Tehnici avansate de programare Curs 3

Cristian Frăsinaru

acf@infoiasi.ro

Facultatea de Informatică

Universitatea "Al. I. Cuza" laşi

JAXB

Cuprins

- Ce este JAXB?
- Generarea claselor
- Marshalling
- Unmarshalling
- Validarea datelor

Ce este JAXB?

JAXB (*The Java Architecture for XML Binding*) este o tehnologie care permite generarea de clase Java corespunzătoare elementelor unui document XML. Caracteristicile principale sunt:

- Ascunde detaliile necesare procesării datelor
- Oferă o modalitate orientată obiect de lucru cu XML
- Generarea claselor este realizată pe baza regulilor de validare xs prin intermediul unui compilator
- Marshalling, Unmarshalling
- Permite separarea activităţii programatorilor de designeri

Legarea structurii XML de clase

Această etapă presupune efectuarea următorilor paşi:

- Scrierea documentului xs ce conţine regulile de validare.
- Crearea (generarea) documentelor XML care respectă schema creată.
- Definirea unei schemă de mapare între unitati XML şi unitati Java.
- Generarea claselor Java din schema XML.
- Compilarea claselor generate.



Legarea datelor XML de obiecte

Această a doua etapă se realizează prin intermediul unei aplicații client ce are interfață cu utilizatorul și care va efectua, în general, următoarele operațiuni:

- Unmarshalling: crearea obiectelor Java corespunzatoare documentului XML, eventul cu validarea acestuia.
- Modificarea datelor din arborele de obiecte prin intermediul aplicaţiei propriu-zise.
- Marshalling: salvarea informaţiei înapoi în documentul XML, eventual executând validarea acesteia.



JAXB API

Tehnologia JAXB este inclusă în JWSDP (Java Web Services Developer Pack)

```
jwsdp
 jaxb
                  <-- utilitare pentru generarea claselor (xjc)
  bin
  docs
                  <-- documentatie
  samples
                  <-- aplicatii demonstrative
   lib
                  <-- arhive jar specifice JAXB:
                      jaxb-api.jar, jaxb-xjc.jar, jaxb-impl.jar, jaxb-l:
 jaxp
   lib
                  <-- jaxp-api.jar
   endorsed
                  <-- arhive pentru procesare XML:
                      sax.jar, dom.jar, xercesImpl.jar, xalan.jar
 jwsdp-shared
                  <-- jax-qname.jar, namespace.jar, xslib.jar, relaxngDa
   lib
 apache-ant
   lib
                  <-- ant.jar
```

Generarea claselor

Generarea claselor se face cu ajutorul compilatorului de scheme xjc.

- -nv: inhibă validare documentului;
- -b <fisier>: specifica un fisier de mapare;
- -p <pachet>: specifica numele pachetului ce va fi generat;
- -classpath <fisier>: specifica unde se gasesc clasele utilizatorului;
- -b <fisier>: specifica un fisier de mapare;
- -xmlschmea: fisierul cu reguli este de tip XML Schema (implici);
- -relaxng: fisierul cu reguli este de tip RELAX NG (neimplementat);
- -dtd: fisierul cu reguli este de tip DTD (neimplementat);

Exemplu

Să considerăm fișierul catalog.xsd

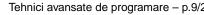
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="catalog" type="CatalogDef"/>
<xs:complexType name="CatalogDef">
  <xs:sequence>
    <xs:element name="film" type="Film" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Film">
  <xs:sequence>
    <xs:element name="nume" type="xs:string"/>
    <xs:element name="durata" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Compilarea schemei

Compilarea se va face astfel:

xjc catalog.xsd -p demo şi va avea ca rezultat crearea directorului demo care va conține:

- interfeţele Catalog, CatalogDef, Film
- directorul impl care include clasele CatalogImpl, FilmImpl
- clase specifice JAXB



Interfețele generate

```
package demo;
public interface Film {
    java.lang.String getNume();
    void setNume(java.lang.String value);
    int getDurata();
    void setDurata(int value);
package demo;
public interface CatalogDef {
    java.util.List getFilm();
package demo;
public interface Catalog
    extends javax.xml.bind.Element, demo.CatalogDef
{}
```

Clasele generate

Clasa FilmImpl corespunde tag-ului XML film:

```
protected java.lang.String _Nume;
protected int _Durata;

java.lang.String getNume() {...};

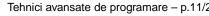
void setNume(java.lang.String value) {...};
int getDurata() {...};

void setDurata(int value) {...};
```

Clasa CatalogImpl corespunde tag-ului XML catalog:

```
protected ListImpl _Film = new ListImpl(new java.util.ArrayList());
public java.util.List getFilm() {...}
```

Există o mapare implicită între tipurile de date XSD şi tipurile de date Java.

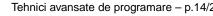


Compilarea claselor

Unmarshalling: din XML în Java

Un document XML de test

Să considerăm fiserul catalog.xml ce respectă schema anterioară:



Unmarshalling

```
import java.io.*;
import java.util.*;
import javax.xml.bind.*;
import demo.*;
public class Main {
  public static void main( String[] args ) {
  try {
    //1. Crearea unui obiect de tip JAXBContext pentru manevrarea
    //claselor utilitare din pachetul demo
    JAXBContext jc = JAXBContext.newInstance("demo");
    //2. Crearea unui obiect de tip Unmarshaller
    Unmarshaller u = jc.createUnmarshaller();
    //3. Unmarshall: transferul datelor din XML in objecte
    Catalog c = (Catalog)u.unmarshal(
                  new FileInputStream("catalog.xml"));
```

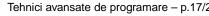
Unmarshalling (continuare)

```
//4. Vizualizarea informatiei citite
  afiseaza(c);
} catch(JAXBException e) {
  e.printStackTrace();
} catch(IOException e) {
  e.printStackTrace();
afiseaza(Catalog c) {
  for(Iterator it = c.getFilm().iterator(); it.hasNext(); ) {
Film f = (Film)it.next();
    System.out.println( f.getNume() + ": " + f.getDurata() + " min.")
```

Validarea datelor XML citite

Se face prin apelarea metodei setValidating(true) pentru obiectul de tip Unmarshaller, înainte de operația propriu-zisă și tratarea excepțiilor de tip UnmarshallException ce pot fi generate dacă datele XML nu respectă schema specificată.

```
try {
   //2.1 Activarea validarii
   u.setValidating(true);
   ....
} catch(UnmarshalException e) {
   System.out.println("Date invalide: " + e);
}
```



Marshalling: din Java în XML

Actualizarea datelor

Folosind clasele create putem accesa și modifica obiectele ce reprezintă informațiile XML.

```
//5. Procesarea/Actualizarea informatiei
ObjectFactory obj = new ObjectFactory();
Film f = obj.createFilm();
f.setNume("Tarzan");
f.setDurata(80);
List filme = c.getFilm();
filme.add(f);
```

Marshalling

Transferăm datele, în format XML, către o destinație externă reprezentată printr-un flux de ieşire pe octeți.

```
//6. Marshall: transferul datelor in format XML
Marshaller m = jc.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
m.marshal(c, System.out);
```

Validarea obiectelor Java

Putem verifica dacă obiectele Java ce vor fi transformate în XML respectă schema specificată. Acest lucru se face prin intermediul unui obiect de tip Validator:

```
try {
    //5.1 Procesarea/Actualizarea informatiei
    ...
    //5.2 Validarea
    Validator v = jc.createValidator();
    boolean valid = v.validateRoot(c);
    System.out.println(valid);

    //6. Marshalling
    ...
} catch( ValidationException e) {
        System.out.println( "Date invalide:" + e );
}
```