ArcObjects
GIS 应用开发
——基于 C#.NET

（第一版）

兰小机　刘德儿编著

江西理工大学

2006 年 9 月

目 录

1 开发基础 1

1.1 ARCOBJECTS基础 2
1.1.1 ArcObjects简介 2
1.1.2 ArcObjects的组件库 4
1.1.3 理解对象模型图 9
1.2 组件对象模型 13
1.2.1 软件开发历史 13
1.2.2 什么是组件对象模型 15
1.3 .NET平台概述 19
1.3.1 通用语言运行时 21
1.3.2 .NET Framework 类库 24
1.4 .NET组件与COM组件的互操作 26
1.4.1 COM包装（COM Wrappers） 27
1.4.2 ESRI提供的Primary Interop Assemblies 30
1.4.3 向COM公开.NET对象 31
1.4.4 COM到.NET的类型转换 34
1.5 ARCGIS开发资源 35
1.5.1 ArcGIS Software Developer Kit (SDK) 35
1.5.2 ESRI Developer Network Web site 41
1.6 ARCGIS的二次开发方式 41
1.6.1 使用VBA进行ArcGIS桌面的定制 42
1.6.2 使用DLL组件扩展ArcGIS桌面 49
1.6.3 使用控件构建独立应用程序 64

2 ARCMAP用户界面定制 71

2.1 APPLICATION对象 71
2.1.1 IApplication接口 72
2.1.2 IMxApplication接口 74
2.1.3 IWindowPostion接口 74
2.1.4 IEnumprinterNames 75
2.2 DOCUMENT对象 75
2.2.1 IMxDocument接口 78
2.2.2 IDocumentDefaultSymbols接口 79
2.2.3 IDocumentEvents、IDocumentEventsDisp接口 81

2.3 定制ARCMAP的界面 82
2.3.1 定制状态栏(StatusBar) 82
2.3.2 定制可停靠窗口(DockableWindow) 84
2.3.3 快捷键表 85
2.3.4 CommandBars和 CommandBar对象 86
2.3.5 CommandItem对象 89
2.4 UICONTROL的使用 90
2.5 使用EXTENSION扩展ARCGIS 91
3 使用ARCOBJECTS控件编程 94
3.1 MAPCONTROL控件 95
3.1.1 主要接口 95
3.1.2 MapControl控件与MXD文件 97
3.1.3 鼠标与控件的交互 102
3.1.4 数据选择 104
3.1.5 实现鹰眼功能 104
3.2 PAGELAYOUTCONTROL控件 108
3.2.1 PageLayoutControl操作 MXD文件 109
3.2.2 PageLayout与MapControl联动 110
3.2.3 在 Pagelayout里面绘制元素 112
3.3 TOCCONTROL控件 115
3.3.1 TOCControl控件概述 115
3.3.2 应用开发实例 118
3.4 TOOLBARCONTROL及相关对象 122
3.4.1 概述 122
3.4.2 ToolbarControl控件 124
3.4.3 ToolbarItem 127
3.4.4 ToolbarMenu 组件类 128
3.4.5 CustomizeDialog 129
3.5 CONTROLCOMMANDS 132
3.5.1 概述 132
3.5.2 ControlCommands with the ToolbarControl 132
3.5.3 ControlCommands without the ToolbarControl 132
3.6 应用开发实例 134
3.6.1 概述 134
3.6.2 自定义命令开发实例 138
3.6.3 自定义工具开发实例 140
3.6.4 自定义菜单开发实例 143

4 几何形体对象--GEOMETRY 149

4.1 GEOMETRY模型 149
4.2 POINT和MULTIPOINT对象 153
4.2.1 Point对象 153
4.2.2 Multipoint对象 160
4.3 包络线 ENVELOPE 163
4.4 曲线(CURVE) 166
4.4.1 Segment对象 168
4.4.2 路径(Path)对象 178
4.4.3 环(Ring)对象 180
4.4.4 PolyCurve对象 181
4.5 GEOMETRY集合接口 187
4.5.1 IGeoemtryCollection接口 187
4.5.2 ISegmentCollection接口 189
4.5.3 IPointCollection接口 190
4.6 动态跟踪 190
4.7 空间参考 192
4.7.1 什么是空间参考 192
4.7.2 两种坐标系统 195
4.7.3 A0 中空间参考的设置 198
4.7.4 SpatialReference对象 200

5 地图的组成 202

5.1 MAP对象 202
5.1.1 IMap接口 203
5.1.2 IGraphicsContainer接口 206
5.1.3 IActiveView接口 208
5.1.4 IActiveViewEvents接口 211
5.1.5 IMapBookmark接口 211
5.1.6 ITableCollection接口 211
5.2 图层对象 212
5.2.1 ILayer接口 212
5.2.2 要素图层 213
5.2.3 往地图中加入CAD文件 221
5.2.4 TIN图层 224
5.2.5 GraphicsLayer 225
5.3 SCREENDISPLAY对象 225

5.4 地图排版 229
5.4.1 PageLayout对象 230
5.4.2 Page对象 231
5.4.3 SnapGrid对象 232
5.4.4 SnapGuides对象 233
5.4.5 RulerSettings对象 234
5.5 ELEMENT对象 234
5.5.1 图形元素 235
5.5.2 框架元素 241
5.5.3 元素的选择跟踪对象 244
5.6 MAPGRID对象模型 245
5.6.1 MapGrid对象 246
5.6.2 MapGridBorder对象 248
5.6.3 MapGridLabel对象 249
5.6.4 MapGridFactory对象 252
5.7 MAPSURROUND对象 254
5.7.1 图例对象 254
5.7.2 指北针对象 257
5.7.3 比例尺对象 257
5.7.4 比例文本对象 261
5.8 使用STYLE对象 262
5.8.1 StyleGallery对象 263
5.8.2 StyleSelector对象 265
5.8.3 IStyleDialog接口 266
5.9 IDENTIFYDIALOG对象 268
6 空间数据的符号化 271
6.1 颜色对象 271
6.1.1 Color对象 274
6.1.2 颜色对话框 275
6.1.3 ColorRamp对象 278
6.2 SYMBOL对象 280
6.2.1 MarkerSymbol对象 282
6.2.2 LineSymbol对象 285
6.2.3 FillSymbol对象 288
6.2.4 TextSymbol对象 291
6.2.5 3DChartSymbol对象 294
6.3 制作专题图 295

6.3.1 SimpleRenderer着色法 297
6.3.2 ClassBreakRenderer着色法 299
6.3.3 UniqueValueRenderer着色法 304
6.3.4 ProportionalSymbolRenderer着色法 308
6.3.5 ChartRenderer着色 312
6.3.6 DotDensityRenderer着色法 323
6.3.7 着色中使用的计算方法 326
6.4 地图的标注 330
6.4.1 TextElement标注 330
6.4.2 ArcObjects中的标注 333
6.5 如何与地图进行互动 336
6.5.1 产生新Geometry的Feedback对象 337
6.5.2 移动几何形体对象上的节点 344
6.5.3 移动整个对象 349
6.5.4 其他 DisplayFeedback介绍 352

7 地图的输出 358

7.1 地图的打印输出 358
7.1.1 Printer类 358
7.1.2 Paper对象 360
7.1.3 在控件中打印输出 362
7.2 地图的转换输出 363
7.2.1 基于影像格式的输出 363
7.2.2 基于矢量格式的输出 366
7.2.3 ExportFileDialog对象 369

8 空间数据管理 373

8.1 GEODATABASE的基础知识 374
8.2 WORKSPACE 及相关对象 378
8.2.1 打开工作空间 379
8.2.2 工作空间 383
8.2.3 PropertySet 387
8.2.4 名称对象 388
8.3 DATASET对象 389
8.3.1 GeoDataset类 390
8.3.2 FeatureDataset对象 390
8.4 表、对象类和要素类 393
8.4.1 Table对象 395

8.4.2 ObjectClass对象 398
8.4.3 FeatureClass对象 401
8.4.4 字段集与字段 403
8.4.5 与字段相关的对象 408
8.5 ROW，OBJECT 和 FEATURE 对象 414
8.5.1 RowBuffer和Row对象 415
8.5.2 Object和Feature对象 416
8.5.3 点选法获取要素 420
8.5.4 更新要素 423
8.6 关系与关系类 425
8.7 数据转换与装载 428
9 空间分析 433
9.1 空间查询 433
9.1.1 Cursor对象 434
9.1.2 QueryFilter与SpatialFilter对象 435
9.1.3 QueryDef对象 439
9.1.4 TableSort对象 440
9.1.5 要素选择集 442
9.2 空间拓扑运算 445
9.2.1 ITopologicalOperator接口 445
9.2.2 开发实例 451
9.3 空间关系运算 457
9.3.1 IRelationaloperator接口 457
9.3.2 开发实例--查找某一多边形要素的所有邻接要素 461
9.4 IPROXIMITYOPERATOR接口 465
9.5 叠加分析 468
9.5.1 矢量图层叠加分析 468
9.5.2 栅格图层的叠加分析 477
10 空间数据编辑--MAPEDITING实例剖析 480
10.1 IWORKSPACEEDIT接口 480
10.2 MAPEDITING功能剖析 482
10.2.1 功能概述 482
10.2.2 代码分析 483
11 ARCCATALOG对象 493
11.1 GXOBJECT对象 495

11.2 GXDIALOG对象 499
12 RASTER数据简介 502
12.1 RASTERWORKSPACE对象 503
12.2 栅格数据集 505
12.3 栅格图层对象 507
参考文献 509

1 开发基础

随着计算机软件技术的发展，组件技术已经成为 GIS 应用开发的主流。组件技术具有面向对象、语言无关等特性，把 GIS 的各大功能模块划分成若干个组件，每个组件完成不同的功能，用户可根据实际需要选择购买和安装所需的 GIS 组件。各个 GIS 组件之间，以及 GIS 组件和其它非 GIS 组件之间，可以方便地通过可视化开发工具集成起来，形成兼有属性数据处理和空间数据处理的应用系统。

ArcObjects 是美国 ESRI(Environmental Systems Research Institute，Inc.)公司提供的一个 GIS 技术框架，它是基于 Microsoft 的 COM 技术开发的一套 COM 组件对象集，这个组件集提供了几乎全部的底层 GIS 功能，就连 ArcGIS 软件本身都是使用 ArcObjects 的组件开发出来的。由于 ESRI 开放了这个组件库的 APIs，因此普通程序员也可以使用 ArcObjects 开发自己的 GIS 应用程序。

使用 ArcObjects 进行的 GIS 二次开发，就是采用通用的软件开发平台如 C# .NET、VB .NET、VB、Visual C++、Delphi、PowerBuilder 等，按照项目的具体需求将 ArcObjects 的功能组件重新进行组装，以集成一个更强大、更复杂的 COM 组件或应用程序的过程。使用 ArcObjects 进行组件式 GIS 二次开发，是目前最流行、最灵活和最稳定的 GIS 开发方式。

C#是.NET 平台的母语， 虽然.NET 也支持 Java 以外的各种主要语言，但要开发.NET 平台应用程序，C#显然是第一选择。事实上，.NET 平台本身的许多重要应用（比如 ASP.NET）就是用 C#编写的。所以，本书的开发语言采用 C# .NET。

本章将讨论开发 ArcObjects 时需要知道的一些基本概念、方法，它们将为读者理解本书的其他内容打下基础。它包括以下重要内容：

- ArcObjects 基础
- 组件对象模型
- .NET 平台概述
- .NET 组件与 COM 组件的互操作
- ArcGIS 开发资源
- ArcGIS 的二次开发方式

1.1 ArcObjects 基础

1.1.1 ArcObjects 简介

ArcObjects 是 ESRI 公司 ArcGIS 系列产品的开发平台，它是基于 Microsoft COM 技术所构建的一系列 COM 组件产品。ArcObjects 不是为最终用户而是专门为开发人员提供的二次开发软件，通过 ArcObjects，用户可以非常方便地开发出功能强大的 GIS 应用系统。到 ArcGIS 8.3 为止，ArcObjects 还不是一个独立的应用产品，而是捆绑在用户购买的 ArcGIS Desktop 产品中的软件开发包。也就是说，只要用户购买了 ArcGIS Desktop 的任何一个产品，不管是 ArcView 还是 ArcInfo，都将拥有功能强大的 ArcObjects 组件集，利用 ArcObjects 提供的组件对象可进行应用开发。从 ArcGIS 9.0 开始，ESRI 推出了 ArcEngine，该产品使得 ArcObjects 可以作为独立的产品进行发布和使用。

ArcObjects 是一套 ArcGIS 的可重用的通用的二次开发组件产品，它可以用于大量开发框架中，包括流行的像 .NET 、Visual C++、Visual Basic、Delphi 等开发环境，因此开发人员可以在自己熟悉的开发环境中利用 ArcObjects 开发 GIS 应用。

ESRI 公司将其软件使用 COM 技术重新建构以后，于 1999 年推出了全新的 GIS 产品 Arclnfo 8。2001 年，ArcGIS 8.1 出现了。2004 年，ArcGIS 的版本变为 9。ArcGIS 是一套全面的、完善的和可伸缩的软件平台。如图 1-1 所示，ArcGIS 软件体系分为四个部分：桌面版 GIS，嵌入式 GIS，GIS 服务器和移动 GIS。桌面版 GIS、嵌入式 GIS 和服务器 GIS 的一部分，都是使用 ArcObjects 开发的，这些 GIS 程序正是通过 ArcObjects 组件对象来获取数据，完成地理分析任务并输出地图的。

桌面版 GIS 是一套完整的、可升级的软件系统。在这套软件系统中，功能最弱的是 ArcReader，它是一个免费分发的用于查看地图的软件。比它功能强大的依次是 ArcView、ArcEdit 和 Arclnfo。ArcView、ArcEdit 和 Arclnfo 并不是一个软件的名称，它是桌面版 GIS 的一种版本代码，这三种版本的软件系统都是由 ArcMap、ArcCatalog、ArcScene 等单个软件组成的，但是它们包含的 GIS 功能不一样。在三个版本中，ArcView 功能最弱，Arclnfo 的功能最强，它包含了 ArcGIS 全部的 GIS 功能，当然后者的费用也远远高于前者。

在全新的 ArcGIS 桌面版软件中，数据的显示分析和制图使用 ArcMap 软件来完成，数据的管理使用 ArcCatalog 来实现，而数据的转换和空间运算等操作则通过 ArcToolBox 来进行，与以前的版本不同，在 ArcGIS9 之后的版本中，ArcToolBox 不再作为一个单独的软件出现，它是前两个程序的一部分。除此以外，ArcGIS9 中还有用于三维分析的软件 ArcScene、ArcGlobe 和用于阅读地图的 ArcReader 软件。

前两个软件必须有 3D 模块支持才可以使用。

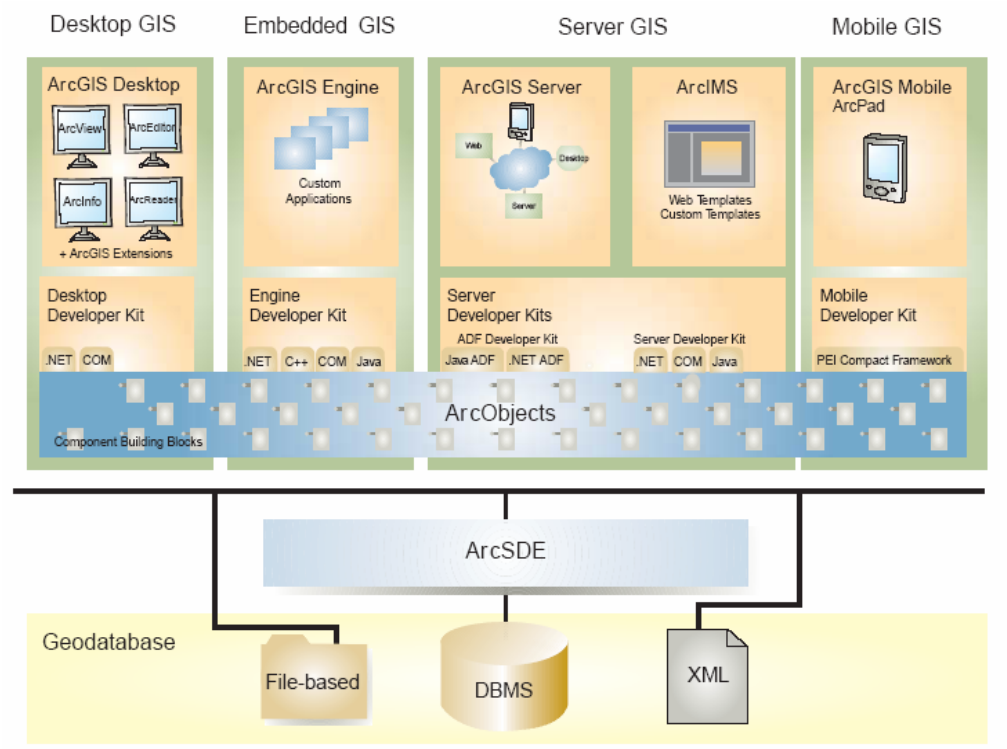


图 1-1 ArcGIS 9 产品系列

嵌入式 GIS 产品是 ArcGIS Engine。由于使用桌面版的 ArcObjects 开发出来的程序仍然无法脱离 ArcGIS 平台，出于产品战略上的考虑，ESRI 认识到只有让终端用户得到更多实惠才能够更进一步拓展市场。它们将 ArcObjects 中的一些组件单独打包出来，并把它命名为 ArcGIS Engine(AE)。AE 是一套用于构建应用的嵌入式 GIS 程序的组件库，使用它开发的程序不需要安装桌面版程序，它运行只需要购买单独的 Runtime 就可以了。这套产品在灵活性和费用上都比较有优势。除此以外，AE 还提供了 Java API 供 Java 程序员使用，这点是使用 Desktop 版本的 ArcObjects 开发所不支持的。

GIS 服务器可以通过网络发布地理信息，它包括 ArcGIS Server、ArcIMS 和 ArcSDE 三种软件。ArcGIS Server，通常用于建构企业级别的互联网 GIS 应用；ArcIMS 则是一个可定制扩展的，能够在网络上发布地理信息的网络地图发布系统；ArcSDE 是一个空间数据引擎，它可以用于管理关系数据库，以实现地理数据的海量存储等高级特性。

移动 GIS 提供了移动领域内的 GIS 应用方案。ArcPad 提供了从地理信息创建

到访问的功能。

学习 ArcObjects 的程序员，必须熟练掌握 ArcGIS 软件的一般操作，尤其是 ArcMap 和 ArcCatalog 两个软件，这对于 ArcObjects 的开发学习是大有裨益的。

1.1.2 ArcObjects 的组件库

ArcObjects 是一套庞大的 COM 组件集合，ArcGIS 9.0 与 9.1 各种 COM 类型统计见下表：

	Enum	Struct	Interface	Coclass	Total
ArcGIS 9.0	628	35	3029	2322	6014
ArcGIS 9.1	991	37	3986	3051	8065

Note: Hidden or restricted types in COM type library are not included. Please refer to Migrating from 9.0 to 9.1 for additional information.

为了更好地管理 ArcObjects 中数目众多的 COM 对象，ESRI 将它们放置在不同的组件库中，从.NET 的角度看，它们被组织到不同的命名空间中。

组件库是对一个或多个 COM 组件中所有的组件类、接口、方法和类型的描述，这种描述是二进制级别的。所有的这些组件库组件都位于<ArcGIS 安装目录>＼com 文件夹中，而其真正的实现在<ArcGIS 安装目录>＼bin 文件夹的众多 DLL 文件中。

C#.NET 的命名空间(NameSpace)以一种逻辑分层的方式来组织元素的方法。对于 ArcObjects 中众多的 COM 对象，当程序员需要使用它们的时候，必须记住每个对象的名字，这是非常困难的。因此，许多高级程序语言提供了一种逻辑上的聚合方法，以实现更高层次的组件管理。

命名空间将功能相同或者相似的 COM 对象在逻辑上松散组织起来。在 ArcObjects 中，众多的组件放在不同命名空间。如果要进行地理数据操作，需要引入 GeoDatabase 等相关的命名空间；如果涉及几何形体对象的处理，就需要引入 Geometry 等命名空间。这种方法让程序员在寻找具体的 COM 对象时更有目标性。

ArcGIS Desktop 版本的 ArcObjects 核心对象被放在 53 个组件库中，不同的组件库的功能是不一样的。作为一个程序员，熟悉所有的 ArcObjects 组件库是一件不可能的事情。但是读者应该在了解一些最基本的组件库后，在将来的实际开发中继续学习需要掌握的组件库。

在这些命名空间中，有一些是经常使用到的，如 Carto、Geometry、system、systemUI、FrameWork 等，需要程序员熟练掌握。如果不记得某个接口或对象属于

哪个命名空间的时候，可以通过开发帮助查找到。

学习 ArcObjects 的过程，也就是不断了解这些组件库本身以及库与库之间关系的过程。本节将介绍一些最核心的组件库，以给读者大致了解 ArcObjects 提供帮助。

- **System 库**

System 库（即 ESRI.ArcGIS.esriSystem 名称空间）是 ArcGIS 框架中最底层的一个库，它提供了一些可以被其他组件库使用的组件。这些组件都是非常基本的，如数组(Array)、集合(Set)、Xml 对象、Stream 对象、分级(Classify)对象和数字格式(NumberFormat)对象等。

数组和集合都是基本的数据单元；XML 对象则给 ArcObjects 提供了操作 XML 类型文件的能力；Stream 对象可以将数据以流的形式保存为任何格式的文件。

分级对象和数字格式对象都和数值数据有关。前者是使用统计函数将数值数据进行不同类型的分级，这个对象大多使用在分级着色中；后者可以让输出的数值的格式互相转变，如角度转弧度、设置小数点等。

- **SystemUI 库**

SystemUI 库（即 ESRI.ArcGIS.SystemUI 名称空间）定义了一些被 ArcGIS 用户界面组件所使用的对象，如 ICommand、ITool 等。在第二章中将专门介绍这些程序界面定制的内容。

- **Geometry 库**

Geometry 库（即 ESRI.ArcGIS.Geometry 名称空间）包含了核心的几何形体对象，如点、线、面等，即在 ArcObjects 中的要素和图形元素的几何形体都可以在这个组件库中寻找到。除此以外，这个库还包含了空间参考对象，包括 GeographicCoordinateSystem(地理坐标系统)、ProjectedCoordinateSystem (投影坐标系统)和 GeoTransformations(地理变换)对象等。

几何形体对象和空间参考内容，都是 ArcObjects 中比较重要的部分，本书将有专门的章节讲述。

- **Display 库**

Display 库（即 ESRI.ArcGIS.Display 名称空间）包含在输出设备上显示图形所需要的组件对象，它包括 Display 对象、Color 对象、ColorRamp 对象、DisplayFeedback 对象、RubberBand 对象、Tracker 对象和 Symbol 对象。

这个库中的对象主要负责 GIS 数据的显示，如 Color 和 ColorRamp 对象可以产生颜色对象，它配合 Symbol 对象，可以对地理数据进行符号化操作，以产生丰富

多彩的地图。Symbol 对象是用于修饰几何形体对象的，任何一种几何形体都必须使用某种 Symbol 才能显示在视图上。

Display 对象直接管理地理数据的绘制和显示，DisplayFeedback 则是 ArcObjects 中可以使用鼠标与地理视图进行交互的对象，它的内容非常庞大，可以用于绘制图形或移动图形等高级任务。RubberBand 对象则相当于一个“橡皮筋”，可以用于在 Display 上绘制丰富的几何形体对象，如 Circle、Rectangle、Polyline 和 Polygon 等。

- **DisplayUI 库**

DisplayUI 库（即 ESRI.ArcGIS.DisplayUI 名称空间）提供了具有可视化界面的对象用于辅助图形显示，它包括 Property Pages (属性页)对象和 StyleGalleryClass 对象，前者可以用于设置 Symbol 对象，而后者则可以用于管理和获取 Style (样式)和 Symbol (符号)对象。

- **Controls 库**

Controls 库包含了在程序开发中可以使用的可视化组件对象，如 MapControl、PageLayoutControl 等，Controls 库分为以下 7 个子库：

- MapControl：对应 ESRI.ArcGIS.MapControl 名称空间
- PageLayoutControl：对应 ESRI.ArcGIS.PageLayoutControl 名称空间
- TOCControl：对应 ESRI.ArcGIS.TOCControl 名称空间
- ToolbarControl：对应 ESRI.ArcGIS. ToolbarControl 名称空间
- ControlCommands：对应 ESRI.ArcGIS. ControlCommands 名称空间
- ReaderControl：对应 ESRI.ArcGIS. ReaderControl 名称空间
- LicenseControl：对应 ESRI.ArcGIS. LicenseControl 名称空间（ArcGIS 9.1 版本才开始提供）

本书后面有专门章节详细阐述 ArcGIS 控件及应用开发。

- **ArcMapUI 库**

ArcMapUI 库（即 ESRI.ArcGIS. ArcMapUI 名称空间）中的对象为 ArcMap 程序提供了某些可视化的用户界面，这些对象不能在 ArcMap 之外使用。IMxApplication 和 IMxDocument 接口都被定义在这个库中，但是它们的实现却在 ArcMap 库。ArcMap 的 TOC（Table Of Contents--内容表）对象也是在这个库中被实现的。

程序员可以扩展这个库的内容，为 ArcMap 程序产生自定义的命令或工具。

- **Framework 库**

ArcGIS 程序存在一个内在的框架，所有的 ArcObjects 组件对象都在这个框架中扮演了不同的角色，它们的协作可以完成 ArcGIS 提供的 GIS 功能。这个框架中的某些核心对象被放置在 Framework 库中。

Framework 库（即 ESRI.ArcGIS.Framework 名称空间）提供了 ArcGIS 程序的某些核心对象和可视化组件对象。这个库中的一些对象可以让 ArcGIS 程序扩展它们的定制环境，以改变 ArcGIS 程序的外观界面。同时，这个库中也提供了诸如 ComPropertySheet、ModelessFrame 和 MouseCursor 等对象，它们是一些对话框，用于在 ArcGIS 上实现用户的交互。

Application 对象是 ArcGIS 程序的核心，它掌握着 ArcGIS 程序的生命周期和管理扩展对象；DockableWindows 是 ArcGIS 中的可停靠窗体；TOC 对象就是一种可停靠窗体，它也被定义在这个库中。CommandBars 和 Commands 对象也在这个库中定义，它们可以用于用户定制某些命令，这些内容将在第二章中详细探讨。

Framework 库不能被扩展，但是程序员可以通过实现在库中定义的某些接口来使用 UI 组件扩展 ArcGIS 程序。

- **Carto 库**

Carto 库（即 ESRI.ArcGIS.Carto 名称空间）包含了为数据显示服务的各种组件对象，如：MapElements (包含 Map 对象的框架容器)；Map 和 PageLayout (地理数据和图形元素显示的两个主要对象)；MapSurrounds (与一个 Map 对象相关联的用于修饰地图的对象集)；Map Grids (地图网格对象，用于设置地图的经纬网格或数字网格，起到修饰地图的作用)；Renderers (着色对象，用于制作专题图)，Labeling，Annotation，Dimensions (标注对象；用于修饰在地图上产生文字标记以显示信息)，Layers (图层对象，用于传递地理数据到 Map 或 PageLayout 对象中去显示),MapServer, ArclMS Layers、Symbols 和 Renderers，GPS Support 等。

- **CartoUI 库**

CartoUI 库（即 ESRI.ArcGIS.CartoUI 名称空间）中的对象也是为了数据显示服务的，在 ArcObjects 中所有的以 UI 结尾的库中的对象都具有可视化的界面。CartoUI 库中包含了诸如 IdentifyDialog、SQLQueryDialog、QueryWizard 等对象，这些对象都以一个对话框的形式出现，本书会在后面章节中讨论如何使用 IdentifyDialog 对象来获取地理对象的信息。

- **GeoDatabase 库**

CeoDatabase 库（即 ESRI.ArcGIS.CeoDatabase 名称空间）中包含的 COM 对象是用于操作地理数据库的，地理数据库是一种在关系型数据库和面向对象型数据库基础上发展起来的全新的地理数据库模型，它被称为“第三代地理数据库”。

这个库中的对象包括核心地理数据对象，如 Workspace (工作空间)、Dataset (数据集)等；它也包含了几何网络、拓扑、TIN 数据、版本对象、数据转换等多方面的丰富内容。

这些对象由于数目众多，而且在 ArcObjects 中占据极为重要的位置，本书在后面有专门的章节对它们进行介绍。

- **DataSourcesFile 库**

地理数据保存在不同形式的文件中，如 Coverage、shapefile 和 CAD 文件都是以普通二进制文件的形式保存的，为了在 GIS 程序中获取这些格式的数据，需要使用特定的 WorkspaceFactory (工作空间工厂)对象来打开这些数据。DataSourcesFile 库（即 ESRI.ArcGIS. DataSourcesFile 名称空间）中的对象正是起到打开文件格式地理数据的作用。

- **DataSourcesGDB 库**

DataSourcesGDB 库(即 ESRI.ArcGIS.DataSourcesGDB 名称空间)中的 COM 对象用于打开数据源为 Access 数据库或任何 ArcSDE 支持的大型关系数据库的地理数据。这个库中的对象不能被开发者所扩展。

DataSourceGDB 库中的主要对象是工作空间工厂，一个工作空间工厂可以让用户在设置了正确的连接属性后打开一个工作空间，而工作空间就代表了一个数据库，其中保存着一个或者多个数据集对象。这些数据集包括表、要素类、关系类等。这个库中的对象主要有 AccessWorkspaceFactory，用于打开一个基于 Access 数据库的 Personal GeoDatabase；ScratchWorkspaceFactory，用于产生一个临时的工作空间，用于存放选择集对象；SdeWorkspaceFactroy，用于打开 SDE 数据库。

- **DataSourcesOleDB 库**

DataSourcesOleDB 库（即 ESRI.ArcGIS.DataSourcesOleDB 名称空间）中的对象具有专门的 API 函数，用于操作任何一种支持 OlE DB 的数据库。这些数据库类型非常丰富。在这个库还可以使用 TextFileWorkspaceFactory 对象来打开一个文本文件，这对于 GIS 载入某些文本数据，如坐标文件等非常有用。

除此以外，DataSourcesOleDB 库还提供了一种使用 ADO 连接已经存在的工作空间的方式，它是一种高效数据获取方法。

- **DataSourcesRaster**

DataSourcesRaster 库（即 ESRI.ArcGIS.DataSourcesRaster 名称空间）中的 COM 对象用于获取保存在多种数据源中的栅格数据，这些数据源包括文件系统、个人地理数据库或者企业地理数据库(SDE 数据库)。这个库还提供了用于栅格数据转

换等功能的对象。

- **Output 库**

Output 库（即 ESRI.ArcGIS.Output 名称空间）包含了 ArcObjects 所有的输出对象，它包括两大类，即打印输出对象 Printer 和转换输出对象 Export。前者可以将视图上的地图通过打印机进行输出，而后者包含的丰富对象，可以将地图转换为多种格式的矢量或栅格形式的数据，如 EMF、PDF、JPEG、TIFF 等。

在 ArcObjects 组件库中，除了 SystemUI 库外，其他以 UI 结尾的库都是属于 Desktop 版本专用的，它们负责实现 ArcGIS Desktop 程序的“用户界面”，只能用于基于 Desktop 版本的开发之中。

介绍 ArcObjects 中的这些组件库，是为了让读者对 ArcObjects 庞大的结构有个感性的认识，初步了解自己需要解决的问题可能在什么地方得到答案。对于 ArcObjects 组件库的深入学习，理解库内众多对象之间的关系、库与库之间的关系，是贯穿于本书的一条主线。ArcObjects 的学习，就是为了了解这些关系，只有掌握了这条脉络，ArcObjects 程序员才能够提纲挈领，更快地理解 ArcObiects 的方方面面。

1.1.3 理解对象模型图

要学习基于 COM 标准的 ArcObjects，阅读对象模型图 (Object Model Diagram，OMD)是基本的功力，通过阅读 ArcObjects 的 OMD，可以很快地熟悉 ArcObjects 的结构和不同组件之间的关系。

OMD 是基于 UML(Unified Modeling Language，统一建模语言)基础上的，它补充了在对象浏览器中看不到的信息。OMD 好比城市的道路图，可以帮助程序员来了解类之间的关系，如何从一个类到另一个类，选择正确的接口，获取需要的属性和方法等，这对于走出 ArcObjects 的迷宫有非常重要的帮助。

ESRI 提供了多种方法让用户了解 ArcObjects 组件及不同组件之间的关系，其中一种是它自己开发的对象浏览器(ESRI Object Browser)。在 VS.NET 环境中，可以使用对象浏览器来查看 ArcObjects 组件库中各种对象有关信息，对象浏览器如图 1-2 所示。

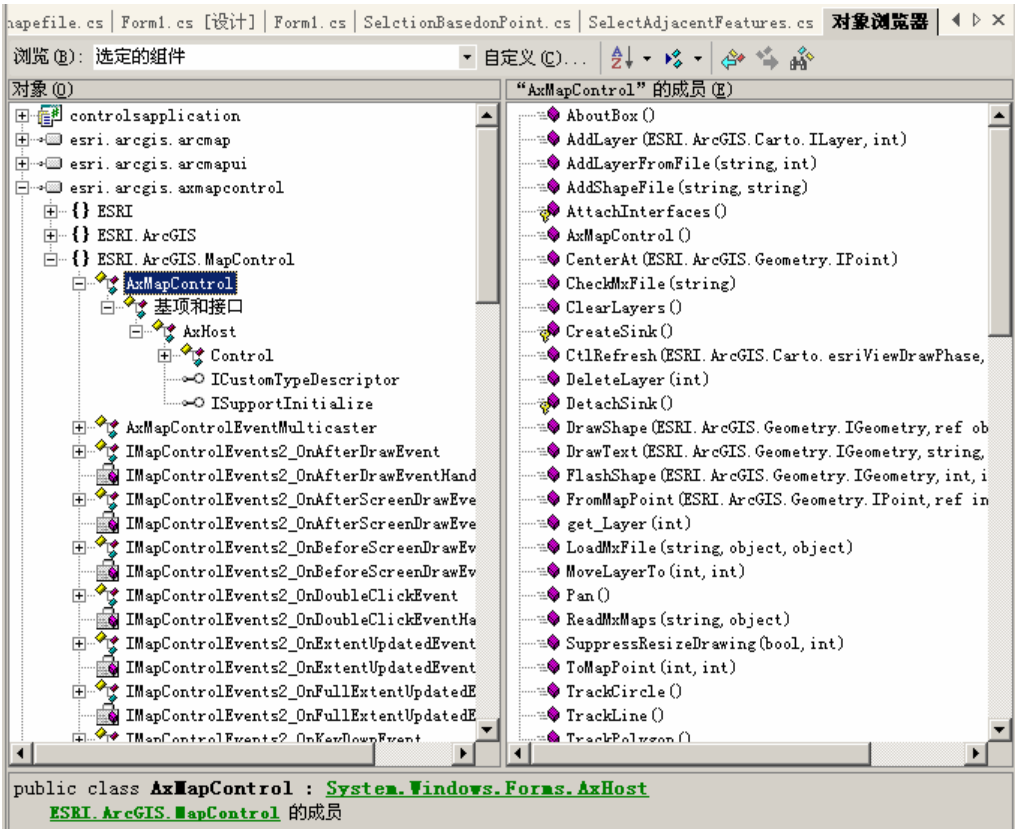


图 1-2　ESRI Object Browser

另一种方法就是 ESRI 提供的一系列 OMD 的 PDF 文件。这些文档以更加直观的方法揭示了 ArcObjects 的内部结构。ArcObjects 程序员在学习 ArcObjects 二次开发中，应该养成一个良好的阅读 OMD 的习惯。灵活地使用对象浏览器，开发帮助和 OMD 图，是熟悉和掌握 ArcObjects 的最佳途径。

1.1.3.1 类与对象

在 ArcObjects 中存在三种类型的类：抽象类(Abstract Class)，组件类(CoClass)和普通类(Class)。

抽象类不能用于产生一个新的对象，但是可以用于定义一个子类。

组件类是一个可以直接创建对象实例的类，它的实例对象不依赖其他对象的存在而存在，其生存周期也不用其他的对象管理。在 C#.NET 环境中，可以使用这样的语句：

```
IMap pMap = new MapClass();
```

这里的 MapClass 就是一个组件类，它可以使用 new 关键字来产生一个对象 pMap。

普通类不能够直接产生一个对象，但它可以用其他的普通类或组件类的方法产生，而不是使用 new 关键字来完成。如下面的例子，A 和 B 为普通类对象，C 为组件类对象，D 为一个普通类，如下面的代码：

A = B.writeA ( );　　正确

A = C.writeA ( );　　正确

A = new　D();　　不正确，普通类不能通过 new 的方法产生

A 可以由 B 和 C 的 writeA 方法产生，A 的生命周期是由产生它的对象 B 和 C 控制的，如果 B 或 C 对象在内存中被释放了，则此对象也将从内存中消失。

上面的例子对于读者了解这三种类之间的关系是非常有用的。很多初学者在写代码的时候往往搞不清楚他们能否直接产生一个对象出来，看了上面的介绍，应该可以理解其中的关系了。

1.1.3.2 类与类的关系

ArcObjects 的类之间存在着不同的关系，它们通过 UML 都可以描述出来。这些关系主要分为四种，即依赖关系、关联关系、组合关系和类型继承。图 1-3 是 OMD 中的类结构关系，它们是理解整个 OMD 的基础和掌握 ArcObjects 的关键。

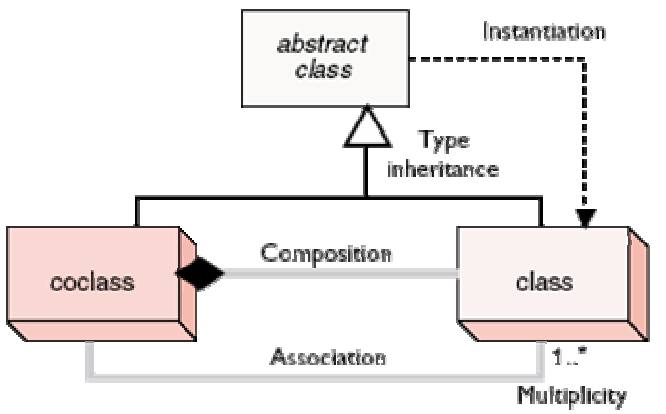


图 1-3　ArcObjects 类结构关系图

- **依赖关系**

如图 1-4 所示，它表明一个对象有方法产生另一个对象，如 Pole 有一个方法可以产生一个 Transfomer 对象。当 Pole 的状况发生变化的时候，Transformer 也会发生变化，如果 Pole 消失，那么 Transfomer 对象也会消失掉。即前者的生命周期决定后者的生命周期。

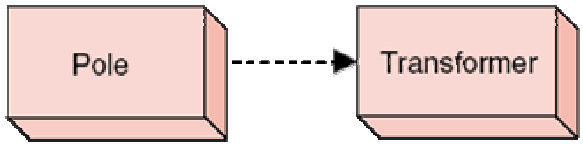


图 1-4　依赖关系

在 ArcObjects 中，WorkspaceFactory 有 3 个方法（Create、Open 和 OpenFromFile ）来创建或打开一个 Workspace 。Workspace 依赖于 WorkspaceFactory。

- **关联关系**

如图 1-5 所示，图中的两个对象是松散的关联关系，可以从一个类的对象访问到另一个类的对象。如一个土地所有者可能有多块土地，一块土地也可能被多个所有者拥有。当其中一个不存在时，另一个不会消失。

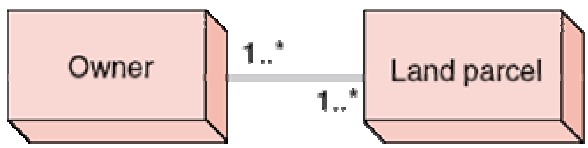


图 1-5　关联关系

关联关系是有方向的。如果只存在一个方向，称为单向关联；如果是两个方向都存在关系，则称为双向关联。

在 ArcObjects 中，关联关系也很普遍，如 Workspace 与 Dataset、Table 与 Fields、MapControl 与 Map 的关系等。

- **组合关系**

如图 1-6 所示，图中的两个对象属于紧密的组合关系。Crossarm 是 Pole 的一部分，当 Pole 消失的时候，Crossarm 也将不复存在。读者可以认为左边的 Pole 是一个集合，右边的 Crossarm 是这个集合内的一个对象，当集合为 null 的时候，集合内的对象也将是 null，即 Crossarm 的生命周期由 Pole 对象控制，但 Crossarm 的产生不受 Pole 对象的控制。

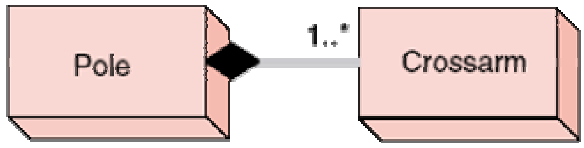


图 1-6　组合关系

组合关系在 ArcObjects 中很常见，如地图文档由一个或多个地图组合而成；地图（Map）由图层（Layer）组成，工具条控件（ToolbarControl）由工具条项（ToolbarItem）组成等等。

- **类型继承**

类型继承是任何一本面向对象语言教材中都会讲解的内容。在图 1-7 中，line 是一个抽象类，Primary line 和 Secondary line 是它的子类。Line 作为一个抽象类，不能直接产生一个对象出来，只能通过产生一个子类的方法实现自己，子类将继承父类全部的非私有方法和属性，而且子类对象可以看做是父类对象的一种。

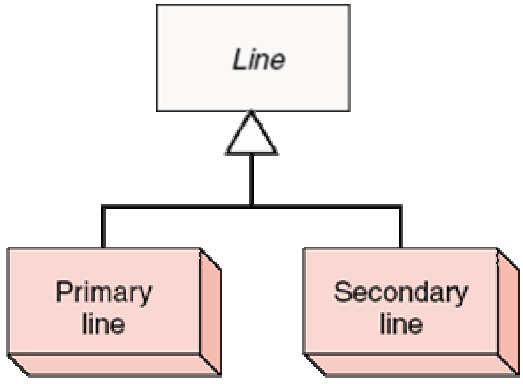


图 1-7　类型继承

类之间的继承可以看做是类功能的扩展，即子类在保存父类的属性和行为的基础上会增加自己特有的行为和属性。正是这种扩展使得对象具有了更强的生命力和使用范围。

1.2 组件对象模型

1.2.1 软件开发历史

在计算机技术的发展过程中，相继出现了多种计算机高级语言，从 Pascal、Basic、C、Smalltalk、C++、Java 到.NET 平台上的各种语言，它们的发展步伐，可以看做是计算机语言从面向过程向面向对象发展的一段历史，也是不断提高软件重用和开发效率的历史。在早期（面向过程语言），人们为了重用，将一些基本的数学计算函数和界面设计函数设计成函数库，或者称为类库，通过应用程序接口让其他软件开发人员调用。它为应用开发者提供了方便，但是它的粒度太小，往往一个函数就是一个单一的功能，它相当于机械制造中的零件，利用零件组装一辆汽车太费时间。

面向对象语言的出现，其实就为一个理由--提高编程的粒度。面向对象语言的基本单位是类(Class)，它封装了数据成员(属性)和函数成员 (方法)，将最小组件的粒度提高了一个等级，程序员需要直接操作的不是过程和函数了，而是更高层次上的类。但是，做成了 Class 并没有解决编程中所有问题，新的问题随之而来。一个类，它提供了很多种方法和属性，这些方法和属性其实可以分组，为不同的功能服务，但是类并没有做这个管理，它只是一个属性和方法的容器。在 ArcObjects 中，Map 对象拥有很多类型的功能，像管理图层、管理元素、管理选择集、显示地图等，每种不同的功能群都有好多方法和属性，如果这些属性和方法是杂乱无章，没有任何区别地堆积在一个类里面，当程序员需要寻找一个方法的时候，不得不遍历它们，这样做很不方便。

接口(Interface)的出现，解决了这个问题，就是将类的内部属性和方法进行分类。例如在 Map 类中可以做好几个接口，在这些接口中定义不同功能组的方法和属性，Map 实现了这些接口，这样就可以使用接口进行定义，如：

```
IGraphicsContainer pGraphicsContainer ;
pGraphicsContainer = pActiveView.FocusMap as IGraphicsContainer ;
```

pGraphicsContainer 对象现在可以使用的属性和方法就只能是 IGraphicsContainer 接口定义的那部分，而不能使用其他接口定义的方法和属性，那如何使用其他接口定义的属性和方法呢？这就是所谓的 QI(QueryInterface)功能，即从对象的一个接口查询另一个接口定义的属性和方法，如：

```
IActiveView pActiveView;
pActiveView = pGraphicsContainer as IActiveView;
```

通过上面的操作，pActiveView 现在就可以使用 Map 类中 IActiveView 接口定义的属性和方法了，这就实现了在一个类的不同接口之间的转换。

接口是一种用来定义程序的协定。实现接口的类要与接口的定义严格一致。有了这个协定，系统就可以抛开编程语言的限制。接口可以从多个父接口继承，而类可以实现多个接口。接口可以包含方法、属性、事件和索引器，它本身并不提供它所定义的成员的实现，而只是指定实现该接口的类或接口必须提供的成员。在可以

使用类的地方，都可以使用接口来替代，除了使用类产生一个对象外。

接口可以看做是一个特殊的类形式，除了不能被实例化为一个对象外，它可以实现类能够完成的任何任务，如声明对象为某种接口类型，接口也可以继承等。接口继承机制是非常有用的，如一个子类对象可以看做一个父类对象，接口也具备这样的特性，在很多时候程序员可以将一个子接口类型的对象定义为父接口类型的对象，从而实现一般化的操作，如：

```
Private void CreateGeometry (IPolygon pPolygon)
Private void CreateGeometry (IGeometry pPolygon)
```

上面的两个过程的参数一个是使用 IPolygon 对象，另一个是使用 IGoemetry 对象，后者可以使用得更广泛和更安全一点，如果不慎传进去的是一个 IPoint 对象，在第二个方法里面也是合法的，因为 IGeometry 的对象可以是任何一种几何形体对象，而这种做法在第一个过程中就是错误的。

一个类可以实现多个接口，一个接口也可以被多个类实现。

计算机语言的发展历史，其实就是一部不断地寻找更好组件粒度的历史，不断提高代码重用的历史。以前程序员使用过程和函数，后来使用类，接口乃至包、命名空间等，都是为了一个目的，那就是让程序员能够操作的组件在具体和抽象之间寻找一个平衡点，这不是一件容易的事情——太具体了，如过程和函数，就没有了框架；太抽象了，如类，就无法深入细微之处。

1.2.2 什么是组件对象模型

对于软件开发者来说，软件重用和开发效率始终是一个核心问题。程序员希望能够编写一次代码，在任何地方都可以运行，即使这个使用环境连代码编写者都没有想到过。当一个程序员修改了自己发布给别人使用的函数后，使用者应该不需要改变或者重新编译程序就可以使用新的功能。对实现这一目标的早期的努力是使用类库，这种做法在 C++中比较常见，但是它有很大缺陷，要共享 C++的二进制代码并不容易。为了解决这个问题，程序界开始试图建立一种组件(Component)标准去实现代码在二进制级别上的共用。组件将数据和操作数据的方法隐藏在被封装好的接口后面，它保证了系统的安全性。由于组件是面向对象的，因而它们支持继承性和多态性。继承性是一种可以重用其他组件功能的机制，而多态性则保证了一个组件可以在不同的环境中被正常使用。

尽管存在多种组件标准，如 COM、JavaBean 和 CORBA，但微软提出的 COM 被认为是开发高效、交互式桌面程序和服务器 / 客户端程序的最好选择。它具有良好的定义，并且标准成熟，易于理解，并且微软开发了很好的开发工具。

微软的 COM 模型是一种二进制标准，它允许任意两个组件按照一定规则互相通信，而不管它们是在什么计算机上运行(只要计算机是相连的)，不管组件所在计算机运行的是什么操作系统(只要该系统支持 COM)，也不管该组件是用什么语言编写的(只要该语言支持编写的组件遵守 COM 标准)。COM 还提供了位置透明性，当程序员编写组件时，其他组件是在进程内的 DLL、本地的 EXE 还是位于其他计算机上的组件，对程序员而言都无所谓。

COM 模型可以看做是一种客户端 / 服务器的关系。在 COM 中，一个组件如果可以提供服务，它就是一个 COM 服务端。服务端将提供客户端所需要的功能，两者之间是隔绝的，客户端需要知道的只是哪些功能可以使用。在这种情况下，COM 表现为两者之间的通信协议。

遵守 COM 标准的类的实例称为 COM 对象，COM 对象有三个公认的特点：抽象、多态和继承。

抽象是指 COM 对象被很好地封装起来，程序员无法获得对象的内部实现细节。实际上，ArcObjects 程序员也并不需要知道这些细节，除非想窥探到程序的核心算法。对象的封装是如此的严密，以至于 COM 对象通常被描绘为盒子。尽管实现细节不会告诉使用者，但程序员可以通过接口来访问 COM 对象中的相关的属性和方法。即 COM 对象之间并不能够直接联系，它们是通过接口来接触的。

多态是指可以通过同样的方法去处理不同类型的对象，这些对象将能够正常运行，这是因为不同的对象会以不同的方式去实现这些方法。当外部访问某个对象的时候，对象会自动知道自己该调用什么方法去完成任务。

继承分为实现继承和接口继承两种类型。实现继承是子类对象可以从其父类对象那里继承其程序，当外界调用子类中继承的方法的时候使用的其实是父类的方法，在 C++中就是使用的这种继承；对于接口继承而言，对象继承的只是其父类方法的定义，对象必须自己去完成这些方法。

COM 有多种扩展型态，例如 DCOM 与 COM+，其中 D 表示为 Distributed，即它可通过网络获得 COM 为基础的软件；COM+是 COM 的第二代，它拥有 DCOM 与服务器的技术。

ActiveX 与 OLE 也是 COM 别称，微软把所有以 COM 为基础的技术统称为 ActiveX，而 OLE(嵌入式链接对象)是一种以 COM 为基础而发展的复合文件(compound document)，可用来存取不同应用程序的文件，例如在 word 文件中存取 excel 表格。

概括地讲，COM 具有如下一些优越性：

第一，COM 技术让编程技术难度和工作量降低，开发周期变短，开发成本降低。

由于组件式开发实现分层次的编程，从而促进了软件的专业化生产。专业人员可以开发出具有很强专业性的软件组件，这样既保证了普通的编程人员能够完成所需要的应用开发，又不至于降低使用的性能。应用人员不便实现的组件模块可以让专业人员定做。ESRI 的程序员们使用 C++语言开发了一个个 COM 组件给二次开发者使用，用户可以像堆砌积木一样按照需求开发不同的 GIS 程序。

第二，COM 使软件的复用性得到了提高并延长了使用寿命。由于组件编程体系使大量的编程问题局部化，这让软件的更新和维护变得快速和容易，软件的成本也大大降低。当程序员让系统升级后，可以覆盖以前的 COM 组件，以让程序使用上最新的功能。当然这个操作是具有风险的。

第三，COM 对象是语言独立的，可以使用任何一种语言去编写 COM 组件，而使用这些组件所用的语言也只需要支持 COM 标准即可，不必和组件的编写语言一致。

在 ArcObjects 编程中，程序员只是使用别人做好的“积木块”，并不需要用深奥的 ArcObjects 知识来编写 COM 对象，学习 ArcObjects 主要知道以下内容即可：

第一，COM 不是接口，也不是类，它是一种二进制级别的组件通信标准。它告诉组件之间该如何通信、一个 COM 对象之间的不同接口如何查询等。

第二，符合 COM 标准的对象--COM 对象，是实现了很多接口的对象，也是基于面向对象标准的。COM 对象可以以 DLL 或者 EXE 文件形式存在，它包含着接口的具体实现方法。使用者可以通过接口来获取它内部的函数或者方法。COM 对象的接口一旦被公布，就不能再修改，这一点很重要。

COM 对象可以看做一个大仓库，其中摆放着用户可以使用的对象(方法和属性)，这个仓库排放的物品如此之杂，以致必须对它们进行登记造册，将相近或相似的物品登记在一起，并由专人来管理这一类的东西。而接口就承担了这个任务。

当外部用户需要取用这个仓库的物品时，必须找到这些物品的管理者(接口)，通过他来实现要求，这个管理者仅仅能拿出自己权限范围内的东西而不能越俎代庖(通过一个接口仅仅能使用它定义的属性和方法)；当外部用户又需要这个仓库中另一类物品的时候，必须去寻找另一个管理员，这个过程就是所谓的“接口查询”。外部用户需要保证两个管理员都是这个仓库的合法管理者(两个接口是同一个 COM 对象的接口)。

第三，COM 对象必须实现 IUnknown 接口，它负责管理 COM 对象生命周期并在运行时刻提供类型查询，当 COM 对象不使用的时候，是这个接口定义的方法负责释放内存。一个 COM 对象可以没有任何别的接口，但是这个接口是必需的，它是缺省的接口。

第四，QI，即所谓查询接口。由于一个 COM 对象有很多个接口，不同的接口管理着 COM 的不同类型的方法，因此从一个接口可以使用的方法转到另一个接口可以使用的方法的过程称为 QI。这个过程也是由 IUnknown 接口管理的。

第五，缺省接口。每一个 COM 对象在产生后如果没有指定接口的话，它们都有一个缺省接口。ESRI 对象库中的 COM 对象都使用 IUnknown 作为它们的缺省接口，当然也有点例外。如 Application 对象使用的缺省接口是 IApplication。

第六，每个组件都有一个独一无二的标识，这就是所谓的全球唯一标识符 GUID。这个标识符就是 COM 组件的身份号码，是供机器使用的，它是一个 128bits 的数字，由系统自由分配。用户不用担心这个标识会有重复的一天，因为即使计算机每秒产生 1000 万个 UID，那么到公元 5770 年才可能遇到重复的 UID。组件还有一个供用户使用的标识，它在程序集内部是一个字符串，在 ArcObjects 中常常可以使用 UID 对象来引用这些组件。

接口的 GUID 称为 IID，而组件类的 GUID 称为 CLSID。CLSID 还有一个文本别名，即所谓的 ProgID，它是一个由工程名加组件类类名组成的字符串。

第七，一个 COM 对象可以有多个接口，一个接口也完全可以被多个 COM 对象实现。

第八，接口分为两种，即内向接口和外向接口。前一种接口是组织 COM 对象相关的方法和属性，COM 对象必须实现所有的接口内容；后一种接口是用于组织 COM 对象相关的事件，这种接口在实现的时候不必写出所有的事件。

第九，COM 组件必须被注册后才能使用，它必须到注册表那里去登记“户口”。

第十，在 COM 对象的开发过程中，程序员可以逐步加上接口，在 ArcObjects 中有很多以“2”结尾的接口，都是这样发展的产物。比如 ILayer2、ITable2 等。它们是在后来的版本逐渐添加的。

第十一，COM 对象可以被编译为 DLL 和 EXE 两种格式的文件进行传播。

COM 组件模型有很多优点，但它也有致命的缺陷。由于 COM 对象可以被重用，这样多个程序可能使用一个 COM 对象，如果这个 COM 组件升级了，就很可能出现其中某个程序无法使用新组件，导致程序不能运行的情况，这种情况被称为“DLL HELL”（DLL 灾难，COM 对象常常被编译为 dll 文件）。有时安装了新软件后很多其他的软件都无法使用，往往就是这个原因。

这并不是个小问题，它可是微软提出.NET 的一个主要原因。本书主要是讲解 ArcObjects 的可视化控件在.NET 平台下的开发，其中的代码绝大部分都将使用 C#.NET 语言进行编写。

1.3 .NET 平台概述

本书采用的开发环境是基于.NET 平台的 C#，所以有必要对.NET 平台作一简要阐述。

Windows 平台虽然是主流操作系统，但在在这个平台上仍然存着很多严重的、难以弥补的缺陷：

- Windows 使用了动态链接库技术。在开发 Windows 平台早期，由于内存和硬盘的容量都很小，微软的程序员们采取了一个聪明的方法--将一些公共组件编写为 DLL 文件注册到操作系统，需要使用这些组件的应用程序不必每个都拥有一份组件拷贝，它们只需在使用的时候从操作系统中调用同一个 DLL 文件就行。这种方法的确节省了硬盘和内存空间，但是它的弊端也非常明显，首先是组件必须“上户口”，需要在注册表中登记；其次 DLL 组件是不可靠的，由于多个程序都使用同一份文件，难免不会出现某个程序将这个组件升级改写后造成其他程序无法正常使用的情况。程序员们将这个情况成为“DLL Hell(DLL 灾难)”，很多人安装新软件后导致其他很多软件无法正常使用的情况大部分都是这个原因。

- 互联网的迅速发展，使得网络成为现代生活中的一部分，但是今天的因特网在很大程度上还在模仿旧式大型计算机的工作方式。尽管有充足的带宽资源，但大量的信息还是被“囚禁”在了中央数据库里。用户必须依靠网络服务器来完成所有的上网操作，网站好像一个个孤立的小岛，并不能按照用户的指令在它们之间进行有意义的交流。今天的网络只能通过单个的网站向用户提供有限的服务，如酒店网站只能告诉用户住宿情况而无法提供交通信息，使用者不得不将个人隐私泄露给他们所访问的每一个网站。用户希望能通过多种方式来获取信息，而不仅仅是浏览器，如果提供这些 Web 服务的应用程序可以由外界通过编程来访问，那么外界就可以更容易且更有效率地使用这套重要的服务。如何在网络上更加方便快捷地传播信息，让互联网不存在“信息孤岛”，如何构建更加安全和强大的网络服务，成为人们追求的目标。

- 可以使用在 Windows 平台上的高级计算机语言很多，如 VB、VC++、Delphi 等，每一门语言都有自己的一套独特语法、一套控制结构、一道数据类型(Data Types)，以及一套面向对象的概念。语言设计所作的决策，会受其目标应用程序(Target Application)的影响，也就是说，“语言使用者可能拿这个语言来干什么”会影响语言当初的设计。这些语言编写的程序之间是不能直接通信的，因为它们使用的都是自己专有的对象库，不同的语言的编译环境也是迥异的。能不能把这些语言统一起来让使用不同语言的程序员

编写的代码能够互相调用，编写界面尽可能统一? VB 简单易学，但在执行效率上无法和 C++匹敌，而且在实现某些高级功能的时候不得不调用那些复杂无比的 APIs；VC++灵活方便，但是其学习曲线又数倍于 VB。怎么样解决这个棘手的问题呢?

针对 Windows 平台存在的这些缺陷，微软提出了全新的.NET 技术体系，其主要包括 .NET Framework 和 Visual Studio.NET 两部分。

Visual Studio .NET 是一套完整的开发工具，用于生成 ASP Web 应用程序、XML Web services、桌面应用程序和移动应用程序。Visual C# .NET、 Visual Basic .NET、Visual C++ .NET 和 Visual J# .NET 全都使用相同的集成开发环境(IDE)，该环境允许它们共享工具并有助于创建混合语言解决方案。另外，这些语言利用了.NET Framework 的功能，此框架提供对简化 ASP Web 应用程序和 XML Web services 开发的关键技术的访问。

.NET Framework 是生成、部署和运行 Web 服务及应用程序的平台。.NET Framework 旨在实现下列目标：

- 提供一个一致的面向对象的编程环境，而无论对象代码是在本地存储和执行，还是在本地执行但在 Internet 上分布，或者是在远程执行的。
- 提供一个将软件部署和版本控制冲突（DLL 灾难）最小化的代码执行环境。
- 提供一个可提高代码（包括由未知的或不完全受信任的第三方创建的代码）执行安全性的代码执行环境。
- 提供一个可消除脚本环境或解释环境的性能问题的代码执行环境。
- 使开发人员的经验在面对类型大不相同的应用程序（如基于 Windows 的应用程序和基于 Web 的应用程序）时保持一致。
- 按照工业标准生成所有通信，以确保基于 .NET Framework 的代码可与任何其他代码集成。

.NET Framework 具有两个主要组件：通用语言运行时（Common Language Runtime--CLR）和.NET Framework 类库。通用语言运行时是 .NET Framework 的基础。您可以将运行时看作一个在执行时管理代码的代理，它提供内存管理、线程管理和远程处理等核心服务，并且还强制实施严格的类型安全检查。事实上，代码管理的概念是运行时的基本原则。以运行时为目标的代码称为托管代码，而不以运行时为目标的代码称为非托管代码。.NET Framework 的另一个主要组件是类库，它是一个综合性的面向对象的可重用类型集合，您可以使用它开发多种应用程序，这些应用程序包括传统的命令行或图形用户界面 (GUI) 应用程序，也包括基于 ASP.NET 所提供的最新创新的应用程序（如 Web 窗体和 XML Web services）。

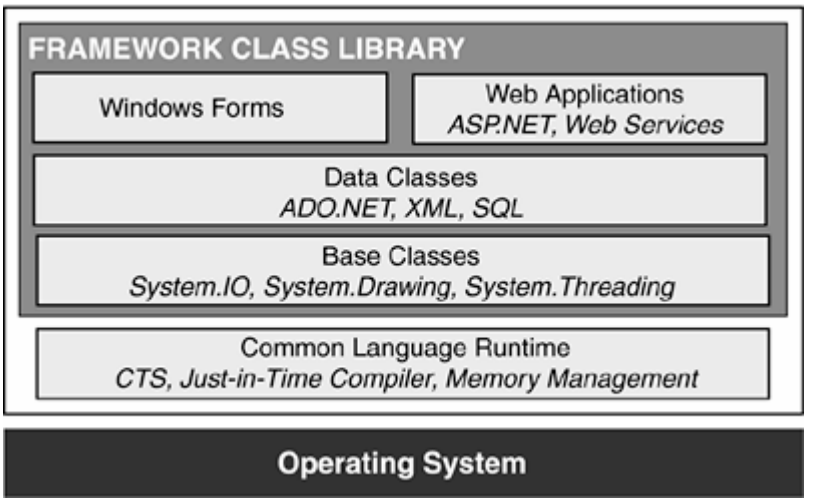


图 1-8　.NET Framework 体系结构

1.3.1 通用语言运行时

通用语言运行时管理内存、线程执行、代码执行、代码安全验证、编译以及其他系统服务。这些功能是在通用语言运行时上运行的托管代码所固有的。

CLR 处于.NET Framework 的最底层，是.NET Framework 的基础，它为C#.NET、VB.NET、C++、J#等多种语言提供了统一的运行环境，使得用某种高级语言编写的应用程序可以在不同的操作系统和硬件上运行，而不需要按照特定环境重写应用程序。

为实现 CLR 对多种编程语言的支持，就需要在传统的源代码层和编译后的本机代码层中间添加一个代码层，即应用程序代码要先编译成独立于机器的中间语言代码，然后编译成与机器对应的本机代码。这样，无论应用程序使用什么语言编写，只要能编译成中间语言代码，就可以在 CLR 的支持下运行。其运行步骤大体如下：由某种语言编写的应用程序，通过编译器（Compiler）生成中间语言 MSIL（Microsoft Intermediate Language）代码，然后由即时编译器 JIT（Just In Time Compiler）生成相应的机器代码。

整个编译执行的过程如图 1-9 所示。

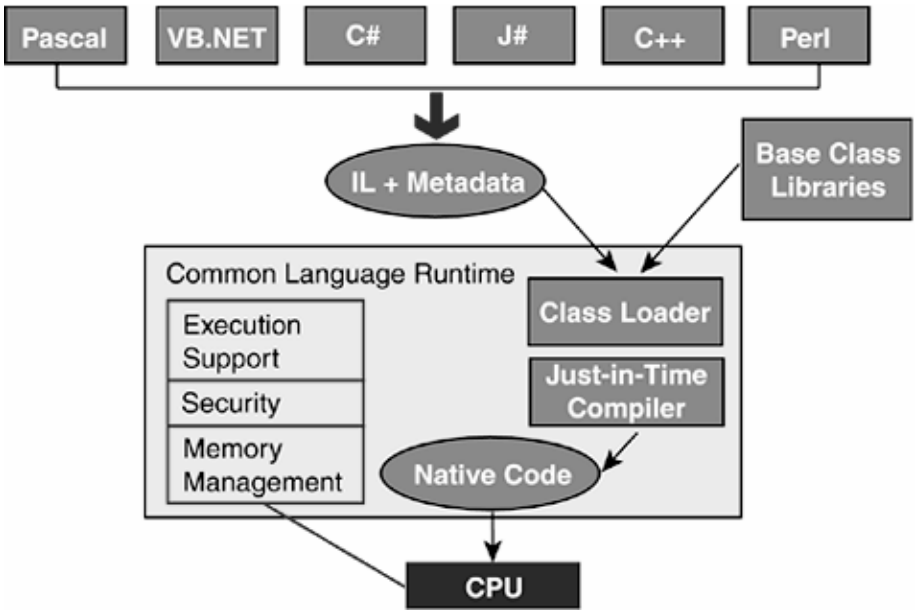


图 1-9　.NET 环境下程序编译执行过程

具体的运行过程是，首先使用 VB、C# 或任一种 CLR 支持的编程语言编写应用程序源代码，然后由编程语言对应的编译器将源代码翻译成一个程序集（Assembly）-- .NET 组件，为一 EXE 或 DLL 文件。这个程序集包括中间语言 MSIL 代码、元数据和一个文件清单，如图 1-10 所示。MSIL 代码也简称 IL 代码，其特点是独立于特定操作系统和硬件平台，这就意味着可以编译成 MSIL 的程序语言是互通的。在.NET Framework 中，任何语言编写的应用程序，都要编译成 MSIL 代码，从而实现应用程序向其它平台的无缝移植。元数据详细描述了 MSIL 代码正确执行所需的各种相关数据类型，包括描述 Assembly 的信息，如标识、输出类型、依赖的其它组装、需要运行的安全许可等；描述类型的信息，如名字、基类、接口实现、成员等；描述其它属性的信息，如描述用户自定义的属性、描述编译器定义的属性、描述 Framework 定义的属性等。这样就使.NET 应用程序代码具备了自描述特性。CLR 使用这种元数据来增强安全性，并确保获得它所需的任何组件的正确版本。清单则列出了程序集中的所有文件和软件组件，同时指出了 CLR 在哪里可以找到应用程序运行时需要的其它组件。

程序集需要在 CLR 的控制下编译成特定于机器的汇编指令才能运行。CLR 在功能上如同一块虚拟的 CPU，它执行 IL 代码、操作数据。为了加载一个应用程序，CLR 使用文件清单来确定应用程序所需程序集的正确版本，并检查 MSIL 代码与描述它的元数据，保证程序不会访问它不具有相应许可的内存或其它资源，从而确认代码是“类型安全”的。然后 CLR 加载 IL 代码，同时收集有关应用程序的事实信息，例如：它是从哪里下载或安装的、它需要执行什么功能、什么用户试图运行它、程

序集是否拥有来自信任的开发者的数字签名，以及进行数字签名后程序集是否有改动，并按管理策略决定是否允许应用程序运行，甚至还可告之应用程序它无法授权申请的所有权限，并允许应用程序自行决定是否继续运行等。之后即时编译器 JIT 将 IL 代码以调用的对象方法为单位动态编译成可执行的本机代码。通过程序集中的元数据，JIT 编译器可以了解代码做些什么以及支持哪些平台，它能够利用目标机器的硬件特点，根据平台的具体情况对代码进行即时优化操作，如合理地分配 CPU 寄存器、通过监控当前的物理和虚拟内存需求来高效地利用内存、产生特定的平台指令以充分利用实际的处理器模式等。经过编译后的代码保存在内存中直到整个过程完成。不同的机器使用不同的 JIT 编译器，所以同一个应用程序的 MSIL 代码经不同机器的 JIT 编译器编译之后，便可以在不同的机器上执行。

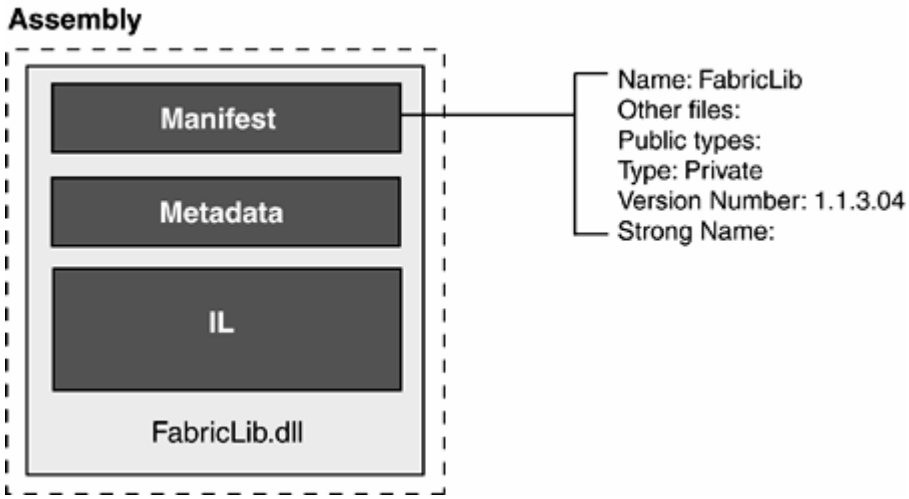


图 1-10　Single file assembly

借助 CLR，用一种语言编写的类能被另一种语言继承，象调试之类的操作也不用考虑编写代码所用的语言，这就意味着编写可重用类库时，不需要为每种语言生成一个版本，使用类库时也不受编程语言的限制，从而使多种语言达到深度集成。

CLR 借助通用类型系统（Common Type System，CTS）对代码类型进行严格的安全检查避免了不同组件之间可能存在的类型不匹配的问题。CTS 中定义的基础类型如图 1-11 所示。

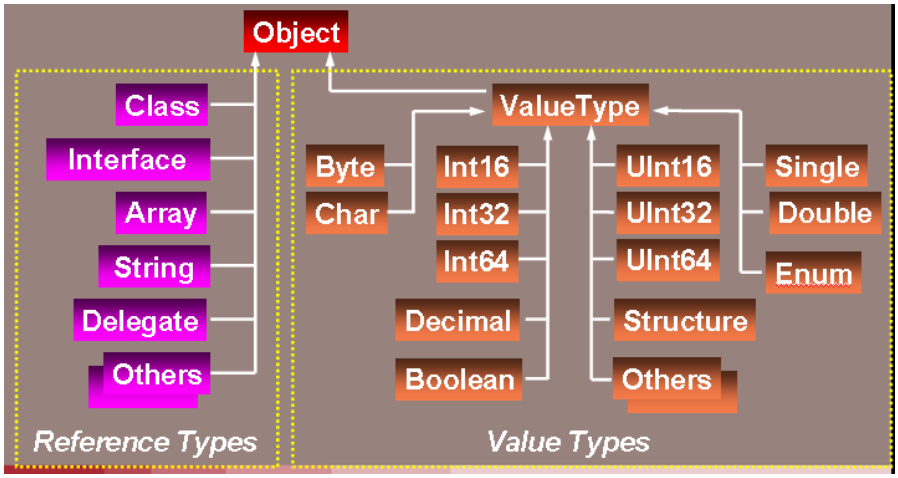


图 1-11 CTS 中定义的基础类型

1.3.2 .NET Framework 类库

.NET Framework 类库(FCL)是一套软件组件，用于编写在 CLR 控制下运行的代码。类库分级组成了一个庞大的功能集，包括显示图形用户界面、读写 XML 数据、在 Internet 上通信、访问数据库等多种功能。

如图 1-8 所示，在 CLR 之上是一套与通用语言运行环境紧密集成在一起的基础类库（Base Classes），所有的类库都建立在基础类库之上。

在.NET 平台上编写的代码，不管使用哪一种语言，它们都会被编译为中间代码而不是直接的机器码，这些中间代码将在 CLR 上被运行。并且，无论使用哪一种语言，它们使用的类库都是 FCL。.NET 还带来了多种语言之间的无缝集成，例如一个系统同时可以采用多用编程语言来开发，C#.NET 编写的类可以让 VB .NET 直接继承使用，这就大幅度提高了开发效率。

基础类库包括集合、字符串、数据、图像、线程等从输入输出到数据访问的各个方面，它提供了一个统一的、面向对象的、层次化的、可扩展的编程接口。要引用类库中的类，可顺着名称空间的层次结构通过“.”号分隔的方法找出具体要使用的类。使用时，只需在应用中添加所需的类库的引用，就可以使用这个类库中的所有方法、属性等等。跟传统的 Windows 编程相比，使用和扩展基础类库都比较容易，这使得开发者能够高效、快速地构建基于互联网的网络应用。

.NET Framework 通过 ADO.NET 提供了对数据的访问。ADO.NET 是位于 System.Data 名称空间中的类集合，它为.NET 平台与后端各类数据源（如 Microsoft SQL Server，通过 OLEDB 桥接的各种数据库，XML 数据源等）交互提供了统一的

编程模型。ADO.NET 定义了连接数据库、对数据库发送命令及从中获取结果的类。这些类依靠数据供应器实现。目前，ADO.NET 只发行了两个供应器，一个是 SQL Server.NET 数据供应器，这是用于 Microsoft SQL Server7.0 或更高版本数据库的供应器，另一个是 OLEDB.NET 数据供应器，这是一个用于管理 OLEDB 数据源的供应器。

ADO.NET 的主要特点是引入了离线数据集（Disconnected DataSet）。所有的数据都可以离线使用而不必一直连接到数据库，只有需要对数据库进行编辑时才连接到数据库。对于 Web 应用程序而言，因为开启数据库联机在分布式的环境中会占用较多资源，所以离线数据存取可使应用程序具有高度的延展性。DataSet 是一个驻于内存的数据缓冲区，是一个分离的用于表示数据集合的独立实体，它提供了数据的关系型视图。不管数据来源于一个关系型数据库，还是一个 XML 文档，DataSet 都使用同样的程序模板和相同的内存区进行操作。它替代了原有的 Recordset 对象，提高了程序的交互性和可扩展性，因而非常适合于结构松散的、本质非链接的 Web 应用程序。

ADO.NET 另一特点是对于 XML 的本原支持（XML Native Support），在 ADO.NET 中通过 XMLReader，XMLWriter，XMLNavigator，XMLDocument 等可以方便地创建和使用 XML 数据，并且支持 W3C 的 XSLT、DTD、XDR 等标准，从而使面向网络的数据交互得以高效、安全、容易地进行。ADO.NET 对 XML 的支持也为 XML 成为 Microsoft.NET 中数据交换的统一格式奠定了基础。XML 组件是.NET Framework 的重要组成部分。

XML（eXtensible Markup Language，可扩展标记语言）最突出的特点是将业务规则从数据中分离出来，从而提供了一种通用的数据格式表示法。在 XML 环境下，参与数据交换的任一方都可以定义自己的标记结构，而不必受限于某个标准框架。XML 组件使.NET Framework 成为一个有机整体，.NET Framework 就是建立在 XML 的数据结构之上，各种数据交换几乎全部使用标准的 XML 进行。

在类库的上层是两种编程模板，一种是基于 ASP.NET 的 Web 网络应用程序模板 Web Forms 和 Web Services，另一种是基于传统 GUI 交互的 Windows 应用程序模板 Win Forms。

Web Forms 是通过浏览器接收用户输入并动态地生成 Web 页面的 Web 用户接口（UI）。WebForms 在支持传统的将 HTML 内容与脚本代码混合的同时，通过代码后置，将应用程序代码和用户接口内容分离，使网页能动态地编译到受控类中，从而提高了系统性能。ASP.NET 还提供了丰富的 Web 控件，包括运行在服务端的 from 控件，如 Button、TextBox 等，也包括一些特殊用途的控件，如广告轮换控件、日历控件、用户验证控件等。这些控件可以在 System.Web.UI.WebControls 名字空间下找到。

Web Services 与 Web Forms 不同，它不从浏览器获得请求并返回相应的 Web 页面，而是从应用程序接收请求，执行任务，然后向应用程序返回响应信息。ASP.NET 为 Web Services 提供了一个基础构造环境，从而使开发人员能够为这个服务模型创建服务。基于 ASP.NET 的 Web Services 就是编写一个后缀为.ASMX 的文件，在文件中加入要表现的方法。ASMX 文件包含受控类的引用，这个类是由 ASP.NET 提供的 Services 类所派生的。公有的类方法在标记上属性后，就会成为网络服务方法，把 HTTP 请求发送到 ASMX 文件中的 URL 后，这些方法就会被调用。当被调用者发出请求时，ASP.NET 会检查类的元数据，自动生成 SCL 文件。

Win Forms 编程模板为 Windows 开发者提供了基于表单控件的编程方式。Win Forms 是一整套可以重用的类库，包括了 VB 组件库（Visual Basic component library）和 Windows 基础类库（Windows Foundation Classes）。使用 Win Forms 模板可以利用 Windows 丰富的用户接口特点，包括现在的 ActiveX 控件和 Windows 2000 的新特点，如透明的、分层的浮动窗口，还可以基于 CLR 进行开发，可以使用 ADO .NET、网络服务等，从而充分利用.NET Framework 的各种新特性。

.NET Framework 提供了多种语言开发执行的环境以及全新的数据库访问技术和网络应用开发技术。利用.NET Framework 的功能，可缩短开发时间，提高应用程序的可靠性和性能，使得快速构建新一代的网络应用程序和 Web 服务成为可能。

1.4 .NET 组件与 COM 组件的互操作

组件对象模型 COM 提供了一种客户/服务器标准，已经广泛应用于操作系统和应用程序中。但由于 CO M 组件的开发、使用与维护比较复杂，并且由于 CO M 技术自身缺陷带来的“DLL HELL”问题，对系统稳定性的影响已经日益严重，因此微软在.Net 平台下提出了程序集(Assembly)组件解决方案。组件的维护和销毁由.Net Framework 来负责，极大简化了组件的开发过程。同时，.N et 组件由于具有“自说明”特性，不再需要注册，具有一种新的组件发布方式--XCOPY， 极大简化了组件的部署，微软认为“.NET 平台应用程序将最终取代那些用 COM 开发的应用程序”。

.Net 平台是不能够对非托管的 COM 直接进行访问，要把 COM 组件全部移植到.Net 平台下的工作量是巨大的，不仅需要测试新的代码，而且必须测试移植的代码，但由于长期积累的 COM 组件对于.Net 平台仍是一个巨大资源，所以一段时间内软件项目的开发通过托管与非托管代码的混合编程来实现.Net 平台下对 COM 访问是不可避免的。

.Net 平台下通过 COM Interop 技术来实现与 COM 组件的互操作。为了支持向后兼容，COM interop 提供了不需要修改现有组件而能访问现有 COM 组件的方法。可以通过使用 COM interop 提供的类型库导入工具（Type Library Importer utility，

tlbimp.exe）导入相关的 COM 类型到.NET Framework 的应用中。一旦导入，COM 的类型就可以使用了。COM interop 同时也提供了向前兼容使得 COM 的客户可以像访问其他的 COM 对象一样访问托管的代码，COM interop 提供了从程序集中导出元数据（metadata）到类型库并且像传统 COM 组件一样注册托管组件的方法。无论是导出还是导入工具处理的结果都与 COM 规范一致。在运行时，如果需要的话 CLR 在 COM 对象和托管代码之间列集（marshals）数据。

.NET 提供了大量的类库来方便地实现与 COM 的相互操作，其中很重要的一个名称空间就是：System.Runtime.InteropServices，这个名称空间提供了一系列的类来对 COM 对象进行操作。下面的例子中，我们来调用系统自带的 Win32 函数 MessageBoxA，这个函数位于系统的 COM 组件 user32.dll 当中，我们调用的代码如下：

```
using System;
using System.Runtime.InteropServices;
class Test
{
    [DllImport("user32.dll")]
    public static extern int MessageBoxA(inthWnd,string strMsg,string
    strCaption,intnType);
    public static void Main()
    {
        int myMsg;
        myMsg = MessageBoxA(0,"Hello!","test",0);
      }
}
```

1.4.1 COM 包装（COM Wrappers）

为了实现传统的 COM 程序与.NET 程序之间的相互调用，.NET 提供了两个包装类：运行时可调用包装（runtime callable wrapper，RCW）和 COM 可调用包装（COM callable wrapper，CCW）。每当一个.NET 客户程序调用一个 COM 对象的方法时就会创建一个 RCW 对象；相反，每当一个 COM 客户程序调用一个.NET 对象的方法时就会创建一个 CCW 对象，如图 1-12 所示。

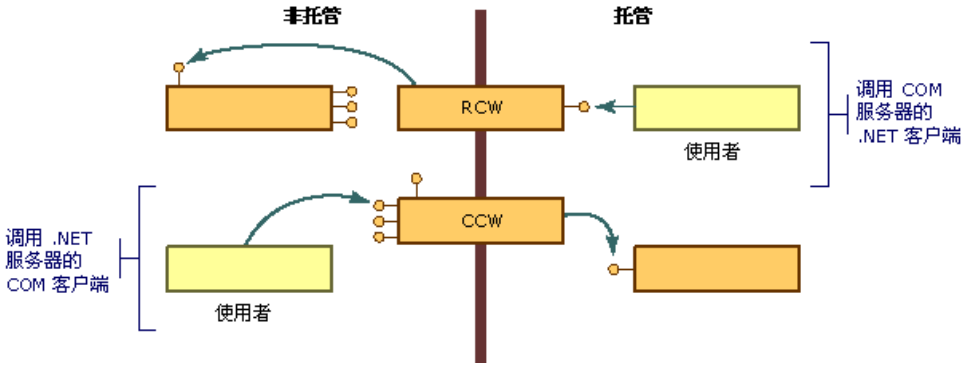


图 1-12　.NET 通过包装类实现与 COM 的互操作

1.4.1.1 运行时可调用包装

公共语言运行时通过名为运行时可调用包装 (RCW) 的代理来公开 COM 对象。虽然 RCW 在 .NET 客户端看来是普通的对象，但它的主要功能是封装在 .NET 客户端和 COM 对象之间传递的调用。

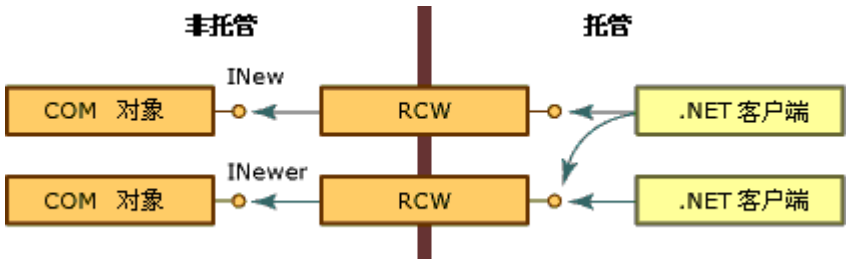


图 1-13　通过运行时可调用包装来访问 COM 对象

RCW 的主要功能：

(1) RCW 实际上是 runtime 生成的一个.NET 类，它包装了 COM 组件的方法，并内部实现对 COM 组件的调用。

(2) 列集(marshal).NET 客户与 COM 对象之间的调用，列集的对象包括方法的参数、返回值等，比如 C#中的 string 与 COM 中的 BSTR 之间的转换。

(3) CLR 为每个 COM 对象创建一个 RCW，与对象上的引用数无关，就是说每个 COM 对象有且只会有一个 RCW 对象。

(4) RCW 中包含了 COM 对象的接口指针，并管理 COM 对象的引用计数。RCW 自身的释放通过 Garbage Collection 机制管理。

无论一个 COM 对象存在多少个引用，运行时只为每个 COM 对象创建一个 RCW。运行时只为每个对象维护一个每进程的 RCW。如果在一个应用程序域或单元中创建 RCW，然后将引用传递至另外一个应用程序域或单元，则会使用第一个对象的代理。如图 1-14 所示，对于公开 Inew 和 Inewer 接口的 COM 对象，任意数目的托管客户端都可以持有引用。

利用从类型库中导出的元数据，运行时将创建所调用的 COM 对象和该对象的包装。每个 RCW 都会在它所包装的 COM 对象上维护一个接口指针缓存，并且在不再需要 RCW 时释放它对该 COM 对象的引用。运行时将对 RCW 执行垃圾回收。

1.4.1.2 COM 可调用包装

当 COM 客户端调用一个 .NET 对象时，公共语言运行时将创建托管对象以及该对象的 COM 可调用包装 (CCW)。由于无法直接引用 .NET 对象，COM 客户端会将 CCW 用作托管对象的代理。

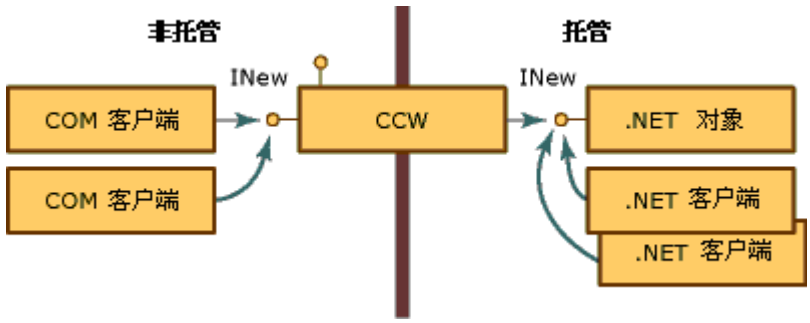


图 1-14　通过 COM 可调用包装来访问 .NET 对象

CCW 的主要功能：

(1) CCW 实际上是 runtime 生成的一个 COM 组件，它在注册表注册，有 CLSID 和 IID，实现了接口，内部包含了对.NET 对象的调用。

(2) 列集(marshal).NET 对象与 COM 客户之间的调用。

(3) 每个.NET 对象只有一个 CCW，多个 COM 客户调用同一个 CCW。

(4) COM 客户以指针的方式调用 CCW，所以 CCW 分配在 non-collected 堆上，不受 runtime 管理。而.NET 对象则分配在 garbage-collected 堆上，受 runtime 管理，享受 CLR 的种种好处。

(5) CCW 实际上是 COM 组件，所以它遵循引用计数规则。当它的引用计数为

0 时，会释放它对它管理的.NET 对象的引用，并释放自己的内存空间。当.NET 对象上引用计数为 0 时，则会被 Garbage Collection 回收。

运行时只为每个托管对象创建一个 CCW，而不管有多少个 COM 客户端在请求其服务。如图 1-15 所示，对于公开 Inew 接口的 CCW，多个 COM 客户端可以持有引用。而该 CCW 又对实现接口的托管对象持有单个引用，并且会进行垃圾回收。COM 和 .NET 客户端可以同时对同一托管对象发出请求。

COM 可调用包装对于在 .NET Framework 中运行的其他类来说是不可见的。它们的主要用途是封装在托管和非托管代码之间传递的调用；但是，CCW 还管理对象标识及其包装的托管对象的对象生存期。

对象标识

运行时从其垃圾回收堆中为 .NET 对象分配内存，这样，运行时就可以在必要时将对象在内存中来回移动。与之相对，运行时从非回收堆中为 CCW 分配内存，从而使 COM 客户端可以直接引用该包装。

对象生存期

与它所包装的 .NET 客户端不同，CCW 以传统的 COM 方式来进行引用计数。当 CCW 上的引用计数达到零时，该包装将释放它对托管对象的引用。在下一次垃圾回收循环中，将回收无剩余引用的托管对象。

1.4.2 ESRI 提供的 Primary Interop Assemblies

Primary interop assemblies（PIAs，主互操作程序集）是 COM 软件厂商提供的为其 COM 组件在.NET 环境下运行的互操作程序集，PIAs 暴露 COM 组件中的所有类, 接口和常量，作为 .NET 托管类。

ESRI 为所有的 ArcObjects 类型库提供了 PIAs，ArcGIS 的.NET 开发人员只能使用这些安装在 GAC (Global Assembly Cache，缺省安装目录为 C:\WINDOWS\assembly，如图 1-15 所示)下的 PIAs. 这些 PIAs 的公钥标记（public key）为 8FC3CC631E44AD86.

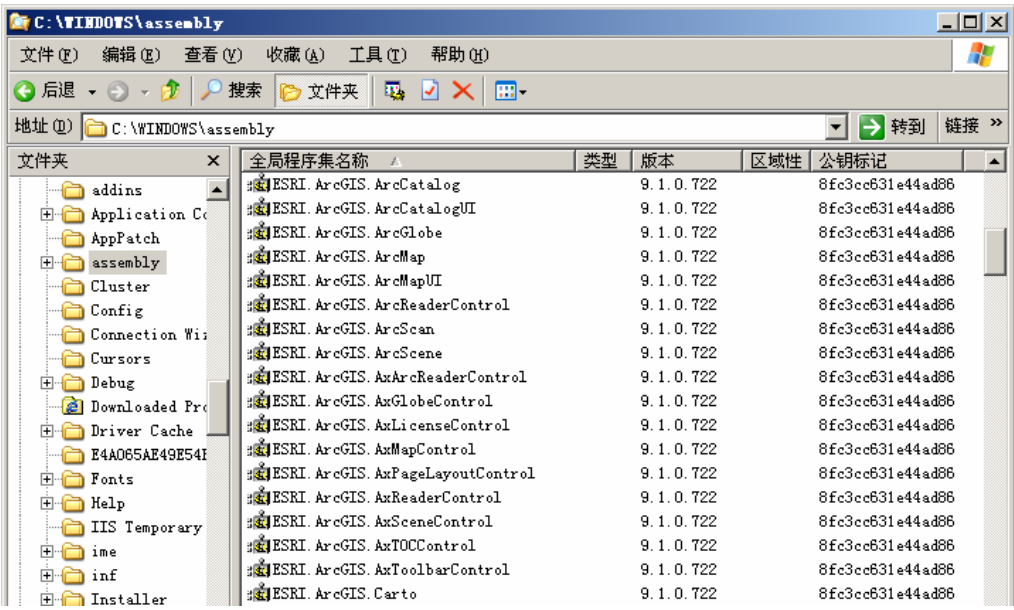


图 1-15　ESRI 提供的 PIAs

1.4.3　向 COM 公开.NET 对象

1.4.3.1　注册.NET 组件

使用定制的.NET 组件来扩展 ArcGIS 应用程序，需要在 COM 注册表中注册.NET 组件，并将.NET 的程序集（assemblies）输出为 COM 的类型库(tlb)，完成这一任务有两条途径：一是使用.NET Framework 提供的 Regasm 工具，二是使用 Visual Studio.NET 开发环境（Register for COM Interop）。

程序集注册 (Regasm.exe) 的命令行工具用来注册或注销与 COM 一起使用的程序集。Regasm.exe 会将有关类的信息添加到系统注册表中，以便 COM 客户端透明地使用 .NET 类。RegistrationServices 类提供了等效的功能。例如：

regasm EditTools.dll /tlb:EditTools.tlb /codebase

将.NET 组件 EditTools.dll 注册为 COM 组件，其 COM 类型库为 EditTools.tlb（由 tlb 参数指定），/codebase 选项表示指向.NET 组件的路径包括在注册表中。

在 Visual Studio.NET 开发环境中，通过下列步骤来来自动完成.NET 组件的注册：

- 在“解决方案资源管理器”中，鼠标右键点击要注册的项目（即正在开发的.NET 组件）， 然后点击“属性”；
- 在项目“属性页”对话框中，选择“配置属性”，然后点击“生成”，在右边的窗格中，将“为 COM Interop 注册”设置为“True”，如图 1-16 所示，点击“确定”按钮，即可完成.NET 组件的自动注册。

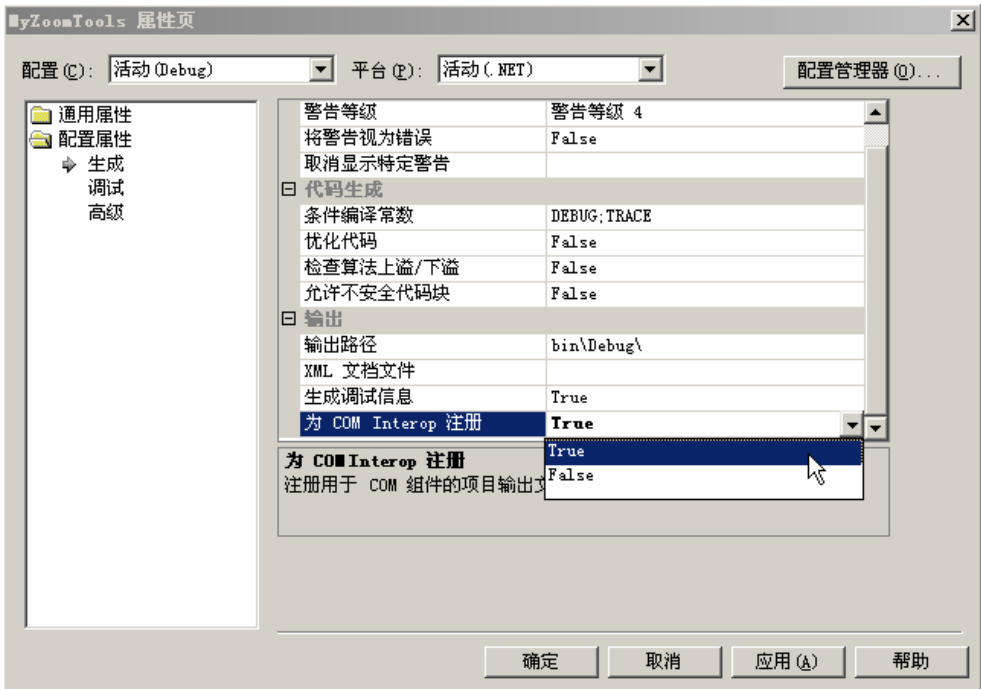


图 1-16 项目属性设置

1.4.3.2 在 COM 组件类目中注册.NET 类

大量的 ArcGIS 扩展都依赖 COM 组件类目。事实上，大部分定制的 ArcGIS 组件必须注册到合适的 COM 组件类目中才能正常使用，例如，所有的 ArcMap 命令和工具必须注册到“ESRI Mx Commands”组件类目中。 可以通过下列途径将.Net 类注册到 COM 组件类目中。

(I) 自定义对话框

在.NET 环境下自定义的 ArcGIS 命令和工具可以通过 ArcGIS 的 Customize 对话框中的 Add From File 按钮快速地添加到工具条中。在这种情况下，只需要简单地浏览一下 TLB 并打开它。ArcGIS 框架会自动地将从类型库中选中的类添加到合适的组件类目中。

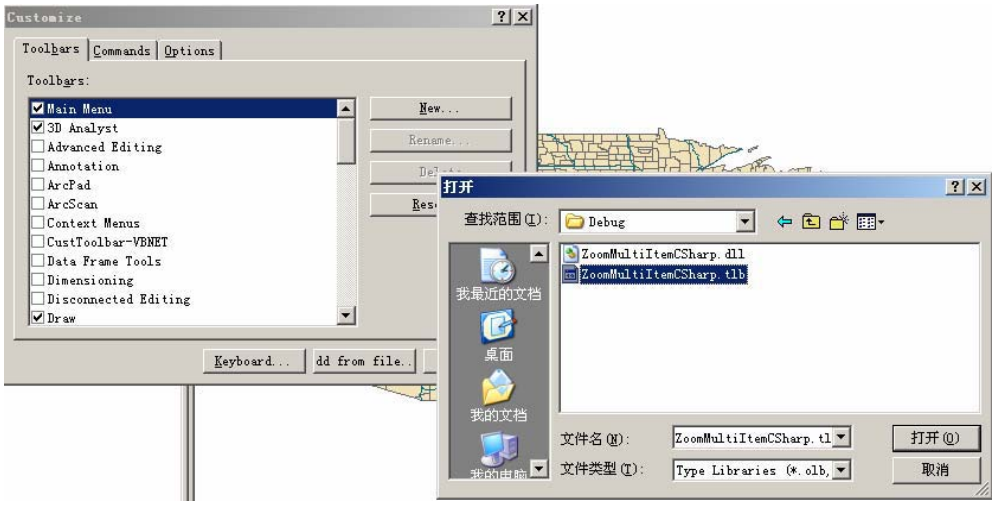


图 1-17 通过 Customize 对话框注册组件类

(II) 组件类目管理工具

另外一个选项是使用 Component Catesories Manager（Categories.exe）。在这种情况下，可以在应用程序中选择组件类目，浏览类型库，并选择合适的类目。

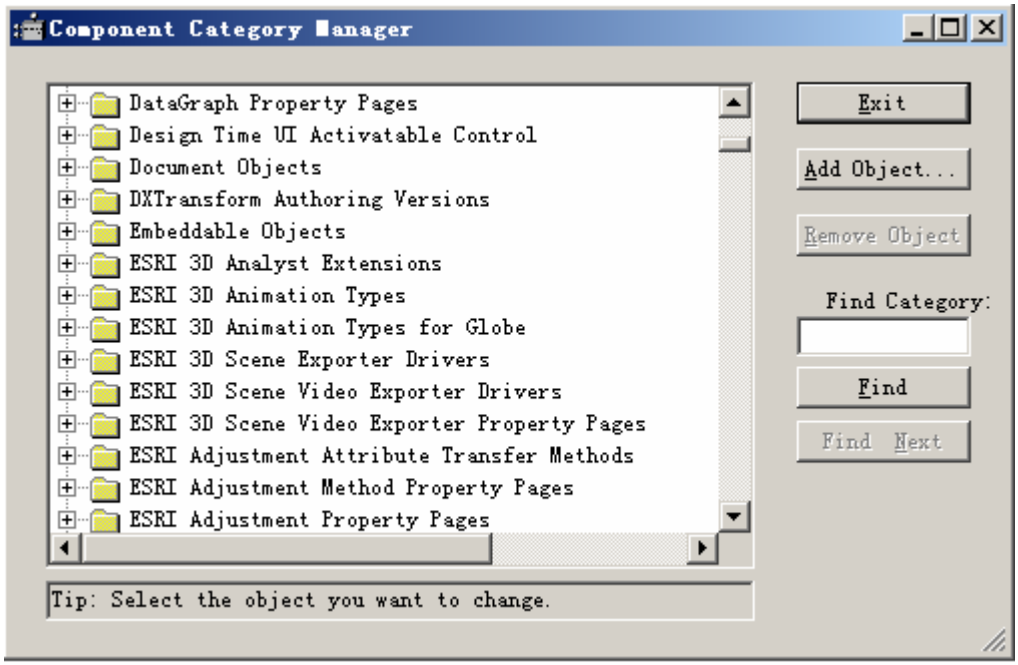


图 1-18 通过 Categories 工具注册组件类

(III) 使用 COM 注册函数（COMRegisterFunction）

最后的也是推荐的解决方案是添加代码到.NET 类中，这样每当注册.NET 组件的时候，会在特定的组件类目中自动注册.NET 类。.NET Framework 类库中包含两个属性类（ComRegisterFunction 和 ComUnregisterFunction），它们允许用户指定方法注册或注销.NET 类。 两个方法都传递当前被注册类的 CLSID，使用这个信息，可以在这两个方法内编写适合的代码注册或注销.NET 类。在组件类目内注册一个组件类，需要程序员知道组件类目的唯一 ID（CATID）。

为了帮助程序员简化这一注册过程，ESRI 提供了 utility 程序集（ESRI.ArcObjects.Utility.dll），其中包含 ESRI.ArcGIS.Utility.CATIDs 名称空间，这个名称空间中包含代表 ArcGIS 组件类目的类，这些类知道组件类目的 CATID，并提供两个静态方法(Register and Unregister)来添加或删除组件类。下面的例子展示了一个定制的 PanTool 工具注册到“ESRI Mx Commands”组件类目中方法。

```
 [Guid("d36096cd-4680-42d6-8e5a-8c88cf86170b")]
public sealed class PanTool : BaseTool
{
   #region "Component Category Registration"
    [ComRegisterFunction()]
    static void Reg(string regKey)
    {
     MxCommands.Register(regKey);
     }
    [ComUnregisterFunction()]
    static void Unreg(string regKey)
    {
     MxCommands.Unregister(regKey);
     }
   #endregion
...
}
```

1.4.4 COM 到.NET 的类型转换

通常来说，类型库导入工具导入与最初在 COM 中名字相同的类型。所有导入的类型都添加到一个遵循下面命名规则的命名空间中：ESRI.AICGIS 加上库名。例如 Geometry 库的命名空间是 ESRI.ArcGIS.Geometry。所有类型通过它们的完整的命名空间和类型名识别。

所有的 COM 组件类被转换成为托管类；托管类的类名为起初的类名加上“Class”，例如，Point 组件类的运行时可调用包装（RCW）为 PointClass， Map 组件类的运行时可调用包装为 MapClass. 所有类也有一个和组件类相同名称的接口，该接口对应于组件类缺省接口。例如，PointClass 有一个 IPoint 接口。

C＃中不支持有引用传递或多个参数的属性。在这种情况下，必须使用存取方法代替。下面摘录的代码展示了一个实例：

```
ILayer pLayer = axMapControl1.get_Layer(0);
MessageBox.Show（pLayer.Name）;
```

在上例中带参数的 Layer 属性，在 C#中是不支持的，所以改用 get_Layer()方法。

1.5 ArcGIS 开发资源

ESRI has created two essential resources for ArcGIS developers: the ArcGIS Software Developer Kit and the ESRI Developer Network (EDN) Web site (EDN.esri.com).

1.5.1 ArcGIS Software Developer Kit (SDK)

The ArcGIS SDK is the collection of diagrams, utilities, add-ins, samples, and documentation geared to help developers implement custom ArcGIS functionality.

The graphic below depicts a typical SDK installation.

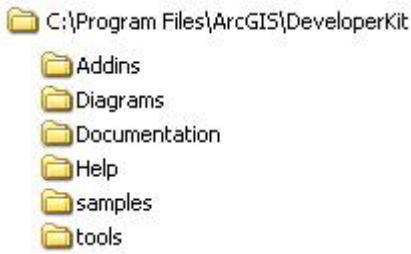


图 1-19 ArcGIS SDK 安装目录

1.5.1.1 ArcGIS Developer Help System

The ArcGIS Developer Help System is the gateway to all the SDK documentation including help for the add-ins, developer tools, and samples; in addition, it serves

as the complete syntactical reference for all object libraries.

Each supported API has a version of the help system that works in concert with it. Regardless of the API you choose to use, you will see the appropriate library reference syntax and have a help system that is integrated with your development environment. For example, if you are a Visual Basic 6 developer you will use ArcGISDevHelp.chm which has the VB6 syntax and integrates with the VB6 IDE thereby providing F1 help support in the Code Window.

The help systems reside in the DeveloperKit\Help folder but are typically launched from the start menu or F1 help in Visual Basic 6 and Visual Studio.NET 2003. The graphic below shows the start menu options for opening the help systems.

1.5.1.2 Samples

The ArcGIS Developer Kit contains over 600 samples many of which are written in several languages. The samples are described in the help system and the source code and project files are installed in the DeveloperKit\samples folder. The help system's table of contents for the samples section mirrors the samples directory structure.

The help system organizes samples by functionality. For example, all the Geodatabase samples are grouped under Samples\Geodatabase. Most first tier groupings are further subdivided. You can also find samples in the SDK using the 'Query the Samples' topic in the help system which lists all the samples alphabetically; in addition, you can sort the list by language. For example, you can elect to only list the available C# samples.

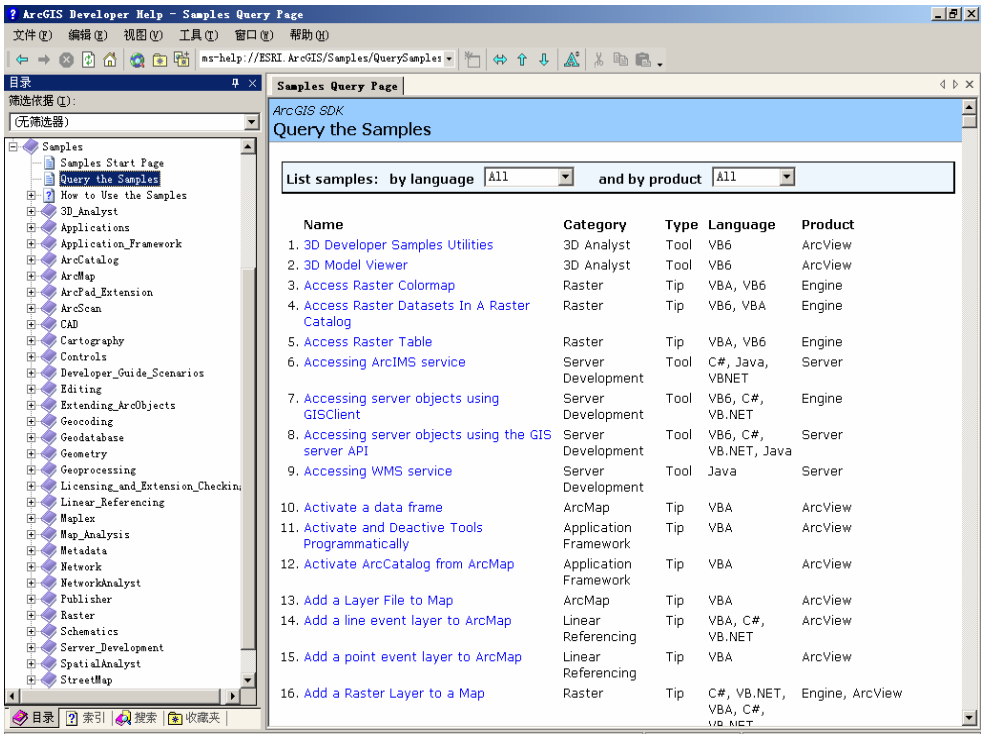


图 1-20 ArcGIS 帮助系统提供的 samples

Installing the samples source code and project files is an option in the install. The samples are installed under the ArcGIS\DeveloperKit\samples folder. If you don't have this folder on your computer, you can rerun the install program and check on Samples under Developer Kit.

1.5.1.3 Developer Tools

The ArcGIS Developer Tools are executables that ESRI has provided to facilitate your ArcObjects development. You may find some of these tools essential. For example, if you are a Visual Basic 6 desktop developer you will likely use the Categories.exe tool to register components in component categories.

Tools available with each ArcGIS developer kit

- Component Categories Manager—Registers components within a specific component category.
- Fix Registry Utility—Fixes corruptions in the Component Categories section of the registry.

- GUID Tool—Generates GUIDs, in registry format, for use within source code.
- Library Locator—Identifies object library containing a specified interface, coclass, enumeration, or structure.

Additional tools available in the Desktop developer kit

- ESRI Object Browser--Advanced object browser.
- Extract VBA--Extracts VBA code from a corrupt map document (.mxd).

The developer tools are installed under the DeveloperKit\tools folder. There is one exception, the Component Category Manager is located in the ArcGIS\bin folder.

1.5.1.4 Add-ins

The ESRI add-ins automate some of the tasks performed by the software engineer when developing with ArcObjects, as well as provide tools that make debugging code easier. ESRI provides add-ins for the Visual Basic 6 IDE and the Visual Studio .NET IDE.

Visual Basic 6
ESRI provides the following add-ins for Visual Basic 6. Note that certain add-ins are only available to Desktop developers.

- ESRI Align Controls with Tab Index
- ESRI Automatic References
- ESRI Code Converter
- ESRI Command Creation Wizard
- ESRI Compile and Register
- ESRI ErrorHandler Generator
- ESRI ErrorHandler Remover
- ESRI Interface Implementer
- ESRI License Initializer
- ESRI Line Number Generator
- ESRI Line Number Remover

Visual Studio.NET
ESRI provides the following add-ins for Visual Studio.NET.

- ESRI .NET AutomaticReferences add-in
- ESRI .NET CodeConverter add-in
- ESRI .NET ComponentCategoryRegistrar add-in

- ESRI .NET GuidGen add-in
- ESRI .NET LicenseInitializer add-in

(I) How to set up the ESRI add-ins for Visual Studio .NET

(1) Installing the ESRI add-ins for Visual Studio .NET

The add-ins are installed with the ArcGIS Developer Kit; select the .NET feature from the install options. If you have not already installed Visual Studio .NET or the .NET Framework Software Development Kit, the .NET feature will be absent from the ArcGIS Developer Kit install options.

(2) Registering the ESRI add-ins for Visual Studio .NET

The add-ins are installed to the <ArcGIS install>/DeveloperKit/Addins folder.To register the .NET addins, run the 'RegisterDotNetAddin.bat' file also in this folder.To unregister the add-ins run 'UnregisterDotNetAddin.bat'.

Running these files uses the utility esriRegDotNetAddin.exe to register the addins with Visual Studio .NET. This command-line utility has the following syntax:

esriRegDotNetAddIn.exe [addin file name] [options]

The options are:

- /all - register all ESRI addins to be used with the latest version of Visual Studio.Net installed. NOTE: If you use this option, you should not specify an addin file name.
- /version:Version - Visual Studio .Net version number. Version 7.1 corresponds to Visual Studio .NET 2003.
- /unregister - unregister ESRI Add-Ins.
- /silent - Silent mode: do not display message box.
- /? or /help - Display the usage message.

Examples

esriRegDotNetAddIn.exe /all /version:7.1 /silent

esriRegDotNetAddIn.exe ComponentRegistrar.dll /version:7.1

(3) To enable an add-in in Visual Studio .NET

1. Click the Tools menu and click Add-in Manager.
2. In the Available Add-ins list, check the add-in you want to enable.
3. If the add-in has a command, a corresponding menu item will appear near the top of the Tools menu.
4. If you want this add-in to be enabled everytime you start Visual Studio .NET, check the Startup checkbox for this add-in.
5. Click OK to dismiss the Add-in Manager dialog.

1.5.1.5 THE ArcGIS DEVELOPER DOCUMENTATION SERIES

The ***ArcGIS Desktop Developer Guide*** is for developers who want to customize or extend one of the ArcGIS Desktop applications, such as ArcMap or ArcCatalog. Developers can use Visual Basic for Applications (VBA) to customize and either Visual Basic, Visual C++, or .NET to extend the applications.

The ***ArcGIS Engine Developer Guide*** provides information for developers who want to create applications based on ArcGIS Engine. ArcGIS Engine allows you to embed GIS functionality within other applications and create desktop-like applications using the supplied ArcGIS controls, such as MapControl, Toolbar, and PageLayout controls. ArcGIS Engine is also based on ArcObjects components and may be programmed through a number of APIs.

The ***ArcGIS Server Administrator and Developer Guide*** is for developers who want to use ArcGIS Server to build custom server applications. Server developers can build Web services and Web applications that do simple mapping or include advanced GIS functionality. Several scenarios illustrate with code examples some of the different types of applications that can be developed using one of the multiple ArcGIS Server Developer Kits. This book also serves as the administration guide to ArcGIS Server.

1.5.1.6 Library Reference

Library Reference provides detail information about all components, Classes, Interfaces etc. contained in ArcGIS.

1.5.2 ESRI Developer Network Web site

The EDN Web site is the place to find the most up-to-date ArcGIS 9 developer information including sample code, technical documents, object model diagrams, and the complete object library reference.

The web site is a reflection of the ArcGIS Developer Help system except it is online and therefore more current. The web site has some additional features including an advanced search utility that enables you to control the scope of your searches. For example, you can create a search that only scans the library reference portion of the help system.

Visit the site today (http://edn.esri.com).

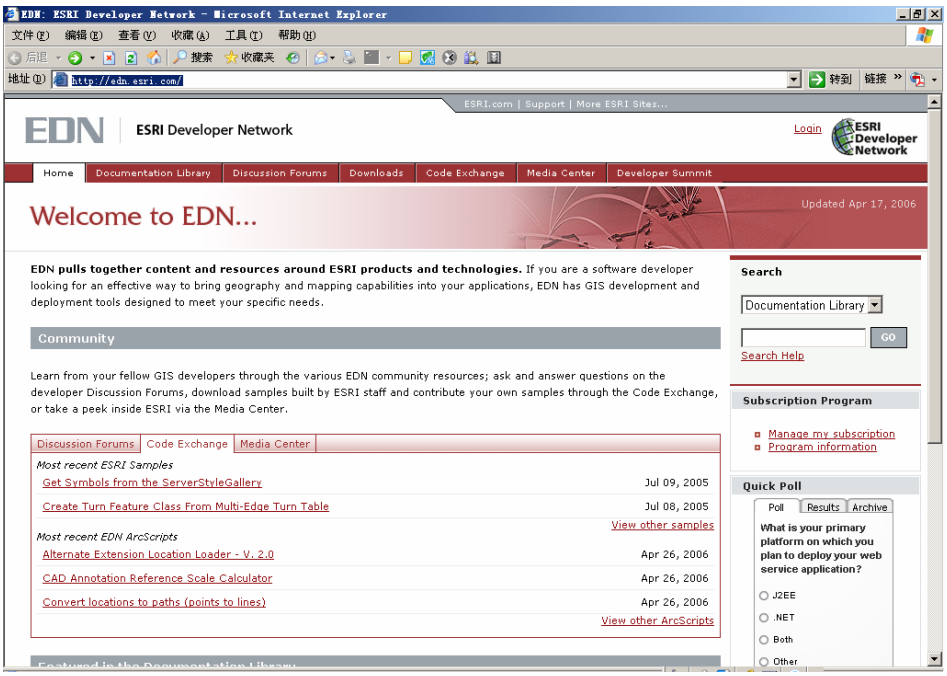


图 1-21 ESRI 开发人员网络站点

1.6 ArcGIS 的二次开发方式

ArcObjects 为开发者提供了丰富的 GIS 功能，开发者可以使用标准的开发框架来扩展 ArcGIS 桌面，使用 ArcGIS Engine 构建独立的 GIS 应用程序，使用 ArcGIS

Server 开发企业级 GIS 服务应用程序。

ArcGIS 提供了强大的二次开发工具，程序员可以依据实际业务和项目的需要

- 使用 VBA 进行 ArcGIS 桌面的定制
- 使用 DLL 组件开发扩展 ArcGIS 桌面
- 使用 ArcObjects 提供的可视化控件来编写一个脱离了 ArcGIS 框架的独立软件。

这三种开发方式并没有哪一种是最合适的，选择哪一种方法，完全取决于实际的项目需求。

ArcObjects 提供四种开发 APIs--COM、.NET、Java 和 C++，开发人员可以使用任何支持这些 API 的开发环境进行应用程序开发。对于非可视化应用程序，常用的语言选择包括 C++和 Java。对于可视化应用程序，则有许多可用的 Windows 语言，如 Visual Basic 6.0、C#.NET、Java 和 Visual C++等。下面列出了适合 ArcObjects 的一些可能的 APIs、开发环境和语言。

- COM：Visual Studio 6.0(VB，VC++)
- .NET：Visual Studio.NET(VB.NET，C#)
- Java：JBuilder、Eclipse、Websphere Studio 等
- C++：Visual Studio 6.0，VC.NET，C++ Builder

开发环境的选择最终取决于开发人员熟悉的编程语言、希望提供给终端用户的功能以及是否要与其他现有应用程序或技术集成。

1.6.1 使用 VBA 进行 ArcGIS 桌面的定制

在 ArcMap 和 ArcCatalog 等 ArcGIS 程序中可以使用 VBA (VisualBasic for Application)进行嵌入式二次开发，VBA 并不是一个独立的程序，它是嵌在 ArcGIS 程序中的一种工业化开发工具。VBA 是 VB 的子集，因此 VB 程序员在 VBA 环境中将没有任何的不适应。在 VBA 环境中，用户甚至可以仅仅通过拖曳已经存在的命令和工具来定制程序界面，而不需要写一行代码。

使用 VBA 开发环境进行 ArcGIS 的定制和扩展是最简单方便的，因为 VBA 编程实际上是处于一个程序框架中，在这个环境里面，Application 和 ThisDocument 两个全局变量可以让程序员直接获取程序本身和程序所使用的文档对象，而且标准的 ESRI 组件库也自动引入编程环境中，同时程序员还可以使用系统内的调试器来

查找任何编码错误。

VBA 代码可以让程序员添加模块、类模块和用户窗体，它适合中小型的二次开发。使用 VBA 还有一大好处，就是 ESRI 提供的开发帮助大部分是采用 VBA 编写的，这对于初学者是一笔巨大的财富。

本节将介绍如何给 ArcMap 添加一个工具栏、新建一个宏(Macros)并且执行它，以及产生一个 UIControl。

1.6.1.1 添加一个工具栏

启动 ArcMap 程序，点击“Tools”菜单的“Customize”项后程序会弹出一个“Customize”对话框，如图 1-23 所示，选择“Toolbars”一项，然后点击“new”按钮，会弹出一个“new Toolbar”对话框（图 1-24），将要新建的工具条命名，然后选择保存的位置，如果是保存在 normal.mxt，这个新建的工具栏将会保存在模板中，所有由这个模板产生的文档都将可以使用这个工具条，如果是保存在一个当前使用的 mxd 文件上，那只能在这个文档文件上才可以使用新建的工具栏。

在“ToolbarName”中填入“MyToolbar”，选择保存在模板上，点击“OK”，如图 1-24 所示。

这时将会出现一个新建的工具栏，它的名称是“MyToolbar”，将它拖放到 ArcMap 上，使之不再处于浮动(floating)状态。如果希望它消失，只需要在“Toolbars''中寻找到“MyToolbar”，然后将它前面的选择框取消就可以了。

现在新建一个菜单栏的任务已经完成了。读者也可以在这里编辑、删除和隐藏一个工具栏。

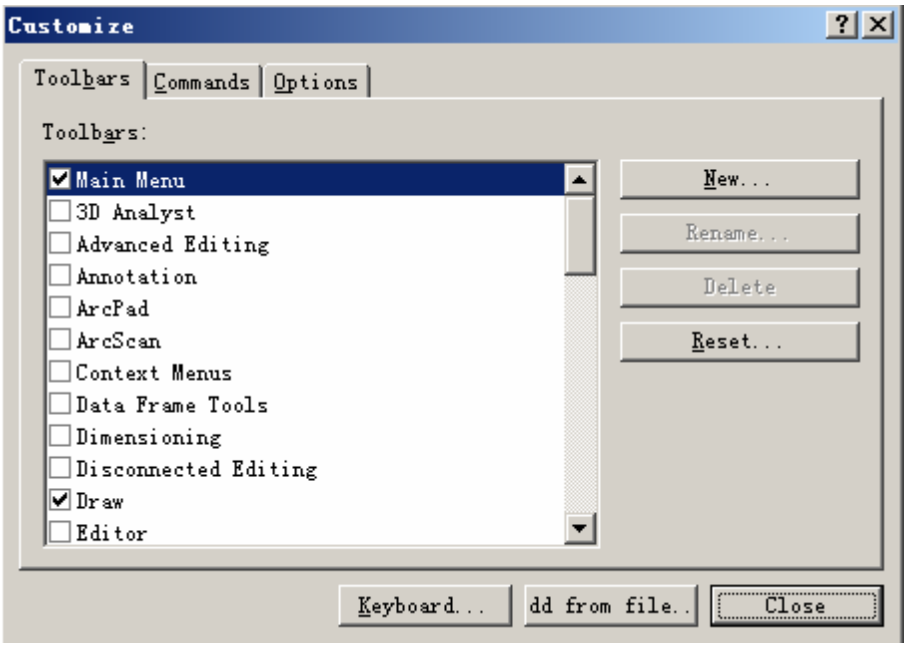


图 1-23　Customize 对话框

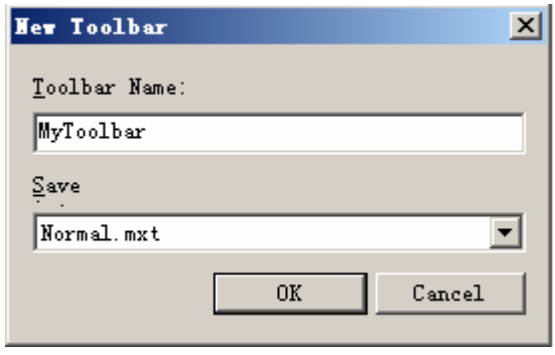


图 1-24　新建工具栏对话框

1.6.1.2 编写一个宏命令

程序员将那些能自动执行某种操作的命令统称为“宏”。宏是一种操作命令，它和菜单操作命令一致，可以认为宏和普通的命令是一样的。使用宏可以完成很多重复的动作。

点击 ArcMap 的菜单栏 Tools，指向 Macros 然后点击 Macros，这个时候会弹出一个 Macros 对话框，如图 1-25 所示。

在 Macro Name 中键入：MyZoomIn，然后点击“Create"按钮，需要注意的是“Macros in”右边的下拉菜单，如果选择的是 Project，那么这个宏命令将保存在程序正在使用的 MXD 文件中，MyZoomln 仅仅能在本档中使用；如果选择的是 normal，那将会保存在 normal 模板中，这个宏命令将出现在任何一个 MXD 文件中，因为每个 MXD 文件都是从 normal.mxt 中派生的。

点击按钮后将弹出个 VBE(Visual Basic Editor)，它已经自动写下了如下代码:

```
Sub MyZoomIn()

End Sub
```

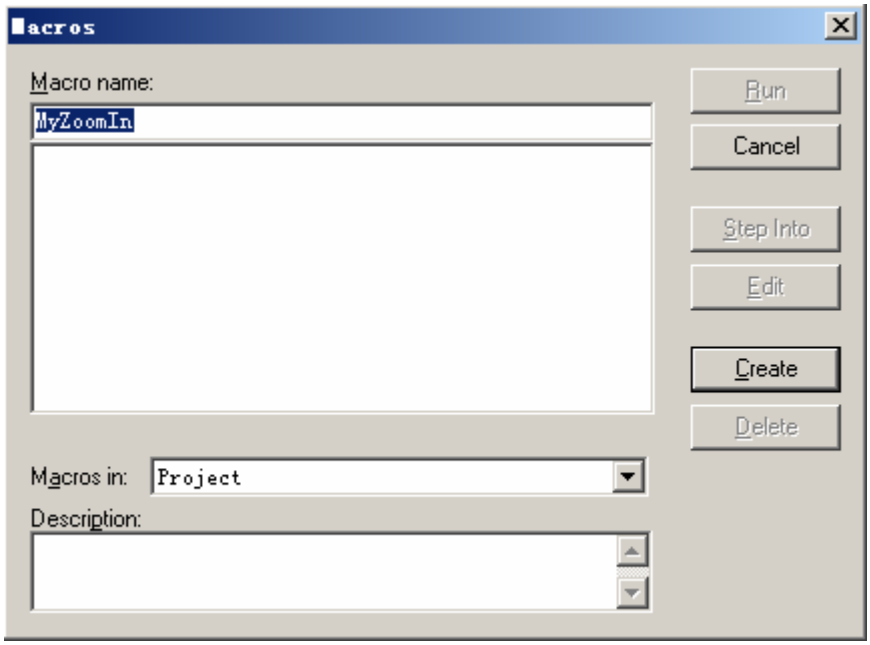


图 1-25 新建宏对话框

现在需要做的就是在这两个语句添上必要的执行代码就可以了，键入下面的内容：

```
Dim pDoc As IMxDocument
Dim pEnv As IEnvelope
'获得文档对象
Set pDoc = ThisDocument
'获取文档视图目前的显示范围
Set pEnv = pDoc.ActiveView.Extent
'改变范围
pEnv.Expand 0.2, 0.2, True
```

```
'传入新的显示范围
pDoc.ActiveView.Extent = pEnv
'刷新视图
pDoc.ActiveView.Refresh
```

点击保存按钮，关闭 VBA 环境后返回 ArcMap 窗口，点击“工具”菜单的“定制”(Customize)项后会弹出一个“自定义”对话框，点击中间的“命令”(Commands)按钮，在 Categorie 一栏中寻找到“Macros”一项，这个时候在 Commdans 中将出现在本文档中所有可以使用的宏命令，如图 1-26 所示。

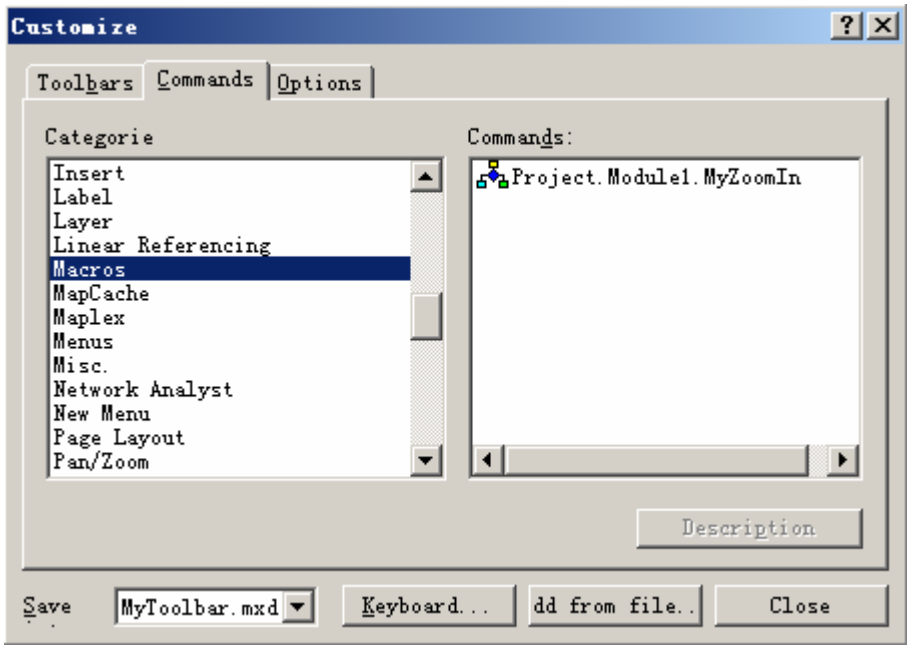


图 1-26 Customize 对话框中的命令

用户可以将新建的宏命令拖放到新建的工具栏上，拖放过程十分简单，选择 Project.Module1.MyZoomIn，不松开鼠标将它拖曳到“MyToolbar”上就可以了。

向 ArcMap 程序中添加地理数据，然后点击这个宏命令的图标，读者可以看到地图的范围在变大。这个简单的代码就是一个放大地图的命令。

1.6.1.3 交互式工具

使用宏命令可以将需要重复的过程包装起来，让用户可以更方便使用，但是如果涉及使用鼠标交互，如使用鼠标拖曳的方式来确定地图的显示范围等，宏命令则无法做到，ArcMap 给二次开发者提供了使用 UIControl 进行鼠标交互的方法。这种

方法称为工具(Tool)。

点击“工具”菜单的“定制”(Customize)项后会弹出一个“定制”对话框，点击中间的“命令”(Commands)按钮，在 Categories 一栏中寻找到“UIControls”一项，点击窗体上的“new UIControl”，会弹出一个对话框，如图 1-27 所示。

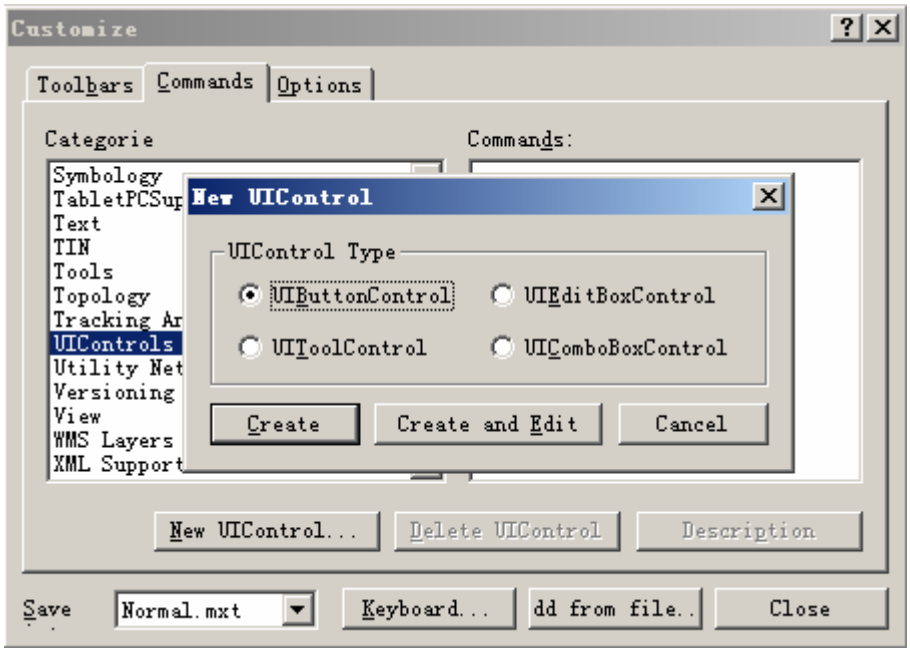


图 1-27 新建 UIControl 对话框

这个对话框可以新建自定义的控件，它罗列了四种不同的类型，用户可以选择“UIToolControl”，点击中间的按钮“Create and Edit”，此时会弹出一个 VBE，代码区中已经自动添加如下代码：

```
Private Sub UIToolConrol1_Select()


End Sub
```

与宏大不一样，UIToolControl1 像一个对象，是拥有可以触发的事件的。

现在要添加的代码的功能是根据鼠标拖曳的范围来确定地图的显示范围。首先删除已经存在的代码，然后从右边工具栏中选择“MouseDown”事件，系统会自动添加下面的代码片段：

```
Private Sub UIToolControll_MouseDown(ByVal button As Long, ByVal shift As Long, ByVal x As Long, ByVal y As Long)


End Sub
```

在这个过程中加入如下代码：

```
    Dim pDoc As IMxDocument
    Set pDoc = ThisDocument

    Dim pScreenDisp As IScreenDisplay
    Set pScreenDisp = pDoc.ActiveView.ScreenDisplay

    Dim pRubber As IRubberBand
    Dim pEnv As IEnvelope
    Set pRubber = new RubberEnvelope
    Set pEnv = pRubber.Tracknew(pScreenDisp, Nothing)
    pDoc.ActiveView.Extent = pEnv
    pDoc.ActiveView.Refresh
```

选择 CursorID 事件，点击它后在代码区内填入如下内容：

```
Private Function UIToolControl1_CursorID() As Variant
    '用于确定鼠标形状
      UIToolControll_CursorID = 3
End Function
```

读者可以自己试验一下完成不同的过程，如 Enable，Message 等后的结果，看看在执行的时候会出现什么结果。

完成代码添加后，点击保存，关闭 VBA 环境返回 ArcMap，类似拖曳一个宏命令一样，将新建的 UIToolControll 控件放置到新建的工具条上。

往 ArcMap 中载入数据，点击 UIToolControll 的图标，然后将鼠标移动到地图区，在地图上拖曳出一个矩形区域，将看到地图的显示范围改变为这个矩形区域。

通过上面的两个简单例子，读者学习了如何使用 VBA 环境来进行二次开发，由于提供了两个公共变量和自动引入类库，因此在写 VBA 代码的时候显得非常容易、快捷。

1.6.1.4 VBA 代码的安全性

也许读者会产生疑惑，上面编写的 VBA 代码保存在什么地方?如何让自己写的过程和函数给第三方使用?毕竟二次开发的最终成果别人可以应用才有意义。

VBA 代码保存在文档或者模板中，也就是说它保存在一个 mxd 文件或者 mxt 文件里面，由于这两种文件可以传播发布，因此读者编写的代码可以很容易地给别人使用。在“Customize”对话框中选择“Options”面板，如图 1-28 所示。

首先看“Save customizations to Normal template bydef”，这个选择是默认将程序员所做的任何定制都保存在 normal.mxt 文件上。

如果文档就这么容易地被人使用，肯定是不妥的，因为代码的安全性和知识产权不能得到保护。这种方式意味着任何人都可以使用和修改这份代码，那么有没有办法保护这些劳动成果呢?点击“Lock Customization”按钮，用户会被要求输入文档保护密码。当输入密码后，尝试打开 VBE，将会出现一个认证对话框要求输入正确的密码才能进入。

读者肯定听说过“宏病毒”这个名称，这是使用宏命令开发出来的病毒程序。为了避免自己的代码成为宏病毒的载体，点击“Change VBA Security”按钮，可以设置 VBA 代码的安全级别，最高级别的安全性将只能运行有签名的宏，而潜在的不安全的命令将会被删除。

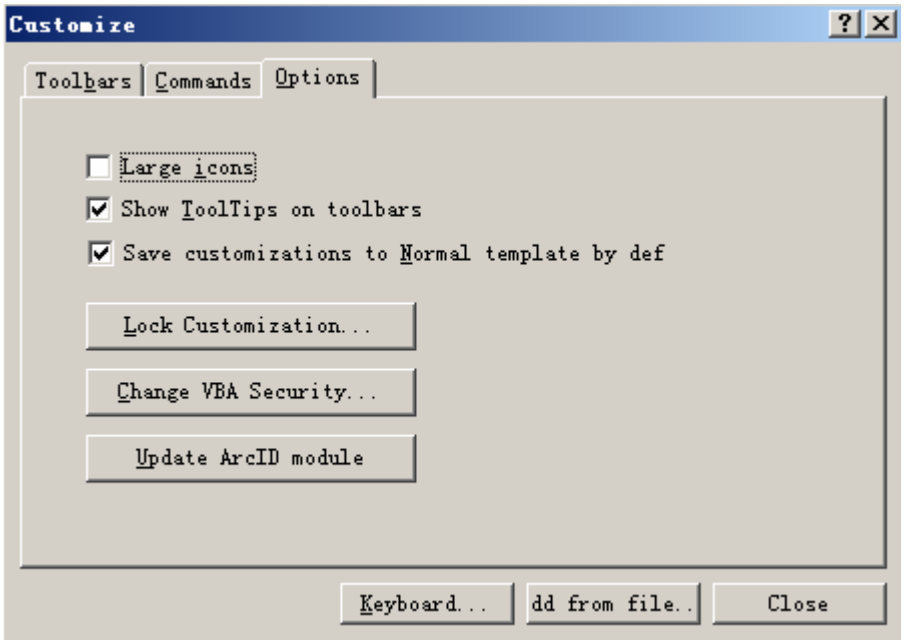


图 1-28 设置宏的安全级别

1.6.2 使用 DLL 组件扩展 ArcGIS 桌面

If you want to use a programming language other than VBA, or if you want to package ArcObjects functionality into a COM DLL, EXE, OCX, or .NET assembly, you will have to work outside the VBA development environment. This approach generally requires creating a project, referencing the ArcObjects type libraries

required, implementing ArcObjects interfaces, adding code, then compiling the source into a binary file.

