

<<编程匠艺>>

图书基本信息

书名：<<编程匠艺>>

13位ISBN编号：9787121069802

10位ISBN编号：7121069806

出版时间：2008年

出版时间：电子工业出版社

作者：（美）古德利弗（Goodliffe, P.）著

页数：582

字数：840000

译者：韩江，陈玉译

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

作为从事软件开发的程序员，你肯定遇到过这样的情况：自认为完美的代码，在项目快要结束的时候，却总是会发现还有好多内容需要修改。

更有甚者，由于人员的变动，那些他们遗留下来的“老代码”，作为时间留给程序员与项目组的最大遗产，却可能会成为项目组的灾难。

除了受制于人类自身的缺陷之外，还有由于组织而带来的问题，如客户需求不断变更、必须在有限的时间和预算之内完成项目，来自内部所谓“项目管理”的种种压力，等等。

天哪，这些问题我们绝大部分人都赶上了。

列宁曾在监狱中写下了《怎么办？

》，指导了俄国的十月革命。

<<编程匠艺>>

内容概要

如果你可以编写出合格的代码，但是想更进一步、创作出组织良好而且易于理解的代码，并希望成为一名真正的编程专家或提高现有的职业技能，那么《编程匠艺——编写卓越的代码》都会为你给出答案。

本书的内容遍及编程的各个要素，如代码风格、变量命名、错误处理和安全性等。

此外，本书还对一些更广泛的编程问题进行了探讨，如有效的团队合作、开发过程和文档编写，等等。

本书各章的末尾均提供一些思考问题，这些问题回顾了各章中的一些关键概念，可以促使你像专家一样思考，从而使本书成为那些渴望作为团队的一分子，职业并高效地编程的新手们的一本绝佳的参考书。

