XML Parsing using JAVA

Prof. (Dr.) Vipul K. Dabhi
Associate Professor,
Dept. of Information Technology,
D. D. University

JAVA XML Parsers

- Some commonly used JAVA XML Parsers are
  - JAXP
    - DOM (Document Object Model)
    - SAX (Simple API for XML)
    - StAX (Streaming API for XML)
  - JAXB (Java Architecture for XML Binding)
- There are some other APIs also available for XML parsing in java, for example JDOM and JiBX.

DOM Parser

- DOM stands for Document Object Model. DOM is part of the Java API for XML processing (JAXP).
- DOM Parser is the easiest JAVA XML Parser to learn and implement.
- DOM parser loads the XML file into memory and constructs tree representation of XML file.
- The tree representation can be traversed node by node to parse the XML.
- DOM Parser is good for small XML files but when file size increases it performs slow and consumes more memory.

DOM Parser

- Element nodes and text nodes are the two most common types of nodes.
- With DOM functions nodes can be created, removed, the contents of nodes can be changed.

DOM Parser Packages

- import org.w3c.dom.*;
  - W3C definitions for a DOM, DOM exceptions, entities and nodes
- import javax.xml.parsers.*;
  - JAXP API Classes. All XML parsing related classes and methods are inside JAXP.
- import java.io.*;
  - classes used to read the sample XML file and manage output

Example : Students.xml

```
<students>
   <student id="1">
      <firstname>Pinal</firstname>
      <lastname>Shah</lastname>
      <marks>23</marks>
   </student>
   <student id="2">
      <firstname>Martin</firstname>
      <lastname>Patel</lastname>
      <marks>31</marks>
   </student>
   <student id="3">
      <firstname>Vivek</firstname>
      <lastname>Gandhi</lastname>
      <marks>28</marks>
   </student>
</students>
```

DOM Parser : XML File Reading

```
public static void main(String argv[]) throws SAXException, IOException,
    ParserConfigurationException {

    File xmlFile = new
    File("C:\\Users\\abc\\Documents\\NetBeansProjects\\DOMParserTest\\src\\dom
    parser\\Students.xml");

     /* obtain an instance of a factory that can give instance of document builder
    */
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder dBuilder = factory.newDocumentBuilder();

     /*DocumentBuilder contains the API to obtain DOM Document instances
    from an XML document. The parse() method parses the XML file into a
    Document. */
    Document doc = dBuilder.parse(xmlFile);
```

DOM Parser : XML File Reading

```
doc.getDocumentElement().normalize();

// get the root element of the document
System.out.println("Root element: " +
    doc.getDocumentElement().getNodeName());

/* get a NodeList of user elements in the document
    with getElementsByTagName() */
NodeList nList = doc.getElementsByTagName("student");
```

```
for (int i = 0; i < nList.getLength(); i++)    {
        Node nNode = nList.item(i);
        System.out.println("\nCurrent Element: " + nNode.getNodeName());

        if (nNode.getNodeType() == Node.ELEMENT_NODE)    {
           Element elem = (Element) nNode;
           String uid = elem.getAttribute("id");  // get element attribute with getAttribute()

            // get the text content of the three subelements of the user element
           Node node1 = elem.getElementsByTagName("firstname").item(0);
           String fname = node1.getTextContent();
           Node node2 = elem.getElementsByTagName("lastname").item(0);
           String lname = node2.getTextContent();
           Node node3 = elem.getElementsByTagName("marks").item(0);
           String marks = node3.getTextContent();

           System.out.printf("User id: %s%n", uid);
           System.out.printf("First name: %s%n", fname);
           System.out.printf("Last name: %s%n", lname);
           System.out.printf("Marks: %s%n", marks);              }     }
```

DOM Parser : XML File Reading

Output

Root element: students

Current Element: student
User id: 1
First name: Pinal
Last name: Shah
Marks: 23

Current Element: student
User id: 2
First name: Martin
Last name: Patel
Marks: 31

Current Element: student
User id: 3
First name: Vivek
Last name: Gandhi
Marks: 28

