tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

About the Tutorial

XML (Extensible Markup Language) is a very popular simple text-based language that can be used as a mode of communication between different applications. It is considered as a standard means to transport and store data.

JAVA provides excellent support and a rich set of libraries to parse, modify or inquire XML documents.

This tutorial will teach you basic XML concepts and the usage of various types of Java based XML parsers in a simple and intuitive way.

Audience

This tutorial has been prepared for beginners to help them understand the basic-to-advanced concepts related to XML parsing using Java Programming language.

After completing this tutorial, you will find yourself at a moderate level of expertise in XML parsing using Java from where you can take yourself to higher levels of expertise.

Prerequisites

Knowledge of computers is not a prerequisite to follow the contents of this tutorial. This tutorial assumes no background in computers or computer programming, though basic knowledge of computer terminologies will help in understanding the given concepts very easily.

Disclaimer & Copyright

Table of Contents

1. JAVA XML – OVERVIEW

What is XML?

XML is a simple text-based language which was designed to store and transport data in plain text format. It stands for Extensible Markup Language. Following are some of the salient features of XML.

- XML is a markup language.
- XML is a tag based language like HTML.
- XML tags are not predefined like HTML.
- You can define your own tags which is why it is called extensible language.
- XML tags are designed to be self-descriptive.
- XML is W3C Recommendation for data storage and data transfer.

Example

```
<?xml version="1.0"?>
<Class>
   <Name>First</Name>
   <Sections>
      <Section>
         <Name>A</Name>
         <Students>
            <Student>Rohan</Student>
            <Student>Mohan</Student>
            <Student>Sohan</Student>
            <Student>Lalit</Student>
            <Student>Vinay</Student>
         </Students>
      </Section>
      <Section>
```

```
          <Name>B</Name>
          <Students>
             <Student>Robert</Student>
             <Student>Julie</Student>
             <Student>Kalie</Student>
             <Student>Michael</Student>
          </Students>
       </Section>
    </Sections>
</Class>
```

Advantages

Following are the advantages that XML provides:

- **Technology agnostic** - Being plain text, XML is technology independent. It can be used by any technology for data storage and data transfer purpose.
- **Human readable** - XML uses simple text format. It is human readable and understandable.
- **Extensible** - In XML, custom tags can be created and used very easily.
- **Allow Validation** - Using XSD, DTD and XML structures can be validated easily.

Disadvantages

Following are the disadvantages of using XML:

- **Redundant Syntax** - Normally XML files contain a lot of repetitive terms.
- **Verbose** - Being a verbose language, XML file size increases the transmission and storage costs.

2. JAVA XML – PARSERS

XML Parsing refers to going through an XML document in order to access or modify data.

What is XML Parser?

XML Parser provides a way to access or modify data in an XML document. Java provides multiple options to parse XML documents. Following are the various types of parsers which are commonly used to parse XML documents.

- **Dom Parser** - Parses an XML document by loading the complete contents of the document and creating its complete hierarchical tree in memory.
- **SAX Parser** - Parses an XML document on event-based triggers. Does not load the complete document into the memory.
- **JDOM Parser** - Parses an XML document in a similar fashion to DOM parser but in an easier way.
- **StAX Parser** - Parses an XML document in a similar fashion to SAX parser but in a more efficient way.
- **XPath Parser** - Parses an XML document 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. It provides support for DOM, SAX, and JAXP.

There are JAXB and XSLT APIs available to handle XML parsing in object-oriented way. We'll elaborate each parser in detail in the subsequent chapters of this tutorial.

Java DOM Parser

3. JAVA DOM PARSER – OVERVIEW

The Document Object Model (DOM) is an official recommendation of the World Wide Web Consortium (W3C). It defines an interface that enables programs to access and update the style, structure, and contents of XML documents. XML parsers that support DOM implement this interface.

When to Use?

You should use a DOM parser when:

- You need to know a lot about the structure of a document.
- You need to move parts of an XML document around (you might want to sort certain elements, for example).
- You need to use the information in an XML document more than once.

What you get?

When you parse an XML document with a DOM parser, you get back a tree structure that contains all of the elements of your document. The DOM provides a variety of functions you can use to examine the contents and structure of the document.

Advantages

The DOM is a common interface for manipulating document structures. One of its design goals is that Java code written for one DOM-compliant parser should run on any other DOM-compliant parser without having to do any modifications.

DOM Interfaces

The DOM defines several Java interfaces. Here are the most common interfaces:

- **Node** - The base datatype 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

When you are working with DOM, there are several methods you'll use often:

- **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, it returns the attribute with the requested name.

