

<<C#与.NET 3.0高级程序设计>>

图书基本信息

书名：<<C#与.NET 3.0高级程序设计>>

13位ISBN编号：9787115168078

10位ISBN编号：7115168075

出版时间：2008-3

出版时间：人民邮电出版社

作者：特罗尔森

页数：972

译者：王少葵

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

<<C#与.NET 3.0高级程序设计>>

内容概要

《C#与.NET 3.0高级程序设计（特别版）》是C#领域久负盛名的经典著作，深入全面地叙述了C#编程语言和.NET平台核心，并以大量示例剖析相关概念。

书中介绍了C#的各种语言构造、.NET 2.0的类、核心API、公共中间语言（CIL）、动态程序集和ASP.NET扩展等内容；同时也介绍了.NET 3.0中的新的编程API包括WPF、WCF和WF的功能；另外，还介绍了最新的C# 3.0编程语言和LINQ编程技术。

《C#与.NET 3.0高级程序设计》由微软C# MVP Andrew Troelsen编写，历经多次修订，适合各层次.NET开发人员阅读。

作者简介

特罗尔森 (Andrew Troelsen) , 世界级C#专家, 微软Visual C#MVP。
他是著名的微软技术咨询企业Intcrtech的合伙人和副总裁, 该公司的客户包括微软、霍尼韦尔、美国航天局等。
他曾为MSDN网站和MacTec网站撰写了有关各种操作系统平台上.NET技术的文章, 并经常在业界主要技术会议上发表演讲和开设技术讲座。

书籍目录

第一部分 C#和.NET平台简介第1章 .NET之道1.1 了解.NET之前的世界1.1.1 C/Win32 API程序员的生活1.1.2 C++/MFC程序员的生活1.1.3 Visual Basic 6.0程序员的生活1.1.4 Java/J2EE程序员的生活1.1.5 COM程序员的生活1.1.6 Windows DNA程序员的生活1.2 .NET解决方案1.3 .NET平台构造块 (CLR、CTS和CLS) 简介1.4 C#的优点1.5 其他支持.NET的编程语言1.6 .NET程序集概览1.7 单文件程序集和多文件程序集1.8 CIL的作用1.8.1 CIL的好处1.8.2 将CIL编译成特定平台的指令1.9 .NET类型元数据的作用1.10 程序集清单的作用1.11 理解CTS1.11.1 CTS类类型1.11.2 CTS结构类型1.11.3 CTS接口类型1.11.4 CTS枚举类型1.11.5 CTS委托类型1.11.6 CTS类型成员1.11.7 内建的CTS数据类型1.12 理解CLS1.13 理解CLR1.14 程序集/命名空间/类型的区别1.14.1 以编程方式访问命名空间1.14.2 引用外部程序集1.15 使用ildasm.exe1.15.1 查看CIL代码1.15.2 查看类型元数据1.15.3 查看程序集元数据1.16 部署.NET运行库1.17 .NET的平台无关性1.18 小结第2章 构建C#应用程序2.1 安装.NET Framework 2.0 SDK2.2 C#命令行编译器 (csc.exe) 2.2.1 配置C#命令行编译器2.2.2 配置其他.NET命令行工具2.3 使用csc.exe构建C#应用程序2.3.1 引用外部程序集2.3.2 使用csc.exe编译多个源文件2.3.3 引用多个外部程序集2.4 使用csc.exe响应文件2.5 命令行调试器 (cordbg.exe) 2.6 使用TextPad构建.NET应用程序2.6.1 启用C#关键字着色2.6.2 配置*.cs文件过滤器2.6.3 与csc.exe关联2.6.4 将运行命令与菜单项相关联2.6.5 启用C#代码片段2.7 使用SharpDevelop构建.NET应用程序2.7.1 SharpDevelop2.7.2 Project Scout和Classes Scout2.7.3 Assembly Scout2.7.4 Windows窗体设计器2.8 使用Visual C# 2005 Express构建.NET应用程序2.9 使用Visual Studio 2005构建.NET应用程序2.9.1 Visual Studio 20052.9.2 Solution Explorer工具2.9.3 Class View工具2.9.4 Code Definition窗口2.9.5 Object