国外著名高等院校
信息科学与技术优秀教材

WILEY-
INTERSCIENCE

面向对象编程
C++和Java比较教程

Programming with Objects:
A Comparative Presentation of Object Oriented Programming with C++ and Java

〔美〕Avinash C. Kak 著
徐 波 译

中文版

人民邮电出版社
POSTS & TELECOM PRESS

国外著名高等院校信息科学与技术优秀教材

面向对象编程 C++和 Java 比较教程

[美] Avinash C.Kak 著

徐 波 译

人 民 邮 电 出 版 社

图书在版编目（CIP）数据

面向对象编程 C++和 Java 比较教程 /（美）卡克（Kak, A. C.）著；徐波译. —北京：人民邮电出版社，2004. 7

国外著名高等院校信息科学与技术优秀教材

ISBN 7-115-12277-6

Ⅰ. 面… Ⅱ. ①卡… ②徐… Ⅲ. ①JAVA 语言—程序设计—高等学校—教材 ②C 语言—程序设计—高等学校—教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2004）第 045340 号

版 权 声 明

国外著名高等院校信息科学与技术优秀教材

面向对象编程 C++ 和 Java 比较教程

◆ 著 [美] Avinash C.Kak

译 徐 波

责任编辑 陈冀康

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号

邮编 100061 电子函件 315@ptpress.com.cn

网址 http://www. ptpress.com.cn

读者热线 010-67132705

北京汉魂图文设计有限公司制作

北京隆昌伟业印刷有限公司印刷

新华书店总店北京发行所经销

◆ 开本：787×1092 1/16

印张：51. 25

字数：1 590 千字 2004 年 7 月第 1 版

印数：1-3 500 册 2004 年 7 月北京第 1 次印刷

著作权合同登记 图字：01 - 2003 - 4967 号

ISBN 7-115-12277-6/TP • 3975

定价：76.00 元

内容提要

C++和 Java 是目前两种主流的面向对象编程语言。本书从基本结构到如何进行应用层次的编程，对这两种语言进行比较和对照，具有重要的学术价值和现实意义。

全书共 20 章，分别从基础概念和机制、基本语言要素、OO 编程特性、专业编程应用等方面，介绍了 C++和 Java 的精髓和区别。本书还有一个独特的学习方式，就是用 C++重新编写一个特定的 Java 程序，实现相同的功能，或者反过来，用 Java 重写 C++程序。这种方法能够培养学生更加灵活地处理那些比较困难的项目，无论是 C++还是 Java 项目。每章最后的“更多阅读推荐”提供了丰富的可查询资料。

本书适合高等院校计算机专业用做面向对象程序设计课程的教材，对于熟悉 C++和 Java 两种语言中的一种，而又想了解和掌握另一种语言的程序员也非常有帮助。

前 言

本书用C++和Java来描述面向对象编程，它们是面向对象编程的主要语言。本书所作的绝大部分描述都采用了比较的形式，从基本的语言结构直到应用层次的话题如图形编程、网络编程和数据库编程等。本书要求读者对C的一些重要特性相当熟悉，如指针、字符串、数组和结构等。

我有一种强烈的信念，在讲述一门编程语言时，除了它的语法之外，通过它的应用完整地体现它的美感和力度是至关重要的。如果我们在讲授编程语言时把语言和它的应用分离，那么我们就好像只通过语法来教英语一样，虽然这种做法在许多教学计划中并不少见。

本书从一个迎合特定学术需求的尝试脱胎换骨而来，它是对一种新的面向对象编程领域教学计划的全面诠释。我们需要一个不太教条的教学计划，不拘泥面向对象编程的某种固定风格（或者某种固定语言，因为语言常常决定风格）。在讲授编程技巧时，我们可以把精力集中在一种单一的语言上。但是在更广阔的意义上，编程教育要求我们提供更为丰富和广泛的风格和概念。其结果就是读者在本书中看到的：对C++和Java的集成描述。对两种语言进行比较和对照（从基本的语言结构到语言如何用于应用层次的编程）是非常有价值的。这种比较甚至可能激发富有进取心的学生构思一种新的更为强大的未来面向对象编程语言。为了进一步提高这种比较形式的教学价值，本书同时包含了一些普通C的面向对象模拟特性的描述，其中GNOME/GTK+是一个主要的例子。

