Web services REST avec Java (JAXB)

Hyacinthe MENIET

3 août 2019

Les Web services ont été créés dans le but de connecter des systèmes disparates, de définir des sources de données communes et de maximiser la re-utilisation du code. Concrètement, si vous considérez une entreprise dont le système d'information a plusieurs dizaines d'années derrière lui. Vous aurez rarement un unique fournisseur, un seul logiciel et un seul type de machine. De plus pour des raisons de coût ou plus simplement de concurrence, il est rarement indiqué de tout uniformiser.
C'est à ce stade qu'interviennent les Web services car ils permettent à des éléments d'un parc hétérogène d'interagir (e.g. échanger des données, exécuter des méthodes à distance ...) entre eux. Il y a trivialement deux technologies derrière le terme Services Web : Les Web services SOAP et les Web services REST dont je parle ici.
REST (REpresentational State Transfer) n'est ni un protocole, ni un format, c'est trivialement une description de la manière dont le Web fonctionne. Vous accédez à une ressource via son URI et vous pouvez changer l'état de cette ressource en utilisant les méthodes natives de HTTP (GET, POST, PUT et DELETE). Dans cet article je vais indiquer comment déployer un Web service REST via une simple Servlet. Puis j'expliquerai comment le consommer.

1. Pré-requis

- Vous êtes familier de Java 8 et de sa syntaxe.
- Vous êtes familier des Web services et idéalement webmaster.
- Vous disposez du JDK 8 minimum.
- Vous êtes familier de Tomcat et vous avez installé et configuré sa version 8.
- Vous êtes familier de Maven et disposez de sa version 3.3 minimum

2. Vue d'ensemble

2.1 Vue d'ensemble de REST

REST propose une vision orientée ressource des Web services. C'est à dire qu'avec REST les variables, ce sont les objets (ou ressources). Selon l'approche REST, chaque ressource est identifiable de manière non ambiguë et unique via son URI (Universal Resource Identifier) de la forme http://localhost:8080/weatherws/cities/307.
Avec REST pour une ressource donnée, vous proposez un groupe de services (ou opérations) de base. Idéalement toujours les mêmes : créer, consulter, modifier et supprimer (voir CRUD). Les opérations à réaliser sur ces ressources, coïncident avec les méthodes natives de HTTP :

HTTP	REST	Description
GET	consulter	Retourne une représentation XML de la ressource. L'état de la ressource est inchangé.
POST	créer	Créé une nouvelle ressource.
PUT	modifier	Met à jour une ressource existante.
DELETE	supprimer	Supprime la ressource du système.

2.2 Vue d'ensemble de l'article

Dans ce document, je parcourai une application qui propose pour chaque ville, la météo courante. Ce sera un Web service fidèle aux fondamentaux de REST. Pour la sérialisation et la désérialisation du java en XML, j'utiliserai JAXB dont j'ai déjà parlé dans Sérialisation et désérialisation XML avec Java (JAXB).

3. Web service côté serveur

Le Web service qui donne pour chaque ville la météo courante sera une Servlet (voir CityServlet). Chaque ville sera représentée par un bean annoté pour JAXB (voir City).
Créez un nouveau projet de type webapp avec Maven (commande à taper à la racine de votre workspace) :

```
$ mvn archetype:generate -DgroupId=net.dotmyself.weatherws -
    DartifactId=weatherws -Dversion=1.0 -Dpackage=net.dotmyself.
    weatherws -DarchetypeArtifactId=maven-archetype-webapp -
    DinteractiveMode=false
```

Ecrasez le pom.xml généré, dans le dossier weatherws, par ce pom.xml.

3.1 Le bean annoté pour JAXB

En format XML une ville ressemble à ceci :

```
<?xml version="1.0" encoding="UTF-8"?>
<city id="207">
<name>Grenoble</name>
<temperature>25</temperature>
<humidity>61</humidity>
<weather>Soleil</weather>
<uri>http://localhost:8080/weatherws/cities/207</uri>
<lastUpdate>24-08-07 05:43</lastUpdate>
</city>
```

