Edition

1

Date: 2002-07-16, 9:15:31 PM

ANDREAS SCHAEFER

The JBoss Group

JBoss 3.0 Quick Start Guide

ANDREAS SCHAEFER, AND THE JBOSS GROUP

JBoss 3.0: Quick Start Guide

Table of Content

Table of Listings

Table of Figures

Preface

Forward

About the Authors

Andreas Schaefer, was born in Liestal, Switzerland in 1964. He started as laboratory assistance for Norvartis, went back to high school and studied theoretical physics in Basel, Switzerland, where he graduated with a B.S. After working for over 10 years as software engineer he immigrated to Los Angeles, CA. Andreas currently serves as Senior Software Engineer for the JBoss Group LLC.

JBoss Group LLC, headed by Marc Fleury, is composed of over 100 developers worldwide who are working to deliver a full range of J2EE tools, making JBoss the premier Enterprise Java application server for the Java 2 Enterprise Edition platform.

JBoss is an Open Source, standards-compliant, J2EE application server implemented in 100% Pure Java. The JBoss/Server and complement of products are delivered under a public license. With 50,000+ downloads per month, JBoss is the most downloaded J2EE based server in the industry.

Dedication

I dedicate this book to my son Andreas Carl jr. filling my heart with joy and my wife Manuela giving me the support to work for JBoss.

We dedicate this book to the users who submit good bug reports.

Acknowledgments

I want to thank all contributors of this documentation for their hard work and time they spent to make ease the usage of JBoss 3 for all users.

1. Introduction

Welcome to JBoss Community by Marc Fleury

To be completed

2. First Steps: Bring JBoss to Life

Download, compile, start and test your JBoss 3.0 Application Server

Before you can start deploying your J2EE applications you need to download, compile or install and test your JBoss 3.0 application server. This chapter shows how to do this with the source and binary distribution. At the end you will have a complete and tested application server ready to serve you.

Binary Download

You can download JBoss ready to run. Because JBoss comes with a simple Database (Hypersonic) and Jetty / Tomcat web server you can use it out of the box without any initial configuration.

You can find the binary downloads either on from the

- JBoss Project Page on Source Forge: here you will always find the most and up to date downloads, the bug and patch databases and much more
- JBoss Home Page: here you will find much useful information about JBoss, the official, free forum, project information etc. and also the download page.

All downloads are binary downloads except the one listed with "-src" at the end of the file (before the extension). Therefore this file "JBoss-2.4.4-src.tgz" contains the source code.

To download a file just click on the appropriate link and save it to your disk. For some files (on windows box files with extension "tgz") must be forced to be downloaded because the browsers tries to download and present it as regular page. On Windows make a right-click on the link and select "Save As" (Netscape) or "Save Target As" (IE).

Source Code Download

As mentioned above source can be downloaded from the JBoss download pages. When you like either to get the most accurate source or want to start coding in JBoss it is preferable to download the source from CVS repository.

Here is a checklist how to download JBoss from CVS:

- Check if you have a CVS client available (CVS, jCVS, WinCVS). If not then download and install either it either from www.cvs.org, www.jcvs.org, or www.wincvs.org.

- Another way for Windows users is to download and install Cygwin (http://sources.redhat.com/cygwin) a Unix emulator. You can do so by downloading and starting the Cygwin-Installer. Now check that CVS and VI or VIM is selected and start the installation. After the installation is finished you can open a "bash" shell with most Unix commands available (inclusive CVS). Now you can work in this shell nearly as if you were working on a Unix box. Noteworthy is that you can also run windows scripts (*.bat) in this shell as well.

- For CVS clients without developer access type:

```
cvs -d:pserver:anonymous@cvs.jboss.sourceforge.net:/cvsroot/jboss login
cvs -z3 -d:pserver:anonymous@cvs.jboss.sourceforge.net:/cvsroot/jboss co jboss-all
```

- For jCVS or WinCVS clients set CVS-Server to "cvs.jboss.sourceforge.net", select server-type as "PServer", set CVS Repository to "/cvsroot/jboss" and user name to "anonymous" with empty password. Then checkout the CVS Module "jboss-all".

- After you have checked out the "jboss-all" module the next time you want to update the source code just go in the root directory (/jboss-all) and perform an update with:

```
cvs -z3 -d:pserver:anonymous@cvs.jboss.sourceforge.net:/cvsroot/jboss update -dP
```

or with the appropriate commands in jCVS or WinCVS. This will update your source with the current in the CVS repository. At the end please check that you do not have a conflict on a file (in CVS the file is marked with C at the beginning of the line and a warning is printed out).

Compile and Test JBoss

After you downloaded JBoss from CVS or downloaded JBoss source archive you can now start to compile and test the compilation afterwards

- Change to the project root directory.

- Then change to the build directory.
- Start the build file without a parameter. When you want to clean it beforehand then add the following parameters “clean most”.
- After a successful compilation you can test the implementation by running against the test suites available in JBoss 3.0.
- First change to the project root directory.
- Then change to the build/output/jboss-3.XXX/bin directory.
  Note: XXX in the directory above means that here comes in the current version like "0.0beta2".
- Start JBoss 3.0 by using the startup file. Wait until JBoss server is up and running.
- Open a new window (or shell) and change to the project root directory.
- Then change to the test suite directory (/testsuite)
- Now you can start the test suite by entering at the prompt “build.bat run-basic-testsuite”
  Note: There are more test suites, to see what is available enter the follwing parameter to the build script: “help”.
- Open the file “testsuite/output/reports/html/index.html” in your browsers to see the reports of the test suite.

Start JBoss

When you want to start JBoss you have to go to the “bin” directory either of your distribution or when you compiled it from scratch in the “build/output/jboss-3.XXX” directory. Now you just start the appropriate batch file (run.bat or run.sh) and the window will list the output of the application server according to the logging settings. When you see the line “JBoss kernel started” JBoss is ready to serve. Please note that because no custom application is deployed you will most likely not be able to see any web pages served by JBoss meaning that when you open a web server to see `http://localhost:8080` you would get an error page. To check if JBoss is running please open a browser and enter `http://localhost:8082` which will list all JBoss components running. This page is served by the HTML-Adaptor and is a way to manage JBoss.

JBoss Startup Options

JBoss 3 has like JBoss 2.x the feature to start a specific environment. But unlike JBoss 2.x 3 allows to start a completely different server evironment. This means you have set up a server with the specific configuration, deployments and libraries. Thus you can avoid carrying around a lot of unnecessary files. To select the appropriate environment specify the directory name of the server environment with "-c" option. To start JBoss 3 with the "all" server environment would look like:

```
run.bat -c all
```

The server environements are added in the "server" directory in the JBoss root directory. Whenevern you need a new environment copy the most appropriate server environment and adjust it to your needs. With the JBoss distribution you have these three server environements available:

- **minimal:** which is the bare minimum to start JBoss 3. It contains logging, JNDI server and URL deployment scanner to find new deployments. There is no EJB container, JMS, clustering etc.
- **default:** which is the default server environment that is started when no server environment is specified. It contains all except clustering and RMI/IIOP service.
- **all:** contains all available services

JBoss as Unix / Windows Startup Service

Install JBoss as Windows NT / W2K Service

There are several NT Service wrappers around to start a Java program as NT / W2K Service and I choose a program available from Alexandria's JavaService. First check if a never version is available but if you don't find it take version 1.2.0. After download extract it do your desired location with the restriction that it must be available when the service is started (I would discourage network paths, removable hard disks etc. and strongly recommend local hard disks).

Now we are going to install the W2K service. In order to do that you need your JBoss 3.0 distribution available also to the local system, ensure that you have administrator rights to do so and the local system has permission to start JBoss or that you know the username/password of a registered user having the permission to start JBoss.

Take this code and save it as a ".BAT" file where your JavaService.exe is located

Listing 2-1, JBoss30.bat script file

```
@echo off

if "%1" == "uninstall" goto uninstall
if "%1" == "-uninstall" goto uninstall
if "%1" == "" goto usage
if "%2" == "" goto usage
if "%3" == "" goto usage
if "%1" == "-help" goto usage
if "%1" == "-?" goto usage
if "%1" == "/?" goto usage

:install
JavaService.exe -install JBoss30 %1\jre\bin\%3\jvm.dll -
Djava.class.path=%1\lib\tools.jar;%2\bin\run.jar -start org.jboss.Main -stop org.jboss.Main
-method systemExit -out %2\bin\out.txt -current %2\bin
goto eof

:uninstall
JavaService.exe -uninstall JBoss30
goto eof

:usage
echo -------- To Install JBoss 3.0 do
echo Usage:   %0 jdk_home jboss_home (classic/hotspot/server)
echo NOTE:    You MAY NOT use spaces in the path names. If you know how
echo          to fix this, please tell me.
echo          JDK 1.3 does not come with hotpot server by default, you must
echo          install this seperately if you wish to use it.
echo Example: %0 c:\progra~1\jdk c:\progra~1\jboss30 hotspot
echo --------
echo -------- To Uninstall JBoss 3.0 do
echo Usage:   %0 uninstall
echo --------
goto eof

:eof
```

Start this program with the JDK home path as first argument, the JBoss home path as the second and type of JVM (classic, hotspot or server) as third. Please check if the desired type of JVM is available in JDK_HOME/jre/bin and see if the classic, hotspot or server directory is available. This is an example of how to start the script: JBoss30.bat

```
JBoss30.bat c:\java\jdk1.3.1 c:\java\jboss30 server
```

Open the "Control Panel". Then for WNT click on "Services" and for W2K click on "Administrative Tools" and then click on "Services". Now look for the "JBoss30" service (or how you named it in the JBoss30.bat script) and open it.

Check that the service its startup type is "Automatic" if it should be started when the Windows is started up or "Manual" if you want to start it here manually. Then ensure on the "Log On" tab that the service is started with the right user having the proper permissions to start JBoss.

Now finally start the service and check that it is serving (takes some time). The console output is redirected to the file specified in the script with "-out" parameter.

ATTENTION: Note that the order of the parameter of the JavaService.exe program matters and that you should not use spaces in the paths.

Install JBoss as Unix Service

To start installing JBoss as Unix Service you have to have a JBoss 3.0 distribution, an "init.d" script and a JBoss startup script.

Important:
This solution will run JBoss as super-user and therefore when someone can break into your system this is a security problem. Whenever you want to start a service at a port lower than 1024 you have to run it as super-user like web server port 80. A way to go around this is to call the JBoss script as another user (like "nobody") and redirect the port 8080 to port 80 (see Unix command "ipchain" or "iptables" for more).

Copy the JBoss 3.0 distribution at its destination and check that you can let it run as super-user.

Take this script, save it in your /etc/init.d (could also be somewhere else like /sbin/init.d etc.) and make it runnable by the super-user.

Listing 2-2, JBoss "init.d" shell script

```
################################
### Contents of jboss
###
### (Make sure #!/bin/sh is the first line of the file and the file
### has execute permissions for root.)
################################

#!/bin/sh
```

```
#
# Startup script for JBOSS, the J2EE EJB Server
#
# chkconfig: 2345 95 15
# description: JBoss is an EJB Server
# processname: jboss
# pidfile: /var/run/jboss.pid
# config: /usr/local/jboss/conf/default/jboss.conf
# logfile: /usr/local/jboss/log/server.log
#
# version 1.0 -
# version 1.1 - kjenks - Start Tomcat, too.
#

# Source function library.
. /etc/rc.d/init.d/functions

#SET THE FOLLOWING LINE TO YOUR JAVA_HOME
export JAVA_HOME=/usr/java/bin

#SET THE FOLLOWING LINE TO YOUR CORRECT JBOSS_HOME
export JBOSS_HOME=/usr/local/jboss

export PATH=$PATH:$JBOSS_HOME/bin:$JAVA_HOME/bin:$JAVA_HOME/jre/bin

#IF YOU NEED SPECIAL CLASSES IN YOUR CLASSPATH
#AT STARTUP, ADD THEM TO YOUR CLASSPATH HERE
#export CLASSPATH=

RETVAL=0

# See how we were called.
case "$1" in
   start)
         cd $JBOSS_HOME/bin
         echo -n "Starting jboss daemon: "
         daemon $JBOSS_HOME/bin/go.sh start
         RETVAL=$?
         echo
         [ $RETVAL -eq 0 ] && touch /var/lock/subsys/jboss
         ;;
   stop)
         echo -n "Stopping jboss daemon: "
         killproc jboss
         RETVAL=$?
         echo
         [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/jboss
         ;;
   restart)
         echo -n "Restarting jboss daemon: "
  $0 stop
  sleep 2
  $0 start
         ;;
```

```
esac
```

As you see this need another script that must be placed into JBOSS_HOME/bin. It looks like this:

Listing 2-3, JBoss "go" shell script

```
################################
### Contents of go.sh
###
### (Make sure #!/bin/sh is the first line of the file and the file
### has execute permissions for root.)
################################

#!/bin/sh
#
# go.sh
# Shell script to start and stop integrated Tomcat/jBoss

export JBOSS_HOME=/usr/local/jboss
export JAVA_HOME=/usr/java

JAVACMD=$JAVA_HOME/bin/java

# Minimal jar file to get JBoss started.
CLASSPATH=$CLASSPATH:$JBOSS_HOME/bin/run.jar

# Add the tools.jar file so that Tomcat can find the Java compiler.
CLASSPATH="$CLASSPATH:$JAVA_HOME/lib/tools.jar"

if [ "$1" = "start" ] ; then
   shift
   $JAVACMD $JBOSS_OPTS -classpath $CLASSPATH org.jboss.Main –c tomcat > /dev/null 2>&1 &
   echo $! > /var/run/jboss.pid

elif [ "$1" = "stop" ] ; then
   shift
   kill -15 `cat /var/run/jboss.pid`
   rm -rf /var/run/jboss.pid

elif [ "$1" = "run" ] ; then
   shift
   $JAVACMD $JBOSS_OPTS -classpath $CLASSPATH org.jboss.Main –c tomcat "$@"

else
   echo "Usage:"
   echo "jboss (start|run|stop)"
   echo "        start - start jboss in the background"
   echo "        run   - start jboss in the foreground"
   echo "        stop  - stop jboss"
   exit 0
fi
```

Before you go any further test your installation by calling the startup script in the /etc/init.d by entering this at the prompt "./jboss start" and "./jboss stop" and check if JBoss is started and also stopped respectively.

Now you have only to create two symbolic links in the run-level 3 "/etc/init.d/rc3.d" and 5 "/etc/init.d/rc5.d" directories. The first link is the start link and the second is the kill link whereas the first letter indicates start (S) or kill (K). The next two digits indicate the order they are started or stopped (the lower the number the later it gets called) and when digits are equal then the order is not defined for this number. The rest is a string indicating the service being started. JBoss should be started late and stopped early. Therefore an example would be "S01JBoss" and "K23JBoss". Create these two symbolic links at the prompt with "ln -s ../jboss S01JBoss" and "ln -s ../jboss K23JBoss". Note that in command "ln" the target comes before the name of the link.

Now restart your Unix server and check if started properly.

3. Sample Project

How to write a JBoss Project

Skeleton of a JBoss Project

On of the bigger hurdle for a novice to JBoss application server is how to start a project, to know which components you have to provide and how to wrap it into a time saving project environment to reduce overhead at very beginning.

Here we will discuss how a project should be structured. It is not the only way and most likely you will come up with a better solution for your needs but this chapter will help you jump start your project. Whenever you are familiar with the setup of a project please skip this chapter and jump to the next one.

Note: we will not discuss a version control tool (like CVS) because this is out of scope of this book. More for information and help about this topic please have a look a **www.cvs.org**.

A J2EE project contains for different parts:

- Build system
- EJBs / Resources
- Clients (Web, Java, etc.)
- Tests

You will find all these components in JBoss as well and if it is working for JBoss it will also work for you inclusive the testing. Now let us have a closer look at them.

Build System

A build system allows you to rebuild your system after you made changes in the code or added new components. In general we have 4 ways to do so:

- Call the compiler etc. by hand on the command line
- Use a script file containing the list of commands to build your system
- Use the "make" tool
- Use the "ant" tool

For Java projects the time you save increased down the list. The first item is not even suitable for a simple test because you will compile and run several times until it is working you will spend too much time anyhow. The script file will always recompile all source files etc. and therefore repetitive use will slow you down. The good old "make" tool is suitable when the most of your components are not Java or you have to use it because of some company policies. The drawback is that it calls the Java compiler for each file and therefore it is pretty slow especially when you recreate the project complete. "make" is also pretty difficult to us.

All this was the reason for David Duncan Davidson (see Jakarta) to write a simple build tool based on XML. It was used to build Jakarta's Tomcat web server and became famous as "ant". Currently it is one of or when not the best build tool available. After spending some time to learn it you will never build a project without it.

"ant" provides many tasks to help you (compiling, archiving, copy with filtering, run JavaDoc, run other programs etc.) but it also allows you to create your own tasks which we will see later with XDoclet. On of the weaknesses is that you have difficulty to apply conditional compiling but maybe this will be added in the future.

For now we want to go with "ant" because it is sexy, Java-based, open-source and you can use it for free. Be aware that JBoss build system "buildmagic" is ant-based but for multiple projects within the JBoss project.

Enterprise Java Beans and J2EE Resources

In J2EE EJBs and the J2EE Resources is the core of any type of applications. The EJBs represents either business objects and their logic or a more advanced persistence data object. The J2EE Resources enable EJBs or other components like Servlets to access persistent data stores, connection to other (sometimes legacy) systems or communicate with others.

The EJBs are integrated in a mostly vendor neutral set of APIs like JMS, JAAS, JavaMail, JCA etc. and their implementation represents the J2EE resources. This means that an EJB can be coded in a vendor neutral way to run on different J2EE servers and J2EE resource implementation.

The EJB container of an application server also provides some services to the EJB like transactions, security or persistence which the programmer / deployer can choose to use or not. When we choose to use the container service we speak of "Container Managed" (CM) otherwise of "Bean Managed" (BM). The container managed service is normally not as powerful as the bean managed service but is defined in the deployment descriptor which makes it much more flexible to adjust to another environment. For example the container managed persistence allows the programmer to specify a list of persistent attributes which the deployer can map at deployment to the table attributes in the target database. This means that the same EJB with CMP can work on different tables and / or different databases without any changes in the EJB.

Some of the classes and deployment descriptors contain redundant definitions. Writing this files can take much time to create and to maintain and is pretty error prone. There are some code generation tools available to help you. Some are based on database structures, is taking XML input and other are code based. One of the free tools is XDoclet, which uses JavaDoc to define additional information and to generate the additional files. This means that the information is kept centralized in the EJB class. So no need to manage home and remote interface classes, deployment descriptors, primary keys etc. and they can be regenerated when necessary.

XDoclet normally only needs one file, the EJB implementation, and generates all the other necessary files to deploy an EJB. The generated files are:

- EJB and vendor (JBoss, WebLogic, WebSphere, Orion) specific deployment descriptors
- Home and Remote Interface
- Primary Key Class for Entity Beans
- Bulk Data Object (also know as Value Object)
- EJB Wrapper classes

If necessary you can stop the generation of any of these files, you can change the output generated and you can add additional code / description as predefined merge points.

Web and Other Clients

The best EJBs and resources are worthless if there is no client using it. In J2EE we have two types of client, the web client is a composition of servlets and JSPs, which are in the end servlets as well, and other clients running outside of the application server. The separation is important because web clients can take advantage of running inside the application server like accessing the “java:” namespace inside the JNDI server or using local method calls on EJBs etc. Clients running outside of the application server can mostly maintain a state and cache values / results compensating their disadvantages.

Tests

This topic is mostly neglected by most projects because it is tedious, boring and does not reward you. Tedious is true in most cases but it does not have to be boring and after some nights of debugging you maybe start thinking of a reward for test cases.

Due the fact that we are coding in a distributed environment bugs are also distributed and are hard to find through all the layers. Most of the time you need more time to find a bug than to fix it. When the components are tested than you are sure that all the test cases works and all the bugs are found on a lower level. If you found a new bug you can add the according test case to the list of test cases and improve the quality of your tests. But the most important fact is that you can run your test cases every time you made a change and be sure that you did not bring in a new bug.

There are two different types of test. A test where the code is inspected and based on the code a test plan is created is called white-box tests. When you take the code and just call it and check the output then it is called black-box tests.

White box test tools are much easier to use because it frees you from creating the test cases but it is up to the tool to define test cases, which is not suitable for testing specifications. It also reports any problem even this is not a problem for you. So you could spend more than checking the output than writing the test cases.

Black box testing requires you to write the test cases but you can test specifications and ignore problems when it is not one from your view. Thus you can save time on the long haul because you only get the problems listed you are interested in. But is it necessary that the test cases are designed and written well to ensure high quality. A good test tool is jUnit, which is an open-source project and used by many other open-source projects.

Template Project

The basic idea of this template project is to give you a starting point how a project can be setup and being ready to use JBoss in no time. It is also the basic for all the examples in this manual. Thus all the examples look similar and can be rebuild every time you want to play with them. The template only uses open-source tools which you can download for free and can produce all the necessary components for a J2EE application like EJBs, MBeans, web applications, Java clients etc.

The template uses the following tools, which you have to download before you can use it:

- Ant 1.4.1 or higher: http://jakarta.apache.org/ant
- XDoclet 1.1.2 or higher: http://www.sf.net/projects/xdoclet
- JBoss 3.0 or higher: http://www.jboss.org

Before going any further it is helpful to setup an environment variable pointing to the "ant" home directory like "ANT_HOME" because you have to start the build file without a script. The "ant" script can be found in "ANT_HOME/bin/" directory and is name "ant".

In order to create a new project take the template and copy it to your destination, rename it and then adjust the ".ant.properties" file:

- jboss.home: has to point to the root directory of your JBoss 3.0 installation
- xdoclet.home: has to point to the root directory of your XDoclet installation
- servlet-lib.path: when you have a web application then uncomment it and adjust it to point to the servlet archive file
- adjust the other settings if necessary

Whenever you need to adjust properties in the templates "build.xml" file please overwrite them in the ".ant.properties" instead. Just before we can start coding let us have a look at the directory structure. At the beginning there is only the "/src" directory which contains:

- "/etc": contains additional files
    - "/bin": script files to run the Java client. The build file will later copy the file to the "/build/bin" directory and replace "jboss.home" and "java.home" according to its setting (see .ant.properties). Thus you can run it later without adjusting any directories.
    - "/meta-inf": contains not generated deployment descriptors and manifest files
- "/main": contains the Java source except resource files and JSP pages
    - "/client": contains the client Java source files
    - "/ejb": contains the EJBs source files
    - "/servlet": contains servlet source files
- "/resources": contains resource files for XDoclet

- “/web”: contains the JSP pages for your web application

After the run the build system for the first time you will also have a “/build” directory which contains:

- “/bin”: contains the final script files to run a Java client if available
- “/classes”: compiled classes (both yours and generated Java classes)
- “/deploy”: contains the files which will be deployed automatically to your JBoss 3.0 distribution
- “/generate”: contains all the Java files which are generated by XDoclet. You can use them to see what XDoclet generated and to find problems if the generated source is not correct
- “/META-INF”: contains all the generated deployment descriptors by XDoclet.
- “/war”: to be defined

How to Code in the Template

The template comes already with a template application EJBs, web application, Java client and you can take them and adjust or you can create your own project but you should follow the directory structure and use XDoclet to generate the redundant files.

Most likely you want to add you own package structure and that is not a problem because under “/main/client”, “/main/ejb” or “/main/servlet” you can use any directory structure you like but must not add client code to another directory than “/main/client” and the same applies for EJBs and servlets.