Following are some advantages of building custom components:

- They can be easily delivered to end users via custom setup programs.
- You can hide your code and logic in a binary file.
- You can extend and customize virtually every aspect of the ArcGIS technology.

Components can be broadly categorized into two areas of customization: those that reside at the application level, such as custom buttons, toolbars, windows, and extensions, and those that reside at the geodatabase level, such as custom feature class extensions and features. Some of these more advanced customizations cannot be accomplished through the VBA environment.

编写 VBA 程序来让大家共享自己开发的代码固然是一种最简单的开发方式，但是这种方法的弊端是很多的。首先，使用 VBA 编码程序员无法保护自己的代码不被复制和修改(密码的保险系数不高)；其次，对方必须安装 ArcGIS 以后才可以使用。这种方法虽然简便，却在安全性和共享性上最差。

为了达到更好的安全性，ArcObjects 程序员还可以使用 DLL 组件的方法进行 ArcGIS 的二次开发的选择。

DLL 就是所谓的"动态链接库"(Dynamic Link Library)，它其实是 WINDOWS 平台下的一种类的文件编译格式，这种格式的文件可以被任何支持 COM 规则的语言所支持和引用。ArcObjects 组件本身的物理文件就是 DLL 文件，读者也可以写自己的 DLL 组件来进行二次开发。

本节仍然使用"放大视图一倍"的这个例子来写一个 DLL 组件。

In this section you will create a class in .NET that implements the *esriSystemUI ICommand* interface. You will create the command from first principles, without using any of the ESRI utility classes or add-ins for Visual Studio.

➢ 新建类库项目

- 打开 Visual Studio.net 2003 开发环境，点击"新建项目"，在"Visual C#项目"中选择"类库"，名称中填入"MyZoomTools"，选择保存位置，点击"确定"，这样就新建了一个项目，如图 1-29 所示。



图 1-29 新建项目对话框

- **引用 ESRI 对象库**

在使用 ESRI 提供的.NET 对象库之前，必须将必需的 ESRI 对象库引用到当前项目中。ESRI 对象库在安装 Devkit 开发包时被安装，默认的安装路径为：ArcGIS 安装根目录\DotNet。

- 在"解决方案管理器"的引用中单击右键，选择"添加引用"，在"添加引用"对话框的".NET"栏中选择 ESRI.ArcGIS.Geometry、ESRI.ArcGIS.ArcMapUI、ESRI.ArcGIS.Carto、ESRI.ArcGIS.Framework、ESRI.ArcGIS.SystemUI 五个对象库，逐个双击它们，在"选定的组件"中将可以看到被选择的组件，如图 1-30 所示，点击"确定"。

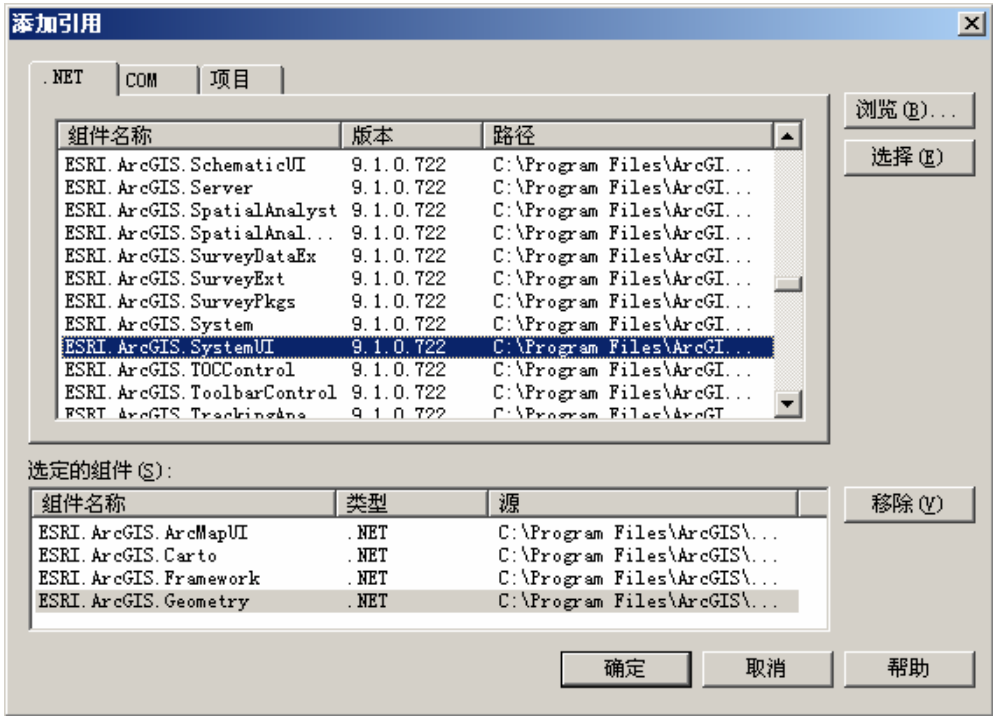


图 1-30 添加 ESRI 对象库引用

➢ 新建 ZoomIn 命令

➢ 在“解决方案管理器”中选择 MyZoomTools，点击右键出现下面的菜单，选择”添加”→”添加新项”，如图 1-31 所示，会弹出”添加新项”对话框。

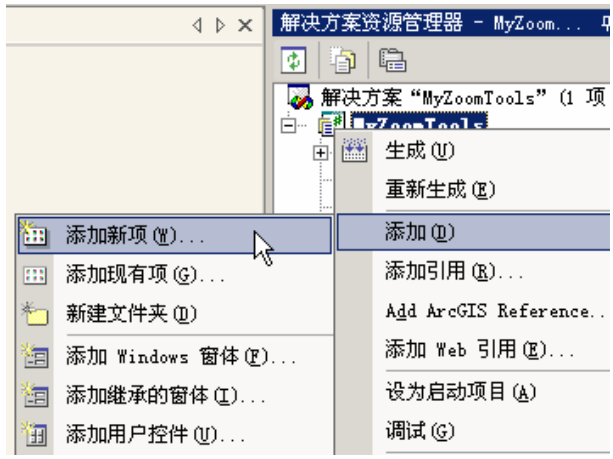


图 1-31 添加新项

- 在“添加新项”的对话框中选择“类”，键入类名“ZoomInCsharp”，如图 1-32 所示，点击“确定”后会出现代码区，在“解决方案管理器”中删除已经存在的文件 Class1.cs。



图 1-32 添加新项对话框

➢ 添加命名空间引用

- 回到 ZoomInCsharp.cs 文件的代码编辑区，在最顶端加上下面的语句，它们是引入对象库：

```
using ESRI.ArcGIS.ArcMapUI;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Framework;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.SystemUI;
```

➢ 实现 ICommand 接口

- 在 ZoomInCsharp.cs 代码窗口中，指定类 ZoomInCsharp 实现 ICommand 接口，即：

```
public class ZoomInCsharp : ICommand
```

具体如图 1-33 所示：

```
using System;
using ESRI.ArcGIS.ArcMapUI;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Framework;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.SystemUI;

namespace MyZoomTools
{
    /// <summary>
    /// ZoomInCsharp 的摘要说明。
    /// </summary>
    public class ZoomInCsharp : ICommand
    {
        public ZoomInCsharp()
        {
            //
            // TODO: 在此处添加构造函数逻辑
            //
        }
```

图 1-33 ZoomInCsharp.cs 代码窗口

- 按住“CTRL+SHIFT+C”或从“视图”菜单选择“类视图”子菜单 ，打开“类视图”窗口（Class View window）。
- 在“类视图”窗口，找到 ZoomInCsharp 类的“基项和接口”，选择 ICommand 接口并点击鼠标右键，从右键菜单中选择“添加”--“实现接口”，如图 1-34 所示。这样就在 ZoomInCsharp.cs 代码窗口中自动添加 ICommand 接口的实现框架。

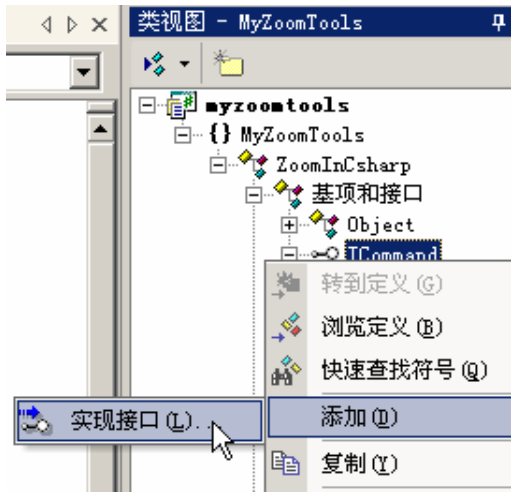


图 1-34 实现 ZoomInCsharp 类的基项

- **为 ICommand 的成员添加代码**
  - 先在类中定义全局变量

```
public class ZoomInCsharp : ICommand
{
    private IApplication m_app;
    private bool m_enabled;
    private System.Drawing.Bitmap m_bitmap;
    private IntPtr m_hBitmap;
  ...
  }
```

  - 按照过程逐个添加如下的代码：

```
    public void OnCreate(object hook)
    {
        if (hook != null)
        {
            if (hook is IMxApplication)
            {
                m_app = (IApplication) hook;
                m_enabled = true;
            }
        }
    }

    public void OnClick()
    {
        IMxDocument mxDoc = (IMxDocument) m_app.Document;
        IActiveView activeView = mxDoc.ActiveView;

        IEnvelope currExtent = (IEnvelope) activeView.Extent;
        currExtent.Expand(0.5D, 0.5D, true);
        activeView.Extent = currExtent;
        activeView.Refresh();
    }

    public string Caption
    {
```

```
        get
        {
            return "Zoom In x 0.5 C#";
        }
    }

    public string Category
    {
        get
        {
            return "Developer Samples";
        }
    }

    public bool Checked
    {
        get
        {
            return false;
        }
    }

    public bool Enabled
    {
        get
        {
            return m_enabled;
        }
    }

    public string Message
    {
        get
        {
            return "Zooms the display to half the current extent.";
        }
    }
```

```
public string Name
{
    get
    {
        return "Developer Samples_Zoom In C#";
    }
}

public string Tooltip
{
    get
    {
        return "Zoom In x 0.5 C#";
    }
}

public int HelpContextID
{
    get
    {
        // TODO:  Add ZoomIn.HelpContextID getter implementation
        return 0;
    }
}

public string HelpFile
{
    get
    {
        // TODO:  Add ZoomIn.HelpFile getter implementation
        return null;
    }
}
```

- **设置命令图标**

将现存的位图文件加入项目，并这一位图文件作为命令图标。

➢ 在“解决方案资源管理器”中， 鼠标右键点击项目 MyZoomTools， 从右键菜单中点击“添加”--“添加现有项”，如图 1-35 所示。

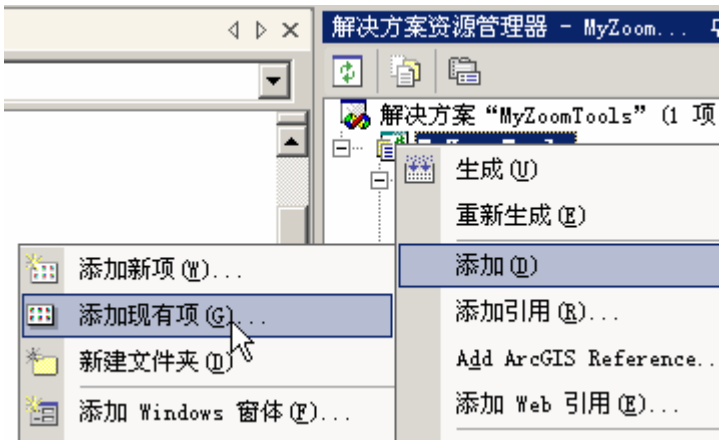
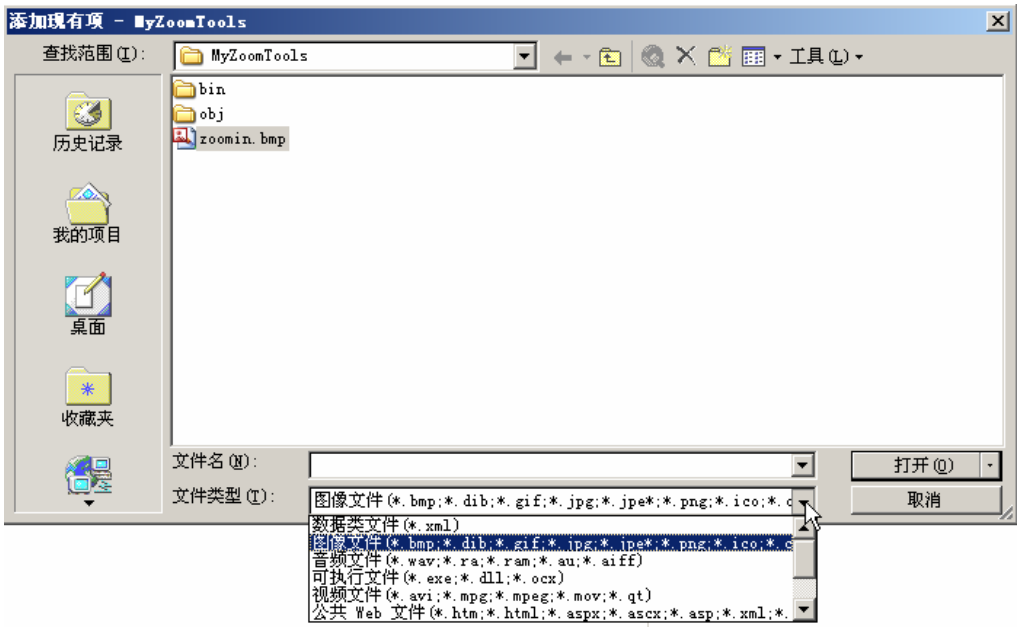


图 1-35 项目右键菜单

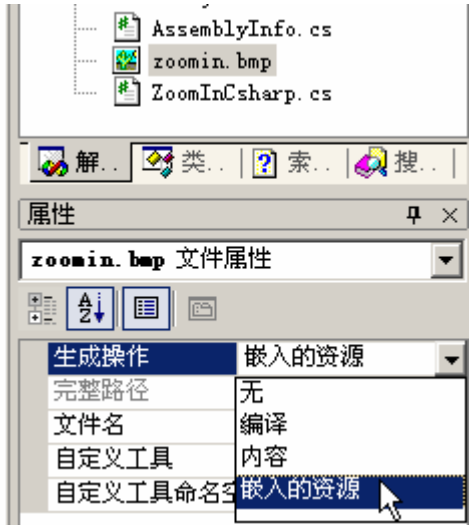
➢ 在“添加现有项”对话框中，点击“文件类型”下拉列表框，并选择“图像文件”，如图 1-36 所示。



如图 1-36 “添加现有项”对话框

➢ 浏览到所需的图标文件 zoomin.bmp，并点击对话框中的“打开”按钮，将图标文件拷贝到项目目录下。

➢ 在“解决方案资源管理器”中，选中刚加入的图标文件，在其属性窗口中将“生成操作”属性设置为“嵌入的资源”，如图 1-37 所示。



如图 1-37 设置图标文件的属性

➢ 返回代码窗口，在类的构造器中添加如下代码：

```
public ZoomInCsharp()
{
    m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"zoomin.bmp"));
    if (m_bitmap != null)
    {
        m_bitmap.MakeTransparent(m_bitmap.GetPixel(0,0));
        m_hBitmap = m_bitmap.GetHbitmap();
    }
}
```

➢ 在类的声明部分添加如下代码：

```
public class ZoomInCsharp : ICommand
{
    ……
    [DllImport("gdi32.dll")]
    static extern bool DeleteObject(IntPtr hObject);
    ...
```

注：DeleteObject 函数用于释放.NET 环境下不被管理对象。

- 在类代码中，添加如下析构函数：

```
~ZoomInCsharp()
{
    if (m_hBitmap.ToInt32() != 0)
        DeleteObject(m_hBitmap);
}
```

- 最后，在 ICommand::Bitmap 属性中添加如下代码，返回该命令的位图句柄。

```
public int Bitmap
{
    get
    {
        return m_hBitmap.ToInt32();
    }
}
```

- **Expose the ZoomInCsharp class to COM**

我们开发的 DLL 文件要在 ArcMap 环境下使用，ArcGIS 要求我们开发的命令类应该是一个 COM 类。所以在.NET 环境下开发的.NET 类必须暴露为 COM 类。

- 在“解决方案资源管理器”中，鼠标右键点击项目 MyZoomTools， 然后点击“属性”；
- 在“MyZoomTools 属性页”对话框中，选择“配置属性”，然后点击“生成”，在右边的窗格中，将“为 COM Interop 注册”设置为“True”，如图 1-16 所示，点击“确定”按钮，返回代码窗口。

注：“为 COM Interop 注册”设置为“True” 将调用 the Assembly Registration Tool (Regasm.exe). This will add the information about your class to the registry that a COM client would expect to find.

- 在代码窗口中，将光标置于类的声明之前，调用 GuidGen.exe 工具（点击“工具”菜单—“ESRI GuidGen”子菜单，如图 1-38 所示）为 ZoomInCsharp 类生成一个 GUID：

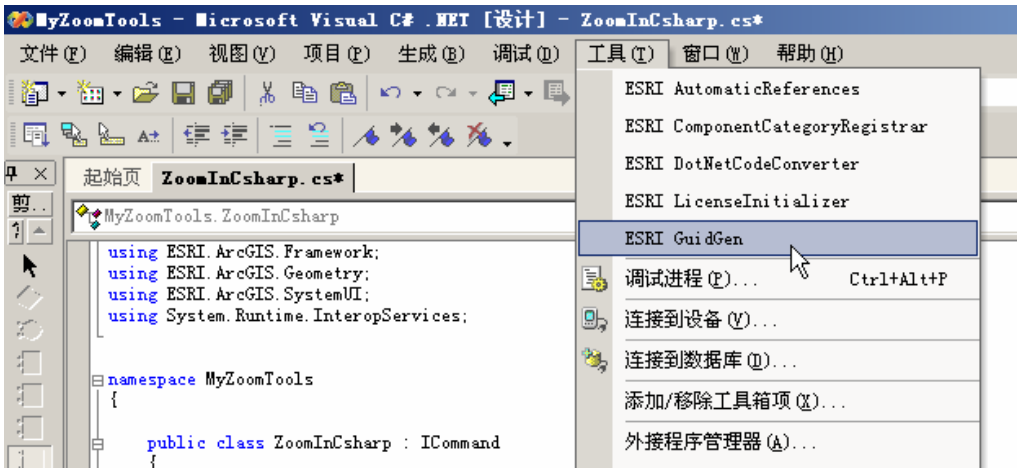


图 1-38 生成 GUID

```
[Guid("fe414d99-589a-46fa-9149-aa0054e9cb98")]
 public class ZoomInCsharp : ICommand
     {
    ...
```

- **添加 COM 类目（COM category）注册函数**

项目中的每个类需要进行注册与反注册。

- 将光标放在类中变量声明的底部，点击”工具”菜单，选择”ESRI ComponentCategoryRegistrar”选项，会弹出一个组件注册对话框，如图 1-39 所示。

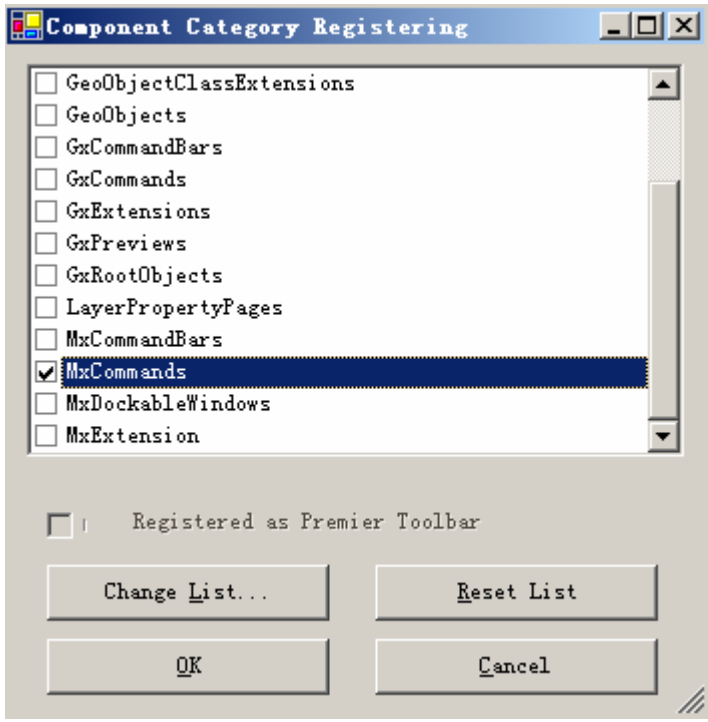


图 1-39 COM 类别注册

- 选择”MxCommands”后点击”OK”，将看到在代码区中自动添加了几行代码，它是用于组件注册使用的，使用这个 Addin 工具在工程的 BIN 目录中会生成两个批处理文件：Reg.Bat 及 UnReg.Bat，用于手工注册或手工取消注册生成的 DLL 文件。

```
#region "Component Category Registration"
[ComRegisterFunction()]
static void Reg(string regKey)
{
        MxCommands.Register(regKey);
}

[ComUnregisterFunction()]
static void Unreg(string regKey)
{
        MxCommands.Unregister(regKey);
}
#endregion
```

“ESRI Component CategoryRegistrar”在 .NET 开发环境中不会默认出现，它是 ArcGIS 提供的一个工具，专门用于 COM 组件注册的。使用 ArcObjects 组件库二次开发的 DLL 组件，必须在不同的类别中注册才可以使用，如实现了 ICommand 接口的组件必须注册到 MxCommands 组件库中。如果 .NET 环境里面没有这个选项，打开”ArcGIS 安装目录＼DeveloperKit＼Addins”，寻找到”RegisterDotNetAddin” 批处理文件，双击它就可以了。再启动 .NET 的开发环境时，在”工具”中就可以找到这个工具了。

➢ 编译项目

- 在.NET 开发环境中，通过“生成”--“生成解决方案”将项目编译为 DLL 文件，如图 1-40 所示。

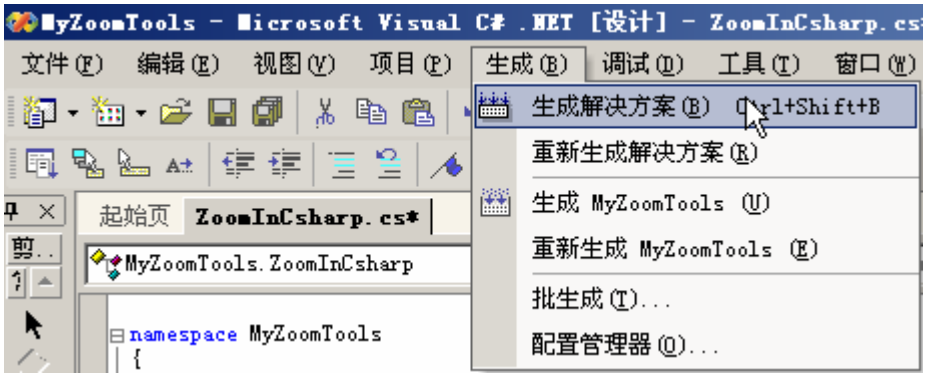


图 1-40 编译项目（解决方案）

➢ 在 ArcMap 中使用创建的命令

现在，可用在 ArcMap 环境下，使用刚创建的 ZoomInCsharp 命令。

- 启动 ArcMap，并打开一地图文档；
- 鼠标右键点击某一工具条，打开工具定制对话框，在工具定制对话框中点击 Commands 标签，选择 “MyZoomTools”类别，并将“Zoom In x 0.5 C#”命令拖放到某一工具条上，如图 1-41 所示，关闭工具定制对话框，就可以使用“Zoom In x 0.5 C#”命令了。

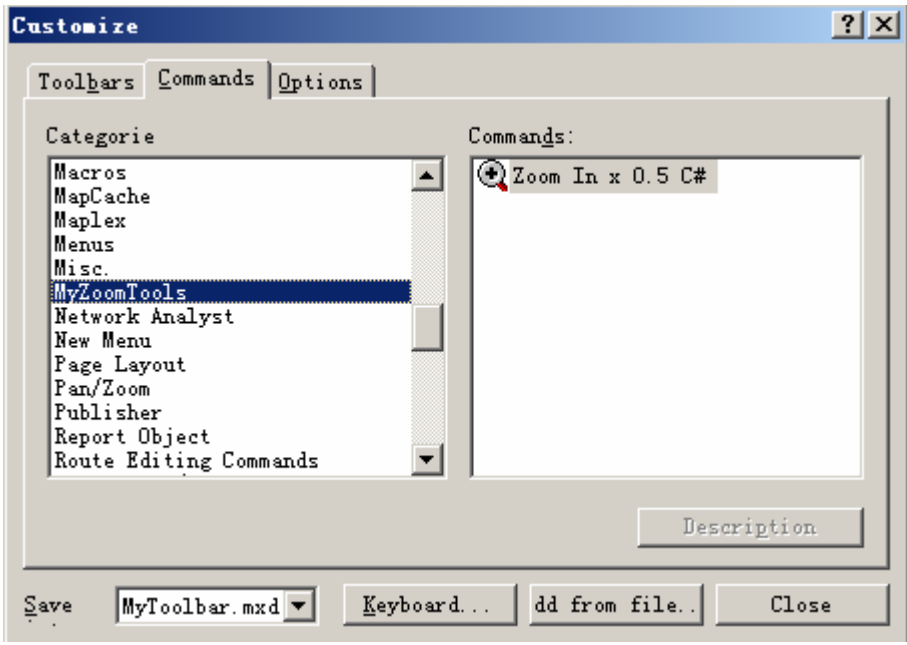


图 1-41 将新建的命令拖放到工具条上

1.6.3 使用控件构建独立应用程序

这种开发方式可以结合实际应用的要求，使用 ArcObjects 的可视化控件开发出独立界面的 GIS 应用系统，具有很高的灵活性和扩展性能，因而是很多行业和部门的选择。

本节实例实现一个简单的 GIS 功能--拉框放大。

打开 VisualStudio.net 2003 开发环境，点击“新建项目”，在“Visual C#项目”中选择“Windows 应用程序”，如图 1-42 所示。键入项目名称 ControlsApp，选择保存的位置后点击“确定”按钮，系统将会新建一个窗体，在“工具箱”中点击右键后出现菜单，选择“添加 / 移除项”，会弹出一个“自定义工具箱”对话框，选择“.NET Framework 组件”一栏，如图 1-43 所示。



图 1-42 新建 Windows 应用程序项目对话框

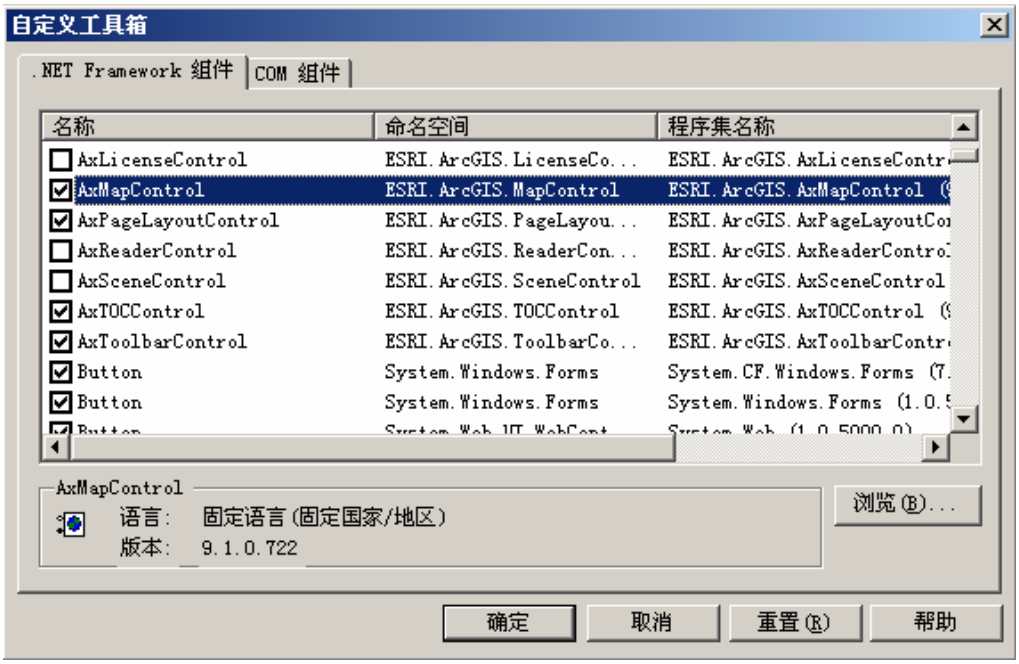


图 1-43 自定义工具箱对话框--引入控件

选择 AxMapControl 和 AxPageLayoutControl 两项，点击确定后在“工具箱”中出现这两个控件的图标，如图 1-44 所示。



图 1-44 工具箱上的 ArcObjects 控件

现在可以像使用普通组件一样使用这两个 ArcObjects 控件了。选择 AxMapControl 将它拖放到一个窗体上，将创建一个名为 axMapControl1 控件，可以设置它的大小等属性，如图 1-45 所示。

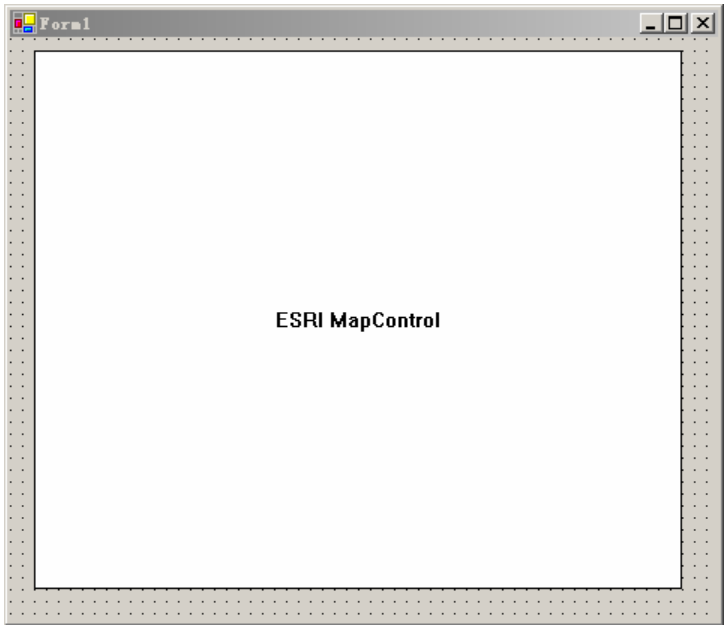


图 1-45 将 MapControl 控件放到窗体上

为了实现不同的功能，需要引入添加不同的引用。比如程序需要使用到 Geodatabse 的内容，就必须引入 GeoDatabase 对象库；如果是显示图形，则需要的是 Display 对象库。

在“解决方案资源管理器”中，点击“添加引用”按钮，会弹出一个“添加引用”对话框，在“.NET”一栏中寻找到“ESRI.ArcGIS.Geodatabase”，双击此项，在“选定的组件”中将会出现需要引入的组件名，点击确定。但是这样并没有把组件引入代码，读者还需要在 Form1 的代码区的最顶端写下如下代码：

```
using ESRI.ArcGIS.Geodatabase
```

这样就可以在这个窗体中使用 Geodatabse 组件库中定义的接口、类、方法和属性了。

选中 axMapControl1 控件，点击右键选择“属性”，会弹出一个“属性”对话框，如图 1-46 所示。

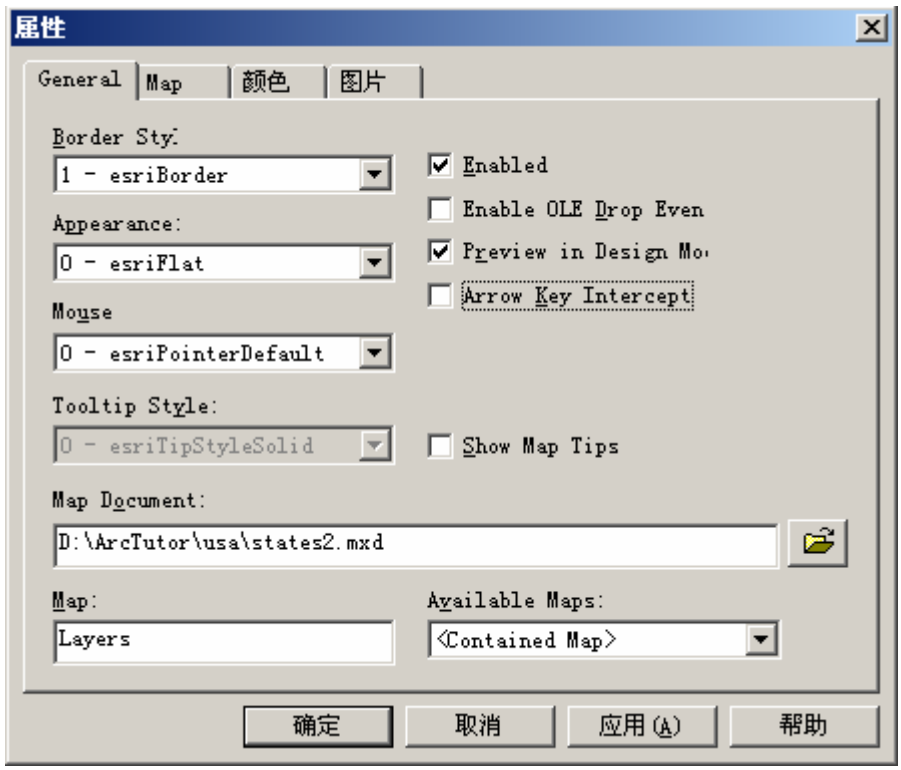


图 1-46 控件属性对话框

使用这个属性对话框，可以往 axMapControl1 中引入地图文档文件或者单独的 shape 文件、图层、要素类等，它提供了一种快速导入数据的方法。当然，开发者完全可以通过编程的方式导入地理数据，使用代码方法的灵活性更高。本例中，通过打开“Map Document”的按钮，引入一个已经存在的 MXD 文件。

通过这个属性对话框，还可以设置控件的外观，如边框类型、外观、鼠标样式

也可以在颜色和图片状态条中设置地图的颜色属性和添加背景图片。

打开 Form1 的代码区，在代码区最上端键入下面两行：

```
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry ;
```

在类 Form1 的变量声明部分，声明两个全局变量 pMap 和 pActiveView：

```
public class Form1 : System.Windows.Forms.Form
    {
        IMap pMap ;
        IActiveView pActiveView ;
        ……
```

在Form1的Load事件键入如下代码：

```
private void Form1_Load(object sender, System.EventArgs e)
    {
        pMap = axMapControl1.Map; //获取控件中的地图
        pActiveView = pMap as IActiveView;     //QI
        pActiveView.Extent =axMapControl1.FullExtent;//显示全局视图
        pActiveView.Refresh(); //设置后刷新地图
        }
```

在 axMapControl1 的 MouseDown 事件中键入如下代码：

```
private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
    {
        axMapControl1.MousePointer =
ESRI.ArcGIS.SystemUI.esriControlsMousePointer.esriPointerCrosshair;//设置鼠标样式为
Crosshair
        IEnvelope pEnv ;
        pEnv = axMapControl1.TrackRectangle(); //通过TrackRectangle方法获得一个
Envelope类型对象
        pActiveView.Extent =pEnv; //将地图的视图范围设置为新范围对象pEnv
        pActiveView.Refresh(); //设置后刷新地图

    }
```

编译程序，在窗体出现后使用鼠标在控件上拖曳，会发现显示范围随着鼠标拖曳的范围变化而变化了，如图 1-47、48 所示。

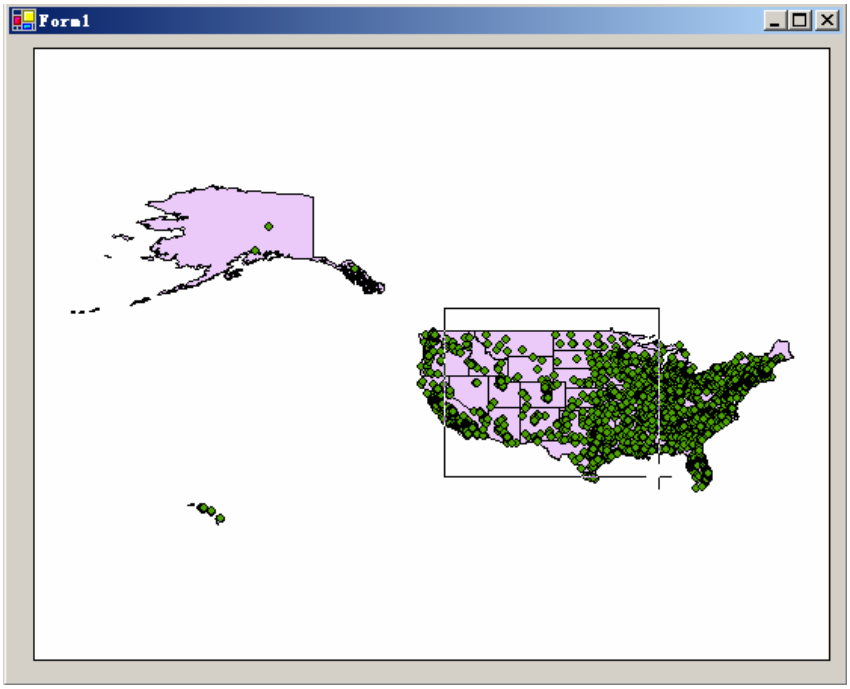


图 1-47 拉框放大示意图 1

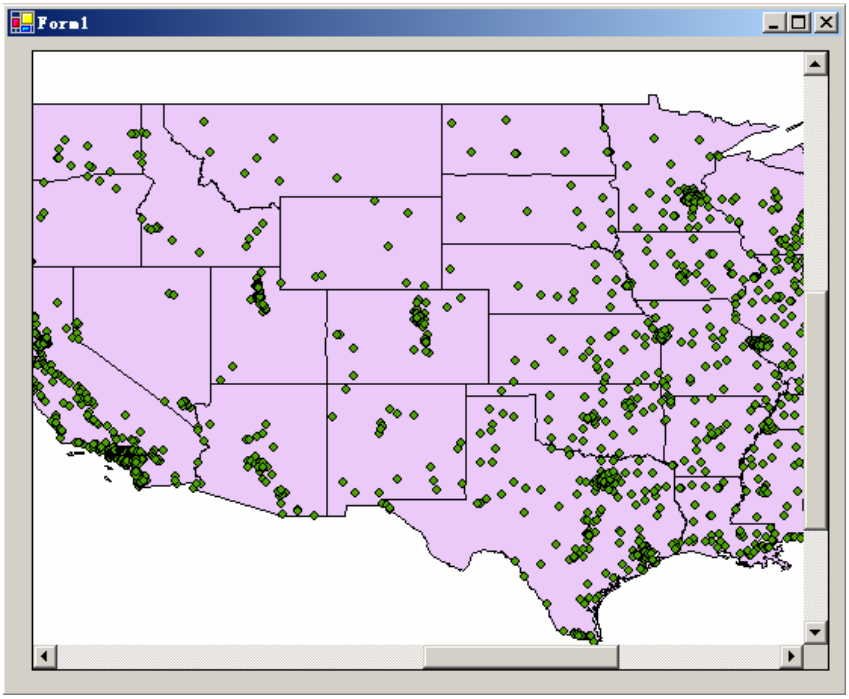


图 1-48 拉框放大示意图 2

这样就完成了一个简单的独立 ArcObjects 程序开发例子。在整个过程中 ArcObjects 程序员不需要打开 ArcMap 程序，生成的程序外观完全是程序员自己设定的。通过这个简单的例子，读者可以对 ArcObjects 开发独立程序的方式有个简单的了解和认识。但是这个程序的运行必须在一台安装了 ArcGIS 或 ArcGIS Engine 的机器上。

通过对以上三种 ArcObjects 开发方法的简单介绍，相信读者对于 ArcObjects 的各种二次开发方式及其优缺点都有了一定的了解。实际上，没有哪一种开发方法是最好的，不同的二次开发方法适合不同的情况，能够最好满足工程要求的方法才是最好的方法。

2 ArcMap 用户界面定制

使用 ArcObjects 进行 GIS 二次开发，最简单的方式是在 ArcGIS 程序自带的 VBA 环境中编写代码。VBA 可以使用 ArcGIS 的内在结构框架，它提供了 Application 和 ThisDocument 两个全局变量，它们直接指向程序本身和程序使用的文档对象上，而且在 VBA 中所有可以使用的对象库都被自动引入了开发环境，使用起来非常方便。

对于 ArcMap 程序而言，Application 对象和 MxDocument 对象是最重要的。前者掌管了程序本身的产生和中止，后者控制着程序运行过程中使用的地理数据。

使用 ArcObjects 可以快速定制 ArcMap 和 ArcCatalog 等程序的界面和开发特定的 GIS 功能。在 ArcMap 框架下，开发者可以使用各种用户界面元素--菜单、对话框、工具栏等——定制用户界面。

本章讲解的是如何在 VBA 环境下，利用 ArcMap 提供的框架来定制程序。注意：本章代码不能在未安装 ArcInfo 等桌面版程序的状态下运行。

学习完本章后，读者应该了解的内容有：

- 什么是 Application 对象
- 文档对象
- 如何定制 ArcMap 程序的界面
- 使用 UIControl
- 简单介绍扩展 ArcGIS 的方法

2.1 Application 对象

每一个运行的 ArcMap 程序都是一个 Application 的对象实例，Application 代表了应用程序本身。在 ArcMap 中，以 Application 对象为根本，Application 管理 ArcMap 程序的启动和关闭、加载扩展模块等。

Application 对象下面的有几个重要的对象

- DataWindow：负责管理数据窗口；

- Extension：负责管理所有的 DLL 扩展模块对象，读者自己编写的 DLL 扩展组件要在 ArcMap 里面使用也是它管理的；
- AppDisplay：负责管理地理数据的图形显示对象；
- Document：负责管理地理数据和元素 (Element) 对象，它也是 ArcObjects 编程中最常用的对象之一，堪称 ArcMap 框架结构里面的“二号人物”；
- StatusBar：可以用于改变程序的状态栏外观；
- Templates：是 Application 启动后必须使用的文档模板对象。

关于这些对象的结构读者可以查看图 2-1。

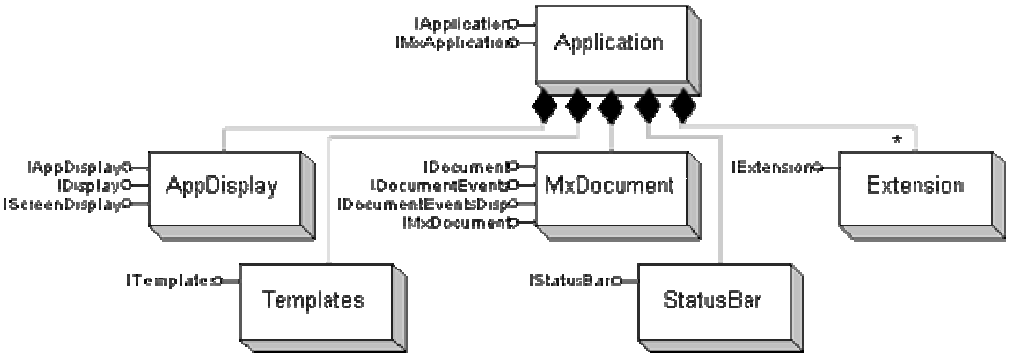


图 2-1 Application 相关对象结构图

在 VBA 环境中，Application 始终是一个全局变量，在任何地方都可以直接使用使用下面的代码获得并显示程序的名字：

```
MsgBox Application.Name
```

启动 ArcMap 的时候，系统会自动产生一个 Application 对象。它支持很多接口来管理众多的属性和方法，以完成不同类型的应用，以下介绍它的主要接口。

2.1.1 IApplication 接口

Application 类是在 Framework 库中定义的，它是应用程序的一个抽象类。IApplication 定义了 ArcGIS 中所有应用程序的一般功能，其属性、方法如图 2-2 所示。

这个接口管理程序的 caption、document、template 和 VBE 等 ArcMap 和 ArcCatalog 应用程序中都拥有的方法和属性。

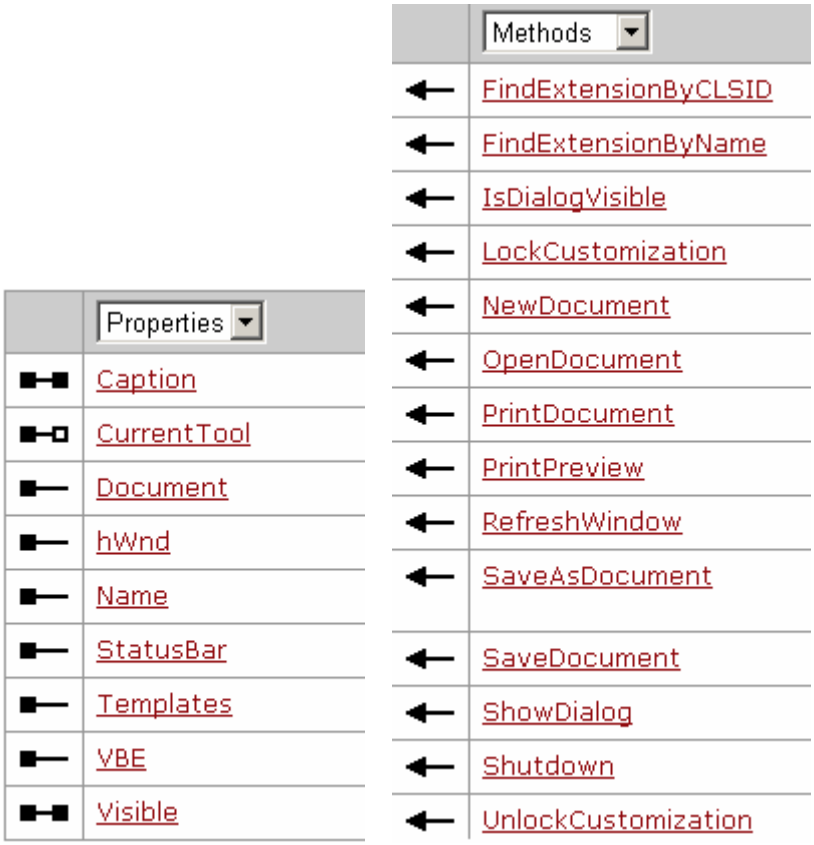


图 2-2 IApplication 中的属性、方法

当直接使用 VBA 环境进行开发的时候，它意味着可以直接联系到程序本身的某些属性和特征，如下面的代码：

```
Sub Caption()
    Dim pApp As IApplication
    Set pApp = Application
    '返回应用程序的标题"
    MsgBox(pApp.Caption)

End Sub
```

这个接口也定义了方法去打开、保存和打印文档，选择是否锁定文档的定制状态。

下面是打开一个 MXD 文件的代码段：

```
    Dim pApp As IApplication
    Set pApp = Application
```

```
pApp.OpenDocument("d:\test.mxd")
```

2.1.2 IMxApplication 接口

这是 ArcMap 程序特有的接口，用于管理 SelectionEnvironment、Display、Paper 和 Print 等对象。这些对象和地图的显示、打印输出有关， IMxApplication 中定义的属性、方法如图 2-3 所示。

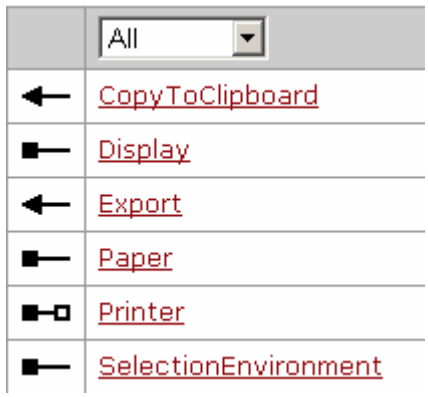


图 2-3 IMxApplication 中定义的属性、方法

SelectionEnvironment 是用于设置系统中被选择元素或要素属性的对象,它可以改变选择集的样式。下面是一个改变 SelectionEnvironment 的例子：

```
Dim pSelEnv As ISelectionEnvironment
Set pSelEnv = New SelectionEnvironment
'改变选择集颜色
Dim pColor As IRgbColor
Set pColor = New RgbColor
pColor.RGB = RGB(255, 0, 0)
Set pSelEnv.DefaultColor = pColor
```

2.1.3 IWindowPostion 接口

任何有可视化窗体的类都实现了 IWindowPosition 接口，如 DockableWindow、CommandBar 和各种 Application 类。Application 作为一个有窗体的类当然也不例外，这个接口定义了 Application 的尺寸和位置，如 width 和 height 确定了窗体的大小，而 left 和 top 规定了窗体的左上角距离屏幕的左上角之间的距离。

这个接口还定义了一个属性 state，它用于确定程序窗口的形式，如

esriWSNormal(正常模式)、esriWSMaximize(最大化模式)、esriWSMinimize(最小化模式)和 esriWSFloating(浮动模式)四种。

下面是一个改变窗体位置及大小的宏命令：

```
Sub MoveWindow()
    Dim pWinPos As IWindowPosition
    Set pWinPos = Application
    pWinPos.Height = 300
    pWinPos.Width = 300
    pWinPos.Left = 50
    pWinPos.Top = 50
    pWinPos.State = esriWSMinimize

End Sub
```

这个接口的方法也可以移动窗体和动态改变窗体的大小。

2.1.4 IEnumprinterNames

这个接口定义的对象用于与安装了 ArcGIS 软件的电脑相连接的打印机对象。

在基于 ArcMap 框架的 VBA 编程中，Application 对象扮演着“终极对象”的角色。**但如果使用的是 MapControl 进行编程，这个对象就没有什么用处了，MapControl 和 PageLauoutControl 才是各自的主角。**这其中的详细情况将在后面的章节中给读者进行叙述。

2.2 Document 对象

只要打开 ArcMap 程序，就会有一个默认的地图文档 (文件格式是 mxd)存在。文档文件有多种，如 ArcMap 中的文档称为 MxDocument 对象，ArcCatalog 程序中的文档是 GxDocument，在 ArcScene 中称为 SxDocument。

在程序中一次只能有一个文档文件存在，它在数据存储和显示中扮演了一个关键的角色，并且保存了许多非常重要的对象：一个或者多个 Map、一个 PageLayout、一个 TOCView、StyleGallery 和 DataGraphs 等。用户可以通过 IApplication::Document 来获取一个文档对象的指针或者在 VBA 编程的框架中直接使用 ThisDocument 这个全局变量。

理解文档对象对于 ArcGIS 的使用和编程都非常重要。打开 ArcMap，在其中载入各种类型的地理数据，进行不同类型的地理分析，进行地图着色制作各种专题图，

使用 VBA 开发定制不同 GIS 功能，最后产生一张精美的地图。用户肯定希望能够将劳动成果保存起来，而不愿意下一次再进行逐个导入数据、进行分析和编写代码等重复劳动。这些过程可以保存在一个 mxd 文件上，即保存在一个文档文件中。一句话概之，文档对象保存了 ArcGIS 中某一次操作的状态。

当载入一个 mxd 文件后，打开 VBE(可以使用 Ctrl+F11)，会看到如图 2-4 中的窗体。

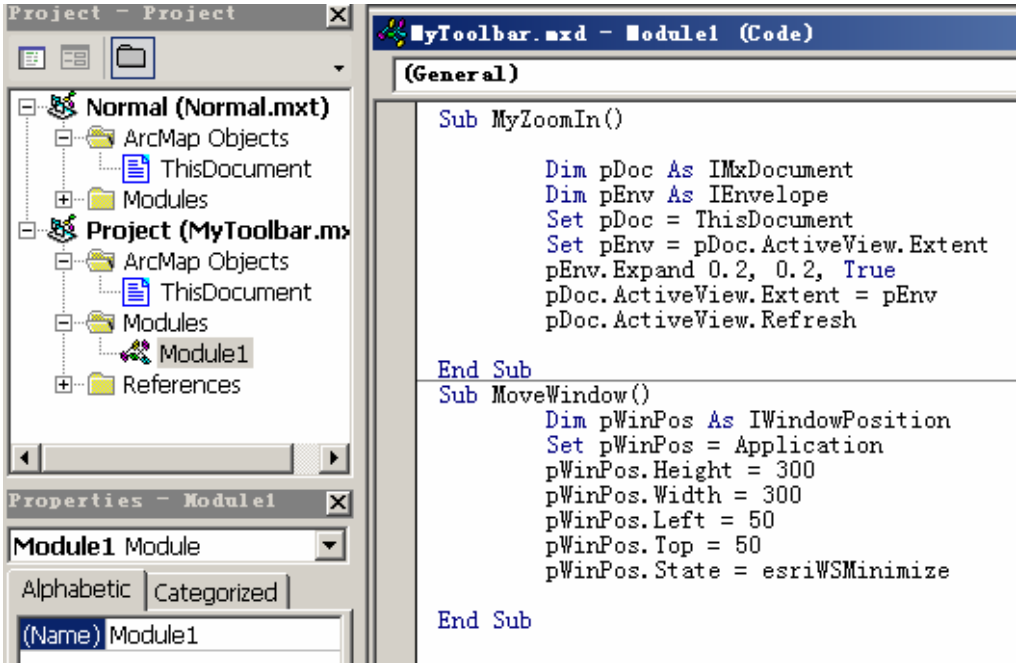


图 2-4 ArcMap VBE 环境中引入的对象

上图就是程序框架包含的文档文件，文档文件包括三种：Normal 模板、工程模板和项目文档。

打开 ArcMap 程序后，如果没有任何文档文件载入，系统就会自动产生一个从 normal.mxt 派生出来的无标题的 mxd 文件。normal.mxt 是生成其他模板文件和文档文件的基础，其他的文档文件都是在这个模板上派生出来的，对 normal.mxt 的任何改动都会影响到其他的模板和文档。如果删除这个文件，ArcMap 将会在启动的时候自动重建它，读者可以在“C:\Documents and Settings＼<用户名>＼Application Data＼ESRI＼ArcMap＼Templates”文件夹中(Windows2000 操作系统)寻找到这个文件。

Normal 模板保存了默认的命令、工具条、菜单栏和 TOC 等的位置，打开 ArcMap 程序后出现的界面都是这个文件设置安排的。

如果将在 VBE 中写的代码保存在 normal.mxt 文件上，任何从这个文件派生的

工程模板和项目文档都可以使用这些 VBA 代码，它的影响范围是全局级的。如果一个项目文档不是从工程模板中产生，那它就一定是从 normal 模板中产生的。上图的文档就是从默认模板中产生出来的。

工程模板是从 normal.mxt 派生出来的新模板，它的扩展名也是 mxt。工程模板对 normal.mxt 文件进行了一定的定制，也可以用于产生文档文件，对于项目模板的修改会影响到任何使用这个项目模板产生的文档文件。在“<ArcGIS 安装目录>\Bin\Templates”文件夹中可以看到多个已经预定义的模板文件，图 2-5 是在 ArcGIS 的“NEW”对话框中可以使用的工程模板文件。

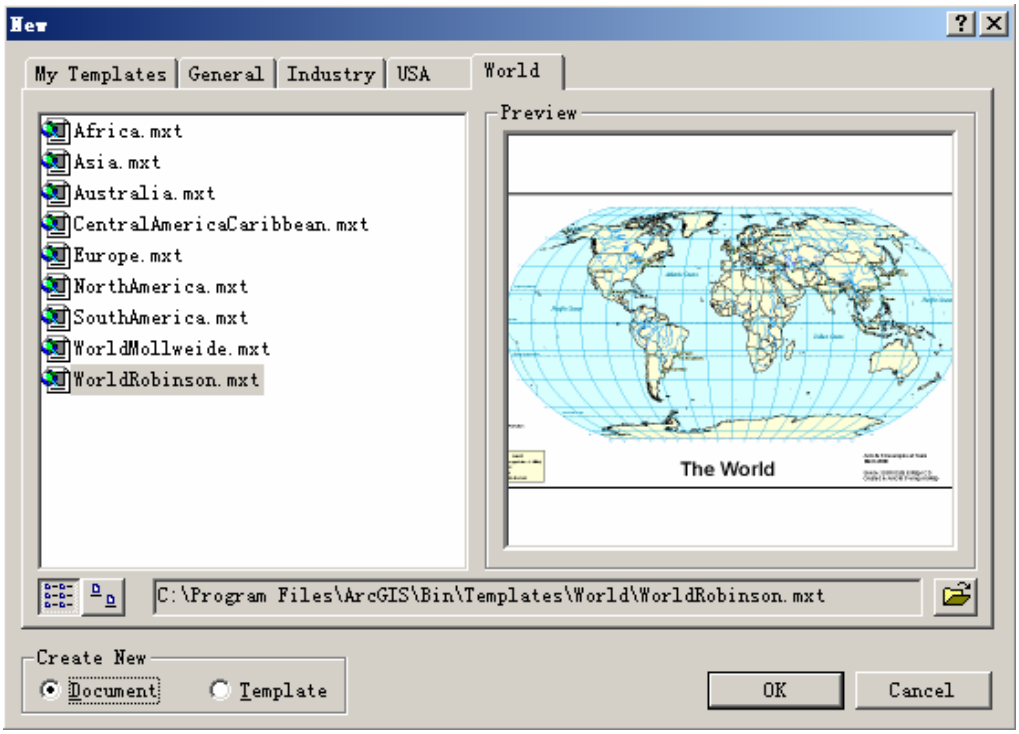


图 2-5 ArcMap 的各种过程模板

模板文件方便用户传播自己定制的代码。在一个项目组内部或者在一个公司内部，如果都需要某些相同的功能的时候，可以把这些功能写在一个工程模板文件上分发出去。不同的开发者就可以按照自己的需求在这个工程模板上产生不同的项目文档文件以进行自己的工作，那些公共的功能就不必一一重写了。

项目文档可以使用任何一种模板文件产生出来，它的扩展名是 mxd。在 ArcMap 中直接用来进行操作的文档文件就是这种类型的。

ArcMap 中的文档文件被称为 MxDocument 对象，以下介绍它的主要接口：

2.2.1 IMxDocument 接口

IMxDocument 是文档对象的默认接口，也是 MxDocument 类最主要的文档接口，该接口拥有很多文档的缺省属性设置，如缺省颜色，缺省字体等。IMxDocument 中定义的属性、方法如图 2-6 所示。

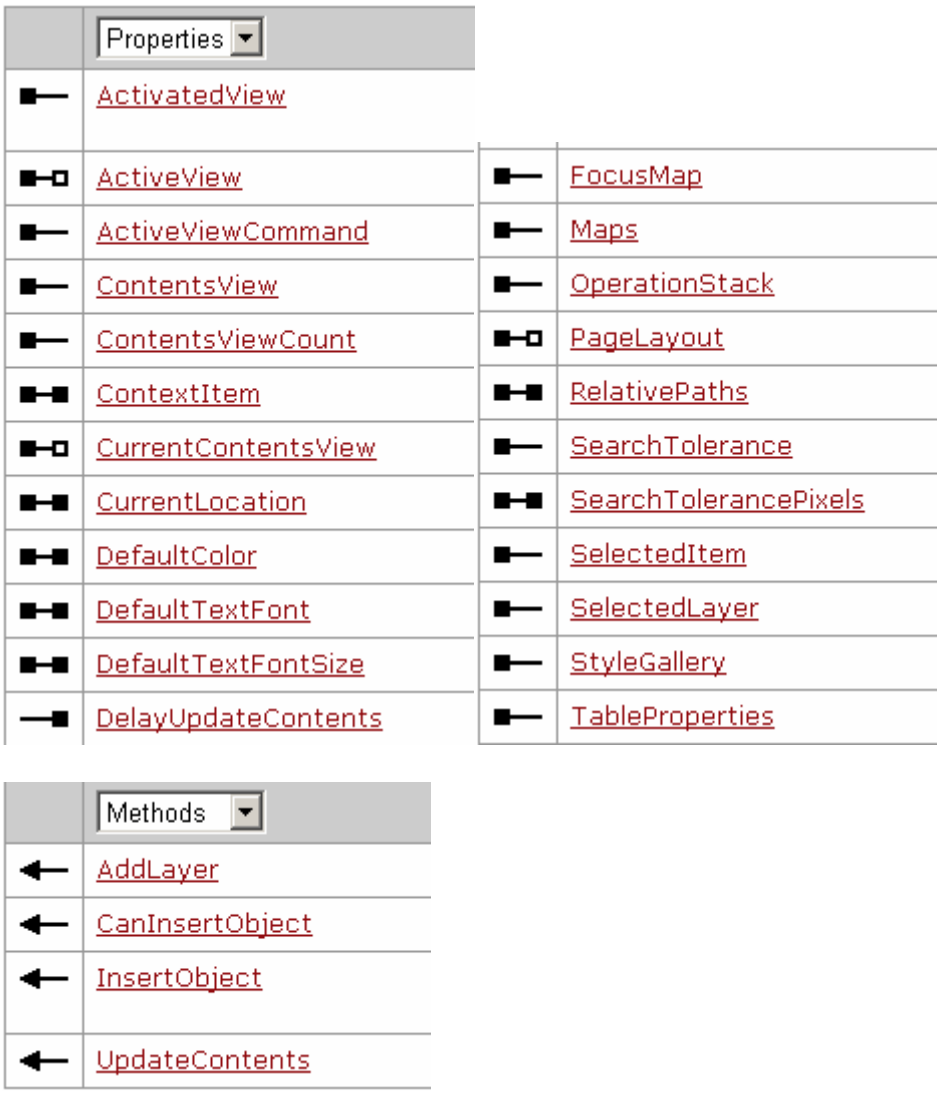


图 2-6 IMxDocument 中定义的属性、方法

IMxDocument 也是获取 Map 对象的主要接口，这个对象扮演着地理数据显示和地理数据容器的双重身份，通过文档对象的 ActiveView 属性获得 Map 的数据显

示身份；而 FocusMap 属性获得当前正在使用的 Map 对象的数据容器身份。一个文档对象可能拥有多个 Map 对象，但是在同一时刻内仅仅只能有一份地图处于使用状态。

下面的代码段是获得文档的当前使用地图对象：

```
Dim pApp As IApplication
Dim pDoc As IMxDocument
Dim pMap As IMap
Set pApp = Application
Set pDoc = pApp.Document
Set pMap = pDoc.FocusMap
```

如果是要获得其他的地图对象，可以使用：

```
Dim pMap As IMap
pMap = pDoc.Maps.Item(2)
```

2.2.2 IDocumentDefaultSymbols 接口

IDocumentDefaultSymbols 接口是文档缺省符号设置接口，用于设置缺省颜色、符号，图 2-7 为其定义的属性，该接口中不包含方法。

在 ArcMap 里面打开一个 MXD 文件后，会发现很多属性都是默认设置的，这些默认的设置是可以改变的。打开“Tools”的“Options"选项，会弹出一个“Options 对话框”面板，如图 2-8。

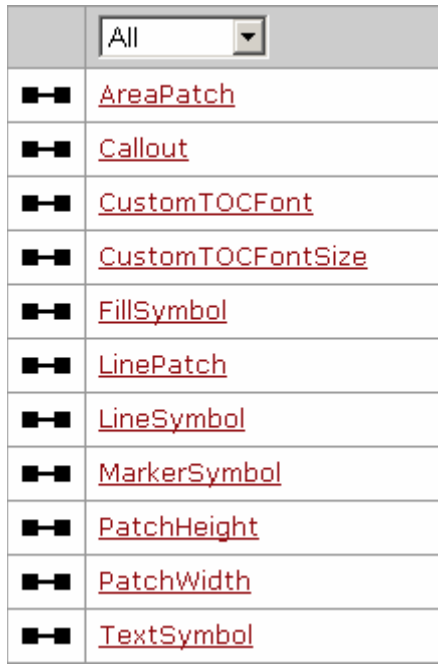


图 2-7 IDocumentDefaultSymbols 中的属性

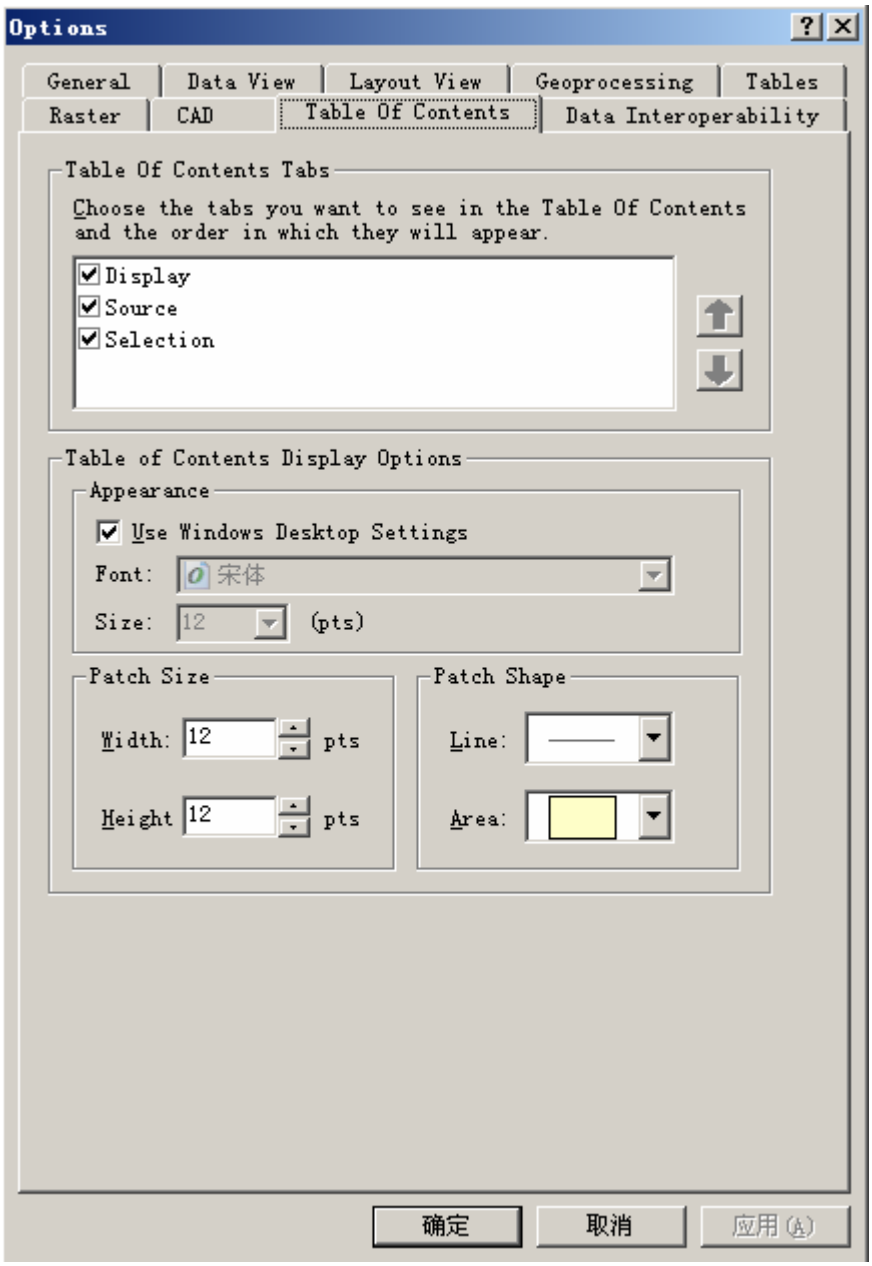


图 2-8 Options 对话框

在对话框中可以看到 TOC(Table of Content)的字体、颜色等属性和 Patch 的各种属性特征。使用 IDocumentDefauhSymbols 接口定义的属性，读者可以修改这些内容。下面是一个改变默认 TextSymbol 的例子：

```
Sub changeTextSym()
    Dim pDoc As IDocument
```

```
    Set pDoc = ThisDocument
    'QI，获得文档缺省符号对象
    Dim pDocSym As IDocumentDefaultSymbols
    Set pDocSym = pDoc
    Dim pRGB As IRgbColor
    Set pRGB = new RgbColor
    pRGB.Red = 110
    pRGB.Green = 60
    pRGB.Blue = 150
    Dim pTextSym As ITextSymbol
    Set pTextSym = new TextSymbol
    pTextSym.Color = pRGB
    pDocSym.TextSymbol = pTextSym
End Sub
```

2.2.3 IDocumentEvents、IDocumentEventsDisp 接口

IDocumentEvents 是 MxDocument 对象的一个外向事件接口，使用这个接口可以监听与地图文档对象有关的事件，如 newDocument 事件，CloseDocument 事件等。需要注意的是这些接口的使用方法与内向接口是不一样的。图 2-9 为 IDocumentEvents 中定义的事件。

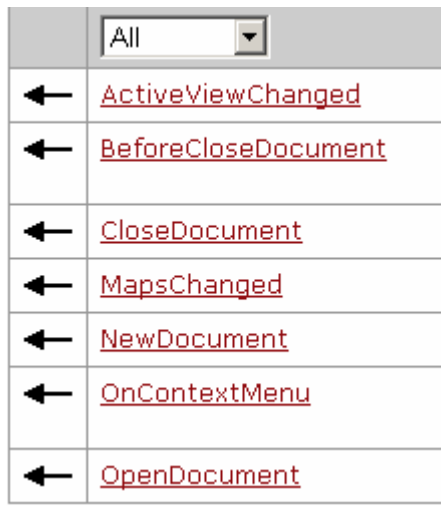


图 2-9 IDocumentEvents 中的事件

IDocumentEvents 和 IDocumentEventsDisp 非常相似，但是后者可以在 VBA 的编辑器中自动使用。使用 VBA 进行编程的时候，可以直接使用文档对象的一系列事件。IDocumentEventsDisp 定义了诸如 MapsChanged、CloseDocument 等事件。

下面的例子介绍如何使用 newDocument 事件：

打开 ArcMap 程序，选择“Tools”的“Macros”，然后打开 VBE 环境，会看到图 2-4 中的情况。无论是 normal 还是 project 都有一个 ArcMap Objects 文件夹，其中的有一个 ThisDocument 的类模块，这是一个全局变量。

双击 normal 中的 ThisDocument 对象，会弹出 VBE 的代码区，在“(general)”下拉框中选择 MxDocument，这时右边的下拉框会自动出现事件名，其中的事件都是文档对象可以使用的，在其中选择“newDocument”，代码区内会出现下面的代码：

```
Private Function MxDocument_newDocument() As Boolean

End Function
```

在这个函数中输入下列简单代码：

```
MsgBox("你在创建一个新文档!!")
```

然后保存 VBA 代码，关闭 VBE 返回 ArcMap 程序，点击新建文档的图标，新建一个文档文件。会发现自动弹出一个对话框“你在创建一个新文档”。这是因为 MxDocument 监听了 newDocument 事件的发生而作出的响应。

2.3 定制 ArcMap 的界面

ArcMap 的外观界面可以通过代码进行定制。ArcMap 的外表是由诸如状态栏、DockableWindow、AcceleratorTable、CommandBars 及 CommandBar 等可视化部分组成的。它们主要用于构造程序的界面，其中有的对象由 Application 直接管理，有的由 MxDocument 对象管理。通过改变这些对象，可以修改 ArcMap 的界面，但这些内容仅仅能在 VBA 环境中定制或者使用 DLL 组件开发，不能在使用 ArcObjects 控件的程序中完成。

2.3.1 定制状态栏(StatusBar)

状态栏 StatusBar 是 ArcGIS 程序中用于显示程序操作状态的区域，当用户按下某个按钮时，它都会显示相应的提示信息。StatusBar 是 Application 对象直接管理的，程序员可以通过 StatusBar 的定制可以改变状态栏的外观。

StatusBar 类实现了 IStatusBar 接口，后者定义的属性和方法可以方便地改变 StatusBar 的外观。状态栏一般被划分为几个区域，这些区域称为面板(panel)，用户可以通过 IStatusBar::Panes 属性来得到目前处于显示状态的任何一个面板对象。IStatusBar 中定义的属性、方法如图 2-10 所示。

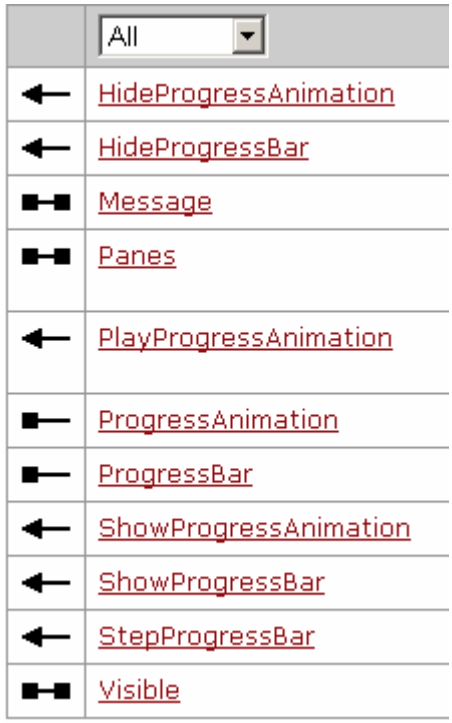


图 2-10 IStatusBar 中定义的属性、方法

esriStatusBarPanes 的参数定义了让哪一个 panel 处于显示状态，ArcObjects 中可以使用多个 esriStatusBarPanes 值来确定程序状态栏的样式。如 esriStatusBarPanes 值为 0 代表 MainPane，esriStatusBarPanes 值为 1 代表 AnimationPane，esriStatusBarPanes 值为 2 代表 PositionPane，esriStatusBarPanes 值为 3 代表 PagePositionPane。通常程序使用的就是这四个 pane，因而状态栏的 esriStatusBarPanes 缺省值是 0+1+2+3=7。

如果希望出现所有的 pane，可以设置这个值为 255。

下面是改变状态栏的例子：

```
Sub StatusBar()
    Dim pApp As IApplication
    Set pApp = Application
    Dim pStatusBar As IStatusBar
    Set pStatusBar = pApp.StatusBar
    pStatusBar.Panes = 255
End Sub
```

下面简单介绍一下 IStatusBar 接口的几个属性：

Message 属性：IStatusBar 接口的 Message 属性是可以在状态栏上显示提示字符信息，一般这个字符都是显示在状态栏最左边的 MainPane 上。

IStatusBar 接口的 ProgressBar 属性和 HideProgressBar、ShowProgressBar 和 StepProgressBar 都是用于控制状态栏中的进度条对象。如果进度条会显示的话，它将出现在主状态栏的面板上。进度条对象是 StepProgressor，它实现了 IStepProgressor 接口。

下面是一个显示进度条例子代码：

```
Sub ProgressBar()
    Dim pStatusBar As IStatusBar
    Dim i As Long
    Dim pProgbar As IStepProgressor
    Set pStatusBar = Application.StatusBar
    Set pProgbar = pStatusBar.ProgressBar
    '设置状态栏的属性且让其运行
    pProgbar.Position = 0
    pStatusBar.ShowProgressBar "载入...", 0, 900000, 1, True
    For i = 0 To 900000
        pStatusBar.StepProgressBar
    Next
    pStatusBar.HideProgressBar

End Sub
```

除了显示进度条外，状态栏上还可以出现动画进度条，通过 IStatusBar::AnimationProgressor 可以得到状态栏上的动画进度条对象。关于这个对象的详细情况，可以参看开发帮助的内容。

2.3.2 定制可停靠窗口(DockableWindow)

DockableWindow 是一种能够处于浮动状态或者停靠在主程序上的窗体，起到显示数据的辅助作用，它是一种无模式(modeless)的窗体。如 ArcMap 的 TOC 对象和 ArcCatalog 的 TreeView 都是一种 DockableWindow 对象。开发者可以使用 IDockableWindowManager::GetDockableWindow 方法，通过传入浮动窗体的 UID 属性来得到一个 DockableWindow 对象。

IDockableWindow 接口能查询一个浮动窗体的属性，如 Caption、ID 和名字等。它也有方法返回这个窗体以及是否可见或者停靠在哪一个区域。

DockableWindow 类也实现了 IWindowPosition 接口用于改变浮动窗体在屏幕上的位置。注意，使用这个接口的任何属性和方法前，要保证它必须处于浮动状态。下面是一个移动 TOC 的代码例子：

```
Public Sub MoveTOC()
    Dim pDocWinMgr As IDockableWindowManager
    Dim pTOC As IDockableWindow
    Dim pWinPos As IWindowPosition
    Set pDocWinMgr = Application            'QI
    '使用UID来获得一个浮动窗体
    Set pTOC = pDocWinMgr.GetDockableWindow(arcid.TableofContents)
    Set pWinPos = pTOC          'QI
    If pTOC.IsVisible Then
        pTOC.Dock esriDockFloat
        pWinPos.Move 100, 100, 120, 320
    End If
End Sub
```

程序员也可以在程序中新建自己的 DockableWindow 对象，使用 IDockableWindowDef 接口中的方法可以完成这个任务。这个接口能够让程序员设置属性，如标题和名字；ChildHWND 属性用于确定哪一个对象将放在这个浮动窗体上，而 OnCreate 方法可以让窗体进行初始化。当这个窗体被卸载的时候，可以调用 OnDestroy 方法。

需要提醒读者注意的是，使用 IDockableWindowDef 创建 DockableWindow 的时候仅仅是定义了一个类，而并非实际存在这个对象，只有当将这个类注册到 DockableWindow Component Categories 中的时候，程序才能使用定义的窗体去建立一个实际的浮动窗体对象。

2.3.3 快捷键表

为了方便用户快速操作程序，很多软件都提供了快捷键的功能，它是实际键盘按键和命令之间的一种映射，当用户按下与某个命令相关联的按键时，这个命令就将被执行。如 Ctrl+C 就是读者最熟悉的系统拷贝操作。

ArcMap 也一样提供了这种方式，它使用 AcceleratorTable 对象来保存这种双向的映射关系，用户通过 IMxDocument::AcceleratorTable 可以得到保存在一个文档文件中的这个快捷键表对象。这个方法也说明了快捷键设置其实是保存在一个文档文件中的。

AcceleratorTable 中保存着 Accelerator(快捷键)的列表，可以通过 IAcceleratorTable 接口定义的属性和方法获得或者添加一个快捷键对象，每一个快捷键对象代表了一个实际可以使用的快捷键。使用这个接口，还可以知道快捷键表中的快捷键数目。Accelerator 类实现了 IAccelerator 接口，使用此接口可以设置一个快捷键的属性。

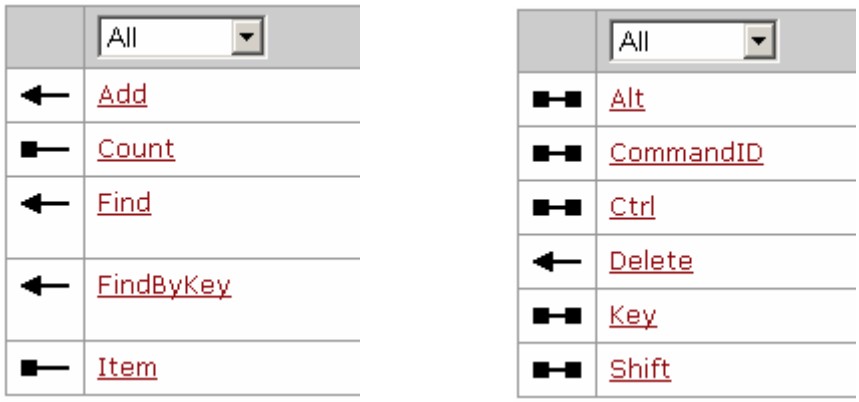


图 2-11　IAcceleratorTable（左）、IAccelerator（右）中定义的属性、方法

下面是使用 Add 方法添加一个 Accelerator 到 AcceleratorTable 中的例子：

```
Public Sub AssignAccelerator()
    Dim pAccTable As IAcceleratorTable
    Set pAccTable = ThisDocument.Accelerators
    Dim addAcc As Boolean
    addAcc = pAccTable.Add(ArcID.File_AddData, vbKeyD, True, False, False)
End Sub
```

现在读者就可以使用“Ctrl+D”的快捷键来执行“Add Data”的命令了。

2.3.4 CommandBars 和 CommandBar 对象

CommandBars 是一个与文档相关的命令栏(commandbar)的集合，后者是一个抽象类，其子类可以是工具条(Toolbar)、菜单栏(Menu bar)、菜单(Menu)和弹出菜单(ContextMenu)，通过这个对象可以获得程序中某个 CommandBar 的指针。

一个工具条(Toolbar)是由很多命令对象组成的，这些命令对象(Commands)按照其特点分为 UIControl、Button、Macro 和 Menu 四种。其中 UIControl 对象又有四种类型：Button、Tool、ComboBox 和 EditBox，如图 2-12 所示。

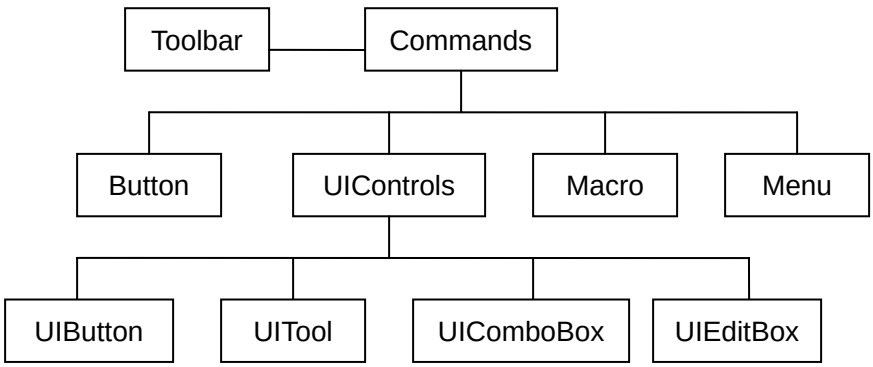


图 2-12 工具栏的结构

ICommandBars 接口是 CommandBars 类的默认接口。它允许程序员设置 CommandBar 的属性，也可以让程序员新建、查找和隐藏 CommandBar。如 LargeIcons 可以设定是否使用大图标；ShowToolTip 用于设置是否显示工具提示；Create 方法可以快速新建一个空白的工具栏或者快捷菜单。使用这个方法需要输入的参数有 CommandBar 的名称和类型。

ArcObjects 定义的 CommandBar 的类型有三种：esriCmdBarTypeToolbar (工具栏)、esdCmdBarTypeMenu (菜单栏)和 esriCmdBarTypeShortcutMenu(快捷菜单栏)。

注意：不要使用 ICommandBars::Create 方法去新建一个 esriCmdBarTypeMenu 类型的菜单，应该使用 ICommandBar::CreateMenu 方法来代替。

CommandBar 类默认实现 ICommandBar 接口，它可以添加一个 Command、Menu 或者 Macro 到命令栏上。程序员可以通过 Item 属性得到 commandbar 中的每一个条目(Item)。

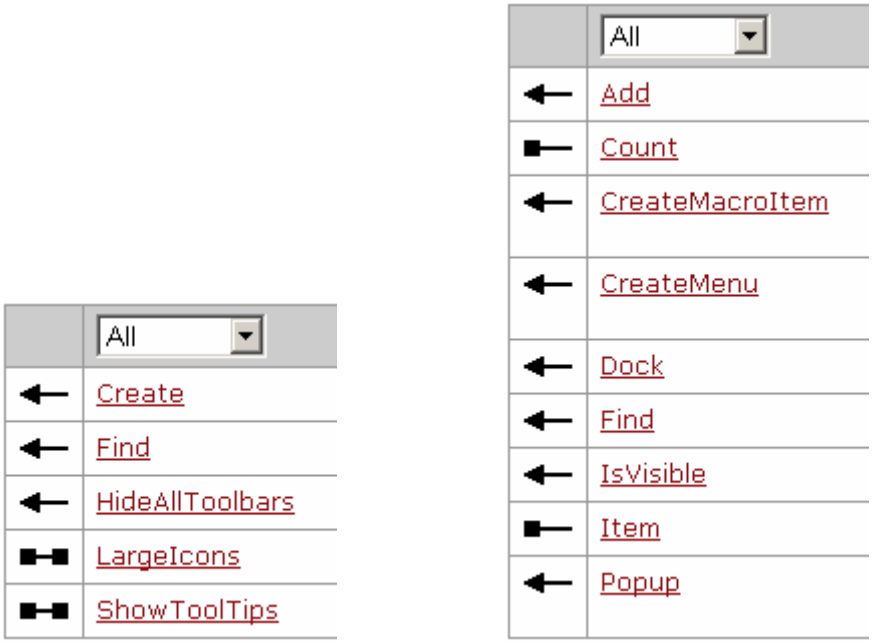


图 2-13 ICommandBars（左）、ICommandBar（右）中定义的属性、方法

使用 Dock 方法可以显示或者隐藏 CommandBar，并且可以将它放在程序窗体的某个位置上或者处于浮动状态。

下面的代码是产生一个 ToolBar：

```
Sub CreateBar()
    '得到CommandBars集合
    Dim pCmdBars As ICommandBars
    Set pCmdBars = ThisDocument.CommandBars
    '产生一个新的工具栏
    Dim pnewBar As ICommandBar
    Set pnewBar = pCmdBars.Create("MyBar", esriCmdBarTypeToolbar)
    '添加两个ArcMap已经存在的命令到新工具栏上
    pnewBar.Add ArcID.File_AddData
    pnewBar.Add ArcID.PanZoom_FullExtent
    '将产生的工具栏停靠在标准工具栏上
    pnewBar.Dock esriDockBottom, pCmdBars.Find(arcid.Standard_Toolbar)
End Sub
```

执行完这段代码后，读者将发现在程序上方的工具栏区域上出现了一个新的工具条，上面有两个命令，一个是“加载数据”，另一个是“全区域显示地图”。

用户自己定制的 CommandBar 有两种类型：基于文档的 CommandBar 和基于 COM 的 CommandBar。前者可以在程序的 VBA 环境中产生，它保存在一个文档或模板文件中；后者可以使用一种支持 COM 标准的编译语言来制作出来，它将保存在一个 DLL 文件中。

在 ArcGIS 程序中使用 Customize 对话框能够创建一个基于文档的 CommandBar，这种方法在第—章使用 VBA 进行二次开发的章节中已经仔细讲解过。

使用 DLL 方式创建 CommandBar 需要使用到命令条的定义对象，如 ToolBarDef 或者 MenuDef 去产生一个 CommandBar。一个实现了 IToolbarDef 的类仅仅是定义了一个工具条，它并没有实际产生一个工具条对象。只有这个类被注册到正确的组件库中，程序才会创建一个实际的工具条。

为了产生一个自定义的 Toolbar，需要实现在 SystemUI 库中 IToolbarDef 接口，这个接口用于定义一个工具条的属性。它提供的属性和方法可以设置工具条的标题、名称以及确定哪些 command 放置在这个工具条上。

为了产生一个菜单，需要实现 IMenuDef 接口，这个接口与 IToolbarDef 相似。如果产生一个快捷菜单，需要实现 IMenuDef 和 IShortcutMenu 两个接口，后者告诉程序该菜单仅仅被当做一个快捷菜单出现；如果需要产生的是一个根菜单(即菜单下面还有子菜单)，实现的接口是 IMenuDef 和 IRootLevelMenu。

2.3.5 CommandItem 对象

CommandItem 代表出现在 CommandBar 上的任何对象，如一个 Button、Menu 和 Tool 等。ICommandItem 接口是这个对象的默认接口，这个接口定义的方法可以获取或者设置 CommandItem 的属性，如设置标题、命令按钮图像、提示信息、显示样式和进行编组等。这个接口也定义了方法执行、删除、刷新和重置这些 CommandItem。

使用 ICommandBars::Find、ICommandBar::Find 或 ICommandBar::Item 都可以得到某个特定 CommandItem 的引用。

在 ArcObjects 开发中程序员可以创建的 Command 有三种：COM 命令、宏命令和 UIControl。

COM 命令是使用一种支持 COM 标准的工业化语言编写的，它以一个 DLL 组件的形式存在，可以编写下列类型的命令：

1. Buttons(按钮)和 Menu 是最简单的命令类型。Button 一般是作为—出现在

工具条上的图标的形式存在，而 Menu 是作为菜单栏的一个菜单条存在。

2. Tools 和 Buttons 非常类似，但是它可以和程序使用鼠标进行交互。如点击 tool 后可以在地图上拖曳一个矩形以改变地图的显示区域。

3. ToolControl(工具控件)类似下拉框或者编辑框。

4. MultiItem 对象是一种出现在菜单上的动态命令，它的出现取决于程序的状态。

有关命令、工具、菜单的开发详见“使用 ArcObjects 控件编程”一章。

2.4 UIControl 的使用

使用 VBA 可以在 ArcMap 和 ArcCatalog 里面产生新的 Button、Tool、Combo 和 Editbox，这些统称为 UIControl，这些 UIControl 也可以存在事件，供用户与之交互使用。

UIControl 是基于 VBA 的命令，这意味着这些命令是保存在一个文档文件或者模板文件中的。如果在文档中创建一个新的 UIControl，那仅仅可以在这个文档中使用；如果这个 UIControl 保存在一个模板中，那可以在任何基于这个模板创建的文件模板和文档上使用了。如果这个 UIControl 是保存在 normal 模板上，那么每一个文档文件都可以使用这个对象。

读者可以使用 ArcGIS 的定制面板来创建一个新的 UIControl 控件。当 Customize 对话框打开的时候，可以将新产生的 UIControl 控件放置到任何工具栏上。当然也可以为 UIControl 来编写代码，定制它的行为。

在 VBE 中，可以看到这些 UIControl 和普通的按钮、combo 一样，都有可触发的事件。这意味着程序员可以像使用普通控件一样来对它们进行编程。

UIControl 一共有四种：UIButtonControl、UIComboBoxControl、UIEditBoxControl 和 UIToolControl。

1. UIButtonControl 在点击后会类似一个命令或菜单项执行一个简单任务。可以通过编码的方式改变这个按钮的状态栏信息、提示信息和选择状态等属性。

2. UIToolControl 与一个实现了 ITool 的 COM 对象非常相似，它也可以用于和程序进行鼠标交互。

3. UIEditBoxControl 是一种可以放在工具条上的文本输入控件，和普通的文本

框组件差不多。当改变文本框中的文字时候，会触发 Change 和 Keydown 事件。

4. UIComboBoxControl 是一个可以添加在工具栏上的下拉组合框控件，它的属性和方法允许改变、添加和删除下拉组合框中的项目。当改变下拉框的文字和选项的时候，会触发 EditChange、SelectionChange 和 Keydown 事件。

由于本书的侧重点并不是使用 VBA 进行编程，因而对于 UIControl 的介绍到此为止，需要更多了解的读者可以阅读帮助或者阅读《Exploring ArcObjects》一书中的相关章节。

2.5 使用 Extension 扩展 ArcGIS

Extension 提供给开发者一个扩展 ArcGIS 核心功能的强大机制，这种机制是基于 COM 组件的。一个 Extension(扩展对象)可以给 ArcGIS 新建一个工具栏，也可以对应用程序发生的某些事件进行监听并做出响应。

当开发者要创建命令或者工具时，Extenion 经常会成为开发者参考引用的中心。而由于这些要创建的工具或者命令会经常共享数据或者访问 UI 工具，又因为每个运行着的程序(Application)仅仅只能有一个 Extension 的实例允许运行，使得 Extension 就成为存储数据和创建 UI 工具的地点。Extension 是被 Application 对象直接管理的。

任何被注册到程序的 Extension 都会自动被程序载入或者卸载掉。例如一个被添加到 ESRI Mx Extensions Category 的扩展对象会在 ArcMap 启动的时候自动启动，在 ArcMap 关闭的时候自动关闭，用户对此不需要进行任何的干预操作。

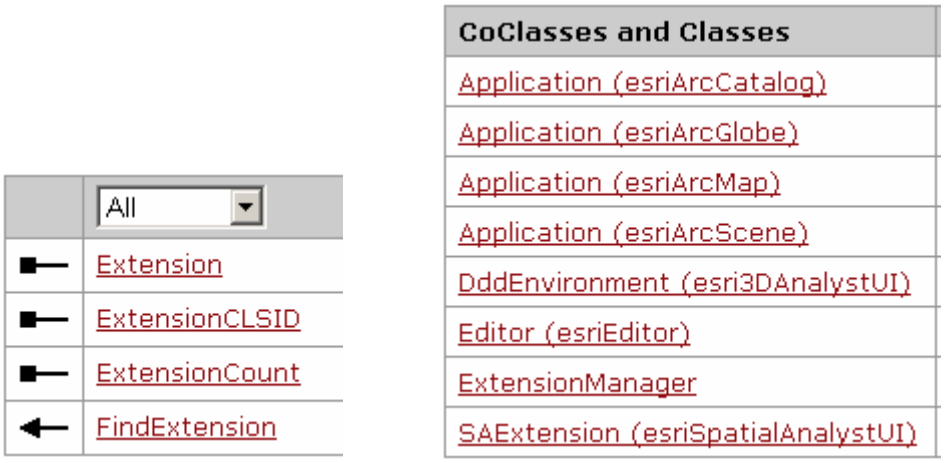
当使用扩展对象的时候，有必要了解程序的启动顺序，以便于理解扩展是如何被使用。基本的启动顺序是这样的：

1. 用户启动程序。
2. Application 自动产生。
3. 文档对象创建。
4. 扩展对象被载入。
5. 如果 ArcMap 是通过双击一个文档文件打开的，这个时候文档将被载入。否则在新文档创建后，如果用户打开一个文档文件，则将载入这个文档。

6. 程序启动完毕。

扩展被载入的次序是不能被控制的，程序需要使用合适的扩展类别来被载入程序。这就是为什么创建的 DLL 文件需要被“ComponentCategoryRegistrar"工具注册的原因。

Application 有一个 IExtensionManager 接口，通过这个接口的属性可以得到特定 Extension 的引用，或者计算有多少个 Extension 已经被程序装载。



(a) (b)

图 2-14 (a) IExtensionManager 中定义的属性、方法

(b) 实现 IExtensionManager 接口的组件类

要使用这个 manager 可以通过使用 QI 实现。另外，需要知道的是其他一些对象也可以执行 IExtensionManager 接口。例如 ArcMap 的 Editor 工具条是一个管理 Editor Extensions 的 Extension。所以，Editor 对象也执行 IExtensionManager 接口。

以下这段代码使用 Application 的 Extension manager 对软件装载的所有 Extension 进行循环检测，然后报告各 Extensions 的名称。

```
Dim pExtMgr As IExtensionManager
Dim pExt As IExtension
Dim i As Integer
  Set pExtMgr = Application
For i = 0 To pExtMgr.ExtensionCount - 1
    Set pExt = pExtMgr.Extension(i)
    MsgBox(pExt.Name)
Next
```

实际上，以上这种方法并不是得到 Extension 引用的唯一方法，IApplication 接口还提供了 FindExtensionByCLSID 和 FindExtensionByName 方法用于得到 Extension 的引用。

开发者通常用 IExtension 接口来查询一个扩展的属性或者创建自己的扩展。这个接口定义了两个方法：ShutDown 和 StartUp。ShutDown 用于关闭一个扩展对象，而 StartUp 则用于初始化一个扩展对象。传进 Extension 的初始值是一个 Object 类型，这是因为用户无法清楚地知道这个扩展对象将使用的数据类型。

有关 Extension 的例子可以参考“ArcGIS Developer Help”提供的例子“Sample Extension”。

本章讨论了如何使用 VBA 环境来进行 ArcMap 的界面定制，什么是 Application，什么是文档和模板，如何产生不同类型的自定义控件，如何使用 Extension 来开发 ArcGIS 的扩展模块等内容。这些内容都是在基于一个已经存在的程序框架中构建的，它只是带领读者步入 ArcObjects 开发的大门。从下一章开始，读者将了解使用独立控件开发时需要使用的各种对象和接口，那是本书最重要的内容部分。

3 使用 ArcObjects 控件编程

为了能够快速建构一个具有 GIS 功能的独立应用程序，ArcObjects 给开发者提供了多个可视化的控件，如 MapControl、PageLayoutControl、TOCControl 和 ToolbarControl 等。

MapControl 和 PageLayoutControl 对应于 ArcMap 桌面应用程序的“数据”和“布局”视图。MapControl 控件主要应用于地理数据的显示和分析；PageLayoutControl 是用于地图的修饰和整理，可以用于生成一幅成品地图。MapControl 封装了 Map 对象，而 PageLayoutControl 封装了 PageLayout 对象。这种封装简化了开发过程，但是并不限制对这些细粒度的 ArcObjects 对象的访问。

MapControl 和 PageLayoutControl 不仅可以读取地图文档，它们还可以写入地图文档(*.mxd)。这两个控件都实现了 IMxContents 接口，使地图文档(MapDocument)对象可以将 MapControl 和 PageLayoutControl 的内容写到一个新的地图文档中。

TOCControl 和 ToolbarControl 控件分别对应于 ArcMap 中的“Table of Contents”、各种工具条，这两个控件都要与一个“伙伴控件”协同工作。“伙伴控件”可以是 MapControl、PageLayoutControl、ReaderControl、SceneControl 或 GlobeControl。TOCControl 用“伙伴控件”来显示其地图、图层和符号体系内容的一个交互树视图并保持其内容与“伙伴控件”同步。ToolbarControl 可以驻留操作其“伙伴控件”的命令、工具和菜单。

在 Visual Studio .NET 环境中使用 ArcObjects 开发 GIS 应用程序，需要使用 ESRI interop 程序集(Interop Assemblies)，它服务于被放置在.NET 窗体上的、位于.NET 窗体控件(.NET Windows Controls)中的 ArcGIS 控件，这些程序集在托管的.NET 代码和非托管的 COM 代码之间起了桥梁作用。对 ArcGIS 控件(COM ArcGIS Controls)成员的引用都要经过 Interop 程序集，然后到达实际的 COM 对象。同样，也从 COM 对象经过 Interop 程序集到达.NET 应用程序。每个 ArcGIS 控件具有方法、属性与事件，它们能够被控件嵌入的容器(如，.NET 窗体)访问。每个控件对象及其功能可以与其他 ESRI ArcObjects 和自定义控件组合使用，创建用户化的客户应用程序。

ESRI interop 程序集为 ArcGIS 控件提供了的能够位于.NET 窗体上的控件，这些控件名前缀为“Ax”，如 AxMapControl、AxPageLayoutControl、AxTOCControl 和 AxToolbarControl.

在这一章将向读者详细阐述以上四个常用的控件，并结合实际，使用 C#.NET

开发平台解决一些基本应用问题。

3.1 MapControl 控件

MapControl 对应 ArcMap 中的数据视图，它封装了 Map 对象，并提供了额外的属性、方法、事件用于：

- 管理控件的外观、显示属性和地图属性；
- 添加并管理控件中的数据层（data layers）；
- 装载 Map 文档（mxd）控件中；
- 从其它应用程序拖放数据到控件中；
- tracking shapes and drawing to the display.

ArcGIS 的每种控件都有属性可以设置，它们可以在可视化环境中进行编辑。将控件拖放到窗体上后，用户可以右键点击，选中“属性”菜单来编辑这些属性，它们可以让用户在不编写一行代码的情况下快速建构一个 GIS 应用程序。

MapControl 实现的主要接口有：IMapControlDefault、IMapControl2、IMapControl3 和事件接口 IMapControlEvents2。

3.1.1 主要接口

3.1.1.1 IMapControlDefault 接口

IMapControlDefault 接口是地图控件缺省接口，多数开发环境自动使用这个接口定义的属性、方法。由于 MapControl 是一个自动化控件，当它被放到一个容器--如窗体上后，它会自动产生一个被称为 axMapControl1 的对象，这个对象可以直接使用缺省接口定义的属性和方法。这个接口也代表了控件最新版本的接口，MapControl 当前最新版本接口为 IMapControl3.

当需要使用这个接口的时候，可以使用下面的代码：

```
IMapControlDefault pMapControl;
pMapControl = axMapControl1.Object as IMapControlDefault;
```

3.1.1.2 IMapControl2 接口

这个接口是任何一个与 MapControl 相关的任务的出发点，如设置控件外观，设置 Map 对象或控件的显示属性，添加或者管理数据图层、地图文档，在控件上绘制图形和返回 Geometry 等。

IMapControl2 定义了控制控件外观界面的各种属性和方法，如 Appearance、BorderStyle 和 BackColor。它定义的主要方法有 TrackLine、TrackPolygon、TrackRectangle 和 Track Circle 方法让程序员可以直接在控件上获得 Geoemtry 对象，它还提供了 Drawshape 和 DrawText 方法让用户直接在控件上绘制图形。

3.1.1.3 IMapControl3 接口

该接口继承 IMapControl2，并增加了以下 8 个属性和一个方法：

- CustomProperty：设置自定义控件属性
- DocumentFilename：返回 MapControl 装入的地图文档的文件名
- DocumentMap：返回 MapControl 最后装入的地图（Map）名称
- KeyIntercept：返回或设置 MapControl 截取键盘按键信息，详见表 3-1
- Object：返回潜在的 MapControl 控件，When querying interface to IMapControl2 in Visual Basic .NET or Visual C# .NET the Object property or container specific code must be used. This is because .NET contains the real control inside a wrapper object known as an host.

  当使用 IMapControl2 接口时：

  ```
  IMapControl2 pMapControl;
  pMapControl = axMapControl1.Object as IMapControl2;

  或
  pMapControl = axMapControl1.GetOcx() as IMapControl2;
  ```

  当使用 IMapControl3 接口时：

  ```
  IMapControl3 pMapControl;
  pMapControl = axMapControl1.Object as IMapControl3;

  或
  pMapControl = axMapControl1.GetOcx() as IMapControl3;
  ```

- ShowMapTips：确定是否显示地图的 Map Tips
- TipDelay：设置 Map Tips 的延迟时间

- TipStyle：设置 Map Tips 的显示样式
- SuppressResizeDrawing ()：当控件尺寸发生变化时阻止数据实时重绘

esriKeyIntercept Constants：

Options for intercepting key presses handled by a controls container, these values can be combined into a bit mask.

表 3-1 esriKeyIntercept 可能取值

Constant	Value	Description
esriKeyInterceptNone	0	No keys are intercepted.
esriKeyInterceptArrowKeys	1	Intercepts the arrow keys, normally handled by the container to change control focus.
esriKeyInterceptAlt	2	Intercepts the Alt key, normally handled by a container to change focus.
esriKeyInterceptTab	4	Intercepts the Tab key, normally handled by the container to change control focus.
esriKeyInterceptEnter	8	Intercepts the Enter key, normally handled by the container to click the default button.

3.1.1.4 IMapControlEvents 2 接口

IMapControlEvents 2 是一个事件接口，它定义了 MapControl 能够处理的全部事件，如 OnMouseDown、OnAfterDraw、OnMouseMove 等，这些事件用户在建构独立程序过程中经常用到。如 OnAfterscreenDraw 是绘屏结束后触发的事件，OnExtentUpdated 是地图的 Extent 属性发生变化时触发的事件等。

3.1.2 MapControl 控件与 MXD 文件

文档对象在 ArcObjects 控件编程中变成了 MapControl 的一个属性而非“管理者”。MapControl 控件可以“链接”或“包含”地图文档，如果是“链接文档”，控件将保存对地图文档的引用，以后任何对于文档的修改都会出现在控件上；而使“包含文档”，

则控件会复制当前的文档状态，而不会受到后来修改的影响。

对于文档文件，MapControl 控件可以直接使用 LoadMxFile 方法来载入，这是最简单的方法。除此之外，读者也可以使用 IMapDocument 接口定义的属性和方法来加载一个 MXD 文件。下面是一个载入文档的例子：

```
private void LoadMapDocument()
    {
        System.Windows.Forms.OpenFileDialog openFileDialog2;
        openFileDialog2 = new OpenFileDialog();
        openFileDialog2.Title = "Open Map Document";
        openFileDialog2.Filter = "Map Documents (*.mxd)|*.mxd";
        openFileDialog2.ShowDialog();
        string sFilePath = openFileDialog2.FileName;
        if (axMapControl1.CheckMxFile(sFilePath))
        {
            axMapControl1.MousePointer =
esriControlsMousePointer.esriPointerHourglass;
            axMapControl1.LoadMxFile(sFilePath, 0,Type.Missing);
            axMapControl1.MousePointer =
esriControlsMousePointer.esriPointerDefault;
        }
        else
        {
            MessageBox.Show(sFilePath + " is not a valid ArcMap document");
            return;
        }
    }
```

在上例中，首先使用.NET 框架类库提供的 OpenFileDialog 对话框找到要打开的 MXD 文档 sFilePath，然后利用 AxMapControl 提供的方法 CheckMxFile()检查 sFilePath 是否为合法的 MXD 文档，如果合法，则调用 AxMapControl 提供的方法 LoadMxFile ()打开该 MXD 文档，否则，显示有关信息。

如果要装载某个地图文档中的特定地图，则可以使用以下方法：

```
private void LoadMapDocument2()
    {
        System.Windows.Forms.OpenFileDialog openFileDialog1;
        openFileDialog1 = new OpenFileDialog();
        openFileDialog1.Title = "Open Map Document";
```

```
            openFileDialog1.Filter = "Map Documents (*.mxd)|*.mxd";
            openFileDialog1.ShowDialog();
            string sFilePath = openFileDialog1.FileName;
            if (axMapControl1.CheckMxFile(sFilePath))
            {
                IArray pArray ;
                pArray = axMapControl1.ReadMxMaps(sFilePath,Type.Missing);
                int i ;

                IMap pMap;
                //遍历这些Map对象
                for(i = 0;i<=pArray.Count - 1;i++)
                {
                    //QI
                    pMap = pArray.get_Element(i) as IMap;
                    if (pMap.Name == "Layers")
                    {
                        //载入文档对象中特定的Map对象
                        axMapControl1.MousePointer =
esriControlsMousePointer.esriPointerHourglass;
                        axMapControl1.LoadMxFile(sFilePath, 0,Type.Missing);
                        axMapControl1.MousePointer =
esriControlsMousePointer.esriPointerDefault;
                        break;
                    }
                }

            }
            else
            {
                MessageBox.Show(sFilePath + " is not a valid ArcMap document");
                return;
            }
        }
```

IMapDocument 接口定义了操作和管理文档对象的方法和属性。MapDocument 类能够封装地图文档文件，如 mxd、mxt 和 pmf 等，它也可以封装 一个图层文件（*.lyr）。使用这个对象可以获取和更新一个文档的内容，设置文档文件的属性以及

读、写和保存一个文档文件（*.mxd）。

下面的例子是打开、保存、另存为一个文档文件，详细代码如下：

```
 IMapDocument    m_MapDocument;
private void LoadMapDoc()
    {

        m_MapDocument = new MapDocumentClass();

        try
        {
            //打开文件对话框选取MXD文件
            System.Windows.Forms.OpenFileDialog openFileDialog2;
            openFileDialog2 = new OpenFileDialog();
            openFileDialog2.Title = "Open Map Document";
            openFileDialog2.Filter = "Map Documents (*.mxd)|*.mxd";
            openFileDialog2.ShowDialog();
            string sFilePath = openFileDialog2.FileName;

            //将数据载入pMapDocument并与map控件联系起来
            m_MapDocument.Open(sFilePath, "");
            int i;
            for (i = 0 ; i<= pMapDocument.MapCount - 1;i++)
            {
                //一个IMapDocument对象中可能有多个Map对象，遍历每个map对象
                axMapControl1.Map = m_MapDocument.get_Map(i);
            }
            //刷新地图
            axMapControl1.Refresh();
        }
        catch( Exception ex)
        {
            MessageBox.Show(ex.ToString());
        }
    }

    private void SaveDocument()
    {
```

```
            //Check that the document is not read only
            if (m_MapDocument.get_IsReadOnly(m_MapDocument.DocumentFilename)
== true)
            {
                MessageBox.Show("This map document is read only!");
                return;
            }
            //Save with the current relative path setting
            m_MapDocument.Save(m_MapDocument.UsesRelativePaths,true);
            MessageBox.Show("Changes saved successfully!");
        }

        private void SaveAsDocument ()
        {
            //Open a file dialog for saving map documents
            saveFileDialog1.Title = "Save Map Document As";
            saveFileDialog1.Filter = "Map Documents (*.mxd)|*.mxd";
            saveFileDialog1.ShowDialog();

            //Exit if no map document is selected
            string sFilePath = saveFileDialog1.FileName;
            if (sFilePath == "")
            {
                return;
            }

            if (sFilePath == m_MapDocument.DocumentFilename)
            {
                //Save changes to the current document
                SaveDocument();
            }
            else
            {
                //SaveAs a new document with relative paths
                m_MapDocument.SaveAs(sFilePath, true, true);
                //Open document
                OpenDocument((sFilePath));
                MessageBox.Show("Document saved successfully!");
```

```
        }
    }
```

新建一个空的地图文档，并将其载如 MapControl 的代码如下：

```
    private void mnuNew_Click(object sender, System.EventArgs e)
    {
        //   IMap pMap = new MapClass();
        //   axMapControl1.Map = pMap ;
        MapDocument pMapDocument = new MapDocumentClass();
        string sFilePath = @"d:\temp\untitled.mxd";
        pMapDocument.New(sFilePath);
        pMapDocument.Open(sFilePath,"");
        axMapControl1.Map = pMapDocument.get_Map(0);
    }
```

在编程中使用 C#.NET 的异常控制语句 try…catch，是一种良好的结构化排错手段。这是因为在读取文件过程中常常出现无法预测的异常情况。

除此之外，MapControl 控件还可以使用 AddLayerFromFile 方法添加一个图层文件；使用 AddshapeFile 添加一个 Shape 文件；使用 AddLayer 快速添加一个图层、DeleteLayer 快速删除一个图层、MoveLayerTo 改变一个图层的索引等。

3.1.3 鼠标与控件的交互

用鼠标和地图控件进行交互是最常用的操作，例如改变地图显示范围、移动地图，在控件上绘制几何图形等，使用 ArcObjects 控件的好处在于用户不用进行复杂的坐标转换工作，而且可以直接使用控件的事件和产生鼠标在控件上的点对象。

- **使用鼠标拖曳确定地图显示的范围**

下面的代码放在一个 MapControl 控件的 OnMouseDown 事件中，当用户在控件上拖曳一个矩形后，地图视图的显示范围变为刚才拖曳的矩形区域。

```
    private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
    {
        // MousePointer属性可以改变鼠标指针的样式
        axMapControl1.MousePointer =
esriControlsMousePointer.esriPointerCrosshair;
        //改变地图控件显示范围为当前拖曳的区域
```

```
            axMapControl1.Extent = axMapControl1.TrackRectangle();
            //刷新地图
            axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewGeography, null, null);

        }
```

- **移动地图**

在 MapControl 中有一种更简单便利的方法 pan()来移动其中的地图。

下面的方法使用在 MapControl 控件的 OnMoseDown 事件中：

```
    axMapControl1.pan();
```

- **在 MapControl 控件中绘制图形**

MapControl 控件提供了直接在控件上绘制图形和文字的方法 DrawShape 和 DrawText，这两种绘制方法绘制的图形都是缓存（cache），而不能真正保存，一旦出现窗口重绘，这些图形就将消失。

下面方法首先 axMapControl1 的 OnMouseDown 事件中，使用 TrackPolygon() 方法产生一多边形，然后使用绘制方法 DrawShape()将这个多边形对象在 Map 视图上显示出来：

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            // MousePointer属性可以改变鼠标指针的样式
            axMapControl1.MousePointer =
esriControlsMousePointer.esriPointerCrosshair;
            //产生拖曳多边形
            IGeometry pGeom = axMapControl1.TrackPolygon();
            //刷新地图
            DrawMapShape(pGeom);
            axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewGeography, null, null);
        }


        private void DrawMapShape(IGeometry pGeom)
        {
            IRgbColor pColor;
            pColor = new RgbColorClass();
            pColor.Red = 220;
            pColor.Green = 112;
```

```
            pColor.Blue = 60;
            //新建一个绘制图形的填充符号
            ISimpleFillSymbol pFillsyl;
            pFillsyl = new SimpleFillSymbolClass();
            pFillsyl.Color = pColor;
            object oFillsyl = pFillsyl;
            axMapControl1.DrawShape(pGeom, ref oFillsyl);
        }
```

3.1.4 数据选择

在 MapControl 控件中用户可以很方便地使用 SearchByshape 方法来构造一个基于 Map 的选择集。

- **SearchByShape（）方法**

这个方法是由 IMap 定义的，它可以选择控件中所有图层上处于选择范围内的要素，并将其设置为一个选择集。

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            // MousePointer属性可以改变鼠标指针的样式
            axMapControl1.MousePointer =
esriControlsMousePointer.esriPointerCrosshair;
            //产生拖曳多边形
            IGeometry pGeom = axMapControl1.TrackPolygon();
          //清除以前存在的选择集
            axMapControl1.Map.ClearSelection();
axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewGeoSelection,null,null);
            axMapControl1.Map.SelectByShape(pGeom, null, false);
axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewGeoSelection,null,null);
        }
```

- **清除选择集的方法**

```
        axMapControl1.Map.Clearselection();
        axMapControl1.ActiveView.Refresh();
```

3.1.5 实现鹰眼功能

在很多 GIS 应用程序中都可以看到鹰眼窗口，使用这种窗体可以让用户很直观地看到主视图中的地图范围在整个地图范围内的位置，犹如鸟瞰一样，如图 3-1 所示。下面的例子将讨论使用这个功能的简单思路。

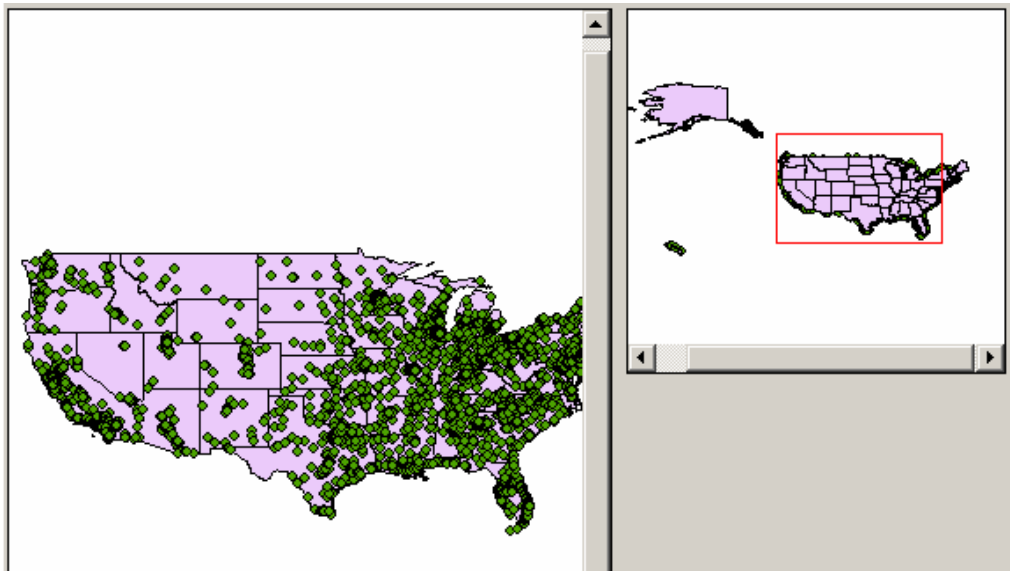


图 3-1　鸟瞰窗口

这个例子中有两个 AxMapControl 控件：主控件 axMapControl1 和鹰眼控件 axMapControl2. 要实现鹰眼功能，关键有两点，一是如何让两个控件显示的数据保持一致，另一点是如何绘制鹰眼控件中的显示方框。让两个控件的数据共享，可以在 axMapControl1 的 OnMapReplaced 事件中加入下列代码：

```
private void axMapControl1_OnMapReplaced(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMapReplacedEvent e)
        {
            IMap pMap;
            pMap = axMapControl1.Map;
            int i ;
            for( i = 0; i<= pMap.LayerCount - 1;i++)
            {
                axMapControl2.Map.AddLayer(pMap.get_Layer(i));
            }

        }
```

当 axMapControl1 中的地图被替换时时，鹰眼控件会自动加载主控件中所有的图层对象，实现两者数据的一致。有的读者也许试图使用传递两个控件之间的 Map 属性，这种方法当然是可以的，但是它的缺陷在于同时传递了地图对象的范围和比

例，鹰眼视图中不能够显示完整的地图。

在主控件（axMapControl1）中使用鼠标拖曳视图的时候，鸟瞰控件（axMapControl12）中出现红色矩形框，下面是实现的代码：

```
private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            if (e.button == 1)
            {
                //点击鼠标左键拖曳一个地图显示范围
                axMapControl1.Extent = axMapControl1.TrackRectangle();
                axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground, null,
null);

            }
            else if (e.button == 2)
            {
                //点击鼠标右键可以移动地图
                axMapControl1.Pan();
                axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground, null,
null);
            }
        }
```

主窗体视图的范围发生变化后，会触发控件的 OnExtentUndated 事件，绘制方框的方法就在这个事件中进行：

```
private void axMapControl1_OnExtentUpdated(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnExtentUpdatedEvent e)
        {
            //得到新范围
            IEnvelope pEnv ;
            pEnv = e.newEnvelope as IEnvelope;
            IGraphicsContainer pGraphicsContainer;
            IActiveView pActiveView;
            pGraphicsContainer = axMapControl2.Map as IGraphicsContainer;
            pActiveView = pGraphicsContainer as IActiveView;
            //在绘制新的矩形框前，清除Map对象中的任何图形元素
            pGraphicsContainer.DeleteAllElements();
            IRectangleElement pRectangleEle;
```

```
            pRectangleEle = new RectangleElementClass();
            IElement pEle;
            pEle = pRectangleEle as IElement;
            pEle.Geometry = pEnv;
            IRgbColor pColor;
            pColor = new RgbColorClass();
            pColor.RGB =255;
            pColor.Transparency = 255;
            //产生一个线符号对象
            ILineSymbol pOutline;
            pOutline = new SimpleLineSymbolClass();
            //设置线符号的属性
            pOutline.Width = 1;
            pOutline.Color = pColor;
            //设置颜色属性
            pColor = new RgbColorClass();
            pColor.RGB = 255;
            pColor.Transparency = 0;
            //设置填充符号的属性
            IFillSymbol pFillSymbol;
            pFillSymbol = new SimpleFillSymbolClass();
            pFillSymbol.Color = pColor;
            pFillSymbol.Outline = pOutline;
            IFillShapeElement pFillshapeEle;
            pFillshapeEle = pEle as IFillShapeElement;
            pFillshapeEle.Symbol = pFillSymbol;
            pEle = pFillshapeEle    as IElement;
            pGraphicsContainer.AddElement(pEle, 0);
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
        }
```

当主窗体移动的时候，在鹰眼窗体中会出现一个红色短形框，它是一个图形元素，可以在窗体重绘的时候仍然保存在视图上，因而读者需要在每次绘制一个新矩形的时候将前一个矩形框元素删除。

在主视图中拖曳或移动地图的时候，用户可以看到在鹰眼视图中有一个红色的矩形框在实时变化追踪主视图的显示范围。

把 MapControl 控件放到 VS.NET 的一个窗体，写下代码并开始编译调试，但

当用户关闭调试状态或者直接点击“关闭”按钮的时候，会有一个对话框弹出来，意思是内存泄露，无法正常关闭。解决这个问题的方法是：

在 form 的 closed 事件中添加如下代码：

```
private void Form1_Closed(object sender, System.EventArgs e)
{
    ESRI.ArcGIS.Utility.COMSupport.ArcObjectsUninitialize.Shutdown();

}
```

实现这个鹰眼功能的完整例子可参考本教材提供的例子 OverviewApp.sln.

