Binding XML - Java

par Denis Thomas

Date de publication : 6 juillet 2012

Dernière mise à jour : 11 octobre 2012

DÉBUTANT

À travers un exemple concret, cet article présente l'API JAXB qui nous permet de transformer un fichier XML en objets Java, et vice versa.

par Denis Thomas

I - Objectif

Il sera d'obtenir une requête SQL à partir de son nom et du groupe auquel elle appartient. Voyons pour commencer à quoi pourrait ressembler l'interface de notre QueryLoader :

interface IQueryLoader
```
1. public interface IQueryLoader {
2.
3.     Map<String, String> getQueries(String queriesName);
4.
5.     String getQuery(String groupName, String queryName);
6.
7. }
```

Créons ensuite notre fichier XML qui nous servira d'exemple :

Fichier exemple
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <queries>
3.     <group name="group1">
4.         <query name="query1">
5.             SELECT field1 FROM table1
6.         </query>
7.         <query name="query2">
8.             SELECT field2 FROM table2
9.         </query>
10.     </group>
11.     <group name="group2">
12.         <query name="query3">
13.             SELECT field3 FROM table3
14.         </query>
15.         <query name="query4">
16.             SELECT field4 FROM table4
17.         </query>
18.     </group>
19. </queries>
```

À partir de ce fichier, nous pouvons écrire assez facilement nos classes Java correspondantes. Tout d'abord notre requête :

class Query
```
1. public class Query {
2.
3.     private String name;
4.     private String query;
5.
6.     public final String getName() {
7.         return name;
8.     }
9.
10.     public void setName(String name) {
11.         this.name = name;
12.     }
13.
14.     public final String getQuery() {
15.         return query;
16.     }
17.
18.     public final void setQuery(String query) {
19.         this.query = query;
20.     }
21. }
```

Puis notre groupe de requêtes :

class QueryGroup
```
1. public class QueryGroup {
2.
3.     private String name;
4.
5.     private List<Query> queries = new LinkedList<Query>();
6.
7.     public String getName() {
8.         return name;
9.     }
10.
11.     public final void setName(String name) {
12.         this.name = name;
13.     }
14.
15.     public final List<Query> getQueries() {
16.         return queries;
17.     }
18.
19.     public final void setQueries(List<Query> queries) {
20.         this.queries = queries;
21.     }
22.
23. }
```

Il nous faut aussi une classe pour contenir la liste des groupes de requêtes :

class Queries
```
1. public class Queries {
2.
3.     private List<QueryGroup> queryGroups = new LinkedList<QueryGroup>();
4.
5.     public final void setQueryGroups(List<QueryGroup> queryGroups) {
6.         this.queryGroups = queryGroups;
7.     }
8.
9.     public final List<QueryGroup> getQueryGroups() {
10.         return queryGroups;
11.     }
12. }
```

Il nous reste à voir l'implémentation de notre interface IQueryLoader, nous le ferons très bientôt. Occupons-nous maintenant de JAXB.

II - Présentation de JAXB

JAXB est l'acronyme de **Java Architecture for XML Binding**. JAXB est donc capable de sérialiser des objets Java en un fichier XML, et de les désérialiser. Il existe également d'autres API pour manipuler le XML en Java, comme SAX (**Simple API for XML**) ou JAXP (**Java API for XML Processing**). SAX fait d'ailleurs partie de JAXP. Mais ces API n'ont pas la même finalité que JAXB, elles servent davantage à manipuler et parcourir l'arborescence XML qu'à réaliser la sérialisation d'objets en XML.

Si on prend l'exemple de SAX, on va parcourir l'arborescence XML avec un parser, instanciÃ© depuis une factory. Ce parser lit ses donnÃ©es Ã partir d'un InputStream, et appelle diverses mÃ©thodes d'un handler, que nous crÃ©ons en hÃ©ritant d'un **DefaultHandler**. Le dÃ©but de l'analyse du document se fait ainsi par l'appel de la mÃ©thode **startDocument()**, l'arrivÃ©e sur une balise avec la mÃ©thode **startElement()**, et ainsi de suite pour chaque Ã©lÃ©ment trouvÃ©. Nous devons donc dÃ©finir le comportement attendu en fonction des cas : est-ce une balise, est-ce celle attendue, ses attributs sont-ils corrects, etc. Tout ceci est plutÃ´t lourd Ã mettre en place, d'autant plus que notre handler ne conserve aucune donnÃ©e en mÃ©moire quand il parcourt le flux XML. C'est Ã notre implÃ©mentation de tout enregistrer.

Si on connait exactement le format des données attendues, ou mieux, si on dispose du schéma, JAXB est beaucoup plus simple à utiliser, puisque quelques classes Java annotées lui suffisent pour analyser le flux XML. Une API qui se rapprocherait de JAXB est **Castor**, qui fait la même chose : transformer un flux XML en POJO, et réciproquement. Mais plus ancienne, elle passe par l'analyse d'un fichier XML de mapping, parfois complexe à écrire.

Il existe d'autres solutions permettant la sÃ©rialisation, par exemple les classes **XMLEncoder** et **XMLDecoder**, **Jakarta Commons Digester** de la fondation Apache, ou encore l'API XStream. Ne connaissant pas ces solutions, je n'en parlerai pas, et vous laisse les dÃ©couvrir Ã travers les tutoriels sur Developpez.com, dont vous trouverez les liens en fin d'article.

III - Utilisation de JAXB

Pour disposer de JAXB dans notre classpath, avec Maven il suffit d'ajouter les dépendances suivantes dans notre pom :

Dependency maven pour JAXB

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.2.4</version>
</dependency>
```