<u>Create EJBs</u>

Write regular EJB implementation but for Entity Beans with CMP only create abstract getter and setter methods for the table attributes

Write the XDoclet class level tags like this example

```
 * @ejb:bean name="test/Manager"
 *           display-name="Manager working on projects to support clients"
 *           type="CMP"
 *           jndi-name="ejb/test/Manager"
 * @ejb:env-entry name="SequenceName"
 *                value="Manager"
 * @ejb:ejb-ref ejb-name="test/SequenceGenerator"
 * @ejb:transaction type="Required"
 * @ejb:data-object extends="test.interfaces.AbstractData"
 *                  setdata="false"
 * @ejb:finder signature="java.util.Collection findAll()"
 * @ejb:finder signature="test.interfaces.Manager findByName( java.lang.String pSurname,
java.lang.String pLastName )"
 * @jboss:finder-query name="findByName"
 *                     query="First_Name = {0} AND Last_Name = {1}"
 * @jboss:table-name table-name="Manager"
 * @jboss:create-table create="true"
 * @jboss:remove-table remove="true"
```

Write the XDoclet method level tags indicating "create", "remote", "local" and CMP attributes methods like a remote interface method

```
   /**
   * Store the data within the provided data object into this bean.
   *
   * @param pManager The Value Object containing the Manager values
   *
   * @ejb:interface-method view-type="remote"
   **/
   public void setValueObject( ManagerData pManager )
      throws
         InvalidValueException
   {
```

Finally write additional classes or XML / Code snippet to merged into the XDoclet input like this class used as super class of all generated Bulk Data objects in the example

```
package test.interfaces;

import java.io.Serializable;

/**
 * Base Data Container for all other Value Objects
 * @author Andreas Schaefer
 **/
public abstract class AbstractData
   implements Cloneable, Serializable
{
   /**
    * Returns a copy of itself. Is necessary because this
    * method is protected within java.lang.Object.
    *
    * @return Copy of this instance
    **/
   public Object clone()
   {
      try
      {
         return super.clone();
      }
      catch( CloneNotSupportedException cnse )
      {
         // This never happens
         return null;
      }
   }
}
```

Coding the clients

Coding the client is pretty simple in the first place. The client only has to look up the Home-Interface of the EJB it wants talk to, create or find an instance of this EJB through the Home-Interface. Now methods on the EJB can be invoked and so on to perform the client's tasks. A simple client look like this:

```
package test.client;

import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import test.interfaces.TestSession;
import test.interfaces.TestSessionHome;

public class TestClient {
   public static void main(String[] args){
      try {
         InitialContext lContext = new InitialContext();
         TestSessionHome lHome = (TestSessionHome) lContext.lookup( "ejb/test/TestSession"
);
         TestSession lSession = lHome.create();
         // Get a new Id of the Test Entity
         int lId = lSession.getNewEntityId();
         System.out.println( "New Entity Id is: " + lId );
         lSession.remove();
      } catch( Exception e ){
         e.printStackTrace();
      }
   }
}
```

How to Run the Template

Now we are ready to let this simple test compile, deploy and run. Three steps are necessary to do so:

- Ensure that your JBoss instance is running (which "jboss.home" property in the file ".ant.properties" points to)

- Go to the root directory of you project and start "ant" by starting the "ant" script in the "bin" directory of your ant installation. Note that you have to be in the root directory of you project to do so.

- Go to the "/build/bin" directory of your project (will be created during the build process) and start either "run-client.bat" or "run-client.sh".

The Test-Client will display how it calls the client and finally if not exception is thrown show the Id of the next TestEntity Bean:

```
$ run-client.sh
/cygdrive/c/java/jdk1.3.1/jre/bin/java -classpath <…> test.client.TestClient
New Entity Id is: 3
```

The console you use to start JBoss will display that the beans are deployed and show you when a bean is called. The output on the console can vary according to the logging settings (see "log4j.properties" file).

4. Naming

How to find your Objects in JBoss by Scott Stark

JBoss provides an implementation of the 1.2.1 Java Naming and Directory Interface (JNDI) in the JBossNS module. Configuration of the JBossNS service is done through the attributes of the org.jboss.naming.NamingService MBean. The configurable attributes for the NamingService are as follows:

- **Port**: The jnp protocol listening port for the NamingService. If not specified default is 1099, the same as the RMI registry default port.
- **RmiPort**: The RMI port on which the RMI Naming implementation will be exported. If not specified the default is 0 which means use any available port.
- **BindAddress**: the specific address the NamingService listens on. This can be used on a multi-homed host for a java.net.ServerSocket that will only accept connect requests on one of its addresses.
- **Backlog**: The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused.
- **ClientSocketFactory**: An optional custom java.rmi.server.RMIClientSocketFactory implementation class name. If not specified the default RMIClientSocketFactory is used.
- **ServerSocketFactory**: An optional custom java.rmi.server.RMIServerSocketFactory implementation class name. If not specified the default RMIServerSocketFactory is used.

JNPServerSocketFactory, An optional custom javax.net.ServerSocketFactory implementation class name. This is the factory for the ServerSocket used to bootstrap the download of the JBossNS Naming interface. If not specified the javax.net.ServerSocketFactory.getDefault() method value is used.

Since JNDI is a core J2EE service is its MBeans are configured server/default/conf/jboss-service.xml configuration file. The default configuration for the NamingService only specifies the bootstrap listening port value to be 1099. The service configuration fragment is given in Listing 4-1.

Listing 4-1, The default configuration for the JNDI service

```
<mbean code="org.jboss.naming.NamingService"
  name="jboss:service=Naming">
  <attribute name="Port">1099</attribute>
</mbean>
```

JNDI Client Configuration

When you use the JNDI API within the same VM that the JBoss server is running do not have to perform any special configuration to create a JNDI InitialContext. You simply create an InitialContext without any arguments. When you use the JBoss naming service to remote JBoss server you need to specify the appropriate InitialContext environment either by passing a Hashtable containing the desired properties or by having a jndi.properties file available on the classpath. The properties required for the InitialContext to work with the JBossNS JNDI provider are as follows:

- **java.naming.factory.initial (or Context.INITIAL_CONTEXT_FACTORY)**, The name of the environment property for specifying the initial context factory to use. This must be the org.jnp.interfaces.NamingContextFactory class for JBossNS.
- **java.naming.provider.url (or Context.PROVIDER_URL)**, The name of the environment property for specifying the location of the JBossNS service provider the client will use. The NamingContextFactory class uses this information to know which JBossNS server to connect to. The value of the property should be a URL string. For JBossNS the URL format is jnp://host:port/[jndi_path]. The jnp: portion of the URL is the protocol and refers to the socket/RMI based protocol used by JBossNS. The jndi_path portion of the URL is an option JNDI name relative to the root context, for example, "apps" or "apps/tmp". Everything but the host component is optional. The following examples are equivalent because the default port value is 1099:
  - jnp://www.jboss.org:1099/
  - www.jboss.org:1099
  - www.jboss.org

- **java.naming.factory.url.pkgs (or Context.URL_PKG_PREFIXES)**, The name of the environment property for specifying the list of package prefixes to use when loading in URL context factories. For JBossNS this must be org.jboss.naming:org.jnp.interfaces. This property is essential for locating the jnp: and java: URL context factories bundled with the JBossNS provider.

- **jnp.socketFactory**, The fully qualified class name of the javax.net.SocketFactory implementation to use to create the bootstrap socket. The default value is org.jnp.interfaces.TimedSocketFactory. The TimedSocketFactory is a simple SocketFactory implementation that supports the specification of a connection and read timeout. These two properties are specified by:

  - **jnp.timeout**, The connection timeout in milliseconds. The default value is 0 which means the connection will block until the VM TCP/IP layer times out.

  - **jnp.sotimeout**, The connected socket read timeout in milliseconds. The default value is 0 which means reads will block. This is the value passed to the Socket.setSoTimeout on the newly connected socket.

A sample jndi.properties file for connecting to a JBoss server running on appserver.dot.com on port 9901 is showing in Listing 4-2.

Listing 4-2, A sample jndi.properties file for connecting to a remote JBoss server

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://appserver.dot.com:9901/
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
# The jnp protocol socket factory class
jnp.socketFactory=org.jnp.interfaces.TimedSocketFactory
# The TimedSocketFactory connection timeout in milliseconds(0 == blocking)
jnp.timeout=0
# The TimedSocketFactory read timeout in milliseconds(0 == blocking)
jnp.sotimeout=0
```

In order for a client to connect to a JBossNS server, the client must include the jndi.properties file along with the jnp-client.jar and jnet.jar if you are using a JDK release prior to 1.4.

Customizing ejb-jar.xml and web.xml ENC Elements

A number of the J2EE enterprise naming context (ENC) elements require deployment environment configuration, or may support deployment environment configuration. In this section you will see how the ENC elements may be customized using the jboss.xml and jboss-web.xml descriptors. Listing 4-3 illustrates an ejb-jar.xml descriptor using the various

ENC elements while Listing 4-4 provides shows the corrresponding jboss.xml descriptor which defines deployment environment mappings.

Listing 4-3, A sample ejb-jar.xml descriptor illustrating the ENC elements

```
<ejb-jar>
    <display-name>ENC Tests</display-name>
    <enterprise-beans>
        <session>
            <description>A session bean on looks up stuff in the ENC</description>
#1          <ejb-name>ENCBean</ejb-name>
            <ejb-class>org.jboss.test.naming.ejb.TestENCBean</ejb-class>
            <home>org.jboss.test.naming.interfaces.TestENCHome</home>
            <remote>org.jboss.test.naming.interfaces.TestENC</remote>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>

#2          <env-entry>
                <description>A boolean flag</description>
                <env-entry-name>hasFullENC</env-entry-name>
                <env-entry-type>java.lang.Boolean</env-entry-type>
                <env-entry-value>true</env-entry-value>
            </env-entry>

#3          <ejb-ref>
                <description>An external EJB reference</description>
                <ejb-ref-name>ejb/RemoteBean</ejb-ref-name>
                <ejb-ref-type>Session</ejb-ref-type>
                <home>org.jboss.test.naming.interfaces.TestENCHome2</home>
                <remote>org.jboss.test.naming.interfaces.TestENC</remote>
            </ejb-ref>

#4          <resource-ref>
                <description>The default DS</description>
                <res-ref-name>jdbc/DefaultDS</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
            </resource-ref>
#5          <resource-ref>
                <description>Default Mail</description>
                <res-ref-name>mail/DefaultMail</res-ref-name>
                <res-type>javax.mail.Session</res-type>
                <res-auth>Container</res-auth>
            </resource-ref>
#6          <resource-ref>
                <description>Default QueueFactory</description>
                <res-ref-name>jms/QueFactory</res-ref-name>
                <res-type>javax.jms.QueueConnectionFactory</res-type>
                <res-auth>Container</res-auth>
            </resource-ref>
#7          <resource-ref>
                <description>The JBoss Web Site HomePage</description>
                <res-ref-name>url/JBossHomePage</res-ref-name>
                <res-type>java.net.URL</res-type>
                <res-auth>Container</res-auth>
            </resource-ref>

#8          <resource-env-ref>
                <description>A test of the resource-env-ref tag</description>
                <resource-env-ref-name>res/aQueue</resource-env-ref-name>
                <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
            </resource-env-ref>
        </session>

    </enterprise-beans>
</ejb-jar>
```

Listing 4-4, The corresponding sample jboss.xml descriptor to the ejb-jar.xml descriptor

```
<jboss>
    <enterprise-beans>
        <session>
#1          <ejb-name>ENCBean</ejb-name>
            <jndi-name>enc/ENCBeanHome</jndi-name>
#2          <!-- No env-entry mapping required -->
#3          <ejb-ref>
                <ejb-ref-name>ejb/RemoteBean</ejb-ref-name>
                <jndi-name>jnp://banshee:1099/ENCTests/ejbs/RemoteENCBean</jndi-name>
            </ejb-ref>

#4          <resource-ref>
                <res-ref-name>jdbc/DefaultDS</res-ref-name>
                <jndi-name>java:/DefaultDS</jndi-name>
            </resource-ref>
#5          <resource-ref>
                <res-ref-name>mail/DefaultMail</res-ref-name>
                <resource-name>DefaultMail</resource-name>
            </resource-ref>
#6          <resource-ref>
                <res-ref-name>jms/QueFactory</res-ref-name>
                <jndi-name>ConnectionFactory</jndi-name>
            </resource-ref>
#7          <resource-ref>
                <res-ref-name>url/JBossHomePage</res-ref-name>
                <res-url>http://www.jboss.org/</res-url>
            </resource-ref>

#8          <resource-env-ref>
                <resource-env-ref-name>res/aQueue</resource-env-ref-name>
                <jndi-name>queue/testQueue</jndi-name>
            </resource-env-ref>
        </session>
    </enterprise-beans>
</jboss>
```

A description of the function the jboss.xml descriptor plays in each ENC element mapping follows.

1. The default value for the location of an EJB home is simply the ejb-name element value. To specify a difference binding you provide a jndi-name element for the bean. Here the ENCBean home interface is specified as being bound under the JNDI name “enc/ENCBeanHome”.

2. An env-entry value is completely described by the ejb-jar.xml descriptor and so there is no corresponding jboss.xml descriptor element.

3. An ejb-ref to an EJB that is not located in the deployment unit of the referencing EJB must specify the JNDI name for the external EJB home interface. Here the home interface is defined to be located at “jnp://banshee:1099/ENCTests/ejbs/RemoteENCBean”.

4. The JDBC DataSource resource-ref must be mapped to the deployment location of the resource factory using the jndi-name element in the jboss.xml descriptor. Here the deployment location is defined to be “java:/DefaultDS”.

5. The JavaMail Session resource-ref must be mapped to the deployment location of the resource factory using the jndi-name element in the jboss.xml descriptor. Here the deployment location is defined to be “DefaultMail”.

6. The JMS QueueConnectionFactory resource-ref must be mapped to the deployment location of the resource factory using the jndi-name element in the jboss.xml descriptor. Here the deployment location is defined to be “ConnectionFactory”.

7. The URL resource-ref must have the URL string specified using the res-url element in the jboss.xml descriptor. Here the URL string is defined to be “http://www.jboss.org/”.

8. A resource-env-ref mapped to the deployment location of the resource using the jndi-name element in the jboss.xml descriptor. Here the deployment location of the JMS Queue is defined to be “queue/testQueue”.

5. Clustering

High Availability and Load-Balancing Services

JBoss Clustering is the final piece of the puzzle that makes JBoss a true Enterprise-Class application server. With it's fail-over, load-balancing, and distributed deployment features, JBoss Clustering provides the means to develop large scalable robust J2EE applications.

Features

The following features are available in JBoss Clsutering.

- Automatic discovery. Nodes in a cluster find each other with no additional configuration.
- Cluster-wide replicated JNDI tree
- Fail-over and load-balancing for JNDI, RMI, and all EJB types.
- Stateful Session Bean state replication
- HttpSession state replication for Jetty and Tomcat
- Farming. Distributed deployment of EJBs and JBoss services.

Getting Started

JBoss 3.0 comes with three different ready-to-use server configurations: minimal, default and all. Clustering is only enabled in this last configuration. To make sure, look in JBoss' deploy directory of the "all" configuration for *cluster-service.xml*. Setting up your EJBs to be cluster-enabled is as simple as setting a clustering flag in *jboss.xml*.

jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>
 <enterprise-beans>
```

```
  <session>
    <ejb-name>MyEJB</ejb-name>
    <jndi-name>MyEJB</jndi-name>
    <clustered>true</clustered>
  </session>
  <entity>
    <ejb-name>MyEntity</ejb-name>
    <jndi-name>MyEntity</jndi-name>
    <clustered>true</clustered>
  </session>
</jboss>
```

That's it! You can startup JBoss on different machines and they will automatically figure out that they are in a cluster. All beans flagged as clustered will automatically have fail-over, load-balancing, and replication abilities.

Cluster-wide JNDI

You can connect to and use the cluster-wide JNDI tree simple by leaving the provider URL undefined. JBoss will use IP multicast to discover clustered JNDI.

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
```

Farming

With JBoss clustering you can hot-deploy across the whole cluster just by plopping your EAR, WAR, or JAR into the deploy directory of one clustered JBoss instance. Hot-deploying on one machine will cause that component to be hot-deployed on all instances within the cluster.

Farming is not enabled by default, so you'll have to set it up yourself. Simply create the XML file shown below and copy it to the JBoss deploy directory *$JBOSS_HOME/server/all/deploy*.

farm-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ===================================================================== -->
<!--                                                                       -->
<!--  Sample Farming Service Configuration                                   -->
<!--                                                                       -->
```

```
<!-- ==================================================================== -->

<server>

   <classpath codebase="lib" archives="jbossha.jar"/>


   <!-- ==================================================================== -->
   <!-- Cluster Partition: defines cluster                                   -->
   <!-- ==================================================================== -->

   <!--
        Farm Deploy Directory and Scanner Name must not be set here because this
        are the default values but when you want to change them do it here.
   -->
   <mbean code="org.jboss.ha.framework.server.FarmMemberService"
          name="jboss:service=FarmMember,partition=DefaultPartition" >
     <depends>jboss:service=DefaultPartition</depends>
     <attribute name="PartitionName">DefaultPartition</attribute>
     <attribute name="FarmDeployDirectory">./farm</attribute>
     <attribute
name="ScannerName">jboss.deployment:type=DeploymentScanner,flavor=URL</attribute>
   </mbean>

</server>
```

After deploying *farm-service.xml* you are ready to rumble.

Trouble Shooting

- Make sure your network switch does not block the multicast IP ranges
- Make sure you have multicast enabled on your box. Here's some help for Linux: http://www.tldp.org/HOWTO/Multicast-HOWTO.html
- We have had problems running a clustered node with Win2K machines running VMWare 3.x. If you have VMWare installed on your machine, disable the VMWare Virtual Ethernet Adapters in the Device Manager
- RedHat Linux, by default, installs a firewall that prevents IP multicast packets from being distributed. Make sure you don't have this option installed or disable it.
- On Linux, you may have to add a route for multicast packets. The following command creates the route for multicast:

```
$ route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
```

- In farming with 3.0.0 FINAL, if you shut down one instance of JBoss gracefully, any farmed deployments on that node will undeploy across the *entire* cluster. Do a kill –9. *This has been fixed in 3.0.1 and later versions.*

If all else fails…

If all else fails then you must use a non-multicast communication stack for JavaGroups. Modify the "PartitionProperties" attribute to have the following JavaGroups communication stack. For TCPPING, put your own hosts and ports (host[port]) in for the *initial_hosts* parameter. DO NOT put a given node's own name in this list. So, each machine's config may have to be different. Check out the JavaGroups documentation at their website for more configuration information: http://www.javagroups.com/.

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition"
         name="jboss:service=DefaultPartition">
  <mbean-ref-list name="SynchronizedMBeans">
    <mbean-ref-list-element>jboss:service=HASessionState</mbean-ref-list-element>
    <mbean-ref-list-element>jboss:service=HAJNDI</mbean-ref-list-element>
  </mbean-ref-list>
  <attribute name="PartitionProperties">
TCP(start_port=7800):TCPPING(initial_hosts=frodo[7800],gandalf[7800];port_range=5;ti
meout=3000;num_initial_members=3;up_thread=true;down_thread=true):VERIFY_SUSPECT(tim
eout=1500;down_thread=false;up_thread=false):pbcast.STABLE(desired_avg_gossip=20000;
down_thread=false;up_thread=false):pbcast.NAKACK(down_thread=true;up_thread=true;gc_
lag=100;retransmit_timeout=3000):pbcast.GMS(join_timeout=5000;join_retry_timeout=200
0;shun=false;print_local_addr=false;down_thread=true;up_thread=true)
  </attribute>
  </mbean>
```

Advanced Configurations

The creators of JBoss Clustering have written detailed documentation that is available for purchase on the JBoss web site. This document goes into the detailed design of JBoss's clustering features and also describes how you can fine-tune your cluster-enabled applications. It is a must read for anybody wanting to do serious clustering work.

6. Write and Deploy J2EE Applications

How to write and deploy EJB and Web applications by Andreas Schaefer

This chapter gives you an introduction on how to write J2EE applications for JBoss but does not discuss how to write EJBs, Servlets or JSPs etc. in general. There are some good books available explaining how to write EJBs and Web applications. Because the Template project from chapter 3 is used here the code snippets are mostly from XDoclet generated files and you will find these files in the "build" directory of the Template project.

EJBs and Web components like Servlets, JSPs etc. are the heart of the server-side J2EE application. They have all in common to reside in their container that is part of the J2EE application server. For convenience sake they can be deployed as Java archives with different extensions. These achieves contain a vendor neutral deployment descriptor (or DD for short), 0 to n vendor specific DDs, other archives if available, compiled Java classes, JSPs, HTML Pages and other resource files. In JBoss they can be hot deployed with a drop in the deployment archive and replaces any existing applications with the same name.

Entity Beans

Entity Beans represent unit of data that is in a relational database a record. It encapsulates the database specifics as well as the integration into transactions and the timing when the data is synchronized with the persistence store.

As a rule of thumb Entity Beans should not contain any business logic but only data integrity and persistence logic.

Bean Managed Persistence (BMP)

Bean Managed Persistence means that the Bean Developer coded the logic how the data is made persistent on the target persistence store like a RDBMS. This also means that most likely the EJB is specific for a certain persistence store even thought the application is written vendor neutral.

The code how to store or retrieve data from the persistence store are added to the ejbStore() and ejbLoad() methods. It is the client's responsibility to retrieve the correct resource adapter, datasource or other data access layer to access the persistence layer. Whenever you retrieve these objects through JNDI it is recommended to specify their name as environment variable in the EJB DD (ejb-jar.xml). The XDoclet tag would look like this:

```
* @ejb:env-entry name="DataSourceName"
 *                value="java:/PostgreSQL"
```

And is then used here:

```
private DataSource getDataSource()
{
   try {
      Context lContext = new InitialContext();

      String lDataSourceName = (String) lContext.lookup(
         "java:comp/env/DataSourceName"
      );
      return (DataSource) lContext.lookup( lDataSourceName );
   }
   catch ( NamingException ne ) {
      throw new EJBException( "Naming lookup failure: " + ne.getMessage() );
   }
}
```

This enables the deployer to adjust there is another JNDI name used for the data source.

The only vendor specific settings for BMPs is to set them as read-only bean and then the ejbStore() method is never called. Of course, you can still save or change data in another method but this is considered erroneous coding.

Depending on your commit options and other settings it is possible that the Entity Beans container invokes the ejbStore() method too often. To avoid these calls there is an option method "isModified()" and if it returns false the ejbStore() is not called. XDoclet normally generates the method and the call to set it dirty or clear already. In your Bean class you have the chance to specify the methods "makeDirty()" and "makeClean()" as abstract methods and then you can set this flag in other, not generated methods.

```
/**
 * Mark the Entity as changed that needs to be saved
 **/
protected abstract void makeDirty();
/**
 * Mark the Entity as synchronized with the DB and does
 * not need to be saved
 **/
protected abstract void makeClean();
```

As example in the TestBMPEntityBean the “ejbLoad()” method sets the values with the setter methods but in turn the set the dirty flag to true which obviously false. With the method call of “makeClean()” it is fixed.

