



JBoss AOP

or

**How to merge Features
without cluttering your
Code**



Andreas Schaefer, Senior Software Engineer

andreas.schaefer@madplanet.com

Founder of Madplanet.com Inc.

www.madplanet.com

Member of JSR-77 expert group

Co-author of JMX book

Former JBoss Core Contributor

JBoss AOP: What's Up



● JBoss AOP Framework:

- ✱ **Intro**
- ✱ Basic Concepts
- ✱ Simple Example
- ✱ Glenn's SwingSet Example
- ✱ How it works
- ✱ Configuration
- ✱ Q & A



- J2EE is a very simple AOP framework
- JBoss used Interceptors to wrap EJB services around the EJBs
- JBoss used adv. ClassLoaders to provide hot deployment for server-level services
- XML Descriptors defined EJB services and its Interceptors
- Now what do you get when combined?

JBoss AOP Framework

Standalone Framework Components



- AOP Framework Archives
 - ✱ jboss-aop-jdk50.jar
 - ✱ jboss-aspect-library-jdk50.jar
 - ✱ jboss-common.jar
 - ✱ pluggable-instrumentor.jar
 - ✱ javassist.jar
 - ✱ bsh-1.3.0.jar, concurrent.jar, qdox.jar, trove.jar
- AOP System ClassLoader
- AOP Compiler (aopc)
- AOP Deployment Descriptor: jboss-aop.xml
- Interceptors, Aspects and Mix-in classes

JBoss AOP: What's Up



- JBoss AOP Framework:
 - ✱ Intro
 - ✱ **Basic Concepts**
 - ✱ Simple Example
 - ✱ Glenn's SwingSet Example
 - ✱ How it works
 - ✱ Configuration
 - ✱ Q & A



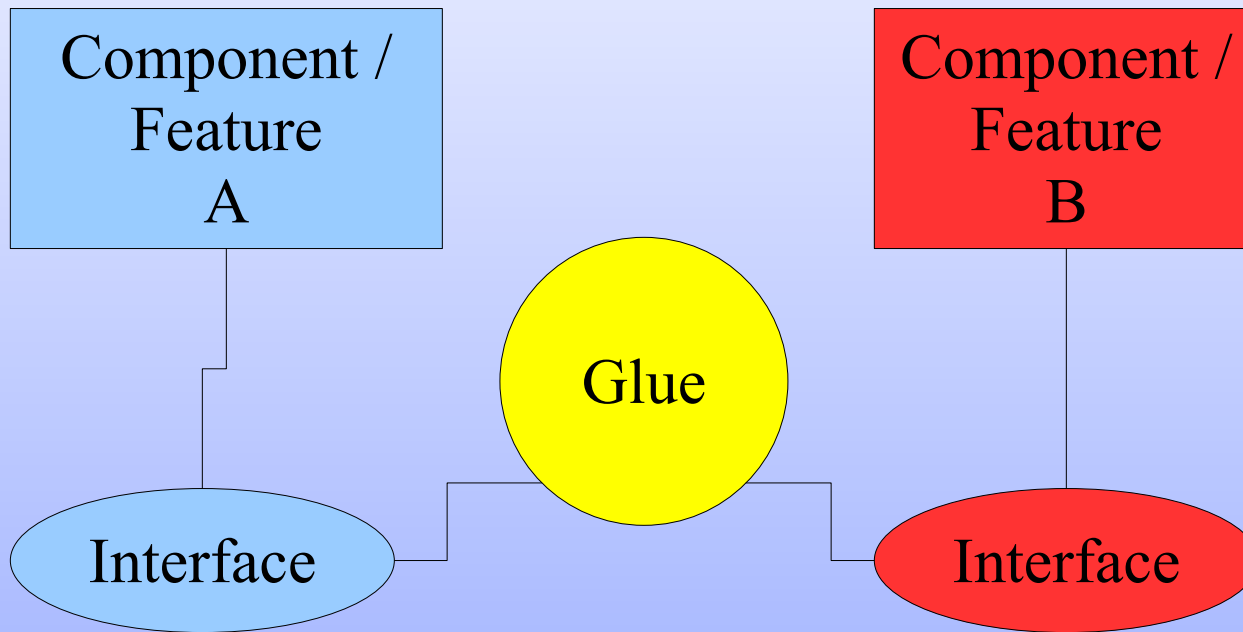
- Components are **hard** to Enhance
- Inheritance
 - ✱ Limited to one Base Class
 - ✱ Need a “Factory” or massive code change to use the Sub Class
 - ✱ Use of other features must be “designed in”
- Enhancements
 - ✱ Code Changes throughout the Project (like auditing)
 - ✱ Limited to available Source Code
 - ✱ Increases inter-dependencies between Components

