创建Maven项目使用AspectJ框架实现AOP

项目中使用AOP：目的是给已经存在的一些类和方法在不改变原来代码的前提下，增加额外的功能。

--AspectJ框架实现AOP的基本步骤：
1、新建maven项目
2、引入依赖
    单元测试依赖Junit
    Spring依赖spring-context
    AspectJ框架依赖spring-aspects
3、创建目标类：接口和其实现类
    需要做的就是给这些类增加额外功能
4、创建切面类：普通类
    1）、在类的上面加上AspectJ框架的注解@Aspect，该注解声明这个类是切面类
    2）、在类中定义方法，就是切面要执行的功能代码
        然后在方法上加入AspectJ中的通知注解，如@Before，表示该方法在目标方法执行之前执行
        通知注解有个value属性，该属性的值填的是指定切入点的表达式:@Before(value="execution(...)")
5、创建Spring配置文件:声明对象，把对象交给容器统一创建，赋值和管理
    声明对象可以用xml或者注解的方式
    1、声明目标类对象
    2、声明切面类对象
    3、如果是注解方式就声明组件扫描器
    4、声明AspectJ框架中自动代理对象生成器标签
        自动代理生成器：用来自动创建代理对象的，到时我们直接用获取目标类对象的方式从Spring容器中拿到的对象就
        是一个代理对象，虽然看起来拿到的是目标类的对象，但底层早就把把原来真正的目标类替换了，但是方法还是一样
        的，当然拿到的对象是代理对象前提是AspectJ实现AOP的前提下，只要有一个步骤错了，拿到的就不是代理对象
        了，我们可以通过反射机制获取对象的Class对象在获取名字验证一下proxy.。
6、创建测试类，测试代理对象

在idea-maven-spring中创建项目模块d-spring-aop-aspectj-04，骨架模板选择maven的quickstart。去掉pom文件中多于的无用的东西，新增resources资源目录。添加Spring依赖、Junit依赖和AspectJ依赖。

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.studymyself</groupId>
  <artifactId>d-spring-aop-aspectj-04</artifactId>
  <version>1.0-SNAPSHOT</version>
```

```
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>

<!--    测试依赖-->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

<!--    spring依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.2.4.RELEASE</version>
    </dependency>

<!--    AspectJ依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aspects</artifactId>
      <version>5.2.3.RELEASE</version>
    </dependency>

  </dependencies>

  <build>

  </build>
</project>
```

1、切面使用@Before：前置通知实现的AOP，可以有JoinPoint类参数

1）、pom文件已经配置好了。在d-spring-aop-aspectj-04中创建一个bao01包，创建SomeService接口和其实现类SomeServiceImpl，这是目标类。然后创建一个切面类MyAspectJ，其中编写切面功能方法代码。具体如下：

```java
package com.studymyself.bao01;

public interface SomeService {

    public void doSome(String name,Integer age);

}
```

```java
package com.studymyself.bao01;

//目标方法
public class SomeServiceImpl implements SomeService {
    @Override
    public void doSome(String name,Integer age) {
        //添加新功能：（目标方法执行之前）输出doSome方法执行时间
        System.out.println("==目标方法doSome执行==");
    }
}
```

```java
package com.studymyself.bao01;


import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import java.util.Date;

/**
 * @Aspect:这是AspectJ框架的注解
 *    作用：表示当前类是切面类
 *    切面类：用来给业务方法增加功能的类，里面有切面功能的代码
 *    位置：写在定义类的上面
 */
@Aspect
public class MyAspectJ {

    /**
     * 切面类中前置通知方法定义规则：
     * 1、公共方法，public修饰
     * 2、无返回值，void
     * 3、方法名自定义，但要见名知意
     * 4、方法参数可以有，也可以没有
     *      如果有参数，不能自定义，已经规定好了哪几类参数可以使用
     */

    /**
     * 注解@Before：前置注解
     *  属性value：值是切入点表达式，表示切面功能的执行位置（就是指定哪个目标方法增加功能）
     *  位置：写在定义方法上面
```

```
     *  该注解特有功能：
     *  1、使该切面在目标方法执行之前执行
     *  2、不改变目标方法执行结果
     *  3、不影响目标方法的执行
     */
    //切入点表达式简介写法：只写两个必须部分
    // @Before(value = "execution(void doSome(String,Integer))")

    //@Before(value = "execution(void
*..SomeServiceImpl.doSome(String,Integer))")

    @Before(value = "execution( public void *..SomeServiceImpl.*(..))")

