

<<精通C# (第6版) >>

图书基本信息

书名：<<精通C# (第6版) >>

13位ISBN编号：9787115321817

10位ISBN编号：7115321817

出版时间：2013-7

出版时间：人民邮电出版社

作者：[美] Andrew Troelsen

译者：姚琪琳,朱 晔,肖 逵,张大磊,王少葵,范 睿 等

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>

<<精通C# (第6版) >>

内容概要

本书是C# 领域久负盛名的经典著作，深入全面地讲解了C# 编程语言和.NET 平台的核心内容，并结合大量示例剖析相关概念。

全书分为八部分：C# 和.NET 平台、C# 核心编程结构、C# 面向对象编程、高级C# 编程结构、用.NET 程序集编程、.NET 基础类库、WPF 和ASP.NET Web Forms。

第6 版是对第5 版的进一步更新和完善，内容涵盖了最先进的.NET 编程技术和技巧，并准确呈现出C# 编程语言的最新变化和.NET 4.5 Framework 的新特性。

本书由微软C# MVP Andrew Troelsen 编写，第6 版专门针对C# 5.0 和.NET 4.5 进行了细致入微的修订，是各层次.NET 开发人员的必读之作。

<<精通C# (第6版) >>

作者简介

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Intertech的合伙人和副总裁，该公司的客户包括微软、霍尼韦尔、美国国家航空航天局等。

他曾是MSDN网站和MacTech网站.NET技术帖专家，并经常在业界主要技术会议上发表演讲和开设技术讲座。

他还是公认的C#和.NET畅销技术书作家。

除本书外，他还著有Pro Expression Blend 4、Pro VB 2008 and the .NET 3.5 Platform和COM and .NET Interoperability等近二十部.NET技术方面的著作。

书籍目录

目 录	第一部分 C#与.NET平台	第1章 .NET之道	21.1 初识.NET平台	21.2 .NET平台构造
			31.2.1 基础类库的作用	31.2.2 C#的优点
			41.2.3 托管代码与非托管代码	51.3 其他支持.NET的编程语言
			51.4 .NET程序集概览	71.4.1 CIL的作用
			81.4.2 .NET类型元数据的作用	101.4.3 程序集清单的作用
			111.5 CTS	121.5.1 CTS类类型
			121.5.2 CTS接口类型	121.5.3 CTS结构类型
			131.5.4 CTS枚举类型	131.5.5 CTS委托类型
			131.5.6 CTS类型成员	141.5.7 内建的CTS数据类型
			141.6 CLS	151.7 CLR
			161.8 程序集/命名空间/类型的区别	171.8.1 Microsoft根命名空间的作用
			201.8.2 以编程方式访问命名空间	201.8.3 引用外部程序集
			211.9 使用ildasm.exe探索程序集	221.9.1 查看CIL代码
			231.9.2 查看类型元数据	231.9.3 查看程序集元数据(即清单)
			241.10 .NET的平台无关性	241.11 Windows 8应用程序简介
			251.11.1 构建Windows 8应用程序	261.11.2 .NET在Windows 8中的作用
			271.12 小结	28第2章 构建C#应用程序
			292.1 .NET Framework 4.5 SDK的作用	292.2 用csc.exe构建C#应用程序
			302.2.1 指定输入输出目标	312.2.2 引用外部程序集
			322.2.3 引用多个外部程序集	332.2.4 编译多个源文件
			332.2.5 使用C#响应文件	342.3 使用Notepad++构建.NET应用程序
			352.4 使用SharpDevelop构建.NET应用程序	362.5 使用Visual C# Express构建.NET应用程序
			382.6 使用Visual Studio构建.NET应用程序	392.6.1 Visual Studio的独特功能
			392.6.2 使用New Project对话框指向.NET Framework	402.6.3 解决方案资源管理器
			402.6.4 Class View工具	422.6.5 Object Browser工具
			432.6.6 集成对代码重构的支持	432.6.7 代码扩展和围绕技术
			452.6.8 可视化Class Designer	472.6.9 集成的.NET Framework 4.5 SDK文档系统
			502.7 小结	51第二部分 C#核心编程结构
			第3章 C#核心编程结构	543.1 一个简单的C#程序
			543.1.1 Main()方法的其他形式	553.1.2 指定应用程序错误代码
			563.1.3 处理命令行参数	573.1.4 使用Visual Studio指定命令行参数
			593.2 有趣的题外话	: System.Environment类的其他成员
			593.3 System.Console类	613.3.1 使用Console类进行基本的输入和输出
			613.3.2 格式化控制台输出	623.3.3 格式化数值数据
			633.3.4 在控制台应用程序外格式化数值数据	643.4 系统数据类型和相应的C#关键字
			643.4.1 变量声明和初始化	653.4.2 内建数据类型与new操作符
			673.4.3 数据类型类的层次结构	673.4.4 数值数据类型的成员
			693.4.5 System.Boolean的成员	693.4.6 System.Char的成员
			693.4.7 从字符串数据中解析数值	703.4.8 System.DateTime和System.TimeSpan
			703.4.9 System.Numerics.dll程序集	713.5 使用字符串数据
			723.5.1 基本的字符串操作	733.5.2 字符串拼接
			733.5.3 转义字符	743.5.4 定义逐字字符串
			753.5.5 字符串和相等性	753.5.6 字符串是不可变的
			763.5.7 System.Text.StringBuilder类型	773.6 窄化和宽化数据类型转换
			783.6.1 checked关键字	803.6.2 设定项目级别的溢出检测
			813.6.3 unchecked关键字	823.7 隐式类型本地变量
			823.7.1 隐式类型变量的限制	843.7.2 隐式类型数据是强类型数据
			843.7.3 隐式类型本地变量的用途	853.8 C#迭代结构
			863.8.1 for循环	863.8.2 foreach循环
			873.8.3 while和do/while循环结构	873.9 条件结构和关系/相等操作符
			883.9.1 if/else语句	883.9.2 关系/相等操作符
			883.9.3 逻辑操作符	893.9.4 switch语句
			893.10 小结	91第4章 C#核心编程结构
			924.1 方法和参数修饰符	924.1.1 默认的参数传递行为
			934.1.2 out修饰符	944.1.3 ref修饰符
			954.1.4 params修饰符	964.1.5 定义可选参数
			974.1.6 使用命名参数调用方法	984.1.7 成员重载
			994.2 C#数组	1014.2.1 C#数组初始化语法
			1024.2.2 隐式类型本地数组	1034.2.3 定义object数组
			1034.2.4 使用多维数组	1044.2.5 数组作为参数(和返回值)
			1054.2.6 System.Array基类	1064.3 枚举类型
			1074.3.1 控制枚举的底层存储	1084.3.2 声明枚举变量
			1094.3.3 System.Enum类型	1104.3.4 动态获取枚举的名称/值对
			1104.4 结构类型	1124.5 值类型和引用类型
			1154.5.1 值类型、引用类型和赋值操作符	1164.5.2 包含引用类型的值类型
			1174.5.3 按值传递引用类型	1194.5.4 按引用传递引用类型
			1204.5.5 值类型和引用类型:最后的细节	1214.6 C#可空类型
			1224.6.1 使用可空类型	1234.6.2 ??操作符
			1244.7 小结	124第3部分 C#面向对象编程
			第5章 封装	1265.1 C#类类型
			1265.2 构造函数	1295.2.1 默认构造函数的作用
			1295.2.2 定义自定义的构造函数	1305.2.3 再谈默认构造函数
			1315.3 this关键字的作用	1325.3.1 使用this进行串联构造函数调用
			1335.3.2 观察构造函数流程	1355.3.3 再谈

