

《C++面向对象高效编程(第2版)》

图书基本信息

书名：《C++面向对象高效编程(第2版)》

13位ISBN编号：9787115329346

出版时间：2013-10-1

作者：[美]Kayshav Dattatri

页数：757

译者：叶尘

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu000.com

《C++面向对象高效编程(第2版)》

内容概要

比肩Thinking in C++、The C++ Programming Language等经典著作；

Design Patterns作者Erich Gamma博士为本书作序；

国内知名技术专家孟岩、方舟联袂推荐；

对于使用C++进行面向对象编程的开发人员来讲，本书是他们的必备读物。本书包含了丰富的面向对象编程知识，可以让他们进一步提升其编程技能。除了讲解C++编程技巧之外，本书还向读者展示了使用C++进行面向对象设计的技术。更为难能可贵的是，开发人员在开发高效编程架构背后的思维过程也在本书中得以淋漓尽致的体现。

Venkat Narayanan

Cisco公司项目经理

加州大学圣克鲁兹分校讲师

Kayshav的这本著作不仅仅会讲解C++的高级功能特性，还会讲解如何使用这些功能特性来设计大型的面向对象软件系统。由于Kayshav是从软件工程师的角度编写了本书，因此对于有志于成为C++开发高手的读者来说，本书的实用性更强。

本书之所以宝贵，一方面是因为本书内容容易理解，另一方面是本书囊括了所有的C++主题知识。更为重要的是，读者还可以学到如何避免C++程序中的“阿喀琉斯之踵（Achilles heel，可以引申为[致命要害]）”——内存泄露。如果读者仅仅掌握了“内存泄露”这一个主题，也可以单凭这“一招鲜”在日后的C++开发生涯中驰骋纵横。

如果读者足够聪明勤奋，则可以全盘吸收掌握本书无所不包的C++对象编程知识。而且，掌握了本书内容的读者，对任何C++开发团队来讲，都是奋力争夺的宝贵人才。

Michael Hennessy

俄勒冈大学计算机科学系资深讲师

即使在学完C++编程的工作机制之后，读者也需要明白C++编程机制之后的原理。本书完美地将这两者结合起来，读者在学完C++和面向对象编程知识之后，不但可以知道实现软件功能的多种方式，而且还可以确定哪种方式是最佳的。这本书之所以能在众多C++图书中脱颖而出，就是因为它以一种良好的写作风格，外加大量优秀且实用的案例代码，清晰地表达了C++编程的本质。

Kenneth Fogle

加拿大魁北克蒙特利尔道森学院计算机系教授

加拿大魁北克蒙特利尔肯高迪亚大学继续教育讲师

本书编排结构清晰，内容引人入胜。Kayshav通过本书向读者展示了C++设计和编程中会遇到的各种陷阱，同时阐明了C++编程语言的力与美。单凭这一点，本书就可以在我的书架中占据一席之地。

Lyle Thompson

HelioSoft公司CEO

《C++面向对象高效编程(第2版)》

作者简介

Kayshav Dattatri当前是Cisco公司的一名高级技术主管，在此之前，曾以项目负责人、技术主管的身份供职于网景公司（Netscape Communications）和Taligent公司，主要从事前沿Web技术、面向对象（OO）应用开发框架的研究。他还是面向对象设计和C++领域的独立顾问/培训师，而且在操作系统、OO架构、OO语言（包括C++、Smalltalk、Eiffel和Modula-2）等领域有25年以上的从业经验。除此之外，他还是加州大学伯克利扩展（Berkeley Extension）项目的知名讲师，而且凭借其渊博的学识和在教学方面的天分备受欢迎。Kayshav早在1987年，就开始使用C++进行编程了。

