Extending Enterprise JavaBeans™ 3.0 Specification With Interceptors

Bill Burke

Chief Architect
JBoss Inc.
www.jboss.org

TS-1365

Agenda

Interceptor Overview

Why Interceptors?

Future of Interceptors

Interceptor Overview

- Enterprise JavaBeans™ (EJB™) 3.0 specification formalizes interceptors
  - Already available in proprietary products
- Intercept incoming business method in container
  - Server side only!
- Intercept EJB specification lifecycle events

Purpose of EJB 3.0 Specification Interceptors

- Aspectizing your applications
- Pluggable annotations
- Ease of Extension
  - Not just ease of use
  - Framework for frameworks

Method Profiling

```
@Stateless
public class BankAccountBean implements BankAccount {

   @PersistenceContext EntityManager entityManager;

   public void withdraw(int acct, double amount) {
      long start = System.currentTimeMillis();
      try {
         Account account = entityManager.find(…);
         validateWithdrawal(account, amount);
         account.withdraw(amount);
         entityManager.flush();
      } finally {
        long time = start = System.currentTimeMillis();
        System.out.println(“withdraw took ” + time);
      }
   }
}
```

What's Wrong With Example?

- Bloated code
  - Profiling has nothing to do with business logic
- Difficult to enable/disable profiling
- Impossible to extend profiling behavior transparently

Interceptors to the Rescue

- Provides structure where none exists in OOP
- Can encapsulate profiling logic in one class
  - Easier to extend
  - Easier to debug
- Facilities to transparently apply profiling logic
  - Easy to apply profiling logic

Interceptor Implementation

```
public class ProfilingInterceptor {

   @AroundInvoke
   public Object profile(InvocationContext invocation)
      throws Exception
   {
      long start = System.currentTimeMillis();
      try {

         return invocation.proceed();

      } finally {
        long time = start - System.currentTimeMillis();
        Method method = invocation.getMethod();
        System.out.println(method.toString() +
                                  " took " + time + " (ms)");
      }
   }
}
```

@AroundInvoke Method

- Intercepts method being invoked
- Called in chain of other applied interceptors
- In same Java call stack as bean method
  - Wraps around
  - Thrown bean exceptions may be caught
- InvocationContext abstraction class for invoked method

javax.interceptor.InvocationContext

```
public interface InvocationContext {
   public Object getTarget();
   public Method getMethod();
   public Object getParameters();
   public void setParameters(Object[] args);
   public Map<String, Object> getContextData();
   public Object proceed() throws Exception;
}
```

Must always be called at the end of the interceptor implementation in order for the invocation to proceed.

InvocationContext Usage

```
public class ProfilingInterceptor {

   @AroundInvoke
   public Object profile(InvocationContext invocation)
      throws Exception
   {
      long start = System.currentTimeMillis();
      try {

         return invocation.proceed();

      } finally {
        long time = start - System.currentTimeMillis();
        Method method = invocation.getMethod();
        System.out.println(method.toString() +
                           " took " + time + " (ms)");
      }
   }
}
```

Interceptor Details

- Run in same tx and security context as method
- Interceptor has same lifecycle as EJB technology
  - Interceptor instance created per EJB technology instance
  - Pooled along with bean as well
  - Destroyed when its EJB technology instance is destroyed
  - Side effect? They can hold state
- Support XML and annotation driven injection
- Belong to the same ENC as EJB technology

Evolving Profiler

```
public class ProfilingInterceptor {
   @Resource SessionContext ctx;
   @PersistenceContext EntityManager manager;

   @AroundInvoke
   public Object profile(InvocationContext invocation)
      throws Exception {
      long start = System.currentTimeMillis();
      try {

         return invocation.proceed();

      } finally {
        long time = start - System.currentTimeMillis();
        Profile prof = new Profile(
            time, invocation.getMethod(),
            ctx.getPrincipal()
        );
        manager.persist(prof);
      }
}
```

Evolving Profiler

```
public class ProfilingInterceptor {

   @EJB Profiler profiler;

   @AroundInvoke
   public Object profile(InvocationContext invocation)
      throws Exception
   {
      long start = System.currentTimeMillis();
      try {

         return invocation.proceed();

      } finally {
        long time = start = System.currentTimeMillis();
        profiler.log(invocation, time);
      }
   }
}
```

Optional Interceptor Declaration

```
<ejb-jar>
   <interceptors>
      <interceptor>
         <interceptor-class>
            com.titan.ProfilingInterceptor
         </interceptor-class>
         <ejb-local-ref>
            …
         </ejb-local-ref>
      </interceptor>
   </interceptors>
</ejb-jar>
```

Applying Interceptors

- Through annotations
  - @javax.interceptors.Interceptors
- Through explicit XML
  - ejb.jar.xml
- Through default XML
  - Default interceptors

