走向灵活软件之路———面向对象的六大原则

原创 xyzso1z 最后发布于2019-01-21 23:05:04 阅读数 59 收藏 编辑 展开

> 1.优化代码的第一步——单一职责原则

单一职责原则的英文名称是Single Responsibility Principle,缩写是SRP.SRP的定义是：就要给类而言，应该仅有一个引起它变化的原因。简单来说，一个类中应该是一组相关性很高的函数、数据的封装。

> 2.让程序更稳定、更灵活——开闭原则

开闭原则的英文全称是Open Close Principle,缩写是OCP,它是Java世界里最基础的设计原则，它指导我们如何建立一个稳定的、灵活的系统。开闭原则的定义是：软件中的对象（类、模块、函数等）应该对于扩展是开放的，但是，对于修改是封闭的。在软件的生命周期内，应为变化、升级和维护等原因需要对原有代码进行修改时，可能会将错误引入原本已经经过测试的旧代码中，破坏原有系统。因此，在现实开发中，只通过继承的方式来升级、维护原有系统只是一个理想愿景，因此，在实际的开发过程中，修改原有代码、扩展代码往往是同时存在的。
软件开发过程中，最不会变化的就是变化本身。产品需要需要不断地升级、维护，没有一个产品从第一版开发完就没有变化了，除非在一个版本诞生之气那它已经被终止。而产品需要升级，修改原来的代码就可能会引发其他的问题。那么，如何确保原有软件模块的正确性，以及尽量少地影响原有模块，答案就是，尽量遵守开闭原则。
开闭原则指导我们，当软件需要变化时，应该尽量通过扩展的方式来实现变化，而不是通过修改已有的代码来实现。这里的“应该尽量”4个字说明OCP原则并不是绝对不可以修改原始类的。当我们嗅到原来的代码“腐化气味”时应该尽早重构，以便使代码恢复到正常”进化“过程，而不是通过继承等方式添加新的实现，这会导致类型的膨胀以及历史遗留代码的雍长。我们的开发过程中也没有那么理想化的状况，完全地不用修改原来的代码，因此，在开发过程中需要自己结合具体情况进行考量，是通过修改旧代码还是通过继承使得软件系统更稳定、更灵活，在保证去除”代码腐化“的同时，也保证原有模块的正确性。

> 3.构建扩展性更好的系统——里氏替换原则

里氏替换原则英文全称是Liskov Substitution Principle，缩写是LSP。LSP的第一种定义是：如果每一个类型为S的对象O1，都有类型为 T 的对象O2,使得以T定义的所有程序P在所有的对象O1都代替换成O2时，程序P的行为没有发生变化，那么类型S是类型T的子类型。上面这种描述确实不太好理解，我们再看看另一个直截了当的定义。里氏替换原则第二种定义：所有引用基类的地方必须能透明地使用其子类的对象。
我们知道，面向对象的语言的三大特点是继承、封装、多态，里氏替换原则就是依赖于继承、多态这两大特性。里氏替换原则简单来说就是，所有引用基类的地方必须能透明地使用其子类的对象。通俗点讲，只要父类能出现的地方子类就可以出现，而且替换为子类也不会产生任何错误或异常，使用者可能根本就不需要知道是父类还是子类。但是，反过来就不行了，有子类出现的地方，父类未必就能适应。说了那么多，其实最终总结就是两个字：抽象。
里氏替换原则的核心原理是抽象，抽象又依赖于继承这个性质，在OOP当中，继承优缺点都相当明显。有点有以下几点：

1. 代码重用，减少创建类的成本，每个子类都拥有父类的方法和属性；
2. 子类与父类基本相似，但又与父类有所区别；
3. 提高代码的可扩展性。

继承的缺点：

1. 继承是侵入性的，只要继承就必须拥有父类的所有属性和方法；
2. 可能造成子类代码冗余、灵活性降低，因为子类必须拥有父类的属性和方法。

> 4. 让项目拥有变化的能力——依赖倒置原则

依赖倒置原则英文全称是Dependence Inversion principle,缩写是DIP。依赖倒置原则指代了一种特定的解耦形式，使得高层次的模块不依赖于低层次模块的实现细节的目的，依赖模块被颠倒了。这个概念有点不好理解，这到底是什么意思呢？
依赖倒置原则有以下几个关键点:

1. 高层模块不应该以来低层模块，两者都应该依赖其抽象；
2. 抽象不应该依赖细节；
3. 细节应该依赖抽象 ；

在Java语言中，抽象就是指接口或抽象类，两者都是不能直接被实例化的；细节就是实现类，是实现接口或继承抽象类而产生的类就是细节，其特点就是，可以直接被实例化，也就是可以加上一个关键字new 产生一个对象。高层模块就是调用端，底层模块就是具体实现类。依赖倒置原则在Java语言中的表现就是：**模块间的依赖通过抽象发生，实现类之间不发生直接的依赖关系，其依赖关系是通过接口或抽象产生的**。这又是一个将理论抽象化的实例，其实一句话就可以概括：**面向接口编程，或者说是面向抽象编程，这里的抽象指的是接口或抽象类**。面向接口编程是面向对象精髓之一，也就是抽象。

> 6.更好的可宽展性——迪米特原则

迪米特原则英文名称为Law of Demeter,缩写是LOD，也称为最小只是原则（Least Knowledege Principle)。虽然名称不同，但描述的是同一个原则：一个对象应该对其它对象有最少的了解。通俗地讲，一个类应该对自己需要耦合或调用的类知道的最少，类的内部如何实现与调用者或者依赖者没关系，调用者或者依赖者只需要知道它需要的方法即可，其它的可以一概不管。类与类之间的关系越密切，耦合度越大，当一个类发生改变时，对另一个类的影响也越大。
迪米特法则还有一个英文解释是Only talk to your immedate friends,翻译过来就是：只与直接的朋友通信。什么叫做直接的朋友呢？每个对象都必然会与其他对象有耦合关系，两个对象之间的耦合就成为朋友关系，这种关系的类型有很多，如组合、聚合、依赖等。

> 总结

在应用开发过程中，最难的不是完成应用的开发工作，而是在后续的升级、维护过程中让应用系统能够拥抱变化。拥抱变化也就意味满足需求且不破坏系统稳定性的前提下保持高可扩展性、高内聚、低耦合，在经历了各版本的变更之后依然保持清晰、灵活、稳定的系统架构。当然这是一个比较理想的情况，但我们必须要朝着这个方向去努力，那么遵循面向对象六大原则就是我们走向灵活软件之路所迈出的第一步。

xyzso1z
原创文章 80　获赞 40　访问量 2万+