Sinon, on peut le télécharger **ici** par exemple.

III-A - Annotations de nos classes

Nous avons dit que tout se faisait par annotations. Nous allons commencer par la classe correspondant à la racine de notre arborescence XML. Il s'agit ici de Queries, qui sera annotée avec javax.xml.bind.annotation.XmlRootElement :

```
@XmlRootElement
public class Queries {
    ?
}
```

Cet élément correspond à la balise racine <queries/>, qui est unique par définition. C'est toujours la première balise rencontrée, il est inutile d'indiquer son nom. Il n'en va pas de même de la balise suivante, <group/>, dont nous devons indiquer le nom. L'annotation est javax.xml.bind.annotation.XmlElement, et se place sur le getter :

```
@XmlElement(name = "group")
public final List<QueryGroup> getQueryGroups() {
    return queryGroups;
}
```

Passons maintenant au contenu de cette balise <queries/>, représenté par la classe QueryGroup. Cette balise contient un attribut, son nom. L'annotation est javax.xml.bind.annotation.XmlAttribute :

```
@XmlAttribute
public String getName() {
    return name;
}
```

On retrouve l'annotation @XmlElement pour la balise <Query/> :

```
@XmlElement(name = "query")
public final List<Query> getQueries() {
    return queries;
}
```

Finissons par la classe Query. Elle possède un attribut XML, son nom :

```
@XmlAttribute(name = "name")
public final String getName() {
    return name;
}
```

Et aussi une valeur, la requête SQL, annotée avec javax.xml.bind.annotation.XmlValue :

```
@XmlValue
public final String getQuery() {
    return query;
}
```

Voilà pour ce qui est des annotations. Il n'y a rien de plus à faire, JAXB est maintenant capable très simplement de faire le lien entre nos POJO et notre fichier XML. La transformation d'objets Java en flux XML se nomme le marshalling, l'unmarshalling étant la transformation de XML en POJO. Et ceci se fait simplement avec une seule ligne de code :

```
Queries queries = JAXB.unmarshal(xmlInputStream, Queries.class);
```

Et c'est tout. Il ne nous reste plus qu'à tester que tout fonctionne.

