转 用于Java和XML绑定的JAXB教程

Java为处理XML结构和文件提供了一系列的选项。JAXB是其中最常见使用的一种。JAXB全称是Java Architecture for XML Binding。它能够将Java对象装换为XML结构，反过来亦然。JAXB是在JRE 1.6的第一个版本中随着JRE标准包一起发布的。

JAXB的第一个规范在2003年3月完成，实现过程由 Java Specification Request 31: https://jcp.org/en/jsr/detail?id=31所跟踪。在这个规范请求中你可发现很多关于JAXB的成长和所有的改进。

就像刚才提到的，JAXB实在1.6中加入JRE包的。在此之前，要使用它必须将它的包加入到工程中。

在JAXB可以使用之前（很久之前），Java处理XML的方式是DOM: http://www.w3.org/DOM/。这种方式并不是最好的，因为没有很好的抽象来映射XML节点，所有的值类型都被认为是字符串。JAXB有许多的优点，比如面向对象方式的操作XML节点和属性，值类型，注解和本文中将要阐述的其他优点。

本文中的所有例子程序都是使用以下版本来实现的: JRE 1.8.0 for 32b。所使用的IDE是Eclipse SDK 版本: Luna (4.4)。但是其他所有包含JAXB API的Java版本和IDE也能够完美的工作，只要是基于Java 8的。

1. 映射（Mapping）

使用一些注解和以下的特定的规则，Java对象就能和XML结构绑定起来。这就是我们称之为映射的原因。在本教程中，我们将使用例子、资源和其他的一些信息来解释以下几点。

我们将展示一些关于如何将Java对象转换为XML结构的例子程序，这种转换被称为**编组**（marshaling）。我们将展示如何使用适配器（adapter）处理原始类型，集合和更加复杂的类型。

我们还会解释如何进行相反的操作，也就是**反编组**（un-marshaling），例如将XML文件转换为Java对象。

所有这些都是通过使用Java注解来完成的。我们会列举并解释和JAXB一起使用的最重要的注解。

我们还会介绍用来校验的XSD（XML Schema），这也是JAXB所支持的一个重要的工具。我们也会看一下XSD在编组时如何起作用。

最后，我们会列举一些与JAXB一起配合使用的工具，它们能在很多的方面都对程序员起到帮助。

2. 编组（Marshal）

在本章中，我们将会看看如何将Java对象转换为XML文件，而且在这个过程中要考虑些什么。这通常被称为编组（marshaling）。首先，我们为JAXB指出哪些Java元素与哪些XML节点在业务模型中是对应的。

```
@XmlType( propOrder = { "name", "capital", "foundation", "continent" , "population"} )
@XmlRootElement( name = "Country" )
public class Country
{

    @XmlElement (name = "Country_Population")
    public void setPopulation( int population )
    {
        this.population = population;
    }
```

```
    @XmlElement( name = "Country_Name" )
    public void setName( String name )
    {
        this.name = name;
    }

    @XmlElement( name = "Country_Capital" )
    public void setCapital( String capital )
    {
        this.capital = capital;
    }
    @XmlAttribute( name = "importance", required = true )
    public void setImportance( int importance )
    {
        this.importance = importance;
    }
...
```

上面的类包含了一些JAXB的注解，它们用来表示我们将要生成哪些XML节点。因此，我们使用注解

@XmlRootElement 作为根元素。

@XmlElement 来与setter方法一起使用。

@XmlAttribute 来传递属性到XML节点。这些属性可以用一些property描述，比如是否必须（required ）。

@XmlType 来表示一些特殊的选项，比如在XML中进行排序。

我们会在后面的章节中详细来讨论这些注解和其他的注解。现在，我只是提及它们。下一步就是从Java对象生成XML文件。因此，我们使用JAXBContext 和它的编组功能创建一个简单的程序：

```
Country spain = new Country();
spain.setName( "Spain" );
spain.setCapital( "Madrid" );
spain.setContinent( "Europe" );
spain.setImportance( 1 );
spain.setFoundation( LocalDate.of( 1469, 10, 19 ) );
spain.setPopulation( 45000000 );

/* 初始化jaxb编组器（marshaler） */
JAXBContext jaxbContext = JAXBContext.newInstance( Country.class );
Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

/* 设置flag为true，表示将格式化输出 */
jaxbMarshaller.setProperty( Marshaller.JAXB_FORMATTED_OUTPUT, true );

/* 编组java对象到xml（输出到文件和标准输出）*/
jaxbMarshaller.marshal( spain, new File( "country.xml" ) );
jaxbMarshaller.marshal( spain, System.out );
```

