第8章 设计原则

8.2 开闭原则

刘其成

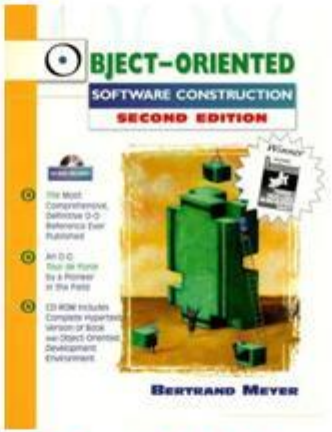
计算机与控制工程学院

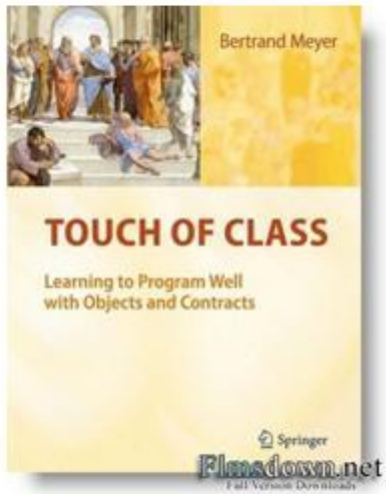
ytliuqc@163.com

2018-09

8.2.1 概念

- 开-闭原则（Open-Closed Principle，OCP）由Bertrand Meyer于1988年提出。





- 定义
  - 指软件应该对扩展开放，对修改关闭。
  - 在设计一个模块的时候，应当使这个模块可以在不被修改源代码的前提下被扩展——改变这个模块的行为。

- 满足“开-闭”原则的软件系统
  - 通过扩展已有的软件系统，可以提供新的行为，以满足对软件的新需求，使变化中的软件系统有一定的适应性和灵活性。
  - 同时，已有的软件模块，特别是最重要的抽象层模块不能再修改，这就使变化中的软件系统有一定的稳定性和延续性。
- 这样的软件系统是一个在高层次上实现了复用的系统，也是一个易于维护的系统。

8.2.2 实现方法

- 抽象化
  - 开闭原则的关键。
- 对可变性封装原则
  - 开闭原则还可以通过一个更加具体的“对可变性封装原则”来描述。

抽象化

- 运用面向对象技术，定义不再更改的抽象层，但允许有无穷无尽的行为在实现层被实现。
  - 面向对象语言中抽象数据类型（例如，Java语言的抽象类或接口），可以规定出所有的具体类必须提供的方法的特征作为系统设计的抽象层；
    - 这个抽象层预见了所有的可能扩展，在任何扩展情况下都不改变；
    - 系统的抽象层不修改，满足了开-闭原则的中对修改关闭的要求。
- 同时，由于从抽象层导出一个或多个新的具体类可以改变系统的行为，因此系统的设计对扩展是开放的，这就满足了开-闭原则对扩展开放的要求。

对可变性封装原则

- 对可变性封装原则(Principle of Encapsulation of Variation, EVP)
  - 找到系统的可变因素并将其封装起来。
- 考虑设计中什么可能会发生变化。注意：考虑的问题
  - 不是什么会导致设计改变
  - 而是允许什么发生变化，而不让这一变化导致重新设计

- 尽管在很多情况下，无法百分之百做到“开-闭”原则
- 但是即使是部分的满足，也可以显著地改善一个系统的结构。

8.2.3 与其他设计原则的关系

- 其他设计原则都是开-闭原则的手段和工具，是附属于开-闭原则的。

（1）里氏代换原则

- 任何父类可以出现的地方，子类一定可以出现。
- 里氏代换原则是对开-闭原则的补充，是对实现抽象化的具体步骤的规范。
- 实现开-闭原则的关键是创建抽象化，并且从抽象化导出具体化。
- 而从抽象化到具体化的导出要使用继承关系和里氏代换原则。
- 一般而言，违反里氏代换原则的，也违背开-闭原则，反过来并不一定成立。

（2）依赖倒转原则

- 要依赖于抽象，不要依赖于实现。
- 依赖倒转原则与开-闭原则之间是目标和手段之间的关系：
  - 要想实现“开-闭”原则这个目标，就应当坚持依赖倒转原则。
  - 违反依赖倒转原则，就不可能达到开-闭原则的要求。