4. JAVA DOM PARSER – PARSE XML DOCUMENT

Steps to Using JDOM

Following are the steps used while parsing a document using JDOM Parser.

- Import XML-related packages.
- Create a SAXBuilder.
- 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();
```

Demo Example

Here is the input XML file that we need to parse:

```
<?xml version="1.0"?>
<class>
   <student rollno="393">
      <firstname>dinkar</firstname>
      <lastname>kad</lastname>
      <nickname>dinkar</nickname>
      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>jasvir</firstname>
      <lastname>singn</lastname>
      <nickname>jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

DomParserDemo.java

```
package com.tutorialspoint.xml;

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 DomParserDemo {
   public static void main(String[] args){

      try {
         File inputFile = new File("input.txt");
         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 roll no : "
                  + eElement.getAttribute("rollno"));
               System.out.println("First Name : "
                  + eElement
                  .getElementsByTagName("firstname")
                  .item(0)
                  .getTextContent());
               System.out.println("Last Name : "
               + eElement
                  .getElementsByTagName("lastname")
                  .item(0)
                  .getTextContent());
               System.out.println("Nick Name : "
               + eElement
                  .getElementsByTagName("nickname")
                  .item(0)
                  .getTextContent());
               System.out.println("Marks : "
               + eElement
                  .getElementsByTagName("marks")
                  .item(0)
                  .getTextContent());
            }
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

This would produce the following result:

```
Root element :class
----------------------------

Current Element :student
Student roll no : 393
First Name : dinkar
Last Name : kad
Nick Name : dinkar
Marks : 85

Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
Nick Name : vinni
Marks : 95

Current Element :student
Student roll no : 593
```

```
First Name : jasvir
Last Name : singn
Nick Name : jazz
Marks : 90
```

5. JAVA JDOM PARSER – QUERY XML DOCUMENT

Demo Example

Here is the input XML file that we need to query:

```
<?xml version="1.0"?>
<cars>
<supercars company="Ferrari">
<carname type="formula one">Ferarri 101</carname>
<carname type="sports car">Ferarri 201</carname>
<carname type="sports car">Ferarri 301</carname>
</supercars>
<supercars company="Lamborgini">
<carname>Lamborgini 001</carname>
<carname>Lamborgini 002</carname>
<carname>Lamborgini 003</carname>
</supercars>
<luxurycars company="Benteley">
<carname>Benteley 1</carname>
<carname>Benteley 2</carname>
<carname>Benteley 3</carname>
</luxurycars>
</cars>
```

QueryXmlFileDemo.java

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;



public class QueryXmlFileDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");

         SAXBuilder saxBuilder = new SAXBuilder();
         Document document = saxBuilder.build(inputFile);

         System.out.println("Root element :"
            + document.getRootElement().getName());

         Element classElement = document.getRootElement();

         List<Element> supercarList = classElement.getChildren("supercars");
         System.out.println("----------------------------");

         for (int temp = 0; temp < supercarList.size(); temp++) {
            Element supercarElement = supercarList.get(temp);
            System.out.println("\nCurrent Element :"
               + supercarElement.getName());
            Attribute attribute =  supercarElement.getAttribute("company");
            System.out.println("company : "
               + attribute.getValue() );
            List<Element> carNameList = supercarElement.getChildren("carname");
            for (int count = 0;
               count < carNameList.size(); count++) {
               Element carElement = carNameList.get(count);
               System.out.print("car name : ");
               System.out.println(carElement.getText());
               System.out.print("car type : ");
               Attribute typeAttribute = carElement.getAttribute("type");
               if(typeAttribute !=null)
                  System.out.println(typeAttribute.getValue());
               else{
                  System.out.println("");
               }
            }
         }
      }catch(JDOMException e){
         e.printStackTrace();
      }catch(IOException ioe){
         ioe.printStackTrace();
      }
   }
}
```

This would produce the following result:

```
Root element :cars
----------------------------

Current Element :supercars
company : Ferrari
car name : Ferarri 101
car type : formula one
car name : Ferarri 201
car type : sports car
car name : Ferarri 301
car type : sports car
```

```
Current Element :supercars
company : Lamborgini
car name : Lamborgini 001
car type :
car name : Lamborgini 002
car type :
car name : Lamborgini 003
car type :
```

