# Java bean a XML con JAXB

De ChuWiki

## Contenido


- 1 Obtener las clases java a partir de un XSD
  - 1.1 con Maven
  - 1.2 Con eclipse
- 2 Anotaciones
- 3 El código de Java a XML
  - 3.1 Con @XmlRootElement
  - 3.2 Sin @XmlRootElement
- 4 De XML a Java
  - 4.1 Con @XmlRootElement
  - 4.2 Sin @XmlRootElement


JAXB (http://www.oracle.com/technetwork/articles/javase/index-140168.html) es una especificación para facilitar el acceso a ficheros XML desde java. Las versiones modernas de java, desde la 1.6 incluida, vienen con su propia implementación, por lo que podemos usarlo sin necesidad de añadir librerías adicionales. Veamos aquí algunos ejemplos de cómo usar *JAXB*.

## Obtener las clases java a partir de un XSD

### con Maven

Si disponemos de uno o varios ficheros *XSD* donde se especifique cómo es un *XML* válido, usando el plugin maven-jaxb2-plugin (https://java.net/projects/maven-jaxb2-plugin/pages/Home) de Maven el proceso es bastante inmediato.

En el fichero *pom.xml* de nuestro proyecto *maven* debemos poner varias cosas.

Lo primero, asegurarnos que se usa una versión de java en la que ya esté incluida una implementación *JAXB*, vale la 1.6 o superior, pondremos la 1.7. Por supuesto, el compilador java que tengamos instalado debe ser también 1.6 o superior.

```
<project>
   ...
   <build>
      ...
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.3</version>
            <configuration>
               <source>1.7</source>
               <target>1.7</target>
            </configuration>
         </plugin>
         ...
      </plugins>
   </build>
</project>
```

Si no ponemos esto, posiblemente no tengamos problemas con maven, pero sí puede darnos algunos warning si importamos el proyecto en algún IDE como eclipse, indicándonos que las clases propias de JAXB no están accesibles.

El siguiente paso, es colocar nuestros ficheros *xsd* en algún sitio del proyecto. La ubicación donde el plugin busca por defecto es *src/main/resources*, así que colocamos ahí un fichero *xsd* como por ejemplo PurchaseOrders.xsd (https://github.com/chuidiang/chuidiang-ejemplos-google-code/blob/master/ejemplos-jaxb/src/main/resources/PurchaseOrders.xsd)

En el fichero *pom.xml* de nuestro proyecto ponemos ahora el plugin de *jaxb* y lo configuramos

```
        <plugin>
           <groupId>org.jvnet.jaxb2.maven2</groupId>
           <artifactId>maven-jaxb2-plugin</artifactId>
           <version>0.12.1</version>
           <executions>
              <execution>
                 <goals>
                    <goal>generate</goal>
                 </goals>
              </execution>
           </executions>
        </plugin>
```

Simplemente colocar el plugin y decirle que genere los fuentes java a partir del xsd. Si compilamos nuestro proyecto maven, los fuentes se generarán y se guardarán en el directorio *target/generated-sources/xjc*. De ahí hacia abajo aparecerá el paquete de las clases java, que coincidirá con el *targetNamespace* que aparezca en el fichero *xsd*. En nuestro ejemplo, el fichero *xsd* tiene esto

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://tempuri.org/po.xsd"
xmlns="http://tempuri.org/po.xsd" elementFormDefault="qualified">
```

donde *targetNamespace* es *http://tempuri.org/po.xsd*, por lo que el paquete de nuestras clases será *org.tempuri.po*.

Las clases compiladas aparecerán en el lugar habitual de *maven*, es decir, en *target/classes*, por lo que las tendremos disponibles si montamosnuestro proyecto maven en nuestro IDE favorito.

Si nos fijamos en el código java generado (pegamos un trozo a continuación)

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "PurchaseOrderType", propOrder = {
    "shipTo",
    "billTo",
    "comment",
    "items"
})
public class PurchaseOrderType {

    @XmlElement(required = true)
    protected USAddress shipTo;
    @XmlElement(required = true)
    protected USAddress billTo;
    protected String comment;
    @XmlElement(required = true)
    protected Items items;
    @XmlAttribute(name = "orderDate")
    @XmlSchemaType(name = "date")
    protected XMLGregorianCalendar orderDate;
```

puede verse que es una clase java en la que se han añadido anotaciones estilo *@XmlType*, *@XmlElement*, etc. Estas anotaciones son propias de *jaxb* y son las que necesitará *jaxb* para luego saber convertir una instancia de nuestra clase java en su equivalente XML y viceversa. Esto nos abre otra posiblidad que vemos a continuación y es que si no tenemos el fichero *xsd*, podemos directamente crear a mano nuestras clases java y poner las anotaciones adecuadas, como se ve en el siguiente apartado.

## Con eclipse

Con eclipse también es inmediato. Basta poner el fichero xsd en nuestro proyecto y luego, botón derecho el ratón sobre el para sacar el menú y seleccionar "generate" -> "JAXB classes".

