

<<更锋利的C#代码>>

图书基本信息

书名：<<更锋利的C#代码>>

13位ISBN编号：9787302179429

10位ISBN编号：7302179425

出版时间：2008-10

出版时间：清华大学出版社

作者：包善东

页数：326

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>

<<更锋利的C#代码>>

前言

这是一本以C#语言为基础，着眼于代码质量的编程指导。虽然没有砖头书的厚重，但却汇集了许许多多开发人员大量的实践经验。每个章节的内容似乎都为大家所熟悉，然而视角却完全不同，通过对那些几乎被人们忽视了的细节的精心处理，不断地提高每一行代码的质量。相信无论是C#初学者，还是具有NET经验的开发人员，都能从本书中得到启发，写出质量更好的代码，开发出更加专业的程序。

<<更锋利的C#代码>>

内容概要

《更锋利的C#代码：编写高质量C#程序》由浅入深、由表及里地讲述存在于C#编码开发中的各种质量问题，让读者清楚地了解什么是应该做的，什么是不应该做的。C#提供的每种语言机制的功能背后，体现了怎样的逻辑含义。当遇到具体的问题时，应该如何选择与取舍。阅读完此书的每一个章节，都会让读者站在更高的角度C#体系拥有更深的认识和把握，不断向软件开发的更高层次迈进。一个好的程序，不仅仅是能得出正确的运行结果，而且还应在其内部保持清晰的代码逻辑和语义，否则，跟随在正常结果之后的也许是艰难的代码维护工作，对程序进行一处修改往往会牵一发而动全身，一不小心就会埋下深深的隐患。从另一个角度来说，如果每一行代码的质量都很高，那么这个软件产品也一定是高质量的。这就像ISO9000的质量体系认证一样，与其在产品生产完成之后再进行检查，不如控制每一步生产环节的质量。

<<更锋利的C#代码>>

作者简介

包善东（网名Richard Bao），群硕软件开发有限公司的一名交互设计师和软件工程师。9岁时萌生了对编程的浓厚兴趣，从此走上了软件开发的道路，至今已积累了十多年的编程经验。作者还曾是其学校交响乐团的大提琴兼钢琴演奏员，在英、法、德、港、台及内地多次进行演出。也许是音乐与艺术思想对编程的渗透，使其在编程中往往善于寻找和谐之美，避免一切生搬硬套。这也许才是《更锋利的C#代码》思想的根源吧。

