VelocityAndWeblogic

Using Velocity With WebLogic

Paw Dybdahl (pdy@csg.csc.dk)

Contents

Assumptions

This short guide can only be kept short because it assumes you already have a working knowledge of Velocity and how to use it in the J2EE environment, for example with Apache-Tomcat.

The Scope Of This Guide

Deploying web-applications to Tomcat can be accomplished by a war-file, but at startup Tomcat explodes the war-file so the directory structure are available for the Velocity engine at runtime and the `getRealPath()` method in `ServletContext` returns the real path to the given directory.

Deploying the same war-file in WebLogic will not have the same effect, since WebLogic keeps its deployed war-files unexploded, and the `getRealPath()` method in `ServletContext` returns `null`. So how should you organize your properties file, templates, stylesheets, servlets and build process in this environment?

The scope of this little guide is to give you answers to these questions, so you can enjoy using Velocity even in the context of a commercial app server.

While all the relevant information can be found in the Javadoc documentation, this quick guide will help you get started quickly.

Where to Put the Template Files

Since the `ClasspathResourceLoader` can load a template from a resource found in the servlet engines classpath, it seems like a very good idea to jar all the template-files together and place this jar (`template.jar` for example) in `WEB-INF/lib` which will make it available as a resource in the classpath.

Setting the Configuration Properties

Although the properties file could be given any name, for this guide we will use `velocity.properties` since all documentation and examples on the Velocity site uses this filename.

This file should include the following two lines to configure the resource loader:

```
    resource.loader = class
    class.resource.loader.class = org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader
```

The file can contain other configurations as well.

Where to put velocity.properties?

Put the `velocity.properties` file directly in the `WEB-INF` directory and include the following lines in your `web.xml`:

```
<servlet>
  <servlet-name>...</servlet-name>
  <servlet-class>...</servlet-class>
    <init-param>
      <param-name>properties</param-name>
      <param-value>/WEB-INF/velocity.properties</param-value>
    </init-param>
</servlet>
```

and include the following implementation in your servlet(s):

```
  protected Properties loadConfiguration (ServletConfig config) throws IOException, FileNotFoundException  {
    String propsFile = config.getInitParameter(INIT_PROPS_KEY);
    Properties p = new Properties();

    if ( propsFile != null ) {
      InputStream iStream = getServletContext().getResourceAsStream( propsFile );

      if ( iStream != null ) {
        p.load( iStream );
      }
    }

    return p;
  }
```

What About Stylesheets?

If you have any stylesheets you can place them anywhere you like, as long as you only make relative references to these files from your template files.

I prefer to have my stylesheets keept in a directory called `stylesheets`
located under the root-directory of the web application (i.e. at the same level as the `WEB-INF` directory).

In your template files you can reference the stylesheets like this:

```
<link REL="stylesheet" TYPE="text/css" HREF="./stylesheets/style.css">
```

Where to Put the Velocity Jar

First of all you have to decide whether you will use the dependency-free version of velocity.jar or the version including all dependend jars. If you are not worried about collisions with Avalon Logkit, Commons Collections or Jakarta Oro, using the jar containing all dependencies is very convenient.

Putting the velocity jar in `WEB-INF/lib` your web application will result in it's classes being available in the classpath for that web application.

Making Life Easier for the Developer

Using Velocity in 'real life' you quickly recognize a lot of similarity between your servlets. The first abstraction that will make life a little easier for the developers is a servlet like my `MainServlet`, which all application specific servlets can subclass. The only method the subclasses must override is the `getTemplateName()` method, which should return the name of this servlets template file.

If the subclasses needs to put data into context (which most servlets do) they can implement the `loadData()` method.

