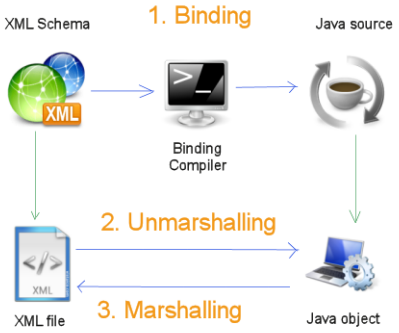
Sérialisation et désérialisation XML avec Java (JAXB)

Hyacinthe MENIET

3 août 2019

JAXB qui est l'acronyme de l'anglais *« Java API for XML Binding »* est une boîte à outils pour sérialisation et désérialisation d'objets java. Elle permet de voyager de manière bidirectionnelle entre les univers Java et XML grâce à un mapping (XML <-> Java). Ce mapping est flexible dans la mesure il peut être généré directement par JAXB à partir d'un schéma XML ou simplement définie par le développeur lui-même via des annotations.
Dans ce document je vais indiquer comment générer des classes java à partir d'un schéma XML puis comment aller d'un fichier XML à un objet Java et vice versa.

- Vous êtes familier de Java 8 et de sa syntaxe.
- Vous êtes familier de XML et notamment des schémas XML.
- Vous disposez du JDK 8 minimum.



JAXB permet de réaliser 3 types d'échanges entre les univers XML et Java :

- L'**association** (en anglais le *binding*) c'est l'action qui consiste à générer des classes java à partir d'un schéma XML (extension .xsd).
- La **désérialisation** (en anglais l'*unmarshalling*) c'est l'action qui consiste à instancier des classes java à partir de données contenues dans des fichiers XML.
- La **sérialisation** (en anglais le *marshalling*) c'est l'action qui consiste à produire un fichier XML à partir d'objets java.

Pour rendre ce document digeste, je vais parcourir les API de JAXB à travers l'exemple d'un magasin de location de DVD et VHS. A chaque article loué j'associe :

- Le type d'article : DVD ou VHS
- L'identifiant du loueur.
- Le titre du film correspondant.
- La date d'emprunt.

Consultez ce fichier rentals.xml listant les locations en cours :
Voici le schéma XML rentals.xsd utilisé par le magasin de location :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema targetNamespace="http://file.dotmyself.net/source/4/
    j2xml" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" xmlns="http://file.dotmyself.
    net/source/4/j2xml">
<xs:element name="begindate" type="xs:string"/>
<xs:element name="rental">
<xs:complexType>
<xs:sequence>
<xs:element ref="user"/>
<xs:element ref="title"/>
<xs:element ref="begindate"/>
</xs:sequence>
<xs:attribute name="type" use="required">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="DVD"/>
<xs:enumeration value="VHS"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="rentals">
<xs:complexType>
<xs:sequence>
<xs:element ref="rental" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="title" type="xs:string"/>
<xs:element name="user" type="xs:int"/>
</xs:schema>
```

Pour générer les classes java, utilisez le compilateur `xjc` comme suite :

```
> xjc -d src rentals.xsd
parsing a schema...
compiling a schema...
net/dotmyself/file/source/_4/j2xml/ObjectFactory.java
net/dotmyself/file/source/_4/j2xml/Rental.java
net/dotmyself/file/source/_4/j2xml/Rentals.java
net/dotmyself/file/source/_4/j2xml/package-info.java
```

Quelques explications :

— L'option -d permet d'indiquer le dossier dans lequel seront générées les classes. Ici c'est le dossier src de mon projet Java. Lisez cette page pour obtenir la liste exhaustive des options
— Le nom du package Java (e.g. *net.dotmyself.file.source._4.j2xml*) correspond au namespace de mon schéma XML (e.g. https://file.dotmyself.net/source/4/j2xml)
— Rental.java : est le java bean qui représente une location.
— Rentals.java : est le java bean qui représente un groupe de locations.
— ObjectFactory.java : est la fabrique d'objets java que nous utiliseront pour instancier les classes java.
— package-info.java : est la classe java qui fait le mapping entre le namespace XML (e.g. https://file.dotmyself.net/source/4/j2xml) et la package java (e.g. *net.dotmyself.file.source._4.j2xml*)

Vous pouvez modifier les classes produites pour les adapter plus finement à vos besoins. Néanmoins, si vous avez le choix, je vous conseille :

— De privilégier les modifications du schéma XML afin de le rendre plus/moins contraignant. A chaque mise à jour du schéma vous devrez régénérer les classes java.
— De créer vos propres fichiers de binding (extension .xjb) et de les passer en paramètre à `xjc` via l'option `-b`.

Ci-dessous la classe XML2Java.java pourvue d'une méthode main. Cette classe instancie un Unmarshaller depuis le JAXBContext puis l'utilise pour lire le fichier rentals.xml et produire des objets java :

