第一图书网, tushu007.com

<<构建高质量的C#代码>>

图书基本信息

书名：<<构建高质量的C#代码>>

13位ISBN编号：9787121197130

10位ISBN编号：7121197138

出版时间：2013-4

出版时间：电子工业出版社

作者：曹化宇

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

前言

本书关注的主题是如何使用C＃写出高质量的代码。
我们知道，高质量的代码是优秀软件的基础，创建高质量的代码是每一个真正的程序员坚持不懈的追求，同时，也是一项艰巨而又充满乐趣的工作。
我们认为，在编写高质量代码的过程中，需要考虑以下三个方面：编写高质量的基础代码。
我们会使用某种编程语言写出语句或基础结构，比如我们使用C＃语言定义变量，或者是创建选择语句、循环语句结构，又或者定义方法、结构、类、接口，等等。
这些内容作为软件代码中的基本元素，与所使用的编程语言的特点是息息相关的。
只有当我们熟练掌握并合理使用了编程语言的特点，才能写出高质量的基础代码。
创建高质量的代码结构。
在这里所说的代码结构，是指在软件开发中，用于实现业务模型的软件架构。
高质量的软件架构可以帮助我们有效完成业务模型所要达到的设计目标，同时，也可以应对不断变化的软件需求。
在软件开发和经验积累的过程中，产生了众多的软件开发技术和方法，它们可能与编程语言无关，但对应于每一种编程语言，实现起来又有一定的独特性。
设计模式就是在不断的开发和积累过程中，发现并总结出的一系列用于优化代码结构的方法。
在软件开发中合理地使用成熟的设计模式，可以帮助我们高效地创建灵活、易扩展、富有弹性的软件架构。
这也是我们在应对“ 需求不断变化 ”这一软件开发中唯一不变真理时的常用且行之有效的开发技术。
逐步改进。
无论是基础代码还是代码结构，要想在初次开发时就能达到很高质量的可能性并不是很大。
因此，这就需要我们对代码进行不断的改进，而改进的步骤应该是循序渐进的，而不是暴风骤雨似的。
在软件开发中，代码和架构的改进过程，也就是重构（Refactoring）的过程，需要我们对每一条语句或语句结构、算法、架构进行逐步的改进和优化。
其中，对于软件架构的重构过程，往往就是应用或去除设计模式的过程。
而我们应该注意的是，重构是在不改变或少量修改代码外在行为的基础上，对其内部结构的优化，以便达到代码和架构设计更合理、更简洁、更灵活的目的。
重要的是，只有在不断的尝试和挫折中才能让我们积累到更多的、有价值的软件开发经验。
此外，我们应该注意代码逐步改进的过程，在这个不断尝试的过程中，会产生多个代码和架构的版本，只有通过对比，我们才能找出最合适的解决方案，这也是经验积累的重要过程。
本书将结合C＃编程语言、重构与设计模式，在掌握C＃语言基础知识的同时，进一步学习如何使用C＃语言实现一些常用的设计模式，而重构则是不断地进行基础代码与软件架构逐步优化的过程，在这一过程中，我们将需要更多地理解高质量代码的进化过程，从而可以帮助我们在使用C＃语言开发软件时，能够更合理、更高效地创建出高质量的代码，并使软件架构变得更易维护、更灵活、更富有弹性。
本书特点享受充满乐趣的学习与开发过程。
软件开发应该是一个充满乐趣的过程，而软件开发的学习过程也应该是这样。
本书使用了通俗易懂的语言，结合比较有趣的示例代码，比如大量使用了模拟创建游戏的代码示例，让大家在一个比较轻松的氛围中学习和应用知识点。
丰富而实用的代码示例。
本书包含了大量的示例代码，并在其中穿插了很多实用的内容，比如随机排序算法、中国农历信息的获取、树状结构，以及大量的设计模式模型，等等。
大家可以在自己的软件中直接使用这些代码，也可以根据需求对其进行修改、扩展或者简化，从而可以更有效、更合理地运用这些算法和代码结构。
大量实用的开发技巧和方法。