作者简介

Pete Goodliffe是一位软件开发专家，他在软件“食物链”上从未驻足不前。

他在各种各样的项目中使用过许多种语言。

他还在教授和指导程序员方面有着丰富的经验，并且常年为ACCU的C Vu杂志（www.accu.org）撰写栏目“编程的职业化”。

Pete痴迷于编写出色的、没有错误的代码

<<编程匠艺>>

书籍目录

第 篇 代码表面第一部分	第1章 善于防守——健壮代码的防御性编程技巧	1.1 向优秀的代码前进	1.2 设想：最坏的选择	1.3 什么是防御性编程	1.4 又大又坏的世界	1.5 防御性编程技巧	1.5.1 使用好的编码风格和合理的设计	1.5.2 不要仓促地编写代码	1.5.3 不要相信任何人	1.5.4 编码的目标是清晰，而不是简洁	1.5.5 不要让任何人做他们不该做的修补工作	1.5.6 编译时打开所有警告开关	1.5.7 使用静态分析工具	1.5.8 使用安全的数据结构	1.5.9 检查所有的返回值	1.5.10 审慎地处理内存（和其他宝贵的资源）	1.5.11 在声明位置初始化所有变量	1.5.12 尽可能推迟一些声明变量	1.5.13 使用标准语言工具	1.5.14 使用好的诊断信息日志工具	1.5.15 审慎地进行强制转换	1.5.16 细则	1.6 约束	1.6.1 约束的内容	1.6.2 移除约束	1.7 总结	1.8 另请参见	1.9 思考	1.9.1 深入思考	1.9.2 结合自己	第2章 精心布局——源代码的版面和样式	2.1 什么是关键	2.2 了解你的读者	2.3 什么是好的样式	2.4 使用括号	2.4.1 K&R括号风格	2.4.2 悬挂式的括号风格	2.4.3 缩进的括号风格	2.4.4 其他的括号风格	2.5 主宰一切的风格	2.6 内部风格（以及在哪里使用它们）	2.7 设立标准	2.8 正义的战争	2.9 总结	2.10 另请参见	2.11 思考	2.11.1 深入思考	2.11.2 结合自己	第3章 名正言顺——为有意义的事物起有意义的名称	3.1 为什么我们应该恰当地命名呢	3.2 我们对什么进行命名	3.3 名字游戏	3.3.1 描述性	3.3.2 技术上正确	3.3.3 符合语言习惯	3.3.4 恰当	3.4 具体细节	3.4.1 命名变量	3.4.2 命名函数	3.4.3 命名类型	3.4.4 命名名字空间	3.4.5 命名宏	3.4.6 命名文件	3.5 玫瑰不叫玫瑰	3.5.1 保持前后一致	3.5.2 利用上下文	3.5.3 使用对你有利的名称	3.6 总结	3.7 另请参见	3.8 思考	3.8.1 深入思考	3.8.2 结合自己	第4章 不言自明——编写“自文档化”代码的技巧	4.1 自文档化的代码	4.2 编写自文档化代码的技术	4.2.1 使用好的样式编写简单的代码	4.2.2 选择有意义的名称	4.2.3 分解为原子函数	4.2.4 选择描述性的类型	4.2.5 命名常量	4.2.6 强调重要的代码	4.2.7 分组相关信息	4.2.8 提供文件头	4.2.9 恰当地处理错误	4.2.10 编写有意义的注释	4.3 实用的自文档化方法	4.3.1 文学性编程	4.3.2 文档化工具	4.4 总结	4.5 另请参见	4.6 思考	4.6.1 深入思考	4.6.2 结合自己	第5章 随篇注释——如何编写代码注释	5.1 什么是代码注释	5.2 注释看上去是什么样的	5.3 多少注释是恰当的	5.4 注释中应该有什么	5.4.1 解释为什么，而不是怎么样	5.4.2 不要描述代码	5.4.3 不要取代代码	5.4.4 确保注释有用	5.4.5 避免分心	5.5 实践	5.6 从审美的角度看注释	5.6.1 一致性	5.6.2 清晰的块注释	5.6.3 缩进的注释	5.6.4 行尾注释	5.6.5 帮助你阅读代码	5.6.6 选择一种维护成本较低的风格	5.6.7 分隔板	5.6.8 标志	5.6.9 文件头注释	5.7 使用注释	5.7.1 帮助你编写例行程序	5.7.2 错误修正通告	5.7.3 注释过时	5.7.4 维护和空洞无物的注释	5.8 总结	5.9 另请参见	5.10 思考	5.10.1 深入思考	5.10.2 结合自己	第6章 人非圣贤——处理不可避免的情况——代码中的错误情形	6.1 从何而来	6.2 错误报告机制	6.2.1 不报告	6.2.2 返回值	6.2.3 错误状态变量	6.2.4 异常	6.2.5 信号	6.3 检测错误	6.4 处理错误	6.4.1 何时处理错误	6.4.2 可能的反应	6.4.3 代码示例	6.5 使地狱浮现	6.6 管理错误	6.7 总结	6.8 另请参见	6.9 思考	6.9.1 深入思考	6.9.2 结合自己	第 篇 代码的神秘生命	第7章 欲善其事，先利其器——使用工具构建软件	7.1 什么是软件工具	7.2 为什么要在意工具	7.3 使工具发挥作用	7.3.1 了解它能做些什么	7.3.2 学习如何驾驭它	7.3.3 了解它适合什么任务	7.3.4 检查它是否可用	7.3.5 找到了解更多信息的途径	7.3.6 查明新版本何时出现	7.4 哪个工具	7.4.1 源代码编辑工具	7.4.2 代码构建工具	7.4.3 调试和调查工具	7.4.4 语言支持工具	7.4.5 其他工具	7.5 总结	7.6 另请参见