```
package net.dotmyself.j2xml;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import net.dotmyself.file.source._4.j2xml.Rental;
import net.dotmyself.file.source._4.j2xml.Rentals;
/**
* Unmarshalles the given XML Document.
* @author Hyacinthe MENIET
*/
public class XML2Java {
public static void main(String[] args) {
try {
if (args == null || args.length < 1) {
throw new IllegalArgumentException("You must indicate a path to
    file to read");
}
String fileName = args[0];
// Unmarshalles the given XML file to objects
JAXBContext context;
context = JAXBContext.newInstance("net.dotmyself.file.source._4.
    j2xml");
Unmarshaller unmarshaller = context.createUnmarshaller();
```

```
Rentals rentals = (Rentals) unmarshaller.unmarshal(new FileReader
    (fileName));
List<Rental> listOfRentals = rentals.getRental();
// Displays objects
for (Rental rental : listOfRentals) {
System.out.println("Type=" + rental.getType() + ", "
+ "User=" + rental.getUser() + ", "
+ "Title=" + rental.getTitle() + ", "
+ "Begin date=" + rental.getBegindate());
}
} catch (JAXBException | FileNotFoundException ex) {
Logger.getLogger(XML2Java.class.getName()).log(Level.SEVERE, null
    , ex);
}
}
}
```

Après compilation et packaging (**j2xml-1.0.jar étant le jar obtenu suite à la compilation puis au packaging des .class du projet java**), le résultat de l'exécution :

```
> java -cp j2xml-1.0.jar net.dotmyself.j2xml.XML2Java rentals.xml
Type=DVD, User=1, Title=Casino Royal, Begin date=15/07/2007
Type=VHS, User=3, Title=Borat, Begin date=16/07/2007
```

Ci-dessous je créé la classe Java2XML.java pourvue d'une méthode main. Cette classe instancie des objets de type Rental et Rentals à partir de l'ObjectFactory et dans lesquels elle stocke des données statiques. Puis grâce au Marshaller elle produit un fichier XML :

```
package net.dotmyself.j2xml;
import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import net.dotmyself.file.source._4.j2xml.ObjectFactory;
import net.dotmyself.file.source._4.j2xml.Rental;
import net.dotmyself.file.source._4.j2xml.Rentals;
/**
* Marshalles the created java Objects to the specified XML
    Document.
* @author Hyacinthe MENIET
*/
public class Java2XML {
public static void main(String[] args) {
try {
if (args == null || args.length < 1) {
throw new IllegalArgumentException("You must indicate a path to
```

```
    file to generate");
}
String fileName = args[0];
// Instanciates an ObjectFactory
ObjectFactory objFactory = new ObjectFactory();
// Creates first rental
Rental rental1 = objFactory.createRental();
rental1.setBegindate("15/07/2007");
rental1.setTitle("Casino Royal");
rental1.setType("DVD");
rental1.setUser(1);
// Creates second rental
Rental rental2 = objFactory.createRental();
rental2.setBegindate("16/07/2007");
rental2.setTitle("Borat");
rental2.setType("VHS");
rental2.setUser(3);
// Stores rentals
Rentals rentals = objFactory.createRentals();
rentals.getRental().add(rental1);
rentals.getRental().add(rental2);
// Marshalles objects to the specified file
JAXBContext context = JAXBContext.newInstance(Rentals.class);
Marshaller marshaller = context.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
marshaller.marshal(rentals, new FileWriter(fileName));
} catch (JAXBException | IOException ex) {
Logger.getLogger(Java2XML.class.getName()).log(Level.SEVERE, null
    , ex);
}
}
}
```

Après compilation, l'exécution :

```
> java -cp j2xml-1.0.jar net.dotmyself.j2xml.Java2XML generated-
    rentals.xml
```

Ce qui génère le fichier generated-rentals.xml.