原文：<http://www.baeldung.com/spring-aop-pointcut-tutorial>

**概括**

这遍文章将介绍Spring AOP切点表达式(下称表达式)语言，首先介绍两个面向切面编程中使用到的术语。

* 连接点(Joint Point)：广义上来讲，方法、异常处理块、字段这些程序调用过程中可以抽像成一个执行步骤（或者说执行点）的单元。从Spring AOP来讲，就是指java的方法和异常处理代码块。
* 切点(Pointcut)：是连接点的描述定义，Spring AOP通过切点来定位到哪些连接点。切点表达式语言就是切点用来定义连接点的语法。

**用例**

表达式会出现在以下几种场景

1. 作为@Pointcut的参数，用以定义连接点
2. **@Pointcut**("within(@org.springframework.stereotype.Repository \*)")

**public** **void** **repositoryClassMethods**() {}

在上面的代码片段中的注解@Pointcut的参数"within(@org.springframework.stereotype.Reposity \*)"就是使用的切点表达式。而上代码中的repositoryClassMethods()方法被AOP AspectJ定义为切点签名方法，作用是使得通知的注解可以通过这个切点签名方法连接到切点，通过解释切点表达式找到需要被切入的连接点。最终的目的都是为了找到需要被切入的连接点。像下面这段代码片段

**@Around**("repositoryClassMethods()")

**public** Object **measureMethodExecutionTime**(ProceedingJoinPoint pjp) **throws** Throwable {

...

}

如果你的项目是基于xml配置的，可以在<aop:pointcut>标签里配置表达式来定位连接点，参考以下代码片段:

<aop:config>

<aop:pointcut id="anyDaoMethod"

expression="@target(org.springframework.stereotype.Repository)"/>

</aop:config>

**切点指示符**

切点指示符是切点定义的关键字，切点表达式以切点指示符开始。开发人员使切点指示符来告诉切点将要匹配什么，有以下9种切点指示符：execution、within、this、target、args、@target、@args、@within、@annotation，下面一一介结这9种切点指示符。

**execution**

execution是一种使用频率比较高比较主要的一种切点指示符，用来匹配方法签名，方法签名使用全限定名，包括访问修饰符（public/private/protected）、返回类型，包名、类名、方法名、参数，其中返回类型，包名，类名，方法，参数是必须的，如下面代码片段所示：

@**Pointcut**("**execution**(**public** String org.baeldung.dao.FooDao.findById(Long))")

上面的代码片段里的表达式精确地匹配到FooDao类里的findById(Long)方法，但是这看起来不是很灵活。假设我们要匹配FooDao类的所有方法，这些方法可能会有不同的方法名，不同的返回值，不同的参数列表，为了达到这种效果，我们可以使用通配符。如下代码片段所示：

@Pointcut("execution(\* org.baeldung.dao.FooDao.\*(..))")

第一个通配符匹配所有返回值类型，第二个匹配这个类里的所有方法，（）括号表示参数列表，括号里的用两个点号表示匹配任意个参数，包括0个

**within**

使用within切点批示符可以达到上面例子一样的效果，within用来限定连接点属于某个确定类型的类。如下面代码的效果与上面的例子是一样的：

@**Pointcut**("**within**(**org**.**baeldung**.**dao**.**FooDao**)")

我们也可以使用within指示符来匹配某个包下面所有类的方法（包括子包下面的所有类方法），如下代码所示：

@**Pointcut**("**within**(**org**.**baeldung**..\*)")

**this 和 target**

this用来匹配的连接点所属的对象引用是某个特定类型的实例，target用来匹配的连接点所属目标对象必须是指定类型的实例；那么这两个有什么区别呢？原来AspectJ在实现代理时有两种方式：  
1、如果当前对象引用的类型没有实现自接口时，spring aop使用生成一个基于CGLIB的代理类实现切面编程  
2、如果当前对象引用实现了某个接口时，Spring aop使用JDK的动态代理机制来实现切面编程  
this指示符就是用来匹配基于CGLIB的代理类，通俗的来讲就是，如果当前要代理的类对象没有实现某个接口的话，则使用this；target指示符用于基于JDK动态代理的代理类，通俗的来讲就是如果当前要代理的目标对象有实现了某个接口的话，则使用target.：

**public** **class** **FooDao** **implements** **BarDao** {

...

}

比如在上面这段代码示例中，spring aop将使用jdk的动态代理来实现切面编程，在编写匹配这类型的目标对象的连接点表达式时要使用target指示符， 如下所示：

@**Pointcut**("**target**(**org**.**baeldung**.**dao**.**BarDao**)")

如果FooDao类没有实现任何接口，或者在spring aop配置属性：proxyTargetClass设为true时，Spring Aop会使用基于CGLIB的动态字节码技为目标对象生成一个子类将为代理类，这时应该使用this指示器：

@**Pointcut**("**this**(**org**.**baeldung**.**dao**.**FooDao**)")

**参数**

参数指示符是一对括号所括的内容，用来匹配指定方法参数：

@Pointcut("execution(\* \*..find\*(Long))")

这个切点匹配所有以find开头的方法，并且只一个Long类的参数。如果我们想要匹配一个有任意个参数，但是第一个参数必须是Long类的，我们这可使用下面这个切点表达式：

@Pointcut("execution(\* \*..find\*(Long,..))")

**@Target**

这个指示器匹配指定连接点，这个连接点所属的目标对象的类有一个指定的注解:

@**Pointcut**("@**target**(**org**.**springframework**.**stereotype**.**Repository**)")

**@args**

这个指示符是用来匹配连接点的参数的，@args指出连接点在运行时传过来的参数的类必须要有指定的注解，假设我们希望切入所有在运行时接受实@Entity注解的bean对象的方法：

@**Pointcut**("@**args**(**org**.**baeldung**.**aop**.**annotations**.**Entity**)")

public void methodsAcceptingEntities() {}

为了在切面里接收并使用这个被@Entity的对象，我们需要提供一个参数给切面通知：JointPoint:

**@Before**("methodsAcceptingEntities()")

**public** **void** **logMethodAcceptionEntityAnnotatedBean**(JoinPoint jp) {

logger.info("Accepting beans with @Entity annotation: " + jp.getArgs()[0]);

}

**@within**

这个指示器，指定匹配必须包括某个注解的的类里的所有连接点：

@**Pointcut**("@**within**(**org**.**springframework**.**stereotype**.**Repository**)")

上面的切点跟以下这个切点是等效的：

@**Pointcut**("**within**(@**org**.**springframework**.**stereotype**.**Repository** \*)")

**@annotation**

这个指示器匹配那些有指定注解的连接点，比如，我们可以新建一个这样的注解@Loggable:

@**Pointcut**("@**annotation**(**org**.**baeldung**.**aop**.**annotations**.**Loggable**)")

public void loggableMethods() {}

我们可以使用@Loggable注解标记哪些方法执行需要输出日志：

**@Before**("loggableMethods()")

**public** **void** **logMethod**(JoinPoint jp) {

String methodName = jp.getSignature().getName();

logger.info("Executing method: " + methodName);

}

**切点表达式组合**

可以使用&&、||、!、三种运算符来组合切点表达式，表示与或非的关系。

**@Pointcut**("@target(org.springframework.stereotype.Repository)")

**public** **void** **repositoryMethods**() {}

**@Pointcut**("execution(\* \*..create\*(Long,..))")

**public** **void** **firstLongParamMethods**() {}

**@Pointcut**("repositoryMethods() && firstLongParamMethods()")

**public** **void** **entityCreationMethods**() {}