可选参数 1375.4 static关键字 1385.4.1 定义静态数据 1385.4.2 定义静态方法 1405.4.3 定义静态构造函数 1415.4.4 定义静态类 1435.5 定义OOP的支柱 1445.5.1 封装的作用 1445.5.2 继承的作用 1445.5.3 多态的作用 1465.6 C#访问修饰符 1475.6.1 默认的访问修饰符 1485.6.2 访问修饰符和嵌套类型 1485.7 第一个支柱：C#的封装服务 1495.7.1 使用传统的访问方法和修改方法执行封装 1495.7.2 使用.NET属性进行封装 1515.7.3 使用类的属性 1545.7.4 只读和只写属性 1555.7.5 静态属性 1565.8 自动属性 1565.8.1 与自动属性交互 1585.8.2 关于自动属性和默认值 1585.9 对象初始化语法 1605.9.1 使用初始化语法调用自定义构造函数 1615.9.2 初始化内部类型 1625.10 常量数据 1635.10.1 只读字段 1645.10.2 静态只读字段 1655.11 分部类型 1655.12 小结 167第6章 继承和多态 1686.1 继承的基本机制 1686.1.1 指定既有类的父类 1696.1.2 多个基类 1706.1.3 sealed关键字 1716.2 回顾Visual Studio类关系图 1726.3 OOP的第二个支柱：继承 1736.3.1 使用base关键字控制基类的创建 1746.3.2 家族的秘密：protected关键字 1766.3.3 增加密封类 1776.4 包含/委托编程 1786.5 OOP的第三个支柱：C#的多态支持 1806.5.1 virtual和override关键字 1816.5.2 使用Visual Studio IDE重写虚方法 1836.5.3 密封虚成员 1846.5.4 抽象类 1846.5.5 构建多态接口 1866.5.6 成员投影 1896.6 基类/派生类的转换规则 1916.6.1 C#的as关键字 1926.6.2 C#的is关键字 1936.7 超级父类：System.Object 1936.7.1 重写System.Object.ToString() 1966.7.2 重写System.Object.Equals() 1966.7.3 重写System.Object.GetHashCode() 1976.7.4 测试修改后的Person类 1986.7.5 System.Object的静态成员 1996.8 小结 199第7章 结构化异常处理 2007.1 错误、bug与异常 2007.2 .NET异常处理的作用 2017.2.1 .NET异常处理的四要素 2027.2.2 System.Exception基类 2027.3 最简单的例子 2037.3.1 引发普通的异常 2057.3.2 捕获异常 2067.4 配置异常的状态 2077.4.1 TargetSite属性 2077.4.2 StackTrace属性 2087.4.3 HelpLink属性 2087.4.4 Data属性 2097.5 系统级异常 2117.6 应用程序级异常 2117.6.1 构建自定义异常，第一部分 2127.6.2 构建自定义异常，第二部分 2137.6.3 构建自定义异常，第三部分 2147.7 处理多个异常 2157.7.1 通用的catch语句 2177.7.2 再次引发异常 2187.7.3 内部异常 2187.7.4 finally块 2197.8 谁在引发什么异常 2207.9 未处理异常的后果 2207.10 使用Visual Studio调试未处理的异常 2217.11 小结 222第8章 接口 2238.1 接口类型 2238.2 定义自定义接口 2268.3 实现接口 2278.4 在对象级别调用接口成员 2298.4.1 获取接口引用：as关键字 2308.4.2 获取接口引用：is关键字 2308.5 接口作为参数 2318.6 接口作为返回值 2338.7 接口类型数组 2338.8 使用Visual Studio实现接口 2348.9 显式接口实现 2358.10 设计接口层次结构 2388.11 构建可枚举类型(IEnumerable和IEnumerator) 2418.11.1 用yield关键字构建迭代器方法 2438.11.2 构建命名迭代器 2448.12 构建可克隆的对象(ICloneable) 2458.13 构建可比较的对象(IComparable) 2498.13.1 指定多个排序顺序 2528.13.2 自定义属性、自定义排序类型 2538.14 小结 253第四部分 高级C#编程结构第9章 集合与泛型 2569.1 集合类的动机 2569.1.1 System.Collections命名空间 2579.1.2 System.Collections.Specialized命名空间 2599.2 非泛型集合的问题 2609.2.1 性能问题 2609.2.2 类型安全问题 2639.2.3 初识泛型集合 2659.3 泛型类型参数的作用 2669.3.1 为泛型类/结构指定类型参数 2679.3.2 为泛型成员指定类型参数 2689.3.3 为泛型接口指定类型参数 2699.4 System.Collections.Generic命名空间 2709.4.1 集合初始化语法 2719.4.2 使用List类 2729.4.3 使用Stack类 2739.4.4 使用Queue类 2749.4.5 使用SortedSet类 2759.5 System.Collections.ObjectModel命名空间 2779.6 创建自定义泛型方法 2799.7 创建自定义泛型结构和类 2829.8 类型参数的约束 2849.8.1 使用where关键字的示例 2849.8.2 操作符约束的不足 2859.9 小结 286第10章 委托、事件和Lambda表达式 28710.1 .NET委托类型 28710.1.1 在C#中定义委托类型 28810.1.2 System.MulticastDelegate 与System.Delegate基类 29010.2 最简单的委托示例 29110.3 使用委托发送对象状态通知 29310.3.1 支持多路广播 29610.3.2 从委托的调用列表中移除成员 29710.3.3 方法组转换语法 29810.4 泛型委托 30010.5 C#事件 30310.5.1 event关键字 30410.5.2 揭开事件的神秘面纱 30510.5.3 监听传入的事件 30610.5.4 使用Visual Studio简化事件注册 30710.5.5 创建自定义的事件参数 30810.5.6 泛型EventHandler委托 30910.6 C#匿名方法 31010.7 Lambda表达式 31310.7.1 剖析Lambda表达式 31510.7.2 使用多个语句处理参数 31610.7.3 含有多个(或零个)参数的Lambda表

