JAXB pour le XML-Binding

Eric BELLARD

Introduction

But du document

L'objectif de ce document est :

- présenter le framework JAXB
- construire pas à pas un framework technique pour effectuer du *round-tripping*[1].

L'application **Create-marshall** issue des exemples SUN d'utilisation de JAXB sera utilisée et modifiée pour tester notre framework**.**

Lexique

JAXB : Java Architecture for XML Binding.
Marshall : Transformation d'un objet en données persistante. Exemple :transformation d'un objet Java en XML.

[1] Voir lexique

Marshalling : Action de transformer d'objets Java en un fichier XML.
Unmarshalling : Action de transformer un fichier XML en objet Java.
Round-tripping : Processus complet de transformation d'un fichier XML en objet Java suivi d'une re-tranformation en fichier XML. Un round-tripping *effectif* doit produire en sortie le même fichier XML qu'en entrée.

Articles liés

Practical Data Binding : Looking into JAXB Part1
Practical Data Binding : Looking into JAXB Part2
Java Architecture for XML Binding
Code Fast, Run Fast with XML Data Binding
Portail articles

JAXB (Java Architecture for XML Binding)

JAXB est une librairie Java implémentée par SUN pour simplifier les processus de transformation d'objets Java en fichier XML, et de fichier XML en objets Java.

L'utilisation de JAXB permet un gain de temps significatif par rapport à l'utilisation de parser XML de bas niveau tel *SAX* ou *DOM* pour effectuer du *round-tripping*[2].

Le fonctionnement de JAXB est illustré par le schéma ci-dessous :

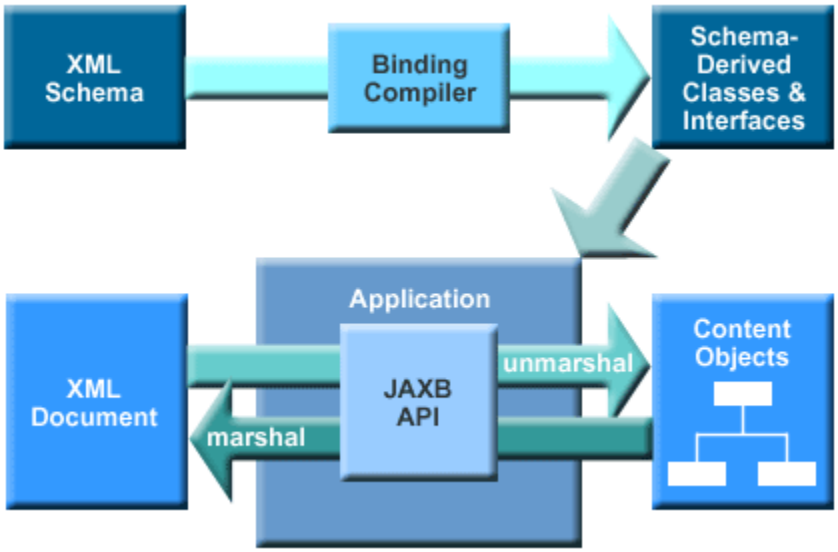

Figure 1.Fonctionnement général de JAXB

Un XML-Schema est compilé avec le compilateur JAXB. Plusieurs types d'objets sont générés :

- **Content Objects** : ensemble de classes et interfaces représentant la structure du

[2] Voir le lexique

fichier XML ;
- **ObjectFactory** : permet la création de *Content Objets* ;
- **Marshaller**[3] : permet de transformer des *Content Objects* en fichier XML ;
- **Unmarshaller**[4] : permet de créer des *Content Objects* à partir d'un fichier XML.

La transformation d'un fichier XML respectant ce xml-schema en *Content Objects* java est alors possible en moins d'une dizaine de lignes de code. Idem pour la transformation d'un fichier XML en *Content Objects*.

Utilisation de JAXB

Compilation du Xml-Schema

Principe

La première étape consiste à compiler le xml-schema afin de générer les objets dérivés.

Figure 2.Compilation du XML-Schema

Réalisation

Pour la compilation, il est nécessaire de spécifier au compilateur :
- le package racine des classes à générer ;
- le répertoire racine où placer ces classes.

