谈谈对Spring AOP的理解

题目标签

学习时长：20分钟

题目难度：中等

知识点标签：Spring AOP的理解

题目描述

谈谈对Spring AOP的理解

面试题分析

将AOP的几个核心概念说出来，将AOP的实现方式说出来

AOP

AOP（Aspect Oriented Programming），即面向切面编程，可以说是OOP（Object Oriented Programming，面向对象编程）的补充和完善。OOP引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过OOP允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切（cross cutting），在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。

AOP技术恰恰相反，它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。

使用"横切"技术，AOP把软件系统分为两个部分：**核心关注点**和**横切关注点**。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。

AOP核心概念

1、横切关注点

对哪些方法进行拦截，拦截后怎么处理，这些关注点称之为横切关注点

2、切面（aspect）

类是对物体特征的抽象，切面就是对横切关注点的抽象

3、连接点（joinpoint）

被拦截到的点，因为Spring只支持方法类型的连接点，所以在Spring中连接点指的就是被拦截到的方法，实际上连接点还可以是字段或者构造器

4、切入点（pointcut）

对连接点进行拦截的定义

5、通知（advice）

所谓通知指的就是指拦截到连接点之后要执行的代码，通知分为前置、后置、异常、最终、环绕通知五类

6、目标对象

代理的目标对象

7、织入（weave）

将切面应用到目标对象并导致代理对象创建的过程

8、引入（introduction）

在不修改代码的前提下，引入可以在**运行期**为类动态地添加一些方法或字段

Spring对AOP的支持

Spring中AOP代理由Spring的IOC容器负责生成、管理，其依赖关系也由IOC容器负责管理。因此，AOP代理可以直接使用容器中的其它bean实例作为目标，这种关系可由IOC容器的依赖注入提供。Spring创建代理的规则为：

1、**默认使用Java动态代理来创建AOP代理**，这样就可以为任何接口实例创建代理了

2、**当需要代理的类不是代理接口的时候，Spring会切换为使用CGLIB代理**，也可强制使用CGLIB

AOP编程其实是很简单的事情，纵观AOP编程，程序员只需要参与三个部分：

1、定义普通业务组件

2、定义切入点，一个切入点可能横切多个业务组件

3、定义增强处理，增强处理就是在AOP框架为普通业务组件织入的处理动作

所以进行AOP编程的关键就是定义切入点和定义增强处理，一旦定义了合适的切入点和增强处理，AOP框架将自动生成AOP代理，即：**代理对象的方法=增强处理+被代理对象**的方法。

下面给出一个Spring AOP的.xml文件模板，名字叫做aop.xml，之后的内容都在aop.xml上进行扩展：

```
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.2.xsd">

</beans>
```

基于Spring的AOP简单实现

注意一下，在讲解之前，说明一点：使用Spring AOP，要成功运行起代码，只用Spring提供给开发者的jar包是不够的，请额外上网下载两个jar包：

1、aopalliance.jar

2、aspectjweaver.jar

开始讲解用Spring AOP的XML实现方式，先定义一个接口：

```
public interface HelloWorld
{
    void printHelloWorld();
    void doPrint();
}
```

定义两个接口实现类：

```
public class HelloWorldImpl1 implements HelloWorld
{
    public void printHelloWorld()
    {
        System.out.println("Enter HelloWorldImpl1.printHelloWorld()");
    }

    public void doPrint()
    {
        System.out.println("Enter HelloWorldImpl1.doPrint()");
        return ;
    }
}
```



```
public class HelloWorldImpl2 implements HelloWorld
{
    public void printHelloWorld()
    {
        System.out.println("Enter HelloWorldImpl2.printHelloWorld()");
    }

    public void doPrint()
    {
        System.out.println("Enter HelloWorldImpl2.doPrint()");
        return ;
    }
}
```

横切关注点，这里是打印时间：

```
public class TimeHandler
{
    public void printTime()
    {
        System.out.println("CurrentTime = " + System.currentTimeMillis());
    }
}
```

有这三个类就可以实现一个简单的Spring AOP了，看一下aop.xml的配置：

```
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.2.xsd">

        <bean id="helloWorldImpl1" class="com.xrq.aop.HelloWorldImpl1" />
        <bean id="helloWorldImpl2" class="com.xrq.aop.HelloWorldImpl2" />
        <bean id="timeHandler" class="com.xrq.aop.TimeHandler" />

        <aop:config>
            <aop:aspect id="time" ref="timeHandler">
                <aop:pointcut id="addAllMethod" expression="execution(*
com.xrq.aop.HelloWorld.*(..))" />
                <aop:before method="printTime" pointcut-ref="addAllMethod" />
                <aop:after method="printTime" pointcut-ref="addAllMethod" />
            </aop:aspect>
        </aop:config>
</beans>
```

写一个main函数调用一下：

```
public static void main(String[] args)
{
    ApplicationContext ctx =
            new ClassPathXmlApplicationContext("aop.xml");

    HelloWorld hw1 = (HelloWorld)ctx.getBean("helloWorldImpl1");
    HelloWorld hw2 = (HelloWorld)ctx.getBean("helloWorldImpl2");
    hw1.printHelloWorld();
    System.out.println();
    hw1.doPrint();

    System.out.println();
    hw2.printHelloWorld();
    System.out.println();
    hw2.doPrint();
}
```


