18 单一职责原则

就一个类而言，应该仅有一个引起它变化的原因。更多应用在JS的对象与方法级别上。如果我们有多种动机去改写一个方法，那么这个方法就拥有多个职责，如果承担过多职责那么在需求变迁过程中就会经常改写

这种方法或者对象就是不稳定的，职责耦合太严重，因此SRP原则体现为：一个对象（方法）只做一件事情

设计模式中的SRP原则

1. 代理模式

例如通过增加虚拟代理方式，把图片预加载职责迁移到代理对象，本体仍然负责往页面中添加img标签的职责。

这样职责分开到两个对象中，各自因需求改变时不会相互影响

2. 迭代器模式

迭代器模式也是负责将遍历对象 or 数组的职责提取出来，让遍历时具体做的操作和遍历本身解耦开来，这样就不需要因为遍历方式改变来涉及修改操作相关的代码块了

3. 单例模式

例如结合单例模式和SPR原则，我们将可以将单例管理的职责和具体业务诉求的职责分开，来实现创建唯一业务要求的功能。

例如业务方要求创建全局唯一登陆浮窗，我们只需要将单例模式和创建浮窗的功能连接在一起就行了，而且也不会互相干扰：

```
1 const getSingle = () => {...}
2 const createLoginModal = () => {...}
3
4 const createSingleLoginModal = getSingle(createLoginModal);
```

4. 装饰者模式

通常我们可以让类或者对象一开始只有一些基础职责，更多职责可以在运行时根据需要装饰到对象上面。例如使用装饰者模式

何时应该分离职责

这里需要注意实际应用，并不是所有的职责都应该一一分离，避免粒度过细导致低内聚

- 如果随着需求变化，多个职责总是同时变化，可以将其封装在一起，不必分离。例如ajax请求的xhr对象创建和发送本身就是串行的职责
- 即使两个职责已经耦合在一起，但它们没有发生改变的征兆就没必要主动分离，可以等代码需要重构时候再进行分离

违反SPR原则

把握好SPR原则的度还是有一点困难的，一方面我们需要了解感受这个原则，另一方面在一些场景下组合可能会简化用户的使用，虽然降低稳定性但是提高了便利性

具体选择稳定还是方便并没有标准答案，取决于实际业务场景，不要一味的进行细粒度拆解，导致代码过于分散难以阅读

SPR原则优缺点

优点：

- 降低单个类或者对象复杂度，粒度变小，解耦
- 利于代码复用
- 利于单元测试

缺点：

- 增加代码复杂度
- 降低对象之间的联系与可读性