DOM Parser : XML File Creation

- The root element will be “DDU Employees”
- The root element will have child element “FoT Employees”.
- The employee information will be written in “Employee” element, which is child element of “FoT Employees”.
- Every employee has an attribute named “id”
- Employee element will have four elements – “name”, “age”, “gender”, “salary”

Sample XML File

```
<DDUEmployees>
   <FoTEmployee>

      <Employee>
            <name>Nilesh Bhatt</name>
            <age>42</age>
            <gender>M</gender>
            <salary>45000</salary>
      </Employee>

      <Employee>
            <name>Priti Soni</name>
            <age>32</age>
            <gender>F</gender>
            <salary>35000</salary>
      </Employee>

   </FoTEmployee>
</DDUEmployees>
```

DOM Parser : XML File Creation

```
public static void main(String[] args) {
   try {
       DocumentBuilderFactory dbFactory =
       DocumentBuilderFactory.newInstance();
       DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
       Document doc = dBuilder.newDocument();
       // root element
       Element rootElement = doc.createElement("DDUEmployees");
       doc.appendChild(rootElement);

       // FoT Employees element
       Element FoTEmp = doc.createElement("FoTEmployees");
       rootElement.appendChild(FoTEmp);
```

DOM Parser : XML File Creation

```
// setting attribute to element
Attr attr = doc.createAttribute("Faculty");
attr.setValue("Faculty of Technology");
FoTEmp.setAttributeNode(attr);

// Employee element
Element Employee = doc.createElement("Employee");
Attr attrType = doc.createAttribute("id");
attrType.setValue("1");
Employee.setAttributeNode(attrType);

Element name = doc.createElement("Name");
name.appendChild(doc.createTextNode("Nilesh Bhatt"));
Element age = doc.createElement("Age");
age.appendChild(doc.createTextNode("42"));
Element salary = doc.createElement("Salary");
salary.appendChild(doc.createTextNode("45000"));
```

DOM Parser : XML File Creation

```
        Employee.appendChild(name);
        Employee.appendChild(age);
        Employee.appendChild(salary);

        FoTEmp.appendChild(Employee);

       // write the content into xml file
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(doc);

        StreamResult result = new StreamResult(new File("C:\\employees.xml"));
        transformer.transform(source, result);
        // Output to console for testing
        StreamResult consoleResult = new StreamResult(System.out);
        transformer.transform(source, consoleResult);
      } catch (Exception e)     {
        e.printStackTrace();     }
}
```

SAX Parser

- SAX is an acronym for Simple API for XML.
- SAX Parser parses the XML file line by line and triggers events when it encounters opening tag, closing tag or character data in XML file.
- This is why SAX parser is called an event-based parser

When to use SAX Parser?

- Process the XML document in a linear fashion from top to down.
- Processing a very large XML document whose DOM tree would consume too much memory.
  - Typical DOM implementations use ten bytes of memory to represent one byte of XML.
- The problem to be solved involves only a part of the XML document.
- Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream.

Drawbacks of SAX Parser

- No random access to an XML document since it is processed in a forward-only manner.
- If there is a need to keep track of data that the parser has seen, you must write the code and store the data on your own.

SAX Parser

- ContentHandler Interface
  - This interface specifies the callback methods that the SAX parser uses to notify an application program of the components of the XML document that it has seen
  - void startDocument()
    - Called at the beginning of a document.
  - void endDocument()
    - Called at the end of a document.
  - void startElement(String uri, String localName, String qName, Attributes atts)
    - Called at the beginning of an element.
  - void endElement(String uri, String localName,String qName)
    - Called at the end of an element
  - void characters(char[] ch, int start, int length)
    - Called when character data is encountered

SAX Parser

- Attributes Interface
  - This interface specifies methods for processing the attributes connected to an element
  - int getLength()
    - Returns number of attributes
  - String getQName(int index)
  - String getValue(int index)
  - String getValue(String qname)

SAX Parser : XML File Read Example

- Input XML File : NewTest.xml
- Java Program (.java) Files
  - SAXHandler.java
    - Extends DefaultHandler class (which implements - ContentHandler and Attribute interfaces)
  - Main.java

