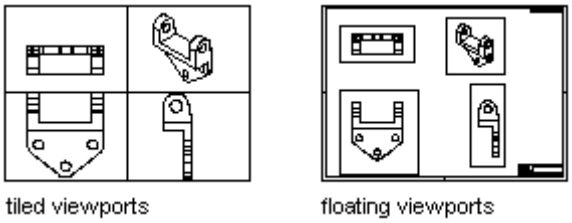
```
Public Sub ChangePlotSetting()
    Dim layoutObj As AcadLayout
    Set layoutObj = ThisDrawing.ActiveLayout

    ' 输出当前布局名和设备名
    ThisDrawing.Utility.Prompt vbLf & "Current layout: " & layoutObj.Name
    ThisDrawing.Utility.Prompt vbLf & "Current device name: " & _
                               layoutObj.ConfigName

    ' 修改当前布局的输出设备名
    layoutObj.RefreshPlotDeviceInfo
    layoutObj.ConfigName = "DWF6 ePlot.pc3"
    layoutObj.CanonicalMediaName = "ANSI_A_(8.50_x_11.00_Inches)"

    ' 输出已更新的布局设备名
    ThisDrawing.Utility.Prompt vbLf & "Current device name: " & _
```

```
 layoutObj.ConfigName & vbLf
End Sub
```

6.3 视口

当工作在模型空间时，我们在 ViewportTableRecord 对象所代表的平铺视口中绘制几何图形。我们一次可以显示一个或多个视口，显示多个平铺视口时，在一个视口编辑修改会影响到其他所有的视口。不过，我们可以单独为每个视口设置放大倍数、观察点、网格和对象捕捉等。

而工作在图纸空间时，我们使用 Viewport 对象所代表的浮动视口进行布局调整，而且可以拥有模型图的多个不同的视图。可以像对象那样对浮动视口进行移动、改变大小、改变形状等操作，以创建一个合适的布局。还可以直接在图纸空间视图上绘制标题栏、注释等对象，而不会影响到模型空间的模型图。



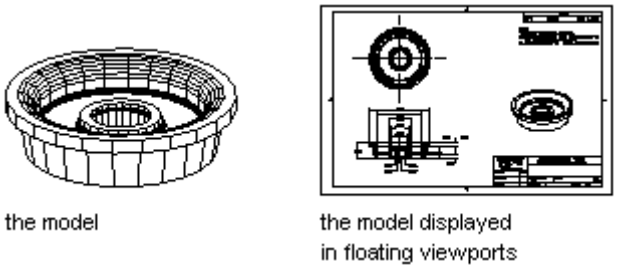
平铺视口　　浮动视口

6.3.1 浮动视口

我们不能在图纸空间编辑模型图。要想在一个 Viewport 对象中访问模型图，应使用 Editor 对象的 `SwitchToModelSpace()`和 `SwitchToPaperSpace()`成员函数从图纸空间切换到模型空间。这样，我们就可以在保持整个布局可见的同时编辑修改模型图了。在 Viewport 对象中编辑和浏览修改结果的能力与在 ViewportTableRecord 对象中几乎是一样的。

但是，对单个视图可以拥有更多的控制。比如，可以冻结或解冻视口中的图层而不影响其他视口，可以切换视口中几何体的显示开或关，还可以在视口之间对齐视图及相对于整个布局缩放视图等。

下图演示了一个模型的不同视图是如何在图纸空间显示的。每个图纸空间图像代表不同视图的一个 Viewport 对象。其中一个视图的尺寸标注图层被冻结了。注意图中的标题栏、边框、注释文字是绘制在图纸空间的，不会出现在模型空间视图上。还有，包含视口边界的图层也被冻结了。



左侧为模型，右侧为图纸空间显示（3 个浮动视口分别显示了模型的 3 个视图）

可以在 Paper 空间使用 Viewport 对象，也可以在 Model 空间使用 Viewport 对象。可以通过检查系统变量 TILEMODE 和 CVPORT 的当前值来确定是否工作在 Model 空间。如何判断见下表：如果 TILEMODE 为 0 而 CVPORT 不等于 2，则工作在 Paper 空间；如果 TILEMODE 为 0 而 CVPORT 等于 2，则工作在 Model 空间；如果 TILEMODE 为 1，则工作在 Model 布局的 Model 空间。

当前空间		
TILEMODE	**CVPORT**	**状态**
0	不等于 2	Model 之外的布局是活动的，且你正工作在图纸空间。
0	2	Model 之外的布局是活动的，且你正工作在浮动视口。
1	任意值	Model 布局是活动的。

注：将布局中的视口切换到模型空间前，布局中应至少有一个视口的 On 属性为 TRUE。

在图纸空间时，AutoCAD 会在图像区域的左下角显示图纸空间用户坐标系（UCS）的图标。出现十字光标表示可以在图纸空间布局区进行编辑操作。

在模型空间和图纸空间之间切换

本示例演示如何在模型空间和图纸空间之间切换。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("ToggleSpace")> _
Public Sub ToggleSpace()
  '' 获取当前文档
```

```
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  '' 获取系统变量 CVPORT 和 TILEMODE 的当前值
  Dim nCvports As Integer = Application.GetSystemVariable("CVPORT")
  Dim nTilemode As Integer = Application.GetSystemVariable("TILEMODE")

  '' 检查 Model 布局是否为活动的（TILEMODE=1 为活动）
  If nTilemode = 0 Then
      '' 检查 Model 空间是否为活动的（CVPORT=2 为活动）
      If nCvports = 2 Then
          acDoc.Editor.SwitchToPaperSpace()
      Else
          acDoc.Editor.SwitchToModelSpace()
      End If
  Else
      '' 切换到 Paper 空间布局
      Application.SetSystemVariable("TILEMODE", 0)
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("ToggleSpace")]
public static void ToggleSpace()
{
  // 获取当前文档
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  // 获取系统变量 CVPORT 和 TILEMODE 的当前值
  object oCvports = Application.GetSystemVariable("CVPORT");
  object oTilemode = Application.GetSystemVariable("TILEMODE");

  // 检查 Model 布局是否为活动的（TILEMODE=1 为活动）
  if (System.Convert.ToInt16(oTilemode) == 0)
  {
      // 检查 Model 空间是否为活动的（CVPORT=2 为活动）
      // CVPORT is 2 if Model space is active
      if (System.Convert.ToInt16(oCvports) == 2)
      {
          acDoc.Editor.SwitchToPaperSpace();
      }
```

```
        else
        {
            acDoc.Editor.SwitchToModelSpace();
        }
    }
    else
    {
        // 切换到 Paper 空间布局
        Application.SetSystemVariable("TILEMODE", 0);
    }
}
```

VBA/ActiveX 代码参考

```
Public Sub ToggleSpace()
    ' 检查 Model 布局是否为活动的
    If ThisDrawing.ActiveLayout.Name <> "Model" Then
        ' 检查 Model 空间局是否为活动的
        If ThisDrawing.MSpace = True Then
            ThisDrawing.MSpace = False
        Else
            ThisDrawing.MSpace = True
        End If
    Else
        ' 切换到 Paper 空间布局
        ThisDrawing.ActiveSpace = acPaperSpace
    End If
End Sub
```

6.3.2 创建图纸空间视口

图纸空间视口的创建方法是，先创建 Viewport 对象实例，然后将其添加到非 Model 布局使用的块表引用。使用 Viewport 对象的构造函数创建新视口对象时不需要任何参数。创建了 Viewport 对象后，可以使用它的 `CenterPoint` 属性、`Width` 属性和 `Height` 属性来定义视图的位置。

创建了 Viewport 对象后，还可以设置视图本身的属性，比如观察方向（`ViewDirection` 属性）、透视图的镜头焦距（`LensLength` 属性）及是否显示网格（`GridOn` 属性）。还可以控制视口本身的属性，比如图层（`Layer` 属性）、线型（`Linetype` 属性）及线型比例（`LinetypeScale` 属性）等。

注： Viewport 对象在被附加到数据库时，其大多数属性会被重置为默认值。因此，应该先将新 Viewport 对象添加到数据库中，然后再更改其属性值。（见示例里的编码顺序）

创建并激活浮动视口

本示例设置图纸空间为活动空间，创建一个浮动视口，定义视口的视图，并激活该视口。

VB.NET

```
Imports System.Runtime.InteropServices

Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

'' 导入 ObjectARX 全局函数 acedSetCurrentVPort，
'' 对 2014 版，该函数在 accore.dll 内。
<DllImport("acad.exe", CallingConvention:=CallingConvention.Cdecl, _

EntryPoint:="?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z"
)> _
Public Shared Function acedSetCurrentVPort(ByVal AcDbVport As IntPtr) As IntPtr
End Function

<CommandMethod("CreateFloatingViewport")> _
Public Sub CreateFloatingViewport()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 Paper 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.PaperSpace), _
                                      OpenMode.ForWrite)

      '' 切换到 Paper 空间布局
      Application.SetSystemVariable("TILEMODE", 0)
      acDoc.Editor.SwitchToPaperSpace()

      '' 创建一个视口
      Dim acVport As Viewport = New Viewport()
      acVport.CenterPoint = New Point3d(3.25, 3, 0)
      acVport.Width = 6
```

```
        acVport.Height = 5

        '' 添加新对象到块表记录和事务
        acBlkTblRec.AppendEntity(acVport)
        acTrans.AddNewlyCreatedDBObject(acVport, True)

        '' 修改观察方向
        acVport.ViewDirection = New Vector3d(1, 1, 1)

        '' 激活视口
        acVport.On = True

        '' 激活视口的模型空间
        acDoc.Editor.SwitchToModelSpace()

        '' 使用导入的 ObjectARX 函数设置新视口为当前视口
        acedSetCurrentVPort(acVport.UnmanagedObject)

        '' 将新对象保存到数据库
        acTrans.Commit()
    End Using
End Sub
```

C#

```
using System.Runtime.InteropServices;
using System;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

// 导入 ObjectARX 全局函数 acedSetCurrentVPort，
// 对 2014 版，该函数在 accore.dll 内。
[DllImport("acad.exe", CallingConvention = CallingConvention.Cdecl,
 EntryPoint =
"?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z")]
extern static private int acedSetCurrentVPort(IntPtr AcDbVport);

[CommandMethod("CreateFloatingViewport")]
public static void CreateFloatingViewport()
{
    // 获取当前文档和数据库，启动事务
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;
```

```
  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录 Paper 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.PaperSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 切换到 Paper 空间布局
      Application.SetSystemVariable("TILEMODE", 0);
      acDoc.Editor.SwitchToPaperSpace();

      // 创建一个视口
      Viewport acVport = new Viewport();
      acVport.CenterPoint = new Point3d(3.25, 3, 0);
      acVport.Width = 6;
      acVport.Height = 5;

      // 添加新对象到块表记录和事务
      acBlkTblRec.AppendEntity(acVport);
      acTrans.AddNewlyCreatedDBObject(acVport, true);

      // 修改观察方向
      acVport.ViewDirection = new Vector3d(1, 1, 1);

      // 激活视口
      acVport.On = true;

      // 激活视口的模型空间
      acDoc.Editor.SwitchToModelSpace();

      // 使用导入的 ObjectARX 函数设置新视口为当前视口
      acedSetCurrentVPort(acVport.UnmanagedObject);

      // 将新对象保存到数据库
      acTrans.Commit();
  }
}
```

VBA/ActiveX 代码参考

```
Sub CreateFloatingViewport()
    ' 将图纸空间设置为活动空间
    ThisDrawing.ActiveSpace = acPaperSpace

    ' 创建图纸空间视口
    Dim newVport As AcadPViewport
    Dim center(0 To 2) As Double
    center(0) = 3.25
    center(1) = 3
    center(2) = 0
    Set newVport = ThisDrawing.PaperSpace.AddPViewport(center, 6, 5)

    ' 修改视口的观察方向
    Dim viewDir(0 To 2) As Double
    viewDir(0) = 1
    viewDir(1) = 1
    viewDir(2) = 1
    newVport.direction = viewDir

    ' 激活视口
    newVport.Display True

    ' 切换到模型空间
    ThisDrawing.MSpace = True

    ' 将 newVport 设置为当前视口
    ThisDrawing.ActivePViewport = newVport
End Sub
```

注： 要想设置或修改视图的属性（观察方向、镜头焦距，等），Viewport 对象的 On 属性必须设置为 FALSE;在将一个视口设置为当前视口前,该视口的 On 属性必须设置为 TRUE。

创建 4 个浮动视口

本例和上一个示例类似，这次创建 4 个浮动视口，并分别将 4 个视图设置为顶视图、前视图、右视图和等轴测视图。每个视口的比例设置为 1:2。为了确保在这些视口中有东西看，你可以在尝试运行本示例代码之前在模型空间创建一个 3D 实心球体或其他 3D 实体。

VB.NET

```
Imports System.Runtime.InteropServices

Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

'' 导入 ObjectARX 全局函数 acedSetCurrentVPort，
'' 对 2014 版，该函数在 accore.dll 内。
<DllImport("acad.exe", CallingConvention:=CallingConvention.Cdecl, _

EntryPoint:="?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z"
)> _
Public Shared Function acedSetCurrentVPort(ByVal AcDbVport As IntPtr) As IntPtr
End Function

<CommandMethod("FourFloatingViewports")> _
Public Sub FourFloatingViewports()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开块表记录 Paper 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.PaperSpace), _
                                      OpenMode.ForWrite)

      '' 切换到 Paper 空间布局
      Application.SetSystemVariable("TILEMODE", 0)
      acDoc.Editor.SwitchToPaperSpace()

      Dim acPt3dCol As Point3dCollection = New Point3dCollection()
      acPt3dCol.Add(New Point3d(2.5, 5.5, 0))
      acPt3dCol.Add(New Point3d(2.5, 2.5, 0))
      acPt3dCol.Add(New Point3d(5.5, 5.5, 0))
      acPt3dCol.Add(New Point3d(5.5, 2.5, 0))
```

```
    Dim acVec3dCol As Vector3dCollection = New Vector3dCollection()
    acVec3dCol.Add(New Vector3d(0, 0, 1))
    acVec3dCol.Add(New Vector3d(0, 1, 0))
    acVec3dCol.Add(New Vector3d(1, 0, 0))
    acVec3dCol.Add(New Vector3d(1, 1, 1))

    Dim dWidth As Double = 2.5
    Dim dHeight As Double = 2.5

    Dim acVportLast As Viewport = Nothing
    Dim nCnt As Integer = 0

    For Each acPt3d As Point3d In acPt3dCol
        Dim acVport As Viewport = New Viewport()
        acVport.CenterPoint = acPt3d
        acVport.Width = dWidth
        acVport.Height = dHeight

        '' 添加新对象到块表记录和事务
        acBlkTblRec.AppendEntity(acVport)
        acTrans.AddNewlyCreatedDBObject(acVport, True)

        '' 修改观察方向
        acVport.ViewDirection = acVec3dCol(nCnt)

        '' 激活视口
        acVport.On = True

        '' 记录创建的最后视口
        acVportLast = acVport

        '' 计数器加 1
        nCnt = nCnt + 1
    Next

    If acVportLast <> Nothing Then
        '' 激活视口的模型空间
        acDoc.Editor.SwitchToModelSpace()

        '' 使用导入的 ObjectARX 函数设置新视口为当前视口
        acedSetCurrentVPort(acVportLast.UnmanagedObject)
    End If

    '' 将新对象保存到数据库
```

```
      acTrans.Commit()
  End Using
End Sub
```

C#

```
using System.Runtime.InteropServices;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

// 导入 ObjectARX 全局函数 acedSetCurrentVPort，
// 对 2014 版，该函数在 accore.dll 内。
[DllImport("acad.exe", CallingConvention = CallingConvention.Cdecl,
 EntryPoint =
"?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PBVAcDbViewport@@@Z")]
extern static private int acedSetCurrentVPort(IntPtr AcDbVport);

[CommandMethod("FourFloatingViewports")]
public static void FourFloatingViewports()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开块表
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开块表记录 Paper 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.PaperSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 切换到 Paper 空间布局
      Application.SetSystemVariable("TILEMODE", 0);
      acDoc.Editor.SwitchToPaperSpace();

      Point3dCollection acPt3dCol = new Point3dCollection();
      acPt3dCol.Add(new Point3d(2.5, 5.5, 0));
```

```
acPt3dCol.Add(new Point3d(2.5, 2.5, 0));
acPt3dCol.Add(new Point3d(5.5, 5.5, 0));
acPt3dCol.Add(new Point3d(5.5, 2.5, 0));

Vector3dCollection acVec3dCol = new Vector3dCollection();
acVec3dCol.Add(new Vector3d(0, 0, 1));
acVec3dCol.Add(new Vector3d(0, 1, 0));
acVec3dCol.Add(new Vector3d(1, 0, 0));
acVec3dCol.Add(new Vector3d(1, 1, 1));

double dWidth = 2.5;
double dHeight = 2.5;

Viewport acVportLast = null;
int nCnt = 0;
foreach (Point3d acPt3d in acPt3dCol)
{
    Viewport acVport = new Viewport();
    acVport.CenterPoint = acPt3d;
    acVport.Width = dWidth;
    acVport.Height = dHeight;

    // 添加新对象到块表记录和事务
    acBlkTblRec.AppendEntity(acVport);
    acTrans.AddNewlyCreatedDBObject(acVport, true);

    // 修改观察方向
    acVport.ViewDirection = acVec3dCol[nCnt];

    // 激活视口
    acVport.On = true;

    // 记录创建的最后视口
    acVportLast = acVport;

    // 计数器加 1
    nCnt = nCnt + 1;
}
if (acVportLast != null)
{
    // 激活视口的模型空间
    acDoc.Editor.SwitchToModelSpace();

    // 使用导入的 ObjectARX 函数设置新视口为当前视口
```

```
            acedSetCurrentVPort(acVportLast.UnmanagedObject);
        }

        // 将新对象保存到数据库
        acTrans.Commit();
    }
}
```

VBA/ActiveX 代码参考

```
Sub FourFloatingViewports()
    Dim topVport, frontVport As AcadPViewport
    Dim rightVport, isoVport As AcadPViewport
    Dim pt(0 To 2) As Double
    Dim viewDir(0 To 2) As Double
    ThisDrawing.ActiveSpace = acPaperSpace
    ThisDrawing.MSpace = True

    ' 获取现有 PViewport 并设置为 topView
    pt(0) = 2.5: pt(1) = 5.5: pt(2) = 0
    Set topVport = ThisDrawing.ActivePViewport
    ' No need to set Direction for top view
    topVport.center = pt
    topVport.width = 2.5
    topVport.height = 2.5
    topVport.Display True
    topVport.StandardScale = acVp1_2

    ' 创建并设置 frontVport
    pt(0) = 2.5: pt(1) = 2.5: pt(2) = 0
    Set frontVport = ThisDrawing.PaperSpace.AddPViewport(pt, 2.5, 2.5)
    viewDir(0) = 0: viewDir(1) = 1: viewDir(2) = 0
    frontVport.direction = viewDir
    frontVport.Display acOn
    frontVport.StandardScale = acVp1_2

    ' 创建并设置 rightVport
    pt(0) = 5.5: pt(1) = 5.5: pt(2) = 0
    Set rightVport = ThisDrawing.PaperSpace.AddPViewport(pt, 2.5, 2.5)
    viewDir(0) = 1: viewDir(1) = 0: viewDir(2) = 0
    rightVport.direction = viewDir
    rightVport.Display acOn
    rightVport.StandardScale = acVp1_2

    ' 创建并设置 isoVport
```

```
    pt(0) = 5.5: pt(1) = 2.5: pt(2) = 0
    Set isoVport = ThisDrawing.PaperSpace.AddPViewport(pt, 2.5, 2.5)
    viewDir(0) = 1: viewDir(1) = 1: viewDir(2) = 1
    isoVport.direction = viewDir
    isoVport.Display acOn
    isoVport.StandardScale = acVp1_2
    ThisDrawing.MSpace = True
    ThisDrawing.ActivePViewport = isoVport

    ’完成：对所有视口执行重生成
    ThisDrawing.Regen True
End Sub
```

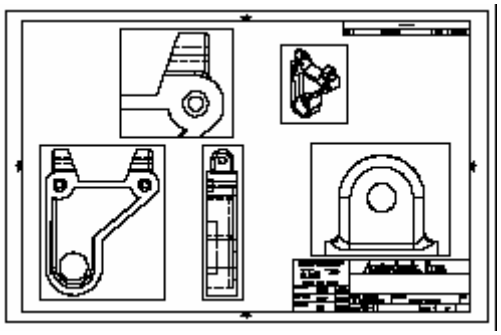
6.3.3 修改视口视图和内容

可以通过 Viewport 对象的成员属性来调整和修改视口内视图的显示：

- **ViewCenter 属性** - 表示视口内视图的观察中心。
- **ViewDirection 属性** – 表示视口内视图的目标点到相机位置的矢量。
- **ViewHeight 属性** – 表示视口内模型空间视图的高度。
- **ViewTarget 属性** – 表示视口内视图的目标点的位置。

6.3.4 相对于图纸空间缩放视图

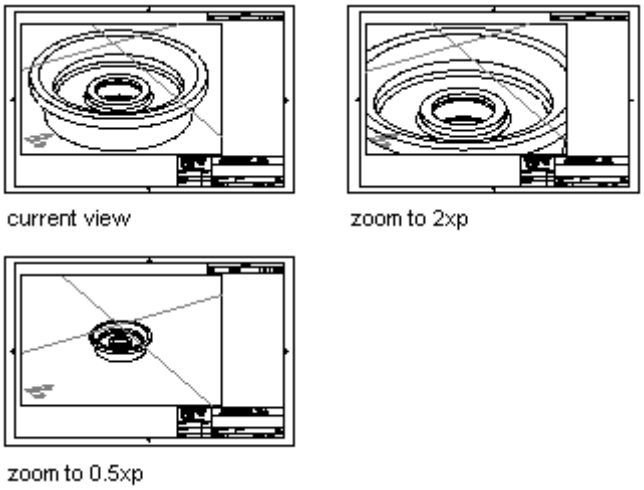
打印前可以给每个视口内的视图设置比例系数，可以相对图纸空间缩放视图，使显示的每个视图的比例一致。例如，下图显示了一个拥有多个视口的图纸空间布局，每个视口的比例和视图都不一样。



工作在图纸空间时，比例系数代表打印出来的图形的尺寸与视口中显示的模型的实际尺寸的比率。计算这个比例的方法是，用图纸空间单位除以模型空间单位。例如，一个 1/4 比例的图形，就是指定比例系数为 1 个图纸空间单位对 4 个模型空间单位（1:4）。

可以使用Viewport对象的StandardScale属性和CustomScale属性来指定视口的比例。StandardScale 属性的取值为 StandardScaleType 枚举定义的值；而 CustomScale 属性的取值为一个表示模型空间与图纸空间单位之间比率的实际数值，例如，1:4 的实际数值为 0.25。

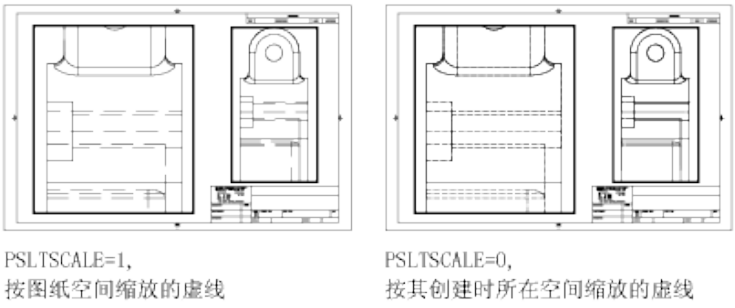
下图演示了一个模型以不同比例显示的视图效果：第 1 个的比例为 1:1；右侧第 2 个的比例为 2:1，模型的视图增大到图纸空间单位的 2 倍；第 3 个的比例为 1:2，显示的模型为实际尺寸的一半。



6.3.5 在图纸空间缩放线型图案

在图纸空间，可以使用两种方法缩放任意类型的线型。线型缩放比例可以基于创建对象所在空间（Model 或 Paper）的绘图单位，也可以是基于图纸空间单位的一个统一比例。 可以使用系统变量 PSLTSCALE 为不同视口中以不同比例显示的对象保持相同的线型比例，该系统变量也影响 3D 视图中的线的显示。

下图显示了图纸空间中按统一比例缩放的线型图案。注意两个视口中的线型的比例是一样的，尽管每个视口的比例不一样。



使用 SetSystemVariable()方法设置系统变量 PSLTSCALE 的值。

6.3.6 使用着色视口

如果图形中包含 3D 面、网格、拉伸对象、曲面或实体，可以使用 ShadePlotTyle 枚举定义的值来打印图纸空间视口。着色视口和渲染视口在预览、打印及打印到文件时都是完全着色和渲染的。

ShadePlotTyle 枚举定义的值如下：