达式 31710.7.4 使用Lambda表达式重新编写CarEvents示例 31810.8 小结 319第11章 高级C#语言
 特性 32011.1 索引器方法 32011.1.1 使用字符串值索引对象 32211.1.2 重载索引器方法
 32311.1.3 多维的索引器 32311.1.4 在接口类型上定义索引器 32411.2 操作符重载 32511.2.1
 重载二元操作符 32511.2.2 +=与-=操作符 32711.2.3 重载一元操作符 32811.2.4 重载相等操作符
 32911.2.5 重载比较操作符 32911.2.6 操作符重载的最后思考 33011.3 自定义类型转换
 33111.3.1 回顾:数值转换 33111.3.2 回顾:相关的类类型间的转换 33111.3.3 创建自定义转换
 例程 33211.3.4 Square类型的其他显式转换 33411.3.5 定义隐式转换例程 33511.4 扩展方法
 33611.4.1 定义扩展方法 33611.4.2 在实例层次上调用扩展方法 33711.4.3 导入扩展方法
 33811.4.4 扩展方法的智能感知 33911.4.5 扩展实现了指定接口的类型 33911.5 匿名类型
 34011.5.1 定义匿名类型 34111.5.2 匿名类型的内部表示方式 34211.5.3 方法ToString()
 和GetHashCode()的实现 34311.5.4 匿名类型的相等语义 34411.5.5 包含匿名类型的匿名类型
 34511.6 指针类型 34611.6.1 unsafe关键字 34711.6.2 *和&操作符 34811.6.3 不安全(与安全)
 交换功能 34911.6.4 通过指针访问字段 35011.6.5 stackalloc关键字 35011.6.6 使用fixed关键字固
 定类型 35111.6.7 sizeof关键字 35211.7 小结 352第12章 LINQ to Object 35312.1 LINQ特有的
 编程结构 35312.1.1 隐式类型本地变量 35412.1.2 对象和集合初始化语法 35412.1.3 Lambda表达
 式 35512.1.4 扩展方法 35512.1.5 匿名类型 35612.2 LINQ的作用 35612.2.1 LINQ表达式是强
 类型的 35712.2.2 核心LINQ程序集 35712.3 将LINQ查询应用于原始数组 35812.3.1 再一次,不
 使用LINQ 35912.3.2 反射LINQ结果集 36012.3.3 LINQ和隐式类型本地变量 36112.3.4 LINQ和
 扩展方法 36212.3.5 延迟执行的作用 36312.3.6 立即执行的作用 36412.4 返回LINQ查询的结果
 36512.5 将LINQ查询应用到集合对象 36712.5.1 访问包含的子对象 36712.5.2 将LINQ查询应用
 于非泛型集合 36812.5.3 使用OfType()筛选数据 36912.6 C# LINQ查询操作符 36912.6.1 基本的
 选择语法 37112.6.2 获取数据子集 37112.6.3 投影新数据类型 37212.6.4 使用Enumerable获取总
 数 37312.6.5 反转结果集 37312.6.6 对表达式进行排序 37412.6.7 维恩图工具 37412.6.8 移除
 重复 37512.6.9 LINQ聚合操作 37612.7 LINQ查询语句的内部表示 37612.7.1 用查询操作符建立
 查询表达式(复习) 37712.7.2 使用Enumerable类型和Lambda表达式来建立查询表达式 37712.7.3 使
 用Enumerable类型和匿名方法来建立查询表达式 37912.7.4 用Enumerable类型和原始委托建立查询表
 达式 37912.8 小结 380第13章 对象的生命周期 38113.1 类、对象和引用 38113.2 对象生命周
 期的基础 38213.2.1 CIL的new指令 38313.2.2 将对象引用设置为空 38413.3 应用程序根的作用
 38513.4 对象的代 38613.5 .NET 1.0至.NET 3.5的并发垃圾回收 38713.6 .NET 4.0及后续版本
 38713.7 System.GC类型 38813.8 构建可终结对象 39113.8.1 重写System.Object.Finalize()
 39213.8.2 终结过程的细节 39313.9 构建可处置对象 39413.10 构建可终结类型和可处置类型
 39713.11 延迟对象实例化 40013.12 小结 403第五部分 用.NET程序集编程第14章 .NET程序
 集入门 40614.1 定义自定义命名空间 40614.1.1 使用完全限定名解决命名冲突 40814.1.2 使用
 别名解决命名冲突 40914.1.3 创建嵌套的命名空间 41014.1.4 Visual Studio 的默认命名空间
 41114.2 .NET程序集的作用 41214.2.1 程序集促进代码重用 41214.2.2 程序集确定类型边界
 41214.2.3 程序集是可版本化的单元 41214.2.4 程序集是自描述的 41314.2.5 程序集是可配置的
 41314.3 .NET程序集的格式 41314.3.1 Windows文件首部 41314.3.2 CLR文件首部 41414.3.3
 CIL代码、类型元数据和程序集清单 41514.3.4 可选的程序集资源 41514.4 构建和使用自定义类
 库 41614.4.1 清单 41814.4.2 CIL 42114.4.3 类型元数据 42114.4.4 构建C#客户端应用程序
 42214.4.5 构建Visual Basic客户端应用程序 42414.4.6 实现跨语言继承 42514.5 私有程序集
 42514.5.1 私有程序集的标识 42614.5.2 探测过程 42614.5.3 配置私有程序集 42714.5.4
 App.Config文件 42814.6 共享程序集 43014.6.1 全局程序集缓存 43014.6.2 强名称 43114.6.3
 在命令行生成强名称 43214.6.4 使用Visual Studio为程序集赋予强名称 43414.6.5 在GAC中安装
 强名称的程序集 43614.7 使用共享程序集 43714.8 配置共享程序集 43914.8.1 冻结当前的共享
 程序集 43914.8.2 构建共享程序集2.0.0.0版本 44014.8.3 动态重定向到共享程序集的特定版本
 44114.9 发行者策略程序集 44314.10 元素 44414.11 System.Configuration命名空间 44614.12
 配置文件架构文档 44714.13 小结 448第15章 类型反射、晚期绑定和基于特性的编程 44915.1

