

Jetty Automated Distribution Testing

Jetty Automated Distribution Testing

We have been working on a new mechanism for testing distributions, an automated way to deploying specially written webapps that exercise behaviors and report back results to a JUnit testing infrastructure that validates success for failure. This page will highlight mechanism important to working with this setup.

The JettyProcess

The JettyProcess is an object that wraps the lifecycle of starting and stopping a jetty distribution. It contains a variety of convenience methods that allow you to customize the distribution. There is also an overlay mechanism supported that allows the user to create more complex test cases overwriting typical files in the distribution. An example would be for the security tests that we initially developed this infrastructure to more fully test.

Test Webapps

We have a fair number of test webapps suitable for testing in a variety of circumstances. These are located the test-wars directory under the jetty7 branch. These wars range from simple webapps that help validate java versions, jsp versions, servlet api versions, to testing jndi setup and many others. These webapps can be used in configuring the JettyProcess instance to be used for a given unit test.

Practical Security Example:

The Practical security example makes use of the jettypolicy project from the eclipse side of jetty. You can read more about that system at <http://wiki.eclipse.org/Jetty/Tutorial/Jetty-Policy> which will help explain the what and why for.

Basically what this test does is deploy the test-war-policy webapp from the test webapps directory to a policy-tests directory in the \$

Unknown macro: {jetty.home}

/work directory and then make a series of calls to test varying bits of functionality. A specific security policy file is created and used for setting the permissions required for the test webapp to execution. The JettyProcess is in control of starting up and configuring the container for the test execution which is performed in the @Before portion of test initialization. Each @Test case is performed against this running container. These test cases make a specific call to a servlet in the webapp that then performs operations of a security conscious form. For example the testFileSystemAccess validates that the webapp that is running in jetty is able to access itself but not other aspects of the container filesystem. It is not able to read the log files, not able to read other webapps, not able to read the lib directory, etc. Another method checks that a webapp can be configured to read certain System Properties but not others. In this way we can check all manner of security related checks from the actual execution environment we are trying to secure. The @Test that is making this request gets back a string of json containing the results of all of the tests and then iterates through the results ensuring that each was a success.

JMX Example

This same testing setup allows us to test things like JMX deployment as well. We can use the JettyProcess to configure the container to startup with jmx support enabled and then against this running distribution validate all manner of jmx activities. We can check that certain beans are reported expected results and we can also programmatically check that other jmx support like the deployment manager support are all working, something that would traditionally require us to manually test.

JSP Support

We can also support automated test of jsp support configurations. We don't have those in place at the moment but this infrastructure is nimble enough that we should be able to support all manner of distribution based mechanism.