面向对象的 6 个基本原则1法则

六尺帐篷 (/u/f8e9b1c246f1)
2017.09.05 13:05 字数 2587 阅读 72 评论 0 喜欢 5
(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/5229761/notes/16705552)

- s(Single-Resposibility Principle): 单一职责原则。
- o(Open-Closed principle): 开放封闭原则。
- l(Liskov-Substituion Principle): 里氏替换原则。
- i(Interface-Segregation Principle): 接口隔离原则。
- d(Dependecy-Inversion Principle): 依赖倒置原则。
- 合成/聚合复用。
- 迪米特法则

单一职责原则(Single-Resposibility Principle)

一个类只做它该做的事情。
是指一个类的功能要单一， 一个类只负责一个职责。 一个类只做它该做的事情(高内聚)。 在面向对象中， 如果只让一个类完成它该做的事， 而不涉及与它无关的领域就是践行了高内聚的原则。

单一职责原则想表达的就是"高内聚"，写代码最终极的原则只有六个字"高内聚、低耦合"，就如同葵花宝典或辟邪剑谱的中心思想就八个字"欲练此功必先自宫"，所谓的高内聚就是一个代码模块只完成一项功能，在面向对象中，如果只让一个类完成它该做的事，而不涉及与它无关的领域就是践行了高内聚的原则，这个类就只有单一职责。

一个好的软件系统，它里面的每个功能模块也应该是可以轻易的拿到其他系统中使用的，这样才能实现软件复用的目标。

开放封闭原则(Open-Closed principle)

软件实体应当对扩展开放， 对修改关闭。 对扩展开放， 意味着有新的需求或变化时，可以对现有代码进行扩展， 以适应新的情况。 对修改封闭， 意味着类一旦设计完成，就可以独立其工作， 而不要对类尽任何修改。 在开发阶段， 我们都知道， 如果对一个功能进行扩展， 如果只是一味地对方法进行修改， 可能会造成一些问题， 诸如可能会引入新的 bug， 或者增加代码的复杂度， 对代码结构造成破坏、 冗余， 还需要重新进行全面的测试。 那么该怎么解决这些问题？ 很简单， 这就需要系统能够支持扩展， 只有扩展性良好的系统，才能在不进行修改已有实现代码的基础上， 引进新的功能。

要做到开闭有两个要点：
①抽象是关键，一个系统中如果没有抽象类或接口系统就没有扩展点；
②封装可变性，将系统中的各种可变因素封装到一个继承结构中，如果多个可变因素混杂在一起，系统将变得复杂而换乱

里氏替换原则(Liskov-Substituion Principle)

任何使用基类的地方， 都能够使用子类替换， 而且在替换子类后， 系统能够正常工作。 子类一定是增加父类的能力而不是减少父类的能力，因为子类比父类的能力更多，把能力多的对象当成能力少的对象来用当然没有任何问题。 一个软件实体如果使用的是

一个基类， 那么当把这个基类替换成继承该基类的子类， 程序的行为不会发生任何变化。 软件实体察觉不出基类对象和子类对象的区别。

关于里氏替换原则的描述，简单的说就是能用父类型的地方就一定能使用子类型。里氏替换原则可以检查继承关系是否合理，如果一个继承关系违背了里氏替换原则，那么这个继承关系一定是错误的，需要对代码进行重构。例如让猫继承狗，或者狗继承猫，又或者让正方形继承长方形都是错误的继承关系，因为你很容易找到违反里氏替换原则的场景。需要注意的是：子类一定是增加父类的能力而不是减少父类的能力，因为子类比父类的能力更多，把能力多的对象当成能力少的对象来用当然没有任何问题。

接口隔离原则(Interface-Segregation Principle)

即应该将接口粒度最小化， 将功能划分到每一个不能再分的子角色， 为每一个子角色创建接口， 通过这样， 才不会让接口的实现类实现一些不必要的功能。 建立单一的接口， 不要建立臃肿的庞大的接口， 也就是说接口的方法尽量少。 接口要小而专， 绝不能大而全。 臃肿的接口是对接口的污染，既然接口表示能力， 那么一个接口只应该描述一种能力， 接口也应该是高度内聚的。