3.2 PageLayoutControl 控件

PageLayoutControl 控件也是一个 ActiveX 组件，主要用于制图，它可以方便地操作各种元素对象，以产生一幅制作精美的地图对象出来。PageLayoutControl 中封装了一个名为 PageLayout 的组件类，它提供了在布局视图中控制元素的属性和方法。

与 MapControl 控件相似，PageLayoutControl 控件并不仅仅只是包含了一个 PageLayout 对象，它也拥有许多附加的事件、属性和方法。

Printer 属性用于处理地图打印时的一系列设定；Page 属性用于处理控件的页面效果；Element 属性则用于管理控件中的地图元素。由于地图总是“包含”在 Map 对象中，而 Map 对象是由一个 MapFrame 所持有，则 PageLayout 对象至少拥有一个 MapFrame 对象，而每个 MapFrame 中也拥有一个 Map。

图 3-2 是 PageLayoutControl 控件的属性对话框，用户可以使用这个对话框快捷地设置它的外观属性和引入文档文件，这个控件不能添加地图图层或地理数据，而是使用一个 MXD 文件来引入这些需要处理的数据。

PageLayoutControl 控件同样实现了多个接口，如 IPageLayoutControlDefault、IPageLayoutControl、IPageLayoutControl2 和 IPageLayoutControlEvents 等，这些接口定义了控件对象可以使用的属性、方法和事件。

PageLayoutControl 控件的属性和方法可以检查和载入 MXD 文件、实现添加元素和移动版式页面等在 ArcMap 程序中布局视图可以实现的功能。具体可以实现的方法和属性，读者可以查阅开发帮助的相关条目。

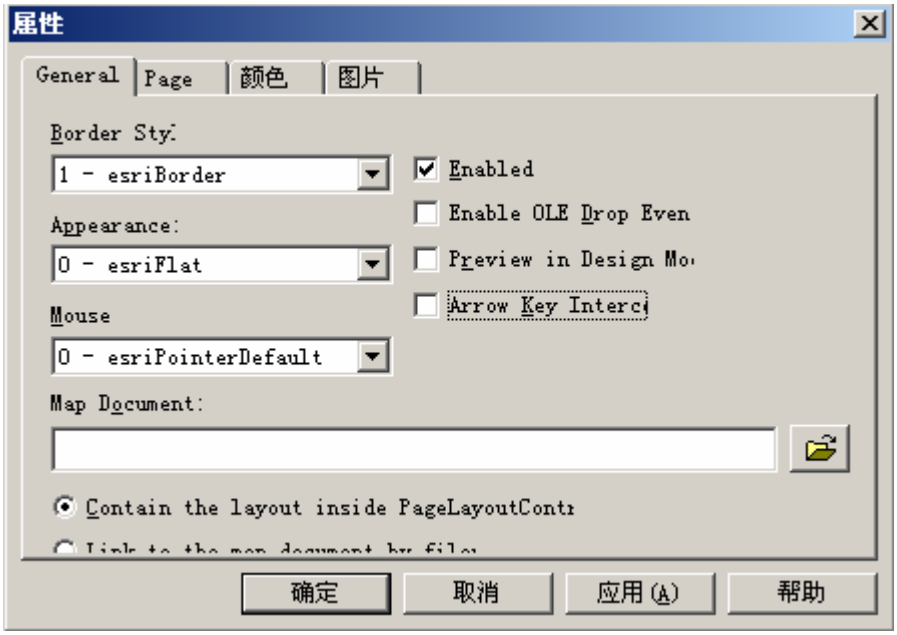


图 3-2 PageLayoutControl 控件属性对话框

3.2.1 PageLayoutControl 操作 MXD 文件

在 AxPageLayoutControl 控件中 MxDocument 不再是 PageLayout 对象的管理者，而仅仅是一个外部数据的传递者。这个控件对 MXD 文件的操作需要使用到 Carto 库中的 IMapDocument 接口定义的方法和属性，这些属性和方法可以读取地图文档文件（.Mxd、mxt、.pmf）和保存 mxd 文件。

IMapDocument 接口提供了多种属性和方法以便用户获取一个 MXD 文件的内容。如 Map 属性用于使用一个索引号返回文档中的一个 Map 对象；MapCount 则用于返回文档中包含的 Map 对象的数目；PageLayout 属性则可以返回一个页面视图 Pagelayout，这个对象包含了 Map 和 MapSurround 集合。

DocumentType 可以返回当前载入的地图文档的，如果是 MXD，则返回 0；文档为 PMF，则返回 1；图层文件为 2；未知类型为 3。

下面是在 PageLayout 控件中打开、保存一个 MXD 文件的过程：

下面的过程用于保存一个文档文件，在 PageLayout 控件上进行任何改变后，如果需要将这些改变保存起来，使用的同样是 IMapDocument 对象。

```
private void OpenMapDocument()
{
```

```
        //产生一个 IMapDocument对象
        IMapDocument pMapDocument;
        pMapDocument = new MapDocumentClass();

        try
        {
            //打开文件对话框选取MXD文件
            System.Windows.Forms.OpenFileDialog openFileDialog1;
            openFileDialog1 = new OpenFileDialog();
            openFileDialog1.Title = "Open Map Document";
            openFileDialog1.Filter = "Map Documents (*.mxd)|*.mxd";
            openFileDialog1.ShowDialog();
            string sFilePath = openFileDialog1.FileName;

            //将数据载入pMapDocument并与map控件联系起来
            pMapDocument.Open(sFilePath, "");

            //将 IMapDocument的数据传给控件
            axPageLayoutControl1.PageLayout = pMapDocument.PageLayout;
            axPageLayoutControl1.Refresh();
        }
        catch( Exception ex)
        {
            MessageBox.Show(ex.ToString());
        }

    }
```

3.2.2 PageLayout 与 MapControl 联动

在 ArcMap 程序中，数据视图和布局视图中的数据改变是实时互动的，那是因为它们本来就是在处理同一份数据。许多用户也希望在使用 ArcObjects 控件开发的独立程序中实现这个功能，当在 MapControl 中绘制一个点后，在 PageLayout 控件中能够实时出现这个图形元素。

由于这两个控件本身并不能共享一个文档文件，所以借鉴 ArcMap 中的经验去寻找一个 MapFrame 进行联动是不能够成立的，**用户只能够通过拷贝的方法来实时传递两个控件的内容**。

将 axMapControl1 中的 Map 拷贝到 axPageLayoutControl1 代码如下：

```
private void CopyAndOverwriteMap()
    {
        //Get IObjectCopy interface
        IObjectCopy objectCopy = new ObjectCopyClass();
        //Get IUnknown interface (map to copy)
        object toCopyMap = axMapControl1.Map;
        //Get IUnknown interface (copied map)
        object copiedMap = objectCopy.Copy(toCopyMap);
        //Get IUnknown interface (map to overwrite)
        object toOverwriteMap = axPageLayoutControl1.ActiveView.FocusMap;
        //Overwrite the PageLayoutControl's map
        objectCopy.Overwrite(copiedMap, ref toOverwriteMap);

    }
```

要实现 axPageLayoutControl1 中的数据与 axMapControl1 中的数据同步，需要在 axMapControl1 的以下事件中写入相应的代码：

```
        private void axMapControl1_OnMapReplaced(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMapReplacedEvent e)
        {
            CopyAndOverwriteMap();
        }

    private void axMapControl1_OnAfterScreenDraw(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnAfterScreenDrawEvent e)
        {
            //Get IActiveView interface
            IActiveView activeView = (IActiveView)
axPageLayoutControl1.ActiveView.FocusMap;
            //Get IDisplayTransformation interface
            IDisplayTransformation displayTransformation =
activeView.ScreenDisplay.DisplayTransformation;
            //Set the visible extent of the focus map
            displayTransformation.VisibleBounds = axMapControl1.Extent;
            //Refresh the focus map
            axPageLayoutControl1.ActiveView.Refresh();
            CopyAndOverwriteMap();
        }
```

```
        private void axMapControl1_OnViewRefreshed(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnViewRefreshedEvent e)
        {
            axTOCControl1.CtlUpdate ();
            CopyAndOverwriteMap();
        }
```

3.2.3 在 Pagelayout 里面绘制元素

这一节的主题是讨论使用样式选择对话框的方式来设置 PageLayoutControl 控件的各种属性。如调整控件的边框（border）、阴影（shadow）、背景（background）和设置 MapFrame 的 Mapgrid 属性的方法：

- **改变 MapFrame 的外观**

改变 MapFrame 的边框（border）：

```
        private void ChangeBorder()
        {
            IActiveView pActiveView;
            IGraphicsContainer pGraphicsContainer;
            IMapFrame pMapFrame;
            IMap pMap;
            pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
            pMap = pActiveView.FocusMap;
            pGraphicsContainer = pActiveView as IGraphicsContainer;
            pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
            IStyleSelector pStyleSelector;
            //新建一个边框选择器对象
            pStyleSelector = new BorderSelectorClass();
            bool m_bOK;
            m_bOK = pStyleSelector.DoModal(axPageLayoutControl1.hWnd);
            if (! m_bOK ) return;
            IBorder pBorder ;
            pBorder = pStyleSelector.GetStyle(0) as IBorder;   //得到一个 IBorder对象
            pMapFrame.Border = pBorder;
axPageLayoutControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground,null,null);
}
```

下面的代码是用于设置 MapFrame 的阴影（Shadow）属性：

```
        private void ChangeShadow()
        {
            IActiveView pActiveView ;
              IGraphicsContainer pGraphicsContainer;
              IMapFrame pMapFrame;
              IMap pMap;
              pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
              pMap = pActiveView.FocusMap;
              pGraphicsContainer = pActiveView as IGraphicsContainer;
              pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
              IStyleSelector pStyleSelector;
              pStyleSelector = new ShadowSelectorClass();
              //显示选择对象框对象给用户
              bool m_bOK;
              m_bOK = pStyleSelector.DoModal(axPageLayoutControl1.hWnd);
              //如果用户按cancel按钮则退出
              if (! m_bOK)    return;
              IShadow pShadow;
              //从选择器中得到一个IShadow对象
              pShadow = pStyleSelector.GetStyle(0) as IShadow;
              IFrameProperties pFrameProperties;
            pFrameProperties = pMapFrame as IFrameProperties;
              //设置框架的阴影属性
              pFrameProperties.Shadow = pShadow;
axPageLayoutControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground,null,null);
}
```

下面的代码是设置 MapFrame 的背景（BackGround）属性：

```
        private void ChangeBackgound()
        {
            IActiveView pActiveView ;
              IGraphicsContainer pGraphicsContainer;
              IMapFrame pMapFrame;
              IMap pMap;

              pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
              pMap = pActiveView.FocusMap;
              pGraphicsContainer = pActiveView as IGraphicsContainer;
```

```
            pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
            IStyleSelector pStyleSelector;
            pStyleSelector = new BackgroundSelectorClass();
            //显示选择对象框对象给用户
            bool m_bOK ;
            m_bOK = pStyleSelector.DoModal(axPageLayoutControl1.hWnd);
            //当用户按下(cancel时退出过程)
            if(! m_bOK ) return;
            IBackground pBackground;
            //获得一个背景对象
            pBackground = pStyleSelector.GetStyle(0) as IBackground;
            //设置背景属性
            pMapFrame.Background = pBackground;
            //刷新背景
axPageLayoutControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground,null,null);
}
```

下面的方法用于选择一个 MapGrid 进行设置，也是使用选择对话框的方式：

```
    private void MapGird()
    {
            IActiveView pActiveView ;
            IGraphicsContainer pGraphicsContainer;
            IMapFrame pMapFrame;
            IMap pMap;

            pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
            pMap = pActiveView.FocusMap;
            pGraphicsContainer = pActiveView as IGraphicsContainer;
            pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;

            IStyleSelector pStyleSelector;
          pStyleSelector = new MapGridSelectorClass();
            bool m_bOK ;
            m_bOK = pStyleSelector.DoModal(axPageLayoutControl1.hWnd);
            if(! m_bOK ) return;

            IMapGrid pMapGrid;
            pMapGrid = pStyleSelector.GetStyle(0) as IMapGrid;
```

```
            IMapGrids pMapGrids;
            pMapGrids = pMapFrame as IMapGrids;
            //先清除已经存在的MapGrid对象
            pMapGrids.ClearMapGrids();
            if (pMapGrid != null)
            {
                //将选择的MapGrid对象添加进MapGrids中
                pMapGrids.AddMapGrid(pMapGrid);
            }
    axPageLayoutControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground,null,null);
    }
```

上面的几个例子都是使用结构类似的方法来设置 MapFrame 的属性，通过样式选择对话框可以非常方便地设置各种控件的某些图形元素。

3.3 TOCControl 控件

3.3.1 TOCControl 控件概述

TOCControl要与一个“伙伴控件”或实现了IActiveView接口的对象协同工作。“伙伴控件”可以是MapControl、PageLayoutControl、ReaderControl、SceneControl或GlobeControl。“伙伴控件”可以在设计时通过TOCControl属性页设置(在支持属性页功能的开发环境中可以这样做)或在驻留TOCControl的容器被显示时用SetBuddyControl方法通过编程设置。TOCControl的每个“伙伴控件”都实现了ITOCBuddy接口。TOCControl用“伙伴控件”来显示其地图、图层和符号体系内容的一个交互树视图并保持其内容与“伙伴控件”同步。例如，如果TOCControl的“伙伴控件”是一个MapControl，而且从该MapControl中删除了一个图层，则该图层也会从TOCControl中删除。同样地，如果终端用户与TOCControl交互并取消了某个图层的Visibility复选框，则该图层在MapControl中不再可见。

TOCControl以树形结构显示其“伙伴控件”的地图、图层及符号的内容表视图。TOCControl通过ITOCBuddy接口来访问其伙伴控件。TOCControl管理图层的可见性和标签的编辑。

TOCControl 的主要接口有：

- ITOCControl : IDispatch

- ITOCControlEvents : IDispatch

3.3.1.1 ITOCControl

ITOCControl 接口是任何与 TOCControl 有关的任务的出发点，如设置控件的外观，设置伙伴控件，管理图层的可见性和标签的编辑。

ITOCControl 接口的成员：

All	Description
AboutBox	Displays a dialog of information about the TOCControl.
ActiveView	The ActiveView used to populate the TOCControl.
Appearance	The appearance of the TOCControl.
BorderStyle	The border style of the TOCControl.
Buddy	The object whose ActiveView is used to populate the TOCControl.
CustomProperty	A property to associate data with a control.
Enabled	Indicates whether the TOCControl can respond to user generated events.
HitTest	Returns the item in the TOCControl at the specified coordinates.
hWnd	Handle to the window associated with the TOCControl.
KeyIntercept	A property that specifies interception of key strokes that are normally handled by the container. When intercepted the OnKeyDown and OnKeyUp events will be called. This value can be a combined bit mask of esriKeyIntercept enum values.
LabelEdit	Label editing state.
LayerVisibilityEdit	Layer visibility editing state.
MouseIcon	Custom mouse icon used if MousePointer is 99.
MousePointer	The mouse pointer displayed over the TOCControl.
Object	A property that returns the underlying control.
SetActiveView	Sets the ActiveView used to populate the TOCControl.
SetBuddyControl	Sets a control to be a buddy of the toolbar, this control must support ITOCBuddy.
Update	Updates the contents of the TOCControl to match its ActiveView.

When querying interface to ITOCControl in Visual Basic .NET or Visual C# .NET the Object property or container specific code must be used. This is because .NET contains the real control inside a wrapper object known as an host.

```
ITOCControl pTOCControl;
pTOCControl = axTOCControl1.Object as ITOCControl;
 或 pTOCControl = axTOCControl1.GetOcx() as ITOCControl;
```

3.3.1.2 ITOCControlEvents

ITOCControlEvents 是一个事件接口，它定义了 TOCControl 能够处理的全部事件，如 OnMouseDown、OnMouseMove、OnMouseUp、OnDoubleClick、OnBeginLabelEdit、OnEndLabelEdit、OnKeyDown、OnKeyUp，这些事件用户在建构独立程序过程中经常用到。如 OnBeginLabelEdit、OnEndLabelEdit 分别为当 TOCControl 中的标签（label，可以是地图、图层或图例）开始编辑、结束编辑时触发，这两个事件参数 e 都有一个名为 *canEdit* 的成员参数，将这个参数设置为 true 或 false 来控制标签是否可编辑，下列代码说明了在这两个事件处理方法中如何使用这个参数。

```
private void axTOCControl1_OnBeginLabelEdit(object sender,
ESRI.ArcGIS.TOCControl.ITOCControlEvents_OnBeginLabelEditEvent e)
{
    IBasicMap map = null;
    ILayer layer = null;
    object other = null;
    object index = null;
    esriTOCControlItem item = esriTOCControlItem.esriTOCControlItemNone;
    //Determine what kind of item has been clicked on
    m_TOCControl.HitTest(e.x, e.y, ref item, ref map, ref layer, ref other, ref index);   //已在其它地方（如窗体的 Load 事件中）声明为 ITOCControl 类型的变量，并已实例化
    //Only layer items can have their labels edited
    if (item != esriTOCControlItem.esriTOCControlItemLayer)
    {
        e.canEdit = false;
    }
}


private void axTOCControl1_OnEndLabelEdit(object sender,
ESRI.ArcGIS.TOCControl.ITOCControlEvents_OnEndLabelEditEvent e)
{
    //Prevent empty labels
    if (e.newLabel.Trim() == "")
    {
        e.canEdit = false;
    }
}
```

3.3.2 应用开发实例

3.3.2.1 调整 TOCControl 控件中图层显示顺序

在 ArcMap 中，调整“Table of Contents”中图层显示顺序，只需要简单的鼠标拖曳就能完成，既简单又直观。但是，TOCControl 控件并没有实现此功能，需要开发人员自己编程实现。

在这个实例中，我们利用鼠标左键拖放实现了 TOCControl 控件中图层顺序的调整，将源图层拖放到目标图层的下方。在该实例中，主要利用了 TOCControl 控件的 OnMouseDown、OnMouseUp 两个事件和 HitTest()、Update()方法；OnMouseDown 事件方法的主要功能是获得源图层 pMoveLayer，OnMouseUp 事件方法的主要功能是：1）获得目标图层及其索引号；2)利用 IMap 提供的 MoveLayer(pMoveLayer,Toindex)将源图层 pMoveLayer 移到目标图层（Toindex）的下方；3）使用 ITOCControl 提供的 Update()方法更新 TOCControl 控件中显示的内容。

注意：事先要声明以下三个全局变量：

```
ITOCControl m_TOCControl;
ILayer pMoveLayer;
int Toindex;
```

然后在窗体的 Load 事件中对 m_TOCControl 进行实例化：

```
m_TOCControl = axTOCControl1.Object as ITOCControl;
```

这个实例的事件处理代码如下：

```
    private void axTOCControl1_OnMouseDown(object sender,
ESRI.ArcGIS.TOCControl.ITOCControlEvents_OnMouseDownEvent e)
        {
            if(e.button==1)
            {
                esriTOCControlItem item = esriTOCControlItem.esriTOCControlItemNone;
                IBasicMap map = null; ILayer layer = null;
                object other = null; object index = null;
                m_TOCControl.HitTest(e.x, e.y, ref item, ref map, ref layer, ref other, ref
index);
                if(item==esriTOCControlItem.esriTOCControlItemLayer)
                {
```

```
                    if(layer is IAnnotationSublayer)return;
                    else
                    {
                        pMoveLayer=layer;
                    }
                }
            }
        }
```

```
        private void axTOCControl1_OnMouseUp(object sender,
ESRI.ArcGIS.TOCControl.ITOCControlEvents_OnMouseUpEvent e)
        {
            if(e.button==1)
            {
                esriTOCControlItem item = esriTOCControlItem.esriTOCControlItemNone;
                IBasicMap map = null; ILayer layer = null;
                object other = null; object index = null;
                m_TOCControl.HitTest(e.x, e.y, ref item, ref map, ref layer, ref other, ref
index);
                IMap pMap = axMapControl1.ActiveView.FocusMap;
                if(item == esriTOCControlItem.esriTOCControlItemLayer||layer!=null)
                {
                    if(pMoveLayer!=layer)
                    {
                        ILayer pTempLayer;
                        for(int i=0;i<pMap.LayerCount;i++)
                        {
                            pTempLayer=pMap.get_Layer(i);
                            if(pTempLayer==layer)
                            {
                                Toindex=i;
                            }
                        }
                        pMap.MoveLayer(pMoveLayer,Toindex);
                        axMapControl1.ActiveView.Refresh();
                        m_TOCControl.Update();
                    }
```

```
            }
        }
    }
```

3.3.2.2 点击鼠标右键显示所在图层的属性表内容

在 ArcMap 中，“Table of Contents”中右键菜单功能非常丰厚，图 3-3 为图层的右键菜单，而 TOCControl 控件并没有提供右键菜单功能。要实现右键菜单功能，需要开发人员自己编程实现。

在这个开发实例中，我们实现了在 TOCControl 控件中点击鼠标右键弹出所在图层的属性表功能。图层的属性表在另外一个窗体（Form2）中显示，这个窗体中有一个 System.Windows.Forms.DataGrid 类型的对象 dataGrid1，为了在其它类中使用 dataGrid1 对象，在窗体 Form2 中定义了如下属性：

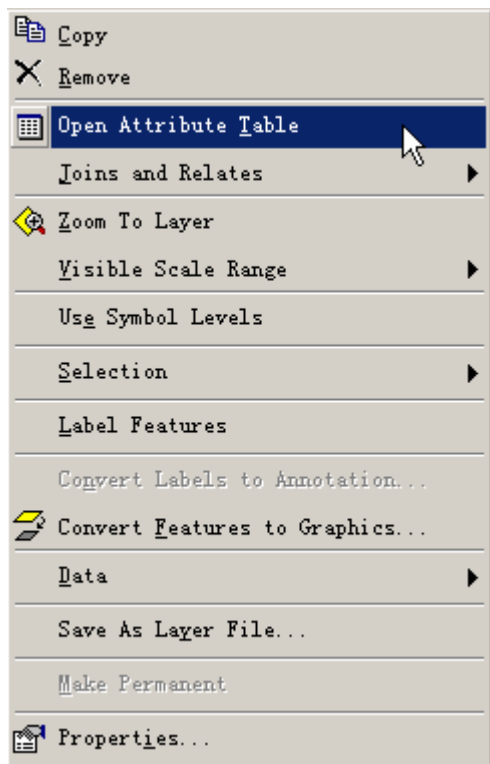


图 3-3　ArcMap 中 TOC 的图层右键菜单

```
public DataGrid GetDataGrid
    {
     get
         {
              return dataGrid1;
         }
     }
```

为了在 TOCControl 控件所在的窗体 Form1 中使用 Form2，首先需要在 Form1 的变量声明部分声明一个 Form2 类型的变量：

```
private Form2 frmOpenAttributeTable;
```

然后在 Form1 的 Load 事件中对其进行实例化：

```
frmOpenAttributeTable = new Form2 ();
```

其余主要代码如下：

```
        private void axTOCControl1_OnMouseDown(object sender,
ESRI.ArcGIS.TOCControl.ITOCControlEvents_OnMouseDownEvent e)
        {
            if (e.button ==2)
            {
              try
               {
                  if (this.axMapControl1.LayerCount > 0)
                   {
                      esriTOCControlItem mItem = new esriTOCControlItem ();
                      IBasicMap pMap = new MapClass();
                      ILayer pLayer = new FeatureLayerClass () ;
                      object pOther = new object ();
                      object pIndex = new object ();
                      this.axTOCControl1.HitTest(e.x, e.y, ref mItem, ref pMap, ref pLayer,
ref pOther, ref pIndex);
                      IGeoFeatureLayer pGeoFeatureLayer;
                      pGeoFeatureLayer = (IGeoFeatureLayer) pLayer;
                      OpenAttributeTable(pGeoFeatureLayer);
                      }
                }
                catch (Exception ex)
                {
                      MessageBox.Show("The following exception occurred: " +
ex.ToString ());
                }
            }

        }


        private void OpenAttributeTable(IGeoFeatureLayer pLayer)
        {
            ITableView2 pTableView3 ;
            ITable pTable;
            tagRECT initialExtent;
            initialExtent = new tagRECT();
```

```
            initialExtent.left = 0;
            initialExtent.right = 600;
            initialExtent.top = 0;
            initialExtent.bottom = 600;
            pTableView3 = new TableViewClass();
            pTable = (ITable) pLayer.FeatureClass;
            IFeatureSelection curSel = (IFeatureSelection) pLayer;
            ISelectionSet curSet = curSel.SelectionSet;
            IFeatureClass curClass = pLayer.FeatureClass;
            pTableView3.Table = pTable;
            pTableView3.SelectionSet = curSet;
            if (!(frmOpenAttributeTable.Created)) frmOpenAttributeTable = new Form2 ();
            DataGrid dataGrid;
            dataGrid = frmOpenAttributeTable.GetDataGrid;
            pTableView3.Show(dataGrid.Handle .ToInt32 (),ref initialExtent, true);
            frmOpenAttributeTable.Text = pLayer.Name + " -- Attribute Table";
            frmOpenAttributeTable.Show ();
            frmOpenAttributeTable.TopMost = true;
        }
```

3.4 ToolbarControl 及相关对象

3.4.1 概述

ToolbarControl 名称空间中包括以下 6 个对象及相关接口：ToolbarControl、ToolbarItem、ToolbarMenu、CommandPool、CustomizeDialog、MissingCommand. 这 6 个对象之间的结构关系如图 3-4 所示。

ToolbarControl 要与一个“伙伴控件”协同工作。“伙伴控件”可以是一个 MapControl、PagcLayoutControl、ReaderControl、SceneControl 或 GlobeControl。“伙伴控件”可以在设计时通过 ToolbarControl 属性页设置或在驻留 ToolbarControl 的容器被显示时用 SetBuddyControl 方法通过编程设置。ToolbarControl 可以驻留操作其“伙伴控件”命令、工具和菜单。

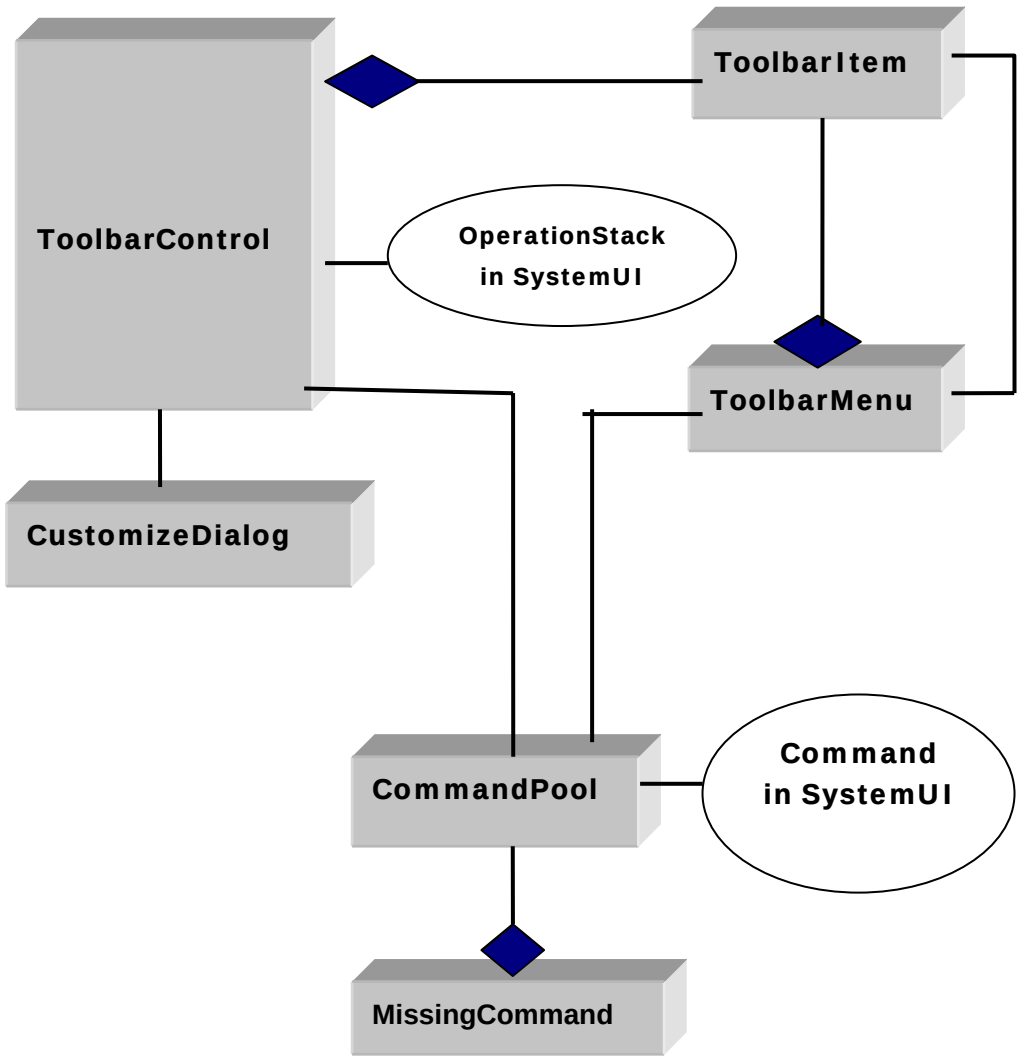


图 3-4 ToolbarControl 及相关对象间的关系

ToolbarControl 的每个“伙伴控件”都实现了 IToolbarBuddy 接口。这个接口用于设置“伙伴控件”的 CurrentTool 属性。例如，设想驻留了一个“页面放大”工具并有一个 PageLayoutControl 作为其“伙伴控件”的 ToolbarControl。当终端用户单击该 ToolbarControl 上的“页面放大”工具时，该“页面放大”工具就会成为 PageLayoutControl 的 CurrentTool。“页面放大”工具的实现会查询 ToolbarControl 以访问其“伙伴控件”(PageLayoutControl)并取回该 PageLayoutControl。然后该 PageLayoutControl 提供实现显示终端用户拉动鼠标所画的框并改变布局的范围。

ToolbarControl 不仅提供了部分用户界面，而且还提供了部分应用程序框架。ArcGIS Desktop 应用程序，如 ArcMap、ArcGlobe 和 ArcScene 等具有强大而灵活

的框架，包括诸如工具条、命令、菜单、泊靠窗口和状态条等用户界面组件。这种框架使终端用户可以通过改变位置、添加和删除这些用户界面组件来定制应用程序。许多开发环境提供简单对话框、窗体和多窗体界面(MDI)应用程序形式的框架。它们也提供普通的用户界面组件，如按钮、状态条和列表框。然而，仍需要为驻留命令的工具条和菜单提供大量的代码，特别是如果这些命令需要由终端用户定制的话。

ToolbarControl 及相关对象可以提供类似于 ArcGIS Desktop 应用程序框架那样的框架组件。用 ToolbarControl 建立应用程序时，开发人员可以使用这些框架组件的一部分或全部。

3.4.2 ToolbarControl 控件

ToolbarControl 使用钩子（hook）来联系命令对象和 MapControl 或 PageLayoutControl 控件，并提供属性、方法、事件用于：

- 管理控件的外观；
- 设置伙伴控件；
- 添加、删除命令项；
- 设置当前工具；
- 定制工具。

ToolbarControl 主要接口有：

- IToolbarControl
- IToolbarControlDefault
- IToolbarControlEvents (default)

3.4.2.1 IToolbarControl

IToolbarControl 接口是任何与 ToolbarControl 有关的任务的出发点，如设置控件的外观，设置伙伴控件，添加或去除命令、工具、菜单，定制 ToolbarControl 的内容。

IToolbarControl 接口提供的主要属性：Buddy、CommandPool、CurrentTool、Customize、CustomProperty、Enabled、Object、OperationStack 、ToolTips、TextAlignment、UpdateInterval 等。

IToolbarControl 接口提供的主要方法： AddItem、AddMenuItem、AddToolbarDef、Find、GetItem、GetItemRect、HitTest、MoveItem、Remove、RemoveAll、SetBuddyControl、Update 等。

3.4.2.2 IToolbarControlDefault

在绝大多数开发环境中，IToolbarControlDefault 接口自动曝露的缺省的 dispatch 接口。例如，在容器（窗体）中放置 ToolbarControl 控件，将产生一个名叫 axToolbarControl1 的对象，这个对象上直接可用的属性和方法对应于 IToolbarControlDefault 接口上的属性和方法，加上容器特有的属性和方法。

IToolbarControlDefault 接口的属性和方法，与 ToolbarControl 的最高编号主接口的属性、方法相同。例如，目前版本中，IToolbarControlDefault 等同于 IToolbarControl，但在未来的版本中，将变为 IToolbarControl2。在软件开发中，使用 IToolbarControlDefault 接口，能够保证总是访问最新版本的 ToolbarControl。

通过纯 dispatch 接口来调用方法会有一个小的性能开销，但可以通过 QI 一个具体接口来避免。在某些开发环境中，不可能在控件上直接 QI 到其它 COM 接口，因为控件被包含在一个包裹对象中，在执行 QI 之前，使用 Object 属性获得真正的控件。

3.4.2.3 IToolbarControlEvents

IToolbarControlEvents 是一个事件接口，它定义了 ToolbarControl 能够处理的全部事件，如 OnDoubleClick、OnItemClick、OnKeyDown、OnKeyUp、OnMouseDown、OnMouseMove、OnMouseUp。

3.4.2.4 ToolbarControl 上的命令

在 ToolbarControl 上可以驻留三类命令：

- 实现了响应单击事件的 ICommand 接口的单击命令（简单命令）。用户单击事件会导致对 ICommand::OnClick 方法的调用，并执行某种动作。通过改变 ICommand::Checked 属性的值，简单命令项的行为就象开关那样。单击命令是可以驻留在菜单中的唯一命令类型。

- 实现了 ICommand 接口和 ITool 接口、需要终端用户与“伙伴控件”的显示进行交互的工具。ToolbarControl 维护着一个 CurrentTool 属性。当终端用户单击 ToolbarControl 上的工具时，该工具就成为 CurrentTool，而前一个工具就处于非活动状态。ToolbarControl 会设置“伙伴控件”的 CurrentTool 属

性。当某个工具为 CurrentTool 时，该工具会从“伙伴控件”收到鼠标和键盘事件。

- 实现了 ICommand 接口和 IToolControl 接口的工具控件。这通常是用户界面组件，象 ToolbarControl 上的列表框或组合框。ToolbarControl 驻留了来自 IToolControl::hWnd 属性窗口句柄提供的一个小窗口。只能向 ToolbarControl 添加特定工具控件的一个例程。

可以用三种方法将命令添加到 ToolbarControl 中，第一种方法是指定唯一识别命令的一个 UID 对象(使用 GUID)，第二种方法指定一个 progID，第三种方法是给 AddItem 方法提供某个现有命令对象的一个例程。

```
//Adding a command by UID
UID uID = new UIDClass();
uID.Value = "esriControlCommands.ControlsMapFullExtentCommand";
axToolbarControl1.AddItem(uID,-1,-1,false,0,esriCommandStyles.esriCommandStyleIconOnly);

//Adding a command by ProgID
string progID = "esriControlCommands.ControlsMapFullExtentCommand";
axToolbarControl1.AddItem(progID,-1,-1,false,0,esriCommandStyles.esriCommandStyleIconOnly);

//Adding a command by ICommand
ICommand command = new ControlsMapFullExtentCommandClass();
axToolbarControl1.AddItem(command,-1,-1,false,0,esriCommandStyles.esriCommandStyleIconOnly);
```

3.4.2.5 更新命令

默认情况下，ToolbarControl 每半秒钟自动更新其自身一次，以确保驻留在 ToolbarControl 上的每个工具条命令项的外观与其底层命令的 Enabled、Bitmap 和 Caption 属性同步。改变 UpdateInterval 属性可以更改更新的频率。UpdateInterval 为 0 会停止任何自动发生的更新，开发人员必须编程调用 Update 方法以刷新每个工具条命令项的状态。在应用程序中首次调用 Update 方法时，ToolbarControl 会检查每个工具条命令项的底层命令的 ICommand::OnCreate 方法来检查是否已经被调用过。如果还没有调用过该方法，则该 ToolbarControl 作为“钩子(hook)”被自动传递给 ICommand::OnCreate 方法。

3.4.2.6 定制

ToolbarControl 有一个 Customize 属性，可以设置该属性以使 ToolbarControl 处于定制模式。这会改变 ToolbarControl 的行为，并允许终端用户重新安排、删除和添加命令项以及改变这些命令项的外观。

- 用鼠标左键选择 ToolbarControl 上的命令项，拖动选中的命令项到一个新的位置，或者拖动并将选中的命令项拖到 ToolbarControl 外以删除该命令项。
- 用鼠标右键选择 ToolbarControl 上的命令项并显示定制菜单。定制菜单可用于删除命令项或改变命令项的样式(位图、标题或两者)以及分组工具条命令项。

当 ToolbarControl 处于定制模式时，开发人员可以编程启动非模态定制对话框(CustomizeDialog)。定制对话框列出了所有的控件命令以及任何自定义命令、工具集和菜单。通过读取“ESRI 控件命令”、“ESRI 控件工具条”和“ESRI 控件菜单”组件类目的条目来完成这个工作。如果需要的话，开发人员可以改变定制对话框(CustomizeDialog)以使用其他组件类目。终端用户可以向 ToolbarControl 中添加这些命令、工具集和菜单，方法是拖动并将它们放到 ToolbarControl 上或双击这些命令、工具集和菜单。

定制对话框(CustomizeDialog)是非模态对话框，以允许用户与 ToolbarControl 进行交互。当用 StartDialog 方法弹出定制对话框时，该方法调用立即返回而定制对话框仍然在屏幕上打开。为了保持当定制对话框处于打开状态时对其的引用，将定制对话框存储为一个类级别的变量并侦听其 ICustomizeDialogEvents 是明智的做法。

3.4.2.7 操作栈(OperationStack)

ToolbarControl 有一个操作栈，用于管理“撤消(undo)”和“重做(redo)”功能。由每个工具条命令项的底层命令将操作添加到操作栈中，以便可以根据需要将操作前滚和后滚。例如，当删除一个图形元素时，可以通过移动该图形回到其原来的位置而撤消该操作。命令是否可以利用操作栈取决于该命令的实现。一般地，开发人员为应用程序创建一个单独的控件操作栈(ControlsOperationStack，默认情况下操作栈属性为 null)，并将其设置给每个 ToolbarControl。撤消和重做命令可以添加到进入了操作栈的 ToolbarControl 中。

3.4.3 ToolbarItem

ToolbarItem 就是驻留在 ToolbarControl 或工具条菜单上的单个 Command,

Tool, ToolControl 或 menu item 菜单。IToolbarItem 接口的属性决定工具条命令项的外观。例如，工具条命令项是否在其左侧有一条垂直线表示是否开始一个命令组(Group)，及命令项的样式是否有一个位图、标题或两者都有。Command 和 Menu 属性返回工具条命令项代表的实际命令或菜单。

ToolbarItem 是一个不可创建的对象。引用不可创建的对象必须通过其它对象获得。ToolbarItem 的主要接口有：IToolbarItem。

IToolbarItem 接口的成员：

All	Description
Command	The command used by the item.
CustomProperty	A property to associate data with a toolbar item.
Group	Indicates if a group separator exists to the left of the item on the ToolbarControl.
GroupSpacing	The width of the group separator in pixels if present.
Menu	If the item is a menu, this property is a ToolbarMenu object.
Style	The style of the item. Displays a combination of text caption and bitmap.
UID	The UID of the command used by the item, which may be empty.

3.4.4 ToolbarMenu 组件类

ToolbarControl 可以驻留下拉菜单项。工具条菜单(ToolbarMenu)表示单击命令项的一个垂直列表。用户必须选择工具条菜单上的一个命令项，或单击工具条菜单之外的地方使其消失。工具条菜单只能驻留命令项(不允许驻留工具或工具控件)。工具条菜单本身可以驻留在 ToolbarControl 上，作为“子菜单”驻留在另一个工具条菜单上，或者作为右键单击“弹出式菜单”出现。

每个 ToolbarControl 和工具条菜单都有一个命令池(CommandPool)，用于管理其使用的命令对象集。一般来说，开发人员不会与命令池(CommandPool)进行交互。当通过 ToolbarControl 属性页或编程将命令添加到 ToolbarControl 中时，该命令自动添加到命令池(CommandPool)中。命令对象要么作为唯一识别该命令的一个 UID 对象(使用 GUID)、要么作为命令对象的一个现有例程被添加到命令池(CommandPool)中。

如果命令对象的一个现有例程被添加，并且该命令没有一个唯一识别符，则命令池中可以有同一命令的多个例程存在。如果提供了一个 UID 对象，命令池可以确定该命令是否已经存在于命令池中，而且如果存在的话就可以重用该命令的前一个

例程。命令池通过追踪是否已经调用过命令的 OnCreate 方法来完成这个工作。如果已经调用过 OnCreate 方法，则将重用该命令并增加其使用次数(UsageCount)。

例如，如果向某个 ToolbarControl 中添加两次“放大”工具并提供 UID，则当 ToolbarControl 上的一个“放大”工具被选择并显示“按下(pressed)”时，另一个“放大”工具也会显示“按下(pressed)”状态，因为它们使用同一个命令对象。当应用程序包含多个 ToolbarControl 或工具条菜单时，开发人员应确保每个 ToolbarControl 和工具条菜单使用相同的命令池(CommandPool)，以保证在应用程序中只创建了命令的一个例程。

ToolbarMenu 的主要接口有：IToolbarMenu，该接口的成员如下：

All	Description
AddItem	Adds an command or menu item to the ToolbarMenu.
AddSubMenu	Adds a sub-menu to the ToolbarMenu.
Bitmap	The bitmap that is used as the icon on this ToolbarMenu.
Caption	The caption used by the ToolbarMenu.
CommandPool	The CommandPool used by the ToolbarMenu.
Count	The number of items on the ToolbarMenu.
Find	Returns the index of the first item containing the given command or menu. Returns -1 if it is not found.
GetItem	Returns the item at the specified index from the ToolbarMenu.
GetMenuDef	Returns the menu definition that is used for this ToolbarMenu.
Hook	The object that is passed as a hook to the OnCreate event of each item's command.
MoveItem	Moves an item from one index to another.
PopupMenu	Pops up the menu at the position specified.
Remove	Removes the item at the specified index from the ToolbarMenu.
RemoveAll	Removes all items from the ToolbarMenu.
SetHook	Sets the object that is passed as a hook to the OnCreate event of each item's command.

3.4.5 CustomizeDialog

CustomizeDialog包含一个无模态对话框，允许程序运行时将 commands, toolbar definitions (toolsets) 和 menu definitions 添加到 ToolbarControl 上，当 IToolbarControl::Customize 设置为true时。

CustomizeDialog的主要接口为：ICustomizeDialog和事件接口ICustomizeDialogEvents。

ICustomizeDialog接口的属性有： DialogTitle、IsDialogActive、DoubleClickDestination、CommandsCategory、MenusCategory、ToolbarsCategory、ShowAddFromFile。

ICustomizeDialog接口的方法有：StartDialog 、CloseDialog、SetDoubleClickDestination。

The CustomizeDialog is modeless to allow interaction with the ToolbarControl, which would otherwise be blocked if the dialog was modal. As such, the ICustomizeDialog::StartDialog method launches the CustomizeDialog on the screen and returns immediately, rather than waiting until the CustomizeDialog closes. Consequently, when the CoClass goes out of scope the CustomizeDialog will be closed. It is sensible practise to ensure the CustomizeDialog object remains alive while the container hosting the ToolbarControl is shown, by storing the CustomizeDialog object as a form (or class) level variable.

ICustomizeDialogEvents 接口成员：OnCloseDialog、OnStartDialog。

以下介绍一个开发实例：你自己开发的 GIS 应用程序，如何让最终用户在运行时实现命令、工具、菜单的定制功能？

首先声明一个 ICustomizeDialog 类型的全局变量 m_CustomizeDialog，并实例化，代码如下：

```
private ICustomizeDialog m_CustomizeDialog = new CustomizeDialogClass();
```

接着，声明如下两个事件变量：

```
private ICustomizeDialogEvents_OnStartDialogEventHandler startDialogE;
private ICustomizeDialogEvents_OnCloseDialogEventHandler closeDialogE;
```

创建一个名为 CreateCustomizeDialog 的方法，在该方法中添加代码，对两个事件变量和定制对话框进行有关设置：

```
        private void CreateCustomizeDialog()
        {
            // Set the customize dialog box events.
            startDialogE = new
ICustomizeDialogEvents_OnStartDialogEventHandler(OnStartDialog);
            ((ICustomizeDialogEvents_Event)m_CustomizeDialog).OnStartDialog +=
startDialogE;
            closeDialogE = new
```

```
ICustomizeDialogEvents_OnCloseDialogEventHandler(OnCloseDialog);
            ((ICustomizeDialogEvents_Event)m_CustomizeDialog).OnCloseDialog +=
closeDialogE;
            // Set the title.
            m_CustomizeDialog.DialogTitle = "Customize ToolbarControl Items";
            // Set the ToolbarControl that new items will be added to.
            m_CustomizeDialog.SetDoubleClickDestination(axToolbarControl1);
        }
```

在 Form1 的 Load 调用 CreateCustomizeDialog 方法：

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
        //……
        CreateCustomizeDialog();
        //……
      }
```

向窗体 Form1 中添加一个复选框并将其命名为 chkCustomize，标题为“Customize”。

在复选框 chkCustomize 的 CheckedChanged 事件处理方法中添加如下代码：

```
    private void chkCustomize_CheckedChanged(object sender, System.EventArgs e)
    {
        // Show or hide the Customize dialog box.
        if (chkCustomize.Checked == false)
        {
            m_CustomizeDialog.CloseDialog();
        }
        else
        {
            m_CustomizeDialog.StartDialog(axToolbarControl1.hWnd);
        }
    }
```

添加下面的 OnStartDialog 和 OnCloseDialog 事件处理器，在打开或关闭定制对话框时会触发这两个方法。

```
    private void OnStartDialog()
    {
        axToolbarControl1.Customize = true;
    }
```

```
private void OnCloseDialog()
{
    axToolbarControl1.Customize = false;
    chkCustomize.Checked = false;
}
```

按以上步骤就能实现 ToolbarControl 的定制。

3.5 ControlCommands

3.5.1 概述

ControlCommands 名称空间中提供了大量的命令、工具条及工具、工具控件和菜单，ArcObjects 程序员在自己的 GIS 应用开发中可以直接使用这些对象。使用这些对象有两种方式：

- 一是通过 ToolbarControl 控件使用
- 二是不通过 ToolbarControl 控件使用

开发人员也可以通过创建自己的自定义命令、工具和菜单来扩展 ArcGIS Engine 提供的命令集。HookHelper、GlobeHookHelper 和 SceneHookHelper 对象可用于简化这种开发。

3.5.2 ControlCommands with the ToolbarControl

通过 ToolbarControl 控件使用 Control Commands 有三条途径：

- 在程序设计阶段，通过 ToolbarControl 控件的属性页设置；
- 在程序代码中，使用 AddItem, AddMenuItem 或 AddToolbarDef 方法将要用到的 Control Commands 的实例对象添加到 ToolbarControl 控件中，命令可用使用 AddItem 或 AddSubMenu 方法添加到 ToolbarMenu 上；
- 最终用户可用通过 Customize 对话框进行设置（当 ToolbarControl 控件的 Customize 设置为 true 时有效）。

3.5.3 ControlCommands without the ToolbarControl

虽然建立带有ToolbarControl的应用程序可以快速提供类似于ArcGIS Desktop应用程序框架那样的框架组成部分，但有的时候应用程序可能并不需要ToolbarControl：

- ToolbarControl 的可视化外观可能不符合应用程序的需要。
- 不需要实现 ToolbarControl 的命令（Command）对象。
- 应用程序中已经有一个现有的应用程序框架。
- ToolbarControl 及其驻留的命令不易于跨多个“伙伴控件”使用。

在这些情况下，开发人员必须直接操作 MapControl、PageLayoutControl、SceneControl、GlobeControl 和 ReaderControl。应用程序需要其他用户界面组件，如命令按钮、状态条和列表框等，可能要由开发环境提供。

例如，将地图导航功能建立到 MapControl 应用程序中可以如下实现：

- 在 MapControl 的 OnMouseDown 事件中将 IMapControl2:: TrackRectangle 方法的结果“封装边界（Envelope）”设置给 IMapControl2:: Extent 属性，以创建“放大”功能。
- 将 IMapControl2:: FullExtent 属性的“封装边界（Envelope）”设置给 IMapControl2:: Extent 属性，以创建“全图显示”功能。代码可以放在开发环境提供的命令按钮的单击（Click）事件中。

作为选择，ArcObjects 提供的控件命令(ControlCommands)或者使用 HookHelper、GlobeHookHelper 或 SceneHookHelper 直接操作单个的 ArcObjects 控件。然而，开发人员要负责在适当的时候调用 ICommand::OnCreate 和 ICommand::OnClick 方法并读取 ICommand 接口上的属性以建立用户界面，方法如下：

通过编程创建命令的一个新例程，并将单个 ArcObjects 控件传递给 OnCreate 事件。例如，如果 3D 的“放大到全图（ZoomToFullExtent）”命令要操作 GlobeControl，则 GlobeControl 必须作为“钩于（hook）”传递给 OnCreate 方法。

开发人员可以使用脱离 ToolbarControl 的命令池（CommandPool）对象，以便管理应用程序所使用的命令。命令池（CommandPool）支持基于命令的“钩子（hook）”属性调用各个命令的 OnCreate 方法。

如果命令只实现了 ICommand 接口，开发人员可以在适当的时候调用 OnClick 方法以执行特定行动。如果命令为实现了 ICommand 接口和 ITool 接口的工具，开发人员必须将该工具设置为 ArcObjects 控件中的 CurrentTool。ArcObjects 控件会将任何键盘和鼠标事件传送给该工具。

例如：

```
ICommand cmd ;
cmd = new Symbology.UniqueValueRenderer();
cmd.OnCreate(axMapControl1.GetOcx());
if (cmd.Enabled) cmd.OnClick();

ICommand pCommand;
pCommand = new ControlsSelectFeaturesToolClass();
pCommand.OnCreate(axMapControl1.Object);
axMapControl1.CurrentTool = pCommand as ITool;
```

命令的 Enabled、Caption 和 Bitmap 属性可以被读写到开发环境提供的命令按钮的属性中，以建立应用程序的用户界面。

虽然这种建立应用程序的方法需要开发人员编写更多的程序代码，但从零开始建立应用程序的确具有更大的灵活性。

3.6 应用开发实例

3.6.1 概述

命令（Command）和工具（Tool）是 ToolbarControl 控件上最常用的两种对象。

在命令对象被驻留到 ToolbarControl 后就会立即调用 ICommand::OnCreate 方法。这个方法将一个句柄(handle)或“钩子(hook)”传递给该命令操作的应用程序。命令的实现一般都要测试以查看该“钩子(hook)”对象是否被支持(也就是说，命令要测试以查看该“钩子(hook)”是不是该命令可以操作的一个对象)。如果不支持该“钩子(hook)”，命令自动失效。如果支持该“钩子(hook)”，则命令存储该“钩子(hook)”以便以后使用。例如，如果“打开地图文档”命令要操作 MapControl 或 PageLayoutControl，而且它们作为“钩子(hook)”被传递给 OnCreate 方法，则该命令会存储该“钩子(hook)”以便以后使用。如果 ToolbarControl 作为钩子(hook)被传递给 OnCreate 事件，则命令一般会通过 Buddy 属性检查与该工具条协同使用的“伙伴控件”的类型。例如，如果驻留在 ToolbarControl 上的一个命令只能操作 ReaderControl，而该 ToolbarControl 的“伙伴”是一个 MapControl，则该命令应自动失效。

HookHelper、GlobeHookHelper 和 SceneHookHelper 可以帮助开发人员创建自定义命令以操作 ArcGIS 控件和 ArcGIS Desktop 应用程序。

- HookHelper 用于帮助开发人员创建能操作 MapControl、PageLayoutControl、ToolbarControl 和 ArcMap 桌面应用程序的自定义命令。
- SceneHookHelper 用于帮助开发人员创建能操作 SceneControl、ToolbarControl 和 ArcScene 桌面应用程序的自定义命令。
- GlobeHookHelper 用于帮助开发人员创建能操作 GlobeControl、ToolbarControl 和 ArcGlobe 桌面应用程序的自定义命令。

并不是由开发人员向命令的 OnCreate 方法中添加代码以确定传递给该命令的“钩子(hook)”的类型，而是由 helper 对象来处理这个任务。

helper 对象用于控制“钩子(hook)”并返回 ActiveView 对象、PageLayout 对象、Map 对象、Globe 对象和 Scene 对象(取决于 helper 对象的类型)，而不管被传递的是何种类型的“钩子(hook)”。

当终端用户单击工具条上的某个命令项时，ICommand::OnClick 方法被调用。取决于命令的类型，一般会使用“钩子(hook)”访问来自“伙伴控件”的所需对象来完成某些工作。

IHookHelper 是 HookHelper 的主接口，其定义的属性如下：

	All	Description
	ActiveView	Active view of hooked control or application.
	FocusMap	Focus map of hooked control or application.
	Hook	Hook object that should be the same as the hook object passed in ICommand::OnCreate.
	OperationStack	Operation stack of hooked control or application used for undo/redo operations.
	PageLayout	Page layout of hooked control or application.

一般情况下，所有的命令对象都要实现 ICommand 接口的所有成员，所有的工具对象都要实现 ICommand 和 ITool 两个接口的所有成员。为了简化自定义命令和工具的开发，ESRI 提供了两个抽象基类(BaseCommand and BaseTool)。这两个类

是抽象类，这意味着虽然它们包含一些实现代码，但自身不能直接被实例化而只能通过被其它类继承使用。这两个基础类都是在 ESRI.ArcGIS.Utility 程序集中定义的，属于 ESRI.ArcGIS.Utility.BaseClasses 命名空间。

这两个基类为 ICommand 和 ITool 每个成员提供了缺省的实现，这样简化了创建自定义命令和工具的过程。开发者不用为每个成员提供实现代码，只需重载自定义命令或工具所需要的成员。如 ICommand::OnCreate，该成员在初始的类中必须被重载。

在 VS.NET 开发环境，为 ArcGIS 应用程序创建命令和工具，推荐使用这些基类。可以从第一法则（实现 ICommand、ITool 所有成员）创建类似的 COM 类，然而用户很快就会发现基类（BaseClasses，ESRI.ArcGIS.Utility.BaseClasses）技术是一个更快速、更简单、更少出错的创建命令和工具的方法。

BaseClasses 有两种：BaseCommand 和 BaseTool，它们都有重载的构造函数，使得用户可以快速地通过构造函数参数设置命令和工具的许多属性，例如 Name 和 Category。

重载的 BaseCommand 构造函数有下面的签名：

```
public BaseCommand(
    System.Drawing.Bitmap bitmap,
    string caption,
    string category,
    int helpContextId,
    string helpFile,
    string message,
    string name,
    string toolTip

      );
```

重载的 BaseTool 构造函数有下面的签名:

```
public BaseTool(
    System.Drawing.Bitmap bitmap,
    string caption,
    string category,
    System.Windows.Forms.Cursor cursor,
    int helpContextId,
    string helpFile,
    string message,
    string name,
```

```
    string toolTip
    );
```

■ **继承基类**

当编写一个新类时，使用这些参数化的构造函数，例如，下面显示了称作 PanTool 的新类继承了 BaseTool 类。

```
public PanTool() : base ( null,"Pan", "My Custom
Tools",  System.Windows.Forms.Cursors.Cross, 0, "","Pans the map.", "PanTool",
"Pan" )
        {
            // ...

            }
```

■ **直接设置基类成员**

作为使用参数化构造函数的备选方案，可以直接设置基类的成员。

基础类暴露其内部成员变量给继承类，每个属性一个，这样可以在继承类中直接访问它们。例如，代替使用构造函数设置 Caption 属性或重载 Caption 函数，可以在继承类的构造函数中设置基类中声明的 m_caption 类成员变量。

```
public PanTool()
        {
            base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream("Namespa
ce.Pan.bmp"));
            base.m_cursor = System.Windows.Forms.Cursors.Cross;
            base.m_category = "My Custom Tools";
            base.m_caption = "Pan";
            base.m_message = "Pans the map.";
            base.m_name = "PanTool";
            base.m_toolTip = "Pan";
        }
```

■ **重载成员**

当创建继承一个基类的自定义命令和工具时，可能不仅需要重载几个成员。当重载类中的一个成员时，会执行用户提供给该成员实现的代码，而不会执行从基类继承而来的缺省实现代码。例如，在 BaseCommand 中 OnClick 方法没有任何的实现代码，同样的在缺省情况下 OnClick 不会做任何事。这样对于一个工具来说是合适的，但是可能对于一个命令来说不适合。

```
        public override void OnClick()
        {
            // Your OnClick
        }
```

自定义命令和工具，既可以为 ArcGIS 桌面应用程序服务，也可以为独立应用程序服务。如果是为前者服务，则需要按第一章介绍的步骤注册.NET 组件及其中的.NET 的类，其它的没有太大的区别。

在下面的开发实例中，我们将结合 HookHelper 和基类（BaseCommand 或 BaseTool）进行自定义命令、工具及菜单的开发。

3.6.2 自定义命令开发实例

自定义命令的开发继承基类 BaseCommand，一般只需重写 OnCreate()和 OnClick()方法，在类的构造函数中，对自定义命令进行初始化。

3.6.2.1 清除 ToolbarControl 上的当前工具

在 GIS 应用程序操作过程中，某个工具一旦使用，如果不使用下一个工具，它一直处于活动状态，这样总觉得不是很舒服。这个实例的功能就是清除 ToolbarControl 上的当前活动工具。

ClearCurrentTool.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.ToolbarControl;

namespace ControlsApplication
{
    /// <summary>
    /// ClearCurrentTool 的摘要说明。
    /// </summary>
    public class ClearCurrentTool:BaseCommand
    {
        IToolbarControl toolbarControl;
        public ClearCurrentTool()
        {
            base.m_caption = "ClearCurrentTool";
            base.m_message = "ClearCurrentTool";
```

```
            base.m_toolTip = "ClearCurrentTool";
            base.m_category = "ClearCurrentTool";
            base.m_enabled = true;
            base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            toolbarControl = hook as IToolbarControl;
        }

        public override void OnClick()
        {
            toolbarControl.CurrentTool = null;
        }
    }
}
```

3.6.2.2 清除当前地图中的选择要素

ClearFeatureSelection.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public sealed class ClearFeatureSelection : BaseCommand
    {
        private IHookHelper m_HookHelper = new HookHelperClass();

        public ClearFeatureSelection()
        {
            //Set the tool properties
            base.m_caption = "Clear Feature Selection";
```

```
            base.m_name = "Sample_Select(C#)_Clear Feature Selection";
            base.m_message = "Clear Current Feature Selection";
            base.m_toolTip = "Clear Feature Selection";
            base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"ClearSelection.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_HookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            //Clear selection
            m_HookHelper.FocusMap.ClearSelection();
            //Get the IActiveView of the FocusMap
            IActiveView pActiveView = (IActiveView) m_HookHelper.FocusMap;
            pActiveView.Refresh();
        }
    }
}
```

要在 ToolbarControl 控件上使用以上两个命令，需要在 Form1 的 Load 事件中加入以下代码：

```
    axToolbarControl1.AddItem( new
ClearCurrentTool(),0,-1,false,0,esriCommandStyles.esriCommandStyleIconOnly);
    axToolbarControl1.AddItem( new
ClearFeatureSelection(),0,-1,false,0,esriCommandStyles.esriCommandStyleIconOnly);
```

3.6.3 自定义工具开发实例

自定义工具的开发继承基类 BaseTool，在类的构造函数中，对自定义工具进行初始化。因为工具需要鼠标与地图类控件（MapControl、PageLayoutControl 等）进行交互以完成某个特定功能，除了要重写 OnCreate()方法外，还要根据具体情况重写其它方法，如 OnMouseDown()、OnMouseMove()、OnMouseUp()等。

下面开发的工具 ZoomOut 实现试图缩小功能。

ZoomOut.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public sealed class ZoomOut : BaseTool
    {
        IMapControl3 mapControl;
        private IHookHelper m_HookHelper = new HookHelperClass();

        public ZoomOut()
        {
            //Set command properties
            base.m_caption = "Zoom Out";
            base.m_toolTip = "Zoom Out";
            base.m_message = "Zoom Out on the Display";
            base.m_name = "Generic_ZoomOut";
            base.m_bitmap = new System.Drawing.Bitmap
(GetType().Assembly.GetManifestResourceStream(GetType(), "zoomout.bmp"));
            base.m_cursor = new System.Windows.Forms.Cursor
(GetType().Assembly.GetManifestResourceStream(GetType(), "ZOOMOUT.CUR"));
        }

        public override void OnCreate(object hook)
        {
            m_HookHelper.Hook = hook;
            if (hook is IToolbarControl)
            {
                IToolbarControl toolbarControl = (IToolbarControl) hook;
mapControl = (IMapControl3) toolbarControl.Buddy;
            }
        }
```

```
        public override bool Enabled
        {
            get
            {
                if(m_HookHelper.FocusMap == null) return false;
                return true;
            }
        }
        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            //Track a rectangle on the active view
            IEnvelope trackExtent = mapControl.TrackRectangle();
            if (trackExtent == null) return;
            //Get the current extent of the active view
            IEnvelope currentExtent = mapControl.Extent;
            IEnvelope newExtent = null;
            //If the tracked envelope is empty
            if (trackExtent.IsEmpty)
            {
                //Expand the current extent
                newExtent = currentExtent;
                newExtent.Expand(2.0, 2.0, true);
                //Center on the new extent
    newExtent.CenterAt(mapControl.ActiveView.ScreenDisplay.DisplayTransformation.ToMa
pPoint(X, Y));
            }
            else
            {
                //Create coordinates for the new extent
                double dWidth = currentExtent.Width * (currentExtent.Width /
trackExtent.Width);
                double dHeight = currentExtent.Height * (currentExtent.Height /
trackExtent.Height);
                double dXmin = currentExtent.XMin - ((trackExtent.XMin -
currentExtent.XMin) * (currentExtent.Width / trackExtent.Width));
                double dYmin = currentExtent.YMin - ((trackExtent.YMin -
currentExtent.YMin) * (currentExtent.Height / trackExtent.Height));
```

```
                    double dXmax = (currentExtent.XMin - ((trackExtent.XMin -
currentExtent.XMin) * (currentExtent.Width / trackExtent.Width))) + dWidth;
                    double dYmax = (currentExtent.YMin - ((trackExtent.YMin -
currentExtent.YMin) * (currentExtent.Height / trackExtent.Height))) + dHeight;

                    //Set the extent coordinates
                    newExtent = new EnvelopeClass();
                    newExtent.PutCoords(dXmin, dYmin, dXmax, dYmax);
                }
                //Set the new extent
                mapControl.Extent = newExtent;
            }
        }

    }
```

要在 ToolbarControl 控件上使用这个工具命令，需要在 Form1 的 Load 事件中加入以下代码：

```
axToolbarControl1.AddItem( new
ZoomOut(),0,-1,false,0,esriCommandStyles.esriCommandStyleIconOnly);
```

3.6.4 自定义菜单开发实例

在 ToolbarControl 上，不仅可以驻留命令和工具，还可以驻留菜单条。这个实例的功能是在 ToolbarControl 上以菜单条的方式驻留地图符号化的常用 6 个命令，实现地图符号化功能，如图 3-5 所示。

要在工具条上驻留菜单条，首先要实现 IMenuDef 接口；此外，在工具条上实现菜单功能，需要将菜单条中用到的所有命令类组织在一个项目文件（即一个 DLL 文件）中，并将该项目的“为 COM Interop 注册”设置为“true”，然后在其它项目中添加该项目的引用。在该实例中，我们将地图符号化的 6 个命令类和实现 IMenuDef 接口的类 SymbologyMenu 组织在一个项目 Symbology 中。

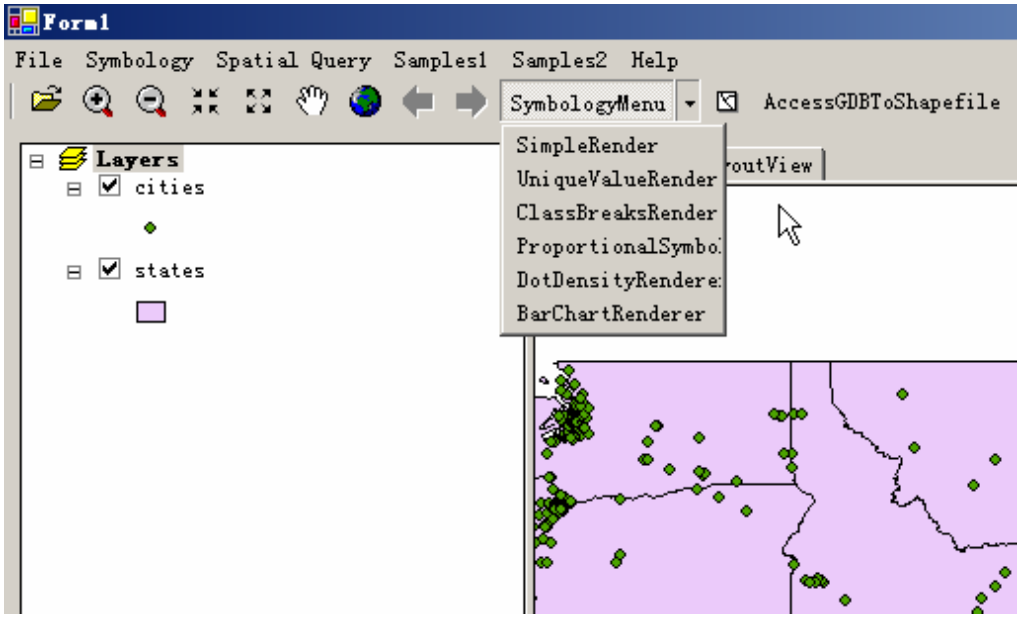


图 3-5 在 ToolbarControl 上实现菜单功能

SymbologyMenu.cs

```
using System.Runtime.InteropServices;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Utility.CATIDs;
using ESRI.ArcGIS.Framework;

namespace Symbology
{
   [Guid("c7d6da41-7e24-44fd-8b46-32cb8208405b")]
       public class SymbologyMenu : IMenuDef
       {
#region "Component Category Registration"
[ComRegisterFunction()]
static void Reg(string regKey)
{
            MxCommandBars.Register(regKey);
}

[ComUnregisterFunction()]
static void Unreg(string regKey)
```

```
{
            MxCommandBars.Unregister(regKey);
}
#endregion

        public SymbologyMenu()
            {
            }

            public void GetItemInfo(int pos, IItemDef itemDef)
            {
                //Commands for the menu
                switch (pos)
                {
                    case 0:
                        itemDef.ID = "Symbology.SimpleRender";
                        break;
                    case 1:
                        itemDef.ID = "Symbology.UniqueValueRenderer";
                        break;
                    case 2:
                        itemDef.ID = "Symbology.ClassBreaksRenderer";
                        break;
                    case 3:
                        itemDef.ID = "Symbology.ProportionalSymbol";
                        break;
                    case 4:
                        itemDef.ID = "Symbology.DotDensityRenderer";
                        break;
                    case 5:
                        itemDef.ID = "Symbology.BarChartRenderer";
                        break;
                }
            }

            public string Caption
            {
                get
```

```
            {
                return "SymbologyMenu";
            }
        }

        public int ItemCount
        {
            get
            {
                return 6;
            }
        }

        public string Name
        {
            get
            {
                return "SymbologyMenu";
            }
        }
    }

}
```

要在 ToolbarControl 控件上使用这个菜单，需要在 Form1 的 Load 事件中加入以下代码：

```
IMenuDef menuDef = new Symbology.SymbologyMenu();
//Add to the end of the Toolbar - it will be the 6th item
axToolbarControl1.AddItem(menuDef, -1, -1, false, -1,
esriCommandStyles.esriCommandStyleTextOnly);
```

如果要在 axMapControl1 中用这个菜单作为上下文菜单，首先需要在类（这里为 Form1）的变量声明部分声明一个 IToolbarMenu 类型的变量 m_ToolbarMenu：

```
private IToolbarMenu m_ToolbarMenu = new ToolbarMenuClass();
```

然后，需要在 Form1 的 Load 事件中加入以下代码，往 m_ToolbarMenu 中添加菜单项，并设置其 Hook：

```
m_ToolbarMenu.AddItem(menuDef, -1, -1, false,
esriCommandStyles.esriCommandStyleMenuBar);
```

```
        //Set the Toolbarmenu's hook
    m_ToolbarMenu.SetHook(axToolbarControl1);
```

最后，在 axMapControl1 的 OnMouseDown 事件中加入以下代码：

```
        if (e.button == 2)
        {
            //Popup the menu
            m_ToolbarMenu.PopupMenu(e.x, e.y, axMapControl1.hWnd);
        }
```

这样就实现了 axMapControl1 控件的右键菜单功能。

除了在 ArcObjects 控件上使用菜单外，我们还可以使用 VS.NET 提供的菜单功能，如图 3-6 所示。

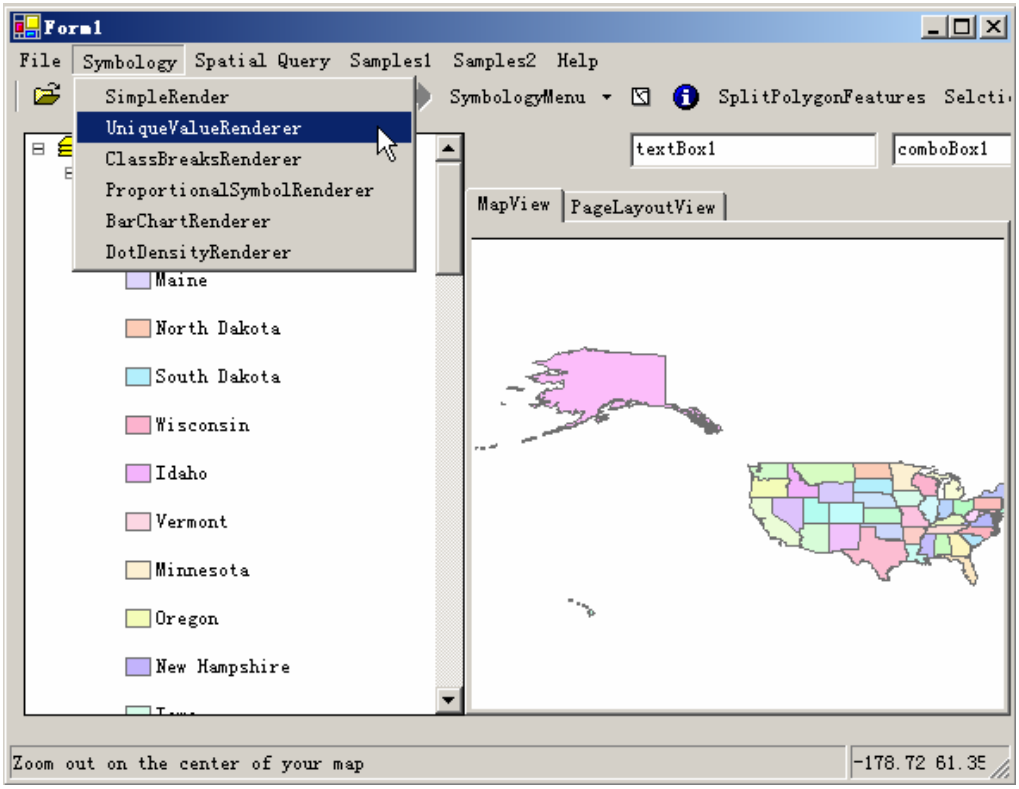


图 3-6 在应用程序中使用 VS.NET 提供的菜单功能

在上面的实例中，我们将实行地图符号化的命令以菜单的形式放置在 ToolbarControl 上，同样，可以将这些功能放置在 VS.NET 提供的菜单（System.Windows.Forms.MainMenu）上，例如，要实现上面的 UniqueValueRenderer 类的功能，只需添加一个菜单项如

mnuUniqueValueRenderer，然后在它的 Click 事件中编写如下代码即可。

```
private void mnuUniqueValueRenderer_Click(object sender, System.EventArgs e)
{
    ICommand cmd ;
    cmd = new Symbology.UniqueValueRenderer();
    cmd.OnCreate(axMapControl1.GetOcx());
    if (cmd.Enabled) cmd.OnClick();

}
```

本章主要介绍了 ArcObjects 提供的常用的几个控件以及多个开发实例，希望读者能够熟练掌握 VS.NET 环境下使用 HookHelper 和两个基类（BaseCommand 和 BaseTool）进行自定义命令、工具及菜单的开发。在后面的章节中我们还会看到更多的自定义命令和工具的开发实例。

4 几何形体对象--GEOMETRY

在 GIS 中，矢量数据模型是地理数据的最主要表现形式，GeoDatabase 中的每一条要素记录都有一个“Shape”字段用于保存它的一个或者多个几何形体对象，这些几何对象可以精确描述具有离散特征的要素在地球上的具体位置和形状，正是由于存在着这种精确的特征，才使得 ArcGIS 可以对这些要素进行不同的空间分析和运算，得出用户需要的结果。

对于创建一个要素或图形元素而言，几何形体对象也是它们的重要属性。由于 Geometry 属性存在，用户才能以图形的方式看到 GIS 要表示的信息。

本章主要介绍 ArcObjects 中丰富的几何形体对象以及它们之间的层次关系。由于 ArcObjects 中使用 Geometry 的地方非常广泛，因而熟练地掌握 Geometry 的知识对于开发者更好地了解 ArcObjects 是非常关键的。它是 GIS 系统的最基础部分，同时也是最重要的部分。

本章包含以下具体内容：

- Geometry 模型
- Point 和 Multipoint 对象
- 包络线对象
- 各种曲线对象
- Geometry 集合对象
- 空间参考

4.1 Geometry 模型

Geometry 是 ArcObjects 中使用最为广泛的对象集之一，用户在新建、删除、编辑和进行地理分析的时候，就是在处理一个包含几何形体的矢量对象；除了显示要素外，在空间选择、要素着色制作专题图、标注编辑等很多过程中也需要 Geometry 参与。

图 4-1 是 Geometry 模型的主要几何对象模型图。

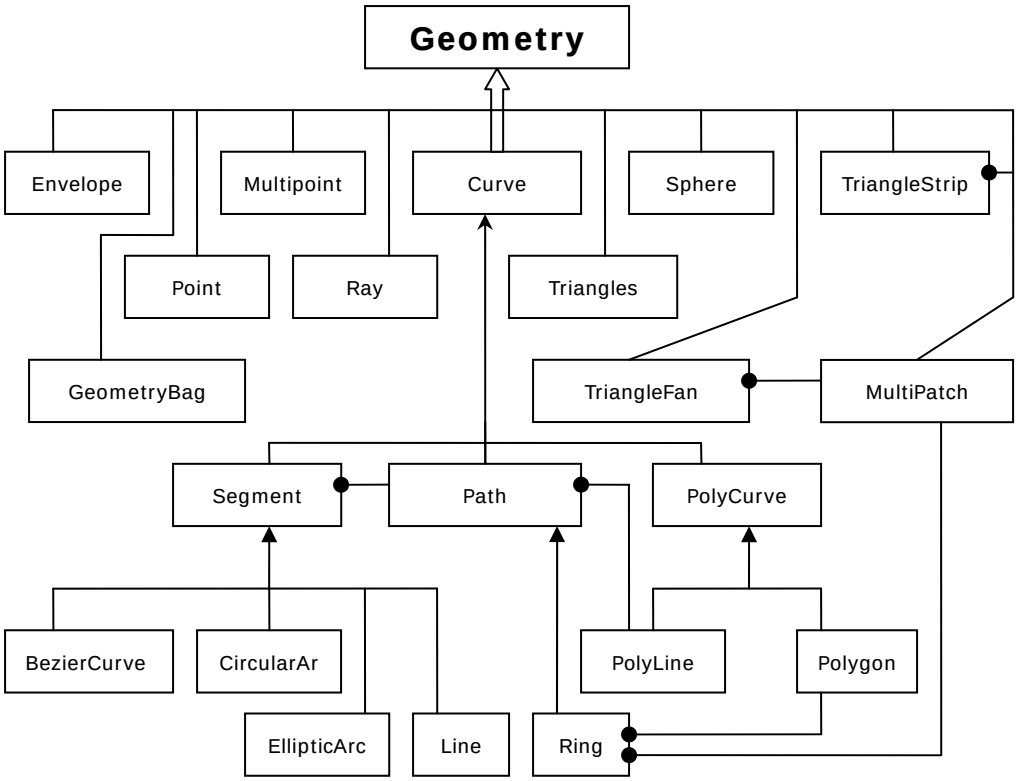


图 4-1 Geometry 模型中的主要几何对象

在 Geometry 模型中，几何形体对象被分为两个层次，一是构成要素形状的几何图形，另一种是组成这些形状的构件。前者被称为“高级几何对象”，具体包括以下类型：

- Point 对象是一个 0 维的几何图形，具有 X、Y 坐标值，以及可选的属性，如高程(Z 值)，测量值(M)和 ID 号，可以用于描述只需要精确定位的对象，如一个城市在全国的位置等。
- Multipoint 点集对象是无序点的群集，它用于表示具有相同属性设置的同一组点。如一家公司不同的营业场所可以使用点集来表示。
- Envelope 包络线是一个矩形，它用于表示要素的空间范围。它覆盖了几何对象的最小坐标和最大坐标，同时也记录了几何形体对象的 Z 值和 M 值的变化范围。所有的几何形体对象都拥有一个包络线，连包络线本身也不例外。
- Polyline 多义线是一个有序路径(Path)的集合，这些路径既可以是连续的，

也可以是离散的。这个对象可以用于表示具有线状特征的对象，如河流、公路和等高线等。用户可以使用单路径构成的多义线(Polyline)来表示简单线，如简单公路；使用具有多个路径的多义线来表示复杂线类型，如具有多个支流的河流等。

- Polygon 多边形是环(Ring)的集合，环是一种封闭的路径。Polygon 可以由一个或者多个环组成，甚至环内套环，形成岛环的情况，但是内外环之间不能重叠。对于一个给定的点而言，它总是在多边形“内部”、“外部”或者“边界上”。它通常用于描述具有面状特性的要素。

上面谈到的 Geometry 都是二维及以下的平面几何形体，实际上，由于 3 维 GIS 的需要，ArcObjects 中也提供了 2.5、3 维的几何形体对象。对 3 维 GIS 感兴趣的读者可以参看 ESRI 的相关开发帮助。

除了“高级几何对象”外，Geometry 模型中的其他对象都是构件几何对象，它们组合构成了高级几何对象，如 Segment 对象构成了 Path，而一个或者多个 Path 对象组成了 Polyline。

Geometry 是一个要素的基本组成部分，它确定了要素在地球上的位置。用户可以通过空间过滤器(SpatialFilter)对这些要素进行空间查询操作，如查询与某个几何对象相交的其他的几何形体等，以返回需要的几何对象。这个工作对于 GIS 用户而言使用非常普遍，比如在 ArcGIS 中查询距离某个居民区 1000 米内的超市等。

GIS 也可以对几何形体对象进行空间运算，如产生缓冲区、相交、合并、差集、距离等，有些空间运算还涉及拓扑、关系等方面的内容。这些操作一般都在两个 Geometry 对象之间进行(缓冲运算只需要一个几何形体对象)。

Geometry 模型中，某些几何对象可以组合产生新的几何形体，如 Polyline 由 Path 组成，而 Path 又可以由 Segment 组成。但是这并不意味着用户必须有“层次”地建立一个高级几何对象，即在新建一个 Polyline 的时候并不需要先生成一个 Line，然后多个 Line 生成 Path，再由 Path 生成 Polyline，实际上，Point 对象可以构成所有的几何形体。

在对象模型图中可以看到，ArcObjects 提供了 IGeometryCollection 接口，GeometryCollection 是具有相同类型的几何对象的集合。它可能是描述一条 Polyline 的路径的集合，或者是描述路径的片段(Segment)集合，甚至可以直接是组成 polyline 的点的集合。

下表列举了这种组合关系以及可以使用的集合接口：

几何对象	构件对象	用于创建和编辑此形状的接口
Path	Segment	ISegmentCollection
Ring	Segment	ISegmentCollection
MultiPatch	TriangleFan、TriangleStrip、Ring	IGeometryCollection
Multipoint	Point	IPointCollection、IGeometryCollection
Polyline	Path	IGeometryCollection
Polygon	Ring	IGeometryColleetion
TriangleFan	Point	IPointColleetion、IGeometryCollection
TriangleStrip	Point	IPointCollection、IGeometryCollection

Geometry 类是所有几何形体对象的父类，它是一个抽象类。IGeometry 接口定义了所有的几何对象都拥有的属性或方法，如 IGeometry::Dimension 可以查询几何形体对象的维度；IGeometry::Envelope 可以返回几何对象的包络线；IGeometry::GeometryType 则可以返回用于对象的几何类型。

在 Geometry 模型中，很多对象都是组件类，这意味着程序员可以直接创建一个几何形体对象。当开发者使用 new 关键字创建一个几何对象后，这个 Geometry 其实是空的，它需要添加具体的信息，如新建一个 Point 对象后，如果不设置它的 X、Y 坐标，这个点是无法明确显示的。

用户可以使用 IGeometry::IsEmpty 来查看这个对象是否为空。IGeometry::SetEmpty 方法可以将一个几何对象设置为空，而仅仅保留它的空间参考属性。当然，如果这个对象事先设置了空间参考的话。

IGeometry::Project 方法用于设置一个几何对象的空间参考属性，用户可以产生或者引用系统自己定义的空间参考，下面的代码是一个例子：

```
IGeometry pGeometry;
//设置pGeoemtry的空间参考
//pGeoemtry实例化省略
ISpatialReferenceFactory2 pFactory;
pFactory = new SpatialReferenceEnvironmentClass();
ISpatialReference pnewSR;
//使用系统中已经定义的墨卡托投影
pnewSR = pFactory.CreateProjectedCoordinateSystem(54004);
pGeometry.Project(pnewSR);
```

4.2 Point 和 Multipoint 对象

点(Point)代表了一个 0 维的、具有 X、Y 坐标的几何对象。点是没有形状的，它既可以用于表示描述点类型的要素，也可以使用在寻址、符号化和用于组成一个网络(Network)，更重要的是，任何几何对象都可以使用点来产生。

构成几何形状的顶点存在三种可以选择的属性，即 Z 值、M 值和 ID 值，Z 值在大多数情况下都可以用于表示一个点的 Z 坐标，即点的高程值，除此以外，还可以将 Z 值作为一个点的辅助值来使用，比如某点的降雨量、空气污染指数等，这些 Z 值的存在就可以构成一个三维的平面。M 值是所谓的 Measure 值，即度量值，它可以以一个路径对象的线性度量值，用于交通工程中一条公路的不同点的位置。ID 值是一个点的唯一标识值，它可以用于唯一确定一个点对象。

点集(Multipoint)是具有相同属性的点的集合，它在构成高级几何对象、几何对象动态模拟等方面都起了重要的作用，由于这两个对象都不是特别复杂，因此本节的重点放在如何构造这些点或者点集对象上。

4.2.1 Point 对象

- **IPoint 接口**

IPoint 定义了 Point 对象的属性和方法，ID 属性可以返回点对象的 ID 号，而 M、Z 值则返回点的测量号和高程值。使用 IPoint::X 和 IPoint::Y 用户可以获得一个点的 X 和 Y 的坐标值。

IPoint::PutCoords 用于设置一个点的 X、Y 坐标值，当用户 new 完成一个 Point 后，可以使用这个方法来建立一个实际的点对象。

IPoint::ConstrainAngle 和 IPoint::ConstrainDistance 方法可以使用参数改变已经存在点的位置。如 ConstrainAngle Projects the base Point to the nearest point on the line defined by an input anchor point and input angle. ConstrainAngle is used by the editor to force a newly created Point to be on the line between a fixed point and a specified angle.

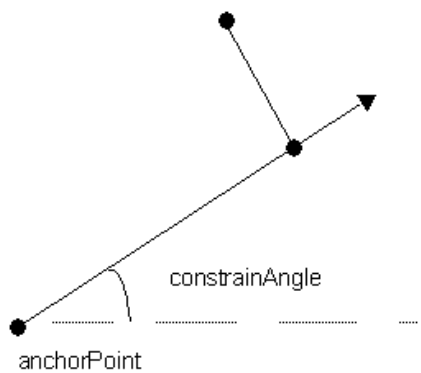


图 4-2 ConstrainAngle 定点法

- **IConstructPoint 接口**

ArcObjects 中 Geometry 模型的一个重要特点是它具有一套丰富的，利用已经存在的几何对象的距离、角度和空间关系，生成新的几何形体对象的方法。这些方法大都与坐标几何(COGO)相关。

使用这些方法常常涉及使用角度和偏转角。需要提醒读者注意的是，在 ArcObjects 中的编写程序时，角度使用的是弧度单位，而长度单位使用地图投影单位。

使用 IConstructPoint 接口，可以使用多达 10 种方法来创建所需要的点，下面简要介绍几种点的构造方法：

1. ConstructAlong 沿线创建法

沿线创建法是基于一个曲线(Curve)上起始点对象，通过给定距离、比例和扩展类型，沿着曲线类创建一个新点。如果距离比曲线的长度要长，那么点将沿着它的切线生成。如图 4-3 所示。

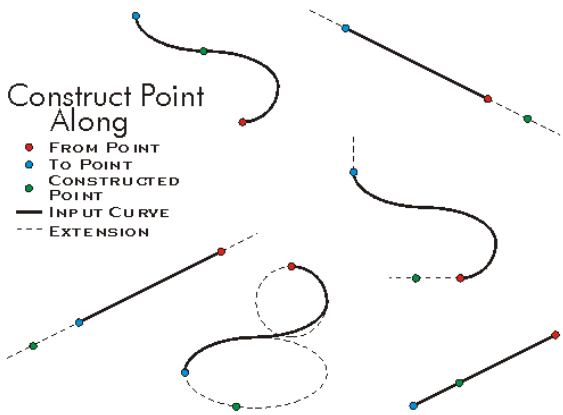


图 4-3　沿线创建法

```
private IPoint   PtConstructAlong(double dDist ,ICurve pCurve , esriSegmentExtension
            extension , bool aRatio)
 {
      IConstructPoint pCPoint;
      pCPoint = new PointClass();
      pCPoint.ConstructAlong(pCurve, extension, dDist, aRatio);
      return pCPoint as IPoint;
   }
```

需要提醒读者注意的是 Extension 参数，不同的延伸设置将会产生不同的点，其可能的取值为：

esriSegmentExtension Constants：

Constant	Value	Constant	Value
esriNoExtension	0	esriExtendTangents	5
esriExtendTangentAtFrom	1	esriExtendEmbedded	10
esriExtendEmbeddedAtFrom	2	esriExtendAtFrom	3
esriExtendTangentAtTo	4	esriExtendAtTo	12
esriExtendEmbeddedAtTo	8		

2. ConstructAngleBisector 角平分线创建法

这种方法使用三个点，即起始点(FromPoint)、通过点(Through Point)和终止点(ToPoint)。该方法通过平分三点形成的夹角，并设置一个距离在平分线寻找到一点。如果长度为负值，则沿着反方向放置点。如图 4-4 所示。

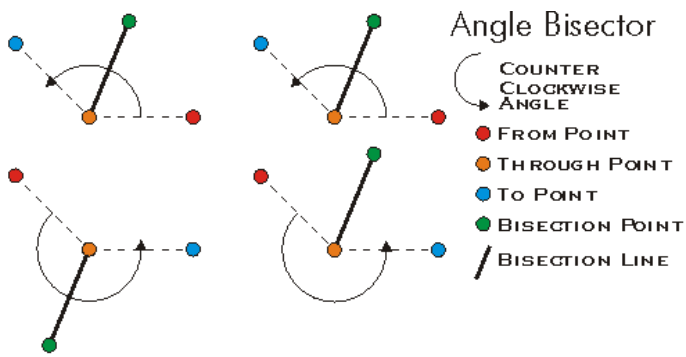


图 4-4 ConstructAngleBisector 创建法

3. ConstructAngleIntersection 构造角度交点

通过给定的两个点和两个角度，这个方法可以在两条射线的交点处产生一个点。两条射线由点和角度来确定。

如图 4-5 所示，通过 Point1、Point2 两点，使用 Angle1 和 Angle2，可以得到唯一的交点 ConstructedPoint。下面是产生这个点的函数：

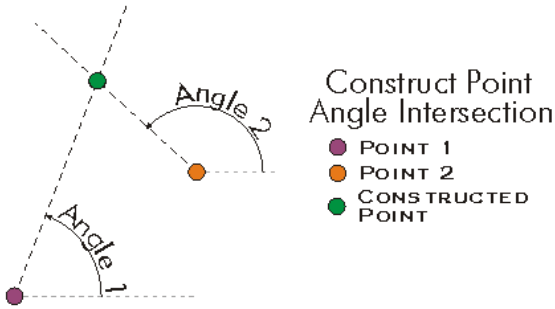


图 4-5 ConstructAngleIntersection 创建法

```
private IPoint ptConstructAngleIntersection(IPoint pPt1 , double dAngle1, IPoint pPt2 ,
double dangle2)
        {
            double PI = 3.14159265358979;
            double dAngleRadl;
            double dAngleRad2;
            IConstructPoint pCPoint;
            pCPoint = new PointClass();
            //角度转弧度
            dAngleRadl = dAngle1 * 2 * PI / 360;
            dAngleRad2 = dangle2 * 2 * PI / 360;
            pCPoint.ConstructAngleIntersection(pPt1, dAngleRadl, pPt2, dAngleRad2);
            return pCPoint as IPoint;
        }
```

4. ConstructAngleDistance 构造角度距离点

通过一个给定点和一个相对点的绝对角度和距离，构造器可以唯一确定一个新点。

下面是一个构造方法：

```
private IPoint PtConstructAngleDistance(IPoint pPoint , double dAngle, double dDist)
        {
            double PI = 3.14159265358979;
            double dAngleRad;
            IConstructPoint pCPoint;
            pCPoint = new PointClass();
            //角度转弧度
            dAngleRad = dAngle * 2 * PI / 360;
```

```
        pCPoint.ConstructAngleDistance(pPoint, dAngleRad, dDist);
        return pCPoint as IPoint;
    }
```

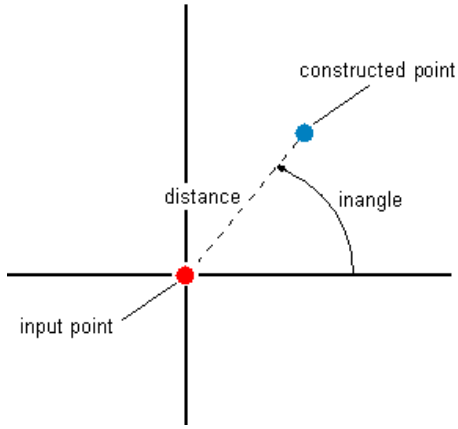


图4-6 ConstructAngleDistance定点法

5. ConstructDeflection 构造偏转角度点

给定一个基准线段，一个偏转角度和一个距离，该方法将沿着偏转角度的射线方向以定距离设置一个新点。图 4-7 是这种方法的原理。

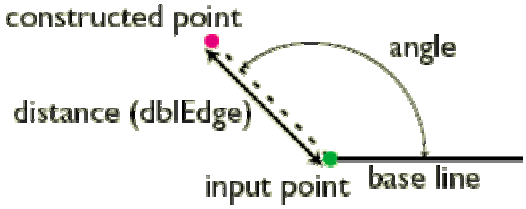


图 4 -7 构造偏转角度点

6. ConstructDeflectionIntersection 构造偏转角交点

这种方法使用一条线段作为基线，从该基线的起始点和终点分别测量偏转角度，引出两条射线，两条射线相交点为新点，这种方法其实就是测量中的前方交会定点法。OnRightSide 参数可以确定点在基线的方位。

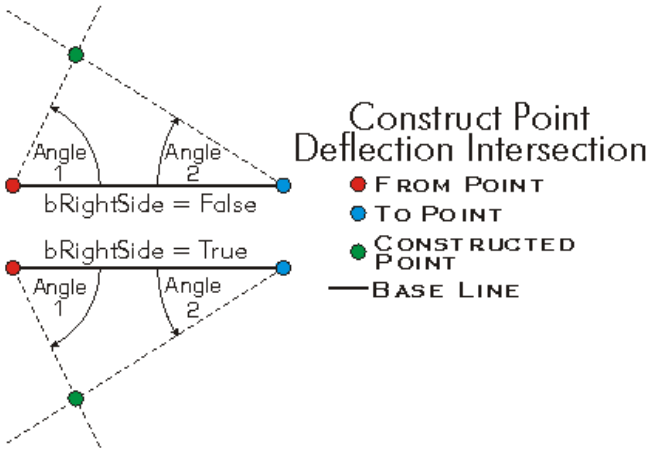


图 4-8 前方交会定点法

7. ConstructOffset 构造偏移点

给定一条路径，沿着曲线的距离或者比例，然后进行一段水平偏移，如果 Offset 值为正则该点向右水平偏移；如果 Offset 值为正，则向左水平偏移，如图 4-9 所示

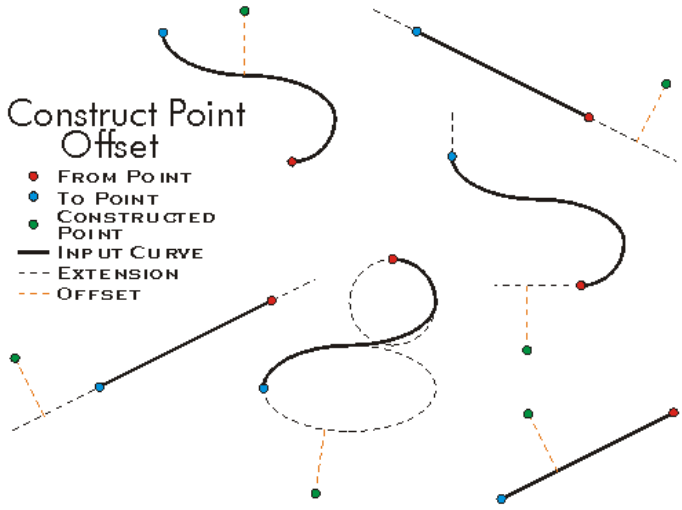


图 4-9 ConstructOffset 定点法

8. ConstructParallel 构造平行线上点

构造平行线上点方法是给定一条直线路径 Path，一个参考点 Point 及一段距离或比率，该构造器将在平行线上构造出一个新点。如图 4-10 所示。

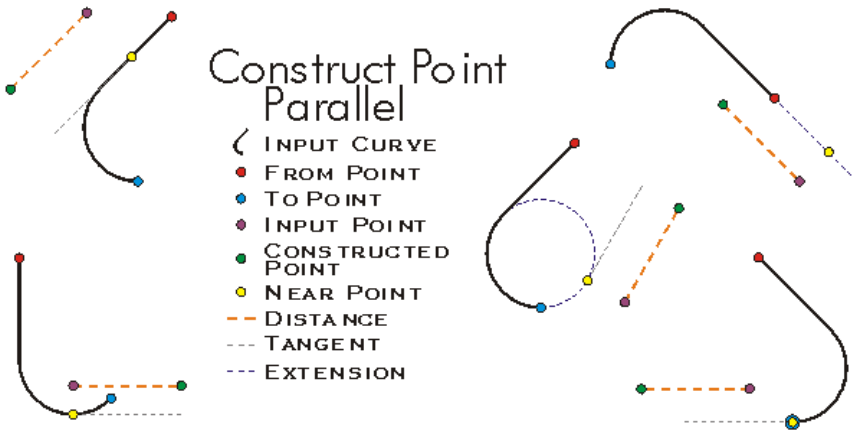


图 4-10 平行线定点法

9. ConstructPerpendicular 构造垂直线上点

ConstructPerpendicular constructs a Point a specified distance from the input point in the direction parallel to the normal (perpendicular to the tangent) of the nearest point on the extended input segment.

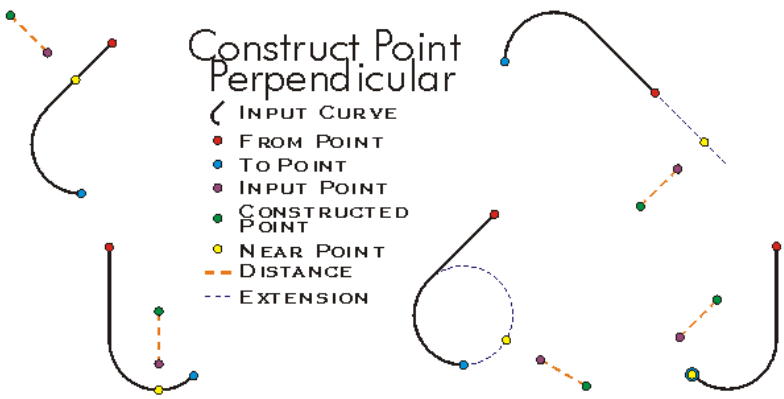


图 4-11 垂线定点法

10. ConstructThreePointResection 后方交会定点

这种定点方法就是我们测量学中后方交会定点法，即在待定点上对三个已知点进行观测，得到两个角度，即可确定待定点的位置。

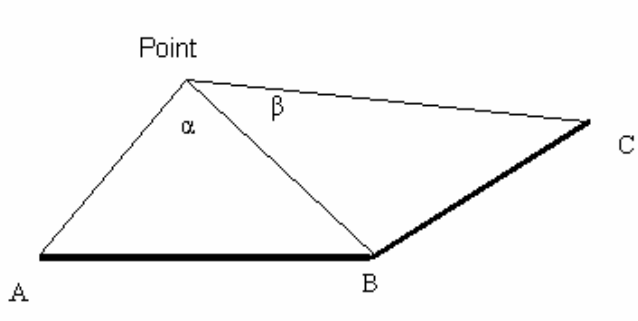


图 4-12 后方交会定点法

4.2.2 Multipoint 对象

Multipoint 对象持有一个点对象的集合，这些点的结构往往是相同的。这个对象常常作为其他运算的结构而出现，如选择一条曲线上的平分点等。

IMultipoint 是 Multipoint 对象主要实现的接口，它并没有方法、属性，但可以使用这个来判断某个对象是否是 Multipoint 对象。

Multipoint 对象可以使用 Add 方法来添加一个点到它的集合中去，这个过程需要使用 IPointCollection 接口来完成，如下面的代码：

```
IPointCollection pPointCollection;
pPointCollection = new MultipointClass();
object Missing = Type.Missing;
object val = 1;
pPointCollection.AddPoint(pPoint, ref Missing, ref val);
```

如同 Point 对象一样，Multipoint 对象也有众多的构造方法，这些构造方法都在 IConstructMultipoint 接口中定义，它们能够基于一个已经存在的几何对象来产生一个点集。下面介绍几种方法

1. ConstructArcPoints 构造圆弧点

这个构造器通过给定的一段圆弧，返回出该圆弧的终止点、起始点、圆心和切线的交点等四个点对象。如图 4-13 所示。

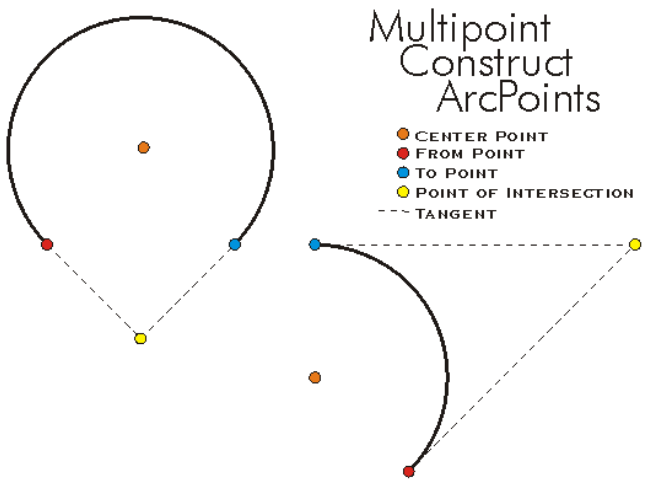


图 4-13　ConstructArcPoints 构造圆弧点

下面是一个演示的函数:

```
private IMultipoint ConstructArcPoints(ICurve pCurve)
{
    IConstructCircularArc pConstructCircularArc;
    IConstructMultipoint pConstructMultipoint ;
    IMultipoint pMultipoint;
    pConstructMultipoint = new MultipointClass();
    pConstructMultipoint.ConstructArcPoints(pCurve as ICircularArc);
    pMultipoint = pConstructMultipoint as IMultipoint;
    return pMultipoint;

}
```

这个函数返回的 pMultipoint 对象中有四个点，分别是输入的曲线的起始点、终止点、圆心和交线的切点，为了取出这些点可以使用 IPointCollection 接口来实现。

2.　ConstructDivideLength 构造等长度点

这个方法通过给定的一条曲线对象和已经定义的长度，返回所有处于这条曲线上的点对象，这些点包含在一个点集对象中。如图 4-14 所示。

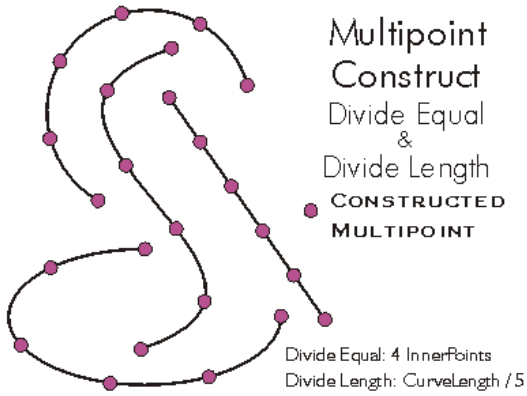


图 4-14　构造等长度点

3.　ConstructDivideEqueal 构造等分点

这个构造器根据输入的一条曲线和需要返回的点的数目来产生一个点集对象。

4.　ConstructIntersection 构造交点

Constructs a Multipoint containing the points of intersection and tangency between two input geometries. The input geometries can be extended by any of the esriSegmentExtension methods. Optionally, the construction method also returns the relative position and tangency of the intersection point along each curve.

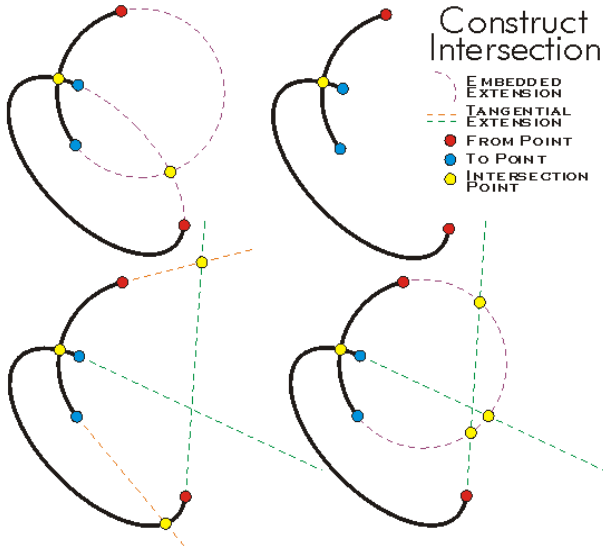


图 4-15 构造交点

5. ConstructTangent 构造切线点

ConstructTangent 这个方法可以用于产生一个点在曲线上的两个切点。如图 4-16 所示。

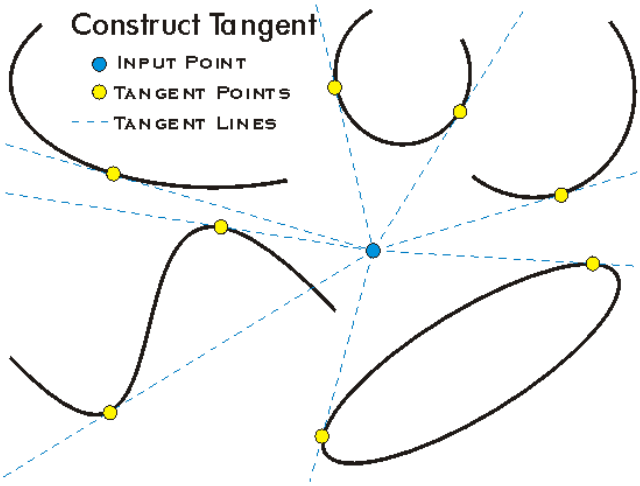
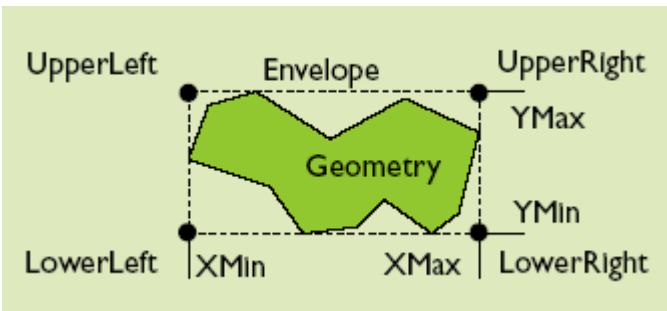


图 4-16 构造切线点

4.3 包络线 Envelope

Envelope (包络线对象)是一个矩形区域，如图 4 - 17 所示，它是作为任何一个几何形体的最小边框区域而存在的，每一个 Geometry 对象都拥有一个包络线对象，即使是包络线本身。除此以外，它也常常作为地图的视图或地理数据库的范围和用户交互操作的结果而被返回。

Envelope 通过它的最大和最小 X、Y 坐标来定义一个矩形形状，因此包络线对象相对于它的空间参考而言总是直角的。包络线对象也定义了最大和最小的 z 值、M 值，这两个值可以分别通过 IZAware 和 IMAware 接口来定义。



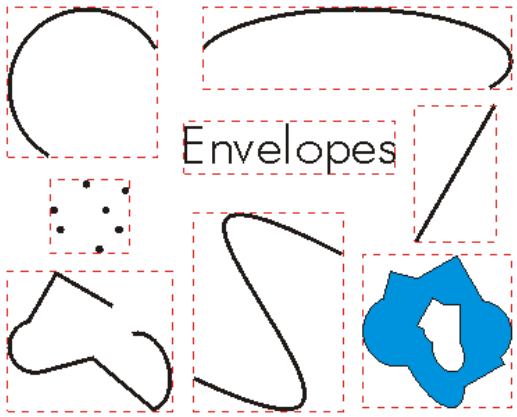


图 4 – 17 几何图形的包络线

IEnvelope 是包络线对象的主要接口，它定义了 XMax、XMin、YMax、YMin、Height 和 Width 属性用于获得和设置一个存在的包络线对象的空间坐标。

IEnvelope::Expand 方法可以用于按比例缩放包络线的范围，产生一个新的包络线对象，在缩放地图的视图操作中经常使用到这个方法，如：

pEnvelope.Expand (5, 0 , false)

这个方法拥有三个参数，前两者是 dx 和 dy ，第三个参数是是否设置比例，如果这个值为 false ，则新的包络线对象的坐标变为（ XMin = XMin - dx , YMin = YMin - dy , xMax = XMax + dx , YMax = YMax + dy ） ；如果它为 true ，则新的包络线坐标为（ XMin = XMin - dx * W idth / 2 , YMin = YMin - dy * Height / 2, XMax = XMax + dx * Width / 2 , YMax =YMax + dy * Height / 2 ）。

无论如何变化，这两个包络线对象的中心点都是在同一个位置。 ArcMap 软件的工具条上有两个缩放视图范围的命令，它们都是使用 IEnvelope::Expand 方法来达到改变视图显示范围目的的。

IEnvelope:: offset 是一个偏移方法，它通过添加一个 X 值和 Y 值（可以是正值也可以是负值)给 XMin、XMax、YMin 和 YMax，移动包络线本身，在这个移动的过程中包络线的面积保持不变。IEnvelope::CenterAt 方法则通过改变包络线的中心点来移动包络线对象。

一个包络线的四个角点都可以直接通过 UpperLeft、UpperRight、LowerLeft 和 LowerRight 得到。通过这四个属性可以来产生一个 Ring(环对象)，下面是这样一个代码例子，其中的 CreateLine 函数可以通过一个起始点和终止点产生一条新的 Line 对象：

```
private void    ConstructArcPoints(ICurve pCurve)
```

```
{
    IEnvelope pEnv;
    pEnv = pActiveView.Extent;
    ISegmentCollection pSegmentColl;
    pSegmentColl = new RingClass();
    pSegmentColl.AddSegment(CreateLine(pEnv.UpperLeft, pEnv.UpperRight));
    pSegmentColl.AddSegment(CreateLine(pEnv.UpperRight, pEnv.LowerRight));
    pSegmentColl.AddSegment(CreateLine(pEnv.LowerRight, pEnv.LowerLeft));
    pSegmentColl.AddSegment(CreateLine(pEnv.LowerLeft, pEnv.UpperLeft));
 }
```

IEnvelope 接口还为两个包络线提供了拓扑运算方法 Intersect 和 Union. IEnvelope::Intersect 是两个包络线的相交方法，IEnvelope::Union 方法则可以返回两个包络线对象的并集，这两个方法返回的对象都是包络线对象。

BaseEnvelope 对象和 InputEnvelope 对象相交，如果两个对象有相交的部分，则返回这个矩形区域，如图 4-18 第一个例子；如果两个包络线对象是分离的，则没有相交部分，见第二个例子；如果 BaseEnvelope 对象的面积比 InputEnvelope 对象小且在后者的内部，则返回的相交部分就是 baseEnvelope 对象，见图 4-18 第三个例子。

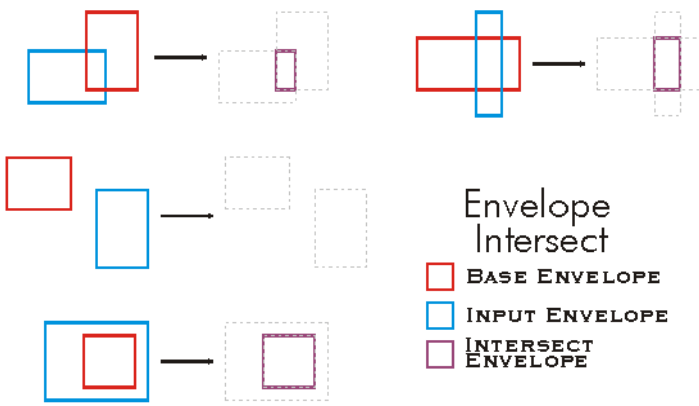


图 4-18 Envelope 的交集

图 4-19 是两个包络线对象的并集运算，第一个例子中两个包络线相交或分离，则返回的并集对象如上；如果 inputEnvelope 对象包含了 baseEnvelope 对象，则返回的是 inputEnvelope 对象，如图 4-19 第三个例子。

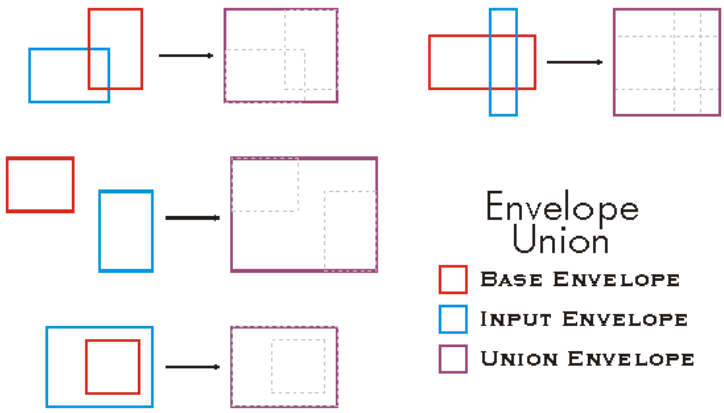


图 4-19 Envelope 的并集

IEnvelope::PutCoords 提供了一种构造包络线的方法，它通过传入 XMin、YMin、XMax 和 YMax 四个点对象而返回一个包络线；IEnvelope::QueryCoords 则可以返回一个包络线的四个值。

Envelope 对象也是支持 IEnvelope2 接口，这个接口提供了设置 Z 值和 M 值的属性，如 PutMCoords 和 PutZCoords 用于设置 M 值和 Z 值，而 QueryMCoords 和 QueryZCoords 方法可以查询包络线对象的 M 值和 Z 值。

4.4 曲线(Curve)

除去点、点集和包络线对象外，几乎其他所有的几何形体对象都可以看做是曲线（Curve），Curve 是具有一维视图或者二维边界形状的几何对象，如 Line 、Polyline 、Polygon、CircularArc、BezierCurve、EllipticArc 和 CircularArc 等都是曲线的一种，这些对象都实现了 ICurve 接口。

ICurve 接口提供了操作任何一种一维形状或二维边界形状的属性和方法，但是它并不能用于产生一个新的曲线对象。ICurve::Length 属性用于返回一个曲线对象的长度；ICurve::FromPnint 和 ICurve::ToPoint 两个属性可以用于获得和设置曲线的起始点和终止点，这两个点也可以通过 QueryFromPoint、 QueryToPoint 方法来得到。在处理 Polygon 和 Ring 两种曲线对象时，为了避免出现潜在的问题，推荐使用 FromPoint 和 ToPoint 属性来获得曲线的两个节点。

ICurve::Reverseorientation 方法可以改变一个曲线的节点次序，即改变一个曲线对象的起始点和终止点互相调换；IsClosed 属性则可以说明一个曲线的起始点和终止点是否在同一个位置，这个属性可以使用在除了 Polygon 外的任何一种几何形

体对象上。 Polygon 可能在它们的组成对象中存在非连接情况，如组成一个 Ring（环对象）的两个 Segment （片断对象)之间并没有首尾相连，而是分离的。IsClosed 属性并不能检查这种情况，为了保证 Polygon 或 Ring 中不出现这种情况，需要保证它们是简单几何对象。

当使用 ICurve 中的方法时，需要注意包含在这些方法中的 esriSegmentExtension 参数，这个参数可以用这些方法假设曲线好像是依据 esriSegmentExtension 的设置延伸了一样。如 ICurve::QueryPoint 方法可以依据特定的长度或比例来获得一个曲线对象上的某一点的位置，如图 4-20 所示。 To return the Point at two units along the curve, write code like this:

```
IPoint pOutPoint ;
pOutPoint = new PointClass();
pCurve.QueryPoint(esriSegmentExtension.esriNoExtension, 2, false, pOutPoint);
```

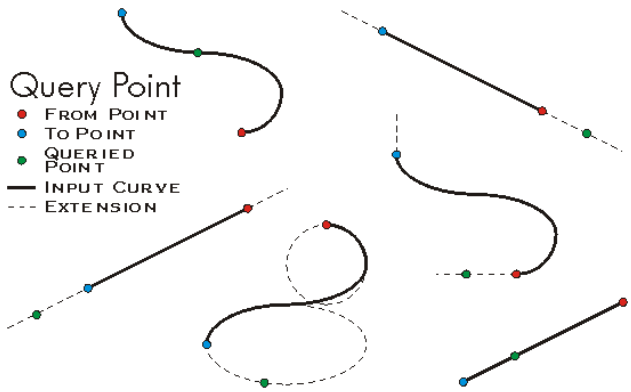


图 4-20　查询曲线上的点

如果 QueryPoint 使用的距离或比率大于曲线长，则 esriSegmentExtension 参数应该设置为 esriExtendTangentAtTo 常量，它可以在沿曲线终止点的切线上获得一个点，尽管这段切线是不存在的，如下面的代码段：

```
pCurve.QueryPoint(esriSegmentExtension.esriExtendTangentAtTo, 1.2, true, pOutPoint);
```

用户也可以从曲线的起始点开始获得一个新点，如下面的代码段

```
pCurve.QueryPoint(esriSegmentExtension.esriExtendTanggentAtFrom, -0.2, true,
pOutPoint)  //主要这里的距离或比率为负数
```

The embedded extensions extend a CircularArc or EllipticArc along the circle or ellipse defined by the Segment.

```
pCurve.QueryPoint(esriSegmentExtension. esriExtendEmbedded, 1.2, true, pOutPoint);
```

使用 ICurve::GetSubcurve 方法可以复制一个曲线对象的某个特定部分，如一条代表 10 千米长公路的曲线，程序员可以使用这个方法来获得 2-5 千米处的部分曲线，如下面的代码段：

```
ICurve pnewCurve;
pnewCurve = new PolylineClass();
pCurve.GetSubcurve(2, 5, true,out pnewCurve);
```

使用 ICurve::QueryTangent 和 ICurve::QueryNormal 方法可以得到基于曲线对象上某一点的曲线的切线和法线，这两个方法返回的是 ILine 对象，如下面的代码段：

```
pCurve.QueryTangent(esriSegmentExtension.esriNoExtension, 0, true, 1, pLine)
pCurve.QueryNormal(esriSegmentExtension.esriNoExtension, 0, true, 1, pLine)
```

4.4.1 Segment 对象

Segment 是由一个起始点、一个终止点以及定义两点之间的曲线的函数组成的一维几何形体对象，它是一条单一的曲线对象。Segment 对象是一个抽象类，它可能是线性的，如线段(Line)；也可能是非线性的，如圆弧(CircularArc)、椭圆弧(EllipticArc)和贝塞尔曲线(BezierCurve)等都是一种 Segment 对象，如图 4-21 所示。

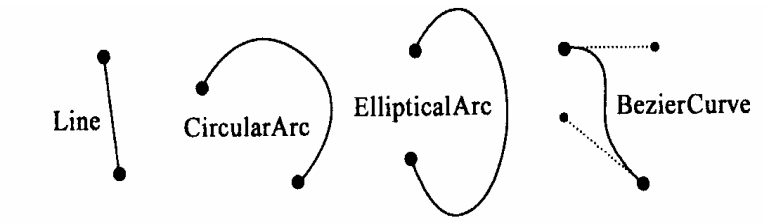


图 4-21　Segment 对象

这些 Segment 对象可以独立存在，也可以用于构成其他的几何形体对象，如 Path、Ring 或者 Polycurve 等。一个 Segment 对象可能有 z 坐标值、m 测量值或 ID 值。

Segment 也是 Curve 的一个子类，它从 Curve 类继承了最基本的属性和方法，如 FromPoint 和 ToPoint 确定了所有曲线的起始点和终止点。

ISegment 是 Segment 对象的主要接口，它提供了两个方法来将一个 Segment 对象“分割”为两个或多个 Segment，但是原始的 Segment 对象并没有被破坏。**ISegment::SplitAtDistance** 方法需要传入一个给定的长度值或比率值，这个长度值用于在 Segment 对象的起始点和终止点之间的路径上确定一个分割点，然后将 Segment 对象分割为两个 Segment，可以通过这个方法的第三个和第四个参数得到

这两个新的 Segment。**ISegment:: SplitDivideLength** 可以将一个 Segment 对象分割为不限数目的新 Segment，这些新产生的 Segment 是一个 Segment 数组，而这个方法将返回一个指向数组中第一个 Segment 元素的指针，下面是使用这个方法将一个 Segment 对象分割为 5 等份的新 Segment 的代码段，之所以分为 5 个等份是因为传入了一个比率值 0.2，返回的 Segment 数组的下标将从 0 到 4，如下：

```
ISegment[] pSegments = new LineClass[4];
int iNumSegs = 0;
ISegmentCollection pnewLine;
pnewLine = new PolylineClass ();
object Missing = Type.Missing;
object val = 1;
pInSegment.SplitDivideLength(0, 0.2, true, ref iNumSegs, ref pSegments[0]);
int i ;
for (i = 0;i <= iNumSegs - 1;i++)
 {
    pnewLine.AddSegment(pSegments[i], ref Missing, ref val);
  }
```

Segment 对象还支持 ISegmentM、ISegmentZ 和 ISegmentID 三个接口，通过它们可以分别获得和设置 Segment 对象的 M 值、Z 值和 ID 号。

更复杂的几何形体对象，如 Ring、Path、Polyline 和 Polygon 等，都可以由 Segment 对象集合来创建，其中的 Ring 和 Path 类支持 ISegmentCollection 接口，而 Polyline 和 Polygon 则支持 IGeometryCollection，后者是一个更一般的接口。通过 ISegmentCollection 接口定义的 AddSegment、RemoveSegment、AddSegmentCollection 和 SegmentCount 等方法可以将一个 Segment 集合变成复杂度更高的集合形体对象。

下面将逐一介绍四种类型的 Segment 对象。

4.4.1.1 CircularArc 对象

CircularArc 对象是一个圆弧，它是圆的一部分，如果使用 CircularArc 来表示一个整圆，则它的 CentralAngle 为∏，且其起始点和终止点的位置是一个点。在几何对象中，CircularArc 是 EllipticArc 的特殊情况。

一个 CircularArc 对象的主要属性如图 4-22 所示。

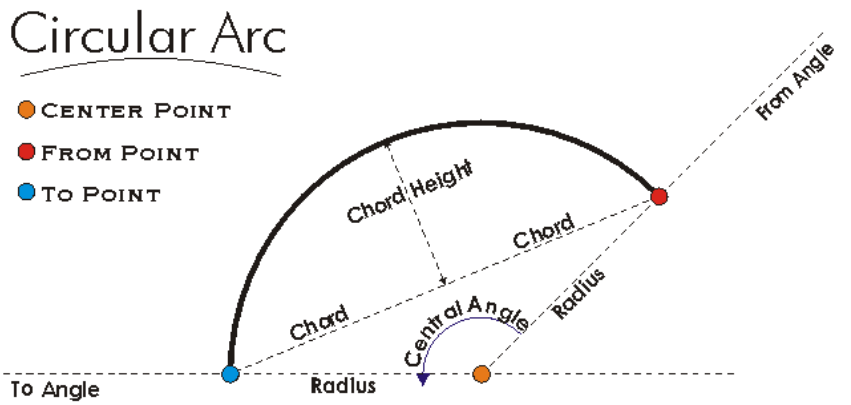
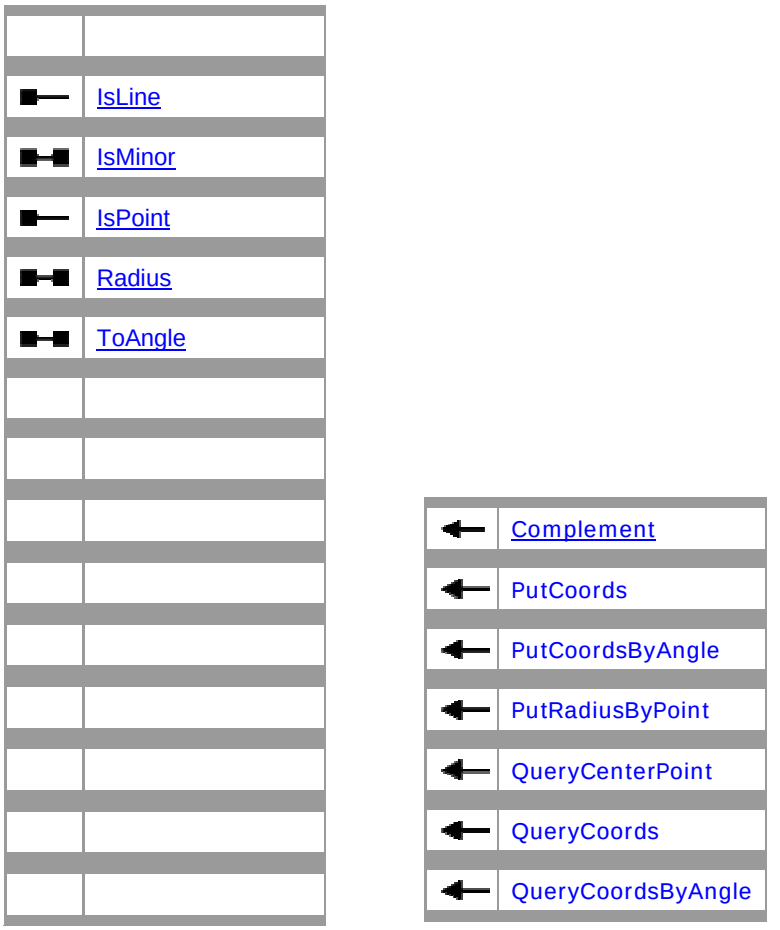


图 4 -22　CircularArc 主要属性图示

ICircularArc 是 CircularArc 的主要接口。通过 ICircularArc 接口，可以获得它的 FromAngle、ToAngle、CentralAngle、CenterPoint、ChordHeight 和 Radius 属性。除了 CenterPoint 外，这些属性都是可读写的，通过改变一个已经存在的 CircularArc 对象的角度、半径等参数，可以改变这个圆弧对象的形状。

	CenterPoint
	CentralAngle
	ChordHeight
	FromAngle
	IsCounterClockwise



ICircularArc 中定义的属性、方法

ICircularArc 接口也有几个属性用于检查圆弧对象的特征.如：

- IsLine——如果为真，则圆弧的 Radius 为无限大
- IsPoint——如果为真，则 Radius 为 0
- lsCounterClockwise——如果为真，则 CentralAngle 为正值
- IsMinor——如果为真，则 CentralAngle 小于半圆

对于一个存在的 CircularArc 对象，可以使用 Complement 方法来将它本身进行封闭，即产生一个圆对象，圆弧对象的形状是新返回的圆对象的一部分。

由于 CircularArc 是一个组件类，ICircularArc 接口除了直接使用 FromAngle、ToAngle、Radius 和 IsCounterClockwise 属性设置一个圆弧的属性外，还可以使用

PutCoords、PutCoordsByAngle 等方法来设置，在 ArcObjects 中推荐使用 PutCoords 方法，因为它可以最精确地确定一个圆弧的位置和形状。

IConstructCircularArc 接口提供了数目高达 35 种的构造器方法来产生一个 CircularArc 对象，这些方法通常都取决于其他已经存在的集合对象。下面将具体介绍几种常用的构造方法：

1. ConstructCircle 构造器

ConstructCircle 构造器是产生一个圆对象的最简单的方法，通过传入 CenterPoint(圆心)和 Radius(半径)两个属性，可以确定一个唯一的圆对象。下面是一个构造的代码段，使用一条线段的起始点作为圆心，其长度作为半径而构造：

```
IConstructCircularArc pCircularArc;
pCircularArc = new CircularArcClass();
pCircularArc.ConstructCircle(pLine.FromPoint, pLine.Length, true);
```

2. ConstructArcDistance 构造器

ConstructArcDistance 构造器通过传入圆心点、起始点和圆弧长度来产生一个新的圆弧对象，当然，这个长度不能超过整个 2 * ∏ * radius。如下面的代码段，其中第三个参数是确定圆弧的方向，True 为顺时针方向。

```
pCircularArc.ConstructArcDistance(pCenterPoint, pStartPoint, true, 150)
```

3. ConstructChordDistance

ConstructChordDistance 构造器基于一个起始点、圆弧的弦长度、圆弧的方向和中心点来产生一个新的圆弧对象。 所谓的圆弧的弦(Chord)是指圆弧对象的起始点和终止点之间的线段。

```
public void ConstructChordDistance (IPoint Center, IPoint from, bool isCCW, double chordDistance);
```

4. ConstructEndPointsChordHeight

ConstructEndPointsChordHeight 构造器是使用一个起始点、一个终止点和一个弦的中点高度，按照顺时针或逆时针的方向产生一个圆弧对象。

5. ConstructFilletPoint

ConstructFilletPoint 构造器可以产生两条线段或圆弧的内切弧，这个方法要求传入两个 Segment 对象以及内切弧在两个 Segment 上的点，其原理如图 4-23 所示。

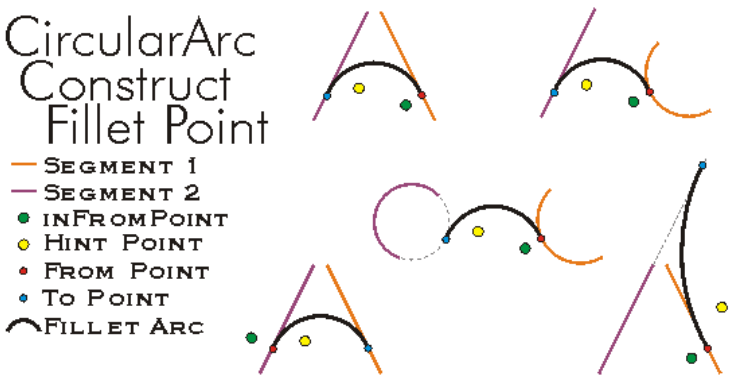


图 4-23　ConstructFilletPoint 构造器

下面是一个构造内切弧的代码段：

```
pConstructCircularArc.ConstructFilletPoint(pLine1, pLine2, pLine1.ToPoint, pHintPoint)
```

6.　ConstructTangentAndPoint

ConstructTangentAndPoint 构造器可以相切于一个 Segment 对象某点的圆弧，如图 4-24 所示。

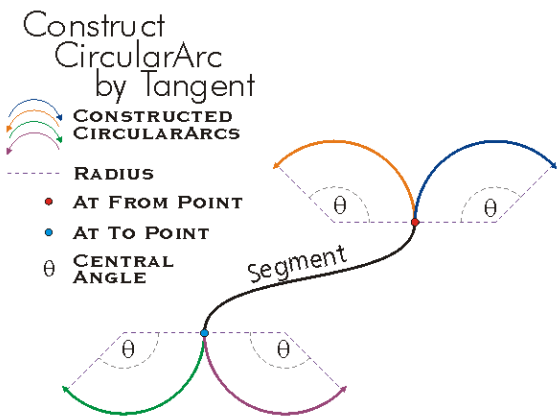


图 2-24 ConstructTangentAndPoint 构造圆弧

如下面的代码段

```
pConstructCircularArc.ConstructTangentAndPoint(pLine1, false, pPoint)
```

构造相切于一个 Segment 对象上某点的圆弧的方法不止一种，除了 ConstructTangentAndPoint 外，还有如 ConstructTangentAngleArc、ConstructTangentAngleChord、ConstructTangentAngleTangent 等方法，都可以用

于产生一个圆弧对象。

7. ConstructThreePoints

ConstructThreePoints 方法是通过三个给定的点对象来产生一个唯一的圆弧，这三个点分别是起始点、终止点和圆弧上的任一点，这个点处于起始点和终止点中间的某一位置。

这个方法的第四个参数 useExistingCenter 仅仅使用在一个已经存在的圆弧上，如图 4-25 所示。

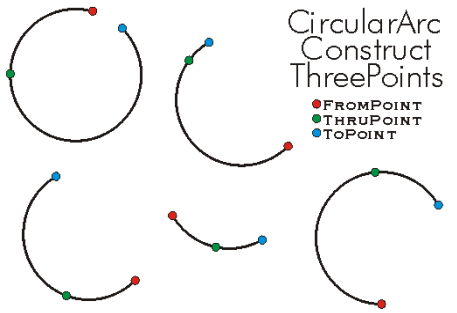


图 2-25 三点构造圆弧

以上介绍的 IConstructCircualArc 方法，都能够以不同的方式构造一个新的圆弧对象出来，这些构造方法都基于几何学原理，更多具体的方法读者可以参看开发帮助的相关条目。

4.4.1.2 Line 对象

Line 是最简单的 Segment，它是由起始点和终止点决定的一条直线段，它是一维几何对象。Line 是最简单和最常使用的 Segment 对象，通常用于构造 Polyline、Polygon、Ring 和 Path 对象等。

ILine 是 Line 对象主要支持的接口，它定义了一系列用于构造和设置线段对象的属性和方法。QueryCoords 属性可以返回通过一个直线上的起始点和终止点的坐标值；而 PutCoords 则用于设置线段的两点坐标；ILine 接口还定义了 Angle 属性，它可以返回线段对象与 X 轴的夹角。

下面是一个设置线段的代码例子：

```
ILine pLine;
IPoint pFromPoint ;
IPoint pToPoint;
```

```
pFromPoint = new PointClass();
pFromPoint.PutCoords(0, 0);
pToPoint = new PointClass();
pToPoint.PutCoords(10, 10);
pLine = new LineClass();
 pLine.PutCoords(pFromPoint, pToPoint);
```

除了使用上面的方法构造一个线段对象外，IConstructLine 接口还提供了两个更复杂的方法来产生一个线段。如 ConstructAngleBiseetor 方法通过传入三个点对象，构造一个夹角，然后通过这个夹角的顶点产生一个角平分线，当然这个方法还需要传入一个线段的长度，如图 4-26 所示。

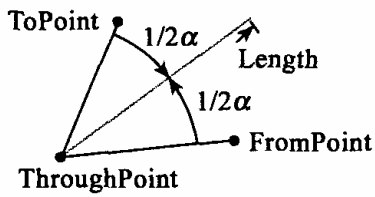


图 4-26　ConstructAngleBiseetor 方法原理

IConstructLine::ConstructExtended 可以通过扩展一个已经存在的线段对象来产生一个新的线段。

4.4.1.3 EllipticArc 对象

EllipticArc(椭圆弧)是一个椭圆的一部分，椭圆是通过一个长轴、一个短轴，中心点和旋转角度值来确定的几何对象，它是椭圆弧的特例，如果旋转角度为 0，则椭圆对象的两个轴分别与 X、Y 轴重合。EllipticArc 对象则还使用诸如 FromAngle、ToAngle 两个参数来确定的，如图 4-27 所示。

CircularArc 对象和 EllipticArc 对象非常类似，但是它们是通过完全不同的方法构造出来的。在使用 EllipticArc 的过程中，如果使用的方法需要 ellipseStd 参数，则有可能改变椭圆弧的坐标系和角度。当它为 False 时，系统使用的是标准笛卡儿坐标系；如果为 True，则有两件事情是不同的，一是所有的角度都是相对坐标，如图 4-27 中，0 度方向不再是笛卡儿直角坐标系，而是和椭圆的长轴一致；二是 FromPoint 和 ToPoint 两点的坐标是与 CentralPoint 是相对的。

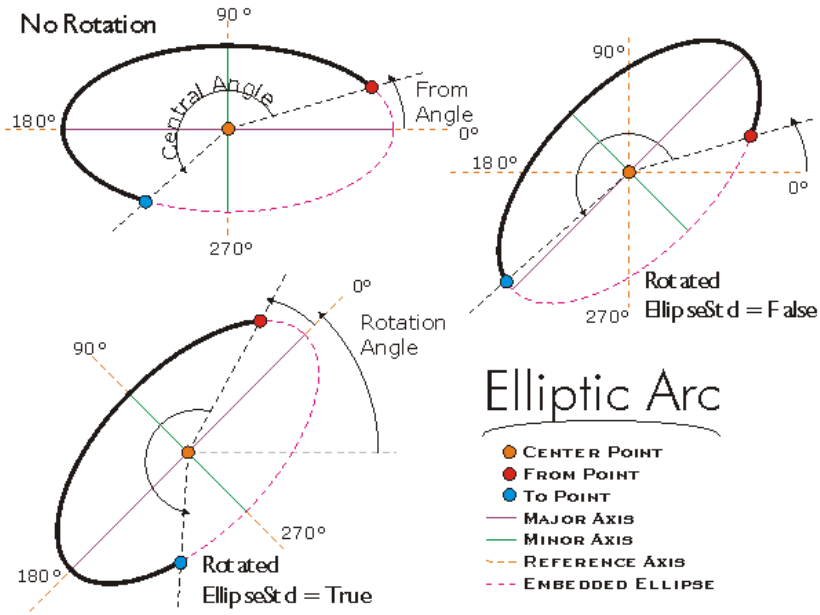


图 4-27　EllipticArc 的主要属性

使用 IEllipticArc 接口定义的属性可以得到一个存在的椭圆弧的 FromAngle、ToAngle 和 CentralAngle，使用 GetAxes 方法则可以得到椭圆弧对象的长半轴和短半轴以及其比例，如下面的代码段：

```
dblFromAngle = pEllipticArc.FromAngle(bEllipseStd);
dblToAngle = pEllipticArc.ToAngle(bEllipseStd);
pEllipticArc.GetAxes(ref dblSemiMajor,ref dblSemiMinor, ref dblMinorMajorRatio);
```

IEllipticArc::PutAxes 方法可以用于改变一个已经存在的椭圆弧对象的半轴长度，在这个改变过程中它的起始点、中心点和终止点还是保持不变的。

IEllipticArc 接口也定义了几个布尔属性用于检查椭圆弧的情况，如：

- IsCircular——长半轴和短半轴长度相等
- IsLine——如果为 True，则短半轴为 0
- IsPoint——如果为 True，则长半轴和短半轴均为 0
- IsCounterClockwise——如果为 True，则 CentralAngle 为正值
- IsMinor——如果为 True，则弧的长度小于椭圆的一半

EllipticArc 对象实现了 IEllipticArc 接口，它拥有诸如 Complement、PutCoord、PutCoords、QueryCoords、QueryCoordsByAngle 和 QueryCenterPoint 等方法用于查询和设置一个椭圆弧对象。这些方法与 ICircularArc 接口定义的类似，只是它们返回的是椭圆的属性。

IEllipticArc::PutCoords 和 PutCoordsByAngle 方法都可以用于设置一个已经定义的椭圆弧的属性，它们使用不同的参数来达到确定一个唯一椭圆弧对象的目的。在 Geometry 中推荐使用后一种方法。

和其他的几何对象一样，EllipticArc 对象也实现了一个专门的构造器接口用于产生椭圆弧，这个接口是 IConstructEllipticArc，本书推荐使用这些高级方法来产生几何对象而不是直接设置各种单独的属性。如图 4-28 所示。

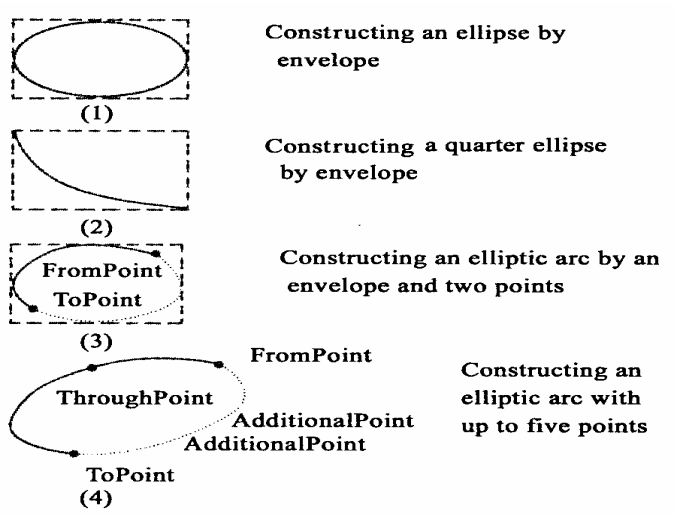


图 4-28　IConstructEllipticArc 定义的构造方法

ConstructEnvelope 可以通过一个给定的包络线对象来产生一个内置的椭圆对象，这个椭圆对象是逆时针方向，其原理如图 4-28 第一个图形。

ConstructQuarterEllipse 构造器要求输入起始点、终止点和方向属性以产生一个椭圆弧，这个椭圆弧是一个 1 / 4 椭圆，如图 4-28 第二个例子，起始点和终止点都是一个椭圆对象的轴点。这个椭圆弧处于一个由起始点和终止点构成的包络线内。

ConsructTwoPointsEnvelope 方法需要输入四个参数，起始点、终止点和包络线，此外需要设置方向属性。这个方法和 ConstructEnelope 相似，但它产生的是一个包络线内的椭圆弧而非整个椭圆对象。

ConstructUpToFivePoints 可以通过输入 5 个点来构造一个椭圆弧，这五个点分别是起始点、终止点、一个弧上任意点以及两个椭圆对象上的附加点。其原理如图 4-28 第四个例子.

