Middleware - Sérialisation Java/XML - MAVEN

26 février 2018

1

Configuration de maven pour JAXB

La sérialisation/dé-serialisation ne se fait pas automatiquement et est un problème majeur à résoudre. Cela nécessite en particulier l'utilisation de *parsers* et de *composers* pour analyser et/ou générer des documents `XML`. Le recours à de tels outils se sont révélés particulièrement complexes et ont encouragé la mise en oeuvre de solutions qui en facilitent l'usage. En particulier, la conversion d'objets `Java` en documents `XML` se fait par l'intermédiaire d'un schéma `XML` qui modélise les données manipulées afin de valider la conversion des objets `Java` en `XML`. Comme indiqué dans la Figure 1, la démarche usuellement empruntée est (i) d'analyser le format des données `XML` que l'on souhaite obtenir (étape ❶), (ii) de déduire un modèle générale des données (étape ❷), (iii) générer des classes Java correspondantes manipulables dans le code de votre application (étape ❸).

1. Créer un nouveau projet `Maven` de type `maven-archetype-quickstart` ayant comme **groupID** `net.serialisation.data` et comme **artifactID** `jaxb`

Base revision (None), (None), (None).

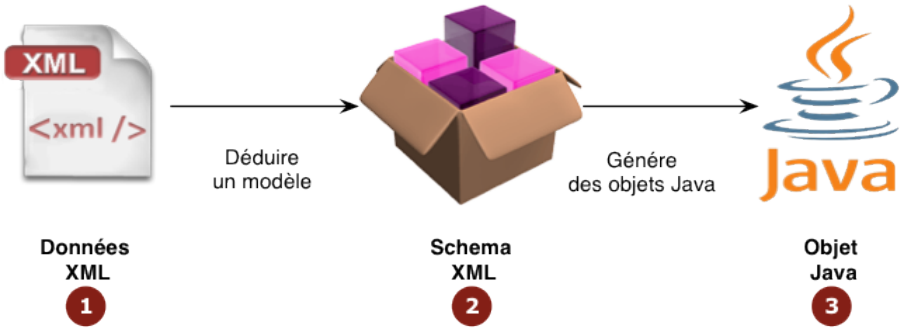

FIGURE 1 – Accès à la page du web service

2. Pour utiliser la librairie `JAXB`, ajouter les dépendances suivante à votre projet.

Listing 1– pom.xml

```
Line 1  <dependencies>
    -    ...
    -     <dependency>
    -       <groupId>javax.xml.bind</groupId>
    5       <artifactId>jaxb-api</artifactId>
    -       <version>2.2</version>
    -     </dependency>
    -
    -     <dependency>
   10       <groupId>com.sun.xml.bind</groupId>
    -       <artifactId>jaxb-impl</artifactId>
    -       <version>2.2.7</version>
    -     </dependency>
    -    ...
   15  </dependencies>
```

3. Les fichiers `Java` sont générés à partir du schéma `XML`. La génération est effectuée grâce à l'outil `xjc` fourni par la librairie `JAXB`. Afin d'automatiser cette étape, vous aller configurer `Maven`, pour lui indiquer où, quand et comment générer les fichiers Java afin que votre application soit capable de sérialiser/dé-sérialiser les objets Java en XML et *vice versa*.

Listing 2– pom.xml

```
Line 1  <build>
        <plugins>
        ...
        <plugin>
5         <groupId>org.jvnet.jaxb2.maven2</groupId>
          <artifactId>maven-jaxb2-plugin</artifactId>
          <version>0.8.3</version>
          <executions>
            <execution>
10            <id>xjc</id>
              <goals>
                <goal>generate</goal>
              </goals>
              <configuration>
15              <extension>true</extension>
                <schemaDirectory>src/main/resources/schema</
                    schemaDirectory>
                <schemaIncludes>
                  <include>**/*.xsd</include>
                </schemaIncludes>
20              <generatePackage>net.serialisation.data</
                    generatePackage>
                <generateDirectory>src/main/java</generateDirectory>
                <removeOldOutput>false</removeOldOutput>
                <forceRegenerate>true</forceRegenerate>
                <verbose> true </verbose>
25            </configuration>
            </execution>
          </executions>
        </plugin>
        ...
30    </plugins>
    </build>
```

- `<extension>`, indique que l'on autorise une utilisation étendue de JAXB.
- `<schemaDirectory>`, permet de spécifier dans quel répertoire se trouve votre schéma XML.

