TURING 图灵计算机科学丛书

PEARSON

Object-Oriented Modeling and Design with UML

Second Edition

UML 面向对象建模与设计

（第2版）

[美] Michael Blaha James Rumbaugh 著
车皓阳 杨 眉 译

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵计算机科学丛书

Object-Oriented Modeling and Design with UML

Second Edition

UML 面向对象建模与设计

（第2版）

[美] Michael Blaha James Rumbaugh 著
车皓阳 杨 眉 译

人民邮电出版社
北京

图书在版编目（CIP）数据

UML面向对象建模与设计 /（美）布莱哈（Blaha,M.），（美）朗博（Rumbaugh,J.）著；车皓阳，杨眉译. --2版. -- 北京：人民邮电出版社，2011.7
（图灵计算机科学丛书）
书名原文：Object-Oriented Modeling and Design with UML，Second Edition
ISBN 978-7-115-22424-8

Ⅰ. ①U… Ⅱ. ①布… ②朗… ③车… ④杨… Ⅲ. ①面向对象语言，UML－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2011)第033507号

内 容 提 要

本书是“面向对象建模与设计”领域的经典著作。全书由四个部分组成。第一部分以一种高层的、独立于语言的方式描述面向对象的概念，以及UML表示法；第二部分从问题陈述到分析、系统设计和类设计，一步一步地描述了软件开发的面向对象方法学；第三部分用面向对象语言和关系数据库描述了面向对象设计的实现；第四部分描述了成功的面向对象开发所需要的软件工程实践。本书还配有丰富的习题，覆盖了一系列应用领域以及实现目标，而且在书的后面给出了部分习题的答案。

本书可以作为高年级本科生或研究生软件工程或面向对象技术课程的教材，也可以供相关技术人员参考。

图灵计算机科学丛书

UML面向对象建模与设计（第2版）

◆ 著 [美] Michael Blaha James Rumbaugh
译 车皓阳 杨 眉
责任编辑 杨海玲
执行编辑 李 哲

◆ 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 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：787×1092 1/16
印张：24.75
字数：618千字 2011年7月第2版
印数：21 001－25 000册 2011年7月北京第1次印刷

著作权合同登记号 图字：01-2005-5234号

ISBN 978-7-115-22424-8

定价：59.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版权声明

目　录

第 1 章　简介 …… 1
1.1　什么是面向对象 …… 1
1.2　什么是 OO 开发 …… 3
1.2.1　对概念而非实现建模 …… 3
1.2.2　OO 方法论 …… 3
1.2.3　三种模型 …… 4
1.3　OO 主题 …… 5
1.3.1　抽象 …… 5
1.3.2　封装 …… 5
1.3.3　组合数据和行为 …… 5
1.3.4　共享 …… 6
1.3.5　强调对象的本质 …… 6
1.3.6　协同 …… 6
1.4　关于 OO 开发有效性的证据 …… 7
1.5　OO 建模历史 …… 7
1.6　本书的组织结构 …… 8
参考文献注释 …… 8
参考文献 …… 9
习题 …… 9

第一部分　建模的概念

第 2 章　建模是一种设计技术 …… 12
2.1　建模 …… 12
2.2　抽象 …… 12
2.3　三种模型 …… 13
2.3.1　类模型 …… 13
2.3.2　状态模型 …… 13
2.3.3　交互模型 …… 14
2.3.4　模型间的关系 …… 14
2.4　小结 …… 14
参考文献注释 …… 14
习题 …… 15
第 3 章　类建模 …… 17
3.1　对象和类的概念 …… 17
3.1.1　对象 …… 17
3.1.2　类 …… 17
3.1.3　类图 …… 18
3.1.4　值和属性 …… 19
3.1.5　操作和方法 …… 20
3.1.6　类表示法小结 …… 21
3.2　链接和关联的概念 …… 21
3.2.1　链接和关联 …… 21
3.2.2　多重性 …… 23
3.2.3　关联端名 …… 24
3.2.4　排序 …… 26
3.2.5　包和序列 …… 26
3.2.6　关联类 …… 26
3.2.7　限定关联 …… 28
3.3　泛化和继承 …… 29
3.3.1　定义 …… 29
3.3.2　泛化的用途 …… 31
3.3.3　覆写特征 …… 31
3.4　类模型的一个示例 …… 32
3.5　类模型导航 …… 34
3.5.1　遍历类模型的 OCL 构件 …… 35
3.5.2　构建 OCL 表达式 …… 35
3.5.3　OCL 表达式的示例 …… 36
3.6　实践技巧 …… 37
3.7　小结 …… 38
参考文献注释 …… 39
参考文献 …… 40
习题 …… 40
第 4 章　高级类建模 …… 47
4.1　高级对象和类的概念 …… 47
4.1.1　枚举 …… 47
4.1.2　多重性 …… 48
4.1.3　作用域 …… 48
4.1.4　可见性 …… 49
4.2　关联端 …… 49
4.3　*n* 元关联 …… 50
4.4　聚合 …… 51
4.4.1　聚合与关联 …… 52
4.4.2　聚合与组合 …… 53
4.4.3　操作的传播 …… 53