NewTest.xml

```
<?xml version="1.0"?>
<company>
    <staff>
        <firstname>Vinkal</firstname>
        <lastname>Desai</lastname>
        <nickname>Vinky</nickname>
        <salary>10000</salary>
    </staff>
    <staff>
        <firstname>Divyang</firstname>
        <lastname>Dalal</lastname>
        <nickname>DD</nickname>
        <salary>20000</salary>
    </staff>
</company>
```

SAXHandler.java

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SAXHandler extends DefaultHandler{
     boolean bfname = false;    boolean blname = false;    boolean bnname = false; boolean bsalary = false;

   @Override
     public void startElement(String uri, String localName,String qName,
           Attributes attributes) throws SAXException {
        System.out.println("Start Element :" + qName);
        if (qName.equalsIgnoreCase("FIRSTNAME")) {
                  bfname = true;
        }
        if (qName.equalsIgnoreCase("LASTNAME")) {
                  blname = true;
        }
```

SAXHandler.java

```
if (qName.equalsIgnoreCase("NICKNAME")) {
                        bnname = true;                }
            if (qName.equalsIgnoreCase("SALARY")) {
                        bsalary = true;                        }
    }
@Override
    public void endElement(String uri, String localName,
            String qName) throws SAXException {
            System.out.println("End Element :" + qName);
    }
```

SAXHandler.java

```
@Override
    public void characters(char ch[], int start, int length) throws SAXException {
            if (bfname)                         {
                    System.out.println("First Name : " + new String(ch, start, length));
                    bfname = false;       }
            if (blname)                         {
                    System.out.println("Last Name : " + new String(ch, start, length));
                    blname = false;       }
            if (bnname)                         {
                    System.out.println("Nick Name : " + new String(ch, start, length));
                    bnname = false;       }
            if (bsalary)                                {
                    System.out.println("Salary : " + new String(ch, start, length));
                    bsalary = false;      }
    }
}
```

Main.java

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

public class Main {
   public static void main(String[] args) {
     try {
         SAXParserFactory factory = SAXParserFactory.newInstance();
         SAXParser saxParser = factory.newSAXParser();
         SAXHandler handler = new SAXHandler();

     saxParser.parse("C:\\Users\\abc\\Documents\\NetBeansProjects\\SAXParserTest\\
     src\\saxparsertest\\NewTest.xml", handler);
      }
      catch (Exception e) {
         e.printStackTrace();
      }
}
}
```

Output

```
Start Element :company
Start Element :staff
Start Element :firstname
First Name : Vinkal
End Element :firstname
Start Element :lastname
Last Name : Desai
End Element :lastname
Start Element :nickname
Nick Name : Vinky
End Element :nickname
Start Element :salary
Salary : 10000
End Element :salary
End Element :staff
End Element :company
```

StAX Parser

- StAX is an acronym for Streaming API for XML.
- Stream-based parsers are very useful when your application has memory limitations.
  - For example, a cellphone running Java Micro Edition.

Stream based Parsing

- Stream based parsing can be classified into
  - Pull Parsing
    - Client application calls for methods on an XML parsing library when it needs to interact with an XML. Client only gets XML data when it explicitly asks for it.
  - Push Parsing
    - XML parser pushes XML data to the client, when it encounters elements in an XML. Parser sends the data to application irrespective of the application being ready to use it or not.

StAX Vs SAX

- The main difference is that StAX uses a pull mechanism instead of SAX's push mechanism (using callbacks).
- This means the control is given to the client to decide when the events need to be pulled.
  - Therefore, there is no obligation to pull the whole document if only a part of it is needed.
- It provides an easy API to work with XML with a memory-efficient way of parsing.
- Unlike SAX, it doesn't provide schema validation as one of its features.

DOM Vs SAX Vs StAX

Feature	DOM	SAX	StAX
API Type	In Memory Tree	Push, Streaming	Pull, Streaming
Ease of Use	High	Medium	High
Xpath Capability	Yes	No	No
CPU & Memory Efficient	Varies	Good	Good
Forward Only	No	Yes	Yes
Read XML	Yes	Yes	Yes
Write XML	Yes	No	Yes
CRUD XML (Create, Read, Update, Delete)	Yes	No	No

XML Parsers

- JAXP: Java API for XML Parsing
  - Handles XML document in native XML format.
  - Made for XML Programmers
  - Parsers
    - SAX: Simple API for XML Parser
      - Stateless and event driven parser - Parser will go through XML file sequentially and generate an event when it encounters any element. Programmer has to write handler for handing event. Parser does not remember which tag it has read earlier. Requires low memory.
    - DOM: Document Object Model Parser
      - Parser creates in-memory representation (tree) of whole XML document. Requires high memory.

