**Лабораторная работа № 6**

**«Технология XML связывания данных с помощью JAXB»**

**Цель работы:** изучить технологию XML связывания данных с помощью API связывания – JAXB.

**Задание на лабораторную работу**

Создать приложение, которое читает данные о студентах группы из XML,

изменяет их и производит вывод измененной информации в XML.

**Краткие теоретические сведения**

***Основы XML***

Расширяемый язык разметки (Extensible Markup Language, аббревиатура -XML) описывает класс объектов XML document, а также частично описывает работу компьютерных программ, обрабатывающих объекты с данными,реализующими этот класс. XML - это прикладной уровень или усеченная формаSGML, Стандартного обобщенного языка разметки. По своему построению,XML документ является полноценным SGML документом.

XML документы состоят из единиц размещения, называемыхсущностями, которые содержат разобранные или неразобранные данные.Разобранные данные состоят из набора символов, часть которых образуютсимвольные данные, часть–разметку. Разметка образует описание схемыразмещения и логической структуры документа. Язык XML дает механизмсоздания ограничений для указанной схемы размещения и логическойструктуры.

Есть три вида XML-документов:

1. **Неправильные** документы не следуют синтаксическим правилам, определенным спецификацией XML. Если разработчик определил правила для документа, которые могут содержаться в DTD или в схеме, и документ не следует этим правилам, такой документ также является неправильным
2. **Правильные** документы следуют синтаксическим правилам XML и правилам, определенным в их DTD или в схеме
3. **Правильно-форматированные** документы следуют синтаксическим правилам XML, но не имеют DTD или в схемы

В общем случае XML- документы должны удовлетворять следующим требованиям:

1. В заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация
2. Каждый открывающий тэг, определяющий некоторую область данных в документе обязательно должен иметь своего закрывающего "напарника", т.е., в отличие от HTML, нельзя опускать закрывающие тэги
3. В XML учитывается регистр символов
4. Все значения атрибутов, используемых в определении тэгов, должны быть заключены в кавычки
5. Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов
6. Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные, и поэтому учитываются все символы форматирования (т.е. пробелы, переводы строк, табуляции не игнорируются)

Если XML-документ не нарушает приведенные правила, то он называется

формально-правильным и все анализаторы, предназначенные для разбора XML-

документов, смогут работать с ним корректно.

Однако кроме проверки на формальное соответствие грамматике языка, в документе могут присутствовать средства контроля над содержанием документа, за соблюдением правил, определяющих необходимые соотношений между элементами и формирующих структуру документа.

Для того, чтобы обеспечить проверку корректности XML-документов,

необходимо использовать анализаторы, производящие такую проверку и называемые верифицирующими.

На сегодняшний день существует два способа контроля правильности XML- документа: DTD - определения(Document Type Definition) и схемы данных(Semantic Schema). В отличии от SGML, определение DTD-правил в XML не является необходимостью, и это обстоятельство позволяет нам создавать любые XML-документы, не ломая пока голову над весьма непростым синтаксисом DTD и XSD.

***Структура языка***

Содержимое XML - документа представляет собой набор элементов,секций CDATA, директив анализатора, комментариев, спецсимволов,текстовых данных. Рассмотрим каждый из них подробней.

Элемент – это структурная единица XML- документа. Заключая слово rose в тэги <flower> </flower>, определяется непустой элемент, называемый <flower>, содержимым которого является rose. В общем случае в качестве содержимого элементов могут выступать как просто какой-то текст, так и другие, вложенные, элементы документа, секции CDATA, инструкции по обработке, комментарии, т.е. практически любые части XML- документа.

Любой непустой элемент должен состоять из начального, конечного тэгов и данных, между ними заключенных. Например, следующие фрагменты будут являться элементами:

<flower>rose</flower>

<city>Chernigov</city>

Набором всех элементов, содержащихся в документе, задается его структура и определяются все иерархическое соотношения. Плоская модель данных превращается с использованием элементов в сложную иерархическую систему со множеством возможных связей между элементами.

