Services web REST - Externalisation de la configuration

La configuration des services web REST est assurée par défaut par le fichier *WEB-INF/beans.xml*.
Ce fichier de configuration *Spring* déclare des éléments **<bean>** paramétrables à l'aide de sous-éléments **<property>**.
La totalité ou une partie des valeurs des propriétés peuvent être regroupées dans un fichier afin d'être externalisées via une ressource JNDI.

Exemple de fichier *beans.xml* avec des propriétés dont les valeurs sont externalisées via une ressource JNDI nommée "java:comp/env/url/jettisonConf"
Pour externaliser une valeur, il faut affecter à l'attribut **value** la valeur : **"${identifiant}"**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jaxrs="http://cxf.apache.org/jaxrs"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:cxf="http://cxf.apache.org/core"
   xmlns:jee="http://www.springframework.org/schema/jee"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans
/spring-beans-3.0.xsd
   http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd
   http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-
3.0.xsd
   http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
   http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd">

   <bean id="conversionService" class="org.springframework.context.support.ConversionServiceFactoryBean" >
    <property name="converters">
        <set>
            <bean id="stringToMapConverter" class="com.hardis.common.StringToMapConverter" />
        </set>
    </property>
  </bean>

  <jee:jndi-lookup id="jettisonConfProperties" jndi-name="java:comp/env/url/jettisonConf" resource-ref="true"/>

  <bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
depends-on="jettisonConfProperties">
    <property name="ignoreUnresolvablePlaceholders" value="true"/>
    <property name="properties" ref="jettisonConfProperties"/>
  </bean>

<bean class="org.apache.cxf.jaxrs.provider.json.JSONProvider">
          <property name="dropRootElement" value="${jettisonProvider.dropRootElement}"/>
          <property name="serializeAsArray" value="${jettisonProvider.serializeAsArray}"/>
          <property name="arrayKeys" value="${jettisonProvider.arrayKeys}"/>
          <property name="supportUnwrapped" value="${jettisonProvider.supportUnwrapped}"/>
    </bean>
</beans>
```

Pour l'externalisation des propriétés, le fichier *beans.xml* ci-dessus déclare les éléments suivants :

- <**jee:jndi-lookup** id="jettisonConfProperties" jndi-name="**java:comp/env/url/jettisonConf**" resource-ref="true"/>
  Permet de désigner le nom de la ressource JNDI utilisée pour la recherche du fichier externalisé.

- <**bean** id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer" depends-on="
  jettisonConfProperties">
  <property name="ignoreUnresolvablePlaceholders" value="true"/>
  <property name="properties" ref="jettisonConfProperties"/>
  </bean>
  Permet de déclarer un objet *PropertyPlaceholderConfigurer* s'appuyant sur le fichier externalisé des valeurs propriétés pour la substitution dans le fichier *beans.xml*.

- <**bean** id="conversionService" class="org.springframework.context.support.ConversionServiceFactoryBean" >
  <property name="converters">
  <set>
  <bean id="stringToMapConverter" class="com.hardis.common.StringToMapConverter" />
  </set>
  </property>
  </bean>
  Permet d'activer par défaut les convertisseurs nécessaires pour l'évaluation des valeurs des propriétés et déclare un convertisseur spécifique d'une *String* vers un objet *Map*.

Exemple (convertisseur *[String List]*, une chaîne composée de plusieurs chaînes séparées par une virgule est convertie en un objet *List*), :
...

<property name="arrayKeys" value="**${jettisonProvider.arrayKeys}**"/>
...

jettisonProvider.arrayKeys=*implantations*, *products*

Equivaut à :

<property name="**arrayKeys**">
  <list>
    <value>*implantations*</value>
    <value>*products*</value>
  </list>
</property>

L'ensemble des couples (identifiant, valeur) sont regroupés dans un fichier .properties.
Exemple : jettisonConf.properties

```
jettisonProvider.dropRootElement=true
jettisonProvider.serializeAsArray=true
jettisonProvider.supportUnwrapped=true
jettisonProvider.arrayKeys=implantations, products
```

Exemple d'une propriété avec pour valeur une *map* d'éléments :
JWTADELIALOGINMODULE.adeliaParameters=[VAAuthProgram:com.hardis.adelia.HFAUWS][SSOEnabled:false][Unicode:true]

Equivaut à :

<property name="**adeliaParameters**">
  <map>
    <entry key="VAAuthProgram" value="com.hardis.adelia.HFAUWS" />
    <entry key="SSOEnabled" value="false" />
    <entry key="Unicode" value="true" />
  </map>
</property>

Déclaration de la ressource JNDI de façon globale ou propre au contexte de l'application web dans la configuration de Tomcat ($TOMCAT_HOME/conf /server.xml ou $TOMCAT_HOME/conf/Catalina/localhost/contextRoot.xml).

Exemple : Déclaration de la ressource JNDI propre au contexte de l'application nommée *mywebapp*.
Le nom de la ressource JNDI est : **url/jettisonConf**
L'emplacement du fichier externalisé des propriétés est : **c:/conf/jettisonConf.properties**

```
<Context path="/mywebapp" docBase="D:\webapps\mywebapp">
   <Resource auth="Container" factory="com.hardis.common.JndiURLPropsFactory" name="url/jettisonConf" type="
java.net.URL" url="file:///c:/conf/jettisonConf.properties"/>
</Context>
```

Remarques :

1. Il est possible de scinder la configuration dans plusieurs fichiers properties afin d'assurer un découpage fonctionnel.
   Par exemple, si une même application joue le rôle de serveur de jetons et de serveur de services, il est possible d'avoir un fichier de configuration propre à la production de jetons et un autre fichier de configuration propre à l'exposition des services.
   Dans ce découpage, 2 ressources JNDI devront être créées, le fichier beans.xml devant alors déclarer 2 <jndi-lookup> et 2 <propertyConfigurer>.

2. Lorsque les valeurs sont externalisées, il faut faire attention à :
   a. Remplacer les entités XML par leurs propres : *&* devient *&*
   b. Doubler les anti-slash : *hardis\domaine* devient *hardis\\domaine*