Leading Open Source Middleware

EJB 3.0 Programmer's Guide

(Florent BENOIT)

- March 2009 -

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 EntityManager [http://www.hibernate.org], Apache OpenJPA [http://openjpa.apache.org/], TopLink Essentials [https://glassfish.dev.java.net/javaee5/persistence/], and Eclipse Link [http://www.eclipse.org/eclipselink/] 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. Writing a HelloWorld Bean

2.1. 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.

2.2. Writing Code for the Bean

The HelloWorld bean is divided into two parts: the business interface, and the class implementing this interface.

2.2.1. 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.

2.2.2. 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.

2.2.3. Defining the EJB Code 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;


import javax.ejb.Remote;
import javax.ejb.Stateless;

/**
 * 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.

2.2.4. 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.

2.3. 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.

2.4. 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 could be extended to perform more functions.

```
    /**
     * 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

Be sure to call the `proceed()` method on the invocationContext object; otherwise, the invocation is broken.

2.5. Writing a First Lifecycle Interceptor

The bean can be notified of certain lifecycle events: for example, when a bean is created or destroyed.

In the following example, a method of the bean will receive 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");
    }
```

Chapter 3. EasyBeans Server Configuration File

3.1. Introduction

EasyBeans is configured with the help of an easy-to-understand XML configuration file.

The following is an example of an EasyBeans XML configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<easybeans xmlns="http://org.ow2.easybeans.server">

    <!-- No infinite loop (daemon managed by WebContainer): wait="false"
         Enable MBeans: mbeans="true"
         No EasyBeans naming, use WebContainer naming: naming="false"
         Use EasyBeans JACC provider: jacc="true"
         Use EasyBeans file monitoring to detect archives: scanning="true"
         Use EasyBeans JMX Connector: connector="true"
         Enable Deployer and J2EEServer MBeans: deployer="true" & j2eeserver="true"
    -->
    <config
        wait="false"
        mbeans="true"
        naming="false"
        jacc="true"
        scanning="true"
        connector="true"
        deployer="true"
        j2eeserver="true" />

    <!-- Define components that will be started at runtime -->
    <components>
        <!-- RMI/JRMP will be used as protocol layer -->
        <rmi>
           <protocol name="jrmp" port="1099" hostname="localhost" />
        </rmi>

        <!-- Start a transaction service -->
        <tm />

        <!-- Start a JMS provider -->
        <jms port="16030" hostname="localhost" />

        <!-- Creates an embedded HSQLDB database -->
        <hsqldb port="9001" dbName="jdbc_1">
            <user name="easybeans" password="easybeans" />
        </hsqldb>

        <!-- Add mail factories -->
        <mail>
          <!-- Authentication ?
          <auth name="test" password="test" />
          -->
          <session name="javax.mail.Session factory example" jndiName="mailSession_1">
            <!-- Example of properties -->
            <property name="mail.debug" value="false" />
          </session>

          <mimepart name="javax.mail.internet.MimePartDataSource factory example"
 jndiName="mailMimePartDS_1">
            <subject>How are you ?</subject>
            <email type="to">john.doe@example.org</email>
            <email type="cc">jane.doe@example.org</email>
            <!-- Example of properties -->
            <property name="mail.debug" value="false" />
          </mimepart>

        </mail>

        <!-- Creates a JDBC pool with jdbc_1 JNDI name -->
        <jdbcpool jndiName="jdbc_1" username="easybeans"
            password="easybeans" url="jdbc:hsqldb:hsql://localhost:9001/jdbc_1"
            driver="org.hsqldb.jdbcDriver" />
```

```
        <!-- Start smartclient server with a link to the rmi component-->
        <smart-server port="2503" rmi="#rmi" />