4.4.1.4 BezierCurve 对象

贝塞尔曲线(BezierCurve)是由四个控制点定义的，由一组三次多项式描述的参数曲线用于描述等高线或者河流等平滑变化的线对象，如图 4-29 所示。

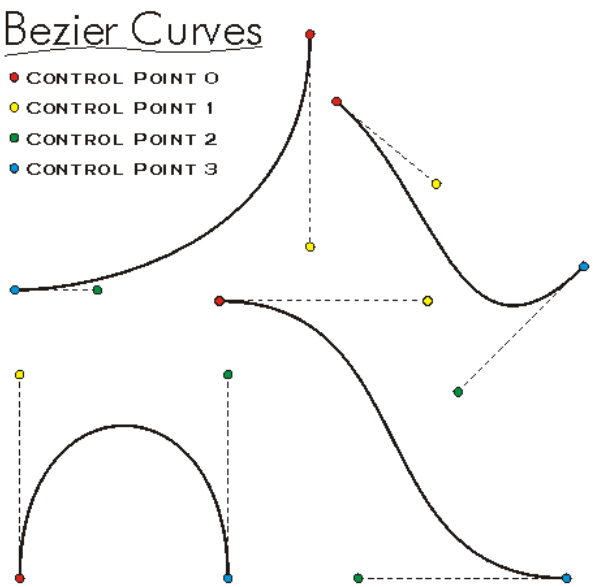


图 4-29 BezierCurve 对象的属性

BezierCurve 对象定义了一条两个节点之间的弧线，如上图，贝塞尔曲线的首尾两点分别与控制点 0 和控制点 1 构成的线段、控制点 2 和控制点 3 构成的线段相切。这四个控制点产生了一条平滑的曲线。

IBezierCurve 接口定义了用于设置和查询曲线属性的方法，IBezierCurve::PutCoord 方法可以用于改变一个已经存在的贝塞尔曲线，这个方法通过一次改变一个控制点来实现，如下面的代码段：

```
IBezierCurve pBezier;
pBezier = new BezierCurveClass();
IPoint pPoint;
pPoint = new PointClass();
pPoint.PutCoords(20, 30);
pBezier.PutCoord(2, pPoint);
```

除了 IBezierCurve 接口定义的方法生成贝塞尔曲线外，还可以使用 IConstructBezierCurve 接口定义的高级方法来实现这个目的。尽管如此，仍然推荐使用 IBezierCurve::PutCoords 方法来产生一个贝塞尔曲线。

4.4.2 路径(Path)对象

路径是连续 Segment 对象的集合，除了路径的第一个和最后一个组成 Segment 外，每一个片断的起始点都是前一个片断的终止点，即路径对象中的 Segment 不能出现分离的情况。路径可以是任意数目的 Line、CircularArc、EllipticArc 和 BezierCurve 的组合。一个或多个路径对象组成一个 Polyline 对象。

IPath 是 Path 对象的主要接口，它定义了设置一个路径对象的多个方法。IPath::Generalizes 方法可以抽象化一个平滑的路径对象，即可以将一个平滑的曲线变成为几条相连的线段；IPath::Smooth 方法则相反，它可以将一个非平滑的路径对象平滑化。IPath::SmoothLocal 方法与 Smooth 方法稍微有些不同，它可以只将某个 Segment 连接点处平滑化，而非如后者一样平滑整条路径对象，如下面的代码段：

```
IPointCollection pPointColl;
pPointColl = pPath as IPointCollection;
int dPoint;
dPoint = 3;
pPath.SmoothLocal(dPoint);
```

一个路径对象的形状可以通过添加新的 Segment 来改变，这个过程需要使用 ISegmentCollection 接口，这个接口被 Path、Ring、Polyline 和 Polygon 实现，这些对象都可以由一个以上的 Segment 对象组成，如下面的代码：

```
pLine1.FromPoint = pPoint1;
pLine1.ToPoint = pPoint2;
pLine2.FromPoint = pPoint2;
pLine2.ToPoint = pPoint3;
ISegmentCollection pPath;
pPath = new PathClass();
pPath.AddSegment(pLinel);
pPath.AddSegment(pLine2);
```

IConstructPath 方法是 Path 对象的构造器接口，ConstructRigidStretch 方法旋转或缩放一个已经存在的路径对象的形状，或者是将这个路径上的某个顶点或片断的位置改变，如图 4-30 所示。

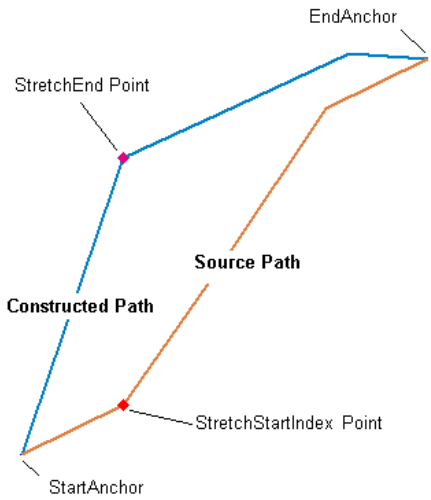


图 4-30　ConsCmctRigidStretch 方法

如图 4-30 所示，移动路径对象上的某个节点或片断，可以改变这个路径对象的形状。

4.4.3 环(Ring)对象

环是一种封闭的路径对象，它的起始点和终止点坐标值是一样的，这种对象具有“内部”和“外部”属性。环是产生 Polygon(多边形)的元素。

组成环的 Segment 对象是有序的，环对象也实现了 IPath 接口. 环必须是封闭的路径，因此它具有以下几个关键特征：

- 它包含一系列首尾相连的同方向的 Segment 对象
- 它是封闭的，即起始点与终止点是同一点
- 它不能自相交

IRing 接口为 Ring 所实现，它定义了多个处理环对象的方法。如 IRing::Close 方法首先检测其起点和终点是否是同一个点，如果是，不作任何处理；否则将添加一条线段到这个开放的环对象上，以连接这个环的起始点和终止点。

下面的代码段可以使用 Segment 对象来组成一个环对象：

```
ISegmentCollection pSegments;
ILine pLine;
IRing pRing;
```

```
pSegments = new RingClass();

//产生一个Segment，然后加入Segment集合对象中
pLine = new LineClass();
pLine.PutCoords(pPoint1, pPoint2);
object Missing = Type.Missing;
object val = 1;
pSegments.AddSegment(pLine as ISegment, ref Missing, ref val);
pLine = new LineClass();
pLine.PutCoords(pPoint2, pPoint3);
pSegments.AddSegment(pLine as ISegment, ref Missing, ref val);
pRing = pSegments as IRing;
//使环封闭
pRing.Close();
```

在 Geometry，封闭的几何形体有三种，即 Envelope、Ring 和 Polygon，因此它们拥有一个别的几何对象没有的特征，即面积，因此这三个对象都实现了 IArea 接口来获取与面积有关的信息。

IArea::Area 可以返回一个具有封闭特性几何形体对象的面积，这些形体都是 2 维或 2.5 维的。IArea::Centroid 可以返回这些几何形体的重心；IArea::LabelPoint 可以返回这些几何形体的标注点。重心和标注点可以在一个位置，也可能在不同的地方，这是因为一个几何形体对象的重心不一定在它身上，但是标注点肯定是在几何对象内。

IArea::QueryCentroid 和 QueryLabelPoint 方法也可以分别返回 2 维或 2.5 维几何形体对象的重心或标注点。

4.4.4 PolyCurve 对象

PolyCurve 是一个抽象类，它代表了一个 Polyline 或 Polygon 对象的边框线。顾名思义，它是由多个曲线构成的对象，Polyline 的每一个组成部分都是一个有效的路径对象；Polygon 的每个组成部分则是环对象。

IPolycurve 接口提供了处理这两种对象的一般方法，如 IPolycurve::SplitAtDistance 和 IPolycurve::SplitAtPoint 都可以通过指定一个距离或点的方式添加一个新的顶点到 Polycurve 上。这些方法都可以用于 Polyeurve 的整形工作。

IPolycurve::Generalize 可以用于给一个 Polycurve 对象进行概化整形，在整形

完毕后，Polyline 或 Polygon 上的平滑曲线将全部变成非平滑曲线(即线段)，这个方法可以参看 IPath::Generalize 的介绍；IPolycurve::Weed 方法与 Generalize 类似；而 IPolycurve::Smooth 方法则刚好相反，它会将一个 Polycurve 对象的每一个组成 Segment 都转换为贝塞尔曲线。事实上，这三种方法的实质都在于它们改变了一个 Polycurve 对象的顶点，它们都需要传入一个偏移值。以 IPolycurve::Smooth 方法为例，如果一条 Polyline 由 10 个线段对象首尾相连组成，则它上面的顶点有 11 个，当对这个对象 Smooth(5)操作后，整条多义线将变平滑，但其顶点的个数却不再是 11 了。读者可以自己试验，看看新的顶点数目。

Polyline 和 Polygon 类都实现了 IGeometryCollection 接口，它可以用于添加、删除和改变一个 Polyline 和 Polygon 对象的组成部分，关于这一类型的 Collection 接口，将在下一节中讨论。

4.4.4.1 Polyline 对象

Polyline(多义线)对象是相连或不相连的路径对象的有序集合，它可以分别是单个路径、多个不相连的路径和多个相连路径的集合，如图 4-31 所示。使用 Polyline 来模拟一条建设中的公路时，如果这个公路中间有未修建的部分，则可以用两个路径对象组成一个 Polyline 对象。Polyline 及相关对象的结构关系如图 4-32 所示。

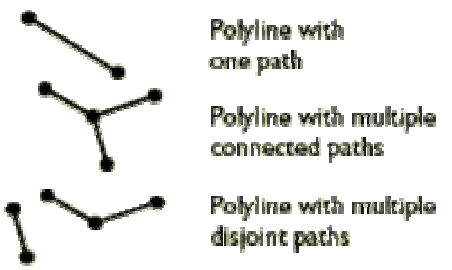


图 4-31　Polyline 由一个或多个 Path 对象组成

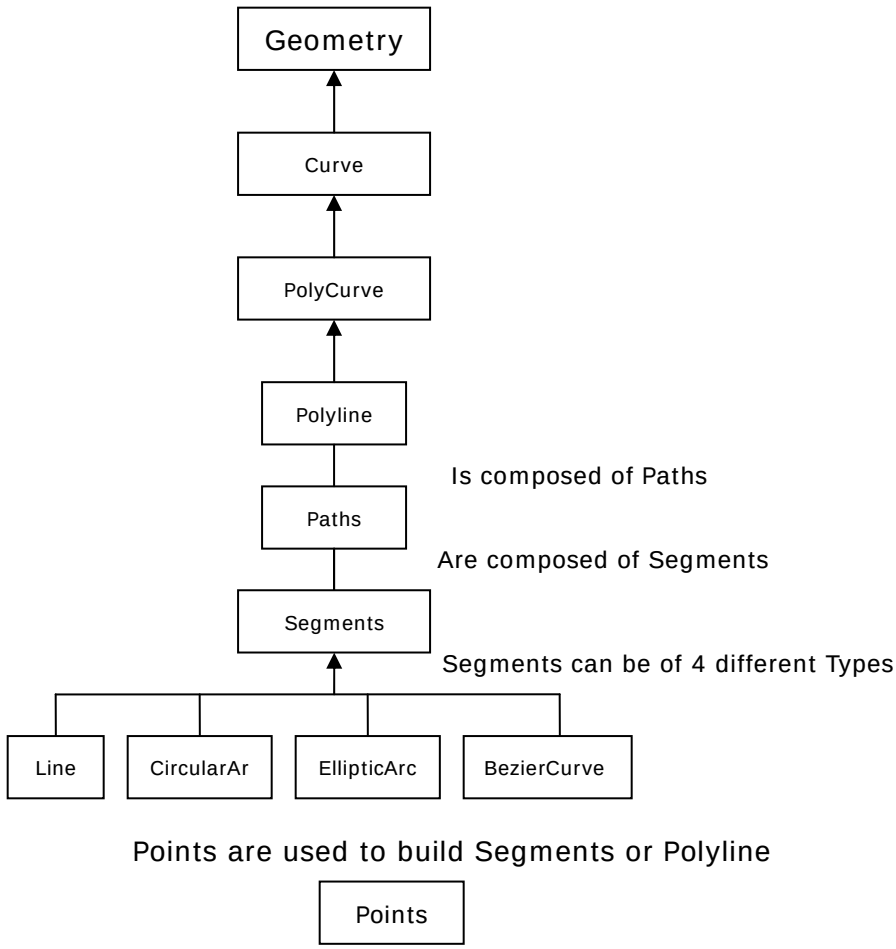


图 4-32 Polyline 及相关对象结构图

一个有效的 Polrline 对象需要满足以下的准则：

- 组成 Polyline 的 Path 对象都是有效的
- Path 不会重合、相交或自相交
- 多个 Path 对象可以连接于某一个节点，也可以是分离的（如上图）
- 长度为 0 的 Path 对象是不被允许的

IPolyline 是 Polyline 类的主要接口，它定义了两个主要的方法. IPolyline::Reshape ，它可以使用一个路径对象给一个存在的 Polyline 整形。另一个方法是 SimplifyNetwork，用于简化网络。

Polyline 可以使用 IGeometryCollection 接口添加路径对象来产生，当使用这个

接口的时候，需要注意以下的情况：

- 每一个路径对象都必须是有效的，或者这个路径对象在使用 IPath::Simplify 方法后有效
- 由于 Polyline 是路径对象的有序集合，因此在添加路径时必须注意顺序和方向
- 为了保证 Polyline 是有效的，可以在产生这个形状后使用 Simplify 方法

下面是一个使用 IGeometryCollection 接口来产生一个 Polyline 的例子

```
//产生构造线段的点
IPoint pPoint1;
IPoint pPoint2;
pPoint1 = new PointClass();
pPoint1.PutCoords(100, 20);
pPoint2 = new PointClass();
pPoint2.PutCoords(20, 310);
IGeometryCollection pPolyline;
pPolyline = new PolylineClass();
ISegmentCollection pPath;
pPath = new PathClass();
//产生线段对象，将其添加到路径对象
ILine pLine ;
object Missing1 = Type.Missing;
object Missing2 = Type.Missing;
pLine = new LineClass();
pLine.PutCoords(pPoint1, pPoint2);
pPath.AddSegment(pLine as ISegment,ref Missing, ref Missing2);
//将路径对象添加到多义线对象
pPolyline.AddGeometry(pPath as IGeometry,ref Missing, ref Missing2);
```

4.4.4.2 Polygon 对象

Polygon(多边形)对象是一个有序环对象的集合，这些环可以是一个或者多个。如图 4-33 中，第一个例子是由一个单环对象构成的多边形；第二个例子是使用多个非连接 Ring 对象构成的多边形；第三个多边形是环中有环，即存在一个岛环。多边形对象通常可以用于描述具有面积的多边形离散矢量对象。Polygon 及相关对象结

构关系如图 4-34 所示。

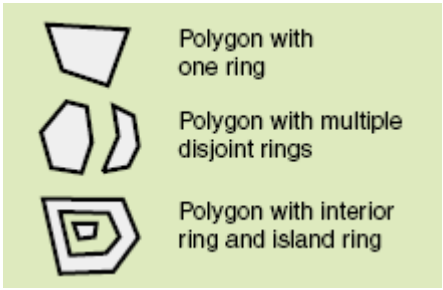


图 4-33　Polygon 对象

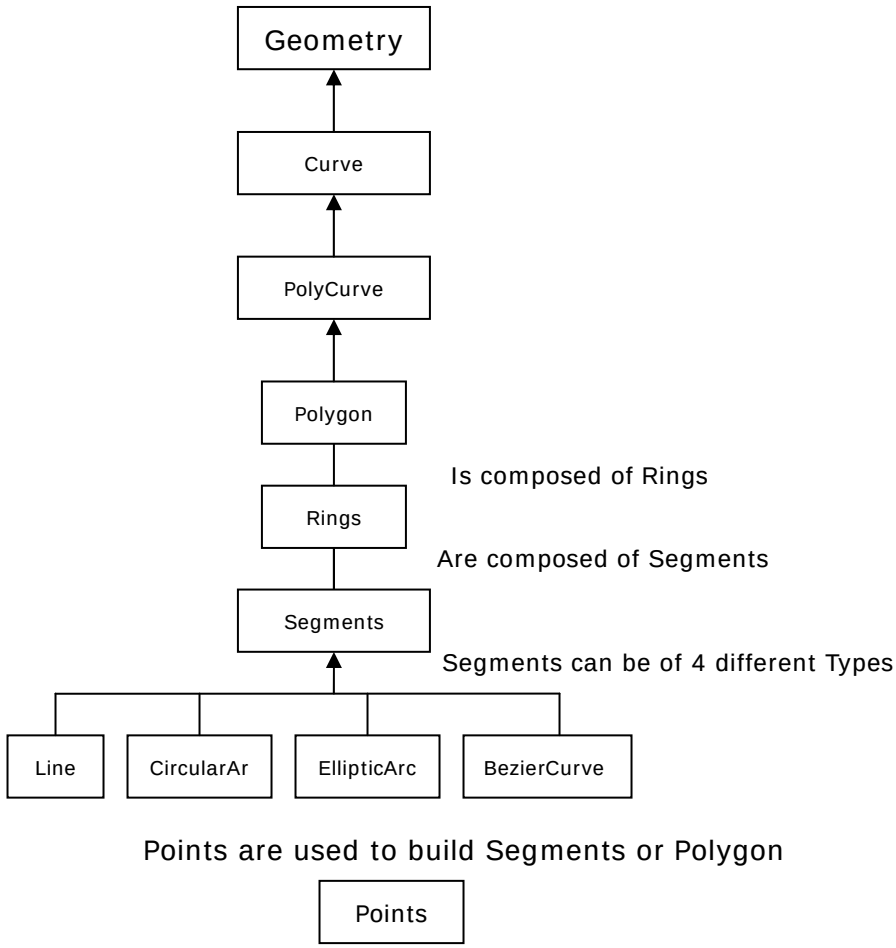


图 4-34 Polygon 及相关对象结构图

一个有效的多边形符合下面的标注：

- 每一个构成的环都是有效的
- 环之间的边界不能重合
- 外部环是有方向的，它是顺时针方向
- 内部环在一个多边形中定义了一个洞(如图 4-33 中第三个例子)，它是逆时针方向
- 面积为 0 的环是不允许的
- 多边形上存在一个 Segment 对象或路径对象是无效的

IPolygon 接口是 Polygon 类主要接口，它定义了一系列方法和属性来控制一个多边形的环。

IPolygon::ExteriorRingCount 属性可以返回一个多边形的全部外部环的数目；IPolygon::InteriorRingCount 则返回一个多边形的内部环数目。

如图 4-35 所示的多边形对象，其中有 5 个外环和 3 个内环，即红色环是外环，绿色环是内环，外环的区域使用了颜色填充，读者可以比较一下这两个对象之间的关系。

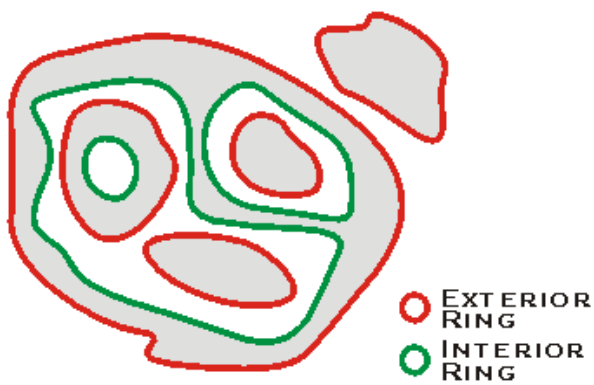


图 4-35　内环与外环

IPolygon : :QueryExteriorRings 和 QueryInteriorRings 两个方法可以分别用于得到所有的外部环和内部环对象。IPolygon::Close 方法会调用 IRing::Close 方法来封闭一个多边形的组成部分，有关这个方法的使用可以参见 Ring 章节的内容。

和 Polyline 对象一样，Polygon 可以使用 IGeometryCollection 接口定义的方法，通过添加环对象来产生一个多边形。在这个过程中需要注意以下几点：

- 每一个组成多边形的环都必须是有效的

- 在产生一个多边形后，可以使用 Simplify 或者 SimplifyPreserveToFrom 来保证它的内环和外环的方向是正确的，环没有相交或自交，而且环是封闭的
- 除了使用 Simplify 方法来获得多边形环和顶点的数目外，还可以使用 IPointCollection 的方法来检查。IPointCollection 接口的方法可以构成任意层次上的集合图形。

下面是一个使用 Segment 构成多边形的例子。

```
//产生一个环，向其中添加Segment对象
ISegmentCollection pSegCol;
pSegCol = new RingClass();
object Missing1 = Type.Missing;
object Missing2 = Type.Missing;
pSegCol.AddSegment(pSegmentA,ref Missing1, ref Missing2);
pSegCol.AddSegment(pSegmentB,ref Missing1, ref Missing2);
//封闭环对象，使其有效
IRing pRing ;
pRing = pSegCol as IRing;;
pRing.Close();
//使用环对象来构造多边形对象
IGeometryCollection pPolygon;
pPolygon = new PolygonClass();
pPolygon.AddGeometry(pRing,ref Missing1, ref Missing2);
```

4.5 Geometry 集合接口

从前面对于具体 Geometry 对象的介绍，读者可以了解一个事实：除了 Point 对象外，其他的几何形体对象都是可以通过集合的方式构成的。如点集对象是点的集合，路径是 Segment 对象的集合，Polyline 是路径的集合等。

ArcObjects 中，Geometry 类型的集合接口主要有 IGeometryCollection、ISegmentCollection 和 IPointCollection 三个，这些接口揭示出 ArcObjects 几何对象模型的实质--它们是一种组合构成的形式，但是这种组合并不一定按照严格的层次结构组织的。

4.5.1 IGeoemtryCollection 接口

IGeometryCollection 是被多种几何对象实现的接口，它们包括 Polygons、Polylines、Multipoints、MultiPatches、Triangles、TriangleStrips、TriangleFans 和 GeometryBags。它的方法可以让程序员添加、改变和移除一个几何对象的组成元素，即它们的子对象。这些元素包括：

- 对于 Polygon 而言，每个组成的 Geometry，即子对象是 Ring 对象
- 对于 Polyline 而言，其子对象是 Path
- 对于 Multipoint 而言，其子对象是 Point
- 对于 MultiPath 而言，其组成的 Geometry 是 TriangleFan、Triangle-Strip 或 Ring(MultiPath 是 3 维几何对象)
- 对于 CeometryBag 对象而言，组成它的 Geoemtry 可以是任何类型的几何形体对象。实际上，GeometryBag 是一个容纳任何类型几何对象的容器.

大部分几何对象都是有序的几何对象的集合，因此它们的子对象是拥有索引值的，而这个索引值确定了它们在组成方向上的排列顺序，使用 IGeometryCollection::Geometry 属性可以通过一个索引值来返回一个组成这个几何对象的某个子对象，而 GeometryCount 则返回这些子对象的数目。

ICeometry::AddGeometry 和 AddGeometries 方法都用于向一个几何对象添加子对象，其差别在于前者一次只能添加一个几何对象，而后者可以一次添加一个几何数组；除此以外，AddGeometry 方法可以将子对象添加到几何的指定索引值位置上，AddGeometries 方法则只能将子对象数组添加到集合的最后。

在使用 AddGeometry 方法添加子对象到 Polygon 对象的过程中，如果子对象(即 Ring)中出现覆盖现象，那多边形就没有封闭或出现了一个包含的关系，这个多边形就不是简单多边形，因此，在新建一个多边形后应该使用 Simplify 方法保证它的有效性，如下面的代码：

```
IGeometryCollection pGeomColl1;
IGeometryCollection pGeomColl2 ;
pGeomColl1 = pExistingPolygon1 as IGeometryCollection;
pGeomColl2 = pExistingPolygon2 as IGeometryCollection;
object Missing1 =Type.Missing ;
object Missing2 =Type.Missing ;
 pGeomColl1.AddGeometry(pGeomColl2.get_Geometry(0),ref  Missing1,  ref
Missing2);
ITopologicalOperator pTopological ;
pTopological = pGeomColl1 as ITopologicalOperator;
pTopological.Simplify();
```

IGeometryCollection::AddGeometryCollection 方法则可以将一个多边形中所有的子对象的引用添加到某个多边形中，即它可以合并两个多边形，如下面的代码段：

```
pGeomColl1.AddGeometryCollection(pGeomColl2);
```

IGeometryCollection::InsertGeometries 或 InsertGeometryCollection 方法可以将子对象加入到一个已经存在的 GeometryCollection 对象中，并且可以指定插入的索引号，如下面的过程：

```
private void AddGeometry(IGeometryCollection pGeomColl1)
{
    IPoint[] pPt = new PointClass [2];
    IPoint p1 ;
    IPoint p2 ;
    p1 = new PointClass();
    p2 = new PointClass();
    p1.PutCoords(1000, 1000);
    p2.PutCoords(2000, 2000);
    pPt[0] = p1;
    pPt[1] = p2;
    IGeometry pGeom = pPt[0] as IGeometry ;
    pGeomColl1.InsertGeometries(3, 2, ref pGeom);
}
```

4.5.2 ISegmentCollection 接口

ISegmentCollection 接口被 Path、Ring、Polyline 和 Polygon 四个类所实现，因此它们都可以被称作是 Segment 集合对象，使用这个接口可以处理 Segment 集合对象中的每一个组成元素--Segment 对象。

SegmentCollection 中的任何一个子对象都可以使用 ISegmentCollection::Segment 属性来获得，这个属性需要输入一个子对象的索引值。

ISegmentCollection::EnumSegments 可以返回一个 SegmentColleetion 对象中的 Segment 对象，它作为一个枚举值返回；AddSegment 方法用于向一个 SegmentColleetion 对象中添加单个的 Segment；AddSegments 可以用于添加一个 Segment 数组对象；AddSegmentColleetion 方法则可以用于添加另一个 SegmentCollection 对象中所有的 Segment，即这个方法可以合并两个 SegmentCollection 对象，需要注意的是这两个 SegmentCollection 都是同一个空间

参考。

ISegmentCollection::InsertSegments、InsertSegmentCollection、RemoveSegments、ReplaeeSegments、ReplaceSegmentCollection、SetSegmentCollection 和 SetSegments 都是用于改变和重新排列这个 Segment 集合对象中的 Segment，这样可以改变 Segment 集合对象的形状。

SetCircle 和 SetRectangle 方法提供了一种简单的在不需要添加 Segment 的情况下构造一个完整的路径、环、多义线或多边形方法。SetCircle 方法和 ICircularArc::ConstructAre 方法类似。

关于 ISegmentCollection 定义的方法如何逐级组合成一个高级几何对象，可以看关于路径对象的内容。

4.5.3 IPointCollection 接口

IPointColleetion 接口被多个几何对象类所实现，这些对象包括都可以由多个点构成，如 Mulipoint、Path、Ring、Polyline、Polygon、TriangleFan、TriangleStrip 和 MultiPateh 等，它们都可以称为 PointCollection 对象。PointCollection 对象可以 IPointColleetion 接口定义的方法获取、添加、插入、查询、移除它中间的某个顶点。

和上面两个接口定义的方法一样，IPointCollection 接口也定义了操作一个点集合对象的方法，如 AddPoint 方法可以往一个 PointCollection 对象中的特定索引位置添加一个点对象，如果不指明这个新点的前后位置，则点将被加在 PointCollection 的最后面。

使用 AddPoints 和 AddPointCollection 方法则可以分别加入多个点和一个点的集合到一个 PointCollection 对象中去。

使用 IPointColleetion::Point 属性可以通过顶点的索引值来得到 PointCollection 中的某个顶点对象，而 PointCount 则可以返回这个点的集合中的子对象的数目。第一个点的索引值为 0，而最后一个点的值为 PointCount -1。

集合接口定义的方法基本上都是类似的，它们很好地诠释了 Geometry 对象模型的实质。关于这三个接口的具体方法的使用，可以参阅开发帮助的相关内容。

4.6 动态跟踪

在 AreObiects 中经常需要动态跟踪的功能，如查找一个公共汽车的线路，如果能让一个汽车符号沿整个线路移动，则它是一种非常直观的表现方式，可以让用户

更清楚看到这条线路的趋势。在 MapObjects 中有一个 trackingLayer 对象专门完成这些功能，但是在 ArcObjects 里面只需要移动一个 element 就可以了，下面介绍一种方法来达到这个目的。

这个算法的原理首先对一条多义线对象进行处理，取出它上面的符合要求的等距点集，然后在 Timer_Tick 事件里面不断改变一个 MarkerElement 的 Geometry 属性，将它有顺序地设置为点集中的不同的点，这样就可以让这个点元素不断移动，因此关键是要知道如何获得所需要的点。

下面是算法获得一个 polyline 的点,这个算法是获取线上的每个节点。pPointCol 和 i 是全部变量：

```
IPointCollection pPointCol;
int i;
IElement   pElek = new MarkerElementClass ();
private void timer1_Tick(object sender, System.EventArgs e)
{
    IPoint pPt ;
    int j;
    if (i > pPointCol.PointCount - 1) i = 0;
    j = i;
    pPt = pPointCol.get_Point(j); //每个tick事件,pPt  就变化一次
    pElek.Geometry = pPt;   //pElek的位置发生变化
    i = i + 1;
axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewGraphics,null,null);
}
```

下面的这个函数用于获取一个 Polyline 上的均匀点集，其参数 nPoints 是需要产生的点的数目：

```
private IGeometryCollection MakeMultipoint(IGeometry pGeometry,int nPoints)
{
    IGeometryCollection pGeometryCollection;
    if ( pGeometry is IPolyline)
    {
        //返回分布在一条多义线上的等长的点构成的点集对象
        IConstructGeometryCollection pConstructGeometryCollection;
        pConstructGeometryCollection = new GeometryBagClass();
        pConstructGeometryCollection.ConstructDivideEqual(pGeometry as
IPolyline, nPoints - 1, esriConstructDivideEnum.esriDivideIntoPolylines);
        IEnumGeometry pEnumGeometry;
```

```
                    pEnumGeometry = pConstructGeometryCollection as IEnumGeometry;
                    pGeometryCollection = new MultipointClass();
                    IPolyline pPolyline;
                    pPolyline = pEnumGeometry.Next() as IPolyline;
                    object Missing1 = Type.Missing;
                    object Missing2 = Type.Missing;
                    pGeometryCollection.AddGeometry(pPolyline.FromPoint,ref Missing1, ref
Missing2);
                    while (pPolyline != null)
                    {
                        pGeometryCollection.AddGeometry(pPolyline.ToPoint,ref Missing1,
ref Missing2);
                        pPolyline = pEnumGeometry.Next() as IPolyline;
                        return pGeometryCollection;
                    }
                }
                return null;

            }
```

获取点后，在 timer_tick 事件中，元素的改变就应该是

```
pElek.Geometry = pGeoCol.get_geometry(i);
```

4.7 空间参考

4.7.1 什么是空间参考

为了空间数据能够被合适地存储和指向地球上某一个位置，它必须设置空间参考。空间参考包含了一个坐标系统和经度。坐标系统(地理坐标系统或投影坐标系统)定义了空间数据在地球上的位置。在同一个地图上显示的地理数据的空间参考必须是一致的，如果两个图层的空间参考不一致，往往导致两幅地图无法正确拼合，因此在一个 GIS 系统中，选择正确的空间参考是非常重要的。

在地理数据库中新建一个要素数据集和——个单独的要素类都必须设置它们的空间参考。图 4-36 是新建一个要素类时设置空间参考的对话框。

精度(Precision)定义了存储在地理数据库中的数据细节程度。地理数据库存储坐标是使用一个 4 比特的正整数，它的最大值就是 2 147 483 648，这个数值被称

为空间域(SpatialDomain)。如果需要使用米作为测量单位，则最大可以为 21 亿 m(大约为地球周长的 53 倍)；如果使用厘米，则最多可以使用 21 亿 cm(大约为地球周长的一半)。这 4 比特整数代表的单位称为存储单位，它是一个数据集中最小的测量单位。

存储单位 = 坐标系统单位 / 精度

如果坐标系单位是米，而精度为 100，则存储单位为 1cm；如果精度为 1 000，则存储单位为 1mm。

空间域与精度的关系是相反的，它表示某个区域的范围。由于可以使用的最大值为 21 亿整数，因此空间域是有一个范围边界的。使用的存储单位越小，则精度越高，而空间域会变小，这个原因很好理解，使用毫米比厘米的精度高，但 21 亿 mm 定义的范围比 21 亿 cm 定义的范围小得多。当用户将要素添加到空间域范围外时，会出现一个“错误坐标系统或精度超出了范围”。在这种情况下，需要改变存储单位来换取更大的空间域范围。

设置空间域范围，必须确定空间域的左下角点和右上角点，这两个点可以确定一个矩形区域。除了 X、Y 坐标外，空间域还可以设置 Z 值和 M 值的范围。

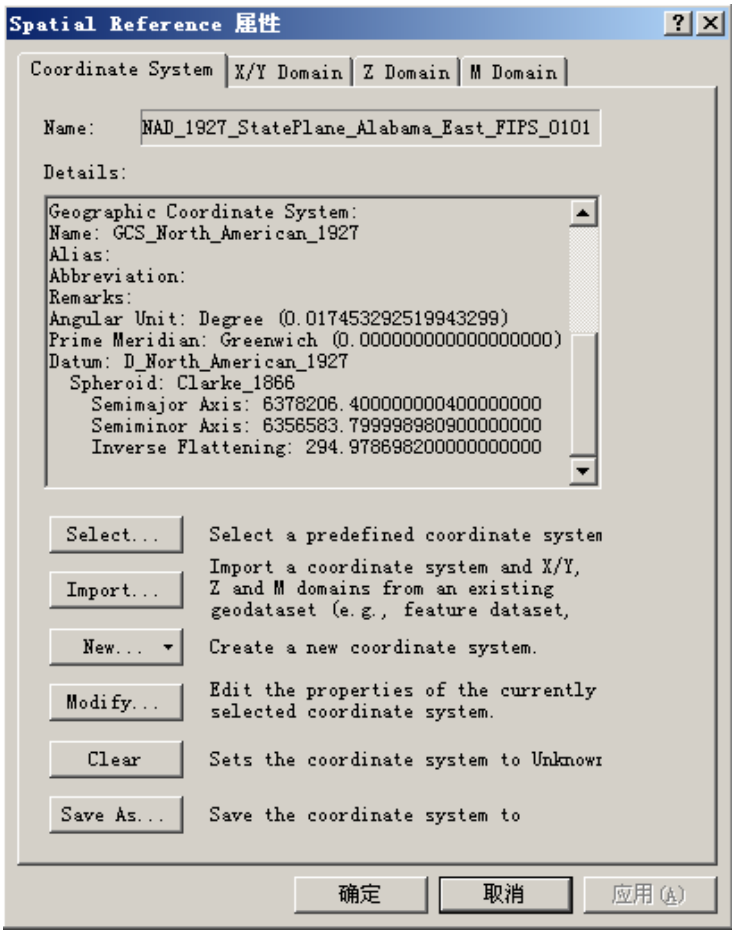


图 4-36　空间参考对话框

- **如何设置一个范围的空间域?**

如果有一张城市的电子地图，希望将它的一个区域作为单独的数据保存在地理数据库中，这时用户可以使用整个城市要素类的空间参考，包括精度和空间域。但在地图上的很大一部分空间并不用来显示任何数据的，数据只出现在定义的空间域的一部分区域。为了将空间域的范围变小而提高显示的精度，可以在新建的要素类中确定一个区域精确的空间域。

打开 ArcMap 程序，将城市地图载入，然后将地图缩放到合适大小，即要选取的数据部分，使用 Draw 工具栏上的“new Rectangle”工具在要选择的地方画一个矩形，右击矩形打开属性栏，如图 4-37 所示。

在上面 Anchor 中的 9 个矩形点中分别点击左下角和右上角两点，可以在 Position 中得到两点的 x、Y 坐标，它们可以确定这个矩形的区域，这就是要选取的

数据的范围域，在新要素类的设计中，可以直接设置这个范围域。

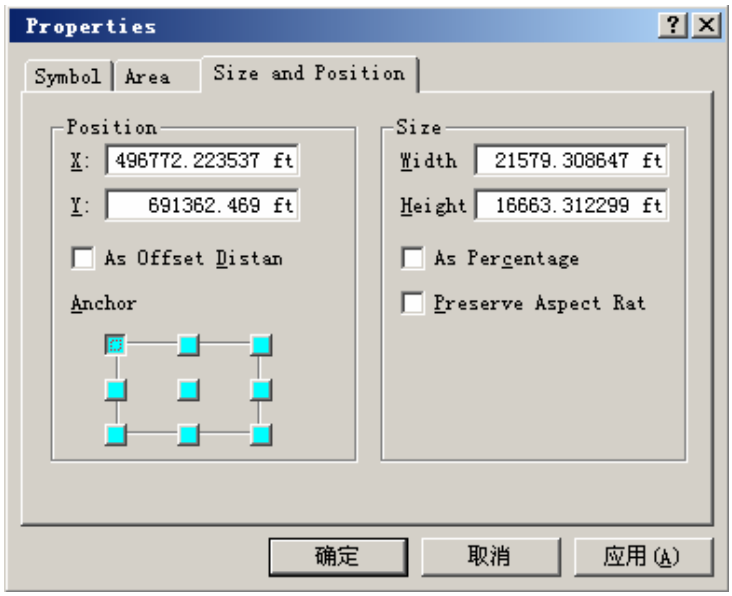


图 4-37　矩形元素属性

4.7.2 两种坐标系统

如何确定地球上的一点的精确位置?这是地理坐标系统要回答的问题。由于地球是一个球体，把地球上的一点转换到二维的平面地图上，依赖用户选择的投影坐标系统。因此，将地球上的一点转换到平面地图上有两个阶段，即确定某点在地球上的唯一标识坐标和将唯一标识坐标进行投影平面化。

这就必须对在 ArcGIS 中经常使用的两种坐标系统--地理坐标系统和投影坐标系统进行认识。

地理坐标系统(Geographic coordinate system)，也可称为真实世界的坐标系，是确定地物在地球上位置的坐标系。它是以经纬度为地图的存储单位的，而经纬度是角度。要确定地球上一点的坐标必须对地球进行数字模拟，要求抽象出一个与地球相似的椭球体，并且这个椭球体拥有以下特点：可以进行量化计算的，具有长半轴、短半轴、偏心率。

由于推求椭球体的年代、使用的方法以及测定的地区不同，其结果往往并不一致，因此地球椭球体的参数值有很多种，如海福特(Hayford)、克拉索夫斯基(Krasovsky)等。中国在 1952 年以前采用海福特椭球体，从 1953—1980 年采用克拉索夫斯基椭球体。随着人造地球卫星的发射，有了更精密的测算地球形体的条件。

1975 年第 16 届国际大地测量及地球物理联合会上通过国际大地测量协会第一号决议中公布的地球椭球体，称为 GRS(1975)，中国自 1980 年开始采用 GRS(1975)新参考椭球体系。由于地球椭球长半径与短半径的差值很小，所以当制作大比例尺地图时，往往把它当做球体看待，这个球体的半径为 6371 公里。

以下几行便是 GRS 1980 椭球及其相应参数。

Spheroid：GRS_1980

Semimaior Axis：6378137.000000000000000000

Semiminor Axis：6356752.314140356100000000

Inverse Flattening：298.257222101000020000

然而有了这个椭球体以后还不够，地理坐标系统还需要一个大地基准面将这个椭球定位，这个基准面将定位地球上点的参照系统，定义经纬线的起点和方向。基准面的建立需要选择一个椭球，然后在地球上选择一个点作为“原点”，椭球上所有其他的点都相对于这个原点进行位置定义。

大地基准面除了全球基准面 WGS84，WGS72 外，不同的地方还可以使用自己的本地基准面，如中国常常使用的北京 1954，西安 80，欧洲基准面 ED50 等，这些基准面之间是可以互相转化的。

在坐标系统描述中，常常可以看到这么一行：

Datum：D_Beijing_1954

这表示大地基准面是 D_ Beijing _1954，即北京 1954 基准面。有了 Spheroid(椭球体)和 Datum(基准面)两个基本条件，地理坐标系统便可以使用。

下面是一个地理坐标系的完整参数：

Alias：

Abbreviation

Remarks：

AngularUnit：Degree(0．0174532925199433299)

Prime Meridian：Greenwich(0．000000000000000000)

Datum：D_ Beijing _1954

Spheroid：Krasovsky_1940

Semimaior Axis：6378245．000000000000000000

Semiminor Axis：6356863．018773047300000000

lnverse Flattening：298．300000000000010000

地理坐标系统是最常用的坐标系对象，它以经纬度来描述地面位置。

经度通常用字母入表示。国际规定通过英国格林尼治天文台的子午线为本初子午线，作为计算经度的起点，该线的经度为 0°，向东 0 -180 ^{o}叫东经，向西 0-180°叫西经。纬度通常以字母φ表示。纬度从赤道起算，在赤道上纬度为 0°，纬线离赤道愈远，纬度愈大，至极点纬度为 90°。赤道以北叫北纬，以南叫南纬。地面上任一点的位置，通常用经度和纬度来决定。经线和纬线是地球表面上两组正交(相交为 90°)的曲线，这两组正交的曲线构成的坐标，称为地理坐标系。地表面某两点经度值之差称为经差，某两点纬度值之差称为纬差。例如北京在地球上的位置可由北纬 39° 56’和东经 116^o24’来确定。

经纬度在度量上是不均匀的，比如经度在赤道上 1^o是 111km，而在北纬 60^o是 55.8km，在南北极点处则是 0。

投影坐标系统(Projection coordinate system)是将三维地理坐标系统上的经纬网投影到二维平面地图上使用的坐标系统，这是非常必要的。因此地理信息系统必然要考虑到地图投影，地图投影的使用保证了空间信息在地域上的联系和完整性，在各类地理信息系统的建立过程中，选择适当的地图投影系统是首先要考虑的问题。

由于地球椭球体表面是曲面，而地图通常是要绘制在平面图纸上，因此制图时首先要把曲面展为平面，但是球面是个不可展的曲面，即把它直接展为平面时，不可能不发生破裂或褶皱。为了防止这种情况的发生，地图学家采用了各种特殊的方式来展开这个球面，这些方法都是在保证某一特性不变的情况下牺牲其他的属性，如等角投影、等积投影和正形投影等。

在大比例地图的选用中，中国一般使用高斯·克吕格，在欧美这种投影方法称为横轴墨卡托投影。

下面是一个高斯·克吕格投影坐标系统中的一些参数

Projection：Gauss_Kruger

Parameters：

False_Easting：500000.000000

False_Northing：0.000000

Central_Meridian：117.000000

Scale _Factor：1.000000

Latitude_Of_Origin：0.000000

Linear Unit：Meter(1.000000)

Geographic Coordinate System：

Name：GCS_Beijng_1954

Alias：

Abbreviation:

Remarks：

Angular Unit：Degree(0．017453292519943299)

Prime Meridian：Greenwich(0．000000000000000000)

Datum：D_Beijing_1954

Spheroid：Krasovsky 1940

Semimajor Axis：6378245．000000000000000000

Semiminor Axis：6356863．018773047300000000

Inverse Flattening：298. 300000000000010000

从参数中可以看出，每一个投影坐标系统都必定会有地理坐标系统。投影坐标系统，实质上就是平面坐标系统，其地图单位通常为米。投影坐标系使用 x，y 坐标来描述地面上的位置，它用地球椭圆球体 spheroid 来模拟地球，它使用 projection 表示投影计算方法，使用 unit 表示单位，用 geocoordsys 表示投影坐标系来源。也就是说，要得到投影坐标就必须得有一个用于投影的球面坐标，然后才能使用投影算法去进行投影，即每一个投影坐标系统都必须要求有地理坐标系统参数。

4.7.3 A0 中空间参考的设置

ArcObjects 中提供了一系列对象供开发者管理坐标系统。对于大部分开发者而言，需要了解到三种主要的 ArcObjects 组件：ProjectedCoordinateSystem，GeographicCoordinateSystem 和 SpatialReferenceEnvironment。对于高级开发者而言，可能需要自定义的坐标系统时，可以使用这些对象：Projection，Datum，AngularUnit，Spheriod，PrimeMeridian 和 GeoTransformation。

在空间参考设置中，用户经常可以看到空间参考对话框，这个对象可以使用ISpatialRefrenceDialog 接口打开。

- 改变一个图层的空间参考

```
IFeatureLayer player ;
player = pMap.get_Layer(0) as IFeatureLayer;
IFeatureClass pFeatureClass;
pFeatureClass = player.FeatureClass;
//QI，从要素类得到数据集
IGeoDataset pGeoDataset;
pGeoDataset = pFeatureClass as IGeoDataset;
//QI，从地理数据集得到IGeoDatasetSchemaEdit
IGeoDatasetSchemaEdit pGeoDatasetEdit;
pGeoDatasetEdit = pGeoDataset as IGeoDatasetSchemaEdit;
if (pGeoDatasetEdit.CanAlterSpatialReference == true)
{
    ISpatialReferenceFactory2 pSpatRefFact;
    pSpatRefFact = new SpatialReferenceEnvironmentClass();
    IGeographicCoordinateSystem pGeoSys;
    pGeoSys =
pSpatRefFact.CreateGeographicCoordinateSystem(4214);//esriSRGeoCSType .esriSRGeoCS
_Beijing1954
    pGeoDatasetEdit.AlterSpatialReference(pGeoSys);
}
    pActiveView.Refresh();
```

- 得到一个图层的空间参考

```
IFeatureLayer pLayer;
pLayer = pMap.get_Layer(0)as IFeatureLayer;
IGeoDataset pGeoDataset;
ISpatialReference pSpatialReference;
pGeoDataset = pLayer as IGeoDataset;
pSpatialReference = pGeoDataset.SpatialReference;
MessageBox.Show(pSpatialReference.Name);
```

- 设置一个地图的空间参考，使用一个空间参考对话框

```
IProjectedCoordinateSystem pSpatialReference;
ISpatialReferenceDialog pDialog ;
```

```
            pDialog = new SpatialReferenceDialogClass();
            pSpatialReference = pDialog.DoModalCreate(true, false,false,0) as
IProjectedCoordinateSystem;
            pMap.SpatialReference = pSpatialReference;
            pActiveView.Refresh();
```

4.7.4 SpatialReference 对象

ArcObjects 中的空间参考对象模型中的有三个组件类：GeographicCoordinateSystem(地理坐标系统)、ProjectedCoordinateSystem(投影坐标系统)和 UnknownCoordinateSystem(未知坐标系统)，它们都实现了 ISpatialRefrence 接口，该接口提供了操作方法和属性来设置一个数据集空间参考属性，如空间域和坐标精度等。这些方法和属性对于三个子类都是可以使用的。一般地，大部分开发者都不需要设置或使用空间参考的属性，他们可以直接使用已经在 ArcObjects 中定义好的坐标文件，这些 prj 文件足够用户使用。

ISpatialReference::Changed 是这个接口中最重要的方法，它可以检测一个坐标系统中的参数是否发生了变化；ISpatialReference::GetDomain 和 SetDomain 方法分别用于获得和设置一个坐标系统的域范围，它是一个矩形范围，用以确定数据显示的范围。因此无论是获取还是设置，都是使用四个值 XMax、XMin、YMax 和 YMin。ISpatialReference::GetFalseOriginAndUnits 和 SetFalseOriginAndUnits 分别是另一种获得和设置空间域的方法，它们的三个参数 falseX 和 falseY 值分别代表了空间域的 Minx 和 MinY，而 xyUnits 则是空间域的精度值。ISpatialReference::GetMDomain 和 GetZDomain 方法分别用于获得一个要素类或要素数据集的 M 值域和 Z 值域，当然，ISpatialReference 接口也定义了它们两个的 Set 方法。

GeographicCoordinateSystem 确定一个地理坐标系统，可以使用 IGeographicCoordinateSystem 接口，设置一个地理坐标系统的 CoordinateUnit(坐标系的角度单位)、Datum(椭球体)、PrimeMeridian(首子午线，即本初子午线)属性。

ProjectedCoordinateSystem 使用 IProjectedCoordinateSystem 接口新建一个投影坐标系统，在一个新的投影坐标系统中，需要设置 GeographicCoordinateSystem，Projection(投影方式)，CoordinateUnit 和 Parameters。

本章介绍了基于 ArcObjects 的几何对象模型和空间参考的内容。ArcObjects 的几何对象模型在实质上是一种逐级组合的关系，通过线段组合为路径，路径组合为

多义线，环组合为多边形，正是这众多的几何对象，才使得 ArcObjects 能够使用更加丰富、精确的方法来模拟现实世界的离散矢量对象，这也是几何模型对象比以前的地理数据文件(Coverage、Shapefiles 等)先进的地方。为了让地理数据能够精确地模拟地球上的实际区域，任何一个 GIS 系统都必须设置坐标系统和空间域，以确定数据的投影方式和显示范围。

5 地图的组成

地图（Map）是 ArcObjects 的主要组成部分，本章将讲述关于地图组成方面的话题，如地图的组成、如何创建地图、如何对地图的组成对象进行操作等，实际上本章就是在讨论 ArcMap 自文档对象之下的框架结构。ArcMap 提供了数据显示、查询、绘制和出版地图的众多工具，使用它可以轻松地对地图进行查询、分析和设计排版。这些功能的实现，都依赖于 ArcObjects 各个组件的“幕后”工作，这些组件之间的配合不是杂乱无章的，而是存在一定的内在逻辑性。

在学习完本章以后，读者将可以了解到如下内容：

- Map 对象
- 图层对象
- PageLayout 对象
- 图形元素与框架元素
- MapGrid 对象
- MapSurround 对象
- 使用样式

5.1 Map 对象

打开 ArcMap 程序后，用户首先看到的是数据视图(data view)。ArcMap 的主要功能查看数据和地理分析都是在这个视图中完成的。数据视图其实是一个 Map 对象。在 ArcMap 中 Map 是由文档对象(MxDocument)控制的，每一个地图文档至少包含一个 Map 对象，但是在一个时刻仅仅只有一个 Map 处于使用状态，程序中这样的 Map 为“焦点地图(FocusMap)”，每一个 Map 对象都有机会称为 FocusMap。IMxDocument 接口可以操作文档中所有的地图对象，如使用 IMxDocument::FocusMap 获得焦点地图，使用 IMxDocument::Maps 得到文档中的全部地图集合。

在 ArcMap 中，可以显示在 Map 上的图像分为两大类，即地理数据和图形 element(元素)，它们的共同特征是这两者都有 Geometry 属性，即拥有明确的几何形状。

地理数据包括矢量类型的要素数据、栅格数据、TIN 等表面数据和位置地址等，这些数据保存在地理数据库或数据文件中，它们是用于 GIS 分析制图的源数据；元素是另一种可以显示在 Map 上的对象，它分为两部分--图形元素和框架元素，只有前者才可以显示出来，后者则充当了更为重要的“容器”角色。在使用 ArcMap 的过程中，用户可以使用“Draw”工具栏的图形工具在 Map 上直接绘制一个矩形或者线等对象，在 PageLayout 视图中也给地图上插入指北针、图例、比例尺等图形对象，这些都是图形元素，它们被用于地图制版过程中。

在 ArcMap 中 Map 对象是由 MxDocument 对象通过 MapFrame 对象管理的，后者是一种框架元素。Map 对象具有双重身份，一方面是数据的管理容器，它可以引入地理数据和可视化元素，扮演了一个数据管理器的角色；另一方面它又可以让用户看到这些数据，即扮演了一个数据显示器的角色。当程序员把地理数据加载到 Map 对象的时候，它是数据的管理者；在刷新地图、改变显示范围的时候，它就是一个数据的显示者。

Map 组件的主要接口有：IMap、IGraphiceContainer、IActiveView、IActiveViewEvents、IMapBookmark 和 ItableCollection 等，如图 5-1 所示。

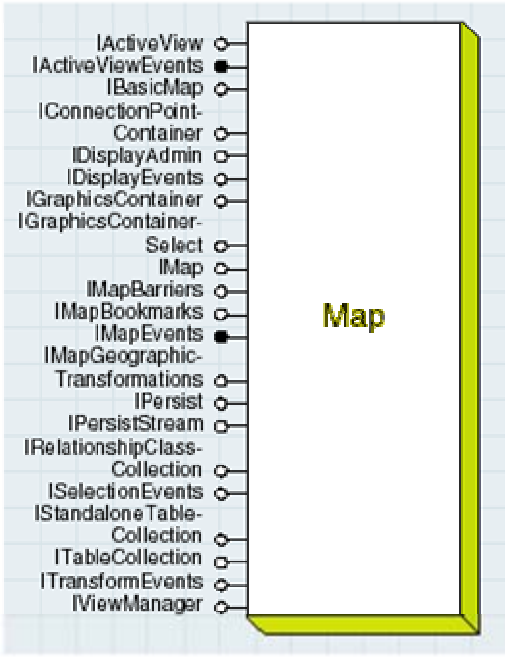


图 5-1 Map 对象的接口

5.1.1 IMap 接口

IMap 接口是开始多数 GIS 任务的起点，它主要用于管理 Map 对象中的 layer 对象、要素选择集、MapSourround 对象、标注引擎和空间参考等对象。

Map 对象可以显示地理数据，而这些地理数据都是通过某个图层引入到地图对象中的，因此可以认为 Map 对象是一个存放 layer 对象的容器，IMap 接口里面定义了大量的方法来操作它其中的图层对象。下面简要介绍几个主要的方法：

AddLayer 方法可以将一个图层对象加入 Map 对象中去，而 AddLayers 则可以一次加入一个或多个图层，当然多个图层对象必须放在一个 EnumLayer 对象中才能被引入，后者是一个图层枚举对象，可以保存多个图层的指针。

ClearLayers 可以清除 Map 中所有的图层；LayerCount 则指明这个地图对象中包含了多少个图层对象。

使用 IMap::get_layer(index)属性可以根据地图中图层的索引值得到具体的图层对象，如：

```
ILayer player;
player = pMap.get_Layer(2);
```

如图 5-2 所示，图中是 MapControl 控件的属性对话框中的图层列表，其中可以看到 3 号图层(states) 放在列表框的最下面：

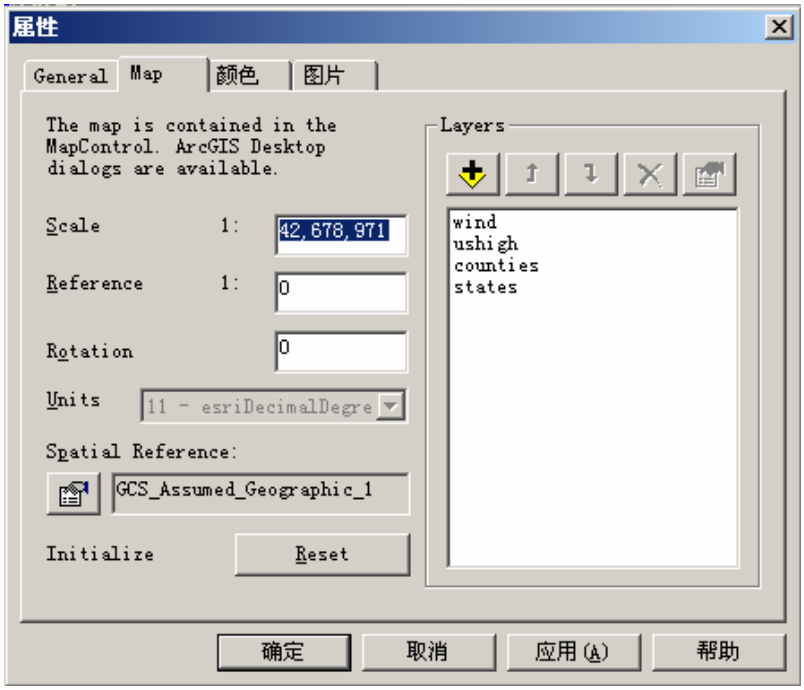


图 5-2 图层的索引

在添加地图的过程中，第一个加入 Map 对象的图层是 0 号，第二个图层就是 1 号，0 号图层放在图层对话框中最上面，但是在地图中实际放在最下面。如果在一个 Map 中放入两个 polygon 类型的图层，会发现第一次加入的图层会被后一次加入的图层所掩盖。这种层概念是数据组织的一种有效形式。

在 ArcMap 使用“Select Features”工具在地图上拖曳后，被选择的要素将会高亮显示，即使这些要素放在不同图层上的。这个工具没有什么限制，不管是点、线还是多边形，它都可以同时选择到。这些被高亮显示的要素称为“FeatureSelection”(要素选择集)，程序有多种方式可以用于选择要素，如 SelectByShape 方法就是其中之一：

SelectByShape 方法实现了 ArcMap 里面“select Features”命令的角色，它选择 Map 里面所有处于拖曳范围内的要素，无论是哪个图层，它把选择的要素添加到 Map 的 Selectionset 里面。

下面是一段选择地图元素的例子：

```
private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            IMap pMap ;
            IActiveView pActiveView;
            pMap = axMapControl1.Map;
            pActiveView = pMap as IActiveView;
            //得到一个Envelope对象
            IEnvelope pEnv ;
            pEnv = axMapControl1.TrackRectangle();
            //新建选择集环境对象
            ISelectionEnvironment pSelectionEnv ;
            pSelectionEnv = new SelectionEnvironmentClass();
            //改变选择集的默认颜色
            pSelectionEnv.DefaultColor = getRGB(110, 120, 210);
            //选择要素，将它们放入选择集
            pMap.SelectByShape(pEnv, pSelectionEnv, false);
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeoSelection, null,
null);
        }

        private IRgbColor getRGB(int r,int g,int b)
        {
```

```
        IRgbColor pColor;
    pColor = new RgbColorClass();
        pColor.Red = r;
        pColor.Green = g;
        pColor.Blue = b;
        return pColor;

}
```

注意：选择集环境对象(SelectionEnvironment)是一个独立的对象，专门用于改变构造选择集时候的某些属性。

SelectFeature 方法可以将在 Map 中获得的一个要素放到这个要素的图层的选择集中去，比如在 A 图层中得到一个要素，可以用这个方法让这个要素在 A 图层上高亮显示，使之处于选择集状态，这个方法在选择上是非常有用的。

IMap::FeatureSelection 属性可以返回 map 中被选择的要素；SelectionCount 属性可以得到 Map 对象的要素选择集中的要素的数目，ClearSelection 方法用于清除 Map 对象中的选择集。关于要素选择集的内容本书还将在 GeoDatabase 一章中做深入的讨论。

一个 Map 对象中还拥有一个 IMapSurrounds 的集合，它们是与一个地图相关的地图周边元素，如图例、指北针和比例尺等，读者可以在后面的章节中了解这个集合的详细信息。

5.1.2 IGraphicsContainer 接口

Map 对象也可以显示图形元素(Graphics Element)，它使用 IGraphicsContainer 接口来管理这些元素对象(包括图形元素和框架元素)。IGraphicsContainer 返回的是 Map 对象中处于活动状态的图形图层(graphics layer)的指针，它要么是一个基本图形图层对象(basic graphics layer)，要么是 CompositeGraphicsLayer 中的一个图层或一个 FDOGraphicsLayer 图层。

使用这个接口的方法往 Map 中添加元素，实际上是往其中的一个图形图层上添加元素对象的过程，如下面代码所示。

```
        IGraphicsContainer pGraphicsContainer;
        pGraphicsContainer = pMap as IGraphicsContainer ;
```

AddElement 方法用于将一个元素放入 Map 对象中，下面是一个添加元素的例子:

```
    //定义一个line类型的Element
```

```
ILineElement pLineEle;
pLineEle = new LineElementClass();
//设置这个Element的符号对象
pLineEle.Symbol = pSymbol;
IElement pEle;
pEle = pLineEle as IElement;
//设置元素的几何形体对象属性
pEle.Geometry = pGeo;
//将元素添加到Map对象中
pGraphicsContainer.AddElement(pEle, 0);
pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics,null, null);
```

上例将一个图形元素 pLineEle 加入到 Map 中去后，一定要刷新一次视图，否则图形元素无法即时显示在地图对象上。

IGraphicsContainer::DeleteElement 方法用于删除 Map 对象中的一个给定的元素。如果在 Map 中改变了一个元素的形状或者符号对象，用户一定希望它能够即时更新后在地图上表现出来，这个更新的方法也非常简单，使用 UpdateElement 即可。

Map 对象提供了 SelectByShape 方法用于选择要素，当然也提供了选择图形元素的方法，不过图形元素的选择更加简单，因为这些元素在没有保存到 MXD 文件前都是缓存(Cache)，只存在内存中而已。

ArcObjects 中可以使用两种方式来选择元素：

IGraphicsContainer::LocateElements 方法是使用一个点来选择元素，它需要传入一个点对象和一个容差值，Map 对象会将所有处于点容差周围的元素选择到。下面的例子中，代码在 MapControl 的 MouseDown 事件中：

```
IPoint pPt;
pPt = new PointClass();
//获得鼠标点击处的点对象
pPt.PutCoords(e.mapX, e.mapY);
IEnumElement pEnumEle;
//选择元素，其中的1为容差值
pEnumEle = pGraphicsContainer.LocateElements(pPt, 1);
IElement pElement;
//获得单个元素
pElement = pEnumEle.Next();
```

另外一个可以使用的是 LocateElementsByEnvelope 方法，它和 SelectByShape 方法类同，是使用鼠标拖曳一个矩形区域，从而选择到这个区域内

所有的元素。

5.1.3 IActiveView 接口

IActiveView 接口定义了 Map 对象的数据显示功能。这个接口管理着 ArcMap 主要的程序窗口和所有绘制图形的方法。使用这个接口的方法可以改变视图的范围，可以得到 ScreenDisplay 对象的指针，可以显示或隐藏标尺和滚动条，也可以刷新视图。

在 ArcObjects 中，有两个对象实现了这个接口，即 PageLayout 和 Map。这两个对象分别代表了 ArcMap 中两种不同的视图：数据(data)视图和布局(layout)视图，在任何一个时刻仅仅只有一个视图能够处于活动状态。IMxDocument::ActiveView 拥有当前活动视图对象的一个指针。如果 ArcMap 处于布局视图状态，那么 IMxDocument::ActiveView 返回的一个 IActiveView 对象是指向 PageLayout 对象的；如果处于在数据视图状态，那么这个属性就会返回一个当前使用的 Map 对象的指针。

图 5-3 是 IActiveView 与两种视图的关系。

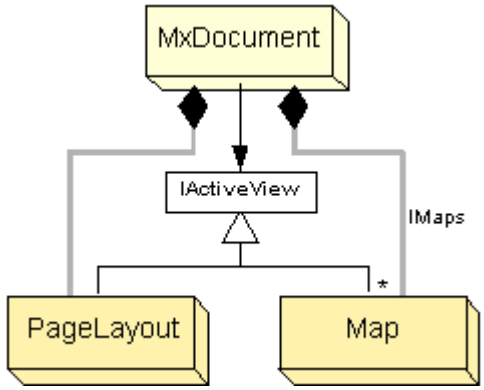


图 5-3　IActiveView 与两种视图的关系

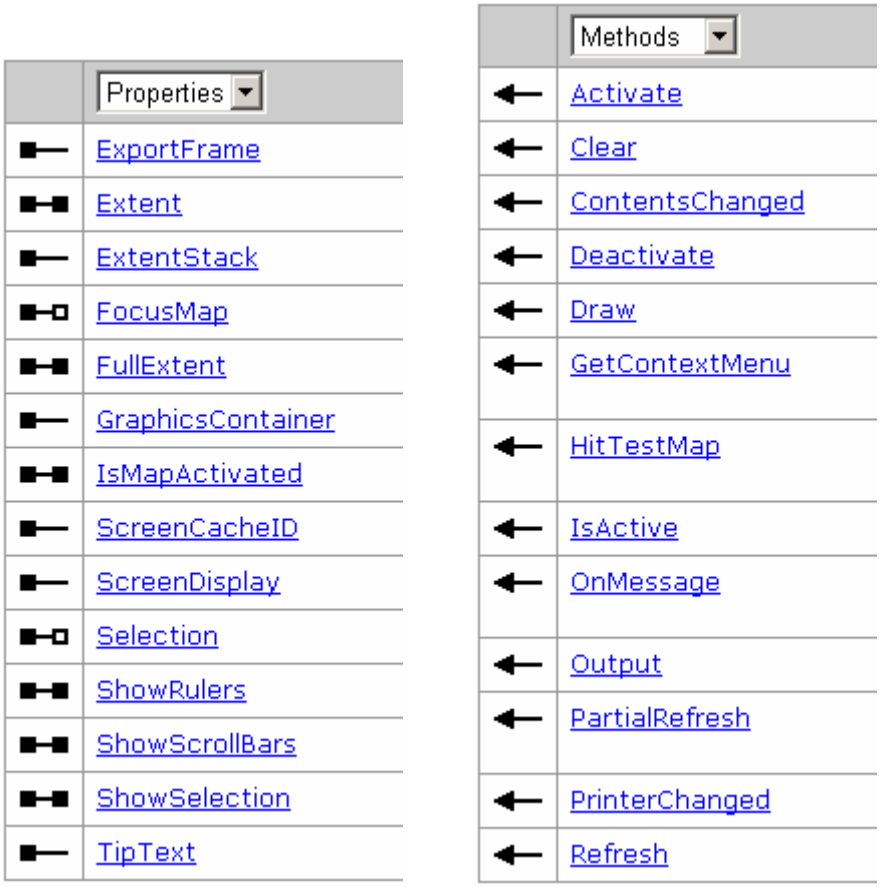


图 5-4　IActiveView 接口定义的属性、方法

尽管使用 ArcObjects 控件开发 GIS 应用时，IMxDocument::ActiveView 已经没有什么意义，但是通过控件的 IActiveView 接口仍然可以获得两个控件的 Map 对象，在使用 MapControl 控件编程的方式下获得 Map 对象：

```
IMap pMap ;
pMap = axMapControl1.Map;
IActiveView pActiveView;
//通过QI的方法得到pActiveView对象
pActiveView = pMap as IActiveView;
或 pActiveView = axMapControl1.ActiveView;
```

在 PageLayoutControl 控件中获得正在使用的地图对象的方法如下:

```
IActiveView pActiveView;
pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
或　pActiveView = axPageLayoutControl1.ActiveView;
pMap = pActiveView.FocusMap;
```

Extent 属性返回 Map 对象当前视图的范围，它是一个 Envelope 对象，即所谓的包络线对象。FullExtent 属性则可以返回视图的全图范围，因此要显示整幅地图的代码为：

```
//将地图的当前范围设置为全图范围
pActiveView.Extent = pActiveView.FullExtent;
pActiveView.Refresh();
```

ScreenDisplay 属性指向一个 ScreenDisplay 对象。每一个视图对象都有一个 ScreenDisplay 对象用于控制视图的图形绘制工作，使用这个属性会得到一个与当前正在使用的视图相关的 ScreenDisplay 对象。例如要在数据视图上绘制一个 polyline，必须先得到一个当前地图对象的 IActiveView 接口，通过 IActiveView::ScreenDisplay 获得地图的 ScreenDisplay 对象的指针。ScreenDisplay 对象将在后面的章节中详细叙述。

地图的数据显示是由一个视图来控制的，而视图的 ScreenDisplay 对象则负责绘制图形。ScreenDisplay 拥有产生任意数量的缓存(cache)的本领，缓存是一个设备相关的位图，它保存在内存中，如果 Map 对象关闭，那与之关联的缓存将全部被释放。

缓存是非常有用的，GIS 中的图形常常需要重绘，如果重绘的图形来自缓存而不是数据库，那速度将快得多。一般地，Map 产生三种类型的缓存：与所有图层相关的缓存、注记(Annotation)和图形元素、要素选择集。如果添加一个图层到 Map 对象中，会触发重绘事件，其图形的重新绘制顺序是地理数据、选择集和标注。

当一个图层的 Cached 属性为 true 时，这个图层将可以产生自己的缓存，关于缓存的具体设置，可以查阅 Map 对象的 IMapCache 接口中定义的方法和属性。

IActiveView::PartialRefresh 方法可以让视图对象使用不同的方式来局部刷新以重绘地图。这些刷新方法见下表：

状态	Map	PageLayout
esriViewBackground	Map grids	Page/snap grid
esriviewGeography	Layers	不使用
esriviewGeoSelection	Feature selection	不使用
esriViewGraphics	Labels/graphics	Graphics
esriViewGraphicSelection	Graphic selection	Element selection
esriViewForeground	不使用	Snap guides

上表说明了在什么情况下使用那一种绘制状态，如在 MapControl 控件中的做要素选择集后需要重绘时，可以使用这个方法：

```
pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeoSelection, null, null)
```

esriViewGeoSelection 在 PageLayout 控件中是不能使用的，其中的道理很简单，因为在这个控件中根本做不了要素选择集，它是专门用于地图排版的。

一个视图对象可以同时进行两种重绘状态的刷新操作，如下面的代码也是同样正确的：

```
pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography +
esriViewDrawPhase.esriViewGeoSelection, null, null)
```

IActiveView 接口还定义了 Refresh 方法，它会强制重绘视图，但是这种方法效率比较低下。在可能的情况下，用户都应该使用 PartialRefresh 方法进行 Map 视图的绘制工作。

5.1.4 IActiveViewEvents 接口

IActiveViewEvents 接口是地图对象缺省的外向接口，它让 Map 对象可以监听某些与活动视图(ActiveView)相关的事件并做出相应的反应。例如 AfterDraw、SelectionChanged 等。许多类都实现了这个接口，当然它们可以引发的事件是不同的，如 Map 对象的焦点地图变化时不会引发事件，但 PageLayout 对象却有事件产生。

5.1.5 IMapBookmark 接口

Map 对象可以管理所有的空间书签对象。使用 IMapBookmarks 接口可以得到一个已经存在的空间书签，也可以进行产生和删除空间书签等操作，一旦获得某个空间书签，可以将当前的地图范围保存在一个书签中。

5.1.6 ITableCollection 接口

ArcMap 程序除了可以添加地理数据外还能够添加纯属性表进去，在很多情况下加入属性表是为了和要素类等对象进行关联和连接。

AddTable 方法可以将一个 table 对象添加进 Map 对象；Table 属性可以依据表的标识号获得特定的属性表，这个操作类似获得图层一样，如：

```
ITable pTable ;
```

```
ITableCollection pTableCol;
pTableCol = pMap as ITableCollection;
pTable = pTableCol.get_Table(0);
```

RemoveAllTables 方法可以让 pTableCol 对象删除地图中的所有属性表，RemoveTable 可以依据表的标识号删除某个属性表，若熟悉 TableCount，则可以返回地图对象中所有的属性表的数目。

Map 对象在 ArcObjects 编程中扮演了重要的角色，在使用 MapControl 控件开发 GIS 的时候，这个对象就取代了 Application，变成了程序的“核心”和“灵魂”。

5.2 图层对象

Map 对象可以装载地理数据，这些数据是以图层的形式放入地图对象的。在 ArcObjects 中，相同类型的地理数据可以使用一个图层被放入地图。Layer 是作为一个数据的“中介”而非“容器”存在的。由于地理数据格式的多样性，因而图层类拥有众多的子类，它们使用统一的方法来操作各种不同类型的数据源。

Map 对象中显示地图需要设置空间参考(SpatialReference)，当第一个图层添加到 Map 中时，Map 对象的空间参考属性就自动设置为这个图层的空间参考，后面加入的图层无论是否已经含有空间参考都将使用 Map 对象已经设置的空间参考。

需要注意的是 Layer 对象本身没有装载数据，而仅仅是获得了数据的引用而已，是用于管理数据源的连接。在 ArcObjects 中，地理数据始终是保存在 GeoDatabase 或者地理文件中的。ArcMap 中也可以在一个要素类上新建一个图层文件，即 lyr 文件，这个文件也是仅仅获取了地理数据的硬盘位置而已，而没有拥有数据。

5.2.1 ILayer 接口

ILayer 是所有图层类都实现了的一般接口，它定义了所有图层的公共方法和属性，如 Name 属性可以返回图层名称，MaximumScale 和 MinimumScale 是两个可读写属性，它们用于显示和设置图层可以出现的最大和最小尺寸。

ILayer::ShowTips 属性用于指示当鼠标放在图层某个要素上的时候，是否会出现 Tip(提示)，而 TipText 确定图层提示显示的区域。SpatialReference 属性用于设置图层的空间参考，这个对象是从 Map 对象中传入的，这对于地理数据是非常重要的。

下面是展示一个显示要素图层(FeatureLayer)的 Tip 的代码例子：

在 C#.NET 里面给一个图层显示 Tip 和 VB 的代码相差很大，VS.NET 中专门有一个 ToolTip 控件可以供使用，显示提示文本需要先在窗体中放入一个 ToolTip 控件，下面是在 MapControl 控件的 MouseMove 事件中：

```
private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            IFeatureLayer pFeatLyr ;
            //得到要显示tip的图层
            pFeatLyr = axMapControl1.Map.get_Layer(1) as IFeatureLayer ;
            //DisplayField属性可以设置“p显示的内容是哪个字段的
            pFeatLyr.DisplayField = "pop1990";
            //图层可以显示tip
            pFeatLyr.ShowTips = true;
            string pTip;
            pTip = pFeatLyr.get_TipText(e.mapX, e.mapY, pActiveView.FullExtent.Width /
100);
            //设置tooltip
            toolTip1.SetToolTip(axMapControl1, pTip);
        }
```

5.2.2 要素图层

要素数据是 GIS 中最常使用的数据类型之一，它可以用于表示离散矢量对象的信息，而承载要素数据的要素图层(FeatureLayer)是本节研究的重点。本节会介绍要素图层的几个重要接口定义的方法和属性。

5.2.2.1IFeatureLayer 接口

IFeatureLayer 接口用于管理要素图层的数据源，即要素类(Featureclass)。IFeatureLayer::DataSourceType 属性返回要素图层的数据源的类型，FeatureLayer 的数据源有：

- Personal Geodatabase
- SDE
- Shapefile
- ArcInfo or PC ArcInfo Coverage (annotation)
- ArcInfo or PC ArcInfo Coverage (point)
- ArcInfo or PC ArcInfo Coverage (line)

- ArcInfo or PC ArcInfo Coverage (polygon)
- Edge
- CAD (annotation)
- CAD (point)
- CAD (line)
- CAD (polygon)

实现了 IFeatureLayer 接口的图层类有：

- CadAnnotationLayer
- CadFeatureLayer
- CoverageAnnotationLayer
- DimensionLayer
- FDOGraphicsLayer
- FeatureLayer
- GdbRasterCatalogLayer
- TemporalFeatureLayer (TrackingAnalyst)

在 C#.NET 中新建一个要素图层对象必须引入 GeoDatabase 库，然后使用下面的代码：

```
IFeatureLayer pFeatLyr;
pFeatLyr = new FeatureLayerClass();
```

IFeatureLayer::FeatureClass 属性返回要素图层使用的要素类。关于要素类的内容，会在 GeoDatabase 一章中详细介绍。

如果需要对要素图层进行查询，可以使用 IFeatureLayer::search 方法，它将返回一个 ICursor 类型的对象。如在一幅世界地图中寻找人口多于 5000 万的国家或者是国土面积大于 100 万平方千米的国家，都可以使用 Search 方法实现。

Search 方法需要传入两个参数，一个是过滤器，它是一个 IQueryFilter 类型的对象；另一个参数是布尔值 Recycling，用于说明返回的要素游标是否循环。一般地，如果是绘制图形或者仅仅读取数据，应该设置这个参数为 true，如果选择出来的要素会被更新，那就应该设为 false。

需要注意的是这个方法不能使用在关联(join)字段上，如果一个要素图层与其他图层或属性表有任何形式的关联关系，则需要使用 IGeoFeatureLayer::SeachDisplayFeatures 方法。

下面给出一个查找要素的方法，它需要传入过滤参数和图层：

```
private void searchFeatures(string sqlfilter,IFeatureLayer pFeatureLayer)
{
```

```
            IFeatureLayer pFeatLyr;
            //得到要进行查询的图层
            pFeatLyr = pFeatureLayer;
            //定义一个过滤器对象
            IQueryFilter pFilter;
            pFilter = new QueryFilterClass();
            //'添加过滤参数sqlfilter，如“area>100000”
            pFilter.WhereClause = sqlfilter;
            IFeatureCursor pFeatCursor;
            pFeatCursor = pFeatLyr.Search(pFilter, true);
            IFeature pFeat;
            pFeat = pFeatCursor.NextFeature();
            while( pFeat != null)
            {
                pFeat = pFeatCursor.NextFeature();
                if (pFeat != null)
                {
                    ISimpleFillSymbol pFillsyl;
                    pFillsyl = new SimpleFillSymbolClass();
                    pFillsyl.Color = getRGB(220, 100, 50);
                    object oFillsyl ;
                    oFillsyl = pFillsyl;
                    IPolygon pPolygon;
                    pPolygon = pFeat.Shape as IPolygon;
                    //使用FlashShape来使要素闪烁
                    axMapControl1.FlashShape(pPolygon, 15, 20, pFillsyl);
                    //为了使被选择的要素显示出来，使用一个polygon来显示
                    axMapControl1.DrawShape(pPolygon, ref oFillsyl);
                }
            }
        }
```

在使用这个方法对一个要素图层进行查询时，符合条件的要素将逐个闪烁，然后呈橘红色显示出来。如果仅仅为了显示要素，这是一种不错的方法。

5.2.2.2IGeoFeaturelayer 接口

IGeoFeaturelayer 接口继承了 ILayer 和 IFeatureLayer 两个接口，用于控制要

素图层中与地理相关的内容，例如要素的着色和标注等功能，CadFeatureLayer、FeatureLayer、GdbRasterCatalogLayer 三个图层类实现了该接口。

IGeoFeaturelayer 有一个 SearchDisplayFeature 方法，使用这个方法只显示符合查询要求的要素，其他的要素都会消失掉，这个方法经常在有关联关系的要素类使用。Renderer 属性用于设置图层的着色对象；DisplayAnnotation 属性可以设置要素图层是否出现标注，当它为 true 的时候，用户可以在这个要素图层上依据要素类的某个字段进行标注。

5.2.2.3IGeoDataset 接口

IGeoDataset 接口仅有两个属性，它们用于管理地理要素集。IGeoDataset::Extent 可以返回当前数据集的范围，是一个 IEnvelope 类型的对象；IGeoDataset::SpatialReference 属性则可以让用户获得这个数据集的空间参考。

ArcObjects 中，有 40 多个类实现了 IGeoDataset 接口，如 CadAnnotationLayer (Carto)、CadFeatureLayer (Carto)、CadLayer (Carto)、CompositeGraphicsLayer (Carto)、CoverageAnnotationLayer (Carto)、DimensionLayer (Carto)、FDOGraphicsLayer (Carto)、FeatureClass、FeatureDataset、FeatureLayer (Carto)、NetworkDataset、NetworkLayer (Carto)、RasterLayer (Carto)、Tin、TinLayer (Carto)、Topology、TopologyLayer (Carto)等。

5.2.2.4IFeatureSelection 接口

IFeatureSelection 接口负责管理一个图层中的要素的选择集的方法和属性。在 ArcObjects 中，实现 IFeatureSelection 接口的类有：CadFeatureLayer、CoverageAnnotationLayer、DimensionLayer、FDOGraphicsLayer、FeatureLayer、GdbRasterCatalogLayer、TemporalFeatureLayer (TrackingAnalyst)。

IFeatureSelection 接口的 Add 方法可以把本图层上的一个要素添加到图层的选择集中；SelectFeatures 方法则可以使用一个过滤器把符合要求的要素放入图层的选择集中；Clear 方法可以用于图层所有的要素选择集。

谈到要素选择集，需要明白在要素图层中 Select 要素和 Search 要素的巨大差异。要素图层在 Search 方法中返回的 ICursor 对象，它仅仅是一个指向要素的指针；而如果使用 IFeatureSelection::SelectFeatures 方法，则符合要求的要素会被标记为“被选择的要素集”，这个选择集是一个 ISelectionSet 对象，可以用 SelectionSet 方法返回，如：

```
IFeatureSelection pSel;
```

```
            ISelectionSet pSelectionSet;
        //QI，pFeatureLayer是IFeatureLayer类型
            pSel = pFeatureLayer as IFeatureSelection;
            //获得图层的要素选择集
            pSelectionSet = pSel.SelectionSet;
```

下面已知一个要素图层和选择条件，寻找出符合要求的要素并闪烁显示的例子，这个例子和 IFeatureLayer 接口例子中的内容相似，但是这里使用的是选择集。

```
        private void searchSelection(string sqlfilter,IFeatureLayer pFeatureLayer)
        {
            IQueryFilter pFilter;     //做一个过滤器
            pFilter = new QueryFilterClass();
            pFilter.WhereClause = sqlfilter;     //如〝area>100000〞
            IFeatureSelection pFeatureSelection;
            pFeatureSelection = pFeatureLayer as IFeatureSelection;        //QI
            pFeatureSelection.SelectFeatures(pFilter,
esriSelectionResultEnum.esriSelectionResultNew, true);
            //将选择集添上颜色
            IRgbColor pColor;
            pColor = new RgbColorClass();
            pColor.Red = 220;
            pColor.Green = 112;
            pColor.Blue = 60;
            pFeatureSelection.SelectionColor = pColor;
    axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewGeoSelection,null,null) ;
            ISelectionSet pFeatSet;   //新建一个Selectionset对象
            pFeatSet = pFeatureSelection.SelectionSet;
            //在要素选择集中也可以使用search方法得到其中的要素
            IFeatureCursor pFeatCursor;
            ICursor pCursor;
            //使用要素游标去获取单个要素
            pFeatSet.Search(null, true, out pCursor);
            pFeatCursor = pCursor as IFeatureCursor;
            IFeature pFeat;
            pFeat = pFeatCursor.NextFeature();
            while( pFeat != null )
            {
                if (pFeat != null)
```

```
                {
                    ISimpleFillSymbol pFillsyl2;
                    pFillsyl2 = new SimpleFillSymbolClass();
                    pFillsyl2.Color = getRGB(220, 60, 60);
                    //被选中的要素闪烁
                    axMapControl1.FlashShape(pFeat.Shape, 15, 20, pFillsyl2);
                }
                pFeat = pFeatCursor.NextFeature();
            }
        }
```

上面的例子在显示的时候除了要素由于被放入选择集而高亮显示外，还会闪烁一段时间。

这种选择的方法仍然很繁琐，下面是一种更加简单的方式，它可以让被选中的要素添加到它自己图层的 SelectionSet 中。pCurrentLayer 是当前正在进行操作的图层，如下例：

```
        private void searchSelection2(string sqlfilter, IMap pMap, IFeatureLayer
pCurrentLayer)
        {
            IFeatureLayer pFeatureLayer;
            pFeatureLayer = pCurrentLayer;
            IQueryFilter pQueryFilter;     //做一个过滤器
            pQueryFilter = new QueryFilterClass();
            pQueryFilter.WhereClause = sqlfilter;
            //查询出一系列符合要求的要素游标
            IFeatureCursor pCursor;
            pCursor = pFeatureLayer.Search(pQueryFilter, true);
            IFeature pFeat;
            pFeat = pCursor.NextFeature();
            while( pFeat != null)
            {
                //通过IMap的SelectFeature将要素放到某个图层的Selectionset中去
                pMap.SelectFeature(pCurrentLayer, pFeat);
                pFeat = pCursor.NextFeature();
            }
            IActiveView pActiveView = pMap as IActiveView ;
            pActiveView.PartialRefresh
(esriViewDrawPhase.esriViewGeoSelection,null,null);
```

```
    }
```

测试代码：

```
private void mnusearchSelection2_Click(object sender, System.EventArgs e)
    {
        IMap pMap;
        pMap = axMapControl1.Map;
        IFeatureLayer pFeatureLayer;
        pFeatureLayer = pMap.get_Layer(1) as IFeatureLayer;
        string sqlfilter = "POP1990 > 5000000";
        searchSelection2(sqlfilter, pMap,   pFeatureLayer);
    }
```

ArcMap 里面的 Selections 有两种：Feature Selections(要素选择集)和 Element Selections(元素选择集)，这两种都实现了 ISelection 接口。在返回值的时候，前者使用 IEnumFeature，后者常常使用 IEnumElement，它们都是枚举对象。每一个 PageLayout 对象和 Map 对象都有自己的选择集对象。差别是前者只有元素选择集，而后者却拥有元素选择集和要素选择集两种。

如果是通过 IActiveView::Selection 获得 Selection，那获得的 Selection 不是 FeatureSelection 就是 ElementSelection，每次只能是一种，具体是哪一种取决于视图 IActiveView 的类型。

关于选择集的本质，将在本书的 GeoDatabase 章节中进行介绍。

5.2.2.5IFeatureLayerDefinition 接口

IFeatureLayerDefinition 接口定义了 CreateSelectionLayer 方法，可以将一个图层选择集中的要素转换为一个单独的要素图层。

CreateSelectionLayer 是从一个已经存在的要素图层的选择集中产生一个新的要素图层。它要求四个参数：LayerName 定义了新图层的名称，如果希望使用当前图层的要素选择集，需要设 useCurrentSelection 为 TRUE；joinTableNames 是一个可能与当前图层关联的表的名称，使用这个方法同样也可以将关联表的数据放入新的图层中去；DefinitionExpression 可以用于设定一个选择过滤，将要素选择集中符合条件的要素放入新的图层。

下面给出从要素图层的选择集产生新的要素图层的代码：

```
    private void CreateSelectionLayer()
    {
```

```
                IFeatureLayer pFeatLyr;
                IMap pMap = axMapControl1.Map;
                IActiveView pActiveView = pMap as IActiveView;
                pFeatLyr = pMap.get_Layer(1) as IFeatureLayer;
                IQueryFilter pQueryFilter ;
                pQueryFilter = new QueryFilterClass();
                pQueryFilter.WhereClause = "area > 100000";
                IFeatureSelection pFeatSel;
                pFeatSel = pFeatLyr as IFeatureSelection;
                pFeatSel.SelectFeatures(pQueryFilter,
esriSelectionResultEnum.esriSelectionResultNew, false);
                IFeatureLayerDefinition pFeatLyrDef;
                pFeatLyrDef = pFeatLyr as IFeatureLayerDefinition;
                pFeatLyrDef.DefinitionExpression = "pop1999 > 1000000";
                //产生新的图层
                IFeatureLayer pnewfeat;
                pnewfeat = pFeatLyrDef.CreateSelectionLayer("new" + pFeatLyr.Name , true, "",
"");
                pFeatSel.Clear();
                pMap.AddLayer(pnewfeat);
                pActiveView.Refresh();
            }
```

新加入的图层是基于 Layer(1)图层中面积大于 100000 的要素选择集的基础上，再次选择出 POP1999 值大于 1000000 的要素而构造成的。

5.2.2.6ILayerFields 接口

尽管要素图层的数据是保存在要素类中，获得要素类的字段需要从要素类着手，但是 ArcObjects 也提供了 ILayerFields 接口，可以直接获取一个要素图层的要素类字段结构。

关于要素类的字段，可以参看本书 GeoDatabase 一章的内容。

5.2.2.7IIdentify 接口

IIdentify 接口定义了获得要素图层单个要素的属性的捷径方法。它有一个 Identify 方法，返回一个 IArray 数组对象。

下面是一个代码段例子，它使用了 IIdentify、IArray、IFeatureIdentifyObj 和 IIdentifyObj 对象，代码需要是在一个 MouseDown 事件中，它可以用于模拟 ArcMap 程序工具栏中的 Identify 命令：

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            IIdentify pIdentify ;
            IPoint pPoint;
            IArray pIDArray ;
            IFeatureIdentifyObj pFeatIdObj;
            IIdentifyObj pIdObj;
            IMap pMap = axMapControl1.Map ;
            pIdentify = pMap.get_Layer(1) as IIdentify;
            pPoint = new PointClass();
            pPoint.PutCoords(e.mapX, e.mapY);
            pPoint.SpatialReference = pMap.SpatialReference ;
            //这个方法是使用一个点对象来选择要素
            pIDArray = pIdentify.Identify(pPoint);
            //获得FeatureIdentifyObject对象
            if ( pIDArray != null)
            {
                pFeatIdObj = pIDArray.get_Element(0) as IFeatureIdentifyObj;
                pIdObj = pFeatIdObj as IIdentifyObj;
                //让被选择的要素闪烁
                pIdObj.Flash(axMapControl1.ActiveView .ScreenDisplay);
                MessageBox.Show("Layer：" + pIdObj.Layer.Name +
System.Environment.NewLine  + "Feature：" + pIdObj.Name);
            }
            else
                MessageBox.Show("没有要素被指定.");
        }
```

如果需要获得更丰富的信息，可以使用 IdentifyDialog 对象，参看后面章节。

5.2.3 往地图中加入 CAD 文件

CAD 数据是 GIS 中常用的一种地理数据，GIS 的数据很多都是从 DWG 文件转换获取的。Map 对象可以载入多种格式的地理和非地理数据，其中也包括 DWG 文

件，这一节介绍将 CAD 文件载入 Map 对象的方法。

在 Map 对象看来，DWG 文件是两种不同形式的混合体：一是要素图层，保存的是矢量数据；另外一种是看做栅格图像，可以作为地图背景使用。对于前者，ArcObjects 使用与 Featurelayer 一样的方法来管理，对于后一种类型的数据则是采用 cadLayer 对象来管理。

当一个 DWG 文件被看做是要素图层的时候，它表现为四种类型的要素类，即点、线、多边形和标注。这种区分并不是按照 DWG 文件本身的图层号来做的，而是简单地考虑 DWG 文件中的几何形体对象的类型。因此，如果一个 DWG 文件中有多个图层，且这些图层都是线类型，那么它们将会无区别地合并在一个线要素图层中。点、线和多边形作为要素数据时，可以使用 FeatureLayer 对象，但 CAD 文件中的文字标注需要使用一个专门的标注图层。

- **CadLayer 对象**

CadDrawingLayer 是将一个 CAD 数据当作栅格类型而使用的数据图层，它可以作为地图的背景而显示，但是不能做地理分析。这个对象实现了 ICadLayer 和 ICadDrawingLayers 两个接口，它们定义了将 CAD 文件作为栅格数据处理的方法。

下面的代码是将一个 CAD 文件添加到地图对象中去：

```
private void   AddCADLayer()
{
    //使用专门的workspacefactory用于打开一个CAD文件的工作空间
    IWorkspaceFactory pCadWorkspaceFactory;
    pCadWorkspaceFactory = new CadWorkspaceFactoryClass();
    IWorkspace pWorkspace;
    pWorkspace =
pCadWorkspaceFactory.OpenFromFile(@"C:\arcgis\ArcTutor\Editor\ExerciseData\EditingCAD",
0);
    ICadDrawingWorkspace pCadDrawingWorkspace ;
    pCadDrawingWorkspace = pWorkspace as ICadDrawingWorkspace;
    //定义并获得CAD文件的数据集
    ICadDrawingDataset pCadDataset;
    pCadDataset =
pCadDrawingWorkspace.OpenCadDrawingDataset("PARCELS.DWG");
    //定义一个CAD图层
    ICadLayer pCadLayer ;
    pCadLayer = new CadLayerClass();
    pCadLayer.CadDrawingDataset = pCadDataset;
```

```
        //加入图层到Map对象中去
        axMapControl1.AddLayer(pCadLayer,0);
    }
```

- **Cad 文件作为 Feature 对象**

CAD 文件作为要素数据被读取的时候，使用的方法和 IFeaturelayer 是一样的，需要注意的是它是作为一个要素数据集的形式而存在的。一个 DWG 文件在 Map 看来是一个要素数据集，其中的类简单分为点、线、面和文字标注四种类型，其中文字需要使用标注图层而不是要素图层。

下面的方法是把 CAD 文件当作要素数据集来获取，然后加入 Map 对象中。

```
        private void AddCADFeatures()
        {
            IWorkspaceFactory pCadWorkspacefactory ;
            pCadWorkspacefactory = new CadWorkspaceFactoryClass();
            IFeatureWorkspace pWorkspace;
            pWorkspace =
pCadWorkspacefactory.OpenFromFile(@"C:\arcgis\ArcTutor\Editor\ExerciseData\EditingCAD",
0) as IFeatureWorkspace;
            //打开一个要素数据集
            IFeatureDataset pFeatDataset;
            pFeatDataset = pWorkspace.OpenFeatureDataset("PARCELS.DWG");
            //pFeatClassContainer可以管理pFeatDataset中的每个要素集
            IFeatureClassContainer pFeatClassContainer;
            pFeatClassContainer = pFeatDataset as IFeatureClassContainer;
            IFeatureClass pFeatClass;
            IFeatureLayer pFeatLayer;
            int i;
            //对CAD文件中的要素集进行遍历处理
            for( i = 0; i<= pFeatClassContainer.ClassCount - 1;i++)
            {
                pFeatClass = pFeatClassContainer.get_Class(i);
                if (pFeatClass.FeatureType ==
esriFeatureType.esriFTCoverageAnnotation)
                {
                    //标注类型，必须设置为单位的标注图层
                    pFeatLayer = new CadAnnotationLayerClass();
                }
```

```
            else
            {
                //点线面类型
                pFeatLayer = new FeatureLayerClass();
            }
            pFeatLayer.Name = pFeatClass.AliasName;
            pFeatLayer.FeatureClass = pFeatClass;
            axMapControl1.AddLayer(pFeatLayer,0);
        }
    }
```

除此以外，如果需要显示 CAD 文件中的文字，则可以使用 CadAnnotationLayer 对象，它作为一个标注图层显示 CAD 文件中所有的标注文本信息，这个对象的详细信息读者可以查阅开发帮助的相关内容。

5.2.4 TIN 图层

TIN(不规则三角网)代表了连续的表面，类似地表高程或者温度梯度都可以使用 TIN 来表示。一般通过将一个 TIN 文件设置不同的阴影颜色，使之看起来具有三维的感觉。如果是一幅地形图，使用渲染着色，可以很清楚地看到山脊、峡谷和山坡，也可以在地图上方便地看到 TIN 的三个属性：坡度、方位和高程。

如果不进行着色，能够看到很多三角网格构成的表面图形。TinLayer 是基于 TIN 的图层，在 ArcGIS 中经常使用它进行渲染着色以进行显示。

ITinLayer 接口定义了 TIN 如何在图层上显示的方法和属性。

ITinLayer::Dataset 显示了 Tin 图层的数据源，而 ITinLayer::DisplayField 是 TinLayer 图层的主要显示字段。

ITinLayer::AddRenderer 方法可以加入一个着色对象；ClearRenderers 可以清除图层中所有的着色对象，而 GetRenderer 可以用一个索引值获得某个着色对象。在一幅 TIN 图中，有多种着色方法，ITinLayer::RendererCount 则可以返回这些着色对象的数目。

下面是一个获得 TIN 着色对象的代码例子

```
ITinLayer pTinLayer;
pTinLayer = pMap.get_Layer(0) as ITinLayer;
ITinRenderer pTinRend;
for( int i = 0; i <= pTinLayer.RendererCount - 1;i++)
```

```
{
    pTinRend = pTinLayer.GetRenderer(i);
    MessageBox.Show(pTinRend.Name);

}
```

在 MapControl 控件中所看到的 TIN 图像始终的是静止的，用户没有办法通过不同的角度来看到三维场景，这种三维是虚假的。但是在 ArcGIS Engine 的 ArcScene 控件中，可以构筑一个三维 GIS。

5.2.5 GraphicsLayer

Map 对象上可以绘制图形元素，如在一个 MapControl 控件上绘制一个圆、多边形等图形都可以，这些图形对象也是放置在一个图层上的，这个图层就是 GraphicsLayer 对象。

GraphicsLayer 对象是用于管理与 Map 相关的图形元素的，它是一个抽象类，拥有两个子类：CompositeGraphicsLayer 和 FDOGraphicsLayer，它们分别实现了 IGraphicsLayer、IGraphicsLayer 和 ISelectionEvents 等接口。

每一个 Map 对象都管理着一个 CompositeGraphicsLayer 对象，它其实是一个图形图层(graphics layer)的集合。这个集合中有一个缺省的图形图层称为“基本图形图层”，地图对象可以通过 IMap::BasicGraphicsLayer 直接获得这个图层。地图的基本图形图层不能从 CompositeGraphicsLayer 对象中删除，但是可往集合中添加和删除一个新的图形图层。如果使用遍历 CompositeGraphicsLayer 中的图层数目，也不会显示出这个基本图形图层。

FDOGraphicslayer 是一个与要素相关的标注图层，在 ArcMap 中给要素进行标注有两种方式，一种是使用 Label 方法，通过鼠标选取要素，逐个给这些要素添加标注文本；另外一种是使用 Annotation 方法，它可以对要素图层进行自动注记，并且这些注记的文本信息都可以保存在数据库中。使用 Label 方法进行标注时，标注的文本将放置在 FDOGraphicsLayer 图层上。

由于 GraphicsLayer 需要管理图形元素，因而它实现了 IGraphicsContainer 接口用于管理这个图层内部的图形元素。关于这个接口的方法已经在前面的 Map 对象中进行了详细的讲述，这里就不再讨论了。

5.3 ScreenDisplay 对象

每一个视图都有一个 ScreenDisplay 对象，用于控制视图中的图形绘制。

ScreenDisplay 是与一个窗体相联系的显示设备，除了管理窗体屏幕的显示属性外，它也管理了发生在显示背后的对象和行为，如缓存和视图屏幕变化等。

很多对象都由 ScreenDisplay 去管理与它们相关的可视化窗体，只要是它们需要显示地图，如每一个活动的视图：Map、PageLayout 或 MapInsertWindow 等。

要得到一个 ScreenDisplay 对象非常简单，如果是 Map 或 PageLayout 对象，一般可以通过 IActiveView::ScreenDisplay 属性获得；如果是 MapInsertWindow 对象，则是使用 ILensWindow::ScreenDisplay 属性获得。

ScreenDisplay 是一个组件类，它可以独立产生，这意味着在建立一个自定义的活动视图时，可以产生一个新的 ScreenDisplay 对象与之相关联。

每一个 ScreenDisplay 对象都拥有一个 DisplayTransformation 对象用于设备单位和地图单位的转换，这个对象可以通过 IDisplay::DisplayTransformation 属性得到。每一个 DisplayTransformation 都与一个 Map 相关，它拥有地图的空间参考属性。使用 MapControl 控件编程时，使用这个对象的机会比较少，其原因在于控件的鼠标事件的参数中可以直接得到鼠标在地图上的坐标单位。如 e.MapX 和 e.MapY 可以返回鼠标在控件上的坐标，而 e.x、e.y 则返回鼠标在屏幕设备上的坐标，两个坐标可以通过 DisplayTransformation 进行转换。

ScreenDisplay 类主要实现了 IScreenDisplay 接口。IScreenDisplay:: AddCache 方法可以将一个新的 cache(缓存对象)放进 ScreenDisplay 对象中，而 IScreenDisplay:: CacheCount 则返回 Cache 的数目。

DrawPoint、DrawPolyline、DrawPolygon 和 DrawRectangle 是在地图控件中经常使用的用于绘制几何对象的方法，除此以外还有一个 DrawText 可以用于在视图上绘制字符对象，所有的这些 Draw 方法在使用前都必须先使用 StartDrawing 方法，在绘制结束后需要使用 FinishDrawing 方法。使用这些 Draw 方法都需要传给要绘制的几何形体对象。

下面是一个使用 DrawPolyline 方法绘制 polyline 的例子，这个例子必须放在 MapControl 控件的 MouseDown 事件中：

```
//新建一个线符号对象
  private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
  {
      IActiveView pActiveView = axMapControl1.ActiveView;
      IScreenDisplay screenDisplay = pActiveView.ScreenDisplay;
      ISimpleLineSymbol lineSymbol = new SimpleLineSymbolClass();
      IRgbColor rgbColor = new RgbColorClass();
```

```
                rgbColor.Red = 255;
                lineSymbol.Color = rgbColor;
//              IRubberBand rubberLine = new RubberLineClass();
//              IPolyline pLine = (IPolyline)rubberLine.TrackNew(screenDisplay,
(ISymbol)lineSymbol);
                IPolyline pLine = axMapControl1.TrackLine() as IPolyline;
                screenDisplay.StartDrawing(screenDisplay.hDC,
(short)esriScreenCache.esriNoScreenCache);
                screenDisplay.SetSymbol((ISymbol)lineSymbol);
                screenDisplay.DrawPolyline(pLine);
                screenDisplay.FinishDrawing();
                }
```

在 MapControl 控件中使用 Pan 方法可以非常容易地移动地图。同样地，ScreenDisplay 也有移动地图的方法，而且还不止一种，使用 TrackPan 方法，用户能够直接在控件的 MouseDown 事件中移动地图。另外一种复杂的方法需要连续使用到 MouseDown、MouseMove 和 MouseUp 三个事件，这三个事件中分别使用 panStart、panMoveto 和 panStop 方法。

旋转地图对于 ScreenDisplay 对象而言也是轻而易举的事情，它同样也提供了 TrackRotate 方法和需要使用三个鼠标事件的 RotateStart、RotateMoveto 和 RotateStop 方法。

鼠标的 MouseDown 事件，开始旋转：

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            IActiveView pActiveView = axMapControl1.ActiveView;
            IEnvelope pAVExtent = pActiveView.Extent ;
            IPoint point = new PointClass();
            //Set the coordinates of current mouse location
            point.PutCoords(e.mapX, e.mapY);
            IPoint m_Point;
            m_Point = new PointClass();
            //Set the coordinates of the center of the current extent
            m_Point.X = pAVExtent.XMin + pAVExtent.Width / 2;
            m_Point.Y = pAVExtent.YMin + pAVExtent.Height / 2;
            pActiveView.ScreenDisplay.RotateStart(point, m_Point);
            //Set the mouse pointer to use the mouse icon
```

```
            axMapControl1.MousePointer = esriControlsMousePointer.esriPointerCustom;
        }
```

鼠标的 MouseMove 事件，确定地图被移到何处：

```
        private void axMapControl1_OnMouseMove(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseMoveEvent e)
        {
            IPoint point = new PointClass();
            //Set the coordinates of current mouse location
            point.PutCoords (e.mapX , e.mapY);
            //Rotate the display based upon the current mouse location
            axMapControl1.ActiveView.ScreenDisplay.RotateMoveTo(point);
            //Draw the rotated display
            axMapControl1.ActiveView.ScreenDisplay.RotateTimer();
        }
```

下面是鼠标的 MouseUp 事件：

```
        private void axMapControl1_OnMouseUp(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseUpEvent e)
        {
            //Get rotation angle
            double dRotationAngle =
axMapControl1.ActiveView.ScreenDisplay.RotateStop();
            //Rotate the MapControl's display
            axMapControl1.Rotation = dRotationAngle;
            //Refresh the display
    axMapControl1.ActiveView.PartialRefresh(ESRI.ArcGIS.Carto.esriViewDrawPhase.esriVi
ewGeography,null,null);
        }
```

ScreenDisplay 对象的功能是非常强大的，在 ArcObjects 中的任何窗体需要绘制图形时，都不可避免地使用到这个对象。在使用 MapControl 控件编程的时候，读者经常会使用 FlashShape 方法来闪烁要素，下面的代码将演示自己进行要素闪烁的过程。

下面的 FlashPolygon 方法只对 Polygon 类型要素有用，它需要输入四个参数，第一个是 MapControl 控件的 ScreenDisplay 对象，第二个参数是需要闪烁的要素几何属性，第三个是闪烁次数，第四个参数是一次闪烁时间。

```
        private void FlashPolygon(IScreenDisplay pDisplay,IGeometry pGeometry,int
nTimer,int time)
```

```
{
    ISimpleFillSymbol pFillSymbol;
    ISymbol pSymbol;
    IRgbColor pRGBColor;
    pRGBColor = new RgbColorClass();
    pRGBColor.Green = 128;
    pFillSymbol = new SimpleFillSymbolClass();
    pFillSymbol.Outline = null;
    pFillSymbol.Color = pRGBColor;
    pSymbol = pFillSymbol as ISymbol;
    pSymbol.ROP2    = esriRasterOpCode.esriROPNotXOrPen;
    int i;
   pDisplay.StartDrawing(0, (short)esriScreenCache.esriNoScreenCache);
    pDisplay.SetSymbol(pSymbol);
    for (i = 0;i<= nTimer - 1;i++)
    {
        pDisplay.DrawPolygon(pGeometry);
        System.Threading.Thread.Sleep(time);
    }
}
```

读者可以在学习了 GeoDatabase 章节后将这个过程改写为更普遍的闪烁过程。

5.4 地图排版

PageLayout 对象对应于 ArcMap 的布局视图，在打开 ArcMap 的时候，文档对象会自动新建一个 PageLayout。文档对象可以通过 IMxDocument::PageLayout 来获得它的 PageLayout 对象，这个属性是可读写的，它意味着用户既可以新建一个 PageLayout，也可以获取文档中自动产生的 PageLayout。

为了更好地显示地图以便打印和输出，PageLayout 会自动产生一些对象来帮助它进行地图的修饰：SnapGuides、SnapGrid、RulerSettings 和 Page。这些对象之间的关系如图 5-5 所示。

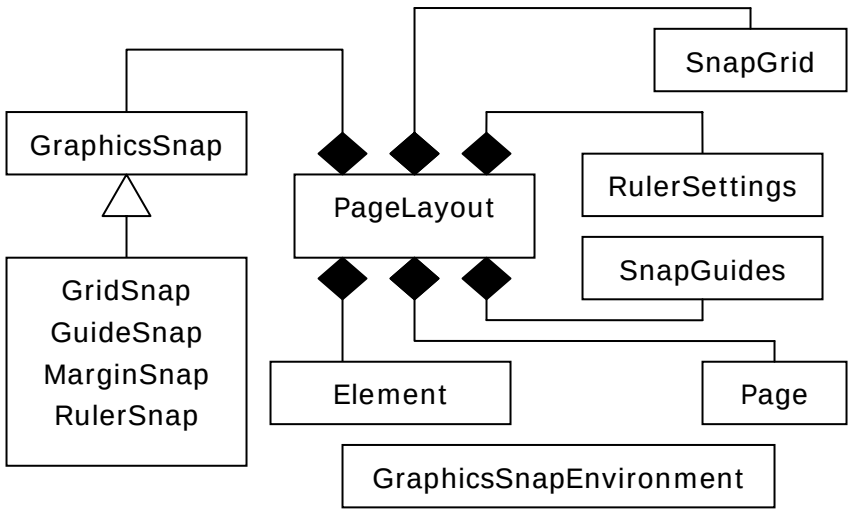


图 5-5 PageLayout 及相关对象结构关系

5.4.1 PageLayout 对象

PageLayout 和 Map 这两个对象看起来非常相似，它们都是视图对象，可以显示地图；都是图形元素的容器，可以容纳图形元素(Graphics Element)。它们虽然都是图形元素的容器，但是所能够保存的图形类型却是有差别的。PageLayout 除了保存图形元素外，还可以保存诸如 MapFrame 的框架元素(Frame Element)。

在 ArcMap 中，如果在布局视图中没有一个地图被激活使用，那这个地图中的图形元素都将保存在 PageLayout 中；如果有一个地图处于被激活状态，那么图形元素将被加入到焦点地图(Focus Map)对象中去。

在 ArcObjects 初学者的眼光看来，PageLayout 对象有点名不副实。在 ArcMap 的结构图中可以看到它和 Map 对象是并驾齐驱的，两者的地位应该一样，但是打开 ArcMap 的布局视图的时，会发现它仅仅是一个“画板”。在这个对象上能够进行的操作只是摆弄不同的图形元素，如指北针、标题、比例尺等，而放在这个 PageLayout 中的地理数据，就好像一幅栅格图片，不能进行任何的地理分析。但是事实并非如此，在 ArcMap 的程序结构中，PageLayout 实际上担当了“第三号人物”的角色，它的重要性仅次于 Application 和 MxDocument。本质上 PageLayout 和 Map 根本不是对等的，而是前者通过框架元素 MapFrame 在管理后者，尽管文档对象可以直接引用到它的 Map 对象上，但这些 Map 对象却是被 PageLayout 的 MapFrame 对象所管理的，任何 Map 对象都是这样，它们并不是被文档对象直接管理。

在使用 PageLayoutControl 控件编程中，自然没有了 MxDocument 这个“管理

者”，PageLayout 与 Map 控件一样，都是自己管理自己。

PageLayout 类主要实现了 IPageLayout 接口，它定义了用于修改页面的版式(layout)的方法和属性，其中包括图形的位置属性、标尺和对齐网格(Snap Grid)的设置以及确定页面是如何显示在屏幕上的方法。使用这个接口可以管理如 RulerSettings、SnapGrid、SnapGuides 和 Page 对象。

IPageLayout::ZoomToWhole 可以让 PageLayout 以最大尺寸显示；IPageLayout::ZoomToPercent 则可以按照输入的比例显示；IPageLayout::ZoomToWidth 可以让视图显示的范围匹配控件对象的宽度。

下面是在 PageLayoutControl 控件中控制显示：

```
private void ZoomPer()
{
    IPageLayout pPageLayout;
    pPageLayout = axPageLayoutControl1.PageLayout;
    //页面缩小到10%
    pPageLayout.ZoomToPercent(10);

}
```

IPageLayout::Page 属性用于获得放在 PageLayout 对象中的 page 对象；RulerSettings 用于获得 PageLayout 控制的标尺对象，而 SnapGrid 和 VerticalSnapGrid 用于控制 PageLayout 对象中显示的网格对象。

PageLayout 对象主要用于管理元素，因而它也实现了 IGraphicsContainer 这个接口，其使用方法和 Map 对象的一样，它定义了属性和方法用于管理 PageLayout 对象中的元素。使用它们可以添加一个新元素，或者获取一个已经存在的元素对象，如一个版式页面上方的地图标题就是一种保存在它里面的文字元素。

PageLayout 还实现了 IGraphicsContainerSelect 接口，这个接口专门用于被选择的元素的管理，如 IGraphicsContainerSelect::UnselectAllElements 也可以清除图形元素选择集中所有的对象。

5.4.2 Page 对象

PageLayout 对象被创建后，会自动产生一个 Page 对象来管理布局视图中的页面，通过 IPageLayout::Page 可以得到它的指针。Page 对象是一个很简单的容器对象，它用于装载地理数据，不提供任何别的分析、查询功能。如果将 PageLayout 当作一个地图画板，那么 page 对象就是在画板中专用于绘制地图图形的部分。

Page 类的主要接口是 IPage，它除了用于管理 Page 的颜色、尺寸和方向等属性外，还可以管理其版式单位、边框类型和打印区域等属性。

使用 esriPageFormID 枚举值可以设置 Page 对象的尺寸，如果要改变 Page 的标准尺寸，可以将某个枚举值传给 IPage::FormID，这个方法比使用 IPage::PutCustomSize 来设置一个自定义的尺寸快得多。esriPageFormSameAsPrinter 是这个枚举值中重要的一项，在将它设置为 Page 的尺寸后，无论打印机的页面设置如何变化，版式页面的尺寸始终会和打印机保持一致。

IPage::Background 属性可以用于改变 Page 的背景样式；BackgroundColor 则可以改变背景的颜色；Border 属性用于设置 Page 的边框，而 Units 则可以获得 page 所使用的单位。

下面的方法是改变 Page 的背景颜色：

```
private void changePageColor()
{
    IPage pPage;
    IPageLayout pPageLayout = axPageLayoutControl1.PageLayout ;
    pPage = pPageLayout.Page;
    IRgbColor pRGBColor ;
    pRGBColor = new RgbColorClass();
    pRGBColor.Red = 112;
    pRGBColor.Green = 200;
    pRGBColor.Blue = 102;
    pPage.BackgroundColor = pRGBColor;

}
```

Page 还是一个拥有众多事件的对象，这些事件通过 IPageEvents 接口来定义的，如 PageColorChanged、PageMarginsChanged、PageSizeChanged 和 PageUnitsChanged 等。Page 对象会负责监听这些事件，并作出相应的反应。如当 Page 的单位发生变化后，布局视图需要更新它的转换参数、Snapgrid 以及 SnapGuides 等附属对象。

5.4.3 SnapGrid 对象

SnapGrid 是 PageLayout 上用于摆放元素而设置的辅助点，这些点有规则呈网状排列，便于用户对齐元素。尽管 SnapGrid 是一个组件类，但是用户一般都不需要单独创建它，而是通过 IPageLayout::SnapGrid 属性方便地得到当前 PageLayout

使用的 SnapGrid 对象的指针。

SnapGrid 类实现了 ISnapGrid 接口，用于设置 SnapGrid 的各种属性。如 HorizontalSpacing 和 VerticalSpacing 属性用于设置网点之间的水平距离和垂直距离，而 IsVisible 用于确定这些网点是否处于可见状态，Draw 方法用于将一个 SnapGrid 对象绘制在 Page 上。

下面是改变一个 SnapGrid 间距尺寸的例子：

```
private void SetSnapGrid()
{
    ISnapGrid pSnapGrid;
    IPageLayout pPageLayout = axPageLayoutControl1.PageLayout ;
    pSnapGrid = pPageLayout.SnapGrid;        //获得PageLayout的SnapGrid对象
    pSnapGrid.VerticalSpacing = 2;        //设置其垂直间距离
    pSnapGrid.HorizontalSpacing = 1;
    pSnapGrid.IsVisible = true ;      //设置为可见
    axPageLayoutControl1.CtlRefresh((esriViewDrawPhase) 65535, Type.Missing,
Type.Missing);

}
```

将这段代码在 PageLayoutControl 控件上使用，会看到控件上出现有规律的网点。

5.4.4 SnapGuides 对象

SnapGuides 是为了更好放置地图而在 PageLayout 出现的辅助线。这个对象有两种类型：一是水平辅助线，另一种是垂直辅助线。它们可以分别通过 IPageLayout::VerticalSnapGuides 或 IPageLayout::HorizontalSnapGuides 获得。每个 SnapGuides 都管理着一个 Guide 集合，即这种辅助线可以同时存在多条。

SnapGuides 类实现了 ISnapGuides 接口，它定义了管理 SnapGuide 的属性和方法。如 AreVisible 设定 SnapGuides 是否可见，GuideCount 可以返回一个 SnapGuides 对象中 Guide 的个数，而使用 Guide 可以按索引值得到某个具体的 Guide 对象。

AddGuides 方法可以将一个 Guide 放在指定位置上；RemoveAllGuides 和 RemoveGuide 方法分别可以清除所有的 Guide 和按照索引值清除 Guide。

下面是一个添加 Guide 的例子：

```
private void AddGuide()
```

```
        {
            ISnapGuides pSnapGuides;
            IPageLayout pPageLayout = axPageLayoutControl1.PageLayout ;
            pSnapGuides = pPageLayout.HorizontalSnapGuides;
            pSnapGuides.AddGuide(5);
            pSnapGuides.AddGuide(7);
            pSnapGuides.AddGuide(9);
            pSnapGuides.AddGuide(11);
            pSnapGuides = pPageLayout.VerticalSnapGuides;
            pSnapGuides.AddGuide(2);
            pSnapGuides.AddGuide(4);
            pSnapGuides.AddGuide(6);
            pSnapGuides.AddGuide(8);
            pSnapGuides.AreVisible = true;
            axPageLayoutControl1.CtlRefresh((esriViewDrawPhase) 65535, Type.Missing,
Type.Missing);
        }
```

5.4.5 RulerSettings 对象

标尺对象也是为了辅助元素的放置而出现在 PageLayout 对象上方和左方的辅助尺，尽管 RulerSettings 也是可以直接创建的组件类，但它一般都是直接通过 IPageLayout::RulerSettings 获得与当前 PageLayout 相关的标尺。

RulerSettings 实现了 IRulerSettings，它仅仅定义了一个属性，即 SmallestDivision 用于设置最小的区分值。

5.5 Element 对象

在 ArcGIS 中可以显示在视图上的图形分为两种，一种是基于 Layer 的要素（Feature）等储存在数据库中的数据，另外一种就是元素（element）。元素是一个地图中除去要素数据外的部分，即在一幅地图中，除了保存在数据库中的地理数据外，其余的对象全部是元素。

在 MapControl 控件中通过 DrawShape 方法绘制的图形，在视图进行了缩放等操作后就会消失，这是因为这些图形本质上都是缓存，仅仅是暂时存在。如果要保存这些元素，就需要使用 MXD 文件。

Element 是一个非常庞大复杂的对象集合，它主要分为两大部分：图形元素（Graphic Element）和框架元素（Frame Element）。

图形元素包括 GroupElement、MarkerElement、LineElement、TextElement、DataGraphElement、PictureElement 和 FillshapeElement 等对象，它们都是作为图形的形式而存在，在视图上是可见的。框架元素包括 FrameElement、MapFrame、MapSurroundFrame、OleFrame 和 TableFrame 等对象，它们都是作为不可见的容器而存在的。

在 Map 或 PageLayout 对象中可以通过 IGraphicsContainer 接口来管理这些元素，使用这个接口定义的方法可以添加、删除和更新单个的位于 Map 或 PageLayout 上的元素。使用 GroupElement 对象还可以将多个元素编组作为单个实体来给用户使用。

IElement 是所有图形元素和框架元素类都实现的接口。这个接口可以让程序员确定元素的 Geometry 属性，它也定义了方法让用户查找元素和绘制元素。Element 只是一个抽象类，在实际编程中必须明确指定使用元素的类型。另外还必须提醒读者的是 IElement 和 ILineElement、ITextElement 等并不是父子关系，后者中没有 Geometry 属性。

5.5.1 图形元素

所有的图形元素（Graphics Element）类都实现了 IGraphicsElement 接口，这个接口仅定义了一个 SpatialReference 属性，用于设置这些图形元素的空间参考。空间参考是一个图形能够在一定范围内正确显示所必须设置的属性。

除此以外，图形元素还实现 ITransform2D 接口，这个接口定义的方法和属性可以让图形元素移动、旋转和缩放。关于这个接口的具体内容，读者可以阅读开发帮助的相关条目。

5.5.1.1LineElement 和 MarkerElement 对象.

