* aop编程

aop( aspect oriented programming ) 面向切面(方面)编程,是对所有对象或者是一类对象编程,核心是( 在**不**增加代码的基础上， **还**增加新功能 )

汇编(伪机器指令 mov jump) 面向机器

c语言(面向过程 )->系统软件(操作系统，数据库, 杀毒软件，防火墙,驱动..)

# 开始编程

步骤:

## 1.定义接口

**public** **interface** TestServiceInter {

**public** **void** sayHello();

}

## 2. 前置通知实现接口MethodBeforeAdvice，，这个前置通知在调用方法的时候才会被调用

**public** **class** MyMethodBeforeAdvice **implements** MethodBeforeAdvice {

/\*\*

\* method: 被调用的方法

\* args: 这个是传给方法的参数有哪些

\* target: 这个是目标对象

\*/

**public** **void** before(Method method, Object[] args, Object target)

**throws** Throwable {

// **TODO** Auto-generated method stub

System.*out*.println("得到的方法..."+method.getName());

}

}

## 3.配置容器文件

<!-- 配置被代理的对象 -->

<bean id=*"test1Service"* class=*"com.hsp.aop.Test1Service"*>

<property name=*"name"* value=*"顺平"* />

</bean>

<!-- 配置前置通知

proxyFactoryBean implements TestServiceInter,TestServiceInter2{

public void sayHello();

}

思考

interface Inter1{};

class A implements Inter1,Inter2{

}

Inter1 a=new A();

Inter2 b=(Inter2)a;

-->

<bean id=*"MyMethodBeforeAdvice"* class=*"com.hsp.aop.MyMethodBeforeAdvice"* />

<!-- 配置代理对象 -->

<bean id=*"proxyFactoryBean"* class=*"org.springframework.aop.framework.ProxyFactoryBean"*>

<!-- 代理接口集 -->

<property name=*"proxyInterfaces"*>

<list>

<value>com.hsp.aop.TestServiceInter</value>

<value>com.hsp.aop.TestServiceInter2</value>

</list>

</property>

<!-- 把通知 织入到代理对象 -->

<property name=*"interceptorNames"*>

<!-- 相当于包MyMethodBeforeAdvice 前置通知和代理对象关联,我们也

可以把通知看出拦截器, struts2核心拦截器 -->

<value>MyMethodBeforeAdvice</value>

</property>

<!-- 配置被代理对象,可以指定 -->

<property name=*"target"* ref=*"test1Service"*/>

</bean>

## 4.测试得到的是代理对象，

1. 编写对象(被代理对象=目标对象)
2. 编写通知（前置通知目标方法调用前调用）
3. 在beans.xml文件配置
   1. 配置 被代理对象=目标对象
   2. 配置通知
   3. 配置代理对象 是 ProxyFactoryBean的对象实例
      1. <!-- 代理接口集 -->
      2. 织入通知
      3. 配置被代理对象

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ApplicationContext ac=**new** ClassPathXmlApplicationContext("com/hsp/aop/beans.xml");

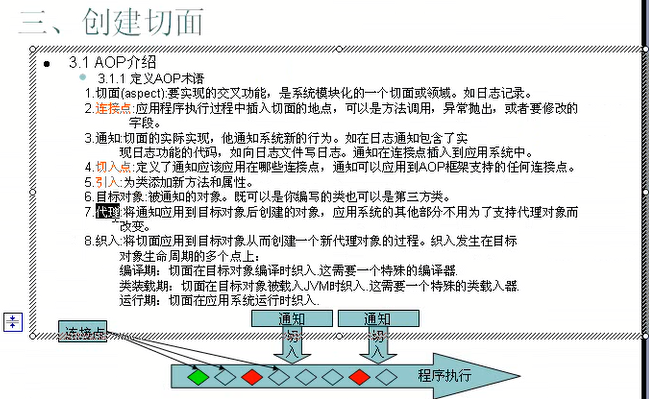
TestServiceInter ts=(TestServiceInter) ac.getBean("proxyFactoryBean");

ts.sayHello();

((TestServiceInter2)ts).sayBye();

}

# 2aop详解



# 3.后置通知AfterReturningAdvice 关闭资源

## 1.实现接口AfterReturningAdvice

**public** **class** MyAfterRunReturningAfterAdvice **implements** AfterReturningAdvice {

@Override

**public** **void** afterReturning(Object arg0, Method arg1, Object[] arg2,

Object arg3) **throws** Throwable {

System.*out*.println("关闭资源"+arg1.getName());

// **TODO** Auto-generated method stub

}

}

## 3.2进行配置

<!-- 把通知 织入到代理对象 -->

<property name=*"interceptorNames"*>

<!-- 相当于包MyMethodBeforeAdvice 前置通知和代理对象关联,我们也

可以把通知看出拦截器, struts2核心拦截器 -->

<list>

<value>MyMethodBeforeAdvice</value>

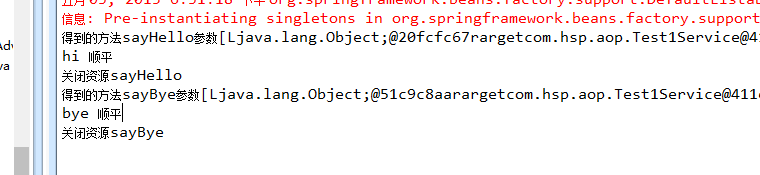
<!-- 后置通知 -->

<value>MyAfterRunReturningAfterAdvice</value>

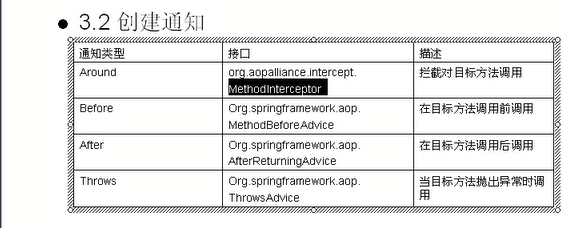
</list>

</property>

## 3.3测试



# 4.环绕通知MethodInterceptor，在中间执行



## 4.1 MethodInterceptor实现

**public** **class** MyAroundAdvice **implements** MethodInterceptor{

@Override

**public** Object invoke(MethodInvocation arg0) **throws** Throwable {

// **TODO** Auto-generated method stub

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.*out*.println("调用方法前");

//这个就是 调用的方法

Object obj = arg0.proceed();

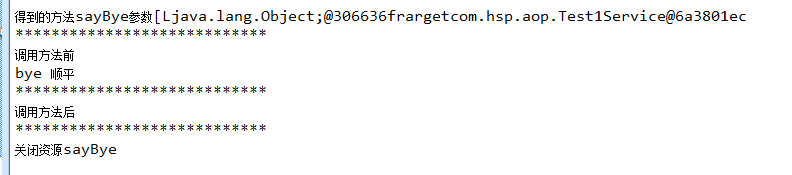
System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.*out*.println("调用方法后");

**return** obj;

}

## 4.2测试



# 5.异常通知ThrowsAdvice

**public** **class** MyThowAdvoce **implements** ThrowsAdvice{

**public** **void** afterThrowing(Method m,Object[] os,Object target,Exception throwable){

System.*out*.println(m+"出大事了"+throwable.getMessage());

}

}

# 6.引入通知