

Day27

java课程-李彦伯

Day27

注解配置Spring

配置步骤

配置到容器中的注解

@scope配置创建对象模式

属性注入的注解

基本数据类型和String

@value注入属性值(对于基本数据类型)

引用类型

@Autowired和@Qualifier

@Resource

声明周期方法

Spring中的JUnit4测试

aop(面向切面编程)

Spring中两种代理模式

动态代理

cglib代理

aop中的名词解释

aop-xml配置

导包

编写通知代码

spring中通知的类型

编写目标对象

进行文件配置

注解配置Spring

配置步骤

- 导包4+2+1(spring包中的aop包)
- 配置Context的schema约束,
- 在application.xml文件中添加一句话

```
<context:component-scan base-package="com.zhiyou100.annotation">
</context:component-scan>
```

会扫描package中设置的包,并且包含其子包中的所有配置注解的类,会将其注入到spring容器中
- 在包下的类中把需要spring容器进行管理的类,添加注解

配置到容器中的注解

把要配置到容器中的类,在其头上添加注解,spring通过扫描就能将其放在spring的容器中,有四种注解的功能都是一样的,推荐使用后三个

- @Component标准一个普通的spring Bean类。
- @Service:用以区分将此类是service层

- @Controller:用以区分将此类是web层
- @Repository:用以区分此类是dao层

```
@Repository("user")
public class User {
    private String name;
    private int age;
    private Car car;
    public User() {
        super();
    }
    public Car getCar() {
        return car;
    }
    public void setCar(Car car) {
        System.out.println("setCar");
        this.car = car;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        System.out.println("setName");
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        System.out.println("setAge");
        this.age = age;
    }
}
```

```
    }  
    @Override  
    public String toString() {  
        return "User [name=" + name + ",  
age=" + age + ", car=" + car + "];"  
    }  
}
```

@scope配置创建对象模式

- @Scope(scopeName="singleton")指定单例模式创建对象,默认是单例
- @Scope(scopeName="prototype")指定多例模式创建对象

```
@Repository("user")
@Scope(scopeName="singleton")
public class User {
    private String name;
    private int age;
    private Car car;
    public User() {
        super();
    }
    public Car getCar() {
        return car;
    }
    public void setCar(Car car) {
        System.out.println("setCar");
        this.car = car;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        System.out.println("setName");
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        System.out.println("setAge");
    }
}
```

```
        this.age = age;
    }
    @Override
    public String toString() {
        return "User [name=" + name + ",
age=" + age + ", car=" + car + "]";
    }
}
```

属性注入的注解

基本数据类型和String

@value注入属性值(对于基本数据类型)

使用此注解是注入的类型是基本类型包括字符串,可以将其放在成员变量上,也可以将其放在set方法上,格式为 `@value("xxx")`


```
@Repository("user")
@Scope(scopeName="singleton")
public class User {
    @Value("张三")
    private String name;
    private int age;
    private Car car;
    public User() {
        super();
    }
    public Car getCar() {
        return car;
    }
    public void setCar(Car car) {
        System.out.println("setCar");
        this.car = car;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        System.out.println("setName");
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    @Value("18")
    public void setAge(int age) {
```

```
        System.out.println("setAge");
        this.age = age;
    }
    @Override
    public String toString() {
        return "User [name=" + name + ",
age=" + age + ", car=" + car + "];"
    }
}
```

引用类型

@Autowired和@Qualifier

会自动找spring容器中的相同类型的值进行匹配,如果容器中有多个符合类型的那么得再添加一个

`@Qualifier("bean的名称")` 注解

```
@Repository("user")
@Scope(scopeName="singleton")
public class User {
    @Value("张三")
    private String name;
    private int age;
    @Autowired
    @Qualifier("car2")
    private Car car;
    public User() {
        super();
    }
    public Car getCar() {
        return car;
    }
    public void setCar(Car car) {
        System.out.println("setCar");
        this.car = car;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        System.out.println("setName");
        this.name = name;
    }
    public int getAge() {
        return age;
    }
}
```

```
@Value("18")
public void setAge(int age) {
    System.out.println("setAge");
    this.age = age;
}
@Override
public String toString() {
    return "User [name=" + name + ",
age=" + age + ", car=" + car + "]";
}
}
```