LineElement 和 MarkerElement 是最简单的图形元素，它们在数据视图（data view）或者布局视图（Pagelayout view）上表现为线和点的形式。要将图形元素显示在视图上，一般的步骤是这样的：

1. 产生一个新的元素对象；
2. 确定元素显示时使用的 Symbol（符号）和 Geometry（几何形体对象）；
3. 使用 IGraphicsContainer::AddElement 把元素添加到视图中去；

4. 刷新视图，让添加的元素可以显示出来。

以 LineElement 为例，添加它到视图需要使用两个接口：IElement 和 ILineElement，前者用于确定线元素的 Geometry，后者用于确定 Symbol。需要注意的是能够使用的 Symbol 和 Geometry 不是混用的，LineElement 元素只能使用用于修饰 LineElement 对象的符号，也只能使用 Line 或者 Polyline 作为 Geometry。MarkerElement 也是一样，使用的是 Marker 类型的 Symbol 和点作为它的 Geometry。

下面的代码是如何添加一个 MarkerElement 到 Map 中去，它放在控件的 MouseDown 事件中：

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            IMap pMap;
            IActiveView pActiveView;
            //获得控件的Map对象
            pMap = axMapControl1.Map;
            //QI
            pActiveView = pMap as IActiveView;
            //新建一个点point
            IPoint pPt;
            pPt = new PointClass();
            pPt.PutCoords(e.mapX, e.mapY);
            //产生一个Marker元素
            IMarkerElement pMarkerElement;
            pMarkerElement = new MarkerElementClass();
            //产生修饰Marker元素的symbol
            ISimpleMarkerSymbol pMarkerSymbol;
            pMarkerSymbol = new SimpleMarkerSymbolClass();
            //设置符号颜色
            pMarkerSymbol.Color = getRGB(11, 200, 145);
            //设置符号大小
            pMarkerSymbol.Size = 2;
            //设置符号类型
            pMarkerSymbol.Style = esriSimpleMarkerStyle.esriSMSDiamond;
            IElement pElement;
            pElement = pMarkerElement as IElement;
            //QI，得到IElement接口对象，用于设置元素的Geometry
```

```
            pElement.Geometry = pPt;
            pMarkerElement.Symbol = pMarkerSymbol;
            IGraphicsContainer pGraphicsContainer;
            pGraphicsContainer = pMap as IGraphicsContainer;
            //将元素添加到Map中
            pGraphicsContainer.AddElement(pMarkerElement as IElement, 0);
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);

        }
```

MarkerElement 类支持 IMarkerElement 接口，这个接口的 Symbol 方法用于在新建一个 MarkerElement 对象时设置点符号（MarkerSymbol）。

5.5.1.2TextElement 对象

地图为了显示图形的附加信息，一般都是需要采用文字标注来完成。地图的标注有两种形式：一种是保存在地理数据库中以标注类的形式存在，另一种是使用文字元素。

TextElement 对象实现了 ITextElement 接口，这个接口定义了设置文字元素特征的属性，如 ScaleText（文字尺寸）、Text（字符）和 Symbol（文字的修饰符号）。

需要注意的是 TextElement 的 Geometry 是一个点，即 Point 对象。

下面是在 Map 视图上添加字符元素的代码：

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            IMap pMap;
            IActiveView pActiveView;
            //获得控件的Map对象
            pMap = axMapControl1.Map;
            //QI
            pActiveView = pMap as IActiveView;
            ITextElement pTextEle;
            IElement pEles;
            //新建文字字符对象并设置它的属性
            pTextEle = new TextElementClass();
            pTextEle.Text = "ArcObjects";
            pEles = pTextEle as IElement;
```

```
        //设置文字字符的几何形体属性
        IPoint pPoint;
        pPoint = new PointClass();
        pPoint.PutCoords(e.mapX, e.mapY);
        pEles.Geometry = pPoint;
        //添加到Map对象中并刷新以实时显示
        IGraphicsContainer pGraphicsContainer;
        pGraphicsContainer = pMap as IGraphicsContainer;
        //将元素添加到Map中
        pGraphicsContainer.AddElement(pEles, 0);
        pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
    }
```

5.5.1.3DataGraphElement 对象

DataGraph 是一种基于地理数据的图形，在 ArcMap 中载入地理数据后，点击“Tools”-“Graphs”-“Create”，系统依据数值数据产生一张统计表。而 DataGraphElement 对象就是这张统计表，它只能放置在 PageLayout 视图上。

IDataGraphElement 接口仅仅被 DataGraphElement 对象所实现，它定义了两个属性，IDataGraphElement::DataGraph 用于设定元素的数据图表对象，IDataGraphElement::Map 则需要被设置为图表的父地图对象。

5.5.1.4GroupElement 对象

GroupElement 对象可以将多个元素编为一组当一个实体来使用。如果用户需要对多个要素进行相同的操作，如同时移动多个要素的时候，可以将它们编为一个组。GroupElement 类默认实现 IGroupElement 接口，它定义了操作 GroupElement 的方法和属性。

IGroupElement::AddElement 方法可以将一个元素添加到 GroupElement 对象；而 IGroupElement::ClearElements 可以清除 GroupElement 中所有的元素；DeleteElement 方法可以根据 GroupElement 中的某个元素的索引值删除这个元素。

GroupElement 是一个组件类，即用户可以新建一个单独的 GroupElement 对象用于处理元素。

下面的代码是将图形元素添加进 GroupElement 中去：

```
    private void GroupElement(IGraphicsContainer pGraphicsContainer)
```

```
{
    //产生新的GroupElement对象
    IGroupElement pGroupEle;
    pGroupEle = new GroupElementClass();
    IElement pEle;
    //pGraphicsContainer是使用 QI方法获得的管理元素的对象
    pGraphicsContainer.Reset();
    //获得 pGraphicsContainer中的第一个元素
    pEle = pGraphicsContainer.Next();
    //遍历元素
    while (pEle != null)
    {
        //将它添加进pGroupEle对象
        pGroupEle.AddElement(pEle);
        pEle = pGraphicsContainer.Next();
    }
  //最后使用ElementCount来测试添加是否成功
    MessageBox.Show(pGroupEle.ElementCount.ToString());
}
```

5.5.1.5FillShapeElement 对象

FillshapeElement 是一个抽象类，它的子类有 CircleElement、EllipseElement、PolygonElement 和 RectangleElement。这些对象的共同特点是它们的 Geometry 属性都是一个二维的封闭图形，在视图上分别表现为圆形元素、椭圆形元素、多边形元素和矩形元素。

IFillshapeElement 是所有 FillshapeElement 类都实现的接口，它定义了用于显示图形元素的 Symbol 属性，这个 Symbol 属性必须设置为 IFillsymbol 对象。

这些类型的图形元素的新建和加入到 Map 或 PageLayout 视图的方法与点、线元素没有区别。下面的代码是添加一个 PolygonElement 元素到 Map 对象中，它也是在 MapControl 控件的 MouseDown 事件中实现的：

```
    private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
    {
        IMap pMap;
        IActiveView pActiveView;
```

```
            //获得控件的Map对象
            pMap = axMapControl1.Map;
            //QI
            pActiveView = pMap as IActiveView;
            IPolygon pPolygon ;
            pPolygon = axMapControl1.TrackPolygon() as IPolygon;
            //产生一个ISimpleFillsymbol符号
            ISimpleFillSymbol pSimpleFillsym;
            pSimpleFillsym = new SimpleFillSymbolClass();
            pSimpleFillsym.Style = esriSimpleFillStyle.esriSFSDiagonalCross;
            pSimpleFillsym.Color = getRGB(102, 200, 103);
        //产生一个PolygonElement对象
            IFillShapeElement pPolygonEle;
            pPolygonEle = new PolygonElementClass();
            pPolygonEle.Symbol = pSimpleFillsym;
            IElement pEle ;
            pEle = pPolygonEle as IElement;
            pEle.Geometry = pPolygon;
            //将元素添加到Map对象中
            IGraphicsContainer pGraphicsContainer;
            pGraphicsContainer = pMap as IGraphicsContainer;
            pGraphicsContainer.AddElement(pEle, 0);
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);


    }
```

5.5.1.6PictureElement 对象

在 ArcMap 的布局视图中制作地图的时，用户可以向 PageLayout 中插入一张位图图片，通过这种方式可以制作更加漂亮的地图以输出使用。这种图片其实也是一种图形元素，即 PlctureElement 对象。

PictureElement 是一个抽象类，它的两个子类 BmpPictureElement 和 EmfPictureElement，它们分别用于往布局视图中插入 bmp 文件和 emf 文件。

IPictureElement 是两个子类都实现的接口，使用它可以用于操作一个图片元素。如 IPictureElement::Filter 属性是供 OpenFileDialog 使用的过滤器；IPictureElement::MaintainAspectRatio 属性可以决定调整图片尺寸时是否保持其长

宽比例；IPictureElement::PictureDescription 可以添加图片的附加描述信息；IPictureElement::SavePictureInDocument 属性则确定这张图片是否会被保存到 MXD 文件上。

IPictureElement::ImportPictureFromFile 是这个接口定义的唯一的方法，它用于取得一张图片文件。下面的代码演示如何在 PageLayout 控件中插入一张 bmp 图片：

```
        private void axPageLayoutControl1_OnMouseDown(object sender,
ESRI.ArcGIS.PageLayoutControl.IPageLayoutControlEvents_OnMouseDownEvent e)
        {
            IPageLayout pPageLayout ;
            pPageLayout = axPageLayoutControl1.PageLayout;
            IGraphicsContainer pGraphicsContainer ;
            pGraphicsContainer = pPageLayout as IGraphicsContainer ;
            IActiveView pActlveView;
            pActiveView = pPageLayout as IActiveView;
            //新建一个 BmpPictureElement对象
            IPictureElement pBmpPicEle;
            pBmpPicEle = new BmpPictureElementClass();
            //确定使用的位图文件
            pBmpPicEle.ImportPictureFromFile(@"C:\My Documents\sim.bmp");
            //图片在缩放的时候长宽保持比例
            pBmpPicEle.MaintainAspectRatio = true;
            IEnvelope pEnv;
            pEnv = axPageLayoutControl1.TrackRectangle();
            IElement pEle;
            pEle = pBmpPicEle as IElement;
            //确定元素的Geometry属性，它是一个矩形
            pEle.Geometry = pEnv;
            pGraphicsContainer.AddElement(pEle, 0);
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
        }
```

5.5.2 框架元素

框架元素（Frame Element）是另一大类的元素对象。框架元素有两个主要对象：一个是 MapFrame（地图框架），另一个是 MapSurroundFrame（地图修饰框架）。

框架元素是一种包含着其他的地图元素的容器。所有的框架元素类（如FrameElement、OleFrame、MapFrame 和 MapsurroundFrame）都实现了IFrameElement 接口，这个接口定义了操作框架元素最一般的属性和方法，如BackGround、Border 属性可以用于设置框架元素的背景和边框。

IFrameElement::Object 属性则可以返回这个框架对象的指针，由于系统并不能明确确定返回的类型，因而它的类型是 object 类型。

框架元素尽管不能被可视化看到，但它们在 ArcMap 程序中扮演了极为重要的角色。图 5-6 是框架元素的实际内在结构。

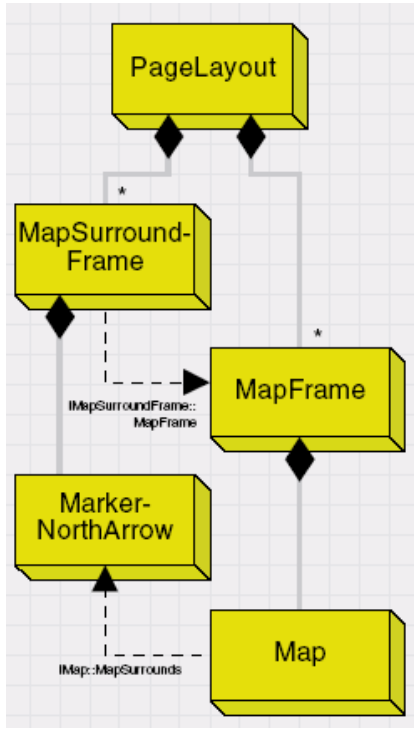


图 5-6　框架元素结构

从图 5-6 可以看到，PageLayout 可以直接得到它管理的 MapFrame 和 MapSurroundFrame 对象的指针，在一个 PageLayout 对象中，可以存在多个框架元素对象。MapFrame 对象是 Map 的容器，它用于管理 Map 对象；而 MapSurroundFrame 对象则用于管理 MapSurround 对象，MapSurround 就是为了修饰地图而使用的比例尺、比例文本和指北针等对象。

MapFrame 对象也可以使用 IMapSurroundFrame::MapFrame 得到，而 MapSurround 对象则可以使用 IMap::CreateMapSurround 方法得到。这种错综复杂的关系揭示了 ArcMap 程序的内部结构：在这个程序内部，PageLayout 对象担当了“三号”角色，是它通过 MapFrame 管理了 Map 对象，PageLayout 和 Map 仅仅是看起来“地位相当”，而实际上绝非是同一级别的对象。

每个 MapSurroundFrame 都是与一个 MapFrame 相联系的。如果一个 MapFrame 被删除了那么它其中所有的 MapSurroundFrame 对象也将被删除。

5.5.2.1MapFrame 对象

MapFrame 对象是由 PageLayout 对象来控制的，它用于管理 Map 对象。MapFrame 对象支持 IMapFrame 接口，使用 IMapFrame 定义的属性和方法可以控制其中的 Map 对象。

IMapFrame::Map 属性可以获得这个地图框架内的地图对象，它是一个只读属性；MapBounds 属性则可以返回地图对象的范围，即一个 Envelope 对象；MapFrame 使用 MapScale 属性来确定地图显示的比例，它们都是用于更新这个框架元素内的地图对象的。

IMapFrame::CreateSurroundFrame 方法用于返回一个 MapSurroundFrame 对象。

MapFrame 对象还实现了 IMapGrids 接口，它可以用于管理地图框架中的 MapGrid（地图格网）。如 AddMapGrid 用于添加一个 MapGrid 对象，ClearMapGrids 和 DeleteMapGrid 可以清除所有的 MapGrid 对象或者使用索引值删除地图框架对象中的某个 MapGrid。

5.5.2.2MapSurroundFrame 对象

MapSurroundFrame 是一种用于管理 MapSurround 对象的框架元素。MapSurround 是指北针、比例尺和图例一类的对象，它们是一种“智能”的，会自动与某个地图对象关联，随着地图视图的变化而变化的对象。

MapSurroundFrame 支持 MapFrameResized 事件，当地图的尺寸改变的时候，它会监听这个事件，并自动更新比例尺等对象。

一个 Mapsurround 对象与 MapFrame 是相关联的，当地图框架发生旋转的时候，指北针对象的方向也会发生变化。IMapsurroundFrame 接口是 MapsurroundFrame 对象的默认接口，IMapSurroundFrame::MapFrame 属性可以得到与自身关联的 MapFrame 对象，而 IMapSurroundFrame::MapSurround 属性则可以得到它持有的 MapSurround 对象。

5.5.2.3TableFrame 对象

TableFrame 是一种可以容纳 Table 对象的框架元素。当用户在 ArcMap 打开一个要素图层的属性表，点击“Options”按钮，选择“Add Table to Layout”后，会在布局视图中发现一份数据表，如图 5-7 所示。

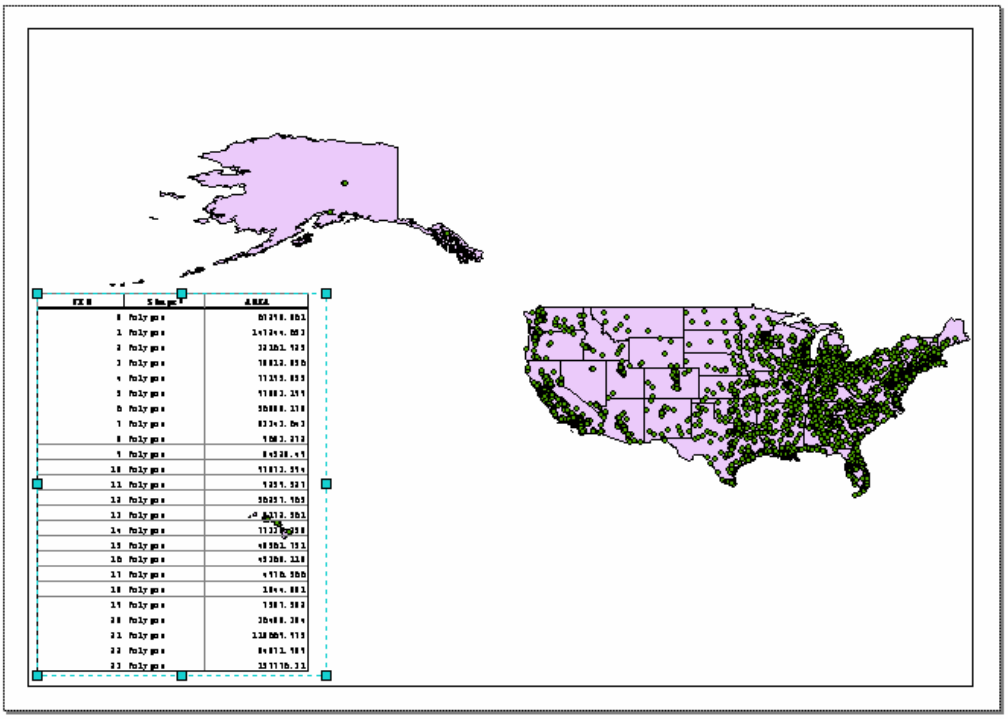


图 5-7　TableFrame 的例子

这个 Table 对象就是由 TableFrame 对象来进行管理的，后者只能放置在布局视图中，而不能添加到 Map 对象中去。ITableFrame 接口是 TableFrame 类实现的唯一接口，它提供了操作框架元素中表的属性和方法。

ITableFrame::StartCol 和 StartRow 可以设置显示表时的列数和行数；而使用 ITableFrame::TableView 属性可以让程序员得到一个 ITabelView 对象去改变表视图的属性，诸如选择显示等。

ITableFrame::Table 属性将会返回给用户一个与框架相关的 ITable 对象，它实际上是普通属性表或是要素类，也就是这个数据表的数据源对象。当然用户也可以使用 ITableFrame::TableProperty 属性得到的 ITableProperty 对象来获得它的指针。

5.5.3 元素的选择跟踪对象

在 IElement 接口中，读者可以看到 Selection tracker 属性，它可以返回每个元素自己的选择跟踪对象，这个对象可以提供整形（reshape）功能。例如，在添加一个元素后，可以通过元素的选择跟踪对象绘制出来，用户也可以通过拖动选择跟踪对象来改变元素的形状。

在 ArcMap 中，用户选择一个元素后，在元素的包络线周围或元素边缘周围会出现绿色的选择线和点，则说明选择了跟踪对象。使用这个功能必须在选择一个元素后才能进行。

下面是一个简单的绘制例子，首先需要选择一个元素 pEle。

```
private void demo(IElement pEle)
{
    //选择一个元素
    IGraphicsContainerSelect pGraphConsel;
    pGraphConsel = pMap as IGraphicsContainerSelect;
    IActiveView pAV = pMap as IActiveView;
    pGraphConsel.SelectElement(pEle);
    pAV.PartialRefresh(esriViewDrawPhase.esriViewForeground, null, null);
}
```

下面的代码产生在一个 OnAfterDraw 事件中：

```
private void axMapControl1_OnAfterDraw(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnAfterDrawEvent e)
{
    if (e.viewDrawPhase.Equals ( esriViewDrawPhase.esriViewForeground))
    {
        if (pEle == null) return;
        ISelectionTracker pSelectionTracker;
        pSelectionTracker = pEle.SelectionTracker;
        pSelectionTracker.Draw(e.display as IDisplay, 0,
esriTrackerStyle.esriTrackerDominant);
    }
}
```

关于 SelectionTracker 的详细内容，读者可以阅读 Exploring ArcObjects 的相关章节和开发帮助的内容。

5.6 MapGrid 对象模型

一幅成品地图中总是会看到地图格网。在小比例尺地图中，经纬网可以让指明某个区域在地球上的确切位置；在大比例地图里，也可以使用方里网将一块区域进行规律划分，它们是修饰一幅地图不可缺少的部分。

ArcObjects 中也提供了用于辅助显示地图的地图格网 MapGrids 对象，它出现

了地图边缘上，用于显示经纬度或者方里网。在本质上它就是 MapFrame 对象，必须出现在 PageLayout 视图而不是 Map 视图。一幅地理网格主要由 GridLine（格网线）、GridLabel（格网标注）和 GridBorder（格网边框）三部分组成，如图 5-8 所示。

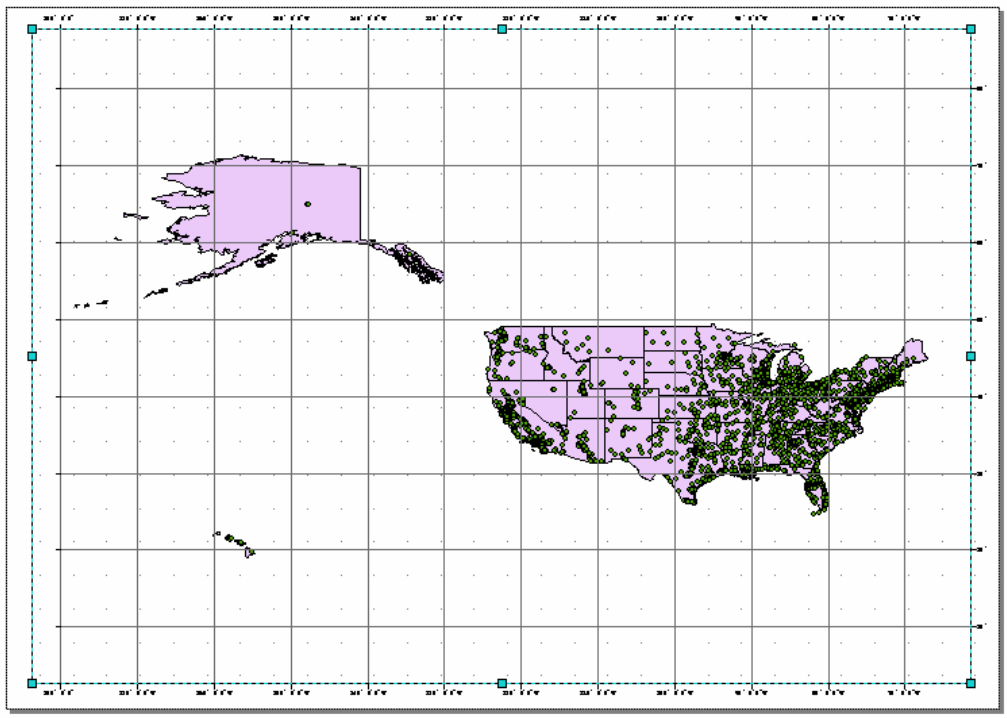


图 5-8　MapGrid 的组成

5.6.1 MapGrid 对象

MapGrid 对象是布局视图中的一种参考线或点，它可以帮助用户快速地确定地图中要素的位置。MapGrid 对象由 MapGrids 来管理，一个 MapGrids 中可以存在多个 MapGrid 对象。

在布局视图中获得一个 MapGrid 的指针非常容易，使用 IGraphicsContainer::FindFrame 可以得到 PageLayout 对象的 MapFrame 对象，然后通过 QI 方法可以得到，如下面的代码段：

```
private IMapGrid GetMapGrid()
{
    IActiveView pActiveView;
    IGraphicsContainer pGraphicsContainer ;
    IMapFrame pMapFrame;
```

```
            IMap pMap;
            pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
            pMap = pActiveView.FocusMap;
            pGraphicsContainer = axPageLayoutControl1.PageLayout as
IGraphicsContainer;
            pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
            IMapGrids pMapGrids;
            pMapGrids = pMapFrame as IMapGrids;
            IMapGrid pMapGrid ;
            pMapGrid = pMapGrids.get_MapGrid(0);
            return   pMapGrid;
        }
```

MapGrid 是一个抽象类，它有四个子类对象，如 MeasuredGrid、Graticule、IndexGrid 和 CustomGridOverlay。

IMapGrid 是所有类型的地图格网类都实现的接口，它用于设置 MapGrid 对象的一般属性和方法，如 Border 属性，用于设置地图网格的边框。IMapGrid::LabelFormat 属性用于设置地图网格上的标签格式；IMapGrid::Linesymbol 则可以用于设置网线的样式。IMapGrid 接口还定义了多个设置 tick 对象的属性，所谓 tick，就是一个小记号点，如网线之间的交点、网线与边的交点等。SetSubTicksVisiblity 等方法可以按照用户的要求来确定这些点。

Graticule 是使用经纬线来划分地图的地图格网对象，它实现了两个接口 IGraticule 和 IMeasuredGrid。由于 Graticule 对象是使用经纬网，因而需要设置空间参考属性。

IMeasuredGrid 接口定义了多个属性用于设置 Origin（原点），如 FixedOrigin 为是否根据计算自动设置起点；Units 是用于设置原点间隔的单位，而 XOrigin 和 YOrigin 用于设置 X 和 Y 方向上的起点；XIntervalSize 和 YIntervalSize 是确定 X、Y 方向上两线之间的间隔。

MeasuredGrid 也是使用经纬度作为地图网格来划分地图的，它与 Graticule 对象的不同之处在于它的空间参考属性可以和 MapFrame 对象一致，也可以不一致。它除了实现 IMeasureGrid 接口外，还实现了 IProjectedGrid 接口用于设置它的投影属性。

IndexGrid 是使用索引值的方式来划分地图的区域的对象，通常南北方向用“ABC”，而东西方向用“123”来表示，它适合小区域内地块的划分等。IndexGrid 类实现了 IIndexGrid 接口，IIndexGrid::XLabel 和 YLabel 属性分别用于设置网格 X、Y 轴上的标签。ColumnCount 和 RowCount 属性分别设置了 MapGrid 网格划分的列

数和行数。

下面是一个 IndexGrid 的例子：

```
private IIndexGrid CreateIndexGrid()
{
    IIndexGrid pIndexGrid;
    pIndexGrid = new IndexGridClass();
    pIndexGrid.ColumnCount = 5;
    pIndexGrid.RowCount = 5;
    int i;
    for (i = 0;i<= pIndexGrid.ColumnCount - 1;i++)
    {
        pIndexGrid.set_XLabel(i,(i + 1).ToString());
    }
    for (i = 0;i<= pIndexGrid.RowCount - 1;i++)
    {
        pIndexGrid.set_YLabel(i, i.ToString () + "A");
    }
    return pIndexGrid;
}
```

5.6.2 MapGridBorder 对象

地图格网是有边框的，这些边框对象类型很多，它们都实现了 IMapGridBorder 接口。边框有两种类型：SimpleMapGridBorder 和 CalibratedMapGridBorder。IMapGridBorder::DisplayName 属性可以得到边框的显示名，它们的 DisplayName 分别是“simple border”和“calibrated border”。

当用户新建一个 MapGridBorder 对象时，不必使用 IMapGridBorder 接口，这是因为这个接口的属性都是只读的。

SimpleMapGridBorder 对象只是使用简单的直线来作为地图的边框，因而在 ISimpleMapGridBorder 接口中必须设置的是 LineSymbol 属性，它用于确定边框线的样式、宽度和颜色。

下面是一个简单地图网格边框的例子：

```
private ISimpleMapGridBorder    CreateSimpleMapGridBorder()
{
    //新建一个简单边框对象
```

```
        ISimpleMapGridBorder pSimpleMapGridBorder;
        pSimpleMapGridBorder = new SimpleMapGridBorderClass();
        //设置简单边框的属性
        ISimpleLineSymbol pLineSymbol;
        pLineSymbol = new SimpleLineSymbolClass();
        pLineSymbol.Style = esriSimpleLineStyle.esriSLSSolid;
        pLineSymbol.Color = getRGB(0, 0, 0);
        pLineSymbol.Width = 2;
        pSimpleMapGridBorder.LineSymbol = pLineSymbol;
        return pSimpleMapGridBorder;
    }
```

CalibratedMapGridBorder 是使用一种渐变线段的边框对象，这个对象支持的接口是 ICalibratedMapGridBorder，它定义了边框的前景色、后景色、宽度，线段的间隔长度等属性。下面的代码段可以产生一个 CalibratedMapGridBorder 对象：

```
        ICalibratedMapGridBorder pCalibratedBorder;
        pCalibratedBorder = new CalibratedMapGridBorderClass();
        //设置复杂边框的属性
        pCalibratedBorder.BackgroundColor = getRGB(255, 255, 255);
        pCalibratedBorder.ForegroundColor = getRGB(100, 100, 60);
        pCalibratedBorder.BorderWidth = 10;
        pCalibratedBorder.Interval = 60;
        pCalibratedBorder.Alternating = true;
        //将边框对象设置给MapGrid的border属性
        pMapGrid.Border = pCalibratedBorder;
```

5.6.3 MapGridLabel 对象

每一个地图格网都有标签，无论是在地图中使用经纬网还是方里网，都必须设置一些标识性字符，这些字符都需要使用到 MapGridLabel 对象来设置。

IGridLabel 接口控制着所有 GridLabel 对象的一般属性。用户需要注意：并不是所有的标签都可以使用在某种格网上，下面的表格列出了 grid 可以使用的 label 种类：

Grid 类型	可以使用的 MapGridLabel	DisplayName
Graticule	DMSLabel	Degrees Minutes Seconds
MeasuredGrid	FormattedLabel	Formatted

	MixedFontLabel	Mixed Font
IndexGrid	ButtonTabStyle	Button Tabs
	RoundedTabStyle	Rounded Tabs
	ContinuousTabStyle	Continuous Tabs
	BackgroundTabStyle	Filled Background

IGridLabel::LabelAlignment 属性可以设置格网标注在格网对象的四个边上的水平和垂直方向，它需要传入一个 esriGridAxisEnum 枚举类型值。

DMSGridLabel对象的特点是其标注字符使用的是经纬度的单位，即度、分和秒，例如 $110^0$10’10”。DMSGridLabel类实现IDMSGridLabel接口用于管理经纬网标注对象的属性，如字体、标注类型等。

FormattedGridLabel 对象可以让标注上的字符格式化显示，它的类实现了 IFormattedGridLabel 接口。IFormattedGridLabel::Format 属性需要传入一个 INumberFormat 对象用于设定字符格式，如字符是否显示负号、小数点后设置多少位等。

MixedFontGridLabel 可以使用两种字体来设置一段标注文本，IMixedFontGridLabel::NumberOfDigits 属性可以确定两种字体是如何应用到标注字符上的，其中的 n 值可以让标签最后的 n 个字符设置为第二种字体和颜色，而剩下的字符使用第一种字体和颜色。MixedFontGridLabel 对象的主要颜色和字体使用 IGridLabel::Color 和 IGridLabel::Font 来设置，而第二种颜色和字体则由 IMixedFontGridLabel::SecondaryColor 和 IMixedFontGridLabel::SecondaryFont 确定。

下面的例子产生了一个新的 MapGrid，并将其加入到 PageLayoutControl 控件中：

```
private void CreateMeasuredGrid()
{
    //获得PageLayout的基本对象
    IActiveView pActiveView;
    IMap pMap;
    pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
    pMap = pActiveView.FocusMap;
    //产生一个measuredgrid对象
    IMapGrid pMapGrid;
```

```
IMeasuredGrid pMeasuredGrid ;
pMeasuredGrid = new MeasuredGridClass();
pMapGrid = pMeasuredGrid as IMapGrid;
pMeasuredGrid.FixedOrigin = true;
pMeasuredGrid.Units = pMap.MapUnits;
pMeasuredGrid.XIntervalSize = 10;
pMeasuredGrid.XOrigin = -180;
pMeasuredGrid.YIntervalSize = 10;
pMeasuredGrid.YOrigin = -90;
//设置投影属性
IProjectedGrid pProjectedGrid;
pProjectedGrid = pMeasuredGrid as IProjectedGrid;
pProjectedGrid.SpatialReference = pMap.SpatialReference;
pMapGrid.Name = "Measured Grid";
//标签格式设置
ICalibratedMapGridBorder pCalibratedBorder ;
pCalibratedBorder = new CalibratedMapGridBorderClass();
//设置ICalibratedMapGridBorder的属性
pCalibratedBorder.BackgroundColor = getRGB(255, 255, 255);
pCalibratedBorder.ForegroundColor = getRGB(0, 0, 0);
pCalibratedBorder.BorderWidth = 3;
pCalibratedBorder.Interval = 72;
pCalibratedBorder.Alternating = true;
pMapGrid.Border = pCalibratedBorder as IMapGridBorder ;
IFormattedGridLabel pFormattedGridLabel;
pFormattedGridLabel = new FormattedGridLabelClass();
//设置IGridLabel的属性
IGridLabel pGridLabel;
pGridLabel = pFormattedGridLabel as IGridLabel;
stdole.StdFont pFont;
pFont = new stdole.StdFontClass();
pFont.Name = "Arial";
pFont.Size = 16;
pGridLabel.Font = pFont as stdole.IFontDisp;
pGridLabel.Color = getRGB(0, 0, 0);
pGridLabel.LabelOffset = 2;
pGridLabel.set_LabelAlignment(esriGridAxisEnum.esriGridAxisLeft, false);
pGridLabel.set_LabelAlignment(esriGridAxisEnum.esriGridAxisRight, false);
```

```
                //设置IFormattedGridLabel的属性
                INumericFormat pNumericFormat;
                pNumericFormat = new NumericFormatClass();
                pNumericFormat.AlignmentOption =
esriNumericAlignmentEnum.esriAlignRight;
                pNumericFormat.RoundingOption =
esriRoundingOptionEnum.esriRoundNumberOfDecimals;
                pNumericFormat.RoundingValue = 2;
                pNumericFormat.ShowPlusSign = false;
                pNumericFormat.UseSeparator = true;
                pNumericFormat.ZeroPad = true;
                pFormattedGridLabel.Format = pNumericFormat as INumberFormat;
            //设置mapgrid的标签
                pMapGrid.LabelFormat = pGridLabel;
                //添加 mapgrid到布局视图
                IGraphicsContainer pGraphicsContainer ;
                IMapFrame pMapFrame;
                pGraphicsContainer = pActiveView as IGraphicsContainer;
                pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
                IMapGrids pMapGrids;
                pMapGrids = pMapFrame as IMapGrids;
                pMapGrids.AddMapGrid(pMapGrid);
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewBackground, null,
null);
        }
```

读者在新建一个地图格网对象后添加到布局视图中常常没有任何的显示效果，此时需要检查一下 pMeasuredGrid.XIntervalSize 和 pMeasuredGrid.YIntervalSize 等值是否设得太大以至无法看到。

5.6.4 MapGridFactory 对象

MapGridFactory 对象可以让程序员快速新建一个地图格网对象，这些新建的格网对象的属性被设置为缺省值。MapGridFactory 是一个抽象类，它唯一的接口是 IMapGridFactory，其中定义的 create 方法可以新建 MapGrid 对象。

MapGridFactory 的子类有 GraticuleFactory、IndexGridFactory、MeasuredGridFactory、CustomOverlayGridFactory 和 MgrsGridFactory 五种。它

们没有自己的接口，全部都是实现 IMapGridFactory。

下面是一个实际使用 MapGridFactory 对象产生一个地图格网的过程：

```
private void CreateMapGridByFactory()
{
    IActiveView pActiveView;
    IGraphicsContainer pGraphicsContainer;
    IMap pMap;
    pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
    pGraphicsContainer = pActiveView as IGraphicsContainer;
    pMap = pActiveView.FocusMap;
    IMapFrame pMapFrame;
    pMapFrame = pGraphicsContainer.FindFrame(pMap)   as IMapFrame;
    IMapGrids pMapGrids ;
    pMapGrids = pMapFrame as IMapGrids;
    //使用  MeasuredGridFactory作空间工厂
    IMapGridFactory pMapGridFactory ;
    pMapGridFactory = new MeasuredGridFactoryClass();
    IMapGrid pMapGrid;
    //产生个新的地图网格对象
    pMapGrid = pMapGridFactory.Create(pMapFrame);
    IMeasuredGrid pMeasuredGrid;
    pMeasuredGrid = pMapGrid as IMeasuredGrid;
    pMeasuredGrid.FixedOrigin = true;
    pMeasuredGrid.Units = pMap.MapUnits;
    pMeasuredGrid.XIntervalSize = 10;
    pMeasuredGrid.XOrigin = -180;
    pMeasuredGrid.YIntervalSize = 10;
    pMeasuredGrid.YOrigin = -90;
    //将地图网格添加到mapgrids中
    pMapGrids.AddMapGrid(pMapGrid);
    pActiveView.PartialRefresh(esriViewDrawPhase.esriViewBackground, null,
null);
}
```

5.7 MapSurround 对象

MapSurround 是与一个地图对象关联的一种用于修饰地图的辅助图形元素对象。它们的形状或数值会随着地图属性的变化而自动改变。如地图视图范围改变后，ScaleBar（比例尺）对象将会自动调整比例，ScaleBarText（比例尺文本）也会相应改变它的比例值，即 MapSurround 类型对象会监听 Map 对象的行为而做出相应反应。

在 ArcMap 中 Mapsurround 是由 MapSurroundFrame 对象管理的。所有的 MapSurround 被放置在布局视图上，而且也只能在这个视图上摆放。由于 MapSurround 与一个 Map 对象有直接的关联，因此可以使用 IMap::Mapsurrounds 来得到它，用户也可以通过 IMap::MapSurroundCount 来遍历一个给定的 Map 对象所有可以使用的 MapSurround 对象。

所有的 MapSurround 对象都支持 IMapSurround 接口，它定义了 MapSurround 对象一般的属性和方法，如使用 Name 属性可以得到某个 MapSurround 的名称，FitToBounds 方法可以改变一个 MapSurround 对象的尺寸。

MapSurround 类也实现了 IMapSurroundEvents 接口，用于触发与 MapSurround 相关的事件，如 AfterDraw、BeforeDraw 等。

5.7.1 图例对象

图例（Legend）是 MapSurround 对象群中最复杂的一个，这是因为它牵扯到太多其它的对象以生成一个美观合理的图例。图例是与一个 Map 对象中图层的着色操作（renderer）相关的对象，着色对象可以在地图上产生专题图。每个着色对象都有一个或者多个 LegendGroup（图例组），而这个数目则取决于地图有多少种着色方案。

每个 LegendGroup 都有一个或者多个 LegendClass（着色类）对象，而每个 LegendClass 代表了一个使用自身的符号和标签制作的图例分类。在 ArcMap 中将一个要素图层进行基于多属性的着色操作后，用户会在 TOC 窗体内发现如图 5-9 中的图形。

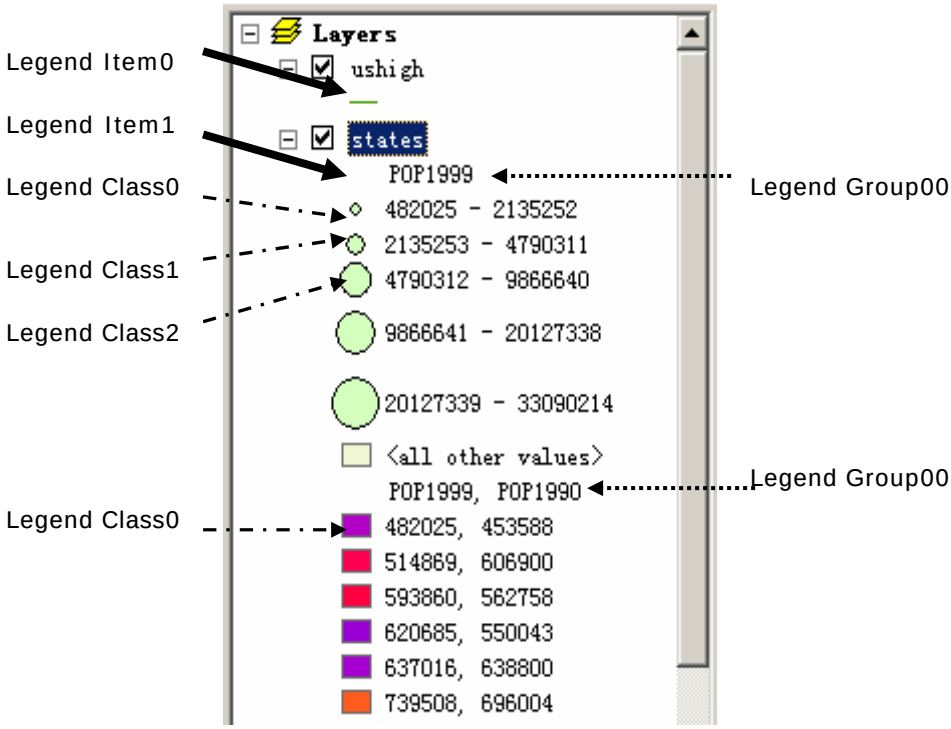


图 5-9　Legend 组成要素

从 TOC 中可以看出 Map 中有两个要素图层--ushigh 和 states。ushigh 图层使用了一种符号来设置它的要素，因此这个图层上所有的要素的外观都是一样的；states 使用了基于多属性的着色方法，其中基于 POP1999 属性使用的是分级作色法，而对 POP1990 属性使用的是唯一值法。

由于两个图层都使用了符号化，因而与 Map 对象相关的图例项有两个，即上图中的 legend item0 和 legend item1，每个 legend item 都和一个图层相关联。states 图层使用的是两个属性来进行符号化着色，它有两个 legend group，其中第一个 legend group 内拥有 5 个 legend class 对象，第二个 legend group 内也有多个 legend class。这个图清楚地解释了图例各组成要素之间的关系。

Legend 类的主要接口是 ILegend，使用它可以修改 legend 的属性和获得它的组成对象，如 ILegend::Item 可以用于获得一个地图图例内的某个条目。当 ILegend::AutoAdd 为 true 时，Map 对象加入新图层后，与 Map 关联的图例对象将会自动添加一个条目，而 AutoReorder 属性则可以让 Legend 中条目的顺序和地图中的图层顺序一致；AutoVisibility 属性为 true 的时候，如果与 ushigh 图层设置为不可见，那图例中的 item0 也将为不可见状态。

ILegend 接口还定义了方法可以让程序员灵活地添加、删除其中的条目，如 AddItem，ClearItems 等。

一般情况下，图例中的描述字符都是在 Patch（Patch 指每个 legend class 前面的用于显示着色符号的小图片）的右边，如果想让它们处于左侧，可以使用 IReadingDirection::RightToLeft 进行设置。

LegendItem 对象有多种类型，它本身是一个抽象类，拥有 HorizontalBarLegendItem、HorizontalLegendItem、NestedLegendItem 和 VerticalLegendItem 四个子类，关于这四个对象的不同表现形式可以在 ArcMap 里面查看。LegendItem 对象默认实现 ILegendItem 接口，这个接口定义了所有 LegendItem 的一般属性，如是否显示标题、是否显示标签，是否出现图层名以及 LegendItem 的宽度和高度等。

LegendItem 的四个子类都有各自相应的接口，如 HorizontalBarLegendItem 的接口为 IHorizontalBarLegendItem 等，使用这些接口可以方便的定制不同类型的图例条目。

LegendClassFormat 对象用于控制单个 legendItem 的外观，如 DescriptionSymbol、LinePatch 和 Patch 的属性等。LegendFormat 对象用于控制一个 legend 的属性，特别是 legend 内不同部分的间隔大小。

Patch 是一个 Legend Class 中的帮助描述要素着色的图片，如上图中的绿色方框、线和橘色的圆，都是一种 patch，这个对象只有两种形式：AreaPatch 和 LinePatch。ArcObjects 使用 LegendClassFormat 和 LegendFormat 对象来管理一个图例项内的 Patch 对象。

除了使用标准的 Patch 对象外，用户还可以使用 IPatch 接口定义的属性和方法来自己创建新的 patch 对象供图例对象使用。下面是一个新建自定义 patch 的例子：

```
private void CreatePatch(IGeometry pGeo)
{
    IPageLayout pPageLayout;
    pPageLayout = axPageLayoutControl1.PageLayout;
    IGraphicsContainer pGraphicsContainer;
    pGraphicsContainer = pPageLayout as IGraphicsContainer;
    IActiveView pActiveView;
    pActiveView = pPageLayout as IActiveView;
    IMap pMap;
    pMap = pActiveView.FocusMap;
    //从 pMap对象直接得到与之相关联的 MapSurround对象 legend
    ILegend pLegend;
    pLegend = pMap.get_MapSurround(0) as ILegend;
    //获得图例内的一个条目
```

```
            ILegendItem pLegendItem;
            pLegendItem = pLegend.get_Item(0);
            ILegendFormat pLegendFormat;
            pLegendFormat = pLegend.Format;
            //新建一个Patch对象
            IPatch pPatch;
            pPatch = new AreaPatchClass();
            pPatch.Geometry = pGeo;
            pLegendFormat.DefaultAreaPatch = pPatch as IAreaPatch;
            pLegend.Refresh();
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
        }
```

上面这个过程将从 PageLayoutControl 控件地图的图例对象，然后改变其中一个图例条目的 Patch 属性。

5.7.2 指北针对象

MarkerNorthArrow 是一种用于指示地图空间方位的图形，它其实是 ESRI North 字库中的字符符号，字库中的任何一种字体的符号都可以当作指北针使用。

MarkerNorthArrow 从抽象类 NorthArrow 继承而来，它是一个 MapSurround 对象。MarkerNorthArrow 对象的两个主要接口是 INorthArrow 和 IMarkerNorthArrow。INorthArrow 接口可以设置指北针对象的一般属性，如颜色、尺寸和引用位置，所谓引用位置是一个 Point 对象，它是地图对象计算出来的指北点，当地图被旋转的时候，指北点会自动发生变化，指北针对象会一直指向这个点。

IMarkerNorthArrow 接口定义了一个属性 MarkerSymbol，它用于设置指北针的符号，在默认状态下这个符号属于 ESRI North 字体

5.7.3 比例尺对象

地图是现实世界的抽象反映，地图上的图形与现实存在的物体间存在一定的比例关系，为了显示这种比例，所有的地图都需要设置一个比例尺来说明地图上的 1cm 或者 1 英寸代表现实世界的实际长度。

在 ArcMap 的布局视图中用户可以使用多种预定好的比例尺对象，这就意味着用户也可以自己制作比例尺对象供地图使用。ScaleBar 对象也是一种 MapSurround，它有多个子类，如 ScaleLine、SinglefillScaleBar 和

DoublefillScaleBar 等，这些类都实现了 IScaleBar 和 IScaleMarks 接口。

IScaleBar 接口可以管理一个比例尺对象的大部分属性，如比例尺颜色、高度，它也定义了管理比例尺对象上 Label 的属性，如 Labelsymbol、LabelPosition 等，它们分别用于设置比例尺中的标识字符符号和位置。IScaleMarks 接口负责管理与一个比例尺相关的单个标记（mark）的属性，如高度、符号和位置等。

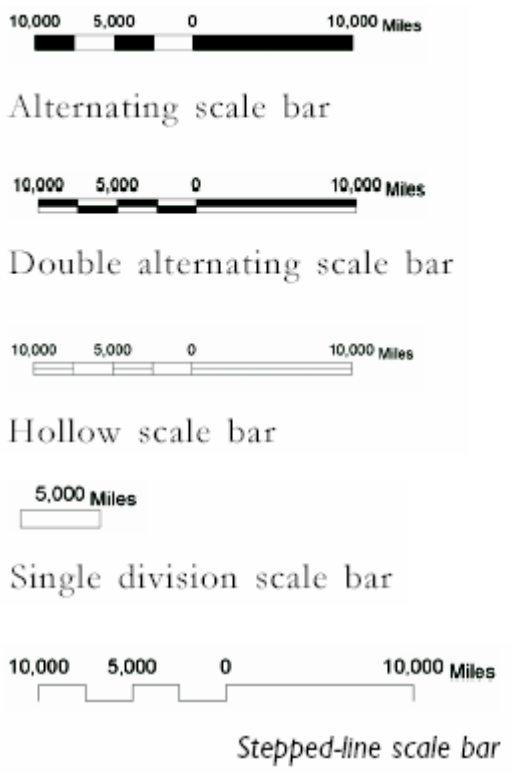


图 5-10　ScaleBar 的种类

SinglefillScaleBar 是个抽象类，它的对象使用一种符号来显示比例尺。这个类实现的接口是 ISinglefillScaleBar，其子类为 SingleDivisionScaleBar，子类没有自己特殊的接口。

DoublefillscaleBar 是使用两种符号来交叉显示比例尺的，其接口是 IDoublefillScaleBar，使用它的 FillSymboll 和 FillSymbol2 属性可以设置 DoublefillScaleBar 中的两种符号。DoublefillScaleBar 有三个子类，即 AlternatingScaleBar、DoubleAlternatingScaleBar 和 HollowscaleBar。

ScaleLine 对象是使用步进线（stepped-line）来显示比例尺的，它本身是一个组件类，实现了 IScaleLine 接口。SteppedScaleLine 对象是 ScaleLine 的子类。

图 5-10 中展示了各种 ScaleBar 的图示。图 5-11 为比例尺对象之间的结构关系。

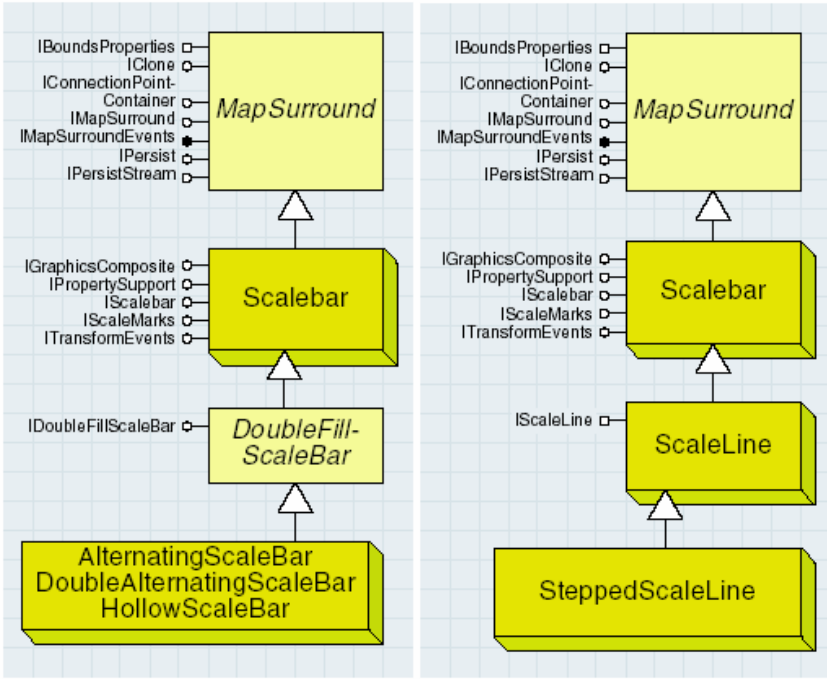


图 5-11 比例尺对象之间的结构关系

下面介绍在 PageLayoutControl 控件中添加一个比例尺对象的方法，这个方法需要传入两个参数：pEnv 用于设置比例尺的放置范围，strBarType 用于设置比例尺对象的类型：

```
private void AddSacleBar(IEnvelope pEnv, int strBarType )
{
    IScaleBar pScaleBar;
    IMapFrame pMapFrame;
    IMapSurroundFrame pMapSurroundFrame;
    IMapSurround pMapSurround;
    IElementProperties pElementPro;
    //产生一个UID 对象，使用它产生不同的MapSurround对象
    UID pUID = new UIDClass();
    pUID.Value = " esriCarto.scalebar";
    IPageLayout pPageLayout;
    pPageLayout = axPageLayoutControl1.PageLayout;
    IGraphicsContainer pGraphicsContainer;
    pGraphicsContainer = pPageLayout as IGraphicsContainer;
    IActiveView pActiveView ;
    pActiveView = pGraphicsContainer as IActiveView;
```

```
IMap pMap;
pMap = pActiveView.FocusMap;
//获得与地图相关的MapFrame
pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
//产生MapsurroundFrame
pMapSurroundFrame = pMapFrame.CreateSurroundFrame(pUID, null);
//依据传入参数不同使用不同类型的比例尺
switch(strBarType)
{
    case 0:
        //单线交互式比例尺
        pScaleBar = new AlternatingScaleBarClass();
        break;
    case 1:
        //双线交互式比例尺
        pScaleBar = new DoubleAlternatingScaleBarClass();
        break;
    case 2:
        //中空式比例尺
        pScaleBar = new HollowScaleBarClass();
        break;
    case 3:
        //线式比例尺
        pScaleBar = new ScaleLineClass();
        break;
    case 4:
        //分割式比例尺"
        pScaleBar = new SingleDivisionScaleBarClass();
        break;
    case 5:
        //阶梯式比例尺
        pScaleBar = new SteppedScaleLineClass();
        break;
    default:
        pScaleBar = new ScaleLineClass();
        break;
}
//设置比例尺的属性
```

```
            pScaleBar.Division = 4;
            pScaleBar.Divisions = 4;
            pScaleBar.LabelGap = 4;
            pScaleBar.LabelPosition = esriVertPosEnum.esriAbove;
            pScaleBar.Map = pMap;
            pScaleBar.Name = "比例尺";
            pScaleBar.Subdivisions = 2;
            pScaleBar.UnitLabel = "千米";
            pScaleBar.UnitLabelGap = 4;
            pScaleBar.UnitLabelPosition = esriScaleBarPos.esriScaleBarAbove;
            pScaleBar.Units = esriUnits.esriKilometers;
            pMapSurround = pScaleBar;
            pMapSurroundFrame.MapSurround = pMapSurround;
            pElementPro = pMapSurroundFrame as IElementProperties;
            pElementPro.Name = "my scalebar";
            //将 MapSurrooundFrame对象添加到控件中
            axPageLayoutControl1.AddElement(pMapSurroundFrame as IElement,
pEnv,Type.Missing,Type.Missing,0);
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
        }
```

5.7.4 比例文本对象

比例尺能够用图形的方式显示出地图上的单位长度在现实世界的距离，但用户一般都希望在地图上能够获得一个明确的比例值。ArcObjects 提供比例文本对象 ScaleText 就可以满足这个要求。ScaleText 对象在本质上是一个文本元素，但是它会随着相关地图的变化而改变比例值。

ScaleText 类实现了 IScaleText 接口，它定义了文本的格式，如 symbol、style 等。用户也可以通过 text 只读对象来得到比例文本的字符值。

通过对 ScaleText 对象不同属性的设置，程序员可以定制任何长度单位之间的比例，如 cm:km；Inch:feet 等。

本节介绍的 MapSurround 对象，也是一种图形元素，而且是被 MapSurroundFrame 管理的图形元素，它们与其他元素最大的不同在于其本身是与一个 Map 对象紧密相关的。当 Map 对象的方向、比例或其中的图层对象等变化时，不同类型的 MapSurround 对象依据自己的方法做出自动反应。

5.8 使用 Style 对象

在 MapSurround 一节中列举的多种不同样式的指北针、比例尺和比例文本，熟悉 ArcMap 制图的读者肯定知道这些对象都是来自 ArcMap 中的样式库。打开 ArcMap，选择“Tools”菜单的“Style”命令，在右侧弹出的菜单中选择“Style Manager”，会出现如图 5-12 中的对话框。选择“North Arrows”后，用户在对话框右边的列表框中看到 ArcMap 程序预定义的指北针对象。这些统统称为样式（style）对象。实际上，ArcMap 里面存在很多预定义的样式对象，比如颜色、符号、图例和各种符号等。当然，读者也完全可以新建自己的样式对象满足特殊的需要。

样式通常按照功能被分为多个类型。例如常常在交通运输中使用的样式被划分为“交通类型”，这些样式都保存在一个 style 文件中，读者可以在＜ArcGIS 安装目录＞＼Bin＼Styles 文件夹中寻找到这些 Style 文件。

ArcMap 程序最常使用的符号和地图元素都保存在 ESRI.style 文件中，Styles 文件夹中也有其他的 style 文件，如 Weather.style、Petrolenum.style 等，这些样式是为了满足不同行业的需求而制作的，使用时它们都需要被引入到 ArcMap 中。

一个样式由多个样式条目（style gallery item）组成，这些 Style 条目提供了得到单个地图元素或符号的方法。相似的条目被组织成样式类（style gallery class）。如在图 5-12 中，鼠标点击的 Scale Bars 是一个样式类，而右边的符号和元素，是组成这个样式类的样式条目。一个样式类中的条目可以依据类型的差异进行分组（Categories），如 Scale Bars 这个样式类的样式条目被分为 ScaleLine、HollowScaleBar 等多个组。

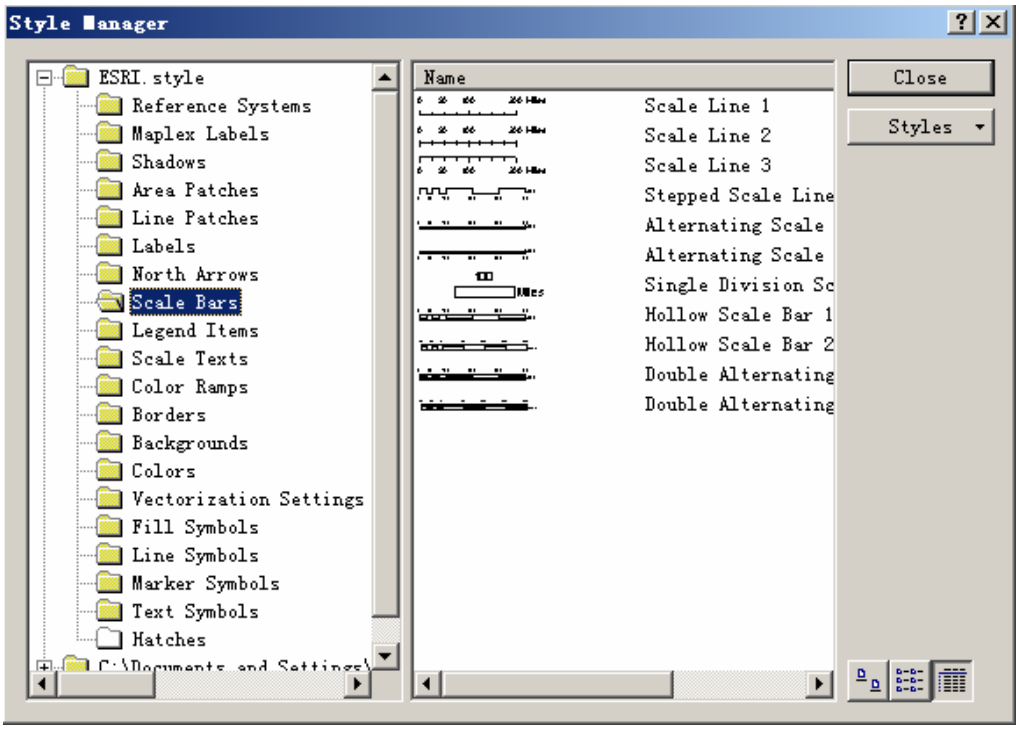


图 5-12　Style Manager 对话框

5.8.1 StyleGallery 对象

StyleGallery 是一个与文档对象相关的 Style 的集合对象，它代表了一个 Style 文件，通过这个对象程序员可以将一个 Style 文件中的样式取出来供系统使用。StyleGallery 类默认实现 IStyleGallery 接口，这个接口定义了操作一个样式中的种类、样式类和样式条目的方法和属性，如添加、删除、更新样式条目和载入新的 Style 文件。

IStyleGallery::Class 属性可以使用一个给定的索引值得到 Style 文件中的某个 StyleGalleryClass 对象，而 ClassCount 属性则会返回这个 Style 对象中所有的样式类数目；Categories 可以返回一个样式类中的样式种类。

使用 IStyleGallery::Items，可以在给定样式类名、style 文件路径和种类名称的情况下返回符合这些条件的样式条目集对象。下面是一个获得样式条目的例子：

```
private void StyleGalleryDemo()
{
    IStyleGallery pStyleGallery;
    pStyleGallery = new StyleGalleryClass();
```

```
            pStyleGallery.LoadStyle(@"C:\Program Files\ArcGIS\Styles\esri.style"," ");
            for(int i = 0;i<= pStyleGallery.ClassCount-1; i++)
            {
                MessageBox.Show(pStyleGallery.get_Class(i).Name);
            }
            IEnumStyleGalleryItem pEnumStyleGalleryItem;
            IStyleGalleryItem pStyleGalleryItem;
            //使用items属性获得esri.style文件内符合条件的样式条目
            pEnumStyleGalleryItem = pStyleGallery.get_Items("Scale Bars", @"C:\Program
Files\ArcGIS\Styles\esri.style", "hollowscalebar");
            pEnumStyleGalleryItem.Reset();
            pStyleGalleryItem = pEnumStyleGalleryItem.Next();
            //遍历所有被符合条件的样式条目
            while(pStyleGalleryItem != null)
            {
                MessageBox.Show(pStyleGalleryItem.Name);
                pStyleGalleryItem = pEnumStyleGalleryItem.Next();
            }
        }
```

StyleGallery 对象还实现了 IStyleGalleryStorage 接口，这个接口提供了在 Stylegallery 对象中获得一个 Style 文件的指针的方法，它也提供了方法让程序员能够添加或删除 Style 文件。

IStyleGalleryStorage::DefaultStylePath 将会返回 Style 文件的缺省目录。IStyleGalleryStorage::TargetFile 属性允许程序员新建一个 Style 文件作为添加、删除和更新样式条目的目标文件。IStyleGalleryStorage::CanUpdate 用于确定是否允许改变一个 Style 文件。

StyleGalleryItem 对象代表了一个具体的样式条目，它包含着一个地图元素或符号以及一些相关信息，其类实现了 IStyleGalleryItem 接口。IStyleGalleryItem::Category 属性确定条目在样式类中的类别，而 Item 属性是一个 object 类型对象，它要么是一个符号，要么是一个元素。

下面是一个新建 style 文件以及插入一个 StyleGalleryItem 的例子：

```
    private void AddStyleItem()
    {
        IStyleGallery pStyleGallery;
        pStyleGallery = new StyleGalleryClass();
        IRgbColor pRgbColor;
```

```
            pRgbColor = new RgbColorClass();
            pRgbColor.Red = 255;
            pRgbColor.Green = 0;
            pRgbColor.Blue = 0;
            //新建一个StyleGalleryItem对象，设置其属性
            IStyleGalleryItem pStyleItem;
            pStyleItem = new StyleGalleryItemClass();
            pStyleItem.Name = " Red";
            pStyleItem.Category = " Defalut";
            pStyleItem.Item = pRgbColor;
            IStyleGalleryStorage pStyleStorage;
            pStyleStorage = pStyleGallery as IStyleGalleryStorage;
            //改变目标文件，如果改文件不存在，将会新建一个
            pStyleStorage.TargetFile = @"D:\temp\test.style";
            //添加一个样式条目
            pStyleGallery.AddItem(pStyleItem);
        }
```

执行完这段代码后，读者会发现在 D 盘 temp 目录下出现了一个 test.style 文件，使用 ArcMap 的样式管理器来查看它，在其 Colors 样式类中，将会看到添加进去的颜色样式条目。

5.8.2 StyleSelector 对象

尽管使用 StyleGallery 对象能够选择系统中已经存在的样式，但是这种选择并不方便，至少不具备可视化图形让用户选择。因此 ArcObjects 提供了 StyleSelector（样式选择器）对象，它是一个对话框，可以让用户用直观的方式来进行样式的选择。

StyleSelector 分为两大类型：一种是 MapGridselector 对象，它是用来设置地图格网的；另一种是用于设置 MapSurround 的，如 BackgroundSelector、BorderSelector、LabelStyleSeleCtor、LegendItemSelector、NorthArrowSelector、ScaleBarSelector 和 ScaleTextSelector 对象。

IStyleSelector 接口定义的方法让程序员可以使用这些对话框进行选择，IStyleSelector::DoModal 方法用于打开对话框，而使用 IStyleSelector::Getstyle 方法则用于选择具体的样式对象。

下面是一个使用例子：

```
private object StyleSelectorScaleBar()
{
    IStyleSelector pStyleSelector;
    pStyleSelector = new ScaleBarSelectorClass();
    bool ok ;
    ok = pStyleSelector.DoModal(0);
    if (ok)
    {
        object oStyle;
        oStyle = pStyleSelector.GetStyle(0);
        return oStyle;
    }
    return null;
}
```

上面的函数可以非常轻松地返回一个颜色对象。

但读者需要注意的是，本文介绍的用于插入 MapSurround 对象的各种选择对话框对象有些是没有办法在 PageLayoutControl 控件中使用的，这是一个软件的 BUG，至少在 9.1 版本的控件中还没有得到解决。

5.8.3 IStyleDialog 接口

IStyleDialog 接口定义了属性和方法来控制 Style Manager 和 Style References 对话框，该接口只提供了一个属性 Title 和一个方法 DoModal，前者用于设置对话框的标题，后者用于打开对话框。实现该接口的类有两个：StyleManagerDialog 和 StyleReferencesDialog。

StyleManagerDialog 用于管理 Map 文档中引用的样式及其中的样式条目（style items），Style Manager 对话框如图 5-12 所示；**StyleReferencesDialog** 用于管理 ArcMap 引用的样式文件，Style References 对话框如图 5-13 所示。

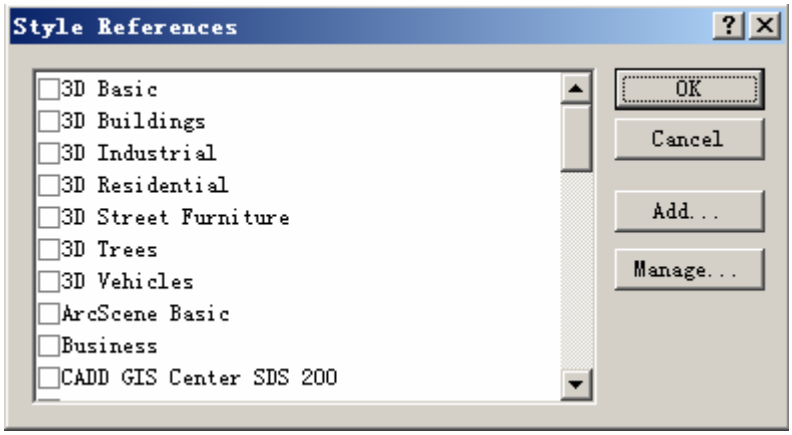


图 5-13　Style References 对话框

以下代码展示了如何使用 StyleManagerDialog 来打开样式管理器：

```
private void mnuStyleManager_Click(object sender, System.EventArgs e)
{
    IStyleGallery pStyleGallery ;
    IStyleDialog pStyleDialog;
    string pStyleGalPath ;
    pStyleGalPath = @"C:\Program Files\ArcGIS\Styles\ESRI.ServerStyle" ;
    pStyleGallery = new ServerStyleGalleryClass();
    pStyleGallery.ImportStyle (pStyleGalPath);
    pStyleDialog = new StyleManagerDialogClass();
    pStyleDialog.DoModal(pStyleGallery, 0);
}
```

以下代码展示了如何使用 StyleReferencesDialog 来打开样式引用对话框：

```
private void mnuStyleReferencesDialog_Click(object sender, System.EventArgs e)
{
    IStyleGallery pStyleGallery ;
    IStyleDialog pStyleDialog;
    string pStyleGalPath ;

    pStyleGalPath = @"C:\Program Files\ArcGIS\Styles\ESRI.ServerStyle" ;
    pStyleGallery = new ServerStyleGalleryClass();
    pStyleGallery.ImportStyle (pStyleGalPath);
    pStyleDialog = new StyleReferencesDialogClass();
    pStyleDialog.DoModal(pStyleGallery, 0);
```

}

5.9 IDentifyDialog 对象

在 ArcMap 和 ArcCatalog 程序中，当用户点击 Identify 按钮后，就会弹出一个属性页面，使用鼠标在屏幕上点击地理数据，属性页面会自动将鼠标点击到的要素属性显示出来。这个功能在 ArcObjects 中可以借助 IDentifyDialog 对象实现，如图 5-14 所示，这个功能只能在 ArcObjects Desktop 版本中才能实现，它无法移植到 ArcEngine 等平台上。

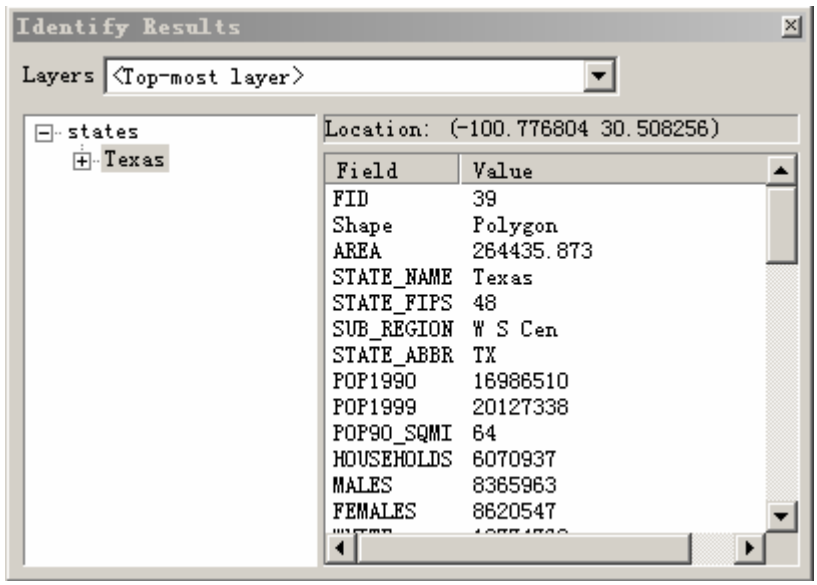


图 5-14　IDentifyDialog 对话框

下面的代码使用 IDentifyDialog 对象实现了 Identify 工具。

IdentifyTool.cs

```
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.CartoUI;
namespace ControlsApplication
{
    public sealed class IdentifyTool: BaseTool
    {
        private IHookHelper m_HookHelper = new HookHelperClass();
        public IdentifyTool()
```

```
        {
            base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),"informa
tion.bmp"));
            base.m_cursor = new
System.Windows.Forms.Cursor(GetType().Assembly.GetManifestResourceStream(GetType(),"
CROSS_R.CUR"));
            base.m_caption="Identify";
            base.m_category="Identify";
            base.m_message="Identify the information of the selected feature";
            base.m_name="CustomCommand_Identify";
            base.m_toolTip="Identify";
        }

        public override void OnCreate(object hook)
        {
            m_HookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            IActiveView activeview = m_HookHelper.ActiveView;
            IEnumLayer m_EnumLayer;
            IIdentifyDialogProps m_IdentifyDP;
            IIdentifyDialog m_IdentifyD = new IdentifyDialogClass();
            m_IdentifyD.Map = m_HookHelper.FocusMap;
            m_IdentifyD.Display = activeview.ScreenDisplay;
            m_IdentifyD.ClearLayers();
            m_IdentifyDP = m_IdentifyD as IIdentifyDialogProps;
            m_EnumLayer = m_IdentifyDP.Layers;
            m_EnumLayer.Reset();
            ILayer m_Layer = m_EnumLayer.Next();
            while(m_Layer != null)
            {
                m_IdentifyD.AddLayerIdentifyPoint(m_Layer,X,Y);
                m_Layer = m_EnumLayer.Next();
            }
            m_IdentifyD.Show();
```

```
        }
    }
}
```

本章的内容初步揭示了 ArcMap 结构的隐藏的秘密，从最常接触的 Map 对象开始，本书使用了较大的篇幅介绍 Map 对象和数据的中介——图层对象。无论在哪一种视图中，可以出现的图形要么是要素等地理数据，要么是图形元素，两者的差别在于前者存储在地理数据库或数据文件中，而后者保存在一个文档文件里面。

除了看得到的图形元素，在 ArcMap 的背后，还藏着一些框架元素对象，它们充当了相应对象的容器，如 MapFrame 管理 Map 对象等。框架元素管理的对象，除了 Map，还有用于地图制版时使用的一些辅助图形，这些被称为 ManSurround 的对象，可以随着地图的变化而自动改变属性。

无论是 MapSurround，还是本书下一章将要介绍的颜色对象、符号对象，它们都可以称为某种样式（Style）。在 ArcMap 中存在一些已经制作好的样式，使用 StyleGallery 和 StyleSelector 对象，可以产生和选择这些样式对象。

本章最后介绍了如何使用 IdentifyDialog 对象来获取地理数据的信息。这些内容都是学习 ArcObjects 中非常重要的部分，只有熟悉了这个初步被揭示的结构，才能在理解 ArcObjects 时游刃有余。

6 空间数据的符号化

地图符号是表达空间数据的基本手段，是地图的语言单位，是可视化表达地理信息内容的基础工具。它不仅能表示事物的空间位置、形状、质量和数量特征，而且还可以表示各事物之间的相互联系及区域总体特征。地图符号由形状不同、大小不一、色彩有别的图形和文字组成，它既是地图的语言，也是一种图形语言。地图符号不仅具有确定客观事物空间位置、分布特点以及质量和数量特征的基本功能，而且还具有相互联系和共同表达地理环境各要素总体的特殊功能。

地图的符号化决定着地图数据最终以何种“面目”呈现在用户或者读者面前，因此，符号化对地图来说有至关重要的意义。ArcObjects 提供了丰富的符号组件来满足地图设计中的各种需求。这些组件对象包括 Renderer、Color、Symbol 三大系列。地图符号化就是通过多个组件对象间的合作来完成的。

学习完本章后，读者将了解以下知识：

- 如何使用颜色对象
- 如何使用符号对象
- 如何对要素进行着色，以生成不同类型的专题图
- 如何进行地图标注
- 使用 Feedback 对象进行控件交互

6.1 颜色对象

颜色（Color）是现实世界中最普遍的事物属性，为了满足不同行业的需要，人们使用了不同的颜色模型来模拟现实的颜色。作为一个程序员可能对于 RGB 颜色模型更清楚一点，很多读者都知道只要改变红色、绿色和蓝色的不同值，就可以调配出丰富的颜色出来，当然，这些颜色的数值必须处于 0 到 255 之间。对于印刷行业而言，RGB 颜色模型是不够的，它们使用的是 CMY 颜色模型。

颜色的模型有多种，如 RGB 颜色模型、CMY 颜色模型（主要用于印刷中）、HSV 颜色模型（由色调 Hue、饱和度 Saturation 和 Value 值组成，这个模型基于一个限制范围的颜色轮带）、GrayColor（灰度颜色）、CIELAB 颜色（在 ArcObjects 中使用，与设备无关）。

- **RGB 模型**

可见光谱中的大部分，都可以用各种比例和强度的红、蓝、绿（RGB）三种色光混合而成。这三种颜色覆盖时，会产生青色、洋红色和黄色。RGB 模型的主要属性也是确定 Red、Blue 和 Green 的值，如图 6-1 所示。

因为 Red、Green、Blue 值加在一起后会变成白色，即所有光都反射回眼睛--所以 RGB 三色称为加色。加色通常用在照明、电视和电脑屏幕。屏幕上的颜色，就是光线射过红、绿、蓝磷光剂所产生的。

红色、蓝色和绿色的混合会产生不同的效果，如：

1. 如果 RGB 的值相等，则产生的颜色是灰色。
2. 如果 RGB 都是 255，则产生的颜色是白色。
3. 如果 RGB 都是 0，那么产生的颜色是黑色。

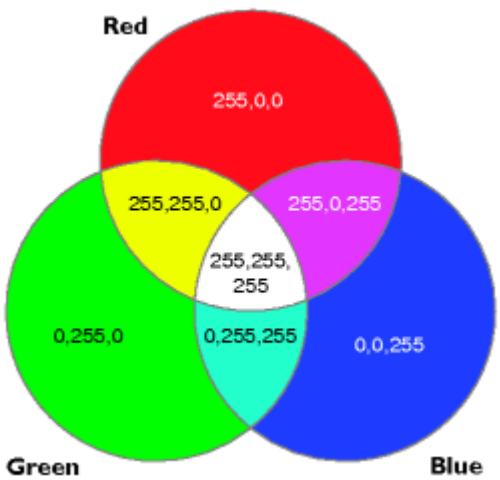


图 6-1 RGB 模型

- **CMYK 模型**

CMYK 模型以打印在纸上的油墨的光线吸收特性为基础。当白光照射到半透明油墨上时，某些可见光波长被吸收，而其他波长则被反射回眼睛。

理论上，纯青色（C）、洋红色（M）和黄色（Y）色素在合成后可以吸收所有光线并产生黑色。如图 6-2 所示。由于打印油墨都包含一些杂质，因此这三种油墨实际生成土灰色，它们必须与黑色（K）油墨合成才能生成真正的黑色，为了避免与蓝色混淆，黑色用 K 而非 B 表示。这些油墨混合重现颜色的过程称为四色印刷。

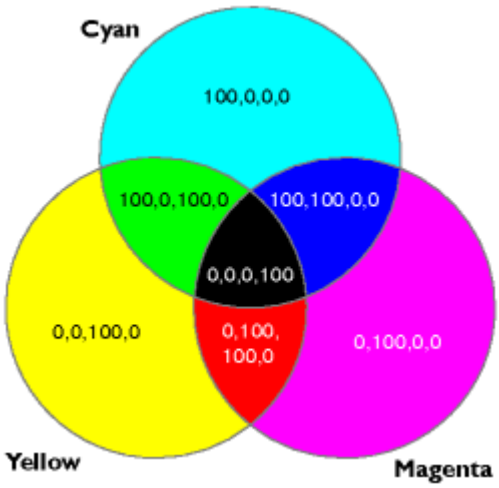


图 6-2 CMYK 模型

- **HSV 模型**

HSV 颜色模型对应于圆柱坐标系的一个圆锥形子集。圆锥的顶面对应于 V = 1，代表的颜色较亮。色彩 H 由绕 V 轴的旋转角给定，红色对应于角度 0º，绿色对应于角度 120º，蓝色对应于角度 240º。如图 6-3 所示。

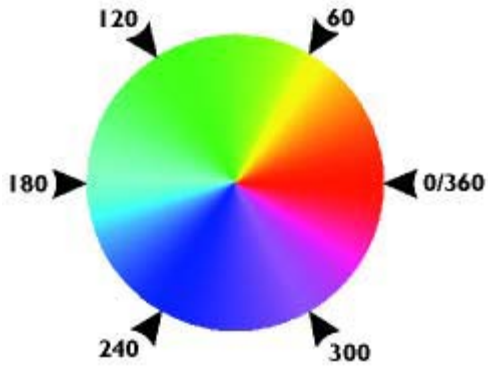


图 4-3　HSV 模型

饱和度 S 取值从 0 到 1，由圆心向圆周过渡。由于 HSV 颜色模型所代表的颜色域是 CIE 色度图的一个子集，它的最大饱和度的颜色的纯度值并不是 100%。在圆锥的顶点处，V＝0，H 和 S 无定义，代表黑色；圆锥顶面中心处 S＝0，V＝1，H 无定义，代表白色，从该点到原点代表亮度渐暗的白色，即不同灰度的白色。任何 V=1，S=1 的颜色都是纯色。

- **Cray 模型**

灰度模型是没有彩色的，灰度图像由 8 位信息组成，并使用 256 级的灰色来模拟颜色的层次。如图 4-4 所示。灰度图像的每个像素有一个 0（黑色）到 255（白色）之间的亮度值。灰度值也可以用黑色油墨覆盖的百分比来表示（0%等于白色，100

%等于黑色）。使用黑白或灰度扫描仪产生的图像常以“灰度”模式显示。用户可以很容易地将位图模式和彩色图像转换为灰度。



图 6-4 Gray 模型

6.1.1 Color 对象

Color 对象是一个抽象类，它有 5 个颜色子类，即 CmykColor、RGBColor、HSVColor、HLSColor 和 GrayColor，它们可以使用 IColor 接口定义的方法设置颜色对象的基本属性。对于一种特定的颜色而言，它在不同的颜色模型中的表示方法是不一样的，因此使用颜色对象时，需要使用 IColor 的子接口产生的具体对象。

IColor::RGB 属性会返回一个 LONG 数值，而不是 RGB 模型需要的 R、G、B 上的三个值，如白色的 RGB 属性返回值为 16777215，而不是用户需要的 255、255、255 三个值。如果采用 ArcObjects 的颜色选择对话框选取一个颜色后使用，程序员就必须使用 RGB 值的写法。下面是两种数值方式互相转换的函数。

- **RGB 值转换为 LONG 值**

```
private long RGBToLong(int Red, int Green ,int Blue )
{
    return Red + (0x100 * Green) + (0x10000 * Blue);
}
```

- **LONG 值转换为 RGB 值**

```
private short[] LongtoRGB(long RGBlong)
{
    short[] pbyte = new short[3] ;
    pbyte[0] = (short) (RGBlong % 0x100) ;
    pbyte[1] = (short) ((RGBlong / 0x100) % 0x100);
    pbyte[2] = (short) ((RGBlong / 0x10000) % 0x100);
    return pbyte;
}
```

在 ArcObjects 中最常使用的两种颜色模型是 RGB 和 HSV，RGB 类实现

IRgbColor 接口，而 HSV 类则实现 IHsvColor 接口，两个接口分别定义了设置一个 RGB 对象和 HSV 对象需传递的值。

下面是两个颜色构造器，分别用于产生 HSV 对象和 RGB 对象。

- **HSV 模型颜色构造器**

```
private IHsvColor HSVColor(int hue , int saturation , int val )
{
    //定义一个IHSVColor类型对象
    IHsvColor pHsvColor;
    pHsvColor = new HsvColorClass();
    //设置它的值
    pHsvColor.Hue = hue;
    pHsvColor.Saturation = saturation;
    pHsvColor.Value = val;
    return pHsvColor;

}
```

- **RGB 颜色构造器**

```
private IRgbColor getRGB(int r,int g,int b)
{
    IRgbColor pColor;
    pColor = new RgbColorClass();
    pColor.Red = r;
    pColor.Green = g;
    pColor.Blue = b;
    return pColor;
}
```

ArcObjects 中的颜色对象的新建和设置方法大同小异，与上面的例子相似。

6.1.2 颜色对话框

除了使用编程的方法得到颜色对象外，用户也可以使用 ArcOjects 中的颜色对话框来获取需要的颜色，而且使用这些可视化对话框更加直观，毕竟用户很少能够知道自己需要的颜色的准确 RGB 值或 CMYK 值。

ArcObjects 中提供了几种颜色对话框供程序员使用。

- **颜色板对象 Colorpalatte**

如图 6-5 所示。ColorPalette 颜色板对象一共排列了 120 种颜色供用户使用。ColorPalette 类实现了两个接口：IColorPalette 和 ICustomColorPalette。



图 6-5 调色板（ColorPalette）

IColorPalette 接口定义了 Color 属性和 TrackPopupMenu 方法，使用这个接口可以从对话框中获得一个颜色对象。下面是使用调色板对象取出一个颜色的例子：

```
IColor pColor;
pColor = new RgbColorClass();
pColor.RGB = 255 ;
//新建一个颜色板对象
IColorPalette pPalette ;
pPalette = new ColorPaletteClass();
//定义一个范围结构
tagRECT pRect = new tagRECT() ;
pRect.left =10;
pRect.top =10;
pPalette.TrackPopupMenu(ref pRect, pColor, false, 0);
//获得新的颜色
pColor = pPalette.Color;
```

读者可以将上面的代码片段改写为一个函数使用。

- **颜色选择器对象**

颜色选择器（ColorSelector）对象提供了另一种选择颜色的方法，不过这种选择方式在精确度上更高，用户可以点击对话框上方右边的小箭头，选择使用 RGB

或 CMYK 等多种模型，通过拖曳颜色带或者直接输入具体颜色值的方法返回颜色对象，如图 6-6 所示。



图 6-6 颜色选择器

下面是一个使用颜色选择器对象获取颜色对象的代码段：

```
//Set the initial color to be diaplyed when the dialog opens
IColor pColor ;
pColor = new RgbColorClass();
pColor.RGB = 255;
IColorSelector pSelector;
pSelector = new ColorSelectorClass();
pSelector.Color = pColor;
// Display the dialog
if (pSelector.DoModal(0))
{
    IColor pOutColor;
    pOutColor = pSelector.Color;

 }
```

- **颜色浏览器对象**

和颜色选择器，颜色浏览器（ColorBrowser）对象提供了更灵活的方法来获取颜色对象，即通过简单的鼠标点击，就可以获得颜色对象，如图 6-7 所示。



图 6-7 颜色浏览器

下面是一个使用颜色浏览器对象对象获取颜色对象的代码段：

```
IRgbColor pInitColor;
IRgbColor pNewColor;
IColorBrowser pColorBrowser;
```

```
bool bColorSet;
pInitColor = new RgbColorClass();
// the dialog will open with red as a default.
pInitColor.Red = 255;
pColorBrowser = new ColorBrowserClass();
pColorBrowser.Color = pInitColor;
bColorSet = pColorBrowser.DoModal(0);
if (bColorSet) pNewColor = (IRgbColor) pColorBrowser.Color;
```

6.1.3 ColorRamp 对象

制作地图专题图的过程中，需要的颜色常常不是一种，而是随机或有序产生的一个颜色带。如果某个着色对象需要 100 种颜色，程序员肯定不能逐个产生出来，而是可以使用 ArcObjects 提供的颜色带（ColorRamp）对象。

ColorRamp 类的对象可以产生颜色带，这个类实现了 IColorRamp，定义了一系列颜色带的公共方法，如 Size（产生多少种颜色），Colors（颜色带 IEnumColor）。下表是四种颜色带模型的例子。

RandomColorRamp	使用 HSV 颜色模型来确定一串颜色
PresetColorRamp	预设的颜色模式，可存储 13 种颜色
AlgorithmicColorRamp	用起始颜色、终止颜色确定一个颜色带，起始、终止颜色使用 HSV 模型
MultiPartColorRamp	叠加产生颜色带

- **AlgorithmicColorRamp 颜色带**

AlgorithmicColorRamp 是通过起止颜色来确定多个在这两个颜色之间的色带，因此它产生的颜色带还是有章可循的。AlgorithmicColorRamp 类实现了两个接口：IColorRamp 和 IAlgorithmicColorRamp，两个接口之间是接口继承关系，后者包含了前者所有的方法和属性。在产生颜色带对象的过程中，使用后一个接口就足够了。

下面是产生一个颜色带的代码的函数，需要传入的参数是起始颜色、终止颜色和产生的颜色数量：

```
private IEnumColors CreateColorRamp(IColor fromColor, IColor toColor, int count)
{
    IAlgorithmicColorRamp pRampColor;
    //产生一个 AlgorithmicColorRamp对象
    pRampColor = new AlgorithmicColorRampClass();
    //传入起止颜色
```

```
        pRampColor.FromColor = fromColor;
        pRampColor.ToColor = toColor;
        pRampColor.Size = count;
        bool ok = true;
        pRampColor.CreateRamp(out ok);
        //返回的是一个颜色枚举型对象
        return pRampColor.Colors;
    }
```

上面的函数返回的是一个颜色枚举对象，如果要获取其中的某个颜色，可以使用下面的方法：

```
IEnumColors pEnumColors;
pEnumColors = CreatColorRamp(...);
IColor pColor;
pColor = pEnumColors.Next();
```

- **IRandomColorRamp 颜色带**

在 UniqueBreak 着色中系统使用的多是随机颜色，即颜色带中的颜色没有规律可循。ArcObjects 也提供了 RandomColorRamp 对象供程序员产生随机颜色带，RandomColorRamp 也需要设定一个范围，但是这个范围是 HSV 颜色模型的，颜色将在这个范围内随机出现。下面是一个随机颜色带的代码段：

```
IEnumColors pEnumColors;
IColor pColor;
IRandomColorRamp pRandomColor;
pRandomColor = new RandomColorRampClass();
pRandomColor.StartHue = 140;
pRandomColor.EndHue = 220;
pRandomColor.MinValue = 35;
pRandomColor.MaxValue = 102;
pRandomColor.MinSaturation = 32;
pRandomColor.MaxSaturation = 245;
//产生的随机颜色数目
pRandomColor.Size = 12;
//pEnumColors对象中随机数产生器的随机种子
pRandomColor.Seed = 7;
bool ok = true;
pRandomColor.CreateRamp(out ok);
pEnumColors = pRandomColor.Colors;
```

```
pColor = pEnumColors.Next();
```

其他的颜色带对象本文不作介绍，读者可以参考开发帮助的相关条目。

6.2 Symbol 对象

地图需要符号和标记来表示地理对象的某些描述性信息。Symbol 就是 ArcObjects 中用于在地图上修饰要素或者图形元素的对象，它作为 Element 的 Symbol 属性而被使用。使用符号可以让一个点对象表现为一个箭头或者一个圆点，也可以让线对象的类型有丰富的变化或让一个面对象的填充颜色和样式丰富多彩。在专题图的制作中，同样的道路线要素可以使用的铁路线和公路线来区别，符号本身的差异才足以让用户了解不同要素之间某种属性上的差别和联系。

GIS 中的离散实体分为三种，即点、线和面实体。ArcObjects 使用三种符号来绘制这些要素的几何图形，即 MarkerSymbol、FillSymbol 和 LineSymbol。此外还有两种特殊的符号：一种是 TextSymbol 用于文字标注；另一种是 3D Chart 符号，用于显示饼图等三维对象。

在 ArcMap 中用户可以通过 StyleGallery 对象调用系统预存的符号对象，这些符号在程序里面都保存为 style 文件。除了调用标准符号外，用户还可以使用符号编辑器(SymbolEditor)，可视化编辑自己需要的符号对象。

所有的符号类都实现了 ISymbol 和 IMapLevel 接口，前者定义了一个符号对象的基本属性和方法；后者定义的 MapLevel 属性可以确定符号的显示层。这个概念很容易被理解，和 CAD 软件中的层概念一样，当绘制了一个面符号和一条线符号后，程序员可以使用这个属性来确定哪一个符号在上面而不被其他的符号遮盖。

- **SymbolSelector 符号选择对话框对象**

ArcMap 里面预存了很多标准符号，使用它可以帮助用户可视化取出其中的某个符号出来使用，如图 6-8 所示。

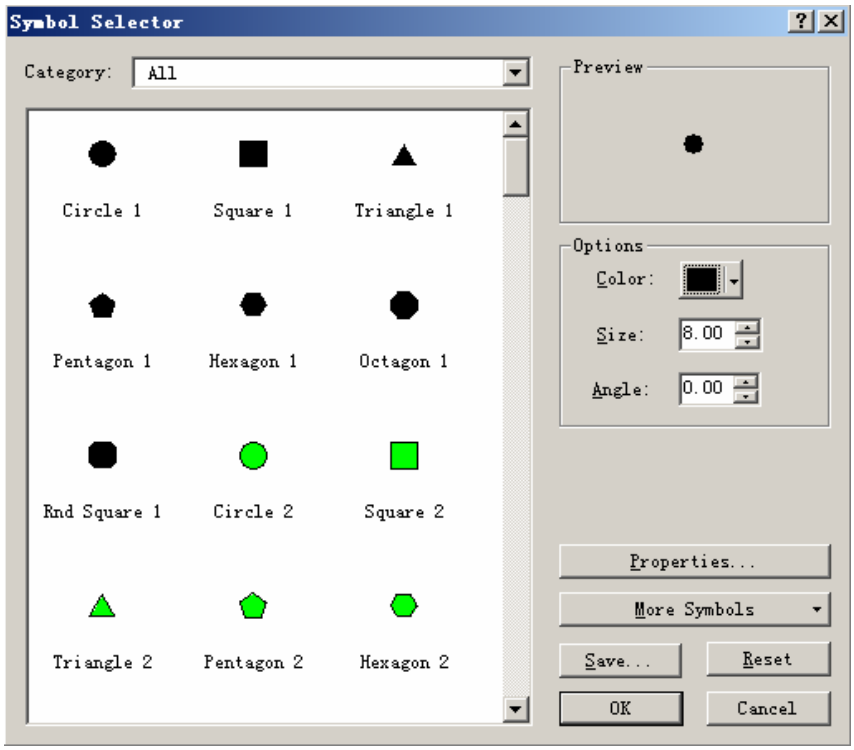


图 6-8 符号选择对话框

SymbolSelector 实现了 ISymbolSelector 接口，通过它定义的方法可以对话框中取出一个 Symbol 对象。下面是使用 ArcObjects 来编写的一个例子：

```
private ISymbol SymbolSelector()
{
    ISymbolSelector pSymbolSelector;
    pSymbolSelector = new SymbolSelectorClass();
    ISimpleMarkerSymbol pMarker;
    pMarker = new SimpleMarkerSymbolClass();
    if (!(pSymbolSelector.AddSymbol(pMarker as ISymbol)))
MessageBox.Show("Could not add symbol");
    else if (pSymbolSelector.SelectSymbol(0))
    {
        ISymbol pSymbol;
        pSymbol = pSymbolSelector.GetSymbolAt(0);
        return pSymbol;
    }
    return null;
}
```

其中 pSymbol 就是从 SymbolSelector 对话框中取出的 Symbol 对象。

6.2.1 MarkerSymbol 对象

MarkerSymbol 对象是用于修饰点对象的符号，它拥有五个子类，其中不同的子类可以产生不同类型的点符号，如图 6-9 所示。所有的 MarkerSymbol 类都实现了 IMarkerSymbol 接口，这个接口定义了标记符号的公共方法和属性，如角度、颜色、大小和 XY 偏移量等。



图 6-9 点状符号的类型

- **SimpleMarkerSymbol 简单类型的点状符号**

SimpleMarkerSymbol 对象有五种类型的符号，它们表现出来的形状分别是矩形、圆形、Cross、菱形和“X”形等，产生一个 SimpleMarkerSymbol 的方法很简单。下面是一个 MapControl 控件 MouseDown 事件中的例子，在控件上点击左键时，在地图上将会出现一个个点，而点的 Symbol 是程序员自己新建的简单点符号。

```
private void axMapControl1_OnMouseDown(object sender, ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            //产生一个简单符号
            ISimpleMarkerSymbol pMarkerSymbol;
            pMarkerSymbol = new SimpleMarkerSymbolClass();
            //Symbol的样式，为圆形
            pMarkerSymbol.Style = esriSimpleMarkerStyle.esriSMSCircle;
            //Symbol的颜色
            pMarkerSymbol.Color = getRGB(60,100,50);
            //Symbol的旋转角度
            pMarkerSymbol.Angle = 60;
            //Symbol的大小
            pMarkerSymbol.Size = 6;
```

```
            //Symbol是否有外轮廓线
            pMarkerSymbol.Outline = true;
            //Symbol外轮廓线的大小为2
            pMarkerSymbol.OutlineSize = 2;
            pMarkerSymbol.OutlineColor = getRGB(166, 122, 166);
            IPoint pPoint;
            pPoint = new PointClass();
            pPoint.PutCoords(e.mapX, e.mapY);
            object oMarkerSymbol = pMarkerSymbol;
            axMapControl1.DrawShape(pPoint, ref oMarkerSymbol);

        }
```

- **ArrowMarkerSymbol 箭头形式的符号**

ArrowMarkerSymbol 类的主要接口是 IArrowMarkerSymbol，它继承自 IMarkerSymbol 接口。IArrowMarkerSymbol 的 Length 属性指箭头的顶点到底边的距离；Width 指的是箭头底边的宽度；而 Style 是箭头符号的样式，它也只有一个样式 esriAMSPlain。

- **CharacterMarkerSymbol 字符形式的符号**

CharacterMarkerSymbol 类实现了 ICharacterMarkerSymbol 接口，这个接口也继承自 IMarkerSymbol。它可以将一个点要素显示为字符状，而字符的字体来自系统中已经安装的字符集，字符本身来通过 CharacterIndex 属性来确定，它其实就是系统字符的 ASCII 码，如 a 的 ASCII 码为 97。

下面是一个产生 CharacterMarkerSymbol 的例子，它需要在一个 MapControl 控件的 MouseDown 事件中完成：

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            //确定字体类型
            stdole.StdFont pFont ;
            pFont = new stdole.StdFontClass();
            pFont.Name = "arial";
            pFont.Size = 37;
            pFont.Italic = true;
            //新建一个IcharacterMarkerSymbol对象
            ICharacterMarkerSymbol pCharMarkerSymbol;
```

```
            pCharMarkerSymbol = new CharacterMarkerSymbolClass();
            //设置字符符号的字体
            pCharMarkerSymbol.Font = pFont as stdole.IFontDisp;
            //设置它的字符，114为'r'
            pCharMarkerSymbol.CharacterIndex = 114;
            pCharMarkerSymbol.Color = getRGB(233, 100, 233);
            pCharMarkerSymbol.Size = 25;
            IPoint pPoint;
            pPoint = new PointClass();
            pPoint.PutCoords(e.mapX, e.mapY);
            object oMarkerSymbol = pCharMarkerSymbol;
            axMapControl1.DrawShape(pPoint, ref oMarkerSymbol);

        }
```

- **PictureMarkerSymbol 以图片为背景的符号**

PictureMarkerSymbol 对象在符号的显示上显得“高级”一点，它可以把一个点对象的外形表示为一张位图图片。PictureMarkerSymbol 类实现了 IPictureMarkerSymbol 接口，这个接口定义了引用位图的方法。

PictureMarkerSymbol 对象将千张图片设置为点符号可以有两个办法，一是使用 IPictureMarkerSymbol::CreateMarkerSymbolFromFile 方法从磁盘上获取一张位图，另外一种就是设置 PictureMarkerSymbol 对象的 Picture 属性，如：PictureMarkerSymbol.picture=image1.picture。

关于 PictureMarkerSymbol 对象，读者可以参看开发帮助的相关条目。

- **MultiLayerMarkerSymbol 使用多个符号叠加生成新的符号**

如果单一的点状或者箭头符号、字符符号仍然不能满足用户的需要，还可以考虑将多个符号进行叠加的方式，生成全新的组合符号，而这种组合的个数是没有限制的。

下面是一个 Marker 符号的例子：

```
        private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
        {
            //产生一个简单符号
            ISimpleMarkerSymbol pMarkerSymbol;
            pMarkerSymbol = new SimpleMarkerSymbolClass();
            pMarkerSymbol.Style = esriSimpleMarkerStyle.esriSMSCross;
```

```
            pMarkerSymbol.Color = HSVColor(60, 100, 50);
            pMarkerSymbol.Angle = 60;
            //产生一个箭头符号
            IArrowMarkerSymbol pArrowMarkerSyl;
            pArrowMarkerSyl = new ArrowMarkerSymbolClass();
            pArrowMarkerSyl.Length = 5;
            pArrowMarkerSyl.Width = 10;
            pArrowMarkerSyl.Color = HSVColor(0, 60, 90);
            //产生一个叠加符号并把前面产生的符号加入
            IMultiLayerMarkerSymbol pMulMarker;
            pMulMarker = new MultiLayerMarkerSymbolClass();
            pMulMarker.AddLayer(pMarkerSymbol);
            pMulMarker.AddLayer(pArrowMarkerSyl);
            IPoint pPoint;
            pPoint = new PointClass();
            pPoint.PutCoords(e.mapX, e.mapY);
            object oMarkerSymbol = pMulMarker;
            axMapControl1.DrawShape(pPoint, ref oMarkerSymbol);
        }
```

本例中的符号是由一个箭头符号和一个 SimpleMarkerSymbol 组合而成的。

最后需要提醒读者的是，使用控件的 DrawShape 方法绘制的图形是无法保存的，它们本质上只是一个缓存而已，一旦窗体被强制重绘，这些符号就将消失。

6.2.2 LineSymbol 对象

LineSymbol 对象是用于修饰线型几何对象的符号，ILineSymbol 作为每一种 LineSymbol 类都要实现的接口定义了两个公共属性，即 Color 和 Width。这两个属性为所有的线符号所需要，前者用于设置线符号的颜色，后者用于设定线符号对象的宽度。

LineSymbol 抽象类有四个子类，分别是 CartographicLineSymbol、SimpleLineSymbol、PictureLineSymbol 和 MultiLayerLineSymbol，如图 6-10 所示。

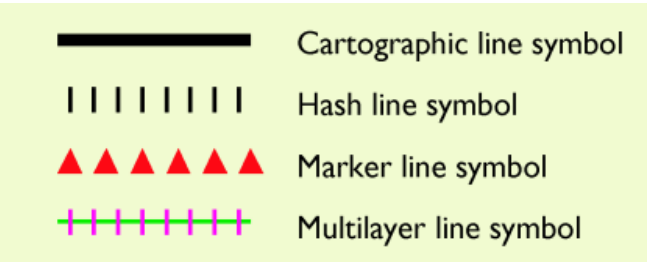


图 6-10 线状符号的类型

本节将逐个介绍这些对象的区别和使用方法：

- **SimpleLineSymbol 简单线符号**

SimpleLineSymbol 对象的主要属性是 Style，它可以确定线的形式，SimpleLineSymbol 对象拥有的线类型有 7 种，具体的内容可以参看开发帮助。

下面是一个 SimpleLineSymbol 的例子：

```
    private void axMapControl1_OnMouseDown(object sender,
ESRI.ArcGIS.MapControl.IMapControlEvents2_OnMouseDownEvent e)
    {
        //新建一个ISimpleLineSymbol对象
        ISimpleLineSymbol pSimpleLineSymbol;
        pSimpleLineSymbol = new SimpleLineSymbolClass();
        //设置线符号的颜色
        pSimpleLineSymbol.Color = getRGB(100, 112, 103);
        //设置线符号的类型
        pSimpleLineSymbol.Style = esriSimpleLineStyle.esriSLSDot;
        //设置线符号的宽度
        pSimpleLineSymbol.Width = 3;
        IGeometry pGeo;
        pGeo = axMapControl1.TrackLine();
        object oLineSymbol = pSimpleLineSymbol;
        axMapControl1.DrawShape(pGeo, ref oLineSymbol);

    }
```

- **CartographicLineSymbol 制图线符号**

CartographicLineSymbol 比 SimpleLineSymbol 拥有复杂的属性和方法来修饰一条线对象。这个类主实现了两个接口：ICartographicLineSymbol 和 ILineProperties。

ICartographicLineSymbol 接口主要用于设置线符号的节点属性，Cap 属性用于设置线要素首尾端点的形状；Join 属性用于设置线要素转折处的样式，如 esriLCSRound 是设置转折处圆滑。

ILineProperties 接口主要用于设置 dash-dot 类型的线要素符号属性，包括 Template 和 LineDecoration 等。它下面有两个子类 HashLineSymbol 和 MarkerLineSymbol(使用 MarkerSymbol 对象)可以使用。

- **MultiLayerLineSymbol 多图层线符号**

MultiLayerLineSymbol 与 MultiLayerMarkerSymbol 一样，也是可以使用重叠符号的方法生成新的线符号。该对象的具体设置方法可以参看开发帮助。

- **PictureLineSymbol 图片线符号**

PictureLineSymbol 对象可以将一条线对象通过一张位图图片表现出来，当系统需要设　置非常特殊的线符号，比如火车线路符号时，可以使用这个对象来完成。

下面是一个 LineSymbol 的例子，它使用了本节讨论的多个线符号对象的属性和方法。

```
//创建一个简单的线符号
ISimpleLineSymbol pLineSyl;
pLineSyl = new SimpleLineSymbolClass();
pLineSyl.Color = HSVColor(30, 60, 90);
pLineSyl.Width = 2;
pLineSyl.Style = esriSimpleLineStyle.esriSLSDashDot;
//产生一个制图线符号
ICartographicLineSymbol pCartoLineSyl;
pCartoLineSyl = new CartographicLineSymbolClass();
pCartoLineSyl.Color = HSVColor(0, 110, 210);
pCartoLineSyl.Cap = esriLineCapStyle.esriLCSButt;
pCartoLineSyl.Join = esriLineJoinStyle.esriLJSRound;
//QI，注意下面的代码，用于修饰线的某些属性
ILineProperties pLinePro ;
pLinePro = pCartoLineSyl as ILineProperties;
ISimpleLineDecorationElement pSimpleLineDecoEle;
pSimpleLineDecoEle = new SimpleLineDecorationElementClass();
pSimpleLineDecoEle.FlipAll = true;
pSimpleLineDecoEle.FlipFirst = true;
ILineDecoration pLineDeco;
pLineDeco = new LineDecorationClass();
```

```
pLineDeco.AddElement(pSimpleLineDecoEle);
//将线的修饰属性给PlinePro
pLinePro.LineDecoration = pLineDeco;
IGeometry pGeo;
pGeo = axMapControl1.TrackLine();
object oLineSymbol = pCartoLineSyl;
axMapControl1.DrawShape(pGeo, ref oLineSymbol);
```

6.2.3 FillSymbol 对象

FillSymbol 是用来修饰如多边形等具有面积的几何形体的符号对象，它实现了 IFillSymbol，这个接口定义了两个属性 Color 和 OutLine，以满足所有类型的 FillSymbol 对象的公共属性设置。

IFillSymbol::Color 可以设置填充符号的基本颜色，当然如果不设置这个属性，填充对象也是有默认颜色的，如 GradientFillSymbol 是蓝色，LineFillSymbol 是中度灰色，其他的填充对象都是黑色。

IFillSymbol::OutLine 属性可以设置填充符号的外边框，这个外边框是一个线对象，因此使用 ILineSymbol 对象修饰，在默认情况下它是一个 Solid 类型的简单线符号。

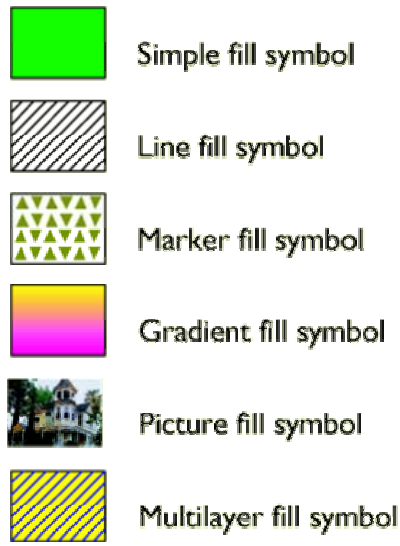


图 6-11 填充（面状）符号的类型

图 6-11 是各种 FillSymbol 符号对象的例子。

- **SimpleFillSymbol 简单填充符号**

SimpleFillSymbol 是最简单的填充符号，它继承自 FillSymbol 对象。SimpleFillSymbol 类实现的接口是 ISimpleFillSymbol，使用它定义的属性和方法可以设置一个简单填充符号不同的 Style 属性，可以选择不同的简单填充类型。SimpleFillSymbol 的类型有 8 种之多，详细的可以查看 tagesriSimpleFillStyle 的各个值。

- **LineFillSymbol 线填充符号**

如上图所示，LineFillSymbol 中的填充符号是重复的线条，它实现了 ILineFillSymbol 接口用于定义它的角度、偏移量和线之间的间隔距离；ILineFillSymbol::LineSymbol 属性用于设置线填充符号的线的样式，线条需要使用一个 ILineSymbol 对象修饰；ILineFillSymbol::Angle 属性是设置线与水平线的夹角，默认情况下这个值是 0，如果设置一个具体值，可以让这些线倾斜，这个实例可以参看上图的情况。

通常情况下，第一条线总是从地图坐标系统中的(0，0)点处通过，如果设置了 ILineFillSymbol::Offset 属性，这个线就将有一个偏移；ILineFillSymbol::Separation 属性决定了单位距　离内线出现的数量，如果这个值小于 LineSymbol 对象的宽度，那么它们将会覆盖。

- **MarkerFillSymbol 点填充符号**

MarkerFillSymbol 是使用一个 Marker 符号作为背景填充符号。这个类实现了两个接口，一个是 IMarkerFillSymbol，它用于设置填充对象的 Marker 对象的属性，如 IMarkerFillSymbol::GridAngle 属性用于设置这些 Marker 点的角度，IMarkerFillSymbol::MarkerSymbol 属性用于设置这些 Marker 的类型，它使用一个 IMarkerSymbol 对象。

另一个接口是 IFillProperties 接口，它可以设置 Marker 在填充区域内的分布情况。在默　认情况下，填充区的圆点处应该是一个 Marker 的圆心，不会发生偏移，但用户可以使用 XOffset 和 YOffset 属性来设置这个偏移量；IFillProperties::XSeparation 和 IFillProperties::YSeparation 可以设置 Marker 之间的水平和垂直距离。这四个属性都是使用像素(points)作为单位，默认情况下是 12 个像素。

如果填充对象的 Style 属性被设置为 esriMFSRandom，即这些点符号随机出现，那么设置上面的四个属性也是有意义的，尽管此时 Marker 会随机分布，但设置的 X 和 Y 属性是随机距离的平均值。

- **GradientFillSymbol 渐变颜色填充符号**

GradientFillSymbol 使用渐变颜色带进行填充，它需要使用到颜色带对象。这些颜色带可以是从一侧到另一侧，也可以是从中间到四方。GradientFillSymbol 类实现了 IGradientFillSymbol 接口，其中的 IGradientFillSymbol::ColorRamp 属性用于设置这个渐变填充符号的颜色带对象，渐变的效果最好是使用 AlgorithmicColorRamp 颜色带对象。在设置颜色条的时候，用户条并不需要使用 size 属性或者 CreateRamp 的方法来产生，用户是使用了 IntervalCount 属性来设置用户所要使用的颜色梯度。

IGradientFillSymbol::Style 属性用于设计渐变填充的样式，这些样式有四种，

即 esriGFSLinear、esriGFSRectangular、esriGFSCircular 和 esriGFSBuffered，它们决定了这些渐变的填充是使用线、矩形、圆形还是缓冲的方法。在上面的图片中就是使用 esriGFSRectangular。

- **PictureFillSymbol 图片填充符号**

PictureFillSymbol 对象使用图片来进行填充，它和 MarkerFillSymbol 非常相似，唯一不同的就是一个使用定制的点，一个使用来源更加丰富的图片。

PictureFillSymbol 类实现了 IPointFillSymbol 接口，如 IPictureFillSymbol::CreateFillSymbolFromFile 就专门用于获取系统外的图片，这个方法需要输入的参数是图片的类型和路径。 但是 PictureFillSymbol 对象能够使用的图片类型很少，只有 EMF 和 BMP 两种图形。

IPictureFillSymbol 接口还定义了其他用于设置填充符号的属性，如 XScale、YScale 属性用于设置填充对象内部图片之间在 x 方向和 Y 方向上的距离。当然，这些只有在填充符号的宽度和高度大于使用图片的高度和宽度时才可以看到效果。

- **MultilayerFillSymbol**

如果程序员需要的话，也可以使用多个填充符号进行叠加以产生新的填充符号对象。这种叠加是一层符号放在另一层符号的上面，因而可能产生覆盖的情况。

MultilayerFillSymbol 对象实现的是 IMultilayerFillSymbol 接口，这个接口定义的方法为了添加、移动、删除和清空构成这个多层填充符号使用的填充符号。使用 IMultilayerFillSymbol::MoveLayer 方法可以移动这些符号层，这样用户可以显示不同的效果。

- **DotDensityFillSymbol 点密度填充符号**

DotDensityFillSymbol 是一种基于数据的填充符号，它一般和 DotDensityRenderer 着色对象一起使用。DotDensityFillSymbol 是使用由 MarkerSymbol 组成的随机位置点来显示数据的属性，而单位面积内点的多少则由 DotDensityRenderer 对象计算出来。

DotDensityFillSymbol 类实现的是 IDotDensityFillSymbol 接口，通过这个接口可以设置填充符号的外观，如 Marker 的数目，尺寸和填充符号的背景颜色等属性。

IDotDensityFillSymbol::BackgroundColor 可以用于设置填充区域的背景颜色，当然它不会覆盖用于填充的随机点。如果新建一个 MultilayerFillSymbol 对象，其中最上一层是 DotDensityFillSymbol，如果要下面的图层可以看见，用户需要设置这个颜色为 NullColor 才行。Color 属性用于设置点的颜色，而 Outline 属性则改变这些点的外框情况。

除了 IDotDensityFillSymbol 外，这个类也实现了 ISymbolArray 接口，DotDensityFillSymbol 可以使用多个点对象来作为填充符号，如果需要使用两种以上的 Marker 对象，那么它们是居于数据中两种不同属性的，此时需要产生一个符号数组来进行设置。用户可以使用类似下面的代码：

```
ISimpleMarkerSymbol pMarker ;
ISymbolArray pSymArray;
pSymArray = new DotDensityFillSymbolClass();
pMarker = new SimpleMarkerSymbolClass();
pMarker.Style = esriSimpleMarkerStyle.esriSMSDiamond;
pSymArray.AddSymbol(pMarker as ISymbol);
pMarker = new SimpleMarkerSymbolClass();
pMarker.Style = esriSimpleMarkerStyle.esriSMSCross;

pSymArray.AddSymbol(pMarker as ISymbol);
```

6.2.4 TextSymbol 对象

TextSymbol 对象是用于修饰文字元素的，文字元素在要素标注等方面很有用处。 TextSymbol 类实现了多个接口，它们定义的属性和方法可以让程序员完成产生普通文本的 操作。此外，它还有两个属性通过单个的 CoClass 来实现，一个是 Textpath 属性，可以用于设置文字的路径；另一个是 TextBackground，用于设置文字的背景。

TextSymbol 符号最重要的设置对象是它的字符，它实现了三个主要的接口来设置字符。

ITextSymbol 接口是定义文本字符样式的主要接口，它定义的 ITextSymbol::Font 属性是产生一个 TextSymbol 符号的关键。至于如何设置文字的字体，程序员可以使用 IFontDisp 接口来设置字体的大小和是否是粗体、倾斜等属性，在 C#.NET 中，系统使用下面的代码来设置一个字体对象：

```
stdole.IFontDisp pFont;
pFont = new stdole.StdFontClass() as stdole.IFontDisp;
pFont.Name = "ESRI Cartography";
//是否为粗体
pFont.Bold = true;
//是否为倾斜
pFont.Italic = true;
//是否有下画线
pFont.Strikethrough = true;
```

在 ArcObjects 中还可以将.NET 的系统字体 System.Drawing.Font 直接转换成 stdole.IFontDisp 类型，它需要使用到 GetIFontDispFromFont 函数，如：

```
System.Drawing.Font dotNetFont ;
dotNetFont = new System.Drawing.Font("Castellar", 25.0F);
ITextSymbol textSym;
textSym = new TextSymbolClass();
textSym.Font =
ESRI.ArcGIS.Utility.COMSupport.OLE.GetIFontDispFromFont(dotNetFont) as
stdole .IFontDisp;
```

但 ArcObjects 中没有将 IFontDisp 直接转成 System.Drawing.Font 类型的函数，用户可以新建一个 System.Drawing.Font 对象，然后读取 IFontDisp 对象里面的属性，赋给 System.Drawing.Font 对象的属性。

使用 ITextSymbol 接口还可以定义 TextSymbol 对象的颜色、角度、水平排列方式、垂直排列方式和文本等内容。读者可以查阅开发帮助。

TextSymbol 类实现 ISimpleTextSymbol 接口来设置它的一些简单属性，如 XOffset 和 YOffset 可以用于设置字符的偏移量，它还定义了一个重要的属性 TextPath，这个属性要传入一个 ITextPath 对象。

字符的设置是非常复杂的，如字符可能有背景对象、阴影等属性，这个时候可以通过 IFormattedTextSymbol 接口定义的方法来设置它们。如 IFormattedTextSymbol::ShallowColor 设置阴影颜色，IFormattedTextSymbol::ShapeXOffset 和 ShapeYOffset 用于设置字体在 X 方向和 Y 方向上的偏移值。IFormattedTextSymbol::CharacterSpcing 和 IFormattedTextSymbol::CharterWidth 用于设置文本符号中单个字符之间的空隙和字符的宽度等属性。

TextPath 可以让确定每一个字符的排列位置，它是一个抽象类，其三个具体 的子类是 BezierTextPath、SimpleTextPath 和 OverposterTextPath。ITextPath 接口提供了方法来计算每一个字符沿文字路径的位置。在设置一个 TextSymbol 的文字路径时，必须通过 ISimpleTextSymbol::TextPath 来获得文字路径的引用；使用 Geometry 属性还可以设置字符路径的几何形状，而使用 XOffset 和 YOffset 属性可以让文字在设置的路径上发生偏移。

SimpleTextPath 对象可以让文字路径是任何一种曲线(1Curve)的轨迹，BezierTextPath 对象可以让文字的路径严格按照贝塞尔曲线进行设置。

背景(Background)可以被认为是一种用于强调文字的方式，它也是文字符号的一部分。下图是用户常常用到的三种背景类型：MarkerTextBackground、

BalloonCallout 和 LineCallout。

MarkerTextBackground 是采用一个定制的 Marker 符号作为 TextSymbol 的背景符号，在很多场合用户都可以使用到这个对象，如一幅城市地图中通过设置不同类型单位的字符作为背景来显示这些单位的分布。MarkerTextBackground 类实现了 IMarkerTextBackground 接口用于定制这个背景的属性，如 IMarkerTextBackgrcund::ScalToFit 属性确定背景对象的缩　放比例是否和文字的大小保持协调，而 IMarkerTextBackground::Symbol 则用于设置作为背　景的符号。

BalloonCallout 绘制一个矩形或者是以一个气球形状的背景来修饰文字，而 LineCallout　则是 BalloonCallout 的简化版本，它仅仅使用一个箭头线来修饰文字。两个对象都支持 ICallout 接口，它可以确定两个 Callout 对象的一般属性，如 AnchorPoint(箭头点)属性就是箭头端点的坐标。

IBalloonCallout 是 BalloonCallout 类的默认接口，用户通过 IBalloonCallout::Style 确定 Callout 的几何外形，如矩形，圆角矩形或者卵形。使用这个接口定义的 Symbol 方法还可以用于设置这个 Callout 内部的填充符号。

LineCallout 默认实现的是 ILineCallout 接口，它定义的方法用于确定箭头线的各种属性，用户也是通过 Style 属性用来设置不同类型的线的。

下面选用一个例子来表现如何在 MapControl 控件上绘制文字，这个过程是在控件的 MouseDown 事件中完成的。

```
IElement pElement;
ITextElement pTextEle;
pTextEle = new TextElementClass();
pElement = pTextEle as IElement;
pTextEle.Text = "ESRI&SCUT";
//注意文字元素的几何形体对象是一个点
IPoint pPoint;
pPoint = new PointClass();
pPoint.PutCoords(e.mapX, e.mapY);
pElement.Geometry = pPoint;
//产生一个新的文字样式符号
IFormattedTextSymbol pTextSymbol;
pTextSymbol = new TextSymbolClass();
//产生文字符号的背景属性，是一个callout模式的
ICallout pCallout;
pCallout = new BalloonCalloutClass();
pTextSymbol.Background = pCallout as ITextBackground ;
```

```
pPoint.PutCoords(e.mapX - 30, e.mapY + 10);
//设置箭头点的位置
pCallout.AnchorPoint = pPoint;
//它作为textelement的symbol属性而使用
pTextEle.Symbol = pTextSymbol;
IGraphicsContainer pGrahpicsContainer ;
pGrahpicsContainer = axMapControl1.Map as IGraphicsContainer;
pGrahpicsContainer.AddElement(pTextEle as IElement , 0);
axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
```

6.2.5 3DChartSymbol 对象

3DChartSymbol 是一个抽象类，它拥有三个子类。它本质上是一种 Marker 符号，一般用于 ChartRenderer 对象的着色，而且这个着色方法常常是基于多个属性的。3DChartSymbol　主要是为了着色对象(renderer)中的柱状、饼状着色服务的。这个符号群有三个子类：BarChart、PieChart 和 StaekedChart。

3DChartSymbol 实现的接口有多个，如 IChartSymbol 接口主要用于计算一个 ChartSymbol 对象中的柱状(Bar)和饼状部分的尺寸，IChartSymbol::Maximum 值是新建一个 3DChartSymbol 对象后必须设置的属性。由于着色一般都是在一个要素图层上进行的，因而可以从它的要素类中获得一系列的数值统计值，如果有三个数值字段参与着色，系统必须对这三个字段中的最大值进行比较，找出最大的那一个值出来赋给 Maximum。

IChartSymbol::Value 属性包含这一个值的数组，这个数组用来设置每个 Bar 的相对高度或者每个 Pie 的宽度。但在使用 ChartRenderer 进行着色中，并不用设置这个属性。

使用这种 3D 符号着色时，符号不止一种，系统使用 ISymbolArray 接口来管理一个着色对象中的多个参与着色的符号对象。ISymbolArray::Symbol 属性可以让程序员通过索引值获得每个 symbol 对象，而 SymbolCount 则会返回参与着色的符号的数目。这个接口还提供了添加、删除和清除符号的方法，如 AddSymbol，DeleteSymbol 和 ClearSymbols 等。

BarChartSymbol 是最常用的三维着色符号，它使用不同类型的柱子来代表一个要素类中不同的属性，而柱子的高度取决于属性值的大小。这个对象支持的是 IBarChartSymbol 接口。

IBarChartSymbol::VerticalBars 属性用于确定使用的柱子(Bar)是水平排列还是垂直排列；柱的宽度和柱之间的空隙可以通过 Width 和 Spacing 属性来调节，当然，

它们也可以使用 IMarkerSymbol::Size 方法进行设置。

ArcObjects 可以使用 IBarChartSymbol::Axes 属性来设置每根 Bar 的轴线，轴线是一个 ILineSymbol 对象，当然如果要这个轴线能够显示，还必须设置 ShowAxes 为 True 才行。

PieChartSymbol 符号进行着色的方法是使用一个饼图来显示不同要素类中的不同属性，不同的属性按照它们的数值大小占有一个饼图中的不同比例的扇形区域。它默认实现的是 IPieChartSymbol 接口，这个接口定义的属性用于设置 Pie 的外观，如 IPieChartSymbol::ClockWise 属性用于确定饼图中颜色的方向。如果 ClockWise 为 True 的时候，饼图中的颜色块呈顺时针方向分布。

当 IPieChartSymbol::UseOutline 为 True 的时候，饼图的外框可以设置外框线；外框线使用 IPieChartSymbol::Outline 设置，它是一个 ILineSymbol 对象。

StackedChartSymbol 也是 ChartRenderer 对象进行着色时最常用的符号，它使用的柱 (StackedBar)是堆垒而成的。这个对象支持的接口是 IStackedChartSymbol，用于设置 StackedChartSymbol 的外观。

IStackedChartSymbol::Width 属性用于设置柱的宽度，而 Outline 和 UseOutline 用于设置符号的外框线。

当 Fixed 属性为 False 时，ChartRenderer 对象的每一个 stackedbar 的尺寸会依据每个要素的属性来计算。如果它为 True 的时候，则 StackedBar 的长度是一样的。

这个主题讨论了如何使用 3D 符号，它们的具体使用读者可以查看本书的制作 ChartRenderer 专题图一节的内容。

6.3 制作专题图

地图着色是一个非常古老的问题。从地图出现的时代开始，地图学家们就研究如何使用不同的符号来修饰地图上的不同要素，使得地图能够清晰、美观和直接地表达用户所需要的不同类型的信息。

ArcMap 提供了多种方法来制作不同类型的专题图，这些专题图是可以依据要素的一个或多个属性的不同而设置不同类型的符号，从而达到区分不同类型要素的目的。在实际项目中用户也经常使用要素着色的功能，如制作价格分布图、土地利用图等。如图 6-12 所示，它是 ArcMap 中用于着色的对话框，又称为符号化 (Symbology)。

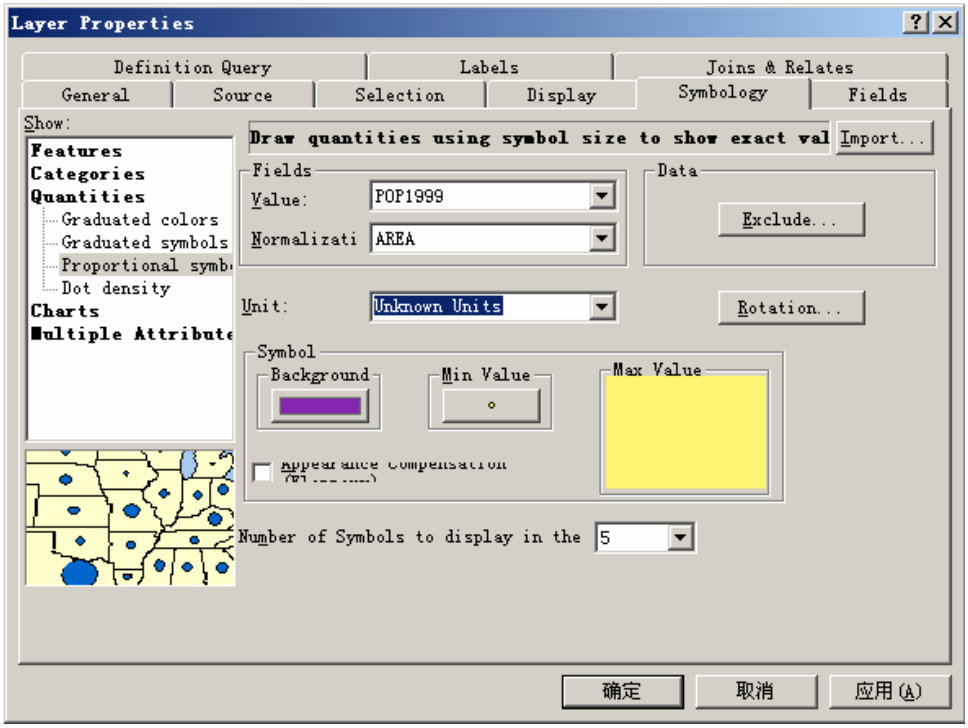


图 6-12　Symbology 对话框

ArcObjects 给用户提供了多个着色对象用于专题图的产生，这个过程中既可以使用标准的着色方案，也可以定制自己的着色方案。ArcObjects 中提供的标准着色方法有 8 种。

读者需要注意的是，地图着色是对一个图层的所有要素进行的，目前没有直接的办法对其中单独挑选出来的要素进行着色。为了解决一个要素图层中部分要素单独着色的问题，程序员可以使用别的方法：一是在这些要素上绘制图形元素，这些图形元素的几何形状就是要着色要素的 Geometry，它们的符号当然是不一样的；另一种方法是先把需要着色的要素放入图层的选择集，使用 IFeatureLayerDefinition::CreateSelectionLayer 方法将它们新建一个新图层，然后对这个新图层进行着色。

FeatureRenderer 是一个抽象类，其子类负责进行不同类型的着色运算。它们都实现了 IFeatureRenderer 接口，这个接口定义了进行地图着色运算的公共属性和方法。

使用不同的着色对象时，用户需要确定是哪个图层需要被着色，而着色对象只是要素图层的一个属性而已，程序员可以通过 IGeoFeatureLayer::Renderer 属性获得一个图层的着色对象。

在着色对象的学习中，本节还要使用 ITable 接口获得图层，讨论采取不同的方法从表中得到一系列希望获得的统计值以帮助着色运算。

下面介绍的几种主要的着色方法，在一般的 GIS 开发中都基本可以满足要求，读者可以在这些基础着色例子的基础上自己加以扩展和发挥。

6.3.1 SimpleRenderer 着色法

使用单一符号来进行着色分类的方式最为简单，它不涉及对要素图层的数据处理。在默认情况下系统都是使用这种着色方法来修饰要素的，当在 MapControl 中打开一幅地图的时候，用户会发现一个要素图层内所有的要素都是一种符号，这是系统自动选择随机 Symbol 的结果。

使用这种着色方法用户需要产生一个 SimpleRenderer 对象，然后将它的 Symbol 进行设置后赋给 IGeoFeaturelayer::Renderer 即可。需要注意的是，它的类还实现了 ITransparencyRenderer 接口，后者可以依据要素的某个数值字段的值来设置要素的显示透明度。

下面是使用 SimpleRenderer 进行简单着色的一个例子，结果如图 6-13 所示，这个例子以命令方式实现。

SimpleRender.cs