<country id="Ukraine">

<cities-list>

<city>

<title>Kiev</title>

<firm id="1">

<title>Ликон</title>

<url>www.licon.com.ua</url>

</firm>

<firm id="2">

<title>Транссервис</title>

<url/>

</firm>

</city>

</cities-list>

</country>

Производя в последствии поиск в этом документе, программа клиента будет опираться на информацию, заложенную в его структуру – используя элементы документа. Т.е. если, например, требуется найти нужную фирму в нужном городе, используя приведенный фрагмент документа, то необходимо будет просмотреть содержимое конкретного элемента <firm>, находящегося внутри конкретного элемента <city>. Поиск при этом, естественно, будет гораздо более эффективен, чем нахождение нужной последовательности по всему документу.

В XML документе, как правило, определяется хотя бы один элемент,называемый корневым и с него программы-анализаторы начинают просмотр документа. В приведенном примере этим элементом является <country>.

В случае, если элемент не имеет содержимого, т.е. нет данных, которые он должен определять, он называется пустым. Примером пустых элементов в HTML могут служить такие тэги HTML, как <br>, <hr>, <img>;. Необходимо

только помнить, что начальный и конечные тэги пустого элемента как бы объединяется в один, и надо обязательно ставить косую черту перед закрывающей угловой скобкой (например, <empty/>;).

Комментариями является любая область данных, заключенная между последовательностями символов <!-- и --> Комментарии пропускаются анализатором и поэтому при разборе структуры документа в качестве значащей информации не рассматриваются.

Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то имеется возможность использовать атрибуты элемента. Атрибут - это пара "название" = "значение", которую надо задавать при определении элемента в начальном тэге. Пример:

<color **RGB**="true">#ff08ff</color>

<color **RGB**="false">white</color>

Для того, чтобы включить в документ символ, используемый для определения каких-либо конструкций языка (например, символ угловой скобки) и не вызвать при этом ошибок в процессе разбора такого документа, нужно использовать его специальный символьный либо числовой идентификатор. Например, &lt; , &gt; &quot; или &#036;(десятичная форма записи), &#x1a (шестнадцатеричная) и т.д. Строковые обозначения спецсиволов могут определяться в XML документе при помощи компонентов (entity).

Инструкции, предназначенные для анализаторов языка, описываются в XML документе при помощи специальных тэгов – <? и ?>;. Программа клиента использует эти инструкции для управления процессом разбора документа. Наиболее часто инструкции используются при определении типа документа (например, <? xml version=”1.0” ?>) или создании пространства имен.

Чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы, но, в отличии от комментариев, иметь возможность использовать их в приложении, необходимо использовать тэги <![CDATA] и ]]>. Внутри этого блока можно помещать любую информацию, которая может понадобится программе-клиенту для выполнения каких-либо действий (в область CDATA, можно помещать, например, инструкции JavaScript). Естественно, надо следить за тем, чтобы в области, ограниченной этими тэгами не было последовательности символов ]].

***Определение DTD***

DTD позволяет задать основную структуру XML-документа.Есть два вида DTD: внешний и внутренний. Внешний файл DTD простосодержит элементы. Для внутреннего DTD эти определения являются частьюсамого XML-файла.

Внешний DTD может определять содержимое многих разных документов, делая их в чем-то более легкими для управления. Однако иногда правильный документ нуждается в том, чтобы подтверждать сам себя. В этом случае нужно включать информацию DTD в сам документ.

<?xml version="1.0"?>

**<!DOCTYPE memories [**

**<!ELEMENT memories (memory+) >**

**<!ELEMENT memory (#PCDATA) >**

**]>**

<memories>

<memory>TBD</memory>

<memory>TBD</memory>

</memories>

Независимо от того, имеем мы дело с внешними или внутренними DTD, элементы являются фундаментом XML-документов, так что они обычно определяются первыми. Все определения элементов состоят из ключевого слова ELEMENT, имени элемента и содержимого, которое может в нем быть. Элемент может содержать текст, другие элементы или ничего вообще (в случае пустого элемента).