本书基于这样一种哲学：通过比较来学习不仅效率很高，而且充满乐趣。有时候我们发现，把记忆和理解固定于类似对象、结构和情况之间的异同能够促进学习和记忆。同时学习C++和Java能够发掘学习者在这方面的知识潜力。根据我的教学实践，学生们对完成相同任务的C++和Java编程结构的比较兴趣颇浓。

把C++和Java放在一起讲授和学习具有一些独特的优点。首先，C++和Java都起源于C，因此它们的基本语言结构具有相当多的共性，把它们放在一起讲授可以节省时间。例如，一旦我们掌握了C++中vector的概念，并且对于一些与C++的vector相关联的有用功能也理解透彻了，然后我们再来学习Java的ArrayList几乎是一点就通。本书对Java部分的讨论，在绝大多数情况下都是对先前所讨论的C++功能在Java中是如何实现而展开的。

本书还有一个独特的学习过程，就是用C++重新编写一个特定的Java程序，实现相同的功能，或者反过来，用Java重写C++程序。根据我的经验，这种方法能够培养学生更加灵活地处理那些更为困难的项目，无论是C++还是Java项目。否则，由于一门课程的时间有限，培养这方面的能力要困难一些。

同时学习两种语言也有不利之处。人们很容易搞混某种特性到底属于哪种语言。幸运的是，在现代编程实践中，这方面的困难大大地得到了缓解。因为现在我们可以一边看一个终端窗口的在线文档，一边在另一个终端窗口

进行编程。C++和 Java 都已变得非常庞大，没有人能够单凭记忆记住所有的类以及每个类所定义的所有属性和函数。因此，即使我们并不同时学习两种语言，我们仍然需要在编程时参考相应的文档。

本书所包含的内容比一个典型的学期课程所包含的内容要多得多。根据我的经验，本书最好是作为两个连续的课程。第一个课程专门讲述基本的语言结构，它们包含在本书的前 15 章。第二个课程则集中讨论应用和设计层次的问题。对于第二个课程，我在最后 5 章中用一本关于设计模式的书作为补充材料。

我不敢奢望像本书这样一个大部头能够不出错误。如果读者能够发现错误，并通过 kak@ purdue.edu 告诉我，我将不胜感激。所有的修正意见将在 www.programming-with-objects.com 在线提供，对于所有提出修正意见的作者，我将列出他们的名字以示谢意。对于指出书中任何错误的读者，我们也一并致谢。本书我所采用的示例程序是我从其他地方直接获得的灵感，我在每章最后的“更多阅读推荐”一节中对它们的作者表示了致谢。

我很高兴能够向未来的教师提供课后作业问题的解决方案。

最后，我还想说明，本书对那些从 C++转到 Java 或者从 Java 转到 C++的人们也应该非常有用。

Avinash C. Kak

West Lafeyette, Indiana

印第安纳州　　2003 年 1 月

目 录

第 1 章 大处着眼——OO 编程的一些共性 …… 1

1.1 什么是面向对象编程 …… 1
1.2 OO 有什么优点 …… 1
1.3 如何精通 OO …… 2

第 2 章 初窥门径 …… 3

2.1 简单程序：对一个整型数组求和 …… 3
2.2 简单程序：终端 I/O …… 8
2.3 简单程序：文件 I/O …… 12
2.4 更多阅读推荐 …… 16
2.5 作业 …… 16