Problems: Example



- Tracing
- Auditing
- Performance Monitoring
- Introduction of foreign Interfaces/Implementations
- Feature Merge

Problems: Feature Merge



- Security
- Auditing
- Performance Monitoring

Basic Concepts of AOP



- Provides Hooks into “Events” of Code: “Join Points”
 - ✱ Constructors
 - ✱ Methods
 - ✱ Field
 - ✱ Calls
- Allows the Injection of Code into these Hooks
- Enhancements are defined outside of the (Sub) Projects
- Pointcuts: selective subset of the available Join Points

Basic Concepts of AOP



- Instrumentation is done on Byte Code
- Binding of Pointcuts + Advices
 - ✱ Runtime Behavior
 - ✱ Marked as **Advised**
- Introduction: adds another Interface
- Mixins: provides the implementation of an Introduction

Basic Concepts of JBoss AOP



- Uses XML to define AOP Environment
- Provides two ways for Code Injection
 - ✱ Interceptor: a class with one Advice
 - ✱ Aspect: a class with zero or more Advices
- Advice: a method that is executed in a Pointcut
- Advices can be stacked
- Advices in an Aspect can be overloaded

JBoss AOP: What's Up



- JBoss AOP Framework:
 - ✱ Intro
 - ✱ Basic Concepts
 - ✱ **Simple Example**
 - ✱ Glenn's SwingSet Example
 - ✱ How it works
 - ✱ Configuration
 - ✱ Q & A



Very Simple Example: Tracing

- The “Hello World” example of AOP
- Tracing of
 - ✱ Constructor
 - ✱ Method
 - ✱ Field
- Log the entering and leaving of any Advised Element
- Trace the **Thread** of a Call



Very Simple Example: I

```
<aop>
  <interceptor name="Tracing"
    class="com.madplanet.aop.simple.TracingInterceptor"
    scope="PER_VM"/>
  <bind pointcut="execution( com.madplanet.aop.simple.Pojo->new( .. ) )">
    <interceptor-ref name="Tracing"/>
  </bind>
  <bind pointcut="execution( * com.madplanet.aop.simple.Pojo->*( .. ) )">
    <interceptor-ref name="Tracing"/>
  </bind>
  ...
</aop>
```

- This “jboss-aop.xml” deployment descriptor is located in the “./META-INF” directory

Very Simple Example: II



```
public Object invoke( Invocation plnvocation)
    throws Throwable {
    try {
        System.out.println(
            ">>> entering "
            + ( plnvocation instanceof MethodInvocation ?
                "method: " + ( (MethodInvocation) plnvocation ).getMethod() : "" )
            + ( plnvocation instanceof ConstructorInvocation ?
                "constructor: "
                + ( (ConstructorInvocation) plnvocation ).getConstructor() : "" )
            + ( plnvocation instanceof FieldInvocation ?
                "field: " + ( (FieldInvocation) plnvocation ).getField() : "" )
            );
        return plnvocation.invokeNext();
    } finally {
```

...

Very Simple Example: III



```
example1>java \  
-javaagent:.lib-50/jboss-aop-jdk50.jar \  
-classpath "$CLASSPATH" \  
com.madplanet.aop.simple.Main  
>>> entering constructor: public  
com.madplanet.aop.simple.Pojo(java.lang.String)  
>>> entering method: public void  
com.madplanet.aop.simpl.Pojo.setTest(java.lang.String)  
>>> entering field: private java.lang.String  
com.madplanet.aop.simple.Pojo.mTest  
<<< leaving field: private java.lang.String  
com.madplanet.aop.simple.Pojo.mTest  
...
```

Factory without One



- Factories are used in Java frequently
- Object-C allows a Constructor to return a Sub Class Instance
- Now with AOP we can do this in Java
- Disclaimer:
 - ✱ Does not work with only Interfaces
 - ✱ Base Class must **not** be **abstract**



Factory without One: I

```
<aop>
  <interceptor
    name="Factory"
    class="com.madplanet.prototype.aop.jboss.factory.FactoryInterceptor"
    scope="PER_VM"
  />
  <bind pointcut="execution(
    com.madplanet.prototype.aop.jboss.factory.Pojo->new( .. ) )">
    <interceptor-ref name="Factory"/>
  </bind>
</aop>
```

- Intercept the Constructor Call

Factory without One: II



```
public Object invoke( Invocation plnvocation)
    throws Throwable {
    if( plnvocation instanceof ConstructorInvocation ) {
        ConstructorInvocation llnvocation = (ConstructorInvocation) plnvocation;
        int IType = (Integer) llnvocation.getArguments()[ 0 ];
        switch( IType ) {
            case Pojo.SUB1_POJO:
                return new SubOnePojo( IType );
            case Pojo.SUB2_POJO:
                return new SubTwoPojo( IType );
        }
    }
    // Otherwise just return the regular Object created by the Constructor
    return plnvocation.invokeNext();
}
```