Browser工具2.9.6 集成对代码重构的支持2.9.7 代码扩展和围绕技术2.9.8 可视化Class Designer2.9.9 对象测试平台2.9.10 集成的帮助系统2.10 其他.NET开发工具2.11 小结第二部分 C#编程语言第3章 C#语言基础3.1 剖析一个简单的C#程序3.1.1 Main()方法的其他形式3.1.2 处理命令行参数3.1.3 使用Visual Studio 2005指定命令行参数3.2 有趣的题外话: System.Environment类3.3 定义类并创建对象3.3.1 构造函数的作用3.3.2 是内存泄露吗3.3.3 定义“应用程序对象”3.4 System.Console类3.4.1 使用Console类进行基本的输入和输出3.4.2 格式化控制台输出3.4.3 .NET字符串格式化标志3.5 设置成员的可见性3.6 类成员变量的默认值3.7 成员变量的初始化语法3.8 定义常量数据3.9 定义只读字段3.10 static关键字3.10.1 静态方法3.10.2 静态数据3.10.3 静态构造函数3.10.4 静态类3.11 方法参数修饰符3.11.1 默认的参数传递行为3.11.2 out修饰符3.11.3 ref修饰符3.11.4 params修饰符3.12 迭代结构3.12.1 for循环3.12.2 foreach循环3.12.3 while和do/while循环结构3.13 判断结构与关系/相等运算符3.13.1 if/else语句3.13.2 switch语句3.14 值类型和引用类型3.14.1 值类型、引用类型和赋值运算符3.14.2 包含引用类型的值类型3.14.3 按值传递引用类型3.14.4 按引用传递引用类型3.14.5 值类型和引用类型: 最后的细节3.15 装箱与拆箱操作3.15.1 实用的装箱和拆箱示例3.15.2 拆箱自定义的值类型3.16 使用.NET枚举3.17 最重要的类: System.Object3.18 重写System.Object的一些默认行为3.18.1 重写System.Object.ToString()3.18.2 重写System.Object.Equals()3.18.3 重写System.Object.GetHashCode()3.18.4 测试重写的成员3.18.5 System.Object的静态成员3.19 系统数据类型 (和C#简化符号) 3.19.1 数值数据类型的实验3.19.2 System.Boolean的成员3.19.3 System.Char的成员3.19.4 从字符串数据中解析数值3.19.5 System.DateTime和System.TimeSpan3.20 System.String数据类型3.20.1 基本的字符串操作3.20.2 转义字符3.20.3 使用C#的逐字字符串3.21 System.Text.StringBuilder的作用3.22 .NET数组类型3.22.1 数组作为参数 (和返回值) 3.22.2 使用多维数组3.22.3 System.Array基类3.23 C#的可空类型3.23.1 使用可空类型3.23.2 ??运算符3.24 定义自定义命名空间3.24.1 类型的完全限定名3.24.2 使用别名定义命名空间3.24.3 创建嵌套的命名空间3.24.4 Visual Studio 2005中的“默认命名空间”3.25 小结第4章 C# 2.0面向对象编程4.1 C#的类类型4.1.1 方法重载4.1.2 使用C#的this进行自引用4.1.3 定义类的公共接口4.2 回顾OOP的支柱4.2.1 封装4.2.2 继承4.2.3 多态4.3 第一个支柱: C#的封装支持4.3.1 使用传统的访问方法和修改方法执行封装4.3.2 另一种形式的封装: 类属性4.3.3 C#属性的内部表示4.3.4 控制属性get/set语句的可见性级别4.3.5 只读和只写属性4.3.6 静态属性4.4 第二个支柱: C#的继承支持4.4.1 使用base控制基类的创建4.4.2 关于多基类4.4.3 保护家族的秘密: protected关键字4.4.4 防止继承: 密封类4.5 为包含/委托编程4.6 第三个支柱: C#的多态支持4.6.1 virtual和override关键字4.6.2 再谈sealed关键字4.6.3 抽象类4.6.4 强制多态活动: 抽象方法4.6.5 成员隐藏4.7 C#的类型转换规则4.7.1 确定Employee的类型4.7.2 数

