## AOP的概述:

### 什么是AOP:

AOP Aspect Oriented Programing 面向切面编程

AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码（性能监视、事务管理、安全检查、缓存）

Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码

AspecJ是一个基于Java语言的AOP框架，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入

术语：面向切面编程、织入、AspectJ（AOP框架）

### AOP底层原理;

就是代理机制:

\* 动态代理:(JDK中使用)

\* JDK的动态代理,对实现了接口的类生成代理.

### Spring的AOP代理:

JDK动态代理:对实现了接口的类生成代理（默认）

CGLib代理机制:对类生成代理

### AOP的术语:

Joinpoint(连接点):所谓连接点是指那些被拦截到的点。在spring中,这些点指的是方法,因为spring只支持方法类型的连接点.

Pointcut(切入点):所谓切入点是指我们要对哪些Joinpoint进行拦截的定义.

Advice(通知/增强):所谓通知是指拦截到Joinpoint之后所要做的事情就是通知.通知分为前置通知,后置通知,异常通知,最终通知,环绕通知(切面要完成的功能)

Introduction(引介):引介是一种特殊的通知在不修改类代码的前提下, Introduction可以在运行期为类动态地添加一些方法或Field.

Target(目标对象):代理的目标对象

Weaving(织入):是指把增强应用到目标对象来创建新的代理对象的过程.

spring采用动态代理织入，而AspectJ采用编译期织入和类装在期织入

Proxy（代理）:一个类被AOP织入增强后，就产生一个结果代理类

Aspect(切面): 是切入点和通知（引介）的结合

Spring是一个javaEE框架，AspectJ是AOP框架，从等级来说是相同的，Spring作为黏合剂，将AspectJ整合到系统中

## AOP的底层实现

### JDK动态代理:

public class JDKProxy implements InvocationHandler{

private UserDao userDao;

public JDKProxy(UserDao userDao) {

super();

this.userDao = userDao;

}

public UserDao createProxy() {

UserDao proxy = (UserDao) Proxy.newProxyInstance(userDao.getClass()

.getClassLoader(), userDao.getClass().getInterfaces(), this);

return proxy;

}

// 调用目标对象的任何一个方法 都相当于调用invoke();

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable {

if("add".equals(method.getName())){

// 记录日志:

System.out.println("日志记录=================");

Object result = method.invoke(userDao, args);

return result;

}

return method.invoke(userDao, args);

}

}

### CGLIB动态代理:

CGLIB(Code Generation Library)是一个开源项目！是一个强大的，高性能，高质量的Code生成类库，它可以在运行期扩展Java类与实现Java接口。 Hibernate支持它来实现PO(Persistent Object 持久化对象)字节码的动态生成

Hibernate生成持久化类的javassist.

CGLIB生成代理机制:其实生成了一个真实对象的子类.

下载cglib的jar包.

\* 现在做cglib的开发,可以不用直接引入cglib的包.已经在spring的核心中集成cglib.

public class CGLibProxy implements MethodInterceptor{

private ProductDao productDao;

public CGLibProxy(ProductDao productDao) {

super();

this.productDao = productDao;

}

public ProductDao createProxy(){

// 使用CGLIB生成代理:

// 1.创建核心类:

Enhancer enhancer = new Enhancer();

// 2.为其设置父类:

enhancer.setSuperclass(productDao.getClass());

// 3.设置回调:

enhancer.setCallback(this);

// 4.创建代理:

return (ProductDao) enhancer.create();

}

public Object intercept(Object proxy, Method method, Object[] args,

MethodProxy methodProxy) throws Throwable {

if("add".equals(method.getName())){

System.out.println("日志记录==============");

Object obj = methodProxy.invokeSuper(proxy, args);

return obj;

}

return methodProxy.invokeSuper(proxy, args);

}

}

**结论:Spring框架,如果类实现了接口,就使用JDK的动态代理生成代理对象,如果这个类没有实现任何接口,使用CGLIB生成代理对象.**

## Spring中的AOP

### Spring的传统AOP :

AOP:不是由Spring定义.AOP联盟的组织定义.

Spring中的通知:(增强代码)

前置通知 org.springframework.aop.MethodBeforeAdvice

\* 在目标方法执行前实施增强

后置通知 org.springframework.aop.AfterReturningAdvice

\* 在目标方法执行后实施增强

环绕通知 org.aopalliance.intercept.MethodInterceptor

\* 在目标方法执行前后实施增强

异常抛出通知 org.springframework.aop.ThrowsAdvice

\* 在方法抛出异常后实施增强

引介通知 org.springframework.aop.IntroductionInterceptor(课程不讲.)

\* 在目标类中添加一些新的方法和属性