Deploying with Tomcat

This article was written by Andrew Lindesay (http://www.lindesay.co.nz) around May 2005. It first appeared as LaTeX PDF and has been transcribed into this Wiki. You use the information contained in this document at your own risk. Please contact the author if you feel there may have been an error in the conversion to Wiki markup.

More information on tomcat deployment using Eclipse/WOLips, including information on how to deploy Project Wonder applications, is available here

Contents

Abstract

From WebObjects 5.2, it has been possible to derive a build product from a WebObjects application project that can be deployed into a J2EE servlet container. This article shows how it is possible to
deploy a WebObjects 5.2 application into a Tomcat environment and achieve a very similar topology to the "native" WebObjects deploy.

Introduction

This document was originally written assuming a Tomcat 5 deployment, but after some difficulties with web services and AXIS, I have modified this document to also cater for a Tomcat 3 deployment. This document covers both circumstances.

Assumptions

This article assumes the following:

- WebObjects 5.2 (likely to work fine with newer versions)
- Java 1.4
- Latest Tomcat release of version 5 (5.5.12 at the time of writing) or 3 (3.3.2 at the time of writing)
- Some sort of UNIX deployment.
- The reader has some conception of the concepts behind servlet technology.
- The reader is familiar with a standard WebObjects deployment topology which will be referred to as *wotaskd* deployment.

For the purposes of this document, it is assumed that Tomcat has been installed at a directory on the local disc called `$TOMCATDIR`. It is also assumed that you will have another directory with the files required to configure and run an instance called `$INSTDIR`. It is assumed also that you will have a directory called `$JKDIR` with the Tomcat apache adaptor in it. In this article, some configuration files require the paths to be shown and these are tabulated below.

`$TOMCATDIR`	`/opt/tomcat`
`$INSTDIR`	`/opt/fooapp`
`$JKDIR`	`/opt/modjk`

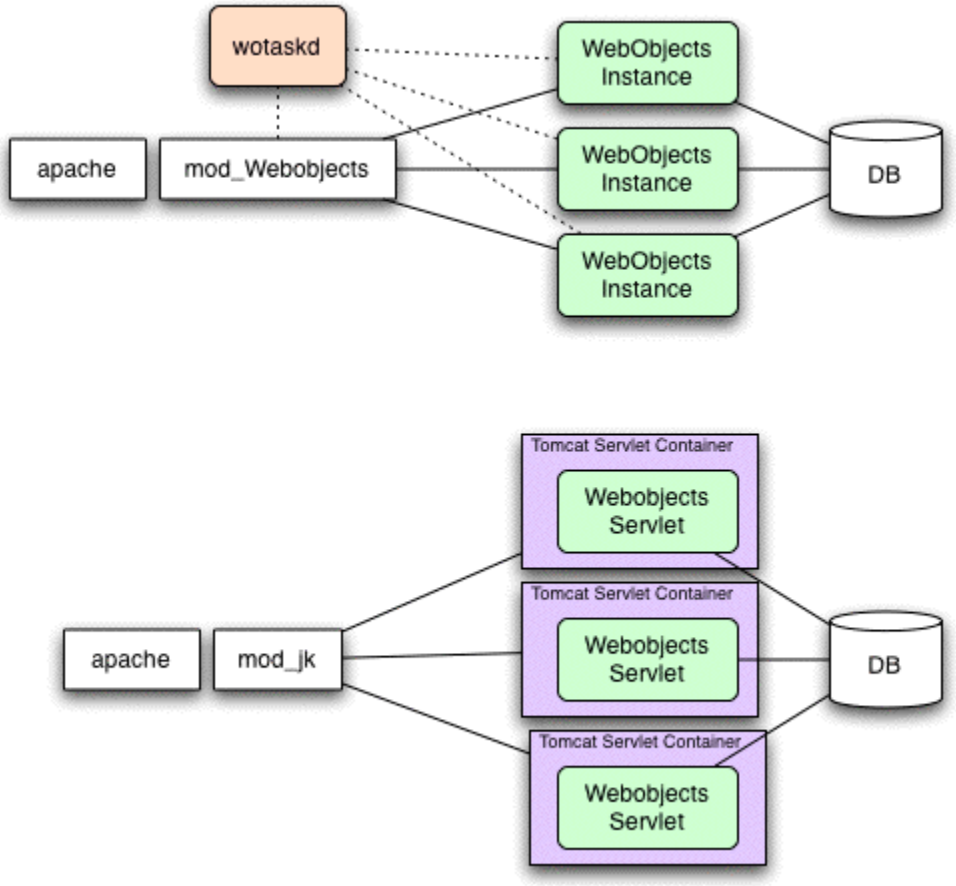
In reality, these directories could be located anywhere.

