AOP in Enterprise

Ramnivas Laddad
Principal, SpringSource
Author, AspectJ in Action
ramnivas.laddad@springsource.com

About Speaker

- Principal at SpringSource
- Spring framework committer
- Author of books and articles
  - AspectJ in Action
- Speaker at many professional conferences
  - No Fluff Just Stuff, JavaOne, JavaPolis, Spring Experience, SpringOne, Software Development, EclipseCon, O’Reilly etc.
- Active involvement in AspectJ since its early form
What is AOP
And Why You Must Know It?

A quick intro to AOP

• Crosscutting concerns
  – Functionality whose implementation spans multiple modules
  – Many examples: Logging and tracing, transaction management, security, caching, error handling, business rules, performance monitoring...

• AOP
  – A programming methodology to help with crosscutting concerns
System Evolution: Conventional

System Evolution: AOP based
Core AOP concepts

- **Join point**
  - An identifiable point in the execution of a program.
  - Central, distinguishing concept in AOP
- **Pointcut**
  - Program construct that selects join points and collects context at those points
- **Advice**
  - Code to be executed at a join point that has been selected by a pointcut
- **Inter-type declarations**
  - Additional data or method to existing types, implementing new interfaces

Tracing Aspect

```java
public aspect TracingAspect {
    private static Logger _logger
        = Logger.getLogger(TracingAspect.class);

    public pointcut traced() : execution(* *(..));

    before() : traced() {
        Signature sig = thisJoinPoint.getSignature();
        _logger.logp(Level.INFO,
            sig.getDeclaringType().getName(),
            sig.getName(), "Entering");
    }
}
```
The @AspectJ syntax

```java
@Aspect
public class TracingAspect {
    private static Logger _logger
        = Logger.getLogger(TracingAspect.class);

    @Pointcut("execution(* *(..))")
    public void traced() {

    @Before("traced()")
    public void log(JoinPoint thisJoinPoint) {
        Signature sig = thisJoinPoint.getSignature();
        _logger.logp(Level.INFO,
                sig.getDeclaringType().getName(),
                sig.getName(), "Entering");
    }
}
```

XML Syntax:
Defining Base aspect

```java
public abstract aspect MonitoringAspect { public abstract pointcut monitored();

Object around() : monitored() {
    long startTime = System.nanoTime();
    Object ret = proceed();
    long endTime = System.nanoTime();

    monitorAgent.record(thisJoinPointStaticPart,
            endTime-startTime);

    return ret;
} ...
```
XML syntax:
Defining concrete aspects

<aspect>
  <aspects>
    <concrete-aspect>
      name="ajia.monitoring.JDBCMonitoringAspect"
      extends="ajia.monitoring.MonitoringAspect">
    </concrete-aspect>
    ...
  </aspects>
  ...
</aspect>

AspectJ weaving options

- Compile-time weaver
  - Aspect and class source code ➔ class files
- Binary weaver ("linker")
  - Aspect and class source/binary files ➔ class files
- Load-time weaver
  - Aspect and class binary files
    - Weave class files when being loaded into VM
Load-time weaving

- Weave while classes are being loaded
  - No explicit compile-time or binary weaving needed
- JVMTI agent

```java
java -javaagent:aspectjweaver.jar ...
```
  - Application-server agnostic
  - Java 5.0+ VMs
LTW deployment: aop.xml

```xml
<aspectj>
  <aspects>
    <aspect name="ajia.security.Security"/>
    <aspect name="ajia.tracing.ExceptionTracer"/>
    <concrete-aspect
      name="ajia.monitoring.JDBCMonitoringAspect"
      extends="ajia.monitoring.MonitoringAspect">
      <pointcut name="monitored"
        expression="call(* java.sql.*.*(..))"/>
    </concrete-aspect>
    ...
  </aspects>
  ...
</aspectj>
```

Spring-driven LTW

- **Without** launch script modifications *(no -javaagent)*
  - Tomcat 5.x+
  - WebLogic 10+
  - OC4J 10.x+
  - More may be supported in future
- **With** Spring’s JPA agent
  - Any application- or web-server running on Java 5
- Configuration

```xml
<context:load-time-weaver/>
```
AspectJ in Enterprise
Tapping power of AspectJ weaver