第 3 章 类及其他一些关键的概念 …… 18

3.1 在 C++中定义类 …… 20
3.2 在 Java 中定义类 …… 23
3.3 创建对象：C++和 Java 的异同 …… 25
3.4 在 C++中定义子类 …… 26
3.5 在 Java 中定义子类 …… 29
3.6 阻断继承 …… 32
3.7 创建对象的打印表示形式 …… 34
3.8 对象的销毁 …… 35
3.9 Java 的程序包（Package） …… 36
3.10 C++的名字空间（namespace） …… 40
3.10.1 using 声明和 using 指令 …… 42
3.10.2 哪个名字空间拥有从其他名字空间中引入的名字 …… 43
3.10.3 using 声明和 using 指令的作用域 …… 44
3.10.4 嵌套名字空间和名字空间别名 …… 45
3.10.5 无名名字空间 …… 46
3.10.6 Koenig 不带限定名称的函数名查找 …… 46
3.11 类成员的访问控制 …… 47
3.12 抽象类和接口 …… 49
3.13 对象的比较 …… 51
3.14 类的静态成员 …… 54
3.15 模板类 …… 55
3.16 嵌套类型 …… 56
3.16.1 C++的嵌套类 …… 56
3.16.2 Java 的嵌套类 …… 60
3.17 在 C 程序中实现 OO 行为 …… 64

3.18 更多阅读推荐 …… 70
3.19 作业 …… 70

第 4 章 字符串 …… 73

4.1 C 的字符串：简单回顾 …… 73
4.2 C 风格字符串的一些弱点 …… 75
4.3 C++的字符串 …… 76
4.3.1 创建 C++字符串对象 …… 77
4.3.2 访问单个字符 …… 77
4.3.3 字符串比较 …… 78
4.3.4 连接字符串 …… 80
4.3.5 查找子字符串和字符 …… 81
4.3.6 提取子字符串 …… 83
4.3.7 删除和插入子字符串 …… 83
4.3.8 长度和容量 …… 84
4.3.9 其他字符串函数 …… 87
4.4 Java 的字符串 …… 88
4.4.1 创建 String 和 StringBuffer 对象 …… 89
4.4.2 访问单独的字符 …… 91
4.4.3 字符串的比较 …… 92
4.4.4 连接字符串 …… 94
4.4.5 查找和替换 …… 95
4.4.6 删除和插入子字符串 …… 96
4.4.7 提取子字符串 …… 96
4.5 更多阅读推荐 …… 97
4.6 作业 …… 97

第 5 章 使用容器类 …… 100

5.1 C++的容器类 …… 101
5.1.1 Vector …… 103
5.1.2 Deque …… 114
5.1.3 List …… 116
5.1.4 Stack …… 118
5.1.5 Queue …… 119
5.1.6 Priority_Queue …… 120
5.1.7 Map …… 121
5.1.8 Set …… 123
5.1.9 泛型算法 …… 124
5.2 Java 的容器 …… 124
5.2.1 List …… 126
5.2.2 Set …… 129
5.2.3 Map …… 130
5.2.4 Vector …… 133
5.2.5 Java 容器的算法 …… 135
5.3 参考资料和更多阅读推荐 …… 137

5.4 作业 …… 138

第 6 章 基本类型及其输入/输出 …… 144

6.1 标记、标识符和变量名 …… 144
6.2 C++和 Java 的基本类型 …… 145
6.3 布尔类型 …… 145
6.4 字符类型 …… 146
6.5 整数类型 …… 150
6.6 浮点类型 …… 151
6.7 基本类型的类型转换 …… 152
6.7.1 C++的隐式类型转换 …… 152
6.7.2 Java 的隐性类型转换 …… 155
6.7.3 C++的显式类型转换 …… 158
6.7.4 Java 的显式类型转换 …… 159
6.8 C++的 I/O 流 …… 161
6.8.1 C++的流层次体系 …… 161
6.8.2 字符流的输入输出操作 …… 162
6.8.3 字节流的输入输出操作 …… 167
6.8.4 控制格式 …… 171
6.8.5 字符串流 …… 174
6.9 Java 的 I/O 流 …… 175
6.9.1 基本类型的写入 …… 177
6.9.2 字符串的写入 …… 181
6.9.3 基本类型的读取 …… 184
6.9.4 字符串的读取 …… 185
6.10 更多阅读推荐 …… 186
6.11 作业 …… 186

