

UNIT III CLIENT SIDE TECHNOLOGIES

- XML
- Document Type Definition
- XML Schema
- Document Object Model
- Presenting XML
- Using XML Parsers: DOM and SAX
- JavaScript Fundamentals
- Evolution of AJAX
- AJAX Framework
- Web applications with AJAX
- AJAX with PHP
- AJAX with Databases

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

What is XML Parsing?

Parsing XML refers to **going through XML document to access data or to modify data** in one or other way

What is XML Parser?

XML Parser provides way how to access or modify data present in an XML document. various types of parsers which are commonly used to parse XML documents,

- **Dom Parser** - Parses the document by loading the complete contents of the document and creating its complete hierarchical tree in memory
- **SAX Parser** - Parses the document on event based triggers. Does not load the complete document into the memory
- **JDOM Parser** - Parses the document in similar fashion to DOM parser but in more easier way
- **StAX Parser** - Parses the document in similar fashion to SAX parser but in more efficient way
- **XPath Parser** - Parses the XML based on expression and is used extensively in conjunction with XSLT
- **DOM4J Parser** - A java library to parse XML, XPath and XSLT using Java Collections Framework , provides support for DOM, SAX and JAXP

DOM Parser

The XML DOM

- All XML elements can be accessed through the XML DOM
- The XML DOM is:
 - A standard object model for XML
 - A standard programming interface for XML
 - Platform- and language-independent
 - A W3C standard
- In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements**

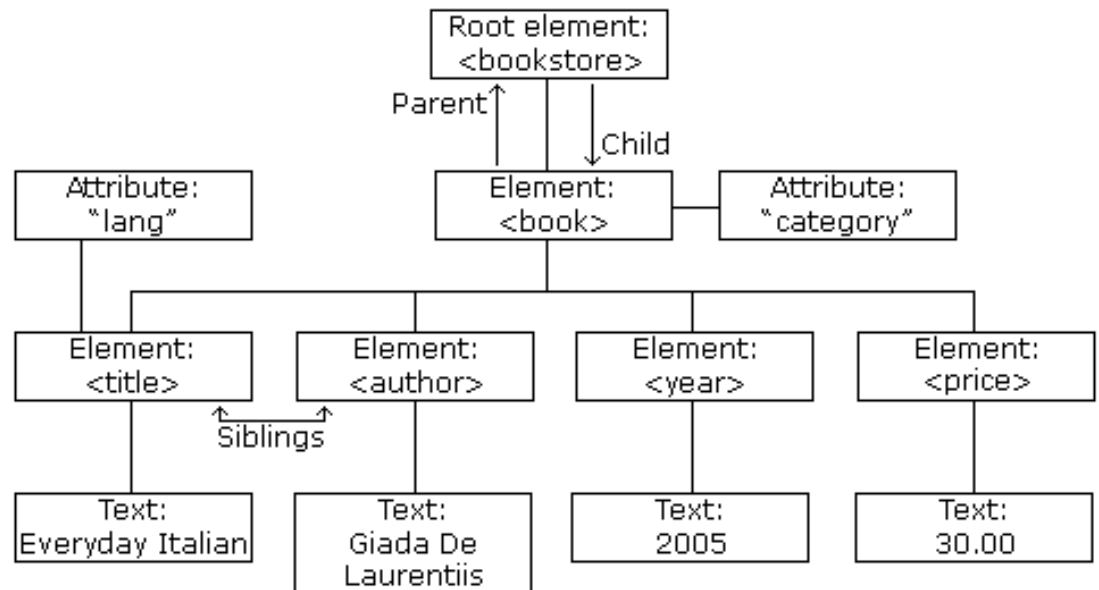
DOM XML Parser in Java

- DOM Stands for **Document Object Model** and it represent an XML Document into tree format which each element representing tree branches
- **DOM Parser** creates an In Memory tree representation of XML file and then parses it, so it requires more memory and it's advisable to have increased the heap size for DOM parser in order to avoid `Java.lang.OutOfMemoryError:java heap space`
- *Parsing XML file using DOM parser is quite fast* if XML file is small but if you try to read a large XML file using DOM parser there is more chances that it will take a long time or even may not be able to load it completely simply because it requires lot of memory to create XML Dom Tree.
- Java provides support DOM Parsing and you can parse XML files in Java using DOM parser. DOM classes are in `w3c.dom` package while **DOM Parser for Java** is in JAXP (Java API for XML Parsing) package