6. JAVA JDOM PARSER – CREATE XML DOCUMENT

Demo Example

Here is the XML we need to create:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars><supercars company="Ferrari">
   <carname type="formula one">Ferrari 101</carname>
   <carname type="sports">Ferrari 202</carname>
</supercars></cars>
```

CreateXmlFileDemo.java

```
package com.tutorialspoint.xml;


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 CreateXmlFileDemo {


   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("cars");
doc.appendChild(rootElement);

//  supercars element
Element supercar = doc.createElement("supercars");
rootElement.appendChild(supercar);

// setting attribute to element
Attr attr = doc.createAttribute("company");
attr.setValue("Ferrari");
supercar.setAttributeNode(attr);

// carname element
Element carname = doc.createElement("carname");
Attr attrType = doc.createAttribute("type");
attrType.setValue("formula one");
carname.setAttributeNode(attrType);
carname.appendChild(
doc.createTextNode("Ferrari 101"));
supercar.appendChild(carname);

Element carname1 = doc.createElement("carname");
Attr attrType1 = doc.createAttribute("type");
attrType1.setValue("sports");
carname1.setAttributeNode(attrType1);
carname1.appendChild(
          doc.createTextNode("Ferrari 202"));
          supercar.appendChild(carname1);

          // 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:\\cars.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();
       }
    }
}
```

This would produce the following result:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars><supercars company="Ferrari">
<carname type="formula one">Ferrari 101</carname>
<carname type="sports">Ferrari 202</carname>
</supercars></cars>
```

7. JAVA JDOM PARSER – MODIFY XML DOCUMENT

Demo Example

Here is the input text file we need to modify:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars>
   <supercars company="Ferrari">
      <carname type="formula one">Ferrari 101</carname>
      <carname type="sports">Ferrari 202</carname>
   </supercars>
   <luxurycars company="Benteley">
      <carname>Benteley 1</carname>
      <carname>Benteley 2</carname>
      <carname>Benteley 3</carname>
   </luxurycars>
</cars>
```

ModifyXmlFileDemo.java

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;
```

```
public class ModifyXMLFileDemo {
   public static void main(String[] args) {
      try {
         File inputFile = new File("input.txt");
         SAXBuilder saxBuilder = new SAXBuilder();
         Document document = saxBuilder.build(inputFile);
         Element rootElement = document.getRootElement();

         //get first supercar
         Element supercarElement = rootElement.getChild("supercars");

         // update supercar attribute
         Attribute attribute = supercarElement.getAttribute("company");
         attribute.setValue("Lamborigini");

         // loop the supercar child node
         List<Element> list = supercarElement.getChildren();
         for (int temp = 0; temp < list.size(); temp++) {
            Element carElement = list.get(temp);
            if("Ferrari 101".equals(carElement.getText())){
               carElement.setText("Lamborigini 001");
            }
            if("Ferrari 202".equals(carElement.getText())){
               carElement.setText("Lamborigini 002");
            }
         }

         //get all supercars element
         List<Element> supercarslist = rootElement.getChildren();
         for (int temp = 0; temp < supercarslist.size(); temp++) {
            Element tempElement = supercarslist.get(temp);
            if("luxurycars".equals(tempElement.getName())){
               rootElement.removeContent(tempElement);
            }
         }

         XMLOutputter xmlOutput = new XMLOutputter();

         // display xml
         xmlOutput.setFormat(Format.getPrettyFormat());
         xmlOutput.output(document, System.out);
      } catch (JDOMException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

This would produce the following result:

```
-----------Modified File-----------
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cars>
<supercars company="Lamborigini">
<carname type="formula one">Lamborigini 001</carname>
<carname type="sports">Lamborigini 002</carname>
</supercars></cars>
```

Java SAX Parser

8. JAVA SAX PARSER – OVERVIEW

SAX (Simple API for XML) is an event-based parser for XML documents. Unlike a DOM parser, a SAX parser creates no parse tree. SAX is a streaming interface for XML, which means that applications using SAX receive event notifications about the XML document being processed an element, and attribute, at a time in sequential order starting at the top of the document, and ending with the closing of the ROOT element.

- Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document.
- Tokens are processed in the same order that they appear in the document.
- Reports the application program the nature of tokens that the parser has encountered as they occur.
- The application program provides an "event" handler that must be registered with the parser.
- As the tokens are identified, callback methods in the handler are invoked with the relevant information.

When to Use?

You should use a SAX parser when:

- You can process the XML document in a linear fashion from top to down.
- The document is not deeply nested.
- You are 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.

Disadvantages of SAX

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

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)**