来自一线软件开发企业技术总监的培训课程**精髓**
了解真正面向对象设计**深层的精义**，而非肤浅机械的**生搬硬套**
是众多大师和**经典著作之提炼归纳和浓缩**，也是通向面向对象**奇妙世界的方向指引**

面向对象开发参考手册

黄磊　编著

人民邮电出版社
POSTS & TELECOM PRESS

面向对象开发参考手册

黄磊　编著

人民邮电出版社
北京

图书在版编目（CIP）数据

面向对象开发参考手册 / 黄磊编著. -- 北京 : 人民邮电出版社, 2014.1
ISBN 978-7-115-33348-3

Ⅰ. ①面… Ⅱ. ①黄… Ⅲ. ①面向对象语言－程序设计－手册 Ⅳ. ①TP312-62

中国版本图书馆CIP数据核字(2013)第238102号

内 容 提 要

面向对象软件设计的经典书籍，如《敏捷软件开发》、《领域驱动设计》、《设计模式》、《测试驱动开发》、《极限编程》、《重构》等，已名声在外，其解读书籍也多如牛毛。但其往往只讲述某个方面，要整体理解，必须通读原著，阅读量颇大，特别原著比较深奥，短时间内很难完全理解。市面上缺乏整体归纳、提炼浓缩的书籍。

本书致力于让读者形成一个整体、全面的概念和印象，浓缩、提炼了经典书籍的精华，结合作者自身十几年的经验，力争呈现一本深入浅出、兼收并蓄、涉及各个方面的综合版本，可以缩减学习的时间成本，并能够像工具书一样翻阅参考。

本书是在作者为其公司开发人员进行的 100 多个课时的培训的基础上按照培训内容整理而成的。从这个角度来说，它特别适合开发人员学习使用，尤其适于那些刚毕业的“菜鸟”们学习使用。当然，资深开发人员也可以经常翻阅本书来寻找灵感。

◆ 编　　著　黄　磊
责任编辑　杨海玲
责任印制　程彦红　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：17
字数：379 千字　　2014 年 1 月第 1 版
印数：1－3 000 册　　2014 年 1 月河北第 1 次印刷

定价：49.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前　言

传统的软件工程学科，正在从冰冷坚硬的技术无机物，逐渐进化为充满人文主义的温暖柔和的生命有机体。

从软件的外观交互而言，用户的心理模型替代了机械化的实现模型，软件从一个粗鲁、丑陋、不自信、孩子气、没有礼貌、反应迟钝、肤浅无知、固执虚伪、需要让人去忍受的家伙，变成了一个斯文、体贴、睿智、乖巧、大度、知书达理的谦谦君子。是人，而不是软件，成为了应用的中心。

从软件的内部构造而言，从笨拙的机制化的过程，变成了对象之间的协作。对象一如我们人类，他们有血肉、有灵魂、有思想、有职责、也有隐私；他们爱恨分明——有所为、也有所不为；他们会交流——使用通用语言和模式语言进行交谈；他们会思考——用抽象来思考领域中发生的事情；他们有个性——各自定义属于自己的行为规则；他们结群而居，又理智地保持着彼此的距离；而代码却日益成为自然语言的表达——去揭示暗藏在复杂世界表象下的本质和规律。

从软件的过程组织而言，瀑布模型把程序员视为没有个性和思想的流水线操作工，傲慢而高高在上的系统分析员、设计人员把所谓“需求”和“设计”交给他们去编码，如同过时的泰勒主义一般人为地制造着形式化的垃圾和团队中的裂痕；而简约和渐进的敏捷过程，则为程序员的生存方式带来变革，沟通、反馈、简单、勇气、尊重成为我们的哲学价值观，管理人员、开发人员和用户之间的互相伤害变成了协调、平衡，这日益成为我们的一种生活方式和文化。

过去的软件，言其无机，因为它只会简单地堆砌功能和模块；言其坚硬，因为软件不软，不能灵活快速地适应客户和环境的变化；言其冰冷，因为纯粹从技术、机制、过程的角度考虑而实现的软件，不就是一台冷冰冰的机器吗？

现在，让我们给它温暖，让它变得柔和，让它充满人文关怀和深刻思想，让它成为绽放活力和价值的生命有机体。

在专业学科和社会分工愈来愈细致的同时，各学科间的界限又变得越来越模糊，因为真理并非属于哪一个学科，相反，它总是相通的。当我像写散文一般写技术文章的时候，我隐约看见，Alexander 深邃的目光穿越了宇宙和未来，这是一个宽广、深刻、充满挑战的世界，语言并不足以传其神妙。