书籍目录

第一部分 概念、实践和应用

第1章 什么是面向对象编程 1

1.1 背景 1

1.1.1 面向过程编程示例 2

1.1.2 银行账户的表示 3

1.1.3 银行账户的安全 4

1.1.4 用面向对象编程解决问题 5

1.2 理解对象模型 7

1.3 术语 8

1.4 理解消息、方法和实例变量 8

1.4.1 对象中包含的内容 9

1.4.2 实例化（或创建）对象 11

1.5 什么可以作为类 11

1.6 什么不是类 12

1.7 类的目的 13

1.8 深入了解对象 15

1.8.1 对象的状态 15

1.8.2 对象状态的重要性 15

1.8.3 谁控制对象的状态 17

1.8.4 对象的行为 18

1.9 面向对象软件开发的阶段 18

1.9.1 面向对象分析（OOA） 18

1.9.2 面向对象设计（OOD） 20

1.10 面向对象编程（OOP） 21

1.11 对象模型的关键要素 21

1.12 OOP范式和语言 24

1.13 面向对象编程语言的要求 24

1.14 对象模型的优点 25

1.15 小结 26

第2章 什么是数据抽象 27

2.1 接口和实现的分离 30

2.2 对象接口的重要性 31

2.3 实现的含义 32

2.4 保护实现 32

2.5 数据封装的优点 34

2.6 接口、实现和数据封装之间的关系 35

2.7 数据封装注意事项 36

2.8 确定封装的内容 36

2.9 抽象数据类型 37

2.10 抽象数据类型——栈的实现 38

2.11 C++中的数据抽象 40

2.12 类中的访问区域 41

2.13 和类一起使用的术语 47

2.14 类的实现者 48

2.15 实现成员函数 49

2.16 识别成员函数的目标对象 49

2.17 程序示例 52

- 2.18 对象是重点 53
- 2.19 对接口的再认识 53
- 2.20 什么是多线程安全类 55
- 2.21 确保抽象的可靠性——类不变式和断言 57
 - 2.21.1 类不变式 57
 - 2.21.2 前置条件和后置条件 57
 - 2.21.3 使用断言实现不变式和条件 59
 - 2.21.4 高效使用断言 60
- 2.22 面向对象设计的表示法 60
- 2.23 Booch表示法 61
- 2.24 Booch中类的关系 61
 - 2.24.1 关联 62
 - 2.24.2 聚集 (has—a) 62
 - 2.24.3 “使用”关系 65
 - 2.24.4 继承关系 (is—a) 66
 - 2.24.5 类范畴 66
- 2.25 统一建模语言 (UML) 67
- 2.26 UML中类的关系 68
- 2.27 关联 69
 - 2.27.1 作为聚集的关联 71
 - 2.27.2 OR关联 72
- 2.28 组合 72
- 2.29 泛化关系 (is—a) 74
- 2.30 has—a关系的重要性 75
- 2.31 小结 76
- 第3章 C++与数据抽象 77
 - 3.1 类概念的基础 77
 - 3.2 类要素的细节 78
 - 3.2.1 访问区域 78
 - 3.2.2 分析 79
 - 3.3 复制构造函数 81
 - 3.4 赋值操作符 89
 - 3.5 this指针和名称重整的进一步说明 95
 - 3.6 const成员函数的概念 98
 - 3.7 编译器如何实现const成员函数 99
 - 3.8 C++中类和结构的区别 100
 - 3.9 类可以包含什么 100
 - 3.10 设计期间的重点——类的接口 101
 - 3.11 类名、成员函数名、参数类型和文档 102
 - 3.12 参数传递模式——客户的角度 103
 - 3.13 采用语义 106
 - 3.14 为参数选择正确的模式 108
 - 3.15 函数返回值 109
 - 3.16 从函数中返回引用 111
 - 3.17 编写内存安全类 112
 - 3.18 客户对类和函数的责任 113
 - 3.19 小结 114
- 第4章 OOP中的初始化和无用单元收集 115
 - 4.1 什么是初始化 115

