

<<C++高级编程>>

图书基本信息

书名：<<C++高级编程>>

13位ISBN编号：9787302298977

10位ISBN编号：7302298971

出版时间：2012-10

出版时间：清华大学出版社

作者：(比)格莱戈尔(Gregoire, M.), (美)索尔特(Solter, N.A.), (美)凯乐普(Kleper, S.J.)著

页数：933

字数：1603000

译者：侯普秀, 郑思遥

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++高级编程>>

内容概要

精通C++语言最新版本：C++11

C++是当今最流行的高级程序设计语言之一，常用于编写性能卓越的企业级面向对象程序，如游戏或大型商业软件。

但一个无法规避的事实是：C++语法纷繁复杂，学习难度较大。

如何才能化繁为简，全面系统地快速掌握C++知识呢？

C++高级编程(第2版)

将为您提供完美答案。

这本权威书籍在大量实例的引导下，解密C++中鲜为人知的特性，揭示最新版本C++11带来的显著变化，并探讨有助于提高代码质量和编程效率的编程方法、可重用设计模式和良好编程风格。

通过阅读本书，您将能得心应手地开发出优秀的C++11程序。

主要内容

提供详尽的代码范例，读者可随手在自己的代码中使用这些代码

全面介绍C++和STL技术，包括该语言不寻常和怪异的方面

展示应用C++语言高级特性的最佳实践，包括操作符重载、内存管理、制作模板和编写多线程代

码

讨论编写跨语言和跨平台代码的技术

讲述代码重用的重要性以及编写易读C++代码的微妙之处

<<C++高级编程>>

作者简介

作者：（比利时）格莱戈尔（Marc Gregoire）（美国）索尔特（Nicholas A.Solter）（美国）凯乐普（Scott J.Kleper）译者：侯普秀 郑思遥 格莱戈尔是一名软件工程师。

