





Advanced AOP
JBoss, a division of Red Hat

Kabir Khan
JBoss AOP Lead
kabir.khan@jboss.com



Introduction

- AOP Primer
- New features in JBoss AOP 2.0
 - Better dynamic per class/instance AOP
 - AOP Scoped by classloader
 - Before/After/Throwing
- AOP/MC integration
- JBoss EJB 3 configured by AOP




AOP Primer

- Extract out cross-cutting concerns
 - code that looks like it should belong somewhere else
 - frequently repeated
 - Difficult to turn on and off
 - Bloat

```

public void transfer(Account from, Account to, float amount){
    tm.begin();
    try{
        try{
            from.debit(amount);
            to.credit(amount);
        }
        tm.commit();
    }catch(Exception e){
        error = true;
    }finally{
        if (error)tm.rollback();
        else tm.commit();
    }
}

```




Refactoring to AOP

- Keep core code as-is

```

@Transactional
public void transfer(Account from, Account to, float amount){
    from.debit(amount);
    to.credit();
}

```



Refactoring to AOP

- Put crosscutting code in advice

```

public class TxInterceptor implements Interceptor{
    public Object invoke(Invocation inv)throws Throwable{
        tm.begin();
        boolean error = true;
        try{
            return inv.invokeNext();
        }catch(Exception e){
            error = true;
        }finally{
            if (error)tm.rollback();
            else tm.commit();
        }
    }
}


```

- Use pointcuts to apply the advice

```


<jboss-aop.xml
<aop>
<bind pointcut="all(@Transactional)*">
<interceptor class="TxInterceptor"/>
</bind>
<aop>

```



New features in JBoss AOP 2.0

- Improved dynamic per class/per instance API
- AOP scoped by classloader
- Before/After/Throwing



Dynamic AOP

- JBoss AOP fully dynamic
 - AspectManager.addXXX()

```
<aop>
  <bind pointcut="all(@Transactional)">
    <interceptor class="TxInterceptor"/>
  </bind>
</aop>
```

- becomes:

```
AdviceBinding binding = new AdviceBinding("all(@Transactional)", null);
AspectFactory factory = new GenericAspectFactory("TxInterceptor", null);
AspectDefinition def =
  new AspectDefinition("TxInterceptor", Scope.PER_VM, factory);
AspectManager.instance().addAspectDefinition(def);

ScopedInterceptorFactory ifac = new ScopedInterceptorFactory(def);
AspectManager.instance().addInterceptorFactory(ifac.getName(), ifac);
binding.addInterceptorFactory(ifac);
AspectManager.instance().addBinding(binding);
```



Prepare

- As long as a joinpoint has been woven can add more aspects to it at runtime
 - bind
 - prepare

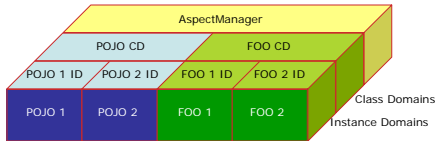
```
<aop>
  <prepare expr="all(@Transactional)"/>
</aop>
```

```
AdviceBinding binding = new AdviceBinding("all(@Transactional)", null);
...
AspectManager.instance().addBinding(binding);
```



Per Class/Instance Bindings

- Domain
 - sub-class of AspectManager
 - domain per class
 - domain per instance
- Hierarchy
 - Can see aspects in the parent, but not siblings or children



Per Class/Instance Bindings

```
@Annotated
public class POJO{
  int field;
  void method();
}

@Annotated
public class Foo{
  int field;
  void method();
}
```

```
<aop>
  <bind pointcut="all(@Annotated)">
    <interceptor class="SomeInterceptor"/>
  </bind>
</aop>
```

```
POJO poj1 = new POJO();
POJO poj2 = new POJO();
Foo foo = new Foo();
AdviceBinding newBinding
  = new AdviceBinding("field(int @Annotated->field)", null);
//Add FieldInterceptor to newBinding
....
```

- Where we add binding has impact on what it is applied to



Per Class Bindings

```
<aop>
  <bind pointcut="all(@Annotated)">
    <interceptor class="SomeInterceptor"/>
  </bind>
</aop>
```

- Add aspects for just the POJO class domain

```
POJO poj1 = new POJO();
POJO poj2 = new POJO();
Foo foo = new Foo();
AdviceBinding newBinding
  = new AdviceBinding("field(int @Annotated->value)", null);
//Add FieldInterceptor to newBinding
...
Advisor poj1CA = ((Advised)poj1)._getAdvisor();
AspectManager pojMgr = poj1CA.getManager();
pojMgr.addBinding(newBinding);

poj1.value = 10; //SomeInterceptor + FieldInterceptor
poj2.value = 15; //SomeInterceptor + FieldInterceptor
foo.value = 20; //SomeInterceptor
```



