DOM SAX

Marco Ronchetti - ©2005

Java XML parsing

Tree-based vs Event-based API

DOM SAX

Marco Ronchetti - ©2005

Tree-based API

A tree-based API compiles an XML document into an internal tree structure. This makes it possible for an application program to navigate the tree to achieve its objective. The Document Object Model (DOM) working group at the W3C is developing a standard tree-based API for XML.

Event-based API

An event-based API reports parsing events (such as the start and end of elements) to the application using *callbacks*. The application implements and registers event handlers for the different events. Code in the event handlers is designed to achieve the objective of the application. The process is similar (but not identical) to creating and registering event listeners in the Java Delegation Event Model.

what is SAX?

DOM SAX

SAX is a set of interface definitions

For the most part, SAX is a set of interface definitions. They specify one of the ways that application programs can interact with XML documents.

(There are other ways for programs to interact with XML documents as well. Prominent among them is the Document Object Model, or DOM)

SAX is a standard interface for event-based XML parsing, developed collaboratively by the members of the XML-DEV mailing list. SAX 1.0 was released on Monday 11 May 1998, and is free for both commercial and noncommercial use.

The current version is SAX 2.0.1 (released on 29-January 2002)

See http://www.saxproject.org/

JAXP

DOM SAX

JAXP: Java API for XML Processing

This API provides a common interface for creating and using the standard SAX, DOM, and XSLT APIs in Java, regardless of which vendor's implementation is actually being used.

The main JAXP APIs are defined in the javax.xml.parsers package. That package contains two vendor-neutral factory classes: SAXParserFactory and DocumentBuilderFactory that give you a SAXParser and a DocumentBuilder, respectively. The DocumentBuilder, in turn, creates DOM-compliant Document object.

The actual binding to a DOM or SAX engine can be specified using the System properties (but a default is provided).

JAXP – other packages

DOM SAX

org.xml.sax *Defines the basic SAX APIs.*

The "Simple API" for XML (SAX) is the event-driven, serial-access mechanism that does element-by-element processing. The API for this level reads and writes XML to a data repository or the Web.

org.w3c.dom *Defines the Document class (a DOM), as well as classes for all of the components of a DOM.*

The DOM API is generally an easier API to use. It provides a familiar tree structure of objects. You can use the DOM API to manipulate the hierarchy of application objects it encapsulates. The DOM API is ideal for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

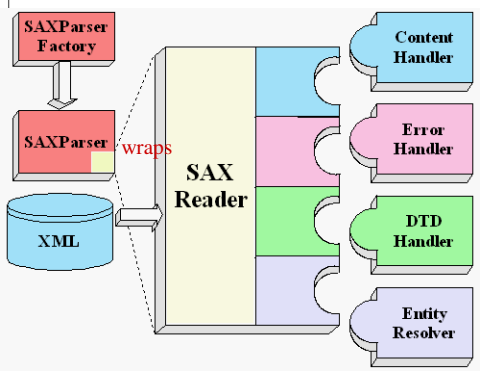
On the other hand, constructing the DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU and memory intensive.

javax.xml.transform *Defines the XSLT APIs that let you transform XML into other forms.*

SAX architecture

DOM SAX


```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true); //optional - default is non-validating
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(File f, DefaultHandler-subclass h)
```



File containing input XML

Default-handler (classe che implementa le callback)

Interfaces implemented by DefaultHandler class

3

SAX packages

DOM SAX

Package	*Description*
org.xml.sax	Defines the SAX interfaces. The name "org.xml" is the package prefix that was settled on by the group that defined the SAX API.
org.xml.sax.ext	Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definitions (DTD) or to see the detailed syntax for a file.
org.xml.sax.helpers	Contains helper classes that make it easier to use SAX -- for example, by defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.
javax.xml.parsers	Defines the SAXParserFactory class which returns the SAXParser. Also defines exception classes for reporting errors.

SAX callbacks

DOM SAX

```
// ----------------------------- ContentHandler methods
void characters(char[] ch, int start, int length)
void startDocument()
void startElement(String name, AttributeList attrs)
void endElement(String name)
void endDocument()
void processingInstruction(String target,String data)
```

SAX example

DOM SAX


```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;

public class CountSax extends DefaultHandler {
  public static void main(String argv[]) throws Exception {
    if (argv.length != 1) {
      System.err.println("Usage: cmd filename");
      System.exit(1);
    }

    // JAXP methods
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser saxParser = factory.newSAXParser();
    saxParser.parse(new File(argv[0]), new CountSax());
  }
```

Obtain a SAX parser,
Parse the file

SAX example 1

DOM SAX


```
package jaxp_demo;
import java.io.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;

public class Echo01
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }
        new Echo01(argv[0]);
    }
```