他毕业于比利时鲁文的天主教大学，获得计算机科学工程硕士学位。

之后，他在该大学获得人工智能的优等硕士学位。

完成学业后，他开始为大型软件咨询公司OrdinaBelgium工作。

他曾在Siemens和Nokia Siemens Networks为大型电信运营商提供有关在Solaris上运行关键2G和3G软件的咨询服务。

这份工作要求与来自南美、美国、欧洲、中东、非洲和亚洲的国际团队合作。

Marc目前在Nikon Metrology任职，负责开发3D扫描软件。

他的主要技术专长是C / C++，特别是Microsoft VC++和MFC框架。

除了C / C++之外，Marc还喜欢C#，并且会用PHP创建网页。

除了在Windows上开发的主要兴趣之外，他还擅长在Linux平台上开发24x7运行的c++程序；例如EIB家庭自动化监控软件。

2007年4月，因为在Visual c++方面的专业才能他获得了年度Microsoft MVP称号。

Marc还是CodeGum论坛的活跃分子（id为Marc G），并且为CodeGum撰写了一些文章和FAQ条目。

他还编写了一些自由软件和共享软件，并通过他的网站www.nuonsoft.com发布。

他还在www.nuonsoft.com / blog / 维护了一个博客。

索尔特是一名计算机程序员，开发的软件范围很广，包括系统软件、游戏、网络服务和其他很多类型。

他在Sun Microsystem的高可用集群上所做的工作获得了3项专利，还就此在国际并行和分布式处理会议上发表了一篇技术论文。

在Sun的时候，他还喜欢参与OpenSolaris，他还是OpenSolaris Bible（Wiley，2009）的第一作者。

现在重新从事Web开发，他很高兴地再次和Scott一起在Context Optional工作。

Nick在斯坦福大学学习计算机科学，他在这所大学获得了本科和理科硕士学位，他的主要研究领域是计算机系统。

他曾在富勒顿社区大学讲授了一年的C++课程。

Nick和他的妻子和两个孩子生活在美丽的科罗拉多，他在科罗拉多享受着雪上运动的乐趣。

凯乐普在小学就开始了他的编程生涯，那时他在Tandy TRS—80上用BASIC语言编写探险游戏。

作为他所在高中的Mac迷，Scott转向了更高级的语言，并且发布了一些屡获殊荣的共享软件。

Scott加入了斯坦福大学，并且在这所大学获得了本科和计算机科学的理学硕士学位，主要研究领域是人机交互。

在上大学的时候，Scott是一门涉及编程入门、面向对象设计、数据结构、GUI框架和小组项目的课程的助教。

他之后在斯坦福的一门课程采用这本书作为课本。

毕业后，Scott是几家公司创始团队中的首席工程师。

2006年，Scott与他人合伙创建了ContextOptional，Inc.，这是一家提供社会营销技术的市场领先的供应商。

在工作之余，Scott还热衷于在线购物、阅读和弹吉他。

<<C++高级编程>>

书籍目录

目录回到顶部 《c++高级编程(第2版)》

第 部分 专业的c++简介

第1章 c++速成

1.1 c++基础知识

1.1.1 小程序的“hello world”

1.1.2 名称空间

1.1.3 变量

1.1.4 运算符

1.1.5 类型

1.1.6 条件

1.1.7 循环

1.1.8 数组

1.1.9 函数

1.2 深入研究c++

1.2.1 指针以及动态内存

1.2.2 c++中的字符串

1.2.3 引用

1.2.4 异常

1.2.5 const的多种用法

1.3 作为面向对象语言的c++

1.4 标准库

1.5 第一个有用的c++程序

1.5.1 雇员记录系统

1.5.2 employee类

1.5.3 database类

1.5.4 用户界面

1.5.5 评估程序

1.6 本章小结

第2章 设计专业的c++程序

2.1 程序设计概述

2.2 程序设计的重要性

2.3 c++设计的特点

2.4 c++设计的两个原则

2.4.1 抽象

2.4.2 重用

2.5 重用代码

2.5.1 关于术语的说明

2.5.2 决定是否重用代码

2.5.3 重用代码的策略

2.5.4 绑定第三方应用程序

2.5.5 开放源代码库

2.5.6 c++标准库

2.6 设计模式以及技巧

2.7 设计一个国际象棋程序

2.7.1 需求

<<C++高级编程>>

- 2.7.2 设计步骤
- 2.8 本章小结
- 第3章 面向对象设计
 - 3.1 过程化的思考方式
 - 3.2 面向对象思想
 - 3.2.1 类
 - 3.2.2 组件
 - 3.2.3 属性
 - 3.2.4 行为
 - 3.2.5 综合考虑
 - 3.3 生活在对象世界里
 - 3.3.1 过度使用对象
 - 3.3.2 过于通用的对象
 - 3.4 对象之间的关系
 - 3.4.1 “有一个”关系
 - 3.4.2 “是一个”关系(继承)
 - 3.4.3 “有一个”与“是一个”的区别
 - 3.4.4 not-a关系
 - 3.4.5 层次结构
 - 3.4.6 多重继承
 - 3.4.7 混入类
 - 3.5 抽象
 - 3.5.1 接口与实现
 - 3.5.2 决定公开的接口
 - 3.5.3 设计成功的抽象
 - 3.6 本章小结
- 第4章 设计可重用代码
 - 4.1 重用哲学
 - 4.2 如何设计可重用的代码
 - 4.2.1 使用抽象
 - 4.2.2 构建理想的重用代码
 - 4.2.3 设计有用的接口
 - 4.2.4 协调通用性以及使用性
 - 4.3 本章小结
- 第5章 编码风格
 - 5.1 良好外观的重要性
 - 5.1.1 事先考虑
 - 5.1.2 良好风格的元素
 - 5.2 为代码编写文档
 - 5.2.1 使用注释的原因
 - 5.2.2 注释的风格
 - 5.2.3 本书的注释
 - 5.3 分解
 - 5.3.1 通过重构分解
 - 5.3.2 通过设计分解
 - 5.3.3 本书中的分解
 - 5.4 命名

