**SpringAOP：**

AOP是Spring提供的关键特性之一。AOP即面向切面编程，是OOP编程的有效补充。使用AOP技术，可以将一些系统性相关的编程工作，独立提取出来，独立实现，然后通过切面切入进系统。从而避免了在业务逻辑的代码中混入很多的系统相关的逻辑——比如权限管理，事物管理，日志记录等等。

**AOP相关概念：**

**Aspect（切面）：** Aspect 声明类似于 Java 中的类声明，在 Aspect 中会包含着一些 Pointcut 以及相应的 Advice。

**Joint point（连接点）：**表示在程序中明确定义的点，典型的包括方法调用，对类成员的访问以及异常处理程序块的执行等等，它自身还可以嵌套其它 joint point。

**Pointcut（切点）：**表示一组 joint point，这些 joint point 或是通过逻辑关系组合起来，或是通过通配、正则表达式等方式集中起来，它定义了相应的 Advice 将要发生的地方。

**Advice（增强）：**Advice 定义了在 Pointcut 里面定义的程序点具体要做的操作，它通过 before、after 和 around 来区别是在每个 joint point 之前、之后还是代替执行的代码。

Target（目标对象）：织入 Advice 的目标对象.。

**Weaving（织入）：**将 Aspect 和其他对象连接起来, 并创建 Adviced object 的过程，就是切入程序的意思

**AOP实现：**

在Spring中，通过**动态代理**和**动态字节码技术**实现了AOP，动态代理需要实现接口，没有接口的类就需要用到cglib来进行操作。

CGLIB采用动态创建子类的方法，对于final修饰的方法无法进行代理

**Spring中有哪些不同的通知类型：**

**前置通知(Before Advice):** 在连接点之前执行的Advice，不过除非它抛出异常，否则没有能力中断执行流。使用 @Before 注解使用这个Advice。

**返回之后通知(After Retuning Advice):** 在连接点正常结束之后执行的Advice。例如，如果一个方法没有抛出异常正常返回。通过 @AfterReturning 关注使用它。

**抛出（异常）后执行通知(After Throwing Advice):** 如果一个方法通过抛出异常来退出的话，这个Advice就会被执行。通

用 @AfterThrowing 注解来使用。

**后置通知(After Advice):** 无论连接点是通过什么方式退出的(正常返回或者抛出异常)都会执行在结束后执行这些Advice。通过 @After 注解使用。

**围绕通知(Around Advice):** 围绕连接点执行的Advice，就你一个方法调用。这是最强大的Advice。通过 @Around 注解使用。