FreeMarker

FreeMarker is a Java-based template engine that is a great alternative to JSP. FreeMarker is ideal for situations where your action results can possibly be loaded from outside a Servlet container. For example, if you wished to support plugins in your application, you might wish to use FreeMarker so that the plugins could provide the entire action class and view in a single jar that is loaded from the classloader.

For more information on FreeMarker itself, please visit the FreeMarker website.

The framework utilizes FreeMarker because the engine includes strong error reporting, built-in internationalization and powerful macro libraries.

Support is also included for Velocity templates. For a comparison of Velocity vs FreeMarker see here.

Getting Started

Getting started with FreeMarker is as simple as ensuring all the dependencies are included in your project's classpath. Typically, the only dependency is `freemarker.jar`. Other than that, struts-default.xml already configures the FreeMarker Result needed to process your application's templates.

struts.xmlxml<action name="test" class="com.acme.TestAction"> <result name="success" type="freemarker">test-success.ftl</result> </action>

Then in `test-success.ftl`:

test-success.ftlxml<html> <head> <title>Hello</title> </head> <body> Hello, ${name} </body> </html>

Where `name` is a property on your action. That's it! Read the rest of this document for details on how templates are loaded, variables are resolved, and tags can be used.

Servlet / JSP Scoped Objects

The following are ways to obtained Application scope attributes, Session scope attributes, Request scope attributes, Request parameters, and framework Context scope parameters:-

Application Scope Attribute

Assuming there's an attribute with name `myApplicationAttribute` in the Application scope.

<#if Application.myApplicationAttribute?exists> ${Application.myApplicationAttribute} </#if>

or

<@s.property value="%{#application.myApplicationAttribute}" />

Session Scope Attribute

Assuming there's an attribute with name `mySessionAttribute` in the Session scope.

<#if Session.mySessionAttribute?exists> ${Session.mySessionAttribute} </#if>

or

<@s.property value="%{#session.mySessionAttribute}" />

Request Scope Attribute

Assuming there's an attribute with name 'myRequestAttribute' in the Request scope.

<#if Request.myRequestAttribute?exists> ${Request.myRequestAttribute} </#if>

or

<@s.property value="%{#request.myRequestAttribute}" />

Request Parameter

Assuming there's a request parameter myParameter (eg. http://host/myApp/myAction.action?myParameter=one).

<#if Parameters.myParameter?exists> ${Parameters.myParameter} </#if>

or

<@s.property value="%{#parameters.myParameter}" />

Context parameter

Assuming there's a parameter with the name myContextParam in framework context.

${stack.findValue('#myContextParam')}

or

<@s.property value="%{#myContextParam}" />

Template Loading

The framework looks for FreeMarker templates in two locations (in this order):

1. Web application
2. Class path

This ordering makes it ideal for providing templates inside a fully-built jar, but allowing for overrides of those templates to be defined in your web application. In fact, this is how you can override the default UI tags and Form Tags included with the framework.

In addition, you can specify a location (directory on your file system) through the `templatePath` or `TemplatePath` context variable (in the `web.xml`). If a variable is specified, the content of the directory it points to will be searched first.

This variable is currently NOT relative to the root of your application.

Variable Resolution

When using FreeMarker with the framework, variables are looked up in several different places, in this order:

1. Built-in variables
2. Value stack
3. Action context
4. Request scope
5. Session scope
6. Application scope

Note that the action context is looked up after the value stack. This means that you can reference the variable without the typical preceding has marker (#) like you would have to when using the JSP `s:property` tag. This is a nice convenience, though be careful because there is a small chance it could trip you up.

xml<@s.url id="url" value="http://www.yahoo.com"/> Click here!