- 4.1.1 使用构造函数初始化 117
- 4.1.2 使用内嵌对象必须遵守的规则 124
- 4.2 无用单元收集问题 125
 - 4.2.1 无用单元 125
 - 4.2.2 悬挂引用 125
 - 4.2.3 无用单元收集和悬挂引用的补救 126
 - 4.2.4 无用单元收集和语言设计 127
 - 4.2.5 在C++中何时产生无用单元 129
 - 4.2.6 对象何时获得资源 130
- 4.3 C++中的无用单元收集 130
- 4.4 对象的标识 132
- 4.5 对象复制的语义 136
- 4.6 对象赋值的语义 142
- 4.7 对象相等的语义 145
- 4.8 为什么需要副本控制 149
 - 4.8.1 信号量示例 150
 - 4.8.2 许可证服务器示例 152
 - 4.8.3 字符串类示例 154
- 4.9 分析 160
- 4.10 “写时复制”的概念 161
 - 4.10.1 何时使用引用计数 167
 - 4.10.2 “写时复制”小结 168
- 4.11 类和类型 169
- 4.12 小结 170
- 第5章 继承的概念 171
 - 5.1 继承的基本知识 172
 - 5.2 is—a关系的含义 186
 - 5.3 继承关系的效果 187
 - 5.4 多态置换原则 187
 - 5.5 用继承扩展类层次 195
 - 5.6 继承的一些基本优点 197
 - 5.7 动态绑定、虚函数和多态性 198
 - 5.7.1 动态绑定含义 201
 - 5.7.2 动态绑定的支持——虚函数 202
 - 5.8 继承对数据封装的影响 204
 - 5.9 多态的含义 206
 - 5.10 有效使用虚函数（动态绑定） 207
 - 5.11 虚析构函数的要求 210
 - 5.12 构造函数和虚函数 214
 - 5.13 一般—特殊的概念 215
 - 5.14 抽象（延期）类的概念 215
 - 5.15 抽象类的用途 219
 - 5.16 强大的继承 232
 - 5.17 有效的代码复用 233
 - 5.18 抽象基类的客户 236
 - 5.19 继承优点小结 237
 - 5.20 继承和动态绑定的危险 238
 - 5.20.1 C++如何实现动态绑定（虚函数） 240
 - 5.20.2 虚函数的开销 240

- 5.20.3 动态绑定和类型检查 241
- 5.21 不必要的继承和动态绑定 242
- 5.22 使用虚函数的不同模式 254
- 5.23 小结 256
- 第6章 多重继承概念 257
 - 6.1 多重继承的简单定义 258
 - 6.2 大学示例 258
 - 6.3 多重继承关系的含义 264
 - 6.4 多重继承场景 265
 - 6.4.1 C++中解决名称冲突 266
 - 6.4.2 二义性基类问题 270
 - 6.5 多重继承的基本优点 271
 - 6.6 多重继承的替换方案 271
 - 6.6.1 第一种替换方案 272
 - 6.6.2 第二种替换方案 274
 - 6.7 重复继承 276
 - 6.8 重复继承的解决方案 279
 - 6.8.1 在C++中通过虚基类共享对象 279
 - 6.8.2 虚基类的优点 282
 - 6.8.3 虚基类产生的新问题 282
 - 6.8.4 比较Eiffel和C++中的多重继承 287
 - 6.9 继承的一般问题 290
 - 6.10 使用mixin类加入静态功能 291
 - 6.10.1 mixin类的定义 291
 - 6.10.2 何时使用mixin类 296
 - 6.11 动态变化情况的设计 296
 - 6.11.1 角色扮演类的设计灵活性 302
 - 6.11.2 如何使用角色扮演类 302
 - 6.11.3 管理角色的另一种方法 311
 - 6.11.4 TUniversityMember类对象的多态用法 312
 - 6.11.5 按要求改动现有类 313
 - 6.11.6 mixin类和角色对象的比较——适用范围 314
 - 6.12 C++的私有派生 316
 - 6.12.1 何时使用私有派生 319
 - 6.12.2 重新导出私有基类的成员 321
 - 6.12.3 私有派生的替换方法——包含 323
 - 6.12.4 需要使用私有派生的情况 324
 - 6.13 mixin类和私有派生的实用示例 327
 - 6.14 继承与包含 333
 - 6.15 小结 334
- 第7章 从类中选择性导出（友元函数） 336
 - 7.1 需要什么 337
 - 7.2 C++的情况 337
 - 7.3 友元关系的含义 340
 - 7.4 非成员函数和友元函数的应用 343
 - 7.4.1 实例1：尽量减少类之间过多的交互 343
 - 7.4.2 实例2：克服语法问题 349
 - 7.4.3 实例3：需要和多个类进行通信的函数 361
 - 7.5 非成员函数的优点 362

