JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2009**

Vol. 8, No. 2, March- April 2009

Creating Servlets with Intellij V8

By Douglas Lyon

Abstract

This paper describes how to configure servlets with Intellij version 8. The GUI intensive approach to enterprise Java has classically led to configuration issues that require both a deployment environment and a lengthy IDE set-up. Students (and, indeed, seasoned professionals) can find this confusing. The article is both a case study and a tutorial on simple enterprise Java configuration and deployment.

Basic questions that we address: Why is deployment so hard? What are these deployments doing? What has led us down a configuration path that is so fragile? What can we do to make things better?

1 THE PROBLEM

There are a number of competing products used for server-side Java programming. This typically involves an *application server* product. Java Enterprise Edition application servers are available from Red Hat, Sybase, BEA, IBM, Adobe, Apache Software Foundation, Oracle, Sun, SAP etc. Even within organizations there are often several application servers available (in various versions).

The IDE manufacturer must find a way to deploy applications to a wide variety of application servers. Each is different from the other, resulting in different procedures for packaging and deployment.

Typically, a programmer will select an application sever, set it up, verify its operation, select an IDE, develop a sample application, deploy it and test. Debugging consists of repeated trials driven by cryptic bug reports. Each deployment can share a virtual machine, but must have an isolated context class loader. A context class loader maps a thread to specific classes and resources. When the thread dies, the classes and resources are freed.

2 THE APPLICATION SERVER

The primary role of the application server is to take care of the business logic. Java applications servers also enable deployment and execution of servlets, the compilation of Java server pages, access to databases and the management of the users' state. Application servers can be loaded on any machine. Often, one or more servers are set up for deployment.

Tomcat is an Apache Software Foundation application server product that is available as an open-source multi-platform distribution http://tomcat.apache.org. On a mac, for example, a download is extracted by typing:

```
tar -xvf apache-tomcat-6.0.18.tar.gz
```

This creates a directory called:

```
apache-tomcat-6.0.18
```

The programmer changes directory into the binary directory of Tomcat and starts it using:

```
cd apache-tomcat-6.0.18/bin
```

The start-up script identifies the location of the application server (called *go.sh*)

```
#!/bin/sh
export
CATALINA_HOME=/Users/lyon/attachments/servlets/apache-
tomcat-6.0.18
export
JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versi
ons/1.5.0/Home
$CATALINA_HOME/bin/startup.sh
sh startup.sh
```

The programmer then changes the modifier list on the file, to make it executable:

```
chmod +x go.sh
```

Now you need to add a user to the Tomcat database:

```
cat >apache-tomcat-6.0.18/conf/tomcat-users.xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
 <role rolename="manager"/>
 <user username="lyon" password="xxx" roles="manager"/>
</tomcat-users>
```

Running is performed with an invocation to *go.sh*. The programmer then verifies the correctness of the installation by visiting http://localhost:8080.

Figure 1-1. The Start-up page for Tomcat

Figure 1-1 shows the start-up page for Tomcat. There are variations on the installation theme that exist for different platforms (Windows users' have an installer, for example).

In summary, the programmer identifies the proper application server, downloads it, uncompresses it, creates a shell script for running it, creates a database, in xml, of users and passwords for administering it, executes the application server and finally tests it. Common problems that can occur include a buggy configuration file setting (try again and restart), incorrect firewall settings (with cryptic output) and occupied ports (something else is already on port 8080). Clearly, this installation was not as easy as a normal application. Some IDEs (like Netbeans) include a customized version of Tomcat. This eases integration, deployment and configuration.

3 THE INTELLIJIDEA

The previous section described the configuration, installation and testing of Tomcat. This section describes the set-up for deploying a servlet to Tomcat. First, start Intellij and select: “create a new project from scratch”.