这本小册子浓缩了笔者所在公司对软件开发人员近一年的培训课程中的精华部分。这些精华，凝聚了前辈大师们的智慧和心血，是软件开发行业的“圣经”。后来者如我们，不需要为之

顶礼膜拜，而是要将其融入自己的思想和智慧。

学习从来是没有诀窍的。如果一定要说有，那么有三条：第一是重复；第二是重复；第三，还是重复。思想的改造更是缓慢而深刻的，或是水滴石穿，或是豁然顿悟，其中的精义都需要我们慢慢去认识和体会。所以，这不是一本可以读完一遍以后就束之高阁的书，您需要把它放在您的桌面，在设计过程中反复参阅以获取设计灵感，即便在闲暇的时候，也可以细细地琢磨、领悟其中的微言大义。

目　录

即使使用了面向对象的编程语言，我们仍然可能以“面向对象之名”，行“面向过程之实”。系统设计仍然可能在结构化设计的圈圈中原地打转。

编程语言仅仅是一种语法规则，不可能依赖编程语言的面向对象机制，来掌握面向对象。单纯从编程语言上获得的面向对象知识，不能胜任面向对象设计与开发。

正如学会了中文，有的人可以写出《红楼梦》，有的人可以造出“写诗机”和“梨花体”。

因此，仅仅知道封装、继承和多态并不足以做出好的面向对象设计，我们需要重新认识对象，以及面向对象设计的精髓。

第1章 转变观念

1.1 传统的面向对象

传统的面向对象教科书中，描述了对象的三个基本特征。

（1）**封装**，即内部的改动不会对外部产生影响。例如，访问数据的对象可以使用ADO或DAO对象模型，也可以直接使用ODBC API函数，但都不会影响其外部特性。

（2）**继承**，通过派生来解决实现的重用问题。例如，从SalesOrder类派生出WebSalesOrder类，在WebSalesOrder中，可以重载父类的Confirm方法（发邮件而不是传真），也可以自动继承实现父类的Total方法，实现相同的行为。

（3）**多态**（可替代性），不论何时创建了派生类对象，在使用基类对象的地方都可以使用此派生类对象。不同类型的对象就可以处理交互时使用的一组通用消息，并且以它们各自的方式进行。如前面的例子中，WebSalesOrder“is a”SalesOrder，也就是说，在任何使用SalesOrder的地方，都可以使用WebSalesOrder。

对象之间的关系有以下四种。

（1）**聚合关系**。比如，A聚合了B，B是A的一部分，则表示为A has a B，例如“飞机场has a飞机”。它在UML静态类图中的表示如图

1-1 所示。

（2）**组合关系**。比如 A 是由 B 组成的，A 包含 B，B 是 A 的一部分，则也表示为 A has a B，例如“飞机 has a 发动机”。它在 UML 静态类图中的表示如图 1-2 所示。

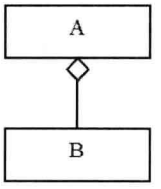

图 1-1　对象的聚合关系

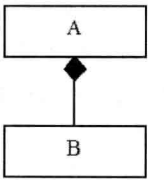

图 1-2　对象的组合关系

（3）**继承关系**。比如 A 派生了 B，B 是 A 的一种，A 是 B 的泛化，则表示为 B is a A。例如，“波音 777 is a 飞机”。它在 UML 静态类图中的表示如图 1-3 所示。

（4）**依赖关系**。比如 A 依赖 B，A 使用 B，则表示为 A use a B。例如，“飞机 use a 飞行员”。它在 UML 静态类图中的表示如图 1-4 所示。

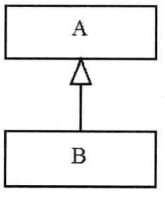

图 1-3　对象的继承关系

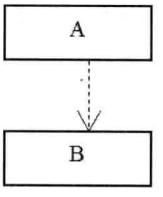

图 1-4　对象的依赖关系

在传统的面向对象中，我们像下面这样实现对象的基本特征。

1．基于组件的封装。

传统的封装是基于组件的封装。这个时候，模块功能的改变仍然会影响到其他模块。例如，当使用一个类的时候，我们必须清楚地知道这个类有哪些方法、属性、行为和数据，而且不能简单地用另外一个类来替换掉这个类，因为我们还必须对所有对这个类进行过引用的代码进行改变。即使不改变类的方法、域、属性的名字，而只改变类的实现代码，也不能轻易地改变，因为还要进行重新编译。当很多不同组件中的类都对另外一个组件中的类进行了引用的时候，情况有可能变得更糟。

