1、Spring是一个开源框架，为了解决企业应用开发的复杂性而创建的，但现在已经不止应用于企业应用。

1. 是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架

-从大小与开销两方面而言Spring都是轻量的

-通过控制反转（IoC）的技术达到松耦合的目的

-提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务进行内聚性的开发

-包含并管理应用对象的配置和生命周期，这个意义上是一种容器

-将简单的组件配置、组合成为复杂的应用，这个意义上是框架

1. Spring作用
2. 容器
3. 提供了对多种技术的支持
4. AOP（事务管理、日志等）
5. 对主流应用框架（Hibernate等）提供了良好的支持
6. Spring使用范围
7. 构建企业应用（SpringMVC+Spring+Hibernate/ibatis）
8. 单独使用Bean容器（Bean管理）
9. 单独使用AOP进行切面处理
10. 其他的Spring功能，如：对消息的支持等
11. 在互联网中的应用
12. 框架的特点

-半成品

-封装了特定的处理流程和控制逻辑

-成熟的、不断升级改进的软件

框架与类库的区别

-框架一般是封装了逻辑、高内聚、类库则是松散的工具组合

-框架专注于某一领域，类库则是更通用的

**IOC**

接口

1. 用于沟通的中介物的抽象化
2. 实体把自己提供给外界的一种抽象化说明，用以由内部操作分离出外部沟通的方法，使其能被修改内部而不影响外界其他实体与其交互的方式
3. 对应Java接口即声明，声明了哪些方法是对外公开提供的
4. 在java8中，接口可以拥有方法体

面向接口编程

1. 结构设计中，分清层次及调用关系，每层只向外（上层）提供一组功能接口，各层间仅依赖接口而非实现类
2. 接口实现的变动不影响各层间的调用，这一点在公共服务中尤为重要
3. 面向接口编程中的接口是用于隐藏具体实现和实现多态性的组件

**面向接口编程 用接口来进行声明 将接口的实现类赋值给对象的声明。**

**Spring注入**

Spring注入是指在启动 Spring容器加载bean配置的时候，完成对变量的赋值行为

常用的两种注入方式

--设值注入

--构造注入

**Bean的配置项**

ID、Class、Scope、Constructor arguments、Properties、Autowiring mode、lazy-initialization mode、Initialization/destruction method

**Bean的作用域**

singleton:单例，指一个Bean容器中只存在一份

prototype:每次请求（每次使用）创建新的实例，destroy方式不生效

request:每次http请求创建一个实例且仅在当前request内有效

session:同上，每次http请求创建，当前session内有效

global session:基于portlet的web中有效（portlet定义了global session），如果是在web中，同session

**Bean的生命周期**

初始化

-实现org.springframework.beans.factory.InitializingBean接口，覆盖afterPropertiesSet方法

-配置init-method

<bean id=”exampleInitBean” class=”xeamples.ExampleBean” init-method=”init”/>

销毁

-实现org.springframework.beans.factory.DisposableBean接口，覆盖destroy方法

-配置destroy-method

<bean id=”exampleInitBean” class=”examples.ExampleBean” destroy-method=”cleanup”/>

**配置全局默认初始化、销毁方法**

在beans xmlns 中添加 default-init-method=”init” default-destroy-method=”destroy”