```
using System;
using System.Drawing;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.esriSystem ;
using ESRI.ArcGIS.Framework;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.CATIDs;

namespace Symbology
{
    public class SimpleRender:BaseCommand
    {
```

```
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap;
        IActiveView pActiveView;
        public SimpleRender()
        {
            base.m_caption = "SimpleRender";
            base.m_message = "SimpleRender";
            base.m_toolTip = "SimpleRender";
            base.m_category = "Symbology";
            base.m_enabled = true;
            //      base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }


        public override void OnClick()
        {
            IGeoFeatureLayer pGeoFeatureL;
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pMap.ReferenceScale = 0;
            pGeoFeatureL = pMap.get_Layer(1) as IGeoFeatureLayer;
            //新建一个填充符号，注意不同类型的图层的着色symbol不同
            IFillSymbol pSimpleFillS;
            ILineSymbol pLineS
            pSimpleFillS = new SimpleFillSymbolClass();
            pSimpleFillS.Color = getRGB(120, 110, 200);
            pLineSymbol = new SimpleLineSymbolClass();
            pLineSymbol.Color = getRGB(255, 0, 0);
            pLineSymbol.Width = 1;
            pSimpleFillS.Outline = pLineSymbol;
            ISimpleRenderer pSimpleRenderer;
```

图 4-18　SimpleRenderer

```
            pSimpleRenderer = new SimpleRendererClass();
            pSimpleRenderer.Symbol = (ISymbol) pSimpleFillS;
            pSimpleRenderer.Description = "USA";
            pSimpleRenderer.Label = "SimpleRenderer";
            ITransparencyRenderer pTransRenderer;
            pTransRenderer = pSimpleRenderer as ITransparencyRenderer;
            //以popl999的值作为要素透明设置的属性
            pTransRenderer.TransparencyField = "POP1999";
            pGeoFeatureL.Renderer = pTransRenderer as IFeatureRenderer;
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);

        }
```

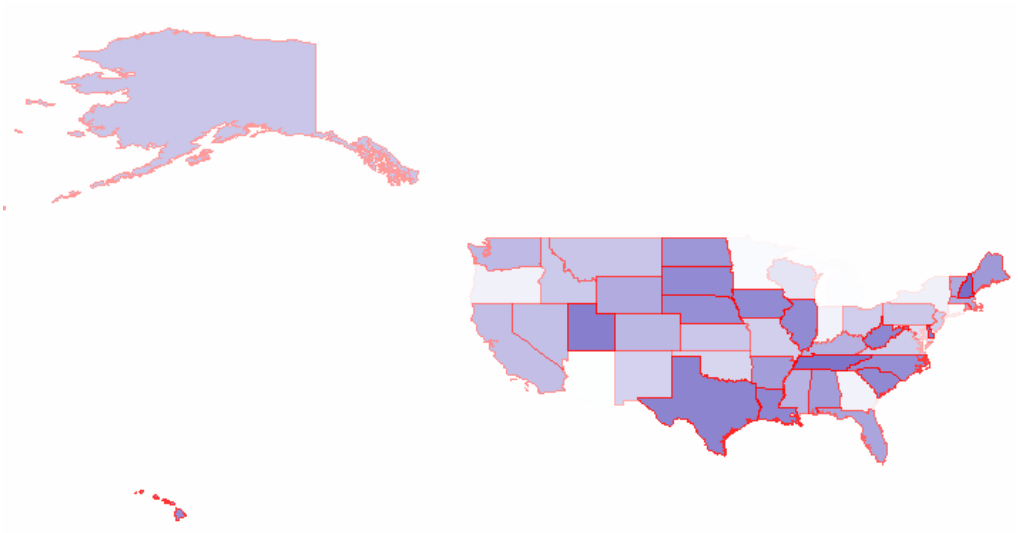


图 6-13　SimpleRenderer 着色结果

图 6-13 既是使用 SimpleRenderer 对象进行着色的结果，其中所有要素显示的颜色都是一样的。用户可以看到有的要素明显是透明的，这个透明度依据要素类的 POP1999 属性值而设定。

ISimpleRenderer 还有两个重要的属性：Description 和 Label，这两个属性是用于设置 Legend(图例)的。图例将会显示图层使用的所有着色 Symbol 的样式。

6.3.2 ClassBreakRenderer 着色法

分级着色法需要获取要素图层的要素类中某个数值字段的值，根据用户的要求将这些值分为几个级别，由于每个级别的 Symbol 不同而显示出不同的外观。如图 6-14 所示，图中的地图就是将所有的要素依据要素类的 POP1990 的值而分为五个级别进行着色区分的。

进行分类着色需要使用到 ClassBreakRenderer 对象，它的类实现了 IClassBreakRenderer 接口。下面这个例子中，为了取得数据，使用了 ClassifyGEN 对象和 Histogram 对象。它们配合 ClassBreakRenderer 着色对象使用，产生了分级着色效果。它有几个属性比较重要：

IClassBreakRenderer::Field 要素类中用于分级着色的字段；IClassBreakRenderer::BreakCount 分级的数目；IClassBreakRenderer::Symbol（i）和 Break（i），它们确定分级的符号和分级点处的临界值。

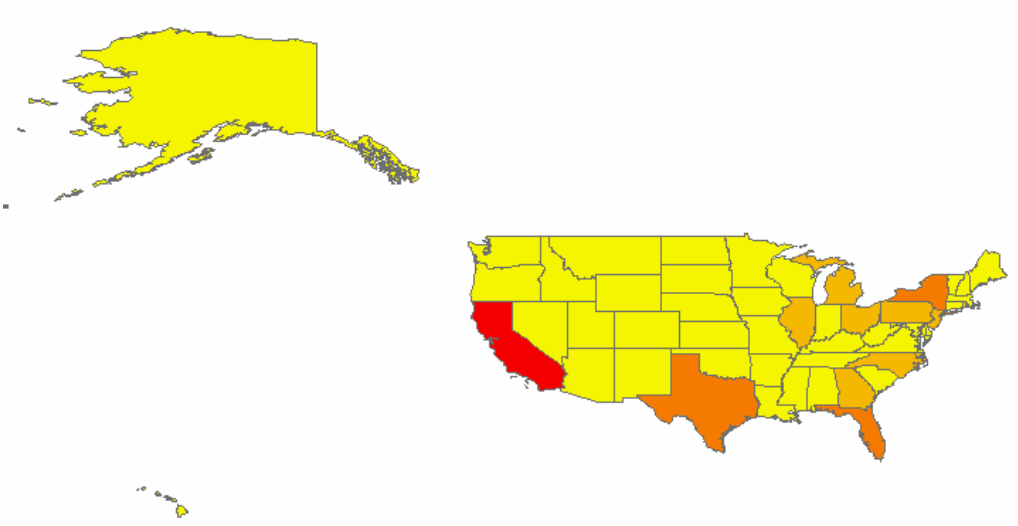


图 6-14　使用 ClassBreakRenderer 着色的地图

下面是一个使用 ArcObjects 代码进行分级着色的例子。

ClassBreaksRenderer.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.TOCControl;
using ESRI.ArcGIS.Display;
```

```
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.CATIDs;

namespace Symbology
{
    public class ClassBreaksRenderer:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap;
        IActiveView pActiveView;
        public ClassBreaksRenderer()
        {
            base.m_caption = "ClassBreaksRenderer";
            base.m_message = "ClassBreaksRenderer";
            base.m_toolTip = "ClassBreaksRenderer";
            base.m_category = "Symbology";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            IGeoFeatureLayer pGeoFeatureL;
            ITable pTable;
            IClassifyGEN pClassify;
            ITableHistogram pTableHistogram;
            IBasicHistogram pHistogram;
            object dataFrequency;
```

```
object dataValues;
double[] Classes;
int ClassesCount;
IClassBreaksRenderer pClassBreaksRenderer ;
IHsvColor pFromColor;
IHsvColor pToColor ;
IAlgorithmicColorRamp pAlgorithmicCR;
IEnumColors pEnumColors;
bool ok;
IColor pColor ;
ISimpleFillSymbol pSimpleFillS;
int lbreakIndex;
string strPopField = "POP1990";
int numDesiredClasses = 5;

 //获得要着色的图层
pActiveView = m_hookHelper.ActiveView;
pMap = m_hookHelper.FocusMap;
pMap.ReferenceScale = 0;
pGeoFeatureL = (IGeoFeatureLayer) pMap.get_Layer(1);
pTable = (ITable) pGeoFeatureL.FeatureClass;
//从pTable的“pop1990”字段中得到信息给datavalues和datafrequency两个数组
pTableHistogram = new BasicTableHistogramClass();
pHistogram = (IBasicHistogram) pTableHistogram;
pTableHistogram.Field = strPopField;
pTableHistogram.Table = pTable;
pHistogram.GetHistogram(out dataValues, out dataFrequency);
//下面是分级方法，用于根据获得的值计算得出符合要求的数据
//根据条件计算出Classes和ClassesCount，5为分级数目
pClassify = new EqualIntervalClass();
try
{
      pClassify.Classify(dataValues,dataFrequency, ref numDesiredClasses);
}
catch(Exception e)
{
      MessageBox.Show( e.Message );
}
```

```
//返回一个数组
Classes = (double[]) pClassify.ClassBreaks;
ClassesCount = Classes.GetUpperBound(0);
pClassBreaksRenderer = new ClassBreaksRendererClass();
pClassBreaksRenderer.Field = strPopField;
//设置着色对象的分级数目
pClassBreaksRenderer.BreakCount = ClassesCount;
pClassBreaksRenderer.SortClassesAscending = true;
//产生分级着色需要的颜色带对象的起止颜色对象
pFromColor = new HsvColorClass();
pFromColor.Hue = 60;// Yellow
pFromColor.Saturation = 100;
pFromColor.Value = 96;
pToColor = new HsvColorClass();
pToColor.Hue = 0 ;
pToColor.Saturation = 100;
pToColor.Value = 96;
//产生颜色带对象
pAlgorithmicCR = new AlgorithmicColorRampClass();
pAlgorithmicCR.Algorithm = esriColorRampAlgorithm.esriHSVAlgorithm;
pAlgorithmicCR.FromColor = pFromColor;
pAlgorithmicCR.ToColor = pToColor;
pAlgorithmicCR.Size = ClassesCount;
pAlgorithmicCR.CreateRamp(out ok);
//获得颜色
pEnumColors = pAlgorithmicCR.Colors;
//需要注意的是分级着色对象中的symbol和break的下标都是从0开始
for(lbreakIndex = 0 ;lbreakIndex <= ClassesCount - 1; lbreakIndex ++)
{

    pColor = pEnumColors.Next();
    pSimpleFillS = new SimpleFillSymbolClass();
    pSimpleFillS.Color = pColor;
    pSimpleFillS.Style = esriSimpleFillStyle.esriSFSSolid;
    //这里就是构造不同颜色着色的方法
    pClassBreaksRenderer.set_Symbol(lbreakIndex,(ISymbol) pSimpleFillS);
     //着色对象的断点
    pClassBreaksRenderer.set_Break(lbreakIndex,Classes[lbreakIndex + 1]) ;
```

```
                }
                pGeoFeatureL.Renderer = (IFeatureRenderer) pClassBreaksRenderer;
        pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);
            }
        }
}
```

在上例中，系统需要使用一个 ITableHistogram 对象先从一个要素类获得 POP1990 字段的所有值和值频率，即 datavalues 和 datafrequency，这两个数组是进行分级的基本数据。取得两个数组后，系统又使用 IClassifyGEN 对象对它们进行分级，得到 Classes 和 ClassesCount，前者是分级临界点值的数组，后者是分级数目。根据得到的分级数目和颜色带对象，用户可以分别设置 ClassBreakRenderer 对象的不同符号，产生不同的效果。在 C#.NET 中程序员必须使用 IClassifyGEN 这个接口定义的对象才能解决分级问题。

6.3.3 UniqueValueRenderer 着色法

UniqueValueRenderer 着色法产生的颜色数量是最多的，它依据要素图层的要素类中某个数值字段的值的不同而给每个要素一个单独的颜色。如果一个要素图层有 100 个要素，它就需要 100 种 Symbol 来对这些要素进行着色。这种着色方式可以区分存在的每一个要素，如图 6-15 所示。

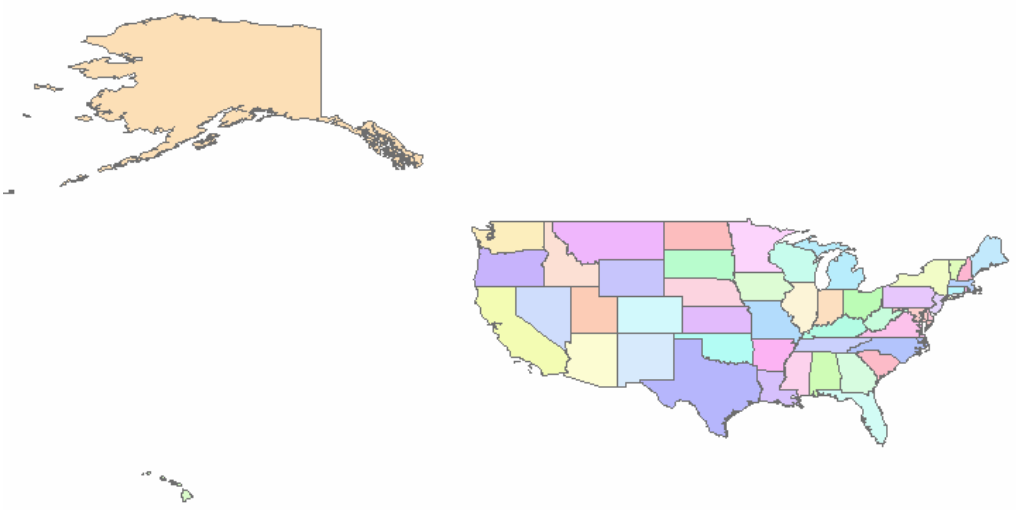


图 6-15　UniqueValue 着色法

UniqueValueRenderer 负责进行 UniqueValue 型的着色运算，它实现了

IUniqueValueRenderer 接口。IUniqueValueRenderer 定义了该着色对象可以使用的各种方法和属性。

UniqueValueRenderer 的使用其实非常简单，一般的思路如下：寻找出要素图层中要素类的所有要素，然后使用 IUniqueValueRenderer::AddValue 方法将单个要素的某个字段值和与之相匹配的着色符号加入 UniqueValueRenderer 对象，再将新建的着色对象赋给要素图层的 Renderer 属性，刷新要素图层即可。

下面的这一段代码涉及如何从一个数据表中读取出每一行数据，但它是通过一个字段，读取这个字段中所有的值出来。它实现了 UniqueValueRenderer 类型的着色运算。

UniqueValueRenderer.cs

```
using System;
using System.Drawing;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.TOCControl;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.CATIDs;

namespace Symbology
{

    public class UniqueValueRenderer:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap;
        IActiveView pActiveView;

        public UniqueValueRenderer()
        {
            base.m_caption = "UniqueValueRenderer";
```

```
            base.m_message = "UniqueValueRenderer";
            base.m_toolTip = "UniqueValueRenderer";
            base.m_category = "Symbology";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            IGeoFeatureLayer m_pGeoFeatureL;
            IUniqueValueRenderer pUniqueValueR;
            IFillSymbol pFillSymbol;
            IColor pNextUniqueColor;
            IEnumColors pEnumRamp;
            ITable pTable;
            int IfieldNumber;
            IRow pNextRow;
            IRowBuffer pNextRowBuffer;
            ICursor pCursor;
            IQueryFilter pQueryFilter;
            string codeValue;
            IRandomColorRamp pColorRamp;
            string strNameField = "State_Name" ;

            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pMap.ReferenceScale = 0;
            m_pGeoFeatureL = (IGeoFeatureLayer) pMap.get_Layer(1);
            pUniqueValueR = new UniqueValueRendererClass();
            pTable = (ITable) m_pGeoFeatureL;
            //找出STATE_NAME在字段中的编号
```

```
lfieldNumber = pTable.FindField(strNameField);
if (lfieldNumber == -1)
{
        MessageBox.Show( "Can't find field called "+ strNameField);
        return;
}
//只用一个字段进行单值着色
pUniqueValueR.FieldCount = 1;
//用于区分着色的字段
pUniqueValueR.set_Field(0,strNameField);
//产生一个随机的着色条，使用的是HSV颜色模式
pColorRamp = new RandomColorRampClass();
pColorRamp.StartHue = 0;
pColorRamp.MinValue = 99;
pColorRamp.MinSaturation = 15;
pColorRamp.EndHue = 360;
pColorRamp.MaxValue = 100;
pColorRamp.MaxSaturation = 30;
//任意产生100个颜色，如果知道要素的数目则可以产生精确的颜色数目
pColorRamp.Size = 100;
bool ok=true;
pColorRamp.CreateRamp(out ok);
pEnumRamp = pColorRamp.Colors;
pNextUniqueColor = null;
//产生查询过滤器对象
pQueryFilter = new QueryFilterClass();
pQueryFilter.AddField(strNameField);
//依据某个字段在表中找出指向所有行的游标对象
pCursor = pTable.Search(pQueryFilter, true);
pNextRow = pCursor.NextRow();
//遍历所有的要素
while (pNextRow != null)
{
        pNextRowBuffer = pNextRow;
        //找出每ROW的"STATE_NAME"的值，即不同的州名
        codeValue = (string)pNextRowBuffer.get_Value(lfieldNumber);
        //获得随机颜色带中的任意一种颜色
        pNextUniqueColor = pEnumRamp.Next();
```

```
                    if (pNextUniqueColor == null)
                    {
                        pEnumRamp.Reset();
                        pNextUniqueColor = pEnumRamp.Next();
                     }
                    pFillSymbol = new SimpleFillSymbolClass();
                    pFillSymbol.Color = pNextUniqueColor;
                   //将每次得到的要素字段值和修饰它的符号放入着色对象中
                    pUniqueValueR.AddValue(codeValue, strNameField, (ISymbol)
pFillSymbol);
                    pNextRow = pCursor.NextRow();
                }
                m_pGeoFeatureL.Renderer = (IFeatureRenderer) pUniqueValueR;
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);
            }
        }
}
```

6.3.4 ProportionalSymbolRenderer 着色法

ProportionalSymbolRenderer(依比例符号着色法)依据要素图层的要素类中某个字段的数值而生成大小不一的符号进行着色。如图 6-22 所示，图中是依据不同州的“POP1990”的值生成不同大小的圆点，从而可以一目了然地看出哪些地方的 1990 年人口多，哪些州的人口少。

ProportionalSymbolRenderer 对象可以完成这个着色运算，它实现了 IProportionalSymbolRenderer 接口，定义了使用依比例符号着色法的方法和属性，这种着色方式用户需要知道最大和最小的圆点各自代表的字段值、着色基于的字段和着色点使用的符号(MarkerSymbol 类型)以及在它的 Legend 要出现的级别数目 LegendSymbolCount。

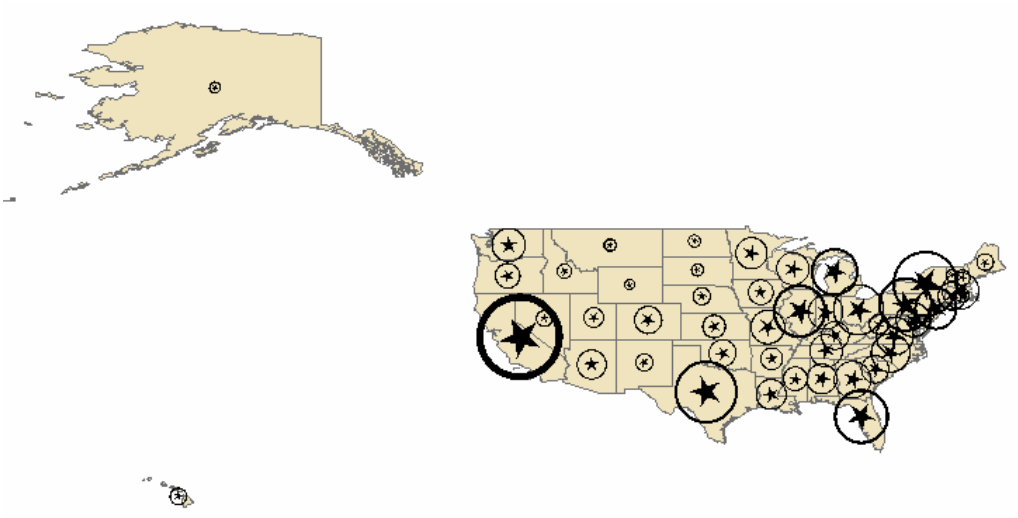


图 6-16　ProportionalSymbolRenderer 着色法

图 6-16 为依比例符号着色法的结果，下面是具体代码：

ProportionalSymbol.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Utility.BaseClasses;
using stdole;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.TOCControl;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.CATIDs;

namespace Symbology
{
    public class ProportionalSymbol:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
```

```
        IMap pMap;
        IActiveView pActiveView;

        public ProportionalSymbol()
        {
            base.m_caption = "ProportionalSymbol";
            base.m_message = "ProportionalSymbol";
            base.m_toolTip = "ProportionalSymbol";
            base.m_category = "Symbology";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            IGeoFeatureLayer pGeoFeatureLayer;
            IFeatureLayer pFeatureLayer;
            IProportionalSymbolRenderer pProportionalSymbolR;
            ITable pTable;
            IQueryFilter pQueryFilter;
            ICursor pCursor;
            IFillSymbol pFillSymbol;
            ICharacterMarkerSymbol pCharaterMarkerS;
            IDataStatistics pDataStatistics;
            IStatisticsResults pStatisticsResult;
            stdole.StdFont pFontDisp;
            IRotationRenderer pRotationRenderer;
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pMap.ReferenceScale = 0;
            pFeatureLayer = (IGeoFeatureLayer) pMap.get_Layer(1);
```

```
pGeoFeatureLayer = (IGeoFeatureLayer) pFeatureLayer;
pTable = (ITable) pGeoFeatureLayer;
pQueryFilter = new QueryFilterClass();
pQueryFilter.AddField("");
pCursor = pTable.Search(pQueryFilter, true);

//Use the statistics objects to calculate the max value and the min value
pDataStatistics = new DataStatisticsClass();
pDataStatistics.Cursor = pCursor;
//Set statistical field
pDataStatistics.Field = "POP1990";
//Get the result of statistics
pStatisticsResult = pDataStatistics.Statistics;
if (pStatisticsResult == null)
{
   MessageBox.Show("Failed to gather stats on the feature class");
       Application.Exit();
}
//Set up the background fill color
pFillSymbol = new SimpleFillSymbolClass();
pFillSymbol.Color = getRGBColor(239, 228, 190);
//Set up the min symbol to a special character marker symbol
pCharaterMarkerS = new CharacterMarkerSymbolClass();
pFontDisp = new stdole.StdFontClass();
pFontDisp.Name = "ESRI Business";
pFontDisp.Size = 20;
pCharaterMarkerS.Font = (IFontDisp) pFontDisp;
pCharaterMarkerS.CharacterIndex = 90;
pCharaterMarkerS.Color = getRGBColor(0, 0, 0);
pCharaterMarkerS.Size = 8;
// Create a new proportional symbol renderer to draw pop1990
pProportionalSymbolR = new ProportionalSymbolRendererClass();
pProportionalSymbolR.ValueUnit = esriUnits.esriUnknownUnits;
pProportionalSymbolR.Field = "POP1990";
pProportionalSymbolR.FlanneryCompensation = false;
pProportionalSymbolR.MinDataValue = pStatisticsResult.Minimum;
pProportionalSymbolR.MaxDataValue = pStatisticsResult.Maximum;
pProportionalSymbolR.BackgroundSymbol = pFillSymbol;
```

```
                pProportionalSymbolR.MinSymbol = (ISymbol) pCharaterMarkerS;
                pProportionalSymbolR.LegendSymbolCount = 5;
                pProportionalSymbolR.CreateLegendSymbols();
                //set rotation renderer by pop1990
                pRotationRenderer = (IRotationRenderer) pProportionalSymbolR;
                pRotationRenderer.RotationField = "POP1990";
                pRotationRenderer.RotationType =
esriSymbolRotationType.esriRotateSymbolGeographic;
                //Set the states layer renderer to the proportional symbol renderer and refresh
the display
                pGeoFeatureLayer.Renderer = (IFeatureRenderer) pProportionalSymbolR;
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);
            }
        }
}
```

6.3.5 ChartRenderer 着色

ChartRenderer 对象使用饼图(PieChart)或柱状图(BarChart)来比较一个要素的多个属性，如要需要对于一个地区的 1990 年人口和 1999 年人口进行比较，并且这些比较在多个要素中都进行的话，这种着色方式就是最好的选择。

ChartRenderer 使用 ChartRenderer 对象来完成着色任务。这种着色方法有两种表现形式，一个是水平排列，另外一种是累积(Stacked)排列，其中的每一根柱子都代表了要素的一个属性。三种 ChartRenderer 类都实现了 IChartRenderer 接口，这个接口定义了一系列方法，如 ChartSymbol 方法用于设置着色对象的着色符号、Label 属性用于设置 Legend 的标签等。

由于这种作色方法是比较一个要素中的不同属性，因而它需要获得被着色图层中的单个要素的一个或者多个字段，这个任务需要使用 RendererField 对象来完成，ChartRenderer 类实现了 IRendererFields 接口，它通过使用 AddField 方法来添加进行比较的字段。

本节主题将讨论如何使用三种不同的 3D 着色方法。

6.3.5.1ChartRenderer 着色

下面是对着色图层的 1990 和 1999 年人口的比较的着色代码，它使用了要素图层的 POP1990 和 POP1999 两个字段中的值。这个着色运算主要分为两大部分，

一是对两个字段中的值进行比较，以获得最大值赋给 ChartSymbol 的 MaxValue；二是设置不同的着色符号 (Symbol)。

在获取最大值的时候本例使用 DataStatistics 对象来寻找最大值，它是一个专门的数值统计对象。下面是产生一个 ChartRenderer 对象的代码段。

BarChartRenderer.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Utility.BaseClasses;
using stdole;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.TOCControl;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.CATIDs;

namespace Symbology
{
    public class BarChartRenderer:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap ;
        IActiveView pActiveView;

        public BarChartRenderer()
        {
            base.m_caption = "BarChartRenderer";
            base.m_message = "BarChartRenderer";
            base.m_toolTip = "BarChartRenderer";
            base.m_category = "Symbology";
            base.m_enabled = true;
            //base.m_bitmap = new
```

```
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            IGeoFeatureLayer pGeoFeatureL;
            IFeatureLayer pFeatureLayer;
            ITable pTable;
            ICursor pCursor;
            IQueryFilter pQueryFilter;
            IRowBuffer pRowBuffer;
            int numFields=2;
            int[] fieldIndecies = new int[numFields];
            int IfieldIndex;
            double dmaxValue;
            bool firstValue;
            double dfieldValue;
            IChartRenderer pChartRenderer;
            IRendererFields pRendererFields;
            IFillSymbol pFillSymbol;
            IMarkerSymbol pMarkerSymbol;
            ISymbolArray pSymbolArray;
            IChartSymbol pChartSymbol;
            string strPopField1 = "POP1990";
            string strPopField2 = "POP1999";

            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pMap.ReferenceScale = pMap.MapScale ;
            pFeatureLayer = (IGeoFeatureLayer) pMap.get_Layer(1);
            pGeoFeatureL = (IGeoFeatureLayer) pFeatureLayer;
            pTable = (ITable) pGeoFeatureL;
```

```
                pGeoFeatureL.ScaleSymbols = true;
                pChartRenderer = new ChartRendererClass();
                //Set up the fields to draw charts of
                pRendererFields = (IRendererFields) pChartRenderer;
                pRendererFields.AddField(strPopField1,strPopField1);
                pRendererFields.AddField(strPopField2,strPopField2);
                pQueryFilter = new QueryFilterClass();
                pQueryFilter.AddField(strPopField1);
                pQueryFilter.AddField(strPopField2);
                pCursor = pTable.Search(pQueryFilter, true);
                //Make an array of the field numbers to iterate accross, this is to keep the code
generic in the number of bars to draw.
                fieldIndecies[0] = pTable.FindField(strPopField1);
                fieldIndecies[1] = pTable.FindField(strPopField2);
                firstValue = true;
                dmaxValue = 0;
                //Iterate across each feature
                pRowBuffer = pCursor.NextRow();
                while (pRowBuffer != null)
                {
                    // iterate   through each data field and update the maxVal if needed
                    for( lfieldIndex = 0 ; lfieldIndex<= numFields - 1;lfieldIndex++)
                    {
                        dfieldValue =(double)
pRowBuffer.get_Value(fieldIndecies[lfieldIndex]);
                        if (firstValue)
                        {
                            // Special case for the first value in a feature class
                            dmaxValue = dfieldValue;
                            firstValue = false;
                        }
                        else
                        {
                            if( dfieldValue > dmaxValue)
                            {
                                // we've got a new biggest value
                                dmaxValue = dfieldValue;
                            }
```

```
            }
        }
        pRowBuffer = pCursor.NextRow();
}
if (dmaxValue <= 0)
{
        MessageBox.Show("Failed to gather stats on the feature class");
        Application.Exit();
}
// Set up the chart marker symbol to use with the renderer
IBarChartSymbol pBarChartSymbol;
pBarChartSymbol = new BarChartSymbolClass();
pChartSymbol = (IChartSymbol) pBarChartSymbol;
pBarChartSymbol.Width = 12;
pMarkerSymbol = (IMarkerSymbol) pBarChartSymbol;
//Finally we've got the biggest value, set this into the symbol
pChartSymbol.MaxValue = dmaxValue;
//This is the maximum height of the bars
pMarkerSymbol.Size = 80;
//Now set up symbols for each bar
pSymbolArray = (ISymbolArray) pBarChartSymbol;
//      Add some colours in for each bar
pFillSymbol = new SimpleFillSymbolClass();
pFillSymbol.Color = getRGB(213, 212, 252);
pSymbolArray.AddSymbol((ISymbol) pFillSymbol);
pFillSymbol = new SimpleFillSymbolClass();
//This is a pastel green
pFillSymbol.Color = getRGB(193, 252, 179);
pSymbolArray.AddSymbol( (ISymbol) pFillSymbol);
//Now set the barchart symbol into the renderer
pChartRenderer.ChartSymbol = (IChartSymbol) pBarChartSymbol;
pChartRenderer.Label = "Population";
//set up the background symbol to use tan color
pFillSymbol = new SimpleFillSymbolClass();
pFillSymbol.Color = getRGB(239, 228, 190);
pChartRenderer.BaseSymbol = (ISymbol) pFillSymbol;
//Disable overpoaster so that charts appear in the centre of polygons
```

```
                pChartRenderer.UseOverposter = false;
                pChartRenderer.CreateLegend();
                //update the renderer and refresh the screen
                pGeoFeatureL.Renderer =(IFeatureRenderer)   pChartRenderer;
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);
            }
        }
}
```

上面的例子是使用水平柱子着色方式进行的，其结果如图 6-17 所示。

如果希望是柱子是堆积排列的话，只需要将代码中的下面语句

```
IBarChartSymbol pBarChartSymbol;
pBarChartSymbol = new BarChartSymbolClass();
```

更换为：

```
IStackedChartSymbol pBarChartSymbol;
pBarChartSymbol = new StackedChartSymbolClass();
```

上面的例子中，求不同字段的最大值的方法必须使用两个 Cursor 对象，读者可以尝试　只使用一个 Cursor 会发生什么结果以及思考为什么会出现这种错误。

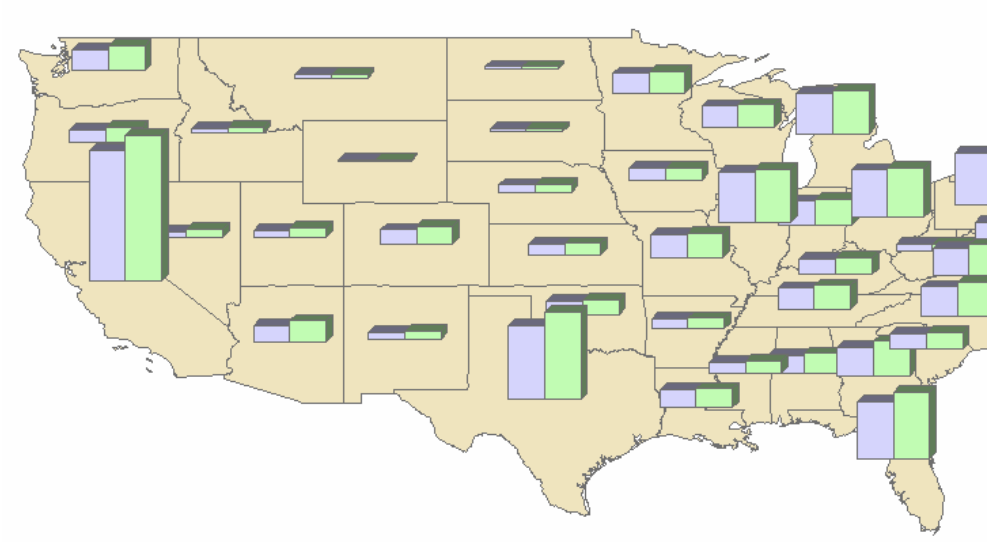


图 6-17　BarChart 着色

6.3.5.2PieChartRenderer 着色

PieChartRenderer 着色法是使用饼图来表现要素的多个属性之间的比率关系，通过这种饼图可以一目了然地看出哪些属性的值更多一点。PieChartRenderer 也是 ChartRenderer 的一种，它实现的是 IPieChartRenderer 接口，这种着色方式必须使用 PieChartSymbol 符号来修饰要素。

和上例一样，PieChartRenderer 对象同样需要找到参与着色的要素的某个字段的最大值给 ChartSymbol.MaxValue，这两种不同的着色方法不过是使用的 symbol 不一样而已。

如图 6-18 所示，它是使用饼图进行着色的一个例子。它包含了两种信息，一方面饼的大小可以显示不同要素的值的大小；另外，一个饼图中的不同颜色划分体现了一个要素的两个属性值的比例。如果绘制人口分布图，这种着色方式是非常理想的。下面是产生这个图像的代码段。

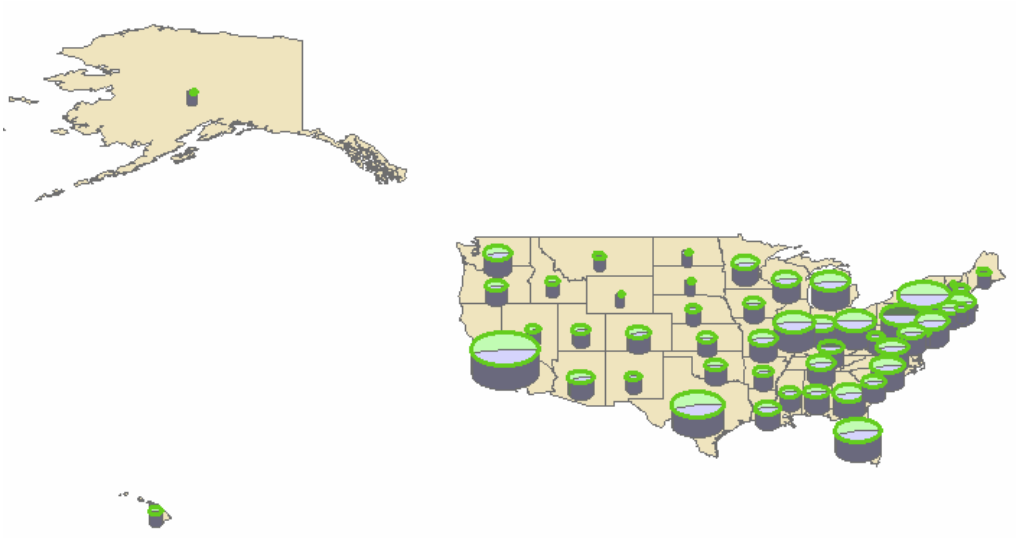


图 6-18　PieChartRenderer 例子

PieChartRenderer.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Utility.BaseClasses;
using stdole;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.TOCControl;
```

```
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.CATIDs;

namespace Symbology
{
    public class PieChartRenderer:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap ;
        IActiveView pActiveView;

        public PieChartRenderer()
        {
            base.m_caption = "PieChartRenderer";
            base.m_message = "PieChartRenderer";
            base.m_toolTip = "PieChartRenderer";
            base.m_category = "Symbology";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            IGeoFeatureLayer pGeoFeatureL;
            IFeatureLayer pFeatureLayer;
            ITable pTable;
            ICursor pCursor;
```

```
IQueryFilter pQueryFilter;
IRowBuffer pRowBuffer;
int numFields=2;
int[] fieldIndecies = new int[numFields];
int lfieldIndex;
double dmaxValue;
bool firstValue;
double dfieldValue;
IChartRenderer pChartRenderer;
IPieChartRenderer pPieChartRenderer;
IRendererFields pRendererFields;
IFillSymbol pFillSymbol;
IMarkerSymbol pMarkerSymbol;
ISymbolArray pSymbolArray;
IChartSymbol pChartSymbol;
string strPopField1 = "POP1990";
string strPopField2 = "POP1999";

pActiveView = m_hookHelper.ActiveView;
pMap = m_hookHelper.FocusMap;
pMap.ReferenceScale = pMap.MapScale ;
pFeatureLayer = (IGeoFeatureLayer) pMap.get_Layer(1);
pGeoFeatureL = (IGeoFeatureLayer) pFeatureLayer;
pTable = (ITable) pGeoFeatureL;
pGeoFeatureL.ScaleSymbols = false;
pChartRenderer = new ChartRendererClass();
pPieChartRenderer = pChartRenderer as IPieChartRenderer;
//Set up the fields to draw charts of
pRendererFields = (IRendererFields) pChartRenderer;
pRendererFields.AddField(strPopField1,strPopField1);
pRendererFields.AddField(strPopField2,strPopField2);

pQueryFilter = new QueryFilterClass();
pQueryFilter.AddField(strPopField1);
pQueryFilter.AddField(strPopField2);
pCursor = pTable.Search(pQueryFilter, true);

//Make an array of the field numbers to iterate accross, this is to keep the code
```

```
generic in the number of bars to draw.
                fieldIndecies[0] = pTable.FindField(strPopField1);
                fieldIndecies[1] = pTable.FindField(strPopField2);
                firstValue = true;
                dmaxValue = 0;

                //Iterate across each feature
                pRowBuffer = pCursor.NextRow();
                while (pRowBuffer != null)
                {
                    // iterate   through each data field and update the maxVal if needed
                    for( lfieldIndex = 0 ; lfieldIndex<= numFields - 1;lfieldIndex++)
                    {
                        dfieldValue =(double)
pRowBuffer.get_Value(fieldIndecies[lfieldIndex]);
                        if (firstValue)
                        {
                            // Special case for the first value in a feature class
                            dmaxValue = dfieldValue;
                            firstValue = false;
                        }
                        else
                        {
                            if( dfieldValue > dmaxValue)
                            {
                                // we've got a new biggest value
                                dmaxValue = dfieldValue;
                            }
                        }
                    }
                    pRowBuffer = pCursor.NextRow();
                }
                if (dmaxValue <= 0)
                {
                    MessageBox.Show("Failed to gather stats on the feature class");
                    Application.Exit();
                }
```

```
// Set up the chart marker symbol to use with the renderer
IPieChartSymbol pPieChartSymbol;
pPieChartSymbol = new PieChartSymbolClass();
pChartSymbol = (IChartSymbol) pPieChartSymbol;
//饼图使用顺时针方法
pPieChartSymbol.Clockwise = true;
//饼图有外轮廓线
pPieChartSymbol.UseOutline = true;
ILineSymbol pOutline;
pOutline = new SimpleLineSymbolClass();
pOutline.Color = getRGB(100, 205, 30);
pOutline.Width = 2;
//设置外轮廓线的样式
pPieChartSymbol.Outline = pOutline;
pMarkerSymbol = (IMarkerSymbol) pPieChartSymbol;
//Finally we've got the biggest value, set this into the symbol
pChartSymbol.MaxValue = dmaxValue;
pMarkerSymbol.Size = 30;
//Now set up symbols for each bar
pSymbolArray = (ISymbolArray) pPieChartSymbol;
//      Add some colours in for each bar
pFillSymbol = new SimpleFillSymbolClass();
pFillSymbol.Color = getRGB(213, 212, 252);
pSymbolArray.AddSymbol((ISymbol) pFillSymbol);
pFillSymbol = new SimpleFillSymbolClass();
//This is a pastel green
pFillSymbol.Color = getRGB(193, 252, 179);
pSymbolArray.AddSymbol( (ISymbol) pFillSymbol);

//Now set the barchart symbol into the renderer
pChartRenderer.ChartSymbol = (IChartSymbol) pPieChartSymbol;
pChartRenderer.Label = "Population";
//set up the background symbol to use tan color
pFillSymbol = new SimpleFillSymbolClass();
pFillSymbol.Color = getRGB(239, 228, 190);
pChartRenderer.BaseSymbol = (ISymbol) pFillSymbol;
//Disable overpoaster so that charts appear in the centre of polygons
pChartRenderer.UseOverposter = false;
```

```
                pPieChartRenderer.MinSize = 3;
                //设置最小值，用于尺寸比例
                pPieChartRenderer.MinValue = 453588;
                pPieChartRenderer.FlanneryCompensation = false;
                pPieChartRenderer.ProportionalBySum = true;
                pChartRenderer.Label = "Population";
                //产生图例对象
                pChartRenderer.CreateLegend();
                //update the renderer and refresh the screen
                pGeoFeatureL.Renderer =(IFeatureRenderer)   pChartRenderer;
                pGeoFeatureL.ScaleSymbols = true;
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);
            }
        }
}
```

6.3.6 DotDensityRenderer 着色法

DotDensityRenderer 着色法使用 DotDensityRenderer 对象来实现。DotDensityRenderer 对象使用 DotDensityFillSymbol 符号来对一个 Polygon 类型的要素图层进行着色。这种着色法根据随机分布的点的密度来表现要素的某个属性值的大小，而这些点的多少是由 DotValue 值来决定的。

DotDensityRenderer 也可以对图层的多个属性进行着色，这个着色使用不同的点符号 (MarkerSymbol)，但是这种着色方式并不值得推荐，因为点密度大的情况下往往引起混乱。

DotDensityRenderer 类实现了 IDotDensityRenderer 接口，这个接口定义了使用点密度着　色的方法和属性，如 IDotDensityRenderer::DotDensitySymbol 是用来确定着色用的点符号；　IDotDensityRenderer::CreateLegend 方法用于产生图例，当然图例并不会在 MapControl 控件中显示出来。

如图 6-19 所示，它是一幅使用点密度着色法制作的专题图。

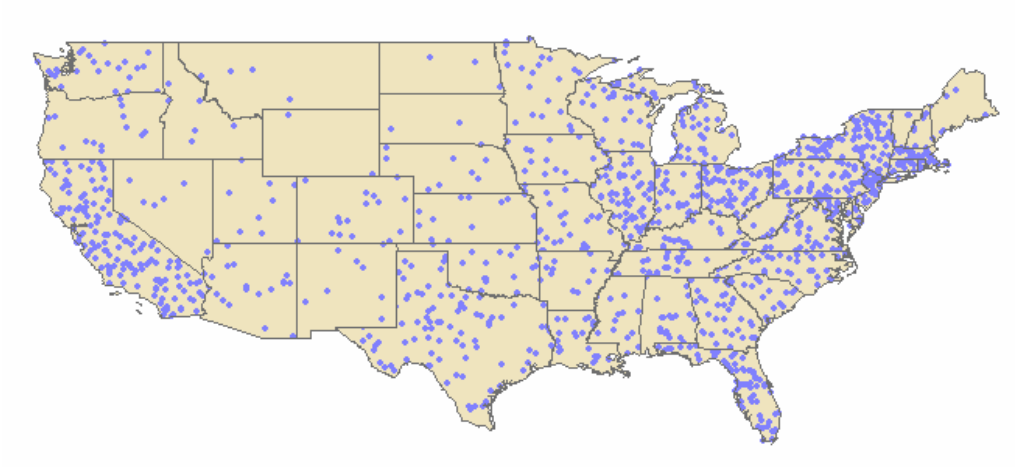


图 6-19　DotDensityRenderer 例子

具体实现代码如下：

DotDensityRenderer.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Utility.BaseClasses;
using stdole;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.TOCControl;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;
using ESRI.ArcGIS.Utility.CATIDs;

namespace Symbology
{
    public class DotDensityRenderer:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap;
```

```
        IActiveView pActiveView;

        public DotDensityRenderer()
        {
            base.m_caption = "DotDensityRenderer";
            base.m_message = "DotDensityRenderer";
            base.m_toolTip = "DotDensityRenderer";
            base.m_category = "Symbology";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            IGeoFeatureLayer pGeoFeatureL;
            IDotDensityRenderer pDotDensityRenderer;
            IDotDensityFillSymbol pDotDensityFillS;
            IRendererFields pRendererFields;
            ISymbolArray pSymbolArray;
            ISimpleMarkerSymbol pSimpleMarkerS;
            string strPopField = "POP1990";

            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pGeoFeatureL = (IGeoFeatureLayer) pMap.get_Layer(1);
            pDotDensityRenderer = new DotDensityRendererClass();
            pRendererFields = (IRendererFields) pDotDensityRenderer;
            pRendererFields.AddField(strPopField,strPopField);
          pDotDensityFillS = new DotDensityFillSymbolClass();
            pDotDensityFillS.DotSize = 4;
            pDotDensityFillS.Color = getRGB(0, 0, 0);
```

```
                pDotDensityFillS.BackgroundColor = getRGB(239, 228, 190);
                pSymbolArray = (ISymbolArray) pDotDensityFillS;
                pSimpleMarkerS = new SimpleMarkerSymbolClass();
                pSimpleMarkerS.Style = esriSimpleMarkerStyle.esriSMSCircle ;
                pSimpleMarkerS.Size = 4;
                pSimpleMarkerS.Color = getRGB(128, 128, 255);
                pSymbolArray.AddSymbol( (ISymbol) pSimpleMarkerS);
                pDotDensityRenderer.DotDensitySymbol = pDotDensityFillS;
                pDotDensityRenderer.DotValue = 200000;
                pDotDensityRenderer.CreateLegend();
            pGeoFeatureL.Renderer = (IFeatureRenderer) pDotDensityRenderer;
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);

        }
    }
}
```

6.3.7 着色中使用的计算方法

在使用着色对象制作地图专题图的过程中，系统常常需要使用这些要素图层的数值字段值来进行各种统计分析，对这些数值数据的处理，ArcObjects 也提供了专门的对象来完成。本节将介绍 ArcObjects 中关于处理数值的对象。

- **Classify 对象**

尽管作者在 ClassBreakRenderer 着色法中曾经告诉读者 Classify 对象不能在 C#.NET 中使用，但是介绍 Classify 对象对 VB 程序员还是有帮助的。Classify 是 ArcObjects 中的一个分级对象集合，它们专门用于将数值字段按照不同的方式进行分级。

Classify 是一个抽象类，它拥有五个子分级对象，即 DefinedInterval、EqualInterval、NaturalBreaks、StandardDeviation 和 Quantile。所有的这些对象都要求使用 Histogram 型数据，在给定需要划分的级数后，它就可以对一个数组的数据进行分级计算，得出合适的分级点。

Histogram 数据是由两个数组组成的，第一个数组是排好序的数值数组，第二个数组是与之相关联的频率值数组，这个数据可以通过 TableHistogram 对象来得到。

Histogram 对象有三种，如果图层对象支持 ITable 接口，就可以使用

TableHistogram 对象来获取它的数值字段的值；如果 Histogram 数据已经保存在一个 Histogram 对象中，则还可以使用 DataHistogram 来获得到；第三种是 TinHistogram 对象，它可以在数据源为 Tin 的时候使用。

使用 ITableHistogram 接口可以设置需要传入的表和字段，ITableHistogram::Table 和 ITableHistogram::Field 属性是获得一个表的数值字段的 Histogram 数据时必须设置的。在设置完这两个属性后，程序员需要 QI 到 IHistogram 接口，使用 GetHistogram 方法得到两个数组。

关于这个对象的使用代码，读者可以查看着色对象的内容。除此之外，这个对象还支持 IStatisticsResults 接口，通过这个接口它可以得到一个数值字段的各种统计值，如数值和、最大值、最小值和标准方差等。

IClassify 接口定义了所有的分级对象都使用的属性和方法，它多用于传入 Histogram 数据到一个具体的分级对象中去。IClassify::ClassBreaks 用于设置分级的数目，而 SetHistogramData 方法则可以将 Histogram 对象传出的两个数组传入分级对象。

EqualInterval 对象支持 IClassifyMinMax 接口，它可以得到用于设置在需要分级的字段的数据中人为设置需要分级的最大值和最小值范围。这种分类方法的取值范围是等间距划分的，比如取值范围为 0-90，当需要被分为三级时，其范围分别是 0-30、31-60、61-90，其范围是相等的。

DefinedInterval 对象支持 IntervalRange 接口，这种对象是通过设定间距的方式来进行划分，例如在人口统计中，用户可以设置 10 岁为一个增量值，则分级范围是 0-10、11-20 等，因此，落入这个每个范围的要素的数目可能是不一样的。

NaturalBreaks 对象是通过一个 Jenk 的方法来对数据进行分类，它保证了类间的变化最大化，类内的变化最小化。

Quantile 对象是分数位分类，它是使用相同数目的特征来进行分类的。这种分类方法对于分等级的数值特别有效。

StandardDeviation 对象会创建偶数个类，这些类会表示相对于所有要素的平均值的整体或者部分偏差。

关于这些分级方法的具体用法和特点，读者可以参阅 ESRI 的相关资料。本书对此不作过多的理论介绍。

- **已知一个图层和这个图层要素类中的一个数值字段，求这个字段的各种属性**

下面是一个非常实用的函数，专门用于计算各种统计值，如多个数组的最大值、

最小值、和、平均值、均方差等。它使用了 DataStatistics 对象，需要传入的参数是一个要素图层名和需要计算的字段名，以及需要返回的值类型。

```
private double sumValues(IGeoFeatureLayer pGeoFeatLayer, string strfield, int status)
    {
        ITable pTable;
        pTable = pGeoFeatLayer as ITable;
        ICursor pCursor;
        double dValue = 0;
        pCursor = pTable.Search(null, true);
        IDataStatistics pDataStatistics;
        IStatisticsResults pStatisticsResults;
        pDataStatistics = new DataStatisticsClass();
        pDataStatistics.Cursor = pCursor;
        pDataStatistics.Field = strfield;
        pStatisticsResults = pDataStatistics.Statistics;
        //依照state的不同而返回不同的统计值
        switch(status)
        {
            case 0:
                dValue = pStatisticsResults.Count;
                break;
            case 1:
                dValue =  pStatisticsResults.Maximum;
                break;
            case 2:
                dValue =  pStatisticsResults.Mean;
                break;
            case 3:
                dValue =  pStatisticsResults.Minimum;
                break;
            case 4:
                dValue =  pStatisticsResults.StandardDeviation;
                break;
            case 5:
                dValue =  pStatisticsResults.Sum;
                break;
        }
        return dValue;
```

```
}
```

在这个函数中有“pCursor=pTable.Search(null, true)”一句代码，其null代表的是一个过滤器对象(QueryFilter，null表示这里无任何过滤条件)，它意味着程序员还可以选择一个字段的某些值进行统计。如计算一个States图层中面积大于100000的州1990年人口的各种统计值。读者可以自己改写这个函数。

- **CalcRendererValues 对象**

CalcRendererValues 对象是 ChartRenderer 对象对一个要素图层进行着色运算时需要使用的工具，它用于计算最大值、最小值和要素的数目。

这个过程首先必须设置 ChartRenderer 对象，然后使用 ICalcRendererValues::setData 方法。而计算最大值和最小值，只需要调用 CalcMinMax 方法就可以了。

使用 ChartRenderer 对象计算最大、最小值会出现两种情况：

1. 如果 Pie 图的尺寸比率是基于单个字段的，那么在进行最大、最小值计算时必须设置 PieChartRenderer::ProportionalField 属性，这个属性需要设置一个数值型字段。
2. 如果 Pie 图的尺寸是由所有字段的和(Sum)所决定的，那么在计算之前需要设置 IPieChartRenderer.ProportionalBySum=True，则它返回的最大值和最小值是所有字段的最大值和最小值之和。

下面是一段代码来演示这个对象的使用情况：

```
pPieChartRenderer.ProportionalField = "POP1990";
//……
ICalcRendererValues pCalcRendererValues;
pCalcRendererValues = new CalcRendererValuesClass();
//设置图层和着色对象
pCalcRendererValues.SetData(pFeatLayer, pPieChartRenderer as
IFeatureRenderer);
double minVal;
double maxVal;
//计算最大值和最小值
pCalcRendererValues.CalcMinMax(out minVal,out maxVal);
pPieChartRenderer.MinValue = minVal;
```

6.4 地图的标注

对于一幅地图而言标注是非常重要的。一张标注良好的地图可以通过文字清晰地显示地图要素中蕴涵的各种内在信息。ArcObjects 中的文字标注分为两种，一个是标注(Label)，另一个是注记(Annotation)。它们都是在要素图层上以文字来揭示地图要素的某些信息，如地名、人口数目、特产等。

在 ArcObjects 中程序员可以使用两种方法做到标注，一种是自己添加 TextElement 到文档对象，另一种就是使用 ArcObjects 提供的 Annotation 对象，它的功能更加强大和丰富。

6.4.1 TextElement 标注

使用 TextElement 对象来添加标注的思路简单，而且它可以控制标注字体的样式。标注的流程是首先在一个要素图层进行查询，然后通过一个 cursor 取得所有要进行标注的要素，然后遍历要素做如下操作：

1. 获取单个要素的 Envelope 属性，即要素几何图形的包络线。
2. 新建一个 TextElement 对象，其 Text 为要素的某个字段属性，而 Geometry 是要素包络线的中间点。
3. 将新建的文字元素加入地图中。
4. 刷新视图，让标注文本显示出来。

下面是一个使用文本元素来进行标注的过程。这个例子使用了本书前面几个章节介绍　的内容。过程需要传入的参数是图层所在的 Map 对象，要素图层名和要标注的字段名。代　码如下：

```
        private void TextElementLabel(IMap pMap,IFeatureClass pFeatClass, string
strName)
        {
            //获得要素类查询后的要素游标对象
            IFeatureCursor pFeatCursor;
            pFeatCursor = pFeatClass.Search(null, true);
            IFeature pFeat;
            //得到第一个要素
            pFeat = pFeatCursor.NextFeature();
            while( pFeat != null )
            {
```

```
                    //寻找要素类的字段集合
                    IFields pFields;
                    pFields = pFeat.Fields;
                    int i;
                    //找出标注字段的索引号
                    i = pFields.FindField(strName);
                    //得到要素的包络线
                    IEnvelope pEnv;
                    pEnv = pFeat.Extent;
                    IPoint pPoint;
                    pPoint = new PointClass();
                    //确定字符元素的Geometry，它是包络线的中心点
                    pPoint.PutCoords(pEnv.XMin + pEnv.Width / 2, pEnv.YMin + pEnv.Height
/ 2);
                    //新建字体对象，设置字体的属性
                    stdole.IFontDisp pFont;
                    pFont = new stdole.StdFontClass() as stdole.IFontDisp;
                    pFont.Name = "arial";
                    //新建RGB颜色对象
                    IRgbColor pRGB;
                    pRGB = new RgbColorClass();
                    pRGB.Red = 110;
                    pRGB.Blue = 200;
                    pRGB.Green = 60;
                    //产生一个文本符号
                    ITextSymbol pTextSymbol;
                    pTextSymbol = new TextSymbolClass();
                    //设置文本符号的大小
                    pTextSymbol.Size = 3;
                    pTextSymbol.Font = pFont;
                    pTextSymbol.Color = pRGB;
                    //产生一个文本元素对象
                    ITextElement pTextEle;
                    IElement pEle;
                    pTextEle = new TextElementClass();
                    pTextEle.Text = pFeat.get_Value(i).ToString();
                    pTextEle.ScaleText = true;
                    pTextEle.Symbol = pTextSymbol;
```

```
                pEle = pTextEle as IElement;
                pEle.Geometry = pPoint;
                IActiveView pActiveView;
                IGraphicsContainer pGraphicsContainer;
                pActiveView = pMap as IActiveView;
                pGraphicsContainer = pMap as IGraphicsContainer;
                //将元素添加进Map对象
                pGraphicsContainer.AddElement(pEle, 0);
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null,
null);
                pPoint = null;
                pEle = null;
                pFeat = pFeatCursor.NextFeature();
            }
        }
```

这个过程的功能是很强大的，读者注意一下这条语句：pFeatCursor = pFeatClass.Search(null, true)，它返回的要素游标将指向要素图层的所有要素，用户可以将 null 换成一个 QueryFilter 对象，使用 SQL 语句来确定到底哪些要素需要进行标注，读者可以自己改写这个过程。

可用下面的方法对以上文本标注进行测试，其结果如图 6-20 所示，与使用自动标注的效果差不多。

```
    private void mnuTextElementLabel_Click(object sender, System.EventArgs e)
        {
            IMap pMap;
            IFeatureLayer pFeatureLayer;
            IFeatureClass pFeatureClass;
            string pFieldName = "STATE_NAME";
            pMap = axMapControl1.Map;
            pFeatureLayer = pMap.get_Layer(1) as IFeatureLayer;
            pFeatureClass = pFeatureLayer.FeatureClass;
            TextElementLabel(pMap,pFeatureClass,pFieldName);

        }
```

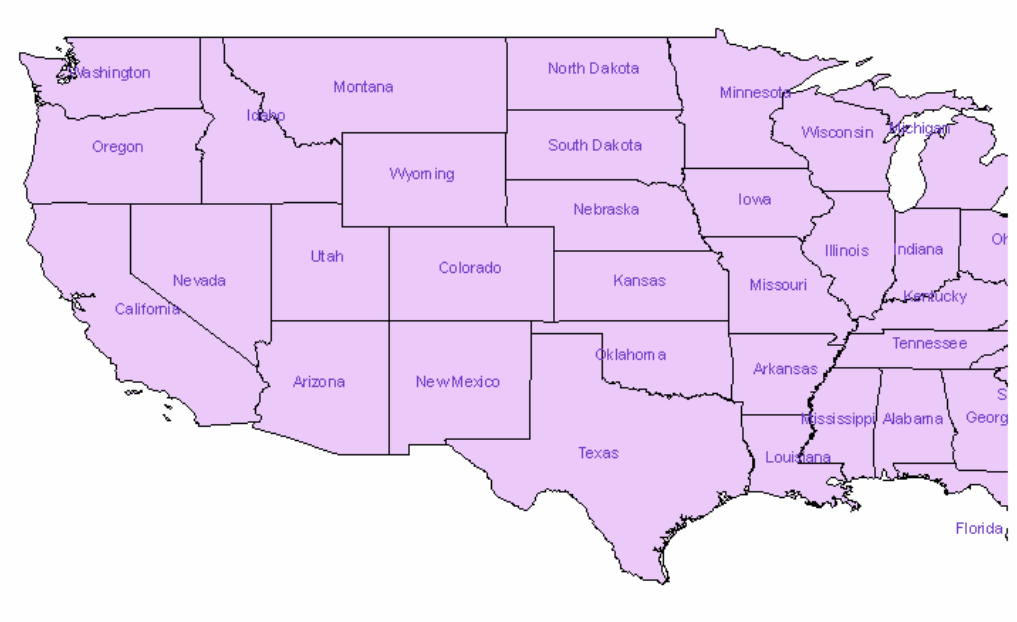


图 6-20 使用 TextElement 标注

这种标注方法仍然不能实现标注文字随视图大小而自动缩放，为了实现这个效果，读者可以采取以下思路解决：

- 寻找到一个地图视图与字体大小适应的比例关系，通过这个关系计算出某种尺寸视图中合适的字体大小值。
- 在视图缩放后的事件中清除已经标注的文本。
- 执行 TextElementLabel 过程，这个过程需要改变字体大小参数。

6.4.2 ArcObjects 中的标注

除了上一节介绍的使用 TextElement 对象来进行要素标注外，A0 库中也有专门用于标注操作的对象供程序员使用，使用这些对象可以在地图上进行自动标注，可以使用 SQL 语句对需要标注的要素进行选择，可以自动调整标注字符防止互相覆盖等。

ArcObjects 提供的标注对象进行的操作有两种形式：一种是 Label(标注)，在 ArcMap 中表现为设置图层的 Label 属性后，图层就以 Label 属性中设置的字段进行标注；另一种是 Annotation(注记)，它以更复杂的方法和属性对要素图层进行注记，这个过程是自动进行的，不需要用户进行干预，而且注记的内容还可以保存到地理数据库中。这两种标注方法在 ArcObjects 中使用的是一个标注对象集，本节主题将讨论这个复杂的标注过程。

AnnotateLayerPropertiesCollection 对象是一个要素图层的属性，它是一个标注集对象 (LabelingSets)的集合，在这个集合内可以存在多个不同的标注集，即 LabelEngineLayerProperties 对象。标注集是与某个要素图层相关联的，用于描述要素图层是如何被标注的。

程序员可以使用 IGeoLayer::AnnotationProperties 属性来得到一个要素图层的 AnnotateLayerPropertiesCollection 对象。当然，由于这个对象是一个 CoClass 对象，用户也可以自己新建一个 AnnotateLayerPropertiesCollection 对象。AnnotateLayerPropertiesCollection 作为一个集合对象，支持 IAnnotateLayerPropertiesCollection 接口，这个接口的属性和方法可以让用户获得、添加或者删除 LabelEngineLayerProperties 对象，也就是说 AnnotateLayerPropertiesCollection 对象中管理的是 LabelEngineLayerProperties 对象。

LabelEngineLayerProperties 对象维持着一个要素图层的标注实例。对一个要素图层而言，标注是一种属性，它用于设定标注对象如何在要素上排列和修饰要素，它也可以基于要素图层中要素类的某个属性或者脚本表达式来设置标注，如使用什么符号来修饰标注的文字、标注文字在要素的位置、标注对象有冲突时哪一种的权重大等。

LabelEngineLayerProperties 类实现了 IAnnotateLayerProperties 接口，它定义的属性和方法用于回答哪些要素被标注和标注尺寸的问题。IAnnotateLayerProperties::WhereClause 属性用于设置一个 SQL 语句，可以用于确定哪些要素可以被标注；IAnnotateLayerProperties:: AnnotationMaximunScale 和 AnnotationMinimumScale 可以设置标注文字的最大和最小范围。

ILabelEngineLayerProperties 接口也被 LabelEngineLayerProperties 对象实现，它则用于控制标注过程中的主要属性，如设置脚本、文字符号和标注文字排放等。ILabelEngineLayerProperties::BasicOverposterLayerProperties 属性可以用于设置标注文本如何被放置，而 ILabelEngineLayerProperties::Expression 则可以用于输入 VBScript 或 JavaScript 脚本，这是一个很高级的功能。

ILabelEngineLayerProperties::BasicOverposterLayerProperties 属性需要设置一个 BasicOverposterLayerProperties 对象，这个对象用于设置标注文本的位置，如标注文字如何放置，如何处理文字之间的冲突等。这个对象有几个重要的属性，如 IBasicOverposterLayerProperties::LineLabelPlacementPriorities、LineLabelPosition 和 PointPlacementPriorities 等，它们分别用于设置标注文本的摆设路径的权重，标注文本的排放位置和与一个点相关的标注路径的权重等。

ArcObjects 中的标注对象比较复杂，下面给出简单的标注代码，它可以实现基于要素图层的某个属性进行自动的标注。

```
        private void Annotation(IGeoFeatureLayer pGeoFeatLyr, string annoField)
        {
            IGeoFeatureLayer pGeoFeatLayer;
            pGeoFeatLayer = pGeoFeatLyr;
            //得到图层的标注属性集合对象
            IAnnotateLayerPropertiesCollection pAnnoProps;
            pAnnoProps = pGeoFeatLyr.AnnotationProperties;
            //清空这个集合内的对象
            pAnnoProps.Clear();
            IAnnotateLayerProperties pAnnoLayerProps;
            ILineLabelPosition pPosition ;
            ILineLabelPlacementPriorities pPlacement;
            IBasicOverposterLayerProperties pBasic;
            ILabelEngineLayerProperties pLabelEngine ;
            //新建标注文本的文本符号
            ITextSymbol pTextSyl;
            pTextSyl = new TextSymbolClass();
            stdole.StdFont pFont;
            pFont = new stdole.StdFontClass() ;
            pFont.Name = "verdana";
            pFont.Size = 10;
            pTextSyl.Font = pFont as stdole.IFontDisp;
            pTextSyl.Color = HSVColor(250, 160, 200);
            //设置标注文本的位置
            pPosition = new LineLabelPositionClass();
            pPosition.Parallel = false;
            pPosition.Perpendicular = true;
            pPlacement = new LineLabelPlacementPrioritiesClass();
            pBasic = new BasicOverposterLayerPropertiesClass();
            pBasic.FeatureType =
esriBasicOverposterFeatureType.esriOverposterPolyline;
            pBasic.LineLabelPlacementPriorities = pPlacement;
            pBasic.LineLabelPosition = pPosition;
            //新建一个图层标注引擎对象，设置它的属性
            pLabelEngine = new LabelEngineLayerPropertiesClass();
            pLabelEngine.Symbol = pTextSyl;
            pLabelEngine.BasicOverposterLayerProperties = pBasic;
            //field必须是这种样式，如"[STATE_NAME]"，它用于设置按照要素的哪个属性进
```

```
行标注
                pLabelEngine.Expression = annoField;
                pAnnoLayerProps = pLabelEngine as IAnnotateLayerProperties;
                pAnnoProps.Add(pAnnoLayerProps);
                pGeoFeatLyr.DisplayAnnotation = true;
    axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground,null,null);
        }
```

测试代码如下：

```
        private void mnuAnnotation_Click(object sender, System.EventArgs e)
        {
            IMap pMap;
            IGeoFeatureLayer pGeoFeatureLayer;
            string pFieldName = "[STATE_NAME]";
            pMap = axMapControl1.Map;
            pGeoFeatureLayer = pMap.get_Layer(1) as IGeoFeatureLayer;
            Annotation(pGeoFeatureLayer,pFieldName);

        }
```

消除标注的方法也很简单，程序员只要设置将进行标注的要素图层的 DisplayAnnotation 设置为 False，并刷新即可。

```
        pGeoFeatLyr.DisplayAnnotation = false
    axMapControl1.CtlRefresh(esriViewDrawPhase.esriViewBackground,null,null);
```

6.5 如何与地图进行互动

DisplayFeedback 是让用户能够使用鼠标与控件进行可视化交互的对象集，这种交互包括添加、移动要素或者图形元素以及改变它们的几何形状，例如产生和修改要素或者图形元素的 Geometry 属性等。这个对象也可以用于不产生任何 Geometry 的任务中，如测量两点之间的距离等。在 ArcMap 中，这些对象多用于绘制和编辑任务中，当然这些功能也可以在二次开发中使用。

在 MapControl 控件的 MouseDown 事件中用户可以使用的 TrackLine、TrackPolygon 等方法获得 Line、Polygon 等几何形体。对于“事件”这个概念很熟悉的读者可能会产生一个疑问，即通过这些方法得到一个几何形体对象时，鼠标其实产生了 MouseDown，MouseMove 和 MouseUp 三个事件，而控件是如何通过一个 MouseDown 事件就产生了一个几何形体对象的呢?显然控件对这些内容进行了封装。

了解在 MapControl 或 PageLayoutControl 控件上产生一个几何形体对象的本质，就需要探讨 DisplayFeedback 对象集的功能。

DisplayFeedback 对象需要在鼠标事件的配合下完成它的功能，但并不是所有的 DisplayFeedback 对象使用的鼠标事件都是一样的，使用何种事件取决于任务的类型，如添加一个新的 Polygon 对象，会使用到 MouseDown、MouseMove 和 MouseUp 三种事件，而对 Polygon 进行整形的时候则会使用 MouseDown、MouseMove 和 MouseDblClick 事件。

尽管所有的 DisplayFeedback 对象(GroupFeedback 除外)都有相似的函数方法，但它们的行为大概分为两种：

1. DisplayFeedback 对象返回一个新的 Geometry。这些对象的接口定义了一个 Stop 方法来返回新的几何形体对象，这些对象包括下面几种：

NewEnvelopeFeedback	NewBezierCurveFeedback	NewDimensionFeedback
NewLineFeedback	NewPolygonFeedback	ModifyDimensionFeedback
MoveEnvelopeFeedback	MoveLineFeedback	MovePointFeedback
MovePolygonFeedback	BezierMovePointFeedback	LineMovePointFeedback
PolygonMovePointFeedback	ReshapeFeedback	ResizeEnvelopeFeedback
StretchLineFeedback		

2. DisplayFeedback 对象仅仅为了显示的目的，这时系统需要得到计算新的几何形体对象。例如使用鼠标的起始点位置去更新或者改变一个几何形体对象，这些对象有 MoveGeometryFeedback、MoveImageFeedback、NewMultiPointFeedback 和 VertexFeedback 等。

所有的 DisplayFeedback 类都实现了 IDisplayFeedback 接口，这个接口定义了它们的公共方法和属性，如 Display、Symbol、MoveTo 和 Refresh 等方法。

一般地，当 DisplayFeedback 对象初始化的时候，程序员需要给它们设置 Display 和 Symbol 属性，然后才让它们开始 MoveTo。如果不明确设置一个 Symbol 属性，系统将会给 DisplayFeedback 一个缺省的 Symbol 对象。

6.5.1 产生新 Geometry 的 Feedback 对象

6.5.1.1NewLineFeedback、NewPolygonFeedback、NewBezierCurveFeedback

NewLineFeedback，NewPolygonFeedback，NewBezierCurveFeedback 三个对象用来返回各自的新几何形体对象，如 Polyline、Polygon 和 BezierCurve(贝塞尔曲线)。这三个 Feedback 对象在产生几何形体时需要的鼠标事件都是 MouseDown、MouseMove 和 MouseDblClick，这是因为在产生多义线、多边形和贝塞尔曲线都需要先点击一点作为起点，然后鼠标移动到下一点处，这样逐个产生新点。如果要结束这个过程，则需要双击鼠标左键。

三个对象尽管产生的几何形体对象不一样，但是它们使用的方法都是大同小异的。三个类实现的接口定义的方法也基本一致，如在 MouseDown 事件中使用 Start 方法添加一个起始点或者添加点；在 MouseMove 事件中使用 MoveTo 方法来移动几何形体对象到一个新的点；在 MouseDblClick 事件中 Stop 方法将返回产生的几何形体对象。

下面以工具方式实现了创建 Polyline（通过 NewLineFeedback 对象）功能：

CreatePolylineByFeedback.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class CreatePolylineByFeedback : BaseTool
    {
        //类的公共成员变量
        IMap pMap;
        IActiveView pActiveView;
        IGraphicsContainer pGraphicsContainer;
        //定义一个NewLineFeedback对象
        INewLineFeedback pLineFeedback;
        IHookHelper m_hookHelper = new HookHelperClass();
        public CreatePolylineByFeedback()
        {
```

```
            base.m_caption = "CreatePolylineByFeedback";
            base.m_message = "CreatePolylineByFeedback";
            base.m_toolTip = "CreatePolylineByFeedback";
            base.m_category = "CreateGeometry";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }
        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pGraphicsContainer = pMap as IGraphicsContainer;

            //获得鼠标在控件上点击的位置，产生一个点对象
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
            if ( pLineFeedback == null )
            {
                //如果还没有开始画线，则产生新的newLineFeedback对象，并用start方法
开始
                pLineFeedback = new NewLineFeedbackClass();
                pLineFeedback.Display = pActiveView.ScreenDisplay;
                pLineFeedback.Start(pPt);
            }
            else
            {
                //如果已经画了第一条，则只需要使用AddPoint添加新点
                pLineFeedback.AddPoint(pPt);
            }
        }
```

```
        public override void OnMouseMove(int Button, int Shift, int X, int Y)
        {
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
            //MoveTo方法继承自IDisplayFeedback接口的定义
            if (pLineFeedback != null)   pLineFeedback.MoveTo(pPt);
        }

        public override void OnDblClick()
        {
            IGeometry pGeom;
            pGeom = pLineFeedback.Stop();
            pLineFeedback = null;
            //使用下面的方法添加一个元素
            AddLineElement(pGeom, pGraphicsContainer);
        }

        private void AddLineElement(IGeometry pGeom , IGraphicsContainer
pGraphicsContainer)
        {
            ISimpleLineSymbol pLineSym;
            pLineSym = new SimpleLineSymbolClass();
            pLineSym.Color = getRGB(110, 20, 45);
            pLineSym.Width = 2;
            pLineSym.Style = esriSimpleLineStyle.esriSLSDashDotDot;
            ILineElement plineEle;
            plineEle = new LineElementClass();
            plineEle.Symbol = pLineSym;
            IElement pEles;
            pEles = plineEle as IElement;
            pEles.Geometry = pGeom;
            pGraphicsContainer.AddElement(pEles, 0);
            //必须刷新一下，否则是看不到添加的图形元素的
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
        }

        private IRgbColor getRGB(int r,int g,int b)
        {
```

```
                IRgbColor pColor;
                pColor = new RgbColorClass();
                pColor.Red = r;
                pColor.Green = g;
                pColor.Blue = b;
                return pColor;
            }
        }
}
```

6.5.1.2绘制圆形和 Envelope 对象

newEnvelopeFeedback 和 newCircleFeedback 两个对象返回的也是几何图形，但是它们的事件却是另外的三种类型：MouseDown，MouseMove，MouseUp。因为这矩形和圆形的绘制不需要双击鼠标来显示图形绘制完成。

当绘制一个圆形对象的时候，用户首先选择一个点作为圆心，然后移动鼠标，在释放鼠标的点处确定了圆的半径，这样一个圆就绘制完成了。

下面的代码简要介绍了如何生成一个 Circle 图形：

CreateCircleByFeedback.cs

```
using System;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class CreateCircleByFeedback : BaseTool
    {
        IMap pMap;
        IActiveView pActiveView;
        IGraphicsContainer pGraphicsContainer;
        //定义一个NewCircleFeedback对象
       INewCircleFeedback pCircleFeedback;
        IHookHelper m_hookHelper = new HookHelperClass();
```

```
        public CreateCircleByFeedback()
        {
            base.m_caption = "CreateCircleByFeedback";
            base.m_message = "CreateCircleByFeedback";
            base.m_toolTip = "CreateCircleByFeedback";
            base.m_category = "CreateGeometry";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pGraphicsContainer = pMap as IGraphicsContainer;

            //获得鼠标在控件上点击的位置，产生一个点对象，即圆心
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
            if (pCircleFeedback == null)
            {
                pCircleFeedback = new NewCircleFeedbackClass();
                pCircleFeedback.Display = pActiveView.ScreenDisplay;
                pCircleFeedback.Start(pPt);
            }
        }

    //鼠标放开产生的事件，返回一个 pGeom，将这个几何形体对象设置到一个图形元素上，然
后将元素添加进控件：
        public override void OnMouseMove(int Button, int Shift, int X, int Y)
```

```
        {
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
            //MoveTo方法继承自IDisplayFeedback接口的定义
            if (pCircleFeedback != null)    pCircleFeedback.MoveTo(pPt);
        }
        public override void OnMouseUp(int Button, int Shift, int X, int Y)
        {
            IGeometry pGeo;
            pGeo = pCircleFeedback.Stop();
            pCircleFeedback = null;
            AddCircleElement(pGeo, pActiveView);
        }

        private void AddCircleElement(IGeometry pGeo, IActiveView pAv )
        {
            ISegmentCollection pSegColl;
            pSegColl = new PolygonClass();
            object Missing1 = Type.Missing;
            object Missing2 = Type.Missing;
            pSegColl.AddSegment(pGeo as ISegment,ref Missing1, ref Missing2);
            ISimpleLineSymbol pLineSym;
            pLineSym = new SimpleLineSymbolClass();
            pLineSym.Color = getRGB(110, 22, 125);
            pLineSym.Style = esriSimpleLineStyle.esriSLSSolid;
            pLineSym.Width = 2;
            ISimpleFillSymbol pSimpleFillSym;
            pSimpleFillSym = new SimpleFillSymbolClass();
            pSimpleFillSym.Color = getRGB(66, 55, 145);
            pSimpleFillSym.Outline = pLineSym;
            pSimpleFillSym.Style = esriSimpleFillStyle.esriSFSCross;
            IElement pPolygonEle;
            pPolygonEle = new CircleElementClass();
            pPolygonEle.Geometry = pSegColl as IGeometry ;
            IFillShapeElement pFillEle;
            pFillEle = pPolygonEle as IFillShapeElement;
            pFillEle.Symbol = pSimpleFillSym;
            IGraphicsContainer pGraphicsContainer;
```

```
            pGraphicsContainer = pAv as IGraphicsContainer;
            pGraphicsContainer.AddElement(pFillEle as IElement , 0);
            pAv.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
        }

        private IRgbColor getRGB(int r,int g,int b)
        {
            IRgbColor pColor;
            pColor = new RgbColorClass();
            pColor.Red = r;
            pColor.Green = g;
            pColor.Blue = b;
            return pColor;
        }

    }
}
```

不同几何类型的元素使用的 Symbol 和 Element 方法是不一样的，这里是添加圆形元素的方法：

NewEnvelopeFeedback 的接口 INewEnvelopeFeedback 定义了产生一个矩形包络线的方法和属性，读者注意一下其中的 Constraint 属性，它是用于限制鼠标交互产生的矩形包络线形状的，如：

- esriEnvelopeConstraintsNone 矩形形状不作限制。
- esriEnvelopeConstralntsSquare 返回的矩形是一个正方形。
- esriEnvelopeConstraintsAspect 返回的矩形对象的长宽成一定比例，这个比例由 InewEnvelopeFeedback::AspectRation 属性设置。如果为 3，则宽与长的比率为 3。

6.5.2 移动几何形体对象上的节点

上一节讨论了 5 种产生新几何形体对象的 DisplayFeedback 对象，这一节则关心如何移动一个几何形体对象上的节点。不管用户编辑的是要素还是图形元素，其实都是在改变它们的 Geometry 属性而已，而改变这些形体，除了直接移动整个图形外，还包括移动形体上的某个节点对象。

首先本节要讨论的是 Polyline、Polygon 以及贝塞尔曲线这三个对象，因为这些几何形体对象上有节点（Vertex）存在。对 LineMovePointFeedback、BezierMovePointFeedback 和 PolygonMovePointFeedback 三个对象而言，它们移动节点的方法是一样的，且使用的鼠标事件也都是 MouseDown、MouseMove 和 MouseUp 三种。LineMovePointFeedback、BezlerMovePointFeedback 类都实现了 ILineMovePointFeedback 接口，因而可以使用一个公共变量来定义两种对象。

ILineMovePointFeedback 和 IPolygonMovePointFeedback 接口中定义的 Start 方法是需要深入讨论的，这个方法要求传入三个参数，即需要移动节点的几何形体对象，移动的点索引号和点击该几何形体对象的点。而程序员最需要寻找的就是 Polyline 或者 Polygon 上距离鼠标在视图上的点击点最近的节点的索引号。为了获得这个参数，系统需要使用 IHitTest 接口定义的方法来实现。

实现了 IHitTest 接口的对象有 Envelope、Point、Multipoint、Polyline 和 Polygon 五种几何形体对象，这个接口只有一个方法 IHitTest::HitTest，通过这个方法可以得到查询点查询范围内的节点索引号。

下面是 HitTest 方法的参数：

```
public bool HitTest (
        IPoint QueryPoint,
        double searchRadius,
        esriGeometryHitPartType geometryPart,
        IPoint hitPoint,
        ref double hitDistance,
        ref int hitPartIndex,
        ref int hitSegmentIndex,
        ref bool bRightSide

        );
```

这个方法中 QueryPoint 点是鼠标查询点，即鼠标在视图上的点击点；searchRadius 是查找半径，只有在这个范围内的几何形体对象上的点才能被查询到，需要用户设置；geometryPart 是选择类型，也需要用户自己进行确定；hitPoint 是被选择点的坐标；hitDistance 是两点之间的距离；HitPartIndex 是被选择到的部分的索引号；hitSegmentIndex 是被选择的片断（Segment）的索引号，bRightSide 是指明输入点是否为在几何形体对象的右侧。

这个方法的第 1、2、3、8 个参数需要明确设置，而第 4、5、6、7 个参数则由 HitTest 方法产生。

下面的代码显示了如何移动 Polyline 和贝塞尔曲线上的点：

MovePolylineAndBezierPoint.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using System.Windows.Forms;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    /// <summary>
    /// MovePolylineAndBezierPoint 的摘要说明。
    /// </summary>
    public class MovePolylineAndBezierPoint: BaseTool
    {
        IMap pMap;
        IActiveView pActiveView;
        IGraphicsContainer pGraphicsContainer;
        //pHitElement为取出的要素
        IElement pHitElement;
        //定义公共变量pLineMovePoint，它可以表示两种对象
        ILineMovePointFeedback pLineMovePoint ;

        IHookHelper m_hookHelper = new HookHelperClass();
        public MovePolylineAndBezierPoint()
        {
            base.m_caption = "CreateBezierCurveByFeedback";
            base.m_message = "CreateBezierCurveByFeedback";
            base.m_toolTip = "CreateBezierCurveByFeedback";
            base.m_category = "CreateGeometry";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
```

```
            m_hookHelper.Hook = hook;
        }
            public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            /* 在下面的代码中，如果点击鼠标左键，则产生一个LineMovePointFeedback对象；如果是点击右键，则产生一个BezierMovePointFeedback，这种使用父接口定义对象，而对象实现时却有不同的状态的例子在本书中出现多次。用户可以再次领略到这种“统一描述，分别实现”的好处，它只需要一个公共变量，便于管理，而且代码也更简洁。这种机制也是接口继承的一个重要特点。 */
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pGraphicsContainer = pMap as IGraphicsContainer;

            IPolyline pPolyline;
            IHitTest pHitTest;
            bool BoolHitTest;
            IPoint pPtHit = new PointClass();
            double DblHitDis = 2;
            int LngPrtIdx = 1 ;
            int LngSegIdx = 1;
            bool BoolHitRt = true;
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X, Y);

            //dist是点选几何形体对象时候的容差
            double dist;
            dist = pActiveView.Extent.Width/100;
            //使用函数返回第一个点选到的图形元素
            pHitElement = getElement(pPt, dist);
            if (pHitElement != null)
            {
                pPolyline = pHitElement.Geometry as IPolyline;  //获得这个元素的几何属性
                pHitTest = pPolyline as IHitTest;
                //下面的方法用于取出要移动的点的索引号
                BoolHitTest = pHitTest.HitTest(pPt, dist, esriGeometryHitPartType.esriGeometryPartVertex, pPtHit, ref DblHitDis, ref LngPrtIdx, ref LngSegIdx, ref BoolHitRt);
```

```
                if (BoolHitTest)
                {
                    if (Button == 1)
                    {
                        //有击鼠标左键
                        pLineMovePoint = new LineMovePointFeedbackClass();
                    }
                    else if (Button == 2)
                    {
                        //点击右键
                        pLineMovePoint = new BezierMovePointFeedbackClass();
                    }
                    //设置pLineMovePoint的DisPlay属性
                    pLineMovePoint.Display = pActiveView.ScreenDisplay;
                    pLineMovePoint.Start(pPolyline, LngSegIdx, pPt);
                }
            }
        }

        //当获得了要移动的几何形体对象的节点号后，在鼠标移动事件中，用户就可以将这个节点移到新的位置
        public override void OnMouseMove(int Button, int Shift, int X, int Y)
        {
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X, Y);
            //MoveTo方法继承自IDisplayFeedback接口的定义
            if (pLineMovePoint != null)   pLineMovePoint.MoveTo(pPt);
        }

        public override void OnMouseUp(int Button, int Shift, int X, int Y)
        {
            IGeometry pGeom;
            if (pLineMovePoint != null)
            {
                pGeom = pLineMovePoint.Stop();
                if (pGeom != null)
                {
                    pHitElement.Geometry = pGeom;
```

```
                    //更新元素
                    pGraphicsContainer.UpdateElement(pHitElement);
                }
                pLineMovePoint = null;
                pHitElement = null;
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null,
null);
            }
        }

        private IElement getElement(IPoint pPt, double dist)
        { //它用于使用鼠标在视图上获得一个图形元素对象并返回：
            IEnumElement pEnumElement;
            IElement pEle ;
            //找出所有被选择的元素
            pEnumElement = pGraphicsContainer.LocateElements(pPt, dist);
            if (pEnumElement != null)
            {
                pEle = pEnumElement.Next();
                while (pEle != null )
                {
            //只用找出第一条Polyline类型的元素即可，贝塞尔曲线也是Polyline的一种
                    if (pEle.Geometry.GeometryType ==
esriGeometryType.esriGeometryPolyline)
                    {
                        return pEle;
                    }
                    pEle = pEnumElement.Next();
                }
            }
            return null;
        }
    }
}
```

6.5.3 移动整个对象

用户也常常需要移动整个几何对象的功能，比如把点、线、包络线（Envelope）或者多边形整体移动，这个任务要交给 MovePointFeedback、MoveLineFeedback、MovePolygonFeedback 和 MoveEnvelopeFeedback 四个对象来完成。它们移动整个几何形体对象而不改变几何形体对象的形状。

每一个这样的 Feedback 对象都有自己的接口，不过它们定义的方法都是相似的，Start 方法用于开始移动，Stop 方法用于停止移动。下面是这个例子的代码，它可以移动 Polygon 和贝塞尔曲线：

MoveEntireLine.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class MoveEntireLine: BaseTool
    {
        //类全局变量
        IMap pMap;
        IActiveView pActiveView;
        IGraphicsContainer pGraphicsContainer;
        IElement pHitElement;
        IDisplayFeedback pDisplayFeedback;
        IHookHelper m_hookHelper = new HookHelperClass();
        public MoveEntireLine ()
        {
            base.m_caption = "MoveEntireLine ";
            base.m_message = "MoveEntireLine ";
            base.m_toolTip = "MoveEntireLine ";
            base.m_category = "EditGeometry";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }
        public override void OnCreate(object hook)
```

```
        {
            m_hookHelper.Hook = hook;
        }

        //鼠标在控件上按下左键，通过一个点选择到几何形体对象：
        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pGraphicsContainer = pMap as IGraphicsContainer;
            //获得鼠标在控件上点击的位置，产生一个点对象
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
            double dist;
            dist = pActiveView.Extent.Width/100;
            pHitElement = getElement(pPt, dist);
            IGeometry pGeomEle;
            if (pHitElement != null)
            {
                pGeomEle = pHitElement.Geometry;
                if (pGeomEle.GeometryType == esriGeometryType.esriGeometryPolyline)
                {
                    //产生一个 MoveLineFeedback对象
                    pDisplayFeedback = new MoveLineFeedbackClass();
                    pDisplayFeedback.Display = pActiveView.ScreenDisplay;
                    IMoveLineFeedback pMoveLineFeed;
                    pMoveLineFeed = pDisplayFeedback as IMoveLineFeedback;
                    //选择地理对象，在点选处开始移动
                    pMoveLineFeed.Start(pGeomEle as IPolyline, pPt);
                }
                //if ( pGeomEle.GeometryType == 其它几何类型 )
                //{
                    //.....
                //}
            }
        }

        public override void OnMouseMove(int Button, int Shift, int X, int Y)
```

```
            {
                IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
                pDisplayFeedback.MoveTo(pPt);
            }
            //移动到指定地方后，鼠标按键放开，产生 MouseUp 事件
            public override void OnMouseUp(int Button, int Shift, int X, int Y)
            {
                IGeometry pGeom;
                IGeometry pGeomEles ;
                if (pHitElement != null )
                {
                    pGeomEles = pHitElement.Geometry;
                    if (pGeomEles.GeometryType ==
esriGeometryType.esriGeometryPolyline)
                    {
                        IMoveLineFeedback pMvLnFeed;
                        pMvLnFeed = pDisplayFeedback as IMoveLineFeedback;
                        pGeom = pMvLnFeed.Stop();
                        //将被选择的元素pHitElement的Geometry换为移动后的新几何形体对
象，并更新这个元素
                        pHitElement.Geometry = pGeom;
                        pGraphicsContainer.UpdateElement(pHitElement);
                        pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics,
null, null);
                    }
                    //if ( pGeomEle.GeometryType == 其它几何类型 )
                    //{
                        //.....
                    //}
                }
            }
            //getElement(IPoint pPt, double dist)与上节例子一样，这里不再写出。
        }
}
```

6.5.4 其他　DisplayFeedback 介绍

以上章节介绍了几种常用的 DisplayFeedback，读者可以看出其实这些对象的鼠标事件和使用的属性、方法是差不多的。由于篇幅所限，本书不能给读者介绍所有的 DisplayFeedback 对象，因此在这一节中将有选择地介绍几种常用的 DisplayFeedback 对象，至于没有介绍的部分，读者可以参看 ESRI 提供的帮助资料。

- **MoveGeometryFeedback**

如果打算同时移动多个几何形体对象，可以使用 MoveGeometryFeedback 来实现。IMoveGeometryFeedback::AddGeometry 方法用于添加要移动的对象；MoveGeometryFeedback::ClearGeometry 可以清除 DisplayFeedback 中所有要移动的几何行体对象。Start 方法用于设置移动的起始点。

- **ReshapeFeedback 对象**

ReshapeFeedback 对象可以用于对一个实现了 IPath 接口的对象进行整形的任务，实现 IPath 接口的对象包括 Path 和 Ring。它可以对 Polyline 和 Polygon 整形。IReshapeFeedback::Start 的 Stretch 参数则决定了变形的方式。

- **ResizeEnvelopeFeedback 对象**

ResizeEnvelopeFeedback 对象用于改变一个 Envelope 对象的尺寸外观，读者可以使用 ResizeEdge 属性来确定是移动 Envelop 的边或角。Constraints 属性可以确定整形时的对象尺寸比例，如整形为一个正方行或者整形过程中 Envelope 的宽度和高度保持一定的比例。

- **StretchLineFeedback 对象**

如果希望能够旋转或者伸缩一个已经存在的 polyline 对象，则可以使用 StretchLineFeedack 来完成这个任务。

IStretchLineFeedback 接口定义了一个属性 Anchor，它用于确定旋转过程中的固定点，这个点一般设置为 Polyine 的起始点或者终止点。如果没有设置，则系统默认使用起始点作为这个锚点，Anchor 属性需要在 IStretchLineFeedback::Start 方法使用后进行设置。

下面给出旋转 Polyline 对象的一个实际例子。

RotatePolylineByFeedback.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
```

```
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    /// <summary>
    /// RotatePolyline 的摘要说明。
    /// </summary>
    public class RotatePolyline : BaseTool
    {
        IMap pMap;
        IActiveView pActiveView;
        IGraphicsContainer pGraphicsContainer;
        IStretchLineFeedback pDisplayFeedback;
        IElement pHitElement;

        IHookHelper m_hookHelper = new HookHelperClass();

        public RotatePolyline()
        {
            base.m_caption = "RotatePolyline";
            base.m_message = "RotatePolyline";
            base.m_toolTip = "RotatePolyline";
            base.m_category = "EditGeometry";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));

        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
```

```
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            pGraphicsContainer = pMap as IGraphicsContainer;

            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
            IPolyline pPolyline;
            pHitElement = GetHitElement(pPt, 2);
            //如果可以获得几何形体为Polyline的元素，则进行下面的处理
            if (pHitElement != null)
            {
                pPolyline = pHitElement.Geometry as IPolyline;
                pDisplayFeedback = new StretchLineFeedbackClass();
                pDisplayFeedback.Display = pActiveView.ScreenDisplay;
                pDisplayFeedback.Start(pPolyline, pPt);
                //设置转动的锚点为Polyline的起始点
                pDisplayFeedback.Anchor = pPolyline.FromPoint;
            }
        }

        public override void OnMouseMove(int Button, int Shift, int X, int Y)
        {
            IPoint pPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X,
Y);
            pDisplayFeedback.MoveTo(pPt);
        }

        public override void OnMouseUp(int Button, int Shift, int X, int Y)
        {
            IGeometry pGeom;
            if ( pDisplayFeedback != null )
            {
                //改变位置后，将得到的geometry传给元素更新
                pGeom = pDisplayFeedback.Stop();
                if ( pGeom != null )
                {
                    pHitElement.Geometry = pGeom;
                    pGraphicsContainer.UpdateElement(pHitElement);
```

```
                }
                pDisplayFeedback = null;
                pHitElement = null;
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null,
null);
            }
        }

        private IElement GetHitElement(IPoint pPt, double dist)
        {
            //使用一个点对象和一个搜索范围来获取元素
            IEnumElement pEnumElement;
            IElement pElemCur;
            //LocateElements将返回多个元素
            pEnumElement = pGraphicsContainer.LocateElements(pPt, dist);
            //如果返回的元素非空，则函数返回第一个几何形体是Polyline的元素
            if( pEnumElement != null )
            {
                pElemCur = pEnumElement.Next();
                while ( pElemCur != null )
                {
                    if (pElemCur.Geometry.GeometryType ==
esriGeometryType.esriGeometryPolyline)
                    {
                        return pElemCur;
                    }
                    pElemCur = pEnumElement.Next();
                }
            }
            return null;
        }
    }
}
```

执行这段代码，可以非常方便地将一个 Geometry 类型为 Polyline 的图形元素以起始点为锚点进行旋转和缩放。

GIS 作为一个信息系统，最大的特殊之处在于它需要显示图形信息，图形信息就是 ArcObjects 中的 Geometry 对象，为了产生和显示几何形体对象，ArcObjects

提供了一套复杂而基本的对象库，这些对象专门用于修饰 Geometry 对象，如颜色对象、符号对象。

本章也使用较大的篇幅详细介绍了如何产生一幅专题图，专题图可以揭示一个图层中不同要素的差别和联系，它是产生一幅成品地图必须进行的操作。为了揭示更多图形无法显示的信息，地图必须进行标注，而本章也给读者介绍了两种基于要素的地图标注方法。

本章最后讨论了使用鼠标和视图上的图形对象进行交互操作的 DisplayFeedback 对象，这些对象可以完成诸如产生 Geometry，移动 Geometry 等高级操作。这一章的对象众多且复杂，读者需要进行认真的阅读和实践。

7 地图的输出

地图输出分为两种类型：一种是地图的打印输出，即调用与计算机连接的打印设备将地图打印在纸质媒介上；另一种是地图的转换输出，即将地理数据输出为不同格式的文件，如 JPEG、PDF、BMP 等。

本章将介绍地图的打印和输出两方面的内容，它包括如何传递数据到一个硬拷贝设备，如打印机；或将地图输出为一种特定格式的数据文件。在 ArcObjects 中这两种不同的过程使用了不同的输出对象集来完成，前者使用的是 Printer 对象，后者使用的是 Exporter 对象。

学习完本章后，读者应该掌握以下的知识：

- 如何调用 IPrinter 对象打印地图
- 如何将地图输出为矢量和栅格类型的文件
- 使用输出对话框产生矢量和栅格类型文件

7.1 地图的打印输出

熟悉 Windows 的程序员都知道，在 Windows 平台下，打印是一种与设备无关的活动，Windows 以逻辑设备的方式来管理打印机。每个打印机都对应着一种由特定打印机厂家提供的驱动程序，操作系统通过一个打印机硬件抽象层来向上隐藏具体的打印机特性。在应用程序看来，所有的打印机都是一样的，即 Windows 提供了与具体打印设备无关的图形接口。用户只需要调用这些公共接口编写需要的自己的打印程序，而文件如何打印出来，则由 Windows 自己调用相应的驱动程序去处理完成。这样编写的打印代码就不必依赖于某种具体的打印设备，不用考虑用户的打印机的型号。

7.1.1 Printer 类

ArcObjects 中对于地图的打印使用的是 Printer 类，这是一个抽象类。有三个打印类继承自这个抽象类：EmfPrinter、ArcPressPrinter 和 PsPrinter，这三种类的对象都分别支持各自类型的硬拷贝设备，但是它们有不同的方法来实现打印功能。

因此，程序员需要选择的打印对象并不是随便决定的，它取决于程序使用的打

印设备类型和驱动程序的类型。EmfPrinter 对象是通过使用 EMF 数据格式来作为打印驱动的；PsPrinter 对象则使用 PostScript 作为驱动来打印输出地图；而 ArcPressPrinter 则是使用 ArcPressPrinterDriver 来输出地图。

IPrinter 接口定义了所有打印对象的一般方法和属性。如 IPrinter::Paper 属性用于初始化与系统关联的缺省的打印机，它是一个 Paper 对象，用户可以产生一个独立的 Paper 对象去使用另一台打印机。

IPrinter::PrintToFile 属性可以设置是否输出到一个文件；IPrinter::StartPrinting 方法用于返回一个打印设备的 hDC，以便 IActiveView::Output 对象可以使用它将地图输出到一个打印机设备上；IPrinter::FinishPrinting 则用于清除打印后的缓存对象。

- **EmfPrinter 对象**

EMF 即是所谓 Enhanced Windows Metalfile，EmfPrinter 是使用 Windows Enhanced Metafile 格式作为打印驱动的打印对象，它支持 IEmfPrinter 接口，不过这个接口没有任何的属性和方法，它仅仅是指明用户是否使用这个对象来进行打印工作。

- **ArcPressPrinter 对象**

ArcPressPrinter 是一种使用 ESRI ArcPress 打印驱动 ArcPressPrinterDriver 的打印对象。ArcPress 是 ArcView，ArcEditor 和 ArcInfo 程序中的一个可选的地图打印扩展模块。作为 ESRI 的栅格影像处理器（RIP），ArcPress 将标准的图形交换格式和内置的打印机语言表达成打印文件传递到工业标准的宽幅格式桌面打印机上。

由于大的 GIS 地图可能包括大量的矢量数据、复杂的符号和极大的影像，这使得在桌面打印机上打印地图是非常困难和耗时的。ArcPress 扩展模块的角色是在打印机上迅速地打印高质量的地图，并且不需要额外的内存或硬件。ArcPress 可以在不需要昂贵的硬件升级的情况下，将一台计算机变为一台打印处理器，使之能连续地打印。

注意：不要试图在没有安装 ArcPress 软件模块的情况下使用这个对象。这些扩展模块需要向 ESRI 单独购买。

- **PsPrinter 对象**

PsPrinter 对象是使用 PostScript 作为驱动的打印对象。

PostScript 是一种与设备无关的打印机语言，即在定义图像时可以根本不考虑输出设备的特性（如打印机的分辨率、纸张大小等），而且它对文本和图形实行同样的处理过程，这给字体处理带来了极高的灵活性。

由于 PostScript 的设备无关特性，在输出到特定输出设备，譬如对分辨率、纸

张大小、进纸盒进行选择时，PostscriPt 通过打印机描述（PostScript Printer Description，PPD）文件来实行各种打印机的不同特性。PPD 文件主要提供以下与打印机有关的特定信息：默认/最高分辨率、是否支持半色调监控、用户设定的监控信息、页面大小定义、页面可打印区域、默认字体（通常为 Courier）和是否支持双面打印等。每一种不同的 PostScript 打印机分别对应有专门的 PPD 文件。通过 PS 驱动程序，各种不同的应用软件将各自的数据转换成 PostScript 格式。

PS 驱动对象在 ArcObjects 中称为 PsDriver，它用于设置 PostScript 的属性，包括颜色信息等，PsDriver 对象由 SpotPlate 对象组成，后者可以产生基于单独某种颜色的图形信息。

PsPrinter 对象使用 FontMapEnvironment 对象来设置打印过程中的字体信息。

FontMapCollection 是一个集合对象，它的生命周期是由产生它的对象控制的，如 PsPrinter 或者 PsExPorter。这个对象控制着被 PostScript 驱动使用的 FontMap 对象，用户可以通过 IFontMapEnvironment::FontMapCollection 来获得这个对象。缺省情况下这个集合中存在一些被设置为默认值的 FontMap，这些值是由系统确定的，当然程序员也可以自己添加 FontMap 对象到集合中。

FontMap 对象是由 FontMapCollection 对象来掌握控制的，它维系着系统的 TrueType 字体和地图字体之间的映射联系。

7.1.2 Paper 对象

Paper 对象是 Printer 对象要求的一个关键属性对象，它的属性和方法用于维持 Printer 对象使用的打印机和打印纸张之间的联系。

当应用程序启动的时候，一个 Paper 对象就自动产生了，这个创建是基于系统的缺省打印机的。

如果用户需要使用另一台打印机，就需要产生一个新的 Paper 对象，并且将它的 PrinterName 设置为打印机名。Paper 对象可以通过 IPrinter::Paper 属性和一个打印机对象相关联。

下面的代码是产生一个 Paper 对象并将它关联到一个打印设备上：

```
IPaper pPaper;
IPrinter pPrinter;
IPsPrinter pPsPrinter ;
//SCUT＼Scut为打印机名
pPaper = new PaperClass();
pPaper.PrinterName = @"\\SCUT\Scut";
```

```
            pPsPrinter = new PsPrinterClass();
            pPrinter = pPsPrinter as IPrinter;
            pPrinter.Paper = pPaper;
```

IPaper 接口可以在 Paper 对象和打印设备间产生关联，一旦关联后，用户可以通过这个接口获得或者设置 Paper 属性。

IPaper::Orientation 属性可以用于获取和设置打印的方向，它是一个整型值，当它值为 1 时为纵向方向，值为 2 时为横向方向；IPaper::QueryPapersize 方法则用于获得页面的尺寸。

下面是一个在 PageLayoutControl 控件中获得系统默认打印设备属性的例子：

```
    private void GetPrinter()
    {
        IPrinter pPrinter;
        IPaper pPaper;
        double dWidth;
        double dheight;
        if ( axPageLayoutControl1.Printer != null)
        {
            //通过IPrinter接口获得控件的打印机对象
            pPrinter = axPageLayoutControl1.Printer;
            pPaper = pPrinter.Paper;
            //确定打印机纸张的方向
            if ( pPaper.Orientation ==1 )
                lblPrinterOrientation.Text = "纵向方向";
            else
                lblPrinterOrientation.Text = "横向方向";
            //获得打印机名
            lblPrinterName.Text = pPaper.PrinterName;
            //确定打印机页面的尺寸
            pPaper.QueryPaperSize(out dWidth, out dheight);
            lblPrintersize.Text = dWidth.ToString("###.00") + " by" +
dheight.ToString("###.00") + "Inches";
        }
    }
```

这个过程执行后，用户可以在 lblPrinterOrientation、lblPrinterName 和 lblPrintersize 三个控件中看到 PageLayoutControl 控件默认的打印机的属性。

7.1.3 在控件中打印输出

IPageLayoutControl::PrintPageLayout 方法可以用于打印控件中的布局视图，用户需要注意的是由于打印的部分与控件的 Page 对象息息相关，所以用户有必要在地图打印前对 Page 对象进行设置。如果控件中的页面 Page 无法与打印对象的 Paper 相匹配，可以使用 IPage::PageToPrinterMapping 来决定是否让页面和打印文本相适配，如果两个对象的页面尺寸不一致，是伸缩地图还是切割地图等。

使用 PageLayoutControl 控件打印的例子如下：

```
private void PrintPageLayout()
{
    IPrinter pPrinter;
    pPrinter = axPageLayoutControl1.Printer;
    IPaper pPaper;
    pPaper = pPrinter.Paper;
    IPage pPage;
    pPage = axPageLayoutControl1.Page;
    //设置页面的尺寸，为A4
    pPage.FormID = esriPageFormID.esriPageFormA4;
    //设置打印方向
    pPage.Orientation = 1;
    //设置匹配值
    pPage.PageToPrinterMapping =
esriPageToPrinterMapping.esriPageMappingScale;
    if (pPaper.Orientation != pPage.Orientation) pPaper.Orientation =
pPage.Orientation;
    axPageLayoutControl1.PrintPageLayout(1, 1, 0);

}
```

PrintPageLayout 方法有三个参数，第一个是 StartPage，起始页；EndPage，中止页；OverLap 是页与页之间的重叠范围。上面的命令是打印一张地图，文件数是一份。

打印也是一个可能出现异常的过程，下面的参数可以用于用户检查异常：

1001 800A03E9：控件没有打印对象可以使用

1002 800A03EA：起始页面超出了范围

1003 800A03EB：中止页面超出了范围

7.2 地图的转换输出

有些时候地图并不需要被打印出来，第三方用户可能更希望得到保存为图片格式或 PDF 格式的文件。ArcObjects 也能够满足这些需求，它使用 Export 对象集来完成这类任务。需要提醒读者注意的是由于版本的变化，ArcObjects 的转换输出对象已经从 Exporter 变化为 Export 了，IExport 接口替代了以前的 IExporter 接口，ESRI 希望用户能够使用最新版本的转换输出对象。

相对于将地图打印的所谓硬拷贝而言，将它转换为一种格式文件的过程称为软拷贝。在新版本的 ArcObjects 中，可以转化的文件格式分为两大类，一类是基于栅格的 ExportImage，另一类是基于矢量的 ExportVector。有些以前版本可以支持的转换格式现在已经没有了，如 CGM 等。

Export 类是所有转换输出类的父类，它是一个抽象类，实现了 IExport 接口用于定义转换输出对象的一般方法和属性，如 FileName 和 Resolution（分辨率）和 Filter 等。

Windows 平台一般的屏幕分辨率使用的是 96dpi，这个值也是 ArcGIS 影像文件格式输出的缺省分辨率。而矢量格式转换输出，如 PDF、AI 等，它们的缺省分辨率是 300dpi。分辨率越高，意味着图片越清晰，但随之而来的缺点是文件就越大。

FileExtension、Filter 和 Name 属性是基于用户要使用的转换输出对象的。例如要产生一个 PDF 文件，文件的扩展名 FileExtension 就是“.Pdf”。PixelBounds 用于确定输出文件的输出范围，它是一个矩形区域。

7.2.1 基于影像格式的输出

ExportImage 类及其子类的对象都是用于将地图输出为栅格格式文件的对象。ExportImage 是 Export 的一个子类，它本身也是一个抽象类，实现了 IExportImage 接口，这个接口定义了所有操作基于栅格格式文件的一般方法和属性。IExportImage 接口定义的属性有 BackgroundColor，用于确定文件的背景颜色 ；width 和 height 用于确定文件尺寸；通过 ImageType 则可以设置或者获取图片的类型信息。

ExportImage 类还实现了 IWorldFilesettings 接口，通过这个接口可以设置或查询 MapExtent 和 OutputworldFile 属性。

ExportImage 的子类有五种：ExportBMP、ExportJPEG、ExportPNG、ExportTIFF 和 ExportGIF。不同的转换输出对象的操作方法基本上是一致的，用户通过它们可以分别将地图生成 BMP、JPEG、PNG、TIFF 和 GIF 格式的图形文件。

- **ExportBMP 对象**

BMP 是一种典型的栅格图形，它由一个个像素点组成，其特点是信息量非常丰富，但文件太大。EXportBMP 对象可以将一个布局视图的内容转换为一个 BMP 格式的图片文件。

ExportBMP 类实现了 IExportBMP 接口，可以使用这个接口定义的属性来设置要产生的 BMP 文件。如 IExportBMP::Platte 用于获取位图的颜色调色板；IExportBMP::Bitmap 用于获取位图的句柄；IExportBMP::RLECompression 用于设定位图的压缩的方案。

下面是一个产生 BMP 格式图形文件的代码例子：

```
private void ExportBMP()
{
    IActiveView pActiveView;
    IExport pExport ;
    IEnvelope pPixelBoundsEnv;
    int iOutputResolution;
    int iScreenResolution;
    int hDC;
    pActiveView = axPageLayoutControl1.ActiveView;
    pExport = new ExportBMPClass();
    pExport.ExportFileName = @"d:\temp\ExportTest." +
Microsoft.VisualBasic.Strings.Right(pExport.Filter, 3);
    //由于输出分辨率不同于屏幕分辨率，所以需要设置两个值来用于尺寸计算
    iScreenResolution = 96;
    iOutputResolution = 300;
    pExport.Resolution = iOutputResolution;
    //ExportFrame属性可以给出一个使用屏幕分辨率的合适的尺寸
    tagRECT pExportFrame ;
    pExportFrame = pActiveView.ExportFrame;
    tagRECT exportRECT;
    exportRECT.left = 0;
    exportRECT.top = 0;
    exportRECT.right = pActiveView.ExportFrame.right * (iOutputResolution /
iScreenResolution);
    exportRECT.bottom = pActiveView.ExportFrame.bottom * (iOutputResolution /
iScreenResolution);
    //使用PixelBoundsEnv来设置输出范围
```

```
                pPixelBoundsEnv = new EnvelopeClass();
                pPixelBoundsEnv.PutCoords(exportRECT.left, exportRECT.top,
exportRECT.right, exportRECT.bottom);
                pExport.PixelBounds = pPixelBoundsEnv;
                hDC = pExport.StartExporting();
                pActiveView.Output(hDC, (int)pExport.Resolution, ref exportRECT, null, null);
                pExport.FinishExporting();
                pExport.Cleanup();

        }
```

执行这个过程后，可以在 d 盘\temp 中找到一个 ExportTest.bmp 的图片文件。PageLayoutControl 控件视图中的地图图像会出现在这张图片上。

- **ExportJPEG 对象**

JPEG 图片以 24 位颜色存储单个光栅图像，它是一种与平台无关的图形格式，支持最高级别的压缩，但是这种压缩是有损耗的。

用户可以提高或降低 JPEG 文件压缩的级别，但是文件大小是以图像质量为代价的。JPEG 的压缩比率可以高达 100:1（JPEG 格式可在 10:1 到 20:1 的比率下轻松地压缩文件，而 JPEG 图片的质量不会下降）。JPEG 压缩可以很好地处理写实摄影作品。

使用 ExportJPEG 对象可以将布局视图中的地图转换输出为一张 JPEG 格式的图像。IExportJPEG 接口定义的 Quality 属性用于设置图片的压缩比，它的缺省值是 100 的无损最高质量。

产生一张 JPEG 图片的代码很简单，将上例中的“pExport＝new ExportBMPClass();”改为”pExport＝new ExportJPEGClass();”就可以了。

- **ExportGIF 对象**

使用 ExportGIF 对象可以将地图转换输出 GIF 格式的文件。GIF 是一种标准的栅格数据文件，通常使用在网络上，这种文件包含的颜色不能包含超过 256 种，这使得它比同样内容的其他文件格式体积小得多。

IExportGIF 接口定义的属性可以用于设置一个转换输出的 GIF 文件，IExportGIF::TransparentColor 可以用于设置图片的透明度。

- **ExportPNG 对象**

PNG 格式文件是 20 世纪 90 年代中期开发的图像文件存储格式，它出现的目的是意图替代 GIF 和 TIFF 文件格式，同时增加一些 GIF 文件格式所不具备的特性。

流式网络图形格式（Portable Network Graphic Format，PNG）名称来源于非官方的“PNG’s Not GIF”，是一种位图文件（bitmap file）存储格式，读成“ping”。

PNG 用来存储灰度图像时，灰度图像的深度可多到 16 位，存储彩色图像时，彩色图像的深度可多到 48 位，并且还可存储多到 16 位的 α 通道数据。PNG 是采用无损算法，这使得它在图像编辑软件中传播时能够保证其良好的信息质量。

IExportPNG 接口可用于管理输出的 PNG 文件，如 IExportPNG::TransparentColor 可以设置某种颜色的透明度。同样地，产生一个 PNG 格式数据的代码也仅仅只需要将上例的 “pExport＝new ExportBMPClass();”改为”pExport＝new ExportPNGClass();”就可以了。

- **ExportTIFF 对象**

TIFF 是一种比较灵活的图像格式，它的全称是 Tagged image File Format，文件扩展名为 TIF 或 TIFF。该格式支持 256 色、24 位真彩色、32 位色、48 位色等多种色彩位，同时支持 RGB、CMYK 以及 YCbCr 等多种色彩模式，支持多平台。TIFF 文件可以是不压缩的，文件体积较大，也可以是压缩的，它支持 RAW、RLE、LZW、JPEG、CCITT3 组和 4 组等多种压缩方式。

TIFF 格式是最常用的一种存储格式，此格式是以多种分辨率灰级和颜色存储图像的常用方法，这种格式可支持跨平台的应用软件。

ExportTIFF 对象可以将版式地图保存为 TIFF 格式文件，IExportTIFF 接口定义了产生这种文件的属性。如 IExportTIFF::CompressionType 是指 TIFF 文件的压缩格式，它是一个 esriTIFFComPression 枚举值，用户可以设置诸如不压缩、JPEG 压缩、LZW 压缩等。

IExportTIFF::GeoTiff 是一个选项，用于选择是否设置 GeoTiff 标签属性。

7.2.2 基于矢量格式的输出

基于矢量格式文件的转换输出是 ExportVector 类及其五个子类对象的任务。ExportVector 是一个抽象类，它实现了多个接口用于设置这些矢量转换输出对象的一般方法和属性。IExportVector 接口是空的，没有任何的属性和方法；IExportVectorColorSpaceSettings 接口的 ColorSpace 可以设置输出文件的颜色空间等。

ExportVector 下面的子类有 ExportEMF、ExportAI、ExportPDF、ExportPS 和 ExportSVG 五种，它们分别用于生成五种不同格式的矢量数据文件。

- **ExportEMF 对象**

EMF（Windows Enhanced Metafile）“增强型图元文件”是 32 位格式，可以同时包含矢量信息和位图信息。此格式是对“Windows 图元文件格式”的改进，包含了一些扩展功能，如内置的缩放比例信息，与文件一起保存的内置说明，以及调色板和设备独立性方面的改进。

EMF 格式是可扩展的格式，这意味着程序员可以修改原始规范以添加功能或满足特定的需要。此修改可能会导致不同类型的 EMF 图片之间不兼容。

使用 ExportEMF 对象可以非常方便地将布局视图上的地图转换为一个 EMF 文件，下面是一个实例代码：

```
private void ExportEMF()
{
    IActiveView pActiveView;
    pActiveView = axPageLayoutControl1.ActiveView;
    //使用ExportEMF对象来产生一个EMF文件
    IExport pExport;
    pExport = new ExportEMFClass();
    pExport.ExportFileName = @"d:\temp\ExportTest1.emf";
    //设置分辨率为300
    pExport.Resolution = 300;
    //对于Map和PageLayout，ExportFrame属性总是一个tagRECT结构，用于获取合适的输出尺寸
    tagRECT exportRECT;
    exportRECT = pActiveView.ExportFrame;
    //产生一个新的包络线对象
    IEnvelope pPixelBoundsEnv;
    pPixelBoundsEnv = new EnvelopeClass();
    pPixelBoundsEnv.PutCoords(exportRECT.left, exportRECT.top, exportRECT.right, exportRECT.bottom);
    pExport.PixelBounds = pPixelBoundsEnv;
    int hDC;
    hDC = pExport.StartExporting();
    //输出图片
    pActiveView.Output(hDC, (int)pExport.Resolution, ref exportRECT, null, null);
    //完成输出后立刻清除临时文件
    pExport.FinishExporting();
    pExport.Cleanup();
}
```

- **ExportAI 对象**

AI（Adobe IIllustrator）格式是一种使用 Adobe IIllustrator 软件产生的高质量格式文件。它通常是使用于印刷领域的交换格式。

在 ArcObjects 中使用 ExportAI 对象可以生成 AI 格式的文件，IExportAI 接口用于设置输出文件的属性，但是其中唯一的属性 EmbedFonts 在目前的版本中还不可以使用。

- **ExportPDF 对象**

Adobe 可移植文档格式（PDF）是全世界电子版文档分发的公开实用标准。Adobe PDF 是一种通用文件格式，能够保存任何源文档的所有字体、格式、颜色和图形，而不管创建该文档所使用的应用程序和平台。Adobe PDF 文件为压缩文件，任何人都可以使用免费的 Adobe Acrobat Reader 共享、查看。测览和打印。

IExportPDF 是 ExportPDF 对象实现的设置转换输出的 PDF 文件属性的接口。IExportPDF::Compressed 属性用于设置 PDF 文件是否压缩，
IExportPDF::EmbedFonts 是文件内嵌入字体的选项，
IExportPDF::ImageCompression 属性则可以设置文件中图片的压缩。

下面的代码将一个 PageLayoutControl 控件中的地图转换为 PDF 文件：

```
private void ExportPDF()
    {
        IActiveView pActiveView;
        pActiveView = axPageLayoutControl1.ActiveView;
        IEnvelope pEnv;
        pEnv = pActiveView.Extent;
        //使用ExportPDF对象来产生一个PDF文件
        IExport pExport;
        pExport = new ExportPDFClass();
        pExport.ExportFileName = @"d:\temp\ExportTest1." +
Microsoft.VisualBasic.Strings.Right(pExport.Filter, 3);
      //设置分辨率
        pExport.Resolution =
pActiveView.ScreenDisplay.DisplayTransformation.Resolution;
        //确定exportRECT的尺寸
        tagRECT exportRECT;
        exportRECT.top = 0;
        exportRECT.left = 0;
        exportRECT.right = (int)pEnv.Width;
```

```
            exportRECT.bottom = (int)pEnv.Height;
            //产生一个新的包络线对象
            IEnvelope pPixelBoundsEnv;
            pPixelBoundsEnv = new EnvelopeClass();
            pPixelBoundsEnv.PutCoords(exportRECT.left, exportRECT.bottom,
exportRECT.right, exportRECT.top);
            pExport.PixelBounds = pPixelBoundsEnv;
            int hDC ;
            hDC = pExport.StartExporting();
            //输出图片
            pActiveView.Output(hDC, (int)pExport.Resolution, ref exportRECT, null, null);
            //完成输出后立刻清除临时文件
            pExport.FinishExporting();
            pExport.Cleanup();

        }
```

- **ExportPS 对象**

使用 ExportPS 组件类可以将地图输出为 Encapsulated PostScript（EPS）格式。EPS 文件是使用 PostScript 页面描述语言来描述矢量和栅格对象的。它是出版工业的标准之一。EPS 文件也可以使用多种软件进行编辑。

使用 ExportPS 对象可以进行这种转换输出, IExportPS::Image 用于设置 PS 中的图片描述；IExportPS::ImageCompression 用于设置文件中的图片压缩属性等。

- **ExportSVG 对象**

SVG 是 Scalable Vector Graphics 的首字母缩写，含义是可缩放的矢量图形。它是一种开放标准的矢量图形语言，可以用于设计高分辨率的 Web 图形页面。SVG 被开发的目的是为 Web 提供非栅格的图像标准。

SVG 格式是基于 XML（eXtensible Markup Language），并且被 World Wide Web Consortiurn（W3C）联盟进行开发。它是一个开放的标准，SVG 不属于任何公司，它是工业界先驱们同心合作的产物。SVG 标准对任何想使用它们的公司或个人都是开放的，已经有很多公司的软件支持 SVG 的创建、编辑和测览。

ArcObjects 使用 ExportSVG 对象来让用户生成这种最新的文件格式。

7.2.3 ExportFileDialog 对象

前面两节介绍的两类共计 10 个地图转换输出对象可谓是功能强大，但是使用它们的时候并不是非常方便。本节主题将介绍一个 ArcObjects 封装好的对象 ExportFileDialog，使用它可以通过可视化对话框将地图保存为任何 ArcObjects 支持的格式，而且代码简洁清晰。

ExportFileDialog 对象取代了以前版本中的 ExportDialog 对象。使用这个对象需要引入 ESRI.ArcGIS.OutputUI 对象库。IExportFileDialog 接口用于管理这个对象的属性和方法，IExportFileDialog::DoModal 可以弹出对话框，如图 7-1 所示，而 IExportFileDialog::Export 则可以返回一个转换对象。

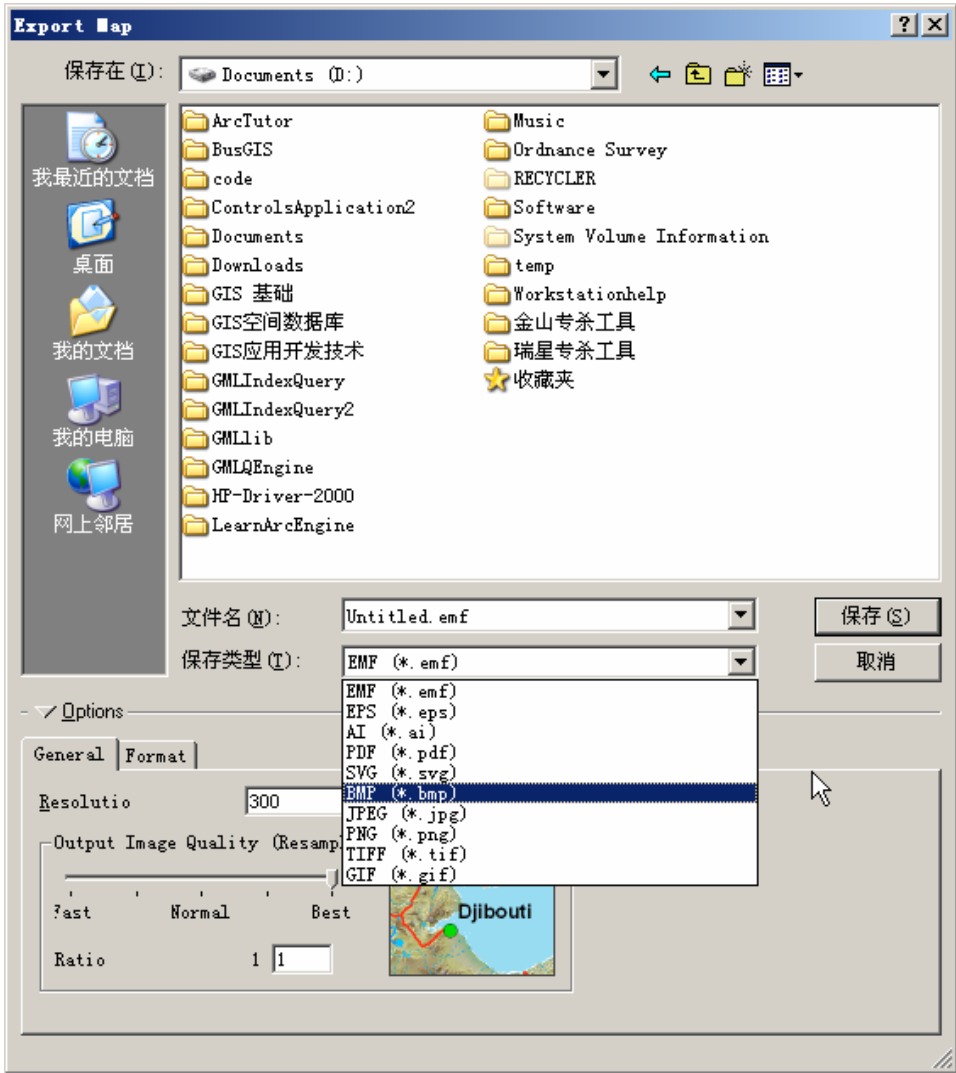


图 7-1 输出文件对话框

打开对话框后，可以看到这个对象能够将地图保存为 10 种格式的文件，在

“Options”选项中，项目还可以设置各种输出的属性，以 PDF 输出为例，如分辨率、图片质量、图片描述模式等。

下面是一个使用 ExportFileDialog 对象的例子。

```
private void ExportFileDialog()
{
    IActiveView pActiveView ;
    IEnvelope pPixelBoundsEnv ;
    int iOutputResolution;
    int iScreenResolution;
    int hDC ;
    pActiveView = axPageLayoutControl1.ActiveView;
    //设置默认分辨率
    iScreenResolution = 96;
    iOutputResolution = 300;
    tagRECT exportRECT ;
    exportRECT.left = 0;
    exportRECT.top = 0;
    exportRECT.right = pActiveView.ExportFrame.right * (iOutputResolution /
iScreenResolution);
    exportRECT.bottom = pActiveView.ExportFrame.bottom * (iOutputResolution /
iScreenResolution);
    //设置输出范围
    pPixelBoundsEnv = new EnvelopeClass();
    pPixelBoundsEnv.PutCoords(exportRECT.left, exportRECT.top,
exportRECT.right, exportRECT.bottom);
  //使用输出对话框对象
    IExportFileDialog pExportDialog;
    pExportDialog = new ExportFileDialogClass();
    bool pBool;
    pBool = pExportDialog.DoModal(pPixelBoundsEnv, pActiveView.Extent,
pActiveView.Extent, 300);
    //如果按下“取消”键，则退出过程
    if (!(pBool)) return;
  //设置转换输出对象的属性
    IExport pExport;
    pExport = pExportDialog.Export;
    pExport.Resolution = iOutputResolution;
```

```
            pExport.PixelBounds = pPixelBoundsEnv;
            hDC = pExport.StartExporting();
            //产生文件
            pActiveView.Output(hDC, (int)pExport.Resolution, ref exportRECT, null, null);
            pExport.FinishExporting();
            pExport.Cleanup();
        }
```

本章的内容介绍了 ArcObjects 中的打印输出和转换输出两大对象集，读者可以看出,ArcObjects 的输出功能是非常强大的，用户完全可以依据实际的需要选择不同格式输出文件或者打印地图。

需要再次提醒读者注意的是，由于转换输出对象集较之以前有了很大的变化，因此这部分的具体内容需要查看最新的 ESRI 开发帮助文档。

8 空间数据管理

空间数据管理在数据模型主要经历了早期的 CAD 模型、地理相关数据模型、对象-关系数据模型等几个发展阶段。

20 世纪 60 - 70 年代，计算机的一个重要应用就是计算机辅助设计，其模型即 CAD 数据模型用二进制文件存储地理数据，用点、线、面描述空间实体，几何要素与相关的颜色、形状等属性存放在一起，而图层和注记号则是它对属性的主要表达方式。空间数据不存贮在数据库中，并且缺乏对属性数据的支持是 CAD 数据模型给 GIS 软件开发和 GIS 数据共享带来的主要困难。

1981 年 ArcInfo 开始使用 Coverage 文件作为地理数据的存储介质，由于受到当时计算机软硬件技术的限制，人们不得不将数据存储分割开来，使用表来存储普通属性数据，使用索引化的二进制文件来存储空间信息数据，而两者通过一个公共标识符相联系。Coverage 文件在长期的使用中取得了巨大的成功，并可以通过 AML 等语言对其进行行为的扩展，但是它的数据存储方式本身有很大缺陷——空间数据无法很好地与其行为相对应，并且它的 多用户并发操作和海量 GIS 数据存储管理上无法满足实际需要。

近些年来，关系型数据库在理论和技术上都得到了很大的发展，通过扩展数据库的面向对象特性，成熟的商用数据库（如 Oracle、DB2、SQL Server 等）不但可以存储普通类型的数据，而且可以存储图片、声音文件和视频文件和空间数据。

这种扩展的一种方法是增加新的数据类型，即将复杂的数据类型抽象为一个二进制流。 这种方法使用二进制数据类型来存储复杂数据，但是并不能够建立索引来查询这种数据类型。扩展的另一种方式是将复杂数据类型作为对象放在关系数据库中，并且在管理和检索这些类型数据上提供了方法，这种扩展后的数据库称为对象—关系数据库，这也是目前较为流行的一种扩展方式，如 ESRI 的 ArcSDE 就是一个基于关系数据库的空间数据引擎，使用这个引擎可以方便地操作关系数据库，存储海量的地理数据。

地理数据库（GeoDatabase）是 ESRI 随着 ArcInfo 而推出的一个统一的数据模型、一种地理数据的管理机制和一套完整的地理数据开发组件。它是一种以紧密耦合特征和行为为基础的新型数据模型。在物理级别上，它有两种数据存储形式。一是使用大型关系数据库加上 ArcSDE（空间数据引擎），这种地理数据库可以方便地对海量数据进行松散存储，可以实现多用户并发操作、事务管理、数据库恢复和空间数据无缝管理等。由于关系数据库管理系统一般都可以运行在 Windows / Unix 平台上，因此一些大型 GIS 系统可以直接使用 Unix 服务器，而客户端则使用

Windows，这样增强了数据管理的安全性和稳定性，同时也使应用界面更方便。另一种是个人级别地理数据库，即 Personal GeoDatabase，使用 Access 数据库作为其数据存储介质。它和 SDE 数据库最大的差别在于它最大的数据容量为 2G，并且不能实现多用户并发操作等复杂应用。

在 ArcGIS9.2 中，ESRI 将使用一种全新的文件格式的 GeoDatabase，它能模拟由 DBMS 维护的 GeoDatabase 的全部信息模型。区别是这种格式只支持单用户编辑模型（类似于 Personal GeoDatabase）。这种文件的性能和 shape 文件类似，最大的容量限制在每个表格 1TB。基于文件的 GeoDatabase 可以多用户读取，但是只能有一个用户写 / 编辑（类似 coverage 文件）。ESRI 的最终目的是把这种格式作为 Personal GeoDatabase 和 shapefile 的替代者。

在逻辑级别上，GeoDatabase 模型采用了统一的框架处理地理数据。从这个模型的视角上看，除了在获得不同类型地理数据的操作上有所差别外，对已经获得的要素类、要素数据集、表、TIN（不规则三角形网络）、栅格数据等数据对象的处理方式都是一样的，这个模型大大地降低了理解和二次开发 GeoDatabase 的难度。

学习完本章后，读者应该能够对地理数据库有个基本的理解：

- 如何打开三种不同类型的工作空间
- 如何理解数据集对象
- 理解表，对象和要素类等对象
- 字段集及其相关对象
- 行、对象和要素的联系与差别
- 关系类
- 地理数据的转换

8.1 GeoDatabase 的基础知识

GeoDatabase 是第三代地理数据模型，这种模型相比过去的数据模型，最大的特点是 GeoDatabase 中的要素更加智能化，每个要素不再仅仅是一条有几何字段的记录，而是一个拥有属性和行为的对象。即它是一个基于面向对象模型的关系数据库，而不仅仅是一个普通关系数据库。

GeoDatabase 模型如同 Coverage 模型一样，支持要素间的拓扑关系，并且扩展了基于要素和其他面向对象类型数据的复杂网络和关系功能。使用 GeoDatabase

数据模型操作保存在数据库中的地理数据和操作 shape 文件和 Coverage 文件使用的是几乎一样的方式。实际上，由于 GeoDatabase 模型使用了一般性数据框架描述，从这个框架的视角上看，同类型的不同格式数据之间除了存储的物理文件有所不同外，在操作上使用的代码是没有什么差别的。

GeoDatabase 的每一种数据集都是对世界某一方面的空间描述，它包括：

- 基于矢量的离散要素（点、线和面）的有序集合
- 基于数字高程模型 DEM 和影像的栅格数据集合，描述连续的地理现象
- 模拟物流、人流等形式的拓扑网络
- 地球表面的三维模拟，如 TIN 等数据集
- 地址和定位器等其他数据集

GeoDatabase 实际上是一个在关系数据库基础上扩展的“面向对象数据库”，由于它将数据看成不同类型的对象而不是简单的一行数据，从而可以给这些对象更加准确的行为和关系。对于一个对象而言，它具备多态性、封装性和继承性三大基本特点。

对于对象的多态性，它意味着对象的行为可以适应对象的变化而改变。一个要素（Feature），无论它放在什么类型的数据源中--Shape 文件、Personal GeoDatabase 或 SDE 数据库，它的操作方式都是一样的。

对象的封装性表现为它是一个“黑盒子”，外界并不需要去了解对象内部的具体工作机理，只需要通过已经提供的函数，就可以对对象进行任意的操作。

对象的继承性使得对象具有了开放性的特征，用户可以在已经存在的对象上继承和派生出具有新的功能的对象，这一点在 GeoDatabase 中显得尤为重要。在 GeoDatabase 模型中，时刻都可以看到这种扩展性的产物，如对象类是表的扩展、要素类是对象类的扩展等。

地理数据库实现了数据的统一存储，即可以将所有的地理数据保存在一个数据库中。Geodatabse 将地理数据作为一个对象而保存，这些对象以行（Row）的形式被存储在要素类、对象类或要素数据集中。对象类和要素类的区别在于前者存储的是非空间数据,而后者保存的是具有相同字段的几何形体对象及其属性信息，例如一个要素类中要么全部存储的是点要素，要么全部是线要素，而且这些要素的字段都是相同的。

GeoDatabase 提供了多种数据输入的智能验证手段，如有效性规则、值域、拓扑等，它可以防止地理数据库在数据录入过程和处理过程中出现差错。

由于具有面向对象的特性，因此 GeoDatabase 具有很强的可扩展性能。在 GeoDatabase 中创建点和线的时候，可以把它们看做是对现实世界的精确模拟，比如点代表水坝或者变压器、线代表河流或者电力线。当它们代表不同对象的时候，它们的行为是有差别的。这一点正是 Coverage 模型所不具备的。

除了面向对象的优秀特性外，GeoDatabase 还提供了“版本”（version）的功能，这个功能必须依赖 SDE 数据库实现。一份数据可以有多个版本，以便于多人同时编辑一份数据，比如不同的电力工作组同时编辑一份电力网络数据，这些在现实生活中是常常遇到的。版本并不是一份数据的不同拷贝，它是对一份数据的不同状态记录。

地理数据被存储在大型关系数据库中是一个明智的选择和业界发展必然的趋势，由于数据库技术的快速发展，用户甚至可以将不同的数据分开在不同的地方放置，比如电力网络数据存放在电力部门，水利网络数据存放在不同城市的水利单位，使用 ArcSDE 这个工具，就可以像操作本机上的数据一样方便地获取到不同地方的地理数据。只要网络支持，实现一个全省、全国乃至世界范围内的 GIS 数据库都是可能的。

GeoDatabase 作为第三代的地理数据模型，以其良好的数据存储、编辑和可扩展能力，对于 ArcGIS 的成功起到了决定性的作用。

为了更好地操作地理数据和扩展地理数据功能，读者需要深入了解 GeoDatabase 的模型结构，图 8-1 就是 GeoDatabase 的核心结构模型图，要熟悉地理数据库模型，对图 8-1 的深入理解是必需的。

图 8-1 以简要的形式描述了 GeoDatabase 模型的主要对象，下面对这个模型做一个简单的归纳：

- 地理数据库模型的 Workspace（工作空间）对象代表了一个地理数据库，或者 shapefile 文件的文件夹或 Coverage 的工作空间
- Dataset（数据集）是数据的高级容器，它是任何数据的集合，这个数据范围非常广泛，可以是 Row、Table（表）甚至是要素类
- GeoDataSet 是一个包含了地理数据的数据集
- FeatureDataset（要素数据集）由要素类、几何网络和拓扑组成
- Table 是数据库中的一个二维表，它由 row（行）组成，其列属性由字段集设置
- ROW 是表中的一条记录，一个表中的记录的字段集是相同的

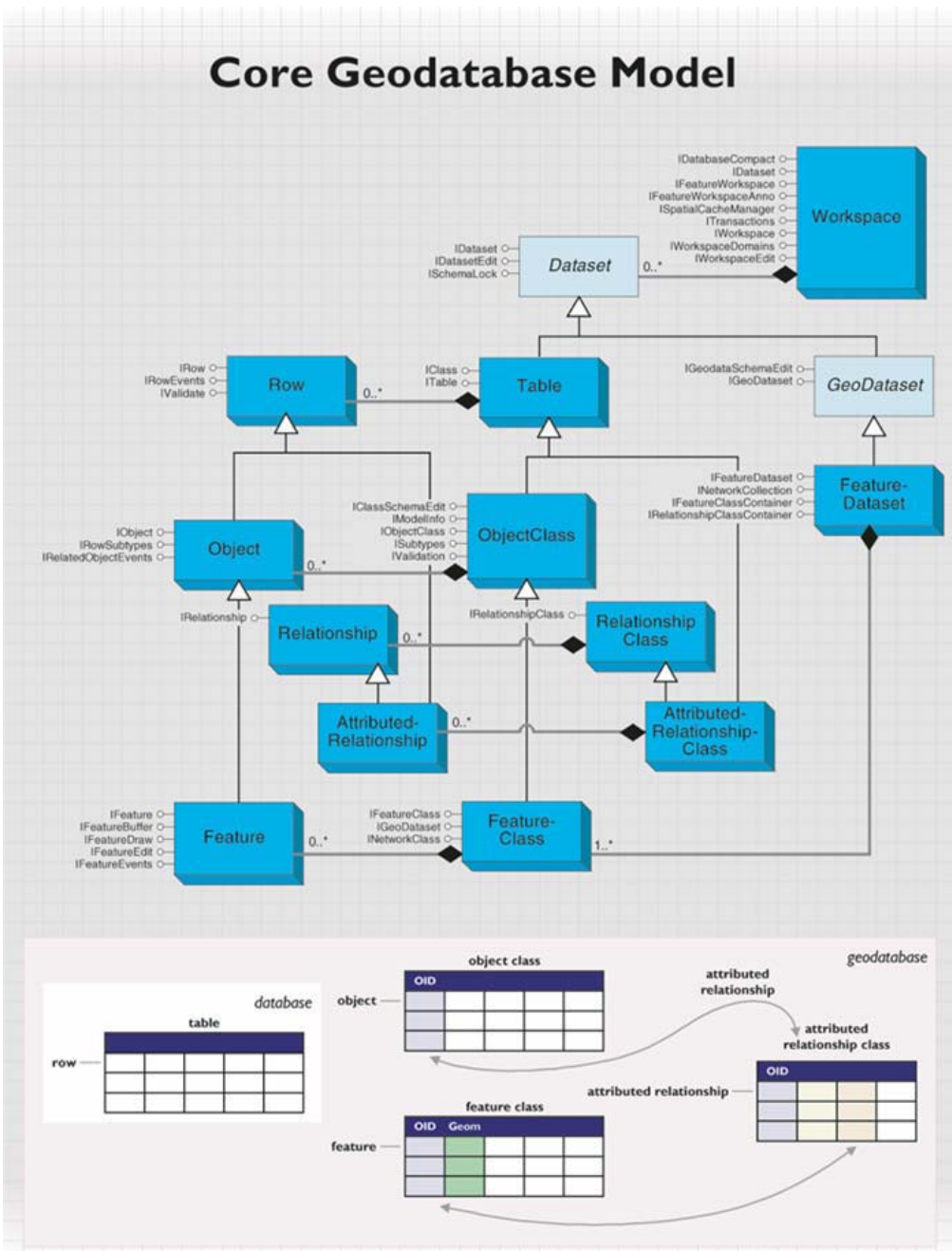


图 8-1 Geodatabase 模型图

- ObjectClass（对象类）是 Table 的扩展，它是一种具有面向对象特性的表，用于存储非空间数据
- Object 代表了一个具有属性和行为的实体（entity）对象，而不是简单的 row，

具有 OID

- FeatureClass（要素类）是一种可以存储空间数据的对象类，它是对象类的扩展
- Feature（要素）是要素类的一条记录，它是一个有几何字段的对象
- NetworkFeature 是一种参与了几何网络的要素
- RelationshipClass（关系类）代表关系，这个关系是通过表的外键（Foreign keys）建立的
- Relationship 是一对象类之间、要素类之间的联系，它可以控制这些对象之间的行为
- AttributedRelationshipClass（属性关系类）是一种用于存储关系的表
- AttributedRelationship 代表了多对多的关系

下面的章节将依据地理数据库模型的结构分别介绍 GeoDatabase 中的主要对象及其接口。

8.2 Workspace 及相关对象

在 GeoDatabase 中 Workspace 对象相当于一个地理数据库，它里面保存着数据集、要素类、属性表等数据对象；在物理级别上，它相当于地理数据库本身，如一个 Access 数据库或 SDE 数据库、coverage 作空间或者 Shapefile 文件夹。由于 GeoDatabase 模型是使用统一的方法来管理数据，因此，对于存储在不同数据库或文件中的数据，除了获得 Workspace 的方法不一样外，其余操作方法都是一致的，这种做法给用户带来了很大的方便。

由 esriWorkspaceType 枚举类型指定的 Workspace 有三种类型：esriFileSystemWorkspace、 esriLocalDatabaseWorkspace 和 esriRemoteDatabaseWorkspace. 基于文件类型的 Shapefiles 和 ArcInfo 的工作空间属于 esriFileSystemWorkspace. Personal geodatabase 属于 esriLocalDatabaseWorkspace.Enterprise geodatabase 属于 esriRemoteDatabaseWorkspace.