<<C#与.NET 3.0高级程序设计>>

值类型转换4.8 C#的分部类型4.9 通过XML生成C#源代码的文档4.9.1 XML代码注释格式化字符4.9.2 转换XML代码注释4.10 小结第5章 对象的生命周期5.1 类、对象和引用5.2 对象生命周期的基础5.3 应用程序根的作用5.4 对象的代5.5 System.GC类型5.6 构建可终结对象5.6.1 重写System.Object.Finalize()5.6.2 终结过程的细节5.7 构建可处置对象5.8 构建可终结类型和可处置类型5.9 小结第6章 结构化异常处理6.1 错误、bug与异常6.2 .NET异常处理的作用6.2.1 .NET异常处理的四要素6.2.2 System.Exception基类6.3 最简单的例子6.3.1 引发普通的异常6.3.2 捕获异常6.4 配置异常的状态6.4.1 TargetSite属性6.4.2 StackTrace属性6.4.3 HelpLink属性6.4.4 Data属性6.5 系统级异常 (System.SystemException) 6.6 应用程序级异常 (System.ApplicationException) 6.6.1 构建自定义异常, 第一部分6.6.2 构建自定义异常, 第二部分6.6.3 构建自定义异常, 第三部分6.7 处理多个异常6.7.1 通用的catch语句6.7.2 再次引发异常6.7.3 内部异常6.8 finally块6.9 谁在引发什么异常6.10 未处理异常的后果6.11 使用Visual Studio 2005调试未处理的异常6.12 小结第7章 接口与集合7.1 使用C#定义接口7.2 使用C#实现接口7.3 接口与抽象基类的对比7.4 在对象级别调用接口成员7.4.1 获取接口引用: as关键字7.4.2 获取接口引用: is关键字7.5 接口作为参数7.6 接口作为返回值7.7 接口类型数组7.8 显式接口实现7.9 构建接口层次结构7.10 使用Visual Studio 2005实现接口7.11 构建可枚举类型 (IEnumerable和IEnumerator) 7.12 构建可克隆的对象 (ICloneable) 7.13 构建可比较的对象 (IComparable) 7.13.1 指定多个排序的顺序 (IComparer) 7.13.2 自定义属性、自定义排序类型7.14 System.Collections命名空间的接口7.14.1 ICollection接口的作用7.14.2 IDictionary接口的作用7.14.3 IDictionaryEnumerator接口的作用7.14.4 IList接口的作用7.15 System.Collections命名空间中的类7.15.1 操作ArrayList类型7.15.2 操作Queue类型7.15.3 操作Stack类型7.16 System.Collections.Specialized命名空间7.17 小结第8章 回调接口、委托与事件8.1 回调接口8.2 .NET委托类型8.3 使用C#定义委托8.4 System.MulticastDelegate与System.Delegate基类8.5 最简单的委托示例8.6 使用委托改造Car类型8.7 更复杂的委托示例8.7.1 委托作为参数8.7.2 分析委托代码8.8 委托协变8.9 C#事件8.9.1 揭开事件的神秘面纱8.9.2 监听传入的事件8.9.3 使用Visual Studio 2005简化事件注册8.9.4 严谨规范的事件8.10 C#匿名方法8.11 C#方法组转换8.12 小结第9章 高级C#类型构造技术9.1 构建自定义索引器9.2 类型索引器的内部表示方式9.3 索引器: 最后的细节9.4 运算符重载9.5 重载二元运算符9.6 重载一元运算符9.7 重载相等于运算符9.8 重载比较运算符9.9 重载运算符的内部表示形式9.10 在不支持重载运算符的语言中使用重载运算符9.11 运算符重载的最后思考9.12 自定义类型转换9.12.1 回顾: 数值转换9.12.2 回顾: 相关的类类型间的转换9.13 创建自定义转换例程9.14 定义隐式转换例程9.15 自定义转换例程的内部表示9.16 C#的高级关键字9.16.1 checked关键字9.16.2 unchecked关键字9.16.3 指针类型9.16.4 sizeof关键字9.17 C#预处理指令9.17.1 指定代码区域9.17.2 条件代码编译9.18 小结第10章 泛型10.1 再论装箱、拆箱和System.Object之间的关系10.2 装箱/拆箱操作的问题10.2.1 类型安全与强类型集合10.2.2 装箱与强类型集合10.3 System.Collections.Generic命名空间10.4 创建泛型方法10.5 创建泛型结构 (或类) 10.6 创建自定义泛型集合10.6.1 使用where约束类型参数10.6.2 运算符约束的不足10.7 创建泛型基类10.8 创建泛型接口10.9 创建泛型委托10.9.1 在.NET 1.1下模拟泛型委托10.9.2 嵌套委托相关简介10.10 小结第三部分 .NET程序集编程第11章 .NET程序集入门11.1 .NET程序集的作用11.1.1 程序集促进代码重用11.1.2 程序集确定类型边界11.1.3 程序集是可版本化的单元11.1.4 程序集是自描述的11.1.5 程序集是可配置的11.2 .NET程序集的格式11.2.1 Win32文件首部11.2.2 CLR文件首部11.2.3 CIL代码、类型元数据和程序集清单11.2.4 可选的程序集资源11.2.5 单文件程序集和多文件程序集11.3 构建和使用单文件程序集11.3.1 清单11.3.2 CIL11.3.3 类型元数据11.3.4 构建C#客户端应用程序11.3.5 构建Visual Basic .NET客户端应用程序11.3.6 实现跨语言继承11.4 构建和使用多文件程序集11.4.1 ufo.netmodule文件11.4.2 airvehicles.dll文件11.4.3 使用多文件程序集11.5 私有程序集11.5.1 私有程序集的标识11.5.2 探测过程11.5.3 配置私有程序集11.5.4 配置文件和Visual Studio 200511.5.5 .NET Framework 2.0配置工具简介11.6 共享程序集11.6.1 强名称11.6.2 为CarLibrary.dll赋予强名称11.6.3 使用Visual Studio 2005为程序集赋予强名称11.6.4 在GAC中安装和移除共享程序集11.6.5 延迟签名的作用11.7 使用共享程序集11.8 配置共享程序集11.8.1 冻结当前的共享程序集11.8.2 构建共享程序集2.0.0.0版本11.8.3 动态重定向到共享程序集的特定版本11.8.4 再次研究.NET Framework 2.0 配置工具11.9 研究GAC的内部结构11.10 发行者策略程序集11.11 < codeBase >元素11.12 System.Configuration命名空间11.13 机器配置文件11.14 程序集绑定总体流程图11.15 小结第12章 类型反射、晚期绑定和基于特性的编程12.1 类型元数据的必要性12.1.1 查看 (部分) EngineState枚举的元数

