**原则一：（SRP：Single responsibility principle）单一职责原则又称单一功能原则**

核心：解耦和增强内聚性（高内聚，低耦合）

描述：

类被修改的几率很大，因此应该专注于单一的功能。如果你把多个功能放在同一个类中，功能之间就形成了关联，改变其中一个功能，有可能中止另一个功能，这时就需要新一轮的测试来避免可能出现的问题。

**原则二：开闭原则（OCP：Open Closed Principle）**

核心思想：对扩展开放，对修改关闭。

根据开闭原则，在设计一个软件系统模块（类，方法）的时候，应该可以在不修改原有的模块（修改关闭）的基础上，能扩展其功能（扩展开放）。

扩展开放：

某模块的功能是可扩展的，则该模块是扩展开放的。软件系统的功能上的可扩展性要求模块是扩展开放的。

修改关闭：

某模块被其他模块调用，如果该模块的源代码不允许修改，则该模块修改关闭的。软件系统的功能上的稳定性，持续性要求是修改关的。

**原则三：里氏替换原则（LSP：Liskov Substitution Principle）**

核心：

1.在任何父类出现的地方都可以用他的子类来替代（子类应当可以替换父类并出现在父类能够出现的任何地方）子类必须完全实现父类的方法。在类中调用其他类是务必要使用父类或接口，如果不能使用父类或接口，则说明类的设计已经违背了LSP原则。

2.子类可以有自己的个性。子类当然可以有自己的行为和外观了，也就是方法和属性

3.覆盖或实现父类的方法时输入参数可以被放大。即子类可以重载父类的方法，但输入参数应比父类方法中的大，这样在子类代替父类的时候，调用的仍然是父类的方法。即以子类中方法的前置条件必须与超类中被覆盖的方法的前置条件相同或者更宽松。

4.覆盖或实现父类的方法时输出结果可以被缩小。

**原则四：依赖倒转原则(DIP：Dependence Inversion Principle)**

别名：依赖倒置原则或依赖反转原则

核心：要依赖于抽象，不要依赖于具体的实现

1.高层模块不应该依赖低层模块，两者都应该依赖其抽象（抽象类或接口）

2.抽象不应该依赖细节（具体实现）

3.细节（具体实现）应该依赖抽象。

三种实现方式:

1.通过构造函数传递依赖对象

2.通过setter方法传递依赖对象

3.接口声明实现依赖对象

原则五：接口分离原则(ISP：Interface Segregation Principle)

核心思想：

不应该强迫客户程序依赖他们不需要使用的方法。

接口分离原则的意思就是：一个接口不需要提供太多的行为，一个接口应该只提供一种对外的功能，不应该把所有的操作都封装到一个接口当中.分离接口的两种实现方法：

1.使用委托分离接口。（Separation through Delegation）

2.使用多重继承分离接口。（Separation through Multiple Inheritance）

**原则六：合成复用原则（CRP：Composite Reuse Principle）**

核心思想：

尽量使用对象组合，而不是继承来达到复用的目的。该原则就是在一个新的对象里面使用一些已有的对象，

使之成为新对象的一部分：新的对象通过向这些对象的委派达到复用已有功能的目的。

复用的种类：

1.继承

2.合成聚合

注：在复用时应优先考虑使用合成聚合而不是继承

**原则七：迪米特原则（LOD：Law of Demeter）**

又叫最少知识原则

核心思想：

一个对象应当对其他对象有尽可能少的了解,不和陌生人说话。（类间解耦，低耦合）意思就是降低各个对象之间的耦合，提高系统的可维护性；在模块之间只通过接口来通信，而不理会模块的内部工作原理，可以使各个模块的耦合成都降到最低，促进软件的复用

注：

1.在类的划分上，应该创建有弱耦合的类；

2.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；

3.在类的设计上，只要有可能，一个类应当设计成不变；

4.在对其他类的引用上，一个对象对其它对象的引用应当降到最低；

5.尽量降低类的访问权限；

6.谨慎使用序列化功能；

7.不要暴露类成员，而应该提供相应的访问器(属性)