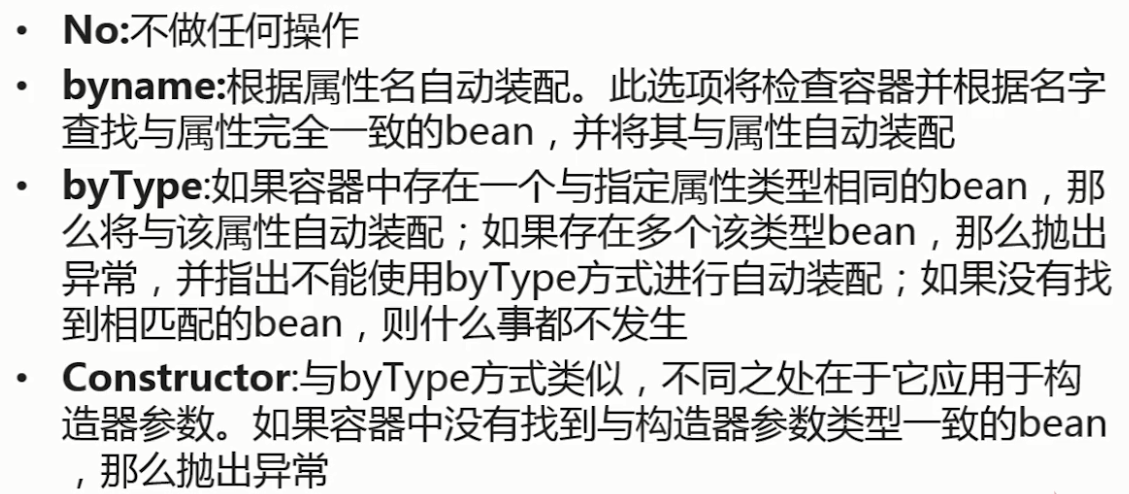
**Aware**

Spring中提供了一些以Aware结尾的接口，实现了Aware接口的bean在被初化之后，可以获取相应资源

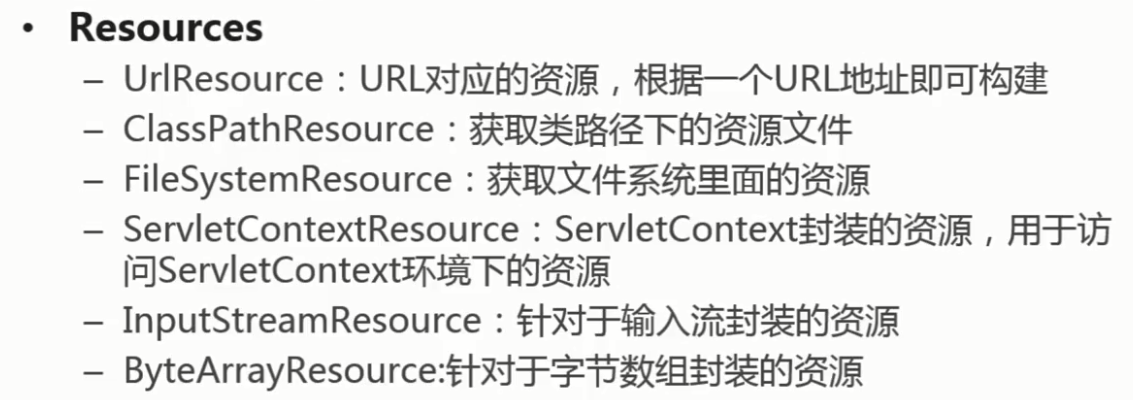
通过Aware接口，可以对Spring相应资源进行操作

为对Spring进行简单的扩展提供了方便的入口

**Bean的自动装配**



**Resources**



ResourcesLoader对Resources进行加载

**Bean管理的注解实现及例子**















AOP