Per Instance Bindings

```
<aop>
  <bind pointcut="all(@Annotated)">
    <interceptor class="SomeInterceptor"/>
  </bind>
</aop>
```

- Add aspects for just the POJO 1 instance domain

```
POJO poj1 = new POJO();
POJO poj2 = new POJO();
Foo foo = new Foo();
AdviceBinding newBinding
  = new AdviceBinding("field(int @Annotated->value)", null);
//Add FieldInterceptor to newBinding
...
Advisor poj1IA =
  (Advisor)((InstanceAdvised)poj1)._getInstanceAdvisor();
AspectManager pojMgr = poj1IA.getManager();
pojMgr.addBinding(newBinding);

poj1.value = 10; //SomeInterceptor + FieldInterceptor
poj2.value = 15; //SomeInterceptor
foo.value = 20; //SomeInterceptor
```

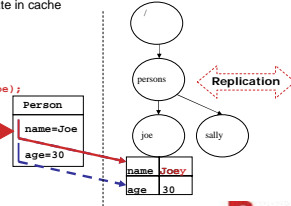


Per Instance Bindings - Example

- Example of use is POJO Cache
 - attach an object to cache
 - dynamically insert field interceptors
 - state is kept in cache
 - object becomes "window" to state in cache

```
Person joe = new Person();
joe.setName(joe);
joe.setAge(30);
cache.attach("/persons/joe", joe);
```

```
joe.setName("Joey")
```



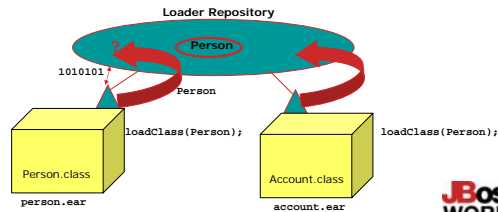
JBoss
WORLD
BERLIN 2006
PRESENTED BY RED HAT

Class Loading in JBoss AS & Scoped AOP

JBoss
WORLD
BERLIN 2006
PRESENTED BY RED HAT

Class Loading in JBoss AS - Default

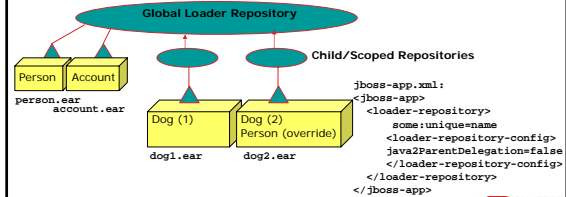
- One class loader per deployment
 - backed by a class loading domain
 - All loaders in domain share repository
 - Full class visibility among all applications



JBoss
WORLD
BERLIN 2006
PRESENTED BY RED HAT

Scoped Class Loading in JBoss AS

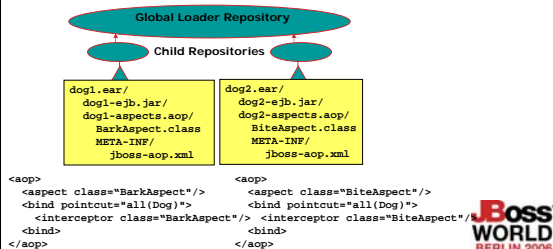
- Sometimes we want our classes to be "invisible" to others
 - Two versions of same application
 - Overriding server classes
 - Can still see classes in global domain, but parent/siblings cannot see ours



JBoss
WORLD
BERLIN 2006
PRESENTED BY RED HAT

Scoped AOP in JBoss AS

- Aspects deployed in the global domain are visible to all applications
- Aspects deployed in a child domain are only visible within that domain



JBoss
WORLD
BERLIN 2006
PRESENTED BY RED HAT

Around vs Before/After/Throwing

JBoss
WORLD
BERLIN 2006
PRESENTED BY RED HAT

Around vs Before/After/Throwing

- Around is driven by invocation
 - method
 - metadata
 - target object
 - args
 - etc.

```
<aspect class="DemoAspect"/>
<bind pointcut="execute(double POJO->calculate(int)">
  <advice name="DemoAspect" name="invoke"/>
</bind>
```

```
public class DemoAspect{
  public Object invoke(MethodInvocation inv) throws Throwable{
    Object[] args = inv.getArguments();
    Annotation ann = inv.resolveAnnotation(Annotation.class);
    String metadata = inv.getMetadata("security", "principal");
    Method m = inv.getMethod();
    Object target = inv.getTargetObject();

    return inv.invokeNext();
  }
}
```