<!DOCTYPE memories [

<ELEMENT memories (memory) >

<ELEMENT memory (subdate, donor) >

<ELEMENT subdate (#PCDATA) >

<ELEMENT donor EMPTY>

]>

Элемент, который содержит текст, определяется с ключевым словом #PCDATA. Это сокращение для разбираемых символьных данных (parsed character data); оно относится к любому тексту внутри элемента и не может содержать разметку. Примером являются элемент subdate. Элементы memory и memories показывают синтаксис, используемый для задания элементов, которые содержат в себе только другие элементы. Элемент может также быть определен как EMPTY, как элемент donor.

Иногда необходимо предоставить возможность выбора для содержимого элемента. Например, структура данных содержит элемент с именем location, который может содержать либо элемент place, либо элемент description. Это записывается следующим образом, с использованием символа вертикальной черты (|), разделяющего варианты:

**<!ELEMENT location (place | description) >**

<!ELEMENT place (#PCDATA) >

<!ELEMENT description (#PCDATA) >

DTD, которое приводились выше, является весьма специфичным. Каждый элемент должен появляться один раз. Для того чтобы элементы могли повторяться несколько раз существуют модификаторы. Модификаторы предлагают большую гибкость в проекте. Они таковы:

* \*: Элемент, определенный с модификатором \*, может появляться 0 или более раз
* +: Элемент, определенный с модификатором +, должен появляться 1 или более раз
* ?: Элемент, определенный с модификатором ?, должен появляться 0 или 1 раз.

Хотя и возможно создать структуру XML, которая не содержит ничего, кроме элементов, более общей ситуацией является ситуация элементов с атрибутами. Атрибуты должны также быть определены, если они появляются в элементах правильного документа.

Можно определить атрибуты несколькими способами. Первый – просто определить их как символьные данные или CDATA:

**<!ATTLIST memory tapeid CDATA #REQUIRED >**

В этом случае DTD назначает элементу memory атрибут tapeid. Атрибут tapeid состоит из символьных данных и является обязательным. Элемент может также быть определен как #IMPLIED или #FIXED, в этом случае должно быть также задано значение по умолчанию.

Некоторые атрибуты перечисляемые, это значит, что значение должно выбираться из предопределенного списка. Например:

<!ATTLIST media type (8mm | vhsc | vhs ) '8mm' #IMPLIED >

В этом случае документ должен выбрать значение из списка, если же значение не обеспечено, парсер использует значение по умолчанию 8mm. Это случай для любого документа, для которого существует DTD, даже, если парсер непроверяющий. В одном определении ATTLIST может быть определено много атрибутов:

<!ATTLIST media type (8mm |vhsc |vhs) '8mm' #IMPLIED length

CDATA >

***Определение XSD***

В отличие от DTD, документы схемы XML строятся на самом XML.Документ схемы XML - это просто XML-документ с предопределеннымиэлементами и атрибутами, описывающими структуру другого XML-документа.

XML-документы строятся из элементов. Определение элемента в документе схемы XML состоит в придании ему имени и типа.

