Spring笔记

# 第一章 基础知识

## 一 IoC

控制反转（IoC，Inversion of Control）：控制反转就是对对象控制权的转移，从程序代码本身反转到了外部容器。通过容器实现对象的创建，属性赋值，依赖的管理。减少了对象之间的依赖耦合。

IoC 是一个概念，是一种思想，其实现方式多种多样。当前比较流行的实现方式是依赖注入。

依赖注入(DI，Dependency Injection)：指程序运行过程中，若需要调用另一个对象协助时，无须在代码中创建被调用者，而是依赖于外部容器，由外部容器创建后传递给程序。

Spring 框架使用依赖注入（DI）实现 IoC。

## 二 AOP

### 1、概念

AOP（Aspect Orient Programming），面向切面编程。面向切面编程是通过运行期动态代理实现程序功能的统一维护的一种技术。可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

AOP 底层就是采用动态代理模式实现的。采用了两种代理：JDK 的动态代理，与 CGLIB的动态代理。

面向切面编程，就是将交叉业务逻辑封装成切面，利用 AOP 容器的功能将切面织入到主业务逻辑中。所谓交叉业务逻辑是指，通用的、与主业务逻辑无关的代码，如安全检查、事务、日志、缓存等。

### 2、术语

1. **切面（Aspect）**

切面泛指交叉业务逻辑。上例中的事务处理、日志处理就可以理解为切面。是一个整体概念，常用的切面是通知（Advice）。实际就是对主业务逻辑的一种增强。

1. **连接点（JoinPoint）**

连接点指可以被切面织入的具体方法。通常业务接口中的方法均为连接点。

1. **切入点（Pointcut）**

切入点指声明的一个或多个连接点的集合。通过切入点指定一组要加入切面的方法。被标记为 final 的方法是不能作为连接点与切入点的。因为最终的是不能被修改的，不能被增强的。

1. **目标对象（Target）**

目标对象指将要被增强的对象。即包含主业务逻辑的类的对象 。

1. **通知（Advice）**

通知表示切面的执行时间，Advice 也叫增强。定义了增强代码切入到目标代码的时间点，是目标方法执行之前执行，还是之后执行等。通知类型不同，切入时间不同。

切入点定义要加入切面的方法，通知定义切入的时间。



## 三 AspectJ 对 AOP 的实现

AspectJ 是一个优秀面向切面的框架，它扩展了 Java 语言，提供了强大的切面实现，支持注解开发。

### 1、AspectJ 的通知类型

AspectJ 中常用的通知有五种类型：

（1）前置通知

（2）后置通知

（3）环绕通知

（4）异常通知

（5）最终通知

### 2、AspectJ 的切入点表达式

AspectJ 定义了专门的表达式用于指定切入点。表达式的原型是：

execution(modifiers-pattern? ret-type-pattern declaring-type-pattern?

name-pattern(param-pattern) throws-pattern?)

解释：

modifiers-pattern 访问权限类型

ret-type-pattern 返回值类型

declaring-type-pattern 包名类名

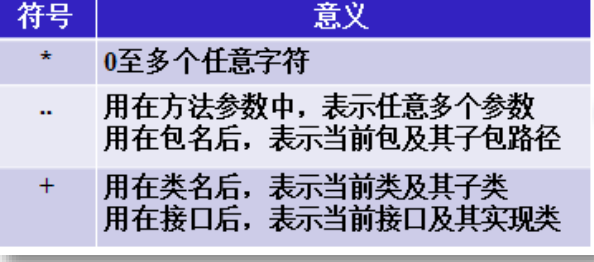
name-pattern(param-pattern) 方法名(参数类型和参数个数)

throws-pattern 抛出异常类型

？表示可选的部分  
以上表达式共 4 个部分：

execution(访问权限 方法返回值 包名类名 方法声明(参数) 异常类型)

其中可以使用以下符号：



例如：execution(void \*..SomeServiceImpl.doSome(String, Integer))

### 3、AspectJ 的开发环境

**（1）maven依赖**

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.2.5.RELEASE</version>

</dependency>

//插件

<build>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

### 4、AspectJ 基于注解的 AOP 实现

**（1）实现步骤**

**A、Step1：定义业务接口与实现类。**JDK动态代理需要实现类。

**B、Step2：定义切面类。**

在类定义的上面加上注解@Aspect，表示是用来给业务方法增加功能的类，在这个类中有切面的功能代码。

类中定义的方法是实现切面功能的。有一定要求：

1. 公共方法public。
2. 方法没有返回值。
3. 方法名称自定义
4. 方法可以有参数,也可以没有参数，如果有参数,参数不是自定义的,有几个参数类型可以使用。

**C、Step3：声明目标对象切面类对象。**

<!--把对象交给spring容器,由spring容器统一创建,管理对象-->

<!--声明目标对象-->

<bean id="someService" class="com.bjpowernode.ba01.SomeServiceImpl"/>

<!--声明切面类对象-->

<bean id="myAspect" class="com.bjpowernode.ba01.MyAspect"/>

**D、Step4：注册 AspectJ 的自动代理。**

在定义好切面 Aspect 后，需要通知 Spring 容器，让容器生成“目标类+ 切面”的代理对象。这个代理是由容器自动生成的。只需要在 Spring 配置文件中注册一个基于 aspectj 的自动代理生成器，其就会自动扫描到@Aspect 注解，并按通知类型与切入点，将其织入，并生成代理。

<aop:aspectj-autoproxy/>底层是由AnnotationAwareAspectJAutoProxyCreator 实现的。从其类名就可看出，是基于 AspectJ 的注解适配自动代理生成器。

其工作原理是，<aop:aspectj-autoproxy/>通过扫描找到@Aspect定义的切面类，再由切面类根据切入点找到目标类的目标方法，再由通知类型找到切入的时间点。

<!--

如果你希望目标类有接口时仍使用cglib代理

proxy-target-class="true" : 告诉框架，要使用cglib动态代理

-->

<aop:aspectj-autoproxy proxy-target-class="true"/>

**（2）@Before 前置通知**

在目标方法执行之前执行。

位置：在方法的上面。

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

特点：

1. 在目标方法之前先执行的。
2. 不会改变目标方法的执行结果。
3. 不会影响目标方法的执行。

指定通知方法中的参数：JoinPoint。JoinPoint为要加入切面功能的业务方法。作用是可以在通知方法中获取方法执行时的信息,例如方法名称,方法的实参。如果你的切面功能中需要用到方法的信息,就加入JoinPoint。这个JoinPoint参数的值是由框架赋予,必须是第一个位置的参数

**（3）@AfterReturning 后置通知**

属性:

1．value 切入点表达式。

2．returning 自定义的变量,表示目标方法的返回值的。自定义变量名必须和通知方法的形参名一样。

@AfterReturning(value = "execution(\* \*..SomeServiceImpl.doOther(..))",returning = "res")

位置：在方法定义的上面。

特点：

1.在目标方法之后执行的。

2.能够取到目标方法的返回值,可以根据这个返回值做不同的处理功能。相当于 Object res = doOther()。

3.可以修改这个返回值。获取目标方法的返回值进行处理时，传值(最后方法的调用结果不可改变)，传引用(最后方法的调用结果会改变)。

后置通知的执行顺序：先执行目标方法，后执行切面方法。

**（3）@Around 环绕通知**

## 二 事务

## 三 Spring与Web

## 四 常用配置