Spring的基于AspectJ的AOP开发

课程安排

- 使用AspectJ 实现AOP
  - 注解方式
  - XML方式

基于AspectJ的注解AOP开发

AspectJ 简介

- AspectJ是一个基于Java语言的AOP框架
- Spring2.0以后新增了对AspectJ切点表达式支持
- @AspectJ 是AspectJ1.5新增功能，通过JDK5注解技术，允许直接在Bean类中定义切面
- 新版本Spring框架，建议使用AspectJ方式来开发AOP
- 使用AspectJ 需要导入Spring AOP和 AspectJ相关jar包
  - spring-aop-4.2.4.RELEASE.jar
  - com.springsource.org.aopalliance-1.0.0.jar
  - spring-aspects-4.2.4.RELEASE.jar
  - com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar

注解开发：环境准备

```
<?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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop.xsd">
  <!-- 开启AspectJ自动代理-->
  <aop:aspectj-autoproxy />
</beans>
```

@AspectJ提供不同的通知类型

- @Before 前置通知，相当于BeforeAdvice
- @AfterReturning 后置通知，相当于AfterReturningAdvice
- @Around 环绕通知，相当于MethodInterceptor
- @AfterThrowing异常抛出通知，相当于ThrowAdvice
- @After 最终final通知，不管是否异常，该通知都会执行
- @DeclareParents 引介通知，相当于IntroductionInterceptor (不要求掌握)

在通知中通过value属性定义切点

- 通过execution函数，可以定义切点的方法切入
- 语法：
  - execution(<访问修饰符>?<返回类型><方法名>(<参数>)<异常>)
- 例如
  - 匹配所有类public方法 execution(public * *(..))
  - 匹配指定包下所有类方法 execution(* com.imooc.dao.*(..)) 不包含子包
  - execution(* com.imooc.dao..*(..)) ..*表示包、子孙包下所有类
  - 匹配指定类所有方法 execution(* com.imooc.service.UserService.*(..))
  - 匹配实现特定接口所有类方法
    execution(* com.imooc.dao.GenericDAO+.*(..))
  - 匹配所有save开头的方法 execution(* save*(..))

为目标类，定义切面类

- 定义切面类

```
@Aspect
public class MyAspectAnno {
```

@Before前置通知

- 可以在方法中传入JoinPoint对象，用来获得切点信息

```
// 要增强的代码：
@Before("execution(* com.imooc.spring.demo1.UserDao.save(..))")
public void before(JoinPoint joinPoint){
    System.out.println("========前置通知======"+joinPoint);
}
```

@AfterReturing 后置通知

- 通过returning属性 可以定义方法返回值，作为参数

```
@AfterReturning(value="execution(* com.imooc.spring.demo1.UserDao.update(..))",returning="returing")
public void afterReturing(Object returing){
    System.out.println("========后置通知======"+returing);
}
```

@Around 环绕通知

- around方法的返回值就是目标代理方法执行返回值
- 参数为ProceedingJoinPoint 可以调用拦截目标方法执行

```
@Around("execution(* com.imooc.spring.demo1.UserDao.delete(..))")
public Object around(ProceedingJoinPoint joinPoint) throws Throwable{
    System.out.println("========环绕前通知======");
    Object obj = joinPoint.proceed();
    System.out.println("========环绕后通知======");
    return obj;
}
```

重点：如果不调用 ProceedingJoinPoint的 proceed方法，那么目标方法就被拦截了

@AfterThrowing 异常抛出通知

- 通过设置throwing属性，可以设置发生异常对象参数

```
@AfterThrowing(value="execution(* com.imooc.spring.demo1.UserDao.find(..))",throwing="e")
public void afterThrowing(Throwable e){
    System.out.println("========异常抛出通知======"+e.getMessage());
}
```

@After 最终通知

- 无论是否出现异常,最终通知总是会被执行的

```
@After(value="execution(* com.imooc.spring.demo1.UserDao.findAll(..))")
public void after(){
    System.out.println("========最终通知======");
}
```

通过@Pointcut为切点命名

- 在每个通知内定义切点，会造成工作量大，不易维护，对于重复的切点，可以使用@Pointcut进行定义
- 切点方法：private void 无参数方法，方法名为切点名
- 当通知多个切点时，可以使用|| 进行连接

基于AspectJ的XML方式的AOP开发

使用XML配置切面

- 编写切面类

```
public class MyAspectJXML {

    public void before(){
        System.out.println("前置增强=============");
    }

    public void afterReturing(Object obj){
        System.out.println("后置增强============="+obj);
    }

    public Object around(ProceedingJoinPoint joinPoint) throws Throwable{
        System.out.println("环绕前增强===========");
        Object obj = joinPoint.proceed();
        System.out.println("环绕后增强===========");
        return obj;
    }

    public void afterThrowing(Throwable e){
        System.out.println("异常抛出通知==========="+e.getMessage());
    }

    public void after(){
        System.out.println("最终通知=============");
    }
}
```

使用XML配置切面

- 完成切面类的配置

```
<bean id="myAspectJXML" class="com.imooc.spring.demo2.MyAspectJXML"/>
```

使用XML配置切面

- 配置AOP完成增强

```
<!-- aop的配置 -->
<aop:config>
    <!-- 定义切入点:哪些类的哪些方法需要增强 -->
    <aop:pointcut expression="execution(* com.imooc.spring.demo2.ProductDao.save(..))" id="pointcut1"/>
    <aop:pointcut expression="execution(* com.imooc.spring.demo2.ProductDao.update(..))" id="pointcut2"/>
    <aop:pointcut expression="execution(* com.imooc.spring.demo2.ProductDao.delete(..))" id="pointcut3"/>
    <aop:pointcut expression="execution(* com.imooc.spring.demo2.ProductDao.find(..))" id="pointcut4"/>
    <aop:pointcut expression="execution(* com.imooc.spring.demo2.ProductDao.findAll(..))" id="pointcut5"/>

    <!-- 配置切面 -->
    <aop:aspect ref="myAspectJXML">
        <aop:before method="before" pointcut-ref="pointcut1"/>
        <aop:after-returning method="afterReturing" pointcut-ref="pointcut2" returning="obj"/>
        <aop:around method="around" pointcut-ref="pointcut3"/>
        <aop:after-throwing method="afterThrowing" pointcut-ref="pointcut4" throwing="e"/>
        <aop:after method="after" pointcut-ref="pointcut5" />
    </aop:aspect>
</aop:config>
```

课程总结