第8章 设计原则

8.1 概述

刘其成

计算机与控制工程学院

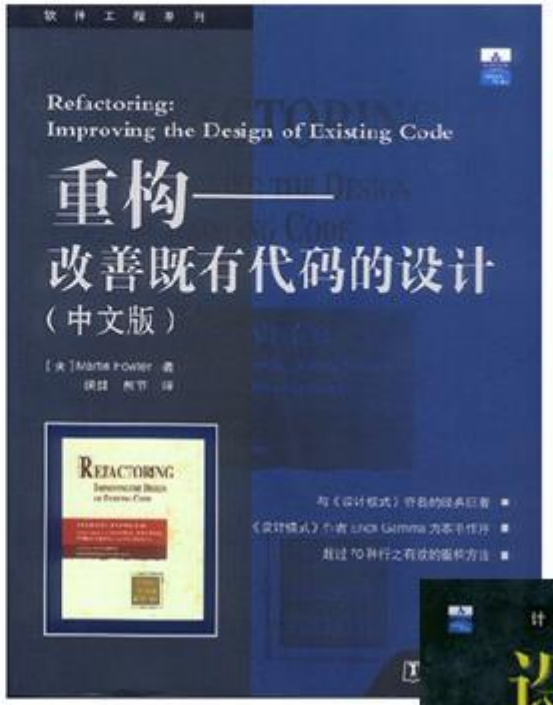
ytliuqc@163.com

2018-09

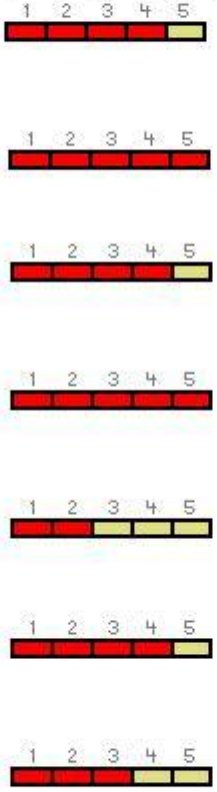
主要内容

- 学习目标
  - 理解面向对象设计原则的基本概念
  - 掌握主要设计原则的概念、实现方法和实例
- 可维护性和可复用性是两个独立的目标；可维护性复用是指在支持可维护性的同时，提高系统的可复用性。
- 在面向对象的设计里，可维护性复用是以设计原则和设计模式为基础的。本章讲述主要的面向对象设计原则，包括开-闭原则、里氏代换原则、合成/聚合原则、依赖倒转原则、迪米特法则、接口隔离原则和单一职责原则。

本章教学内容



- 面向对象设计原则概述
- 单一职责原则
- 开闭原则
- 里氏代换原则
- 依赖倒转原则
- 接口隔离原则
- 合成复用原则
- 迪米特法则



- 通常认为，一个可维护性较好的系统，就是复用率较高的系统；而一个可复用性好的系统，就是一个可维护性好的系统。
- 但是实际上，可维护性和可复用性是两个独立的目标。
- 对于面向对象的软件系统设计来说，在支持可维护性（Maintainability）的同时，提高系统的可复用性（Reuseability）是一个核心的问题。

8.1.1 软件系统的可维护性

1. 软件的维护

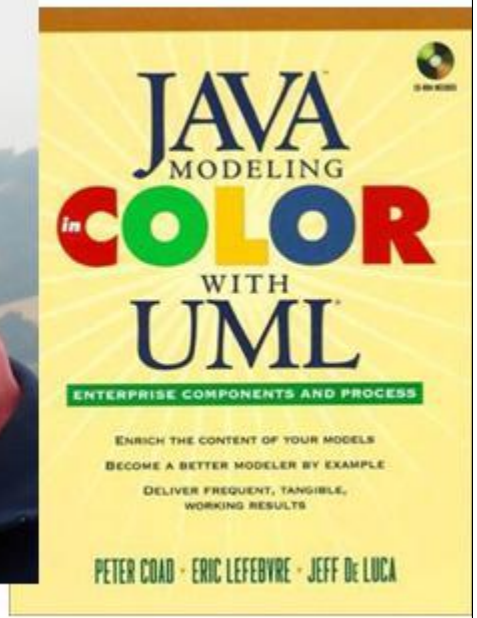
- 像电视机这样的家用电器，常常是买下后一用就是好几年，几乎不需要维修。而且，不可能把一个黑白电视机改成彩色电视，或者将一个小电视改成为一个大电视。
- 一个软件的维护则不同，有的软件开发只需要半年，维护则可能需要很多年。它不仅包括清除错误和缺陷，而且还要包括对已有性能的扩充，以满足新的设计要求。