        <!-- JNDI Resolver -->
        <jndi-resolver />

        <!-- JMX component -->
        <jmx />

        <!-- Statistic component -->
        <statistic event="#event" jmx="#jmx" />
    </components>
</easybeans>
```

By default, an `easybeans-default.xml` file is used. To change the default configuration, the user must provide a file named `easybeans.xml`, which is located at classloader/CLASSPATH.

Note

The namespace used is http://org.ow2.easybeans.server.

3.2. Configuration

Each element defined inside the `<components>` element is a component.

Note that some elements are required only for the standalone mode. JMS, RMI, HSQL, and JDBC pools are configured through JOnAS server when EasyBeans runs inside JOnAS.

3.2.1. RMI Component

The RMI configuration is done using the `<rmi>` element.

To run EasyBeans with multiple protocols, the `<protocol>` element can be added more than once.

The hostname and port attributes are configurable.

Protocols could be "jrmp, jeremie, iiop, cmi". The default is jrmp.

Note

Some protocols may require libraries that are not packaged by default in EasyBeans.

3.2.2. Transaction Component

The Transaction Component is defined by the `<tm>` element.

A `timeout` attribute, which is the transaction timeout (in seconds), can be defined on this element. The default is 60 seconds.

The implementation provided by the JOTM [http://jotm.objectweb.org] objectweb project is the default implementation.

3.2.3. JMS Component

The JMS component is used for JMS Message Driven Beans. Attributes are the port number and the hostname.

Also, the workmanager settings can be defined: minThreads, maxThreads and threadTimeout. The values are printed at the EasyBeans startup.

The default implementation is the implementation provided by the JORAM [http://joram.objectweb.org] objectweb project.

3.2.4. HSQL Database

EasyBeans can run an embedded database. Available attributes are the port number and the database name. The `<hsqldb>` may be duplicated in order to run several HSQLDB instances.

Users are defined through the `<user>` element.

3.2.5. JDBC Pool

This component allows the JDBC datasource to be bound into JNDI. The jndi name used is provided by the `jndiName` attribute.

Required attributes are `username`, `password`, `url` and `driver`.

Optional attributes are `poolMin`, `poolMax` and `pstmtMax`. This component provides the option to set the minimum size of the pool, the maximum size, and the size of the prepared statement cache.

3.2.6. Mail component

Mails can be sent by using the mail component that provides either Session or MimePartDataSource factories.

3.2.7. SmartServer Component

This component is used by the Smart JNDI factory on the client side. This allows the client to download missing classes. The client can be run without a big jar file that provides all the classes. Classes are loaded on demand.

Note

Refer to the Chapter titled, Smart JNDI Factory, for more information about this feature.

3.3. Advanced Configuration

This configuration file can be extended to create and set properties on other classes.

3.3.1. Mapping File

A mapping file named `easybeans-mapping.xml` provides the information that rmi is the CarolComponent, tm is the JOTM component, and jms is the Joram component. This file is located in the org.objectweb.easybeans.server package.

The following is an extract of the `easybeans-mapping.xml` file.

Note

The mapping file is using a schema available at http://easybeans.ow2.org/xml/ns/xmlconfig/xmlconfig-mapping_10.xsd [http://easybeans.ow2.org/xml/ns/xmlconfig/xmlconfig-mapping_1_0.xsd]

```
<?xml version="1.0" encoding="UTF-8"?>
 <xmlconfig-mapping xmlns="http://easybeans.ow2.org/xml/ns/xmlconfig"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://easybeans.ow2.org/xml/ns/xmlconfig
                                        http://easybeans.ow2.org/xml/ns/xmlconfig/xmlconfig-
mapping_1_0.xsd">

    <class name="org.ow2.easybeans.server.ServerConfig" alias="config">
        <attribute name="shouldWait" alias="wait" />
        <attribute name="useMBeans" alias="mbeans" />
        <attribute name="useNaming" alias="naming" />
        <attribute name="initJACC" alias="jacc" />
```

```
        <attribute name="directoryScanningEnabled" alias="scanning" />
        <attribute name="startJMXConnector" alias="connector" />
        <attribute name="registerDeployerMBean" alias="deployer" />
        <attribute name="registerJ2EEServerMBean" alias="j2eeserver" />
        <attribute name="description" />
    </class>

    <class name="org.ow2.easybeans.component.Components"
        alias="components" />

    <class name="org.ow2.easybeans.component.util.Property"
        alias="property" />

