Expressing Architecture Using Pointcuts

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

pluralsight
hardcore developer training

Architecture Problems

- **Architecture often described in documents**
- **For each call to a service**
  - Call must be traced
  - Exceptions must be logged

- **Not read**
- **Not followed**
- **Lots of boilerplate code**

- **AOP to the rescue!**

- **Architecture in code!**

Prejudices Against AOP

- AOP adds random code to random parts of the system
- It is hard to reason about the system
- What happens when?

- Not true if you use AOP properly
- Will show concept in this module

Architecture Problems

- **For each call to a service**
  - Call must be traced
  - Exceptions must be logged
- **For each call to a repository**
  - Call must be traced
  - Performance must be traced
  - Exceptions must be logged
- **Specific behavior should be added**
  - Tracing, exception handling etc
- **to specific parts of the the architecture**
  - Repositories, services etc

Proper Use of AOP

- Idea: Add behavior to parts of the architecture using AOP

- Step 1 : Define architecture as Pointcuts
- Step 2 : Define behavior using Advices
- Step 3 : Add Advices to correct Pointcuts

- Result: No more technical boiler plate

Architecture Using Annotations

- **Use an annotation for each part of the architecture**

- **Might use annotations from org.springframework.stereotype**
  - @Service for a service
  - @Repository for a repository
  - @Component
- **No difference to Spring dependency injection**

Pointcuts Using Annotations

Step 1: Define architecture as Pointcuts

```
public class SystemArchitecture {

  @Pointcut(                                  Any class annotated with @Repository
   "execution(* (@org.springframework.stereotype.Repository *).*(..))")
  public void Repository() {}

  @Pointcut(                                  Any class annotated with @Service
   "execution(* (@org.springframework.stereotype.Service *).*(..))")
  public void Service() {}

}
```

Using the Pointcut

Step 2 : Define behavior using Advices

Exceptions must be logged

Step 3 : Add Advices to correct Pointcuts

Services and Repositories

```
@Component
@Aspect
public class ExceptionLoggingAspect {

 Logger logger = LoggerFactory.getLogger(ExceptionLoggingAspect.class);

  @AfterThrowing(pointcut =
   "SystemArchitecture.Repository() || SystemArchitecture.Service()",
   throwing = "ex")
  public void logException(Exception ex) {
    logger.error("Exception", ex);
  }

}
```

Pointcuts Using Package Names

The rest of the code stays the same!!

```
public class SystemArchitecture {

        Any class in a subpackage repository of com.ewolff
        @Pointcut("execution(* com.ewolff..repository.*.*(..))")
        public void Repository() {
        }

         Any class in a subpackage service of com.ewolff
        @Pointcut("execution(* com.ewolff..service.*.*(..))")
        public void Service() {
        }

}
```

More Fun With Pointcuts

- Configure Spring's Dependency Container
- No dependencies on Spring in the code

```
<beans ...
  default-autowire="constructor">

  <context:component-scan base-package="com.ewolff"
   use-default-filters="false">
    <context:include-filter type="aspectj"
     expression="com.ewolff..service.*" />
    <context:include-filter type="aspectj"
     expression="com.ewolff..repository.*" />
    <context:include-filter type="annotation"
     expression="org.aspectj.lang.annotation.Aspect" />
  </context:component-scan>

  <aop:aspectj-autoproxy />

</beans>
```

AspectJ type expression

@Aspect annotated class become Spring Beans
Could use your own annotations

Summary

- **AOP adds behavior to specific parts of the system**

- **Pointcuts can express architecture**
- **Should be added to the project early**

- **Powerful tool**
  - Architecture in code – not in documentation
  - AspectJ expressions can be used to configure Spring DI container
  - Avoid boilerplate code!