- JAXB: Java API for XML Binding
  - Creates application (object) model – classes from XML schema. Useful to create object complying given XML schema.
  - Made for Application Programmers

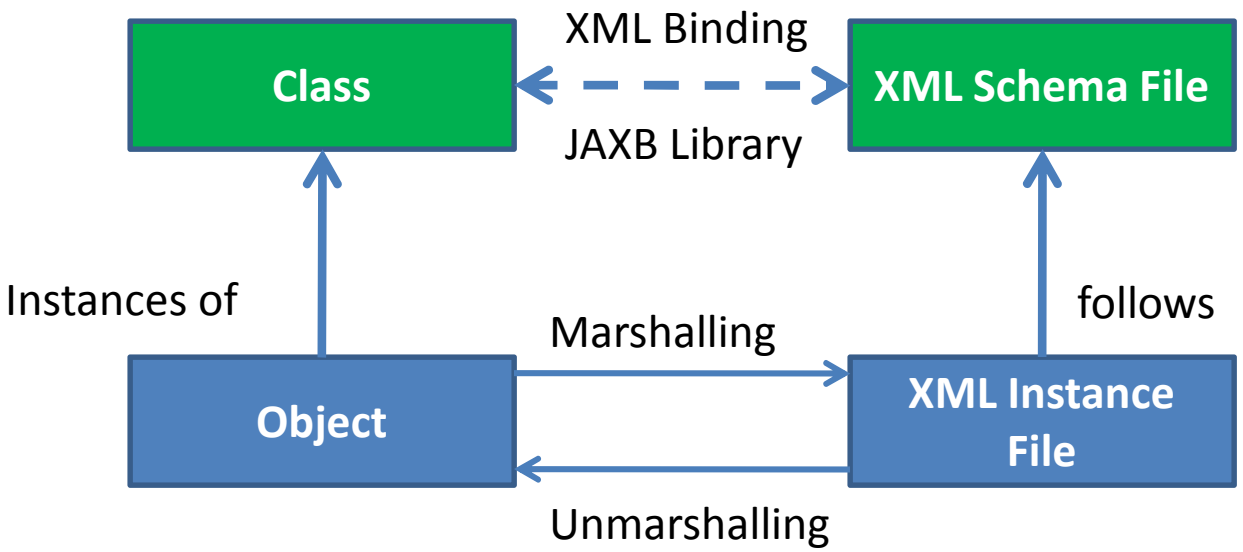
JAXB

- JAXB provides API, tools and a framework that automate the mapping between XML documents and Java objects.
  - Provides compiler that compiles XML schema to Java classes
- JAXB creates classes from given XML schema. JAXB creates objects from XML instance document.

Why JAXB?

- Provides an efficient and standard way of mapping between XML and Java code
  - Programmers don't have to create application specific Java objects
  - Programmers don't have to deal with XML structure, instead deal with business data
- Issues with DOM/SAX model
  - Why do we walk a Parse Tree
  - Why do we write Event Handlers to map XML content to Java Classes.

JAXB- Java API for XML Binding



Marshalling = Objects → XML
Unmarshalling = XML → Objects

- Like objects are instances of classes, XML instance files are instances of XML schema (XSD) file. JAXB Library can generate classes from XML schema file.
- JAXB Library is used to perform marshalling and un-marshalling operations.
- XML Binding - Objects can be marshalled in XML instance file or un-marshalled from XML instance file.

Comparison of Parsers

Feature	JAXP			JAXB
	DOM	SAX	StAX	
Memory Efficient	No	Yes	Yes	Yes
Bidirectional Navigation of XML Document	Yes	No (Forward Only)	No	Yes
XML Manipulation (Create, Read, Update, Delete)	Yes	No	No	Yes
Data Binding	No	No	No	Yes