That's it. The rest is done by `MainServlet` (and `VelocityServlet` of course).

```
public abstract class MainServlet extends VelocityServlet {

  public Template handleRequest(HttpServletRequest request, HttpServletResponse response, Context ctx) {
    loadData( request, response, ctx );
    return getMyTemplate();
  }

  protected Properties loadConfiguration (ServletConfig config) throws IOException, FileNotFoundException  {
    String propsFile = config.getInitParameter(INIT_PROPS_KEY);
    Properties p = new Properties();

    if ( propsFile != null ) {
      InputStream iStream = getServletContext().getResourceAsStream( propsFile );

      if ( iStream != null ) {
        p.load( iStream );
      }
    }

    return p;
  }

  protected Template getMyTemplate( ) {
    Template template = null;
    try {
      template =  getTemplate( getTemplateName() + ".vm" );
    }
    catch (ParseErrorException pee) {
      mylog("Parse error for template " + pee);
    }
    catch (ResourceNotFoundException rnfe) {
      mylog("Template not found " + rnfe);
    }
    catch (Exception e) {
      mylog("Error " + e);
    }
    return template;
  }

  /**
   * Gets the name of the template file to be processed from this servlet. Should be
   * overridden by every subclass.
   */
  abstract protected String getTemplateName();

  /**
   * Load data into context.
   */
  protected void loadData(HttpServletRequest request, HttpServletResponse response, Context ctx ) {
    ctx.put( "dummy", "dummy" );
  }
}
```

A Build Process

A simple build process that supports the outlined directory contains the following steps:

1. Define shortcuts for the directories used during the build process 2. Prepare the build by coping all necessary files to the staging (build) directory 3. Compile all java files 4. Make a jar containing all templates 5. Make a war containing the web application (in the distribution directory)

A concrete example following the above scheme:

```
<project name="carImportAdmin" default="all" basedir=".">

  <!--  Source directory for deployment descriptors, manifest, properties,...  -->
  <property name="dd" value="./dd"/>
```

```
<!--  Work directory used during build process  -->
<property name="build" value="./staging"/>

<!--  Target directory for the final war-file  -->
<property name="dist" value="../build-ear/modules"/>

<!--  Source directory for all java files  -->
<property name="src" value="./src"/>

<!--  Source directory for template files  -->
<property name="templates" value="./templates"/>

<!--  Source directory for stylesheets  -->
<property name="stylesheets" value="./stylesheets"/>

<!--  Libraries used in compile  -->
<property name="lib" value="${dist}/CarImport.jar;${dist}/FPGuiden.jar;${dist}/velocity-dep-1.2-rc3.jar"/>


<target name="all" depends="init, compile, jar, war"/>

<target name="init" depends="clean">
  <tstamp/>

  <mkdir dir="${build}"/>
  <mkdir dir="${build}/stylesheets"/>
  <mkdir dir="${build}/WEB-INF"/>
  <mkdir dir="${build}/WEB-INF/classes"/>
  <mkdir dir="${build}/WEB-INF/lib"/>

  <copy todir="${build}/WEB-INF">
    <fileset dir="${dd}">
      <include name="velocity.properties"/>
    </fileset>
    <fileset dir="${dd}/WEB-INF">
      <include name="web.xml"/>
    </fileset>
  </copy>
  <copy todir="${build}/WEB-INF/lib">
    <fileset dir="${dist}">
      <include name="velocity-dep-1.2-rc3.jar"/>
    </fileset>
  </copy>
  <copy todir="${build}/stylesheets">
    <fileset dir="${stylesheets}">

    </fileset>
  </copy>
</target>


<target name="compile">
  <javac srcdir="${src}" destdir="${build}/WEB-INF/classes" classpath="${CLASSPATH};${lib}">
  </javac>
</target>

<target name="jar">
  <jar jarfile="${build}/WEB-INF/lib/templates.jar"
    basedir="${templates}">
  </jar>
</target>

<target name="war" depends="init">
  <jar jarfile="${dist}/carImportAdmin.war"
    basedir="${build}">
  </jar>
</target>

<target name="clean">
  <delete dir="${build}"/>
</target>
```

```
</project>
```