- 7.6 选择友元函数还是成员函数 365
- 7.7 小结 366
- 第8章 操作符重载的概念 367
- 8.1 语言类型和程序员定义类型的区别 367
- 8.2 什么是重载操作符 370
- 8.3 操作符重载的优点和缺点 371
- 8.3.1 更加简洁的抽象数据类型 372
- 8.3.2 令人费解的操作符重载 372
- 8.3.3 无法理解优先级和结合规则 373
- 8.4 C++中的重载操作符 376
- 8.5 ++和——操作符的另一个应用 380
- 8.6 更复杂的操作符——成员访问操作符：—> 387
- 8.7 非成员函数的操作符 395
- 8.7.1 作为成员函数的操作符 396
- 8.7.2 作为非成员函数实现的操作符 398
- 8.7.3 为什么需要转换 402
- 8.8 转换函数 402
- 8.8.1 转换构造函数和转换函数之间的相互影响 405
- 8.8.2 消除对临时对象的需求 409
- 8.9 从操作符函数返回结果 411
- 8.10 赋值操作符 415
- 8.11 小结 415
- 第9章 泛型类型 417
- 9.1 重复性编码问题 417
- 9.2 智能解决方案——泛型编程 424
- 9.3 泛型类型（类）的基本知识 427
- 9.4 泛型类型和代码重复 433
- 9.5 泛型类实现者与客户之间的契约 434
- 9.5.1 这是否是良好的设计 439
- 9.5.2 泛型类实现中的操作符和成员函数 441
- 9.5.3 替换解决方案——泛型类的特殊化 443
- 9.6 模板特殊化 444
- 9.6.1 模板成员函数的特殊化 444
- 9.6.2 另一种替换方案：分离对象的比较 446
- 9.6.3 不能随意停用模板类的原因 448
- 9.7 模板类特殊化 449
- 9.8 泛型函数的概念 451
- 9.9 C++中模板类和成员函数的实例化 455
- 9.10 泛型类型和类型检查 462
- 9.11 约束泛型和无约束泛型 463
- 9.11.1 C++中对模板参数的约束 467
- 9.11.2 C++中模板参数的特定类型 468
- 9.11.3 模板参数的默认值 469
- 9.12 C++中对模板参数执行约束 470
- 9.13 泛型类和选择性导出 473
- 9.14 继承和泛型类 476
- 9.15 泛型类继承的用途 483
- 9.16 控制对象创建的一般技巧 485
- 9.17 实现计数指针 487

- 9.18 尽量减少模板对象的代码重复 496
 - 9.18.1 程序的内存占用 498
 - 9.18.2 减少模板代码的方案 498
- 9.19 模板类和源代码保护 510
- 9.20 共享（动态）库中的模板类 510
 - 9.20.1 共享库中的模板类——多实例问题 513
 - 9.20.2 消除共享库中的多个实例 515
 - 9.20.3 和现有共享库链接 516
 - 9.20.4 容器类 517
- 9.21 泛型类和继承的比较 518
- 9.22 小结 519
- 第10章 处理异常情况 520
 - 10.1 处理错误状况的原因 520
 - 10.2 错误码的替换方案 522
 - 10.3 C++异常处理模型 523
 - 10.3.1 C++异常机制的工作方式 524
 - 10.3.2 try块的重要性 526
 - 10.3.3 throw表达式的重要性 526
 - 10.3.4 理解动态调用链 528
 - 10.3.5 处理多个异常 530
 - 10.3.6 catch块的责任 531
 - 10.4 Eiffel中的异常模型 532
 - 10.5 Eiffel和C++异常模型的优缺点 536
 - 10.6 有效地使用C++异常 538
 - 10.7 创建异常层次 538
 - 10.7.1 catch处理代码的顺序 541
 - 10.7.2 编写异常安全函数 543
 - 10.8 项目中的异常处理架构 545
 - 10.9 项目中错误管理的成功策略 547
 - 10.9.1 函数不是防火墙 549
 - 10.9.2 设计异常层次 549
 - 10.10 异常环境中的资源管理 552
 - 10.10.1 自动资源管理 553
 - 10.10.2 泛化资源管理解决方案 556
 - 10.11 异常和构造函数 558
 - 10.11.1 从函数中返回安全资源 558
 - 10.11.2 管理对象数组的辅助类 562
 - 10.11.3 自动无用单元收集的开销 567
 - 10.12 构造函数的部分完成 568
 - 10.13 使用异常创建安全数组 568
 - 10.14 小结 574
- 第二部分 构建强大的面向对象软件
- 第11章 掌握数据抽象 575
 - 11.1 隐藏抽象的实现细节 575
 - 11.1.1 使用句柄的优点 579
 - 11.1.2 使用句柄的缺点 579
 - 11.2 将指针作为数据成员使用（惰性求值） 584
 - 11.3 控制对象创建 586
 - 11.3.1 只允许使用new（）操作符创建对象 586