4.5 抽象类 …… 54
4.6 多重继承 …… 55
4.6.1 多重继承的种类 …… 55
4.6.2 多重分类 …… 56
4.6.3 应对方案 …… 57
4.7 元数据 …… 58
4.8 具体化 …… 59
4.9 约束 …… 59
4.9.1 对象上的约束 …… 60
4.9.2 泛化集上的约束 …… 60
4.9.3 链接上的约束 …… 60
4.9.4 使用约束 …… 61
4.10 派生数据 …… 61
4.11 包 …… 62
4.12 实践技巧 …… 62
4.13 小结 …… 63
参考文献注释 …… 64
参考文献 …… 64
习题 …… 65
第5章 状态建模 …… 69
5.1 事件 …… 69
5.1.1 信号事件 …… 69
5.1.2 变更事件 …… 70
5.1.3 时间事件 …… 70
5.2 状态 …… 71
5.3 迁移和状态 …… 72
5.4 状态图 …… 73
5.4.1 状态图示例 …… 73
5.4.2 单触发状态图 …… 74
5.4.3 状态图的基本表示法小结 … 75
5.5 状态图行为 …… 76
5.5.1 活动效应 …… 76
5.5.2 do 活动 …… 76
5.5.3 进入和退出活动 …… 77
5.5.4 完成迁移 …… 78
5.5.5 发送信号 …… 78
5.5.6 带有活动的状态图示例 …… 78
5.6 实践技巧 …… 79
5.7 小结 …… 80
参考文献注释 …… 80
参考文献 …… 81
习题 …… 81
第6章 高级状态建模 …… 85
6.1 嵌套状态图 …… 85
6.1.1 平面状态图的问题 …… 85
6.1.2 扩展状态 …… 85
6.2 嵌套状态 …… 86
6.3 信号泛化 …… 88
6.4 并发 …… 89
6.4.1 聚合并发 …… 89
6.4.2 对象内的并发 …… 90
6.4.3 并发活动的同步 …… 90
6.5 状态模型示例 …… 91
6.6 类和状态模型的关系 …… 95
6.7 实践技巧 …… 96
6.8 小结 …… 97
参考文献注释 …… 97
参考文献 …… 98
习题 …… 98
第7章 交互建模 …… 102
7.1 用例模型 …… 102
7.1.1 参与者 …… 102
7.1.2 用例 …… 103
7.1.3 用例图 …… 104
7.1.4 用例模型的准则 …… 105
7.2 顺序模型 …… 106
7.2.1 场景 …… 106
7.2.2 顺序图 …… 106
7.2.3 顺序模型的准则 …… 108
7.3 活动模型 …… 109
7.3.1 活动 …… 110
7.3.2 分支 …… 110
7.3.3 初始和终止 …… 111
7.3.4 并发活动 …… 111
7.3.5 可执行活动图 …… 111
7.3.6 活动模型的准则 …… 111
7.4 小结 …… 112
参考文献注释 …… 112
参考文献 …… 112
习题 …… 113
第8章 高级交互建模 …… 115
8.1 用例关系 …… 115
8.1.1 包含关系 …… 115

8.1.2 扩展关系 …… 115
8.1.3 泛化 …… 116
8.1.4 用例关系的组合 …… 117
8.1.5 用例关系的准则 …… 118
8.2 过程式顺序模型 …… 118
8.2.1 带有被动对象的顺序图 …… 118
8.2.2 带有临时对象的顺序图 …… 119
8.2.3 过程化顺序模型的准则 …… 120
8.3 活动模型的特殊结构 …… 120
8.3.1 发送和接收信号 …… 120
8.3.2 泳道 …… 121
8.3.3 对象流 …… 122
8.4 小结 …… 122
参考文献 …… 123
习题 …… 123
第9章 概念小结 …… 126
9.1 类模型 …… 126
9.2 状态模型 …… 126
9.3 交互模型 …… 126
9.4 模型之间的关系 …… 127
9.4.1 泛化 …… 127
9.4.2 聚合 …… 128

第二部分 分析和设计

第10章 过程概述 …… 130
10.1 开发阶段 …… 130
10.1.1 系统构思 …… 130
10.1.2 分析 …… 131
10.1.3 系统设计 …… 131
10.1.4 类设计 …… 132
10.1.5 实现 …… 132
10.1.6 测试 …… 132
10.1.7 培训 …… 132
10.1.8 部署 …… 132
10.1.9 维护 …… 132
10.2 开发生命周期 …… 133
10.2.1 瀑布式开发 …… 133
10.2.2 迭代开发 …… 133
10.3 小结 …… 133
参考文献注释 …… 134
习题 …… 134
第11章 系统构思 …… 135
11.1 形成系统概念 …… 135
11.2 阐释概念 …… 135
11.3 准备问题陈述 …… 137
11.4 小结 …… 139
习题 …… 139
第12章 领域分析 …… 141
12.1 分析概述 …… 141
12.2 领域类模型 …… 142
12.2.1 寻找类 …… 143
12.2.2 保留正确的类 …… 144
12.2.3 准备数据词典 …… 145
12.2.4 寻找关联 …… 145
12.2.5 保留正确的关联 …… 146
12.2.6 寻找属性 …… 150
12.2.7 保留正确的属性 …… 150
12.2.8 使用继承来细化 …… 151
12.2.9 测试访问路径 …… 153
12.2.10 迭代类模型 …… 154
12.2.11 变换抽象的层次 …… 155
12.2.12 把类分组打包 …… 156
12.3 领域状态模型 …… 157
12.3.1 确定具有状态的类 …… 157
12.3.2 寻找状态 …… 158
12.3.3 寻找事件 …… 158
12.3.4 构建状态图 …… 158
12.3.5 评价状态图 …… 159
12.4 领域交互模型 …… 159
12.5 将分析迭代 …… 159
12.5.1 细化分析模型 …… 160
12.5.2 重述需求 …… 160
12.5.3 分析和设计 …… 161
12.6 小结 …… 161
参考文献注释 …… 161
参考文献 …… 162
习题 …… 162
第13章 应用分析 …… 169
13.1 应用程序交互模型 …… 169
13.1.1 确定系统边界 …… 169
13.1.2 寻找参与者 …… 170
13.1.3 寻找用例 …… 170