Bookstore.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

DOM Tree



Programming Interface

- The **DOM models XML as a set of node objects**. The nodes can be accessed with JavaScript or other programming languages
- The programming interface to the DOM is defined by a set standard properties and methods

DOM interfaces

- The DOM defines several Java interfaces. Here are the most common interfaces:
 - **Node** - The base data type of the DOM.
 - **Element** - The vast majority of the objects you'll deal with are Elements.
 - **Attr** Represents an attribute of an element.
 - **Text** The actual content of an Element or Attr.
 - **Document** Represents the entire XML document. A Document object is often referred to as a DOM tree.

Common DOM methods

- **Document.getDocumentElement()** - Returns the root element of the document.
- **Node.getFirstChild()** - Returns the first child of a given Node.
- **Node.getLastChild()** - Returns the last child of a given Node.
- **Node.getNextSibling()** - These methods return the next sibling of a given Node.
- **Node.getPreviousSibling()** - These methods return the previous sibling of a given Node.
- **Node.getAttribute(attrName)** - For a given Node, returns the attribute with the requested name.

Steps to Using DOM

- Following are the steps used while parsing a document using DOM Parser.
 - Import XML-related packages.
 - Create a DocumentBuilder
 - Create a Document from a file or stream
 - Extract the root element
 - Examine attributes
 - Examine sub-elements

Import XML-related packages

```
import org.w3c.dom.*;  
import javax.xml.parsers.*;  
import java.io.*;
```

Create a DocumentBuilder

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
DocumentBuilder builder = factory.newDocumentBuilder();
```

Create a Document from a file or stream

```
StringBuilder xmlStringBuilder = new StringBuilder();  
xmlStringBuilder.append("<?xml version='1.0'?> <class> </class>");  
ByteArrayInputStream input = new  
ByteArrayInputStream( xmlStringBuilder.toString().getBytes("UTF-8"));  
Document doc = builder.parse(input);
```

Extract the root element

```
Element root = document.getDocumentElement();
```

Examine attributes

```
//returns specific attribute getAttribute("attributeName");  
//returns a Map (table) of names/values getAttributes();
```

Examine sub-elements

```
//returns a list of subelements of specified name  
getElementsByTagName("subelementName");  
//returns a list of all child nodes getChildNodes();
```

studDetails.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <student id="1">
    <name>Arjun</name>
    <gender>male</gender>
    <dob>16-12-1991</dob>
    <dept>ECE</dept>
    <cgpa>10</cgpa>
  </student>
  <student id="2">
    <name>Ajay karthick</name>
    <gender>male</gender>
    <dob>19-02-1992</dob>
    <dept>EEE</dept>
    <cgpa>10</cgpa>
  </student>
  <student id="2">
    <name>Swetha Mohan</name>
    <gender>female</gender>
    <dob>06-01-1991</dob>
    <dept>ECE</dept>
    <cgpa>10</cgpa>
  </student>
</students>
```

Example- Read xml data using DOM Parser and print in the screen

```
import java.io.File;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
public class DOMParserEx
{

    public static void main(String[] args)
    {
        try {
            File inputFile = new File("studDetails.xml");
            DocumentBuilderFactory dbFactory
                = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();
            System.out.println("Root element :"+
doc.getDocumentElement().getNodeName());
```

```

NodeList nList = doc.getElementsByTagName("student");
        System.out.println("-----");
        for (int temp = 0; temp < nList.getLength(); temp++)

{
    Node nNode = nList.item(temp);
    System.out.println("\nCurrent Element :"+ nNode.getNodeName());
    if (nNode.getNodeType() == Node.ELEMENT_NODE)
    {
        Element eElement = (Element) nNode;
        System.out.println("Student ID : " + eElement.getAttribute("id"));
        System.out.println("Name : " +
eElement.getElementsByTagName("name").item(0).getTextContent());
        System.out.println("Gender : " +
eElement.getElementsByTagName("gender").item(0).getTextContent());
        System.out.println("Date of Birth : " +
eElement.getElementsByTagName("dob").item(0).getTextContent());
            System.out.println("Department : " +
eElement.getElementsByTagName("dept").item(0).getTextContent());
            System.out.println("CGPA : " +
eElement.getElementsByTagName("cgpa").item(0).getTextContent());
        }
    }
    } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}