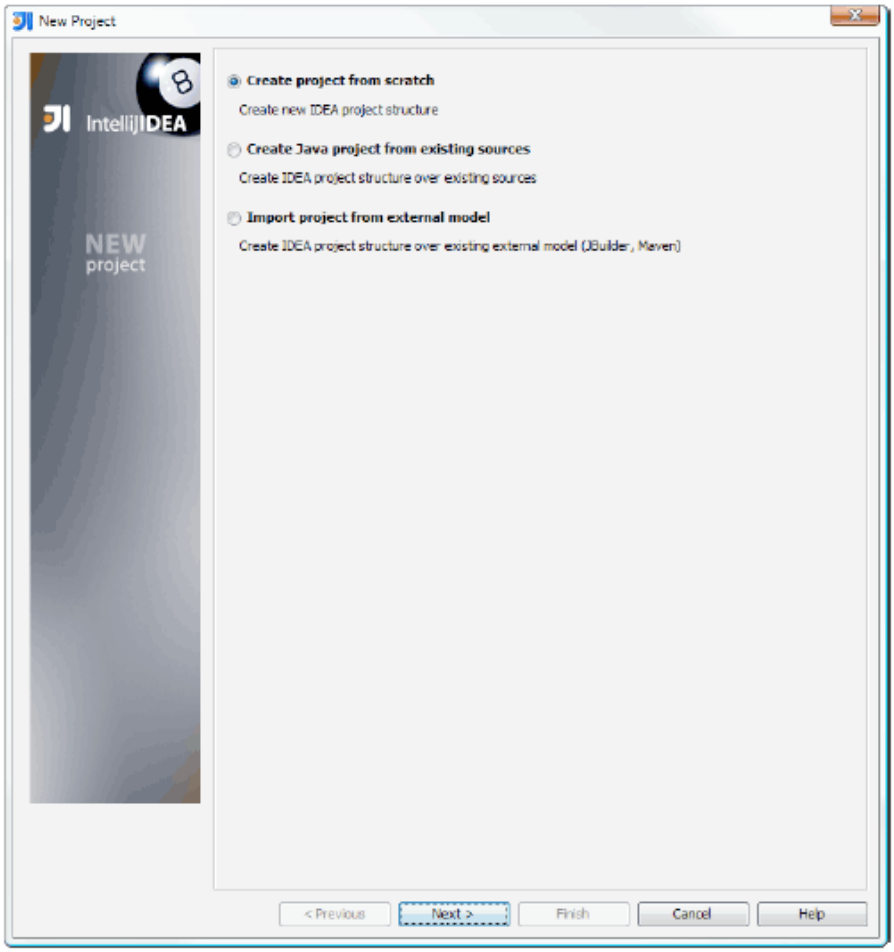

Figure 2-1. Starting Intellij

Figure 2-1 shows the new project dialog that shows when starting Intellij.

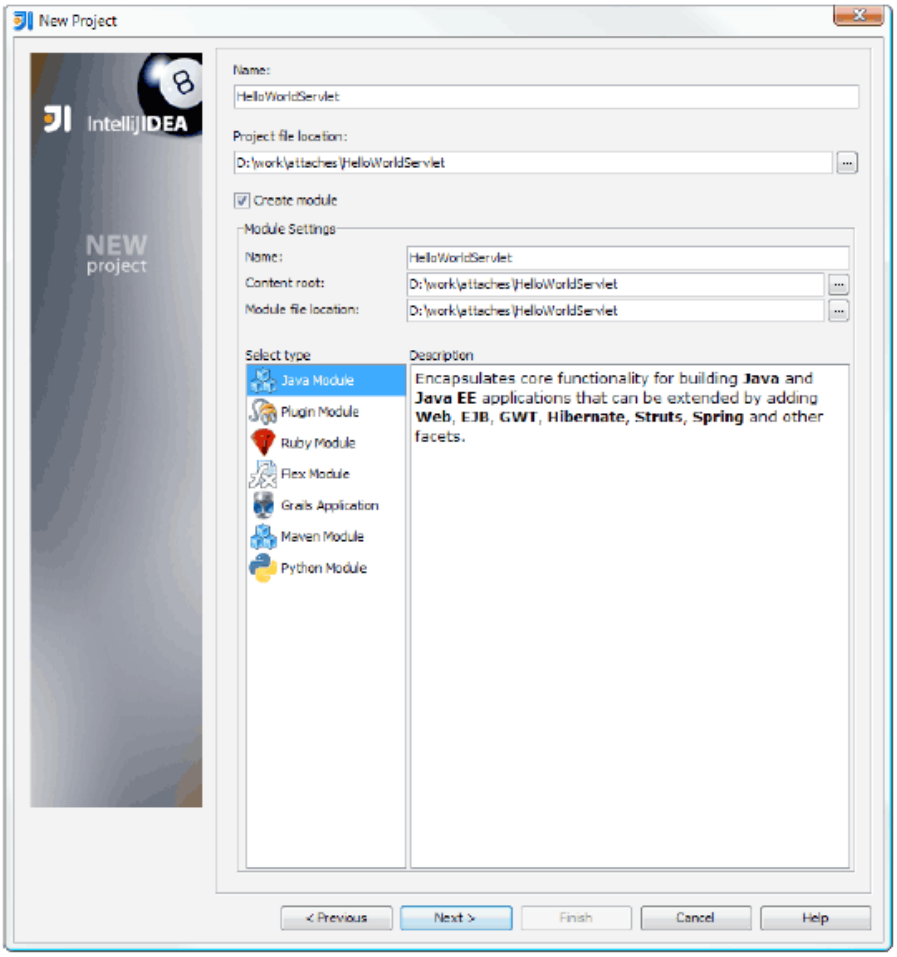

Figure 2-2. Creating the new Project

Figure 2-2 shows a dialog with a new project name (*HelloWorldServlet*) and project file location

