Spring's Aspect Library

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

pluralsight
hardcore developer training

Spring's Aspect Library

- Spring offers a lot of aspects
- Often no need to implement your own
- High quality code provided to you
- Save time & effort

Tracing

```
public void doSomething() {
  final String METHODNAME = "doSomething";
  logger.trace("entering " + CLASSNAME + "." + METHODNAME);
  TransactionStatus tx = transactionManager.getTransaction(
    new DefaultTransactionDefinition());
  try {
    // Business Logic
  } catch (RuntimeException ex) {
    logger.error("exception in "+CLASSNAME+"."+METHODNAME, ex);
    tx.setRollbackOnly();
    throw ex;
  } finally {
    transactionManager.commit(tx);
    logger.trace("exiting " + CLASSNAME + "." + METHODNAME);
  }
}
```

Tracing

Tracing

```
<beans …>

 <context:component-scan base-package="com.ewolff" />

 <bean id="debugInterceptor"
  class="org.springframework.aop.interceptor.DebugInterceptor"
 />

 <aop:config>
  <aop:advisor
   advice-ref="debugInterceptor"
   pointcut=
    "SystemArchitecture.Service() || SystemArchitecture.Repository()"
   />
 </aop:config>

</beans>
```

Predefined aspect

XML AOP configuration

Configure "legacy" aspect

Other Tracing Aspects

- All in org.springframework.aop.interceptor
- CustomizableTraceInterceptor: Can customize the trace output
- SimpleTraceInterceptor: Basic information
- DebugInterceptor: Full information
- PerformanceMonitorInterceptor: Uses a StopWatch to measure performance
- Milliseconds only
- You don’t to implement your own tracing!

Other Aspect

- AsyncExecutionInterceptor to process method calls asynchronously
- ConcurrencyThrottleInterceptor to limit the number of threads in an object
- Spring Security includes aspects for security
- Very powerful
- Also supports AspectJ
- CacheInterceptor for caching results of method calls

Transaction

```
public void doSomething() {
  final String METHODNAME = "doSomething";
  logger.trace("entering " + CLASSNAME + "." + METHODNAME);
  TransactionStatus tx = transactionManager.getTransaction(
    new DefaultTransactionDefinition());
   try {
     // Business Logic
   } catch (RuntimeException ex) {
     logger.error("exception in "+CLASSNAME+"."+METHODNAME, ex);
     tx.setRollbackOnly();
     throw ex;
   } finally {
     transactionManager.commit(tx);
     logger.trace("exiting " + CLASSNAME + "." + METHODNAME);
   }
}
```

Transactions

Transaction

```
@Transactional
public void doSomething() {
  final String METHODNAME = "doSomething";
  logger.trace("entering " + CLASSNAME + "." + METHODNAME);
  try {
     // Business Logic
   } catch (RuntimeException ex) {
     logger.error("exception in "+CLASSNAME+"."+METHODNAME, ex);
     throw ex;
   } finally {
     logger.trace("exiting " + CLASSNAME + "." + METHODNAME);
   }
}
```

Can add @Transactional annotation

To method or class

Could we use pointcuts?

Transactions

```
<beans …>

  <context:component-scan base-package="com.ewolff" />

  <tx:advice id="txAdvice"
   transaction-manager="transactionManager">
    <tx:attributes>
     <tx:method name="find*" read-only="true" />
     <tx:method name="*" />
    </tx:attributes>
   </tx:advice>

   <aop:config>
     <aop:advisor advice-ref="txAdvice"
                  pointcut=
       "SystemArchitecture.Service()||SystemArchitecture.Repository()"
      />
    </aop:config>

</beans>
```

Summary

- Spring provides a library of aspects
- Include tracing, transactions…
- Using pointcuts enterprise services can be transparently added to the business logic