类型元数据的必要性 44915.1.1 查看(部分)EngineState枚举的元数据 45015.1.2 查看(部分)Car类型的元数据 45115.1.3 研究TypeRef 45215.1.4 记录定义的程序集 45215.1.5 记录引用的程序集 45315.1.6 记录字符串字面量 45315.2 反射 45415.2.1 System.Type类 45415.2.2 使用System.Object.GetType()得到Type引用 45515.2.3 使用typeof()得到Type引用 45515.2.4 使用System.Type.GetType()得到Type引用 45515.3 构建自定义的元数据查看器 45615.3.1 反射方法 45615.3.2 反射字段和属性 45715.3.3 反射实现的接口 45715.3.4 显示其他信息 45815.3.5 实现Main() 45815.3.6 反射泛型类型 46015.3.7 反射方法参数和返回值 46015.4 动态加载程序集 46115.5 反射共享程序集 46415.6 晚期绑定 46515.6.1 System.Activator类 46615.6.2 调用没有参数的方法 46715.6.3 调用有参数的方法 46815.7 .NET特性的作用 46915.7.1 特性的使用者 47015.7.2 在C#中使用特性 47015.7.3 C#特性简化符号 47115.7.4 为特性指定构造参数 47215.7.5 Obsolete特性 47215.8 构建自定义特性 47315.8.1 应用自定义特性 47315.8.2 命名属性语法 47415.8.3 限制特性使用 47415.9 程序集级别特性 47515.10 使用早期绑定反射特性 47715.11 使用晚期绑定反射特性 47815.12 反射、晚期绑定和自定义特性的使用背景 47915.13 构建可扩展的应用程序 48015.13.1 构建CommonSnappable-Types.dll 48015.13.2 构建C#插件 48115.13.3 构建Visual Basic插件 48215.13.4 构建可扩展的Windows Forms应用程序 48215.14 小结 485第16章 动态类型和动态语言运行时 48616.1 dynamic关键字的作用 48616.1.1 调用动态声明的数据的成员 48816.1.2 Microsoft.CSharp.dll程序集的作用 48916.1.3 dynamic关键字的作用域 49016.1.4 dynamic关键字的限制 49116.1.5 dynamic关键字的实际用途 49116.2 DLR的作用 49216.2.1 表达式树的作用 49216.2.2 System.Dynamic命名空间的作用 49316.2.3 表达式树的动态运行时查找 49316.3 使用动态类型简化后期绑定调用 49416.4 使用动态数据简化COM互操作 49716.4.1 主互操作程序集的作用 49816.4.2 嵌入互操作元数据 49916.4.3 普通COM互操作的难点 50016.5 使用C# 动态数据进行COM互操作 50016.6 不使用C# 动态数据进行COM互操作 50416.7 小结 505第17章 进程、应用程序域和对象上下文 50617.1 Windows进程的作用 50617.2 .NET平台下与进程进行交互 50817.2.1 列举运行中的进程 51017.2.2 特定的进程 51117.2.3 进程的线程集合 51117.2.4 进程中的模块集合 51317.2.5 以编程方式启动或结束进程 51417.2.6 使用ProcessStartInfo类控制进程的启动 51517.3 .NET应用程序域 51617.4 与默认应用程序域进行交互 51817.4.1 枚举加载的程序集 51917.4.2 接收程序集加载通知 52017.5 创建新的应用程序域 52117.5.1 在自定义应用程序域中加载程序集 52217.5.2 以编程方式卸载应用程序域 52317.6 对象上下文边界 52517.6.1 上下文灵活和上下文绑定类型 52517.6.2 定义上下文绑定对象 52617.6.3 研究对象的上下文 52617.7 进程、应用程序域和上下文小结 52817.8 小结 528第18章 CIL和动态程序集的作用 52918.1 学习CIL语法的原因 52918.2 CIL指令、特性和操作码 53018.2.1 CIL指令的作用 53018.2.2 CIL特性的作用 53018.2.3 CIL操作码的作用 53118.2.4 区别CIL操作码和CIL助记符 53118.3 入栈和出栈: CIL基于栈的本质 53218.4 正反向工程 53318.4.1 CIL代码标签的作用 53618.4.2 与CIL交互: 修改*.il文件 53618.4.3 使用ilasm.exe 编译CIL代码 53718.4.4 peverify.exe的作用 53818.5 CIL指令和特性 53918.5.1 在CIL中指定外部引用程序集 53918.5.2 在CIL中定义当前程序集 53918.5.3 在CIL中定义命名空间 54018.5.4 在CIL中定义类类型 54018.5.5 在CIL中定义和实现接口 54118.5.6 在CIL中定义结构 54218.5.7 在CIL中定义枚举 54218.5.8 在CIL中定义泛型 54318.5.9 编译CILTypes.il文件 54318.6 .NET基础类库、C#和CIL数据类型的映射 54418.7 在CIL中定义类型成员 54418.7.1 在CIL中定义数据字段 54518.7.2 在CIL中定义类型的构造函数 54518.7.3 在CIL中定义属性 54618.7.4 定义成员参数 54618.8 剖析CIL操作码 54718.8.1 .maxstack指令 54818.8.2 在CIL中声明本地变量 54918.8.3 在CIL中映射参数到本地变量 55018.8.4 this隐式引用 55018.8.5 在CIL中使用循环结构 55118.9 使用CIL构建.NET程序集 55118.9.1 构建CILCars.dll 55118.9.2 构建CILCarClient.exe 55418.10 动态程序集 55518.10.1 System.Reflection.Emit命名空间 55618.10.2 System.Reflection.Emit.ILGenerator的作用 55718.10.3 产生动态的程序集 55718.10.4 产生程序集和模块集 55918.10.5 ModuleBuilder类型的作用 56018.10.6 产生HelloClass类型和字符串成员变量 56118.10.7 产生构造函数 56118.10.8 产生SayHello()方法 56218.10.9 使用动态产生的程序集 56318.11 小结 564