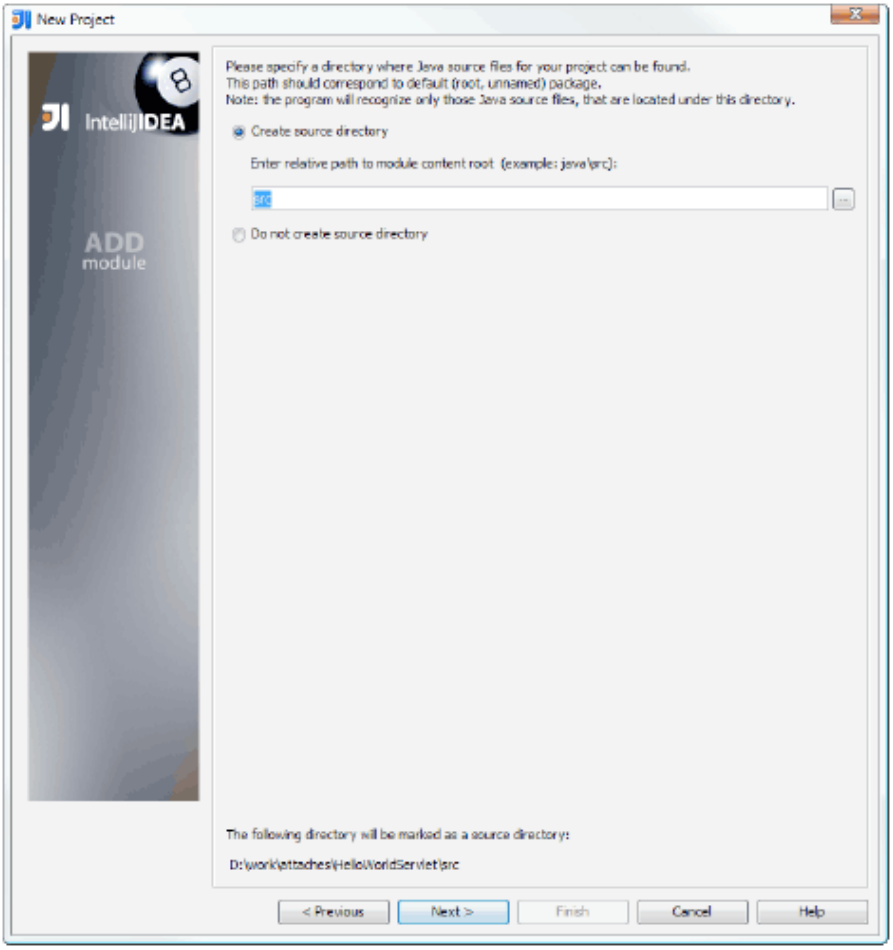

Figure 2-3. Create a new source directory

Figure 2-3 shows a dialog that enables the creation of a new source directory, called *src*.