在讨论知识点的时候，我们列举了大量的实用开发技巧和方法，并通过实际测试来解答一些技术上的疑问，真正做到能见到树木，亦能见到森林。
本书内容本书主要包括以下几个部分的内容：C＃编程语言，涵盖了使用C＃开发应用软件过程中常用的内容，如数据类型、结构化语句、数组、结构类型、枚举类型、类、委托、泛型、接口、多线程与资源同步等。
并对一些不太常用的内容做了简单的介绍，如预处理、特性、不安全代码、分部类型和可空类型等。
设计模式，介绍了在C＃和．NETFramework中应用的四种设计模式，包括访问者、迭代器、适配器和模板方法模式。
然后，我们详细介绍了11种设计模式，它们是策略、工厂方法、抽象工厂、生成器、单件、观察者、享元、组合、装饰者、状态和桥接模式，并创建了这些设计模式的基本模型，这些都是在C＃开发应用软件过程中可能会经常使用到的模式。
读者应该在学习和使用设计模式的过程中领悟到软件开发的精髓和本质所在，而不只是照葫芦画瓢。
读者应该明白，应用设计模式并不是我们的目标，我们的目标是创建高质量的软件，任何技术的取舍都应该基于这一目标。
在贯穿全书的内容中，我们穿插介绍了各种编写高质量代码、架构，以及逐步改进的方法、技巧，并给出了一些代码演化的过程。
最后，我们讨论了C＃、设计模式、重构，以及软件构建中涉及的一些主题，感兴趣的读者可以选择一些主题进行深入的学习和研究，以扩展自己在软件开发方面的知识面。
读者类型对于本书的读者，你可以是：软件开发的初学者，可以从基础的C＃编程语言开始学习，然后，逐步学习设计模式的应用过程，以提高自己对代码结构整体的把握能力。
从其他语言转到C＃语言的读者，无论你是不是已经会使用VB、C＋＋，或者Java，都可以很快地适应C＃开发的便利性，并且可以通过设计模式在C＃中的应用，更深入地理解C＃语言的特点和软件构建方法。
C＃使用者，如果你正在使用C＃语言进行软件的开发工作，可以从本书学习到如何在C＃中应用一些常见的设计模式，以便提高我们对软件代码的重构和优化能力。
同时，从C＃编程语言的角度，本书也可以作为参考手册，供你随时查询相关的知识点。
本书的使用如果你对C＃语言和．NETFramework不太熟悉，可以从第1章开始逐章阅读，一步步深入学习相关知识。
如果你已经对C＃比较熟悉，可以直接阅读在C＃中使用设计模式的相关内容。
本书也可以作为C＃语言以及设计模式的参考手册，读者在开发的过程中，可以随时在本书中查询C＃语言与应用设计模式，以及了解如何改进和优化软件代码和架构。
此外，代码与架构重构的过程介绍，穿插于不同的主题之中，方便读者直观地了解应用相关技术需要注意的问题，以及代码和架构优化，即逐步改进的过程。
学习方法你可以先通读一遍。
如果你的时间比较紧，至少应该先看一下目录，了解一下本书有哪些内容，这样，可以做到对全书的结构和内容有一个初步的了解。
然后，你可以从第1章开始，或者选择自己感兴趣的内容阅读，但你应该充分理解每一个知识点和示例。
学习软件开发最好的方法就是实践，读者应该多读代码、多写代码。
建议有条件的读者可以自己输入本书中的所有示例，并尽可能地去尝试修改代码，测试不同的方式带来的运行结果上的差异。
相信这么多的代码敲下来，读者对代码的理解会更深入。
另外，这样做还会有一个额外的收获，那就是你的打字速度可能会有所提高。
在阅读过程中，对于不太明白的地方，不用着急，当你系统地学习后，一些难题就会迎刃而解。
请注意，如果你对哪个知识点暂时不能理解，可以先放松一下，如玩玩休闲小游戏、睡一觉，或者出去散散步，再静下心来时，答案就很可能自然地出现在脑海里了。
最后一点，在学习的过程，手边一定要有笔和纸，随便写写、画画对我们的学习都会有所帮助，有时

还会有意外的收获。
交流与反馈就像开发软件一样，撰写一本书也不太可能在一开始就达到完美的境地。
如果读者发现什么问题，或者有些什么建议，又或者只是交流一下软件开发的心得，都可以给我发来E-mail。
祝大家在软件开发的过程中玩得开心！
作者2013年1月