# Anotaciones

Nuestro bean java necesita llevar unas anotaciones especiales que ayuden a convertir el java bean en XML. Es también importante, al ser un java bean, que tenga un constructor sin parámetros.

La clase debe llevar la anotación *@XmlRootElement* si va a poder ser raíz del documento, o *@XmlType* si no necesita serlo.

```
@XmlRootElement
public class UnaClase {
...
}
```

```
@XmlType
public class OtraClase {
...
}
```

A cualquiera de ellos podemos ponerle entre paréntesis el nombre de tag que queramos que salga en el fichero XML para esta clase. Sería algo como esto

```
@XmlRootElement(name="Un_Nombre")
@XmlType(name="Otro_Nombre")
```

Para cada atributo de la clase que queramos que salga en el XML, el método get correspondiente a ese atributo debe llevar una anotación *@XmlElement*, a la que a su vez podemos ponerle un nombre

```
@XmlRootElement(name="La_Clase")
public class UnaClase {
   private String unAtributo;

   @XmlElement(name="El_Atributo")
   String getUnAtributo() {
      return this.unAtributo;
   }
}
```

Si el atributo es una colección (array, list, ...) debe llevar dos anotaciones, *@XmlElementWrapper* y *@XmlElement*, esta última, por supuesto, con un nombre si lo queremos

```
@XmlRootElement(name="La_Clase")
public class UnaClase {
   private String [] unArray;

   @XmlElementWrapper
   @XmlElement(name="Elemento_Array")
   String [] getUnArray() {
      return this.unArray;
   }
}
```

Si el atributo es otra clase (otro java bean), le ponemos igualmente *@XmlElement* al método get, pero la clase que hace de atributo debería llevar a la vez sus anotaciones correspondientes.

# El código de Java a XML

## Con @XmlRootElement

El código java para conseguir el fichero XML es muy fácil. Sería el siguiente

```
// Instanciamos el contexto, indicando la clase que será el RootElement.
JAXBContext jaxbContext = JAXBContext.newInstance(UnaClase.class);

// Creamos un Marshaller, que es la clase capaz de convertir nuestro java bean
// en una cadena XML
Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

// Indicamos que vamos a querer el XML con un formato amigable (saltos de linea,
// sangrado, etc)
jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

// Hacemos la conversión llamando al método marshal, pasando una instancia del java
// bean que quermos convertir a XML y un OutpuStream donde queramos que salga el XML,
// en esta caso, la salida estándar. Podría ser un fichero o cualquier otro Stream.
jaxbMarshaller.marshal(unaInstanciaDeUnaClase, System.out);
```

## Sin @XmlRootElement

Este código funciona siempre que partamos de la clase que tenga la anotación *@XmlRootElement*, como en este ejemplo que hemos hecho a mano. Pero si cogemos el ejemplo del primer apartado, en el que generamos la clase java PurchaseOrderType.java a partir del XSD, ahí no se ha generado un *@XmlRootElement*, por lo que este código tal cual nos daría error. El código correcto cuando partimos de una clase java que no está anotada como *@XmlRootElement*, sino sólo como *@XmlType*, puede ser el siguiente

```
    /* Instanciamos la clase y rellenamos algunos de sus campos */
    PurchaseOrderType purchase = new PurchaseOrderType();
    purchase.setComment("This is a comment");
    USAddress address = new USAddress();
    address.setCity("New York");
    purchase.setShipTo(address);

    /* Contexto JAXB con la clase que nos interesa */
    JAXBContext context = JAXBContext.newInstance(PurchaseOrderType.class);

    /* Se crea el "marshaller", que es la clase que convierte de java a XML.
       Indicamos que queremos la salida formateada, para que quede bonito */
    Marshaller marshaller = context.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

    /* Este es el truco al no ser @XmlRootElement. */
    QName qName = new QName("info.source4code.jaxb.model", "purchaseOrder");
    JAXBElement<PurchaseOrderType> root = new JAXBElement<PurchaseOrderType>(
          qName, PurchaseOrderType.class, purchase);

    marshaller.marshal(root, System.out);
```

El código es más o menos como antes. Primero se instancia la clase *PurchaseOrderType* que queremos convertir en *XML* y le damos algunos valores a sus campos. Luego creamos el contexto *JAXB* y el *marshaller*, encargado de hacer la conversión a *XML*.

La "diferencia" está en el *JAXBElement root* que hemos creado. Al no tener la clase *PurchaseOrderType* un *@XmlRootElement*, al *marshaller* no le podemos pasar la clase tal cual, sino que debemos crear y pasarle un *JAXBElement*. Ese *JAXBElement* se crea de esa forma tan rara:

- Creando un *QName* en el que el primer parámetro es "info.source4code.jaxb.model" y el segundo el nombre del tag XML que queremos como raíz del documento XML, es decir, en este caso, <purchaseOrder>.... </purchaseOrder>
- Creando el *JAXBElement*, pasando como parámetro el *QName*, la clase y la instancia de esa clase.