```
public void ejbLoad()
{
   DataSource lDataSource = getDataSource();
   Connection lConnection = null;
   PreparedStatement lStatement = null;
   try {
      lConnection = lDataSource.getConnection();
      lStatement = lConnection.prepareStatement(
         "SELECT Id, First_Name, Last_Name FROM TestEntity WHERE id = ?"
      );
      int lId = ( (TestBMPEntityPK) mContext.getPrimaryKey() ).id;
      lStatement.setInt( 1, lId );
      ResultSet lResult = lStatement.executeQuery();
      lResult.next();
      setId( lResult.getInt( 1 ) );
      setFirstName( lResult.getString( 2 ) );
      setLastName( lResult.getString( 3 ) );
      // Because this method used the attribute setter method
      // the bean is automatically marked as dirty. Therefore
      // reverse this here because it is obviosly not true
      makeClean();
   }
   catch ( SQLException se ) {
      throw new EJBException( "Could not read record from DB: " + se.getMessage() );
   }
   finally {
      if( lStatement != null ) {
         try {
            lStatement.close();
         }
         catch( Exception e ) {}
      }
      if( lConnection != null ) {
         try {
            lConnection.close();
         }
         catch( Exception e ) {}
      }
   }
}
```

Container Managed Persistence (CMP)

This type of Entity Beans uses an Object – Relational Mapper (O/R Mapper) to store data on persistence store and recreated them. Most O/R Mappers supports various databases and allows a bean developer to create Entity Beans that are not only server vendor neutral but also database vendor neutral.

Because the CMP implementation is so important and became somewhat complex this book explain it in its own chapter 7, so please have a look there.

Session Beans

Session Beans represent business objects and offer its functionality. As example a bank's customer Enity Bean delivers information like id, address, number of accounts but the customer Session Bean offers services like to create a new type of account for this customer, transfer money between accounts of this customer etc.

Session Beans contain the business logic and uses Entity Beans to retrieve and store data persistently.

A Session Bean contains an "ejbCreate()" method where no parameters are allowed. This is due to the fact that even when a client calls the "create()" method on the Home interface it does not mean automatically that a Session Bean instance is created. It is up to the application server to decide when to create a new instance and when to pick one from a pool of inactive Session Beans. The next method is the "ejbRemove()" which is called by the application server when the Session Bean is destroyed and not automatically when the client calls "remove()" on this instance because the application server can put this instance in the instance pool. Finally there are the methods "ejbPassivate()" and "ejbActive" which are called just before the bean is made persistent from the pool or recreate and put into the pool. This allows the bean developer to close or reopen used resources.

Because the application server can use advanced features to manage and pool Session Beans the client should hold on them as brief as possible. This means you should create a Session Bean just before you use it and should remove it afterwards. It is bad programming practice to keep a reference to a Session Bean for a unnecessary long time like for example as a member variable in the client program. The length of this period is naturally much longer for Stateful Session Beans than for Stateless Session Beans.

But it is ok to keep the Home interface as a member variable with the exception that you have to be prepared that in a hot redeployement this Home interface is not available anymore and a new copy has to be looked up on the JNDI server.

Stateless Session Beans

Stateless Session Beans (SLSB for short) do, as the name implies, not maintain any state between calls. This means that one client could call a SLSB and afterwards another client could call the same or another method on the same SLSB. Note that this does not mean that a SLSB cannot store informations like various Home interfaces of Enity Beans it is using but they must not be assigned to a particular call or client.

The advantage of a SLSB is that the application server can easily balance the number of SLSBs because the assignment to a client is limited to the call. So even there are thousands of concurrent users they may have only a few SLSB available because the time elapsed of a call is short compared to the time waiting for the next call.

The disadvantage of a SLSB is that the client has to maintain states and transactions between calls. So for local client like Web applications maintaining states can be difficult and for remote clients handle transactions over several calls can be costly.

Speaking of transactions in a Bean Managed Transaction (BMT for short) SLSB any incoming transaction is suspended and a created transaction within the call must be committed or rolled back before the method returns.

For a SLSB it is not uncomment to use one or more Entity Beans. During development the JNDI names of the EJBs maybe different to what they are in the production environment because a JNDI name is already used or there is a different naming convention. That is why the developer should use EJB references instead of the “public” JNDI name of the EJB. This allows the deployer to change the “public” JNDI name without breaking the application except for a remote client. The same applies also to resources like mail or datasources. In XDoclet another EJB is referenced by:

```
* @ejb:ejb-ref ejb-name="test/TestEntity"
 *              ref-name="mytest/TestEntity"
 *
```

Later on it can be used this way:

```
Context lContext = new InitialContext();
TestEntityHome lHome = (TestEntityHome) PortableRemoteObject.narrow(
   lContext.lookup(
      "java:comp/env/ejb/mytest/TestEntity"
   ),
   TestEntityHome.class
);
```

The EJB DD contains then:

```
<ejb-ref >
   <ejb-ref-name>ejb/mytest/TestEntity</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <home>test.interfaces.TestEntityHome</home>
   <remote>test.interfaces.TestEntity</remote>
   <ejb-link>test/TestEntity</ejb-link>
</ejb-ref>
```

Now the deployer can adjust the JNDI names of the EJBs in the JBoss specific DD (jboss.xml) without changing anything else. The issue becomes a little bit trickier when an EJB is referenced in another application but the principle stays the same.

Stateful Session Beans

Stateful Session Beans (SFSB for short) is either in a ready pool (unused) or assigned to a particular client. Thus a client call to a SFSB will always reach the same SFSB with the exceptions when a SFSB times out or when the client removes and creates a new SFSB.

In the transaction example the Teller Session Bean is s SLSB meaning that when information like to which bank the teller belongs has to be delivered by the client on every call. With a SFSB it can contain these informations and free the client from doing so.

The disadvantage is that for every client using this Session Bean there is a SFSB copy reserved. So when you have 10,000 concurrent users could have 10,000 instances of this SFSB even when they are rarely used.

Another feature of the SFSB is that a user transaction (BMT) started in a method does not have to be finished within this method call. This allows you to have a method creating an order and then several method calls to add items to this order. At the end the client places the order and thus finishes the transaction. Important to note that the client has to know what starts and what ends a transaction because a transaction cannot be finished when not started in the first place and when a transaction is never finished it will be rolled back by the server anyway and therefore everything is lost.

Message Driven Beans

Message Driven Beans (or MDB for short) are like SLSB but with the exception that they do not have a Home interface and therefore cannot be called by a client directly. Instead they are listening for JMS messages from a particular Destination. Thus any client able to send a message to the JMS provider will invoke the "onMessage()" on the MDB. This client must not be a Java (JMS) client but it could be a mainframe computer sending a message.

To write a MDB it is necessary to add the MessageListener interface to the implements list even this looks superfluous. The idea behind is to be able to add other interfaces than the MessageListener in the future. The Template project already contains an example of a MDB that listens on the "queue/testQueue" queue.

Unfortunately XDoclet 1.1.2 does not support some advanced features of JBoss MDB settings. The DTD description for MDBs looks like:

```
<!ELEMENT message-driven (
   ejb-name , destination-jndi-name , mdb-user? , mdb-passwd? , mdb-client-id? ,
   mdb-subscription-id? , configuration-name? , security-proxy? , ejb-ref* ,
   resource-ref* , resource-env-ref*
)>
```

The "mdb-user" and "mdb-passwd" is use to create a connection with this user name – password pair, the "mdb-client-id" allows you to specify a certain client id that is associated with the created connection and “mdb-subscription-id” is the name of the durable subscription to be used.

In the “onMessage()” method you can access another EJB, a resource like a database, mail or send a copy or a new message to another destination. Please take into consideration that it is up to the application server how many instances of the same MDB it creates as it is with SLSB. Therefore processing of messages can be concurrently and out of order. So you could receive a “place order” message before “create order” message.

Web Applications

Web applications are running inside the JBoss application server. So they have access to the container reserved JNDI namespace “java:/comp” but can also refer to other EJBs deployed on the same application server. Because the JBoss web servers are running within the same JVM its servlets (at runtime JSPs are servlets as well) can access all the resources in JBoss locally.

The DTD of the JBoss specific DD “jboss-web.xml” is defined as:

```
<!ELEMENT jboss-web (
   security-domain?, context-root?, virtual-host?,
   resource-env-ref*, resource-ref* , ejb-ref*
)>
```

The “context-root” allows you to specify another document root for the standalone web application but do not overwrite a document root in the Enterprise application DD “application.xml”. The “virtual-host” allow you to specify to which virtual host this application is deployed too. Finally the “resource-ref” and “ejb-ref” allows you to specify the references to resources and EJBs deployed on this server, which then later on can be renamed by the deployer if necessary.

A Web application is typically deployed as WAR file that is nothing else than a regular Java archive (JAR) with a different extension. It can contain the Web application DD “web.xml” as well as the JBoss specific DD “jboss-web.xml”. Note that these files must be added to the “WEB-INF” directory instead of the “META-INF”. Due the fact that a Web application is a client to EJBs, if used, you have to add the Java archive containing the client EJB classes into either the “WEB-INF/lib” or “WEB-INF/classes” directory.

Enterprise Applications

Enterprise applications are just a convenience package containing EJB applications, Web applications and client EJB archives so that it can be deployed as single file. Currently there is no additional features and therefore also no JBoss specific deployment descriptor for the Enterprise applications.

Conclusion

As a skeleton project how to write and build an EJB -, Web- and Enterprise application please have a look at the template project in Chapter 3. It contains all the described parts in this chapter. But keep in mind that XDoclet can generate the Web application DD as well as the JBoss specific DD when you write Servlets.

The Template shows you that to write an application is not that difficult as long as you do simple projects. Of course, the problems are hidden in the details that come with the advanced programming.

7. Transactions

A Unit of Work in Action

An EJB container must provide some service to the applications deployed on and the user using the applications. The transaction service allows an application and / or the user of an application to specify the unit of work that is defined by four different aspects called ACID for short:

- Atomic: A transaction is performed completely or not at all meaning that any changes made during the transaction must be reversed if the transaction is not successfully completed.
- Consistency: The transaction must ensure that the system (data store) is in a consistent state after the transaction has ended. Of course the system must have been in a consistent state before the transaction was started otherwise this does not apply. Note that the application developer must still make his/her part to ensure consistency.
- Isolation: The transaction must run isolated from any other processes or transactions until transaction is finished. The two extremes are that either all other sees the changes during a transaction or that only the process in the transaction sees the changes and the other see the old data. As you maybe can imagine the isolation comes with a performance hit. Therefore in J2EE are several types of isolations defined.
- Durable: The changes in a transaction must be made persistent when a transaction finishes.

Transactions are pretty important and most people take it for given but a good implementation needs a good design from the beginning. Imagine that you want to order a flight on-line. The airline would not be very happy when you could reserve the seats but never pay for the flight. On the other hand you want to get the selected seats after you paid for the tickets. These requirements mean that the seats must be locked when they are selected until the customer pays and if not the seats must be released.

Transactions can become trickier when you need to use third party applications if internal or external. Now a transaction must include this third party application by handing over the transactions to the other party meaning we need a distributed transaction. This sounds not much more complicated but considering the fact that the connection to the third party application is lost or it does not respond in time we maybe have to determine the outcome heuristic to go on with our application. Heuristic because the application does not know where the failure was, the client could not have received the request, the request could have been processed successful or could have failed or when the third party application responded but the connection dropped beforehand.

Transactions are associated with threads. When a thread calls a component, the transaction is automatically propagated with the thread of execution. This happens for local as well as remote calls.

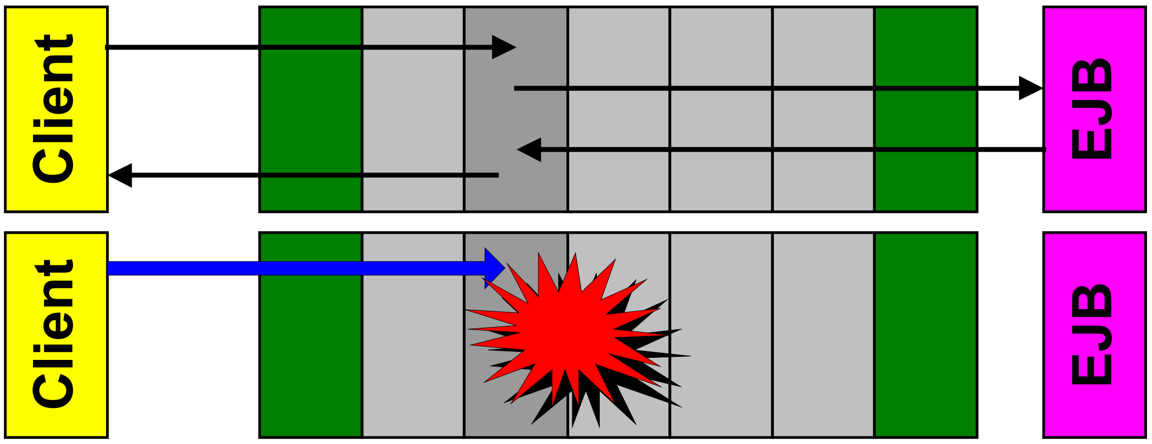
In J2EE nested transactions are not allowed. Therefore we cannot have a transaction inside another transaction. But using the javax.transaction.TransactionManager interface it is possible to suspend a transaction and then use no transaction or another transaction. This allows the application server to perform actions outside the given transaction in its own transaction or completely without one. After these steps are done the transaction is reactivated and the next steps can be performed. Please note that the TransactionManager interface is not meant to be used by components hosted by the J2EE server. Components should always use the javax.transaction.UserTransaction interface instead.

In J2EE the bean developer has two choices how to use transactions. Either he can let the container managed the transactions or he can managed transactions in the bean.

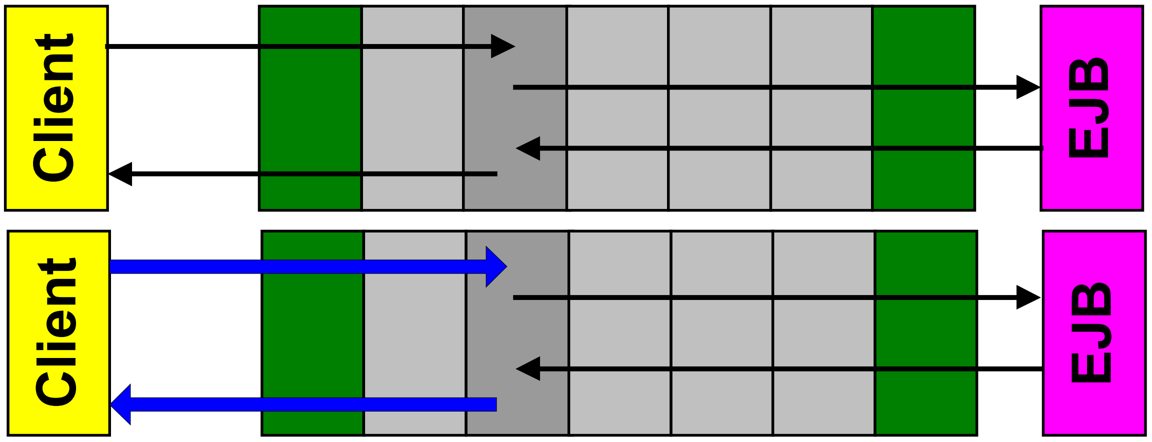
Container Managed Transaction (CMT)

In a CMT bean the container manage the transactions. How to do so is specified by the transaction attributes in the bean deployment descriptor ("trans-attribute"). The following graphics uses a black arrow for a call without a transaction, a blue arrow for a transaction coming from the caller, a red arrow for a new transaction created in the transaction interceptor that is indicated by the dark gray box. There are six different transaction attributes:

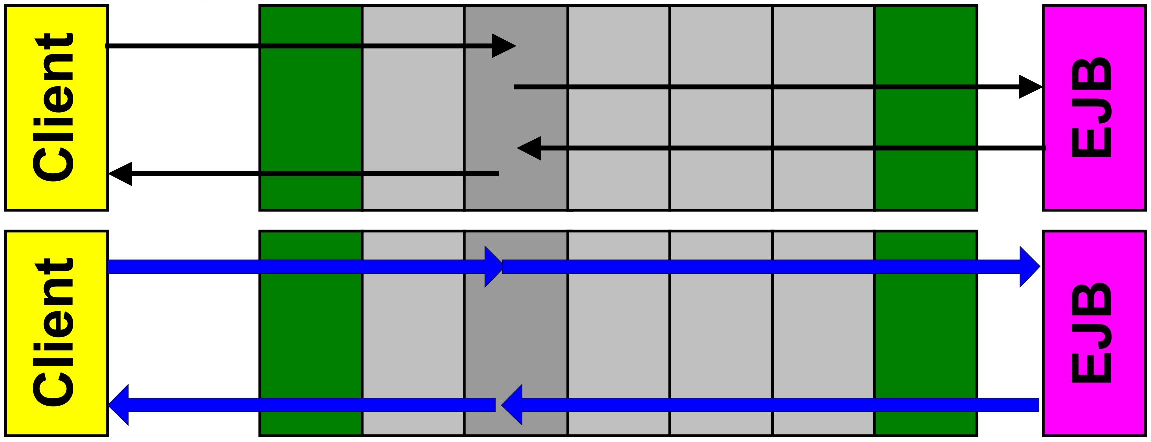
- Never: the client is not allowed to call with a transaction and if then an exception will be thrown



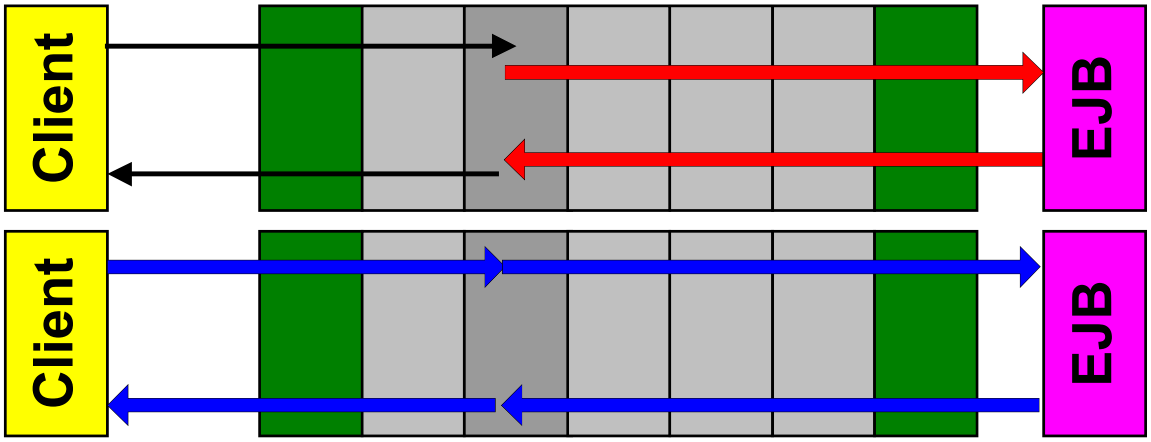
- NotSupported: any existing transaction is suspended during the call. This is useful when the bean developer knows that the bean cannot participate in a transaction because it cannot undo its changes or wants to prevent the creation of a new transaction



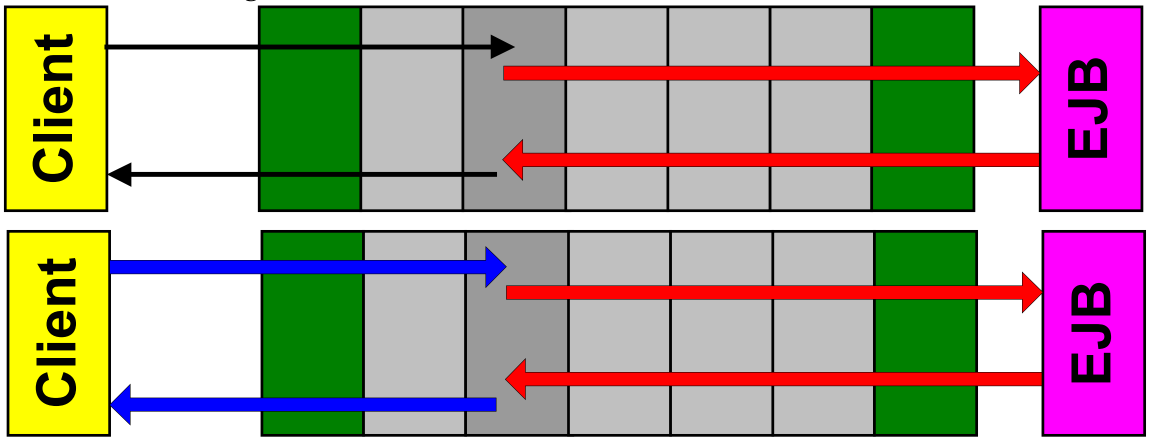
- Supported: call will be made with or without a transaction provided by the caller. This is useful when the bean developer does not care if it part of a transaction but it is not very transparent therefore this should be used with care



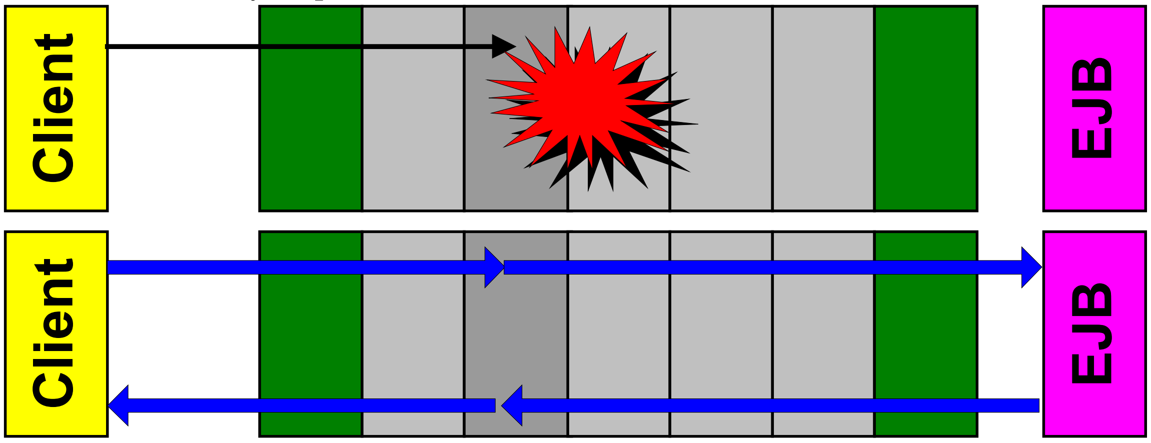
- Required: if no transaction is available one will be created otherwise the existing one will be used. A bean developer can ensure that its code runs always in a transaction like a bank account transfer



- RequiresNew: a new transaction will be created and if one already exists it will be suspended. A bean developer can ensure that its code will run in its own transaction because the changes have to be committed or rolled back at the end



- Mandatory: the client has to provide a transaction and if not an exception is thrown. A bean developer wants to ensure that the bean always is part of a transaction because it is always a part of a transaction like a bank account withdrawal



Note:

That when a new transaction is created in the transaction interceptor this transaction ends when the call returns to the transaction interceptor. If a system exception (either a java.rmi.RemoteException or a java.lang.RuntimeException) is thrown the transaction interceptor will rollback the transaction, otherwise it will commit the transaction. When a bean does not want to throw a system exception but wants the transaction rolled back it can mark the transaction for rollback using the javax.ejb.EJBContext.setRollbackOnly(). When a transaction is marked for rollback it is impossible to commit the transaction, and the only possible outcome is a rollback. From the caller s view these two ways looks the same because the transaction interceptor will throw an exception indicating that the transaction is rolled back. The caller can decide, if it is another bean, if it wants to go on with its own transaction by catching this exception; otherwise the exception will propagate out of the bean code, and the original transaction will be rolled back, too.

How to set the Transaction Attributes

The transaction attributes are set in the EJB deployment descriptor "ejb-jar.xml". You can set a transaction attribute for all methods, for a particular method name or for a particular method (specified by its parameters) whereas the more specific specification overwrites the less specific one. The part in the EJB deployment descriptor defining the transaction attributes could look like:

```
<container-transaction>
   <method >
      <ejb-name>bank/CustomerSession</ejb-name>
      <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method >
      <ejb-name>bank/CustomerSession</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>createCustomer</method-name>
   <trans-attribute>Mandatory</trans-attribute>
</container-transaction>
<container-transaction>
   <method >
      <ejb-name>bank/CustomerSession</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>createCustomer</method-name>
      <method-params>
         <method-param>java.lang.String</method-param>
         <method-param>java.lang.String</method-param>
         <method-param>float</method-param>
      </method-params>
   </method>
   <trans-attribute>RequiresNew</trans-attribute>
```

```
</container-transaction>
```

The first “container-transaction” declares that in general all methods have transaction attribute “Required”. The second declares that any methods with the name of “createCustomer” have transaction attribute “Mandatory” and the last says that the “createCustomer” method with parameter signature “(String, String,float)” has the transaction attribute “RequiresNew”.

Note:
The “container-transaction” elements in the EJB deployment descriptor are embedded in the “assembly-descriptor” which is embedded in the root “ejb-jar” element. This means that the transaction attributes are not specified in the “enterprise-beans” block as you maybe expect.

Bean Managed Transactions (BMT)

Whenever a bean needs to have more control over the transaction it has to manage the transactions on its own. The rules for the beans are:

- Entity Bean: because transactions are integrated with the Entity Beans life cycle an Entity Bean must be a CMT.
- Stateless Session Bean: because this bean cannot maintain a state between method calls, a transaction must be committed or rolled back before the method returns
- Stateful Session Bean: A transaction can spawn more than one method call. If the bean is called with a transaction, that transaction is suspended, and the bean instance own transaction (possibly none, if the bean has not started a transaction) is used while executing in the bean code. Often the caller must know which bean methods to call, and in which order, to open and close the transaction in the correct order.
- Message Driven Bean: the destination only calls one method “onMessage” and therefore any transaction must be started and finished within this method

Like for clients the BMT uses an instance of a UserTransaction, which must be obtained by the "javax.ejb.EJBContext" and not from the JNDI server because it must be managed by the container. The user transaction can be started and ended as many times as necessary, but only a single transaction may be active at any time in any BMT bean instance.

Note that for most cases a BMT can be replaced by redesigned CMTs that are easier to understand than BMTs.

Client Transactions

A client can begin and end a transaction by getting the UserTransaction from the naming server, begin and then commit or roll back the transaction at the end. In JBoss there are two different user transactions: a fast, in-VM implementation for clients executing in the server VM, and a remote implementation that works for thin remote clients.

A client can begin and end a transaction by getting the UserTransaction from the naming server, begin and then commit or roll back the transaction at the end. In JBoss there are two different user transactions: a fast, in-VM implementation and a remote implementation able to work for remote clients.

Transaction Isolation and Locking

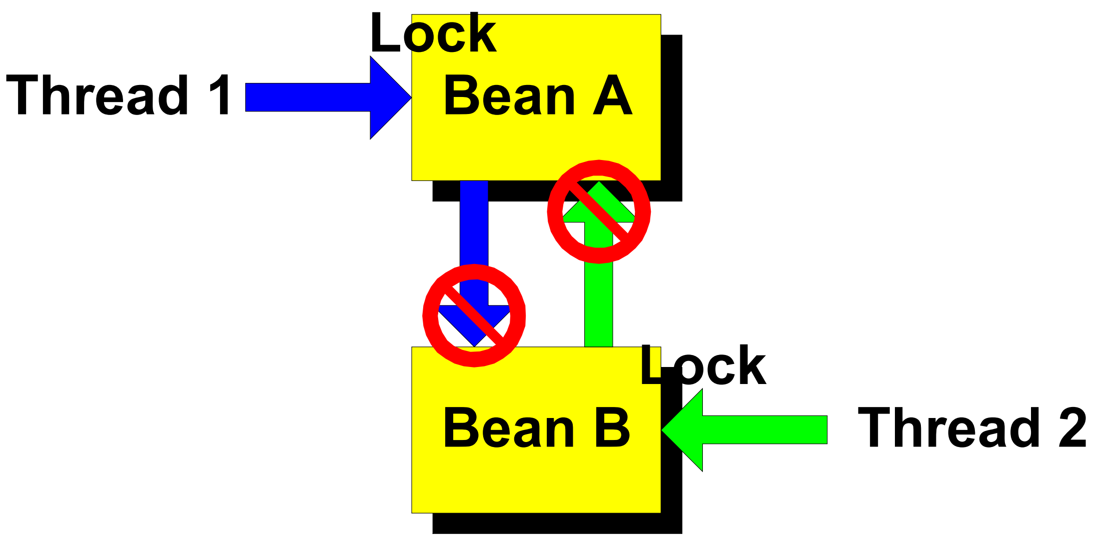
Transactions should be isolated from each other to avoid conflicts because a transaction in progress could change data that another transactions depend on. In J2EE there are four different levels of isolation:

- Read uncommitted: a transaction can read changes made by another transaction even not committed or rolled back.
- Read committed: a transaction cannot read any changed by another transaction until it is committed.
- Repeatable reads: a transaction cannot change data read by another transaction
- Serializable: a transaction has exclusive read and write access to the data it is using.

Note that even though these isolation levels are specified by the J2EE specification the implementation depends heavily on the underlying database. Thus, even on the same application server with different databases you may observe slightly different behaviors.

Deadlocks

Because of transaction and method isolation it is possible to create an application deadlock. When a thread “1” calls bean “A” and another thread “2” calls bean “B” then “1” tries to call bean “B” and “2” tries to call bean “A” we have a deadlock because both tries to call another, already locked bean and none of them can give their lock.



In JBoss 3.0 there is a deadlock detection system in place, which will throw a deadlock exception on one of the threads freeing the other thread to call the other bean. Now the caller of the interrupted thread receives the exception and can decide to retry the call, to go over and throw an exception as well.

Example

Derived from the Template the Transaction example mimics a bank where you have banks, customers, accounts and tellers. The CMP Entity Beans are hidden behind Session Beans, which contains the business logic, and should only be accessed by them.

Because in this example as well XDoclet is used the specification of the transaction attributes is very simple:

```
/**
 * The Session bean represents the customer's business interface
…
 * @ejb:transaction type="Required"
…
 */
public class CustomerSessionBean
   extends SessionSupport
{

   /**
    * @ejb:interface-method view-type="remote"
    * @ejb:transaction type="RequiresNew"
    **/
   public CustomerData createCustomer(
      String pBankId, String pName, float pInitialDeposit
   )
      throws CreateException, RemoteException
   {
```

As you see on the top to define the default transaction attribute is as simple as adding the XDoclet tag “ejb:transaction”. Below you see how you can specify a transaction attribute for a specific method (which does not exists like this in the real example).

Now to the more interesting part: the client. In this example there is only a remote Java client, which only differs from the web-client in the way that the remote client needs a user transaction for remote clients. The code looks like this:

```
InitialContext lContext = new InitialContext();

Object lObject = lContext.lookup( "ejb/bank/BankSession" );
BankSessionHome lBankHome = (BankSessionHome)
   PortableRemoteObject.narrow( lObject, BankSessionHome.class );
BankSession lBankSession = lBankHome.create();
BankData lBank = lBankSession.createBank( "Test", "1234 Test Avenue, Test City" );
lObject = lContext.lookup( "ejb/bank/TellerSession" );
TellerSessionHome lTellerHome = (TellerSessionHome)
   PortableRemoteObject.narrow( lObject, TellerSessionHome.class );
TellerSession lTeller = lTellerHome.create();
CustomerData lCustomerA = lTeller.createCustomer( lBank.getId(), "Andreas Schaefer", 100 );
AccountData lCheckingAccountA = lTeller.getAccount( lCustomerA.getId(), Constants.CHECKING
);
AccountData lSavingsAccountA = lTeller.createAccount( lCustomerA.getId(), Constants.SAVING,
250 );
lTeller.transfer( lSavingsAccountA.getId(), lCheckingAccountA.getId(), 125 );
CustomerData lCustomerB =  lTeller.createCustomer( lBank.getId(), "Marc Fleury", 500 );
AccountData lCheckingAccountB =  lTeller.getAccount( lCustomerA.getId(), Constants.CHECKING
);
UserTransaction lTransaction = (UserTransaction) lContext.lookup( "UserTransaction" );

lTransaction.begin();
try {
   lTeller.withdraw( lCheckingAccountA.getId(), 165 );
   lTeller.deposit( lCheckingAccountB.getId(), 165 );
   lTransaction.commit();
}
catch( Exception e ) {
   lTransaction.rollback();
}
```

The client first creates a bank and look up for a Teller Session Bean. Then trough the teller it creates a customer and a second account, transfers money from one account to another. For this transfer there is no user transaction necessary because the transfer method creates it own transaction (attribute is set to “Required”). Now the Teller creates another customer and retrieves a user transaction because the withdrawal on one’s customer’s account and the deposit on the other’s customer’s account is not part of a transaction. But here the client wants to have the withdrawal and deposit either done completely or not done at all. Therefore the client starts the user transaction (lTransaction.begin()), makes the withdrawal and deposit and commits the transfer afterwards (lTransaction.commit()) if no exception is thrown. If, indeed, an exception is thrown the transaction is rolled back (lTransaction.rollback()). Whenever the transaction would be marked for rollback the commit would fail and throw an exception that finally rolls back the transaction.

8. Security

Controlling J2EE Component Access by Scott Stark

JBoss provides a JAAS based security manager that supports the J2EE declarative security model defined in the EJB and servlet specifications. This chapter will introduce the security services configuration and the steps needed to secure EJBs and web applications.

Security Services Configuration

There are three MBean services that control the security layer configuration, SecurityConfig, XMLLoginConfig and JaasSecurityManagerService. They are configured in the server/<config-name>/conf/jboss-service.xml core services descriptor.

org.jboss.security.plugins.SecurityConfig

The SecurityConfig service MBean manages the active JAAS login configuration implementation. It support replacing the default JAAS configuration as well as chaining configurations together. Its sole attribute is:

- **LoginConfig**, the ObjectName string of the mbean that provides the default JAAS login configuration. This name is used to lookup the MBean which provides the javax.security.auth.login.Configuration implementation to install as the default. The named MBean must implement an operation with this signature:
- `javax.security.auth.login.Configuration getConfiguration(javax.security.auth.login.Configuration parent)`

org.jboss.security.auth.login.XMLLoginConfig

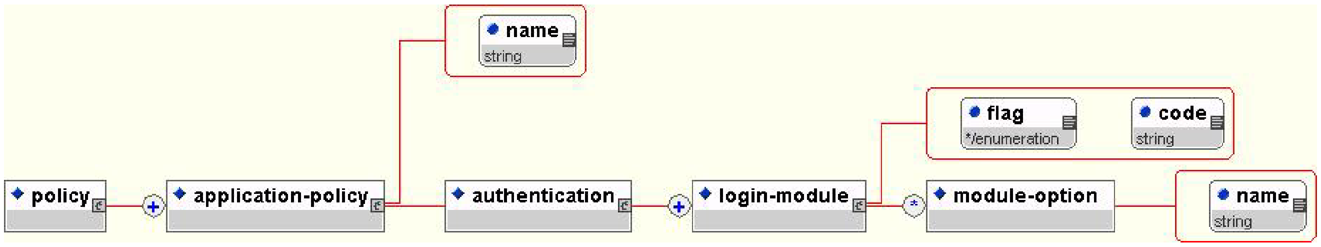
The XMLLoginConfig service MBean provides an implementation of javax.security.auth.login.Configuration that uses an XML configuration file. The configurable attributes of the XMLLoginConfig service include:

- **ConfigURL**, Set the URL of the XML login configuration file that should be loaded by this mbean on startup.

- **ConfigResource**, Set the resource name of the XML login configuration file that should be loaded by this mbean on startup. The configuration file will be loaded using the current thread context ClassLoader.getResource(String) method.

The DTD for the configuration file parsed by the XMLLoginConfig service is given in Figure 8-1.

Figure 8-1, the configuration file DTD supported by the XMLLoginConfigservice.



- A **policy/application-policy** element defines an application security domain configuration.
- The name attribute gives the name of the security domain.
- The **policy/application-policy/authentication** element defines the login module configuration stack for the application security domain.
- A **policy/application-policy/authentication/login-module** element defines a login module configuration entry.
- The flag attribute must be one of:
    - required - The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.
    - requisite - The LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).
    - sufficient - The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the LoginModule list). If it fails, authentication continues down the LoginModule list.

  - optional - The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

- The code attribute gives the fully qualifed class name of the javax.security.auth.spi.LoginModule interface implementation for the login module.
- A **policy/application-policy/authentication/login-module/module-option** element specifies a login module option name/value pair.
- The name attribute specifies the name of the login module option.
- The element value is the option value string representation.

The following listing shows a sample configuration file.

Listing 8-1, A sample login configuration for the XMLLoginConfigservice.

```
<policy>
   <application-policy name = "sample-domain">
      <authentication>
         <login-module code = "org.jboss.security.auth.spi.UsersRolesLoginModule"
             flag = "required">
             <module-option name="usersProperties">sample.users</module-option>
             <module-option name="rolesProperties">sample.roles</module-option>
         </login-module>
      </authentication>
   </application-policy>
</policy>
```

JAAS LoginModules Bundled With JBoss

JBoss comes with a number of JAAS LoginModule implementations that support commonly used security stores such as JDBC databases and LDAP servers. The most commonly used login modules are presented in the following subsections.

org.jboss.security.auth.spi.UsersRolesLoginModule

The UsersRolesLoginModule is another simple login module that supports multiple users and user roles, and is based on two Java Properties formatted text files. The username-to-password mapping file is called "users.properties" and the username-to-roles mapping file is called "roles.properties". The properties files are loaded during initialization using the initialize method thread context class loader. This means that these files can be placed into the J2EE deployment jar, the JBoss configuration directory, or any directory on the JBoss server or system classpath. The primary purpose of this login module is to easily test the security settings of multiple users and roles using properties files deployed with the application.

The users.properties file uses a "username=password" format with each user entry on a separate line as show here:

```
username1=password1
username2=password2
...
```

The roles.properties file uses as "username=role1,role2,..." format with an optional group name value. For example:

```
username1=role1,role2,...
username1.RoleGroup1=role3,role4,...
username2=role1,role3,...
```

The supported login module configuration options include the following:

- **unauthenticatedIdentity=name**, Defines the principal name that should be assigned to requests that contain no authentication information. This can be used to allow unprotected servlets to invoke methods on EJBs that do not require a specific role. Such a principal has no associated roles and so can only access either unsecured EJBs or EJB methods that are associated with the unchecked permission constraint.

- **password-stacking=useFirstPass**, When password-stacking option is set, this module first looks for a shared username and password under the property names "javax.security.auth.login.name" and "javax.security.auth.login.password" respectively in the login module shared state Map. If found these are used as the principal name and password. If not found the principal name and password are set by this login module and stored under the property names "javax.security.auth.login.name" and "javax.security.auth.login.password" respectively.

- **hashAlgorithm=string**: The name of the java.security.MessageDigest algorithm to use to hash the password. There is no default so this option must be specified to enable hashing. When hashAlgorithm is specified, the clear text password obtained from the CallbackHandler is hashed before it is passed to UsernamePasswordLoginModule.validatePassword as the inputPassword argument. The expectedPassword as stored in the users.properties file must be comparably hashed.

- **hashEncoding=base64|hex**: The string format for the hashed pass and must be either "base64" or "hex". Base64 is the default.

- **hashCharset=string**: The encoding used to convert the clear text password to a byte array. The platform default encoding is the default.

- **usersProperties=string**: (2.4.5+) The name of the properties resource containing the username to password mappings. This defaults to users.properties.
- **rolesProperties=string**: (2.4.5+) The name of the properties resource containing the username to roles mappings. This defaults to roles.properties.

A sample login configuration entry that assigned unauthenticated users the principal name "nobody" and contains based64 encoded, MD5 hashes of the passwords in a "usersb64.properties" file is:

```
<application-policy name = "testUsersRoles">
   <authentication>
      <login-module code = "org.jboss.security.auth.spi.UsersRolesLoginModule"
          flag = "required">
          <module-option name="usersProperties">usersb64.properties</module-option>
          <module-option name="hashAlgorithm">MD5</module-option>
          <module-option name="hashEncoding">base64</module-option>
          <module-option name="unauthenticatedIdentity">nobody</module-option>
      </login-module>
   </authentication>
</application-policy>
```

org.jboss.security.auth.spi.LdapLoginModule

The LdapLoginModule is a LoginModule implementation that authenticates against an LDAP server using JNDI login using the login module configuration options. You would use the LdapLoginModule if your username and credential information are store in an LDAP server that is accessible using a JNDI LDAP provider.

The LDAP connectivity information is provided as configuration options that are passed through to the environment object used to create JNDI initial context. The standard LDAP JNDI properties used include the following:

- **java.naming.factory.initial**, The classname of the InitialContextFactory implementation. This defaults to the Sun LDAP provider implementation com.sun.jndi.ldap.LdapCtxFactory.
- **java.naming.provider.url**, The ldap URL for the LDAP server
- **java.naming.security.authentication**, The security level to use. This defaults to "simple".
- **java.naming.security.protocol**, The transport protocol to use for secure access, such as, ssl

- **java.naming.security.principal**, The principal for authenticating the caller to the service. This is built from other properties as described below.
- **java.naming.security.credentials**, The value of the property depends on the authentication scheme. For example, it could be a hashed password, clear-text password, key, certificate, and so on.

The supported login module configuration options include the following:

- **principalDNPrefix=string**, A prefix to add to the username to form the user distinguished name. See principalDNSuffix for more info.
- **principalDNSuffix=string**, A suffix to add to the username when forming the user distiguished name. This is useful if you prompt a user for a username and you don't want the user to have to enter the fully distinguished name. Using this property and principalDNSuffix the userDN will be formed as:

  ```
  String userDN = principalDNPrefix + username + principalDNSuffix;
  ```
- **useObjectCredential=true|false**, Indicates that the credential should be obtained as an opaque Object using the org.jboss.security.auth.callback.ObjectCallback type of Callback rather than as a char[] password using a JAAS PasswordCallback. This allows for passing non-char[] credential information to the LDAP server.
- rolesCtxDN=string, The distinguished name to the context to search for user roles.
- **roleAttributeID=string**, The name of the attribute that contains the user roles. If not specified this defaults to "roles".
- **uidAttributeID=string**, The name of the attribute in the object containing the user roles that corresponds to the userid. This is used to locate the user roles. If not specified this defaults to "uid".
- **matchOnUserDN=true|false**, A flag indicating if the search for user roles should match on the user's fully distinguished name. If false, just the username is used as the match value against the uidAttributeName attribute. If true, the full userDN is used as the match value.
- **unauthenticatedIdentity=string**, The principal name that should be assigned to requests that contain no authentication information. This behavior is inherited from the UsernamePasswordLoginModule superclass.
- **password-stacking=useFirstPass**, When the password-stacking option is set, this module first looks for a shared username and password under the property names "javax.security.auth.login.name" and "javax.security.auth.login.password"

respectively in the login module shared state Map. If found these are used as the principal name and password. If not found the principal name and password are set by this login module and stored under the property names "javax.security.auth.login.name" and "javax.security.auth.login.password" respectively.

- **hashAlgorithm=string**: The name of the java.security.MessageDigest algorithm to use to hash the password. There is no default so this option must be specified to enable hashing. When hashAlgorithm is specified, the clear text password obtained from the CallbackHandler is hashed before it is passed to UsernamePasswordLoginModule.validatePassword as the inputPassword argument. The expectedPassword as stored in the LDAP server must be comparably hashed.

- **hashEncoding=base64|hex**: The string format for the hashed pass and must be either "base64" or "hex". Base64 is the default.

- **hashCharset=string**: The encoding used to convert the clear text password to a byte array. The platform default encoding is the default.

The authentication of a user is performed by connecting to the LDAP server based on the login module configuration options. Connecting to the LDAP server is done by creating an InitialLdapContext with an environment composed of the LDAP JNDI properties described previously in this section. The Context.SECURITY_PRINCIPAL is set to the distinguished name of the user as obtained by the callback handler in combination with the principalDNPrefix and principalDNSuffix option values, and the Context.SECURITY_CREDENTIALS property is either set to the String password or the Object credential depending on the useObjectCredential option.

Once authentication has succeeded by virtue of being able to create an InitialLdapContext instance, the user's roles are queried by performing a search on the rolesCtxDN location with search attributes set to the roleAttributeName and uidAttributeName option values. The roles names are obtaining by invoking the toString method on the role attributes in the search result set.

A sample login configuration entry is:

```
<application-policy name = "testLdap">
  <authentication>
    <login-module code = "org.jboss.security.auth.spi.LdapLoginModule"
      flag = "required">
      <module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</module-option>
      <module-option
name="java.naming.provider.url">ldap://ldaphost.jboss.org:1389/</module-option>
      <module-option name="java.naming.security.authentication">simple</module-option>
      <module-option name="principalDNPrefix">uid=</module-option>
      <module-option name="principalDNSuffix">,ou=People,o=jboss.org</module-option>
      <module-option name="uidAttributeID">userid</module-option>
      <module-option name="roleAttributeID">roleName</module-option>
      <module-option name="rolesCtxDN">cn=JBossSX Tests,ou=Roles,o=jboss.org</module-
option>
    </login-module>
  </authentication>
</application-policy>
```

To help you understand all of the options of the LdapLoginModule, consider the sample LDAP server data shown in Figure 8-2. This figure corresponds to the testLdap login configuration just shown.

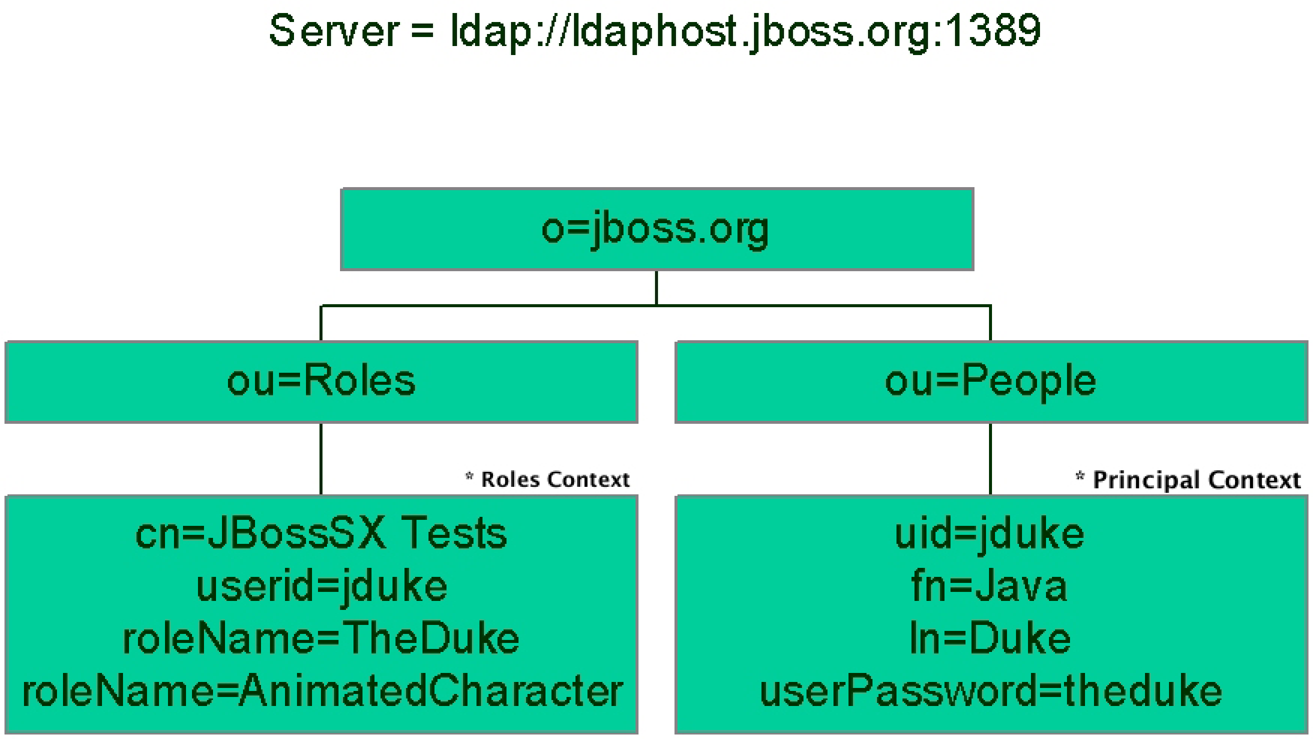


Figure 8-2, An LDAP server configuration compatible with the testLdap sample configuration.

org.jboss.security.auth.spi.DatabaseServerLoginModule

The DatabaseServerLoginModule is a JDBC based login module that supports authentication and role mapping. You would use this login module if you have your username, password and role information in a JDBC accessible database. The DatabaseServerLoginModule is based on two logical tables:

```
Table Principals(PrincipalID text, Password text)
Table Roles(PrincipalID text, Role text, RoleGroup text)
```

The Principals table associates the user PrincipalID with the valid password and the Roles table associates the user PrincipalID with its role sets. The roles used for user permissions must be contained in rows with a RoleGroup column value of Roles. The tables are logical in that you can specify the SQL query that the login module uses. All that is required is that the java.sql.ResultSet has the same logical structure as the Principals and Roles tables described previously. The actual names of the tables and columns are not relevant as the results are accessed based on the column index. To clarify this notion, consider a database with two tables, Principals and Roles, as already declared. The following statements build the tables to contain a PrincipalID 'java' with a Password of 'echoman' in the Principals table, a PrincipalID 'java' with a role named 'Echo' in the 'Roles' RoleGroup in the Roles table, and a PrincipalID 'java' with a role named 'caller_java' in the 'CallerPrincipal' RoleGroup in the Roles table:

```
INSERT INTO Principals VALUES('java', 'echoman')
INSERT INTO Roles VALUES('java', 'Echo', 'Roles')
INSERT INTO Roles VALUES('java', 'caller_java', 'CallerPrincipal')
```

The supported login module configuration options include the following:

- **dsJndiName**: The JNDI name for the DataSource of the database containing the logical "Principals" and "Roles" tables. If not specified this defaults to "java:/DefaultDS".

- **principalsQuery**: The prepared statement query equivalent to: "select Password from Principals where PrincipalID=?". If not specified this is the exact prepared statement that will be used.

- **rolesQuery**: The prepared statement query equivalent to: "select Role, RoleGroup from Roles where PrincipalID=?". If not specified this is the exact prepared statement that will be used.

- **unauthenticatedIdentity=string**, The principal name that should be assigned to requests that contain no authentication information.

- **password-stacking=useFirstPass**, When password-stacking option is set, this module first looks for a shared username and password under the property names "javax.security.auth.login.name" and "javax.security.auth.login.password" respectively in the login module shared state Map. If found these are used as the principal name and password. If not found the principal name and password are set by this login module and stored under the property names "javax.security.auth.login.name" and "javax.security.auth.login.password" respectively.

- **hashAlgorithm=string**: The name of the java.security.MessageDigest algorithm to use to hash the password. There is no default so this option must be specified to enable hashing. When hashAlgorithm is specified, the clear text password obtained from the CallbackHandler is hashed before it is passed to UsernamePasswordLoginModule.validatePassword as the inputPassword argument. The expectedPassword as obtained from the database must be comparably hashed.
- **hashEncoding=base64|hex**: The string format for the hashed pass and must be either "base64" or "hex". Base64 is the default.
- **hashCharset=string**: The encoding used to convert the clear text password to a byte array. The platform default encoding is the default

As an example DatabaseServerLoginModule configuration, consider a custom table schema like the following:

```
CREATE TABLE Users(username VARCHAR(64) PRIMARY KEY, passwd VARCHAR(64))
CREATE TABLE UserRoles(username VARCHAR(64), userRoles VARCHAR(32))
```

The corresponding DatabaseServerLoginModule configuration would be:

```
<application-policy name = "testDB">
  <authentication>
    <login-module code = "org.jboss.security.auth.spi.DatabaseServerLoginModule"
        flag = "required">
      <module-option name="dsJndiName">java:/MyDatabaseDS</module-option>
      <module-option name="principalsQuery">select passwd from Users username where
username=?</module-option>
      <module-option name="rolesQuery">select userRoles, 'Roles' from UserRoles where
username=?</module-option>
    </login-module>
  </authentication>
</application-policy>
```

org.jboss.security.ClientLoginModule

The ClientLoginModule is an implementation of LoginModule for use by JBoss clients for the establishment of the caller identity and credentials. This simply sets the org.jboss.security.SecurityAssociation.principal to the value of the NameCallback filled in by the CallbackHandler, and the org.jboss.security.SecurityAssociation.credential to the value of the PasswordCallback filled in by the CallbackHandler. This is the only supported mechanism for a client to establish the current thread's caller. Both stand-alone client applications and server environments, acting as JBoss EJB clients where the security environment has not been configured to use JBossSX transparently, need to use the ClientLoginModule.

Note that this login module does not perform any authentication. It merely copies the login information provided to it into the JBoss server EJB invocation layer for subsequent authentication on the server. If you need to perform client-side authentication of users you would need to configure login modules in addition to the ClientLoginModule.

The supported login module configuration options include the following:

- **multi-threaded=true|false**, When the multi-threaded option is set to true, each login thread has its own principal and credential storage. This is useful in client environments where multiple user identities are active in separate threads. When true, each separate thread must perform its own login. When set to false the login identity and credentials are global variables that apply to all threads in the VM. The default for this option is false.

- **password-stacking=useFirstPass**, When password-stacking option is set, this module first looks for a shared username and password using "javax.security.auth.login.name" and "javax.security.auth.login.password" respectively in the login module shared state Map. This allows a module configured prior to this one to establish a valid username and password that should be passed to JBoss. You would use this option if you want to perform client-side authentication of clients using some other login module such as the LdapLoginModule.

A sample login configuration for ClientLoginModule is the default configuration entry found in the JBoss distribution client/auth.conf file. The configuration is:

```
other {
    // Put your login modules that work without JBoss here

    // jBoss LoginModule
    org.jboss.security.ClientLoginModule required;

    // Put your login modules that need JBoss here
};
```

org.jboss.security.plugins.JaasSecurityManagerService

The JaasSecurityManagerService manages the configuration of the security service. Its responsibilities include the externalization of the scurity manager implementation class, authentication caches and JNDI namespace management. The configurable attributes of the JaasSecurityManagerService include:

- **SecurityManagerClassName**, Set the name of the class that provides the security manager implementation. This requires a class that implements the org.jboss.security.AuthenticationManager and org.jboss.security.RealmMapping interfaces. The default value is the JAAS based security manager org.jboss.security.plugins.JaasSecurityManager.

- **SecurityProxyFactoryClassName**, Set the name of the class that provides the org.jboss.security.SecurityProxyFactory implementation. Security proxies provide support for advanced security beyond that supported by the J2EE declarative security model. The default value is org.jboss.security.SubjectSecurityProxyFactory.

- **AuthenticationCacheJndiName**, Set the location of the security credential cache policy. This is first treated as a javax.naming.spi.ObjectFactory location that is capable of returning org.jboss.util.CachePolicy instances on a per security domain basis by appending a "/security-domain-name" string to this name when looking up the CachePolicy for a domain. If this fails then the location is treated as a single CachePolicy for all security domains.

- **DefaultCacheTimeout**, Set the default timed cache policy timeout in seconds. This is the period over which authentication credentials will be cached. The default value is 1800 seconds. This has no affect if the AuthenticationCacheJndiName has been changed from the default value.

- **DefaultCacheResolution**, Set the default timed cache policy resolution in seconds. This is the frequency at which cached credentials are checked for expiration. The default value is 60 seconds. This has no affect if the AuthenticationCacheJndiName has been changed from the default value.

Default Security Service Configuration

The default configuration for the security services configuration is given in Listing 8-2 for reference.

Listing 8-2, The server/default/conf/jboss-service.xml descriptor security services configuration.