Applying Interceptors

Accepts an array of classes

```
@Stateless
@Interceptors(ProfilingInterceptor.class)
public class BankAccountBean implements BankAccount {

   @PersistenceContext EntityManager entityManager;

   public void withdraw(int acct, double amount) {
      Account account = entityManager.find(…);
      validateWithdrawal(account, amount);
      account.withdraw(amount);
      entityManager.flush();
   }
}
```

@Interceptors on Class

- One or more can be applied
- Every method is intercepted
- Executed in order they are declared

XML Binding

```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>BankAccountBean</ejb-name>
         <interceptor-class>
            com.titan.ProfilingInterceptor
         </interceptor-class>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

One or more <interceptor-class> entries allowed

Per-method Interceptors

- Interceptor executes for one given method
- Executed after any class level interceptors

Per-method Interceptors

```
@Stateless
public class BankAccountBean implements BankAccount {

   @PersistenceContext EntityManager entityManager;

   @Interceptors(ProfilingInterceptor.class)
   public void withdraw(int acct, double amount) {
       …
   }

   public void deposit(int acct, double amount) {
       …
   }
}
```

Per-method XML

```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>BankAccountBean</ejb-name>
         <interceptor-class>
            com.titan.ProfilingInterceptor
         </interceptor-class>
         <method>
            <method-name>withdraw</method-name>
         </method>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Default Interceptors

- You can apply a set of interceptors to every EJB specification
  - Per deployment only
  - Simple ejb-jar.xml description
- ‘*’ wildcard in <ejb-name>

Default Interceptors

```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>*</ejb-name>
         <interceptor-class>
            com.titan.ProfilingInterceptor
         </interceptor-class>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Exception Handling

- Allowed to abort invocation
- Allowed to catch and retry an invocation
- Allowed to throw a different exception

Aborting Invocation: Validation

```
public class WithdrawValidation {

   @Resource(name=“maxWithdraw”)
   double maxWithdraw = 500.0;

   @AroundInvoke
   public Object validate(InvocationContext ctx)
      throws Exception
   {
      double amount = (Double)ctx.getParameters()[0];
      if (amount > maxWithdraw) {
         throw new RuntimeException(“Max Withdraw is “
                   + maxWithdraw);
      }
      return ctx.proceed();
   }
}
```

Aborting Invocation: Validation

```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>BankAccountBean</ejb-name>
         <interceptor-class>
            com.titan.WithdrawInvalidation
         </interceptor-class>
         <method>
            <method-name>withdraw</method-name>
         </method>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Custom Security

```
public class RuleBasedAuthorization {
   @EJB SecurityRulesEngine ejb;

   @AroundInvoke
   public Object authorize(InvocationContext ctx)
      throws Exception
   {
      if (!ejb.authorized(ctx.getMethod(),
                          ctx.getParameters()) {
         throw new EJBAccessException(
             “Failed to Authorized”
         );
      }
      return ctx.proceed();
   }
}
```

Exception Wrapping

- Map SQLException to an exception hierarchy
- Take vendor errno and convert it to
  - DeadlockException
  - InvalidSqlException
  - Etc…
- Allows user to catch concrete exception
  - Vendor-specific error handling abstracted

SQLException Mapper

```
public class SQLExceptionMapper {

   @AroundInvoke
   public Object wrap(InvocationContext ctx)
      throws Exception
   {
      try {
         return ctx.proceed();
      } catch (SQLException ex) {
         switch (ex.getErrorCode()) {
           case 3344:
              throw new DeadlockException(ex);
           case 4223:
              throw new InvalidSqlException(ex);
           …
         }
      }
   }
}
```

SQLException Wrapper

```
@Stateless
@Interceptors(com.titan.SQLExceptionMapper)
public class MyDAOBean implements MyDAO {

   List queryStuff() throws SQLException {
     …
   }
}
```

Intercepting Lifecycle Events

- Re-use callback annotations
- Same signature as @AroundInvoke methods
- In same Java™ based call stack as any bean callbacks
  - If the bean has the callback

Custom Injection Annotation

- Java Platform, Enterprise Edition has no annotations to inject directly from Java Naming and Directory Interface™ API
- Let's create a @JndiInjected annotation
  - Use callback interception to implement

Custom Injection Annotation

```
@Stateless
public class MyBean {

   @JndiInjected("jboss/employees/bill/address")
   Address address;

…

}
```

Step 1: Implement Annotation

```
package com.titan;

public @interface JndiInjected {
   String value();
}
```

Step 2: Write Interceptor

```
public class JndiInjector {

   @PostConstruct
   public void injector(InvocationContext inv) {
      InitialContext ctx = new InitialContext();
      Object target = ctx.getTarget();

      for (Field f : target.getClass().getFields()) {
         JndiInjected ji =
           f.getAnnotation(JndiInjected.class);
         if (ji != null) {
            Object obj = ctx.lookup(ji.value());
            f.set(target, obj);
         }
      }
      … // do same for setter methods

      inv.proceed();
   }
}
```

