EJB3

The next Generation of EJBs

Your Host

Andreas Schaefer, Senior Software Engineer

andreas.schaefer@madplanet.com

Member of JSR-77 expert group
Co-author of JMX book

Former JBoss Core Contributor

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

Goals of the EJB3 Development

- Use Annotations to simplify the EJB Development
- Create as many defaults as possible so only special values must be set
- Simplify EJB Types
- Eliminate the need to implement EJB interfaces
- Eliminate the need for a Home Interface
- Eliminate EJB Callback Methods
- ...

EJB3 Ready AS

- JBoss
  - 4.0.3
  - Plus EJB3 RC3
- Oracle TNG
- Sun's Glassfish Project

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: Hello World: Bean

```
package com.madplanet.ejb3.helloWorld;

import javax.ejb.Stateless;

@Stateless
public class HelloWorldEJB3
        implements HelloWorld {

    public void greeting( String pName ) {
        System.out.println( "Hello " + pName );
    }
}
```

EJB3: Hello World: Remote Interface

```
package com.madplanet.ejb3.helloWorld;

import javax.ejb.Remote;

@Remote
public interface HelloWorld {
    public void greeting( String pName );
}
```

EJB3: Hello World: Compile and Package

- Compile the Remote Interface and the Bean
- Jar them up in an archive with extension: **.ejb3**
- Copy them into the JBoss /deploy directory

- And yes, that's it

EJB3: Hello World: Deployment

- Now what's happended:
  - JBoss loads every class in the .ejb3 file
  - JBoss looks for a class annotation indicating a
    - EJB Type: @Stateless
    - Remote Interface: @Remote
    - Local Interface: @Local
  - Deploys the bean(s) accordingly

EJB3: Hello World: Client

```
package com.madplanet.test.ejb3.helloWorld;
import ...

public class HelloWorldTest extends TestCase {
    public void testBeanCall() throws Exception {
        Hashtable lProperties = new Hashtable();
...
        InitialContext lContext = new InitialContext( lProperties );
        HelloWorld lBean = (HelloWorld) lContext.lookup(
            HelloWorld.class.getName()
        );
        lBean.greeting( "Code Camp" );
    }
}
```

Lookup returns Remote Interface !!

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: B2B: Bean

```
package com.madplanet.ejb3.b2b;

import javax.ejb.Stateless;
import javax.annotation.EJB;

@Stateless
public class RedirectorEJB3
        implements Redirector {
    @EJB
    HelloWorld mHelloWorldBean;
    public void doGreeting( String pName ) {
        mHelloWorldBean.greeting( pName );
    }
}
```

Remote Interface

EJB3: B2B

- Now what's happened:
  - When an instance of the Redirector Stateless Session Bean is invoked
    - JBoss will inject a Remote Interface instance of the HelloWorld EJB into the appropriate field
    - There is no need for a JNDI lookup anymore

EJB3: Resources

```
...
@Stateless
public class WriterEJB3
        implements Writer {
    @Resource ( mappedName="java:/OracleDS" )
    DataSource mDataSource;
    public void insert( String pFirstName, String pLastName ) {
        try {
            Connection lConnection = mDataSource.getConnection();
            ...
        } catch( Exception e ) {
        }
    }
}
```

EJB3: Resources

- Now what's happened
  - JBoss injected the Data Source looked up from the JNDI server using the mappedName as JNDI name
- Other Resources:
  - Mail
  - JMS
  - Known objects like:
    - Timer Service
    - Session Context
    - ...

