Spring AOP - why do i need aspectjweaver?

i wrote a very simple Aspect with Spring AOP. It works, but i have some problems understanding what is really going on. I don't understand why i have to add the aspectjweaver.jar? The Spring-AOP documentation clearly states that i don't need aspectj compiler or weaver as long as i just use Spring-AOP:

> The AOP runtime is still pure Spring AOP though, and there is no dependency on the AspectJ compiler or weaver.

My configuration looks like this:

```
<aop:aspectj-autoproxy />

@Aspect
@Service
public class RemoteInvocationAspect {

    @Before("execution(* at.test.mypackage.*.*(..))")
    public void test() {
        System.out.println("test");
    }
    ...
```

I also tried XML configuration, didn't change anything though. Maybe i could just let it go, but i really would like to understand why aspectj-weaver is used? If i don't add the dependency in maven i get `java.lang.ClassNotFoundException: org.aspectj.weaver.reflect.ReflectionWorld$ReflectionWorldException`

java spring aop aspectj spring-aop

asked Jul 12 '12 at 7:24

Mario B
1,014 1 14 31

Does your imports say that there is no reference to aspectj as well? Check the top of your document see what JARs you are referencing please :) – Gleeb Jul 12 '12 at 7:39

I just use imports from org.aspectj.lang.annotation.* which are within aspectjrt.jar. For that reason i understand why i need the rt, but why do i need the weaver as well? – Mario B Jul 12 '12 at 7:45

It seems like aspectjtools is doing the job as well. It has a better name than weaver. – Koray Tugay Nov 1 '17 at 9:01

5 Answers

Spring AOP implementation I think is reusing some classes from the aspectj-weaver. It still uses dynamic proxies - doesn't do byte code modification.

The following comment from the spring forum might clarify.

> Spring isn't using the AspectJ weaver in this case. It is simply reusing some of the classes from aspectjweaver.jar.
>
> -Ramnivas

answered Jul 12 '12 at 7:51

gkamal
16k 4 43 51

That seems to be correct. Good to know! Thanks for your help – Mario B Jul 12 '12 at 11:36

I recently had a similar question Why does spring throw an aspectj error if it does not depend on aspectj?

To use Spring AoP without an AspectJ dependency it must be done in xml. The annotations are a part of AspectJ.

Also, the really cool expression language is only supported by AspectJ. So you have to define explicit point-cuts. See Section 6.3.2. Declaring a pointcut:
http://static.springsource.org/spring/docs/2.0.x/reference/aop.html section

I'm still having trouble finding any elaborate documentation on this technique.

edited May 23 '17 at 11:53 answered May 20 '13 at 16:07

Community ♦
1 1

beta-brad
178 2 10

You are using AspectJ style pointcut-expression `@Aspect` and `@Before` are part of AspectJ. Check this link.

Regarding the `AspectJ-weaver` , its actually a bytecode weaver which **weaves aspects into classes at load time**.

answered Jul 12 '12 at 7:42

Santosh
14k 3 34 63

I understand that. Still, regarding to the documentation i should be able to use AspectJ style pointcuts without using aspectj-weaver. – Mario B Jul 12 '12 at 8:05

1 As people have pointed out here, spring is reusing some of classes from AspectJ. Those classes might require AspectJ specific bytecode weaving and hence aspectj-weaver. – Santosh Jul 12 '12 at 9:02

You need the aspectjtools or the aspectjweaver dependencies when you use the AspectJ pointcut expression language.

Please see the following classes:

Foo.java

```
public interface Foo {
    void foo();
    void baz();
}
```

FooImpl.java

```
public class FooImpl implements Foo {
    @Override
    public void foo() {
        System.out.println("Foo!");
    }

    @Override
    public void baz() {
        System.out.println("Baz!");
    }
}
```

MethodBeforeAdviceBarImpl.java

```
import org.springframework.aop.MethodBeforeAdvice;
import java.lang.reflect.Method;

public class MethodBeforeAdviceBarImpl implements MethodBeforeAdvice {
    @Override
    public void before(Method method, Object[] args, Object target) throws Throwable {
        System.out.println("Bar!");
    }
}
```

