



Interception-Based AOP Or How does JBoss AOP Work?



Andreas Schaefer, Senior Software Engineer

andreas.schaefer@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**
- ✱ 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
- ✱ **Simple Example**
- ✱ Glenn's SwingSet Example
- ✱ How it works
- ✱ Configuration
- ✱ Q & A



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  
...
```

JBoss AOP: What's Up



- JBoss AOP Framework:
 - ✱ Intro
 - ✱ 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()
        );
    }
}
```

...



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 = plnvoation.getInterceptors();
    INewInterceptors[ 1 ] = ICurrentInterceptors[ ICurrentInterceptors.length - 1 ];
    IReturn = plnvoation.invokeNext( INewInterceptors );

```



- JBoss AOP Framework:
 - ✱ Intro
 - ✱ 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 pInvocation )
        throws Throwable {
        // Before Interception
        Object IReturn = pInvocation.invokeNext();
        // After Interception
        return IReturn;
    }
}
```



- 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



- 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



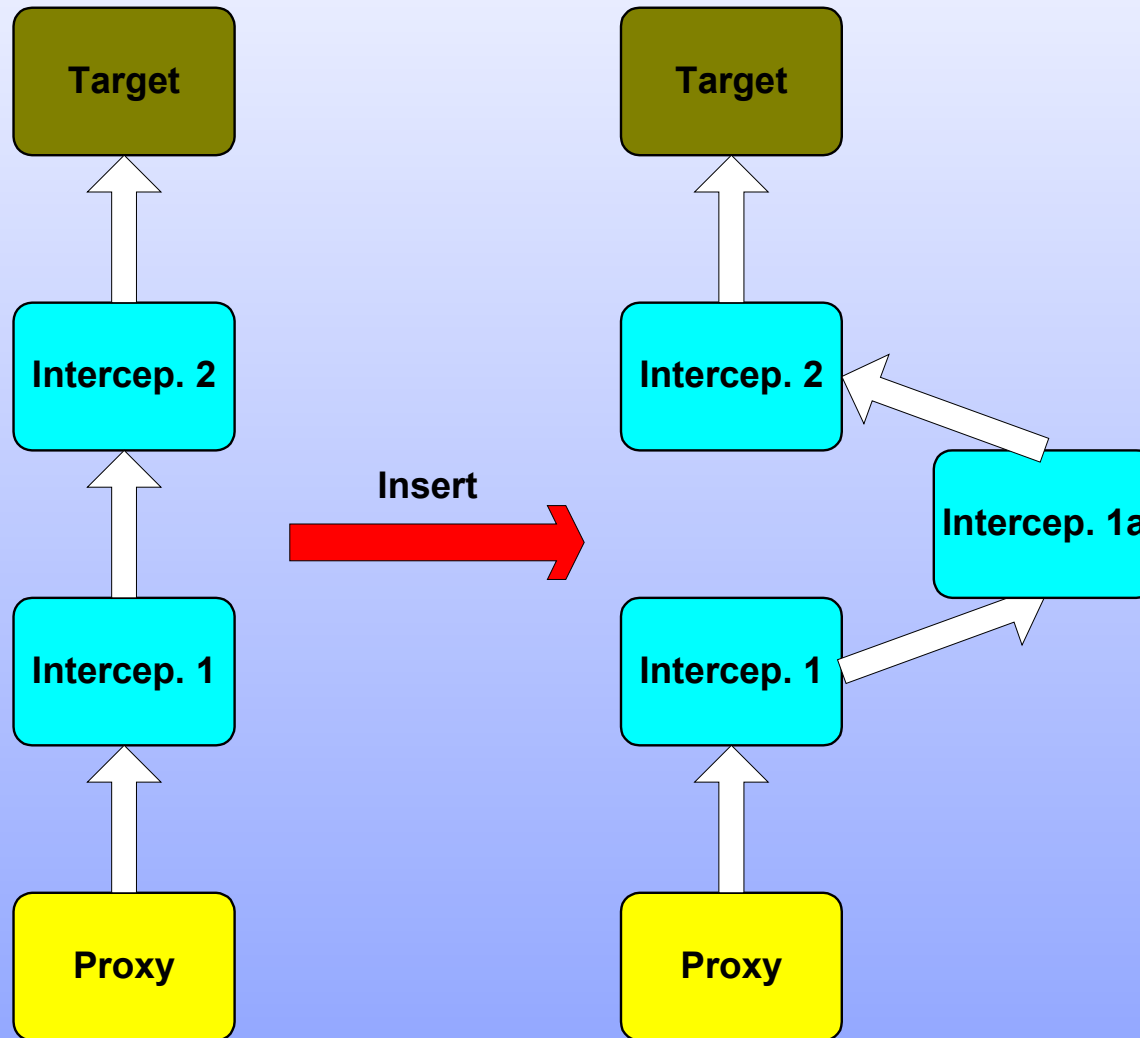
```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <aspect name="Measure"
    class="com.gateway.aop.metric.MyAspect"
    scope="PER_VM">
    <attribute name="MinimumDelayInMillis">250</attribute>
  </aspect>
  <bind pointcut="execution( public * com.madplanet.*->*(..) )">
    <advice name="measure"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

JBoss AOP: What's Up

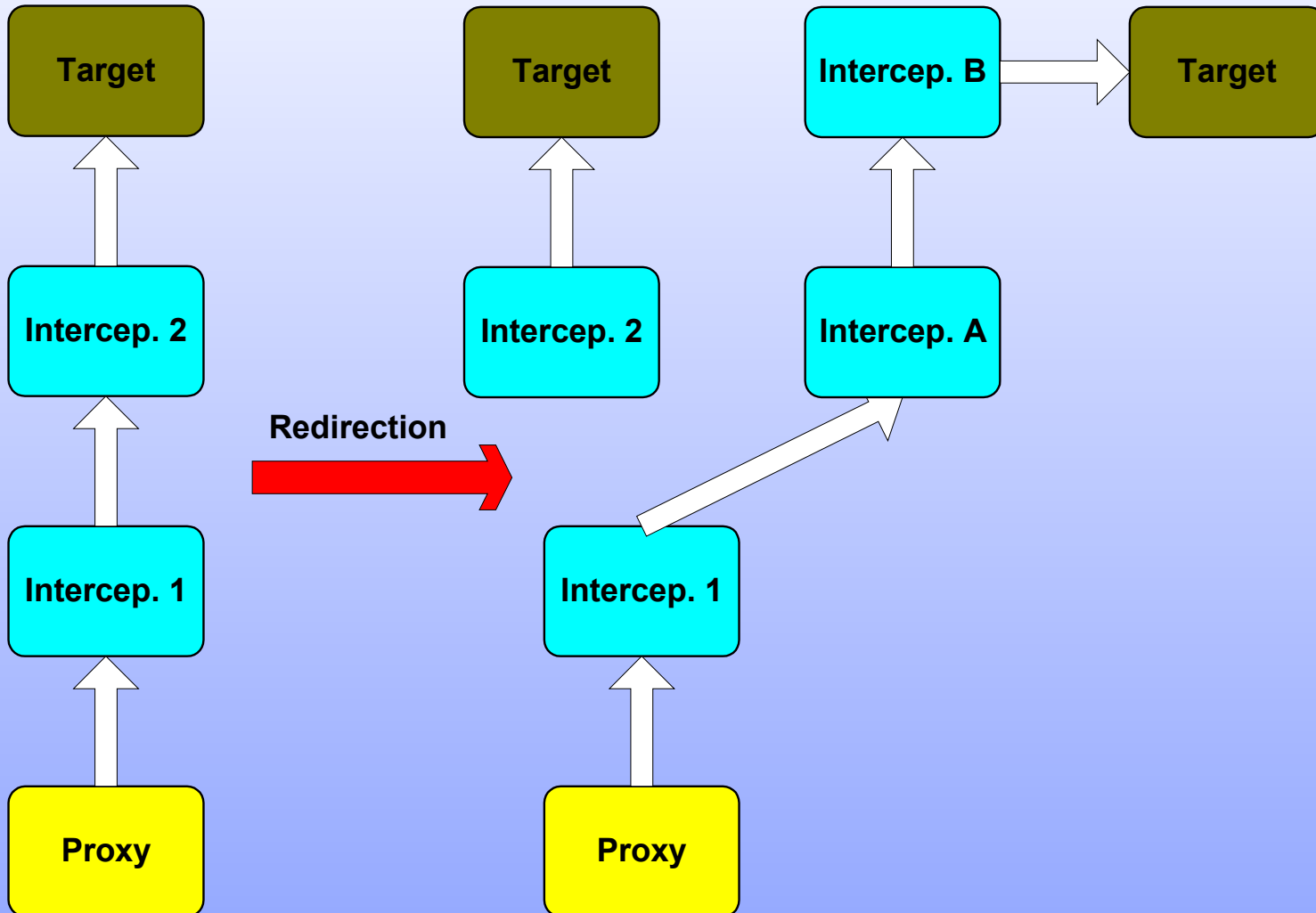


- JBoss AOP Framework:
 - ✱ Intro
 - ✱ Simple Example
 - ✱ Glenn's SwingSet Example
 - ✱ How it works
 - ✱ **Configuration**
 - ✱ Q & A

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.madpalnet.*->*(..) )">
    <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



```
<?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`



```
<?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`

Field Pointcut



