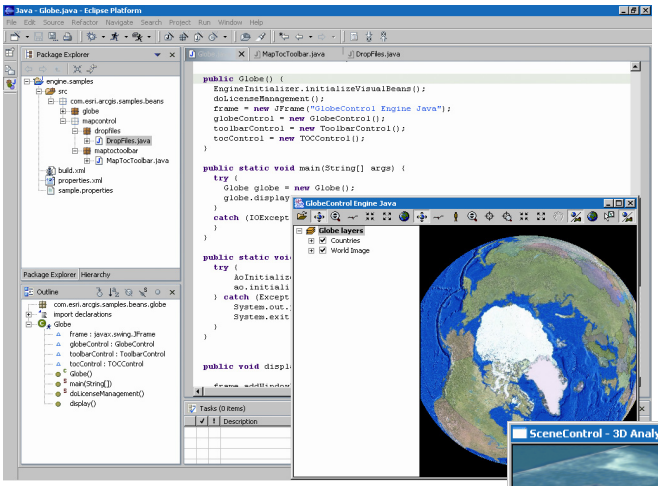
三维可视化及其他

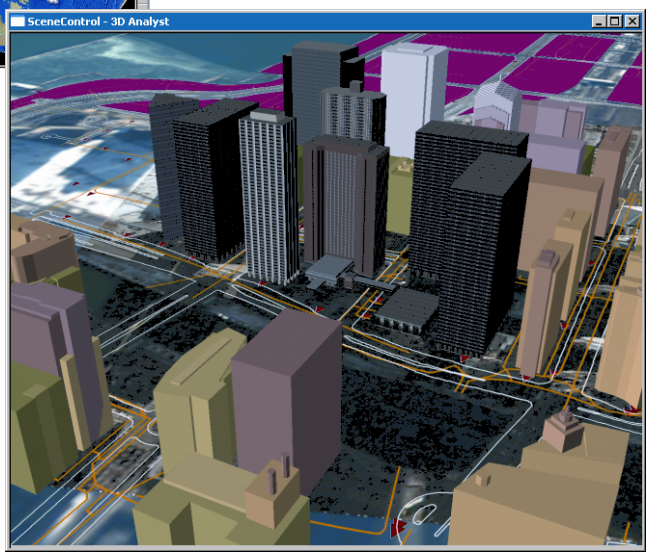
ArcGIS Engine 运行时 3D 选项进一步扩展了 ArcGIS Engine 的功能，使开发人员可以用 Scene 和 Globe 控件建立高效可视化和分析表面和全球（Globe）数据的应用程序。开发人员可以创建能从多个视点查看表面、查询表面、确定表面上被选位置的可视区域、通过在表面上叠加栅格和矢量数据显示逼真的透视影像。

例如，开发人员可以：

- 显示 Scene 和 Globe 文档
- 执行交互式透视图查看，包括漫游和缩放、旋转、倾斜和飞行模拟以便表达和分析。
- 显示现实世界表面要素，如建筑物等
- 执行可视区分析和视线(line-of-sight)分析、现场高度插值、剖面图制作及确定最陡路径。

基于 GlobeControl 应用程序的 Java 代码





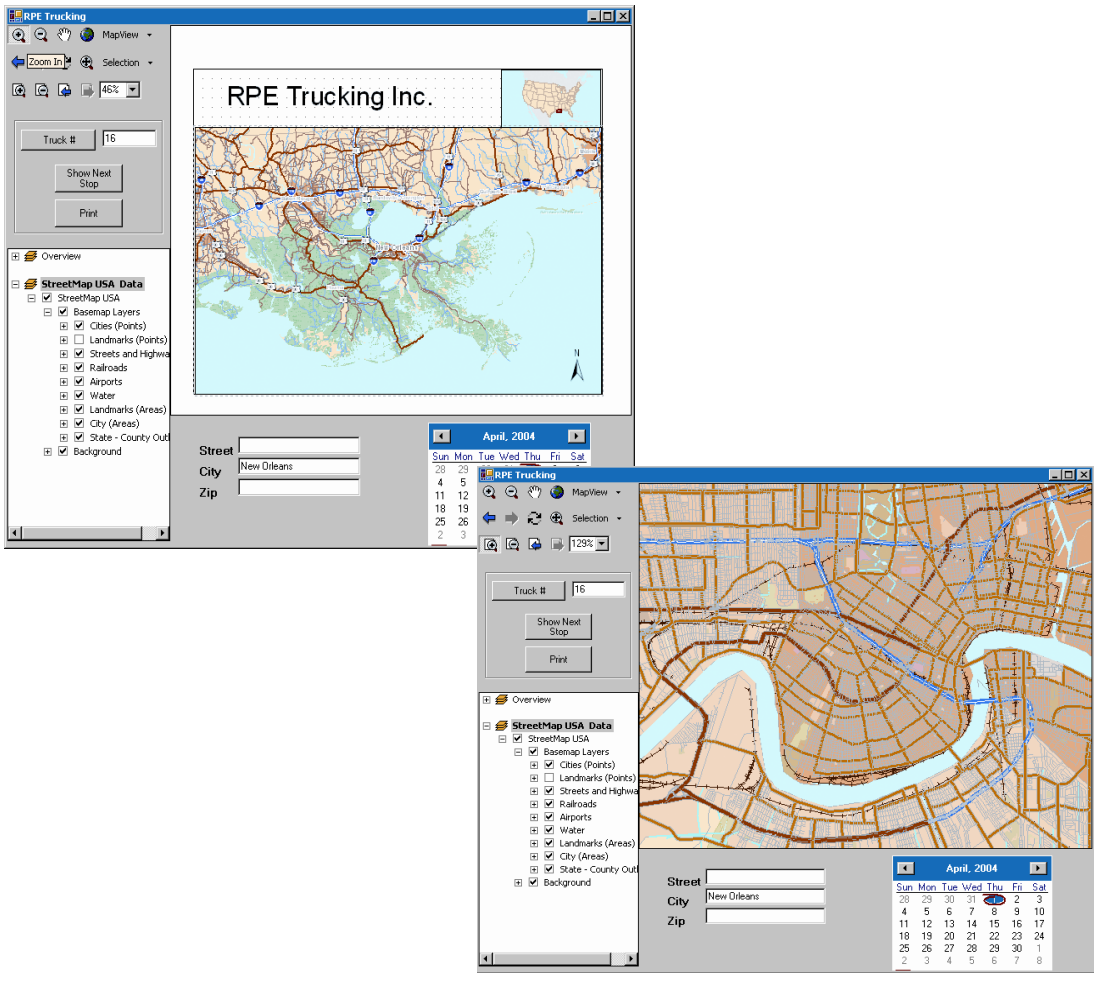
显示了一个基于 SceneControl 的应用程序

街道级数据

StreetMap USA 提供了全美详细的街道数据。使用 StreetMap USA 运行时选项，用户可以在其 ArcGIS Engine 应用程序中使用这些数据。使用这个选项，用户可以象使用任何其他要素类数据集一样使用 StreetMap 数据源。StreetMap USA 选项扩展了 ArcObjects 内的底层地理数据库对象模型，这样用户应用程序可以将 StreetMap 数据源作为地理数据库对象无缝地使用。

StreetMap USA 运行时选项提供下列功能：

- 全国性的地址匹配
- 街道和里程碑数据库
- StreetMap 图层组，可以在不同比例尺上显示不同的细节层次
- 基本的街道路线选择



一旦安装了 ArcGIS Engine 开发工具包，就可以开始开发 ArcGIS Engine 应用程序。然而，好的应用程序需要仔细地规划；使用 ArcObjects 进行开发也不例外。在开始进行开发前，通读并使用（必要的话）本节中的讨论和列出的可选清单。这些信息可以帮助用户做出规划并确保用户处于正确的起点上。

确定应用程序类型

第六章“开发情景”提供了一些 ArcGIS Engine 应用程序的示例。ArcGIS 开发帮助系统中也包括一些其他示例。

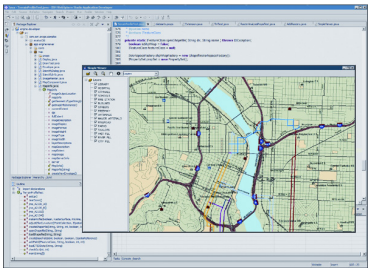
使用 ArcGIS Engine 可以开发多种应用程序。这些应用程序可以从执行诸如数据库编辑和分析等简单的控制台到包含用于用户交互和地理数据显示的控件和可视化组件的复杂 Windows 应用程序。一般地，有三类 ArcGIS Engine 应用程序：

1. 独立的非可视化应用程序，如控制台和实用工具应用程序。
2. 独立的可视化应用程序，如 Windows 和基于控件的应用程序。
3. 嵌入式应用程序，如嵌入到现有应用程序中的组件。

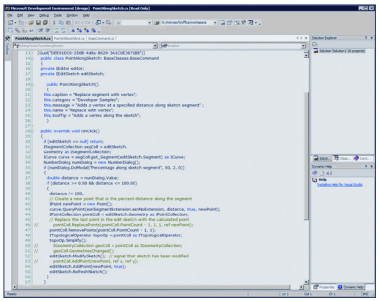
开发的应用程序的类型最终将取决于当前项目的功能需求。

清单：

- ☐ 要开发的是何种类型的应用程序？是非可视化、可视化、还是嵌入式应用程序？
- ☐ 用户是否打算将开发的功能移植到 ArcGIS Desktop 或 ArcGIS Server 产品中？
- ☐ 现在和未来用户希望支持何种平台？Windows？UNIX？还是两者都支持？



websphere studio



visual studio.net

选择 API 和开发环境

ArcGIS Engine 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（VB）6、C#.NET、Java 和 Visual C++等。下面列出了适合 ArcGIS Engine 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—Eclipse，WebsphereStudio，Intelli J，Jbuilder 等
- C++—Visual Studio 6.0，C++ Builder

本书的大部分地方使用 VB6 作为展示多数代码概念的语言，而且也往往是开始时最容易掌握的语言。第四章“开发环境”阐述了 VB 及 ArcGIS Engine APIs 支持的其他一些开发环境的编程方针。

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。

清单：

- □ 开发人员最熟悉何种开发环境与语言？
- □ 打算使用何种 ArcGIS Engine API？
- □ 何种开发环境与语言最适合于待进行的开发类型？

开发应用程序

此时，假设已经制定好了一个合适的项目开发计划，开发人员准备着手研究 ArcGIS Engine 软件开发工具包并开始开发应用程序。开发人员可能希望开始查找应用程序功能必需的类库和对象。这个过程可以使用开发帮助资源，包括 ArcGIS 开发帮助系统、开发指南系列、帮助系统中的示例以及 ArcGIS 开发在线网站资源等。

为成功编译和运行应用程序，使用的所有 ArcObjects 功能组或库都要在开发环境中引用。ArcGIS Engine 中可用的各种类库将在第二章“ArcGIS 软件体系结构”中讨论。

清单：

- □ 查找需要的 ArcObjects 功能。
- □ 需要引用何种 ArcGIS Engine 类库？
- □ 运行应用程序需要何种 ArcGIS 许可？
- □ 是否需要 ArcGIS Engine 运行时选项？
- □ 计划如何部署应用程序？
- □ 是否实现了正确的许可检出代码？

“ArcGIS开发在线”可以从这个网址访问：http://arcgisdeveloperonline.esri.com。

部署应用程序

应该早在开始应用程序开发之前考虑应用程序的部署问题。ArcGIS Engine 应用程序的部署方式有多种，也可能有多种终端用户软件和许可配置方式。因此，开发人员需要考虑许多问题。

清单：

- □ 是否会安装 ArcGIS Desktop 或 ArcGIS Engine 运行时或者两者都安装？应用程序将采用何种许可？
- □ 终端用户在其系统中有何种 ArcGIS 许可？ArcInfo，ArcEditor 还是 ArcView ？
- □ 开发人员如何打包和部署应用程序？
- □ 将来是否需要提供新的版本？
- □ 如何分发开发的应用程序？

第五章“许可与部署”将讨论这些清单的各个方面。

使用本书

本书《ArcGIS Engine 开发指南》是一本介绍 GIS 独立应用程序开发的入门教程。本书将介绍 ArcGIS Engine 开发工具包的所有组件、讨论建立应用程序的相关方面、介绍支持的 APIs 并提供建立实际 GIS 应用程序的开发情景，以帮助开发人员熟悉 ArcGIS Engine 组件模型。

为服务于大多数开发人员，本书中提供的代码示例大多数采用 COM Visual Basic 6 API。然而，开发情景涵盖了所有支持的 APIs 并有一章专门用于阐述 API 的使用问题。

本书的前两章概览了 ArcGIS Engine 及其功能，包括其体系结构和组件。其他几章集中阐述 ArcGIS Engine 支持的各种 API 在开发应用程序中的使用。

章节安排

第一章，“ArcGIS Engine 简介”，向开发人员概述了 ArcGIS Engine 产品、功能及相关资源。

第二章，“ArcGIS 软件体系结构”，描述了 ArcGIS Engine 的体系结构及软件组件在系统内的交互方式。

第三章，“使用 ArcGIS 控件”，描述了各种控件并提出了在应用程序开发中使用它们时应考虑的问题。

第四章，“开发环境”介绍了 ArcGIS Engine 支持的多种 APIs。本章逐一介绍了各种 API 从基本到高级的使用问题。

第五章，“许可与部署”详细介绍了各种许可选项，讨论了应用程序的部署策略，包括初始化和许可校验。

第六章，“开发情景”指导开发人员利用每种支持的 API 进行几种类型的独立应用程序的创建和部署。

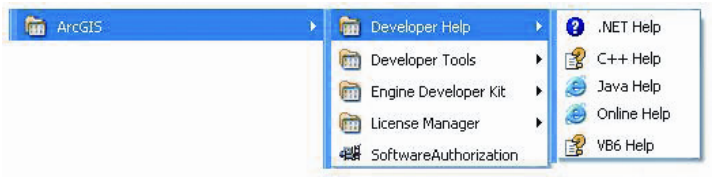
本书还包含几个附录，提供了有关 ArcGIS 开发帮助系统及其他开发资源中对象模型图的详细信息。

下面描述了 ArcGIS 开发人员可用的一些资源。有关开发人员可用资源的更深入讨论列在附录 B 中。

ArcGIS 开发帮助系统

ArcGIS 开发帮助系统是 ArcObjects 开发新手和熟练者的一个基本资源。该系统包含使用 ArcObjects 进行开发的一些信息，包括示例代码、技术文档和对象模型图。此外该系统还是 ArcObjects 中各个对象相关信息的参考指南。Visual Basic、.NET、Java 和 C++的开发人员都可以获得该帮助系统。可以从 Windows“开始”按钮下的 ArcGIS 程序组中启动 ArcGIS 开发帮助系统。

典型安装中安装的是 VB6 版本的 ArcGIS 开发帮助。按照自定义安装过程可以安装其 C++、Java 或.NET 版本。

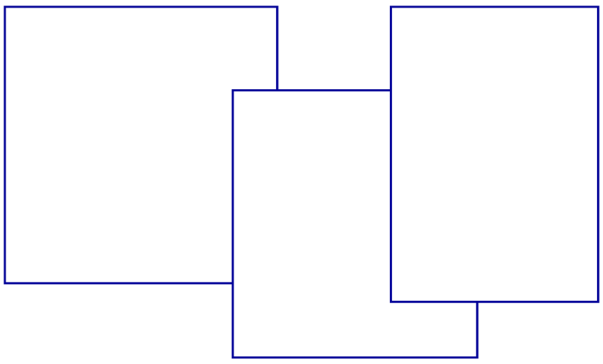


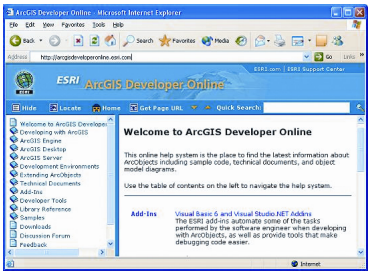
ArcGIS 开发系列

本书是 ArcGIS 开发指南丛书中的一本。

《ArcGIS Desktop 开发指南》用于指导开发人员定制或扩展 ArcGIS Desktop 应用程序，如 ArcMap 或 ArcCatalog。开发人员可以使用 Visual Basic for Application（VBA）定制或用 Visual Basic、Visual C++ 或.NET 扩展应用程序。

《ArcGIS Server 管理员与开发指南》用于指导开发人员使用 ArcGIS Server 建立自定义服务器应用程序。服务器开发人员可以建立用于简单制图或包含高级 GIS 功能的 Web 服务和 Web 应用程序。具有代码示例的几个情景展示了应用多个 ArcGIS Server 开发工具包之一可以开发的应用程序的不同类型。该书也可以作为 ArcGIS Server 管理员指南。

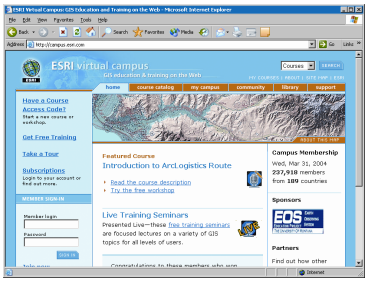




http://arcgisdeveloperonline.esri.com上的ArcGIS开发在线网站



http://support.esri.com网站上的ESRI技术支持中心



http://campus.esri.com上的ESRI虚拟校园网站

ArcGIS 开发在线网站

ArcGIS 开发在线是网络版的 ArcGIS 开发帮助系统，可以通过下面的 URL 访问：

http://arcgisdeveloperonline.esri.com。

在线网站的优点之一就是可以通过 Web 浏览器和一个到 Internet 的连接来访问。该网站在不断更新，使开发者可以获得最新的帮助信息。

ESRI 技术支持中心

ESRI技术支持中心网站http://support.esri.com上包含所有ArcGIS产品的软件信息、技术文档、示例、论坛和知识库。

ArcGIS 开发人员可以充分利用论坛、知识库和示例部分来辅助其 ArcGIS 应用程序的开发。

培训

ESRI 为 ArcGIS 开发人员提供了许多课堂讲授和基于网络的培训课程。这些课程包括从简单的 VBA 入门到更高级的 ArcGIS Desktop、 Engine 和 Server 组件开发。

要了解更多信息，访问http://www.esr.com并选择“培训和事件（Training and Events)”选项卡。

ESRI虚拟校园可以直接通过http://campus.esri.com找到。

2 ArcGIS 软件体系结构

ArcGIS 软件已经在前几个版本的基础上演化为模块化、可伸缩和跨平台的体系结构，该体系结构通过一个称为 ArcObjects 的软件组件集来实现。

本章的核心内容是介绍 ArcGIS 9 的主要进展，并向读者介绍组成 ArcGIS 系统的各种类库。

ArcGIS 软件体系结构

ArcGIS 软件已经在前几个版本的基础上演化为模块化、可伸缩和跨平台的体系结构，它通过一个称为 ArcObjects 的软件组件集来实现。

本节的核心内容是介绍 ArcGIS9 的主要进展，并向读者介绍组成 ArcGIS 系统的各种类库。

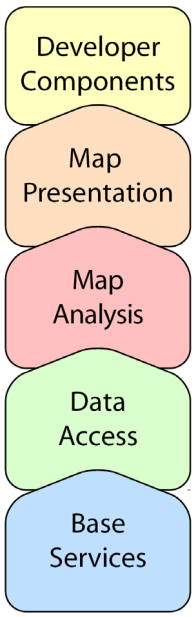
ArcGIS 软件体系结构支持许多产品，每个产品都有自己独特的需求。作为构成 ArcGIS 的组件，ArcObjects 就是为支持这些需求而设计和建立的。本章将介绍 ArcObjects。

