



CS544 EA
Spring

AOP: Pointcut Expression Language

PointCut Express Language

- Written as a String
 - Part of the advice annotation (@Before / ...)
 - No compile time checking
 - If it doesn't match properly it fails silently
- Expressions can be combined with boolean operators
 - && (boolean and)
 - || (boolean or)
 - ! (boolean not)

Expressions

Pointcut expressions have to start with one of the following pointcut designators

- execution
- args
- within
- target
- @annotation
- @args
- @within
- @target

Execution

- `execution` is the most used designator

`execution(modifiers-pattern? ret-type-pattern declaring-type-pattern?name-pattern(param-pattern) throws-pattern?)`

Optional modifier
(public, protected, private)

Return type
* indicates any type

Optional type
(package and class)
ending in `.*` includes all classes in package
ending in `..*` includes classes in sub-packages

Method name
Use `.` to connect with type
May contain, or just be `*`

Parameters
Comma separated list
(`..`) means any parameters
(`*`) means one param any type
(`int, *`) = a int and one other type

Optional throws
Comma separated list of types
Optionally including `!` (not)

Execution Examples

- Examples from the Spring Documentation

See: <https://docs.spring.io/spring/docs/5.1.6.RELEASE/spring-framework-reference/core.html#aop-pointcuts-examples>

`execution(public * *(..))` // any public method

`execution(* set*(..))` // any method whose name starts with set

`execution(* com.xyz.service.AccountService.*(..))` // any method of AccountService

`execution(* com.xyz.service.*.*(..))` // any method of any class in the service package

`execution(* com.xyz.service..*.*(..))` // any method in the service package or sub packages

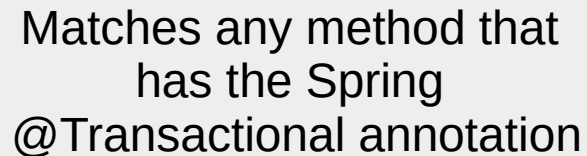
`execution(* *(int))` // any method taking a single int

`execution(* put*(String, int))` // any method starting with put, taking a String and an int

@annotation

- Matches any method that is annotated with the given annotation

`@annotation(org.springframework.transaction.annotation.Transactional)`



Matches any method that
has the Spring
@Transactional annotation

args and @args

- args(int, String)
 - Matches only methods that take an int and a String
- @args(org.springframework.stereotype.Service)
 - Matches only methods that take one object whose class is annotated as being a Service

within and @within

- `within(cs544.spring40.aop.CustomerService)`
 - Any method within this class
- `within(cs544.spring40..*)`
 - Any method within this package, or sub-packages
- `@within(org.springframework.stereotype.Service)`
 - Any methods within a class annotated as a Spring service

target and @target

- `target(cs544.spring40.aop.ICustomerService)`
 - Specifies what the type of the Target has to be
 - Type can be an interface (then matches all classes that implement)
 - Matches any methods in classes with the specified type
- `@target(org.springframework.stereotype.Service)`
 - Specifies annotation that the Target has to have
 - Matches any methods in classes annotated with it

Boolean Operators

- Boolean operators work as you would expect

```
package cs544.spring42.aop.boolops;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class TestAspect {
    private static Logger logger = LogManager.getLogger(TestAspect.class.getName());

    @Before("execution(* cs544.spring42.aop.boolops.CustomerService.*(..)) "
            + " && @target(org.springframework.stereotype.Service)")
    public void logBefore(JoinPoint joinpoint) {
        logger.warn("About to exec: " + joinpoint.getSignature().getName());
    }
}
```

Enforces that
CustomerService is
annotated with @Service

Named Pointcuts

```
package cs544.spring42.aop.boolops;
...
@Aspect
@Component
public class CheckOrderAspect {
    @Pointcut("execution(* cs544.spring42.aop.boolops.OrderService.*(..))")
    public void checkOrder() {
    }
    @Before("checkOrder()")
    public void checkOrder(JoinPoint joinpoint) {
        System.out.println("check order");
    }
    @After("checkOrder()")
    public void logOrderEvent(JoinPoint joinpoint) {
        System.out.println("log order event");
    }
}
```

```
@Service
public class OrderService implements IOrderService {
    @Override
    public void createOrder(Customer customer, ShoppingCart shoppingCart) {
        System.out.println("Create Order");
    }
    @Override
    public void deleteOrder(String ordernumber) {
        System.out.println("Delete Order");
    }
    @Override
    public void shipOrder(String ordernumber) {
        System.out.println("Ship Order");
    }
}
```

Named PointCut (other class)

```
package cs544.spring42.aop.boolops;
...
@Aspect
@Component
public class NamedPointCuts {
    @Pointcut("execution(* cs544.spring42.aop.boolops.OrderService.*(..))")
    public void checkOrder() {
    }
}
```

```
package cs544.spring42.aop.boolops;
...
@Aspect
@Component
public class CheckOrderAspect {
    @Before("NamedPointCuts.checkOrder()")
    public void checkOrder(JoinPoint joinpoint) {
        System.out.println("check order");
    }

    @After("NamedPointCuts.checkOrder()")
    public void logOrderEvent(JoinPoint joinpoint) {
        System.out.println("log order event");
    }
}
```