Practical EJB 3.0

Easier for application and framework developers

Bill Burke
JBoss Fellow
Red Hat

Agenda

Using EJB with JPA

- How EJBs makes JPA easier for application developers

Practical usage of EJB Interceptors

- How EJBs makes it easier for framework developers

Speaker's Qualifications

JBoss contributor since 2001

Lead JBoss EJB 3.0 project

EJB, JPA, and Java EE JSRs

Author O'Reilly's EJB 3.0 5th edition

Most importantly….

- Of Polish decent

Using EJB 3.0 with JPA

Ease of use

Java Persistence API is an object-relational mapping layer

Allows you to map plain Java classes to a relational database

- Really a standardization of popular ORM tools like Hibernate and Toplink

```
@Entity
Public class Customer {
    @Id @GeneratedValue
    private int id;

    private String name;
    private String address;

    public getId() { return id; }
    public void setId(int id) { this.id = id; }

…
}
```

The EntityManager interface manages entity bean instances

- Analogous to the Hibernate Session

Entity bean instances must be associated with an EntityManager instance to be persistable

- Otherwise they are just plain Java objects

Entity instances associated with an EntityManager are watched

- Changes to object are persisted

The EntityManager interface manages entity bean instances

- Analogous to the Hibernate Session

Lifecycle must be maintained by user code

```
EntityManagerFactory factory =
    Persistence.createEntityManagerFactory(…);

try {
   EntityManager em = factory.createEntityManager();

   Customer cust = new Customer();
   cust.setName("Bill");

   try {
      em.getTransaction().begin();
      em.persist(cust);
      em.getTransaction().commit();
   } catch (Exception ex) {
      em.rollback();
   } finally {
      em.close();
   }
} finally {
   factory.close();
}
```

It's a little painful to…

- Manage the lifecycle of EntityManager and EntityManagerFactory
  - Tons of try/catch blocks
- Manage your own local entity transactions
- Pass EntityManager instances around
  - If other objects/methods interact within the same transaction

EJBs to the rescue

EJBs can

- Easily reference EntityManager instances
- Manage EntityManager lifecycle
- Automatically associate EntityManager with JTA transactions
- Propagate EntityManager instances automatically between nested component calls

EJBs to the rescue


```
public interface CustomerRegistry {
   public void createCustomer(String name, String address);
}

@Stateless
public class CustomerRegistryBean implements CustomerRegistry
{

   @PersistenceContext EntityManager em;

   public void createCustomer(String name, String address) {
      Customer cust = new Customer(name, address);
      em.persist(cust);
   }
}
```

EJBs to the rescue

@PersistenceContext gives us a reference to an EntityManager

Bean methods automatically start/end a JTA transaction

Lifecycle of EntityManager managed by EJB

- EntityManager instance created for duration of JTA transaction
- Destroyed at the end of the JTA transaction

EntityManager instance is automatically associated with the transaction

- This means that any other EJB that uses an EntityManager injected with @PersistenceContext in the same transaction, will use the same persistent session
- You don't have to pass around EntityManager instances to EJBs that are within the same transaction
- (Very similar to using a JTA enabled JDBC connection)

Transaction Scoped EntityManagers

@PersistenceContext injected is a transaction-scoped EntityManager

- EntityManager instance destroyed at end of the transaction
- Entities being managed by EntityManager become plain pojos
  - If you modify them, changes will not be automatically persisted

Extended EntityManagers

Extended EntityManagers can be injected into Stateful Session Beans

The EntityManager instance becomes tied with SFSB lifecycle

- Lives and dies with the SFSB instance

Extended EntityManagers and SFSBs


```
@Stateful
public class ShoppingCart implements ShoppingCart {

    @PersistenceContext(type=EXTENDED) EntityManager em;
    private Customer customer;

    public void setCustomer(String name) {
        Query q = em.creatQuery(
            "Select c from Customer where name='" + name + "'");
        this.customer = (Customer)q.getSingleResult();
    }

    public void updateAddress(String address) {
        customer.setAddress(address);
    }
    …
}
```

Change is automatically persisted

Buffering Database Updates

Interacting with an Extended persistence context outside of a transactions queues any updates/inserts/deletes

You could use this within a web application

- Page 1: create the customer
- Page 2: create the order
- Page 3: Shop and add to cart
- Page 4: Submit order

Buffering Database Updates


```
@Stateful
public class ShoppingCart implements ShoppingCart {

    @PersistenceContext(type=EXTENDED) EntityManager em;
    private Customer customer;
    private Order order;

    @TransactionAttribute(NOT_SUPPORTED)
    public void setCustomer(String name) {
        Query q = em.creatQuery(
            "Select c from Customer where name='" + name + "'");
        this.customer = (Customer)q.getResultList().get(0);
        if (this.customer == null) {
            this.customer = new Customer(name);
            em.persist();
        }
    }

…
}
```