Layering violation enforcement

• Problem
  – UI layer often directly use DAO layer contrary to architectural policies
  – No automatic enforcement

• Solution
  – One simple, reusable aspect to enforce the policies
    • Just add to your build system
Layering policy enforcement

```java
public aspect LayeringPolicyEnforcementAspect {
    pointcut dataAccess() :
        call(* ajia.dao..*.*(..))
        || call(* org.hibernate..*.*(..))
        || call(* ..sql..*.*(..));

    declare error : dataAccess()
        && within(ajia.ui..*)
        : "UI layer shouldn't directly access data;
        Use business layer instead";
}
```

First failure data capture (FFDC)

- **Problem:**
  - Exception at low level trickles through layers; however, original context is lost
- **Solution:**
  - FFDC aspect logging the exception at the first failure site
**FFDC: Base aspect**

```java
public abstract aspect FFDC {
    public abstract pointcut ffdcOp();

    after() throwing(Exception ex) : ffdcOp() {
        if(_logger.isLoggable(Level.WARNING)) {
            Signature sig = thisJoinPoint.getSignature();
            logFFDC(sig.getDeclaringType().getName(),
                    sig.getName(), ex,
                    thisJoinPoint.getThis(),
                    thisJoinPoint.getArgs());
        }

        public void logFFDC(String className, String methodName,
                             Exception ex, Object thiz, Object[] args) {
            ...
        }
    }
}
```

**FFDC: Derived aspect**

```java
public aspect BlogFFDC extends FFDC {
    public pointcut ffdcOp()
        : execution(* com.mycompany.service.*.*(..))
            || execution(* com.mycompany.dao.*.*(..));
}
```
Spring for Configuring Aspects

- AspectJ have dependencies
  - Just the same way as any “bean”
- Spring is good at configuration
  - Configure aspects as a Spring bean

public aspect CacheAspect {
    private Cache cache;
    private boolean enabled = true;

    public pointcut cachedAccess(Cachable cachable) :
        execution(* Cachable * *(..)) && @annotation(cachable);

    Object around(Cachable cachable) : cachedAccess(cachable) {
        ... Caching logic ...
    }

    public void setCache(Cache cache) {
        this.cache = cache;
    }

    public boolean getEnabled() {
        return this.enabled;
    }

    public void setEnabled(boolean enabled) {
        this.enabled = enabled;
    }
}
Configuring caching aspect

```xml
<beans ...>
  <bean id="cacheAspect" class="ajia.CacheAspect"
       factory-method="aspectOf">
    <property name="cache" ref="cache"/>
  </bean>

  <bean id="cache"
       class="com.opensymphony.oscache.base.Cache">
    ... Cache properties ...
  </bean>
</beans>
```

Exposing aspect through JMX

```xml
<beans ...>
  <bean id="cacheAspect" ...>
  <bean id="cache" ...>

  <bean id="cacheAspect"
       class="org.springframework.jmx.export.MBeanExporter">
    <property name="beans">
      <map>
        <entry key="ajia:service=cacheAspect"
               value-ref="cacheAspect"/>
        <entry key="ajia:service=cache"
               value-ref="cache"/>
      </map>
    </property>
  </beans>
</beans>
```
Transaction management

• Issue:
  – How to designate methods
  – How to specify transaction attributes

• Solution:
  – Annotations to designate methods
  – Annotation properties to specify transaction attributes

Making a service transactional

```java
@Transactional
class DefaultAccountService implements AccountService {

    public void debit(Account account, float amount) {
        //
    }

    @Transactional(readOnly=true)
    public float getBalance(Account account) {
        //
    }

    ...
}
```
Transaction management: XML configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

  <bean id="accountService"
       class="service.DefaultAccountService" />

  <bean id="txManager"
       class="o.s...DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
  </bean>

  <tx:annotation-driven
       transaction-manager="txManager"/>

</beans>
```

Authorization

- **Problem:**
  - Many methods in the system needs to check for authorized access
- **Solution:**
  - An aspect using Spring Security as the underlying authorization mechanism
Authorization: Reusable base aspect

```java
public abstract aspect AbstractAuthAspect {
    // Dependencies
    private AccessDecisionManager accessDecisionManager;
    private ObjectDefinitionSource objectDefinitionSource;

    public abstract pointcut authOperations();

    before() : authorizationOperations() {
        ConfigAttributeDefinition cad
            = objectDefinitionSource.getAttributes(thisJoinPoint);
        Authentication auth
            = SecurityContextHolder.getContext().getAuthentication();
        accessDecisionManager.decide(auth, null, cad);
    }