And please see **App.java** version - 1

```
import org.springframework.aop.MethodBeforeAdvice;
import org.springframework.aop.framework.ProxyFactory;
import org.springframework.aop.support.NameMatchMethodPointcutAdvisor;

public class App {

    public static void main(String[] args) {
        final MethodBeforeAdvice advice = new MethodBeforeAdviceBarImpl();

        final NameMatchMethodPointcutAdvisor nameMatchMethodPointcutAdvisor = new
NameMatchMethodPointcutAdvisor();
        nameMatchMethodPointcutAdvisor.setMappedName("foo");
        nameMatchMethodPointcutAdvisor.setAdvice(advice);

        final ProxyFactory proxyFactory = new ProxyFactory();
        proxyFactory.addAdvisor(nameMatchMethodPointcutAdvisor);

        final Foo foo = new FooImpl();
        proxyFactory.setTarget(foo);

        final Foo fooProxy = (Foo) proxyFactory.getProxy();
        fooProxy.foo();
        fooProxy.baz();
```

```
    }
}
```

The output of running this example will be:

```
Bar!
Foo!
Baz!
```

I only need the **org.springframework:spring-context.jar** in my classpath. Now instead of a NameMatchMethodPointcutAdvisor, lets use **AspectJExpressionPointcutAdvisor**:

```
import org.springframework.aop.MethodBeforeAdvice;
import org.springframework.aop.aspectj.AspectJExpressionPointcutAdvisor;
import org.springframework.aop.framework.ProxyFactory;

public class App {

    public static void main(String[] args) {
        final MethodBeforeAdvice advice = new MethodBeforeAdviceBarImpl();

        final AspectJExpressionPointcutAdvisor aspectJExpressionPointcutAdvisor = new
AspectJExpressionPointcutAdvisor();
        aspectJExpressionPointcutAdvisor.setAdvice(advice);
        aspectJExpressionPointcutAdvisor.setExpression("execution(void
biz.tugay.spashe.Foo.foo())");

        final ProxyFactory proxyFactory = new ProxyFactory();
        proxyFactory.addAdvisor(aspectJExpressionPointcutAdvisor);

        final Foo foo = new FooImpl();
        proxyFactory.setTarget(foo);

        final Foo fooProxy = (Foo) proxyFactory.getProxy();
        fooProxy.foo();
        fooProxy.baz();
    }
}
```

Again, if I only have the spring-context.jar in my classpath, I will get:

```
An exception occured while executing the Java class. null: InvocationTargetException:
org/aspectj/weaver/reflect/ReflectionWorld$ReflectionWorldException:
org.aspectj.weaver.reflect.ReflectionWorld$ReflectionWorldException
```

When you investigate the **AspectJExpressionPointcutAdvisor** class, you will see that it extends **AbstractGenericPointcutAdvisor** and which delegates the work to an instance of **AspectJExpressionPointcut**. And you can see that AspectJExpressionPointcut has the following import statements:

```
import org.aspectj.weaver.patterns.NamePattern;
import org.aspectj.weaver.reflect.ReflectionWorld.ReflectionWorldException;
import org.aspectj.weaver.reflect.ShadowMatchImpl;
import org.aspectj.weaver.tools.ContextBasedMatcher;
import org.aspectj.weaver.tools.FuzzyBoolean;
import org.aspectj.weaver.tools.JoinPointMatch;
import org.aspectj.weaver.tools.MatchingContext;
import org.aspectj.weaver.tools.PointcutDesignatorHandler;
import org.aspectj.weaver.tools.PointcutExpression;
import org.aspectj.weaver.tools.PointcutParameter;
import org.aspectj.weaver.tools.PointcutParser;
import org.aspectj.weaver.tools.PointcutPrimitive;
import org.aspectj.weaver.tools.ShadowMatch;
```

You will need the **aspectjtools** dependency in your classpath in runtime so AspectJExpressionPointcut can load the classes it needs.

answered Nov 1 '17 at 20:00

Koray Tugay

6,844 26 91 188

You can browse spring website and find the answer on page of docs.spring.io

> The @AspectJ support can be enabled with XML or Java style configuration. In either case you will also need to ensure that AspectJ's aspectjweaver.jar library is on the classpath of your application (version 1.6.8 or later). This library is available in the 'lib' directory of an AspectJ distribution or via the Maven Central repository.

answered Jul 7 '15 at 1:49

Richard Xue

117 1 4

This does not answer the question. – Koray Tugay Nov 1 '17 at 9:00