Sérialisation XML et JSON

Master Technologies de l'Internet 1ère année

Eric Cariou

Université de Pau et des Pays de l'Adour
UFR Sciences Pau – Département Informatique

Eric.Cariou@univ-pau.fr

Mars 2021

Sérialisation XML d'objets Java

- Stockage de contenu d'objets Java dans des fichiers XML
- 3 possibilités en Java
  - Sérialisation native d'objets Java
    - Dépendance avec les classes Java, fichier XML difficile à manipuler indépendamment de cette sérialisation
  - JAXP
    - Java Architecture for XML Processing
    - Permet de parcourir un fichier XML en passant de nœud en nœud
    - Accès bas niveau, doit faire la correspondance à la main entre le contenu XML et les objets Java ainsi que le parcours du XML soi même
  - JAXB
    - Java Architecture for XML Binding
    - Correspondance structure XML / structure classes comme en JPA
      - Via annotation des classes
    - Lecture directe d'objets à partir d'un contenu XML

Sérialisation native Java

- La sérialisation Java permet de sauvegarder le contenu d'objets Java et de les instancier avec ce contenu
  - Sous forme binaire ou XML
- Pour la sérialisation en XML
  - Les champs à sauvegarder doivent être public ou avoir un getter et un setter publics
- Ex. classes `Sport` et `Discipline` du package `donnees`
  - ```
    public class Sport implements java.io.Serializable {
       private int codeSport;
       private String intitule;
       private Set<Discipline> disciplines;
       // getter, setter, constructeurs...
    ```
  - ```
    public class Discipline implements java.io.Serializable {
       private int codeDiscipline;
       private String intitule;
       private Sport sport;
       // getter, setter, constructeurs...
    ```


Sérialisation native Java

- Exemple de sérialisation/désérialisation

```
// création d'un sport avec deux disciplines
Sport sp = new Sport(12, "Ski");
Discipline ds = new Discipline(10, "Descente", sp);
sp.getDisciplines.add(ds);
ds = new Discipline(11,"Biathlon", sp);
sp.getDisciplines.add(ds);

// sérialisation du sport dans un fichier sport.xml
XMLEncoder encoder = new XMLEncoder(new FileOutputStream("sport.xml"));
encoder.writeObject(s);
encoder.close();

// désérialisation du sport à partir du fichier sport.xml
XMLDecoder decoder = new XMLDecoder(new FileInputStream("sport.xml"));
Sport s = (Sport)decoder.readObject();
System.out.println("Sport : "+s.getIntitule());
for (Discipline d : s.getDisciplines())
    System.out.println(" -> "+d.getIntitule());
```