    ...
}
```

Authorization: Derived aspect

```java
public aspect AnnotationDrivenAuthAspect extends AbstractAuthAspect {
    public pointcut authOperations() :
        execution(@Secured * *(..))
            || execution(* (@Secured *).*(..));
}
```
Configuring aspect

```xml
<bean id="authenticatonAspect"
   class="ajla.AnnotationDrivenAuthAspect"
   factory-method="aspectOf">
   <property name="accessDecisionManager"
             ref="accessDecisionManager" />
   <property name="objectDefinitionSource"
             ref="annotationObjectDefinitionSource"/>
</bean>
```

Injecting dependencies using Aspect

- Limitations of Spring’s “traditional” DI
  - Injection limited to beans created through configuration
  - Not sufficient for objects created thru other mechanisms: Hibernate, JDO, fine grained objects
  - Prevents richer domain models prescribed by domain-driven design (DDD)
- Solution: Use aspects to inject dependencies
Dependency declaration: Based on template class

```java
@Configurable
public class Order {
    private Emailer emailer;

    public void process() {
        ...
        emailer.send(...)
    }

    public void setEmailer(Emailer emailer) {
        this.emailer = emailer;
    }
}
```

Dependency configuration: Based on template class

```xml
<beans ...>
    <bean id="smtpEmailer" ...

    <bean class="example.Order" scope="prototype">
        <property name="emailer" ref="smtpEmailer"/>
    </bean>

    <context:spring-configured/>
</beans>
```
Spring AOP
Making Simple Things Simple
And
Powerful Things Possible

AspectJ for Spring

- A proxy-based AOP framework
  - JDK proxies
  - CGLIB proxies
- Spring AOP provides pragmatic solution for typical enterprise applications
  - Use AspectJ, when more power is needed
- Multiple levels of integration
  - Support for @AspectJ syntax
  - Spring-driven weaving
AspectJ weaving

Spring AOP schematic
Dependency injection

AccountController

<table>
<thead>
<tr>
<th>AccountService</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOP Proxies</td>
</tr>
</tbody>
</table>

Power of AspectJ in Spring:
Syntax without weaver

- Proxy-based mechanism—no special compiler
  - @AspectJ aspects
    - Express aspects with little XML
  - Schema-style aspect
    - XML-based aspects definition with plain Java classes
  - AspectJ pointcut expression
    - Easier and expressive way to select pointcuts with old-style Spring AOP
@AspectJ integration

```java
package example;

@Aspect
public class EmailLogger {
    @Before("execution(* send(String, String))
            && args(address, *)")
    public void log(JoinPoint.StaticPart tjsp, String address) {
        System.err.println("Invoking " + tjsp.getSignature() + " for " + address);
    }
}
```

Using @AspectJ aspect

```xml
<beans ...>
        <aop:aspectj-autoproxy/>
        <bean id="emailerLogger"
                class="example.EmailLogger"/>
        ... Other beans ...
</beans>
```
Choosing Spring vs. AspectJ AOP

- Use Spring AOP when
  - Method-only interception is sufficient
    - Full power of AOP overwhelming
  - Don’t want to use special compiler
  - Domain object’s don’t need to be crosscutted
  - Pre-written aspects meet your needs
- Use AspectJ AOP when
  - Otherwise...
Summary

- Spring AOP is simple, yet powerful way to modularize crosscutting concerns
- AspectJ integration bring full power of AOP in the Spring environment
- Incremental path from Spring AOP to AspectJ AOP is available

The fun has just began...

Resources

- Spring books
  - *Pro Spring*, by Rob Harrop and Jan Machacek
  - *Professional Java Development with the Spring Framework*, by Rod Johnson, Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu
  - *Spring in Action*, by Craig Walls
- AspectJ books
  - *AspectJ in Action*, by Ramnivas Laddad
  - *Eclipse AspectJ*, by Adrian Colyer, Andy Clement, George Harley, Matthew Webster
- AOP@Work series
  - [http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=aop@work](http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=aop@work)
Questions?

Ramnivas Laddad
ramnivas.laddad@springsource.com