```
public enum ShadePlotType {
  AsDisplayed,     // 显示什么样儿就打印什么样儿
  Wireframe,       // 打印线框，不管显示
  Hidden,          // 打印隐藏的，不管显示
  Rendered,        // 打印渲染的，不管显示
  VisualStyle,     // 使用 ShadePlotId 引用的视觉样式设置打印
  RenderPreset     // 使用 ShadePlotId 引用的选项设置打印
}
```

如何打印着色视口（Shaded Viewport）使用 Viewport 对象的 ShadePlot 属性来设置。

注：可以使用 Viewport 对象的 HiddenLinesRemoved 属性来设置是否显示对象的隐藏线，该属性为布尔类型：设置为 TRUE 表示删除隐藏线；设置为 FALSE 表示绘制隐藏线。

6.4 打印出图

可以将图形像在模型空间看到的那样打印出来，或者打印一个已准备好的图纸空间布局。当想要在创建图纸空间布局之前查看或检查图形时，通常先选择从模型空间打印。模型都检查没问题了，就可以准备和打印图纸空间布局了。

注：BACKGROUNDPLOT 系统变量用来控制使用前台打印还是后台打印，BACKGROUNDPLOT 设置为 0 表示前台打印（直接打印到打印机）。

打印输出图纸涉及到使用多个不同的对象：PlotFactory、PlotEngine、PlotInfo、PlotSettings、PlotSettingsValidator、PlotInfoValidator 和 PlotPageInfo 对象等。PlotEngine 对象负责基于 PlotInfo 对象提供的信息生成一个打印输出。

PlotEngine 对象用来生成布局的输出。使用 PlotEngine 对象，可以做到：

- 打印到文件
- 打印到绘图仪或打印机
- 显示布局的打印预览

6.4.1 从模型空间打印

通常情况下，当绘制很大的图形如建筑平面图时，可以指定一个比例将实际绘图单位转换为打印后的英寸或毫米。不过，从模型空间打印时，如果不特别设置的话，会使用默认设置，这些默认设置包括打印到系统打印机、打印当前的显示范围、打印比例为布满图纸、不旋转、0 偏移等。

先使用 PlotSettings 对象来修改要打印的布局的打印设置，接着使用 PlotSettingsValidator 对象对打印设置进行检验， 然后将 PlotSettings 对象传给 PlotInfo 对象。定义好 PlotInfo 对象后，就可以使用 PlotEngine 对象创建打印输出，并将包含要打印的图纸或布局信息的 PlotInfo 对象传递给 PlotEngine 对象。

打印模型布局的范围

本例在生成最后的打印输出之前设置了多个打印设置项，最后生成的输出是一个名为 Myplot 的 DWF 文件。记住在运行本示例前，应先在模型空间画点儿什么。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.PlottingServices

<CommandMethod("PlotCurrentLayout")> _
Public Sub PlotCurrentLayout()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 引用布局管理器 LayoutManager
      Dim acLayoutMgr As LayoutManager
      acLayoutMgr = LayoutManager.Current
      '' 获取当前布局，在命令行窗口显示布局名字
      Dim acLayout As Layout
      acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
                                    OpenMode.ForRead)
      '' 从布局中获取 PlotInfo
      Dim acPlInfo As PlotInfo = New PlotInfo()
      acPlInfo.Layout = acLayout.ObjectId
      '' 复制布局中的 PlotSettings
      Dim acPlSet As PlotSettings = New PlotSettings(acLayout.ModelType)
      acPlSet.CopyFrom(acLayout)
```

```
      '' 更新 PlotSettings 对象
      Dim acPlSetVdr As PlotSettingsValidator = PlotSettingsValidator.Current
      '' 设置打印区域
      acPlSetVdr.SetPlotType(acPlSet, _
                             Autodesk.AutoCAD.DatabaseServices.PlotType.Extents)
      '' 设置打印比例
      acPlSetVdr.SetUseStandardScale(acPlSet, True)
      acPlSetVdr.SetStdScaleType(acPlSet, StdScaleType.ScaleToFit)
      '' 居中打印
      acPlSetVdr.SetPlotCentered(acPlSet, True)

      '' 设置使用的打印设备
      acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3", _
                                          "ANSI_A_(8.50_x_11.00_Inches)")
      '' 用上述设置信息覆盖 PlotInfo 对象，
      '' 不会将修改保存回布局
      acPlInfo.OverrideSettings = acPlSet
      '' 验证打印信息
      Dim acPlInfoVdr As PlotInfoValidator = New PlotInfoValidator()
      acPlInfoVdr.MediaMatchingPolicy = MatchingPolicy.MatchEnabled
      acPlInfoVdr.Validate(acPlInfo)

      '' 检查是否有正在处理的打印任务
      If PlotFactory.ProcessPlotState = Autodesk.AutoCAD.PlottingServices. _
                                            ProcessPlotState.NotPlotting Then
          Using acPlEng As PlotEngine = PlotFactory.CreatePublishEngine()

              '' 使用 PlotProgressDialog 对话框跟踪打印进度
              Dim acPlProgDlg As PlotProgressDialog = New PlotProgressDialog(False,
_
                                                                             1,
_

True)

              Using (acPlProgDlg)
                  '' 定义打印开始时显示的状态信息
                  acPlProgDlg.PlotMsgString(PlotMessageIndex.DialogTitle) = _
                                                          "Plot Progress"

acPlProgDlg.PlotMsgString(PlotMessageIndex.CancelJobButtonMessage) = _
                                                          "Cancel Job"

acPlProgDlg.PlotMsgString(PlotMessageIndex.CancelSheetButtonMessage) = _
```

```
                                                        "Cancel Sheet"

acPlProgDlg.PlotMsgString(PlotMessageIndex.SheetSetProgressCaption) = _
                                                        "Sheet Set
Progress"

acPlProgDlg.PlotMsgString(PlotMessageIndex.SheetProgressCaption) = _
                                                        "Sheet Progress"

                '' 设置打印进度范围
                acPlProgDlg.LowerPlotProgressRange = 0
                acPlProgDlg.UpperPlotProgressRange = 100
                acPlProgDlg.PlotProgressPos = 0
                '' 显示打印进度对话框
                acPlProgDlg.OnBeginPlot()
                acPlProgDlg.IsVisible = True
                '' 开始打印
                acPlEng.BeginPlot(acPlProgDlg, Nothing)
                '' 定义打印输出
                acPlEng.BeginDocument(acPlInfo, _
                                      acDoc.Name, _
                                      Nothing, _
                                      1, _
                                      True, _
                                      "e:\myplot")
                '' 显示当前打印任务的有关信息
                acPlProgDlg.PlotMsgString(PlotMessageIndex.Status) = _
                                          "Plotting: " & acDoc.Name & _
                                          " - " & acLayout.LayoutName
                '' 设置图纸进度范围
                acPlProgDlg.OnBeginSheet()
                acPlProgDlg.LowerSheetProgressRange = 0
                acPlProgDlg.UpperSheetProgressRange = 100
                acPlProgDlg.SheetProgressPos = 0
                '' 打印第一张图/布局
                Dim acPlPageInfo As PlotPageInfo = New PlotPageInfo()
                acPlEng.BeginPage(acPlPageInfo, _
                                  acPlInfo, _
                                  True, _
                                  Nothing)

                acPlEng.BeginGenerateGraphics(Nothing)
                acPlEng.EndGenerateGraphics(Nothing)
```

```
                    '' 结束第一张图/布局的打印
                    acPlEng.EndPage(Nothing)
                    acPlProgDlg.SheetProgressPos = 100
                    acPlProgDlg.OnEndSheet()

                    '' 结束文档局的打印
                    acPlEng.EndDocument(Nothing)

                    '' 打印结束
                    acPlProgDlg.PlotProgressPos = 100
                    acPlProgDlg.OnEndPlot()
                    acPlEng.EndPlot(Nothing)
                End Using
            End Using
        End If
    End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.PlottingServices;

[CommandMethod("PlotCurrentLayout")]
public static void PlotCurrentLayout()
{
  // 获取当前文档和数据库，启动事务
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 引用布局管理器 LayoutManager
      LayoutManager acLayoutMgr;
      acLayoutMgr = LayoutManager.Current;
      // 获取当前布局，在命令行窗口显示布局名字
      Layout acLayout;
      acLayout =
acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout),
                                      OpenMode.ForRead) as Layout;
      // 从布局中获取 PlotInfo
      PlotInfo acPlInfo = new PlotInfo();
      acPlInfo.Layout = acLayout.ObjectId;
```

```
// 复制布局中的 PlotSettings
PlotSettings acPlSet = new PlotSettings(acLayout.ModelType);
acPlSet.CopyFrom(acLayout);
// 更新 PlotSettings 对象
PlotSettingsValidator acPlSetVdr = PlotSettingsValidator.Current;
// 设置打印区域
acPlSetVdr.SetPlotType(acPlSet,
                Autodesk.AutoCAD.DatabaseServices.PlotType.Extents);
// 设置打印比例
acPlSetVdr.SetUseStandardScale(acPlSet, true);
acPlSetVdr.SetStdScaleType(acPlSet, StdScaleType.ScaleToFit);
// 居中打印
acPlSetVdr.SetPlotCentered(acPlSet, true);

// 设置使用的打印设备
acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3",
                                    "ANSI_A_(8.50_x_11.00_Inches)");

// 用上述设置信息覆盖 PlotInfo 对象，
// 不会将修改保存回布局
acPlInfo.OverrideSettings = acPlSet;

// 验证打印信息
PlotInfoValidator acPlInfoVdr = new PlotInfoValidator();
acPlInfoVdr.MediaMatchingPolicy = MatchingPolicy.MatchEnabled;
acPlInfoVdr.Validate(acPlInfo);

// 检查是否有正在处理的打印任务
if (PlotFactory.ProcessPlotState == ProcessPlotState.NotPlotting)
{
    using (PlotEngine acPlEng = PlotFactory.CreatePublishEngine())
    {
        // 使用 PlotProgressDialog 对话框跟踪打印进度
        PlotProgressDialog acPlProgDlg = new PlotProgressDialog(false,
                                                                1,
                                                                true);

        using (acPlProgDlg)
        {
            // 定义打印开始时显示的状态信息
            acPlProgDlg.set_PlotMsgString(PlotMessageIndex.DialogTitle,
                                          "Plot Progress");
```

```
acPlProgDlg.set_PlotMsgString(PlotMessageIndex.CancelJobButtonMessage,
                                              "Cancel Job");

acPlProgDlg.set_PlotMsgString(PlotMessageIndex.CancelSheetButtonMessage,
                                              "Cancel Sheet");

acPlProgDlg.set_PlotMsgString(PlotMessageIndex.SheetSetProgressCaption,
                                              "Sheet Set Progress");

acPlProgDlg.set_PlotMsgString(PlotMessageIndex.SheetProgressCaption,
                                              "Sheet Progress");

                    // 设置打印进度范围
                    acPlProgDlg.LowerPlotProgressRange = 0;
                    acPlProgDlg.UpperPlotProgressRange = 100;
                    acPlProgDlg.PlotProgressPos = 0;

                    // 显示打印进度对话框
                    acPlProgDlg.OnBeginPlot();
                    acPlProgDlg.IsVisible = true;

                    // 开始打印
                    acPlEng.BeginPlot(acPlProgDlg, null);
                    // 定义打印输出
                    acPlEng.BeginDocument(acPlInfo,
                                          acDoc.Name,
                                          null,
                                          1,
                                          true,
                                          "e:\\myplot");
                    // 显示当前打印任务的有关信息
                    acPlProgDlg.set_PlotMsgString(PlotMessageIndex.Status,
                                              "Plotting: " + acDoc.Name + " - "
+
                                              acLayout.LayoutName);
                    // 设置图纸进度范围
                    acPlProgDlg.OnBeginSheet();
                    acPlProgDlg.LowerSheetProgressRange = 0;
                    acPlProgDlg.UpperSheetProgressRange = 100;
                    acPlProgDlg.SheetProgressPos = 0;
```

```
                // 打印第一张图/布局
                PlotPageInfo acPlPageInfo = new PlotPageInfo();
                acPlEng.BeginPage(acPlPageInfo,
                                  acPlInfo,
                                  true,
                                  null);

                acPlEng.BeginGenerateGraphics(null);
                acPlEng.EndGenerateGraphics(null);
                // 结束第一张图/布局的打印
                acPlEng.EndPage(null);
                acPlProgDlg.SheetProgressPos = 100;
                acPlProgDlg.OnEndSheet();
                // 结束文档局的打印
                acPlEng.EndDocument(null);
                // 打印结束
                acPlProgDlg.PlotProgressPos = 100;
                acPlProgDlg.OnEndPlot();
                acPlEng.EndPlot(null);
            }
        }
    }
  }
}
```

VBA/ActiveX 代码参考

```
Sub PlotModelSpace()
    ' 确认活动空间是模型空间
    If ThisDrawing.ActiveSpace = acPaperSpace Then
        ThisDrawing.MSpace = True
        ThisDrawing.ActiveSpace = acModelSpace
    End If

    ' 设置打印区域的范围和比例
    ThisDrawing.ModelSpace.Layout.PlotType = acExtents
    ThisDrawing.ModelSpace.Layout. _
                          StandardScale = acScaleToFit

    ' 居中打印
    ThisDrawing.ModelSpace.Layout.CenterPlot = True

    ' 设置使用的打印设备
    ThisDrawing.ModelSpace.Layout. _
                          ConfigName = "DWF6 ePlot.pc3"
```

```
    ' 设置打印输出的尺寸
    ThisDrawing.ModelSpace.Layout. _
                          CanonicalMediaName = "ANSI_A_(8.50_x_11.00_Inches)"

    ' 设置打印份数为 1
    ThisDrawing.Plot.NumberOfCopies = 1

    ' 启动打印
    ThisDrawing.Plot.PlotToFile "c:\myplot"
End Sub
```

设备名称可以使用 ConfigName 属性指定，可以使用 PlotToDevice ()方法通过指定一个 PC3 文件来选择别的打印设备。

6.4.2 从图纸空间打印

打印图纸空间布局与打印模型布局一样。打印图纸空间布局前，确认已将布局初始化并且设置好了各视口的设置信息和想要的模型视图。关于在图纸空间布局定义视口的内容，参见（§ 6.3.2 *创建图纸空间视口*）。打印图纸空间的一个布局，请参考上一节（§ 6.4.1 *从模型空间打印*）。

第 7 章 使用事件

事件是 AutoCAD 发出的通知或消息，用以通知用户当前的会话状态，或提醒用户发生了什么情况。例如，当保存图形时会触发 BeginSave 事件。当关闭图形时，启动一个命令时，甚至修改一个系统变量时，都会有事件被触发。知道了这些信息，我们就可以编写一个子程序，或事件处理程序，使用这些事件来跟踪对图形的修改，或者跟踪记录用户在绘制某一特定图形时所花费的时间，等。

本章主要内容：

7.1 了解 AutoCAD 中的事件

AutoCAD 中有许多不同类型的事件。下面是一些常见的事件类型：

- **Application 事件** – 对 AutoCAD 关闭、修改系统变量、开始双击以及进入和离开模式的状态等作出反应。对于系统变量的修改也有文档级事件。
- **Database 事件** – 对保存图形、添加删除修改对象、插入块参考、添加和修改外部图形（xrefs）等作出反应。
- **Document 事件** – 对关闭图形、运行 AutoCAD 命令、运行 AutoLISP 命令或函数、修改系统变量等作出反应。
- **DocumentCollection 事件** – 对文档的创建与销毁、成为活动文档或进入非活动状态、以及文档的锁定模式发生变化等作出反应。
- **Editor 事件** – 对请求用户输入期间发生的变化作出反应。
- **Graphics 事件** – 对视图的创建与销毁、视图的配置发生变化等作出反应。
- **Plotting 事件** – 对打印布局作出反应。

- **Publishing 事件** – 对发布布局作出反应。
- **Runtime 事件** – 对加载与卸载模块、变量发生变化或正在修改变量等作出反应。
- **Windows 事件** – 对窗体的状态栏、托盘项目、调色板和信息中心的变化做出反应。

响应事件的子程序称之为事件处理程序（*event handlers*），每当指定的事件被触发时，就会自动执行事件处理程序。事件返回的参数所包含的信息，如 SystemVariableChanging 事件中的系统变量名，会从事件处理程序传递给 SystemVariableChangingEventArgs 对象。

7.2 事件处理程序的原则

事件只是简单地提供了关于 AutoCAD 的状态或发生在 AutoCAD 中的活动的信息，记住这一点非常重要的。尽管可以编写事件处理程序来响应那些事件，但触发事件处理程序的操作中间是有个 AutoCAD 在那儿的。因此，如果想让事件处理程序与 AutoCAD 及数据库一起使用时提供安全可靠的操作，必须对事件处理程序能做什么不能做什么有所限制。

- 原则 1：不要依赖于事件的顺序。

  编写事件处理程序时，不要依赖于你所认为的事件发生的确切顺序序列。例如，如果你运行 OPEN 命令，CommandWillStart 事件、DocumentCreateStarted 事件、DocumentCreated 事件和 CommandEnded 事件都会被触发。然而，这些事件可能不会每次都以确切的顺序发生。你唯一可以依赖的是成对儿发生的那两个事件：开始事件和结束事件。
- 原则 2：不要依赖于操作的顺序。

  如果你删除了对象 1，然后又删除了对象 2，不要依赖一个事实，即您将先收到对象 1 的 ObjectErased 事件，然后收到对象 2 的 ObjectErased 事件。你可能会先收到对象 2 的 ObjectErased 事件。
- 原则 3：不要从事件处理程序内尝试任何交互功能。

  试图从事件处理程序内执行交互功能会引起严重问题，因为事件被触发时 AutoCAD 可能仍在处理命令。因此，应该牢记避免从事件处理程序中执行下列操作：在命令提示行请求输入、请求选择对象以及使用 SendStringToExecute()方法等。
- 原则 4：不要从事件处理程序内启动对话框。

  一般认为对话框是一种交互功能，会干扰到 AutoCAD 的当前操作，而消息框和警告框不是交互功能，可以放心使用；不过，在下列事件的处理程序里发出消息框可能会导致意想不到的结果序列： EnterModal、LeaveModal、DocumentActivated、DocumentToBeDeactivated 等。
- 原则 5：可以向数据库中的任何对象写入数据，但应避免修改引发事件的那个对象。

  很显然，引发事件的那个对象已经打开并且还处在当前的操作过程中。因此，应避免从该对象的事件处理程序修改这个对象。不过，可以放心地从触发事件的对象读取信息。
- 原则 6：不要从事件处理程序执行可能会触发相同事件的任何操作。

如果在事件处理程序中执行触发同一事件的相同动作，就会进入一个无限循环（死循环）。例如，永远不要试图在 ObjectOpenedForModify 事件的处理程序中打开一个对象，否则 AutoCAD 就会不停地打开对象、打开对象……

- 原则 7：当 AutoCAD 显示一个模式对话框时没有事件被触发。

7.3 事件的注册与撤销

响应一个事件前，必须先在 AutoCAD 中注册该事件。注册事件的方法是，新创建一个所需类型的事件处理程序，然后将该事件处理程序赋给要在其中注册事件的那个对象。一旦处理完事件，最好撤销该事件的注册，以减少与其他处理程序的冲突，以及减少 AutoCAD 为维护你的事件处理程序额外增加的对内存和 CPU 的占用。

注册事件

通过将事件处理程序添加给事件来注册一个事件。事件处理程序对象需要一个子程序，必须先在项目中定义好。该子程序通常有 2 个参数：一个是类型 Object，另一个表示事件的返回参数。注册事件使用 VB.NET 的 AddHandler 语句，或使用 C#的 += 操作符。

下面的代码将一个 SystemVariableChangedEventHandler 类型的名为 appSysVarChanged 的子程序注册给了 SystemVariableChanged 事件。该子程序接受两个参数：Object 和 SystemVariableChangedEventArgs，其中 SystemVariableChangedEventArgs 对象返回事件被触发时被修改的系统变量的名称（详见下一节的示例代码）。

