**Java DOM**

*last modified July 8, 2023*

Java DOM tutorial shows how to use Java DOM API to read and write XML documents.

**Advertisements**

**DOM**

*Document Object Model (DOM)* is a standard tree structure, where each node contains one of the components from an XML structure. Element nodes and text nodes are the two most common types of nodes. With DOM functions we can create nodes, remove nodes, change their contents, and traverse the node hierarchy.

**Java DOM**

DOM is part of the Java API for XML processing (JAXP). Java DOM parser traverses the XML file and creates the corresponding DOM objects. These DOM objects are linked together in a tree structure. The parser reads the whole XML structure into the memory.

SAX is an alternative JAXP API to DOM. SAX parsers are event-based; they are faster and require less memory. On the other hand, DOM is easier to use and there are tasks, such as sorting elements, rearranging elements or looking up elements, that are faster with DOM. A DOM parser comes with JDK, so there is no need to download a dependency.

DocumentBuilderFactory enables applications to obtain a parser that produces DOM object trees from XML documents. DocumentBuilder defines the API to obtain DOM Document instances from an XML document or to create a new DOM Document. DocumentTraversal contains methods that create iterators to traverse a node and its children. NodeFilter is used to filter out nodes. NodeIterator is used to go through a set of nodes. TreeWalker is used to navigate a document tree or subtree using the view of the document defined by their whatToShow flags and filter of the document.

**Advertisements**

**EzoicNode types**

The following is a list of some important Node types:

|  |  |
| --- | --- |
| *Node types* | |
| **Type** | **Description** |
| Attr | represents an attribute in an Element object |
| CDATASection | escapes blocks of text containing characters that would otherwise be regarded as markup |
| Comment | represents the content of a comment |
| Document | represents the entire HTML or XML document |
| DocumentFragment | a lightweight or minimal Document object used to represent portions of an XML Document larger than a single node |
| Element | element nodes are basic branches of a DOM tree; most items except text are elements |
| Node | the primary datatype for the entire DOM and each of its elements |
| NodeList | an ordered collection of nodes |
| Text | represents the textual content (called character data in XML) of an Element or Attr |

**XML example files**

We use the following XML files:

**users.xml**

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

<users>

<user id="1">

<firstname>Peter</firstname>

<lastname>Brown</lastname>

<occupation>programmer</occupation>

</user>

<user id="2">

<firstname>Martin</firstname>

<lastname>Smith</lastname>

<occupation>accountant</occupation>

</user>

<user id="3">

<firstname>Lucy</firstname>

<lastname>Gordon</lastname>

<occupation>teacher</occupation>

</user>

</users>

This is the users.xml file.

**continents.xml**

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

<continents>

<europe>

<slovakia>

<capital>

Bratislava

</capital>

<population>

421000

</population>

</slovakia>

<hungary>

<capital>

Budapest

</capital>

<population>

1759000

</population>

</hungary>

<poland>

<capital>

Warsaw

</capital>

<population>

1735000

</population>

</poland>

</europe>

<asia>

<china>

<capital>

Beijing

</capital>

<population>

21700000

</population>

</china>

<vietnam>

<capital>

Hanoi

</capital>

<population>

7500000

</population>

</vietnam>

</asia>

</continents>

This is the continents.xml file.

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.6.0</version>

<configuration>

<mainClass>com.zetcode.JavaReadXmlDomEx</mainClass>

</configuration>

</plugin>

</plugins>

</build>

The examples use the exec-maven-plugin to execute the Java main class from Maven.

## Java DOM reading example

In the following example, we read an XML file with a DOM parser.

**JavaXmlDomReadEx.java**

package com.zetcode;

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;

import java.io.File;

import java.io.IOException;

import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

public class JavaXmlDomReadEx {

public static void main(String argv[]) throws SAXException,

IOException, ParserConfigurationException {

File xmlFile = new File("src/main/resources/users.xml");

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = factory.newDocumentBuilder();

Document doc = dBuilder.parse(xmlFile);

doc.getDocumentElement().normalize();

System.out.println("Root element: " + doc.getDocumentElement().getNodeName());

NodeList nList = doc.getElementsByTagName("user");

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");

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("occupation").item(0);

String occup = 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("Occupation: %s%n", occup);

}

}

}

}

The example parses the users.xml file. It utilizes the names of the tags in the code; for instance: elem.getElementsByTagName("lastname").

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = factory.newDocumentBuilder();

From the DocumentBuilderFactory, we get the DocumentBuilder. DocumentBuilder contains the API to obtain DOM Document instances from an XML document.

Document doc = dBuilder.parse(xmlFile);

The parse method parses the XML file into a Document.

doc.getDocumentElement().normalize();

Normalizing the document helps generate correct results.

System.out.println("Root element:" + doc.getDocumentElement().getNodeName());

We get the root element of the document.

NodeList nList = doc.getElementsByTagName("user");

We get a NodeList of user elements in the document with getElementsByTagName.

for (int i = 0; i < nList.getLength(); i++) {

We go through the list with a for loop.

String uid = elem.getAttribute("id");

We get the element attribute with getAttribute.

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("occupation").item(0);

String occup = node3.getTextContent();

We get the text content of the three subelements of the user element.

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("Occupation: %s%n", occup);

We print the text of the current user to the console.

$ mvn -q exec:java

Root element: users

Current Element: user

User id: 1

First name: Peter

Last name: Brown

Occupation: programmer

Current Element: user

User id: 2

First name: Martin

Last name: Smith

Occupation: accountant

Current Element: user

User id: 3

First name: Lucy

Last name: Gordon

Occupation: teacher