<xsd:element name="subdate" type="xsd:date"/>

<xsd:element name="donor" type="xsd:string"/>

<xsd:element name="subdate" type="xsd:string"/>

<xsd:element name="place" type="xsd:string" />

Таблица 1– Предопределенные простые типы для значений

|  |  |
| --- | --- |
| **Простые типы** | **Пример/разъяснение** |
| string | "это строка" |
| boolean | true, false |
| float | плавающая точка одинарной точности |
| double | 64-битная плавающая точка двойной точности |
| decimal | 123.45 |
| integer | -12345; 0; 12345 |
| non-positive-integer | -12345; 0 |
| negative-integer | -12345 |
| long | 1234567890123 |
| int | 123456789 |
| short | 12345 |
| byte | 123 |
| non-negative-integer | 0; 123456 |
| unsigned-long | 123456789012345 |
| unsigned-int | 1234567801 |
| unsigned-short | 12345 |
| unsigned-byte | 123 |
| positive-integer | 123456 |
| date | 2000-02-16 |
| time | 15:55:00.000 |
| timeInstant | 2000-12-31T05:55:00.000 (5:55 am, Coordinated Universal Time) |
| timeDuration | P5Y1M4D10H20M1.2S (5 лет; 1 месяц; 4 дня; 20 минут; 1.2 секунды) |
| recurringInstant | --02-16T5:55:00 (16 февраля каждого года в 5:55 am. Coordinated Universal Time) |
| binary | битовое поле |
| uri-reference | стандартный URL |
| ID\* | Для совместимости с XML 1.0 |
| IDREF\* | Для совместимости с XML 1.0 |
| ENTITY\* | Для совместимости с XML 1.0 |
| NOTATION\* | Для совместимости с XML 1.0 |
| language | en-US (см. xml:lang в XML 1.0) |
| IDREFS\* | Для совместимости с XML 1.0 |
| ENTITIES\* | Для совместимости с XML 1.0 |
| NMTOKEN\* | Для совместимости с XML 1.0 |
| NMTOKENS\* | Для совместимости с XML 1.0 |
| Name | см. XML 1.0 |
| QName | (префикс ':')? NCName = LocalPart (напр..; po:Address) |
| NCName | (буква | '\_') (NCNameChar)\*  \*Использовать только в атрибутах для обеспечения совместимости с XML 1.0 DTD. |

Кроме встроенных простых типов, можно создавать новые простые типы. Спектр этих типов - от текста в определенном формате (такого, как телефонный номер или артикул товара) до числового ряда или перечисляемого списка. Для примера создадим тип, который ограничивает значение для целого числа между 1 и 100000.

<xsd:simpleType name="idNumber">

<xsd:restriction base="xsd:integer">

<xsd:minInclusive value="1" />

<xsd:maxInclusive value="100000" />

</xsd:restriction>

</xsd:simpleType>

Можно использовать другой тип ограничения, создавая список перечислений.

<xsd:simpleType name="mediaType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="8mm" />

<xsd:enumeration value="vhs" />

<xsd:enumeration value="vhsc" />

</xsd:restriction>

</xsd:simpleType>

Одно из ограничений для простого типа состоит в том, что такие элементы не могут содержать атрибутов. Чтобы добавить атрибуты в элемент, необходимо преобразовать его в complexType.

Одним из способов сделать это - является использование анонимных сложных типов. Он включает в себя добавление элемента complexType как потомка элемента element.

<xsd:element name="media">

**<xsd:complexType>**

**<xsd:attribute name="mediaid" type="xsd:integer" />**

**</xsd:complexType>**

</xsd:element>

Добавление в элемент потомков также требует использования сложных типов. Проще всего перечислить один или более дочерних элементов, используя элемент sequence:

<xsd:element name="memories">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="memory" type="memoryType"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

Элемент sequence показывает все возможные дочерние элементы для данного элемента. В некоторых случаях, однако, необходимо выбрать один элемент из списка альтернатив. Для этого вам нужен элемент choice:

<xsd:complexType name="locationType">

<xsd:choice>

<xsd:element name="description" type="xsd:string" />

<xsd:element name="place" type="xsd:string" />

</xsd:choice>

</xsd:complexType>

До сих пор все элементы и атрибуты, добавляемые в схему, должны были появляться ровно один раз. Очевидно, что это не всегда желательно.

