XML parser; concepts and some examples

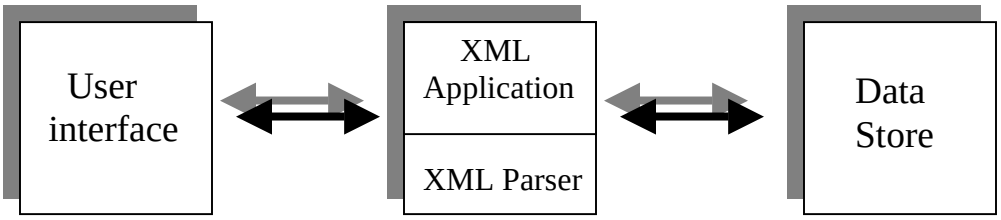
XML Programming

SAX & DOM

By Hamid Mosavi-Porasl

1 XML application architecture

An XML application is typically built around an XML parser. It has an interface to its users, and an interface to some sort of back-end data store.



2 Parser

An XML parser is a piece of code that reads a document and analyzes its structure.

2.1 How to use a parser

Followings step should be taken:

1. create a parser object
2. Pass your XML document to the parser
3. Process the results

Building an XML application is obviously more involved than this, but this is the typical flow of an XML application.

2.2 Kind of parsers

There are several different ways to categorize parses:

- Validating versus non-validating parses
- Parses that support the Document Object Model (DOM)
- Parses that support the Simple API for XML (SAX)
- Parses written in a particular language(java,C++,Perl, etc)

2.3 Validation versus non-validation parses

The XML specification requires all parsers to report errors when they find that a document is not well-formed. Validation, however, is different issue. Validating parsers validate XML documents as they parse them. Non-validating parsers ignore any

validation errors. In other words, if an XML document is well-formed, a non-validating parser doesn't care if the document follows the rules specified in its DTD.

2.4 The Document Object Model (DOM)

The Document Object Model is an official recommendation of the Word Wide Web Consortium(W3C). It defines an interface that enables programs to access and update the style, structure, and contents of XML documents. XML parses that support the DOM implement that interface. The first version of specification, DOM level 1, is available at http://www.w3.org/TR/REC-DOM-level-1.

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

2.5 The simple API for XML (SAX)

The SAX API is an alternate way of working with the contents of XML documents. To see the complete SAX standard, check out www.megginson.com/sax.

2.5.1 What we get from a SAX parser

When we parse an XML document with a SAX parser, the parser generates events at various points in our documents. It's up to us to decide what to do with each of those events.

A SAX parser generates events at the start and end of an element, when it finds characters inside an element, and at several other points. We can write java code that handles each event, and we decide what to do with the implementation we get from parser.

2.6 Why use SAX ? and why use DOM?

In general we should use a DOM parser when:

- We need to know a lot about the structure of a document
- We need to move parts of the document around (e.g. to sort certain elements)
- We need to use information in the document more than once.

We use SAX parser:

- if we only need to extract a few elements from an XML document.
- if we don't have much memory to work with
- if we are only going to use the information in the document once

3 XML parser in different languages

XML parsers and libraries exist for most languages used on the web, including Java, C++, Perl and Python.

4 The Document Object Model (DOM)

The COM is a common interface for manipulating document structures. DOM parser returns a tree structure that represents our entire documents.

4.1 DOM interface

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

- Node: The base datatype of the DOM
- Element: The vast majority of the objects we will 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 three.

4.2 Common DOM methods

When we are working with the DOM, there are several methods we use often:

- ***Document.getDocumentElement()*** returns the root element of the document
- ***Node.getFirstChild()*** and ***Node.getLastChild()*** returns the first or last child of a given Node
- ***Node.getNextSibiling()*** and ***Node.getPreviousSibiling()*** return the next or previous sibling of a given Node.
- ***N***ode.getAttribute(attrName) for a given Node, returns the attribute with the requested name. For example, if we want the Attr object for the attribute named id, use getAttribute("id").

