[spring aop expression简单说明](http://www.cnblogs.com/smile361/archive/2012/09/25/2701252.html)

<aop:config>

<aop:pointcut id="userDAO"

expression="execution(public \* cn.dao.IUserDAO.\*(..))" />

<aop:advisor advice-ref="tx" pointcut-ref="userDAO"/>

</aop:config>

在上面的代码中

execution   是方法运行

public         是指定public的方法，也可以不写直接：execution(\* cn.dao.IUserDAO.\*(..)

\*                  是任意返回值，可以有返回值，也可以是void没有返回值的方法

cn.dao.IUserDAO.\*                  是指定目录下的指定类任意方法

cn.dao.IUserDAO.insert\*       是指定目录下的指定类insert开头的任意方法

cn.dao.IUserDAO.\*.\*              是指定目录下的任意类下的任意方法

cn.dao..\*.\*                                是指定目录下的任意目录下任意类下的任意方法

(..)                                              是任何参数，可以是没有参数

在execution中是可以有多个的方法，例如：

execution(\* com.action.userinfoAction..\*(..))&&execution(\* com.action.memberAction..\*(..))&&!execution(\* get\*(..))&&!execution(\* set\*(..))

Pointcut 是指那些方法需要被执行"AOP",是由"Pointcut Expression"来描述的.   
Pointcut可以有下列方式来定义或者通过&& || 和!的方式进行组合.   
args()   
@args()   
execution()   
this()   
target()   
@target()   
within()   
@within()   
@annotation   
其中execution 是用的最多的,其格式为:   
execution(modifiers-pattern? ret-type-pattern declaring-type-pattern? name-pattern(param-pattern)throws-pattern?)   
returning type pattern,name pattern, and parameters pattern是必须的.   
ret-type-pattern:可以为\*表示任何返回值,全路径的类名等.   
name-pattern:指定方法名,\*代表所以,set\*,代表以set开头的所有方法.   
parameters pattern:指定方法参数(声明的类型),(..)代表所有参数,(\*)代表一个参数,(\*,String)代表第一个参数为任何值,第二个为String类型.   
举例说明:   
任意公共方法的执行：   
execution(public \* \*(..))   
任何一个以“set”开始的方法的执行：   
execution(\* set\*(..))   
AccountService 接口的任意方法的执行：   
execution(\* com.xyz.service.AccountService.\*(..))   
定义在service包里的任意方法的执行：   
execution(\* com.xyz.service.\*.\*(..))   
定义在service包和所有子包里的任意类的任意方法的执行：   
execution(\* com.xyz.service..\*.\*(..))   
定义在pointcutexp包和所有子包里的JoinPointObjP2类的任意方法的执行：   
execution(\* com.test.spring.aop.pointcutexp..JoinPointObjP2.\*(..))")   
\*\*\*> 最靠近(..)的为方法名,靠近.\*(..))的为类名或者接口名,如上例的JoinPointObjP2.\*(..))   
  
pointcutexp包里的任意类.   
within(com.test.spring.aop.pointcutexp.\*)   
pointcutexp包和所有子包里的任意类.   
within(com.test.spring.aop.pointcutexp..\*)   
实现了Intf接口的所有类,如果Intf不是接口,限定Intf单个类.   
this(com.test.spring.aop.pointcutexp.Intf)   
\*\*\*> 当一个实现了接口的类被AOP的时候,用getBean方法必须cast为接口类型,不能为该类的类型.  
  
带有@Transactional标注的所有类的任意方法.   
@within(org.springframework.transaction.annotation.Transactional)   
@target(org.springframework.transaction.annotation.Transactional)   
带有@Transactional标注的任意方法.   
@annotation(org.springframework.transaction.annotation.Transactional)   
\*\*\*> @within和@target针对类的注解,@annotation是针对方法的注解   
  
参数带有@Transactional标注的方法.   
@args(org.springframework.transaction.annotation.Transactional)   
参数为String类型(运行是决定)的方法.   
args(String)   
Pointcut 可以通过[**Java**](http://lib.csdn.net/base/17)注解和XML两种方式配置,如下所示:

复制代码

<aop:config>

<aop:aspectref="aspectDef">

<aop:pointcutid="pointcut1"expression="execution(\* com.test.spring.aop.pointcutexp..JoinPointObjP2.\*(..))"/>

<aop:before pointcut-ref="pointcut1" method="beforeAdvice" />

</aop:aspect>

</aop:config>

@Component

@Aspect

public class AspectDef {

//@Pointcut("execution(\* com.test.spring.aop.pointcutexp..JoinPointObjP2.\*(..))")

//@Pointcut("within(com.test.spring.aop.pointcutexp..\*)")

//@Pointcut("this(com.test.spring.aop.pointcutexp.Intf)")

//@Pointcut("target(com.test.spring.aop.pointcutexp.Intf)")

//@Pointcut("@within(org.springframework.transaction.annotation.Transactional)")

//@Pointcut("@annotation(org.springframework.transaction.annotation.Transactional)")

@Pointcut("args(String)")

public void pointcut1() {

}

@Before(value = "pointcut1()")

public void beforeAdvice() {

System.out.println("pointcut1 @Before...");

}