EJB3 on the JBoss Application Server

Will Provost
Ken Kousen

Version 4.2

EJBOnJBoss. EJB3 on the JBoss Application Server
Version 4.2

A publication of Capstone Courseware, LLC.
877-227-2477
www.capstonecourseware.com

Published in the United States.

This book is printed on 100% recycled paper.

Introduction

- **This add-on module shows how the JBoss application server supports EJB 3.0 development.**
- **It is designed to expand on training available in Capstone's course "Introduction to EJB," which targets the Java EE 5.0 SDK as its application server, compilation class path, and JPA environment.**
- **Understanding JBoss's support for EJB3 involves a few key topics:**
  - How to use **JNDI** in JBoss
  - Differences in JBoss's **EJB naming**
  - Which **Java EE 5** specifications JBoss 4.2 does and doesn't implement
  - Quirks and foibles of **Hibernate** as a **JPA provider**
- **We'll see that JBoss supports EJB3 completely, but that as of the 4.2 release it does not yet support many surrounding Java-EE-5 specifications, such that compromises are necessary when using EJBs from outside the EJB container.**

Code Organization

- The examples in this module follow the code organization of the primary EJB course.

```
<root> (e.g. Examples/Wholesale)
    build.properties
    build.xml
    data
        create_DB.sql
        persistence.xml (for SE applications)
        remove_DB.sql
    doc (with generated javadoc from the src tree)
    docroot
        this.jsp
        that.jsp
        central.css
        WEB-INF
            web.xml
    META-INF
        application.xml
    ejb
        META-INF (when EJB descriptors are needed)
            ejb-jar.xml
            persistence.xml (for EJBs)
    src
        (all Java packages and source here)
```

- The **Ant** directory holds different files:
  - **JBoss4.2Targets.xml** instead of **AS9.0Targets.xml**
  - **JBoss4.2.properties** instead of **AS9.0.properties**
- We'll discuss the key differences in building and deploying throughout this chapter.

Tools and Environment

- **This module relies on the following tools for lab software:**
  - A **Java 5.0 developer's kit**
  - The **JBoss** application server, version **4.2**
  - The **Ant** make utility, version **1.6**
- **Assure that the following environment variables are set:**
  - **CC_MODULE** is typically **c:\Capstone\EJBOnJBoss**
  - **JBOSS_HOME** must point to the root of your JBoss installation
  - The executable **PATH** must include the Java **bin** directory and that for Ant, which is usually **c:\Capstone\Tools\Ant1.6\bin**.
- **Start JBoss now by running run.bat from the directory server/default/bin under JBOSS_HOME.**
- **Also, we use MySQL as the management system for our relational databases.**
- **To start the MySQL RDBMS, simply run the following command from c:/Capstone/Tools/MySQL5.0/bin:**

```
mysqld
```

  - Note that the console you use to run this command will appear to hang; this is the MySQL daemon process listening for connections at its default port 3306.
- **When you want to shut the server down again, run this prepared script from the same directory:**

```
shutdown
```

JBoss and JNDI

- **JNDI has been infamous over the years: difficult to use from all but the most managed environments, and rather opaque – that is it's often difficult to diagnose problems when they occur.**
- **By contrast, it's really simple to use JNDI on JBoss, and we'll soon see a simple application that makes it easy to view the tree of published names and objects at any time.**
- **Components (servlets, EJBs, etc.) running in JBoss containers have automatic visibility to their component environments, which are simple, private namespaces populated with names defined with a given component's needs in mind.**
  - No additional configuration is necessary to support JNDI lookups from within such a managed component.
- **Standalone (Java SE) applications have a little more work to do, since they don't enjoy the services of a Java EE container.**
  - The **class path** must include certain JARs to make JBoss's JNDI implementation available.
  - Two **system properties** must be set to assure that an initial JNDI context object knows how to connect to a server.
  - These could also be set programmatically using a different constructor for the **InitialContext** class.