内容概要

《构建高质量的C#代码》介绍了高质量C#代码的成就过程，即从基础代码到软件结构，以及不断优化和重构的过程。
主要内容包括C#的基本语法、结构与应用特点，常用开发资源与技术要点，设计模式在C#中的应用等，以及特别重要的一点，即我们应该养成良好的开发习惯，不仅要注重技术细节，还要从更宽阔的视野角度来重新审视代码的构建工作。

作者简介

曹化宇，独立软件开发人，主要研究方向为软件工程、设计模式与交互设计；最大的兴趣就是coding、技术写作、文学创作，以及研究各种软件开发技术。
在知识的海洋里，每个人的力量如水滴一般微不足道，但我们可以努力去做好自己的事情，从而共同创建一片更加广阔而美好的知识海洋。

书籍目录

第1章开启C#之旅 1.1软件开发概述 1.1.1开发应用软件 1.1.2软件开发的一般步骤 1.2代码构建的重要性 1.3认识C#编程语言 1.3.1C#简史 1.3.2C#关键字及相关标识符 1.3.3C#中的语句和表达式 1.4构建C#开发环境 1.4.1安装开发环境 1.4.2C#程序的基本结构 1.4.3如何创建窗体程序 1.5高质量代码准则：可阅读 1.5.1养成好的代码编写习惯 1.5.2代码不是私人财产，而是艺术品 1.5.3代码应该具有可维护性 1.5.4注释你的代码 1.6高质量代码准则：用实践证明一切 1.7高质量代码准则：好心情带来高质量 1.8准备你的开发工具箱 第2章处理数据 2.1C#数据处理基础 2.1.1C#数据类型 2.1.2变量与常量 2.1.3高质量代码准则：给变量起个好名字 2.1.4值类型与引用类型 2.1.5字面值 2.2整数 2.2.1整数的一般运算 2.2.2溢出检查 2.2.3++与——运算 2.2.4移位运算 2.2.5高质量代码准则：准确使用整数 2.3浮点数与deCimal类型 2.3.1浮点数与deCimal数据的运算 2.3.2高质量代码准则：合理使用浮点数和deCimal类型 2.4布尔类型 2.4.1布尔运算 2.4.2布尔类型数据的用途 2.5字符 2.5.1Char类型 2.5.2asCii码表（0～127） 2.6字符串 2.6.1转义字符 2.6.2逐字字符串（@） 2.6.3字符串的常用操作 2.6.4单个字符时使用Char类型 2.6.5使用stringbuilder类型 2.7日期与时间的处理 2.7.1使用datetime结构类型 2.7.2获取中国农历信息 2.8数据类型的转换 2.8.1显式转换 2.8.2隐式转换 2.8.3tryparse（）方法转换 2.8.4使用Convert类 2.8.5转换是有代价的 2.9objeCt类型 2.9.1一切皆对象 2.9.2装箱和拆箱 2.9.3高质量代码准则：objeCt不应作为万能类型 2.10C#运算符 第3章控制程序流程 3.1if语句 3.1.1使用if语句 3.1.2注意事项 3.2switCh语句 3.2.1使用switCh语句 3.2.2注意事项 3.3for语句 3.3.1使用for语句 3.3.2注意事项 3.4while语句 3.4.1使用while语句 3.4.2注意事项 3.5do语句 3.5.1使用do语句 3.5.2注意事项 3.6foreaCh语句 3.6.1使用foreaCh语句 3.6.2注意事项 3.7break语句 3.8Continue语句 3.9goto语句与标签 第4章处理异常 4.1C#中的异常处理 4.1.1exCeption类型 4.1.2try—CatCh语句结构 4.1.3处理具体的异常类 4.1.4使用finally语句块 4.1.5throw语句 4.2高质量代码准则：处理异常智能化、模式化 4.3在visualC#中调试代码 第5章数组 5.1C#中的数组 5.1.1简单数组 5.1.2洗牌——随机排序算法 