```
<?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>
```



• Classes

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

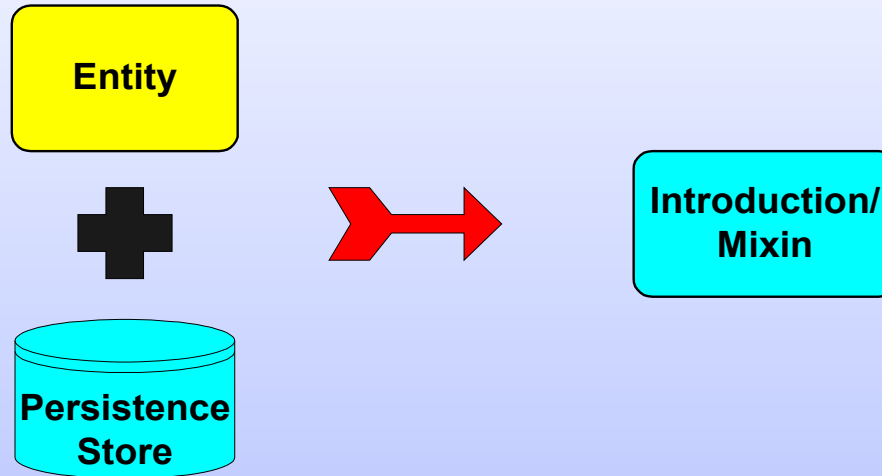
• Fields

- 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 “pointcutted” 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 so that it does not pollute the Interceptors

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 allows to provide an implementation for the interfaces of the Introduction.

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.Transaction** Interface
- Client upcast POJO to **Transaction** in order to commit or rollback.

The Icing: TB Declaration



```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <introduction class="com.madplanet.aop.tx.Pojo">
    <mixin>
      <interfaces>javax.transaction.Transaction</interfaces>
      <class>com.madplanet.aop.tx.mixins.TransactionMixin</class>
      <construction>
        new com.madplanet.aop.tx.mixins.TransactionMixin(this)
      </construction>
    </mixin>
  </introduction>
  <interceptor name="Transaction"
    class="com.madplanet.aop.tx.interceptor.TransactionInterceptor"
    scope="PER_VM"/>
  <bind pointcut="set( public * com.madplanet.aop.tx.Pojo->* )">
    <interceptor-ref name="Transaction"/>
  </bind>
  <bind pointcut="execution( com.madplanet.aop.tx.Pojo->new( .. ) )">
    <interceptor-ref name="Transaction"/>
  </bind>
</aop>
```

The Icing: Interceptor



```
public Object invoke( Invocation plnvocation)
    throws Throwable {
    if( plnvocation instanceof FieldWriteInvocation ) {
        FieldWriteInvocation IFieldInvocation = (FieldWriteInvocation) plnvocation;
        Object ITarget = plnvocation.getTargetObject();
        TransactionMixin IMixin = (TransactionMixin) mMixins.get( ITarget );
        if( IMixin != null ) {
            IMixin.setField( IFieldInvocation.getField() );
        }
    }
    Object IReturn = plnvocation.invokeNext();
    if( plnvocation instanceof ConstructorInvocation ) {
        ConstructorInvocation IInvocation = (ConstructorInvocation) plnvocation;
        Object IMixin = TransactionMixin.ThreadContext.get();
        Object ITarget = ( (TransactionMixin) IMixin ).getTarget();
        mMixins.put( ITarget, IMixin );
    }
    return IReturn;
}
```

The Icing: Transaction Mixin



```
public void rollback() {  
    // Reset the save values but we have to mark this as 'in rollback'  
    mInRollback = true;  
    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();  
        }  
    }  
    // Clear the original values because they are reinstated  
    mOriginalValues = new HashMap();  
    mInRollback = false;  
}
```

The Icing: Output



```
trans.bean.exmample>java \  
-Djava.system.class.loader=\  
org.jboss.aop.standalone.SystemClassLoader POJO  
TransactionInterceptor.<init>  
>>> Hello World! value = test1 #: 1  
Test after commit: test1 #: 1  
>>> Hello World! value = test22 #: 1  
>>> Hello World! value = test22 #: 2  
>>> Hello World! value = test22 #: 3  
Test after rollback (should be the same as above): test22 #: 1
```

The Icing: Possibilities



- Transaction Mixin could provide a XAResource implementation
- Pojo could implement XAResource through introduction
- A GUI could use a Transaction Manager (TM) to enlist and delist UI components
- Through the TM the program could reset the UI components though `Transaction.rollback()`
- A Wizard can be easily reset and so can be kept static

Current Limitations



- 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
- JBoss 4.0.3 with EJB3 RC3 does not allow for loadtime instrumentation
- 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
 - ✱ 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/jboss.tea