Around vs Before/After/Throwing

- Can get full b/a/t behaviour with an around advice
 - "Problem": Invocation allocated for every call

```
public class DemoAspect{
  public Object invoke(MethodInvocation inv) throws Throwable{
    Object ret = null;
    try{
      //Before

      ret = inv.invokeNext();
      //After

      return ret;
    }catch(Exception e){
      //Throwing
    }finally{
      //Finally
    }
  }
}
```



Before/After/Throwing

- Not invocation-driven
 - Microbenchmarks show 15-20x faster

```
<aspect class="BATAspect"/>
<bind pointcut="execute(double POJO->calculate(int)">
  <before name="BATAspect" name="before"/>
  <after name="BATAspect" name="after"/>
  <throwing name="BATAspect" name="throwing"/>
</bind>
<bind pointcut="field(* POJO->i)">
  <before name="BATAspect" name="before"/>
</bind>
```

```
public class BATAspect{
  void before(
    @JoinPoint MethodInfo info,
    @Target POJO tgt,
    @Arg int i){

    log.info("Calling " + info.getMethod().getName()
      + "(" + i + " on " + tgt);
  }

  void before(@Target POJO tgt){
  }
}
```



Integration with Microcontainer

The JBoss Microcontainer

- Dependency injection framework
- Replaces the legacy JMX Microkernel in JBoss 5
 - JMX standard
 - Dependency/lifecycle management beyond JMX
- Services as simple POJOs
- JMX available as an aspect
- Allows for running JBoss services standalone/in other containers
 - "JBoss everywhere"



Deploying MC Bean

- Deploy in JBoss using -beans.xml

```
some-beans.xml
<deployment>
  <bean name="Simple" class="SimpleBean">
    <property name="value">Hello</property>
  </bean>
  <bean name="Dependent" class="DependentBean">
    <property name="dependency"><inject bean="Simple"/></property>
  </bean>
</deployment>
```

```
public class SimpleBean{
  String value;
  public void setValue(String val){
    value = val;
  }
}

public class DependentBean{
  SimpleBean dependency;
  public void setDependency(SimpleBean dep){
    dependency = dep;
  }
}
```



AOP + Microcontainer Integration

- Managed aspects
- Per bean annotations
- Propagated dependencies from aspects to beans



Managed Aspects

```

some-beans.xml
<deployment>
  <bean name="TxManager" class="TxManager"/>
  <aop:aspect xmlns:aop="urn:jboss:aop-beans:1.0"
    name="Transactional"
    code="TxInterceptor"
    pointcut="all(@Transactional)*">
    <property name="txManager"><inject bean="TxManager"/></property>
  </aop:aspect>
  <bean name="SimpleNotTx" code="SimpleBean"/>
  <bean name="SimpleTx" code="SimpleBean">
    <annotation>@Transactional</annotation>
  </bean>
</deployment>

```

```

public class TxInterceptor implements Interceptor{
  TxManager tm;
  public void setTxManager(TxManager tm){
    this.tm = tm;
  }

  public Object invoke(Invocation inv) throws Throwable{
    ...
  }
}

```



JMX in JBoss 5

- MC is core of JBoss 5
 - MBean server installed as a bean
- Old -service.xml deployments still work
 - Registers beans in MC and in JMX
- -beans.xml installs beans in MC
 - No JMX...
 - Can register in MBeanServer by annotating with @JMX

```

<deployment>
  <bean name="MockMBean" class="SomeBean">
    <annotation>@JMX(name="service:MockMBean",
      exposedInterface=SomeBeanInterface.class)
    </annotation>
  </bean>
</deployment>

```

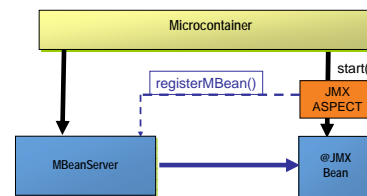
```

public interface SomeBeanInterface{
  int getAttribute();
  void setAttribute(int i);
  String operation(String s);
}

```



JMX as an aspect

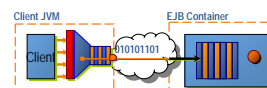


JBoss EJB 3 Container



JBoss EJB 3

- Evolution of the EJB 2.0 container
 - Client side dynamic proxy with interceptors
 - Container behaviour added by interceptors
- Uses AOP to configure the container and client proxies
 - SLSB, SFSB, MDB
- Default mappings per bean-type
 - Overridable/customizable
 - deploy/ejb3-interceptors-aop.xml
- Internally the container uses the AOP-style invocations and interceptors
 - created in client dynamic proxy
 - passed through the server's container