Используя minOccurs и maxOccurs, можно управлять тем, должен ли компонент появляться обязательно и может ли он повторяться. В данном примере схема требует, чтобы элемент subject присутствовал, и позволяет ему появляться до пяти раз в одном элементе:

<xsd:element name="subject" minOccurs="1" maxOccurs="5" type="xsd:string"/>

Иногда не нужен верхний предел. Например, элемент memory может быть определен как необязательный, но если он есть, он может появляться неограниченное число раз в элементе memories:

<xsd:element name="memories">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="memory" **minOccurs="0"**

**maxOccurs="unbounded"** type="memoryType"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

Иногда может ограничиваться содержимое элемента, но не порядок, в котором оно появляется. Это, в частности, так для смешанного содержимого в схемах. Чтобы создать элемент, который не ограничивает порядок его потомков, используйте элемент all вместо sequence.

<xsd:element name="subject">

<xsd:complexType >

<xsd:all>

<xsd:element name="i" minOccurs="0" maxOccurs="1"

type="xsd:string" />

<xsd:element name="b" minOccurs="0" maxOccurs="1"

type="xsd:string" />

</xsd:all>

</xsd:complexType>

</xsd:element>

Необходимо заметить, что maxOccurs установлен в 1, а не в unbounded. Это обязательно для применения элемента all. Атрибуты minOccurs и maxOccurs должны быть либо 0, либо 1.

***Технология JAXB***

Технология JAXB (Java API for XML Binding) объединяет достоинства SAX (Simple API for XML) и DOM (Document Object Model) анализаторов, исключает их недостатки и дает дополнительные преимущества. На основе метаописания документа, представленного в виде XML схемы, JAXB позволяет сгенерировать следующие Java классы:

1. объектную модель (набор Java классов и связей между ними), в точности соответствующую структуре XML документа
2. код, позволяющий отображать текстовое представление XML документа в объектную модель и обратно
3. код для верификации XML документа представленного как объектная модель или как текст

JAXB позволяет при необходимости гибко настраивать генерируемый код. Например, можно определить правила отображения XML схемы на генерируемые классы объектной модели, задать специфические преобразования типов данных.

Перед началом изучения JAXB, необходимо ознакомиться с двумя понятиями, как маршаллизация и демаршаллизация.

**Маршаллизация -** это процесс преобразования находящихся в памяти данных в формат их хранения. Так, для технологий Java и XML, маршаллизация представляет собой преобразование некоторого набора Java-объектов в XML-документ (или документы). Для баз данных - размещение Java-данных в базе данных.

**Демаршаллизация** - это процесс преобразования данных из формата среды хранения в память, т.е. процесс, прямо противоположный маршаллизации. Иначе говоря, вы можете демаршиллизировать XML-документ в Java VM. Сложность здесь заключается не в декомпозиции данных в плоскую структуру, что не является необходимым, а в отображении нужных данных в нужные переменные Java-кода.

JAXB предлагает быстрый и удобный способ создания двухстороннего преобразования между XML документами и классами языка Java (рисунок 1).

Задавая XSD, компилятор JAXB создает набор классов Java, содержащий весь код, который необходим для анализа XML документов, основанных на данной схеме. Разработчик, использующий созданные классы, может строить дерево объектов Java, изображающих XML документ, обрабатывать содержимое этого дерева и заново создавать XML документы из него.



В таблице 2 приведены соответствия типов XML схемы классам Java.

Таблица 2- Соответствие типов XML схемы классам (типам) Java

|  |  |
| --- | --- |
| **Типы XML Schema** | **Классы (типы) Java** |
| xsd:string | java.lang.String |
| xsd:integer | java.math.BigInteger |
| xsd:int | int |
| xsd:long | long |
| xsd:short | short |
| xsd:decimal | java.math.BigDecimal |
| xsd:float | float |
| xsd:double | double |
| xsd:boolean | boolean |
| xsd:byte | byte |
| xsd:QName | javax.xml.namespace.QName |
| xsd:dateTime | javax.xml.datatype.XMLGregorianCalendar |
| xsd:base64Binary | byte[] |
| xsd:hexBinary | byte[] |
| xsd:unsignedInt | long |
| xsd:unsignedShort | int |
| xsd:unsignedByte | short |
| xsd:time | javax.xml.datatype.XMLGregorianCalendar |
| xsd:date | javax.xml.datatype.XMLGregorianCalendar |
| xsd:g | javax.xml.datatype.XMLGregorianCalendar |
| xsd:anySimpleType | java.lang.Object |
| xsd:anySimpleType | java.lang.String |
| xsd:duration | javax.xml.datatype.Duration |
| xsd:NOTATION | javax.xml.namespace.QName |

