ProjectBuilding

With the new ability of the XConfPatch Task to take parameters from Ant's build.properties, we have developed handy new ways to structure projects that we are developing with Cocoon. I will outline our approach here.

Criteria

- multiple developers are working on the project
- we are sharing the project in a CVS
- each developer has their own version of the project in their local sandbox
- each developer has Cocoon running on their system
- no developer has their system setup in such a way that everything is in the same place
- there is a client-facing development server that can use the same built scripts, but mounts our project as the root of a virtual host, using Apache.

Outline

- each developer checks out Cocoon from CVS and builds it `./build.sh webapp`
- each developer sets the Environment Variable: `export COCOON_HOME=~/Development/Checkouts/Apache/cocoon-2.1` or equivalent
- each developer runs Cocoon using the built-in Jetty Servlet Container `./cocoon.sh servlet`
- each developer checks out the shared Project from CVS
- the Project Build process patches the Project into Cocoon in such a way that Cocoon runs the Project directly in the project's `webapp` folder

Project Structure

Our Project Folders look something like this:

- build.properties
- build.sh
- build.xml
- local.build.properties
- lib/ <-- the project's jar files
- src/ <-- the project's Java Source files
- webapp/ <-- the project's mountable webapp directory, the internal structure is dependant on the needs of your project
    - sitemap.xmap <-- required, the top-level sitemap for your project
    - content/
    - flow/
    - resources/
    - stylesheets/
    - xsp/

The Build Process

We have several Ant Targets:

- `init` - set things up
- `clean` - strips our classes out of Cocoon
- `compile` - compiles our Java Classes to `$COCOON_HOME/webapp/WEB-INF/classes`
- `customise` - runs a set of XConfPatch Tasks to modify Cocoon so that our Project runs
- `install` - installs our project's `/lib/*jar` and (in our case) Hibernate mapping files etc. into Cocoon's webapp
- `javadoc` - builds the JavaDocs of our Project's Classes
- `test` - runs jUnit tests on our Project's Classes

While a developer is working on the Java Classes of a project, the typical Ant Target they will use is `test`. When their tests are passed, the developer will restart Cocoon and can immediately access the Project in their Browser.

While a developer is working on XML, XSLT, CSS, FlowScript etc. They do not need to use any build targets, because Cocoon is serving the Project live out of the Project's `webapp` folder.

When a developer wishes to update from CVS their copy of Cocoon, once Cocoon is re-built, they only need to invoke the `install` target, to ready Cocoon for this Project again.

The Projects build.xml

The init target

{{{<target name="init">
<property environment="env"/>
<property file="local.build.properties" />
<property file="build.properties" />
<echo message="${now}: Building '${project.mount}' into ${cocoon.webapp.home}"/>
<taskdef name="xpatch" classname="XConfToolTask" classpath="${tools.tasks.dest}"/>
</target>
}}}
This target loads up the Ant Properties for the project and defines the `xpatch` task as using `XConfToolTask`.

The install target

{{{<target name="install" depends="compile, customise-webapp">
<copy todir="${cocoon.webapp.home}/WEB-INF/lib">
<fileset dir="lib" includes="**.jar"/>
</copy>
<!-- copy hibernate config files (this is specific to our application) -->
<copy todir="${cocoon.webapp.home}/WEB-INF/classes">
<fileset dir="src" includes="*.cfg.xml"/>
<fileset dir="src" includes="**/*.hbm.xml"/>
</copy>
</target>
}}}
This target copies over any `.jars` in our `/lib/` folder, plus our Hibernate files

The customise target

{{{<target name="customise" depends="init">
<xpatch file="${cocoon.webapp.home}/sitemap.xmap" srcdir="">
<include name="${customconf}/*.xmap" />
<include name="${customconf}/*.xpipe" />
</xpatch>
<xpatch file="${cocoon.webapp.home}/WEB-INF/cocoon.xconf" srcdir="">
<include name="${customconf}/*.xconf" />
</xpatch>
<xpatch file="${cocoon.webapp.home}/WEB-INF/logkit.xconf" srcdir="">
<include name="${customconf}/*.xlog" />
</xpatch>
<xpatch file="${cocoon.webapp.home}/WEB-INF/web.xml" srcdir="">
<include name="${customconf}/*.xweb" />
</xpatch>
</target>
}}}
This target applies all of our XConfPatch files which prepare Cocoon for serving our Project.