<<C++高级编程>>

- 5.4.1 选择一个恰当的名称
- 5.4.2 命名约定
- 5.5 使用具有风格的语言特性
 - 5.5.1 使用常量
 - 5.5.2 使用引用代替指针
 - 5.5.3 使用自定义异常
- 5.6 格式
 - 5.6.1 关于大括号对齐的争论
 - 5.6.2 关于空格以及圆括号的争论
 - 5.6.3 空格以及制表符
- 5.7 风格的挑战
- 5.8 本章小结
- 第 部分 专业的c++编码方法
- 第6章 熟悉类和对象
 - 6.1 电子表格示例介绍
 - 6.2 编写类
 - 6.2.1 类定义
 - 6.2.2 定义方法
 - 6.2.3 使用对象
 - 6.3 对象的生命周期
 - 6.3.1 创建对象
 - 6.3.2 销毁对象
 - 6.3.3 对象赋值
 - 6.3.4 复制以及赋值的区别
 - 6.4 本章小结
- 第7章 掌握类与对象
 - 7.1 对象的动态内存分配
 - 7.1.1 spreadsheet类
 - 7.1.2 使用析构函数释放内存
 - 7.1.3 处理复制以及赋值
 - 7.2 定义数据成员的类型
 - 7.2.1 静态数据成员
 - 7.2.2 常量数据成员
 - 7.2.3 引用数据成员
 - 7.2.4 常量引用数据成员
 - 7.3 与方法有关的更多内容
 - 7.3.1 静态方法
 - 7.3.2 const方法
 - 7.3.3 方法重载
 - 7.3.4 默认参数
 - 7.3.5 内联方法
 - 7.4 嵌套类
 - 7.5 类内的枚举类型
 - 7.6 友元
 - 7.7 运算符重载
 - 7.7.1 示例：为spreadsheetcell实现加法

<<C++高级编程>>

- 7.7.2 重载算术运算符
- 7.7.3 重载比较运算符
- 7.7.4 创建具有运算符重载的类型
- 7.8 创建稳定的接口
- 7.9 本章小结
- 第8章 揭秘继承技术
 - 8.1 使用继承构建类
 - 8.1.1 扩展类
 - 8.1.2 重写方法
 - 8.2 使用继承重用代码
 - 8.2.1 weatherprediction类
 - 8.2.2 在子类中添加功能
 - 8.2.3 在子类中替换功能
 - 8.3 利用父类
 - 8.3.1 父类构造函数
 - 8.3.2 父类的析构函数
 - 8.3.3 使用父类方法
 - 8.3.4 向上转型以及向下转型
 - 8.4 继承与多态性
 - 8.4.1 回到电子表格
 - 8.4.2 设计多态性的电子表格单元格
 - 8.4.3 电子表格单元格的基类
 - 8.4.4 独立的子类
 - 8.4.5 利用多态性
 - 8.4.6 考虑将来
 - 8.5 多重继承
 - 8.5.1 从多个类继承
 - 8.5.2 名称冲突以及歧义基类
 - 8.6 有趣而晦涩的继承问题
 - 8.6.1 修改重写方法的特征
 - 8.6.2 继承构造函数(仅限c++11)
 - 8.6.3 重写方法时的特殊情况
 - 8.6.4 子类中的复制构造函数以及赋值运算符
 - 8.6.5 virtual的真相
 - 8.6.6 运行时类型工具
 - 8.6.7 非public继承
 - 8.6.8 虚基类
 - 8.7 本章小结
- 第9章 理解灵活而奇特的c++
 - 9.1 引用
 - 9.1.1 引用变量
 - 9.1.2 引用数据成员
 - 9.1.3 引用参数
 - 9.1.4 引用作为返回值
 - 9.1.5 使用引用还是指针
 - 9.1.6 右值引用(仅限c++11)
 - 9.2 关键字的疑问

<<C++高级编程>>

- 9.2.1 const关键字
- 9.2.2 static关键字
- 9.2.3 非局部变量的初始化顺序
- 9.3 类型以及类型转换
 - 9.3.1 typedef
 - 9.3.2 函数指针typedef
 - 9.3.3 类型别名(仅限c++11)
 - 9.3.4 类型转换
- 9.4 作用域解析
- 9.5 c++11
 - 9.5.1 统一初始化
 - 9.5.2 可选函数语法
 - 9.5.3 空指针文本
 - 9.5.4 尖括号
 - 9.5.5 初始化列表
 - 9.5.6 显式转换运算符
 - 9.5.7 特性
 - 9.5.8 用户定义的字面量
- 9.6 头文件
- 9.7 c的实用工具
 - 9.7.1 变长参数列表
 - 9.7.2 预处理器宏
- 9.8 本章小结
- 第10章 错误处理
 - 10.1 错误与异常
 - 10.1.1 异常的含义
 - 10.1.2 c++中异常的优点
 - 10.1.3 c++中异常的缺点
 - 10.1.4 我们的建议
 - 10.2 异常机制
 - 10.2.1 抛出并捕获异常
 - 10.2.2 异常类型
 - 10.2.3 抛出并捕获多个异常
 - 10.2.4 未捕获的异常
 - 10.2.5 抛出列表
 - 10.3 异常与多态性
 - 10.3.1 标准异常体系
 - 10.3.2 在类层次结构中捕获异常
 - 10.3.3 编写自己的异常类
 - 10.3.4 嵌套异常(仅限c++11)
 - 10.4 堆栈的释放与清理
 - 10.4.1 使用智能指针
 - 10.4.2 捕获、清理并重新抛出
 - 10.5 常见的错误处理问题
 - 10.5.1 内存分配错误
 - 10.5.2 构造函数中的错误
 - 10.5.3 构造函数的function-try-blocks