@Resource

对于引用类型直接指定注入哪个名称的对象

```
@Repository("user")
@Scope(scopeName="singleton")
public class User {
    @Value("张三")
    private String name;
    private int age;
    @Resource(name="car")
    private Car car;
    public User() {
        super();
    }
    public Car getCar() {
        return car;
    }
    public void setCar(Car car) {
        System.out.println("setCar");
        this.car = car;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        System.out.println("setName");
        this.name = name;
    }
    public int getAge() {
        return age;
    }
}
```

```
@Value("18")
public void setAge(int age) {
    System.out.println("setAge");
    this.age = age;
}
@Override
public String toString() {
    return "User [name=" + name + ",
age=" + age + ", car=" + car + "]";
}
}
```

声明周期方法

注意需要设置容器的close才能看到销毁操作,并且只支持singleton的对象

- @PostConstruct放在初始化方法上面
- @PreDestroy放在销毁的方法上面

```
@PostConstruct
public void init(){
    System.out.println("构造方法调用后调用");
}
@PreDestroy
public void destroy(){
    System.out.println("容器关闭前销毁");
}
```

Spring中的JUnit4测试

- 导包4+3+1(spring包中的test包)
- 在测试的类上方添加注解
`@RunWith(SpringJUnit4ClassRunner.class)` 表示让SpringJUnit4ClassRunner这个类帮我们创建容器
- 然后在类上添加

```
//配置相关配置文件的路径
@Configuration("classpath:com/zhiyou100/annotation/applicationContext.xml")
```

- 在需要获取容器中的对象的时候设置为成员变量,使用Resource或者Autowired进行获取
- 正常使用test进行测试

```
//使用这个类帮助我们去创建容器
@RunWith(SpringJUnit4ClassRunner.class)
//配置配置文件的路径,必须要写classpath
@ContextConfiguration("classpath:com/zhiyou100/annotation/applicationContext.xml")
public class TestDemo {
    @Resource(name="user")
    private User u;
    @Test
    public void test01(){
        System.out.println(u);
    }
}
```

aop(面向切面编程)

Aspect Oriented Programming是一种编程思想,进行重复的代码进行横向的抽取.

Spring中两种代理模式

动态代理


```

@Test
public void dynamicProxy(){
    UserService us = new UserServiceImpl();
    UserService proxy = (UserService)Proxy.newProxyInstance(UserServiceImpl.class.getClassLoader(), UserServiceImpl.class.getInterfaces(), new InvocationHandler() {
        @Override
        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
            System.out.println("开启事务");
            Object returnObject = method.invoke(us, args);
            System.out.println("关闭事务");
            return returnObject;
        }
    });
    proxy.addUser();
}

```

cglib代理

```

public class CGLibProxyTest implements MethodInterceptor{
    public Object cglibProxy(){
        //创建生成代理工具类
        Enhancer eh = new Enhancer();
        //设置代理对象的父类
        eh.setSuperclass(UserServiceImpl.class);
        //调用方法的时候进行回调
        eh.setCallback(this);
        //创建代理对象
        Object usProxy = eh.create();
        return usProxy;
    }
    //proxyObj代理对象
    //method原始方法
    //arg2方法参数
    //methodProxy代理方法
    @Override
    public Object intercept(Object proxyObj, Method method, Object[] arg2, MethodProxy methodProxy) throws Throwable {
        System.out.println("开启事务");
        Object returnObj = methodProxy.invokeSuper(proxyObj, arg2);
        System.out.println("关闭事务");
        return returnObj;
    }
}