- 11.3.2 防止使用new () 操作符创建对象 589
- 11.4 使用指针和引用代替内嵌对象 589
- 11.5 避免用大型数组作为自动变量 (或数据成员) 590
- 11.6 使用对象数组和对象指针数组 591
- 11.7 用对象代替基本类型指针作为数据成员和成员函数的返回值 593
- 11.8 与C的兼容性 596
- 11.9 合理选择实现：对象大小和代码效率 598
- 11.10 避免临时对象 601
- 11.11 使用复制构造函数初始化对象 602
- 11.12 有效使用代理对象 603
 - 11.12.1 代理对象有助于安全共享对象 604
 - 11.12.2 代理对象易于使用 605
 - 11.12.3 代理对象是远程对象的替身 606
 - 11.12.4 智能代理对象提供额外的功能 607
 - 11.12.5 代理对象解决语法 / 语义的问题 608
 - 11.12.6 泛型下标代理技术 611
- 11.13 使用简单的抽象建立更复杂的抽象 613
- 11.14 抽象必须允许客户以不同的方式使用类 614
- 11.15 小结 616
- 第12章 高效使用继承 617
 - 12.1 用继承实现简洁的菜单和命令 617
 - 12.2 封装创建对象的细节 624
 - 12.3 虚构造函数的概念 626
 - 12.4 为协议控制而组合使用虚函数和非虚函数 629
 - 12.5 双分派概念 638
 - 12.6 设计和实现容器类 645
 - 12.7 设计可处理不同类型的容器 647
 - 12.8 用泛型编程实现同质容器类 659
 - 12.8.1 设计目的 660
 - 12.8.2 基于模板的同质容器的优点 665
 - 12.9 基于模板的容器的缺点 666
 - 12.10 导航容器 669
 - 12.11 主动迭代器 673
 - 12.12 管理容器和迭代器——客户的角度 682
 - 12.12.1 样式1：在容器中创建并返回迭代器供用户使用 683
 - 12.12.2 样式2：按值返回用户可使用迭代器控制的容器 683
 - 12.13 C++标准模板库 (STL) 685
 - 12.13.1 STL容器 686
 - 12.13.2 迭代器 687
 - 12.13.3 STL中的算法 687
 - 12.14 小结 690
 - 12.15 TArray容器的实现代码 691
- 第13章 理解C++对象模型 701
 - 13.1 高效实现 701
 - 13.2 C++表示对象的方式 701
 - 13.2.1 没有虚函数的类 702
 - 13.2.2 成员函数 702
 - 13.2.3 静态数据成员 703
 - 13.2.4 构造函数 704

13.3 包含虚函数的类	705
13.4 在共享库之间共享虚函数表	708
13.5 虚函数和多重继承（非虚基类）	709
13.6 虚基类	715
13.6.1 虚基类的成员访问	715
13.6.2 带虚函数的虚基类	717
13.7 RTTI（运行时类型识别）的实现支持	719
13.8 基于对象和面向对象编程	720
13.9 引用、指针和值	721
13.9.1 引用和指针的赋值	721
13.9.2 复制构造函数	722
13.9.3 构造函数的职责	723
13.10 复制构造函数的责任	726
13.11 优化对象的按值传递和按值返回	727
13.11.1 按值传递	727
13.11.2 按值返回	729
13.12 运行时初始化	732
13.13 小结	732
附录 A	733
参考书目和推荐读物	737
索引	741

《C++面向对象高效编程(第2版)》

精彩短评

1、 0.0

虽然年代久远，但是这是一本介绍面向对象的好书。

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:www.tushu000.com