EasyBeans User's guide

Florent BENOIT, ObjectWeb consortium

Table of Contents

Chapter 1. Introduction to EJB3

1.1. Overview

EJB3 is included in the next J2EE specification called JAVA EE 5. (http://java.sun.com/javaee/5/ [http://java.sun.com/javaee/5/])

The EJB3 specification is also called JSR 220, which can be found at http://www.jcp.org/en/jsr/detail?id=220

it is divided in three parts :

1. Core part
2. Persistence part (Which is the persistence provider)
3. Simplified specification : it contains new features.

The EJB3 persistence provider are plugged into the EJB3 container. The persistence provider available are Hibernate [http://www.hibernate.org], Speedo [http://speedo.objectweb.org] (An ObjectWeb product), etc.

1.2. Why EJB3 ?

EJB 2.x were too complex, for example, developer were using tools for making it easier.

- XDoclet (Attribute oriented programming) : http://xdoclet.sourceforge.net
- Hibernate for the persistence part : http://www.hibernate.org

The main focus on this specification is on EoD. This keyword means Ease Of Development. This is done for example by using metadata attribute annotations.

By simplifying EJB development, there should have a wider range of Java EE developers.

1.3. EJB2 vs EJB3 : EoD

The deployment descriptors are no longer required. All stuff can be done by using metadata annotations.

The CMP (Container Managed Persistence) has been simplified. This is more like Hibernate or JDO.

Programmatic defaults : For example, the transaction model is set to REQUIRED by default. The developer sets the value only if he doesn't want use this specific value.

Reduction of the use of checked exceptions : The RemoteException is not mandatory anymore on each remote business methods.

Inheritance is now allowed, so beans can extends some base code.

The native SQL queries are supported as an EJB-QL (Query Language) enhancement.

1.4. What's new

1.4.1. Metadata annotations

Metadata annotations is new : To define a stateless session bean, the @Stateless annotation is declared on the bean class.

1.4.2. Business interceptors

Interceptors are new in EJB3. It's allow the developper to intercept each business method of the bean. The parameters can be changed and the returned values too. In order to know the time that a method takes for its execution, it can be done with this kind of interceptor.

1.4.3. Lifecycle interceptors

In addition to business interceptors, the EJB2 callbacks, like `ejbActivate()` method, are now defined by using annotation. For `ejbActivate()` method, this is done with the help of @PostActivate annotation. This annotation is set on a method which will be called by the container.

1.4.4. Dependency injection

Dependency injection is a new feature. It allows to request that the container inject resources instead of trying to get them. For example, with the previous specification, in order to get an EJB, the following code was used :

```
try {
Object o = new InitialContext().lookup("java:comp/env/ejb/MyEJB");
myBean = PortableRemoteObject.narrow(o, MyInterface.clas);
} catch (NamingException e) {
....
}
```

Now, this is done by using only this code :

```
@EJB private MyInterface myBean;
```

If the @EJB annotation is found in the class, the container will lookup and inject an instance of the bean in the `myBean` variable.

1.4.5. Persistence

New things are linked to the persistence layer. For example EJB3 entities are POJO (Plain Old Java Object). It means that they can be created by using the `new()` constructor : `new MyEntity();`

`Also` entities are managed by an EntityManager. ie : entitymanager.persist(entity);

Entities have callbacks available too.

Chapter 2. Getting EasyBeans from SVN repository

Anyone can checkout source code from the SVN server. To do this, the following command should be used (For GUI SVN client use, configure it appropriately):

svn checkout svn://svn.forge.objectweb.org/svnroot/easybeans easybeans

Chapter 3. Using the examples

3.1. Compiling the examples

3.1.1. Requirements

Before trying to run the examples, the requirements in order to compile and run EasyBeans have to be followed.

3.1.2. Compile

The ant tool is used to build the examples. This time, the `build.xml` file that is used is located in the `examples` directory.

The command **`ant install_all_examples`** needs to be launched in the `examples` directory :

```
Buildfile: /home/benoitf/workspace/easybeans/examples/build.xml
install_all_examples:
init:
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/dist/clients
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/dist/ejbjars
[mkdir] Created dir: /home/benoitf/workspace/easybeans/clients
compile:
[javac] Compiling 5 source files to /home/benoitf/workspace/easybeans/output/classes
install.persistence:
install:
[copy] Copying 4 files to /home/benoitf/workspace/easybeans/ejb3s/stateless.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-stateless.jar
init:
compile:
[javac] Compiling 3 source files to /home/benoitf/workspace/easybeans/output/classes
install.persistence:
install:
[copy] Copying 2 files to /home/benoitf/workspace/easybeans/ejb3s/stateful.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-stateful.jar
init:
compile:
[javac] Compiling 4 source files to /home/benoitf/workspace/easybeans/output/classes
install.persistence:
[mkdir] Created dir: /home/benoitf/workspace/easybeans/ejb3s/entitybean.jar/META-INF
[copy] Copying 1 file to /home/benoitf/workspace/easybeans/ejb3s/entitybean.jar/META-INF
install:
[copy] Copying 4 files to /home/benoitf/workspace/easybeans/ejb3s/entitybean.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-entitybean.jar
BUILD SUCCESSFUL
Total time: 4 seconds
```

The examples are copied under the `ejb3s/` folder of the project and are available for the deployment.