A JNDI Report

EXAMPLE

- In **Examples/JNDIReport** is a Java SE application that reads the entire naming tree published by the JBoss server.
- See **src/cc/jndi/JNDIReport.java**:
  - The **main** method creates an initial naming context and passes it to the recursive method **list**:

```
list (new InitialContext (), "", showSomeClass);
```

  - This method gets the simple name associated with the given context ...

```
NamingEnumeration enumeration = context.list ("");
while (enumeration.hasMore ())
{
  NameClassPair pair =
    (NameClassPair) enumeration.next ();
  System.out.println (indent + pair.getName ());
  ...
```

  - ... and then recurses, if the type of the object at this node is a JNDI **Context** itself:

```
  ...
  if (Context.class.isAssignableFrom
      (Class.forName (pair.getClassName ())))
    list ((Context)context.lookup (pair.getName()),
      indent + "  ", showSomeClass);
  ...
}
```

A JNDI Report

EXAMPLE

- **This code would work fine from many different execution contexts; the question is how that InitialContext object is connected to a naming context, and what that context will be.**
  - For managed components, it will be the component environment.
  - For this application, we must set system properties that will be read by the constructor.
  - We've encoded these into our Ant **run-jndi-client** target – see **Ant/JBoss4.2Targets.xml**:

```
<target name="run-jndi-client" >
  ...
  <java fork="on" classname="${app-class}" >
    <classpath>
      <path refid="launch-path" />
    </classpath>
    <sysproperty
      key="java.naming.factory.initial"
      value=
        "org.jnp.interfaces.NamingContextFactory"
    />
    <sysproperty
      key="java.naming.provider.url"
      value="jnp://${jndi-host}:${jndi-port}"
    />
    <arg line="${args}" />
  </java>
</target>
```

A JNDI Report

EXAMPLE

- With your server running, build and test this application:

```
ant
report (which runs 'ant run-jndi-client')
TopicConnectionFactory
jmx
  invoker
    RMIAdaptor
  rmi
    RMIAdaptor
HTTPXAConnectionFactory
ConnectionFactory
UserTransactionSessionFactory
...
```

- We'll use this tool in later examples, to check a few things about how JBoss names EJBs.

JBoss and EJB3

- **JBoss 4.2 includes a complete update to the EJB container that fully supports the EJB 3.0 specification – including:**
  - EJB3 annotations such as **@Stateless** and **@MessageDriven**
  - The Java Persistence API, hence support for **@Entity**s
  - Lifecycle annotations such as **@PostConstruct** and **@PreDestroy**
  - Dependency injection for annotations including **@EJB**, **@Resource**, and **@PersistenceContext**
- **As we're about to see, code from the "Introduction to EJB" course – which was written to work with the Java EE 5 SDK server – ports easily to JBoss, with no changes to Java code at all.**
  - There is a **jboss.xml** that replaces **sun-ejb-jar.xml**, and this is normal for declaring server-specific values such as JNDI names.
- **There are some variations:**
  - JBoss doesn't support the **mappedName** attribute; this is optional under the specification, and JBoss opts out.
  - JBoss defines different **default JNDI names** for EJBs.