基本上最重要的部分就是类javax.xml.bind.JAXBContext的使用。此类提供了一个框架来校验、编组和反编组XML到（从）Java对象，它是JAXB API的入口。更多关于此类的信息可以在此找到: https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/JAXBContext.html。在我们的小例子中，我们仅仅使用此类来创建JAXB的上下文以用于编组由参数传递进的对象。这是由下面的代码完成的:

```
JAXBContext jaxbContext = JAXBContext.newInstance( Country.class );
Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
```

这个程序的输出结果会是:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Country importance="1">
 <Country_Name>Spain</Country_Name>
 <Country_Capital>Madrid</Country_Capital>
 <Country_Foundation_Date></Country_Foundation_Date>
 <Country_Continent>Europe</Country_Continent>
 <Country_Population>45000000</Country_Population>
</Country>
```

在上面的JAXB程序中，我们仅仅将容器类中的简单类型（String和Integer）转换为XML节点。我们可以看到基于日期的属性，比如 foundation 丢失了，我们会稍后解释如何解决这个问题。看上去很简单吧，JAXB支持各种类型的Java对象，比如其他元素类型，集合，日期范围等等。如果我们要映射一组列表到XML，我们可以写:

```
@XmlRootElement( name = "Countries" )
public class Countries
{
    List countries;

    /**
     * 将会被编组到xml的元素
     */
    @XmlElement( name = "Country" )
    public void setCountries( List countries )
    {
        this.countries = countries;
    }
```

从这段代码中我们可以看到需要一个新的容器类来告诉JAXB一个包含list的类存在。一个类似于上面的程序的输出是:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Countries>
        <Country importance="1">
                <Country_Name>Spain</Country_Name>
                <Country_Capital>Madrid</Country_Capital>
                <Country_Foundation_Date></Country_Foundation_Date>
                <Country_Continent>Europe</Country_Continent>
                <Country_Population>0</Country_Population>
        </Country>
        <Country importance="0">
                <Country_Name>USA</Country_Name>
                <Country_Capital>Washington</Country_Capital>
                <Country_Foundation_Date></Country_Foundation_Date>
                <Country_Continent>America</Country_Continent>
                <Country_Population>0</Country_Population>
        </Country>
</Countries>
```

在处理集合时有许多的选项可以使用:

我们可以使用包装注解（wrapper annotations）：注解javax.xml.bind.annotation.XMLElementWrapper 能够创建XML结构的包装。这个包装可以包含元素的列表。

我们可以使用基于集合的注解比如javax.xml.bind.annotation.XMLElements 或 javax.xml.bind.annotation.XMLRefs ，它们提供了集合的功能，但是灵活性有所降低。

我们可以为集合提供一个容器。一个容器类 (本例中是Countries) 有一个类型java.util.Collection (本例中Country)的成员。这是我的最喜欢的方式，因为提供了更多的灵活性。这就是我们所演示的方式。

3. 反编组（Un-marshal）

在本章中，我们将会看到如何进行反向操作: 反编组XML文件到java对象以及在此操作中需要考虑的事情。首先，我们创建用来反编组的XML结构:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Countries>
        <Country>
                <Country_Name>Spain</Country_Name>
                <Country_Capital>Madrid</Country_Capital>
                <Country_Continent>Europe</Country_Continent>
        </Country>
        <Country>
                <Country_Name>USA</Country_Name>
                <Country_Capital>Washington</Country_Capital>
                <Country_Continent>America</Country_Continent>
        </Country>
```