Objective

The objective of this article is to show that a WebObjects application can be deployed into a servlet container and keep some of the desirable attributes of a WebObjects deployment topology. Some of these traits are itemised below.

- Clustering over a number of hardware nodes to prevent system downtime from a single hardware failure incident.
- Clustering over a number of virtual machine *instances* on each hardware node to avoid downtime from a single software failure incident.
- Ability to make efficient use of lower cost server hardware rather than encouraging use of large, expensive servers.
- Make maximum use of memory availabile in each virtual machine as cache to minimise database traffic and lower stress on the database server.
- Ability to make sessions "sticky" to a given virtual machine *instance*, whilst being multiplexed through a single web server front-end adaptor.
- Ability to load-balance requests to instances which are operational.

A standard WebObjects deployment topology is shown in the figure below alongside what is to be achieved with Tomcat. A typical J2EE deployment may have a different topology from this.

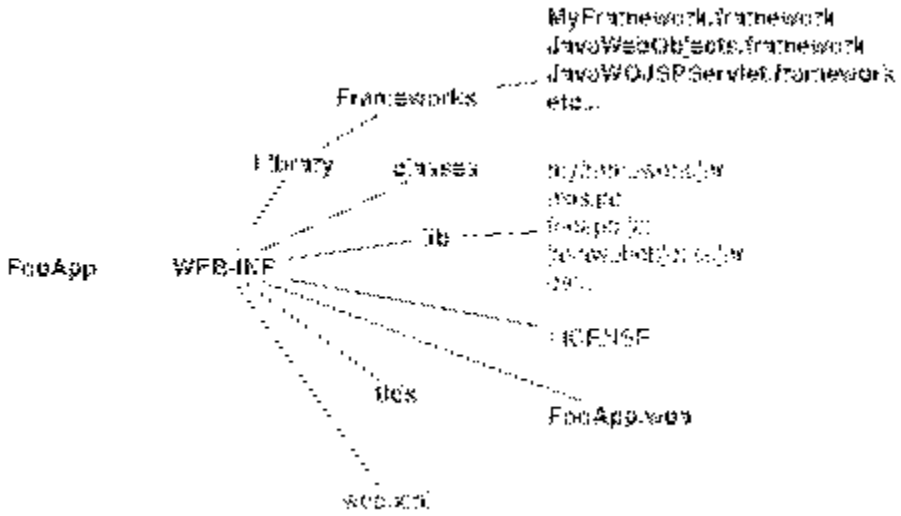


Servlet Build Products

Creating a servlet build product from a WebObjects application project is covered in some depth by documentation that is supplied by Apple for WebObjects. This is not going to be repeated here, but here is a brief overview of the process.

- Include the `JavaWOJSPServlet.framework` framework into your project.
- In the build settings, set the `SERVLET_SINGLE_DIR_DEPLOY` value to `YES` to create the most trouble-free form of deployment servlet.
- Edit the `SERVLET_DEPLOY_LICENSE` to contain your valid deployment license key if you need one for the version of WebObjects you are using.
- Edit the `SERVLET_WEBAPPS_DIR` to point to `$INSTDIR/webapps/` or some place where you want the build product to go.

Now when you choose a *Deployment* build, you will also get the servlet assembled. The end result is a directory structure similar to that shown in the figure below.