ArcObjects 是用 C++ 语言编写的独立于平台的一套软件组件，提供支持桌面系统胖客户端和瘦客户端 GIS 应用及服务器 GIS 应用服务。

要了解有关 COM 的详细解释，请参阅第四章“开发环境”的 COM 部分。

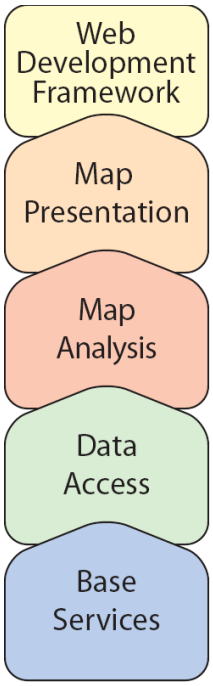
如上所述，ArcObjects 开发采用了 C++语言；此外，ArcObjects 还采用了微软的组件对象模型（COM）。人们通常认为 COM 只是简单地规定了对象在内存中如何实现和建立以及组件之间如何交流。虽然这一点是正确的，但 COM 还在操作系统层次上提供了一个坚实的基础设施以支持使用 COM 构造任何系统。在微软 Windows 操作系统中，COM 基础设施直接构建到操作系统中。对于其他非微软 Windows 的操作系统而言，必须为 ArcObjects 提供 COM 基础设施。

并非所有 ArcObjects 组件都以同样的方式创建。除了基本功能之外，对特定对象的需求随该对象的最终用途而变化。这种最终用途广义上属于三个 ArcGIS 产品家族中的一个：

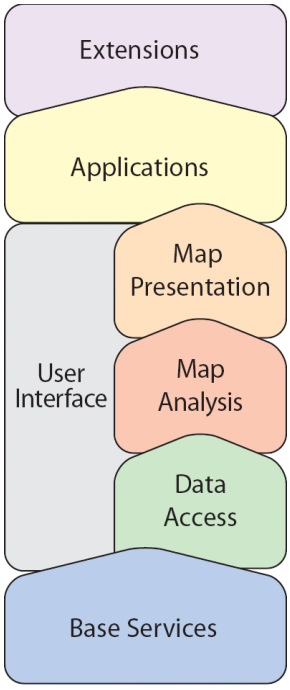


ArcGIS Engine

- ArcGIS Engine—对象在自定义应用程序中使用。Engine 中的对象必须支持各种应用；简单的地图对话框、多线程服务器和复杂的 Windows 桌面应用程序都有可能使用 Engine 对象。必须很好地理解 Engine 内对象的依赖性。必须仔细考虑添加外部依赖性到 ArcObjects 中的影响，因为新的依赖性会把不合需要的复杂性引入到构建于 Engine 之上的应用程序安装中。
- ArcGIS Server－对象在服务器框架中使用，通常对象的客户大多数为远程客户。客户的远近程度可以从本地（可能在同一计算机或网络上）到很远的地方（客户可能在互联网上）。服务器内运行的对象必须是可伸缩的，且线程必须是安全的，以允许在多线程环境下执行。
- ArcGIS Desktop－对象在 ArcGIS 桌面应用程序中使用。许多用户都具有丰富的 ArcGIS 桌面应用程序使用经验，其中包含大量的对话框和属性页，使用户可以有效的使用对象的功能。包含应用程序的用户可以修改其属性的对象应该具创建这些属性的属性页。并非所有的对象都需要属性页。



ArcGIS Server



ArcGIS Desktop

许多构成 ArcGIS 的 ArcObjects 同时在这三个 ArcGIS 产品中都有使用。这两页上的产品图显示了这三个产品中都包含的基础服务、数据访问、地图分析和地图表达组件类目中的对象。这四个类目包括了 ArcGIS 提供给开发者和使用者的主要 GIS 功能。

对于开发者来说理解 ArcGIS 所有产品之间的公共功能是很重要的，因为这意味着当使用特定的类目进行工作时，大量开发工作可以在 ArcGIS 产品之间移植而只需作少量更改。归根结底，这正是 ArcGIS 体系结构开发的方式。代码重用是构建模块化体系结构的主要好处，但代码重用并非仅指以模块化方式创建组件。

ArcGIS 体系为开发者提供了丰富功能，但它并非一个封闭系统。ESRI 之外的开发者可以对 ArcGIS 体系进行扩展。开发者已经对 ArcGIS 体系进行了多年的扩展；而 ArcGIS9 也不例外，同样可以进行扩展。但是，ArcGIS 9 引入了许多使用 ESRI 和用户创建的对象的新方法。要实现这些新方法，这些组件必须满足其他一些要求，以确保这些组件能够在这个新的、显著改进的 ArcGIS 系统中运作。在外观上，ArcGIS 8 到 ArcGIS 9 的一些变化似乎很肤浅，其中一个例子就是原来的类库被划分为更小的类库。但 ArcGIS 8.3 对象的方法和属性在 ArcGIS 9 中仍然可用的事实表明，ArcObjects 的内部结构已经进行了大的更改。

ArcGIS 9 体系的变化主要集中在以下四个关键概念上：

- 模块化－在模块化系统中，组件间的依赖性可以在一个灵活的系统中很好地定义。
- 伸缩性—ArcObjects 必须在所有预期操作环境中运行良好，包括从单用户桌面应用程序到多用户/多线程服务器应用程序。
- 多平台支持－ArcGIS Engine 和 Server 的 ArcObjects 能在多种计算机平台上运行。
- 兼容性－ArcObjects 9.0 的功能和编程方法应与 ArcGIS 8.3 保持相同。

模块化

作为 ArcGIS 8.3 的一部分，esriCore 对象库有效地把所有 ArcObjects 打包为一个大的 GIS 功能块，在其中间组件间没有区别。ArcObjects 组件分成许多小的组件群，这些组件群被打包在动态连接库（DLL）中。这个大的类库尽管简化了外部开发人员的开发任务，但阻碍了软件的模块化。将类型信息添加到所有的 DLL 中虽然是可能的，但会大大增加外部开发者的负担，因此不是一个好的选择。此外，DLL 的结构并不总是反映基于功能和依赖性的软件组件的最佳模块化方案。

通过特性和功能分析与用户需求和基于三个 ArcGIS 产品家族的部署选项匹配，ESRI 已经开发了 ArcGIS 9 的模块化体系结构。鼓励已经用自定义组件对 ArcGIS 8 体系结构进行了扩展的开发人员进行同样的过程，重新把他们的源代码构建到类似的模块化结构中。

在考虑体系结构的模块化时，总要在性能和易管理性之间维持一种平衡。对于每个标准，要考虑终端用户及支持终端用户所需的模块化。例如，系统可以分解为许多仅仅包含很少对象的 DLLs。虽然这样可以为部署选项提供很大的灵活性，需要最小的内存，但由于大量 DLLs 的载入和载出会影响系统的性能。相反地，包含所有对象的大 DLL 也不是最合适的解决方案。了解对组件的需求可以将它们有效的打包到 DLLs 中。

要做的一个明显功能区分就是用户界面和无用户界面代码。UI 类库趋向于只包括在 ArcGIS Desktop 产品中。

ArcGIS 9 体系结构被分解为许多的类库。一个类库中可以包含任意数量的 DLLs 和 EXEs。组件包含到某个类库中必须满足的要求要有明确的定义。例如，诸如 esriGeometry 这样的类库（来自基本服务模块集）要求线程安全、可伸缩、没有用户界面组件和能够在多种平台上部署等条件。这些要求就与 esriArcMap（来自应用程序类目）不同，后者的确需要用户界面组件并且只能包含在 Windows 类库中。

类库中的所有组件都共享放置在类库上的相同要求的集合中。不能将一个类库拆分为更小的单元来分发。类库为所有的组件定义了命名空间，并以适合用户所选 API 的形式出现。

- 类型库（Type Library） －COM
- .NET 互操作套件（Interop Assembly）－.NET
- Java 包－Java
- 头文件－C++

在这里，线程安全是指对多线程并发对象的访问。

伸缩性

ArcGIS Engine 和 ArcGIS Server 中的 ArcObjects 组件必须是可伸缩的。Engine 对象是可伸缩的，因为它们可被用在许多不同类型的应用程序中；这些应用程序有的需要伸缩性而有的不需要。Server 对象也要求是可伸缩的，以确保服务器可以处理许多用户的连接，并保证随着服务器配置的增长，在服务器上运行的 ArcObjects 组件性能也能提高。

系统的伸缩性通过许多变量，包括系统的硬件和软件来实现。在这点上，ArcObjects 通过有效地使用对象中的内存和在多线程过程中执行对象的能力来支持伸缩性。

在讨论多线程应用程序时有两点考虑：线程安全和伸缩性。所有对象的线程安全非常重要，但是仅有线程安全的对象并不是自动地意味着创建多线程应用程序是非常简单的事情，或者所创建的应用程序在性能上的确会有极大的改进。

包含在基础服务、数据访问、地图分析和地图表达类目中的 ArcObjects 组件都是线程安全的。这就意味着应用程序开发人员可以在多线程应用程序中使用它们；但是，程序员仍然必须以这样一种方式编写多线程代码，以避免由于死锁等情况造成的应用程序失败。

除了 ArcGIS9 中的 ArcObjects 组件需要线程安全之外，对 ArcObjects 使用的单元线程模型进行了分析，以确保 ArcOjects 可以在多线程进程中有效地运行。一个称为“Threads in Isolation”的模型被用来保证 ArcObjects 体系能被有效地使用。

经典的每过程独生模型指应用线程的所有线程仍会访问驻留了独生对象的主线程。这样可以有效地将应用程序变为单线程应用程序。

这个模型的工作方式是：将跨线程通信变为一个绝对最小化的通信，或者最好完全移除该通信。为了使这种工作方式生效，ArcGIS 9 中的独生对象更改为每线程独生和每过程非独生。在一个过程中驻留多个独生对象的资源开销要小于停止跨线程通信获得的性能，在跨线程通信中，在某个线程（一般是主 STA）中创建一个独生对象，而在另一个线程中访问该独生对象。

ArcGIS 是可扩展系统，而为了使“Threads in Isolation”模型生效，所有的独生对象必须遵循这个规则。如果用户在开发中创建了独生对象，必须保证这些对象遵循这个规则。

多平台支持

如前所述，ArcObjects 组件都是 C++对象，意味着具有 C++编译器的任何计算机平台都是 ArcObjects 的潜在平台。除 C++编译器外，平台还必须支持 ArcObjects 需要的一些基本服务。

微软的 Windows 是一个小端格式的平台，而 Sun 公司的 Solaris 是一个大端格式的平台。

尽管许多平台差异不会影响 ArcObjects 组件的开发方式，但是有些差异还是会影响代码开发的方式。不同计算体系结构的字节顺序在小端格式和大端格式之间会发生变化。这最容易在对象向磁盘读写数据时看到。用某种计算平台写入的数据如果用另一种计算平台读取会不兼容，除非执行一些解码。所有的 ArcGIS Engine 和 ArcGIS Server 对象都支持这种多平台保存模型。ArcObjects 组件总是使用小端格式模型保存自己。当对象读取保存后的数据时，就会将其转换为合适的本地字节顺序。除了字节顺序差异外，平台之间还存在其他一些功能上的差异；例如，Windows 和 Unix 的目录结构使用不同的分隔符，分别为“\”和“/”。另一个例子就是与具体平台有关的功能，如 OleDB 等。

兼容性

尽管 ArcGIS 版本的目的是限制 API 中的变化，但开发者还是应该在以后的版本中彻底测试自己的软件。

维护 ArcGIS 系统不同版本之间的兼容性非常重要，可以保证外部开发人员不用修改其代码以在最新版本中工作。维护对象级别的兼容性是

ArcGIS9 开发工作的一个基本目标。尽管对象级别的兼容性得到了维护，但 ArcGIS 8 和 ArcGIS 9 体系结构之间还是有一些变化会影响开发人员，主要是软件编译方面的问题。

尽管用在 ArcGIS 8 中的软件只需很小的改变就可以用在 ArcGIS 9 中，但理解这一点很重要：为了在 ArcGIS 9 上实现 ArcObjects 体系结构中的任何已有投入，用户都必须审查有关 ArcGIS Engine 、ArcGIS Server 和 ArcGIS Desktop 的开发。

ESRI 理解统一软件架构的重要性，并且已经对 ArcGIS 9 进行了重大调整，以便在 ArcObjects 上投资可以在多个产品中实现。如果用户已经为 ArcGIS 8 体系结构创建了一些组件，就应该考虑新的 ArcGIS 9 体系结构会如何影响用户开发的组件的实现方式。

ArcGIS 应用程序接口

ArcObjects 的功能可以通过四个应用程序接口（APIs）来访问。选择使用哪个 API 并不是一个简单的问题，而要取决于许多因素，包括：要开发的 ArcGIS 产品、要开发的最终用户功能以及使用特定开发语言的经验。ArcGIS Engine 支持的四种 API 是：

重要的是不要混淆 VC++支持的 COM API 与本地 C++ API。

- COM——任何 COM 生成语言（Visual Basic、Visual C++、Delphi 等）都可以使用这个 API。
- .NET——这个 API 支持 Visual Basic.NET 和 C#。
- Java——Sun 公司的 Java2 平台标准编辑器。
- C++ ——微软 VC++6.0 、微软 VC++.NET2003 、Sun Solaris Forte6 Update2、Linux GCC3.2 支持此 API。

操作 ArcObjects 时开发者可以使用 ArcObjects 提供的功能或用自己的组件扩展 ArcObjects 的功能。在引用这些 APIs 时，使用和扩展 ArcObjects 架构之间是存在差别的。

使用 API

由于 ArcObjects 是用 C++开发的，有的情况下出于性能考虑使用了一些与 C++兼容的数据类型。这些性能考虑大多数会影响 ArcObjects 的内部 ，因此，使用一般接口不应该对性能或 ArcObjects 开发产生不利的影响。

四个 APIs 全都支持使用 ArcObjects 的功能；但是，并不是所有 ArcObjects 实现的接口都在所有平台上支持。有些情况下，接口使用的数据类型与某种 API 不兼容。在这种情况下，提供了接口的其他实现以便开发者使用。接口后缀为“GEN”的命名习惯被用来表示这种接口；IFoo 将有 IFooGEN 接口。所有 APIs 都支持这种接口；但是，如果 API 支持非 GEN 接口，则可以继续使用特定 API 接口。

扩展 API

扩展 ArcObjects 就是指创建用户自己创造的对象并将它们添加到 ArcObjects 架构中。ArcObjects 几乎在所有方面都可以进行扩展。对 ArcObjects 体系结构扩展的支持随 API 不同而变化，而且有时也随 API 语言的不同而变化。

COM API 提供了最强大的系统扩展能力。这个 API 的限制就是对 Visual Basic 语言的限制。Visual Basic 不支持具有以下特征或特征之一的接口的实现：

- 不是从 IUnknown 和 IDispatch 继承而来的接口。从 IGeometry 接口继承而来的 Icurve 接口就是由于这个原因不能在 VB 中实现。
- 名字以下划线开头的接口方法。用户在 ArcObjects 中找不到以“_”开头的函数。
- 方法中参数使用的数据类型不被 VB 支持。由于这个原因 IActiveView 不能在 VB 中实现。

除了 VB 支持的接口限制外，COM 集合的二进制重用技术也不被 VB 支持。这就意味着 ArcObjects 体系中有部分组件是不能扩展的；自定义特性（Custom Features）就是这样一个例子。实际上，上述 VB 限制对绝大多数开发者没有多大的影响，因为受影响的 ArcObjects 所占百分比很小，而且对于这么小百分比的部分而言，开发者似乎没有必要扩展 ArcObjects 体系。其他 COM 语言如 Visual C++等就没有任何限制。

AO 支持的 API 之间的主要区别在数据类型上。所有的 API 完全支持右边显示的所有自适应数据类型。非 OLE 自适应数据间会产生的差异。

除了使用非 OLE 自适应数据类型的接口外，.NET API 支持 ArcObjects 的完全扩展（下表列出了所有 OLE 自适应数据类型）。

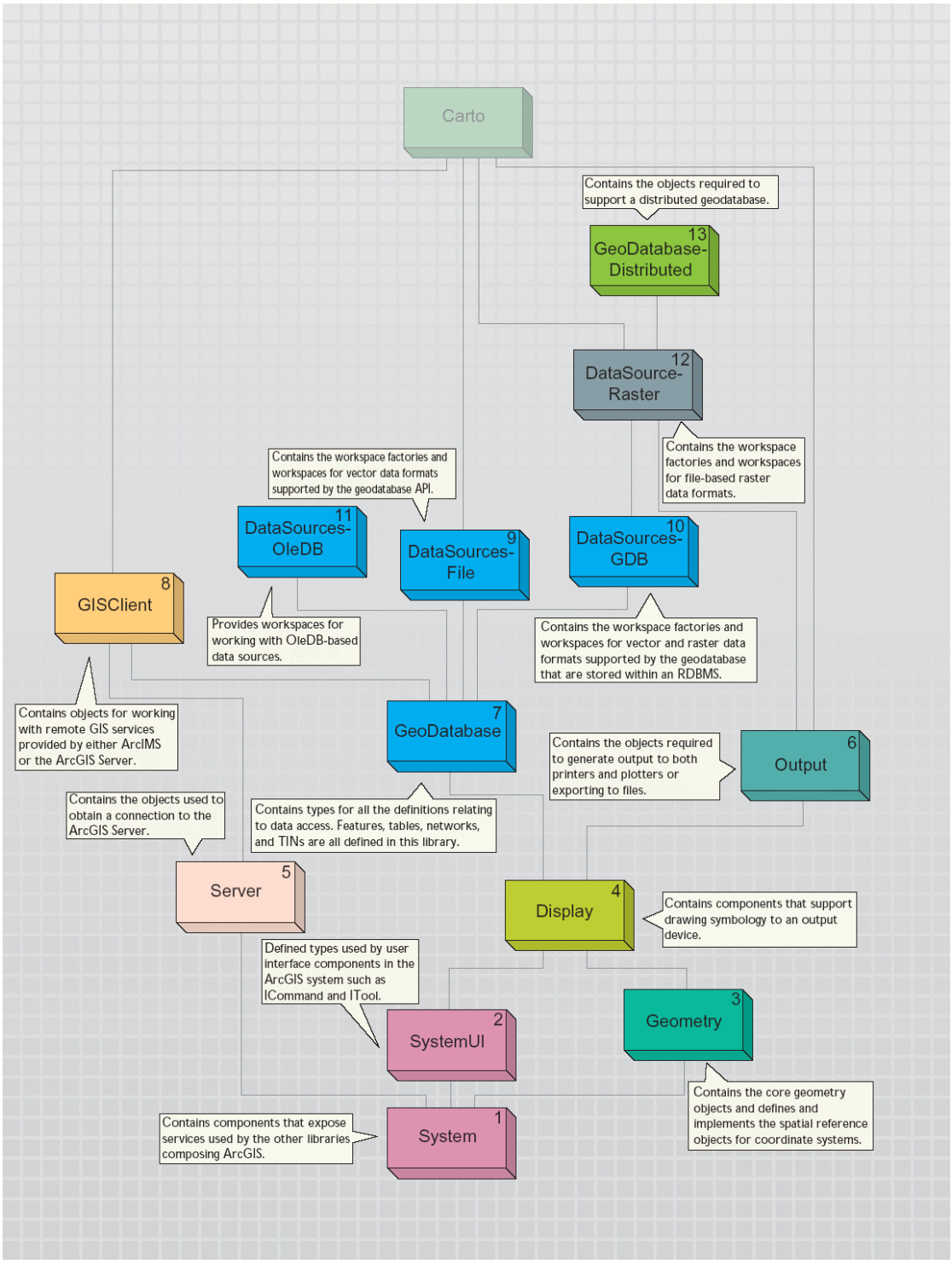
类型	描述
Boolean	值为 True 或 False 的数据项
unsigned char	8 位无符号数据项
double	64 位 IEEE 浮点数
float	32 位 IEEE 浮点数
int	有符号整型，大小与系统有关
long	32 位有符号整型
short	16 位有符号整型
BSTR	以长度为前缀的字符串
CURRENCY	8 字节固定点数
DATE	64 位，从 1899 年 12 月 30 日起的日期浮点分数
SCODE	用于 16 位系统，与 VT_ERROR 相应的内嵌错误
Typed enum myenum	有符号整型，大小与系统有关
Interface IDispatch	指向 IDispatch 接口的指针
Interface IUnkown	指向不是源自 IDispatch 的接口的指针
dispinterface Typename *	指向源自 IDispatch 的接口的指针
Coclass Typename *	指向一个 Coclass 名称（VT_UNKNOWN）的指针
[oleautomation] interface Typename*	指向源自 IDispatch 的接口的指针
SAFEARRAY (TypeName)	TypeName 为上面任一种类型，这些类型的数组
TypeName*	TypeName 为上面任一种类型，指向某个类型的指针
Decimal	96 位无符号二进制整型，以 10 的变量幂作为位制。一个十进制数据类型提供一个数的大小和位制（如坐标）。