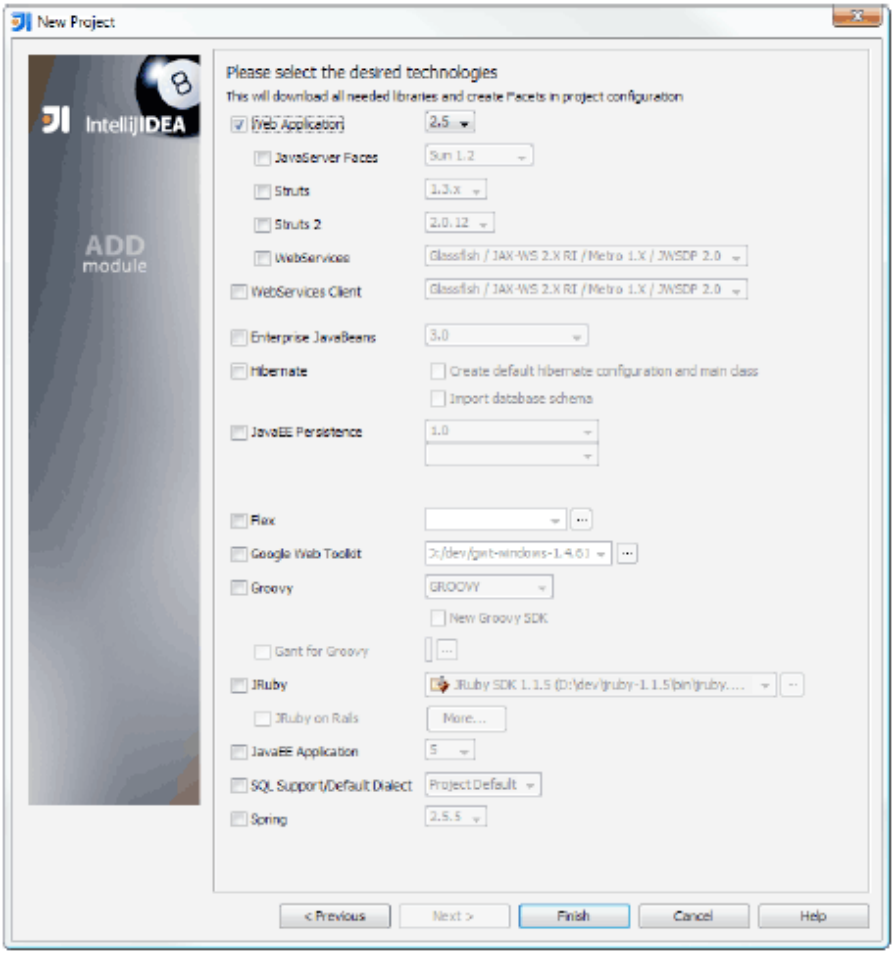

Figure 2-4. The Technology Dialog Box

Figure 2-4 shows an image of the technology dialog box. A web application is characterized by making use of a thin client (like a browser) in order to access the business logic (which typically runs on an application server). After the user hits the *finish* button, the IDE creates the new project.

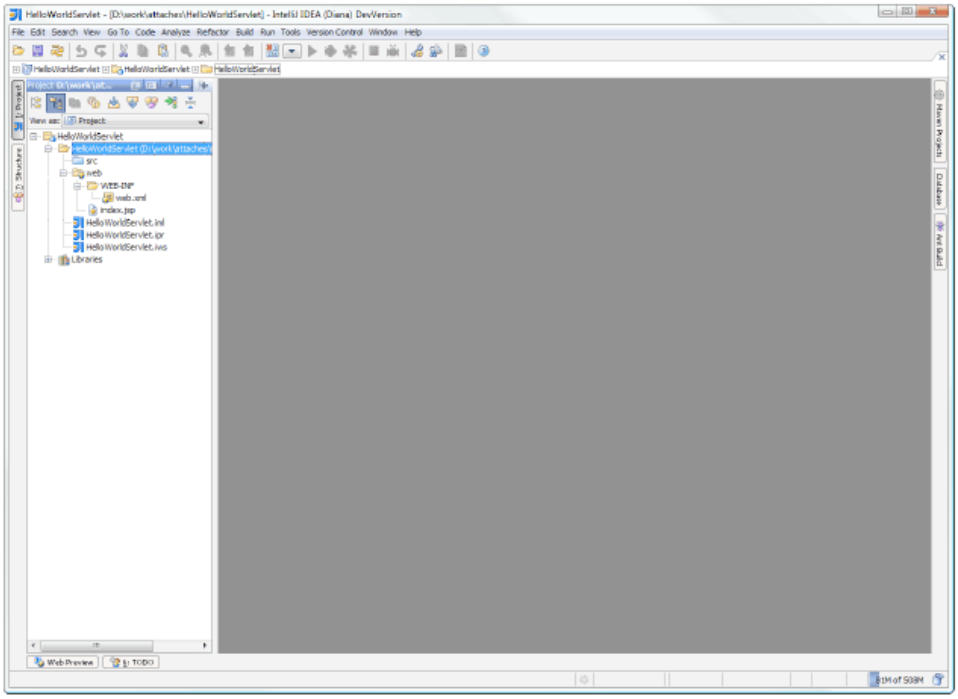

Figure 2-5. The new project

Figure 2-5 shows a new project window in Intellij. We are now ready to write our first servlet.

4 CREATING YOUR FIRST SERVLET

The previous section outlined the steps for setting up a browser to enable the programmer to start to write servlets. This section describes how to write the servlet, using the IntellijIDEA IDE. Mouse right on the project and select the new menu.