4.3 A DOM application

Among others Apache project demonstrates an example for DOM parser as follow:

```
// JAXP packages
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;
```

```
import java.io.IOException;
import java.io.PrintStream;
import java.io.File;

/**
 * Program to echo a DOM tree using DOM Level 2 interfaces.  Use JAXP to
 * read in and create a DOM tree.  DOM currently does not provide a method
 * to do this.  (This is planned for Level 3.)  See the method main for the
 * three basic steps.  Once the application obtains a DOM Document tree, it
 * dumps out the nodes in the tree and associated node attributes for each
 * node.
 *
 * Notes: Program flags may be used to create possibly non-conformant (but
 * useful) DOM trees.  Program also shows an example of using an
 * ErrorHandler.
 *
 * @author Edwin Goei <edwingo@apache.org>
 */

public class DOMEcho {
   /** Output goes here */
   private PrintStream out;

   /** Indent level */
   private int indent = 0;

   /** Indentation will be in multiples of basicIndent  */
   private final String basicIndent = "  ";

   DOMEcho(PrintStream out) {
      this.out = out;
   }

   /**
    * Echo common attributes of a DOM2 Node and terminate output with an
    * EOL character.
    */
   private void printlnCommon(Node n) {
      out.print(" nodeName=\"" + n.getNodeName() + "\"");

      String val = n.getNamespaceURI();
      if (val != null) {
         out.print(" uri=\"" + val + "\"");
      }

      val = n.getPrefix();
```

```
    if (val != null) {
        out.print(" pre=\"" + val + "\"");
    }

    val = n.getLocalName();
    if (val != null) {
        out.print(" local=\"" + val + "\"");
    }

    val = n.getNodeValue();
    if (val != null) {
        out.print(" nodeValue=");
        if (val.trim().equals("")) {
            // Whitespace
            out.print("[WS]");
        } else {
            out.print("\"" + n.getNodeValue() + "\"");
        }
    }
    out.println();
}

/**
 * Indent to the current level in multiples of basicIndent
 */
private void outputIndentation() {
    for (int i = 0; i < indent; i++) {
        out.print(basicIndent);
    }
}

/**
 * Recursive routine to print out DOM tree nodes
 */
private void echo(Node n) {
    // Indent to the current level before printing anything
    outputIndentation();

    int type = n.getNodeType();
    switch (type) {
    case Node.ATTRIBUTE_NODE:
        out.print("ATTR:");
        printlnCommon(n);
        break;
    case Node.CDATA_SECTION_NODE:
        out.print("CDATA:");
```

```
        printlnCommon(n);
        break;
    case Node.COMMENT_NODE:
        out.print("COMM:");
        printlnCommon(n);
        break;
    case Node.DOCUMENT_FRAGMENT_NODE:
        out.print("DOC_FRAG:");
        printlnCommon(n);
        break;
    case Node.DOCUMENT_NODE:
        out.print("DOC:");
        printlnCommon(n);
        break;
    case Node.DOCUMENT_TYPE_NODE:
        out.print("DOC_TYPE:");
        printlnCommon(n);

        // Print entities if any
        NamedNodeMap nodeMap = ((DocumentType)n).getEntities();
        indent += 2;
        for (int i = 0; i < nodeMap.getLength(); i++) {
            Entity entity = (Entity)nodeMap.item(i);
            echo(entity);
        }
        indent -= 2;
        break;
    case Node.ELEMENT_NODE:
        out.print("ELEM:");
        printlnCommon(n);

        // Print attributes if any.  Note: element attributes are not
        // children of ELEMENT_NODEs but are properties of their
        // associated ELEMENT_NODE.  For this reason, they are printed
        // with 2x the indent level to indicate this.
        NamedNodeMap atts = n.getAttributes();
        indent += 2;
        for (int i = 0; i < atts.getLength(); i++) {
            Node att = atts.item(i);
            echo(att);
        }
        indent -= 2;
        break;
    case Node.ENTITY_NODE:
        out.print("ENT:");
        printlnCommon(n);
```

```
            break;
        case Node.ENTITY_REFERENCE_NODE:
            out.print("ENT_REF:");
            printlnCommon(n);
            break;
        case Node.NOTATION_NODE:
            out.print("NOTATION:");
            printlnCommon(n);
            break;
        case Node.PROCESSING_INSTRUCTION_NODE:
            out.print("PROC_INST:");
            printlnCommon(n);
            break;
        case Node.TEXT_NODE:
            out.print("TEXT:");
            printlnCommon(n);