- <generatePackage>, indique le nom du package auquelles appartiendront les classes Java générées.
- <generateDirectory>, indique dans quel répertoire seront générées les fichiers sources Java.
- <version>, permet de spécifier explicitement la version du plugin

4. JAXB générant des fichiers sources Java qui contiennent des annotations, vous devez utiliser au minimum la version 1.5 du compilateur Java. Modifier en conséquence le fichier pom.xml de votre projet.

Listing 3– pom.xml

```
Line 1  <build>
    -    <plugins>
    -     ...
    -     <plugin>
    5      <groupId>org.apache.maven.plugins</groupId>
    -      <artifactId>maven-compiler-plugin</artifactId>
    -      <version>2.3.2</version>
    -      <configuration>
    -       <source>1.5</source>
   10       <target>1.5</target>
    -      </configuration>
    -     </plugin>
    -     ...
    -    </plugins>
   15  </build>
```

5. On souhaite dorénavant tester l'utilisation de l'outil `xjc` avec un exemple simple. A cette fin, écrire le schéma `XML` correspondant aux données `XML` sous-jacente. Le schéma sera écrit dans un fichier `books.xsd` placé dans le répertoire `src/main/resources/schema/`. On pourra s'aider de la documentation en ligne de `JAXB` à l'url suivante : http://jaxb.java.net/tutorial/

Listing 4– books.xml

```
Line 1  <CatalogData>
    -    <book id="123456789">
    -     <author>David Bromberg</author>
    -     <title>Programmation Android</title>
    5     <genre>Programmation</genre>
    -     <price>0.0</price>
    -     <publish_date>2012-01-30</publish_date>
    -     <description>Apprendre la programmation sur Android</
              description>
    -    </book>
   10    <book id="987654321">
    -     <author>David Bromberg</author>
    -     <title>Programmation Iphone</title>
    -     <genre>Programmation</genre>
    -     <price>0.0</price>
   15     <publish_date>2012-01-30</publish_date>
    -     <description>Apprendre la programmation sur Iphone</
              description>
    -    </book>
    -   </CatalogData>
```

Listing 5– books.xsd

```
Line 1  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    -    //TO DO
    -   </xsd:schema>
```

6. Une fois le schéma écrit et placé dans le répertoire `src/main/resources/schema/`, faire un `mvn clean install` pour générer les classes java correspondantes. Vérifier que les fichiers sources sont bien générés dans le répertoire `src/main/java`. En particulier, 3 fichiers devraient être générés, à savoir `Bookdata.java`, `CatalogData.java` et `ObjectFactory.java`.

> Naturellement les trois classes `Bookdata.java`, `CatalogData.java` et `ObjectFactory.java` se trouvent dans le répertoire correspondant à leur package. Par conséquent ces classes doivent se retrouver dans le répertoire `src/main/java/net/serialisation/data`.

7. Créer une classe `App.java` intégrant les objets `Java` fraichement généré et permettant de manipuler les données `XML`. Créer programmatiquement le fichier `XML books.xml` donné précédemment en manipulant les objets Java qui ont été générés.

   - Faire un `import net.serialisation.data.*;`, pour manipuler les classes générées.
   - Instancier la classe `ObjectFactory` pour récupérer les instances des objets `Java` correspondant aux balises `xml`.
   - Faire un `import javax.xml.bind.*;`, pour utiliser `JAXB`.

> Naturellement `App.java` doit se retrouver dans le répertoire `src/main/java/net/serialisation/` et avoir comme nom de package `net.serialisation`.

8. Une fois la hiérarchie de classe Java instanciée, sérialisez les données grâce à l'API `JAXB`. Afin de faire un *pretty print* du document `XML` résultant, utliser la méthode `setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, new Boolean(true));` sur l'instance de votre *marshaller*

9. Modifier votre fichier `pom.xml` afin d'executer votre application `App` lorsque vous saisissez `mvn exec:exec`