第 7 章 声明、定义和初始化 …… 191

7.1 什么时候声明同时也是定义 …… 191
7.2 C++的变量在定义时是否同时进行了缺省初始化 …… 193
7.2.1 如果不提供无参构造函数会出现什么情况 …… 195
7.2.2 const 和引用成员的特殊情况 …… 197
7.3 在 Java 中变量在定义时会不会进行缺省的初始化 …… 198
7.3.1 缺省初始化是否受类成员的缺省值影响 …… 200
7.3.2 如果构造函数为一个类成员指定了一个值，那么它的缺省值是否忽略 …… 200
7.4 在 C++中声明指针类型 …… 201
7.5 C++的指针数组 …… 203
7.6 声明多个名字 …… 204
7.7 C++标识符的作用域 …… 205
7.8 Java 标识符的作用域 …… 206
7.9 C++的数组及其初始化 …… 207
7.10 Java 的数组及其初始化 …… 210
7.10.1 Java 数组是个对象 …… 212
7.10.2 实现排序、查找等功能的 java.lang.Arrays 类 …… 213

7.11 符号常量 …… 214
7.12 C++的宏 …… 215
7.13 C++的枚举类型 …… 216
7.14 参考资料和更多阅读推荐 …… 219
7.15 作业 …… 219

第 8 章 对象引用和内存分配 …… 222

8.1 C++的对象引用 …… 222
8.2 Java 的对象引用 …… 224
8.3 C++的内存分配 …… 224
8.4 Java 的内存分配 …… 225
8.5 C++的结构 …… 226
8.6 作业 …… 228

第 9 章 函数和方法 …… 232

9.1 函数声明 …… 232
9.2 C++的参数传递 …… 232
9.2.1 按照传值方式传递一个基本类型的参数 …… 233
9.2.2 按照传指针模式传递一个基本类型的参数 …… 233
9.2.3 按照传引用模式传递一个基本类型的参数 …… 234
9.2.4 按照传值模式传递一个类类型的参数 …… 235
9.2.5 按照传指针模式传递一个类类型的参数 …… 236
9.2.6 按照传引用模式传递一个类类型的参数 …… 237
9.3 Java 的参数传递 …… 238
9.3.1 按照传值模式传递一个基本类型的参数 …… 238
9.3.2 按照传递对象引用模式传递一个类类型的参数 …… 238
9.4 返回引用类型的 C++函数 …… 241
9.5 C++的内联函数 …… 243
9.6 C++的静态变量 …… 244
9.7 C++函数的 const 参数和 const 返回类型 …… 245
9.8 Java 方法的 final 形参 …… 247
9.9 数组参数 …… 247
9.10 C++的函数重载解析 …… 248
9.11 Java 的函数重载解析 …… 251
9.12 C++函数的缺省参数 …… 252
9.13 C++的函数指针 …… 253
9.14 更多阅读推荐 …… 255
9.15 作业 …… 255

第 10 章 异常处理 …… 260

10.1 C 用于多层返回的 setjmp/longjmp 机制 …… 260
10.2 C++的异常处理 …… 263
10.3 C++异常处理的一些用法模式 …… 264
10.4 C++和 Java 的异常处理的区别 …… 270
10.5 Java 的异常处理语法 …… 271

