一.面向切面编程(AOP):是对面向对象编程的补充,能减少代码的冗余

1.面向切面编程的术语:

1.切面(Aspect):一个关注点的模块化,这个关注点可能横切多个对象

2.连接点(JointPoint):程序执行到某个位置

3.通知(Advice):切面在连接点执行的动作

4.切入点(Pointcut):切面选择的连接点

5.目标对象(TargetObject):被切面通知的对象

6.织入(Weaving):把切面分对象关联,并创建该对象的代理对象的过程

7.引入(Introduction):在不修改代码的前体下,引入可以在运行期为类动态添加一些方法和字段

8.增强:为类织入的过程,简单理解为为类添加新功

2.通知的类型:

1.before:前置通知,在连接点之前通知

2.after return:返回后通知,在连接点正常完成后执行

3.around:环绕通知,可以自定义在连接点前或后执行

4.after throwing:抛出异常后通知,在连接点指向后抛出异常后通知

5.after:后置通知,在连接点执行后(包括正常执行和抛出异常)通知

3.pointcut expression:切入点的表达式,用于匹配指定的连接点(Spring中连接点都是方法级)

博客:https://www.cnblogs.com/imzhuo/p/5888007.html

bean(bean的id):匹配bean中的所有方法

execution(访问修饰符 返回值数据类型 方法名(参数类型)):匹配指定方法

this:匹配指定的类

target:匹配指定接口的实现类

within:匹配指定包中的

确保AspectJ能用要求，aspectjweaver.jar在1.6.8以上的版本

添加识别AspectJ的两种方式

1.XML中配置

```
1 1 <aop:aspectj-autoproxy/>
```

2.java文件中配置

```
1 1 @Configuation
2 2 @EnableAspectJAutoProxy
3 3 public class Config{
4 4
5 5 }
```

3.切面

当切面类使用@Aspect注解配置，拥有@Aspect的任何bean被Spring自动是被并应用

用@Aspect注解的类可以有方法和字段，也可以有切入点，通知，和引入声明

@Aspect注解不能通过类路径自动检测发现，需要配合@Comtpnonent注释或xml的bean配置

```
1 1 @Aspect
2 2 public class 类名{
3 3
4 4 }
```

4.切入点

一个切入点是由一个普通方法通过@Pointcut注解生成的，且方法返回值必须为void

```
1 @Aspect
2 public class 类名{
3 @Pointcut("切入点表达式")
4 public void 方法名(){
5 方法体
6 }
7 }
```

5.通知

```
1 @Aspect
2 public class 类名{
3 @Pointcut("切入点表达式")
4 public void pointcut(){
5 方法体
6 }
7 //前置通知
8 @Before("可以是切入点表达式，也可以是上面定义过的切入点例如：pointcut()")
9 public void 方法名(){
10 方法体
11 }
12 //后置通知
13 //返回后的通知
14 //异常后的通知
15 //都与前置通知一样
16
17 //环绕通知
18 @Around("切入点表达式")
19 public Object 环绕通知方法名(ProceedingJoinPoint pjp){
20 //在调用方法之前
21 System.out.print("这是在方法之前输出的");
22 //指的是具体业务方法执行
23 Object 对象 = pjp.proceed();
24 return 对象
25 //在调用方法之后
26 System.out.print("这是在方法之后输出的");
27 }
```