13.1.4 寻找初始和终止事件 …… 171
13.1.5 准备普通场景 …… 171
13.1.6 增加变化和异常场景 …… 172
13.1.7 寻找外部事件 …… 172
13.1.8 编制复杂用例的活动图 … 175
13.1.9 组织参与者和用例 …… 175
13.1.10 检查领域类模型 …… 175
13.2 应用类模型 …… 176
13.2.1 确定用户界面 …… 176
13.2.2 定义边界类 …… 177
13.2.3 确定控制器 …… 177
13.2.4 检查交互模型 …… 178
13.3 应用状态模型 …… 178
13.3.1 使用状态来确定应用类 … 179
13.3.2 寻找事件 …… 179
13.3.3 构建状态图 …… 179
13.3.4 检查其他状态图 …… 181
13.3.5 检查类模型 …… 182
13.3.6 检查交互模型 …… 182
13.4 增加操作 …… 183
13.4.1 来自类模型的操作 …… 183
13.4.2 来自用例的操作 …… 183
13.4.3 购物清单操作 …… 183
13.4.4 简化操作 …… 183
13.5 小结 …… 184
参考文献注释 …… 185
参考文献 …… 185
习题 …… 185
第14章 系统设计 …… 188
14.1 系统设计概述 …… 188
14.2 估算性能 …… 189
14.3 制订复用计划 …… 189
14.3.1 库 …… 189
14.3.2 框架 …… 190
14.3.3 模式 …… 190
14.4 将系统拆分成子系统 …… 191
14.4.1 分层 …… 192
14.4.2 分区 …… 192
14.4.3 组合分层和分区 …… 192
14.5 确定并发性 …… 193
14.5.1 识别内部的并发性 …… 193
14.5.2 定义并发任务 …… 194
14.6 分配子系统 …… 194
14.6.1 估算硬件资源需求 …… 194
14.6.2 权衡硬件和软件 …… 195
14.6.3 给处理器分配任务 …… 195
14.6.4 确定物理连通性 …… 196
14.7 管理数据存储 …… 196
14.8 处理全局资源 …… 197
14.9 选择软件控制策略 …… 198
14.9.1 过程驱动型控制 …… 198
14.9.2 事件驱动型控制 …… 199
14.9.3 并发控制 …… 199
14.9.4 内部控制 …… 199
14.9.5 其他范型 …… 199
14.10 处理边界条件 …… 200
14.11 设定权衡优先级 …… 200
14.12 常见的架构风格 …… 201
14.12.1 批处理转换 …… 201
14.12.2 连续型转换 …… 202
14.12.3 交互式界面 …… 203
14.12.4 动态仿真 …… 203
14.12.5 实时系统 …… 204
14.12.6 事务管理器 …… 204
14.13 ATM 系统的架构 …… 204
14.14 小结 …… 205
参考文献注释 …… 206
参考文献 …… 207
习题 …… 207
第15章 类设计 …… 212
15.1 类设计概述 …… 212
15.2 填补空白区 …… 213
15.3 实现用例 …… 214
15.4 设计算法 …… 215
15.4.1 选择算法 …… 215
15.4.2 选择数据结构 …… 216
15.4.3 定义内部类和操作 …… 217
15.4.4 把操作分配给类 …… 217
15.5 向下递归 …… 219
15.5.1 功能分层 …… 219
15.5.2 机制分层 …… 219
15.6 重构 …… 220
15.7 设计优化 …… 220
15.7.1 为了高效访问而增加冗余的关联 …… 221
15.7.2 为了效率而重新调整

执行顺序 …… 222
15.7.3 将派生值保存下来以避免重新计算 …… 222
15.8 行为具体化 …… 223
15.9 调整继承 …… 224
15.9.1 重新调整类和操作 …… 224
15.9.2 提取公共行为 …… 224
15.9.3 使用委托来共享行为 …… 225
15.10 组织类设计 …… 227
15.10.1 信息隐藏 …… 227
15.10.2 实体的内聚性 …… 227
15.10.3 微调包 …… 228
15.11 ATM 示例 …… 228
15.12 小结 …… 229
参考文献注释 …… 230
参考文献 …… 230
习题 …… 231
第 16 章 过程小结 …… 235
16.1 系统构思 …… 235
16.2 分析 …… 236
16.2.1 领域分析 …… 236
16.2.2 应用分析 …… 236
16.3 设计 …… 236
16.3.1 系统设计 …… 236
16.3.2 类设计 …… 236