--------------	------------------------	--------------	--------------	--------------	-------------	-------------	----------------------	-----------------	---------------	----------------------	-------------------------	-------------------	----------------	-----------------	----------------	--------------------------	---------------------	--------------------	-----------------	---------------------	------------------	-----------	--------	-------------	------------	--------	----------	--------	------------	------------	---------------------	-----------	------------	-------------	----------	---------------	----------------	---------------	---------------	-------------	---------------------	----------	-----------	--------	-----------	---------	-------------	-------------	--------------------------	-------------------	---------------	----------	-----------	-------------	--------------	----------	----------	------------	------------	------------	--------------	-----------	------------	------------	--------------	-------------	-----------------	--------	----------	--------	------------	------------	-------------------------	-------------	-----------------	---------------------	----------------	---------------	----------------	------------	---------------	--------------	-------------	---------------	-----------------	---------------	-------------	-------------	--------	----------	--------	------------	------------	--------------------	-------------	----------------	--------------	--------------	--------------------	--------------	--------------	--------------	------------	--------	---------------	-----------	--------------	-------------	------------	---------------	---------------------	-----------	----------	-------------	----------	-----------------	--------------	------------	------------------	--------	----------	---------	-------------	-------------	-------------------------------	----------	------------	-----------	-----------	--------------	----------	----------	----------	----------	--------------	-------------	------------	-----------	----------	--------	----------	--------	------------	------------	-------------	-------------------------	-------------	--------------	-------------	----------------	---------------	-----------------	---------------	-------------------	-----------------	----------	---------------	--------------	---------------	--------------	------------	--------	----------