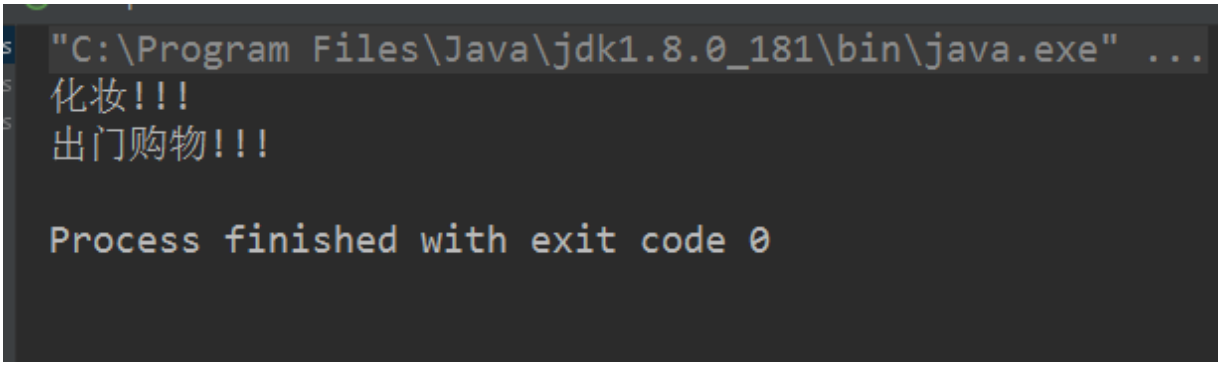
```
28 28
29 29 }
30
```

具体操作：

在resource中配置xml，eg：女孩出门前化妆

```
1  <bean id="action" class="com.lanou.aop.Action"/>
2  <bean id="girl" class="com.lanou.aop.Girl"/>
3
4  <!--配置app-->
5  <aop:config>
6   <!--定义切面-->
7   <aop:aspect id="aspect1" ref="action">
8   <!--定义切入点-->
9   <!--<aop:pointcut id="pointcut1" expression="bean(girl)"/>-->
10   <aop:pointcut id="pointcut1" expression="execution(public void shopping
())"/>
11   <!--定义通知-->
12   <!--针对哪个切入点-->前
13   <aop:before method="makeup" pointcut-ref="pointcut1"/>
14   </aop:aspect>
15
16  </aop:config>
17
```

执行后:



2.注解,包扫描:新建boy类和work类

```
1  <context:component-scan base-package="com.lanou.annonation"/>
2
3
```

```
<!--支持aop注解-->
<aop:aspectj-autoproxy/>
```

```
@Component
public class Boy {
 @Pointcut("bean(boy)")
 public void coding(){
 System.out.println("编码!");
 }
}
```

work类中:

```
@Aspect
@Component
public class Work {
 @Pointcut("bean(boy)||bean(girl)")
 public void abc() {
 }

 @Before("abc()")
 public void printTime1() {
  //打印当前时间
  System.out.println(System.currentTimeMillis());
  }

  @After("abc()")
  public void printTime2() {
  //打印当前时间
  System.out.println(System.currentTimeMillis());
  }
}
```

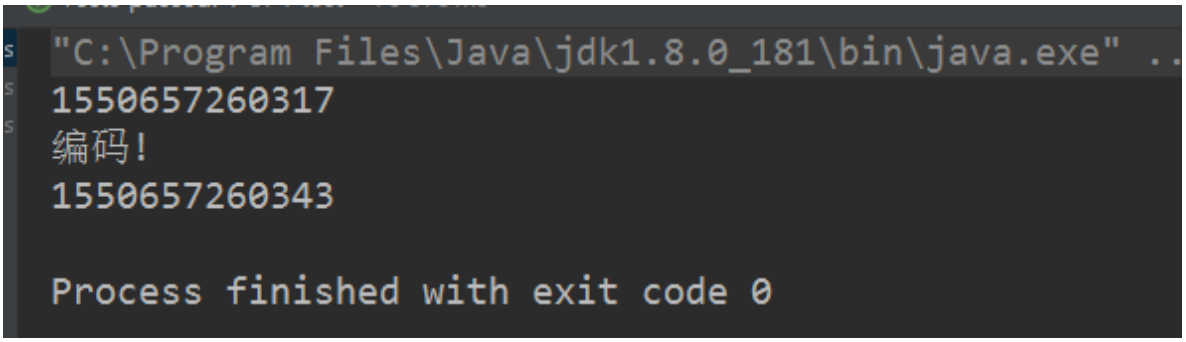
测试类:

```
@Test
void test2() {
  ClassPathXmlApplicationContext context=new
ClassPathXmlApplicationContext("spring-context1.xml");

  Boy boy = (Boy) context.getBean("boy");
  boy.coding();
```

```
7 }
```

执行结果:


"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ..
1550657260317
编码!
1550657260343
Process finished with exit code 0