程序员应知道这十大面向对象设计原则

2015-10-29 萌码

面向对象设计原则是OOPS编程的核心， 但大多数Java程序员没有把足够多的注意力放在学习面向对象的分析和设计上面。当然，学习面向对象编程像“抽象”、“封装”、“多态”、“继承” 等基础知识是重要的，但同时为了创建简洁、模块化的设计，了解这些设计原则也同等重要。

所以，还等什么，学习来吧！！！

DRY

我们第一个面向对象设计原则是：DRY ，从名称可以看出DRY(don’t repeat yourself)意思是不写重复代码，而是抽象成可复用的代码块。如果有两处以上相同的代码块，请考虑把它们抽象成一个单独的方法；或者您多次使用了 硬编码的值，请把它们设置成公共常量。这种面向对象设计原则的优点是易于维护。重要的是不要滥用此原则，重复不是针对代码而是针对功能来说。它的意思是， 如果使用通用代码来验证OrderID和SSN，这并不意味着它们是相同的或者他们今后将保持不变。通过把通用代码用于实现两种不同的功能，或者把这两种不同的功能密切地联系在一起；当OrderID格式改变时，SSN验证代码将会中断。所以要当心这种耦合，而且不要把彼此之间没有任何关系却 类似的代码组合在一起。

封装经常修改的代码

在软件领域永远不变的是“变化”，所以把认为或怀疑将来要被修改的代码封装起来。这种面向对象设计模式的优点是：易于测试和维护恰当封装的代 码。 如果在用Java编程，那么请遵守以下原则：变量和方法的访问权限默认设置为私有，并且逐步放开它们的访问权限，例如从“private”到 “protected ”、“not public”。Java中的一些设计模式使用了封装，工厂设计模式就是一个例子，它封装了创建对象的代码而且提供了以下灵活性：后续生成新对象不影响现 有的代码。

打开/关闭设计原则

类、方法/函数应当是对扩展(新功能)开放，对修改闭合。这是另外一个优雅的SOLID 设计原则，以防止有人修改通过测试的代码。理想情况下假如添加了新功能，那么代码要经过测试，这就是打开/关闭设计原则的目标。顺便说一句，SOLID中的字母“O”指的是打开/关闭设计原则

顺便说一句，SOLID中的字母 O 指的是打开/关闭设计原则。

单一职责原则

单一职责原则是另外一个SOLID设计原则，SOLID中的字母“S”指的就是它。按照SRP，一个类修改的原因应当有且只有一个，或者一个类应 当 总是实现单一功能。如果您在Java中的一个类实现了多个功能，那么这些功能之间便产生了耦合关系；如果修改其中的一个功能，有可能就打破了这种耦合 关系，那么就要进行另一轮测试以避免产生新的问题。

依赖注入/反转原则

不要问框架的依赖注入功能将会带来什么益处，依赖注入功能在spring框架里已经很好的得到了实现，这一设计原则的优雅之处在于：DI框架注入的任何一个类都易于用模拟对象进行测试，并且更易于维护，因为创建对象的代码在框架里是集中的而且和客户端代码是隔离的。有多种方法可以实现依赖注入， 例如使用字节码工具，其中一些AOP(面向切面编程)框架如切入点表达式或者spring里使用的代理。想对这种SOLID设计原则了解更多，请看IOC 和 DI设计模式中的例子。 SOLID中的字母“D”指的就是这种设计原则。

优先使用组合而非继承

如果可以的话，要优先使用组合而非继承。组合允许在运行时通过设置属性修改一个类的行为，通过使用多态即以接口的形式实现类之间的组合关系，并且为修改组合关系提供了灵活性。甚至 Effective Java也建议优先使用组合而非继承。

里氏替换原则

根据里氏替换原则，父类出现的地方可以用子类来替换，例如父类的方法或函数被子类对象替换应该没有任何问题。LSP和单一职责原则、接口隔离原则 密 切相关。如果一个父类的功能比其子类还要多，那么它可能不支持这一功能，而且也违反了LSP设计原则。为了遵循 LSP SOLID设计原则，派生类或子类(相对父类比较)必须增强功能，而非减少。SOLID中的字母“L”指的就是 LSP设计原则。

接口隔离原则

接口隔离原则指，如果不需要一个接口的功能，那么就不要实现此接口。这大多在以下情况发生：一个接口包含多种功能，而实现类只需要其中一种功能。 接 口设计是一种棘手的工作，因为一旦发布了接口，您就不能修改它否则会影响实现该接口的类。在Java中这种设计原则的另一个好处是：接口有一个特点，任何 类使用它之前都要实现该接口所有的方法，所以使用功能单一的接口意味着实现更少的方法。

编程以接口(而非实现对象)为中心

编程总是以接口(而非实现对象)为中心，这会使代码的结构灵活，而且任何一个新的接口实现对象都能兼容现有代码结构。所以在Java中，变量、方 法 返回值、方法参数的数据类型请使用接口。这是许多Java程序员的建

议， Effective Java 以及 head first design pattern 等书也这样建议。

代理原则

不要期望一个类完成所有的功能，可以适当地把一些功能交给代理类实现。代理原则的典范是：Java 中的 equals() 和 hashCode() 方法。为了比较两个对象的内容是否相同，我们让用于比较的类本身完成对比工作而非它们的调用方。这种设计原则的好处是：没有重复编码而且很容易修改类的行 为。

当然，以上原则都是一些理论，而最终是否能够帮助你写出灵活、优雅的代码，还是要靠小伙伴们自己去实践，去学习。