5.1.3多维数组 5.2使用array类 5.2.1排序 5.2.2反序排列 5.2.3修改成员数量 5.3使用arraylist类 5.4使用hashtable类 5.5表驱动法——完善中国农历信息的处理 第6章结构 6.1创建结构类型 6.2结构成员 6.2.1结构中的方法 6.2.2结构中的属性 6.2.3其他成员 6.3结构的应用 第7章枚举 7.1C#中的枚举 7.2枚举的应用 第8章类 8.1创建类的作用与目的 8.2C#中的类 8.2.1创建类 8.2.2创建类的实例 8.2.3构造函数与析构函数 8.2.4构造函数的重载 8.2.5构造函数链 8.3类成员的可访问性 8.4变量 8.5属性 8.6方法 8.6.1创建方法 8.6.2方法的参数 8.6.3方法的返回值 8.6.4方法的重载 8.6.5高质量代码准则：创建高质量的方法 8.7索引器 8.8静态类与静态成员 8.8.1静态成员 8.8.2静态类 8.9关于类的更多内容 8.10高质量代码准则：创建高质量的类 8.11封装中国农历信息类 8.11.1第一个版本 8.11.2重构版本 第9章类的继承 9.1继承概述 9.1.1现实世界中的继承 9.1.2C#中的继承 9.1.3能否继承 9.2子类 9.2.1扩展功能 9.2.2隐藏基类成员 9.2.3区分子类与基类成员 9.2.4重写虚成员 9.3抽象类与抽象成员 9.4继承的局限性 第10章接口 10.1接口的特点 10.2创建接口 10.3实现接口 10.4接口的继承 10.5实现多个接口 第11章委托与事件 11.1访问者模式 11.2委托 11.3事件 11.4多路广播委托 第12章命名空间 12.1命名空间的成员 12.2命名空间的组织 12.3引用与别名 12.3.1引用命名空间 12.3.2命名空间别名 12.4使用miCrosoft.visualbasiC命名空间 12.4.1获取操作系统与内存信息 12.4.2播放波形文件 12.4.3网络基本应用 第13章泛型 13.1泛型方法 13.2泛型类 13.3约束 第14章运算符重载 14.1在结构中重载运算符 14.2在类中重载运算符 第15章资源同步与自动清理 15.1多线程 15.2易失域（volatilefield） 15.3loCk语句 15.4using语句 第16章关于C#的其他主题 16.1预处理 16.1.1根据条件编译代码 16.1.2发布警告或错误 16.1.3定义代码区域 16.2特性 16.2.1使用dllimportattribute特性 16.2.2自定义特性 16.3不安全代码 16.3.1指针与寻址运算 16.3.2sizeof运算 16.4分部类型（partialtype） 16.5可空类型（nullabletype） 第17章代码的进化 17.1从代码到架构 17.1.1基础代码 17.1.2代码集成 17.2重构 17.3设计模式 17.4关注代码改进的过程 17.5找寻.netframework中的设计模式 17.5.1迭代器模式 17.5.2适配器模式 17.5.3模板方法模式 第18章策略模式 18.1虚拟战争游戏示例：使用策略模式组合作战单位 18.1.1第一设计方案 18.1.2陆、海、空 18.1.3真正的设计方案 18.1.4组合第一个作战单位 18.1.5这一切是怎样发生的 18.2应用分析 18.2.1问题 18.2.2解决方案 18.2.3应用特点 第19章工厂方法模式 19.1虚拟战争游戏示例：控制作战单位的创建 19.1.1收起自由创建单位的权力 19.1.2控制作战单位类型 19.1.3统一作战单位的创建方法 19.2应用分析 19.2.1问题 19.2.2解决方案 19.2.3使用空对象 19.2.4应用特点 19.3工厂方法的应用 19.3.1代替构造函数 19.3.2隐