第六部分 .NET基础类库第19章 多线程、并行和异步编程 56619.1 进程、应用程序域、上下文及线程之间的关系 56619.1.1 并发问题 56719.1.2 线程同步的作用 56819.2 .NET委托的简短回顾 56819.3 委托的异步性 57019.3.1 BeginInvoke()和EndInvoke()方法 57019.3.2 System.IAsyncResult接口 57019.4 异步调用方法 57119.4.1 同步调用线程 57219.4.2 AsyncCallback委托的作用 57319.4.3 AsyncResult类的作用 57519.4.4 传递和接收自定义状态数据 57519.5 System.Threading命名空间 57619.6 System.Threading.Thread类 57719.6.1 获得当前执行线程的统计信息 57819.6.2 Name属性 57919.6.3 Priority属性 57919.7 手工创建次线程 58019.7.1 使用ThreadStart委托 58019.7.2 使用ParameterizedThreadStart委托 58219.7.3 AutoResetEvent类 58319.7.4 前台线程和后台线程 58419.8 并发问题 58519.8.1 使用C#的lock关键字进行同步 58819.8.2 使用System.Threading.Monitor类型进行同步 58919.8.3 使用System.Threading.Interlocked类型进行同步 59019.8.4 使用[Synchronization]特性进行同步 59119.9 使用TimerCallback编程 59219.10 CLR线程池 59319.11 使用任务并行库进行并行编程 59519.11.1 任务并行库API 59519.11.2 Parallel类的作用 59619.11.3 使用Parallel类的数据并行 59619.11.4 在次线程中访问UI元素 59819.11.5 Task类 59919.11.6 处理取消请求 59919.11.7 使用并行类的任务并行 60119.12 并行LINQ查询(PLINQ) 60319.12.1 使用PLINQ查询 60419.12.2 取消PLINQ查询 60419.13 .NET 4.5下的异步调用 60519.13.1 C# async和await关键字初探 60619.13.2 异步方法的命名约定 60719.13.3 返回void的异步方法 60919.13.4 具有多个await的异步方法 60919.13.5 用async/await改进AddWithThreads示例 61019.14 小结 611第20章 文件输入输出和对象序列化 61220.1 研究System.IO命名空间 61220.2 Directory(Info)和File(Info)类型 61320.3 使用DirectoryInfo类型 61420.3.1 使用DirectoryInfo类型枚举出文件 61620.3.2 使用DirectoryInfo类型创建子目录 61620.4 使用Directory类型 61720.5 使用DriveInfo类类型 61820.6 使用FileInfo类 61920.6.1 FileInfo.Create()方法 62020.6.2 FileInfo.Open()方法 62120.6.3 FileInfo.OpenRead()和FileInfo.OpenWrite()方法 62220.6.4 FileInfo.OpenText()方法 62220.6.5 FileInfo.CreateText()和FileInfo.AppendText()方法 62320.7 使用File类型 62320.8 Stream抽象类 62520.9 使用StreamWriter和StreamReader类型 62720.9.1 写文本文件 62820.9.2 读文本文件 62820.9.3 直接创建StreamWriter/StreamReader类型 62920.10 使用StringWriter和StringReader类型 63020.11 使用BinaryWriter和BinaryReader 63120.12 以编程方式“观察”文件 63220.13 对象序列化 63420.14 为序列化配置对象 63720.14.1 定义可序列化的类型 63720.14.2 公共字段、私有字段和公共属性 63820.15 选择序列化格式化程序 63820.15.1 IFormatter和IRemoting-Formatting接口 63920.15.2 在格式化程序中的类型保真 64020.16 使用BinaryFormatter序列化对象 64120.17 使用SoapFormatter序列化对象 64220.18 使用XmlSerializer序列化对象 64320.19 序列化对象集合 64520.20 自定义Soap/Binary序列化过程 64720.20.1 深入了解对象序列化 64720.20.2 使用ISerializable自定义序列化 64820.20.3 使用特性定制序列化 65020.21 小结 651第21章 ADO.NET之一：连接层 65321.1 ADO.NET的宏观定义 65321.2 ADO.NET数据提供程序 65521.2.1 微软提供的ADO.NET数据提供程序 65621.2.2 关于System.Data.Oracle-Client.dll 65721.2.3 选择第三方的数据提供程序 65721.3 其他的ADO.NET命名空间 65821.4 System.Data命名空间的类型 65821.4.1 IDbConnection接口的作用 65921.4.2 IDbTransaction接口的作用 65921.4.3 IDbCommand接口的作用 66021.4.4 IDbDataParameter和IDataParameter接口的作用 66021.4.5 IDbDataAdapter和IDataAdapter接口的作用 66121.4.6 IDataReader和IDataRecord接口的作用 66121.5 使用接口的抽象数据提供程序 66221.6 创建AutoLot数据库 66521.6.1 创建Inventory表 66521.6.2 为Inventory表添加测试记录 66721.6.3 编写GetPetName()存储过程 66821.6.4 创建Customers和Orders表 66921.6.5 可视化创建表关系 67121.7 ADO.NET数据提供程序工厂模型 67221.7.1 完整的数据提供程序工厂的例子 67321.7.2 数据提供程序工厂模型的潜在缺陷 67521.7.3 元素 67621.8 ADO.NET的连接层 67721.8.1 使用连接对象 67821.8.2 使用ConnectionStringBuilder对象 67921.8.3 使用命令对象 68021.9 使用数据读取器 68121.10 构建可重用的数据访问库 68321.10.1 增加连接逻辑 68421.10.2 增加插入逻辑 68521.10.3 增加删除逻辑 68621.10.4 增加更新逻辑 68621.10.5 增加选择逻辑 68721.10.6 使用参数化的命令对象 68821.10.7 执行存储过程 69021.11 创建控制台UI前端 69121.11.1 实

