Лекция 12. Работа с XML-данными в Java

NetCracker®

XML – Extensible Markup Language

Для чего нужен:

- Интеграция данных из различных источников
- Локальная обработка данных на клиенте.
- Просмотр и манипулирование данными в различных разрезах.
- Возможность частичного обновления данных.

XML. Конструкции языка

Элементы данных

Элементами могут выступать как обычный текст, так и другие, вложенные, элементы документа, секции CDATA, инструкции по обработке, комментарии.

Например:

```
<country id="Russia">
  <city>
    <title>Новосибирск</title>
    <state>Siberia</state>
    <universities-list>
      <university id="1">
        <title>СуМГУ
        </title>
        <address URL="www.sumdu.edu.ua"/>
        <description>родной университет</description>
      </university>
    </universities-list>
  </city>
</country>
```

Well-formed XML

- Каждый **открывающий тэг**, определяющий некоторую область данных в документе обязательно **должен иметь своего закрывающего "напарника“**.

- В XML учитывается **регистр символов**.

- Все **значения атрибутов**, используемых в определении тэгов, должны быть **заключены в кавычки**.

- 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**, как в HTML).

- Документ должен иметь только **один элемент верхнего уровня** (элемент Документ или корневой элемент). Все другие элементы должны быть вложены в элемент верхнего уровня.

XML. Конструкции языка

Комментарии

```
<!-- комментарий -->
```

Специальные символы

```
&lt; , &gt; &quot; или &#036;(десятичная форма записи), &#x1a
```

Атрибуты

- имя должно начинаться с буквы или символа подчеркивания (_), после чего могут следовать или не следовать другие буквы, цифры, символы точки (.), тире (–) или подчеркивания;
- каждое имя атрибута может только один раз присутствовать в одном и том же начальном теге или в теге пустого элемента.

Например:

```
<LIST _1stPlace="Sam"> - правильно
<LIST 1stPlace="Sam">  - неправильно
```

XML. Атрибуты

- строка может быть заключена как в одинарные ('), так и в двойные кавычки (");
- строка не может содержать внутри себя тот же символ кавычек, которыми она ограничена;
- строка может содержать ссылку на символ или ссылку на внутренние примитивы общего назначения;
- строка не может содержать символ < (Синтаксический анализатор может воспринять этот символ как начало описания XML-разметки.)
- строка не может содержать символ &, если это не ссылка на символ или примитив.

XML. Конструкции языка

Директивы анализатора

```
<?тип инструкция ?>
```

CDATA `<![CDATA[...]]>`

Необходима 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.

DTD (Document Type Definition) `<!DOCTYPE ... >`

- Содержит правила, описывающие структуру документа
- Документ автоматически проверятся на соответствие этим правилам
- Описывает дочерние элементы и атрибуты для каждого элемента

Valid XML

Document Type Definition может быть описан в документе или во внешнем файле:

- `<!DOCTYPE document-root [ ... ]>`
- `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- `<!DOCTYPE people_list SYSTEM "example.dtd">`

```
<!DOCTYPE COLLECTION [
  <!ENTITY greeting "helloworld">
  <!ELEMENT tag_name1 EMPTY>
  <!ELEMENT tag_name2 (#PCDATA|ANY|(tag,tag2)*,tag3?)>
  <!ATTLIST payment type CDATA #REQUIRED>
]>
```

XML Schema

Schema

- Предназначена для того же что и DTD
- Для описания правил используется XML
- Более гибкие возможности, чем у DTD
- Сложнее в восприятии и создании средств её обработки

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="страна" type="страна"/>
  <xs:complexType name="страна">
    <xs:sequence>
      <xs:element name="название" type="xs:string"/>
      <xs:element name="население" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```
<страна
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="country.xsd">

  <название>Франция</название>
  <население>59.7</население>
</страна>
```

Пример. Входные данные

Есть xml-документ:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE students SYSTEM "students.dtd">
<students>
  <student>
    <name>Nikolaj</name>
    <surname>Ivanov</surname>
    <age>23</age>
    <group>PF-11</group>
  </student>
  <student>
    <name>Petr</name>
    <surname>Kilkin</surname>
    <age>22</age>
    <group>FP-22</group>
  </student>
  <student>
    <name>Petr</name>
    <surname>Taranov</surname>
    <age>43</age>
    <group>FP-33</group>
  </student>
</students>
```

```
<!ELEMENT students (student*)>
<!ELEMENT student (name,surname,age,group)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT group (#PCDATA)>
```

Пример. Желанный результат

```
Student:
  name: Nikolaj
  surname: Ivanov
  age: 23
  group: PF-11
Student:
  name: Petr
  surname: Kilkin
  age: 22
  group: FP-22
Student:
  name: Petr
  surname: Taranov
  age: 43
  group: FP-33
```

Разбор XML документов

XML парсеры:

- **DOM-парсер (Document Object Model)** – предварительно анализируется XML-документ и сохраняется дерево элементов XML в оперативной памяти. Требователен к ресурсам.