<<C++高级编程>>

- 10.5.4 析构函数中的错误
- 10.6 综合应用
- 10.7 本章小结
- 第11章 深入探讨标准库
 - 11.1 编码原则
 - 11.1.1 使用模板
 - 11.1.2 使用运算符重载
 - 11.2 c++标准库概述
 - 11.2.1 字符串
 - 11.2.2 i/o流
 - 11.2.3 本地化
 - 11.2.4 智能指针
 - 11.2.5 异常
 - 11.2.6 数学工具
 - 11.2.7 时间工具(仅限c++11)
 - 11.2.8 随机数(仅限c++11)
 - 11.2.9 编译时有理数运算(仅限c++11)
 - 11.2.10 元组(仅限c++11)
 - 11.2.11 正则表达式(仅限c++11)
 - 11.2.12 标准模板库
 - 11.2.13 stl算法
 - 11.2.14 stl中还缺什么
 - 11.3 本章小结
- 第12章 理解容器与迭代器
 - 12.1 容器概述
 - 12.1.1 元素的需求
 - 12.1.2 异常和错误检查
 - 12.1.3 迭代器
 - 12.1.4 c++11的变化
 - 12.2 顺序容器
 - 12.2.1 vector
 - 12.2.2 vector[bool]特化
 - 12.2.3 deque
 - 12.2.4 list
 - 12.2.5 array(仅限c++11)
 - 12.2.6 forward_list(仅限c++11)
 - 12.3 容器适配器
 - 12.3.1 queue
 - 12.3.2 priority_queue
 - 12.3.3 stack
 - 12.4 关联容器
 - 12.4.1 pair工具类
 - 12.4.2 map
 - 12.4.3 multimap
 - 12.4.4 set
 - 12.4.5 multiset
 - 12.5 无序关联容器/哈希表(仅限c++11)

<<C++高级编程>>

- 12.5.1 哈希函数
- 12.5.2 unordered_map
- 12.5.3 unordered_multimap
- 12.5.4 unordered_set/unordered_multiset
- 12.6 其他容器
 - 12.6.1 标准c风格数组
 - 12.6.2 string
 - 12.6.3 流
 - 12.6.4 bitset
- 12.7 本章小结
- 第13章 掌握stl算法
 - 13.1 算法概述
 - 13.1.1 find和find_if算法
 - 13.1.2 accumulate算法
 - 13.1.3 在算法中使用c++11的移动语义
 - 13.2 lambda表达式(仅限c++11)
 - 13.2.1 语法
 - 13.2.2 捕捉块
 - 13.2.3 将lambda表达式用作返回值
 - 13.2.4 将lambda表达式用作参数
 - 13.2.5 示例
 - 13.3 函数对象
 - 13.3.1 算术函数对象
 - 13.3.2 比较函数对象
 - 13.3.3 逻辑函数对象
 - 13.3.4 按位函数对象(仅限c++11)
 - 13.3.5 函数对象适配器
 - 13.3.6 编写自己的函数对象
 - 13.4 算法详解
 - 13.4.1 工具算法
 - 13.4.2 非修改算法
 - 13.4.3 修改算法
 - 13.4.4 排序算法
 - 13.4.5 集合算法
 - 13.5 算法示例：审核选民登记
 - 13.5.1 选民登记审核问题描述
 - 13.5.2 auditvoterrolls函数
 - 13.5.3 getduplicates函数
 - 13.5.4 测试auditvoterrolls函数
 - 13.6 本章小结
- 第14章 使用字符串与正则表达式
 - 14.1 动态字符串
 - 14.1.1 c风格字符串
 - 14.1.2 字符串字面量
 - 14.1.3 c++ string类
 - 14.1.4 原始字符串字面量(仅限c++11)
 - 14.2 本地化