III-B - Tests

On écrit une petite classe de test unitaire pour JUnit. Pour tester l'unmarshalling, on part du fichier, et on vérifie que toutes les requêtes sont bien présentes. Pour le test du marshalling, on commence par notre objet « unmarshallé », et qu'on « marshallera » dans une String. Et on « unmarshalle » à nouveau cette String dans un objet Queries, qu'il nous reste à comparer au premier. Voici le code du test :

Classe de test

```
import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;
// Autres import
public class QueriesTest {

    private Queries queries;

    @Before
    public void before() {
        InputStream xmlStream = Queries.class.getResourceAsStream("queriesTest.xml");
        queries = JAXB.unmarshal(xmlStream, Queries.class);
     }

     @Test
     public void unmarshallingTest() throws Exception {
        List<QueryGroup> queryGroups = queries.getQueryGroups();
        assertThat(queries.getQueryGroups().size(), is(2));

        QueryGroup group = queryGroups.get(0);
        assertThat(group.getName(), is("group1"));
        assertThat(group.getQueries().size(), is(2));

        List<Query> queryList = group.getQueries();
        Query query = queryList.get(0);
        assertThat(query.getName(), is("query1"));
        assertThat(query.getQuery().trim(), is("SELECT field1 FROM table1"));

        query = queryList.get(1);
        assertThat(query.getName(), is("query2"));
        assertThat(query.getQuery().trim(), is("SELECT field2 FROM table2"));

        group = queryGroups.get(1);
        assertThat(group.getName(), is("group2"));
        assertThat(group.getQueries().size(), is(2));
```

Classe de test

```
35.
36.         queryList = group.getQueries();
37.         query = queryList.get(0);
38.         assertThat(query.getName(), is("query3"));
39.         assertThat(query.getQuery().trim(), is("SELECT field3 FROM table3"));
40.
41.         query = queryList.get(1);
42.         assertThat(query.getName(), is("query4"));
43.         assertThat(query.getQuery().trim(), is("SELECT field4 FROM table4"));
44.
45.     }
46.
47.     @Test
48.     public void marshallingTest() throws Exception {
49.         StringWriter writer = new StringWriter();
50.         JAXB.marshal(queries, writer);
51.
52.         String xmlString = writer.toString();
53.         System.out.println(xmlString);
54.         Queries queries2 = JAXB.unmarshal(new StringReader(xmlString), Queries.class);
55.
56.         checkQueries(queries2, queries);
57.     }
58.
59.     private void checkQueries(Queries queries, Queries expected) {
60.         List<QueryGroup> groups = queries.getQueryGroups();
61.         List<QueryGroup> expectedGroups = expected.getQueryGroups();
62.         checkGroups(groups, expectedGroups);
63.     }
64.
65.     private void checkGroups(List<QueryGroup> groups, List<QueryGroup> expectedGroups) {
66.         assertThat(groups.size(), is(expectedGroups.size()));
67.         Iterator<QueryGroup> iterator1 = groups.iterator();
68.         Iterator<QueryGroup> iterator2 = expectedGroups.iterator();
69.         while (iterator1.hasNext()) {
70.             checkGroup(iterator1.next(), iterator2.next());
71.         }
72.     }
73.
74.     private void checkGroup(QueryGroup group, QueryGroup expectedGroup) {
75.         assertThat(group.getName(), is(expectedGroup.getName()));
76.         List<Query> queriesList = group.getQueries();
77.         List<Query> expectedQueriesList = expectedGroup.getQueries();
78.         checkQueriesList(queriesList, expectedQueriesList);
79.     }
80.
81.     private void checkQueriesList(List<Query> queriesList, List<Query> expectedQueriesList) {
82.         assertThat(queriesList.size(), is(expectedQueriesList.size()));
83.         Iterator<Query> iterator1 = queriesList.iterator();
84.         Iterator<Query> iterator2 = expectedQueriesList.iterator();
85.         while (iterator1.hasNext()) {
86.             checkQuery(iterator1.next(), iterator2.next());
87.         }
88.     }
89.
90.     private void checkQuery(Query query, Query expectedQuery) {
91.         assertThat(query.getName(), is(expectedQuery.getName()));
92.         assertThat(query.getQuery().trim(), is(expectedQuery.getQuery().trim()));
93.     }
94.
95. }
```

Nous avons imprimé notre String XML, ce qui nous permet de vérifier visuellement à quoi elle ressemble. Et maintenant que tout est correct, il nous reste à implémenter notre QueryLoader.