OLE 自适应数据类型

Java 和 C++ APIs 在扩展 ArcObjects 方面有类似的限制。使用这些 APIs 的开发者只能创建自定义命令和工具。这些命令和工具可以在 ToolbarControl 中使用。这看起来是一种严厉的限制，但尽管存在这种限制，这些 APIs 仍然对开发者具有很强的吸引力。ToolbarControl

和其他 ArcGIS 控件一起提供了丰富的开发环境。ArcGIS Desktop 应用程序是具有大量功能的专业 GIS 应用程序，但是如果简单来看，这些应用程序可以分解为一系列工具条、一个 TOC 和地图浏览区。通过添加命令和工具可以扩展桌面应用程序。通过类似的方法，开发者可以使用四种 ArcGIS Engine APIs 中的任何一种构建具有丰富功能的应用程序。

COM 和.NET APIs 只在微软 Windows 平台上被支持，而 Java 和 C++ APIs 可以在 ArcGIS Engine 支持的所有平台上都支持。


Carto
Contains the objects required to support a distributed geodatabase.
13 GeoDatabase-Distributed
12 DataSource-Raster
Contains the workspace factories and workspaces for file-based raster data formats.
Contains the workspace factories and workspaces for vector data formats supported by the geodatabase API.
11 DataSources-OleDB
9 DataSources-File
10 DataSources-GDB
8 GISClient
Provides workspaces for working with OleDB-based data sources.
Contains the workspace factories and workspaces for vector and raster data formats supported by the geodatabase that are stored within an RDBMS.
Contains objects for working with remote GIS services provided by either ArcIMS or the ArcGIS Server.
7 GeoDatabase
Contains the objects required to generate output to both printers and plotters or exporting to files.
6 Output
Contains the objects used to obtain a connection to the ArcGIS Server.
Contains types for all the definitions relating to data access. Features, tables, networks, and TINs are all defined in this library.
5 Server
4 Display
Contains components that support drawing symbology to an output device.
Defined types used by user interface components in the ArcGIS system such as ICommand and ITool.
2 SystemUI
3 Geometry
Contains the core geometry objects and defines and implements the spatial reference objects for coordinate systems.
Contains components that expose services used by the other libraries composing ArcGIS.
1 System

要了解每个类库的全部讨论，请参考 ArcGIS 开发帮助系统中类库参考部分的类库概览。

下面将概述 ArcGIS Engine 中的各个类库。本节中的各个模型图展示了 ArcGIS Engine 的类库架构。理解类库结构、它们的依赖关系和基本功能将有助于开发人员了解 ArcGIS Engine 的组件。

下面根据依赖关系的顺序对类库进行讨论。模型图中在每个类库框的右上角显示了其序列号。例如，作为 ArcGIS 体系结构基础的 System 类库，其编号为 1，而编号为 7 的 GeoDatabase 类库依赖于模型图中其前面的 6 个类库—SyStem、SystemUI、Geometry、Display、Server 和 Output。

System 类库

System 类库是 ArcGIS 体系结构中最底层的类库。System 类库包含给构成 ArcGIS 的其他类库提供服务的组件。System 类库中定义了大量开发者可以实现的接口。AoInitializer 对象就是在 System 类库中定义的，所有的开发者必须使用这个对象来初始化 ArcGIS Engine 和解除 ArcGIS Engine 的初始化。开发者不能扩展这个类库，但可以通过实现这个类库中包含的接口来扩展 ArcGIS 系统。

了解类库的依赖关系顺序非常重要，因为它会影响到开发者在软件开发时与类库的交互方式。例如，C++程序员必须根据类库依赖关系顺序依次包含类库，以保证能正确编译。对依赖关系的理解也有助于部署开发人员开发的应用程序。

SystemUI 类库

SystemUI 类库包含用户界面组件接口定义，这些用户界面组件可以在 ArcGIS Engine 中进行扩展。包含 ICommand、ITool 和 IToolControl 接口。开发者用这些接口来扩展 UI 组件，ArcGIS Engine 开发人员自己的组件将使用这些 UI 组件。这个类库中包含的对象是一些实用工具对象，开发人员可以通过使用这些对象简化用户界面的开发。开发者不能扩展这个类库，但可以通过实现这个类库中包含的接口来扩展 ArcGIS 系统。

Geometry 类库

Geometry 类库处理存储在要素类中的要素几何图形或形状或其他图形元素。大多数用户会遇到的基本几何图形对象为 Point、MultiPoint、Polyline 和 Polygon 。除了这些顶级实体外，就是作为多义线和多边形的组成部分的几何图形，是组成几何图形的子要素。它们是 Segement、Path 和 Ring。Polyline 和 Polygon 由一系相连接的、构成列 Path 的片段组成.一个片段由两个不同的点，即起始点和结束点，以及一个定义这两点之间弯曲度的元素类型组成。片段的类型有 CircularArc、Line、EllipticArc 和 BezierCurve。所有几何图形对象都可以有与其顶点相关联的 Z、M 和 IDs。所有的基本几何图形对象都支持诸如 Buffer，Clip 等几何操作。几何子要素不可以由开发者扩展。

GIS 中的实体指的是现实世界中的要素;现实世界中要素的位置由一个带有空间参考的几何图形来定义。空间参考对象，包括投影坐标和地理坐标系统，都包括在 Geometry 类库中。开发者可以通过添加新的空间参考和投影来扩展空间参考系统。

Display 类库

Display 类库包含用于显示 GIS 数据的对象。除了负责实际输出图像的主要显示对象外，这个类库还包含表示符号和颜色的对象，它们用来控制在显示上绘制时实体的属性。Display 类库还包含在与显示交互时提供给用户可视化反馈的对象。开发者与 Display 最常用的交互方式就是类似于 Map 对象或 PageLayout 对象提供的视图。Display 类库的所有部分都能进行扩展；通常扩展的对象包括符号、颜色和显示反馈。

Server 类库

Server 类库包含允许用户连接并操作 ArcGIS Server 的对象。开发人员用 GISServerConnection 对象来访问 ArcGIS Server。通过 GISServerConnection 可以访问 ServerObjectsMananger 对象。用这个对象，开发人员可以操作 ServerContext 对象，以处理运行于服务器上的 ArcObjects。开发人员还可以用 GISClient 类库与 ArcGIS Server 进行交互。

Output 类库

Output 类库用于创建图形输出到诸如打印机和绘图仪等设备及诸如增强型元文件和栅格图像格式（JPG、BMP 等）等硬拷贝格式。开发人员用这个类库中的对象及 ArcGIS 系统的其他部分创建图形输出。通常是 Display 和 Carto 类库中的对象。开发者可以为自定义设备和输出格式扩展 Output 类库。

GeoDatabase 类库

Geodatabase 类库为地理数据库提供了编程 API。地理数据库是建立在标准工业关系型和对象关系数据库技术之上的地理数据仓库。Geodatabase 类库中的对象为 ArcGIS 支持的所有数据源提供了一个统一编程模型。GeoDatabase 类库定义了许多由 ArcObjects 架构中更高级的数据源提供者实现的接口。开发者可以扩展地理数据库，以支持特定类型的数据对象（要素、类等）；此外，Geodatabase 类库还有用 PlugInDataSource 对象添加的自定义矢量数据源。地理数据库支持的本地数据类型不能扩展。

GISClient 类库

GISClient 类库允许开发者使用 Web 服务;这些 Web 服务可以由 ArcIMS 和 ArcGIS Server 提供。GISClient 类库中包含用于连接 GIS 服务器以

使用 Web 服务的对象。该类库支持 ArcIMS 的图像和要素服务。GISClient 类库提供以无态方式直接或通过 Web 服务目录操作 ArcGIS Server 对象的通用编程模型。在 ArcGIS Server 上运行的 ArcObjects 组件不能通过 GISClient 接口来访问。要直接获得访问在服务器上运行的 ArcObjects，开发人员应使用 Server 类库中的功能。

DataSourcesFile 类库

DataSourcesFile 类库包含用于基于文件数据源的 GeoDatabase API 实现。这些基于文件的数据源包括 shapefile、coverage、TIN、CAD、SDC、StreetMap 和 VPF。开发者不能扩展 DtaSourcesFile 类库。

DataSourcesGDB 类库

DataSourcesGDB 类库包含用于数据库数据源的 GeoDatabase API 实现。这些数据源包括 Microsoft Access 和 ArcSDE 支持的关系型数据库管理系统—IBM、DB2、Informix、Microsoft SQL Server 和 Oracle。开发者不能扩展 DataSourcesGDB 类库。

DataSourcesOleDB 类库

DataSourcesOleDB 类库包含用于 Microsoft OLE DB 数据源的 GeoDatabase API 实现。此类库只能用在 Microsoft Windows 操作系统上。这些数据源包括支持数据提供者和文本文件工作空间的所有 OLE DB。开发者不能扩展 DataSourcesOleDB 类库。

DataSourcesRaster 类库

栅格数据对象（RDO）是一个 COM API，提供支持基于文件的栅格数据的显示和分析功能。

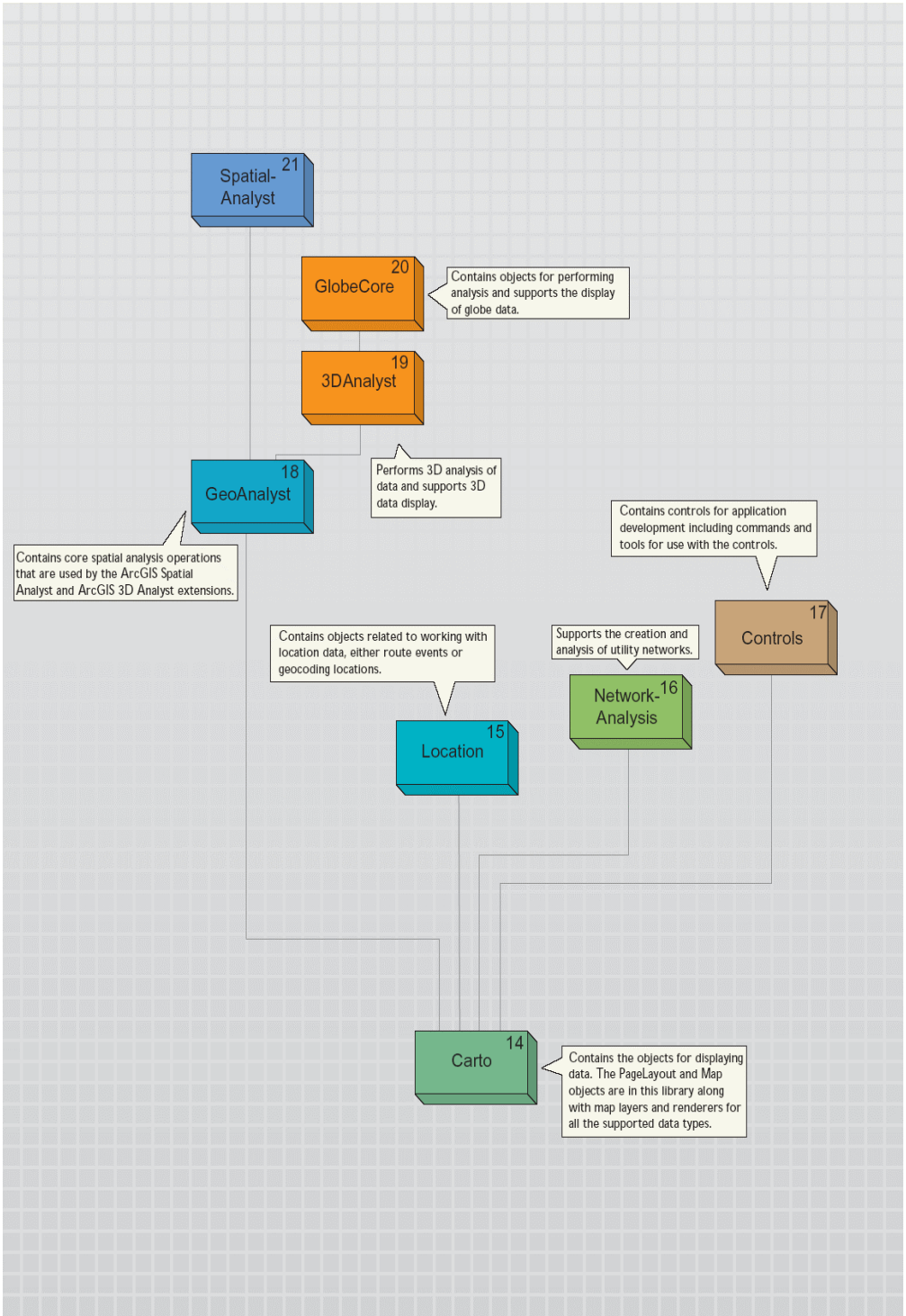
DataSourcesRaste 类库包含用于栅格数据源的 GeoDatabase API 实现。这些数据源包括 ArcSDE 支持的关系型数据库管理系统—IBM、DB2、Informix、Microsoft SQL Server 和 Oracle，以及其支持的 RDO 栅格文件格式。当需要支持新的栅格格式时，开发者不扩展这个类库，而是扩展 RDO。开发者不能扩展 DataSourcesRaster 类库。

GeoDatabaseDistributed 类库

GeoDatabaseDistributed 类库通过提供地理数据库数据导入和导出工具，可以支持对企业级地理数据库的分布式访问。开发者不能扩展 GeoDatabaseDistribute 类库。

Carto 类库

Carto 类库支持地图的创建和显示;这些地图可以在一幅地图或由许多地图及其地图元素组成的页面中包含数据。PageLayout 对象是驻留一幅或多幅地图及其底土元素的容器。地图元素包括指北针、图例、比例尺等。Map 对象包括地图上所有图层都有的属性—空间参考、地图比例尺等，以及操作地图图层的方法。可以将许多不同类型的图层加载到地图中。


Spatial-Analyst 21
GlobeCore 20
Contains objects for performing analysis and supports the display of globe data.
3DAnalyst 19
Performs 3D analysis of data and supports 3D data display.
GeoAnalyst 18
Contains core spatial analysis operations that are used by the ArcGIS Spatial Analyst and ArcGIS 3D Analyst extensions.
Contains controls for application development including commands and tools for use with the controls.
Controls 17
Contains objects related to working with location data, either route events or geocoding locations.
Supports the creation and analysis of utility networks.
Network-Analysis 16
Location 15
Carto 14
Contains the objects for displaying data. The PageLayout and Map objects are in this library along with map layers and renderers for all the supported data types.

ArcGIS Server 为其 MapServer 使用 MapServer 对象。

不同的数据源通常有相应的图层负责数据在地图上的显示，矢量要素由 FeatureLayer 对象处理，栅格数据由 RasterLayer 对象处理，TIN 数据由 TinLayer 对象处理，等等。必要的话，图层可以处理与之相关数据的所有绘图操作，但通常图层都是一个相关的 Renderer 对象。Renderer 对象的属性控制着数据在地图中的显示方式。Renderers 通常用 Display 类库中的符号来进行实际绘制，而 Renderer 只是将特定符号与待绘实体的属性相匹配。Map 对象和 PageLayout 对象可以包含元素。元素用其几何图形定义其在地图或页面上的位置，用行为控制元素的显示。包括用于基本形状、文字标注和复杂标注等的元素。Carto 类库还支持地图注释和动态标注。

尽管开发者可以在其应用程序中直接使用 Map 和 PageLayout 对象,但通常来说开发者更经常使用更高级的对象，如 MapControl、PageLayoutControl 或 ArcGIS 应用程序。这些高级对象简化了一些任务，尽管它们也提供对更低级别的 Map 和 PageLayout 对象的访问，允许开发者更好的控制对象。

Map 和 Pagelayout 对象并不是 Carto 类库中提供地图和页面绘制的仅有对象。MxdServer 和 MapServer 对象都支持地图和页面的绘制，但不是绘制到窗口中，而是直接绘制到文件中。

开发者可以用 MapDocument 对象保存地图和地图文档（.mxd）中页面布局的状态，以便在 ArcMap 或 ArcGIS 控件中使用。

Carto 类库通常可以在许多方面进行扩展。自定义 Renderer、Layer 等都很普遍。自定义图层通常是向地图应用程序中加载自定义数据最简单的方法。

Location 类库

Location 类库包含支持地理编码和操作路径事件的对象。地理编码功能可以通过细粒度对象来完全控制访问，或通过 GeocodeServer 对象提供的简化 API 来访问。开发者可以创建自己的地理编码对象。线性参考功能提供对象用于向线性要素添加事件，用各种绘制方法来绘制这些事件。开发者可以扩展线性参考功能。

NetworkAnalysis 类库

NetworkAnalysis 类库提供用于在地理数据库中加载网络数据的对象并提供对象用于分析加载到地理数据库中的网络。开发者开以扩展 NetworkAnalysis 类库以便支持自定义网络追踪。这个类库目的在于操作公共网络：供气管线、电力供应线网等。

Controls 类库

开发者用 Controls 类库来构建或扩展具有 ArcGIS 功能的应用程序。ArcGIS Controls 通过封装 ArcObjects 并提供粗粒度的 API 简化了开发过程。尽管这些控件封装了细粒度的 ArcObjects，但是并不限制对这些细粒度的 ArcObjects 的访问。MapControl 和 PageLayoutControl 分别封装了 Carto 类库的的 Map 和 PageLayout 对象。ReaderControl 同时封装了 Map 和 PageLayout 对象，且在操作控件时提供了简化的 API。如果授权了地图发布程序，开发者可以以访问 Map 和 PageLayout 控件类似的方式访问内部对象。Controls 类库还包含实现一个目录表的 TOCControl 及驻留操作合适控件的命令和工具的 ToolbarControl。

Map 和 PageLayout 控件的内容可以通过程序指定，或者加载到 MapDocument 中。.