（3）合成/聚合复用原则

- 要尽量使用合成/聚合实现复用，而不是继承。
- 遵守合成/聚合复用原则是实现开-闭原则的必要条件。

（4）接口隔离原则

- 应当为客户端提供尽可能小的单独的接口，而不要提供大的总接口。
- 这是对一个软件实体与其他的软件实体的通信的限制。
- 遵循接口隔离原则，会使一个软件系统在功能扩展的过程当中，对一个对象的修改不会影响到其他的对象。

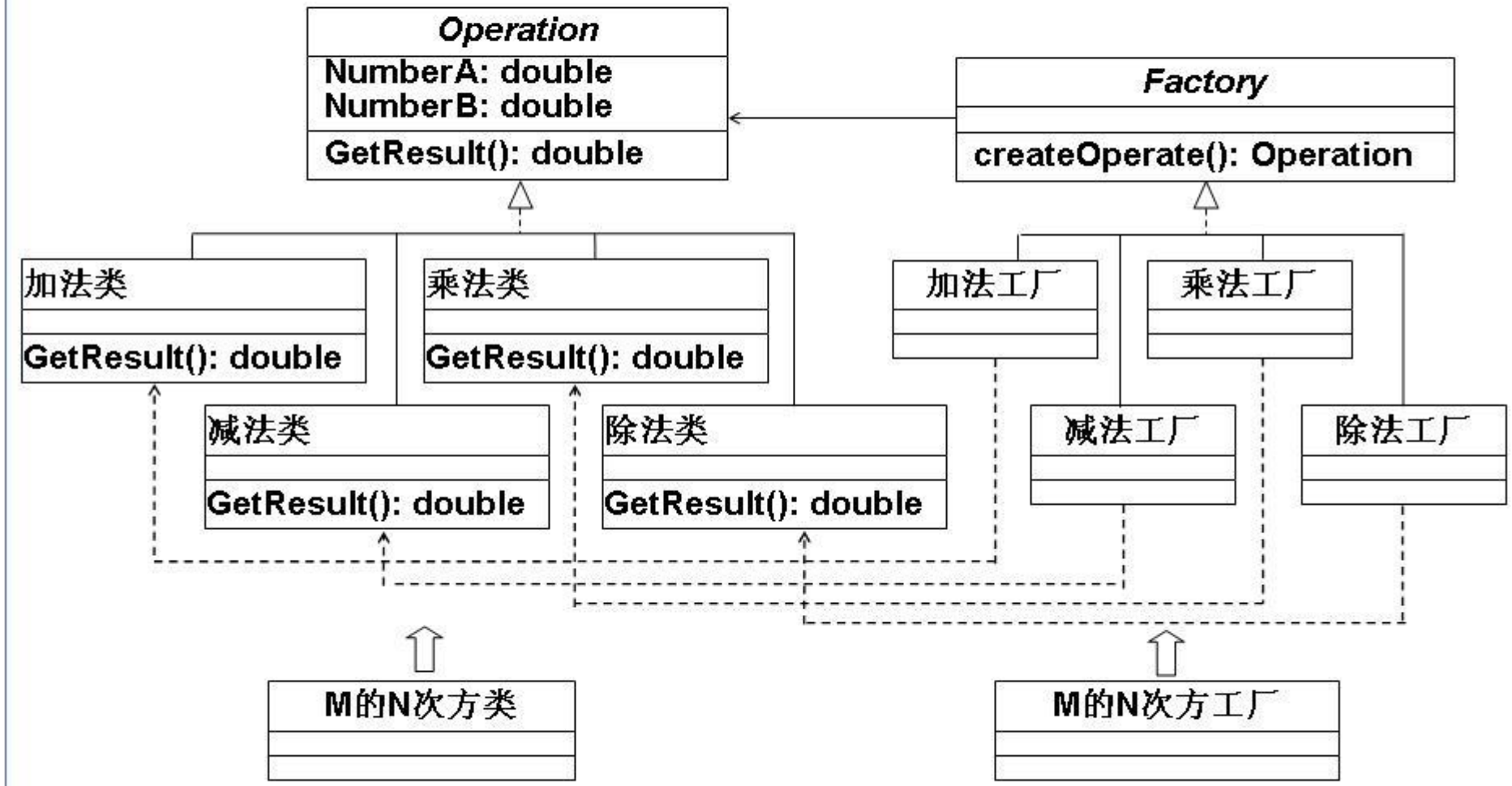
（5）迪米特法则

- 一个软件实体应当与尽可能少的其他实体发生相互作用。
- 当一个系统功能扩展时，模块如果是孤立的，就不会影响到其他模块。
- 遵守迪米特原则的系统在功能需要扩展时，会相对更容易地做到对修改的关闭。