基于组件的封装最明显的后果就是“DLL 灾难”。多重应用会依赖于同一个 DLL，只要其中的一个应用被更新，这个 DLL 就要相应地改动以适应这个应用的新版本，随之而来的是其他所有依赖于这个 DLL 的应用都要改动。

2．基于实现的继承。

传统的继承是基于“白盒重用”的实现继承，是紧耦合的重用。这样会将父类的内部结构暴露出来，继承会把子类紧紧耦合在其父类上，这意味着对父类的修改可能会是子类的灾难。改变层顶端的类通常需要改变许多次级类。而另一方面，冻结关键上级类接口通常会产生一个不能扩展的系统。还有，没有必要的继承层次结构，往往是低内聚、紧耦合的，继承必须限制在一个单独的层次结构中（除非使用多继承）。例如图 1-5 所示的继承结构。

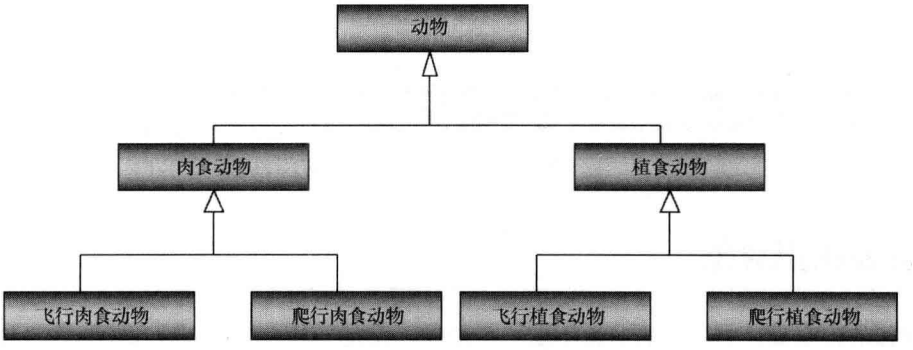

图 1-5 基于实现的继承例子

最初我们按照进食特性从动物派生出肉食动物和植食动物，然后按照其移动特性分别从肉食动物和植食动物中派生出了飞行肉食动物、爬行肉食动物、飞行植食动物、爬行植食动物。

可是如果需求变化了，飞行的动物也要求能在地上行走，怎么办？更多的需求变化来了，比如要按哺乳方式分类，怎么办？继续往下派生？进而产生按级数增长的天文数字般的类？

下面我们再来看一个典型的为了适应多个不同客户需求的应用软件的统一版本的“标准面向对象解决方案”，如图 1-6 所示。

这个解决方案存在以下一些问题。

- 多态：无法获得跨客户的多态。
- 冗余：XXX For C1 和 XXX For C2 业务之间存在冗余。
- 杂乱。
- 紧耦合：不同业务间接地相互关联（例如，某个客户的某个管理模式需要体现在各个不同的业务中）。

- 弱内聚：相同的业务处理分散在多个类中。
- 类爆炸：如果发生变化（更多的客户，或者更多的业务），存在着失控的危险。

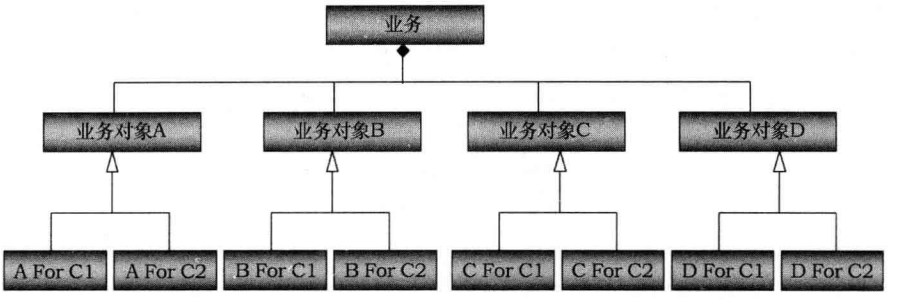

图 1-6　统一版本的“标准面向对象解决方案”