VB.NET

```
AddHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
```

C#

```
Application.SystemVariableChanged +=
            new SystemVariableChangedEventHandler(appSysVarChanged)
```

撤销注册一个事件

撤销注册一个事件，从该事件中移除事件处理程序即可。使用的语法和注册事件的语法类似，只是将 AddHandler 换成 RemoveHandler，或将+=操作符换成-=操作符。

与上面的代码相反，下列代码从 SystemVariableChanged 事件中撤销对 SystemVariableChangedEventHandler 类型的子程序 appSysVarChanged 的注册。

VB.NET

```
RemoveHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
```

C#

```
Application.SystemVariableChanged -=
```

```
new SystemVariableChangedEventHandler(appSysVarChanged)
```

7.4 处理 Application 事件

Application 对象事件用来响应应用程序窗口。一旦注册了 Application 事件，它将保持注册知道关闭 AutoCAD，或事件被撤销。

Application 对象的可用事件如下：

BeginQuit

AutoCAD 会话结束前触发。

PreTranslateMessage

在 AutoCAD 将要翻译消息前触发。

QuitAborted

当试图关闭 AutoCAD 被终止时触发。

QuitWillStart

在 BeginQuit 事件之后、关闭开始前触发。

SystemVariableChanged

已经作出修改系统变量的试图时触发。当通过 setvar 命令或通过在命令行中输入变量名来修改一个系统变量时，该事件一定会发生。对于其他的引起系统变量改变的 AutoCAD 内置命令，不能保证会触发该事件。

SystemVariableChanging

将要作出修改系统变量的试图前触发。

激活一个 AutoCAD 对象事件

本示例演示如何注册 SystemVariableChanged 事件的事件处理程序。注册之后，当你在命令行修改一个系统变量后，会弹出一个消息框，显示你修改的系统变量的名称及修改后的变量值。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("AddAppEvent")> _
```

```
Public Sub AddAppEvent()
  AddHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
End Sub

<CommandMethod("RemoveAppEvent")> _
Public Sub RemoveAppEvent()
  RemoveHandler Application.SystemVariableChanged, AddressOf appSysVarChanged
End Sub

Public Sub appSysVarChanged(ByVal senderObj As Object, _
                            ByVal sysVarChEvtArgs As
Autodesk.AutoCAD.ApplicationServices. _
                            SystemVariableChangedEventArgs)

  Dim oVal As Object = Application.GetSystemVariable(sysVarChEvtArgs.Name)

  '' 弹出消息框，显示系统变量的名称及新值
  Application.ShowAlertDialog(sysVarChEvtArgs.Name & " was changed." & _
                              vbLf & "New value: " & oVal.ToString())
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
[CommandMethod("AddAppEvent")]
public void AddAppEvent()
{
  Application.SystemVariableChanged +=
      new Autodesk.AutoCAD.ApplicationServices.
          SystemVariableChangedEventHandler(appSysVarChanged);
}

[CommandMethod("RemoveAppEvent")]
public void RemoveAppEvent()
{
  Application.SystemVariableChanged -=
      new Autodesk.AutoCAD.ApplicationServices.
          SystemVariableChangedEventHandler(appSysVarChanged);
}

public void appSysVarChanged(object senderObj,
                             Autodesk.AutoCAD.ApplicationServices.
                             SystemVariableChangedEventArgs sysVarChEvtArgs)
{
```

```
  object oVal = Application.GetSystemVariable(sysVarChEvtArgs.Name);

  // 弹出消息框，显示系统变量的名称及新值
  Application.ShowAlertDialog(sysVarChEvtArgs.Name + " was changed." +
                              "\nNew value: " + oVal.ToString());
}
```

VBA/ActiveX 代码参考

```
Public WithEvents ACADApp As AcadApplication

Sub Example_AcadApplication_Events()
    ' 初始化公共变量(ACADApp)
    '
    ' 先运行这个子程序

    Set ACADApp = ThisDrawing.Application
End Sub

Private Sub ACADApp_SysVarChanged(ByVal SysvarName As String, _
                                  ByVal newVal As Variant)
    ' 这个子程序截获 Application 的 SysVarChanged 事件

    MsgBox (SysvarName & " was changed." & _
            vbLf & "New value: " & newVal)
End Sub
```

7.5 处理 Document 事件

Document 对象事件用来响应文档窗口。当注册一个文档事件时，该事件只与事件文档所在的 Document 对象相关联。因此，如果需要将一个事件注册给所有文档，应使用 DocumentCollection 对象的 DocumentCreated 事件给每个新建的或打开的图形文件注册事件。

Document 对象的可用事件如下：

BeginDocumentClose

收到关闭图形的请求后触发。

CloseAborted

终止试图关闭图形时触发。

CloseWillStart

BeginDocumentClose 事件之后开始关闭图形前触发。

CommandCancelled

当一个命令在执行完之前被取消时触发。

CommandEnded

在一个命令执行完时立即触发。

CommandFailed

命令没有被取消但运行失败时触发。

CommandWillStart

启动一个命令后（执行完之前）立即触发。

ImpliedSelectionChanged

当前 Pickfirst 选择集发生变化时触发。

LispCancelled

当一个 LISP 表达式的计算被取消时触发。

LispEnded

当一个 LISP 表达式的计算完成时触发。

LispWillStart

AutoCAD 收到计算 LISP 表达式的请求后立即触发。

UnknownCommand

在命令提示行键入一个未知命令时立即触发。

激活一个 Document 对象事件

下面这个示例使用 BeginDocumentClose 事件提示用户是否继续关闭当前图形。事件处理程序显示一个含有 Yes 按钮和 No 按钮的消息框， 单击 No 按钮将使用事件处理程序返回的参数的 Veto()方法终止关闭图形。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("AddDocEvent")> _
Public Sub AddDocEvent()
  '' 获取当前文档
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  AddHandler acDoc.BeginDocumentClose, AddressOf docBeginDocClose
```

```
End Sub

<CommandMethod("RemoveDocEvent")> _
Public Sub RemoveDocEvent()
  '' 获取当前文档
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

  RemoveHandler acDoc.BeginDocumentClose, AddressOf docBeginDocClose
End Sub

Public Sub docBeginDocClose(ByVal senderObj As Object, _
                            ByVal docBegClsEvtArgs As
DocumentBeginCloseEventArgs)

  '' 显示消息框提示是否继续关闭文档
  If System.Windows.Forms.MessageBox.Show( _
                      "The document is about to be closed." & _
                      vbLf & "Do you want to continue?", _
                      "Close Document", _
                      System.Windows.Forms.MessageBoxButtons.YesNo) = _
                      System.Windows.Forms.DialogResult.No Then
      docBegClsEvtArgs.Veto()
  End If
End If
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("AddDocEvent")]
public void AddDocEvent()
{
  // 获取当前文档
  Document acDoc = Application.DocumentManager.MdiActiveDocument;

  acDoc.BeginDocumentClose +=
      new DocumentBeginCloseEventHandler(docBeginDocClose);
}

[CommandMethod("RemoveDocEvent")]
public void RemoveDocEvent()
{
  // 获取当前文档
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
```

```
  acDoc.BeginDocumentClose -=
      new DocumentBeginCloseEventHandler(docBeginDocClose);
}

public void docBeginDocClose(object senderObj,
                             DocumentBeginCloseEventArgs docBegClsEvtArgs)
{
  // 显示消息框提示是否继续关闭文档
  if (System.Windows.Forms.MessageBox.Show(
                        "The document is about to be closed." +
                        "\nDo you want to continue?",
                        "Close Document",
                        System.Windows.Forms.MessageBoxButtons.YesNo) ==
                        System.Windows.Forms.DialogResult.No)
  {
      docBegClsEvtArgs.Veto();
  }
}
```

VBA/ActiveX 代码参考

```
Private Sub AcadDocument_BeginDocClose(Cancel As Boolean)
    ' 这个子程序截获 Document 的 BeginDocClose 事件

    If MsgBox("The document is about to be closed." & _
              vbLf & "Do you want to continue?", vbYesNo, _
              "Close Document") = vbNo Then

          ' 否决文档的关闭
          Cancel = True
    End If
End Sub
```

7.6 处理 DocumentCollection 对象事件

DocumentCollection 对象事件用来响应应用程序中打开的文档。与 Document 对象事件不同，DocumentCollection 事件会保持其注册状态直到关闭 AutoCAD 或被撤销注册为止。

DocumentCollection 对象的可用事件如下：

DocumentActivated

激活一个文档窗口时触发。

DocumentActivationChanged

活动文档窗口被关闭或销毁后触发。

DocumentBecameCurrent

一个文档窗口被设置为当前时，如果与前一个活动文档窗口不同，则触发此事件。

DocumentCreated

当创建一个文档窗口后触发。该事件发生在新建一个图形或打开一个图形之后。

DocumentCreateStarted

当将要创建一个文档窗口前触发。该事件发生在新建一个图形或打开一个图形之前。

DocumentCreationCanceled

在新建一个图形或打开一个图形的请求被取消时触发。

DocumentDestroyed

文档窗口被销毁并且与其关联的数据库对象被删除之前触发。

DocumentLockModeChanged

文档的锁定模式发生改变之后触发。

DocumentLockModeChangeVetoed

修改文档的锁定模式被否决后触发。

DocumentLockModeWillChange

文档的锁定模式发生改变之前触发。

DocumentToBeActivated

文档将要被激活时触发。

DocumentToBeDeactivated

文档将要被关闭时触发。

DocumentToBeDestroyed

文档将要被销毁时触发。

激活一个 DocumentCollection 对象事件

下面这个示例使用 DocumentActivated 事件显示出文档窗口是什么时候被激活的。事件发生时，会弹出一个消息框，显示被激活的图形的名称。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
```

```
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("AddDocColEvent")> _
Public Sub AddDocColEvent()
  AddHandler Application.DocumentManager.DocumentActivated, _
      AddressOf docColDocAct
End Sub

<CommandMethod("RemoveDocColEvent")> _
Public Sub RemoveDocColEvent()
  RemoveHandler Application.DocumentManager.DocumentActivated, _
      AddressOf docColDocAct
End Sub

Public Sub docColDocAct(ByVal senderObj As Object, _
                        ByVal docColDocActEvtArgs As
DocumentCollectionEventArgs)
  Application.ShowAlertDialog(docColDocActEvtArgs.Document.Name & _
                              " was activated.")
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("AddDocColEvent")]
public void AddDocColEvent()
{
  Application.DocumentManager.DocumentActivated +=
      new DocumentCollectionEventHandler(docColDocAct);
}

[CommandMethod("RemoveDocColEvent")]
public void RemoveDocColEvent()
{
  Application.DocumentManager.DocumentActivated -=
      new DocumentCollectionEventHandler(docColDocAct);
}

public void docColDocAct(object senderObj,
                         DocumentCollectionEventArgs docColDocActEvtArgs)
{
  Application.ShowAlertDialog(docColDocActEvtArgs.Document.Name +
                              " was activated.");
```

```
}
```

VBA/ActiveX 代码参考

```
Private Sub AcadDocument_Activate()
    ' 截获文档激活事件

    MsgBox ThisDrawing.Name & " was activated."
End Sub
```

7.7 处理 Object 级事件

Object 事件用来响应打开、添加、修改、删除图形数据库中对象这些操作。有两类与对象有关的事件：Object 级事件和 Database 级事件。 Object 级事件被定义为对一个数据库中的特定对象作出反应，而 Database 级事件则对一个数据库中的所有对象作出回应。

通过给数据库对象的一个事件注册一个事件处理程序来定义对象级事件；通过给打开的 Database 对象的一个事件注册一个事件处理程序来定义数据库级事件。

DBObjects 类定义的可用事件如下：

Cancelled

当对象打开被取消时触发。

Copied

对象被复制之后触发。

Erased

当对象标记为要删除或撤销删除时触发。

Goodbye

由于关联的数据库被销毁而将要从内存中删除对象时触发此事件。

Modified

对象被修改时触发。

ModifiedXData

附加到对象上的 XData 被修改时触发。

ModifyUndone

上一个修改对象的操作被撤销时触发。

ObjectClosed

对象被关闭时触发。

OpenedForModify

对象被修改前触发。

Reappended

当执行一个 Undo 操作后把对象从数据库中删除了，而后又执行 Redo 操作将对象重新添加到数据库时，触发该事件。

SubObjectModified

对象的一个子对象被修改时触发。

Unappended

当执行一个 Undo 操作后把对象从数据库中删除时触发。

下列事件用来响应对 Database 类型对象的修改：

ObjectAppended

一个对象被添加到数据库时触发。

ObjectErased

从数据库删除或撤销删除一个对象时触发。

ObjectModified

对象已被修改时触发。

ObjectOpenedForModify

对象被修改前触发。

ObjectReappended

当执行一个 Undo 操作后把对象从数据库中删除了，而后又执行 Redo 操作将对象重新添加到数据库时，触发该事件。

ObjectUnappended

当执行一个 Undo 操作后把对象从数据库中删除时触发。

激活 Object 事件

本实例创建一个轻量级多段线并注册多段线对象的 Modified 事件。每当修改多段线时，事件处理程序就显示闭合多段线的新面积。要想触发事件，只需在 AutoCAD 中修改多段线的大小即可。记住激活事件处理程序前必须先运行 CreatePLineWithEvents 子程序（VBA）。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

'' Polyline 对象全局变量
Dim acPoly As Polyline = Nothing

<CommandMethod("AddPlObjEvent")> _
Public Sub AddPlObjEvent()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开 BlockTable
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以写模式打开 BlockTable 记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                      OpenMode.ForWrite)

      '' 创建闭合多段线
      acPoly = New Polyline()
      acPoly.AddVertexAt(0, New Point2d(1, 1), 0, 0, 0)
      acPoly.AddVertexAt(1, New Point2d(1, 2), 0, 0, 0)
      acPoly.AddVertexAt(2, New Point2d(2, 2), 0, 0, 0)
      acPoly.AddVertexAt(3, New Point2d(3, 3), 0, 0, 0)
      acPoly.AddVertexAt(4, New Point2d(3, 2), 0, 0, 0)
      acPoly.Closed = True

      '' 添加新对象到块表记录及事务
      acBlkTblRec.AppendEntity(acPoly)
      acTrans.AddNewlyCreatedDBObject(acPoly, True)

      AddHandler acPoly.Modified, AddressOf acPolyMod

      '' 保存新对象到数据库
      acTrans.Commit()
  End Using
```

```
End Sub

<CommandMethod("RemovePlObjEvent")> _
Public Sub RemovePlObjEvent()
  If acPoly <> Nothing Then
      '' 获取当前文档和数据库，启动事务
      Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
      Dim acCurDb As Database = acDoc.Database

      Using acTrans As Transaction =
acCurDb.TransactionManager.StartTransaction()
          '' 以读模式打开多段线
          acPoly = acTrans.GetObject(acPoly.ObjectId, _
                                     OpenMode.ForRead)

          If acPoly.IsWriteEnabled = False Then
              acPoly.UpgradeOpen()
          End If

          RemoveHandler acPoly.Modified, AddressOf acPolyMod
          acPoly = Nothing
      End Using
  End If
End Sub

Public Sub acPolyMod(ByVal senderObj As Object, ByVal evtArgs As EventArgs)
  Application.ShowAlertDialog("The area of " & _
                              acPoly.ToString() & " is: " & _
                              acPoly.Area)
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;

// Polyline 对象全局变量
Polyline acPoly = null;

[CommandMethod("AddPlObjEvent")]
public void AddPlObjEvent()
{
  // 获取当前文档和数据库，启动事务
```

```
  Document acDoc = Application.DocumentManager.MdiActiveDocument;
  Database acCurDb = acDoc.Database;

  using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
  {
      // 以读模式打开 BlockTable
      BlockTable acBlkTbl;
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                                   OpenMode.ForRead) as BlockTable;

      // 以写模式打开 BlockTable 记录 Model 空间
      BlockTableRecord acBlkTblRec;
      acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                                      OpenMode.ForWrite) as BlockTableRecord;

      // 创建闭合多段线
      acPoly = new Polyline();
      acPoly.AddVertexAt(0, new Point2d(1, 1), 0, 0, 0);
      acPoly.AddVertexAt(1, new Point2d(1, 2), 0, 0, 0);
      acPoly.AddVertexAt(2, new Point2d(2, 2), 0, 0, 0);
      acPoly.AddVertexAt(3, new Point2d(3, 3), 0, 0, 0);
      acPoly.AddVertexAt(4, new Point2d(3, 2), 0, 0, 0);
      acPoly.Closed = true;

      // 添加新对象到块表记录及事务
      acBlkTblRec.AppendEntity(acPoly);
      acTrans.AddNewlyCreatedDBObject(acPoly, true);

      acPoly.Modified += new EventHandler(acPolyMod);

      // 保存新对象到数据库
      acTrans.Commit();
  }
}

[CommandMethod("RemovePlObjEvent")]
public void RemovePlObjEvent()
{
  if (acPoly != null)
  {
      // 获取当前文档和数据库，启动事务
      Document acDoc = Application.DocumentManager.MdiActiveDocument;
      Database acCurDb = acDoc.Database;
```

```
      using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
      {
          // 以读模式打开多段线
          acPoly = acTrans.GetObject(acPoly.ObjectId,
                                     OpenMode.ForRead) as Polyline;

          if (acPoly.IsWriteEnabled == false)
          {
              acPoly.UpgradeOpen();
          }

          acPoly.Modified -= new EventHandler(acPolyMod);
          acPoly = null;
      }
  }
}

public void acPolyMod(object senderObj, EventArgs evtArgs)
{
  Application.ShowAlertDialog("The area of " +
                              acPoly.ToString() + " is: " +
                              acPoly.Area);
}
```

VBA/ActiveX 代码参考

```
Public WithEvents PLine As AcadLWPolyline
Sub CreatePLineWithEvents()
    ' 创建多段线
    Dim points(0 To 9) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 3
    points(8) = 3: points(9) = 2

    Set PLine = ThisDrawing.ModelSpace. _
    AddLightWeightPolyline(points)
    PLine.Closed = True

    ThisDrawing.Application.ZoomAll
End Sub

Private Sub PLine_Modified(ByVal pObject As AutoCAD.IAcadObject)
    ' 多段线大小改变时触发这个事件
```

```
    ’ 多段线被删除也会触发 Modified 事件，因此我们使用
    ’ 错误处理代码避免从已删除的对象读取数据。

    On Error GoTo ERRORHANDLER
    MsgBox "The area of " & pObject.ObjectName & " is: " & pObject.Area
    Exit Sub

ERRORHANDLER:
    MsgBox Err.Description
End Sub
```

附：**Database** 类定义的事件：

```
public sealed class Database : RXObject, IDynamicMetaObjectProvider
{
    // Events
    public event EventHandler AbortDxfIn;
    public event EventHandler AbortDxfOut;
    public event EventHandler AbortSave;
    public event IdMappingEventHandler BeginDeepClone;
    public event IdMappingEventHandler BeginDeepCloneTranslation;
    public event EventHandler BeginDxfIn;
    public event EventHandler BeginDxfOut;
    public event BeginInsertEventHandler BeginInsert;
    public event DatabaseIOEventHandler BeginSave;
    public event BeginWblockBlockEventHandler BeginWblockBlock;
    public event BeginWblockEntireDatabaseEventHandler BeginWblockEntireDatabase;
    public event BeginWblockObjectsEventHandler BeginWblockObjects;
    public event BeginWblockSelectedObjectsEventHandler BeginWblockSelectedObjects;
    public static  event EventHandler DatabaseConstructed;
    public event EventHandler DatabaseToBeDestroyed;
    public event EventHandler DeepCloneAborted;
    public event EventHandler DeepCloneEnded;
    public event EventHandler Disposed;
    public event DatabaseIOEventHandler DwgFileOpened;
    public event EventHandler DxfInComplete;
    public event EventHandler DxfOutComplete;
    public event EventHandler InitialDwgFileOpenComplete;
    public event EventHandler InsertAborted;
    public event EventHandler InsertEnded;
```

```
    public event IdMappingEventHandler InsertMappingAvailable;
    public event ObjectEventHandler ObjectAppended;
    public event ObjectErasedEventHandler ObjectErased;
    public event ObjectEventHandler ObjectModified;
    public event ObjectEventHandler ObjectOpenedForModify;
    public event ObjectEventHandler ObjectReappended;
    public event ObjectEventHandler ObjectUnappended;
    public event EventHandler PartialOpenNotice;
    public event ProxyResurrectionCompletedEventHandler ProxyResurrectionCompleted;
    public event DatabaseIOEventHandler SaveComplete;
    public event SystemVariableChangedEventHandler SystemVariableChanged;
    public event SystemVariableChangingEventHandler SystemVariableWillChange;
    public event EventHandler WblockAborted;
    public event EventHandler WblockEnded;
    public event IdMappingEventHandler WblockMappingAvailable;
    public event WblockNoticeEventHandler WblockNotice;
    public static  event EventHandler XrefAttachAborted;
    public event EventHandler XrefAttachEnded;
    public event XrefBeginOperationEventHandler XrefBeginAttached;
    public event XrefBeginOperationEventHandler XrefBeginOtherAttached;
    public event XrefBeginOperationEventHandler XrefBeginRestore;
    public event XrefComandeeredEventHandler XrefComandeered;
    public event XrefPreXrefLockFileEventHandler XrefPreXrefLockFile;
    public event XrefRedirectedEventHandler XrefRedirected;
    public event EventHandler XrefRestoreAborted;
    public event EventHandler XrefRestoreEnded;
    public event XrefSubCommandAbortedEventHandler XrefSubCommandAborted;
    public event XrefSubCommandStartEventHandler XrefSubCommandStart;

}
```

7.8 使用.NET 注册基于 COM 的事件

AutoCAD COM Automation 库提供了一些在.NET API 中没有的独特事件。注册 COM 库定义的事件与使用 VB 或 VBA 初始化事件的方法不同。 给事件注册一个处理程序的方法是使用 VB.NET 的 AddHandler 语句或 C#的+=操作符。事件处理程序需要事件发生时要调用的子程序的地址。

注册一个基于 COM 的事件

本示例演示使用 COM 交互注册 BeginFileDrop 事件。BeginFileDrop 事件是 AutoCAD COM Automation 库中 Application 对象的成员。在 AutoCAD 加载程序（netload 命令），在命令提示行键入 AddCOMEvent 命令，然后拖拽一个 DWG 文件到绘图窗口。这是会弹出一个消息框，询问是否继续。使用 RemoveCOMEvent 命令移除事件处理程序。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

