Automatability: Tomcat vs Glassfish

By Kohsuke Kawaguchi

Background

I was writing an end-to-end test harness for the JAX-WS RI and WSIT. One of the requirements for this attempt was to completely automate the interaction with containers. This was needed so that we can reliably run multiple instances of tests, perhaps with different configurations, on the same machine. To put this into another way, I needed to programatically control containers. I did this for both Tomcat and Glassfish. This document summarizes what I found, in the hope that the Glassfish team will find useful.

FULL DISCLOSURE: when I attempted this work, I had more experience of working with Tomcat than with Glassfish.

Comparisons

The following table compares Tomcat and Glassfish from key automation requirements.

Regarding "initial setup" and "create new configuration" --- To automate containers, you first need to have a master copy of a container. This is where all the jars will be loaded. Then for each run of container you need to have a "configuration". This is where we tell things like "use HTTP port 18520 and put logs into /var/tmp/wstest19303/logs/foo.log". This is analogous to class/instance relationship in Java.

Metrix	Tomcat	Glassfish
Initial setup	Extract tomcat bundle. Done. The same image can be used on any test platforms, and can be moved to any directory freely.	Run the poorman's installer, which occasionally hangs on some system. Then run "ant -f setup.xml". The master Glassfish set up in this way cannot be moved to other directories, because $GLASSFISH_HOME/config/asenv.conf is hard-coded to a particular location. Also, I need to prepare a separate image for each platform (linux-amd64, solaris-sparcv9, and solaris-i586) Also, it hard-codes path to JDK, making it even harder to move.
Create a new configuration for a test run	Create a few directories. Then copy a few XML files into conf/ folder, and the manager app if you want. Configuration can be changed by modifying these XML files (such as HTTP port and logger setting.) The whole thing takes less than a second. Configuration files are well-documented, and it's easy to learn how to tweak them.	Invoke "asadmin create-domain" as a separate process. Some configurations can be specified as command-line options (such as HTTP port), but not logging. The whole operation takes 5 to 10 seconds to complete. Generated configuration XML (config/domain.xml) says "Avoid manual edits", and it's not documented.
Configure	conf/server.xml shows how to do this. I can send log output to stdout/stderr, so that	

logging so that I can capture logs in a way I want.	I can capture all Tomcat output at once. With some additional work, I can also choose to capture logs per web application, allowing concurrent tests to differenciate log output.	I took a look inside config/domain.xml, but failed to find the configuration.
Start and stop container	Invoke "java -jar $CATALINA_HOME/bin/bootstrap.jar" as a separate process with Runtime.exec()	Invoke "asadmin start-domain" as a separate process. Note that you can't just use Runtime.exec(). See bug 885. Also, if start-domain fails, the caller can't find that out. See bug 884. Finally, Glassfish takes considerably longer to launch than Tomcat.
Deploy and undeploy applications	If server and client are on the same machine, I can use the manager app to deploy any WAR or directory anywhere in the file system. This is fast, because there's no packaging and file copying involved. I can also choose to put a war on webapps directory to be auto-deployed, or copy a WAR over network (via manager app) for remote deployment. This whole feature is well-documented.	Deployment via JSR-88. This requires a WAR to be created, which is additional processing that takes more time. JSR-88 was an OK API to use, but you need additional jars in the test harness to use JSR-88. I couldn't locate JSR-88 sample with Glassfish on the web, even though some steps are GF-specific (such as initial connection establishment.) But when I asked, friendly fellow GF developers sent me a few pointers quickly.
Embedded container	Possible, although not well-documented. This set up works very quickly, because (1) it avoids overhead of creating a new JVM, (2) I don't need to rebundle JAX-WS/WSIT runtime into each war, (3) a simple System.exit() will terminate both the container and the test, and (4) debugging is easy because you only need to have one debug session. I found this mode to be ideal for the JAX-WS test harness.	Impossible.
Source zip, when I need to debug into container	Available for download, even though it doesn't contain all the source code for everything. Source files were split to many sub directories per each module, so it's not easy to configure IDE to recognize needed source files.	The same as Tomcat. The problem is worse with Glassfish, however, because there are many modules.
	There are two kinds of people who read the source; one who wants to hack the code, and the other who wants to learn what's going on. I was the latter; I only wanted to step into GF's code, just to make sure I'm not doing something stupid in my code, or to diagnose the problem better. For this, all I needed was a single source zip that contains all source files according to their package structure. It would have been a lot nicer if the container comes with two kinds of source bundle, for these two kinds of audience.	

Conclusion

Automatability is important, because automation improves productivity. More people are interested in automation these days, as you can see in the raise of Ant, Maven, and countless CI tools.

Unfortunately, on most of the key points that mattered to me, I have to report that Glassfish is lagging behind Tomcat. I hope this situation will be improved in future.