Client side

- Stack per bean type defines interceptors used in client proxy

```
ejb3-interceptors-aop.xml
<stack name="StatelessSessionClientInterceptors">
  <interceptor-ref name="IsLocalInterceptor"/>
  <interceptor-ref name="SecurityClientInterceptor"/>
  <interceptor-ref name="ClientTxPropagationInterceptor"/>
  <interceptor-ref name="InvokeRemoteInterceptor"/>
</stack>
<stack name="ClusteredStatelessSessionClientInterceptors">
  <interceptor-ref name="IsLocalInterceptor"/>
  <interceptor-ref name="SecurityClientInterceptor"/>
  <interceptor-ref name="ClientTxPropagationInterceptor"/>
  <interceptor-ref name="ClusterChooserInterceptor"/>
  <interceptor-ref name="InvokeRemoteInterceptor"/>
</stack>
<stack name="StatefulSessionClientInterceptors">.../stack>
<stack name="ClusteredStatefulSessionClientInterceptors">.../stack>
...
```

```
@Remote({TestRemote.class})
@Stateless
@Clustered
public class TestBean implements TestRemote{...}
```



Server side

```
ejb3-interceptors-aop.xml
<domain name="Stateless Bean">
  <bind pointcut="execution(public * @SecurityDomain->*(...))">
    <interceptor-ref name="AuthorizationAndAuthentication"/>
  </bind>
  <bind pointcut="execution(public * @RunAs->*(...))">
    <interceptor-ref name="RunAsSecurityInterceptorFactory"/>
  </bind>
  <bind pointcut="execution(public * @Clustered->*(...))">
    <interceptor-ref name="ReplicantsManagerInterceptorFactory"/>
  </bind>
  <bind pointcut="execution(public * *->*(...))">
    <interceptor-ref name="StatelessInstanceInterceptor"/>
    <interceptor-ref name="TxInterceptorFactory"/>
    <interceptor-ref name="EJB3InterceptorsFactory"/>
  </bind>
</domain>
<domain name="Stateful Bean">.../domain>
...
```

```
@Remote({TestRemote.class})
@Stateless
@Clustered
public class TestBean implements TestRemote{...}
```



Customising the interceptor stack

```
public class MyClientInterceptor implements Interceptor{
  public String getName{return "MyClientInterceptor";}

  public Object invoke(Invocation inv) throws Throwable{
    invocation.getMetaData().addMetaData(
      "CUSTOM",
      "ip",
      getLocalIpAddress());
    return inv.invokeNext();
  }
}
```

```
public class MyServerInterceptor implements Interceptor{
  public String getName{return "MyServerInterceptor";}

  public Object invoke(Invocation inv) throws Throwable{
    System.out.println("Call coming from" +
      invocation.getMetaData( "CUSTOM", "ip"));
    return inv.invokeNext();
  }
}
```



Customising the interceptor stack

```
my-custom-aop.xml
<aop>
  <stack name="MyClientStack">
    <interceptor-ref name="IsLocalInterceptor"/>
    <interceptor-ref name="SecurityClientInterceptor"/>
    <interceptor-ref name="ClientTxPropagationInterceptor"/>
    <interceptor-ref name="MyClientInterceptor"/>
    <interceptor-ref name="InvokeRemoteInterceptor"/>
  </stack>
  <domain name="MyDomain" extends="Stateless Bean" inheritsBindings="true">
    <bind pointcut="execution(* *->*(...))">
      <interceptor class="MyServerInterceptor"/>
    </bind>
  </domain>
</aop>
```

```
myejb-jar
MyClientInterceptor.class
MyServerInterceptor.class
TestBean.class
TestRemote.class
my-custom-aop.xml
META-INF/
ejb-jar.xml

@Remote({TestRemote.class})
@Stateless
@RemoteBinding(interceptorStack="MyClientStack")
@AspectDomain("MyDomain")
public class TestBean implements TestRemote{...}
```



Round-up

- JBoss AOP is core to JBoss
 - Microcontainer
 - POJO Cache, JBoss EJB 3, JBoss Messaging
- Fully dynamic
 - Can deploy/undeploy aspects at runtime
 - global/per class/per instance class
- Understands the JBoss class loading mechanism
 - AOP scoped by deployment
- Before/After/Throwing will offer performance enhancements
- Also usable standalone or for your applications