<<C++高级编程>>

- 14.2.1 本地化字符串字面量
- 14.2.2 宽字符
- 14.2.3 非西方字符集
- 14.2.4 locale和facet
- 14.3 正则表达式(仅限c++11)
- 14.3.1 ecmascript语法
- 14.3.2 regex库
- 14.3.3 regex_match()
- 14.3.4 regex_search()
- 14.3.5 regex_iterator
- 14.3.6 regex_token_iterator
- 14.3.7 regex_replace()
- 14.4 本章小结
- 第15章 c++ i/o揭秘
- 15.1 使用流
- 15.1.1 流的含义
- 15.1.2 流的来源和目标
- 15.1.3 流式输出
- 15.1.4 流式输入
- 15.1.5 对象的输入输出
- 15.2 字符串流
- 15.3 文件流
- 15.3.1 通过seek()和tell()在文件中转移
- 15.3.2 将流连接在一起
- 15.4 双向i/o
- 15.5 本章小结
- 第16章 其他库工具
- 16.1 std::function
- 16.2 有理数
- 16.3 chrono库
- 16.3.1 持续时间
- 16.3.2 时钟
- 16.3.3 时点
- 16.4 生成随机数
- 16.4.1 随机数引擎
- 16.4.2 随机数引擎适配器
- 16.4.3 预定义的引擎和引擎适配器
- 16.4.4 生成随机数
- 16.4.5 随机数分布
- 16.5 元组
- 16.6 本章小结
- 第17章 自定义和扩展stl
- 17.1 分配器
- 17.2 迭代器适配器
- 17.2.1 反向迭代器
- 17.2.2 流迭代器
- 17.2.3 插入迭代器

<<C++高级编程>>

- 17.2.4 移动迭代器(仅限c++11)
- 17.3 扩展stl
 - 17.3.1 扩展stl的原因
 - 17.3.2 编写一个stl算法
 - 17.3.3 编写一个stl容器
- 17.4 本章小结
- 第 部分 掌握c++的高级特性
- 第18章 c++运算符重载
 - 18.1 运算符重载概述
 - 18.1.1 重载运算符的原因
 - 18.1.2 运算符重载的限制
 - 18.1.3 运算符重载的决策
 - 18.1.4 不要重载的运算符
 - 18.1.5 可重载运算符小结
 - 18.1.6 右值引用(仅限c++11)
 - 18.2 重载算术运算符
 - 18.2.1 重载一元负号和一元正号
 - 18.2.2 重载递增和递减运算符
 - 18.3 重载按位运算符和二元逻辑运算符
 - 18.4 重载插入运算符和提取运算符
 - 18.5 重载下标运算符
 - 18.5.1 通过operator[]提供只读访问
 - 18.5.2 非整数数组索引
 - 18.6 重载函数调用运算符
 - 18.7 重载解除引用运算符
 - 18.7.1 实现operator*
 - 18.7.2 实现operator-]
 - 18.7.3 operator-]*的含义
 - 18.8 编写转换运算符
 - 18.8.1 转换运算符的多义性问题
 - 18.8.2 用于布尔表达式的转换
 - 18.9 重载内存分配和释放运算符
 - 18.9.1 new和delete的工作原理
 - 18.9.2 重载operator new和operator delete
 - 18.9.3 重载带有额外参数的operator new和operator delete
 - 18.9.4 显式地删除/默认化operator new和operator delete(仅限c++11)
 - 18.10 本章小结
- 第19章 利用模板编写泛型代码
 - 19.1 模板概述
 - 19.2 类模板
 - 19.2.1 编写类模板
 - 19.2.2 编译器处理模板的原理
 - 19.2.3 将模板代码分布在多个文件中
 - 19.2.4 模板参数
 - 19.2.5 方法模板
 - 19.2.6 模板类特例化
 - 19.2.7 子类化模板类