III-C - Implémentation du QueryLoader

Prenons de bonnes habitudes, et commençons par notre classe de test :

class XmlQueryLoaderTest

```
1. import static org.hamcrest.CoreMatchers.is;
2. import static org.junit.Assert.assertThat;
3. // Autres imports
4. public class XmlQueryLoaderTest {
5.
6.     private XmlQueryLoader queryLoader;
7.
8.     @Before
9.     public void before() {
10.         InputStream xmlStream = XmlQueryLoader.class.getResourceAsStream("queriesTest.xml");
11.         queryLoader = new XmlQueryLoader(xmlStream);
12.     }
13.
14.     @Test
15.     public void getQueryTest() throws Exception {
16.         assertThat(queryLoader.getQuery("group1", "query1"), is("SELECT field1 FROM table1"));
17.     }
18.
19.     @Test
20.     public void getQueriesTest() throws Exception {
21.         Map<String, String> queries = queryLoader.getQueries("group2");
22.         assertThat(queries.size(), is(2));
23.         assertThat(queries.get("query3"), is("SELECT field3 FROM table3"));
24.         assertThat(queries.get("query4"), is("SELECT field4 FROM table4"));
25.     }
26.
27. }
```

Et enfin, l'implémentation à tester :

class XmlQueryLoader

```
1. public class XmlQueryLoader implements IQueryLoader {
2.
3.     private final Map<String, Map<String, String>> queriesGroup = new LinkedHashMap<String,
 Map<String, String>>();
4.
5.     public XmlQueryLoader(InputStream xmlStream) {
6.         Queries queries = JAXB.unmarshal(xmlStream, Queries.class);
7.         for (QueryGroup group : queries.getQueryGroups()) {
8.             String groupName = group.getName();
9.             Map<String, String> queryGroup = new LinkedHashMap<String, String>();
10.             queriesGroup.put(groupName, queryGroup);
11.             for (Query query : group.getQueries()) {
12.                 String queryName = query.getName();
13.                 String sqlQuery = query.getQuery();
14.                 queryGroup.put(queryName, sqlQuery.trim());
15.             }
16.         }
17.     }
18.
19.     @Override
20.     public Map<String, String> getQueries(String groupName) {
21.         return queriesGroup.get(groupName);
22.     }
23.
24.     @Override
25.     public String getQuery(String groupName, String queryName) {
26.         return queriesGroup.get(groupName).get(queryName);
27.     }
28.
29. }
```

Nous avons utilisé une **LinkedHashMap**, car elle nous permet de parcourir les requêtes SQL dans leur ordre de déclaration si on veut les exécuter les unes à la suite des autres, par exemple pour enchainer la création de tables.

IV - Schéma XML

IV-A - Génération du schéma

JAXB offre la fonctionnalité de générer un schéma XML, ce qui nous permettra de valider le flux XML reçu. Pour ceci, les quelques lignes de code suivantes suffisent :

Génération du schéma
```
1. final File baseDir = new File(".");
2. class MySchemaOutputResolver extends SchemaOutputResolver {
3.
4.     @Override
5.     public Result createOutput(String namespaceUri, String suggestedFileName) throws IOException {
6.         return new StreamResult(new File(baseDir, suggestedFileName));
7.     }
8. }
9.
10. JAXBContext context = JAXBContext.newInstance(Queries.class);
11. context.generateSchema(new MySchemaOutputResolver());
```

Nous obtenons un fichier schema1.xsd :