```
<mbean code="org.jboss.security.plugins.SecurityConfig"
  name="jboss.security:name=SecurityConfig">
  <attribute name="LoginConfig">jboss.security:name=XMLLoginConfig</attribute>
</mbean>
<mbean code="org.jboss.security.auth.login.XMLLoginConfig"
  name="jboss.security:name=XMLLoginConfig">
  <attribute name="ConfigResource">login-config.xml</attribute>
</mbean>

<!-- JAAS security manager and realm mapping -->
<mbean code="org.jboss.security.plugins.JaasSecurityManagerService"
  name="jboss.security:name=JaasSecurityManager">
  <attribute name="SecurityManagerClassName">
    org.jboss.security.plugins.JaasSecurityManager
  </attribute>
</mbean>
```

Securing Your Application

To enable security in your EJB and Web applications, you must declare the EJB method permissions and web content constraints using the standard ejb-jar.xml and web.xml descriptors respectively. In addition, you must specify the security domain which JBoss will use to perform the authentication and authorization checks. This is done using the security-domain element in the jboss.xml EJB descriptor and jboss-web.xml Web application descriptor. Listing 8-3 gives examples of an ejb-jar.xml descriptor that makes use of the standard declarative security elements while Listing 8-4 gives an example jboss.xml descriptor that specifies the required security domain information. The security related elements are highlighted in ***bold-italic*** and numbered for discussion.

Listing 8-3, A sample ejb-jar.xml descriptor illustrating the use of the security elements.

```
<ejb-jar>
    <display-name>SecurityTests</display-name>
    <enterprise-beans>
        <session>
            <description>A secured trival echo session bean</description>
            <ejb-name>StatelessSession</ejb-name>
            <home>org.jboss.test.security.interfaces.StatelessSessionHome</home>
            <remote>org.jboss.test.security.interfaces.StatelessSession</remote>
            <ejb-class>org.jboss.test.security.ejb.StatelessSessionBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>

      <session>
         <description>A secured trival echo session bean that calls
         getCallerPrincpal in ejbCreate</description>
         <ejb-name>SecureCreateSession</ejb-name>
         <home>org.jboss.test.security.interfaces.StatelessSessionHome</home>
         <remote>org.jboss.test.security.interfaces.StatelessSession</remote>
         <ejb-class>org.jboss.test.security.ejb.StatelessSessionBean4</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
      </session>

        <session>
            <description>A secured trival echo session bean</description>
            <ejb-name>org/jboss/test/security/ejb/StatelessSession_test</ejb-name>
            <home>org.jboss.test.security.interfaces.StatelessSessionHome</home>
            <remote>org.jboss.test.security.interfaces.StatelessSession</remote>
            <ejb-class>org.jboss.test.security.ejb.StatelessSessionBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
#1          <!-- Use the 'EchoCaller' role name in the bean code to test role linking
            with use of isCallerInRole().
            -->
            <security-role-ref>
                <role-name>EchoCaller</role-name>
                <role-link>Echo</role-link>
            </security-role-ref>
        </session>

        <session>
            <description>A secured trival echo session bean that uses Entity</description>
            <ejb-name>StatelessSession2</ejb-name>
            <home>org.jboss.test.security.interfaces.StatelessSessionHome</home>
            <remote>org.jboss.test.security.interfaces.StatelessSession</remote>
            <ejb-class>org.jboss.test.security.ejb.StatelessSessionBean2</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
            <ejb-ref>
                <ejb-ref-name>ejb/Entity</ejb-ref-name>
                <ejb-ref-type>Entity</ejb-ref-type>
                <home>org.jboss.test.security.interfaces.EntityHome</home>
                <remote>org.jboss.test.security.interfaces.Entity</remote>
                <ejb-link>Entity</ejb-link>
            </ejb-ref>
            <ejb-ref>
                <ejb-ref-name>ejb/Session</ejb-ref-name>
                <ejb-ref-type>Session</ejb-ref-type>
                <home>org.jboss.test.security.interfaces.StatelessSessionHome</home>
                <remote>org.jboss.test.security.interfaces.StatelessSession</remote>
                <ejb-link>StatelessSession</ejb-link>
            </ejb-ref>
        </session>

     <session>
         <description>An unsecured trival echo session bean</description>
         <ejb-name>UnsecureStatelessSession</ejb-name>
         <home>org.jboss.test.security.interfaces.StatelessSessionHome</home>
         <remote>org.jboss.test.security.interfaces.StatelessSession</remote>
         <ejb-class>org.jboss.test.security.ejb.StatelessSessionBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
     </session>

      <entity>
         <description>A trival echo entity bean</description>
         <ejb-name>Entity</ejb-name>
         <home>org.jboss.test.security.interfaces.EntityHome</home>
         <remote>org.jboss.test.security.interfaces.Entity</remote>
         <ejb-class>org.jboss.test.security.ejb.EntityBeanImpl</ejb-class>
         <persistence-type>Bean</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
         <reentrant>False</reentrant>
      </entity>
      <entity>
         <description>A trival echo entity bean that should only be
             accessible via other beans</description>
```

```
         <ejb-name>PrivateEntity</ejb-name>
         <home>org.jboss.test.security.interfaces.EntityHome</home>
         <remote>org.jboss.test.security.interfaces.Entity</remote>
         <ejb-class>org.jboss.test.security.ejb.EntityBeanImpl</ejb-class>
         <persistence-type>Bean</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
         <reentrant>False</reentrant>
         <security-role-ref>
             <role-name>InternalRole</role-name>
             <role-link>InternalRole</role-link>
         </security-role-ref>
      </entity>

      <message-driven>
         <description>A trival echo entity bean</description>
         <ejb-name>RunAsMDB</ejb-name>
         <ejb-class>org.jboss.test.security.ejb.RunAsMDB</ejb-class>
         <transaction-type>Container</transaction-type>
         <message-driven-destination>
            <destination-type>javax.jms.Queue</destination-type>
            <subscription-durability>NonDurable</subscription-durability>
         </message-driven-destination>
         <ejb-ref>
             <ejb-ref-name>ejb/Entity</ejb-ref-name>
             <ejb-ref-type>Entity</ejb-ref-type>
             <home>org.jboss.test.security.interfaces.EntityHome</home>
             <remote>org.jboss.test.security.interfaces.Entity</remote>
             <ejb-link>PrivateEntity</ejb-link>
         </ejb-ref>
#2       <security-identity>
            <description>Use a role that is not assigned to any users to
               access restricted server side functionallity</description>
            <run-as>
               <role-name>InternalRole</role-name>
            </run-as>
         </security-identity>
      </message-driven>
   </enterprise-beans>

    <assembly-descriptor>
#3      <security-role>
            <description>The role required to invoke the echo method</description>
            <role-name>Echo</role-name>
        </security-role>
        <security-role>
            <description>The role used to prevent access to the PrivateEntity
                bean from external users.
            </description>
            <role-name>InternalRole</role-name>
        </security-role>

        <!-- The methods the Echo role can access -->
        <method-permission>
#4          <role-name>Echo</role-name>
```

```
            <method>
                <ejb-name>StatelessSession</ejb-name>
                <method-name>create</method-name>
            </method>
            <method>
                <ejb-name>StatelessSession</ejb-name>
                <method-name>remove</method-name>
            </method>
            <method>
                <ejb-name>StatelessSession</ejb-name>
                <method-name>echo</method-name>
            </method>
            <method>
                <ejb-name>StatelessSession</ejb-name>
                <method-name>npeError</method-name>
            </method>

            <method>
               <ejb-name>org/jboss/test/security/ejb/StatelessSession_test</ejb-name>
                <method-name>*</method-name>
            </method>

            <method>
                <ejb-name>SecureCreateSession</ejb-name>
                <method-name>*</method-name>
            </method>

            <method>
                <ejb-name>StatelessSession2</ejb-name>
                <method-name>*</method-name>
            </method>

            <method>
                <ejb-name>Entity</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>

        <!-- The methods the InternalRole role can access -->
        <method-permission>
#5          <role-name>InternalRole</role-name>

            <method>
                <ejb-name>PrivateEntity</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>

        <!-- Anyone can access the unchecked() method of the StatelessSession bean -->
        <method-permission>
#6          <unchecked/>
            <method>
                <ejb-name>StatelessSession</ejb-name>
                <method-name>unchecked</method-name>
            </method>
        </method-permission>

        <!-- No one can access the excluded() method of the
            StatelessSession and StatelessSession2 beans -->
#7      <exclude-list>
            <description>A method that no one can access in this deployment</description>
            <method>
                <ejb-name>StatelessSession</ejb-name>
                <method-name>excluded</method-name>
            </method>
            <method>
                <ejb-name>StatelessSession2</ejb-name>
                <method-name>excluded</method-name>
            </method>
        </exclude-list>

    </assembly-descriptor>
</ejb-jar>
```

Listing 8-4, The jboss.xml descriptor that specifies the security domains for the Listing 8-3 ejb-jar.xml descriptor.

```
<jboss>
    <container-configurations>
        <!-- StatelessSession beans are secure by default -->
        <container-configuration>
            <container-name>Standard Stateless SessionBean</container-name>
#8          <security-domain>java:/jaas/spec-test</security-domain>
        </container-configuration>

        <!-- Entity beans are secure by default -->
        <container-configuration>
            <container-name>Standard BMP EntityBean</container-name>
#9          <security-domain>java:/jaas/spec-test</security-domain>
        </container-configuration>

        <!-- A stateless session config that is not secured -->
        <container-configuration extends="Standard Stateless SessionBean">
            <container-name>Unsecure Stateless SessionBean</container-name>
#10         <security-domain/>
        </container-configuration>
    </container-configurations>

  <enterprise-beans>
      <session>
         <ejb-name>UnsecureStatelessSession</ejb-name>
#11      <container-name>Unsecure Stateless SessionBean</container-name>
      </session>
      <message-driven>
         <ejb-name>RunAsMDB</ejb-name>
         <destination-jndi-name>queue/A</destination-jndi-name>
      </message-driven>
  </enterprise-beans>
</jboss>
```

Listing 8-5, A sample jboss-web.xml descriptor illustrating specifying the security domain for a war.

```
<jboss-web>
    <security-domain>java:/jaas/spec-test</security-domain>
</jboss-web>
```

The highlighted items are:

9. A security-role-ref element declares the role name that the session bean with use in calls to the EnterpriseContext.isCallerInRole method. Here the declaration states that “EchoCaller” will be used and this name used by the bean is mapped to the application logical name “Echo”.

10. The security-identity element declares that when the message driven bean invokes methods on other beans it will do so with a role “InternalRole”. It is common to use this construct with MDBs when they need to used secured beans as MDBs have no standard way to assign a caller identity.

11. The security-role elements declare the declarative roles used by the EJBs. This will be used to map from the “EchoCaller” string to the “Echo” string when the session bean calls isCallerInRole. A principal caller will match the beans check if a role named “Echo” has been assigned. The “InternalRole” declaration is really only for documentation and portability to other application servers.

12. This is the method permissions section for the “Echo” role. Each method element declares a method of an EJB the Echo role is allowed to execute.

13. This is the method permission section for the “InternalRole” role. This is used to restrict access to the PrivateEntity entity bean to only other EJBs in this application that assume the InternalRole via a run-as declaration.

14. The unchecked element declares methods that any authenticated user may access. The unchecked element declares that no specific roles are required to execute the given methods, but callers must be authenticated users.

15. The excluded-list element declares methods that no principal is able to execute in the deployment. It is a mechanism to prevent access to methods regardless of the caller and their roles.

16. Moving to the jboss.xml descriptor, the security-domain declaration in the “Standard Stateless SessionBean” configuration is declaring that by default stateless session bean in this deployment are secured. This is because the

"Standard Stateless SessionBean" is the default configuration used for stateless session beans in the absence of a configuration override declaration. The value of the security-domain element here is defining that the JAAS login configuration named "spec-test" will be executed to authentication principals attempting to access stateless session beans. The "java:/jaas/" prefix is a naming convention that is supported by the JaasSecurityManagerService MBean to dynamically create security managers for domains.

17. Here BMP entity beans are also declared to be security by default since "Standard BMP EntityBean" is the default configuration name for BMP entity beans.

18. The "Unsecure Stateless SessionBean" configuration declaration is defining an extension of the "Standard Stateless SessionBean" configuration that overrides the security-domain to null and thus disables security for beans that use this configuration.

19. The EJB named "UnsecureStatelessSession" is declaring that the "Unsecure Stateless SessionBean" container configuration be used for its instances. Therefore, security is not used with UnsecureStatelessSessions.

9. CMP 2.0

Container Managed Persistence by Dain Sundstrom

Entity Basics

JBossCMP is a powerful persistence engine compliant with the EJB 2.0 CMP 2.0 specification. Although several new features have been added, the basic entity bean structure has not changed much in CMP 2.0. One small, yet very important, change is a CMP field is longer declared using a class field in the bean implementation class, but instead is declared with a set of abstract accessor methods. Abstract accessors are similar to JavaBean property accessors, except no implementation is given. The following listings declare the local home, local, and bean implementation class with gangsterId, name, nickName, and badness cmp-fields:

```
// Gangster Local Home Interface
public interface GangsterHome extends EJBLocalHome {
   Gangster create(Integer id, String name, String nickName) throws CreateException;
   Gangster findByPrimaryKey(Integer id) throws FinderException;
}
```

Listing 9-1, Entity Local Home Interface

```
// Gangster Local Interface
public interface Gangster extends EJBLocalObject {
   Integer getGangsterId();
   String getName();
   String getNickName();
   void setNickName(String nickName);
}
```

Listing 9-2, Entity Local Interface

```
// Gangster Implementation Class
public abstract class GangsterBean implements EntityBean {
   private EntityContext ctx;
   private Category log = Category.getInstance(getClass());

   public Integer ejbCreate(Integer id, String name, String nickName)
         throws CreateException {

      log.info("Creating Gangster " + id + " '" + nickName + "' "+ name);
```

```
        setGangsterId(id);
        setName(name);
        setNickName(nickName);
        return null;
    }

    public void ejbPostCreate(Integer id, String name, String nickName) { }

    // CMP field accessors -----------------------------------------------------
    public abstract Integer getGangsterId();
    public abstract void setGangsterId(Integer gangsterId);

    public abstract String getName();
    public abstract void setName(String name);

    public abstract String getNickName();
    public abstract void setNickName(String nickName);

    // EJB callbacks -----------------------------------------------------------
    public void setEntityContext(EntityContext context) { ctx = context; }
    public void unsetEntityContext() { ctx = null; }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() { log.info("Removing " + getName()); }
    public void ejbStore() { }
    public void ejbLoad() {}
}
```

Listing 9-3, Entity Implementation Class

The entity bean implementation class must be abstract, because the cmp field abstract accessors are abstract. Also, each cmp-field is required to have both a getter and a setter method, and each accessor method must be declared public abstract.

Entity Declaration

The declaration of an entity in the ejb-jar.xml file has not changed much in CMP 2.0. The declaration of the GangsterEJB follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC
   "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
   "http://java.sun.com/j2ee/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
   <enterprise-beans>
      <entity>
         <display-name>Gangster Entity Bean</display-name>
         <ejb-name>GangsterEJB</ejb-name>

         <local home>org.jboss.docs.cmp2.crimeportal.GangsterHome</local home>
```

```
          <local>org.jboss.docs.cmp2.crimeportal.Gangster</local>
          <ejb-class>org.jboss.docs.cmp2.crimeportal.GangsterBean</ejb-class>

          <persistence-type>Container</persistence-type>
          <prim-key-class>java.lang.Integer</prim-key-class>
          <reentrant>False</reentrant>
          <cmp-version>2.x</cmp-version>
          <abstract-schema-name>gangster</abstract-schema-name>

          <cmp-field><field-name>gangsterId</field-name></cmp-field>
          <cmp-field><field-name>name</field-name></cmp-field>
          <cmp-field><field-name>nickName</field-name></cmp-field>

          <primkey-field>gangsterId</primkey-field>
       </entity>
   </enterprise-beans>
</ejb-jar>
```

Listing 9-4, The ejb-jar.xml Entity Declaration

The most important part of the ejb-jar.xml file is the DOCTYPE declaration. When JBoss deploys an EJB jar file, the DOCTYPE of the ejb-jar.xml deployment descriptor is used to determine the version of the EJB jar. If the public identifier of the DOCTYPE is "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN", JBoss will use the JBossCMP persistence engine, otherwise the old JAWS CMP 1.1 persistence engine will be used. The new local home and local elements are equivalent to the home and remote elements. The cmp-version element is new and can be either 1.x or the default 2.x. This element was added so 1.x and 2.x entities could be mixed in the same application. The abstract-schema-name element is also new and is used to identify this entity type in EJB-QL queries.

Entity Mapping

The optional JBossCMP configuration is declared in the jbosscmp-jdbc.xml file, which is located in the META-INF directory of the ejb-jar file. This file has the same overall structure as the ejb-jar.xml file, except the root element is jbosscmp-jdbc instead of ejb-jar. The optional configuration of the GangsterEJB follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jbosscmp-jdbc PUBLIC
   "-//JBoss//DTD JBOSSCMP-JDBC 3.0//EN"
   "http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_3_0.dtd">

<jbosscmp-jdbc>
   <enterprise-beans>
          <ejb-name>GangsterEJB</ejb-name>
          <table-name>gangster</table-name>

          <cmp-field>
             <field-name>gangsterId</field-name>
```

```
                <column-name>id</column-name>
             </cmp-field>
             <cmp-field>
                <field-name>name</field-name>
                <column-name>name</column-name>
                <not-null/>
             </cmp-field>
             <cmp-field>
                <field-name>nickName</field-name>
                <column-name>nick_name</column-name>
                <jdbc-type>VARCHAR</jdbc-type>
                <sql-type>VARCHAR(64)</sql-type>
             </cmp-field>
             <cmp-field>
                <field-name>badness</field-name>
                <column-name>badness</column-name>
             </cmp-field>

             <!-- Load Groups -->

             <!-- Queries -->

         </entity>
     </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 9-5, The jbosscmp-jdbc.xml Entity Mapping

In this case the DOCTYPE declaration is optional, but will reduce configuration errors. In addition, all of the elements are optional except for ejb-name, which is used to match the configuration to an entity declared in the ejb-jar.xml file. Unless noted otherwise, the default values come from the defaults section of the jbosscmp-jdbc.xml file, which is discussed in Appendix B of the JBossCMP Documentation. A detailed description of each entity element follows:

Table 9-1, entity Tags

Tag Name	Description	Required
ejb-name	This is the name of the EJB to which this configuration applies. This element must match the ejb-name of an entity in the ejb-jar.xml file.	Yes
datasource	This is the jndi-name used to look up the datasource. All database connections used by an entity are obtained from the datasource. Having different datasources for entities is not recommended, as it vastly constrains the domain over which finders and ejbSelects can query.	No, default is java:/DefaultDS

Tag Name	Description	Required
datasource-mapping	This specifies the name of the type-mapping, which determines how Java types are mapped to SQL types, and how EJB-QL functions are mapped to database specific functions. Type-mapping is discussed in Appendix C of the JBossCMP Documentation.	No, default is Hypersonic SQL
create-table	If true, JBossCMP will attempt to create a table for the entity. When the application is deployed, JBossCMP checks if a table already exists before creating the table. If a table is found, it is logged, and the table is not created. This option is very useful during the early stages of development when the table structure changes often.	No, default is true
remove-table	If true, JBossCMP will attempt to drop the table for each entity and each relation-table mapped relationship. When the application is undeployed, JBossCMP will attempt to drop the table. This option is very useful during the early stages of development when the table structure changes often.	No, default is false
read-only	If true, the bean provider will not be allowed to change the value of any fields. This option is discussed in Chapter 3 of the JBossCMP Documentation.	No, default is false
read-time-out	This is the amount of time in milliseconds that a read on a read-only field is valid. This option is discussed in Chapter 3 of the JBossCMP Documentation.	No, default is 300
row-locking	If true, JBossCMP will lock all rows loaded in a transaction. Most databases implement this by using the SELECT FOR UPDATE syntax when loading the entity, but the actual syntax is determined by the row-locking-template in the datasource-mapping used by this entity.	No, default is false
pk-constraint	If true, JBossCMP will add a primary key constraint when creating tables.	No, default is true
read-ahead	This controls caching of query results and cmr-fields for the entity. This option is discussed in Chapter 6 of the JBossCMP Documentation.	No, see Chapter 6 of the JBossCMP Documentation.

Tag Name	Description	Required
list-cache-max	This specifies the number of read-lists that can be tracked by this entity. This option is discussed in Chapter 6 of the JBossCMP Documentation.	No, default is 1000
table-name	This is the name of the table that will hold data for this entity. Each entity instance will be stored in one row of this table.	No, default is ejb-name

In the cmp-field element, you can control the name and datatype of the column. A cmp-field can also be mapped to several columns, and this is discussed in Chapter 3 of the JBossCMP Documentation. A detailed description of each element is shown in Table 9-2.

Table 9-2, cmp-field Tags

Tag Name	Description	Required
field-name	This is the name of the cmp-field that is being configured. It must match the name of a cmp-field declared for this entity in the ejb-jar.xml file.	Yes
column-name	This is the name of the column to which the cmp-field is mapped.	No, default is field-name
not-null	If this empty element is present, JBossCMP will add NOT NULL to the end of the column declaration when automatically creating the table for this entity.	No, default for primary key fields and primitives is not-null
jdbc-type	This is the JDBC type that is used when setting parameters in a JDBC PreparedStatement or loading data from a JDBC ResultSet. The valid types are defined in java.sql.Types.	Only required if sql-type is specified, default is based on datasource-mapping
sql-type	This is the SQL type that is used in create table statements for this field. Valid sql-types are only limited by your database vendor.	Only required if jdbc-type is specified, default is based on datasource-mapping

Container Managed Relationships

Container Managed Relationships (CMRs) are a powerful new feature of CMP 2.0. Programmers have been creating relationships between entity objects since EJB 1.0 was introduced (not to mention since the introduction of databases), but before CMP 2.0 the programmer had to write a lot of code for each relationship in order to extract the primary key of the related entity and store it in a pseudo foreign key field. The simplest relationships were tedious to code, and complex relationships with referential integrity required many hours to code. With CMP 2.0 there is no need to code relationships by hand. The container can manage one-to-one, one-to-many and many-to-many relationships, with referential integrity. One restriction with CMRs is that they are only defined between local interfaces. This means that a relationship cannot be created between two entities in different virtual machines.[1]

There are two basic steps to create a container managed relationship: create the cmr-field abstract accessors and declare the relationship in the ejb-jar.xml file. The following two sections describe these steps.