```
        <Country>
                <Country_Name>Japan</Country_Name>
                <Country_Capital>Tokyo</Country_Capital>
                <Country_Continent>Asia</Country_Continent>
        </Country>
</Countries>
```

值得一提的是我们删除了foundation 日期。如果它们存在，我们会得到一个error，因为JAXB不知道如何去反编组它们。我们稍后会看到如何解决这一问题。在此之后，我们创建一个程序用来读这个XML文件并把它解析到合适的java对象:

```
File file = new File( "countries.xml" );
JAXBContext jaxbContext = JAXBContext.newInstance( Countries.class );

/**
* 和编组的唯一区别的操作
*/
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
Countries countres = (Countries)jaxbUnmarshaller.unmarshal( file );
System.out.println( countres );
```

我们可以发现上面的代码和上一章中所讲的编组操作没有太大的区别。我们也使用类javax.xml.bind.JAXBContext ，但是这里用的方法是createUnmarshaller()，它会返回一个类型为javax.xml.bind.Unmarshaller的对象。这个对象在之后负责反编组XML 。本程序使用被注解的类 Country如下:

```
@XmlType( propOrder = { "name", "capital", "foundation", "continent" , "population"} )
@XmlRootElement( name = "Country" )
public class Country
{

    @XmlElement (name = "Country_Population")
    public void setPopulation( int population )
    {
        this.population = population;
    }


    @XmlElement( name = "Country_Name" )
    public void setName( String name )
    {
        this.name = name;
    }

    @XmlElement( name = "Country_Capital" )
    public void setCapital( String capital )
    {
        this.capital = capital;
    }
    @XmlAttribute( name = "importance", required = true )
    public void setImportance( int importance )
    {
        this.importance = importance;
    }
...
```

我们会发现和上一章中使用的类没有什么区别，使用了相同的注解。运行这段程序会在标准输出中产生如下输出:

```
Name: Spain
Capital: Madrid
Europe
Name: USA
Capital: Washington
America
Name: Japan
Capital: Tokyo
Asia
```

在编组操作中所阐述的那些事项也同样适用于此。可以反编组对象和其他原始数据类型比如数字类型，也可

以反编组基于集合的元素比如list或set。我们可以看到上述以及上一章中所提供过得例子中，类型java.time.LocalDate的属性没有被转换。这是由于JAXB不知道如何去处理它。可以使用适配器来处理复杂类型，我们将会在下一章中看到。

4. 适配器（Adapters）

当遇到JAXB不能直接处理的复杂类型时，我们需要写一个适配器来告诉JAXB如何处理。我们将会用一个编组（和反编组）java.time.LocalDate类型的元素的例子来解释它.

```
public class DateAdapter extends XmlAdapter
{

    public LocalDate unmarshal( String date ) throws Exception
    {
        return LocalDate.parse( date );
    }

    public String marshal( LocalDate date ) throws Exception
    {
        return date.toString();
    }

}
```

上述代码使用适当的类型和结构来演示了接口javax.xml.bind.annotation.adapters.XmlAdapter的编组和反编组方法的实现，然后，我们告诉JAXB在何处使用注解 @XmlJavaTypeAdapter来使用它:

```
 ...
@XmlElement( name = "Country_Foundation_Date" )
@XmlJavaTypeAdapter( DateAdapter.class )
public void setFoundation( LocalDate foundation )
{
    this.foundation = foundation;
}
...
```

本教程的第一个程序的输出XML将会如下:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Country importance="1">
        <Country_Name>Spain</Country_Name>
        <Country_Capital>Madrid</Country_Capital>
        <Country_Foundation_Date>1469-10-19</Country_Foundation_Date>
        <Country_Continent>Europe</Country_Continent>
        <Country_Population>45000000</Country_Population>
</Country>
```

这可以适用于所有JAXB不直接支持，但是我们希望在XML中包含的复杂类型。我们仅需要实现接口javax.xml.bind.annotation.adapters.XmlAdapter 并实现他们的方法javax.xml.bind.annotation.adapters.XmlAdapter.unmarshal(ValueType) 和javax.xml.bind.annotation.adapters.XmlAdapter.marshal(BoundType)。