Factory without One: III



```
example1>java \  
-javaagent:..lib-50/jboss-aop-jdk50.jar \  
-classpath "$CLASSPATH" \  
com.madplanet.aop.factory.Main  
new Pojo( Pojo.SUB1_POJO ): \  
com.madplanet.prototype.aop.jboss.factory.SubOnePojo  
new Pojo( Pojo.SUB2_POJO ): \  
com.madplanet.prototype.aop.jboss.factory.SubTwoPojo  
...
```



Factory without One: Possibilities

- Reduce the number of Factories
- Allows to replace a given Instance with a Sub Class
- Easy to create and insert Mock objects

JBoss AOP: What's Up



- JBoss AOP Framework:
 - ✱ Intro
 - ✱ Basic Concepts
 - ✱ Simple Example
 - ✱ **Glenn's SwingSet Example**
 - ✱ How it works
 - ✱ Configuration
 - ✱ Q & A

Swing Set AOP DD: Menu Extension



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <interceptor name="TraceSetupInterceptor"
    class="com.madplanet.aop.swing.TraceSetupInterceptor"
    scope="PER_VM"/>

  <!-- Intercept the constructor call on the JMenuBar -->
  <bind pointcut="call( public javax.swing.JMenuBar->new( .. ) )">
    <interceptor-ref name="TraceSetupInterceptor"/>
  </bind>

  ...
```

Swing Set AOP DD: Trace Setup



```
...
<interceptor name="TracingInterceptor"
  class="com.madplanet.aop.swing.TracingInterceptor"
  scope="PER_VM"/>
<!-- First Intercept the call to a constructor, then the call to a method -->
<bind pointcut="call( public javax.swing.*->new( .. ) )">
  <interceptor-ref name="TracingInterceptor"/>
</bind>
<bind pointcut="call( public * javax.swing.*->*( .. ) )">
  <interceptor-ref name="TracingInterceptor"/>
</bind>
</aop>
```

Swing Set: Enhancing the SwingSet Menu



```
public Object invoke( Invocation pInvocation )
    throws Throwable {
    Object IReturn = pInvocation.invokeNext();
    // Create and add 'Debug' Menu to Swing Set Menu
    if( pInvocation instanceof ConstructorCalledByMethodInvocation ) {
        if( IReturn instanceof JMenuItem ) {
            addMenu( (JMenuBar) IReturn );
        }
    }
    return IReturn;
}
```

Swing Set: Insertion of Debug Menu



```
public void addMenu( JMenuBar pMenuBar) {  
    pMenuBar.add( sDebugMenu );  
    sDebugMenu.add( sConstructorFlag );  
    sDebugMenu.add( sMethodFlag );  
    sDebugMenu.add( sFieldFlag );  
    sDebugMenu.add( sDynamicInterceptorFlag );  
    sInfoWin.pack();  
    sInfoWin.setVisible( true );  
}
```

Swing Set: Interception for Tracing



```
public Object invoke( Invocation pInvocation)
    throws Throwable {
    Object IReturn;
    if( TraceSetupInterceptor.isEnabled( TraceSetupInterceptor.CONSTRUCTOR )
        && pInvocation instanceof ConstructorCalledByMethodInvocation ) {
        ConstructorCalledByMethodInvocation ci =
            (ConstructorCalledByMethodInvocation) pInvocation;
        TraceSetupInterceptor.trace(
            "Entering Constructor: " + ci.getCalledConstructor().getName()
        );
        IReturn = pInvocation.invokeNext();
        TraceSetupInterceptor.trace(
            "Leaving Constructor: " + ci.getCalledConstructor().getName()
        );
    }
}
```

...

Swing Set: Demo



Demo

The Surprise: Dynamic Interceptor