Imports Autodesk.AutoCAD.Interop
Imports Autodesk.AutoCAD.Interop.Common

'' 定义一个全局变量，用于 AddCOMEvent 命令和 RemoveCOMEvent 命令
Dim acAppCom As AcadApplication

<CommandMethod("AddCOMEvent")> _
Public Sub AddCOMEvent()
  '' 用全局变量保存到应用程序的引用
  '' 并注册 COM 事件 BeginFileDrop
  acAppCom = Application.AcadApplication
  AddHandler acAppCom.BeginFileDrop, AddressOf appComBeginFileDrop
End Sub

<CommandMethod("RemoveCOMEvent")> _
Public Sub RemoveCOMEvent()
  '' 撤销注册的 COM 事件处理程序
  RemoveHandler acAppCom.BeginFileDrop, AddressOf appComBeginFileDrop
  acAppCom = Nothing
End Sub

Public Sub appComBeginFileDrop(ByVal strFileName As String, _
                               ByRef bCancel As Boolean)
```

```
  '' 显示消息框，提示是否继续插入 DWG 文件
  If System.Windows.Forms.MessageBox.Show("AutoCAD is about to load " & _
                           strFileName & vbLf & _
                           "Do you want to continue loading this file?", _
                           "DWG File Dropped", _
                           System.Windows.Forms.MessageBoxButtons.YesNo) = _
    System.Windows.Forms.DialogResult.No Then
      bCancel = True
  End If
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

using Autodesk.AutoCAD.Interop;
using Autodesk.AutoCAD.Interop.Common;

// 定义一个全局变量，用于 AddCOMEvent 命令和 RemoveCOMEvent 命令
AcadApplication acAppCom;

[CommandMethod("AddCOMEvent")]
public void AddCOMEvent()
{
  // 用全局变量保存到应用程序的引用
  // 并注册 COM 事件 BeginFileDrop
  acAppCom = Application.AcadApplication as AcadApplication;
  acAppCom.BeginFileDrop +=
      new
_DAcadApplicationEvents_BeginFileDropEventHandler(appComBeginFileDrop);
}

[CommandMethod("RemoveCOMEvent")]
public void RemoveCOMEvent()
{
  // 撤销注册的 COM 事件处理程序
  acAppCom.BeginFileDrop -=
      new
_DAcadApplicationEvents_BeginFileDropEventHandler(appComBeginFileDrop);
  acAppCom = null;
}

public void appComBeginFileDrop(string strFileName, ref bool bCancel)
```

```
{
  // 显示消息框，提示是否继续插入 DWG 文件
  if (System.Windows.Forms.MessageBox.Show("AutoCAD is about to load " +
strFileName +
                                           "\nDo you want to continue loading this
file?",
                                           "DWG File Dropped",

System.Windows.Forms.MessageBoxButtons.YesNo) ==
                                           System.Windows.Forms.DialogResult.No)
  {
      bCancel = true;
  }
}
```

VBA/ActiveX 代码参考

```
Public WithEvents ACADApp As AcadApplication
Sub Example_AcadApplication_Events()
    ' 初始化公共变量(ACADApp)
    ' 用于拦截 AcadApplication 事件
    '
    ' 先运行这个子程序！
    Set ACADApp = ThisDrawing.Application
End Sub
Private Sub ACADApp_BeginFileDrop(ByVal FileName As String, Cancel As Boolean)
    ' 这个子程序拦截 Application 的一个 BeginFileDrop 事件。
    ' 当把一个图形文件拖拽进 AutoCAD 时该事件被触发。
    '
    ' 使用"Cancel"变量停止加载拖拽的那个文件
    ' 使用"FileName"变量告诉用户拖拽的文件的名称

    If MsgBox("AutoCAD is about to load " & FileName & vbCrLf _
              & "Do you want to continue loading this file?", _
              vbYesNoCancel + vbQuestion) <> vbYes Then
        Cancel = True
    End If
End Sub
```

附： Autodesk.AutoCAD.Interop.dll 中定义的 COM 事件

1、 AcadApplication 事件

```
public interface _DAcadApplicationEvents_Event
{
    // Events
    event _DAcadApplicationEvents_AppActivateEventHandler AppActivate;
    event _DAcadApplicationEvents_AppDeactivateEventHandler AppDeactivate;
    event _DAcadApplicationEvents_ARXLoadedEventHandler ARXLoaded;
    event _DAcadApplicationEvents_ARXUnloadedEventHandler ARXUnloaded;
    event _DAcadApplicationEvents_BeginCommandEventHandler BeginCommand;
    event _DAcadApplicationEvents_BeginFileDropEventHandler BeginFileDrop;
    event _DAcadApplicationEvents_BeginLispEventHandler BeginLisp;
    event _DAcadApplicationEvents_BeginModalEventHandler BeginModal;
    event _DAcadApplicationEvents_BeginOpenEventHandler BeginOpen;
    event _DAcadApplicationEvents_BeginPlotEventHandler BeginPlot;
    event _DAcadApplicationEvents_BeginQuitEventHandler BeginQuit;
    event _DAcadApplicationEvents_BeginSaveEventHandler BeginSave;
    event _DAcadApplicationEvents_EndCommandEventHandler EndCommand;
    event _DAcadApplicationEvents_EndLispEventHandler EndLisp;
    event _DAcadApplicationEvents_EndModalEventHandler EndModal;
    event _DAcadApplicationEvents_EndOpenEventHandler EndOpen;
    event _DAcadApplicationEvents_EndPlotEventHandler EndPlot;
    event _DAcadApplicationEvents_EndSaveEventHandler EndSave;
    event _DAcadApplicationEvents_LispCancelledEventHandler LispCancelled;
    event _DAcadApplicationEvents_NewDrawingEventHandler NewDrawing;
    event _DAcadApplicationEvents_SysVarChangedEventHandler SysVarChanged;
    event _DAcadApplicationEvents_WindowChangedEventHandler WindowChanged;
    event _DAcadApplicationEvents_WindowMovedOrResizedEventHandler WindowMovedOrResized;
}
```

2、 AcadDocument 事件

```
public interface _DAcadDocumentEvents_Event
{
    // Events
    event _DAcadDocumentEvents_ActivateEventHandler Activate;
    event _DAcadDocumentEvents_BeginCloseEventHandler BeginClose;
    event _DAcadDocumentEvents_BeginCommandEventHandler BeginCommand;
    event _DAcadDocumentEvents_BeginDocCloseEventHandler BeginDocClose;
    event _DAcadDocumentEvents_BeginDoubleClickEventHandler BeginDoubleClick;
    event _DAcadDocumentEvents_BeginLispEventHandler BeginLisp;
    event _DAcadDocumentEvents_BeginPlotEventHandler BeginPlot;
    event _DAcadDocumentEvents_BeginRightClickEventHandler BeginRightClick;
    event _DAcadDocumentEvents_BeginSaveEventHandler BeginSave;
    event _DAcadDocumentEvents_BeginShortcutMenuCommandEventHandler BeginShortcutMenuCommand;
    event _DAcadDocumentEvents_BeginShortcutMenuDefaultEventHandler BeginShortcutMenuDefault;
    event _DAcadDocumentEvents_BeginShortcutMenuEditEventHandler BeginShortcutMenuEdit;
    event _DAcadDocumentEvents_BeginShortcutMenuGripEventHandler BeginShortcutMenuGrip;
    event _DAcadDocumentEvents_BeginShortcutMenuOsnapEventHandler BeginShortcutMenuOsnap;
    event _DAcadDocumentEvents_DeactivateEventHandler Deactivate;
    event _DAcadDocumentEvents_EndCommandEventHandler EndCommand;
    event _DAcadDocumentEvents_EndLispEventHandler EndLisp;
    event _DAcadDocumentEvents_EndPlotEventHandler EndPlot;
    event _DAcadDocumentEvents_EndSaveEventHandler EndSave;
    event _DAcadDocumentEvents_EndShortcutMenuEventHandler EndShortcutMenu;
    event _DAcadDocumentEvents_LayoutSwitchedEventHandler LayoutSwitched;
    event _DAcadDocumentEvents_LispCancelledEventHandler LispCancelled;
    event _DAcadDocumentEvents_ObjectAddedEventHandler ObjectAdded;
    event _DAcadDocumentEvents_ObjectErasedEventHandler ObjectErased;
    event _DAcadDocumentEvents_ObjectModifiedEventHandler ObjectModified;
    event _DAcadDocumentEvents_SelectionChangedEventHandler SelectionChanged;
    event _DAcadDocumentEvents_WindowChangedEventHandler WindowChanged;
    event _DAcadDocumentEvents_WindowMovedOrResizedEventHandler WindowMovedOrResized;
}
```

第 8 章 使用 VB.NET 和 C#开发应用程序

许多程序设计任务涉及到的不仅仅是简单地使用 AutoCAD .NET API 对象模型。本章简要概述了如何处理错误以及如何分发你的应用程序等内容。

记住，微软的 VB.NET 文档和 C#文档包含有更多关于这方面主题的内容。

本章主要内容：

- 处理错误
- 发布应用程序

8.1 处理错误

大多数开发环境都提供了默认的错误处理机制。对于 C#和 VB.NET 来说，对错误的默认反应就是显示一条出错信息并终止应用程序。这一行为在应用程序的开发阶段还算合适，但对于用户来说就没有成效了。

在运行过程中只要是可能发生的错误，都应得到处理，而不是留给用户。在进行应用程序设计时，应捕捉所有可能的错误，并确定如何作出响应。有些情况下的错误可以安全地忽略，而有些情况下，为了让应用程序能继续运行，必须用特定的响应来处理错误。

当捕捉到一个错误时，默认错误信息会被禁用，应用程序也不会自动终止。因此，可以显示一个自定义的错误信息，或让应用程序来处理错误。

一般情况下，在请求用户输入时、文件 I/O 操作过程中、访问数据库或对象时，都要进行错误处理。即使你确保文件或对象可用，也还可能会有你想不到的地方引起错误。

注： 本指南中大多数代码示例都没有使用错误处理或错误处理范围很有限。这样可以使例子简单，重点突出。然而，任何编程语言都一样，适当的错误处理是一个强大的应用所必不可少的。

8.1.1 应用程序的错误类型

有三种不同类型的错误，可能会在你的应用程序中遇到，分别是：编译时错误、运行时错误和逻辑错误。

- 编译时错误发生在构建应用程序过程中。这些错误多数都是语法错误，及变量范围和数据类型问题。使用 C# 和 VB.NET 的话，这些类型的错误会被开发环境捕捉到。当你输入了一行有错误的代码时，系统会将该代码行用下划线标出，并且当光标放到下划线文本上时，该代码行存在的问题会显示在一个工具提示框内。编译时错误必须在生成应用程序的.NET 程序集前得到纠正。
- 发现并改正运行时错误有点儿难。运行时错误发生在代码运行过程中，并且常常涉及到用户提供的信息或预期存在的文件。例如，如果应用程序要求用户输入一个图形文件名而用户输入的文件不存在，这时就出现运行时错误。为了有效处理运行时错误，开发人员必须预测可能会发生哪些问题、如何捕获它们，然后编写代码来处理这些情况。
- 逻辑错误是最难发现和改正的。逻辑错误的症状是，程序既没有编译时错误，也没有运行时错误，但运行结果还是不正确。这种情况也就是程序员所说的缺陷或 bug。这种缺陷有的很容易找到，有的很难追查。

关于如何查找和纠正错误的内容可以在你的开发环境文档中找到。AutoCAD 特有的错误属于运行时错误范畴，这类错误在本章内讨论。

8.1.2 捕捉运行时错误

VB.NET 语言和 C#语言支持通用的及语言专有的运行时错误处理方式。两种语言均支持捕捉错误的 `Try` 语句； VB.NET 语言还拥有 `On Error` 语句，C#只有 `Goto` 语句与之相对应（不建议使用）。

8.1.2.1 使用 Try 语句

使用 VB.NET 和 C#语言，可以用 Try 语句捕获运行时错误。从字面上看，这条语句为系统设置了一个陷阱。当 Try 语句内出现错误时，程序执行会绕过系统默认的错误处理，而被重定向到 Catch 子句上。

`Try` 语句有 3 种格式：

- `Try-Catch`
- `Try-Finally`
- `Try-Catch-Finally`

Try-Catch 语句

当需要对错误作出响应时使用 Try-Catch 语句。该语句捕获到错误后，不显示默认的错误消息也不终止应用程序，而是转到 try 语句的 catch 子句。

catch 子句可以包含一个 Exception 类型的可选参数，该 Exception 对象含有遇到的错误的相关信息。如果遇到的错误得不到解决，应该显示一条自定义错误消息，并友好地退出程序。

关于 Exception 对象的内容，参见下一节 ***使用 Exception 对象*** 。

Try-Finally 语句

当不需要提供具体的错误处理时使用 Try-Finally 语句。该语句捕获到错误后，显示默认的错误消息但不终止应用程序。遇到错误时，在默认消息框中点击继续运行后，程序的执行就转到 Try 语句的 Finally 子句。当应用程序处于开发和测试差错阶段时，最好使用 Try-Finally 语句。

Try-Catch-Finally 语句

Try-Catch-Finally 语句是 Try-Catch 语句和 Try-Finally 语句的组合。该语句捕获到错误后，不显示默认的错误消息也不终止应用程序，程序先转到 Try 语句的 Catch 子句，执行完 Catch 子句后，程序转到 Finally 子句，给程序最后一次机会选择是继续运行还是友好退出。

使用与不使用 Try-Catch-Finally 语句处理错误测试

下面的示例代码试图打开 C:盘上名为“Drawing123”的文件。如果文件不存在，会抛出一个 eFileNotFound 错误。第一个命令没有捕捉 ReadDwgFile()方法抛出的错误，因此在 AutoCAD 运行该命令时显示默认的消息框。第二个命令使用 Try-Catch-Finally 语句捕捉了错误。

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

<CommandMethod("NoErrorHandler")> _
Public Sub NoErrorHandler()
  '' 创建一个没有文档窗口的新数据库
  Using acDb As Database = New Database(False, True)
      '' 读取 C:盘上的图形文件“drawing123.dwg”
      '' 如果文件“drawing123.dwg”不存在，抛出 eFileNotFound 异常
      '' 异常被抛出，程序停止
      acDb.ReadDwgFile("c:\Drawing123.dwg", _
                       System.IO.FileShare.None, False, "")
  End Using
```

```
  '' ReadDwgFile 引发的异常没有被处理，故显示信息语句没有执行
  Application.ShowAlertDialog("End of command reached")
End Sub

<CommandMethod("ErrorTryCatchFinally")> _
Public Sub ErrorTryCatchFinally()
  '' 创建一个没有文档窗口的新数据库
  Using acDb As Database = New Database(False, True)
      Try
          '' 读取 C:盘上的图形文件“drawing123.dwg”
          '' 如果文件“drawing123.dwg”不存在，抛出 eFileNotFound 异常
          '' 并由 catch 语句处理错误
          acDb.ReadDwgFile("c:\Drawing123.dwg", _
                           System.IO.FileShare.None, False, "")
      Catch Ex As Autodesk.AutoCAD.Runtime.Exception
          Application.ShowAlertDialog("The following exception was caught:" & _
                                      vbLf & Ex.Message)
      Finally
          '' 处理完 ReadDwgFile 引发的异常后显示信息
          Application.ShowAlertDialog("End of command reached")
      End Try
  End Using
End Sub
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[CommandMethod("NoErrorHandler")]
public void NoErrorHandler()
{
  // 创建一个没有文档窗口的新数据库
  using (Database acDb = new Database(false, true))
  {
      // 读取 C:盘上的图形文件“drawing123.dwg”
      // 如果文件“drawing123.dwg”不存在，抛出 eFileNotFound 异常
      // 异常被抛出，程序停止
      acDb.ReadDwgFile("c:\\Drawing123.dwg",
                       System.IO.FileShare.None, false, "");

  }
```

```
  // ReadDwgFile 引发的异常没有被处理，故显示信息语句没有执行
  Application.ShowAlertDialog("End of command reached");
}

[CommandMethod("ErrorTryCatchFinally")]
public void ErrorTryCatchFinally()
{
  // 创建一个没有文档窗口的新数据库
  using (Database acDb = new Database(false, true))
  {
      try
      {
          // 读取 C:盘上的图形文件“drawing123.dwg”
          // 如果文件“drawing123.dwg”不存在，抛出 eFileNotFound 异常
          // 并由 catch 语句处理错误
          acDb.ReadDwgFile("c:\\Drawing123.dwg",
                           System.IO.FileShare.None, false, "");
      }
      catch (Autodesk.AutoCAD.Runtime.Exception Ex)
      {
          Application.ShowAlertDialog("The following exception was caught:\n" +
                                      Ex.Message);
      }
      finally
      {
          // 处理完 ReadDwgFile 引发的异常后显示信息
          Application.ShowAlertDialog("End of command reached");
      }
  }
}
```

8.1.2.2 使用 Exception 对象

Exception 对象用来获取 Try 语句的 Catch 子句捕捉到的错误的相关信息。用于 AutoCAD .NET API 的 Exception 对象是基于 Microsoft® .NET Framework 的 Exception 对象定义的。我们可以使用 Exception 对象的属性来确定 Catch 子句捕捉到的错误。用于获取错误的相关信息的 Exception 对象成员属性有：

- ErrorStatus - 返回异常的 Autodesk.AutoCAD.Runtime.ErrorStatus 枚举值。
- Message - 返回解释异常的文本消息。
- Source - 返回引起异常的程序或对象。
- StackTrace - 返回异常发生时表示调用栈结构的字符串。

- TargetSite - 返回抛出异常的方法。

8.1.2.3 On Error 语句 (VB.NET)

如果使用的是 VB.NET 语言，运行时错误还可以通过 `On Error` 语句捕捉。该语句在应用程序中设置一个普通陷阱，当错误发生时，该语句自动将运行跳转到你特别编写的错误处理程序，系统默认的错误处理被绕过。

`On Error` 语句有三种格式：

- `On Error Resume Next`
- `On Error GoTo Label`
- `On Error GoTo 0`

注： `Try` 语句和 `On Error` 语句不能在一个过程中同时使用。

On Error Resume Next 语句

想忽略错误时使用 `On Error Resume Next` 语句。该语句捕获到错误后，不显示默认的错误消息也不终止应用程序，而是转到下一行代码继续运行。

例如，想要编写一个程序遍历模型空间并修改每个实体的颜色，我们知道，当试图修改锁定图层上的实体的颜色时，AutoCAD 将抛出一个错误。这时不用中止程序，只需跳过锁定图层上的实体继续处理剩下的实体即可。`On Error Resume Next` 语句刚好能让你做到这点。

On Error GoTo Label 语句

当需要编写一个显式错误处理程序时使用 `On Error GoTo Label` 语句。该语句捕获到错误后，不显示默认的错误消息也不终止应用程序，而是跳转到代码的特定位置，然后你的代码可以用适合应用程序的方式响应该错误。例如，你可以跳转到以行号 `0` 标识的程序开始处，或跳转到像 ErrNoFileFound 这样的命名标签处。命名标签用下面的语法定义：

```
HandlerName:
```

使用 Err 对象

Exception 对象用于 `Try` 语句，而 Err 对象用于 `On Error` 语句，用来提供 `On Error` 语句捕捉到的错误的类型等信息。 Err 对象有多个成员属性：Number、Description、Source、HelpFile、HelpContext 和 LastDLLError 等。Err 对象的属性信息由最近刚发生的错误填写。其中的 Number 属性和 Description 属性最重要，Number 属性包含与错误关联的唯一的错误码，Description 属性包含可以正常显示的错误消息。

可以在错误处理程序中将错误的 Number 属性与期望的值作比较，以帮助确定发生的错误的性质。知道了正在处理的是什么样的错误，就可以采取适当的措施了。

8.1.2.4 VBA/VB 与.NET 错误处理比较

VBA 或 VB 语言的错误处理通过使用 `On Error` 语句来实现。虽然在 VB.NET 中使用 `On Error` 语句没有任何问题，但还是推荐使用 `Try` 语句来代替。相比 `On Error Resume Next` 语句和 `On Error GoTo Label` 语句，`Try` 语句更灵活。

使用 `On Error Resume Next` 语句和 `On Error GoTo Label` 语句的代码可以改写成使用 `Try-Catch` 语句。下面演示如何将 `On Error GoTo Label` 语句改写成 `Try-Catch` 语句。

On Error - VBA

```
Sub ColorEntities()
    On Error GoTo MyErrorHandler

    Dim entry As Object
    For Each entry In ThisDrawing.ModelSpace
        entry.color = acRed
    Next entry

    ' 重要！在错误处理程序前退出子程序
  Exit Sub

MyErrorHandler:
    MsgBox entry.EntityName + " is on a locked layer." + _
                              " The handle is: " + entry.Handle
  Resume Next
End Sub
```

Try-Catch - VB.NET

```
<CommandMethod("ColorEntities")> _
Public Sub ColorEntities()
  '' 获取当前文档和数据库，启动事务
  Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim acCurDb As Database = acDoc.Database

  Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
      '' 以读模式打开块表
      Dim acBlkTbl As BlockTable
      acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                   OpenMode.ForRead)

      '' 以读模式打开块表记录 Model 空间
      Dim acBlkTblRec As BlockTableRecord
      acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
```

```
                                        OpenMode.ForRead)

      Dim acObjId As ObjectId

      '' 遍历 Model 空间的每个对象
      For Each acObjId In acBlkTblRec
          Try
              Dim acEnt As Entity
              acEnt = acTrans.GetObject(acObjId, OpenMode.ForWrite)

              acEnt.ColorIndex = 1
          Catch
              Application.ShowAlertDialog(acObjId.ObjectClass.DxfName & _
                          " is on a locked layer." & _
                          " The handle is: " & acObjId.Handle.ToString())
          End Try
      Next

      acTrans.Commit()
  End Using
End Sub
```

8.1.3 响应用户输入错误

用户输入方法提供了一定数量的内在错误捕捉功能，这些方法要求用户输入特定类型的数据。如果用户试图输入其他类型的数据，AutoCAD 会拒绝该输入并重新提示用户。通过使用 PromptXXXOption 对象及适合的 getXXX 方法或 SelectXXX 方法提供了对用户输入的额外的控制，但在继续执行之前，还是可以引入必须经过验证的其他附加条件。关于检查用户输入状态的示例，参见（§2.8 *提示用户输入*）。

8.2 发布应用程序

可以两种编译类型来生成你的.NET 应用程序：调试 debug 和发行 release.