- Affiche dans la console

```
Sport Ski
-> Descente
-> Biathlon
```


Sérialisation native Java

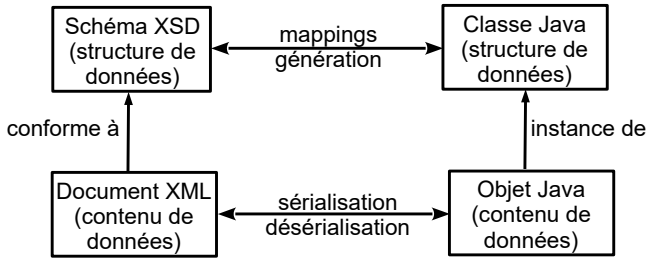
- Contenu du fichier sport.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_162" class="java.beans.XMLDecoder">
  <object class="donnees.Sport" id="Sport0">
    <void property="codeSport">
      <int>12</int>
    </void>
    <void property="disciplines">
      <void method="add">
        <object class="donnees.Discipline">
          <void property="codeDiscipline">
            <int>10</int>
          </void>
          <void property="intitule">
            <string>Descente</string>
          </void>
          <void property="sport">
            <object idref="Sport0"/>
          </void>
        </object>
      </void>
      <void method="add">
        <object class="donnees.Discipline">
          <void property="codeDiscipline">
            <int>11</int>
          </void>
          <void property="intitule">
            <string>Biathlon</string>
          </void>
          <void property="sport">
            <object idref="Sport0"/>
          </void>
        </object>
      </void>
    </void>
    <void property="intitule">
      <string>Ski</string>
    </void>
  </object>
</java>
```


Sérialisation native Java

- Code Java
  - Simple et efficace
- XML généré
  - Très verbeux
    - Beaucoup de références à Java comme les noms de classes
    - Des balises `void` un peu étranges
    - Une balise d'une méthode `add` ?
    - ...
  - Difficile à lire et à comprendre
- Point intéressant
  - La balise définissant un sport possède un `id="Sport0"`
    - Cela permet dans les disciplines de faire le lien vers le sport
  - Les objets forment un graphe alors qu'un fichier XML est un arbre : permet de former le graphe par dessus l'arbre

Sérialisation avec JAXB



- JAXB permet de sérialiser à plus haut niveau
  - Définition de correspondances entre une structure XML et les structures des classes Java
- Le schéma XSD est optionnel
  - Peut déduire la structure du contenu XML (les données) à partir de la structure de la classe et inversement

Principes JAXB

- Package javax.xml.bind
- Deux modes de mappings
  - Annoter des classes Java existantes pour préciser comment/quoi sérialiser en XML
  - Générer un ensemble de classes Java à partir d'un schéma XML
- Dans le code de l'application
  - Utilisation de « marshaller », « unmarshaller » pour écrire ou lire le contenu d'objets dans du XML
  - Transparence complète vis-à-vis de la structure XML, ne manipule que des objets

Annotations JAXB

- Quelques annotations
  - @XmlRootElement : à placer sur une classe pour la définir comme le contenu principal/premier d'un fichier XML
  - @XmlTransient : sur un getter
    - Si on ne veut pas que le contenu de l'attribut soit sérialisé
    - Par défaut, tout attribut d'une classe est sérialisé
  - @XmlType : gestion des types XML
    - Par exemple pour ordonner les champs dans le fichier XML
  - @XmlEnum : pour mapper une énumération
  - ...

Exemple : sport et discipline

- Un sport est composé de disciplines
  - On veut enregistrer un sport avec ses disciplines
  - Classe Sport du package donnees

```
// la classe Sport est l'élément principal
@XmlRootElement
// ordre de la sérialisation des attributs
@XmlType(propOrder = {"codeSport", "intitule", "disciplines"})
public class Sport implements java.io.Serializable {

  private int codeSport;
  private String intitule;
  private Set<Discipline> disciplines;

  // getter, setter, constructeurs...
```

  - Si on ne veut pas sérialiser les disciplines

```
@XmlTransient
public Set<Discipline> getDisciplines() {
  return disciplines;
}
```


Exemple : sport et discipline

- Classe Discipline du package donnees
  - Rien de particulier à préciser, on la définit comme un POJO classique, sans annotations
  - public class Discipline implements java.io.Serializable {

```
private int codeDiscipline;
private String intitule;
private Sport sport;

// getter, setter, constructeurs...
```


Contexte

- JAXBContext
  - Classe qui définit un contexte JAXB à partir d'une liste de classes sérialisables
  - Méthode statique newInstance, deux versions principales
    - static JAXBContext newInstance(Class...)
      - Une ou plusieurs descriptions de classes
    - static JAXBContext newInstance(String package)
      - Nom d'un package dans lequel se trouvent des classes générées à partir d'un schéma XML
  - Avec un contexte, on récupère un « marshaller » ou un « unmarshaller »
    - Marshaller : permet d'enregistrer des objets Java en XML
      - public Marshaller createMarshaller();
    - Unmarshaller : permet d'instancier des objets Java à partir d'un contenu XML
      - public Unmarshaller createUnmarshaller();

Marshaller

- Classe Marshaller
  - void marshal(Object obj, support) : sérialise l'objet obj avec support pouvant être
    - File : descripteur de fichier
    - OutputStream / Writer : flux binaire / texte de sortie
    - XMLEventWriter / XMLStreamWriter : flux spécialisés XML
    - Node : arbre DOM
    - …
- Classe Unmarshaller
  - Object unmarshal(support) : désérialise l'objet à partir d'un support pouvant être
    - File : descripteur de fichier
    - InputStream / Reader : flux binaire / texte d'entrée
    - XMLEventReader / XMLStreamReader : flux spécialisés XML
    - Node : arbre DOM
    - …

Exemple de sérialisation