Change not persisted

Buffering Database Updates


```
@Stateful
public class ShoppingCart implements ShoppingCart {
    …

    @TransactionAttribute(NOT_SUPPORTED)
    public void startShopping() {
        this.order = new Order();
        this.order.setCustomer(this.customer);
        em.persist(this.order);
    }

    @TransactionAttribute(NOT_SUPPORTED)
    public void addToCart(Product product, int quantity) {
        OrderEntry entry = new OrderEntry(product, quantity);
        this.order.addEntry(entry);
    }

    @Remove
    public void checkout() {
        em.flush(); // commit all changes
    }
}
```

Changes finally persisted

Summary

EJBs are easy to write

EJBs provide nice persistence context management

- Lifecycle of EntityManagers is transparent
- Transaction Association
- Buffered Database Updates

EJB 3.0 Interceptors

Ease of Extension

Interceptors Overview

EJB 3.0 specification formalizes interceptors

- Already available in proprietary products

Intercept incoming business method

- Server side only!

Intercept EJB lifecycle events

- create, destroy, activate, passivate

Purpose of Interceptors

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

Simple example: 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);

      } finally {
         long time = System.currentTimeMillis() - start;
         System.out.println("withdraw took" + time);
      }
   }
}
```

What's wrong with the 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
- Encapsulates 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 = System.currentTimeMillis() - start;
            Method method = invocation.getMethod();
            System.out.println(method.toString() +
                               " took " + time + " (ms)");
        }
    }
}
```

@AroundInvoke method

Intercepts business method invoked

Called in chain of other applied interceptors

In same Java call stack as bean method

- Wraps around
- Thrown bean exceptions may be caught

Abstraction for invoked bean method

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

Must always be called at the end of the interceptor implementation in order 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 = System.currentTimeMillis() - start;
            Method method = invocation.getMethod();
            System.out.println(method.toString() +
                               " took " + time + " (ms)");
        }
    }
}
```

InvocationContext usage


```
public class ProfilingInterceptor {

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

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

Call next interceptor or invoke bean method

InvocationContext usage

```
public class ProfilingInterceptor {

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

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

Returns object representation of bean method result

Interceptor Details

Runs in same transaction and security context as EJB method

Supports XML and annotation driven dependency injection

Belongs to the same ENC as intercepted EJB

Interceptor Lifecycle

Interceptor instance instantiated at same time as bean instance

Pooled along with bean instance

Destroyed when bean destroyed

Side effect? They can hold state

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

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 = System.currentTimeMillis() - start;
         profiler.log(invocation, time);
      }
   }
}
```

XML: Annotation Alternative


```
<ejb-jar>
   <interceptors>
      <interceptor>
         <interceptor-class>
            org.jboss.ProfilerInterceptor
         </interceptor-class>
         <around-invoke>
            <method-name>profile</method-name>
         </around-invoke>
         <ejb-ref>
           …
         </ejb-ref>
         <env-entry>
           …
         </env-entry>
      </interceptor>
   </interceptors>
</ejb-jar>
```

Applying Interceptors

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

Applying Interceptors

Accepts an array of interceptor classes

```
@Stateless
@Interceptors(org.jboss.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);
   }
}
```

@Interceptors on Class

One or more can be applied

Every business method is intercepted

Executed in order they are declared

Binding through XML


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

Binding through XML

Multiple <intereptor-class> entries allowed

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

Binding through XML


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

This is a complete descriptor!

Per-method interception

Interceptors can be applied per-method

Bean method is annotated

Executed after any class level interceptors

Applying Per-Method Interceptors


```
@Stateless
public class BankAccountBean implements BankAccount {

    @PersistenceContext EntityManager entityManager;

    public void withdraw(int acct, double ammount) {
        …
    }

    @Interceptors(org.jboss.ProfilingInterceptor.class)
    public void debit(int acct, double amount) {
        …
    }
}
```

Binding through XML


```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>BankAccountBean</ejb-name>
         <interceptor-class>
            org.jboss.ProfilingInterceptor
         </interceptor-class>
         <method>
            <method-name>debit</method-name>
         </method>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Default Interceptors

A set of interceptors can be assigned to every EJB

- Per deployment only
- Simple ejb-jar.xml description

‘*’ wildcard in <ejb-name>

Binding through XML


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

Interceptor Exception Handling

- Allowed to abort invocation
- Allowed to catch and rethrow different exceptions
- Allowed to catch and retry invocation

Aborting an Invocation Usecase

We're a Billing ISV

Want to provide customizable validation

Turn on/off validation as needed

Configure validation parameters

Optional "enterpri$e" validation

- Pay more for fraud detection

Aborting an Invocation


```
public class WithdrawValidation {

   @Resource(name="maxWithdraw")
   double maxWithdraw = 500.0;

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

Configure the interceptor


```
<ejb-jar>
   <session>
      <ejb-name>BankAccountBean</ejb-name>
      <env-entry>
         <env-entry-name>maxWithdraw</env-entry-name>
         <env-entry-type>java.lang.Double</env-entry-type>
         <env-entry-value>10000.0</env-entry-value>
      </env-entry>
   </session>
…
</ejb-jar>
```