Ci-dessous un script « .bat » paramétrable pour compiler un xml-schema :

```
# emplacement du compilateur JAXB
set XJC=C:\jwsdp-1.3\jaxb\bin\xjc.bat

# package racine dans lequel doivent etre generer les classes derives
set PACKAGE=primer.po

# emplacement du xml schema
set XSD=C:\jwsdp-1.3\jaxb\samples\create-marshal\po.xsd

# repertoire dans lequel generer les sources
set SRC=C:\jwsdp-1.3\jaxb\samples\create-marshal\src

# execution de la commande de compilation
```

[3] Voir lexique *Marshalling*
[4] Voir lexique *Unmarshalling*

```
%XJC% -p %PACKAGE% %XSD% -d %SRC%
```

Figure 3.Script paramétrable pour la compilation d'un XML-Schema

Dans ce script :

- L'emplacement du compilateur est défini par la variable XJC ;
- Le package racine des classes à générer est défini par la variable PACKAGE ;
- L'emplacement du XML-Schema est défini par la variable XSD ;
- L'emplacement du répertoire racine des sources est défini par la variable SRC.

Pour pouvoir réutiliser ce script dans tout contexte, modifier les valeurs des variables décrites ci-dessus.

XML-Binding avec JAXB

Principe

Une fois le XML-Schema compilé, JAXB est utilisable pour effectuer des transformations *content objects* ⇔ document XML.
Les figures 4 et 5 illustrent ce fonctionnement.

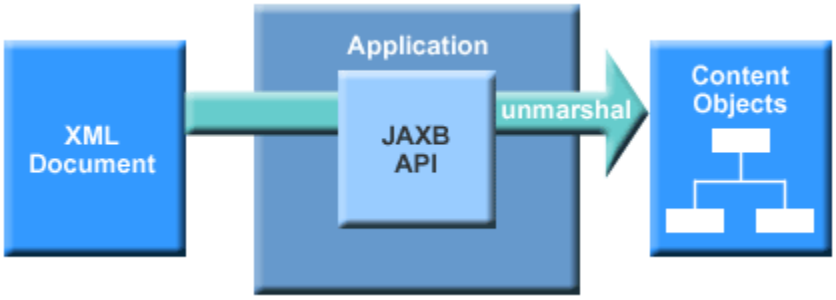

Figure 4.Transformation d'un document XML en objets Java dérivés du XML-Schema

La transformation ci-dessus s'effectue en utilisant le *marshaller* généré.

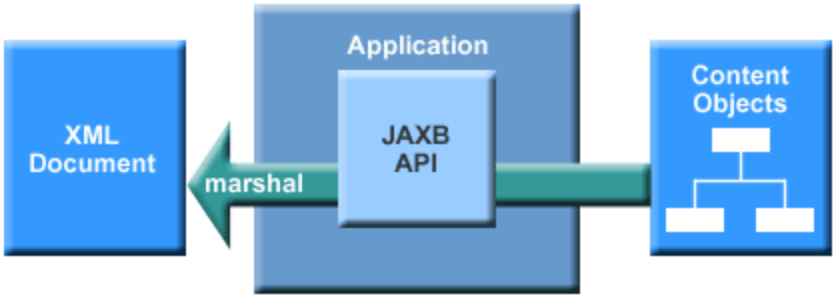

Figure 5.Transformation d' objets Java dérivés du XML-Schema en un document XML

La transformation ci-dessus s'effectue en utilisant le *unmarshaller* généré.

Réalisation dans le framework technique

Il est intéressant de rassembler ces deux fonctionnalités au sein d'une même classe. Elle aura en plus la responsabilité d'initialiser le contexte JAXB. Elle sera nommée JaxbXmlSerializer.
Son comportement sera défini dans l'interface IXmlSerializer.
L'exception de type XmlSerializerException sera lancée en cas de problème: validation, marshall, unmarshall...

Ces classes sont représentées dans le diagramme de classe ci-dessous :