10.6 Java 异常处理的一些用法模式 …… 272
10.7 Java 的 checked 和 unchecked 异常 …… 277
10.8 更多阅读推荐 …… 278
10.9 作业 …… 278

第 11 章 类：剩余的故事 …… 281

11.1 构造函数的访问控制 …… 281
11.1.1 限制对象数量 …… 281
11.1.2 限制对 C++无参构造函数的访问 …… 283
11.2 多个构造函数可否互相调用 …… 285
11.3 C++的静态成员 …… 285
11.4 Java 的静态成员 …… 291
11.5 C++的 const 成员函数 …… 295
11.6 C++的自身引用 …… 295
11.7 Java 的自身引用 …… 298
11.8 C++的析构函数 …… 299
11.9 Java 的对象销毁 …… 303
11.10 C++的拷贝构造函数和拷贝赋值操作符 …… 306
11.11 Java 的赋值操作符的语义 …… 310
11.12 Java 的对象克隆 …… 311
11.13 C++的指向类成员的指针 …… 316
11.14 交叉类（Interleaved Class） …… 318
11.15 C++研究：一个具有适度复杂性的交叉类 …… 319
11.16 Java 研究：一个具有适度复杂性的交叉类 …… 329
11.17 更多阅读推荐 …… 335
11.18 作业 …… 335

第 12 章 C++的操作符重载 …… 343

12.1 操作符标记和操作符函数 …… 343
12.2 操作符的全局重载定义 …… 344
12.3 操作符的成员函数重载定义 …… 345
12.4 单目操作符的全局重载定义 …… 347
12.5 单目操作符的成员函数重载定义 …… 348
12.6 操作符重载的案例研究 …… 349
12.7 灵巧指针（smart pointer）：解引用操作符的重载 …… 360
12.8 自增和自减操作符的重载 …… 367
12.9 用户定义的转换 …… 371
12.10 “()”操作符的重载 …… 374
12.11 通过“<”操作符的重载对类类型的对象进行排序 …… 376
12.12 参考资料和更多阅读推荐 …… 379
12.13 作业 …… 379

第 13 章 泛型和模板 …… 381

13.1 C++的参数化类和函数 …… 383
13.1.1 一个 C++链表程序的实现 …… 383

13.1.2 一个参数化的链表程序 …… 386
13.1.3 模板特化 …… 391
13.1.4 模板声明的通用语法 …… 393
13.2 重温迭代器 …… 395
13.2.1 泛型算法的迭代器类型 …… 395
13.2.2 如何声明迭代器 …… 396
13.3 Java 的参数化类 …… 397
13.3.1 在 Java 中创建自己的参数化类型 …… 399
13.3.2 方法的参数化 …… 403
13.3.3 限制参数 …… 405
13.4 参考资料和更多阅读推荐 …… 408
13.5 作业 …… 408

第 14 章 OO 编程的模型图 …… 410

14.1 用例图 …… 410
14.2 类图 …… 412
14.2.1 类之间的关联关系 …… 413
14.2.2 类之间的聚合和合成关系 …… 414
14.2.3 表示属性 …… 415
14.2.4 表示操作 …… 415
14.2.5 类别模板（stereotype） …… 416
14.3 交互图 …… 416
14.3.1 顺序图 …… 417
14.3.2 协作图 …… 421
14.4 包图 …… 421
14.5 状态图 …… 423
14.6 活动图 …… 427
14.7 参考资料和更多阅读推荐 …… 429
14.8 作业 …… 429

第 15 章 类的扩展 …… 431

15.1 C++子类的公共派生 …… 431
15.2 C++派生类的构造函数 …… 434
15.3 C++派生类的拷贝构造函数 …… 436
15.4 C++派生类的拷贝赋值操作符 …… 438
15.5 C++派生类的操作符重载 …… 440
15.6 C++派生类的析构函数 …… 443
15.7 C++的虚拟成员函数 …… 448
15.7.1 虚拟函数声明的限制 …… 452
15.7.2 多层类层次体系中的虚拟函数 …… 452
15.7.3 操作符能否具有多态行为 …… 454
15.7.4 多态类型 …… 454
15.8 C++函数的静态绑定和动态绑定 …… 455
15.9 C++函数覆盖的限制 …… 458
15.10 C++的虚拟析构函数 …… 461