```
...
} else if( TraceSetupInterceptor.isEnabled( TraceSetupInterceptor.DYNAMIC_INTERCEPTOR ) ) {
    if( INewInterceptor == null ) {
        Class IDynamicInterceptorClass =
            Thread.currentThread().getContextClassLoader().loadClass(
                "com.madplanet.aop.swing.DynamicTracingInterceptor"
            );
        INewInterceptor = (Interceptor) IDynamicInterceptorClass.newInstance();
    }
    Interceptor[] INewInterceptors = new Interceptor[ 2 ];
    INewInterceptors[ 0 ] = INewInterceptor;
    // Add last element from the current interceptor list to the new one
    Interceptor[] ICurrentInterceptors = pInvocation.getInterceptors();
    INewInterceptors[ 1 ] = ICurrentInterceptors[ ICurrentInterceptors.length - 1 ];
    IReturn = pInvocation.invokeNext( INewInterceptors );
...

```

JBoss AOP: What's Up



- JBoss AOP Framework:
 - ✱ Intro
 - ✱ Basic Concepts
 - ✱ Simple Example
 - ✱ Glenn's SwingSet Example
 - ✱ **How it works**
 - ✱ Configuration
 - ✱ Q & A

Loadtime Instrumentation



- Class Loader loads bytecode
- Class Loader instruments the code using “javassist” to add additional code
- Class Loader let the VM resolve the class to create the Class instance
- Pro
 - Instrumentation can be changed by simple changing the AOP descriptor between runs
- Con
 - Performance hit during class loading
 - Can only instrument classes that are not already loaded

Compiletime Instrumentation



- AOP Compiler (AOPC) loads the class
- AOPC instruments code like the constructor
- AOPC writes the newly generated class(es) back as class file
- Pro
 - Can instrument any class it likes
 - No performance hit due instrumentation at runtime
- Con
 - Memory and Time consuming
 - Need to rerun AOPC everytime new instrumentation is needed



Interceptor Code (abstract):

```
public class SampleInterceptor
implements Interceptor {
public SampleInterceptor() {
}
public String getName() {
return "SampleInterceptor";
}
public Object invoke( Invocation plnvoation )
throws Throwable {
// Before Interception
Object IReturn = plnvoation.invokeNext();
// After Interception
return IReturn;
}
}
```

Interceptor



- Instanceof on the Interceptor tells what type of Interceptor it is
- Invocation can be upcasted to (according to its type):
 - ✱ ConstructorInvocation
 - ✱ MethodInvocation
 - ✱ FieldRead/WriteInvocation
 - ✱ “Callers Invocation”
- invokeNext() invokes the next element in the Stack
- InvokeNext(Interceptor[]) invokes the next element in a new Stack (redirection) for this very call (temp.)
- Advisor allows you to perm. change the Stack
- Last element in the Stack has to invoke the target

Aspects



- Aspect is a class contains zero or more Advices
- An Advice is a method taking an Invocation as single argument
- An Advice resembles an Interceptor
- Advices can be overloaded taking different kind of Invocations
- A Pointcut can use an Advice instead of an Interceptor containing the method name and the aspect (class) name



Aspect Code (abstract):