其它 workspace 类型包括：

- Raster Workspace: A workspace containing Grids and Images.
- Tin Workspace: A workspace containing TINs.

- CAD Workspace: A workspace containing CAD drawings.
- VPF Workspace: A workspace containing VPF data.

8.2.1 打开工作空间

Workspace（工作空间）是一个类（Class），这意味着用户不能直接新建它。为了获得一个工作空间，需要使用 WorkspaceFactory（工作空间工厂）对象来创建一个 Workspace。

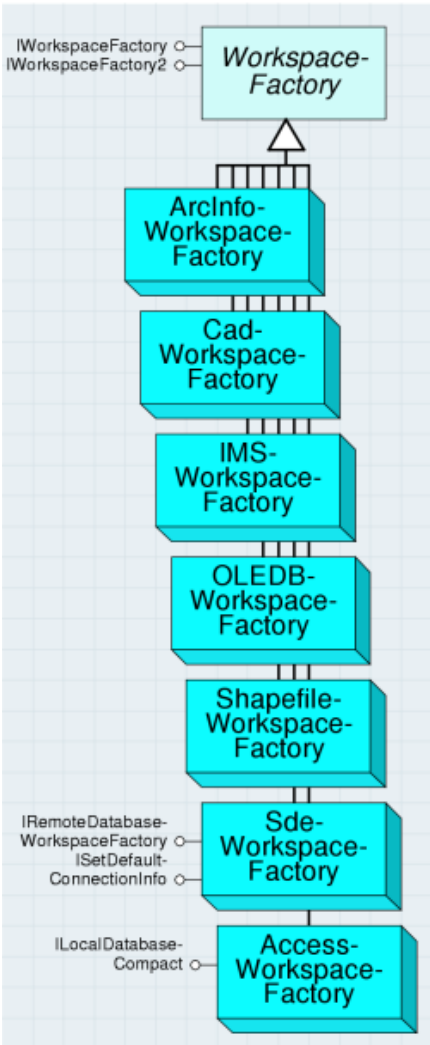


图 8-2 WorkspaceFactory 类型

WorkspaceFactory 是 GeoDatabase 的入口。它是一个抽象类，派生了很多的子类，例如 IMSWorkspaceFactory，AccessWorkspaceFactory，ShapefileWorkspaceFactory 等，如图 8-2 所示。不同类型的文件需要不同的工作空间工厂对象来打开一个工作空间。

IWorkspaceFactory 接口定义了所有工作空间对象的一般属性和方法，用户可以通过它管理不同类型的工作空间，所有的工作空间对象都可以通过这个接口产生。

使用 IWorkspaceFactory::Create 方法可以用于产生一个新的工作空间名称对象，如下面的例子：

```
private void CreateAccessWorkspace ()
{
   IWorkspaceFactory
pAccessWorkspaceFactory ;
   pAccessWorkspaceFactory = new
AccessWorkspaceFactoryClass();
   IWorkspaceName pWorkspaceName;
   pWorkspaceName =
pAccessWorkspaceFactory.Create(@"d:\temp",
"test", null, 0);
            }
```

这个方法的前两个参数是数据库的路

径和名称，第三个参数是输入一个属性集（propertyset）对象，Access 型数据库不需要传入这个参数，它通常是在新建 SDE 工作空间时才使用。

执行这个过程，打开 d:\temp 目录，将看到其中已经存在一个 test.mdb 文件。打开这个文件，里面已经存在许多建立好的表，如图 8-3 所示。

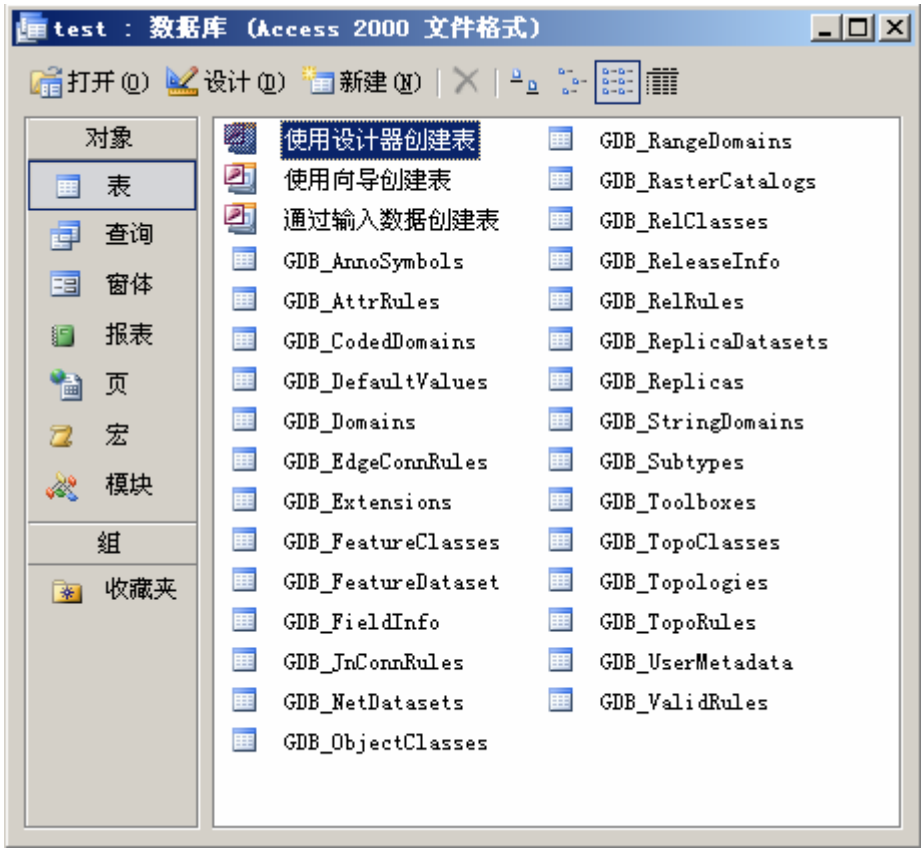


图 8-3 一个新建的 Access 工作空间

现在来讨论一个物理级别上的工作空间的结构，这是一个新的工作空间，其中没有任何的表、要素类或其他的数据集对象。可以看到的是，在这个工作空间中，系统已经自动建立了很多系统表，它用于管理工作空间的要素类、要素数据集、某些字段的缺省值、子类、拓扑关系一、拓扑规则和关系类等，正是由于系统表在逻辑上组织工作空间中存储的地理数据，使得一个工作空间的数据具有结构和层次关系，使得一个普通的关系型数据库得以变成一个“面向对象”的数据库。

不同类型的 WorkspaceFactory 对象的 Create 方法产生的工作空间名称对象是不同的。如 AccessWorkspaceFactory 产生的 Personal GeoDatabase 类型的工作空间名称是一个 mdb 文件，而 ShapefileWorkspaceFactory 对象则是产生的工作空间名称对象是一个文件夹。读者可以将名称对象认为是一种轻量级的工作空间，它

没有工作空间对象那么多的方法和属性，在后面的内容中将谈到这个对象。

IWorkspaceFactory::Open 方法和 IWorkspaceFactory::OpenFromFile 方法可以用于打开一个已经存在的工作空间。一般前者用于打开一个 SDE 数据库，后者用于打开一个文件类型数据。

IWorkspaceFactory::WorkspaceType 属性可以返回创建的工作空间的类型，工作空间的类型分为三种：esriLocalDatabaseWorkspace（本地数据库工作空间），如 Access 数据库的 Personal GeoDatabas 就是这个类型；esriFileSystemWorkspace（文件工作空间），如 shapefile；esriRemoteDatakaseWorkspace（远程数据库工作空间），如使用 ORACLE 作为地理数据库。

下表是程序员常用的工作空间工厂及相应的工作空间对象。

工作空间工厂对象	使用文件类型	工作空间对象类型
AccessWorkspaceFactory	打开 Personal GeoDatabase	esriLocalDatabaseWorkspace
ArcInfoWorkspaceFactory	打开 ArcInfo 工作空间	esriFileSystemWorkspace
CadWorkspaceFactory	打开 CAD 文件	esriFileSystemWorkspace
OLEDBWorkspaceFactory	使用 OLEDB 打开数据库的工作空间	esriRemoteDatakaseWorkspace
SdeWorkspaceFactory	打开 SDE 数据库	esriRemoteDatakaseWorkspace
TinWorkspaceFactory	打开 TIN 数据文件的工作空间	esriFileSystemWorkspace
TextFileWorkspaceFactory	打开文本文件	esriFileSystemWorkspace

下面分别介绍一些使用工作空间工厂对象打开 GeoDatabase 的例子。

- **打开 SDE 数据库**

```
public IWorkspace OpenSDEWorkspace(string Server, string Instance, string User,
string Password, string Database, string version)
        {
            IWorkspace ws = null;
            IPropertySet pPropSet    = new PropertySetClass();
            //使用SDE的工作空间工厂，不同类型的数据库连接的时候使用不同的工厂对象
```

```
        IWorkspaceFactory pSdeFact = new SdeWorkspaceFactoryClass();
        pPropSet.SetProperty("SERVER", Server);
        pPropSet.SetProperty("INSTANCE", Instance);
        pPropSet.SetProperty("DATABASE", Database);
        pPropSet.SetProperty("USER", User);
        pPropSet.SetProperty("PASSWORD", Password);
        pPropSet.SetProperty("VERSION", version);
        try
        {
            ws = pSdeFact.Open(pPropSet, 0);
        }
        catch( Exception ex)
        {
            MessageBox.Show("SDE Cann't Connection");
        }
        return ws;
    }
```

- 直接打开 **ORACLE** 数据库中工作空间

```
    private void OpenOracleWorkspace()
    {
        IWorkspaceFactory pWorkspaceFactory;
        pWorkspaceFactory = new OLEDBWorkspaceFactoryClass();
        IPropertySet pPropertyset;
        pPropertyset = new PropertySetClass();
        pPropertyset.SetProperty("CONNECTSTRING", "Provide＝
oraoledb.oracle;Data Source＝sde;User Id＝sde;Password＝sde");
        IFeatureWorkspace pWorkspace ;
        pWorkspace = pWorkspaceFactory.Open(pPropertyset, 0) as
IFeatureWorkspace;
        }
```

- 打开 **Personal GeoDatabae** 数据库

```
    public IWorkspace OpenAccessWorkspace(string connString)
    {
        IWorkspace ws = null;
        IWorkspaceFactory wsf = new AccessWorkspaceFactoryClass();
        ws = wsf.OpenFromFile(connString,0);
```

```
        return ws;
    }
```

- **打开 Shapefile 工作空间**

```
    public IWorkspace OpenShapfileWorkspace(string Location)
    {
        IWorkspace ws = null;
        IWorkspaceFactory wsf = new ShapefileWorkspaceFactoryClass();
        ws = wsf.OpenFromFile(Location,0);
        return ws;
     }
```

8.2.2 工作空间

工作空间（Workspace）在逻辑上是一个包含空间数据集和非空间数据集的数据容器，从 GeoDatabase 的对象模型图中可以看到，这些数据包括要素类、栅格数据集、表等对象。Workspace 对象也提供了用于创建新数据集和操作已经存在的数据集的方法。

工作空间对象 Workspace 是一个单一的对象，它没有子类，它代表了不同的保存地理数据的数据库或文件本身。Workspace 对象拥有众多的接口，管理着庞大的属性和方法集。

下面介绍 Workspace 对象主要的一些接口：

- **IWorkspace 接口**

IWorkspace 接口定义了一个工作空间最普通的属性和方法。则 IWorkspace::ConnectionProperties 返回工作空间的连接属性集对象；IWorkspace::Dataset 属性可以按照数据集(Dataset)的类型而返回一个数据集枚举对象；IWorkspace::Type 和 IWorkspace::WorkspaceFactory 属性则可以分别返回工作空间的类型和工作空间工厂的种类。

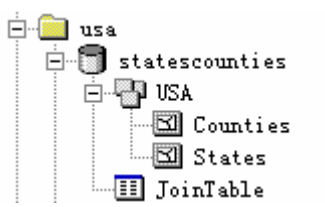


图 8-4 statescounties 工作空间

如图 8-4 所示，它是一个获取工作空间内数据集的例子，使用以下代码遍历一个 Access 数据库 statescounties，它是一个工作空间对象。

```
        private void getDataset()
        {
            IWorkspaceFactory pAccessWorkspaceFactory;
            pAccessWorkspaceFactory = new AccessWorkspaceFactoryClass();
            //打开工作空间
            IWorkspace pWorkspace;
            pWorkspace =
pAccessWorkspaceFactory.OpenFromFile(@"D:\Statescountles.mdb", 0);
            //获取工作空间内的数据集，参数为1时会取出所有类型的数据集
            IEnumDataset pEnumDataset;
            pEnumDataset = pWorkspace.get_Datasets(esriDatasetType.esriDTAny);
            IDataset pDataset ;
            pEnumDataset.Reset();
            pDataset = pEnumDataset.Next();
            //遍历所有数据集
            while( pDataset != null)
            {
                MessageBox.Show(pDataset.Name);
                pDataset = pEnumDataset.Next();
            }
        }
```

遍历后的结果为“usa”和“joinTable”两项。Usa 要素数据集内部的两个要素类不会被遍历出来。

- **IFeatureWorkspace 接口**

这个接口主要用于管理基于要素的数据集，如表（Table）、对象类（ObjectClass）、要素类（FeatureClass）、要素数据集（FeatureDataset）和关系类（RelationshipClass）等。

接口中所有的以 Open 开头的方法都需要传入一个要素集的名字，如 OpenTable、OpenDataset 或者是 OpenFeatureClass 方法。

IFeatureworkspace::OpenTable 方法可以用于打开一个已经存在的表或者对象类，这是因为对象类本身也是一个表，它也实现了 ITable 接口。

IFeatureworkspace::OpenFeatureClass 方法可以用于打开一个已经存在的要素类，无论这个要素类是在工作空间中还是在一个要素数据集中。

下面的代码是从工作空间中得到一个要素类，并将它放入 MapControl 控件。

```
        private void AddFeatureClassToMap(IWorkspaceFactory pWorkspaceFactory,
IPropertySet pPropertyset)
        {
            IFeatureWorkspace pFeatWorkspace ;
            pFeatWorkspace = pWorkspaceFactory.Open(pPropertyset, 0) as
IFeatureWorkspace;
            IFeatureClass pFeatClass ;
            pFeatClass = pFeatWorkspace.OpenFeatureClass("sde.sta");
            IFeatureLayer pFeatLyr;
            pFeatLyr = new FeatureLayerClass();
            pFeatLyr.FeatureClass = pFeatClass;
            pFeatLyr.Name = pFeatClass.AliasName;
            IMap pMap;
            pMap = axMapControl1.Map;
            pMap.AddLayer(pFeatLyr);

        }
```

IFeatureworkspace::OpenFeatureQuery 方法可以通过一个预查询语句打开一个虚要素类，这个虚要素类是基于多个实际存在的要素类之间的关系，它可以将多个有关系的要素类作为一个要素类放入 Map 对象中去。读者可以将它理解为一个数据库中的视图对象。

接口中所有的 CreateFeatureClass 方法可以产生一个要素类，这个要素类是直接放置在工作空间中的，数据集的物理格式要么是一张二维表，要么是一个 shapefile 文件等，产生的格式取决于工作空间的类型。

下面是在一个 Access 数据库中产生一个要素类的例子。

```
        private IFeatureClass CreateAccessFeatureClass(string strWorkspace,string
strBrowseName, esriGeometryType geomType)
        {
            //打开工作空间
            IWorkspaceFactory pWSF;
            pWSF = new AccessWorkspaceFactoryClass();
            IWorkspace pWS;
            pWS = pWSF.OpenFromFile(strWorkspace, 0);
            IFeatureWorkspace pFWS;
            pFWS = pWS as IFeatureWorkspace;
            //设置GeometryDef属性，给SHAPE字段使用
            IGeometryDefEdit pGeomDef ;
```

```
            pGeomDef = new GeometryDefClass();
            pGeomDef.GeometryType_2 = geomType;
            pGeomDef.SpatialReference_2 = new UnknownCoordinateSystemClass();
            //新建字段
            IFieldEdit pField;
            IFieldsEdit pFieldsEdit;
            pFieldsEdit = new FieldsClass();
            //几何字段
            pField = new FieldClass();
            pField.Type_2 = esriFieldType.esriFieldTypeGeometry;
            pField.GeometryDef_2 = pGeomDef;
            pField.Name_2 = "Shape";
            pFieldsEdit.AddField(pField);
            //ID字段
            pField = new FieldClass();
            pField.Type_2 = esriFieldType.esriFieldTypeDouble;
            pField.Name_2 = "ID";
            pFieldsEdit.AddField(pField);
            //产生OID字段
            pField = new FieldClass();
            pField.Name_2 = " OBJECTID";
            pField.Type_2 = esriFieldType.esriFieldTypeOID;
            pFieldsEdit.AddField(pField);
        //返回新建的要素类
            return pFWS.CreateFeatureClass(strBrowseName, pFieldsEdit, null, null,
esriFeatureType.esriFTSimple, " Shape", "");
        }
```

注意：用户可以使用 null 关键字来设置某个方法可以忽略的参数，对于 ArcSDE+大型关系数据库构成的 SDE 地理数据库而言，如果要在里面添加一个要素类，CreateFeatureClass 的最后一个参数不能为 null，而必须是一个空字符串“”。

- **IGeoDatabaseRelease 接口**

使用 ArcSDE+大型数据库构成的多用户地理数据库的一个重要特点是它支持多用户操作，即同样一份数据可以让多个用户同时进行操作，当然，这并不是因为 ArcSDE 将地理数据库中的数据给每个用户都拷贝了一份，而是采用了所谓的“版本”技术。

所谓版本，其实是指地理数据的一个“编辑状态”，假如一份数据同时被 A、B、C 三个人进行编辑，系统需要给三个人建立一份自己的版本，如“版本 A、版本 B 和版本 C”，无论三人如何编辑这份数据，如更新、删除、添加，在地理数据库中都只是对它们的编辑行为进行了记录。如 A 删除了某个要素，地理数据库将记录在版本 A 上某条要素被删除，并且让这个要素在版本 A 中不可见，但并不意味着在数据库中某条记录被删除了。

当所有人的编辑都完成以后，可以将三个人的编辑工作进行汇总，即合并版本。如果 ArcSDE 发现三个人对同一个要素进行编辑，且编辑状态不一样的时候，就会出现“版本冲突”，对于版本的最后确定，将取决于用户的权限。

用户没有新建一个版本的时候，地理数据库始终处于“default 版本”状态，即 SDE 数据库必须拥有一个版本状态。

IGeoDatabaseRelease 用于管理地理数据库的版本操作。如 IGeoDatabaseRelease::CurrentRelease 用于获取当前版本，IGeoDatabaseRelease::Upgrade 方法可以将数据库升级为目前的版本状态。

8.2.3 PropertySet

Propertyset 对象是一个专门用于设置属性的对象，它是一种 name-value 对应的集合，类似 Hash 表。属性名必须是字符串，属性值可以是字符串、数值或日期，也可以是一个对象。属性值支持通过名字来查找属性的方法。

IWorkspaceFactory::Open 方法要求用 Propertyset（属性集合）来设置打开一个 Workspace，这个类使用一个 SetProperty 方法来设置属性，它携带了全部的连接信息。可以查看上面打开 SDE 数据库使用的方法。

如打开一个 SDE 数据库的时候，进行如下配置（本例使用的是 Oracle9i＋ArcSDE）：

```
IPropertySet pPropertyset;
pPropertyset = new PropertySetClass();
pPropertyset.SetProperty("Server", "data") ;   //服务器
pPropertyset.SetProperty("Instance", "esrisde");   //SDE实例
pPropertyset.SetProperty("user", "sde") ;   //SDE数据库的用户名
pPropertyset.SetProperty("password", "sde");   //SDE数据库的密码

pPropertyset.SetProperty("version", "sde.DEFAULT");   //默认版本
```

关于 Oracle 数据库的知识，可以查阅相关书籍。

使用数据集对象来打开工作空间使用在 SDE 数据库中，打开一个 Personal GeoDatabase 或者文件类型的工作空间并不需要这个对象。

如果其他的方法需要通过一个对象传入多个信息，也可以使用 IPropertyset 对象来完成。

8.2.4 名称对象

名称（Name）对象标识并且定义了 GeoDatabase 对象（如数据集或者工作空间）或地图对象（如图层）。尽管名称对象只是它代表的对象的一个“代理”，但它支持程序员使用实例化的特定对象的 Open 方法。

下面是一个使用 Open 方法实例化的例子：

```
IName pName;
IFeatureClass pFeatureClass;
pName = pFeatureClassName as IName;

pFeatureClass = pName.Open () as IFeatureClass;
```

由于名称对象可以创建和返回一个 GeoDatabase 实例，因而它和 GeoDatabase 对象相比是一个轻量级对象。如一个要素数据集中包含很多要素类，但是一个要素数据集名称对象却并不如此。

名称对象的子类众多，如 TableName、FeatureClassName、ObjectClassName 等。本书中在 WorkspaceFactory 一节中曾经使用过 Create 方法创建过一个工作空间对象，事实上，创建的并不是一个 Workspace，而是产生了一个 WorkspaceName 对象，如果需要产生 Workspace 对象，可以使用 Open 方法，如下例：

```
private void CreateShapefileWorkspace()
{
    IWorkspaceFactory pWorkspaceFactory;
    pWorkspaceFactory = new ShapefileWorkspaceFactoryClass();
    IWorkspaceName pWorkspaceName;
    pWorkspaceName = pWorkspaceFactory.Create("d:\temp", "test", null, 0);
    IName pName;
    pName = pWorkspaceName as IName;
    IWorkspace pShapeWorkspace;
    pShapeWorkspace = pName.Open() as IWorkspace;

}
```

在上例中的 pShapeWorkspace 才是一个工作空间对象。

8.3 Dataset 对象

Dataset（数据集）是一个代表了 Workspace 中所谓的数据集合的抽象类，它是一个高级别的数据容器。在理解 Dataset 对象的时候需要使用广义的概念来看待，即应该意识到其实 Table，Feature，RasterDataset，几何网络等对象都是 Dataset 的一种，即所有放在工作空间的对象都是一种数据集对象，数据集中的数据可以是一行记录、一个字段、一个关系乃至一个表、一个数据集。因此，GeoDatabase 也可以被称为一个数据集对象。

数据在关系数据库中都是以表的形式存在的。在 GeoDatabase 中看来也是这样，尽管是地理数据，但是它们也是存储在一个表里面，只不过某个字段可以用来存储图形信息而已。

Dataset 对象分为两大类，一种是 Table，一种是 GeoDataset。前一种可以简单看成是一张二维表，它是由一条条记录组成的，是保存记录 Row 的容器；后一种是要素类的容器，它是由一个个要素类组成的，保存在一个 GeoDataset 中的要素类具有相同的空间参考。

- **IDataset 接口**

打开工作空间后，需要进行处理的就是数据集对象，所有的数据集都实现了 IDataset 接口，它定义了所有的要素集都有的属性和方法。

IDataset::FullName 属性可以返回与这个数据集相联系的名称对象；IDataset::PropertySet 返回数据集的属性集对象；IDataset::Subsets 可以返回这个数据集中包含的数据集，如果是一个要素数据集的话，这个属性将会返回它包含的要素类对象。

IDataSet 接口的 CanCopy、CanDelete、CanRename 方法用于设置数据集是否能够被拷贝、删除和重命名。当然，它也定义了 Copy、Delete 和 Rename 方法供用户来操作数据集。

- **IDatasetEdit 接口**

当用户在一个工作空间中启动编辑流程进行数据编辑时，可以使用 IWorkspaceEdit::StartEditing 方法。一旦这个过程开始，就可以使用 IDatasetEdit::IsBeingEdited 方法来获知一个特定的数据集是否处于编辑流程中。

并不是工作空间内所有的数据集都会处于编辑流程，这是因为连接用户可能没有编辑数据集的权利，或者这个数据集没有被注册为版本。在多用户地理数据库中，如果数据没有被注册为某个版本状态，是不能被编辑的。

- **ISchemaLock 接口**

当一个数据集被建立后，它的 Schema（模式）可以被修改，Schema 是一个要素集的结构而非数据本身，用户可以将它看做是关系数据库中的数据字典。一份数据在同一时刻不能被两个用户同时执行物理上的操作，当修改 Schema 或者是执行其他需要独享数据的操作时，需要对数据库建立一个排他性的锁定，以免数据在使用过程中出现异常。

Schema 的锁定有两种，一个是独占型（exclusive）锁定，另一种是共享型（shared）锁定。使用 ISchemaLock::ChangeschemaLock 方法，可以将一个数据集的 Schema 进行锁定。

关于 Schema 将在本书会在要素类的章节中详细讨论。

8.3.1 GeoDataset 类

GeoDataset（地理数据集）是一个抽象类，它代表了拥有空间属性的数据集。GeoDataset 的例子包括要素数据集 FeatureDataset、要素类 FeatureClass、TIN 和栅格数据集 RasterDataset。非 GeoDataset 的数据集包括非空间属性的对象类 ObjectClass 和关系类 RelationshipClass 等。

简单而言，如果一个数据集可以设置空间参考（SpatialRefrence），那么它就是 GeoDataset 对象。

IGeoDataset 接口定义了 GeoDataset 对象的空间信息，包括空间参考和范围属性。通过 IGeoDataset::SpatialRefrence 可以获得一个 GeoDataset 对象的空间参考，IGeoDataset::Extent 则可以获得要素集的定义范围。

IGeoDatasetSchemaEdit 接口可以让程序员去改变一个 GeoDataset 的空间参考，IGeoDataset::AlterSpatialRefrence 方法并没有改变数据集中的数据，它只是重新设置了与数据集关联的空间参考而已。这个方法多使用于给一个空间参考为 Unknown 的地理数据集设置空间参考。

8.3.2 FeatureDataset 对象

要素数据集对象在 GeoDatabase 是一个简单要素类的容器，在这个容器内放置着具有相同空间参考的要素类。使用要素数据集的情况十分广泛，例如几何网络必须建立在一个要素数据集中。除了简单要素类外，几何网络和关系类也可以放在要素数据集中。

由于地理数据通常是由不同的部门和机构获得的，这些数据使用的空间参考常

常各不相同，为了便于管理这些数据，用户也可以将不同部门提交的数据放在一个要素集中，这样要素数据集实际上成为一个数据组织者。

在工作空间中对一个要素类进行编程的时候，需要注意这个要素类放在什么地方，是直接放在工作空间中（这种要素类被称为 standalone featureclass），还是放在一个要素数据集中。当使用 IWorkspace::Datasets 属性来遍历一个工作空间内的数据集时，返回的只是直接放在工作空间的数据集，而保存在一个要素数据集中的要素类则不会被遍历（参看工作空间中的代码例子）。

每一个在地理数据库中的数据集都拥有一个独一无二的名字。用户特别需要注意的是，如果一个要素类的名字和同工作空间的要素数据集中的要素类的名字相同的情况是不允许的，即如果在工作空间中存在一个名为 A 的要素类和 B 的要素数据集，B 中如果再产生一个名为 A 的要素类是不会成功的，这一点和基于文件的数据模型不一样。在 Shapfile 中，如果两个同名文件在不同文件夹中是可以的。之所以产生这样的问题，其原因是无论一个要素类是直接放在工作空间中，还是放在工作空间的一个要素数据集中，这些差别仅仅是逻辑上的，而它们的物理组成都是数据库中的一张二维表，并且表名就是要素类的名字，在一个数据库中不能出现两个同名的二维表，因此也就不能产生两个同名的要素类。

使用 IFeatureWorkspace::OpenFeatureClass 方法可以打开工作空间中的任何一个要素类，无论它是直接存放在工作空间还是存放在工作空间中的一个要素数据集中。

下面介绍一下要素数据集中的主要接口。

- **IFeatureDataset**

这个接口是从 IDataset 接口继承而来的，使用 IFeatureDataset::CreateFeatureClass 方法可以用来在要素数据集中产生一个新的要素类。这个方法和 IFeatureWorkspace::CreateFeaureClass 的方法类似，但需要注意的是 IFeatureDataset::CreateFeatureClass 产生的要素类的空间参考必须与要素数据集的空间参考一致。

新建要素类的空间参考是通过它的几何字段来设置的。

下面的代码片段可以用于遍历一个要素数据集中的要素类：

```
private void GetFeatureClass()
{
    IWorkspaceFactory pWorkspaceFactory;
    pWorkspaceFactory = new AccessWorkspaceFactoryClass();
    IFeatureWorkspace pWorkspace;
   pWorkspace = pWorkspaceFactory.OpenFromFile(@"d:\arcgisdata\test.mdb", 0)
```

```
as IFeatureWorkspace;
            //得到名称为cade的数据集
            IFeatureDataset pFeatureDataset;
            pFeatureDataset = pWorkspace.OpenFeatureDataset("cade");
            IEnumDataset pEnumDataset;
            //遍历要素数据集中的子类，也是数据集对象
            pEnumDataset = pFeatureDataset.Subsets;
            pEnumDataset.Reset();
            IDataset pDataset;
            pDataset = pEnumDataset.Next();
            while (pDataset != null)
            {
                MessageBox.Show(pDataset.Name + " type:" + pDataset.Type);
                pDataset = pEnumDataset.Next();
            }
        }
```

- **IFeatureClassContainer**

IFeatureClassContainer 接口用于管理要素数据集里面的要素类。该接口的 ClassByName 和 Class(index)等属性都可以用来获取数据集中的特定的要素类。

IFeatureClassContainer::ClassCount 和 Classes 属性分别可以获得要素数据集中的要素类的数目和得到一个要素类的枚举对象。

在 SDE 地理数据库中新建一个 Table 对象，即建立一张二维数据表，如果该表没有被注册为版本，则它就是一张普通的数据表；当表被注册到某个版本后，它就是一个对象类。在地理数据库中，每一个对象类都拥有一个独一无二的 OID 号码用于与其他的对象类相区分。IObjectClass::ObjectID 属性可以返回这个 ID 号，它是一个长整型数值。

对象类的 ID 号码(OID)是在它创建或者被注册到地理数据库的时候由系统自动分配的，它不能被修改。如果一个要素类或者表存储在数据库中，但是没有被注册，它的 ID 号将总是 - 1。

IFeatureClassContainer::ClassByID 属性可以让程序员通过对象类的 OID 值返回一个特定的对象类。

- **IDatasetContainer**

IDatasetContainer 可以管理数据集对象中的数据集，它可以在工作空间和要素数据集之间移动数据集对象，工作空间也实现了这个接口。移动的过程可以使用

IDatasetContainer::AddDataset 方法完成。

需要注意的是，当用户将要素类从工作空间移动进一个要素数据集的时候，源要素类的空间参考必须和目标数据集匹配。

下面的过程是一个移动要素类的例子，pFeatDataset 是工作空间中的一个要素数据集，pDataset 为要移动的数据。如果想将数据直接放在工作空间，bStangalone 则为 true，否则为 false：

```
private void MoveDataset(IFeatureDataset pFeatDataset , IDataset pDataset, bool
bStandalone)
        {
            IDatasetContainer pDatasetContainer;
            //如果要将数据对象集直接放在工作空间内，则bStandalone为true
            if (bStandalone)
            {
                pDatasetContainer = pFeatDataset.Workspace as IDatasetContainer;
            }
            else
            {
                pDatasetContainer = pFeatDataset as IDatasetContainer;
            }
            //将数据集放入选择的容器
            pDatasetContainer.AddDataset(pDataset);

        }
```

- **IRelationshipClassContainer 接口**

要素数据集中还可以存储关系类对象，可以通过这个接口来添加、新建和获得要素数据集中的关系类对象。

IRelationshipClassContainer::CreateRelationshipClass 方法可以让程序员在要素数据集中产生一个新的关系类；而 AddRelationshipClass 方法则可以移动一个关系类到要素数据集中，实际上，使用 IDatasetContainer::AddDataset 方法也可以做到这一点。

8.4 表、对象类和要素类

在数据库中存储的对象，无论它们是什么类型--Table、FeatureClass 还是其他，都是存储在数据库的二维表中，因此表(Table）可以被认为是数据的最小的容器，如图 8-5 所示。

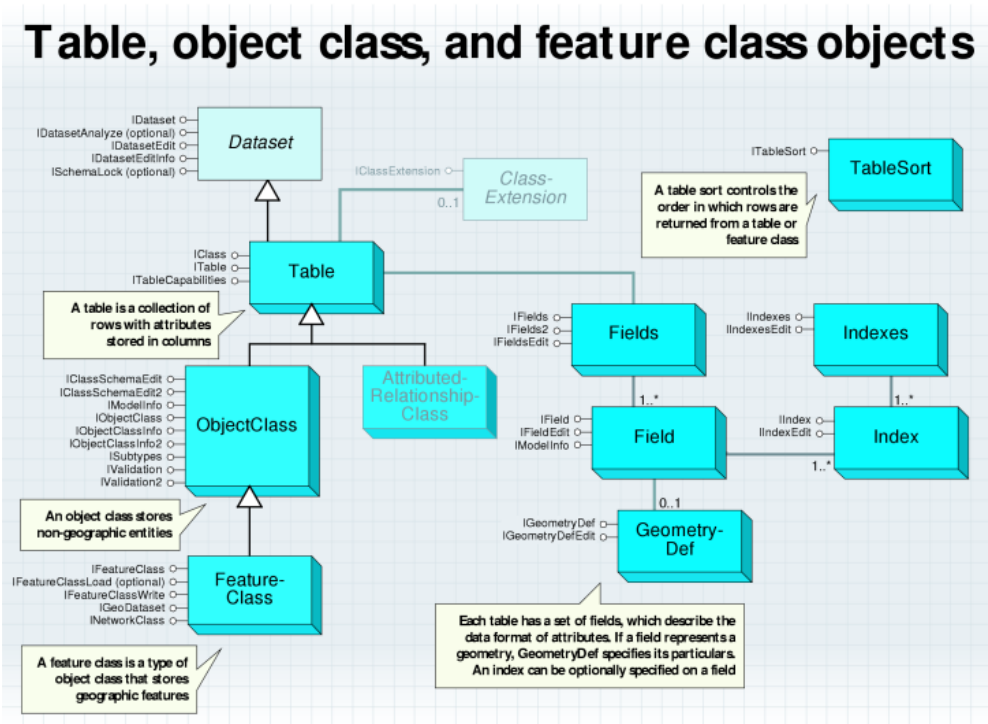


图 8-5 Table 及其子类结构

图 8-5 所示是 GeoDatabase 模型中一个很重要的关系，它揭示了 GeoDatabase 是如何在普通关系数据库基础上发展的。普通关系数据库，无论是 Access 还是 Oracle，其组成部分都是二维表；而 Table 对象，就代表了关系数据库中的一张数据表。这张表仅仅能够简单存储属性数据，它是由 Row 对象，即一条条记录组成，Table 对象只能存储非空间数据。

ObjectClass 对象是在普通关系数据库基础上的扩展，它不仅仅是一张关系表，而且是一个具有属性和行为的数据表，它也是用于存储非空间数据。它与 Table 对象的差别在于，ObjectClass 是由 Object 对象组成，尽管 Object 的物理形式仍然是一条记录，但它是具有属性和行为的一个对象，而非简单的数据记录了。

FeatureClass 则是在 ObjectClass 对象上的更进一步发展，它除了具有面向对象的特性外，还能够存储空间数据，这些空间数据以一个个具有属性和行为的要素为表现形式。要素类使用 Shape 字段作为它的数据字段，用于保存一个几何形体对

象。

当用户在一个 Personal GeoDatabase 中新建一个要素类后，打开这个 Access 数据库看看里面新出现的表，会发现其实属性数据和空间数据都是存储在一张单独的表中；如 States 这个要素类，它是由两张表组成的：States 和 States_Shape_Index。States 所有的数据都被放在 States 要素类中，其几何字段数据被当作二进制流存放在 SHAPE 字段中。由于二进制数据的索引方式与一般的文本、数字方式不同，因而需要一张专门的表来制作这个字段的索引。

正是由于 ArcObjects 的 GeoDatabase 模型的逻辑操作脱离了物理层次上的区别，使得用户可以不再考虑他们操作的是什么样类型的地理数据，不管是文件类型、Access 数据库还是 Oracle 或 SQLServer 数据库，对其中的空间数据和非空间数据使用的操作方式都是一样的。这种统一的数据模型观点，正是 GeoDatabase 最吸引 GIS 程序员的地方之一。

8.4.1 Table 对象

从关系数据库的角度而言，Table 对象代表了 GeoDatabase 里面的一张二维表或者视图（view）、它是最简单的数据容器对象，存储的元素是 Row 对象，因此它也是一个数据集对象。Table 是 ObjectClass 对象和 FeatureClass 对象的父类。

Table 是由 Row 对象组成的，Row 对象代表了表中的一条记录。表有一列或多列，它被称为字段，关于字段将在后面的内容中详细谈及到。

Table 对象由于是基本的数据容器，它实现了几个接口用于管理这些存储在表中的数据：

- **IClass 接口**

所有的 Table 类（包括 Table，ObjectClass 和 FeatureClass）都实现了 IClass 接口，通过这个接口定义的属性和方法可以实现表对象的最一般的操作。

IClass::Fields 可以获得一个表的字段结构，IClass::HasOID 属性则可以返回一个表是否有 OID 字段，用户通过它的返回值可以知道这个表是否被注册为版本。而 IClass::OIDFieldName 属性返回 OID 字段的字段名。

IClass 接口定义的方法是用于操作表的字段和索引（Index）的，如 AddField 用于给表对象添加一个字段、AddIndex 用于给表增加一个索引。索引是在数据库表中对一个或多个列的值进行排序的结构，索引可以使您更快地得到信息。

IClass::FindField 方法可以通过一个字段的字段名获得它在表的字段集中的索引号。

- **ITable 接口**

Table 主要实现了 ITable 接口，它继承自 IClass，因而这两个接口之间不用 QI。这个接口定义的方法可以供用户查询、选择、插入、更新、删除表中的记录。在一张表中获得记录或者对记录（Row）的修改都是通过这个接口定义的方法。

CreateRow 方法可以用于在数据库中产生一个 Row 对象，当它产生后，除了系统自动给它一个 OID 值外，其他的字段的值都是空的。当然，如果在字段设计的时候设置了某个字段初始化有缺省值，那么这个字段在产生的新行中默认为缺省值。

一旦产生了一个新的 Row 对象，就可以使用 IRow::Value 属性来设置这个对象的不同字段的值，这个属性是可读写的。当一个 Row 对象的所有非空字段值都被设置后，就可以使用 Store 方法将这一新的 Row 放入表中，否则它是不会被保存到数据库的。

在 ArcCatalog 中新建一个表 Test，设计表字段的时用户会发现系统中已经自动存在一个 ObjectID 字段，在它下面添加 Name 和 Age 两个字段，其字段类型分别是“Text”和“ShortInteger”。然后向这个表中添加数据。下面是一个往表中插入新 Row 的例子：

```
private void AddRow()
{
    IWorkspaceFactory pWorkspaceFactory;
    pWorkspaceFactory = new AccessWorkspaceFactoryClass();
    IFeatureWorkspace pWorkspace;
   pWorkspace = pWorkspaceFactory.OpenFromFile(@"d:\arcgisdata\test.mdb", 0)
as IFeatureWorkspace;
    ITable pTable;
    //打开一个属性表
    pTable = pWorkspace.OpenTable("test");
    IRow pRow;
    pRow = pTable.CreateRow();
    //给row对象插入值后保存
    pRow.set_Value(1, "jack");
    pRow.set_Value(2, 20);
    pRow.Store();

}
```

执行这个过程，刷新 ArcCatalog，可以看到新建的 Test 表中已经有了一个新记录，其 OID 将自动为 1。

CreateRowBuffer 方法可以用于产生一个 RowBuffer（缓冲行）对象，这个对象可以被 InsertCursor（插入型游标）使用，用于往表中插入一条新记录。RowBuffer 并不是一个实际存在的 Row 对象，它是一个临时对象，没有 OID 值。

GetRow 方法可以根据一条记录的 OID 值来获得 Row 对象本身。可以将得到的 Row 对象的值改变，然后使用 IRow::Store 方法，达到更新一条记录的目的，如下面的代码段：

```
IRow pRow;
pRow = pTable.GetRow(1);
pRow.set_Value(1, "jack");
pRow.set_Value(2, 25);
pRow.Store();
```

读者再次查看 Test 表，其第一条记录的 Age 字段值是否已经变成了 25。

Insert 方法用于得到一个具有插入记录功能的插入型游标，它和 CreateRowBuffer 产生的 RowBuffer 对象需要配合使用。

RowCount 方法可以得到表中满足要求的记录条数。它需要设置一个 QueryFilter 过滤器对象来确定查询条件，如果需要返回所有的值，则将这个过滤器设置为 null 即可。

Search 方法则可以用于查询满足要求的记录，它的返回值是一个查询型游标，这个游标指向所有满足条件的 Row 对象。

Update 方法的参数和 Search 差不多，但是它的返回游标是用于“更新”目的的，是更新型游标，不同于 Insert 的“插入”和 Search 的“寻找”目的。

下面的例子是从表中获得记录：

```
IWorkspaceFactory pWorkspaceFactory;
pWorkspaceFactory = new AccessWorkspaceFactoryClass();
IFeatureWorkspace pWorkspace;
pWorkspace = pWorkspaceFactory.OpenFromFile(@"d:\arcgisdata\test.mdb", 0)
as IFeatureWorkspace;
ITable pTable;
//打开一个属性表
pTable = pWorkspace.OpenTable("test");
ICursor pCursor;
//在表中寻找符合要求的Row，过滤器为null时意味着寻找全部的记录
pCursor = pTable.Search(null, false);
IRow pRow;
```

```
//获得第一条记录
pRow = pCursor.NextRow();
while( pRow != null)
{
    //弹出对话框显示row第3个字段的值
    MessageBox.Show(pRow.get_Value(2).ToString());
    pRow = pCursor.NextRow();
}
```

Select 方法返回的是一个选择集对象，它与 Search 方法的不同在于 Search 方法返回的是一个指向被寻找数据的游标，而它则要么返回被选择 Row 对象的 ID 值，要么就是 Row 对象本身，这个差别取决于构造选择集时使用的参数。

下面的例子是在表中构造选择集：

```
//为选择集产生一个scratch workspace
    IWorkspace pScratchWorkspace;
    IScratchWorkspaceFactory pScratchWorkspaceFactory;
    pScratchWorkspaceFactory = new ScratchWorkspaceFactoryClass();
    pScratchWorkspace = pScratchWorkspaceFactory .DefaultScratchWorkspace;
    //产生一个过滤器
    IQueryFilter pQFilt;
    pQFilt = new QueryFilterClass();
    pQFilt.WhereClause = "OID>77000";
    //使用过滤器选择row
    ISelectionSet pSelectionSet;
    pSelectionSet = pTable.Select(pQFilt,
esriSelectionType.esriSelectionTypeIDSet, esriSelectionOption.esriSelectionOptionNormal,
pScratchWorkspace);
```

上面的例子在表 pTable 中构造了一个新的选择集，注意其中的 pScratchWorkspaceFactory 参数，它是一个 IScratchWorkspaceFactory 对象，它用于在操作系统的 Temp 目录产生一个临时的 Personal GeoDatabase 供选择集使用。而 Select 方法的第二个参数 esriSelectionTypeIDSet 侧让选择集返回的是一个存储着被选择到的 Row 对象的 ID 号的二维表。

关于选择集的问题，本书将在后面的章节中深入探讨。

8.4.2 ObjectClass 对象

ObjectClass（对象类）是 Table 对象的子类，它也是由 Row 组成的表。在对象类中的 Row 对象是作为一个对象实体而被存储的，这个实体不仅仅拥有 Table 中的一条记录的特性，而且像一个对象一样拥有属性和行为，它也支持 IRow 和 IObject 接口。ObjectClass 其实是 Table 的一种扩展，它就是一张“智能化”的表。

一个对象类和其他对象类之间可以存在任意数目的关系（Relationship），这些关系可以用于控制对象的行为。

一个对象类可以使用一个字段用于将它自身划分为几个子类（Subtypes）。所有的子类都有与对象类相同的字段设置，而且它们被存储在同一个表中。用户可以将 ObjectClass 的子类认为是将一个对象类以某个字段来逻辑划分的对象类的一部分。

假如有一个存储土地信息的对象类，可以依据土地的“使用类型”字段可以将这个对象类划分为几个子类，如“商业土地类”、“工业土地类”、“生活土地类”等，尽管它们都放在同一个表中，但是 GeoDatabase 可以将一个子类看做一个独立的对象。事实上，GeoDatabase 也是将它们作为单独的对象类看待的。

每个对象类都有一个非负的对象类 ID 号，它是在这个对象类创建时或是在它被注册到地理数据库时由系统自动赋给的。对象类还有一个 AliasName（别名）属性，它是出于方便显示的目的而使用的，它用于设置这个对象类的附加注释信息。

每个对象类都拥有 Name 属性，这个值和存储在关系数据库中的对象类的物理表的名字是一致的。

- **IObjectClass 接口**

所有的对象类都支持 IObjectClass 接口，它是从 IClass 接口继承而来的。IObjectClass::AliasName 属性可以返回对象类的别名，如果一个对象类没有设置别名属性，那么这个别名将和对象类的 Name 属性一样。

IObjectClass::ObjectClassID 属性则可以返回这个对象类的 OID 值。其实一个没有被注册的对象类也是有 OID 值的，但是这个值总是为-1，而且通过 IClass::HasOID 属性来查询的时候，返回值为 false。

由于对象类之间可以存在关系，一个对象类在关系中担当的角色有三种：Origin（关系源角色）、Destination（关系目的角色）和两种关系角色同时存在。关于关系类的内容将在后面的章节中详细讨论。

使用 RelationshipClasses 属性可以根据关系类型返回对象类中的关系。

```
IRelationshipClass pRelationshipClass;
IEnumRelationshipClass pEnumRelaClass;
pEnumRelaClass =
pObject.get_RelationshipClasses(esriRelRole.esriRelRoleAny);
```

```
            pEnumRelaClass.Reset();
            pRelationshipClass = pEnumRelaClass.Next();
            while (pRelationshipClass != null)
            {
                MessageBox.Show(pRelationshipClass.OriginClass.AliasName + "：" +
pRelationshipClass.DestinationClass.AliasName);
                pRelationshipClass = pEnumRelaClass.Next();
            }
```

pObject 为 IObjectClass 类型的变量。

- **IClassSchemaEdit 接口**

IClassSchemaEdit 和 IClassSchemaEdit2 接口都可以用于修改一个对象类的属性。如对象类的别名，对象类某个字段的缺省值、值域，改变对象类字段的别名等。这些已经设计好的字段结构，就称为模式（Schema），它是一个对象类的结构描述，而非数据本身。

使用 IClassSchemaEdit::RegisterAsObjectClass 方法可以将一个存在的表注册到 SDE 地理数据库，这个过程其实是给表添加一个 OID 字段，并将这个它作为一个对象类登记到地理数据库的数据字典中去。

使用 IClassschemaEdit 中的方法时，用户必须取得这个对象类的完全控制权，以免在修改的过程中有别的程序使用这个对象类，为此需要使用 SchemaLock 接口来给数据集（对象类也是一个数据集）设置一个独占型的锁定。

现在用户在一个 Access 数据库 Test 中新建一个要素类为“build”，设计这个要素类的字段，在其中新建一个字段为“Age”，其别名设置为“Building_Age”。执行下面的语句将改变这个字段的别名：

```
        private void ChangeSchema()
        {
            IWorkspaceFactory pWorkspaceFactory;
            pWorkspaceFactory = new AccessWorkspaceFactoryClass();
            //打开工作空间
            IWorkspace pWorkspace;
            pWorkspace = pWorkspaceFactory.OpenFromFile(@"d:\arcgisdata\test.mdb",
0);
            //接口查询到IFeatureWorkspace
            IFeatureWorkspace pFeatWorkspace;
            pFeatWorkspace = pWorkspace as IFeatureWorkspace;
            //打开准备修改Schema的数据集
```

```
        IObjectClass pObjclass;
        ITable pTable;
        pTable = pFeatWorkspace.OpenFeatureClass("build") as ITable;
        pObjclass = pTable as IObjectClass;
        //QI到 IClassSchemEdit接口
        IClassSchemaEdit pOcSchemaEdit;
        pOcSchemaEdit = pObjclass as IClassSchemaEdit;
        //对这个对象类设置一个独占型的锁定
        ISchemaLock pSchLock ;
        pSchLock = pObjclass as ISchemaLock;
        pSchLock.ChangeSchemaLock(esriSchemaLock.esriExclusiveSchemaLock);
        //改变要素类的字段的别名
        pOcSchemaEdit.AlterFieldAliasName("Age", "BuildingAge");
        //释放锁定
        pSchLock.ChangeSchemaLock(esriSchemaLock.esriSharedSchemaLock);
    }
```

执行完后，使用 ArcCatalog 程序打开这个要素类查看其字段结构，可以发现“Age”字段的别名已经变成“BuildingAge”。

- **ISubtypes 接口**

ISubtypes 接口用于管理和查询对象类的子类、值域（Domain）和缺省值（default value）。值域是一个字段的值的范围，关于字段的值域特性，将在后面的章节中进行介绍。

每一个对象类都有一个缺省的子类代码，如果用户没有明确地设置一个缺省的子类或子类字段，ISubtypes::DefaultSubtypeCode 返回的子类代码值为 0。而 HasSubtype 将返回为 false，SubtypeFieldindex 的值为-1。

每一种类型的字段，除了 Shape 字段外，都可以设置缺省值，而与这个字段相联系的值域对象将会检查这个缺省值的有效性。当一条记录的某个字段没有被赋值就保存时，系统会自动将缺省值添加到这个字段中去。

8.4.3 FeatureClass 对象

FeatureClass（要素类）是可以存储空间数据的对象类，它是 ObjectClass 的扩展。要素类是一个空间实体（Spatial Entity）的集合，这些空间实体就是要素，

它被用于模拟离散的，具有空间属性的对象。

在一个要素类中的所有的要素都使用同样的字段结构，要素类与表、对象类的最大的区别在于它拥有一个几何字段，即 Shape 字段，用于存储要素的几何信息，这使得用户可以在地图上看到一个要素的形状和位置。

- **IFeatureClass 接口**

IFeatureClass 接口是程序员操作要素类时使用的主要接口，这个接口定义的属性和方法可以用于操作要素类。

IFeatureClass::ShapeFieldName 属性可以返回要素类的集合字段的名称，AreaField 和 LengthField 则可以返回要素类中用于存储要素面积和长度的字段对象。

IFeatureClass::ShapeType 返回的是 ShapeField 的类型，即要素类存储的是什么类型的几何形体对象。

IFeatureClass::FeatureType 可以让程序员知道要素类的类型，如简单要素类、标注类、注记类还是复杂要素类等。

IFeatureClass::FeatureClassID 返回的是要素类在地理数据库中的唯一标识符，它和 IObiectClass::ObiectClassID 属性是一样的。

IFeatureClass 接口也定义了许多方法用于操作要素类本身，这些方法和 IObjectClass 接口定义的方法都非常相似，如 Insert、Search 和 Update 方法可以分别得到一个“插入”、“查询”和“更新”功能的游标对象。

使用 IFeatureClass::Select 方法可以在一个要素类中得到要素选择集，而获得的要素选择集的数目和种类取决于使用的过滤器对象（QueryFilter）。

CreateFeature 方法可以新建一个新的要素对象，用户在新建一个 Feature 对象后，需要对它的值进行设定，然后使用 IFeature::Store 方法将新的要素存储到要素类中去。

CreateFeatureBuffer 方法则用于产生一个 FeatureBuffer 对象，它并不是一个真正的要素，而是一个暂时存在的对象，它需要使用“插入”游标，将新建的“缓冲要素”保存到要素类中去。

- **INetworkClass 接口**

如果要素类加入到一个几何网络中,那么将可以使用 INetworkClass 接口来对这些要素类进行管理，可以加入网络的要素有两种，即几何字段类型为点和线的要素类。

加入到网络中的要素分为四种：简单节点类型、复杂节点类型、简单边类型和复杂边类型。

使用 INetworkClass::GeometricNetwork 可以得到这个要素参与的几何网络对象；而使用 NetworkAncillaryRole 属性则可以得到这个要素在网络中担当的角色，如 Source（源）、Sink(汇）或普通角色。

8.4.4 字段集与字段

字段集是表的列（Column）属性的集合。一张表必须有一个字段集，而一个字段集中至少有一个字段。字段集是一个组件类，它可以独立于表存在，在地理数据库中，程序员需要使用到字段集和字段的地方相当频繁，如给表产生一个索引（Index）、排序、数据转换等。

使用 ArcCatalog 在 GeoDatabase 中新建一个 Table 或者一个 FeatureClass 时，必须设置它们的字段集（Fields）和字段，由于是组件对象类，可以便用 new 关键字来创建。在新建一个字段后，必须将它加入到一个字段集中供新建的表或要素类使用。

设置字段集的过程包括设定字段集中字段（Field）的类型（Type）和值域（Domain），缺省值，别名（Alias）和是否允许为空（Allow nulls）等属性。这些属性构成了要素的行为限制的条件，它们可以保证数据输入的有效性。

图 8-6 所示是一个在 ArcCatalog 中新建要素类的对话框。

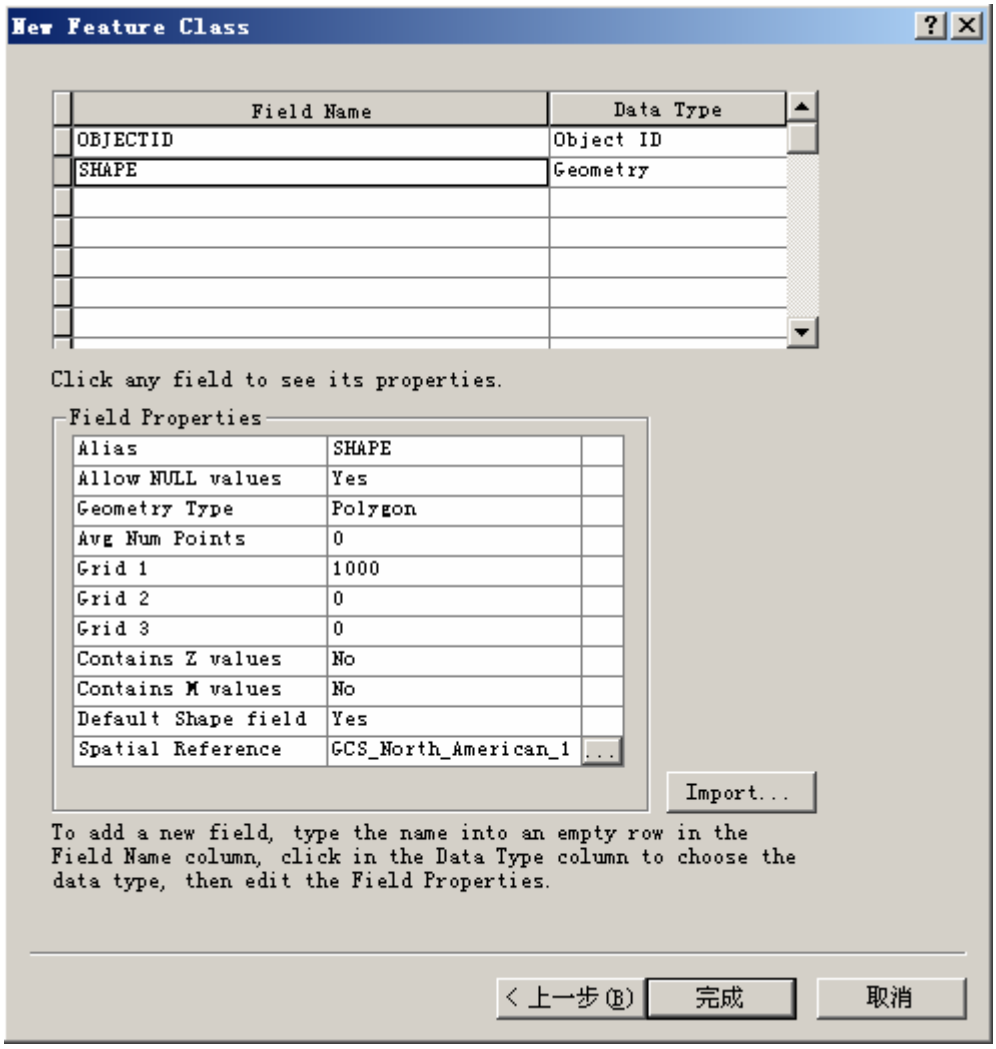


图 8-6 设计一个要素类的字段

尽管使用 ArcCatalog 程序可以很容易地设计表的字段集，但是程序员仍然需 ArcObjects 的角度来讨论字段集和字段的相关问题，如获取字段、修改字段等。

- **IFields 接口**

字段集是与一个表相关联的，如果需要获取一个表对象的字段集，可以使用 ITable::Fields 属性来实现。程序员使用 IFields 接口定义的属性和方法，可以非常容易地得到字段集中的某个字段，继而获得某个特定字段的值。

IFields 有两个属性，其中一个是 Field(index)，它通过传入不同的字段索引值可

以返回某个字段，如：

```
IField pField;

pField = pFields.get_Field(0);
```

而 IFields::FieldCount 属性可以返回字段集合的字段数目。

IFields 定义了两种方法，一种是 FindField，另一种是 FindFieldByAliasName。前者是用字段名作为字段的索引，返回一个字段的数字索引值，后者是使用别名来找出字段的索引号。

下面是获取一个名为“City_name”字段的代码段：

```
int i ;
IFields pFields;
IField pField;
//获得字段
pFields = pFeatClass.Fields;
//寻找到名称为”City__name的字段在字段集中的索引值
i = pFields.FindField("City_name");
//获得名称为” City_name”的字段对象
pField = pFields.get_Field(i);
```

- **IFieldsEdit 接口**

字段集里面的字段需要能够修改、更新和删除，实现这些要求的方法由 IFieldsEdit 接口定义，这个接口从 IFields 继承而来。

IFieldsEdit::AddField 方法，可以添加一个新的字段到字段集中。当需要修改一个表或者要素类的结构时，这个方法非常有用。DeleteAllFields 方法可以一次删除表中所有的字段。

下面是建立一个新表的代码，在建立过程中，主要考虑的就是字段的设置，即模式的设计。

```
private ITable CreateTable(IFeatureWorkspace pFeatWorkspace, string strName)
{
    ITable pTable ;
    IFields pFields;
    IField pField;
    IFieldEdit pFieldEdit;
    IFieldsEdit pFieldsEdit;
    //创建一个新的字段集合
```

```
            pFields = new FieldsClass();
            pFieldsEdit = pFields as IFieldsEdit;
            pFieldsEdit.FieldCount_2 = 2; //设定字段数目
            //产生一个OID字段
            pField = new FieldClass();
            pFieldEdit = pField as IFieldEdit;
            pFieldEdit.Name_2 = "OBJECTID";
            pFieldEdit.AliasName_2 = "FID";
            pFieldEdit.Type_2 = esriFieldType.esriFieldTypeOID;
            pFieldsEdit.set_Field(0, pField);
            //产生一个文本字段
            pField = new FieldClass();
            pFieldEdit = pField as IFieldEdit;
            pFieldEdit.Length_2 = 30;
            pFieldEdit.Name_2 = "Owner";
            pFieldEdit.Type_2 = esriFieldType.esriFieldTypeString;
            pFieldsEdit.set_Field(1, pField);
            pTable = pFeatWorkspace.CreateTable(strName, pFields, null, null, "");
            return pTable;

        }
```

需要提醒注意的是，使用 VB6 编写代码时，IFields 定义的属性是可读的，而 IFieldsEdit 定义的属性是可写的。但在.NET 中，IFieldsEdit 接口定义的属性能够直接进行读写了，但需要注意的是可写方法是以“_2”结尾的，如 Field 是可读的，而 Field_2 是可写的。

IFieldEdit 接口并不能够给一个已经存在的数据集加入一个字段，它定义的方法只能用于新建一个数据集的时候。给已经存在的 Table 或者 FeatureClass 添加一个字段需使用数据集本身的 AddField 方法。

下面是给一个已经存在的数据集添加字段的代码：

```
        private void AddFieldToFeatureClass()
        {
            //打开工作空间
            IWorkspaceFactory pWSF;
            pWSF = new AccessWorkspaceFactoryClass();
            IFeatureWorkspace pWS;
            pWS = pWSF.OpenFromFile(@"d:\arcgisdata\test.mdb", 0) as
IFeatureWorkspace;
```

```
IFeatureClass pFeatClass;
pFeatClass = pWS.OpenFeatureClass("build");
//新建一个字段
IField pField ;
pField = new FieldClass();
IFieldEdit pFieldEdit;
pFieldEdit = pField as IFieldEdit;
pFieldEdit.Type_2 = esriFieldType.esriFieldTypeDouble;
pFieldEdit.Name_2 = "salary";
//将字段添加进要素类
pFeatClass.AddField(pField);
}
```

Field（字段）是一个表的列的描述，字段组成了字段集。字段拥有很多属性，如名称和数据类型。读者可以从帮助中查阅到这些字段类型。

在一个简单要素类中，OID 和 Shape 字段是必需的，这两个字段的别名和几何类型可以改变，但是字段本身不能够被删除。要素类中存在一个几何字段，这个字段是描述要素类存储何种类型的要素，一个要素类中要么全部保存的是点要素，要么全部是线要素或多边形要素，不能将点、线和面要素混合存储在一个要素类中。

不同类型的要素类有一些特别的字段用于显示要素的状态，它们称为“要素追踪字段”。在线状要素类中，会有个 Shape_Length 字段来追踪某个要素的长度；在面状要素类中，有 Shape_Area 和 Shape_Length 两个字段（在 SDE 数据库里面这两个字段是 Shape.Area 和 Shape.Length），前者记录面要素的面积，后者记录周长；点和多点类型的要素没有这种字段。这种追踪要素的字段在使用 ArcCatalog 建立新表的时候不会显示在“新建表向导”对话框中，当然也就不能给它们设置缺省值、值域和删除它们。

- **IField 接口**

IField 接口是字段对象的主要接口，可以从这个接口中得到字段的主要属性，如名称、别名、字段的值域、字段类型等，下面的代码段是将一个要素类的非几何字段添加到一个 ComboBox 中：

```
int i;
IFields pFields;
IField pField;
pFields = pFeatureClass.Fields;
for( i = 0 ; i <= pFields.FieldCount - 1;i++)
{
```

```
            pField = pFields.get_Field(i);
            if (pField.Type != esriFieldType.esriFieldTypeGeometry )
            {
                comboBox1.Items.Add(pField.Name.ToString());
            }
        }
```

- **IFieldEdit 接口**

和 IFieldsEdit 接口一样，本来 IFieldEdit 接口中的属性都是可写的，用于设置一个新建的字段的属性，但是在新版本中的 IFieldEdit 中的属性有了改变，如 AliasName 可以用于读取字段的别名，而 AliasName_2 用于设置一个字段的值。

8.4.5 与字段相关的对象

新建一个表或者要素类的时候，对字段的设计是很复杂的，有时候为了更好地设置要素的行为和快速查找数据，需要设置表的索引、新建子类、设计几何字段和确定值域等。这些内容都和字段相关，本节将简要讲解这方面的内容

8.4.5.1 几何字段的设计

用户在给一个新的要素类设计字段的时候，必须考虑到它必不可少的几何（Geometry）字段，几何字段用于存储要素的几何形状，以便于确定要素的形状和它在地球上的位置。几何字段与一般的数值或者字符字段是不一样的，它没有缺省值。

设计一个几何字段的时候，有一个属性是必须设置的，即 IField::GeometryDef。它是一个 GeometryDef 对象，用于预定义一个几何字段的几何属性。

GeometryDef 对象的两个接口分别是 IGeometryDef 和 IGeometryDefEdit，由于两个接口定义的属性都是可读写的，如 GeometryType 为可读，GeometryType_2 为可写，因此只需要讨论一个接口就足够了。

IGeometryDef::GeometryType 属性用于设置几何字段的几何类型，如点、点集、多边形、环等；SpatialRefrence 属性则用于设置几何字段的空间参考。对于一个几何字段而言，这两个属性是必需的。

下面的代码用于设计一个几何字段：

```
private void DefineGeometryField(ISpatialReference pSpatRef)
```

```
        {
            //新建一个IGeometryDef对象
            IGeometryDef pGeoDef ;
            IGeometryDefEdit pGeoDefEdit;
            pGeoDef = new GeometryDefClass();
            pGeoDefEdit = pGeoDef as IGeometryDefEdit;
            pGeoDefEdit.AvgNumPoints_2 = 5;
            pGeoDefEdit.GeometryType_2 = (esriGeometryType)
esriFieldType.esriFieldTypeGeometry;
            pGeoDefEdit.GridCount_2 = 2;
          pGeoDefEdit.set_GridSize(0, 200);
          pGeoDefEdit.set_GridSize(1, 500);
          pGeoDefEdit.HasM_2 = false;
          pGeoDefEdit.HasZ_2 = true;
            pGeoDefEdit.SpatialReference_2 = pSpatRef;
            //产生一个几何字段
            IField pField;
            IFieldEdit pFieldEdit;
            pField = new FieldClass();
            //QI到 IFieldEdit 接口
            pFieldEdit = pField as IFieldEdit;
            pFieldEdit.Name_2 = "SHAPE";
          pFieldEdit.Type_2 = esriFieldType.esriFieldTypeGeometry;
          pFieldEdit.GeometryDef_2 = pGeoDef;
          pFieldEdit.IsNullable_2 = true;

        }
```

8.4.5.2 索引集和索引

索引有助于地理数据库快速查找和排序记录，它是基于表中的一个字段来建立的。在表中使用索引，就像在书中使用索引一样：查找某个数据时，会先在索引中找到数据的位置，这样会大大提高数据查询的速度。

索引提供指向存储在表的特定字段中的数据值的指针，然后根据指定的次序对这些指针进行排序。数据库使用索引与在书本中使用索引一样；它搜索索引以查找到特定值，然后跟随到指向包含该值的行的指针跳转到所需的内容。

在创建索引之前，必须确定所要使用的字段以及所要创建的索引类型。

索引集（Indexes）对象和字段集对象相似，它用于管理被设置为索引的字段，IIndexes 接口用于管理索引集对象，如 IIndexes::index 属性可以按照索引值得到一个表中的某个特定的索引。

索引对象代表了一个表的索引。一个要素类存在两种形式的索引：空间索引和属性索引。空间索引存在一个要素类的几何字段中，当一个要素类被创建的时候，系统会自动给这个要素类创建一个空间索引。对于 GeoDatabase 而言，它也会给要素类自动建立一个属性索引，这个索引是基于 OID 字段的，当然，用户也可以基于其他的字段建立新的索引。

Index 对象支持 IIndex 接口和 IIndexEdit 接口，使用这两个接口可以产生一个索引并设置它的属性，下面是一个建立索引的方法：

```
private void AddIndexToTable(ITable pTable,string strFieldName)
{
    //建立字段集
    IFields pFields;
    IFieldsEdit pFieldsEdit;
    IField pField;
    int i;
    pFields = new FieldsClass();
    pFieldsEdit = pFields as IFieldsEdit;
    pFieldsEdit.FieldCount_2 = 1;
    i = pTable.FindField(strFieldName);
    if(i == -1)
    {
        MessageBox.Show (strFieldName + "字段没有被找到");
        return;
    }
    pField = pTable.Fields.get_Field(i);
    pFieldsEdit.set_Field(0, pField);
    //产生索引对象
    IIndex pIndex;
    IIndexEdit pIndexEdit;
    pIndex = new IndexClass();
    pIndexEdit = pIndex as IIndexEdit;
    pIndexEdit.Fields_2 = pFields;
    pIndexEdit.Name_2 = "Idx_1";
    //向表中添加索引
    pTable.AddIndex(pIndex);
```

```
}
```

8.4.5.3 子类

子类（Subtypes）是一种用于给要素类或者对象类中的要素或对象分组的方式。通过给定不同子类的不同的缺省值和有效性规则，用户可以定制要素类中不同子类要素的行为。

当将一个要素类中的要素划分为不同的子类后，用户可以将这些子类当做单独的对象来使用。下面考虑一种情况：在一个要素数据集中，有一个 Land 要素类和一个 building 要素类，用户希望两个要素类之间的要素是满足严格的拓扑关系的，比如商业建筑必须在商业土地区内，工业建筑不能在居住土地区等。这时可以将一个 Land 的要素类分为三个子类——business、residential 和 industry，分别用于存储三个要素类；将 building 的要素也分为三种：business 要素、residential 要素和 industry 要素。将两个要素类各自分成子类后，可以在地理数据集中建立拓扑关系（拓扑需要在一个地理数据集 GeoDataset 中才能建立），并用这些拓扑规则来保证要素类的设计要求（关于地理数据库中的拓扑及其使用，可以参阅相关的 ESRI 资料）。

为了建立一个对象类的子类和设置子类字段的缺省值，程序员可以使用 ISubtypes 接口，下面的例子将一个“ Buildings”的要素类分为三个子类：Building，子类代码为 0；Commercial，代码值为 1；Residential，子类代码为 2。

```
        private void CreateSubtypes()
        {
            //打开工作空间
            IWorkspaceFactory pFact;
            pFact = new AccessWorkspaceFactoryClass();
            IWorkspace pWorkspace;
            pWorkspace = pFact.OpenFromFile(@"d:\Data\GeoDatabases\MyGDB.mdb",
0);
            IFeatureWorkspace pFeatws;
            pFeatws = pWorkspace as IFeatureWorkspace;
            IFeatureDataset pFeatds;
            pFeatds = pFeatws.OpenFeatureDataset("SNLandbase");
            IFeatureClassContainer pFeatclscont;
            pFeatclscont = pFeatds as IFeatureClassContainer;
            //得到数据集中的要素类 SNBuildings
            IFeatureClass pFeatcls ;
```

```
        pFeatcls = pFeatclscont.get_ClassByName("SNBuildings");
        // QI 到 ISubtypes 接口
        ISubtypes pSubType;
        pSubType = pFeatcls as ISubtypes;
        // 如果 SYMBOL 没有成为子字段，则将 SYMBOL 字段作为划分子字段的字段
        if (pSubType.SubtypeFieldName == "SYMBOL") return;
        pSubType.SubtypeFieldName = "SYMBOL";
        pSubType.AddSubtype(0, "Building");
        pSubType.AddSubtype(1, "Commercial");
        pSubType.AddSubtype(2, "Residential");
        pSubType.DefaultSubtypeCode = 0;
        //给每个子类设置缺省值
        pSubType.set_DefaultValue(0, "TYPE_", "BLD");
        pSubType.set_DefaultValue(1, "TYPE_","COM");
        pSubType.set_DefaultValue(2, "TYPE_", "RES");
    }
```

8.4.5.4 值域

Domain （值域）是一个抽象类，它可以用于限制一个对象类或一个要素类的某个字段的允许值。即当用户给一个对象类的某个特定字段设置值域后，这个字段能够被赋予的值必须在这个值域的限制内。值域可以用于探测一个字段值的有效性，它是子类的基础。

值域与一个工作空间相关，因而一个值域可以给多个字段使用。在 ArcObjects 中可以通过 IWorkspaceDomains::AddDomain 来添加一个值域对象。

RangeDomain 和 CodedValueDomain 是 Domain 的子类，它们是组件类，并且都支持 IDomain 接口。IDomain 接口定义了所有的值域对象的最一般的方法和属性。除了 IDomain::Type 外，其余所有的属性都是可读写的。当一个新的值域和某个字段被关联的时候，Name 和 FieldType 两个属性是必需的。

Rangedomain 是范围值域对象，它确定某个字段的值在一个范围内，显然地，这个对象规定的值只能是数值型或者日期型，如考试的分数在 0 到 100 的范围内，日期在 2005-1-1 到 2006-1-1 的范围内。IRangeDomain 接口规定了范围的最大和最小值属性。

CodedValueDomain 是代码值域对象，它是一个“值-名称”对象的集合，它用于确定一个字段所有可能的值。如用户将土地分为三种使用类型(“工业土地”、“农业土

地”、“居住土地”)后，这个土地要素类中的类型字段可以出现的值就是用户规定的三种，而不能出现别的。代码值域一般都是字符类型。ICodedValueDomain 规定了代码值域对象的特定的属性和方法，如 AddCode 方法可以用于添加一个“值-名称”对象，而使用 Name 和 Value 可以通过一个索引值来得到一个“值-名称”对象。

下面是一个添加值域对象的例子：

```
private void AddDomain()
{
    IWorkspaceFactory pWorkspaceFactory;
    pWorkspaceFactory = new SdeWorkspaceFactoryClass();
    IWorkspace pWorkspace;
    pWorkspace = pWorkspaceFactory.OpenFromFile(@"d:\MyConns\gdb.sde", 0);
    //QI到IWorkspaceDomains接口
    IWorkspaceDomains pWorkspaceDomains;
    pWorkspaceDomains = pWorkspace as IWorkspaceDomains;
    //产生一个RangeDomain对象
    IRangeDomain pRange;
    pRange = new RangeDomainClass();
    pRange.MinValue = 9.0;
    pRange.MaxValue = 600000.0;
    //QI到IDomain，设置其他的属性
    IDomain pDomain;
    pDomain = pRange as IDomain;
    pDomain.Name = "Area constraint";
    pDomain.FieldType = esriFieldType.esriFieldTypeDouble;
    pDomain.Description = "Constrains the area Of buildings";
    pDomain.MergePolicy = esriMergePolicyType.esriMPTAreaWeighted;
    pDomain.SplitPolicy = esriSplitPolicyType.esriSPTGeometryRatio;
    //将值域对象添加到工作空间
    pWorkspaceDomains.AddDomain(pDomain);
    //产生一个coded value domain对象
    ICodedValueDomain pCodeValue;
    pCodeValue = new CodedValueDomainClass();
pCodeValue.AddCode("RES", "Residential");
pCodeValue.AddCode("COM", "Commercial");
pCodeValue.AddCode("IND", "Industrial");
pCodeValue.AddCode("BLD", "Building");
    //QI到IDomain，设置属性
```

```
            IDomain pDomainCodeValue;
            pDomainCodeValue = pCodeValue as IDomain;
        pDomainCodeValue.FieldType = esriFieldType.esriFieldTypeString;
        pDomainCodeValue.Name = "Building types";
        pDomainCodeValue.Description = "Valid building type codes";
        pDomainCodeValue.MergePolicy = esriMergePolicyType.esriMPTDefaultValue;
        pDomainCodeValue.SplitPolicy = esriSplitPolicyType.esriSPTDuplicate;
            //添加到工作空间
            pWorkspaceDomains.AddDomain(pDomainCodeValue);
    }
```

上例中的 MergePolicy 和 SplitPolicy 属性是指当要素被合并或者切割的时候，其值在变化时应该遵守的准则。

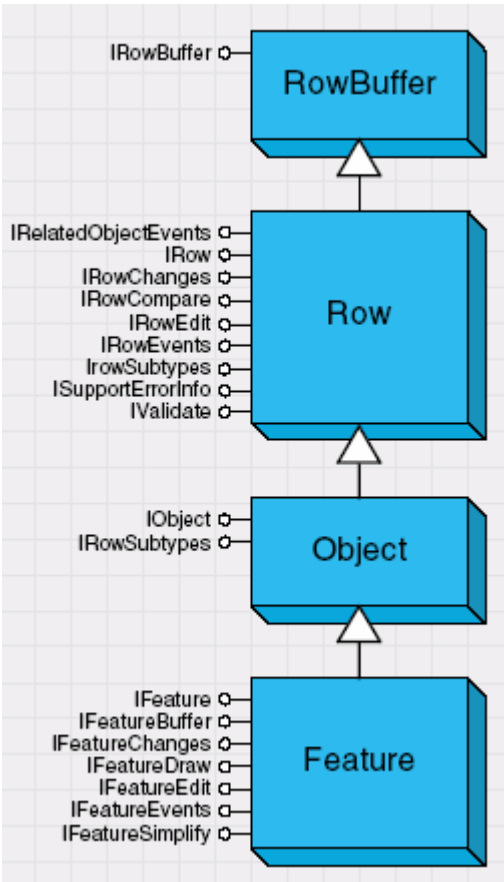


图 8-7　Row、Object 和 feature 的关系

8.5 ROW，Object 和 Feature 对象

ArcGIS 中使用的地理数据按照 GeoDatabase 的观点来看都是以二维表的形式存在的，而对于这些表的操作是不用区分数据的类型和来源的。从表、对象类和要素类的关系中，读者可以知道其实它们全部都是二维表结构，其关系不过是一级级在前者的基础上进行扩展的结果。

表、对象类和要素类是一个数据集，作为它们的组成部分，Row(行)、Object(对象)和 Feature(要素)就是最小的数据单元。它们的物理组成就是地理数据库中的一行记录。

在 ArcObjects 开发中，插入、更新和删除数据库中的一条记录是非常普遍的操作。本节将对这三

个对象进行深入的介绍和分析。如图 8-7 所示，它是这三个对象的继承结构图。

8.5.1 RowBuffer 和 Row 对象

RowBuffer(缓冲行)对象可以以一个行(Row)对象的状态出现，但是它并不是一个对象实体，即它是临时存在内存中的对象。这个对象主要被使用在一个插入游标的 InsertRow 方法中，作为一个参数使用。用户可以使用 ITable::CreateRowBuffer 方法来获得一个缓冲行对象。

IRowBuffer 接口用于获得 RowBuffer 对象的属性，如 Fields 用于获得缓冲行对象的字段集，而 Value 属性可以通过字段的索引值得到某个字段的值，通常用于被设置字段值。

Row 是一个组件类对象，它代表了一个表对象中的一行记录。程序员可以根据一个表的游标来获得 Row 对象，如 ICursor::Row，当然，用户可以根据它的 OID 值来获得，如 ITable::GetRow。 ArcObjects 中一般都使用 ITable::CreateRow 来产生一个新的 Row 对象，在得到这个对象后，需要给这个对象不同字段的值进行设置，Row 对象也有字段集，而且这个字段集和它所属于的表的字段集是一样的，因而用户还需要通过 IFields 对象的各种方法和属性来获得某个具体的字段，然后赋值给它，这些方法是多种多样的，非常灵活. ”

Row 对象主要支持 IRow 接口，这个接口从 IRowBuffer 继承而来，除继承了 Fields 和 Value 两个属性外，还可以通过 IRow::HasOID 来探知 Row 对象是否有 OID 属性，它可以判断这个 Row 的表是否被注册到了地理数据库中。

IRow::OID 属性可以直接得到特定 Row 对象的 OID 值，这是 Row 对象在表中区别于其他 Row 对象的标记。在地理数据库自动建立索引的时候，系统会将 OID 字段设置为索引。这是因为这个字段是 Row 对象的唯一身份，不会出现重复；IRow::Table 属性，则可以返回 row 对象的容器--表。

一旦程序员修改了某个 Row 对象的值，就需要使用 Store 方法来将它保存到表中去，在这个方法被使用和执行的时候，系统会触发 IRowEvents::OnChanged 方法；如果是产生了一个新的 Row 对象并保存时，则会触发一个 Onnew 方法。

下面给出一个关于新建、更新和删除 Row 对象代码段：

```
IRow pRow;
int i ;
i = pTable.FindField("Name");
//添加一条记录
pRow = pTable.CreateRow();
```

```
pRow.set_Value(i, "esri");
pRow.Store();
//更新一条记录
pRow.set_Value(i,"esrichina");
pRow.Store();
//删除一条记录
pRow.Delete();
```

8.5.2 Object 和 Feature 对象

ObjectClass 对象是 Object(对象)的集合，后者代表了一个具有属性和行为的实体对象。在地理数据库中，对象类其实也是一张表，所以 Object 对象也就是表中的一条记录。

IObject 接口是 Objeet 对象的主要接口，它和 IRow 接口非常相似，且是从后者继承的。

IObject::Class 属性可以获得对象的容器--对象类。

Feature(要素)是在 ArcObjects 二次开发中最常使用的对象。它是要素类中的一条记录，是从对象派生出来的既具有属性和行为，而且还能保存空间数据的对象。要素中的几何形体对象定义了要素类的类型。一个要素中可能保存的几何形体有点、点集、多边形和多义线。

通常用户处理的都是简单的要素对象，但在某些复杂环境中，要素还可以参与拓扑、网络和标注等运算。

Feature 对象的主要接口是 IFeature，它定义了要素对象特有的属性，如 Extent 可以返回一个要素对象的包络线，它用于显示要素的空间范围；FeatureType 可以让用户知道要素的类型；Shape 属性将返回要素的几何形体对象；而通过 ShapeCopy 属性，可以得到这个几何形体对象的一份拷贝，它一般用于修改一个要素集中的几何属性。

IFeatureBuffer 接口定义的属性和方法可以让程序员产生并得到一个 FeatureBuffer(缓冲要素)对象的各种属性，下面介绍另一种添加要素进入要素类的方法，它主要使用 IFeatureBuffer 和 IFeatureCursor 接口，下面是这个方法的代码段：

```
        private void AddFeatureToFeatureClass(IFeatureClass pFeatClass, IGeometry
pGeom)
        {
```

```
                //产生一个要素游标和要素缓冲接口
                IFeatureCursor pFeatCur;
                IFeatureBuffer pFeatBuf;
                //打开要素游标和要素缓冲，pFeatcls的一个要素类
                pFeatCur = pFeatClass.Insert(true);
                pFeatBuf = pFeatClass.CreateFeatureBuffer();
                //得到字段
                IFields pFlds;
                IField pFld ;
                int i ;
                IPolygon pPolygon;
                //pGeom对象是一个需要传入的几何形体对象
                pPolygon = pGeom as IPolygon;
                //开始对要素类的字段进行遍历，并设置属性值
                pFlds = pFeatClass.Fields;
                for (i = 1 ; i <= pFlds.FieldCount - 1; i++)
                {
                    pFld = pFlds.get_Field(i);
                    //如果是几何字段的话，将pGeom传入
                    if (pFld.Type == esriFieldType.esriFieldTypeGeometry)
                    {
                        pFeatBuf.set_Value(i,pGeom);
                    }
                    else
                    {
                        //传入各种试验值
                        if (pFld.Type == esriFieldType.esriFieldTypeInteger)
pFeatBuf.set_Value(i, 0);
                        if (pFld.Type == esriFieldType.esriFieldTypeDouble )
pFeatBuf.set_Value(i,0.0);
                        if (pFld.Type == esriFieldType.esriFieldTypeSmallInteger)
pFeatBuf.set_Value(i, 0);
                        if (pFld.Type == esriFieldType.esriFieldTypeString)
pFeatBuf.set_Value(i,"noname");
                    }
                }
                //将要素缓冲对象通过插入游标保存进要素类中
                pFeatCur.InsertFeature(pFeatBuf);
```

```
}
```

在上面的例子中演示了一个试验性的代码，首先需要得到一个几何对象，然后将这个几何对象保存进要素类的 Shape 字段中去，如果要素类存在其他类型的字段，则依据这个字段的类型设置一些默认的数值或字符串。

IFeatureEdit 接口定义的方法可以用于编辑单个的要素或者多个要素组成的要素集。

这个接口定义的 MoveSet、RotateSet 和 IRow::DeleteSet 方法既可以使用在单个要素上，也可以使用在多个要素组成的要素集上。

IFeatureEdit::Split 和 IFeatureEdit::SplitAttributes 操作是使用在一个单一的要素上，前者可以通过一个或多个点分割多义线，或者使用多义线分割多边形。在分割后，新的要素会产生，而旧的要素会被删除掉，这种操作就是分割几何形体。

自然地，在分割了一个或者多个要素时，用户需要通过一定的机制来分割要素的属性字段中的值，SplitAttributes 方法将会执行这个过程，而属性分割机制是由字段的值域来决定的。但程序员并不需要在 Split 要素后执行 SplitAttributes 方法，因为后者会自动执行。

下面是一个分割要素的代码：

SplitPolygonFeatures.cs

```
using System;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class SplitPolygonFeatures: BaseTool
    {
        IMap pMap;
        IActiveView pActiveView;
        IHookHelper m_hookHelper = new HookHelperClass();
        public SplitPolygonFeatures()
        {
```

```
            base.m_caption = "SplitPolygonFeatures";
            base.m_message = "SplitPolygonFeatures";
            base.m_toolTip = "SplitPolygonFeatures";
            base.m_category = "EditGeometry";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));

        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            IFeatureLayer pFeatureLayer = pMap.get_Layer(1) as IFeatureLayer;
            IScreenDisplay screenDisplay = pActiveView.ScreenDisplay;
            ISimpleLineSymbol lineSymbol = new SimpleLineSymbolClass();
            IRgbColor rgbColor = new RgbColorClass();
            rgbColor.Red = 255;
            lineSymbol.Color = rgbColor;
            IRubberBand rubberLine = new RubberLineClass();
            IPolyline pPolyline = (IPolyline)rubberLine.TrackNew(screenDisplay,
(ISymbol)lineSymbol);
            screenDisplay.StartDrawing(screenDisplay.hDC,
(short)esriScreenCache.esriNoScreenCache);
            screenDisplay.SetSymbol((ISymbol)lineSymbol);
            screenDisplay.DrawPolyline(pPolyline);
            screenDisplay.FinishDrawing();
            IFeatureSelection pFeatSel = (IFeatureSelection)pFeatureLayer;
    pFeatSel.SelectFeatures(null,esriSelectionResultEnum.esriSelectionResultNew,false);
            ISelectionSet pSelectionSet = pFeatSel.SelectionSet;
            SplitFeatures(pSelectionSet,pPolyline);
```

```
        }

        private void SplitFeatures(ISelectionSet pSelectionSet, IPolyline pPolyLine)
        {
            //使用空间过滤器来获得将与polyline进行分割的要素集
            IFeatureCursor pFeatCursor;
            ICursor pCursor;
            ISpatialFilter pSpatialFilter;
            pSpatialFilter = new SpatialFilterClass();
            pSpatialFilter.Geometry = pPolyLine;
            pSpatialFilter.SpatialRel = esriSpatialRelEnum.esriSpatialRelCrosses;
            pSelectionSet.Search(pSpatialFilter, true, out pCursor);
            pFeatCursor = pCursor as IFeatureCursor;
            //清理将被分割的要素
            ITopologicalOperator pTopoOpo;
            pTopoOpo = pPolyLine as ITopologicalOperator;
            pTopoOpo.Simplify();
            //遍历要素来分割它们
            IFeature pFeature;
            pFeature = pFeatCursor.NextFeature();
            while (pFeature != null)
            {
                IFeatureEdit pFeatureEdit ;
                pFeatureEdit = pFeature as IFeatureEdit;
                pFeatureEdit.Split(pPolyLine);
                pFeature = pFeatCursor.NextFeature();
            }
        }
    }
}
```

需要注意的是，如果对要素或要素集的编辑牵扯拓扑和几何网络，那么这些编辑就必须放在一个编辑流程内进行。

8.5.3 点选法获取要素

ArcObjects 中选择一个要素或者一个要素集(FeatureSelection)的方法很多，如 IMap::SelectByShape、ILayer::Search、IFeatureSelection::SelectFeature 等方法

都可以办到。在前面的章节中详细讨论了如何使用普通属性和空间属性计算的方法得到要素，介绍了“线选择”、“多边形选择”和“圆形选择”要素的任务。这一节将介绍如何使用点选的方法。由于 ArcObjects 中没有提供点选的函数，它需要使用其他的思路。

下面的代码片段是使用点选的方式选择点、多义线和多边形要素，其原理是将用户点击在 Map 视图上的点对象进行缓冲区计算，得到一个具有形状的圆形几何形体，然后使用这个圆形对象的包络线和要选择的要素类进行空间过滤运算，这样可以得到被选择的要素，并将它们加入到选择集中去。

下面是一个点选法的代码：

SelctionBasedonPoint.cs

```
using System;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class SelctionBasedonPoint: BaseTool
    {
        IMap pMap;
        IActiveView pActiveView;
        IHookHelper m_hookHelper = new HookHelperClass();
        public SelctionBasedonPoint()
        {
            base.m_caption = "SelctionBasedonPoint";
            base.m_message = "SelctionBasedonPoint";
            base.m_toolTip = "SelctionBasedonPoint";
            base.m_category = "SpatialQuery";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }
```

```
        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            IFeatureLayer pFeatureLayer = pMap.get_Layer(1) as IFeatureLayer;
            IFeatureClass pFeatureClass = pFeatureLayer.FeatureClass;

            //pPoint是点击的点
            IPoint pPoint =
pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X, Y);

            //对点对象做缓冲区运算
            ITopologicalOperator pTopo;
            pTopo = pPoint as ITopologicalOperator;
            IGeometry pBuffer;
            pBuffer = pTopo.Buffer(4);
            IGeometry pGeometry = pBuffer.Envelope;
            //新建一个空间过滤器
            ISpatialFilter pSpatialFilter;
            pSpatialFilter = new SpatialFilterClass();
            pSpatialFilter.Geometry = pGeometry;
            //依据被选择的要素类的类型不同，设置不同的空间过滤关系
            switch( pFeatureClass.ShapeType )
            {
                case esriGeometryType.esriGeometryPoint:
                    pSpatialFilter.SpatialRel =
esriSpatialRelEnum.esriSpatialRelContains;
                    break;
                case esriGeometryType.esriGeometryPolyline:
                    pSpatialFilter.SpatialRel =
esriSpatialRelEnum.esriSpatialRelCrosses;
                    break;
```

```
                    case esriGeometryType.esriGeometryPolygon:
                        pSpatialFilter.SpatialRel =
esriSpatialRelEnum.esriSpatialRelIntersects;
                        break;
                }
                IFeatureSelection pFeatureSelection;
                pFeatureSelection = pFeatureLayer as IFeatureSelection;

    pFeatureSelection.SelectFeatures(pSpatialFilter,esriSelectionResultEnum.esriSelectionR
esultNew, false) ;
                ISelectionSet pFeatSet;
                pFeatSet = pFeatureSelection.SelectionSet;
                ICursor pCursor;
                pFeatSet.Search(null, true, out pCursor) ;
                IFeatureCursor pFeatureCursor = pCursor as IFeatureCursor;
                IFeature pFeature = pFeatureCursor.NextFeature();

                //遍历所有符合要求的要素，使用IMap::SelectFeature方法将它们加到图层的要素
选择集中()
                while(pFeature != null)
                {
                    //将在当前图层中获得的要素放入当前图层的要素选择集中
                    pMap.SelectFeature(pFeatureLayer, pFeature);
                    pFeature = pFeatureCursor.NextFeature();
                }
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeoSelection, null,
null);
            }
        }
}
```

缓冲区运算是 GIS 中最常见的一种几何运算方式。关于这个主题，读者可以参阅 Geometry 章节中的相关内容。

8.5.4 更新要素

在 GIS 数据库中使用 GeoDatabase 模型更新一个要素的数据，就如同在普通数据库中更新一行记录一样方便。ArcObjects 提供了多种方法：如 IFeature::Store

和 IFeature::Update 都可以达到目的，但前一种方法一次只能够改变一个要素的值，而后者可以一次更新多个要素的值。

下面是更新要素属性值的代码段：

```
private void UpdateFeatures()
{
    IFeatureClass pFeatcls;
    IFeatureLayer pFeatLayer;
    //得到要更新的要素类pFeatcls
    IMap pMap = axMapControl1.Map;
    pFeatLayer = pMap.get_Layer(0)as IFeatureLayer;
    pFeatcls = pFeatLayer.FeatureClass;
    //产生一个过滤器
    IQueryFilter pQueryFilter;
    pQueryFilter = new QueryFilterClass();
  pQueryFilter.WhereClause = "subtype='COM'";
  pQueryFilter.SubFields = "FID,Type";
    int iFld ;
    IFields pFlds;
    IFeatureCursor pFeatCur;
    IFeature pFeat;
    //获得指向被选择要素的游标
    pFeatCur = pFeatcls.Update(pQueryFilter, false);
    pFlds = pFeatcls.Fields;
    iFld = pFlds.FindField("Type");
    pFeat = pFeatCur.NextFeature();
    while ( pFeat != null)
    {
        pFeat.set_Value(iFld, "ABC");
        pFeatCur.UpdateFeature(pFeat);
        pFeat = pFeatCur.NextFeature();
    }

}
```

上面的方法通过使用一个更新型游标，将选择出来的要素 COM 子类的 Type 字段的值全部更新为“ABC”。

8.6 关系与关系类

熟悉例如 SQLServer 和 Oracle 等关系数据库的读者对“关系”这个术语一定不会陌生，关系数据库中的关系是通过两张表的主键和外键来实现的。在地理数据库中，所谓关系是指两个对象之间的某种关联；而关系类则是一个或者多个关系的集合。用户可以将关系看成是关系类表中的一条记录实体。

一个关系中的相关两个对象，一个是 OriginObject(起始对象)，另外一个则是 DestinationObject(目标对象)。如有两个要素类--房地产要素类和房地产附加属性类，用户可以在这两个类之间建立一个简单关系，而前者设置为起始对象，后者设置为目标对象。这些设置读者可以自己使用 ArcCatalog 软件亲自实践验证。

地理数据库中的关系作为两个对象类之间的某种关联，它有三种形式：一对一、一对多和多对多，这和一般的关系数据库是一致的。这三种关系类型描述了自然界对象之间的普遍联系，如一名学生与学号之间是一对一的关系，一名学生只有一个学号，而一个学号也只能给一名学生使用；班级和学生之间是一对多的关系，一个班级可以有多名学生；班级和老师之间则是多对多的关系，一个班级可以有多位老师任教，而一位老师也可以在多个班级任教。

关系(Relationship)是一个抽象类，它有两种子类对象，一个是 SimpleRelationship，它代表了一对一或者一对多的关系结构，这种关系是通过一个外键来关联两个对象类的；第二种子类对象是 AttributedRelationship(属性关系)，它代表了一种多对多的关系结构，由于多对多关系的复杂性，使得 ArcObjects 在描述它时往往需要使用额外的信息，这个关系通常存储在一张二维表中，如图 8-8 所示。

这两个关系类都实现了 IRelationship 接口，这个接口提供了获得一个关系对象信息的几个只读属性，如 DestinationObject、OriginObject 和 RelationshipClass，程序员可以通过这些属性来得到一个关系对象的源对象、目的对象以及保存这个关系对象的关系类。

关系类(RelationshipClass)代表了两个对象类之间的联系，这两个对象类，同样一个作为起始对象，一个作为目标对象，而关系类就是这两个对象类中对象的关系集合。关系类分为两种：一种是 RelationshipClass(为了避免和关系类这个通称混淆，本文将 RelationshipClass 翻译为简单关系类)；另一种是 AttributedRelationshipClass(属性关系类)。前者保存的是简单关系，后者保存的是属性关系。用户使用 Access 程序打开一个 Personal GeoDatabase 的物理数据库，就可以看到这两种关系类的区别，在数据库中用户会发现一个名为 GDB_RelClasses 的表，它中间存储着工作空间中所有的关系类，无论是简单关系

类还是属性关系类，而属性关系类除了作为一条记录保存在这个表中外，还有一个单独的二维表存在，如果这个属性关系类名为 test，那在数据库中就存在一个 test 表。

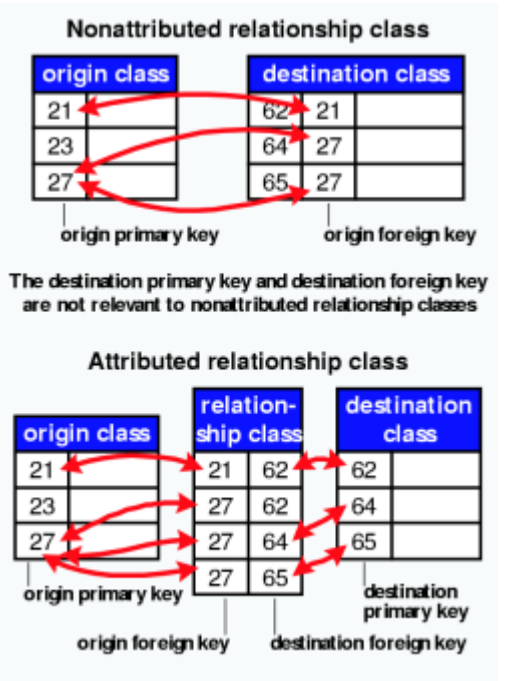


图 8-8 两种关系类

关系类同时也支持 IDataset 接口，但是它并不支持 IClass 接口(除非这个关系类中的关系都是属性关系)，这是因为非属性关系类(即简单关系类)并不包含任何它自己拥有的字段，因为它没有单独用于描述关系的表，而拥有这种表的只有属性关系类。

IRelationshipClass 接口提供了操作一个关系类的属性和方法，下面对它们进行介绍。

OriginPrimaryKey 是起始对象类的主键，熟悉数据库的读者都知道，主键是标识一张表中记录唯一性的字段，即它的值不可以出现重复；OriginForeignKey 是目标对象类中的外键，它是用于和起始对象类中主键相关联的字段，一张表的主键往往是其他表的外键。

由于属性关系类在工作空间中存在一张单独的存储信息的二维表，因此在起始对象类、属性关系类和目标对象类三者之间的联系一共是四个字段参与的(如图 8-9)，起始对象类中的唯一标识字段为 OriginPrimaryKey，属性关系类中有两个字段，一个与起始对象类关联，它称为 OriginForeignKey，一个与目标对象类关联，

称为 DestinationForeignKey，而目标对象的主键则为 DestinationPrimaryKey。

IRelationshipClass::IsAttributed 属性可以用于返回一个要素类是否有属性信息，而 IRelationshipClass::GetRelationship 方法对于获得关系属性非常有用。

AttributedRelationshipClass 是一种特殊的关系类，在物理实现上它是一张二维表；对于关系类而言，这些关系都是使用外键值保存在对象本身中的。如果想知道一个关系类是否为属性关系类，有个简单的方法，即：

```
if ( pRelClass is ITable)
  MessageBox.Show("Attributed Relationship Class");
```

读者应该很清楚为什么可以使用这个方法了。因为只有属性关系类才有一张表在工作空间中存在，而简单关系类却是“虚”的。

用户可以使用 IRelationshipClassContainer 或者 IFeatureWorkspace 接口来产生一个关系类，如 IFeatureWorkspace::CreateRelationshipClass 方法可以新建一个新的独立的关系类(即这个关系类不放在一个要素数据集中)。

```
public IRelationshipClass CreateRelationshipClass (
    string relClassName,
    IObjectClass OriginClass,
    IObjectClass DestinationClass,
    string forwardLabel,
    string backwardLabel,
    esriRelCardinality Cardinality,
    esriRelNotification Notification,
    bool IsComposite,
    bool IsAttributed,
    IFields relAttrFields,
    string OriginPrimaryKey,
    string destPrimaryKey,
    string OriginForeignKey,
    string destForeignKey

    );
```

这个方法的参数众多，如 cardinality 是关系的类型，如果它是属性关系类，则是多对多；relAttrFields 是一个可选的参数，对于简单关系类而言通常设置为 0 或者 null。如果是多对多的关系的属性关系类，这个方法要求设置所有的四个字段，即 OriginPrimaryKey、DestinationPrimaryKey、OriginForeignKey 和 DestinationForeignKey；如果关系类型是一对一或一对多的简单关系类，则只需要设置 OriginPrimaryKey 和 OriginForeignKey 两个属性就可以了。

8.7 数据转换与装载

数据的转换和数据装载在地理数据库中是经常进行的两种操作，在 ArcObjects 中这两种操作涉及两个核心对象：一个是 FeatureDataConverter，它可以让程序员在 GeoDatabase、Shapefiles 和 Coverage 中自由转换数据，它支持的数据类型非常广泛；另一个是 0bjectLoader，它可以向一个已经存在的表或者要素类中装载数据。

在数据的转换与装载操作中，还有许多其他的对象和接口参与，如 IFieldChecker 对象可以检查字段名中潜在的问题，IEnumInvalidObject 则可以用于检查在转换过程中被拒绝的数据。

FeatureDataConverter 对象支持 IFeatureDataConverter 接口，它提供了进行数据转换的三种方法 ConvertFeatureClass、ConvertFeatureDataset 和 ConvertTable，它们分别可以将要素类、要素数据集和表转换为 GeoDatabase 中的要素类、要素数据集和表。

下面是一个要素类转换 shapefile 的方法，输入的参数是需要转换的要素类 pFeatureClass 和输出的工作空间 pOutWorkspace：

```
        private void ExportToShapefile(IFeatureClass pFeatureClass, IWorkspace
outWorkspace)
        {
            IDataset inDataSet = pFeatureClass as IDataset;
            IFeatureClassName inFCName = inDataSet.FullName as IFeatureClassName;
            IWorkspace inWorkspace = inDataSet.Workspace;

            IDataset outDataSet = outWorkspace as IDataset;
            IWorkspaceName outWorkspaceName = outDataSet.FullName as
IWorkspaceName;

            IFeatureClassName outFCName = new FeatureClassNameClass();
            IDatasetName dataSetName = outFCName as IDatasetName;
            dataSetName.WorkspaceName = outWorkspaceName;
            dataSetName.Name = pFeatureClass.AliasName.ToString();
            IFieldChecker fieldChecker = new FieldCheckerClass();
            fieldChecker.InputWorkspace = inWorkspace;
            fieldChecker.ValidateWorkspace = outWorkspace;
            IFields fields = pFeatureClass.Fields;
```

```
            IFields outFields = null;
            IEnumFieldError enumFieldError = null;
            fieldChecker.Validate(fields,out enumFieldError,out outFields);
            IFeatureDataConverter featureDataConverter = new
FeatureDataConverterClass();
    featureDataConverter.ConvertFeatureClass(inFCName,null,null,outFCName,null,outField
s,"",100,0);
        }
```

测试代码：

```
        private void mnuExportToShapefile_Click(object sender, System.EventArgs e)
        {
            IFeatureLayer pFeatureLayer;
            IFeatureClass pFeatureClass;
            IMap pMap = axMapControl1.Map;
            pFeatureLayer = pMap.get_Layer(1)as IFeatureLayer;
            pFeatureClass = pFeatureLayer.FeatureClass;
            IWorkspace pOutWorkspace;
            IWorkspaceFactory pFact;
            pFact = new ShapefileWorkspaceFactoryClass();
            pOutWorkspace = pFact.OpenFromFile(@"d:\temp\states", 0);
            ExportToShapefile(pFeatureClass,pOutWorkspace);
        }
```

使用 ConvertFeatureClass 和 ConvertFeatureDataset 方法的时候，如果将 OutputGeometryDef 参数设置为 null，则空间参考会从输入的要素类、要素数据集或者目标要素集中获得，缺省的空间索引数据也会产生。而 InputQueryFilter 参数则可以在数据转换的时候进行数据过滤，它需要使用 IQueryFilter 对象。

FeatureDataConverter 对象还支持 IFeatureDataConverter2 接口，它可以将一个要素类的要素选择集进行转换。

ObjectLoader 对象用于向一个已经存在要素类中添加要素或者向已经存在的表中添加行对象，与数据转换过程不一样，装载操作是在一个编辑流程(edit session)内进行的，如果数据装载失败的话，会产生一个 EnumInvalidObject 枚举变量，显示无效的原因。

ObjectLoader 对象实现了 IObjectLoader 接口，它只有一种方法即 LoadObjects，它需要的参数众多，如：

```
        public ISelectionSet LoadObjects (
```

```
IEditor Editor,
ITable inputTable,
IQueryFilter inputQueryFilter,
ITable outputTable,
IFields OutputFields,
bool setSubType,
int subTypeCode,
bool snapToFeatures,
bool applyValidation,
int flushInterval,
ref IEnumInvalidObject ppInvalidObjectEnum

);
```

下面是一个装载数据的过程代码，它将 D 盘 Data 文件夹中 USA 文件夹的 states 文件装载到 D 盘 Data 文件夹 statescities 数据库的 statescopy 要素类中，statescopy 和 states 的表结构是一致的：

```
private void LoadObject()
{
    //设置输出要素类
    IFeatureClass pOutFClass;
    IWorkspaceFactory pOutWSF;
    pOutWSF = new AccessWorkspaceFactoryClass();
    IFeatureWorkspace pOutFeatWS;
    pOutFeatWS = pOutWSF.OpenFromFile(@"D:\data\statescities.mdb", 0) as
IFeatureWorkspace;
    pOutFClass = pOutFeatWS.OpenFeatureClass("statescopy");
    //设置输入的要素类，即数据源
    IFeatureClass pInFClass;
    IWorkspaceFactory pWSF;
    pWSF = new ShapefileWorkspaceFactoryClass();
    IFeatureWorkspace pFeatWS;
    pFeatWS = pWSF.OpenFromFile(@"D:\data\usa", 0) as IFeatureWorkspace;
    pInFClass = pFeatWS.OpenFeatureClass("states");
    //设置输入数据的子集，这要通过一个过滤器对象来完成
    IQueryFilter pQueryFilter;
    pQueryFilter = new QueryFilterClass();
    pQueryFilter.SubFields = "SHAPE,STATE_NAME,STATE_FIPS,AREA";
    pQueryFilter.WhereClause = "STATE_NAME = 'California' OR STATE_NAME =
```

```
'Oregon'";
                //输出字段的参数需要和输入过滤器对象定义的字段匹配，这是为了保证Schema
一致
                IFields pAllFields;
                pAllFields = pOutFClass.Fields;
                IFields pOutFields ;
                pOutFields = new FieldsClass();
                IFieldsEdit pOutFieldsEdit;
                pOutFieldsEdit = pOutFields as IFieldsEdit;
                //得到过滤器的子字段，即上面设置的四个,它们是一个数组，遍历每个对象然后
添加到输出字段()
                string[] sSubFields;
                sSubFields = pQueryFilter.SubFields.Split( ",".ToCharArray());
                int i, j ;
                for (j = sSubFields.GetLowerBound(0); j<= sSubFields.GetUpperBound(0);j++)
                {
                    i = pAllFields.FindField(sSubFields[j]);
                    if (i == -1)
                    {
                        MessageBox.Show("字段:" + sSubFields[j] + "  不存在");
                        return;
                    }
                    pOutFieldsEdit.AddField(pAllFields.get_Field(i));
                }
                IObjectLoader pObjectLoader;
                pObjectLoader = new ObjectLoaderClass();
                IEnumInvalidObject pEnumInvalidObject;
                pObjectLoader.LoadObjects(null, pInFClass as ITable, pQueryFilter,
pOutFClass as ITable, pOutFields, false, 0, false, false, 10,out pEnumInvalidObject);
                IInvalidObjectInfo pInvalidObject;
                pInvalidObject = pEnumInvalidObject.Next();
                //如果pInvalidObject不为空，则表明有数据没有被载入
                if ( pInvalidObject != null)
                {
                    MessageBox.Show("有些要素未被载入");
                }
            }
```

在这个转换过程中，只有 SHAPE、STATE_NAME、STATE_FIPS 和 AREA 四个字段的值会被载入到 States 要素类中。

IObjectLoader::LoadObjects 方法返回的是一个 ISelectionSet 对象，如果 applyValidation 参数设置为 false 的话，它将为空值；如果它是 true，则会执行有效性检验，且返回的选择集对象中也包含了无效的要素对象。在这个例子中，为了保证两个数据的结构一致，IQueryFilter::SubField 必须和 IObjectLoader::OutputFields 相匹配，在数据装载的时候，所有字段的 IsNullable 和 Editable 参数必须设置为 True，否则装载会失败。

本章讨论了 GeoDatabase 的基本知识点。GeoDatabase 既是 ESRI 推出的一种全新的地理数据模型，也是一种全新的物理存储方式——它可以使用 Access 数据库或 Oracle 作为物理数据库。GeoDatabase 使用了统一数据描述方式，在这个逻辑模型的数据操作中，无论地理数据保存在什么类型的文件或数据库中，它们更新、删除、添加数据的方式都是一样的。

在这一章中，用户讨论了各种数据集对象以及与数据集相关的——几何字段、索引、子类和值域等对象。除了这些实际数据对象外，还讨论了查询和选择数据的各种方式。最后，本章讨论数据的转换和装载等操作方法。

地理数据库是一个 GIS 系统中最基本的内容，也是最重要的内容之一。为了理解整个 ArcObjects 体系，读者需要认真阅读这一章的内容。

9 空间分析

空间分析是地理信息系统软件的基本功能之一，其应用范围十分广泛，从地质、地理现象到社会经济现象，各种具有空间属性的变量都可以通过空间分析揭示其空间特征。空间分析是基于地理对象的位置和形态特征的空间数据分析技术，其目的在于提取和传输空间信息。通过空间分析，不但可以知道数据库中的数据，而且可以通过这些数据去揭示更深刻、更内在的规律和特征，它特有的对地理信息(特别是隐含信息)的提取、表现和传输功能，是地理信息系统区别于一般信息系统的主要功能特征。空间分析根据使用的数据性质不同，可以分为：(1)基于空间图形数据的分析运算；(2)基于非空间属性的数据运算；(3)空间和非空间数据的联合运算。其最终目的是解决人们涉及的地理空间实际问题，提取和传输地理空间信息，特别是隐含信息，为将来的发展提供决策支持。

Spatial operations comprise an essential part of most GIS systems, allowing spatial querying and modification of spatial entities. The interfaces described below, implemented by many of the geometry coclasses, provide a wide range of spatial operations, including topological, relational, and proximity operations.

Geometries used in spatial operations should share the same coordinate system (SpatialReference), or the result of the operation may be meaningless. Some spatial operations may raise geometry error 553 (inconsistent spatial references) if comparing two geometries with different spatial references. IGeometry::Project can be used to convert a Geometry from one coordinate system to another, prior to attempting a spatial operation.