1.2　重新认识对象

在现代的程序设计中，面向对象不是一个选择，而是一种必需。在现代的程序设计语言中，万物皆对象。

过去我们以为，有了可视化编程，然后把数据和方法装到一个类里面，就是面向对象了。现在我们知道，这种认识有多肤浅。因为面向对象并非是目的，我们的目的是设计“高内聚、松耦合”的软件以应对变化。只有“好的”面向对象设计才能做到这一点。

传统的 OOP 认识到的不是面向对象的全部，甚至只是其浅陋的部分。传统的 OOP 没有回答面向对象的根本性问题，即我们为什么要使用面向对象，我们应该怎样使用三大机制来实现“好的面向对象”，我们应该遵循什么样的面向对象原则。

传统 OOP 的三大机制“封装、继承、多态”可以表达面向对象的所有概念，但并没有刻画出面向对象的核心精神。程序员既可以用这三大机制做出“好的面向对象设计”，也可以用这三大机制做出“差的面向对象设计”。

面向对象，就是只考虑它的意图，而不是它的实现机制。

面向对象的精髓在于“封装”。

传统的面向对象认为封装就是隐藏数据。实际上，对象是有责任的实体，封装是隐藏一切，包括数据、设计细节、实现细节、派生类、实例化规则等。

简而言之，就是封装对象的一切“实现机制”，而只表现出对象的“意图”。

图 1-7 所示的例子中，封装包含了以下三个方面的内容。

- 数据封装。点、线、方、圆对象中所有的数据对其他对象是隐藏的。
- 方法封装。例如，圆对象中的 SetLocation 方法。
- 类型封装。这是最重要的，在“设计模式”中，这就是通常的“封装”的含义。在上面的例子中，除了“圆”以外，其他对象都不知道“椭圆”的存在；使用“形状”的客户不知道“点、线、方、圆”对象的存在。

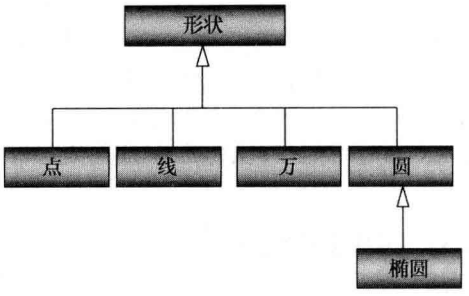

图 1-7 对象封装的例子

下面是另一个例子，有如下两个程序。

（1）我去邮局寄包裹给张三。

（2）我去邮局，找了一个邮递员，委托他送包裹。要他先坐车去火车站，买火车票，搭今天晚上 9:27 的 T110 次火车，坐 36 个站，明天下午 3:13 到达县城，在县城搭 38 路汽车坐 11 个站到镇上的汽车站下车，然后往西北方向步行 371 步在十字路口左转，再步行 167 步在三叉路口右转，再……找一个叫张三的人，把包裹交给他，然后原路回来，最后向我汇报结果：0 表示成功，1 表示没找到人，2 表示对方拒收，3 表示……

第一个程序是面向对象的；第二个程序是面向过程的。第一个程序委托有责任的实体，无知、懒惰，却很幸福；第二个程序事事亲历亲为，最后可能落得劳累过度、吐血身亡。第一个程序是自然语言的表达，是真实生活中的自然场景；第二个程序是机制化的过程，是荒诞世界里的黑色幽默。第一个程序可以应对变化，比如火车、汽车班次的改变；第二个程序遇到变化就会死机。

让这一切截然不同的，就是对象的封装。

对象如果像人类，那么他应该是这样的一种人。

- 无知：只了解自己，不了解他人，让他们无知地幸福着吧！
- 自私：个人自扫门前雪，休管他人瓦上霜，永远不要热心帮助别人。
- 懒惰：一个人就只干一件工作（一个职责、一种变化），要推卸责任（不该自己负责的），休想让我身兼两职。
- 孤僻：独立地做自己的事，尽可能地少联络、少依赖其他人。
- 内向：不想要别人知道的东西，绝不让别人知道，包括自己的父母和儿女。

在构成实现上，对象应同时包含数据和行为，数据好比是躯干、肢体、血肉、器官，而方法好比是灵魂、思想、行为、言语。没有方法的纯数据类是“行尸走肉”，因为纯数据类一定是任

