**Java SAX**

*last modified July 8, 2023*

Java SAX tutorial shows how to use Java SAX API to read and validate XML documents.

**SAX**

*SAX (Simple API for XML)* is an event-driven algorithm for parsing XML documents. SAX is an alternative to the Document Object Model (DOM). Where the DOM reads the whole document to operate on XML, SAX parsers read XML node by node, issuing parsing events while making a step through the input stream. SAX processes documents state-independently (the handling of an element does not depend on the elements that came before). SAX parsers are read-only.

SAX parsers 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 SAX parser comes with JDK, so there is no need to dowload a dependency.

**Java SAX parsing example**

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

**resources/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>

We are going to read this XML file.

**com/zetcode/model/User.java**

package com.zetcode.model;

public class User {

int id;

private String firstName;

private String lastName;

private String occupation;

public User() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getOccupation() {

return occupation;

}

public void setOccupation(String occupation) {

this.occupation = occupation;

}

@Override

public String toString() {

StringBuilder builder = new StringBuilder();

builder.append("User{").append("id=").append(id)

.append(", firstName=").append(firstName)

.append(", lastName=").append(lastName)

.append(", occupation=").append(occupation).append("}");

return builder.toString();

}

}

This is the user bean; it will hold data from XML nodes.

**com/zetcode/MyRunner.java**

package com.zetcode;

import com.zetcode.model.User;

import org.xml.sax.SAXException;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import java.io.File;

import java.io.IOException;

import java.nio.file.Paths;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

public class MyRunner {

private SAXParser saxParser = null;

private SAXParser createSaxParser() {

try {

SAXParserFactory factory = SAXParserFactory.newInstance();

factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);

saxParser = factory.newSAXParser();

return saxParser;

} catch (ParserConfigurationException | SAXException ex) {

Logger lgr = Logger.getLogger(MyRunner.class.getName());

lgr.log(Level.SEVERE, ex.getMessage(), ex);

return saxParser;

}

}

public List<User> parseUsers() {

var handler = new MyHandler();

String fileName = "src/main/resources/users.xml";

File xmlDocument = Paths.get(fileName).toFile();

try {

SAXParser parser = createSaxParser();

parser.parse(xmlDocument, handler);

} catch (SAXException | IOException ex) {

Logger lgr = Logger.getLogger(MyRunner.class.getName());

lgr.log(Level.SEVERE, ex.getMessage(), ex);

}

return handler.getUsers();

}

}

MyRunner creates a SAX parser and launches parsing. The parseUsers returns the parsed data in a list of User objects.

SAXParserFactory factory = SAXParserFactory.newInstance();

factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);

saxParser = factory.newSAXParser();

From the SAXParserFactory, we get the SAXParser.

SAXParser parser = createSaxParser();

parser.parse(xmlDocument, handler);

We parse the document with the parse method. The second parameter of the method is the handler object, which contains the event handlers.

**com/zetcode/MyHandler.java**

package com.zetcode;

import com.zetcode.model.User;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

import java.util.ArrayList;

import java.util.List;

public class MyHandler extends DefaultHandler {

private List<User> users = new ArrayList<>();

private User user;

private boolean bfn = false;

private boolean bln = false;

private boolean boc = false;

@Override

public void startElement(String uri, String localName,

String qName, Attributes attributes) {

if ("user".equals(qName)) {

user = new User();

int id = Integer.parseInt(attributes.getValue("id"));

user.setId(id);

}

switch (qName) {

case "firstname" -> bfn = true;

case "lastname" -> bln = true;

case "occupation" -> boc = true;

}

}

@Override

public void characters(char[] ch, int start, int length) {

if (bfn) {

user.setFirstName(new String(ch, start, length));

bfn = false;

}

if (bln) {

user.setLastName(new String(ch, start, length));

bln = false;

}

if (boc) {

user.setOccupation(new String(ch, start, length));

boc = false;

}

}

@Override

public void endElement(String uri, String localName, String qName) {

if ("user".equals(qName)) {

users.add(user);

}

}

public List<User> getUsers() {

return users;

}

}

In the MyHandler class, we have the implementations of the event handlers.

public class MyHandler extends DefaultHandler {

The handler class must extend from the DefaultHandler, where we have the event methods.

@Override

public void startElement(String uri, String localName,

String qName, Attributes attributes) {

if ("user".equals(qName)) {

user = new User();

int id = Integer.parseInt(attributes.getValue("id"));

user.setId(id);

}

switch (qName) {

case "firstname" -> bfn = true;

case "lastname" -> bln = true;

case "occupation" -> boc = true;

}

}

The startElement method is called when the parser starts parsing a new element. We create a new user if the element is <user>. For other types of elements, we set boolean values.

@Override

public void characters(char[] ch, int start, int length) {

if (bfn) {

user.setFirstName(new String(ch, start, length));

bfn = false;

}

if (bln) {

user.setLastName(new String(ch, start, length));

bln = false;

}

if (boc) {

user.setOccupation(new String(ch, start, length));

boc = false;

}

}

The characters method is called when the parser encounters text inside elements. Depending on the boolean variable, we set the user attributes.

@Override

public void endElement(String uri, String localName, String qName) {

if ("user".equals(qName)) {

users.add(user);

}

}

At the end of the <user> element, we add the user object to the list of users.

**com/zetcode/JavaReadXmlSaxEx.java**

package com.zetcode;

import com.zetcode.model.User;

import java.util.List;

public class JavaReadXmlSaxEx {

public static void main(String[] args) {

var runner = new MyRunner();

List<User> lines = runner.parseUsers();

lines.forEach(System.out::println);

}

}

JavaReadXmlSaxEx starts the application. It delegates the parsing tasks to MyRunner. In the end, the retrieved data is printed to the console.