CMR-Field Abstract Accessors

A cmr-field abstract accessor has the same signatures as cmp-field, except a single-valued cmr-field must return the local interface of the related entity, and a multi-valued cmr-field can only return a java.util.Collection (or java.util.Set) object. For example, to declare a one-to-many relationship between Organization and Gangster, first add the following to the OrganizationBean class:

```
public abstract class OrganizationBean implements EntityBean {
   public abstract Set getMemberGangsters();
   public abstract void setMemberGangsters(Set gangsters);
}
```

Listing 9-6, Collection Valued cmr-field Abstract Accessor Declaration

Second, add the following to the GangsterBean class:

```
public abstract class GangsterBean implements EntityBean {
   public abstract Organization getOrganization();
   public abstract void setOrganization(Organization org);
}
```

Listing 9-7, Single Valued cmr-field Abstract Accessor Declaration

Although in *Listing 9-6* and *Listing 9-7* each bean declared a cmr-field, only one of the two beans in a relationship must have a set of accessors. As with cmp-fields, a cmr-field is required to have both a getter and a setter method.

[1] The EJB specification does not even allow for relationships between entities in different applications within the same VM.

Relationship Declaration

The declaration of relationships in the ejb-jar.xml file is complicated and error prone. The XML used to declared relationships is as inconsistent as Visual Basic syntax. The best way to configure a relationship is to use a tool, such as XDoclet, or cut and paste a working relationship. The ejb-jar.xml declaration of the Organization-Gangster relationship follows:

```
<ejb-jar>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Organization-Gangster</ejb-relation-name>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>org-has-gangsters</ejb-relationship-role-name>
            <multiplicity>One</multiplicity>
            <relationship-role-source>
               <ejb-name>OrganizationEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
               <cmr-field-name>memberGangsters</cmr-field-name>
               <cmr-field-type>java.util.Set</cmr-field-type>
            </cmr-field>
         </ejb-relationship-role>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-belongs-to-org</ejb-relationship-role-name>
            <multiplicity>Many</multiplicity>
            <cascade-delete/>
            <relationship-role-source>
               <ejb-name>GangsterEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
               <cmr-field-name>organization</cmr-field-name>
            </cmr-field>
         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</ejb-jar>
```

Listing 9-8, The ejb-jar.xml Relationship Declaration

After adding the cmr-field abstract accessors and declaring the relationship, the relationship should be functional. For more information on relationships, see Chapter 4 of the JBossCMP Documentation, or Section 10.3 of the Enterprise JavaBeans Specification Version 2.0 Final Release. The next section discusses the database mapping of the relationship.

Relationship Mapping

Relationships can be mapped using either a foreign key or a separate relation-table. One-to-one and one-to-many relationships use the foreign key mapping style by default, and many-to-many relationships use only the relation-table mapping style. The mapping of a relationship is declared in the relationships section of the jbosscmp-jdbc.xml file. Relationships are identified by the ejb-relation-name from the ejb-jar.xml file. The basic template of the relationship mapping declaration for Organization-Gangster follows:

```
<jbosscmp-jdbc>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Organization-Gangster</ejb-relation-name>

         <!--
          | Mapping style declaration
          | <foreign-key> or <relation-table>
          -->

         <read-only>false</read-only>
         <read-time-out>300</read-time-out>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>org-has-gangsters</ejb-relationship-role-name>

            <fk-constraint>true</fk-constraint>

            <key-fields>
               <!-- Organization primary key field mappings -->
            </key-fields>

            <read-ahead><strategy>on-load</strategy></read-ahead>

         </ejb-relationship-role>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-belongs-to-org</ejb-relationship-role-name>
            <fk-constraint>true</fk-constraint>

            <key-fields>
               <!— Gangster primary key field mappings -->
            </key-fields>

            <read-ahead><strategy>on-load</strategy></read-ahead>

         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</jbosscmp-jdbc>
```

Listing 9-9, The jbosscmp-jdbc.xml Relationship Mapping Template

After the ejb-relation-name of the relationship being mapped is declared, the mapping style must be declared using a foreign-key-mapping element or a relation-table-mapping element, both of which are discussed later. The ejb-relationship-role elements are optional, but if one is declared, the other must also be declared. The read-only and read-time-out elements are described in Chapte 4 of the JBossCMP Documentation.

Relationship Role Mapping

Each of the two ejb-relationship-role elements contains mapping information specific to an entity in the relationship. A detailed description of the main elements follows:

Table 9-3, ejb-relationship-role Tags

Tag Name	Description	Required
ejb-relationship-role-name	This is the name of the role to which this configuration applies. This element must match the name of one of the roles declared for this query in the ejb-jar.xml file.	Yes
fk-constraint	If true, JBossCMP will add a foreign key constraint to the tables. JBossCMP will only add the constraint if both the primary table and the related table were created by JBossCMP during deployment.	No, default is false
key-fields	This specifies the mapping of the primary key fields of the current entity. This element is only necessary if exact field mapping is desired. Otherwise, the key-fields element must[2] contain a key-field element for each primary key field of the current entity. The details of this element are described below.	No, default depends on mapping type
read-ahead	This controls the caching of this relationship. This option is discussed in Chapter 6 of the JBossCMP Documentation.	No, see Chapter 6 of the JBossCMP Documentation.

As noted in *Table 9-3* the key-fields element contains a key-field for each primary key field of the current entity. The key-field element uses the same syntax as the cmp-field element of the entity, except that key-field does not support the not-null option. Key-fields of a relation-table are automatically not null, because they are the primary key of the table. On

[2] Note that with foreign key mapping this element can be empty; this means that there will be not be a foreign key for the current entity. This is required for the many side of a one-to-many relationship, such a Gangster in the Organization-Gangster example.

the other hand, foreign key fields must always be nullable.[3] A detailed description of the elements contained in the key-field element follows:

Table 9-4, key-field Tags

Tag Name	Description	Required
field-name	This identifies the field to which this mapping applies. This name must match a primary key field of the current entity.	Yes
column-name	Specifies the column name in which this primary key field will be stored. If this is relationship uses foreign-key-mapping, this column will be added to the table for the related entity. If this relationship uses relation-table-mapping, this column is added to the relation-table. This element is not allowed for mapped dependent value class; instead use the property element described in Chapter 3 of the JBossCMP Documentation.	No, default depends on mapping type
jdbc-type	This is the JDBC type that is used when setting parameters in a JDBC PreparedStatement or loading data from a JDBC ResultSet. The valid types are defined in java.sql.Types.	Only required if sql-type is specified, default is based on datasource-mapping
sql-type	This is the SQL type that is used in create table statements for this field. Valid sql-types are only limited by your database vendor.	Only required if jdbc-type is specified, default is based on datasource-mapping

Foreign Key Mapping

Foreign key mapping is the most common mapping style for one-to-one and one-to-many relationships, but is not allowed for many-to many relationships. The foreign-key-mapping element is simply declared by adding an empty foreign key-mapping element to the ejb-relation element.

[3] The current implementation of JBossCMP inserts a row into the database for a new entity between ejbCreate and ejbPostCreate. Since the EJB specification does not allow a relationship to be modified until ejbPostCreate, a foreign key will be initially set to null. There is a similar problem with removal. This limitation will be removed in a future release.

As noted in the previous section, with a foreign key mapping the key-fields declared in the ejb-relationship-role are added to the table of the related entity. If the key-fields element is empty, a foreign key will not be created for the entity. In a one-to-many relationship, the many side (Gangster in the example) must have an empty key-fields element, and the one side (Organization in the example) must have a key-fields mapping. In one-to-one relationships, one or both roles can have foreign keys.

The foreign key mapping is not dependent on the direction of the relationship. This means that in a one-to-one unidirectional relationship (only one side has an accessor) one or both roles can still have foreign keys.

The complete foreign key mapping for the Organization-Gangster relationship follows:

```
<jbosscmp-jdbc>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Organization-Gangster</ejb-relation-name>
         <foreign-key-mapping/>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>org-has-gangsters</ejb-relationship-role-name>
            <key-fields>
               <key-field>
                  <field-name>name</field-name>
                  <column-name>organization</column-name>
               </key-field>
            </key-fields>
         </ejb-relationship-role>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-belongs-to-org</ejb-relationship-role-name>
            <key-fields/>
         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</jbosscmp-jdbc>
```

Listing 9-10, The jbosscmp-jdbc.xml Foreign Key Mapping

Relation-table Mapping

Relation-table mapping is less common for one-to-one and one-to-many relationships, but is the only mapping style allowed for many-to-many relationships. The relation-table-mapping for the Gangster-Job relationship follows:

```
<jbosscmp-jdbc>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Gangster-Jobs</ejb-relation-name>
         <relation-table-mapping>
            <table-name>gangster_job</table-name>
         </relation-table-mapping>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-has-jobs</ejb-relationship-role-name>
            <key-fields>
               <key-field>
                  <field-name>gangsterId</field-name>
                  <column-name>gangster</column-name>
               </key-field>
            </key-fields>
         </ejb-relationship-role>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>job-has-gangsters</ejb-relationship-role-name>
            <key-fields>
               <key-field>
                  <field-name>name</field-name>
                  <column-name>job</column-name>
               </key-field>
            </key-fields>
         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</jbosscmp-jdbc>
```

Listing 9-11, The jbosscmp-jdbc.xml Relation-table Mapping

The relation-table-mapping element contains a subset of the options available in the entity element, and these elements are discussed in Chapter 4 of the JBossCMP Documentation.

Queries

Another powerful new feature of CMP 2.0 is the introduction of the EJB Query Language (EJB-QL) and ejbSelect methods. In CMP 1.1, every EJB container had a different way to specify finders, and this was a serious threat to J2EE portability. In CMP 2.0, EJB-QL was created to specify finders and ejbSelect methods in a platform independent way. The ejbSelect method is designed to provide private query statements to an entity implementation. Unlike finders, which are restricted to only return entities of the same type as the home interface on which they are defined, ejbSelect methods can return any entity type or just one field of the entity.

EJB-QL is beyond the scope of this guide, so only the basic method coding and query declaration will be covered here. For more information, see Chapter 11 of the Enterprise JavaBeans Specification Version 2.0 Final Release or one of the many excellent articles written on CMP 2.0.

Finder and ejbSelect Declaration

The declaration of finders has not changed in CMP 2.0. Finders are still declared in the home interface (local or remote) of the entity. Finders defined on the local home interface do not throw a RemoteException. The following code declares the findBadDudes finder on the GangsterHome interface:

```
public interface GangsterHome extends EJBLocalHome {
   Collection findBadDudes(int badness) throws FinderException;
}
```

Listing 9-12, Finder Declaration

The ejbSelect methods are declared in the entity implementation class, and must be public abstract just like cmp-field and cmr-field abstract accessors. Select methods must be declared to throw a FinderException, but not a RemoteException. The following code declares an ejbSelect method:

```
public abstract class GangsterBean implements EntityBean {
   public abstract Set ejbSelectBoss(String name) throws FinderException;
}
```

Listing 9-13, ejbSelect Declaration

EJB-QL Declaration

The EJB 2.0 specification requires that every ejbSelect or finder method (except findByPrimaryKey) have an EJB-QL query defined in the ejb-jar.xml file.[4] The EJB-QL query is declared in a query element, which is contained in the entity element. The following is the declaration for the queries defined in *Listing 9-12* and *Listing 9-13*:

```
<ejb-jar>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findBadDudes</method-name>
               <method-params><method-param>int</method-param></method-params>
            </query-method>
            <ejb-ql><![CDATA[
               SELECT OBJECT(g)
               FROM gangster g
               WHERE g.badness > ?1
            ]]></ejb-ql>
         </query>
         <query>
            <query-method>
```

[4] Currently this is not enforced by JBossCMP, but a future release will enforce this by throwing an exception during deployment.

```
                <method-name>ejbSelectBoss</method-name>
                <method-params>
                   <method-param>java.lang.String</method-param>
                </method-params>
             </query-method>
             <ejb-ql><![CDATA[
                SELECT DISTINCT underling.organization.theBoss
                FROM gangster underling
                WHERE underling.name = ?1 OR underling.nickName = ?1
             ]]></ejb-ql>
          </query>
       </entity>
    </enterprise-beans>
</ejb-jar>
```

Listing 9-14, The ejb-jar.xml Query Declaration

EJB-QL is similar to SQL but has some surprising differences. The following are some important things to note about EJB-QL:

- EJB-QL is a typed language, meaning that it only allows comparison of like types (i.e., strings can only be compared with strings).

- In an equals comparison a variable (single valued path) must be on the left hand side. Some examples follow:[5]

```
g.hangout.state = 'CA'                  Legal
'CA' = g.shippingAddress.state          NOT Legal
'CA' = 'CA'                             NOT Legal
(r.amountPaid * .01) > 300              NOT Legal
r.amountPaid > (300 / .01)              Legal
```

- Parameters use a base 1 index like java.sql.PreparedStatement.

- Parameters are only allowed on the right hand side of a comparison. For example:

```
gangster.hangout.state = ?1             Legal
?1 = gangster.hangout.state             NOT Legal
```

- Datetime variables (simgle valued paths) are only allowed to use the > and < operators and EJB-QL does not have datetime literals. For example:

```
someEntity.datetimeField > ?1           Legal
someEntity.datetimeField >= ?1          NOT Legal
someEntity.datetimeField BETWEEN ?1 AND ?2                   NOT Legal
someEntity.datetimeField > ?1 AND someEntity.datetimeField < ?2     Legal
```

[5] The example "(r.amountPaid * .01) > 300" is presented on page 244 of "Enterprise JavaBeans 3rd Edition" by Richard Monson-Haefel to demonstrate the use of arithmetic operators in a WHERE clause, and is included here to highlight the fact that it is not legal EJB-QL syntax.

Overriding the EJB-QL to SQL Mapping

The EJB-QL to SQL mapping can be overridden in the jbosscmp-jdbc.xml file. The finder or ejbSelect is still required to have an EJB-QL declaration in the ejb-jar.xml file, but the ejb-ql element can be left empty. Currently the SQL can be overridden with JBossQL, which removes som of the restricutions of EJB-QL and adds support for a ORDER BY clause, DynamicQL, which allows for the runtime declaration and execution of EJB-QL and JBossQL queries, DeclaredSQL, which supports the declaration of the exact SQL for a query, or a BMP style custom ejbFind method, which allows the programmer to code the query by hand. All of these overrides are discussed in Chapter 5 of the JBossCMP Documentation.

10. Connecting to Databases and other Resource Managers

The world of JCA by David Jencks

All connections to databases and other resource managers in jboss 3 are handled through the JCA framework. JCA provides a well thought out model in which the application server handles transactions, security, and pooling or resource management, and the client application code is left free to concentrate on doing useful work. Few databases currently supply JCA adapters, so normally database drivers are used through jca-jdbc wrappers. Databases such as firebird and other resource managers such as SAP, CICS, etc can be accessed directly through jca adapters.

To use a JCA adapter or wrapped jdbc driver, you must configure 3 mbeans and a JAAS security domain. Examples of configurations for popular databases are supplied in the docs/examples/jca directory. The mbeans are shown there in a nested format which makes their relationship clear. The ConnectionManager mbean determines the level of transaction support for the adapter deployment. The choices are org.jboss.resource.connectionmanager.NoTxConnectionManager, for adapters that do not support transactions or adapters for which you want only autocommit behavior, org.jboss.resource.connectionmanager.LocalTxConnectionManager, for adapters such as wrapped non-xa jdbc drivers that support only local (one phase) transactions, and org.jboss.resource.connectionmanager.XaTxConnectionManager, for adapters that support xa transactions. If you have a choice, use xa transaction support for better concurrency and less chance of resource deadlock. Along with the references to the other two required mbeans, the crucial configuration bit here is the <attribute name="SecurityDomainJndiName">java:/jaas/MyAdapterRealm</attribute> which identifies the JAAS security domain used for container managed security. For the example here, you must include a realm in your login-conf.xml file like this:

```
<application-policy name = "MyAdapterRealm">
    <authentication>
        <login-module code = "org.jboss.resource.security.ConfiguredIdentityLoginModule"
          flag = "required">
            <module-option name = "principal">yourprincipal</module-option>
            <module-option name = "userName">yourusername</module-option>
            <module-option name = "password">yourpassword</module-option>
            <module-option name = "managedConnectionFactoryName">
```

```
                jboss.jca:service=XaTxCM,name=MyAdapterDS
            </module-option>
        </login-module>
    </authentication>
</application-policy>
```

Include the security info needed for connections. There are two necessary cross references here: the same name must appear in the mbean config and as the name of the realm, and the managedConnectionFactoryName in the authentication configuration must exactly match the ObjectName of the ConnectionManager mbean you are configuring for your adapter. This particular example is using the ConfiguredIdentityLoginModule. Other login modules will be available soon, and if your adapter requires specific credentials you may need to supply your own login module. Most adapters with such requirements should come with appropriate login modules.

The RARDeployment mbean indicates properties of the jca adapter (such as a jdbc driver wrapper) you are using. At the moment some of these are determined from a legacy mbean that deploys the classes in the adapter package and reads the adapter deployment descriptor. This is referenced by this tag:

```
<depends optional-attribute-name="OldRarDeployment">
  jboss.jca:service=RARDeployment,name=JBoss LocalTransaction JDBC Wrapper
</depends>
```

Here the \u"name\u" part of the ObjectName must match the DisplayName attribute of the desired resource adapter. If necessary you can determine this by dropping the rar package into deploy and looking at the resulting automatically created org.jboss.resource.RARDeployment mbean in the jmx viewer. The properties to set on the ManagedConnectionFactory instance are specified in the ManagedConnectionFactoryProperties attribute in an xml element containing elements such as this:

```
<config-property name="ConnectionURL" type="java.lang.String">
    jdbc:informix-sqli://myhost.mydomain.com:1557/mydb:INFORMIXSERVER=myserver
</config-property>
```

specifying the property name, type, and value. If a property does not need to be set, leave it out. The RARDeployment mbean previously mentioned will also show the available properties.

The other crucial attribute in this mbean is the JndiName specifying where to bind the ConnectionFactory or DataSource that your application will use.

Finally the ManagedConnectionPool mbean configures pooling for the ManagedConnections from the resource adapter instance. MinSize and MaxSize are pretty clear. BlockingTimeoutMillis indicates the maximum time to wait for a connection to be returned to the pool if none are available: it does not affect waiting for the driver to create a new connection. IdleTimeoutMinutes indicates approximately how long a connection can be unused before being discarded. This is especially useful if your driver closes unused connections without telling you. Generally an idle connection should be removed within 1.5 * IdleTimeoutMinutes.

Finally, and very importantly, the criteria attribute indicates how the ConnectionManager decides if two connections are interchangeable. You can usually get connections by either getConnection() or by getConnection(userinfo). The former relies on the container to manage security, the latter on the application. If you are using container managed security, set the criteria to ByContainer: if you are using only application managed security, use ByApplication. Do not use both with one ConnectionManager. If you are using default user/password values specified in the ManagedConnectionFactoryProperties, or if your adapter supports reauthentication, use ByNothing. Note that if you are using Application managed security you must leave the SecurityDomainJndiName attribute empty to avoid exceptions and general failure. If you can set other properties in your getConnection(stuff) call, such as transaction isolation perhaps, you may need ByContainerAndApplication. Note that the MinSize and MaxSize apply to each distinguishable set of connections. If you are using ByContainer, and have 5 user identities, and a MaxSize of 10, you can get up to 50 connections, 10 per identity. Currently there is no support for a "global" MaxSize attribute.

If you are using a jdbc driver, you have two choices of jca-jdbc wrapper. If your driver supports only the Driver interface, use the jboss-local-jdbc.rar. If your driver provides an XADataSource implementation, use the jboss-xa.rar. Note that with the jboss-xa.rar, one of the ManagedConnectionFactoryProperties is a semicolon separated list of name=value pairs for the XADataSource properties of your driver. As of this writing, the xa wrapper has some problems if you hold connections over calls to other ejbs or if you attempt to hold connections between method calls.

Examples for popular databases are provided in the docs/examples/jca directory.

Deployment of your adapter configuration

If you are deploying a wrapped database driver, put the driver jar in lib or use a classpath element to force its loading before the mbean deployment. With the current system, the depends elements will force the resource adapter module (rar) to be deployed before your ConnectionManager configuration. So: make sure the driver.jar, if required, is in lib, deploy the rar by copying it to deploy, and deploy the ConnectionManager mbeans by copying the *-service.xml file to deploy. Don't forget that you need to set up security information in login-conf.xml.

11. Using the JMS API

Messaging Service for Asynchronous Calls by Hiram Chirino

The JMS API stands for Java™ Message Service Application Programming Interface, and it is used by applications to send asynchronous "business quality" messages to other applications. In the JMS world, messages are not sent directly to other applications. Instead, messages are sent to destinations, also known as "queues" or "topics". The applications sending messages do not need to worry if the receiving applications are up and running, and conversely, receiving applications do not need to worry about the sending application's status. Both senders, and receivers only interact with the destinations.

The JMS API is the standardized interface to a JMS provider, sometimes called a Message Oriented Middleware (MOM) system. JBoss comes a with JMS 1.0.2b compliant JMS provider called JBossMQ. When you use the JMS API with JBoss, you are using the JBossMQ engine transparently. JBossMQ implements the JMS spec fully, and adds few extra features, so the best JBossMQ user guide is the JMS specification! For more information about the JMS API please visit the `JMS Tutorial` or `JMS Downloads & Specifications`. This rest of this chapter will assume you are already familiar with JMS and it will mainly cover all the JBoss specifics.

Using JMS with JBoss

Looking up the ConnectionFactory

To use JMS in a client application, it is recommended that JNDI[6] be used to locate the JMS ConnectionFactory objects that create connections to the JBoss JMS provider, JBossMQ.

Most standalone JMS client applications[7] should use the OIL invocation protocol. The OIL uses and optimized socket protocol to communicate with the server. Listing 11-1, looking up

[6] More information about the Java Naming and Directory Services (JNDI) can be found at the java.sun.com web site.

[7] A standalone JMS client application is considered to be one that does not run inside the JBoss application server.

a QueueConnectionFactory and Listing 11-2, looking up a TopicConnectionFactory shows you the typical way the OIL ConnectionFactory is created in a client application.

Listing 11-1, looking up a QueueConnectionFactory

```
InitialContext ctx = new InitialContext();
QueueConnectionFactory qcf=(QueueConnectionFactory)ctx.lookup("ConnectionFactory");
```

Listing 11-2, looking up a TopicConnectionFactory

```
InitialContext ctx = new InitialContext();
TopicConnectionFactory qcf=(TopicConnectionFactory)ctx.lookup("ConnectionFactory");
```

A ConnectionFactory can also be constructed without using JNDI via JBossMQ specific APIs but it out of the scope of this document.

Looking up Queues and Topics

Once you have a ConnectionFactory, the next thing you will need to do is find out how to get the Destination (Queue and Topic) objects. There are two ways you can get references to the Destination objects.

1. Use JNDI to lookup the Destination object.

    - Queues are stored under the “queue” subcontext. Therefore, to lookup a queue named *queueName,* the JNDI location you would look up should be “queue/*queueName*”.

    - Topics are stored under the “topic” subcontext. Therefore, to lookup a topic named *topicName,* the JNDI location you would look up should be “topic/*topicName*”.

2. Use the QueueSession.createQueue(String queueName) or TopicSession.createTopic(String topicName) methods. These methods do not create NEW Destinations, they create references to existing Destinations.

Configuring JBoss JMS Objects

All JBoss JMS object such as the ConnectionFactorys, Topics and Queues are configured via a JMX MBean[8]. In addition to the standard JMS objects, there are also JMX MBean that let

[8] See the **Error! Reference source not found.** chapter for more info on MBeans.

you configure the JBossMQ kernel. The core, kernel level, MBeans are configured in the JBossMQ-service.xml file. Unless you are an advanced user, you should not have to edit this file. If you undeploy the JBossMQ-service.xml file, you in effect shutdown JBossMQ.

The deployed JBossMQ-destinations-service.xml file exists for you to define the destinations that your applications need. Most users will have to edit this file to create new Queues or Topics. An example of a destination definition in the JBossMQ-destinations-service.xml is shown in Listing 11-3, the definition for a Queue Named "testQueue" and in Listing 11-4, the definition for a Topic Named "testTopic". As you can see from the examples, role bases security authorization can be assigned on a Destination by destination basis.

Listing 11-3, the definition for a Queue Named "testQueue"

```
<mbean code="org.jboss.mq.server.QueueManager"
       name="jboss.mq.destination:service=Queue,name=testQueue">
    <depends optional-attribute-name="JBossMQService">jboss.mq:service=Server</depends>
    <depends optional-attribute-
name="SecurityManager">jboss.mq:service=SecurityManager</depends>
    <attribute name="SecurityConf">
      <security>
        <role name="guest" read="true" write="true"/>
        <role name="publisher" read="true" write="true" create="false"/>
        <role name="noacc" read="false" write="false" create="false"/>
      </security>
    </attribute>
</mbean>
```

??? We can save space by getting rid of this Topic config, It is VERY similar to the Queue config ????