La température est exprimée en Celsius et l'humidité est un pourcentage.
Créez la structure de données java associée (City.java) :

```
package net.dotmyself.weatherws;
import java.math.BigDecimal;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
/**
* A JAXB class useful for containing data from a city
*
* @author Hyacinthe MENIET
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
"name",
"temperature",
"humidity",
"weather",
"uri",
"lastUpdate"
})
@XmlRootElement(name = "city")
public class City {
@XmlElement(required = true)
protected String name;
@XmlElement(required = true)
protected BigDecimal temperature;
@XmlElement(required = true)
protected long humidity;
@XmlElement(required = true)
protected String weather;
@XmlElement
protected String uri;
@XmlElement
protected String lastUpdate;
@XmlAttribute
protected Long id;
/**
* Gets the value of the name property.
*
* @return
*     possible object is
*     {@link String }
*
*/
public String getName() {
return name;
```

```
}
/**
* Sets the value of the name property.
*
* @param value
*     allowed object is
*     {@link String }
*
*/
public void setName(String value) {
this.name = value;
}
/**
* Gets the value of the temperature property.
*
* @return
*     possible object is
*     {@link BigDecimal }
*
*/
public BigDecimal getTemperature() {
return temperature;
}
/**
* Sets the value of the temperature property.
*
* @param value
*     allowed object is
*     {@link BigDecimal }
*
*/
public void setTemperature(BigDecimal value) {
this.temperature = value;
}
/**
* Gets the value of the humidity property.
*
*/
public long getHumidity() {
return humidity;
}
/**
* Sets the value of the humidity property.
*
*/
public void setHumidity(long value) {
this.humidity = value;
}
/**
```

```
* Gets the value of the weather property.
*
* @return
*     possible object is
*     {@link String }
*
*/
public String getWeather() {
return weather;
}
/**
* Sets the value of the weather property.
*
* @param value
*     allowed object is
*     {@link String }
*
*/
public void setWeather(String value) {
this.weather = value;
}
/**
* Gets the value of the uri property.
*
* @return
*     possible object is
*     {@link String }
*
*/
public String getUri() {
return uri;
}
/**
* Sets the value of the uri property.
*
* @param value
*     allowed object is
*     {@link String }
*
*/
public void setUri(String value) {
this.uri = value;
}
/**
* Gets the value of the lastUpdate property.
*
* @return
*     possible object is
*     {@link String }
```

```
*
*/
public String getLastUpdate() {
return lastUpdate;
}
/**
* Sets the value of the lastUpdate property.
*
* @param value
*     allowed object is
*     {@link String }
*
*/
public void setLastUpdate(String value) {
this.lastUpdate = value;
}
/**
* Gets the value of the id property.
*
* @return
*     possible object is
*     {@link Long }
*
*/
public Long getId() {
return id;
}
/**
* Sets the value of the id property.
*
* @param value
*     allowed object is
*     {@link Long }
*
*/
public void setId(Long value) {
this.id = value;
}
}
```

Le bean est annoté via des tags JAXB. Ces annotations faciliteront la sérialisation et désérialisation de java vers XML.

3.2 La mémoire

Pour donner un peu de réalisme à cet exemple, j'utilise une mémoire volatile (e.g. à chaque redémarrage de Tomcat les données sont perdues) mais suffisante pour contextualiser les échanges. Créez la classe (CityMemory.java) :

```
package net.dotmyself.weatherws;
import java.util.HashMap;
import java.util.Map;
/**
* Data Access Object which allows to Create, Retrieve, Update and
    Delete
* {@link City}.
*
* @author Hyacinthe MENIET
*/
public class CityMemory {
private final Map<Long, City> mapCities = new HashMap();
private long nextId = 0;
/**
* Inserts the given {@link City}.
*
* @param item The {@link City} to insert.
* @return The key associated with the {@link City}.
*/
public Long create(City item) {
Long key = nextKey();
item.setId(key);
mapCities.put(key, item);
return key;
}
/**
* Gets a {@link City}.
*
* @param key The key associated with the {@link City}.
* @return The {@link City} or <code>null</code> if no {@link City
   } could be
* found.
*/
public City retrieve(Long key) {
return mapCities.get(key);
}
/**
* Updates an existing {@link City}.
*
* @param key The key associated with the {@link City}.
* @param item The {@link City}.
*/
public void update(Long key, City item) {
if (mapCities.containsKey(key)) {
item.setId(key);
mapCities.put(key, item);
} else {
throw new IllegalArgumentException("There is no City with key=" +
```

```
    key);
}
}
/**
* Removes the {@link City} for this key if present.
*
* @param key The key associated with the {@link City}.
*/
public void delete(Long key) {
mapCities.remove(key);
}
/**
* Searches for the given {@link City}, testing for equality using
    the
* equals method.
*
* @param item The {@link City}.
* @return The key or <code>null</code> if no key could be found.
*/
public Long getKey(City item) {
for (Map.Entry<Long, City> entry : mapCities.entrySet()) {
if (entry.getValue().equals(item)) {
return entry.getKey();
}
}
return null;
}
/**
* Returns the next key available.
*
* @return The next key available.
*/
protected Long nextKey() {
return ++nextId;
}
}
```

