How-to use spring configurations to extend Jahia

Jahia's next-generation, open source CMS stems from a widely acknowledged vision of enterprise application convergence – web, document, search, social and portal – unified by the simplicity of web content management.

Summary

1 Introduction

In Jahia 6.5, you can now embed spring configurations directly in your modules without modifying the core application files. This approach provides two major benefits: reusability (you can reuse your module from project to project) and safer upgrade path (a Jahia upgrade doesn't impact the non Jahia modules). The goal of this documentation is to show some examples of what you can do with those spring files.

1.1 Pre-requisites

- Have a running Maven configuration
- Have a Java IDE that supports Maven (Eclipse + M2 / IntelliJ / Netbeans / …)

1.2 Generate your module

In command line, enter:

mvn archetype:generate -DarchetypeCatalog=http://maven.jahia.org/maven2

When prompted, choose:

3: http://maven.jahia.org/maven2 -> jahia-module-archetype (Jahia archetype for creating a new module (for Jahia version >= 6.5))

Then fulfill the different values :

artifactId : springDemo

jahiaPackageVersion : 6.5.0

moduleName : Spring Demo

Your module is now generated. It is a maven project than you can open in your IDE. In the webapp /META-INF/spring folder, you a file springDemo.xml.disabled. You can rename it to springDemo.xml to start playing with spring configuration.
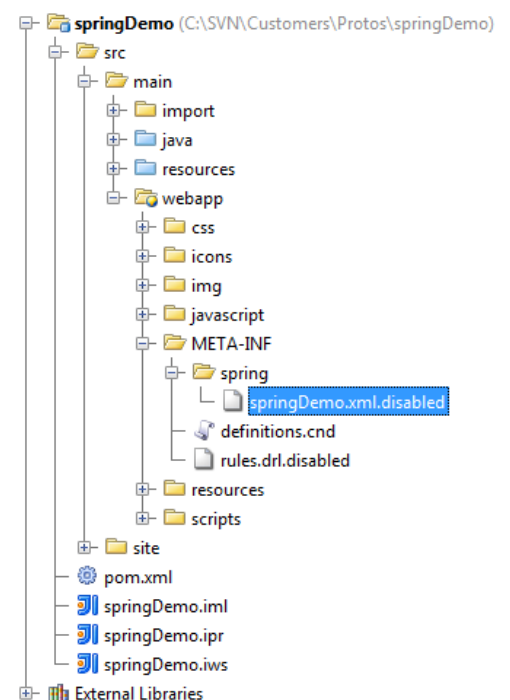
1.3   Use spring to expose some java bean to Jahia

Jahia has its own internal behavior. But sometimes we want to complete it… An elegant way to do that is to use spring configuration.

1.3.1 Filters

At the hearth of Jahia is the mechanism of filtering. During the rendering of web pages, content grabbed from the JCR goes through several consecutive filters (like a chain) that operates transformations or internal operations (like putting fragments in cache for instance).

As a developer, you can add your own filters to add your own operations. One of the main interests is that the content delivered to the visitors is modified according to the filters it went through but remains clean and untouched in the repository.

For example let's assume you want to add a new filter in the render chain of Jahia, to encrypt e-mail addresses on the fly. For that you implement your Java class (see the related documentation about filters for more details), and then you need to include it in the render chain.

The render chain configuration is in applicationcontext-renderer.xml in the WEB-INF/etc/spring/ folder of Jahia. You could directly modify this file to reference your bean, but thanks to the XML configuration directly embed in modules, you can put this bean in the springDemo.xml file :

```xml
<bean class="org.jahia.services.render.filter.FilterExample">
    <property name="priority" value="21" />
</bean>
```

As you can see, we must specify a priority to define the ordering of execution. When you put all beans in the same configuration file, the order is defined but the position of beans, but when they are exploded into several configuration files, a position is mandatory.

That's all; it is enough to reference your filter in the render chain. It means that you now have an independent module which brings the requested configuration to work. If you don't deploy the module, you don't have useless configuration that could lead to errors. If you deploy your module, you have the

guarantee that everything is well configured and you don't have to put a readme file "please modify the applicationcontext-renderer.xml" which usually leads to errors or defaults.

You can train yourself and create your first filters following the Howto_use_filters.pdf

You can learn more about filters here:

http://www.jahia.com/community/documentation/jahiapedia/jahia-modules/renderingFilter.html

1.3.2 Actions

We can use the same mechanism to reference new **actions**. For example, we want to be able to push some data to an external system when submitting a form. For that we will implement a Java class which extends action (see the related documentation for more details) and then we need to put some configuration to be able to call urls like:

http://localhost:8080/mypath/mypage.pushData.do

```jsp
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="ui" uri="http://www.jahia.org/tags/uiComponentsLib" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="template" uri="http://www.jahia.org/tags/templateLib" %>
<%@ taglib prefix="jcr" uri="http://www.jahia.org/tags/jcr" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

<c:url value="${url.base}${currentNode.path}.pushData.do" var="url" />

<form class="Form" id="externalDataPostForm" name="externalDataPostForm" method="POST" action="${url}">
    [...]
</form>
```

So we edit the springDemo.xml file and put inside the bean, with the property "name" that will be used as keyword in the URL to call the action :

```xml
<bean class="org.jahia.modules.springDemo.actions.PushData">
    <property name="name" value="pushData"/>
</bean>
```

That's all! As soon as your module is deployed, your action becomes available.

You can learn more about actions here:

http://www.jahia.com/community/documentation/jahiapedia/actions/actions.html

1.3.3 Choice List Initializers

In your definitions, you sometimes need a choice list initializer. As usual, it is a java class to implement (see the related documentation), and you have to reference it in the spring configuration to be able to use it.

[jnt :myComponent] > jnt :content

- selectList (string, choicelist[bannerlinktype])

So we edit the springDemo.xml file and put inside the bean, with the property "key" that will be used as keyword in the definition files to call the initializer:

```
<bean id="bannerlinktypechoicelist" class="org.jahia.services.content.nodetypes.initializers.BannerContent
    <property name="key" value="bannerlinktype"/>
</bean>
```

Your can learn more about choiceList initializers here:

http://www.jahia.com/community/documentation/jahiapedia/initializers_renderers/initializers.html

Jahia Solutions Group SA

9 route des Jeunes,
CH-1227 Les acacias
Geneva, Switzerland

http://*www.jahia.com*