第三部分 实现

第 17 章 实现建模 …… 238
17.1 实现概述 …… 238
17.2 微调类 …… 238
17.3 微调泛化 …… 240
17.4 实现关联 …… 241
17.4.1 分析关联遍历 …… 241
17.4.2 单向关联 …… 242
17.4.3 双向关联 …… 242
17.4.4 高级关联 …… 243
17.4.5 ATM 示例 …… 243
17.5 测试 …… 244
17.5.1 单元测试 …… 244
17.5.2 系统测试 …… 244
17.6 小结 …… 245
参考文献注释 …… 245
参考文献 …… 245
习题 …… 246
第 18 章 OO 语言 …… 247
18.1 简介 …… 247
18.1.1 C++简介 …… 247
18.1.2 Java 简介 …… 248
18.1.3 比较C++和 Java …… 249
18.2 简化的 ATM 模型 …… 250
18.3 实现结构 …… 250
18.3.1 数据类型 …… 251
18.3.2 类 …… 253
18.3.3 访问控制 …… 253
18.3.4 泛化 …… 255
18.3.5 关联 …… 258
18.4 实现功能 …… 260
18.4.1 创建对象 …… 261
18.4.2 对象的生存期 …… 263
18.4.3 对象销毁 …… 264
18.4.4 链接创建 …… 265
18.4.5 链接销毁 …… 267
18.4.6 派生属性 …… 267
18.5 实践技巧 …… 268
18.6 小结 …… 269
参考文献注释 …… 270
参考文献 …… 270
习题 …… 270
第 19 章 数据库 …… 274
19.1 简介 …… 274
19.1.1 数据库的概念 …… 274
19.1.2 关系数据库的概念 …… 275
19.1.3 范式 …… 276
19.1.4 选择 DBMS 产品 …… 276
19.2 简化的 ATM 模型 …… 277
19.3 实现结构——基础 …… 277
19.3.1 类 …… 277
19.3.2 关联 …… 278
19.3.3 泛化 …… 280
19.3.4 标识 …… 281
19.3.5 RDBMS 实现的基本规则小结 …… 282
19.4 实现结构——高级 …… 283
19.4.1 外键 …… 283
19.4.2 检查约束 …… 284
19.4.3 索引 …… 284

19.4.4 视图 …… 284
19.4.5 小结 RDBMS 实现的高级规则 …… 285
19.5 为 ATM 示例实现结构 …… 285
19.6 实现功能 …… 288
19.6.1 将程序设计语言耦合到数据库中 …… 288
19.6.2 数据转换 …… 290
19.6.3 封装与查询优化 …… 290
19.6.4 使用 SQL 代码 …… 291
19.7 面向对象数据库 …… 291
19.8 实践技巧 …… 292
19.9 小结 …… 293
参考文献注释 …… 293
参考文献 …… 293
习题 …… 294
第 20 章 程序设计风格 …… 298
20.1 面向对象的风格 …… 298
20.2 可复用性 …… 298
20.2.1 可复用性的类别 …… 298
20.2.2 可复用性的风格准则 …… 299
20.2.3 使用继承 …… 300
20.3 可扩展性 …… 301
20.4 健壮性 …… 302
20.5 大规模程序设计 …… 303
20.6 小结 …… 305
参考文献注释 …… 306
参考文献 …… 306
习题 …… 306

第四部分 软件工程

第 21 章 迭代开发 …… 310
21.1 迭代开发概述 …… 310
21.2 迭代开发与瀑布式开发 …… 310
21.3 迭代开发与快速原型法 …… 311
21.4 迭代的适用范围 …… 311
21.5 执行一次迭代 …… 312
21.6 规划下一次迭代 …… 313
21.7 建模和迭代开发 …… 313
21.8 识别风险 …… 314
21.9 小结 …… 314
参考文献注释 …… 315
参考文献 …… 315
第 22 章 管理建模 …… 316
22.1 管理建模概述 …… 316
22.2 模型的类型 …… 316
22.3 建模的缺陷 …… 317
22.4 建模会话 …… 318
22.4.1 密室建模 …… 318
22.4.2 轮转建模 …… 319
22.4.3 实况建模 …… 319
22.5 组织人员 …… 320
22.6 学习技术 …… 321
22.7 教授技术 …… 322
22.8 工具 …… 322
22.8.1 建模工具 …… 322
22.8.2 配置管理工具 …… 323
22.8.3 代码生成器 …… 323
22.8.4 模拟工具 …… 323
22.8.5 库 …… 324
22.9 估算建模工作量 …… 324
22.10 小结 …… 324
参考文献注释 …… 325
参考文献 …… 325
第 23 章 遗留系统 …… 327
23.1 逆向工程 …… 327
23.1.1 逆向工程与正向工程 …… 327
23.1.2 逆向工程的输入 …… 327
23.1.3 逆向工程的输出结果 …… 328
23.2 构造类模型 …… 328
23.2.1 实现复原 …… 328
23.2.2 设计复原 …… 329
23.2.3 分析复原 …… 329
23.3 构造交互模型 …… 329
23.4 构造状态模型 …… 330
23.5 逆向工程的技巧 …… 330
23.6 包装 …… 330
23.7 维护 …… 331
23.8 小结 …… 332
参考文献注释 …… 332
参考文献 …… 332
附录 A UML 图形化表示法 …… 334
附录 B 术语表 …… 335
部分习题答案 …… 346
索引 …… 368

第1章 简 介

面向对象建模与设计是使用现实世界的概念模型来思考问题的一种方法。其基本结构是对象，对象既包含数据结构，又包含行为。对于理解问题、与应用领域专家交流、建模企业级应用、编写文档、设计程序和数据库来说，面向对象模型都非常有用。本书提出了一套面向对象的表示法并且从分析到设计再到实现扩展出了一种过程。在开发过程的所有阶段里，都可以应用相同的表示法。