The built-in variables that Struts-FreeMarker integration provides are:

Name	Description
stack	The value stack itself, useful for calls like ${stack.findString('ognl expr')}
action	The action most recently executed
response	The HttpServletResponse
res	Same as response
request	The HttpServletRequest
req	Same as request
session	The HttpSession
application	The ServletContext
base	The request's context path

Tag Support

FreeMarker includes complete tag support. See the FreeMarker Tags documentation for information on how to use the generic Struts Tags provided by Struts. In addition to this, you can use any JSP tag, like so:

xml<#assign mytag=JspTaglibs["/WEB-INF/mytag.tld"]> <@mytag.tagx attribute1="some ${value}"/>

Where **mytag.tld** is the JSP Tag Library Definition file for your tag library. Note: in order to use this support in FreeMarker, you must enable the `JSPSuppo rtServlet` in `web.xml`:

Adding JspSupportSerlvet to web.xmlxml<servlet> <servlet-name>JspSupportServlet</servlet-name> <servlet-class>org.apache.struts2.views. JspSupportServlet</servlet-class> <load-on-startup>1</load-on-startup> </servlet>

Tips and Tricks

There are some advanced features that may be useful when building Struts applications with FreeMarker.

Type Conversion and Locales

FreeMarker has built in support for formatting dates and numbers. The formatting rules are based on the locale associated with the action request, which is by default set in struts.properties but can be over-ridden using the I18n Interceptor. This is normally perfect for your needs, but it is important to remember that these formatting rules are handled by FreeMarker and not by the framework's Type Conversion support.

If you want the framework to handle the formatting according to the Type Conversion you have specified, you shouldn't use the normal ${...} syntax. Instead, you should use the property tag. The difference is that the property tag is specifically designed to take an OGNL expression, evaluate it, and then convert it to a String using any Type Conversion rules you have specified. The normal ${...} syntax will use a FreeMarker expression language, evaluate it, and then convert it to a String using the built in formatting rules.

⚠️ The difference in how type conversion is handled under Freemarker is subtle but important to understand.

Extending

Sometimes you may with to extend the framework's FreeMarker support. For example, you might want to extend the Struts tags that come bundled with the framework.

To extend the Freemarker support, develop a class that extends `org.apache.struts2.views.freemarker.FreemarkerManager`, overriding methods as needed, and plugin the class through the struts.properties:

nonestruts.freemarker.manager.classname = com.yourcompany.YourFreeMarkerManager

ObjectWrapper Settings

Once you get familiar with FreeMarker, you will find certain *subtleties* with it that may become frustrating. The most common thing you'll likely run in to is the BeansWrapper provided by FreeMarker. If you don't know what this is, don't worry. However, if you do, know this:INLINE{snippet: id=javadoc|javadoc=true|url=org.apache.struts2.views.freemarker.StrutsBeanWrapper}

Syntax Notes

As of FreeMarker 2.3.4, an alternative syntax is supported. This alternative syntax is great if you find that your IDE (especially IntelliJ IDEA) makes it difficult to work with the default syntax. You can read more about this syntax here.

Cache

You can enable FreeMarker cache mechanism by specifying below options in struts.xml:

- `<constant name="struts.freemarker.mru.max.strong.size" value="250" />` - this option will be used by freemarker.cache.MruCacheStorage
- `<constant name="struts.freemarker.templatesCache.updateDelay" value="1800" />` - default update cache interval (5 seconds)
- `<constant name="struts.freemarker.templatesCache" value="true" />` - ***DEPRECATED*** this option will use a internal ConcurrentHashMap in FreemarkerTemplateEngine but not freemarker native cache

Setting `devMode` to true will disable cache and updateDelay immediately, but you can explicit specify these constants to enable cache even in `devMode`, see devMode

Incompatible Improvements

By default Struts is using FreeMarker in way to be backward compatible as much as possible but if you need to enable new features you can do it via `free marker.properties` by defining incompatible improvements settings, ie.:

freemarker.propertiesincompatible_improvements=2.3.22

You can also pass this setting via `ServletContext` `<init-param/> (since Struts 2.5.13):`

xml<init-param> <param-name>freemarker.incompatible_improvements</param-name> <param-value>2.3.22</param-value> </init-param>

This can impact your freemarker powered pages and Struts tags as well, so please careful test this change.

Next: FreeMarker Tags