**七种设计原则总结**

单一职责原则：一个类只应该做和一个职责相关的事情，不要把过多的业务放在一个类中完成。

迪米特法则：软件实体之间应该做到最少的交互。不要和陌生人说话。调用方只关心他需要使用的方法

接口隔离原则：使用专门的接口，比用统一的接口要好。便于分工，在实现接口时，不应该看到自己不用关心的方法。

开闭原则：软件实体应该对扩展开放，对修改关闭。开闭原则是设计原则的核心原则，其他的设计原则都是开闭原则表现和补充。实现开闭原则的方法就是抽象。

聚合/组合复用原则。多使用聚合/组合达到代码的重用，少使用继承复用。

依赖倒置原则：面向抽象编程，不要面向具体编程。

**先从整体认识这七种设计思想。**

一、开闭原则：

这一条放在第一位来理解，它的含义是对扩展开放，对修改关闭。解释一下就是，我们写完的代码，不能因为需求变化就修改。我们可以通过新增代码的方式来解决变化的需求。

当然，这是一种理想的状态，在现实中，我们要尽量的缩小这种修改。

再解释一下这条原则的意义所在，我们采用逆向思维方式来想。如果每次需求变动都去修改原有的代码，那原有的代码就存在被修改错误的风险，当然这其中存在有意和无意的修改，都会导致原有正常运行的功能失效的风险，这样很有可能会展开可怕的蝴蝶效应，使维护工作剧增。

以下几种原则，都是为此原则服务的。

二、里氏替换选择：

此原则的含义是子类可以在任何地方替换它的父类。

里氏替换原则的潜台词是：尽量使用精准的抽象类或者接口。

三、单一职责原则：

单一职责的含义是：类的职责单一，引起类变化的原因单一。如果一个类做的事情太多，在组合的时候，必然会产生不必要的方法出现，这实际上是一种污染。

 单一职责的潜台词是：拆分到最小单位，解决复用和组合问题。

四、接口隔离原则：

接口隔离原则可以说是单一职责的必要手段，它的含义是尽量使用职能单一的接口，而不使用职能复杂、全面的接口。

接口隔离原则的潜台词是：拆分，从接口开始。

五、依赖倒置原则：

想要理解依赖倒置原则，必须先理解传统的解决方案。面相对象的初期的程序，被调用者依赖于调用者。也就是调用者决定被调用者有什么方法，有什么样的实现方式，这种结构在需求变更的时候，会付出很大的代价，甚至推翻重写。

依赖倒置原则就是要求调用者和被调用者都依赖抽象，这样两者没有直接的关联和接触，在变动的时候，一方的变动不会影响另一方的变动。

依赖倒置的潜台词是：面向抽象编程，解耦调用和被调用者。

六、迪米特原则：

迪米特原则要求尽量的封装，尽量的独立，尽量的使用低级别的访问修饰符。这是封装特性的典型体现。

一个类如果暴露太多私用的方法和字段，会让调用者很茫然。并且会给类造成不必要的判断代码。所以，我们使用尽量低的访问修饰符，让外界不知道我们的内部。

另外，迪米特原则要求类之间的直接联系尽量的少，两个类的访问，通过第三个中介类来实现。

迪米特原则的潜台词是：不和陌生人说话，有事去中介。

七、组合/聚合复用原则：

此原则的含义是，如果只是达到代码复用的目的，尽量使用组合与聚合，而不是继承。

继承的耦合性更大，比如一个父类后来添加实现一个接口或者去掉一个接口，那子类可能会遭到毁灭性的编译错误，但如果只是组合聚合，只是引用类的方法，就不会有这种巨大的风险，同时也实现了复用。

组合聚合复用原则的潜台词是：我只是用你的方法，我们不一定是同类。