```
// création du contexte pour la classe Sport
JAXBContext jc = JAXBContext.newInstance(Sport.class);
// création du marshaller
Marshaller marshaller = jc.createMarshaller();
// positionnement d'une propriété pour que le XML soit formaté en sortie
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

// création d'un sport avec deux disciplines
Sport sp = new Sport(12, "Ski");
Discipline ds = new Discipline(10, "Descente", sp);
sp.getDisciplines().add(ds);
ds = new Discipline(11,"Biathlon", sp);
sp.getDisciplines().add(ds);

// sérialisation du sport dans un fichier sport.xml
marshaller.marshal(sp,new File("sport.xml"));
```


Exemple de sérialisation

- L'exécution du code précédent donne le fichier suivant

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sport>
  <codeSport>12</codeSport>
  <intitule>Ski</intitule>
  <disciplines>
    <codeDiscipline>10</codeDiscipline>
    <intitule>Descente</intitule>
  </disciplines>
  <disciplines>
    <codeDiscipline>11</codeDiscipline>
    <intitule>Biathlon</intitule>
  </disciplines>
</sport>
```

- Beaucoup plus lisible et concis que la sérialisation native de Java
- Fichier XML manipulable en dehors d'un programme Java

Exemple de désérialisation

- Si maintenant on désérialise le fichier « sport.xml »

```
// création d'un unmarshaller
JAXBContext jc = JAXBContext.newInstance(Sport.class);
Unmarshaller unmarshaller = jc.createUnmarshaller();

// récupère l'instance du sport dans le fichier XML
Sport sp = (Sport)unmarshaller.unmarshal(new File("sport.xml"));
System.out.println(" Sport " +sp.getIntitule());
for (Discipline ds : sp.getDisciplines()) {
    System.out.println(" -> "+ds.getIntitule());
}
```

- Cela donne l'affichage suivant

```
Sport Ski
-> Descente
-> Biathlon
```


Génération d'un schéma

- A partir de classes annotées, on peut générer le schéma XML correspondant
  - Outil « schemagen » en ligne de commande
    $ schmagen -cp . donnees/Sport.java
  - Donne le fichier XSD suivant

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="sport" type="sport"/>

  <xs:complexType name="sport">
    <xs:sequence>
      <xs:element name="codeSport" type="xs:int"/>
      <xs:element name="intitule" type="xs:string" minOccurs="0"/>
      <xs:element name="disciplines" type="discipline" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="discipline">
    <xs:sequence>
      <xs:element name="codeDiscipline" type="xs:int"/>
      <xs:element name="intitule" type="xs:string" minOccurs="0"/>
      <xs:element ref="sport" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```


Limites du mapping XML

- Le schéma généré est syntaxiquement valide mais sémantiquement faux
  - Un sport référence des disciplines et une discipline référence un sport : références croisées
  - Si on génère les classes Java (voir juste après) avec ce schéma et qu'on essaye de sérialiser un sport avec des disciplines, ça plante

    com.sun.istack.internal.SAXException2: A cycle is detected in the object graph. This will cause infinitely deep XML: Sport{codeSport=12, intitule=Ski} -> Discipline{codeDiscipline=11, intitule=Biathlon, sport=Sport{codeSport=12, intitule=Ski}} -> Sport{codeSport=12, intitule=Ski}]
  - Données en XML = arbre avec éléments imbriqués
    - Ne peut pas gérer une référence croisée entre deux éléments, les deux éléments se retrouveraient imbriqués mutuellement l'un dans l'autre
  - Solutions
    - Supprimer la référence vers un sport dans la classe Discipline
    - Ou marquer le getter du sport comme transient :
      @XmlTransient
      public Sport getSport() { return sport }