- 一个好的软件设计，一个可维护性较好的系统，必须能够允许新的设计要求容易地加入到已有的系统中去。
- 因为用户的要求经常变化，如果一个系统的设计不能预测系统的性能要求会发生什么样的变化，这就导致一个系统的设计无法与新的性能要求相容，造成系统设计无法跟上变化。
- 即便新的性能可以添加到系统中去，也不得不以某种破坏原始设计意图和设计框架的方式加入进去。

2. 具有可维护性的设计的目标

- 软件系统的性能要求都是会变化的，系统的设计应该为日后的变化留出足够的空间。
- 软件工程和建模大师Peter Coad认为，一个好的系统设计应该有如下的性质
  - 可扩展性
  - 灵活性
  - 可插入性

Peter Coad



（1）可扩展性

- 新的性能可以很容易地加入到系统中去。要加入一个新性能，在增加一个独立的新模块的同时，不会影响很多其他模块，不会造成几个模块的改动。
- 比如一个新的制动防滑系统应该可以在不影响汽车其他部分的情况下加入到系统中。如果加入了这个防滑系统之后，汽车的方向盘因此出问题，那么这个系统就不是扩展性好的系统。

（2）灵活性

- 代码修改不会波及到很多其他的模块。
- 比如一辆汽车的空调发生了故障，技师把空调修好之后，系统的发动机不能启动了，这就不是一个很灵活的系统。

（3）可插入性

- 可以很容易地将一个类用另一个有同样接口的类代替。
- 一段代码、函数、模块可以在新的模块、或者新系统中使用，这些已有的代码不会依赖于一大堆其他的东西。
  - 比如应该可以很容易地将一辆汽车的防撞气囊取出来，换上一个新的。
  - 如果将气囊拿出来后，汽车的传动杆不工作了，那么这个系统就不是一个可插入性很好的系统。

- 可能会觉得，这些汽车的比喻都很荒唐：
  - 加入防滑系统怎么会影响到汽车的方向盘呢？
  - 修理了空调，怎么会使得发动机不能启动呢？
  - 换一个气囊，怎么会使传动杆停止工作呢？
- 可是如果读者想一想自己所经历的软件系统的设计，就不会觉得荒唐了，因为软件系统中的问题比这些荒唐的比喻还要荒唐。

8.1.2系统的可复用性

软件的复用(Reuse)或重用拥有众多优点

- 提高生产效率（开发效率），节约开发成本。
  - 软件成分的重复使用可以为将来的使用节省费用。
  - 一个组件被复用的频率越高，组件的初始开发投资就相对越少。
- 提高软件质量。
  - 可复用的软件成分比不能复用的软件成分有更多的质量保证。
- 恰当的复用可改善系统的可维护性。
  - 如果一个复用率高的软件组件有程序缺陷的话，这种缺陷可以快速、彻底地被排除。

1. 传统的复用

- （1）代码的剪贴复用
  - 虽然代码的剪贴复用比完全没有复用好一些，但是代码的剪贴复用在具体实施时，要冒着产生错误的风险。
    - 管理人员不可能跟踪大块代码的变异和使用。
    - 由于同样或类似的源代码同时被复制到多个软件成分中，当这块软代码发生程序错误需要修改时，程序员需要独立地修改每一块拷贝。
    - 同样，多个软件成分需要独立地检测。
  - 复用所能节省的初期投资十分有限。