RenderControl 只支持发布地图文件（PMF）。

开发者通过创建自己的用于操作控件的命令和工具来扩展 Controls 类库。为此 Controls 类库提供 HookHelper 对象。这个对象使得创建一个操作任何控件及操作诸如 ArcMap 这样的 ArcGIS 应用程序的命令变得非常简单。

ArcGIS Engine 带有 150 多个命令。

GeoAnalyst 类库

GeoAnalyst 类库包含支持核心空间分析功能的对象。这些功能用在 SpatialAnalyst 和 3Danalyst 两个类库中。开发者可以通过创建新类型的栅格操作来扩展 GeoAnalyst 类库。为使用这个类库中的对象，需要 ArcGIS Spatial Analyst 或 3D Analyst 扩展模块许可，或者 ArcGIS Engine 运行时空间分析或 3D 分析选项许可。

3DAnalyst 类库

3Danalyst 类库包含操作 3D 场景的对象，其方式与 Carto 类库包含操作 2D 地图的对象类似。Scene 对象是 3Danalyst 类库中主要对象之一，因为该对象与 Map 对象一样，是数据的容器。Camera 和 Target 对象规定在考虑要素位置与观察者关系时场景如何浏览。一个场景由一个和多个图层组成；这些图层规定了场景中包含的数据及这些数据如何显示。开发者很少扩展 3Danalyst 类库。为使用这个类库中的对象，需要 ArcGIS 3D Analyst 扩展模块许可或 ArcGIS Engine 运行时 3D 分析选项许可。

GlobeCore 类库

GlobeCore 类库包含操作 globe 数据的对象，其方式与 Carto 类库包含操作 2D 地图的对象类似。Globe 对象是 GlobeCore 类库中主要对象之一，因为该对象与 Map 对象一样，是数据的容器。GlobeCamera 对象规定在考虑 globe 位置与观察者关系时 golbe 如何浏览。一个 golbe 有一个和多个图层；这些图层规定了 golbe 中包含的数据及这些数据如何

显示。

GlobeCore 类库中有一个开发控件及与其一起使用的命令和工具。该开发控件可以与 Controls 类库中的对象协同使用。

开发者很少扩展 GlobeCore 类库。为使用这个类库中的对象，需要 ArcGIS 3D Analyst 扩展模块许可或 ArcGIS Engine 运行时 3D 分析选项许可。

SpatialAnalyst 类库

SpatialAnalyst 类库包含在栅格数据和矢量数据上执行空间分析的对象。开发者通常使用这个类库中的对象，而不扩展这个类库。为使用这个类库中的对象，需要 ArcGIS 空间分析扩展模块许可或 ArcGIS Engine 运行时空间分析选项许可。

3 使用ArcGIS控件进行开发

ArcGIS Engine 提供大量高级开发控件，使开发人员可以建立或扩展 GIS 应用程序和创建高质量的地图用户界面。这些控件包括 MapControl、PageLayoutControl、ReaderControl、TOCCONTROL和ToolbarControl。GlobeControl和SceneControl也可以获得，但使用这两个控件的应用程序必须具有 ArcGIS Engine 的 3D 分析选项授权。

本章的主要内容包括：

- *概览各个控件*
- *讨论各个 ArcGIS 控件共同的主题和概念*
- *考虑建立带或不带 ToolbarControl 的应用程序*

什么是 ArcGIS 控件？

ArcGIS 控件是高级开发组件，首先它们允许开发人员建立和扩展 GIS 应用程序，其次它们提供了图形用户界面（GUI）。

下列所有 ArcGIS 控件都可以作为 ActiveX 控件、.NET Windows 控件和 Visual JavaBean 获得：

- MapControl—类似于“数据（data）视图”
- PageLayoutControl
- ToolbarControl
- TOCControl（内容表控件）
- SceneControl
- GlobeControl
- ReaderControl

部署用 GlobeControl 或 SceneControl 建立的应用程序需要 ArcGIS Engine 的 3D 分析选项。

ArcGIS 控件可以通过两种方式建立应用程序：首先，ArcGIS 控件可以嵌入到现有的应用程序中以增加制图功能；其次，ArcGIS 控件可用于创建新的独立应用程序。在前一种情况中，单个的 ArcGIS 控件可以嵌入到应用程序中，在后一种情况中，TOCControl 和 ToolbarControl 可以与其他 ArcGIS 控件结合起来组成应用程序框架的组成部分。

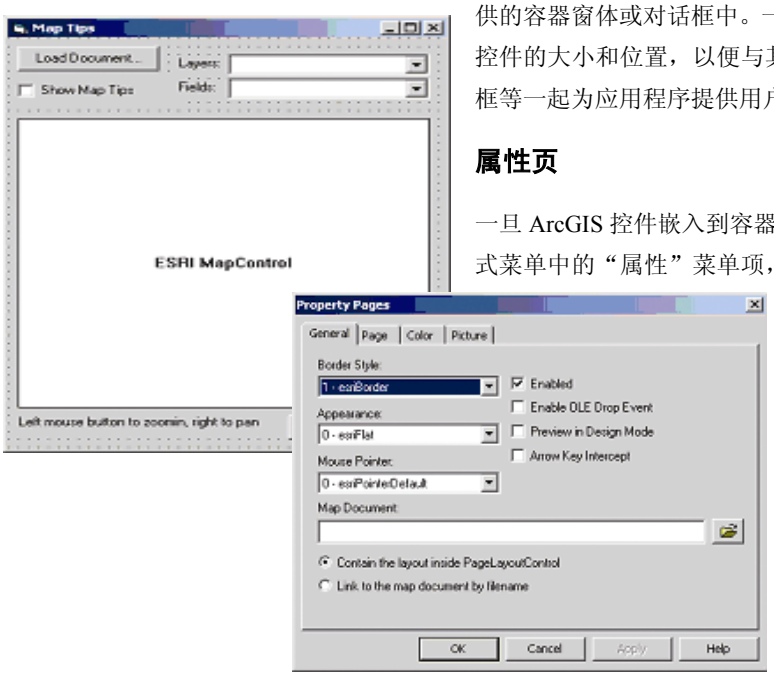
为了使用 ArcGIS 控件有效地建立应用程序，应该理解所有 ArcGIS 控件共同的一些主题和概念。

嵌入式组件

每个 ArcGIS 控件都是一个嵌入式组件，可以拖放到可视化设计环境提供的容器窗体或对话框中。一旦放在容器内，就可以改变 ArcGIS 控件的大小和位置，以便与其他嵌入式组件，如命令按钮和组合框等一起为应用程序提供用户界面。

属性页

一旦 ArcGIS 控件嵌入到容器中，通过右键点击该控件并选择弹出式菜单中的“属性”菜单项，就可以在大多数可视化设计环境中访问各个 ArcGIS 控件的属性页集合。这些属性页提供了选择控件属性和方法的快捷方式，使开发人员只需写很少代码甚至不用写代码就可以建立应用程序。



通过属性页可以访问的所有属性都可以由开发人员用代码设置。

ArcObjects

每个 ArcGIS 控件都封装了粗粒度 ArcObjects，同时又提供对细粒度 ArcObjects 的访问，这样简化了开发过程。例如，PageLayoutControl 封装了布局（PageLayout）对象。布局（PageLayout）对象至少包含一个 MapFrame 元素，该 MapFrame 元素包含一个地图（Map）对象，而地图（Map）对象又可以包含多个栅格、矢量或自定义图层（Layer）对象。每个 ArcGIS 控件为其封装的 ArcObjects 上频繁使用的属性和方法提供了快捷方式。例如，MapControl 有一个 SpatialReference 属性，该属性就是地图（Map）对象的 SpatialReference 属性的一个快捷方式。每个 ArcGIS 控件还有一些执行公共任务的帮助方法。例如，MapControl 有一个 AddShapeFile 方法。ArcGIS 控件一般都是开发应用程序的起始点，不仅因为这些控件提供了用户界面，而且还因为通过它们可以直接进入对象模型。

事件

每个 ArcGIS 控件都会响应终端用户的键盘和鼠标交互而触发事件。响应发生在控件内的行为会触发一些其他事件。例如，当装载一个地图文档到 MapControl 时，就会触发 OnMapReplaced 事件；或者当通过拖放将一个对象在 MapControl 上拖动时，就会触发 OnOleDrop 事件。

伙伴控件

ToolbarControl 和 TOCControl 都要与另一个“伙伴控件”协同工作。“伙伴控件”一般是 MapControl、PageLayoutControl、ReaderControl、SceneControl 或 GlobeControl。“伙伴控件”可以在设计时通过控件的属性页设置（在支持属性页功能的开发环境中可以这样设置），或者用 SetBuddyControl 方法编程设置。

要了解控件如何协同工作方面更详细的信息，请参阅下面的“TOCControl”和“ToolbarControl”部分。

控件命令

ArcGIS Engine 提供了一系列命令、工具和菜单与 ArcGIS 控件协同工作。例如，MapControl 和 PageLayoutControl 有一套地图导航、要素选择和图形元素命令与其协同工作。类似的，SceneControl、GlobeControl 和 ReaderControl 也有一套命令与其协同工作。对于使用单个控件的应用程序，这些命令可以通过编程创建命令的一个新例程并将该控件传递给命令的 OnCreate 事件而直接操作控件。对于使用 ToolbarControl 及其“伙伴控件”的应用程序，这些命令可以通过设计阶段设置属性页或通过编程设置，或者如果 ToolbarControl 处于定制模式可以在运行时由终端用户设置。

开发人员也可以通过创建他们自己的自定义命令、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。要了解有关如何使用 HookHelper 对象建立自定义命令方面的信息，请参考第六章“开发情景”中的“建立应用程序”情景。

地图制作

ArcGIS Desktop 应用程序可用于预制作能被装载到 ArcGIS 控件中的文档，以便快速生产高质量的地图。例如，可以用 ArcMap 制作能被装载到 MapControl 和 PageLayoutControl 中的地图文档。预先制作文档可以节省大量时间，因为它不必从零开始编程建立地图及其符号体系。一旦文档被装载到 ArcGIS 控件中，如果随后需要改变其外观，就可以通过对象模型编程访问任何图层、元素和符号。

下表总结了可以被装载到各个 ArcGIS 控件中的文档类型。

	地图文档（*.mxd, *.mxt）	图层文件（*.lyr）	Scene 文档（*.sxd,*.sxt）	Globe 文档（*.3dd,*.sdt）	发布地图文件（*.pmf）		
					不允许装载到定制应用程序中（只能装载到 ArcReader 应用程序）	允许装载到定制应用程序中	允许装载到定制应用程序中且其内容的访问不受限制
MapControl	是	是	否	否	否	否	是
PageLayoutControl	是	**是	否	否	否	否	是
SceneControl	否	**是	是	否	否	否	否
GlobeControl	否	**是	否	是	否	否	否
ReaderControl	否	否	否	否	否	是	是
*ArcReaderControl	否	否	否	否	否	是	是

**ArcReaderControl 只能用 ArcGIS Publisher 扩展模块获得。然而，这里列出来是由于其与 ReaderControl 的类似性。*

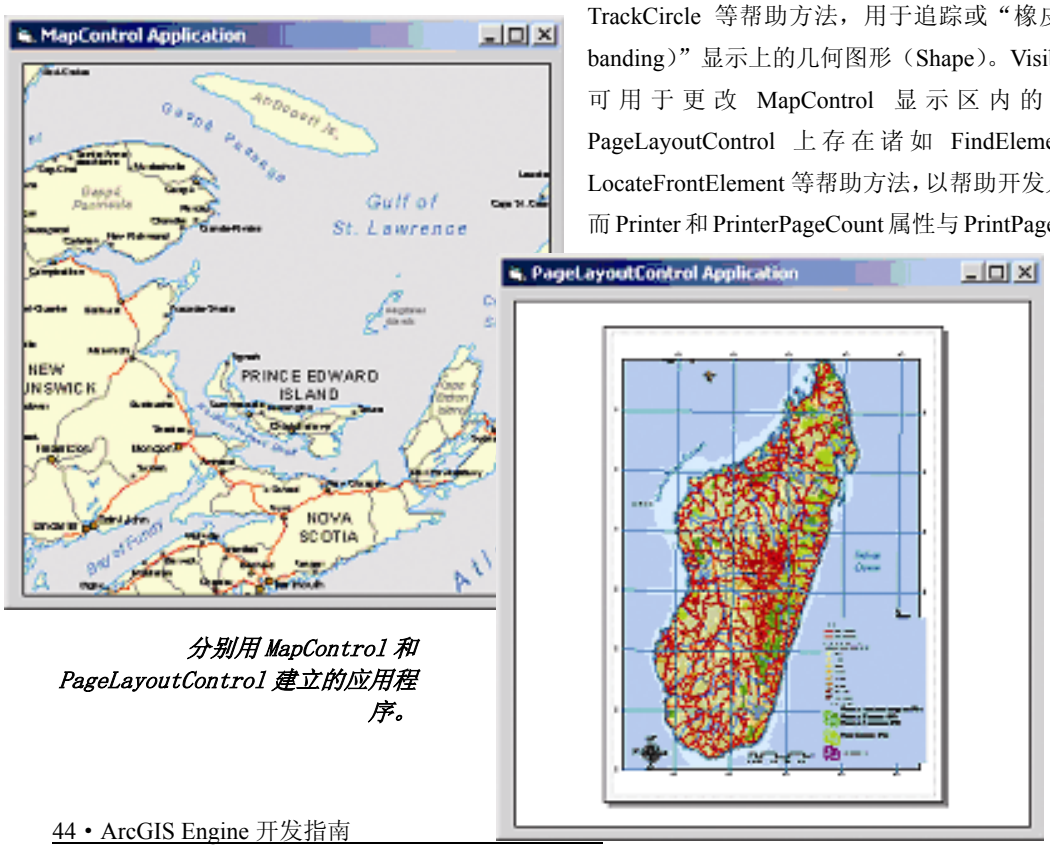
***ArcGIS 控件上没有可用的属性直接装载图层文件（*.lyr）。但是它们可以间接地通过 MapDocument 对象装载。*

MapControl 和 PageLayoutControl 对应于 ArcMap 桌面应用程序的“数据”和“布局”视图。MapControl 封装了 Map 对象，而 PageLayoutControl 封装了 PageLayout 对象。用 ArcMap 创作的地图文档可以装载到 MapControl 和 PageLayoutControl 中，以节省开发人员编程创作地图的时间。

地图文档可以在设计时通过 MapControl 和 PageLayoutControl 的属性页（在支持属性页功能的开发环境中可以这样做）设置，控件可以被设置为“链接”或“包含”地图文档。若为“链接”控件，则每当“链接”控件被创建时都会读取地图文档并显示该地图文档的最新更新。而如果为“包含”控件，则“包含”控件会将地图文档的内容复制到控件中，但不会显示该地图文档后来所作的任何更新。另外，地图文档页可以使用 LoadMxFile 方法通过编程装载到控件中。

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。

MapControl 上存在诸如 TrackRectangle、TrackPolygon、TrackLine 和 TrackCircle 等帮助方法，用于追踪或“橡皮圈住（rubber banding）”显示上的几何图形（Shape）。VisibleRegion 属性可用于更改 MapControl 显示区内的几何图形。PageLayoutControl 上存在诸如 FindElementByName 和 LocateFrontElement 等帮助方法，以帮助开发人员管理元素，而 Printer 和 PrinterPageCount 属性与 PrintPageLayout 方法则共同完成打印任务。

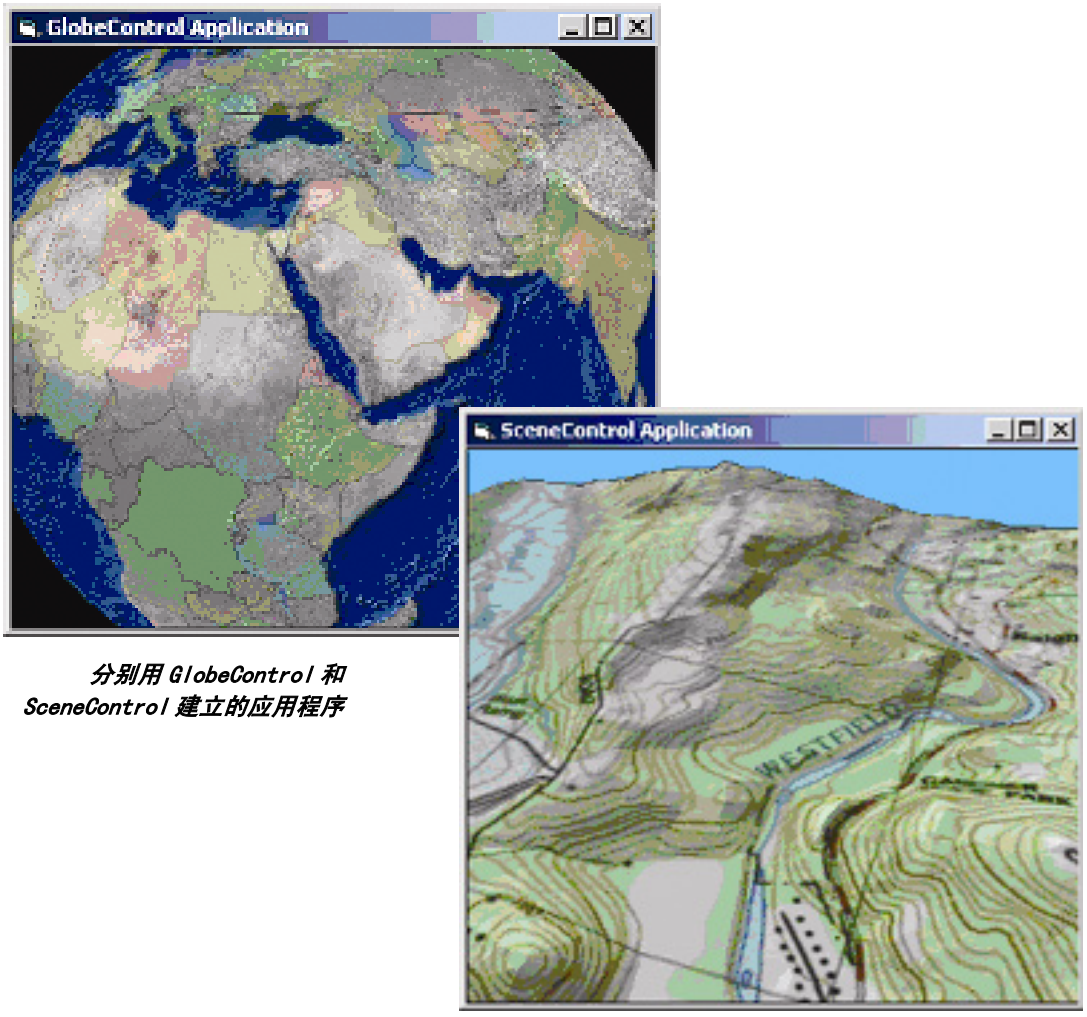


分别用 MapControl 和 PageLayoutControl 建立的应用程序。

部署用 GlobeControl 或 SceneControl 建立的应用程序需要 ArcGIS Engine 的 3D 分析运行时选项。

GlobeControl 和 SceneControl 对应于 ArcGlobe 和 ArcScene 桌面应用程序的三维视图。GlobeControl 封装了 GlobeViewer 对象，而 SceneControl 封装了 SceneViewer 对象。用 ArcGlobe 和 ArcScene 应用程序创作的 Globe 和 Scene 文档可以分别装载到 GlobeControl 和 SceneControl 中，以节省开发人员编程创作这两种地图的时间。

GlobeControl 和 SceneControl 都具有内置的导航功能，允许终端用户移动三维视图和可视化三维数据，而不必使用控件命令或自定义命令。要使用内置的导航工具，必须通过属性页或者编程设置 Navigate 属性。终端用户可以用鼠标左键对显示进行前后左右的移动，使用鼠标右键放大和缩小显示。



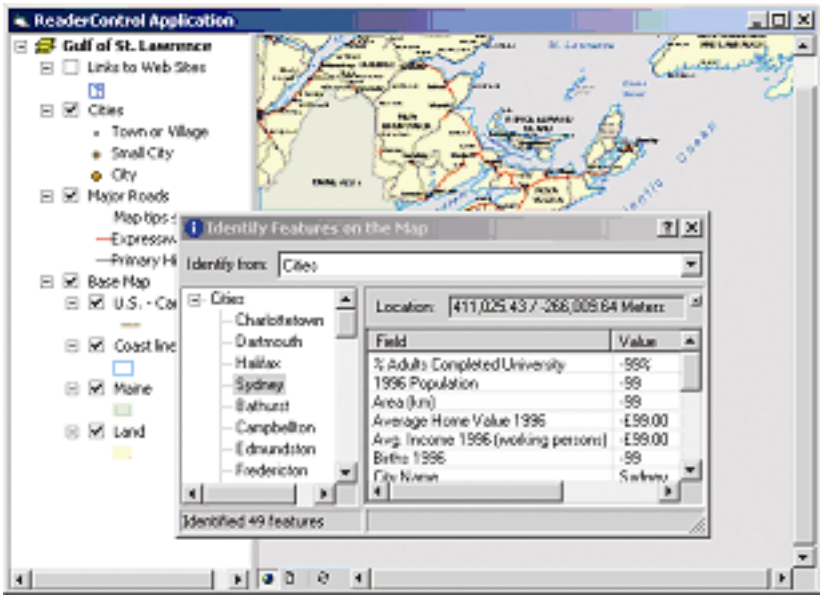
分别用 GlobeControl 和 SceneControl 建立的应用程序

ArcReader可以下载。要获得其免费的拷贝，请访问 http://www.esri.com/software/arcgis/arcreader。

ReaderControl 对应于 ArcReader 桌面应用程序的“数据”和“布局”视图及其内容表（TOC）。ReaderControl 还包含 ArcReader 桌面应用程序使用的内部窗口和工具，如 Find 窗口和 Identify 工具等。用 ArcMap 桌面应用程序创作并用 ArcGIS Publisher 扩展模块发布的发布地图文件（PMF）如果在发布时设置为允许装载到定制的 ArcReader 应用程序中，则可以装载到 ReaderControl 中。

