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, JAVA EE 5. (http://java.sun.com/javaee/5/ [http://java.sun.com/javaee/5/])

The EJB3 specification is defined in JSR 220, which can be found at the following location: http://www.jcp.org/en/jsr/detail?id=220

The publication is published as three separate files:

1. The core
2. The persistence provider
3. The simplified specification, which contains new features

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

1.2. The Advantage of EJB3

EJB 2.x was too complex. Developers were using additional tools to make it easier.

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

The main focus for this specification is on Ease Of Development (EoD). One major way this has been simplified is by using metadata attribute annotations suppported by JDK 5.0.

Simplifying EJB development should produce a wider range of Java EE developers.

1.3. EJB2 vs EJB3: EoD

The deployment descriptors are no longer required; everything can be accomplished using metadata annotations.

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

Programmatic defaults have been incorporated. For example, the transaction model is set to REQUIRED by default. The value needs to be set only if a specific value other than the default value is desired.

The use of checked exceptions is reduced; the RemoteException is no longer mandatory on each remote business method.

Inheritance is now allowed; therefore, beans can extend some of the base code.

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

1.4. New Features

1.4.1. Metadata Annotations

Metadata annotations is new. FOr example, to define a stateless session bean, the @Stateless annotation is declared on the bean class.

1.4.2. Business Interceptors

The new business interceptors allow the developer to intercept each business method of the bean. The parameters and the returned values can be changed. For example, an interceptor can be used to determine the time that a method takes to execute.

1.4.3. Lifecycle Interceptors

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

1.4.4. Dependency Injection

Dependency injection makes it possible to request that the container inject resources, instead of trying to get them. For example, with the EJB2 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) {
....
}
```

With EJB3 this is done using only the following code:

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

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

1.4.5. Persistence

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

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

In addition, entities have callbacks available.

Chapter 2. Getting EasyBeans From the SVN Repository

Anyone can check out source code from the SVN server using the following command (for GUI SVN client use, configuration values are the same as for command line use):

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 running the examples, be sure to follow the requirements for compiling and running these EasyBeans examples.

3.1.2. Compile

The ant tool is used to build the examples. To compile the examples, use the `build.xml` file that is located in the `examples` directory.

The command **`ant install_all_examples`** must 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 the 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; this allows each example to be run independently.

3.2.1. Stateless Session Bean

The `build.xml` file for this example is located in the `examples/statelessbean` folder.

3.2.1.1. Description

This example is a stateless session bean. It contains a `helloWorld()` method that displays text on the server side. Additionally, it demonstrates the use of EJB3 annotation, such as @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, must follow this signature:

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

Note

As a new feature of the EJB3, the bean's interface does not need to extend the Remote interface.

3.2.1.2. Running the Server

If the server is not available, it must be run by following the steps described in Chapter 3, "Running the EasyBeans Server."

3.2.1.3. Deploying the Bean

The stateless session bean must be deployed. If the bean has been installed in the `ejb3s` folder, this is done automatically.

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
```

Once this information is displayed on the screen, the container is ready to receive client calls.

3.2.1.4. Running the Client

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

Run the client 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 the PortableRemoteObject.narrow() call is no longer required.

3.2.2. Stateful Session Bean

The `build.xml` file for this example is located in the `examples/statefulbean` folder.

3.2.2.1. Description

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

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

3.2.2.2. Running the Server

If the server is not available, it must be run by following the steps described in Chapter 3, "Running the EasyBeans Server."

3.2.2.3. Deploying the Bean

The stateful session bean must be deployed. It is done automatically if the bean has been installed in the `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
```

Once this information is displayed on the screen, the container is ready to receive client calls.

3.2.2.4. Running the Client

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

Run the client 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 for this example is located in the `examples/entitybean` folder.

3.2.3.1. Description

This is an example of an entity bean. It describes how to use the new Java Persistence Model of an EJB3 persistence provider. To access EJB3 entities that 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 is managed by the persistence provider.

Currently, the persistence provider is supplied by the Hibernate product, but the ObjectWeb Speedo product should be available soon. Users will have the choice between providers.

This example uses the @Stateful annotation and uses the default transaction model, which is REQUIRED.

The example shows an entity bean using EJB3 Hibernate-prototype persistence provider.

3.2.3.2. Running the Server

If the server is not available, it must be run following the steps described in Chapter 3, "Running the EasyBeans Server."

3.2.3.3. Deploying the Bean

The entity bean must be deployed. It is done automatically if the bean has been installed in the `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
```

Once this information is displayed on the screen, the container is ready to receive client calls.

3.2.3.4. Running the Client

Once 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 a HelloWorld bean

4.1. Requirements

This example illustrates the basics of an EJB3 application, showing all the steps used to build and run the EJB.

The only addtional information required is to know how to run the server.

4.2. Writing Code for the Bean

The HelloWorld bean is divided into 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 does not 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 EJB code has been written, it is time to define the EJB application.

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

In addition, the interface must be a remote interface to be available for remote clients. 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`) must be compiled.

Then, a folder named `ejb3s/helloworld.jar/` must be created and classes placed in this folder. They will be deployed and loaded automatically.

4.3. Writing the Client Code

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

```
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 does not call the PortableRemoteObject.narrow() method. Also, no create() method is required.

4.4. Writing a 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 a 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 may be defined in another class.

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