```

Output

```
Command Prompt
G:\JAVA_PGMS>javac DOMParserEx.java

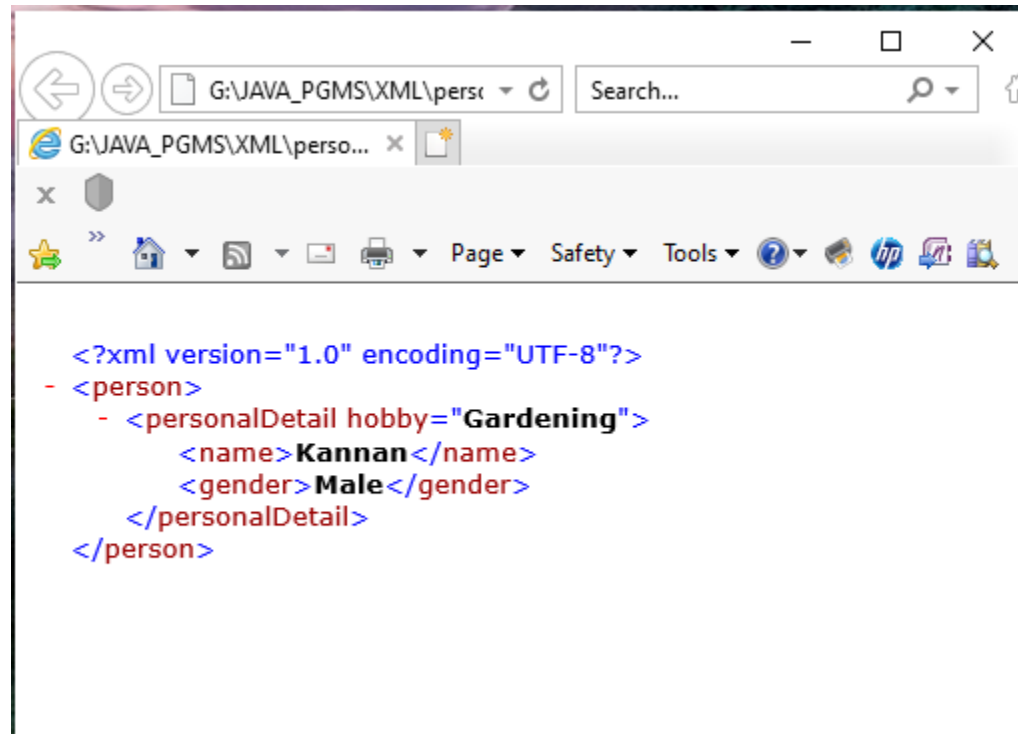
G:\JAVA_PGMS>java DOMParserEx
Root element :students
-----

Current Element :student
Student ID : 1
Name : Arjun
Gender : male
Date of Birth : 16-12-1991
Department : ECE
CGPA : 10

Current Element :student
Student ID : 2
Name : Ajay karthick
Gender : male
Date of Birth : 19-02-1992
Department : EEE
CGPA : 10

Current Element :student
Student ID : 3
Name : Swetha Mohan
Gender : female
Date of Birth : 06-01-1991
Department : ECE
CGPA : 10
```

Creating xml file from java



CreateXML.java

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import java.io.File;
```

```
public class CreateXML{
```

```
    public static void main(String argv[]) {
```

```
        try {
```

```
            DocumentBuilderFactory dbFactory =
            DocumentBuilderFactory.newInstance();
```

```
            DocumentBuilder dBuilder =
            dbFactory.newDocumentBuilder();
```

```
            Document doc = dBuilder.newDocument();
```

```
// root element
Element rootElement = doc.createElement("person");
doc.appendChild(rootElement);

