超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

目　录

第 1 章　Visual Basic.NET 入门 …… 1
1.1　Visual Basic.NET 概述 …… 1
1.2　Visual Basic.NET 的新特点 …… 1
1.3　Visual Basic.NET 的集成开发环境 …… 4
1.4　Visual Basic.NET 的第一个应用程序 …… 6
1.4.1　创建新的 Visual Basic.NET 工程 …… 6
1.4.2　创建应用程序的用户界面 …… 7
1.4.3　设置用户界面中各对象的属性 …… 8
1.4.4　编写程序代码 …… 9
1.4.5　保存和运行程序 …… 10
1.4.6　创建可执行文件 …… 10
1.4.7　小结 …… 10
第 2 章　工程管理 …… 11
2.1　工程概述 …… 11
2.2　工程文件的操作 …… 13
2.2.1　创建、打开及保存工程 …… 13
2.2.2　在工程中添加、删除和保存文件 …… 13
2.3　定制工程特性 …… 14
2.3.1　定制工程的属性 …… 14
2.3.2　在工程中添加引用 …… 16
第 3 章　Visual Basic.NET 语言体系结构 …… 18
3.1　Visual Basic.NET 的数据类型 …… 18
3.1.1　Numeric 数据类型 …… 18
3.1.2　Byte 数据类型 …… 19
3.1.3　String 数据类型 …… 19
3.1.4　Boolean 数据类型 …… 20
3.1.5　Date 数据类型 …… 21
3.1.6　Object 数据类型 …… 21
3.1.7　用户自定义类型 …… 21
3.1.8　数组 …… 22
3.2　Visual Basic.NET 的运算 …… 25

5.3　关于窗体的其他描述 …… 81

3.2.1 算术运算 …… 26
3.2.2 赋值运算符 …… 26
3.2.3 二进制运算符 …… 27
3.2.4 比较运算符 …… 27
3.2.5 连接运算符 …… 28
3.2.6 逻辑运算符 …… 28
3.2.7 运算符的优先级 …… 29
3.3 Visual Basic.NET 的常量和变量 …… 29
3.3.1 常量 …… 29
3.3.2 变量 …… 31
3.4 Visual Basic.NET 的流程和控制结构 …… 34
3.4.1 条件分支结构 …… 34
3.4.2 循环结构 …… 36
3.5 Visual Basic.NET 的过程和函数 …… 39
3.5.1 Sub 过程 …… 39
3.5.2 Function 过程 …… 40
3.5.3 调用过程 …… 41
3.5.4 向过程传递参数 …… 42
3.6 Visual Basic.NET 的类和对象基础 …… 44

第 4 章 Visual Basic.NET 的常用控件 …… 47

4.1 Label 控件 …… 47
4.2 Button 控件 …… 48
4.3 TextBox 控件 …… 49
4.4 MainMenu 控件 …… 51
4.5 CheckBox 控件 …… 52
4.6 RadioButton 控件 …… 53
4.7 GroupBox 控件 …… 54
4.8 PictureBox 控件 …… 55
4.9 ListBox 控件 …… 56
4.10 CheckedListBox 控件 …… 60
4.11 ComboBox 控件 …… 61
4.12 ListView 控件 …… 64
4.13 TreeView 控件 …… 67
4.14 ImageList 控件 …… 68
4.15 Timer 控件 …… 69
4.16 HScrollBar、VScrollBar 控件 …… 70
4.17 ProgressBar 控件 …… 71
4.18 ToolBar 控件 …… 72
4.19 StatusBar 控件 …… 73

第 5 章 应用程序界面 …… 75
5.1 界面样式 …… 75
5.2 多文档界面应用程序 …… 77
5.2.1 创建 MDI 应用程序 …… 78
5.2.2 MDI NotePad 应用程序 …… 79
5.2.3 MDI 窗体的进一步研究 …… 80
5.3 关于窗体的其他描述 …… 81
5.4 对话框 …… 83
5.4.1 模式与无模式的对话框 …… 83
5.4.2 预定义对话框的使用 …… 84
5.4.3 用窗体作为自定义对话框 …… 85
5.4.4 各种显示类型的设计 …… 86
5.5 界面设计的基本原则 …… 87
第 6 章 Visual Basic.NET 的 OOP 结构 …… 94
6.1 OOP 的相关概念 …… 94
6.2 VB.NET 的面向对象性 …… 96
6.2.1 VB.NET 的共享成员（Share Members） …… 99
6.2.2 类模块和标准代码模块的区别和比较 …… 100
6.2.3 对象浏览器 …… 101
6.3 建立和使用对象 …… 102
6.3.1 对象的生命周期：对象的建立和销毁 …… 102
6.3.2 设置和重设属性 …… 103
6.3.3 用方法来表现动作 …… 104
6.3.4 对象变量的声明 …… 105
6.3.5 对一个对象进行多种操作 …… 106
6.3.6 使用 New 关键字 …… 108
6.3.7 释放对对象的引用 …… 109
6.3.8 把对象传递到一个过程 …… 110
6.4 在程序运行中得到一个类的信息 …… 111
6.4.1 发现对象属于哪个类 …… 111
6.4.2 用一个字符名称调用一个属性或者方法 …… 111
6.5 在 Visual Basic.NET 中创建一个类 …… 112
6.5.1 给一个类添加方法 …… 112
6.5.2 命名属性，方法和事件 …… 113
6.5.3 事件和事件处理 …… 113
6.5.4 委托（delegates）、safe function points 以及 addressof 的概念 …… 116
6.5.5 声明和引发一个事件的例子 …… 117
6.5.6 事件处理的例子 …… 118
6.5.7 创建一个类的实例 …… 121

建议浏览器提醒您：
使用本复制品
请尊重相关知识产权！

6.6 Visual Basic.NET 的接口（Interfaces） ……128
6.7 类的继承（Inheritance） ……130
6.7.1 继承的规则 ……131
6.7.2 用继承建立一个继承类 ……132
6.7.3 重载 Windows 控件 ……133
6.7.4 什么时候使用继承 ……135
6.7.5 命名空间（Namespace） ……138
6.7.6 关于继承的例子 ……139
6.8 多态性 ……143
6.8.1 用继承实现多态性 ……143
6.8.2 用接口来实现多态性 ……144
第 7 章 数据库访问技术 ……147
7.1 理解数据库 ……147
7.2 Visual Basic.NET 开发数据库初步，数据绑定显示 ……148
7.2.1 数据集（DataSets）的概念 ……148
7.2.2 在 Windows 窗体中显示单个表的数据 ……150
7.2.3 在窗体上显示带参数的数据查询 ……156
7.2.4 用 Data Form Wizard 生成数据绑定控件 ……162
7.2.5 利用 Data Form Wizard 创建一个 Master/Detail 窗体 ……166
7.3 SQL 语言基础 ……170
7.3.1 SQL 发展历史 ……170
7.3.2 数据类型 ……171
7.3.3 运算符 ……172
7.3.4 变量 ……173
7.3.5 流控制语句 ……173
7.4 SQL 语句函数 ……175
7.4.1 日期函数 ……175
7.4.2 字符串函数 ……175
7.4.3 数学函数 ……176
7.4.4 集合函数 ……177
7.4.5 文本和图像函数 ……177
7.4.6 转换函数 ……177
7.5 表、视图与索引 ……178
7.5.1 表 ……178
7.5.2 表数据操作 ……179
7.5.3 索引 ……181
7.5.4 视图 ……182
7.5.5 查询 ……182
7.5.6 统计 ……185

7.5.7 利用查询结果创建新表 …… 187
7.5.8 使用 UNION 运算符实现多查询联合 …… 187
7.5.9 连接 …… 188
7.5.10 子查询 …… 189
7.6 数据库访问对象 DAO …… 190
7.6.1 何时使用 DAO …… 190
7.6.2 DAO 和 Jet 数据库引擎 …… 191
7.6.3 DAO 对象模型 …… 192
7.7 远程数据对象 RDO …… 197
7.7.1 RDO 对象模型 …… 197
7.8 ODBC API …… 202
7.8.1 ODBC 结构 …… 202
7.8.2 使用 ODBC API 访问数据库 …… 204
7.9 ADO 数据对象 …… 206
7.9.1 ADO 对象模型 …… 206
7.10 ADO.NET …… 211
7.10.1 ADO.NET 简介 …… 211
7.10.2 使用 ADO.NET 的基本方法 …… 212
7.11 用面向对象思想处理数据库的例子 …… 214
第 8 章 Visual Basic.NET 的多线程 …… 232
8.1 自由线程（Free Threading） …… 232
8.1.1 建立和使用一个新的线程 …… 232
8.1.2 关于线程的参数和返回值的问题 …… 232
8.1.3 并发性 …… 233
8.2 自由线程的举例 …… 234
第 9 章 VB.NET 的数组、文件和出错处理 …… 238
9.1 数组 …… 238
9.1.1 数组（Arrays） …… 238
9.1.2 数组列表（ArrayLists） …… 239
9.2 出错处理 …… 240
9.2.1 出错处理 …… 240
9.2.2 多种出错处理（Multiple Exceptions） …… 241
9.2.3 忽略错误（Throwing Exceptions） …… 243
9.3 文件处理（File Handling） …… 243
9.3.1 文件对象（file object） …… 243
9.4 文件处理的出错处理 …… 244
9.5 检测文件的结束（Testing for End of File） …… 245
9.6 文件的静态方法（Static File Methods） …… 246

使用本复制品
请尊重相关知识产权!

第 1 章　Visual Basic.NET 入门

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

本章包括：

Visual Basic.NET 概述
Visual Basic.NET 的新特点
Visual Basic.NET 的集成开发环境
Visual Basic.NET 的第一个应用程序

1.1　Visual Basic.NET 概述

Visual Basic.NET（又称 Visual Basic 7.0 或 VB 7.0）是 Microsoft 公司在 Visual Basic 6.0 之后推出的最新版本，集成在 Visual Studio.NET 7.0 中，与 Visual C++ 7.0 以及 C# 组成了“.NET”构架。

Visual Basic 是 Windows 环境下简单、易学、高效的一种编程语言，其快速开发的特性深受程序员的喜爱，但是.NET 以前版本的 Visual Basic 面向对象的能力远远不能满足程序员的需要，这也是越大项目越少用到 Visual Basic 的原因。Visual Basic.NET 新增和加强了许多新的面向对象的特性比如继承、重载等等。语言的新特点也包括了对进程的控制和低层结构的操作，这些新特性使得 Visual Basic.NET 再次成为程序员关注的焦点。

1.2　Visual Basic.NET 的新特点

一直以来，大家总是针对 Visual Basic 究竟是“面向对象”还是“基于对象”，甚至“面向组件”的语言而争论不休。但是，这些争论在 Visual Basic.NET 面前，马上就会偃旗息鼓——因为，不管依照怎样的标准来定义“面向对象”，Visual Basic.NET 绝对是符合面向对象的每一个标准的。下面就来看看它有着怎样的特性。

1. 构造函数

当一个对象被创建的时候，它是否能够被正确地初始化，这是我们比较关心的问题。利用构造函数我们就可以同时给这个对象中的成员赋初值，这样有助于我们正确地初始化一个对象。Visual Basic 以前的版本中，在创建对象的时候，必须要等到对象创建完毕之后，另外调用一个独立的方法来对它进行初始化。在 Visual Basic.NET 中，我们可以利用构造函数给对象赋初值——这样我们就不需要再去进行烦琐的调用赋初值了。构造函数简化了编码的过程，也减少了出错的机会。

2. 封装特性

从字面上理解，封装就是将某事物包围起来，使外界不知道其实际内容。在程序设计中，封装的意思就是程序提供一个包含了一系列过程和函数的接口，其他的程序可以直接利用对象的这个接口，而不需要去了解接口里面的具体代码。接口中的代码一般来说总是实现一些相对完整的功能。

3. 自由线程

线程是进程中的一个实体，是被系统独立调度和分派的基本单位。线程是一个很复杂的概念，一时半会儿解释不了。但是，基本上来讲，线程的意思就是让我们的程序可以在同一时间段内做两件以上的事情。比如，我们也许需要让程序在后台打印一个文档，同时还要让它响应当前用户不断发出的命令。线程在创建具有高度扩展性服务端程序的时候显得比较有用，并且，利用线程来编写具有极强交互性用户界面的时候也十分容易，例如我们可以为一个需要长时间运行的程序加上一个"Cancel"按钮，以便用户有更多的选择。以前 Visual Basic 编程的时候对线程想尽了办法，调试线程的复杂和不稳定性使线程编写相当的复杂（因为只能使用 API 来写，还是伪线程的，说是多线程不如说是多进程恰当）。现在，Visual Basic 提供了编写线程的能力，线程编写在调试上是相当复杂的，Visual Basic.NET 终于以一种相对简单的形式实现出来。

4. 继承

继承是面向对象系统中的另一个重要的概念，一个语言是否具有继承特性常被人们用作判断是否是面向对象语言的关键标准。继承所表达的就是一种类的相交关系。它使得某类可以继承另外一种类的特征和能力，类间具有继承关系应有三种特征：类间具有共享特征；类间具有细微的差别或新增部分；类间具有层次结构。

从对象的角度来看，继承是这样的一种概念：如果一个对象能够获得另外一个对象的接口和方法，并且可以扩展这些接口和方法，我们就称这个对象继承了另一个对象。举个实际的例子，我们在产品存储程序中，可以创建一个通用的处理所有产品的产品对象"Product"。从这个对象中，根据是否需要上税，又可以派生出一个免税产品对象"NonTaxableProduct"和一个需税产品对象"TaxableProduct"。两个对象都将继承原始产品对象 Product 的接口和所有方法，但是将根据各自不同的实际情况在需要的地方修改或者扩展原始对象的方法。

5. 初始化函数

以前在 Visual Basic 中还需要首先声明一个变量，然后才能对它赋初值 —— 这样用户需要写两行代码。初始化函数可以让用户将这两个步骤合并在一行代码中完成。虽然这是一个微小的变化，但是从代码的角度来看，这个改进提供了更少、更简单、更容易维护的代码。

6. 基于对象

对于一种计算机语言来讲，做到基于对象要比面向对象相对容易些。不过，什么是基于对象，到现在也只有一个很模糊的定义——大概是指一种语言具有直接和对象进行交互的能力。VB 3.0 就已经是基于对象的语言了，当时它能够和 DAO 对象以及控件交互；后来随着版本的提升，这些可交互的对象已经扩大到了 ActiveX 控件、RDO、ADO 对象等。

7. 面向对象

一种面向对象的语言至少需要满足三个条件。封装性（Visual Basic 4.0 中就已经实现了），继承性（Visual Basic.NET 中才有），多态性（多态性指的是多形态，就是对各个对象

的相同接口，Visual Basic 3.0 中就有了）。所以在 Visual Basic.NET 中，实现了完全的面向对象，因为它已经完全满足了这三个条件。

8. 重载

重载的含义是指通过为函数或运算符创建附加定义而使它们的名字可以重载。也就是说相同名字的函数或运算符在不同场合可以表现出不同的行为。对象们经常在类似的数据上开展同样的功能，比如，我们需要能够加服务和产品项到一个订购对象上。现在，我们必须分开对订购对象执行一个方法（添加服务）和另一个方法（添加产品项）。利用函数重载，我们仅需要执行 ASS 方法，然后让 Visual Basic 语言自己根据情况去选择执行正确的函数。

9. 覆盖

当使用继承功能的时候，新类从父类那里得到了一切方法。但是，也许我们需要让其中的某一种或者几种函数执行另外的动作，可以通过覆盖原始的方法来实现这样的功能。我们的新代码也可以调用原始的父类对象的原始方法，这样的过程可以使用 XX 关键字来实现（比如 MY）。

10. 多态性

简单地说，这就是让两个不同的类型的对象，执行同一种方法的能力。另一种解释的方法是，描述同一个消息可以根据发送消息对象的不同采用多种不同的行为方式。Visual Basic 支持后来的捆绑，而在 Visual Basic 5.0 中通过执行关键字和多个 COM 接口来实现多态。现在，在 Visual Basic 版本中，我们将看到继承接口同样也可以具有多态性。最后，Visual Basic.NET 在执行多态性的时候给程序相当大的灵活性。

11. 共享成员

共享成员又叫做静态成员和类级成员。共享成员就是对类的所有的实例来说，相同的方法或变量。每一个对象创建的时候，基于一个给定的类。分享这些相同的变量和函数。这意味着，比如，我们可以计算出一个类有多少个实例创建了，只要我们声明一个共享的计数器变量。

12. 结构化错误处理

替代了不稳定的也极不灵活的“On Error Goto”语句，我们有一个新的结构来处理错误。这个结构包含了“TRY”、“CATCH”和“END TRY”等关键字。在“TRY”和“CATCH”之间的代码块被保护，如果有个错误出现了，这个“CATCH”代码快就会运行，在“CATCH”运行完之后，模块内的代码就做错误结束处理完之后的清除和收尾的工作。

13. 类型安全保证

今天，Visual Basic 在变量的类型转换上显得特别前卫。它会自动将数字转换成字符串打印出来，并且还有更多的类似的功能。有时候这样的情况会导致不可知的副作用。有了类型安全控制后，我们将可以自己选择让 Visual Basic 避免自动做类型转换，在程序编写时避免一些不可预知的错误是十分有用的。

14. 用户界面继承

光有语言继承还是不够的，Visual Basic.NET 还可以实现表单的继承特性。这意味着我们能创建一个基本的表单模板，也许模板中包含一个标准的工具条、公司标志和颜色，然后从这个模板中派生出我们所需要的其他模板。所有其他的模板将从原始模板继承模板的样式和代码。一个对于原始模板的改变，可以自动地波及所有由此模板派生出来的其他表单，只需要简单的编译即可。

15. Web 表单

很多年以来，Visual Basic 很广泛地被认为是强大的窗体（Form）设计工具。我们也喜欢并习惯于在控件上双击出来一个编码窗口，然后我们就可以在窗口里面写那个控件的事件了。WEB 表单将这个特性带给了 HTML 开发。WEB 表单是浏览器为平台的，所有的代码都运行在 WQEN 服务器上，但同时它给我们拉放表单设计和双击控件编写服务端的时间代码的能力，而且这是真正的 Visual Basic 代码不是 VBS 代码。

16. Web 服务

一个 Web 服务是一个组件，运行于一个 Web 服务器上并且允许客户程序通过 HTTP 调用其方法。在组件上的每个方法都表现为一个 URL 并且可以返回数据（或许是 XML 文档）和接受参数输入。这样的技术基于的是开放 SOAP 标准，所以现在用户的服务端组件实际上可以被任何客户访问到，而不管是什么语言或者平台。

有了这些改变及上述提到的这些新特性，Visual Basic.NET 成为了一个完全的面向对象的语言，Visual Basic 仍然是世界上最广泛使用的开发工具。

1.3 Visual Basic.NET 的集成开发环境

Visual Basic.NET 的集成开发环境（IDE）集成了许多功能，如设计、编辑、编译、调试等，新建一个 Visual Basic 项目时，可以看到集成开发环境的界面，如图 1.1 所示。

图 1.1 Visual Basic 集成开发环境

Visual Basic.NET 集成开发环境由以下元素组成：

- 菜单栏

显示所有 Visual Basic.NET 使用的命令与设置。除了提供标准的“File”（文件）、“Edit”（编辑）、“View”（视图）、“Tools”（工具）、“Windows”（窗口）、“Help”（帮助）之外，

还提供了“Project”（项目）、“Format”（格式）、“Debug”（调试）、“Build”（全程编译）的功能菜单。

● 工具栏

在编程环境下提供对于常用命令的快速访问。单击工具栏上的按钮，则执行该按钮所代表的操作。将鼠标光标悬浮按钮上，会有按钮所对应的功能提示弹出。Visual Basic.NET 新增了“Debug”（调试）和“Search”（查找）的功能，方便了程序的调试。其他按钮可以从“View”（视图）菜单里的“Toolbar”（工具栏）子菜单选定。工具栏可依次紧贴在菜单之下，或以垂直条状紧贴在左边框上，如果将它从工具栏中拖开，则它将“悬”在窗口中。

使用本复制品
侵犯相关知识产权!

● 工程资源管理窗口

列出当前工程中的所有的窗体和模块，可以按照“浏览器”（Solution Explorer）的形式浏览，也可以按“类试图”（Class View）的形式浏览。

● 属性窗口

列出所选定对象的属性值，为对象的初始化赋值。

● 窗体设计器

作为应用程序编写的基础，窗体设计器承担了应用程序界面的设计任务。在窗体上添加控件、组件、图形、图片等对象来设计出应用程序的外观。应用程序中的每一个窗体都有自己的窗体设计器窗口。

● 活动栏

Visual Basic.NET 新添了“活动栏”，可以将“服务器浏览器”及“工具箱”等收在左边框上，也可以单击“活动栏”，再单击活动栏上的“ ”按钮，使活动栏与工程资源管理器对称，同时“ ”变为“”形状，再单击“”，活动栏恢复原来状态。

● 代码编辑窗口

双击窗体设计器或单击工程资源管理器上的“ ”图标，则出现代码编辑窗口，如图 1.2 所示。

图 1.2　代码编辑窗口

可以看到代码编辑窗口新增了“⊟”图标，点击这个图标后，图标后面对应的过程或函数将被挂起，只保留过程或函数的名字，整个代码界面将显得十分简洁。“对象列表框”显示所选对象的名称。单击列表框右边的箭头，显示与该窗体有关的所有对象的清单；“过程列表框”列出对象的过程或事件，在该列表框中选定过程的名称，选取该列表框右边的箭头可以显示这个对象的全部事件。

1.4 Visual Basic.NET 的第一个应用程序

超星阅览器提醒
使用本复制品
请尊重相关知识产权!

创建 Visual Basic.NET 应用程序有以下几个主要步骤：

（1）创建新的 Visual Basic 工程
（2）创建应用程序的用户界面
（3）设置用户界面中各对象的属性
（4）编写程序代码
（5）保存和运行程序
（6）创建可执行文件

为了实现上面这些过程，可以按照以下步骤来创建一个简单应用程序，该应用程序由一个文本框（TextBox）和一个按钮（Button）组成。单击按钮，文本框会出现“Hello,World!”，同时按钮上的文本由“Birthing”变成“Birthed”。

1.4.1 创建新的 Visual Basic.NET 工程

在.NET 框架中创建 Visual Basic.NET 的过程有一点类似用 MFC 创建 VC 应用程序的过程，需要在向导中填写必要的信息，它的过程如下：

（1）打开 Visual Studio.NET。
（2）点击 File→New→Project...，弹出如图 1.3 所示的对话框。

图 1.3 创建新的 Visual Basic.NET 工程

（3）在“Project Type”（项目类型）中选择“Visual Basic Projects”（Visual Basic 项目）

选项。

（4）在“Templates”中选择“Windows Application”（Windows 应程序）选项。

（5）在“Name”中添入应用程序的名称（默认为“WindowsApplication&”，&为一常数）。

（6）在“Location”（位置）中添入应用程序所在目录的名称，也可以通过“Browse...”直接选择目录。

（7）其他项采用默认值，单击“OK”即可生成应用程序。

1.4.2　创建应用程序的用户界面

窗体是创建应用程序的基础。通过使用窗体可将窗体和对话框添加到应用程序中。也可以把窗体作为项的容器，这些项是应用程序界面中的不可视部分。例如，应用程序中可能有一个作为图形容器的窗体，而这些图形是打算在其他窗体中显示的。

建造 Visual Basic 应用程序的第一步是创建窗体，这些窗体将是应用程序界面的基础。然后在创建的窗体上绘制构成界面的元素，通常是在工具箱中选择控件或组件，对于当前的应用程序，可在工具箱中选择两个控件，具体过程如下：

（1）单击活动栏“Toolbox”，再单击要绘制的控件的工具，在这里是“TextBox”控件。

（2）将指针移到窗体上，该指针变成十字线。

（3）将十字线放在控件的左上角。

（4）拖动十字线画出适当的大小。

（5）释放鼠标按钮。

（6）以相同方法在窗体上添加“Button”控件。

经上述操作之后，控件将会出现在窗体上。

在窗体上添加控件的另一个简单方法是双击工具箱控件中的按钮。这样会在窗体缺省位置创建一个尺寸为缺省值的控件；然后再将该控件移到窗体中的其他位置。

对于绘制窗体的控件，有时还要调整它的大小、移动和锁定控件，下面对这些操作进行详细介绍。

● 调整控件尺寸

（1）用鼠标单击需要调整大小的控件，选定的控件上出现尺寸句柄。

（2）将鼠标指针定位到尺寸句柄上，拖动该尺寸柄直到控件达到所希望的大小为止。角上的尺寸句柄可以同时调整控件水平和垂直方向的大小，而边上的尺寸句柄调整控件一个方向的大小。

（3）释放鼠标按钮。

也可以按【Shift】键加上箭头键调整选定控件的大小。

● 移动控件

（1）用鼠标把窗体上的控件移动到一个新位置或用属性窗口改变“Top”和“Left”属性。

（2）选定控件后，可用【Ctrl】键加箭头键每次移动控件的一个网格单元。如果该网格关闭，控件每次移动一个像素。

● 锁定所有控件位置

这个操作将把窗体中的所有控件锁定在当前位置，以防止已处于理想位置的控件因为

不小心而被移动。本操作只锁定选定窗体上的全部控件，不影响其他窗体上的控件，这是一个切换命令，因此，也可以用来解锁定。

● 调整锁定控件的位置

按住了【Ctrl】键，再用合适的箭头可微调已获焦点的控件的位置，或者可以在“属性”窗口中改变控件的“Top”和“Left”属性。

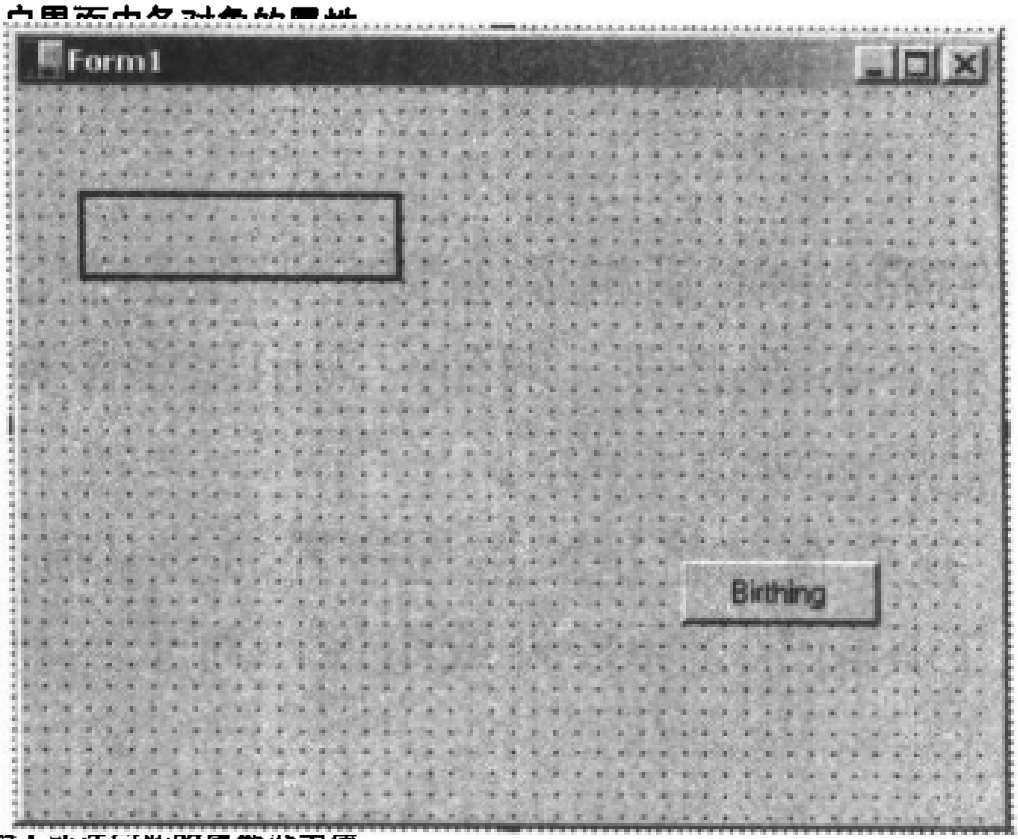
现在已经生成了“Hello,World!”应用程序界面，如图 1.5 所示。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

图 1.4 用工具箱绘制文本

图 1.5 “Hello,World!”应用程序界面

1.4.3 设置用户界面中各对象的属性



排列顺序

图 1.6

可直接选中窗体

在右列中输入或选定新的属性设置值。

在本例中，需要改变三种属性的设置值，其他属性值采用缺省的值，如表 1.1 所示。

表 1.1　设定的三种属性值

对　象	属　性	设　置
Form1	Text	FirstVB.NET
TextBox	Text	缺省
Button	Text	Birthing

使用本复制品
请尊重相关知识产权!

1.4.4　编写程序代码

Visual Basic.NET 在创建应用程序的同时，生成了一部分代码，这些代码实现了程序的初始化等问题，代码的编写要在代码编辑窗口中，程序员需在 Visual Basic 生成的应用程序的框架中编写代码。

1. 打开代码窗口

双击想要编写代码的窗体或控件，或从“工程资源管理器”窗口，选定窗体或模块的名称，然后选取“[图]”按钮，则可显示如图 1.7 所示的代码编窗口。

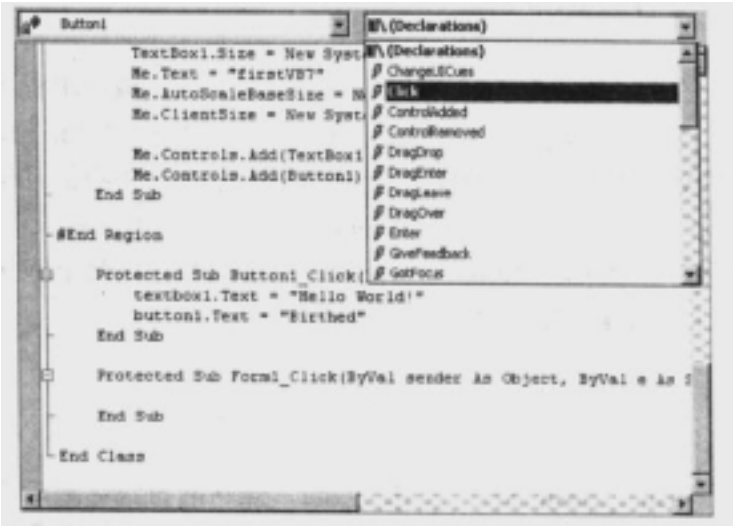


图 1.7　代码编辑器窗口

2. 事件过程的创建

Visual Basic 应用程序的代码被分为称为过程的小的代码块。事件过程，正如下面将要创建的一样，包含了事件发生（例如，单击按钮）时要执行的代码。控件的事件过程由控件的实际名称（“Name”属性中所指定的）、下划线（_）和事件名组合而成。例如，在单击一个名为“Button1”的命令按钮时调用的“Name”事件过程，可称为“Button1_Click”事件过程。下面就来详细讨论创建事件的过程。

（1）在对象列表框中，选定活动窗体中的一个对象名（活动窗体是指当前有焦点的窗体）。在这一例子中，命令按钮为“Button1”。

（2）在过程列表中，选择指定对象的事件名。此时“Click”过程已经被选定，因为它是命令按钮的缺省过程。注意这时事件过程的模板已经显示在“代码”窗口中了。

（3）在“Sub”和“End Sub”语句之间输入下面的代码：

```
Textbox1.Text="Hello,World!"
Button1.Text="Birthed"
```

使用本复制品，
请尊重相关知识产权！

这个事件过程应该是：

```
Protected Sub Button1_Click()
    Textbox1.Text="Hello,World! "
    Button1.Text="Birthed"
End Sub
```

1.4.5 保存和运行程序

在创建 Visual Basic.NET 的应用程序的时候，需要给应用程序创建一个目录，专门保存应用程序所有的文件。Visual Basic.NET 每次编译运行的时候，都会首先保存改变的应用程序的窗体或模块到这个默认的目录下。点击工具栏上的“▶”按钮或按【F5】键即可运行应用程序。

1.4.6 创建可执行文件

与以往版本不同的是，Visual Basic.NET 并没有专门的生成应用程序（.exe 文件）的选项，应用程序在工程建立的同时就已经生成，保存在工程目录“Obj→Debug→Temp”中，但是这时的应用程序仅仅完成显示界面的功能，而不完成程序本身的功能。在经过至少一次的编译后，这个可执行文件有了用户所要求的功能，同时在工程目录“Obj→Debug”中，生成了与工程同名的应用程序，“Temp”中和“Debug”中的这两个应用程序在运行结果上完全一致，而且随着每次的编译时时更新最新的功能。

1.4.7 小结

本章简单介绍了 Visual Basic.NET（.NET）的一些新的特性，通过本章的学习，读者应该对 Visual Basic.NET 的集成开发环境有了初步的了解，在后面的章节中，将会详细地对 Visual Basic.NET 的程序设计、语言的体系结构、系统的功能调用等进行介绍。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

第2章 工 程 管 理

本章要点：
工程概述
工程文件的操作
定制工程特性

2.1 工 程 概 述

一个应用程序的开发过程，往往是程序员与一系列文件，如工程文件（扩展名为“.sln”）、窗体文件（扩展名为“.vb”）打交道的过程，而程序员所做的就是在窗体中添加控件、编写程序运行时的代码，至于这些文件的管理关系、组织关系等并不用程序员需考虑的，Visual Basic.NET 集成开发环境已经完成了大部分的工作了，它将为程序创建一系列的文件形式来存储有关信息。而“工程”就是为这些应用程序创建的文件的集合。

在开发应用程序时，要使用工程来管理构成应用程序的所有不同的文件。工程包括“本地工程”和“网络工程”。

本地工程的文件元素如表 2.1 所示。

表 2.1 本地工程的文件元素

工程文件元素	文件扩展名	工程文件描述
工程文件	.sln 或.vbproj	用来管理所有工程文件
Windows 窗体	.vb	应用程序的最基本的图形界面
继承窗体	.vb	一个从另一个已有的窗体派生出的窗体，可以继承、增加或改变原有的窗体中的成员变量或函数
类	.vb	一个代码文件，包含一个简单、空的类的声明
模块	.vb	一个代码文件，包含一些变量或函数
代码文件	.vb	一个空的 Visual Basic 代码文件，不包含任何代码
Windows 服务	.vb	一个用来建立 Windows 服的文件，包含一个可视的设计器及一个类，这个类包含能够建立服务的方法
Crystal 报表	.rpt	一个 Crystal 报表（第三方控件，用来生成数据库报表的工具）文件，当打开这个文件时，自动生成 Crystal 报表设计器
XSD 图表	.xsd	一个 XSD 图表文件
XML 文件	.xml	一个 XML 文档
XSLT 文件	.xslt	用来转换 XML 和 XSD 文档的文件
HTML 页面	.htm	一个典型的网页格式

使用本复制品
请尊重相关知识产权!

网络工程的文件元素如表 2.2 所示。

表 2.2　网络工程的文件元素

工程文件元素	文件扩展名	工程文件描述
Web 窗体	.aspx	用在网络应用程序的窗体，类似 Windows 窗体于本地工程的关系，提供一个可视化的图形界面设计器
HTML 页面	.htm	同本地工程
活动服务网页	.asp	一个 ASP 页面，用来声明 HTML 的格式以及服务器端的脚本
Web 服务	.asmx	一个类，提供访问“HTTP”的方法，有图形设计界面
类	.vb	同本地工程
模块	.vb	同本地工程
代码文件	.vb	同本地工程
Crystal 报表	.rpt	同本地工程
XSD 图表	.xsd	同本地工程
网络结构文件	.web	ASP.NET 用这个文件确定网络工程中“Web”页的设置，这个文件有一个确定的网页的名字，这个名字不能更改
全局应用类	.asax	也可以叫做“asax”文件，这个文件允许编写代码来处理全局 ASP.NET 的应用等级事件，如：Session_OnStart（会话开始）、Aplication_OnStart（应用开始），这个文件的名称是“Global.asax”，并且不能更改名称

工程资源管理器：

工程资源管理器是 Visual Basic 集成开发环境对工程文件管理的工具，通过它可以创建、添加以及删除一个工程中的可编辑的文件，如图 2.1 所示。

Visual Basic.NET 的工程资源管理器较以前版本有所变化，除了显示的内容不同外，形式也多样化了，新增了类视图，类视图将工程中的所有工程中的元素基于对象表示出来，并且可以显示对象的属性、方法以及事件等。类视图可以快速找到一个函数或过程，在面向对象的编程过程中，起了很大的作用，类视图如图 2.2 所示。

图 2.1　Visual Basic.NET 工程资源管理器

图 2.2　Visual Basic.NET 的类视图

图 2.3　类视图的四种排序

点击工程资源管理器左上角的图标则显示如图 2.3 所示的下拉列表。

类视图有 4 种排列方式：

- 按字母排序（Class View Sort Alphabetically）
- 按类型排序（Class View Sort By Type）

- 按访问排序（Class View Sort By Access）
- 按组别排序（Class View Group By Type）

2.2　工程文件的操作

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

2.2.1　创建、打开及保存工程

1. 创建新工程

创建新工程的步骤可以参见第 1 章中“1.4.1 创建新的 Visual Basic.NET 工程”。

2. 打开现存的工程

如果希望打开已有的工程，可以通过下面的步骤：

（1）点击“File”（文件）菜单下的“Open”（打开）→“Project…”（工程），则弹出如图 2.4 所示的对话框。

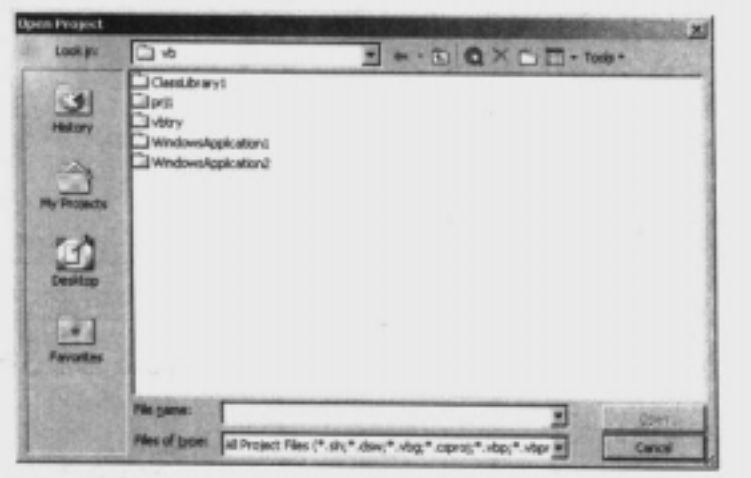


图 2.4　打开工程对话框

（2）在指定的目录下选择要打开的工程文件（.sln 或.vbproj 文件），点击“Open”按钮即可。

3. 保存工程

点击“File”（文件）菜单下的“Save Form”（或“Save Module”等），则保存当前所编辑的窗体（或模块等），也可以按快捷键【Ctrl+S】。

点击“File”（文件）菜单下的“Save All”，则保存当前工程所有的正在编辑的文件，也可以按快捷键【Ctrl+Shift+S】。

2.2.2　在工程中添加、删除和保存文件

1. 向工程中添加文件

在工程的设计过程中，需要向工程中不断添加一些文件（如窗体、类模块等）。添加的

文件分两种：

一种是新创建的文件，可以按以下步骤：

（1）点击“Project”（工程）菜单下的“Add New Item…”（添加新选项），则弹出如图 2.5 所示的对话框。

（2）选择新的文件种类，然后在“Name”中给新文件添一个名称，点击“Open”即可。

第二种是添加已有文件，添加已有文件的步骤如下：

（1）点击“Project”（工程）菜单下的“Add Existing Item…”（添加已存选项），则弹出如图 2.5 所示的对话框。

（2）在指定的目录下选择需要插入的项目，点击“Open”即可。

添加新项目可以利用可重用的资源，从而节省代码编写的时间。和以前版本的 Visual Basic 有所不同，Visual Basic.NET 并不是在工程中记录新添加的已存文件的路径，而是将这个文件复制到当前工程所处的目录下，从某种程度上减少了类似 Visual Basic 以前版本的窗体加载这类的错误。

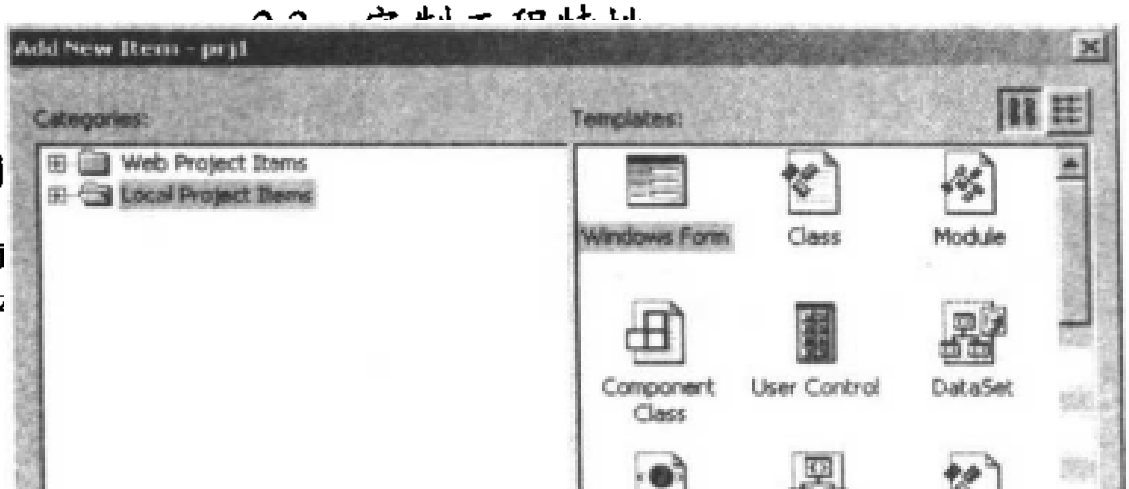
图 2.5　添加新项目对话框

图 2.6　添加已存文件对话框

2. 删除和保存工程中的文件

如果工程中有些文件，在工程中失去了作用，如将两个模块合并，这个时候就需要将多余的模块从工程中删除。删除工程中的文件的方法比较简单，只需在工程资源管理器中右键单击待删除的文件名，在弹出的菜单中选择“Delete”按钮即可，这时会弹出一个警告对话框，以确定是否完全删除文件，一旦选择“OK”按钮，则这个文件将被彻底从工程中删除，这个过程是不可逆的，就是说，在工程所在的目录里也将该文件删除了。因此在删除文件之前，可以先将文件另存在其他目录，以供其他工程使用。

2.3 定制工程特性



2.3.1 定制

工程的属

object）等基

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

共享（Sharing）的高级信息。

工程的属性可以按以下步骤定制：

（1）在工程资源管理器中右键单击工程的名称，这个名称一定是以“.vbproj”为扩展名的工程名。

（2）在弹出的菜单中选择“Properties”，则弹出如图 2.7 所示的对话框，在这个对话框中就可以设置工程的各种属性了。

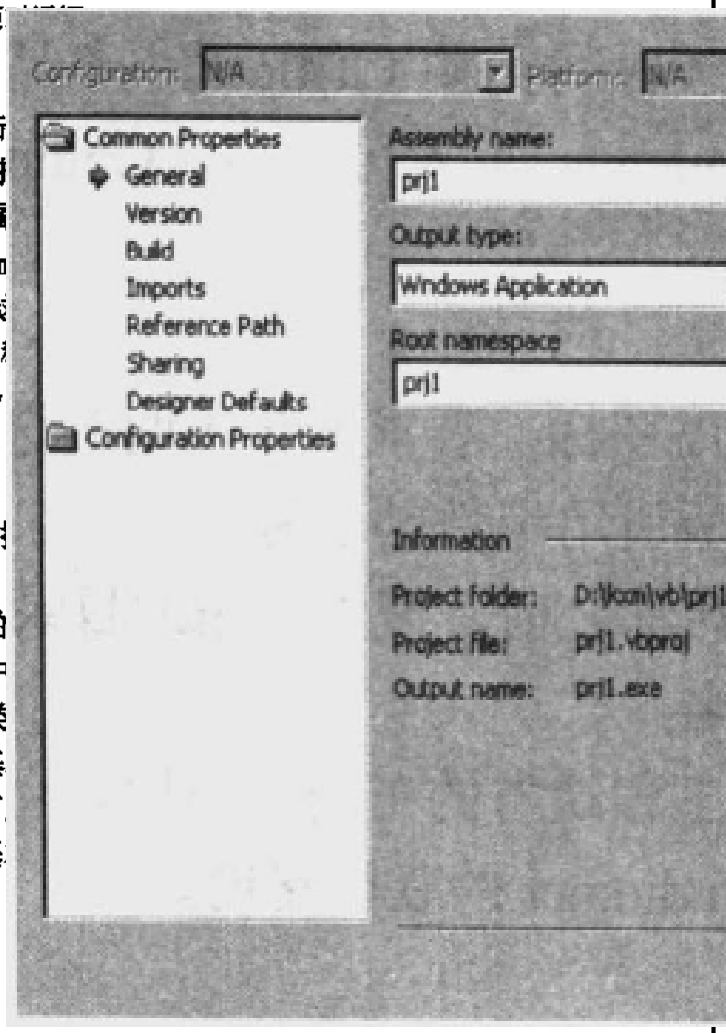


图 2.7　工程属性页

1. 设置基本信息

工程的通用信息是工程的最基础的信息，点击选项即可获得工程的基本信息，工程的基本信息包括

（1）汇编名称（Assembly name）：可更改的属的名称，改变这个属性的同时，工程目录中的“Bin

（2）工程种类（Output type）：可选的属性，这要求而设置的，根据不同的应用程序可以设置工程序（Console Application）以及类库（Class Library性只能被设置为类库的形式。

（3）启动对象（Startup object）：可选的属性，通常这个属性设置为主窗体，或者程序运行前需要入口点，因此这个属性为“None”。

（4）根命名空间（Root namespace）：可更改的程名，如果当前工程名称为“Project1”，而工程中的其他文件可以用“Project1.Class1”来访问这个类

（5）工程所在目录（Project folder）：工程的基

（6）工程文件名（Project file）：工程的基本信

（7）应用程序名称（Output name）：工程的基程序的名称，随汇编名称的改变而改变。

2. 工程的版本信息

工程的版本信息是记录由当前工程生成应用程序的版本，点击属性页左侧的配置管理器中的“Version”选项即可获得工程的版本信息，版本信息包括：

（1）版本号（Version Number）：版本号由四位整数组成，分别代表主版本（Major）、次版本（Minor）、修正（Revision）、编译（Build），在版本号后面有一个复选框，根据修正和编译次数自动升级（Generate Revision and Build numbers），默认为选定该项，表示自动获取修正及编译信息来更改版本号。

（2）版本信息（Version Information）：包括标题（Title）、产品（Product）、描述（Description）、版权（Copyright）、商标（Trademark）及公司名（Company）。

3. 属性页中的其他选项

属性页配置管理器中还包括：

（1）编译信息（Build）：包括设置应用程序图标，要求变量声明等选项。

（2）输入项（Imports）：用来定义工程中所引用组件的名称。

（3）引用路径（Reference Path）：在工程启动之前，搜索引用的目录。

（4）共享（Sharing）：给工程设置的共享名。

（5）缺省设置（Designer Defaults）：设计工程缺省的布局及结构。

这些设置的具体使用方法请参阅 Visual Studio.net 的联机帮助。

2.3.2 在工程中添加引用

给工程添加引用有两种：一种是添加本机的引用（Add Reference）；一种是添加网络引用（Add Web Reference）。通过引用可使用其他应用程序的对象或一些组件以及第三方控件等。

1. 添加本机的引用

选择“Project”菜单下的“Add Reference…”则会出现如图 2.8 所示的组件选择对话框，这个对话框罗列着各种类型的可供引用的组件，组件选择对话框由以下元素组成：

（1）Component Name 是组件的全称，点击此标题，则组件按名称排序；

（2）Version 是组件的版本号，点击此标题，则组件按版本号排列；

（3）Path 是组件所在本地路径，点击此标题，则组件按路径排列；

（4）Select 将选中的组件添加到选中组件列表中，也可以双击组件名完成此功能；

（5）Browse 可以添加附加的组件；

（6）Selected Components 显示所有选中的组件；

（7）Type 为列表中的组件类型，点击此标题，则组件按类型排列；

（8）Source 为组件的名称及路径，点击此标题，则组件按路径排列；

（9）Remove 删除选中的组件，但并不是真正从系统中删除。

2. 添加网络引用

应用此功能可以将网络中存在且提供服务的组件引用到本地的应用程序中，在“Project”菜单中点击“Add Web Reference…”，则弹出如图 2.9 所示的对话框。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

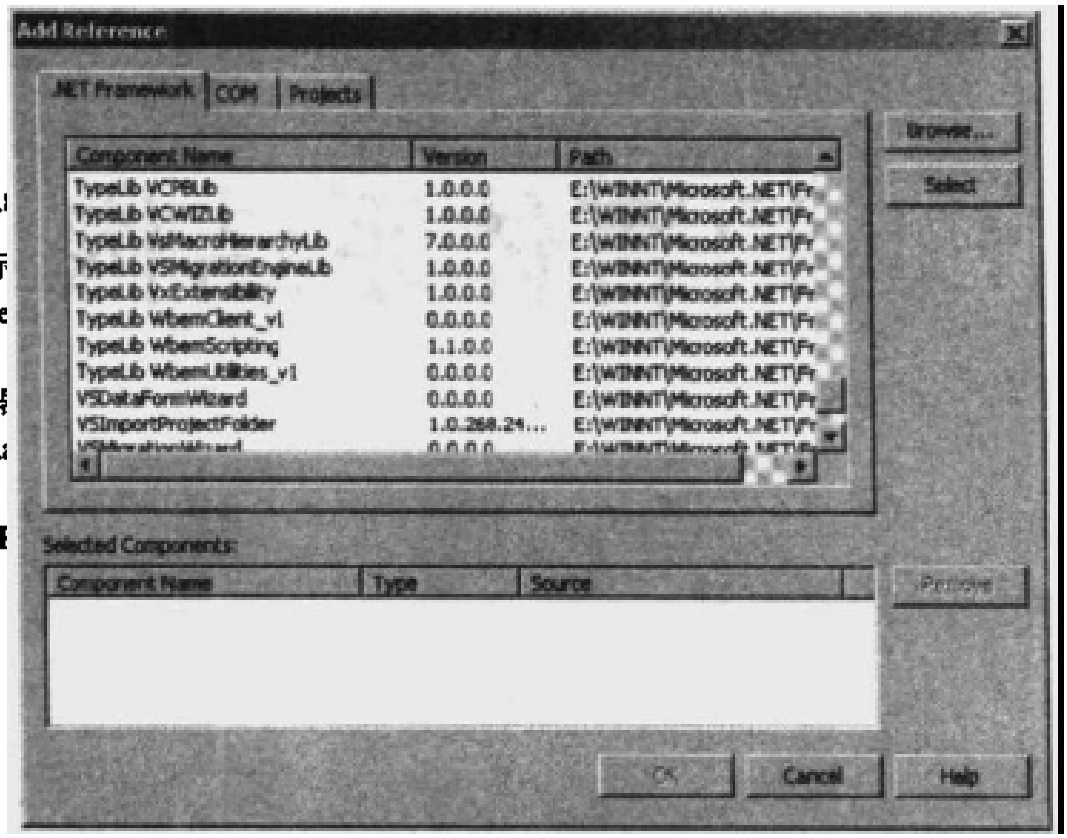


图 2.

图 2.9 所示

（1）Addre

址；

（2）浏览

（3）Avail

息；

（4）Add

第 3 章　Visual Basic.NET 语言体系结构

本章包括:

Visual Basic.NET 的数据类型
Visual Basic.NET 的运算
Visual Basic.NET 的常量和变量
Visual Basic.NET 的流程和控制结构
Visual Basic.NET 的过程和函数
Visual Basic.NET 的类与对象基础

3.1　Visual Basic.NET 的数据类型

本节将重点介绍 Visual Basic.NET 的数据类型结构，数据可以说是程序的基础，决定着计算机如何存储变量。计算机在为变量分配空间时，首先就要根据变量的类型来确定内存的分配，在编译系统语义分析时，也要检查数据与变量的类型是否匹配，如果不匹配，可能会造成空间的浪费，甚至编译的失败，因此数据类型对编程的意义非常重要。下面将主要介绍 Visual Basic.NET 的基本数据类型。

3.1.1　Numeric 数据类型

Visual Basic.NET 支持更多种类的 Numeric 数据类型，这些类型包括："Integer"（整型）、"Long"（长整型）、"Single"（单精度浮点型）、"Double"（双精度浮点型）以及新增的"Decimal"（十进制型）和"Short"（短整型），但是少了"Current"（货币型）。

如果知道变量总是存放整数（如 12）而不是带小数点的数字（如 3.57），就应当将它声明为"Integer"类型、"Long"类型或"Short"。整数的运算速度较快，而且比其他数据类型占据的内存要少。在 For...Next 循环内作为计数器变量使用时，整数类型尤为有用。"Decimal"数据类型可以表示 28 位十进制数，而小数点的位置可根据数的范围及精度要求而定。"Decimal"数据类型不但取代了 Visual Basic 以前版本的"Current"类型，而且是 Visual Studio.net 构架中的通用数据类型，浮点（"Single"和"Double"）数比"Decimal"类型的有效范围大得多，但有可能产生小的进位误差。

注意：浮点数值可表示为"mmmEeee"，其中"mmm"是尾数，而"eee"是指数（以 10 为底的幂）。"Single"数据类型的最大正数值为 3.402823E+38，或 3.4 乘以 10 的 38 次方；"Double"数据类型的最大正数值是 1.79769313486232E+308 或 1.8 乘以 10 的 308 次方。表 3.1 对数字型的数据类型进行了归纳。

表 3.1　Visual Basic.NET 的 Numeric 数据类型

类型	.NET 的运行时的结构	占用空间	描　述
Integer	System.Int32	4 个字节	变量存储为 32 位整数型，范围是从－2,147,483,648 到 2,147,483,648
Long	System..Int64	8 个字节	变量存储为 64 位整数型，范围是从－9,223,372,036,854,775,808 到 9,223,372,036,854,775,807
Short	System.Int16	2 个字节	变量存储为 16 位整数型，范围是从－32,768 到 32,767
Single	System.Single	4 个字节	变量存储为 32 位浮点数值形式，范围：负数为－3.402823E38 到－1.401298E-45；正数为 1.401298E－45 到 3.402823E38
Double	System.Double	8 个字节	变量存储为 64 位浮点数值形式，范围：负数为－1.79769313486231E308 到－4.94065645841247E-324 正数为 4.94065645841247E-324 到 1.79769313486231E308
Decimal	System.Decimal	12 个字节	无小数点的整数范围是－79228162514264337593543950335 到 79228162514264337593543950335，绝对值最小的非零数是+/－0.0000000000000000000000000001

3.1.2　Byte 数据类型

如果变量包含二进制数，则将它声明为“Byte”数据类型的数组。在转换格式期间用“Byte”变量存储二进制数据就可保留数据。当“String”类型变量在“ANSI”和“Unicode”格式间进行转换时，变量中的任何二进制数据都会遭到破坏。在下列任何一种情况下，Visual Basic 都会自动在“ANSI”和“Unicode”之间进行转换：

（1）读文件时

（2）写文件时

（3）调用 DLL 时

（4）调用对象的方法和属性时

注意：除一元减法之外，所有可对整数进行的运算符均可操作“Byte”数据类型，因为“Byte”类型在 Visual Basic 中占的存储位数是 8 位，表示的是从 0~255 的无符号整数类型，不能表示负数。因此，在进行一元减法运算时，Visual Basic 首先将“Byte”转换为符号整数。

“Byte”数据类型可以转换成“Integer”类型、“Long”类型、“Short”类型、“Single”、“Double”、“Decimal”，不会出现溢出的错误。

3.1.3　String 数据类型

如果变量总是包含字符串而从不包含数值，就可将其声明为“String”类型。一个字符串可以包含大约 2 亿（2 的 31 次方）的“Unicode”字符，字符的机内码从 0 到 65535，前 128 个字符代表英文字母以及标准的键盘上的字符，128 到 255 个字符代表特殊的字符，比如拉丁字母、货币符号、分数等。剩下的字符编码是留给范围更广的字符的。声明一个字符串变量可以按照下面的语法：

```
Dim S As String
```

然后可将字符串赋予这个变量，并用字符串函数对它进行操作。

S = "Database"

按照缺省规定，“String”类型变量或参数是一个可变长度的字符串。随着对字符串赋予新数据，它的长度可增可减。

对 String 类型操作的基本函数:

（1）Len 函数

“Len”函数返回字符串的长度，“Len”的返回值是长整型，它的语法是：

Len(string | varname)

Len 函数的参数说明：string 为任何有效的字符串表达式；varname 为任何有效的变量名称。如果 varname 是 Object，Len 会视其为 String 并且总是返回其包含的字符数，例：

```
Dim MyString As String
Dim MyLen As Integer
MyString = "Hello World"          '初始化字符串
MyLen = Len(MyString)             '返回 11
```

（2）Trim、LTrim 及 RTrim

Trim、RTrim 及 LTrim 函数完成将字符串中的一些或全部空格去掉，Trim 将字符串中全部字符去掉，LTrim 将字符串起始的空格都去掉，RTrim 将字符串末尾的空格都去掉，例：

```
Dim MyString , TrimString As String
MyString = "     <-Trim->     "    '初始化字符串
TrimString = Trim(Mystring)       'TrimString = "<-Trim->"
TrimString = LTrim(MyString)      'TrimString = "<-Trim->      "
TrimString = RTrim(MyString)      'TrimString = "     <-Trim->"
```

（3）Substring 方法

Substring 方法取代以前 Visual Basic 版本的“Right”、“Left”以及“Mid”等标准函数，它的用法是：

```
StrName.Substring( startChar , Length)
```

例如：

```
s.Substring(0,2)                  '相当于 Left(s, 2)
s.Substring(s.Length() ,- 4)      '相当于 Right(s,4)
```

3.1.4 Boolean 数据类型

若变量的值只是“true/false”、“yes/no”、“on/off”信息，则可将它声明为 Boolean 类型。尽管 Boolean 仅存储“True”或“False”，但是仍占用 4 个字节。如果其他 Numeric 类型转换成 Boolean 类型，则“0”转换成“False”，其他的非零数转换成“True”，Boolean 的缺省值为“False”。在下面的例子中，“blnRunning”是 Boolean 变量，存储简单的“yes/no”设置。

```
Dim blnRunning As Boolean         '查看磁带是否在转。
If Recorder.Direction = 1 Then
    blnRunning = True
End if
```

```
While blnRunning
    ……
While End
```

3.1.5　Date 数据类型

Date（日期）和 Time（时间）可包含在 Date 数据类型中，Date 类型的变量存储在 64 位（8 个字节）的长整形中，代表的时间从公元 1 年 1 月 1 日到公元 9999 年 12 月 31 日，表示的时间从 0:00:00 到 23:59:59。

Date 类型的数据要写在两个“#”中间，如“# January 1,1993 #”或“# 1 Jan 93 #”，而且日期和时间的表示方式取决于计算机。

DateAndTime 类：

DateAndTime 类可以返回各种形式的时间信息，常用的属性有“Now”、“Today”等，常用的方法有“Year”、“Month”、“MonthName”、“Weekday”、“WeekdayName”等，如下例，将当前的日期及时间返回给“MyDate”：

```
Dim MyDate As Date
Dim MyWeekdayName As String
MyDate = DateAndTime.Now
MyWeekdayName= DateAndTime.WeekdayName(1) & CStr(MyDate)
```

3.1.6　Object 数据类型

Object 变量作为 32 位（4 个字节）地址来存储，该地址可引用应用程序中或某些其他应用程序中的对象。可以随后指定一个被声明为 Object 的变量去引用应用程序所识别的任何实际对象。Object 变量也可以用来存储各种类型的数据变量，这个功能使“Object”类型取代了 Visual Basic 以前版本的“Variant”类型，如下例：

```
Dim objDb As Object
objDb = New DAO.Field()
```

在声明对象变量时，请使用特定的类，而不用一般的 Object（例如用 TextBox 而不用 Control，或者像上面的例子那样，用“Field”取代 Object）。运行应用程序之前，Visual Basic 可以决定引用特定类型对象的属性和方法。因此，应用程序在运行时速度会更快。在“对象浏览器”中列举了特定的类。当使用其他应用程序的对象，并在“对象浏览器”中的“类”列表中列举对象时，应该声明对象，这样可确保 Visual Basic 能够识别引用的特定类型对象，在运行时解决引用问题。

3.1.7　用户自定义类型

用户自定义类型在 Visual Basic.NET 中叫做“structure”（结构），包含有一个或多个不同种类的数据类型，尽管结构中的数据可以单独被访问，但是这些数据仍被认作是一个集合。在 Visual Basic 以前的版本中用户自定义类型的关键字是“Type”，之所以改成“Structure”，

大概是为了与 C++的语法结构靠近吧。

一个结构的定义以“Structure”关键字开始，以“End Structure”关键字结束，结构中的元素可以是任意的数据类型的组合，包括其他结构。结构一旦定义出来后就可以被用作变量声明、参数传递以及函数的返回值等用途，下面是定义一个结构的语法：

```
[ Public | Private | Protected | ] Structure structname
    {Dim | Public | Private | Friend} member1 As datatype1
    ……
    {Dim | Public | Private | Friend} memberN As datatypeN
End Structure
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

如下例定义一个“Employee”结构：

```
Structure Employee
    Public GivenName As String        ' 雇员的姓
    Public FamilyName As String       ' 雇员的名
    Public Extension As Long          ' 雇员的电话
    Private Salary As Decimal         ' 雇员的收入
End Structure
```

用户自定义数据类型占用内存空间是其包含的所有数据类型所占用内存空间的总和。

3.1.8 数组

1. 数组

数组（Arrays），可以用相同名字引用一系列变量，并用数字（索引）来识别它们。在许多场合，使用数组可以缩短和简化程序，因为可以利用索引值设计一个循环，高效处理多种情况。数组的元素下标是连续的。因为 Visual Basic 对每一个索引值都分配空间，所以不要不切实际声明一个太大的数组。

在 Visual Basic.NET 中，所有的数组都是以“0”为起始长度的，这和 Visual Basic 以前的版本完全不一样，在 Visual Basic 6.0 中，如果写下以下语句：

```
Dim x(10) As Single
```

可以假定 x 数组的元素从 1 到 10 的，但它实际上总是包含第 0 个元素。换句话来说，x 数组实际上是含有 11 个元素。

在 Visual Basic.NET 中，这样的数组含有 10 个元素，编号为 0 到 9，这使得我们在处理数组的长度和编号的情况和 C、C++、C#和 Java 具有一致性，如下例：

```
Dim Max as Integer
Max=10
Dim x(Max)
For j = 0   to Max-1
      X(j) = j
Next j
```

从上面的例子，可以看出，数组的最后一个元素的编号，总是数组长度减去 1。

2. 数组列表（ArrayLists）

除了现在数组的长度计数是基于 0 开始的这个变化外，Visual Basic.NET 还引进了一个数组列表（ArrayList）对象代替原来的集合（Collection）对象，集合对象的长度计数总是从 1 开始的，而且在需要的时候，数组列表长度可以是不定的。数组列表的基本的方法和集合一样，只不过它还具有一些新的功能方法。

```
Dim Arl as New ArrayList   '构建一个数组列表
For j = 0 to 10
    Arl.Add(j)
Next j
```

超星阅览器提醒您：
此为试读，需要完整PDF请访问：

所有的数组变量都有一个长度属性，这样就可以得知这个数组有多大：

```
Dim z(20) As single
Dim j As Integer
For j = 0 To z.Length –1
    Z(j) =j
Next j
```

在 Visual Basic.NET 中所有的数组都是动态的，可以在任何时候重新定义数组的长度，然而，在 Visual Basic.NET 中已经没有 ReDim Preserve 表述了，可以使用 New 关键字来对任何一个数组进行引用，并且重新定位：

```
'在类模块级声明
    Dim x() As Single
'重新定位
    x = New Single (20) { }
```

注意：在数组类型后面的大括号不要忘记写。

和集合对象一样，数组列表含有一个“Count”属性和一个“Item”属性允许使用“index”来访问数组列表中的元素。而且，和集合对象一样，这个属性也可以省略，感觉就像操作数组一样：

```
For i = 0 to ar.Count –1
    Console.WriteLine(ar.Item(i))
    Console.writeLine(ar(i))
Next I
```

可以使用如表 3.2 列出的一些数组列表的方法：

表 3.2 数组列表的方法

Clear	把数组列表的内容清空
Contains(object)	如果数组列表含有该对象则返回 true
CopyTo(array)	把一个数组列表的拷贝到一个一维的数组中去
IndexOf(object)	返回第一个元素的值
Insert(index,object)	在指定的位置插入一个元素
Remove(object)	把一个元素从列表中删除
RemoveAt(index)	把一个指定位置的元素从列表中删除
Sort	对列表进行排序

注意：这一部分讨论的数组是程序中声明的变量数组。它们不同于控件数组，控件数组是在设计时通过设置控件的“Index”属性规定的。变量数组总是连续的；与控件数组不同的是，不能从一个数组的中部加载或卸载数组元素。一个数组中的所有元素具有相同的数据类型。当然，当数据类型为“Object”时，各个元素能够包含不同种类的数据（对象、字符串、数值等）。可以声明任何基本数据类型的数组，包括用户定义的类型和对象变量。此外，数组在 Visual Basic 中已经是完全的动态的了。

3. 多维数组

多维数组可以理解成为“数组的数组”，比一维数组多了“维数”的概念。可以用多维数组记录复杂的信息。例如，为了追踪记录计算机屏幕上的每一个像素，需要引用它的 X、Y 坐标。这时应该用多维数组存储值。可用 Visual Basic 声明多维数组。

声明多维数组可以使用下面的语法：

Dim ArrayName(VarNumber1,VarNumber2,...,VarNumbern) As Type

可以将一维数组所有规则推广到二维以上的数组。例如，下面的语句声明了一个 10×10 的二维数组以及 4×10×15 的三维数组：

```
Dim MatrixA (10. 10) As Double
Dim MultiD (4, 10, 15) As Single
```

元素总数为各个维的维数的乘积，即为 100、600。

注意：在增加数组的维数时，数组所占的存储空间会大幅度增加，所以要慎用多维数组。使用“Object”数组时更要格外小心，因为它们需要更大的存储空间。用循环操作数组可以用“For”循环嵌套来有效地处理多维数组。例如，在“MatrixA”中基于每个元素在数组中的位置为其赋值：

```
Dim I, J As Integer
Dim MatrixA(10, 10) As Double
For I = 0 To 9
    For J = 0 To 9
        MatrixA (I, J) = I * 10 + J
    Next
Next
```

4. 保留动态数组的内容

每次执行 ReDim 语句时，当前存储在数组中的值都会全部丢失。Visual Basic 重新将数组元素的值置为“Nothing”（对 Object 数组）、置为 0（对 Numeric 数组）、置为零长度字符串（对 String 数组）。在为新数据准备数组，或者要缩减数组大小以节省内存时，这样做是非常有用的。有时希望改变数组大小又不丢失数组中的数据。使用具有 Preserve 关键字的 ReDim 语句就可做到这点。例如，使用 UBound 函数引用上界，使数组扩大、增加一个元素，而现有元素的值并未丢失：

```
ReDim Preserve MyArray (UBound (MyArray) + 10)
```

在用 Preserve 关键字时，只能改变多维数组中最后一维的上界；如果改变了其他维或最后一维的下界，那么运行时就会出错。所以可这样编程：

```
ReDim Preserve Matrix (10, UBound (Matrix, 2) + 1)
```

而不可这样编程：

ReDim Preserve Matrix (UBound (Matrix, 1) + 1, 10)

注意：UBound 函数可以获得数组某一维的元素个数减去 1。

5. 类型转换函数

在程序设计的过程中经常会遇到不同数据类型之间的转换，比如说要将“Date”类型显示在文本框中，就必须先进行类型转换，使其转换成“String”类型。虽然 Visual Basic 本身能进行一定的类型转换，但是这种转换是很有限制的，而且这种隐含的类型转换存在不稳定的因素，对程序的可读性、可维护性都有一定的影响，因此 Visual Basic 就提供了一定的类型转换函数。例如用“CStr”函数将操作数转换成“String”：

使用本复制品
请尊重相关知识产权!

strDate = CStr(DateAndTime.Now)

表 3.3 给出了常用的类型转换函数。

表 3.3　常用的类型转换函数

转换函数	返回值	对表达式的要求
CBool	Boolean	任何字符串及数字类型
CByte	Byte	0 到 255
CChar	Char	0 到 65535
CDate	Date	任何合法的日期及时间
CDbl	Double	负数的值从－1.79769313486231E308 到－4.94065645841247E-324；正数的值从 4.94065645841247E-324 到 1.79769313486231E308
CDec	Decimal	无小数点的整数范围是－79,228,162,514,264,337,593,543,950,335 到 79,228,162,514,264,337,593,543,950,335，绝对值最小的非零数是 +/-0.0000000000000000000000000001
CInt	Integer	－2,147,483,648 到 2,147,483,647；允许分数，小数部分采取四舍五入的原则
CLong	Long	－9,223,372,036,854,775,808 到 9,223,372,036,854,775,807；允许分数，小数部分采取四舍五入的原则
CObj	Object	任何合法的表达式
CShort	Short	－32,768 到 32,767；允许分数，小数部分采取四舍五入的原则
CSng	Single	负数从－3.402823E38 到－1.401298E-45；正数从 1.401298E－45 到 3.402823E38
CStr	String	可以是“Boolean”类型，返回“True”或“False”；也可以是“Date”类型，返回日期的缩写形式；或者是数字类型，返回代表这个数字的字符串

注意：类型转换时，一定要注意待转换的表达式相对函数的返回值是否有效，比如“String”里有非数字的字符就不能转换成“Integer”类型，而“Long”类型转换成“Integer”时，“Long”型数则必须在“Integer”数据类型的范围内。

3.2　Visual Basic.NET 的运算

在程序设计时经常用到的有 6 种运算：算术（Arithmetic）运算、赋值（Assignment）运算、二进制（Bitwise）运算、比较（Comparison）运算、连接（Concatenation）运算以及

逻辑（Logical）运算。每种运算在不同的方面都起到一定的作用，下面对这 6 种运算进行简单的说明。

3.2.1 算术运算

算术运算是高级语言所要实现的最基本的功能，算术运算也就是通常所说的数学运算，在 Visual Basic.NET 中新增了一些算术运算符，这些新增的算术运算符是由基本运算符所组成的。表 3.4 列出了这些算术运算符。

超星阅览器提醒您：
使用本复制品
[illegible]

表 3.4　算术运算符

运 算 符	名　　称	类　　型	说　　明
+	加号	双目运算符	加号两边的操作数是数字类型时，要注意有没有溢出的可能；此外，“Single”类型与“Long”类型相加时，返回值为“Double”类型，如果两个操作数都为“Empty”，则返回值为“Integer”；一个是“Empty”，另一个不是时，另一个操作数即是返回值
–	减号或负号	双目运算符或单目运算符	做双目运算符时同加号。单目运算时，表示一个数的相反数
*	乘号	双目运算符	同加号
/	浮点除号	双目运算符	操作数同时为“Byte”、“Integer”或“Single”，返回值在不溢出时为“Single”或“Double”
\	整除号	双目运算符	操作数可以是任何类型的数，但在运算时都被取整，返回值一般为“Byte”、“Integer”、“Long”。
MOD	求余号	双目运算符	返回第一个操作数整除第二个操作数的余数，例如 10 MOD 3 结果为 1；12.6 MOD 5 结果为 3
^	求指号	双目运算符	对第一操作数进行连乘，连乘次数为第二个操作数，注意求指号的运算顺序为从右到左，例如 3^3^3 结果为 19683，而（3^3）^3 结果为 729

3.2.2 赋值运算符

表 3.5 列出常用的赋值运算符。

表 3.5　赋值运算符

运 算 符	名　　称	类　　型	说　　明
=	赋值号	双目运算符	第二个操作数值传给第一个操作数
+=	加等号	双目运算符	第一个操作数加上第二个操作数传给第一个操作数
-=	减等号	双目运算符	第一个操作数减去第二个操作数传给第一个操作数
*=	乘等号	双目运算符	第一个操作数乘以第二个操作数传给第一个操作数
/=	浮点除等号	双目运算符	第一个操作数除以第二个操作数传给第一个操作数
\=	整除等号	双目运算符	第一个操作数整除第二个操作数传给第一个操作数
^=	求指等号	双目运算符	第一个操作数连乘第二个操作数次传给第一个操作数
&=	连接等号	双目运算符	第一个操作数连接第二个操作数传给第一个操作数

赋值运算符大部分是 Visual Basic.NET 新增的运算符，除“=”（赋值号）外，其他赋值运算符在给变量赋值的时候，一般都先进行算术运算，如 i+=1 相当于 i=i+1。

3.2.3　二进制运算符

操作数无论是什么类型，在内存中都是以二进制的形式存储的，如“Byte”类型的数 255 在内存中的存储形式为“11111111”；“Short”类型的数 4 在内存中的存储形式为“0000000000000100”。而二进制运算就是对这些二进制数进行操作的运算，二进制运算应用很广泛，比如说网络中的子网掩码的算法就是利用二进制运算。表 3.6 列出了二进制运算的基本运算符。

表 3.6　二进制运算符

运 算 符	名　称	类　型	说　明
BitAnd	按位与	双目运算符	0 BitAnd 0 值为 0 0 BitAnd 1 值为 0 1 BitAnd 0 值为 0 1 BitAnd 1 值为 1
BitNot	按位非	单目运算符	BitNot 0 值为 1 BitNot 1 值为 0
BitOr	按位或	双目运算符	0 BitOr 0 值为 0 0 BitOr 1 值为 1 1 BitOr 0 值为 1 1 BitOr 1 值为 1
BitXor	按位异或	双目运算符	0 BitXor 0 值为 0 0 BitXor 1 值为 1 1 BitXor 0 值为 1 1 BitXor 1 值为 1

3.2.4　比较运算符

比较运算符就是比较大小，结果为“True”或“False”，如果操作数包含“Empty”，则按“0”进行处理。常用的比较运算符如表 3.7 所示。

表 3.7　常用比较运算符

运 算 符	名　称	类　型	说　明
<	小于	双目运算符	操作数可以是任何合理的表达式
<=	小于等于	双目运算符	同“小于”号
>	大于	双目运算符	同“小于”号
>=	大于等于	双目运算符	同“小于”号
=	等于	双目运算符	同“小于”号
<>	不等于	双目运算符	同“小于”号

此外，Visual Basic.NET 还有两种比较运算符：“Is”以及“Like”。

1. Is 运算符

Is 运算符的操作数要求是“Object”类型，如果两个操作数表示同一个对象，那么返回“True”，反之则返回“False”，例如：

```
Dim MyObject, YourObject, ThisObject, OtherObject, ThatObject as Object
Dim MyCheck as Boolean
YourObject = MyObject
```

超星阅览器提示
使用本复制品
请尊重相关知识产权!