运行结果为：

```
CurrentTime = 1446129611993
Enter HelloWorldImpl1.printHelloWorld()
CurrentTime = 1446129611993

CurrentTime = 1446129611994
Enter HelloWorldImpl1.doPrint()
CurrentTime = 1446129611994

CurrentTime = 1446129611994
Enter HelloWorldImpl2.printHelloWorld()
CurrentTime = 1446129611994
```

```
CurrentTime = 1446129611994
Enter HelloWorldImpl2.doPrint()
CurrentTime = 1446129611994
```

看到给HelloWorld接口的两个实现类的所有方法都加上了代理，代理内容就是打印时间

基于Spring的AOP使用其他细节

1、增加一个横切关注点，打印日志，Java类为：

```
public class LogHandler
{
    public void LogBefore()
    {
        System.out.println("Log before method");
    }

    public void LogAfter()
    {
        System.out.println("Log after method");
    }
}
```

```
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.2.xsd">

        <bean id="helloWorldImpl1" class="com.xrq.aop.HelloWorldImpl1" />
        <bean id="helloWorldImpl2" class="com.xrq.aop.HelloWorldImpl2" />
        <bean id="timeHandler" class="com.xrq.aop.TimeHandler" />
        <bean id="logHandler" class="com.xrq.aop.LogHandler" />

        <aop:config>
            <aop:aspect id="time" ref="timeHandler" order="1">
                <aop:pointcut id="addTime" expression="execution(*
com.xrq.aop.HelloWorld.*(..))" />
                <aop:before method="printTime" pointcut-ref="addTime" />
                <aop:after method="printTime" pointcut-ref="addTime" />
            </aop:aspect>
            <aop:aspect id="log" ref="logHandler" order="2">
                <aop:pointcut id="printLog" expression="execution(*
com.xrq.aop.HelloWorld.*(..))" />
                <aop:before method="LogBefore" pointcut-ref="printLog" />
                <aop:after method="LogAfter" pointcut-ref="printLog" />
            </aop:aspect>
        </aop:config>
</beans>
```


测试类不变，打印结果为：

```
CurrentTime = 1446130273734
Log before method
Enter HelloWorldImpl1.printHelloWorld()
Log after method
CurrentTime = 1446130273735

CurrentTime = 1446130273736
Log before method
Enter HelloWorldImpl1.doPrint()
Log after method
CurrentTime = 1446130273736

CurrentTime = 1446130273736
Log before method
Enter HelloWorldImpl2.printHelloWorld()
Log after method
CurrentTime = 1446130273736

CurrentTime = 1446130273737
Log before method
Enter HelloWorldImpl2.doPrint()
Log after method
CurrentTime = 1446130273737
```

要想让logHandler在timeHandler前使用有两个办法：

（1）aspect里面有一个order属性，order属性的数字就是横切关注点的顺序

（2）把logHandler定义在timeHandler前面，Spring默认以aspect的定义顺序作为织入顺序

2、我只想织入接口中的某些方法

修改一下pointcut的expression就好了：

```
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.2.xsd">

        <bean id="helloWorldImpl1" class="com.xrq.aop.HelloWorldImpl1" />
        <bean id="helloWorldImpl2" class="com.xrq.aop.HelloWorldImpl2" />
        <bean id="timeHandler" class="com.xrq.aop.TimeHandler" />
        <bean id="logHandler" class="com.xrq.aop.LogHandler" />

        <aop:config>
            <aop:aspect id="time" ref="timeHandler" order="1">
                <aop:pointcut id="addTime" expression="execution(*
com.xrq.aop.HelloWorld.print*(..))" />
                <aop:before method="printTime" pointcut-ref="addTime" />
                <aop:after method="printTime" pointcut-ref="addTime" />
            </aop:aspect>
            <aop:aspect id="log" ref="logHandler" order="2">
```

```
                <aop:pointcut id="printLog" expression="execution(*
com.xrq.aop.HelloWorld.do*(..))" />
                <aop:before method="LogBefore" pointcut-ref="printLog" />
                <aop:after method="LogAfter" pointcut-ref="printLog" />
            </aop:aspect>
        </aop:config>
</beans>
```

表示timeHandler只会织入HelloWorld接口print开头的方法，logHandler只会织入HelloWorld接口do开头的方法

3、强制使用CGLIB生成代理

前面说过Spring使用动态代理或是CGLIB生成代理是有规则的，高版本的Spring会自动选择是使用动态代理还是CGLIB生成代理内容，当然我们也可以强制使用CGLIB生成代理，那就是aop:config里面有一个"proxy-target-class"属性，这个属性值如果被设置为true，那么基于类的代理将起作用，如果proxy-target-class被设置为false或者这个属性被省略，那么基于接口的代理将起作用

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理：

①JDK动态代理只提供接口的代理，不支持类的代理。核心InvocationHandler接口和Proxy类，InvocationHandler 通过invoke()方法反射来调用目标类中的代码，动态地将横切逻辑和业务编织在一起；接着，Proxy利用 InvocationHandler动态创建一个符合某一接口的的实例, 生成目标类的代理对象。

②如果代理类没有实现 InvocationHandler 接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成指定类的一个子类对象，并覆盖其中特定方法并添加增强代码，从而实现AOP。CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的。

静态代理与动态代理区别在于生成AOP代理对象的时机不同，相对来说AspectJ的静态代理方式具有更好的性能，但是AspectJ需要特定的编译器进行处理，而Spring AOP则无需特定的编译器处理。