- **Debug 编译** - 包含调试信息，因而生成的.NET 程序集文件比发行版的大。
- **Release 编译** - 不包含调试信息。

你必须选择应用程序使用的编译类型，两种编译类型生成的.NET 程序集都可以加载到 AutoCAD。通常在开发和测试应用程序时使用 Debug 编译，当将你的应用程序分发给公司内或公司外的许多计算机上使用时选择 Release 编译。

生成 Release 版.NET 程序集

下列步骤解释如何生成一个 Release 版的.NET 程序集。

1. 运行 Microsoft Visual Studio，打开要编译的项目。
2. 依次点击 **生成**菜单 ➤ **配置管理器**。
3. 进入**配置管理器**对话框，单击**活动解决方案配置**下拉列表，选择 Release。
4. 单击**关闭**。
5. 回到 Microsoft Visual Studio，单击**生成**菜单 ➤ **生成解决方案**。

加载.NET 程序集

确定了你的.NET 程序集的编译类型后，下一步需要确定如何将你的程序集加载到 AutoCAD 中。可以手动加载.NET 程序集文件，或按需加载。

- 手动加载 - 在命令提示行或 AutoLISP 文件内使用 NETLOAD 命令。更多内容参见（*附录 A.6 加载程序集到 AutoCAD*）。
- 按需加载 - 定义一个指向 AutoCAD 启动时要加载的应用程序的注册表键，该键必须放在所使用指定版本 AutoCAD 的 Application 键的下面。

  应用程序的键可以包含下列子键：

  DESCRIPTION

  .NET 程序集的描述，可选。

  LOADCTRLS

  控制何时、如何加载.NET 程序集。

  - 1 - 检测到代理对象时加载应用程序
  - 2 - 启动时加载应用程序
  - 4 - 启动一个命令时加载应用程序
  - 8 - 用户或其它应用程序请求时加载应用程序

- 16 - 不加载应用程序
- 32 - 透明加载应用程序

LOADER

指定加载哪个.NET 程序集文件（含完整路径和文件名）。

MANAGED

指定要加载的文件是.NET 程序集还是 ObjectARX 文件。键值为 1 表示加载的是.NET 程序集文件。

按需加载.NET 应用程序

下面的代码示例演示了如何创建和移除 AutoCAD 启动时加载.NET 程序集文件所需的注册表键。使用 RegisterMyApp 命令时，创建所需的注册表键，下次 AutoCAD 启动时会自动加载应用程序。 UnregisterMyApp 命令从注册表中移除按需加载信息，这样，下次 AutoCAD 启动时就不会加载应用程序了。

VB.NET

```
Imports Microsoft.Win32
Imports System.Reflection

Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices

<CommandMethod("RegisterMyApp")> _
Public Sub RegisterMyApp()
  '' 获取 AutoCAD 的 Applications 键

  '' 2012 版使用 RegistryProductRootKey 属性
  '' 2014 版使用 MachineRegistryProductRootKey 属性
  Dim sProdKey As String = HostApplicationServices.Current.RegistryProductRootKey
  Dim sAppName As String = "MyApp"

  Dim regAcadProdKey As RegistryKey = Registry.CurrentUser.OpenSubKey(sProdKey)
  Dim regAcadAppKey As RegistryKey = regAcadProdKey.OpenSubKey("Applications",
True)

  '' 检查”MyApp”键是否存在
  Dim subKeys() As String = regAcadAppKey.GetSubKeyNames()
  For Each sSubKey As String In subKeys
      '' 如果应用程序已经注册，就退出
      If (sSubKey.Equals(sAppName)) Then
          regAcadAppKey.Close()
```

```
            Exit Sub
        End If
    Next

    '' 获取本模块的位置（注册本程序自己）
    Dim sAssemblyPath As String = Assembly.GetExecutingAssembly().Location

    '' 注册应用程序
    Dim regAppAddInKey As RegistryKey = regAcadAppKey.CreateSubKey(sAppName)
    regAppAddInKey.SetValue("DESCRIPTION", sAppName, RegistryValueKind.String)
    regAppAddInKey.SetValue("LOADCTRLS", 14, RegistryValueKind.DWord)
    regAppAddInKey.SetValue("LOADER", sAssemblyPath, RegistryValueKind.String)
    regAppAddInKey.SetValue("MANAGED", 1, RegistryValueKind.DWord)

    regAcadAppKey.Close()
End Sub

<CommandMethod("UnregisterMyApp")> _
Public Sub UnregisterMyApp()
    '' 获取 AutoCAD 的 Applications 键
    Dim sProdKey As String = HostApplicationServices.Current.RegistryProductRootKey
    Dim sAppName As String = "MyApp"

    Dim regAcadProdKey As RegistryKey = Registry.CurrentUser.OpenSubKey(sProdKey)
    Dim regAcadAppKey As RegistryKey = regAcadProdKey.OpenSubKey("Applications",
True)

    '' 删除应用程序的注册表键
    regAcadAppKey.DeleteSubKeyTree(sAppName)
    regAcadAppKey.Close()
End Sub
```

C#

```
using Microsoft.Win32;
using System.Reflection;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;

[CommandMethod("RegisterMyApp")]
public void RegisterMyApp()
{
    // 获取 AutoCAD 的 Applications 键
```

```
  // 2012 版使用 RegistryProductRootKey 属性
  // 2014 版使用 MachineRegistryProductRootKey 属性
  string sProdKey = HostApplicationServices.Current.RegistryProductRootKey;
  string sAppName = "MyApp";

  RegistryKey regAcadProdKey = Registry.CurrentUser.OpenSubKey(sProdKey);
  RegistryKey regAcadAppKey = regAcadProdKey.OpenSubKey("Applications", true);

  // 检查”MyApp”键是否存在
  string[] subKeys = regAcadAppKey.GetSubKeyNames();
  foreach (string subKey in subKeys)
  {
      // 如果应用程序已经注册，就退出
      if (subKey.Equals(sAppName))
      {
          regAcadAppKey.Close();
          return;
      }
  }

  // 获取本模块的位置（注册本程序自己）
  string sAssemblyPath = Assembly.GetExecutingAssembly().Location;

  // 注册应用程序
  RegistryKey regAppAddInKey = regAcadAppKey.CreateSubKey(sAppName);
  regAppAddInKey.SetValue("DESCRIPTION", sAppName, RegistryValueKind.String);
  regAppAddInKey.SetValue("LOADCTRLS", 14, RegistryValueKind.DWord);
  regAppAddInKey.SetValue("LOADER", sAssemblyPath, RegistryValueKind.String);
  regAppAddInKey.SetValue("MANAGED", 1, RegistryValueKind.DWord);

  regAcadAppKey.Close();
}

[CommandMethod("UnregisterMyApp")]
public void UnregisterMyApp()
{
  // 获取 AutoCAD 的 Applications 键
  string sProdKey = HostApplicationServices.Current.RegistryProductRootKey;
  string sAppName = "MyApp";

  RegistryKey regAcadProdKey = Registry.CurrentUser.OpenSubKey(sProdKey);
  RegistryKey regAcadAppKey = regAcadProdKey.OpenSubKey("Applications", true);

  // 删除应用程序的注册表键
```

```
  regAcadAppKey.DeleteSubKeyTree(sAppName);
  regAcadAppKey.Close();
}
```

RegisterMyApp 命令生成的注册表信息如下：

```
[HKEY_CURRENT_USER\Software\Autodesk\AutoCAD\R19.1\ACAD-D001:804\Applications\MyApp]
"DESCRIPTION"="MyApp"
"LOADCTRLS"=dword:0000000e
"LOADER"="C:\\Users\\SunnyPC\\Documents\\Visual Studio
2010\\Projects\\chapter_08\\chapter_08\\bin\\Debug\\chapter_08.dll"
"MANAGED"=dword:00000001
```

附录 A Microsoft Visual Studio 使用入门

本章介绍 AutoCAD® .NET API 和 Microsoft® Visual Studio®开发环境的一些基本知识，同时还将学会 AutoCAD 中的 NETLOAD 命令，以及贯穿本指南的其他一些术语。

主要内容：

- 了解 Microsoft Visual Studio 项目
- 定义项目组件
- 查看项目信息
- 使用 Microsoft Visual Studio 项目
- 编辑现有项目或解决方案
- 加载程序集到 AutoCAD
- 访问和查找引用库（对象浏览器）
- 练习：创建你的第一个项目
- 相关 AutoCAD 命令和术语
- 更多内容

A.1 理解 Microsoft Visual Studio 项目

Microsoft Visual Studio 创建的项目文件不是专为 AutoCAD 的，不过确实包含一些你应该熟悉的特定的项目设置，以便创建能够加载到 AutoCAD 的 DLL 程序集。本指南讲解如何使用或不使用 AutoCAD 2012 .NET Wizard 向导来创建 Visual Basic®(VB) .NET 项目和 Visual C#® 项目。

项目是代码文件和资源文件的集合；类模块、WPF 窗体和 Windows 窗体一起共同实现所需的功能。创建项目时，会同时创建包含该项目的解决方案。解决方案用来引用一个或多个项目。通常情况下我们不需要直接使用解决方案，除非你要引用其他项目提供的功能。

一个多项目解决方案的例子是，你有一个项目，又新建一个项目，现有项目中包含新建项目要用到的一组方法、函数和类。这时，你可以直接在现有项目所在的解决方案下创建新项目，或将现有项目添加到新建项目所在的解决方案下。

解决方案可以包含 VB.NET 项目和 C# 项目的任意组合。这样在单个解决方案中使用不同类型的项目有一个好处，就是允许每个开发人员可以根据自己的喜好选择编程语言。

项目文件和解决方案文件使用下列扩展名：

- VBPROJ - Microsoft Visual Basic (VB) .NET 项目文件
- CSPROJ - Microsoft Visual C# 项目文件
- SLN - Microsoft Visual Studio 解决方案文件

A.2 定义项目组件

每个项目可以包含许多不同的组件，包括类模块、窗体、引用及资源等。

类模块

类模块（Class）是包含公共及私有过程很函数的组件。类模块用来定义用户命名空间，在命名空间里定义程序用到的过程，以及实现自定义命令和 AutoLISP 函数的结构等。

窗体

窗体组件包含项目中使用的自定义对话框。除非你创建的是独立的应用程序，否则项目中的窗体应通过子过程或函数加载来显示。组成项目的窗体类型有 Windows 窗体、WPF 窗体和用户控件等。本指南中，窗体是指 Windows 窗体或 WPF 窗体。

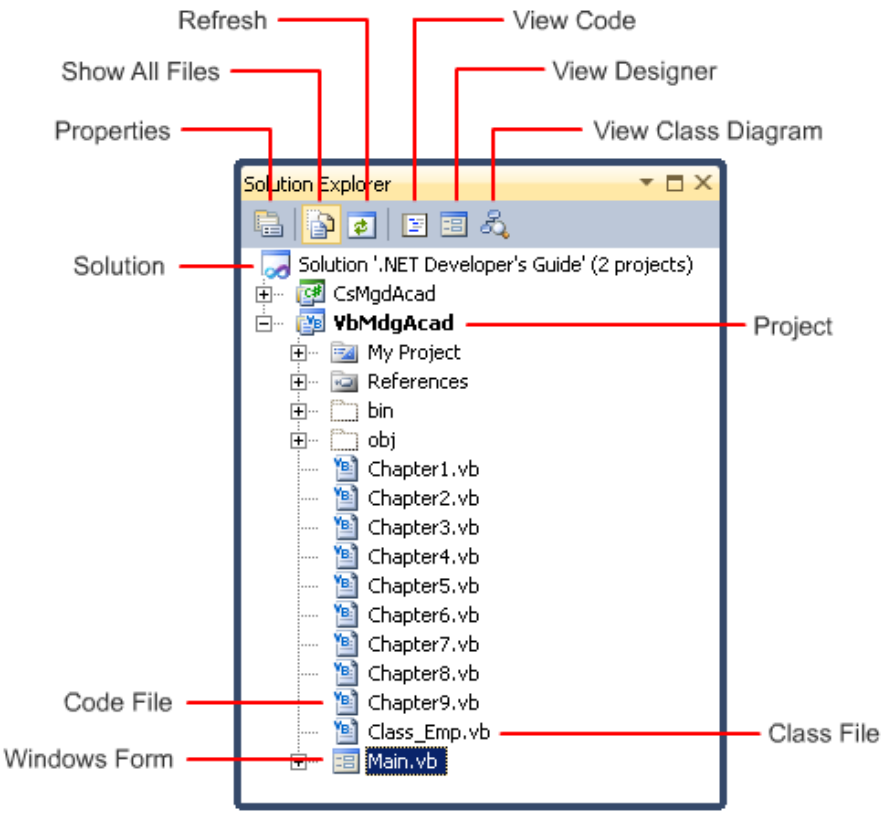
引用

引用表示你的项目用到的项目或库。

A.3 查看项目信息

解决方案资源管理器（Solution Explorer）用来显示当前解决方案和打开的所有项目的列表，同时还显示每个项目包含的代码文件、类模块文件和窗体模块文件，以及每个项目用到的对外部库的引用。

解决方案资源管理器拥有自己的工具条，可以用来打开各种不同的项目组件，以便编辑。使用查看代码（View Code）按钮可以在编辑器窗口打开选中的模块的代码，使用查看设计器（View Designer)按钮可以打开选中的窗体的可视化设计器窗口，使用查看类图(View Class Diagram）按钮可以添加或打开类关系图，以便查看项目的类结构。



默认情况下解决方案资源管理器是可见的。如果不可见，单击**视图**菜单 ➤ **解决方案资源管理器** 或按组合键 Ctrl+Alt+L。

A.4 使用 Microsoft Visual Studio 项目

打开的解决方案或项目可以从解决方案资源管理器窗口查看和访问，解决方案资源管理器允许你给解决方案添加新项目、给项目添加新组建、从当前解决方案卸载一个项目、修改打开的解决方案或项目的属性，及给项目添加引用等等。针对当前解决方案或项目的所有可用任务均可以通过解决方案资源管理器的右键菜单进行访问。

注： 如果你使用的是 Microsoft Visual Studio 2008，应在创建新项目前安装 Service Pack 1。

操作步骤

启动 Microsoft Visual Studio

按下列步骤操作：

- 点击**开始**菜单，单击**所有程序** ➤ Microsoft Visual Studio 2010 ➤ Microsoft Visual Studio 2010

A.4.1 创建新项目

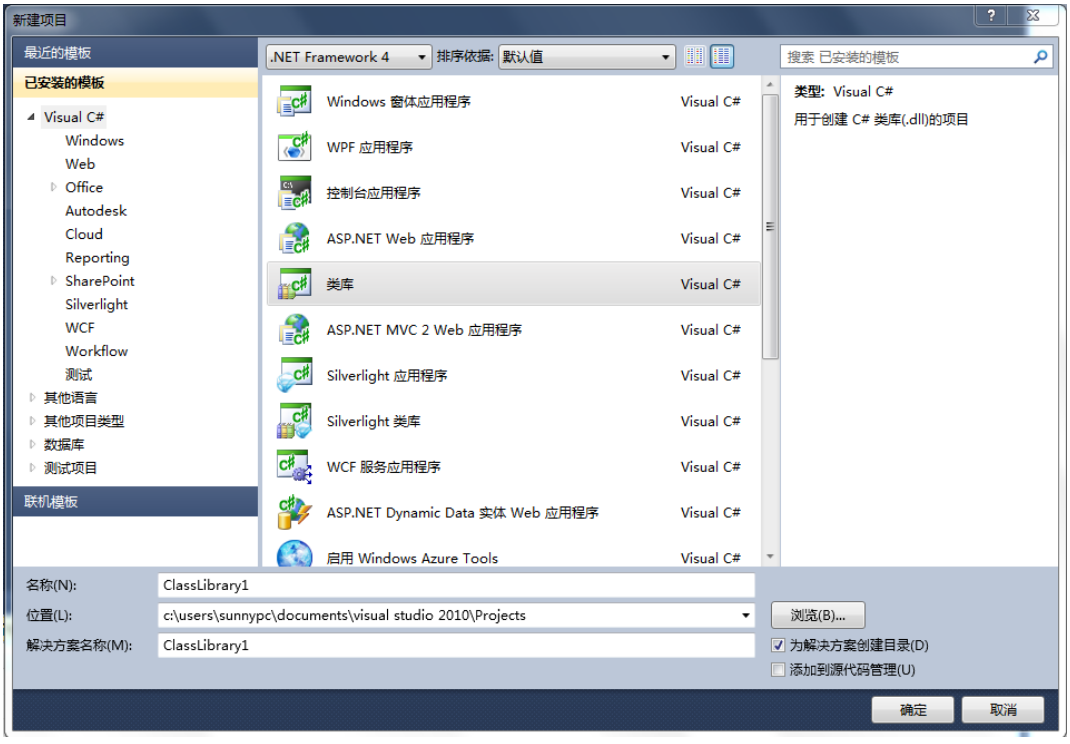
新项目基于项目模板来创建。当要创建一个新项目以生成加载到 AutoCAD 的 DLL 程序集时，可以使用 Microsoft Visual Studio 自带的类库模板，或使用和 **AutoCAD 2012 .NET 向导** 一起安装的 AutoCAD 托管项目模板。这两种模板对 VB.NET 语言和 C#语言都可用。

创建了新项目后，需要在项目中引用组成 AutoCAD.NET API 的文件。关于引用 AutoCAD.NET API 相关文件以及安装 AutoCAD 2012 .NET 向导的内容，请参见（§0.3 *AutoCAD .NET API 的组件*）。

操作步骤

使用标准模板创建新项目

1. 进入 Microsoft Visual Studio，单击**文件**菜单 ➤ **新建** ➤ **项目**。
2. 在打开的**新建项目**对话框中，左侧的**已安装模板**树内，展开**其他语言** ➤ Visual Basic 或 **Visual C#**，单击 **Windows**。
3. 在中间的模板窗格内选中**类库**。



4. 在**名称**（Name）输入栏内输入新项目的名称。
5. 在**位置**（Location）输入栏内为新项目输入或浏览选择一个文件夹。

6. 在**解决方案名称**输入栏内给新解决方案输入一个名称，新项目将被添加到这个解决方案中。

7. （可选）选中**为解决方案创建目录**复选框 可以在创建新解决方案和项目前先创建一个子目录； 选中**添加到源代码管理**复选框可以将解决方案和项目添加到源码管理数据库。

8. 单击**确定**。

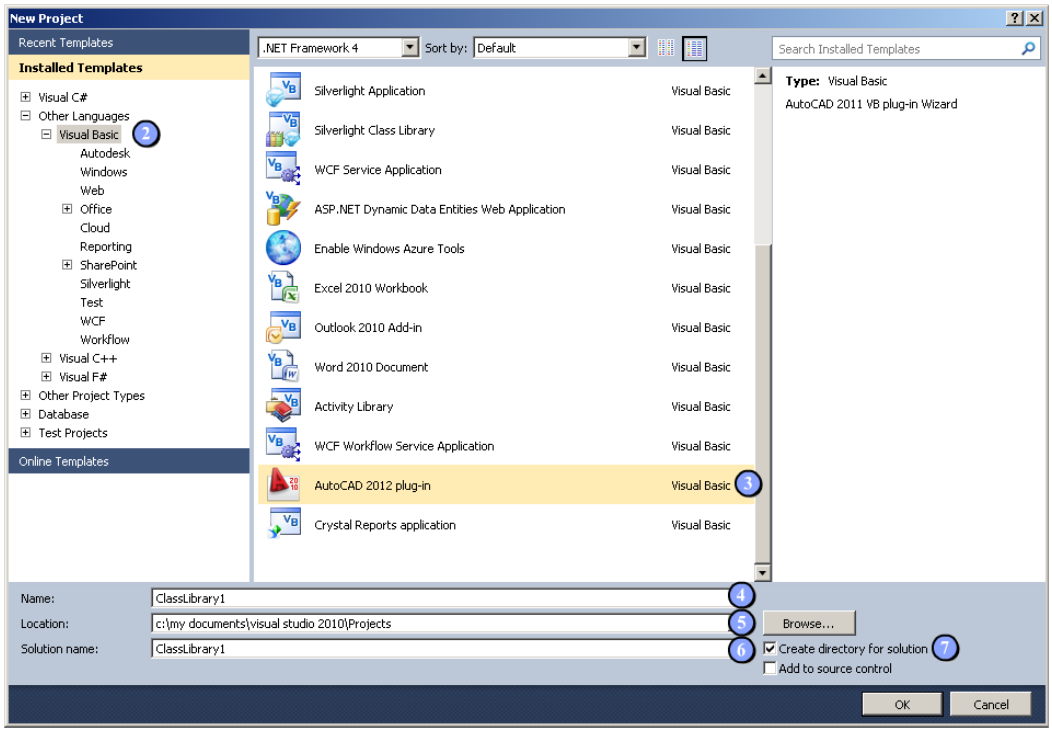
使用 AutoCAD 托管模板创建新项目

使用 AutoCAD 托管模板之前，必须先下载安装 AutoCAD 2012 ObjectARX SDK 开发包。关于安装 AutoCAD 托管项目模板的内容，详见（§0.3 *AutoCAD .NET API 的组件*）。

1. 进入 Microsoft Visual Studio，单击**文件**菜单 ➤ **新建** ➤ **项目**。

2. 在**新建项目**对话框中，从**已安装模板**目录树里展开**其他语言** ➤ **Visual Basic** 或 **Visual C#** 并单击 **Windows**。