Таблица 3- Соответствие классов (типов) Java типам XML схемы

|  |  |
| --- | --- |
| **Классы (типы) Java** | **Типы XML Schema** |
| java.lang.String | xs:string |
| java.math.Biglnteger | xs:integer |
| java.math.BigDecimal | xs:decimal |
| java.util.Calendar | xs:dateTime |
| java.util.Date | xs:dateTime |
| javax.xml.namespace.QName | xs:QName |
| java.net.URl | xs:string |
| javax.xml.datatype.XMLGregorianCalendar | xs:anySimpleType |
| javax.xml.datatype.Duration | xs:duration |
| java.lang.Object | xs:anyType |
| java.awt.lmage | xs:base64Binary |
| javax.activation.DataHandler | xs:base64Binary |
| javax.xml.transform.Source | xs:base64Binary |
| java.util.UUlD | xs:string |

**Создание проекта**

В этом разделе описывается поддержка инструментов IDE NetBeans для JAXB. В частности, речь пойдет о следующих трех вопросах:

* Использование специального мастера в среде IDE для привязки схемы документа XML и его распаковки в набор классов Java, представляющих эту схему
* Использование созданного JAXB класса в программном коде
* Упаковка кода Java в схему XML

В качестве примера будет рассматриваться работа с информацией о книгах (название книги, авторы, год издания), которая в рамках JAXB-технологии предполагает следующую последовательность действий:

1. Создать XML-схему для указанной предметной области

Books.xsd

<?xml version="1.0" encoding="utf-8"?>

<xsd:schema elementFormDefault="qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="book">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="author">

<xsd:complexType>

<xsd:sequence>

<xsd:element minOccurs="0" name="firstName" type="xsd:string"/>

<xsd:element minOccurs="0" name="lastName" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name="title" type="xsd:string"/>

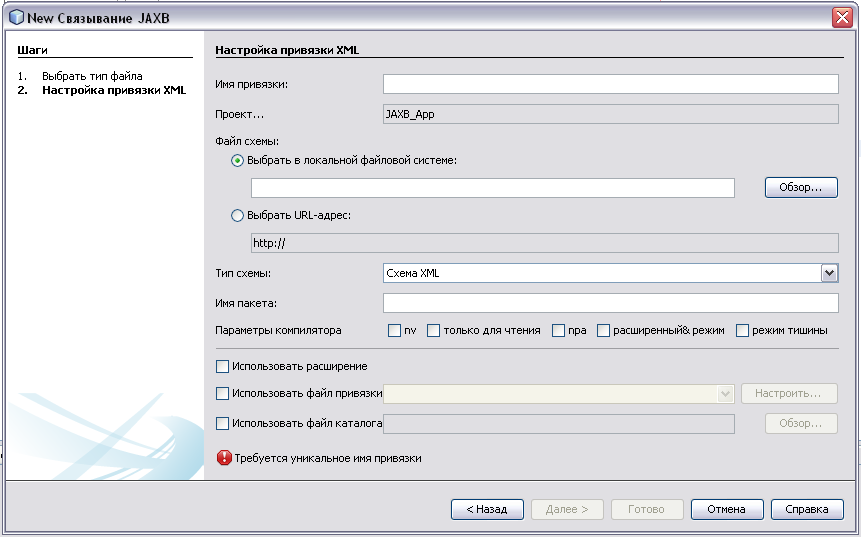
<xsd:element name="year" type="xsd:int"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

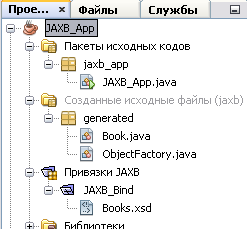
</xsd:schema>

1. Создать новый проект для консольного приложения Java (через пункты главного меню **Файл/Создать проект…**, **Java/ Приложение Java/ Далее>**)
2. В окне **«Проекты»** щелкните правой кнопкой мыши узел созданного проекта и выберите **«Новый» > «Другое» > «XML» > «Связывание JAXB»**. Затем нажмите кнопку **«Далее»**. Появится мастер создания привязки JAXB.