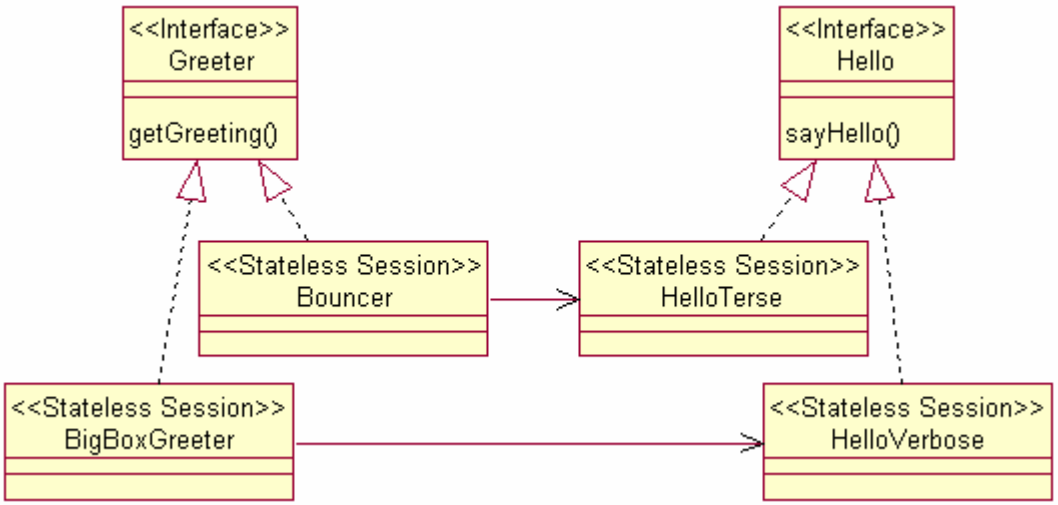
JBoss and Java EE 5

- **JBoss 4.2 is not a complete Java EE 5 application server.**
- **It is a J2EE 1.4 application server with certain plug-ins that support specific Java EE 5 technology, including EJB3.**
- **The JBoss web and application-client containers don't yet support lifecycle annotations.**
- **Neither do they support dependency injection.**
  - This may be a surprise: the **@EJB** annotation won't work, except for an EJB that needs to use another EJB!
  - Servlets and other components will need to perform explicit **JNDI lookups**.
  - Even there, we run into some trouble with J2EE 1.4-5.0 compatibility: the schema for J2EE 1.4 requires an EJB reference to define a **home interface**, while a 3.0 EJB has none.
  - We fake this by using simpler **resource references** – incorrect, but it's about the best we can do until JBoss 5.0 cleans things up.
- **Note that JSF managed beans in JBoss do enjoy Java EE 5 annotation support.**

Greeters

EXAMPLE

- In **Examples/Hello** is a simple EJB application:



  - Two **local beans** implement the **Hello** interface.
  - Two **remote beans** implement the **Greeter** interface, and each delegates to one of the local beans.
  - A client application looks up each of the two remote beans and calls its **getGreeting** method.
- **The remote beans find their local delegate beans in the usual way – see ejb/META-INF/ejb-jar.xml.**

Greeters

EXAMPLE

- **These same two remote beans take two different approaches to defining their public JNDI names:**
  - For **BigBoxGreeter** there is a JBoss-specific description that defines the JNDI name:

```
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>BigBoxGreeter</ejb-name>
      <jndi-name>ejb/BigBoxGreeter</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```

  - **Bouncer** declares a JNDI name using the **mappedName** attribute, but we'll see that this is ineffective with JBoss and that a default name will be generated instead.

Greeters

EXAMPLE

- **Build and deploy the application:**

ant

- **Run the JNDIReport application (from Examples/JNDIReport) and see the new names – for remote as well as local beans, by the way:**

```
report
Hello
  Bouncer
    remote
  HelloTerse
    local
  HelloVerbose
    local
...
ejb
  BigBoxGreeter
```

  - Notice especially that the **Bouncer** bean winds up being published at a name defined by JBoss using the application name, the bean class name, and the token "remote".
  - It's **mappedName** attribute is ignored.

Greeters

EXAMPLE

- The client application in **src/Client.java** performs explicit JNDI lookups to these two names:

```
Context context = new InitialContext ();
bouncer = (Greeter)
  context.lookup ("Hello/Bouncer/remote");
bigBox = (Greeter)
  context.lookup ("ejb/BigBoxGreeter");
```

  - Again, the **@EJB** annotation would not work here.

- Run this client application and see that it connects cleanly:

```
ant run-jndi-client
Greeting from bouncer:
Hi. ID, please?