<<编程匠艺>>

- 7.7 思考 7.7.1 深入思考 7.7.2 结合自己 第8章 测试时代——测试代码的魔术
- 8.1 反思现实 8.2 谁、是什么、何时以及为什么 8.2.1 我们为什么要测试 8.2.2
- 谁来进行测试 8.2.3 测试的内容有些什么 8.2.4 何时进行测试 8.3 测试并不难...
- ... 8.4 测试的类型 8.5 选择单元测试用例 8.6 为测试而设计 8.7 看！
- 不要用手！
- 8.8 面对故障该怎么办 8.9 你能管理它吗 8.9.1 缺陷跟踪系统 8.9.2 bug审
- 查 8.10 总结 8.11 另请参见 8.12 思考 8.12.1 深入思考 8.12.2 结合自
- 己 第9章 寻找缺陷——调试：当事情进展得不顺利时该怎么办 9.1 生活的真相 9.2 bug
- 的种类 9.2.1 从远处看 9.2.2 从近处看 9.2.3 从更近处看 9.3 消灭害虫
- 9.3.1 地下之路 9.3.2 地上之路 9.4 搜寻bug 9.4.1 编译时错误 9.4.2
- 运行时错误 9.5 如何修正缺陷 9.6 预防 9.7 除蜂剂、驱虫剂、捕蝇纸 9.7.1 调
- 试器 9.7.2 内存访问校验器 9.7.3 系统调用跟踪 9.7.4 内核转储 9.7.5
- 日志 9.8 总结 9.9 另请参见 9.10 思考 9.10.1 深入思考 9.10.2 结合自
- 己 第10章 代码构建——将源代码转换为可执行代码的过程 10.1 语言障碍 10.1.1 解
- 释型语言 10.1.2 编译型语言 10.1.3 字节编译型语言 10.2 小题大做 10.3 构
- 建软件版本 10.4 怎样才算是一个优秀的构建系统 10.4.1 简洁 10.4.2 一致
- 10.4.3 可重复和可靠 10.4.4 原子性 10.4.5 能够应付错误 10.5 技术细节
- 10.5.1 目标的选择 10.5.2 内务处理 10.5.3 依赖关系 10.5.4 自动构建
- 10.5.5 构建配置 10.5.6 递归地使用make 10.6 请发布我吧 10.7 构建大师是全能的
- 吗 10.8 总结 10.9 另请参见 10.10 思考 10.10.1 深入思考 10.10.2 结合
- 自己 第11章 追求速度——优化程序和编写高效的代码 11.1 优化是什么 11.2 是什么使
- 代码不尽如人意 11.3 为什么不进行优化呢 备选方案 11.4 为什么要进行优化
- 11.5 优化的具体细节 11.5.1 证明你需要进行优化 11.5.2 找出运行得最慢的代码
- 11.5.3 测试代码 11.5.4 优化代码 11.5.5 优化之后 11.6 优化的技术
- 11.6.1 设计更改 11.6.2 代码更改 11.7 编写高效的代码 11.8 总结 11.9 另请
- 参见 11.10 思考 11.10.1 深入思考 11.10.2 结合自己 第12章 不安全感综合症
- 编写安全的程序 12.1 危险 12.2 敌人 12.3 借口，都是借口 12.4 感到很脆
- 弱 12.4.1 不安全的设计和体系结构 12.4.2 缓冲溢出 12.4.3 嵌入的查询字符串
- 12.4.4 竞争状况 12.4.5 整数溢出 12.5 防范措施 12.5.1 系统安装技术
- 12.5.2 软件设计技术 12.5.3 代码实现技术 12.5.4 规程技术 12.6 总结
- 12.7 另请参见 12.8 思考 12.8.1 深入思考 12.8.2 结合自己 第 篇 代码的形成
- 过程 第13章 崇尚设计——如何创作出优秀的软件设计 13.1 边设计边编程 13.2 我们要设
- 计什么 13.3 为什么这么忙乱 13.4 良好的软件设计 13.4.1 简洁 13.4.2 优雅
- 13.4.3 模块化 13.4.4 良好的接口 13.4.5 可扩展性 13.4.6 避免重复
- 13.4.7 可移植性 13.4.8 符合语言习惯 13.4.9 良好地文档化 13.5 如何设计代
- 码 13.5.1 设计方法和过程 13.5.2 设计工具 13.6 总结 13.7 另请参见
- 13.8 思考 13.8.1 深入思考 13.8.2 结合自己 第14章 软件体系结构——奠定软件设
- 计的基础 14.1 什么是软件体系结构 14.1.1 软件蓝图 14.1.2 视图 14.1.3
- 在何时和何处进行体系结构设计 14.1.4 用体系结构来做什么 14.1.5 关于组件和连接
- 14.2 什么是良好的体系结构 14.3 体系结构风格 14.3.1 没有体系结构 14.3.2
- 分层的体系结构 14.3.3 管道和过滤器体系结构 14.3.4 客户端/服务器体系结构
- 14.3.5 基于组件的体系结构 14.3.6 框架 14.4 总结 14.5 另请参见 14.6 思考
- 14.6.1 深入思考 14.6.2 结合自己 第15章 改良与革命——代码是如何成长的
- 15.1 软件腐烂 15.2 警告信号 15.3 代码是如何成长的 15.4 相信不可能之事
- 15.5 对此我们可以做些什么 15.5.1 编写新代码 15.5.2 维护现有代码 15.6 总结
- 15.7 另请参见 15.8 思考 15.8.1 深入思考 15.8.2 结合自己 第 篇 “一群
- ”程序员第一部分 第16章 代码猴子——培养正确的编程态度和方法 16.1 各种各样的猴子