3.3 La Servlet

Terminons par la Servlet (CityServlet.java) :

```
package net.dotmyself.weatherws;
import java.io.IOException;
import java.io.OutputStream;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
```

```
import java.util.Locale;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
/**
* Servlet which uses HTTP verbs (GET,POST,PUT and DELETE) to
   perform a REST web
* service.
*
* @author Hyacinthe MENIET
*/
public class CityServlet extends HttpServlet {
private static final long serialVersionUID = 7159907909932997509L
   ;
private static final String CHARACTER_ENCODING = "UTF-8";
private static final String CONTENT_TYPE = "application/xml";
private final CityMemory memory;
public CityServlet() {
this.memory = new CityMemory();
}
@Override
protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws IOException {
try {
City city = getCityFromMemory(request);
city.setLastUpdate(getCurrentDate());
city.setUri(getCityUri(request, city.getId()));
// Marshalling retrieved City to the given OutputStream
JAXBContext context = JAXBContext.newInstance(City.class);
Marshaller marshaller = context.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
marshaller.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
response.setContentType(CONTENT_TYPE);
response.setCharacterEncoding(CHARACTER_ENCODING);
OutputStream os = response.getOutputStream();
marshaller.marshal(city, os);
} catch (IllegalStateException e) {
response.sendError(HttpServletResponse.SC_NOT_FOUND, e.getMessage
   ());
} catch (IOException | JAXBException e) {
response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
   e.getMessage());
}
}
@Override
```

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws IOException {
try {
City city = getCityFromXml(request);
Long key = memory.create(city);
response.setHeader("Location", getCityUri(request, key));
response.setStatus(HttpServletResponse.SC_CREATED);
} catch (IOException | JAXBException e) {
response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
    e.getMessage());
}
}
@Override
protected void doPut(HttpServletRequest request,
    HttpServletResponse response) throws IOException {
try {
City city = getCityFromMemory(request);
city.setLastUpdate(getCurrentDate());
memory.update(city.getId(), getCityFromXml(request));
response.setStatus(HttpServletResponse.SC_NO_CONTENT);
} catch (IOException | JAXBException e) {
response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
    e.getMessage());
}
}
@Override
protected void doDelete(HttpServletRequest request,
    HttpServletResponse response) throws IOException {
memory.delete(getKey(request));
response.setStatus(HttpServletResponse.SC_NO_CONTENT);
}
private String getCityUri(HttpServletRequest request, Long key) {
return request.getScheme() + "://" + request.getServerName() +
    ":"
+ request.getServerPort() + request.getContextPath()
+ request.getServletPath()
+ "/" + key;
}
private String getCurrentDate() {
LocalDateTime now = LocalDateTime.now();
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-
    yy HH:mm");
return now.format(formatter);
}
private City getCityFromXml(HttpServletRequest request) throws
    JAXBException, IOException {
JAXBContext context = JAXBContext.newInstance(City.class);
Unmarshaller unmarshaller = context.createUnmarshaller();
City city = (City) unmarshaller.unmarshal(request.getReader());
```

```
return city;
}
private City getCityFromMemory(HttpServletRequest request) {
City city = memory.retrieve(getKey(request));
if (city == null) {
throw new NullPointerException("Can't find the requested city");
}
return city;
}
private Long getKey(HttpServletRequest request) {
String[] parts = request.getPathInfo().split("/");
if (parts.length <= 1) {
throw new IllegalStateException("Missing id");
}
return Long.valueOf(parts[1]);
}
}
```

Pour déployer votre application sous Tomcat vous aurez besoin d'un descripteur de déploiement (web.xml) :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
version="3.1"
metadata-complete="true">
<display-name>Weather of the world</display-name>
<description>
This application is a Web service which allows to retrieve
the weather of several cities in the world
</description>
<init-param>
<param-name>readonly</param-name>
<param-value>false</param-value>
</init-param>
<servlet>
<servlet-name>CityServlet</servlet-name>
<servlet-class>net.dotmyself.weatherws.CityServlet</servlet-class
    >
</servlet>
<servlet-mapping>
<servlet-name>CityServlet</servlet-name>
<url-pattern>/cities/*</url-pattern>
</servlet-mapping>
</web-app>
```

Créez un fichier index.jsp qui servira de page d'accueil Web pour votre Web service :
Générez le war de votre projet maven, l'arborescence devrait être la suivante :

```
index.jsp
WEB-INF/web.xml
WEB-INF/classes/net/dotmyself/weatherws/CityMemory.class
WEB-INF/classes/net/dotmyself/weatherws/City.class
WEB-INF/classes/net/dotmyself/weatherws/CityServlet.class
```

Quand vous avez terminé, disposez simplement votre war dans **$CATALINA_HOME/webapps/** et redémarrez Tomcat. Vous pouvez tester l'application Web en vous connectant à l'adresse http://localhost:8080/we

4. Le Web service côté client

Une fois n'est pas coutume, pour interagir avec le Web service ci-dessus, je ne créerai pas de client Java. Premièrement parce que j'aborde cet aspect dans Clients Web services REST avec Java (JAXB). Ensuite, c'est l'occasion de présenter l'outil en ligne de commande : curl fort utile pour déboguer un Web service ou toute application réseau.