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: Callback Methods

```
...
import javax.ejb.PostConstruct;

@Stateless
public class CallMeEJB3
        implements CallMe {
    URL mPropertiesURL;
    @PostConstruct
    public void init() {
        mPropertiesURL =
            Thread.currentThread().getContextClassLoader().getResource(
                "MyProperties.txt"
            );
    }
...
```

EJB3: Callback Methods

- Now what's happened:
  - JBoss invoked the 'init()' method after the EJB was created but before the client could use it
  - This is the same as the EJB2's ejbCreate() method
- Other Callback Methods:
  - PreDestroy (SL/SF)
  - PrePassivate (SF)
  - PostActivate (SF)
  - Init (SF)
  - Remove (SF)

EJB3: Callback Methods

- Callback Methods can be collected in a separate class
- Class has to be tagged with @CallbackListener annotation
- Each callback method takes the Bean instance as single parameter

```
@PostConstruct
public void init( MyBean bean ) {
...
```

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: MDBs

```
@MessageDriven(
    activateConfig = {
        @ActivationConfigProperty(propertyName="destinationType",
            propertyValue="javax.jms.Queue"),
        @ActivationConfigProperty(propertyName="destination",
            propertyValue="queue/mdb")
    }
)
public class MdbEJB3
        implements MessageListener {
    public void onMessage( Message pMessage ) {
        System.out.println( "Got Message: " + pMessage );
    }
}
```

EJB3: MDBs

- Now what's happened:
  - JBoss used the Activation Configuration to hook the MDB with the desired JMS destination
    - JMS Queue with the JNDI Name 'queue/mdb'
  - Any message send to this destination will be sent to the MDB for processing

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: Entities

```
import javax.persistence.Entity;
...
@Entity
@Table( name = "fund" )
public class Fund implements Serializable {
   private int mId;
   private String mName;
   private double mGrowthRate;

   public Fund() { }
   public Fund( String pName, double pGrowth ) {
      mName = pName;
      mGrowthRate = pGrowth;
   }
```

EJB3: Entities

```
@Id( generate = GeneratorType.AUTO )
public int getId() {
  return mId;
}
public void setId( int pId ) {
  mId = pId;
}

public String getName() {
  return mName;
}
public void setName (String name) {
  mName = name;
}
...
```

EJB3: Entities

- Now what's happened:
  - JBoss uses Entity 'Fund' to access records on table 'fund'
  - The key of the Fund record is auto generated by the DB

EJB3: Entities 1-2-n

```
import javax.persistence.Entity;
...

@Entity
@Table( name = "record" )
public class Record implements Serializable {
   protected int mId;
   protected Fund mFund;

   public Record () { }

   public Record( Fund pFund ) {
     mFund = pFund;
   }
```

EJB3: Entities 1-2-n

```
  @Id( generate = GeneratorType.AUTO )
  public int getId() {
    return mId;
  }
...
  @ManyToOne( optional=false )
  @JoinColumn( name="my_fundid" )
  public Fund getFund () {
    return mFund;
  }

  public void setFund( Fund pFund ) {
    mFund = pFund;
  }
```

EJB3: Entities 1-2-n

- Now what's happened:
  - JBoss uses Entity 'Record' to access records on table 'record'
  - The key of the Fund record is auto generated by the DB
  - The record table has a foreign relationship to table fund
  - The foreign relationship is many-to-one meaning that a Fund can referenced in many records but a record must have one reference to a fund

EJB3: Entities Save

```
@Stateless
public class FundRaiserEJB3 implements FundRaiser {
  @PersistenceContext ( unitName="cal" )
  protected EntityManager mEntity;
  public void addFund( String pName, double pGrowth ) {
    Fund lFund = new Fund( pName, pGrowth );
    mEntity.persist( lFund );
  }
...
}
```

EJB3: Entities Retrieval

```
@Stateless
public class FundRaiserEJB3 implements FundRaiser {
  @PersistenceContext ( unitName="cal" )
  protected EntityManager mEntity;
...
  public double calculate( int pFundID, double pSavings ) {
    Fund fund =
      mEntity.find( Fund.class, Integer.valueOf( pFundID ) );
    ...
    return -1;
  }
}
```

EJB3: Entities Entity Manager Definition

persistence.xml file content:

```
<entity-manager>
  <name>cal</name>
  <jta-data-source>java:/DefaultDS</jta-data-source>
  <properties>
    <property
       name="hibernate.hbm2ddl.auto"
       value="create-drop"
    />
  </properties>
</entity-manager>
```

