正文
面向对象设计的目标之一在于支持可维护性复用性，一方面需要实现设计方案或者源码的重用，另一方面要确保系统能够易于扩展和修改，具有较好的灵活性。 常用的设计原则有七个原则：

一、单一职责原则（single responsibility principle，SPR)

There should never be more than one reason for a clas s to change

当一个类承担的职责过多，就等于把这些职责耦合在一起，一个职责的变化可能会削弱或者抑制这类完成其他职责的能力。所以为了减轻这个类的负担，就要进行职责分离，将不同的支付封装在不同的类中。

单一职责原则(SRP)：就一个类而言，应该仅有一个引起它变化的原因。
单一职责原则是实现高内聚、低耦合的指导方针，它是最简单但又最难运用的原则，需要设计人员发现类的不同职责并将其分离，而发现类的多重职责需要设计人员具有较强的分析设计能力和相关实践经验。

例子：手机与照相机，它们都有照相的功能，但是手机的功能不仅仅只有照相还可以打电话，听音乐；但是照相机就只负责照相。因此这里面的手机职能就过多，在维护等方面就会比单一职责的困难很多。

二、开闭原则（Open-Close Principe，OCP)

没有十全十美的软件，随着时间的推移，软件也是需要更新换代的。当软件需要增加新的需求的时候，程序员应该尽量保证系统设计框架的稳定。如果一个软件符合开放—封闭原则，那么就可以很方便的进行扩展，无需修改其他的代码。

开放—封闭原则(ASD)：是说软件实体（类、模块、函数等等）应该可以可扩展，但是不可以修改。
此原则是面向对象设计的核心，遵循这个原则可以带来很多好处，当出现频繁变化的那部分的时候，就可以将其抽象出来。但是抽象的时候一定要注意：拒绝不成熟的抽象。

例子：考研与求职，在考研的时候，大家一贯的想法就是一心一意，断绝其他出路，却让自己失去了很多机会。其实在考研的同时完全可以抽出时间来去写一写简历，然后去求职，这

样还可以让自己放松一下，而且还能了解一些有关求职的事情，岂不是两全其美。既不影响自己考研，又可以有更多的机会去求职。这就是对扩展的开放，对修改的关闭的意义。

三、里氏替换原则（Liskov Substitution Principe，LSP)

一个软件实体如果使用的是一个父类的话，那么一定适用于其他子类，而且他察觉不出父类对象和子类对象的区别。换言之：**在软件里面，把父类都替换成他的子类，程序的行为没有发生变化。**

如果对于每一个类型为A的对象a，都有类型为B的对象b，使得A定义的所有程序P在所有对象a都替换成b时，程序P没有变化，那么B是A的子类型。

里式替换原则(LSP)：子类型必须能够替换掉它们的父类型
例子：企鹅是一种特殊的鸟，但是企鹅不会飞，若是在鸟这类中添加，飞这个行为，，那么企鹅将不能继承这个类。

四、依赖倒转原则（Dependence Inversion Principle)

高层模块不应该依赖与底层模块，二者都应该依赖于抽象，抽象不应该依赖于细节，细节应该依赖于抽象;针对接口而非实现编程。

这句话意思就是，不管你是北京人,还是北京朝阳群众，你都是中国人，这么理解就没错了.实际上，我还是感觉上句话好理解一点.

尽量引用高层抽象的类，即**使用接口和抽象类进行变量类型声明，方法返回类声明的转换等，而不要用具体的类来做这些事情。为了确保该原则的应用，一个具体类应当只实现接口或抽象类中声明过的方法，而不要给出多余的方法。**

依赖倒转原则(ASD)：
A 高层模块不应该依赖底层模块。两个都应该依赖抽象。
B 抽象不应该依赖细节。细节不应该依赖抽象

例子：我们是会用的电脑，cpu，内存等是分开的，当有那个地方出问题的时候，只需要将出问题的零件替换掉，但是想一想收音机，里面错综复杂，每个零件之间相互依赖，也许碰到一个其他的也会因此出问题。

五、接口隔离原则（Interface Segregation Principle, ISP)

只做自己应该做的事情，而不干不该干的事情。每个接口应该承担一种独立的角色。

接口隔离原则(ISP)：使用多个专门的接口，而不使用单一的总接口，即客户端不应该依赖那些它不需要的接口。
在使用接口隔离原则时，我们需要**注意控制接口的粒度**，接口不能太小，如果太小会导致系统中接口泛滥，不利于维护；接口也不能太大，太大的接口将违背接口隔离原则，灵活性较差，使用起来很不方便。这个原则可以和单一职责原则配合使用。

从接口隔离原则的定义可以看出，他似乎跟SRP有许多相似之处。 是的其实ISP和SRP都是强调职责的单一性， 接口隔离原则告诉我们在定义接口的时候要根据职责定义“较小”的接口，不要定义“高大全”的接口。也就是说接口要尽可能的职责单一，这样更容易复用，暴露给客户端的方法更具有“针对性”， 比如定义一个接口包括一堆访问数据库的方法， 有包括一堆访问网络的方法，还包括一些权限认证的方法。 把这么一摊子风牛马不相及的方法封装到一个接口里面，显然是不合适的， 如果客户程序只想用到数据访问的一些功能，但是调用接口的时候你把访问网络的方法和权限认证的方法暴露给客户，这使得客户程序感到“疑惑”，那么这个接口就不ISP，它很显然的构成了接口污染。

例子：还是手机吧，现在的智能手机简直是太强大了，若是每个接口不好好承担自己应该承担的，去做了其他的接口的事情那么智能手机就乱了，放歌的时候，开相机，打电话的时候，发短信。

六、合成复用原则（Composite Reuse Principe，CRP)

这个原则就是在一个新的对象中通过关联关系（组合关系和聚合关系）来使用一些已有的对象，使之成为新对象的一部分，**尽量使用合成/聚合，尽量少用继承**。

尽量使用对象的组合，而不是继承来达到复用的目的。

七、迪米特法则（Law of Demeter,LoD)

也称为：**最少知识原则**，清掉了类之间的松耦合，降低了系统的耦合度。

对象间尽量最少了解，彻底将API接口和具体实现相分离，模块间仅仅通过API进行通信。

迪米特法则(J&DP)：如果两个类不必彼此直接通信，那么这两个类就不应该发生直接的相互作用。如果其中一个类需要调用另一个类的某个方法的话，可以通过第三者转发这个调用。

在类的划分上，应当**尽量创建松耦合的类**，类之间的耦合度越低，就越有利于复用，一个处在松耦合中的类一旦被修改，不会对关联的类造成太大波及；在类的结构设计上，每一个类都应当尽量降低其成员变量和成员函数的访问权限；在类的设计上，只要有可能，一个类型应当设计成不变类；在对其他类的引用上，一个对象对其他对象的引用应当降到最低。

例子：还记得小的时候“不和陌生人说话”吗？也就是说你只能和你的朋友发生直接的交流，而不能和陌生人发生直接的交流，以免你出现危险，也就是带来不必要的影响。