      Dans la balise XML définissant une discipline, ne génère alors plus la ligne <xs:element ref="sport" minOccurs="0"/>

Limites du mapping XML

- Retour sur exemple de sérialisation (transparents 14 à 16)
  - La classe Discipline a un attribut de type Sport
  - Dans les balises de disciplines, aucune référence vers leur sport
    - A la sérialisation, le moteur JAXB a ignoré de lui-même la référence croisée
    - … mais pas le générateur de schéma
  - Lors de la désérialisation d'un sport, en conséquence, l'attribut sport d'une discipline n'est pas positionné
  - ...

```
for (Discipline ds : sp.getDisciplines()) {
    System.out.println(" -> "+ds.getIntitule() + " de sport "+ds.sport);
}
```

donne maintenant

```
Sport Ski
-> Descente de sport null
-> Biathlon de sport null
```


Limites du mapping XML

- Format de représentation des données
  - Java : graphe d'objets avec des références entre objets
  - XML : arbre avec imbrication des éléments
- Quand on prévoit de sérialiser en XML
  - Faire attention à la structure des classes coté Java
  - Utiliser une structure à sérialiser formant un arbre

Génération de classes annotées

- Opération inverse, on génère des classes Java annotées à partir d'un schéma XSD
  - Exemple de schéma (similaire dans le principe au précédent avec un sport composé de disciplines)
  - Fichier « sport.xsd »

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="sport">
      <xs:complexType>
         <xs:sequence>
            <xs:element type="xs:byte" name="codeSport"/>
            <xs:element type="xs:string" name="intitule"/>
            <xs:element ref="discipline" maxOccurs="unbounded" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="discipline">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="codeDiscipline" type="xs:byte"/>
            <xs:element name="intitule" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```


Génération de classes annotées

- Exécution de l'outil « xjc » en ligne de commande
  $ xjc -p donnees sport.xsd
  - Génère les deux classes Sport et Discipline

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"codeSport","intitule","discipline"})
@XmlRootElement(name = "sport")
public class Sport {
    protected byte codeSport;
    @XmlElement(required = true)
    protected String intitule;
    protected List<Discipline> discipline;
…}
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"codeDiscipline","intitule"})
@XmlRootElement(name = "discipline")
public class Discipline {
    protected byte codeDiscipline;
    @XmlElement(required = true)
    protected String intitule;
… }
```


Génération de classes annotées

- Une factory d'objets est également générée avec une méthode de création de chacune des classes métier générées
  - @XmlRegistry

```
public class ObjectFactory {

    public ObjectFactory() {
    }

    public Sport createSport() {
        return new Sport();
    }

    public Discipline createDiscipline() {
        return new Discipline();
    }
}
```


Utilisation de classes générées

- Les classes Sport et Discipline sont des POJOs classiques
  - On les utilise normalement
- Si on crée un contexte par rapport à un package
  - Le moteur JAXB s'attend à trouver une classe ObjectFactory pour gérer la création des objets
- Exemple pour le package « donnees »
  - JAXBContext jc = JAXBContext.newInstance("donnees");
  - Le reste du code de marshalling/unmarshalling ne change pas ensuite

Sérialisation JSON

- JSON : JavaScript Object Notation
  - Format textuel de données structurées
  - Mais sans description explicite de la structure des données
- Nombreux outils Java de manipulation de contenu JSON
- Dans la standardisation Java
  - JSON-P : JSON Processing
    - API « bas-niveau » pour lire du JSON
    - Équivalent de JAXP pour XML
  - JSON-B : JSON Binding
    - API haut-niveau de correspondance entre classes Java et contenu JSON
    - Équivalent de JAXB pour XML
    - Implémentation de référence : Yasson

Sérialisation JSON

- Correspondance objet / contenu JSON
  - Attribut d'une classe = paire clé / valeur
  - Utilisation d'annotations pour contrôler la sérialisation
  - Par défaut, tout attribut à accès public d'un objet est sérialisé
    - Attribut public ou avec un getter et un setter public
    - Les objets de valeur null ne sont pas sérialisés
- On reprend les POJO Sport et Discipline sans annotation