面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统

一维护的一种技术。

主要功能：日志记录，性能统计，安全控制，事务处理，异常处理等。

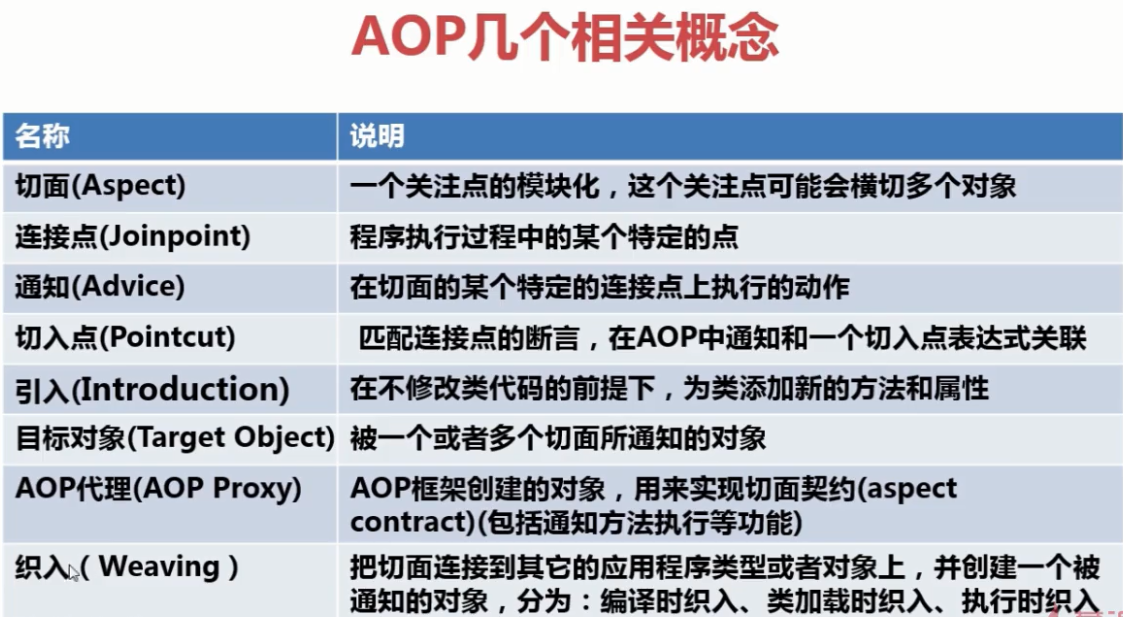
**AOP实现方式**

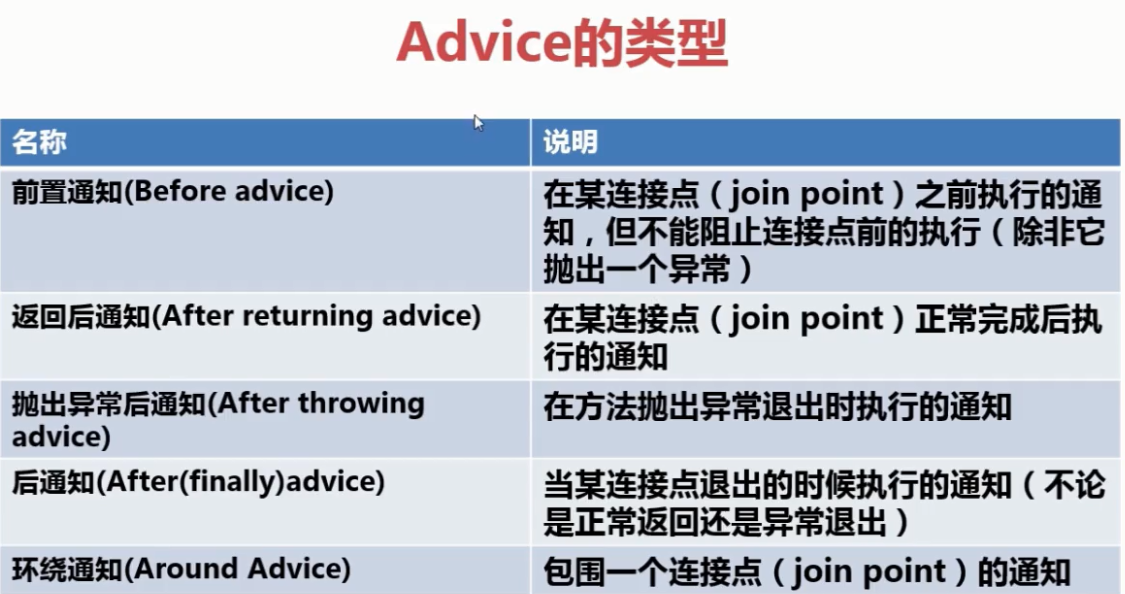
预编译：

AspectJ

运行期动态代理（JDK动态代理、CGLib动态代理）

SpringAOP、JbossAOP