Removing the DOCTYPE from .plist Files

Many property-list files (often called plist files) have a document type at the top. This can refer to files on a MacOS-X machine or to files on Apple servers. In either case this can cause problems with deployments which are not on MacOS-X servers. The following script can be run with the argument of the `WEB-INF` folder to remove these. The WebObjects application runs fine without this information in the plist files. This script can easily be incorporated one way or another as a step in the build process for your WebObjects project.

```
# [apl 3.may.2006]
# This will remove any DOCTYPE's from the top of plists so that
# they do not attempt to validate the DTD which is either
# extracted from /System or the internet over HTTP.

if [ -z $1 ]; then
  echo "syntax: stripdocype.sh <directory>"
  exit 1
fi

for PLISTFILE in `find $1 -name *.plist`
do
  sed \
    '/<!DOCTYPE [Deploying with Tomcat^>]*>/s/.*//' \
    $PLISTFILE \
    > /tmp/remove-plist-temp

    cp /tmp/remove-plist-temp $PLISTFILE
done
```

Application Configuration with Tomcat

A WebObjects project with servlet support has a file called `web.xml.template` in it. By default, this is located in `/Resources/Servlet Resources /WEB-INF` in your project. The `web.xml.template` file is used as a template for creating `web.xml` which is also known as the `servlet deployment descriptor`. This deployment descriptor is used as the means of communicating settings to the application as well as the servlet container in which the application runs. This section covers some common changes to that file as well as a discussion around a means of general configuration of the application when it is running inside a servlet container.

Data Source for Model Database Configuration

Some WebObjects engineers use the `setConnectionDictionary(...)` method on a model to set the JDBC database connection parameters for the model. However, the servlet container has it's own `data source` mechanism for supplying database information which will override any connection dictionary information which is set into the model. If you don't want this to happen, and you want your `setConnectionDictionary(...)` to take effect, comment out the `resource-ref` item with the title `jdbc/DefaultDataSource` in the `web.xml.template` file.

If you have more than one data source Resource defined in either your WEB-INF/web.xml or in your META-INF/Context.xml file, WebObjects will complain about there being more than one configuration with this error:

`An exception occurred while trying to open a channel: com.webobjects.jdbcadaptor.JDBCAdaptorException: Found multiple data sources. Please map the EOModels to a data source explicitly!`

You must set the name parameter of each data source Resource in the web.xml/Context.xml with the same name as your eomodel file. For example:

EOModel name: **MyModel**.eomodel

data source Resource definition:

```
  <Resource
        auth="Container"
        description="MyApp Data Source"
HERE -> name="MyModel"
        type="javax.sql.DataSource"
        driverClassName="com.microsoft.jdbc.sqlserver.SQLServerDriver"
        url="jdbc:microsoft:sqlserver://serveraddress:1433;databaseName=TEST"
        username="user"
        password="pass"
        maxActive="4"
        maxWait="5000"
        maxIdle="10"
  />
```

Serving WebServerResources

To stop the web server resources (images, CSS files and other static data) from being served out of the java environment, you need to configure the `context-param` with the name `WOAppMode` to be `Deployment` in the `web.xml.template` file.

Application Specific Configuration

This area of configuration covers items such as the following ficticious examples;

1. Email address of person to contact when system fails.
2. Frequency of polling some resource.
3. Optional connection information for databases.
4. GST rate for New Zealand.

In other words, these are application-specific configuration values. One way to configure your application specific parameters in a servlet container is to load your config into `env-entry`-s in your `web.xml` file. Here is such an example of one such entry;

```
<env-entry>
  <env-entry-name>foo/nz.co.foo.FooAppMailFrom</env-entry-name>
  <env-entry-value>bar@foo.co.nz</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

You can retrieve these value inside the application with some code as shown below. See the LEConfig class from LEWOStuff, the WebObjects framework from Lindesay Electric for an example. LEWOStuff also comes with a tool to help load standard java properties files into the servlet deployment descriptor.