<<C++高级编程>>

- 19.2.8 继承还是特例化
- 19.2.9 模板别名(仅限c++11)
- 19.2.10 替换函数语法(仅限c++11)
- 19.3 函数模板
 - 19.3.1 函数模板特例化
 - 19.3.2 函数模板重载
 - 19.3.3 类模板的friend函数模板
- 19.4 本章小结
- 第20章 模板的高级特性
 - 20.1 深入了解模板参数
 - 20.1.1 深入了解模板类型参数
 - 20.1.2 模板参数模板介绍
 - 20.1.3 深入了解非类型模板参数
 - 20.2 模板类部分特例化
 - 20.3 通过重载模拟函数部分特例化
 - 20.4 模板递归
 - 20.4.1 一个n维网格：初次尝试
 - 20.4.2 一个真正的n维网格
 - 20.5 类型推导(仅限c++11)
 - 20.5.1 auto关键字
 - 20.5.2 decltype关键字
 - 20.5.3 结合模板使用auto和decltype
 - 20.6 可变参数模板(仅限c++11)
 - 20.6.1 类型安全的可变长度参数列表
 - 20.6.2 可变数目的混入类
 - 20.7 元编程
 - 20.7.1 编译时阶乘
 - 20.7.2 循环展开
 - 20.7.3 打印元组(仅限c++11)
 - 20.7.4 类型trait(仅限c++11)
 - 20.7.5 结论
 - 20.8 本章小结
- 第21章 高效的内存管理
 - 21.1 使用动态内存
 - 21.1.1 如何描绘内存
 - 21.1.2 分配和释放
 - 21.1.3 数组
 - 21.1.4 使用指针
 - 21.2 数组-指针的对偶性
 - 21.2.1 数组就是指针
 - 21.2.2 并非所有的指针都是数组
 - 21.3 低级内存操作
 - 21.3.1 指针运算
 - 21.3.2 自定义内存管理
 - 21.3.3 垃圾回收
 - 21.3.4 对象池
 - 21.3.5 函数指针

<<C++高级编程>>

- 21.3.6 方法和成员的指针
- 21.4 智能指针
 - 21.4.1 旧的过时的auto_ptr
 - 21.4.2 新的c++11智能指针
 - 21.4.3 编写自己的智能指针类
- 21.5 内存常见的陷阱
 - 21.5.1 分配不足的字符串
 - 21.5.2 内存泄漏
 - 21.5.3 双重删除和无效指针
 - 21.5.4 访问内存越界
- 21.6 本章小结
- 第22章 c++多线程编程
 - 22.1 简介
 - 22.2 原子操作库
 - 22.2.1 原子类型示例
 - 22.2.2 原子操作
 - 22.3 线程
 - 22.3.1 通过函数指针创建线程
 - 22.3.2 通过函数对象创建线程
 - 22.3.3 通过lambda创建线程
 - 22.3.4 通过成员函数创建线程
 - 22.3.5 线程本地存储
 - 22.3.6 取消线程
 - 22.3.7 从线程获得结果
 - 22.3.8 复制和重新抛出异常
 - 22.4 互斥
 - 22.4.1 互斥体类
 - 22.4.2 锁
 - 22.4.3 std::call_once
 - 22.4.4 互斥体的用法示例
 - 22.5 条件变量
 - 22.6 future
 - 22.7 示例：多线程日志记录器类
 - 22.8 线程池
 - 22.9 线程设计和最佳实践
 - 22.10 本章小结
- 第 部分 c++软件工程
- 第23章 充分利用软件工程方法
 - 23.1 过程的必要性
 - 23.2 软件生命周期模型
 - 23.2.1 分段模型和瀑布模型
 - 23.2.2 螺旋模型
 - 23.2.3 rational统一过程
 - 23.3 软件工程方法学
 - 23.3.1 敏捷
 - 23.3.2 scrum
 - 23.3.3 极限编程(xp)