<<C#与.NET 3.0高级程序设计>>

据12.1.2 查看 (部分) Car类型的元数据12.1.3 研究TypeRef12.1.4 记录定义的程序集12.1.5 记录引用的程序集12.1.6 记录字符串字面量12.2 反射12.2.1 System.Type类12.2.2 使用System.Object.GetType()得到Type引用12.2.3 使用System.Type.GetType()得到Type引用12.2.4 使用typeof()得到Type引用12.3 构建自定义的元数据查看器12.3.1 反射方法12.3.2 反射字段和属性12.3.3 反射实现的接口12.3.4 显示其他信息12.3.5 实现Main()12.3.6 反射方法参数和返回值12.4 动态加载程序集12.5 反射共享程序集12.6 晚期绑定12.6.1 System.Activator类12.6.2 调用没有参数的方法12.6.3 调用有参数的方法12.7 特性编程12.7.1 特性的使用者12.7.2 在C#中使用预定义特性12.7.3 为特性指定构造参数12.7.4 Obsolete特性12.7.5 C#特性简化符号12.8 构建自定义特性12.8.1 应用自定义特性12.8.2 限制特性使用12.9 程序集级别 (和模块级别) 特性12.10 使用早期绑定反射特性12.11 使用晚期绑定反射特性12.12 反射、晚期绑定和自定义特性的使用背景12.13 构建可扩展的应用程序12.13.1 构建CommonSnappable-Types.dll12.13.2 构建C#插件12.13.3 构建Visual Basic.NET插件12.13.4 构建可扩展的Windows窗体应用程序12.14 小结第13章 进程、应用程序域、上下文和CLR宿主13.1 回顾传统的Win32进程13.2 .NET平台下与进程进行交互13.2.1 列举运行中的进程13.2.2 研究特定的进程13.2.3 研究进程的线程集合13.2.4 研究进程中的模块集合13.2.5 以编程方式启动或结束进程13.3 .NET应用程序域13.3.1 列举进程中的应用程序域13.3.2 以编程方式创建新的应用程序域13.3.3 以编程方式卸载应用程序域13.4 对象上下文边界13.4.1 上下文灵活和上下文绑定类型13.4.2 定义上下文绑定对象13.4.3 研究对象的上下文13.5 进程、应用程序域和上下文小结13.6 承载CLR13.6.1 CLR的并行执行13.6.2 加载特定的CLR版本13.6.3 其他的CLR宿主13.7 小结第14章 构建多线程应用程序14.1 进程、应用程序域、上下文及线程之间的关系14.2 .NET委托的简短回顾14.3 委托的异步天性14.3.1 BeginInvoke()和EndInvoke()方法14.3.2 System.IAsyncResult接口14.4 异步调用方法14.4.1 同步调用线程14.4.2 AsyncCallback委托的作用14.4.3 AsyncResult类的作用14.4.4 传递和接收自定义状态数据14.5 System.Threading命名空间14.6 System.Threading.Thread类14.6.1 获得当前线程的统计信息14.6.2 Name属性14.6.3 Priority属性14.7 以编程方式创建次线程14.7.1 使用ThreadStart委托14.7.2 使用Parameterized-ThreadStart委托14.7.3 前台线程和后台线程14.8 并发问题14.8.1 使用C#的lock关键字进行同步14.8.2 使用System.Threading.Monitor类型进行同步14.8.3 使用System.Threading.Interlocked类型进行同步14.8.4 使用[Synchronization]进行同步14.9 使用Timer Callback编程14.10 CLR线程池14.11 小结第15章 CIL和动态程序集的作用15.1 CIL编程的本质15.2 研究CIL指令、特性和操作码15.2.1 CIL指令的作用15.2.2 CIL特性的作用15.2.3 CIL操作码的作用15.2.4 区别CIL操作码和CIL助记符15.3 入栈和出栈: CIL基于栈的本质15.4 正反向工程15.4.1 CIL代码标签的作用15.4.2 与CIL交互: 修改*.il文件15.4.3 使用ilasm.exe编译CIL代码15.4.4 使用SharpDevelop编译CIL代码15.4.5 使用ILIDE#编译CIL代码15.4.6 peverify.exe的作用15.5 CIL指令和特性15.5.1 在CIL中指定外部引用程序集15.5.2 在CIL中定义当前程序集15.5.3 在CIL中定义命名空间15.5.4 在CIL中定义类类型15.5.5 在CIL中定义和实现接口15.5.6 在CIL中定义结构15.5.7 在CIL中定义枚举15.5.8 编译CILTypes.il文件15.6 .NET基类库、C#和CIL数据类型的映射15.7 在CIL中定义成员15.7.1 在CIL中定义数据字段15.7.2 在CIL中定义类型的构造函数15.7.3 在CIL中定义属性15.7.4 定义成员参数15.8 剖析CIL操作码15.8.1 了解.maxstack指令15.8.2 在CIL中声明局部变量15.8.3 在CIL中映射参数到局部变量15.8.4 隐式this引用15.8.5 在CIL中使用循环结构15.9 使用CIL构建.NET程序集15.9.1 构建CILCars.dll15.9.2 构建CILCarClient.exe15.10 动态程序集15.10.1 System.Reflection.Emit命名空间15.10.2 System.Reflection.Emit.ILGenerator的作用15.10.3 产生动态的程序集15.10.4 产生程序集和模块集15.10.5 ModuleBuilder类型的作用15.10.6 产生HelloClass类型和字符串成员变量15.10.7 产生构造函数15.10.8 产生HelloWorld()方法15.10.9 使用动态产生的程序集15.11 System.CodeDom简单说明15.12 小结第四部分 使用.NET库编程第16章 System.IO命名空间16.1 研究System.IO命名空间16.2 Directory (Info) 和File (Info) 类型16.3 使用DirectoryInfo类型16.3.1 FileAttributes枚举16.3.2 使用DirectoryInfo类型列出文件16.3.3 使用DirectoryInfo类型创建子目录16.4 使用Directory类型16.5 使用DriveInfo类类型16.6 使用FileInfo类16.6.1 FileInfo.Create()方法16.6.2 FileInfo.Open()方法16.6.3 FileInfo.OpenRead()和FileInfo.OpenWrite()方法16.6.4 FileInfo.OpenText()方法16.6.5 FileInfo.CreateText()和FileInfo.AppendText()方法16.7 使用File类型16.8 Stream抽象类16.9 使用StreamWriter和StreamReader类型16.9.1 写文本文件16.9.2 从文本文件读16.9.3 直接创建StreamWriter/StreamReader类型16.10 使用StringWriter和StringReader16.11 使用BinaryWriter