```
ThisObject = MyObject
ThatObject = OtherObject
MyCheck = YourObject Is ThisObject    ' 返回 True
MyCheck = ThatObject Is ThisObject    ' 返回 False
MyCheck = MyObject Is ThatObject      ' 返回 False
```

2. Like 运算符

Like 的第一操作数要求是“String”类型，第二操作数要求是“String”或字符串的标准样式。字符串的标准样式主要由 5 点组成：

（1）“?”代表单个字符

（2）“*”代表 0 或多个字符

（3）“#”代表 0—9 的单个数字

（4）[字符列表]代表任何在列表中的字符

（5）[!字符列表]代表任何不在列表中的字符

下面的例子说明了“Like”的用法：

```
Dim MyCheck as String
MyCheck = "aBBBa" Like "a*a"              ' 返回 True.
MyCheck = "F" Like "[A-Z]"                ' 返回 True.
MyCheck = "F" Like "[!A-Z]"               ' 返回 False.
MyCheck = "a2a" Like "a#a"                ' 返回 True.
MyCheck = "aM5b" Like "a[L-P]#[!c-e]"     ' 返回 True.
MyCheck = "BAT123khg" Like "B?T*"         ' 返回 True.
MyCheck = "CAT123khg" Like "B?T*"         ' 返回 False.
```

3.2.5　连接运算符

连接运算就是将两个表达式连接在一起，运算符有“&”和“+”，如果操作数是“String”类型，那么这两个运算符完成的功能是一样的，如表 3.8 所示。

表 3.8　连接运算符

运 算 符	名　　称	类　　型	说　　明
+	加连接符	双目运算符	如果两个操作数为“String”，就把这两个操作数连接成一个字符串，否则按加号处理
&	连接符	双目运算符	将操作数强行转换成“String”，再进行连接运算

3.2.6　逻辑运算符

逻辑运算通常用来表示复杂的关系。例如，有这样的条件：如果 x 大于或等于零，则对 x 取平方根，在 Visual Basic.NET 中可以表示为：

```
If x > 0 OR x = 0 Then
    x = Math.Sqrt(x)
End If
```

超星阅览器版权所有：
使用本复制品
请尊重相关知识产权！

逻辑运算符如表 3.9 所示。

表 3.9　逻辑运算

运 算 符	名　称	类　型	说　明
AND	与	双目运算符	A AND B 表示 A 与 B，A、B 都为“True”时，结果为“True”，否则为“False”
NOT	非	单目运算符	NOT A 表示非 A，A 为 1 时，结果为 0，A 为 0 时，结果为 1
OR	或	双目运算符	A OR B 表示 A 或 B，A、B 都为“False”时，结果为“False”，否则为“True”
XOR	异或	双目运算符	A XOR B 表示 A 异或 B，A、B 不相等时，结果为“True”，否则为“False”

3.2.7　运算符的优先级

在每个表达式中进行操作时，每一部分的运算都要按着一定的顺序进行计算，这个顺序就是运算的优先顺序，也叫运算符的优先级。

在表达式中，若运算符不只一种时，首先处理算术运算符，然后是比较运算符，最后是逻辑运算符。同一种运算符中，单目运算符的优先级高于双目运算符。优先级相同的运算符，按从左到右的顺序进行处理（求指运算除外）。

字符串连接运算符（&）不是运算符，但就其优先级而言，它排在所有算术运算符之后，以及所有比较运算符之前。

还可以使用括号来改变有限顺序，强制命令表达式中的某些部分优先执行。在括号内部的符号的优先级高于括号外边的运算符，同一个括号内的运算符优先级不变，使用括号既省去了死记运算符优先级的麻烦，而且有利于程序的可读性和可维护性。表 3.10 列出了各运算符的优先顺序。

表 3.10　运算符的优先顺序

算术、二进制及连接运算符	比较运算符	逻辑运算符
求指号（^）	等号（=）	Not
负号（－）	不等号（<>）	And
乘除号（*、/）	小于（<）	Or
整除号（\）	大于（>）	Xor
求余号（MOD）	小于等于（<=）	
加减号（+、－）	大于等于（>=）	
二进制运算符（BitNot，BitAnd，BitOr，BitXor）	Like，Is	
连接运算符（&）		

3.3　Visual Basic.NET 的常量和变量

3.3.1　常量

在编程时，会遇到这种情况：代码包含一些常数值，它们反复出现。而且代码要用到

很难记住的数字，而那些数字没有明确意义。在这些情况下，可用常数来方便地改进代码的可读性和可维护性。常数是用意义的名字取代经常用到的数值或字符串。尽管常数有点像变量，但不能像对变量那样对其值进行修改，也就是说对常量赋初值后就不能再次赋值了。常数有两种来源：

（1）内部的或系统定义的常数是由应用程序和控件提供的。这些常数是在 Visual Basic 对象库中定义的。

（2）用户定义的常数是用 Const 语句来声明的。

来自 Visual Basic 对象库的常数由以下形式构成：

NameSpace1.NameSpace2.….ConstName，例如：

Microsoft.VisualBasic.MsgBoxStyle.OKOnly

是 Visual Basic 对话框中的一个按钮常数，值为 0。

1. 自定义常数

声明常数的语法是：

[Public | Private | Protected | Friend | Protected Friend] Const constname [As type] = expression

参数“constname”是有效的符号名，“expression”由数值常数或字符串常数以及运算符组成；但在“expression”中不能使用函数调用。“Const”语句可表示数量、日期和时间：

Const conPi = 3.14159265358979

Public Const conMaxPlanets As Integer = 9

Const conReleaseDate = #1/1/95#

也可用 Const 语句定义字符串常数：

Public Const conVersion = "07.10.A"

Const conCodeName = "Enigma"

如果用逗号进行分隔，则在一行中可放置多个常数声明：

Public Const conPi = 3.14, conMaxPlanets = 9, _conWorldPop = 6E+09

等号左边必须是左值，等号右边的表达式通常是数字或文字串，但也可以是其结果为数或字符串的表达式（尽管表达式不能包含函数调用），甚至可用先前定义过的常数定义新常数。

Const conPi2 = conPi * 2 当定义常数后，就可将其放置在代码中，使代码更可读。例如：

Const conPi=3.14

Area=conPi * dblR ^ 2

2. 避免循环引用

由于常数可以用其他常数定义，因此必须小心，在两个以上常数之间不要出现循环或循环引用。当程序中有两个以上的公用常数，而且每个公用常数都用另一个去定义时就会出现循环。例如：

'在 Module1 中：

Public Const conA = conB * 2　　' 在整个应用程序中有效。

'在 Module2 中：

Public Const conB = conA / 2　　' 在整个应用程序中有效。

如果出现循环，在试图运行此应用程序时，Visual Basic 就会产生错误信息。不解决循

环引用就不能运行程序。为避免出现循环，可将公共常数限制在单一模块内，或最多只存在于少数几个模块内。

超星阅览器提醒您：
使用本复制品，
请尊重相关知识产权！

3.3.2　变量

在 Visual Basic 中执行应用程序期间，用变量临时存储数值。变量具有名字（用来引用变量所包含的值的词）和数据类型（确定变量能够存储的数据的种类）。可以把变量看作内存中存放未知值的所在处。例如，假定正在为水果铺编一个销售苹果的程序。在销售实际发生之前并不知道苹果的价格和销量。此时，可以设计两个变量来保存未知数，将它们命名为“ApplePrice”和“ApplesSold”。每次运行程序时，用户就这两个变量提供具体值。为了计算总的销售额，并且将结果显示在名叫“txtSales”的文本框中，代码应该是这样的：

```
txtSales.txt = ApplePrice * ApplesSold
```

每次根据用户提供的数值，这个表达式返回不同的金额。由于有了变量，就可以设计一个算式，而不必事先知道实际的输入是多少。在这个例子中，“ApplePrice”的数据类型是十进制型，而“ApplesSold”的数据类型是整数。变量还可以表示许多其他数值，比如：文本数值、日期、各种数值类型，甚至对象也包括在内。

1. 存储和检索变量中的数据

用赋值语句进行计算，并将结果赋予变量：

```
ApplesSold = 10                    '将值 10 传给变量。
ApplesSold = ApplesSold + 1        '变量值增一。
```

注意：例子中的等号是赋值符，并不是等于操作符；它将数值 10 赋予变量 ApplesSold。

2. 声明变量

声明变量就是事先将变量通知程序。要用“Dim”语句声明变量，“Dim”语句提供了变量名：

```
Dim Variablename As Type
```

在过程内部用“Dim”语句声明的变量，只有在该过程执行时才存在。过程一结束，存储该变量的内存空间也就释放了。此外，过程中的变量值对过程来说是局部的，也就是说，无法在一个过程中访问另一个过程中的变量。由于这些特点，在不同过程中就可使用相同的变量名，而不必担心冲突发生以及编译时出现错误。

变量名有以下命名原则：

- 必须以字母开头。
- 不能包含嵌入的（英文）句号或者嵌入的类型声明字符。
- 不得超过 255 个字符。
- 在同一个范围内必须是惟一的。

范围就是可以引用变量的域，如一个过程、一个函数等。由于“Dim”语句中的可选的“As Type”子句，可以定义被声明变量的数据类型或对象类型。数据类型定义了变量所存储信息的类型。变量也可以包含来自 Visual Basic 或其他应用程序的对象，如“Form”和“TextBox”等。

注意：“As Type”在默认的情况下是必写的，如果要将“As Type”变成可选的，则需将工程属性页中的“Build”选项中“Option Strict”设置为“Off”，这样在没有“As Type”

的变量声明中，“Object”类型是其默认的数据类型。

（1）隐式声明

在工程属性页中，将“Build”选项中的“Option Explicit”选项设置为“Off”，则工程中便允许隐式声明，即在使用一个变量之前并不必先声明这个变量。例如，可以书写这样一个函数，在其中就不必在使用变量 TempVal 之前先声明它：

```
Function SafeSqr (num)
    TempVal = Abs (num)
    SafeSqr = Sqr (TempVal)
End Function
```

Visual Basic 用这个名字自动创建一个变量，使用这个变量时，可以认为它就是声明过的。虽然这种方法很方便，但是如果把变量名拼错了的话，会导致一个难以查找的错误。例如，假定这个函数写成：

```
Function SafeSqr (num)
    TempVal = Abs (num)
    SafeSqr = Sqr (TemVal)
End Function
```

看起来，这两段代码好像是一样的。但是因为在倒数第二行把“TempVal”变量名写错了，所以函数总是返回 0。当 Visual Basic 遇到新名字，它分辨不出这是变量名写错了，于是用这个名字再创建一个新变量。

（2）显式声明

为了避免写错变量名引起的麻烦，可以规定，只要遇到一个未经明确声明就当成变量的名字，Visual Basic 都发出错误警告。要显式声明变量，只需将工程属性页中的“Build”选项中的“Option Explicit”设置为“On”即可，如果对包含“SafeSqr”函数的窗体或标准模块执行该语句，那么 Visual Basic 将认定“TempVal”和“TemVal”都是未经声明变量，并为两者都发出错误信息。随后就可以显式声明“TempVal”：

```
Function SafeSqr (num)
    Dim TempVal
    TempVal = Abs (num)
    SafeSqr = Sqr (TemVal)
End Function
```

因为 Visual Basic 对拼错了的“TemVal”显示错误信息，所以能够立刻明白是什么问题。由于“Option Explicit”有助于编译系统的出错处理，所以一般来说在编写代码之前要将其设置为“On”。

3. 理解变量的范围

变量的范围确定了能够访问该变量存在的那部分代码。在一个过程内部声明变量时，只有过程内部的代码才能访问或改变那个变量的值；它有一个范围，对该过程来说是局部的。但是，有时需要使用具有更大范围的变量，例如这样一个变量，其值对于同一模块内的所有过程都有效，甚至对于整个应用程序的所有过程都有效。Visual Basic 允许在声明变量时指定它的范围。

指定变量的有效范围：一个变量在划定范围时被看作是过程级（局部）变量，还是模

块级变量，这取决于声明该变量时采用的方式。范围专用公用过程级变量对于这种过程是专用的，在该过程中出现了这些变量不可使用。不能在一个过程中声明公用变量模块级变量对于这种过程是专用的，在该过程中出现了这些变量可用于所有模块

超星阅览器提醒您：
此为试读，需要完整PDF请访问：

4. 过程内部使用的变量

过程级变量只有在声明它们的过程中才能被识别，它们又称为局部变量。用“Dim”关键字来声明它们。例如：

```
Dim intTemp As Integer
```

在整个应用程序运行时，用“Dim”声明的变量只在过程执行期间才存在。对任何临时计算来说，局部变量是最佳选择。例如，可以建立十几个不同的过程，每个过程都包含称作“intTemp”的变量。只要每个“intTemp”都声明为局部变量，那么每个过程只识别它自己的“intTemp”版本。任何一个过程都能够改变它自己的局部的“intTemp”变量的值，而不会影响别的过程中的“intTemp”变量。

5. 模块内部使用的变量

按照缺省规定，模块级变量对该模块的所有过程都可用，但对其他模块的代码不可用。可在模块顶部的声明段用 Private 关键字声明模块级变量，从而建立模块级变量。例如：

```
Private intTemp As Integer
```

在模块级，“Private”和“Dim”之间没有什么区别，但“Private”更好些，因为很容易把它和“Public”区别开来，使代码更容易理解。所有模块使用的变量为了使模块级的变量在其他模块中也有效，用“Public”关键字声明变量。公用变量中的值可用于应用程序的所有过程。和所有模块级变量一样，也在模块顶部的声明段来声明公用变量。例如：

```
Public intTemp As Integer
```

注意：不能在过程中声明公变量，只能在模块的声明段中声明公用变量。

6. 静态变量

变量有其生存的周期，在这一期间变量能够保持它们的值。在应用程序的存活期内一直保持模块级变量和公用变量的值。但是，对于“Dim”声明的局部变量以及声明局部变量的过程，仅当过程在执行时这些局部变量才存在。通常，当一个过程执行完毕，它的局部变量的值就已经不存在，而且变量所占据的内存也被释放。当下一次执行该过程时，它的所有局部变量将重新初始化。但可将局部变量定义成静态的，从而保留变量的值。在过程内部用“Shared”关键字声明一个或多个变量，其用法和“Dim”语句完全一样：

```
Shared Depth
```

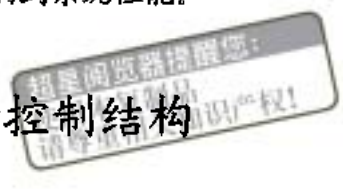
例如，下面的函数将存储在静态变量“Accumulate”中的以前的运营总值与一个新值相加，以计算运营总值。

```
Function RunningTotal (num)
    Shared ApplesSold As Integer
    ApplesSold = ApplesSold + num
    RunningTotal = ApplesSold
End Function
```

如果用“Dim”，而不用“Shared”声明“ApplesSold”，则以前的累计值不会通过调用函数保留下来，函数只会简单地返回调用它的那个相同值。在模块的声明段声明“ApplesSold”，并使它成为模块级变量，由此也会收到同样效果。但是，这种方法一旦改

变变量的范围，过程就不再对变量排他性存取。由于其他过程也可以访问和改变变量的值，所以运营总值也许不可靠，代码将更难于维护。

注意：Visual Basic 以前版本是用“Static”关键字来声明变量，在 Visual Basic.NET 中“Shared”关键字取代了“Static”关键字，静态变量要慎用，因为一旦声明了静态变量，这个变量就会常驻内存，如果声明的静态变量很多，有可能影响到系统性能。



3.4 Visual Basic.NET 的流程和控制结构

3.4.1 条件分支结构

1. 简单 If 条件语句

使用在称之为分支结构的特殊语句块中的条件表达式控制了程序中哪些语句被执行以及以什么样的次序执行。“If ... Then”分支结构在程序中计算条件值，并根据条件值决定下一步执行的操作。最简单的“If ... Then”分支结构可以只写在一行：

If Condition Then Statement [Else statement]

注意：“If...Then”分支结构用于给程序添加逻辑控制能力。这里，“Condition”是个条件表达式，“Statement”是条有效的 Visual Basic 语句。例如：

If Score>= 20 Then Label1.Text = "You Win!" 是个使用了下述条件表达式的分支结构：Score >= 20

根据这个表达式的值，程序决定是否把 Label1 对象的“Text”属性设置为“You Win!”。如果“Score”变量的值大于等于 20，Visual Basic 设置该属性的值，否则，Visual Basic 跳过这条赋值语句，然后执行事件过程中的下一行语句。这类比较运算的结果不是“True”就是“False”，条件表达式从来不会产生模棱两可的值。

2. If ...Then...Else 语句

Visual Basic 还支持另一种格式的“If ... Then”分支结构，该结构中包含几个条件表达式，由多行语句组成，其中包含了重要关键字“ElseIf”、“Else”以及“End If”。

```
If Condition1 Then
    Statements
ElseIf Condition2 Then
    Statements
[其他 ElseIf 子句及其相应的执行语句]
Else
    Statements
End If
```

这个结构中，“Condition1”首先被计算。如果这个条件表达式的值为“True”，那么这个条件表达式下的语句块被执行；如果第一个条件的值不是“True”，那么计算第二个表达式（Condition2）的值，如果第二个条件的值为“True”，那么这个条件表达式下的语句块被执行（如果要判断更多的条件，那么继续增加“ElseIf”子句以及该子句下的语句块）；如果

所有的条件表达式的值都不是“True”，那么执行“Else”子句下的语句块；最后，整个结构使用“End If”关键字结束。多行“If ... Then”结构特别适合于分段计算问题，比如税费方面的计算。下面的代码展示了如何使用多行“If ... Then”结构来确定递进税计算问题（收入和税率的对应关系取自美国国内收入服务 1997 年税率表）：

```
If  AdjustedIncome <= 24650 Then
    '15%税段
    TaxDue = AdjustedIncome * 0.15
ElseIf  AdjustedIncome <= 59750 Then
    '28%税段
    TaxDue = 3697 + (( AdjustedIncome  -  24650 ) * 0.28)
ElseIf  AdjustedIncome <= 124650 Then
    '31%税段
    TaxDue = 13525 + (( AdjustedIncome  -  59750 ) * 0.31)
ElseIf  AdjustedIncome <= 271050 Then
    '36%税段
    TaxDue = 33644 + (( AdjustedIncome  -  124650 ) * 0.36)
Else
    '39.6%税段
    TaxDue = 86348 + (( AdjustedIncome  -  271050 ) * 0.396)
End If
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

注意：总是可以添加更多的“ElseIf”块到“If...Then”结构中去。但是，当每个“ElseIf”都将相同的表达式比作不同的数值时，这个结构编写起来很乏味。在这种情况下可以使用“Select Case”判定结构。

3. Select Case 结构

Visual Basic 还支持在程序中使用“Select Case”分支结构来控制语句的执行。“Select Case”结构与“If ...Then... Else”结构相似，但在处理依赖于某个关键变量或称作测试情况的分支时效率更高。并且，使用“Select Case”结构可以提高程序的可读性。“Select Case”结构的语法如下所示：

```
Select Case  Variable
    Case  Value1
        Statements
    Case  Value2
        Statements
    Case  Value3
        Statements
    ...
End Select
```

“Select Case”结构以关键字“Select Case”开始，以关键字“End Select”结束。“Select Case”结构中的“Variable”可以是变量、属性或表达式，“Value1”、“Value2”、“Value3”可以是数值、字符串或与要测试的其他情况相关的其他值，如果其中某个值与变量相匹配，

那么该“Case”子句下的语句被执行，然后 Visual Basic 执行“End Select”语句后面的语句。Select Case 结构中可以使用任意多个“Case”子句，“Case”子句中也可以包括多个“Value”值，多个“Value”值之间使用逗号分隔。

下面的示例展示了程序中如何使用“Select Case”结构打印与某人年龄相关的信息。当“Age”变量与某个“Case” 值匹配时，相应的信息显示在标签对象中。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
Select Case  Age
Case  16
     Label1.Text = "You can drive now!"
Case  18
     Label1.Text = "You can vote now!"
Case  21
     Label1.Text = "You can drink wine with your meals."
Case  65
     Label1.Text = "Time to retire and have fun!"
End Select
```

注意：“Select Case”结构比功能等效的“If ... Then”结构更清晰易读。“Select Case”结构还支持“Case Else”子句，该子句可用于当不满足所有“Case”条件时显示信息。下面是说明“Case Else”子句用法的一个示例：

```
Select Case  Age
Case  16
     Label1.Text = "You can drive now!"
Case  18
     Label1.Text = "You can vote now!"
Case  21
     Label1.Text = "You can drink wine with your meals."
Case  65
     Label1.Text = "Time to retire and have fun!"
Case  Else
     Label1.Text = "You're a great age!Enjoy it!"
End Select
```

注意：“Select Case”结构每次都要在开始处计算表达式的值。而“If...Then...Else”结构为每个“ElseIf”语句计算不同的表达式。只有在“If”语句和每一个“ElseIf”语句计算相同表达式时，才能用“Select Case”结构替换“If...Then...Else”结构。

3.4.2 循环结构

1. For...Next 循环

“For...Next”循环在事件过程中重复执行指定的一组语句，直到达到指定的执行次数为止。当要执行几个相关的运算、操作屏幕上的多个元素或者处理几段用户输入时，这种方法就十分有用了。“For...Next”循环实际上是一大串程序语句的一种简略写法，由于这一长

串语句中的每一组语句都完成相同的任务，Visual Basic 只定义其中的一组语句并按照程序的需要重复执行这组语句，直至达到规定的次数。“For...Next”循环的语法如下所示：

```
For variable = start To end
    Statements
Next [ variable ]
```

上述语法中，“For”、“To”、“Next”是必须的关键字，等号（=）也不能省略。“variable”是数值型变量的名称，它记录了当前的循环次数，这个变量用你程序中需要的数值型变量来代替。“start”和“end”是两个数值值，表示循环的初值和终值，你也需要使用程序中所需的相应值来代替它们。“For”和“Next”之间的语句一条或多条语句是该循环将重复执行的语句。例如，下面的“For...Next”循环在程序运行时计算机扬声器快速响铃四声：

```
For i=1 To 4
    Beep
Next
```

上面的循环与过程中重复书写 4 条 Beep 语句的功能等价。对编译器来说，上述循环相当于：

```
Beep
Beep
Beep
Beep
```

上面的循环中使用的变量是字母“i”，这是个习惯用法，它表示“For...Next”循环中的整型计数器。循环每执行一次，该计数器变量增加 1（第一次执行循环时，该变量的值为 1，也就是“start”成分指定的初值；最后一次执行循环时，该变量的值为 4，也就是“end”成分指定的终值）。正如下面的示例所展示的，在循环中使用计数器变量是十分方便的。

2. Do 循环

程序中除了使用“For...Next”循环外，也可以使用“Do”循环重复执行一组语句，直到某个条件为“True”时终止循环。对于事先不知道循环要执行多少次的情况来说，“Do”循环十分有用和方便。例如，你要求用户向数据库中输入姓名，直到用户在输入框中键入单词“Done”时终止输入。这时，你可以使用“Do”语句构造一个无穷循环，当用户输入字符串“Done”时退出循环。根据循环条件的放置位置以及计算方式，“Do”循环有几种格式。其中常用的语法格式为：

```
Do [ {While | Until}   condition ]
    Statements
    [Exit Do]
    Statements
Loop
```

或

```
Do
    Statements
    [Exit Do]
    Statements
```

```
Loop [{While | Until} condition]
```

如果条件为“Null”，则这个条件被认为是“False”。例如，下面的“Do”循环重复处理用户输入，直到用户键入单词“Done”时为止：

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
Do While InpName <> "Done"
InpName = InputBox("Enter your name or type Done to quit.")
If InpName <> "Done" Then
    Label1.Text = InpName
End If
Loop
```

注意：测试条件的放置位置影响 Do 循环的执行方式。这个循环中的条件是 InpName <> "Done"，Visual Basic 编译器把这个条件翻译成“只要 InpName 变量的值不等于单词 'Done'，就一直重复执行循环体语句”。这表明：当 Do 循环第一次执行时，如果循环顶部的条件值不是 True，那么 Do 循环中的语句就不会执行。对上面的示例来说，如果在循环开始执行之前（可能在事件过程中使用某个赋值语句进行赋值），InpName 变量的值等于字符串“Done”，那么 Visual Basic 将跳过整个循环体，并在 Loop 语句后面的语句继续执行。需要注意的是，这种格式的 Do 循环需要在循环体中写上一条 If ... Then 语句，以避免用户键入的退出值显示出来。如果希望程序中的循环体至少执行一次，那么把条件放置在循环的尾部，例如：

```
Do Until InpName = InputBox("Enter your name or type Done to quit.")
If InpName <> "Done" Then
    Label1.Text = InpName
Loop While InpName <> "Done"
```

这个循环与前面介绍的 Do 循环相似，但是，这里的循环条件在接收了 InputBox 函数中的姓名后进行测试。这种循环方式的优点是在测试循环条件前更新变量 InpName 的值，这样，即使 InpName 在进入循环前的值为“Done”，也不会直接退出循环。在循环的尾部测试条件值保证了循环体至少执行一次，但是，一般来说，这种格式的循环体中往往要增加一些额外的数据处理语句。

3. While 循环

While 循环执行到给定的条件为“True”才终止循环，与 Do While 相似。While 循环的语法为：

```
While condition
    Statements
    [ Exit While ]
    Statements
End While
```

如果条件为“Null”，则这个条件被认为是“False”，如果条件为“True”，则所有的语句将被执行，直到“End While”，这时候控制权返还给 While，condition 再次被检查，如果 condition 为“True”，则继续执行 While 内部的语句，如果 condition 为“False”，则继续执行 End While 后面的语句，下例将说明 While 循环的用法：

```
Dim Check as Boolean = True
Dim Counter as Integer = 0
```

```
Do          '外层循环
    While Counter <20
        Counter += 1                    '计数器加一
        If Counter = 20 Then
            Check = False
            Exit While                  '退出内层循环
        End If
    End While
Loop Until Check                        '退出外层循环
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

4. For Each...Next 循环

For Each...Next 循环与 For...Next 循环类似，但它对数组或对象集合中的每一个元素重复一组语句，而不是重复语句一定的次数。如果不知道一个集合有多少元素，For Each...Next 循环非常有用。For Each...Next 循环的语法如下：

```
For Each element In group
    Statements
Next element
```

例如，下面的例子利用 For Each...Next 循环来查找所有元素的 Text 属性是否为“Hello”：

```
Dim Found as Boolean
Dim MyObject , MyCollection as Object
For Each MyObject In MyCollection
    If MyObject.Text = "Hello" Then
        Found = True
        Exit For
    End If
Next
```

使用 For Each...Next 时的两点说明：

（1）对于集合或数组中的元素，element 既可以是普通的“Object”类型，也可以是特殊的“Object”对象

（2）Group 可以是数组，也可以是一个集合。

3.5　Visual Basic.NET 的过程和函数

3.5.1　Sub 过程

“Sub”过程是在响应事件时执行的代码块。将模块中的代码分成“Sub”过程后，在应用程序中查找和修改代码变得更容易了。“Sub”过程的语法是：

[Private | Public | Friend] Sub subname ([arguments list])

Statements

End Sub

其中“arguments list”是参数列表，可以像声明变量一样声名参数。

每次调用过程都会执行 Sub 和 End Sub 之间的 statements。可以将子过程放入标准模块、类模块中。按照缺省规定，所有模块中的子过程为 Public（公用的），这意味着在应用程序中可随处调用它们。在 Visual Basic 中应区分通用过程和事件过程这两类子过程。

1. 通用过程

通用过程告诉应用程序如何完成一项指定的任务。一旦确定了通用过程，就必须由专有应用程序来调用。反之，在响应用户引发的事件或系统引发的事件而调用事件过程之前，事件过程通常总是处于空闲状态。建立通用过程就是为了将几个不同的事件过程所要执行的同样语句“提”出来。将公共语句放入一个分离开的过程（通用过程）并由事件过程来调用它，这样一来就不必重复代码，也容易维护应用程序。面向过程的编程思想就是每个事件对应相应的过程，一般说来，过程的大小应在 60~200 行代码之间，如果小于这个范围，就要考虑这个过程是否需要单独提出来，如果大于这个范围，就应当考虑是否应将大的过程细化，一个好的程序风格总会看到其层次关系，也就是过程既有它需要调用的子过程，还有调用它的“父过程”。

2. 事件过程

当 Visual Basic 中的对象对一个事件的发生做出认定时，便自动用相应于事件的名字调用该事件的过程。因为名字在对象和代码之间建立了联系，所以说事件过程是附加在窗体和控件上的。

（1）一个控件的事件过程将控件的实际名字（在 Name 属性中规定的）、下划线 (_) 和事件名组合起来。

例如，如果希望在单击了一个名为 btnPlay 的命令按钮之后，这个按钮会调用事件过程，则要使用 btnPlay_Click 过程。

（2）一个窗体事件过程将窗体的名字空间、下划线和事件名组合起来。如果希望在单击窗体之后，窗体会调用事件过程，则要使用 Form_Click 过程（和控件一样，窗体也有惟一的名字，但不能在事件过程的名字中使用这些名字。）

用户虽然可以自己编写事件过程，但使用 Visual Basic 提供的代码过程会更方便，这个过程自动将正确的过程名包括进来。从“对象框”中选择一个对象，从“过程框”中选择一个过程，就可在“代码编辑器”窗口选择一个模板。在开始为控件编写事件过程之前先设置控件的 Name 属性，这样可以避免编译时产生一定的错误隐患。如果对控件附加一个过程之后又更改控件的名字，那么也必须更改过程的名字，以符合控件的新名字。否则，Visual Basic 无法使控件和过程相符。过程名与控件名不符时，过程就成为通用过程。

3.5.2 Function 过程

Visual Basic.NET 包含内置的或内部的函数，如 MsgBox、CStr 等。此外，还可用 Function 语句编写自己的 Function 过程。函数过程的语法是：

Private | Public | Friend] Function functionname (*argument list*) [As data type]

Statements

```
End Function
```

与 Sub 过程一样，Function 过程也是一个独立的过程，可读取参数、执行一系列语句并改变其参数的值。与 Sub 过程不同的是，Function 过程可返回一个值到调用的过程。在 Sub 过程与 Function 过程之间有三点区别：

（1）一般说来，语句或表达式的右边包含函数过程名和参数（returnvalue = function），这就调用了函数。

（2）与变量完全一样，函数过程有数据类型。这就决定了返回值的类型（如果没有 As 子句，缺省的数据类型为 Object）。

（3）可以给 functionname 赋一个值，即为返回的值。

Function 过程返回一个值时，该值可成为表达式的一部分。例如，下面是已知直角三角形两直角边的值，计算第三边（斜边）的函数：

```
Function Hypotenuse (A As Integer, B As Integer) As String
    Hypotenuse = Math.Sqrt (A ^ 2 + B ^ 2)
End Function
```

在 Visual Basic 中调用 Function 过程的方法和调用任何内部函数的方法是一样的：

```
Label1.Text = CStr(Hypotenuse(CInt(Text1.Text), CInt(Text2.Text)))
strX = CStr(Hypotenuse (Width, Height))
```

3.5.3　调用过程

1. 调用 Sub 过程

与 Function 过程不同，在表达式中，Sub 过程不能用其名字调用，调用 Sub 过程的是一个独立的语句。Sub 过程还有一点与函数不一样，它不会用名字返回一个值。但是，与 Function 过程一样，Sub 过程也可以修改传递给它们的任何变量的值。调用 Sub 过程有两种方法：

```
'以下两个语句都调用了名为 MyProc 的 Sub 过程。
Call MyProc (FirstArgument, SecondArgument)
MyProc (FirstArgument, SecondArgument)
```

2. 调用函数过程

通常，调用自行编写的函数过程的方法和调用 Visual Basic 内部函数过程（例如 Abs）的方法一样，即在表达式中写上它的名字。

下面的语句都调用函数 MyFunc：

```
TextBox1.Text = CStr(10 * MyFunc)
X = MyFunc()
```

就像调用 Sub 过程那样，也能调用函数。下面的语句都调用同一个函数：

```
Call Year (Now)
Year (Now)
```

当用这种方法调用函数时，Visual Basic 放弃返回值。

3. 调用其他模块中的过程

在工程中的任何地方都能调用类模块或标准模块中的公用过程。可能需要指定这样的模块，它包含正在调用的过程。调用其他模块中的过程的各种技巧，取决于该过程是在类模块中还是标准模块中。

在类模块中调用过程要求调用与过程一致并且指向类实例的变量。例如，DemoClass 是类 Class1 的实例：

```
Dim DemoClass as New Class1
```

DemoClass.SomeSub 在引用一个类的实例时，不能用类名作限定符。必须首先声明类的实例为对象变量（在这个例子中是 DemoClass）并用变量名引用它。

标准模块中的过程如果过程名是惟一的，则不必在调用时加模块名。无论是在模块内，还是在模块外调用，结果总会引用这个惟一过程。如果过程仅出现在一个地方，这个过程就是惟一的。如果两个以上的模块都包含同名的过程，那就有必要用模块名来限定了。在同一模块内调用一个公共过程就会运行该模块内的过程。例如，对于 Module1 和 Module2 中名为 CommonName 的过程，从 Module2 中调用 CommonName 则运行 Module2 中的 CommonName 过程，而不是 Module1 中的 CommonName 过程。从其他模块调用公共过程名时必须指定那个模块。例如，若在 Module1 中调用 Module2 中的 CommonName 过程，要用下面的语句：Module2.CommonName (arguments)

3.5.4 向过程传递参数

过程中的代码通常需要某些关于程序状态的信息才能完成它的工作。信息包括在调用过程时传递到过程内的变量。当将变量传递到过程时，称变量为参数。

1. 参数的数据类型

过程的参数被缺省为具有 Object 数据类型。不过，也可以声明参数为其他数据类型。例如，下面的函数接受一个字符串和一个整数：

```
Function WhatsForLunch(WeekDay As String, Hour As Integer) As String
    '根据星期几和时间，返回午餐菜单。
    If WeekDay = "Friday" then
        WhatsForLunch = "Fish"
    Else
        WhatsForLunch = "Chicken"
    End If
    If Hour > 4 Then WhatsForLunch = "Too late"
End Function
```

2. 按值传递参数

按值传递参数方式在 Visual Basic.NET 中是缺省的。按值传递参数时，传递的只是变量的副本。如果过程改变了这个值，则所作变动只影响副本而不会影响变量本身。用“ByVal”关键字指出参数是按值来传递的。例如：

```
Sub PostAccounts (ByVal intAcctNum as Integer)
    '这里放语句。
```

```
End Sub
```

注意："ByVal" 关键字可以省略。

3. 按地址传递参数

按地址传递参数是过程用变量的内存地址去访问实际变量的内容。结果，将变量传递给过程时，通过过程可改变变量值。按地址传递参数如果给按地址传递参数指定数据类型，就必须将这种类型的值传给参数。Visual Basic 计算表达式，如果可能的话，还会按要求的类型将值传递给参数。把变量转换成表达式的最简单的方法就是把它放在括号内。用 "ByRef" 关键字指出参数是按地址来传递的，按地址传递的效率比较高，因为，无论变量是什么类型，传进的都只是 4 个字节。如下例：参数 "RunningTotal" 是按地址传进来的，因此传进的变量值为参数 "AcctNum" 的值。

```
Sub PostAccount(ByVal AcctNum As Integer, ByRef RunningTotal As Single)
    RunningTotal = AcctNum
End Sub
```

4. 使用可选的参数

在过程的参数列表中列入 "Optional" 关键字，就可以指定过程的参数为可选的。指定可选参数，根据以下三条规则：

（1）每个可选的参数一定要有一个缺省值；

（2）可选参数的缺省值必须是个常数；

（3）可选参数的后面的所有参数也需是可选参数。

下面的例子给出了带有可选参数的过程的定义：

```
Sub subName(Optional ByVal MyCountry As String = "China")
    ......
End Sub
```

当调用这个过程的时候，可以选择是否给过程传递参数，如果不传递参数，则过程使用缺省的参数。

有的时候也许不想给可选参数传值，而且也不希望使用缺省值，但这种情况 Visual Basic.NET 是不允许的，这时只要给可选参数设定一个不会用到的缺省值就可以解决这个问题了，如下例：

```
Sub PlaceInfo(ByVal State As String, Optional ByVal Country As String = "QJZ")
    If Country = "QJZ" Then
        Debug.WriteLine("Country not supplied -- using Australia")
        Country = "Australia"
    End If
    MsgBox("The state of " & State & " is in " & Country)
End Sub
```

可以这样调用这个过程：

```
PlaceInfo( State:="Maryland" , Country:="USA")
'...
PlaceInfo( State:="Queensland" )
```

5. 使用不定数量的参数

一般说来，过程调用中的参数个数应等于过程说明的参数个数。可用一个参数数组向过程传递参数，当定义过程的时候，不必知道参数数组中的元素个数，参数数组的大小由每次调用过程时决定。

用关键字“ParamArray”表示数组参数，其规则如下：

（1）一个过程只能有一个参数数组，而且参数数组必须在其他参数的后面。

（2）参数数组必须是按值传递的，在过程定义此参数数组时，明确有关键字“ByVal”。

（3）参数数组必须是一维数组，参数数组本身的每个元素必须是同一种类型的，如果没定义，按“Object”类型处理。

（4）参数数组一旦声明就是可选参数，它的缺省值就是每种类型的“Empty”值。

下面的例子说明了参数数组的使用：

```
Sub StudentScores(Name As String, ByVal ParamArray Scores() As Object)
    Dim I As Integer
    Debug.WriteLine("Scores for " & Name & ":")
    ' Use UBound function to determine largest subscript of array.
    For I = 0 To UBound(Scores())
        Debug.WriteLine("Score " & I & ": " & Scores(I))
    Next I
End Sub
```

可以这样调用这个过程：

```
StudentScores("Jamie", 10, 26, 32, 15, 22, 24, 16)
  '...
StudentScores("Kelly", "High", "Low", "Average", "High")
```

3.6　Visual Basic.NET 的类和对象基础

OOP（Object-Oriented-Programming）是相对于结构化程序设计（Structure Programming）而言的，表示采用面向对象的思想进行软件的编制。它是当今最流行的编程模式。“面向对象”技术追求的是软件系统对现实世界的直接模拟，尽量实现将现实世界中的事物直接映射到软件系统的解空间。

对象是一些把属性（Properties）、字段（fileds）、方法（methods）、事件（event）作为一个单独的数据类型进行处理的实体。编程还需要对象满足和支持下面三个特性：封装性、继承性和多态性。对象可以使用户只需首先声明一些变量和方法以后，就可以在需要使用时候，调用一个它的对象进行重用。

在 Visual Basic 的早期版本中（1.0-3.0）并没有包括面向对象的功能，从 VB 4.0 开始，用户可以像建立一个新的窗体那样建立一个新的类，并把它作为一个新的对象，类在 Visual Basic.NET 中是一个非常重要的部分，几乎所有正规的程序都包括了一个或者几个类，在

Visual Basic.NET 中，类模块和窗体的区别已经不存在了，几乎所有的程序都是由类组成的，为了更好的使用这些类，它们被按照不同的功能分在不同的功能类库里。

Visual Basic.NET 在声明、构造以及执行类上比 Visual Basic 6.0 有很大的变化，同样在对象的处理上也有不小的改变。这些改变影响了定义一个对象、引用和取消引用对象以及使用捆绑技术的方法。

Visual Basic.NET 没有使用 CreateObject 语句来创建对象。CreateObject 是 Visual Basic 与 COM 密切相关的一个产物。因为 Visual Basic.NET 不再使用 COM，所以从 Visual Basic.NET 开始就不再使用 CreateObject。Visual Basic.NET 使用 New 语句来创建对象。用户可以在代码的任何地方使用 New。下面就是创建一个变量并且在类的一个实例中创建一个对象的实例：

```
Dim obj As TheClass
obj = New TheClass()
```

可以简化上面的语句：

```
Dim obj As New TheClass()
```

在 Visual Basic 6.0 中，以上的两段语句会存在一些问题，但是在 Visual Basic.NET 中上面的两段语句之间没有区别，只是第二段语句缩短了而已。如果用户在一个模块中声明一个变量，那个变量就只有在该模块中才有效。在许多情况下，用户想在方法的作用域中声明一个变量，或者想在模块比如 Try...End 或者 loop 循环结构）中创建一个类的实例。在这样的情况下，用实例化来组合声明有点不妥。

上面的语句为用户定义了一个变量并实例化了一个类。这条语句可能在处理继承或者多个界面的时候会更有用。用户可以声明变量为其中一种类型并且基于要执行界面的类来实例化对象：

```
Dim obj As MyInterface = New TheClass()
```

也可以同时利用更复杂的语句。假如现在有个需要对象引用的方法，就可以这样来实例化对象：

```
DoSomething(New TheClass())
```

以上的语句是调用 DoSomething 方法并且传递 TheClass 的一个新的实例作为参数。这个新的对象就只有存在于这个方法的调用的事件内，即当方法完成之后，这个对象就自动被取消引用。

这里还得注意一下，取消引用一个对象不是意味着马上终止这个对象。这一点前面的介绍中有过解释。对象只有在.NET 的垃圾收集处理程序的时候才将它们从内存清除掉。

下面的例子可能更复杂。不是利用一个对象引用，这里的方法需要一个字符串。用户可以从一个方法提供一个字符串数值到对象中来实例化对象并调用方法：

```
DoSomething(New TheClass().GetStringData())
```

很显然，用户可能需要仔细观察一下这条语句的可读性。语句的压缩往往降低了可读性，这一点是应该注意到。

当处理对象的时候，这里没有使用 Set 语句。在 Visual Basic 6.0 中，处理对象引用的时候，用户不得不使用 Set 命令处理来自其他数据类型的对象。而在 Visual Basic.NET 中来自其他数据类型的对象是采用不同的方法来处理的，这里可以使用直接的参数来处理对象，就像处理整型或者字符型数据类型。这个 Set 命令在 Visual Basic.NET 中不再有效。

在 Visual Basic 6.0 中，用户可以通过设置对象引用为 Nothing（空）来取消引用对象。这点和 Visual Basic.NET 中的处理方法是一样的。

```
Dim obj As TheClass
obj = New TheClass()
obj = Nothing
```

但是在 Visual Basic.NET 中这条语句有不同的效果。因为 Visual Basic.NET 不是使用引用计算来终止对象，而是依靠垃圾收集机理。而在 Visual Basic 6.0 中，当没有变量访问对象的引用，这个对象就被终止。在 Visual Basic.NET 中，以下这样看法是错误的：当垃圾收集处理程序发现对象没有被引用它就被终止。正确的应该是，在最后一个引用被移除之后的一段时间后才将对象从内存中清除。但是这并没有清除被取消对象的数值。如果用户有一个长时间运行的算法，那么最好在处理程序中显式地取消引用这个对象，这样就可以在可能的情况下由垃圾收集程序清除它们。只要代码保留了对象的引用，那么对象就会保持在内存中而不会被垃圾收集程序清除。

Visual Basic.NET 可以创建处理代码，这些处理代码是运行在.NET 框架上的。所有的管理代码可以和其他类型的处理代码交互作用，而不管用户是用什么编程语言来创建这些组件的。这就意味着用户可以在一种编程语言上创建一个类，然后应用到另外的编程语言中，当然也包括继承。这正是交叉语言混继承的混合编程机理。实际上，现在有许多程序设计人员已经在应用这个技术了。许多的.NET 系统类库是在 C# 上编写的，而用户在 VB.NET 上编写程序的时候，可以继承这些类作为基类。

举个例子，在 Visual Basic.NET 创建一个类库工程取名为 vblib 并且增加一个简单的类为 Parent，代码如下：

```
Public Class Parent
Public Sub DoSomething()
MsgBox(Parent DoSomething, MsgBoxStyle.Information, Parent)
End Sub
End Class
```

可能许多程序设计人员还不知道 Visual Basic.NET 支持 Windows 窗体的可视化继承。可视化继承，就是说用户可以创建一个 Windows 窗体，然后可以继承这个窗体使得其他窗体具有与该窗体相同的版面布置、控件和行为。用户也可以使用继承来创建自己的 Windows 控件。比如，用户可以创建一个性能提高了的 TextBox（文本框）来实现数据输入的特殊的有效性校验。具体的做法是：通过继承创建一个由原始文本框控件类继承而来的子类，并进行适当的修改以提高文本框的性能，使之可以实现对数据输入的有效性校验。这一点跟 Web 表单控件是相同的，Web 表单控件可以由一个已经存在的 Web 表单控件来创建一个子类。用户的子类可以重载已有的函数或者增加一些新的函数。

Visual Basic.NET 不仅提供了继承，而且还提供了其他许多重要的新特性。Visual Basic.NET 改进了创建和处理多界面的方法，使得用户可以更容易地使用它们。另外，Visual Basic.NET 支持事件作为界面的一个部分，使得现在可以在界面上表达所有的元素：方法、属性、事件。此外，在 Visual Basic.NET 中终止对象不是通过引用计数来实现，而是利用“垃圾收集”处理程序来将对象从内存中清除。总而言之，Visual Basic.NET 比起 Visual Basic 以前的版本来说，其面向对象的能力大大增加了，而且 Visual Basic.NET 也保留了低版本绝大多数的特性。

关于面向对象的详细理论请参阅第 6 章 Visual Basic.NET 的 OOP 结构。

第 4 章　Visual Basic.NET 的常用控件

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

本章包括:

Visual Basic.NET 的一些标准控件的用法以及常用组件的使用，这些控件与组件用来接受用户输入，显示程序的输出信息。

4.1　Label 控 件

Label（标签）控件用来显示文本，是设计程序应用程序界面时经常要用到的控件之一，主要是用来显示其他控件名称，描述程序运行状态或标识程序运行的结果信息等等，响应程序的事件或跟踪程序运行的结果，Label 控件在工具箱中的图标如图 4.1 所示。

使用 Label 控件的情况很多，但用 Label 控件显示的信息一般都不是描述 Label 控件自身的特性，而是描述其他控件的属性或特征（如名称等）。例如，可用 Label 控件为文本框（TextBox）、列表框（ListBox）、组合框（ComboBox）等控件添加描述性的信息。

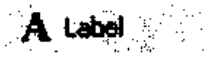


图 4.1　Label 控件

还可以编写代码改变 Label 控件显示的文本内容以响应程序运行时的事件或状态信息。如果一个程序在运行的过程中，有些对象随着不同时间段，各种信息都在变化，这时就可以用 Label 控件处理状况消息。

注意：Label 控件不接受焦点。

1. 设置标签的文本

在 Label 控件中显示文本，使用“Text”属性。在开发应用程序时。首先选择 Label 控件，再选择“属性窗口”，再在“属性窗口”中设置该属性为某个字符串量即可。

“Text”属性的长度最长可设置为 1024 字节。

2. 设置标签中文本属性

Labe 控件中的文本默认时的排列方式为居左（从左侧起依次排列到右侧），通过设置“TextAlign”属性可以改变排列方式，设置“TextAlign”为“Right”，排列方式为居右，设置“TextAlign”为“Center”，排列方式为居中。

缺省的时候当输入到“Text”属性的文本超过控件的宽度时，文本会自动换行，而且在超过控件的高度时，超出的部分将无法显示出来。

3. 标签的其他属性

描述 Label 控件的边框的属性是“BorderStyle”，如果将“BorderStyle”属性设成“FixedSingle”（可以在设计时进行），那么 Label 控件就有了一个边框；如果将“BorderStyle”

属性设成“Fixed3D”，那么 Label 控件就有了一个立体边框，看起来像一个 TextBox（文本框）。还可以通过设置 Label 控件的“BackColor”（背景颜色）、“ForeColor”（字体颜色）和“Font”（字体）等属性来改变 Label 控件的其他外观。

4. 用标签创建访问键

可以将“Text”属性中的字符定义成访问键，想要将 Label 控件的“Text”属性定义成访问键，首先要将“UseMnemonic”属性设置为“True”。定义了 Label 控件的访问键后，用户按【Alt＋指定的字符】组合键，就可将焦点按【Tab】键次序移动到下一个控件。在作为访问键的字母之前添加一个连字符（&），就可为其他不具有标题的控件（如 TextBox 控件）创建访问键。上面已经提到了标签不接受焦点，因此焦点会按照【Tab】键次序自动移动到下一控件处。可用这种技术为文本框、图片框、组合框、列表框、驱动器列表框、目录列表框、网格和图像等指定访问键。要将标签指定为控件的访问键，请执行下述步骤：

（1）首先绘制标签，然后再绘制控件。或者以任意顺序绘制控件，并将标签的“TabIndex”属性设置为控件的“TabIndex”属性减 1；

（2）在标签的“Text”属性中用连字符为标签指定访问键。

注意：有时可能要在 Label 控件中显示连字符而不是用它们创建访问键。如果在一次记录集中，数据包含连字符，而且要将 Label 控件绑定到记录集的某个字段，就会出现所说情况。为在 Label TextBox 控件中显示连字符，应将“UseMnemonic”属性设置为 False。

4.2 Button 控件

图 4.2　Button 控件

Button（按钮）控件一般接受鼠标单击事件被用来启动、中断或结束一个进程（相当于 Visual Basic 以前版本的 CommandButton 控件）。单击 Button 控件时将调用已写入 Click 事件过程中的过程。Button 控件在大多数 Visual Basic 应用程序中都有会用到，用户可以单击按钮执行操作。单击时，按钮不仅能执行相应的操作，而且看起来与被按下和松开一样。Button 控件在工具箱中的图标如图 4.2 所示。

1. 向窗体添加按钮

在应用程序中很可能要使用多个按钮。就像在其他容器控件上绘制按钮那样，在窗体上添加按钮。可用鼠标调整按钮的大小，也可通过设置“Location”（坐标，用来确定控件相对窗体左上方顶点的位置）和“Size”（大小，“Width”代表宽度，“Height”代表高）属性进行调整。

2. 设置按钮显示文本

可用“Text”属性改变按钮上显示的文本。设计时，可在控件的“属性窗口”中设置此属性。在设计时设置“Text”属性后将动态更新按钮文本。“Text”属性最多包含 255 个字符。若标题超过了命令按钮的宽度，则会折到下一行。但是，如果控件无法容纳其全部长度，则标题会被剪切。可以通过设置“Font”属性改变在命令按钮上显示的字体。

3. 创建键盘快捷方式

可通过“Text”属性创建按钮的访问键快捷方式，为此，只需在作为访问键的字母前添

加一个连字符（&）。例如，要为标题“OK”创建访问键，应在字母“O”前添加连字符，于是得到“&OK”。运行时，字母“O”将带下划线，同时按【Alt+O】键就可执行单击按钮程序所执行的动作。

注意：如果不创建访问键，而又要使标题中包含连字符，应添加两个连字符(&&)。这样，在标题中就只显示一个连字符。

超星阅览器提醒您
使用本复制品
请尊重相关知识产权!

4. 选定按钮

运行时，可用鼠标或键盘通过下述方法选定按钮：

（1）用鼠标单击按钮；

（2）按【Tab】键，将焦点转移到按钮上，然后按【Spacebar】或【Enter】键选定按钮；

（3）按按钮的访问键（【Alt+带有下划线的字母】）。

5. Click 事件

选定按钮时将触发按钮的“Click”事件并执行写入“Click”事件过程中的代码，同时单击按钮的过程中也将生成“MouseMove”、“MouseLeave”、“MouseDown”和“MouseUp”等事件。如果要在这些相关事件中附加事件过程，则应确保操作不发生冲突。对控件的操作不同，这些事件过程发生的顺序也不同。Button 控件的单击事件发生的顺序为：“MouseMove”、“MouseDown”、“Click”、“MouseUp”、“MouseLeave”。

注意：如果用户试图双击按钮控件，则其中每次单击都将被分别处理；也即按钮控件不支持双击事件。

6. 增强按钮的视觉效果

按钮控件像复选框和选项按钮一样，可通过“Image”属性设置 Button 控件上的图标以增强视觉效果，然后设置图标（图片）的属性：“ImageAlign”显示图标（图片）的位置。通过设置“ImageIndex”（图片在图片框中的索引）以及“ImageList”（图片框）则可实现如下的效果，比如要向按钮添加图标或位图，或者在单击、禁止控件时显示不同的图像等等。

4.3　TextBox 控件

TextBox（文本框）控件也是在应用程序中经常要用到的控件之一，主要用来在程序运行时接受用户输入，也可以显示运行的结果，以完成用户与程序的交互。TextBox 控件在工具箱中的图标如图 4.3 所示。

图 4.3　TextBox 控件

TextBox 是一种通用控件，可以由用户输入文本或显示文本。除非把 TextBox 的“Locked”属性设为“True”，否则不能用 TextBox 显示不希望用户更改的文本。TextBox 中显示的实际文本是受“Text”属性控制的。“Text”属性可以用三种方式设置：设计时在“属性”窗口进行、运行时通过代码设置或在运行时由用户输入。

1. 设置文本框的文本

通过读“Text”属性能在运行时检索 TextBox 的当前内容。TextBox 在缺省情况下只显示单行文本，且不显示“ScrollBar”（滚动条）。如果文本长度超过可用空间，则只能显示部

分文本。通过设置“MultiLine”和“ScrollBars”两种属性（只能在设计程序时设置），可以改变 TextBox 的外观和行为。注意：不要把“ScrollBars”属性与“ScrollBar”控件混淆，“ScrollBar”控件并不属于 TextBox，它具有自己的属性集。把“MultiLine”属性设为“True”，可以使 TextBox 在运行时接受或显示多行文本。只要没有水平方向“ScrollBar”，多行 TextBox 中的文本会自动按字换行。“ScrollBars”属性的缺省值被设置为 0（None）。自动按字换行省去用户在行尾插入换行符的麻烦。当一行文本已超过所能显示的长度时，TextBox 自动将文本折回到下一行显示。在设计时，不能在“属性”窗口输入换行符。在过程中，可以通过插入一个回车加上换行符（ANSI 字符 13 和 0）来产生一个行断点。

注意：如果将“MultiLine”属性设为“False”，则文本框的宽度就无法改变了，这个宽度由字体的大小决定。

使用 TextBox 中的文本利用 TextBox 的“SelectionStart”、“SelectionLength”和“SelectedText”属性，可以控制 TextBox 的插入点和选择行为。这些属性仅能在运行时使用。当一个 TextBox 首次得到焦点时，TextBox 缺省的插入点和光标位置在文本的最左边。用户可以用键盘和鼠标移动它们。当 TextBox 失去焦点而后再得到时，插入点位置与用户最后设置的位置一样。在有些情况下，它可能与用户设置不一致。如：在字处理应用程序中，用户会希望新字符出现在已有文本后面；在数据项应用程序中，用户会希望他的输入替换原有条目。使用“SelectionStart”和“SelectionLength”属性，用户可以根据需要改变 TextBox 的行为。“SelectionStart”属性是一个数字，指示文本串内的插入点，其中值 0 表示最左边的位置。如果“SelectionStart”属性值大于或等于文本中的字符数，那么插入点将被放在最后一个字符之后。“SelectionLength”属性是一个设置插入点宽度的数值。把“SelectionLength”设为大于 0 的值，会选中并突出显示从当前插入点开始的“SelectionLength”个字符。如果有一段文本被选中，此时用户键入的文字将替换被选中的文本。有些情况下，也可以使用粘贴命令用新文本替换原有的文本。“SelectedText”属性是一串文本，可以在运行时给它赋值以替换当前选中的文本。如果没有选中的文本，“SelectedText”将在当前插入点插入文本。

2. 创建密码文本框

密码框是一个文本框的特殊且常用的形式，它允许在用户输入密码的同时显示星号之类的占位符。Visual Basic7.0 提供“PasswordChar”和“MaxLength”这两个文本框属性，大大简化了密码文本框的创建。“PasswordChar”指定显示在文本框中的字符。例如，若希望在密码框中显示星号，则可在“属性”窗口中将“PasswordChar”属性指定为“*”，如图 4.4 所示，无论用户输入什么字符，文本框中都显示星号。可用“MaxLength”设定输入文本框的字符数；输入的字符数超过“MaxLength”后，系统不接受多出的字符并发出嘟嘟声。

图 4.4　密码示例

3. 创建只读文本框

可用“ReadOnly”属性防止用户编辑文本框内容。将“ReadOnly”属性设置为“True”后，用户就可滚动文本框中的文本并将其突出显示，但不能作任何变更。将“ReadOnly”属性设置为“True”后就可在文本框中使用“复制”命令，但不能使用“剪切”和“粘贴”

命令。“ReadOnly”属性只影响运行时的用户交互。这时仍可变更“Text”属性，从而在运行时通过程序改变文本框的内容。

4. 显示字符串中的引号

引号（“”）有时出现在文本的字符串中。

例如：She said, "You deserve a treat!"因为赋予变量或属性的字符串都用引号（“”）括起来，所以对于字符串中要显示的一对引号，必须再插入一对附加的引号。Visual Basic 将并列的两对引号解释为嵌入的引号。

例如，要显示上面的字符串就应使用下述代码：

```
TextBox1.Text = "She said, ""You deserve a treat!"" "
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

可用引号的 ASCII 字符（34）达到相同效果：

```
TextBox1.Text = "She said, " & Chr(34) + "You deserve a treat!" & Chr(34)
```

4.4　MainMenu 控 件

MainMenu（主菜单）控件是 Visual Basic.NET 新增的控件，，以取代 Visual Basic 以前版本的“菜单编辑器”，MainMenu 控件较“菜单编辑器”有很大优势，因为以控件的“身份”，更有利于应用程序对整个菜单的操作，而且，可以实现不同窗体的多次利用，而不必每个窗体都要重新设计菜单。MainMenu 控件在工具箱中的图标如图 4.5 所示。

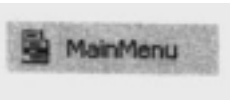


图 4.5　MainMenu 控件

1. 在设计时添加 MainMenu 控件

添加 MainMenu 控件与添加其他控件的方法一样（双击或拖动），但是 MainMenu 控件本身并不存在于窗体之上，而是在窗体下方的“组件栏”中（如图 4.6 所示），在这里的控件都不出现在窗体中，但它（们）的方法、过程、属性都影响着窗体及其上的其他控件，类似的控件还有“Timer”（时间）控件、“HelpProvide”（帮助提供）控件等。

2. 制作菜单

单击“组件栏”中的“MainMenu”控件，则在窗体的左上方出现“Type Here”字样，单击“Type Here”，更改菜单的标题，如图 4.7 所示。

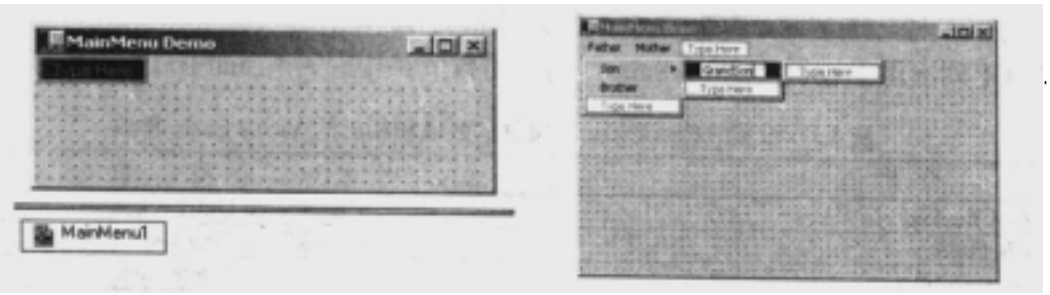


图 4.6　添加 MainMenu 控件　　图 4.7　制作菜单

右键单击任意一项子菜单，在弹出的菜单中选择“Edit Names”（更改名称），如图 4.8 所示。再次单击子菜单时，就可以更改菜单的名称了。第三次点击“Edit Names”则恢复正

常的菜单编辑状态。

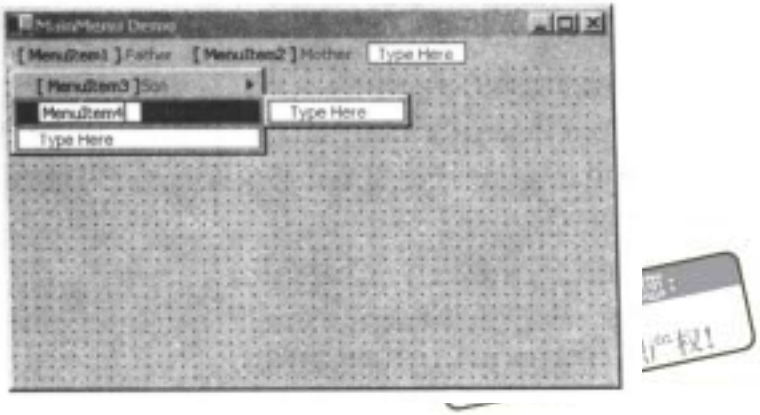


图 4.8　更改菜单的名称

3. 菜单的 Click 事件

菜单最常用的事件就是“Click”事件，一般说来，菜单的“Click”事件总是和工具栏中按钮的“Click”事件对应。

4.5　CheckBox 控件

CheckBox（复选框）控件用来标识某个选项是否为选定的状态。因此通常用此控件提供“Yes/No”或“True/False”选项。可用分组的 CheckBox 控件显示多组不同类型的选项，用户可从中一个组选择一个或多个选项。CheckBox 控件在工具箱中的图标如图 4.9 所示。

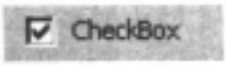


图 4.9　CheckBox 控件

CheckBox 控件与 RadioButton（单选框）控件都可以用来指示用户是否对某个选项作出选择。不同之处在于，对于一个组内 RadioButton 控件，一次只能选定其中的一个，而对所有的 CheckBox 控件，则可选定任意数目的复选框。RadioButton 控件将在后面进一步介绍。

1. CheckState 属性

CheckBox 控件的“CheckState”属性指示复选框处于选定、未选定或禁止状态（暗淡的）中的哪一种。选定时，“CheckState”设置值为 1。

表 4.1 列出用于设置“CheckState”属性的数值和相应的 Visual Basic 常数。

表 4.1　设置“CheckState”属性的数值和相应的 Visual Basic 常数

设 置 值	值	常　数
UnChecked	0	CheckState.UnChecked
Checked	1	CheckState.Checked
Indeterminate	2	CheckState.Indeterminate

用户单击 CheckBox 控件指定选定或未选定状态，然后可检测控件状态并根据此信息编写应用程序以执行某些操作。缺省时，CheckBox 控件设置为“CheckState.Unchecked”。若要预先在一列复选框中选定若干复选框，则应在“New”或“InitializeComponent”过程中

将“CheckState”属性设置为“CheckState.Checked”以选中复选框；可将“CheckState”属性设置为“CheckState.Indeterminate”以禁用复选框。例如，有时可能希望满足某条件之前禁用复选框。

2. Click 事件

无论何时单击 CheckBox 控件都将触发 Click 事件，然后编写应用程序，根据复选框的状态执行某些操作。在下例中，每次单击 CheckBox 控件时都将改变其“Text”属性以指示选定或未选定状态：

```
Protected Sub CheckBox1_Click(ByVal  sender As Object, ByVal e As _
                              System.EventArgs ) Handles CheckBox.Click
    If CheckBox1.CheckState = CheckState.CheckedThen
        CheckBox1.Text = "Checked"
    ElseIf CheckBox1.CheckState = CheckState.UnChecked Then
        CheckBox1.Text = "Unchecked"
    End If
End Sub
```

使用本文制作
请尊重相关知识产权!

注意：如果试图双击 CheckBox 控件，则将双击当作两次单击，而且分别处理每次单击；这就是说，CheckBox 控件不支持双击事件。

3. 响应鼠标和键盘

在键盘上使用【Tab】键并按【Spacebar】键，由此将焦点转移到 CheckBox 控件上，这时也会触发 CheckBox 控件的 Click 事件。可以在“Text”属性的一个字母之前添加连字符，创建一个键盘快捷方式来切换 CheckBox 控件的选择。

4. 增强 CheckBox 控件的视觉效果

CheckBox 控件像 Button 和 RadioButton 控件一样，可通过更改“Style”属性的设置值后使用“Image”、“ImageAlign”、“ImageIndex”和“ImageList”属性增强其视觉效果。例如，有时可能希望在复选框中添加图标或位图，或者在单击或禁止控件时显示不同的图像等。

4.6　RadioButton 控件

RadioButton（选项按钮）控件和 CheckBox（复选框）一样，也是被用来标识某个选项是否为选定的状态（相当于 Visual Basic 以前版本的 OptionButton 控件）通常以一组选项按钮的形式出现，但用户在一个组中只能选择一个选项。也就是说，当用户选定一个选项按钮时，同组中的其他选项按钮会自动失效。相应的是，用户可选定任意数目的复选框。RadioButton 控件在工具箱中的图标如图 4.10 所示：

图 4.10　RadioButton 控件

1. 创建一组选项按钮

选项按钮一般是以组的形式存在的，一般来说，绘制在相同容器控件的同一类 RadioButton 就完成以组的形式存在了，像 GroupBox 控件、PictureBox 控件或窗体都可以作

为 RadioButton 组的容器。运行时，用户在每个选项组中只能选定一个选项按钮。例如，如果把选项按钮分别添加到窗体和窗体上的一个 GroupBox 控件中，则相当于创建两组不同的选项按钮。所有直接添加到窗体的选项按钮成为一组选项按钮。要添加附加按钮组，应把按钮放置在框架或 PictureBox 控件中，要将框架或图片框中的 RadioButton 控件分组，应首先绘制框架或图片框，然后在内部绘制 RadioButton 控件。设计时，可选择 GroupBox 控件或 PictureBox 控件中的选项按钮，并把它们作为一个单元来移动。要选定 GroupBox 控件、PictureBox 控件或窗体中所包含的多个控件时，可在按住【Ctrl】键的同时用鼠标在这些控件周围绘制一个方框。

2. 运行时选择选项按钮

在运行时有若干种选定选项按钮的方法：用鼠标单击某个选项按钮，使用【Tab】键将焦点转移到控件，使用【Tab】键将焦点转移到一组选项按钮后再用方向键从组中选定一个按钮，在选项按钮的标题上创建快捷键，或者在代码中将选项按钮的“Checked”属性设置为“True”。

3. Click 事件

选定选项按钮时将触发其 Click 事件。是否有必要响应此事件，这将取决于应用程序的功能。例如，当希望通过更新 Label 控件的标题向用户提供有关选定项目的信息时，对此事件作出响应是很有益的。

4. Checked 属性

选项按钮的“Checked”属性指出是否选定了此按钮。选定时，数值将变为“True”。可通过在代码中设置选项按钮的“Checked”属性来选定按钮。例如：RadioButton1.Checked=True，要在选项按钮组中设置缺省选项按钮，可在设计时通过“属性”窗口设置“Checked”属性，也可在运行时在代码中用上述语句来设置“Checked”属性。在向用户显示包含选项按钮的对话框时将要求他们选择项目，确定应用程序下一步做什么。可用每个 RadioButton 控件的“Checked”属性判断用户选定的选项并作出相应的响应。

5. 禁止选项按钮

要禁止选项按钮，应将其“Enabled”属性设置成“False”。运行时将显示暗淡的选项按钮，这意味着按钮无效。

4.7 GroupBox 控件

GroupBox（控件组）控件一般是作为其他控件的组的容器的形式存在的，这样有利于用户识别，使界面变得更加友好（GroupBox 控件相当于 Visual Basic 以前版本的 Frame 控件）。使用控件组控件可以将一个窗体中的各种功能进一步进行分类，例如，将各种选项按钮控件分隔开。在大多数的情况下，对控件组控件没有实际的操作，我们用它对控件进行分组，但是通常没有必要响应它的事件。不过，它的“Name”、“Text”和“Font”等属性可能回经常被修改，以适应应用程序在不同阶段的要求。GroupBox 控件在工具箱中的图标如图 4.11 所示。

图 4.11　GroupBox 控件

1. 在窗体中添加一个 GroupBox 控件

在使用控件组控件给其他控件分组的时候，首先绘出控件组控件，然后再绘制它内部的其他控件，其他的控件就以这个控件组控件为容器，这样在移动控件组的时候，可以同时移动它包含的控件。

2. 在控件组内部控制控件

要将控件加入到控件组控件中，只需将它们绘制在控件组控件的内部即可。如果将控件绘制在控件组控件之外，或者在向窗体添加控件的时候使用了双击方法，再将它移动到控件组控件内部，那么控件也将从属于这个控件组控件（如图 4.12 所示）。。这也是 Visual Basic.NET 中的 GroupBox 区别于以前版本的 Frame 的地方

3. 选择框架中的多个控件

要选择框架中的多个控件，在使用鼠标点击在控件组内包围控件的时候需要按下【Ctrl】键，在释放鼠标的时候，位于框架之内的控件将被选定，如图 4.13 所示。

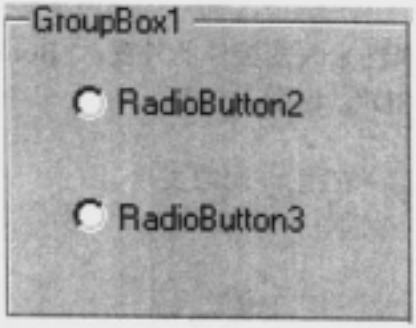


图 4.12 位于控件组内的控件

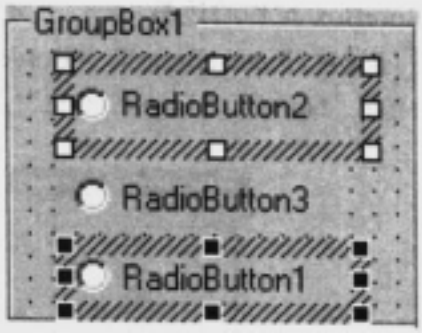


图 4.13 选择控件组中的控件

4.8 PictureBox 控件

PictureBox（图片箱）控件被用来显示图形，可以作为其他控件的容器。PictureBox 控件在工具箱中的图标如图 4.14 所示。

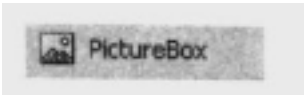


图 4.14 PictureBox 控件

1. 支持的图形格式

PictureBox 控件可显示下述任何格式的图片文件：位图、图标、图元文件、增强型图元文件、JPEG 或 GIF 文件。

2. 将图形加载到 PictureBox 控件中

在设计时，从“属性”窗口中选定并设置“Image”属性就可将图片加载到 PictureBox 控件中，也可在运行时设置 PictureBox 控件的“Image”属性：

```
PictureBox1.Image = Image.FromFile("c:\Windows\Winlogo.cur")
```

或

```
PictureBox1.Image.FromFile("c:\Windows\Winlogo.cur")
```

3. 使用剪贴板

设计时也可这样向 PictureBox 控件添加图形：从其他应用程序中复制图形后把它粘贴到 PictureBox 控件中。例如，有时可能希望添加由 WindowsPaint 创建的位图图像。直接

把图像复制到剪贴板，选定 PictureBox 控件，然后使用键盘快捷方式【Ctrl+V】或使用“编辑”菜单的“粘贴”命令。

4. 设置图片大小的属性

图片大小用“SizeMode”属性来设置。缺省时这个属性值为“Normal”，加载到图片框中的图形保持其原始尺寸，这意味着如果图形比控件大，则超过的部分将被剪裁掉，即 PictureBox 控件不提供滚动条。要使 PictureBox 控件自动调整大小以显示完整图形，应将“SizeMode”属性设置为“AutoSize”。这样控件将自动调整大小以适应加载的图形；将“SizeMode”属性设置为“StretchImage”时，插入的图片将按 PictureBox 的大小完整填充显示在其中；将“SizeMode”属性设置为“CenterImage”时，图片显示在 PictureBox 的中央。

PictureBox 控件不能伸展图像以适应控件尺寸。

5. 用 PictureBox 控件作容器

可用 PictureBox 控件作为其他控件的容器。例如，因为可将 PictureBox 控件放置到 MDI 子窗体的内部区域，所以通常用它手工创建工具条或状态条。

6. PictureBox 控件的边框

可以通过设置 PictureBox 的“BorderStyle”属性来改变控件的外观，“BorderStyle”共有三个属性值：“None”、“FixedSingle”及“Fixed3D”，如图 4.15 所示。

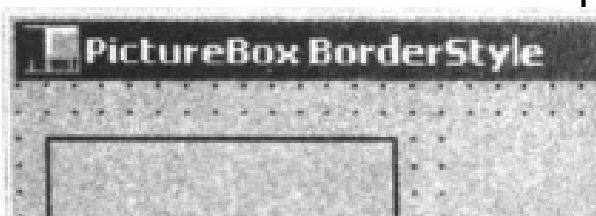


图 4.15　PictureBox 的 BorderStyle 属性

4.9　ListBox 控件

图 4.16　ListBox 控件

ListBox（列表框）控件可以显示一组项目的列表，用户可以根据需要从中选择一个或多个选项。列表框可以为用户提供所有选项的列表。虽然也可设置列表框为多列列表的形式，但在缺省时列表框单列垂直显示所有的选项，如果项目数目超过了列表框可显示的数目，控件上将自动出现滚动条。这时用户可在列表中上、下、左、右滚动。ListBox 控件在工具箱中的图标如图 4.16 所示。

1. Click 和 Double-Click 事件

对于列表框的双击事件，特别是当列表框参与触发应用程序的某一部分功能时，可以添加一个 Button 控件，并把该按钮同列表框并用。按钮的“Click”事件过程应该使用列表框的选项执行适于应用程序的操作。

双击列表中的项目与先选定项目然后单击按钮，这两者应该具有相同的效果。为此，应在 ListBox 控件的“DoubleClick”过程中调用按钮的“Click”过程：

```
Public Sub ListBox1_DoubleClick(ByVal sender As Object, _
                    ByVal e As System.EventArgs )Handles ListBox1.DoubleClick
    Button1_Click( sender , e )
End Sub
```

使用本复制品
请尊重相关知识产权!

这将为使用鼠标的用户提供快捷方式，同时也没有妨碍使用键盘的用户执行同样的操作。

注意：没有与“DoubleClick”事件等价的键盘命令，因此无法通过快捷键的方式达到 ListBox 控件的“DoubleClick”的功能。

2. 向列表添加项目

为了向列表框中添加项目，应使用 InsertItem 方法，其语法如下：

ListboxName.InsertItem（index As integer，Item As Object）

或

ListboxName.Items.Insert（index As Integer，Item As Object）

其中 ListboxName 为列表框的名称“item”添加到列表中的字符串表达式。若“item”是文字常数，则用引号将它括起来；“index”指定在列表中插入新项目的位置。“index”为 0 表示第一个位置。如果在列表框的第一个位置添加项目也可以用：

ListboxName.Items.Add（Item As Object）

通常在窗体设计时或 New 过程或 InitializeComponent（该过程存在于 Windows Form Designer generated code）中添加列表项目，但也可在任何时候使用 InsertItem 方法添加项目，于是可动态（响应用户的操作）添加项目。下列代码将“Germany”、“India”、“France”和“USA”添加到名为 ListBox1 的列表框中：

```
Public Sub New()
    MyBase.New
    Form1 = Me
    'This call is required by the Win Form Designer
    IntializeComponent
    ListBox1.InsertItem (0,"Germany")
    ListBox1.InsertItem (1,"India")
    ListBox1.InsertItem (2,"France")
    ListBox1.InsertItem (3,"USA")
    'TODO : Add any initialization after the InitializeComponent() call
End Sub
```

只要在运行时加载窗体就会出现如图 4.17 所示效果。

注意：“InitializeComponent”中的代码是在每次窗体计后自动生成的，因此尽量一次设计好窗体，因为如果重复设计窗体，以前所添加的代码将被覆盖掉，所以可以把向 ListBox

中添加代码编成一个过程，"'AddItem（）'过程"，在"New（）"或"InitializeCmpnent（）"调用这个过程。如下：

```
Public Sub New()
    MyBase.New
    Form1 = Me
    'This call is required by the Win Form Designer
    IntializeComponent
    Call AddItem（）
     'TODO : Add any initialization after the InitializeComponent() call
End Sub
Private Sub AddItem（）
    ListBox1.InsertItem (0,"Germany")
    ListBox1.InsertItem (1,"India")
    ListBox1.InsertItem (2,"France")
    ListBox1.InsertItem (3,"USA")
End Sub
```

3. 设计时添加项目

通过设置 ListBox 控件属性窗口的"Items"属性还可在设计时向列表添加项目。在选定了"Items"属性选项并单击"…"时，弹出"String Collection Editor"（字符串编辑框）可输入列表项目并按【Enter】键换行。只能在列表末端添加项目。所以，如果要将列表按字母顺序排序，则应将"Sorted"属性设置成"True"。

图 4.17 表框的内容

图 4.18 "Items"属性框

4. 排序列表

可以指定要按字母顺序添加到列表中的项目，为此将"Sorted"属性设置为"True"并省略索引。排序时不区分大小写；因此单词"japan"和"Japan"将被同等对待。"Sorted"属性设置为"True"后，"InsertItem"方法可能会导致不可预料的非排序结果。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

5. 从列表中删除项目

从列表框中删除项目可用下面的语法：

ListboxName.Items.Remove(Index As Integer)

“Remove”有一参数“index”，它指定删除的项目；它有一个重载函数：

ListboxName.Items.Remove(value As Object)

以适应不同参数输入的需要。要删除列表框中的所有项目，应使用 Clear 方法：

ListBox1.Items.Clear()。

6. 通过“Text”属性获取列表内容

通常，获取当前选定项目值的最简单方法是使用“Text”属性。“Text”属性总是对应用户在运行时选定的列表项目。例如，下列代码在用户从列表框中选定“Canada”时显示有关加拿大人口的信息：

```
Public Sub Listbox1_DoubleClick (Byval sender As object , Byval e As System.EventArgs)
    If  Listbox1.Text = "Canada"  Then
        Textbox1.Text = "Canada has 24 million people."
        End If
End Sub
```

“Text”属性包含当前在 Listbox1 列表框中选定的项目。代码检查是否选定了“Canada”，若已选定，则在 Text 框中显示信息。

7. 用“Items”属性访问列表项目

可用“Items”属性访问列表的全部项目。此属性包含一个数组，列表的每个项目都是数组的元素。每个项目以字符串形式表示。引用列表的项目时应使用如下语法：

ListboxName.Items(index)

“ListboxName”参数是列表框的名称，“index”是项目的位置。顶端项目的索引为 0，接下来的项目索引为 1，依此类推。例如，下列语句在一个文本框中显示列表的第三个项目（index = 2）：

Textbox1.Text = Ctr(Listbox1.Items(2))。

8. 用“SelectedIndex”属性判断位置

如果要了解列表中已选定项目的位置，则用“SelectedIndex”属性。此属性只在运行时可用，它设置或返回控件中当前选定项目的索引。设置列表框的“SelectedIndex”属性也将触发控件的“Click”事件。

如果选定第一个（顶端）项目，则属性的值为 0，如果选定下一个项目，则属性的值为 1，依此类推。若未选定项目，则“SelectedIndex”值为－1。

9. 使用“Count”属性返回项目数

为了返回列表框中项目的数目，应使用“Count”属性。例如，下列语句用“Count”属性判断列表框中的项目数：

```
Textbox1.Text = "You have " & Listbox1.Items.Count & " _entries listed"
```

4.10　CheckedListBox 控件

超星阅览器提醒您：
使用本复制品，
请尊重相关知识产权！

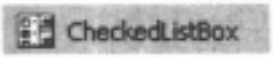


图 4.19　CheckedListBox 控件

CheckedListBox（复选列表框）控件可以说是 ListBox 控件的派生控件，因此继承了 ListBox 控件的很多方法和属性，CheckedListBox 在工具箱中的图标如图 4.19 所示。

CheckedListBox 控件完成的功能大致与 ListBox 相同，只是在添加的项目中有是否选定这一项。

1. CheckedListBox 的 Double_Click 事件

与列表框相似，复选列表框作为对话框的一部分出现时，建议添加一个按钮，并把该按钮同列表框并用。按钮的“Click”事件过程应该使用列表框的选项执行适于应用程序的操作。

双击列表中的项目与先选定项目然后单击按钮，这两者应该具有相同的效果。为此，应在 ListBox 控件的“DoubleClick”过程中调用按钮的“Click”过程：

```
Public Sub ListBox1_DoubleClick(ByVal sender As Object, _
                    ByVal e As System.EventArgs )Handles ListBox1.DoubleClick
    Button1_Click( sender , e )
End Sub
```

改为:

```
Public Sub CheckedListBox1_DoubleClick(ByVal sender As Object, _
                    ByVal e As System.EventArgs )Handles ListBox1.DoubleClick
    Button1_Click( sender , e )
End Sub
```

2. 设计时添加项目

通过设置 CheckedListBox 控件属性窗口的“Items”属性还可在设计时向列表添加项目。在选定了“Items”属性选项并单击“...”时，弹出“String Collection Editor”（字符串编辑框）可输入列表项目并按【Enter】换行。只能在列表末端添加项目。所以，如果要将列表按字母顺序排序，则应将“Sorted”属性设置成“True”。图 4.20 显示了复选列表框的项目编辑框。

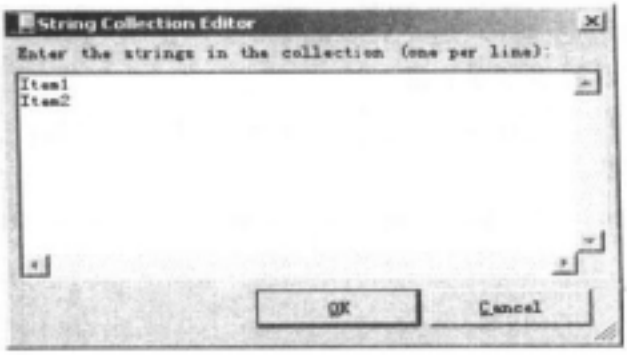


图 4.20　项目编辑框

单击“OK”按钮，则窗体上显示出添加了两个项目的复选列表框，如图 4.21 所示。

使用本复制品
请尊重相关知识产权!