Finalmente, al *marshaller* se le pasa el *JAXBElement* creado (*root*), en vez de directamente la instancia de la clase como antes (*purchase*).

Afortunadamente, el plugin de maven crea una pequeña clase de utilidad, *ObjectFactory*, que nos hace más fácil la creación de este *JAXBElement*. El código equivalente a esas líneas puede ser este

```
//      QName qName = new QName("info.source4code.jaxb.model", "purchaseOrder");
//      JAXBElement<PurchaseOrderType> root = new JAXBElement<PurchaseOrderType>(
//            qName, PurchaseOrderType.class, purchase);

      ObjectFactory factory = new ObjectFactory();
      JAXBElement<PurchaseOrderType> root = factory.createPurchaseOrder(purchase);
```

que es más sencillo. Hemos comentado las líneas anteriores, que ya no hacen falta, y hemos puesto su equivalente usando la clase ObjectFactory que genera el plugin jaxb de maven.

# De XML a Java

Una vez que tenemos el XML, podemos volver a obtener la clase java con el siguiente código, dependiendo si la clase tiene la anotación @XmlRootElement o no.

## Con @XmlRootElement

Al igual que antes, si la clase tiene la anotación *@XmlRootElement*, el código es más sencillo

```
      String xml = "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>"
            + "<La_Clase>"
            + "<El_Atributo>ey you</El_Atributo>"
            + "</La_Clase>";

      JAXBContext jc = JAXBContext.newInstance(UnaClase.class);
      Unmarshaller ums = jc.createUnmarshaller();

      UnaClase theObject = (UnaClase) (ums.unmarshal(new StringReader(xml)));

      System.out.println(theObject.getUnAtributo());
```

Metemos el *XML* en un String. Se crea el contexto de *JAXB* y el *unmarshaller*, para convertir de *XML* a java.

Y listo, basta llamar al método *unmarshall()*, pasando un *Reader*, para obtener directamente la instancia de la clase ya rellena. Debemos hacer el "cast" a *UnaClase*, puesto que la obtenemos como *Object*.

Sólo nos queda trabajar con ella, en nuestro ejemplo, sacando por pantalla el atributo de la clase para ver que está relleno.

## Sin @XmlRootElement

Si la clase no lleva la anotación *@XmlRootElement*, el asunto es un poco más complejo. Al *Unmarshaller* debemos decirle que nos de un *JAXBElement* indicándole qué clase queremos. Para ello usamos un método *unmarshall()* que admite dos paraémtros, el *XMLStreamReader* y la clase que queremos obtener, como se ve en el siguiente código.

```
      // Un String con el XML que queremos convertir a java. Podría estar en un fichero en vez de en un String
      // o cualquier otra fuente.
      String xml = "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\n"
            + "<ns2:purchaseOrder xmlns=\"http://tempuri.org/po.xsd\" xmlns:ns2=\"info.source4code.jaxb.model\">"
            + "<shipTo>"
            + "<city>New York</city>"
            + "</shipTo>"
            + "<comment>This is a comment</comment>" + "</ns2:purchaseOrder>";

      // Creacion del contexto JAXB y el unmarshaller, que convierte de XML a Java.
      JAXBContext jc = JAXBContext.newInstance(PurchaseOrderType.class);
      Unmarshaller unmarshaller = jc.createUnmarshaller();

      // Se crea un XMLStreamReader a partir del String.
      XMLStreamReader reader = XMLInputFactory.newFactory()
            .createXMLStreamReader(new StringReader(xml));

      // Se convierte el XML a un JAXBElement. root.getValue() nos devuelve
```

```
    // la instancia de PurchaseOrderType ya rellena.
    JAXBElement<PurchaseOrderType> root = unmarshaller.unmarshal(reader,
          PurchaseOrderType.class);

    // Escribmos por pantalla un par de valores para ver que están correctamente
    // rellenos.
    System.out.println("City = " + root.getValue().getShipTo().getCity());
    System.out.println("Comment = " + root.getValue().getComment());
```

El código no es complejo. Primero tenemos el *XML*, en este caso lo hemos metido en un *String* para simplificar. Se crea el contexto *JAXB* para la clase *PurchaseOrderType* y el *unmarshaller*, que es la clase encargada de convertir de *XML* a *Java*.

Necesitamos un *XMLStreamReader* para alimentar al *unmarshaller*. La forma de obtenerlo a partir de un *String* puede ser como la línea que se muestra en el código. Seria algo similar si en vez de *String* tuviéramos el *XML* en un fichero.

Ya sólo queda llamar a *unmarshall()*. Se le pasa el *stream reader* y la clase que queremos obtener, nos la devolverá dentro de un *JAXBElement*. El método *getValue()* de ese *JAXBElement* nos dará directamente una instancia de la clase *PurchaseOrderType* ya rellena.

Solo nos queda trabajar con ella, en el ejemplo, sacando por pantalla un par de sus valores.



Categoría: Java

---

- Esta página fue modificada por última vez el 12 oct 2015 a las 15:03.