GroovyResult

Groovy Plugin Available

There is now a Struts 2 Groovy plugin that largely superceded this page: http://cwiki.apache.org/S2PLUGINS/groovy-plugin.html

GroovyResult - Groovy scripts as a view

This is an attempt to create a Result type that uses Groovy (http://groovy.codehaus.org) files as a view. It exposes the current ActionContext to a groovy script. This doesn't really have much practical use, but it's fun nonetheless and shows how easy it is to create a Result. There is another Result (JFreeChartResult) in the Cookbook

Installation

Not much - just make sure you have Groovy in your classpath, and the antlr, asm-* and groovy jars available to your webapp.

Configuration

acdtion.xml

```
<result-types>
   <result-type name="groovy" class="myapp.action2.extensions.GroovyResult"/>
</result-types>
```

xwork.xml - action definitions

```
<action name="MyAction" class="myapp.action2.actions.MyAction">
  <result name="success" type="groovy">
    <param name="file">test.groovy</param>
  </result>
</action>
```

The result type takes one parameter (for now), namely 'file', which contains the name of the groovy script in our script directory.

Show me the code !

Here's the code of the actual GroovyResult. This is a verbose version, with a lot of error checking.

- source code

GroovyResult.java

```
public class GroovyResult implements Result {

        public final static String GROOVY_DIR_NAME = "groovy";

        private final static Logger logger = Logger.getLogger(GroovyResult.class);
        //our groovy source file name
        private String file;
        //a groovy shell
        private GroovyShell shell;
        //our parsed script
        private Script script;
        //the outputstream that will replace the 'out' in our groovy stream
        private OutputStream out;
        //directory containing groovy scripts
        private String scriptDirectory;
        /*
         * (non-Javadoc)
         *
```

```
         * @see com.opensymphony.xwork.Result#execute(com.opensymphony.xwork.ActionInvocation)
         */
        public void execute(ActionInvocation inv) {

                //check the scriptDirectory - if it doesn't exists, use the default one
                //WEBAPP + Groovy files directory
                if (scriptDirectory == null) {
                        //not pretty, but this allows us to get the app root directory
                        String base = ServletActionContext.getServletContext().getRealPath(
                                        "/");
                        //if for some reason (.war, apache connector, ..) we can't get the
                        // base path
                        if (base == null) {
                                logger
                                                .warn("Could not translate the virtual path \"/\" to set
the default groovy script directory");
                                return;
                        }
                        scriptDirectory = base + GROOVY_DIR_NAME;
                        //issue a warning that this directory should NOT be world readable
                        // !!
                        logger
                                        .warn("Please make sure your script directory is NOT world
readable !");
                }

                // first of all, make sure our groovy file exists, is readable, and is
                // an actual file

                File groovyFile = new File(scriptDirectory, file);
                if (!groovyFile.exists()) {
                        //log an error and return
                        logger.warn("Could not find destination groovy file: "
                                        + groovyFile.getAbsolutePath());
                        return;
                }
                if (!groovyFile.isFile()) {
                        //log an error and return
                        logger.warn("Destination is not a file: "
                                        + groovyFile.getAbsolutePath());
                        return;
                }
                if (!groovyFile.canRead()) {
                        //log an error and return
                        logger.warn("Can not read file: " + groovyFile.getAbsolutePath());
                        return;
                }

                if (logger.isDebugEnabled())
                        logger.debug("File " + groovyFile.getPath()
                                        + " found, going to parse it ..");

                /*
                 * Here we create a Binding object which we populate with the webwork
                 * stack
                 */
                Binding binding = new Binding();

                binding.setVariable("context", ActionContext.getContext());

                /*
                 * We replace the standard OutputStream with our own, in this case the
                 * OutputStream from our httpResponse
                 */
                try {
                        //the out will be stored in an OutputStream
                        out = ServletActionContext.getResponse().getOutputStream();
                } catch (IOException e1) {
                        logger.error("Could not open outputstream", e1);
                }
                if (out != null){
```

```
                        binding.setVariable("out", out);
                }
                else {
                        logger
                                        .warn("OutputStream not available, using default System.out
instead");
                        binding.setVariable("out", System.out);
                }

                //create a new shell to parse and run our groovy file
                shell = new GroovyShell(binding);
                try {
                        //try to parse the script - the returned script could be cached for
                        //performance improvent
                        script = shell.parse(groovyFile);
                } catch (CompilationFailedException e) {
                        logger.error("Could not parse groovy script", e);
                        return;
                } catch (IOException e) {
                        logger.error("Error reading groovy script", e);
                        return;
                }
                //the binding is set, now run the script
                Object result = script.run();

                if (logger.isDebugEnabled()) {
                        logger.debug("Script " + groovyFile.getName()
                                        + " executed, and returned: " + result);
                }
                try {
                        out.flush();
                } catch (IOException e2) {
                        logger.error("Could not flush the outputstream", e2);
                }
        }

        /**
         * @return Returns the script.
         */
        public Script getScript() {
                return script;
        }
        /**
         * @param file
         *            The file to set.
         */
        public void setFile(String file) {
                this.file = file;
        }
        /**
         * @param out
         *            The out to set.
         */
        public void setOut(OutputStream out) {
                this.out = out;
        }
```

Explanation

The first part of the result is little more than:

- determining the script directory - defaults to `MYWEBAPP/groovy/`
- checking the file - make sure it exists, is readable, ..

> ⚠ Make sure the groovy scripts directory is not world readable !

The groovy part

```
Binding binding = new Binding();
binding.setVariable("context", ActionContext.getContext());
```

A Binding object allows us to 'bind' objects to a groovy script, so they can be used as variables. In this case, I took the ActionContext and exposed it as 'context'.

```
out = ServletActionContext.getResponse().getOutputStream();
...
binding.setVariable("out", out);
```

We also bind an OutputStream to the groovy script (as 'out') - it simply serves as a replacement for the standard System.out, so any printing goes directly to the http response outputstream.

```
shell = new GroovyShell(binding);
```

Next step; we create a GroovyShell, and pass our populated Binding to the constructor. Any script ran by this shell will have access to the passed variables (ActionContext and OutputStream).

```
script = shell.parse(groovyFile);
```

Before you can run a groovyFile, you need to parse it. Any syntax errors will be reported here - I also suggest adding a better error reporting in this case if you actually want to use this Result.
Upon successful parsing, a Script is returned (which could be cached if you want to increase performance) which will be run by our Shell.

```
Object result = script.run();
```

As a test, you might want to create a little 'groovy' script to test our Result.
test.groovy - a simple groovy script

```
for (item in context.contextMap){
        println "item: ${item}"
}
```

Place the test.groovy file in your groovy scripts directory. You should now see the result when you invoke MyAction.action in your browser.

Possible improvements are binding all objects on the stack so they become available to the groovy script, refactoring to an InputStream instead of a File, etc .. Comments welcome !