Schéma
```
1. <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2. <xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
3.
4.     <xs:element name="queries" type="queries"/>
5.
6.     <xs:complexType name="queries">
7.         <xs:sequence>
8.             <xs:element name="group" type="queryGroup" minOccurs="0" maxOccurs="unbounded"/>
9.         </xs:sequence>
10.     </xs:complexType>
11.
12.     <xs:complexType name="queryGroup">
13.         <xs:sequence>
14.             <xs:element name="query" type="query" minOccurs="0" maxOccurs="unbounded"/>
15.         </xs:sequence>
16.         <xs:attribute name="name" type="xs:string"/>
17.     </xs:complexType>
18.
19.     <xs:complexType name="query">
20.         <xs:simpleContent>
21.             <xs:extension base="xs:string">
22.                 <xs:attribute name="name" type="xs:string"/>
23.             </xs:extension>
24.         </xs:simpleContent>
25.     </xs:complexType>
26. </xs:schema>
```

IV-B - Validation du flux

Nous pouvons utiliser le schéma pour valider le flux XML :

Génération avec validation
```
1. SchemaFactory schemaFactory = SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
2. Schema schema = schemaFactory.newSchema(new File(baseDir, "schema1.xsd"));
3.
4. Unmarshaller unmarshaller = context.createUnmarshaller();
5. unmarshaller.setSchema(schema);
6. Queries unmarshall = (Queries) unmarshaller.unmarshal(xmlStream);
```

En cas de non-conformité du flux XML par rapport au schéma, la méthode unmarshall() lève une javax.xml.bind.UnmarshalException.

De la même manière que nous avons obtenu notre Unmarshaller, on obtient un Marshaller, auquel on peut préciser le schéma, et qui transformera nos objets en flux XML. Nemek me fait remarquer que la sérialisation (le marshalling) ne produit pas nécessairement un XML valide vis-à-vis du XSD, et qu'il est donc conseillé de préciser le schéma.

IV-C - Génération des classes

Le schéma ne sert pas qu'à la validation du flux XML. Nous pouvons aussi l'employer pour générer nos classes, à l'aide de l'utilitaire xjc du JDK :

```
xjc schema.xsd
```

Les classes générées sont par défaut dans le package generated. On retrouve bien nos trois classes Queries, QueryGroup et Query, ainsi qu'une classe ObjectFactory. Je vous laisse découvrir à quoi ressemblent ces classes, elles sont très proches des nôtres, avec essentiellement quelques commentaires Javadoc en plus.

Les deux options les plus utiles de xjc sont :

- -help : affiche l'aide et les différentes options disponibles ;
- -p : permet de préciser le package. Par exemple avec -p fr.atatorus.gen, les classes seront dans fr/atatorus/gen et non plus dans generated ;
- -d : précise le répertoire où seront les classes générées.

V - Conclusion

Voilà, j'espère que ce petit tutoriel vous a permis de découvrir JAXB, et d'apprécier sa facilité d'utilisation. Pour ma part, j'ai vraiment aimé sa simplicité. Ma première expérience avec le binding Java XML a commencé avec JAXP, où on devait tout faire soi-même. La découverte de Castor a été un soulagement, même si j'ai dû me débattre avec les fichiers de mapping. À côté, JAXB est un vrai bonheur !

Si vous voulez découvrir toutes les possibilités de JAXB, je vous renvoie à sa **documentation**.

VI - Remerciements

Je tiens à remercier **Nemek**, **le_y@ms**, **Gueritarish** et **Keulkeul** pour leur aide et leurs critiques apportées à la rédaction de ce tutoriel, ainsi que **_Max_** et **Claude Leloup** pour leur relecture.

VII - Liens

Tutoriel sur la sérialisation XML en Java avec XMLEncoder et XMLDecoder.

Tutoriel sur Jakarta Commons Digester.

Tutoriel sur la sérialisation XML avec XStream.

Tutoriel d'origine sur mon blog.