AspectJ and Spring 2.0

Ramnivas Laddad
Principal, Interface21
Author, AspectJ in Action
ramnivas.laddad@interface21.com
Speaker introduction

- Principal at Interface21
  - Specializing in aspect-oriented programming and enterprise Java
- Author of books and articles
  - AspectJ in Action: Practical Aspect-Oriented Programming
  - Several articles on IBM developerWorks, TheServerSide, JavaWorld and other publications
- Speaker at many professional conferences
  - No Fluff Just Stuff, JavaOne, JavaPolis, Software Development, EclipseCon, O’Reilly OSCON etc.
- Active involvement in AspectJ since its early form
Agenda

An introduction to AOP
AspectJ AOP
Spring AOP
Spring AspectJ integration
Demos
Q&A
Agenda

An introduction to AOP

AspectJ AOP
Spring AOP
Spring AspectJ integration
Demos
Q&A
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

ATM

Account

...
System Evolution: AOP based

ATM

Account

Logging aspect

Access control aspect

[Diagram showing ATM, Account, and aspects with colors for respectively logging and access control]
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

- **Introduction**
  - Additional data or method to existing types, implementing new interfaces
Dynamic crosscutting: Advice

- Code to be executed at a join point that has been selected by a pointcut
- Three kinds:
  - **Before**
    ```
    before() : call(* ATM.*(..)) {
        ...
    }
    ```
  - **After**
    ```
    after(Account account) returning : accountOperation(account) {
        ...
    }
    ```
  - **Around**
    ```
    Object around() : call(* Remote+.*(..) throws RemoteException) {
        try {
            ... proceed(); ... Retry logic ...
        } catch (...) {
        }
    }
    ```
Static crosscutting

- Introduce new parent types
  
  ```java
  declare parents: banking..* implements Loggable;
  ```

- Introduce new members
  
  ```java
  public Logger Loggable._logger;
  ```

- Soften exceptions
  
  ```java
  declare soft: SQLException
    : within(banking.model..*);
  ```

- Compile-time errors and warning
  
  ```java
  declare error
    : call(* Thread.*(..))
    & within(javax.ejb.EnterpriseBean+)
    : "Use of thread disallowed from EJBs";
  ```
Agenda

An introduction to AOP
AspectJ AOP
Spring AOP
Spring AspectJ integration
Demos
Q&A
More power with AspectJ

- Spring AOP provides pragmatic AOP for typical enterprise applications
- When Spring’s AOP isn’t enough
  - Use AspectJ
- 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
public aspect BankLoggingAspect {
    private static Logger _logger
        = Logger.getLogger("banking");

    public pointcut loggedOperations() : execution(* banking..*..*(..))
        && !within(BankLoggingAspect);

    before() : loggedOperations() {
        Signature sig =
            thisJoinPointStaticPart.getSignature();
        _logger.logp(Level.INFO,
            sig.getDeclaringType().getName(),
            sig.getName(), "Entering");
    }
}
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
public aspect LayeringPolicyEnforcementAspect {
  pointcut dataAccess() {
    call (* com.mycompany.dao..*.*(..)) || 
    call (* org.hibernate..*.*(..)) || 
    call (* *..sql..*.*(..));

  declare error : dataAccess() {
    && within(com.mycompany.ui..*) : "UI layer shouldn’t directly access data; Use business layer instead";
  }
}
public aspect RepositoryCrossContaminationEnforcement {
    pointcut inRoot1()
        within (com.mycompany.root1..*);

    declare error: set (com.mycompany.root2..* .*.*
        && inRoot1());
        : "Objects in the root1 should not hold onto objects from other roots";
}

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

    after() throwing(Exception ex) : ffdcOp() {
        if (_logger.isLoggable(Level.WARNING) {
            Signature sig = thisJoinPointStaticPart.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) {
        ...
    }
}
public aspect BlogFFDC extends FFDC {
    public pointcut ffdcOp() {
        execution(* com.mycompany..*.*(..)
            throws RemoteException) || execution(* com.mycompany..dao.*.*(..));
    }
}
Authorization

- **Problem:**
  - Many methods in the system need to check for authorized access

- **Solution:**
  - An aspect using JAAS as the underlying authorization mechanism
public abstract aspect AbstractAuthAspect {
    public abstract pointcut authorizedOperations();
    public abstract Permission getPermission(
        JoinPoint.StaticPart jsp);

    before() : authorizedOperations() {
        AccessController.checkPermission(
            getPermission(thisJoinPointStaticPart));
    }
}
public aspect BankingAuthAspect extends AbstractAuthAspect {
    public pointcut authorizedOperations() :
        execution(public * banking..Account.*(..));

    public Permission getPermission(JoinPoint.StaticPart jpsp) {
        return new BankingPermission(jpsp.getSignature().getName());
    }
}
Mix-in implementation

- Problem:
  - Boilerplate implementation of an implementation

- Solution:
  - Provide the default implementation using static crosscutting
public interface ServiceCenter {
    public String getId();
    public void setId(String id);

    public String getAddress();
    public void setAddress(String address);
}

public interface ServiceCenterMixin extends ServiceCenter {
    static aspect IMPL {
        private String ServiceCenterMixin.id;
        private String ServiceCenterMixin.address;

        public String ServiceCenterMixin.getId() {
            return id;
        }

        public void ServiceCenterMixin.setId(String id) {
            this.id = id;
        }

        public String ServiceCenterMixin.getAddress() {
            return address;
        }

        public void ServiceCenterMixin setAddress(String address) {
            this.address = address;
        }
    }
}
public class ATM extends Teller
    implements ServiceCenterMixin {
    ... No set/getId(), set/getAddress() 
    ... No id, address
}

public class BrickAndMortar extends Teller
    implements ServiceCenterMixin {
    ... No set/getId(), set/getAddress() 
    ... No id, address
}
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
@Configurable("emailerClient")

public class Order {

    private Eemailer emailer;