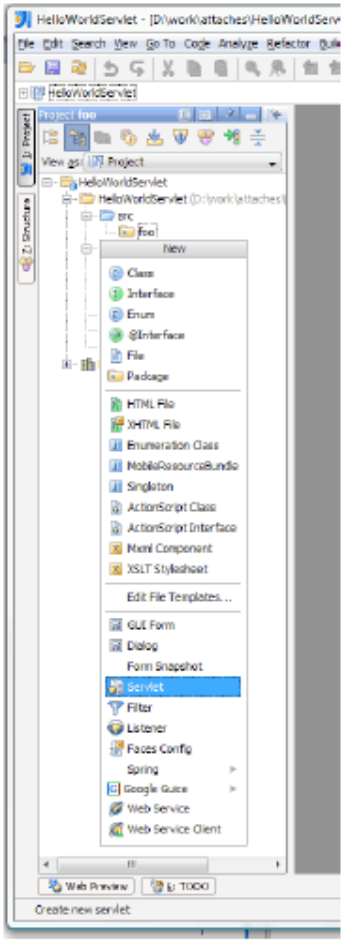

Figure 3-1. Selecting the new servlet menu item.

Figure 3-1 shows the programmer selecting the servlet menu item from the pop-up dialog box.

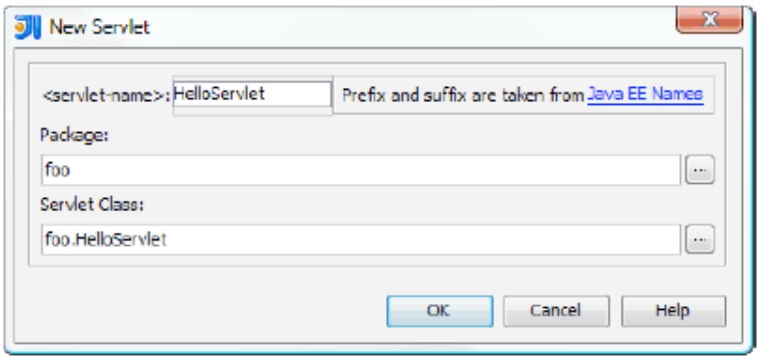

Figure 3-2. Name the servlet

Figure 3-2 shows the package name (*foo*) and the servlet name (*HelloServlet*) entered into the *New Servlet* dialog box.

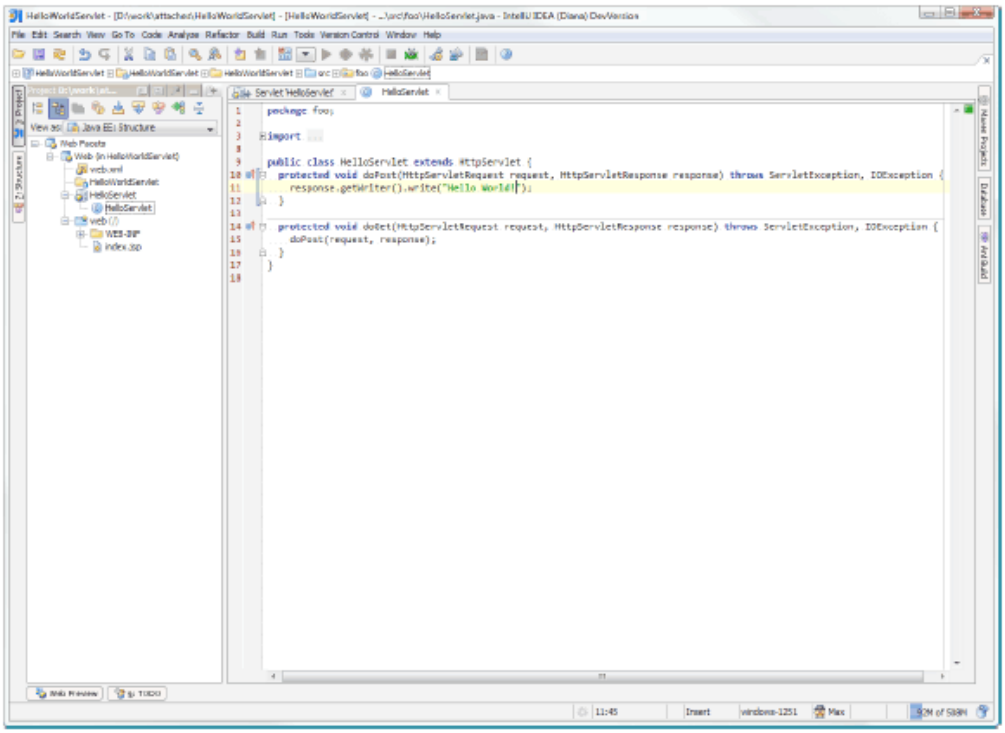

Figure 3-3. Keying in the new servlet

Figure 3-3 shows the user entering in the new servlet.