```
public class MyAspect {  
    public Object measure( ConstructorInvocation plnvocation )  
        throws Throwable {  
        // Before Interception  
        Object IReturn = plnvocation.invokeNext();  
        // After Interception  
        return IReturn;  
    }  
    public Object measure( MethodInvocation plnvocation )  
        throws Throwable {  
        // Before Interception  
        Object IReturn = plnvocation.invokeNext();  
        // After Interception  
        return IReturn;  
    }  
    ...  
}
```

AOP Deploy Descriptor: Aspects



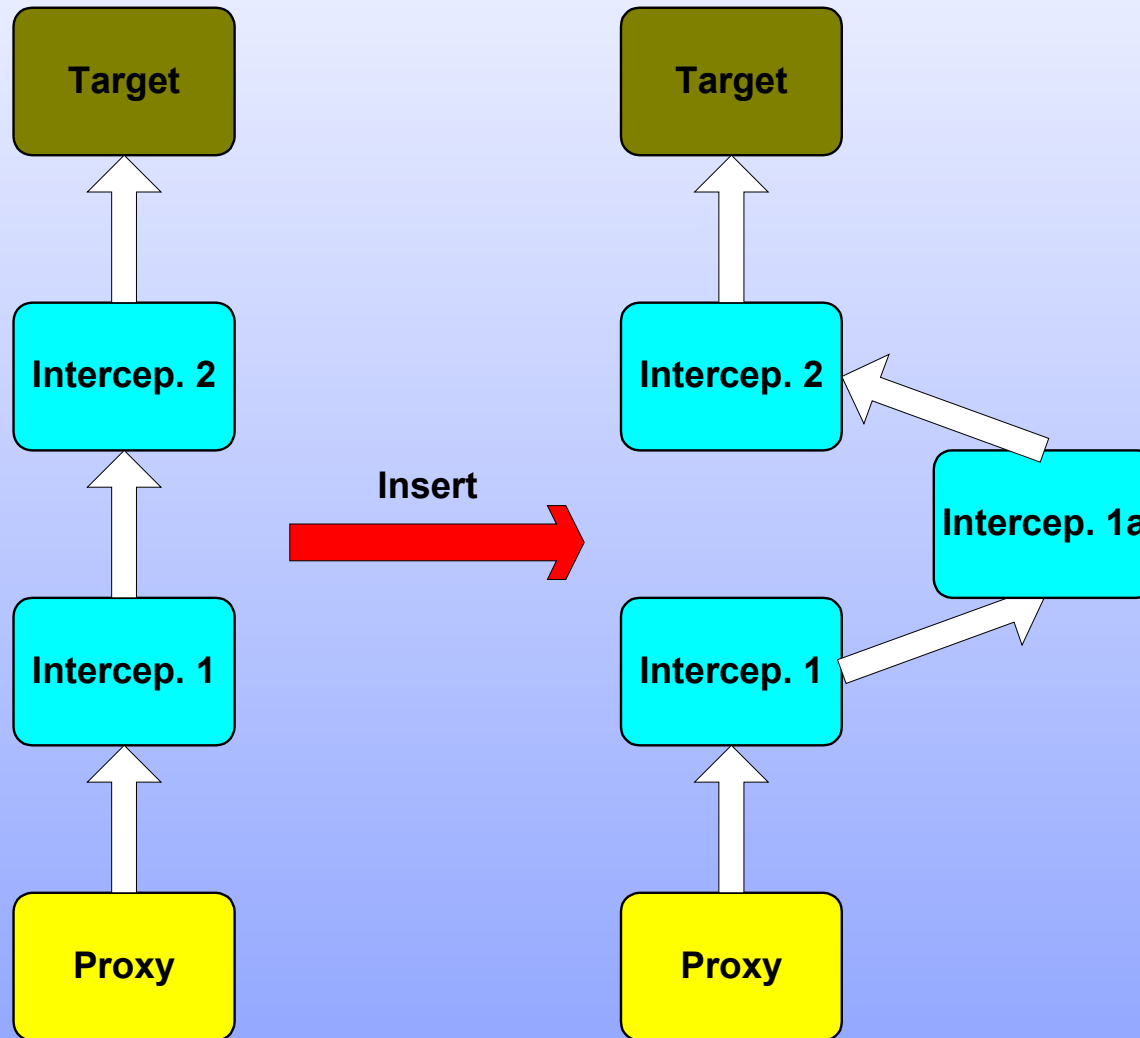
```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <aspect name="MyAspect"
    class="com.gateway.aop.metric.MyAspect"
    scope="PER_VM">
    <attribute name="MinimumDelayInMillis">250</attribute>
  </aspect>
  <bind pointcut="execution( public * com.madplanet.*->*(..) )">
    <advice name="measure" aspect="MyAspect"/>
  </bind>
}
```

JBoss AOP: What's Up

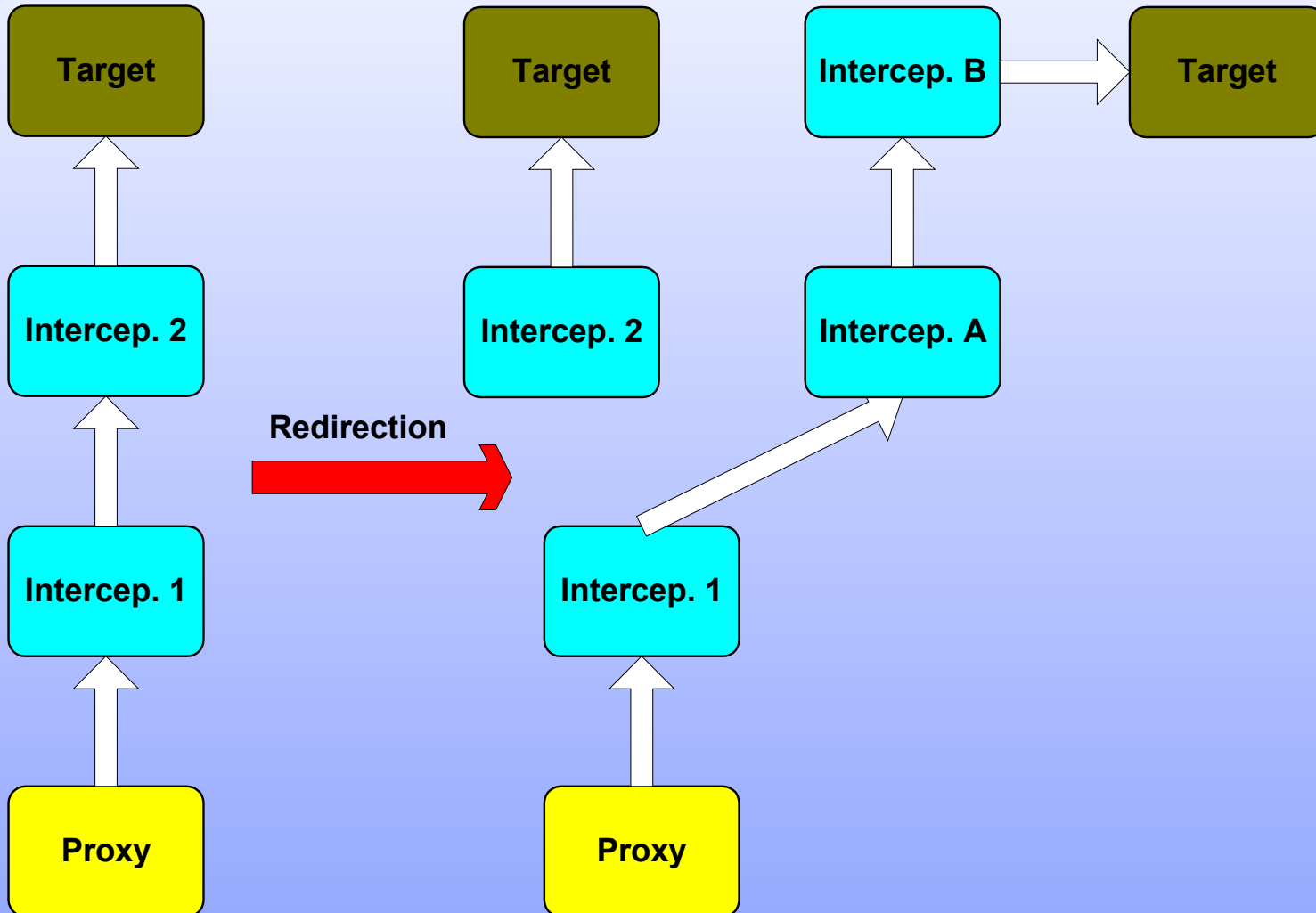


- JBoss AOP Framework:
 - ✱ Intro
 - ✱ Basic Concepts
 - ✱ Simple Example
 - ✱ Glenn's SwingSet Example
 - ✱ How it works
 - ✱ **Configuration**
 - ✱ Q & A

Interceptor Stack: Insertion



Interceptor Stack: Redirection



Interceptor Stack: Definition



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <stack name="myStack">
    <interceptor class="SimpleInterceptor1"/>
    <interceptor class="SimpleInterceptor2"/>
    <interceptor class="SimpleInterceptor3"/>
  </stack>

  <bind pointcut="execution( * com.madplanet.*->*(..) )">
    <stack-ref name="myStack"/>
  </bind>
</aop>
```

