里氏替换法则

里氏替换法则(Liskov Substitution Principle LSP)是面向对象设计的六大基本原则之一（单一职责原则、里氏替换原则、依赖倒置原则、接口隔离原则、迪米特法则以及开闭原则）。这里说说里氏替换法则：父类的一个方法返回值是一个类型T，子类相同方法(重载或重写)返回值为S，那么里氏替换法则就要求S必须小于等于T，也就是说要么S和T是同一个类型，要么S是T的子类，为什么呢？分两种情况，如果是重写，方法的输入参数父类子类是相同的，两个方法的范围值S小于等于T，这个是重写的要求，这个才是重中之重，子类重写父类的方法，天经地义；如果是重载，则要求方法的输入参数不相同，在里氏替换法则要求下就是子类的输入参数大于等于父类的输入参数，那就是说你写的这个方法是不会被调用到的，参考上面讲的前置条件。

当然，覆盖或实现父类的方法时，输入参数可以被放大，比如：

```
public class Father {
    public Collection doSomething(HashMap map){
        System.out.println("父类被执行...");
        return map.values();
    }
}

public class Son extends Father {
    //放大输入参数类型
    public Collection doSomething(Map map){
        System.out.println("子类被执行...");
        return map.values();
    }
}

public class Client {
    public static void invoker(){
        //父类存在的地方，子类就应该能够存在
            Father f = new Father(); HashMap map = new HashMap();
            f.doSomething(map);
    }
    public static void main(String[] args) {
        invoker();
    }
}
```

输出结果为：父类被执行...

根据里氏替换法则说是父类出现的地方子类就能出现，我们把上面的黄色部分修改为子类，程序如下：

```
public class Client {
    public static void invoker(){
        //父类存在的地方，子类就应该能够存在
        Son f =new Son(); HashMap map = new HashMap();
        f.doSomething(map);
    }

    public static void main(String[] args) {
        invoker();
    }
}
```

输出结果为：父类被执行...

父类方法的输入参数是HashMap类型，子类的输入参数是Map类型，也就是说子类的输入参数类型的范围扩大了，子类代替父类传递到调用类用，子类的方法永远都不回被执行，这是正确的，如果你想让子类的方法运行，你就必须重写父类的方法。