5 CONFIGURATION OF THE IDE FOR DEPLOYMENT

This section describes how to configure the *IntellijIDEA* for deployment of the application.

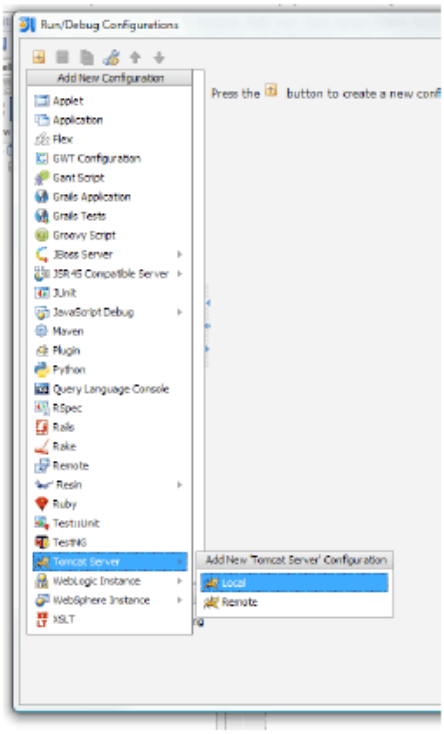

Figure 4-1. Selecting the local server

Figure 4-1 shows the *Run/Debug Configurations* dialog box that appears when the user selects *Edit Configurations* and clicks on the + button. Figure 4-1 shows the user selecting a local configuration for deployment.

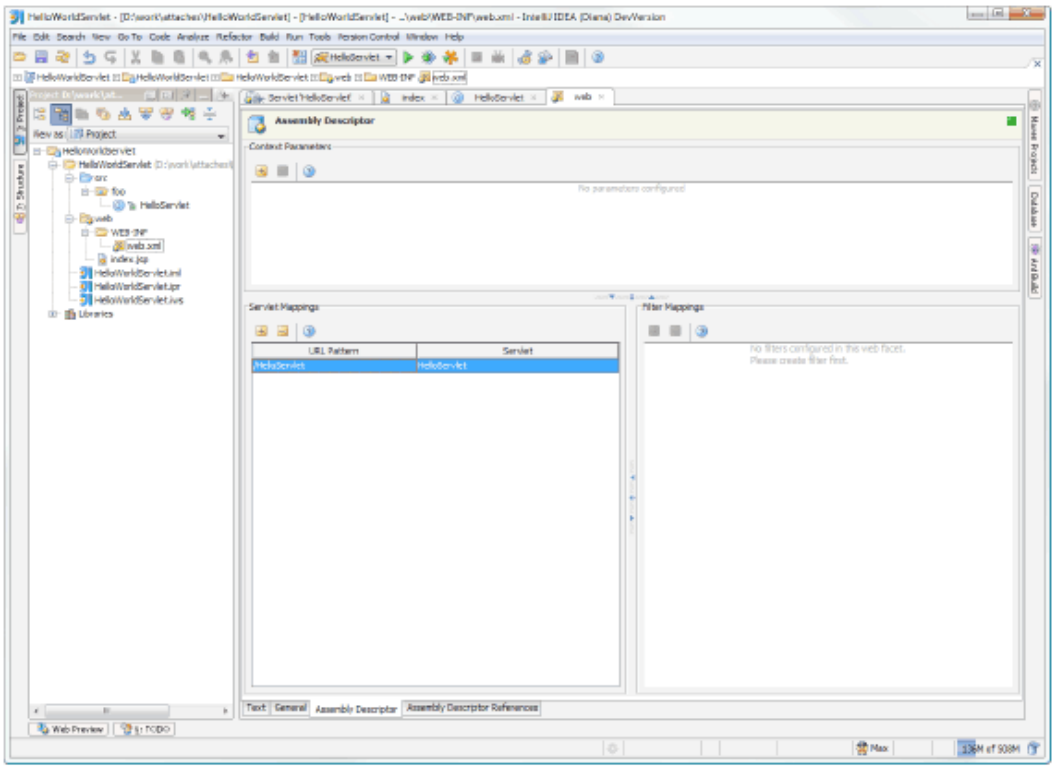

Figure 4-2. Adding a servlet mapping

The programmer double clicks on the web facet, selects the assembly descriptor and enters in the new URL pattern to map to the *HelloServlet*.