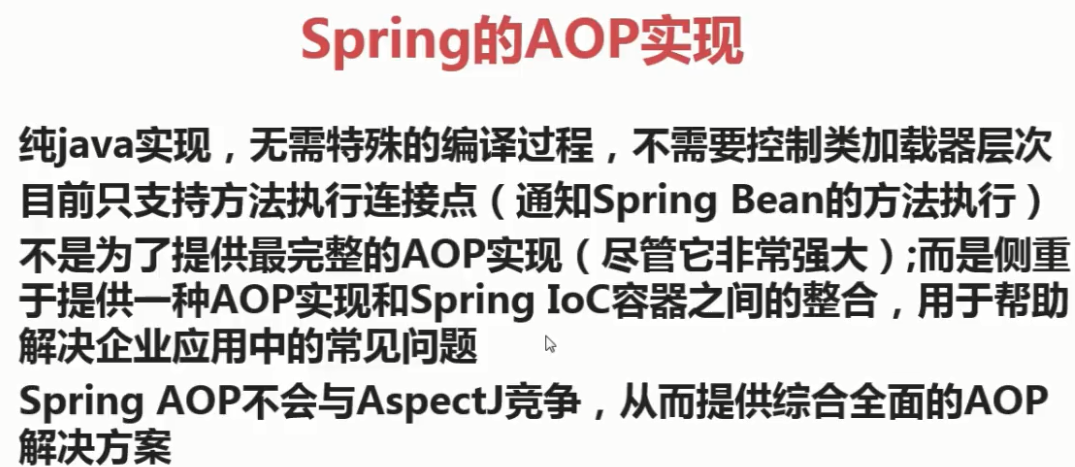


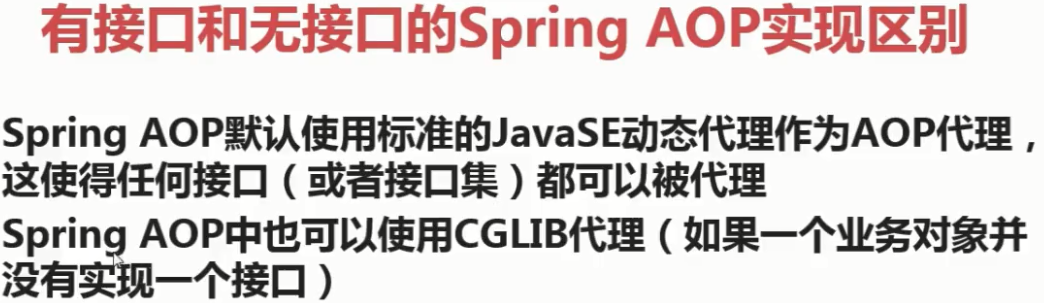


**Spring框架中AOP的用途**

提供了声明式的企业服务，特别是EJB的替代服务的声明

允许用户定制自己的方面，以完成OOP与AOP的互补使用





Schema——based AOP

Spring所有的切面和通知器都必须放在一个<aop:config>内（可以配置包含多个<aop:config>元素），每一个<aop:config>可以包含pointcut, advisor和aspect元素（它们必须按照这个顺序进行声明）

