Spring AOP vs. AspectJ

Eberhard Wolff
http://ewolff.com
eberhard.wolff@gmail.com

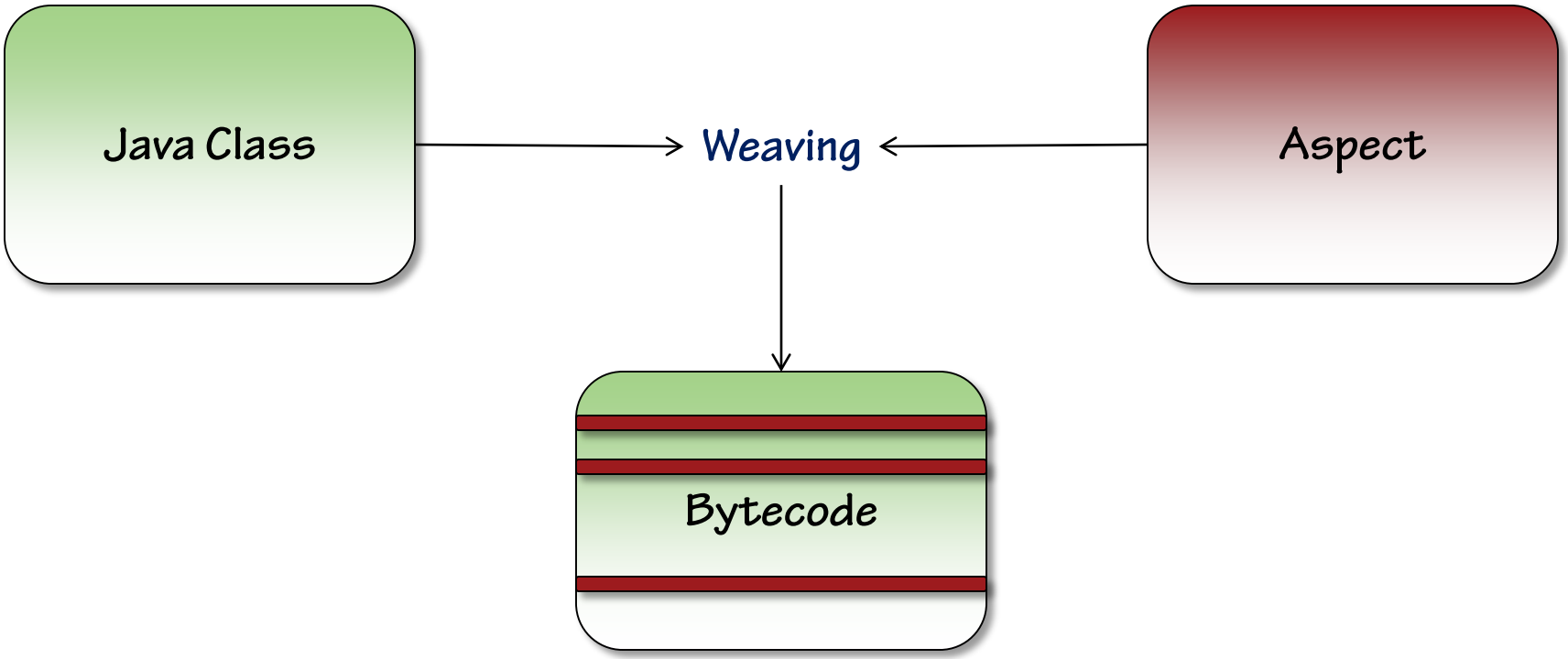
pluralsight
hardcore developer training

AspectJ

- **A different system for AOP for Java**
  - Faster
  - More powerful
- **Spring AOP uses the same syntax**
  - @Aspect
  - @Before, @After, @Pointcut etc
  - Pointcut expressions
- **Only bean Pointcut Expression won't work**
- **Aspect are applied differently for AspectJ**

AspectJ

- AspectJ uses Bytecode Weaving
- i.e. classes and aspect are both “woven” into the Bytecode
- Weaving might happen
  - When classes are loaded
  - When code is compiled



Load Time Weaving

- Aspect are woven when classes are loaded
- Weaving configured by META-INF/aop.xml

```
<aspectj>
  <weaver>
    <include within="com.ewolff..*" />
  </weaver>
  <aspects>
     <aspect name="com.ewolff.loadtimeweaving.DemoAspect" />
  </aspects>
</aspectj>
```

- AspectJ needs a hook to modify the Bytecode!

Load Time Weaving: AspectJ Java Agent

- **Java Agents can modify Bytecode etc**
- **Add -javaagent:pathto/aspectjweaver.jar to the java Command Line**

- **Spring also helps with AspectJ Load Time Weaving**
- **Load Time Weaving without an Agent**
  - Only certain environments (Tomcat, WebLogic etc.)

- **Springs own Agent**
  - Or use -javaagent:pathto/org.springframework.instrument-{version}.jar to the java Command Line
  - Load Time Weaving enabled per ClassLoader
- **Need to configure Spring**

Add LTW Support to Spring Configuration

- **XML**
  - Enable Load Weaving using <context:load-time-weaver/>

```
<beans …>
  <context:component-scan base-package="com.wolff" />
  <context:load-time-weaver />
</beans>
```

- **JavaConfig**
  - Use @EnableLoadTimeWeaving

```
@Configuration
@ComponentScan(basePackages="com.wolff")
@EnableLoadTimeWeaving
public class SystemConfiguration { }
```

- **Still need META-INF/aop.xml !**

Compile Time Weaving

- Replace Java compiler with AspectJ compiler
- Compiler weaves Aspects into the Bytecode

- Integrated support in Eclipse
- AspectJ is an Eclipse project

- Need to modify Maven pom.xml

Introduce Error / Warnings

- AspectJ can add compile error or warnings

```
@Aspect
public class DeclareErrorAspect {

  @DeclareError(
   "(call (* java.sql..*.*(..)) || call (* javax.sql..*.*(..)))" +
                                        "&& !within(com.ewolff..repository..*)")
  public static final String jdbcOnlyInRepository =
   "JDBC may only be used in Repositories!";

}
```

- String contains error message
- call : like execution
  - call : “this” is still the calling object
  - execution : “this” is the object that is called
  - call only works with AspectJ
- within : depends on code location

DeclareError demo

- Zeigen, was passiert, wenn man JDBC außerhalb eines Repositories nutzt
- Zweites DeclareError zeigen
- Dort wichtig: + für subclasses
- Und zeigen, was passiert, wenn man printStackTrace aufruf
- DeclareWarning zeigen

Compile Time Weaving vs. Load Time Weaving

- **Pros Compile Time Weaving**
  - No modification to the environment
  - No aop.xml needed
  - Faster application startup
  - Can add errors / warning

- **Cons Compile Time Weaving**
  - Cannot weave classes w/o source code
  - i.e. LTW can even modify JDK classes
  - Different compiler
  - Compiler slower (Eclipse AJDT helps)

- **Prefer Compile Time Weaving – the simplest approach**

AspectJ vs. Spring AOP

- **Pros AspectJ**
  - Performance much better
  - Clearer model – also indirect calls can be advised
  - Can also advise constructor, field access, protected methods etc

- **Cons AspectJ**
  - Spring AOP performance usually sufficient
  - Need to use different compiler / modify execution environment
  - Spring AOP is already built into Spring

- **AspectJ is more powerful and has better performance**

Summary

- AspectJ uses the same syntax as Spring AOP

- Load Time Weaving: Modify Bytecode when loaded
- Compile Time Weaving: Modify Bytecode when compiled

- AspectJ is faster and more powerful than Spring