Listing 11-4, the definition for a Topic Named "testTopic"

```
<mbean code="org.jboss.mq.server.TopicManager"
       name="jboss.mq.destination:service=Topic,name=testTopic">
  <depends optional-attribute-name="JBossMQService">jboss.mq:service=Server</depends>
    <depends optional-attribute-
name="SecurityManager">jboss.mq:service=SecurityManager</depends>
    <attribute name="SecurityConf">
      <security>
        <role name="guest" read="true" write="true"/>
        <role name="publisher" read="true" write="true" create="false"/>
        <role name="durpublisher" read="true" write="true" create="true"/>
      </security>
    </attribute>
</mbean>
```

Using Message Driven Beans (MDB) with Jboss

A message driven bean is JMS message listner that is managed by a J2EE container. An example of a MDB is show in Listing 11-5, the source code for a simple MDB.

Listing 11-5, the source code for a simple MDB

```
package example.beans;

import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.ejb.EJBException;
import javax.jms.MessageListener;
import javax.jms.Message;

public class PrintBean implements MessageDrivenBean, MessageListener {
  public void setMessageDrivenContext(MessageDrivenContext c) throws EJBException {}
  public void ejbCreate() {}
  public void ejbRemove() {}

  public void onMessage(Message message) {
    System.err.println("Got Message: " + message.toString());
  }
}
```

The ejb-jar.xml deployment descriptor for the example MDB configures how messages are delivered to a MDB. For example, Listing 11-6, the ejb-jar.xml deployment descriptor configures a bean named PrintBean which will receive messages from a queue with no message selector and who's transaction will be bean managed and messages will be auto-acknowledged by the JMS provider.

Listing 11-6, the ejb-jar.xml deployment descriptor

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar>
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>PrintBean</ejb-name>
      <ejb-class>example.beans.PrintBean</ejb-class>
      <message-selector></message-selector>
      <transaction-type>Bean</transaction-type>
      <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
      <message-driven-destination>
      <destination-type>javax.jms.Queue</destination-type>
      </message-driven-destination>
    </message-driven>
  </enterprise-beans>
</ejb-jar>
```

Since the ejb-jar.xml deployment descriptor says vendor neutral, the jboss.xml deployment descriptor contains all the needed information to complete the configuration for the MDB. Listing 11-7, the jboss.xml deployment descriptor shows the destination queue is assigned to the PrintBean MDB.

Listing 11-7, the jboss.xml deployment descriptor

```
<?xml version="1.0"?>
<jboss>
  <enterprise-beans>
    <message-driven>
      <ejb-name>PrintBean</ejb-name>
      <configuration-name>Standard Message Driven Bean</configuration-name>
      <destination-jndi-name>queue/testQueue</destination-jndi-name>
    </message-driven>
  </enterprise-beans>
</jboss>
```

Summary

This chapter covered how to use JMS from JBoss. The main topics covered were the JMS provider specific issues like:

- How to locate a ConnectionFactory, a Topic, or a Queue
- What files are used to configure JBossMQ and how to create a new Topic or Queue
- How to create a simple MDB.

12. MBean Configuration and Dependency management

How to deploy and configure your mbeans by David Jencks

Almost everything important in JBoss is an mbean, such as ejbs, the naming service, resource adapters, etc. Most of these (except, as of now, ejbs) are configured through jboss-specific xml deployment descriptors in *-service.xml files. These may be packed with the classes they need in a sar (service archive) or deployed separately. We will cover the format of these configuration files, the structure of sar files, and the two types of dependency management: between mbeans, and between an mbean and its class. We will also discuss the mbean lifecycle.

*Listing 10-12-1, *-service.xml files look like this:*

```
<?xml version="1.0" encoding="UTF-8"?>
<service>
  <mbean code="org.jboss.resource.connectionmanager.XATxConnectionManager"
     name="jboss.jca:service=XaTxCM,name=FirebirdDS">
    <depends>jboss.jca:service=RARDeployer</depends>
    <depends optional-attribute-name="ManagedConnectionFactoryName">
      jboss.jca:service=XaTxDS,name=FirebirdDS
    </depends>
    <attribute name="SecurityDomainJndiName">java:/jaas/FirebirdDBRealm</attribute>
    <attribute name="TransactionManager">java:/TransactionManager</attribute>
  </mbean>
</service>
```

There is one service element containing one or more mbean elements. The mbean element must include the class name (code) and ObjectName (name) attributes. ObjectNames must be unique server-wide. It is also possible to include constructor parameters for a non-default constructor.

There are three kinds of attributes allowed in mbeans;

- Normal "attribute" attributes, which can be any reasonable type or an xml Element. They must correspond to an mbean attribute.

- Depends attributes, which do not need to correspond to an mbean attribute. If it corresponds to an mbean attribute, that attribute type must be ObjectName and the name must be specified with optional-attribute-name. For convenience you may specify an ObjectName or include the complete mbean configuration as a nested element.

- Depends-list attributes, which contain a list of ObjectNames or mbeans: <depends-list optional-attribute-name="blah" <depends-list-element>jboss.jca:service=something</depends-list-element></depends-list>. Again, you can supply either an ObjectName or a complete mbean configuration. The mbean attribute type should be "List".

Service Lifecycle.

Generally mbeans should implement org.jboss.system.Service or extend org.jboss.system.ServiceMBeanSupport to participate in the JBoss lifecycle management. This consists of 6 steps:

20. Object creation and configuration. Using the xml file format just discussed, the mbean is instantiated on the mbean server and configured with the specified attribute values. It is not yet ready for use.

21. Create step: JBoss calls the create method (filtered to createService in ServiceMBeanSupport). In this step, your mbean should set up its internal configuration to expose some representation of everything it will make available. However, it should not refer to or use anything outside itself (such as mbeans specified in depends elements or jndiname references).

22. Start step. Here your mbean can look into the outside world and use services of mbeans it has references to. After this step, the mbean is ready for use.

23. Stop step. This should be the reverse of the start step. It should not have state information imported from other mbeans after this step. The mbean is no longer usable.

24. Destroy step. This should reverse the Create step.

25. Unregister. The mbean is removed from the mbean server.

MBean-Class Dependency Management

JBoss manages the dependency between an mbean and its class. If you attempt to deploy an mbean whose class is not loaded, the configuration information is put on a waiting list. When the class becomes available due to some additional deployment, your mbean will be deployed. If the class of an existing mbean is undeployed, the mbean will also be undeployed and its last configuration put on the same waiting list, so if the class is redeployed the mbean will be resurrected.

MBean-MBean Dependency Management

JBoss manages dependencies between mbeans using the depends and depends-list attribute styles. An mbean will wait to progress through the create and start steps until all the mbeans it depends on have been created and started. Similarly, stopping or destroying an mbean will stop or destroy mbeans that depend on it. Circular dependencies are not handled. You can use this to assure that mbeans are started in the correct order, even if they are in separate configuration files deployed in the wrong order.

SAR file format and package nesting

*-service.xml files may be deployed alone or packaged into a sar (service archive) file. In a sar, the file must have the name jboss-service.xml and be located in META-INF (note case). The sar file may also contain classes, jars, or any other deployable package such as ear, war, rar, or (ejb) jar. In addition, all of these packages may include additional nested deployable packages. These packages are deployed from inside (most deeply nested) out. Note that most of these combinations are JBoss specific and not j2ee compliant. Generally, although manifest classpath entries can be used, they are usually unnecesary in JBoss.

Classpath Element

The *service.xml file may also contain one or more classpath elements, of the form <classpath codebase="lib" archives="first.jar, second.jar"/>. The codebase attribute indicates where the packages are to be found, and the archives attributes contains a comma separated list of packages to be deployed. JBoss will attempt to deploy these before processing the mbean configurations. This dependency management can usually be left to the automatic Mbean-Class dependency management.

LocalDirectory Element

The *service.xml file may also contain one or more local-directory elements, of the form <local-directory path="some-path"/>. This can only be used in a sar file. This will result in whatever is located at some-path within the sar being copied verbatim to the "db" directory in your JBoss installation. If a file is already present, it is not overwritten and these files are not removed when the sar is undeployed. This can be used to install prebuilt configurations, database files, etc.

Deployment

Now that you have your mbeans written and the configuration determined, you can deploy your package by copying it into a directory watched by the URLDeploymentScanner, typically server/default/deploy. For specific deployments you can list the URLs to deploy explicitly in the URLDeploymentScanner. You can also deploy programmatically by calls on the mbean server:

```
server.invoke(
    new ObjectName( jboss.system:service=MainDeployer ),
    "deploy",
    new Object[] { myUrl},
    new String[] { "java.net.URL" }
);
```

(Exception handling and remote access to the mbean server ommitted).

Chapter 13

13. Web Integration

How to Serve Web Content by Scott Stark and Jan Bartel

JBoss supports embedding of web containers through abstract MBean. Two popular servlet containers have embedded services bundled with JBoss. The Jetty-4.x servlet container is included as the default servlet container in the standard JBoss 3.0 distribution. A separate JBoss/Tomcat-4.x distribution includes an embedded version of Tomcat-4.0.3. This chapter provides the basic configuration details of the Jetty and Tomcat services. For the full details of these services as well as the servlet container integration interface see the full JBoss 3.0 documentation.

Configuring Jetty

What is Jetty

Jetty is a pure Java web server and servlet container compliant to the HTTP1.1, Servlet 2.3 & JSP 1.2 specifications developed by `Mort Bay Consulting (http://www.mortbay.com)`. It has been designed to be fast, lightweight, extensible, and embeddable. This section discusses the embedding of Jetty within JBoss, but for more general information on Jetty, visit the `Jetty website (http://jetty.mortbay.org).`

Integration with JBoss

Jetty is fully integrated with the JBoss environment in terms of:

In-JVM optimized calls.

The overhead of RMI is avoided when the servlet and EJB containers are run in the same JVM.

Implementing a web container service

The Jetty integration extends the `org.jboss.web.AbstractWebContainer` class to enable Jetty to conform to the standard JBoss web container service interface. This allows the Jetty Service to be stopped and restarted, to hot-deploy webapps and for those webapps to be able to reference EJBs , resources and other objects in the J2EE JNDI environment.

Logging

Debug and informational log output from the Jetty Service is adapted to the standard JBoss logging service.

Security

The Jetty integration classes adapt the servlet security environment to the JBoss securityenvironment. This allows webapps performing basic or form based authentication to transparently access the JBossSX framework.

JMX

As a compliant JBoss service, Jetty can be controlled from the mbean viewer available on port 8082. Jetty makes available each of its constituent components as mbeans allowing detailed management of configuration, debugging and statistics gathering. Additionally, Jetty creates an mbean for every deployed web application context, allowing individual contexts to be stopped and (re)started without undeploying the webapp itself.

•Clustered Sessions

The clustered HTTP Session service can be used to provide distributed sessions.

Deployment

Jetty is packaged as a service archive file called `jetty-plugin.sar`. It deploys automatically with JBoss with a default configuration:

1.it will listen on port 8080 for HTTP requests (note that as no demonstration webapp is provided, hitting `localhost:8080/` will result in your receiving a "`404 NotFound`")

2.the HTTP request log is written to the standard JBoss log directory as files with names of the form `yyyy_mm_dd.request.log` which rollover daily

3.output from Jetty such as debug and informational messages are directed to the standard JBoss log

Configuration

The default configuration can be modified by editing the Jetty configuration file found inside the sar as `jetty-plugin.sar/META-INF/jboss-service.xml`. To modify it, first unpack the sar, make your changes, repack it and copy it back to the deploy directory. JBoss will reload and restart Jetty with it's new configuration. Alternatively, for non-permanent configuration changes, you can use the JMX Agent on port 8082.

The default Jetty `jboss-service.xml` file looks like:

13.0 Standard Jetty service configuration file jboss-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <!-- ================================================================== -->
  <!-- Web Container                                                      -->
  <!-- ================================================================== -->

  <!--
     | Be sure to check that the configuration values are valid for your
     | environment.
   -->

  <mbean code="org.jboss.jetty.JettyService" name="jboss.web:service=Jetty">

    <!-- ================================================================= -->
    <!-- Uncomment the following line ONLY if you want to provide a custom -->
    <!-- webdefault.xml file in place of the standard one. Place your      -->
    <!-- file in the src/etc directory to have it automatically included   -->
    <!-- in the build.                                                     -->
    <!-- ================================================================= -->

    <!--
    <attribute name="WebDefault">webdefault.xml</attribute>
    -->

    <!-- ================================================================== -->
    <!-- If true, .war files are unpacked to a temporary directory. This    -->
    <!-- is useful with JSPs.                                               -->
    <!-- ================================================================== -->

    <attribute name="UnpackWars">true</attribute>

    <!-- ================================================================== -->
    <!-- If true, Jetty first delegates loading a class to the webapp's     -->
    <!-- parent class loader (a la Java 2). If false, Jetty follows the     -->
    <!--  Servlet 2.3 specification, and tries the webapp's own loader      -->
    <!-- first (for "non-system" classes)                                   -->
    <!-- ================================================================== -->

    <attribute name="Java2ClassLoadingCompliance">true</attribute>
    <!-- ==================================================================== -->
    <!-- Configuring Jetty. The XML fragment contained in the                 -->
    <!-- name="ConfigurationElement" attribute is a Jetty-style               -->
    <!-- configuration specification.  It is used to configure Jetty with     -->
    <!-- a listener on port 8080, and a HTTP request log location.            -->
    <!-- The placement here of other Jetty XML configuration statements       -->
    <!-- for deploying webapps etc is not encouraged: if you REALLY NEED      -->
    <!-- something extra, place it in WEB-INF/jetty-web.xml files             -->
    <!-- ==================================================================== -->

    <attribute name="ConfigurationElement">
      <Configure class="org.mortbay.jetty.Server">

          <!-- ============================================================ -->
        <!-- Add the listener                                           -->
          <!-- ============================================================ -->
          <Call name="addListener">
            <Arg>
             <New class="org.mortbay.http.SocketListener">
              <Set name="Port"><SystemProperty name="jetty.port"
default="8080"/></Set>
              <Set name="MinThreads">5</Set>
              <Set name="MaxThreads">255</Set>
              <Set name="MaxIdleTimeMs">30000</Set>
              <Set name="MaxReadTimeMs">10000</Set>
              <Set name="MaxStopTimeMs">5000</Set>
              <Set name="LowResourcePersistTimeMs">5000</Set>
              </New>
            </Arg>
          </Call>

          <!-- ============================================================ -->
        <!-- Add the HTTP request log                                   -->
          <!-- ============================================================ -->
          <Set name="RequestLog">
            <New class="org.mortbay.http.NCSARequestLog">
              <Arg><SystemProperty name="jboss.server.home.dir"/><SystemProperty
name="jetty.log" default="/log"/>/yyyy_mm_dd.request.log
             </Arg>
              <Set name="retainDays">90</Set>
              <Set name="append">true</Set>
              <Set name="extended">true</Set>
              <Set name="LogTimeZone">GMT</Set>
            </New>
          </Set>

          <!-- ============================================================ -->
          <!-- Uncomment and set at least the Keystore, Password and      -->
        <!-- KeyPassword fields to configure an SSL listener            -->
          <!-- ============================================================ -->
        <!--
           <Call name="addListener">
             <Arg>
```

```
               <New class="org.mortbay.http.SunJsseListener">
                 <Set name="Port">8443</Set>
                 <Set name="MinThreads">5</Set>
                 <Set name="MaxThreads">255</Set>
                 <Set name="MaxIdleTimeMs">30000</Set>
                 <Set name="MaxReadTimeMs">10000</Set>
                 <Set name="MaxStopTimeMs">5000</Set>
                 <Set name="LowResourcePersistTimeMs">2000</Set>
                 <Set name="Keystore"><SystemProperty name="jetty.home"
default="."/>/etc/demokeystore</Set>
                 <Set name="Password">dummy</Set>
                 <Set name="KeyPassword">dummy</Set>
               </New>
             </Arg>
           </Call>
          -->
      </Configure>
     </attribute>

    <!-- ================================================================ -->
    <!-- Options for distributed session management are:                  -->
    <!--     org.jboss.jetty.session.CoarseDistributedStore               -->
    <!--     org.jboss.jetty.session.ClusteredStore                       -->
    <!-- ================================================================ -->

    <attribute
name="HttpSessionStorageStrategy">org.jboss.jetty.session.ClusteredStore</attribute>

    <!-- ================================================================ -->
    <!-- Options for synchronizing distributed sessions:                  -->
    <!--     never/idle/request/<num-seconds>                             -->
    <!-- ================================================================ -->

    <attribute name="HttpSessionSnapshotFrequency">never</attribute>

    <!-- ================================================================ -->
    <!-- Options for the notification of HttpSessionActivationListeners   -->
    <!-- around snapshotting are:                                         -->
    <!--    neither                                                       -->
    <!--    activate                                                      -->
    <!--    passivate                                                     -->
    <!--    both                                                          -->
    <!-- ================================================================ -->

    <attribute name="HttpSessionSnapshotNotificationPolicy">neither</attribute>

    <!-- ================================================================ -->
    <!-- If you require JAAS authentication, configure the name of the    -->
    <!-- attribute in which you expect to find the JAAS active subject:   -->
    <!--                                                                  -->
    <!-- Commenting out this configuration will disable JAAS support      -->
    <!-- ================================================================ -->

    <attribute name="SubjectAttributeName">j_subject</attribute>
```

```
  </mbean>
  <!-- ==================================================================== -->
  <!-- ==================================================================== -->

</server>
```

Unpacking wars on deployment

By default, Jetty will unpack your war as it is deployed. This is because JSP compilers typically can only compile unpacked classes. To change this behaviour, set the following property:

```
      <attribute name="UnpackWars">false</attribute>
```

Classloading behaviour

By default, Jetty follows the Java 2 specification for class loading. That is, when loading a class, Jetty first delegates to the webapp's parent class loader. This should be the norm within JBoss to take advantage of the unified class loading mechanism. However, it is possible to force Jetty to follow the Servlet 2.3 class loading specification, whereby Jetty first tries the webapp's own loader when loading "non-system" classes. If you are sure you need this kind of behaviour, set the following:

```
      <attribute name="Java2ClassLoadingCompliance">false</attribute>
```

Changing the default HTTP listener port

By default, Jetty listens on port 8080. To change this, modify the `Port` property of the `addListener` element:

```
       <Set name="Port"><SystemProperty name="jetty.port" default="9090"/></Set>
```

Changing other HTTP listener port attributes

The `jboss-service.xml` file specifies several extra attributes for the operation of Jetty which you may find useful to customise to your environment:

13.1 Listener port attributes

```
       <Call name="addListener">
            <Arg>
             <New class="org.mortbay.http.SocketListener">
              <Set name="Port"><SystemProperty name="jetty.port"
default="8080"/></Set>
              <Set name="MinThreads">5</Set>
              <Set name="MaxThreads">255</Set>
              <Set name="MaxIdleTimeMs">30000</Set>
              <Set name="MaxReadTimeMs">10000</Set>
              <Set name="MaxStopTimeMs">5000</Set>
              <Set name="LowResourcePersistTimeMs">5000</Set>
```

```
          </New>
        </Arg>
      </Call>
```

Attribute	**Description**
`MinThreads`	The minimum number of threads allowed.
`MaxThreads`	The maximum number of threads allowed.
`MaxIdleTimeMs`	Time in MS that a thread can be idle before it may expire.
`MaxReadTimeMs`	The maximum time in milliseconds that a read can be idle.
`MaxStopTimeMs`	Time in MS that a thread is allowed to run when stopping.
`LowResourcePersistTimeMs`	Time in ms to persist idle connections if low on resources.
`IntegralPort`	Port to redirect to for integral connections specified in a security constraint.
`IntegralScheme`	Protocol to use for integral redirections.
`ConfidentialPort`	Port to redirect to for confidential connections. 0 if not supported.
`ConfidentialScheme`	Protocol to use for confidential redirections.
`LingerTimeSecs`	The maximum time in seconds that a connection lingers during close handshaking.

Using SSL

The `jboss-service.xml` file includes a commented out example of how to set up Jetty for SSL:

```
        <!-- ======================================================= -->
        <!-- Uncomment and set at least the Keystore, Password and   -->
        <!-- KeyPassword fields to configure an SSL listener         -->
        <!-- ======================================================= -->
        <!--
           <Call name="addListener">
             <Arg>
               <New class="org.mortbay.http.SunJsseListener">
                 <Set name="Port">8443</Set>
                 <Set name="MinThreads">5</Set>
                 <Set name="MaxThreads">255</Set>
                 <Set name="MaxIdleTimeMs">30000</Set>
                 <Set name="MaxReadTimeMs">10000</Set>
                 <Set name="MaxStopTimeMs">5000</Set>
                 <Set name="LowResourcePersistTimeMs">2000</Set>
                 <Set name="Keystore"><SystemProperty name="jetty.home"
default="."/>/etc/demokeystore</Set>
                 <Set name="Password">dummy</Set>
                 <Set name="KeyPassword">dummy</Set>
               </New>
```

```
          </Arg>
        </Call>
      -->
```

You will also find some useful tips on the Jetty website at **http://jetty.mortbay.org/jetty/doc/SslListener.html**

Using JAAS

JAAS support is configurable across the JettyService instance via specifying the JAAS name of the attribute in which the active subject is transported:

```
<attribute name="SubjectAttributeName">j_subject</attribute>
```

Using Distributed HttpSessions

What distribution means

An `HttpSession` is an object used in a webapp to store conversational state between requests. It is configured in the webapp by specifying 'distributable' in it's `WEB-INF/web.xml`.

The J2EE specification requires that a 'distributable' app may be 'migrated' between nodes of a cluster - i.e. taken down on one node and brought up on another. Extant `HttpSessions` must continue to be available to the new webapp instance. Many appservers extend this functionality from simply allowing migration to providing failover i.e. if a webapp is not undeployed from it's node cleanly (eg the node crashes, hangs, becomes overloaded) it's `HttpSessions` are still made available to other instances of the same webapp within the cluster.

This extension is problematic since J2EE requires that on being undeployed, a distributed webapp should notify `HttpSession` attributes implementing `HttpSessionActivationListener` before passivating/distributing them. When the webapp has been re-deployed and it re-activates an `HttpSession`, the same attributes must be notified again. If, because of the reuse of this functionality to provide fail-over, attributes do not receive passivation events on one node before receipt of activation events on another, an asymmetry - which would not happen on a fully compliant appserver - occurs.

The Jetty integration allows the user to specify whether this extended behaviour (called 'snapshotting') is required and, if so, exactly what combination of events attributes should expect.

Configuring it

In order to use distributed http sessions, you need to perform the following series of steps:

1.Copy the `jbossha-httpsession.sar` (usually found in `$JBOSS_HOME/cluster/output/lib`) into the deploy director.

2.Edit the `jetty-plugin.sar/META-INF/jboss-service.xml` file:

1.Ensure the following property is set (as it is by default):

```
<attribute name="HttpSessionStorageStrategy">
      org.jboss.jetty.session.ClusteredStore
</attribute>
```

1.Set the snapshot frequency which affects the synchronization of distributed sessions:

```
<attribute name="HttpSessionSnapshotFrequency">never</attribute>
```

Options for the value are:

- never
- idle
- request
- <number of seconds>

3.Set the snapshot notification policy, which will affect when `HttpSessionActivationListeners` are notified:

```
<attribute name="HttpSessionSnapshotNotificationPolicy">neither</attribute>
```

Options for the value are:

- never
- activate
- passivate
- both

Other Jetty Configuration Tips

Deploying a war to context '/'

Deploying a webapp called `foo.war` will result in it being deployed at context `/foo`. To deploy it instead to the root context, choose one of the following mechanisms:

0.The standard J2EE way: wrap your `.war` in an `.ear` and in the `.ear`'s `META-INF/application.xml` you can specify the required context.

1. The proprietary JBoss extension: put a `jboss-web.xml` into your .war's WEB-INF directory and specify the context root in that.

2. Tomcat style: call the file ROOT.war and deploy it.

Using virtual hosts

This is supported as of JBoss2.4.5 via a proprietary extension mechanism.

To define a virtual host, add a line of the following form to your webapp's `WEB-INF/jboss-web.xml` file (and set up your DNS to route requests for this hostname):

```
<virtual-host>myvirtualhost</virtual-host>
```

.You can also specify a context path in the `WEB-INF/jboss-web.xml` file like so:

```
<context-root>/mycontextpath</context-root>
```

You should be careful as a context path specification in a `META-INF/application.xml` file will take precedence over the `WEB-INF/jboss-web.xml` specification.

Running on port 80

As port 80 is a privileged port, it is usually better to set up a mapping from it to a non-privileged port (such as 8080) where the HTTP server is running. The set-up required is operating system specific. For a how-to for Unix systems, see http://jetty.mortbay.org/jetty/doc/User80.html.

Running with Apache front-ending Jetty

It is not necessary to configure Apache to use Jetty, as Jetty is a fully featured HTTP server. However, if you have a special requirement for Apache, you can layer it in front of Jetty. Instructions for doing this can be found at http://jetty.mortbay.com/jetty/doc/JettyWithApache.html

Configuring Tomcat

In this section we'll discuss configuration issues specific to the JBoss/Tomcat-4.x integration bundle. The Tomcat-4.x release, which is also known by the name Catalina, is the latest Apache Java servlet container. It supports the Servlet 2.3 and JSP 1.2 specifications. The JBoss/Tomcat integration layer is controlled by the JBoss MBean service configuration. The MBean used to embed the Tomcat-4.x series of web containers is the org.jboss.web.catalina.EmbeddedCatalinaServiceSX service, and it is a subclass of the AbstractWebContainer class. Its configurable attributes include:

- **CatalinaHome**, sets the value to use for the catalina.home System property. This is used to . If not specified this will be determined based on the location of the jar containing the org.apache.catalina.startup.Embedded class assuming a standard catalina distribution structure.