<<更锋利的C#代码>>

书籍目录

第1章 基本的代码风格1.1 换行的讲究1.1.1 寻找最佳的断行位置1.1.2 每行只写一条语句1.1.3 分行定义变量1.2 避免代码过于拥挤1.2.1 使用空行分隔代码块1.2.2 使用空格降低代码密度1.3 如何缩进1.3.1 嵌套或包含关系引起的缩进1.3.2 因换行而产生的缩进1.3.3 使用空格还是Tab键1.4 大括号1.4.1 大括号的位置1.4.2 空的大括号结构1.4.3 仅包含单个语句的结构体1.5 保持项目文件的条理性1.5.1 解决方案的结构呼应1.5.2 代码文件的结构1.5.3 使用#region标记来隐藏细节第2章 养成良好的注释习惯2.1 何时需要注释2.1.1 解释代码的意图2.1.2 对局部变量的说明2.1.3 充当代码标题2.1.4 指出例外情况2.1.5 开发过程的提示2.2 注释的格式2.2.1 单行注释2.2.2 多行注释2.3 正确使用XML文档注释2.3.1 结构与类的XML文档注释2.3.2 属性的XML文档注释2.3.3 方法的XML文档注释2.3.4 构造函数的XML文档注释2.3.5 事件的XML文档注释2.3.6 枚举类型的XML文档注释2.3.7 泛型的XML文档注释2.3.8 其他标记第3章 一般命名规范3.1 选用合适的名称3.1.1 使用字符的限制3.1.2 使用含义明确的英语3.2 大小写规则3.2.1 Pascal规则3.2.2 Camel规则3.2.3 首字母缩写词与简写词3.2.4 应在何时使用何种大小写规则3.3 考虑跨语言编程3.3.1 不要通过大小写区分标识符3.3.2 避免与其他语言的关键字重复3.3.3 避免使用特定语言的术语3.4 命名一致与冲突3.4.1 大小写无关原则3.4.2 对基类型的命名暗示3.4.3 对参数与属性的关系暗示3.4.4 属性名称与自身类型同名3.4.5 与命名空间相关的命名冲突3.5 匈牙利命名法3.5.1 匈牙利命名法的弊端3.5.2 考虑为控件应用匈牙利命名法第4章 处理数据4.1 关于数据类型4.1.1 整数4.1.2 浮点数4.1.3 布尔类型4.1.4 字符与字符串4.2 变量的使用4.2.1 尽可能使用内置关键字4.2.2 初始化一切变量4.2.3 集中使用变量4.3 使用枚举4.3.1 何时使用枚举4.3.2 如何为枚举命名4.3.3 关于枚举项4.3.4 标记枚举4.4 魔数——以字面数值出现在代码中的常量4.5 复杂的表达式4.5.1 运算符的副作用4.5.2 简化表达式第5章 分支结构5.1 使用if结构5.1.1 “==”与“=”的问题5.1.2 如何处理复杂的条件5.2 使用switch结构5.2.1 break语句5.2.2 使用default子句要注意的问题5.3 选择if还是switch5.4 关于判断顺序的设计5.4.1 先判断最有可能成立的条件5.4.2 预防因条件短路而丢失操作5.5 慎用goto语句第6章 循环结构6.1 使用for还是while6.1.1 for和while的语义比较6.1.2 简单的数值迭代——for和while的思维模式的差异6.1.3 预知循环次数——微波炉加热的启示6.1.4 集合迭代——独特的foreach结构6.2 循环变量的使用6.2.1 循环变量的命名6.2.2 循环变量的定义6.2.3 避免循环变量的非常规应用6.3 提高循环效率6.3.1 避免不必要的重复劳动6.3.2 避免不必要的循环第7章 如何使用函数7.1 为什么要使用函数7.1.1 函数与方法7.1.2 代码复用7.1.3 隐藏细节——使用函数进行抽象7.2 函数重载7.2.1 重载的语义——为调用者提供方便7.2.2 保持核心代码唯一7.3 参数的设计7.3.1 参数的命名7.3.2 不要使用保留项7.3.3 何时使用变长参数列表7.3.4 何时使用ref参数和out参数7.3.5 参数的顺序7.3.6 重载函数的参数一致性体现7.4 参数检查的必要性7.4.1 检查零值及空引用7.4.2 检查枚举类型7.4.3 防止数据被篡改7.4.4 在何处检查合法性7.5 函数的出口——离开函数的三种方式7.5.1 返回值的用法7.5.2 离开函数的时机第8章 结构与类8.1 结构与类的比较8.1.1 值类型与引用类型8.1.2 何时应当使用结构8.1.3 何时应当使用类8.2 结构与类的命名8.2.1 措辞8.2.2 避免与命名空间冲突8.2.3 不要使用“C”前缀8.2.4 后缀的使用8.3 如何搭建一个典型的结构8.3.1 找准核心数据8.3.2 数据的表现形式8.3.3 定义等价原则8.3.4 实现基本运算8.4 如何真正面向对象第9章 封装9.1 构造函数9.1.1 构造函数的语义9.1.2 何时使用静态构造方法9.1.3 构造函数的参数及其初始化9.2 Finalize函数9.2.1 垃圾回收器9.2.2 IDisposable接口——显式释放资源的方法9.2.3 释放资源的一般范式9.3 何时应该使用字段9.3.1 存储核心数据9.3.2 维持中间结果9.3.3 常量字段9.4 如何使用字段9.4.1 字段的命名9.4.2 访问控制9.5 何时应该使用属性9.5.1 属性的语义9.5.2 数据访问控制9.5.3 需要的后续操作9.5.4 简单的数据处理9.5.5 预定义的对象实例9.6 如何使用属性9.6.1 属性的命名9.6.2 访问控制9.6.3 提供合理的默认值9.6.4 保持轻量级的操作9.6.5 在属性中抛出异常9.7 何时应该使用方法9.7.1 表示某种操作9.7.2 耗时的任务——方法在形式上的暗示作用9.7.3 有副作用的操作9.7.4 返回不确定的值9.7.5 返回数组或集合对象9.8 如何使用方法9.8.1 方法的命名9.8.2 检查传入的参数9.9 静态类型及成员9.10 嵌套类型及其适用场合9.11 可变类型的安全性9.12 使用程序集与命名空间9.12.1 程序集的划分9.12.2 为什么要使用命名空间9.12.3 命名空间的命名9.12.4 命名空间的管理第10章 继承与多态10.1 如何利用类继承10.1.1 自上而下逐步细化10.1.2 自下而上逐步抽象10.2 继承限制10.2.1 强制继承的抽象类型10.2.2 密封类型10.2.3 扩展方法——直接向已有类型添加功能10.3 关于接口10.3.1 接口的语义10.3.2 接口的命名10.3.3 使用接口还是类继承10.4 何时应当显式实现接口10.4.1 解决