            break;
        default:
            out.print("UNSUPPORTED NODE: " + type);
            printlnCommon(n);
            break;
        }

        // Print children if any
        indent++;
        for (Node child = n.getFirstChild(); child != null;
             child = child.getNextSibling()) {
            echo(child);
        }
        indent--;
    }

    private static void usage() {
        System.err.println("Usage: DOMEcho [-opts] <filename>");
        System.err.println("       -v = validation");
        System.err.println("       -ws = do not create whitespace nodes");
        System.err.println("       -co[mments] = do not create comment nodes");
        System.err.println("       -cd[ata] = put CDATA into Text nodes");
        System.err.println("       -e[ntity-ref] = create EntityReference nodes");
        System.exit(1);
    }

    public static void main(String[] args) {
        String filename = null;
        boolean validation = false;
```

```
boolean ignoreWhitespace = false;
boolean ignoreComments = false;
boolean putCDATAIntoText = false;
boolean createEntityRefs = false;

for (int i = 0; i < args.length; i++) {
   if (args[i].equals("-v")) {
      validation = true;
   } else if (args[i].equals("-ws")) {
      ignoreWhitespace = true;
   } else if (args[i].startsWith("-co")) {
      ignoreComments = true;
   } else if (args[i].startsWith("-cd")) {
      putCDATAIntoText = true;
   } else if (args[i].startsWith("-e")) {
      createEntityRefs = true;
   } else {
      filename = args[i];

      // Must be last arg
      if (i != args.length - 1) {
         usage();
      }
   }
}
if (filename == null) {
   usage();
}

// Step 1: create a DocumentBuilderFactory and configure it
DocumentBuilderFactory dbf =
   DocumentBuilderFactory.newInstance();

// Optional: set various configuration options
dbf.setValidating(validation);
dbf.setIgnoringComments(ignoreComments);
dbf.setIgnoringElementContentWhitespace(ignoreWhitespace);
dbf.setCoalescing(putCDATAIntoText);
// The opposite of creating entity ref nodes is expanding them inline
dbf.setExpandEntityReferences(!createEntityRefs);

// At this point the DocumentBuilderFactory instance can be saved
// and reused to create any number of DocumentBuilder instances
// with the same configuration options.

// Step 2: create a DocumentBuilder that satisfies the constraints
```

```
    // specified by the DocumentBuilderFactory
    DocumentBuilder db = null;
    try {
        db = dbf.newDocumentBuilder();
    } catch (ParserConfigurationException pce) {
        System.err.println(pce);
        System.exit(1);
    }

    // Set an ErrorHandler before parsing
    db.setErrorHandler(new MyErrorHandler(System.err));

    // Step 3: parse the input file
    Document doc = null;
    try {
        doc = db.parse(new File(filename));
    } catch (SAXException se) {
        System.err.println(se.getMessage());
        System.exit(1);
    } catch (IOException ioe) {
        System.err.println(ioe);
        System.exit(1);
    }

    // Print out the DOM tree
    new DOMEcho(System.out).echo(doc);
  }

  // Error handler to report errors and warnings
  private static class MyErrorHandler implements ErrorHandler {
    /** Error handler output goes here */
    private PrintStream out;

    MyErrorHandler(PrintStream out) {
        this.out = out;
    }

    /**
     * Returns a string describing parse exception details
     */
    private String getParseExceptionInfo(SAXParseException spe) {
        String systemId = spe.getSystemId();
        if (systemId == null) {
            systemId = "null";
        }
        String info = "URI=" + systemId +
```

```
            " Line=" + spe.getLineNumber() +
            ": " + spe.getMessage();
        return info;
    }

    // The following methods are standard SAX ErrorHandler methods.
    // See SAX documentation for more info.

    public void warning(SAXParseException spe) throws SAXException {
        out.println("Warning: " + getParseExceptionInfo(spe));
    }

    public void error(SAXParseException spe) throws SAXException {
        String message = "Error: " + getParseExceptionInfo(spe);
        throw new SAXException(message);
    }

    public void fatalError(SAXParseException spe) throws SAXException {
        String message = "Fatal Error: " + getParseExceptionInfo(spe);
        throw new SAXException(message);
    }
  }
}
```

