设计模式原则 - 单一职责原则

寂然

注：该课件内容都是课上录制时手敲的，重要的点，经过了下整理，完整可观看视频哈

官方定义

> 单一职责原则（Single Responsibility Principle, SRP），有且仅有一个原因引起类的变更

顾名思义：一个类只负责一项职责

基本介绍

即对类来说，一个类应该只负责一项职责。如类 A 负责两个不同职责：职责 1，职责 2，当职责 1 需求变更而

改变 A 时，可能造成职责 2 执行错误，所以需要将类 A 的粒度分解为 A1，A2

案例：动物世界

> 需求：有一个动物类，里面定义一个在森林奔跑的方法，创建动物的实例，调用方法，方法内执行打印操作

解决方案一

拆分类，拆分类为更小的粒度

```
public class SingleDemo {

    public static void main(String[] args) {

        /*new Animal().run("老虎");
        new Animal().run("狮子");
        new Animal().run("老鹰");*/

        //方案一
```

```
        new ForestAnimal().run("老虎");
        new ForestAnimal().run("狮子");
        new ForestAnimal().run("老狼");
        new AirAnimal().fly("老鹰");
    }
}
//需求：有一个动物类，里面定义一个在森林奔跑的方法，创建动物的实例，调用方法，方法内执行打印操作

/*class Animal{

    //森林奔跑的方法
    public void run(String animal){

        System.out.println(animal + "在森林里愉快的奔跑");

    }
}*/

class ForestAnimal{

    public void run(String animal){

        System.out.println(animal + "在森林里愉快的奔跑");

    }
}

class AirAnimal{

    public void fly(String animal){

        System.out.println(animal + "在天空中愉快的飞翔");

    }
}
```

方案一分析

优势：符合了业务逻辑，符合单一职责原则

劣势：改动幅度大，不光是要拆分类，客户端的代码也要进行大幅度的改动

？？

解决方案二

直接在原来类的基础上进行改动？？

方案二分析

优势：改动幅度下，包括客户端，完成了业务逻辑

劣势：？是否违反单一职责原则？（方法级别的遵守）

单一职责原则：各司其职

> 通过上面两种方案，大家可以看到，方案一，类级别遵守了单一职责原则，但是改动的代价很高，方案二方
>
> 法级别遵守了单一职责原则，改动幅度较小，综上所述，单一职责原则最核心的什么？
>
> 各司其职

注意事项&细节

- 降低类的复杂度，一个类只负责一项职责

  （一个类职责少了，相应的复杂度不就低了嘛）

- 提高类的可读性以及可维护性

  （相应的复杂度降低了，代码量就会减少，可读性也就提高了，可维护性自然也就提高了）

- 降低变更引起的风险

  （一个类职责越多，变更的可能性就会越大，变化带来的风险就会越大）

- 通常情况下，我们应当遵守**类级别**单一职责原则

  （只有逻辑足够简单，才可以在代码中违反单一职责原则）

如何遵守单一职责原则?

其实就是**合理的职责分解**，

从业务出发，从需求出发，识别出同一个类型的职责

需要说明一点：单一职责原则不是面向对象语言特有的，只要是模块化的程序设计，都要遵守单一职责原则

下节预告

接口隔离原则