现Main()方法 69121.11.2 实现ShowInstructions()方法 69321.11.3 实现ListInventory()方法
 69321.11.4 实现DeleteCar()方法 69421.11.5 实现InsertNewCar()方法 69421.11.6 实
 现UpdateCarPetName()方法 69521.11.7 实现LookUpPetName() 69621.12 数据库事务 69721.12.1
 ADO.NET事务对象的主要成员 69721.12.2 为AutoLot数据库添加CreditRisks表 69821.12.3
 为InventoryDAL添加事物方法 69921.12.4 测试数据库事务 70021.13 小结 701第22章 ADO.NET
 之二：断开连接层 70222.1 ADO.NET断开连接层 70222.2 DataSet的作用 70322.2.1 DataSet的主
 要属性 70422.2.2 DataSet的主要方法 70422.2.3 构建DataSet 70522.3 使用DataColumn 70522.3.1
 构建DataColumn 70622.3.2 启用自增列 70722.3.3 把DataColumn对象加入DataTable 70722.4
 使用DataRow 70722.4.1 RowState属性 70922.4.2 DataRowVersion属性 71022.5 使用DataTable
 71022.5.1 将DataTable插入到DataSet中 71122.5.2 获取DataSet中的数据 71222.5.3 使
 用DataTableReader对象处理DataTable 71322.5.4 序列化DataTable/DataSet对象为XML 71422.5.5 以
 二进制格式序列化Data-Table/DataSet对象 71522.6 将DataTable对象绑定到用户界面 71622.6.1 从
 泛型List合成DataTable 71722.6.2 从DataTable中删除行 71922.6.3 根据筛选条件选择行 72022.6.4
 在DataTable中更新行 72222.6.5 使用DataView类型 72322.7 使用数据适配器 72422.7.1 一个简
 单的数据适配器示例 72522.7.2 映射数据库名称为友好名称 72622.8 向AutoLotDAL.dll添加断开连
 接功能 72722.8.1 定义初始类类型 72722.8.2 使用SqlCommandBuilder来配置数据适配器
 72822.8.3 实现GetAllInventory() 72922.8.4 实现UpdateInventory() 72922.8.5 设置版本号
 72922.8.6 测试非连接的功能 73022.9 多表DataSet对象和数据关系 73122.9.1 建立数据适配器
 73222.9.2 建立表间关系 73322.9.3 更新Database表 73322.9.4 在关联表中切换 73422.10
 Windows Forms数据库设计器工具 73622.10.1 可视化设计DataGridView 73622.10.2 生成
 的App.config文件 74022.10.3 强类型的DataSet 74022.10.4 强类型的DataTable 74122.10.5 强类型
 的DataRow 74222.10.6 强类型的数据适配器 74322.10.7 完成Windows Forms应用程序 74422.11
 将强类型的数据库代码隔离到类库中 74422.11.1 查看生成的代码 74622.11.2 用生成的代码选择数
 据 74722.11.3 用生成的代码插入数据 74822.11.4 用生成的代码删除数据 74822.11.5 用生成的
 代码调用存储过程 74922.12 LINQ to DataSet 75022.12.1 DataSet Extensions库的作用 75122.12.2
 获取与LINQ兼容的DataTable 75222.12.3 DataRowExtensions.Field-()扩展方法的作用 75322.12.4
 从LINQ查询中生成新的DataTable 75422.13 小结 754第23章 ADO.NET之三：Entity Framework
 75623.1 Entity Framework的作用 75623.1.1 实体的作用 75823.1.2 Entity Framework的基础知识
 76023.2 创建和分析EDM 76423.3 对概念模型进行编程 77423.4 AutoLotDAL 4.0版，加入实体
 77923.4.1 导航属性的作用 78023.4.2 在LINQ to Entity查询中使用导航属性 78123.4.3 调用存储
 过程 78223.5 将数据实体绑定到Windows Forms GUI 78323.6 展望.NET数据访问API的未来
 78623.7 小结 787第24章 LINQ to XML简介 78824.1 两个XML API的故事 78824.1.1 更优秀
 的DOM——LINQ to XML 78924.1.2 更优秀的LINQ to XML——VB字面量语法 79024.2
 System.Xml.Linq命名空间的成员 79124.2.1 LINQ to XML的轴方法 79324.2.2 奇妙的XName
 和XNamespace 79424.3 使用XElement和XDocument 79524.3.1 从数组和容器中生成文档 79724.3.2
 加载和解析XML内容 79824.4 在内存中操作XML文档 79824.4.1 构建LINQ to XML应用程序
 的UI 79924.4.2 引入Inventory.xml文件 79924.4.3 定义LINQ to XML辅助类 80024.4.4 将UI组装
 到辅助类 80124.5 小结 802第25章 WCF 80325.1 各种分布式计算API 80325.1.1 DCOM的作用
 80425.1.2 COM+/企业服务的作用 80425.1.3 MSMQ的作用 80525.1.4 .NET Remoting的作用
 80525.1.5 XML Web服务的作用 80625.2 WCF的作用 80725.2.1 WCF特性概览 80725.2.2
 SOA概览 80825.2.3 WCF概要 80925.3 WCF核心程序集 80925.4 Visual Studio WCF项目模板
 81025.5 WCF应用程序的基本构成 81125.6 WCF的ABC 81225.6.1 WCF契约 81325.6.2 WCF
 绑定 81425.6.3 WCF地址 81625.7 构建WCF服务 81625.7.1 [ServiceContract]特性 81825.7.2
 [OperationContract]特性 81925.7.3 作为操作契约的服务类型 81925.8 承载WCF服务 82025.8.1
 在App.config文件中创建ABC 82025.8.2 针对ServiceHost类型进行编程 82125.8.3 指定库地址
 82125.8.4 ServiceHost类型的功能 82325.8.5 元素的细节 82425.8.6 启用元数据交换 82525.9
 构建WCF客户端应用程序 82725.9.1 使用svcutil.exe生成代理代码 82725.9.2 使用Visual Studio生成

代理代码 82825.9.3 配置基于TCP的绑定 83025.10 简化配置设置 83125.10.1 使用默认终结点
 83125.10.2 使用多重绑定公开单独的WCF服务 83225.10.3 修改WCF绑定的设置 83325.10.4 使用默认MEX行为配置 83525.10.5 刷新客户端代理和选择绑定 83625.11 使用WCF服务库项目模板 83725.11.1 构建简单的Math服务 83725.11.2 使用WcfTestClient.exe测试WCF服务 83825.11.3 使用SvcConfigEditor.exe修改配置文件 83925.12 以Windows服务承载WCF服务 84025.12.1 在代码中指定ABC 84125.12.2 启用MEX 84325.12.3 创建Windows服务安装程序 84325.12.4 安装Windows服务 84525.13 从客户端异步调用服务 84525.14 定义WCF数据契约 84725.14.1 使用Web相关的WCF服务项目模板 84825.14.2 实现服务契约 84925.14.3 *.svc文件的作用 85125.14.4 更新web.config文件 85125.14.5 测试服务 85125.15 小结 852第26章 Windows Workflow Foundation简介 85326.1 定义业务流程 85326.2 构建简单的工作流 85426.3 Workflow运行时 85726.3.1 使用WorkflowInvoker承载工作流 85726.3.2 使用WorkflowApplication承载工作流 86026.3.3 第一个工作流示例回顾 86126.4 检查Workflow中的活动 86126.4.1 控制流活动 86126.4.2 流程图活动 86226.4.3 消息传递活动 86226.4.4 状态机活动 86326.4.5 运行时活动与基元活动 86326.4.6 事务活动 86326.4.7 集合活动和错误处理活动 86426.5 构建流程图工作流 86426.5.1 在流程图中连接活动 86526.5.2 使用InvokeMethod活动 86626.5.3 定义工作流变量 86726.5.4 使用FlowDecision活动 86826.5.5 使用TerminateWorkflow活动 86826.5.6 构建“true”条件 86926.5.7 使用ForEach活动 87026.5.8 完成应用程序 87226.5.9 我们做了什么 87326.6 在专门的DLL中构建Sequence工作流 87426.6.1 定义初始化项目 87426.6.2 引入程序集和命名空间 87626.6.3 定义工作流参数 87726.6.4 定义工作流变量 87726.6.5 使用Assign活动 87826.6.6 使用If和Switch活动 87926.6.7 构建自定义代码活动 88226.7 使用工作流库 88426.8 小结 886第七部分 WPF第27章 WPF和XAML 88827.1 WPF背后的动机 88827.1.1 统一多种不同的API 88927.1.2 通过XAML将关注点分离 88927.1.3 提供优化的呈现模型 89027.1.4 简化复杂的UI编程 89027.2 各种形式的WPF应用程序 89127.2.1 传统的桌面应用程序 89127.2.2 基于导航的WPF应用程序 89227.2.3 XBAP应用程序 89327.2.4 WPF/Silverlight关系 89427.3 WPF程序集 89427.3.1 Application类的作用 89627.3.2 Window类的作用 89727.4 创建不使用XAML的WPF应用程序 90027.4.1 创建强类型的Window类 90227.4.2 创建简单的用户界面 90227.4.3 与应用程序级别的数据交互 90427.4.4 处理Window对象的关闭 90527.4.5 拦截鼠标事件 90627.4.6 拦截键盘事件 90727.5 仅使用XAML构建WPF应用程序 90827.5.1 用XAML定义窗体对象 90927.5.2 用XAML定义应用对象 91027.5.3 通过msbuild.exe处理XAML文件 91127.6 将标记转换为.NET程序集 91327.6.1 将窗口XAML标记映射到C#代码 91327.6.2 BAML的作用 91427.6.3 将应用程序XAML标记映射到C#代码 91527.6.4 XAML到程序集的过程摘要 91627.7 WPF XAML语法 91627.7.1 Kaxaml 91727.7.2 XAML XML命名空间和XAML关键字 91827.7.3 控制类和成员变量的可见性 92027.7.4 XAML元素、XAML特性和类型转换器 92027.7.5 XAML属性元素语法 92127.7.6 XAML附加属性 92227.7.7 XAML标记扩展 92327.8 使用代码隐藏文件构建WPF应用程序 92427.8.1 为MainWindow类添加代码文件 92527.8.2 为MyApp类添加代码文件 92527.8.3 用msbuild.exe处理代码文件 92627.9 使用Visual Studio构建WPF应用程序 92727.9.1 WPF项目模板 92727.9.2 工具箱和XAML设计器/编辑器 92827.9.3 使用Properties窗口设置属性 92927.9.4 使用Properties窗口处理事件 93027.9.5 在XAML编辑器中处理事件 93127.9.6 Document Outline窗口 93227.9.7 查看自动生成的代码文件 93327.10 使用Visual Studio构建自定义XAML编辑器 93327.10.1 设计窗口的GUI 93427.10.2 实现Loaded事件 93527.10.3 实现按钮的Click事件 93627.10.4 实现Closed事件 93727.10.5 测试应用程序 93727.10.6 探索WPF文档 93827.11 小结 939第28章 使用WPF控件编程 94128.1 WPF核心控件概述 94128.1.1 WPF Ink控件 94228.1.2 WPF Document控件 94228.1.3 WPF公共对话框 94228.1.4 文档中的细节 94328.2 Visual Studio WPF设计器 94428.2.1 在Visual Studio中使用WPF控件 94428.2.2 使用Document Outline编辑器 94528.3 使用面板控制内容布局 94628.3.1 在Canvas面板中放置内容 94828.3.2 在WrapPanel面板中放置内容 94928.3.3 在StackPanel面板中放置内容 95128.3.4 在Grid面板中放置内容 95228.3.5 在DockPanel面板中放置内容 95428.3.6 启用Panel类型的滚动功能 95528.3.7 使用Visual Studio设