5. XSDs

XSD是XML schema。它包含关于应该被遵守的XML文件和结构的规则和限制的信息。这些规则适用于XML的结构也适用于内容。规则能由各种结构组合，从而创建非常复杂的规则，本章中我们将会展示如何使用XSD来校验和编组。下面就是一个能够用于本教程中的Country 类的XML schema:

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

```
                <xs:sequence>
                        <xs:element name="Country_Name" type="xs:string" />
                        <xs:element name="Country_Capital" type="xs:string" />
                        <xs:element name="Country_Foundation_Date" type="xs:string" />
                        <xs:element name="Country_Continent" type="xs:string" />
                        <xs:element name="Country_Population" type="xs:integer" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
</xs:schema>
```

5.1. 使用XSD来校验

XSD能够用来做XML校验。JAXB使用XSD来校验XML，从而来确保对象和XML结构符合预期的规则。要校验一个对象符合XSD，我们需要先创建XSD，如下所示:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:simpleType name="continentType">
                <xs:restriction base="xs:string">
                        <xs:pattern value="Asia|Europe|America|Africa|Oceania"/>
                </xs:restriction>
        </xs:simpleType>

<xs:element name="Country">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="Country_Name" type="xs:string" minOccurs='1' />
                        <xs:element name="Country_Capital" type="xs:string" minOccurs='1' />
                        <xs:element name="Country_Foundation_Date" type="xs:string" minOccurs='1' />
                        <xs:element name="Country_Continent" type="continentType" minOccurs='1' />
                        <xs:element name="Country_Population" type="xs:integer" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
</xs:schema>
```

我们能在程序中使用这个schema来告诉JAXB我们想使用什么XSD。通过创建一个类javax.xml.validation.Schema的实例，我们就能像下面这样校验:

```
/**
* 创建schema
*/
SchemaFactory sf = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI );
Schema schema = sf.newSchema( new File( "countries_validation.xsd" ) );
```

校验过程需要一个错误处理器来处理所有的错误。需要实现接口org.xml.sax.ErrorHandler 和它的错误方法:

public class MyErrorHandler implements ErrorHandler {

@Override

public void warning(SAXParseException exception) throws SAXException {

throw exception;

} ...

然后，就可以校验类javax.xml.bind.util.JAXBSource的实例:

```
/**
 * 创建context，用来创建每一个country的source
 */
JAXBContext jaxbContext = JAXBContext.newInstance( Country.class );
JAXBSource sourceSpain = new JAXBSource( jaxbContext, spain );
JAXBSource sourceAustralia = new JAXBSource( jaxbContext, australia );
```

下面的是完整的程序:

```
/**
     * 抛出error，因为continent是必需的
     */
    Country spain = new Country();
    spain.setName( "Spain" );
    spain.setCapital( "Madrid" );
    spain.setFoundation( LocalDate.of( 1469, 10, 19 ) );
    spain.setImportance( 1 );


    /**
     * ok
     */
    Country australia = new Country();
    australia.setName( "Australia" );
    australia.setCapital( "Camberra" );
    australia.setFoundation( LocalDate.of( 1788, 01, 26 ) );
    australia.setContinent( "Oceania" );
    australia.setImportance( 1 );


    /**
     * 创建schema
     */
    SchemaFactory sf = SchemaFactory.newInstance( XMLConstants.W3C_XML_SCHEMA_NS_URI );
    Schema schema = sf.newSchema( new File( "countries_validation.xsd" ) );

    /**
     * 创建context，用来创建每一个country的source
     */
    JAXBContext jaxbContext = JAXBContext.newInstance( Country.class );
    JAXBSource sourceSpain = new JAXBSource( jaxbContext, spain );
    JAXBSource sourceAustralia = new JAXBSource( jaxbContext, australia );

    /**
     * 初始化校验
     */
    Validator validator = schema.newValidator();
    validator.setErrorHandler( new MyErrorHandler() );

    //使用校验器
    try
    {
        validator.validate( sourceSpain );
        System.out.println( "spain has no problems" );
    }
    catch( SAXException ex )
    {
        ex.printStackTrace();
        System.out.println( "spain has problems" );
    }
    try
    {
        validator.validate( sourceAustralia );
        System.out.println( "australia has no problems" );
    }
    catch( SAXException ex )
    {
        ex.printStackTrace();
        System.out.println( "australia has problems" );
    }
```

输出是:

```
org.xml.sax.SAXParseException
        at javax.xml.bind.util.JAXBSource$1.parse(JAXBSource.java:248)
        at javax.xml.bind.util.JAXBSource$1.parse(JAXBSource.java:232)
        at com.sun.org.apache.xerces.internal.jaxp.validation.ValidatorHandlerImpl.validate(
...
spain has problems
```

```
australia has no problems
```

我们能看到Australia 没有问题，但是Spain ...

5.2. 使用XSD编组

XSD也可以用于从XML文件绑定和生成java类，反之亦然。我们将会看到一个编组的例子来看看如何使用它。和前面使用相同的XML schema，我们将会写一个程序来使用给定的XSD初始化一个javax.xml.validation.Schema, 和一个用于编组（Country）的类的javax.xml.bind.JAXBContext 。本程序会使用一个javax.xml.bind.Marshaller来处理需要的操作:

```
 /**
 * 由于continent是必需的，校验会失败
 */
Country spain = new Country();
spain.setName( "Spain" );
spain.setCapital( "Madrid" );
spain.setFoundation( LocalDate.of( 1469, 10, 19 ) );

SchemaFactory sf = SchemaFactory.newInstance( XMLConstants.W3C_XML_SCHEMA_NS_URI );
Schema schema = sf.newSchema( new File( "countries_validation.xsd" ) );

JAXBContext jaxbContext = JAXBContext.newInstance( Country.class );

Marshaller marshaller = jaxbContext.createMarshaller();
marshaller.setProperty( Marshaller.JAXB_FORMATTED_OUTPUT, true );
marshaller.setSchema( schema );
//schema使用一个校验处理器来校验对象
marshaller.setEventHandler( new MyValidationEventHandler() );
try
{
    marshaller.marshal( spain, System.out );
}
catch( JAXBException ex )
{
    ex.printStackTrace();
}

/**
 * continent是错误的，校验会失败
 */
Country australia = new Country();
australia.setName( "Australia" );
australia.setCapital( "Camberra" );
australia.setFoundation( LocalDate.of( 1788, 01, 26 ) );
australia.setContinent( "Australia" );

try
{
    marshaller.marshal( australia, System.out );
}
catch( JAXBException ex )
{
    ex.printStackTrace();
}

/**
 * 最终一切都ok
 */
australia = new Country();
australia.setName( "Australia" );
australia.setCapital( "Camberra" );
australia.setFoundation( LocalDate.of( 1788, 01, 26 ) );
australia.setContinent( "Oceania" );


try
{
    marshaller.marshal( australia, System.out );
}
```

```
catch( JAXBException ex )
{
    ex.printStackTrace();
}
```

我们刚刚讨论了在编组之前的关于XSD的校验。在编组对象到XML时也是可以用来校验的。如果你的对象不符合一些特定的规则，也会得到一些校验错误，即使我们没有显示的校验。这种情况下，我们没有使用实现了org.xml.sax.ErrorHandler的错误处理器，而是一个javax.xml.bind.ValidationEventHandler:

```
public class MyValidationEventHandler implements ValidationEventHandler
{

    @Override
    public boolean handleEvent( ValidationEvent event )
    {
        System.out.println( "Error catched!!" );
        System.out.println( "event.getSeverity():  " + event.getSeverity() );
        System.out.println( "event:  " + event.getMessage() );
        System.out.println( "event.getLinkedException():  " + event.getLinkedException() );
        //locator包含了很多的信息，比如line，column等
        System.out.println( "event.getLocator().getObject():  " + event.getLocator().getObject() );
        return false;
    }

}
```

我们可以看到方法 javax.xml.bind.ValidationEventHandler.handleEvent(ValidationEvent) 已经被实现。而输出将会是:

```
javax.xml.bind.MarshalException
 - with linked exception:
[org.xml.sax.SAXParseException; lineNumber: 0; columnNumber: 0; cvc-pattern-valid: Value 'Australia' i
        at com.sun.xml.internal.bind.v2.runtime.MarshallerImpl.write(MarshallerImpl.java:311)
        at ...
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Country>
<Country_Name>Australia</Country_Name>
<Country_Capital>Camberra</Country_Capital>
<Country_Foundation_Date>1788-01-26</Country_Foundation_Date>
<Country_Continent>Oceania</Country_Continent>
<Country_Population>0</Country_Population>
</Country>
```

很显然，关于XSD有很多东西需要解释，因为所有的规则组合起来所适用的范围会很广。但是这超出了本教程的范畴。如果你需要关于如何在JAXB中使用XML Schema的更多信息，你可以访问以下非常有趣的链接:

http://www.w3schools.com/schema/schema_example.asp.
http://blog.bdoughan.com/2010/12/jaxb-and-marshalunmarshal-schema.html .

6. Annotations

在本教程中我们使用了若干JAXB中的注解来编组和反编组XML。我们在下面列出了最重要的一些注解:

XmlAccessorOrder: 本注解控制类里的字段和属性在XML中出现的顺序。更多信息请看:
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlAccessorOrder.html

XmlAccessorType: 表示一个元素是否可以被序列化。它可以和 javax.xml.bind.annotation .XMLAccessType一起使用。 更多信息请看:
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlAccessorType.html

XmlAnyAttribute: 映射一个元素到通配符（wildcard ）属性的Map。更多信息请看
http://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlAnyAttribute.html .

XmlAnyElement: 在没有映射被预定义时，作为反编组操作的一个缺省。 更多信息请看

https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlAnyElement.html

XmlAttribute: 本注解是最基础和最常使用的一个。它映射一个Java元素(property, attribute, field) 到一个XML节点属性. 本教程中多个例子中用到了它。 更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlAttribute.html

XmlElement: 使用name映射一个Java元素到XML节点。 更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlElement.html

XmlElementRef: 使用type (不同于上者, name被用来做映射)映射一个Java元素到一个XML节点 . 更多信息请看 https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlElementRef.html

XmlElementRefs: 标记一个指向XmlElement 和JAXBElement所注解的类。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlElementRefs.html

XmlElements: 这是一个包含多个XMLElement 注解的容器。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlElements.html

XmlElementWrapper: 它生成一个围绕XML结构的包装器，旨在和集合一起使用，本教程中我们看到了有不同的方式来处理集合。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlElementWrapper.html

XmlEnum: 提供emum到XML的映射。它和XmlEnumValue一起工作。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlEnum.html

XmlEnumValue:映射一个enum常量到一个XML元素。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlEnumValue.html

XmlID: 映射一个属性到XML id。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlID.html

XmlList: 另一种在JAXB中处理list的方式。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlList.html

XmlMimeType: 控制被注解的属性的表现形式。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlMimeType.html

XmlMixed: 被注解的元素包含混合的内容。内容可以是文本或未知的（unknown）。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlMixed.html

XmlRootElement: 这可能是JAXB中使用最多的注解了。它用于映射一个类到XML元素。它基本上是每一个JAXB的入口点。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlRootElement.html

XmlSchema: 映射一个package到XML命名空间。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlSchema.html