```
import javax.naming.*;

// ...later in the same class...

Object valueO = null;

try
{
  InitialContext context = new InitialContext();
  valueO = context.lookup("java:comp/env/foo/nz.co.foo.FooAppMailFrom");
}
catch(javax.naming.NamingException ne)
{ /* handle gracefully */ }
```

It is probably easiest to apply these settings in some automated fashion to the `web.xml` file as part of a further automated build or deploy process.

Application Binary

Put your application servlet build product at `$INSTDIR` such that the following path exists.

```
$INSTDIR/webapps/FooApp/WEB-INF
```

Tomcat Server Configuration Files

You need to create a tomcat configuration file for each of the instances that you would like to have. The pattern is followed here of having an instance number proceeded by the lower case letter "i". Put the first server configuration file at the following location.

```
$INSTDIR/server_i1.xml
```

Here is an example of how this file might look. There is no coverage of the individual settings here as the reader is expected to review the tomcat documentation to discover the specific meanings of these settings.

Tomcat 5

```
<Server port="7071" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8081" maxHttpHeaderSize="8192"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" acceptCount="100"
               connectionTimeout="20000" disableUploadTimeout="true" />

    <Connector port="9091" enableLookups="false" protocol="AJP/1.3" />

    <Engine name="i1" defaultHost="appserver1.foo.co.nz" jvmRoute="i1">
      <Host name="appserver1.foo.co.nz"
        appBase="/home/fooapp/webapps"
        unpackWARs="true" autoDeploy="false"
        xmlValidation="false" xmlNamespaceAware="false">
        <Context cookies="false" docBase="FooApp"
          path="FooApp" reloadable="false">
          <Manager distributable="false" />
        </Context>
      </Host>
    </Engine>

  </Service>
</Server>
```

Tomcat 3

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Server>
  <ContextManager workDir="work">
    <LoaderInterceptor11 useApplicationLoader="true"/>

    <AutoDeploy source="modules" target="modules" redeploy="true"/>
    <AutoWebApp dir="modules" host="DEFAULT" trusted="true"/>
    <AutoWebApp dir="/home/fooapp/webapps" trusted="true" reloadable="false"/>

    <SimpleMapper1/>

    <SessionExpirer checkInterval="60"/>
    <SessionIdGenerator randomClass="java.security.SecureRandom"/>

    <WebXmlReader validate="false"/>
    <ErrorHandler showDebugInfo="true"/>

    <Jdk12Interceptor/>
    <LoadOnStartupInterceptor/>
    <Servlet22Interceptor/>

    <SessionId cookiesFirst="false" noCookies="true"/>
    <SimpleSessionStore maxActiveSessions="256"/>

    <Http10Connector port="8081" secure="false"/>
    <Ajp13Connector port="9091" tomcatAuthentication="false" shutdownEnable="true"/>
  </ContextManager>
</Server>
```

Assuming that there will be three instances in this example deploy, this entire file should be replicated and modified twice for the other two instances. For the other instances' server configuration files, change the "i1" (Tomcat 5 only) by modifying the numerical component and change the port numbers by making the last digit the instance number. For example, the ports `7071`, `8081` and `9091` are used here. For "i2", use `7072`, `8082` and `9092`.

You should have three files now present called `server_i1.xml`, `server_i2.xml` and `server_i3.xml` in the directory `$INSTDIR`.

Startup An Instance

To startup an instance, issue a command as follows. You should issue this command for each of the server configuration files. The `$JAVA_HOME` shell environment variable should have been setup correctly before launching an instance.