计数器配置Panel 95628.4 使用嵌套面板构建窗口框架 95928.4.1 构建菜单系统 96028.4.2 构建工具条 96228.4.3 构建状态条 96328.4.4 完成UI设计 96328.4.5 实现MouseEnter/MouseLeave事件处理程序 96428.4.6 实现拼写检查逻辑 96428.5 WPF命令 96528.5.1 内置的命令对象 96528.5.2 将命令连接到Command属性 96628.5.3 将命令连接到任意行为 96728.5.4 使用Open和Save命令 96828.6 深入了解WPF API和控件 97028.7 构建Ink API选项卡 97228.7.1 设计工具条 97328.7.2 RadioButton控件 97528.7.3 处理Ink API选项卡的事件 97728.7.4 InkCanvas控件 97728.7.5 ComboBox控件 98028.7.6 保存、加载和清除InkCanvas数据 98128.8 Documents API 98228.8.1 块元素和内联元素 98228.8.2 文档布局管理器 98228.9 构建Documents选项卡 98328.9.1 使用代码填充FlowDocument 98428.9.2 启用批注和便签 98528.9.3 保存和加载流文档 98628.10 WPF数据绑定模型 98728.10.1 构建Data Binding选项卡 98828.10.2 使用Visual Studio建立数据绑定 98828.10.3 DataContext属性 99028.10.4 使用IValueConverter进行数据转换 99128.10.5 在代码中建立数据绑定 99228.10.6 构建DataGrid选项卡 99228.11 小结 994第29章 WPF图形呈现服务 99529.1 理解WPF的图形呈现服务 99529.2 使用形状呈现图形数据 99629.2.1 在画布中添加矩形、椭圆形和线条 99829.2.2 在画布中移除矩形、圆形和线条 100129.2.3 折线和多边形 100229.2.4 路径 100229.3 WPF画刷和画笔 100529.3.1 使用Visual Studio配置画刷 100629.3.2 在代码中配置画刷 100829.3.3 配置画笔 100929.4 图形变换 100929.4.1 变换概览 101029.4.2 变换Canvas数据 101129.5 使用Visual Studio变换编辑器 101229.5.1 构建初始布局 101229.5.2 在设计时应用变换 101429.5.3 在代码中变换画布 101529.6 使用绘图和几何图形呈现图形数据 101529.6.1 使用几何图形构建DrawingBrush 101729.6.2 用DrawingBrush进行绘画 101729.6.3 在DrawingImage中使用绘图类型 101829.7 Expression Design的作用 101929.7.1 将示例设计文件导出为XAML 101929.7.2 将图像数据导入WPF对象 102129.7.3 与熊共舞 102229.8 使用可视化层呈现图形数据 102229.9 小结 1028第30章 WPF资源、动画和样式 102930.1 理解WPF资源系统 102930.2 使用对象(逻辑)资源 103430.2.1 Resources属性的作用 103430.2.2 定义窗口级别的资源 103530.2.3 {StaticResource}标记扩展 103730.2.4 {DynamicResource}标记扩展 103730.2.5 应用程序级别的资源 103830.2.6 定义合并的资源字典 103930.2.7 定义只含资源的程序集 104130.3 理解WPF动画服务 104230.3.1 动画类型的作用 104330.3.2 To、From和By属性 104330.3.3 Timeline基类的作用 104430.3.4 用C#代码创建动画 104430.3.5 控制动画的速度 104530.3.6 动画的反转和循环 104630.4 用XAML创建动画 104730.4.1 演示图板的作用 104730.4.2 事件触发器的作用 104830.4.3 使用不连续的关键帧创建动画 104830.5 WPF样式的作用 104930.5.1 定义并使用样式 105030.5.2 重写样式设置 105030.5.3 使用TargetType自动应用样式 105130.5.4 继承已有的样式 105230.5.5 未命名样式的作用 105230.5.6 使用触发器定义样式 105330.5.7 使用多个触发器定义样式 105330.5.8 动画样式 105430.5.9 以编程方式设置样式 105430.6 小结 1056第31章 依赖属性、路由事件和模板 105731.1 依赖属性的作用 105731.1.1 已知的依赖属性 105931.1.2 CLR属性包装器的重要说明 106131.2 构建自定义依赖属性 106131.2.1 添加数据验证例程 106531.2.2 响应属性的改变 106531.3 路由事件 106631.3.1 路由冒泡事件的作用 106731.3.2 继续或中止冒泡 106831.3.3 路由隧道事件的作用 106831.4 逻辑树、可视树和默认模板 107031.4.1 以编程方式查看逻辑树 107031.4.2 以编程方式查看可视树 107231.4.3 以编程方式查看控件的默认模板 107331.5 使用触发器框架构建自定义控件模板 107531.5.1 模板资源 107631.5.2 使用触发器添加可视提示 107831.5.3 {TemplateBinding}标记扩展的作用 107931.5.4 ContentPresenter的作用 108031.5.5 融合模板和样式 108131.6 小结 1082第八部分 ASP.NET Web Form第32章 ASP.NET Web Form 108432.1 HTTP的作用 108432.1.1 HTTP请求/响应循环 108432.1.2 HTTP是无状态协议 108532.2 Web应用程序和Web服务器 108532.2.1 IIS虚拟目录的作用 108532.2.2 ASP.NET Development Web Server 108632.3 HTML的作用 108732.3.1 HTML文档结构 108732.3.2 HTML表单的作用 108832.3.3 Visual Studio HTML设计器工具 108832.3.4 构建HTML表单 109032.4 客户端脚本的作用 109132.5 回发到Web服务器 109332.6 ASP.NET API概览 109432.6.1 ASP.NET 2.0及其后续版本的主要特性 109532.6.2 ASP.NET 3.5(和.NET 3.5 SP1)的主要特性 109632.6.3 ASP.NET 4.0和4.5的主要特性 109632.7 构建单个文件的ASP.NET网页