// Personal Detail element
Element perDet = doc.createElement("personalDetail");
rootElement.appendChild(perDet);
// setting attribute to element
Attr attr = doc.createAttribute("hobby");
attr.setValue("Gardening");
perDet.setAttributeNode(attr);

// Personal details element
Element name = doc.createElement("name");
name.appendChild(doc.createTextNode("Kannan"));
perDet.appendChild(name);
Element gen = doc.createElement("gender");
gen.appendChild(doc.createTextNode("Male"));
perDet.appendChild(gen);
```

```
// 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("G:\\JAVA_PGMS\\XML\\person.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 XML Parser in Java

- SAX Stands for **Simple API for XML Parsing**
- This is an event based XML Parsing and it *parse XML file step by step* so much suitable for large XML Files.
- SAX XML Parser fires an event when it encountered opening tag, element or attribute, and the parsing works accordingly
- It's recommended to **use SAX XML parser for parsing large XML files in Java** because it doesn't require to load whole XML file in Java and it can read a big XML file in small parts
- Java provides support for SAX parser and you can parse any XML file in Java using SAX Parser
- One disadvantage of using SAX Parser in java is that **reading XML file in Java using SAX Parser** requires more code in comparison of DOM 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.
- **void ignorableWhitespace(char[] ch, int start, int length)** - Called when a DTD is present and ignorable whitespace is encountered.
- **void processingInstruction(String target, String data)** - Called when a processing instruction is recognized.
- **void setDocumentLocator(Locator locator)** - Provides a Locator that can be used to identify positions in the document.
- **void skippedEntity(String name)** - Called when an unresolved entity is encountered.
- **void startPrefixMapping(String prefix, String uri)** - Called when a new namespace mapping is defined.
- **void endPrefixMapping(String prefix)** - Called when a namespace definition ends its scope.

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)**

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

<students>

<student id="1">

<name>Arjun</name>

<gender>male</gender>

<dob>16-12-1991</dob>

<dept>ECE</dept>

<cgpa>10</cgpa>

</student >

<student id="2">

<name>Ajay karthick</name>

<gender>male</gender>

<dob>19-02-1992</dob>

<dept>EEE</dept>

<cgpa>10</cgpa>

</student>

<student id="2">

<name>Swetha Mohan</name>

<gender>female</gender>

<dob>06-01-1991</dob>

<dept>ECE</dept>

<cgpa>10</cgpa>

</student>

</students>

studDetails.xml

SAXParserEX.java

```
import java.io.File;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

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

public class SAXParserEx {
    public static void main(String[] args){

        try {
            File inputFile = new File("studDetails.xml");
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            UserHandler userhandler = new UserHandler();
            saxParser.parse(inputFile, userhandler);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
class UserHandler extends DefaultHandler
{
    boolean name = false;
    boolean gender = false;
    boolean dob = false;
    boolean dept = false;
    boolean cgpa = false;
    @Override
    public void startElement(String uri,String localName, String qName, Attributes attributes)
throws SAXException
    {
        if (qName.equalsIgnoreCase("student")) {
            String id = attributes.getValue("id");
            System.out.println("Student Id : " + id);
        }
        else if (qName.equalsIgnoreCase("name"))
        {
            name = true;
        }
        else if (qName.equalsIgnoreCase("gender"))
        {
            gender = true;
        }
    }
}
```

```
else if (qName.equalsIgnoreCase("dob"))
{
    dob = true;
}
else if (qName.equalsIgnoreCase("dept"))
{
    dept = true;
}
else if (qName.equalsIgnoreCase("cgpa"))
{
    cgpa = true;
}
}
@Override
public void endElement(String uri,String localName, String qName) throws SAXException
{
    if (qName.equalsIgnoreCase("student")) {
        System.out.println("End Element :" + qName);
    }
}
```

