Ant Jetty Plugin

Ant Jetty plugin

There is a new plugin available in the /extras/ant location, which makes it possible to start Jetty web server directly from the Ant build script, and, actually, to embed the Jetty web server inside your build process.
It's purpose is to provide almost the same functionality as Jetty plugin for Maven, e.g. dynamic application reloading, working directly on web application sources, tight integration with build system, etc.

For the impatient ones, there is already a demo provided in jetty-contrib repository for the Ant Jetty plugin.
It's good to have jetty compiled before running these tests, because the demo depends on Jetty JARs from the Maven repository. Alternatively, you can download these JARs manually from one of the Maven repositories. Please take a look at build.xml file in jetty-ant-demo folder for a list of needed JARs.

Modifying build.xml from jetty-ant-demo is also the easiest way to start playing with this plugin.

How to start?

(Obviously) Ant plugin depends on Jetty JARs, and so far it's necessary to build Jetty and then Ant plugin directly from sources. Here's how it goes:

1. Get the latest jetty sources from http://dist.codehaus.org/jetty/
2. Unpack jetty and build it with 'mvn install' (If you don't have Maven yet, grab it from http://maven.apache.org/)
3. Go to the extras/ant/ folder and build ant plugin with 'mvn install'
4. Go to <your M2 repo root>/repository/org/mortbay/jetty/jetty-ant and grab the jetty-ant plugin JAR.
5. Get these files to run Jetty with JSP/Servlets support (find these in <your M2 repo root>/repository/org/mortbay/jetty):
    - jetty.jar
    - jetty-util.jar
    - jetty-plus.jar
    - jetty-naming.jar
    - servlet-api-2.5.jar
    - jsp-api-2.1.jar
    - jsp-2.1.jar
    - and core-3.1.1.jar from <M2 repository>/org/eclipse/jdt/core/3.1.1/
6. Put all the JARs inside one lib directory (let's say jetty-lib).

How to prepare build.xml file?

Let's start with an empty build.xml [file:
]

```
<project name="Jetty-Ant integration test"
basedir=".">
</project>
```

Then, a *jetty* task must be defined:

```
<project name="Jetty-Ant integration test"
basedir=".">
  <path id="jetty.plugin.classpath">
    <fileset dir="jetty-lib"
includes="*.jar"/>
  </path>
  <taskdef
classpathref="jetty.plugin.classpath"
resource="tasks.properties"
loaderref="jetty.loader" />
</project>
```

Now, we are ready to add a new target with Jetty container declaration:

```
<project name="Jetty-Ant integration test"
basedir=".">
  <path id="jetty.plugin.classpath">
    <fileset dir="jetty-lib"
includes="*.jar"/>
  </path>
  <taskdef
classpathref="jetty.plugin.classpath"
resource="tasks.properties"
loaderref="jetty.loader" />

  <target name="jetty.run">
    <jetty />
  </target>
</project>
```

Starting jetty on port 8080 is then a simple matter of executing *ant jetty.run* from the command line.

How to deploy web applications?

Adding web applications to the Jetty web server is now a simple matter of putting extra *webapp* tags inside *jetty* tag. Notice, that a *tempDirectory* attribute has been added to the *jetty* task to specify where temporary files are stored.

```
<project name="Jetty-Ant integration test"
basedir=".">
  <path id="jetty.plugin.classpath">
    <fileset dir="jetty-lib"
includes="*.jar"/>
  </path>
  <taskdef
classpathref="jetty.plugin.classpath"
resource="tasks.properties"
loaderref="jetty.loader" />

  <target name="jetty.run">
    <jetty tempDirectory="jetty-temp">
      <webApp name="webapp1"
warfile="webapp1.war" contextpath="/webapp1"
/>
    </jetty>
  </target>
</project>
```

It's possible to put many *webapp* tags inside *jetty* tag, thus to deploy many web applications simultaneously.