Invocation Class



- `org.jboss.aop.Invocation` provides:
 - Type of Invocation through Class
 - Array of Interceptors (Stack)
 - Complete Meta Data or single Meta Data Object
 - Target Object

General Pointcut



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

  <interceptor name="Metrics"
    class="com.madplanet.aop.Metrics"
    scope="PER_VM"/>

  <bind pointcut="all( com.madplanet.aop.Pojo )">
    <interceptor class="Metrics"/>
  </bind>

</aop>
```

- Intercepts everything: Constructors, Methods and Field Access

Constructor Pointcut



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
...
  <bind pointcut="execution( public *.oc25EJBLayer.*->new(..) )">
    <advice name="measure" aspect="Measure"/>
  </bind>
</aop>
```

- Intercepts all Constructors (but only Constructors)
- Invocation is of type ConstructorInvocation and provides:
 - Arguments (marshalled)
 - `java.lang.reflect.Constructor`

Method Pointcut



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
...
  <bind pointcut="execution( public * *.oc25EJBLayer.*->get*(..) )">
    <advice name="measure" aspect="Measure"/>
  </bind>
</aop>
```

- Intercepts only Method with prefix “get”
- Invocation is of type MethodInvocation and provides:
 - Arguments (marshalled)
 - `java.lang.reflect.Method`



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
...
  <bind pointcut="field( public * *.oc25EJBLayer.*->test* )">
    <advice name="measure" aspect="Measure"/>
  </bind>
</aop>
```

- Intercepts only Field Access with name prefix “test”
- Invocation is of type FieldRead/WriteInvocation:
 - `java.lang.reflect.Field`
 - value (for Write only)

Caller Pointcut



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <aspect name="Measure"
    class="com.gateway.aop.metric.MetricAspect"
    scope="PER_VM">
    <attribute name="MinimumDelayInMillis">250</attribute>
  </aspect>

  <bind pointcut="call( public *.oc25common.*->new(..) )">
    <advice name="measure" aspect="Measure"/>
  </bind>