和BinaryReader16.12 以编程方式“观察”文件16.13 实现异步文件I/O操作16.14 小结第17章 对象序列化17.1 对象序列化17.2 为序列化配置对象17.3 选择序列化格式化程序17.3.1 IFormatter和IRemoting-Formatting接口17.3.2 在格式化程序中的类型保真17.4 使用BinaryFormatter序列化对象17.5 使用SoapFormatter序列化对象17.6 使用XmlSerializer序列化对象17.7 持久化对象集合17.8 自定义序列化过程17.8.1 深入了解对象序列化17.8.2 使用ISerializable自定义序列化17.8.3 使用特性自定义序列化17.9 可序列化对象的版本处理17.10 小结第18章 .NET远程处理层18.1 定义.NET远程处理18.2 .NET远程处理命名空间18.3 .NET远程处理框架18.3.1 代理和消息18.3.2 信道18.3.3 .NET格式化程序的作用18.3.4 综合讨论18.3.5 扩展默认管道的简单介绍18.4 .NET远程处理数据交换的术语18.4.1 对象封送方式：MBR还是MBV18.4.2 选择MBR的激活类型：WKO还是CAO18.4.3 WKO类型的状态配置：单例还是单一调用18.4.4 MBR对象类型特性小结18.5 .NET远程处理项目的基本部署18.6 构建第一个分布式应用程序18.6.1 构建普通程序集18.6.2 构建服务器端程序集18.6.3 建立SimpleRemoteObject-Client.exe程序集18.6.4 测试远程处理应用程序18.7 ChannelServices类型18.8 RemotingConfiguration类型18.9 WKO类型激活模式18.10 把服务器部署成远程机器18.11 利用TCP通道18.12 简单谈谈IpcChannel18.13 远程处理配置文件18.13.1 构建服务器端*.config文件18.13.2 构建客户端*.config文件18.14 使用MBV对象18.14.1 构建普通程序集18.14.2 构建服务器端程序集18.14.3 构建客户端程序集18.15 客户端激活的对象18.16 CAO/WKO-Singleton对象基于租约的生存期18.16.1 默认的租约行为18.16.2 改变默认租约特性18.16.3 服务器端租约调整18.16.4 客户端租约调整18.17 服务器端（和客户端）租约主办方机制18.18 远程对象的其他宿主18.18.1 使用Windows服务承载远程对象18.18.2 使用IIS承载远程对象18.19 异步远程处理18.20 小结第19章 使用System.Windows.Forms 构建更好的窗体19.1 System.Windows.Forms命名空间概述19.2 使用Windows窗体类型19.2.1 手动创建主窗口19.2.2 重视分离关注点19.3 Application类的作用19.3.1 Application类的使用19.3.2 System.EventHandler委托19.4 剖析Form19.5 Control类的功能19.5.1 Control类的使用19.5.2 响应MouseMove事件19.5.3 检测被单击的鼠标键19.5.4 响应键盘事件19.6 Form类的功能19.7 使用Visual Studio 2005构建窗口应用程序19.7.1 启用过时的控件19.7.2 研究Visual Studio 2005 Windows窗体项目19.7.3 在设计时处理事件19.7.4 Program类19.7.5 被自动引用的程序集19.8 MenuStrips和ContextMenuStrips的使用19.8.1 向MenuStrip添加TextBox19.8.2 创建上下文菜单19.8.3 选择菜单项19.9 使用StatusStrip19.9.1 设计菜单系统19.9.2 设计StatusStrip19.9.3 用Timer类型工作19.9.4 切换显示19.9.5 显示菜单选择提示符19.9.6 建立“Ready”状态19.10 使用ToolStrip工作19.11 构建MDI运用程序19.11.1 构建父窗体19.11.2 构建子窗体19.11.3 复制子窗体19.12 小结第20章 使用GDI+绘制图形20.1 GDI+命名空间概述20.2 System.Drawing命名空间概述20.3 System.Drawing实用类型20.3.1 Point(F)类型20.3.2 Rectangle(F)类型20.3.3 Region类20.4 Graphics类20.5 Paint会话20.5.1 使窗体的客户区域失效20.5.2 在Paint事件处理程序外获取Graphics对象20.5.3 关于Graphics对象的释放20.6 GDI+坐标系统20.6.1 默认度量单位20.6.2 指定另一种度量单位20.6.3 指定另一个原点20.7 定义颜色值20.8 操作字体20.8.1 使用字体族20.8.2 使用字体名和字体大小20.8.3 枚举安装的字体20.8.4 FontDialog类20.9 System.Drawing.Drawing2D命名空间概述20.10 使用Pen20.11 使用Brush20.11.1 使用HatchBrush20.11.2 使用TextureBrush20.11.3 使用LinearGradient-Brush20.12 呈现图像20.13 PictureBox控件的拖动和单击测试20.13.1 呈现图像的单击测试20.13.2 非矩形图像的单击测试20.14 .NET资源格式20.14.1 System.Resources命名空间20.14.2 以编程方式创建*.resx文件20.14.3 构建*.resources文件20.14.4 把*.resources文件绑定到.NET程序集20.14.5 使用ResourceWriter20.14.6 使用Visual Studio 2005生成资源20.14.7 通过编程读取资源20.15 小结第21章 Windows窗体控件编程21.1 Windows窗体控件21.2 手动给窗体添加控件21.3 使用Visual Studio 2005给窗体添加控件21.4 基本控件的使用21.4.1 Label的作用21.4.2 TextBox的作用21.4.3 MaskedTextBox的作用21.4.4 Button的作用21.4.5 CheckBox、RadioButton和GroupBox的作用21.4.6 CheckedListBox的作用21.4.7 ListBox的作用21.4.8 ComboBox的作用21.5 配置选项卡的次序21.6 设置窗体的默认输入按钮21.7 更多奇特的控件21.7.1 MonthCalendar控件的作用21.7.2 ToolTip控件的作用21.7.3 TabControl控件的作用21.7.4 TrackBar的作用21.7.5 Panel的作用21.7.6 UpDown控件的作用21.7.7 ErrorProvider的作用21.7.8 TreeView的作用21.7.9 WebBrowser的作用21.8 创建自定义Windows窗体控件21.8.1 创建图像21.8.2 构建设计时UI21.8.3 实现核心的CarControl21.8.4 定义自定义事件21.8.5 定义自定义属性21.8.6 控制动画21.8.7 显示昵称21.9 测试CarControl类型21.10 创建自定义CarControl窗体宿主21.11 System.ComponentModel命名空间的作