<<编程匠艺>>

- 16.1.1 卖力工作的程序员 16.1.2 代码猴子 16.1.3 权威 16.1.4 半权威
 16.1.5 傲慢的天才 16.1.6 牛仔 16.1.7 规划者 16.1.8 老前辈 16.1.9 狂热者
 16.1.10 单线条程序员 16.1.11 拖沓者 16.1.12 勉强的团队领导
 16.1.13 你 16.2 理想的程序员 16.3 那么该怎么办 16.4 最愚蠢的人 16.5 总结
 16.6 另请参见 16.7 行为表格 16.8 思考 16.8.1 深入思考 16.8.2 结合自己
 第17章 团结就是力量——团队合作与个人程序员 17.1 我们的团队——概览 17.2 团队组织
 17.2.1 管理方法 17.2.2 责任划分 17.2.3 组织和代码结构 17.3 团队合作工具
 17.4 团队疾病 17.4.1 巴别塔 17.4.2 独裁制 17.4.3 民主制
 17.4.4 卫星站 17.4.5 大峡谷 17.4.6 流沙 17.4.7 旅鼠 17.5 良好团队合作的个人技巧和特点
 17.5.1 沟通 17.5.2 谦虚 17.5.3 处理冲突 17.5.4 学习和适应能力
 17.5.5 了解你的不足之处 17.6 团队合作原则 17.6.1 集体代码所有制
 17.6.2 尊重别人的代码 17.6.3 编码准则 17.6.4 定义成功 17.6.5 定义责任
 17.6.6 避免倦怠 17.7 团队的生命周期 17.7.1 团队的创建 17.7.2 团队的成长
 17.7.3 团队合作 17.7.4 团队结束 17.8 总结 17.9 另请参见
 17.10 行为表格 17.11 思考 17.11.1 深入思考 17.11.2 结合自己 第18章 安全措施——源代码控制与自我控制
 18.1 我们的责任 18.2 源代码控制 18.2.1 修订控制 18.2.2 访问控制
 18.2.3 处理代码库 18.2.4 在代码树上创建分支 18.2.5 源代码控制简史
 18.3 配置管理 18.4 备份 18.5 发布源代码 18.6 应该将源代码放在哪里
 18.7 总结 18.8 另请参见 18.9 思考 18.9.1 深入思考 18.9.2 结合自己
 第19章 注意细节——编写软件规范 19.1 规范到底是什么 19.2 规范的类型
 19.2.1 需求规范 19.2.2 功能规范 19.2.3 系统体系结构规范
 19.2.4 用户界面规范 19.2.5 设计规范 19.2.6 测试规范 19.3 规范应当包含哪些内容
 19.4 规范编写过程 19.5 我们为什么会不编写规范 19.6 总结
 19.7 另请参见 19.8 思考 19.8.1 深入思考 19.8.2 结合自己
 第20章 代码审查——执行代码审查 20.1 什么是代码审查 20.2 何时进行审查
 20.2.1 是否要进行审查 20.2.2 审查哪些代码 20.3 执行代码审查 20.3.1 代码审查会议
 20.3.2 集成审查 20.4 审查你的态度 20.4.1 作者的态度 20.4.2 审查人员的态度
 20.5 完美的代码 20.6 代码审查之外 20.7 总结 20.8 另请参见 20.9 清单
 20.10 思考 20.10.1 深入思考 20.10.2 结合自己 第21章 时间估计——软件时间范围估计的魔术
 21.1 在黑暗中摸索 21.2 为什么估计这么困难？
 21.3 压力之下 21.4 实用的估计方法 21.5 计划游戏 21.6 坚持！
 21.7 总结 21.8 另请参见 21.9 思考 21.9.1 深入思考 21.9.2 结合自己
 第22章 程序秘方——代码开发的方法和过程 22.1 编程风格 22.1.1 结构化编程
 22.1.2 面向对象的程序设计 22.1.3 函数式编程 22.1.4 逻辑编程 22.2 烹饪方法：做什么与怎样做
 22.2.3 SSADM和PRINCE 22.2.4 V模型 22.2.5 原型设计 22.2.6 迭代和增量开发
 22.2.7 螺旋模型 22.2.8 敏捷的方法 22.2.9 其他开发过程 22.4 已经够了！
 22.5 选择一种过程 22.6 总结 22.7 另请参见 22.8 思考 22.8.1 深入思考
 22.8.2 结合自己 第23章 编程领域大观——不同的编程分支 23.1 应用程序编程
 23.1.1 塑装软件 23.1.2 定制应用程序 23.2 游戏编程 23.3 系统编程 23.4 嵌入式编程
 23.5 分布式编程 23.6 网络应用程序编程 23.7 企业编程 23.8 数字编程
 23.9 那又怎样 23.10 总结 23.11 另请参见 23.12 思考 23.12.1 深入思考
 23.12.2 结合自己 第24章 下一步呢——结果好就一切都好 但下一步该做什么呢？
 答案和讨论 参考书目 索引