Greeting from big-box greeter:
Well, hi, there!  Nice to see you. Can I help you
find something today?
```

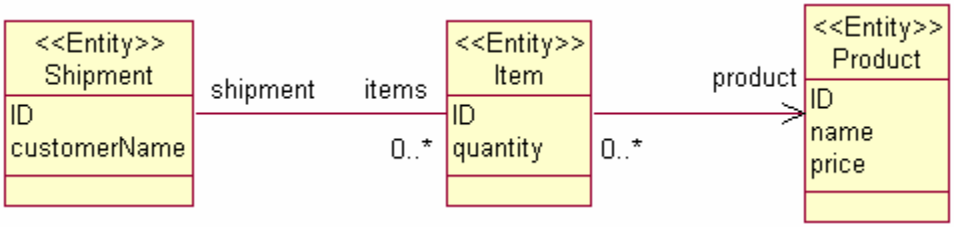
Hibernate as a JPA Provider

- **JBoss includes the Hibernate ORM framework for persistence.**
- **In JBoss 4.2, Hibernate is re-purposed as a JPA provider.**
  - This means that JPA annotations including **@Entity** are supported.
  - The **persistence.xml** file takes the place of the usual **hibernate.cfg.xml**, and Hibernate is the default JPA provider.
  - Hibernate **mapping files** are no longer needed, because JPA annotations such as **@Table**, **@Column**, **@ID**, **@OneToMany**, etc.
- **Some surprises in configuring JPA persistence using JBoss and Hibernate, however:**
  - **persistence.xml** must declare the set of persistent Java classes that Hibernate will support; JARs in the application will not be scanned for **@Entity** classes automatically as they are with TopLink (the default provider for the Java EE 5 SDK).
  - **Eager fetching** as defined by JPA annotations doesn't seem to be supported. The example we'll see in a moment relies on Java code to carry out an explicit fetch of sub-objects, which it shouldn't need to do.
  - The JPA **EntityManager**'s **merge** operation doesn't cascade as it should. Some of our updates in the upcoming example will be incomplete!

Wholesale Entities

EXAMPLE

- In **Examples/Wholesale** are the **Shipment**, **Item**, and **Product** entities familiar from the "Introduction to EJB" course:



- We use exactly the same database schema as well.
- We define the JPA persistence unit to use a JDBC data source (defined for JBoss in the separate file **data/Wholesale-ds.xml**) – see **ejb/META-INF/persistence.xml**:

```
<persistence-unit
  name="WholesaleService"
  transaction-type="JTA"
>
  <jta-data-source>java:/Wholesale
    </jta-data-source>
  <class>cc.sales.Item</class>
  <class>cc.sales.Product</class>
  <class>cc.sales.Shipment</class>
  <properties>
    <property
      name="hibernate.dialect"
      value="org.hibernate.dialect.MySQLDialect"
    />
  </properties>
</persistence-unit>
```

  - Note too the **<class>** elements, informing Hibernate (acting as a JPA provider) what Java classes to manage as entities.

Wholesale Entities

EXAMPLE

- Much of the rest of the application is unchanged.
- However, when retrieving a **Shipment** by ID, we must perform a deep fetch from the Java code, to assure that the whole tree of **Item**s and **Product**s is instantiated.
  - See **src/cc/sales/ejb/RecordsImpl.java** for the **getShipment** method:

```
public Shipment getShipment (int shipmentID)
{
  Shipment s =
    em.find (Shipment.class, shipmentID);
  if (s != null)
  {
    for (Item i : s.getItems ());
    return s;
  }
  ...
```

  - This is necessary even though the **Shipment** class requests eager fetching of its **items** collection:

```
@OneToMany(...)
@Basic(fetch=FetchType.EAGER)
private List<Item> items = new ArrayList<Item> ();
```

Wholesale Entities

EXAMPLE

- **With these minor adjustments, the original code runs smoothly under JBoss; build and test the JPA pieces as follows:**
  - Start MySQL, if you haven't already, and build the database:

```
ant create-DB
...
5 of 5 SQL statements executed successfully
```

  - Build and test the Java SE applications that prime and read the database:

```
ant
run PrimeWithData
...
Persisting products ...
Persisting shipments ...
run ListShipments
...
Shipment (1,Herring and Elephant Co.)
    Item: (1,1,Red herring)
    Item: (2,2,White elephant)
Shipment (2,Just Herring)
    Item: (3,3,Red herring)
Shipment (3,Collectors, Inc.)
    Item: (4,1,Red herring)
    Item: (5,1,White elephant)
    Item: (6,1,Rubber biscuit)
    Item: (7,1,Wooden nickel)
```

Wholesale Web Tier

EXAMPLE

- **The web tier of the Wholesale application has changed only slightly, to work around the lack of Java EE 5 dependency injection in JBoss's Servlets-2.4 container.**
  - In **src/cc/sales/web/InitServlet.java**, instead of using the **@EJB** annotation, we perform a more traditional JNDI lookup, getting the necessary absolute name from our component environment:

```
public void init ()
  throws ServletException
{
  try
  {
    config.getServletContext ().setAttribute
      ("records", new InitialContext ().lookup
        ("java:comp/env/Records"));
  }
  ...
```

  - **ProcessingServlet.java** carries out a similar lookup to get its **Fulfillment** delegate.

Wholesale Web Tier

EXAMPLE

- **Both lookups are satisfied with the help of the component environment definition in docroot/WEB-INF/jboss-web.xml:**

```
<jboss-web>
  <resource-ref>
    <res-ref-name>Records</res-ref-name>
    <res-type>cc.sales.ejb.Records</res-type>
    <jndi-name>Wholesale/RecordsImpl/local
      </jndi-name>
  </resource-ref>
  <resource-ref>
    <res-ref-name>Fulfillment</res-ref-name>
    <res-type>cc.sales.ejb.Fulfillment</res-type>
    <jndi-name>Wholesale/FulfillmentImpl/local
      </jndi-name>
  </resource-ref>
</jboss-web>
```

  - Again – full disclosure! – we're using **<resource-ref>** and not the more proper **<ejb-local-ref>**, because the incompatibility between the J2EE 1.4 servlets container and the Java EE 5 EJB container makes the latter approach unworkable.

Wholesale Web Tier

EXAMPLE

- **One other note: the lazy-fetching problem mentioned earlier wouldn't be a problem (or it would only be a performance problem) if we had a single Hibernate session per web request.**
  - A JPA-aware web tier could do this, but instead we get a **Hibernate session per EJB method call**.
  - It's the detachment of the **Shipment** and it's later use in another EJB method call that trips up Hibernate.
  - Without our eager-fetching code, Hibernate would later complain of its inability to "lazy-fetch" sub-objects of a **Shipment** instance that wasn't retrieved in the current session.
- **Run the web application by visiting the following URL.**

```
http://localhost:8080/Wholesale
```

- **You'll see that processing orders works fine:**

Wholesale Web Tier

EXAMPLE

- **Edit a shipment definition, however, and you may notice that some operations work, and some don't.**
  - Specifically, the **merge** operation called when you click the **Done** button will catch any **new or modified objects**, but it will fail to **remove objects** that are no longer in the **items** collection.
  - Try editing the "Herring and Elephant Co." shipment by adding a Rubber Biscuit, changing the Red Herring quantity to 12, and removing the White Elephant line item.
  - Click **Done** and then return to view the shipment again:

Select	Product	Price	Quantity
☐	Red herring	$10.00	12
☐	White elephant	$20.00	2
☐	Rubber biscuit	$1.99	1

  - The add and edit worked; the remove did not.
  - We've left this part of the code "un-fixed" to illustrate the problem; the necessary workaround would involve looping through the items explicitly and inserting, updating, or removing them.