用21.11.1 增强CarControl的设计时外观21.11.2 定义默认的属性和默认的事件21.11.3 指定自定义的工具箱位图21.12 创建自定义对话框21.12.1 DialogResult属性21.12.2 窗体继承21.13 动态定位Windows窗体控件21.13.1 Anchor属性21.13.2 Dock属性21.13.3 表和流布局21.14 小结第22章 使用ADO.NET访问数据库22.1 ADO.NET高层次定义22.2 ADO.NET的数据提供者22.2.1 微软提供的数据库提供者22.2.2 选择第三方的数据库提供者22.3 其他的ADO.NET命名空间22.4 System.Data类型22.4.1 IDbConnection接口的作用22.4.2 IDbTransaction接口的作用22.4.3 IDbCommand接口的作用22.4.4 IDbDataParameter/IData-Parameter接口的作用22.4.5 IDbDataAdapter/IData-Adapter接口的作用22.4.6 IDataReader/IDataRecord接口的作用22.5 使用接口抽象数据提供者22.6 使用应用程序配置文件增加灵活性22.7 .NET 2.0提供者工厂模型22.7.1 为数据提供者工厂注册22.7.2 完整的数据提供者的例子22.8 <connectionStrings>元素22.9 安装Cars数据库22.10 ADO.NET的连接式访问22.10.1 使用连接对象22.10.2 使用.NET 2.0的ConnectionStringBuilder22.10.3 使用命令对象22.11 使用数据读取器22.12 使用命令对象修改表22.12.1 插入新的记录22.12.2 删除现有记录22.12.3 更新现有记录22.13 使用参数化的命令对象22.14 使用DbCommand执行存储过程22.15 .NET 2.0的异步数据访问22.16 ADO.NET断开式访问方式22.17 DataSet的作用22.18 使用DataColumn22.18.1 构建DataColumn22.18.2 启用列自增22.18.3 把DataColumn加入DataTable22.19 使用DataRow22.20 使用DataTable22.21 持久化DataSet (和DataTable) 成为XML22.22 把DataTable呈现到用户界面22.22.1 以编程方式删除行22.22.2 应用过滤和排序22.22.3 更新行22.23 使用DataView类型22.24 使用数据适配器22.24.1 使用数据适配器填充DataSet22.24.2 映射数据库名称为友好名称22.25 使用数据适配器对象更新数据库22.25.1 设置InsertCommand属性22.25.2 设置UpdateCommand属性22.25.3 设置DeleteCommand属性22.26 使用CommandBuilder类型自动生成SQL命令22.27 多表DataSet和DataRelation对象22.28 最后看一下 (数据) 向导22.28.1 强类型化的DataSet22.28.2 自动生成的数据组件22.29 小结第五部分 Web应用程序和XML Web服务第23章 ASP.NET 2.0网页和Web控件23.1 HTTP的作用23.2 Web应用程序和Web服务23.2.1 使用IIS虚拟目录工作23.2.2 ASP.NET 2.0开发服务器23.3 HTML的作用23.3.1 HTML文档结构23.3.2 HTML表单开发23.3.3 构建基于HTML的用户界面23.4 客户端脚本的作用23.4.1 客户端脚本示例23.4.2 验证default.htm表单数据23.5 提交表单数据 (GET和POST) 23.6 构建传统的ASP页面23.7 传统ASP相关问题23.7.1 ASP.NET 1.x的主要优点23.7.2 ASP.NET 2.0的主要改进23.8 ASP.NET 2.0命名空间23.9 ASP.NET网页代码模型23.9.1 使用单文件页面模型23.9.2 使用代码隐藏页面模型23.10 ASP.NET站点目录结构细节23.10.1 Bin文件夹的作用23.10.2 App_Code文件夹的作用23.11 ASP.NET 2.0页面编译周期23.11.1 单文件页面的编译周期23.11.2 多文件页面的编译周期23.12 页面类型的继承链23.13 与传入的HTTP请求交互23.13.1 获得浏览器统计数据23.13.2 访问传入的表单数据23.13.3 IsPostBack属性23.14 与输出HTTP响应交互23.14.1 提交HTML内容23.14.2 重定向用户23.15 ASP.NET网页的生命周期23.15.1 AutoEventWireUp特性的作用23.15.2 Error事件23.16 Web控件的本质23.16.1 取得服务器端事件处理权23.16.2 AutoPostBack属性23.17 System.Web.UI.Control类型23.17.1 枚举所包含的控件23.17.2 动态添加 (和删除) 控件23.18 System.Web.UI.WebControls.WebControl类型的关键成员23.19 ASP.NET Web控件的类别23.20 构建简单的ASP.NET 2.0站点23.20.1 使用母版页工作23.20.2 定义Default.aspx内容页面23.20.3 设计Inventory内容页面23.20.4 设计Build a Car内容页面23.21 验证控件的作用23.21.1 RequiredFieldValidator23.21.2 RegularExpression-Validator23.21.3 RangeValidator23.21.4 CompareValidator23.21.5 创建ValidationSummary23.22 小结第24章 ASP.NET 2.0 Web应用程序24.1 状态问题24.2 ASP.NET状态管理技术24.3 ASP.NET视图状态的作用24.3.1 演示视图状态24.3.2 添加自定义视图状态数据24.3.3 控件状态简述24.4 Global.asax文件的作用24.4.1 全局最后异常事件处理程序24.4.2 HttpApplication基类24.5 应用程序状态与会话状态差别24.5.1 维护应用程序级的状态数据24.5.2 修改应用程序数据24.5.3 处理Web应用程序的关闭24.6 使用应用程序缓存24.6.1 使用数据缓存24.6.2 修改*.aspx文件24.7 维护会话数据24.8 cookie24.8.1 创建cookie24.8.2 读取传入的cookie数据24.9 使用Web.config配置ASP.NET应用程序24.9.1 通过< trace >启用跟踪24.9.2 通过< customErrors >自定义错误输出24.9.3 通过< sessionState >存储状态24.9.4 ASP.NET 2.0站点管理工具24.10 配置继承24.11 小结第25章 XML Web服务25.1 XML Web服务的作用25.1.1 XML Web服务的优点25.1.2 定义XML Web服务客户端25.1.3 XML Web服务的基础25.1.4 概述XML Web服务发现25.1.5 概述XML Web服务描述25.1.6 概述传输协议25.2 .NET XML Web服务命名空间25.3 手动构建XML Web服务25.3.1 使用WebDev.WebServer.exe测试XML Web服