图 4.21　添加项目的复选列表框

3. 运行时对项目的操作

有两种方法向复选列表框中添加项目：

- CheckListBoxName.Items.Add(item as Object , IsChecked as Boolean)
- CheckListBoxName.InsertItem(index as integer , item as Object)

第一种方法可以向复选列表框中添加一个标识选定与否的项目，第二种方法可以在指定的位置添加项目。例如向复选列表框中添加如下两个项目：

```
CheckedListBox1.Items.Add("dd", True)
CheckedListBox1.InsertItem(2, "kk")
```

设置项目的“Checked”的属性可以用以下语句：

```
CheckedListBox.SetItemChecked(index as Integer , value as Boolean)
```

可将“kk”项目的Checked属性设置为True：

```
checkedlistbox1.SetItemChecked(2, True) '假设"kk"项目的index属性为2
```

4.11　ComboBox 控件

ComboBox（组合框）控件相当于将文本框和列表框的功能结合在一起。这个控件可以实现输入文本来选定项目，也可以实现从列表中选定项目这两种选择项目的方法。如果项目数超过了组合框能够显示的项目数，控件上将自动出现滚动条。用户即可上下或左右滚动列表。ComboBox 控件在工具箱中的图标如图 4.22 所示。

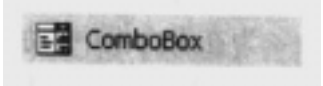


图 4.22　ComboBox 控件

1. 使用组合框和列表框

通常，组合框适用于建议性的选项列表，而当希望将输入限制在列表之内时，应使用列表框。组合框包含编辑区域，因此可将不在列表中的选项输入列区域中。此外，组合框节省了窗体的空间。只有单击组合框的向下箭头时（“Style”属性值为“1”的组合框除外，它总是处于下拉状态）才显示全部列表，所以无法容纳列表框的地方可以很容易地容纳组合

框。

超星阅览器提醒您：
使用本复制品
须遵守知识产权!

2. 组合框的样式

此处有三种组合框样式。每种样式都可在设计时或运行时来设置，而且每种样式都使用数值或相应的 Visual Basic 常数来设置组合框的样式。样式值常数下拉式组合框值为 0，Visual Basic.NET 中的常数值为 DropDown；简单组合框值为 1，Visual Basic.NET 中的常数值为 Simple；下拉式列表框值为 2，Visual Basic.NET 中的常数值为 DropDownList。

3. 下拉式组合框

在缺省设置（Style=0）下，组合框为下拉式。用户可（像在文本框中一样）直接输入文本，也可单击组合框右侧的附带箭头打开选项列表。选定某个选项后，将此选项插入到组合框顶端的文本部分中。当控件获得焦点时，也可按【Alt+↓】键打开列表（如图 4.23 所示）。

4. 简单组合框

将组合框“Style”属性设置为 1 将指定一个简单的组合框，任何时候都在其内显示列表。为显示列表中所有项，必须将列表框绘制得足够大。当选项数超过可显示的限度时将自动插入一个垂直滚动条。用户可直接输入文本，也可从列表中选择。像下拉式组合框一样，简单组合框也允许用户输入那些不在列表中的选项（如图 4.24 所示）。

5. 下拉式列表框

下拉式列表框（Style=2）与正规列表框相似 —— 它显示项目的列表，用户必须从中选择。但下拉式列表框与列表框的不同之处在于，除非单击框右侧的箭头，否则不显示列表（如图 4.25 所示）。

图 4.23　下拉式组合框　　图 4.24　简单组合框　　图 4.25　下拉式列表框

这种列表框与下拉式组合框的主要差别在于，用户不能在列表框中输入选项，而只能在列表中选择。当窗体上的空间较少时，可使用这种类型的列表框。

6. 添加项目

为在组合框中添加项目，应使用“Insert”方法，其语法如下：

ComboboxName.Items.Insert(index As Integer, item As Object)

“ComboboxName”为列表框或组合框名称，“item”为在列表中添加的字符串表达式，则用引号括起来。“Index”用来指定新项目在列表中的插入位置。“Index”为 0 表示第一个位置。当在第一个位置时，也可以用语法：

Combobox.Items.Add(item As Object)

通常在设计时或在 New 程中添加列表项目，但也可在任何时候使用“Insert”方法。这

样一来就能够动态地(响应用户的操作)在列表中添加项目。以下代码将“Chardonnay”、“Fum Blanc”、“Gewztraminer”和“Zinfandel”放置到名为 Combobox1，“Style”属性为 0（DropDown）的组合框中：

```
Public Sub New()
    ......
    ......
    Combobox1.Items.Insert "Chardonnay"
    Combobox1.Items.Insert "Fum Blanc"
    Combobox1.Items.Insert "Gewztraminer"
    Combobox1.Items.Insert "Zinfandel"
End Sub
```

运行时，只要加载窗体，而且用户单击向下箭头，则将显示如图 4.26 所示的列表。

图 4.26　组合框内容

7. 设计时添加项目

在设计时，也可设置组合框控件“属性窗口”的“Items”属性，从而在列表中添加项目。选定“Items”属性选项并单击“…”按钮后就可输入列表项目，然后按【Enter】键换到新的一行。

只能将项目添加到列表的末尾。所以，如果要将列表按字母顺序排序，则应将“Sorted”属性设置为“True”。

8. 指定位置添加项目

为了在列表指定位置添加项目，应在新项目后指定索引值。例如，下行代码将“Pinot Noir”插入到第一个位置并把其他项目的位置向下调整：

```
Combobox1.Items.Insert(0, "Pinot Noir")
```

注意：指定列表中的第一个位置的是 0 而不是 1

9. 排序列表

将“Sorted”属性设置为“True”并省略索引，则可在列表中指定按字母顺序添加的项目。排序时不区分大小写；所以“chardonnay”和“Chardonnay”被看作一个词。将“Sorted”属性设置为“True”之后，“Items.Insert”方法将导致不可预料的非排序结果。

10. 删除项目

可在组合框中用 Items.Remove 方法删除项目。Items.Remove 有一个参数 index，它指定要删除的项目：Combobox1.Items.Remove（index）及 index 参数和 Items.Insert 中的参数相同。例如，为了删除列表中的第一个项目，应添加下面一行代码：

```
Combobox1. Items.Remove(0)
```

为了在组合框中删除所有列表项目，应使用 Clear 方法：

```
Combobox1.Clear
```

11. 用“Text”属性获取列表内容

获取当前选定项目值的最简单的常用方法就是使用“Text”属性。在运行时无论向控件的文本框部分输入了什么文本，“Text”属性都与这个文本相对应。它可以是选定的列表选项，或者是用户在文本框中输入的字符串。例如，如果用户选定列表框中的“Chardonnay”，则通过下列代码显示有关“Chardonnay”的信息：

```
Private Sub Combobox1_Click (Byval  sender As Object, Byval e As System.EventArgs)
    If Combobox1.Text = "Chardonnay" Then
        Textbox1.Text = "Chardonnay is a medium-bodied _white wine."
    End If
End Sub
```

其中“Text”属性包含 Combobox1 列表框中当前选定的项目。代码查看是否选择了“Chardonnay”，若是如此，则在文本框中显示信息。

12. 用“Items”属性访问列表选项

有了“Items”属性就可访问列表中所有项目。该属性包含一个数组，而且列表中的每个项目都是数组的元素。每一项都表示为字符串的形式。为了引用列表中的项目，应使用如下语法：

```
ComboboxName.Items(index)
```

ComboboxName 为组合框名称，而 index 是项目的位置。顶端项目的索引为 0，下一个项目的索引为 1，依次类推。例如，在文本框中，以下语句显示列表中的第三个项目（index=2）：

```
Text1.Text = CStr(Combobox1.Items(2))
```

13. 用“SelectIndex”属性判断位置

想要知道组合框列表中选定项目位置，可以由“SelectIndex”属性得到。该属性设置或返回控件中当前选定项目的索引值，而且只在运行时有效。对组合框的“SelectIndex”属性进行设置也会触发控件的“Click”事件。若选定第一个（顶端）项目，则属性值“0”，选定的下一个项目属性值“1”，依次类推。如果未选定项目，或者用户在组合框中输入选项（样式“0”或“1”）而不在列表中选择现有项目，则“SelectIndex”为“−1”。

14. “Items.Count”属性返回项目数

为了返回组合框中的项目数，应使用“Items.Count”属性。例如，下列语句用“Items.Count”属性判断组合框中的项目数：

```
TextBox1.Text = "You have " & Combobox1.  “Items.Count”  & " _entries listed"
```

4.12 ListView 控件

图 4.27 ListView 控件

ListView（列表查看）控件用来以图形的形式显示项目，可使用 4 种不同视图。通过此控件，可将项目组成带有或不带有列标头的列，并显示伴随的图标和文本。ListView 控件在工具箱中的图标，如图 4.27 所示。

可使用 ListView 控件将称作 ListItem 对象的列表条目组织成下列 4 种不同的视图之一：

（1）大（标准）图标：可用鼠标操作，使用户能够拖放该对象，并重新排列这些对象，如图 4.28 所示。

（2）小图标：同时能显示更多的对象，用户也可重新排列这些对象，如图 4.29 所示。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

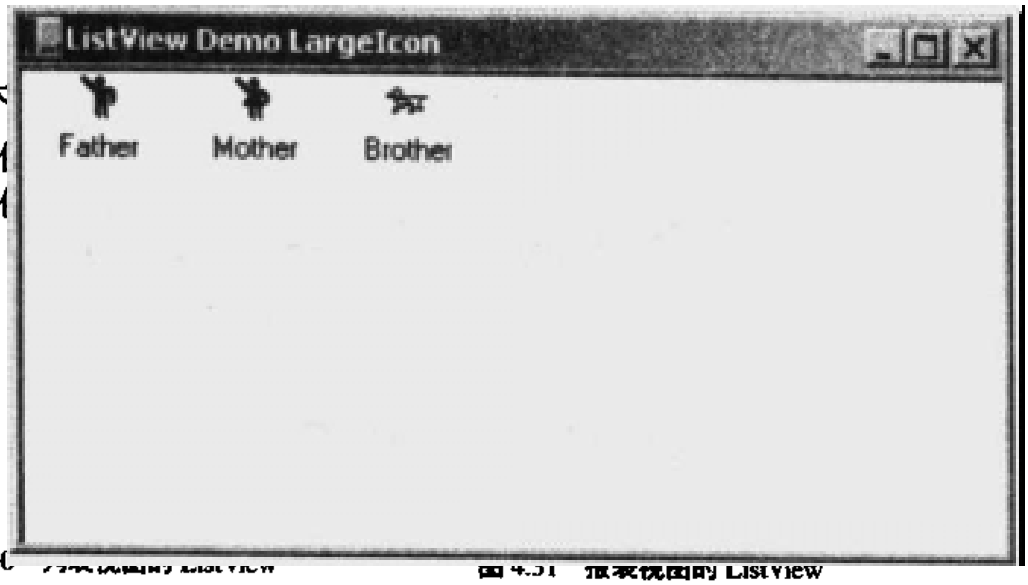


图 4.28　大

（3）列表：提

（4）报表：提

图 4.30　列表视图的 ListView　　图 4.31　报表视图的 ListView

1. 用“View”属性改变视图

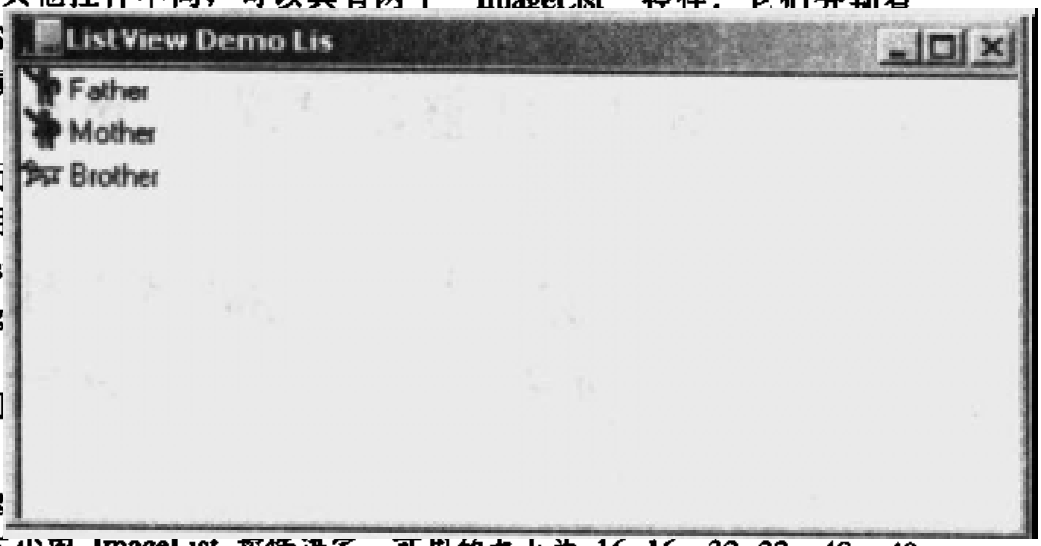
要改变视图，可以使用“View”属性。下面的代码将“View”属性设置为报表视图（如图 4.31 所示），代码中使用了内部常数 View.LargeIcon，该控件的名称为“ListView1”。

ListView1.View = View.LargeIcon

用“View”属性，可使最终用户动态地改变视图。

2. 用于图标视图和小图标视图的两个 ImageList 控件

“ListItem”对象由标签（“Text”属性）和由“ImageList”控件提供的可选图像构成。然而，ListView 控件和其他控件不同，可以具有两个“ImageList”控件，它们分别有“LargeImageList”和“S



用的视图，它由“View”

对象。在这三种视图下，

供图像。在设计时或运行

这些图像。在设计时，用

为具体的某个“ImageLis

ListView1.SmallImag

但是在图标视图下，

用“属性页”对话框，可

时使用下面的代码：

ListView1.LargeImag

注意：所用图标的大小由 ImageList 控件决定，可用的大小为 16×16、32×32、48× 48 和自定义大小。

如果计划使用多种视图，并且显示图像，就必须为每个 ListItem 对象设置

"SmallImageList"和"LargeImageList"属性。

超星阅览器提醒您：
使用本复制品
须遵守相关知识产权！

3. InsertItem 方法

添加 ListItem 对象到 ListView 控件的 ListItems 集合中并返回新创建对象的引用。应用于 ListItem 对象，ListItems 集合。语法为：

```
ListViewName.InsertItem(index as Integer , text as String , imageIndex as Integer , _
                        subItems() as String)
```

InsertItem 方法的语法包含下面部分：

（1）ListViewName 必需的。对象表达式，其值是 ListItems 集合；

（2）Index 必需的。指定在何处插入 ListItem 的整数。若未指定索引，则将 ListItem 添加到 ListItems 集合的末尾；

（3）Text 必需的。指定加入 ListItem 的标题；

（4）imageIndex 可选的。当 ListView 控件设为图标视图时，此整数设置从 ImageList 控件中选定欲显示的图标；

（5）subItems() 可选的。当 ListView 控件为报表视图时，用来设置列标题的数组。

注意：设置 imageIndex 属性之前必须先初始化。有两种初始化方法：在设计时，使用 ListView 控件属性页初始化；或在运行时，使用下列代码初始化：

```
ListView1. LargeImageList = ImageList1    '假设 Imagelist 为 ImageList1。
ListView1. SmallImageList = ImageList2
```

如果列表尚未排序，则可使用 index 参数将 ListItem 对象插入到任意位置。如果列表已排序，则将忽略 index 参数并根据排序顺序把 ListItem 对象插入到适当的位置。

若未提供 index，则 ListItem 对象将被添加一个索引，此索引等于集合中 ListItem 对象的数目加 1。

4. 在报表视图中显示 ColumnHeaders

在属性页中选择"Columns"选项，则弹出一个列标题编辑对话框（ColumnHeader Collection Edit），如图 4.32 所示。

点击"Add"按钮，将会在成员（Members）列表中增加一个列表题，然后通过对话框右边的属性可以设置列标题的名称（Name）、修饰语（Modifiers）、文本（Text）、文本排列（TextAlign）以及列宽（Width）。

图 4.32　列标题对话框

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

4.13　TreeView 控件

一个 TreeView（树型结构）控件是显示结点（Node）对象的等级体系结构，每个 Node 对象包含了一个标签和可选的点位图。TreeView 控件通常用于显示文档头、索引中的条目、磁盘上的文件和目录或者可以显示为等级结构的各种其他信息。Windows 资源管理器左侧部分就是一个典型的树形结构。TreeView 控件在工具箱中的图标如图 4.33 所示。

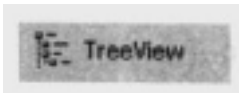


图 4.33　TreeView 控件

1. TreeView 在设计时设计好结点的组织结构

要建立如图 4.34 所示的树形结构可以按以下步骤进行：

（1）在工具栏中添加一个 TreeView 控件，名字为“TreeView1”以及一个 ImageList 控件，名字为“ImageList1”；

（2）在 ImageList 控件中添加位图文件，具体方法见 4.1.14 节 ImageList 控件；

（3）将 TreeView 属性页中的“ImageList”选项设置为“ImageList1”；

（4）点击 TreeView 属性页中的“Nodes”选项，则会弹出树结点编辑对话框，如图 4.35 所示；

（5）在树结点编辑框中编辑结点的层次关系。

图 4.34　树形结构演示

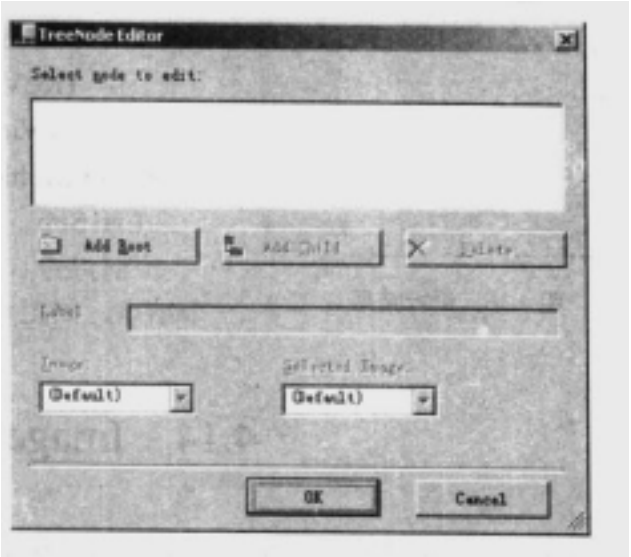


图 4.35　结点编辑框

2. 在树结点编辑框中编辑结点

可以按以下步骤：

（1）点击“Add Root”按钮，则在结点编辑框中出现了一个结点（默认名为 NodeN），可根据树型的需要，将其他根结点添加到这棵树上；

（2）选择需要添加子结点的结点，单击“Add Child”按钮，则在当前选中的结点上添加了一个子结点；

（3）选择需要添加位图的结点，然后在“Image”（图标）、“Selected Image”（选择图标）中，在 ImageList1 中选择需要的位图即可；

（4）点击“OK”按钮，在 TreeView1 中就会显示编辑的树了。

3. 运行时给树添加结点

给 TreeView 添加结点用到的是“Nodes.Add”方法，首先选定要添加子结点的结点才能应用这个方法，TreeView 中的结点的组织关系是父结点管理子结点的关系，也就是说，子结点组成的集合就是父结点的“Nodes”属性，子结点的“Index”属性，是根据其在子结点集合中的位置而决定的，而不是整棵树中结点的位置。根据这个特点，若想找到指定结点须按以下的语法：

TreeViewName.Nodes.Item(Index1).Nodes.Item(Index2)…

而添加结点的方法为：

TreeViewName.Nodes.Item(Index1).Nodes.Item(Index2)….Add("NodeText")

或

TreeViewName.Nodes.Item(Index1).Nodes.Item(Index2)….Add(objNode)

例如在上面 TreeView1 的 node2 结点中添加子结点“node2child1”，再给结点 node2child1 添加子结点“node2child1child1”，可以写成如下代码：

```
Protected Sub Button1_Click(ByVal sender As Object,
ByVal e As System.EventArgs)
    Dim nodx As New TreeNode()
    nodx.Text = "node2child1"
    nodx.ImageIndex = 2
    TreeView1.Nodes.Item(2).Nodes.Add(nodx)
    TreeView1.Nodes.Item(2).Nodes.Item(0).Nodes.Add(
"node2child1child1")
End Sub
```

运行后点击“OK”按钮可以看到 TreeView 中加的两个结点，如图 4.36 所示。

图 4.36　运行结果

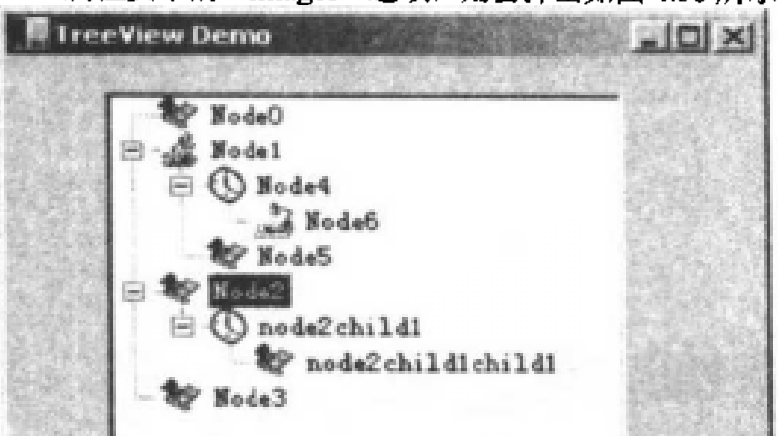
4.14　ImageList 控件

ImageList（图片列表）控件，主要就是为了给其他控件提供位图的支持，它仅仅是个为图标的承载工具，本身并不参与界面的设计，ImageList 控件在工具箱中的图标如图 4.37 所示。

向 ImageList 中添加位图文件：

点击 ImageList 属性页中的“Images”选项，则会弹出如图 4.38 所示的图片编辑对话框。

再点击“A



ImageList 控件中了。通过“Sor

属性。

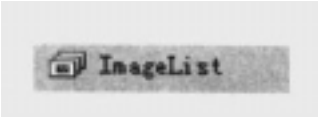


图 4.37　ImageList 控件

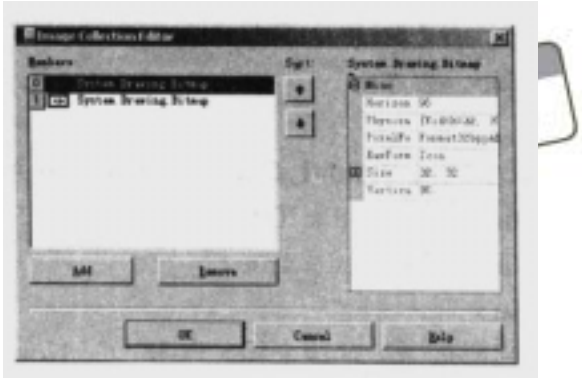


图 4.38　图片编辑框

4.15　Timer 控件

Timer（定时器）控件是用来产生一定的时间间隔，在每个时间间隔中都可根据应用程序的要求有相同或不同的事件或过程发生，Timer 控件在工具箱中的图标如图 4.39 所示。

1. 设置“Interval”、“Enable”属性

“Interval”属性是 Timer 控件最重要的属性之一，它决定着事件或过程发生的时间间隔，“Interval”属性以千分之一秒为基本单位，就是事件发生的最短间隔是一毫秒，但是这样的时间间隔对系统的要求很高，因此按时间精度的要求适当设置这个属性也是工程运行速度和可靠性的一种保证。

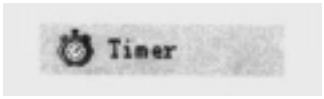


图 4.39　Timer 控件

“Enable”属性可以设置 Timer 控件是否为激活状态，一旦这个属性为“False”，那么 Timer 控件将失去作用。反之，如果在某个条件下将这个属性设置为“True”，Timer 控件将会被激活，事件和过程将间隔发生。

2. Timer 的 Tick 事件

使用 Tick 事件时，可用此事件在每次 Timer 控件时间间隔过去之后通知 Visual Basic 应该做什么：“Interval”属性指定 Tick 事件之间的间隔。无论何时，只要 Timer 控件的“Enabled”属性被设置为“True”，而且“Interval”属性大于 0，则 Tick 事件以“Interval”属性指定的时间间隔发生。如下例将实现标题栏滚动的效果。其中 Button1、Button2 为按钮，它们的“Text”属性分别为“Go Now”和“Stop Here”；Label1 为一个标签，“Text”属性为“Welcome To VB7”；Timer1 为一个定时器控件，如图 4.40 所示。

添加如下代码：

```
Protected Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
    label1.Left += 40
    label1.Left = label1.Left Mod Me.Width
End Sub

Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
```

超星阅览器指南
使用本复制品
请尊重相关知识产权!

```
        timer1.Enabled = False
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        timer1.Interval = 100
        timer1.Enabled = True
    End Sub
```

添加代码后，运行该工程，点击“Go Now”按钮，即可看到标题栏在滚动，点击“Stop Here”按钮，则标题栏停止滚动。

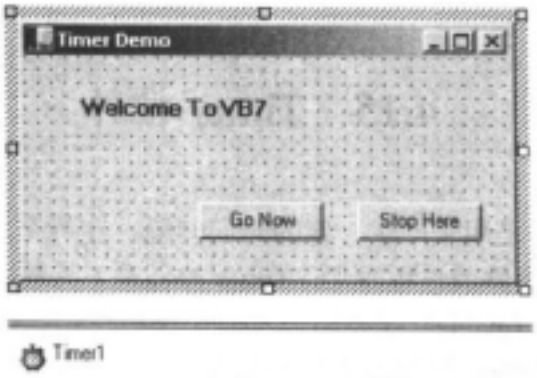


图 4.40 Timer 控件演示

4.16 HScrollBar、VScrollBar 控件

有了 HScrollBar（水平滚动条，如图 4.41 所示）和 VScrollBar（垂直滚动条，如图 4.42 所示），就可在应用程序的窗体或控件容器中水平或垂直滚动，相当方便地巡视一长列项目或大量信息。

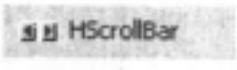


图 4.41 HScrollBa 控件

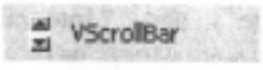


图 4.42 VScrollBar 控件

水平、垂直滚动条控件不同于 Windows 中内部的滚动条或 Visual Basic 中那些附加在文本框、列表框、组合框或 MDI 窗体上的滚动条。无论何时，只要应用程序或控件所包含的信息超过当前窗口（或者设置了 ScrollBars 属性的 TrueBox 控件）所能显示的信息，那些滚动条就会自动出现。在较早的 Visual Basic 版本中，通常把滚动条作为输入设备。但目前的 Windows 界面指南则建议用滑块取代滚动条作为输入设备。滚动条在 Visual Basic 中仍然有价值，因为它为那些不能自动支持滚动的应用程序和控件提供了滚动功能。

1. Scroll Bar 控件如何工作

滚动条控件用“Scroll”和“ValueChange”事件监视滚动框（有时用拇指替代）沿滚动

条的移动。“ValueChange”在滚动框移动后发生；“Scroll”在 ValueChange 发生后释放鼠标时发生。在单击滚动箭头或滚动条时，也是先发生 ValueChange 事件，再发生 Scroll 事件。下面的代码中，Scroll 事件将当前窗体的标题设置为 Value 值，ValueChange 事件将当前窗体的标题设置为两倍的 Value 值：

```
Public Sub HScrollBar1_Scroll(ByVal sender As Object, _
ByVal e As System.WinForms.ScrollEventArgs) Handles HScrollBar1.Scroll
        Me.Text = CStr(hscrollbar1.Value)
End Sub
Public Sub HScrollBar1_ValueChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles HScrollBar1.ValueChanged
    Me.Text = CStr(hscrollbar1.Value * 2)
End Sub
```

2. “Value”属性

“Value”属性（缺省值为 0）是一个整数，它对应于滚动框在滚动条中的位置。当滚动框位置在最小值时，它将移动到滚动条的最左端位置（水平滚动条）或顶端位置（垂直滚动条）。当滚动框在最大值时，它将移动到滚动条的最右端或底端位置。同样，滚动框取中间数值时将位于滚动条的中间位置。除了可用鼠标单击改变滚动条数值外，也可将滚动框沿滚动条拖动到任意位置。结果取决于滚动框的位置，但总是在用户所设置的“Minimum”和“Maximum”属性之间。

3. “LargeChange”和“SmallChange”属性

为了指定滚动条中的移动量，对于单击滚动条的情况可用“LargeChange”属性，对于单击滚动条两端箭头的情况可用“SmallChange”属性。滚动条的“Value”属性增加或减少的长度是由“LargeChange”和“SmallChange”属性设置的数值。要设置滚动框在运行时的位置，可将“Value”属性设为“Minimum”到“Maximum”中的某个数值。

4.17　ProgressBar 控件

ProgressBar（进程条）控件，是个水平放置的指示器，直观地显示某个操作正进行了多长时间。进度条并不显示计算机执行某项特定任务要花多少分钟或秒钟，它提供的是直观的视觉反馈，使用户确信没有理由中止操作或关掉计算机。进度条能减轻与等待复杂的运算结果有关的紧张和不安，提供了测量某项计算任务进度的切实措施。ProgressBar 在工具箱中的图标如图 4.43 所示。

1. Value、Maximum 和 Minimum 属性

“Value”属性决定该控件被填充多少。其“Maximum”和“Minimum”属性设置该控件的界限。要进行需要几秒钟才能完成的操作时，就要使用 ProgressBar 控件。同时还必须知道该过程到达已知端点需要持续多长时间，并将其作为该控件的“Maximum”属性来设置。

要显示某个操作的进展情况，“Value”属性将持续增长，直到达到了由“Maximum”

属性定义的最大值。这样该控件显示的填充块的数目总是“Value”属性与“Maximum”和“Minimum”属性之间的比值。

例如，如果“Minimum”属性被设置为“1”，“Maximum”属性被设置为“100”，“Value”属性为“50”，那么该控件将显示百分之五十的填充块，如图 4.44 所示。

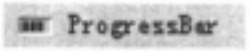


图 4.43　ProgressBar 控件

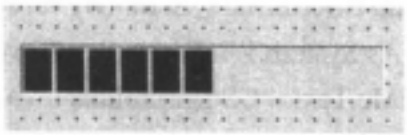


图 4.44　50%的进程条

使用本复制品
请尊重相关知识产权!

2. 将“Maximum”属性设置为已知的界限

要对 ProgressBar 进行编程，则必须首先确定“Value”属性攀升的界限。例如，如果正在下载文件，并且应用程序能够确定该文件有多少千字节，那么可将“Maximum”属性设置为这个数。在该文件下载过程中，应用程序还必须能够确定该文件已经下载了多少千字节，并将“Value”属性设置为这个数。

4.18　ToolBar 控件

ToolBar（工具栏）控件包含用来创建工具栏的 ToolBarButton 对象的集合。工具栏可与应用程序相关联。ToolBar 在工具箱中的图标如图 4.45 所示。

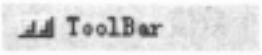


图 4.45　ToolBar 控件

一般情况下，工具栏中的按钮与应用菜单中的菜单项相对应，可以用它们来访问应用程序最常用的功能和命令。要创建工具栏，必须先将 Button 对象加入 Buttons 集合，每个 Button 对象可以拥有可选的文本，或者（并且）拥有相关联的 ImageList 控件提供的图像。可以用“Text”属性为每个 Button 对象设置文本，用“ImageIndex”属性设置图像。在设计时，可通过 ToolBar 的“Buttons”对话框（如图 4.46 所示）加入 Button 对象。

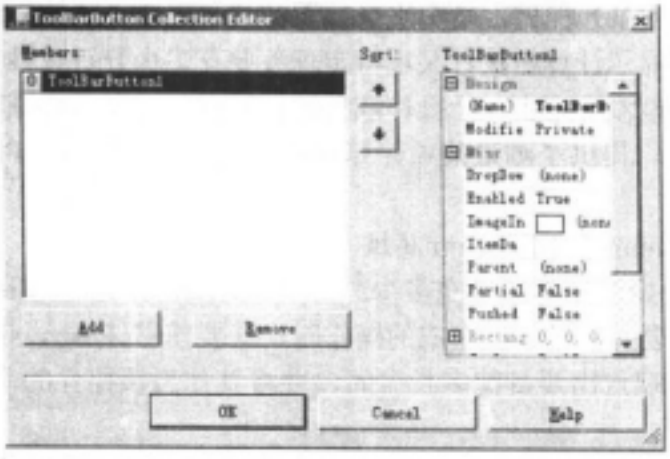


图 4.46　Buttons 对话框

在运行时，可以用 Add 和 Remove 方法将按钮加入 Buttons 集合或从中删除。要在设计时添加其他控件，只需在工具栏上将所需要的控件拖入工具栏即可。

1. Buttons 集合

ToolBar 控件由 Buttons 集合中的一个或多个 ToolBarButton 对象构成。在设计时和运行时均可创建 Button 对象。每个按钮可有图像、标题、工具提示，并且可以同时具有上述三种特性，如图 4.47 所示为一个工具栏的实例。

每个按钮对象还有“Style”属性（在下面讨论），该属性决定了按钮的行为。

2. 在运行时创建按钮

在运行时创建 ToolBarButton 对象的集合按以下的步骤：

图 4.47　工具栏实例

（1）声明 ToolBarButton 类型的对象变量。在添加 ToolBarButton 对象时，该变量用来包含对新产生对象的引用。该引用可以用来设置新 ToolBarButton 对象的各种属性。

（2）用带 Add 方法的 Set 语句，将对象变量赋值为新 ToolBarButton 对象。

（3）用该对象变量设置新 ToolBarButton 对象的属性。

下面的代码在 Button 对象的 Click 事件中创建一个 ToolBarButton 对象，然后设置新的 ToolBarButton 对象的“ImageIndex”、“Text”、“ToolTipText”和“Style”属性。

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim MyButton As New ToolBarButton()
    ToolBar1.Buttons.Add(MyButton)
    MyButton.ImageIndex = 1
    MyButton.Text = "Left"
    MyButton.ToolTipText = "Left1"
    MyButton.Style = ToolBarButtonStyle.PushButton
End Sub
```

4.19　StatusBar 控件

StatusBar（状态栏）控件是由若干个面板构成的框架，用它可以显示出应用程序的运行状态。该控件最多可以包含 16 个框架。另外该控件具有一种“简单”样式（用“Style”属性设置），为了显示某些特殊的信息，可以用该样式将多个面板转换为单个面板。StatusBar 控件可以放置在应用程序的顶部、底部或侧面。该控件还可以随意地在应用程序的客户区中“漂浮”。StatusBar 控件在工具箱中的图标如图 4.48 所示。

1. Panel 对象和 Panels 集合

StatusBar 控件是由 Panels 集合构成的。在该集合中至多可包含 16 个 Panel 对象。每个对象可以显示一个图像和文本，如图 4.49 所示。

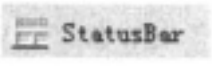


图 4.48　StatusBar 控件

在运行时，可以通过“Text”、“Icon”、“ToolTipText”、“Style”和“Width”属性动态地改变任何 Panel 对象的文本、

图像、提示文本、风格或宽度。要在设计时添加 Panel 对象，可以用鼠标单击 StatusBar 控件的“Panels”属性，即可打开“StatusBarPanel Collection”分格设计对话框，如图 4.50 所示。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

图 4.49　StatusBar 实例

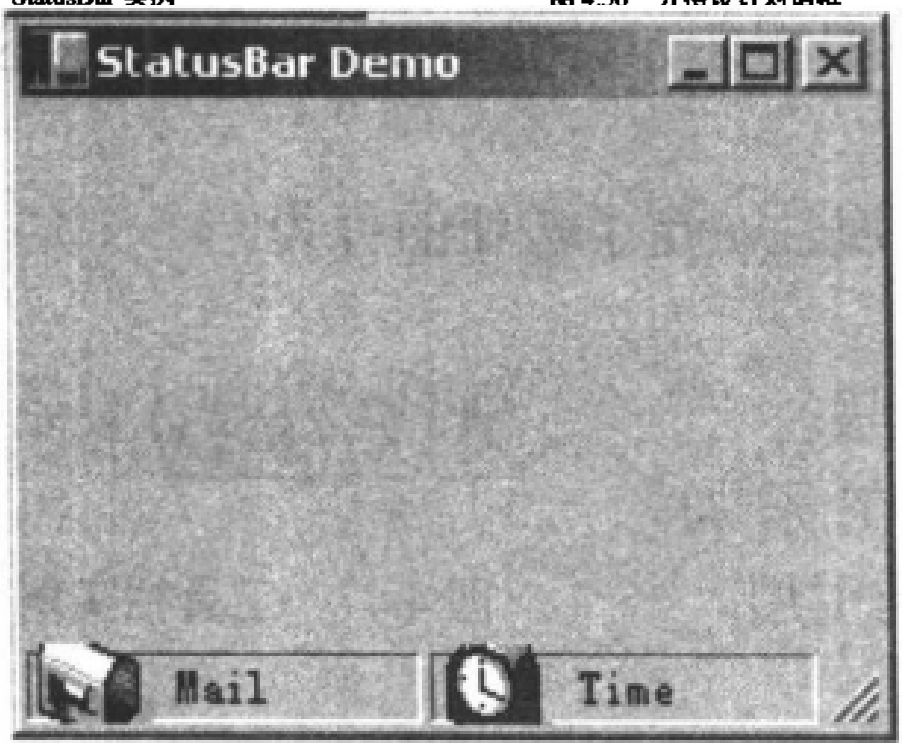


图 4.50　分格设计对话框

使用该对话
在运行时使
对象变量，然后
Protected Su
Dim M
StatusB
MyPan
MyPan
MyPan
End Sub
2. 改变 Pa
有两个属性
分格栏的宽度，
以及“Sunken”，

类型的

th”设置

“None”

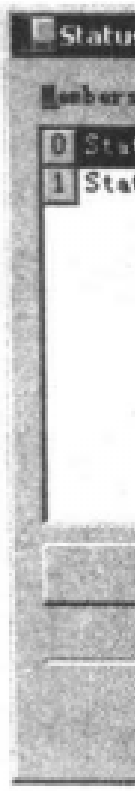


图 4.51　BorderStyle 的三个设定值

第 5 章　应用程序界面

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

本章包括：
界面样式
关于窗体的其他描述
对话框
界面设计的基本原则

用户界面是一个应用程序最重要的部分，它是最直接的现实世界。对用户而言，界面就是应用程序，他们感觉不到幕后正在执行的代码。不论花多少时间和精力来编制和优化代码，应用程序的可用性仍然依赖于界面。设计一个应用程序时，需要做出有关界面的若干决定。应该使用单文档还是多文档样式?需要多少不同的窗体?菜单中将包含什么命令?要不要使用工具栏重复菜单的功能?提供什么对话框与用户交互?需要提供多少帮助?在开始设计用户界面之前，需要考虑应用程序的目的。经常使用的主要应用程序，其设计应该与只是偶尔使用的不同。用来显示信息的应用程序与用来收集信息的应用程序的需求不同。预期的用户也应该影响设计。目标是针对初学者的应用程序，它的设计要求简单明了，而针对有经验的用户却可以复杂一些。读者所使用过的其他应用程序也会影响他们对应用程序操作方式的期望。如果计划发布到全球，那么语言与文化也是设计所必须考虑的部分。用户界面的设计最好是作为一个反复过程进行，很难在第一遍就能提出一个完美的设计。本章将介绍在 Visual Basic 中设计用户界面的过程，同时介绍在为用户创建重要应用程序时需用的一些工具。

5.1　界 面 样 式

界面设计是开发中最重要的方面，并将涉及到整个开发队伍。有效的界面设计经常是预见的过程，设计目标是开发者根据自己对用户需求的理解而制定的。设计界面的艺术综合了技术、艺术，心理学上的技能。优秀的界面简单且用户乐于使用，这意味着设计需适应硬件的局限。

以用户的角度来看，他所关心的应用程序首先就是体现在界面上，而真正在运行时执行的代码他是感觉不到的，也不会去关心。界面总是给用户最直观的印象，不管程序做得多么巧妙，算法设计得多么高效，如果没有好的应用程序的界面，它们的价值都是无法体现。相反地，一个好的风格的用户程序界面往往可以一定程度上的掩盖程序设计的缺陷。

在设计一个应用程序界面时，需要大体确定一个界面的雏形。比如是使用单文档界面（如图 5.1 所示）还是多文档界面；包含多少个独立的窗体；菜单中将包含什么选项；工具栏是否有必要重复菜单的功能；与用户交互时使用何种形式；应用程序能够提供多少帮助信息。在开始设计用户界面之前，需要考虑应用程序要完成的功能以及使用该应用程序的用户，

根据要完成的功能和用户的不同考虑的问题也有所不同，比如：经常使用的主要应用程序，其设计应该与只是偶尔使用的不同。用来显示信息的应用程序与用来收集信息的应用程序的需求不同。针对初学者的应用程序，它的界面设计要求简单明了，而针对有经验的用户却可以复杂一些，比如包括一些快捷键等等。读者所使用过的其他应用程序也会影响他们对应用程序操作方式的期望。如果计划发布到全球，那么语言与文化也是设计所必须考虑的部分。用户界面的设计最好是作为一个反复过程进行，很难在第一遍就能提出一个完美的设计。本章将介绍在 Visual Basic 中设计用户界面的过程，同时介绍在为用户创建重要应用程序时需用的一些工具。

而多文档界面的实例有 Microsoft Access 和 Microsoft Word 这样的应用程序，它们允许同时显示多个文档，每一个文档都显示在自己的窗口中，在 VB 里，MDI(Multiple Document Interface)窗体是这样定义的："MDI 窗体作为一个程序的后台窗口，包含着 IsMDIContainer 属性为 True 的窗体"。在一个 VB 程序中，至多只能存在一个 MDI 父窗体，可以有多个 MDI 子窗体，如图 5.2 所示为 MDI 的 Microsoft Access 实例：

按照是否包含有能在窗口或文档之间进行切换的子菜单的 Windows 菜单项，就能识别出 MDI 应用程序。要决定使用哪种界面样式，需要根据应用程序的功能。一个图书管理系统的应用程序可能要用多文档界面 MDI 样式，一个管理员很可能会同时处理一类以上图书信息的录入，或者需要对两本书信息进行比较；日历程序最好设成 SDI（单文档界面）样式，因为不大需要同时打开一个以上日历；在极少的情况下，也可以再打开一个单文档界面应用程序的实例。

图 5.1　单文档界面 —— 写字板　　图 5.2　多文档界面 —— Access 数据库

应用程序 SDI 样式更常用，本书的大多数实例都采用 SDI 应用程序。关于创建 MDI 应用程序有一些需要单独考虑的问题和技巧，放在本章后面"创建 MDI 应用程序"一节。除了以上两个最常用的界面样式 SDI 与 MDI 外，还有第三种界面样式，这就是资源管理器样式的界面如图 5.3 所示。资源管理器样式界面是包括有两个窗格或者区域的一个单独的窗口，通常是由左半部分的一个树型的或者层次型的视图和右半部分的一个显示区所组成，如在 Microsoft Windows 的资源管理器中所见到的那样。这种样式的界面可用于定位或浏览大量的文档、图片或文件。

资源管理器样式越来越被更多的应用程序所使用，但是在 Visual Basic 中，并没有直接

超星阅览器提醒您：
使用本复制品
须遵守相关知识产权！

的这个选项，一般说来是由 TreeView 以及一些特殊的文本框或第三方控件（如 Active ToolBar）组合而成，并且需要一定的代码支持。

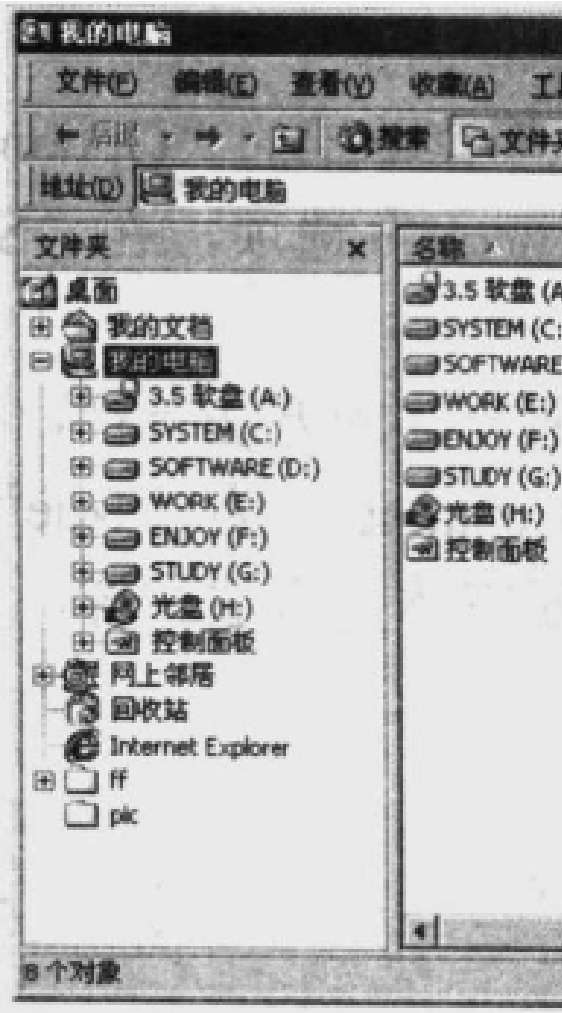


图 5.3 Windows 资源管理器

5.2 多文档界面应用程

多文档界面（MDI）允许创建在单个容器窗体（即父窗体序。像 Microsoft Access 与 Microsoft Word 这样的应用程序就具有多文档界面。MDI 应用程序允许用户同时显示多个文档，每个文档显示在各自的窗口中。文档或子窗口被包含在父窗口中，父窗口为应用程序中所有的子窗口提供工作空间。例如：Microsoft Access 允许创建并显示不同样式的多文档窗口。每个子窗口都被限制在 Access 父窗口的区域之内。当最小化 Access 时，所有的文档窗口也被最小化，只有父窗口的图标显示在任务栏中。子窗体就是某一个窗体类的一个实例。一个应用程序可以包含许多相同、相似或者不同样式的 MDI 子窗体。在运行时，子窗体显示在 MDI 父窗体工作空间之内（其区域在父窗体边框以内及标题与菜单栏之下）。当子窗体最小化时，它的图标显示在 MDI 窗体的工作空间之内，而不是在任务栏中，如图 5.4 所示。

注意：一个应用程序也可以包括标准的、不是包含在 MDI 窗体之内的非 MDI 窗体。MDI 应用程序中标准窗体的典型用法是显示模式的对话框。MDI 窗体就是一个普通窗体类的实例，只不过要将这个窗体实例的“MDIParent”指向其父窗体。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

5.2.1　创建 MDI 应用程序

使用下列步骤来创建 MDI 窗体及其子窗体：

（1）设置工程为 MDI 环境。点击“Tools”菜单中的“Option...”则会弹出如图 5.5 所示的对话框，选择“Environment”中的“General”选项，在“Settings”中，选择“MDI environment”，其他选项取默认值。

（2）创建应用程序的父窗体。与创建普通窗体的方法一样，在设计好菜单、工具栏以及状态栏等基本控件后，再设置属性“IsMDIContainer”为“True”就可以了。

（3）创建应用程序的子窗体，与创建普通窗体的方法一样，可以根据不同子窗体的需要，设计不同的窗体。

图 5.5　“Option”对话框

注意：设置“MDI environment”环境时，必须重新启动 Visual Basic.NET 集成开发环境才会使这一设置有效。

1. 生成子窗体的实例

子窗体一定是一个普通窗体类的实例，下面代码演示了如何生成一个子窗体的实例：

```
Dim i As Integer
Protected Sub MenuItem3_Click(ByVal sender As Object, _ByVal e As System.EventArgs)
    Dim MDIChild As New Form2()                    ' 声明一个窗体的实例
    MDIChild.MDIParent = Me                        ' 定义
    MDIChild.Text = "Untitle " & CStr(i + 1)       ' 设置
    i += 1
    MDIChild.Show()                                ' 显示
End Sub
```

2. MDI 窗体运行时的特性

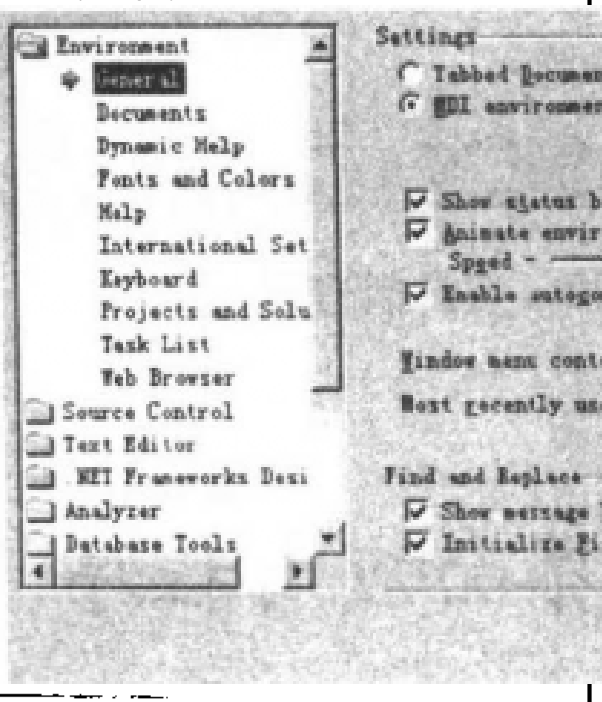
在运行时，MDI 窗体及其所有的子窗体都呈现特定的

（1）所有子窗体均显示在 MDI 窗体的工作空间内。

窗体和改变子窗体的大小，不过，它们被限制于这一工作

（2）当最小化一个子窗体时，它的图标将显示于 MDI

5.6 所示。当最小化 MDI 窗体时，此 MDI 窗体及其所有

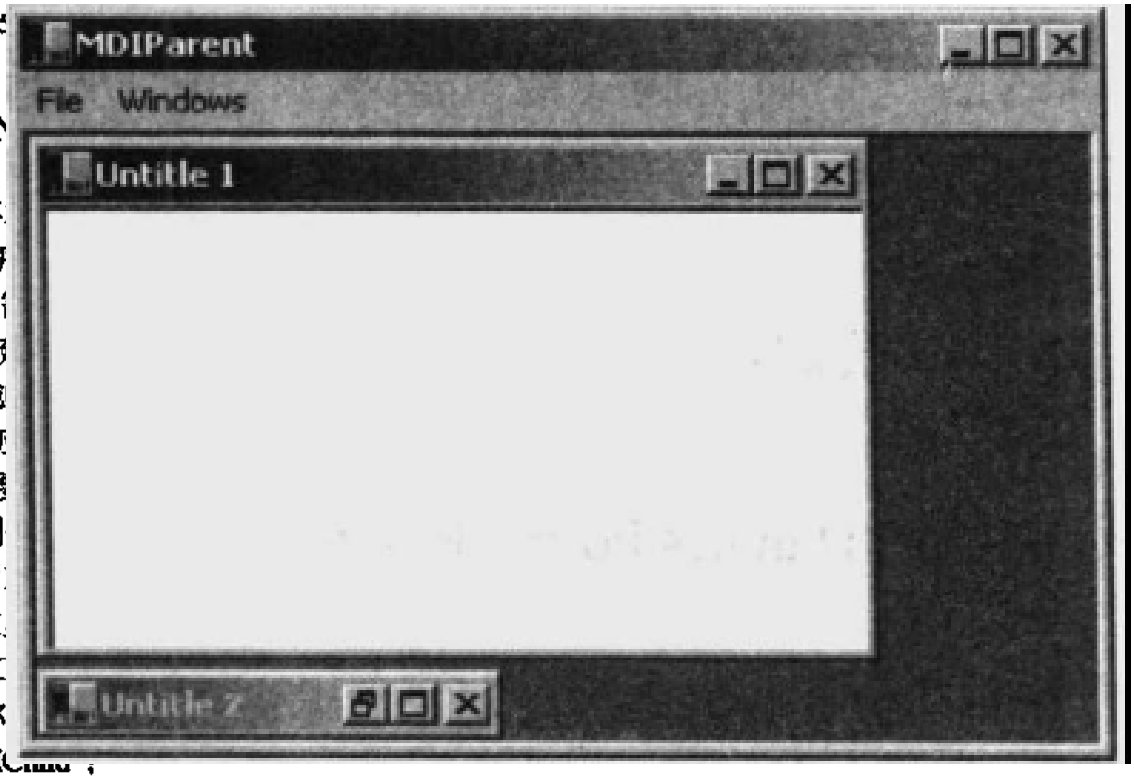


原 MDI 窗体时，MDI 窗体及其所有子窗体将按最小化之前的状态显示出来。

（3）当最大化一个子窗体时，它的标题会与 MDI 窗体的标题组合在一起并显示于 MDI 窗体的标题栏上，如图 5.7 所示。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

图5



5.2.2 MDI N

MDI 记事
记事本应用程
行时，若用户
用程序就会创
Basic 中为了
体。设计时，
文档的模板。

（1）在制

（2）从“
称等信息，这

（3）从“
窗体文件名称
名称为“MDI

（4）在 MDIChild 上创建一个文本框（TextBox1）。设置 TextBox1 的“MultiLine”和“WordWrap”属性为“True”；

（5）为 MDIParent 中添加一个 MainMenu 控件，由“File”、“Edit”、“Windows”两个菜单组成，其中 File 菜单下包含一个“New”子菜单，Edit 菜单下包含一个“Copy”子菜单；

（6）在 mnuFileNew_Click 过程中增加以下代码：

```
Private Sub mnuFileNew_Click (ByVal sender As Object, _
                              ByVal e As System.EventArgs)
    ' 创建名为 NewDoc 的窗体 Form1 的一个新实例。
    Dim NewDoc As New MDIChild() ' 显示此新窗体。
```

```
    NewDoc.MDIParent = Me
    NewDoc.Text = "NewDoc"
    NewDoc.Show()
End Sub
```

这个过程创建并显示 MDIChild 的名为 NewDoc 的一个新实例（或其副本）。每当从 File 菜单中选取 New 命令时，将会创建一个与 MDIChild 完全相同的实例，它包含 MDIChild 所包含的所有控件和代码。

（7）给 MDIChild 窗体的 Form_Resize 过程添加以下代码：

```
Public Sub Form2_Resize(ByVal sender As Object, _
                        ByVal e As System.EventArgs) Handles Form2.Resize
    TextBox1.Height = Me.Height
    TextBox1.Width = Me.Width
End Sub
```

Form_Resize 事件过程的代码，像 MDIChild 中的所有代码一样，能为 MDIChild 的每一个实例所共享。当显示窗体的几个副本时，每个窗体都能识别各自的事件。当一个事件出现时，该事件过程的代码就会被调用。由于相同的代码为每个实例所共享，关于调用该代码的窗体是如何引用的，尤其是每个实例都具有相同的名字（Form2）时。这个问题将在本章下一节的“使用 MDI 窗体及其子窗体”中讨论。

（8）按【F5】键可运行该应用程序。

5.2.3　MDI 窗体的进一步研究

1. 使用 MDI 窗体及其子窗体

当 MDI 应用程序在一次会话中要打开、保存和关闭几个子窗体时，应当能够引用活动窗体和保持关于子窗体的状态信息。这个主题描述了一些用来指定活动子窗体或者控件、加载和卸载 MDI 窗体及其子窗体、以及保持子窗体的状态信息的编码技巧。

2. 指定活动子窗体或控件

有时要提供一条命令，它用于对当前活动子窗体上具有焦点的控件进行操作。例如，假设想从子窗体的文本框中把所选文本复制到剪贴板上。在 Mdinote.vbproj 示例应用程序中，“Edit”菜单上的“Copy”项的 Click 事件将会调用 EditCopyProc，它是把选定的文本复制到剪贴板上的过程。由于应用程序可以有同一子窗体的许多实例，EditCopyProc 需要知道使用的是哪一个窗体。为了指定这一点，使用 MDI 窗体的“ActiveForm”属性，该属性可以返回具有焦点的或者最后被激活的子窗体。

注意：当访问“ActiveForm”属性时，至少应有一个 MDI 子窗体被加载或可见，否则会返回一个错误。当一个窗体中有几个控件时，也需要指定哪个控件是活动的。像“ActiveForm”属性一样，“ActiveControl”属性能返回活动子窗体上具有焦点的控件。下面是副本例程的示例，从子窗体菜单、MDI 窗体菜单或者是工具栏按钮上可对它进行调用：

```
Private Sub EditCopyProc ()
    ' 将当前激活窗体的“Text”设置为“Actived”。
    Form2.ActiveForm.ActiveControl.Text = "Actived"
```

End Sub

假如正在编写被多个窗体实例调用的代码，不用窗体标识符访问窗体的控件或属性是一个好办法。例如，用 Text1.Height 引用 Form1 上文本框的高度，而不是使用 Form1.Text1.Height。这样，该代码总是影响当前窗体。在代码中指定当前窗体的另一种方法是用 Me 关键字。用 Me 关键字来引用当前其代码正在运行的窗体。当需要把当前窗体实例的引用参数传递给过程时，这个关键字很有用。

3. 设置子窗体的基本属性

MDI 子窗体在其装载时，其初始的高度、宽度和位置取决于设计时基本窗体的高度、宽度和位置。通过设置基本窗体的“BorderStyle”、“MaximizeBox”、“MinimizeBox”等属性，可以设定子窗体的基本属性。

4. 维护子窗体的状态信息

在用户决定退出 MDI 应用程序时，必须有保存信息的机会。为了使其能够进行，应用程序必须随时都能确定自上次保存以来子窗体中的数据是否有改变。通过在每个子窗体中声明一个公用变量来实现此功能。例如，可以在子窗体的声明部分声明一个变量：

```
Public blnchange As Boolean
```

TextBox1 中的文本每改变一次时，子窗体文本框的 Change 事件就会将“blnchange”设置为“True”。可添加此代码以指示自上次保存以来 TextBox1 的内容已经改变。

```
Protected Sub TextBox1_TextChanged(ByVal sender As Object, _ByVal e As System.EventArgs)
    blnchange = False
End Sub
```

反之，用户每次保存子窗体的内容时，文本框的 Change 事件就将“blnchange”设置为“False”，以指示 TextBox1 的内容不再需要保存。在下列代码中，假设有一个叫做“保存”（mnuFileSave）的菜单命令和一个用来保存文本框内容的名为 FileSave 的过程：

```
Protected Sub mnuFileSave _Click(ByVal sender As Object, _
                                 ByVal e As System. EventArgs)
    '保存 Text1 的内容。
    Call FileSave()  '设置状态变量。
    blnchange = False
End Sub
```

5.3　关于窗体的其他描述

学会在不同环境下，使用不同样式的窗体很重要，除此之外，知道如何控制窗体什么时候在应用程序中显示也很重要，能够熟练运用应用程序启动、卸载时的技巧以及掌握必要的处理手段也是非常必要的。下面介绍一些这方面的技巧。

1. 设置启动窗体

缺省的情况下，应用程序中把第一个设计的窗体指定为启动窗体。在运行应用程序的时

候，这个窗体首先被显示出来（最先执行的代码是该窗体的“InitializeComponent”事件中的代码）。如果想在应用程序启动时显示别的窗体，那么就得改变启动窗体。要改变启动窗体，请按照以下步骤执行：

（1）点击工程文件的名称（.vbproj）；

（2）从“Project”菜单中，选取“Properties”；

（3）选取“General”；

（4）在“Startup object”列表中，选取要作为新启动窗体的窗体；

（5）点击“OK”。

超星阅览器提示：
使用本复制品
请尊重相关知识产权！

2. 没有启动窗体时的启动

有时候不需要应用程序在启动的时候加载任何窗体。例如：可能想先运行装入数据文件的代码，然后再根据数据文件的内容决定显示几个不同窗体中的一个。要做到这一点，可在标准模块（Module）中创建一个名为“Main”的子过程，在这个过程中添加必要的代码，如下面的例子所示，窗体“frmMain”为主窗体，“frmPassword”为显示密码框的窗体，当用户状态（UserStatus）为“1”时，显示主窗体，其他则显示密码框窗体：

```
Sub Main()
    Dim intStatus As Integer
    '调用一个函数过程来检验用户状态。
    intStatus = GetUserStatus'根据状态显示某个启动窗体。
    If intStatus = 1 Then
        frmMain.Show()
    Else
        frmPassword.Show()
    End If
End Sub
```

这个过程必须是一个子过程，且不能在窗体模块内。欲将 Sub Main 过程设为启动对象，可从“Project”菜单中，选取“Properties”，再选“General”，然后从“Startup object”列表框中选定“Sub Main”。

3. 设定启动时的快速显示

如果初始化一个应用程序需要很长时间，比如要从数据库中提取大量的数据或者要装入一些大型位图，如果应用程序很长时间都处于等待状态，很可能会使用户怀疑是否应用程序死掉了，这时就需要在启动时给出一个快速显示，来实时地跟踪应用程序初始化的信息。快速显示是一种窗体，它通常显示的是诸如应用程序名，版权信息和一个简单的位图等内容。启动 Visual Basic 时所显示的屏幕就是一个快速显示。要显示快速显示，同样需用“Sub Main”过程作为启动对象，并用“Show”方法显示该窗体：

```
Private Sub Main()
    '显示快速显示。
    frmSplash.Show()        '在此处添加启动过程。
    ...                     '显示主窗体并卸载快速显示。
    frmMain.Show()
    frmSplash.Close
```

End Sub

当一些启动例程正在执行时，快速显示能吸引用户的注意，造成应用程序装载很快的错觉。当这些启动例程完成以后，可以装入第一个窗体并使其快速显示卸载。对于快速显示的设计来说，以尽量简单为好。如果使用大量位图或者大量控件，则快速显示本身载入将会变慢。

4. 如何终止应用程序

当所有窗体都已关闭并且没有代码仍在运行时，则应用程序就停止运行。如果最后一个可见窗体关闭时仍有隐藏窗体存在，那么，应用程序表现为已经结束了（因为没有可见的窗体），可实际上应用程序仍在继续运行，直至所有隐藏窗体都关闭为止。之所以出现这种情况，是因为对已卸载窗体的属性或控件的任何访问，都将导致隐含地、不予显示地加载那个窗体。避免出现关闭应用程序时的这类问题，最好的办法是确保所有的窗体都已卸载。“End”语句使应用程序立即结束：在“End”语句之后的代码不会执行，也不会再有事件发生。特别是，Visual Basic 将不执行任何窗体的“Hide”、“Close”事件过程。对象的各个引用将被释放。除“End”语句以外，“Stop”语句可以暂停一个应用程序。然而，“Stop”语句只能在调试时使用，因为它不释放对象的引用。

5.4 对 话 框

在基于 Windows 的应用程序中，对话框被用来：

（1）提示用户提供应用程序继续执行所需要的数据。

（2）向用户显示信息。例如，在 Visual Basic 中，用“打开文件”对话框来显示已存在的工程。

Visual Basic 中的“关于”对话框也是一个如何使用对话框来显示信息的例子。在菜单栏上单击“帮助”，选择“关于 Visual Basic”菜单项，则显示“关于”对话框。

5.4.1 模式与无模式的对话框

对话框不是模式就是无模式的。模式对话框，在可以继续操作应用程序的其他部分之前，必须被关闭（隐藏或卸载）。例如，如果一个对话框，在可以切换到其他窗体或对话框之前要求先单击“确定”或“取消”，则它就是模式的。Visual Basic 的“关于”对话框是模式的。显示重要消息的对话框总应是模式的 —— 那就是说，在继续做下去之前，总是要求用户应当先关上对话框或者对它的消息作出响应。无模式的对话框允许在对话框与其他窗体之间转移焦点而不用关闭对话框。当对话框正在显示时，可以在当前应用程序的其他地方继续工作。无模式对话框很少使用。Visual Basic 中“编辑”菜单中的“查找”对话框就是一个无模式对话框的实例。无模式对话框用于显示频繁使用的命令与信息。要将窗体作为模式对话框显示，使用 ShowDialog 方法。例如：

```
'将 frmAbout 作为模式对话框显示。
frmAbout.ShowDialog()
```

要将窗体作为无模式对话框显示，则使用 Show 方法。例如：

```
'将 frmAbout 作为无模式对话框显示。
frmAbout.Show()
```

注意：如果窗体显示为模式对话框，则只有当对话框关闭之后，在 ShowDialog 方法后的代码才能执行。然而，当窗体被显示为无模式对话框时，在该窗体显示出来以后，Show 方法后面的代码紧接着就会执行。ShowDialog 方法有一个可选参数（owner）可用来指定窗体的父子关系。可将某个窗体名传给这个参数，使得这个窗体成为新窗体的拥有者。要显示一个窗体，作为另一个窗体的子窗体。例如：

```
' 将 frmAbout 显示为 frmMain 的模式子窗体。
frmAbout. ShowDialog(frmMain)
```

5.4.2 预定义对话框的使用

在应用程序中添加对话框最容易的方法是使用预定义对话框，因为不必考虑设计、装载或者显示对话框方面的问题。然而，控件在其外观上要受到限制。预定义的对话框总是模式的。

Visual Basic 应用程序中经常用到的预定义对话框函数是“InputBox”和“MsgBox”。InputBox 函数在对话框中显示一个命令提示，并返回所接收的任何东西；MsgBox 函数在对话框中显示一条消息，并返回一个表示命令按钮被单击的值。

1. 用输入框来提示输入

应用“InputBox”函数请求提供数据。这个函数显示要求输入数据的模式对话框。图 5.9 所示的文本输入框提示输入登陆的密码：

要在应用程序运行的过程中显示图 5.8 所示的对话框，则需添加以下代码：

```
strPass = InputBox("Your", "Login")
```

注意：切记当使用 InputBox 函数时，对对话框的各部分的控制非常有限。只能改变标题栏中的文本、显示给用户的命令提示、对话框在屏幕上的位置以及它默认时显示的字符串，而大小，按钮的标题等信息都是无法改变的。

2. 用消息对话框显示信息

可以用 MsgBox 函数获得“是”或者“否”的响应，并显示简短的消息，比如：错误、警告或者对话框中的期待。看完这些消息以后，可选取一个按钮来关闭该对话框。如果文件不能打开，那么名为 Text Editor 的应用程序就会显示如图 5.9 所示的消息对话框。

图 5.8 InputBox 对话框

图 5.9 MsgBox 对话框

以下代码显示如图 5.9 所示的消息框：

```
MsgBox( "Error encountered while trying to open file, _
```

please retry.", Microsoft.VisualBasic.MsgBoxStyle.OKOnly, "Text Editor")

注意：所谓模式的，既可以局限于应用程序中，也可以局限于系统中。如果消息框的模式局限在应用程序中（缺省），则在这个对话框未消失之前不能切换到该应用程序的其他部分，但是可以切换到其他应用程序。在消息框未消失之前系统的模式消息框不允许切换到别的应用程序。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

5.4.3　用窗体作为自定义对话框

自定义对话框就是用户所创建的含有控件的窗体 —— 这些控件包括命令按钮、选项按钮和文本框 —— 它们可以为应用程序接收信息。通过设置属性值来自定义窗体的外观。也可以编写在运行时显示对话框的代码。要创建自定义对话框，可以从新窗体着手，或者自定义现成的对话框。如果重复过多，可以建造能在许多应用程序中使用的对话框的集合。对话框通常不包括菜单栏、窗口滚动条、最小化与最大化按钮、状态条或者尺寸可变的边框。下面的部分将讨论创建典型类型的对话框方法。

1. 添加标题

对话框应当有标识它的标题。要创建标题，设置该窗体的“Text”属性为将在标题条中显示的文本字符串。通常，这一步是在设计时使用“属性”窗口来完成的，但也可以用代码来完成这一步。例如：

```
frmAbout.Text = "About"
```

注意：如果想完全删除此标题栏，可以设置窗体的“ControlBox”、“MinimizeBox”和“MaximizeBox”为“False”；设置“BorderStyle”为尺寸不可变的设置（非“Sizable”和“SizableToolWiindow”），并设置“Text”为空字符串（""）。

2. 设置标准对话框的属性

一般来说，用户响应对话框时，先提供信息，然后用“确定”或者“取消”按钮关闭对话框。因为对话框是临时性的，用户通常不需要对它进行移动、改变尺寸、最大化或最小化等操作。其结果是：随新窗体出现的可变尺寸边框类型、“控制”菜单框、“最大化”按钮以及“最小化”按钮，在大多数对话框中都是不需要的。通过设置“BorderStyle”、“ControlBox”、“MaximizeBox”和“MinnimizeBox”属性，可以删除这些项目。例如，“关于”对话框可能使用以下的属性设置。属性设置效果“BorderStyle1”改变边框类型为固定的单个边框，防止对话框在运行时被改变尺寸；设置“ControlBox”为“False”，删除控制菜单框；设置“MaximizeBox”为“False”，删除最大化按钮，防止对话框在运行时被最大化；设置“MinnimizeBox”为“False”，删除最小化按钮，防止对话框在运行时被最小化记住，如果删除“控制”菜单框（ControlBox = False），则必须向用户提供退出该对话框的其他方法。实现的办法通常是在对话框中添加“确定”、“取消”或者“退出”按钮，并在隐藏或卸载该对话框的“Click”按钮事件中添加代码。

3. 添加和放置命令按钮

模式对话框必须至少包含一个退出该对话框的按钮。通常用两个按钮：其中一个按钮开始动作，而另一个按钮关闭该对话框而不做任何改变。典型状态是，这两个按钮的“Text”属性设置“OK”与“Cancel”。虽然“OK”与“Cancel”是最常用的按钮，其他的按钮标题组合也可使用。显示消息的对话框通常使用 Label 控件来显示错误消息或者命令提示，并

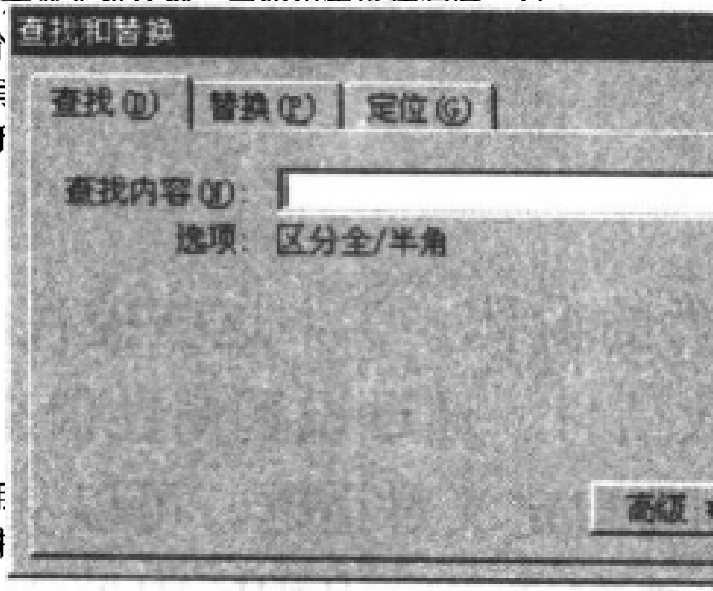
超星阅读器试用版
使用本复制品
请尊重知识产权!

且用一至两个按钮来执行动作。例如，也许给标识的“Text”属性赋予错误消息或者命令提示，而给两个按钮控件的“Text”属性指定“Yes”与“No”。当用户选取“Yes”，则发生一个动作，当选取“No”时，则发生另一个动作。这种类型对话框的按钮通常被放置在对话框的底部或右边，而顶部或左边的按钮为缺省按钮，如图 5.10 所示。

图 5.10　对话框中按钮的位置

4. 显示自定义对话框

使用与在应用程序中显示其他窗体同样的方法来显示对话框。当应用程序运行时，启动窗体会自动装入。如果要在应用程序中出现第二个



码来实现。同样，想要窗体或对话框消失，也要编写

选取“帮助”菜单中的“关于”菜单项时显示“关于

```
Private Sub mnuHelpAbout_Click ()
    '对话框显示为模式的。
    frmAbout.ShowDialog()
End Sub
```

5.4.4　各种显示类型的设计

Visual Basic 开发的应用程序是基于窗口的应用

即可以在许多不同显示分辨率与颜色浓度的计算机中

虑到这些因素。

1. 设计与分辨率无关的窗体

缺省情况下，当改变屏幕分辨率时，Visual Basic 不会改变窗体与控件的尺寸。这就意味着在分辨率为 1024×768 的屏幕上设计的窗体，在分辨率为 640×480 的屏幕中运行时可能会伸出屏幕的边界之外。如果想创建不管使用什么样的屏幕分辨率都能有相同比例的窗体和控件，必须在最低的分辨率下设计窗体，或者将改变窗体的代码添加到程序中去。避免尺寸问题的最简单的方法是在 640×480 的分辨率下设计窗体。

在运行时，Visual Basic 根据设计时的位置来放置窗体。如果设计时在 800×600 的分辨率上运行，并把窗体放到屏幕的右下角，则当它在比较低的分辨率下运行时该窗体可能看不见。为了避免这种情况的发生，可以在运行时用下面的 InitializeComponent 过程中的代码来设置窗体的位置：

```
Private Sub InitializeComponent ()
    Me.SetDesktopBounds( 0, 0 , 800 ,600)
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

End Sub

尽管设置窗体的 Left、Top、width、Height 的属性也能有同样的效果，但 SetDesktopBounds 方法只要一步就能完成。Visual Basic 使用与设备无关的度量单位（缇）是用来计算尺寸与位置的单位。

2. 设计不同浓度的颜色

在设计应用程序时，也需要考虑运行应用程序的计算机可能的颜色显示能力。有些计算机可以显示 256 色或更多的颜色，而另一些只能显示 16 种颜色。如果使用 256 色的调色板来设计窗体，那么在 16 色上显示时，抖动（模仿无效颜色的一种方法）会使窗体上的一些元素消失。为了避免这种情况，最好把应用程序使用的颜色局限于 Windows 标准的 16 种颜色。这些颜色由 Visual Basic 的颜色常数来表示（如 vbBlack、vbBlue 和 vbCyan 等）。如果在应用程序中需要用 16 种以上的颜色，那么对于文本、按钮以及其他界面元素仍然应坚持用标准颜色。