- **SAX-парсер (Simple API for XML)** — парсер, основывающийся на событиях (event-based). Быстр, за счет разбора только конкретной части документа. Занимает мало памяти.

SAX API

SAX API (на примере org.xml.sax)

Представляет следующие интерфейсы для манипулирования XML:

- **ContentHandler** – ключевой интерфейс. Вызывая различные методы интерфейса XMLReader сообщает приложению о содержимом разбираемого документа
- **DTDHandler** – определяет методы которые используются для получения от XMLReader информации о нотации и необрабатываемых обьявлениях сущностей в DTD документа.
- **ErrorHandler** – используется для генерации ридером предупреждений, ошибке или неисправимой ошибке.
- **InputSource** – описывает источник входных данных: поток байт или символов, файл и т.п. откуда парсеру считывать документ.
- **XMLReader** – основной интерфейс содержащий методы парсера реализованые в других классах, таких как SAXParser или SAXParcerFactory
- Пакет **org.xml.sax.helpers** – содержит вспомогательные классы необходимые при работе с SAX-парсерами: DefaultHandler, XMLReaderFactory, XMLReaderImpl, ParserAdapter.

Пример. SAX-парсинг

```
1.  import javax.xml.parsers.*;
2.  import org.xml.sax.*;
3.  import java.io.IOException;

4.  public class SAXExample extends org.xml.sax.helpers.DefaultHandler {

5.    public void process(String filename)
6.        throws SAXException, IOException, ParserConfigurationException {

7.      SAXParserFactory.newInstance().
8.        newSAXParser().parse(filename, this);
9.    }

10.   public static void main(String[] args) throws Exception {
11.
12.     new SAXExample().process(args[0]);
13.   }
14.   ...
```

Пример. SAX-парсинг. Продолжение

```
14.    ...
15.    private int level = 0;
16.    private boolean inStudent = false;
17.    private StringBuffer text = new StringBuffer();
18.    @Override
19.    public void startElement(String uri, String localName, String qName, Attributes attributes)
20.        throws SAXException {
21.
22.        level++;
23.        if (level == 2 && qName.equals("student")) {
24.            inStudent = true;
25.            System.out.println("Student:");
26.        }
27.        text.setLength(0);
28.    }
29.    @Override
30.    public void characters(char[] ch, int start, int length) throws SAXException {
31.      text.append(ch, start, length);
32.     }
33.     @Override
34.    public void endElement(String uri, String localName, String qName) throws SAXException {
35.        if (level == 3 && inStudent)
36.            System.out.println(" "+ qName +": "+ text);
37.        if (level == 2)
38.            inStudent = false;
39.        level--;
40.    }
```

DOM API

DOM API (на примере org.w3c.dom)

Представляет следующие интерфейсы:

- **Node** – представляет произвольный элемент дерева (включая текст и атрибуты).
  - **Document** – представляет документ DOM и служит корнем дерева документа.
  - **DocumentType** – данный интерфейс представляет DTD документа
  - **Element** – представляет элемент (тэг) документа, который может иметь подузлы.
  - **DocumentFragment** – представляет часть (или фрагмент) документа, один или несколько смежных узлов со всеми подэлементами.
  - **CDATASection** – представляет раздел CDATA.
- **NodeList** – представляет упорядоченное множество узлов предназначенных только для чтения.
- **DOMException** – при порождении ошибки API DOM создается экземпляр данного класса