务25.3.2 使用IIS测试XML Web服务25.3.3 查看WSDL合约25.4 自动生成测试页面25.5 使用Visual Studio 2005构建XML Web服务25.6 WebService基类的作用25.7 [WebService]特性25.7.1 Namespace和Description属性的作用25.7.2 Name属性25.8 [WebServiceBinding]特性25.8.1 忽略BP 1.1一致性验证25.8.2 禁用BP 1.1一致性验证25.9 [WebMethod]特性25.9.1 通过Description属性为Web方法归档25.9.2 通过MessageName属性避免WSDL名称冲突25.9.3 用EnableSession属性构建有状态的Web服务25.10 探索WSDL25.10.1 定义WSDL文档25.10.2 < types >元素25.10.3 < message >元素25.10.4 < portType >元素25.10.5 < binding >元素25.10.6 < service >元素25.11 再谈XML Web服务报文协议25.11.1 HTTP GET和HTTP POST绑定25.11.2 SOAP绑定25.12 wsdl.exe命令行的效用25.12.1 将WSDL转换成服务器端XML Web服务框架25.12.2 将WSDL转换为客户端代理类25.13 查看代理服务器代码25.13.1 默认的构造函数25.13.2 同步调用支持25.13.3 异步调用支持25.13.4 构建客户端应用程序25.14 使用Visual Studio 2005生成代理类25.15 从Web方法公开自定义类型25.15.1 公开数组25.15.2 公开结构25.15.3 公开ADO.NET数据集25.15.4 Windows窗体客户端25.15.5 客户端类型代理25.16 发现服务协议 (UDDI) 25.17 小结第六部分 .NET 3.0扩展编程第26章 建立.NET 3.0编程环境26.1 .NET 3.0技术介绍26.2 C# 3.0和LINQ技术介绍26.3 欢迎使用.NET 3.026.4 安装.NET Framework 3.0运行库组件26.5 安装Windows软件开发包26.5.1 选择安装项26.5.2 研究SDK的内容26.6 安装Visual Studio “ Orcas ” 开发工具26.6.1 安装WPF和WCF项目支持26.6.2 安装Visual Studio 2005为WF提供的扩展26.7 安装C# 3.0和LINQ社区预览版26.8 小结第27章 WPF介绍27.1 WPF背后的动机27.1.1 通过XAML将关注点分离27.1.2 提供优化的呈现模型27.2 WPF程序集详解27.2.1 Application类的作用27.2.2 Window类的作用27.3 创建 (不使用XAML的) WPF应用程序27.3.1 扩展Window类27.3.2 创建简单的用户界面27.4 XAML介绍27.4.1 用XAML定义MainWindow27.4.2 用XAML定义应用对象27.4.3 通过msbuild.exe处理XAML文件27.5 将标记转换为.NET程序集27.5.1 XAML到C#代码的映射27.5.2 BAML的作用27.5.3 XAML到程序集的过程摘要27.6 使用代码隐藏文件实现的关注点的分离27.7 在XamlPad中练习使用XAML27.8 使用Visual Studio “ Orcas ” 创建WPF应用程序27.9 使用微软表达式交互设计器生成XAML27.10 使用面板控制内容布局27.10.1 在Canvas面板中放置内容27.10.2 在WrapPanel面板中放置内容27.10.3 在StackPanel面板内放置内容27.10.4 在Grid面板中放置内容27.10.5 在DockPanel面板中放置内容27.10.6 使用嵌套的面板创建窗体的框架27.11 WPF控件27.11.1 配置WPF控件27.11.2 使用WPF控件属性27.11.3 处理WPF控件事件27.11.4 应用控件样式27.12 WPF图形显示服务简介27.12.1 WPF图形服务详解27.12.2 使用基本的形状27.12.3 WPF动画服务介绍27.12.4 使用微软Expression图形设计器生成XAML27.13 XAML浏览器应用程序简介27.14 小结第28章 WCF介绍28.1 WCF背后的动机28.2 探究WCF核心程序集28.3 WCF基础28.3.1 WCF契约28.3.2 WCF绑定28.3.3 WCF地址28.4 构建完整的WCF应用程序28.4.1 组成WCF应用程序的相关程序集28.4.2 契约的定义与实现28.5 承载WCF服务28.5.1 指明ABC28.5.2 ServiceHost类型的功能28.6 < system.ServiceModel >元素的细节28.7 与WCF服务进行通信28.7.1 使用svcutil.exe生成代理代码28.7.2 使用Visual Studio 2005生成代理代码28.8 WCF的数据类型表示28.8.1 更新ICarOrder服务契约28.8.2 对CarOrderServiceClient程序集重新编码28.8.3 使用XmlSerializer进行数据编码28.8.4 使用二进制格式传输数据28.9 使用服务配置编辑器生成WCF配置文件28.10 小结第29章 WF介绍29.1 WF背后的动机29.2 WF的积木块29.2.1 WF中的集成服务29.2.2 WF活动初览29.2.3 顺序工作流和状态机工作流的作用29.2.4 深入探讨工作流29.3 WF程序集和核心命名空间29.4 建造一个启用工作流的简单应用29.4.1 研究初始工作流的代码29.4.2 添加Code活动29.4.3 添加While活动29.4.4 研究WF引擎承载代码29.4.5 添加定制的起初参数29.5 在工作流中调用Web服务29.6 构建可重用的WF代码库29.6.1 编写简单的工作流29.6.2 创建启用工作流的Windows Forms应用程序29.7 关于自定义活动的简要说29.8 小结第30章 C# 3.0的语言功能30.1 使用C# 3.0命令行编译器30.2 理解隐式类型化的局部变量30.2.1 隐式类型化变量的限制30.2.2 隐式类型化的局部数组30.2.3 隐式数据类型化的最后注意事项30.3 理解扩展方法30.3.1 定义扩展方法30.3.2 在实例层次上调用扩展方法30.3.3 静态调用扩展方法30.3.4 导入定义了扩展方法的类型30.3.5 构建和使用扩展库30.4 理解对象初始化器30.4.1 使用初始化语法调用自定义构造函数30.4.2 初始化内部类型30.4.3 理解集合的初始化30.5 理解匿名类型30.5.1 匿名类型的内部表示方式30.5.2 方法ToString()和方法GetHashCode()的实现30.5.3 匿名类型的相等语义30.5.4 包含匿名类型的匿名类型30.6 理解Lambda表达式的角色30.6.1 Lambda表达式是更好的匿名方法30.6.2 剖析Lambda表达式30.6.3 Lambda表达式的两种风格30.6.4 使用Lambda表达式重新编写CarDelegate示例30.6.5 含有多个 (