    <package name="org.ow2.easybeans.component.carol">
        <class name="CarolComponent" alias="rmi" />
        <class name="Protocol" alias="protocol">
          <attribute name="portNumber" alias="port" />
        </class>
    </package>

    <class name="org.ow2.easybeans.component.cmi.CmiComponent" alias="cmi">
        <attribute name="serverConfig" alias="config" />
        <attribute name="eventComponent" alias="event" />
    </class>

    <class
        name="org.ow2.easybeans.component.smartclient.server.SmartClientEndPointComponent"
        alias="smart-server">
        <attribute name="portNumber" alias="port" />
        <attribute name="registryComponent" alias="rmi" />
    </class>

    <class name="org.ow2.easybeans.component.jotm.JOTMComponent"
        alias="tm" />

    <class name="org.ow2.easybeans.component.joram.JoramComponent" alias="jms">
       <attribute name="topic" isList="true" getter="getTopics" setter="setTopics"
element="true"/>
    </class>

    <class
        name="org.ow2.easybeans.component.jdbcpool.JDBCPoolComponent"
        alias="jdbcpool" />

    <class
        name="org.ow2.easybeans.component.remotejndiresolver.RemoteJNDIResolverComponent"
        alias="jndi-resolver">
    </class>

    <package name="org.ow2.easybeans.component.hsqldb">
        <class name="HSQLDBComponent" alias="hsqldb">
            <attribute name="databaseName" alias="dbName" />
            <attribute name="portNumber" alias="port" />
        </class>
        <class name="User" alias="user">
            <attribute name="userName" alias="name" />
        </class>
    </package>

    <package name="org.ow2.easybeans.component.quartz">
        <class name="QuartzComponent" alias="timer" />
    </package>

    <package name="org.ow2.easybeans.component.mail">
      <class name="MailComponent" alias="mail" />
      <class name="Session" alias="session">
        <attribute name="JNDIName" alias="jndiName" />
      </class>
      <class name="MimePart" alias="mimepart">
        <attribute name="subject" element="true" />
        <attribute name="JNDIName" alias="jndiName" />
      </class>
      <class name="MailAddress" alias="email" element-attribute="name" />
      <class name="Auth" alias="auth">
        <attribute name="username" alias="name" />
      </class>
    </package>

    <class name="org.ow2.easybeans.component.event.EventComponent" alias="event">
      <attribute name="eventService" alias="event-service" optional="true" />
    </class>

    <class name="org.ow2.easybeans.component.jmx.JmxComponent" alias="jmx">
```

```
      <attribute name="commonsModelerExtService" alias="modeler-service" optional="true" />
    </class>

    <class name="org.ow2.easybeans.component.statistic.StatisticComponent"
 alias="statistic">
        <attribute name="eventComponent" alias="event" />
        <attribute name="jmxComponent" alias="jmx" />
    </class>

    <package name="org.ow2.easybeans.component.depmonitor">
        <class name="DepMonitorComponent" alias="depmonitor">
        </class>
        <class name="ScanningMonitor" alias="scanning">
            <attribute name="waitTime" alias="period" />
        </class>
        <class name="LoadOnStartupMonitor" alias="loadOnStartup">
        </class>
    </package>


</xmlconfig-mapping>
```

Note

This mapping file is referenced by the easybeans configuration file using the XML namespace : xmlns="http://org.ow2.easybeans.server".

Each element configured within this namespace will use the mapping done in the `org.ow2.easybeans.server` package.

Users can define their own mapping by providing a file in a package. The name of the the file must be `easybeans-mapping.xml or element-mapping.xml`.

Example: For the element `<easybeans xmlns="http://org.ow2.easybeans.server">`, the resource searched in the classloader is `org/ow2/easybeans/server/easybeans-mapping.xml`. And for an element <pool:max>2</pool:max> with xmlns:pool="http://org.ow2.util.pool.impl", the resource searched will be org/ow2/util/pool/impl/easybeans-mapping.xml or org/ow2/util/pool/impl/pool-mapping.xml.

3.3.2. Other Configuration Files

EasyBeans can be configured through other configuration files as it uses a POJO configuration. If done this way, it can be configured using the Spring Framework component or other frameworks/tools.