<aop:config>风格的配置大量使用了Spring的自动代理机制

**pointcut**



**Introductions**

简介允许一个切面声明一个实现指定接口的通知对象，并且提供了一个接口实现类来代表这些对象

由<aop:aspect>中的<aop:declare-parents>元素声明该元素用于声明所匹配的类型拥有一个新的parent（因此得名）

**Advisors**

advisors就像一个小的自包含的方面，只有一个advice

切面自身通过一个bean表示，并且必须实现某个advice接口，同时，advisor也可以很好的利用AspectJ的切入点表达式

Spring通过配置文件中<aop:advisor>元素支持advisor实际使用中，大多数情况下它会和transactional advice配合使用

为了定义一个advisor的优先级以便让advice可以有序，可以使用order属性来定义advisor的顺序

**Spring AOP API**

Introduction advice

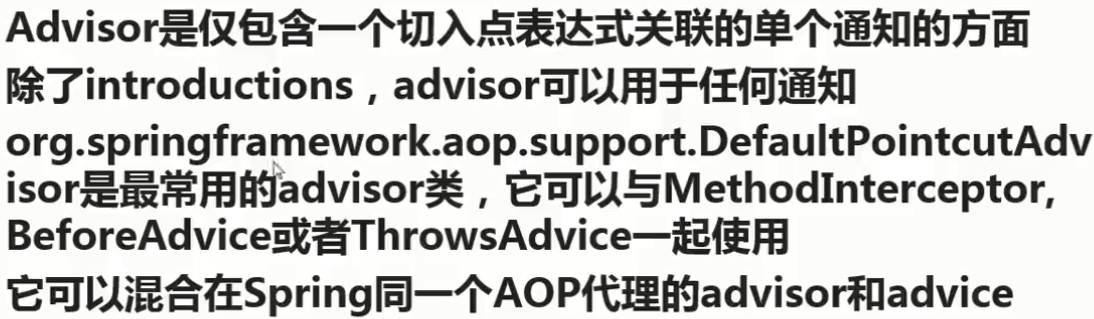
Spring把引入通知作为一种特殊的拦截通知

需要IntroductionAdvisor和IntroductionInterceptor

仅适用于类，不能和任何切入点一起使用



Introduction advisor比较简单，持有独立的LockMixin实例



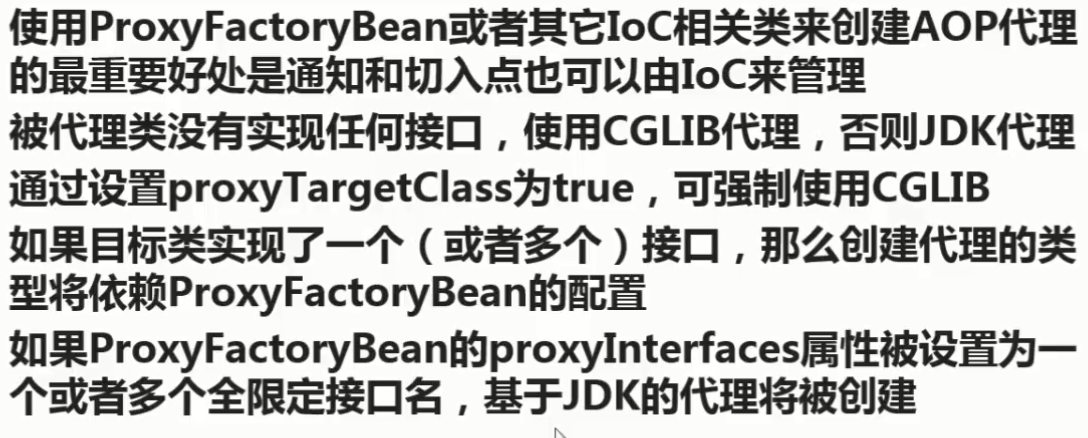
ProxyFactoryBean

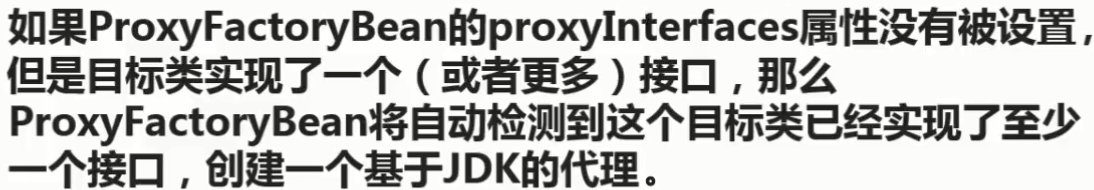
创建Spring AOP 代理的基本方法是使用

org.springframework.aop.framework.ProxyFactoryBean

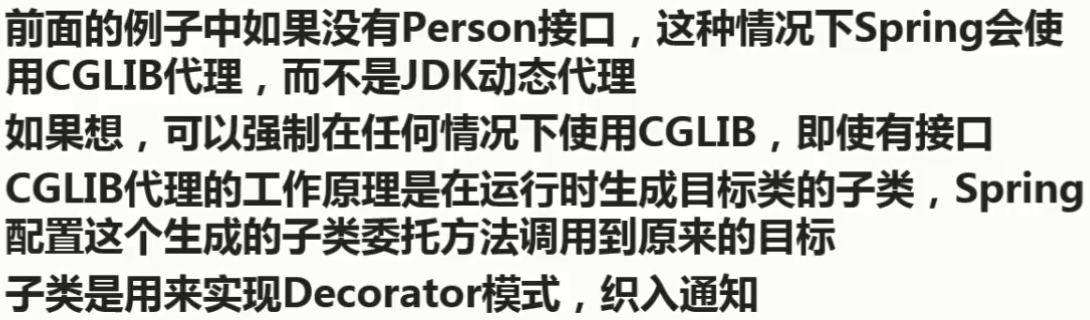
这可以完全控制切入点和通知（advice）以及他们的顺序

**创建一个bean id为foo的ProxyFactoryBean，引用foo这个对象看到的不是ProxyFactoryBean本身，而是这个类getObject（）方法创建的对象。GetObject方法将创建一个AOP代理包装一个目标对象。**





