Services web REST - Plusieurs configurations au sein d'une même application web

Les services REST Adélia s'appuient d'une part sur le framework CXF pour leur implémentation (jaxrs 2.0) et d'autre part sur le framework Spring pour leur configuration.
Il est - sous certaines conditions - possible d'avoir une configuration différente pour 2 (ou n) jeux de services/d'APIs REST : le 1er jeu peut par exemple utiliser une sérialisation *jettison* et le second jeu une sérialisation *jackson*.

La distribution d'un jeu d'Apis REST vers une configuration spécifique peut se faire de 2 façons.

A. Déclaration d'une servlet CXF spécifique à un jeu d'APIs (utilisation d'un fichier de configuration Spring *beans.xml* dédié).

web.xml (extrait)

```
<servlet>
    <servlet-name>CXFServlet1</servlet-name>
    <display-name>CXF Servlet1</display-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <init-param>
      <param-name>config-location</param-name>
      <param-value>/WEB-INF/beans1.xml</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>CXFServlet2</servlet-name>
    <display-name>CXF Servlet2</display-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <init-param>
      <param-name>config-location</param-name>
      <param-value>/WEB-INF/beans2.xml</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>CXFServlet1</servlet-name>
    <url-pattern>/ws1/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>CXFServlet2</servlet-name>
    <url-pattern>/ws2/*</url-pattern>
  </servlet-mapping>
```

La servlet CXFServlet1 utilise la configuration WEB-INF/beans1.xml et son servlet-mapping est /ws1/*
La servlet CXFServlet2 utilise la configuration WEB-INF/beans2.xml et son servlet-mapping est /ws2/*
La servlet CXFServletN utilise la configuration WEB-INF/beansN.xml et son servlet-mapping est /wsN/*

beans1.xml

```
<context:component-scan base-package="com.hardis.adelia.webservice,hardis.fr" />

<jaxrs:server id="SRVRS1" address="/" transportId="http://cxf.apache.org/transports/http">
</jaxrs:server>
```

beans2.xml

```
<context:component-scan base-package="com.hardis.adelia.webservice,svlet2.fr" />
<bean class="org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider"/>

<jaxrs:server id="SRVRS2" address="/" transportId="http://cxf.apache.org/transports/http">
</jaxrs:server>
```

Remarques :

- Ce mode permet d'utiliser le *context:component-scan* de Spring et permet ainsi d'enregistrer de façon automatique tous les services (sans avoir à les nommer) présents dans la liste de package *base-package* et tous les composants annexes (feature, provider, etc.).
  Pour scinder les services en n jeux distincts, il faut donc utiliser **n packages différents** (*hardis.fr* et *svlet2.fr* dans l'exemple).
- Le fichier *beans2.xml,* en déclarant un bean faisant référence à la classe *JacksonJaxbJsonProvider* force CXF à utiliser la librairie jackson pour la sérialisation des messages.
- Aujourd'hui la déclaration du fichier de configuration spring *beans.xml* est faite dans un *web-fragment* placé dans un .jar. Il faut par conséquent conserver (en plus des fichiers beans1, beans2...beansn) un fichier quasiment vide (<beans/>) nommé *beans.xml*.
- Les n jeux d'APIs sont accessibles via un préfixe d'URL différent défini par le couple (<contextRoot>, <CXFServletMapping>) => /contextRoot /ws1/... pour le 1er jeu, /contextRoot/ws2/... pour le second, etc.

B. Déclaration de N <jaxrs:server> dans un même fichier de configuration Spring *beans.xml*.

Il est possible de conserver une seule servlet CXF et un unique fichier de configuration *beans.xml*. En revanche il n'est alors plus possible d'utiliser le *conte xt:component-scan* de Spring.
La distribution des différents jeux d'APIs vers des endpoints distincts, des configurations spécifiques, se fait à l'aide de l'élément <*jaxrs:server*> avec l'attribution d'une *address* et des propriétés propres.