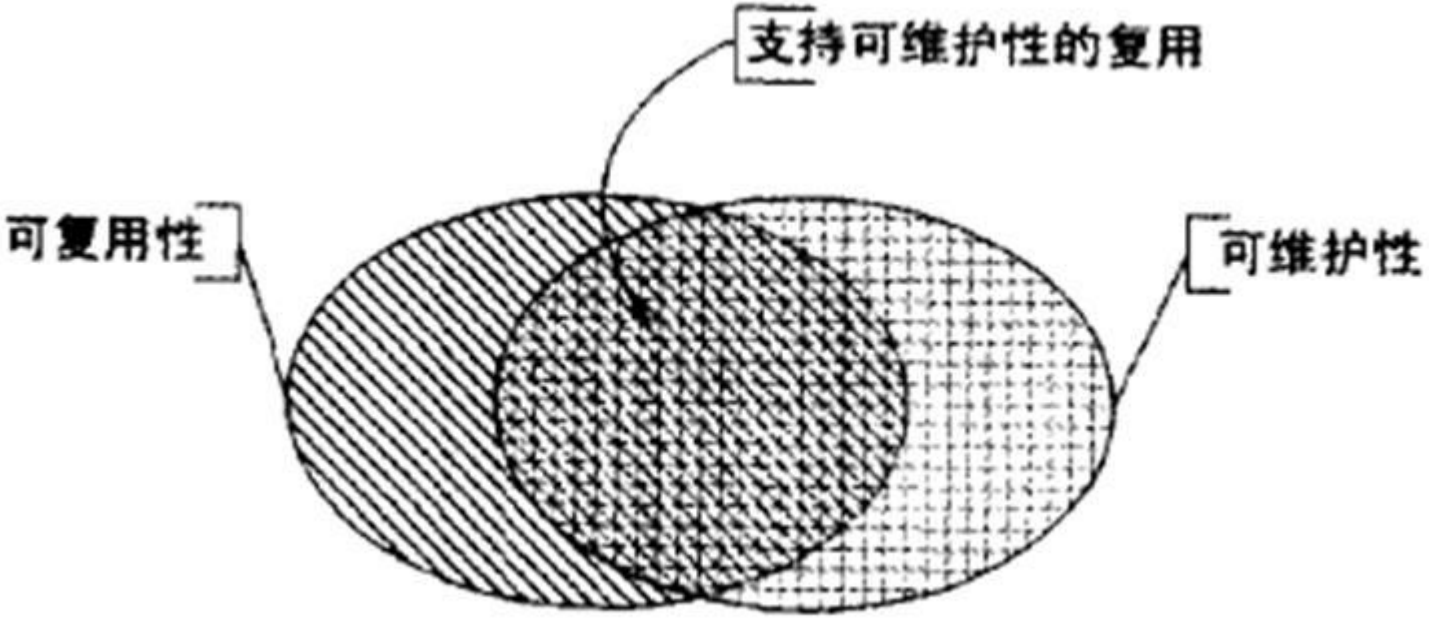
- （2）算法的复用：
  - 各种算法比如排序算法得到了大量的研究。应用程序编程时通常不会建立自己的排序算法，而是在得到了很好的研究的各种算法中选择一个。
- （3）数据结构的复用：
  - 队列、列表等数据结构得到了十分透彻的研究，可以方便地拿过来使用。

可维护性与复用的关系

- 传统复用的缺陷就是复用常常是以破坏可维护性为代价的。
  - 比如两个模块A和B同时使用另一个模块C中的功能。那么当A需要C增加一个新的行为的时候，B有可能不需要、甚至不允许C增加这个新行为。
- 如果坚持使用复用，就不得不以系统的可维护性为代价；而如果从保持系统的可维护性出发，就只好放弃复用。·

可维护性与复用的关系

- 可维护性与可复用性是有共同性的两个独立特性
- 它们就像是两只同时在奔跑的兔子
- 如图所示



- 因此，重要的是支持可维护性的复用，也就是在保持甚至提高系统的可维护性的同时，实现系统的复用。
- 那么怎样才能设计一个系统，以达到提高可维护性的复用目的呢？

2. 面向对象设计的复用

- 面向对象的语言（例如Java语言）中，数据的抽象化、继承、封装和多态性等语言特性使得一个系统可在更高的层次上提供可复用性。
  - 数据的抽象化和继承关系使得概念和定义可以复用
  - 多态性使得实现和应用可以复用
  - 而抽象化和封装可以保持和促进系统的可维护性

- 复用的焦点
  - 不再集中在函数和算法等具体实现细节上
  - 而是集中在最重要的含有宏观商业逻辑的抽象层次上。
- 抽象层次的复用是在提高复用性的同时保持和提高可维护性的关键。

- 抽象是提高复用性同时保持和提高可维护性的关键
  - 这并不是说实现细节的复用不再重要，而是因为这些细节上的复用往往已经做得很好
  - 抽象层次是比这些细节更值得强调的复用
- 抽象层次的模块的复用较为容易
  - 如果抽象层次的模块相对独立于具体层次的模块的话，具体层次内部的变化就不会影响到抽象层次的结构

8.1.3可维护性复用、设计原则和设计模式

- 软件的可维护性和可复用性
  - 面向对象设计复用的目标在于实现支持可维护性的复用。
  - 在面向对象的设计里面，可维护性复用都是以面向对象设计原则(以及设计模式)为基础的
    - 这些设计原则首先都是复用的原则
    - 遵循这些设计原则可以有效地提高系统的复用性，同时提高系统的可维护性。

