4 - Sep - 2012 initiated and completed

The intention of this blog is to explain **AspectJ** Pointcut Expressions in **Spring** Applications. In **Aspect Oriented Programming**, a pointcut is a set of joinpoints. A joinpoint is a point in program execution where you can add additional behavior. Spring applications only support method based joinpoints. So, you can use AspectJ pointcut expressions to define method pointcuts in Spring 2.x applications. Lets discuss some AspectJ pointcut expressions patterns.

Method Signature Patterns

The most typical pointcut expressions are used to match a number of methods by their signatures. A common method based pointcut expression is something like

```
expression(<method scope> <return type> <fully qualified class name>.*(parametes))
```

1.method scope: Advice will be applied to all the methods having this scope. For e.g., public, private, etc. Please note that Spring AOP only supports advising public methods.
2.return type: Advice will be applied to all the methods having this return type.
3.fully qualified class name: Advice will be applied to all the methods of this type. If the class and advice are in the same package then package name is not required
4.parameters: You can also filter the method names based on the types. Two dots(..) means any number and type of parameters.

Examples

•execution(* com.aspects.pointcut.DemoClass.*(..)) : This advice will be applied to all the methods of DemoClass.
•execution(* DemoClass.*(..)): You can omit the package if the DemoClass and the advice is in the same package.
•execution(public * DemoClass.*(..)): This advice will be applied to the public methods of DemoClass.
•execution(public int DemoClass.*(..)): This advice will be applied to the public methods of DemoClass and returning an int.
•execution(public int DemoClass.*(int, ..)): This advice will be applied to the public methods of DemoClass and returning an int and having first parameter as int.
•execution(public int DemoClass.*(int, int)): This advice will be applied to the public methods of DemoClass and returning an int and having both parameters as int.

Type Signature Patterns

These pointcut expressions are applied to all joinpoint of certain types. A common type signature patterns looks like

```
within(type name)
```

Here type name is either the package name or the class name.

not interface names ???
no, i tested. i cannot use interface without +

Examples

•within(com.aspects.blog.package.*) : This will match all the methods in all classes of com.aspects.blog.package.
•within(com.aspects.blog.package..*) : This will match all the methods in all classes of com.aspects.blog.package and its sub packages. The only difference is the extra dot(.) after package.
•within(com.aspects.blog.package.DemoClass) : This will match all the methods in the DemoClass.
•within(DemoClass) : Again, if the target class is located in the same package as this aspect, the package name can be omitted.
•within(DemoInterface+) : This will match all the methods which are in classes which implement DemoInterface.

this is enough for me

Bean Name Patterns

Spring 2.5 supports a new pointcut type that is used to match bean names. For example, the following pointcut expression matches beans whose name ends with Service.

```
bean(*Service)
```

This pointcut is not supported by AspectJ annotation, hence you can declare them only in spring context files.

Combining Pointcut Expressions

Pointcut expressions can be combined using && (and), ||(or), and !(not). For example,

```
within(DemoInterface1+) || within(DemoInterface2+)
```

The above patterns will match all join point in all classes which implement DemoInterface1 or DemoInterface2

Declaring Pointcut Parameters

One way to access join point information is by reflection (i.e., via an argument of type org.aspectj.lang.JoinPoint in the advice method). Besides, you can access join point information in a declarative way by using some kinds of special pointcut expressions. For example,

so, joint point information is read and given to advice method. super... no need to call any API to get those information.

```
1 @Aspect
2 public class SomeAspect {
3
4     @Before(&amp;quot;execution(* *.*(..)) &amp;amp;&amp;amp; target(target) &amp;amp;&amp;amp; args(a,b)&amp;quot;)
5     public void someMethod(Object target, int a, int b) {
6         log.info(&amp;quot;Target class : &amp;quot; + target.getClass().getName());
7         log.info(&amp;quot;Arguments : &amp;quot; + a + &amp;quot;, &amp;quot; + b);
8     }
9 }
```

Here, target captures the target object and args captures the parameters.

Applying arbitrary patterns

~~Although the syntax of the AspectJ pointcut expressions is pretty rich, you may find some scenarios in which they are not sufficient to provide the necessary behaviour.~~ In such scenarios, you may choose to create an annotation and use that to match joinpoints. This is done in the following manner

First create an annotation

this is best for me to debug.
but i have to touch the code to add this annotation

```
1 @Target(ElementType.Method)
2 @Retention(RetentionPolicy.RUNTIME)
3 @Documented
4 public @Interface ApplyAspect {
5 }
```

Once you have created this annotation, you can created pointcut expressions like

```
@annotation(ApplyAspect)
```

This will match those methods which are annotated with @ApplyAspect. You can also apply this to classes, for example

```
@within(ApplyAspect)
```

This will match all joinpoints in classes annotated with @ApplyAspect.

4 - Sep - 2012

Conclusion

In this blog, I have discussed the expressions for AspectJ pointcuts, which are powerful mechanism to filter the joinpoints.