设计模式原则 - 依赖倒转原则

寂然

官方定义

依赖倒转原则，又称依赖倒置原则（Dependence Inversion Principle），又称DIP原则，官方定义为：

> High level modules should not depend upon low level modules. Both should depend upon abstractions.
>
> （上层模块不应该依赖底层模块，它们都应该依赖于抽象）
>
> Abstractions should not depend upon details. Details should depend upon abstractions.
>
> （抽象不应该依赖于细节，细节应该依赖于抽象）

基本介绍

> 抽象？ 接口或者抽象类
>
> 细节？ 实现类
>
> 换句话说 依赖倒转原则 核心的理念 相对于细节来说，抽象要稳定得多
>
> **要求我们 面向接口编程**
>
> 抽象类和接口 价值？ 设计

案例演示 - 钉钉消息

案例分析

解决方案

案例总结

> 依赖倒转原则本质上 就是通过抽象（抽象类和接口）使得各个类或者模块实现彼此独立，互相不影响，实现
>
> 模块间松耦合，要先顶层再细节的方式来进行代码设计

依赖关系传递方式

一、通过接口传递

```
//通过接口传递
//假设消息规范

interface IMessage{

    void sendMessage(Produce produce);
}

//消息真正的生产者
interface Produce{

    void produceMessage();
}

class Worker implements IMessage{

    @Override
    public void sendMessage(Produce produce) {
        produce.produceMessage();
    }
}
```

二、通过构造方法传递

```
//通过构造器来传递
interface IMessage{

    void sendMessage();
}

//消息真正的生产者
interface Produce{

    void produceMessage();
}

class Worker implements IMessage{

    public Produce produce;

    public Worker(Produce produce){

        this.produce = produce;
    }

    @Override
    public void sendMessage() {
        this.produce.produceMessage();
    }
}
```

三、通过set()方法传递

```
//通过set()方法
interface IMessage{

    void sendMessage();
}

//消息真正的生产者
interface Produce{

    void produceMessage();
}

class Worker implements IMessage{

    public Produce produce;

    public void setProduce(Produce produce){
        this.produce = produce;
    }
```

```
    @Override
    public void sendMessage() {
        this.produce.produceMessage();
    }
}
```

注意事项&细节

- 低层模块尽量都要有抽象类或接口，或者两者都有，程序稳定性更好
- 变量的声明类型尽量是抽象类或者接口，这样我们的变量引用和实际对象间，就存在一个缓冲层，利于程序扩展和优化
- 继承时遵循里式替换原则（下一节）

下节预告