例如，琴棋书画就应该分别设计为四个接口，而不应设计成一个接口中的四个方法，因为如果设计成一个接口中的四个方法，那么这个接口很难用，毕竟琴棋书画四样都精通的人还是少数，而如果设计成四个接口，会几项就实现几个接口，这样的话每个接口被复用的可能性是很高的。Java中的接口代表能力、代表约定、代表角色，能否正确的使用接口一定是编程水平高低的重要标识。

依赖倒置原则(Dependecy-Inversion Principle)

即我们的类要依赖于抽象， 而不是依赖于具体， 也就是我们经常听到的“要面向接口编程” 。 （该原则说得具体一些就是声明方法的参数类型、 方法的返回类型、 变量的引用类型时， 尽可能使用抽象类型而不用具体类型， 因为抽象类型可以被它的任何一个子类型所替代） 。 依赖倒置原则的本质就是通过抽象（抽象类或接口） 使各个类或模块的实现彼此独立， 不相互影响，实现模块间的松耦合。 减少类间的耦合性。

合成聚合复用原则

优先使用聚合或合成关系复用代码。

通过继承来复用代码是面向对象程序设计中被滥用得最多的东西，因为所有的教科书都无一例外的对继承进行了鼓吹从而误导了初学者，类与类之间简单的说有三种关系，Is-A关系、Has-A关系、Use-A关系，分别代表继承、关联和依赖。其中，关联关系根据其关联的强度又可以进一步划分为关联、聚合和合成，但说白了都是Has-A关系，合成聚合复用原则想表达的是优先考虑Has-A关系而不是Is-A关系复用代码，原因嘛可以自己从百度上找到一万个理由，需要说明的是，即使在Java的API中也有不少滥用继承的例子，例如Properties类继承了Hashtable类，Stack类继承了Vector类，这些继承明显就是错误的，更好的做法是在Properties类中放置一个Hashtable类型的成员并且将其键和值都设置为字符串来存储数据，而Stack类的设计也应该是在Stack类中放一个Vector对象来存储数据。记住：任何时候都不要继承工具类，工具是可以拥有并可以使用的，而不是拿来继承的。

迪米特法则

迪米特法则又叫最少知识原则，一个对象应当对其他对象有尽可能少的了解。

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。解决方案：尽量降低类与类之间的耦合。

自从我们接触编程开始，就知道了软件编程的总的原则：低耦合，高内聚。无论是面向过程编程还是面向对象编程，只有使各个模块之间的耦合尽量的低，才能提高代码的复用率。低耦合的优点不言而喻，但是怎么样编程才能做到低耦合呢？那正是迪米特法则

要去完成的。

迪米特法则又叫最少知道原则，最早是在1987年由美国Northeastern University的Ian Holland提出。通俗的来讲，就是一个类对自己依赖的类知道的越少越好。也就是说，对于被依赖的类来说，无论逻辑多么复杂，都尽量地的将逻辑封装在类的内部，对外除了提供的public方法，不对外泄漏任何信息。迪米特法则还有一个更简单的定义：只与直接的朋友通信。首先来解释一下什么是直接的朋友：每个对象都会与其他对象有耦合关系，只要两个对象之间有耦合关系，我们就说这两个对象之间是朋友关系。耦合的方式很多，依赖、关联、组合、聚合等。其中，我们称出现成员变量、方法参数、方法返回值中的类为直接的朋友，而出现在局部变量中的类则不是直接的朋友。也就是说，陌生的类最好不要作为局部变量的形式出现在类的内部。


六尺帐篷 (/u/f8e9b1c246f1)
写了 245418 字，被 15240 人关注，获得了 1429 个喜欢
(/u/f8e9b1c246f1)

被以下专题收入，发现更多相似内容

投稿管理

+ 收入我的专题

 程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)

 Android... (/c/5139d555c94d?utm_source=desktop&utm_medium=notes-included-collection)

 半栈工程师 (/c/383594e9265f?utm_source=desktop&utm_medium=notes-included-collection)

 Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)

 iOS Dev... (/c/3233d1a249ca?utm_source=desktop&utm_medium=notes-included-collection)