```

aop中的名词解释

1. Joinpoint(连接点)

目标对象中所有可以增强的方法叫做连接点

2. Pointcut(切入点)

目标对象中要增强的方法

3. Advice(通知/增强)

增强的代码

4. Target(目标对象)

被代理对象

5. Weaving(织入)

将通知应用到连接点的过程

6. Proxy(代理)

生成的代理对象

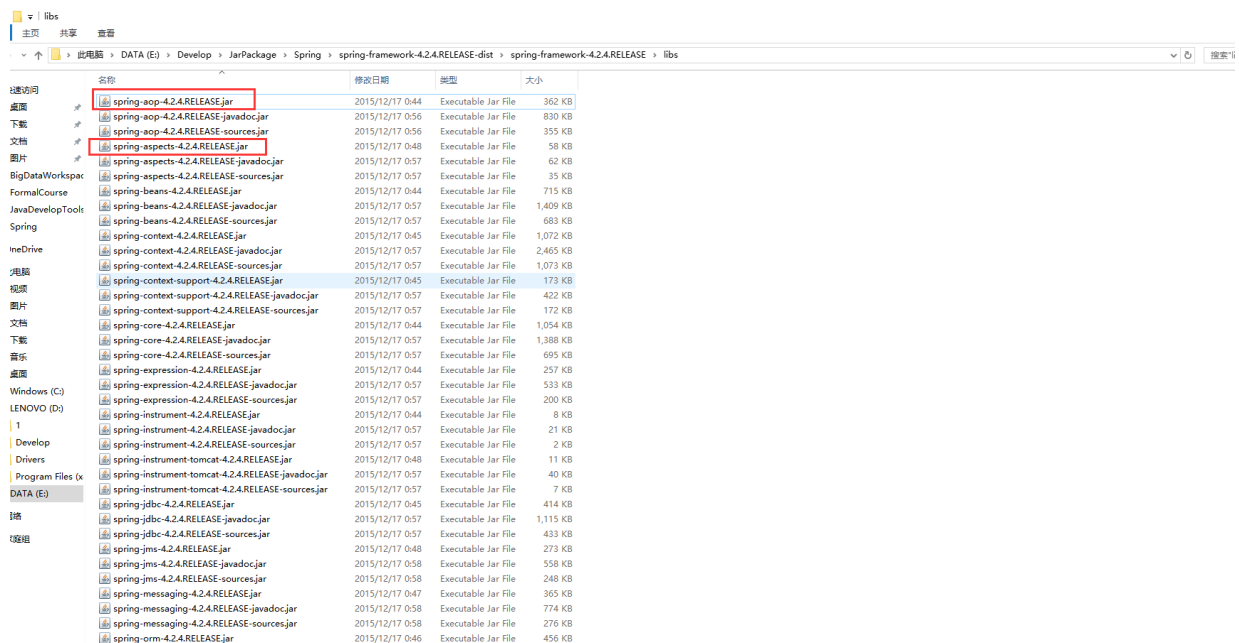
7. Aspect(切面)

切入点+通知就形成了切面

aop.xml配置

导包

导包4+2+1(spring包中的aop包)+1(spring中的aspect包)+1(spring依赖包中aopalliance)+1(spring依赖包中org.spectj包中的weaver包)