本章介绍了 GIS 中空间分析的部分基本功能，包括空间关系运算、空间拓扑运算、空间查询、缓冲区分析、叠加分析和网络分析，并描述如何结合 C#与 ArcObjects 实现这些功能。

9.1 空间查询

在 ArcMap 程序里有两种方法找到需要的数据：一种是基于非空间属性选择，另一种是基于空间属性选择。前者可以回答诸如“高度在 100 米以上的建筑是哪些”、“全市营业额在 100 万元的商店有多少”的问题，后者可以告诉用户“中山公园 1000 米范围内的停车场在什么地方”等空间查询问题。

ArcObjects 可以办到 ArcGIS 软件所有的事情，这一节将讨论 ArcObjects 中查询操作使用的各种对象。

9.1.1 Cursor 对象

Cursor 是数据处理中运用频率非常高的对象，凡是和数据操作有关的地方几乎都可以找到它，无论是查找数据还是在表中插入或者更新一条记录。

Cursor 的中文名是“游标”，它本质上是一个指向数据的指针，本身并不包含数据内容，它只是提供了一个连接到 Row 对象或要素对象的桥梁而已。游标有三种类型，即查询游标、插入游标和更新游标。每一种游标都是通过与之相适应的方法来获得的，如 Search、Insert 和 Update 方法。更新和插入游标都需要使用一个过滤器(Filter)对象，因为它们首先必须获得需要进行操作的要素。

Cursor 对象支持的接口是 ICursor，它定义了操作一个 Row 集合或一个 Row 对象的属性和方法，下面是获得插入型、更新型和查询型游标的方法：

pCursor = ITable::Insert()

Insert 方法返回一个插入型游标，它通常用于往表中插入一条记录。这个方法比使用 CreateRow 和 Store 两个方法插入一条记录的速度要快。

pCursor = ITable::Update()

Update 方法会返回一个更新型游标，它用于更新或者删除一条记录。这个方法使用起来比调用 Store 和 Delete 方法来处理的速度要快。

pCursor=ITable::Search()

使用 ITable::Search 方法对表进行查询后，可以得到一个查询型 Cursor 对象，它指向一个或多个 Row 对象，这个时候可以使用下面的方法来获得一个 Row 对象：

```
IRow prow;
prow = pCursor.NextRow();
while (prow != null)
{
    //…….对Row对象进行操作
    prow = pCursor.NextRow();
}
```

在 ArcObjects 中程序员常用的对象是 FeatureCursor，它是 Cursor 的一个子类，指向一个或多个要素。它实现了 IFeatureCursor 接口，这个接口定义的属性和方法与 ICursor 类似。

在帮助中搜索会发现，凡是 Search 方法，如 ILayer::search、ITable::search 等都可以返回一个游标对象。这个 Cursor 指向了那些符合选择条件的要素，如下面

的代码段：

```
IFeatureCursor pCursor;
pCursor = pFeatlyr.search();    //返回游标对象
IFeature pFeat ;
pFeat = pCursor.NextFeature();  //下面做一个循环可以遍历游标指向的要素
while (pFeat != null)
{
    //.....
    pFeat = pCursor.NextFeature();
}
```

同样地，IFeatureCursor 接口也定义了更新和插入一个要素对象的方法。

9.1.2 QueryFilter 与 SpatialFilter 对象

在 ArcObjects 中进行的查询或选择，都需要传给系统一个查找条件，即系统需要知道它将返回满足什么条件的数据出来。在关系数据库中，这些操作都是使用 SQL 语句的 WHERE 后面的子语句完成，如“Select * from pTable where age>18"这条 SQL 语句可以将 pTable 表中任何 age 大于 18 的行都选择出来。

ArcObjects 中并不能直接使用简洁方便的 SQL 语句，它是通过多个不同对象的配合来实现数据选择的目的的。QueryFilter 对象就承担了这个任务，它是一个依据属性查询表格型数据的过滤器。在一个打开的二维表中，执行这样的操作是最普遍的，毕竟用户并不总是需要所有的数据，而仅仅对于自己需要的那一部分感兴趣。使用这个过滤器将方便地取出某些记录，然后去更新或删除它们，这些与用户在普通关系型数据库中的做法并没有太大的区别。

IQueryFilter 接口定义了 QueryFilter 类用于过滤数据时使用的属性和方法。新建一个过滤器对象后，如 WhereClause 属性是必须设置的，它用于确定选择什么样的数据出来；而 SubFields 和 AddField 则用于设置返回的数据包含哪些字段，在默认的情况下，它被设置为“*”，意思是返回所有的字段，这与 SQL 语句的“Select * from”中间的条件非常相似。

如果希望返回的数据的字段中包括几何字段，那么 IQueryFilter::OutputSpatialReference 属性就必须设置为这个字段的空间参考。下面的例子显示了如何在一幅美国地图中选出 California 来，它仅仅选择了基于 STATE_NAME 属性中一条记录，当然这条记录还是由一个 Cursor 对象来指向的：

```
IQueryFilter pQueryFilter;
pQueryFilter = new QueryFilterClass();
```

```
//设置过滤器对象的属性
pQueryFilter.WhereClause = "STATE_NAME = 'California'";
IFeatureSelection pFeatureSelection;
pFeatureSelection = pFeatureLayer as IFeatureSelection;
pFeatureSelection.SelectFeatures(pQueryFilter,
esriSelectionResultEnum.esriSelectionResultNew, false);
```

如果在选择的过程中不过滤任何的记录，而是返回所有的数据，则可以使用关键字“null”来替代 pQueryFilter 对象，这种情况非常常见，比如用户如果希望知道一个要素类中的要素数目，通常使用这个语句：

```
pFeatureClass.FeatureCount(null)
```

SQL 语句是不区分大小写的，数据库中的字段名也是不区分大小写的，但是在查询数据的时候，就必须注意查询字符串的大小写，比如查询一个城市名为北京的地方，如果在数据库中北京的值为“Beijing”，执行的 WhereClause 语句就必须是 ”cityname='Beijing'" (注意 Beijing 外面还带有单引号)。如果“Beijing”为“beijing”就是错误的。

除了在选择条件中使用完全查询外，过滤器对象也支持模糊查询，如选择一个全国城市 SDE 数据库中所有名称以‘B’开始的城市，过滤器的 WhereClause 属性应该为：

```
pQueryFilter. WhereClause="STATE_NAME like 'B%'"
```

如果数据库是 Access 的，则代码为：

```
pQueryFilter. WhereClause= "STATE_NAME like 'B * '"
```

在 Access 和其他数据库中，对于 SQL 语句中的字符的差别如下:

	单个字符	多个字符	示例
Access 数据库	?	*	State like “M?”
SDE，SHP，Coverage	_	%	State like “M_”

如果选择城市名的第二个字母为'B'，这 WhereClause 属性为

```
pQueryFilter.WhereClause="STATE_NAME like '_B%'"
```

如果 GIS 只能通过普通属性来查询数据，那么它就不能称为“地理”型的信息系统。在 GIS 中更频繁的是执行具有空间因素的查询，如“在 100 米内的电话亭有哪些?”等，显然地，这些查询是基于要素几何数据的。而进行这种空间查询过滤运算的，是 IQueryFilter 的子类 ISpatialFIter。

图 9-1 是 ArcMap 中进行空间查询的对话框。

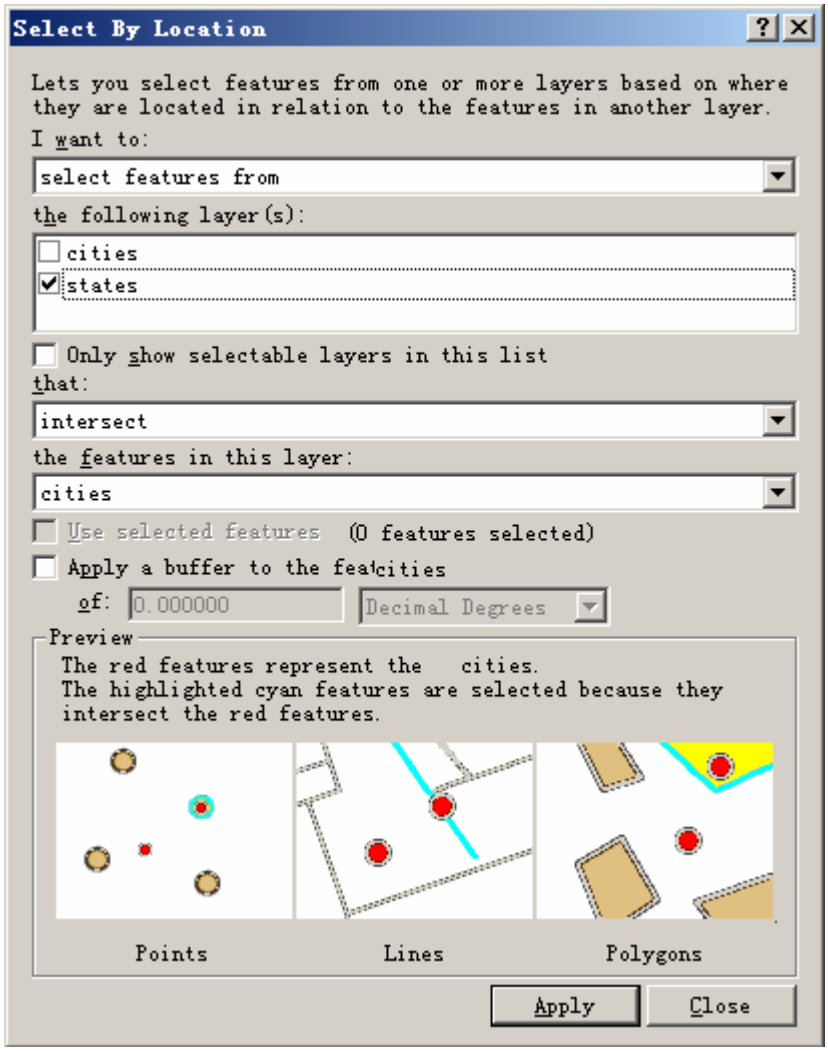


图 9-1 ArcMap 中空间查询对话框

用户可以进行的空间过滤范围非常广泛，如

- 寻找与某个选择区域相覆盖的要素
- 寻找某个要素附近的对象
- 为要素的显示产生一个有限的地理范围

由于是 IQueryFilter 的子类，ISpatialFilter 对象就包含了空间属性和普通属性两种限制，因此可以使用它执行两种类型的查询，这个特点非常有用，比如需要查询“在

某一点 2000 米范围内的服装店”，它就需要查询两个特征：一是 2000 米范围的空间属性查询，二是服装店要素的普通属性查询。

ISpatialFilter 过滤器是基于空间关系的，这个接口定义了许多空间过滤时需要使用的属性，其中 Geometry、GeometryField 和 SpatialRel 属性是这个过滤器在新建的时候必须进行设置的。SpatialRel 属性是一个确定查询几何对象与目标要素几何对象关系的枚举值，这个空间关系非常关键，它支持几种基本的空间关系，具体如下：

空间关系	描述(A 为查询几何对象，B 为目标几何对象)
esriSpatialRelUndefined	关系未定义
esriSpatialRelIntersects	A 与 B 相交
esriSpatialRelEnvelopeIntersects	A 的包络线和 B 的包络线相交
esriSpatialRelTouches	A 与 B 相接，即其边界处相接
esriSpatialRelOverlaps	A 与 B 相叠加，它们必须是同维对象，如都是多边形等
esriSpatialRelCrosses	A 与 B 相交，如两条线交于一点，面和线交于一条线，面与面无此关系
esriSpatialRelWithin	A 在 B 的内部
esriSpatialRelContains	A 被 B 包含

下面的代码显示如何使用一个空间过滤器，它也是在一个查询方法中使用的：

```
ISpatialFilter pFilter;
pFilter = new SpatialFilterClass();
//设置空间过滤器的三个必须属性
pFilter.Geometry = pPoint;
pFilter.GeometryField = "SHAPE";
pFilter.SpatialRel = esriSpatialRelEnum.esriSpatialRelIntersects;
IFeatureCursor pFeatureCursor;
pFeatureCursor = pFeatureClass.Search(pFilter, false);
```

在上面的代码段中，pFeatureClass 要素类的 Search 方法将返回一个查询型游标，这个游标指向要素类中所有与 pPoint 点具有相交拓扑关系的要素。

ISpatialFilter::SearchOrder 属性用于设置选择顺序，如果用户使用的空间查询

同时包括普通属性和空间属性的时候，就可以用它决定哪一种属性的查询在先。这种方法在某种程度上可以提高查询速度，如“查询在某点周围 2000m 内的高度在 50m 以上的建筑物”。在默认情况下，空间属性过滤在普通属性过滤之前，即 esriSearchOrderSpatial 属性为默认设置。

9.1.3 QueryDef 对象

QueryDef 对象代表了数据库中基于一个或多个表、要素类进行的属性查询。通过这个对象，用户可以在多个表间建立连接，并且保证在这个连接基础上的查询可以实现。

QueryDef 中的表必须放在一个工作空间内，而 QueryDef 对象也是使用 IFeatureWorkspace::CreateQueryDef 方法来产生的，它并不是一个组件类。需要注意的是，它不能在 Shapefile 和 Coverage 数据中使用，而仅仅可以使用在 GeoDatabase 数据库中。QueryDef 的查询结果是 Cursor 对象返回的，系统可以依据返回的 Cursor 来获得基于查询的要素集合。但是使用这种方式获得的 Cursor 对象，仅仅是只读的，Cursor 指向的行对象并没有和它们的父表相关联，因而如果试图对这些被选择出来的数据使用 store 方法会出现错误。

IQueryDef 是 QueryDef 对象实现的主要接口，它定义了在多个表中定义连接需要使用的属性和方法。SubFields 属性是产生一个 QueryDef 对象时的可选项，默认情况下它为“*”，它表示返回所有的字段。

下面的例子用于显示如何产生一个 QueryDef 对象，并使用它在两个表中产生一个连接。

```
//产生一个QueryDef对象
IQueryDef pQueryDef;
pQueryDef = pFeatureWorkspace.CreateQueryDef();
//基于连接的两个表必须是在一个工作空间内，设置它的各种属性
pQueryDef.Tables = "Counties,States";
pQueryDef.SubFields = "Counties.Shape,Counties.NAME,States.STATE_ABBR";
pQueryDef.WhereClause = "Counties.STATE_FIPS = States.STATE_FIPS";
```

IFeatureWorkspace::OpenFeatureQuery 方法会产生一个基于 QueryDef 对象的要素类，这个要素类可以被作为一个要素图层被添加到一个 Map 对象中去，它和 ArcSDE 中的“视图”概念很类似。

下面的例子就是基于一个上面产生的 QueryDef 对象来新建一个要素类的例子。注意 IQueryDef::SubFields 中必须设置一个字段，以保证新建的要素类最少拥有一

个属性字段。

```
//新建一个要素类
    IFeatureClass pFeatureClass;
    IFeatureClassContainer pFeatureClassContainer;
    IFeatureDataset pFeatureDataset;
    pFeatureDataset = pFeatureWorkspace.OpenFeatureQuery("My counties join ",
pQueryDef);
    pFeatureClassContainer = pFeatureDataset as IFeatureClassContainer;
    //判断IFeatureClassContainer对象中是否有要素类存在
    if (pFeatureClassContainer.ClassCount !=1)
    {
        MessageBox.Show("Failed to create feature class by query!");
        return;
    }
    pFeatureClass = pFeatureClassContainer.get_Class(0);
    //添加要素类到Map对象
    IFeatureLayer pFeatureLayer;
    pFeatureLayer = new FeatureLayerClass();
    pFeatureLayer.FeatureClass = pFeatureClass;
    pFeatureLayer.Name = pFeatureClass.AliasName;
    pMap.AddLayer(pFeatureLayer);
```

QueryDef 对象使用的 SQL 语法和关系数据库中使用的 SQL 语法是一样的，程序可以使用工作空间对象的 ISqlSyntax 接口来获得这个数据库中的 SQL 语法信息。

QueryDef 代表了一个查询的子集，在数据库中，用户也可以使用 SELECT 语句来得到同样的结果。但是 QueryDef 并不能完整地支持 SQL 语句。在 ArcObjects 中的 SQL 语句不能使用 SQL 语句的 order by，如果需要对选择的结果进行排序，可以使用下一节介绍的 ITablesort 接口进行操作。

9.1.4 TableSort 对象

ArcObjects 虽然支持使用 SQL 语句来对数据进行查询，但是这种支持的程度很低。如果想要依据某个字段来排列查找出来的数据，在普通关系数据库中仅仅是在 Select 语句后面加上“Order by * * * ”，但是 ArcObjects 对这个操作不能直接支持。

ArcObjects 的帮助中指出，ESRI 会在 ArcObjects 的下个版本中提供这个基本的功能。在它们还不能支持前，用户可以使用 TableSort 对象来完成这个任务。

TableSort 对象是表的排序对象，它可以将查找的结果按照某个字段进行排序，然后返回所有符合条件的要素的游标。

ITableSort 接口可以设置 TableSort 对象的属性并且执行排序。对一个 TableSort 而言，Table 和 Fields 属性是必须被设置的，而其他的属性都是可以选择的。除了使用表、对象类和要素类外，选择集也可以作为一个需要排序的数据源。

Fields 是一个用于排序的字段列表，它可以有多个字段输入，当使用 ITableSort::Sort 方法排序的时候，它会首先根据第一个字段进行排序，然后根据第二个字段进行排序,依此类推。

用户也可以使用 Cursor 属性用于排序，在设置 Cursor 时，必须使用 Table 属性来设置这个 Cursor 指向的表对象。

除此以外，用户还可以设置 Ascending、CaseSensitive 等属性，前者用于确定使用哪个字段升序排列，后者确定使用大小写敏感设置。

下面的代码段是一个排序的例子：

```
private void FeatureSort()
{
    IWorkspaceFactory pFact;
    pFact = new ShapefileWorkspaceFactoryClass();
    IFeatureWorkspace pWorkspace ;
    pWorkspace = pFact.OpenFromFile(@"d:\arcgis\data\", 0) as
IFeatureWorkspace;
    IFeatureClass pFeatClass;
    pFeatClass = pWorkspace.OpenFeatureClass("states");     //产生一个临时的工
作空间
    IWorkspace pScratchWorkspace ;
    IScratchWorkspaceFactory pScratchWorkspaceFactory;
    pScratchWorkspaceFactory = new ScratchWorkspaceFactoryClass();
    pScratchWorkspace = pScratchWorkspaceFactory.DefaultScratchWorkspace;
    //新建一个过滤器
    IQueryFilter pQFilt;
    pQFilt = new QueryFilterClass();
    pQFilt.WhereClause = "Area > 77000";
    //获得选择集
    ISelectionSet pSelectionSet;
    pSelectionSet = pFeatClass.Select(null,
esriSelectionType.esriSelectionTypeIDSet, esriSelectionOption.esriSelectionOptionNormal,
```

```
pScratchWorkspace);
                //新建一个TableSort对象
                ITableSort pTableSort;
                pTableSort = new TableSortClass();
            //进行排序的字段
            pTableSort.Fields = "Area,state_name";
                //Area字段不是升序排列
            pTableSort.set_Ascending("Area",false);
                pTableSort.set_Ascending("State_name", true);
                //设置排列的数据源
            pTableSort.SelectionSet = pSelectionSet;
                pTableSort.Sort(null);
                IFeatureCursor pCursor;
                pCursor = pTableSort.Rows as IFeatureCursor;
                IFeature pFeat;
                pFeat = pCursor.NextFeature();
                int i;
                while ( pFeat != null)
                {
                    MessageBox.Show(pFeat.get_Value(3).ToString() + ":"+
pFeat.get_Value(2).ToString());
                    pFeat = pCursor.NextFeature();
                }
        }
```

上面的例子是使用一个从要素类中提取出来的要素选择集作为数据源，对这个要素选择集进行排序的操作。其中要素的第三个字段为“State_Name”，第二个字段为“Area”。

另外，还必须提醒读者的是，需要注意 IFeatureClass::Select 方法和 IFeatureSelection::Select 方法之间的区别。

9.1.5 要素选择集

在查询数据的过程中，程序员经常遇到选择集的问题。熟悉 ArcMap 的用户都知道，要得到符合某种选择条件的数据，一般有两种不同的手段：一种是使用 Search 方法，它返回的是一个指向数据的查询型 Cursor 对象；另一种是使用 Select 方法构造一个选择集(SelectionSet)，然后在选择集中再得到 Cursor，后者虽然走了弯路，

但在显示的效果上却比前者好，即得到的要素选择集在 Map 上会呈高亮显示。

通过 IMap::SelectByShape 方法用户可以得到选择集，使用要素图层的 IFeatureSelection 接口也可以获得选择集，但是这些都仅仅是在“显示表面”上探讨，那么要素选择集在地理数据库中到底表现为什么东西呢?

SelectionSet 对象是程序获得的被选择的行对象的集合，这些行只能来自单个表或要素类，但是一个表或者要素类可以产生多个选择集对象。选择集有两种形式，它要么是基于被选择的行对象的 OID 集合，要么就是实实在在的行对象本身，在后面这种情况下，选择集会提供方法让程序员与在选择集中的行对象进行交互。至于使用何种方式来得到选择集，可以使用 SelectionType 属性来设置。

选择集的类型如果是 esriSelectionlDSet，那代表选择集使用的是一个 OID 集合，这些 OID 值可能保存在一个物理表中，也可能保存在内存中，这取决于数据源的类型；当它为 esriSelectionTypeSnapShot 时，它表明选择集使用的是保存在内存中的实际的行对象；如果是 esriSelectionTypeHybird，那情况就比较灵活，当选择数量少的时候选择集使用在内存中的行对象，当数量多时则使用 OID 集合。

使用标识集合(如 OID 集合)可以表示数目巨大的选择集，它也是程序最常使用的一种方式，当数据源每一次被选择的对象还需要进行查询的时候，这种方式可以保证选择集是动态的，而且在数据源改变的时候自动变化。Snapshot 类型选择集速度最快，而且一旦选择集构造完成后就不再要求与数据源发生查询，但是它对少量数据是有效的，如果选择的数据多，占用内存大时，它的优势就发挥不出来了；Hybird 类型选择集包含了前两者的优点，它可以依据选择数据的大小而自动选用不同的选择方式，但是这个选择数据的大小并不能由程序来控制。

下面是一个在要素类中构造要素选择集的例子：

```
ISelectionSet pSelectionSet ;
pSelectionSet = pFeatcls.Select(pQueryFilter,
esriSelectionType.esriSelectionTypeHybrid, esriSelectionOption.esriSelectionOptionNormal,
null);
```

上面代码段中的 Select 方法的最后一个参数被设置为 null，这是它的缺省设置，这个参数被用于传递一个存储 OID 集合的工作空间，而这个 OID 集合结构如同一个二维表，这个表是临时的，且保存在同一个数据库中。如果程序员是在基于文件的工作空间中构造的选择集，则这个表被保存在内存中。无论是哪一种情况，这个参数设置为 null 都是合适的。

使用 ArcCatalog 软件新建一个 Personal GeoDatabase，然后在数据库中任意添加数个要素类，关闭 ArcCatalog 后直接打开新建的 Access 数据库物理文件，将可以看到这个 PersonalGeoDatabase 的物理结构(如图 8-3)，在这个数据库的表中

用户将不会发现任何关于选择集内容的表。

打开 ArcMap 程序，然后载入刚才添加在地理数据库中的要素类，使用工具栏的“SelectFeatures”命令构造选择集，然后再次打开 Access 数据库，用户将发现在数据库中新出现了两张表 SelectedObjects 和 Selections。分别打开这两张表，如果选择集中的要素数目多，将会在表中发现这些要素的 OID 集合，如果选择的要素数目少，则在两张表中根本不会出现任何记录，这是因为这张临时表保存在内存中了。这就是选择集的物理结构实质。

SelectionSet 对象主要支持的是 ISelectionSet 接口，它用于管理和查询一个选择集。

ISelectionSet::Search 方法通常用于在选择集内进行再选择，它返回的是一个查询型 Cursor 对象，当然它也需要一个过滤器对象才行。当程序员使用 QueryFilter 对象时，选择集的类型一律是使用的标识集合，即使它开始是 Snapshot 类型的选择集。ISelectionSet::Select 方法则用于在目前的选择集中构造一个新的选择集对象，它也是使用一个过滤器对象来进行设置的。

Add、AddList 和 ReomveList 方法可以用于选择集中添加和移除对象，这些方法都是通过 OID 属性来完成的。ISelectionSet::Combine 方法则可以用于绑定两个选择集，这两个选择集必须是来自同一个目标表或者要素类。

SelectionSet 对象也支持 ISelectionSet2 接口，它提供的方法用于更新或删除选择集中的对象，更新和删除都使用到了 Cursor 对象。关于更新、删除等操作使用的 Cursor 对象，当然类型是不同的，关于这些内容，可以参看 Cursor 对象一节。

下面的代码段是使用 IDataStatistics 对象获得一张表中所有数据的 pop1990 数值字段的平均值：

```
        private void SelectionStatistic()
        {
            IFeatureLayer pFeatureLayer;
            IMap pMap = axMapControl1.Map;
            pFeatureLayer = pMap.get_Layer(1)as IFeatureLayer;
            IFeatureSelection pFeatureSelection;
            pFeatureSelection = pFeatureLayer as IFeatureSelection;
            pFeatureSelection.SelectFeatures(null,esriSelectionResultEnum.esriSelectionR
esultNew,false);
            ISelectionSet pSelectionSet;
            pSelectionSet = pFeatureSelection.SelectionSet;
            //得到要素类中全部的要素
            IFeatureCursor pFeatureCursor;
```

```
            ICursor pCursor;
            pSelectionSet.Search(null, true, out pCursor);
            pFeatureCursor = pCursor as IFeatureCursor;
            //设置IDataStatistics对象的属性
            IDataStatistics pDataStatistics;
            pDataStatistics = new DataStatisticsClass();
            pDataStatistics.Cursor = pCursor;
            pDataStatistics.Field = "POP1990";
            MessageBox.Show(pDataStatistics.Statistics.Mean.ToString());
        }
```

为了选择要素，用户不一定非要使用选择集，实际上使用 IFeatureSelection::SelectFeatures 方法更为简单，它可以将已经通过 Search 方法得到的要素添加到一个选择集中去。这种方法在某些情况下是非常有用的。

9.2 空间拓扑运算

9.2.1 ITopologicalOperator 接口

通过一系列基于一个或者多个几何图形中点间的逻辑比较，然后返回另外一些几何图形，这个过程就是空间几何图形的拓扑运算。它是空间分析的基础，各种空间分析的结果都可通过几何图形之间的拓扑运算实现。例如：为了判断哪些购物中心被地震破坏的危险度最大，需查询沿一条地震的断层两侧 1000m 范围内的所有购物中心。这是一个典型的缓冲区查询问题，但实际上就是将缓冲区多边形与购物中心进行交运算的过程。

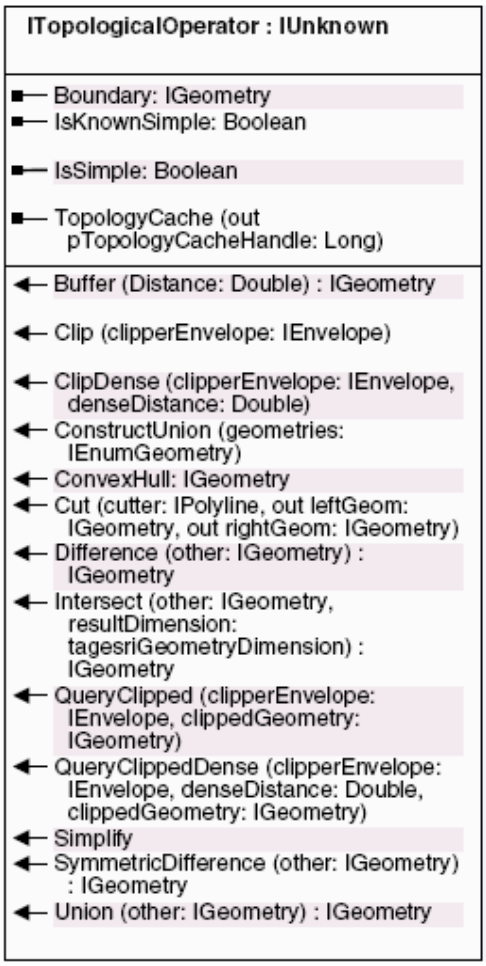


图 9-2 ITopologicalOperator 接口

空间几何图形的拓扑运算包括裁切(Clip)、凸多边形(Convex hull)、切割(Cut)、差分(Difference)、交集(Intersect)、对称差分(又称为异或，Symmetric difference)和并集(Union)等，这些拓扑运算在 ITopologicalOperator 接口中定义，在 GeometryBag、Multipoint、Point、Polygon、Polyline 类中实现。注意 ITopologicalOperator 接口的方法仅仅能使用在高级几何对象上，即 Point、Multipoint、Polyline 和 Polygon；如果要使用在低等级的几何对象上，如 Segment、Path 或 Ring 上，它们需要先组合成高级别几何对象才行。

ITopologicalOperator::Boundary 可以返回一个几何对象的边界，Boundary 的维度比原对象要低一维，如多边形的 Boundary 是多义线，多义线的 Boundary 则是它的顶点集合，而点集对象的 Boundary 则是不存在，因为没有低于 0 维的集合对象。

ITopologicalOperator::Buffer 可以给一个高级别几何对象产生一个缓冲区，无论是点、多边形还是多义线，它们的缓冲区都是一个具有面积的几何对象。

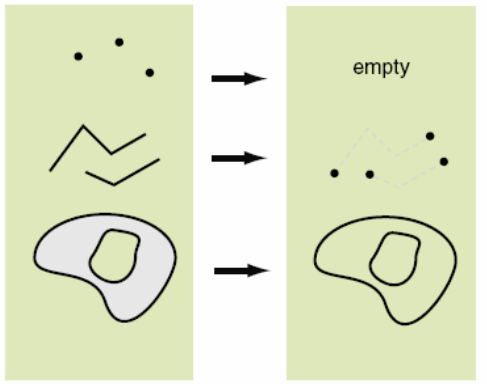


图 9-3 几何对象的边界（Boundary）

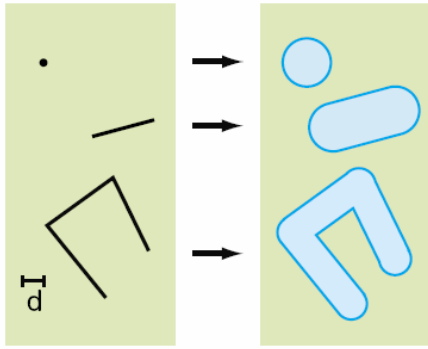


图 9-4 几何对象的缓冲区操作（Buffer）

ITopologicalOperator::Clip 方法可以将一个几何对象使用一个包络线来进行裁切，裁切的结果为几何对象被包络线包围的部分，如图 9-5 所示。

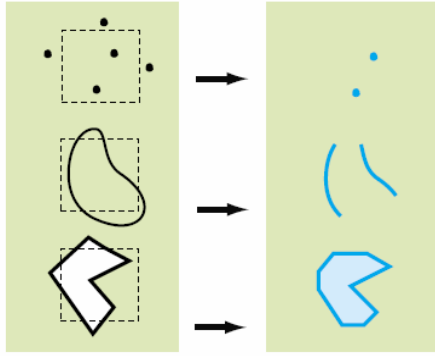


图 9-5 Clip 方法

ITopologicalOperator::ConstructUnion 方法可以将一个几何对象的枚举(包含

了多个几何对象的枚举值)与同维度的单个几何对象合并，这种方法在大量几何对象合并的时候是非常有效的；Union 方法则可以合并两个同维度的单个几何对象，合并后的两个单个几何对象将变成一个几何对象，如图 9-6 所示；ITopologicalOperator:: ConvexHull 方法可以产生一个几何图形的最小的边框凸多边形，如图 9-7 所示。凸多边形是一个没有凹面且包含其他几何图形的最小多边形。

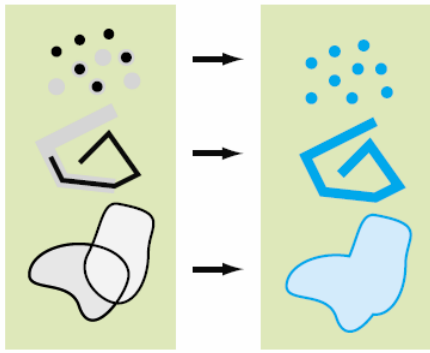


图 9-6 Union 方法

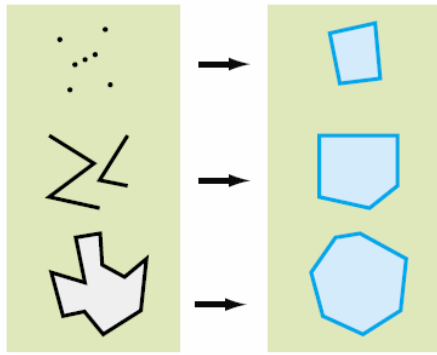


图 9-7　ConvexHull 方法

ITopologicalOperator:: Cut 方法指定一条切割曲线和一个几何图形，经过切割运算后把几何图形分为左、右两部分，左、 右两部分是相对曲线的方向而言的(如图所示)。注意：点和多点是不能被切割的，而折线和多边形只有与切割曲线相交时才能进行切割运算。

ITopologicalOperator::Difference 方法可以产生两个几何对象的差集。如 A 是源对象，B 是参与运算的几何对象，则 C=A.Difference(B)，C 是 A 减去 A 与 B 的交集后剩下的部分；而 SymmetricDifference(对称差分)方法则是将 A 与 B 的并集减去 A 与 B 的交集部分。

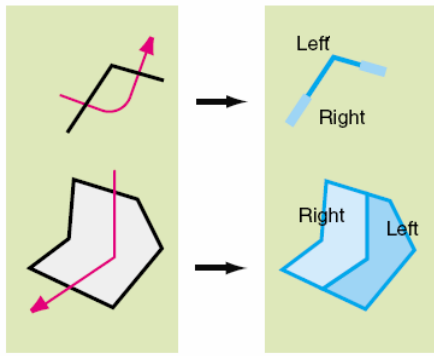


图 9-8 Cut 方法

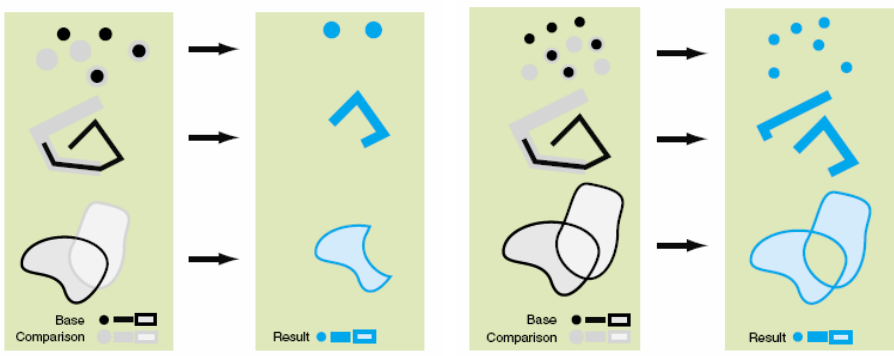


图 9-9 Difference（左）、SymmetricDifference（右）方法

ITopologicalOperator::Intersection 则可以返回两个同维度几何形体对象的交集，即两个对象的重合部分。

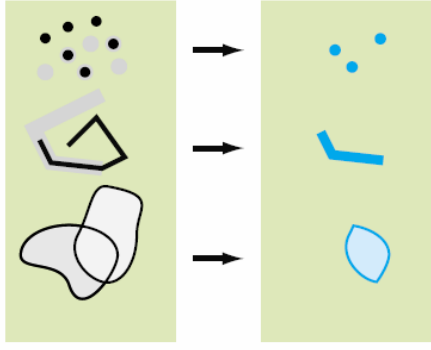


图 9-10 Intersection 方法

参与空间拓扑运算的几何形体，必须是拓扑上简单的（topologically simple），否则会产生 esriGeometryError 536 错误。

The IsKnownSimple property returns True if the Geometry has not been

changed since simplicity was last confirmed, whereas IsSimple actually checks the Geometry for topological simplicity. It may be more efficient, therefore, to check IsKnownSimple before using IsSimple, especially inside a code loop, as shown in this code:

```
  IEnumGeometry pEnumGeom;
pEnumGeom = pGeometryBag as IEnumGeometry;
ITopologicalOperator pTopoOp;
pTopoOp = pEnumGeom.Next() as ITopologicalOperator;
while ( pTopoOp != null)
{
      if(!( pTopoOp.IsKnownSimple))      //We check IsKnownSimple first here
      {                                        //because its quicker than IsSimple.
            if (!( pTopoOp.IsSimple))          // This could save time if the enum
                  pTopoOp.Simplify();                 // is particularly large.
      }
   pTopoOp = pEnumGeom.Next() as ITopologicalOperator;
}
```

ITopologicalOperator::Simplify 方法可以让一个几何对象变得在拓扑上一致，例如在一个 PointCollection 中，它可以让所有的重合点(即两个点拥有相同坐标值)被移除(触发拥有不同的属性)；对于 SegmentCollection，它将移除重合的线段、而相交的线段将会变成非相交线段(即在相交点处产生一个顶点)；对于 Polygon 而言，所有相交的环将被移除，所有的内外的方向将修正，未封闭的环将被封闭，如图 9-11 所示。

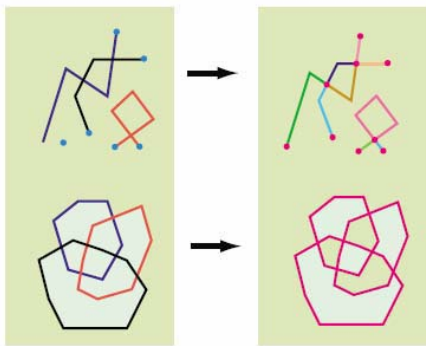


图 9-11　Simplify 方法

拓扑是 Geoemtry 中一个非常重要并且基本的概念，在地理数据分析中承担了重要的责任，关于拓扑的实质和众多的拓扑方法，读者可以阅读相关的 GIS 书籍。

9.2.2 开发实例

9.2.2.1BufferToShapefile

该实例的功能是对选择的要素作缓冲，并将结果保存为一 shapefile 文件。

BufferToShapefile.cs

```
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.DataSourcesFile;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry ;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class BufferToShapefile:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap;
        IActiveView pActiveView;

        public BufferToShapefile()
        {
            base.m_caption = "BufferToShapefile";
            base.m_message = "BufferToShapefile";
            base.m_toolTip = "BufferToShapefile";
            base.m_category = "BufferToShapefile";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
```

```
{
        m_hookHelper.Hook = hook;
}

public override void OnClick()
{
        IEnumFeature pEnumFeature;
        IFeature pFeature;
        ITopologicalOperator pTopoOp;
        IPolygon pPoly;
        int bufferDistance;

        pActiveView = m_hookHelper.ActiveView;
        pMap = m_hookHelper.FocusMap;
        if (pMap.SelectionCount == 0)    return;
        bufferDistance = 2;
        pEnumFeature = pMap.FeatureSelection as IEnumFeature;
        pEnumFeature.Reset();
        pFeature = pEnumFeature.Next();
        IFeatureClass pFClass;
        pFClass = CreateNewShapefile(@"d:\Temp", "MyBuffers");
        IFeatureBuffer pFeatureBuffer;
        pFeatureBuffer = pFClass.CreateFeatureBuffer();
        IFeatureCursor pFeatureCursor;
        pFeatureCursor = pFClass.Insert(true);
        int iFldAttribute;
        iFldAttribute = pFClass.FindField("Text");
        while ( pFeature != null)
        {
                pTopoOp = pFeature.Shape as ITopologicalOperator;
                pPoly = new PolygonClass();
                pPoly = pTopoOp.Buffer(bufferDistance) as IPolygon;
                pFeatureBuffer.Shape = pPoly;
                pFeatureBuffer.set_Value(iFldAttribute, pFeature.OID);
                pFeatureCursor.InsertFeature(pFeatureBuffer);
                pFeature = pEnumFeature.Next();
        }
        pFeatureCursor.Flush();
```

```
        }

        private IFeatureClass CreateNewShapefile(string Path, string Name)
        {
            IWorkspaceFactory workspaceFactory ;
            workspaceFactory = new ShapefileWorkspaceFactoryClass();
            IFeatureWorkspace featureWorkspace ;
            featureWorkspace = workspaceFactory.OpenFromFile(Path, 0) as
IFeatureWorkspace;

            IFields fields;
            fields = new FieldsClass();
            IFieldsEdit fieldsEdit;
            fieldsEdit = fields as IFieldsEdit;

            IField field;
            IFieldEdit fieldEdit;

            field = new FieldClass();
            fieldEdit = field as IFieldEdit;
            fieldEdit.Name_2 = "Shape";
            fieldEdit.Type_2 = esriFieldType.esriFieldTypeGeometry;

            IGeometryDef geometryDef;
            geometryDef = new GeometryDefClass();
            IGeometryDefEdit geometryDefEdit;
            geometryDefEdit = geometryDef as IGeometryDefEdit;
            geometryDefEdit.GeometryType_2 = esriGeometryType.esriGeometryPolygon;
            geometryDefEdit.SpatialReference_2 = new UnknownCoordinateSystem() as
ISpatialReference ;
            fieldEdit.GeometryDef_2 = geometryDef;
            fieldsEdit.AddField(field);

            field = new FieldClass();
            fieldEdit = field as IFieldEdit ;
            fieldEdit.Name_2 = "Text";
            fieldEdit.Type_2 = esriFieldType.esriFieldTypeString;
            fieldEdit.Length_2 = 25;
```

```
                fieldsEdit.AddField(field);

                IFeatureClass featureClass;
                featureClass = featureWorkspace.CreateFeatureClass(Name, fields, null, null,
esriFeatureType .esriFTSimple, "Shape", "");
                return featureClass;
            }
        }
}
```

9.2.2.2缓冲区查询

该实例的功能是根据鼠标点击位置产生一缓冲区，然后找出缓冲区内的要素。

SelectionBasedonPoint

```
using System;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class SelectionBasedonPoint: BaseTool
    {
        IMap pMap;
        IActiveView pActiveView;

        IHookHelper m_hookHelper = new HookHelperClass();

        public SelectionBasedonPoint()
        {
            base.m_caption = "SelctionBasedonPoint";
            base.m_message = "SelctionBasedonPoint";
            base.m_toolTip = "SelctionBasedonPoint";
```

```
            base.m_category = "SpatialQuery";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            IFeatureLayer pFeatureLayer = pMap.get_Layer(1) as IFeatureLayer;
            IFeatureClass pFeatureClass = pFeatureLayer.FeatureClass;

            //pPoint是点击的点
            IPoint pPoint =
pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X, Y);

            //对点对象做缓冲区运算
            ITopologicalOperator pTopo;
            pTopo = pPoint as ITopologicalOperator;
            IGeometry pBuffer;
            pBuffer = pTopo.Buffer(4);
            IGeometry pGeometry = pBuffer.Envelope;
            //新建一个空间过滤器
            ISpatialFilter pSpatialFilter;
            pSpatialFilter = new SpatialFilterClass();
            pSpatialFilter.Geometry = pGeometry;
            //依据被选择的要素类的类型不同，设置不同的空间过滤关系
            switch( pFeatureClass.ShapeType )
            {
                case esriGeometryType.esriGeometryPoint:
                    pSpatialFilter.SpatialRel =
```

```
esriSpatialRelEnum.esriSpatialRelContains;
                    break;
                case esriGeometryType.esriGeometryPolyline:
                    pSpatialFilter.SpatialRel =
esriSpatialRelEnum.esriSpatialRelCrosses;
                    break;
                case esriGeometryType.esriGeometryPolygon:
                    pSpatialFilter.SpatialRel =
esriSpatialRelEnum.esriSpatialRelIntersects;
                    break;
            }
            IFeatureSelection pFeatureSelection;
            pFeatureSelection = pFeatureLayer as IFeatureSelection;
    pFeatureSelection.SelectFeatures(pSpatialFilter,esriSelectionResultEnum.esriSelectionR
esultNew, false) ;
            ISelectionSet pFeatSet;
            pFeatSet = pFeatureSelection.SelectionSet;
            ICursor pCursor;
            pFeatSet.Search(null, true, out pCursor) ;
            IFeatureCursor pFeatureCursor = pCursor as IFeatureCursor;
            IFeature pFeature = pFeatureCursor.NextFeature();
            //遍历所有符合要求的要素，使用IMap::SelectFeature方法将它们加到图层的要素
选择集中
            while(pFeature != null)
            {
                //将在当前图层中获得的要素放入当前图层的要素选择集中
                pMap.SelectFeature(pFeatureLayer, pFeature);
                pFeature = pFeatureCursor.NextFeature();
            }
            pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeoSelection, null,
null);
        }
    }
}
```

9.3 空间关系运算

9.3.1 IRelationaloperator 接口

几何对象除了可以做各种拓扑运算外，它们之间还拥有某种关系属性，如包含、相等、在内部、相交、叠加等。这些关系都可以通过 IRelationaloperator 的方法获得。关系运算是在两个几何对象中进行的，它们都返回一个布尔值来说明某种关系是否存在。有些关系运算需要两个几何对象的维度相同，有些关系运算没有这个限制。

所有支持 ITopologicaloperator 的几何对象的类都实现了 IRelationaloperator 接口，其中包括 Envelope ，它意味着还可以对两个几何对象的包络线进行关系检查。

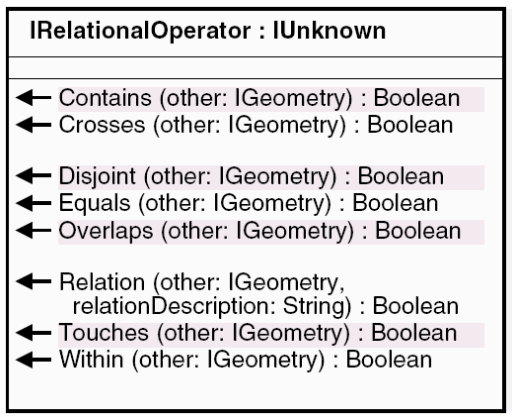


图 9-12 IRelationaloperator 接口

IRelationaloperator::Contains 方法用于检查两个几何对象的包含关系，如果 A 包含 B，则 A 的维度必须等于或高于 B ，即点只可以包含点对象；多义线可以包含点和多义线；多边形可以包含点、多义线和多边形。如果 A 包含 B ，则 B 与 A 的关系就是在 A 内部，即 Within ，这个属性可以通过 IRelationaloperator::Within 方法检测。

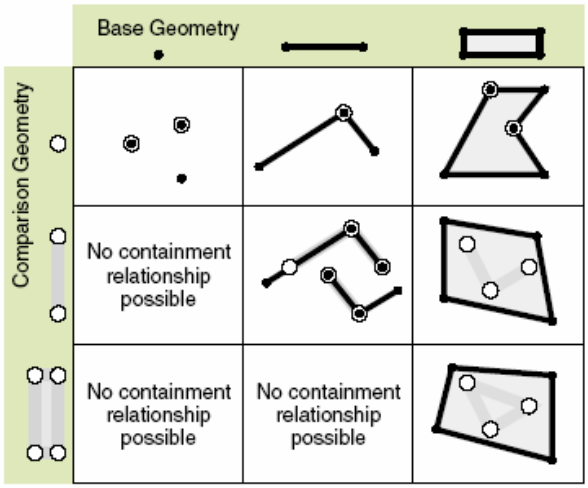


图9-13 Does the current geometry contain the other geometry?

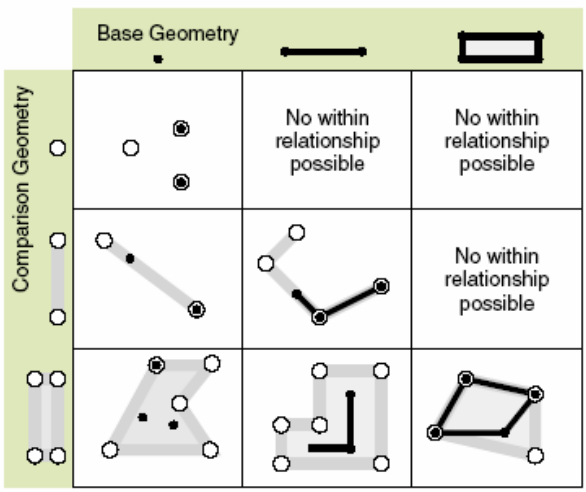


图9-14　Is the current geometry within the other geometry?

IRelationaloperator::Cross 方法检测两个几何对象是否相交，两个几何对象相交于较低维的几何对象而不是较高的，这说明点对象与点、多义线和多边形都构不成相交关系；多义线与多义线相交于一个点；多义线与多边形相交于一条线；而多边形与多边形没有相交关系，它们是一种叠加（overlap）关系。如果两个几何对象的交集为空，则它们之间是不相交（ disjoint ）关系，即 b = A.disjoint (B) , b 返回为 true。两个存在叠加关系的几何对象必须是同一维度的，如点集与点集可以叠加、多义线与多义线可以叠加、多边形与多边形可以叠加。如果多义线 A 与多义线 B 叠加，则意味着 A 的全部路径对象与 B 的一部分路径是一样的。叠加关系可以使用 IRelationaloperator::overlap 方法检测。

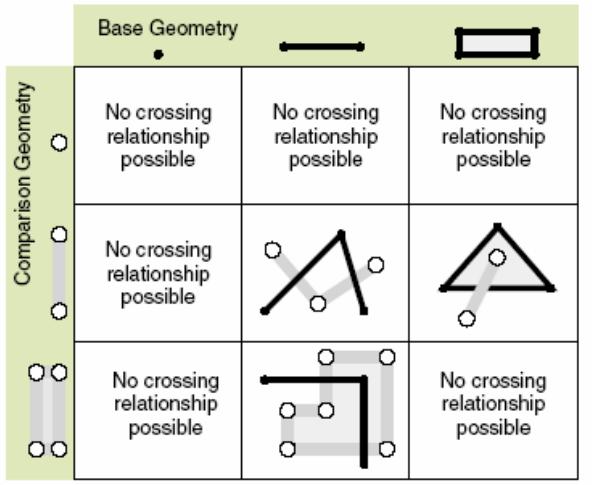


图9-15　Does the current geometry cross the other geometry?

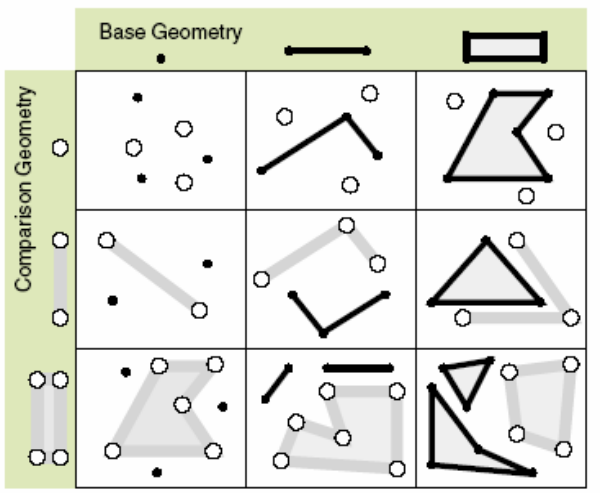


图9-16　Is the current geometry disjoint from the other geometry?

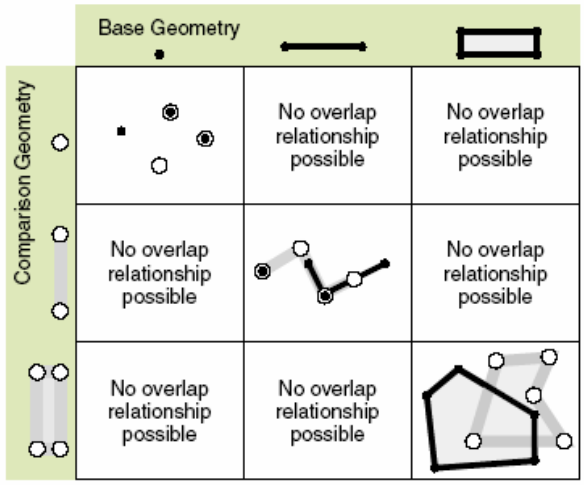


图9-17　Does the current geometry overlap the other geometry?

IRelationalOPerator::Equal 方法可以检测两个几何对象是否相等，具备相等关系的几何对象的形状相同，且组成部分也是相同的，它们的属性也相同，这意味着只有点与点、线与线、多边形与多边形才可能有相等关系。处于相等关系的两个几何对象的 M 值和 Z 值可以不一样，如果检测这两个属性是否一致，可以使用 IClone::IsEqual 方法。

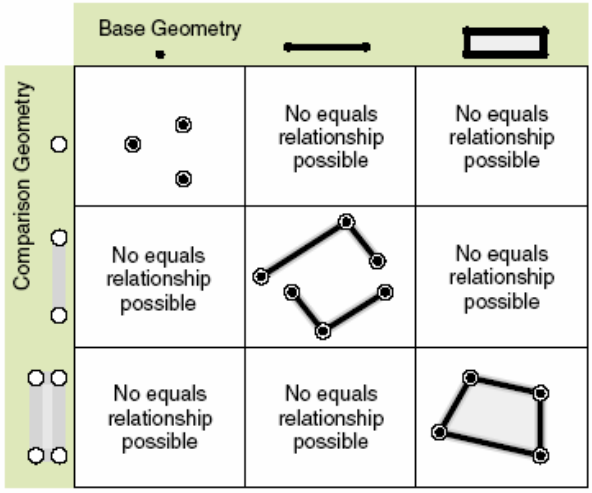


图9-18 Does the current geometry equal the other geometry?

IRelatinnaloPerator::Touch 方法可以检测两个几何对象的相接关系，所谓相接是指两个几何对象在它们的边界处相交，因此除了点与点外所有类型的高级几何对象都可能拥有这种空间关系。如点与多义线相接于点，多义线与多义线、多边形相接于线等。

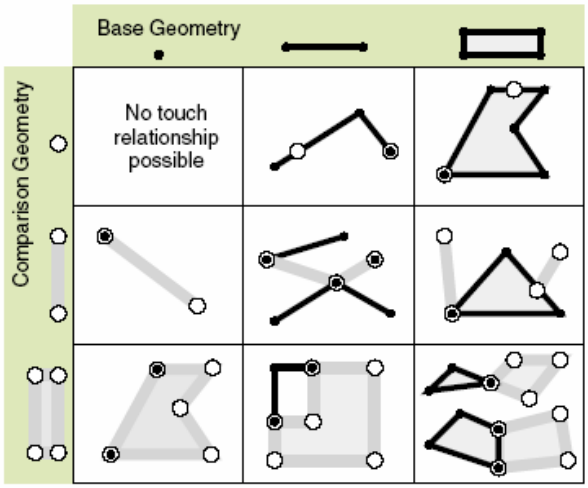


图9-19　Does the current geometry touch the other geometry?

空间关系也是几何对象中的基本属性，关于这些内容，读者可以查阅相关 CIS 理论书籍。

9.3.2 开发实例--查找某一多边形要素的所有邻接要素

该实例的功能是：先鼠标点击某一多边形，然后找出这一多边形的所有邻接多边形。

SelectAdjacentFeatures.cs 文件：

```
using System;
using System.Drawing;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.ToolbarControl;
using ESRI.ArcGIS.MapControl;
using ESRI.ArcGIS.TOCControl;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Framework ;
using System.Windows.Forms;
using ESRI.ArcGIS.ControlCommands;
```

```
using ESRI.ArcGIS.GeoDatabaseUI;
using ESRI.ArcGIS.esriSystem ;

namespace ControlsApplication
{
    public class SelectAdjacentFeatures: BaseTool
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap;
        IActiveView pActiveView;
        public SelectAdjacentFeatures()
        {
            base.m_caption = "SelectAdjacentFeatures";
            base.m_message = "SelectAdjacentFeatures";
            base.m_toolTip = "SelectAdjacentFeatures";
            base.m_category = "SpatialQuery";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }
        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }
        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            IFeatureLayer pFeatureLayer;
            IFeatureCursor pFeatureCursor;
            ISpatialFilter pSFilter;
            IFeature SrcFeature;
            IFeatureSelection pFeatureSelection;
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            //获取要查找的多边形图层
            int iLayerIndex;
            iLayerIndex = GetLayerByName("Select Polygon Layer");
            if (iLayerIndex < 0) return;
```

```
                pFeatureLayer = pMap.get_Layer(iLayerIndex) as IFeatureLayer;
                //获取鼠标点所在的多边形要素
                IPoint pPoint =
pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X, Y);
                pPoint.SpatialReference = pMap.SpatialReference;
                pFeatureSelection = pFeatureLayer as IFeatureSelection;
                pFeatureLayer.Selectable = true;
                pSFilter = new SpatialFilterClass();
                pSFilter.Geometry = pPoint;
                pSFilter.GeometryField = pFeatureLayer.FeatureClass.ShapeFieldName;
                pSFilter.SpatialRel = esriSpatialRelEnum.esriSpatialRelWithin;

                pFeatureSelection.SelectFeatures(pSFilter,esriSelectionResultEnum.esriSelect
                ionResultnew, true) ;
                ISelectionSet pFeatSet;
                pFeatSet = pFeatureSelection.SelectionSet;
                ICursor pCursor;
                pFeatSet.Search(null, true, out pCursor) ;
                pFeatureCursor = pCursor as IFeatureCursor;
                SrcFeature = pFeatureCursor.NextFeature();
                //获取鼠标点所在的多边形要素的所有邻接多边形要素
                 if (SrcFeature != null)
                {
                    pMap.ClearSelection();
                    IRelationalOperator pRelOperator;
                    pRelOperator = SrcFeature.Shape as IRelationalOperator;

                    //设置被选择要素的颜色
                    IRgbColor pColor;
                    pColor = new RgbColorClass();
                    pColor.Red = 255;
                    pFeatureSelection.SelectionColor = pColor;
                    //查找邻接要素，并置于地图的选择集中
                    pFeatureCursor = pFeatureLayer.Search(null, false);
                    IFeature pCompFeature ;
                    pCompFeature = pFeatureCursor.NextFeature();
                    while (pCompFeature != null)
                    {
```

```
                    if (pRelOperator.Touches(pCompFeature.Shape))
                        pMap.SelectFeature (pFeatureLayer,pCompFeature);
                    pCompFeature = pFeatureCursor.NextFeature();
                }
                //刷新视图
                pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeoSelection,
null, pActiveView.Extent);
            }
        }

        public int GetLayerByName(string strTitle )
        {
            IEnumLayer pEnumLayer;
            ILayer pLayer ;
            IListDialog pListDlg;
            bool boolOK ;
            if (pMap.LayerCount == 0) return -1;

            //Build the Layer enumeration
            pListDlg = new ListDialogClass();
            pEnumLayer = pMap.get_Layers(null, false);
            pEnumLayer.Reset();
            pLayer = pEnumLayer.Next();

            // Add layer names to the List Dialog
            while ( pLayer != null)
            {
                pListDlg.AddString(pLayer.Name);
                pLayer = pEnumLayer.Next();
            }

            // Get user's pick and return choice
            boolOK = pListDlg.DoModal(strTitle, 0, 0);
            return pListDlg.Choice;
        }

        private ILayer GetLayerByName (IMap pMap,string sLayerName)
        {
```

```
                int i ;
                ILayer pLayer;
                for( i = 0; i<= pMap.LayerCount - 1; i++)
                {
                    if (pMap.get_Layer(i).Name == sLayerName)
                    {
                        pLayer = pMap.get_Layer(i);
                        return pLayer;
                    }
                }
                return null;
            }
        }
}
```

在 Form 的 Load 事件中键入下列代码：

```
private void Form1_Load(object sender, System.EventArgs e)
{
  ……
  axToolbarControl1.AddItem( new
  SelectAdjacentFeatures(),0,-1,false,0,esriCommandStyles.esriCommandStyleTextOnly);
  …….
  }
```

9.4 IProximityOperator 接口

IProximityOperator 接口定义的方法：QueryNearestPoint、ReturnDistance、ReturnNearestPoint，主要用于得到两个几何对象之间的距离或得到一个给定点到某个几何对象的最近点之间的距离。在很多 GIS 系统中都需要拥有“长度量算”的功能，而这个功能就可以使用该接口定义的 ReturnDistance 方法.

IProximityOperator 接口被众多几何对象的类所实现，包括 BezierCurve、CircularArc、EllipticArc、Envelope、Line、Multipoint、Point、Polygon 和 Polyline 对象。

IProximityOperator::QueryNearestPoint 和 ReturnNearestPoint 方法都可以用于查询一个几何对象上的点对象，如图 9-20 所示，

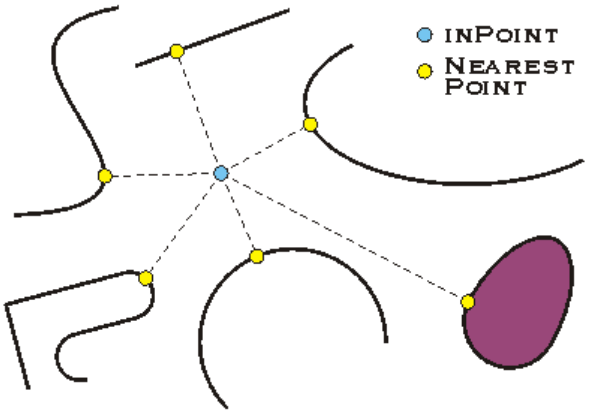


图 9-20 最近点查询

上图中中间的点对象与周围的几何对象都分别存在一个最近点，而这个最近点就可以使用上面的两个方法查询和获得。前一种方法会“拷贝”这个最近点到几何对象上，而后一种方法会返回这个最近点。

IProximityOperator::ReturnDistance 方法则可以返回两个几何对象之间的最短距离，下面是一个简单的过程，用于返回在 MapControl 上先后点击的两点之间的距离：

Distance2Points.cs

```
using System;
using System.Windows.Forms;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    /// <summary>
    /// NearestPoint 的摘要说明。
    /// </summary>
    public class Distance2Points : BaseTool
    {
        IMap pMap;
```

```
        IActiveView pActiveView;
        IPoint pPointStd;

        IHookHelper m_hookHelper = new HookHelperClass();

        public Distance2Points()
        {
            base.m_caption = "Distance2Points";
            base.m_message = "Distance between 2 Points";
            base.m_toolTip = "Distance between 2 Points";
            base.m_category = "SpatialQuery";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));

        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnMouseDown(int Button, int Shift, int X, int Y)
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;

            //产生一个当前的点击的点对象
            IPoint pPoint;
            pPoint = new PointClass();
            pPoint.SpatialReference = pMap.SpatialReference ;
            pPoint = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(X, Y);


            // 判断该点是否为第一个点
            if (pPointStd == null)
```

```
                {
                    pPointStd = pPoint;
                    //在屏幕上绘制该点
                    IScreenDisplay screenDisplay = pActiveView.ScreenDisplay;
                    screenDisplay.StartDrawing(screenDisplay.hDC,
(short)esriScreenCache.esriNoScreenCache);
                    screenDisplay.SetSymbol(new SimpleMarkerSymbolClass());
                    screenDisplay.DrawPoint(pPointStd);
                    screenDisplay.FinishDrawing();
                }
                else
                {
                    IProximityOperator pProximity;
                    pProximity = pPointStd as IProximityOperator;
                    double pDistance;
                    pDistance = pProximity.ReturnDistance(pPoint);
                    //标签对象上出现两点之间的距离
                    MessageBox.Show(pDistance.ToString());
                }
            }
        }
}
```

上面的例子非常简单，连两点之间的“橡皮筋”都没有出现，读者可以根据本书的IDisplayFeedback 的内容，将这个例子变得更丰富。

9.5 叠加分析

叠加分析是地理信息系统提取空间隐含信息的常用手段之一。它与位置查询不同，通过叠加分析可生成新图层，而且有些分析操作，输入图层的要素（线或多边形）会被叠加图层的边界所分割。叠加分析根据数据层组织方式不同，可分为矢量图层叠加分析和栅格图层叠加分析。

9.5.1 矢量图层叠加分析

9.5.1.1概述

矢量图层叠加分析可分为点与多边形叠加分析、线与多边形叠加分析、多边形与多边形叠加分析。

(I) 点与多边形叠加分析

点与多边形叠加分析是简单的多边形对点的包含关系分析，利用它可找到哪个点落入哪个多边形中。例如：统计每个人口普查区内有多少杂货店。新的输出杂货店点要素类将带有人口普查区的属性，可通过对商店进行查询，找出它落在哪个人口普查区中，或者统计每个人口普查区内商店的总数。

(II) 线与多边形叠加分析

线与多边形叠加分析是比较线上坐标与多边形坐标的关系，判断线是否落入多边形内的分析。利用它可查找一个线要素类和一个多边形要素类共在的地理空间。例如：利用一个河流要素类和一个宗地要素类，可找出哪条河流落在每块宗地内，以及确定这条河流有多大部分落在这块宗地内。因为叠加要素物理地分割了输入要素，所以此分析结果比位置查询精确。

(III) 多边形与多边形叠加分析

多边形与多边形叠加分析是将两个或多个多边形进行叠加产生一个新多边形的操作。利用它可研究两要素类之间的公共区域，例如：一个宗地多边形要素类和一个洪水灾害区多边形要素类叠加，可获取每幅宗地是否被淹以及受淹的准确面积。

根据实际应用的需要，矢量图层叠加分析包括叠置求交（Interset）和叠置求和（Union）。

1、 叠置求交（Interset）

叠置求交是将两个图层中共同区域内的要素和属性组合到第三图层，其输出结果要素类型与输入图层几何类型保持一致，且既包含输入图层属性也包含叠加图层属性。输入图层可以是线要素层或者多边形要素层，但叠加图层必须是多边形要素层。它的主要用途是组合两图层的公共区域到第三图层，例如：为判断某区域内公路下的土壤特征，可将土壤图层和公路图层进行叠置求交操作，生成新图层，其中道路将被土壤多边形的边界切割，并继承土壤的属性。根据输出结果，可发现哪条道路建立在易侵蚀土壤上，危险系数较大。

2、 叠置求和（Union）

叠置求和是将两个多边形图层中所有要素合并到一个新图层中，输出结果包含所有要素和属性字段。注意：输入图层和叠加图层都必须是多边形图层。

它的主要用途是将两个多边形要素层合并到新图层，例如：为寻找植被和土壤

之间的关系，可将植被图层和土壤图层进行叠置求和操作，生成新的多边形图层，此图层中植被多边形被土壤多边形分割，并继承了土壤的属性。可通过新图层查询植被区，了解其具有的土壤特征，也可生成表达植被和土壤之间关系的报表。

9.5.1.2IBasicGeoProcessor 接口

矢量图层叠置分析需要用到的主要接口是 IBasicGeoProcessor，它提供了基本的空间数据处理的方法和属性，其中包括叠置求交（Interset）及叠置求和（Union）。组件类 BasicGeoprocessor 封装了该接口。该接口共包括 5 个方法(Intersect、Union、Clip、Dissolve 和 Merge) 和 2 个属性(SpatialReference 和 CancelTracker)。

Intersect 用于两个图层要素的叠加求交，Union 用于两个图层要素的叠加求和。这两个方法的参数如下：

```
public IFeatureClass Intersect (
        ITable inputTable,
        bool useSelectedInput,
        ITable overlayTable,
        bool useSelectedOverlay,
        double Tolerance,
        IFeatureClassName outputName
        );
public IFeatureClass Union (
        ITable inputTable,
        bool useSelectedInput,
        ITable overlayTable,
        bool useSelectedOverlay,
        double Tolerance,
        IFeatureClassName outputName

        );
```

Merge 方法用于合并同类要素，其参数如下：

```
public IFeatureClass Merge (
        IArray tables,
        ITable fieldsTable,
        IFeatureClassName outputName
        );
```

Clip 方法用于要素的裁剪，Dissolve 用于要素的融合。

9.5.1.3开发实例

(I) 叠加求交

该实例的功能是实现了地图上两指定矢量图层之间的叠置求和，将其结果保存为一 shapefile 文件，并加入当前地图中。

Intersect.cs

```
using System.Windows.Forms;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.DataSourcesFile;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    /// <summary>
    /// Intersect 的摘要说明。
    /// </summary>
    public class Intersect:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap ;
        IActiveView pActiveView;
        public Intersect()
        {
            base.m_caption = "Intersect";
            base.m_message = "Intersect";
            base.m_toolTip = "Intersect";
            base.m_category = "SpatialAnalysis";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
```

```
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            // Get the input layer and feature class
            ILayer pLayer;
            pLayer = pMap.get_Layer(0);
            IFeatureLayer pInputFeatLayer ;
            pInputFeatLayer = pLayer as IFeatureLayer;
            //Use the Itable interface from the Layer (not from the FeatureClass)
            ITable pInputTable;
            pInputTable = pLayer as ITable;
            // Get the input feature class.
            //The Input feature class properties, such as shape type, will be needed for the
output
            IFeatureClass pInputFeatClass;
            pInputFeatClass = pInputFeatLayer.FeatureClass ;
            //Get the overlay layer
            //Use the Itable interface from the Layer (not from the FeatureClass)
            pLayer = pMap.get_Layer(1);
            IFeatureLayer pOverlayFeatLayer ;
            pOverlayFeatLayer = pLayer as IFeatureLayer;
            ITable pOverlayTable ;
            pOverlayTable = pOverlayFeatLayer as ITable;
            //Error checking
            if (pInputTable == null)
            {
                MessageBox.Show("Table QI failed");
                return;
            }
            if (pOverlayTable == null)
```

```
                {
                    MessageBox.Show("Table QI failed");
                    return;
                }
                // Define the output feature class name and shape type (taken from the
properties of the input feature class)
                IFeatureClassName pFeatClassName ;
                pFeatClassName = new FeatureClassNameClass();
                pFeatClassName.FeatureType = esriFeatureType.esriFTSimple;
                pFeatClassName.ShapeFieldName = "Shape" ;
                pFeatClassName.ShapeType = pInputFeatClass.ShapeType ;
                //Set output location and feature class name
                IWorkspaceName pNewWSName;
                pNewWSName = new WorkspaceNameClass();
                pNewWSName.WorkspaceFactoryProgID =
"esriDataSourcesFile.ShapefileWorkspaceFactory";
                pNewWSName.PathName = @"d:\temp" ;
                IDatasetName pDatasetName;
                pDatasetName = pFeatClassName as IDatasetName;
                pDatasetName.Name = "Intersect_result";
                pDatasetName.WorkspaceName = pNewWSName;
                //Set the tolerance
                double tol = 0.1;
                //Perform the intersect
                IBasicGeoprocessor pBGP;
                pBGP = new BasicGeoprocessorClass();
                pBGP.SpatialReference = pMap.SpatialReference;
                IFeatureClass pOutputFeatClass;
                pOutputFeatClass = pBGP.Intersect(pInputTable, false, pOverlayTable, false,
tol, pFeatClassName) ;
                //Add the output layer to the map
                IFeatureLayer pOutputFeatLayer;
                pOutputFeatLayer = new FeatureLayerClass();
                pOutputFeatLayer.FeatureClass = pOutputFeatClass;
                pOutputFeatLayer.Name = pOutputFeatClass.AliasName;
                pMap.AddLayer(pOutputFeatLayer);
            }
        }
```

```
}
```

测试代码：

```
private void mnuIntersect_Click(object sender, System.EventArgs e)
{
    ICommand cmd ;
    cmd = new Intersect();
    cmd.OnCreate(axMapControl1.GetOcx());
    if (cmd.Enabled) cmd.OnClick();
}
```

(II) 同类要素或图层的合并

该实例的功能是合并两个具有相同几何类型的图层或相同类型的要素类。可以用于图幅的合并。

Merge.cs

```
using System.Windows.Forms;
using ESRI.ArcGIS.DataSourcesFile;
using ESRI.ArcGIS.Utility.BaseClasses;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.ControlCommands;

namespace ControlsApplication
{
    public class MergeCMD:BaseCommand
    {
        IHookHelper m_hookHelper = new HookHelperClass();
        IMap pMap ;
        IActiveView pActiveView;

        public MergeCMD()
        {
            base.m_caption = "Merge";
            base.m_message = "Merge";
            base.m_toolTip = "Merge";
```

```
            base.m_category = "SpatialAnalysis";
            base.m_enabled = true;
            //base.m_bitmap = new
System.Drawing.Bitmap(GetType().Assembly.GetManifestResourceStream(GetType(),
"x.bmp"));
        }

        public override void OnCreate(object hook)
        {
            m_hookHelper.Hook = hook;
        }

        public override void OnClick()
        {
            pActiveView = m_hookHelper.ActiveView;
            pMap = m_hookHelper.FocusMap;
            IFeatureClass pOutputFeatClass;
            string pathLayer1 = @"C:\arcgis\ArcTutor\usa";
            string pathLayer2 = @"D:\ArcTutor\usa";
            string pathMergeResult = @"d:\temp";
            string nameLayer1 = "states";
            string nameLayer2 = "states";
            string nameMergeResult = "MergedStates";
            pOutputFeatClass = Merge(pathLayer1, pathLayer2, pathMergeResult,
nameLayer1, nameLayer2, nameMergeResult);
            //Add the output layer to the map
            IFeatureLayer pOutputFeatLayer;
            pOutputFeatLayer = new FeatureLayerClass();
            pOutputFeatLayer.FeatureClass = pOutputFeatClass;
            pOutputFeatLayer.Name = pOutputFeatClass.AliasName;
            pMap.AddLayer(pOutputFeatLayer);
        }

        private IFeatureClass Merge(string pathLayer1, string pathLayer2, string
pathMergeResult, string nameLayer1, string nameLayer2,string nameMergeResult)
        {
            //分别读取图层一,图层二到FeatureClass和Table中
            IWorkspaceFactory pWorkspaceFactory;
```

```
                IFeatureWorkspace pWorkspace1;
                IFeatureWorkspace pWorkspace2;
                IFeatureClass pFirstFeatClass;
                IFeatureClass pSecondFeatClass;
                ITable pFirstTable;
                ITable pSecondTable;
                IFeatureLayer pFeatLayer1;
                pFeatLayer1 = new FeatureLayerClass();
                IFeatureLayer pFeatLayer2;
                pFeatLayer2 = new FeatureLayerClass();
                pWorkspaceFactory = new ShapefileWorkspaceFactoryClass();
                pWorkspace1 = pWorkspaceFactory.OpenFromFile(pathLayer1, 0) as
IFeatureWorkspace;
                pWorkspace2 = pWorkspaceFactory.OpenFromFile(pathLayer2, 0) as
IFeatureWorkspace;
                pFirstFeatClass = pWorkspace1.OpenFeatureClass(nameLayer1);
                pSecondFeatClass = pWorkspace2.OpenFeatureClass(nameLayer2);
                pFeatLayer1.FeatureClass = pFirstFeatClass;
                pFirstTable = pFeatLayer1 as ITable;
                pFeatLayer2.FeatureClass = pSecondFeatClass;
                pSecondTable = pFeatLayer2 as ITable;;
                // 检查错误
                if (pFirstTable == null)
                {
                    MessageBox.Show("Table QI failed");
                    return null;
                }
                if (pSecondTable == null)
                {
                    MessageBox.Show("Table QI failed");
                    return null;
                }
                // 定义输出要素类名称和shape类型
                IFeatureClassName pFeatClassName;
                pFeatClassName = new FeatureClassNameClass();
                pFeatClassName.FeatureType = esriFeatureType.esriFTSimple;
                pFeatClassName.ShapeFieldName = "Shape";
                pFeatClassName.ShapeType = pFirstFeatClass.ShapeType;
```

```
                // 定义输出shapefile位置与名称
                IWorkspaceName pNewWSName;
                pNewWSName = new WorkspaceNameClass();
                pNewWSName.WorkspaceFactoryProgID =
"esriDataSourcesFile.ShapefileWorkspaceFactory";
                pNewWSName.PathName = pathMergeResult;
                IDatasetName pDatasetName;
                pDatasetName = pFeatClassName as IDatasetName;
                pDatasetName.Name = nameMergeResult;
                pDatasetName.WorkspaceName = pNewWSName;
                // 定义Merge参数
                IArray inputArray;
                inputArray = new ArrayClass();
                inputArray.Add(pFirstTable);
                inputArray.Add(pSecondTable);
                // 进行Merge操作
                IBasicGeoprocessor pBGP;
                pBGP = new BasicGeoprocessorClass();
                IFeatureClass pOutputFeatClass;
                pOutputFeatClass = pBGP.Merge(inputArray, pFirstTable, pFeatClassName);
                return pOutputFeatClass;
            }
        }
}
```

9.5.2 栅格图层的叠加分析

9.5.2.1概述

栅格数据结构具有空间信息隐含但属性信息明显的特点，故可通过各种各样的函数关系将不同层的栅格图层进行叠加运算以揭示某种空间现象或者空间过程，例如：山体滑坡和高度、倾角、植被等因素有关，可根据多年统计的经验方程，把高程、倾角、植被等分别作为栅格图层输入，然后通过数学运算得到山体滑坡危险系数评估图。

栅格图层叠置分析主要包含以下几种类型：

（1）基于常数进行的代数运算；

（2）基于数学变换进行的数学变换（指数、对数、三角变换等）；

（3）基于多个栅格要素层的运算，包括代数运算（加、减、乘、除、乘方等）和逻辑运算（与、或、非、异或等）。

该模块实现了地图上两指定栅格图层叠置相加的功能。

9.5.2.2RasterMathops 组件类

RasterMathops 组件类提供了栅格运算的操作机制。其主要包括 IMathOP、ILogicalOP、ITrigOP 等。

IMathop 接口接口提供了栅格数学运算的方法。

ILogicalOP 接口提供了栅格逻辑运算方法。

ITrigOP 接口提供了栅格三角运算方法。

以下代码展示了两个栅格数据集的简单相加：

```
private void RasterPlus()
{
    //Create the MathOps object
    IMathOp pMathOp;
    pMathOp = new RasterMathOpsClass();
    // Declare the input geoDataset1 object
    IGeoDataset pInputDataset1;
    //Calls function to open a raster dataset from disk
    pInputDataset1 = OpenRasterDataset(@"D:\SpatialData", "inputraster1") as
IGeoDataset;
    // Declare the input geoDataset2 object
    IGeoDataset pInputDataset2;
    // Calls function to open a raster dataset from disk
    pInputDataset2 = OpenRasterDataset(@"D:\SpatialData", "inputraster2") as
IGeoDataset;
    // Declare the output raster object
    IGeoDataset pOutputRaster;
    // Calls the method
    pOutputRaster = pMathOp.Plus(pInputDataset1, pInputDataset2);
}
```

几何对象之间可以进行空间运算，它们也存在空间关系。拓扑关系是几何对象模型中重要的部分，在 ArcGIS 中，常常依赖这种拓扑关系的检测来判断大量地理数据的有效性和完整性，它通过在 GeoDatabase 中可以专门创建一个拓扑关系类实现。读者需要认真理解几何对象之间的组合关系和空间关系。

10 空间数据编辑--MapEditing 实例剖析

空间数据编辑是 GIS 的基本功能之一，使用 ArcObjects 实现该功能是一项比较复杂的工作。本章主要依据 ArcGIS 提供的 MapEditing 例子来阐述 ArcObjects 提供的空间数据编辑功能。

10.1 IWorkspaceEdit 接口

IWorkspaceEdit 接口是 ArcObjects 实现空间数据编辑功能的重要接口，它可以让程序启动或停止一个编辑流程，在这个编辑流程内，地理数据库中的数据可以进行更新操作。由于 GIS 的编辑操作往往都是长事务过程，因而需要确保用户在这段时间内所做的编辑能够恢复。该接口定义了 13 个方法，没有属性，如图 10-1 所示。



图 10-1 IWorkspaceEdit 接口

一个编辑流程是使用 StartEditing 方法开始的，这个方法的 withUndoRedo 参数用来确定工作空间是否支持“恢复 / 取消恢复”的操作。如果希望在编辑过程中能够提供“恢复 / 取消恢复”的功能，那么这个数据库中所有对象的相关的改变都需要放在一个流程内。在启动编辑后，程序员可以使用 StartEditOperation 方法开启编辑操作，这个编辑操作相比编辑流程而言是短时间的。

如果在编辑过程中出现了异常，程序可以使用 AbortEditoperation 方法来取消所有的编辑操作，以免发生不可恢复的破坏。

在完成编辑后，用户可以使用 StopEditoperation 方法来确保编辑操作的完成。

IWorkspaceEdit::UndoEditoperation 方法可以用于编辑状态的回滚操作，如果发现编辑过程有误，通过执行这个方法可以恢复到最近变化前的状态。

在整个编辑流程完成后，可以使用 StopEditing 方法来完成编辑。当执行完这个方法后，就意味着不能再进行“恢复／取消恢复”了。

因此，在数据编辑过程中，如果需要提供“恢复／取消恢复”（UNDO）功能，所有的数据操作必须在一个编辑流程内的编辑操作中，这个编辑流程保证了 UNDO 的可能性。

下面是一个关于要素编辑流程的例子。

```
private void WorkspaceEdit()
{
    IWorkspaceFactory pWorkspaceFactory;
    pWorkspaceFactory = new AccessWorkspaceFactoryClass();
    IFeatureWorkspace pFeatureWorkspace;
    pFeatureWorkspace = pWorkspaceFactory.OpenFromFile(@"D:\Usa.mdb", 0)
as IFeatureWorkspace;
    //打开文件
    IFeatureClass pFeatureClass;
    //获得要编辑的要素类
    pFeatureClass = pFeatureWorkspace.OpenFeatureClass("States");
    IWorkspaceEdit pWorkspaceEdit;
    pWorkspaceEdit = pFeatureWorkspace as IWorkspaceEdit;
    IFeature pFeature ;
    bool bHasEdits = true;
    //开始编辑流程
    pWorkspaceEdit.StartEditing(true);
    //启动编辑操作
    pWorkspaceEdit.RedoEditOperation();
    //取出第一个要素
    pFeature = pFeatureClass.GetFeature(1);
    //删除这个要素
    pFeature.Delete();
    //停止编辑操作
    pWorkspaceEdit.StopEditOperation();
    DialogResult iResponse;
    iResponse = MessageBox.Show("Edit Operation", "Undo operation?",
```

```
MessageBoxButtons.YesNo);
                if (iResponse == DialogResult.Yes)
                {
                    //如果需要“恢复”，则执行Undo操作
                    pWorkspaceEdit.UndoEditOperation();
                }
                pWorkspaceEdit.HasEdits(ref bHasEdits);
                //如果已经编辑，则启动停止编辑流程的命令
                if (bHasEdits)
                {
                    iResponse = MessageBox.Show("Edit Operation","Save edits?",
MessageBoxButtons.YesNo);
                    if (iResponse == DialogResult.Yes)
                    {
                        pWorkspaceEdit.StopEditing(true);
                    }
                    else
                    {
                        pWorkspaceEdit.StopEditing(false);
                    }
                }
            }
```

10.2 MapEditing 功能剖析

10.2.1功能概述

MapEditing 实例的功能是实现空间数据的编辑修改、删除、新建、恢复等，其运行界面如图 10-2 所示。整个界面是一个窗体（Form），命名为 frmMapEditing；界面的主体部分是一个 MapControl，命名为 axMapControl1，顶端布置了 4 个命令按钮：Start Editing、Delete、Undo 和 Stop Editing，它们的名称分别为 btnStartEditing、btnDelete、btnUndo 和 btnStopEditing，功能分别是启动编辑流程、删除选择的要素、恢复操作和终止编辑流程；此外，还有两个下拉列表框：cboTasks 和 cboLayers，前者用于选择、显示任务列表，这里有 3 个任务：Edit Feature、Create Feature 和 Select Feature；cboLayers 下拉列表框用于选择、显示当前地图中的图层，cboTasks 下拉列表框中的任务是针对某个图层的。

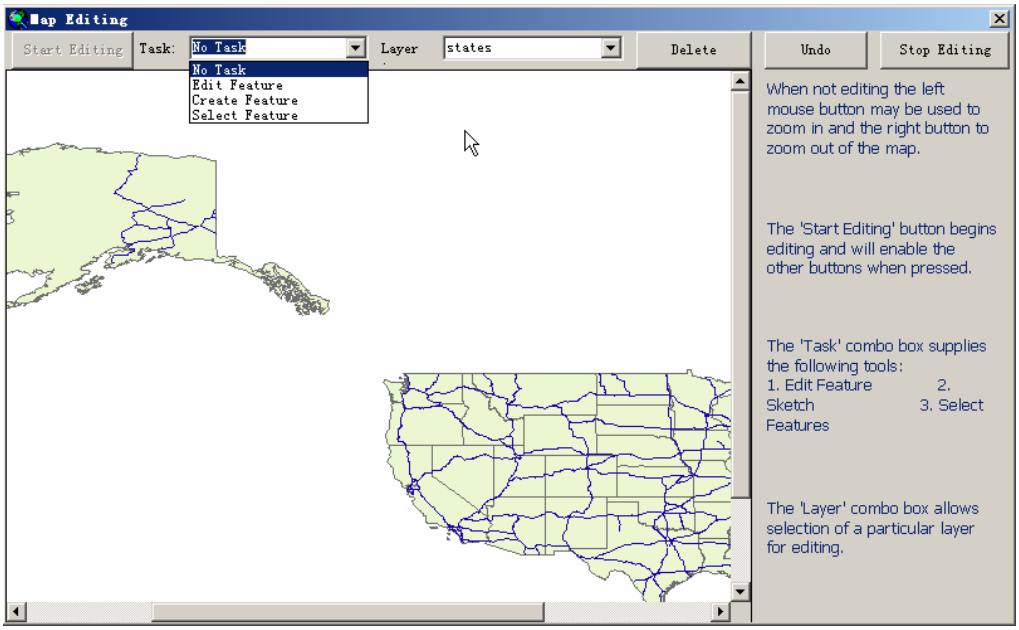


图 10-2 MapEditing 运行界面

10.2.2 代码分析

10.2.2.1 全局变量

空间数据的编辑需要在 Workspace 的一个编辑流程中完成，有时还需要撤销某一操作，这些工作不是简单地点击一次鼠标或点击一个命令按钮来完成的，而是相互关联的，所以需要设置一些全局变量来确保编辑工作的顺利完成。在 MapEditing 中的主要全局变量有：

```
private IMapControl3 m_MapControl;   //地图控件
private IMap m_pMap;                 //地图控件中的地图
private ILayer m_pCurrentLayer;      //地图中的当前操作图层
private IFeature m_pEditFeature;     //当前编辑的要素
private IPoint m_pPoint;             //当前鼠标点击位置
private IDisplayFeedback m_pFeedback;   //用于地图显示
private IPointCollection m_pPointCollection;   //当前要素的点集
private bool m_bInUse;       //是否处于同一个编辑状态
private bool m_bEditingFtr;   //是否为编辑要素状态
private bool m_bSketching;    //是否为新建要素状态
private bool m_bSelecting;   //是否为选择要素状态
```

10.2.2.2 初始化工作

程序运行时的初始化工作一般是在主窗体的Load事件中完成的，在MapEditing实例初始化代码如下，其功能是获得两个全局变量m_MapControl、m_pMap的初始值，初始化两个下拉列表框，并设置4个命令按钮的状态。

```
private void frmMapEditing_Load(object sender, System.EventArgs e)
{
  // Setup the application
  m_MapControl = (IMapControl3) axMapControl1.GetOcx(); //获得地图控件
  m_pMap = m_MapControl.Map;
  cboTasks.Items.Add( "No Task");
  cboTasks.Items.Add( "Edit Feature");
  cboTasks.Items.Add ("Create Feature");
  cboTasks.Items.Add( "Select Feature");
  cboTasks.SelectedIndex = 0;
  LoadLayers();    //加载地图图层，并用这些图层来初始化cboLayers下拉列表框
  SetControlStates();  //设置4个命令按钮的状态
}

private void LoadLayers()
{
  //Find sample data
  string sFilePath = @"C:\Program Files\ArcGIS\DeveloperKit\samples\data\usa\";
  // Load layers from shapefiles
  m_MapControl.AddShapeFile(sFilePath, "states.shp");
  layerInfo li = new layerInfo();
  li.layerName = "states";
  li.layerAttr = GetFileAttributes(sFilePath + "states.shp");
  cboLayers.Items.Add(li);
  m_MapControl.AddShapeFile(sFilePath, "ushigh.shp");
  layerInfo li1 = new layerInfo();
  li1.layerName = "ushigh";
  li1.layerAttr = GetFileAttributes(sFilePath + "ushigh.shp");
  cboLayers.Items.Add(li1);
  m_MapControl.AddShapeFile(sFilePath, "wind.shp");
  layerInfo li2 = new layerInfo();
  li2.layerName = "wind";
```

```
        li2.layerAttr = GetFileAttributes(sFilePath + "wind.shp");
        cboLayers.Items.Add(li2);

        cboLayers.Visible = true;
        cboLayers.SelectedIndex = 0; //将触发cboLayers的SelectedIndexChanged事件，获得当前编辑图层
    }
```

10.2.2.3 启动编辑流程

要对空间数据进行编辑，首先要通过 IWorkspaceEdit:: StartEditing 方法启动编辑流程，编辑当前图层 m_pCurrentLayer。

```
    private void btnStartEditing_Click(object sender, System.EventArgs e)
    {
        StartEditing();
        SetControlStates();
    }

    private void StartEditing()
    {
        // Check edit conditions before allowing edit to start
        if (m_pCurrentLayer == null) return;
        if ( (IGeoFeatureLayer)m_pCurrentLayer == null) return;
        IFeatureLayer pFeatureLayer = (IFeatureLayer)m_pCurrentLayer;
        IDataset pDataset = (IDataset)pFeatureLayer.FeatureClass;
        if (pDataset== null) return;

        // Start editing, making sure that undo/redo are enabled
        IWorkspaceEdit pWorkspaceEdit = (IWorkspaceEdit)pDataset.Workspace;
        if (!pWorkspaceEdit.IsBeingEdited())
        {
            pWorkspaceEdit.StartEditing(true);
            pWorkspaceEdit.EnableUndoRedo();
        }
    }
```

10.2.2.4 地图控件的鼠标事件处理分析

空间数据的编辑，不同的编辑任务需要对地图控件的不同事件作出相应的处理。对于“选择要素”，只需要对地图控件的 OnMouseDown 事件作出响应；对于“编辑要素”，需要对地图控件的 OnMouseDown、OnMouseMove 、OnMouseUp 事件作出响应；对于“新建要素”，需要对地图控件的 OnMouseDown、OnMouseMove 和 OnDoubleClick 事件作出响应。MapEditing 中的鼠标事件处理代码如下：

```
private void OnMouseDown(object sender, IMapControlEvents2_OnMouseDownEvent e)
{
  //首先判断当前用户选择了何种任务，并去完成相应的任务；如果没有选择任何任务，即cboTasks的选项为"No Task"时，该事件处理程序用于放大、缩小地图，鼠标左键用于放大，其它键（右键或中间键）用于缩小。

  m_pMap.ClearSelection();  //清除地图中已选择的要素
  if( m_bSketching)
    SketchMouseDown(e.x, e.y);  //新建要素任务MouseDown处理
  else if (m_bSelecting)
    SelectMouseDown(e.x, e.y);    // 选择要素任务MouseDown处理
  else if (m_bEditingFtr)
    EditFeature(e.x, e.y);            //编辑要素任务MouseDown处理
  else
  {
    // Zoom in/out depending on which button was pressed
    IActiveView pActiveView = (IActiveView)m_pMap;
    IEnvelope pEnvelope = pActiveView.Extent;
    ESRI.ArcGIS.Geometry.Point pnt = new ESRI.ArcGIS.Geometry.Point();
    IPoint iPnt = pnt;
    iPnt.X = e.mapX;
    iPnt.Y = e.mapY;
    pEnvelope.CenterAt(iPnt);
    if (e.button == 1) // left button
      pEnvelope.Expand(0.5, 0.5, true);    //放大视图
    else
      pEnvelope.Expand(2, 2, true);          //缩小视图
    pActiveView.Extent = pEnvelope;
    pActiveView.Refresh();
  }
```

```
}

private void OnMouseMove(object sender, IMapControlEvents2_OnMouseMoveEvent e)
{
  if (m_bSketching)
    SketchMouseMove(e.x, e.y);    //新建要素任务MouseMove处理
  else if (m_bEditingFtr)
    FtrEditMouseMove(e.x, e.y);    //编辑要素任务MouseMove处理
}

private void OnMouseUp(object sender, IMapControlEvents2_OnMouseUpEvent e)
{
  if (m_bEditingFtr)
    EndFtrEdit(e.x, e.y);       //结束编辑要素任务处理
}

private void OnDoubleClick(object sender, IMapControlEvents2_OnDoubleClickEvent e)
{
  if (m_bSketching)
    EndSketch();                //结束新建要素任务处理
}
```

10.2.2.5 下拉列表框事件处理分析

本实例用了两个下拉列表框 cboTasks 和 cboLayers，分别用于选择编辑任务和选择编辑图层。这里只需对下拉列表框的 SelectedIndexChanged 事件作出响应，具体代码如下：

```
private void cboTasks_SelectedIndexChanged(object sender, System.EventArgs e)
{
  //该事件处理程序的目的是：根据不同的编辑任务，设置相应的鼠标形状，并设置相应的全局变量为true。
  m_MapControl.MousePointer = esriControlsMousePointer.esriPointerDefault;
  m_bEditingFtr = false;
  m_bSketching = false;
```

```
        m_bSelecting = false;
        // Select an operation and mouse pointer depending on the user choice
        switch(cboTasks.SelectedIndex)
        {
          case 0:
            // Do Nothing - the tool has already been reset
            break;
          case 1:
            m_MapControl.MousePointer = esriControlsMousePointer.esriPointerCrosshair;
            m_bEditingFtr = true;
            break;
          case 2:
            m_MapControl.MousePointer = esriControlsMousePointer.esriPointerPencil;
            m_bSketching = true;
            break;
          case 3:
            m_MapControl.MousePointer = esriControlsMousePointer.esriPointerCrosshair;
            m_bSelecting = true;
            break;
         }
      }


      private void cboLayers_SelectedIndexChanged(object sender, System.EventArgs e)
      {
        //该事件处理程序的目的是：根据用户的选择，获得地图的当前编辑图层，同时判断该图层对应的数据文件是否为“只读”，如果是，则该图层不能编辑修改、新建要素等。
        if(m_pMap == null)    return;
        // Clear any existing selection
        m_pMap.ClearSelection();
        IActiveView ipActiveView;
        ipActiveView = (IActiveView)m_pMap;
        ipActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeoSelection, null, null);
        layerInfo li = (layerInfo)cboLayers.SelectedItem;
        // Rethe current layer pointer
        for (int i = 0; i <    m_pMap.LayerCount; i++)
        {
          if(m_pMap.get_Layer(i).Name == li.layerName)
```

```
        {
            m_pCurrentLayer = m_pMap.get_Layer(i); //获得地图的当前图层
            break;
        }
    }

    // Enable/disable controls according to whether the layer is read-only
    if((li.layerAttr & 0x00000001) > 0) // FILE_ATTRIBUTE_READONLY
    {
        DisableControls();
        cboTasks.SelectedIndex = 0;
        if (Form.ActiveForm != null)
            Form.ActiveForm.Text = "Map Editing    *** Read-Only Layer ***";
    }
    else
    {
        SetControlStates();
        if (frmMapEditing.ActiveForm != null)
            frmMapEditing.ActiveForm.Text = "Map Editing";
    }
}
```

10.2.2.6 删除要素处理程序

```
private void btnDelete_Click(object sender, System.EventArgs e)
    {
        DeleteSelectedFeatures();
    }

    private void DeleteSelectedFeatures()
    {
        if (m_pCurrentLayer == null) return;
        // If there are no features currently selected then nothing to do
        IFeatureCursor pFeatureCursor = GetSelectedFeatures(); //获得当前图层的选择要素
        if (pFeatureCursor == null) return;
        m_pMap.ClearSelection();
```

```
        // Loop over the selected features deleting each in turn
        IWorkspaceEdit pWorkspaceEdit = GetWorkspaceEdit();   //获得当前图层的工作空间
        pWorkspaceEdit.StartEditOperation();
        IFeature pFeature = pFeatureCursor.NextFeature();
        while (pFeature != null)
        {
            pFeature.Delete();
            pFeature = pFeatureCursor.NextFeature();
        }
        pWorkspaceEdit.StopEditOperation();
        IActiveView pActiveView = (IActiveView)m_pMap;
        pActiveView.Refresh();
    }
```

10.2.2.7 Undo/Redo 代码分析

Undo/Redo 操作是任何数据编辑中常有的功能。同样，空间数据编辑也应该具有这样的功能，以免用户的误操作而带来无可挽回的损失。

```
 private void btnUndo_Click(object sender, System.EventArgs e)
    {
        UndoEdit();
    }

    private void UndoEdit()
    {
        // Check that editing is possible
        if (m_pCurrentLayer == null) return;
        IFeatureLayer pFeatureLayer = (IFeatureLayer)m_pCurrentLayer;
        IDataset pDataset = (IDataset)pFeatureLayer.FeatureClass;
        if (pDataset == null) return;

        /// If edits have taken place then roll-back the last one
        IWorkspaceEdit pWorkspaceEdit = (IWorkspaceEdit)pDataset.Workspace;
        bool bHasUndos = false;
        pWorkspaceEdit.HasUndos(ref bHasUndos); //如果是Redo，换为HasRedos
        if (bHasUndos)    pWorkspaceEdit.UndoEditOperation();//如果是Redo，换为
RedoEditOperation
```

```
    IActiveView pActiveView = (IActiveView)m_pMap;
    pActiveView.Refresh();
}
```

10.2.2.8 结束编辑流程

完成所有编辑工作后，需要结束编辑流程，并作相应的后续处理。

```
private void btnStopEditing_Click(object sender, System.EventArgs e)
{
    StopEditing();
    SetControlStates();
    cboTasks.SelectedIndex = 0;
}

private void StopEditing()
{
    // Check edit conditions before allowing edit to stop
    if (m_pCurrentLayer == null) return;
    IFeatureLayer pFeatureLayer = (IFeatureLayer)m_pCurrentLayer;
    if (pFeatureLayer.FeatureClass == null) return;
    IDataset pDataset = (IDataset)pFeatureLayer.FeatureClass;
    if ( pDataset == null) return;

    // If the current document has been edited then prompt the user to save changes
    IWorkspaceEdit pWorkspaceEdit = (IWorkspaceEdit)pDataset.Workspace;
    if (pWorkspaceEdit.IsBeingEdited())
    {
        bool bHasEdits = false;
        pWorkspaceEdit.HasEdits(ref bHasEdits);
        bool bSave = false;
        if (bHasEdits)
        {
            DialogResult result;
            result = MessageBox.Show(this, "Do you want to save your edits?", "Question
Dialog", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);
            if (DialogResult.Yes == result)
                bSave = true;
```

```
        }
        pWorkspaceEdit.StopEditing(bSave);
    }

    m_pMap.ClearSelection();
    IActiveView pActiveView = (IActiveView)m_pMap;
    pActiveView.Refresh();
}
```

11 ArcCatalog 对象

在基于 ArcObjects 可视化控件的 GIS 二次开发中，使用到的 Catalog 对象并不多，由于 ArcCatalog 是一个用于产生和管理地理数据的应用程序，它本身已经设计的很完备了。通过 ArcCatalog，用户可以新建不同类型的地理数据，如要素类、要素数据集、网络、栅格数据集等、建立拓扑、改变一个要素类的空间参考、导入数据等。

但就这些数据管理功能的其本质而言，它们使用 API 函数都不是 ArcCatalog 对象提供的，ArcCatalog 仅仅是提供了一个框架而已，而用户在这个框架内调用了 GeoDatabase 等其他组件库中的对象。

ArcCatalog 通常被看做是一个以数据为中心的应用程序，它看起来和 Windows 的 Explorer 非常相似，每一种不同类型的文件都会使用一种特别的图标来标识，如图 11-1 所示。

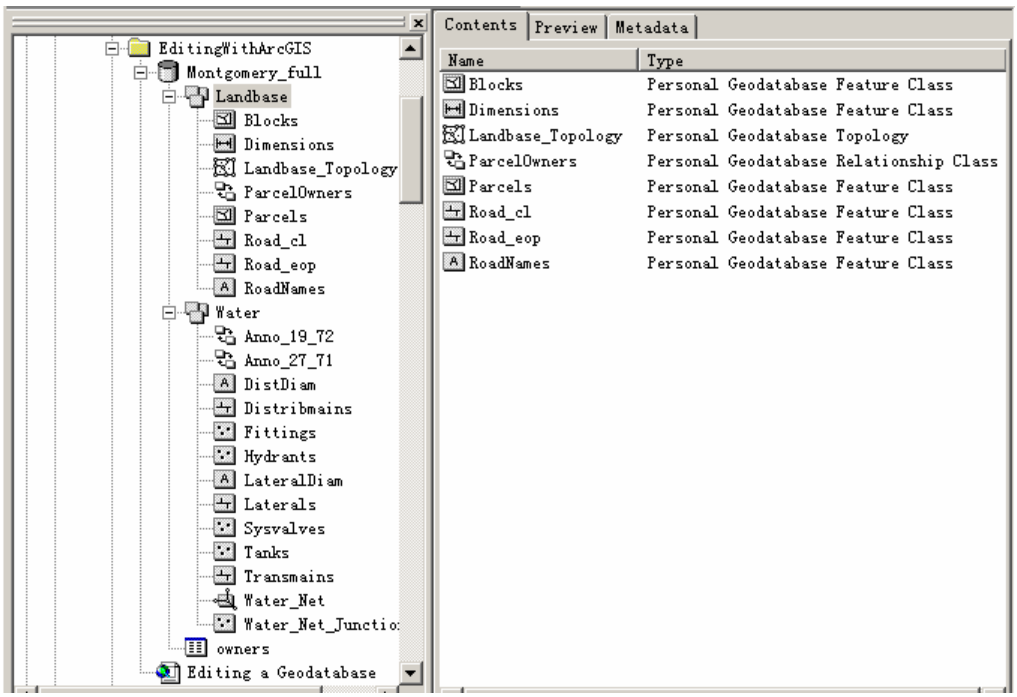


图 11-1 ArcCatalog 中的各种图标及视图

ArcCatalog 可以使用三种类型的视图来查看地理数据，即内容视图、预览视图和元数据视图。通过这三种视图，用户以不同数据表现方式来获得地理数据的信息。

除了上面的三种视图外，用户还可以使用编程的方法来创建自己的视图模型，这个扩展也是使用 VBA 环境来实现的，而编写的代码同样是保存在一个文档中，这种文档的后缀名是.gxt。与 ArcMap 类似，ArcCatalog 中也有一个 Application 对象，它代表了 ArcCatalog 对象本身，被称为 GxApplication，它是定制 ArcCatalog 的起点，在 VBA 环境中定制 ArcCatalog 时，它同样是一个全局级的对象，不需要被额外引入。

视图对象与 GxApplication 相关联，这些视图对象被成为 GxView，它们和 ArcCatalog 的视图模式相对应，这些视图模式有 GxContentsView 内容视图)、GxTableView 表视图，可以以表的形式查看地理数据)、GxGeographicView 地理视图，可以直接查看地理信息）和 GxTreeView（树状视图）等。如图 10-1，这些视图都代表了不同类型的数据表现形式。

GxObject 对象代表了在 ArcCatalog 中出现的对象，它的内容非常广泛，包括磁盘链接、数据库、数据集、图层、文件夹、投影文件等。用户从 ArcCatalog 程序中获得的任何对象，都可以看做是一个 GxObject，它也是需要被载入到控件进行处理的对象。

GxCatalog 对象也和 GxApplication 相关联，它是 GxObject 的一种，拥有两个重要的属性，一个是 Selection，用于获得一个 Gxselection 对象，它指向当前选择的对象；另一个是 GxFileFilter，它可以用于过滤文件，以显示和隐藏特定的文件类型。

GxFile 是一种 GxObject 对象，它实现了 IGxObject 接口，这个接口定义了操作文件的方法，如打开，关闭，获得文件路径等。

在使用 ArcObiects 控件开发的 GIS 程序中使用到的 ArcCatalog 对象并个是很多，一般都使用 GxDialog 对象来作为一个载入地理数据的对话框，而使用这个对象，就必须了解 GxbjectFilter 以及 GxDialog 可以取出的 GxObject 对象。因此本章不会全面讲解 ArcCatalog 的内容.而仅仅介绍在使用控件开发中常用的 ArcCatalog 对象。

学习完本章后，读者应该掌握以下内容：

- 什么是 GxObject 对象
- 对象过滤器
- 使用 GxDialog 来打开地理数据

11.1 GxObject 对象

GxObject 也许是 ArcCatalog 中最为重要的抽象类，出现在树状视图或列表视图中的每一个项目其实都是一种 GxObject，而 ArcCatalog 程序中这些 GxObject 的数目完全可以使用“打”来做计量单位，它的范围可以从 GxDatabase 到 GxPrjFile。

在图 10-1 中看到的每一种数据，包括文件夹，在 ArcCatalog 中都是当做一种 GxObject 对象。

GxObject 支持 IGxObject 接口，这个接口定义了所有 GxObject 对象的公共属性和方法。如 IGxObject::Name 是指某个 GxObject 的名字，这个名字包括了对象的扩展名；IGxObject::BaseName 则是不包括扩展名的对象名称；而 IGxObject::FullName 则会返回这个对象的全名，它还包括了对象到根目录的路径。

Parent 属性可以返回某个 GxObject 对象的上一级对象，毫无疑问，后者也是一个 GxObject 对象。

下面的一段 VBA 代码显示的是遍历所有在 ArcCatalog 中被选择的 GxObject 对象：

```
Sub demo()
    Dim pApp As IGxApplication
    Dim pGxSel As IGxSelection
    Dim pEnumGxobj As IEnumGxObject
    Dim pGxobj As IGxObject
    '得到程序中被选择的 GxObject 对象，作为一个枚举对象被返回
    Set pApp = Application
    Set pGxSel = pApp.Selection
    Set pEnumGxobj = pGxSel.SelectedObjects
    '遍历这些 GxObject 对象
    Set pGxobj = pEnumGxobj.Next
    Do While Not pGxobj Is Nothing
        MsgBox pGxobj.Category
        Set pGxobj = pEnumGxobj.Next
    Loop
End Sub
```

GxObject 的子类很多，主要分为 GxObjectContainer、GxFile、GxMetadata 和 GxObjectWIzard 四种，下面逐一介绍这些子类。

- **GxObjectContainer 对象**

某些 GxObject 会实现 IGxObjectContainer 接口，这些对象的特点是它们内部可以包含其他的 GxObject 对象作为其子对象，例如 GxDatabase 能够容纳 GxDataset 等对象，因而它是一种 GxObjectContainer 对象。

下面是部分的 IGxObjectContainer 对象：

类	描述
GxCadDataset	一个 CAD 要素类或数据集
GxCatalog	代表 Catalog 对象
GxCoverageDataset	一个 Coverage 要素类或信息表数据集
GxDatabase	代表一个数据库
GxDataset	代表一个数据集对象
GxDiskConnection	代表一个磁盘链接
GxFolder	代表一个文件夹
GxRasterDataset	代表一个栅格数据集
GxRemoteDatabaseFolder	代表 SDE 数据库链接的文件夹
GxSpatialReferencesFolder	包含空间参考文件的文件夹
IMSConnection	代表一个 ArcIMS 链接
IMSFeatureClass	代表一个 ArcIMS 要素类
SearchResult	代表一个搜索结果
IMSMetadataService	代表一个 ArcIMS 元数据服务

IGxObjectContainer::HasChildren 属性可以用于判断一个 GxObject 对象中是否拥有子类，而 Children 属性则可以返回一个枚举对象，这些对象是包含在某个 GxObject 中的子对象。

下面的 VBA 代码可以遍历一个 GxObjectContainer 对象中的子对象：

```
Sub GxObjectContainer()
        Dim pApp As IGxApplication
        Dim pGxSel As IGxSelection
        Dim pGxObj As IGxObject
```

```
        Set pApp = Application
        Set pGxSel = pApp.Selection
        Set pGxObj = pGxSel.Location
        If TypeOf pGxObj Is IGxObjectContainer Then
            Dim pGxObjCont As IGxObjectContainer
            Dim pEnum As IEnumGxObject
            Dim pObject As IGxObject
            '遍历其所有的子对象，作为一个枚举对象返回
            Set pGxObjCont = pGxObj
            Set pEnum = pGxObjCont.Children
            Set pObject = pEnum.Next
            Do While Not pObject Is Nothing
                MsgBox (pObject.Category)
                Set pObject = pEnum.Next
            Loop
        End If
    End Sub
```

- **GxFile 对象**

GxFile 对象代表了所有可以显示在 ArcCatalog 中的文件类型，这些类型非常的丰富，可以是磁盘中的任何一种文件。用户可以使用下面的 Option 对话框中的 FileTypes 来加入一个新的文件类型，如图 11-2 所示。

点击右边的按钮，会弹出一个“Registered Files TyPes”对话框，用户可以在里面查看到所有注册到操作系统的文件，选择一种文件后，点击“OK”，就可以在 ArcCatalog 中看到这种类型的文件了。

下面的代码段是新建一个 GxFile 对象并将其处于编辑模式：

```
Dim pFile As IGxFile
pFile = new GxFile
pFile.Path = "d:\tools\Various\labeling.doc"
pFile.Edit()
```

所有的 GxFile 对象都实现了 IGxFile 接口，这个接口可以往磁盘上写入信息。IGxFile::Open 可以打开一个文件，而 IGxFile::Edit 可以让一个文件处于编辑模式。

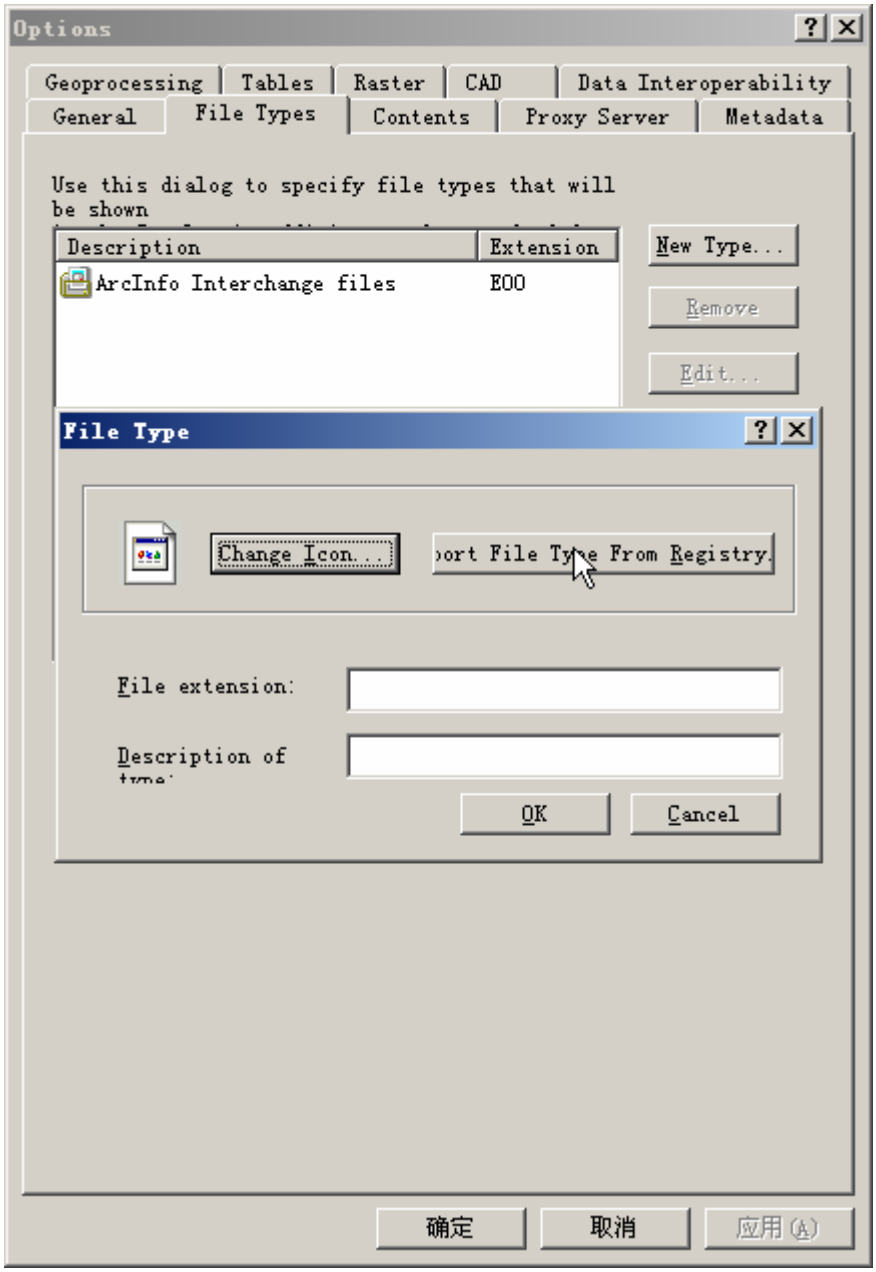


图 11-2 ArcCatalog 中添加文件类型

- **GxMetadata 对象**

不同类型的 GxObject 对象的元数据都保存在一个 XML 文件中，例如一个 Parcels 的 shapefile 文件的元数据被存储在一个名为 Parcel.shp.xml 的文件中。

元数据通过 XML 来描述数据本身的信息，它是现在非常流行的一种数据传递方

式。

11.2 GxDialog 对象

GxDialog 对象是一个可视化的对话框，它提供了一种简单直观的方法让用户输入输出数据。在 GxDialog 对话框中可以打开或者保存地理数据。

在使用 GxDialog 之前，用户需要设置它的一些属性，这些属性由 IGxDialog 接口来定义，如 IGxDialog::AllowMultiSelect 用于设定对话框是否允许一次选择多个对象；IGxDialog::StartingLocation 属性则用于设定该对象打开时的起始路径，如设置为“我的文档”等。Title 和 ButtonCaption 属性分别用于设置对话框的标题和按钮的标题。

GxDialog 对象还有一个比较重要的属性是 ObjectFilter，它使用 GxObjectFilter 对象作为文件显示的过滤器。

使用 IGxDialog::DoModalopen 方法可以打开对话框，为了确定选择的对象，读者可以传入一个 IEnumGxObject 作为一个打开对话框的参数，它可以含有用户选择的所有文件；在 IGxDialog::DoModalSave 方法中，就没有这个参数了。

下面是一个使用 GxDialog 的代码例子：

```
private void AddDataToMapByGxDialog()
{
    IGxDialog pGxDialog ;

    pGxDialog = new GxDialogClass();
    pGxDialog.AllowMultiSelect = true;
    pGxDialog.Title = "Select Feature Classes to Add to Map";

    IGxObjectFilter pGxFilter;
    pGxFilter = new GxFilterDatasetsAndLayersClass();
    pGxDialog.ObjectFilter = pGxFilter;

    IEnumGxObject pGxObjects;
    pGxDialog.DoModalOpen(0, out pGxObjects);     //打开对话框
    if (pGxObjects == null) return;
    pGxObjects.Reset();

    IFeatureLayer pLayer;
```

```
            IGxDataset pGxDataset;
            pGxDataset = pGxObjects.Next() as IGxDataset;
            while (pGxDataset != null)
            {
                pLayer = new FeatureLayerClass();
                pLayer.FeatureClass = pGxDataset.Dataset as IFeatureClass;
                pLayer.Name = pLayer.FeatureClass.AliasName;
                axMapControl1.AddLayer(pLayer,0);
                pGxDataset = pGxObjects.Next()   as IGxDataset;
            }
            axMapControl1.ActiveView .PartialRefresh
(esriViewDrawPhase.esriViewGeography, null, null);

        }
```

图 11-3 是这段代码执行后的对话框。

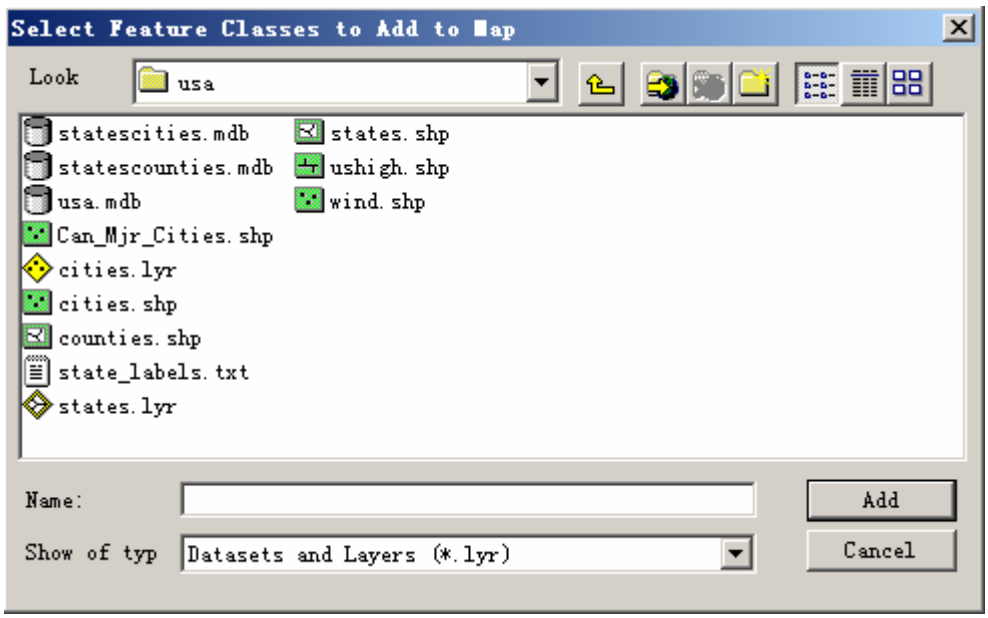


图 11-3 GxDialog 对话框

在上面的例子中，GxDialog 对象返回了一个 GxObjects 对象，里面可能包含了一个或者多个的 GxDataset 对象，它取决于选择的对象数目。当用户二次开发程序的时候，可以使用这个对象载入数据而不是使用 C#.NET 提供的 open 对话框。这个例子使用 IGxObjectFilter 对象作为过滤器，用于确定能够显示在 GxDialog 对话框中的数据类型，它有众多的对象，GxFilterAnnotationFeatureClasse 用于显示

标注要素类；GxFilterCadDrawingDatasets 用于显示 CAD 数据的图形数据集对象；GxFilterDatasets 用于显示数据集对象；GxFilterFeatureClasses 用于显示要素类对象。

注意，使用 GxDialog 对象需要引入 CatalogUI 组件库。

本章探讨了如何使用 ArcCatalog 中的对象等问题。由于 ArcCatalog 程序的功能非常完善，使得一般的 GIS 程序都不必自己开发专门的数据管理模块，而使用最多的就是 GxDialog 对象。学习完这些内容后，读者就可以使用 ArcObjects 提供的专门的数据对话框来加载地理数据。

12 Raster 数据简介

除了离散的矢量数据外，GIS 中还有一种用于描述连续现像的栅格数据，即 Raster 数据。Raster 数据是由多个等间距矩形栅格组成的，它们可以用于模拟在一定的空间范围内连续变化的地理现象或者图片数据，这种格式的数据非常常见，如卫星图片、扫描地图、航空影像、图片等。

Raster 数据分为两种类型，一种是专题数据，它可以用于进行地理分析；另外一种是影像数据，它一般用于地图的背景显示。在 ArcObjects 中，栅格数据对象也被用于分析栅格数据和显示栅格数据，但是它提供了一种统一的方式来管理任何类型的栅格数据，这意味着用户可以像处理要素数据一样，使用同样的函数来操作如 GRID、TIFF、ERDAS、PNG 或其他的栅格数据。

有一些栅格数据类型，如 GRID 和 TIFF，可以支持投影、数据统计等操作；而如 GIF 和 JPEG 等数据，就仅仅可以作为一个地图的背景来使用。

栅格数据模型用连续的空间像元来表达要素，像元表现为一个矩形的区域，一般情况下，其宽度和高度固定且相等，它的大小取决于原始地图的比例尺和绘制地图时最小的单位。像元过大将导致一些信息的丢失，像元过小则在没有增加地图精度的情况下，需要更大的存储空间和更长的操作时间。栅格是一个二维的动态像元矩阵，每个栅格单元都有一个值，用于表示某个位置的某种属性，如高程、反射率、颜色等。栅格中含有一个或多个波段，在多波段的栅格中，一个波段表示为传感器收集的一段电磁波。

栅格数据集是存储在文件或数据库中的，以某种栅格格式存在的数据对象，它可以管理数据的栅格和栅格波段对象，而将栅格数据载入到 MapControl 控件中，需要使用栅格图层来实现。本章将简要介绍栅格数据的使用。

在学习完本章后，读者将可以获得以下知识：

- 栅格工作空间对象打开栅格数据集
- 栅格数据集的相关知识
- 使用栅格图层将栅格数据集加入到 Map 中去

12.1 RasterWorkSpace 对象

RasterWorkspaceFactory 通常用于产生一个 RasterWorkspace 对象，RasterWorkspace 对象就是一个栅格数据集的容器，与 FeatureWorkspace 对象类似，RasterWorkspacefactory 类也实现了 IWorkspaceFactory 和 IWorkspaceFactory2 接口，它们可以用于产生所有类型的工作空间。

下面的例子是使用 RasterWorkspaceFactory 对象的函数用于返回一个 RasterWorkspace 对象:

```
private IRasterWorkspace SetRasterWorkspace(string sName)
{
    IWorkspaceFactory pWKSF ;
    pWKSF = new RasterWorkspaceFactoryClass();
    IRasterWorkspace pWKS ;
    pWKS = pWKSF.OpenFromFile(sName, 0) as IRasterWorkspace;
    return pWKS;
}
```

IRasterWorkspace 接口定义的方法可以用于打开和设置一个数据集对象，如果指定位置是一个有效的 RasterWorkspace 对象，则 IRasterWorkspace::IsWorkspace 属性返回为 true；如果 CanCopy 属性为 true，则 IRasterWorkspace::Copy 方法可以拷贝一个工作空间中所有的栅格数据集到一个新的或已经存在的工作空间中去。

IRasterWorkspace::OpenRasterDataset 方法可以打开一个栅格数据集，它是获得一个栅格要素集对象的主要方法。下面的例子提供了一个函数用于得到一个栅格数据集对象的方法:

```
private IRasterDataset OpenRasterDataset(string sPath,string sFileName)
{
    //sPath是栅格数据库的路径，sFileName是栅格数据集名
    //产生一个RasterWorkspaceFactory对象
    IWorkspaceFactory pWSF;
    pWSF = new RasterWorkspaceFactoryClass();
    //获取一个RasterWorkspace对象
    IRasterWorkspace pRasWS;
    if (pWSF.IsWorkspace(sPath))
    {
        pRasWS = pWSF.OpenFromFile(sPath, 0) as IRasterWorkspace;
```

```
            return pRasWS.OpenRasterDataset(sFileName);
        }
        return null;

    }
```

IRasterWorkspace2 接口除了定义了 IRasterWorkspace 接口定义的方法和属性外，还提供了一个函数 CreateRasterDataset，它用于产生一个新的基于文件的栅格数据集对象，但是这个数据的格式为 GRID、TIFF 和 ERDAS Image 三种之一。其语法格式如下：

```
    public IRasterDataset CreateRasterDataset (
        string Name,
        string Format,
        IPoint Origin,
        int columnCount,
        int RowCount,
        double cellSizeX,
        double cellSizeY,
        int numBands,
        rstPixelType pt,
        ISpatialReference sr,
        bool Permanent

        );
```

这个函数需要输入多个参数，如 Name 参数是栅格数据库中的数据集名；RowCount 和 ColumnCount 确定了栅格的行和列的数目；sizeX 和 sizeY 参数用于设置栅格数据集的范围；numBands 参数确定栅格数据集的波段数；Permanent 参数则可以确定系统需要时是否可以新建一个临时的数据集，在地理分析中，系统经常需要产生一个临时数据集。

下面是产生一个栅格文件的方法，它可以生成一个 TIF 文件：

```
    private IRasterDataset CreateRasterDataset(string sPath, string sFileName)
    {
        //产生一个 RasterW0rkspace
        IRasterWorkspace2 pRWS ;
        IWorkspaceFactory pWSF;
        pWSF = new RasterWorkspaceFactoryClass();
        pRWS = pWSF.OpenFromFile(sPath, 0) as IRasterWorkspace2;
        //设置栅格数据集的Origin参数
```

```
            IPoint pOrigin;
            pOrigin = new PointClass();
            pOrigin.X = 0;
            pOrigin.Y = 0;
            //产生一个栅格数据集
            IRasterDataset pRasterDS;
            pRasterDS = pRWS.CreateRasterDataset(sFileName, "TIFF ", pOrigin, 100,
100, 30, 30, 1, rstPixelType.PT_LONG, null, true);
            return pRasterDS;
        }
```

12.2 栅格数据集

栅格数据集 RasterDataset 对象代表了存储在磁盘或数据库中的一个数据集对象，这个数据集是由一个或多个栅格波段组成。和要素数据集一样，所有类型的栅格数据都可以看做是一种栅格数据集对象。

RasterDataset 对象有一些基本属性，如文件格式、范围 Extent、空间参考和波段数目，这些属性可以使用两种方式修改。第一种是改变与数据集相关联的空间参考属性，这不会改变数据集中的投影，而是改变它的坐标系统；第二种是修建一个栅格金字塔，或者是降低数据集的分辨率，这种方式对于显示大范围的栅格数据非常有帮助。

为了获得一个栅格数据集，除了使用 IRasterWorkspace::OpenRasterDataset 方法外，还可以通过 IRasterBand::RasterDataset 方法来得到。

RasterDataset 类实现了多个接口，如 IRasterDataset，这个接口定义的方法可以设置一个栅格数据集的栅格格式、压缩类型和数据集的名称等属性。如果 CanCopy 属性为 true，则 Copy 方法可以从这个栅格数据集中产生一个新的拷贝对象。

IRasterDataset::CreateDefaultRaster 方法可以产生一个缺省状态的栅格对象，这个缺省的栅格有高度、宽度、范围和空间参考等属性，这些属性都来自栅格数据集；这个栅格中还可能包含一个或三个波段，前一种栅格是单波段栅格对象，后者是多波段栅格对象。

使用 IRasterDataset::CreateFullDataset 方法则可以在栅格数据集中产生一个全波段的栅格对象；IRasterDataset::CreateDataset 则可以产生一个空栅格。

IRasterPyramid 接口可以用于栅格数据集产生栅格金字塔，栅格金字塔是将栅格图像逐级抽稀，形成多级分辨率的重采样数据，并将其分割成块，按一定的文件格式(金字塔文件格式)存储为磁盘文件或存储在 SDE 数据库中，在以后进行图像显示处理时，只需将要显示的部分所覆盖的块从磁盘文件直接读进内存缓冲区显示即可。从金字塔的所有层中寻找一个与所要求显示的比例相近或匹配的层，并将该层从某一点起的一定范围的图像所覆盖的所有块加载到内存缓冲区，提取所需部分并形成图像。

如果一个栅格数据集中存在一个金字塔，则 IRasterPyramid::Present 属性将返回为 true；MinimumSize 属性将返回一个金字塔的最小尺寸，而 Create 方法则可以产生一个金字塔对象，如下面的代码段：

```
private void BuildPyramid(IRasterDataset pRasterDS)
{
    IRasterPyramid pRasPyramid ;
    pRasPyramid = pRasterDS as IRasterPyramid;
    //产生栅格金字塔
    if(!(pRasPyramid.Present)) pRasPyramid.Create();
}
```

IGeoDataset 接口也被 RasterDataset 类实现，它可以获得一个数据集的空间参考和范围等属性；而 IGeoDatasetSchemaEdit 接口定义的方法则可以用于设置一个栅格数据集的空间参考属性。

如果 CanAlterSpatialReference 属性为 true，则 AlterSpatialReference 方法可以对一个栅格数据集进行投影运算，下面是一个进行投影的方法：

```
private void AlterRasterSpatialReference(IRasterDataset pUTMRasterDataset,
IRasterDataset pNoPrjRasterDataset)
{
    IGeoDataset pGeoDataset;
    pGeoDataset = pUTMRasterDataset as IGeoDataset;
    IGeoDatasetSchemaEdit pGeoDatasetSchemaEdit;
    pGeoDatasetSchemaEdit = pNoPrjRasterDataset as IGeoDatasetSchemaEdit;
    if (pGeoDatasetSchemaEdit.CanAlterSpatialReference)
pGeoDatasetSchemaEdit.AlterSpatialReference(pGeoDataset.SpatialReference);
}
```

IRasterBandCollection 接口的方法可以用于管理栅格数据集中的波段对象，如 IRasterBandCollection::Count 属性可以返回一个栅格数据集中的波段数目；BandByName 和 BandIndex 属性可以分别通过一个波段名和波段的索引值得到栅格数据集中的一个波段对象。

IRasterBandCollection::Item 方法也可以从一个栅格数据集中返回一个栅格波段；SaveAs 方法可以用于将一个栅格数据集转换为一种不同的栅格数据格式，这些格式有 GRID、TIFF 和 Imagine 三种。

IRasterBandCollection::Add、Clear 和 Remove 方法并不能使用在栅格数据集上，而是用于操作栅格(Raster)对象。当一个波段对象被添加到栅格对象或从一个栅格对象中移除时，栅格会重新计算缺省的栅格中的一些属性，如范围和控件参考等，如果用户没有明确地设定这些属性，那么栅格对象将被设置为缺省属性。

ITemporaryDataset 接口支持一个临时的栅格数据集。临时栅格数据集在很多 ArcView 程序和 ArcGIS 程序的空间分析都需要使用到，这些临时栅格数据集可以通过 IRasterWorkspace::CreateRasterDataset 方法产生出来。

如果数据集是临时的，则 ITemporaryDataset::IsTemporary 属性返回为 true；ITemporaryDataset::MakePermanent 方法则可以将一个数据集从临时状态持久化，即保存在磁盘或数据库中；如果某个数据集处于持久化状态，那么这个方法就没有什么作用了。

ITemporaryDataset::MakePermanentAs 方法可以通过产生一个新数据集的方式将一个临时数据集持久化，这个新数据集可以保存在任何一种栅格工作空间中，也可以保存在 GRID、TIFF 或 Imagine 格式中。

12.3 栅格图层对象

RasterLayer 是一个栅格图层对象，它包含了栅格数据，并且可以将这些数据传入地图对象中用于显示。RasterLayer 类实现了 IRasterLayer 接口，它定义了多种栅格数据方法和属性用于操作一个栅格图层对象。

IRasterLayer::BandCount 可以返回一个栅格图层的波段数目；IRasterLayer::Cached 则用于确定一个栅格图层是否设置缓存对象以便于快速显示；IRasterLayer::DisplayResolutionFactor 属性用于获得或设置栅格图层的分辨率，这对于栅格类型的数据是非常重要的，分辨率越高，数据将越精细，这个值在 0-100 之间。

在一个栅格图层对象新建后是一个空的对象，用户需要使用具体的数据来实现它，这个过程可以使用的方法有 IRasterLayer::CreateFromDataset，它可以从一个栅格数据集来产生一个图层；IRasterLayer::CreateFromFilePath 方法，它可以通过传入一个栅格数据路径的方式产生一个图层；CreateFromRaster 方法，它是使用一个栅格对象来产生图层。

下面的例子是使用 CreateFromFilePath 方法产生一个栅格图层对象的代码：

```
private void demo(IRasterDataset pRasterDataset,IRaster pRaster)
{
    //产生一个新的栅格图层
    IRasterLayer pRasLyr;
    pRasLyr = new RasterLayerClass();
    //使用三种方式来初始化pRasLyr
    pRasLyr.CreateFromFilePath(@"D:\data\dem.Bsq");
    pRasLyr.CreateFromDataset(pRasterDataset);
    pRasLyr.CreateFromRaster(pRaster);

}
```

和矢量图层一样，栅格图层也有最大和最小显示范围，这个范围值可以通过 IRasterLayer::MinimumScale 和 IRasterLayer::MaximumScale 属性来获得。

IRasterLayer::Raster 属性则用于获得一个栅格图层的栅格对象。

要素图层中的数据可以进行着色以制成专题图，同样地，栅格图层也具有这个能力。

IRasterLayer::Renderer 就用于设置栅格图层的着色属性。

栅格数据是 GIS 中经常需要处理的另一大类地理数据，它们用于模拟自然界的具有连续分布特征的对象，由于 ArcObjects 在数据处理中使用了一个统一的数据模型，因此 ArcObjects 中的栅格数据操作方法和本书前面介绍的矢量数据具有很大的相似性。本章只是简单地介绍了栅格数据的简单原理和栅格对象、栅格数据图层的基本知识。关于栅格数据模型的详细信息，读者可以参考专门的书籍或开发帮助。

参考文献

[1] 蒋波涛. ArcObjects 开发基础与技巧--基于 VisualBasic.NET [M]武汉：武汉大学出版社，2006

[2] 韩鹏，徐占华等. 地理信息系统开发--ArcObjects 方法[M]武汉：武汉大学出版社，2005

[3] ArcGIS Engine Developer Guide. ESRI Press，2004

[4] ArcGIS Desktop Developer Guide. ESRI Press，2004

[5] GeoDatabase Workbook．ESRI Press，2004

[6] Geoprocessing in ArcGIS．ESRI Press，2004

[7] MichaelZeiler. Exploring ArcObjests ESRI Press，2001

[8] MichaelZeiler. Modeling 0ur World. ESRI Press，1999

[9] ESRI Developer Network: http://edndoc.esri.com/arcobjects/9.1/

[10] ESRI 中国(北京)有限公司在线支持中心: http://support.esrichina-bj.cn/esrilink/index.php