Note

If EasyBeans server is running, it will detect these new applications and deploy them automatically.

3.2. Running examples

Each example has its own `build.xml` file in order to be independent from each other.

3.2.1. Stateless session bean

The `build.xml` file to use is in `examples/statelessbean` folder.

3.2.1.1. Description

This example is a stateless session bean. It contains an `helloWorld()` method which displays a text on the server side. Also, it demonstrates the use of EJB3 annotation like @Stateless.

The `trace()` method is annotated with @AroundInvoke EJB3 annotation. This method will be called at each call on a business method. The business methods are defined in the interface implemented by the SessionBean class.

The signature of the method annotated by @AroundInvoke when it is defined in the bean class, needs to follow this signature :

```
(private|protected|public) Object methodName(InvocationContext invocationContext)
   throws Exception;
```

Note

As a new feature of the EJB3, the bean's interface doesn't need to extend anymore the Remote interface.

3.2.1.2. Run the server

If the server is not available, it needs to be run by following the steps described in this guide.

3.2.1.3. Deploying the bean

The stateless session bean needs to be deployed. It is done automatically if the bean has been installed in `ejb3s` folder.

On the server side, the following output should be seen :

```
[java] INFO: Creating container for archive
/home/benoitf/workspace/easybeans/ejb3s/stateless.jar.
[java] INFO: Analyze elapsed during : 95 ms
[java] INFO: Binding bean XXX with interface XXX into registry with jndi name XXX
[java] INFO: Enhancement elapsed during : 105 ms
[java] INFO: Container started in : 274 ms
```

If these informations are on the screen, it means that the container is ready to receive client calls.

3.2.1.4. Run the client

As the container has been started, the client can be launched.

The client is run with the following ant command : **ant run.client**

If the client runs successfully, the following output is displayed :

```
[java] Calling helloWorld method...
[java] Add 1 + 2...
[java] Sum = '3'.
```

Note

In the client's code, the use of PortableRemoteObject.narrow() call is not required anymore.

3.2.2. Stateful session bean

The `build.xml` file to use is in `examples/statefulbean` folder.

3.2.2.1. Description

This example is a stateful session bean using the `SessionSynchronization` interface.

It uses the @Stateful annotation and use the default transaction model which is REQUIRED.

3.2.2.2. Run the server

If the server is not available, it needs to be run by following the steps described in this guide.

3.2.2.3. Deploying the bean

The stateful session bean needs to be deployed. It is done automatically if the bean has been installed in `ejb3s` folder.

On the server side, the following output should be seen :

```
[java] INFO: Creating container for archive
/home/benoitf/workspace/easybeans/ejb3s/stateful.jar.
[java] INFO: Analyze elapsed during : 89 ms
[java] INFO: Enhancement elapsed during : 76 ms
[java] INFO: Binding bean XXX with interface XXX into registry with jndi name XXX
[java] INFO: Container started in : 251 ms
```

If these informations are on the screen, it means that the container is ready to receive client calls.

3.2.2.4. Run the client

As the container has been started, the client can be launched.

The client is run with the following ant command : **ant run.client**

If the client runs successfully, the following output is displayed :

```
[java] Start a first transaction
[java] First request on the new bean
[java] Second request on the bean
[java] Commit the transaction
[java] Start a second transaction
[java] Buy 50 amount.
[java] Rollback the transaction
[java] after rollback, value = 30
[java] Request outside any transaction
[java] Check that value = 30
[java] ClientStateful OK. Exiting.
```

3.2.3. Entity bean

The `build.xml` file to use is in `examples/entitybean` folder.

3.2.3.1. Description

This example is an entity bean.

It describes how to use the new Java Persistence Model of an EJB3 persistence provider. To access EJB3 entities which are POJO, a stateless session bean is used. It is a facade bean.

The Entity class is a POJO class annotated with @Entity. The entities class are managed by the persistence provider.

Currently the persistence provider is provided by the Hibernate product, but ObjectWeb Speedo product should be used sooner. Users will have the choice between providers.

It It uses the @Stateful annotation and use the default transaction model which is REQUIRED.

This example shows the use of an entity bean and using EJB3 persistence provider which is in this prototype Hibernate. In a next version, the ObjectWeb Speedo product will provide an EJB3 persistence provider, so users will have the choice between these providers.

3.2.3.2. Run the server

If the server is not available, it needs to be run by following the steps described in this guide.

3.2.3.3. Deploying the bean

The entity bean needs to be deployed. It is done automatically if the bean has been installed in `ejb3s` folder.

On the server side, the following output should be seen :

```
[java] INFO: Creating container for archive
/home/benoitf/workspace/easybeans/ejb3s/entitybean.jar.
[java] INFO: Analyze elapsed during : 95 ms
[java] INFO: Enhancement elapsed during : 102 ms
[java] INFO: No persistence provider was set, set to value
org.hibernate.ejb.HibernatePersistence.
[java] INFO: Hibernate 3.1.1
[java] INFO: Using provided datasource
[java] INFO: RDBMS: HSQL Database Engine, version: 1.8.0
[...]
[java] INFO: Binding bean XXX with interface XXX into registry with jndi name XXX
[java] INFO: Container started in : 2010 ms
```

If these informations are on the screen, it means that the container is ready to receive client calls.