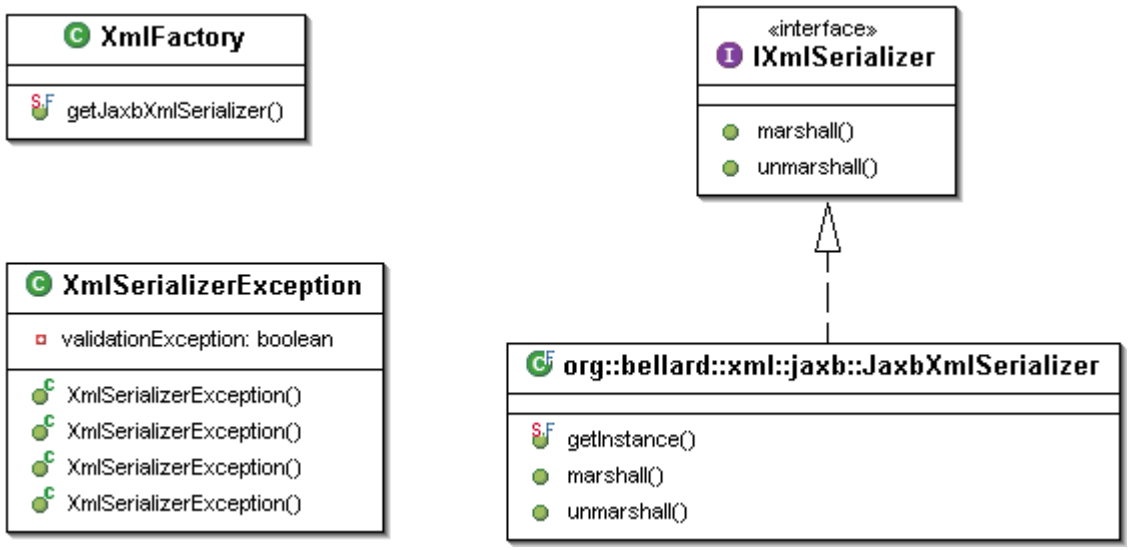

Figure 6.Diagramme de classe du serializer XML JAXB

Implémentation des classes de transformation objets JAXB ⇔ objets métiers

Principe

JAXB permet de manipuler uniquement les content objects issus du XML-Schema : seul ces types d'objets peuvent être serializer/deserializer.

Il est donc nécessaire de créer un **pont** entre les objets d'une application existante et ces *content objets* dérivés du XML-Schema. Dans un cas type la structure des objets de l'application cible est très proche de celle des objets dérivés. La transformation est donc simple.
Le meilleur des cas se présente quand les objets applicatifs sont identiques aux *contents objects* c'est-à-dire qu'ils possèdent les même accesseurs (getter/setters) sur leurs propriétés. Dans ce cas, il suffit de faire dériver les objets applicatifs des interfaces définissant le comportement des *content objects* et les objets applicatifs sont alors directement « *mashallizable/unmarshallizabl*e ». Dans la pratique ce cas ne se présente que très rarement.
Il ne faut donc pas compter dessus.
Le pont est réalisé par l'implémentation d'objet dit « **Transformer »** associé à chaque objet applicatif pour les transformer en objet proche du XML-Schema. Cet objet possèdera 2 méthodes :

- **toContentObject** pour transformer un objet applicatif en *content object*;
- **toApplicationObject** pour transformer un *content object* en objet applicatif.

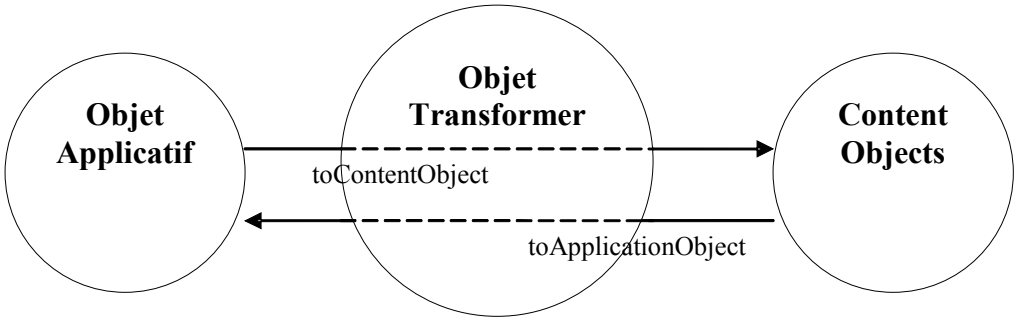

Figure 7.Pont entre les objets applicatifs et les objets dérivés du XML-Schema

Réalisation dans le framework technique

Ce comportement est décrit dans l'interface **ITransformer**.
Dans l'implémentation du serializer, un objet de type ITransfomer sera utilisé pour transformer :

- les content objects après unmarshallisation en objets applicatifs ;
- les objets applicatifs en content objects pour marshallisation.

Le diagramme de classe de la figure 6 est alors modifié pour intégrer ce nouveau type d'objet :

Figure 8.Diagramme de classe du Serializer XML JAXB

Un **Helper** est fourni pour simplifier la transformation à la chaîne d'objets (Collection, List…). Il peut être enrichi au besoin. (voir la classe *TransformerHelper*)

Exemple d'utilisation : create-marshall

Create-marshall est un exemple de mashallisation de *content objects* en fichier XML fourni avec JAXB. L'objet principal est un **PurchaseOrder**.

Cet exemple a été modifié pour intégrer les classes techniques décrites dans les parties précédentes et pour offrir ainsi des fonctionnalités de mashallisation/unmarshallisation.

La classe **PrimerXmlSerializer** permet mashalliser/un mashalliser des objets PurchaseOrder en XML.

Des objets applicatifs proches des contents objects ont été créé pour simuler l'utilisation dans un contexte réel : **PurchaseOrderBo**, **ItemBo**, **USAddressBo**.

Les objets Transformer associés ont été implémentés : **PurchaseTransformer**, **USAddressTransformer**, **ItemTransformer**.