<<更锋利的C#代码>>

接口之间的命名冲突10.4.2 提供强类型操作10.4.3 隐藏仅用于通过接口访问的成员10.5 使用多态10.5.1 何时应进行重写10.5.2 应当重写哪个成员10.5.3 保持参数名称一致10.6运算符重载10.6.1 可重载的运算符10.6.2 符合运算符的本意10.6.3 运算符的关联性10.6.4 类型转换运算符的重载第11章 泛型机制11.1 装箱与取消装箱11.2 何时使用泛型11.3 泛型的类型参数设计11.3.1 类型参数的命名11.3.2 使用类型参数的时机第12章 事件与委托12.1 何为事件驱动模式12.2 如何响应事件12.2.1 事件处理函数12.2.2代码的分配12.2.3 事件侦听器使用12.3如何提供事件12.3.1 何时应当提供事件12.3.2 事件的命名12.3.3 传递与事件相关的数据12.3.4 用于事件的委托及其要遵守的约定12.3.5 触发事件12.4 使用委托12.4.1 何时使用委托12.4.2 何时使用匿名方法12.4.3 基类型与派生类型第13章 集合类型13.1 系统内置集合类型13.1.1 数组13.1.2 列表13.1.3 字典13.1.4 其他类型13.2 选用适当的集合类型要考虑的几个方面13.2.1 容量13.2.2 进出次序13.2.3 定位的问题——索引 / 键访问13.2.4 元素结构13.2.5 排序13.3 性能比较13.4 提供自己的集合类型13.4.1 何时应提供集合类型13.4.2 集合类型的命名13.4.3 提供与内置集合类型一致的行为13.4.4 索引器及其应遵守的规则13.4.5 迭代器第14章 LINQ查询14.1 提高LINQ查询的效率14.1.1 查询语法和方法语法的区别14.1.2 LINQ查询的创建、执行与性能14.1.3 减少返回的数据量14.2 LINQ中的错误处理——采用防御式编程14.3 LINQ查询的相关机制14.3.1 匿名类型14.3.2 隐式类型的局部变量14.3.3 Lambda表达式与匿名函数第15章 异常15.1 处理异常时应遵守的规范15.2 抛出异常15.2.1 异常的语义15.2.2 不应使用异常的位置15.2.3 控制异常15.2.4 异常的重新抛出——重新包装时要注意的15.3 选用合适的异常类型15.3.1 常见的异常类型15.3.2 不应使用的异常类型15.4 异常提示信息15.5 设计自定义异常及应遵循的约定第16章 全球化与本地化16.1 分离与特定区域相关的信息16.2 处理特定区域性的数据16.2.1 区分区域性与界面区域性16.2.2 在内部使用Unicode16.2.3 文本的比较与排序16.2.4 不要假定区域性的行为16.3 何时使用固定区域性16.4 用户界面应注意的细节16.4.1 使用资源16.4.2 术语16.4.3 界面布局16.4.4 歧义16.4.5 组合文本附录A : C#、VB.NET、J#关键字表附录B : 常用的异常类型

<<更锋利的C#代码>>

章节摘录

第1章基本的代码风格 假设我们写的是文章而不是程序，那么你一定觉得诸如文章应该分为若干个自然段、每段开头空两格之类的规则是理所当然的。如果段落的开头不空两格，或者干脆把整个文章写成单独的一段，仔细想来似乎也不会影响文章实质内容的表达。

既然如此，我们为什么还要在形式上下功夫呢？

设想一下，如果你手中的这本书既无章节也无目录，正文中的不同内容都使用同样的字体字号印刷，几百页纸从头至尾洋洋洒洒如念经般地“一气呵成”，你还有耐心看下去吗？

这是一个人人都能理解的道理，可是当文章变成程序的时候，就不是每个人都能想得通的了。不仅仅是初学者，甚至一些熟练的开发人员，也会写出凌乱不堪的代码。

许多人一定有过这样的经历：一年半载之后，自己原来写的程序就完全看不懂了。

如果这段程序只是为了交作业，或者临时一用，那还可以不去追究，但如果这是一个商业软件，现在需要根据客户的要求进行修改的话，工作量可就大了——你不得不先花时间把你原来的思路看懂。

肯定会有人反驳：代码是给机器运行的，又不是给人看的，写那么好看有什么用？

他的话只对了前半句：代码确实是给机器运行的，可是机器总共才需要看它几分钟？

你花一个月编写的程序，机器顶多两三分钟就编译好了——在这两三分钟之前，这代码不都是你在看吗？

开发软件编写代码不是一朝一夕的事情，更多的情况下，一个软件的开发要经历很长的时间，并且常常由多人合作完成。

一个庞大的软件项目，可能会动用上千名程序员工作数年！

如果把代码写得连自己都看不明白，怎么与别人交流？

同一个开发团队内，一定要保持良好且一致的代码风格，才能最大化地提高开发效率。

有的初学者会问：我现在只是一个人写程序，并不需要和其他人合作，这些条条框框还有什么必要吗？

要知道，团队协作只是一个方面。

我经常遇到这类情况，一些初学者拿着他的程序来说：“这个怎么不能编译？”

我帮他吧代码整理了半天，发现有一个地方丢了半个大括号。

如果他写程序的时候能够稍加注意一些的话，相信此类错误完全可以避免。

保持良好的编程习惯，能够避免的错误还远不止这些。

<<更锋利的C#代码>>

编辑推荐

《更锋利的C#代码：编写高质量C#程序》由清华大学出版社出版。

一个好的程序，不仅仅是能得出正确的运行结果。

每个章节的内容似乎都为大家所熟悉，然而视角完全不同。

通过对那些几乎被人们忽视了的细节的精心处理，不断地提高每一行代码的质量。

它们为什么必须是，而并非形式主义。

C#提供的每种语言机制的功能背后，体现了怎样的逻辑含义。

读完此书，你会站在更高的角度与C#体系拥有更深的认识和把握。

<<更锋利的C#代码>>

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>