8.2.4 实例

- 绝对的对修改关闭是不可能的，设计人员必须对于他设计的模块应该对哪种变化封闭做出选择。必须先猜测出最有可能发生的变化种类，然后构造抽象来隔离那些变化，有时会把本该简单的设计做得非常复杂。
- 但可以在发生小变化时，就及早去想办法应对发生更大变化的可能。在最初编写代码时，假设变化不会发生。当变化发生时，就创建抽象来隔离以后发生的同类变化。

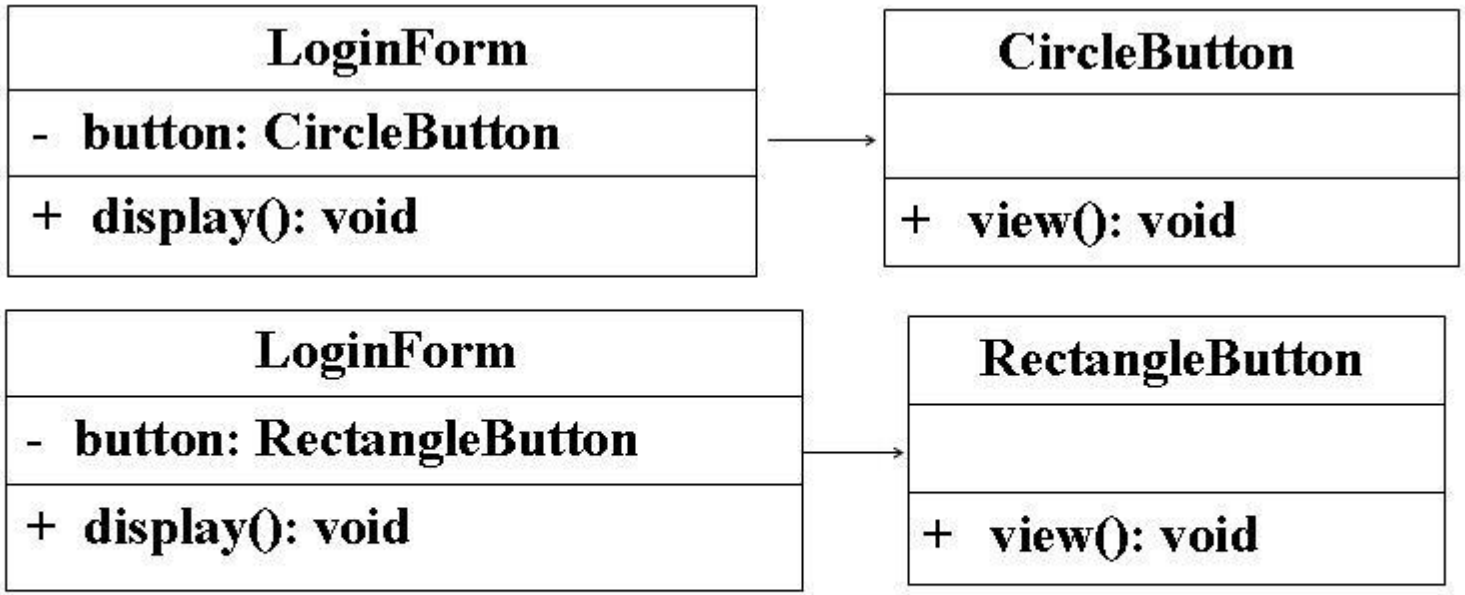
- 比如加法程序，很快在一个client类中就完成，此时变化还没有发生。然
- 后加一个减法功能，会发现增加功能需要修改原来这个类，这就违背了开-封原则，于是就该考虑重构程序，增加一个抽象的运算类，通过一些面向对象的手段，如继承，多态等来隔离具体加法、减法与client耦合，需求依然可以满足，还能应对变化。
- 这时又要再加乘除法功能，就不需要再去更改client以及加法减法的类了，而是增加乘法和除法子类就可。
- 即面对需求，对程序的改动是通过增加新代码进行的，而不是更改现有的代码。
- 这就是开-封原则。


Operation
NumberA: double
NumberB: double
GetResult(): double
Factory
createOperate(): Operation
加法类
GetResult(): double
乘法类
GetResult(): double
减法类
GetResult(): double
除法类
GetResult(): double
加法工厂
乘法工厂
减法工厂
除法工厂
M的N次方类
M的N次方工厂