**Proxying classes**



CGLIB的代理对用户是透明的，需要注意：

——final方法不能被通知，因为它们不能被覆盖

——不用把CGLIB添加到classpath中

使用global advisors

用\*做通配，匹配所有拦截器加入通知链

简化的proxy定义

使用父子bean定义，以及内部bean定义，可能会带来更清洁和更简洁的代理定义（抽象属性标记父bean定义为抽象的这样它不能被实例化）

**AspectJ**

@AspectJ的风格类似纯java注解的普通java类

Spring可以使用AspectJ来做切入点解析

AOP的运行时仍旧是纯的Spring AOP，对AspectJ的编译器或者织入无依赖性

@AspectJ切面使用@Aspect注解配置，拥有@Aspect的任何bean将被Spring自动识别并应用

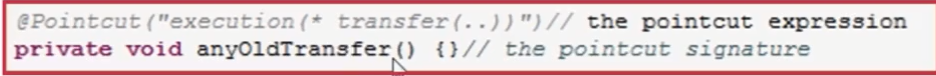
用@Aspect注解的类可以有方法和字段，他们也可能包括切入点（pointcut），通知（Advice）和引入（introduction）声明

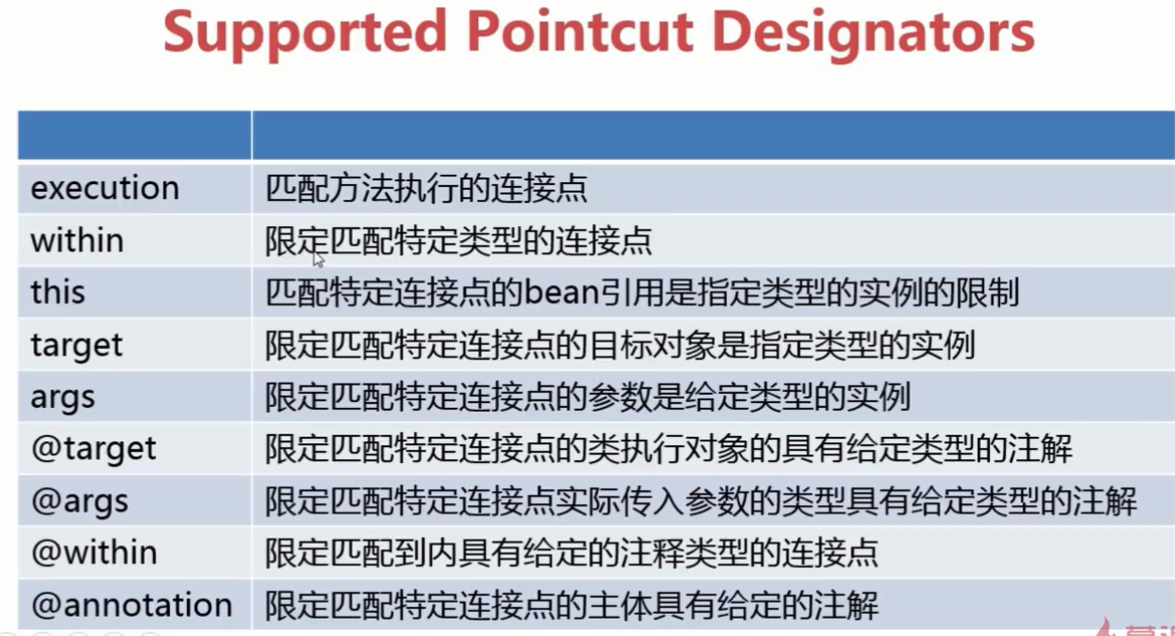
@Aspect注解是不能够通过类路径自动检测发现的，所以需要配合使用@Component注释或者在xml配置bean

**pointcut**

一个切入短通过一个普通的方法定义来提供，并且切入点表达式使用@Pointcut注解，方法返回类型必须为void

定义一个名为‘anyOldTransfer’，这个切入点将匹配任何名为“transfer”的方法的执行





组合pointcut

切入点表达式可以通过&&、||和！进行组合，也可以通过名字引入切点表达式

通过组合，可以建立更加复杂的切入点表达式

After（finally）advice

最终通知必须准备处理正常和异常两种返回情况，它通常用于释放资源

Around advice

环绕通知使用@Around注解来声明，通知方法的第一个参数必须是ProceedingJoinPoint类型

在通知内部调用ProceedingJoinPoint的proceed（）方法会导致执行真正的方法，传入一个Object[]对象，数组中的值将被作为参数传递给方法