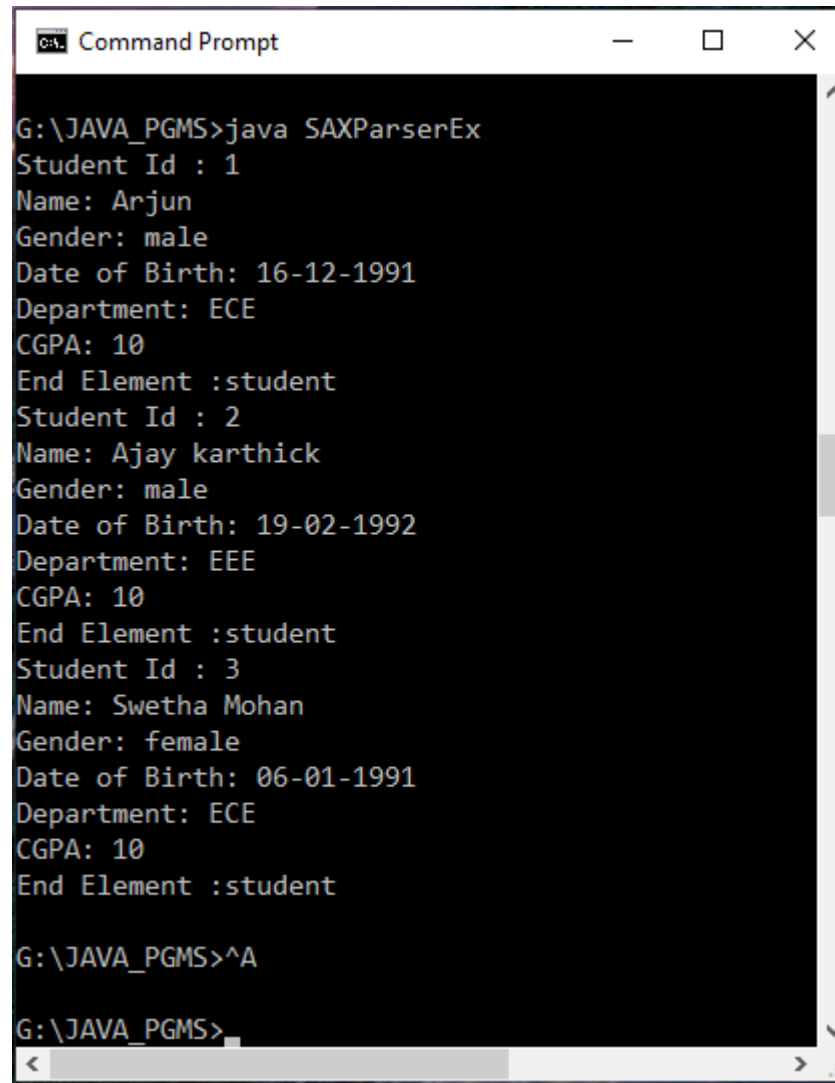
@Override

```
public void characters(char ch[],int start, int length) throws SAXException
{
    if (name)
    {
        System.out.println("Name: " + new String(ch, start, length));
        name = false;
    }
    else if (gender)
    {
        System.out.println("Gender: " + new String(ch, start, length));
        gender = false;
    }
    else if (dob)
    {
        System.out.println("Date of Birth: " + new String(ch, start, length));
        dob = false;
    }
}
```

```
else if (dept)
{
    System.out.println("Department: " + new String(ch, start, length));
    dept = false;
}
else if (cgpa)
{
    System.out.println("CGPA: " + new String(ch, start, length));
    cgpa = false;
}
}
```

Get Data using SAX Parser

Output



```
Command Prompt

G:\JAVA_PGMS>java SAXParserEx
Student Id : 1
Name: Arjun
Gender: male
Date of Birth: 16-12-1991
Department: ECE
CGPA: 10
End Element :student
Student Id : 2
Name: Ajay karthick
Gender: male
Date of Birth: 19-02-1992
Department: EEE
CGPA: 10
End Element :student
Student Id : 3
Name: Swetha Mohan
Gender: female
Date of Birth: 06-01-1991
Department: ECE
CGPA: 10
End Element :student

G:\JAVA_PGMS>^A

G:\JAVA_PGMS>
```

Difference between DOM and SAX XML Parser

- DOM parser loads whole XML document in memory while SAX only loads a small part of the XML file in memory
- DOM parser is faster than SAX because it access whole XML document in memory
- SAX parser in Java is better suitable for large XML file than DOM Parser because it doesn't require much memory
- DOM parser works on Document Object Model while SAX is an event based XML parser