3. 在中间的模板窗格内选择 **AutoCAD 2012 Plug-in**。



4. 在下面的**名称（Name）**输入栏内输入新项目的名称。

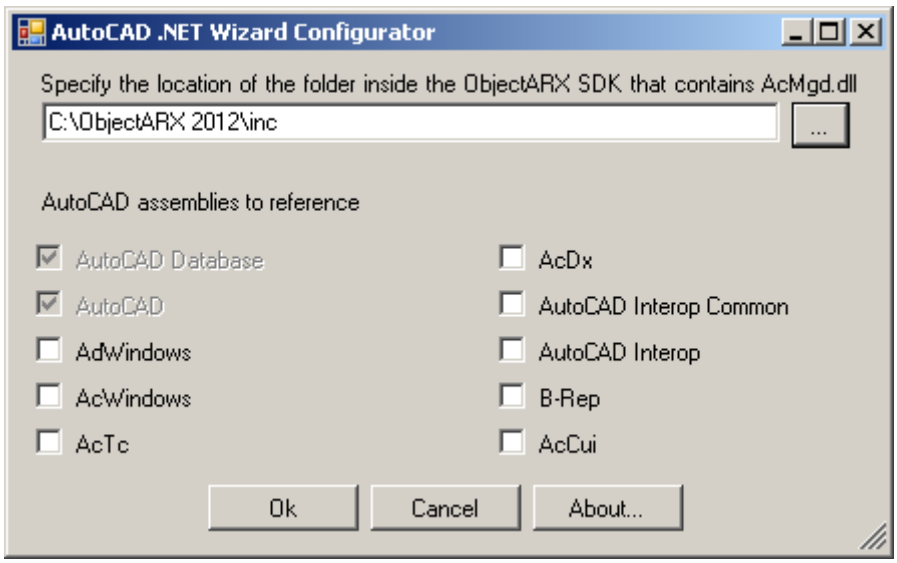
5. 在**位置（Location）**输入栏内为新项目输入或浏览选择一个文件夹。

6. 在**解决方案名称**输入栏内给新解决方案输入一个名称，新项目将被添加到这个解决方案中。

7. （可选）选中**为解决方案创建目录**复选框 可以在创建新解决方案和项目前先创建一个子目录； 选中**添加到源代码管理**复选框可以将解决方案和项目添加到源码管理数据库。

8. 单击**确定**。

9. 在打开的 AutoCAD .NET Wizard Configurator 对话框中，指定项目需要引用的库。



10. 单击 OK。

A.4.2 打开现有项目或解决方案

在 Microsoft Visual Studio 中打开一个项目或解决方案时，代码编辑器窗口和 Windows 窗体设计器窗口会按照项目最后一次保存时的状态打开。打开项目或解决方案后，可以使用解决方案资源管理器浏览项目中的文件。关于处理打开的项目或解决方案的更多内容，参见（§A.5 *编译现有项目或解决方案*）。

操作步骤

打开现有项目或解决方案文件

1. 进入 Microsoft Visual Studio，单击**文件**菜单 ➤ **打开** ➤ **项目/解决方案**。

2. 在**打开项目**对话框中，浏览并选择想要打开的项目文件。

   打开项目对话框允许打开各种各样的项目文件，包括 VB.NET 和 C#项目、Microsoft Visual Studio .NET 解决方案、Microsoft Visual C++项目，甚至传统的 Microsoft VB6 项目。使用类型下拉列表中的对象可以控制你在**打开项目**对话框的文件选择区域所看到的项目或解决方案的类型。

注：你不能用 Microsoft Visual Studio 打开一个保存为 DVB 文件的 VBA 项目。如果你要从 VBA 迁移到 VB.NET 或 C#，可以从 AutoCAD 的 VBA 开发环境将你的代码复制到 Microsoft Visual Studio 中已打开的代码窗口内，并进行必要的修改。

3. 单击**打开**。

A.4.3 保存项目或解决方案

Microsoft Visual Studio 是一个基于上下文的开发环境，这意味着当前选择的项目决定了在开发环境中可用的功能。控制保存哪些更改了的文件以及如何保存，是由解决方案资源管理器来确定的。

从解决方案资源管理器窗口选中一个项目或解决方案后，可以使用当前名称保存它，或执行**另存为**创建一个备份。通过在选择项目前按住 Ctrl 键可以一次从解决方案资源管理器选择一个或多个项。大多数情况下，选择同时保存所有修改了的项。

操作步骤

保存项目或解决方案

1. 进入 Microsoft Visual Studio，从**解决方案资源管理器**窗口选中要保存的项目或解决方案。
2. 单击**文件**菜单 ➤ **保存*<项目或解决方案名>*** 或***<项目或解决方案名>*另存为**。如果选择***<项目或解决方案名>*另存为**，会显示**另存文件为**对话框。
3. 如果显示了**另存文件为**对话框，浏览到一个存储位置为项目或解决方案创建一个新拷贝，并输入新名称，单击**保存**。

保存所有已修改项

- 进入 Microsoft Visual Studio，单击**文件**菜单 ➤ **全部保存**。

A.4.4 在一个解决方案中使用多个项目

一个解决方案可以包含多个项目。使用多个项目与使用单个项目没有多大区别。可以使用解决方案资源管理器浏览当前解决方案中的各个项目。

添加一个项目到解决方案

你可能想往解决方案中添加一个项目以便在项目间复制代码，你可能想要引用一个项目中的子程序、函数和类，并将它们用到别的项目中去。通过在一个解决方案中添加多个项目，允许你把一个项目当作通用工具使用，这样，你就可以有多个项目可用。

从解决方案中卸载一个项目

当不再需要时，可以从解决方案中卸载项目。如果你不再想让一个项目随解决方案一起加载，可以卸载该项目，然后引用该项目已编译好的 DLL 程序集来代替。使用编译好的 DLL 文件可以帮助避免意外修改源代码。

操作步骤

添加一个项目到解决方案中

进入 Microsoft Visual Studio，执行下列操作之一：

- 单击**文件**菜单 ➤ **添加** ➤ **新建项目**来创建一个新项目并添加到当前解决方案中。关于创建新项目的内容，参见（§A.4.1 *创建新项目*）。
- 单击**文件**菜单 ➤ **添加** ➤ **现有项目**，显示**添加现有项目**对话框并添加一个现有项目到当前解决方案。在**添加现有项目**对话框里，浏览并选择要添加打项目。

从解决方案中卸载一个项目

进入 Microsoft Visual Studio，在**解决方案资源管理器**窗口里，右键单击想要卸载的项目并单击**卸载项目**。

A.5 编辑现有项目或解决方案

在 Microsoft Visual Studio 里打开项目或解决方案后，就可以在开发环境中编辑其中的项目、类模块、窗体及引用，还可以在开发环境里调试和运行项目。

A.5.1 添加新建项

可以给项目添加诸如类模块和 Windows 窗体这样的新建项。更新新建项的属性（如名称）以及编写相应的代码是你的责任。命名新建项时，记住别的开发人员可能会在以后的应用程序中使用你的项目。因此，应确保遵循你所在企业制定的命名约定或微软公司制定的标准。

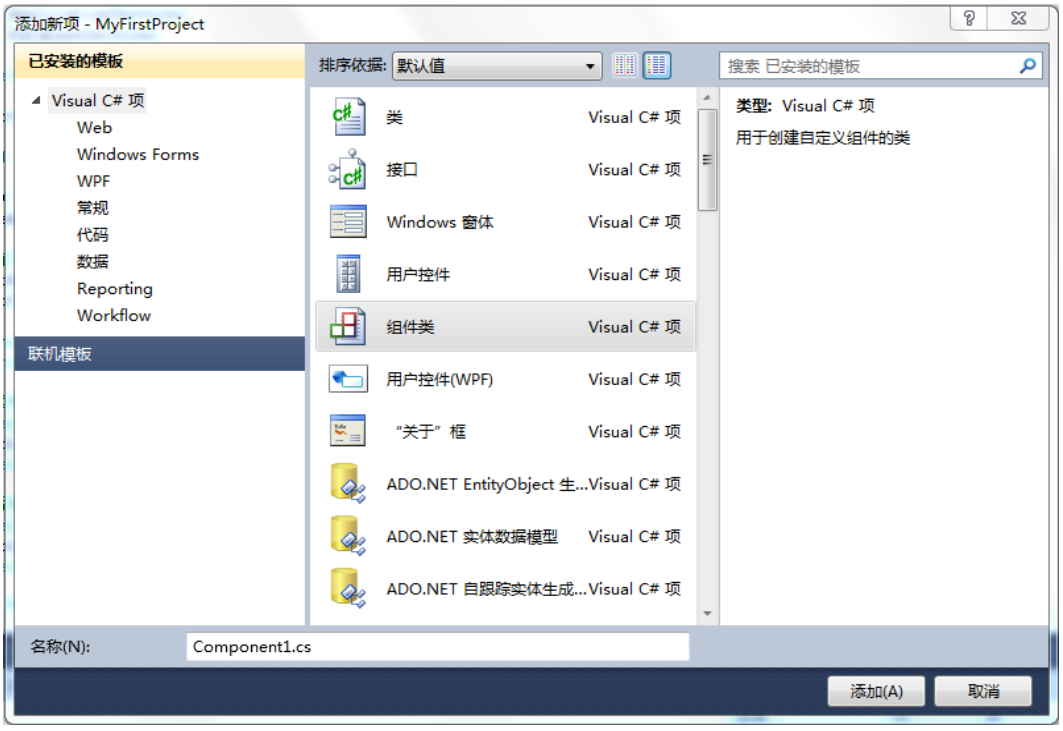
操作步骤

给项目添加新建项

1. 进入 Microsoft Visual Studio，在**解决方案资源管理器**窗口里右键单击要添加新建项的项目，单击**添加** ➤ **新建项**。

   除了单击**新建项**外，还可以从该右键菜单中单击 Windows 窗体、模块（仅限 VB.NET 项目）或类来为项目添加这些类型的项。

2. 进入**添加新项 – *<某项目>***对话框，选择代表你要添加的项的模板。
   - **代码文件** – 代表一个代码模块
   - **Windows 窗体** – 代表一个窗体（或对话框）
   - **类** – 代表一个类模块



3. 为新建项输入一个名称并单击**添加**。

A.5.2 导入现有项

导入允许将其他项目中已编辑好的现有项添加到你的项目中来。可以导入代码、类模块及窗体。具有 VB 扩展名的文件只能导入到 VB.NET 项目，具有 CS 扩展名的文件只能导入到 C#项目。以 CD 文件存储的类图两种类型的项目都可以导入。

导入一个项时，导入文件的一个拷贝或链接会被添加到你的项目里。当添加的是拷贝时，原文件会在原处保持不变，这样对在多个项目中共享通用代码有利。

如果导入的项与项目中现有项重名，会提示你是否用导入项替换项目中的现有项。

导入的组件会添加到你的项目中，并出现在解决方案资源管理器窗口里。要想编辑该项的属性，在解决方案资源管理器窗口选中该项并在属性窗口修改其属性即可。

操作步骤

将现有项导入到项目里

1. 进入 Microsoft Visual Studio，在**解决方案资源管理器**窗口内，右键单击你想要添加现有项的项目，然后单击**添加** ➤ **现有项**。
2. 在**添加现有项 - *〈项目〉*** 对话框里，浏览选择包含你要添加的项的文件。
3. 单击**添加**创建所选文件的一个拷贝，或单击**添加**按钮右侧的向下箭头并选择**添加为链接**来创建一个到文件的链接，代替创建文件的拷贝。

A.5.3 编辑项目

我们在通用的开发环境中编辑代码和类模块，以及设计窗体。编辑代码和类模块在代码窗口进行，设计和编辑窗体在 Windows 窗体设计器中进行。

你有多少项目就可以打开多少窗口，允许查看不同窗体或模块中的代码，并相互间复制粘贴。

操作步骤

编辑项目中的项

- 进入 Microsoft Visual Studio，在**解决方案资源管理器**窗口内双击你要编辑的项。

  在 Windows 窗体设计器里如果想要编辑窗体上的窗体或控件的代码，直接双击窗体或控件就会显示出窗体的代码窗口。

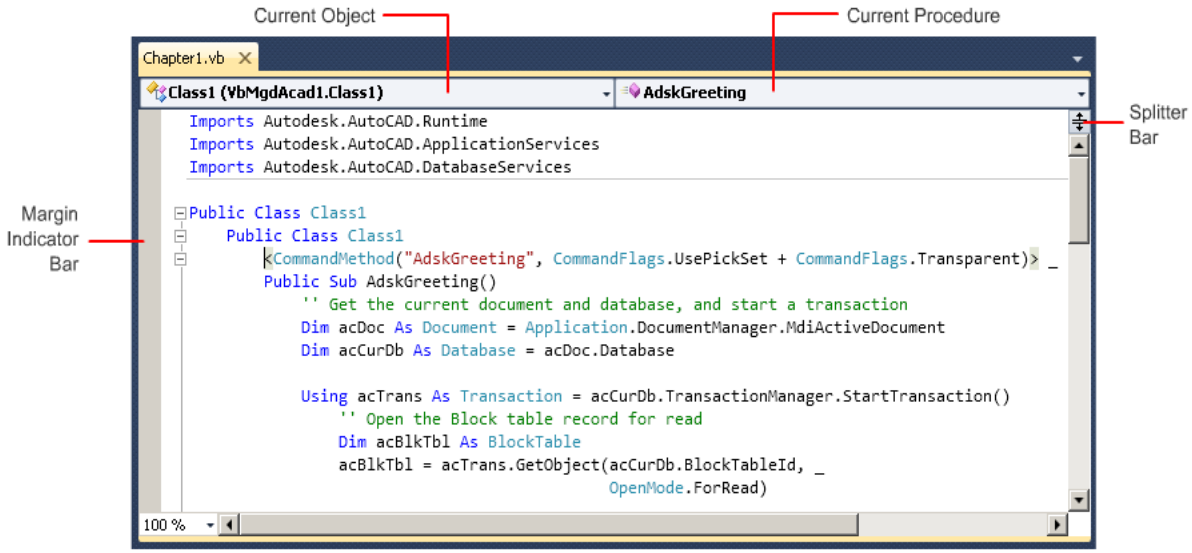
A.5.3.1 使用代码窗口

代码窗口包含两个下拉列表、一个分割条和一个边沿指示条。

代码窗口顶部的两个下拉列表，左边那个显示当前对象（类或窗体），右边那个显示当前对象的子程序。可以通过在下拉列表切换对象或子程序在项目中移动，以查看不同部分的代码。

代码窗口右边的分割条（splitter bar）允许你水平分割窗口。简单向下拖拉分割条就可以出来第二个窗格。这一特性允许你同时查看一个模块的两部分代码。想关闭第二个窗格，将分割条拖回原位或双击分割条即可。

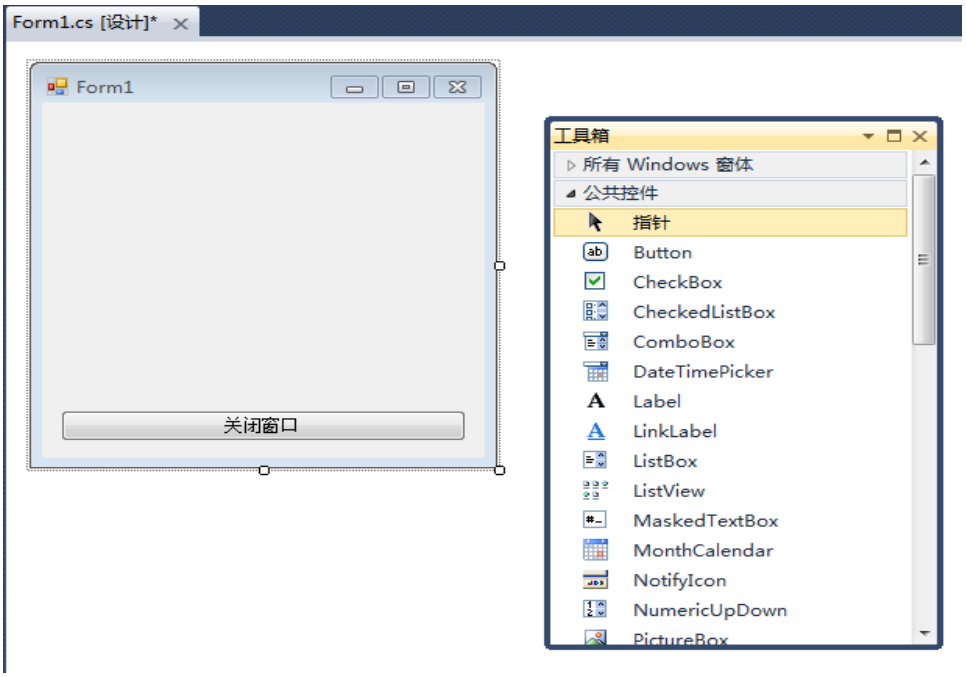
边沿指示条（margin indicator bar）竖在代码窗口的左侧，用来显示编辑代码或调试代码时的边沿指示器。



A.5.3.2 使用 Windows 窗体设计器

使用 Windows 窗体设计器可以为你的项目创建自定义对话框。

要将控件添加到窗体，先从工具箱窗口单击想要的控件激活它，然后在 Windows 窗体设计器的窗体中拖动鼠标，画出位置和大小合适的控件。可以从**选项对话框**中的 **Windows 窗体设计器**文件夹下的**常规**分组里设置控件对其到网格或与其他控件对其。在**常规**分组里还可以进行其他与网格有关的设置，如网格大小、布局模式（layout mode）等。



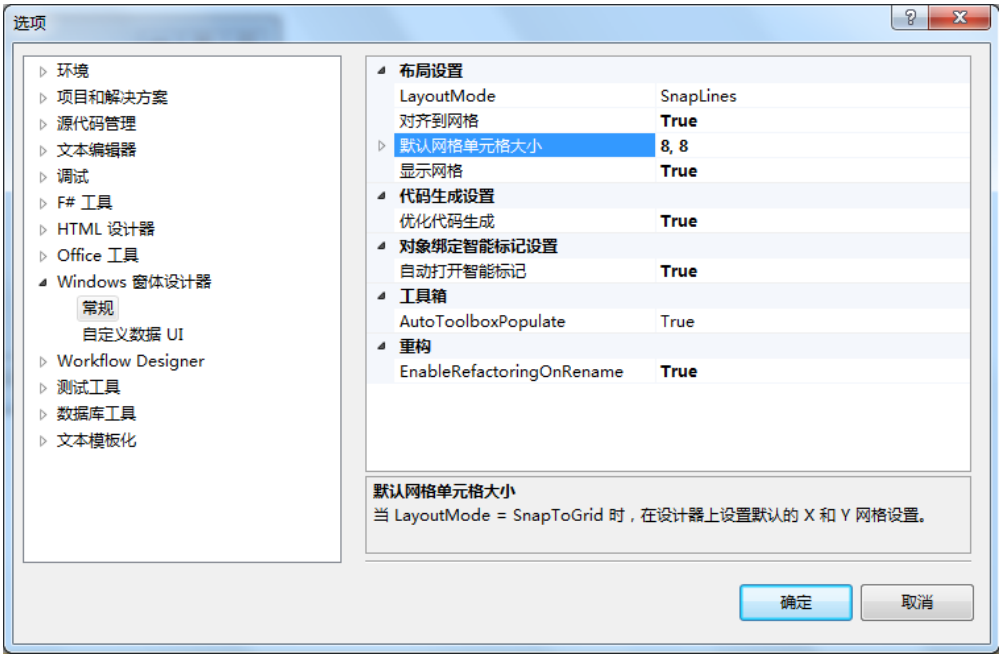
你设计的每个窗体都有标准最大化、最小化、关闭按钮。这些按钮是作为 Windows 窗体项目模板的一部分实现的。可以从 Windows 窗体设计器选中设计的窗体，然后使用属性对话框来修改窗体的外观和样式。

要想给窗体或控件事件添加代码，可以双击窗体或窗体上的控件。这时会打开窗体的代码窗口，并添加了控件的默认事件。使用代码窗口右上方的下拉列表可以为窗体或控件选择一个不同的事件，并添加到代码窗口。

操作步骤

设置网格对齐及网格大小

1. 进入 Microsoft Visual Studio，单击**工具**菜单 ➤ **选项**。
2. 在打开的**选项**对话框里，展开 **Windows 窗体设计器**节点，选中**常规**。



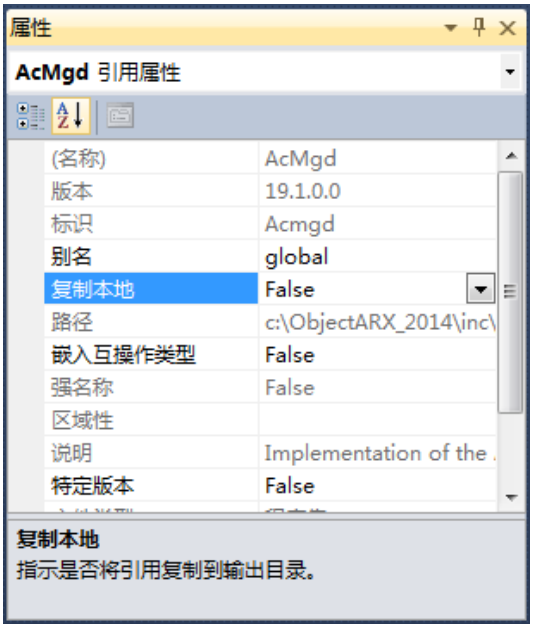
3. 在右侧的属性窗格里单击**对齐到网格**并从下拉列表里选择 True。

4. 单击**默认网格单元格大小**，然后单击右侧的输入框，输入网格的间隔值，格式为***宽，高***。

5. 单击**确定**。

A.5.3.3 使用属性窗口

属性窗口是用来修改当前项目或解决方案中文件的属性，以及 Windows 窗体中摆放的窗体和控件的属性的。



当选中你要修改的文件或对象（Windows 窗体或控件）时，其属性就会显示在属性窗口内。在属性窗口的表格内选中你要修改的属性，然后就可以输入或选择新属性值。

操作步骤

修改属性

1. 进入 Microsoft Visual Studio，从解决方案资源管理器选中一个文件或引用，或者从 Windows 窗体设计器窗口选中一个对象（窗体或控件）。
2. 如果没显示属性窗口，通过下列方法之一调出属性窗口：
   - 单击**视图**菜单 ➤ **其他窗口** ➤ **属性窗口**（或单击**视图**菜单 ➤ **属性窗口**）；
   - 按组合键 Alt+Enter ；
   - 右键单击选中的文件、引用或对象，单击**属性**。
3. 进入属性窗口，单击你要编辑的属性，然后在该属性右侧的字段里输入新属性值，或从右侧的下拉列表中选择新属性值。

A.5.4 项目重命名

创建一个项目时，项目文件名会和项目一起保存。如果项目文件名在 Visual Studio 外面被修改了，保存的项目名不会作相应的修改。作为最佳实践，只能从 Visual Studio 中重命名你的项目或解决方案，以保持一致性。关于创建项目的更多内容，参见（§A.4.1 *创建新项目*）。