Параметры настройки мастера, представленные выше, служат следующим целям:

* **Имя привязки**. Указывается имя новой привязки JAXB, которое будет использоваться для ее идентификации
* **Проект**. Отображается имя текущего проекта
* **Файл схемы**. Рабочий файл, который может быть доступен локально или в интерактивном режиме
* **Тип схемы**. Поддерживаются следующие языки схемы XML:
* **Схема XML**
* **Relax NG** – простой язык схемы XML. Синтаксис XML
* **Relax NG Compact** – простой язык схемы XML. Синтаксис не типа XML
* **XML DTD** – альтернатива схеме XML предыдущего поколения
* **WSDL** – язык определения веб-служб. Язык схемы XML для определения веб-служб на основе SOAP.
* **Имя пакета**. Указывается пакет. в котором будут созданы объекты Java
* **Параметры компилятора**. Доступны многие параметры компилятора, как описано здесь, в учебном курсе по Java EE 5. Однако для мастера JAXB релевантны только перечисленные ниже параметры, для которых можно установить флажки в мастере:
* **nv**. Не выполняйте строгую проверку достоверности схемы ввода. По умолчанию перед обработкой выполняется строгая проверка достоверности исходной схемы. Это не означает, что во время привязки компилятор не выполняет никакой проверки достоверности; это просто означает, что производится менее строгая проверка.
* **readOnly**. Принудительно с помощью компилятора присвойте пометку 'только для чтения' для созданных ресурсов Java. По умолчанию компилятор не защищает создаваемые исходные файлы Java от записи.
* **npa**. Подавляется создание аннотаций уровня пакета в \*\*/package-info.java. Установка этого флажка приводит к тому, что создаваемый код встраивает эти аннотации в другие создаваемые классы.
* **подробный**. Получения максимального результата компилятора, такого как сведения о ходе выполнения и предупреждения.
* **тихий**. Подавление результатов компилятора, таких как сведения о ходе выполнения и предупреждения.
* Использование расширения. По умолчанию компиляторы строго принудительно реализуют правила, описанные в главе 'Совместимость' спецификации по JAXB. В стандартном (строгом) режиме также существуют ограничения на использование только тех настроек привязки, которые определены в спецификации. При выборе этого параметра разрешается использовать расширения JAXB Vendor Extensions.
* Использовать файл привязки. Позволяет импортировать и редактировать один или несколько файлов настройки привязки JAXB.
* Использовать файл каталога. Позволяет импортировать и редактировать файлы каталога OASIS.

1. Введите текст в поле «Имя привязки»
2. Установите в области «Файл схемы» флажок «Выбрать из локальной файловой системы». Нажмите кнопку «Обзор» и перейдите к файлу схемы XSD
3. Разверните раскрывающийся список в области "Тип схемы" и выберите "Схема XML" (если этот параметр не был выбран автоматически)

В среде IDE будут созданы объекты Java на основе указанной XML-схемы:

а) ObjectFactory.java

package generated;

import javax.xml.bind.annotation.XmlRegistry;

@XmlRegistry

public class ObjectFactory

{

public ObjectFactory() {}

public Book createBook() {return new Book();}

public Book.Author createBookAuthor() {return new Book.Author();}

}

б) Book.java

package generated;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name = "", propOrder = {"author","title","year"})

@XmlRootElement(name = "book")

public class Book

{

@XmlElement(required = true)

protected Book.Author author;