15.11 C++构造函数的顺序依赖性 …… 462
15.12 C++的抽象类 …… 464
15.13 C++的保护和私有派生类 …… 468
15.14 扩展 Java 类 …… 472
15.15 Java 方法覆盖的限制 …… 475
15.16 Java 构造函数的顺序依赖性 …… 477
15.17 Java 的抽象类 …… 478
15.18 Java 的接口 …… 481
15.18.1 在 Java 中实现多个接口 …… 485
15.18.2 在 Java 中扩展接口 …… 485
15.18.3 接口中的常量 …… 488
15.19 C++个案研究：一个具有适当复杂度的小型类层次体系 …… 489
15.20 Java 个案研究：一个具有适当复杂度的小型类层次体系 …… 501
15.21 参考资料和更多阅读推荐 …… 510
15.22 作业 …… 510

第 16 章 C++的多重继承 …… 515

16.1 MI 的一些例子 …… 515
16.2 重复继承可能导致的问题 …… 520
16.3 多重继承的虚基类 …… 522
16.4 虚基类和拷贝构造函数 …… 527
16.5 虚基类和赋值操作符 …… 530
16.6 避免成员函数的名字冲突 …… 536
16.7 处理数据成员的名字冲突 …… 538
16.8 一个重复继承例子的实现 …… 540
16.9 使用混合（mixin）类 …… 548
16.10 使用角色扮演类 …… 555
16.11 C++的运行时类型确定 …… 566
16.12 参考资料和更多阅读推荐 …… 567
16.13 作业 …… 567

第 17 章 图形用户界面 OO 编程 …… 572

17.1 工具箱历史的简要介绍 …… 573
17.2 AWT/Swing 组件 …… 574
17.3 Qt 部件 …… 575
17.4 GNOME/GTK+部件 …… 575
17.5 最简单的 AWT/Swing GUI 程序 …… 576
17.6 最简单的 Qt GUI 程序 …… 579
17.7 最简单的 GNOME/GTK+程序 …… 582
17.8 GUI 程序的布局管理器 …… 585
17.9 AWT/Swing 的布局管理器 …… 585
17.9.1 Border Layout …… 586
17.9.2 Flow Layout …… 588
17.9.3 Box Layout …… 590
17.9.4 Grid Layout …… 593

17.9.5 Card Layout ····· 596
17.9.6 Grid-Bag Layout ····· 599
17.10 Qt 的布局管理器 ····· 602
17.10.1 Box Layout ····· 603
17.10.2 Grid Layout ····· 605
17.11 GNOME/GTK+的布局管理器 ····· 608
17.11.1 Box Layout ····· 608
17.11.2 Table Layout ····· 609
17.12 GUI 程序的事件处理 ····· 612
17.13 AWT/Swing 的事件处理 ····· 614
17.14 Qt 的事件处理 ····· 624
17.14.1 一个需要使用元对象编译的 Qt 例子 ····· 627
17.14.2 信号函数和 slot 函数的全面总结 ····· 634
17.15 GNOME/GTK+的事件处理 ····· 634
17.15.1 GNOME/GTK+中事件与其他部件的通信 ····· 636
17.15.2 GNOME/GTK+回调函数的全面总结 ····· 641
17.16 AWT/Swing 中带菜单的窗口 ····· 643
17.17 Qt 中带菜单的窗口 ····· 647
17.18 GNOME/GTK+中带菜单的窗口 ····· 653
17.19 在 AWT/Swing 中绘制形状、文本和图像 ····· 661
17.20 在 Qt 中绘制形状、文本和图像 ····· 673
17.21 在 GNOME/GTK+中绘制形状、文本和图像 ····· 678
17.22 Java Applet ····· 687
17.22.1 Applet 的生命周期 ····· 688
17.22.2 Applet 标签 ····· 688
17.22.3 一个 Applet 例子 ····· 690
17.22.4 双重用途的 Applet 编程 ····· 696
17.22.5 AppletContext 接口 ····· 699
17.22.6 与 Applet 相关的安全问题 ····· 703
17.23 参考资料和更多阅读推荐 ····· 704
17.24 作业 ····· 704