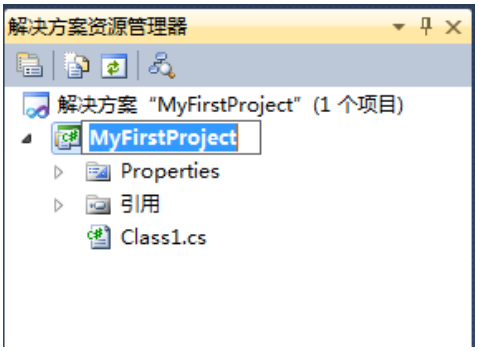
可以从**属性窗口**或通过 Windows 资源管理器修改项目的文件名。项目名称是在项目第一次创建时设置的，不过可以通过解决方案资源管理器窗口来修改。

警告 虽然可以通过 Windows 资源管理器来修改项目文件名，但不推荐这样做。这样做会破坏与解决方案文件或其他项目文件的引用关系。如果破坏了这种引用关系，当你打开解决方案或项目文件时，会提示找不到引用。

操作步骤

修改项目名称

1. 进入 Microsoft Visual Studio，在**解决方案资源管理器**内右键单击你想要重命名的项目并单击**重命名**。
2. 在就地文本编辑框内，键入一个新名称。



3. 按回车键或单击就地文本编辑框外边，完成重命名。

A.5.5 添加和引用其他项目

添加和引用项目允许我们在多个项目间共享代码。你可以创建一个常用方法、函数和类的中央库，然后在需要的时候引用该库。可以通过下列方式引用一个项目：

- 将项目添加到解决方案，然后在解决方案中的一个项目中引用它。
- 在一个项目中引用编译好的程序集文件。

在向解决方案中添加一个项目时，该项目的一个新节点会显示在解决方案资源管理器窗口内。此节点的标题与被引用的项目的是一样的(看不出引用与被引用关系)。添加完一个项目后，必须使用**添加引用**对话框将被引用项目添加给它。

如果被引用项目已经编译成 DLL 程序集文件，你可以使用**添加引用**对话框引用该 DLL 文件到一个项目。在你的项目中引用了另一个项目或 DLL 程序集文件后，为了利用引用所提供的对象（类或窗体），必须在你的项目中使用 Imports 语句或 using 语句声明所引用的命名空间或项目名。

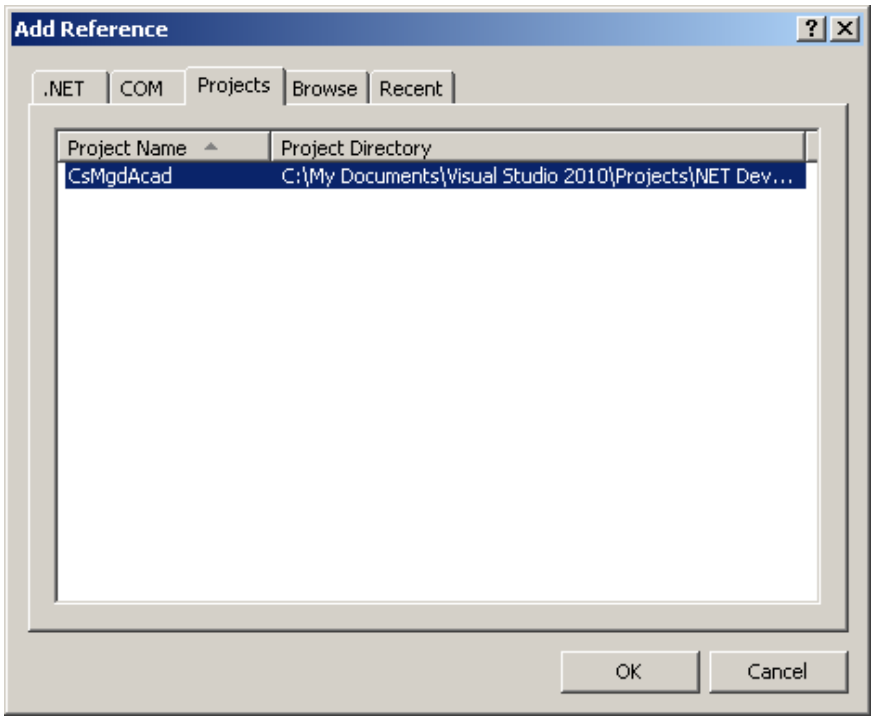
操作步骤

引用另一个项目

1. 添加一个新项目或现有项目到当前解决方案。

2. 添加完后，在**解决方案资源管理器**窗口，右键单击你要给它添加项目引用的那个项目，在右键菜单中单击**添加引用**。

3. 在**添加引用**对话框的**项目（Projects）**页，选中你要引用的项目，单击确定。



选中的项目会出现在**解决方案资源管理器**的**引用**文件夹下。

4. 在**解决方案资源管理器**中，双击要使用被引用项目提供的公共函数、方法或对象的代码模块。

5. 在代码模块顶端，为你要使用的函数、方法或对象所在的项目或命名空间添加 Imports 声明或 using 声明。

   例如，如果所引用的项目中包含公共类的命名空间的名称为 AdskUtilities，你应该给代码模块添加下列声明语句，表示你要引用项目中的 AdskUtilities 命名空间：

   VB.NET

   ```
   Imports AdskUtilities
   ```

   C#

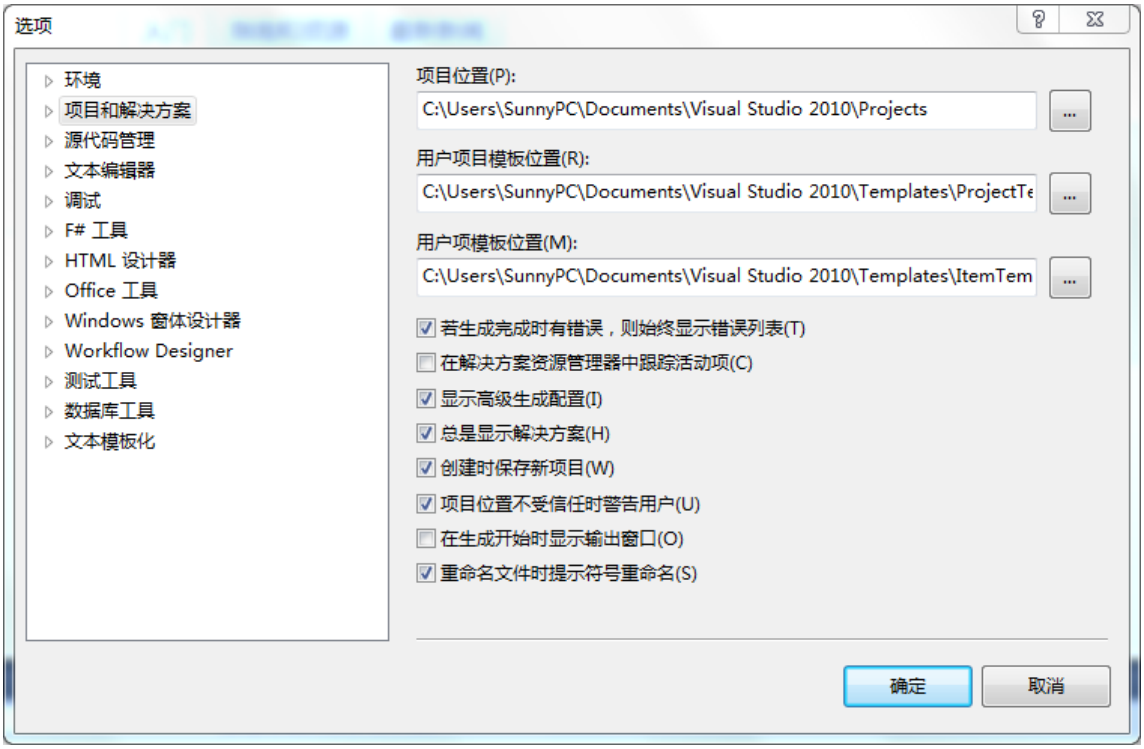
   ```
   using AdskUtilities;
   ```

6. 在项目中引用其他.NET 库或 COM 库时，用同样的方法声明使用的命名空间。

A.5.6 设置 Microsoft Visual Studio 选项

我们可以使用**选项**对话框修改常用开发环境的特征。

选项对话框包含各种设置选项，这些设置选项会影响到开发环境、代码编辑窗口、Windows 窗体设计器以及调试等。这些设置选项在对话框左侧的导航窗格内以文件夹的形式组织，每个文件夹内包含用于组织相关选项的分组。选中一个文件夹或分组后，该文件夹或分组内的设置选项就会显示在对话框的右侧。



常用的设置文件夹有：

- **环境**-包含用于控制开发环境内元素的显示和行为的设置选项。
- **项目和解决方案**-包含用于创建与编译项目和解决方案的设置选项。
- **文本编辑器**-包含用于控制代码窗口内输入的文本的行为的设置选项。
- **调试**-包含用于控制调试环境的设置选项。
- **Windows 窗体设计器**-包含用于控制 Windows 窗体设计器的显示和行为的设置选项。

关于选项对话框内所有可用选项的更多内容，可以从对话框左侧导航窗格内选中一个文件夹，然后单击窗口右上角的‘？’按钮。 这时 Microsoft Visual Studio 帮助文件中相关主题内容就会显示出来。（如果没有安装 Microsoft Visual Studio 帮助文件，会通过浏览器打开 MSDN 上的相关主题内容。 - 译者注）

操作步骤

显示选项对话框

- 进入 Microsoft Visual Studio，依次单击**工具**菜单 ➤ **选项**。

A.6 加载程序集到 AutoCAD

创建了解决方案和项目，定义了命名空间和类，实现了一个或多个命令或 AutoLISP 函数之后，接下来就可以使用 NETLOAD 命令将.NET 程序集加载到 AutoCAD 中运行。

使用调试环境

加载.NET 程序集前，应先确定是否需要使用 Microsoft Visual Studio 的调试环境对程序和函数中定义的业务逻辑进行测试。调试环境允许在运行.NET 程序集时，实时地单步执行代码。执行代码时你可以检查变量的值并观察程序执行的逻辑路径。

关于调试环境的更多内容，请参见开发环境自带的文档。

操作步骤

在 AutoCAD 中通过调试模式加载.NET 程序集

1. 进入 Microsoft Visual Studio，在**解决方案资源管理器**中右键单击你要加载到 AutoCAD 中的项目，单击**属性**。
2. 在〈项目〉属性页，单击**调试**选项页。
3. 在**调试**选项页的**启动操作**下面，单击**启动外部程序**并单击文本框右侧的省略号按钮。
4. 在打开的**选择文件**对话框中，浏览到 *C:\Program Files\Autodesk\AutoCAD 2012* 并选中文件 *acad.exe*，单击**打开**。
5. 在**解决方案资源管理器**中选中该项目的情况下，单击**调试**菜单 > **启动调试**。
6. 进入 AutoCAD，在命令提示栏输入 **netload** 并按回车键。
7. 在打开的选择.**NET 程序集** 对话框中，浏览找到生成的调试版程序集文件，单击**打开**。

   提示 生成的程序集文件的位置在 Microsoft Visual Studio 的输出窗格里有显示。
8. 在命令提示栏，输入命令或 AutoLISP 函数的名称及需要的参数。

在 AutoCAD 中使用 NETLOAD 加载.NET 程序集

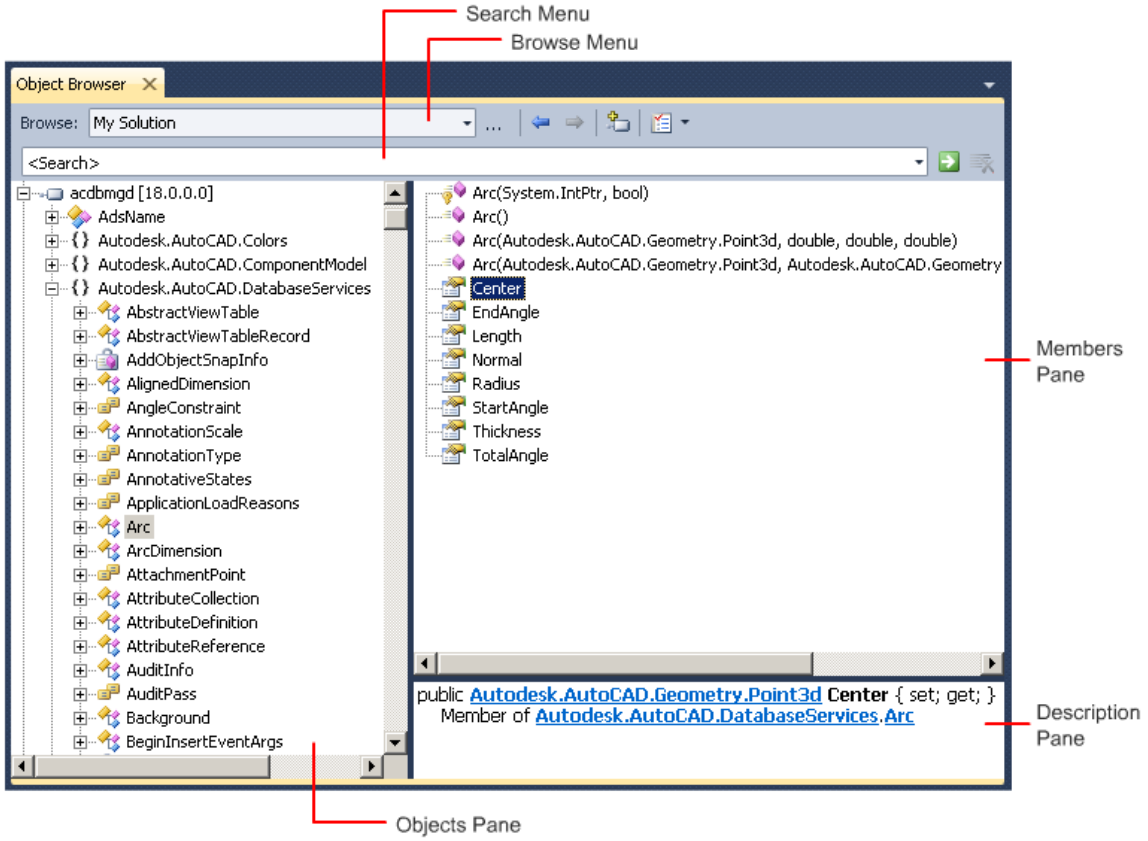
1. 在打开有解决方案或项目的 Microsoft Visual Studio 中，单击**生成**菜单 ➤ **生成解决方案**或**生成** *〈项目名称〉*。
2. 在 AutoCAD 的命令提示栏，键入 **netload** 并按回车键。
3. 在打开的选择.**NET 程序集** 对话框中，浏览找到生成的程序集文件，单击**打开**。

提示 生成的程序集文件的位置在 Microsoft Visual Studio 的输出窗格里有显示。

4. 在命令提示栏，输入命令或 AutoLISP 函数的名称及需要的参数。

A.7 访问和查找引用库（对象浏览器）

对象浏览器用来查看当前解决方案中项目所引用的库中定义的对象。通过对象浏览器，我们可以像使用字典结构那样浏览引用库，我们还可以使用**搜索（Search）**菜单根据输入的关键字来定位一个对象。



我们在对象浏览器中查看到的库的范围由**浏览（Browse）**菜单下拉列表来限定。你可以限定查看范围为当前解决方案、指定版本的.NET Framework 或一个自定义组件列表。

对象浏览器的左侧是一个对象导航窗格，显示的是可用的库。右侧分为两部分：上部（对象成员 Member 窗格）和下部（描述 Description 窗格）。

操作步骤

使用对象浏览器

1. 进入Microsoft Visual Studio，单击**视图**菜单 ➤ **对象浏览器**(或按组合键Ctrl+Alt+J)。

对象浏览器窗口默认以停靠选项卡形式出现在开发环境中间。

2. （可选）从浏览菜单为浏览或查找组件选择一个范围。
3. （可选）在搜索菜单内输入或选择一个关键字，用来过滤对象窗格的内容。
4. 在对象窗格，导航到你要查看的对象并选中它。
5. 在对象成员窗格，从显示出来的对象成员中选中一个，对它进行更多的了解。

A.8 练习：创建第一个项目

到目前为止我们学习了使用 Microsoft Visual Studio 的基本知识，接下来我们创建一个定义了一个新命令的项目。定义好新命令后，我们将加载编译生成的.NET 程序集到 AutoCAD。

A.8.1 练习：创建新项目

在本练习中，我们将创建一个名为“MyFirstProject”的新项目。

创建一个名为“MyFirstProject”的新项目

1. 在开始菜单，依次单击**所有程序** ➤ **Microsoft Visual Studio 2010** ➤ **Microsoft Visual Studio 2010**，启动 Microsoft Visual Studio。
2. 进入 Microsoft Visual Studio，依次单击**文件**菜单 ➤ **新建** ➤ **项目**。
3. 进入新建项目对话框，在已安装的模板树内，展开**其他语言** ➤ **Visual Basic** 或 **Visual C#** 并单击 **Windows**。
4. 在窗口中间的模板栏，选择类库。
5. 在**名称**输入框，输入 **MyFirstProject**。
6. 在**位置**输入框，单击**浏览**指定一个新位置或接受默认位置。

   提示 可以修改项目的默认存储位置，方法是进入**工具**菜单 ➤ **选项** ➤ **项目和解决方案**，输入或浏览选择新默认存储位置。
7. 单击**确定**。

A.8.2 练习：引用 AutoCAD .NET API 文件

在本练习中，我们将引用.NET 程序集 *acmgd.dll* 和 *acdbmgd.dll*。引用这两个文件后，还要调整这两个文件的属性，使之不被复制到编译目录来。

关于引用 AutoCAD .NET API 文件的更多内容，参见（§0.3 *AutoCAD .NET API 的组件*）。

引用 AutoCAD .NET API 文件

1. 如果没有显示解决方案资源管理器窗口，单击**视图**菜单 ➤ **解决方案资源管理器**，显示资源管理器窗口。
2. 单击解决方案资源管理器工具条上的**显示所有文件**按钮。
3. 右键单击**引用**节点，单击**添加引用**。
4. 在打开的**添加引用**对话框中，单击**浏览**选项页，浏览 AutoCAD ObjectARX 安装文件夹的 inc 目录，选择 *acmgd.dll*，按住 Ctrl 键，再选择 *acdbmgd.dll*。单击**确定**。
5. 回到解决方案资源管理器窗口，单击引用节点左侧的+号展开。
6. 按住 Ctrl 键，选中 AcDbMgd 和 AcMdg。
7. 在选中的引用上单击右键，单击**属性**。
8. 进入属性窗口，单击**复制本地**字段并从下拉列表中选择 **False**。

> **注：** 将复制本地属性设置为 False 意在告诉 Microsoft Visual Studio 在生成的项目输出中不要包含引用的 DLL 文件。如果引用的 Dll 文件被复制到编译输出文件夹，在 AutoCAD 加载程序集时会造成意外的结果。

A.8.3 练习：创建新命令

现在我们已经创建了一个项目，并添加了需要的引用库，接下来我们创建一个命令。该命令在模型空间创建一个多行文字（MText）。 相关概念会在其他章节深入探讨。

定义创建新 Mtext 对象的新命令

1. 在解决方案资源管理器窗口双击项目中的 *Class1.vb* 或 *Class1.cs*。
   这时会打开 Class1 模块的代码窗口，并显示如下内容：

VB.NET

```
Public Class Class1
End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.Linq; using System.Text;
namespace MyFirstProject1
{ public class Class1 { }}
```

2. 按照下面的内容修改代码窗口中的代码：

VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices

Public Class Class1
    <CommandMethod("AdskGreeting")> Public Sub AdskGreeting()
        '' 获取当前文档和数据库，启动事务
        Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
        Dim acCurDb As Database = acDoc.Database
        Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
            '' 以读模式打开块表
            Dim acBlkTbl As BlockTable
            acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                         OpenMode.ForRead)
            '' 以写模式打开块表记录模型空间
            Dim acBlkTblRec As BlockTableRecord
            acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                            OpenMode.ForWrite)
            '' 新建一个MText对象
            Dim objText As MText = New MText
            '' 指定MText对象的插入点
            objText.Location = New Autodesk.AutoCAD.Geometry.Point3d(2, 2, 0)
            ''设置MText对象文字串内容
            objText.Contents = "Greetings, Welcome to the AutoCAD .NET Developer's
Guide"
            '' 设置MText对象的文字样式
            objText.TextStyleId = acCurDb.Textstyle
            '' 添加MText对啊到模型空间
            acBlkTblRec.AppendEntity(objText)
            '' 添加新MText对象到当前事务
            acTrans.AddNewlyCreatedDBObject(objText, True)
            '' 保存修改到数据库，关闭事务
            acTrans.Commit()
        End Using
```

```
    End Sub
End Class
```

C#

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;

[assembly: CommandClass(typeof(MyFirstProject1.Class1))]

namespace MyFirstProject1
{
    public class Class1
    {
        [CommandMethod("AdskGreeting")]
        public void AdskGreeting()
        {
            // 获取当前文档和数据库，启动事务
            Document acDoc = Application.DocumentManager.MdiActiveDocument;
            Database acCurDb = acDoc.Database;
            // 使用Transaction Manager启动一个新事务
            using (Transaction acTrans =
acCurDb.TransactionManager.StartTransaction())
            {
                // 以读模式打开块表
                BlockTable acBlkTbl;
                acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                    OpenMode.ForRead) as BlockTable;
                // 以写模式打开块表记录模型空间
                BlockTableRecord acBlkTblRec;
                acBlkTblRec =
acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                    OpenMode.ForWrite) as BlockTableRecord;
                /* 新创建一个MText对象，
                 * 并设置插入位置、
                 * 文字内容及文字样式。 */
                MText objText = new MText();
                // 指定MText对象的插入点位置
                objText.Location = new Autodesk.AutoCAD.Geometry.Point3d(2, 2, 0);
                // 设置MText对象的字串内容
                objText.Contents = "Greetings, Welcome to the AutoCAD .NET Developer's
Guide";
                // 设置MText对象的文字样式
```

```
                    objText.TextStyleId = acCurDb.Textstyle;
                    // 添加MText对象到模型空间
                    acBlkTblRec.AppendEntity(objText);
                    // 添加MText对象到当前事务
                    acTrans.AddNewlyCreatedDBObject(objText, true);
                    // 保存修改到数据库并关闭事务
                    acTrans.Commit();
                }
            }
        }
    }
```