- **CatalinaBase**, sets the value to use for the catalina.base System property. This is used to resolve relative paths. If not specified the CatalinaHome attribute value will be used.

- **Java2ClassLoadingCompliance**, enables the standard Java2 parent delegation class loading model rather than the servlet 2.3 load from war first model. This is true by default as loading from wars that include client jars with classes used by EJBs causes class loading conflicts. If you enable the servlet 2.3 class loading model by setting this flag to false, you will need to organize your deployment package to avoid duplicate classes in the deployment.

- **Config**, an attribute that provides support for extended configuration using constructs from the standard Tomcat server.xml file to specify additional connectors, and so on. Note that this is the only mechanism for configuring the embedded Tomcat servlet container as none of the Tomcat configuration files such as the conf/server.xml file are used. An outline of the configuration DTD that is currently supported is given in Figure 13-1, and the elements are described in the following section.

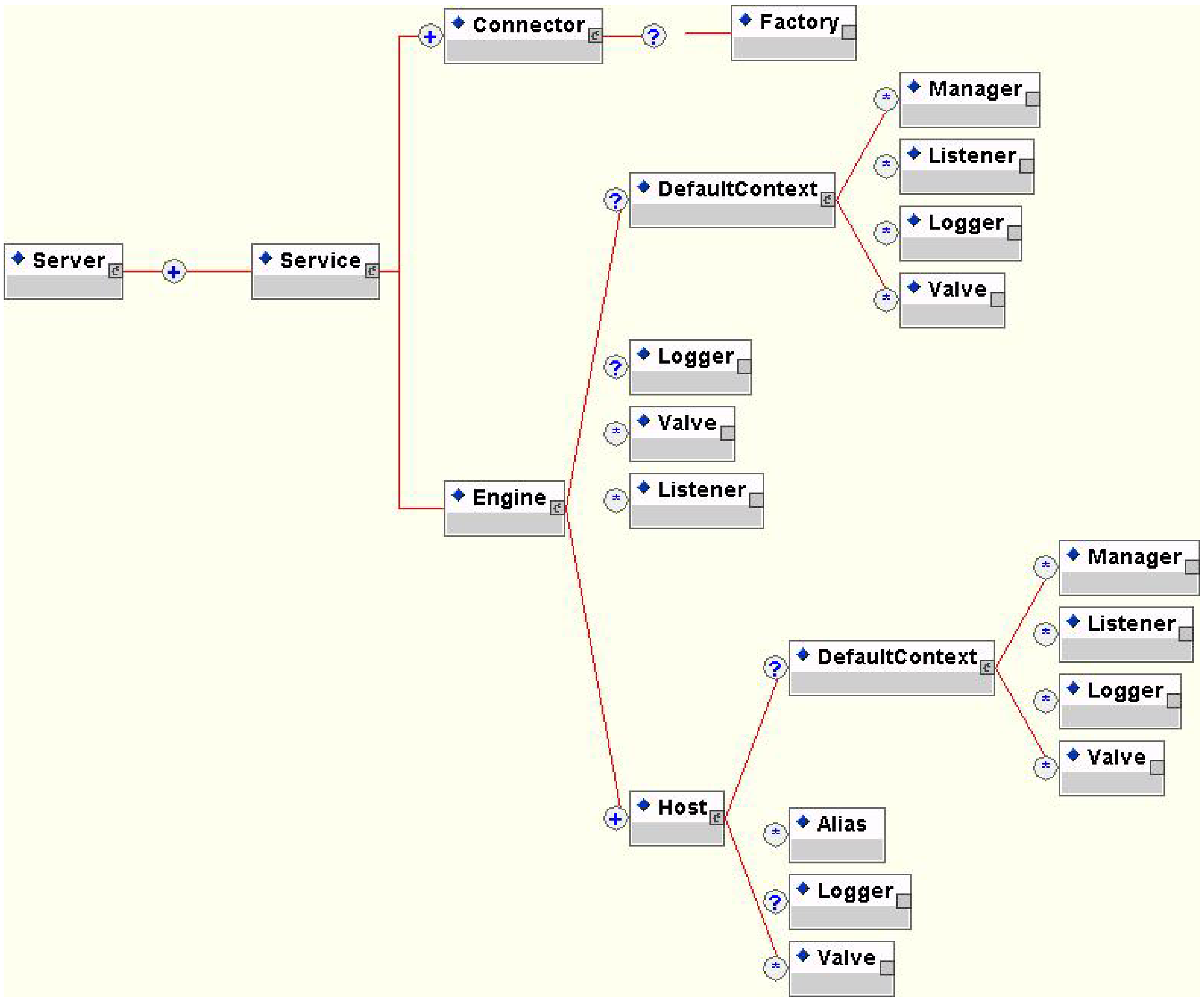


Figure 13-1, An overview of the Tomcat-4.0.3 configuration DTD supported by the EmbeddedCatalinaServiceSX Config attribute.

Using SSL with the JBoss/Tomcat bundle

To configure SSL for use with Tomcat you need to define a org.jboss.security.SecurityDomain implementation that JSSE should obtain the SSL KeyStore from. This requires establishing a SecurityDomain using the org.jboss.security.plugins.JaasSecurityDomain MBean. The SecurityDomain is then specified to an SSL socket factory in a Tomcat HTTPConnector configuration. A jboss.jcml configuration file fragment that illustrates the setup of SSL using this approach is given in Listing 13-1.

Listing 13-1, the JaasSecurityDoman and EmbeddedCatalinaSX MBean configurations for setting up Tomcat-4.x to use SSL as its primary connector protocol.

```
<server>
...
 <!-- The SSL domain setup -->
 <mbean code="org.jboss.security.plugins.JaasSecurityDomain"
     name="Security:service=JaasSecurityDomain,domain=RMI+SSL">
   <constructor>
      <arg type="java.lang.String" value="RMI+SSL"/>
   </constructor>
   <attribute name="KeyStoreURL">chap8.keystore</attribute>
   <attribute name="KeyStorePass">rmi+ssl</attribute>
 </mbean>
...
  <!-- The embedded Tomcat-4.x(Catalina) service configuration -->
  <mbean code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
    name="DefaultDomain:service=EmbeddedCatalinaSX">
    <attribute name="Config">
      <Server>
        <Service name = "JBoss-Tomcat">
          <Engine name="MainEngine" defaultHost="localhost">
            <Logger className = "org.jboss.web.catalina.Log4jLogger"
              verbosityLevel = "trace" category = "org.jboss.web.localhost.Engine"/>
              <Host name="localhost">
                <Valve className = "org.apache.catalina.valves.AccessLogValve"
                   prefix = "localhost_access" suffix = ".log"
                   pattern = "common" directory = "../server/default/log" />
                <DefaultContext cookies = "true" crossContext = "true" override = "true" />
             </Host>
           </Engine>

           <!-- SSL/TLS Connector configuration -->
           <Connector className = "org.apache.catalina.connector.http.HttpConnector"
             port = "443" scheme = "https" secure = "true">
             <Factory className = "org.jboss.web.catalina.security.SSLServerSocketFactory"
               securityDomainName = "java:/jaas/RMI+SSL" clientAuth = "false"
               protocol = "TLS"/>
           </Connector>
         </Service>
      </Server>
    </attribute>
  </mbean>
</server>
```

A jboss.jcml configuration file fragment that illustrates such the setup of both SSL and non-SSL connectors is given in Listing 13-2.

Listing 13-2, the JaasSecurityDoman and EmbeddedCatalinaSX MBean configurations for setting up Tomcat-4.x to use both non-SSL and SSL enabled HTTP connectors.

```
<server>
```

```
...
 <!-- The SSL domain setup -->
 <mbean code="org.jboss.security.plugins.JaasSecurityDomain"
     name="Security:name=JaasSecurityDomain,domain=RMI+SSL">
   <constructor>
      <arg type="java.lang.String" value="RMI+SSL"/>
   </constructor>
   <attribute name="SecurityManagerService">jboss.security:name=JaasSecurityManager
   </attribute>
   <attribute name="KeyStoreURL">chap8.keystore</attribute>
   <attribute name="KeyStorePass">rmi+ssl</attribute>
 </mbean>
...
  <!-- The embedded Tomcat-4.x(Catalina) service configuration -->
  <mbean code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
    name="DefaultDomain:service=EmbeddedCatalinaSX">
    <attribute name="Config">
      <Server>
        <Service name = "JBoss-Tomcat">
          <Engine name="MainEngine" defaultHost="localhost">
            <Logger className = "org.jboss.web.catalina.Log4jLogger"
              verbosityLevel = "trace" category = "org.jboss.web.localhost.Engine"/>
              <Host name="localhost">
                <Valve className = "org.apache.catalina.valves.AccessLogValve"
                   prefix = "localhost_access" suffix = ".log"
                   pattern = "common" directory = " ../server/default/log" />
                <DefaultContext cookies = "true" crossContext = "true" override = "true" />
             </Host>
           </Engine>

           <!-- HTTP Connector configuration -->
           <Connector className = "org.apache.catalina.connector.http.HttpConnector"
             port = "8080" redirectPort = "443"/>
           <!-- SSL/TLS Connector configuration -->
           <Connector className = "org.apache.catalina.connector.http.HttpConnector"
             port = "443" scheme = "https" secure = "true">
             <Factory className = "org.jboss.web.catalina.security.SSLServerSocketFactory"
               securityDomainName = "java:/jaas/RMI+SSL" clientAuth = "false"
               protocol = "TLS"/>
           </Connector>
         </Service>
      </Server>
    </attribute>
  </mbean>
```

Setting up Virtual Hosts with the JBoss/Tomcat-4.x bundle

As of the 2.4.5 release, support for virtual hosts has been added to the servlet container layer. Virtual hosts allow you to group web applications according to the various DNS names by which the machine running JBoss is known. As an example, consider the jboss.jcml configuration fragment given in Listing 13-3. This configuration defines a default host named localhost and a second host named banshee.starkinternational.com. The banshee.starkinternational.com also has the aliases www.starkinternational.com associated with it.

Listing 13-3, An example virtual host configuration.

```
<!-- The embedded Tomcat-4.x(Catalina) service configuration -->
<mbean code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
  name="DefaultDomain:service=EmbeddedCatalinaSX">
  <attribute name="Config">
    <Server>
      <Service name = "JBoss-Tomcat">
        <Engine name="MainEngine" defaultHost="localhost">
          <Logger className = "org.jboss.web.catalina.Log4jLogger"
            verbosityLevel = "debug" category = "org.jboss.web.CatalinaEngine"/>
          <DefaultContext cookies = "true" crossContext = "true" override = "true" />
            <Host name="localhost">
              <Logger className = "org.jboss.web.catalina.Log4jLogger"
                verbosityLevel = "debug" category = "org.jboss.web.Host=localhost"/>
              <Valve className = "org.apache.catalina.valves.AccessLogValve"
                 prefix = "localhost_access" suffix = ".log"
                 pattern = "common" directory = " ../server/default/log" />
           </Host>
            <Host name="banshee.starkinternational.com">
              <Alias>www.starkinternational.com</Alias>
              <Logger className = "org.jboss.web.catalina.Log4jLogger"
                verbosityLevel = "debug" category = "org.jboss.web.Host=www"/>
              <Valve className = "org.apache.catalina.valves.AccessLogValve"
                 prefix = "www_access" suffix = ".log"
                 pattern = "common" directory = " ../server/default/log" />
           </Host>
        </Engine>

        <!-- A HTTP Connector on port 8080 -->
        <Connector className = "org.apache.catalina.connector.http.HttpConnector"
           port = "8080" minProcessors = "3" maxProcessors = "10" enableLookups = "true"
           acceptCount = "10" connectionTimeout = "60000"/>
      </Service>
    </Server>
  </attribute>
</mbean>
```

When a WAR is deployed, it will be by default associated with the virtual host whose name matches the defaultHost attribute of the containing Engine. To deploy a WAR to a specific virtual host you need to use the jboss-web.xml descriptor and the virtual-host element. For example, to deploy a WAR to the virtual host www.starkinternational.com virtual host alias, the following jboss-web.xml descriptor would be need to be included in the WAR WEB-INF directory. This demonstrates that an alias of the virtual host can be used in addition to the Host name attribute value.

Listing 13-4, An example jboss-web.xml descriptor for deploying a WAR to the www.starkinternational.com virtual host

```
<jboss-web>
   <context-root>/</context-root>
   <virtual-host>www.starkinternational.com</virtual-host>
</jboss-web>
```

When such a WAR is deployed, the server console shows that the WAR is in fact deployed to the www.starkinternational.com virtual host as seen by the “Host=www” category name in the log statements.

Using Apache with the JBoss/Tomcat-4.x bundle

To enable the use of Apache as a front-end web server that delegates servlet requests to a JBoss/Tomcat bundle, you need to configure an appropriate connector in the EmbeddedCatalinaSX MBean definition. For example, to configure the use of the Ajpv13 protocol connector with the Apache mod_jk module, you would use a configuration like that given in Listing 13-5.

Listing 13-5, an example EmbeddedCatalinaSX MBean configuration that supports integration with Apache using the Ajpv13 protocol connector.

```
<server>
  <!-- The embedded Tomcat-4.x(Catalina) service configuration -->
  <mbean code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
    name="DefaultDomain:service=EmbeddedCatalinaSX">
    <attribute name="Config">
      <Server>
        <Service name = "JBoss-Tomcat">
          <Engine name="MainEngine" defaultHost="localhost">
            <Logger className = "org.jboss.web.catalina.Log4jLogger"
              verbosityLevel = "trace" category = "org.jboss.web.localhost.Engine"/>
              <Host name="localhost">
                <Valve className = "org.apache.catalina.valves.AccessLogValve"
                   prefix = "localhost_access" suffix = ".log"
                   pattern = "common" directory = " ../server/default/log" />
                <DefaultContext cookies = "true" crossContext = "true" override = "true" />
             </Host>
           </Engine>

           <!-- AJP13 Connector configuration -->
          <Connector className="org.apache.ajp.tomcat4.Ajp13Connector"
              port="8009" minProcessors="5" maxProcessors="75"
              acceptCount="10" />
         </Service>
      </Server>
    </attribute>
  </mbean>
</server>
```

The configuration of the Apache side proceeds as it normally would as bundling Tomcat inside of JBoss does not affect the how Apache interacts with Tomcat. For example, a fragment of an httpd.conf configuration to test the Listing 13-5 setup with a WAR deployed with a context root of "/jbosstest" might look like:

```
...
LoadModule    jk_module    libexec/mod_jk.so
AddModule     mod_jk.c

<IfModule mod_jk.c>
   JkWorkersFile /tmp/workers.properties
   JkLogFile     /tmp/mod_jk.log
   JkLogLevel    debug
   JkMount       /jbosstest/* ajp13
</IfModule>
```

Other Apache to Tomcat configurations would follow the same pattern. All that would change it the Connector element definition that is placed into the EmbeddedCatalinaSX MBean configuration.

Chapter

14

14. Web Services

How to Make Your Java Logic Globally Interoperable by Christoph G. Jung, Peter Braswell and Frederik M. Brier

Web Services are self-contained pieces of code that have three distinguishing properties:

1. They communicate in an interoperable XML protocol, such as SOAP.
2. They describe themselves in an interoperable XML meta-format, such as WSDL.
3. They are able to federate globally through XML-based registry services, such as UDDI.

JBoss supports both the construction of Java-based Web Services as well as the integration of possibly non-Java based external Web Services through the JBoss.Net extension. A special focus is placed on patterns and tools to expose J2EE™-based logic, such as session beans and entity beans. A special focus is also placed on interoperability with Microsoft .Net™ as a prominent client platform. JBoss.Net bases on the Apache Axis implementation of the JAX-RPC API. This chapter provides some basic configuration details. For details about XML-serialization using Axis, please see the Axis User Guide. For details about particular features of JBoss.Net, such as the possibility to map CMP2.0 bean structures to typed .Net datasets and Macromedia Flash support, please see the full JBoss 3.0 documentation (which is still to be completed in this respect).

What is all this buzz?

What is XML and XML-Schema?

Come on, dude? You really missed that XML is the ASCII of the new millenium? Get real!

Example of XML document - CGJ

The XML Schema description language is a meta-language to describe and analyze the structure of well-formed XML documents according to particular schema types. Types can be declared, referred, imported, and even inherited. They are then instantiated by particular tags in a document, such as properly structured elements and attributes.

Example of XML-Schema document - CGJ

What is SOAP, WSDL and UDDI?

The Simple Object Access Protocol (SOAP) is an RPC-protocol that uses XML as the encoding language and that uses standard internet protocols, such as HTTP and SMTP as the transport medium.

Example of SOAP message - CGJ

For SOAP, there exists also corresponding meta-language that is called Web Service Description Language (WSDL). WSDL has a similar function than the Interface Definition Language has in Corba. It describes service interfaces (port types) in terms of the operations (request-messages and response-messages) that can be invoked. It describes particular service instances by binding a port-type to a particular transport endpoint.

Example of WSDL description - CGJ

The Universal Description Discovery Integration (UDDI) is a standardized interface to Web Service registries – a kind of global and platform neutral version of JNDI in which WSDL descriptions are hosted and can be searched. UDDI registries are themselves Web Services in that they support the SOAP protocol.

What are JAXM, JAX-RPC & JAXR?

We can distinguish two modes of XML/SOAP messaging. The simplest, but very unconvenient mode is to interact with the XML/DOM-documents representing method requests and responses immediately. It is suitable for untyped applications or special-purpose logic that is based on XML-processing anyway. There is an API under development, JAXM (Java Api for XML Messaging), which will support this mode.

Whenever we are not primarily interested in manipulating XML and have a strongly-typed environment, such as in J2EE™ logic, we need a more elaborated mode that is an extension to Remote Method Invocation (RMI). The corresponding extension API to the Java2™ platform is called JAX-RPC. In JAX-RPC, an XML-message is processed through a chain of so-called handlers before it is (de-)serialized into/from Java objects using a set of so-called type-mappings.

Type-mappings associate a Java class and a corresponding XML type by dedicated Serializer and Deserializer implementations. This is a very powerful, yet extensible pattern and you see that even SUN can learn from flaws they built-into their native serialization API.

JAXR is the upcoming API to interface XML registries, such as UDDI, from Java. It is the youngest of the presented API´s and yet rarely supported.

What is Axis?

Axis is an Open Source implementation of the JAX-RPC API from Apache. It realizes the basic concepts of handlers and type-mappings driven by a lightweight SAX-based processing engine. Axis comes with a dedicated deployment format called Web Service Deployment Descriptor (WSDD) which configures the runtime engine with new request/response flows (these are named chains of handlers that together perform a reasonable preprocessing), service providers (these are special handlers that construct the end-points of processing chains and that perform the actual logic of a Web Service call), and type-mappings. Axis comes with a ready-made http transport listener in the form of a Servlet that can be installed in any compliant web-container.

Example of WSDD description - CGJ

Configuring JBoss.Net

What is JBoss.Net

JBoss.Net is an integration module that fits the Axis JAX-RPC package optimally into the hot-deployment environment of JBoss:

- JBoss.Net provides a new archive format called Web Service aRchive (WSR) which bundles serialization code, web service logic and WSDD-descriptors.
- JBoss.Net comes with a set of suitable providers which interface the proper container-managed JMX & J2EE™ logic.
- JBoss.Net has the ability to bind JAX-RPC-enabled service stubs to external Web Services into the JNDI-tree and hence make them visible to all parts of the container and your application. Usually, such stubs have been generated from WSDL descriptions.
- JBoss.Net has the ability to automatically publish your Web Services to global UDDI registries. JBoss.Net will have a non-production-use UDDI-server implementation for test purposes.

- JBoss.Net comes with a set of suitable handlers which use the JBoss infrastructure to perform useful processing tasks, such as authentication, authorization, transaction management, classloading and the like.
- Finally, JBoss.Net contains special-purpose type-mappings and serialization hooks to tie particular Web Service clients, such as Macromedia Flash and Microsoft .Net applications, to the JBoss server. This is an ongoing effort.

Picture of the integration - CGJ

Installation - PB

JBoss.Net is packaged as a service archive file called `jboss-net.sar`. It deploys automatically with JBoss with a default configuration:

4. registers with the MainDeployer

5. installs the Servlet in the WebContainer

6. some basic type-mappings

Configuration - FB

The default configuration can be modified by editing the JBoss.Net configuration file found inside the sar as `jboss-net.sar/META-INF/jboss-service.xml`. To modify it, first unpack the sar, make your changes, repack it and copy it back to the deploy directory. JBoss will reload and restart JBoss.Net with it's new configuration. Alternatively, for non-permanent configuration changes, you can use the JMX Agent on port 8082.

The default JBoss.Net `jboss-service.xml` file looks like:

Basic Security

How to set the main security domain and shield the servlet.

Using Tomcat as the WebContainer

```
Change the dependency
```

Building A Pojo Web Service with JBoss.Net - FB

How to bundle wsdd. How to obtain wsdl.

Building An EJB-based Web Service with JBoss.Net - CGJ

Including how to use the xdoclet subtask

External Web Services and UDDI - PB

Appendix
A

15. Appendix A

About The JBoss Group

JBoss Group LLC, is an Atlanta-based professional services company, created by Marc Fleury, founder and lead developer of the JBoss J2EE-based Open Source web application server. JBoss Group brings together core JBoss developers to provide services such as training, support and consulting, as well as management of the JBoss software and services affiliate programs. These commercial activities subsidize the development of the free core JBoss server. For additional information on the JBoss Group see the JBoss site http://www.jboss.org/jbossgroup/services.jsp.

16. Appendix B

Introducation to ANT Build System

Before the days of ANT therefore was only "make" available to build your application with dependencies to avoid recompilation every time you needed a new build. But "make" is hard to write and maintain, is most of the time platform dependent and does not perform well in Java because "make" calls the Java compiler for each file instead all at once.

So on a flight David Duncanson (father of Tomcat) was tired of all this hassle and decided to write a Java based build tool to ease the development of Tomcat. As soon as Tomcat was put on the CVS repository of `http://jakarta.apache.org` ANT became a success maybe even more than Tomcat.

ANT (`http://jakarta.apache.org/ant`) is a pure Java build system taking one or several XML files as input telling ANT what and how to build your application. This build file (by default named "build.xml") contains three parts:

- Definition of properties which can (if setup) be overwritten by another file
- List of build-in tasks to compile, jar, copy etc.
- List of targets and the dependencies to none, one or more other targets

ANT starts with the specified or the main target (if not specified) and creates graph of the dependent targets (directly or indirectly dependent of the specified target). Then the targets are executed in reverse order (from the farthes away to the specified target). Finally each dependent target is executed once and only once.

Normally it is the best practice to adjust a build file instead of writing your own. But in order to write a build file from scratch create the project (root element) first, then the main target, define its dependencies, create these targets and repeat the last two steps as necessary. Try to avoid static text (paths, values, etc.) and use ANT properties instead. These properties can be changed on a central place (where they are defined) and can be overwritten from a properties file (if enabled in the build file). This properties file enables clients to change settings without changing files in a CVS repository etc.

Appendix C

17. Appendix C

Introduction to XDoclet

Have you ever thought that creating and maintaining the Home and Remote interface, the deployment descriptors, Value Objects etc. is a pain. Also did Rickard Oeberg, architect of JBoss 2, and created EJBDoclet that was renamed afterwards to XDoclet because it does generated EJB files but also web, JMX etc files.

XDoclet, as the name implies, is a special JavaDoc doclet implementation and reads JavaDoc comments and uses this and class information to generated other Java source code, deployment descriptors and other files. The basic idea behind is that all the redundant information in these additional classes are generated to speed up project development and shorten testing because you don't have to worry about Java classes out of sync or wrong references in or between deployment descriptors. Finally and maybe the most important fact is that all the information are in **one, central** Java class. No jumping around to figure out the JNDI name, the datasource name, resource name etc. and also only one file is added to a version control tool like CVS because all the other files are generated every time necessary.

XDoclet is heavily incorporate into ANT build system (see Appendix B) and therefore you must run XDoclet as an ANT task. To use XDoclet you have to:

- Create a "Taskdef" in the ANT build file before you can use XDoclet
- Write the XDoclet task, specify the task attributes and the file set (list of Java classes inspected by XDoclet)
- Write the XDoclet subtask and specify its attributes

Finally a list of advanced features of XDoclet:

- Able to merge in pieces of code, XML snippets etc. at specific merge points
- Able to change the layout files XDoclet uses to generate the files and use them without changing XDoclet
- A GUI tool to write XDoclet definitions (XDocletGUI)

Appendix

D

18. Appendix D

Title D

Bla Bla

Appendix
E

19. Appendix E

Title E

Bla Bla

20. Appendix F

Title F

Bla Bla

21. Index