The test target

{{{<target name="test" depends="install">
<path id="test.classpath">
<fileset dir="${cocoon.webapp.home}/WEB-INF/lib">
<include name="**/*.jar"/>
</fileset>
<pathelement location="${cocoon.webapp.home}/WEB-INF/classes"/>
</path>
<junit printsummary="yes" haltonfailure="yes" fork="yes">
<classpath refid="test.classpath"/>
<formatter type="plain" usefile="no" />
<batchtest>
<fileset dir="${cocoon.webapp.home}/WEB-INF/classes">
<include name="**/*TestCase.class"/>
<include name="**/*Test.class" />
<exclude name="**/AllTest.class" />
<exclude name="**/*$$*Test.class" />
<exclude name="**/Abstract*.class" />
</fileset>
</batchtest>
</junit>
</target>
}}}
This target runs any jUnit tests we may have in our Project's Class Hierarchy.

The compile target

{{{<target name="compile" depends="init">
<path id="build.classpath">
<fileset dir="lib">
<include name="**/*.jar"/>
</fileset>
<fileset dir="${cocoon.webapp.home}/WEB-INF/lib">
<include name="**/*.jar"/>
</fileset>
</path>
<javac srcdir="src"
destdir="${cocoon.webapp.home}/WEB-INF/classes"
classpathref="build.classpath"
encoding="${encoding}"
debug="${debug}"
optimize="${optimize}"
deprecation="off" >
</javac>
</target>
}}}
This target compiles our Project's Java Classes directly into Cocoon's `WEB-INF/classes`.

I will leave the other targets as an exercise for the reader ....

The Patch files.

Patching the SiteMap mount-point

file: `project.xmap`
{{{<xmap xpath="/sitemap/pipelines/pipeline"
unless="match[contains(@pattern,'${project.mount}')]"
insert-after="match[contains(@pattern,'api')]"
>

<!-- Mount the Project -->
<map:match pattern="${project.mount}/**">
<map:mount src="file:/${basedir}/webapp/"
uri-prefix="${project.mount}"
check-reload="yes"/>
</map:match>
</xmap>
}}}

Depending on your Project's needs, you may also need patches to add input-modules, datasources, components, etc. to Cocoon's `cocoon.xconf`, and maybe add a databse driver to Cocoon's `web.xml`.

See the patch files that come with Cocoon for further examples. (Each block has a set).

Build Properties

Finally we have the Project's `build.properties` file. This provides all of the Properties referenced above, that are able to be localised for each developer's needs (because everyone keeps things in different places).

Here is a subset of ours:

{{{# NOTE: don't modify this file directly but copy the properties you need
to modify over to a file named 'local.build.properties' and modify that.

WEBAPP - the webapp you are testing in
get the directory of Cocoon from a command-line parameter, supplied by build.sh cocoon=${env.COCOON_HOME}
the location of the running webapp within the Cocoon directory cocoon.webapp=build/webapp
put them together cocoon.webapp.home=${cocoon}/${cocoon.webapp}

BUILD
where to find Cocoon's Ant XPatch Task tools.tasks.dest=${cocoon}/tools/anttasks

PROJECT - properties specific to this webapp
this project's folder within the webapp project.mount=my-project-name
source directory src.dir=src
this project's patch files to modify the webapp customconf=${src.dir}/confpatch
}}}

How could this be better?

Well, we are pretty happy with the adaptability of the scheme so far, but one of the issues we are thinking about is how to use this with IDEs that can use Ant, like Eclipse, NetBeans IDEA etc.

I believe one of the issues we need to deal with is to find a way to make the `build.xml` file work when it is called directly from Ant, rather than by invoking `build.sh`. We need to find the right way to setup the `java.endorsed.dirs` parameter..

What do you think?

2003-11-22 Updated JeremyQuinn: Now using Ant's built-in ability to retrieve Environment Variables, thanks to a suggestion by Upayavira.