第 18 章 面向对象多线程编程 ····· 707

18.1 在 Java 中创建和运行简单的线程 ····· 708
18.2 Java 的 Runnable 接口 ····· 711
18.3 线程的状态 ····· 712
18.4 Java 的线程冲突 ····· 713
18.5 Java 的线程同步 ····· 719
18.6 Java 用于处理死锁的等待-通知机制 ····· 722
18.7 Java 线程之间的数据 IO ····· 726
18.8 Java Applet 的线程 ····· 728
18.9 AWT/Swing 的 Event Dispatch 线程 ····· 730
18.10 C/C++的多线程编程 ····· 737
18.10.1 用 POSIX 线程演示线程冲突 ····· 742
18.10.2 处理 POSIX 线程冲突的 Mutex ····· 744

18.10.3 POSIX 线程：处理死锁的条件变量和等待-信号机制 …… 746
18.11 C++的面向对象多线程编程 …… 750
18.12 参考资料和更多阅读推荐 …… 756
18.13 作业 …… 756

第 19 章 网络编程 …… 758

19.1 在 Java 中与现有服务器建立 Socket 连接 …… 758
19.2 Java 的服务器 Socket …… 761
19.3 在 C++中与现有服务器建立 Socket 连接 …… 766
19.4 C++（Qt）的服务器 Socket …… 771
19.5 更多阅读推荐 …… 778
19.6 作业 …… 779

第 20 章 数据库编程 …… 780

20.1 关系数据库 …… 780
20.2 MySQL 数据库管理系统 …… 781
20.3 SQL …… 783
20.4 JDBC 编程：通过 Java 调用 SQL …… 790
20.5 Mysql++编程：通过 C++调用 SQL …… 794
20.6 更多阅读推荐 …… 799
20.7 作业 …… 799

参考文献 …… 800

第 1 章

大处着眼——OO 编程的一些共性

1.1 什么是面向对象编程

虽说在我们阅读这本书的时候，这个问题的答案会逐渐浮现出来；但是，在现在这个时候，在面向对象编程（OO）和我们的现实世界之间提取一些共性还是颇有益处的。

大型面向对象程序的基本思想就是把大型软件（常常由数以百万计的代码行组成）看成是一个由对象所组成的社会：对象拥有足够的智能，能够理解从其他对象接收到的信息，并且以适当的行为对此作出反应；对象能够从高层对象继承属性和行为，并允许低层对象从自己继承属性和行为等。如果程序员对软件销售商所提供的对象感到不满意，在绝大多数情况下他都能够以一种相对容易的方式对这些对象进行扩展，对它们进行定制以适应他的特别需求。如果一个大型的采用非集中化组织的软件有一个组件出了问题，对其进行故障排除是非常简单的，这是由于它的局部化本质所决定的。

如果我们能够把共享一些共同特征的对象归纳成一个组，其效果会更好。我们可以把这些组称为类。例如，所有致力于健康保健事业的人肯定会有一些职业上的共性。我们可以定义一个 health-care professional 类，它包含了健康保健工作者的所有共同属性。我们还可以定义一个 medical doctor 类（代表所有的医生），把它作为 health-care professional 类的子类，它必须拥有 health-care professional 类的所有属性。同时，这个类必须具有一些额外的属性，代表医生所接受的专业教育和训练。

这个比喻可以直接运用到基于对象的软件设计中。拥有相同属性、展示相同行为的所有对象被分在一组，成为一个单一的类。事实上，我们首先定义一个类，然后通过一个称为“对类进行实例化”的过程创建这个类的多个单体对象。如果有一些对象除了拥有一个以前所定义的类的所有属性和行为外，另外还有一些额外的、更加具体的属性和行为，那么它们便被认为是以前所定义的那个类的子类。