    //切入点完整的表达式写法，注意目标方法不是写接口的,参数名不用写，只写参数类型
    //@Before(value = "execution( public void
com.studymyself.bao01.SomeServiceImpl.doSome(String,Integer))")
    public void myBefore(){
        //增加的功能，即切面代码
        System.out.println("使用@Before前置通知的切面功能：输出目标方法执行时间\n"+new
Date());
    }
}
//从上面我们就可以理解说一个切面要知道其三要素了，上面的
// @Before注解确定了切面的执行时间,而其中的value属性填的值是切入点表达式确定了该切面执行位置
//myBefore方法就确定了切面的功能。这三要素。
```

2）、创建spring配置文件applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

<!--    把包括切面类的所有对象交给容器生成和管理等-->
<!--    声明目标对象-->
    <bean id="someService" class="com.studymyself.bao01.SomeServiceImpl"/>

<!--    声明切面类对象-->
    <bean id="myAspect" class="com.studymyself.bao01.MyAspectJ"/>

<!--    声明自动代理生成器：使用AspectJ框架的内部功能创建目标对象的代理对象-->
<!--    创建代理对象是在内存中实现的，原理是修改目标对象的内存结构，然后作为-->
<!--    代理对象，所以代理对象就是修改后的目标对象，获取的方法和获取目标对象-->
<!--    的方式不变。-->
    <aop:aspectj-autoproxy />
<!--    aspectj-autoproxy启动后就会扫描Spring容器中的所有对象，用反射机制获取这些对象中的
AspectJ-->
<!--    框架中的注解，如@Aspect，根据该注解的判断哪些对象是切面对象；通知注解如@Before，由该
注解-->
<!--    中的value属性判定是否是目标类，读取注解中的切入点表达式，获取切入点的类名-->
<!--    从而判断Spring容器中那些对象是目标类对象，然后为其生成代理对象。-->
<!--    所以会把Spring容器中所有需要添加切面目标类对象一次性全部生成代理对象-->
</beans>
```

3）、创建MyTest01测试类，如下

```
package com.studymyself;

import com.studymyself.bao01.SomeService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest01 {

    /**
     * 测试前置通知的AOP实现
     */
    @Test
    public void testMyBefore(){

        //定义Spring配置文件路径
        String config = "applicationContext.xml";

        //创建Spring容器对象
        ApplicationContext ac = new ClassPathXmlApplicationContext(config);

        //获取目标类对象或代理对象（前提这个类是实现了AOP情况下）
        SomeService proxy = (SomeService) ac.getBean("someService");

        //用反射获取该对象的完整类名，验证是是否是代理对象
        System.out.println("proxy对象完整类名："+proxy.getClass().getName());
        //proxy对象完整类名：com.sun.proxy.$Proxy8  -->确实是JDK动态代理的代理类名
        //当我们把切面类的切面表达式中方法参数写错或少写后输出如下:
        //proxy对象完整类名：com.studymyself.bao01.SomeServiceImpl
        //下面执行doSome方法也没有进行功能增强，所以写切面表达式时需要注意

        //直接执行接口中doSome方法，验证是否进行了功能增强
        proxy.doSome("jhfjkah",75757);
    }
}
```

4）、修改切面类中的方法，在方法中添加参数。

上面有提到切面方法中的参数不能随便添加，所以这里添加的是JointPoint类型参数是可添加类型之一，该参数是这些通知方法中都可以用的。很明显这个类型是我们所熟知的连接点，即单个业务目标方法，具体作用如下所示：

```
package com.studymyself.bao01;


import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
```

```java
import java.util.Date;

/**
 * @Aspect:这是AspectJ框架的注解
 *    作用：表示当前类是切面类
 *    切面类：用来给业务方法增加功能的类，里面有切面功能的代码
 *    位置：写在定义类的上面
 */
@Aspect
public class MyAspectJ {

    /**
     * 切面类中前置通知的方法定义规则：
     * 1、公共方法，public修饰
     * 2、无返回值，void
     * 3、方法名自定义，但要见名知意
     * 4、方法参数可以有，也可以没有
     *      如果有参数，不能自定义，已经规定好了哪几类参数可以使用
     */


    /**
     * 指定通知方法中的参数：JoinPoint
     * JoinPoint：连接点，即要加入切面功能的业务方法
     *       作用：
     *       在通知方法中获取业务方法的信息，如方法名、方法的实参等等。
     *       如果切面功能需要用到这些信息就加入JoinPoint类型参数
     *       该参数是由框架赋予值的，而且必须是第一位置的参数
     *       具体如下：
     */
    @Before(value = "execution( public void *..SomeServiceImpl.*(..))")
    public void myBefore(JoinPoint jp){

        //获取方法否完整定义
        System.out.println("方法的签名（定义）="+jp.getSignature());
        System.out.println("方法名="+jp.getSignature().getName());

        //获取方法的实参，有两个，一个返回的是Object数组，一个返回的是第一个实参
        Object[] args = jp.getArgs();
        for (Object arg:args){
            System.out.println("参数="+arg);
        }

        //增加的功能，即切面代码
        System.out.println("使用@Before前置通知的切面功能：输出目标方法执行时间\n"+new
Date());
    }
}
```

测试方法运行结果图如下：