In short: following steps are necessary:

```
// Step 1: create a DocumentBuilderFactory
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

    // Step 2: create a DocumentBuilder
    DocumentBuilder db = dbf.newDocumentBuilder();

    // Step 3: parse the input file to get a Document object
    Document doc = db.parse(new File(filename));
```

Result of running DOMEcho on following XML file:

```
<?xml version="1.0" encoding="us-ascii" ?>
- <examples>
  - <section xmlns="http://www.example.com/books-r-us">
      <title>Book-Signing Event</title>
    - <signing>
        <author title="Mr." name="Vikram Seth" />
        <book title="A Suitable Boy" price="$2295" />
      </signing>
    </section>
  - <reservation xmlns:html="http://www.w3.org/TR/REC-
      html40">
```

```
      <name html:class="largeSansSerif">Seth, Vikram</name>
      <seat class="Y" html:class="largeMonotype">33B</seat>
      <html:a href="/servlets/ResStatus">Check
        Status</html:a>
      <departure>1997-05-24T19:22:00+5</departure>
    </reservation>
  </examples>
```

is following Node tree:

```
DOC: nodeName="#document"
  ELEM: nodeName="examples"
    TEXT: nodeName="#text" nodeValue=[WS]
    ELEM: nodeName="section"
        ATTR: nodeName="xmlns" nodeValue="http://www.example.com/books-
r-us"
          TEXT: nodeName="#text"
nodeValue="http://www.example.com/books-r-us"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="title"
        TEXT: nodeName="#text" nodeValue="Book-Signing Event"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="signing"
        TEXT: nodeName="#text" nodeValue=[WS]
        ELEM: nodeName="author"
            ATTR: nodeName="name" nodeValue="Vikram Seth"
              TEXT: nodeName="#text" nodeValue="Vikram Seth"
            ATTR: nodeName="title" nodeValue="Mr."
              TEXT: nodeName="#text" nodeValue="Mr."
        TEXT: nodeName="#text" nodeValue=[WS]
        ELEM: nodeName="book"
            ATTR: nodeName="price" nodeValue="$2295"
              TEXT: nodeName="#text" nodeValue="$2295"
            ATTR: nodeName="title" nodeValue="A Suitable Boy"
              TEXT: nodeName="#text" nodeValue="A Suitable Boy"
        TEXT: nodeName="#text" nodeValue=[WS]
      TEXT: nodeName="#text" nodeValue=[WS]
    TEXT: nodeName="#text" nodeValue=[WS]
    ELEM: nodeName="reservation"
        ATTR: nodeName="xmlns:html"
nodeValue="http://www.w3.org/TR/REC-html40"
          TEXT: nodeName="#text" nodeValue="http://www.w3.org/TR/REC-
html40"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="name"
          ATTR: nodeName="html:class" nodeValue="largeSansSerif"
            TEXT: nodeName="#text" nodeValue="largeSansSerif"
        TEXT: nodeName="#text" nodeValue="Seth, Vikram"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="seat"
          ATTR: nodeName="class" nodeValue="Y"
            TEXT: nodeName="#text" nodeValue="Y"
          ATTR: nodeName="html:class" nodeValue="largeMonotype"
            TEXT: nodeName="#text" nodeValue="largeMonotype"
        TEXT: nodeName="#text" nodeValue="33B"
```

```
  TEXT: nodeName="#text" nodeValue=[WS]
  ELEM: nodeName="html:a"
      ATTR: nodeName="href" nodeValue="/servlets/ResStatus"
        TEXT: nodeName="#text" nodeValue="/servlets/ResStatus"
    TEXT: nodeName="#text" nodeValue="Check Status"
  TEXT: nodeName="#text" nodeValue=[WS]
  ELEM: nodeName="departure"
    TEXT: nodeName="#text" nodeValue="1997-05-24T19:22:00+5"
  TEXT: nodeName="#text" nodeValue=[WS]
TEXT: nodeName="#text" nodeValue=[WS]
```