	名称	修改日期	类型	大小
快速访问				
桌面	spring-aop-4.2.4.RELEASE.jar	2015/12/17 0:44	Executable Jar File	362 KB
下载	spring-aop-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:56	Executable Jar File	830 KB
文档	spring-aop-4.2.4.RELEASE-sources.jar	2015/12/17 0:56	Executable Jar File	355 KB
图片	spring-aspects-4.2.4.RELEASE.jar	2015/12/17 0:48	Executable Jar File	58 KB
	spring-aspects-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	62 KB
BigDataWorkspac	spring-aspects-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	35 KB
FormalCourse	spring-beans-4.2.4.RELEASE.jar	2015/12/17 0:44	Executable Jar File	715 KB
JavaDevelopTool	spring-beans-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	1,409 KB
Spring	spring-beans-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	683 KB
meDrive	spring-context-4.2.4.RELEASE.jar	2015/12/17 0:45	Executable Jar File	1,072 KB
计算机	spring-context-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	2,465 KB
视频	spring-context-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	1,073 KB
图片	spring-context-support-4.2.4.RELEASE.jar	2015/12/17 0:45	Executable Jar File	173 KB
文档	spring-context-support-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	422 KB
下载	spring-context-support-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	172 KB
音乐	spring-core-4.2.4.RELEASE.jar	2015/12/17 0:44	Executable Jar File	1,054 KB
	spring-core-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	1,388 KB
桌面	spring-core-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	695 KB
音乐	spring-expression-4.2.4.RELEASE.jar	2015/12/17 0:44	Executable Jar File	257 KB
	spring-expression-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	533 KB
Windows (C)	spring-expression-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	200 KB
LENOVO (D)	spring-instrument-4.2.4.RELEASE.jar	2015/12/17 0:44	Executable Jar File	8 KB
1	spring-instrument-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	21 KB
Develop	spring-instrument-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	2 KB
Drivers	spring-instrument-tomcat-4.2.4.RELEASE.jar	2015/12/17 0:48	Executable Jar File	11 KB
Program Files (x	spring-instrument-tomcat-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	40 KB
DATA (E)	spring-instrument-tomcat-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	7 KB
	spring-jdbc-4.2.4.RELEASE.jar	2015/12/17 0:45	Executable Jar File	414 KB
网络	spring-jdbc-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:57	Executable Jar File	1,115 KB
虚拟机	spring-jdbc-4.2.4.RELEASE-sources.jar	2015/12/17 0:57	Executable Jar File	433 KB
	spring-jms-4.2.4.RELEASE.jar	2015/12/17 0:48	Executable Jar File	273 KB
	spring-jms-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:58	Executable Jar File	558 KB
	spring-jms-4.2.4.RELEASE-sources.jar	2015/12/17 0:58	Executable Jar File	248 KB
	spring-messaging-4.2.4.RELEASE.jar	2015/12/17 0:47	Executable Jar File	365 KB
	spring-messaging-4.2.4.RELEASE-javadoc.jar	2015/12/17 0:58	Executable Jar File	774 KB
	spring-messaging-4.2.4.RELEASE-sources.jar	2015/12/17 0:58	Executable Jar File	276 KB
	spring-orm-4.2.4.RELEASE.jar	2015/12/17 0:46	Executable Jar File	456 KB

桌面	java.xml.soap	2017/6/26 9:44	文件类
下载	java.xml.stream	2017/6/26 9:22	文件类
文档	java.xml.ws	2017/6/26 9:22	文件类
图片	net.sourceforge.cgitb	2017/6/26 9:22	文件类
BigDataWorkspac	net.sourceforge.ehcache	2017/6/26 9:22	文件类
FormalCourse	net.sourceforge.iso-relax	2017/6/26 9:22	文件类
JavaDevelopTool	net.sourceforge.jasperreports	2017/6/26 9:22	文件类
Spring	net.sourceforge.jseclipse	2017/6/26 9:22	文件类
	net.sourceforge.jxb	2017/6/26 9:22	文件类
	net.sourceforge.serp	2017/6/26 9:22	文件类
OneDrive	net.sourceforge.xbhtml	2017/6/26 9:22	文件类
	org.antlr	2017/6/26 9:22	文件类
此电脑	org.apache.lance	2017/6/26 9:22	文件类
视频	org.apache.axis	2017/6/26 9:22	文件类
图片	org.apache.beol	2017/6/26 9:22	文件类
文档	org.apache.catalina	2017/6/26 9:22	文件类
下载	org.apache.commons	2017/6/26 9:22	文件类
音乐	org.apache.coyote	2017/6/26 9:22	文件类
桌面	org.apache.derby	2017/6/26 9:22	文件类
Windows (C:)	org.apache.batis	2017/6/26 9:22	文件类
LENOVO (D:)	org.apache.juli	2017/6/26 9:22	文件类
1	org.apache.log4j	2017/6/26 9:22	文件类
Develop	org.apache.openp	2017/6/26 9:22	文件类
Drivers	org.apache.poi	2017/6/26 9:22	文件类
Program Files (x	org.apache.regex	2017/6/26 9:22	文件类
DATA (E:)	org.apache.struts	2017/6/26 9:22	文件类
	org.apache.taglib	2017/6/26 9:22	文件类
网络	org.apache.tiles	2017/6/26 9:22	文件类
家庭组	org.apache.velocity	2017/6/26 9:22	文件类
	org.apache.xerces	2017/6/26 9:22	文件类
	org.apache.xml	2017/6/26 9:22	文件类
	org.apache.xmlbeans	2017/6/26 9:22	文件类
	org.apache.xmlcommons	2017/6/26 9:22	文件类
	org.aspectj	2017/6/26 9:22	文件类
	org.beanshell	2017/6/26 9:22	文件类
	org.codehaus.crastr	2017/6/26 9:22	文件类

