VelocityAndSpringStepByStep

This page contains supplemental information to the Developing a Spring Framework MVC application step-by-step article by Thomas Risberg, showing the changes needed to use Velocity instead of JSP.

Part 1 - Basic Application and Environment Setup

Step 2 - index.jsp

Of course, we will be using Velocity, so let's name this file `index.vm`.

Step 4 - Test the application

Naturally, the URL is http://localhost:8080/springapp/index.vm

Step 6 - Modify web.xml in WEB-INF directory

`web.xml` : Change the welcome-file to...

```
<welcome-file-list>
  <welcome-file>
    index.vm
  </welcome-file>
</welcome-file-list>
```

Step 12 - Create a View

Again, this file will be `hello.vm`.

`SpringappController.java` : return the template.

```
   public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {

       logger.info("SpringappController - returning hello view");

       return new ModelAndView("hello.vm");
   }
```

Part 2 - Developing and Configuring the Application

Step 13 - Improve index.jsp

We won't be needing an `include.vm`, but let's make the `springapp/war/WEB-INF/velocity` directory anyways.

Add Velocity jar to `springapp/war/WEB-INF/lib`

`web.xml` : Change welcome file to `hello.htm` and add a servlet mapping to it

```
  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>/hello.htm</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>
      hello.htm
    </welcome-file>
  </welcome-file-list>
```

Now, we won't even need an `index.vm`. Thanks to Marcelo Bello for the post in bugzilla, and Maarten Bosteels for finding this. 🙂

Step 14 - Improve the view and the controller

Move the view `hello.vm` to the `WEB-INF/velocity` directory.

`springapp/war/WEB-INF/velocity/hello.vm` : Use VTL instead of JSTL

```
<p>Greetings, it is now ${now}
```

`springapp/src/SpringappController.java` : Change the view to `WEB-INF/velocity/hello.vm`.

```
        return new ModelAndView("WEB-INF/velocity/hello.vm", "now", now);
```

Those of the impatient ones should now be wondering, "How come the ${now} doesn't get replaced by the time?". That is because Spring does not know of Velocity yet. So, we'll need to go ahead and add Velocity settings to the `springapp-servlet.xml`.

`springapp/war/WEB-INF/springapp-servlet.xml` : Add Velocity settings

```
    <bean id="velocityConfig" class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">
        <property name="resourceLoaderPath">
            <value>/</value>
        </property>
    </bean>

    <bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityViewResolver">
    </bean>
```

Now, you should be able to see the same screen.

Step 15 - Decouple the view and the controller

From this step, the Controller class is the same for JSP and Velocity. Isn't that wonderful?

`springapp/war/WEB-INF/springapp-servlet.xml`

```
    <bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityViewResolver">
        <property name="prefix"><value>/WEB-INF/velocity/</value></property>
        <property name="suffix"><value>.vm</value></property>
    </bean>
```

Step 17 - Modify the view to display business data and add support for message bundle

`springapp/war/WEB-INF/velocity/hello.vm` : VTL instead of JSTL

```
<html>
<head><title>#springMessage("title")</title></head>
<body>
<h1>#springMessage("heading")</h1>
<p>#springMessage("greeting") ${model.now}
</p>
<h3>Products</h3>
#foreach ($prod in ${model.products})
  ${prod.description} <i>${prod.price}</i><br><br>
#end
</body>
</html>
```

In order to enable the Spring Velocimacros, we need to set the "exposeSpringMacroHelpers" property to "true" for `org.springframework.web.servlet.view.velocity.VelocityViewResolver`.

`springapp/war/WEB-INF/springapp-servlet.xml`

```
    <bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityViewResolver">
        <property name="prefix"><value>/WEB-INF/velocity/</value></property>
        <property name="suffix"><value>.vm</value></property>
        <property name="exposeSpringMacroHelpers"><value>true</value></property>
    </bean>
```

Part 3 - Adding Unit Tests and a Form to the Application

Step 22 - Adding a form

As we are using Velocity, we won't need to add the taglib entry to `web.xml`.

`springapp/war/WEB-INF/jsp/priceincrease.vm` : Use VTL

```
<html>
<head><title>#springMessage("title")</title></head>
<body>
<h1>#springMessage("priceincrease.heading")</h1>
<form method="post">
  <table width="95%" bgcolor="f8f8ff" border="0" cellspacing="0" cellpadding="5">
    <tr>
      <td alignment="right" width="20%">Increase (%):</td>
      #springBind("priceIncrease.percentage")
        <td width="20%">
          <input type="text" name="percentage" value="${status.value}">
        </td>
        <td width="60%">
          <font color="red">${status.errorMessage}</font>
        </td>
    </tr>
  </table>
  <br>
  #springBind("priceIncrease.*")
  #if(${status.error})
    <b>Please fix all errors!</b>
  #end
  <br><br>
  <input type="submit" alignment="center" value="Execute">
</form>
<a href="#springUrl("/hello.htm")">Home</a>
</body>
</html>
```

`springapp/war/WEB-INF/jsp/hello.vm` : Use VTL

```
<br>
<a href="#springUrl("/priceincrease.htm")">Increase Prices</a>
<br>
```

A few things to note about the Spring Velocimacro.

- `#springBind` is not a block, hence there is no `#end` and the `$status` reference stays in the Context. See a JIRA issue about this.
- `#hasBindErrors` does not exist, so you need to use an alternative.
- `#springUrl` will not add a leading `/` to the URL, so you need to add it by yourself.

Part 4 - Implementing Database Persistence

Nothing to change. 🙂