<<C++高级编程>>

- 23.3.4 软件分流
- 23.4 构建自己的过程和方法
 - 23.4.1 对新思想采取开放态度
 - 23.4.2 提出新想法
 - 23.4.3 知道什么行得通什么行不通
 - 23.4.4 不要逃避
- 23.5 源代码控制
- 23.6 本章小结
- 第24章 编写高效的c++程序
 - 24.1 性能和效率概述
 - 24.1.1 提升效率的两种方式
 - 24.1.2 两种程序
 - 24.1.3 c++是不是低效的语言
 - 24.2 语言层次的效率
 - 24.2.1 高效地操纵对象
 - 24.2.2 使用内联方法和函数
 - 24.3 设计层次的效率
 - 24.3.1 尽可能多地缓存
 - 24.3.2 使用对象池
 - 24.4 剖析
 - 24.4.1 使用gprof的剖析范例
 - 24.4.2 使用visual c++ 2010的剖析范例
 - 24.5 本章小结
- 第25章 开发跨平台和跨语言的应用程序
 - 25.1 跨平台开发
 - 25.1.1 硬件架构问题
 - 25.1.2 实现问题
 - 25.1.3 平台相关的特性
 - 25.2 跨语言开发
 - 25.2.1 混合使用c和c++
 - 25.2.2 转移范例
 - 25.2.3 和c代码链接
 - 25.2.4 混合使用c#与c++
 - 25.2.5 通过jni混合java和c++
 - 25.2.6 混合c++使用perl和shell脚本
 - 25.2.7 混合使用c++和汇编代码
 - 25.3 本章小结
- 第26章 成为测试专家
 - 26.1 质量控制
 - 26.1.1 测试是谁的职责
 - 26.1.2 bug的生命周期
 - 26.1.3 bug跟踪工具
 - 26.2 单元测试
 - 26.2.1 单元测试的方法
 - 26.2.2 单元测试过程
 - 26.2.3 单元测试实例
 - 26.3 更高级别的测试

<<C++高级编程>>

- 26.3.1 集成测试
- 26.3.2 系统测试
- 26.3.3 回归测试
- 26.4 成功测试的技巧
- 26.5 本章小结
- 第27章 熟练掌握调试技术
- 27.1 调试的基本定律
- 27.2 bug分类学
- 27.3 避免bug
- 27.4 为bug做好规划
- 27.4.1 错误日志
- 27.4.2 调试跟踪
- 27.4.3 断言
- 27.4.4 静态断言(仅限c++11)
- 27.5 调试技术
- 27.5.1 重现bug
- 27.5.2 调试可重复的bug
- 27.5.3 调试不可重现的bug
- 27.5.4 调试内存问题
- 27.5.5 调试多线程程序
- 27.5.6 调试示例：文章引用
- 27.5.7 从articlecitations示例中总结的教训
- 27.6 本章小结
- 第28章 将设计技术和框架结合使用
- 28.1 c++编码示例
- 28.1.1 编写一个类
- 28.1.2 子类化已有的类
- 28.1.3 抛出和捕获异常
- 28.1.4 从文件中读取
- 28.1.5 写入文件
- 28.1.6 写一个模板类
- 28.2 肯定有更好的方法
- 28.2.1 双分派
- 28.2.2 混入类
- 28.3 面向对象的框架
- 28.3.1 使用框架
- 28.3.2 模型-视图-控制器范例
- 28.4 本章小结
- 第29章 应用设计模式
- 29.1 迭代器模式
- 29.2 单实例模式
- 29.2.1 示例：一种日志机制
- 29.2.2 实现一个单实例
- 29.2.3 使用一个单实例
- 29.2.4 单实例模式和多线程
- 29.3 工厂模式
- 29.3.1 示例：汽车工厂模拟

<<C++高级编程>>

- 29.3.2 实现一个工厂
- 29.3.3 使用一个工厂
- 29.3.4 工厂的其他用途
- 29.4 代理模式
 - 29.4.1 示例：隐藏网络连接的问题
 - 29.4.2 实现一个代理
 - 29.4.3 使用代理
- 29.5 适配器模式
 - 29.5.1 示例：适配一个logger类
 - 29.5.2 实现一个适配器
 - 29.5.3 使用适配器
- 29.6 装饰器模式
 - 29.6.1 示例：在网页中定义样式
 - 29.6.2 装饰器的实现
 - 29.6.3 使用一个装饰器
- 29.7 责任链模式
 - 29.7.1 示例：事件处理
 - 29.7.2 责任链的实现
 - 29.7.3 责任链的使用
- 29.8 观察者模式
 - 29.8.1 示例：事件处理
 - 29.8.2 观察者的实现
 - 29.8.3 使用观察者
- 29.9 本章小结
- 附录a c++面试
- 附录b 带注解的参考文献
- 附录c 标准库头文件