编写通知代码

在spring中通知代码是通过方法的功能来实现的,需要我们写一个类将需要的通知写在不同的方法中

```
public class MyAdvice {  
    /*  
    * 1.前置通知  
    * 2.后置通知(出现异常不会调用)  
    * 3.环绕通知  
    * 4.异常拦截通知  
    * 5.后置通知(无论是否出现异常都会调用)  
    */  
    public void beforeAdvice(){  
        System.out.println("前置通知");  
    }  
    public void afterInterruptedException  
Advice(){  
        System.out.println("后置通知,会被异  
常阻断");  
    }  
    public void aroundAdvice(ProceedingJo  
inPoint pjp) throws Throwable{  
        System.out.println("环绕前通知");  
        pjp.proceed();  
  
        System.out.println("环绕后通知");  
    }  
    public void ExceptionAdvice(){  
        System.out.println("异常拦截通知");  
    }  
    public void afterWithoutExceptionAdvi  
ce(){
```

```
        System.out.println("后置通知,不会被  
异常阻断");  
    }  
  
}
```

spring中通知的类型

- 前置通知:目标方法之前进行调用
- 后置通知(出现异常不会调用):目标方法之后调用
- 环绕通知:目标方法前后调用
- 异常拦截通知:出现异常的时候调用
- 后置通知(无论是否出现异常都会调用):目标方法之后调用

编写目标对象

```
public class UserServiceImpl implements U
serService {
    @Override
    public void addUser() {
        System.out.println("添加用户");
    }
    @Override
    public void deleteUser() {
        System.out.println("删除用户");
    }
    @Override
    public void updateUser() {
        System.out.println("修改用户");
    }
    @Override
    public void findUser() {
        System.out.println("查询用户");
    }
}
```

进行文件配置


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/200
1/XMLSchema-instance"
    xmlns="http://www.springframework.or
g/schema/beans"
    xmlns:context="http://www.springframe
work.org/schema/context"
    xmlns:aop="http://www.springframewor
k.org/schema/aop"
    xsi:schemaLocation="http://www.spring
framework.org/schema/beans http://www.spr
ingframework.org/schema/beans/spring-bean
s-4.2.xsd
    http://www.springframework.org/sc
hema/context http://www.springframework.o
rg/schema/context/spring-context-4.2.xsd
    http://www.springframework.org/sc
hema/aop http://www.springframework.org/s
chema/aop/spring-aop-4.2.xsd">

<context:component-scan base-package="co
m.zhiyou100.aop"></context:component-sca
n>

<bean name="userServiceTarget" class="co
m.zhiyou100.service.impl.UserServiceImp
l"></bean>

<bean name="advice" class="com.zhiyou10
0.aop.MyAdvice"></bean>
```

```
<aop:config>
```

```
<!--
```

配置切入点

id表示给切入点起的名字

```
expression是表达式 格式为execution()
```

切入点表达式

表示具体方法

```
public void com.zhiyou100.service.impl.UserServiceImpl.addUser()
```

修饰符可以不写