5 The simple API for XML (SAX)

SAX is an event-driven API for parsing XML documents. In our DOM parsing examples, we sent the XML document to the parser, the parser processed the complete document, then we got a Document object representing our document.

In SAX model, we send our XML document to the parser, and the parser notifies us when certain event happen. It's up to us to decide what we want to do with those events; if we ignore them, the information in the event is discarded.

5.1 SAX events

The SAX API defines a number of events. We can write Java code that handles all of the events we care about. If we don't care about a certain type of event, we don't have to write any code at all. Just ignore the event, and the parser will discard it.

5.2 A listing of SAX events

Above you can find a list of SAX events below:

- ***startDocument***: Signals the start of the document
- ***endDocument***: Signals the end of the document
- ***startElement***: Signals the start of an element. The parser fires this event when all of the contents of the opening tag have been processed. That includes the name of the tag and any attributes it might have.
- ***endElement***: Signals the end of an element
- ***characters***: Contains character data similar to a DOM Text Node
- ***ignorableWhitespace***: This event is analogous to the useless DOM nodes we discussed earlier. One benefit of this event is that it's different from the character event; if you don't care about whitespace, you can ignore all whitespace nodes by ignoring this event.
- ***warning, error and fatalError***: These three events indicate parsing errors. We can respond to it as we wish.

- ***setDocumentLocator***: The parser sends this event to allow us store a SAX Locator object can be used to find out exactly where in the document an event occurred.

5.3 SAX Interfaces

The SAX API defines four interfaces for handling events:

- **EntityHandler**
- **DTDHandler**
- **DocumentHandler**
- **ErrorHandler**

All of these interfaces are implemented by ***HandleBase***. Most of time our Java code will extend the HandlerBase class.

5.4 SAX application (Overview)

The structure of SAX application is different from DOM applications:

1. An SAX class extends Handler Base
2. An SAX class has a number of methods, each of which corresponds to a particular SAX event. This simplifies source code because each type of event is completely handled by each method.

5.4.1 Structure of an SAX application

```
public class SAXApplication extends HandlerBase{
        …….
//This method has no arguments
  public void startDocument()
    ….
//Name is the name of the element that juststarted,
//and attrs contains all of the element's attribute
public void startElement(String name, AttributeList attrs)
…..
//ch is an array of characters, start is the position in the array of
//the first character in this event, and length is the number of
//characters for this event.
public void characters(char ch[], int start, int length)
….
public void ignoreableWhitespace(char ch[],int start, int length)
…
// name is the name of the element that just ended.
public void endElement(String name)
…
```

```
public void warning(SAXParseException ex)
…
public void error(SAXParseException ex)
….
Public void fatalError(SAXParseException ex) throws SAXException
…..
}

// Process the command line
public static void main(String argv[]){

if(argv.length==0)
{
        System.out.println("Usage:……..");
        …..
        System.exit(1);
}
SAXApplication s = new SAXApplication();
s.parseURI(argv[0]);
}

// Create a Parser object
SAXParser parser = new SAXParser();
Parser.setDocumentHandler(this);
Parser.setErrorHandler(this);
try{
        parser.parse(url);
}
```