5.5　界面设计的基本原则

除非创建 Visual Basic 应用程序完全只供自己使用，否则创作的价值只能由其他人来评价。应用程序的用户界面对用户有极大的影响 —— 无论代码在技术上多么卓越，或者优化得多么的好，如果用户发现应用程序很难使用，那么用户就难于很好地接受它。作为程序员，毫无疑问，对计算机方面技术已非常熟悉，因而很容易忘记大多数用户不理解（而且也许并不在意）隐藏在应用程序后面的技术。把应用程序看作达到目的工具：完成任务的方法想象中应比没有计算机的帮助更容易。一个设计得好的用户界面把用户与基础技术隔离开来，从而使完成预定的任务变得很容易。在设计应用程序用户界面的过程中，需要时时想到用户。如何能无需指导就发现应用程序的各种各样的功能？当错误发生时，应用程序如何响应？在帮助或辅助用户方面将提供些什么？设计是否以一种艺术美来使用户高兴？以上这些问题的答案以及其他有关以用户为中心的设计问题，在本节中都将涉及到。

1. 界面设计的基础

不必成为创建用户界面的艺术家 —— 大多数用户界面设计的原则，与任意一门基础艺术课中所讲授的基础设计的原则相同。构图、颜色等的基本设计原则，就像它们应用在纸张或油画上一样，也能很好地在一台计算机的屏幕上应用。虽然 Visual Basic 能通过简单地将控件拖动并放置到窗体上而使得创建用户界面非常容易，但是，在设计之前稍微规划一下就能使应用程序的可用性有很大的改观。可能需先在纸上画出窗体开始设计，决定需要哪些控件，不同元素的相对重要性，以及控件之间的关系。

2. 构图：应用程序的观感与感觉

窗体的构图或布局不仅影响它的美感，而且也极大地影响应用程序的可用性。构图包括诸如控件的位置、元素的一致性、动感、空白空间的使用以及设计的简单性等因素。

3. 控件的位置

在大多数界面设计中，不是所有的元素都一样重要。仔细设计是很有必要的，以确保越是重要的元素越要更快地显现给用户。重要的或者频繁访问的元素应当放在显著的位置

上，而不太重要的元素就应当放在不太显著的位置上。在大多数语言中，用户习惯于在一页之中从左到右、自上到下地阅读。对于计算机屏幕也如此，大多数用户的眼睛会首先注视屏幕的左上部位，所以最重要的元素应当放在屏幕的左上部位。例如，如果窗体上的信息与客户有关，则它的名字字段应当显示在它能最先被看到的地方。而按钮，如“确定”或“下一个”，应当放置在屏幕的右下部位；用户在未完成对窗体的操作之前，通常不会访问这些按钮。把元素与控件分成组也很重要。尽量把信息按功能或关系进行逻辑分组。因为它们的功能彼此相关，所以定位数据库的按钮应当被形象地分成一组，而不是分散在窗体的四处。对信息也是一样，名字字段与地址通常分在一组，因为它们联系紧密。在许多情况下，可以使用框架控件来帮助加强控件之间的联系。

4. 界面元素的一致性

在用户界面设计中，一致性是一种优点。一致的外观与感觉可以在应用程序中创造一种和谐，任何东西看上去都那么协调。如果界面缺乏一致性，则很可能引起混淆，并使应用程序看起来非常混乱、没有条理、价值降低，甚至可能引起对应用程序可靠性的怀疑。为了保持视觉上的一致性，在开始开发应用程序之前应先创建设计策略和类型约定。诸如控件的类型、控件的尺寸、分组的标准以及字体的选取等设计元素都应该在事先确定。可以创建设计样板来帮助进行设计。在 Visual Basic 中有大量的控件可供使用，这可能使有人想使用所有的控件。为了避免这种引诱，应该选取能很好地适合特定应用程序的控件子集。虽然列表框、组合框、网格以及树等控件都可用来表示信息列表，最好还是尽可能使用一种类型。还有，尽量恰当地使用控件，虽然 TextBox 控件可以设置成只读并用来显示文本，但 Label 控件通常更适用于该目的。在为控件设置属性时请保持一致性，如果在一个地方为可编辑的文本使用白色背景，除非有很好的理由，否则不要在别的地方又使用灰色。在应用程序中不同的窗体之间保持一致性对其可用性有非常重要的作用。如果在一个窗体上使用了灰色背景以及三维效果，而在另一个窗体上使用白色背景，则这两个窗体就显得毫不相干。选定一种类型并在整个应用程序保持一致，即使这意味着要重新设计某些功能。

5. 动感：窗体与其功能匹配

动感是对象功能的可见线索。虽然读者对这个术语也许还不熟悉，但动感的实例四处可见。自行车上的把手，手放在它的上面，动感会将把手用手扣紧这件事显现出来。按下按钮、旋转旋钮和点亮电灯的开关等都能进行动感表示，一看到它们就可以看出其用处。用户界面也使用动感。例如，用在命令按钮上的三维立体效果使得它们看上去像是被按下去的。如果设计平面边框的命令按钮的话，就会失去这种动感，因而不能清楚地告诉用户它是一个命令按钮。在有些情况下，平面的按钮也许是适合的，比如游戏或者多媒体应用程序；只要在整个应用程序中保持一致就很好。文本框也提供了一种动感，用户可以期望带有边框和白色背景的框，框中包含可编辑的文本。显示不带边框的文本框（BorderStyle=None）也有可能，这使它看起来像一个标签，并且不能明显地提示用户它是可编辑的。

6. 空白空间的使用

在用户界面中，使用空白空间有助于突出元素和改善可用性。空白空间不必非得是白色的 —— 它被认为是窗体控件之间以及控件四周的空白区域。一个窗体上有太多的控件会导致界面杂乱无章，使得寻找一个字段或者控件非常困难。在设计中需要插入空白空间来突出设计元素。各控件之间一致的间隔以及垂直与水平方向元素的对齐也可以使设计更可用。就像杂志中的文本那样，安排得行列整齐、行距一致，整齐的界面也会使其容易阅读。

Visual Basic 提供了几个工具，使得控件的间距、排列和尺寸的调整非常容易。“排列”、“按相同大小制作”、“水平间距”、“垂直间距”和“在窗体中央”等命令都可以在“格式”菜单中找到。

7. 保持界面的简明

界面设计最重要的原则就是简单化。对于应用程序而言，如果界面看上去很难，则可能程序本身也很难。稍稍深入考虑一下便有助于创建简明的界面。从美学的角度来讲，整洁、简单明了的设计常常更可取。在界面设计中，一个普遍易犯的错误就是力图用界面来模仿真实世界的对象。例如，想像一下要求创建完整的保险单的应用程序。很自然的反应就是在屏幕上设计完全仿照保险单的界面。这样做会出现几个问题：保险单的形状与尺寸和屏幕上的有很大不同，要非常完善地复制这样的表格会将其限制在文本框与复选框中，而对用户并没有真正的好处。最好是设计出自己的、也能提供原始保险单打印副本（带打印预览）的界面。通过从原始保险单中创建字段的逻辑组，并使用有标签的界面或几个链接的窗体，就可以不要求滚动屏幕而显示所有的信息。也可以使用附加的控件，比如带有选取预装入的列表框，这些控件可以减少打字工作量。也可以取出不常用的函数并把它们移到它们自己的窗体中去，来简化许多应用程序。提供缺省有时也可以简化应用程序；如果 10 个用户中有 9 个选取加粗的文本，就把文本粗体设为缺省值，而不要叫用户每次都选取一遍（不要忘记提供一个选项可以覆盖该缺省值）。向导也有助于简化复杂的或不常用的任务。简化与否最好的检验就是在应用中观察应用程序。如果有代表性的用户没有联机帮助就不能立即完成想要完成的任务，那么就需要重新考虑设计了。

8. 使用颜色与图像

在界面上使用颜色可以增加视觉上的感染力，但是滥用的现象也时有发生。许多显示器能够显示几百万种颜色，这很容易使人要全部使用它们。如果在开始设计时没有仔细地考虑，颜色也会像其他基本设计原则一样，出现许多问题。每个人对颜色的喜爱有很大的不同，用户的品味也会各不相同。颜色能够引发强烈的情感，如果正在设计针对全球读者的程序，那么某些颜色可能有文化上的重大意义。一般说来，最好保守传统，采用一些柔和的、更中性化的颜色。当然，预期的读者以及试图传达的语气与情绪也会影响对颜色的选取。明亮的红色、绿色和黄色适用于小孩子使用的应用程序，但是在银行应用程序中它很难带来财务责任心。少量明亮色彩可以有效地突出或者吸引人们对重要区域的注意。作为经验之谈，应当尽量限制应用程序所用颜色的种类，而且色调也应该保持一致。如果可能的话，最好坚持标准的 16 色的调色板；在 16 色显示器上观看时，抖动会使得其他一些颜色显示不出来。使用颜色时另一个需要考虑的问题就是色弱。有一些人不能分辨不同的基色（如红色与绿色）组合之间的差别。对于有这种情况的人，绿色背景上的红色文本就会看不见。

9. 图像和图标

图片与图标的使用也可以增加应用程序的视觉上的趣味，但是，细心的设计也是必不可少的。不用文本，图像就可以形象地传达信息，但常常不同的人对图像的理解也不一样。带有表示各种功能的图标的工具栏，它是一种很有用的界面设备，但如果不能很容易地识别图标所表示的功能，反而会事与愿违。在设计工具栏图标时，应查看一下其他的应用程序以了解已经创建了什么样的标准。例如，许多应用程序用一张角上有卷边的纸来表示“新建文件”图标。也许还有更好的比喻来表示这一功能，但改用其他的表示方法会引起用户的混淆。考虑图像文化上的意义也非常重要。许多程序使用田园风格的带一面旗的邮箱（如图

5.11）来代表邮件功能。这原本是美国的图标，其他国家/地区或文化的用户也许不把它看作邮箱。

在设计自己的图标与图像时，应尽量使它们简单。具有多种颜色的复杂的图片，作为 16×16 像素的工具栏图标，或者在高分辨率的屏幕上显示时，都不能很好地适应。

图 5.11　邮件图标

10. 选取字体

字体也是用户界面的重要部分，因为它们常常给用户传递重要的信息。需选取在不同的分辨率和不同类型的显示器上都能容易阅读的字体。最好尽量坚持使用简单的无衬线字体或者衬线字体。通常手写字体或者其他装饰性字体的打印效果比屏幕上的效果更好，而且字体越小读起来越难。除非计划按应用程序来配置字体，否则应当坚持使用标准 Windows 字体，如 Arial、New Times Roman 或者 System。如果用户的系统没有包含指定的字体，系统会使用替代的字体，其结果可能与设想的完全不一样。如果正在为[illegible]查在预想的语言里可用什么字体。还有，在为其他语言设计时，需要[illegible]有些语言的文本串可以多占 50%以上的空间。还有，在选取字体时[illegible]要。大多数情况下，不应当在应用程序中使用两种以上字体。太多的[illegible]上去像罚款通知单。

11. 可用性设[illegible]

任何应用程序的[illegible]决定。界面设计是一个需要多次反复的过程：在为应用程序设计界面时，第一步就设计出非常完美的界面的情况非常少见。用户参与设计过程越早，花的气力越少，创建的界面越好、越可用。

12. 好的界面风格

设计用户界面时，开始时最好是先看看 Microsoft 或其他公司的一些热卖的应用程序。毕竟，界面很差的应用程序不会卖得很好，你将会发现许多通用的东西，比如：工具栏、状态条、工具提示、上下文菜单以及标记对话框。Visual Basic 具有把所有这些东西添加到应用程序中的能力，这并不偶然。也可以凭借自己使用软件的经验。想一想曾经使用过的一些应用程序，哪些可以工作、哪些不可以以及如何修改它。但要记住，个人的喜好不等于用户的喜好，必须把自己的意见与用户的意见一致起来。还要注意到大多数成功的应用程序都提供选择来适应不同的用户的偏爱。例如，Microsoft Windows“资源管理器”允许用户通过菜单、键盘命令或者拖放来复制文件。提供选项会扩大应用程序的吸引力，至少应该使所有的功能都能被鼠标和键盘所访问。

13. Windows 界面准则

Windows 操作系统的主要优点就是为所有的应用程序提供了公用的界面。知道如何使用基于 Windows 的应用程序的用户，很容易学会使用其他应用程序。而与已创建的界面准则相差太远的应用程序不易让人很快明白。菜单就是这方面很好的例子 —— 大多数基于 Windows 的应用程序都遵循这样的标准：“File”（文件）菜单在最左边，然后是“Edit”（编辑）、“Tools”（工具）等可选的菜单，最右边是“Help”（帮助）菜单。如果说 Documents 会比 File 更好，或者“Help”菜单要放在最前，这就值得讨论一下了。没有任何事情阻止这样做，但这样做会引起用户的混淆，降低应用程序的可用性。每当在应用程序与其他程序之间切换时，用户都不得不停下来想一想。子菜单的位置也很重要。用户本期望在“Edit”菜单下找到“Copy”（复制）、“Cut”（剪切）与“Paste”（粘贴）等子菜单，若将它们移到

“File”菜单下会引起用户的混乱。不要偏离已经创建的准则太远，除非有很好的理由这样做。

14. 可用性的检测

测试界面可用性的最好方法是在整个设计过程中请用户参与。不论是正在设计大型的压缩包应用程序，还是小型的有限使用的应用程序，设计的过程应当完全相同。使用已创建的设计准则，界面设计应从纸上开始。下一步是创建一个或者多个原型，在 Visual Basic 中设计窗体。还需要增加足够的代码来启动原型：显示窗体、用示例数据填充列表框等。然后准备可用性测试。可用性测试可以是个不拘形式的过程：与用户一起审查设计；也可以是在已创建的可用性实验室中进行的正式的过程。这两种方法目的是一样的：从用户那里可以得到哪儿设计得很好，哪儿还需要改进的第一手材料。让用户与应用程序在一起，然后观察它们；这种方式比询问用户更为有效。当用户试图完成一系列任务时让他们表达其思考过程：“要想打开新文档，所以要在‘File’菜单中找一找。”记下哪些地方的界面设计没有反应他们的思考过程。与不同类型的用户一起测试，如果发现用户完成某个特定的任务有困难，该任务可能需要多加关照。下一步，复查一下记录，考虑如何修改该界面使它更加可用。修改界面并再测试。一旦对应用程序可用性满意，就准备开始编码。在开发的过程中也需要不时地测试来确保对原型的设想是正确的。

15. 功能的可发现性

可用性测试的关键概念是可发现性。如果用户不能发现如何使用某个功能（或者甚至不知道有此功能存在），则此功能很少有人去使用。例如，Windows 3.1 的大多数用户都从来不知道【Alt】和【Tab】的组合键可以用于在打开的应用程序之间切换。界面中没有任何地方可提供线索来帮助用户发现这一功能。

为了测试功能的可发现性，不解释如何做就要求用户完成一个任务。如果他们不能完成这个任务，或者尝试了好多次，则此功能的可发现性还需要改进。

16. 当用户或系统出错时与用户交互

在理想世界里，软件与硬件都会无故障地一直工作下去，用户也从不出错。而现实中错误总是难免的。决定当事情出毛病时应用程序如何响应，是用户界面设计的一部分。常用的响应是显示一个对话框，要求用户输入应用程序该如何处理这个问题。不太常用（但更好）的响应是简单地解决问题而不打扰用户。毕竟，用户主要关心的是完成任务，而不是技术细节。在设计用户界面时，考虑可能出现的错误，并判断哪一个需要用户交互作用，哪一个可以按事先安排的方案解决。

17. 创建容易理解的对话框

偶尔应用程序中会出现错误，需要为解决这种情况做出判断。这通常作为代码的分支出现 —— If...Then 语句或者 Case 语句。如果这个判断需要与用户交互，此问题通常用对话框来提交用户。对话框是用户界面的一部分，像界面的其他部分一样，它们的设计在应用程序可用性中发挥了作用。有时有这样的感觉，好像许多程序对话框的设计员，不会讲使人容易理解的话。比如这样的消息：“硬盘 C 的扇区被损坏或不能访问。中止、重试、忽略？”，这对一般的用户而言不大好理解。这等于有侍者问顾客：“我们没有汤或者厨房正在生火，中止、重试、忽略?”以用户能理解的方式或短语描述问题（和选择）是重要的。在前面的例子中，更好的消息可以是“在 C 盘上保存文件有问题，请把文件保存于 A 盘。文件保存与否?”

当为应用程序创建对话框时，心里想着用户。这个消息给用户传达了有用的信息吗？它容易理解吗？命令按钮表示的选择明确吗？这选择适合给定的条件吗？记住，仅仅一个讨厌的消息框就会使用户对应用程序产生坏印象。如果正在设计自定义对话框，尽量坚持用标准类型。如果与标准消息框布局相差太远，用户可能不会把它认作是对话框。

使用正版软件
请尊重相关知识产权!

18. 不用对话框的错误处理

当错误出现时不一定要打断用户。有时更可取的是不通知用户而用代码来处理错误，或者以不停止用户工作流程的方法来提醒用户。这个技术的很好的例子是 Microsoft Word 中的“自动更正”功能：如果普通单词拼错了，Word 自动修改它；如果不常用单词拼错了，在其下划一条红线提醒用户以后改正。有大量的技术可以使用；哪些技术适用于应用程序应由自己决定。这里有几个建议：

（1）在“Edit”（编辑）菜单中添加“Undo”（撤销）功能。对于删除等情况，与其用“OK”（确定）对话框来打断用户，还不如确保他们作出正确的决定并提供“Undo”功能以备他们以后改变主意。

（2）在状态栏或图标上显示消息。如果错误不影响用户当前的任务，不要停止应用程序。使用状态栏或亮色警告图标来警告用户，当他们准备好后可以处理该问题。

（3）改正问题。有时错误的解决办法很显然。例如，当用户试图存文件时磁盘已满，则在其他驱动器中检查系统寻找空间。如果空间可用，则保存该文件；在状态栏中显示一条消息告诉用户做了些什么。

（4）保存消息等候处理。因为不是所有的错误都是紧要的，或要求马上注意的；考虑把这些记录到文件中，当用户退出应用程序时或其他方便的时候再把它们显示给用户。如果用户发生输入错误（如：把 Main St.写成 Mian St.），记录它。添加“Review Entries”按钮和显示差异的函数，以便用户可以改正它们。

（5）不要做任何事。有时错误并不重要，不足以成为警告的原因。例如，LPT1 上的打印机的纸张没准备好这一事实，在准备打印之前并没有多大关系。等待，直到消息合乎当前的任务。

19. 设计用户辅助模式

不论用户界面设计得多么好，有时用户总需要帮助。应用程序的用户辅助模式包括诸如联机帮助和打印出来的文档等东西；它也可以包括用户辅助设备，如工具提示、状态条、“这是什么”帮助以及向导。像应用程序的其他任何部分一样，用户辅助模式设计应当在开始开发之前。模式的内容将随着应用程序的复杂程度与预期读者的不同而不同。

20. 帮助与文档

联机帮助是任何应用程序的重要部分，它通常是用户有问题时最先查看的地方。甚至简单的应用程序也应该提供“帮助”，不提供它就好像是假定用户从来不会有问题。在设计“帮助”系统时，记住它的主要目的是回答问题。创建主题名称与索引条目时尽量用用户的术语，例如，“我如何格式化页面？”。不要忘记上下文相关性；对大多数用户而言，如果他们按下【F1】键寻求一指定字段的帮助，却发现自己在内容主题上，则他们会感到受挫折。基本概念的文档，不管是打印的和/或由压缩盘提供的，对所有的应用程序都是有帮助的，除了最简单的以外。它可以提供那些用简短的“帮助”主题难以传达的信息。至少，应该在 ReadMe 文件窗体中提供用户在需要时可以打印的文档。

21. 用户辅助设备

在用户界面中，有几种对用户提供辅助的技术。用 Visual Basic 在应用程序中添加工具提示、“这是什么”帮助、状态显示和向导是很容易的。这些设备中的哪些适用于自己的应用程序应由自己决定。

22. 工具提示

当用户在用户界面上搜索时，工具提示（如图 5.12 所示）是一种向他们显示信息的好方法。工具提示是个小标签，当鼠标指针在控件上停留片刻即显示，通常包含此控件的功能描述。正常情况下，工具提示与工具栏结合使用，它在界面的大多数部分也能很好工作。

图 5.12　工具提示

23. 状态显示

状态显示也可用与工具提示差不多的方法来提供用户辅助设备。状态显示是提供那些不太适合工具提示的指令或消息的一种好方法。包括在 Visual Basic 的状态条控件能很好地显示消息。

第 6 章　Visual Basic.NET 的 OOP 结构

使用本复制品
请尊重相关知识产权!

本章包括:

OOP 的相关概念
VB.NET 的面向对象性
建立和使用对象
在程序运行中得到一个类的信息
在 Visual Basic.NET 中创建一个类
Visual Basic NET 的接口
类的继承
多态性

6.1　OOP 的相关概念

在这一小节里，主要是使用相关的篇幅来阐述面向对象的概念，如果您对面向对象的概念已经非常的清楚了，建议您跳过这一节，如果您对这个概念不清楚或者是不知道，那么这一章的概念将对以后的学习大有帮助。

OOP（Object-Oriented-Programming）是相对于结构化程序设计（Structure Programming）而言的，表示采用面向对象的思想进行软件的编制。它是当今最流行的编程模式。“面向对象”技术追求的是软件系统对现实世界的直接模拟，尽量实现将现实世界中的事物直接映射到软件系统的解空间。面向对象编程和以前的编程思想有所不同，因为它把一个新的概念——对象，作为程序代码的整个结构的基础和组成元素。而类就是对象的抽象和概括，类是数据、属性和方法的封装，从某种角度来讲，类就像一个没有界面的控件。

对象的定义：对象是现实世界中的一个实体。它有如下特征：有一个名字以区别于其他的对象；有一个状态用来描述它的默写特征；有一组操作，每一个操作决定对象的一种功能或行为，对象的操作可分为两类：一类是自身承受的操作，一类是加于其他对象的操作，是其自身所具有的状态特征及可以对这些状态施加的操作结合在一起所构成的独立实体。

例如下面这个例子：

对象的状态：

对象名：张三
性别：女
身高：1.65 米
体重：55 公斤

对象的功能：

回答体重
回答身高
回答性别
教概率课
当家教

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

在计算机世界中可以把对象看成是存储中的一个可标识的区域，它能保存固定或可变数目的数值。对象的划分并没有惟一的标准，它依赖于设置对象的目的和所需要的操作。一个对象的状态并不是完全用来直接为外界服务的，但其本身是能够为外界服务的基础。所以对象的特征表现为：模块的独立性，也就是模块内部状态不因外界的干扰而改变，模块间依赖性小，各模块可独立为系统所组合选用和复用；动态连结性，即通过消息激活机制，把对象之间动态联系在一起，使整个机制运转起来，便称为对象的连结性；易维护性，就是对象的功能被“隐蔽”，修改完善功能被局限于该对象的内部，不会波及到外部。

在面向对象系统中，对象之间的联系是通过消息来传递的。消息，是对象之间相互请求或相互协作的途径，是要求某个对象执行某个功能操作的规格的说明。它有如下的性质：

（1）同一个对象可接收不同形式的多个消息，产生不同的响应；

（2）相同形式的消息可以送给不同对象，所做出的响应可以是截然不同的；

（3）消息的发送可以不考虑具体的接收者，对象可以响应消息，也可以对消息不予理会，对消息的响应并不是必须的。

消息分为公有消息和私有消息，由外界对象直接向其发送的是公有消息；而由自己向本身发送的，不对外开放，外界不必了解的是私有消息。

以上我们主要扼要地讨论了一下对象及它的一些特性，现在我们就可以提出类的概念了。类，是对一组客观对象的抽象，它将该组对象所具有的共同特征（包括结构特征和行为特征）集中起来，以说明该组对象的能力和性质。在计算机世界的系统构成上，类形成了一个具有特定功能的模块和一种代码共享的手段。

面向对象系统都具有三个特性：封装性、继承性、多态性。

封装：将一个数据和与这个数据有关的操作集合放在一起，形成一个能动的实体 —— 对象，用户不必知道对象行为的实现细节，只需根据对象提供的外部特性接口访问对象即可。目的在于将对象的用户与设计者分开，用户不必知道对象行为的细节，只需用设计者提供的协议命令对象去做就可以。

继承：继承所表达的就是一种对象类之间的相交关系。它使得某类对象可以继承另外一类对象的特征和能力。

若类间具有继承关系，则它们之间应具有下列几个特性：

（1）类间具有共享特征（包括数据和程序代码的共享）；

（2）类间具有细微的差别或新增部分（包括非共享的程序代码和数据）；

（3）类间具有层次结构。

多态：多态性描述的是同一个消息可以根据发送消息对象的不同采用多种不同的行为方式。

所以说，面向对象的程序可以看成就是这样一些具有数据、方法的对象之间的作用。用户在设计程序的时候就需要特别注意对类的选取和设计。

6.2　VB.NET 的面向对象性

您是浏览器提醒您：
使用本复制品
请尊重相关知识产权！

在 VB 的早期版本中（1.0-3.0）并没有包括面向对象的功能，从 VB 4.0 开始，用户可以像建立一个新的窗体那样建立一个新的类，并把它作为一个新的对象，这一节将阐述在程序里面使用类模块的一些优点，以后的一些章节将扩张这些概念，一直到一种完全面向对象的编程语言 VB.NET。

Visual Basic 中处理的任何事几乎都和对象有关系，如果读者对面向对象编程没有认识，则需要建立以下一些概念。

在 Visual Basic 中处理的任何事都和对象有关系，如果读者对面向对象编程没有认识，需要建立以下一些概念。

对象是一些把属性（Properties）、字段（fileds）、方法（methods）和事件（event）作为一个单独的数据类型进行处理的实体。对象可以使用户只需首先声明一些变量和方法以后，用户就可以在感觉需要使用的时候，调用一个它的对象进行重用就可以了。

例如想保存一辆汽车的信息，就可以通过定义一些变量来描述一辆车的颜色和马力，但问题是，这个变量只是描述这一辆汽车的参数，如果又要描述另一辆汽车的参数的话，则又要定义一些其他的变量来存储这些信息，但是如果使用类，就可以解决这个问题，只需要定义一个通用的汽车的类，每当需要描述一辆汽车的时候只需定义一个对象就可以了，这样就达到了代码重用的目的。

类描述了对象的字段、属性、方法以及对象的事件，对象是类的一个实例，在定义了一个类以后，可以在需要的时候定义很多对象。

1. 字段，属性，方法和事件

类是由属性，字段，方法和事件组成的，字段就是一个对象含有的片断的信息。用户可以像使用变量一样使用属性来存储一些信息。例如，一个对象名叫“Car”中有一个字段名为“Color”，下面的代码就是显示怎样改变“color”字段的值的：

Car.Color=“Blue”

属性的检索和设置与字段相似，但是它们在类中是以“Property Get”和“Property Set”过程来定义的。

方法表示了类可以做的一些事情，例如 Car 对象可以有 StartEngine、Drive 和 Stop 方法。

事件是表示一个对象从外部得到的信息的一个标示，事件允许对象任何时候在事件调用的时候做出相应的动作。例如，类 Car 的一个事件可以是 Check_Engine。因为微软的 Windows 是一个事件驱动的系统，所以事件可能会来自于任何一个地方。比如，当用户点击键盘或鼠标时，就是触发了一个事件。当然，有的时候，一些事件的触发可能来自于其他的对象。

2. 封装，继承和多态

属性、方法和事件只是面向对象编程的部分概念，真正的面相对象编程需要对象满足和支持下面三个特性：封装性、继承性和多态性。

封装性表示一个对象的属性和方法。对外界而言，是不能直接访问的。而对象本身对于属性和方法具有绝对的控制，比如一个对象可以在属性允许变化之前确认一下变量的值。

继承性表示可以在一个类的基础上建立一个新类。派生类，也就是继承类，可以具有基本类的所有属性、方法和事件，并且可以加上一些其他的属性和方法，例如可以根据基本类 Car 建立一个新类 SUV。SUV 类可以从 Car 类中继承 Color 属性，而且还可以加上一些其他的属性例如 FourWheelDrive。

多态性是指一些不同的类可以建立一个具有相同名称的属性或者方法。多态性对于面向对象编程是非常有必要的，因为它可以使用户不论在什么时候，也不论用户正在处理什么对象，都可以调用这个它们具有共同名称的方法。例如有一个基本类 Car，多态性可以允许它的派生类具有不同定义的 StartEngine 方法，派生类 DieselCar 的 StartEngine 方法可能和基本类的 StartEngine 方法完全不一样，但是也可以调用的。当然，其他的方法或者属性也可以这样处理。

类是面向对象的基础。类和 Type 数据结构的区别在于，Type 数据结构只有数据成员，而类的成员包括数据成员、属性、和方法。类就是一个没有图形界面的 ActiveX 控件，用户可以使用它，但不能看到它。在 VB.NET 以前的版本中，类是不能继承的，但是值得庆幸的是在 NET 版本中，就已经实现了类的继承。用户可以通过以下的描述来比较类和 Type 数据结构的区别。

Structures 是一个把一些相关的数据归类的强有力的工具，如，一个用户定义的一个名为 udtAccount 的数据类型：

```
Public Type udtAccount
    Number As Long
Type As Byte
CustomerName As String
Balance As Double
End Type
```

用户可以申明一个 udtAccount 类型的一个实例，然后设置里面字段的值，把整个记录打印出来，把它存到数据库中等操作。

虽然 Structures 类型设置的功能很强大，但是在代码中，Structures 可能会带来一些问题。用户可能建立一个 Withdrawal 过程，但在 Withdrawal 超过了它的平衡点的时候会产生错误，而且也没有办法阻止程序的其他地方改变 Balance 字段。

换一句话说，操作过程和 Structures 的联系靠的只是一些规定、人为的记忆，以及程序员对代码维护知识的了解。

而面向对象编程就可以解决这个问题，它把数据和操作过程结合在一起，并且组成一个独立的实体。用户定义 Account 为一个类来代替 udtAccount Structures 时，这些数据都变成了私有的，而且那些访问这些数据的操作过程也被定义为这个类一部分，成为了这个类的属性和方法，这就是所谓的封装性，也就是说一个对象就是一个包含数据和操作的单元。

当用户通过一个类建立一个 Account 对象时，就可以避免这种错误，用户只能通过这个对象的属性才能访问它的一些数据字段，下面的代码片断显示了 Account 类中的过程怎样支持封装性：

```
' The account balance is hidden from outside code.
Private mdblBalance As Double
' The read-only Balance property allows outside code
```

```
' to find out the account balance.
Public Property Get Balance() As Double
    Balance = mdblBalance
End Property
' The Withdrawal method changes the account balance,
' but only if an overdraft error doesn't occur.
Public Sub Withdrawal(ByVal Amount As Double)
    If Amount > Balance Then
        Err.Raise Number:=VBObjectError + 2081, _
        Description:="Overdraft"
    End If
    mdblBalance = mdblBalance - Amount
End Sub
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

用户并不需要担心如何使用类中的操作过程，也不必担心属性、过程、私有变量的一些语法，最重要的就是要记住，用户可以定义一个类来封装自己的数据和操作方法。

用户也不必要担心是否使用了正确的对 Account 的操作，因为这些操作现在也被包含在对象之中了。

类在 VB.NET 中是一个非常重要的部分，几乎所有正规的程序都包括了一个或者几个类，在 VB.NET 中，类模块和窗体的区别已经不存在了，几乎所有的程序都是由类组成，为了更好地使用这些类，它们被按照不同的功能分在不同的功能类库里。

除去这些表面现象，这些类库都是在独立的 DLL 动态库中，用户只需要用 Imports 说明就可以引用它们了，这样这些类里面的函数就可以用了。如：

```
Import System.Collections 'Use Collection namespace classes
```

逻辑上，每一个类库代表了一个独立的命名空间（namespace），当用户引用了一个命名空间后，编译器将可以找到这些命名空间所对应的类和方法。最常用的命名空间是“System”命名空间，它是一个缺省的引用，并不需要用户去申明它，其中包括了一些用于访问最基本的类和方法，如：Application、Arry、Console、Exception、Object 以及一些基本的标准的对象，如：Byte、Boolean、Single、Double、String。在下面的最简单的控制台程序中，我们可以看一看关于 System 的引用。

```
Imports System
'Simple VB Hello World program
public Class cMain
    shared sub Main() '表明启动函数是 Main
        '在控制台中写文本
        Console.WriteLine("Hello VB World")
    End sub
End Class
```

上面这个程序只是在 DOS 窗体写了一句文本“Hello VB World”，程序必须以 Sub Main 子例程开始启动，而且在类模块中，它必须声明为“Shared”，而在 VB.NET 模块中这个程序则应该写成：

```
'Simple VB Hello World program
public Module cMain
    Sub Main()
        Console.WriteLine ("Hello VB World")
    End Sub
End Module
```

超星阅览器提醒您：
此为试读，需要完整PDF请访问：

在上面这两个例子，除了在类模块中，Sub Main 被声明为“Share”。读者可以看出它们是很相像的，在 VB.NET 中模块和早期的版本的 VB 中模块的作用相似，在模块中声明的方法和常量在整个程序中都是公有的。

6.2.1　VB.NET 的共享成员（Share Members）

共享成员就是被类的实例所共享的属性、过程、字段等等。在 Visual Basic.NET 的继承中，在所有的类的实例中把一个数据成员或者函数的一个单独的实例设置成共享是非常有用的。共享成员独立于任何一个特定的类的实例。共享的过程在对象中必须显式地声明。由于这个原因，在共享方法中，引用不共享的成员是允许的。公有的共享的成员可以远程地访问，并且可以在对象中后期绑定。

注意：在其他的编程语言中，共享成员也被称为 Static 或者 class 成员。

有一个类库，该类库含有一些附加的过程，这些过程对于这个类库而言是一些概念性的部分，需要单独运行，并不需要用户去声明这个类中的一些类型。

假设开发了一个应用程序，它需要知道有多少个用户现在登录了该应用程序。当用户安装了用户的应用程序以后，注册信息被写入了注册表，而且用户的运行程序每次在运行的时候都要访问用户声明的 UserInfo 对象。一个 Long 型共享成员 UserCount 就可以跟踪用户的个数，并且用一个属性 CanCreateUser 来检查 UserCount 共享成员，是否要声明已经达到了登录人数的极限最大值，并且初始化 UseInfo 的值来决定是让用户进入。

用户也可以在以下情况使用共享成员，假设有很多对象，而且这些对象需要共享一些信息。例如，假设已经建立了一个制定行程计划的软件用来计算所选择的一个旅游城市离国会大厦是不是在 100 里之内。在 City 类中有 Latitude 和 Longitude 成员。对于半径在 100 里之内的城市。City 的共享的 NearCapitol 属性为“Incremented”，而对于在半径在 100 里之外的城市，City 的共享的 NearCapitol 属性为“Decremented”。

在以上这些情况时，如果没有这些共享成员的话，需要建立一个 a.bas 模块来记录这些信息，这样做可能会给代码的维护带来一些困难，而且理解类也会更加困难。在一个类中保留共享成员会是用户更好地理解它们的目的，而且在工程中，也将会更好地重用这些代码。

下面这个例子声称了一个具有一个属性和共享数据成员的类。当用户运行这个程序的时候，每一个 PortDescriptor 类的对象均有一个惟一的“Descript”属性，但是共享着一个共享的数据成员 Load，在对象 PD1 中改变属性“Load”属性的话将会改变对象 PD2 的属性“Load”的值，其代码如下：

```
Imports system
Namespace ANameSpace
  Module Module1
```

```
Class PortDescriptor
  Public Shared Load as Integer
  Private strDesc as String
  Property descript as String
    Get
      descript = strDesc
    End Get
    Set
      strDesc = Value
    End Set
  End Property
End Class
Shared Sub Main()
  Dim PD1 as PortDescriptor = New PortDescriptor
  Dim PD2 as PortDescriptor = New PortDescriptor
  PD1.descript = "Idle"
  PD1.Load = 80
  Console.WriteLine("PD1 Descript = : " & PD1.descript)
  Console.WriteLine("PD1 Load = : " & PD1.Load)
  PD2.descript = "Active"
  PD2.Load = 540
  Console.WriteLine("PD2 Descript = : " & PD2.descript)
  Console.WriteLine("PD2 Load = : " & PD2.Load)
  Console.WriteLine("PD1 Descript now = : " & PD1.descript)
  Console.WriteLine("PD1 Load now = : " & PD1.Load)
 End Sub
End Module
End Namespace
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

6.2.2　类模块和标准代码模块的区别和比较

在实际的编程中，什么时候使用类，选择标准的代码模块呢？类和标准代码模块的不同主要是概念上的。用户要完成的工作主要是针对于某个或几个特定的对象，那么就可以使用类。对象的动作就是类的方法，对象的属性必须用类的属性过程来实现。反之，如果用户有一个过程是针对通常的一组事务，而不是某些特定的对象，那么这个例程最好在标准的代码模块中实现。而且类模块和标准模块的不同点还在于存储数据方式的不同。标准模块的数据只有一个备份，这意味着标准模块中的一个公共变量的值改变以后，在后面的程序中再读取该变量时，它将得到同一个值，而类模块的数据，是相对于类实例（也就是由类创建的每一个对象）而独立存在的。同样，标准模块中的数据在程序作用域内存在，也就是说，它存在于程序的活期中；而类实例中的数据值存在于对象的存活期，它随对象的创建而创建，随

对象的撤销而消失。最后当变量在标准模块中声明为 Public 时，则它在工程中任何地方都是可见的；而在类模块中的 Public 变量，只有当对象变量含有对某一类实例的引用时才能访问。

一般来说，在如下两种情况下使用类：

（1）用户想用类的方法和属性来提高代码的封装性；

（2）用户希望创建大量性质相近的对象。

在其他的情况下，一般使用标准的代码模块。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

6.2.3 对象浏览器

对象浏览器可以使用户通过一些组件（components）检验和发现一些对象，如：命名空间（namespaces）、类（class）、结构（structures）、接口（interfaces）、类型（types）、枚举（enums）等等，以及它们的成员属性（properties）、方法（methods）、事件（events）、变量（variables）、常量（constants）、枚举项（enum items），这些组件可以是用户的解决方案（solution）里的工程（project），可以是这些工程里引用的组件，也可以是一些外部组件。这些组件都可以通过改变对象浏览器里的浏览窗口进行不同的选择。这样对使用的类通关全局的了解将有很大的意义。

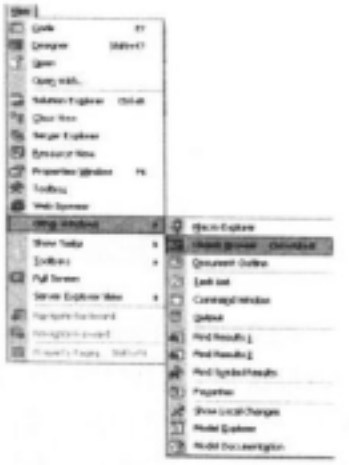


图 6.1 打开对象浏览器

可以通过直接按【F2】键或者在“view”菜单下的“other Windows”下的“objects browser”就可以打开对象浏览器了，如图 6.1 所示，对象浏览器的界面如图 6.2 所示。

如图 6.2 所示，用户可以在通过组件浏览器选择一个组件，则它所包括的对象出现在左端，关于选定的对象（对象窗口）及对象的成员（成员窗口）出现在右端，关于这些对象、成员的描述信息在对象浏览器底部的描述窗口出现。

对象栏：显示当前浏览范围内所有的对象，包括命名空间（namespaces）、类（classes）、结构（structures）、接口（interfaces）、类型（types）、以及枚举（enums），它是以树的形式出现的，用户可以通过双击树的节点或点击“+”展开节点下包括的内容。

对象成员栏：每个对象都包括诸如：属性（properties）、方法（methods）、事件（events）、变量（variables）、常量（constants）、枚举项（enum items）等等内容。在对象浏览器里，这些内容统统被称作为成员，在对象栏里选择对象，它可以认为是一个对象的集合，则它的成员就会显示在成员栏中，点击成员，则在描述栏及时更新，并显示成员的详细描述。

描述栏：描述栏在对象浏览器的底部，用以显示选中的对象或成员的详细信息。

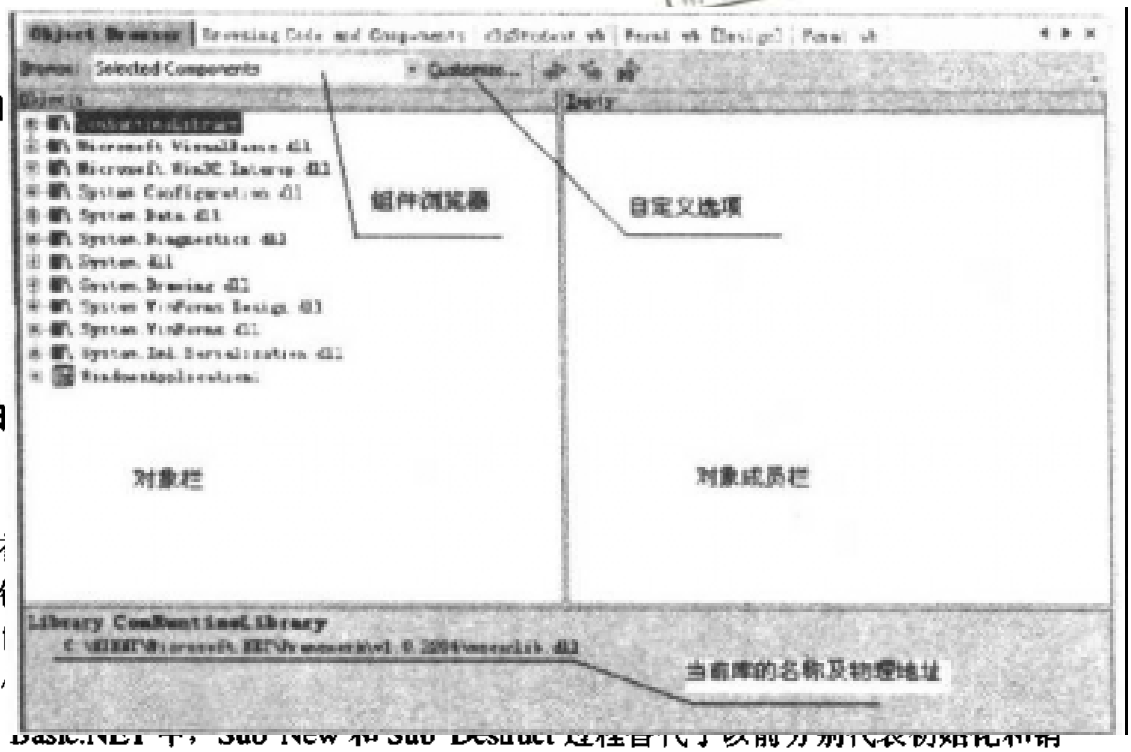
自定义按钮：用以定义要浏览的组件，可以是工程所引用的组件，也可以是一些外部组件。

当用户要搜索特别的内容的时候，点击“Find Symbol”按钮，确定搜索范围，搜索类型和方式，就可以在搜索结果窗口得到结论，如图 6.3 所示。

在类视图（Class View）中，列出了当前工程中的一些类的继承及结构关系，通过使用类视图，用户可以发现编制的代码的一些不足之处，并对程序进行编辑，调整对象之间的结

构关系。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！



6.3.1 对象的

构造函数

一般情况下，

册表中的某个

值为 Nothing

个类的对象的

在 Visual Basic.NET 中，Sub New 和 Sub Destruct 过程首代子以前分别代表初始化和销毁一个对象的 Class_Initialize 和 Class_Terminate 方法。不像 Class_Initialize 方法，Sub New 方法只是在类建立时运行一次，而且它不能在代码行的其他地方显式调用，除了在本身的类模块或者在自己的派生类的代码的构造过程中。同时，在 Sub New 方法中的代码比其他的在类模块中的代码都要更早地运行。

当一个对象被销毁的时候，Sub Destruct 方法被系统自动调用，但是不能被显式调用。.NET 框架能够在系统自认为一个对象并没有使用价值时自动把该对象销毁。但是切记，不像 Class_Terminate 和 Sub New 那样，用户不能确定.NET 框架什么时候会调用 Sub Destruct 方法。用户只能知道，系统将会在上一次引用一个对象之后调用 Sub Destruct 这个方法。

1. 带参数的 Sub New 的构造函数

用户可以通过在类的定义的任何地方定义 Sub New 过程来实现对类的构造函数。构造函数的第一行必须是调用它的基本类的构造函数或者是当前类的其他构造函数。用户必须确认基本类是在其他的类的初始化过程之前初始化的。但是令人惊讶的是，即使创建基本类，用户也必须调用构造函数 Mybase.New，因为所有的类都是最终从一个名为 Object 的类继承而来的。在调用它的父类的构造函数以后，用户可以在 Sub New 过程中加入一些另外的初始化的代码。Sub New 可以支持带参数的构造函数，这些参数通过调用构造函数时，从过程

中传入的，例如：

```
Dim t as truck = New Truck(18)
```

2. 使用 Sub Destruct 作为析构函数

为了建立一个类的析构函数，用户可以建立一个名为 Sub Destruct 的过程，并且可以放在类的定义中的任何地方。在析构函数中写入的代码可以用来释放其他的对象，关闭一个文件或者做其他的结束工作。下面的例子说明了怎样通过一个带参数的构造函数建立一个类，并且初始化一个属性。

```
Imports System
Namespace TruckNamespace
Module Module1
Class Truck
  Private iWheels as Integer
    Sub New(initialWheels as integer)
        MyBase.New
        iWheels = initialWheels
    End Sub
    Sub Destruct()
        ' Place cleanup code here.
    End Sub
    Property Wheels as Integer
        Get
            Wheels = iWheels
        End Get
    Set
      iWheels = Value
    End Set
        End Property
End Class
Shared Sub Main()
        Dim t as truck = New Truck(18)
        Console.WriteLine("The truck has " & _
          t.wheels & " wheels when initialized")
End Sub
End Module
```

6.3.2 设置和重设属性

当用户可以设置或者得到属性的值的时候，这些属性就在改变。有些属性可以在设计的时候改变。用户可以在属性窗口设置这些属性的值，而不需要写任何代码，而有些代码在设计的时候是不能改变的，所以必须在运行的程序中写入一些设置属性的值的代码。

那些可以在运行的时候设置和读取的属性被称作“read-write”属性。而那些只能在运行的时候读取的代码称为“read-only”属性。

1. 设置属性值

当需要改变一个对象的外观或者行为时，就可以改变对象的属性值。例如可以通过改变一个 text box 控件的“Text”属性值，来改变 Text Box 里面的内容。

设置一个属性的值，可以用以下的语法：

```
object.property = expression
```

如下一些实例：

```
TextBox1.Top = 200          ' 让 Top 属性值为 200 缇.
TextBox1.Visible = True     ' 显示 text box.
TextBox1.Text = "hello"     '在 text box 中显示'hello'
```

2. 得到属性值

当代码要对一个对象进行一些操作时，用户可能需要得到一个对象的状态，例如，在对一个 Text Box 控件进行下一步操作时，可能需要得知 Text box 的“text”属性的值，等等。

通常情况下，用户可以通过下面这样的语法得到一个对象的属性值：

```
variable = object.property
```

用户可以把一个对象的属性值作为一个复杂表达式的一部分，而不需要先把属性值赋给一个变量再进行计算，下面这个例子就是改变一个 radio 按钮的属性的代码：

```
Private Sub cmdAdd_Click()
    '[statements]
RadioButton1.Top = RadioButton1.Top + 20
    '[statements]
End Sub
```

注意：如果用户需要多次使用一个属性值的话，那么先把属性值赋给一个变量，将会使效率更快一点。

6.3.3 用方法来表现动作

方法（methods）是和对象有关的过程（procedures），方法表示一个对象可以进行的动作，和属性相比较，属性是表示一个对象的信息的，而方法是可以影响属性的值。例如，在模拟收音机中，用户可以用 SetVolume 方法来改变属性 Volume 的值，简而言之，在 Visual Basic 中，list box 控件有一个“list”属性，而且可以用 Clear 和 Add 方法进行改变。

1. 使用一个不需要参数的方法

语法如下：

```
object.method()
```

在下面的例子中，refresh 方法可以重新刷新 picture box：

```
PictureBox1.Refresh()    ' Forces a repaint of the control.
```

像 refresh 这种方法是不需要参数的，也不返回值。

2. 使用需要参数的方法

首先，用逗号把参数隔开，Visual Basic.NET 需要所有的参数都用括号括起来。

在下面的例子中，Messagebox.Show 方法就需要一个参数用于标示对话框的显示方式以及内容，如下：

```
MessageBox.Show("Database update complete", "My Application", _
                MessageBox.OK Bitor Messagebox.IconExclamation)
```

3. 使用一个返回值的方法

用户可以使用返回值的方法，它基本上和使用一个函数是一样的。例如，如果想从Messagebox.Show 方法中得到返回值，并把返回值存储起来，则只需要在方法左边的等号的左边加上一个变量就行了：

```
intresponse = MessageBox.Show("Do you want to exit?", "My Application", _
                MessageBox.YesNo Bitor  MessageBox.IconQuestion)
```

6.3.4 对象变量的声明

用户可以使用一般的声明的描述来声明一个对象变量，但是必须把变量声明为 Object 或者一个具体的类的名称，用下列语法进行对象的声明：

```
Dim variable As [New] {Object | class}
```

用户能够使用 Protected、Friend、Private、Shared，或者 Static 进行对象的声明。下面的对象的描述都是合法的：

```
Private ObjA As Object        ' Declare ObjA as generic Object data type.
Static ObjB As Label          ' Declare ObjB as Label class type.
Dim ObjC As System.Buffer     ' Declare ObjC as Buffer class type.
```

注意：如果没有声明一个对象变量，则该对象的数据类型是缺省的 Object。然而这种不声明变量的方法不推荐使用。

有的时候，对象的类型在过程没有运行之前还是不确定的，在这种情况下，可以声明这个对象变量的类型为 Object 数据类型。这可以创建一个对任何对象的引用。

然而，如果知道对象具体属于哪一个类的话，最好把它声明为该类的对象，正如刚才的那些例子，如果已经知道了对象是类 Label 的一个实例，那就应该把该变量声明为 As Label。

把一个对象声明为一个特定的类的一个实例，有如下好处：

（1）动态检查类型。

（2）在代码中得到微软的 intellisense 支持。

（3）增加可读性。

（4）减少代码的错误率。

（5）代码运行效率提高。

当在声明一个对象的类型的时候，声明的类型决定了对象变量的灵活性和适用范围。例如，如果用户在应用程序之中定义了一个窗体命名为 Form2，那用户就可以把一个对象变量声明为一个 Form2 的对象：

```
Dim MyForm As New Form2       ' Can refer only to an object of class Form2
```

用户也可以把这个对象变量声明为一个普通的窗体：

```
Dim AnyForm As Form           ' Can refer to any Form, but only a Form
```

也可以把这个对象变量声明为一个普通的控件：

```
Dim AnyControl As Control   ' Can refer to any type
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

6.3.5　对一个对象进行多种操作

用户可能经常对一个对象进行一系列的多种操作。例如，可能需要对一个对象设置几种属性，可采用如下多种方法：

1. 使用多种表述

如下代码：

```
Private Sub InitializeComponent()
    Button1.Text = "OK"
    Button1.Visible = True
    Button1.Top = 200
    Button1.Left = 5000
    Button1.Enabled = True
End Sub
```

也可以更容易地使用 With...End With 提高代码的效率，代码如下：

```
Private Sub InitializeComponent ()
    With Button1
        .Text = "OK"
        .Visible = True
        .Top = 200
        .Left = 5000
        .Enabled = True
    End With
End Sub
```

可以在 With...End With 模块中使用嵌套，如下代码所示：

```
Imports color = system.Drawing.Color
Sub SetupForm()
    Dim frmAnotherForm As New Form1()
    With frmAnotherForm
        .show() ' show the new form
        .Top = 250
        .left = 250
        .ForeColor = Color.LightBlue
        .BackColor = color.DarkBlue
        With frmAnotherForm.textbox1
            .BackColor = Color.Thistle ' change the background
            .Text = "Some Text" ' Place some text in the textbox
        End With
```

```
        End With
    End Sub
```

2. 把窗体看作对象

窗体是描述用户的应用程序和用户接口的类，当窗体显示以后，一个窗体类的对象就被建立并且像其他任何的对象一样，用户可以给窗体加入一个自定义的属性和方法，并且从其他的类访问这些属性和方法，要给一个窗体加入一个方法，只需要在代码中加入一个声明为 Public 的过程就可以了，如下代码：

超星阅览器提醒您
使用本复制品
请尊重相关知识产权!

```
' Custom method on Form1
Public Sub PrintMyJob()
'
        <statements>
End Sub
```

要给一个窗体加入一个数据成员，只需把一个公有的变量在窗体模块中声明就可以了：

```
Public IDNumber As Integer
```

当引用一个窗体的名称，其实确切地讲，引用的是这个窗体属于的类，而不是窗体这个对象本身，一个经常容易犯的错误就是企图不使用对象变量而直接引用一个类的属性，例如：

```
Form1.PrintMyJob ' Error, this will not work unless you created form1
```

为了访问一个不同的窗体上的方法需要先建立一个那种窗体类的实例，并把它赋给一个对象变量，如下面的例子：

```
Dim frmForm1 as New Form1
frmForm1.PrintMyJob
```

注意到上面这个例子建立了一个新的窗体对象，而且没有显示它。如果用户需要使用一个窗体对象的方法的话，没有必要一定把这个窗体显示出来。如果需要显示一个新的窗体，只需用如下代码即可：

```
frmForm1.show
```

对于一个窗体类的实例而言，它的属性值是确定的，用户可以很容易就能访问当前的窗体的属性值，但如果想访问任意一个窗体的话，情况就会更加复杂一点。例如：假如有这样一个应用程序，它有一个主窗体名为 form1，而且有两个子窗体分别名为 form2 和 form3，它们之间需要相互得到一些对方的信息，如果主窗体建立和显示了 form2 和 form3 以后，其他的窗体将不能访问 form1 的一些信息。一个解决办法就是把需要在主窗体中声明 Public Shared 对象变量来对其他的窗体进行引用。Public Shared 对象变量是在所有实例中共享相同值的类成员。

下面的例子说明了怎样在 form1 对象中设置和使用 Public Shared 对象变量，如果有很多窗体的话，可以建立一个 Public Shared 对象变量窗体的集合。

用户可以通过以下方法来建立一个公有的共享的对象变量：

（1）把应用程序需要使用的窗体都设计好，确认在其他的窗体需要访问的控件都设置为了 Public，例如：

```
Public txtMyTextBox As System.WinForms.TextBox
```

（2）在主窗体中声明 Public Shared 变量，并让它能够存储对其他窗体的引用，例如：

```
Public Shared frmForm2 As form2
```

```
Public Shared frmForm3 As form3
```

（3）建立其他的窗体类的实例，并且把对象的引用设置为 Public Shared 变量：

```
form1.frmForm2 = New Form2 ' Create form2 and save reference
form1.frmForm3 = New Form3 ' Create form3 and save reference
```

（4）在合适的时候显示窗体，例如：

```
form1.frmform2.Show()
 form1.frmform3.Show()
```

超星阅览器提醒您：
请尊重知识产权！

（5）现在就可以像访问主窗体的属性一样，访问其他窗体的数据，例如：

```
Me.txtSomeTextBox.Text = form1.frmForm3.txtMyTextBox.text
form1.frmForm3.BackColor = system.Drawing.Color.DarkGray
```

6.3.6 使用 New 关键字

使用 New 关键字建立一个新的对象和用它属的类来定义是一样的。New 关键字可以被用作来建立一个窗体、类模块，以及集合的实例。用户在设计的时候建立的每一个窗体都是一个类。New 关键字能够用来建立一个类的实例。

可以通过以下步骤来建立一个类的实例：

（1）在窗体上画一个按钮和几个其他的控件。

（2）按钮的 Click 事件过程中加入以下代码：

```
Dim x As New Form1
x.Show
```

（3）运行程序，并且点击几次按钮，把最前面的窗体移开，因为窗体是一个有可视界面的类，此时可以看见有很多窗体的拷贝，每一个窗体具有相同的控件，并且出现在一个地方。

（4）将列代码加到按钮的 Click 事件过程中：

```
Dim f As Form1
f = New Form1
f.text = "hello"
f.Show
```

使用 New 关键字可以从类模块中定义的类中建立一个新的集合和对象，要知道它们怎样工作，可以参考下面的例子：

（1）建立一个新的工程，并且在一个名为 Form1 的窗体上画一个按钮控件。

（2）在 Project 菜单下，选择“Add Class”给工程添加一个类。

（3）把新的类命名为 ShowMe.VB

（4）在该新类中加入如下代码：

```
Public Class ShowMe
    Sub ShowFrm()
    Dim frmNew As Form1
    frmNew = New Form1
    frmNew.Show()
    frmNew.WindowState = 1
```

```
    End Sub
End Class
```

（5）在 Button1_Click 事件的过程中加入如下代码：

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs)
    Dim clsNew As New ShowMe()
    clsNew.ShowFrm()
End Sub
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

运行程序，并且点击几次按钮，将看见每创建一个 ShowMe 类的新的实例时，在桌面上有一个最小化的窗体的图标。

New 关键字只能用来建立一个类的对象，不能建立一个基于基本的数据类型的对象，例如 Integer。而且，不能建立一个基于一个具体对象的对象。例如，下面的代码根据一个名为 SomeClass 的类，建立了一个名为 Objx 的对象，而后又错误地企图根据对象 Objx 再建立一个其他的对象：

```
Dim ObjX As New SomeClass()
Dim ObjY As New ObjX()        ' 错误 !
```

以上代码是非法的。

6.3.7 释放对对象的引用

每一个对象都使用内存空间和系统资源，每次不再使用一个对象的时候，都应该养成释放资源的习惯。

（1）从内存中卸载一个窗体和控件

使用 Close 陈述，语法如下：

```
Form1.Close()
```

或者

```
ActiveForm.Close()
```

关联窗体的父窗体关闭以后，它就会自动关闭。

（2）释放对象变量使用的资源

可以给一个对象变量赋值为 Nothing，语法如下：

```
MyObject = Nothing
```

（3）释放一个被类使用的对象

如果要释放一个被类使用的对象，只需在类的析构过程里面，把这些对象赋值为 Nothing 就可以了。

注意：.NET 框架使用一个名为 reference tracing garbage collection 的系统调用过程来定期释放那些没有用的资源。所以，把一个对象设置为 Nothing，只是确认系统的垃圾收集器可以把这个对象使用的空间释放掉，并不是真正地释放掉这个对象所使用的资源。垃圾的收集是自动的，它不可能知道什么时候一个对象需要被系统释放资源，所以重要的是要把一个没有用的对象设置为 Nothing，这样的话，垃圾收集器就可以定期把这些对象清空了。

6.3.8　把对象传递到一个过程

在 Visual Basic.NET 中，用户可以传递一个对象到过程中。下面的例子建立了一个窗体类的对象并且把它传递到一个过程中去。为了使用这个例子，用户可以把一个名为 button1 的按钮添加到窗体中，并把下面这些代码复制到 button1_click 事件中去。

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs)
    Dim frmform As New Form1()
    frmform.show()
    CenterForm(frmform)
End Sub
Sub CenterForm(ByVal TheForm As Form)
    ' Centers the form on the screen.
    Dim RecForm As rectangle = Screen.GetBounds(TheForm)
    Theform.Left = CInt((RecForm.Width - Theform.Width) / 2)
    Theform.Top = CInt((RecForm.Height - Theform.Height) / 2)
End Sub
```

用户可以通过先引用一个对象作为一个参数，然后，在过程中，把这个参数声明为一个新的对象。下面的例子就是把一个对象引用到其他的一个窗体的一个过程中去，步骤如下：

（1）建立一个工程，并确认里面有一个窗体名为 form1；

（2）在工程中加入另一个窗体名为 form2；

（3）在这两个窗体中分别加入一个 picture box 控件；

（4）把在窗体 form1 中的 picture box 命名为 picturebox1；

（5）把在窗体 form2 中的 picture box 命名为 picturebox2；

（6）通过在 picturebox2 的属性窗口中点击“image”属性，给该属性赋一个图片，基本上所有的小图片都可以赋值给“image”属性，可以在 Windows 目录下找到.bmp 和.bmp 文件；

（7）把下面的代码加入到 form1 的 form1_Click 事件中去：

```
Protected Sub Form1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs)
    Dim frmform2 As New Form2()
    frmform2.GetPicture(Picturebox1)
End Sub
```

（8）把下列代码加入到 form2 中去：

```
Public Sub GetPicture(ByVal x As PictureBox)
' Assign the passed-in picture box to an object variable.
    objX = x
    ' Assign the value of the Picture property to Form1 picture box.
    objX.Image = picturebox2.image
```

End Sub

运行程序以后，点击 form1，则 form2 的图标将会出现在窗体 form1 上。form1_Click 过程事件调用了 form2 窗体的 GetPicture 过程，而且传递了一个空的 picture box 对象，而后，在 form2 窗体的 GetPicture 过程中，把在 form2 窗体的 picture box 的“image”属性值传到这个空的 picture box 对象中，那么在 form2 窗体的图像就出现在 form1 窗体中的 picture box 对象中了。

6.4 在程序运行中得到一个类的信息

6.4.1 发现对象属于哪个类

通常情况下，这些对象变量（也就是被声明为对象的变量）能够充当很多类的对象。类似地，由 Visual Basic 内部的各种窗体和控件声明的对象变量将包含各类里面的窗体和控件。

当用户再使用这些类型的对象变量的时候，必须根据不同的对象采取特定的方法，例如，有一些对象也许不支持一种特定的属性或方法。Visual Basic 提供了两种方法可以查到对象变量属于哪个类：关键字 TypeOf 和函数 TypeName。

关键字 TypeOf 只能用在 If...Then...Else 表述中。用户必须直接在代码中包含这个类的名称。例如：

If TypeOf MyControl Is CheckBox Then

函数 TypeName 则更加灵活一点。用户可以在代码的任何一个地方使用它，并且由于它以字符的形式返回类的名称，用户可以通过字符之间的比较得到它属于哪个类。

6.4.2 用一个字符名称调用一个属性或者方法

在设计的大部分时候，用户可以发现一个对象的属性和方法，并且可以写一些代码来处理这些方法。然而在一些时候，不能提前知道一个对象的属性和方法，或者只是想在运行程序的时候能具有指定属性和运行不同方法的灵活性。

举一个例子，一个客户端的运行程序通过给一个 COM 组件一个操作来计算一个表达式。假想现在正在给服务器加一个需要新的操作描述的函数，不幸的是，这时必须重新编译和发布客户端软件才能使用新的操作。为了避免这种情况，用户可以使用 CallByName 函数来把新的操作当成一个字符给服务器加载这个函数，而不需要改变运用程序。

CallByName 函数允许用户在运行程序的时候使用一个字符来指定一个属性或者方法，具体使用方法如下：

Result = CallByName(Object, ProcedureName, CallType, Arguments())

第一个参数 Object 表示用户需要操作的对象，第二个参数 ProcedureName 指的是用户要操作的方法、属性或者过程名称的字符表示，CallType 是一个常数选项，当被操作的是一个方法时，值为“VBMethod”；当被操作的是设置属性时，值为“VBLet”；当操作的是得

到属性时，值为“VBGet”；当被操作的是设置对象属性的值时，值为“VBSet”，最后一项选项是可选的。它包括了一个变量数组，该数组包含了过程的所有参数。

假设有一个COM组件叫做MathServer，含有一些平方根运算的函数。程序有两个TextBox控件：第一个控件用来输入需要进行计算的表达式，第二个控件用来输入需要进行的计算方法，就可以在进行计算按钮的点击事件中加入以下代码：

```
Private Sub Command1_Click()
    Text1.Text = CallByName(MathServer, Text2.Text, VBMethod, Text1.Text)
End Sub
```

如果在第一个 Text 中输入“64/4”，在第二个 Text 中输入“SquareRoot”，则上面的代码就会调用求平方根的函数并且返回“4”在第一个 Text 中，但是如果在第一个 Text 中输入了不合法的字符，或者在第二个 Text 中并没有输入一个方法，而是输入了一个属性，或者该方法还需要一个参数的话，则会产生运行错误。正如猜测的一样，如果需要使用这种用字符名称调用一个属性或者方法的话，就必须把出错处理做得非常好。

CallByName 在有一些情况下非常有用，但是必须考虑它的一些缺点，比如使用 CallByName 调用一个过程会比后期绑定慢一些，但如果用户要调用一个经常要重复的函数的话，例如一个内部循环，那么使用 CallByName 将会表现得快一些。

6.5　在 Visual Basic.NET 中创建一个类

6.5.1　给一个类添加方法

类的方法就是在类中用户声明的那些公有的（public）Sub 或者 Function 过程。

例如，下面的代码就是在一个 Account 中加入一个 Withdrawal 方法，用户可以把这个公有的函数过程加入到类模块中去：

```
Public Function WithDrawal(ByVal Amount As Currency, _
ByVal TransactionCode As Byte) As Double
    ' (Code to perform the withdrawal and return the
    ' new balance, or to raise an Overdraft error.)
End Function
```

如果，类 Account 具有“balance”属性的话，用户也可以不用返回 balance 的值，因为用户可以在调用完 WithDrawal 函数以后，非常方便地访问“balance”属性，所以用户可以把这个函数用一个公有的过程来代替。

注意：如果发现把 WithDrawal 设置成过程以后，每次调用 WithDrawal 过程之后还要访问一遍 balance 属性的值的话，那么把 WithDrawal 设置成函数返回 balance 的值，将会更有效率一点。这是因为对于一个类的属性而言，每次得到一个属性的值也是在调用一个公用的函数 Property Get，也是访问一些公有的变量，不管 Property Get 函数是显性声明的，还是隐性声明的。

在类模块中，通过定义属性和方法，就构成了一个类对外的接口，正如类的数据封装性，如果用户把一个过程声明为私有的话，那么它就不能构成接口的一部分。这意味着用户可以通过改变类里面的一些私有的内部过程，而不需要改变使用类的代码就可以达到改变整个程序的运行变化。

更重要的是，用户也可以通过改变公有作为类的方法的 Sub 或者 Function 过程，而不用涉及那些使用类的代码就能改变整个程序的一些功能，正如不能改变过程的参数和函数的返回类型一样，从外部而言，用户也不能改变一个类的接口。

把一个对象的具体情况隐藏在类的接口之外，可以认为是类的封装性的一种表现，封装性允许用户在不改变使用类的代码的情况下，提高类的一个方法的表现，或者完全改变一个类的方法的运行功能。

6.5.2 命名属性，方法和事件

用户给一个类加入的属性、方法和事件定义了这个类的对象的接口的一些操作，当用户在命名这些单元和它的参数的时候，可以发现当按照以下规则命名时，将会带来一些方便之处：

（1）情况允许的情况下，使用完整的一个单词，例如“SpellCheck”，而简写的话，可能会带来一些重复，从而引起混乱。如果整个单词太长的话，就是用第一个音节。

（2）一个名称中有几个单词连写的时候，每个单词的头一个字母用大写，例如“ShortcutMenus”，或者“AsyncReadComplete”。

（3）类的集合使用正确的复数形式，例如“Worksheets”、“Forms”等，如果这个类的名称本身就带有“s”结尾的话，就在类的名称后面加上“Collection”，例如“SeriesCollection”。

（4）尽量使用诸如 verb/object、object/verb 的形式来命名方法的名称。例如“InsertWidget”、“InsertSprocket”等等，或者总是把 object 放在前面，例如“WidgetInsert”或者“SprocketInsert”。

这样命名的好处可以通过看一个单元名称，就能很容易得知它的意义和目的以及作用的对象。

6.5.3 事件和事件处理

有的时候，用户可能觉得 Visual Studio 的工程是一系列的顺序发生的过程，但是在大多数情况下，其实程序是事件驱动的（event driven）—— 也就是说，程序的运行都是取决于一些诸如点击键盘，点击鼠标键以及一些用户定义的事件。不像线性进行的程序那样，它在有些时候不可能决定是要运行哪一个事件驱动的特定的过程。事件对现在的编程而言可以说是一个基本的组成部分，因为这些事件可以让用户写出能对对象之外的世界做出反应的程序，诸如用户的输入和其他对象输入的信息。

1. 事件

事件是用来告知应用程序，一些重要的事情的发生。例如，当用户点击了窗体中的一个控件时，一个 Click 事件被触发了，并且可以调用一个事件处理过程。事件也可以让一些不相关的过程联系在一起。例如，有一个过程需要做很多事情，诸如一系列的操作，用户可

以把一系列的事件组成一个独立的线程。如果用户决定要取消一个操作，应用程序可以发出一个 cancel 事件，来阻止这一系列的操作的进行。

2. 事件发送器

一个可以引发一个事件的对象是一个事件发送器。例如：窗体、控件、以及用户定义的对象。

3. 事件的声明

在类中，事件的声明是用关键字 Event 来进行的。例如：

Event AnEvent(ByVal EventNumber As Integer)

4. 事件的引发

事件就像一件非常重要的事情发生了，而宣布这件重要的事情的行为就叫做事件的引发。从实践的角度来讲，一个事件不能像方法那样被使用，就如同不能用 Button1.Click 去引发一个 botton 的点击事件一样。事件必须使用 RaiseEvent 关键字来引发。下面的例子引发了一个名为 AnEvent 的事件：

RaiseEvent AnEvent(EventNumber)

5. 事件的处理

事件的处理是当一个相关事件发生以后调用的一个过程，如果用户在编辑环境下双击一个窗体上的一个控件，或者从属性窗口的下拉框中选择一个事件的话，Visual Basic.NET 将会自动建立一个事件处理过程。Visual Basic 使用标准的约定的表述来命名这个过程，这种命名方法就是把事件的发生器（引发事件的对象）和事件的名称用下滑线连起来。例如，名为 button1 的点击事件可以这样表示：sub button1_click。用户可以用这种命名法对自己定义的事件进行命名，但是如果用户用了 Handles 关键字的话就没有必要这样做了。

6. 把事件和事件处理联系起来

在一个事件处理可以使用之前，用户需要把它和一个事件用 WithEvents 或者 AddHandler 联系起来。

WithEvents 的陈述和 Handles 从句声明一个事件处理的方法，无论是怎样的 WithEvents 变量，分配的对象都能够激发一个特定的事件的处理。

AddHandler 和 RemoveHandle 陈述能够允许用户动态地连接和断开事件与事件处理的过程。

在有些情况下，例如把一些事件和一些标准的控件联系，Visual Basic.NET 将会把事件处理和一个事件联系在一起。例如，在 Visual Basic 先前的版本的设计环境中，用户可以双击一个按钮控件，然后一个空的事件处理过程就产生了，用户可以在里面加入代码，在代码的背后，Visual Basic 通过隐型地产生一个 WithEvents 变量就可以把事件处理过程和事件联系起来，这个空的实践处理过程如下：

```
Private Sub Command1_Click()
End Sub
```

正如这个过程一样，Visual Basic.NET 也可以通过在设计环境下双击一个按钮控件来建立一个事件处理过程，下面就是当双击一个按钮控件以后，系统自动生成的代码：

```
Private WithEvents Button1 As System.WinForms.Button
Protected Sub Button1_Click(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs)
```

您的浏览器提醒您：
使用本复制品
请尊重相关知识产权!

End Sub

虽然从代码本身来看有很多不同，但是事件处理是一样的。新加的参数对于控件的事件处理而言是必需的，在大多数情况下，用户并不需要使用它们。

7. 给一个类加事件

用户可以通过使用 Events 表述来声明一个类的事件，声明包含了事件的名称和参数。

首先，在定义类的类模块的声明栏中使用 Event 陈述声明一个有参数的事件，例如：

```
Public Event PercentDone(ByVal Percent As Single, _
                         ByRef Cancel As Boolean)
```

事件总是被声明为公有的，而且，事件没有返回值、可选参数、或者数组参数。

给一个类加一个事件说明了只要是这种类的对象就能够引发这种特定的事件，用户需要使用 RaiseEvent 表述来让一个事件真正运行起来，一旦事件运行起来以后，用户就可以使用 WithEvents 或者 AddHandler 把事件和事件处理过程联系起来。

8. 事件处理过程

用户可以使用 WithEvents 和 AddHandler 关键字编写事件处理过程。

用 WithEvents 处理事件：

（1）在能够处理事件的模块的声明栏中，用关键字 WithEvents 加入类的变量，这个变量必须是一个模块级的变量。

（2）在代码编辑环境的左边的下拉框中，选择用 WithEvents 声明的变量。

（3）在代码编辑环境的右边的下拉框中，选择希望能处理的事件（如果可能的话，可以对一个对象处理多种事件）。

（4）把代码加入到事件处理过程中，并使用相关的参数。

使用 AddHandler 处理事件：

AddHandler 和 RemoveHandler 结合起来使用，可以实现 WithEvents 表述的功能。但是允许用户动态地添加，移除，以及改变和事件有关的错误处理。AddHandler 有两个参数、事件发送器（比如控件）的名称和一个表达式，这个表达式表明一种委托关系。当用户使用 AddHandler 时，没有必要显性地声明委托类，对委托关系的引用，可以通过 AddressOf 来返回。下面的例子把一个事件处理过程和一个按钮的点击事件联系在一起，委托关系来自于建立一个新的 System.EventHandler 的实例。

```
AddHandlerButton1.Click,NewSystem.EventHandler(AddressOf
me.Button1ClickEventHandler
```

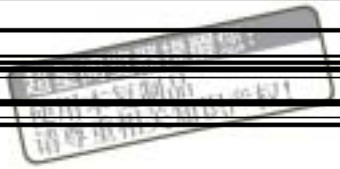
（1）要处理事件的模块的声明栏中加入一个类的变量。

（2）用 AddHandler 表述并且表明事件发送器的名称，以及含有事件处理过程的 AddressOf 表述代码。例如：

```
Dim h As New HH()
AddHandler h.Oevent, AddressOf Me.HandleTheEvent
```

下面的例子建立了一个引发和处理在 0 到 6 秒之间的事件的一个对象：

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs)
    Dim evntobj As New RndmEvnt()
    AddHandler evntobj.RndmEvntThrown, AddressOf Me.HandleEvent
```



一个基础，可以用以下的语法来实现：

Event anevent As DelegateType

因为 EventHandler 是一个在系统命名空间中定义的标准的委托类型，通过 EventHandler 可以传递多种事件。

6.5.5　声明和引发一个事件的例子

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

本例向读者演示怎样为一个名为 widget 的类声明和引发一个事件。

先介绍一下 widget 类，该类具有一个能够运行特别长时间的方法，而且能够得到一些表示该方法运行情况的信息。

当然，用户可以让 widget 类显示一个进程百分比的对话框，这样的话，用户可能在每一个用到 widget 类的工程里面都需要涉及这样的对话框，除非 widget 的整个目的就是要处理窗体或者对话框，一个好的设计习惯就是使应用程序使用一个专门处理用户接口的类。

Widget 类的目的就是要处理一些其他的任务，所以最好能添加一个 PercentDone 事件，而且让一个过程能够调用 Widget 的一个能处理事件和更新状态的方法。PercentDone 事件还能够提供取消任务的功能。

（1）建立一个 Visual Basic.NET 的 Windows 工程，在窗体 form1 中加入两个按钮和一个 label 控件。

（2）在工程的菜单中，给工程加一个类模块。

（3）按照表 6.1 给各个对象命名。

表 6.1　各个对象命名

对　象	属　性	设　置
Class	Name	Widget
First button	Text	Start task
Second button	Text	Cancel
Label	Name Text	LblPercentDone “0”

（4）在类模块中的声明栏中，用 Event 关键字声明一个事件。注意到可以含有 ByVal 和 ByRef 参数，下面就是 Widget 类的 PercentDone 事件：

```
Public Event PercentDone(ByVal Percent As Single, _
                         ByRef Cancel As Boolean)
```

在 Widget 类中加入以下代码：

```
Public Sub LongTask(ByVal Duration As Single, _
                    ByVal MinimumInterval As Single)
    Dim sngThreshold As Single
    Dim sngStart As Single
    Dim blnCancel As Boolean
' The Timer function returns the fractional number
    ' of seconds since Midnight, as a Single.
    sngStart = CSng(Timer)
    sngThreshold = MinimumInterval

    Do While Timer < (sngStart + Duration)
        ' In a real application, some unit of work would
        ' be done here each time through the loop.
        If Timer > (sngStart + sngThreshold) Then
            RaiseEvent PercentDone( _
            sngThreshold / Duration, blnCancel)
            ' Check to see if the operation was canceled.
            If blnCancel Then Exit Sub
            sngThreshold = sngThreshold + MinimumInterval
        End If
    Loop
End Sub
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

则 Widget 类将在 LongTask 方法被调用以后，每隔 MinimumInterval 时间段就会引发 PercentDone 事件，当事件返回以后，LongTask 方法将会检查 Cancel 参数是不是被设为 ture。

6.5.6 事件处理的例子

接着上一节的话题，这一小节主要是讨论怎样编写事件处理的代码。用的例子同上一节一样。

处理 Widget 类中的 PercentDone 事件

把下列代码放在 form1 窗体的声明栏中：

```
Private WithEvents mWidget As Widget
Private mblnCancel As Boolean
```

WithEvents 关键字确定了 mWidget 变量将用来处理对象的事件。注意到 mWidget 变量之所以在窗体的声明栏中定义，主要是因为 WithEvents 变量必须是模块级的变量，这和怎样的模块没有关系，变量 mblnCancel 将用来取消 LongTask 方法。

1. 写出事件处理的代码

当用户用 WithEvents 变量声明了一个变量以后，就可以在代码模块的编辑环境的左边的下拉框中选择 mWidget，则 mWidget 类的事件将会出现在右边的下拉框中，选择一个方

法以后，在编辑环境中将会出现一个以 mWidget 名称和 mWidget 事件名称组合而成的过程代码行。

在 mWidget_PercentDone 事件过程中加入以下代码：

```
Private Sub mWidget_PercentDone(ByVal Percent As _
                                Single, ByRef Cancel As Boolean)
    lblPercentDone.Caption = CInt(100 * Percent) & "%"
    Application.DoEvents
    If mblnCancel Then Cancel = True
End Sub
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

一旦 PercentDone 事件被引发以后，事件处理过程将在一个 label 控件中显示 LongTask 方法进行的进程百分数。注意 DoEvents 表述允许重绘 label 控件，并且给用户提供点击 Cancel 按钮的机会。

把下列代码加入到 Cancel 按钮的 Click 事件中去。

```
Private Sub Command2_Click()
    mblnCancel = True
End Sub
```

如果用户在 LongTask 方法运行的时候，点击了 Cancel 按钮，只要 DoEvents 表述允许事件处理执行的话，则 Command2_Click()事件将会运行起来，而且模块级的变量 mblnCancel 将会被赋值为 True。

2. 关联一个 WithEvents 变量和一个对象

现在 Form1 只是处理 Widget 类的事件，剩下的就是找到 Widget 类的所在之处，当声明了一个 WithEvents 变量以后，并没有对象与其有关，这个 WithEvents 和其他的对象一样，用户必须建立一个对象去引用 WithEvents 对象变量。

首先把下列代码加入到 Form_Activated 事件过程中去，并建立一个 Widget 类的对象：

```
Protected Sub Form1_Activated(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs)
    mWidget = New Widget
End Sub
```

当运行这几行代码的时候，Visual Basic 创建了一个 Widget 的对象并且把 Widget 所有的事件处理过程和 mWidget 对象的事件处理过程联系起来。从这一刻起，每当 Widget 引发了它的 PercentDone 事件的话，mWidget_PercentDone 就会被运行。

为了调用 LongTask 方法，把下列代码加入到 StartTask 按钮的 Click 事件的代码中去：

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs)
    mblnCancel = False
    lblPercentDone.text = "0%"
    lblPercentDone.Refresh()
mWidget.LongTask(12.2, 0.33)
    If Not mblnCancel Then lblPercentDone.text = CStr(100)
End Sub
```

确认 Form1_Activated，Button1_Click 和 Button2_Click 事件在 InitializeComponent 过程中具有 AddHandler 表述。如果需要的话，加入下面几行代码：

```
AddHandler Button1.Click, _
                New System.EventHandler(AddressOf Me.Button1_Click)
AddHandler Button2.Click, _
                New System.EventHandler(AddressOf Me.Button2_Click)
AddHandler Me.Activated, _
                New System.EventHandler(AddressOf Me.Form1_Activated)
```

使用本复制品
请尊重相关知识产权!