```
public class Sport implements java.io.Serializable {
    private int codeSport;
    private String intitule;
    private Set<Discipline> disciplines;
    // getter, setter, constructeurs...
```

```
public class Discipline implements java.io.Serializable {
    private int codeDiscipline;
    private String intitule;
    private Sport sport;
    // getter, setter, constructeurs...
```


Sérialisation JSON

- Package javax.json.bind
  - Classes et interfaces permettant la sérialisation
  - JsonbBuilder
    - Pour sérialiser et désérialiser
    - Dans un flux (fichier …) ou une chaîne de caractère
- Exemple précédent

```
// création d'un sport avec deux disciplines
Sport s1 = new Sport(12, "Ski");
Discipline ds = new Discipline(10, "Descente", s1);
sp.getDisciplines.add(ds);
ds = new Discipline(11,"Biathlon", s1);
sp.getDisciplines.add(ds);

// sérialisation du sport dans une chaîne
Jsonb builder = JsonbBuilder.create();
String json = builder.toJson(s1);
System.out.println(json);

// désérialisation de la chaine et affichage du sport
Sport s2 = builder.fromJson(json, Sport.class);
System.out.println("Sport : "+s2.getIntitule());
for (Discipline d : s2.getDisciplines())
    System.out.println(" --> "+d.getIntitule() + " de sport "+d.getSport());
```


Sérialisation JSON

- Code très simple : 2 lignes de code pour sérialiser
  - Mais ne marche pas, lève plusieurs exceptions

```
javax.json.bind.JsonbException: Unable to serialize property 'disciplines' from Sport
javax.json.bind.JsonbException: Unable to serialize property 'sport' from Discipline
javax.json.bind.JsonbException: Recursive reference has been found in class class Sport
```

  - On retrouve le même problème qu'en XML
    - Les objets ont des références croisées et forment un graphe
- Solution
  - Rendre transient l'attribut sport dans la classe Discipline

```
@JsonbTransient
private Sport sport;
```

  - Donne le résultat suivant avec le problème que l'attribut sport n'est pas initialisé dans les objets Discipline

```
{"codeSport":12,"disciplines":[{"codeDiscipline":10,"intitule":"Descente"},
{"codeDiscipline":11,"intitule":"Biathlon"}],"intitule":"Ski"}

Sport : Ski
--> Biathlon de sport null
--> Descente de sport null
```


Sérialisation JSON

- On peut configurer la sérialisation
  - Renommer les clés au lieu d'utiliser le nom de l'attribut par défaut, sérialiser les attributs de valeurs null, gérer des énumérations, formater le JSON généré (par défaut s'écrit sur une seule ligne) ...
- Exemple : formatage avec retour à la ligne, renommage automatique des clés et sauvegarde dans un fichier

```
JsonbConfig config = (new JsonbConfig()).withFormatting(true);
config.withPropertyNamingStrategy(PropertyNamingStrategy.LOWER_CASE_WITH_DASHES);
Jsonb builder = JsonbBuilder.create(config);
builder.toJson(s1, new FileOutputStream("sports.json"));
```

- Contenu du fichier sports.json

```
{
  "code-sport": 12,
  "disciplines": [
    {
      "code-discipline": 10,
      "intitule": "Descente"
    },
    {
      "code-discipline": 11,
      "intitule": "Biathlon"
    }
  ],
  "intitule": "Ski"
}
```


Conclusion

- Sérialisation d'objets Java dans des fichiers textuels structurés
  - XML avec JAXB
  - JSON avec JSON-B
- Code Java très simple
  - Quelques annotations dans les POJO
  - Quelques lignes de code pour sérialiser/désérialiser
- Limites/contraintes
  - Les objets doivent former des structures en arbre ou en liste
  - Les graphes d'objets avec des références croisées ne sont pas gérés correctement