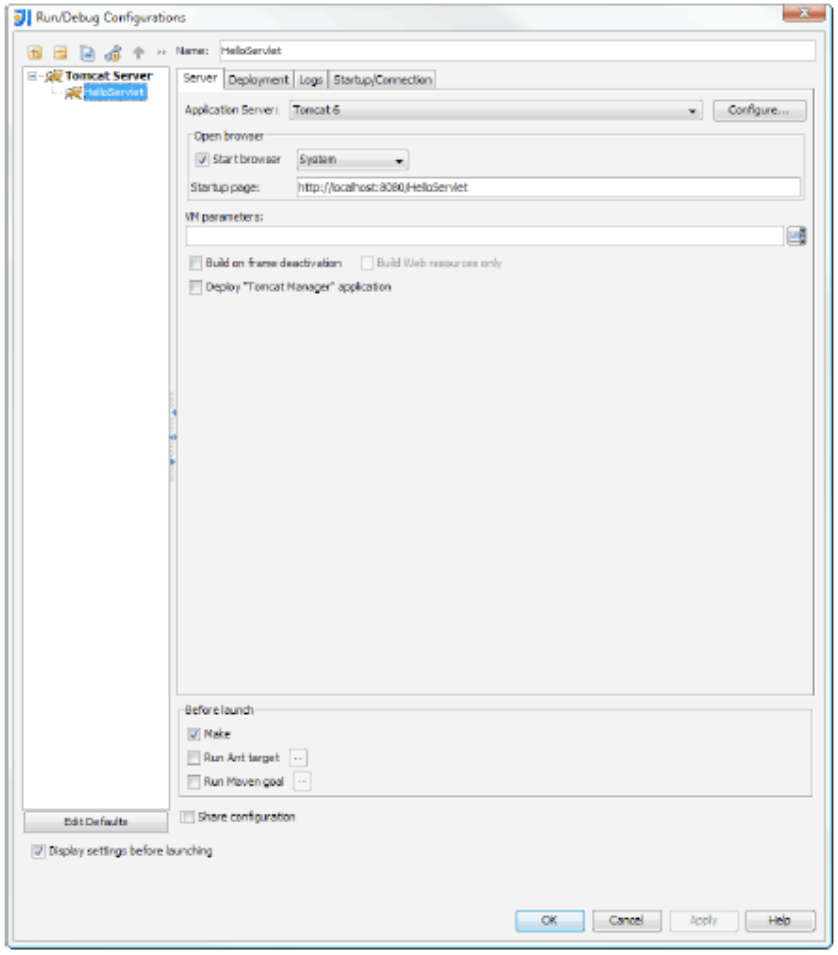

Figure 4-3. The configurations dialog box

Figure 4-3 shows the *Run/Debug Configurations* dialog box, with the new Tomcat application server selected.