109732.7.1 引用AutoLotDAL.dll 109832.7.2 设计UI 109832.7.3 添加数据访问逻辑 109932.7.4 ASP.NET指令的作用 110132.7.5 脚本块 110232.7.6 ASP.NET控件声明 110332.8 使用代码文件构建ASP.NET Web页面 110432.8.1 引用AutoLotDAL.dll程序集 110632.8.2 更新代码文件 110732.8.3 调试并跟踪ASP.NET页面 110732.9 ASP.NET Web Site和ASP.NET Web Application 110832.10 ASP.NET网站目录结构 111032.10.1 引用程序集 111032.10.2 App_Code文件夹的作用 111132.11 页面类型的继承链 111132.12 与传入的HTTP请求交互 111332.12.1 获得浏览器统计数据 111332.12.2 访问传入的表单数据 111432.12.3 IsPostBack属性 111532.13 与输出HTTP响应交互 111532.13.1 提交HTML内容 111632.13.2 重定向用户 111632.14 ASP.NET网页的生命周期 111732.14.1 AutoEventWireUp特性的作用 111832.14.2 Error事件 111832.15 web.config文件的作用 112032.16 小结 1121第33章 ASP.NET Web控件、母版页和主题 112233.1 Web控件的本质 112233.1.1 服务器端事件处理 112333.1.2 AutoPostBack属性 112333.2 Control和WebControl基类 112433.2.1 枚举所包含的控件 112533.2.2 动态添加和删除控件 112733.2.3 与动态创建的控件交互 112833.2.4 WebControl基类的功能 112833.3 ASP.NET Web控件的类别 112933.3.1 关于System.Web.UI.HtmlControls的简短说明 113133.3.2 Web控件的文档 113233.4 构建ASP.NET汽车网站 113233.4.1 使用ASP.NET母版页工作 113333.4.2 定义默认的内容页面 113833.4.3 设计Inventory内容页面 114033.4.4 设计Build-a-Car内容页面 114333.5 验证控件的作用 114633.5.1 开启客户端JavaScript验证支持 114733.5.2 RequiredFieldValidator 114833.5.3 RegularExpressionValidator 114833.5.4 RangeValidator 114833.5.5 CompareValidator 114933.5.6 创建ValidationSummary 115033.5.7 定义验证分组 115133.6 使用主题 115233.6.1 *.skin文件 115333.6.2 应用网站级别的主题 115533.6.3 在页面级别应用主题 115533.6.4 SkinID属性 115533.6.5 以编程方式分配主题 115633.7 小结 1157第34章 ASP.NET状态管理技术 115834.1 状态问题 115834.2 ASP.NET状态管理技术 116034.3 ASP.NET视图状态的作用 116034.3.1 演示视图状态 116134.3.2 添加自定义视图状态数据 116234.4 Global.asax文件的作用 116334.4.1 全局最后异常事件处理程序 116434.4.2 HttpApplication基类 116534.5 应用程序状态与会话状态的差别 116534.5.1 维护应用程序级的状态数据 116634.5.2 修改应用程序数据 116834.5.3 处理Web应用程序的关闭 116934.6 使用应用程序缓存 116934.6.1 使用数据缓存 117034.6.2 修改*.aspx文件 117234.7 维护会话数据 117434.8 cookie 117734.8.1 创建cookie 117734.8.2 读取传入的cookie数据 117834.9 元素的作用 117934.9.1 在ASP.NET会话状态服务器中保存会话数据 117934.9.2 把会话数据保存在专门的数据库中 118034.10 ASP.NET用户配置API 118134.10.1 ASPNETDB.mdf数据库 118134.10.2 在web.config中定义用户配置 118234.10.3 以编程方式访问用户配置数据 118334.10.4 分组用户配置数据并且持久化自定义对象 118534.11 小结 1186索引 1187

<<精通C# (第6版) >>

编辑推荐

《精通C#(第6版)》被誉为“C#圣经”的经典著作，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，并入选Jolt大奖提名。

探讨了C#语言和.NET平台的各种特性，包括面向对象编程，委托、事件和Lambda表达式的关系，LINQ编程，多线程、并行和异步编程，ADO.NET、WCF、WF、WPF等技术。

新版更透彻阐述了C# 5.0和.NET 4.5的新功能。

《精通C#(第6版)》作者为世界级C#专家、C#超级畅销书作家Andrew Troelsen，英文原版一出即成为亚马逊销量最好的C#图书。

第5版中文版在豆瓣评分高达9.1分，是众多C#程序员力荐的经典好书。

不论是从零开始的菜鸟，还是小有水平的中级程序员，抑或是已经跻身高手梯队的老码农，都需要用这本书来武装自己，正如一位读者所说，“不藏此书，便不像一名真正的C#程序员”。

<<精通C# (第6版) >>

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>