A.8.4 练习：设置项目的目标架构

在本练习中，我们将指定编译项目时用到的目标架构。

设置“MyFirstProject”项目的目标架构（target framework）

1. 如果没有显示解决方案资源管理器窗口，单击**视图**菜单 ➤ **解决方案资源管理器**，显示解决方案资源管理器窗口。
2. 单击解决方案资源管理器工具条上的**显示所有文件**按钮。
3. 右键单击 MyFirstProject 项目节点并单击**属性**。
4. 在打开的项目设计器窗口单击**应用程序**选项页。
5. 根据使用的语言做如下设置：
   - **VB.NET** - 在**编译**选项页单击高级编译选项按钮，从**目标 Framework** 下拉列表中选择.NET Framework 4.0。
   - **C#** - 在**应用程序**选项页的**目标 Framework** 下拉列表中选择.NET Framework 4.0。

A.8.5 练习：编译并加载.NET 程序集到 AutoCAD

到目前为止，我们已经创建了一个项目，并定义了一个命令，你是不是已经急不可待地要到 AutoCAD 中运行这个命令了？ 在能够运行定义的 AutoCAD 命令之前，首先需要作的是为项目编译生成一个.NET 程序集。

对于本练习来说，我们将为项目生成 Debug 版的程序集，不过我们不再这里讲解如何使用 Debug 环境。关于调试项目的更多内容，请参见开发环境自带的文档。关于在 Debug 环境下加载程序集到 AutoCAD 的内容，参见（§A.6 *加载程序集到 AutoCAD*）。

如果想要发布项目给别人使用，就需要从项目生产一个.NET 程序集。关于和他人分享.NET 程序集的更多内容，参见（§8.2 *发布应用程序*）。

编译生成.NET 程序集后，就可以使用 NETLOAD 命令将其加载到 AutoCAD 中，并像运行 AutoCAD 内置命令那样运行我们定义的命令。

编译项目并加载生成的.NET 程序集到 AutoCAD

1. 单击 Microsoft Visual Studio 的**生成**菜单 ➤ **生成 MyFirstProject**。

   项目应该会顺利通过编译，除非项目代码有错误。查看输出窗口，上面会显示项目编译状态信息，生成的 MyFirstProject.dll 文件的存储位置也会显示在输出窗口内。

2. 运行 AutoCAD（如果没运行的话）。

3. 在 AutoCAD 的命令提示行输入 **netload** 并回车。

4. 在打开的选择.NET 程序集对话框中，浏览到 MyFirstProject.dll 所在位置并选中该文件，单击**打开**。

5. 在 AutoCAD 的命令提示行输入 **adskgreeting** 并按回车键。

   你会看到在 AutoCAD 图形窗口创建了一个 Mtext 对象，坐标位置为(2,2)，文字内容为“Greetings, Welcome to the AutoCAD .NET Developer's Guide”。

A.9 相关 AutoCAD 命令和术语

命令

NETLOAD

加载一个.NET 程序集到 AutoCAD。

运行该命令会弹出一个标准文件选择对话框，选择需要加载的.NET 程序集。

当系统变量 FILEDIA 被设置为 0 时，NETLOAD 命令显示下面的命令行提示：

程序集文件名：输入文件名并按 Enter 键。

术语

程序集 Assembly

编译生成的具有 DLL 扩展名的项目文件。

VB.NET 项目

用 Microsoft Visual Studio 创建的具有 VBPROJ 扩展名的项目文件。

C#项目

用 Microsoft Visual Studio 创建的具有 CSPROJ 扩展名的项目文件。

代码编辑器窗口

用来编辑存储在类模块或窗体中的代码的窗口。

解决方案 Solution

加载到 Microsoft Visual Studio 用来管理一个或多个项目的文件。

引用 Reference

指向项目所使用的 API 库文件的链接。一个项目也可以被其他项目引用。

A.10 更多内容

更多关于 Microsoft Visual Studio 开发环境以及 VB.NET 或 C# 编程语言的内容，可以从微软提供的帮助文件获得，在 choose one of the options from the Help menu 从 Microsoft Visual Studio 的帮助菜单选择需要的选项即可。还可以考虑购买一本 Microsoft® Press 或第三方出版社出版的书来参考。

附录 B 比较 VBA/VB 与 VB.NET/C#

虽然大多数程序设计语言在语法和功能上彼此都不一样，但还是有一些基础概念和逻辑是相通的。本附录作为一个参考，意在帮助熟悉 VBA/VB 的开发人员方便查找对应的 VB.NET 或 C#语言功能。

B.1 比较 VBA/VB 与 VB.NET /C#

下面的表格将 VBA 函数与 VB.NET 和 C#类似的函数及运算符进行了比较。ActiveX 库由“AutoCAD.Application”表示，对等的.NET 托管库由“Autodesk.AutoCAD”表示， 并列出了对等的 VB.NET 或 C#函数或运算符。

数学函数	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(除注明者两种语言一样)
+ (加运算符)	+ (加运算符)
- (减运算符)	- (减运算符)
* (乘运算符)	* (乘运算符)
/ (除运算符)	/ (除运算符)
^ (幂运算符)	^ (幂运算符)
Abs 函数	System.Math.Abs 函数
Atn 函数	System.Math.Atan 函数
Cos 函数	System.Math.Cos 函数
Exp 函数	System.Math.Exp 函数
Log 函数	System.Math.Log 函数
Max 函数	System.Math.Max 函数
Min 函数	System.Math.Min 函数
Mod 函数	VB.NET： Mod 函数 C#： % (运算符) VB.NET and C#： System.Math.DivRem 函数
Sin 函数	System.Math.Sin 函数
Sqr 函数	System.Math.Sqrt 函数

条件语句和循环语句	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(两种语言一样除注明者)
Do Until… 循环语句	VB.NET Do Until... 循环语句 C# Use do... while 语句
Do While… 循环语句	VB.NET Do While... 循环语句 C# do... while 语句
For Each...Next 语句	VB.NET For Each...Next 语句 C# Foreach and For 语句 s
If… Then… Else...End If 语句	VB.NET If… Then… Else...End If 语句 C# if... else... 语句
Select Case 语句	VB.NET Select Case 语句 C# Switch 语句
While… Wend 语句	VB.NET While... Wend 语句 C# while... 语句

逻辑语句	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(除注明者两种语言一样)
= (等于比较操作符)	VB.NET = (等于比较操作符) C# == (等于比较操作符)
<> (不等于比较操作符)	VB.NET <> (不等于比较操作符) C# != (不等于比较操作符)
< (小于比较操作符)	< (小于比较操作符)
<= (小于等于比较操作符)	<= (小于等于比较操作符)
> (大于比较操作符)	> (大于比较操作符)
>= (大于等于比较操作符)	>= (大于等于比较操作符)
And 函数	VB.NET And 操作符 C# &&操作符
Eqv 操作符	未提供，使用位比较方法代替
Imp 操作符	未提供，用 = 等于比较符代替
Is 操作符	VB.NET *object* Is *object* C# *object* is *object*
IsArray 函数	VB.NET IsArray 函数 或

	TypeOf *arrayName* Is Array comparison C# typeof(*arrayName*) == Array comparison VB.NET and C# *varName*.GetType().IsArray
IsNull 函数	VB.NET IsDBNull 函数 C# Use == null 比较
Like 操作符	VB.NET Like 操作符 VB.NET and C# *stringVariable*.Contains 函数
Not 操作符	VB.NET Not 操作符 C# != (不等于比较操作符)
Or 函数	VB.NET Or 函数 C# ||操作符

数据类型转换函数	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(除注明者两种语言一样)
Asc 函数	VB.NET Asc 函数 C# (int)'*letter*'
AutoCAD.Application.ActiveDocument. Utility.AngleToReal 方法	Autodesk.AutoCAD.Runtime.Converter. StringToAngle 方法
AutoCAD.Application.ActiveDocument. Utility.AngleToString 方法	Autodesk.AutoCAD.Runtime.Converter. AngleToString 方法
AutoCAD.Application.ActiveDocument. Utility.RealToString 方法	Autodesk.AutoCAD.Runtime.Converter. DistanceToString 函数
CDbl 函数	VB.NET CDbl 函数 VB.NET and C# System.Convert.ToDouble 函数
Chr 函数	VB.NET Chr 函数 VB.NET and C# System.Convert.ToChar
CInt 函数	VB.NET CInt 函数 VB.NET and C# System.Convert.ToInt16, System.Convert.ToInt32, 或 System.Convert.ToInt64 函数
Fix 函数	VB.NET Fix 函数 VB.NET and C#

	System.Convert.ToInt16, System.Convert.ToInt32, 或 System.Convert.ToInt64 函数
Int 函数	VB.NET Int 函数 VB.NET and C# System.Convert.ToInt16, System.Convert.ToInt32, 或 System.Convert.ToInt64 函数
Str 函数	VB.NET Str 函数 VB.NET and C# System.Convert.ToString 函数
StrConv 函数	VB.NET StrConv 函数 VB.NET and C# System.Text.Encoding.Convert 函数

字符串操作函数	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(除注明者两种语言一样)
&操作符 (连接字符串)	VB.NET & 或 +操作符 C# + 操作符
Len 函数	VB.NET Len 函数 VB.NET and C# *stringVariable*.Length 属性
Mid 函数	VB.NET Mid 函数 VB.NET and C# *stringVariable*.Substring 函数

从 AutoCAD 命令提示行获取输入的函数	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(除注明者两种语言一样)
AutoCAD.Application.ActiveDocument.Utility.GetAngle 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetAngle 函数
AutoCAD.Application.ActiveDocument.Utility.GetCorner 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetCorner 函数
AutoCAD.Application.ActiveDocument.Utility.GetDistance 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetDistance 函数
AutoCAD.Application.ActiveDocument.Utility.GetEntity 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetEntity 函数
AutoCAD.Application.ActiveDocument.Utility.GetInteger 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetInteger 函数
AutoCAD.Application.ActiveDocument.Utility.GetKeyword 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetKeyword 函数
AutoCAD.Application.ActiveDocument.Utility.GetOrientation 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetAngle 函数
AutoCAD.Application.ActiveDocument.Utility.GetPoint 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetPoint 函数
AutoCAD.Application.ActiveDocument.Utility.GetReal 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetDouble 函数
AutoCAD.Application.ActiveDocument.Utility.GetString 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetString 函数
AutoCAD.Application.ActiveDocument.Utility.InitializeUserInput	Autodesk.AutoCAD.EditorInput.PromptKeywordOptions

AutoCAD 应用程序及图形数据库基本函数	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(除注明者两种语言一样)
AutoCAD.Application.ActiveDocument. Utility.AngleFromXAxis 方法	Autodesk.AutoCAD.Geometry.Point2d(*point1*). GetVectorTo(*point2*).Angle 属性
AutoCAD.Application.ListARX 方法	Autodesk.AutoCAD.Runtime.SystemObjects. DynamicLinker.GetLoadedModules 函数
AutoCAD.Application.LoadARX 方法	Autodesk.AutoCAD.Runtime.SystemObjects. DynamicLinker.LoadModule 方法
AutoCAD.Application.UnloadARX 方法	Autodesk.AutoCAD.Runtime.SystemObjects. DynamicLinker.UnloadModule 方法
AutoCAD.Application. Documents.Close 方法	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument. CloseAndDiscard 方法 或 Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument. CloseAndSave 方法
AutoCAD.Application.ActiveDocument. SendCommand 方法	Autodesk.AutoCAD.ApplicationServices.Application. DocumentManager.MdiActiveDocument. SendStringToExecute 方法
AutoCAD.Application.ActiveDocument. Dictionaries.Add 方法	VB.NET *dictionaryObj* = *transactionObj*. GetObject(*workingDatabaseObj*. NamedObjectsDictionaryId, *openMode*) *dictionaryObj*.SetAt 函数 C# *dictionaryObj* = *transactionObj*. GetObject(*workingDatabaseObj*. NamedObjectsDictionaryId, *openMode*) as Autodesk.AutoCAD.DatabaseServices. DBDictionary; *dictionaryObj*.SetAt 函数
AutoCAD.Application.ActiveDocument. Dictionaries.Item 方法	VB.NET *dictionaryObj* = *transactionObj*.

	GetObject(*workingDatabaseObj*. NamedObjectsDictionaryId, *openMode*) *dictionaryObj*.GetAt 函数 C# *dictionaryObj* = *transactionObj*. GetObject(*workingDatabaseObj*. NamedObjectsDictionaryId, *openMode*) as Autodesk.AutoCAD.DatabaseServices. DBDictionary; *dictionaryObj*.GetAt 函数
AutoCAD.Application.ActiveDocument. ModelSpace 属性	VB.NET *blockTableObj* = *transactionObj*. GetObject(*workingDatabaseObj*.BlockTable Id, *openMode*) *blockTableRecordObj* = *transactionObj*. GetObject(*blockTableObj*(BlockTableRecor d. ModelSpace), *openMode*) C# *blockTableObj* = *transactionObj*. GetObject(*workingDatabaseObj*.BlockTable Id, *openMode*) as Autodesk.AutoCAD. DatabaseServices.BlockTable; *blockTableRecordObj* = *transactionObj*. GetObject(*blockTableObj*[BlockTableRecor d. ModelSpace], *openMode*) as Autodesk.AutoCAD.DatabaseServices. BlockTableRecord;
AutoCAD.Application.ActiveDocument. ModelSpace.Item 方法	VB.NET

	blockTableObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.BlockTable
	Id, *openMode*)
	blockTableRecordObj = *transactionObj*.
	GetObject(*blockTableObj*(BlockTableRecor
	d.
	ModelSpace), *openMode*)
	dbObj = *blockTableRecordObj*(*index*)
	
	C#
	blockTableObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.BlockTable
	Id, *openMode*) as Autodesk.AutoCAD.
	DatabaseServices.BlockTable;
	blockTableRecordObj = *transactionObj*.
	GetObject(*blockTableObj*[BlockTableRecor
	d.
	ModelSpace], *openMode*) as
	Autodesk.AutoCAD.DatabaseServices.
	BlockTableRecord;
	foreach(*objecId* in *blockTableRecordObj*)
	{
	objObject = *transactionObj*.GetObject(*objec
	Id*);
	}

AutoCAD.Application.ActiveDocument.
ModelSpace.Count 属性
	VB.NET
	blockTableObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.BlockTable
	Id, *openMode*)
	blockTableRecordObj = *transactionObj*.
	GetObject(*blockTableObj*(BlockTableRecor
	d.
	ModelSpace), *openMode*)
	Dim *nCount* As Integer = 0
	For Each *objectId* In *blockTableRecordObj*
	nCount = *nCount* + 1
	Next
	
	C#
	blockTableObj = *transactionObj*.
	GetObject(*workingDatabaseObj*.BlockTable
	Id,
	openMode) as Autodesk.AutoCAD.
	DatabaseServices.BlockTable;
	blockTableRecordObj = *transactionObj*.
	GetObject(*blockTableObj*[BlockTableRecor
	d.
	ModelSpace], *openMode*) as Autodesk.
	AutoCAD.DatabaseServices.BlockTableRe
	cord;
	int cnt = 0;
	foreach(*objectId* in *blockTableRecordObj*)
	{
	cnt = cnt + 1;
	}

AutoCAD.Application.ActiveDocument.ModelSpace.Add<entityname>方法	VB.NET
	blockTableObj = *transactionObj*.GetObject(*workingDatabaseObj*.BlockTableId, *openMode*)
	blockTableRecordObj = *transactionObj*.GetObject(*blockTableObj*(BlockTableRecord.ModelSpace), *openMode*)
	blockTableRecordObj.AppendEntity(*someEntity*)
	transactionObj.AddNewlyCreatedDBObject(*someEntity*, True)
	
	C#
	blockTableObj = *transactionObj*.GetObject(*workingDatabaseObj*.BlockTableId,
	openMode) as Autodesk.AutoCAD.DatabaseServices.BlockTable;
	blockTableRecordObj = *transactionObj*.GetObject(*blockTableObj*[BlockTableRecord.ModelSpace], *openMode*) as Autodesk.AutoCAD.DatabaseServices.BlockTableRecord;
	blockTableRecordObj.AppendEntity(*someEntity*);
	transactionObj.AddNewlyCreatedDBObject(*someEntity*, true);

AutoCAD.Application.ActiveDocument.ActiveSpace 属性	VB.NET *blockTableRecordObj* = *transactionObj*.GetObject(*workingDatabaseObj*.CurrentSpaceId, *openMode*) C# *blockTableRecordObj* = *transactionObj*.GetObject(*workingDatabaseObj*.CurrentSpaceId, *openMode*) as Autodesk.AutoCAD.DatabaseServices.BlockTableRecord;

AutoCAD.Application.ActiveDocument. PaperSpace 属性	VB.NET *blockTableObj* = *transactionObj*. GetObject(*workingDatabaseObj*.BlockTable Id, *openMode*) *blockTableRecordObj* = *transactionObj*. GetObject(*blockTableObj*(BlockTableRecor d. PaperSpace), *openMode*) C# *blockTableObj* = *transactionObj*. GetObject(*workingDatabaseObj*.BlockTable Id, *openMode*) as Autodesk.AutoCAD. DatabaseServices.BlockTable; *blockTableRecordObj* = *transactionObj*. GetObject(*blockTableObj*[BlockTableRecor d. PaperSpace], *openMode*) as Autodesk.AutoCAD.DatabaseServices. BlockTableRecord;
AutoCAD.Application.ActiveDocument. ActiveLayout 属性	VB.NET *layoutObj* = *transactionObj*. GetObject(*layoutManagerObj*. GetLayoutId(*layoutManagerObj*. CurrentLayout), *openMode*) *blockTableRecordObj* = *transactionObj*. GetObject(*layoutObj*.BlockTableRecordId, *o* *penMode*) C# *layoutObj* = *transactionObj*. GetObject(*layoutManagerObj*. GetLayoutId(*layoutManagerObj*. CurrentLayout), *openMode*) as Autodesk.AutoCAD. DatabaseServices.Layout; *blockTableRecordObj* = *transactionObj*. GetObject(*layoutObj*.BlockTableRecordId, *openMode*) as Autodesk. AutoCAD. DatabaseServices.BlockTableRecord;
AutoCAD.Application.ActiveDocument.	HostApplicationServices.WorkingDatabase.

PurgeAll 方法	Purge 方法
AutoCAD.Application.GetVariable 方法	Autodesk.AutoCAD.ApplicationServices.Application.GetSystemVariable 函数
AutoCAD.Application.MenuBar 属性	Autodesk.AutoCAD.ApplicationServices.Application.MenuBar 属性
AutoCAD.Application.MenuGroup 属性	Autodesk.AutoCAD.ApplicationServices.Application.MenuGroups 属性
AutoCAD.Application.ActiveDocument.PickfirstSelectionSet 属性	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.SelectImplied 函数
AutoCAD.Application.ActiveDocument.Utility.PolarPoint 方法	Not provided, use the Point2d and Point3d classes from the Geometry namespace to calculate a new point
AutoCAD.Application.Preferences 属性	Autodesk.AutoCAD.ApplicationServices.Application.Preferences 属性
AutoCAD.Application.ActiveDocument.Utility.Prompt 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.WriteMessage 方法
AutoCAD.Application.Quit 方法	Autodesk.AutoCAD.ApplicationServices.Application.Quit 方法
AutoCAD.Application.ActiveDocument.SelectionSets.Add 方法	Not needed/provided
AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.Item 方法	Autodesk.AutoCAD.EditorInput.SelectionSet.*selectionSet*.Item(*object*)方法
AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.Delete 方法	Autodesk.AutoCAD.EditorInput.SelectionSet.*selectionSet*.Item(*object*).Delete 方法
AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.SelectOnScreen 方法	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.GetSelection 方法
AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.Count 属性	Autodesk.AutoCAD.EditorInput.SelectionSet.*selectionSet*.Count 属性
AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.SelectAtP	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor.

oint 方法	SelectCrossingWindow 方法
AutoCAD.Application.SetVariable 方法	Autodesk.AutoCAD.ApplicationServices.Application.SetSystemVariable 方法
AutoCAD.Application.ActiveDocument.Utility.TranslateCoordinates 方法	Not provided, use the Matrix3d class from the Geometry namespace to translate points between different coordinate systems
AutoCAD.Application.Version 属性	Autodesk.AutoCAD.ApplicationServices.Application.Version 属性
ThisDrawing	Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument 属性 and HostApplicationServices.WorkingDatabase 属性

VBA 和 VB6 基本函数和语句	
ActiveX/VBA/VB6	VB.NET/C# 对等功能(除注明者两种语言一样)
AppActivate AutoCAD.Application.Caption 函数	VB.NET AppActivate Autodesk.AutoCAD. ApplicationServices.Application. MainWindow.Text VB.NET and C# 调用 Win32 的 ShowWindow 和 SetWindowPos
Dir 函数	System.IO.Directory.Exists 函数
Error 对象/方法/属性	VB.NET Error 对象/方法/属性 VB.NET and C# Try Catch 异常处理语句

Function 和 End Function 关键字	VB.NET Function 和 End Function 关键字并使用 *Return* 返回一个值 C# 定义一个过程并使用 *Return* 返回一个值
Input 函数	VB.NET Input 方法 VB.NET and C# *fileStream*.Read 方法
LBound(*arrayName*) 函数	VB.NET LBound(*arrayName*) 函数 VB.NET and C# *arrayName*.GetLowerBound 函数
Line Input 函数	VB.NET LineInput 方法 VB.NET and C# *fileStream*.Read 方法
MsgBox 函数	MessageBox.Show 方法
object(n) syntax	VB.NET object(n) syntax C# object[n] syntax
Open 函数	System.IO.File.Open 函数
ReDim 语句	VB.NET ReDim *arrayName*(*newSize*) VB.NET and C# *arrayName*.Resize
Set 语句	不需要/未提供

Shell 函数	VB.NET Shell 函数 VB.NET and C# System.Diagnostics.Process.Start 函数
Sub 和 End Sub 关键字	VB.NET Sub 和 End Sub 关键字 C# 定义一个过程
TypeName 函数	VB.NET TypeName 函数 VB.NET and C# *varName*.GetType().Name or *varName*.GetType().FullName 函数 s
UBound(*arrayName*) 函数	VB.NET UBound(*arrayName*) 函数 VB.NET and C# *arrayName*.GetUpperBound 函数