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

    public void setEmailer(Emailer emailer) {
        this.emailer = emailer;
    }
}
Dependency configuration: Based on template id

```xml
<beans>

  <aop:spring-configured/>

  <bean id="smtpEmailer" .../>

  <bean id="emailerClient" scope="prototype">
    <property name="emailer" ref="smtpEmailer"/>
  </bean>

</beans>
```
Dependency declaration: Based on template class

```java
@Configurable
class Order {
    private Eemailer emailer;

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

    public void setEmailer(Eemailer emailer) {
        this.emailer = emailer;
    }
}
```
<beans...>

<aop:spring-configured/>

<bean id="smtpEmailer" .../>

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

</beans>
Agenda

An introduction to AOP
AspectJ AOP
Spring AOP
Spring AspectJ integration
Demos
Q&A
Spring AOP basics

- A proxy-based AOP framework
  - JDK proxies
  - CGILIB proxies
- Targets many J2EE use cases
- Powerful integration with AspectJ
  - Allowing tapping into full AOP features
Spring AOP schematic

- Pointcut
- Advice
- Advisor
- Target Object
- Proxy
Configuring through XML: Defining advisor

```xml
<bean id="spamPreventionAdvisor"
     class="o.s.aop.support.DefaultPointcutAdvisor">

    <property name="advice">
        <bean class="example.SpamPreventionInterceptor"/>
    </property>

    <property name="pointcut">
        <bean class="o.s.aop.support.JdkRegexpMethodPointcut">
            <property name="pattern" value=".*send.*"/>
        </bean>
    </property>

</bean>
```
Configuring through XML: Creating proxy

```xml
<bean id="emailerTarget" class="example.EmailerImpl"/>

<bean id="emailer" class="o.s.aop.framework.ProxyFactoryBean">
    <property name="target" ref="emailerTarget"/>
    <property name="interceptorNames">
        <list>
            <value>spamPreventionAdvisor</value>
        </list>
    </property>
    <property name="proxyInterfaces">
        <list>
            <value>example.Emailer</value>
        </list>
    </property>
</bean>
```

- **Target**
- **Advice/Advisor**
- **Proxy interface (optional)**
package example;

public class Main {
    
    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
        Eemailer emailer = (Eemailer)context.getBean("emailer");
        emailer.send("ramnivas.laddad@interface21.com", "Hi! Your paypal account...");
        emailer.send("ramnivas.laddad@interface21.com", "Hi, I have an AspectJ question...");

    }

}
Agenda

An introduction to AOP
AspectJ AOP
Spring AOP
Spring AspectJ integration
Demos
Q&A
Where AspectJ helps Spring AOP

- AspectJ Expression Pointcut
- Typed Advice
- @AspectJ Aspects
- Target Object
- Proxy
Spring-AspectJ integration

- Provides power of AspectJ in Spring
- Levels of integration
  - Proxy-based mechanism—no special compiler
    - AspectJ pointcut expression
      - Easier and expressive way to select pointcuts
    - Typed advice
      - XML-based aspects definition with plain Java classes
    - @AspectJ aspects
      - Express aspects with little XML
  - Special compiler or weaver
    - Full power AspectJ
AspectJ pointcut expression

- **AspectJExpressionPointcut**
  - Still uses proxy-based mechanism
    - Limitation of method-only interception

```xml
<bean id="spamPreventor"
      class="o.s.aop.support.DefaultPointcutAdvisor">
  <property name="advice">
    <bean class="example.SpamPreventionInterceptor"/>
  </property>
  <property name="pointcut">
    <bean class="o.s.aop.aspectj.AspectJExpressionPointcut">
      <property name="expression" value="execution(* send(..))"/>
    </bean>
  </property>
</bean>
```
Typed advice

- Problem: MethodInterceptor and its cousins take parameters of type `Object`.
- Solution: Use AspectJ style type-safe interception
Typed advice: Defining aspect

```xml
<beans ...>
<aop:config>
  <aop:aspect id="emailLoggerBean"
    ref="emailerLogger">
    <aop:before
      pointcut="execution(* send(String, String))
      and args(address, *)"
      method="log"/>
  </aop:aspect>
</aop:config>

<bean id="emailerLogger"
  class="example.EmailLogger"/>
...
</beans>
```
package example;

public class EmailLogger {
    public void log(JoinPoint.StaticPart tjpsp, String address) {
        System.err.println("Invoking " + tjpsp.getSignature() + " for " + address);
    }
}

void before(Method method, Object[] args, Object target) throws Throwable;
package example;

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

    <aop: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...
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

- **DI using aspect**
  - Adrian Coyler, Dependency injection with AspectJ and Spring
Core AOP:
Simplifying Enterprise Applications with AOP

November 7-10, Washington, DC

http://interface21.com/training
Spring Conference: The Spring Experience 2006
December 7th – 10th, Hollywood Florida
by Interface21 and NoFluffJustStuff Java Symposums

◆ World-class technical conference for the Spring community
◆ 3 full days, 5 concurrent tracks, 60 sessions
  • Core Spring 2.0
  • Core Enterprise 2.0
  • Core Web 2.0
  • Domain Driven Design
  • Just Plain Cool
◆ Enjoy five-star beach resort and amenities
◆ Converse with core Spring team and industry experts
  ◆ Rod Johnson, Adrian Colyer, Ramnivas Laddad, Juergen Hoeller, etc.
◆ Registration at http://www.thespringexperience.com
Questions?

Ramlivnas Laddad
Email: ramlivnas.laddad@interface21.com
Web: http://interface21.com
Blog: http://ramnivas.com/blog