设计原则：

SOLID

单一职责原则：Single Responsibility Principle，一个类或者模块只负责一个职责或功能。不要设计大而全的类，要设计粒度小、功能单一的类。如何判断类的设计是否符合单一职责原则呢，要根据具体的业务需求而定。

开闭原则：Open Closed Principle，软件实体（模块、类、方法等）对扩展开放，对修改关闭，简单理解就是，当添加一个功能时，应该在已有的代码的基础上扩展代码（新增类、模块、方法等），而非修改已有代码。com/subei/designprinciple/ocp/useocp此包下为列举的样例（一个消息告警的例子）。开闭原则并不是说完全杜绝修改，而是以最小的修改代码的代价来完成新功能的开发。同样的修改代码，在粗粒度下去看，可能是修改，在细粒度下去看，又可被认定为扩展。

里氏替换原则：Liskov Substitution Principle,子类对象能够替换程序中父类对象出现的任何地方，并且保证原来程序的逻辑行为不变及正确性不被破坏。看起来和多态有点类似，但是他们关注的角度是不一样的，多态是面向对象编程的一大特性，它是一种代码实现的思路。而里氏替换是一种原则，用来指导继承关系中子类该如何设计，子类的设计要保证在替换父类的时候，不改变原来的程序逻辑以及不破坏原有程序的正确性。进一步解释就是，子类在设计的时候要遵守父类的行为约定。父类定义了函数的行为约定，这里的行为约定包括：函数声明要实现的功能，对输入、输出、异常的约定。例如，子类重写父类的方法，父类中的方法，是没有抛出异常的，但是在子类中，抛出了异常，这就属于不符合里氏替换原则的代码。

接口隔离原则：Interface Segregation Principle，调用者不应该被强迫依赖它不需要的接口。

如果把接口理解为一组接口集合，可以是微服务的接口，也可以是某个类库的借口。如果部分借口只被部分调用者使用，我们就需要将这部分接口隔离出来，单独给这部分调用者使用，而不应强迫调用者也依赖这部分不会用到的接口。

如果把接口理解为API接口或函数，部分调用者只用到函数中的部分功能，那应该把函数拆分成粒度更小的函数，让调用者只依赖它需要的那个细粒度函数。

如果把接口理解为OOP中的接口，那接口的设计要尽量单一，不要让接口的实现者和调用者，依赖它不需要的接口。

依赖反转：DIP，高层模块不依赖低层模块，它们共同依赖同一个抽象。抽象不依赖具体实现细节，具体实现细节依赖抽象。高层模块是指调用者。举例：tomcat不依赖应用程序，它们共同依赖servlet规范，servlet不依赖tomcat和应用程序的实现细节，tomcat和应用程序依赖servlet规范。控制反转，这里的控制反转和spring中的控制反转不一样，控制是指对程序流程的控制，反转是指流程由程序员交由框架；依赖注入是一种具体的编码技巧，不使用new方式创建对象，而是将依赖的类对象在外部创建好之后，通过构造函数或者setter方式注入。

KISS：

YAGNI：

DRY：

LOD：迪米特法则

装饰模式：为了扩展类的功能，相比较于继承方式的话，装饰模式更灵活，可以自由组合需要扩展的功能。比如java中的IO就是使用该模式实现。