Debugging native Maven jobs

Hudson runs significant amount of code inside the Maven process for the native maven2 job type, and during the plugin development one would often like to debug this. Here's how you do it:

During development

Hudson looks at the system property "hudson.maven.debugPort" and if this is set, every managed Maven process is launched with this port as the debugger port. Therefore, if you run `mvn -Dhudson.maven.debugPort=5001 hpi:run`, then you can start a build normally from Hudson, then attach a debugger to port 5001 to see what's going on inside the managed Maven process.

If the plugin doesn't show up in the UI, run `mvn clean hpi:run` to force the regeneration of the Jelly bindings.

Eclipse users might separate Eclipse generated classes by running `mvn -DdownloadJavadocs=true -DoutputDirectory=target/eclipse-classes eclipse:eclipse` when generating configuration.

Debug jobs selectively

The previous approach causes every job to launch with the debugger port, but you can also selectively enable the debugger support. This can be even used on the production system, and therefore very handy if you need to trouble-shoot a problem in a live system.

To do so, click "Advanced..." under the build section and add "`-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=5001`" to the MAVEN_OPTS field. (And when you do this, there's no need to have the `hudson.maven.debugPort` system property set. In this way, only builds of this job will be started with the debugger support.

Debug jobs on a slave

Debugging a maven job that runs on a slave works in a similar way. If port 5001 is already taken by Hudson's slave agent, then change the port to another value, e.g. 5002: "`-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=5002`". Then connect your debugger to the IP address of your slave and the specified port (5001 or 5002).