1.1 什么是面向对象

表面来看，**面向对象**（OO）这个术语的意思是，把软件组织成一系列离散的、合并了数据结构和行为的对象。这与以前的开发方法中数据结构和行为只是松散关联是不同的。关于OO方法究竟需要哪些特征，人们还有争议，但大体上它们都包含标识（identity）、分类（classification）、继承（inheritance）和多态（polymorphism）四个方面。

标识的意思是指，数据被量化成称为**对象**（object）的离散的、可辨识的实体。本章的第一段、我的工作站、棋盘中的白棋皇后都可以称作一个对象。图1-1显示了另外一些对象。对象可以是具体的，例如文件系统中的文件，也可以是概念上的，例如多处理器操作系统中的调度策略。每个对象都有它自己的内部标识。换句话说，即使所有属性值（例如姓名和大小）都相同，两个对象也是有差别的。

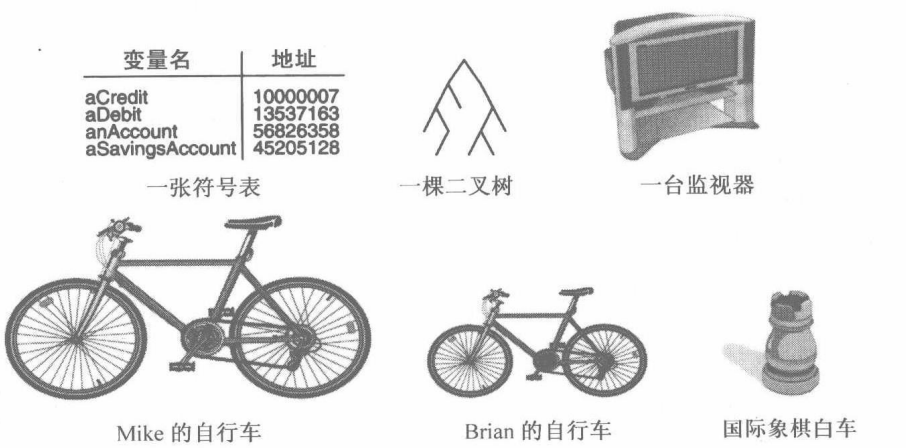

图1-1 对象。对象是面向对象技术的核心概念

在现实世界中，对象只是一种简单的存在，但在编程语言中，每个对象都有一个唯一的句柄，借助这个句柄就可以引用对象。不同的语言实现句柄的方式不同，例如地址、数组下标或人为编号。这样的对象引用是一致并且独立于对象内容的，允许创建混合的对象集合，例如同时包含文件和子目录的文件系统目录。

分类指的是有着相同数据结构（**属性**）和行为（**操作**）的对象被分组为一个类。段落、显示器和棋子都是类的示例。**类**（class）是一种抽象，描述对于一项应用来说很重要的属性，并且忽略其余属性。类的选择是随意的，视应用而定。

每个类都描述了由各个对象组成的无限集合。每个对象都是该类的一个**实例**（instance）。对于每种属性，对象都有其自己的取值，但会和此类的其他实例共享属性名和操作。图 1-2 显示了两个类和它们各自的一些实例。对象包含了到其自身类的隐含引用，它“知道自己是什么东西”。

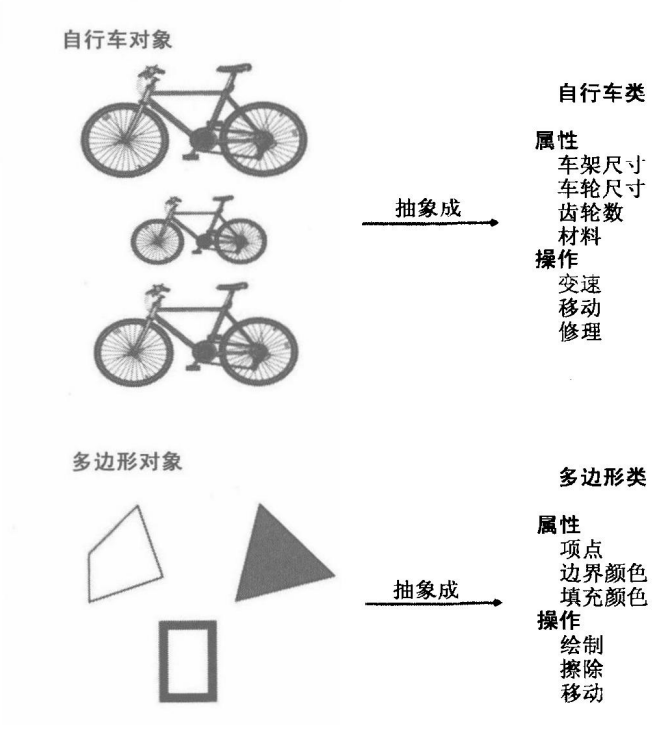

图 1-2 对象和类。每个类都描述了由各个对象组成的无穷集合

继承指的是多个类基于一种分层关系，共享类间属性和操作（合称**特征**）。**父类**（superclass）拥有**子类**（subclass）提炼和详细说明的通用信息。每个子类包含（或称继承）其父类全部的特征，再加上它自己所特有的特征。子类不需要重复其父类的特征。例如，*ScrollingWindow* 和 *FixedWindow* 是 *Window* 的子类。两个子类都继承 *Window* 的特征，例如屏幕上的可见区。*ScrollingWindow* 增加了滚动条和偏移。把几个类的公共特征提取出来组成一个父类，这种功能可以大大减少设计和程序内部的重复劳动，这是 OO 技术的一项主要的优点。