  <bind pointcut="call( public * *.oc25common.*->*(..) )">
    <advice name="measure" aspect="Measure"/>
  </bind>
</aop>
```

Caller vs. Callee Pointcut



• Advantage of **Caller** Pointcut:

- No need to instrument target
- Only Caller matching the Pointcut will be *advised*
- A way to intercept code inside of the Caller's Method

• Advantage of **Callee** Pointcut:

- Any Caller's call is *advised*
- Callee must be *instrumentable*
- System classes need instrumented at compile time (AOPC)



• Classes

- `org.jboss.aop.*`, `org.jboss.util.*`, `javassist.*`
- All Generated and System classes (!)

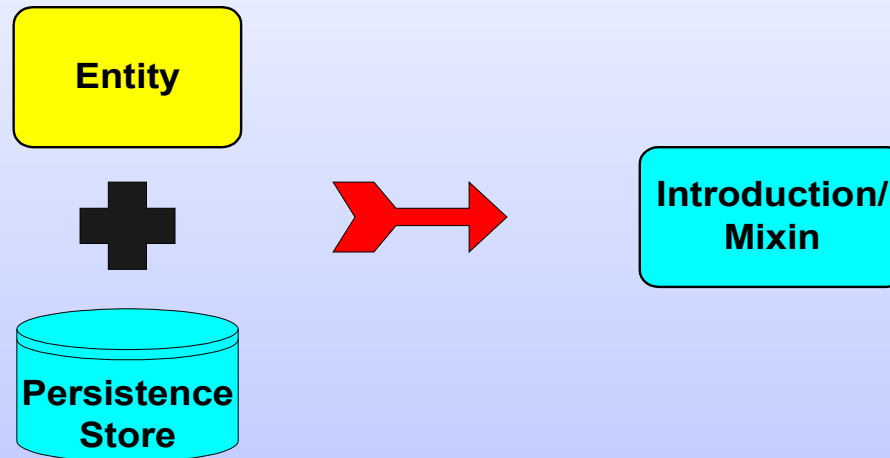
• Fields

- Final, with prefix ‘_’ and ‘\$’
- Fields added by the Framework

• Methods

- Abstract and Native Methods
- Method added by the framework
- Method with prefix ‘_’

Introduction Pointcut



- **Introduction** allows you to wrap Interfaces around a “**advised**” class so that it can be used by the user as if it had implemented this class
- **Mixin** allows you to define an implementation of the given Interfaces of an Introduction

Introduction Pointcut: Definition



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <introduction class="com.madplanet.aop.IO.*">
    <interfaces>
      javax.jdo.spi.PersistenceCapable
    </interfaces>
  </introduction>
</aop>
```

Introduction Pointcut: Usage



```
{  
  POJO IObject = new POJO();  
  PersistenceCapable IJDO = (PersistenceCapable) IObject;  
  ...  
}
```

- Introduction allows the developer to upcast to an Interface that is not provide by the object itself
- Mixin classes provides the implementation for such an Interface

Introduction Pointcut: Mixin



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <introduction class="com.madplanet.aop.IO.*">
    <mixin>
      <interfaces>
        javax.jdo.spi.PersistenceCapable
      </interfaces>
      <class>com.madplanet.aop.PersistentMixin</class>
      <construction>
        new com.madplanet.aop.PersistentMixin(this, "test",2)
      </construction>
    </mixin>
  </introduction>
</aop>
```



The Icing: Transactional Beans

- Goal: make any POJO transactional
- Interceptor for Field Writes
- Transaction Mixin that stores the original values and implements the **javax.transaction.XAResource** Interface
- Client upcast POJO to **XAResource** in order to enlist and delist from the transaction.

The Icing: TB Declaration



```
<introduction
  name="TransactionMixin"
  class="com.madplanet.prototype.aop.test.jboss.tx.Pojo">
  <mixin>
    <interfaces>javax.transaction.xa.XAResource</interfaces>
    <class>com.madplanet.prototype.aop.jboss.tx.TransactionMixin</class>
    <construction>
      new com.madplanet.prototype.aop.jboss.tx.TransactionMixin(this)
    </construction>
  </mixin>
</introduction>
<interceptor
  name="Transaction"
  class="com.madplanet.prototype.aop.jboss.tx.TransactionInterceptor"
  scope="PER_VM"
/>
<bind pointcut="set( public * com.madplanet.prototype.aop.test.jboss.tx.Pojo ->* )">
  <interceptor-ref name="Transaction"/>
</bind>
```

The Icing: Interceptor