3.2.3.4. Run the client

As the container has been started, the client can be launched.

The client is run with the following ant command : **ant run.client**

If the client runs successfully, the following output is displayed :

```
[java] Employee with id 1 = Florent
[java] Employee with id 2 = Whale
```

Chapter 4. Writing an HelloWorld bean

4.1. Requirements

This example shows the basis of an EJB3 application with all steps used to build and run the EJB.

The only thing to know is how to run the server.

4.2. Writing the bean's code

The HelloWorld bean is divided in two parts. the business interface and the class implementing this interface.

4.2.1. Writing the Interface

The interface declares only one method, `helloWorld()` :

```
package org.objectweb.easybeans.tutorial.helloworld;

/**
* Interface of the HelloWorld example.
* @author Florent Benoit
*/
public interface HelloWorldInterface {

/**
* Hello world.
*/
void helloWorld();

}
```

Note

Even if this interface is used as a remote interface, it doesn't need to extend `java.rmi.Remote` interface.

4.2.2. Writing the business code

The following code implements the existing interface :

```
package org.objectweb.easybeans.tutorial.helloworld;

/**
* Business code for the HelloWorld interface.
* @author Florent Benoit
*/
public class HelloWorldBean implements HelloWorldInterface {

/**
* Hello world implementation.
*/
public void helloWorld() {
System.out.println("Hello world !");
}

}
```

Note

At this moment, the bean is not an EJB, this is only a class implementing an interface.

4.2.3. Defining it as a stateless session bean

Now that the code of the EJB has been written, it's the time to define the EJB application.

This bean will be a stateless session bean, so the class will be annotated with @Stateless annotation.

And the interface needs to be available for remote clients, so it will be a remote interface. This is done by using the @Remote annotation.

```
package org.objectweb.easybeans.tutorial.helloworld;

/**
* Business code for the HelloWorld interface.
* @author Florent Benoit
*/
@Stateless
@Remote(HelloWorldInterface.class)
public class HelloWorldBean implements HelloWorldInterface {

/**
* Hello world implementation.
*/
public void helloWorld() {
System.out.println("Hello world !");
}

}
```

Note

If a class implements a single interface, this interface is defined as a local interface by default.

4.2.4. Packaging the bean

The two classes (`HelloWorldInterface` and `HelloWorldBean`) have to be compiled.

Then, a folder named `ejb3s/helloworld.jar/` needs to be created and classes have to go in this folder. They will be deployed and loaded automatically.

4.3. Writing the client's code

The client access directly to the business interface and can call directly the methods of the bean.

```
package org.objectweb.easybeans.tutorial.helloworld;

import javax.naming.Context;
import javax.naming.InitialContext;

/**
* Client of the helloworld bean.
* @author Florent Benoit
*/
public final class Client {

/**
* JNDI name of the bean.
*/
private static final String JNDI_NAME =
"org.objectweb.easybeans.tutorial.helloworld.HelloWorldBean"
+ "_" + HelloWorldInterface.class.getName() + "@Remote"

/**
* Utility class. No public constructor
*/
private Client() {
}

/**
* Main method.
* @param args the arguments (not required)
* @throws Exception if exception is found.
*/
public static void main(final String[] args) throws Exception {
Context initialContext = new InitialContext();

HelloWorldInterface businessItf =
```

```
(HelloWorldInterface) initialContext.lookup(JNDI_NAME);

System.out.println("Calling helloWorld method...");
businessItf.helloWorld();
}

}
```

Note

The client doesn't call PortableRemoteObject.narrow() method. Also, no create() method is required.

4.4. Writing my first business method interceptor

An interceptor can be defined in the bean class or in another class. In this example, it will be defined in the bean's class. A business interceptor is defined by using the @AroundInvoke annotation.

The following interceptor will print the name of the method that is invoked. Of course, this can be extended to do more stuff.

```
/**
* Dummy interceptor.
* @param invocationContext contains attributes of invocation
* @return method's invocation result
* @throws Exception if invocation fails
*/
@AroundInvoke
public Object intercept(final InvocationContext invocationContext) throws Exception {
System.out.println("Intercepting method '" + invocationContext.getMethod().getName()
+ "'.");
try {
return invocationContext.proceed();
} finally {
System.out.println("End of intercepting.");
}
}
```

Caution

Don't forget to call the `proceed()` method on the invocationContext object. Else, the invocation is broken.

4.5. Writing my first lifecycle interceptor

The bean can be notified of some lifecycle events. For example when a bean is created or destroyed.

In the following example, a method of the bean will received an event when an instance of the bean is built. This is done by using the @PostConstruct annotation.

Llifecycle interceptors of a bean can be defined in another class.

```
/**
* Notified of postconstruct event.
*/
@PostConstruct
public void notified() {
System.out.println("New instance of this bean");
}
```