多态是指对于不同的类来说，相同的操作会有不同的动作。例如，在国际象棋游戏中，兵和后的*移动*（move）操作是不同的。**操作**是对象执行或被执行的一个过程或转换。*右对齐*（RightJustify）、*显示*（display）和*移动*都是操作的例子。由特定类实现的操作被称为**方法**（method）。因为 OO 操作符是多态的，所以它可能会有多种实现方法，对于不同对象的类来说，每一个都会有一种实现方法。

在现实世界中，操作只不过是不同的对象之间相似行为的一种抽象。每种对象“知道如何”执行它自己的操作。然而，在 OO 编程语言中，基于操作名和被操作对象的类，语言会

自动选择正确的方法来实现操作。操作的用户不需要知道有多少种方法实现了给定的多态操作。无须改动现有代码，开发者就可以增加新类，只要他们为每一种可行操作提供方法即可。

1.2 什么是 OO 开发

本书讨论的是 OO 开发，是基于现实世界以及程序中的抽象为基础来思考软件的方式。在此种背景下，**开发**（development）指的是软件生命周期，即分析、设计和实现。OO 开发的本质是识别和组织应用领域中的概念，而不是以一种编程语言最终表示这些概念。Brooks 观察到，因为问题内在的复杂性，软件开发中困难的部分是对其本质的操作，而不是将它映射成某种语言的这些次要方面［Brooks-95］。

本书没有明确地讲述集成、维护和完善等问题，但用准确的表示法完成清晰的设计会有利于软件的整个生命周期。用来表示设计的 OO 概念和表示法也能提供有用的文档。

1.2.1 对概念而非实现建模

过去，OO 界的大多数人都专注于编程语言，有关文献的重点也都在实现上，而不是分析和设计上。最初在解决传统开发语言中那些不灵活的问题时，OO 编程语言显示出强大的威力。对于软件工程来说，这种专注可以说是某种意义上的倒退——它过度注重实现机制，而不是它们所支持的底层思维过程。

真正的收效来源于解决前端的概念性问题，而不是后端的实现细节。修正在实现过程中显现出来的设计缺陷要花费高昂的代价，不如在早期发现它们。过早专注于实现会限制设计决策，常常会导致劣质产品的出现。OO 开发方法鼓励软件开发者在软件生命周期内应用其概念来工作和思考。只有较好地识别、组织和理解了应用领域的内在概念，才会有效表达出数据结构和函数的细节。

OO 开发只有到了最后几个阶段才不是独立于编程语言的概念过程。根本上来讲，OO 开发是一种思维方式，而不是一种编程技术。它的最大好处在于，帮助规划人员、开发者和客户清晰地表达抽象的概念，并将这些概念互相传达。它可以充当规约、分析、文档、接口以及编程的一种媒介。

1.2.2 OO 方法论

我们提出一种 OO 开发过程和一种图形化表示法来表示 OO 的概念。这个过程先是构建一种应用模型，然后在设计中增加细节。从分析到设计再到实现使用的是相同的无缝表示法，这样某一个开发阶段增加的信息就不会在下一阶段中丢失或转化。这种方法论包括下面几个阶段。

- **系统构思**。软件开发始于业务分析人员或用户构思一项应用，并制订临时性需求。
- **分析**。通过创建模型，分析人员仔细审查并严格地重新描述系统构思阶段的需求。因为问题描述很少会是完整的或是正确的，分析人员必须与请求者一起工作以理解问题。分析模型是一种简明准确的抽象，它描述目标系统要做哪些事情，而不是要如何来做这些事情。分析模型不应该包含任何实现决策。例如，工作站窗口系统中的 *Window* 类可以用其可视化属性和操作来描述。

分析模型有两个部分：**领域模型**（domain model），描述系统内部反映的现实世界的对象；**应用模型**（application model），描述用户可见的应用系统本身的组成部分。例如，对于股票经纪人系统来说，领域对象包括股票、债券、交易和佣金，应用对象会控制交易的执行过程，并给出结果。一些应用专家虽然本身不是程序员，但他们能够理解并评议好的模型。

4

- **系统设计**。开发团队设计出一种高层的策略——**系统架构**（system architecture），用于解决应用程序的问题。他们也制定政策，默认作为后面更加详细的设计内容。系统设计人员必须要确定优化哪些性能特性，选择何种策略来解决问题，完成哪些临时性的资源分配。例如，系统设计人员决定工作站屏幕上的变化必须快速且平滑，即使是在窗口移动或删除的情况下也是如此，他们要选择一种适当的通信协议和内存缓冲策略。
- **类的设计**。根据系统设计策略，类的设计者给分析模型添加细节。类的设计者使用相同的 OO 概念和表示法阐释领域对象和应用对象，尽管它们存在于不同的概念层面上。类设计的焦点在于实现每个类的数据结构和算法。例如，类的设计者现在也要为 *Window* 类的每项操作确定数据结构和算法。
- **实现**。实现人员将类的设计阶段开发的类及其关系转换到某种编程语言、数据库或硬件上。程序设计应该是简单直接的，因为所有困难的决策都已经完成。在实现阶段，遵循良好的软件工程实践是很重要的，这样，设计的可追溯性就非常清晰，系统将保持灵活性和可扩展性。例如，实现人员会借助工作站下层图形系统的调用，用某种语言编写出 *Window* 类。

