

## 定义

应该有且仅有一个原因引起类的变更

## 说明

单一职责原则为我们提供了一个编写程序的准则，要求我们在编写类，抽象类，接口时，要使其功能职责单一纯粹，将导致其变更的因素缩

如果一个类承担的职责过多，就等于把这些职责耦合在一起。一个职责的变化可能会影响或损坏其他职责的功能。而且职责越多，这个类变低

## 举例说明

例如下面的工厂类，负责 将原料进行 预处理 和 加工成产品X和产品Y

```
1 public class Factory {
2
3     private String preProcess(String material){
4         return "*" + material + "*";
5     }
6     public String processX(String material) {
7         return preProcess(material) + "加工成: 产品X";
8     }
9
10    public String processY(String material) {
11        return preProcess(material) + "加工成: 产品Y";
12    }
13 }
```

现因市场需求，优化 产品X 的生产方案，需要改变原料预处理的方式  
将预处理方法

```
1 private String preProcess(String material){
2     return "#" + material + "#";
3 }
```

在以下场景类中运行

```
1 public class Client {
2     public static void main(String args[]) {
3         Factory factory = new Factory();
4         System.out.println(factory.processX("原料"));
5         System.out.println(factory.processY("原料"));
6     }
7 }
```

运行结果如下：

```
1 修改前：
2 *原料*加工成: 产品X
3 *原料*加工成: 产品Y
4 修改后：
5 #原料#加工成: 产品X
6 #原料#加工成: 产品Y
```

从运行结果中可以发现，在使产品X可以达到预期生产要求的同时，也导致了产品Y的变化，但是产品Y的变化并不在预期当中，这便导致程

那么按照单一职责划分

```
1 // 定义一个抽象类
2 public abstract class AFactory {
3
4     protected abstract String preProcess(String material);
5
6     public abstract String process(String material);
7 }
8
9
10 // x产品生成的独立实现
11 public class FactoryX extends AFactory{
12
13     @Override
14     protected String preProcess(String material) {
15         return "*" + material + "*";
16     }
17
18     @Override
19     public String process(String material) {
20         return preProcess(material) + "加工成: 产品X";
21     }
22 }
23
24 // y产品生成的独立实现
25 public class FactoryY extends AFactory{
26
27     @Override
28     protected String preProcess(String material) {
29         return "*" + material + "*";
30     }
31
32     @Override
33     public String process(String material) {
34         return preProcess(material) + "加工成: 产品Y";
35     }
36 }
37
38 public class Client2 {
39
40     public static void main(String args[]) {
41         produce(new FactoryX());
42         produce(new FactoryY());
43     }
44
45     private static void produce(AFactory factory) {
46         System.out.println(factory.process("原料"));
47     }
48
49 }
```