```
public Object invoke( Invocation pInvocation)
    throws Throwable {
    if( pInvocation instanceof FieldWriteInvocation ) {
        FieldWriteInvocation IFieldInvocation = (FieldWriteInvocation) pInvocation;
        Object ITarget = pInvocation.getTargetObject();
        TransactionMixin IMixin = getMixin( ITarget );
        if( IMixin != null ) {
            IMixin.setField( IFieldInvocation.getField() );
        }
    }
    Object IReturn = pInvocation.invokeNext();
    return IReturn;
}
public TransactionMixin getMixin( Object pTarget ) {
    return mMixins.get( pTarget );
}
public void registerMixin( Object pTarget, TransactionMixin pMixin ) {
    mMixins.put( pTarget, pMixin );
}
}
```

The Icing: Transaction Mixin



```
public TransactionMixin( Object pTarget ) {
    mTarget = pTarget;
    AspectManager IManager = ( (Advised) pTarget )._getAdvisor().getManager();
    TransactionInterceptor IInterceptor = (TransactionInterceptor)
        IManager.getPerVMAAspect( "Transaction" );
    IInterceptor.registerMixin( pTarget, this );
}
...
public void rollback() {
    Iterator i = mOriginalValues.keySet().iterator();
    while( i.hasNext() ) {
        Field IField = (Field) i.next();
        try {
            IField.set( mTarget, mOriginalValues.get( IField ) );
        } catch( Exception e ) {
            e.printStackTrace();
        }
    }
    mOriginalValues = new HashMap();
}
```

The Icing: Usage



```
public void testSimpleBeanWithRollback()
    throws Throwable {

    Pojo IPojo = new Pojo( "Pojo-rollback-01" );
    int IExpectedValue = IPojo.getCounter();
    mTxManager.begin();
    Transaction ITx = mTxManager.getTransaction();
    ITx.enlistResource( (XAResource) IPojo );
    IPojo.doubleIt();
    IPojo.doubleIt();
    IPojo.doubleIt();
    IPojo.doubleIt();
    ITx.delistResource( (XAResource) IPojo, 0 );
    mTxManager.rollback();
    assertTrue(
        "After rollback the value must match the original value",
        IExpectedValue == IPojo.getCounter()
    );
}
```

The Icing: Output



```
trans.bean.exmample>java \  
-Djava.system.class.loader=\  
org.jboss.aop.standalone.SystemClassLoader POJO  
Pojo after creation: Pojo [ name: Pojo-commit-01, mCounter: 1  
Pojo after enlistment: Pojo [ name: Pojo-commit-01, mCounter: 1  
Pojo after first 'double it': Pojo [ name: Pojo-commit-01, mCounter: 2  
Pojo after second 'double it': Pojo [ name: Pojo-commit-01, mCounter: 4  
Pojo after third 'double it': Pojo [ name: Pojo-commit-01, mCounter: 8  
Pojo after commit: Pojo [ name: Pojo-commit-01, mCounter: 8  
Pojo after creation: Pojo [ name: Pojo-rollback-01, mCounter: 1  
Pojo after enlistment: Pojo [ name: Pojo-rollback-01, mCounter: 1  
Pojo after first 'double it': Pojo [ name: Pojo-rollback-01, mCounter: 2  
Pojo after second 'double it': Pojo [ name: Pojo-rollback-01, mCounter: 4  
Pojo after third 'double it': Pojo [ name: Pojo-rollback-01, mCounter: 8  
Pojo after fourth 'double it': Pojo [ name: Pojo-rollback-01, mCounter: 16  
Pojo after commit: Pojo [ name: Pojo-rollback-01, mCounter: 1
```

The Icing: Possibilities



- A UI could use a Transaction Manager (TM) to enlist and delist UI components
- Through the TM the program could reset the UI components through `Transaction.rollback()`
- A Wizard can be easily reset and so can be kept static
- The client-side Transaction can be used to also handle server-side transactions and so unite them

Current Limitations of JBoss AOP



- Framework needs full control of the Class Loader
- Cannot Advice/Instrument System classes
- Cannot Advice/Instrument Interceptors
- Slow Performance
- Dynamic changes only work on Advised Classes
- Default JBoss AOP deployment descriptor (META-INF/jboss-aop.xml) **must not** be set in startup script otherwise duplicate instrumentation happens
- JBoss does not validate AOP deployment descriptor

JBoss AOP: What's Up



- JBoss AOP Framework:
 - ✱ Intro
 - ✱ Basic Concepts
 - ✱ Simple Example
 - ✱ Glenn's SwingSet Example
 - ✱ How it works
 - ✱ Configuration
 - ✱ **Q & A**

Q & A



Andreas Schaefer

andreas.schaefer@madplanet.com

Web Log

www.madplanet.com/weblog/blog/schaefa/

Presentation can be found on

www.madplanet.com/mpg/profile/presentations.html