在调用 LongTask 方法之前，显示任务完成的百分比的 label 控件必需初始化，模块级的标识是否取消任务的逻辑变量设置为 False。

LongTask 任务将运行 12.2 秒钟，每 1/3 秒钟引发一次 PercentDone 事件。每次引发 PercentDone 事件，都将运行一次 mWidget_PercentDone 过程。

当 LongTask 完成以后，mblnCancel 将被检验一次，看是否 LongTask 是常规的结束，或者是因为 mblnCancel 被设置为 Ture 后被停止。

3. 运行程序

（1）点击【F5】运行程序

（2）点击“Start Task”按钮。每一次 PercentDone 被引发后，label 控件被刷新一次，并且显示任务完成的百分比。

（3）点击“Cancel”按钮停止任务。注意到 Cancel 按钮的外观并没有马上变化。Click 事件直到 Application.DoEvents 表述允许对事件进行处理。

（4）用【F11】运行程序，并且一行一行的运行代码将会是一件很有启发意义的事情。用户将会很清楚地看见程序是怎样进入 LongTask 方法的，并且在每一次 PercentDone 事件被引发时是怎样重新载入到窗体 Form1 中去的。

4. 为不同的 Widget 处理事件

用户可以让变量对象 mWidget 为不同的 Widget 对象的事件作处理，只要把 mWidget 对象变量引用一个新的 Widget 即可，则先前的 Widget 对象的事件将不会被引发，如果 mWidget 是惟一的一个引用以前的 Widget 的对象变量，则对象将会被销毁。

注意：用户可以根据自己需要声明任意个 WithEvent 对象，但是 WithEvent 对象数组是不支持的。

5. 终止 WithEvent 变量的事件处理

当一个 Widget 对象被派遣给变量 mWidget，只要 Widget 引发一个事件则这些事件过程将会和 mWidget 有关。

为了终止一个事件处理过程，用户可以把 mWidget 设置为 Nothing，如下代码：

```
'Terminate event handling for mWidget.
Set mWidget = Nothing
```

当一个 WithEvent 变量被设置成 Nothing 以后，Visual Basic 将会把对象的事件和这个变量的事件过程断开联系。

注意：一个 WithEvent 变量包含了一个对象的引用，就像其他的对象一样，这个对象的引用是使得 WithEvent 变量具有意义的重要因素，当用户把所有的引用都设置成为 Nothing 时，用户就可以把这个 WithEvent 变量销毁，但是不要忘记了声明这个变量时，要带上 WithEvent。

6.5.7 创建一个类的实例

用户除了可以引用一些 VB 自带的类以外，还可以本着面向对象的程序开发的思想，自行开发一些类模块，进行数据和方法的封装。下面首先举一个例子，以说明在 VB 中，类的用法。

例如现在需要建立一个类，来描述“学生”这一个类，该类的属性有 ID（表示学生的一个代号），Name（表示学生的名字），MathMark（表示学生的数学成绩），ChineseMark（表示学生的语文成绩），方法 GetSumMark()表示得到学生的总成绩。再有一个窗体，用以引导整个程序的进程，并和用户进行一些数据上的交流，并把该工程的启动属性设置为“WindowsApplication1.form1”，表示工程的启动，是以窗体 form1 开始的。

首先打开一个新的标准的 VB 的 Windows 程序，在工程浏览器中，用鼠标右击工程的图标，并选择“Add”（如图 6.4 所示），鼠标左击弹出菜单，选择“Add Class...”，点击鼠标左键，弹出选择菜单如图 6.5 所示，选择类的类型以后就可以给当前工程添加类。

图 6.4 给工程添加类

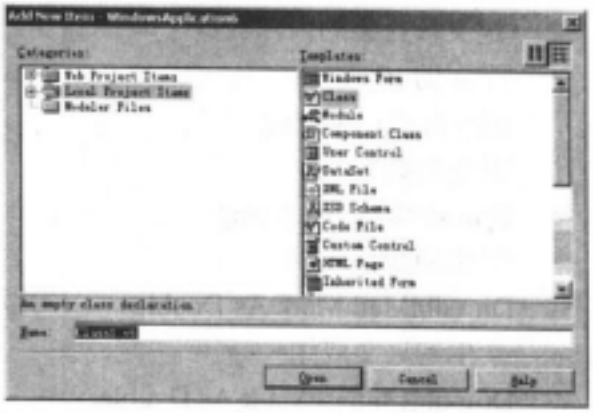


图 6.5 选择类的类型

在图 6.5 中，“Name”的内容为该类所在的文件名而不是这个类的名字，这些类模块的类型分别表示如下：

Windows Form：Windows 窗体类
Class：类
Module：模块
Component：组件
User Control：用户控件
DataSet：数据集
XML File：XML 文件
XSD Schema：XSD 图表
Code File：代码文件
Custom Control:：自定义控件
HTML Page：超级链接文本页
Inherited Form：继承窗体

Crystal Report：报告
Web Custom Control：网络自定义窗体
Inherited User Control：继承用户控件
Windows Service：Windows 服务
Text File：文本文件
Frameset：框架
XSLT File：XSLT 文件
Style Sheet：脚本
Installer Class：安装类
Jscript File：Java 脚本文件
VBScript File：VB 脚本文件
Windows Script Host：Windows 主脚本

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

这里选择 Class，表示添加一个类模块，并在类的名称里键入“clsStudent”，在这个类模块中写入如下代码：

```
Public Class clsStudent
    '定义被封装的内部数据
    '学生的 ID
    Dim lngID As Long
    '学生的姓名
    Dim strName As String
    '学生的数学成绩
    Dim dblMathMark As Double
    '学生的语文成绩
    Dim dblChineseMark As Double
    '学生的总成绩
    Dim dblSummark As Double
    '类的初始化，把各个内部变量和数据初始化
    Public Sub New()
        lngID = 0
        strName = ""
        dblMathMark = 0
        dblChineseMark = 0
        dblSummark = 0
    End Sub

  '编辑属性 ID 的赋值和获取
    Property ID() As Long
        Get
            Return lngID
        End Get
```

```
            Set
                lngID = Value
            End Set
        End Property

        '编辑属性"姓名"的赋值和获取
        Property Name() As String
            Get
                Return strName
            End Get
            Set
                strName = Value
    End Set
        End Property

        '编辑属性"语文成绩"的赋值和获取
        Property ChineseMark() As Double
            Get
                Return dblChineseMark
            End Get
            Set
                dblChineseMark = Value
            End Set
        End Property

        '编辑属性"数学成绩"的赋值和获取
        Property Mathmark() As Double
            Get
                Return dblMathMark
            End Get
            Set
                dblMathMark = Value
            End Set
    End Property
        '定义函数可以得到总成绩
        Public Function GetSunMark() As Double
            Return dblMathMark + dblChineseMark
        End Function

    End Class
```

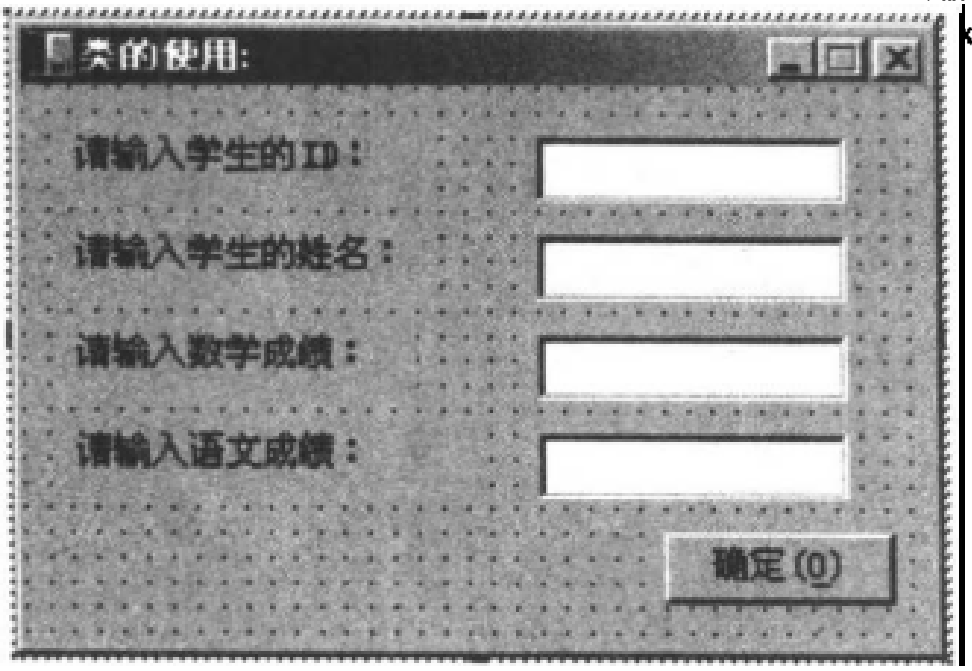
超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

以上这些代码记录了这些属性和方法的设置，对窗体进行编辑界面如图 6.6 所示。

把窗体的“text”设为：“类的使用:”，属性“MaximizeBox”设置为：“False”，其余的设置为默认的属性值。

从上至下的四个 text 控件的“Name”属性分别改为：“txtID”，“txtName”，“txtMath”，“txtChi”。四个 text 控件的“textAlign”属性设为：“Right”，表示 text 的内容向右对齐，这个四个文本框分别表示学生的“ID”，“Name”，“MathMark”，“ChineseMark”属性对应的输入框，

图 6.6




```
Imports Sy
Imports Sy
Imports Sy
Public Cla
    Inheri
    Publi
        I
        I
```

```
        'This call is required by the Win Form Designer.
        InitializeComponent()

        'TODO: Add any initialization after the InitializeComponent() call
    End Sub

    'Form overrides dispose to clean up the component list.
    Overrides Public Sub Dispose()
        MyBase.Dispose
        components.Dispose
    End Sub

#Region " Windows Form Designer generated code "

    'Required by the Windows Form Designer
```

```
Private components As System.ComponentModel.Container
Private WithEvents txtChi As System.WinForms.TextBox
Private WithEvents txtMath As System.WinForms.TextBox
Private WithEvents txtName As System.WinForms.TextBox
Private WithEvents txtID As System.WinForms.TextBox
Private WithEvents btnOK As System.WinForms.Button
Private WithEvents Label4 As System.WinForms.Label
Private WithEvents Label3 As System.WinForms.Label
Private WithEvents Label2 As System.WinForms.Label
Private WithEvents Label1 As System.WinForms.Label

Dim WithEvents Form1 As System.WinForms.Form

'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container()
    Me.Label4 = New System.WinForms.Label()
    Me.txtChi = New System.WinForms.TextBox()
    Me.Label2 = New System.WinForms.Label()
    Me.Label1 = New System.WinForms.Label()
    Me.btnOK = New System.WinForms.Button()
    Me.Label3 = New System.WinForms.Label()
    Me.txtID = New System.WinForms.TextBox()
    Me.txtName = New System.WinForms.TextBox()
    Me.txtMath = New System.WinForms.TextBox()

    '@design Me.TrayHeight = 0
    '@design Me.TrayLargeIcon = False
    '@design Me.TrayAutoArrange = True
    Label4.Location = New System.Drawing.Point(16, 112)
    Label4.Text = "请输入语文成绩："
    Label4.Size = New System.Drawing.Size(136, 23)
    Label4.TabIndex = 3

    txtChi.Location = New System.Drawing.Point(168, 112)
    txtChi.TabIndex = 8
    txtChi.Size = New System.Drawing.Size(100, 21)
    txtChi.TextAlign = System.WinForms.HorizontalAlignment.Right
```

建议浏览器提醒您：
使用本复制品
请尊重相关知识产权！

```
Label2.Location = New System.Drawing.Point(16, 48)
Label2.Text = "请输入学生的姓名："
Label2.Size = New System.Drawing.Size(120, 23)
Label2.TabIndex = 1

Label1.Location = New System.Drawing.Point(16, 16)
Label1.Text = "请输入学生的 ID："
Label1.Size = New System.Drawing.Size(120, 23)
Label1.TabIndex = 0

btnOK.Location = New System.Drawing.Point(208, 144)
btnOK.Size = New System.Drawing.Size(75, 23)
btnOK.TabIndex = 4
btnOK.Text = "确定(&O)"

Label3.Location = New System.Drawing.Point(16, 80)
Label3.Text = "请输入数学成绩："
Label3.Size = New System.Drawing.Size(112, 23)
Label3.TabIndex = 2

txtID.Location = New System.Drawing.Point(168, 16)
txtID.TabIndex = 5
txtID.Size = New System.Drawing.Size(100, 21)
txtID.TextAlign = System.WinForms.HorizontalAlignment.Right

txtName.Location = New System.Drawing.Point(168, 48)
txtName.TabIndex = 6
txtName.Size = New System.Drawing.Size(100, 21)
txtName.TextAlign = System.WinForms.HorizontalAlignment.Right

txtMath.Location = New System.Drawing.Point(168, 80)
txtMath.TabIndex = 7
txtMath.Size = New System.Drawing.Size(100, 21)
txtMath.TextAlign = System.WinForms.HorizontalAlignment.Right
Me.Text = "类的使用:"
Me.MaximizeBox = False
Me.AutoScaleBaseSize = New System.Drawing.Size(6, 14)
Me.ClientSize = New System.Drawing.Size(296, 181)

Me.Controls.Add(txtChi)
```

超星浏览器提醒您：
使用本复制品
请尊重相关知识产权！

```
            Me.Controls.Add(txtMath)
            Me.Controls.Add(txtName)
            Me.Controls.Add(txtID)
            Me.Controls.Add(btnOK)
            Me.Controls.Add(Label4)
            Me.Controls.Add(Label3)
            Me.Controls.Add(Label2)
            Me.Controls.Add(Label1)
        End Sub

    #End Region

        Protected Sub btnOK_Click(ByVal sender As Object, ByVal e As System.EventArgs)
            '定义一个对象是类 clsStudent 的实例
            Dim student As New clsStudent()
            '定义一个字符串以表示要输出的内容
            Dim strLine As String
            '把 TXTID.TEXT 转换成数字型，并给这个对象 clsStudent 的 ID 属性赋值
            student.ID = CLng(txtID.Text)
            '把 txtName.Text 付给 clsStudent 对象的 Name 属性
            student.Name = txtName.Text
            '把 txtChi.text 转换成数字型并给 clsStudent 对象的 ChineseMark 属性赋值
            student.ChineseMark = CDbl(txtChi.Text)
            '把 txtMath.Text 转换成数字型并给 clsStudent 对象的 Mathmark 属性赋值
            student.Mathmark = CDbl(txtMath.Text)
            '刷新 strLine 的内容
            strLine = student.Name & "的 ID 为：" & str(student.ID) & "，语文成绩为：" &
student.ChineseMark _
                        & "，数学成绩为：" & student.Mathmark & "，他的总成绩为：" &
str(student.GetSunMark())
            '输出结论
            msgbox(strLine)
        End Sub
    End Class
```

编译工程，运行程序并在界面输入如图 6.7 所示。

点击“确定”按钮，弹出对话框（如图 6.8 所示）。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

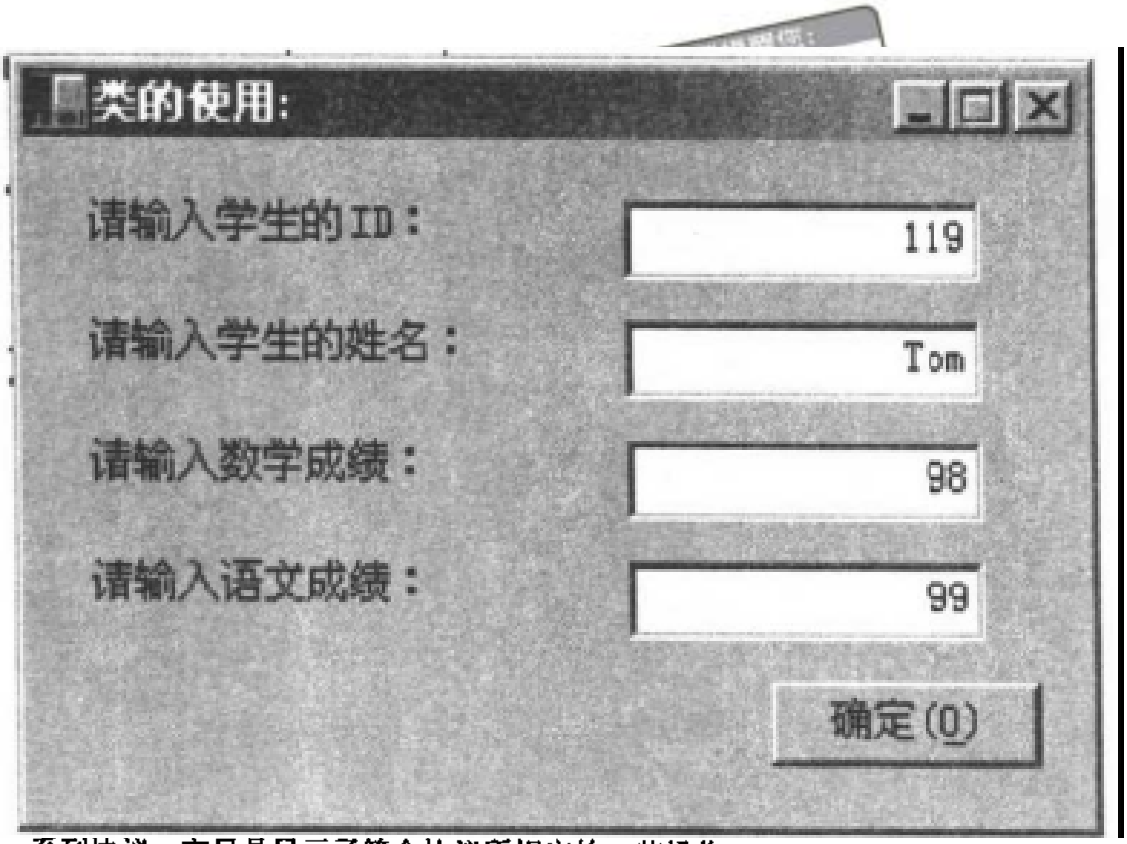


图 6.7

通过上面
是哪种面相对
什么深奥，关
以及对象的划
向对象的思想
而现在这些以
飞跃。

接口描述
是为用户定义一系列协议，它只是显示了符合协议所规定的一些操作。

把接口定义成一系列的密切相关的可以描述组件（component）特征的函数是有好处的，因为用户可以通过以后添加组件的接口函数进行接口的扩充，这样的话就能比较容易地保证接口的兼容性，因为当用户给组件添加和改善接口时，新的接口组件的新版本能够继续执行以前版本的一些功能。

1. 以前版本的 Visual Basic 接口

Visual Basic 以前的版本就已经提供接口的概念，但是并不能直接改变这些接口。只是提供了一个工作区可以建立一个新的类，并且建立一些新的方法。最后这些空的类被定义成了接口，这些接口的命名也和事件过程的命名方式相似，例如在接口 Interface1 中的某个方法 Method1 就可以把它命名为 Interface1_Method1，这种建立接口的方式是可行的，但是对接口类而言，这种方法又是笨拙的和有风险的。

Visual Basic.NET 允许用户使用 Interface 表述来定义真正的接口，并且使用高版本的 Implements 关键字来开发真正的接口。

2. 在 Visual Basic.NET 中定义接口

接口的定义从关键字 Interface 表述开始，并且到 End Interface 结束。接口的定义开始可以是对其他接口的一个继承，然后剩下的表述可以是包括诸如 Event、Sub、Function 以及

Property 等的描述。

接口的描述可以出现在任何模块的声明区域（declarations），并且是公有的（Public），即使它们可以被声明为 Public，Friend，Protected 或者 Private。

惟一的可以在接口定义中(诸如 Sub、Function、或者 Properties)改变的关键字是 Overloads 和 Defult 关键字，而其他的是不能改变的，如，Public、Private、Friend、Protected、Share、Static、Overrides、MustOverride、NotOverrid_able。

3. 接口举例

下面的这个例子定义了两个接口，其中第二个接口 interface2 从 class1 中继承了 interface1。

```
Public Class class1
    Interface interface1
        Sub sub1(ByVal I as Integer)
    End Interface
End Class

Public Class InterfaceClass
    Interface interface2
        Inherits class1.interface1
        Sub M1(ByVal x as Integer) as Double
        ReadOnly Property Num() As Integer
    End interface
End class
```

4. Implements 关键字

Visual Basic.NET 允许用户使用 Implements 表述来给 Implemented 成员函数和属性命名。Implements 表述允许用户列出多重的接口。

Implements 关键字需要用一个逗号来隔开着一系列的名称，一般情况下，列表包含了单独的一个名称，但是多个名称是可以区别的，一个合格的全称应该具有接口的名称，以及接口里的函数名。其中 Implements 成员函数应该可以被合法的引用，而不是像早期版本那样被局限在一个 Interface_MethodName 命名的过程，参数的类型和返回值的类型，必须和接口定义的属性具有相同的类型。

所有的那些在声明方法的属性都能够被用来声明一个接口方法的 Implementation：Overloads、Overrides、NotOverridable、Public、Private、Protected、Friend、Defult 和 Static，但是 Share attribute 是非法的。因为它是用来定义一个类，而不是定义一个 instance 方法。

```
——— Class Addin
Implements AddIn
Public Sub BeginProcess(PID as Long) Implements AddIn.Startup
    ...
End Sub
```

Implements 的例子：

```
Protected Sub M1 Implements I1.M1,I1.M2,I2.M3,I2.M4
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

5. Implementing 接口

那些充当接口的类必须实现所有的属性以及接口的方法，下面的例子，说明了这些设置：

```
Public Class ImplementationClass
    Implements interfaceclass.interface2
    Dim inum as Integer =0
    Sub sub1(ByVal I as Integer)Implements interfaceclass.interface2.sub1
End sub
    Sub M1(ByVal x as Integer) Implements interfaceclass.interface2.M1
        X+=1
    End sub
    Function M2(ByVal y as Integer) as double Implements _
interfaceclass.interface
            m2=cdbl(y+1)
      end function
    ReadOnly Property Num() As Integer Implements_
                Interfaceclass.interface2.num
        Get
            Num=inum
        End Get
      End Get
    End Property
    End  Class
```

6.7　类的继承（Inheritance）

Visual Basic.NET 支持类的继承，允许用户定义一个类，作为其他类继承的依据，继承类能够继承，扩充基本类的属性和方法，继承类也可以重载基本类的一些方法，所有在 Visual Basic.NET 建立的类缺省的属性都是可继承的，因为窗体现在看成了一个类，所以在 Visual Basic.NET 中用户可以根据现有的窗体继承出一个新的窗体。

继承的一个目的就是使代码可以重用。有两种方法可以提高代码的重用率，合成（composition）和继承（Inheritance），一直以来，Visual Basic 都支持合成这种方法来达到代码的重用，用户可以建立和重用 COM 组件，而继承却是 Visual Basic.NET 的新特性，因为这样的话，用户就可以建立一个新的类作为其他类的基本类。

继承和接口（Interface）都允许使用多态，也就是说可以定义两个或多个类，而且，每一个类都定义了一个具有相同名字的方法或属性，当在客户端运行该方法或属性时，可以根

据特殊情况，选择不同的类的方法和属性。多态在面相对象的编程中是必需的，因为它允许用户使用相同的名字进行调用属性或方法，无论何时何种对象在使用。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

6.7.1 继承的规则

用户可以使用一个新的关键字“Inherts”在一个基本的类的基础上去建立一个新的类，则继承类可以继承，扩充基本类的属性、方法、事件、数据成员和事件处理程序。

Visual Basic.NET 引入以下表述来支持继承的表述：

（1）Inherits 表述 —— 用来说明当前的类是继承了哪一个已经存在的类（基本类），Inherits 只有在类中才能使用。

（2）NotInheritable modifier —— 防止程序把一个类作为一个基本类。

（3）MustInherit modifier —— 声明这个类是不能创建的，必须继承下来的，惟一一种可以使用该类的方法就是继承它。

新的继承的类可以重载一些基本类的方法，Visual Basic.NET 使用以下这些关键字来控制属性和方法的重载：

（1）Overridable ——允许一个属性或者方法在继承类中可以重载。公用的方法的缺省的值为 NotOverridable。

（2）Overrides —— 允许用户重载一个基本类的属性或者是方法。

（3）NotOverridable（缺省）—— 防止一个属性或方法在继承类中被重载。

（4）MustOverride —— 当该关键字被使用时，需要继承的类一定要把该属性或方法重载，其中方法的表述只包括 Sub，Function，或者 Property。

注意：其他的描述是不允许的，这里没有 End Sub、End Function 描述。有 MustOverride 方法的类必须被声明为 MustInherit，公有的方法的缺省值为 NotOverridable。

并且关于继承，还有下列准则：

（1）虽然，一个继承类只能够从一个类继承而来，但是它可以产生无数个接口。

（2）一个公有的类不能继承一个 friend 的或者私有的类，而且一个 friend 类不能继承一个私有的类。

1. 使用 MyBase 访问基本类的方法

当用户在继承类中重载基本类的方法时，可以使用 MyBase 调用基本类中的方法，以下的表述将向用户提供关于 MyBase 的一些更详细的描述。

（1）MyBase 是对基本的类和它的继承成员的引用，而且可以访问它的基本类定义的公有成员。它不能访问基本类的私有成员。

（2）在 MyBase 中限定的一些方法，没有必要在 MyBase 中再进行定义；它可以间接地在继承类中进行定义，为了使 MyBase 可以正确地引用和编译，一些基础类必须在引用时包含一个和其参数名称和类型匹配的方法。

（3）MyBase 不能用来限定本身，所以下面描述是错误的：

MyBase. MyBase.BtnOK_Click()

（4）MyBase 是一个关键字。

（5）MyBase 不能被用成一个变量，或者是过程，或者用在“Is”比较中，MyBase 并不是一个真正的对象。

（6）MyBase 可以被用作一个共享成员（shared members）；这时它是有值的，因为共享成员是可以被 shadowed。

（7）MyBase 不能在模块中使用。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

2. 使用 MyClass

（1）MyClass 允许调用一个可以重载的方法，并且确认调用的是方法里的 implementation 过程，而不是继承类里的重载的方法，以下的使用方法是合法的，用 MyClass 在一个类中去限定一个方法，这个方法在基本类里面有定义，但是在这个类中没有这个方法的定义。这种引用和 MyBase.Method 具有一样的意义。

（2）MyClass 是一个关键字。

（3）MyClass 不能被用成一个变量，或者是过程，或者用在“Is”比较中，MyBase 并不是一个真正的对象。

（4）MyClass 可以引用包含的类以及它的继承成员，并且能够被用作访问在类中定义的公有的成员，但是不能访问类中的私有成员。

（5）MyClass 可以被用作共享成员的限定。

（6）MyClass 不能被用在标准模块中。

6.7.2　用继承建立一个继承类

“Inherits”关键字可以用在使一个类继承另一个类的属性、方法、事件等等，所有的类缺省的都是可以继承的，除非被设置为“NotInheritable”关键字。

下面这个例子定义了两个类，第一个类是一个基础类，并且含有一个属性和两个方法，第二个类从第一个类继承了这个属性和两个方法，重载了第二个方法，并且定义了一个新的属性“intProp2”。

```
Class Class1
    Private intProp1 as Integer
    Sub Method1()
        Messagebox.Show("这是在基本类中的一个方法")
    End Sub
    Overridable Sub aMethod()
        Messagebox.Show("这是在基本类中的另一个方法")
    End Sub
    Property Prop1 as Integer
        Get
          Prop1 = intProp1
        End Get
        Set
            intProp1 = Value
        End Set
    End Property
End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
Class Class2
   Private intProp2 as Integer
   Inherits Class1
   Property Prop2 as Integer
        Get
             Prop2 = intProp2
        End Get
        Set
            IntProp2 = Value
        End Set
  End Property
   Overrides Sub aMethod()
        Messagebox.Show("这是在继承类中的一个方法")
  End Sub
End Class

Protected Sub TestInheritance()
    Dim C1 As New class1()
    Dim C2 As New class2()
    C1.Method1()
    C1.aMethod()
    C2.Method1()
    C2.aMethod()
End Sub
```

当用户运行过程 TestInheritance 以后，可以看见如下的信息：

“这是在基本类中的一个方法”

“这是在基本类中的另一个方法”

“这是在基本类中的一个方法”

“这是在继承类中的一个方法”

6.7.3 重载 Windows 控件

在 Visual Basic.NET 中，用户可以根据一个已经存在的控件继承出一个新的 Windows 控件，假设现在需要一种 Text 控件，当用【Tab】键移到该控件时，整个 Text 中的内容都被 highlight 了，在 VB 6.0 中，这可以通过写一个新的 DLL 动态库，但是 Textbox 控件在工具栏中被屏蔽掉了，并且失去了 Textbox 的一些有用的事件。而在 VB 7.0 中则可以通过继承 Textbox 类而派生出一个新的类。

下面举一个例子，通过 Windows 设计环境，建立一个窗体，在这个窗体中有两个 TextBox，通过在“Project\Add User Control”菜单给工程加入一个名为“HitextBox.VB”的对象，如图 6.9 所示。

使这个控件继承 TextBox 的属性，而不是要自定义一个全新的控件，如下：

```
Public class HtextBox
Inherits Textbox
```

使用本复制品
请尊重相关知识产权!

下面先编译一下整个工程，新加入的 HtextBox 控件将会出现在左边的工具栏的底部。如图6.10所示。用户可以在建立的任何一个 Windows 窗体中建立一个 HtextBox 类的对象。

在类的代码中，插入如下代码以实现 highlight 的功能：

当你用【Tab】移到的 Textbox 时，Textbox 的全部内容变为 highlight。

```
Public class HtextBox
    Inherits Textbox
    Public Sub New
        Mybase.New
'Add Event event handler
AddHandler Enter, _
    New System.EventHandler (AddressOf _
                Me.HT_Enter)
    End Sub
'-----------------------------
'Enter event handler is inside the class
Protected Sub HT_Enter(ByVal sender as Object , _
        ByVal e as System.EventArgs)
    Me.selectionStart=0
    Me.selectionLength = Me.text.length
End Sub
    End Class
```

这就是所有的过程，通过添加大概 10 行代码就做出了一个继承 Windows 控件派生出来的控件，读者可以看一看如图6.11所示的结果，其中上面的 Textbox 控件是 Windows 自带的控件，而下面的控件是用户自己做的 Text 控件。

图 6.9　加入新的控件继承对象　　图 6.10 产生新的控件　　图6.11　运行结果

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

6.7.4 什么时候使用继承

继承是一个非常有用的概念，但是却很容易被用得不合适，通常用接口来实现可能会更好，本小节的目的就是使用户懂得怎样更好地使用类的继承。

当遇到如下情况时，继承将是一个好的选择：

（1）简化一个低等级的不使用类的 API 函数。

（2）从基本类中得到重用的代码。

（3）需要对不同的数据类型使用相同名称的类和方法。

（4）类的层次相当，最多 4 到 5 级，而且不仅增加一级或两级。

（5）通过改变一个基本类，就改变所有的派生类。

1. 简化一个低等级的不使用类的 API 函数

类库使得一系列的函数的调用更加有条理，特别是当调用的这些函数不是以类的形式组织的时候，举一个例子，就像那些为简化在窗体中画图表的 API 函数而设计的类一样，这种类库在有些时候被叫做“Wrapper”。这些把 API 函数包括起来的类库通常可以简化操作，不过这样做并没有企图把 API 函数的一些基本特性给隐藏掉。Windows graphics API 函数有很多，但用户并不需要了解它底层的一些变化。用户在设计类的时候也是这样，必须考虑类库的用户对底层的了解程度。实际上，很多类库都是根据 API 函数来组织的，通常它们有两个趋势：使得 API 函数对高级用户更加方便，或者通过高度的概括，把这些底层的信息隐含起来，这意味着把 Windows API 函数的具体信息对用户封装。

但是这里又有一个隐含的关于开发类库的问题，被打成包的类导致了非常多的文字说明，把 API 简化得越多，需要的文字说明就越多。经常性的，几百个 API 函数被组织成一打或一百个类，每一个类具有几十个或几百个重叠的属性、方法和事件，整理或者浏览这个类将是工作量非常大的一件事。

设计和维护一个把各种 API 打包的类库，是一件工作量非常大事情，设想现在要写一个类库来简化一个为 Web 服务的 HTML 的网页的生成过程，这就需要先考虑一下自己是不是需要知道“HTML”；是不是需要把这些诸如“tags”或“angle brackets”的概念隐含掉；是不是向用户显示一些诸如字体等这些属性；是不是只要能处理 HTML 就行了，还是需要处理 XML，能不能叫 XHML？等等。

假设一个基本类含有一个 MustOverride 方法叫做 ParseHTML，然而当需要支持 XML 时，就必须把 HTML 支持替换掉，此时需要把方法 ParseHTML 清空，惟一的选择是建立一个 ParseXML 方法，但不能把 ParseHTML 方法移除掉，因为当把该方法移除以后，就会导致 HTML 业务的丧失，因为整个类是分布的，用户必须让 ParseHTML 和 ParseXML 方法同时存在，但这让内存和运行时间效率都减低了。

诸如这种问题在网络开发环境还是存在的，用户可能是过早设计了 HTML 类库，也许一个基于接口的系统可以更好地解决这个问题。它让用户建立一系列的 HTML 接口，当在确认 XHTML 或者 XML 是更好的基底时，就可以放弃或改善 HTML 接口。

2. 从基本类中得到重用的代码

使用继承的一个重要的原因就是为了代码的重用，然而这也是最危险的。正是因为代码的重用，即使是设计的最好的系统，有的时候也会出现一些设计者难以预料的改变。

一个经典的表示代码重用的高效率的例子就是一个数据管理库，设想现在有一个大型的应用程序，用来管理几个在内存中的清单，另一个是从顾客的数据中拷贝来的清单，该数据结构可能如下所示：

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
Class CustomerInfo
    Public PreviousCustomer As CustomerInfo
    Public NextCustomer As CustomerInfo
    Public ID As Integer
    Public FullName As String
     ' Adds a CustomerInfo to list
    Function InsertCustomer As CustomerInfo
     ...
    End Function
    ' Removes a CustomerInfo from list
     Function DeleteCustomer As CustomerInfo
     ...
     End Function
      ' Obtains next CustomerInfo in list
    Function GetNextCustomer As CustomerInfo
          ...
    End Function
      ' Obtains previous CustomerInfo
    Function GetPrevCustomer As CustomerInfo
     ...
    End Function
End Class
```

可能还有一张顾客的采购的东西的卡片清单，如下：

```
Class ShoppingCartItem
    PreviousItem As ShoppingCartItem
    NextItem As ShoppingCartItem
    ProductCode As Integer
    Function GetNextItem As ShoppingCartItem
          ...
    End Function
...
End Class
```

从这里，就可以看见类的结构和模式，都有相同之处 —— insertions（插入）、deletion（删除），以及 traversal（浏览），只不过是对不同数据类型进行操作而已，显然，要维护这两段几乎具有相同函数的代码很没有必要，又一个显然的解决方案就是把对清单的处理抽象出来，做成一个它自己的类，然后，就可用这个类派生出不同数据类型的子类。如：

```
Class ListItem
PreviousItem As ListItem
    NextItem As ListItem
    Function GetNextItem As ListItem
...
    End Function
...
End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

这样的话只需要对 ListItem 类调试一次，而且用户只需要把它编译一次以后，就不再需要管理关于清单管理的代码，用户只需要使用它就可以了：

```
Class CustomerInfo
    Inherits ListItem
    ID As Integer
    FullName As String
End Class
Class ShoppingCartItem
    Inherits ListItem
    ProductCode As Integer
End Class
```

3. 对不同的数据类型使用相同名称的类和方法

一个决定是否要使用继承办法就是是否有以下问题：

（1）是否需要对不同的数据类型进行相似的操作；

（2）需不需要（想不想）访问程序源代码。

举一个非常具有代表性的例子就是画图的软件包，假想它可以画圆、线和长方形。一个非常简单的而且效率很高的方法就是不需要使用继承，而用 Select Case 语句来实现，如下：

```
Sub Draw(Shape As DrawingShape, X As Integer, Y As Integer, _
Size As Integer)
      Select Case Shape.Type
 Case shpCircle
    ' Circle drawing code here
      End Case
 Case shpLine
 ' Line drawing code here
 End Case
 ...
 End Sub
```

但是这样处理的话将会出现一些问题，将来如果需要添加画椭圆功能的话，那么就必须重新改写代码，也许最终用户不需要访问程序的源代码，但是可能画一个椭圆还需要一些参数，因为椭圆需要的参数是一个长半径和一个短半径。但是这些参数又和其他图形的参数无关，如果现在又要画一个折线（有多条线组成），则又需要加入一些其他的参数，而且这

些参数又和其他的图形的参数无关。

继承就可以非常好地解决这些问题，一个设计得很好的类可以给用户留下一个很好的空间（MustInherit 方法），那样的话，每一种图形都可以满足，但是一些类的属性（诸如 XY 坐标）可以作为基本类的属性，因为这个属性是每一种图形都需要的，如：

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
Class Shape
    Sub MustInherit Draw()
        X As Integer
        Y As Integer
End class
```

其他的图形都可以继承这个类，如要画一条线，代码如下：

```
Class Line Extends Shape()
      Length As Integer
Sub Overrides Draw
' Implement Draw here
      End Sub
End Class
```

长方形的类，又应该如下表示：

```
Class Rectangle Extends Line()
      Width As Integer
      Sub Overrides Draw()
      ' Implement Draw here
      End Sub
End Class
```

它的继承子类（例如，长方形）能够在图像类的二进制的文件的基础上继承，而不是在基础类的代码的基础上继承。

4. 基类的方法需要改变

Visual Basic（以及其他的提供继承的编程语言）使得用户有能力重载基本类的方法，只要在派生类中可以访问的基类的方法都可以被重载（除了那些 NotInheritable）。假如用户感觉基本类的 DisplayError 方法需要给出一个错误号，而不只是错误的描述，则用户可以在派生类中改变这种方法的行为，则派生类会调用它本身的方法，而不是基类的方法。

5. 通过改变一个基本类，就改变了所有的派生类

继承的一个强大的作用就是当改变一个基本类的属性时，就改变了所有的派生类的相应的结构，但是，这也是一个危险的行为 —— 有可能改变了自己设计的基类以后，别人由该基类而派生出来的派生类就会产生错误。

6.7.5 命名空间（Namespace）

命名空间是为了把一些类和类的实例更好地管理而定义的把这些类和实体集合起来的一个团体，它是_Namespace system class 的一个实例，或者是_Namespace 类的一个派生类，_Namespace 类只有一个属性：Name。通过这个属性用户可以把某个命名空间和其他的命

名空间区分开来，注意命名空间的Name不能包括头缀和后缀的下滑线。

命名空间可以相互联系和组织成一个具有任意层次的由类和类的实例组成的网状结构，这种结构有点像文件系统的树状结构，在同一个层次的情况下，命名空间必须具有互异的名称。

为了表示它们的层次结构你可以用右下线表示这种关系，如下：

Namespace1\Namespace2\Namespace3.....\LastNamespace

虽然说这些命名空间可以表示成这样的层次结构，但是，这些命名空间之间并没有类的继承关系，也就是说，子的命名空间中的类不会自动继承父的命名空间中的类。

通常情况下，一个命名空间包含了在某种环境下一系列的类和类的实例。例如那些在Win32下定义和运行的类即使是和其他的命名空间里的类具有相同的名字也不会出现冲突，然而在建立一个新的类的时候，最好还是不要和已经建立的类的名字相同，这为将来WMI的发布减少了一些问题。

所有的WMI包含了下面这些预定义了的命名空间：

- root
- toor\default
- root\cimv32

root命名空间是专门为包含其他的命名空间而设计的命名空间，WMI把其他的命名空间都放在这个命名空间下面。toor\default命名空间包含了基本上系统的类，而root\cimv32命名空间主要是包含了在Win32环境下运行的一些类。比如：Win32_LogicalDisk和Win32_OperatingSystem。很多操作基本上都是在root\cimv32命名空间下发生的。

6.7.6 关于继承的例子

这次列举一个在窗体上画出一些图形，如：长方形，正方形等的例子。

在Visual Basic.NET中，控件可以被Windows系统重新绘制，用户可以通过重载OnPaint事件来画自己的图形。PaintEventArgs是系统底层调入子例程的一个对象，用户可以通过这个对象在窗体的表面画一些图形，用户还需要建立一个Pen对象，它的颜色的缺省值是黑色，线条的宽度的缺省值是1。下面这个例子说明了这个对象的用法。

```
Protected Overrides Sub OnPaint(e as PaintEventArgs)
Dim g as Graphics = e.Graphics
Dim rpen As new Pen(Color.Black)
g.drawLine(rpen, 10,20,70,80)
End Sub
```

下面画一个简单的长方形，先写一个长方形的类，它可以把自己画在Windows窗体上，只有两个方法，一个是构造函数，一个是draw方法，先声明一个命名空间，然后再写下长方形的类的实现代码，如下：

```
Namespace VBPatterns
Public Class Rectangle
Private x, y, h, w As Integer
Protected rpen As Pen
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

```
Public Sub New(ByVal x_ As Integer, _
ByVal y_ As Integer, _
ByVal h_ As Integer, _
ByVal w_ As Integer)
MyBase.New()
x = x_
y = y_
h = h_
w = w_
rpen = New Pen(Color.Black)
End Sub
End Class
End Namespace
```

现在每次在 Main Form 窗体中声明一个变量，就把它声明为命名空间中的一个成员。下面，首先把这个长方形的类声明为 VBPatterns 命名空间的一个成员：

```
Public Class RectForm
Inherits System.WinForms.Form
Private rect As VBPatterns.Rectangle
```

并且建立一个长方形类的对象：

```
Public Sub New()
MyBase.New()
'This call is required by the Win Form Designer.
InitializeComponent()
rect = New VBPatterns.Rectangle(40, 20, 30, 80)
End Sub
```

现在重载 OnPaint 事件：

```
Protected Overrides Sub OnPaint( _
ByVal e As PaintEventArgs)
Dim g As Graphics
g = e.Graphics
rect.draw(g)
End Sub
```

画长方形的结果如图6.12所示。

图6.12 画长方形

超星阅览器提醒您:
使用本复制品
须遵守相关知识产权!

从长方形建立一个正方形:

正方形只不过是长方形的一个实例，可以从长方形的类派生出正方形的类而不需要写太多的代码。下面就是实现的代码:

```
Namespace VBPatterns
Public Class Square
Inherits Rectangle
Public Sub New(ByVal x As Integer, _
ByVal y As Integer, ByVal w As Integer)
MyBase.New(x, y, w, w)
End Sub
End Class
End Namespace
```

正方形的类实际上只包含了一个构造函数，不过它还调用了长方形的一些方法，注意到正方形类实际上没有 draw 方法，如果用户不声明一个新的方法，则基本类的方法就会自动被运用。

下面是通过构造函数建立了一个长方形和一个正方形:

```
Public Sub New()
InitializeComponent
rect = New Rectangle(10, 10, 30, 80)
sq = New Square (50, 50, 50)
End Sub
```

并重载 OnPaint 方法:

```
Protected Overrides Sub OnPaint( _
ByVal e As PaintEventArgs)
Dim g As Graphics
g = e.Graphics
rect.draw(g)
sq.draw(g)
End Sub
```

运行以后的结果如图 6.13 所示。

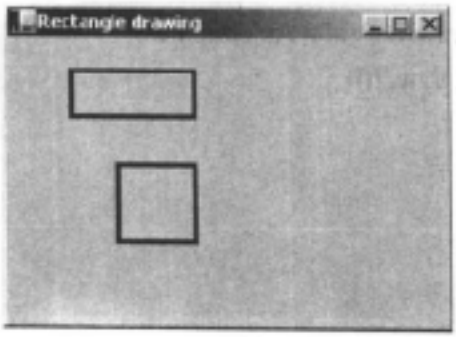


图 6.13 同时画一个长方形和正方形

在 Visual Basic 6.0 中，用户可以把一个变量或者类声明为公有或者是私有，在类中公有变量可以被其他的类访问，而私有变量只能在类中被访问。在 Visual Basic.NET 中引入了关键字 protected 变量和方法都能够被保护，protected 能在类中和它派生出来的类访问，也就是说 protected 变量不能在类的外面被公有地访问。

假设现在需要派生一个新的类叫做 DoubleRect，它是从类 Rectangle 派生出来的，它的功能是画两个稍微错开的长方形，而且颜色不一样。在构造函数中，将用红色来处理长方形的颜色，代码如下：

```
Namespace VBPatterns
Public Class DoubleRect
Inherits Rectangle
Private redPen As Pen
'----
Public Sub New(ByVal x As Integer, _
ByVal y As Integer, ByVal w As Integer, _
ByVal h As Integer)
MyBase.New(x, y, w, h)
redPen = New Pen(Color.FromARGB(255, _
Color.Red), 2)
End Sub
```

这意味着新类 DoubleRect 将具有自己的 draw 方法，现在基本的类已经有一个 draw 方法了，所以必须再建立一个具有相同名字的新方法，所以重载这种方法，代码如下：

```
Public Overrides Sub draw(ByVal g As Graphics)
MyBase.draw(g)
g.drawRectangle(redPen, x + 4, y + 4, w, h)
End Sub
```

注意：这里想使用坐标系，而且长方形的大小已经在构造函数时被给定了，但有时用户需要改变这些参数，那么可以在 DoubleRect 中保留这些参数的备份，自己再声明一些参数，或者在基本类 Rectangle 中把这些参数设为 protected 型，如：

```
Protected x, y, h, w As Integer
```

还必须告诉编译器，通过声明 Rectangle 的 draw 方法为 overridable，这样就可以给 draw 方法重载了，代码如下：

```
Public Overridable Sub draw(ByVal g As Graphics)
g.DrawRectangle(rpen, x, y, w, h)
End Sub
```

最后的结果如图 6.14 所示。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

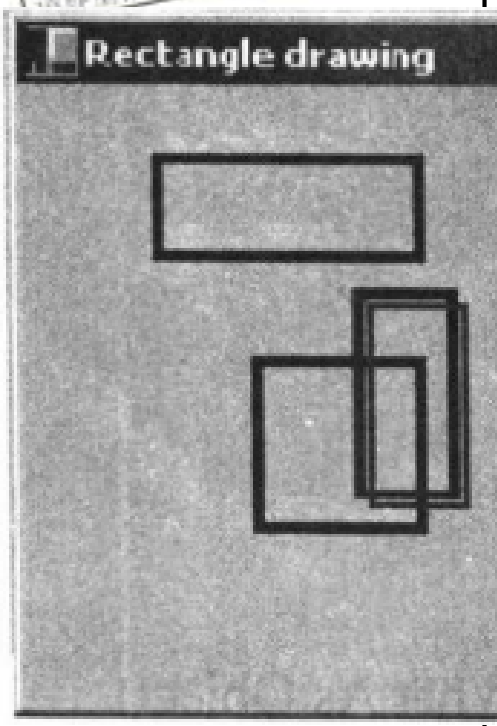


图6.14 显示最后继承结果

6.8 多 态 性

多态性表示很多的类可以具有一个相同的属性或方法，而且调用者在调用之前无需知道这个对象属于哪个类。

Visual Basic 中传统的使用多态性的办法就是使用接口。现在，现这种功能，但现在也可以用继承来实现多态性。正如处理一些其他的使用接口和继承都各有优势，当在创建一个以后派生类可以扩充的基用继承。而当用户要用多种并不是很相同的工具来实现多种功能相似会更好一点。

6.8.1 用继承实现多态性

大部分面向对象的程序开发系统都是通过继承来实现多态性。比如说跳蚤类和狗类都是从动物类继承过来的。为了突出每一种动物走动的特点，则每一种特定动物类都要重载动物类的“Move”方法。

多态性的问题是因为用户可能需要在还不知道是要对哪种特定动物进行处理的时候，就要调用多种从动物类中派生出来的特定的动物类中的“Move”方法。

在下面的这个 TestPolymorphism 过程中，用继承来实现多态性：

```
MustInherit Public Class Animal ' 基本类
        MustOverride Public Sub Bite(ByVal What As Object)
        MustOverride Public Sub Move(ByRef Distance As Double)
    End Class

    Public Class Flea
        Inherits Animal
        Overrides Sub bite(ByVal What As Object)
            ' Bite something
        End Sub
```

```
        Overrides Sub Move(ByRef Distance As Double)
            distance = Distance + 1
        End Sub
    End Class

    Public Class Dog
        Inherits Animal
        Overrides Public Sub bite(ByVal What As Object)
            ' Bite something
        End Sub
        Overrides Sub Move(ByRef Distance As Double)
            distance = Distance + 100
        End Sub
    End Class

Sub TestPolymorphism()
    Dim aDog As New Dog()
    Dim aFlea As New Flea()
    UseAnimal(aFlea) 'Pass a flea object to UseAnimal procedure
    UseAnimal(aDog) 'Pass a Dog object to UseAnimal procedure
End Sub

Sub UseAnimal(ByVal AnAnimal As Animal)
    Dim distance As Double = 0
    '    UseAnimal does not care what kind of animal it is using.
    '    The Move method of both the Flea and the Dog are inherited
    '    from the Animal class and can be used interchangeably.
    AnAnimal.Move(distance)
    If distance = 1 Then
        MessageBox.Show("The animal moved: " & CStr(distance) & _
                        " units, so it must be a Flea.")
    ElseIf distance > 1 Then
        MessageBox.Show("The animal moved: " & CStr(distance) & _
                        " units, so it must be a Dog.")
    End If
End Sub
```

超星阅览器提醒：
使用本复制品
请尊重相关知识产权!

6.8.2　用接口来实现多态性

用户可以在 Visual Basic.NET 中使用接口来完成多态性的实现。通过使用多接口，用户

可以在不中断运行代码的情况下允许运行多种软件的系统组件。

接口像类那样描述属性和方法，但是它和类不同的是，接口不能提供任何implementation。

使用本复制品
相关知识产权!

为了使用接口来实现多态性，用户需先建立一个接口，并且通过其他的几个类实现该接口。用户可以用几乎相同的方法调用其他对象已经实现的方法。

下面这个例子就是使用接口的方法实现多态性：

```
Namespace PolyNamespace
    Interface Animal
        Sub Move(ByRef Distance As Double)
        Sub Bite(ByVal What As Object)
    End Interface

    Class Flea
        Implements animal
        Public Sub bite(ByVal What As Object) Implements animal.bite
            ' Bite something
        End Sub
        Sub Move(ByRef Distance As Double) Implements animal.move
            distance = Distance + 1
        End Sub
    End Class

    Class Dog
        Implements animal
        Public Sub bite(ByVal What As Object) Implements animal.bite
            ' Bite something
        End Sub
        Sub Move(ByRef Distance As Double) Implements animal.move
            distance = Distance + 100
        End Sub
    End Class

End Namespace

' add this section to the your form
Protected Sub Button1_Click(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs)
        Dim aFlea As New Flea()
        Dim anobj As Object()
        Dim aDog As New Dog()
```

```
        GetFood(aflea, anobj)
        GetFood(aDog, anobj)
End Sub
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
    Public Sub GetFood(ByVal Critter As Animal, ByVal Food As Object)
        Dim dblDistance As Double
        ' Code to calculate distance to food (omitted).
        Critter.Move(dblDistance) ' Early bound (vtable).
        Critter.Bite(Food) ' Early bound (vtable).
    End Sub
```

第 7 章　数据库访问技术

您的浏览器提醒您：
盗版制品
请尊重相关知识产权！

本章包括：
理解数据库
VB 7.0 开发数据库初步，数据绑定显示
SQL 语言基础
SQL 语句函数
表、视图与索引
数据库访问对象 DAO
远程数据对象 RDO
ODBC API
ADO 数据对象
ADO.NET
用面向对象思想处理数据库

7.1　理解数据库

几乎所有的商业应用程序都需要处理大量的数据，并将其组织成易于读取的格式。这种要求通常可以通过数据库管理系统（MDBS）实现。MDBS 是用高级命令操作表格式数据的机制。数据库管理系统隐藏了数据在数据库中的存放方式之类的底层细节，使编程人员能够集中精力管理信息，而不是考虑文件的具体操作或数据连接关系的维护。

下面，先介绍几个基本的概念。

数据库：数据库就是一组排列成易于处理或读取的相关信息。数据库中的实际数据存放成表格（table），类似于随机访问文件。表格中的数据由行（row）和列（column）元素组成，行中包含结构相同的信息块，类似于随机访问文件中的记录，记录则是一组数值（或称为字段）的集合（如图 7.1 所示）。

记录集：记录集（RecordSet）是表示一个或几个表格中的对象集合的多个对象。在数据库编程中，记录集等于程序中的变量。数据库中的表格不允许直接访问，而只能通过记录集对象进行记录的浏览和操作。记录集是由行和列构成的，它和表格相似，但可以包含多个表格中的数据。如图 7.2 所示网格的内容来自于一个表格，形成一个记录集。图中所示的查询结果是所有作者的资料。

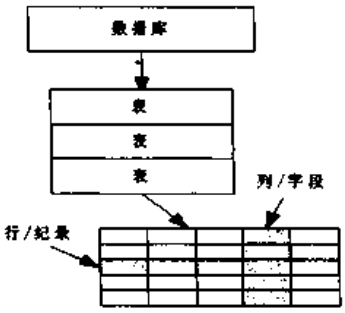


图 7.1　数据库和表格结构的图形表示

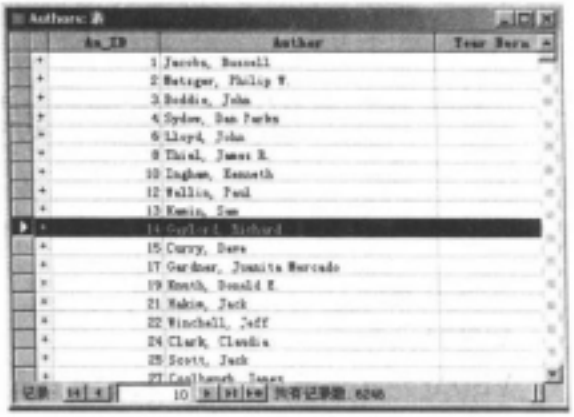


图 7.2　BIBLIO 数据库的 Authors 表，所选的行是 Authors 的相关记录

注意：可以把记录看成一种浏览数据库的工具，用户可根据需要指定要选择的数据，记录集的类型有三类：

（1）DynaSets：这是可修改的显示数据；

（2）SnapShots：这是静态（只读）的显示数据；

（3）Tables：这是表格的直接显示数据。

DynaSets 和 SnapShots 通常用 SQL（结构化查询语言）语句生成，SQL 将在以后介绍，但现在只要知道 SQL 语句是从指定数据库中读取数据的标准命令即可。DynaSets 在每次用户改变数据库时更新，而对记录集的改变会反映在基础表格中。SnapShots 是同一数据的静态显示，其中包含生成 SnapShots 时请求的记录（基础表格中的改变不会在 SnapShots 中反映出来），自然也不会更改 SnapShots。

DynaSets 是最灵活、最强大的记录集。虽然 Table 类型记录集需要大量间接成本。SnapShots 是最缺少灵活性的记录集，但所要的间接成本最少。如果不需要更新数据库，只要浏览记录，可以用 SnapShots 这种类型。

SnapShots 类型还有一个变型正向型 SnapShots，这种类型 SnapShots 的限制更多，只能正向移动，但速度更快。正向型 SnapShots 可以用于要扫描多个记录并顺序处理（进行数值计算，复制所选记录到另一个表格中，等等）数据库记录的情况。这个记录集不提供反向方法，所以间接成本少。

Tables 型记录组可用于调用数据库表格。Tables 比其他记录集类型的处理速度都快，可以保持表格与数据库中的数据同步，也可用于更新数据库。但 Table 只限于一个表格。此外，通过 Tables 型记录集访问表格时，可以利用 Tables 的索引值进行快速查找。

7.2　Visual Basic.NET 开发数据库初步，数据绑定显示

7.2.1　数据集（DataSets）的概念

这一小节主要说明一些关于数据库访问的相关的必要概念，如果读者已经有这些概念

了，则可以跳过这一节。

1. 基本概念

数据集是一种离线了的缓存存储数据，它的结构和数据库一样，具有表格、行、列的一种层次模型，另外还包括了为数据集所定义的数据间的约束和关联关系。

用户可以通过图 7.3 所示的这些.NET 框架的命名空间（NameSpace）来创建和操作数据集。

用户可以通过一些诸如属性（properties）、集合（collections）这些标准的构成来了解 Dataset 这个概念。如：

（1）数据集（DataSet）包括数据表格的“Tables”这个集合以及 relation 的“Relations”集合。

（2）DataTable 类包括了数据表格 row 的“Rows”集合，数据 columns 的“Column”集合，以及数据 relation 的“ChildRelations”和“ParentRelations”集合。

（3）DatataRow 类包括“RowState”属性，这些值是用来显示数据表格首次从数据库被加载后是否被修改过，这个属性的值可以为：“Deleted”、“Modified”、“New”、以及“Unchanged”。

图 7.3　DataSet Namespace

2. 定型（Typed）和未定型（Untyped）的数据集

数据集有定型的和未定型之分，定型的数据集是基本的 DataSet 类的一个子类，并且含有图表文件（.xsd 文件），它用来描述数据集所拥有的表格的结构。这些图表文件，包括了表的名字和列名、列所代表的数据的类型信息，以及数据间的约束关系。而一个位定型的数据集则没有这些图表的描述。

在程序中用户可以使用任意两种类型的数据集，然而，定型的数据集可以使得用户对数据的操作更加明了，并且可以减少一些不必要的错误，定型的数据集可以生成一些对象模型，这些模型的第一层次的类（first-class）就是数据集所包含的表和列，假设用户正在对一个定型的数据集进行操作，则可以用如下的语法来指向一个列。

```
' 指向表 titles 第一行的 title_id 列
s = dsPubs1.titles(0).title_id
```

但是如果用户是在操作一个未定型的数据集的话，就需要这样编写代码了：

```
' 指向表 titles 第一行的 title_id 列
s = dsPubs1.Tables("titles")(0).Columns("title_id")
```

使用定型类不但可以使代码编制起来更加容易，而且，这种定型类的语法还为用户提供了检查代码正确与否的功能，减少了在指向数据集成员值时代码的错误率。

3. 关系表（Related Tables）和关系对象（Related Objects）

如果数据集中包含了多张表，这些表之间可能具有相关联的关系，然而，数据集不像数据库，它并没有关于关联关系的相关信息，所以当用户在处理关系表的时候，可以创建一些关联（relations）来描述这些在数据集中各个表之间的关联关系。关联关系可以通过一些代码，人为地从父行（parent rows）到相关的子行（child rows）。或者从子行再返回到父行。

例如，下面这个关于学校课程设置的例子，有一个关于教师的一个表格（如表 7.1 所示）。

表 7.1　教师信息

InstructorID	Name	Hire Date
i444	张三	09/01/1992
i777	李四	09/01/1998
i123	王二	09/01/1983

其中数据集中可能还包括另一张关于课程信息的表，这张表包括了教师的 ID，并把它作为一个外约束键（foreign key），如表 7.2 所示。

超星阅览器提示
使用本复制品
请尊重相关知识产权!

表 7.2　课程信息

Department	Course Number	Name	InstructorID
自然科学系	999	生物	i444
自然科学系	101	物理	i777
数学系	101	数值计算	i777

因为每个老师可能不只教一门课，所以在课程描述表和教员描述表之间就存在一种一对多的关系。举一个例子，假设教师 i777（李四）教 2 门课。则可以用数据的关联（data relations）把一个数据表的某个特指的行指向另外一个数据表的一个列，这样就可以在这两个表之间双向地进行查询导航，例如，用户可以从一行描述张三教师的行，浏览到描述它教的课这一行，反之，也可以从描述课程 Science999（生物）的行浏览到描述教这门课的教师张三的这一行。

4. 更新数据集和数据存储

当用户改变数据集中的数据时，这些改变将要被重新写回到数据库中，类“DataRow”是用来对单独的记录进行操作的，其中它包括属性“RowState”，它的值是用来表示自从数据从数据库中第一次被调用以后，行是怎样变化的，该属性的值可以为：“Deleted”、“Modified”、“New”以及“Unchanged”。

为了把这些数据集的变化写回到数据库中，用户可以调用方法“Update”，这个方法将会访问“RowState”的属性值，并且决定怎样对数据库进行一系列的操作，如 add，edit，delete，将都会被运行。

7.2.2　在 Windows 窗体中显示单个表的数据

上一小节说明了数据集的概念，这一小节以及以下的介绍将通过几个例子，来对一些关于数据库访问的基本操作进行一些说明和阐述。

在程序开发中，在一个窗体中显示数据是一种最常见的应用，下面这个例子将会创建一个简单的窗体，在窗体上的 data grid 控件中显示一个表的内容，尽管结果不是很复杂，但是通过这个例子，读者可以了解到很多通过窗体访问数据库的一些基本的过程。

为了完成这个例子，你需要访问一个服务器上的数据库，例如 SQL Server 的较简单的数据库。

整个过程分为以下几个步骤：

（1）创建一个窗体。

（2）创建和配置一个绑定在窗体上的数据集（dataset），其中包括创建一个查询语句，通过这个查询语言用户可以从数据库中得到数据集。

（3）把 DataGrid 控件加入到窗体中，并且把控件绑定到相应数据。

1. 创建一个工程和窗体

第一步为创建一个 Windows 窗体。

（1）点击“File”菜单，点击“New”，选择“Project”；

（2）在工程类型选择类型面版中，选择“Visual Basic Project”，并且在运行程序类型选择的时候，选择“Windows Application”；

（3）在工程名称中输入工程的名称，例如“Walkthrough_Simple1”，当做完这些以后，点击“OK”，则 Visual Studio 建立了一个新的工程，并且在 Windows Form Designer 中显示了一个新的窗体。

2. 创建并配置一个数据集（DataSet）

正如那些在 Visual Studio 中常用的访问数据库的例子一样，用户将面临着处理数据集的问题，在本例子中用户只需进行以下几步就可以让 Visual Studio 自行生成一个数据集，关于数据集的一些概念，读者可以参阅 7.2.1 小节。

3. 配置一个数据库连接（Data Connection）和数据集命令（Dataset Command）

首先，需要创建一个数据集命令（Dataset Command），数据集命令（Dataset Command）包括了一些 SQL（标准结构化查询语言）命令，通过运行这些命令，用户可以从数据中产生需要的数据集，这里先定义一个对数据库的连接。

从工具条（Toolbox）的数据（Data）页选项中，把一个“ADODataSetCommand”对象加入到窗体中，如图 7.4 所示。

注意：用户也可以用“SQLDataSetCommand”，但这个对象主要是用来处理 SQL Server 7.0 或更高版本的一些事务的，建议使用“ADODataSetCommand”，因为该对象更具有一般性，可以提供对任何 OLE DB-compatible 数据源的 ADO.NET 访问。

在添加 ADODataSetCommand 对象的同时，便开始了 DataSetCommand 配置向导，它将会帮助用户创建对数据库的连接和对数据库的访问命令。

在向导中，进行如下的一些步骤：

（1）进如向导界面，如图 7.5 所示，点击“Next”进入配置向导。

图 7.4　给窗体添加 ADODataSetCommand 对象　　图 7.5　数据集命令配置向导开始界面

（2）进入第二步，选择或创建对 SQL Server Pubs 数据库的指向，如图 7.6 所示。

这里假设已有一台在局域网中的计算机中装载了 SQL Server，该计算机的名称为“Server”并且已经有名为“SERVER.pubs.dbo”数据连接选择，该选择表示指向 server 计算机 pubs 数据库的数据连接。选择该连接，并点击“Next”进行下一步的配置，如果本机并没有符合本例要求的数据连接的话，则点击“New Connection...”按钮，进行数据库连接的创建，如图 7.7 所示。

超星版权
使用本复制品
请尊重相关知识产权!

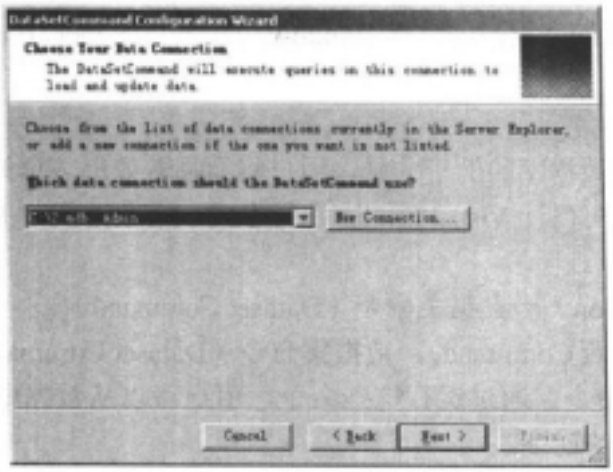


图 7.6　选择数据库连接

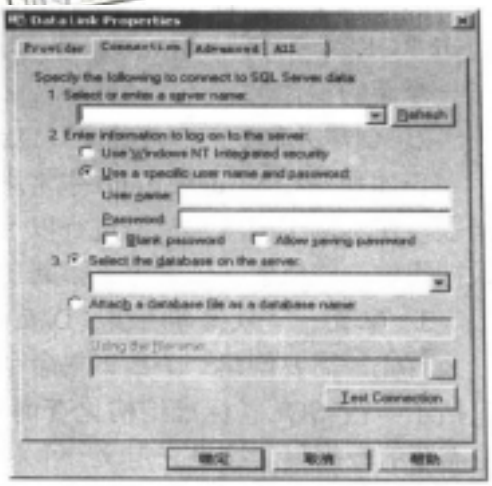


图 7.7　配置数据连接

首先选择服务器的名称，如图 7.8 所示，在这里选择“SERVER”。

在登录用户名和密码中，键入相应的选项，如图 7.9 所示。

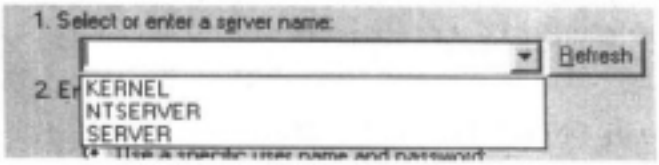


图 7.8　选择服务器

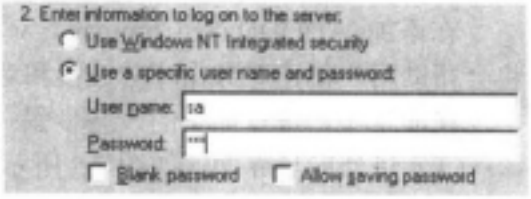


图 7.9　登录数据库服务器

在选择数据库的下拉框中，选择“pubs”数据库文件，如图 7.10 所示。

点击“确认”按钮，弹出 SQL Server 登录对话框，如图 7.11 所示，键入登录密码，完成对数据连接的建立。

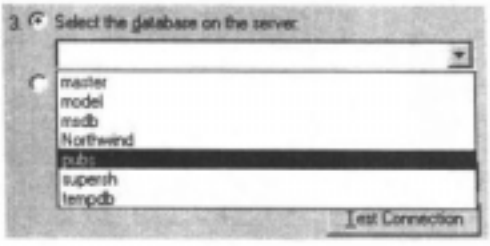


图 7.10　选择服务器

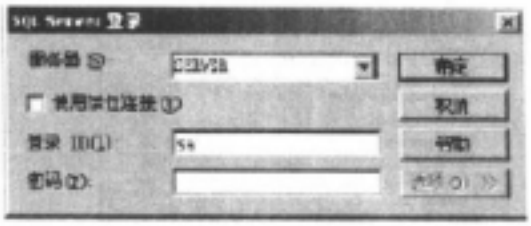


图 7.11　SQL Server 登录

（3）进入第三步，选择 SQL 语句访问数据方式，如图 7.12 所示，这里有三种组织 SQL 语句的方式，第一种，Use SQL Statement ,SQL 语句普通表述；第二种，Use newly created stored procedures，新建一个 SQL 存储过程；第三种，Use existing stored procedures，使用已有的存储过程。这里选择第一种 SQL 查询语句组织方式。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

（4）进入第四步写入如下 SQL 语句：

select au_id,au_lname,au_fname,city,state,phone from authors

当然用户也可以使用“SQL builder”进行编辑，产生以上 SQL 语句。

当向导完成，点击“finish”，如图 7.13 所示，则将会在工程里含有一个名为“ADOConnection1”的连接，它包含了刚才所写入的关于怎样访问一个指定的数据库的信息。用户也将拥有一个名为“ADODataSetCommand1”的数据集命令，它定义了一条查询数据库的相关表和列的 SQL 查询语句。

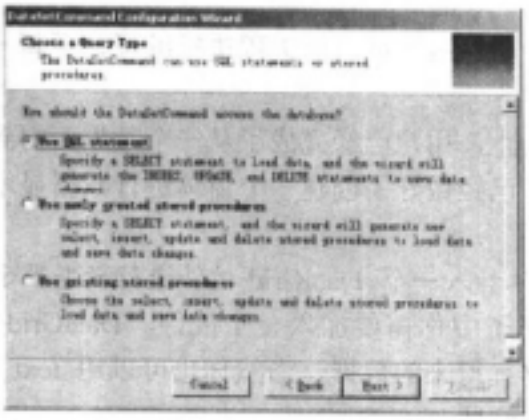


图 7.12　确认 SQL 语句访问数据方式

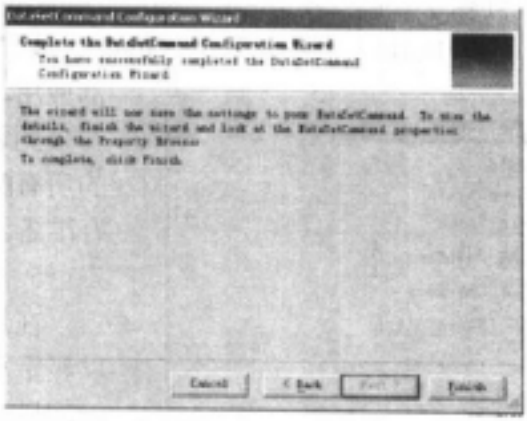


图 7.13　完成向导

4. 建立数据集（DataSet）和访问方法

在建立完 ADOConnection 和 ADODataSetCommand 以后，用户就可以通过 Visual Studio 自行生成数据集了。Visual Studio 可以根据用户设置的 SQL 语句数据集命令，自动生成数据集。如果用户需要更新数据库的话，将需要通过数据集写入信息，并且返回到数据库，为了简化过程，Visual Studio 可以生成一些方法和函数来进行这些操作。

下面通过实际的操作来实现数据集的自动生成。

（1）在“DataClass”菜单中，点击“Generate DataSet”，如 7.14 所示。

注意：如果此时没有看见“DataClass”的话，点击窗体，也就是说必须是在前窗体处于焦点状态时，才能出现“DataClass”菜单。

弹出数据集命名对话框如图 7.15 所示。

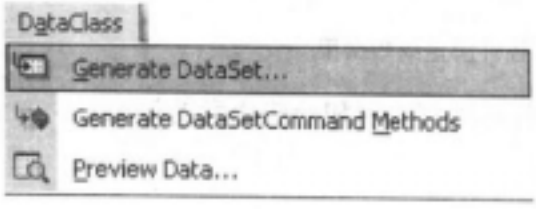


图 7.14　生成数据集

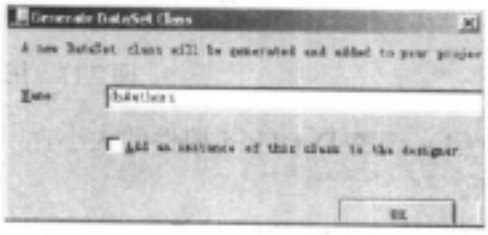


图 7.15　键入数据集类的名称

把该数据集的名称命名为：dsAuthors，并且选择“Add an instance of the class to the designer”，点击“OK”，SQL Server 身份验证完毕后，完成配置。

Visual Studio 生成定型的数据集类（dsAuthors），并且用户可以看见有一个新的图表文件（dsAuthors.xsd）被用来描述这个类，在 Solution 浏览器中可以看见这个文件。

注意：在 Solution 浏览器中点击“Show All Files”可以看见所有的.vb 或.cs 文件，都包含了定义新数据集类的代码。

最后，Visual Studio 在这个窗体中加入了一个新数据集类（dsAuthors）。

（2）在“DataClass”菜单中，选择点击“Generate DataSetCommand Methods”给数据集类生成一些方法，设计环境切换到代码编辑环境，并且显示编辑环境生成的方法，对于这些生成方法，不要做任何的修改。

注意：Visual Studio 生成的方法可以对数据库进行读取和修改，但本例只需把数据集赋值就可以了。

在“Bulid”菜单，选择“build”或“build project”对当前工程进行编译，更新一些内部应用。

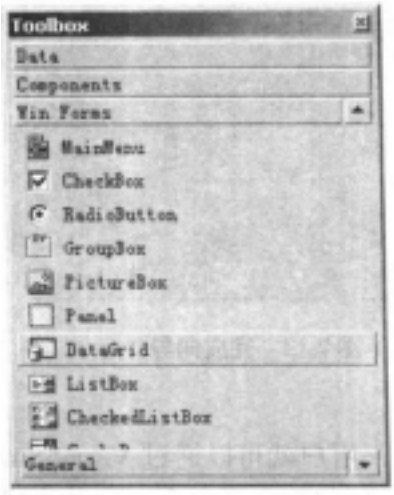


图 7.16　添加 DataGrid 控件

到现在为止，用户已经完成了从一个数据库得到信息的所有配置，已经做好了创建一个窗体进行数据显示的所有准备。

5. 给窗体加入一个 DataGrid 控件进行数据显示

在这个例子中你需要加入一个控件“DataGrid”，用以显示从数据集得来的数据，当然你也可以用 text boxes 一次显示一条信息。

注意：关于用 text boxes 绑定显示数据的方法，下面的几个例子将会对其进行阐述。

DataGrid 必须绑定在数据集中才能用于显示数据。

（1）如图 7.16 所示给窗体加入 DataGrid 控件。

（2）点击【F4】显示属性窗口。

（3）在“DataSource”属性选项中，选择“dsAuthors1.authors”作为数据源（注意在本例中不要单独选择 dsAuthors1）。做完这一步后，就已经把 DataGrid 绑定到数据集的 authors 表中了，如图 7.17 所示。

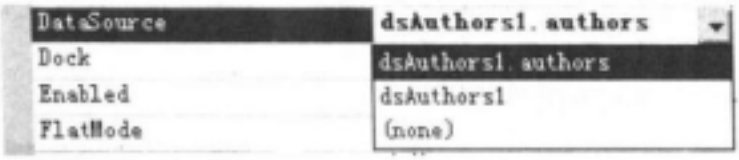


图 7.17　把 DataGrid 控件绑定到数据集

（4）调整 DataGrid 控件的大小，以便可以看见所有的列和一些作者的记录行，如图 7.18 所示。

6. 填充 DataGrid 控件

虽然 DataGrid 控件已经绑定到所建立的数据集中了，但是 DataGrid 控件还不会自动把

数据集的信息填充到 DataGrid 中，用户必须自己通过在代码中调用一个以前生成的方法。关于这些概念，可以参考上一小节关于数据集的一些概念。

（1）给窗体加入按钮控件。

（2）给按钮控件命名为“btnLoad”并且把它的“text”属性改为“Load”。

（3）双击按钮，建立双击事件方法。

（4）在该方法中，首先，刷新建立的数据集，而后，调用窗体的“FillDataSet”属性，把数据传入到绑定的 DataGrid 控件中。

其代码编制如下：