其他对象蹂躏、糟踏的对象，自己没有任何主见，也不可能对自己的数据进行封装；而没有数据的纯方法类是“孤魂野鬼”，因为它们只能去操作别人的数据，或者等着别人把数据送上门；如果对象的行为和自己的数据没有任何关系，则是“鬼魂附体”，借用别人的身体表现自己的行为。在设计中应该尽量避免出现这样的对象。既有数据又有方法，才能成为身体健全、人格完整、行为独立、有责任、有隐私的健康的人（对象）。

1.3　如何分解对象？

面对复杂的领域，如何将其分解为对象呢？

传统方法从问题域中寻找名词，并创建对象来表示它们；然后找到与这些名词相关的动词，并在对象中添加方法来实现这些动词。这通常会导致得到比预期更大的类层次结构。

应该尝试将问题域分解为责任，然后定义必需的对象来实现这些责任，让它对自己的行为负责。

从另一个角度理解，“责任”就是对象的意图，其实也就是“变化”。

按“变化”进行领域分解和设计。

发现变化，并且封装它！正如《笑傲江湖》中的独孤九剑那样，哪里有破绽，剑尖就指向哪里；哪里有变化，封装和模式就应用在哪里！

千万注意，不要让一个类封装两个要变化的事物，除非这些变化明确地耦合在一起。否则，会降低内聚性，变化之间的耦合也无法松散。

如何发现变化的地方呢？可以使用很多方法，例如，if/switch 语句往往预示着变化。作为通用的办法，可以使用共性和可变性分析方法（CVA）。CVA 的目的是寻找变化，并用高内聚、松耦合的类封装变化。其原则是每个共性一个问题。否则设计中就不能有较强的内聚。

所谓共性分析，就是寻找一些共同的要素，它们能帮助我们理解系列成员的共同之处在哪里，找到不可能随时间而改变的结构，为架构提供长效的要素。

所谓可变性分析，就是揭示系列成员之间的不同，要找到可能变化的结构，促进架构适应实际使用的需要。变化是相对不变而言的，可变性是相对共性而言的，可变性只有在给定了共性之后才有意义。

表 1-1 所示的是一个国际电子商务案例的 CVA 分析矩阵。在行中体现共性（概念、抽象），在列中体现变化（在不同国家的具体实现）。

表 1-1 电子商务案例的 CVA 分析矩阵

	美国	加拿大	英国
运费计算规则			
邮编验证规则			
税费计算规则			
货币			
日期格式			
最大质量限制			

CVA 告诉我们如何明确系统中的变化，然后找到应该在设计中使用什么模式。在某种情况下，找到最重要的特性，用矩阵组织它们，用特性所表示的概念为每个特性标记；继续处理其他情况，按需要扩展矩阵；处理每一情况时应独立于其他情况；用新的概念扩展该分析矩阵：用行发现规则，用列发现特定情况；从分析中确定模式，得到高层设计。

CVA 在问题越大、特殊情况越多、人脑越无法得到总体视图时越有用，而且经常会用到子矩阵。

另外，在确定变化点的时候，要保证隔离，确保不同的变化之间划清界限，绝不要拖泥带水。

图 1-8 所示的是对前文中所举的动物类的继承和派生设计进行改进的例子。

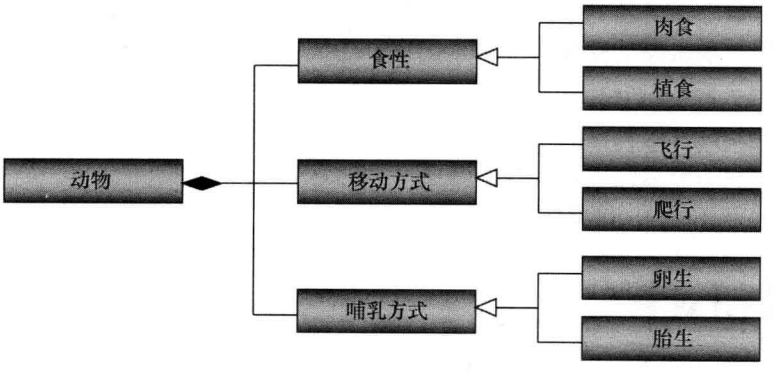

图 1-8 按“变化”进行重新设计的动物类

在改进的设计中，飞行的动物也要求能在地上行走了，让“移动方式”去处理就好了（同时支持飞行与爬行）；更多的需求变化（如新增“哺乳方式”的划分）的时候，只需要增加一个或几个类，而不会导致类爆炸。