1. AOP介绍
   1. 什么是AOP

在软件业，AOP为Aspect Oriented Programming的缩写，意为：[面向切面编程](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%88%87%E9%9D%A2%E7%BC%96%E7%A8%8B/6016335)，通过[预编译](https://baike.baidu.com/item/%E9%A2%84%E7%BC%96%E8%AF%91/3191547)方式和运行期动态代理实现程序功能的统一维护的一种技术。AOP是[OOP](https://baike.baidu.com/item/OOP" \t "_blank)的延续，是软件开发中的一个热点，也是[Spring](https://baike.baidu.com/item/Spring" \t "_blank)框架中的一个重要内容，是[函数式编程](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0%E5%BC%8F%E7%BC%96%E7%A8%8B/4035031" \t "_blank)的一种衍生范型。利用AOP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](https://baike.baidu.com/item/%E8%80%A6%E5%90%88%E5%BA%A6/2603938)降低，提高程序的可重用性，同时提高了开发的效率。

应用：事务管理、性能监控、日志、缓存等

AspectJ是一个基于Java语言的AOP框架

* 1. AOP实现原理

Aop底层采用代理机制实现

接口+类；spring采用jdk的动态代理Proxy

实现类；spring 采用cglib字节码增强

* 1. AOP术语

1. **target：目标类**，需要被代理的类
2. **JoinPoint（连接点）**所谓连接点是指那些可能被拦截的方法。例如，所有的方法
3. **PointCut 切入点**，已经被增强的连接点。例如，addUser(){}
4. **advice 通知/增强**，增强代码，例如 after() 、before()，面向切面要完成的事情
5. **weaving(织入)** 指把增强的advice应用到目标对象target来创建新的代理对象proxy的过程
6. **proxy 代理类 –由工厂生成**
7. **Aspect(切面)** 是切入点（PointCut）和通知（advice）的结合

UserService{

addUser()

updateUser()

deleteUser()

}

1.target

2.JoinPoint

3.PointCut

A

Before(){

//开启事务

}

After(){

//提交事务

}

4.advice

7.Aspect 切面

代理类

6.proxy

5.weaving 织入过程

------------------------

1. 手动代理—JDK动态代理

JDK动态代理 对“装饰者”设计模式的简化。使用前提：必须有接口

目标类：

切面类：

工厂类：生产对象

1. 手动代理—CGLIB代理（字节码增强）
2. 没有接口，只有实现类；
3. 采用字节码增强框架(如 cglib)，在运行时创建目标类的子类，从而对目标类进行增强
4. 导入jar包：cglib-2.2.jar 和 asm-3.3.jar

spring-core.jar 已经整合以上两个jar包

------------------------

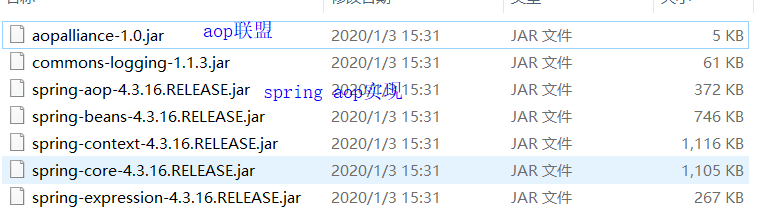
1. Spring工厂bean创建代理（半自动）

（1）、让spring 创建代理对象，从spring容器中手动获取代理对象。

（2）、导入jar包：

核心：4+1，spring环境jar包

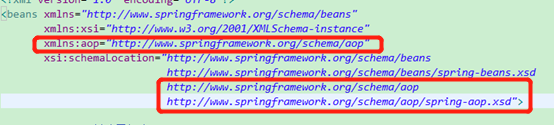
AOP：AOP联盟(规范)、spring-aop（实现）



1. 传统spring aop编程 （全自动）

导入jar包：org.aspectj-weaver.jar

|  |
| --- |
| 3、aop 编程  3.1、导入命名空间  3.2、使用<aop:config>进行配置  <aop:pointcut> 切入点，从目标对象获取具体方法  <aop:advisor> 特殊的切面，只有一个通知和一个切入点  advice-ref 通知引用  pointcut-ref 切入点引用  3.3 切入点表达式  execution(\* proxy\_spring\_aop.UserServiceImpl.\*(..))  选择方法 返回值任意 包 类名 方法名任意 参数任意 |



|  |
| --- |
| <aop:config>  <aop:pointcut expression="execution(\* proxy\_spring\_aop.UserServiceImpl.deleteUser(..))" id="myPointCut"/>  <aop:advisor advice-ref="myAspectId" pointcut-ref="myPointCut"/>  </aop:config> |

|  |
| --- |
| import org.aopalliance.intercept.MethodInterceptor;  import org.aopalliance.intercept.MethodInvocation;  public class MyAspect implements MethodInterceptor{  @Override  public Object invoke(MethodInvocation mi) throws Throwable {    System.out.println("鸡头");  // 手动执行目标方法  Object obj = mi.proceed();  System.out.println("凤尾");  return obj;  }  } |

------------------------

1. AspectJ介绍

Aspectj是一个基于java的AOP框架

支持注解

主要用途，自定义开发(将自己代码融入到别人代码中)

1. AspectJ切入点表达式

1、execution() 用于描述方法

语法：execution(修饰符 返回值 包.类.方法名(参数)throws 异常)

修饰符，一般省略

public 公共方法

\* 任意

返回值，

void 没有返回值

String 返回字符串

\* 任意

包，[可省略]

类，[可省略]

方法名，[不可省略]

(参数)

() 无参

(int) 一个整型

(int，int) 两个整型

(..) 参数任意

throws 异常，[可省略]一般不写

1. AspectJ通知类型

(1)、aop联盟

(2)、

(3)、通知类型个数：

1. AspectJ开发—前置通知

-------------------------

1. AspectJ开发基于xml
2. AspectJ开发基于注解