```
Protected Sub btnLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        '进行数据库登录的身份验证
        Me.ADOConnection1.UserID = "sa"
        Me.ADOConnection1.Password = "aaa"
        '清空数据集
        Me.dsAuthors1.Clear()
        '给 DataGrid 控件加载数据

End Sub
```

超星阅览器提示
使用本复制品
请尊重相关知识产权!

7. 检验

现在用户可以检验窗体是否可以在 DataGrid 控件中加载数据了。

（1）按【F5】运行窗体。

（2）当窗体显示完以后，点击“Load”按钮，验证数据是否被加载在 DataGrid 控件当中了，如图 7.19 所示。

图 7.18 调整 DataGrid 控件的大小

图 7.19 验证最后结果

8. 下面几步

相信通过上面这个例子读者学会了访问数据库的最基本的一些方法，还可以做以下的一些改动：

（1）改变 DataGrid 的一些格式，如颜色，字体等。

（2）可以在窗体的 New 方法中加入数据的加载行为，这样就可以不用手动地去点击“Load”按钮了，可以把按钮“Load”去掉。

7.2.3 在窗体上显示带参数的数据查询

有时，用户需要显示一些带参数查询的数据，例如，只想知道一个特定的顾客的订单，或者一个特定的作者写的所有的书等。在这一小节里，用户可以在一个窗体中输入一些信息，然后程序按照用户输入的这些信息进行查询，也就是说，程序进行的查询是按照这些参数进行的，查询的结果，也只是用户所需要的。

带有参数的查询可以对数据查询进行处理，得到的是指定的、过滤的和有序的一些记录，这使得程序工作得更加有效率。相比之下，如果用户通过局域网，把整个数据表传到客户机上，而且通过程序或人工去判断哪些信息是需要的话，程序就会变得非常慢和效率低下。

在这一小节里，将要建立一个 Windows 窗体从标准 SQL Server Pubs 数据库中读取 authors 的信息，用户将可以输入 U.S.的州名代码（如 CA 代表 California），则程序把在该州生活的作者的信息加载在数据集中，其界面如图 7.20 所示。

为了完成这个例子，需要访问一个服务器上的数据库，例如 SQL Server 的 Pubs 数据库或较简单的数据库。整个过程分为以下几个步骤：

（1）创建一个窗体。

（2）创建和配置一个绑定在窗体上的数据集（dataset），其中包括创建一个查询语句，通过这个查询语言，用户可以从数据库把数据加载到数据集。

（3）把控件加入到窗体中去。

图 7.20 用户界面

（4）编制代码，用新参数刷新数据集。

（5）编制代码，在 authors 表中进行浏览。

1. 建立工程和窗体

第一步创建一个 Windows 窗体。

（1）点击“File”菜单，点击“New”，选择“Project”。

（2）在工程类型选择类型面版中，选择“Visual Basic Project”，并且在模板选择的时候，选择“Windows Application”。

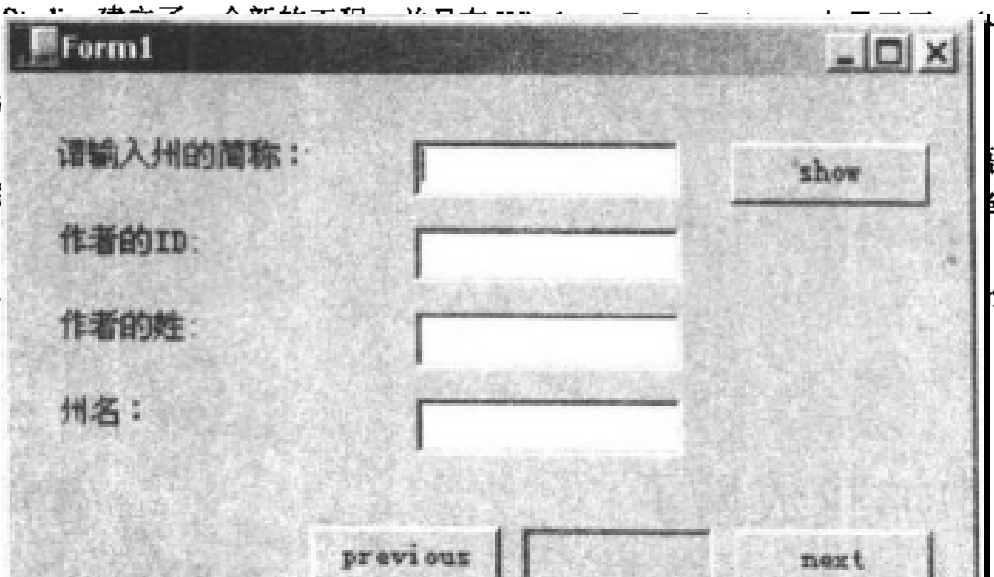
（3）在工程名称中输入工程的名称，例如“Walkthrough_Parameters”，当做完这些以后，点击“OK”。

则 Visual [illegible] 新的窗体。

2. 建立[illegible]

在窗体中[illegible]句的 Where 子[illegible]地加载数据。

首先，从[illegible]nd”对象加入到窗



超星阅览器提醒您：使用本复制品请尊重相关知识产权！

注意：用户也可以用“SQLDataSetCommand”，但这个对象主要是用来处理 SQL Server7.0 或更高版本的一些事务的，建议读者使用“ADODataSetCommand”，因为该对象更具有一般性，可以提供对任何 OLE DB-compatible 数据源的 ADO.NET 访问。

在配置向导中，进行如下操作：

在第二个面板中，选择或创建对 SQL Server Pubs 数据库的指向，如图 7.6 所示。

在第三个面板中，选择第一种 SQL 查询语句组织方式。

在第四个面板中，建立如下 SQL 语句：

```
select au_id,au_lname,state
from authors
WHERE (state =?)
```

其中问号（?）代表一个参数。

注意：当然用户也可以使用“SQL builder”进行编辑产生以上 SQL 语句。

当向导结束后，将会看到在工程里，含有一个名为“ADOConnection1”的连接，它包含了刚才所写入的关于怎样访问一个指定的数据库的信息。用户也将拥有一个名为“ADODataSetCommand1”的数据集命令，它定义了一条用户想查询数据库的相关表和列的 SQL 查询语句。下一步就是要生成数据集类和方法。

3. 建立数据集（DataSet）和访问方法

在建立完 ADOConnection 和 ADODataSetCommand 以后，用户就可以通过 Visual Studio 自行生成数据集了。Visual Studio 可以根据用户设置的 SQL 语句数据集命令，自动生成数据集。如果用户需要更新数据库的话，将需要通过数据集写入信息，并且返回到数据库，为了简化过程，Visual Studio 可以生成一些方法和函数来进行这些操作。

下面通过实际的操作来实现数据集的自动生成。

（1）在“DataClass”菜单中，点击“Generate DataSet”，如图 7.14 所示。

注意：如果未出现“DataClass”的话，点击窗体，也就是说必须是在前窗体处于焦点状态时，才能出现“DataClass”菜单。

弹出数据集命名对话框如图 7.15 所示。

把该数据集的名称命名为：dsAuthors，并且选择“Add an instance of the class to the designer”，点击“OK”，SQL Server 身份验证完毕后，完成配置。

Visual Studio 生成定型的数据集类（dsAuthors），并且用户可以看见有一个新的图表文件（dsAuthors.xsd）被用来描述这个类，在 Solution 浏览器中可以看见这个文件。

注意：在 Solution 浏览器中点击“Show All Files”可以看见所有的.vb 或.cs 文件，它们都包含了定义新数据集类的代码。

最后，Visual Studio 在这个窗体中加入了一个新数据集类（dsAuthors）。

（2）在“DataClass”菜单中，选择点击“Generate DataSetCommand Methods”给数据集类生成一些方法，设计环境切换到代码编辑环境，并且显示编辑环境生成的方法，对于这些生成方法，不要做任何的修改。

注意：Visual Studio 生成的方法可以对数据库进行读取和修改，但在本例中，只需给数据集赋值就可以了。

在“Bulid”菜单，选择“build”或“build project”对当前工程进行编译，更新一些内部应用。

超星提醒您：
使用本文献，
请尊重相关知识产权！

4. 在窗体中加入控件以显示数据

在本例中，窗体将需要用户输入州名的代码缩写，由该参数产生查询，并显示要查询得到的数据。

在窗体中加入控件：

在窗体中加入以下控件，详细情况如表 7.3 所示。

表 7.3　控件设置

控　件	目　的	名　称
TextBox	输入参数，代表州的简称	txtStateParameter
Button	运行查询，加载数据	btnShow
TextBox	显示作者的 ID	txtAuthorID
TextBox	显示作者的姓	txtAuthorLName
TextBox	显示作者所在的州名	txtAuthorState

5. 加入使数据集加载数据的代码

每次用户点击“show”按钮的话，在“DataSetCommand”中的查询命令就会重新执行，并且返回到数据集中，在本例中，查询语句需要一个参数，这个参数就是用户在“txtStateParameters”text box 输入的值。

可以通过刚才产生的一个方法进行在代码中执行查询。

双击“show”按钮，建立 Click 事件的方法，加入处理以下事件的代码：

（1）调用数据集的 Clear 方法。如果用户不先清空数据集的话，则通过查询返回的数据将会和原有的数据集进行叠加。

（2）调用窗体的 FillDataSet 方法。

其添加的代码为：

```
Protected Sub btnshow_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        '设置连接数据集的用户
        Me.ADOConnection1.UserID = "sa"
        '设置连接数据集的密码
        Me.ADOConnection1.Password = "aaa"
        '清除数据集的内容
        Me.dsauthors1.Clear()
        '在数据集中加载数据
        Me.FillDataSet(dsauthors1, txtStatePara.Text)
        End Sub
```

6. 把 Text Boxes 绑定到数据集

这三个 text boxes 是用来显示在数据集数据表 Auyhors 中的单条记录的信息。为了让它能够自动显示数据，必须要把 texts boxes 绑定在表的列上，窗体的绑定机制确认用户可以在浏览下一条信息的时候，自动更新当前的显示。

（1）选定第一个表示作者的 text box，按【F4】打开属性窗口。

（2）打开属性窗口的“Binding”节点，在“text”属性下框中选择“dsAuthors1.authors.au_id”，如图 7.21 所示。

（3）重复第二步，在其他的两个“text”的绑定属性里分别选择“dsAuthors1.authors.au_lname”和“dsAuthors1.authors.state”。

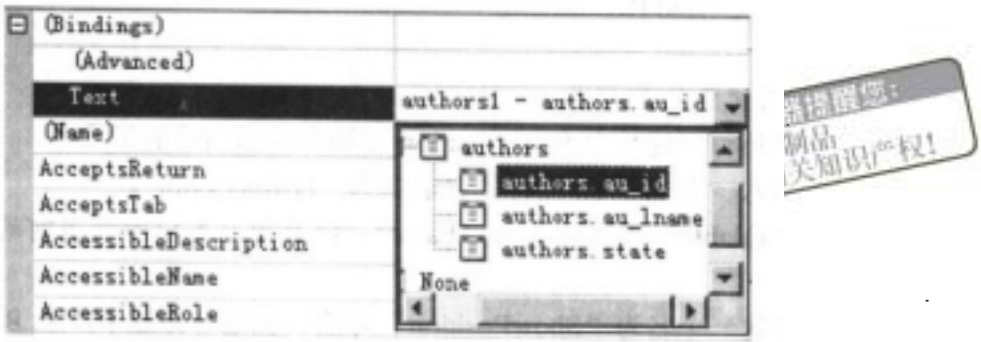


图7.21 把控件的text绑定到数据表的某列

7. 加入导航控件

最后，用户需要为窗体加上数据导航控件。在本例中，需要加上“Previous”和“Next”按钮（也可以加上“Last”和“First”按钮，为了简单起见，在本例中，只加上“Previous”和“Next”按钮），用户也需要加上一个text box以显示当前记录的位置。

在Windows窗体中，通过一个对象（如text boxs）来进行数据的浏览称之为绑定机制管理对象（“BindingManager”），该对象就像在窗体和窗体引用的所有数据表之间媒介，为每一张数据表提供相关的信息（在本例中窗体只引用了一个数据表，但是一个窗体实际上是可以引用很多数据源的）。

当前的记录在数据表的位置实际上可以在“BindingManager”对象的“Position”属性里得到。为了在数据集里导航，用户可以改变这个属性的值。为了得知在数据表里有多少记录，用户可以访问该对象的“Count”属性。

按照下列表中的名称，给窗体加入控件（如表7.4所示）。

表7.4 控件设置

控　件	名　称
Button	BtnNext
Button	BtnPrevious

（1）为“Previous”按钮创建Click事件，改变BindingManager对象的“Position”属性，具体代码如下：

```
Protected Sub btnPreview_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        If Me.BindingManager(dsAuthors1, "authors").Position > 0 Then
            Me.BindingManager(dsAuthors1, "authors").Position -= 1
        End if
    End Sub
```

（2）为“Next”按钮创建Click事件，同时避免超出记录的数目，可以使用以下的代码：

```
Protected Sub btnnext_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim iCnt As Integer
        iCnt = Me.BindingManager(dsAuthors1, "authors").Count - 1
```

```
            If Me.BindingManager(dsAuthors1, "authors").Position < iCnt Then
                Me.BindingManager(dsAuthors1, "authors").Position += 1
            End If
    End Sub
```

8. 显示当前记录位置

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

最后，可以自行创建一个 ShowPosition 方法来显示当前的记录。

在窗体中加入一个 textbox 控件，并取名为“txtRecCount”。

注意：把“Enable”属性设置为“False”，以至用户可以看见当前的数据位置，但是不能改变它的值。

在窗体中创建一个名为“ShowPosition”方法。在这个方法中，从 BindingManager 对象得到当前的位置，并把它显示在 text box 中，具体的代码如下：

```
    Private Sub showPosition()
            Dim iCnt As Integer
            Dim iPos As Integer
            iCnt = Me.BindingManager(dsAuthors1, "authors").Count
            iPos = Me.BindingManager(dsAuthors1, "authors").Position + 1
            txtRecCount.Text = iPos & " of " & iCnt
    End Sub
```

在其他可能会改变数据位置的地方，加入 ShowPositon 的方法，在本例中，在如下的位置加入这些代码：

（1）在“Show”按钮被点击以后，在 FillDataSet 方法被调用以后；

（2）在点击完“Previous”和“Next”按钮以后，加入如下代码：

```
    Protected Sub btnnext_Click(ByVal sender As Object, ByVal e As System.EventArgs)
            Dim iCnt As Integer
            iCnt = Me.BindingManager(dsAuthors1, "authors").Count - 1
            If Me.BindingManager(dsAuthors1, "authors").Position < iCnt Then
                Me.BindingManager(dsAuthors1, "authors").Position += 1
            End If
        ShowPosition()
        End Sub

    Protected Sub btnPreview_Click(ByVal sender As Object, _
                                   ByVal e As System.EventArgs)
        If Me.BindingManager(dsAuthors1, "authors").Position > 0 Then
            Me.BindingManager(dsAuthors1, "authors").Position -= 1
        End If
        ShowPosition()
    End Sub
```

9. 测试工程

可以通过测试工程判断你输入的参数有没有效。

（1）按【F5】运行窗体。

（2）窗体显示完以后，在州名的地方输入：“CA”，并且点击“Show”，则第一个在 California 作者的信息将会显示在窗体上，如图 7.22 所示。

（3）点击“Previous”和“Next”按钮，在作者之间浏览。

（4）在 text boxs 输入一个新的值，例如“UT”，点击“Show”再一次验证其正确性。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

图 7.22　验证程序更功能

10. 错误处理

如果用户输入了一个错误的州名的代号的话，系统将会产生一个错误的信号，这个错误的信息是由“FillDataSet”方法产生的，原因是因为查询没有返回任何数据记录。

可以在窗体的代码中定义“Try……Catch”模块来处理类似的出错处理。

（1）首先，在窗体的代码中找到 FillDataSet 方法。注意，在该方法中已经有一个含有代码的 Try 模块用来处理数据库的连接，加载数据，关闭对数据库的连接。

（2）对数据库连接开始断开以后在 Try 模块后添加一个 Catch 模块，在 Catch 模块中加入错误处理信息，然后断开数据库连接。

下面是 FillDataSet 方法的完全代码：

```
Public Sub FillDataSet(ByVal dataSet As System.Data.DataSet, _
                       ByVal Parameter1 As String)
    'Open connections for ADO and SQLServer DataSetCommands
    Try
        Me.ADOConnection1.Open()
        'Turn off constraint checking in the DataSet while retrieving data from
        datasource
        dataSet.EnforceConstraints = False
        'Fill DataSet with data from all DataSetCommands
        Me.ADODataSetCommand1FillDataSet(dataSet, Parameter1)
        'Turn on constraint checking in the DataSet
        dataSet.EnforceConstraints = True
        'Close connections for ADO and SQLServer DataSetCommands
        Me.ADOConnection1.Close()
        Catch e As System.Exception
        MessageBox.Show("Invalid state code!", "State Code Error")
        Me.ADOConnection1.Close()
    End Try
End Sub
```

11. 下一步

当完成这个例子后，相信读者已经会做简单的数据绑定数据操作了，而且也可以作如

下的一些尝试：

（1）可以用一个下拉框来代替现在用户手工输入州名的简称，可以用另外一个 SQL 表达式“SELECT UNIQUE STATE FROM authors”来加载数据集，并把结果绑定在下拉框的 list 中。

（2）把最后的结果绑定在一个 grid 控件中，这样的话，就可以一次看见所有的结果了。

7.2.4 用 Data Form Wizard 生成数据绑定控件

超星阅览器提醒您：
使用本复制品
需用关知识产权!

一个在 Visual Studio 中简单地，快捷地生成数据绑定的窗体的方法就是利用 Data Form Wizard。该向导可以非常简单地在数据集中引用数据表和列，并且在窗体中生成数据绑定的控件用来显示数据。

Data Form Wizard 既可以在 Windows 窗体，也可以在 Web 窗体中运行，Data Form Wizard 在这两种窗体中运行的过程很接近，在本例中用户将在 Windows 窗体中运行 Data Form Wizard。

在本例中，用户需要进行以下的准备：

访问一个服务器上的数据库，例如 SQL Server 的 Pubs 数据库或较简单的数据库。本例的步骤如下：

（1）建立一个 Windows 窗口；

（2）建立和配置一个绑定在窗体上的数据集（dataset），其中包括创建一个查询语句，通过这个查询语言用户可以从数据库把数据加载到数据集；

（3）运行 Data Form Wizard，给窗体产生一个数据绑定的控件。

1. 新建一个工程和窗体

第一步创建一个 Windows 窗体。

（1）点击“File”菜单，点击“New”，选择“Project”。

（2）在工程类型选择类型面版中，选择“Visual Basic Project”，并且在模板选择的时候，选择“Windows Application”。

（3）在工程名称中输入工程的名称，例如“Walkthrough_DFW1”，当做完这些以后，点击“OK”。

则 Visual Studio 建立了一个新的工程，并且在 Windows Form Designer 中显示了一个新的窗体。

2. 新建和配置一个数据集

该步骤已在上几节有详细的描述，这里就不再重复了，详细请见 7.2.1。

3. 配置数据连接（Data connection）和数据加载命令（Data command）

这个步骤将会建立一个数据连接来引用数据库，建立一个 SQL 表述数据集命令来加载数据集。

首先，从工具条（Toolbox）的数据（Data）页选项中，把一个“ADODataSetCommand”对象加入到窗体中，如图 7.4 所示。

注意：用户也可以用“SQLDataSetCommand”，但这个对象主要是用来处理 SQL Server 7.0 或更高版本的一些事务的，建议使用“ADODataSetCommand”，因为该对象更具有一般性，可以提供对任何 OLE DB-compatible 数据源的 ADO.NET 访问。

在配置向导中，有如下操作：

在第二个面板中，选择或创建对 SQL Server Pubs 数据库的指向，如图 7.6 所示。

在第三个面板中，选择第一种 SQL 查询语句组织方式。

在第四个面板中，建立如下 SQL 语句：

```
select au_id,au_lname,au_fname
from authors
WHERE (state =?)
```

其中问号（?）就是代表一个参数。

注意：用户也可以使用“SQL builder”进行编辑产生以上 SQL 语句。

当向导结束后，将会看到工程里，含有一个名为“ADOConnection1”的连接，它包含了刚才所写入的关于怎样访问一个指定的数据库的信息。用户也将拥有一个名为“ADODataSetCommand1”的数据集命令，它定义了一条用户想查询数据库的相关表和列的 SQL 查询语句。下一步就是要生成数据集类和方法。

4. 建立数据集（DataSet）和访问方法

在用户建立完 ADOConnection 和 ADODataSetCommand 以后，就可以通过 Visual Studio 自行生成数据集了。Visual Studio 可以根据设置的 SQL 语句数据集命令，自动生成数据集。如果需要更新数据库的话，将需要通过数据集写入信息，并且返回到数据库。为了简化过程，Visual Studio 可以生成一些方法和函数来进行这些操作。

下面通过实际的操作来实现数据集的自动生成。

在“DataClass”菜单中，点击“Generate DataSet”，如图 7.14 所示。

注意：如果未出现“DataClass”的话，点击窗体，也就是说必须是当前窗体处于焦点状态时，才能出现“DataClass”菜单。

弹出数据集命名对话框如图 7.15 所示。

把该数据集的名称命名为：dsAuthors，并且选择“Add an instance of the class to the designer”，点击“OK”，SQL Server 身份验证完毕后，完成配置。

Visual Studio 生成定型的数据集类（dsAuthors），并且用户可以看见有一个新的图表文件（dsAuthors.xsd）被用来描述这个类，在 Solution 浏览器中可以看见这个文件。

注意：在 Solution 浏览器中点击“Show All Files”可以看见所有的.vb 或.cs 文件，它们都包含了定义新数据集类的代码。

最后，Visual Studio 在这个窗体中加入了一个新数据集类（dsAuthors）。

在“DataClass”菜单中，选择点击“Generate DataSetCommand Methods”给数据集类生成一些方法，设计环境切换到代码编辑环境，并且显示编辑环境生成的方法，对于这些生成方法，不要做任何的修改。

注意：Visual Studio 生成的方法可以对数据库进行读取和修改，本例只需把数据集赋值就可以了。

在“Bulid”菜单，选择“build”或“build project”对当前工程进行编译，更新一些内部应用。

做完这些以后，就做好了在窗体中显示数据的准备了。

5. 利用 Data Form Wizard 创建数据绑定窗体

一旦用户已经为窗体做好了可以引用的数据集以后，有很多方法可以把这些相关的数

据显示出来，本例将利用 Data Form Wizard，该向导可在窗体中生成绑定在数据集中的控件。在控件生成以后，你就可以向编辑其他的控件一样编辑他们了。

Data Form Wizard 给出了两种显示数据的方法：

（1）在 grid 中显示，它允许用户一次把很多行数据显示出来。

（2）在单独的控件中显示，它使得用户一次对一条记录进行操作。向导还能生成导航控件，用户可以使用这个导航控件从一条记录浏览到另一条记录，而且，向导还能生成“Add”、“Update”和“Delete”按钮。

6. 在窗体中生成控件

本例将生成一个 Grid 控件来一次显示所有的数据。

超星阅览器提醒您：
使用本复制品
须遵守相关知识产权！

（1）从工具栏的 data 项里，点击“DataFormWizard”，并放在窗体里，则“Data Form Wizard”运行，如图 7.23 所示。

（2）在第二步中，选择刚才生成的数据集，如图 7.24 所示。

图 7.23　启动 Data Form Wizard

图 7.24　选择数据集

（3）在第三步中，在选项“What method do you want to use toload your data store?”中，选择“FilDataSet”，这个例子不选择“Include Update Method”，如图 7.25 所示。

（4）在第四步中，确认用户需要使用的表的所有列都被选中了，如图 7.26 所示。

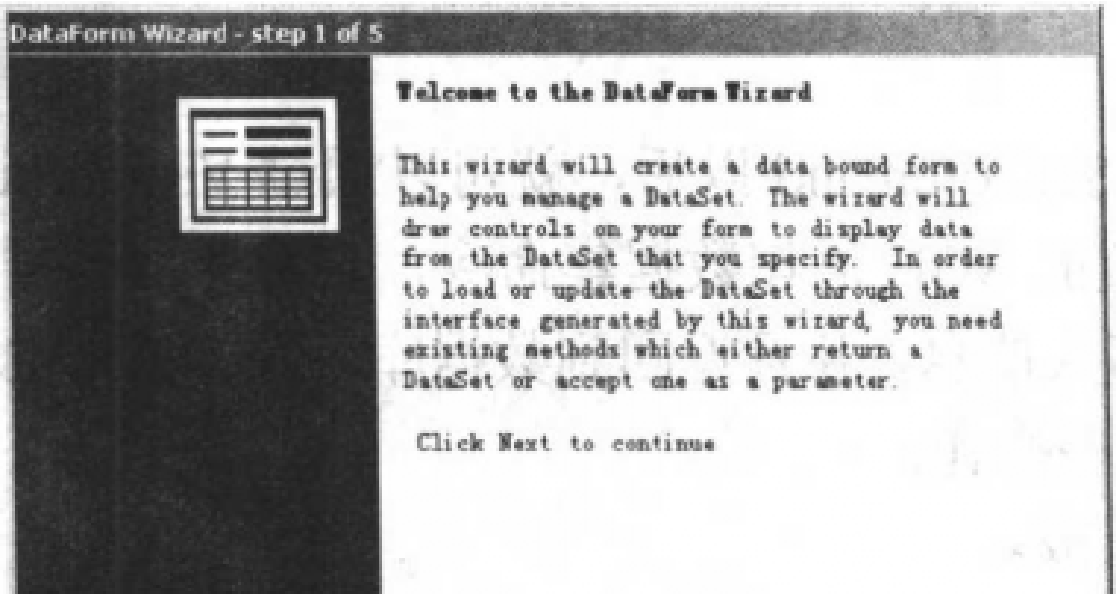


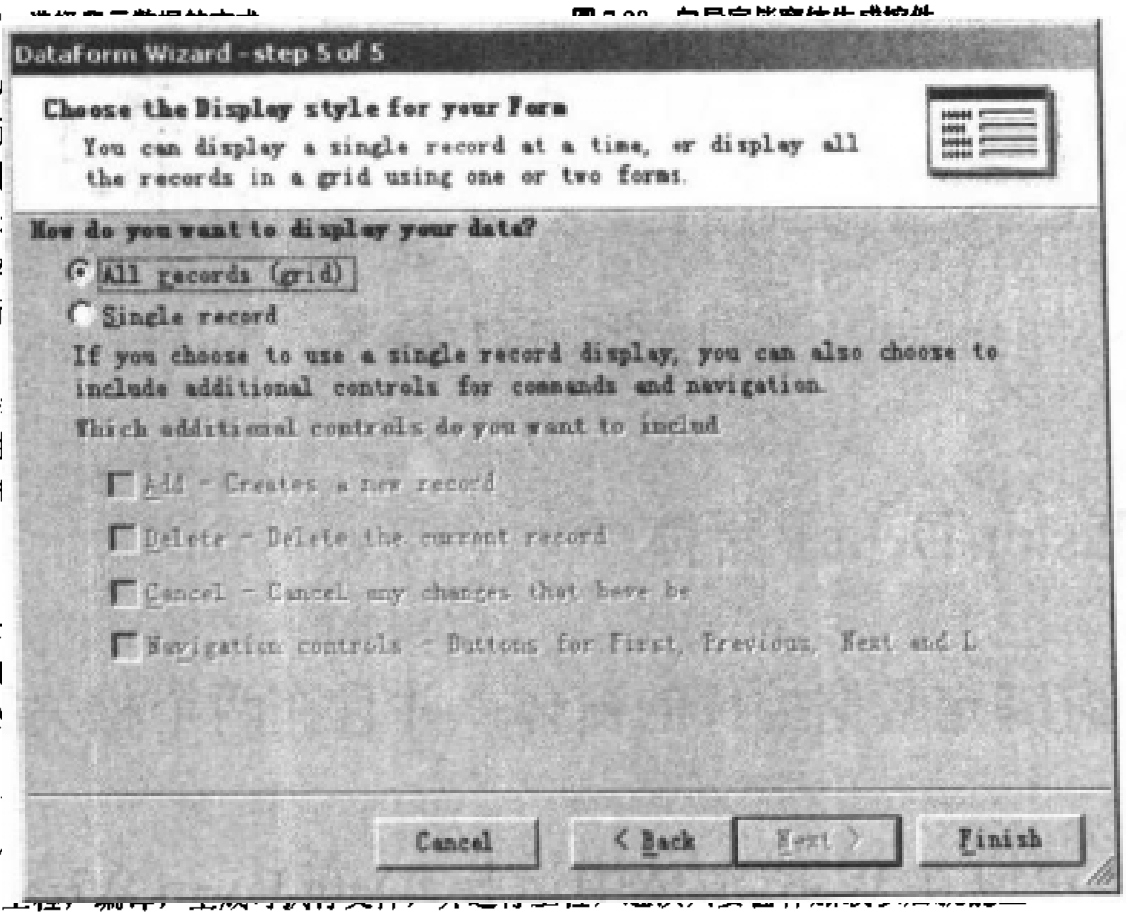
图 7

（5）在第五步中，在选项“How do you want to display your data ?”中选择“All records（grid）”，如图 7.27 所示。

（6）点击“Finish”则向导在窗体中生成了两个控件，一个“Load”按钮和一个“grid”控件，如图 7.28 所示。

超星阅览器提醒您：使用本复制品请尊重相关知识产权！

图 7.27



7. 运行生
当向导运行
要运行窗体
（1）按【F
（2）当窗体
绑定的控件显示
8. 改变生
用 Data Fo
以自己对向导生
如果可以
事件中去。
（1）双击
（2）在 btr
Private Sul
N
End Sub
（3）找到
（4）在窗
（5）保存
即显示数据了，如图 7.30 所示。

使用本复制品
请尊重相关知识产权!

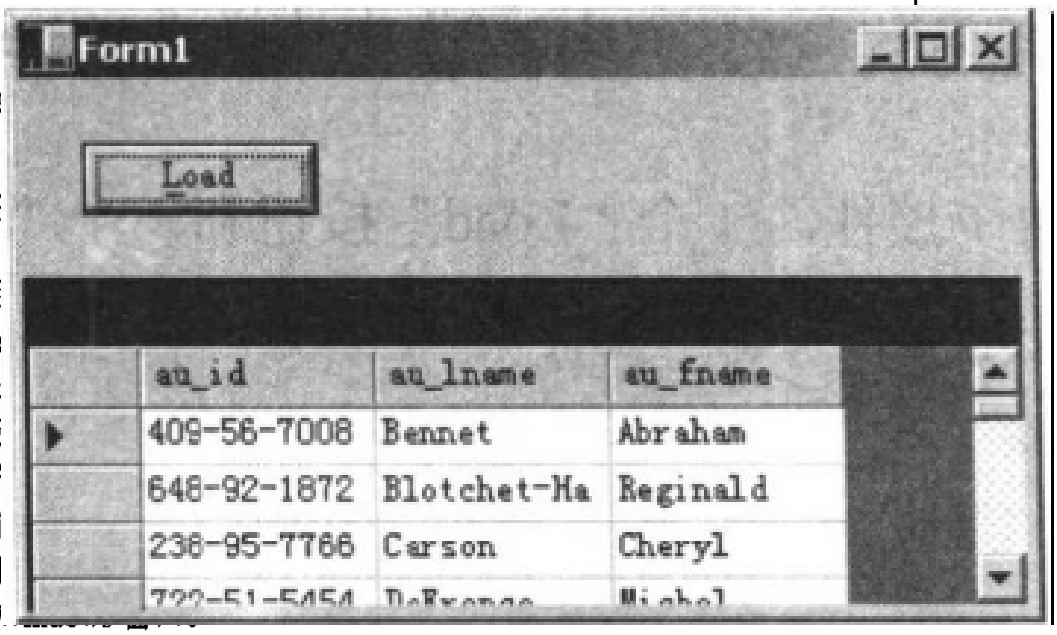


图 7.2

7.2.5　利用 Data F

本例主要是使用
表之间生成一个 Mas
Data Form Wiza
在这两种窗体中运行
在本例中，用户
的 Pubs 数据库或较
本例的步骤如下
（1）建立一个

（2）创建和配置一个绑定在窗体上的数据集（dataset），其中包括创建一个查询语句，通过这个查询语言用户可以从数据库把数据加载到数据集。

（3）在数据集的两个表之间建立一种关联（relationship）。

（4）运行 Data Form Wizard 在窗体上生成一个绑定数据的控件。

1. 创建工程和窗体

第一步创建一个 Windows 窗体。

（1）点击“File”菜单，点击“New”，选择“Project”。

（2）在工程类型选择类型面版中，选择“Visual Basic Project”，并且在模板选择的时候，选择“Windows Application”。

（3）在工程名称中输入工程的名称，例如“Walkthrough_DFW1”，当做完这些以后，点击“OK”，则 Visual Studio 建立了一个新的工程，并且在 Windows Form Designer 中显示了一个新的窗体。

2. 创建和配置一个数据集

正如在 Visual Studio 中大多数访问数据库的应用程序一样，这里将会涉及到数据集，在本例中，数据集中将会包含两个表 —— Publishers 表和 Titles 表，这两个表都是从标准的 SQL Server 中的 Pubs 数据库中得来的，这两个表有一种隐含的关联，pub_id 在 Publishers 表中是主键，而在 Titles 表中是外键。在下面的窗体的控件中，需要把这种隐含的关系变得非常明确。

3. 配置数据连接（Data connection）和数据加载命令（Data command）

在这个步骤中，将会建立两个独立的数据集命令，每一个命令将会在数据集的不同表

中加载数据。

（1）首先，从工具条（Toolbox）的数据（Data）页选项中，把一个“ADODataSetCommand”对象加入到窗体中，如图 7.4 所示。

注意：用户也可以用“SQLDataSetCommand”，但这个对象主要是用来处理 SQL Server 7.0 或更高版本的一些事务的，建议使用“ADODataSetCommand”，因为该对象更具有一般性，可以提供对任何 OLE DB-compatible 数据源的 ADO.NET 访问。

在配置向导中，有如下操作：

在第二个面板中，选择或创建对 SQL Server Pubs 数据库的指向，如图 7.6 所示。

在第三个面板中，选择第一种 SQL 查询语句组织方式。

在第四个面板中，建立如下 SQL 语句：

```
select pub_id,pub_name,city
from publishers
```

注意：用户也可以使用“SQL builder”进行编辑以产生以上 SQL 语句。

当向导结束后，所建的工程里含有一个名为“ADOConnection1”的连接，它包含了刚才所写入的关于怎样访问一个指定的数据库的信息。用户也将拥有一个名为“ADODataSetCommand1”的数据集命令，它定义了一条查询数据库的相关表和列的 SQL 查询语句。

（2）在窗体中加入第二个“ADODataSetCommand”对象，则“DataSetCommand Configuration”向导又重新运行。

（3）重复前面几步，不过在描述访问数据库的 SQL 命令的时候，写入如下语句：

```
select title_id,title,pub_id
from titles
```

注意：必须包括 pub_id 列。

当向导完成以后，用户将会看见“ADOConnection2”和“ADODataSetCommand2”被加入到你的窗体里了。

4. 创建数据集

现在用户可以创建据集了，因为这里含有两个数据集命令，所以数据集将含有两个表，当生成数据集的时候将会含有一个标准的数据集的类，并且具有相关的 XSD 图表文件。

（1）在“DataClass”菜单中，点击“Generate DataSet”，如图 7.14 所示。

注意：如果未出现“DataClass”的话，点击窗体，也就是说必须是当前窗体处于焦点状态时，才能出现“DataClass”菜单。

弹出数据集命名对话框如图 7.15 所示。

（2）把该数据集的名称命名为：dsAuthors，并且选择“Add an instance of the class to the designer”，点击“OK”，SQL Server 身份验证完毕后，完成配置。

Visual Studio 生成定型的数据集类（dsAuthors），并且可以看见有一个新的图表文件（dsAuthors.xsd）被用来描述这个类，在 Solution 浏览器中可以看见这个文件。

注意：在 Solution 浏览器中点击“Show All Files”可以看见所有的.vb 或.cs 文件，均包含了定义新数据集类的代码。

最后，Visual Studio 在这个窗体中加入了一个新数据集类（dsAuthors）。

超星阅览器提醒您：
使用本文献请尊重相关知识产权！

5. 在数据集的表中建立相关关系

现在数据集含有两个表，并且具有一个“一对多”的关系。然而，数据集只是一个被动的容器（passive container），它不是一个真正的数据库，它不能利用那些隐含的数据表之间的关联。所以，用户必须显性地表示出那些表之间的关联，必须创建一个关联对象。

注意：用户并不是创建了一条在两个有关联的表之间的查询语句，而是分别在两个独立的表建立了两条分别独立的查询。这使得用户可以分别在两个表中，进行导航、关联、更新，这比在根据一条关联关系连接的两个表之间查询数据，更有控制性。

（1）在 Solution Explorer 中，双击刚才创建数据集的图表文件（dsPublishersTitles.xsd）。如图 7.31 所示，在 XML 编辑环境打开图表视图，可以在数据集中看见这两个表。

（2）在 XML Desiger 选项的工具栏中，把一个“Relation”对象拖到 Titles 表（子表）中，则 Relation Editor 打开两个表的缺省的值，如图 7.32 所示。

图 7.31　dsPublishersTitles.xsd 图表视图

图 7.32　Relation Editor

（3）在 Relation Editor 中点击“OK”按钮。

在 Schema Designer 中，在两个表之间出现了一个表示相关性的图标，如图 7.33 所示。如果用户需要改变它们之间相关的关系，则右击该相关关系，并且选择 Edit Relation，如图 7.34 所示。

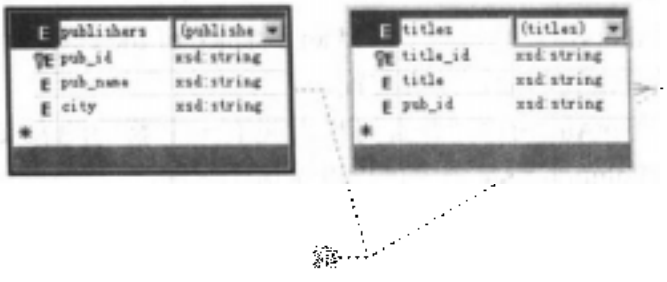


图 7.33　出现相关性图标

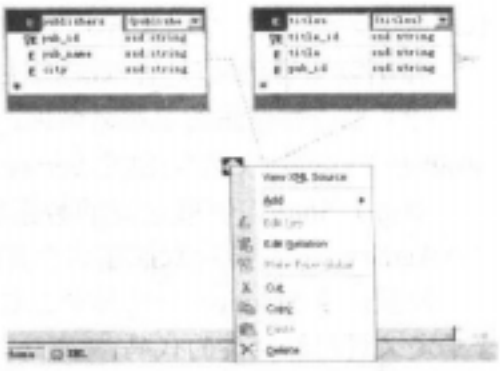


图 7.34　编辑相关性关系

使用本复制品
请尊重相关知识产权!

6. 为数据集创建一个访问的方法

在完成对表之间的相关关系的配置之后，也就完成了对数据集的配置，则最后一步就是为数据集的访问方法生成访问的方法，通过该方法，用户可以给数据集加载和更新数据。

在“DataClass”菜单中，选择点击“Generate DataSetCommand Methods”给数据集类生成一些方法，设计环境切换到代码编辑环境，并且显示编辑环境生成的方法，对于这些生成方法，用户不许做任何的修改。

注意：Visual Studio 生成的方法可以对数据库进行读取和修改，但在本例中，只需对数据集赋值就可以了。

在“Bulid”菜单，选择“build”或“build project”对当前工程进行编译，更新一些内部应用。

到现在为止，已经完成了要从一个数据库得到信息的所有配置，已经做好了创建一个窗体进行数据显示的所有准备。

7. 使用 Data Form Wizard 创建一个 Master/Detail 窗体

现在已经有一个含有两个表以及在它们之间的联系的一个记录集。用户可以使用 Data Form Wizard 给窗体生成一个 master/detail 形式的空间来显示数据。例如，当用户在浏览一条新的 master 记录的时候，控件可以在另一个控件上，显示详细的信息。

在工具栏的 data 项里，把 Data Form Wizard 拖到窗体里。则向导开始运行，在第二个窗口，选择刚才生成的数据集，则向导能够判断数据集含有两个表。

在第三个窗口，在选项“What method do you want to use to load your data store?”中选择“FillDataSet”，在本例中，不要选择“Include Update Method”选项，如图 7.35 所示。

在第四个窗口中，确认两个表的数据均被选中（缺省情况下，只有 master 表中的列被选中），如图 7.36 所示。

图 7.35　设置访问数据的方法

图 7.36　选择两个表的列

在第五个窗口中，在选项“How do you want to display your data ?”中选择“All Records (grid)”，如图 7.37 所示。

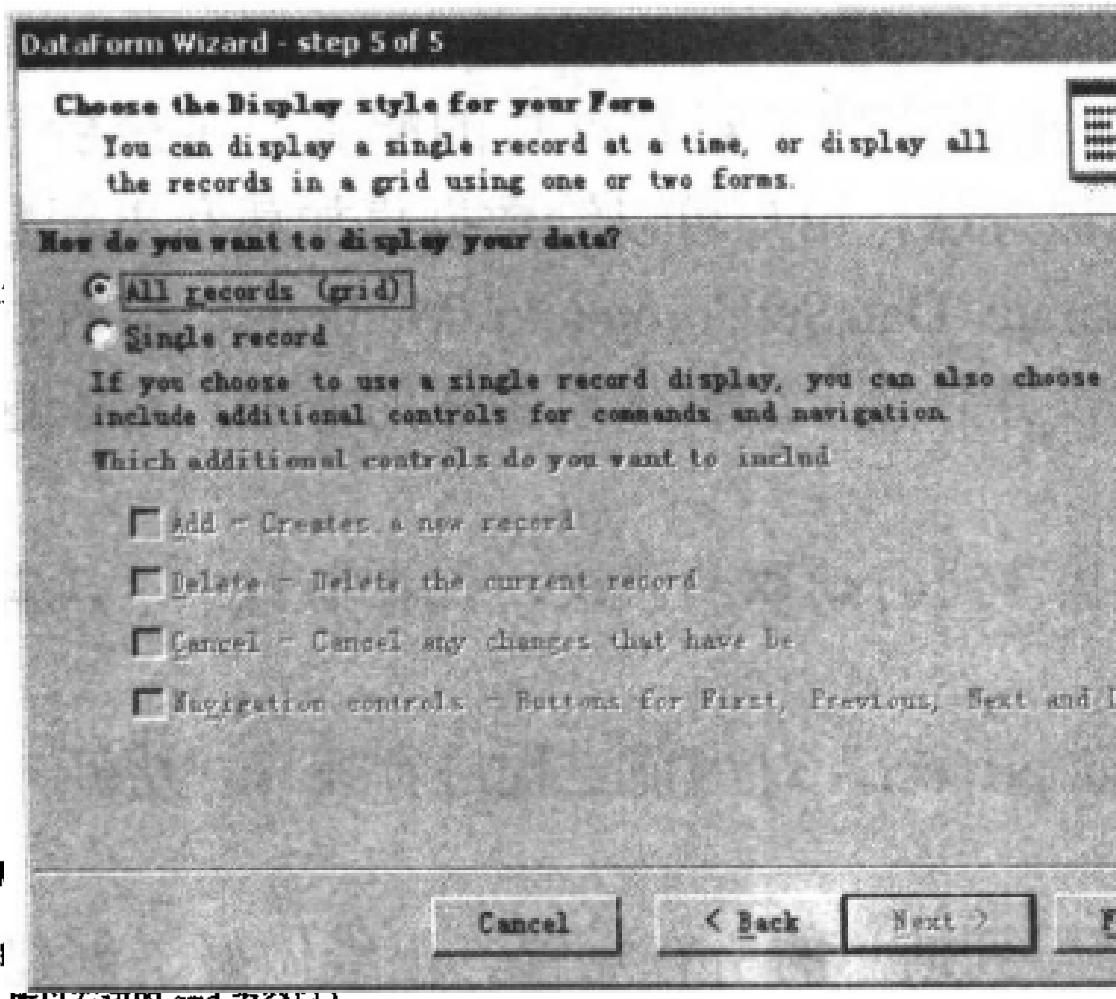


图 7.

图 7.39　运行结果

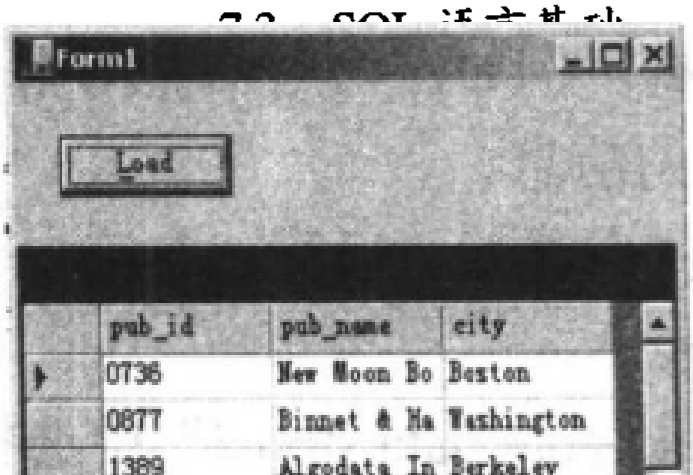
（4）加载 publis

publisher 没有 titles，所以右边的 grid 为空白）。

注意：要在代码中加入对 SQL Server 数据库的访问用户和访问密码，在本例中，已经略去了其中的过程。

8. 下一步

下面可进行的改动就是把 publisher 的信息用单独的控件表示，而 titles 的信息用 grid 表示。



7.3 SQL 语言基础

7.3.1 SQL

SQL 是结　　写，它是使用关系模型的

数据库应用语言，由 IBM 在 70 年代开发出来，作为 IBM 关系数据库原型 System R 的原型关系语言，实现了关系数据库中的信息检索。

80 年代初，美国国家标准局（ANSI）开始着手制定 SQL 标准，最早的 ANSI 标准于 1986 年完成，它也被叫做 SQL-86。标准的出台使 SQL 作为标准的关系数据库语言的地位得到加强。SQL 标准几经修改和完善，目前新的 SQL 标准是 1992 年制定的 SQL-92，它的全名是“International Standard ISO/IEC 9075:1992，Database Language SQL”。

请尊重知识产权!

7.3.2　数据类型

数据类型是指列、存储过程参数和局部变量数据特征，它决定了数据的存储格式，代表着不同的信息类型。

数据类型可分为系统数据类型（如表 7.5 所示）和用户定义数据类型两种。

表 7.5　系统数据类型

数据类型名称	定 义 标 识
二进制型	Binary[(n)]，varbinary[(n)]
字符型	Char[(n)]，varchar[(n)]
日期时间型	Datetime，smalldatetime
整数型	Int,smallint，tinyint
精确数值型	Decimal，numeric
近似数值型	Float，real
货币型	Money，smallmoney
位型	Bit
时间戳型	Timestamp
文本型	Text
图像型	Image

1. 二进制型

二进制数据类型代表二进制数，其最大长度为 255 字节，二进制数可以是 0~9 和 A~F 或 a~f 的字符组成，二进制数以 0x 标识开头，其中每两个字符为一组，构成一个字节，如 0xf5。

分类：定长二进制数据类型（用 binary[(n)]声明）；

变长二进制数据类型（用 varbinary[(n)]声明）。

[例]：DECLARE　@var1　binary(10)

SELECT　@var1 '0x10 ff　aa'

DECLARE　@var2 varbinary(64)

注意：在数据类型定义语句或变量声明时，如果不指定 n 值，则其长度为 1，然而在 CONVERT 转换函数中如果不指定 n 值时，其缺省长度则为 30。

定长和变长的区别：对于定长的数据类型，不管其输入值的实际长度是多少，它都占用 n 字节。如果所赋数据长度超过 n 时，超出部分将被截断。而变长二进制列的存储长度是可变的，它为字符串的实际长度，但最大不得超过 n 所规定的值。当输入数据长度超过 n 时，余下部分将被截断。

2. 字符类型

在输入字符数据时应将数据引在单引号内。字符类型也有定长（char[(n)]）和变长（varchar[(n)]）二种。

超星阅览器
使用本复制品
请尊重相关知识产权!

3. 日期时间类型

分类：datetime 和 smalldatetime

datetime 类型的数据长度为 8 字节，smalldatetime 类型的数据长度为 4 字节。

4. 整数型

分类：int、smallint、tinyint。int 长度为 4 个字节，smallint 长度为 2 个字节，tinyint 长度为 1 字节。

5. 精确数值型

分类：decimal[(p[,s])]和 numeric[(p[,s])]{p 指精度，s 指小数位}

6. 近似数值型

分类：float[(n)]、real。float 数据类型可以存储正、负浮点数，n 确定二进制数精度，它可以为 1 到 15。当其精度为 1 到 7 之间时，等同于 real 数据类型。

7. 货币型

分类：money（占 8 个字节）、smallmoney（占 4 个字节）

8. 位数据类型

位数据类型用 bit 关键字声明，其数据有两种取值：0 和 1。在输入 0 以外的其他值时系统均将它们当作 1 看待。

9. 时间戳数据类型

用 timestamp 声明。

10. 文本和图像类型

文本（text）和图像（image）类型是两种可变长度的数据类型。向 text 列中插入数据时，应将数据引在单引号内。向 image 列中插入数据时，应在数据前加 0x 引导符。

标识符：

标识符的命名规则如下：

（1）标识符长度为 1 到 30 字符。

（2）标识符的第一个字符必须为字母或_、@、#符号。其中@和#符号具有特殊意义：当标识符开头为@时，表示它是一局部变量；标识符首字符为#时，表示是一临时数据库对象，对于表或存储过程，名称开头含一个#号时表示为局部临时对象，含两个##号时表示为全局临时对象。

（3）标识符中第一个字符后面的字符可以为字母、数字或#、$、_符号。

（4）缺省情况下，标识符内不允许有空格，也不允许使用关键字等作为标识符，但可以使用引号来定义特殊标识符。

7.3.3 运算符

1. 算术运算符

+、-、*、/、%（取模）：

2. 位运算运

&（与）、|（或）、^（异或）、~（求反）；

3. 比较运算符

<、>、=、>=、<=、<>；

4. 字符串运算符

字符串运算符(+)实现字符串之间和连接操作。如'abc'+'123'结果为：'abc123'。

超星阅览器提醒您：
使用本复制品
须尊重知识产权！

5. 运算符的优先级

- 括号()；
- 取反运算：~；
- 乘、除、求模运算：*、/、%；
- 异或运算：^；
- 与运算：|；
- NOT 连接；
- AND 连接；
- OR 连接。

7.3.4　变量

声明格式：DECLARE　@<变量名>　<变量类型>[，@<变量名>　<变量类型>...]；

赋值：用 SELECT 语句。

[例]：DECLARE　@var1　int, @var2　money

SELECT　@var1=100,　@var2=$27.95

SELECT　@var1,　@var2

7.3.5　流控制语句

1. IF...ELSE...语句

格式：

```
IF 布尔表达式
    {SQL 语句或语句块}
  [ELSE
    {SQL 语句或语句块}]
```

2. ENGIN.....END 语句

将多条 SQL 语句封装起来，构成一个语句块。

格式：

```
BEGIN
    {SQL 语句或语句块}
END
```

3. WHILE、BREAK、CONTINUE 语句

[例]：求 1 到 10 之间的奇数据和

```
DECLARE @i  smallint, @sum smallint
```

```
SELECT  @i=0,@sum=0
WHILE @i>=0
BEGIN
   SELECT  @i=@i+1
   IF @i>10
    BREAK
   IF (@i  %  2)=0
    CONTINUE
   ELSE
    SELECT  @sum = @sum + @I
  END
SELECT  @sum
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

4. WAITFOR 语句

格式：WAITFOR {DELAY 'time'|TIME 'time'}

time:datetime

[例]：WAITFOR TIME '10:00'　设置在 10:00 执行一次命令；

WAITFOR DELAY '1:00' 设置在一小时后执行一次命令。

5. RETURN 语句

使程序从一个查询或存储过程中无条件地返回，其后面的语句不再执行。

RETURN([整数表达式])

CASE 表达式

（1）简单 CASE 表达式

```
CASE 表达式
WHEN 表达式 1 THEN 表达式 2
   [WHEN 表达式 3 THEN 表达式 4[……]]
   [ELSE 表达式 N]
END
```

[例]：

```
select name=convert(varchar(15),au_lname),
  contract=case contract
        when 0 then 'invalid contract.'
        when 1 then 'valid contract.'
  End
from authors
```

（2）搜索型 CASE 表达式

```
  CASE 表达式
     WHEN 布尔表达式 1 THEN 表达式 1
     [WHEN 布尔表达式 2 THEN 表达式 2[……]]
     [ELSE 表达式 N]
  END
```

[例]：

```
select name=convert(varchar(15),au_lname),
       contract=case contract
          when cintract=0 then 'invalid contract.'
          when contract<>0 then 'valid contract.'
End
from authors
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

（3）CASE 关系函数

CASE 关系函数有以下 3 种形式：

- COALESCE（表达式 1，表达式 2）
- COALESCE（表达式 1，表达式 2，…，表达式 N）
- NULLIF（表达式 1，表达式 2）

7.4　SQL 语句函数

7.4.1　日期函数

日期函数如表 7.6 所示。

表 7.6　日期函数

函　数	参　数	功　能
GETDATE	（）	返回系统当前日期和时间
DATEADD	（datepart，number，date）	返回 datetime 或 smalldatetime 类型数据，其值为 date 值加上 datepart 和 number 参数指定的时间间隔
DATEDIFF	（datepart，date1，date2）	返回 date1 和 date2 间的时间间隔，其单位由 datepart 参数指定
DATENAME	（datepart，date）	返回日期中指定部分对应的字符串
DATEPART	（datepart，date）	返回日期中指定部分对应的整数值

DATENAME 和 DATEPART 函数分别返回日期中指定日期元素对应的字符串和整数值。

7.4.2　字符串函数

字符串函数如表 7.7 所示。

表 7.7　字符串函数

函　数	参　数	功　能
+	(expression,expression)	连接两个或多个字符串、二进制串、列名
ASCII	(char_expr)	返回表达式中最左边一个字符的 ASCII 值
CHAR	(integer_expr)	返回整数所代表的 ASCII 值对应的字符

（续表）

函　数	参　数	功　能
LOWER	(char_expr)	将大写字符转换为小写字符
UPPER	(char_expr)	将小写字符转换为大写字符
LTRIM	(char_expr)	删除字符串开始部分的空格
RTRIM	(char_expr)	删除字符串尾部的空格
RIGHT	(char_expr,integer_expr)	返回 char-expr 字符串中 integer-expr 个字符以后的部分字符串，integer-expr 为负时，返回 NULL
SPACE	(integer_expr)	返回一个由空格组成的字符串，空格等于 integer-expr 值，integer-expr 为负时，返回 NULL
STR	(float_expr[,length[,decima]])	将一个数值型数据转换为字符串，length 为字符串的长度，decimal 为小数点的位数
STUFF	(char_expr1,start,length,char_expr2)	从 char-expr1 字符的 star 个字符位置处删除 length 个字符，然后把 char-expr2 字符串插入到 char-expr1 的 start 处
SUBSTRING	(expression,start,length)	从 expression 的第 start 个字符处返回 length 个字符
REVERSE	(char_expr)	返回 char-expr 的逆序
CHARINDEX	('pattern',expression)	返回指定 pattern 字符串在表达式中的起始位置
DIFFERENCE	(char_expr1,char_expr2)	比较两个字符串，返回它们的相似性，返回值为 1~4
PATINDEX	('%prattern%',expression)	返回 expression 中首次出现 pattern 的起始位置
REPLICATE	(char_expr,integer_expr)	返回一个由 char_expr 重复 intrger_expr 次组成的字符串
SOUNDEX	(char_expr)	返回一个四代码，说明字符串读音的相似性

7.4.3　数学函数

数学函数如表 7.8 所示。

表 7.8　数学函数

函　数	参　数	功　能
ASIN、ACOS、ATAN	(float_expr)	求 float_expr 的反正弦、反余弦、反正切
ATN2	(float_expr1,float_expr2)	求 float_expr1/float_expr2 的反正切
SIN、COS、TAN、COT	(float_expr)	求 float_expr 的正弦、余弦、正切
DEGREES	(numeric_expr)	将弧度转换为度
RADIANS	(numeric_expr)	将度转换为弧度
EXP	(float_expr)	求 float_expr 的指数值
POWER	(numeric_expr,y)	求 numberic_expr 的 y 次方
SQRT	(float_expr)	求 float_expr 的平方根
LOG	(float_expr)	求 float_expr 的自然对数
LOG10	(float_expr)	求 float_expr 以 10 为底的对数
ABS	(numeric_expr)	求 numeric_expr 的绝对值
CEILING	(numeric_expr)	返回大于等于 numberic_expr 的最小整数
FLOOR	(numeric_expr)	返回小于等于 numberic_expr 的最大整数
RAND	([seed])	返回 0 到 1 之间的随机浮点数，可以使用整数表达式指定其初值
PI	( )	返回常数 3.141492653589793
ROUND	(numeric_expr,length)	将 numberic_expr 小数点后的值四舍五入，保留的小数位数为 length
SIGN	(numeric_expr)	numberic_expr 的值为正数、0 或负数时分别返回 1、0、−1 数值

7.4.4　集合函数

集合函数如表 7.9 所示。

表 7.9　集合函数

函　数	参　数	功　能
COUNT	([ALL|DISTICT]expression)	计算表达式中非空值的数量，可用于数字型列，使用 DISTICT 时删除重复值
COUNT	(*)	计算所有行数，包括空值行，对 COUNT(*)不能使用 DISTINCT 关键字
MIN	(expression)	计算表达式最小值，可用于数字开型、字符型和日期时间型列，但不能用于 bit、text、image 列，MIN 函数忽略表达式中的空值
MAX	(expression)	计算表达式最大值，可用于数字开型、字符型和日期时间型列，但不能用于 bit、text、image 列，MIN 函数忽略表达式中的空值
SUM	([ALL|DISTICT]expression)	计算表达式所有值的和，它忽略表达式中的空值，使用 DISTINCT 关键字时删除表达式中的重复值，它适用于数字型列
AVG	([ALL|DISTICT]expression)	计算表达式的平均值，它忽略空值，使用 DISTINCT 关键字时删除表达式中的重复值，适用于数字型列

7.4.5　文本和图像函数

文本和图像函数如表 7.10 所示。

表 7.10　文本和图像函数

函　数	参　数	功　能
TEXTPTR	(column_name)	以 varbinary 数据类型返回指向 text 或 image 列首页数据的指针，如果还未使用 insert 或 update 语句初始化 text 或 image 列时，返回 null
TEXTVALID	('table_name.column_name',text_ptr)	检查指向 table_name.column_name 的 text_ptr 指针的有效性，如果指针有效则返回 1，否则返回 0

7.4.6　转换函数

CONVERT (datatype[(length)],expression[,style])

参数 expression 转换后的数据类型，它只能为系统数据类型，当 datatype 参数为 char、varchar、binary 或 varbinary 数据类型时，length 说明转换后数据的长度。length 的最大值为 255，缺省长度为 30。

将 datetime 或 smalldatetime 数据类型转换为字符数据时，用 style 参数说明转换后的字符串格式。

7.5　表、视图与索引

7.5.1　表

超星阅览器提醒您：
使用本复制品，
请尊重相关知识产权！

数据表可分为永久表和临时表两种，临时表在用户退出或系统恢复时被自动删除。临时表又分为局部临时表和全局临时表两种，在创建表时，系统根据表名来确定是临时表还是永久表，临时表的表名开头包含两个#。表名的最大长度（包括#在内）为 20 个字符。

1. 建立数据表

使用 CREATE TABLE 语句建立表，其格式为：

```
CREATE TABLE [database.[owner.]]table_name
(
{col_name datatype [NULL | NOT NULL | IDENTITY[(seed,increment)]]
[constraint [constraint [...constraint]]] | [[,] constraint]}
[[,]{next_col_name | next_constraint}...]
)
[ON segment_name]
```

database 指定所建表的存放位置，缺省时为当前数据库。

owner 指定表所有者，缺省时为当前用户。

table_name 是新建表的名称。

col_name 定义表的列名，在一个表中，列名必须惟一，但在同一个数据库的不同表中列名可以相同。

datatype 指定列的数据类型。

IDENTITY 指定该列为 IDENTITY 列，其列值由系统自动插入。每个表中能有一个 IDENTITY 列，该列值不能由用户更新，也不允许空值。IDENTITY 列的数据类型只能为 int、smallint、tinyint、numeric、decimal 等系统数据类型，IDENTITY 列数据类型为 numeric 和 decimal 时，不允许出现小数位。对于 IDENTITY 列，seed 为 IDENTITY 列的基值，increment 为 IDENTITY 列的列值增量。缺省时，seed 和 increment 的值均为 1。

[例]：

```
CREATE TABLE person
(
  person_id INT IDENTITY(1,10),
  name CHAR(8) NOT NULL
)
```

2. 修改表

使用 ALTER TABLE 语句可以修改表结构，为其添加列，或打开、关闭已有约束，增加、删除约束等操作。

ALTER TABLE 语句格式为：

```
ALTER TABLE [database.[owner]]table_name
```

[WITH {CHECK | NOCHECK}]
{{ CHECK | NOCHECK}CONSTRAINT {constraint_name | ALL}
 |
[ADD
{column_definition [column_constraints] | [[,] table_colstraint]}
[,{column_definition [column_constraints] | [[,] table_colstraint]}……]
 |
[DROP CONSTRAINT]
constraint_name [,constraint_name2]……]}

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

ADD 项参数说明向表中增加列或表约束，其中列定义与 CREATE TABLE 语句中的列定义方法相同。

DROP 项说明删除表中现有约束。

[例]：

ALTER TABLE person
ADD
country char(2) NULL

3. 删除表

DROP TABLE 语句的格式为：

DROP TABLE [[database.]owner.]table_name
[,[[database.]owner.]table_name……]

[例]：DROP TABLE person

7.5.2　表数据操作

1. 添加数据

格式：

INSERT [INTO]
{table_name | view_name} {(column_list)}
{
DEFAULT VALUES | values_list | select_statement |
EXECute {procedure_name | @procedure_name_var}
[[procedure_name=] {value | @variable [OUTPUT] | DEFAULT}
[, {[procedure_name=] {value | @variable [OUTPUT] | DEFAULT}]……]

column_list 列出要添加数据的列名。在给表或视图中部分列添加数据时，必须使用该选项说明这部分列名。

DEFAULT VALUES 说明向表中所有列插入其缺省值。对于具有 INDENTITY 属性或 timestamp 数据类型的列，系统将自动插入下一个适当值。对于没有设置缺省值的列，根据它们是否允许空值，将插入 null 或返回一错误消息。

values_list 的格式为：

VALUES (DEFAULT | constant_expression

[,DEFAULT | constant_expression]……)

[例 1]:

INSERT publishers

VALUES('9900','DELPHI','Beijing',null,'china')

超星阅览器提醒您:
使用本复制品
请尊重相关知识产权!

[例 2]:

INSERT publishers(pub_id,pub_name,contry,city)

VALUES('9900','DELPHI','china','Beijing')

[例 3]: 假定有两个表 tab1 和 tab2，它们列的排列顺序分别为：col1，col2，col3 和 col1，col3，col2。这时，可使用下面两种方法来实现数据拷贝：

INSERT tab1(col1,col3,col2)

SELECT * FROM tab2

或

INSERT tab1

SELECT col1,col2,col3 FROM tab2

2. 修改数据

UPDATE {table_name | view_name}

SET

{column_name = {column_list1 | variable_list1 | variable_and_column_list1}

[[,column_name = {column_list2 | variable_list2 | variable_and_column_list2}

……

{,column_name = {column_listn | variable_listn | variable_and_column_listn}]]

| expression]

[WHERE clause]

SET 子句指定被修改的列名及其新值，WHERE 子句说明修改条件，指出表或视图中的哪些行需要修改。

[例 1]: 使用 SET 子句将 discounts 表中所有行的 discounts 值增加 0.1:

UPDATE discounts

SET discount=discount+0.1

[例 2]: 同时修改 discounts 表中折扣类型为 volume discount 的 lowqty 可 doscount 列值:

UPDATE discounts

SET discount=discount+0.5 , lowqty=lowqty+200

WHERE discounttype='volume discount'

3. 删除数据

DELETE 和 TRUNCATE TABLE 语句都可以用来删除表中的数据，DELETE 语句的格式为:

DELETE [FROM] {table_name | view_name}

[WHERE clause]

TRUNCATE TABLE 语句的格式为:

TRUNCATE TABLE [[database.]owner.]table_name

TRUNCATE TABLE 语句删除指定表中的所有数据行，但表结构及其所有索引继续保留。

为该表所定义的约束、规则、缺省和触发器仍然有效。如果所删除表中包含有 IDENTITY 列，则该列将复位到它的原始基值。使用不带 WHERE 子句的 DELETE 语句也可以删除表中所有行，但它不复位 IDENTITY 列。

TRUNCATE TABLE 不能删除一个被其他表通过 FOREIGN KEY 约束所参照的表。

[例 1]:

```
DELETE discounts
TRUNCATE TABLE discounts
```

[例 2]:

```
DELETE titles
WHERE type='business'
```

[例 3]:

```
DELETE titles
WHERE title_id  IN
   (SELECT titleauthor.title_id FROM authors,titles,titleauthor
   WHERE authors.au_id=titleauthor.au_id
      AND titleauthor.title_id=titles.title_id
      AND city='gary')
```

7.5.3　索引

用 CREATE INDEX 语句创建索引，其格式为：

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]  INDEX index_name
ON [[database.]owner]table_name (column_name[,column_name]……)
[WITH
[PAD_INDEX,]
[[,] FILLFACTOR = fillfactor]
[[,] IGNORE_DUP_KEY]
[[,] SORTED_DATA | SORTED_DATA_REORG]
[[,] IGNORE_DUP_ROW | ALLOW_DUP_ROW]]
[ON segment_name]
```

（1）惟一索引：在调用 CREATE INDEX 语句时，使用 UNIQUE 选项创建惟一索引。

（2）复合索引：是对一个表中的两列或多列的组合值进行索引，复合索引的最大列数为 16，这些列必须在同一个表中。

[例]：CREATE INDEX ta ON titleauthor(au_id,title_id)

（3）簇索引（排序）：在调用 CREATE INDEX 语句时，使用 CLUSTERED 选项创建簇索引。

（4）非簇索引：在调用 CREATE INDEX 语句时，使用 NONCLUSTERED 选项创建簇索引。

7.5.4 视图

建立视图

格式：

```
CREATE VIEW [owner.]view_name
[(column_name [, column_name].....)]
[WITH ENCRYPTION]
AS select_statment [WITH CHECK OPTION]
```

7.5.5 查询

用 SELECT 语句实现数据库的查询操作。同时，它还可以使用各种子句对查询结果进行分组统计、合计、排序等操作。SELECT 语句还可将查询结果生成另一个表（临时表或永久表）。

SELECT 语句的语法格式为：

```
SELECT [ALL | DISTINCT] select_list
[INTO [new_table_name]]
[FROM {table_name | view_name} [(optimizer_hints)]
[[,table_name2 | view_name2] [(optimizer_hints)]
    .....
[,table_name16 | view_name16] [(optimizer_hints)]]]
[WHERE clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause]
[COMPUTE clause]
[FOR BROWSE]
```

在 SELECT 语句中，子句可以省略，但在列出时必须按照以上顺序。

简单查询：

简单的 SQL 查询只包括 SELECT 列表、FROM 子句和 WHERE 子句，它们分别说明所查询列、查询操作的表或视图、以及搜索条件等。

[例]：查询 title 表中 1991 年所出版的'business'类图书及其价格。

```
SELECT title,price
FROM titles
WHERE DATEPART(YEAR,pubdate) = 1991 AND type = 'business'
```

（1）Select 列表语句

SELECT 列表语句（select_list）指定所选择的列，它可以为一组列名列表、星号、表达式、变量（包括局间变量和全局变量）等构成。

① 选择所有列

用星号表示选择指定表或视图中的选择所有列。

SELECT *FROM discounts

选择指定列并指定它们的显示次序

在 SELECT 列表语句中指定列名来选择不同的列，各列之间用逗号分隔，显示结果中数据的排列顺序为列名的列表顺序。

SELECT discount,discounttypeFROM discounts

在 SELECT 列表中，还可以对数值列进行算术运算（包括加、减、乘、除、取模等）。其中，加、减、乘、除操作适用于任何数值型列（如 int，smallint，tinyint，decimal，numeric，float，real，money，smallmoney 等），而取模操作不能用于 money 和 smallmoneyu 数据类型列。

对数值列进行运算时，如果列值为空（null），则所执行的所有算术操作的结果仍为空：

```
SELECT discounttype,'lowqty+50'=lowqty+50
FROM discounts
```

在 SELECT 列表中，还可以指定字符串常量或变量改变输出结果的显示格式：

```
DECLARE @var char(12)
SELECT @var='折扣幅度'
SELECT '折扣名称：', discounttype,@var,discount
SELECT discounts
```

② 删除重复行

SELECT 语句中可以使用 ALL | DISTINCT 选项来显示所有行（ALL）或删除重复的行（DISTINCT），缺省时为 ALL。使用 DISTINCT 选项时，对于所有数据重复的 SELECT 列表值只显示一次。

```
SELECT DISTINCT country FROM publishers
```

（2）FROM 子句

FROM 子句指定 SELECT 语句查询及与查询相关的表或视图。在 FROM 子句中最多可指定 16 个表或视图，它们相互之间用逗号分隔，如果这些表或视图属于不同的数据库，可用“数据库.所有者名称.对象”格式限定表或视图对象。

```
SELECT au_id,titles.title_id
   //由于在两个表中都存在“title_id”列，加入“titles.”避免二义性
FROM titles,titleauthor
WHERE titles.title_id = titleauthor.title_id
```

在 FROM 子句中，可为每个表或视图指定一个别名，别名紧跟在对象名称之后，之间用空格分隔，然后可以使用别名引用表中各列。

```
SELECT au_id,t.title_id
FROM titles t,titleauthor ta
WHERE t.title_id = ta.title_id
```

用 WHERE 子句限定搜索条件，SELECT 语句中使用 WHERE 子句指定查询条件。

WHERE 语句中可包含的运算符，如表 7.11 所示。

表 7.11 条件运算符

运算符分类	运 算 符	意 义
比较运算符	>、>=、=、<=、<、<>、!>、!<	大小比较
范围运算符	BETWEEN...AND...	判断表达式值是否在指定范围之内
	NOT BETWEEN...AND...	
列表运算符	IN	判断表达式值是否为列表中的指定项
	NOT IN	
模式匹配符	LIKE	判断列值是否与指定的字符通配格式相符
	NOT LIKE	
空值判断符	IS NULL	判断表达式值是否为空
	NOT IS NULL	
逻辑运算符	AND	用于多条件的逻辑连接
	OR	
	NOT	

[例 1]：范围运算符：查询书价为$10~$30 之内的图书

```
SELECT title,price
FROM titles
WHERE price BETWEEN $10 AND $30
```

[例 2]：列表运算符：列出 publishers 表中社址在德国和法国的出版社

```
SELECT pub_name
FROM publishers
WHERE country IN ('germany','france')
```

（3）模式匹配符

模式匹配符[NOT] LIKE 常用于模糊条件查询，它判断列值是否与指定的字符串格式相匹配，可用于 char、varchar、datetime 和 smalldatetime 数据类型。可使用的通配字符有以下几种：

- 百分号%：可匹配任意类型长度的字符；
- 下划线_：匹配单个任意字符，它常用来限制表达式的字符长度；
- 方括号[]：指定一个字符、字符串或范围，要求所匹配对象为它们中的任一个字符；
- [^]：其取值与[]相同，但它要求所匹配对象为指定字符以外的任一个字符。

[例 3]：查找名称以“publishing”字符串结尾的出版社

```
SELECT pub_name
FROM publishers
WHERE pub_name LIKE '%publishing'
```

[例 4]：查找名称以 A —— F 字符开头的出版社

```
SELECT pub_name
FROM publishers
WHERER pub_name LIKE '[A-F]%'
```

[例 5]：查找名称以 A——F 以外字符开头的出版社

```
SELECT pub_name
FROM publishers
WHERER pub_name LIKE '[^A-F]%'
```

[例 6]：列出名称长度为个 5 个字符，且以'GG'开头的出版社

```
SELECT pub_name
FROM publishers
WHERER pub_name LIKE 'GG___'
```

[例 7]：空值判断符：查找目前仍未定价的图书

```
SELECT title,price
FROM titles
WHERE price IS NULL
```

超星阅览器提醒您：
版权所有

[例 8]：逻辑运算符：列出收价低于$5 或高于$15、且当年销量小于 5000 的图书

```
SELECT price,ytd_sales,title
FROM titles
WHERE (price < $5 OR  price>$15) AND ytd_sales<5000
```

查询结果排序

在 SELECT 语句中，使用 ORDER BY 子句对查询结果按一列或多列进行排序。ORDER BY 子句的语法格式为：

```
ORDER BY {{table_name. | view_name.}colcumn_name
    | select_list_number | expression} [ASC | DESC]
    [.....{{table_name16. | view_name16.}colcumn_name
    | select_list_number | expression} [ASC | DESC]]
```

[例]：列出“business”类的图书标识和价格，结果排序方式为：价格由高到低、title_id 列由低到高。

```
SELECT title_id ,price
FROM titles
WHERE type='business'
ORDER BY price DESC,title_id, ASC
```

7.5.6　统计

在 SELECT 语句中，可以使用集合函数、行集合函数、GROUP BY 子句和 COMPUTE 子句对查询结果进行统计。GROUP BY 子句可与行集合函数或集合函数一起使用，而 COMPUTE 子句只能与行集合函数一起使用。

在 SELECT 语句中，也可以单纯使用集合函数进行统计，这时它将所有符合条件的数据统计到一起，形成一行统计数据，这种统计方法叫做标量统计。

[例]：统计“business”类图书的平均价格：

```
SELECT 'average price'=AVG(price)
FROM titles
WHERE type='business'
```

1. GROUP BY 子句

GROUP BY 子句的语法格式为：

```
GROUP BY [ALL] aggregate_free_expression
```

[,aggregate_free_expression]……

[例]：统计每类图书的平均价格

```
SELECT type,AVG(price) 'average price'
FROM titles
GROUP BY type
```

2. HAVING 子句

在使用 GROUP BY 子句时，还可以用 HAVING 子句为分组统计进一步设置统计条件，HAVING 子句与 GROUP BY 子句的关系和 WHERE 子句与 SELECT 子句的关系类似。

HAVING 子句可以参照选择列表中的任一项，在 HAVING 子句中还可以使用逻辑运算符连接多个条件，最多为 128 个。

[例]：按图书类别分组统计出未指定类型以外的其他图书的平均价格

```
SELECT type,AVG(price) 'average price'
FROM titles
GROUP BY type
HAVING type<>'UNDECIDED'
```

在 HAVING 子句中可以使用集合函数。

[例]：按图书类别分组统计图书的平均价格，但排除只包括一种图书的图书类别

```
SELECT type,AVG(price) 'average price'
FROM titles
GROUP BY type
HAVING COUNT(*)<>1
```

统计结果排序：

[例]：按图书类别统计其平均价格，并按平均价格进行排序

```
SELECT type,AVG(price) 'average price'
FROM titles
GROUP BY type
  HAVING type<>'UNDECIDED'
    ORDER BY AVG(price) DESC
```

3. GROUP BY 子句中的 ALL 选项

在 SELECT 语句中同时使用 GROUP BY 和 WHERE 子句进行分组统计时，在统计列表中只列出符合 WHERE 子句所定条件的数据项。

如果在 GROUP BY 子句中使用 ALL 选项，则在统计列表中将列出被统计表中所有的分组，即使是不符合 WHERE 子句指定条件的分组也将列出，但并不对这些分组进行统计。

[例]：

```
SELECT type,AVG(price) 'average price'
FROM titles
WHERE type<>'business'
GROUP BY ALL type
```

结果列出了 business 类图书，但它并不说明 business 类图书的平均价格为空，而是该类图书不符合统计条件。

使用 GROUP BY 子句应注意的事项：

（1）在 GROUP BY 子句中不能使用集合函数。

（2）必须在 GROUP BY 子句中列出 SELECT 选择列表中所有的非集合项。

[例]：下面语句的 GROUP BY 子句中必须全部列出 type，pub_id 列，缺一不可：

```
SELECT type,pub_id,AVG(price) 'average price'
FROM titles
GROUP BY type,pub_id
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

4. COMPUTE 子句

它不仅显示统计结果，而且还显示统计数据的细节。

COMPUTE 子句的语法格式为：

```
COMPUTE row_aggreggate(column_name)
[,row_aggregate(column_name)……]
[BY column_name[,column_name]……]
```

在 COMPUTE 子句中使用 BY 选项可以对数据进行分组统计。

在使用 COMPUTE 子句时，必须遵守以下规则：

（1）在集合函数中不能使用 DISTINCT 关键字。

（2）COMPUTE BY 子句必须与 ORDER BY 子句同时联合使用，并且 COMPUTE BY 子句中 BY 后的列名列表必须与 ORDER BY 子句中的相同，或为其子集，且二者从左到右的排列顺序必须一致。

（3）COMPUTE 子句中不使用 BY 选项时，统计出来的为合计值。

7.5.7　利用查询结果创建新表

SELECT 语句中使用 INTO 选项可以将查询结果写进新表中，新表结构与 SELECT 语句选择列表中的字段结构相同。

[例]：

```
SELECT * INTO #new_publishers
FROM publishers
WHERE country='USA'
```

7.5.8　使用 UNION 运算符实现多查询联合

UNION 运算符可以将两个或两个以上的查询结果合并成一个结果集合显示。UNION 运算符的句法格式为：

```
查询 1[UNION [ALL] 查询 N]……
[ORDER BY 子句]
[COMPUTE 子句]
```

其中，查询 1 至查询 N 的格式为：

```
SELECT select_list
[INTO 子句]
```

[FROM 子句]
[WHERE 子句]
[GROUP BY 子句]
[HAVING 子句]
[例]：将 titles 表和 discounts 表的两个查询结果合并到一起。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
SELECT name=discounttype,value=discount
FROM discounts
UNION
SELECT title,ytd_sales
FROM titles
WHERE title LIKE 'T%'
```

联合查询时，查询结果的列标题为第一个查询语句中的列标题，因此要定义列标题时必须在第一个查询语句中定义。要对联合查询结果排序时，也必须使用第一查询语句中的列名。

```
SELECT name=discounttype,value=discount
FROM discounts
UNION
SELECT title,ytd_sales
FROM titles
WHERE title LIKE 'T%'
ORDER BY value
```

7.5.9 连接

1. 等值连接和自然连接

对于等值连接和自然连接，在 WHERE 子句中使用等于比较运算符，二者的区别在于等值连接的查询结果中列出所连接表中的所有列，包括它们之间的重复列。而自然连接的选择列表中删除被连接表间的重复列。

[例]：

（1）等值连接

```
SELECT *
FROM authors,titleauthor
WHERE authors.au_id=titleauthor.au_id
```

（2）自然连接

```
SELECT a.*,ta.title_id,ta.au_ord,ta.royaltyper
FROM authors a,titleauthor ta
WHERE a.au_id=ta.au_id
```

（3）不等连接

不等连接使用除等于运算符以外的其他比较运算符。这些运算符包括>、>=、<=、<、!>、!<和<>等。

（4）自连接

自连接中，使用同一个表的相同列进行比较，这时，对于同一个表应给出不同的别名。

[例]：使用自连接列出合著的图书标识及其作者姓名

```
SELECT DISTINCT t1.title_id,author=CONVERT(char(8),au_fname)+
                                        ''+CONVERT(char(15),au_lname)
FROM authors a,titleauthor t1,titleauthor t2
WHERE t1.title_id = t2.title_id
   AND t1.au_id<>t2.au_id
   AND t1.au_id=a.au_id
```

使用本复制品
请尊重相关知识产权!