XmlSchemaType: 映射一个Java类型到一个内置的simple schema。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlSchemaType.html

XmlSeeAlso: 告诉JAXB在绑定被注解类时，去绑定其他类。这是必须的，因为Java很难列出一个类的所有子类，使用这种机制，你可以告诉JAXB在处理一个特定类的时候哪一个子类（或其他类）应该被绑定。更多信息请看 https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlSeeAlso.html

XmlType: 用于map一个类或enum到XML Schema中的一个type。更多信息请看
https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlType.html

XmlValue: 允许map一个类到一个包含simpleContent 的XML Schema复杂类型或一个XML Schema的简单类型。更多信息请看https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlValue.html

这是一个很长的列表，但是并不是所有的JAXB的注解。要查看JAXB的所有注解的列表，请查看package 的summaryhttps://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/package-frame.html.

7. 工具

有许多工具可以帮助程序员使用JAXB和XML Schema。我们将会在本章中列出部分并提供一些有用的资源:

schemagen: 表示 Schema Generator. 它用于从Java注解类或源代码生成JAXB Schema（它也可以和字节码一起工作）。能直接从命令行或使用Ant来使用。更多信息请访问Oracle的页面 https://docs.oracle.com/javase/7/docs/technotes/tools/share/schemagen.html 或 https://jaxb.java.net/2.2.4/docs/schemagen.html. 它随着JRE标准包一起发布，能从bin目录启动。

xjc: 这是JAXB的绑定编译器。它用于从XML Schema（XSD）生成Java源代码（类）。它能和Ant一起使用。在执行之前校验XSD。更多关于用法和参数的信息请访问https://jaxb.java.net/2.2.4/docs/xjc.html .

Jsonix: 没有与JAXB的直接联系，但是一个用于XML和JSON之间的互相转换的工具。在使用Javascript时非常有用。这是官方的链接http://confluence.highsource.org/display/JSNX/Jsonix .

Hyperjaxb3: 提供JAXB对象的关系持久化。它使用JAXB的注解来持久化关系数据库中的对象。可以和Java中使用最多的持久化框架Hibernate一起工作。你可以在下面的链接找到更多文档和源码 http://confluence.highsource.org/display/HJ3/Home.

Maven JAXB2 Plugin: Maven插件，提供在转换XML Schema到Java类时xjc工具的所有功能和其他的一些功能。从以下链接可以下载源码和参考文档 https://github.com/highsource/maven-jaxb2-plugin .

JAXB Eclipse Plugin: 有许多的Eclipse插件可用 (对其他 IDE比如 NetBeans 或IntelliJ也一样)，可用来编译XML Schema到Java类，校验XML Schema或生产Schema。它们都包装并使用上面所列的工具。因为没有任何插件可以被视为标准，我建议你在网上搜索并选择你认为合适的。

这些并不是关于JAXB和XSD并进行XML结构和Java类的转换的所有的工具。我仅仅列出了每一个程序员在使用JAXB时应该考虑的重要而且基本的一些工具

8. 最佳实践

虽然这并不是最佳实践的完整列表，但我还是主观的提供一些关于如何使用JAXB的一些最佳用法。

尽量避免有问题的原始类型比如 float, decimal 或负整数。它们在JAXB中没有对应的类型。在这种情况下，应该使用其他的“没有问题”的原始类型。

使用注解@XMLSchemaType以便关于必须使用的类型更加清晰，而不会让JAXB来做决定。

避免匿名类型和混合内容。

J AXZB使用系统默认的locale和country来生成消息和错误消息。要改变他们，你可以传入以下JVM参数到你的应用中:

9. 总结

本教程中我们解释了什么是JAXB和它能做什么。JAXB表示Java Architecture for XML binding，是一个用于以下方面的框架:

编组Java对象到XML结构.

反编组XML文件到Java类.

这些操作可以通过使用XSD来完成 (XML schema).

XSD也可以用作校验.

坦白的说，JAXB用于XML和Java类之间的相互转换。我们解释了关于它的所有主要概念，并列出了JAXB中的主要的注解和类。还有许多的工具可以用于辅助JAXB完成不同的工作，我们解释了其中一些的用法和作用

10. 资源

本教程包含了一些关于JAXB以及它作用于XML和Java对象之间的转换的深入的信息。它也解释了在使用JAXB时如何使用XSD，以及一些最佳实践和窍门。但是当你需要在使用JAXB更进一步时，访问以下链接可能会有

帮助:

http://examples.javacodegeeks.com/core-java/xml/bind/jaxb-unmarshal-example/
http://examples.javacodegeeks.com/core-java/xml/bind/jaxb-unmarshal-example/
http://www.oracle.com/technetwork/articles/javase/index-140168.html
http://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding
https://jaxb.java.net/
http://en.wikipedia.org/wiki/XML_schema

1
-Duser.country=US -Duser.language=en -Duser.variant=Traditional_WIN

原文地址：https://www.javacodegeeks.com/2015/04/%E7%94%A8%E4...

分类：工作日志 字数：5638

longxa

程序员 武汉

+ 关注

粉丝 1 | 博文 3 | 码字总数 872

Ctrl+Enter 发表评论