Tomcat 5	`$TOMCATDIR/bin/startup.sh -config $INSTDIR/server_i1.xml`
Tomcat 3	`$TOMCATDIR/bin/startup -config $INSTDIR/server_i1.xml -home $TOMCATDIR`

Tomcat 3 Environment Variables

To pass java environment variables to your application, set the `TOMCAT_OPTS` shell environment variable before starting up the Tomcat 3 environment. An example of this would be as follows;

```
TOMCAT_OPTS=-Dabc=xyz
export TOMCAT_OPTS
```

Check Availability

You can now see if your instance is up using the following URL.

```
http://appserver1.foo.co.nz:8081/FooApp/WebObjects/FooApp.woa
```

Note that both the AJP (This is the protocol between the apache adaptor and the individual Tomcat instances.) and the regular HTTP engines are accessing the same running application. This means that a test straight onto the application via HTTP is actually testing the application that is being accessed via AJP. This behaviour provides an opportunity to be able to monitor specific instances of Tomcat over direct HTTP.

Check each of your instances.

Problems

Under Tomcat 5, check the log file at `$TOMCATHOME/logs/catalina.out` if you are unable to access your instance.

Shutdown An Instance

To shutdown an instance, issue a command as follows.

Tomcat 5

```
$TOMCATDIR/bin/shutdown.sh -config $INSTDIR/server_i1.xml
```

It is also possible to use the UNIX `telnet` command to connect to the port described in the server configuration file's `Server` tag and type the word supplied in the `shutdown` attribute to take out the Tomcat instance.

Tomcat 3

```
TOMCATDIR/bin/shutdown -ajp13 -port 9091
```

Apache Adaptor Setup

A final deployment usually involves apache feeding inbound requests to instances and handling situations where an instance has gone down and balancing load over the instances that are currently running. This job is undertaken by `mod_jk` which is a module for apache written by the Tomcat group. This module communicates with the tomcat instances using a protocol called `AJP`. This protocol carries all the information required to check instances are operational as well as relaying requests to the instances. It is assumed that the configuration as well as binaries for the `mod_jk` apache module will be located at `$JKDIR`.

Compiling and Installing

Instructions for downloading and installing `mod_jk` can be obtained from the Tomcat website. The `mod_jk.so` binary should be located at the following path.

```
$JKDIR/mod_jk.so
```

Add a line to your system's apache httpd configuration file like this:

```
Include /opt/modjk/apache.conf
```

Edit this file to look like this:

```
LoadModule jk_module /opt/modjk/mod_jk.so
AddModule mod_jk.c

JkLogFile /opt/modjk/mod_jk.log
JkLogLevel info

JkWorkerProperty worker.list=i1,i2,i3,loadbalancer
JkWorkerProperty worker.i1.type=ajp13
JkWorkerProperty worker.i1.port=9091
JkWorkerProperty worker.i1.host=appserver1.foo.co.nz
JkWorkerProperty worker.i1.lbfactor=1
JkWorkerProperty worker.i2.type=ajp13
JkWorkerProperty worker.i2.port=9092
JkWorkerProperty worker.i2.host=appserver1.foo.co.nz
JkWorkerProperty worker.i2.lbfactor=1
JkWorkerProperty worker.i3.type=ajp13
JkWorkerProperty worker.i3.port=9093
JkWorkerProperty worker.i3.host=appserver1.foo.co.nz
JkWorkerProperty worker.i3.lbfactor=1
JkWorkerProperty worker.loadbalancer.type=lb
JkWorkerProperty worker.loadbalancer.sticky_session=1
JkWorkerProperty worker.loadbalancer.local_worker_only=1
JkWorkerProperty worker.loadbalancer.balance_workers=i1,i2,i3

JkMount /FooApp/* loadbalancer
```

Again, the details of the exact settings will not be covered in this article, but this should provide a simple guide to setting up this file to provide for a multi-instance deploy under Tomcat.

Restart Apache

Now restart apache with the following command:

```
sudo apachectl restart
```

Test Application

Now you can test your application using a URL such as this one.

```
http://www.foo.co.nz/FooApp/WebObjects/FooApp.woa
```

All three instances that you have setup should take some of the inbound requests.

Choosing the Right Instance

`mod_jk` ensures that requests that have started a session will be directed to the correct instance where the session originated. This behaviour is known as *sticky sessions*. It does not appear to be possible to nominate the instance from the servlet container in another way.