2. 外连接

内连接中，查询结果中所显示的仅是符合查询条件的行，而采用外连接时，它不仅包含符合连接条件的行，而且还包括左表或右表连接中的所有行。

外连接操作符有*=和=*两种，采用*=连接时，查询结果中将包含第一个表中的所有行，而采用=*连接时，查询结果将包含=*操作符后面表中的所有数据行。

在进行一些统计时，需要使用外连接。例如，假定有两个表，一个表（PERSON）包含人员的姓名（NAME）及其标识（ID），另一个表（DESC）包含人员标识（ID）及其受奖惩情况（MEMO）。在统计单位的所有人员及其奖惩情况时，使用外连接书写查询语句就特别简单。即：

```
SELECT name,memo
FROM person,desc
WHERE person.id*=desc.id
```

7.5.10 子查询

子查询是嵌套在 SELECT、INSERT、UPDATE 和 DELETE 语句的 WHERE 子句和 HAVING 子句中的 SELECT 语句，它也可以嵌套在另一个子查询中。SELECT 语句中子查询的语法格式为：

```
SELECT [ALL | DISTINCT] subquery_select_list
[FROM {table_name | view_name } [optimizer_hints]
[[,{table_name2 | view_name2 } [optimizer_hints]
[.....,{table_name16 | view_name16 } [optimizer_hints]
[WHERER clause]
[GROUP BY clause]
[HAVING clause]
```

1. [NOT] IN 子查询

这种语句的执行分两个步骤：首先执行内部子查询，然后根据子查询的结果再执行外层查询。

```
SELECT title=CONVERT(char(25),title)
FROM titles
WHERE title_id IN
```

```
(SELECT title_id
FROM titleeauthor
WHERE au_id LIKE '99%')
```

2. [NOT] EXISTS 子查询

它返回逻辑值 TRUE 或 FALSE，并不产生其他任何实际值。所以这种子查询的选择列表常用“SELECT *”格式。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
SELECT title=CONVERT(char(25),title)
FROM titles
WHERE title_id EXISTS
(SELECT title_id
FROM titleeauthor
WHERE title_id=titles.title_id AND au_id LIKE '99%')
```

3. 由比较运算符引出的子查询

在使用单一比较操作符引出子查询时，必须保证子查询返回一个单值，否则将引起查询错误。比较运算符与 ALL 或 ANY 修饰符连用时，允许子查询返回多个值。

```
SELECT title
FROM titles
WHERE price>(SELECT AVG(price) FROM titles)
SELECT title
FROM titles
WHERE price>ALL
(SELECT price FROM titles WHERE type='business')
```

7.6　数据库访问对象 DAO

7.6.1　何时使用 DAO

DAO 把后台数据源的具体技术细节隐藏起来，开发人员不用考虑不同数据库系统的差异，例如，下面的代码能够工作在任何类型的数据源上。

```
.....
rsDepts=dbEmployees.Openrecordset("Depts")
Do While Not rsDepts.EOF
    lstDepts.AddItem rsDepts.Field("DeptName").Value
    rsDepts.MoveNext
Loop
RsDepts.Close

.....
```

DAO 可以访问本地的和远程的数据源。如果数据源是本地的或客户端的，DAO 通过 Jet 引擎来访问数据；如果数据源是远程的或 ODBC 驱动的话，最新的 DAO3.6 通过 ODBCDirect 来访问。

注意：其实 ODBCDirect 是一个使用 DAO 名字的 RDO，当使用 ODBCDirect 时，DAO 装载的不是 Microsoft Jet 数据库引擎，而是装载 RDO2.0。

随着 ADO 的推出，DAO 的优点就再也体现不出来了，因此在不久的将来，DAO 将逐渐退出历史舞台。但是，作为一项成熟的技术，DAO 曾经得到了广泛的使用，在一段时间内，DAO 的用户仍然会有大量的存在。总的来说，开发者可以在如下的几种情况下使用 DAO：

（1）应用系统使用的是本地数据库，如 Microsoft Foxpro 和 Microsoft Access 等。

（2）用户已经对 DAO 非常精通，但还没有时间学习 ODBC 或 ADO。

（3）用户需要对一个原有的系统进行修改，而原有的系统采用的是 DAO，且规模较大，把它移植成 ADO 需要大量的时间和金钱。

7.6.2　DAO 和 Jet 数据库引擎

DAO 可以访问本地数据库和远程数据库。这里指的本地和远程并不是地理位置上的概念，它们是两个逻辑概念。本地数据库可以在本地计算机（物理的）上，也可以不在本地计算机上，而远程计算机也并不局限于远程计算机（物理）上。

本地数据库是通过文件系统访问的数据库，它可以放在本地计算机上，也可以放在网络上，它由应用程序直接操作。而远程数据库只能由 DBMS 操作，应用程序对它的访问都要通过 DBMS 来完成，这样就产生了 Client/Server 体系结构，应用程序是 Client 端，DBMS 是 Server 端。

Microsoft Jet 是 Microsoft Access 和 Visual Basic 使用的一种数据库引擎。在 DAO3.1 版本以前，DAO 被定义为“Microsoft Jet 引擎的编程接口”，也就是说，DAO 和 Jet 在历史上几乎是同义词。DAO 访问数据库都要通过 Microsoft Jet 数据库引擎来完成。当然，这两个概念在内涵上是不一样的，Jet 不能被直接使用，只有通过 DAO 或 Access 才能直接使用 Jet。

但是，DAO3.1 之后增加了一项重要的功能，这就是 ODBCDirect 访问。ODBCDirect 使得 DAO 可以跳过 Jet 引擎，直接访问 ODBC 数据源。正是由于有了这项功能，DAO 不再绑定在 Jet 上了，DAO 与 Jet 仍有千丝万缕的联系。一般来说，如果应用程序使用的是本地数据库，那么 DAO 一般都通过 Jet 来访问数据库，反之，如果使用的是远程数据库，那么 DAO 通过 ODBCDirect 来访问数据库。

Jet 是通过 SQL 来驱动的，但是这种 SQL 可能和通常的实现不太一样。这种情况下可能会带来好处，也有可能会带来坏处。当使用 DAO 访问数据库时就必须考虑这种差别。因为 Jet 的 SQL 的运行和一些标准的 SQL 运行性能差异很大。考虑下面这个 SQL 语句：

```
SELECT * FROM Student,Teacher
WHERE Student.Course_No =Teacher.Course_No
```

在 SQL Server 中执行这个 SQL 语句时，SQLServer 会在内部自动把 Student 表和 Teacher 表使用的最优化方式连接起来，即用 Student 表的 Course_No 域的值来索引 Teacher 表，因此，每读取一条 Student 记录，只需读一条 Teacher 记录，整个语句的执行共需读取 2*COUNT

（Student）个记录。

但是，同样是这个语句，通过 DAO 和 Jet 来执行时，Jet 会把两个表进行笛卡儿连接，然后再滤掉一些不合条件的记录。这样一来，每读一条 Student 记录，都必须读取所有的 Teacher 记录，即 COUNT（Teacher）条 Teacher 记录，整个语句的执行共需读取 COUNT（Student）*COUNT（Teacher）条记录。如果这两个表的数据庞大的话，这个语句的执行速度将令人不可忍受。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

7.6.3　DAO 对象模型

DAO 提供了两种不同的对象模型，一种对象模型是通过 Jet 引擎来访问数据（如图 7.40 所示），另一种对象模型是通过 ODBCDirect 来访问数据。这两种对象模型都是层次结构的。下面将分别对这两种对象模型进行介绍。

DBEngine 是一个基对象，它包含了两个重要的集合（Collection），一个是 Errors 集合，另一个是 Workspaces 集合。对 DAO 的操作总会产生一些错误，每产生一个错误，DAO 就生成一个 Error 对象，这些 Error 对象都放在 Errors 集合中，可以用 Errors.Count 来计算错误的个数。事实上，对于每一个集合，都可以用 Collection.Count 来求出该集合中对象的个数。

每一个应用程序只能有一个 DBEngine 对象，但可以有多个 Workspace 对象，这些 Workspace 对象都包含在 Workspaces 集合中。每个 Workspace 对象都包含了一个 Databases 集合，Databases 集合中包含了当前所正在使用的 Database 对象。每个 Database 对象对应了一个数据库，它里面包含了许多用于操作数据库的对象。这些对象中，有一些是 Jet 数据库专用的，如 Container、TableDef 和 Relation 对象，有一些则是对所有数据库都有用的，如 Recordset 对象和 QueryDef 对象。

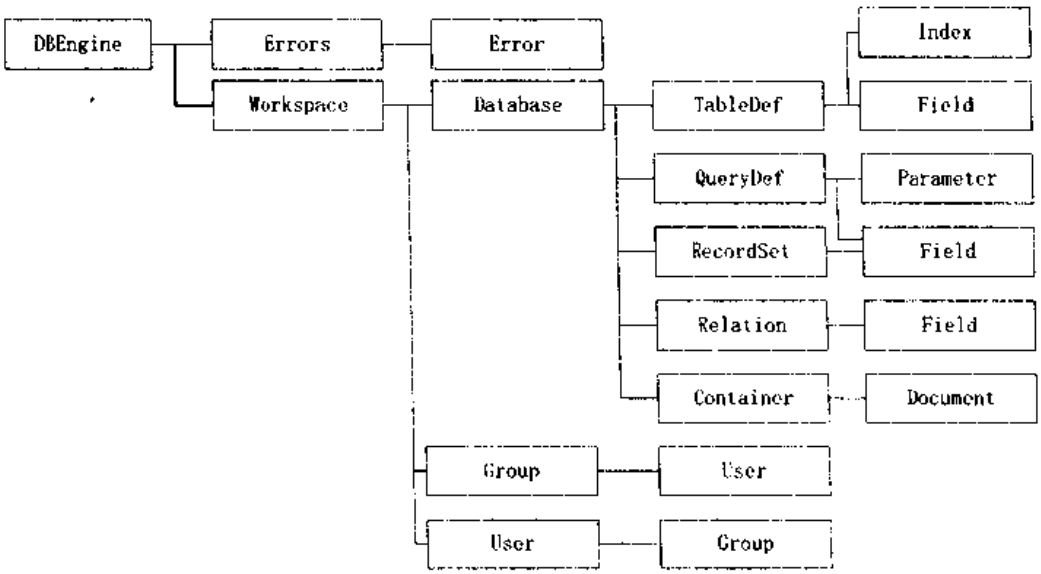


图 7.40　使用 Jet 的 DAO 对象模型

下面对对象模型中的主要对象进行详细的说明。

1. DBEngine 对象

DBEngine 对象是惟一的，不能创建，也不能声明。通常，可以用 DBEngine 对象的属

超星阅览器提醒您：
使用本复制品
须遵守相关知识产权!

性来设置数据库访问的安全性，即设置访问数据库的缺省用户名和缺省口令，如：

```
Dim DbEn As DAO.DBEngine = New DAO.DBEngine()
DbEn.DefaultUser = "RtLinux"
DbEn.DefaultPassword = "aaa"
```

由于 Jet 数据库引擎允许用户定义一个工作组，对于工作组中的每一个用户可以设置不同的数据库访问权限。必须把存储这个工作组有关信息的文件告诉 DAO，方法就是设置 DBEngine 对象的“SystemDB”的属性，如：

```
DbEn.SystemDB = "c:\" & "System.mdw"
```

当使用 Jet 数据库引擎时，必须把 DBEngine 对象的“DefaultType”属性设置为“dbUseJet”。

DBEngine 对象还提供了很多方法来操作工作区（Workspace）和数据库，如：Creat Workspace 方法创建一个工作区，Creatdatabase 创建一个数据库，OpenDatabase 打开一个数据库，CompactDatabase 压缩一个数据库，RepairDatabase 修复一个数据库，等等。

2. Error 对象

Error 对象是 DBEngine 对象的一个子对象。在发生数据库操作错误时，可以用标准的 VB 的 Err 对象来进行错误处理，也可以把错误信息保存在 DAO 的 Error 对象中。Error 对象包含以下属性：

（1）Description 属性。这个属性包含了错误警告信息文本，如果没有进行错误处理，这个文本将出现在屏幕上。

（2）Number 属性。这个属性包含了产生错误的错误号。

（3）Source 属性。这个属性包含了产生错误的对象名。

（4）HelpFile 属性和 HelpContext 属性。这两个属性设置有关这个错误的 Windows 帮助文件和帮助主题。

3. Workspace 对象

一个 Workspace 对象定义一个数据库会话（Session）。会话描绘出由 Microsoft Jet 完成的一系列功能，所有在会话期间的操作形成了一个事务范围，并服从于由用户名和密码决定的权限。所有的 Workspace 对象组合在一起形成了一个 Workspace 集合。可以用 DBEngine 对象的 CreateWorkspace 方法来创建一个新的工作区，只需把工作区的名称和用户信息传递给这个方法，如：

```
Dim ws As DAO.Workspace = dben.Workspaces(0)
    ws = dben.CreateWorkspace("Customers", "Admin", "pswrd", dbUseJet)
dben.Workspaces.Append(ws)
```

注意：当创建了一个新的 Workspace 对象时，它并不会自动添加到 Workspace 集合中，必须用 Append 方法把 Workspace 对象加到 Workspaces 集合中。

4. Database 对象

一旦用 CreateDatabase 创建了一个数据库或用 OpenDatabase 打开了一个数据库，就生成了一个 Database 对象。所有的 Database 对象都自动添加到 Databases 集合中。下面的这段代码就是使用 Database 集合列出了所有的数据库的路径名：

```
db = ws.OpenDatabase(txtMDBFile.text)
```

Database 对象有 5 个子集合，分别是 Recordsets 集合，QueryDefs 集合，TableDefs 集合，

超星阅览器 使用本复制品 版权所有 侵权必究

Relation 集合和 Containers 集合，这些集合分别是 Recordset 对象、QueryDef 对象、TableDef 对象、Relation 对象和 Container 对象的集合。Database 对象提供一些方法来操纵数据库和创建这些对象。

（1）Execute 方法。执行一个 SQL 语句，这个 SQL 语句不仅可以操作数据库中的数据，还可以是 DLL，用来修改数据库的结构。

（2）OpenRecordset 方法。在数据库中执行一个查询，查询可能涉及到表的连接，查询的结果作为一个 Recordset 对象返回。

（3）CreateQuery 方法。在数据库中创建一个存储过程并创建一个 QueryDef 对象。

（4）CreateRelation 方法。创建一个 Relation 对象，定义两个 TableDef 或 QueryDef 之间的关系。

（5）CreatTableDef 方法。在数据库中创建一个数据表，并返回一个 TableDef 对象。

5. Recordset 对象

Recordset 对象是使用最频繁的一个对象，它代表了数据库中一个表或一个查询结果的记录等。例如下面的语句在数据库的第一表中添加一条记录：

```
For Each tmpdb In Workspace(0).Databases
    Debug.Write("datbases(x).Name=", tmpdb.Name)
Next
```

还可以在 Recordset 中任意移动当前的记录的位置，使用的是 Recordset 对象的 MoveNext、MovePrevious、MoveFirst 和 MoveLast 方法。

Recordset 对象中还包含另一个对象 —— Field 对象，这个对象代表了数据表的一个字段，用这个对象可以访问数据表中的任何一个字段，如：下面的语句把表中当前记录的 Name 字段的值赋给变量 sName：

```
Dim db As DAO.Database = ws.OpenDatabase("user", " aaa")
SName = CStr(db.recordsets(0).Fields("name").value)
```

Source 参数可以是一个表名，也可以是一个查询的名字，还可以是一个用来创建 Recordset 对象的 SQL 语句。这个参数是必须的，而其他的三个参数是可选的。

Type 参数是指 Recordset 的类型，这里有必要说明一下 Dynaset（动态集）和 Snapshot（快照）之间的区别。Dynaset 这种 Recordset 对象的功能强大，使用灵活，当 Recordset 被创建时，只有每个记录的主键被取到且被缓存在本地，由于主键的大小总是小于整条记录的大小，所以 Dynaset 创建的速度很快。在 Dynaset 创建以后，如果要查询记录，则用缓存的主键来进行查询。相反，Snapshot 则是把整条记录都取出来存在本地，因此速度很慢，而且，如果别的用户修改了数据库，本用户将无法看到这种改变。

6. TableDef 对象

TableDef 对象也是一个经常使用的对象，它有两个子对象：一个是 Field 对象，另一个是 Index 对象。用 TableDef 对象可以访问单个表的每一个字段（Field 对象）和表的索引（Index 对象）。TableDef 对象的一个重要的方法是 CreateField。CreateField 方法用来在表中创建一个新的字段，它的语法为：

```
Field =tabledef.CreateField(fieldname,fieldtype,fieldlength)
```

三个参数分别指明新增字段的字段名，字段类型和字段长度。

对于动态地创建一个新表或动态地修改表的结构时，这个对象是必不可少的。例如下

使用本复制品
请尊重相关知识产权!

面的语句创建一个只有字段的新表：

```
Dim db As DAO.Database = ws.OpenDatabase("c:\FirstTest.mdb","RtLinux", "aaa")
Tdf= db.CreateTableDef("Teacher")
Fld= Tdf.CreateField("BirthDay",dbText,30)
Fld.Required=True
Fld.AllowZeroLength=True
Fld.DefaultValue="Mike"
Tfd.Fields.Append tld
Db.TableDefs.Append tdf
Db.close
```

7. QueryDef 对象

QueryDef 对象用来定义一个查询。它有两个子对象：一个是 Field 对象，一个是 Parameter 对象。用 Database 对象的 CreateQueryDef 方法来创建一个 QueryDef 对象。用户可以在 Jet 对象模型中这样使用 QueryDef 对象：

（1）用 QueryDef 对象的 SQL 属性来设置或返回查询的 SQL 语句定义；

（2）用 QueryDef 对象的子对象 Parameter 来设置或返回查询定义的参数；

（3）用 QueryDef 对象的 Type 属性来设置查询的类型，查询的类型包括：从已有的表中取出记录、创建一个新表、把一个表中的记录插入到另一个表中、删除记录、更新记录等；

（4）用对象的 MaxRecords 属性来限制查询返回的记录数；

（5）用对象的 ReturnRecords 属性取得查询返回的记录数；

（6）用对象的 Execute 方法来执行查询；

（7）用对象的 RecordsAffected 属性来返回此查询的所影响的记录数；

（8）用对象的 ODBCTimeout 来设置查询 ODBC 数据源时花费的最长时间；

（9）用对象的 Connect 属性来连接 ODBC 数据源进行查询；

（10）用对象的 OpenRecordset 方法返回 Recordset 对象，并用此 recordset 对象来取得查询的结果。

8. Relation 对象

Relation 对象用来定义不同的表或不同查询中字段之间的关系。例如，定义一个表中的主键为另一个表的外键，定义一个表中的字段之间的一对一或一对多的关系等等。用 Database 对象的 CreateRelation 方法来创建一个 Relation 对象。

9. Field 对象

Field 对象是 Jet 对象模型中的最低层的对象，它代表了一个表中的一个字段。可以设置字段的各种属性，如字段类型（“Type”属性）、字段长度（“Size”属性）、当前记录中字段的值（“Value”属性）、字段的缺省值（“DeffaultValue”属性）、字段是否可以为空（“Required”属性）等等。

10. Group 对象和 User 对象

User 对象代表了数据库的一个用户，Group 对象则包含了具有相同权限的一组用户。可以用这两个对象来管理数据库的用户。下面给出了一个函数来创建用户，使用这个函数时，必需具有创建用户的权限。

ODBCDirect 是一种不用加载 Microsoft Jet 数据库引擎就可以操作 ODBC 数据库服务器

的技术。它是 DAO3.5 之后才增加的新特性。ODBCDirect 提供了一种机制，使得 DAO 能把基于 Jet 的访问变为基于 RDO 的访问方法。但是，ODBCDirect 并不能完全替代 RDO。要在应用程序中使用 ODBCDirect 来访问数据库，只需要把下面这些话添加到应用程序的开始就可以了（任何使用 DAO 对象之前）：

```
DbEngine.DefaultType=dbUseODBC
```

也可以在 CreateWorkspace 方法中指定 dbUseODBC 参数来使用 ODBCDirect。如：

```
Dim wrkODBC as dao.Workspace = new dao.Workspaces(0)
Set wrkODBC =CreateWorkspace("ODBCWorkspace","admin","",dbUseODBC)
```

使用 ODBCDirect 的 DAO 对象模型和 Jet 对象模型大同小异，模型的层次结构图如图 7.41 所示。

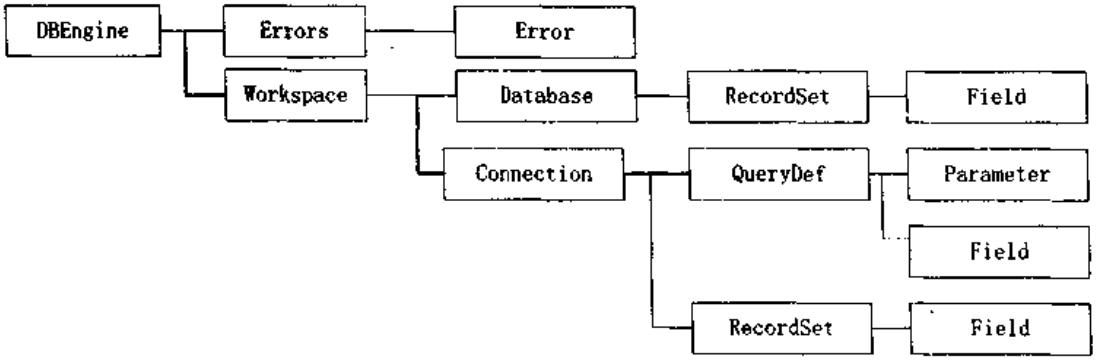


图 7.41　使用 ODBCDirect 的 DAO 对象模型

与 Jet 对象模型相比，ODBCDirect 去掉了用于数据库定义的对象，如 TableDef 对象和 Relation 对象，但是新增 Connection 对象。Connection 对象代表了和 ODBC 数据源的一个连接。一般用 OpenConnect 方法来建立一个连接并创建一个 Connection 对象。OpenConnection 的语法为：

```
conection = workspace.OpenConnection(name,options,readonly,connect)
```

四个参数的含义和 OpenDatabase 方法的四个参数的含义一样，在此就不再赘述了，请参阅前面讲过的 Opendatabase 方法。

使用 ODBCDirect 操作 ODBC 数据源有如下一些优点：

（1）ODBCDirect 通过提供对 ODBC 数据源的直接访问，使程序代码执行得更快、效率更高。因为它不需要加载 Microsoft Jet 数据库引擎，所以在客户端只需消耗较少的资源。ODBC 服务器可以响应所有的查询处理。

（2）ODBCDirect 提供对特定服务器进行访问的功能，这些访问是 Microsoft Jet 使用 ODBC 无法实现的。例如，对于支持光标集的服务器，ODBCDirect 允许用户指定光标集位置是在本地还是在服务器上。此外，也可以通过指定输入值以及检查返回值，来与服务器级别的存储过程进行交互，这在 Microsoft Jet 中是不可以的。

（3）DBCDirect 支持数据的批量更新，可以将 Recordset 对象在本地的更改存入高速缓存，然后批量向服务器导出这些更改。

（4）使用 ODBCDirect，既可以创建简单的无游标集的结果集，也可以创建复杂的游标

使用本复制品

基。它也可以执行返回多个结果集的查询，或是限制返回的行数，并监视所有远程数据源生成的信息和错误，并不影响查询的执行性能。

但是，ODBCDirect 也有一些缺点。下面的一些功能就是 ODBCDirect 无法完成的，但是 Microsoft Jet 能够完成的。

（1）可更新的连接。只有使用 Microsoft Jet 工作区才能更新基于多表连接的 Recordset 对象中的数据。

（2）异种连接。只有使用 Microsoft Jet 工作区才能执行不同数据源种的表连接。

（3）数据定义语言（DDL）的操作。只有使用 Microsoft Jet 工作区才能通过 DAO 进行 DDL 操作。ODBCDirect 不提供 TableDef 对象，所以不能使用 DAO 来创建或修改表。不过，使用 ODBCDirect 可以通过执行 SQL DDL 语句来执行 DDL 操作。

（4）窗体和控件结合。如果应用程序需要将 ODBC 数据源种的数据与窗体相结合，就必须使用 Microsoft Jet，在 ODBCDirect 工作区中访问的数据不能与窗体或控件结合。

如果应用程序不需要上述功能，就可以使用 ODBCDirect。

注意：可以在应用程序中同时定义 Microsoft Jet 和 ODBCDirect 工作区，并通过各种形式将它们组合起来。例如，在同一个函数中，可以定义 Microsoft Jet 工作区来使用 DAO 执行 DDL 操作，同时也可以定义 ODBCDirect 工作区来执行异步查询。

7.7 远程数据对象 RDO

RDO 很好地包装了 ODBC API 的大部分功能，它专门访问 ODBC 数据源。RDO 几乎能完成所有能用 ODBC API 完成的功能。RDO 能很好地支持大型数据库系统，如 SQL Server，Oracle 和 Sybase 等，它还可以处理复杂的存储过程和结果集。

ADO 中包含了 ODBC 的所有功能，且它的使用和 RDO 一样简单，因此，ADO 将逐渐代替 RDO。

一般来说，开发人员可以在以下几种情况下选用 RDO 来访问数据源：

（1）应用系统的后台数据库是一个大型的 ODBC 驱动的数据源，如 SQL Server、Oracle 等。

（2）应用程序系统要求较快的数据库访问速度，且需处理复杂的存储过程，用户又不愿意进入底层 ODBC API 编程，想缩短编程时间。

（3）用户要对一个原有的系统进行修改，而原有的系统采用的是 RDO，且规模较大，把它移植成 ADO 需要花费大量的时间和金钱。

7.7.1 RDO 对象模型

远程数据对象提供了一系列的对象，用来满足远程数据库访问的特殊要求。在 ODBC API 和驱动程序管理器之上，RDO 实现了很薄的一个代码层，用来连接、创建结果集和游标，并且使用尽可能少的工作站资源执行复杂的过程。如果代码创建了 ODBCDirect *Workaspace* 对象，那么 RDO 也被 DAO 访问的，这在上一节中已经介绍过了。

利用 RDO，应用程序不需要使用本地的查询处理程序即可访问 ODBC 数据源。这意味着，在访问远程数据库引擎时，可以获得更好的性能与更大的灵活性。

通过使用 RDO，可以创建简单的无游标结果集或更复杂的游标；可以执行查询并处理任意数量的结果集；可以执行返回结果集的存储过程，无论存储过程是否带有输出参数和返回值；可以执行包括数据操作或数据定义运算在内的动作查询；可以限制返回或处理的数据行数；还可以在不妨碍执行查询的情况下，监视远程数据资源产生的所有信息和错误；RDO 还支持同步、异步或事件驱动的异步处理，因此，即使在执行冗长的查询或者重新定位当前指针时，应用程序也不会被阻塞。

RDO 对象和集合的属性描述了数据库部件的特征，也描述了用来操纵它们的方法。在此总体框架下，可以在对象和集合之间建立联系，这些联系表示了数据库系统的逻辑结构。RDO 的对象模型如图 7.42 所示。

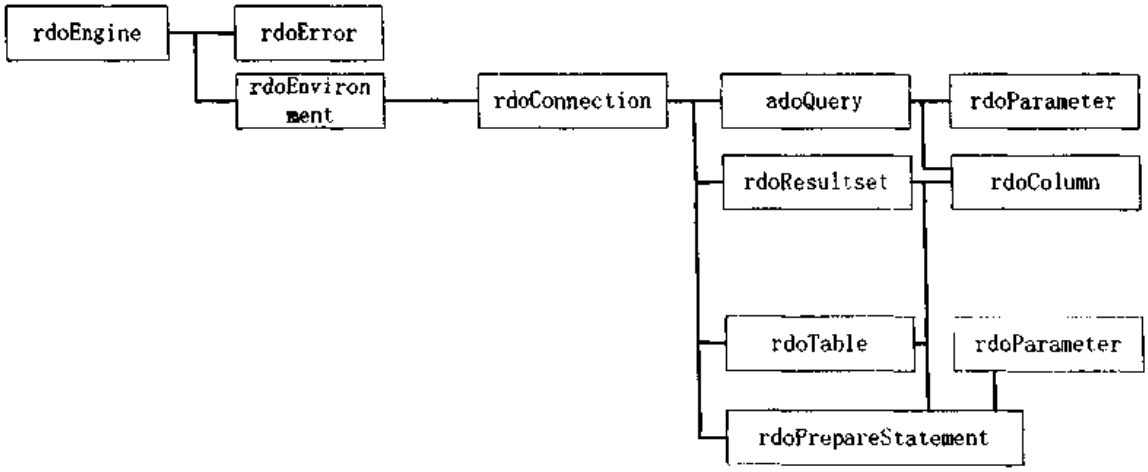


图 7.42　RDO 对象模型

除了 rdoEngine 对象外，每个对象都保存在一个相关的集合中。在首次访问并初始化 RDO 时，RDO 自动创建一个 rdoEngine 和缺省的 rdoEnvironments（0）实例。

远程数据对象编程模式与数据访问对象（DAO）编程模式在很多方面类似。但它的重点集中在处理存储过程及其结果集上，而不是仅用在 ISAM 编程模式的数据访问检索方法上。下面逐一介绍 RDO 对象模型中的各个对象。

1. RdoEngine 对象

这个对象是 RDO 对象模型中最基本的对象。在应用程序的第一次访问 RDO 时，自动创建 rdoEngine 对象。这个对象不能由用户显式地创建或声明。RdoEngine 对象和 DAO 的 DBEngine 对象类似。

2. RdoError 对象

RdoError 对象用于处理 RDO 所产生的所有的 ODBC 错误和消息，当错误发生时，这个对象自动被创建。RdoError 对象和 DAO 的 Error 对象类似。

3. RdoEnvironment 对象

RdoEnvironment 对象为特定的用户名定义了连接及事务作用域的一个逻辑集合。包括打开的和分配的（但未打开）连接，提供了并发事务的机制，并且为数据库的数据操作语言（DML）提供了安全的上下文，在应用程序第一次访问 RDO 时，自动创建了一个 rdoEnvironment 对象并把它加到 rdoEnvironments 集合中，用 rdoEnvironments（0）来引用它。这个 rdoEnvironment 对象是缺省的对象，也可以在程序中显式地创建 rdoEnvironment 对象，如果程序中没有另外创建新的 rdoEnvironment 对象，应用程序使用 rdoEnvironments（0）。

用 rdoEngine 的 rdoCreateEnvironment 方法来创建一个新的 rdoEnvironment 对象：

var=rdoEngine. RdoCreateEnvironment(Name,UserName,Password)

其中 Name 参数是 rdoEnvironment 对象的名字，UserName 和 Password 参数分别是连接 ODBC 数据源的用户名和密码。它们是 rdoEnvironment 对象的属性。

RdoEnvironment 对象和 DAO 的 Workspace 对象类似。

4. RdoConnection 对象

RdoConnection 对象表示远程数据源和该数据源上特定的数据库之间打开的一个连接，或者是一个已分配但仍未连接的对象，该对象可用于随后建立的一个连接。

可以用 rdoEnvironment 对象的 OpenConnetion 方法来创建一个 rdoConnection 对象并建立一个连接。

```
Connection=rdoEnvironment. OpenConnetion(dsname[,prompt[,readonly[,connect[ _
                                                    options]]]])
```

RdoConnection 对象和 DAO 的 Connection 对象类似，因此，上面这个方法的参数含义和前面介绍的一样。

例如，下面的语句展示了应用程序和数据源 WorkDB 建立一个连接，连接时提示用户输入用户名和密码：

```
cn = en.OpenConnetion(dsName:="NtserverTest",Prompt:=rdDriverPrompt)
```

也可以先声明一个 rdoConnection 对象，再用 EstablishConnection 来建立连接，如：

```
dim cn as New RDO.__rdoConnection()
cn.Connect="uid=;pwd=;DSN= NtserverTest"
cn.cursordriver=rdUseOdbc
cn. EstablishConnection rdDriverNoprompt
```

以上讲的都是有 DSN 的连接，用 RDO 还可以创建没有 DSN 的连接。创建没有 DSN 的连接可以简化客户应用程序的设置和安装、加快连接速度、提高安全性。创建没有 DSN 的连接时，必须把 OpenConnection 的 dsName 参数设置为空串，而且在 Connect 串中，DSN 参数必须在 SERVER 和 DRIVER 参数之后。例如下面的语句创建一个没有 DSN 的连接：

```
Dim Cn As rdo.rdoConnection
Dim En As rdo.rdoEnvironment, Conn As String
En = RDO.rdoEnvironments(0)
   Conn$ = "UID=Holly;PWD=kkk;" & "DATABASE=MyDb;" _
   "&"SERVER =STD;" & "DRIVER={SQL SERVER};DSN="";"
   Cn = En.OpenConnection(dsName:="", _
Prompt:=rdDriverNoprompt, connect:=Conn$)
```

RdoConnection 对象和 dao 的 Database 类似。

5. RdoTable 对象

RdoTable 对象表示一个基本表或 SQL 视图的存储定义。使用这个对象可以管理数据库的结构。当已经和数据源建立了连接之后，数据源中的每个基本表或 SQL 视图都成为一个 RdoTable 对象而加入到 RdoTables 集合中。添加表、删除表或修改表的结构就是对 rdoTable 对象的操作。如下面的代码列出了数据源中的所有表的名字：

```
Dim tb As rdo.rdoTable
Dim cn As RDO._rdoConnection
cn = rdo.rdoEnvironments(0).OpenConnection(dsName:="WorkDB", _
Prompt:=rdDriverNoprompt, Connect:="Uid=;pwd=;database=Pubs")
For Each tb In cn.rdoTables
    Debug.Write(tb.Name)
Next
```

RdoTable 对象和 DAO 的 TableDef 对象类似。

6. RdoResultset 对象

RdoResultset 对象表示运行一个查询所产生的数据行。这个对象是 RDO 编程中最常用到的对象。当执行一次查询时，查询的结果就放在结果集中。一个结果集可以包含数据库中一个或多个表中的字段。根据游标类型的不同，可以有 4 种不同类型的 RdoResultset 对象：

（1）记录只能向前移动的结果集（Forward Only）。在这种结果集中，每一行数据都能访问到，但是记录移动只能使用 MoveNext 方法来从前往后移动。

（2）静态类型的结果集（Static-Type）。它是可以用来查找数据或产生报表的记录的一个静态拷贝。如果数据源能够修改，那么静态结果集也能被修改。

（3）键集（Keyset-Type）类型的结果集。这种结果集可以包含可修改的行，在这种结果集中移动是不受限制的。这种结果集可以动态地添加、修改和删除记录。它的成员是固定的。

（4）动态类型的结果集（Dynamic-Type）。这种结果集可以包含可修改的行，可以动态地添加、修改和删除记录，而且，它的成员是不固定的。

可以用 rdoQuery 对象或 rdoConnection 对象的 OpenResultset 方法来创建一个 RdoResultset 对象。

在创建了 RdoResultset 之后，就可以访问其中的数据了。在处理结果集时，需要注意以下几点：

（1）如果该结果集以游标的形式创建，必需尽快将结果置入结果集，因为游标所取得的数据行和页可能被远程数据库引擎锁定。直到定位到最后一行时该结果集才算完成置入。

（2）在很多情况下，无游标的结果集更为有效。RdoResultset 对象的功能以满足需求为宜，不要具有多余的特性。例如，如果并不需要使用游标修改数据，就不要请求允许更新的结果集。如果仅仅要置入到一个列表框中，可以任意滚动的游标也是不必要的。

（3）考虑使用无游标的结果集和动作查询来执行更新。在很多情况下，这种方法能够提高性能。尽管实现起来更复杂一些，在不能直接访问基本数据时，可以采用这种方法。

（4）尽可能地使用异步操作和事件过程，以避免应用程序被锁死，至少能够减少这种可能性。也可以异步处理 Move 方法，例如使用 MoveLast，防止在置入到结果集时被锁死。

（5）复杂的结果集处理创建几个小的结果集，这常常会使速度更快，而且可以更好地管理系统资源。

（6）尽可能从存储过程产生结果集，因为这样可以提高服务器、网络和工作站的效率，并使应用程序的开发更为简单。

（7）从连接上断开已有的，以客户端的批处理游标的形式创建的 RdoResultset 对象。可以继续使用 Edit 或 AddNew 方法修改数据。当准备将所作的修改传回数据库时，可以将其 ActiveConnection 对象设置为一个打开的 rdoConnection 对象，从而将 RdoResultset 关联到一个打开的连接上。

RdoResultset 对象和 DAO 的 Recordset 对象类似。

7. RdoQuery 对象

RdoQuery 对象是一个 SQL 查询定义，可以包括 0 个或多个参数。可以用 rdoConnection 对象的 CreateQuery 方法来创建一个新查询，如：

```
Dim Qd As rdo.rdoQuery
Dim cn As rdo.__rdoConnection
Qd = cn.CreateQuery("TestQuery", "Select * from Student Where Name =?")
```

RdoQuery 对象和 DAO 的 QueryDef 对象类似。

8. RdoColumn 对象

RdoColumn 对象表示具有公共数据类型和公共属性的一列数据。这个对象是 rdoTable、rdoQuery、rdoResultset 对象的子对象。可以使用该对象的“Value”属性来访问某一列的值，也可以使用该对象的“Type”属性或“Size”属性来修改数据库中表的结构。

RdoColumn 对象和 DAO 的 Field 对象相似。

9. RdoParameter 对象

RdoParameter 对象表示与 rdoQuery 对象关连的一个参数。查询参数可以是输入、输出或输入输出参数。这个对象在进行带参数查询时自动创建。可以用 rdoParameter 对象的“Direction”属性来设置参数类型：输入参数（rdParamReturnValue）。例如下面的代码执行一个带有 4 个参数的 SQL 语句。

```
Dim Cqy As rdo.rdoQuery = New rdo.rdoQuery()
Cqy.SQL = "{?=call sp_MyProc(?,?,?)}"
Cqy(0).Direction = rdReturnValue
Cqy(1).Direction = rdreturnImput
Cqy(2).Direction = rdreturnImput
Cqy(3).Direction = rdReturnOutput
Cqy(1).Direction = "Victoria"
Cqy(0).Direction = 21
```

RdoParameter 对象和 DAO 的 Parameter 对象类似。

10. RdoPrepareStatement 对象

这个对象预先定义了一个查询。它已经过时，只是为了保持与以前的 RDO 版本兼容才保留这个对象。在 RDO2.0 之后，应该使用 rdoQuery 对象。

7.8　ODBC API

ODBC API 是一套复杂的函数集，可提供一些通用的接口，以便访问各种后台数据库。对现今的客户机/服务器应用程序来说，每个数据库系统都有各自的接口。例如 SQL Server、Oracle、DB2 和 Infomix 等各种数据库系统提供的接口就完全不一样。对程序员来说，这是一件非常头疼的事情。

ODBC 的出现成为一件大喜事，在添加了一个附加层之后，用户只需要学习和掌握一套函数集，随后就可以直接使用任何数据库系统中的大多数特性。但是，不要认为 ODBC 提供了一个统一的接口，同时也提供了简便的操作。事实上，ODBC API 的使用是非常复杂的。但为什么还要在 VB 中使用 ODBC API 呢？因为尽管 RDO 实现了 ODBC 的一部分功能，但不是全部功能，ODBC API 则能提供最佳的性能。

7.8.1　ODBC 结构

如果应用程序调用一个 ODBC API 函数，ODBC Administrator 或 Driver Manager 会把命令传递给适当的驱动程序。经过翻译之后，驱动程序会将命令传递给特定的后端数据库服务器，采用它能理解的语言或代码。而通过 ODBC 返回的任何结果或结果集都将会沿着相反的方向传递。ODBC 的结构如图 7.43 所示。

对 Visual Basic 应用程序来说，在处理函数调用时，驱动程序和驱动程序管理器（Driver Manage）是一个整体。应用程序用 ODBC API 来完成以下任务：

（1）请求与数据源建立连接，创建一次会话；

（2）向数据源发出 SQL 请求；

（3）定义一个缓冲区和数据格式，用来存储 SQL 请求结果；

（4）提取 SQL 请求的结果；

（5）处理各种错误；

（6）给用户报告结果；

（7）事务提交或事务撤销；

（8）中断与数据源的连接。

上述结构中，驱动程序管理器（Driver Manager）是一个 DLL，它由 Microsoft 提供，是一个带有入口函数库的动态连接，驱动程序管理器的基本任务是加载驱动程序。此外，还具有以下功能：

（1）根据 ODBC.INI 文件，把数据源名映射到相应的驱动程序；

（2）处理几个 ODBC 初始化函数；

（3）进行参数合法化检验。

结构图（图 7.43）中 SQL Server ODBC 驱动程序是一个 DLL，用来完成 ODBC 函数调用并与数据源进行对话。当应用程序调用 SQLDriverConnection 时，驱动程序管理器加载驱动程序。根据相应程序的要求，驱动程序可以完成以下任务：

（1）建立与数据原的连接；

（2）向数据源提交请求；

（3）根据应用程序的需要，完成数据格式的转换；
（4）把结果返回给应用程序；
（5）申请并控制游标（Cursor）；
（6）根据数据源的需要，完成事务初始化。
上述这些功能对应用程序来说都是透明的。

超星阅览器提醒您：
使用本复制品，
请尊重相关知识产权！

结构图（图 7.43）中的物理数据源是 DBMS、操作系统和网络平台的一个综合体。

和 Windows 其他地方一样，为了处理对象，ODBC 需使用相应的句柄。在能真正做些什么事情以前，必须先从 ODBC 或 Windows 获得这种句柄。ODBC 提供了四个句柄：环境、连接、语句和描述符句柄，并且必须在连接数据源之前申请环境句柄。连接句柄的作用是将资源分配给实际的数据源连接。应用程序在与数据源连接以前必须先申请一个连接句柄，每个连接句柄只与一个环境句柄相连，一个环境句柄可与多个连接句柄相连；语句句柄用于管理对系统发出的实际请求，它必须与一个连接关联在一起，而这个连接也必须与环境关联到一起，应用程序在提交 SQL 请求以前必须先请求一个语句句柄，每一个语句句柄只与一个连接句柄相连，一个连接句柄可与多个语句句柄相连；描述符句柄则提供一些特殊的描述信息，例如结果集的数据列信息，或 SQL 语句的动态参数等等。图 7.44 显示了环境句柄、连接句柄和语句句柄之间的关系。

存储在数据源中的数据都有一个数据类型，称为数据源数据类型或 SQL 数据类型，SQL 类型可以是某个数据源特有的。驱动程序在 ODBC SQL 语法及驱动程序数据类型中也定义了一套数据类型，称为 ODBC SQL 数据类型。驱动程序负责 SQL 数据类型和 ODBC SQL 数据类型之间的映射工作。驱动程序通过函数 SQLGetTypeInfo 返回这些映射关系，在函数 SQLColAttributite、SQLDescribeCol 和 SQLDescribeParam 中，驱动程序还用 ODBC SQL 数据类型来描述列和参数的数据类型。

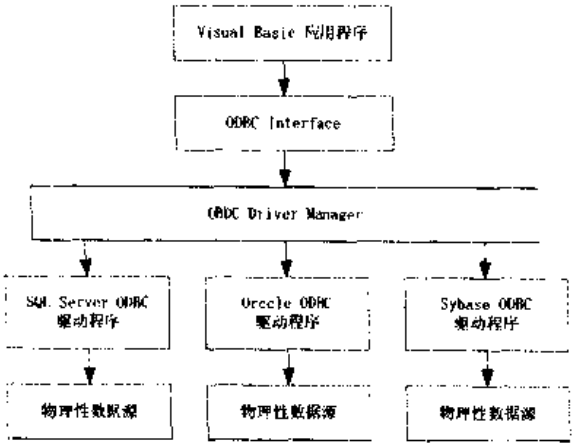


图 7.43　ODBC 的结构

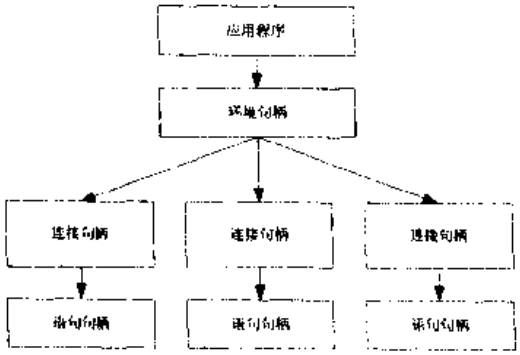


图 7.44　环境句柄，连接句柄，语句句柄之间的关系

每一种 SQL 数据类型对应一种 Visual Basic 数据类型。在默认情况下，驱动程序假设 SQL 语句中的列或参数的数据类型与连接在其上的 Visual Basic 数据类型是相对应的。如果 Visual Basic 数据类型不使用默认的数据类型，应用程序可以用函数 SQLBindCol、SQLGetData、SQLBindParameter 中的参数 fcType 来指定另外一个数据类型。在从数据源返回数据之前，驱动程序将数据转换为指定的数据类型。同样，在把数据传给数据源之前，驱动程序将指定

使用本复制品
请尊重相关知识产权!

的 Visual Basic 数据类型转换成 SQL 数据类型。

7.8.2 使用 ODBC API 访问数据库

为了存取 ODBC 数据源，应用程序应该包含以下的步骤：

1. 连接 ODBC 数据源

要求给定 ODBC 数据源名称以及其他一些必要的信息。

在应用程序调用 ODBC 函数之前，必须初始化 ODBC 接口，建立一个环境句柄。首先声明一个环境句柄变量，如：

Dim hEnv1 as Long

如果使用的是 ODBC 2.X，那么调用函数 SQLAllocEnv，并把环境变量以引用传递的方式传给该函数。驱动程序管理器将初始化 ODBC 环境，为环境句柄分配存储空间，并返回一个环境句柄。如：

SQLAllocEnv（hEnv1）

如果使用的是 ODBC 3.X，那么用参数 SQL_NULL_ENV 调用函数 SQLAllocHandle，如：

SQLAllocHandle（SQL_HANDLE_ENV,SQL_NULL_HANDLE,hEnv1）

以上步骤在一个应用程序中只能进行一次。但在一个 ODBC 环境中可以连接多个数据源。下面就来看如何连接数据源。

在连接数据源之前，应用程序先分配一个或多个连接句柄。首先声明一个连接句柄变量，如：

Dim hDbc1 as Long

如果使用的是 ODBC 2.X，那么调用函数 SQLAllocConnect，并把连接句柄变量以引用传递的方式传给该函数。驱动程序管理器为该连接分配一块储存空间，并返回一个连接句柄。如：

SQLAllocConnect（hEnv1，hDbc1）

如果使用的是 ODBC3.X，那么用参数 SQL_HANDLE_DBC 和环境句柄来调用函数 SQLAllocHandle，如：

SQLAllocHandle（SQL_HANDLE_DBC，hEnv1,hDbc1）

现在就可以指定驱动程序和数据源了，并且用函数 SQLDriverConnect 来连接指定的数据源。如：

SQLDriverConnect（hDbc1，Me.hWnd，“DSN＝Kill；Server＝Kernel；UID＝Rtlinux；PWD=”_　SQL_NTS,OUTConnectionString,SQL_NTS,iStringLength2Ptr,SQL_NTS）

2. 处理 SQL 语句

（1）应用程序把 SQL 语句字符串放入一个缓冲区。如果这个 SQL 语句含有参数，还应该设置参数值。

（2）如果 SQL 语句返回一个结果集，还要为该语句申请一个游标名。

（3）应用程序以“准备”或“立即执行”的方式提交 SQL 语句。

（4）如果 SQL 语句建立一组结果集，应用程序可以查询结果的属性，例如列数、列的数据类型等。为每一列连接一个缓冲并提取结果。

（5）如果 SQL 语句产生错误，则提取错误信息并采取相应的措施。

在处理 SQL 语句之前，首先必须分配一个语句句柄。先声明一个语句句柄变量，如：

Dim hStmtl as Long

如果使用的是 ODBC 2.X，那么调用函数 SQLAllocStmt，如：

SQLAllocStmt1（hDbc1，hStmtl）

如果使用的是 ODBC3.X，那么用参数 SQL_HANDLE_STMT 和参数 hDbc1 调用函数 SQLAllocHandle，如：

SQLAllocHandle（SQL_ HANDLE_STMT, hDbc1, hStmtl）

接下来就可以执行 SQL 语句了。如果采用“准备”方式执行 SQL 语句，应用程序要做如下步骤：

（1）调用 SQLPrepare 函数准备一个 SQL 语句，把 SQL 语句作为函数的一个参数，例如：SQLPrepare（hStmt1，“Select Name，Age From Employee Where AGE＝?”，SQL_NTS）。

（2）设置 SQL 语句中的参数值。如果 SQL 语句中出现了问号（?），那么表明这个 SQL 语句是带参数的，如：SQLBindParameter（hStmt1，1，SQL_Param_INPUT），SQL_C_Clong,SQL_INTEGER，0，0，age，0 name，vbNull）。这样每次查询以后，Name 字段值就存放在变量 name 中。

（3）调用函数 SQLExecute 函数来执行 SQL 语句。如：SQLExecute（hStmt1）。

（4）提取查询结果。这个任务由函数 SQLFetch 来完成，如 SQLFetch（hStmt1）。

（5）根据程序需要，可有选择地进行这五个步骤。

如果是采用“立即执行”的方式执行 SQL 语句，那么省去上面的的第一步，在执行 SQL 语句时，用函数 SQLExecuteDirect。如：

SQLExectDirect（hStmt1，1，SQL_C_CHAR,SQL_CHAR，Name_LEN,0 name,vbNull）。

但是就速度而言，“准备”方式比“立即实行”方式要快，因此，只在下面的情况下，考虑使用“立即执行”方式：

（1）应用程序的 SQL 的语句只执行一次；

（2）应用程序在执行前不需要查询有关结果信息。

3. 结束事务

可以提交事务，也可以撤销事务。有两种提交事务的模式：一种是自动模式，另一种是手动模式。

在自动模式下，每个 SQL 语句都被认为成一个完整的事务而自动提交。在手动模式下，事务由一个或几个 SQL 语句组成，如果应用程序提交一个 SQL 语句时没有活动事务，驱动程序就建立一个新的事务，在后续的 SQL 语句提交过程中，驱动程序保持这个处于活动的事务，直到应用程序调用函数 SQLTransact（ODBC2.X）或 SQLEndTran（ODBC 3.X）进行事务提交或撤销。

如果驱动程序支持 SQL_AUTOCOMMIT 连接选项，则缺省的事务模式是自动提交模式；如果不支持 SQL_AUTOCOMMIT 连接选项，则缺省的事务模式是手动模式。应用程序可以调用函数 SQLSetConnectOption（ODBC2.x）或 SQLSetConnectAttr（ODBC3.x）进行自动/手动提交模式。在进行模式切换时，驱动程序将自动提交当前连接中的所有活动事务。

应用程序应该用函数 SQLTransact 或 SQLEndTrans 来处理事务，而不要用 SQL 语句中的 COMMIT 或 ROLLBACK 来处理事务，COMMIT 或 ROLLBACK 语句的结果取决于驱动程序及相连接的数据源。

注意：不管是用自动模式还是用手动模式处理事务，也不管是提交事务还是撤销事务，

只要是事务处理都将引起数据源关闭游标并删除与该数据源有关的所有存储计划。

4. 中断连接

在完成对数据库的存取后，中断与数据源的连接。

用函数 SQLDisconnect 来中断与数据源的连接。例如，下面的语句中断连接句柄 hDbc1 所指的数据源连接。

SQLDisconnect(hDbc1)

在中断连接连接之后，必须释放所有的句柄，包含语句句柄、连接句柄和环境句柄。当使用 ODBC2.X 时：

（1）释放语句句柄，用函数 SQLFreestmt，如 SQL_DROP；

（2）释放连接句柄，用函数 SQLFreeConnect，如 SQLFreeConnect（hDbc1）；

（3）释放环境句柄，用函数 SQLFeeEnv，如 SQLFreeEnv（hEnv1）。

超星阅览器提醒您：
使用本复制品
请尊重版权！

当使用 ODBC3.X 时，释放连接句柄和环境都用函数 SQLFreeHandle，只不过带的参数不同。当释放连接句柄时，用参数 SQL_HANDLE_DBC；当释放环境句柄时，用参数 SQL_HANDLE_ENV。如果释放语句句柄时用 SQL_DROP，那么，也使用 SQLFreeHandle 函数，参数改为 SQL_HANDLE_STMT，否则和 ODBC2.X 一样使用 SQLFreeStmt。

7.9 ADO 数据对象

OLE DB 是一种低层接口，它提供了很多 COM 接口，结构很复杂，因此，不适宜在 Visual Basic 中直接访问 LOE DB。但是 ActiveX 数据对象（ADO）封装并且实现了 OLB DB 的所有功能。所以，可以通过 ADO 来访问 OLE DB 数据源。

若要在 Visual Basic 中对 ADO2.0 对象进行访问，可设置对合适的 ADO 类型库的访问。有两种 ADO 类型库。一种叫 ADODB，包含在 MSADO15.DLL 中。它以“Microsoft AxtiveX Data Objects 2.0”出现在引用里面。另一种叫做 ADOR，包含在 MSADOR15.DLL 中。它以“Microsoft ActiveX Data Objects Recordset 2.0 Library”出现在引用对话框中。

在两种类型库中，第一种类型库（ADODB）更强大，具有更多功能。包含了主要的 ADO 对象，而且是可能在大多数情况下用户希望使用的。第二中是只支持记录集的 ADODB 类型库的一个“轻量”子集。如果用户只想操作记录集的话，那么可以选择该类型库。

7.9.1 ADO 对象模型

ADO 对象模型定义了一组可以编程对象、可以实现 OLE DB 的几乎所有的功能。ADO 对象模型中包含了 7 个对象：

- Conenection 对象
- Command 对象
- Parameter 对象
- Recordset 对象
- Field 对象

- Property 对象
- Error 对象

ADO 对象模型中还包含了 3 个集合：

- Fields 集合
- Properties 集合
- Errors 集合

它们之间的关系如图 7.45 所示。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

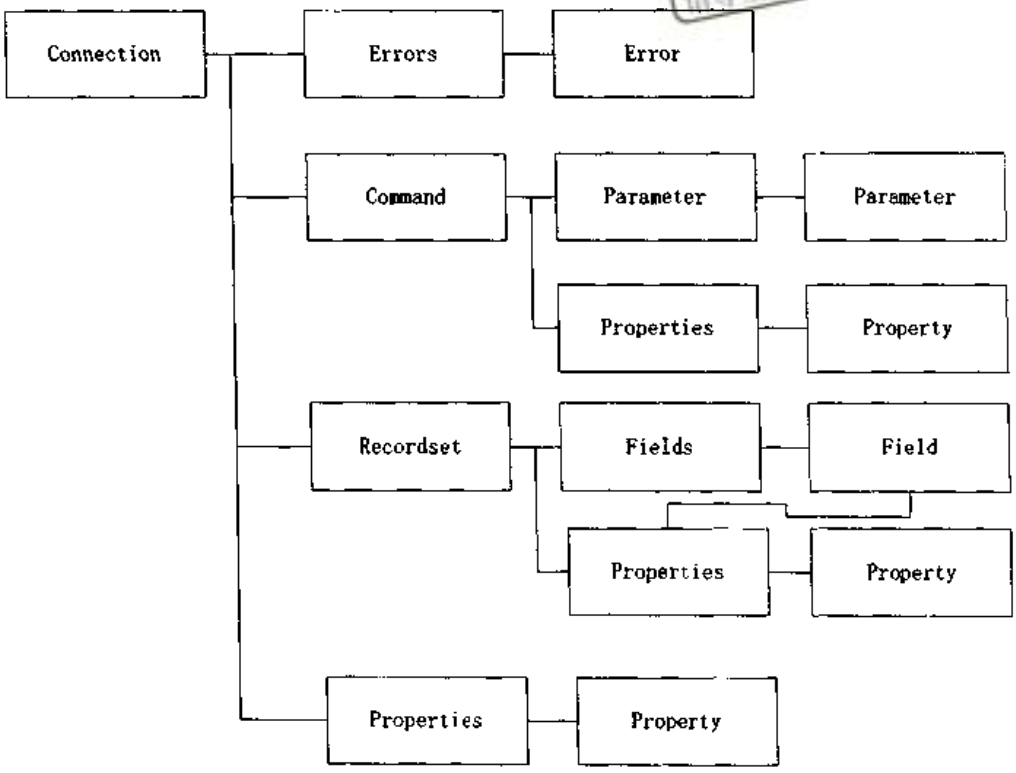


图 7.45　ADO 对象模型

1. Conenection 对象

Conenection 对象用来与数据源建立连接、执行查询以及建立事务处理。在建立连接之前，必须指定使用哪一个 OLE DB 供应者，如果 Provider 属性设为空串，那么连接采用缺省的 OLE DB 供应者 —— ODBC 供应者。也可以设置 Conenection 对象的 ConenectionString 属性来间接设置 Provider 属性。例如按下面语句设置 ConenectionString 属性后，OLE DB 供应者将会是 Microsoft OLE DB Provider For SQL Server。

```
Cn.connectionString = "Provider = SQLOLEDB.1 ;Persist Secrity Info=False; _
        User ID=dba;Initial Catalog =pubs ;Data Source=dbmaster"
```

而如果按照下面的语句的设置，OLE DB 供应者将会是缺省的 ODBC Provider：

```
Cn.ConnectionString ="driver={SQL Server};" & "server=dbmaster;uid=sa;pwd=pwd"
```

当设置好了 Connectionstring 属性以后，用 Conenection 对象的 Open 方法来与数据库建立连接。例如下面的代码用 ODBC 供应者与 SQL Server 数据库建立一个连接：

```
Dim cnn1 as ADODB.Connection= New ADODB.Connection
```

```
StrCnn="driver={SQL Server};server=srv;" & "uid=sa;pwd=;database=pubs"
Cnn1.Open strCnn
```

当建立连接以后可以用 Conenection 对象的 Execute 方法进行查询，包含 SQL 语句、存储过程或者 OLE DB 供应者，它从一个文本文件中返回数据，当把一个目录作为其数据源时，Execute 方法的返回结果将是文本文件 Custom.txt 中的数据。

```
Dim Cn as ADODB.Connection= New ADODB.Connection
Dim rs as ADODB.Recordset
Cn.provider="Myprovider"
Cn.Open "Pubs" , "sa"
Rs.Open "data source=c:\sdks\oledsdk\samples\sampclnt"
Rs=cn.Execute("customer.txt")
```

可以用 Conenection 进行事务处理。在事务处理方面，Conenection 对象提供了三个方法：BeginTrans、CommitTrans 和 RollbackTrans。在数据修改之前，用 BeginTrans 方法开始事务处理，当数据修改以后，可以用 CommitTrans 方法提交事务，或者用 RollbackTrans 方法回滚事务。例如：

```
Dim Cn as ADODB.Connection= New ADODB.Connection
Dim rs as ADODB.Recordset
Cn.provider="Myprovider"
Cn.Open "Pubs" , "sa"
Rs.Open("Select * from titles", Cn, ADODB.CursorTypeEnum.adOpenDynamic,
ADODB.LockTypeEnum.adLockPessimistic)
Cn.BeginTrans
'在这里进行数据修改
Cn.CommiTrans
Cn.Close
```

2. Command 对象

Command 对象执行数据库操作命令，这些命令并不只限于查询串，而是依赖于数据库的 OLE DB 供应者。以下的讨论假定使用 ODBC 供应者。

使用 Command 对象可以建立一个新的连接，也可以使用当前已经建立的连接，这取决于对象“ActiveConection”属性的设置。如果“ActiveConection”属性被设置为一个 Connection 对象的引用，那么 Command 对象就建立一个新的连接，并使用这个新连接。每一个 Connection 对象可以包含多个 Command 对象。

用 Command 对象执行一个查询子串，可以返回一个记录集，也可以返回多个记录集，甚至可以不返回记录集。对象的“CommandText”属性中包含了要执行的查询字串。一个查询可以是一个标准的 SQL 数据操作语言，如 SELECT、DELETE、UPDATE，等；也可以是任何数据定义语言，如 CREATE、DROP 等；还可以是一个存储过程或一个表。究竟 CommandText 中是哪一种查询字串，由对象的“CommandType”属性来决定。“CommandType”属性有四种不同的值：adCmdText、adCmdTable、adCmdStoreProc 和 adCmdUnknown。如果查询字串是一个 SQL 语句，那么“CommandType”属性应设为“adCmdText”；如果查询字串是一个存储过程的名字，那么“CommandType”属性应该为

“adCmdStoreProc”，Command 对象用{call procedure=name}的形式来执行一个存储过程；如果查询字串是一个表名，那么“CommandType”属性应该设为“adCmdTable”，Command 对象用“Select * from TableName”的形式来执行一个查询；如果“CommandType”属性被设置为“adCmdUnknown”，Command 对象必须执行一些额外的步骤来决定查询字串的类型，这样会降低系统的性能。

如果要执行一个带参数的查询，或者要执行一个查询若干次，那么可以用 Command 对象的“Prepared”属性预先建立一个查询字串。

假如有这样一个 SQL Server 的储存过程：

```
drop proc Proc1
go
create proc Proc1 as
create table #student (id int not NULL,name char(10)NOT NULL)
insert into #student value(1, 'Liming')
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

那么可以用以下代码来执行这个过程

```
Dim Cmd as ADODB.Command= New ADODB.Command
Cmd.ActiveConnection="DSN=pubs;UID=sa"
Cmd.Commandtext="Proc1"
Cmd.CommandTimeout=15
Cmd.CommandType=adComStoredProc
Cmd.Execute
```

3. Parameters 集合和 Parameter 对象

Parameters 集合和 Parameter 对象为 Command 对象提供参数信息和数据。当且仅当 Command 对象执行的查询是一个带参数的查询时，Parameters 集合和 Parameter 对象才有用，Parameter 对象包含在 Parameters 集合中。Parameter 对象中可以包含 4 种类型的参数：输入、输出、输入输出和返回值类型。

用 Command 对象的 CreateParameter 方法来创建一个 Parameter 对象，CreateParameter 方法的语法如下，5 个参数分别是：参数名、参数的数据类型、参数类型（输入输出等）、参数的长度和参数的值。

```
Parameter=command.CreateParameter(name,type,Direction,Size,Value)
```

假设有这样一个带参数的 SQL Server 存储过程：

```
drop proc ParaProc1
go
create proc ParaProc1
(@type char(12))
as
select * from titles where type =@type
```

那么可以用下面的代码来执行这个带参数的存储过程：

```
Dim cmd As ADODB.Command = New ADODB.Command()
Dim prm As ADODB.parameter
cmd.ActiveConnection = "DSN=pubs;uid=sa"
```

```
cmd.CommandText = "ParaProc1"
cmd.Commandtype = ADODB.CommandTypeEnum.adCmdStoredProc
cmd.CommandTimeout = 15
prm=cmd.CreateParameter("Type",ADODB.DataTypeEnum.adChar,
ADODB.ParameterDirectionEnum.adParamInput, 6, "Business")
        cmd.Parameters.Append(prm)
cmd.Execute()
```

4. Recorset 对象

Recorset 对象用来操作查询返回的结果集，它可以在结果集中添加、删除、修改和移动记录。当创建了一个 Recorset 对象时，一个游标也被自动创建了。可以用 Recordset 对象的 CursorType 属性来设置游标的类型。游标的类型有 4 种：仅能向前移动的游标、静态的游标、键集游标和动态游标。这已经在 DAO 中的 Recordset 对象和 RDO 的 rdoResultset 对象中介绍过了。

下面这个例子建立一个 Recordset，并循环打印出结果集中的记录的第一个字段的值。

```
Dim rs As ADODB.Recordset = New ADODB.Recordset()
Rs.Open("select * from titles", "DSN=pubs;UID=sa")
  While (Not rs.EOF)
        Debug.Write(rs(0))
        Rs.MoveNext()
        End While
Rs.close()
```

5. Fields 集合和 Field 对象

Fields 集合和 Field 对象用来访问当前记录中的每一列的数据。可以用 Field 对象创建一个新记录、修改已存在的数据等。用 Recordset 对象的 AddNew、Update 和 UpdateBatch 方法来添加新记录和更新记录。也可以用 Field 对象来访问表中每一个字段的一些属性，如下面的例子循环打印出表中当前记录的字段名称、类型和值。

```
Dim rs As ADODB.RecordSet = New ADODB.RecordSet()
Dim fld As ADODB.Field
Dim flds As ADODB.Fields
Rs.Open("select * from titles", "DSN=pubs;UID=sa")
Flds = rs.Fields
Dim TotalCount As Long
TotalCount = flds.Count
Debug.Write("Total:" & TotalCount)
For Each fld In Flds
Debug.Write(fld.Name)
Debug.Write(fld.Type)
Debug.Write(fld.Value)
Next
Rs.Close()
```

6. Properties 集合和 Property 对象

ADO 对象有两种类型的属性：一种是内置的，另一种是动态的。内置的属性不出现在对象的 Properties 集合中。而动态的属性是由 OLE DB 供应者定义的，它们出现在相应的 ADO 对象的 Properties 集合中。Connection、Command、Recordset 和 Field 对象包含有 Properties 集合，Properties 集合中包含了 Property 对象，它们负责提供四个对象的特征信息。Property 对象只有一个“Attributes”属性，这个属性描述了某个特定属性是否被 OLE DB 供应者支持，或者是否必须赋值，或者是否能读写。Property 对象的另外三个属性为“Name”、“Type”和“Value”，分别表示 Property 对象所描述的 ADO 对象的属性的名称、数据类型和属性值。

下面这个例子取得 Connection 对象的“ConnectTimeout”属性和 Command 对象的“CommandTimeout”属性。

使用本复制品
请尊重相关知识产权!

```
Dim cn As ADODB.Connection = New ADODB.Connection()
Dim cmd As ADODB.Command = New ADODB.Command()
Dim rs As ADODB.Recordset = New ADODB.Recordset()
Cn.Open("pubs", "sa")
Debug.Write(Cn.properties("Connect Timeout"))
Cmd.ActiveConnection = cn
Cmd.CommandType = adCmdtable
Rs = cmd.Execute()
Debug.Write(Cmd.Properties("Command Timeout"))
```

7.10　ADO.NET

7.10.1　ADO.NET 简介

ADO.NET 是以几年前 Open Database Connectivity（ODBC）应用程序技术的使用为标志的数据库访问技术发展以来最新的一项技术。正是这样，一些有意思的事情发生了。COM 技术开始涉及数据库领域，而且和 OLE DB 一起达到了一个顶峰状态。后来，ActiveX Data Objects（ADO），一个粗略的也算 OLE DB 自动版本的对象，被选择用在基于 Windows 的数据库开发者的 Visual Basic 和 ASP 的共同使用对象。

现在拥有了.NET，微软提供了一个普及的通用的框架 —— 框架类库，该类库将跨越所有的存在的 Windows API 函数，特别的，它将包含一些经常使用的库，而且用户会发现 XML 和 ADO 对象模型被集成在一个树状的类的集合中，这个集合就叫做 ADO.NET。

不像 ADO 那样，ADO.NET 被设计成为遵循一般的更没有面向数据库缺陷的数据库访问准则，ADO.NET 搜集了所有的和数据访问有关的类，这些类由一些数据容器对象组成，这项对象具有一般的数据处理能力 —— indexing，sorting，Viewing，ADO.NET 是为.NET 数据库应用程序定义的开发办法，ADO.NET 是一个对数据库的整套设计环境，而不是像 ADO 那样只是围绕着数据访问和数据处理的。

ADO.NET 和 ADO 大不一样，它是一个新的数据访问程序模型，需要透彻的理解。然

而，一旦用户使用了 ADO.NET，将会发现所有的 ADO 的技巧，对用户在 ADO.NET 环境中编程是大有帮助的。

一个 ADO.NET 应用程序需要先建立一个连接对象，从数据源中读取数据。它可以是 SQLConnection 或者 ADOConnection，这取决于提供者（Provider）的环境，但是用户需要记住的是，最好还是用 SQL ServerOLE DB 提供者来访问 SQLServer 数据库，用户也可以用 ADO.NET 的类来连接 SQL Server 数据库，惟一的缺点是用户的代码必须要从一层不必要的代码层传递过来，它将调用 ADO 的提供者，而后又调用 SQL ServerOLE DB 提供者。但是如果用户直接使用 SQL Server 提供者的话，只需要直接访问数据就行了。还有一个 ADO.NET 连接对象和 ADO 的区别就是 ADO.NET 连接对象不提供“CursorLocation”属性。在 ADO 中通过指定一个连接和一个数据库访问命令可以建立一个 Recordset 对象。

7.10.2 使用 ADO.NET 的基本方法

ADO 作为 Visual Basic.NET 中的一个工具，具有相当多的相关的对象，先把一些对象的概念说明一下：

（1）ADOConnection：这个对象表示对一个数据库的一个确切的连接，用户可以通过打开和关闭一个连接来使用这个对象的一个实例。

（2）ADOCommand：这个类表示了一条用户将要对一个数据库进行的 SQL 命令，它可以返回值，也可以返回一个空值。

（3）ADODataSetCommand：为数据在一个数据库或一个本地的数据集中的传递提供了桥梁。

（4）DataSet：代表了一个或者多个数据库的表，或者是从本地的数据库中经过查询以后返回的一个数据的集合。

（5）DataTable：从一个数据库或者一条查询中返回的一个单独的表。

（6）DataRow：在 DataTable 中的单独的一列。

1. 连接一个数据库

为了连接一个数据库，用户可以专门为该数据库建立一个关于连接信息的字符串。例如要访问一个 Access 数据库，连接的字符串可以写成：

```
connection = "Provider=Microsoft.Jet.OLEDB.4.0;" + _
"Data Source=" + dbname
```

具体的实现方法是：

```
Dim Adc as ADOConnection
Adc = New ADOConnection(connection)
```

2. 从一个数据表中读取数据

为了从一个数据表中读取数据，用户可以建立一个 ADOCommand，并含有具有合适的参数的 SQL 表达式和连接。

```
Public Function openTable(ByVal tbName As String) _
As DataTable
'create the dataset command connection
Dim dsCmd As New ADODataSetCommand()
```

```
'put the query into the dataset command
Dim query As String = "Select * from " & tbname
dsCmd.SelectCommand = New ADOCommand(query, adc)
```

然后建立一个数据集对象用来接纳访问的结果：

```
'create the destination dataset
Dim dset As New DataSet()
```

接着用户只要简单地告诉 Command 对象通过 Connection 加载 dataset 对象就行了。用户必须像下面的代码那样指明 Filldataset 方法操作的表名：

超星阅览器提示
使用本复制品
请尊重相关知识产权!

```
'open the connection and fill the table in the dataset
ADC.Open()
dsCmd.FillDataSet(dset, tbname)
```

则 DataSet 对象包含了至少一个表，此时可以通过索引或者名称来查看它的内容：

```
'get the table from the result dataset
Dim dtable As DataTable = dset.Tables(0)
adc.Close() 'close the connection
Return dtable 'return the table we read
End Function
```

3. 执行一条查询

执行一条 select 查询和上面的代码类似，只不过是在 SQL 查询语句上更加的复杂，如下所示的选择关系的代码：

```
Public Function openQuery(ByVal query As String) _
As DataTable
Dim dsCmd As New ADODataSetCommand()
Try
dsCmd.SelectCommand = New ADOCommand(query, ADC)
Dim dset As New DataSet()
ADC.Open()
dsCmd.FillDataSet(dset, "mine")
Dim dtable As DataTable = dset.Tables(0)
adc.Close()
Return dtable
Catch e As Exception
messagebox.show(e.Message)
End Try
End Function
```

4. 删除一个表里面的内容

用户可以使用“Delete * from Table”SQL 表达式来删除一个表的内容。然而，既然这里没有 select 表达式联系一个本地的表，所以用户可以使用 ADOCommand 对象的 ExecuteNonQuery 方法：

```
Public Sub delete()
```

```
adc.Open()
Dim adcmd As ADOCommand
adcmd = new ADOCommand("Delete * from " + tablename, adc)
Try
adcmd.ExecuteNonQuery()
adc.Close()
Catch e As Exception
Messagebox.show(e.Message)
End Try
End Sub
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

7.11 用面向对象思想处理数据库的例子

这里举这样一个例子，假设有三种商店在出售几种相同的蔬菜，商店和蔬菜的信息存放在一个 TEXT 文件里，现在，需要用到面向对象的知识和数据库访问的知识，建立几个类分别代表现实中的几个对象，来对数据库进行访问，得到最后的结果。

其中 TEXT 中的内容为：

```
Stop and Shop, Apples,          0.27
Stop and Shop, Oranges,         0.36
Stop and Shop, Hamburger,       1.98
Stop and Shop, Butter,          2.39
Stop and Shop, Milk,            1.98
Stop and Shop, Cola,            2.65
Stop and Shop, Green beans,     2.29
Village Market,     Apples,     0.29
Village Market,     Oranges,    0.29
Village Market,     Hamburger,      2.45
Village Market,     Butter,         2.99
Village Market,     Milk,           1.79
Village Market,     Cola,           3.79
Village Market,     Green beans,    2.19
Waldbaum's,   Apples,               0.33
Waldbaum's,   Oranges,              0.47
Waldbaum's,   Hamburger,           2.29
Waldbaum's,   Butter,              3.29
Waldbaum's,   Milk,                1.89
Waldbaum's,   Cola,                2.99
Waldbaum's,   Green beans,         1.99
```

界面如图 7.46 所示。

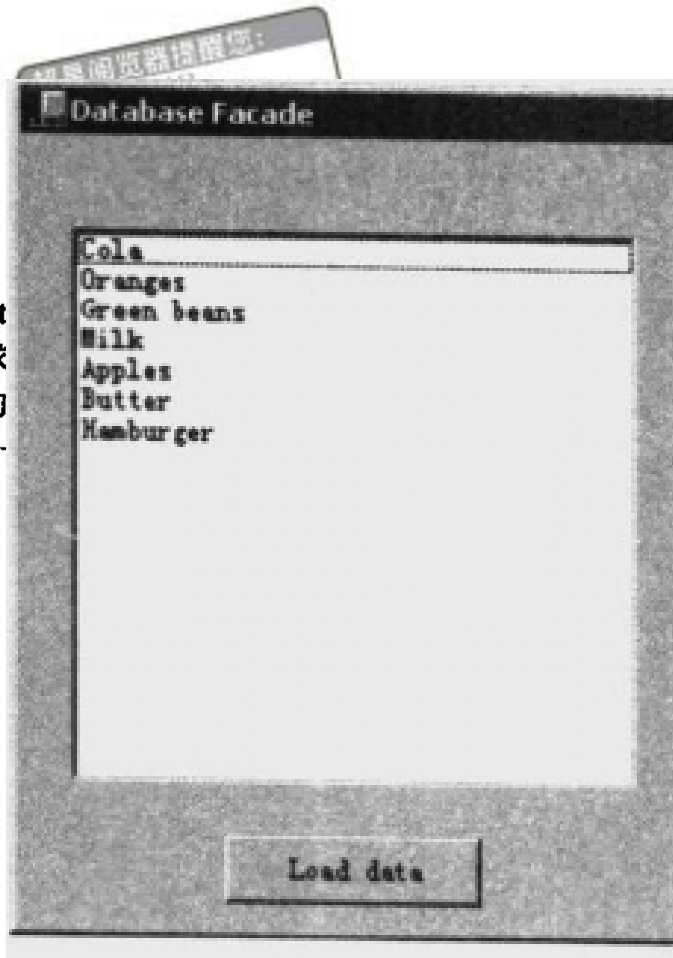


图 7.46　界面

要求用户在点击左边的蔬菜名时，在右边的 List
格。由于在 Text 文本中的信息是不系统的，这就要求
息，创建 Foods、Prices 和 Stores 三个表来系统有序的
根据面向对象的概念，用户可以把这三个表的一
建三个类分别来表现这三个对象：

（1）Foods 类

```
Public Class Foods
    Inherits DBTable
    '------
    Public Sub New(ByVal datab As DBase)
        MyBase.New(datab, "Foods")
    End Sub
    '-------
    Public Overloads Sub makeTable()
        MyBase.makeTable("FoodName")
    End Sub
    '------
    Public Overloads Function getValue() As String
        Return MyBase.getValue("FoodName")
    End Function
End Class
```

（2）Stores 类

```
Public Class Stores
    Inherits DBTable
```

```
        Public Sub New(ByVal datab As DBase)
            MyBase.New(datab, "Stores")
        End Sub
        Public Overloads Sub makeTable()
            MyBase.makeTable("StoreName")
        End Sub
    End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

（3）Prices 类

```
Imports System.Collections
Imports System.Data
Imports System.Data.ADO

Public Class Prices
    Inherits DBTable
    Private priceList As ArrayList
    Public Sub new(ByVal datab As DBase)
        MyBase.New(datab, "Prices")
        pricelist = New ArrayList()
    End Sub
    '------
    Public Sub addRow(ByVal storekey As Long, ByVal foodkey As Long, ByVal price As
Single)
        pricelist.Add(New StoreFoodPrice(storekey, foodkey, price))
    End Sub
    '------
    Public Overloads Sub makeTable()
        'stores current array list values in data table
        Dim adc As ADOConnection
        Dim cmd As adocommand
        Dim dset As New DataSet(tablename)
        Dim row As DataRow
        Dim fprice As StoreFoodPrice
        Dim dtable As New DataTable(tablename)
        dset.Tables.Add(dtable)
        adc = db.getConnection
        adc.Open()
        Dim adcmd As New ADODataSetCommand()
        'fill in price table
        adcmd.SelectCommand = New ADOCommand("Select * from " & tablename, adc)
```

```
            adcmd.TableMappings.Add("Table", tablename)
            adcmd.FillDataSet(dset, tablename)
            Dim ienum As IEnumerator = pricelist.GetEnumerator
            'add new price entries
            While ienum.MoveNext
                fprice = CType(ienum.Current, Storefoodprice)
                row = dtable.NewRow
                row("foodkey") = fprice.getFood
                row("storekey") = fprice.getStore
                row("price") = fprice.getPrice
                dtable.Rows.Add(row)        'add to table
            End While
            adcmd.Update(dset)          'send back to database
            adc.Close()
        End Sub
        '------
        Public Function getPrices(ByVal food As String) As DataTable
            Dim query As String
            query = "SELECT Stores.StoreName, Foods.Foodname, Prices.Price " & _
            "FROM (Prices INNER JOIN Foods ON Prices.Foodkey = Foods.Foodkey) INNER
JOIN Stores ON Prices.StoreKey = Stores.StoreKey " & _
            "WHERE(((Foods.Foodname) = """ & food & """)) " & _
            "ORDER BY Prices.Price;"
            Return db.openQuery(query)
        End Function
    End Class
```

（4）此外还有一些其他的起辅助作用的类

① DataLoader 类，用来对数据库进行加载：

```
    Public Class DataLoader
        Private vfile As vbFile
        Private stor As Stores
        Private fods As Foods
        Private price As Prices
        Private db As DBase
        '-------------
        Public Sub new(ByVal datab As DBase)
            db = datab
            stor = New Stores(db)     'create class instances
            fods = New Foods(db)
            price = New Prices(db)
```

```
    End Sub
    '--------------
    Public Sub load(ByVal datafile As String)
        Dim sline As String
        Dim storekey As Long, foodkey As Long
        Dim tok As StringTokenizer

        'delete current table contents
        stor.delete()
        fods.delete()
        price.delete()
        'now read in new ones
        vfile = New vbFile(datafile)
        vfile.OpenForRead()
        sline = vfile.readLine
        While (sline <> "")
            tok = New StringTokenizer(sline, ",")
            stor.addTableValue(tok.nextToken)     'store name
            fods.addTableValue(tok.nextToken)     'food name
            sline = vfile.readLine
        End While
        vfile.closeFile()
        'construct store and food tables
        stor.makeTable("StoreName")
        fods.makeTable("FoodName")
        vfile.OpenForRead()
        sline = vfile.readLine
        While (sline <> "")
            'get the gets and add to storefoodprice objects
            tok = New StringTokenizer(sline, ",")
            storekey = stor.getKey(tok.nextToken, "Storekey")
            foodkey = fods.getKey(tok.nextToken, "Foodkey")
            price.addRow(storekey, foodkey, tok.nextToken.ToSingle)
            sline = vfile.readLine
        End While
        'add all to price table
        price.makeTable()
        vfile.closeFile()
    End Sub
End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

② 对文件进行处理的类：

```
Imports System
Imports System.IO
Imports System.ComponentModel
Public class vbFile
'encapsulates the common read, write, exists and delete methods
'for text file manipulation
Private opened As Boolean     'true if file is open
Private end_file As Boolean 'true if at end of file
Private errDesc As String     'text of last error message
Private File_name As String 'name of file
private fl as File
private ts as StreamReader
private tok as StringTokenizer
private tokLine as String
private fs as FileStream
private sw as StreamWriter
private errFlag as boolean
private lineLength as integer
private sep as String                'tokenizer separator
'-----
Public sub New(filename as String)
   MyBase.New
   file_name = filename
   fl = New File(file_name)
   tokLine = ""
   sep = ","
end Sub
Public Overloads Function OpenForRead() as Boolean
    OpenForRead = OpenForRead(file_name)
End Function
Public OverLoads Function OpenForRead(Filename As String) As Boolean
  file_name = Filename
  errFlag = false
  end_File = false
  tok = new StringTokenizer("")
try
  ts = fl.Opentext()
catch e as Exception
   errDesc = e.Message
```

```
    Console.writeline(errDesc)
    errFlag = true
  end Try
  openForRead = errFlag
  End Function
  Public Function readLine() As String
   dim s as string
   try
     s = ts.readLine
     lineLength = s.length
   catch e as Exception
     end_file =true
     s=""
   'finally
     'readLine = s
     'return( s)
   end try
   return s
  End Function
  Public Function readToken() As String
    dim token as string
     token = tok.nextToken
            If (token.length < 1) Then
                tokLine = ts.readLine
                tok = New StringTokenizer(tokLine, sep)
                token = tok.nextToken
            End If
            Return token
        End Function
        Public Sub closeFile()
            ts.close()
        End Sub
        Public Function exists() As Boolean
            exists = fl.exists
        End Function
        Public Function getLastError() As String
            getlastError = errDesc
        End Function
        Public Function OpenForWrite(ByVal fname As String) As Boolean
            errFlag = False
```

超星阅览器提醒您:
使用本复制品
请尊重相关知识产权!