1.2 OO 有什么优点

在过去几年里，面向对象编程成了图形用户界面（GUI）编程的首选。面向对象在这方面的能力是如此出众，以致人们在那些并不直接支持面向对象的语言（例如 C）中也创建了一些模拟 OO 的软件结构，以便进行 GUI 编程。这方面最有名的例子很可能是用于 GUI 设计的 GNOME/GTK+工具，它完全是用 C 编写的，但它在编程风格和结构上“非常地 OO”。为了进行对比描述，我们在第 17 章讲述 GUI 编程时将顺便讨论 GNOME/GTK+，当然那章的主要焦点还是 C++和 Java。除了 GUI 之外，OO 在数据库和网络编程方面也逐渐展现出强大的威力。

1.3 如何精通 OO

我们可以使用三管齐下的策略来精通 OO 编程模式，以便解决涉及庞大复杂系统的实际问题。当然，我们必须学习编程语言特有的语法。很显然，如果我们对一种语言的所有主要结构没有相当程度的熟悉，当我们面临一个问题时，我们很可能无法找到最为有效的解决方法。然而，这并不意味着我们必须记住所有的语法细节。例如，我们不可能单凭记忆记住所有不同的 Java 类以及每个类的所有属性和函数。幸运的是，在当今这个基于 Web 的文档时代，我们并不需要这样做。Java 编程的一种标准方法是我们一边在一个窗口编写程序，一边在另一个窗口浏览相关的在线文档。

除了掌握语法之外，我们必须精通每种语言中封装、继承和多态的概念。这三个概念是真正的 OO 语言的基石所在。每个概念的细节在不同的语言中可能存在微妙的差别。例如，C++允许多重继承，这向程序员提供了一定的自由度，但同时也增加了编写病态代码的风险。反之，Java 禁止使用 C++意义上的多重继承，但它允许一个类继承任意数量的接口（interface）。类似地，访问限定符（access modifier）允许我们在一个类中按照不同的访问级别对信息进行封装。但 C++和 Java 在这方面存在一些微小的差别。另外，Java 拥有程序包（package）的概念，它和访问控制有关，但这个概念在 C++中并不存在。多态允许我们按照处理父类型的方式对子类型进行处理。尽管在所有的 OO 语言中，这个概念的本质是一样的，但它的调用方式可能会对编程施加重要的约束条件。例如，在 C++中，如果想获得多态的行为，我们只能通过指针和引用来实现。这样一来，当我们需要对程序中的对象进行引用和操作时，无疑需要多加几番斟酌。

最后，我们还需要学习 OO 的设计。和所有的设计一样，OO 的设计也伴有一定的神秘感。这并不令人惊奇，因为要清楚地阐明涵括所有可能出现的问题（包括那些已经解决的和那些还没有解决的）的设计原则是不可能的。学习设计一个大型复杂问题的 OO 解决方案在很大程度上依赖于实践经验。当然，研究其他人所编写的良好 OO 代码对此可能也会有所帮助。然而，根据过去数年所积累的经验，下面这些方法应该是精通 OO 设计的必由之路：(1)精通一种“元”语言，例如 UML（统一建模语言，Unified Modeling Language），它允许我们在概念层次上可视化地描述我们的设计方案。(2)学习设计模式。在多年的实践中，人们已经总结出了大量的设计模式，它们已经成为许多子问题的模板解决方案。当我们逐步设计一个 OO 程序时很可能需要使用其中的一些模式。本书将在第 14 章对 UML 作了一个简单的介绍。但对于设计模式，本书并没有开辟专门的章节来介绍它们，不过本书所描述的许多示例代码包含了一些模式的实现方案。如果想深入钻研 UML 和设计模式，读者可以参阅这方面的一些非常优秀的书籍，如[7，13，20，21]等。