bean.xml (extrait)

```
<jaxrs:server id="Rest1" address="/jrs1" transportId="http://cxf.apache.org/transports/http">
  <jaxrs:serviceBeans>
     <bean class="hardis.fr.DJRS1AdeliaService" />
  </jaxrs:serviceBeans>
</jaxrs:server>

<jaxrs:server id="Rest2" address="/jrs2" transportId="http://cxf.apache.org/transports/http">
   <jaxrs:serviceBeans>
      <bean class="svlet2.fr.DJRS2AdeliaService" />
   </jaxrs:serviceBeans>
   <jaxrs:providers>
      <bean class="org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider" />
   </jaxrs:providers>
</jaxrs:server>
```

Remarques :

- Ce mode ne permet plus d'utiliser le *context:component-scan* de Spring. Par conséquent il n'est plus nécessaire d'utiliser N packages différents mais il faut déclarer explicitement les programmes de services (Cf. *jaxrs:serviceBeans*), les providers (Cf. *jaxrs:providers*), les features (Cf. *jaxrs: features*) pour chaque *jaxrs:server*.
- Les N jeux d'APIs sont accessibles via un préfixe d'URL différent défini par le 3-uplet (<contextRoot>, <CFXServletMapping>, <address>), => /contextRoot/CXFServletMapping/**jrs1**... pour le 1er jeu, /contextRoot/CXFServletMapping/**jrs2**/... pour le second, etc.
- L'élément <jaxrs:server> *Rest2* déclare un *provider* faisant référence à la classe *JacksonJaxbJsonProvider* ce qui force l'utilisation de la librairie jackson pour la sérialisation des messages dans le cas du endpoint /contextRoot/CXFServletMapping/**jrs2**.

SWAGGER

La feature Swagger2Feature de CXF permet dans le cas d'utilisation du *context:component-scan* de filtrer les Apis à présenter sur un ensemble de packages (dans l'exemple : com.hardis.jaxrs1 et com.hardis.jaxrs2) :

beans.xml

```
<bean id="swagger2Feature" class="org.apache.cxf.jaxrs.swagger.Swagger2Feature" lazy-init="true"
>
   <property name="resourcePackage" value="com.hardis.jaxrs1, com.hardis.jaxrs2"/>
   <property name="supportSwaggerUi" value="true"/> <!-- à partir d'Adélia Studio 13 PTF09 -->
</bean>
```

Dans le cas de la déclaration de plusieurs *jaxrs:server*, il faut spécifier à swagger de créer un contexte spécifique au endpoint.

beans.xml (extrait)

```
 <bean id="swagger2Feature" class="org.apache.cxf.jaxrs.swagger.Swagger2Feature" lazy-init="true"
>
      <property name="supportSwaggerUi" value="true"/> <!-- à partir d'Adélia Studio 13 PTF09 -->
      <property name="usePathBasedConfig" value="true"/>
      <property name="scan" value="false"/>
 </bean>

<jaxrs:server id="Rest1" address="/jrs1" transportId="http://cxf.apache.org/transports/http">
  <jaxrs:serviceBeans>
     <bean class="hardis.fr.DJRS1AdeliaService" />
  </jaxrs:serviceBeans>
  <jaxrs:providers>
      <bean class="org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider" />
  </jaxrs:providers>
  <jaxrs:features>
      <ref bean="swagger2Feature" />
  </jaxrs:features>
</jaxrs:server>

<jaxrs:server id="Rest2" address="/jrs2" transportId="http://cxf.apache.org/transports/http">
   <jaxrs:serviceBeans>
      <bean class="svlet2.fr.DJRS2AdeliaService" />
   </jaxrs:serviceBeans>
   <jaxrs:providers>
      <bean class="org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider" />
   </jaxrs:providers>
   <jaxrs:features>
      <ref bean="swagger2Feature" />
   </jaxrs:features>
</jaxrs:server>
```

Remarques

- La propriété *supportSwaggerUi* (en commentaire dans l'exemple) est une propriété qui sera disponible dans Adélia Studio 13 PTF09 (version intégrant les dernières version de CXF et swaggerUi) permettant avec la seule présence de la librairie swagger-ui-3.x.y.jar d'avoir un accès à swaggerUi via l'URL http://<domain>:<port>/<ContextRoot>/<CXFServletMapping>/api-docs?url=/<RootContext>/<CXFServletMapping>/swagger.json