```
            Try
                file_name = fname
                fs = New FileStream(fname, FileMode.OpenOrCreate, FileAccess.Write)
                sw = New StreamWriter(fs)
            Catch e As Exception
                errDesc = e.Message
                errflag = True
            End Try
            openForWrite = errFlag
        End Function
        Public Sub writeText(ByVal s As String)
            sw.writeLine(s)
        End Sub
        Public Sub setFilename(ByVal fname As String)
            file_name = fname
        End Sub
        Public Function getFilename() As String
            getFilename = file_name
        End Function
        Public Function fEOF() As Boolean
            fEOF = end_file
        End Function
    End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

③ 描述数据库的类 Dbase：这个类的作用主要是表示对数据库的连接，通过一些 Sql 语句来连接和打开数据库里面的表：

```
    Imports System.WinForms
    Imports System.Data.ADO
    Imports System.Data
    Public Class DBase
        Private adc As ADOConnection
        '-----
        Public Overloads Sub New(ByVal connect As String)
            Adc = New ADOConnection(connect)
        End Sub
        Public Overloads Sub New(ByVal dbName As String, ByVal connectionType As String)
            Dim connection As String
            connectiontype = connectiontype.ToLower
            Select Case connectionType
                Case "access"
                    connection = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
```

```
dbname
                Case "sqlserver"
                    connection = "Persist Security Info = False;" & _
                    "Initial Catalog =" & dbname & ";" & _
                    "Data Source = myDataServer;User ID = myName;" & _
                    "password="

                Case Else
                    connection = dbname
            End Select
            Adc = New ADOConnection(connection)
        End Sub
        '-----
        Public Overloads Sub New(ByVal dbname As String, ByVal userid As String, _
            ByVal servername As String, ByVal password As String, _
            ByVal connectionType As String)
            Dim connection As String
            connectiontype = connectiontype.ToLower
            Select Case connectionType
                Case "sqlserver"
                    connection = "Persist Security Info = False;" & _
                    "Initial Catalog =" & dbname & ";" & _
                    "Data Source =" & servername & ";" & _
                    "User ID =" & userid & ";" & _
                    "password=" & password
                Case Else
                    connection = dbname
            End Select
            Adc = New ADOConnection(connection)
        End Sub
        '-------
        Public Function makeTable(ByVal nm As String) As Indexer
            'none of this works so far in ADO
        End Function
        '-------
        Public Function openTable(ByVal tbName As String) As DataTable
            'create the dataset command connection
            Dim dsCmd As New ADODataSetCommand()
            'put the query into the dataset command
            Dim query As String = "Select * from " & tbname
```

```
        dsCmd.SelectCommand = New ADOCommand(query, adc)
        'create the destination dataset
        Dim dset As New DataSet()
        'open the connection and fill the table in the dataset
        ADC.Open()
        dsCmd.FillDataSet(dset, tbname)
        'get the table from the result dataset
        Dim dtable As DataTable = dset.Tables(0)
        adc.Close() 'close the connection
        Return dtable     'return the table we read
    End Function
    '-------
    Public Function openQuery(ByVal query As String) As DataTable
        Dim dsCmd As New ADODataSetCommand()
        Try
            dsCmd.SelectCommand = New ADOCommand(query, ADC)
            Dim dset As New DataSet()
            ADC.Open()
            dsCmd.FillDataSet(dset, "mine")
            Dim dtable As DataTable = dset.Tables(0)
            adc.Close()
            Return dtable
        Catch e As Exception
            messagebox.show(e.Message)
        End Try
    End Function
    Public Function getConnection() As ADOConnection
        Return adc
    End Function
End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

④ 表示数据库中表的类 DBTable，这个类的作用是打开、加载和更新一个数据库中的单独的一个表：

```
Imports System.Collections
Imports System.Data
Imports System.Data.ADO
Imports System.WinForms
Public Class DBTable
    Private names As Hashtable
    Protected db As DBase
    Protected tableName As String
```

```
Private index As Integer
Private dtable As DataTable
Private filled As Boolean
Private columnName As String
Private rowIndex As Integer
Private opened As Boolean
Dim adc As ADOConnection
Dim cmd As adocommand
Dim dset As DataSet
Dim row As DataRow
Public Sub New(ByVal datab As DBase, ByVal tname As String)
    names = New Hashtable()
    db = datab
    tablename = tname
    index = 1
    filled = False
    opened = False
End Sub
'-------
Public Sub createTable()
    'not yet available in ADO
End Sub
'-------
Public Function hasMoreElements() As Boolean
    If opened Then
        Return (rowindex < dtable.Rows.Count)
    else
        Return False
    End If
End Function
'-------
Public Function getKey(ByVal nm As String, ByVal keyname As String) As Long
    Dim i, key As Integer
    Dim dtable As DataTable
    Dim row As DataRow
    If Not filled Then
        Return CType(names.Item(nm), Integer)
    Else
        Dim query As String
        query = "select * from " & tablename & " where " & columnname & "='" & nm
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
& "'"
            dtable = db.openQuery(query)
            row = dtable.Rows(0)
            key = row(keyname).ToString.ToInt32
            Return key
        End If
    End Function
    '-------
    Public Function getValue(ByVal columnName As String) As String
        'returns the next name in the table
        'assuems that openTable has already been called
        If opened Then
            Dim row As DataRow
            row = dtable.rows(rowindex)
            rowindex = rowindex + 1
            Return row(columnName).ToString
        Else
            Return ""
        End If
    End Function
    '-------
    Public Sub openTable()
        dtable = db.openTable(tablename)
        rowindex = 0
        opened = True
    End Sub
    Public Overridable Sub addTableValue(ByVal nm As String)
        'accumulates names in hash table
        Try
            names.Add(nm, index)
            index = index + 1
        Catch e As ArgumentException
            'do not allow duplicate names to be added
        End Try
    End Sub
    '-------
    Public Overridable Sub makeTable(ByVal colName As String)
        'stores current hash table values in data table
        dset = New DataSet(tablename)     'create the data set
        columnName = colname
```

```
    Dim name As String
    dtable = New DataTable(tablename)     'and a datatable
    dset.Tables.Add(dtable)               'add to collection
    adc = db.getConnection
    adc.Open()                            'open the connection
    Dim adcmd As New ADODataSetCommand()
    'open the table
    adcmd.SelectCommand = _
        New ADOCommand("Select * from " & tablename, adc)
    adcmd.TableMappings.Add("Table", tablename)
    'loac curren data into the local table copy
    adcmd.FillDataSet(dset, tablename)
    'get the Enumerator from ther Hashtable
    Dim ienum As IEnumerator = names.Keys.GetEnumerator
    'move through the table, adding the names to new rows
    While ienum.MoveNext
        name = CType(ienum.Current, String)
        row = dtable.NewRow          'get new rows
        row(colname) = name
        dtable.Rows.Add(row)         'add into table
    End While
    'Now update the database with this table
    Try
        adcmd.Update(dset)
        adc.Close()
        filled = True
    Catch e As Exception
        Messagebox.show(e.Message)
    End Try
End Sub
'------
Public Sub delete()
    'deletes entire table
    adc = db.getConnection
    adc.Open()
    Dim adcmd As New ADOCommand("Delete * from " & tablename, adc)
    Try
        adcmd.ExecuteNonQuery()
        adc.Close()
    Catch e As Exception
```

```
                Messagebox.show(e.Message)
            End Try
        End Sub
    End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

⑤ 在数据库中创建表的类 Indexer：

```
Imports system.WinForms
Imports System.Data.ADO
Imports System.Data
Public Class Indexer
    Private db As DBase
    Private tableName As String
    Private dtable As DataTable
    'creates data tables - none of this works in ADO in VB.Net so far
    Public Sub new(ByVal datab As DBase)
        db = datab
    End Sub
    Public Sub makeTable(ByVal name As String)
        'starts creation of table
    End Sub
    Public Sub openTable(ByVal name As String)
        tableName = name
    End Sub
    Public Sub addTable()
    End Sub
    Public Sub createKey(ByVal keyName As String)
    End Sub
    'creates text column
    Public Overloads Sub createColumn(ByVal colName As String, ByVal length As
Integer)
    End Sub
    'creates single precision table columne
    Public Overloads Sub createColumn(ByVal colName As String)
    End Sub
    Public Sub makeIndex(ByVal name As String, ByVal primary As Boolean)
    End Sub
End Class
```

⑥ 描述 TEXT 信息的类 StoreFoodPrice：

```
Public Class StoreFoodPrice
    Private storeKey, foodKey As Long
    Private foodPrice As Single
```

```
    Public Sub new(ByVal storeky As Long, ByVal foodky As Long, ByVal fprice As Single)
        storekey = storeky
        foodkey = foodky
        foodprice = fprice
    End Sub
    Public Function getStore() As Long
        Return storekey
    End Function
    Public Function getFood() As Long
        Return foodKey
    End Function
    Public Function getPrice() As Single
        Return foodprice
    End Function
End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

类 Tokenizer：

```
Imports System
Public class StringTokenizer
Private s As String
private i As Integer
Private sep As String     'token separator
'------
Public Overloads Sub New(st As String)
  MyBase.New
  s = st              'copy in string
  sep = " "           'default separator
End Sub
'------
Public Overloads Sub New(st as String, sepr as String)
   MyBase.New
   s = st
   sep = sepr
End Sub
'------
Public Sub setSeparator(ByVal sp As String)
   sep = sp          'copy separator
End Sub
'------
Public Function nextToken() As String
```

```
Dim tok as String
try
i = s.indexOf(sep)     'look for occurrence of separator

If i > 0 Then          'if found
   tok = s.substring(0, i ).trim                'return string to left
   s = s.substring(i+1).trim    'shorten string
Else
   tok = s             'otherwise return end of string
   s = ""               'and set remainder to zero length
End If
catch e As Exception
   tok = ""
end try
return tok
End Function
      Public Function hasMoreElements() As Boolean
           Return s.Length > 0
      End Function
End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

窗体中的代码：

```
Imports System.ComponentModel
Imports System.Drawing
Imports System.WinForms
Public Class FoodPrices
      Inherits System.WinForms.Form
      Private db As DBase
      Private shops As Stores
      Private prc As prices
      '-----
      Public Sub New()
           MyBase.New()
           Dim i As Integer
           Form1 = Me
           InitializeComponent()
           db = New DBase("newgroc.mdb", "Access")
           shops = New Stores(db)
           prc = New Prices(db)
           Soadfoodtable()
```

```
        End Sub
        '-----
        Private Sub loadFoodTable()
            Dim fods As New Foods(db)
            fods.openTable()
            While fods.hasMoreElements
                lsfoods.Items.Add(fods.getValue)
            End While
        End Sub
        '-----
        'Form overrides dispose to clean up the component list.
        Public Overrides Sub Dispose()
            MyBase.Dispose()
            components.Dispose()
        End Sub
    #Region " Windows Form Designer generated code "
        'Required by the Windows Form Designer
        Private components As System.ComponentModel.Container
        Private WithEvents btLoad As System.WinForms.Button
        Private WithEvents lsPrices As System.WinForms.ListBox
        Private WithEvents lsFoods As System.WinForms.ListBox
        Dim WithEvents Form1 As System.WinForms.Form
        'NOTE: The following procedure is required by the Windows Form Designer
        'It can be modified using the Windows Form Designer.
        'Do not modify it using the code editor.
        Private Sub InitializeComponent()
            Me.components = New System.ComponentModel.Container()
            Me.lsFoods = New System.WinForms.ListBox()
            Me.lsPrices = New System.WinForms.ListBox()
            Me.btLoad = New System.WinForms.Button()
            '@design Me.TrayHeight = 0
            '@design Me.TrayLargeIcon = False
            '@design Me.TrayAutoArrange = True
            lsFoods.Location = New System.Drawing.Point(16, 32)
            lsFoods.Size = New System.Drawing.Size(176, 199)
            lsFoods.TabIndex = 0
            lsPrices.Location = New System.Drawing.Point(248, 32)
            lsPrices.Size = New System.Drawing.Size(168, 199)
            lsPrices.TabIndex = 1
            btLoad.Location = New System.Drawing.Point(64, 240)
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
            btLoad.Size = New System.Drawing.Size(80, 24)
            btLoad.TabIndex = 2
            btLoad.Text = "Load data"
            Me.Text = "Database Facade"
            Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
            Me.ClientSize = New System.Drawing.Size(440, 273)
            Me.Controls.Add(btLoad)
            Me.Controls.Add(lsPrices)
            Me.Controls.Add(lsFoods)
        End Sub
    #End Region
        '-----
        Protected Sub btLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
            Dim dload As New DataLoader(db)
            dload.load("groceries.txt")
            loadfoodtable()
        End Sub
        '-----
        Protected Sub lsFoods_SelectedIndexChanged(ByVal sender As Object, ByVal e As
System.EventArgs)
            Dim food As String = lsfoods.Text
            Dim dtable As DataTable = prc.getPrices(food)
            Dim rw As datarow
            lsprices.Items.Clear()
            For Each rw In dtable.Rows
                lsprices.Items.Add(rw("StoreName").tostring & " " & rw("Price").tostring)
            Next
        End Sub
    End Class
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

运行结果如图 7.47 所示，用户在点击左边的蔬菜名时，在右边的 List 能够显示在不同的商店里该蔬菜的价格。

图 7.47　运行结果

第 8 章　Visual Basic.NET 的多线程

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

本章包括：
自由线程
自由线程的举例

8.1　自由线程（Free Threading）

Visual Studio.NET 允许用户开发多条互不相干的多线程的应用程序。也就是说，当用户在执行一个程序的时候，也可以在另外一条的线程独立地运行另外一个任务程序，这种过程被称为自由线程（free threading）。对于用户而言，自由线程概念的引入，使得应用程序对用户的响应将更加积极，因为任务处理器能够在某一个线程正在运行时，依然保持界面对用户的响应，只要对用户响应的线程正在运行。而且，自由线程在运行大型的程序的时候，也将会很有用，因为随着运行任务的增加，用户可以多开辟几个线程来运行程序。

8.1.1　建立和使用一个新的线程

用户可以通过声明一个变量类型 System.Threading 来建立一个新的线程。并且还提供了一个 AddressOf 操作和一个用户想运行的过程或方法。例如：

Dim mythread as New System.Threading.thread(AddressOf MySub)

用户可以使用“Start”方法来开始一个线程，例如 thread.Start()。用“Stop”方法来终止一个线程。

8.1.2　关于线程的参数和返回值的问题

给一个过程建立一个新的线程是一个简单的问题，但是还是有一些重要的问题需要强调一下，例如当运行一个新的线程的时候，用 AddressOf 操作运行的方法或过程并没有注释和参数，而且不能返回值，为了解决这个问题，下面有几种比较简单的方法，可以给线程设定一些参数，并且从一些独立的线程的过程中返回值。

1. 为新建立的线程设置参数

因为在建立一个新的线程类的时候，只使用了一个 AddressOf 操作，这样就不能给一个即将要独立运行的线程设置一些具有标示性的议论。

一个给线程设置参数的方法就是使用全局变量，采用这种结构后，用户就能区别线程所使用的一些参数和注释，不过这样做，也有一些不足的地方，因为当用户在调试程序的时

候，全局变量会带来困难，而且要维护代码也是一件比较困难的事，而且当使用多个独立线程中对一个过程的拷贝时，事情也将会变得很复杂。

最好的办法就是把一些需要对自由线程中的方法设置的参数设置成为一个新的线程的属性的值。这样做的优点就是每当用户新建一个新的线程对象的时候，这个对象可以有自己的属性值，也就是说，它可以具有它自己的参数。

2. 从自由线程的过程中返回参数

因为当用户运行一个独立的线程的时候，在线程中只能是过程而不能是函数，也不能用 ByRef 来引用，所以用户必须通过一些其他的技术来从线程中得到所需要的信息到自己的程序中去。

最简单的办法就是通过设置全局变量，当某个线程结束以后就给某个全局变量赋值，然而，正如上述这种方法是应该要避免的，而且如果要这样做的话，用户还必须事先判断线程是不是已经结束了，才能去访问那些全局变量，这样的话，无疑降低了程序的效率和准确性。

从一个线程的过程中返回值的最好办法就是，把需要返回的值作为一个参数建立一个事件，通过执行这个事件，就可以得到所需要的返回值。如果事件运行以后，同时用户所需要的返回值是某个全局变量的话，可以干脆把这个事件设置为返回该全局变量。当线程结束以后，主程序就可以利用那些事件返回的值进行下一步的操作。

8.1.3　并发性

当用户在写多线程程序的时候，可能最具有挑战性的就是独立的线程和程序的其他的部分有并发性。例如，程序中的一个独立的线程正在处理一系列的名称，而程序的另一个部分必须等待这个线程完成以后才能进行，通常处理并发性的线程的办法就是判断线程的状态或者通过一个事件去标示和得到一个特征值。

最简单的、也是效率最低的方法就是判断一个线程的 IsAlive 属性的值，当一个线程在开始（Start）以前它的属性 IsAlive 的值是“False”，而在它运行的过程中，它的属性 IsAlive 的值是“True”。这种方法并没有通过一个事件去标示和得到一个特征值可靠，而且如果这样做的话，也失去了自由线程的很多优点和便利之处。例如：

```
While anotherthread.IsAlive =False
    '一直等待到该线程开始
End While
    '线程已经开始，现在等待它结束
'While anotherthread.IsAlive = True
    '等待一直等待到线程结束
End While
    '作一些线程结束以后的事情
```

一个更好的处理线程并发性的办法就是使用事件，每一个线程都能够运行一个事件把它自己的状态以一个标示的形式给主程序一个值，或者是给其他一个线程一个具有标示自身状态的一个值，这样当多个线程在运行一个过程的多个拷贝的时候，这些线程都具有自己特有的事件来标示各自的状态，则在线程之外的程序的其他的部分，就可以通过运行这些事件，

得到线程的信息，并且进行下一步的信息的处理。例如：

Public Event Status (Byval ThreadStatus As Integer, Byval ThreadID As Integer)

Visual Basic.NET 提供了 SyncLock 表述来使得在一个 expression 中一些 statements 可以具有并发性，这保证了多个线程在运行时不在同一时间里，运行相同的 statements，当进入了 SyncLock 模块，共享的方法 System.Monitor.Enter 将在特定的 expression 下运行，一直到这个特定的线程对 expression 返回的对象有 exclusive lock 以后,模块中的代码才会停止运行。注意 expression 的类型必须是可以引用的那种类型。

使用本资料请尊重相关知识产权

补充：SyncLock 表述

语法：

SyncLock expression

...[Block]

End SyncLock

参数：

expression

必需的参数，一个单独的操作和值的集合，并能得到惟一的值。

功能：

SyncLock 表述，能够确保多线程不在同一个时间里，运行相同的 statements，当一个线程运行到一个 SyncLock 模块，并不是马上就执行模块里的代码，而是等到对 expression 返回的对象有锁定（lock）的权力。这使得 expression 在几个线程运行的时候改变了值，以至产生一些不可预料的错误。

注意：expression 的类型必须是引用类型的，如一个类、一个模块、一个接口等等。

8.2　自由线程的举例

下面的例子说明了建立一个独立的线程在一个 File 中查找一个 Word。为了运行这个例子，新建一个 Visual Basic.NET 工程，并且把下列控件加入到主窗体中。

（1）新建一个 Visual Basic.NET Windows 应用程序工程，并且在主窗口中加入两个按钮和 4 个文本框，如图 8.1 所示。

图 8.1　多线程演示

（2）按照表 8.1 中的值，对控件的属性值进行设置：

表 8.1　控件属性值

对　象	属　性	设　置
Class	Name	Words
First Button	Name	Start
Second Button	Name	Cancel
First TextBox	Name Text	SourceFile “”
Second TextBox	Name Text	CompareString “”
Third TextBox	Name Text	WordCount "0"
Fourth TextBox	Name Text	LinesCounted "0"

（3）在 Project 菜单，选择 Add Class 给工程添加一个类，并且取名为 Words。

（4）给 Words 类添加如下代码：

```
Private strloc As Integer
Private RemainingString As String
Private SourceString As String
Public SourceFile As String
Public CompareString As String
Public wordCount As Integer
Private LinesCounted As Integer = 0
Public Event Status(ByVal LinesCounted As Integer)
Public Event FinishedCounting(ByVal NumberOfMatches As Integer)
Sub CountWords()
    Dim f As New System.IO.File(SourceFile)
    Dim mystream As System.IO.StreamReader
    Dim mystr As String = "" ' 初始化使该值不为空
    If SourceFile = system.string.empty _
        Or comparestring = system.string.empty Then
        System.WinForms.messagebox.Show("You must set the sourcefile " & _
            " and comparestring fields before calling CountWords")
        Exit Sub
    End If
    If f.FileExists(SourceFile) = False Then
        System.WinForms.MessageBox.Show("Error, Unable to open the input file")
        Exit Sub
    End If
    Try
        mystream = f.OpenText ' 打开一个新的 stream.
        ' 一直做到在文件结束时 stream 返回一个 Nothing.
```

```
            Do Until isnothing(mystr)
                mystr = mystream.ReadLine
                wordCount += CountInString(1, mystr, CompareString)
                LinesCounted += 1 ' 行数增加
                ' 调用一个可以显示进度的事件
                RaiseEvent Status(LinesCounted)
            Loop
        Catch eof As IO.EndOfStreamException
            ' 当到达 stream 结束时无需做任何事情。
        Catch IOExcep As IO.IOException
            ' 有错误发生
            MessageBox.Show(IOExcep.Message)
        Finally
            myStream.Close() ' 关闭文件
        End Try
        RaiseEvent FinishedCounting(wordcount)
    End Sub
    Private Function CountInString(ByVal StartingPoint As Integer, _
                                   ByVal SourceString As String, _
                                   ByVal CompareString As String) As Integer
        ' 这个函数计算了在一个句子中某个字符出现的次数。
        strloc = strings.InStr(StartingPoint, SourceString, CompareString)
        If strloc <> 0 Then
            CountInString += 1
            CountInString += CountInString(strloc + strings.len(comparestring),
        SourceString, CompareString)
        End If
    End Function
```

（5）线程中处理事件。

在主窗体中加入如下代码：

```
Sub FinishedCountingEventHandler(ByVal wordcount As Integer)
        ' 这个事件处理过程当 wordcounting 线程引发了一个' FinishedCounting 事件
        时发生，意味着整个文件被签出，参数 wordcount 是找到的匹配的字符数目。
        Me.WordCount.text = CStr(wordcount)
        MessageBox.Show("Finished counting words")
    End Sub
    Sub LineCountEventHandler(ByVal ` As Integer)
        ' 这个事件处理过程当 wordcounting 线程在从源文件中' 读取每一行时引
        发的一个事件
        Me.LinesCounted.Text = CStr(LinesCounted)
```

```
End Sub
```

（6）新建一个线程处理方法 Wordcount。

把下列代码加到窗体的 declarations。

```
Dim thread As System.Threading.Thread
```

把下列过程加到窗体中，并且在 Start 按钮点击时调用这个过程。

```
Sub StartThread()
        Dim WC As New Words()
        Me.WordCount.Text = "0"
        ' 设置一些对象的值，因为用户不能建立一个线程并把它传递到一个带有参数
        的方法中去
        wc.CompareString = Me.CompareString.Text ' 要查询的字符
        wc.SourceFile = Me.SourceFile.Text        ' 要查询的文件
        ' 把一个事件处理与新线程结束事件联系起来
        AddHandler wc.FinishedCounting, AddressOf FinishedCountingEventHandler
        ' 把事件处理与读完每一行所引起事件联系起来.
        AddHandler wc.Status, AddressOf LineCountEventHandler
        ' 添加一个新的线程
        thread = New system.Threading.thread(AddressOf wc.CountWords)
        ' 开始一个新的线程
        thread.Start()
End Sub
```

（7）把下列代码中的过程写入到窗体中去，并且在 Cancel 按钮的点击事件中调用该过程。

```
Protected Sub StopThread(ByVal thread As System.Threading.Thread)
        '   用户可以通过一个取消过程来停止一个线程
        thread.Stop()
End Sub
```

运行程序，如图 8.2 所示。

在第一个文本输入框中输入要搜索的 Word 文件的全路径名，第二个输入框显示要查询的字节，第三个输入框显示查询到的字节的个数，第四个输入框显示的是已经查询完毕的行数。

图 8.2　运行程序

第 9 章　VB.NET 的数组、文件和出错处理

本章包括:
数组
出错处理
文件处理
文件处理的出错处理
检测文件的结束
文件的静态方法

超星阅览器提醒您:
使用本复制品
请尊重相关知识产权!

在 Visual Basic.NET 中，在处理数组、文件的问题上和以前的 Visual Basic 版本有很大的不同，而在出错处理上就有完全的变化。所有的这些变化使得用户的编程比以前容易得多。

9.1　数　　组

9.1.1　数组（Arrays）

在 Visual Basic.NET 中，所有的数组都是以 0 为起始长度的，这和 Visual Basic 以前的版本完全不一样，在 Visual Basic 6.0 中，如果用户写下如下语句:

```
Dim x(10) As Single
```

可以假定 x 数组的元素从 1 到 10 的，即使它实际上总是包含第 0 个元素。换句话来说，x 数组实际上是含有 11 个元素。

在 Visual Basic.NET 中，这样的数组只含有 10 个元素，编号为 0 到 9。这使得处理数组的长度和编号的情况和 C、C++、C#和 Java 具有一致性。

```
Dim Max as Integer
Max=10
Dim x(Max)
For j = 0   to Max-1
      X(j) = j
Next j
```

从上面的例子，读者应该看出，就是数组的最后一个元素的编号，应该写成数组长度减去 1。

9.1.2　数组列表（ArrayLists）

既然现在数组的长度计数是基于 0 开始的，那么，VB 7.0 还引进了一个数组列表（ArrayList）对象代替原来的集合（Collection）对象，集合对象的长度计数总是从 1 开始的，而且在需要的时候，数组列表长度可以是不定的。数组列表的基本的方法和集合一样，只不过它还具有一些新的功能方法。

```
Dim arl as New ArrayList   '构建一个数组列表
For j = 0 to 10
    Arl.Add(j)
Next j
```

所有的数组变量都有一个长度属性，这样用户就可以得知这个数组有多大：

```
Dim z(20) As single
Dim j As Integer
For j = 0 To z.Length –1
    Z(j) =j
Next j
```

在 Visual Basic.NET 中所有的数组都是动态的，用户可以在任何时候重新定义数组的长度，然而，在 Visual Basic.NET 中已经没有 ReDim Preserve 表述了，用户可以使用 New 关键字来对任何一个数组进行引用，并且重新定位：

```
'在类模块级声明
Dim x() As Single
'重新定位
x = New Single (20) {}
```

注意：在数组类型后面的大括号。

像集合对象一样，数组列表含有一个“Count”属性和一个“Item”属性，允许用户使用“index”来访问数组列表中的元素。而且，和集合一样，这个属性可以省略，就同数组一样：

```
For i = 0 to ar.Count –1
    Console.WriteLine(ar.Item(i))
    Console.writeLine(ar(i))
Next I
```

用户也可以使用表 9.1 列出的一些数组列表的方法。

表 9.1　数组列表的方法

Clear	把数组列表的内容清空
Contains(object)	如果数组列表含有该对象则返回 true
CopyTo(array)	把一个数组列表的拷贝到一个一维的数组中去
IndexOf(object)	返回第一个元素的值
Insert(index,object)	在指定的位置插入一个元素
Remove(object)	把一个元素从列表中删除
RemoveAt(index)	把一个指定位置的元素从列表中删除
Sort	对列表进行排序

9.2 出 错 处 理

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

9.2.1 出错处理

Visual Basic.NET 中的出错处理是使用 exceptions 而不是 On Error GoTo 语法来实现的，On Error GoTo 现在已经不支持了，Visual Basic 现在提供一个结构化的出错处理过程，使用 Try...Catch...Finally 表述。结构化的出错处理过程是和 exceptions 有关的，类似于 Select Case 或者 While 的一个具有选择关系的代码结构。这使得程序的代码更具有健壮性，可以进行更加复杂的错误处理。exceptions 的运行机理是这样的，当有错误出现时，错误处理过程就会屏蔽掉使得在 Try 模块中产生错误的运行代码，并且使用 Catch 表述来获取错误，语法如下：

```
Try
'表述代码（Statements）
Catch e as Exception
'在错误产生之后运行的代码
Finally
'必须运行的代码
End Try
```

首先，运行 Try 模块中的代码，如果没有错误的话就接着运行 Finally 模块里的代码，如果有错误的话，就运行 Catch 模块里的代码，在 Catch 模块里用户可以做一些出错处理，然后接着运行 Finally 模块里的代码并退出整个模块。

下面的例子就是显示怎样进行出错处理的设置，当用户访问一个数组列表时，超过列表的长度以后，就会产生一个错误，代码如下：

```
Try
    For I = 0 to ar.Count
        Console.write(ar.Item(i))
    Next I
Catch e As Exception
        Console.writeLine(e.Message)
        Console.writeLine(e.stackTrace)
End Try
Console.writeLine("end of loop")
```

上面的例子把错误信息打印出来，并且指出代码产生错误的位置，并且继续运行下去。下面就是运行的结果：

```
0123456789Index is out of range. Must be non-negative and less than size .
Parameter name :index
    At System.Collection.ArrayList.get_Item(Int32)
    At ArrayTest.Main()
End of loop
```

相反，如果用户不进行错误处理的话，将在运行的时候从系统得到错误的信息，并且直接退出运行环境，而不是继续运行下去。系统将会出现如下错误信息：

```
Exception occurred:System.ArgumentOutOfRangException:Index is Out of range.
Must be non-negative and less than size.
    At System.Collection.ArrayList.get_Item(Int32)
    At ArrayTest.Main()
    At vbproject._main(System.String[])
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

表 9.2 列出了一些基本的常用的错误类型。

表 9.2　常用的错误类型

AccessException	在访问一个类的方法或数据字段发生了错误
ArgumentException	方法的参数不匹配
ArgumentNullException	参数为 NULL
ArithmeticException	上溢或者下溢
DivideByZeroException	除数为 0
IndexOutOfRangeException	数组下标超出范围
FileNotFoundException	文件没有发现
EndOfStreamException	访问超出输入流
DirectoryNotFoundException	目录没有发现

9.2.2　多种出错处理（Multiple Exceptions）

用户也可以在一个 Try 模块中获取一系列的错误并且执行不同的处理，例如：

```
Try
For i = 0 To ar.Count
Dim k As Integer = CType(ar(i), Integer)
Console.writeLine(i.toString & " " & k / i)
Next i
Catch e As DivideByZeroException
Console.writeLine(e.Message)
Console.writeLine(e.stackTrace)
Catch e As IndexOutOfRangeException
Console.writeLine(e.Message)
Console.writeLine(e.stackTrace)
Catch e As Exception
Console.writeLine("general exception" + e.Message)
Console.writeLine(e.stackTrace)
End Try
```

在下面的例子中出错处理模块将能获取三种不同的错误信息，如果不同的错误产生了，Visual Basic 将会中断应用程序，执行一些相应的出错处理，并返回错误点。

```
Sub ErrorTest (x,y,z)
  ' Declare the variables.
```

```
Dim x, y, z As Single
' Place the code that may contain errors within a Try block.
Try
x = 1
y = 0
z = x / y
' Create the Catch blocks, which have code that fix potential errors.
' In this example, the error is caught within the first Catch block.
Catch e As DivideByZeroException
  MsgBox("You have attempted to divide by zero!")
  y = 2
  z = x / y
Catch e As OverflowException
  MsgBox("You have encountered an overflow exception.")
  y = 2
  z = x / y
Catch e As ConstraintException
  MsgBox("You have encountered a constraint exception.")
  y = 2
  z = x / y
' After the code is tested for errors (and fixed, if necessary),
' the Finally block runs, whether or not any errors occurred.
Finally
  MsgBox(x & "/" & y & " = " & z)
End Try
End Sub
```

下面的这个例子，主要是说明出错处理除了 IOException 或者 EndOfStreamException，都可以返回出错的接口，而 IOException 或者 EndOfStreamException 这种错误只能中断程序的执行。

```
Function GetStringsFromFile(ByVal FileName As String)
  Dim Strings As Collection
  Dim Stream As StreamReader = File.OpenText(FileName)  'Open the file
  Try
    While True  ' Loop will terminate with an EndOfStreamException
          ' error, when the end of stream is reached.
      Strings.Add(din.ReadLine())
    End While
  Catch EndOfStreamException
    ' No action is necessary; the end of the stream has been reached.
  Catch IOExcep As IOException
```

```
        ' Some kind of error occurred. Report error and clear the collection.
        MsgBox(IOException.Message)
        Strings = Nothing
      Finally
        Stream.Close()  ' Close the file
      End Try
      Return Strings
    End Function
```

您的浏览器提醒您：
使用本文档请尊重相关知识产权！

9.2.3　忽略错误（Throwing Exceptions）

有的时候，当一个错误发生以后，不需要进行特别的出错处理，只需忽略它就行了，在这种时候，用户需要用 Throw 关键字，下面就是忽略出错的代码：

```
Try
'some code
Catch e as Exception
Throw e 'pass on to calling routine
End Try
```

9.3　文件处理（File Handling）

如果用户引入 Microsoft.vb6.compatibility 的话，就可以使用一些 VB 6.0 中的文件处理函数。然而，语法却非常不一样，因为用户需要把所有括号中包含的参数都进行相当的改变，而这种参数却充斥着整个文件处理的代码。例如：

```
Input #f, s 'read a string from a file in VB6
```

现在变成了

```
Input(f, s) 'vb6 compatible string read from file
```

而且，VB 7.0 根本就没有 Line Input 的表述，所以现在在 VB 7.0 中，用户可以更加容易地使用 New File 和 Stream 方法来读写一个文件。

9.3.1　文件对象（file object）

文件对象代表了一个文件，而且有一些非常有用的方法来检验一个文件的存在以及重命名和删除一个文件。例如：

```
Dim fl as File
fl = new File("foo.txt")
If (fl.Exists) 'if the file exists
fl.Delete 'delete it
End If
```

用户也可以使用 File 对象来获取一个 FileStream 对象，然后通过使用 FileStream 对象来读写文件数据：

```
Dim ts as TextStream
Dim fs as FileStream
ts = fl.OpenText 'open a text file for reading
fs = fl.OpenRead 'open any file for reading
```

（1）读取一个文本文件（TextFile）

为了读取一个文本文件，用户可以使用 File 对象去获取一个 StreamReader 对象，然后使用文本流（text stream）的读取方法：

```
Dim ts as StreamReader
ts = fl.OpenText()
s = ts.readLine
```

（2）写一个文本文件（Write a Text File）

要建立和写一个文本文件，用户可以用 CreateText 方法得到一个 StreamWrite 对象，然后进行操作，比如：

```
Dim sw as StreamWriter
sw = fl.CreateText
sw.writeLine("write text into file")
```

如果用户想对一个已经存在的文件进行操作，可以使用一个带有逻辑参数的对象来进行操作：

```
Sw = new StreamWriter(path, true)
```

9.4 文件处理的出错处理

文件的出错处理中很多的错误主要是在文件的输入和输出时发生的。用户可能因为得到一个非法的文件名而出错，因为这个文件可能根本就不存在，或者文件夹不存在，或者是因为非法的参数，或者是因为文件写了保护。所以最好的办法就是把这些包含文件的输入输出操作放在 Try 模块中，并且确认用户已经能够获取所有可能会发生的错误。

例如，用户可以用如下代码来打开一个文件：

```
Try
ts = fl.Opentext()
Catch e as Exception
errDesc = e.Message
errFlag = true
Console.writeLine(errDesc) 'print out any error
End Try
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

9.5　检测文件的结束（Testing for End of File）

用户有两种方法可以确认没有超过文件的范围：考察是否有一个 NULL 的错误或者检验数据流的结束。当在超过文章的方位的区域读取数据时，并没有错误会发生，然而当在读取一个文件范围之外的一个文本字节的时候，它会返回一个 NULL 值。VB 7.0 没有提供 IsNull 方法，但是用户可以通过得到一个文本字节的长度来判断是不是为 NULL，如果此时给一个空字节运行一个 length 方法则系统会产生一个错误，这样，用户就可以通过这一点来判断是不是文件已经到了结束的地方了：

```
Public Function readLine() As String
'Read one line from the file
Dim s As String
Try
s = ts.readLine                'read line from file
lineLength = s.length          'use to catch null exception
Catch e As Exception
end_file = True                'set EOF flag if null
s = ""                         'and return zero length string
Finally
readLine = s
End Try
End Function
```

还有一个方法可以确认没有越出文件的范围就是使用 stream 对象的 peek 方法进行超前取值。它能够返回下一个字符的 ASCII 码，如果没有字符的话，就返回 NULL。

```
'example of alternate appraoch to detecting end of file
Public Function readLineE() As String
'Read one line from the file
Dim s As String
If ts.peek >= 0 Then           'look ahead
s = ts.readLine                'read if more chars
Return s
Else
end_file = True                'Ser EOF flag if none left
Return ""
End If
End Function
```

9.6 文件的静态方法（Static File Methods）

用户也许会认为使用 File 对象的方法就是建立一个具有特定名称的文件而已。然而实际上 File 对象也包含了一些静态方法（static methods）。通过这些方法，用户可以不需要使用 file 类的对象的名称而直接使用 File 类的名称来进行一些操作。在 VB 7.0 的语法中，这些被称为共享的方法（share methods），但是起的作用是一样的。

在表 9.3 中。列出了一些具有代表性意义的文件的静态方法的用法，现在假定已经有名为 fl 的文件对象，那么就可以通过下面这个表来参考一下不用 fl 的方法就能对 fl 进行一些方法的操作：

```
Dim fl as File
Fl = New File ("foo.txt")
```

表 9.3 文件的静态方法的用法

Instance method	Static method
Fl.Exists	File.FileExists（filename）
Fl.Delete	File.Delete（filename）
Sw = Fl.AppendText	File.AppendText（String）
Fl. IsDirectory	
Fl.isFile	
Fl.Length	
	File.Copy（fromFile，toFile）
	File.Move（fromTile，toFile）
	File.GetExtension（filename）
	File.HasExtension（filename）

9.7 VBFile 类

前面版本中，用户要通过写一个 VBFile 类来对一个 text 文件一行一行地读写和标识，在 Visual Basic.NET 中，用户可以通过利用 Visual Basic.NET 的文件处理类对这个类进行再次开发，使它能够在 Visual Basic.NET 中使用，并且具有相同的方法，用户必须把 VBFile 这个类为 Visual Basic.NET 再次开发一个接口，既然现在 Visual Basic.NET 和 Visual Basic 6.0 的语法不一样了，那么用户就需要重新写一个类来支持相同的接口。

主要的区别就是，用户可以在构造函数时，包含文件名和路径：

```
Public Sub New(ByVal filename As String)
'Create new file instance
file_name = filename                    'save file name
fl = New File(file_name)                'get file object
tokLine = ""                            'initialize tokenizer
sep = ","                               'and separator
End Sub
```

用户可以用两种方法来打开和读取一个文件，第一可以把文件包含进来，第二可以把文件名作为一种参数。

```
Public Overloads Function OpenForRead() As Boolean
Return OpenForRead(file_name)
End Function
'---------------
Public Overloads Function OpenForRead(_
ByVal Filename As String) As Boolean
'opens specified file
file_name = Filename                    'save file name
errFlag = False                         'clear errors
end_File = False                        'and end of file
Try
ts = fl.Opentext()                      'open the file
Catch e As Exception
errDesc = e.Message                     'save error message
errFlag = True                          'and flag
End Try
Return Not errFlag                      'false if error
End Function
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

现在就可以从一个 text 文件中读取数据了。

同法，下面的方法允许用户打开一个文件，并且在这个文件中以行为单位，写入文本：

```
Public Overloads Function OpenForWrite( _
ByVal fname As String) As Boolean
errFlag = False
Try
file_name = fname
fl = New File(file_name)            'create File object
sw = fl.CreateText                      'get StreamWriter
Catch e As Exception
errDesc = e.Message
errflag = True
End Try
openForWrite = Not errFlag
End Function
'------------
Public Overloads Function OpenForWrite() As Boolean
OpenForWrite = OpenForWrite(file_name)
End Function
'------------
```

超星阅览器提醒您
使用本复制品
请尊重相关知识产权!

```
Public Sub writeText(ByVal s As String)
sw.writeLine(s)                          'write text to stream
End Sub
```

既然现在已经开发了一个和以前一样具有相同方法的新类，并可以在 Visual Basic.NET 中运行，类以外的代码就无需作修改了。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

第 10 章　从 VB 6.0 到 VB.NET

本章要点：

Visual Basic.NET 的语言的变化

Visual Basic.NET 的面向对象的变化

Windows API 函数的变化

一个应用程序的比较

Visual Basic.NET（VB.NET）与 Visual Basic 6.0 的很多基本语法都是相同的，但是在很多问题上，Visual Basic.NET 可以说是一个全新的语言，和以前版本的 Visual Basic 不同，Visual Basic.NET 是完全的面相对象的语言，所以有很多方法或功能都因为这一特性而改变，基于以上的原因，应该把 Visual Basic.NET 看成为一个新的 .NET 构架的应用程序，而不是所写过的应用程序的重新编译。由于前面的章节已经对 Visual Basic.NET 有了一定的介绍，所以本章的立足点在于比较 Visual Basic 6.0 和 Visual Basic.NET，重点说明 Visual Basic 6.0 的工程升级到 Visual Basic.NET 的过程。

通过本章的介绍，读者将会看到 Visual Basic.NET 的很多优点，正是它的完全面相对象的概念，使得程序的设计周期变得更短。

10.1　语言的变化

10.1.1　语法的变化

用户会发现在 Visual Basic.NET 中，所有的过程函数以及类的调用都会用到括号，在 Visual Basic 6.0 中可以这样调用函数：

```
Dim MyCol As New Collection
MyCol.Add "Mine"
```

而在 Visual Basic.NET 中，必须将参数用括号括起来：

```
Dim MyCol As New Collection
MyCol.Add ("Mine")
```

另一个明显的不同就是，用参数进行传递时，参数的默认传递方式由“按地址传递”的方式变成了“按值传递”的方式，换句话说，对子过程调用时，默认情况下是不会对传进子过程的变量有任何改变的。关键字“ByVal”是默认的，如果想改变变量的值，就要在参数声明的时候加上关键字“ByRef”。

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

在 Visual Basic 6.0 中可以将程序写成：

```
Public Function Squarit( ByVal x As Single)
    x = x * x
    Squarit = x
End Sub
```

在 Visual Basic.NET 中可以省略“ByVal”这个关键字：

```
Public Function Squarit(x As Single)
    x = x * x
    Squarit = x
End Sub
```

还有五个关键字用法变了，或者已经不合法了，这五个关键字是：Set、Variant、Wend、Dim 和 ReDim，见下面两段程序：

在 Visual Basic 6.0 中的程序：

```
Set q = New Collection
Dim y as Variant
While x < 10
x = x + 1
Wend
Dim x as Integer , y as integer
ReDim Z(30) as Single
```

在 Visual Basic.NET 中的程序：

```
q = New Collection
Dim y as Object
While x < 10
x = x + 1
End While
Dim x , y as Integer
Dim Z() as Single
Z = New Single(30)
```

因为 Dim 语句允许将几个同类型的变量一起进行声明，所以下面的语句是合法的：

```
Dim x , y as Integer
```

但是不同变量是不可以一起声明的，下面的语句在 Visual Basic.NET 中是不合法的：

```
Dim x as Integer , y as Single
```

只能将这两个变量分开声明：

```
Dim x as Integer
Dim y as Single
```

这无论是在 Visual Basic.NET 还是在 Visual Basic 6.0 中都是合法的。Visual Basic.NET 的编译系统会对类似上面的很多用法报错，而且 Visual Basic 6.0 中的一些标准函数在 Visual Basic.NET 中不能使用，比如说函数“Instr”、“Left”和“Right”，它们的功能可以用对“String”类操作的方法 indexOf 和 substring 代替。

超星阅览器提醒您：
使用本复制品
须遵守相关知识产权!

注意：这些对字符串操作的方法一般是从 0 开始计数长度的，如表 10.1 所示。

表 10.1　字符串操作方法比较

Visual Basic 6.0	Visual Basic.NET
Insert（s，""）	s.indexOf（","）
Left（s，2）	s.substring（0，2）
Right（s，4）	s.substring（s.Length()，-4）

10.1.2　函数中增加的语句

在 Visual Basic 6.0 中有个笨拙的函数值返回的形式：将函数名作为返回值的赋值对象：

```
Public Function Squarit( x as Single)
    Squarit = x * x
End Function
```

在 Visual Basic.NET 中，可以不受这个束缚，可以简单地用“Return”语句返回函数值，就像其他语言的函数返回值一样：

```
Public Function Squarit( x as Single)
    Return x * x
End Function
```

这使得函数看起来简单，写起来容易。

10.1.3　Visual Basic.NET 中的数字类型

在 Visual Basic.NET 中，所有的定点整数都被定义成“Int32”类型，所有的浮点数都被定义成“Double”类型，如果想把一个整数值赋给一个“Single”类型的变量，要在这个整数后面加一个“F”代表浮点数（Floating Point）：

```
Dim time as Single = 123F
```

通常情况下，Visual Basic.NET 编译系统被设置成“Option Explicit”，用以要求变量声明，也限制了没有定义的类型转换。用户可以将一个“Integer”类型转换成“Single”或“Double”类型，但是若想将一个“Single”类型转换成“Integer”类型就必须将变量特殊的声明：

```
Dim k as Integer = time.ToInt16
```

10.1.4　赋值号的变化

Visual Basic.NET 新添了便捷的复合赋值号，这些符号可以在 C、C++、C#以及 Java 中找到，这些符号允许对变量进行加、减、乘、除的运算，而不需要写两次变量名：

```
Dim i as Integer = 5
    i += 5
    i -= 8
    i *= 4
```

可以用这些语法缩短代码，除法也有相同的用法，但是很少有人这么用，因为这样用降低程序的可读性。

10.2　面向对象的变化

10.2.1　Properties 的变化

Visual Basic 提供“Properties”（属性）这个结构，这个结构很像其他语言的“getXxx”、“setXxx”方法。在 Visual Basic 6.0 中可以通过定义“Get”和“Let”方法来声明一个“Property”（属性），这两个方法允许用户给私有变量赋值以及访问私有变量的值：

```
Property Get fred() as String
    fred = fredName
End Property

Property Let fred( s as String)
    fredName = s
End Property
```

在 Visual Basic.NET 中，这些属性集合到一个过程中：

```
Property fred() as String
    Get
        Return fredName
    End Get
    Set
        fredName = value
    End Set
End Property
```

注意：关键字“value”，在 Visual Basic.NET 中用作给属性传值：

```
Abc.fred = "dog"
```

执行完这条语句后，value 的值为“dog”，这两个给 Property 赋值的过程，都是为了给“Get”和“Set”提供接口，这些属性的用法在两种语言之间不存在差异。如果“fred”是一个“Boy”类的一个属性，可以这样操作这个属性：

```
Dim Kid as Boy
Kid.fred = "smart"

Dim brain as string
brain = kid.fred
```

通常情况下，Property 提供交替变化语句使 getFred 和 setFred 两个函数使用起来很方便，但是 Visual Basic 很多基本的类的属性并没有 Get 及 Let 方法，而且并不影响这些类的属性的使用。Get 和 Let 方法对面向对象的编程并没有明显的帮助。

超星阅览器版权保护
使用本复制品
请尊重相关知识产权!

10.2.2　Visual Basic.NET 的类

类是 Visual Basic.NET 中重要的部分，几乎程序的重要部分都是由一个或多个类组成的。类模块和窗体模块的区别在 Visual Basic.NET 中已经不存在了，而在 Visual Basic 6.0 中，窗体模块是具备图形界面的，相反的类模块不具备图形界面。在 Visual Basic.NET 中大多数程序都是各种各样的类，因为几乎所有的东西都是类，所以类的名称到处可见。这些类都被分组在不同的函数库里面，因此在引用这些类时，一定要注意这些类的名字空间（NameSpace）。

各种独立的 DLLs（动态连接库）包含了所有的这些库函数。运用“Imports”语句引用这些函数库的名称，这些函数库中的函数就可以合法使用了。例：

```
Imports System.Collection
```

执行这个语句后，就可以引用以“Collection”为名字空间的函数了。逻辑上，每个函数库都有不同的名字空间，每个名字空间是一个独立组，这个独立组中包含各种的类和方法，编译器在引用名字空间之后，会认为这些类和方法都是合法的。但是有一点是值得注意的：在引用名字空间的时候，有些类或方法可能是不只一个名字空间的的副本，这些副本会发生冲突。

最常用的名字空间是“System”，缺省时就会引用到它，而不用去声明。它包含很多基础的类和方法，这些类和方法在 Visual Basic.NET 中用来访问基础的类如“Application”、“Array”、“Console”、“Exceptions”、“Object”以及标准的数据类型“Byte”、“Boolean”、“Single”、“Double”及“String”等。下面一个最简单的应用程序在 Dos 界面输出“Hello VB World”：

```
Imports System
Public Class cMain
    Shared Sub Main()
        Console.WriteLine("Hello VB World")
    End Sub
End Class
```

每一个程序都必须有一个“Sub Main”的子程序，在类模块中，这个子程序必须为共有（Public）的。

另一种在 Visual Basic.NET 应用模块的方法，可以将上面的程序写成：

```
Public Module cMain
    Sub Main()
        Console.WriteLine("Hello VB World")
    End Sub
End Module
```

由于 Module 声明为“Public”，所以 Sub Main 不用再定义成共享的。Modules 和以前版本的 Visual Basic 相似，集成开发环境会将其定义成为“.bas”的文件类型。它的一个优点是，所有的方法和成员都是共有的，因此在整个工程中都可以引用，也正因为如此，模块中不能像类一样实现对数据的隐藏。

在 Visual Basic 6.0 中，虽然可以更改类的名字属性，但是类的名字通常和类的文件名

字相同；在 Visual Basic.NET 中，关键字“Class”允许用户将类声明为与类文件毫无关系的名字，事实上，用户必须为每一个类命不同的名字。Visual Basic 6.0 的类模块的扩展名为“.cls”，窗体模块的扩展名为“.frm”；Visual Basic.NET 可以将它们命名为任何文件名和扩展名，缺省时为“.vb”。

10.2.3 窗体及控件的变化

Visual Basic.NET 有一套全新的窗体形式，以及基于 Windows 下的窗体，Windows 下的窗体与 Visual Basic 6.0 的窗体形式有很大的相似性，但是仍有一些关键的地方发生了变化，下面将一一指出。

（1）Windows 窗体不再允许使用 OLE 容器，因此在开发 Visual Basic 6.0 时应尽量避免使用；

（2）在工具箱的控件中，没有“Shape”控件了，正方形或矩形的显示依靠“Label”控件，不支持椭圆和圆的图形，应该尽量避免使用这些图形；

（3）工具箱中的控件中，没有“Line”控件了，水平或垂直的直线可以依靠“Label”控件显示，斜线无法显示，应尽量避免使用；

（4）在 Windows 窗体上有了全新的一套图形命令，这些命令代替了以前的“Circle”、“Cls”、“PSet”、“Line”和“Point”方法。因为新的类模型和 Visual Basic 6.0 有很大的不同，因此这些方法都无法升级；

（5）Timer 控件不支持“Interval”属性值为 0。而在 Visual Basic 6.0 中设置这个属性为

10.3　Windows API 函数的变化

请尊重相关知识产权!

很多的 API 函数的用法基本上同 Visual Basic 6.0 的用法一致，只要小心数据类型的变化就可以了。Visual Basic 6.0 的“Long”数据类型变成 Visual Basic.NET 的“Integer”数据类型；“Integer”数据类型变成“Short”数据类型。在升级 API 函数的过程中，只要把这些数据类型改变，就可以完成与 Visual Basic 6.0 相同的功能。如下例：

```
Private Declare Function GetVersion Lib "kernel32" () As Long
Function GetVer()
    Dim Ver As Long
    Ver = GetVersion()
    MsgBox ("System Version is " & Ver)
End Function
```

变成：

```
Private Declare Function GetVersion Lib "kernel32" () As Integer
Function GetVer()
    Dim Ver As Integer
    Ver = GetVersion()
    MsgBox("System Version is " & Ver)
End Function
```

除了要将数字型的数据类型升级外，Visual Basic 6.0 的固定长度的 String 数据类型在 Visual Basic.NET 中不支持，取而代之的是有固定长度的 String 类，在 Visual Basic 6.0 中的很多情况下只要普通的 String 数据类型就可以实现只有 Visual Basic.NET 的 String 类才能实现的功能。如下例：

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Long) As Long
Function GetUser()
    Dim Ret As Long
    Dim UserName As String
    Dim Buffer As String * 25
    Ret = GetUserName(Buffer, 25)
    UserName = Left$(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox (UserName)
End Function
```

用一个普通的 String 类型的变量，再设定其长度要比声明一个固定长度的数组更好一些：

```
Dim Buffer As String
Buffer = String$(25, " ")
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

上面的 API 函数在 Visual Basic.NET 中应该写成：

```
Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Function GetUser()
    Dim Ret As Integer
    Dim UserName As String
    Dim Buffer As String
    Buffer = New String(CChar(" "), 25)
    Ret = GetUserName(Buffer, 25)
    UserName = Left(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox(UserName)
End Function
```

有三种情况需要在调用 Visual Basic 6.0 的 API 函数时做一些变化：第一种情况是传递用户自定义类型中包含有固定长度的 String 类型或二进制数组，在 Visual Basic.NET 中调用时需要给每一个固定长度的 String 类型及二进制数组添加“MarshallAs”属性（从 System.Runtime.InteropServices 中引用）；第二种情况是用“As Any”定义的变量，这在 Visual Basic.NET 中是不支持的。“As Any”类型的变量经常是传递字符串或者“NULL”值，可以声明两种形式的 API 函数，一个是“Long”类型，一个是“String”类型来代替 Visual Basic 6.0 中的 API，下例中，“GetPrivateProfileString”这个 API 函数中的参数“lpKeyName”就是“As Any”类型：

```
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias
    "GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal
        lpKeyName As Any, ByVal lpDefault As String, ByVal
            lpReturnedString As String, ByVal nSize As Long, ByVal
                lpFileName As String) As Long
```

可以将“As Any”类型声明两个版本，一个接受 Long 类型的参数，另一个接受 String 类型的参数：

```
Private Declare Function GetPrivateProfileStringKey Lib "kernel32" Alias
    "GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal
        lpKeyName As String, ByVal lpDefault As String, ByVal
            lpReturnedString As String, ByVal nSize As Long, ByVal
                lpFileName As String) As Long
Private Declare Function GetPrivateProfileStringNullKey Lib "kernel32"
    Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As String,
        ByVal lpKeyName As Long, ByVal lpDefault As String, ByVal
            lpReturnedString As String, ByVal nSize As Long, ByVal
                lpFileName As String) As Long
```

希望向 API 函数传递 NULL 值时，可以调用 GetPrivateProfileStringNullKey 这个函数，可以用这个方法升级 API 函数。

最后一种可能需要修改 API 函数的地方是利用 API 函数对线程、Windows 子类集、消

息队列的挂起等操作的情况。有的时候，这些函数会在 Visual Basic.NET 中产生实时错误，这些 API 函数有等价的 Visual Basic.NET 中或.NET 框架中的 API 函数，所以必须不断更新这些函数。

10.4　一个应用程序的比较

本节将对同一个应用程序的 Visual Basic 6.0 以及 Visual Basic.NET 的代码进行比较，通过比较更直观地看到 Visual Basic 的变化。

10.4.1　应用程序的界面

这是一个比较简单的应用程序，界面如图 10.1 所示。

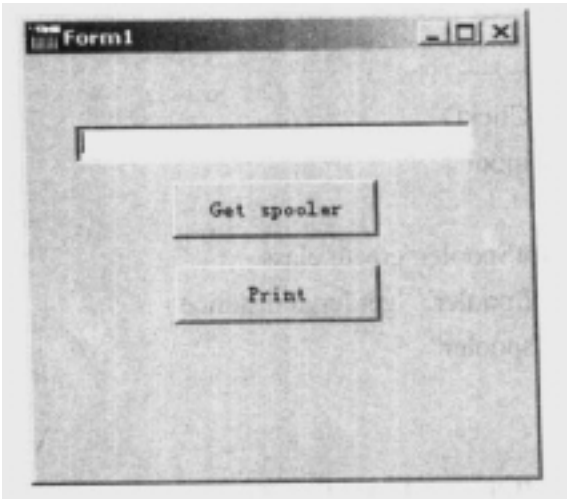


图 10.1　应用程序界面

当点击“Get spooler”按钮时，文本框中显示“Got spooler”；点击“Print”按钮时，弹出如图 10.2 所示的消息框；当再次点击“Get spooler”按钮时，弹出错误提示对话框，如图 10.3 所示。

图 10.2　消息框

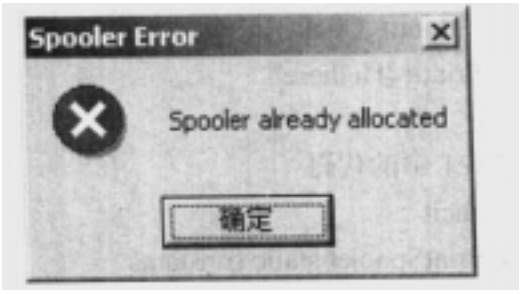


图 10.3　错误提示

其中文本框的名称在 Visual Basic 6.0 中为“errText”，在 Visual Basic.NET 中为“TextBox1”，两个按钮的名称分别为“GetSpooler”和“Printit”（在 Visual Basic 6.0 中）

或“btGetSpooler”和“Print”（在 Visual Basic.NET 中）。

10.4.2 Visual Basic 6.0 中的工程

1. 工程文件

在 Visual Basic 6.0 中，工程文件有：

（1）窗体 Form1.frm

（2）标准模块 Module1

（3）类模块 PrintSpooler

超星阅览器提醒：
使用本复制品
请尊重相关知识产权！

2. 各模块中的代码

（1）Form1 中的代码

```
Option Explicit                                                    1
'PrintSpooler Driver form                                          2
Dim prSp As PrintSpooler                                           3

'-----------------------------------
Private Sub GetSpooler_Click()                                     4
On Local Error GoTo nospool                                        5
 'create a spooler
      Set prSp = New PrintSpooler 'create class                    6
      Set prSp = prSp.GetSpooler   'get legal instance             7
      errText.Text = "Got spooler"                                 8
spexit:                                                            9
      Exit Sub                                                     10
'if the spooler causes an error we will get this message
nospool:                                                           11
   errText.Text = "Spooler already exists"                         12
   Resume spexit                                                   13
End Sub                                                            14
'-----------------------------------
Private Sub Printit_Click()                                        15
      prSp.Printit "Hi there"                                      16
End Sub                                                            17
```

（2）Module1 中的代码

```
Option Explicit                                                    1
'Singleton PrintSpooler static constants
Public spool_counter As Integer                                    2
Public glbSpooler As PrintSpooler                                  3
```

（3）PrintSpooler 中的代码

```
Option Explicit                                                    1
```

```
Private legalInstance As Boolean      'true for only one                    2
'Class PrintSpooler
'-----
Private Sub Class_Initialize()                                              3
If spool_counter = 0 Then          'create and save one instance            4
    legalInstance = True           'flag it                                 5
    Set glbSpooler = Me            'save it                                 6
    spool_counter = spool_counter + 1    'count it                          7
Else                                                                        8
    legalInstance = False          'no the legal one                        9
    MsgBox "Spooler already allocated", 16, "Spooler error"                 10
End If                                                                      11
End Sub                                                                     12
'--------------------------------
Public Function GetSpooler() As PrintSpooler                                13
    Set GetSpooler = glbSpooler    'return the legal one                    14
End Function                                                                15
'--------------------------------
Public Sub Printit(str As String)                                           16
If legalInstance Then     'test to make sure this isn't called directly     17
    MsgBox str, , ""                                                        18
End If                                                                      19
End Sub                                                                     20
'--------------------------------
Private Sub Class_Terminate()                                               21
'terminate legal class
If legalInstance Then                                                       22
  Set glbSpooler = Nothing                                                  23
  spool_counter = 0                                                         24
End If                                                                      25
End Sub                                                                     26
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权!

10.4.3　Visual Basic.NET 中的工程

1. 工程文件

（1）窗体文件 Form1.vb

（2）Spooler.vb 类

（3）SpoolerException.vb 类

2. 各模块中的代码

（1）Form1.vb 中的代码

```
Imports System.ComponentModel                                    1
Imports System.Drawing                                           2
Imports System.WinForms                                          3

Public Class Form1                                               4
    Inherits System.WinForms.Form                                5
    Private Spl As Spooler                                       6
    Public Sub New()                                             7
        MyBase.New                                               8

        Form1 = Me                                               9

        'This call is required by the Win Form Designer.
        InitializeComponent                                      10

        'TODO: Add any initialization after the InitializeComponent() call
    End Sub                                                      11

    'Form overrides dispose to clean up the component list.
    Overrides Public Sub Dispose()                               12
        MyBase.Dispose                                           13
        components.Dispose                                       14
    End Sub                                                      15

#Region " Windows Form Designer generated code "                 16

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.Container        17
    Private WithEvents btGetSpooler As System.WinForms.Button    18
    Private WithEvents Print As System.WinForms.Button           19
    Private WithEvents TextBox1 As System.WinForms.TextBox       20

    Dim WithEvents Form1 As System.WinForms.Form                 21

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Private Sub InitializeComponent()                            22
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
        Dim resources As System.Resources.ResourceManager = New _          23
            System.Resources.ResourceManager(GetType(Form1))               24

        Me.components = New System.ComponentModel.Container()              25
        Me.Print = New System.WinForms.Button()                            26
        Me.btGetSpooler = New System.WinForms.Button()                     27
        Me.TextBox1 = New System.WinForms.TextBox()                        28

        '@design Me.TrayHeight = 0
        '@design Me.TrayLargeIcon = False
        '@design Me.TrayAutoArrange = True
        Print.Location = New System.Drawing.Point(72, 120)                 29
        Print.Size = New System.Drawing.Size(104, 32)                      30
        Print.TabIndex = 1                                                 31
        Print.Text = "Print"                                               32

        btGetSpooler.Location = New System.Drawing.Point(72, 72)           33
        btGetSpooler.Size = New System.Drawing.Size(104, 32)               34
        btGetSpooler.TabIndex = 2                                          35
        btGetSpooler.Text = "Get spooler"                                  36

        TextBox1.Location = New System.Drawing.Point(24, 40)               38
        TextBox1.Text = " "                                                39
        TextBox1.TabIndex = 0                                              40
        TextBox1.Size = New System.Drawing.Size(200, 21)                   41
        Me.Text = "Form1"                                                  42
        Me.AutoScaleBaseSize = New System.Drawing.Size(6, 14)              43
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon) 44

        Me.Controls.Add(btGetSpooler)                                      45
        Me.Controls.Add(Print)                                             46
        Me.Controls.Add(TextBox1)                                          47
    End Sub

#End Region

    Protected Sub Print_Click(ByVal sender As Object, ByVal e As System.EventArgs) 48
        Try                                                                49
            spl.Print("Hi there")                                          50
        Catch ex As Exception                                              51
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

```
            ErrorBox("No spooler allocated")                                    52
        End Try                                                                 53
    End Sub                                                                     54
    '--------
    Private Sub ErrorBox(ByVal mesg As String)                                  55
        MessageBox.Show(mesg, "Spooler Error", Messagebox.IconError)            56
    End Sub                                                                     57
    '------
    Protected Sub btGetSpooler_Click(ByVal sender As Object, ByVal e As _
        System.EventArgs)                                                       58
        Try                                                                     59
            spl = Spooler.getSpooler                                            60
            TextBox1.text = "Got spooler"                                       61
        Catch ex As Exception                                                   62
            ErrorBox("Spooler already allocated")                               63
        End Try                                                                 64
    End Sub                                                                     65

End Class                                                                       66
```

（2）Spooler.vb 中的代码

```
Imports System                                                                  1
Imports System.ComponentModel                                                   2
Imports System.Drawing                                                          3
Imports System.WinForms                                                         4

Public Class Spooler                                                            5
    Private Shared Spool_counter As Integer                                     6
    Private Shared glbSpooler As Spooler                                        7
    Private legalInstance As Boolean                                            8
    '-----
    Private Sub New()                                                           9
        MyBase.New()                                                            10

        If spool_counter = 0 Then      'create and save one instance            11
            glbSpooler = Me            'save it                                 12
            spool_counter = spool_counter + 1     'count it                     13
            legalInstance = True                                                14
        Else                                                                    15
            legalInstance = False                                               16
            Throw New SpoolerException()                                        17
```

```
        End If                                                    18
    End Sub                                                       19
    '-----
    Public Shared Function getSpooler() As Spooler                20
        Try                                                       21
            glbSpooler = New Spooler()                            22
        Catch e As Exception                                      23
            Throw e                 'pass on to calling program   24
        Finally                                                   25
            getSpooler = glbSpooler   'or return the legal one    26
        End Try                                                   27
    End Function                                                  28
    '-----
    Public Sub Print(ByVal str As String)                         29
        If legalInstance Then                                     30
            MessageBox.Show(str)                                  31
        Else                                                      32
            Throw New SpoolerException()                          33
        End If                                                    34
    End Sub                                                       35
    '-----
End Class                                                         36
```

（3）SpoolerException.vb 中的代码

```
Imports System                                                    1
Imports System.ComponentModel                                     2
Imports System.Drawing                                            3
Imports System.WinForms                                           4

Public Class SpoolerException                                     5
    Inherits Exception                                            6
    Private mesg As String                                        7
    '---------
    Public Sub New()                                              8
        MyBase.New()                                              9
        mesg = "Only one spooler instance allowed"                10
    End Sub                                                       11
    '---------
    Public Overrides ReadOnly Property Message() As String        12
        Get                                                       13
            Message = mesg                                        14
```

```
        End Get                                          15
    End Property                                         16
End Class                                                17
```

超星阅览器提醒您：
使用本复制品
请尊重相关知识产权！

10.4.4　代码比较

1. 出错处理方面

Visual Basic 6.0 应用“On Local Error GoTo Label...Resume[Next]”语句（见 Visual Basic 6.0 中 Form1 的第 5 行、第 11 行和第 13 行代码），将这条语句可以由 Visual Basic.NET 的“Try... Catch...End Try”代替（见 Visual Basic.NET 中 Form1.vb 的第 60 到第 64 行代码）。只要将 On Local Error 和 Label 之间的代码添加在 Try 和 Catch 之中，Label 和 Resume 之间的代码添加到 Catch 和 End Try 之间即可完成代码的升级。

2. 类的只读属性

在 Visual Basic 6.0 中，可以把没有“Let”，而只有“Get”的属性理解成为只读的属性，而在 Visual Basic.NET 中，只读属性的定义如下（见 Visual Basic.NET 工程中的 SpoolerException.vb 中的第 12 到第 17 行代码）：

```
Public Overrride ReadOnly Property name() As Type
    Get
        Statements
    End Get
End Property
```

3. 公用变量声明的变化

在 Visual Basic 6.0 中，可以在标准模块中添加公有变量，这个变量对整个工程都是可见的，加上“Static”关键字，则这个变量会在内存中保存直到应用程序运行结束。在 Visual Basic.NET 中，可以用“Public Shared”关键字声明共享变量达到相同的用途。见 Visual Basic.NET 工程中 Spooer.vb 代码中第 5 行、第 7 行代码及第 20 行（共享函数）。