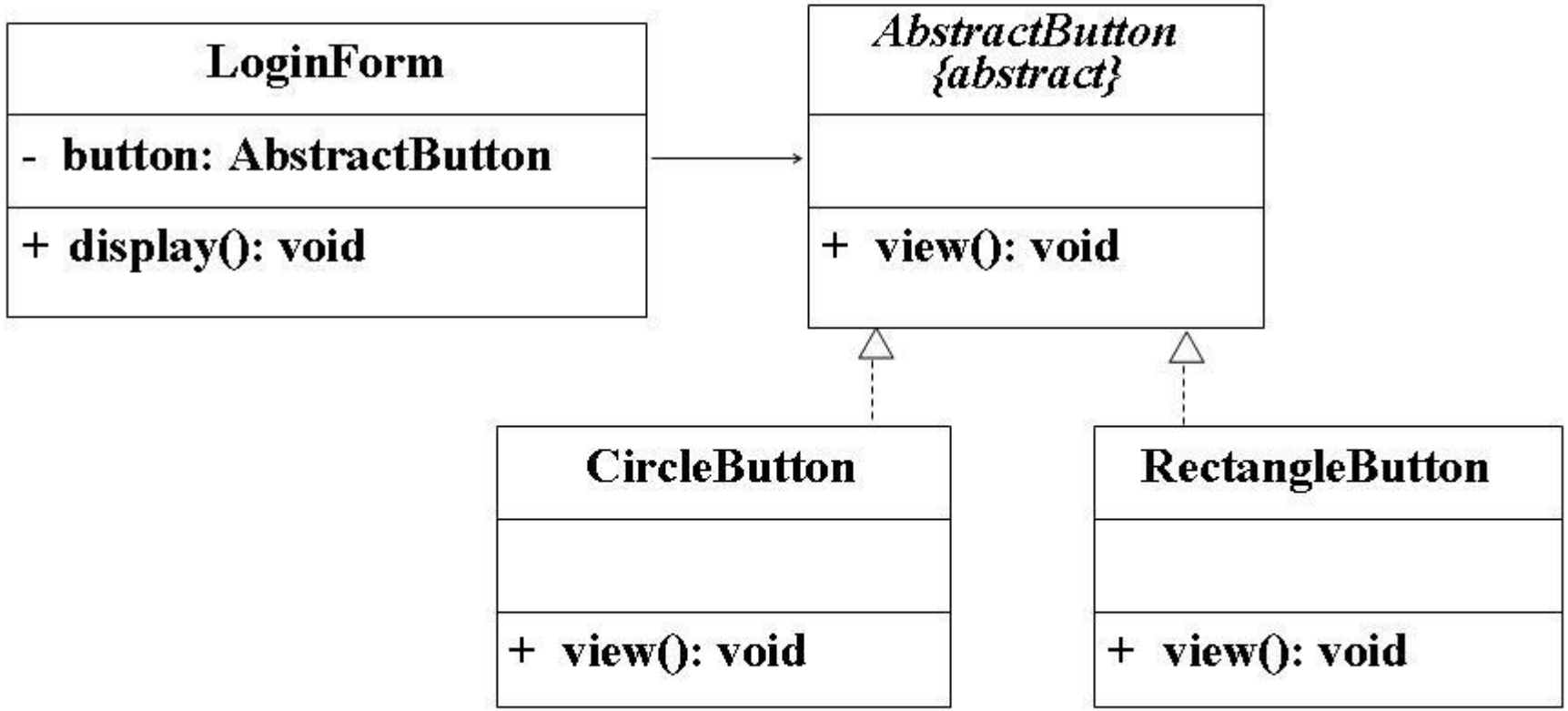
- 开发工作展开不久就要知道可能发生的变化。否则如果加法运算都在很多地方用了，再考虑抽象、分离，就很困难。
- 同时，也不需要对应用程序中的每个部分都刻意地进行抽象，拒绝不成熟的抽象和抽象本身一样重要。