```
void com.zhiyou100.service.impl.UserServiceImpl.addUser()
```

返回值类型不限

```
* com.zhiyou100.service.impl.UserServiceImpl.addUser()
```

方法名称不限

```
* com.zhiyou100.service.impl.UserServiceImpl.*()
```

方法参数不限制

```
* com.zhiyou100.service.impl.UserServiceImpl.*(..)
```

方法参数不限制

```
* com.zhiyou100.service.impl.UserServiceImpl.*(..)
```

以ServiceImpl结尾的

```
* com.zhiyou100.service.impl.*ServiceImpl.*(..)
```

包括impl的子包中符合条件的

```
* com.zhiyou100.service.impl
```

```
l..*ServiceImpl.*(..)
-->
<aop:pointcut expression="execution(*
com.zhiyou100.service..*ServiceImpl.*
(..))" id="pct"/>
<!--
    配置切面,ref表示通知
    pointcut-ref表示切入点
    aop:before 前置通知
    aop:after-returning后置通知,出现异
常会被阻断
    aop:around环绕通知
    aop:after-throwing异常拦截通知
    aop:after后置通知,忽略异常
-->
<aop:aspect ref="advice">
    <aop:before method="beforeAdvice"
pointcut-ref="pct"/>
    <!-- 出现异常不会调用 -->
    <aop:after-returning method="afte
rInterruptedExceptionAdvice" pointcut-re
f="pct"/>
    <aop:around method="aroundAdvice"
pointcut-ref="pct"/>
    <aop:after-throwing method="Excep
tionAdvice" pointcut-ref="pct"/>
    <aop:after method="afterWithoutEx
ceptionAdvice" pointcut-ref="pct"/>
</aop:aspect>
</aop:config>
```

```
</beans>
```

aop-注解配置

- 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/200
1/XMLSchema-instance"
    xmlns="http://www.springframework.or
g/schema/beans"
    xmlns:context="http://www.springframe
work.org/schema/context"
    xmlns:aop="http://www.springframewor
k.org/schema/aop"
    xsi:schemaLocation="http://www.spring
framework.org/schema/beans http://www.spr
ingframework.org/schema/beans/spring-bean
s-4.2.xsd
        http://www.springframework.org/sc
hema/context http://www.springframework.o
rg/schema/context/spring-context-4.2.xsd
        http://www.springframework.org/sc
hema/aop http://www.springframework.org/s
chema/aop/spring-aop-4.2.xsd">

<bean name="userServiceTarget" class="co
m.zhiyou100.service.impl.UserServiceImp
l"></bean>

<bean name="advice" class="com.zhiyou10
0.annotation_aop.MyAdvice"></bean>
<!--开启注解完成织入操作-->
<aop:aspectj-autoproxy></aop:aspectj-auto
proxy>
</beans>
```

- 在通知类中进行注解

```
//表示该类是一个通知类
@Aspect
public class MyAdvice {
    /*
     * 1.前置通知
     * 2.后置通知(出现异常不会调用)
     * 3.环绕通知
     * 4.异常拦截通知
     * 5.后置通知(无论是否出现异常都会调用)
     */
    @Pointcut("execution(* com.zhiyou100.service.impl.*ServiceImpl.*(..))")
    public void pct(){}
    @Before("MyAdvice.pct()")
    public void beforeAdvice(){
        System.out.println("前置通知");
    }
    @AfterReturning("MyAdvice.pct()")
    public void afterInterruptedExceptionAdvice(){
        System.out.println("后置通知,会被异常阻断");
    }
    @Around("MyAdvice.pct()")
    public void aroundAdvice(ProceedingJoinPoint pjp) throws Throwable{
        System.out.println("环绕前通知");
        pjp.proceed();
    }
}
```

```
        System.out.println("环绕后通知");
    }
    @AfterThrowing("MyAdvice.pct()")
    public void ExceptionAdvice(){
        System.out.println("异常拦截通知");
    }
    @After("MyAdvice.pct()")
    public void afterWithoutExceptionAdvice(){
        System.out.println("后置通知,不会被
异常阻断");
    }
}
```