Пример. DOM-парсинг

```
1.  import org.w3c.dom.*;
2.  import javax.xml.parsers.*;
3.  import org.xml.sax.*;
4.  import java.io.IOException;
5.  public class DOMExample {
6.      public static Document parse(String filename)
7.          throws ParserConfigurationException, SAXException, IOException {
8.          return DocumentBuilderFactory.newInstance().
9.              newDocumentBuilder().parse(filename);
10.     }
11.     public static void main(String[] args) throws Exception {
12.         Document doc = parse(args[0]);
13.         NodeList items = doc.getDocumentElement().getChildNodes();
14.         for (int i = 0; i < items.getLength(); i++)
15.             if (items.item(i).getNodeName().equals("student"))
16.                 printStudent(items.item(i));
17.     }
18.     private static void printStudent(Node node) {
19.         System.out.println("Student:");
20.         NodeList nodes = node.getChildNodes();
21.         for (int i = 0; i < nodes.getLength(); i++)
22.             if (nodes.item(i).getNodeType() == Node.ELEMENT_NODE)
23.                 System.out.println(" "+ nodes.item(i).getNodeName() + ": "+
24.                     nodes.item(i).getFirstChild().getNodeValue());
25.     }
26. }
```

XSL – *e**X**tensible **S**tylesheet **L**anguage*

- **XSLT** – трансформации. XML документ, описывающий способ преобразования одного XML документа в другой (не обязательно XML) документ. Пространство имен http://www.w3.org/1999/XSL/Transform.
- **XPath** – язык запросов к XML документу. Состоит из пути к элементу в дереве или вызов функции, результатом запроса будет набор всех элементов, соответствующих пути.

XSLT – *eXtensible Stylesheet Language Transformations*

- `<template match="xpath expression">... body ...</template>`
- `<value-of select="xpath expression"/>`
- `<for-each select="xpath expression">... body ...</for-each>`
- `<sort select="xpath"/>`
- `<if test="xpath">...</if>`
- `<choose>`
  `<when test="xpath">...</when>`
  `<otherwise>...</otherwise>`
  `</choose>`
- `<apply-templates select="xpath"/>`
- `<output method="xml|html|text" version="string"`
  `encoding="string" omit-xml-declaration="yes|no"`
  `doctype-public="string"`
  `doctype-system="string" indent="yes|no"/>`

XPath

- Пример: `/html/body/*/span[@name="span1"`
  Равнозначно: `/child::html/child::body/child::*/child::span[attribute::name="span1"]`
- Оси:
  ancestor, ancestor-or-self, attribute (@), child (*нет*), descendant, descendant-or-self (//), following, following-sibling, namespace, parent (..), preceding, preceding-sibling, self (.)
- Функции:
  document, format-number, node, text, current, position, last, count, id, sum
- Примеры:
  - `node//li[last()]`
  - `../text()`
  - `/root/@attribute`
  - `//book[price > 20]`
  - `document($path)/root/node`

Пример. XSLT-преобразование

```
1.  <?xml version="1.0"?>
2.  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3.    <xsl:output method="text" indent="no"/>
4.    <xsl:strip-space elements="*"/>

5.    <xsl:template match="/students/student">
6.      <xsl:text>Student:&#010;</xsl:text>
7.      <xsl:apply-templates select="./*"/>
8.    </xsl:template>

9.    <xsl:template match="//student/*">
10.     <xsl:text> </xsl:text>
11.     <xsl:value-of select="name()"/>
12.     <xsl:text>: </xsl:text>
13.     <xsl:value-of select="text()"/>
14.     <xsl:text>&#010;</xsl:text>
15.   </xsl:template>
16. </xsl:stylesheet>
```

Пример. XSLT-преобразование из Java

```
1.  import javax.xml.transform.*;
2.  import javax.xml.transform.stream.*;
3.  import java.io.*;

4.  public class TransformXML {

5.    public static void main(String[] args)
6.      throws TransformerException, IOException, TransformerConfigurationException {

7.      TransformerFactory.newInstance().
8.        newTransformer(new StreamSource(args[0])).
9.          transform(
10.           new StreamSource(args[1]),
11.           new StreamResult(System.out));
12.   }
13. }
```

1. Флэнаган Д. Java. Справочник, 4-е издание Пер. с англ. –СПб: Символ-Плюс, 2004.-1040с.
2. http://intuit.ru, курс “Основы XML”
3. http://www.java-tips.org/java-se-tips/java.lang.reflect/
4. http://j2w.blogspot.com/2008/01/xml-dom.html
5. http://java.sun.com/docs/books/tutorial/reflect/class/index.html
6. http://www.w3schools.com
7. Bruce Eckel. Thinking in Java 2. 2000.

Thank you!