实例

- 某图形界面系统提供了各种不同形状的按钮，客户端代码可针对这些按钮进行编程，用户可能会改变需求要求使用不同的按钮，原始设计方案如图8-1所示。

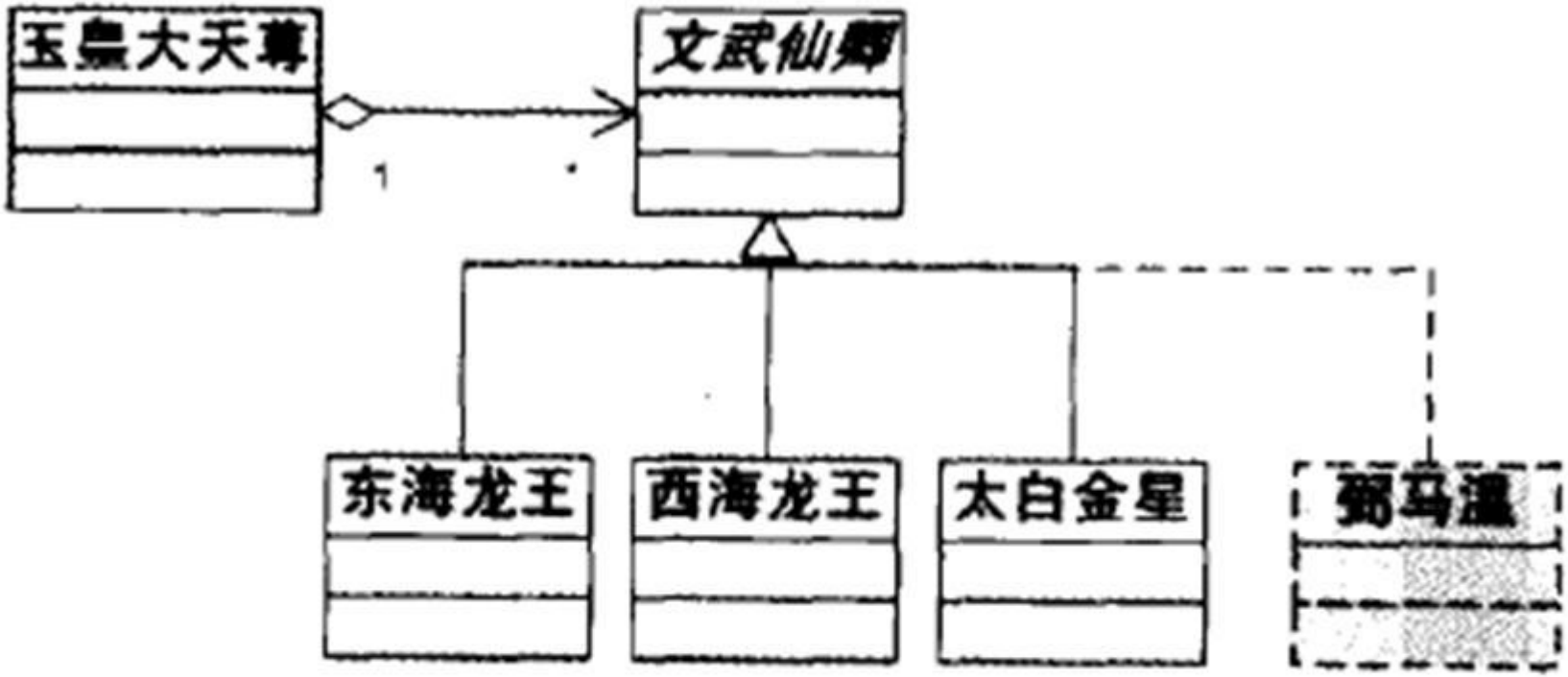


- 现对该系统进行重构，使之满足开闭原则的要求，如图8-2所示。



玉帝招安美猴王

- 当年大闹天宫时的美猴王便是玉帝天庭的新挑战。美猴王说：“‘皇帝轮流做，明年到我家。’只教他搬出去，将天宫让与我!”对于这项挑战，太白金星给玉皇大帝提出的建议是：“降一道招安圣旨，把他宣来上界……与他籍名在篆……一则不动众劳师，二则收仙有道也。”
- 不劳师动众、不破坏天规便是“闭”，收仙有道便是“开”。招安之法便是玉帝天庭的“开-闭”原则，通过给美猴王封一个“弼马温”的官职，便可使现有系统满足变化了的需求，而不必更改天庭的既有秩序，如下图所示。
- 招安之法的关键便是不允许更改现有的天庭秩序，但允许将妖猴纳入现有秩序中，从而扩展了这一秩序。用面向对象的语言来讲，不允许更改的是系统的抽象层，而允许扩展的是系统的实现层。


玉皇大天尊
文武仙卿
1
*
东海龙王
西海龙王
太白金星
哪吒

《太玄》论“固革”

- 西汉杨雄的《太玄》一书说：“知固而不知革，物失其则；知革而不知固，物失其均。”
- 一个系统对修改关闭，就是《太玄》所说的“固”；而系统对扩展开放，就是《太玄》所说的“革”。
- 一个系统不可扩展，就会“物失其则”，或者说系统失去使用的价值；而一个系统动辄需要修改，便会“物失其均”，也就是失去重心。
- 因此，“开-闭”原则非常接近《太玄》一书所说的“固革”原则。