面向对象设计原则概述

- 设计原则
  - 在提高一个系统的可维护性的同时，提高这个系统的可复用性的指导原则。
  - 依照这些设计原则进行系统设计，就可以实现可维护性复用——就可以抓到这两只同时在奔跑的兔子。

面向对象设计原则概述

- 设计原则
  - “开-闭”原则
  - 里氏代换原则
  - 依赖倒转原则
  - 组合聚合复用原则
  - 单一职责原则
  - 迪米特法则
  - 接口隔离原则
  - ……

面向对象设计原则概述

- 面向对象设计原则简介
  - 常用的面向对象设计原则包括7个，这些原则并不是孤立存在的，它们相互依赖，相互补充。

设计原则名称	设计原则简介	重要性
开闭原则 OCP (Open-Closed Principle)	软件实体对扩展是开放的，但对修改是关闭的，即在不修改一个软件实体的基础上去扩展其功能	★★★★★
依赖倒转原则 DIP (Dependency Inversion Principle)	要针对抽象层编程，而不要针对具体类编程	★★★★★
里氏代换原则 LSP (Liskov Substitution Principle)	在软件系统中，一个可以接受基类对象的地方必然可以接受一个子类对象	★★★★☆
合成复用原则 CRP (Composite Reuse Principle)	在系统中应该尽量多使用组合和聚合关联关系，尽量少使用甚至不使用继承关系	★★★★☆
单一职责原则 SRP (Single Responsibility Principle)	类的职责要单一，不能将太多的职责放在一个类中	★★★★☆
迪米特法则 LoD (Law of Demeter)	一个软件实体对其他实体的引用越少越好，或者说如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用，而是通过引入一个第三者发生间接交互	★★★☆☆
接口隔离原则 ISP (Interface Segregation Principle)	使用多个专门的接口来取代一个统一的接口	★★☆☆☆

- “开-闭”原则、里氏代换原则、依赖倒转原则和组合/聚合复用原则可以在提高系统的可复用性同时，提高系统的可扩展性。允许一个具有同样接口的新的类代替旧的类，是对抽象接口的复用。客户端不依赖于一个具体实现类，而是一个抽象的接口，那么这个具体类可以被另一个具体类所取代时，不会影响到客户端。
- “开-闭”原则、迪米特法则、接口隔离原则可以在提高系统的可复用性同时，提高系统的灵活性。如果系统中每一个模块都相对于其他模块独立存在，并只保持与其他模块的尽可能少的通信，那么，其中某一个模块发生代码修改的时候，不会影响到其他的模块。
- “开-闭”原则、里氏代换原则、组合/聚合复用原则以及依赖倒转原则可以在提高系统的可复用性同时，提高系统的可插入性。例如，在一个符合“开-闭”原则的系统中，抽象层封装了与商业逻辑有关的重要行为，而具体实现由实现层给出。当一个实现类不满足需要需要用另一个实现类代替时，旧的类可以很方便地去掉，而由新的类替换。

- 设计模式三大类别，主要有23个。
  - 创建模式
  - 结构模式
  - 行为模式
- 设计模式本身并不能保证一个系统的可复用性和可维护性，但是设计师运用设计模式的思想设计系统可以提高系统设计的复用性和可维护性。
- 设计模式的思想有助于提高设计师的设计风格、设计水平，并促进同行之间的沟通。

国学与软件设计理论

- 老子论“不武”
- 《老子》云：“善为士者不武”。
  - “士”就是软件系统设计师，“武”就是对软件系统的大规模修改
  - 好的设计师不会在他设计的系统投入使用后再进行大规模的修改。
  - “不武”，便是软件设计中的“复用”。

国学与软件设计理论

- 《老子》还说：“天下有道，却走马以粪，天下无道，戎马生于效。”
  - 当天下有道（太平）时，好的跑马却在田间耕作；当天下无道（不太平）时，战马在战场上生出小驹。
- 这相当于说，当一个软件系统是一个复用有道、易于维护的系统时，将新的性能加入到系统中去，或者对一个已有的性能进行修改是不困难的事情，因此，代码高手就无法发挥作用；
- 而当一个软件系统是一个设计低劣、可维护性很差的系统时，代码高手就必须连续作战，才能将新的性能加入到系统中去，或者对一个已有的性能进行修改。

国学与软件设计理论

- 中国古代的思想大师对很多问题的哲学论述，并没有随着时间的转移而失去其影响力。
- 这些论述的精辟和洞察力，都使得它们可以应用到软件设计理论中去，从而发挥意想不到的威力。

谢谢

2018年11月6日