EJB3: Entities Retrieval

```
@Stateless
public class FundRaiserEJB3 implements FundRaiser {
   @PersistenceContext ( unitName="cal" )
   protected EntityManager mEntity;

   public Collection<Fund> filterFunds( double pLow, double pHigh ) {
    return mEntity.createQuery(
        "from fund f where f.growthrate > :low AND f.growthrate < :high")
        .setParameter( "low", new Double( pLow ) )
        .setParameter( "high", new Double( pHigh ) )
        .getResultList();
  }
```

EJB3: Entities Save/Retrieval

- Now what's happened:
  - JBoss injects the Entity Manager (EM) to handle Persistence
  - The EM uses the entity to store data
  - The EM uses the entity class type to know how to search for a record by its primary key
  - Through the EM the user can create a custome query using EJB QL to make a free from query
- Entity Callback Methods:
  - Pre/PostPersist
  - Pre/PostRemoved
  - Pre/PostUpdate
  - PostLoad

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: Transactions

```
@Stateless
public class TransactionsEJB3
        implements Transactions {
   @PersistenceContext ( unitName="cal" )
   protected EntityManager mEntity;

  @TransactionAttribute( TransactionAttributeType.REQUIRED )
  public void updateExchangeRate(double pNewrate )
      throws Exception {
      ...
  }
...
}
```

EJB3: Transactions

- Now what's happened:
  - JBoss set the transaction attribute of the method to Required
- Available Transaction Attributes
  - Required
  - RequiresNew
  - Mandatory
  - Supported
  - NotSupported
  - Never
- Default Tx Attribute if not specified: Required

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: Security

```
@Stateless
@SecurityDomain( "other" )
public class SecureEJB3
        implements Secure {
  @RolesAllowed( { "AdminUser" } )
  public void addFund (String name, double growthrate) {
    ...
  }
  @PermitAll
  public Collection<Fund> getFunds () {
    ...
  }
   ...
}
```

EJB3: Security

- Now what's happened:
  - JBoss used the Security Domain 'other'
  - First Method can only be accessed by a user in Role 'AdminUser'
  - Second Method can be accessed by anybody
-

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

EJB3: Generic Interceptors

```
@Stateless
public class InterceptorsEJB3
        implements Interceptors, Serializable {

  @AroundInvoke
  public Object limitStateSize( InvocationContext ctx)
      throws Exception {

    //... do whatever desired
    // Call the next interceptor or the final method
    return ctx.proceed();
  }
}
```

EJB3: Generic Interceptors

- Now what's happened:
  - JBoss will call the Around Invoke interceptor before any business method is called
  - It is also possible to extract the Interceptor code in another class
-

EJB3: Ext. Generic Interceptors

```
public class Tracer {

  @AroundInvoke
  public Object log (InvocationContext ctx)
                        throws Exception {
     long time = System.currentTimeMillis() - start;
     System.out.println("This method takes " +
                        time + "ms to execute");
    }
  }
}
```

EJB3: Ext. Generic Interceptors

```
@Stateless
@Interceptor( Tracer.class )
public class InterceptorsEJB3
        implements Interceptors, Serializable {
...
}
```

EJB3: Summary

- Pro:
  - Injections removes the need for JNDI lookups
  - Entity Manager removes the need for Home Interfaces and make handling the persistence easier
  - Interceptors allows the developer to add features to a set of unrelated EJBs in a cross-cutting fashion
- Con:
  - Annotations are hidding in the code and making introspection of an application difficult
  - The more severe problems like class loading, unclosed connections etc. are not solved

EJB3: Table of Content

- EJB3 Presentation:
  - **Intro**
  - **Stateless Session Bean Example**
  - **Dependency Injection**
  - **Callbacks**
  - **MDBs**
  - **Entities**
  - **Transaction**
  - **Security**
  - **Generic Interceptors**
  - **Q&A**

Q & A

Andreas Schaefer

andreas.schaefer@madplanet.com

Web Log

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

Presentation can be found on

www.madplaent.com