1.单一职责原则

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。

- 如果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消弱或者抑制这个类完成其他职责的能力。这个耦合会导致脆弱的设计，当变化发生是，设计会遭受到意想不到的破坏。
- 软件设计真正要做的许多内容，就是发现职责并把那些职责相互分离。
- 如果你能够相到多于一个的动机去改变一个类，那么这个类就具有多于一个的职责，就应该考虑类的职责分离。

2.开放-封闭原则

开放–封闭原则，是说软件实体（类，模块，函数等等）应该可以扩展，但是不可修改。即对于扩展是开放的，对于更改是封闭的。

- 软件设计要容易维护又不容易出问题的最好办法就是多扩展，少修改。也就是说设计的时候，时刻要考虑，尽量让这个类足够好，写好了就不要去修改了，如果新需求来，我们增加一些类就行，原来的代码能不动则不动。
- 当初次编写代码时，假设变化不会发生。当变化发生时，我们就创建抽象来隔离以后发生的同类变化。
- 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 开放–封闭原则 的精神所在。
- 应该仅对程序中频繁出现变化的那部分做出抽象。

3.依赖倒置原则

依赖倒置原则，A.高层模块不应该依赖低层模块，两个都应该依赖抽象 B.抽象不应该依赖细节，细节应该依赖于抽象。
里氏替换原则，一个软件实体如果使用的是一个父类的话，那么一定适用于其子类，而且它察觉不出父类对象和子类对象的区别。也就是说在软件里面，把父类都替换成子类，程序的行为不会变化。简单说就是子类型必须能够替换掉它们的父类型。

- 只有当子类可以替换掉父类，软件单位的功能不受到影响时，父类才能真正被复用，而子类也能够在父类的基础上增加新的行为。
- 正是由于子类型的可替换性才使得使用父类类型的模块在无需修改的情况下就可以扩展。
- 依赖倒置其实就是谁也不依赖谁，除了约定的接口，大家都可以灵活自如。

4.迪米特法则

迪米特法则（LOD），如果两个类不必彼此直接通信，那么这两个类就不应当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一个方法的话，可以通过第三方转发这个调用

- 首先强调的是在类的结构设计上，每一个类都应当尽量降低成员的访问权限，也就是说尽量少的使用 public
- 迪米特法则强调了类之间的松耦合，类之间的耦合越弱，越有利于复用，一个处在弱耦合的类被修改，不会对有关系的类造成波及。也就是说信息的隐藏促进了软件的复用。