Bind the interceptor


```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>BankAccountBean</ejb-name>
         <interceptor-class>
            org.billing.WithdrawValidation
         </interceptor-class>
         <method>
            <method-name>withdraw</method-name>
         </method>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Swap in Fraud Detection


```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>BankAccountBean</ejb-name>
         <interceptor-class>
            org.billing.FraudDetection
         </interceptor-class>
         <method>
            <method-name>withdraw</method-name>
         </method>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Aborting an Invocation Usecase

Augment Java EE base security

Use a Rule Engine

Examine content to determine if we're allowed to invoke method

Aborting an Invocation


```
public class RuleBasedSecurity {

   @EJB RulesEngine rules;

   @AroundInvoke
   public Object profile(InvocationContext invocation)
      throws Exception
   {
      if (!rules.authorized(invocation)) {
         throw new SecurityException("Failed to authorize
based on content");
      }
      return invocation.proceed();
   }
}
```

Exception Wrapping Usecase

Map SQLException to an exception hierarchy

Take vendor error numbers and convert them to:

- DeadlockException
- InvalidSqlException
- Etc…

Allow callers to catch concrete exceptions

- To handle specific db errors
- Vendor specific error handling abstracted

Exception Wrapping


```
public class SQLExceptionMapper {

   @AroundInvoke
   public Object profile(InvocationContext invocation)
      throws Exception
   {
      try {
         return invocation.proceed();
      } catch (SQLException ex) {
        switch (ex.getErrorCode()) {
          case 3344:
             throw new DeadlockException();
          case 4422:
             throw new InvalidSqlException();
          …
      }
   }
}
```

Exception Wrapping


```
@Stateless
@Interceptors(org.jboss.jdbc.SQLExceptionMapper.class)
public class MyDAOBean implements MyDao
{
    public List queryStuff() throws SQLException
    {
        …
    }
}
```

Intercepting Lifecycle Events

Re-use callback annotations

- @PostConstruct, @PreDestroy, etc.

Same signature as @AroundInvoke methods

In same Java callbacks as bean callback methods

- If the bean has a callback method

Custom Injection Annotation

Java EE 5 has no defined annotation that can inject directly from Global JNDI

Let's create a @JndiInjected annotation

- Use callback interception to implement

Custom Injection Annotation


```
@Stateless
public class MyBean implements My {

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

…
}
```

Step 1: Define Annotation


```
package org.jboss;

@Target({FIELD}) @Retention(RUNTIME)
public @interface JndiInjected {
   String value();
}
```

Step 2: Write the Interceptor


```
public class JndiInjector {


   @PostConstruct
   public void injector(InvocationContext inv) {

      InitialContext ctx = new InitialContext();
      Object target = inv.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, ji);
         }
      }
      inv.proceed();
   }
}
```

Step 3: Apply interceptor


```
<ejb-jar>
   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>*</ejb-name>
         <interceptor-class>
            org.jboss.JndiInjector
         </interceptor-class>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

Summary of Usecases

Apply orthogonal behavior transparently

- Method profiling

Aspectized exception handling

- Validation, fraud detection, custom security, jdbc wrapper

Custom injection annotations

Interceptors in Action

JBoss Business Activity

- Interceptors that perform compensating transactions

JBoss/Spring integration

- Deploy Spring packages
- Injected deployed Spring beans into EJB instances

JBoss Seam

- Integrates EJB with context of invocation
- Bi-ject HTTP Session attributes into your EJB instances

```
@Stateless
public class HotelReserverBean implements Hotel {

    @BACompensatedBy("cancel")
    @BAResult("reservation")
    public Reservation book(int room, int customer) {
      …
    }

    public void cancel(
        @BAParam("reservation") Reservation reservation) {
      …
    }

}
```

- Intercept methods annotated with @BACompensatedBy
- Log tagged parameters/results
- Register a compensation method

```
@Stateless
public class HotelReserverBean implements Hotel {

    @BACompensatedBy("cancel")
    @BAResult("reservation")
    public Reservation book(int room, int customer) {
      …
    }

    public void cancel(
        @BAParam("reservation") Reservation reservation) {
      …
    }

}
```

- Gets invoked if business activity aborts

EJB/Spring integration


```
@Stateless
public class MyBean implements My {

   @Spring(bean="myBean")
   Address billingAddress;

…
}
```

Spring beans packaged into your own .jar files and deployed by a JBoss Deployer

```
@Stateless
public class MyBean implements My {

   @In @Out Model model;

   public void action(String action) {
      if (model.getData() == something) {
         model.setSomeData("hello world");
      }
   }
}
```

Push/pull "model" from HTTP Session

Summary

Interceptors encapsulate cross-cutting concerns

EJB 3.0 is framework of frameworks

- Ease of extension
- Pluggable annotations

Already being used in OSS products

O'Reilly's EJB 3.0 5th Edition

– By Bill Burke and Richard Monson-Haefel