章节摘录

1.5.11 在声明位置初始化所有变量这是一个显而易见的问题。

如果你初始化了每个变量，它们的用途就会是明确的。

依靠像“如果我不初始化它，我就不关心初始值”的经验主义是不安全的。

代码将会发展。

未初始化的值以后可能随时都会变成问题。

C和C++使这个问题更加复杂化。

如果你意外地使用了一个没有初始化的变量，那么你的程序在每次运行的时候都将得到不同的结果，这取决于当时内存中的垃圾信息是什么。

在一个地方声明一个变量，随后再对它进行赋值，在这之后再使用它，这样会为错误打开一个窗口。

如果赋值的语句被跳过，你就会花费大量的时间来寻找程序随机出现各种行为的原因。

在声明每个变量的时候就对它进行初始化，就可以把这个窗口关上，因为即使初始化时赋的值是错误的，至少出现的错误行为也是可以预知的。

比较安全的语言(如Java和C++)通过为所有变量定义初始值，回避了这个易犯的错误。

在声明变量的时候对它进行初始化仍然是一种好的做法，这样可以提高代码的明确性。

1.5.12 尽可能推迟一些声明变量尽可能推迟一些声明变量，可以使变量的声明位置与使用它的位置尽量接近，从而防止它干扰代码的其他部分。

这样做也使得使用变量的代码更加清晰。

你不再需要到处寻找变量的类型和初始化，在附近声明使这些都变得非常明显。

不要在多个地方重用同一个临时变量，即使每次使用都是在逻辑上相互分离的区域中进行的。

变量重用会使以后对代码重新完善的工作变得异常复杂。

每次都创建一个新的变量——编译器会解决任何有关效率的问题。

1.5.13 使用标准语言工具在这方面，C和C++都是一场噩梦。

它们的规范有许多不同的版本，使得许多情况成为了其他实现的未定义行为。

现如今有很多种编译器，每个编译器都有一些与其他编译器稍有不同行为。

这些编译器大部分是相互兼容的，但是仍然存在大量的绳索会套住你的脖子。

明确地定义你正在使用的是哪个语言版本。

除非你的项目要求你(最好是有一个好的理由)，否则不要将命运交给编译器，或者对该语言的任何非标准的扩展。

如果该语言的某个领域还没有定义，就不要依赖你所使用的特定编译器的行为(例如，不要依赖你的C编译器将char作为有符号的值对待，因为其他的编译器并不是这样的)。

这样做会产生非常脆弱的代码。

当你更新了编译器之后，会发生什么？

一位新的程序员加入到开发团队中，如果他不理解那些扩展，会发生什么？

依赖于特定编译器的个别行为，将导致以后难以发现的错误。

媒体关注与评论

“有些书你不得不读，有些书你必须去读。

Pete的书就属于后者--它不仅非常有用，而且十分有趣，能够让你成为一名更加优秀的程序员。

” -Jez Higginson, AOCU主席Pete Goodlife在业界的年头快要超过好多人的年龄了，此君曾经涉猎多个领域、不同的编程语言以及多种架构，并且曾经在采用不同流程的公司里从事过开发工作。

在本书中，他把多年压箱底的一些观念想法和技巧告诉了大家，这些都是时间与智慧的结合，相信无论是开发人员、项目经理甚至测试人员，都可以从中发现阿里巴巴开启金库的钥匙。

--译者

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>