4.1 Insérer des villes

Ajoutez la ville de Grenoble :

```
$ curl -i -X POST -H 'Content-Type: application/xml' -d '<city><
    name>Grenoble</name><temperature>25</temperature><humidity
    >61</humidity><weather>Soleil</weather></city>' http://
    localhost:8080/weatherws/cities/
HTTP/1.1 201 Cree
Server: Apache-Coyote/1.1
Location: http://localhost:8080/weatherws/cities/1
Content-Length: 0
Date: Fri, 22 Dec 2017 15:06:25 GMT
```

La ville de Grenoble a bien été ajoutée comme le signale le `HTTP/1.1 201 Crée`. La réponse vous retourne également l'URI à partir de laquelle vous consulterez la météo relative à Grenoble :

```
$ curl http://localhost:8080/weatherws/cities/1
<?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;
    standalone=&quot;yes&quot;?>
<city id=&quot;1&quot;>
    <name>Grenoble</name>
    <temperature>25</temperature>
    <humidity>61</humidity>
    <weather>Soleil</weather>
    <uri>http://localhost:8080/weatherws/cities/1</uri>
    <lastUpdate>22-12-17 16:18</lastUpdate>
</city>
```

Ajoutez d'autres villes :

```
$ curl -i -X POST -H 'Content-Type: application/xml' -d '<city><
    name>Paris </name><temperature >21</temperature ><humidity >78</
    humidity ><weather >Nuages lourds </weather ></city >' http://
    localhost:8080/weatherws/cities/
$ curl -i -X POST -H 'Content-Type: application/xml' -d '<city><
    name>Madrid </name><temperature >25</temperature ><humidity >36</
    humidity ><weather >Peu nuageux </weather ></city >' http://
    localhost:8080/weatherws/cities/
$ curl -i -X POST -H 'Content-Type: application/xml' -d '<city><
    name>Tokyo </name><temperature >27</temperature ><humidity >79</
    humidity ><weather >Averses mineures </weather ></city >' http://
    localhost:8080/weatherws/cities/
$ curl -i -X POST -H 'Content-Type: application/xml' -d '<city><
    name>Washington </name><temperature >28</temperature ><humidity
    >74</humidity ><weather >Peu nuageux </weather ></city >' http://
    localhost:8080/weatherws/cities/
```

4.2 Consulter une ville existante

Si vous avez lu le paragraphe ci-dessus, vous savez déjà comment consulter la météo relative à une ville. Par exemple pour consulter la météo associée à Madrid (ville n° 3), tapez :

```
$ curl http://localhost:8080/weatherws/cities/3
<?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;
    standalone=&quot;yes&quot;?>
<city id=&quot;3&quot;>
    <name>Madrid </name>
    <temperature >25</temperature >
    <humidity >36</humidity >
    <weather >Peu nuageux </weather >
    <uri>http://localhost:8080/weatherws/cities/3</uri>
    <lastUpdate >22-12-17 16:20</lastUpdate >
</city >
```

4.3 Modifier une ville existante

Modifiez la ville de Tokyo (ville n° 4) :

```
$ curl -i -X PUT -H 'Content-Type: application/xml' -d '<city><
    name>Tokyo </name><temperature >28</temperature ><humidity >77</
    humidity ><weather >Soleil </weather ></city >' http://localhost
    :8080/weatherws/cities/4
HTTP/1.1 204 Pas de Contenu
Server: Apache-Coyote/1.1
Date: Fri, 22 Dec 2017 15:21:49 GMT
```

Vérifiez que les changements sont pris en compte :

```
$ curl http://localhost:8080/weatherws/cities/4
<?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;
    standalone=&quot;yes&quot;?>
<city id=&quot;4&quot;>
     <name>Tokyo</name>
     <temperature>28</temperature>
     <humidity>77</humidity>
     <weather>Soleil</weather>
     <uri>http://localhost:8080/weatherws/cities/4</uri>
     <lastUpdate>22-12-17 16:22</lastUpdate>
</city>
```

4.4 Supprimer une ville existante

Supprimez la ville de Paris (ville n° 2) :

```
$ curl -i -X DELETE http://localhost:8080/weatherws/cities/2
HTTP/1.1 204 Pas de Contenu
Server: Apache-Coyote/1.1
Date: Fri, 22 Dec 2017 15:22:53 GMT
```

Vérifiez que la ville a correctement été retirée de l'application. En particulier vous obtiendrez une page HTML d'alerte vous indiquant que l'élément n'a pas été trouvé (i.e. *Can't find the requested city*).

```
$ curl http://localhost:8080/weatherws/cities/2
```