Summary of Use Cases

- Framework components
  - Assemble them transparently
- Pluggable annotations
  - Annotations trigger interceptors
- Extending your EJB Specification Container

Real World Use Cases

- JBoss/Spring integration
  - Deploy spring packages
  - Inject deployed spring beans into EJB specification fields
- JBoss SEAM
  - Integrates EJB specifications with the context of your invocation
  - Biject HTTP Session attributes into your EJB specification

JBoss/Spring/EJB 3 Specification Integration

```
@Stateless
public class MyBean implements My {

   @Spring(bean="SomeBean")
   SomeBean bean;

…
}
```

Deployed from myspring-beans.jar

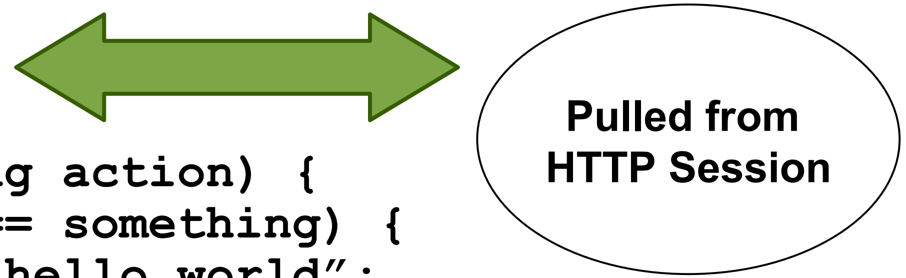
JBoss SEAM

```
@Stateless
public class ControllerBean implements Controller {

    @In @Out Model model;

    public void action(String action) {
        if (model.getData() == something) {
          model.setSomeDate(“hello world”;
        }
    }

}
```



Future Spec Enhancements?

- Extend XML binding
- Annotation Indirection

Expressions

```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>@com.titan.Audit</ejb-name>
         <interceptor-class>
            com.titan.AuditInterceptor
         </interceptor-class>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Expressions

```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>*</ejb-name>
         <interceptor-class>
            com.titan.ValidationInterceptor
         </interceptor-class>
         <method>
            <method-name>@com.titan.Validate</method-name>
         </method>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Annotation Indirection

```
@Interceptors(AuditInterceptor.class)
public @interface Audit {
}
```

- Applying annotation triggers interceptor
- Indirection abstracts implementation
- Very simple to add annotations with behaviour

Summary

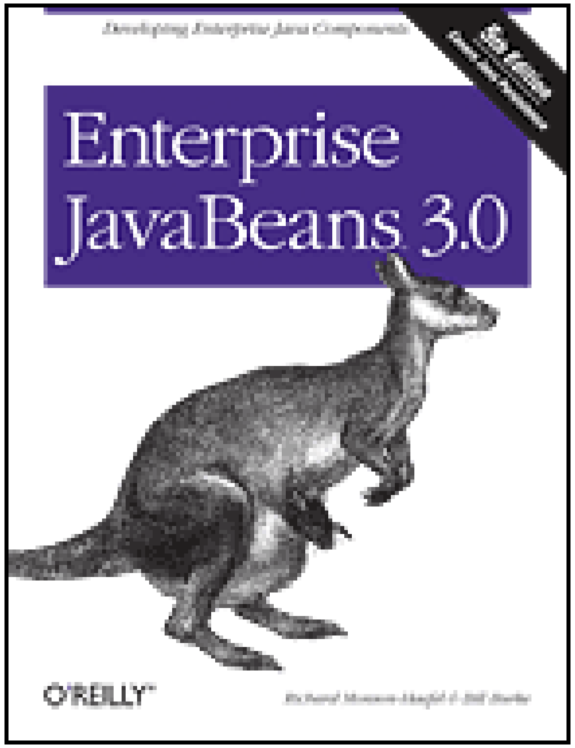
- Interceptors encapsulate cross-cutting concerns
- EJB 3.0 framework of frameworks
  - Ease of extension
  - Pluggable annotations
- Already being used in OSS products
- Specification has room to evolve

For More Information

- Contact: bill@jboss.org
- JBoss EJB3
  - www.jboss.org

Support Molly and Abby College Fund!

Buy O'Reilly's EJB 3.0 5th Edition!


Developing Enterprise Java Components
5th Edition
Enterprise JavaBeans 3.0
O'REILLY
Richard Monson-Haefel & Bill Burke

Q&A

Extending Enterprise JavaBeans™ 3.0 Specification With Interceptors

Bill Burke

Chief Architect
JBoss Inc.
www.jboss.org

TS-1365