藏特殊对象的创建 第20章抽象工厂模式 20.1示例：模拟组装电脑 20.2应用分析 20.2.1问题 20.2.2解决方案 20.2.3应用特点 20.2.4与工厂方法模式的对比 第21章生成器模式 21.1示例：创建汽车配置清单 21.2应用分析 21.2.1问题 21.2.2解决方案 21.2.3应用特点 第22章单件模式 22.1虚拟战争游戏示例：唯一的重要人物 22.1.1重要人物登场 22.1.2应用单件模式 22.2应用分析 22.2.1问题 22.2.2解决方案 22.2.3应用特点 22.2.4多线程中的单件 第23章观察者模式 23.1订阅邮件——现实版观察者模式 23.2虚拟战争游戏示例：空降作战 23.2.1准备登机 23.2.2诺曼底大空降 23.3.netframework4.0中的观察者 23.4应用分析 23.4.1问题 23.4.2解决方案 23.4.3应用特点 第24章享元模式 24.1虚拟战争游戏示例：点缀作战地图 24.1.1第一个方案 24.1.2使用享元模式 24.2应用分析 24.2.1问题 24.2.2解决方案 24.2.3应用特点 第25章组合模式 25.1虚拟战争游戏示例：多武器平台 25.1.1一个平台，多种武器 25.1.2使用组合模式 25.2应用分析 25.2.1问题 25.2.2解决方案 25.2.3应用特点 第26章装饰者模式 26.1三国游戏示例：装备提高战斗力 26.1.1角色与装备 26.1.2使用装饰者模式 26.2应用分析 26.2.1问题 26.2.2解决方案 26.2.3应用特点 第27章状态模式 27.1三国游戏示例：角色被击中后 27.1.1添加角色状态 27.1.2使用状态模式 27.2应用分析 27.2.1问题 27.2.2解决方案 27.2.3应用特点 第28章桥接模式 28.1绘图示例：绘制图形与文本 28.1.1抽象与实现的分离 28.1.2应用桥接模式 28.2应用分析 28.2.1问题 28.2.2解决方案 28.2.3应用特点 第29章软件开发之路 29.1C#与设计模式 29.2如何使用设计模式 29.3继续学习 附录a设计模式名录

章节摘录

版权页： 插图： 17.1.2代码集成 当我们编写了多个相关的“组件”，就需要考虑代码的集成了。
这时，我们可以将多个组件集成进行相关测试，我们可以使用真正的使用者代码，也可以使用模拟的使用者代码来测试这些组件的可靠性、容错性、性能等各方面的指标是否达到了设计要求。
代码的集成测试是我们发现问题的重要方式，我们可以在此验证组件的接口设计和实现是否合理、是否有效，并可以模拟实际运行环境，考验基础代码的正确性、稳定性和性能。
如果在此发现问题，我们将会对基础代码或架构设计进行进一步的改进。
在实际开发中，越早开始代码集成，就能越早发现问题，也就能越早通过重构来修正这些问题。
17.2重构 重构是逐步改进代码和架构的过程，也是不断尝试的过程。
重构是在不修改或少量修改代码的外部行为的情况下，对其内部结构进行调整的过程：重构过程包括代码结构优化、算法的优化，甚至一个变量的使用优化。
在软件开发中，经常会发现一些这样或那样的问题，这时，我们就可能通过重构来解决这些问题。
通过重构，我们可以达到以下目的：优化代码设计，使其更清晰、更易阅读，这也是创建可维护的高质量代码的要求。
优化算法，使其运行更高效。
优化性能，通过合理、科学地组织代码，可以有效提高软件系统的性能。
优化架构，使软件架构更灵活、更高效、更富有弹性。
让我们对代码的理解更深入，可以帮助我们理解如何更好地使用编程技术和方法，这也是编程技术不断提高的有效途径。
在重构的过程中，我们还应该注意以下几点：对于重复代码、过于复杂的算法和结构，以及思路模糊不清的代码，一定要进行彻底的重构，否则当问题累积过多时，将很难进行有效的改进。
重构过程一定要逐步完成、循序渐进，一次只重构一个问题，重构完成一定要进行测试后才能做出结论。
在有条件的情况下，重构可以尝试多种方法，以便找出最优的重构方法。
如果面对一堆不能正确运行的代码，却找不出好的重构方法，那么就完全重写它们，在这种情况下，完全重写可能是比重构更好的选择。
17.3设计模式 在进行软件架构的重构时，往往会在代码结构中使用一些设计模式，那么，什么是软件的设计模式呢？
设计模式是针对软件开发中不断出现的相同问题，通过实践和总结，整理出的一系列与问题相对应的解决方案。

编辑推荐

《构建高质量的C#代码》编辑推荐：431段精选代码助您创建更灵活更富弹性的软件架构。

第一图书网, tushu007.com

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