ReaderControl 中有一个简单的自包含对象模型，提供 ArcReader 桌面应用程序的所有功能而不需要访问 ArcObjects。这样，用 ReaderControl 开发应用程序就不需要有任何使用 ArcObjects 开发程序的经验。但是，如果发布地图文件在发布时设置为不受限制地访问其内容，开发人员就可以访问底层的 ArcObjects，并以与 MapControl 和 PageLayoutControl 相似的方式使用 ReaderControl 进行开发。

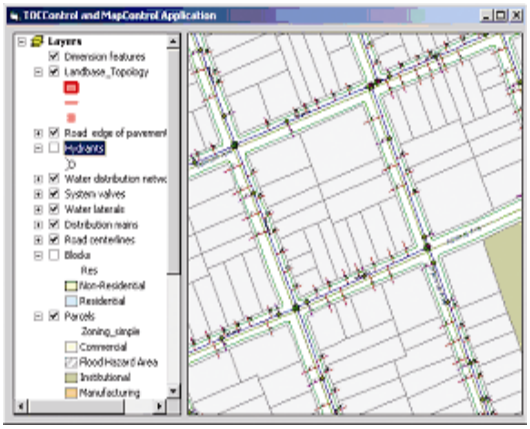
但 ArcReaderControl 并不能通过 ArcGIS Engine 获得，在这里提出来是因为该控件与 ReaderControl 很相似；ArcReaderControl 也有与 ReaderControl 相同的简单自包含对象模型。但是，ArcReaderControl 不能用作“伙伴控件”以便与 TOCControl 或 ToolbarControl 协同工作，而且开发人员也不能访问其底层的 ArcObjects。使用 ArcReaderControl 进行开发需要 ArcGIS Publisher 扩展模块，而且 ArcReaderControl 可以部署到有 ArcReader 桌面应用程序的任何机器上。



一个 ReaderControl 的应用程序

TOCControl

TOCControl 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。“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。



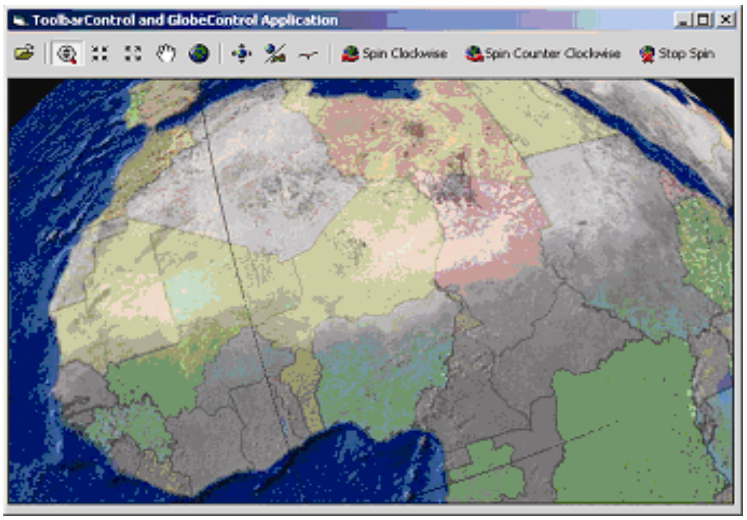
一个 TOCControl 应用程序

ToolbarControl

ToolbarControl 要与一个“伙伴控件”协同工作。“伙伴控件” 可以是一个 MapControl、PageLayoutControl、ReaderControl、SceneControl 或 GlobeControl。“伙伴控件”可以在设计时通过 ToolbarControl 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 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”显示的命令、工具和菜单。

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。

用 ToolbarControl 和 GlobeControl 建立的一个应用程序



ToolbarControl 一般要与一个“伙伴控件”协同工作，并有一个控件命令选择集，以便快速提供功能强大的 GIS 应用程序。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、窗体和多泊靠界面（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 及其类库中的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。

命令

ArcGIS Engine 提供了几套使用 ArcGIS 控件的控件命令，以便执行某种特定动作。开发人员可以通过创建他们自己的执行特定任务的定制命令来扩展这套控件命令。所有的命令对象都实现了 ICommand 接口，ToolbarControl 在适当的时候要使用该接口来调用方法和访问属性。

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。

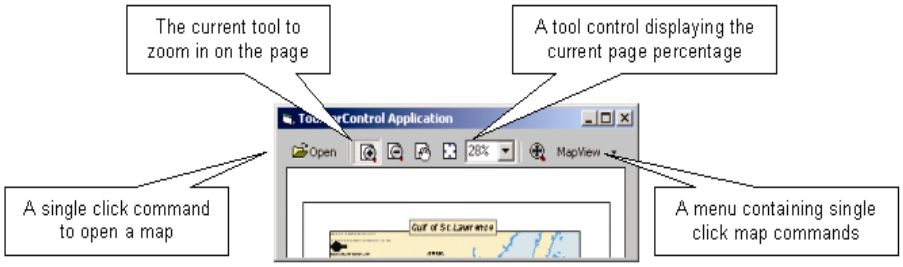
HookHelper、GlobeHookHelper 和 SceneHookHelper 可以帮助开发人员创建自定义命令以操作 ArcGIS 控件和 ArcGIS Desktop 应用程序。

- 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）”。参考第六章“开发情景”中的“建立应用程序”情景，以了解如何使用操作 MapControl、PageLayoutControl 和 ToolbarControl 的 HookHelper 对象建立自定义命令。

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。有三类命令：

- 实现了响应单击事件的 ICommand 接口的单击命令。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”的显示进行交互的命令项或工具。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 接口的命令项或工具控件。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。



可以用两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），第二种方法是给 AddItem 方法提供某个现有命令对象的一个例程。可能的话命令应通过指定一个 UID 添加到 ToolbarControl 中。如果提供了一个 UID，就可以确定这个命令是否已经被添加到 ToolbarControl 上，如果已经添加了该命令，就可以重用该命令的前一个例程。当命令没有唯一识别符并将命令的一个现有例程添加到 ToolbarControl 时，就可以在 ToolbarControl 上添加相同命令的多个例程。

工具条命令项（TOOLBARITEM）

工具条命令项就是驻留在 ToolbarControl 或工具条菜单上的单个命令或菜单。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。

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。

工具条菜单

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。要了解如何建立驻留操作 PageLayoutControl 的控件命令的“弹出式菜单”，请参考第六章“开发情景”中的“建立应用程序”情景。



作为弹出式菜单驻留工具条菜单

命令池

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。

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。要了解当 ToolbarControl 处于定制模式时如何显示定制对话框（CustomizeDialog），请参考第六章“开发情景”中的“建立应用程序”情景。

操作栈（Operation Stack）

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 Nothing），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。

建立不带 ToolbarControl 的应用程序

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，ArcGIS Engine 提供的控件命令或者使用 HookHelper、SceneHookHelper 或 GlobeHookHelper 对象的任何自定义命令将直接操作单个的 ArcGIS 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 ArcGIS 控件传递给 OnCreate 事件。例如，如果 3D 的“放大到全图（Zoom FullExtent）”命令要操作 GlobeControl，则 GlobeControl 必须作为“钩子（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 接口的工具命令，开发人员必须将该工具设置为 ArcGIS 控件中的 CurrentTool。ArcGIS 控件会将任何键盘和鼠标事件传送给该工具。

- 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。

4 开发环境

ArcObjects 基于微软组件对象模型（COM）之上。ArcGIS 应用程序的最终用户不必理解 COM，但打算基于 AO 开发应用程序或使用 AO 扩展已有 ArcGIS 应用程序的开发者必须理解 COM，即使打算用 C++、Java、.Net APIs 而不是用 COM 开发也要理解 COM。需要理解的层次取决于定制和开发的深度。至少需要浏览组件对象模型和使用 ArcObjects 开发部分，然后继续所选的特定 API 部分。

每个特定的 API 部分介绍了支持语言的编程技术并详细阐述了有关使用 ArcObjetcs 进行开发的高级特性。

本章的主题包括：

● 微软组件对象模型 ● 使用 ArcObjects 进行开发

● Visual Basic 平台和开发环境 ● Visual C++ ● .NET API

● Java API ● C++ API

在明确讨论 COM 之前，有必要先一般性地思考一下软件组件使用的广泛性。在软件组件背后有许多的驱使因素，但最基本的就是软件开发是个昂贵和耗时的风险行为。

在理想的世界里，假如曾经写了一段代码，然后通过不同的开发工具一次一次的反复使用它，这种情况是可能的，即便是在最初的开发者不能预见的环境下也能使用。理想情况下，对最初开发者生成的函数功能的改变能在不需要目前用户改变和重编译他们的代码情况下实现。

早期重用代码块的尝试就是创建类库的进展，这些类库通常用 C++开发。早期的尝试有很多的限制，较为明显的是：共享系统组件的困难（共享二进制 C++组件是非常困难的——大部分的尝试是共享源代码）；不重新编译就很难保存和更新 C++组件的问题；缺乏好的建模语言和工具以及属性接口和定制工具。

为了解决这样那样的一些问题，许多软件工程师开始采用基于组件的方法进行系统开发。一个软件组件就是一个可重用代码的二进制单元。

ESRI 选择 COM 作为 ArcGIS 的组件技术的原因是 COM 是一项成熟的技术，能提供很好的性能，目前有很多开发工具支持，而且有很多第三方组件可用于扩展 ArcObjects 的功能。

已经出现了几个不同但相互重叠的开发和共享组件标准。微软的 COM 已成为建立交互式桌面应用程序事实上的标准。在互联网上，JavaBean 是可行的技术。对象管理组（OMG）制定了粗粒度层次上适用于应用程序互操作的公用对象请求代理架构（CORBA）。

为了理解 COM（以及所有基于 COM 的技术），认识到其不是一种面向对象语言而是一个协议或标准是非常重要的。COM 不仅仅是一种技术，还是软件开发的一种方法论。COM 定义了一种连接软件组件或模型的协议。通过使用这种协议，可以建立能在分布式系统中动态交互的可重用软件组件。

组件成功的关键是它们以一种可行的方式实现了目前软件工程中普遍接受的许多面向对象原理。组件方便了软件的重用，因为它们是能很容易组配成更大系统的自包含构建块。

COM 还定义了一种称作基于接口的编程模式。对象封装了能体现定义良好的接口后每个实例化对象特征的操作方法和数据。这增强了系统开发的结构化和安全性，因为一个对象的客户并不知道某个特定方法如何实现的任何细节。COM 不规定应如何构建一个应用程序。语言、结构和实现细节都由使用 COM 的应用程序开发人员决定。

COM 的确规定了一个对象模型和使 COM 对象能与其他 COM 对象交互的编程需求。这些对象可以位于单个过程中，在不同的过程中，或者甚至在远程机器上。它们可以用其他语言编写，也可以用完全不同的方式开发。这就是 COM 被认为是二进制规范或标准的原因—它是一个在程序被编译为二进制机器代码后才应用的标准。

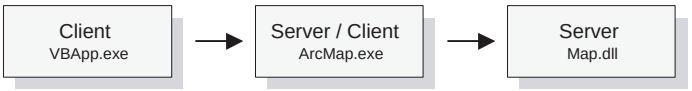
COM 允许对象在二进制水平上重用，意味着第三方开发者即使从最底层扩展系统也不需要访问这些对象的源代码、头文件或对象库。

组件、对象、客户和服务器

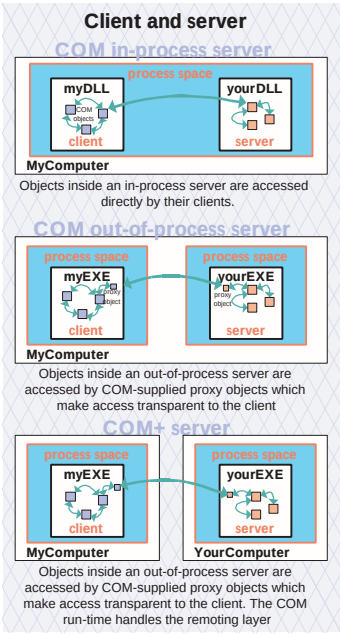
不同的文件中组件（components）、对象（objects）、客户（clients）和服务器（servers）表示不同的意思（各种文件用这些术语指代相同的事物，使得术语使用更加混乱）。因此，定义这本书所要使用的术语是非常有必要的。

COM 是一种客户/服务器（C/S）架构。服务器（或对象）提供某种功能，而客户使用这种功能。COM 使客户和对象之间的交流变得简单。一个对象可以同时是一个客户的服务器和其他对象服务的客户。

对象是为客户提供服务的 COM 类的实例。因此一般说客户端和对象而不是客户和服务端。这些对象经常指 COM 对象和组件对象。本书将把它们简单地指为对象。



客户及其服务器可以存在于同一个进程中或不同的进程中。入进程服务被打包成动态链接库（DLL）格式，当客户第一次访问这些服务时会加载这些 DLL 到客户的地址空间中。出进程服务被编译为可执行文件并在它们自己的地址空间中运行。COM 这种差异对客户透明。



在创建 COM 对象时，开发者必须了解对象驻留在什么类型的服务器中，但如果对象的创建者正确地实现了它们，打包并不会影响客户对对象的使用。

这两种对称相反的打包方法各有优缺点。DLLs 能更快速的装载到内存中，且调用一个 DLL 函数速度较快。另一方面，EXEs（可执行文件）能够提供更健壮的解决方案（如果服务器失败，客户端不会崩溃），同时安全性能更好，因为服务器有其自己的安全保障。

在一个分布式系统中，EXEs 更灵活，即使服务器与客户端有不同的字节顺序也不会有什么问题。大多数 ArcObjects 服务被打包为入进程服务（DLLs）。稍后，用户将看到使用入进程服务的性能优势。

在一个 COM 系统中，客户端或功能的使用者与功能的提供者，对象完全相分离。客户端需要知道所有知识就是可用的功能；有了这些知识，客户端就可以调用对象中的方法并期望对象作出响应。这样，COM 可以认为是客户端和服务器之间的一种协议。如果对象违反了这种协议，那么系统的行为就不能预知。这样，COM 开发是基于功能实现者和功能使用者间相互信任基础上的。

在 ArcGIS 应用程序中，许多对象通过其接口提供了几千个属性和方法。当使用 ESRI 对象库时，可以假定所有这些属性和方法是完全实现了的，如果它们列在对象图中，就可以使用它们。

类工厂（CLASS FACTORY）

在每个服务器中都有一个称为“类工厂”的对象，COM 在运行时与它交互作用以便实例化某个特定类的对象。每个 COM 类都有一个相应的类工厂。通常，当客户端向服务器请求一个对象时，相应的类工厂就会创建一个新对象并将该对象传给客户端。

独生对象（SINGLETON OBJECTS）

虽然这是通常的实现方法，但不是仅有的实现方法。类工厂也可以在第一次创建一个对象实例，在随后的调用中将同一个对象传给客户端。这类实现创建了所谓的“独生对象”，因为每个进程只有一个对象实例。

全局唯一标识符（GLOBALLY UNIQUE IDENTIERS）

一个分布式系统可能具有成千上万个接口、类和服务，在运行时定位和绑定客户端和对象都需要引用它们。毫无疑问，使用人们易读名称可能会导致潜在的冲突，因此 COM 使用全局唯一标识符（GUIDs），它是一个 128 位的数字，实际上能保证其在世界上的唯一性。在公元 5770 年以前可以每秒产生 1000 万个 GUIDs。

COM API 定义了用于产生 GUIDs 的函数；此外，所有适用于 COM 开发的工具都会自动分配适当的 GUIDs。与通用唯一标识符（UUIDs）一样，GUIDs 也由开放集群分布式计算环境（DCE）规范定义。下面就是一个注册表格式的 GUID 例子。

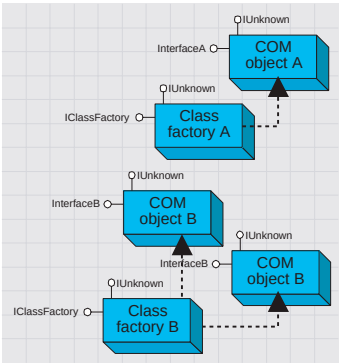
{E6BDAA76-4D35-11D0-98BE-00805F7CED21}

COM 类和接口（COM CLASSES AND INTERFACES）

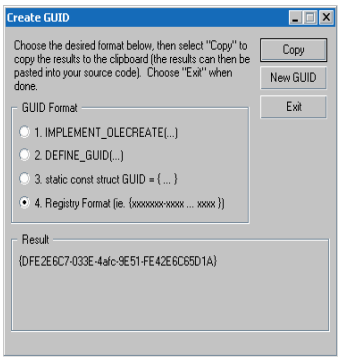
使用 COM 开发就意味着用接口开发，即所谓的基于接口的编程模式。对象间的所有通信都是通过其接口实现的。COM 接口是抽象的，意味着没有与接口相关联的实现；与接口关联的代码都来自类的实现。接口规定了对选择用于实现接口的对象可以作什么请求。

接口的实现方法随对象不同而不同。这样对象继承了接口类型而不是接口的实现，这称为类型继承。功能用接口来抽象模拟并在类的实现中实现。类和接口通常被称为 COM 的“什么”和“怎么”。接口定义对象能做什么，而类定义怎么做。

COM 类提供了一个或多个接口的代码，从而在类中完全封装了其功能。

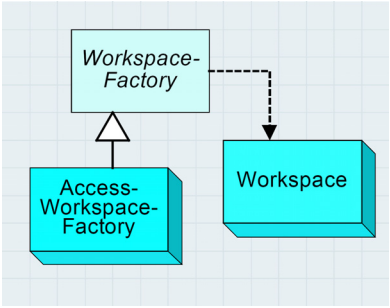


服务是一个二进制文件，包含一个或多个 COM 类需要的所有代码。这些代码既包括用 COM 将对象实例化到内存的代码，也包括执行服务中对象支持的各种方法的代码。



GUIDEN.EXE 是微软 Visual Studio 随带的一个实用工具，并提供生成 GUID 的一个易用用户接口。该工具可以在 VS 安装目录下 \Common\tools 文件夹中找到。

缩写的 GUID 通常读作“gwid”。



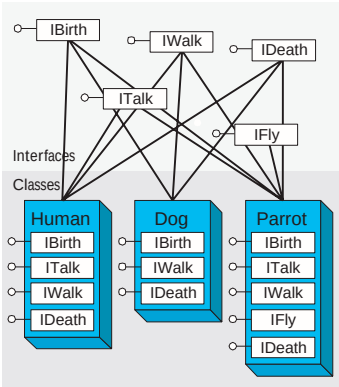
这是 Geodatabase 对象模型的一个简化图，展示了抽象类和组件对象类之间的类型继承及类的实例化。

两个类可以具有相同的接口，但它们实现这些接口的方法可能完全不同。通过这种方式实现接口，COM 展示了经典的面向对象的多态行为。COM 不支持多重继承概念，但由于单个类能实现多个接口，所以这不是一个缺陷。关于多态行为可以看左边的模型图。

ArcObjects 中开发者必须理解的类有三种类型：抽象类（abstract Classes）、组件对象类（coclasses）和类（classes）。抽象类不能创建对象，只是对其子类的实例化进行了规定（通过类型继承）。ArcObjects 的 Dataset 或 Geometry 类就是抽象类的例子。不能创建 Geometry 类型的对象，但可以创建 Polyline 类型的对象。Polyline 对象实现了在 Geometry 基类中定义的接口，因此在基于对象类中定义的任何接口都可以从组件对象类访问。