@XmlElement(required = true)

protected String title;

protected int year;

public Book.Author getAuthor() {return author;}

public void setAuthor(Book.Author value) {this.author = value;}

public String getTitle() {return title;}

public void setTitle(String value) {this.title = value;}

public int getYear() {return year;}

public void setYear(int value) {this.year = value;}

@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name = "", propOrder = {"firstName","lastName"})

public static class Author

{

protected String firstName;

protected String lastName;

public String getFirstName() {return firstName;}

public void setFirstName(String value) {this.firstName = value;}

public String getLastName() {return lastName;}

public void setLastName(String value) {this.lastName = value;}

}

}

1. Создать класс, который будет производить маршалинг и демаршалинг данных

JAXB\_App.java

package jaxb\_app;

import java.io.\*;

import java.util.logging.\*;

import javax.xml.bind.\*;

import javax.xml.bind.annotation.XmlRootElement;

import generated.\*;

public class JAXB\_App

{

public static void main(String[] args)

{

OutputStream os = null;

try

{

// Создаем объекты

ObjectFactory of = new ObjectFactory();

Book poltava = of.createBook();

Book.Author pushkin = of.createBookAuthor();

pushkin.setFirstName("Александр");

pushkin.setLastName("Пушкин");

poltava.setTitle("Полтава");

poltava.setAuthor(pushkin);

poltava.setYear(2008);

//Создаем выходной поток

File ofile = new File("book.xml");

os = new FileOutputStream(ofile);

// Маршаллинг

JAXBContext context = JAXBContext.newInstance(Book.class);

Marshaller m = context.createMarshaller();

m.marshal(poltava, os);

os.close();

// Демаршаллинг

File xmlFile = new File("book.xml");

Unmarshaller u = context.createUnmarshaller();

Book book = (Book) u.unmarshal(xmlFile);

System.out.println(book.getAuthor().getFirstName());

System.out.println(book.getAuthor().getLastName());

System.out.println(book.getTitle());

System.out.println(book.getYear());

}

catch(IOException ex)

{

Logger.getLogger(JAXB\_App.class.getName()).log(Level.SEVERE, null, ex);

}

catch(JAXBException ex)

{

Logger.getLogger(JAXB\_App.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

В итоге будут получены следующие результаты:

Маршаллинг

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<book>

<author>

<firstName>Александр</firstName>

<lastName>Пушкин</lastName>

</author>

<title>Полтава</title>

<year>2008</year>

</book>

Демаршаллинг

Александр

Пушкин

Полтава

2008

**Порядок выполнения работы**

1. Ознакомиться с теоретическими сведениями
2. Создать тестовый XML документ с моделью данных в соответствии с вариантом:

|  |  |
| --- | --- |
| Вариант | Модели данных |
| 1 | Заказ на покупку товаров |
| 2 | Список книг библиотеки |
| 3 | Телефонный справочник |
| 4 | Меню в ресторане |
| 5 | Прайс-лист фирмы по продаже компьютеров |
| 6 | Экзаменационная ведомость |
| 7 | Видеотека |
| 8 | Каталог интернет-ресурсов |

1. Разработать DTD
2. Разработать XML-схему
3. Ввести и отладить приложение, демонстрирующее кругооборот данных. Для технологий Java и XML это означает перемещение данных из XML-документа в экземпляры переменных Java и обратно в XML-документ
4. Составить отчет о выполнении лабораторной работы

**Содержание отчета**

1. Название и тема лабораторной работы
2. Цель лабораторной работы
3. Краткие теоретические сведения
4. Ход выполнения работы
5. Текст программ, XML, XSD, DTD, используемых для выполнения

лабораторной работы;

1. Выводы

**Конрольные вопросы и задания**

1. Что такое XML?
2. Какие существует виды XML-документов?
3. Структура XML документа
4. Что такое XML схема?
5. Чем отличается XML схема от DTD?
6. Объясните назначение модификаторов \*, +, ?
7. Дайте определение XSD
8. Объясните понятия маршаллизация и демаршаллизация
9. Что такое валидация?
10. Какие возможности представляет технология JAXB?