从分析到设计再到实现，OO 概念会应用到系统开发的整个生命周期中。你可在各个阶段使用相同的类，不用改变符号记法，只在后续阶段增添细节。*Window* 的分析和设计模型都是正确的，但它们服务于不同的目标，表示不同的抽象。整个开发过程使用的标识、分类、多态和继承等概念都是相同的。

注意，我们并不推荐瀑布式开发过程，即先捕获需求，再分析，然后设计，最后实现。对于系统的某个部分来说，开发者必须按顺序执行每个阶段，但其实他们不需要并行地开发系统的每个组成部分。我们提倡迭代过程——历经几个阶段开发部分系统，然后增加功能。

有一些类虽然不是分析的一部分，但却会在设计和实现阶段引入。例如，*树*、*散列表*、*链表*等数据结构在现实世界中几乎找不到，对用户来说也看不见摸不着。设计者引入它们是为了支持特殊的算法。这些数据结构对象存在于计算机当中，并不能被直接观察到。

我们也没有把测试看成是一个独立的步骤。测试虽然重要，但它必须是生命周期内整个质量控制的一部分。开发者必须依据现实情况检查分析模型。除了测试实现的正确性之外，他们也必须验证设计模型不会出现各类错误。把质量控制囿于单个步骤，代价会更大，而效率反而会更低。

5

1.2.3 三种模型

我们使用三种模型从不同的视角来描述系统：描述系统内部对象及其关系的类模型，描述对象生命历史的状态模型，以及描述对象之间交互行为的交互模型。每种模型都会在开发的所有阶段中得到应用，并随着开发过程的进行获得更多的细节。对系统的完整描述需要所有这三种视角的模型。

类模型（class model）描述了系统内部对象及其关系的静态结构。类模型界定了软件开发的上下文——论域（universe of discourse）。类模型包含类图。**类图**（class diagram）的结点是类，弧表示类间的关系。

状态模型（state model）描述了对象随着时间发生变化的那些方面。状态模型使用状态图确定并实现控制。**状态图**（state diagram）的结点是状态，弧是由事件引发的状态间的转移。

交互模型（interaction model）描述系统中的对象如何协作以完成更为宽泛的任务。交互模型自用例开始，用例随后会用顺序图和活动图详细描述。**用例**（use case）关注系统的功能，即系统为用户做了哪些事情。**顺序图**（sequence diagram）显示交互的对象以及发生交互的时间顺序。**活动图**（activity diagram）描述重要的处理步骤。

这三个模型描述了一套完整的系统的相互独立的部分，但它们又是交叉相连的。类模型是最基本的，因为在描述何时以及如何发生变化之前，要先描述是哪些内容正在发生变化或转化。

1.3 OO 主题

有几个主题在 OO 技术中无处不在。尽管这些主题并不是 OO 系统所特有的，但 OO 系统对它们的支持却极为出色。

1.3.1 抽象

抽象（abstraction）使我们可以专注于应用程序最本质的那些方面，同时忽略细节。这意味着在确定如何实现功能之前，要先关注对象是什么，做了什么。使用抽象机制，可以有充足的时间去自由地制定决策，避免过早做出不成熟的细节承诺。许多现代的开发语言都提供了数据抽象的功能，但继承和多态使其更加强大。对于 OO 开发来说，抽象的能力可能是最重要的一项技能。

1.3.2 封装

封装（encapsulation）将对象的外部因素（可以被其他对象访问）与内部实现细节（其他对象不可见）分离开来。封装避免程序的各个组成部分过于互相依赖，否则很小的变化也会引起巨大的连锁反应。可以改变对象的实现，而不影响使用它的应用程序。也可以改变对象的实现以改进其性能，调试错误，完善代码或支持移植。封装不是 OO 语言所特有的，但它能将数据结构和行为组织在一个实体中，因此比以前的语言（例如 Fortran、Cobol 和 C）更整洁、更有力。

6

1.3.3 组合数据和行为

操作的调用者不需要考虑有多少种实现存在。多态操作符通过类层次结构来确定使用哪种实现，而不需要负责调用的代码来承担这项工作。例如，显示窗口内容的非 OO 代码必须区分每幅图形的类型，判断它是多边形、圆还是文本，并调用相应的过程来显示它。OO 程序只是调用每幅图形上的 *draw* 操作，每个对象基于自己的类隐含地确定要使用哪个过程。这样