上班迟到1

- 设计软件要容易维护又不容易出问题的最好的办法，就是多扩展，少修改。
- 比如说，我是公司老板，我规定，九点上班，不允许迟到。但有几个公司骨干，老是迟到。如果你是老板你怎么做？这不能简单地严格执行考勤制度，迟到扣钱。实际情况是。有的员工家离公司太远，有的员工每天上午要送小孩子上学，交通一堵就不得不迟到了。但如果让他们有特殊原因的人打报告，然后允许他们迟到。别的不迟到的员工就不答应了，凭什么他能迟到，我就不能，大家都是工作，我上午也完全可以多睡会再来。

上班迟到2

- 那怎么办？老是迟到的确也不好，但不让迟到也不现实。家的远近，交通是否堵塞也不是可以控制的。仔细想想会发现，其实迟到不是主要问题，每天保证8小时的工作量是老板最需要的。甚至8小时工作时间也不是主要问题，业绩目标的完成或超额完成才是最重要的指标，于是应该改变管理方式，比如弹性上班工作制，早到早下班。晚到晚下班，或者每人每月允许三次迟到，迟到者当天下班补时间等等，对市场销售人员可能就更加以业绩为标准，工作时间不固定了——这其实就是对工作时间或业绩成效的修改关闭，而对时间制度扩展的开放。
- 这就需要老板自己很清楚最希望达到的目的是什么，制定的制度才最合理有效。用我们古人的理论来说，管理需要中庸之道。

一国两制

- 大陆的社会主义制度不能修改，而香港澳门长期在资本主义制度下管理和发展，所以回归时强行修改香港澳门的制度也并不合理，所以用“一国两制”来解决制度差异造成的矛盾是最合理的办法。
- 为了回归的大局，增加一种制度又何尝不可，一个国家，两种制度，这在政治上，是伟大的发明。

考研求职

- 考研和求职这两件事，考研是追求，希望考上研究生，可以更上一层楼。有更大的发展空间和机会。所以考研之前，学习计划是不应该更改，雷打不动的。这就是对修改关闭。但要知道，几个月来只埋头学习，就等于放弃了许多好公司招聘的机会，这机会的失去是很不值得的。不可能一天到晚全在学习，那样效果也不会好。所以完全可以抽出一点时间，在不影响复习的前提下，来写写自己的简历，来了解一些招聘大学生的公司的咨讯，这不是很好的事吗？既不影响考研，又可以增大找到好工作的可能性。为考研万一失败后找工作做好了充分的准备。这就是对扩展开放，对修改关闭的意义。
- 全力以赴当然是必需，两手准备也是灵活处事的表现。对痛苦关闭，对快乐开放。

8.2.5 思考

- 尽量在设计时，考虑到需求的种种变化，把问题想得全了，就不会因为需求一来，手足无措。
- 但如果什么问题都考虑得到，就成了未卜先知，这是不可能的。
- 需求时常会在你想不到的地方出现，让你防不胜防

- 绝对的对修改关闭是不可能的，无论模块是多么的“封闭”，都会存在一些无法对之封闭的变化。既然不可能完全封闭，设计人员必须对于他设计的模块应该对哪种变化封闭做出选择。
- 必须先猜测出最有可能发生的变化种类，然后构造抽象来隔离那些变化。猜对了，那是成功，猜错了，那就完全走到另一面去了，把本该简单的设计做得非常复杂，很不划算呀。很难预先猜测，但可以在发生小变化时，就及早去想办法应对发生更大变化的可能。
- 在最初编写代码时，假设变化不会发生。当变化发生时，就创建抽象来隔离以后发生的同类变化。同一地方，摔第一跤不是你的错，再次在此摔跤就是你的不对了。

- 比如加法程序，很快在一个client类中就完成，此时变化还没有发生。然后加一个减法功能，会发现增加功能需要修改原来这个类，这就违背了开放-封闭原则，于是就该考虑重构程序，增加一个抽象的运算类，通过一些面向对象的手段，如继承，多态等来隔离具体加法、减法与client藕合，需求依然可以满足，还能应对变化。这时又要再加乘除法功能，就不需要再去更改client以及加法减法的类了，而是增加乘法和除法子类就可。
- 如果加减运算都在很多地方应用了，再考虑抽象、考虑分离，就很困难。

谢谢

2018年11月6日