组件对象类是公开可创建类。换言之，COM 可以创建组件对象类的实例并把实例对象提供给客户端，从而使客户端能使用这个类接口定义的服务。类不能公开创建，但类的对象可以用 ArcObjects 中其它对象来创建并提供给客户端使用。

左边的模型图解释了在实现接口时 COM 类中展现的多态行为。注意 Human 和 Parrot 类都实现了 ITalk 接口。ITalk 接口定义了一些方法和属性，如 StartTalking、StopTalking、Language 等，但显然这两个类的实现是不同的。



这个模型图展示了表达为接口的公共行为如何在多个对象（本例中是动物）间共享，以支持多态。

内部接口（INSIDE INTERFACES）

COM 接口规定 COM 对象之间如何相互通信。在操作 COM 对象时，开发者从不直接操作 COM 对象，而是通过对象的一个接口来访问对象。COM 接口被设计为逻辑相关的一组功能。实际功能由客户端调用并由服务器实现；这样，对象接口就成为客户端和服务器之间的协议。对象的客户端拥有指向该对象的接口指针。这个指针被称为一个不透明的指针，因为客户端不能获得对象内的任何实现细节或直接访问对象的状态数据。客户端必须通过接口的成员函数来通信。这就允许 COM 提供二进制标准，通过这个标准所有的对象都能有效地通信。

接口允许开发者抽象的定义功能。VC++开发者把接口看作纯虚函数的集合，而 VB 开发者将接口看作属性、函数和子过程的集合。

接口概念是 COM 的基本概念。在讨论 COM 接口时，COM 规范（微软，1995）着重强调以下四点：

1. 接口不是类。接口不能实例化自身，因为其上没有实现。
2. 接口不是对象。接口是一个相关功能组，是客户和对象之间通信的二进制标准。

3. 接口具有很强的类型性。每个接口都有自己的接口标识符，因此消除了具有人们易读相同名字接口间可能的冲突。
4. 接口是不变的。接口从来没有版本。一旦接口被定义和公布就不能改变。

接口的永久性不是简单的限于其方法署名，而是也扩展到其语义行为。例如，某个接口定义了两个方法 A 和 B，没有限制它们的使用方法。如果后来的版本中方法 A 要求 B 先被执行，则违反了 COM 协议。这种变化可能会迫使客户端重新编译。

一旦接口被公布，接口的外部标记就不可以改变。可以在任何时候改变提供接口的对象的实现细节。这种改变可能是细微错误的修改或底层算法的完全重写；接口的客户端不必担心，因为接口对于他们来说似乎没变。这就意味着当对服务器的升级以新的 DLLs 和 EXEs 格式部署时，现有客户端不需要重新编译以使用新的功能。如果接口的外部标识不能再容纳新功能，那么可以创建一个新的接口来提供新功能。旧的或没有什么作用的接口不会从类中删除，以保证已有客户端应用程序能继续与新升级后的服务器通信。新的客户可以选择使用新的或旧的接口。

IUNKNOW 接口

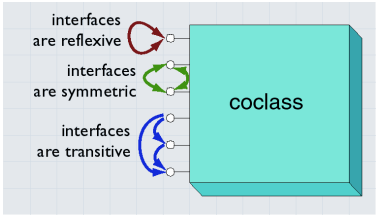
所有 COM 接口都源于 IUnkonow 接口，而且所有 COM 对象都必须实现这个接口。IUnknow 接口执行两个任务：控制对象的生命周期和提供运行时类型支持。客户端在使用对象时通过 IUnknow 接口维护对该对象的引用，而把实际生命周期管理留给对象自身。

对象的生命周期用两个方法 AddRef 与 Release 以及一个内部引用计数来控制。每个对象都必须实现 IUnknow 接口以便控制其自身的生命周期。当创建或复制接口指针时，就会调用 AddRef 方法；当客户端不再需要这个指针时，就会调用相应的 Release 方法。当引用计数为 0 时，对象自动销毁。

IUnknown 名称来自 1988 微软内部称为对象架构一文：处理动态可扩展类库中的 Unknow 或类型安全 (Dealing with the Unknow -or -Type Safety in a Dynamically Extensible Class Library)。

客户也可以使用 IUnknow 来获得对象的其他接口。QueryInterface 方法是当对象其他接口被请求时的一种客户调用。当一个客户调用 QueryInterface 时，对象提供一个接口并调用 AddRef。实际上，返回接口的任何 COM 方法有责任为调用者增加对象引用计数。当接口不再需要时，客户端必须调用 Release 方法。只有在复制接口时客户端才显式调用 AddRef 方法。

开发 COM 对象时，开发者必须遵循 QueryInterface 规则。这些规则规定对象接口是对称的、可传递的和自反的，并且在对象生命周期中一直可用。对客户而言，这意味着根据一个有效的对象接口，通过调用 QueryInterface 来请求对象的其他接口（包括该接口自身）总是有效的。不能因为时间或安全限制而先支持一个接口，后来又拒绝访问该接口。其他机制必须用于提供这个层次的功能。有些类支持可选接口



QueryInterface 规则规定对象接口是自反的、对称的和可传递的。控制对象上的一个有效接口指针就总有可能获取该对象上的其它接口。

QueryInterface 常简写为 QI。

由于 IUnknow 是所有 COM 对象的基础，一般在 AO 文档和类模型图中都不引用该接口。

聪明指针是基于类的聪明类型，本章后面将详细讨论。

MIDL 通常简化地称为 IDL。

IDL 定义开发者在操作 ArcObjects 时使用的公共接口，在编译时，IDL 会创建一个类型库。

概念。取决于组件对象类，可以选择性的实现一个接口；这不会违反这个规则，因为这个接口在类上要么总是可用，要么总是不可用。

在请求一个特定接口时，QueryInterface 方法返回一个已经为该请求接口分配好的内存块，或分配并返回一个新的内存块。当 IUnknow 接口被请求时才会出现必须还回相同内存块的情况。当比较两个接口指针以查看它们是否指向同一个对象时，很重要的是不能执行简单比较。为正确比较两个接口指针以查看它们是否指向同一个对象，都必须请求它们的 IUnknow，而比较必须在 Iunknow 指针上进行。这样，IUnknow 接口被认为可以定义 COM 对象的标识符。

在 VB 中通过将接口赋值为 Nothing 调用 release 来释放其控制的任何资源是好的习惯。即使没有调用 release，VB 也会在不再需要该对象时，即跳出其运行范围时自动调用 release。对于全局变量，必须显式调用 release。在 VB 中，系统会为用户执行所有这些引用计数操作，使 COM 对象的使用变得相对简单。

然而，在 C ++中，必须增加和减少引用计数以便使对象正确控制自己的生命周期。同样，在请求其他接口时必须使用 QueryInterface 方法。C++中聪明（smart）指针的使用可以简化这些操作。这些聪明指针是基于类的，因此具有适当的构造器、销毁器和过载操作符以自动化多数引用计数和查询接口操作。

接口定义语言（INTERFACE DEFINITION LANGUAGE）

微软接口定义语言（MIDL）可用于描述包括其接口的 COM 对象。MIDL 是分布式计算环境（DCE）定义的接口定义语言（IDL）的一个扩展，其中 IDL 用于定义客户端和服务器间的远程调用。MIDL 扩展包括大部分对象定义语言（ODL）声明和属性。ODL 早期用于创建 OLE 自动化类型库。

类型库（TYPE LIBRARY）

类型库最好被看作 IDL 文件的二进制版本。它包含了所有组件对象类、接口、方法和一个或多个服务中包含的类型的二进制描述。

微软提供了几个能操作类型库的 COM 接口。ITypeInfo 和 ITypelib 就是其中两个接口。通过使用这些标准的 COM 接口，各种开发工具和编译器可以获得特定类型库支持的组件对象类和接口方面的信息。

为了支持独立于语言的组件开发思想，涉及 ArcObjects 库的所有相关数据都被加载进类型库中。类型库中没有头文件，源文件或是外部

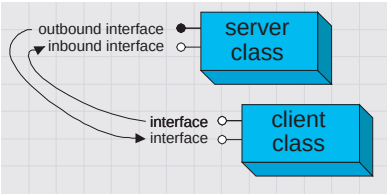
开发者提供或需要的对象文件。

入接口（INBOUD） 和出接口（OUTBOUND）

接口可以是入接口或出接口。入接口是最常见的接口类型，客户端调用包含在对象上的入接口的功能。出接口则是对象调用客户的接口，这是一种与传统的回调机制类似的技术。

这两种接口的实现方式是不同的。入接口的实现程序必须实现该接口的所有功能，否则就违反了 COM 协议。出接口也一样。如果使用 VB，不必实现接口上的所有功能，因为 VB 为那些没有实现的方法提供了存根（stub）方法。另一方面，如果使用 C++，则必须实现所有的纯虚函数以编译类。

连接点是操作 COM 出接口时的一个特殊方法。连接点架构定义了对象间的通信是怎样建立和撤消的。连接点不是初始化双向对象通信的最有效方法，但是一种常用方法，因为许多开发工具和环境支持它们。



在本书中的模型图和 AO 对象模型图中，出接口用实心圈表示。

Dispatch 事件接口（Dispatch event interface）

ArcObjects 中有些对象支持两个出接口事件接口，看起来类似于它们支持的方法。IDocumentEvents 和 IDocumentEventsDisp 就是这样两个接口的例子。后缀“Disp”指示是个纯 Dispatch 接口。VBA 在处理某些应用程序事件时，如载入文档等，会使用这些 Dispatch 接口。VBA 程序员使用 Dispatch 接口，而使用其他开发语言的程序员使用非纯粹的 Dispatch 接口。因为这些 Dispatch 事件接口是针对特定应用程序的，所以细节问题将在本书应用程序章节中讨论，而不在框架章节中讨论。

缺省接口（Default interfaces）

每个 COM 对象都有一个缺省接口，当创建对象时如果没有指定其他接口就会返回缺省接口。除了少数例外，ESRI 对象库中的所有对象都使用 IUnknow 作为其缺省接口。

Iunknown 之所以作为缺省接口的原因是因为 VB 对象浏览器隐藏了缺省接口信息，其隐藏 Iunknown 的事实对 VB 开发者并不重要。

ArcCatalog 和 ArcMap 的 Application 对象的缺省接口都是 IApplication 接口。使用非 IUnknow 的缺省接口是 VBA 的要求，并且在 ArcMap 和 ArcCatalog 的应用程序层次的对象上才可以发现。

这意味着保持接口指针的变量必须以某种方式声明。要了解更多有关这方面的细节，请参考本章后面的代码部分。当创建 COM 对象时，可以请求对象支持的任何接口。

IDispatch 接口

绑定这个术语指给定一个对象指针匹配函数位置的进程。

COM 支持 3 类绑定：

1. 延迟绑定（Late）。这种绑定中直到运行时才发现类型。客户端调用但被对象没有实现的方法在执行时会失败。
2. ID 绑定。方法的 IDs 在编译时被保存，但这个方法的执行仍通过更高级的功能来执行。
3. 自定义 vTable（早期）绑定。在编译时进行绑定。客户可以直接调用对象的方法。

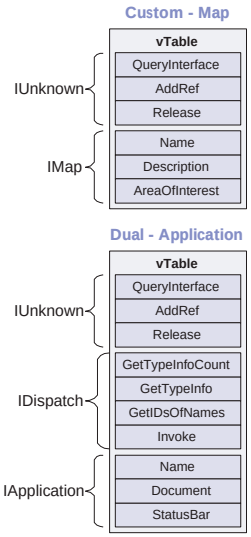
绑定类型	入进程 DLL	出进程 DLL
延迟绑定	**22,250**	**5,000**
自定义 vTable 绑定	**825,000**	**20,000**

这个表显示了在典型的 Pentium III 机器上每秒可以调用的函数数量。

IDispatch 接口支持延迟（late－）和 ID－绑定语言。IDispatch 接口具有允许客户询问对象支持什么方法的方法。

假定支持所需的方法，客户端通过调用 IDispatch::Invoke 方法来执行该方法。调用完这个方法反过来又调用所需方法并将状态和任何参数返回给客户端。

很明显，这不是调用 COM 对象最有效的方式。延迟绑定请求一个对象调用以获得方法 IDs 的列表；客户端必须构建 Invoke 方法调用并调用它。Invoke 方法必须取得参数并调用功能。



这些模型图总结了 AO 中两个类的自定义和 Idispatch 接口。vTable 的布局展示了差异。也说明了实现这些方法的重要性。如果丢失了一个方法，vTable 的布局将出错，因此错误的函数指针将被返回给客户端，导致系统崩溃。

所有这些步骤显著增加了执行该方法所消耗的时间开销。此外，每个对象都必须具有 Idispatch 的一个实现，使所有对象更大而且增加了其开发时间。

ID 绑定比延迟绑定稍微有所改进，方法 IDs 在编译时被调入高速缓存，意味着不需要获取 IDs 的初始调用。然而，仍有很显著的调用开销，因为为了执行对象上所需的方法，仍调用了 IDispatch::Invoke 方法。

早期绑定，通常也称为自定义 vTabel 绑定，不使用 IDispstch 接口。相反地，一个类型库提供编译时需要的信息以允许客户端了解服务器的布局。在运行时，客户端直接调用对象的方法。这是调用对象方法的最快方法，也对编译时的类型检查有益。

支持 IDispatch 和自定义 vTable 的对象被称为双重接口对象。ESRI 对象库中的对象类没有实现 IDsipatch 接口；意味着这些对象库不能使用延迟绑定脚本语言，如 JavaScript 和 VBScript，因为这些语言要求 COM 服务器能支持 IDispatch 接口。

仔细观察 ArcGIS 类模型图，可以看到 Application 对象支持 IDispatch 接口，因为 VBA 要求 IDispatch 接口。

所有 ActiveX 控件都支持 IDispatch。这意味着可以使用 ArcObjects 中的各种 ActiveX 控件从脚本环境中访问其功能。

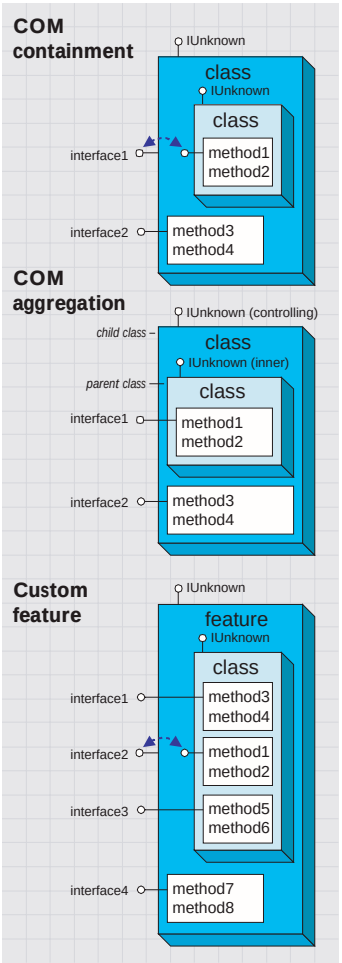
接口继承（INTERFACE INHERITANCEN）

直接从非 IUnknown 接口继承的接口在 VB 中不能实现。

接口由一组方法和属性组成。如果一个接口继承自另一个接口，则父接口中的所有方法和属性在子对象中可以直接获得。

这里最基本的原则是接口继承，而不是可以在诸如 SmallTalk 和 C++ 语言中看到的实现继承。在实现继承中，对象实际上是从父对象那里继承了代码。在接口继承中，传递的是对象方法的定义。实现了接口的组件对象类必须提供所有继承接口的实现。

异构开发环境中不支持实现继承，因为这需要访问源文件和头文件。为重用代码，COM 使用集成（aggregation）和包含（containment）原理。两者都是二进制重用技术。



聚合和包容（AGGREGATION 和 CONTAINMENT）

第三方开发者通过聚合或包容来使用现有的对象，唯一的一个要求就是驻留待聚合或包容对象的服务要安装在开发者和终端用户机器上。并不是所有开发语言都支持集成。

最简单的二进制重用形式是包容。包容允许修改原始对象的方法行为，但是不能改变其签名。通过包容，被包容对象（内部的）并不知道自己被包容在其他对象（外部的）中。外部对象必须实现内部对象支持的所有接口。当请求这些接口时，外部对象简单地将请求委托给内部对象。为支持新功能，外部对象可以在没有传递调用的情况下实现一个接口，或者实现除内部对象接口外的一个全新接口。

COM 聚合涉及一个外部对象，这个外部对象控制从内部对象中选择提供哪些接口。聚合方法不能修改原始对象的方法行为。内部对象知道自己被聚合到另一个对象中，并促进 QueryInterface 调用外部（控制）对象，以便对象作为一个整体服从 COM 法则。

对使用聚合的对象客户来说，没有方法区分外部对象实现了哪些接口，内部对象实现了哪些接口。

自定义要素同时使用聚合和包容。开发者聚合那些不需要定制的接口并包容那些需要定制的接口。被包容接口上的单个方法可以在定制类中实现，这样可以提供自定义功能，或者这个方法的调用可以传递给被包容接口的合适方法。

聚合在这种情况下非常重要，因为要素中定义了一些隐藏接口，它们不能被包容。要了解有关更多自定义要素方面的信息，请参阅第二卷第 8 章“访问地理数据库”。

VB6 不支持聚合，所以不能用于创建自定义要素。

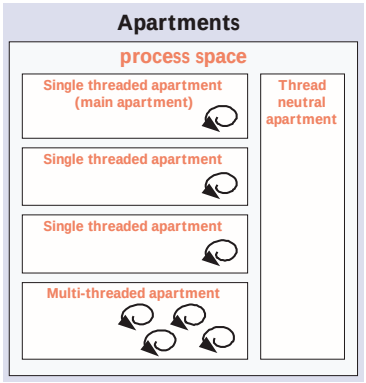
线程、单元和调度（Threads Apartments and Marshaling）

虽然对单元和线程的理解不是使用 ArcObjects 所必须的，但基础知识将有助于理解本章后面重点介绍的某种开发环境中的一些暗示。

线程是贯穿应用程序的进程流。Windows 应用程序中有许多潜在的线程。单元是在进程中操作语境的线程组。对于 COM+, 语境属于一个单元。语境有许多潜在的类型；安全就是一个语境类型的一个例子。在对象间成功地相互通信前，对象必须有兼容的语境。

COM 支持两类单元：单线程单元和多线程单元（MTA）。COM+还支持线程中立单元（TNA）。一个进程可以有很多的 STA；每个进程创建一个 STA 来调用主单元。作为单元创建的线程放置在一个 STA 中。所有的用户接口代码都被放在一个 STA 中，以防止死锁状态。一个进程只能有一个 MTA。作为多线程启动的一个线程被放置在 MTA 中。TNA 没有与其永久关联的线程；而是在适当的时候线程进入和离开单元。

入进程对象在注册表中有一个入口—ThreadingModal—来通知 COM 服务器控制管理器（SCM）将对象放置到哪个单元中。如果对象请求的单元和创建的单元兼容，那么对象就被放置到该单元中。否则，SCM 将另找一个或是新建一个合适的单元。如果没有定义线程模型，对象将被放置在进程的主单元中。ThreadingModel 注册表入口可能有以下几个值：



将 SCM（发音为 scum）想象为 COM 运行时环境。SCM 与对象、服务器和操作系统交互并在客户与其操作的对象间提供透明。

1. Apartment. 对象必须在 STA 中执行。UI 对象一般使用这个值。
2. Free. 对象必须在 MTA 中执行。创建线程的对象一般被放在 MTA 中。
3. Both. 对象与所有的单元类型兼容。对象将在与创建者相同的单元中创建。
4. Neutral. 对象必须在 TNA 中执行。对象使用这个值来确保当从其他单元调用时线程没有切换。Neutral 值只在 COM+下可用。