或零个)参数的Lambda表达式30.7 小结第31章 LINQ介绍31.1 定义LINQ的作用31.2 核心LINQ程序集31.3 LINQ查询表达式初览31.3.1 重访隐型局部变量31.3.2 重访扩展方法31.4 用LINQ查询泛型集合31.4.1 定义LINQ查询31.4.2 重访匿名类型31.5 使用LINQ查询非泛型集合31.6 查询运算符的内部表示31.6.1 用查询运算符建立查询表达式(复习)31.6.2 使用Sequence类型和Lambda表达式来建立查询表达式31.6.3 使用Sequence类型和匿名方法来建立查询表达式31.6.4 用Sequence类型和原始代理建立查询表达式31.7 研究LINQ查询运算符31.8 构建LINQ查询表达式31.8.1 基本的选择语法31.8.2 获取数据子集31.8.3 逆转结果集的顺序31.8.4 对表达式进行排序31.8.5 转换查询结果以及转换延缓执行的作用31.9 使用LINQ到SQL来查询关系数据库31.9.1 实体类的作用31.9.2 DataContext类型的作用31.9.3 一个LINQ到SQL的简单例子31.9.4 建立强类型的DataContext31.9.5 详细介绍[Table]特性和[Column]特性31.10 使用sqlmetal.exe生成实体类31.10.1 研究生成的实体类31.10.2 使用实体类来定义关系31.10.3 强类型的DataContext31.10.4 针对生成的类型来编程31.11 使用Visual Studio 2005建立实体类31.11.1 插入新项31.11.2 更新现有项31.11.3 删除现有项31.12 使用LINQ到XML操作XML文档31.12.1 System.Xml.XLinq命名空间31.12.2 以编程方式创建XML文档31.12.3 装载并分析XML内容31.13 在内存文档中导航31.13.1 使用LINQ到XML来选择元素31.13.2 在XML文档中修改数据31.14 小结

<<C#与.NET 3.0高级程序设计>>

媒体关注与评论

“问：学习C#最好的书是哪一本？”

答：最受推崇的是Andrew Troelsen的Pro C# with.NET3.0。

”——MSDN论坛 “本书极为全面、细致深入地探讨了C#与.NET 2.0框架的各种特性。其中对CIL的介绍和运用尤为精彩，超过了任何其他文章和图书。绝对值得拥有。

”——Slashdot网站 “这是一本不仅应该拥有，而且应该放在键盘旁边随时翻查的C#和.NET图书。

”

<<C#与.NET 3.0高级程序设计>>

编辑推荐

《C#与.NET 3.0高级程序设计(特别版)》是Amazon超级畅销书，C#圣经级著作，全面涵盖C# 3.0和.NET 3.0平台，包括LINQ、WPF、WCF和WF，用中间语言深入揭示各语言特性，让你知其然，更知其所以然，国内多位微软MVP联手翻译。

C#语言作为.NET平台上的第一语言。

自发布以来不断增强，已经成为目前功能最强大的通用语言之一。

《C#与.NET 3.0高级程序设计》是被誉为“C#圣经”的经典巨著，因语言生动流畅、剖析深入、涵盖全面而广受推崇，畅销不衰。

曾经获得Referenceware编程图书大奖。

并入围Jolt大奖提名。

书中探讨了C#语言和.NET平台的各种特性，包括重载运算符、指针、泛型等高级功能和CIL、多线程、远程处理、GDI+、Windows窗体、ASP.NET、ADO.NET等技术，不少概念都通过IL代码透视其背后的本质。

使你知其然。

更知其所以然。

新版还专门用一个部分六章分别讲述了C#3.0新功能和.NET 3.0的新特性，包括LINQ以及相关技术、WPF、WCF和WF。

与同类图书不同，全书由世界级C#专家Andrew Troelsen以一人之力完成，因此写作思路和布局谋篇都独具匠心。

中文版由国内多位微软MVP联手译出。

强大的译者阵容有力地保证了权威原著的重现。

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>