Plugin syntax

There are many ways to configure *jetty* task:

- *tempDirectory* - specifies the Jetty temporary web directory:

```
<jetty tempDirectory="jetty-temp" />
```

- *requestLog* - defines a request logger:

```
<jetty ... requestLog="your.fancy.request.logger"/>
```

- *jettyXml* - applies additional configuration via a standard jetty.xml file:

```
<jetty ... jettyXml="path/to/jetty.xml"/>
```

- *systemProperties* - adds system properties when starting Jetty:

```
<jetty ...>
  <systemProperties>
    <systemProperty name="jetty.status" value="Jetty Rocks!"/>
  </systemProperties>
</jetty>
```

- *connectors* - defines Jetty connectors (a better way still needs to be developed):

```
<typedef name="selectChannelConnector"
classname="org.mortbay.jetty.nio.SelectChannelConnector"
                            classpathref="jetty.plugin.classpath"
loaderref="jetty.loader" />
...
<jetty ...>
  <connectors>
    <selectChannelConnector port="8090" />
  </connectors>
</jetty>
```

-

- *userRealms* - adds authentication via user "realms":

```
<typedef name="jdbcRealm"
classname="org.mortbay.jetty.security.HashUserRealm"
                  classpathref="jetty.plugin.classpath"
loaderref="jetty.loader" />
...
<jetty ...>
  <userRealms>
    <userRealm name="realm1" config="path/to/realm/config" />
  </userRealms>
</jetty>
```

-

There are also many ways to configure web applications with *webapp* tag:

- *name* - name of a web application, which is used by DefaultServlet to display application description

```
<jetty ...>
  <webapp name="name" .../>
</jetty>
```

- *warfile* - path to .war file or a directory with web application contents:

```
<jetty ...>
  <webapp warfile="/path/to/war/file/or/web/application/contents" .../>
</jetty>
```

- *contextPath* - a context path a particular web application will be deployed to, and thus where it will be accessible (http://localhost:8080/webapp in this case):

```
<jetty ...>
  <webapp contextPath="/webapp" .../>
</jetty>
```

- *jettyEnvXml* - path to jettyEnv.xml file with JNDI resource declarations

```
<jetty ...>
  <webapp jettyEnvXml="/path/to/jettyEnv.xml" .../>
</jetty>
```

- *webXml* - path to web.xml file. If not specified, the default (location-of-war-file/WEB-INF/web.xml) location is used:

```
<jetty ...>
  <webapp webXmlFile="/path/to/web/xml" .../>
</jetty>
```

- *scanTargets* and *scanIntervalSeconds* - used to point to a special files/folders which need to be scanned and specify scanner interval in seconds respectively. If any of files changes the web application is being automatically reloaded. Scan interval and scanned files can be adjusted per web application.
  IMPORTANT: Please note, that all files included with <classes/> or <lib/> tags are **automatically** scanned for changes if you provide *scanIntervalSeconds* parameter in the <jetty/> tag. There's no need to add them using the <scanTargets/> section.

```
<jetty ...>
  <webapp ... scanIntervalSeconds="5">
    <scanTargets dir="path/to/included/resources">
      <include name="filename/patterns" />
    </scanTargets>
  </webapp>
</jetty>
```

- *lib* - specifies which libraries needs to be included for a particular web application (treated as files inside WEB-INF/lib folder). Multiple *lib* tags are allowed.

```
<jetty ...>
  <webapp ...>
    <lib dir="path/to/jar/folder" includes="*.jar"
excludes="not-needed.jar" />
  </webapp>
</jetty>
```

- *classes* - specifies which folder contains web application classes (treated as WEB-INF/classes folder). Multiple *classes* tags are allowed.

```
<jetty ...>
  <webapp ...>
    <classes dir="path/to/classes/folder" includes="*.class" />
  </webapp>
</jetty>
```

- *connectors* - specifies additional web application connectors