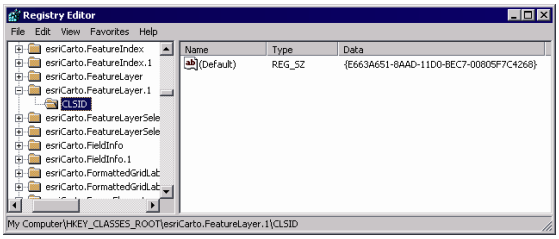
调度使客户端可以透明地调用其他单元中对象的接口功能。调度可以发生在不同机器上的 COM 单元之间、不同进程空间的 COM 单元之间及同一进程空间（如 STA 和 MTA）的 COM 单元之间。COM 提供了一个标准的调度器，来处理使用自适应数据类型的函数调用（见下表）。只要生成了代理存根代码，非自适应数据类型也可以由标准调度器处理；否则，就需要编写自定义调度代码。

类型	描述
(Boolean	具有 True 或 False 值的数据项
unsigned char	8 位无符号数据项
double	64 位 IEEE 浮点数
float	32 位 IEEE 浮点数
int	有符号整数，长度取决于系统
long	32 位有符号整数
short	16 位有符号整数
BSTR	以长度为前缀的字符
CURRENCY	8 位，小数点固定的数值
DATE	64 位，自 1899 年 12 月 30 日以来的浮点分数日期
SCODE	16 位系统中，对应于 VT_ERROR 的内置错误
Typedef enum myenum	有符号整型，长度取决于系统
Interface IDispatch	指向 IDispatch 接口的指针
Interface IUnkown	指向非源自 IDispatch 的接口的指针
Dispinterface Typename	指向源自 IDispatch 的接口的指针
Coclass Typename	指向组件对象类名的指针(VT_UNKNOWN)
[oleautomation]interface Typename	指向源自 IDispatch 的接口的指针
SAFEARRARY(TypeName)	TypeName 是上述类型中的任何一种。这些类型的数组
TypeName	TypeName 是上述类型中的任何一种。指向一种类型的指针
Decimal	以 10 的变量次幂为比例因子的 96 位无符号二进制整数。一种提供了大小和比例因子的十进制数据类型

组件类目

客户应用程序用组件类目来有效查找安装在系统中某个特定类型的所有 COM 类。例如，一个客户应用程序可能支持一个能指定输出格式的数据输出函数，在这种情况下，组件类目可被用于为各种数据格式查找数据输出类。如果不使用组件类目，应用程序必须实例化每个对象，并询问该对象以查看它是否支持所需的功能，这不是一个可行的方法。通过允许客户应用程序的开发者创建和操作属于特定类目的类，组件类目支持 COM 的扩展。如果后来向组件类目中添加了一个新的类，客户应用程序不需更改就可以使用这个新类；在下次读取组件类目时会自动加载该新类。

COM 与注册表



Windows 系统注册表中的 ESRI 键

COM 使用 Windows 系统注册表来存储组成 COM 系统的各个部分的信息。类、接口、DLLs、EXEs、类型库等都分配了唯一的识别符（GUID），引用这些组件时 SCM 使用它们。运行 regedit，然后打开 HKEY_CLASSES_ROOT 就可以查看这样的一个例子。这样打开了在系统中注册的所有类的列表。

COM 使用注册表来完成许多日常管理任务，但最重要和最容易理解的是

将 COM 对象实例化到内存中时注册表的使用。最简单的情况就是入进程服务，其步骤如下：

1. 客户端请求 COM 对象服务。
2. SCM 通过查询类 ID（一个 GUID）寻找被请求对象的注册表入口。
3. 定位 DLL 并将其导入内存。SCM 调用 DLL 中称为 DllGetClassObjects 的函数，将想要得到的类作为第一个参数传递。
4. 类对象正常实现 IClassFactory 接口。SCM 调用该接口上的 CreateInstance 方法以将适合的对象实例化到内存中。
5. 最后，SCM 为客户端请求的接口请求新创建的对象并将该接口回传给客户端。此时 SCM 退出，客户端和对象直接进行通信。

函数 DLLGetClassObjects 是使 DLL 成为 COM DLL 的一个函数。其他的函数，如 DLLRegisterServer 和 DllUnregisterServer 等，有的话当然很好，但不是使 DLL 成为 COM DLL 所必须的。

从上面的步骤序列中很容易想到对象打包方式的变化（DLL 对 EXE）对对象的客户产生的区别是如何的细微。COM 会处理这些区别。

自动化

自动化是单个对象或整个应用程序通过延迟绑定语言来访问其封装功能的技术。通常，自动化被看作写入宏，这些宏为了完成任务可以访问许多应用程序。如前所述，ArcObjects 不支持 IDispatch 接口；因此，自动化控制器不能单独使用。

通过创建文档对象并通过该文档对象或与其连接的对象之一调用 ArcMap 的方法来实例化一个 ArcMap 例程是可能的。但是，这种方法存在许多问题，因为自动化控制器例程和 ArcMap 例程运行在不同的进程中。包含在 ArcObjects 中的许多对象是依赖于进程的，因此简单的自动化不起作用。

ArcGIS应用程序是用ArcObjects构建的，并且可以通过许多APIs来开发。这些APIs包括COM (VB, VC++, Delphi, MainWin)，.NET (VB .NET和C#)，Java和 C++。有些APIs比其它API更合适于某些特定应用程序的开发。后面会简要地讨论这方面的内容，但也应该阅读适合所使用的产品的开发指南，以获得关于使用何种API的更多信息和建议。

本章后面几节讲述不考虑API而用ArcObjects进行开发时的一般性指导方针和要考虑的问题。每种比较通用的API语言都有一节专门描述开发环境、编程技巧、资源和其它用ArcObjects进行开发时必须考虑的问题。

编码标准

为了简单起见，有些示例没有遵循编码标准。例如，建议在用Visual Basic编码时，在ESRI对象库中定义的所有类型都要以库名为前缀，如esriGeometry.IPolyline。只有在名称会冲突的情况下才会这样做。忽略这部分内容使得ArcObjects的新开发人员更容易理解代码。

与特定语言相关的每节都以这种语言的编码标准部分开始。这些标准在ESRI内部使用，且软件随带的示例也遵循这些标准。

要理解标准和指导方针之所以重要的原因，设想在大型软件开发工程中有许多由团队成员代表的背景。每个程序员对代码风格和编写方法都有自己的主张。如果每个程序员都以不同的方式设计代码，会极大地增加共享工作和想法的难度。在一个成功的团队中，每个开发人员都采用团体共同建立的编码方式。这往往意味着调整自己的代码以匹配系统中现有代码的风格。

这样做最初这看起来似乎是负担，但采用一种统一的编程风格和一套不变的技术可以提高了软件的质量。当一个工程中所有的代码遵从一套标准的格式和约定时，浪费在学习单个程序员独特的句法遁词的时间大大减少了，可以将更多的时间花费在校对、调试和扩充代码上。甚至在社会层次上，统一风格鼓励了团队精神而不是个人主义，其前景将是更大的团队统一性、生产力，最终生产更好的软件。

一般编码技巧和资源

本节关于一般编码技巧的内容对所有使用ArcObjects的开发人员都是有益的，不管他们使用什么语言。不过，示例代码是用VBA编写。

类图

获取对象模型的帮助是成功使用ArcObjects的基础。附录“理解对象模型图”通过对象介绍了类图并展示了许多常用的方法。早期学习阶段查看打印的类图最有用。通过模型图，开发人员可以理解ArcObjects实现的对象模型的总体结构。当能够轻松地理解总体结构时，就可以更有效地使用随软件一块发布的PDF文件。这些PDF文件是可搜索的；可在Adobe® Acrobat® Reader®中用Search对话框快速找到类和接口。

对象浏览器

除了类图的PDF文件外，许多依赖于开发平台的对象浏览器可以用来查看类型库信息。

Visual Basic和.NET都有内置的对象浏览器；OLEView（Microsoft的免费应用程序）也可以显示类型库信息。在这种环境中最好的对象浏览器是ESRI的对象浏览器。使用对象浏览器可以查看所引用的任何类型库的类型信息。类和接口的信息可以在Visual Basic、Visual C++或对象图格式中显示。对象浏览器可以用来查看组件对象类和类，但是不能用来查看抽象类。抽象类只有在对象图中才是可见的，在这里使用对象图只是为了简化模型。

Java和C++开发人员应参考ArcObjects JavaDoc或者ArcGIS Developer Help。

组件帮助

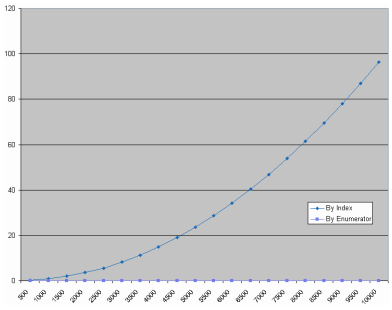
所有接口和组件对象类都记录在组件帮助文件中。当更好地掌握了对象模型时，最终这些帮助是最经常要用到的。

对于Visual Basic和.NET的开发人员而言，这是一个编译过的HTML文件，可以在这些开发环境中直接查看或使用IDE查看。当按下F1键时，如果光标停在一个ESRI类型上，ArcGIS开发帮助系统的ArcObjects Class Help中适当的页面就会显示在编译过的HTML浏览器中。

Java和 C++的开发人员可以参考ArcObjects JavaDoc或ArcGIS Developer Help。

代码向导

在Visual Basic、 Visual C++和.NET中都有大量的有助于创建样板代码的代码生成向导（Code Generation Wizards）。虽然这些向导有助于省去一些沉闷的常规任务，但作为开发人员不能因此而不去理解潜在于这些代码后面的原理。主要目的是通过阅读这些附带的文档并理解这些工具的局限性。



这个曲线图显示了使用一个与元素索引相反的计数器访问集合的性能优势。正如所期望的那样，该曲线为一个经典的指数趋势线(y=cxb)。

集合索引

ArcObjects中所有类似于集合的对象的索引都是从零开始的。并不是所有的开发环境都是如此；Visual Basic既有从零开始的索引，也有基于1的索引。作为通用规则，如果集合基数未知，则假定集合基数为零。这样保证了集合第一次被访问时会出现一个运行时错误（假定集合的访问不是从零开始的）。假定集合以1为基数，那么，以零为基数集合的第一个元素就会被错过，并且当代码被运行时，只有到达集合末尾时，才会出现一个错误。

访问集合元素

顺序访问一个集合元素时，最好使用一个计数器接口。这提供了遍历集合最快的方法。这样做的理由是每次按索引请求一个元素，使用内部计数器定位元素。因此，如果以循环的方式依次获取一个元素，花费的时间以指数增长($y=cx^b$)。

计数器的使用

当从一个对象请求一个计数器接口时，客户端不知道对象如何实现这个接口。对象可能会创建一个新的计数器，或者为了效率决定返回先前创建的一个计数器。如果先前的计数器传递至客户端，元素指针的位置会指向最后一个被访问的元素。为了保证计数器的指针指向集合的首个元素，客户端在使用前需要将计数器复位。

错误处理

异常处理是与特定语言相关的，而且因为COM是语言无关的，所以不支持异常。

接口的所有方法，也就是其它对象可以调用的方法，应该处理内部错误并通过合适的HRESULT指示成功或失败。COM不支持传递接口方法调用外的异常。COM支持COM异常的概念。COM异常使用COM错误对象，通过在COM错误对象中增加相关的信息并返回合适的HRESULT表示失败。客户端接收到HRESULT后就询问COM *Error*对象关于错误的上下文信息。像Visual Basic这样的语言实现了自身的异常处理模板。要了解更多信息，请参阅用于开发的合适API语言。

通知接口

ArcObjects中有许多没有方法的接口。这些接口被称作通知接口。这些接口的目的是通知应用程序框架实现这些接口的类支持特定的功能集。例如，应用程序框架使用这些接口确定一个菜单对象是一个根菜单（*IRootLevelMenu*）还是一个弹出式菜单（*IShortcutMenu*）。

客户端存储

一些ArcObjects方法期望接口指针在调用方法前指向有效的对象。这被称作客户端存储，因为客户端在调用方法前为对象分配所需的内存。假设有一个多边形并想得到它的边界矩形。使用*IPolygon*上的*QueryEnvelope*方法完成这一操作。如果编写下面的代码：

```
Dim pEnv As IEnvelope
pPolygon.QueryEnvelope pEnv
```

就会出错，因为*QueryEnvelope*方法期望客户端创建*Envelope*。这个方法会修改所遍历的外包矩形并返回一个修改过的外包矩形给用户。正确的代码如下：

```
Dim pEnv As IEnvelope
Set pEnv = New Envelope
pPolygon.QueryEnvelope pEnv
```

如何知道什么时候创建和什么时候不创建？一般来说，所有以“Query”开头的方法，例如*QueryEnvelope*期望客户端创建对象。如果方法名是*GetEnvelope*，那么就会为用户创建一个对象。使用客户端存储的原因就是性能。由于对象的方法在一个紧密的循环中被调用，参数只需要创建一次且易于增加。这比每次在方法内部创建一个新的对象要快得多。

属性的传值和引用

有时候，会看见设置为值或引用的属性，这意味着该属性既有*put_XXX*方法，也有*putref_XXX*方法。乍一看这可能有点奇怪—为什么一个属性会支持两种方法呢？Visual C++开发人员将这看作是，简单地给客户一个机会传递资源的所有权给服务器（使用*putref_XXX*方法）。Visual Basic则不同；当然，可能是因为Visual Basic开发人员在属性赋值上支持引用（*By Reference*）和传值（*By Value*）。

为了说明这点，假设一个窗体上有两个文本框，Text1和Text2。使用*propput*，在Visual Basic中可能会写成下面这样：

```
Text1.text = Text2.text
```

也可能写成：

```
Text1.text = Text2
```

或这样：

```
Text1 = Text2
```

所有这些例子都利用*propput*方法将文本框Text2的文本字符串赋给文本框Text1的文本字符串变量。第二和第三个例子中，因为没有专门的属性描述，Visual Basic寻找DISPID为0的属性。

DISPIDs是为了IDispatch接口有效使用Invoke方法调用适当方法而给属性和方法赋与的唯一IDs。

假定被操作的是文本框的文本字符属性，这些语句都是正确的。如果Text2变量引用的实际对象被赋给变量Text1，会发生什么？如果只有一个*propput*方法，这是不可能的；因此需要*propputref*方法。使用*propputref*方法，下面的代码获得对象引用的设置。

```
Set Text1 = Text2
```

注意“Set”的使用。

初始化出接口

初始化一个出接口时，如果变量没有监听来自于服务器对象的事件，只初始化变量是很重要的。不遵守这条规则会导致死循环。

举个例子，假定有一个变量*ViewEvents*标注如下：

```
Private WithEvents ViewEvents As Map
```

为了正确地接收这个事件句柄，可以在一个UI按钮的*OnClick*事件中编写如下代码：

```
Private Sub UIButtonControl1_Click()
   Dim pMxDoc As IMxDocument
   Set pMxDoc = ThisDocument

   ' Check to see that the map is different than what is currently connected
   If (Not ViewEvents Is pMxDoc.FocusMap) Then
      ' Sink the event since listener has not been initialized with this map
      Set ViewEvents = pMxDoc.FocusMap
   End If
End Sub
```

注意到上面的代码中，使用*Is*关键字检查对象的身份。

数据库考虑

依靠数据库编程时，必须遵循大量的规则以确保代码最优。下面是这些规则的详细内容。

如果打算通过编程的方式编辑数据，也就是说，不需要在ArcMap中使用编辑工具，需要遵循这些规则来确保自定义对象行为正确地被调用以响应应用程序对数据库所做的修改。这些自定义行为包括网络拓扑结构的维护或自定义要素定义方法的触发等。为了确保所做的修改在多用户编辑（long transaction）框架之内，也必须遵循这些规则。

编辑会话

地理数据库的所有修改应在一个编辑会话内完成，该会话用大括号将位于*Workspace*对象的*IWorkspaceEdit*接口上的*StartEditing*和*StopEditing*方法调用之间的代码括住。

任何多用户更新的数据库中都要求这种行为。开始一个编辑会话就将数据库的一个状态给应用程序，该数据库保证不会被修改，除非通过编辑程序进行修改。

另外，开始编辑会话将打开地理数据库中这样的行为，以便数据库的查询保证会返回一个内存中存在的对象引用如果，这个对象前面被查询过并且仍在使用中的话。

当操作一组相关对象而同时又修改对象时，也需要这种行为以使应用程序行正确。换言之，不在一个编辑会话内时，每次应用程序从数据库请求一个特定对象时，数据库都会创建一个新的COM对象实例。

编辑操作

将修改归组到编辑操作中去，该操作用大括号将位于*IWorkspaceEdit*接口上*StartEditOperation*和*StopEditOperation*方法调用之间的代

码括起来。

如果需要这样的话，所有的修改可以在一个编辑操作中完成。编辑操作可以撤消和重做。如果操作储存在ArcSDE中的数据，至少需要创建一个编辑操作。创建编辑操作不会有额外的开销。

循环和非循环光标

使用非循环查询光标选择对象或读取将被更新的对象。循环光标仅可用于只读操作，例如绘制和查询要素。

在一个编辑会话中，非循环光标只有被返回的对象在内存中不存在时，才会创建一个新对象。

采用查询过滤读取属性

只要读取的是对象的所有属性；查询过滤器应总使用“*”。为了有效地访问数据库，可以指定从数据库返回对象属性的数量。例如，绘制要素只需要要素的OID和*Shape*属性，因此较简单的绘制方式只从数据库获取这两列数据。这种优化加快了绘制的速度但不适合于编辑要素。

如果不读取所有的属性，那么被触发的特定对象的代码可能会找不到方法所要求的属性。例如，当要素的几何图形改变时，自定义要素开发人员更要编写代码来更新属性A和B。如果只恢复几何图形，那么在*OnChanged*方法上会发现A、B属性丢失。这会导致*OnChanged*方法返回一个错误，从而导致Store返回一个错误并使编辑操作失败。

标记修改过的对象

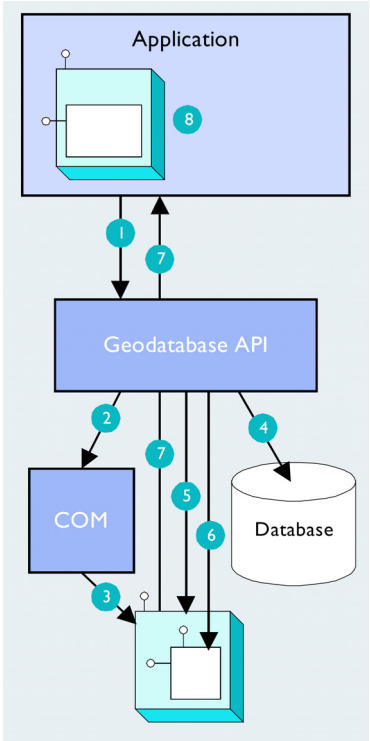
修改一个对象后，通过在对象上调用*Store*来标记对象为修改过的对象（并保证在数据库中更新）。在对象上调用*Delete*来方法删除该对象。设置这些调用的版本也存在，如果为了确保最佳的性能而在一组对象上执行某个操作时会用到。

调用这些方法保证了内置在地理数据库中的所有必要多态性对象行为被执行（如网络拓扑的更新，或响应ESRI提供的对象的其它列中的变化而更新特定列）。这也保证了开发人员提供的行为被正确地触发。

更新和插入光标

永远不要使用更新光标或插入光标来更新或将对象插入到一个已经被加载的地理数据库中具有活动行为的对象类和要素类中。

更新和插入光标是在开始数据库加载时使用的大光标APIs。如果在具有活动行为的对象类或要素类上使用，它们会回避任何与对象创建（例如拓扑创建）和属性或几何图象更新（如自动重新计算其它的相关字段）相关的所有特定对象行为。



上图清楚地显示了一个COM对象Feature有另外一个COM对象作为其几何图形。要素的Shape属性仅将指向这个几何图形对象的IGeometry接口指针向外传递给请求shape的调用者。这意味着如果不止一个客户请求该shape，所有客户都指向同一个几何图形对象。因此这个几何图形对象必须看作只读对象。不应该在从这个属性返回的几何图形上执行任何修改，甚至是临时修改。任何时候要修改要素的shape属性，必须修改ShapeCopy属性返回的几何图形，且更新后的几何图形应随后赋给Shape属性。

Shape 和 ShapeCopy几何图形属性

利用*Feature*对象的*Shape*和*ShapeCopy*属性来最优化地检索一个要素的几何图形。为了更好地理解与要素几何图形相关的这些属性，参考左图以理解来自于某个数据源的要素如何实例化到内存中以便在一个应用程序中使用。

要素按照下面的顺序从数据源中被实例化：

1. 应用程序通过调用合适的地理数据库API方法调用从一个数据源请求一个*Feature*对象。
2. 地理数据库请求COM创建一个期望的COM类（通常是*esriCore.Feature*）的vanilla COM对象。
3. COM创建*Feature* COM对象。
4. 地理数据库从数据源获得属性和几何图形数据。
5. vanilla *Feature*对象增加适当的属性值。
6. 创建*Geometry* COM对象，并在*Feature*对象中设置一个引用。
7. *Feature*对象被传递到应用程序上。
8. *Feature*对象在应用程序中一直存在直到不再需要。

使用类型库

由于ArcObjects中的对象没有实现*IDispatch*，因此为了使编译器早期绑定到正确的数据类型，有必要使用类型库。这适合于所有的开发环境，尽管在Visual Basic、Visual C++和.NET中都有帮助设置这些参考的向导。

ArcObjects需要的类型库位于ArcGIS的安装文件夹中。例如，COM类型库可以在COM文件夹中找到，而.NET Interop互操作套件在DotNet文件夹中。许多不同的文件都包含类型库信息，包括EXEs、DLLs、OLE自定义控件（OCXs）和对象库（OLBs）.