维护会更加容易，因为在增加新类的时候，调用代码不需要修改。在 OO 系统中，数据结构层次与操作继承层次匹配（见图 1-3）。

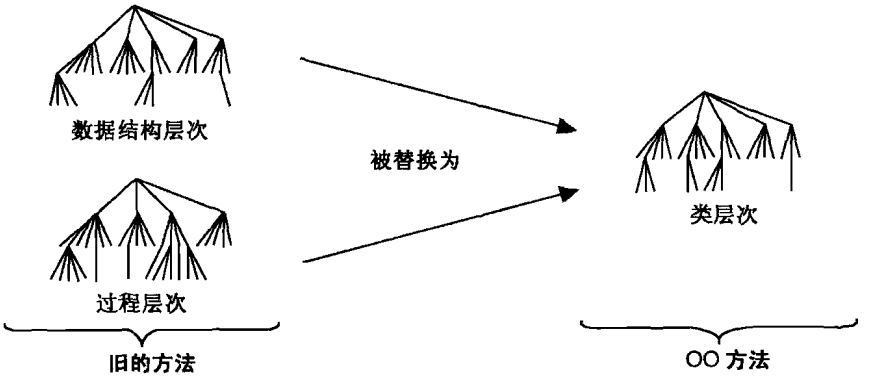

图 1-3　OO 与以前的方法。对于数据和行为，OO 方法有一个统一的层次

1.3.4　共享

OO 技术有利于在不同层次上共享。继承数据结构和行为使得子类可以共享通用代码。通过继承完成共享是 OO 语言的一个主要优点。比节省代码量更重要的是概念上更清晰，认识到不同的操作实际上都在做相同的事情，这样就减少了必须理解和分析的不同情形。

OO 开发不仅使我们可以在应用程序中共享信息，而且还让我们有机会在未来的项目中复
7 用设计和代码。OO 开发提供了抽象、封装和继承等工具，来创建可复用的组件库。但是，复用往往被过度地强调为选用 OO 技术的理由。其实，复用不会想当然地发生，开发者必须思考眼前应用之外的事情，投入额外的精力，才能获得更通用的设计。

1.3.5　强调对象的本质

OO 技术强调对象是什么，而不是如何使用它。对象的用法依赖于应用的细节，经常需要在开发过程中随机应变。随着需求的演化，对象的特性会比使用它的方式更加稳定，因此，构建在对象结构上的软件系统最终也会比较稳定。与功能分解方法相比，OO 开发更注重数据结构，而较少关注过程结构。这样来看，OO 开发与数据库设计中用到的信息建模技术非常相似，只是 OO 开发增加了一个概念——与类相关的行为。

1.3.6　协同

标识、分类、多态和继承都是 OO 语言的特色。这些概念中的每一个都可以分开使用，但结合在一起，它们就会互相促进，互为补充。OO 方法的好处比一开始看上去的要多。强调对象的本质会迫使开发者更加仔细和深入地思考对象是什么，做了些什么。这比只关注数据和操作的做法，所生成的最终系统会更加干净、更加通用和更加健壮。

1.4 关于OO开发有效性的证据

我们的OO开发工作始于GE研发中心的内部应用。我们用OO技术来开发编译器、图形、用户界面、数据库、OO语言、CAD系统、仿真、元模型、控制系统以及其他一些应用。我们用OO模型来评述结构不良、难以理解的程序。我们的实现目标从OO语言，到非OO语言，一直扩展到数据库。我们成功地把这种方法传授给了其他人，并用它来与应用专家交流。

自从20世纪90年代中期以来，我们已将OO技术的实践从GE扩展到了全世界。在我们编写本书第1版的时候，面向对象和OO建模都是相对较新的方法，大家也都没有太多大规模应用的经验。现在，OO技术不会再被认为是一种赶时髦或试探性的方法了，它已经成为计算机科学和软件工程的主流。

每年一届的OOPSLA（面向对象编程系统、语言及应用）、ECOOP（欧洲面向对象编程大会）和TOOLS（面向对象语言与系统技术）等会议都是传播新的OO思想和应用成果的重要平台。这些会议的论文集描述了许多从OO方法中获益的应用。OO系统方面的文章也出现在许多重要的期刊上，例如*IEEE Computer*和*Communications of the ACM*。 8

1.5 OO建模历史

我们在GE研发中心的工作最后产生了对象建模技术（Object Modeling Technique，OMT），它是1991年本书前一版中介绍的内容。OMT是成功的，但其他一些方法也很成功。OO建模的流行出现了新的问题——有太多的表示法可以选择。它们表达了相同的概念，但符号不同，令开发者迷惑不已，使得交流难以为继。

结果，软件界开始集中力量合并这些不同的表示法。在1994年，Jim Rumbaugh加入了Rational公司（现在是IBM公司的一部分），开始和Grady Booch一起工作，统一了OMT和Booch表示法。1995年，Ivar Jacobson也加入了Rational公司，并把Objectory表示法也加入到统一化工作中。

1996年，对象管理组织（OMG）发出请求，提议完成一套标准OO建模表示法。好几个公司随后响应，最后那些竞争性提议被合并为最终提议。Rational公司领导了最终的提议小组，Booch、Rumbaugh和Jacobson都深入地参与了此项工作。在1997年11月，OMG全体一致地接受统一建模语言（UML）作为标准。参与公司将UML的权利转交给OMG，OMG拥有UML的商标和规范，并控制着UML未来的发展。

UML极为成功，取代了各类出版物上的其他表示法。无论出于自愿，还是迫于市场的压力，许多其他方法的作者也都采纳了UML表示法。UML终结了表示法的战争，它现在显然已是公认的OO表示法。在本书中我们使用UML，正是因为它是标准的表示法。

2001年，OMG成员启动了修订工作，增加在最初规范中遗漏的功能，并修正在UML 1的实践过程中发现的问题。本书是以2004年批准的UML 2.0版本为基础的。要访问这些官方的规范文档，请参阅OMG的网站www.omg.org。