Spring_切点表达式

摘要： Spring 中的 AspectJ 切点表达式函数 切点表达式函数就像我们的 GPS 导航软件。通过切点表达式函数，再配合通配符和逻辑运算符的灵活运用，我们能很好定位到我们需要织入增强的连接点上。经过上面的铺垫，下面来看看 Springz 中支持的切点表

Spring 中的 AspectJ 切点表达式函数

切点表达式函数就像我们的 GPS 导航软件。通过切点表达式函数，再配合通配符和逻辑运算符的灵活运用，我们能很好定位到我们需要织入增强的连接点上。经过上面的铺垫，下面来看看 Springz 中支持的切点表达式函数。

1. 方法切点函数

函数	入参	说明	示例
execution()	方法匹配字符串	满足某一匹配模式的的所有目标类方法连接点	execution(* com.yc.service.*.*(..))在配置 service 层的事务管理时常用，定位于任意返回类型（第一个”*”） 在 com.yc.service 包下的所有类（第二个”*”）下的所有方法（第三个”*”),且这个方法的入参为任意类型、数量（体现在 “(..)“）
@annotation()	方法注解类名	标注了特定注解的目标方法连接点上	@anntation(com.yc.controller.needRecord)，定位于 controller 层中任何添加@needRecord 的方法，这可以方便地对控制层中某些方法被调用（如某人某时间登陆、进入后台管理界面）添加日志记录。

1. execution 详解

execution 的语法表达式如下：execution(<修饰符> <返回类型> <类路径> <方法名>(<参数列表>) <异常模式>)

其中，修饰符和异常是可选的,如果不加类路径，则默认对所有的类生效。它常用实例如下：

1. 通过方法签名、返回值定义切点：

- `execution(public * *Service(..))`：定位于所有类下返回值任意、方法入参类型、数量任意，public 类型的方法
- `execution(public String *Service(..))`：定位于所有类下返回值为 String、方法入参类型、数量任意，public 类型的方法

2. 通过类包定义切点：

- `execution(* com.yc.controller.BaseController+.*(..))`:匹配任意返回类型，对应包下 BaseController 类及其子类等任意方法。
- `execution(* com.*.(..))`:匹配任意返回类型，com 包下所有类的所有方法
- `execution(* com..*.(..))`:匹配任意返回类型，com 包、子包下所有类的所有方法

注意.表示该包下所有类，..则涵括其子包。

3. 通过方法入参定义切点

- 这里”*”表示任意类型的一个参数，“..”表示任意类型任意数量的参数
- `execution(* speak(Integer,*))`:匹配任意返回类型，所有类中只有两个入参，第一个入参为 Integer，第二个入参任意的方法
- `execution(* speak(..,Integer,..))`:匹配任意返回类型，所有类中至少有一个 Integer 入参，但位置任意的方法。

2. annotation 详解

此注解用于定位标注了某个注解的目标切点。下面我们来看一个模拟用户登录成功后日志记录用户名、时间和调用方法的示例,

1. 自定义注解

```
@Retention(RetentionPolicy.CLASS)//生命注释保留时长，这里无需反射使用，使用 CLASS 级别
@Target(ElementType.METHOD)//生命可以使用此注解的元素级别类型（如类、方法变量等）
public @interface NeedRecord {
}
```

关于自定义注解的更多属性与说明，可查看我的另一篇文章

http://blog.csdn.net/qwe6112071/article/details/50949663。

2. 定义切面（配置增强和定位切点）

```
@Aspect//将当前类标注成一个切面。
public class Annotation_aspect {
    @AfterReturning("@annotation(test.aop2.NeedRecord)")//这里指向注解类
    public void Record(JoinPoint joinPoint){//切点入参。
        System.out.println("日志记录:用户" +joinPoint.getArgs()[0] + "在" + new SimpleDateFormat("yyyy-MM-dd hh:mm;ss").format(new Date()) + "调用了"+ joinPoint.getSignature()+"方法" );
    }
}
```

3. 定义目标对象

```
@NeedRecord
public void login(String name){
    System.out.println("I'm "+name+" ,I'm logining");
}
```

4. 配置 IOC 容器

```
<aop:aspectj-autoproxy />   <!-- 使@AspectJ 注解生效 -->
<bean class="test.aop2.Annotation_aspect" /><!-- 注册切面，使 AOP 自动识别并进行 AOP 方面的配置 -->
<bean id="userController" class="test.aop2.UserController" /><!-- 注册目标对象 -->
```

这里需注意：

1. 和 标注在目标对象 Annotation_aspect 的注解@Aspect 缺一不可，否则调用 login 方法时，Record 方法不会被调用

2. 必须在 IOC 容器中注册切面和目标对象，以便在下面测试中通过

5. 测试

```
public static void main(String args[]){
        ApplicationContext ac = new ClassPathXmlApplicationContext("classpath:test/aop2/aop.xml");
        UserController userController = (UserController) ac.getBean("userController");
        userController.login("zenghao");
    }
```

调用测试方法后，会打印信息：

```
I’m zenghao ,I’m logining
DEBUG: org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean ‘test.aop2.Annotation_aspect#0’//log4j 的日志记录打印
日志记录:用户 zenghao 在 2016-03-21 08:27;48 调用了 void test.aop2.UserController.login(String)方法
```