章节摘录

版权页：插图：提供整洁的接口 为了避免在接口中遗漏功能，某些程序员走向另一个极端：他们包含了可以想到的所有功能。

使用这个接口的程序员总是可以找到完成任务的方法。

遗憾的是，这个接口可能非常混乱，他们无法指出如何实现这个接口。

不要在接口中提供不必要的功能，保持接口的简单整洁。

乍看上去，这个指导方针与前面避免遗漏必要功能的策略相违背。

为了避免遗漏功能而包含所有想象得到的接口尽管是一个策略，但是并不是一个健全的策略。

应该包含必要的功能并省略不必要甚至起反作用的接口。

再次考虑汽车示例。

开汽车只需要使用几个组件：方向盘、刹车、油门踏板、换挡、后视镜、里程计以及仪表板上的一些其他仪表。

现在想象一下，如果汽车的仪表板与飞机的驾驶员座舱类似，具有上百个仪表、控制杆、监控器以及按钮，这没法用！

由于接口比较简单，开汽车比开飞机容易多了：您不需要关心海拔高度、与控制塔通话或者控制飞机中众多的组件（例如机翼、发动机以及起落装置）。

此外，从发展的观点来看，比较小的库容易维护。

如果您试图让每个人都愉快，那么就on应该留出更多的空间来容纳错误，如果您的实现非常复杂以至于纠缠不清，哪怕一个错误也能让库变得无效。

遗憾的是，设计简洁接口的思想看起来很好，但是实际上非常困难。

这个规则基本上是主观的：您决定什么是必须的，什么不是。

当然，当您的判断出错时客户一定会通知您。

提供文档以及注释 无论接口多么便于使用，都应该提供使用文档。

如果不告诉程序员如何使用，不能期望他们会正确使用库。

应该将库或代码称为供其他程序员使用的产品。

产品应该带有说明其正确用法的文档。

提供接口文档有两种方法：接口自身内部的注释以及外部的文档。

您应该尽量提供这两种文档。

大多数公开的API只提供外部文档：在许多标准Unix以及Windows头文件中都缺少注释。

在Unix中，文档的形式通常是名为man pages的在线手册。

在Windows中，集成开发环境通常附带文档。

虽然多数API以及库都取消了接口内的注释，但是我们认为这种形式的文档才是最重要的。

绝不应该给出一个只包含代码的“赤裸的”头文件。

即使注释与外部文档完全相同，具有友好注释的头文件也比只有代码的头文件看上去舒服，即使是最优秀的程序员也希望经常看到书面语言。

有些程序员使用工具将注释自动转换为文档，第5章详细讨论这一技术。

无论您提供注释、外部文档还是二者都提供，文档都应该描述库的行为而不是实现。

行为包括输入、输出、错误条件以及处理、预定用法以及性能保障。

例如，描述生成单个随机数的调用的文档应该说明这个调用不需要参数，返回一个预先指定范围的整数，还应该列出当出现问题时可能抛出的所有异常。

文档不应该详细解释实际生成数字的线性同余算法，在接口注释中提供太多的实现细节可能是接口开发中最常见的错误。

适用于维护库的人员而不是客户的注释会破坏接口以及实现的良好分离，许多开发人员都见到过这种情况。

当然内部的实现也应该有文档记录，只是不要把它作为接口的一部分公开。

第5章详细讨论如何在代码中恰当地使用注释。

编辑推荐

《C++高级编程(第2版)》并不是讲解语言的大量细节并给出少量真实世界的场景，而是教您如何在真实世界中使用C++。

《C++高级编程(第2版)》还会披露一些鲜为人知的特性，使用这些特性可以让编程更简单；还讲解了可重用的编码模式，模式是区分编程新手和专业程序员的标志。

编程书籍往往重点描述语言的语法，而不是语言在真实世界中的应用。

典型的C++教材在每一章中介绍了语言中的大部分知识，讲解语法并列举示例。

格莱戈尔、索尔特、凯乐普编著的《C++高级编程(第2版)》不遵循这个模式。

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>