COM数据类型

COM对象通过它们的接口会话，因此使用的所有数据类型必须被IDL支持。IDL支持大量的数据类型；然而不是所有支持COM的语言都支持这些数据类型。因此，ArcObjects没有使用IDL中可用的所有数据类型，而是将大多数接口限制在Visual Basic支持的数据类型内。下表显示了IDL支持的数据类型及其在不同语言中的相应的类型。

开发语言	IDL	Microsoft C++	Visual Basic	Java
Base类型	boolean	unsigned char	unsupported	char
	byte	unsigned char	unsupported	char
	small	char	unsupported	char
	short	short	Integer	short
	long	long	Long	int
	hyper	__int64	unsupported	long
	float	float	Single	float
	double	double	Double	double
	char	unsigned char	unsupported	char
	wchar_t	wchar_t	Integer	short
	enum	enum	Enum	int
	Interface Pointer	Interface Pointer	Interface Ref	Interface Ref
Extended 类型	VARIANT	VARIANT	Variant	ms.com.Variant
	BSTR	BSTR	String	java.lang.String
	VARIANT_BOOL	short (-1/0)	Boolean	[true/false]

注意表底部的扩展数据类型：*VARIANT*、*BSTR*、和 *VARIANT_BOOL*。虽然可以用 *char* 和 *wchar_t* 这些数据类型传递字符串，但是有些语言像 Visual Basic 是不支持的。Visual Basic 使用 BSTRs 作为它的文本数据类型。BSTR 是一个以长度为前缀的宽字符串数组，数组的指针指向对象里的文本而不是长度前缀。Visual C++将 *False* 和 *True* 的 *VARIANT_BOOL* 值映射为 0 和-1。这和普通映射 0 和 1 是不同的。因此，在编写 C++代码时，一定要使用正确的宏—*VARIANT_FALSE* 和 *VARIANT_TRUE*，而不是 False 和 True。

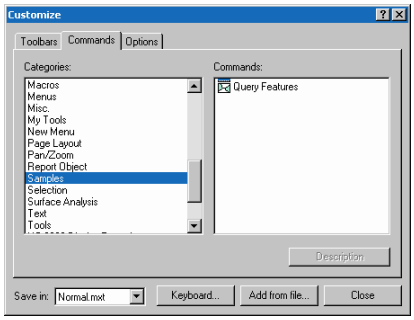
使用组件类目

组件类目在 ArcObjects 中被广泛使用，以便开发人员不需要对 ArcObjects 代码做任何修改就可以使用新功能。

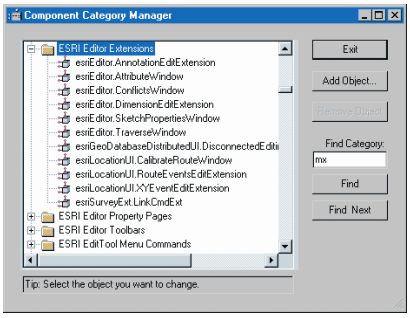
ArcObjects 以两种方式使用组件类目。第一种方式需要类任何时候都要在各自的组件类目内注册，例如 ESRI Mx Extensions. Classes，如果出现在组件类目中，就有一个实现了 *IExtension* 接口的对象，且当 ArcMap 应用程序启动时就被实例化。如果该类从组件类目中被移出，扩展模块不会加载，即使地图文档（MXD 文件）正在引用该扩展模块。

第二种用法出现在当应用程序框架使用组件类目来定位类并将这些类显示给用户以允许一些用户自定义操作时。与第一种方法不同，应用程序能记住（在一个地图文档中）正在被使用的对象并顺序从地图文档中加载它们。一个例子就是 ArcMap 中使用的命令。当自定义对话框显示给用户时，ArcMap 读取 ESRI Mx Commands 类目。这是对该类目的唯一一次读取。一旦用户选择了一个命令并将之加载到工具条中时，地图文档用于确定应实例化什么命令。在本章后面讲到调试 Visual Basic 代码时，就会明白这一点的重要性。

现在已经明白了组件类目的两个用途，下面将会看到如何将类注册到



ArcMap和 ArcCatalog中的自定义对话框



组件类目管理器

正确的组件类目中。开发环境对组件类目有不同级别的支持；ESRI 提供了两种添加类到组件类目中的方法；第一种方法只能用于添加到 ArcMap 或 ArcCatalog 中的命令和命令条。在 Customize 对话框中使用 Add From File 按钮（如左图所示），可能要选择一个服务器。该服务器中所有的类被添加到 ESRI Gx Commands 或 ESRI Mx Commands 中，这取决于要被定制的应用程序。尽管这个工具非常有用，但也有局限性，因为它将服务器上所有的类都添加进来。它不能移除类，并且只支持 ArcObjects 中许多组件类目中的两个类目。

随ArcGIS应用程序一起发布了一个名为组件类目管理器（Component Category Manager）的工具程序，如左图所示。这个小的应用程序可以任意移出和添加系统的任何组件类目，而不仅是ArcObjects类目。展开类目可以显示类目中的类列表。然后使用Add Object按钮显示服务器上所有类的复选清单。复选需要的类，然后这些被复选的类将被添加到类目中。

使用这些ESRI工具并不是与组件类目交互的唯一方式。在目标用户的机器上安装服务器时，可以用注册表脚本将相关的信息添加到注册表中。下面是一个这样的脚本。第一行告诉Windows这个脚本打算使用regedit的哪个版本。最后一行，以“[HKEY_LOCAL_”开始，执行注册命令；文件中所有其它行都是注释。

```
REGEDIT4
; This Registry Script enters coclasses into their appropriate Component Category
; Use this script during installation of the components

; Coclass: Exporter.ExportingExtension
; CLSID: {E233797D-020B-4AD4-935C-F659EB237065}
; Component Category: ESRI Mx Extensions
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{E233797D-020B-4AD4-935CF659EB237065}\Implemented Categories\{B56A7C45-83D4-11D2-A2E9-080009B6F22B}]
```

上面代码中最后一行在脚本中是连续的一行。

后一种方法为服务器自注册代码将服务器中相关的类添加到合适的类目中。不是所有的开发环境允许这种设置。Visual Basic不支持组件类目，虽然也有支持这种功能的插件。请参阅本章后面的Visual Basic开发插件和活动模板库（ATL，Active Template Library）部分。

下表概括了Visual Basic工程中各种元素的命名标准。

模块类型	前缀
Form	frm
Class	cls
Standard	bas
Project	prj

依据模块所提供的所有功能命名模块；不要使用任何缺省名（如“Form1”、“Class1”或Module1”等）。另外，为窗体、类和标准模块名添加三个标识其类型的字母作为前缀，如上表所示。

控件类型	前缀
Check box	chk
Combo box	cbo
Command button	cmd
Common dialog	cdl
Frame	fra
Form	frm
Grid	grd
Graph	gph
Image	img
Image list	iml
List box	lst
Label	lbl
Map control	map
List view	lvw
Map control	map
Menu	mnu
Masked edit	msk
Option button	opt
OLE client	ole
Progress bar	pbr
Picture box	pic
Scroll bar	srl
Rich text box	rtf
Slider	sld
Status bar	sbr
Text box	txt
Tab strip	tab
Tool bar	tbr
Timer	tmr
Tree view	tvw

与模块一样，根据控件提供的功能命名控件；不要使用缺省名，因为这会降低可维护性。用上表中的三字母前缀来标识控件类型。

本节的目标读者是VB6和VBA开发人员。本节明确地标出了不同开发环境的差异。

用户接口标准

要考虑到预加载窗体会增加应用程序的响应时间。注意不要预加载太多窗体（三个或四个比较好）。

操作位图文件、图标文件和相关文件时，使用资源文件（.res）而不是外部文件。

用构造函数和析构函数设置那些只在类被加载时才设置的变量引用。这些VB函数是：*Class_Initialize()*和*Class_Terminate()*，或*Form_Load()*和 *Form_Unload()*。销毁所有对象时将其设置为*Nothing*。

确保窗体的tab键顺序正确设置。不要将滚动条添加到tab键顺序中；这样容易令人产生疑惑。

为标识窗体中重要控件的标签添加访问键（使用*TabIndex*属性）。

如果可能尽量使用系统颜色，而不是代码设定的颜色。

变量声明

- 始终使用*Option Explicit*（或者在VB Options对话框中打开Require Variable Declaration）。这会强制所有变量在使用之前必须声明，因而可以防止因疏忽造成的错误。
- 在模块作用域中用*Public*和*Private*声明变量，而在局部作用域中用*Dim*声明变量。（在*Module*作用域中*Dim*和*Private*的意义相同；但用*Private*能提供更多信息）。不再使用*Global*；它只在向下与VB 3.0或更早的版本兼容时使用。
- 始终为变量、参数和函数声明明确的类型。否则缺省为*Variant*，其效率较低。
- 每行只声明一个变量，除非为每个变量指定类型。

这行中*count*声明为*Variant*，这可能不是开发者的初衷。

```
Dim count, max As Long
```

这行将*count*和 *max*都声明为Long，是开发者想要的类型。

```
Dim count As Long, max As Long
```

这两行也将*count*和 *max*都声明为Long，并且可读性更强。

```
Dim count As Long
Dim max As Long
```

圆括号

使用圆括号使优先运算和逻辑比较语句更容易阅读。

```
Result = ((x * 24) / (y / 12)) + 42
If ((Not pFoo Is Nothing) And (Counter > 200)) Then
```

用下面的句法命名变量和常量：

[<库名.>][<作用域_>]<类型><名称>

<名称>描述变量的作用或内容。<作用域>和<类型>部分应始终使用小写，<名称>应该用大小写混合。

类库名	类库
esriGeometry	ESRI 对象库
stdole	标准OLE COM库
stdole	简单变量数据类型

<类库名>

前缀	变量作用域
c	窗体或类中的常量
g	类窗体或标准模块中定义的公共变量
m	类模块或窗体模块中定义的私有变量
<empty>	局部变量

<作用域_>

前缀	数据类型
b	布尔型
by	字节或无符号字符
d	双精度型
fn	函数
h	句柄
i	整型
l	长整型
p	指针
s	字符型

<类型>

条件限定顺序

与C和C++等语言不同，Visual Basic在条件所有部分上执行条件检验，即使条件的第一部分为False。这意味着不能在条件语句的前面部分进行了有效性检验的对象和接口上执行条件检验。

```
' The following line will raise a run-time error if pFoo is NULL.
If ((Not pFoo Is Nothing) And (TypeOf pFoo.Thing Is IBar)) then
End If
' The correct way to test this code is
If (Not pFoo Is Nothing) Then
   If (TypeOf pFoo.Thing Is IBar) Then
   ' Perform action on IBar thing of Foo.
   End If
End If
```

缩进

用两个空格或两个字符宽度的制表位来缩进。由于VB代码只有一个编辑器，缩进格式问题不象在C++代码中那样至关重要。

缺省属性

除了最常见的例子外，避免使用缺省属性。这会降低可读性。

模块间引用

访问模块间数据或函数时，始终用模块名限定引用。这使得代码更具可读性并且会形成更有效的运行时绑定。

多属性操作

对同一对象的不同属性进行多个操作时，使用With … End With语句。这比每次指定对象的效率要高。

```
With frmHello
   .Caption = "Hello world"
   .Font = "Playbill"
   .Left = (Screen.Width - .Width) / 2
   .Top = (Screen.Height - .Height) / 2
End With
```

数组

对于数组，永远不要改变*Option Base*为任何非零值（0为缺省值）。用Lbound和Ubound迭代数组中所有数据项。

```
myArray = GetSomeArray
For i = LBound(myArray) To UBound(myArray)
   MsgBox cstr(myArray(i))
Next I
```

位运算符

因为*And*, *Or*, 和*Not*是位运算符，确保只用它们来检验所有条件中的Boolean值（当然，除非位运算语义发生变化）。

```
If (Not pFoo Is Nothing) Then
   ' Valid Foo do something with it
End If
```

类型后缀

禁止在变量或函数名上使用类型后缀（如*myString$*或*Right$(myString)*），除非需要区分16位和32位数值。

模糊类型匹配

使用明确的转换运算符（如*CSng*、*CDbl*和*CStr*）来进行模糊类型匹配，而不是靠VB来选择使用那一个运算符。

简单图象显示

用*ImageControl*而不是*PictureBox*来显示简单图象。这样效率更高。

错误处理

始终用*On Error*来保证容错代码。对于每个进行错误检测的函数，用*On Error*跳至程序的单个错误句柄，该句柄处理所有可能遇到的异常情况。错误句柄处理错误后—通常是显示一个消息—应通过执行左侧表中显示的一个恢复语句继续运行程序。

恢复语句	使用频率	含义
Exit Sub	通常	函数失败，将控制返回给调用者
Raise	经常	在调用者的作用域内引发一个新错误代码
Resume	少	移除错误条件，再尝试执行语句
Resume Next	非常少	忽略错误并继续执行下一条语句

Visual Basic中的错误处理与COM中的一般错误处理不同（请参见“操作HRESULTs”一节）。

事件函数

避免在事件函数中放置很多行代码以防止高度断裂和没有条理的代码。事件函数应该简单地指派其他地方的可重用函数。

内存管理

为了保证有效地利用内存资源，应该考虑以下几点：

- 定期卸载窗体。不要保留许多被加载但不可见的窗体，因为这会消耗系统资源。
- 要明白引用一个窗体作用域的变量会加载该窗体。
- 设置不再使用的对象为*Nothing*以释放内存。
- 利用*Class_Initialize()*和*Class_Terminate()*来分配和释放资源。

While Wend结构

避免使用*While … Wend*结构。而使用*Do While … Loop*或*Do Until . . . Loop*，因为可以从这个结构中有条件分支。

```
pFoos.Reset
Set pFoo = pFoos.Next
Do While (Not pFoo Is Nothing)
  If (pFoo.Answer = "Done") Then Exit Loop
  Set pFoo = pFoos.Next
Loop
```

在软件的早期的版本中，VBVM称作VB Runtime。

Visual Basic虚拟机

Visual Basic虚拟机(VBVM)包含成功执行所有Visual Basic开发代码所需的内在Visual Basic控件和服务，如开始和结束一个Visual Basic应用程序。

VBVM被封装为一个DLL，必须要安装在要执行用Visual Basic编写的代码的任何机器上，即使这些代码已经被编译成为本地码。如果查看任何Visual Basic编译文件的依赖关系，就会列出msvbvm60.dll；该DLL封装了虚拟机。

要了解有关VBVM提供的服务方面更多的信息，请参见本章的“与IUnknown接口交互”和“操作HRESULTs”部分。

与IUnknown接口交互

COM一节中详细讨论了有关*IUnknown*接口以及如何以*IUnknown*接口为基础构建所有COM方面的内容。Visual Basic对开发者隐藏了这个接口，并为开发者执行所需的交互（*QueryInterface*、*AddRef*和*Release* 函数调用）。由于包含在VBVM中的功能，可以实现这种交互。这为许多开发者简化了用COM进行的开发，但要成功操作ArcObjects，必须理解VBVM的工作原理。

Visual Basic开发人员习惯于按照下面的方式声明变量：

```
Dim pColn as New Collection 'Create a new collection object.
PColn.Add "Foo", "Bar" 'Add element to collection.
```

这时会发生什么是值得考虑的。快速地查看一下这些代码，看起来好象第一行代码创建了一个集合对象并且给开发人员一个*pColn*形式的对象句柄。然后开发人员调用该对象上的*Add*方法。在本章前面已经了解到对象通过它们的接口会话，而永远不会通过对象自身的一个直接句柄来进行会话。记住，对象通过接口提供它们的服务。如果这是事实，就没有增加某些事物。

真正发生的是在表示对象和接口方式上由VBVM执行的一些“VB魔术”和由Visual Basic编辑器执行的一些把戏。第一行代码实例化了一个集合类实例，然后为该对象将缺省的接口，*_Collection*。指派给变量*pColn*，*在_Collection*接口上定义了方法。Visual Basic隐藏了基于

接口编程的事实以简化开发工作。如果对象实现的所有功能都可以通过一个接口来访问，就不存在这个问题，但是当对象上有多个接口提供服务时，这就是一个问题了。

Visual Basic编辑器通过在IntelliSense完全列表和对象浏览器中隐藏缺省接口来弥补这种缺陷。缺省情况下，任何以下划线开头的接口都不会显示在对象浏览器中（要显示这些接口，打开Show Hidden Member，但这样仍不会显示缺省接口）。

已经知道大多数ArcObjects以*IUnknown*为缺省接口，也知道Visual Basic不提供任何*IUnknown*方法，即*QueryInterface*，*AddRef*和*Release*。假设有一个类*Foo*支持三个接口，*IUnknown*（缺省接口）、*IFoo*和*IBar*。这意味着如果如下面的代码那样来声明变量*pFoo*，*pFoo*变量会指向*IUnknown*接口。

```
Dim pFoo As New Foo ' Create a new Foo object
pFoo.??????
```

由于Visual Basic不允许直接访问*IUnknown*的方法，必须间接使用*QI*查询具有需要调用的方法的一个接口。因此，声明一个保存指向接口指针的变量的正确方法如下：

```
Dim pFoo As IFoo ' Variable will hold pointer to IFoo interface.
Set pFoo = New Foo ' Create Instance of Foo object and QI for IFoo.
```

现在有一个指向对象接口的指针，从对象请求其它任何一个接口都是一件很容易的事了。

```
Dim pBar as IBar 'Dim variable to hold pointer to interface
Set pBar = pFoo 'QI for IBar interface
```

根据惯例，大多数类都有一个与类名相同但有“I”前缀的接口；这是在操作对象时最常用到的一个接口。在实例化一个对象时，请求哪个接口是不受限制的；任何支持的接口都可以被请求，因此下面的代码是有效的。

```
Dim pBar as IBar
Set pBar = New Foo 'CoCreate Object
Set pFoo = pBar 'QI for interface
```

对象控制其自己的生存周期，这就要求每当一个接口指针被赋给另一个变量而被复制时，客户端调用*AddRef*，当不再需要接口指针时，调用*Release*。确保*AddRefs*和*Release*的数目匹配是很重要的，幸运的是，Visual Basic自动执行这些调用。这保证对象不会“泄漏”。甚至当接口指针被重用时，Visual Basic会在将新接口赋给变量时正确地在旧接口上调用*Release*。下面的代码举例说明了这些概念；注意代码执行的不同阶段对象的引用计数。