(http://baeldung.com)

In n to Pointcut E in Spring

Last modified: July 20, 2017

by baeldung (http://www.baeldung.com/author/baeldung/)

Spring (http://www.baeldung.com/category/spring/) +

1. Overview

In this tutorial we will discuss the Spring AOP pointcut expression language.

We will first introduce some terminology used in aspect-oriented programming. A *join point* is a step of the program execution, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method

execution. A *pointcut* is a predicate that matches the join points and a *pointcut expression language* is a way of describing pointcuts programmatically.

2.

A p alue of the *@Pointcut* annotation:

```
1  amework.stereotype.Repository *)")
2  ods() {}
```

The intcut signature**. It provides a name that can be o that pointcut.
(/datadog)

```
@Around("repositoryClassMethods()")
public Object measureMethodExecutionTime(ProceedingJoinPoint pjp) throws
    ...
}
```

A pointcut expression could also appear as the value of the *expression* property of an *aop:pointcut* tag:

```
<aop:config>
    <aop:pointcut id="anyDaoMethod"
      expression="@target(org.springframework.stereotype.Repository)"/>
</aop:config>
```

3. Pointcut Designators

A pointcut expression starts with a **pointcut designator (PCD)**, which is a keyword telling Spring AOP what to match. There are several pointcut designators, such as the execution of a method, a type, method arguments, or annotations.

3.1 *execution*

The primary Spring PCD is *execution*, which matches method execution join points.

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

This y the execution of *findById* method of the *Foo* ery flexible. Suppose we would like to match all t ich may have different signatures, return type e may use wildcards:

```
dung.dao.FooDao.*(..))")
```

Her rn value, the second matches any method nam umber of parameters (zero or more).

3.2 *within*

Another way to achieve the same result from the previous section is by using the *within* PCD, which limits matching to join points of certain types.

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

We could also match any type within the *org.baeldung* package or a sub-package.

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

3.3 *this* and *target*

this limits matching to join points where the bean reference is an instance of the given type, while *target* limits matching to join points where the target object is an instance of the given type. The former works when Spring AOP creates a CGLIB-based proxy, and the latter is used when a JDK-based proxy is created. Suppose that the target class implements an interface:

```
public class FooDao implements BarDao {
    ...
}
```

In this case, Spring AOP will use the JDK-based proxy and you should use the *target* PCD because the proxied object will be an instance of *Proxy* class and implement the *BarDao* interface:

```
1                                    dao.BarDao)")
```


On lement any interface or *pro* then the proxied object will be a subclass of *Foo* :

```
1                                    o.FooDao)")
```

3.4 *@args* (Added in Spring)

This PCD is used for matching particular method arguments:

```
1 @Pointcut("execution(* *..find*(Long))")
```

This pointcut matches any method that starts with find and has only one parameter of type *Long*. If we want to match a method with any number of parameters but having the fist parameter of type *Long*, we could use the following expression:

```
1 @Pointcut("execution(* *..find*(Long,..))")
```

3.5 *@target*

The *@target* PCD (not to be confused with the *target* PCD described above) limits matching to join points where the class of the executing object has an annotation of the given type:

```
1 @Pointcut("@target(org.springframework.stereotype.Repository)")
```

3.6 *@args*

This PCD limits matching to join points where the runtime type of the actual arguments passed have annotations of the given type(s). Suppose that we want to trace all the methods accepting beans annotated with *@Entity* annotation:

```
1  op.annotations.Entity)")
2  ities() {}
```

To a ide a *JoinPoint* argument to the advice:

```
1  es()")
2  ntityAnnotatedBean(JoinPoint jp) {
3  s with @Entity annotation: " + jp.getArgs(
4
```


3.7 *@within*

This PCD limits matching to join points within types that have the given annotation:

```
1  @Pointcut("@within(org.springframework.stereotype.Repository)")
```

Which is equivalent to:

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

3.8 *@annotation*

This PCD limits matching to join points where the subject of the join point has the given annotation. For example we may create a *@Loggable* annotation:

```
1  @Pointcut("@annotation(org.baeldung.aop.annotations.Loggable)")
2  public void loggableMethods() {}
```

Then we may log execution of the methods marked by that annotation:

```
@Before("loggableMethods()")
public void logMethod(JoinPoint jp) {
    String methodName = jp.getSignature().getName();
    od: " + methodName);
```


4. xpressions

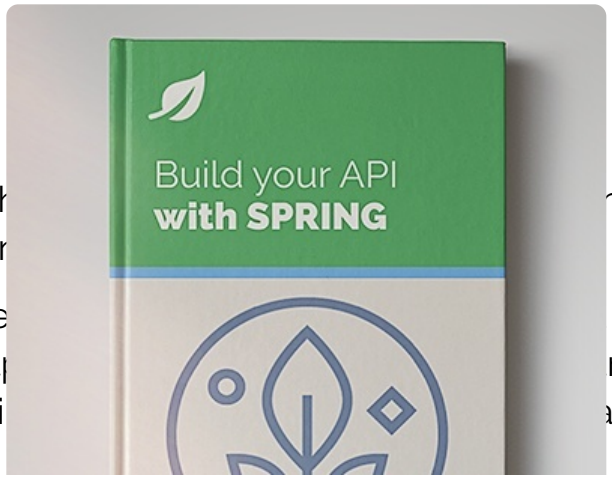
Poi using **&&**, **||** and **!** operators:

```
amework.stereotype.Repository)")
 {}

@Pointcut("execution(* *..create*(Long,..))")
public void firstLongParamMethods() {}

@Pointcut("repositoryMethods() && firstLongParamMethods()")
public void entityCreationMethods() {}
```

5.



In th ntcuts, we illustrated some examples of poi

The in my github project (htt ree/master/spring-mvc-java) – this is an Ecli sy to import and run as it is.

CATEGORIES

SPRING (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)

SECURITY (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO THE BASICS" TUTORIAL (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
HTTPCLIENT 4 TUTORIAL (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
SPRING PERSISTENCE TUTORIAL (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SECURITY WITH SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
THE COURSES (HTTP://COURSES.BAELDUNG.COM)
CONSULTING WORK (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
WRITE FOR BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACT (HTTP://WWW.BAELDUNG.COM/CONTACT)
COMPANY INFO (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TERMS OF SERVICE (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
PRIVACY POLICY (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORS (HTTP://WWW.BAELDUNG.COM/EDITORS)
MEDIA KIT (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)