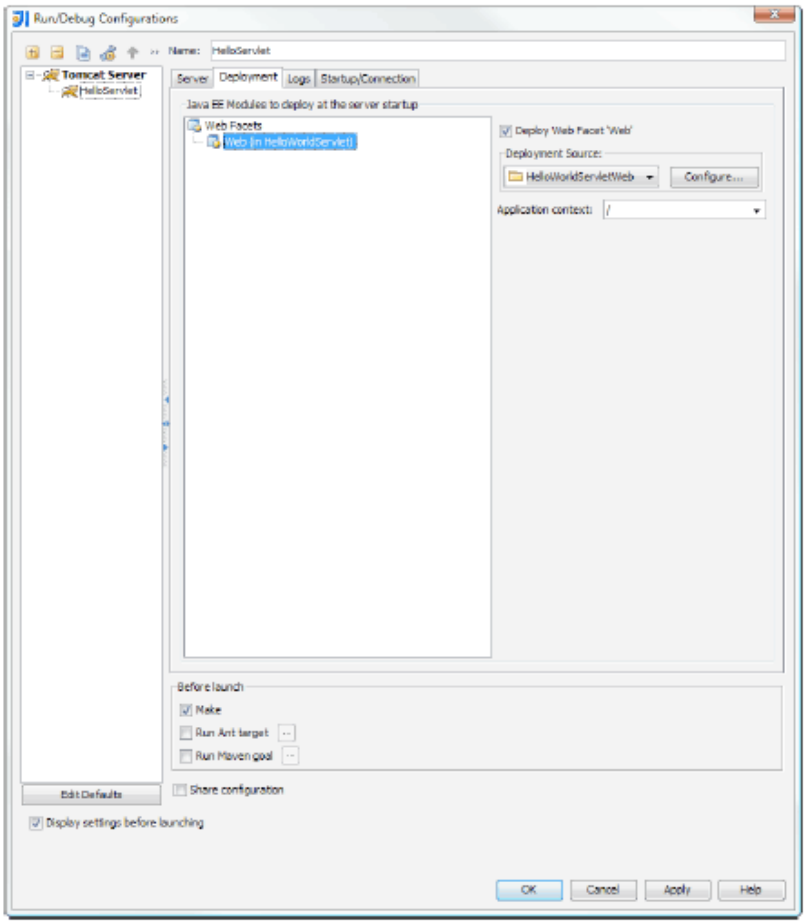

Figure 4-4. The deployment configuration

Figure 4-4 shows the deployment configuration dialog box with the *deploy web facet* selected. If Tomcat is still running, you must run the *shutdown.sh* script to shut it down before deployment. Otherwise you get an *AddressInUse* exception at run-time (since Tomcat is already running).

6 SUMMARY

The problems encountered with deployment have been documented in our 2005 case study involving the JBoss plugin for Intellij [Lyon 2005b-e]. These problems include issues of IDE integration, server communication, file system organization and resource bundling [Lyon 2005a].

Now, three years later, we examine a new deployment scenario involving the simplest of servlets onto one of the more common application servers. Our new work shows that the procedure is a fruitful source of bugs. As is typical, all the effort is expended in set-up and debugging in order to get the system working.

We need to make the process a little easier for the programmer. Perhaps a process involving automated communication in a standard manner would be of help in deployment. System integration is a primary focus of our efforts. An XML file is used to enable a mapping between the servlet name and the class name. This creates a non-

zero probability of a run-time error. Application server set-up and execution could be a less fruitful source of bugs if it were automated. The use of special files and directory structures causes IntelliJ to ask the user to create a new module of web application type. Our simple servlet needed a local server, servlet mapping, a browser for test and a servlet file. Each of these things had to be set up manually.

The expert programmers get so good with configuration that they treat the problems with set-up as trivial, and ignore them. This leaves the programmers new to the IDE to figure set-up on their own. Clearly, the primary element in easing deployment and configuration bugs is in better integration and debugging tools. Other IDE's seem to be better at server-side integration than Intellij, but the question of how to improve the integration with Intellij remains open.

REFERENCES

[Lyon 2005a] "Resource Bundling for Distributed Computing" by Douglas A. Lyon, *Journal of Object Technology*, vol. 4, no. 1, January-February, 2005, pp. 45-58, http://www.jot.fm/issues/issue_2005_01/column4/.

[Lyon 2005b] "The JBoss Integration Plug-in for IntelliJ IDEA", Part 1 by Douglas A. Lyon, Martin Fuhrer and Thomas Rowland, Journal of Object Technology, vol. 4, no. 5, July-August, 2005, pp. 7-17, http://www.jot.fm/issues/issue_2005_07/column1/.

[Lyon 2005c] "The JBoss Integration Plug-in for IntelliJ IDEA", Part 2 by Douglas A. Lyon, Martin Fuhrer and Thomas Rowland, *Journal of Object Technology*, vol. 4, no. 7, September-October, 2005, pp. 25-34, http://www.jot.fm/issues/issue_2005_09/column3/.

[Lyon 2005d] "The JBoss Integration Plug-in for IntelliJ IDEA", Part 3 by Douglas A. Lyon, Martin Fuhrer and Thomas Rowland, *Journal of Object Technology*, vol. 5, no. 3, March-April, 2006, pp. 13-26, http://www.jot.fm/issues/issue_2006_03/column2/.

[Lyon 2005e] "The JBoss Integration Plug-in for IntelliJ IDEA", Part 4 by Douglas A. Lyon, Martin Fuhrer and Thomas Rowland, *Journal of Object Technology*, vol. 4, no. 9, November-December, 2005, pp. 11-21, http://www.jot.fm/issues/issue_2005_11/column2/.

ACKNOWLEDGEMENT

The author is indebted to Serge Baranov, of JetBrains, for his assistance with Intellij IDEA and feedback on this article.

About the author

Douglas A. Lyon (M'89-SM'00) received the Ph.D., M.S. and B.S. degrees in computer and systems engineering from Rensselaer Polytechnic Institute (1991, 1985 and 1983). Dr. Lyon has worked at AT&T Bell Laboratories at Murray Hill, NJ and the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA. He is currently the Chairman of the Computer Engineering Department at Fairfield University, in Fairfield CT, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. Dr. Lyon has authored or co-authored three books (Java, Digital Signal Processing, Image Processing in Java and Java for Programmers). He has authored over 30 journal publications. Email: lyon@docjava.com. Web: http://www.DocJava.com.