Ci-dessous le diagramme de classe de cet exemple :

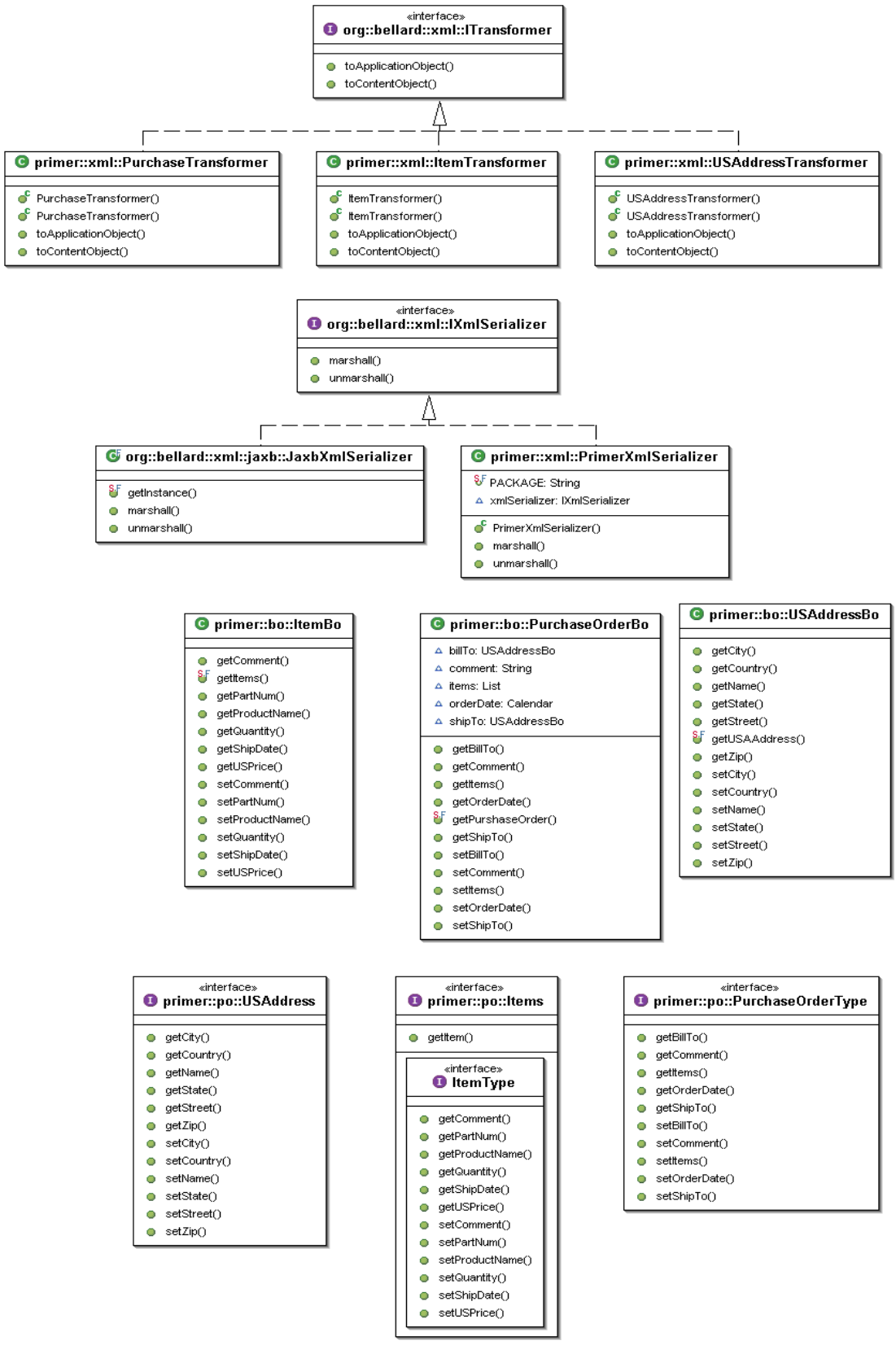

Figure 9.Diagramme de classe de l'exemple Create-marshall

Annexe

Installation de JAXB

JAXB ainsi que ces dépendances sont incluses dans le « *Web Services Developper Pack* » téléchargeable sur le site de SUN. Suivre la documentation en ligne pour l'installation.

Installation de l'exemple create-marshall

Le zip fournit est un projet eclipse. Il fait référence à une librairie utilisateur **JAXB** contenant les jars de JAXB et ses dépendances. Le projet a été compilé/exécuté avec le JDK 1.4.2_04.
JUnit a été utilisé pour la création des tests unitaires.

Table des illustrations

Auteur

Eric BELLARD
eric_bellard@yahoo.com