2. 方法入参切点函数

函数	入参	说明	示例
args()	类名	定位于入参为特定类型的的方法	如 args(com.yc.model.User， com.yc.model.Article),我们要定位于所有以 User,Article 为入参的方法，需要注意的是，类型的个数、顺序必须都一一对应）

@args()	类型注解类名	定位于被特定注解的类作为方法入参的连接点	@args(com.yc.annotation.MyAnnotation)。MyAnnotation 为自定义注解，标注在目标对象方法入参上，被标注的目标都会被匹配。，如方法 public myMethod(@MyAnnotation String args);

1. args()详解：

args 函数接受一个类名或变量名（对应与目标对象方法的入参），并将该类名绑定到增强方法入参中。对上一个实例，我们将切面改造成：

```
@After("args(name)")
public void Record(JoinPoint joinPoint,String name) throws Throwable{
    System.out.println("日志记录:用户" +name+ "在" + new SimpleDateFormat("yyyy-MM-dd hh:mm;ss").format(new Date()) + "调用了"+ joinPoint.getSignature().getDeclaringTypeName()+"方法" );
    /*控制台打印
    I'm zenghao ,I'm logining
    日志记录:用户 zenghao 在 2016-03-21 09:32;31 调用了 test.aop2.UserController 方法
    */
}
```

在这里有几点是值得注意的：

1. 在本例中，我们不能使用 args(String)来匹配，因为我们在方法入参中加入了变量 name,必须通过 args（）绑定连接点入参的机制：通过在方法声明中定义入参 String name，然后 args 会搜寻方法定义中命名参数来获取对应的参数类型（这里是 String)。

2. 如果我们使用 args(name)，但在方法定义体中没声明 String name,则会报错 java.lang.IllegalArgumentException: warning no match for this type name: name [Xlint:invalidAbsoluteTypeName]

3. 在这里如果要使用 args(String）匹配我们的 UserController 中的方法，则必须去掉方法定义中的 String name 参数,或将注解改成@After(value = "args(name)",argNames = "name")方可。否则会抛出异常 java.lang.IllegalArgumentException: error at ::0 formal unbound in pointcut

4. 如果我们需要配置多变量，我们可以使用 args(name,age)来配置，对应对象方法 Annotation_aspect.Record(String name,Integer age)和 UserController.login(String name,Integer age)。

5. 除了 args(),this(),target(),@args(),@within(),@target()和@annotation 等函数都可以指定参数名，来将目标连接点上的方法入参绑定到增强的方法中。

3. 目标类切点函数

函数	入参	说明	示例
within()	类名匹配串	定位于特定作用于下的所有连接点	within(com.yc.service.*ServiceImpl)，可以通过此注解为特定包下的所有以 ServiceImpl 名字结尾的类里面的所有方法添加事务控制。
target()	类名	定位于指定类及其子类	target(com.yc.service.IUserService),则可定位到 IUserService 接口和它的实现类如 UserServiceImpl
@within()	类型注解类名	定位与标注了特定注解的类及其实现类	@within(com.yc.controller.needRecord),比如我们可以在 BaseController 中标注@needRecord，则

			所有继承了 BaseController 的 UserController、ArticleController 等等都会被定位
@target()	类型注解类名	定位于标注了特定注解的目标类里所有方法	@target(com.yc.controller.needRecord)，则可以在 controller 层中，为我们需要日志记录的类标注@needRecord。

1. within()定位连接点的最细粒度是到类，相对于 execution()可定位连接点大到包，小到方法入参的适用范围更窄。
2. 相对于@within，显然@target 的耦合性更低，针对性更强。比如 UserController,ArticleControooler,HomeController 都需要继承 BaseController，如果我们在 BaseController 中标注@needRecord,则三个子 Controller 都会被@within 定位到织入增强，但实际上我们不想让 HomeController 织入增强，显然分别在 UserController 和 ArticleController 中标注@needRecord,然后利用@target 来织入增强才满足要求。
3. @within()如果标注在一个接口上，则不会匹配实现了该接口的子类。
4. 代理类切点函数

主要为 this()，大多数情况使用方法与 target()相同，区别在通过引介增强引入新接口方法时，新的接口方法同样会被 this()定位，但 target()则不会。

切点独立命名

在前面的讲解里，我们都是直接把切点配置到切点表达式函数里的，而这种切点叫做匿名切点。但假如我们有新的需求，要为相同的切点配置多个增强，这就需要我们在多个增强中配置相同的切点。为了提高重用性和降低维护成本，我们可以通过@Pointcut 注解来单独命名切点。

和前面相同的例子，我们将注解切面 Annotation_aspect 改造成如下：

```
@Aspect
public class Annotation_aspect {

    @Pointcut("execution(public * test.aop2.*.*(..))")
    private void ClassInTest_aop2(){}//修饰为 private 表示此切点只能在本类中使用，这里的返回值和方法题没有实际用途

    @Pointcut("args(String)")
    protected void MethodWithArgNames(){}//修饰为 protected 表示此切点只能在本类及其子类中使用

    @After("ClassInTest_aop2() and MethodWithArgNames() ")
    public void Record(JoinPoint joinPoint) throws Throwable{
        System.out.println("日志记录:用户在" + new SimpleDateFormat("yyyy-MM-dd hh:mm;ss").format(new Date()) + "调用了"+ joinPoint.getSignature().getDeclaringTypeName()+"方法" );
    }

}
```

运行测试函数，控制台打印：

I’m zenghao ,I’m logining