SAX example 1

DOM SAX


```
public Echo01(String filename) {
  DefaultHandler handler = new MySaxHandler();
  // Use the default (non-validating) parser
  SAXParserFactory factory = SAXParserFactory.newInstance();
  try {
       SAXParser saxParser = factory.newSAXParser();
       saxParser.parse( new File(filename), handler);
  } catch (Throwable t) {
    t.printStackTrace();
  }
  System.exit(0);
 }
}
```

Obtain a SAX parser, Parse the file

SAX example 1

DOM SAX


```
package jaxp_demo;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.*;
import java.io.*;
public class MySaxHandler extends DefaultHandler {
  int indentCount=0;
  String indentString="    ";
  private PrintStream out = System.out;
  private void emit(String s) {
    out.print(s);
    out.flush();
  }
  private void nl() {
    String lineEnd = System.getProperty("line.separator");
    out.print(lineEnd);
  }
  private void indent(){
     String s="";
     for (int i=1;i<=indentCount;i++) s=s+indentString;
     out.print(s);
  }
```

Utility methods

SAX example 1

DOM SAX

Marco Ronchetti - ©2005

```
//=========================================================
// SAX DocumentHandler methods
//=========================================================

  public void startDocument() throws SAXException {
    emit("<?xml version='1.0' encoding='UTF-8'?>");
    nl();
  }

  public void endDocument() throws SAXException {
    nl();
    out.flush();
  }
```


SAX example 1

DOM SAX

Marco Ronchetti - ©2005

```
public void startElement(String namespaceURI,
                         String lName, // local name
                         String qName, // qualified name
                         Attributes attrs) throws SAXException {
    String eName = lName; // element name
    if ("".equals(eName)) eName = qName;
    indent();
    emit("<" + eName);
    if (attrs != null) {
      for (int i = 0; i < attrs.getLength(); i++) {
        String aName = attrs.getLocalName(i); // Attr name
        if ("".equals(aName)) aName = attrs.getQName(i);
        emit(" ");
        emit(aName + "=\"" + attrs.getValue(i) + "\"");
      }
    }
    emit(">");
    nl();
    indentCount++;
  }
```

SAX example 1

DOM SAX


```
public void endElement(String namespaceURI,
                       String sName, // simple name
                       String qName // qualified name
                       ) throws SAXException {
    indentCount--;
    indent();
    emit("</" + qName + ">");
    nl();
  }

  public void characters(char buf[], int offset, int len)
       throws SAXException {
    //String s = new String(buf, offset, len);
    //emit(s);
  }
}
```


SAX references

DOM SAX

A full tutorial with more info and details

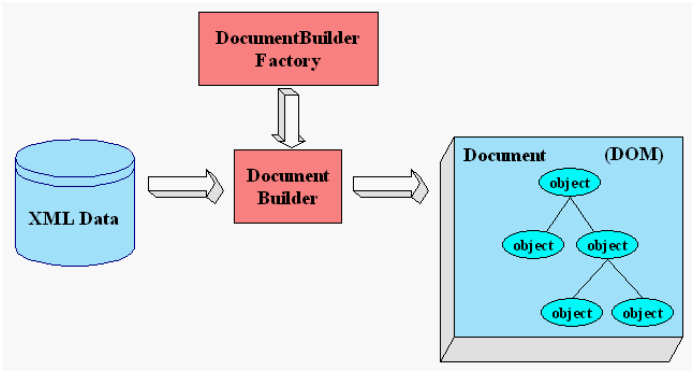
http://java.sun.com/webservices/jaxp/dist/1.1/docs/tutorial/sax/index.html

DOM architecture

DOM SAX

Marco Ronchetti - ©2005

```
DocumentBuilderFactory dbf =  DocumentBuilderFactory.newInstance();
dbf.setValidating(true); // optional – default is non-validating
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(file);
```



J0 17

DOM packages

DOM SAX

Marco Ronchetti - ©2005

Package	*Description*
org.w3c.dom	Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
javax.xml.parsers	Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. The factory that is used to create the builder is determined by the javax.xml.parsers system property, which can be set from the command line or overridden when invoking the newInstance method. This package also defines the ParserConfigurationException class for reporting errors.

J0 18

The Node interface

DOM SAX

Marco Ronchetti - ©2005

public interface Node

The **Node interface** is the primary datatype for the entire DOM. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children. For example, Text nodes may not have children, and adding children to such nodes results in a DOMException being raised.

The attributes **nodeName**, **nodeValue** and **attributes** are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific nodeType (e.g., nodeValue for an Element or attributes for a Comment), this returns null. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

The Document interface

DOM SAX

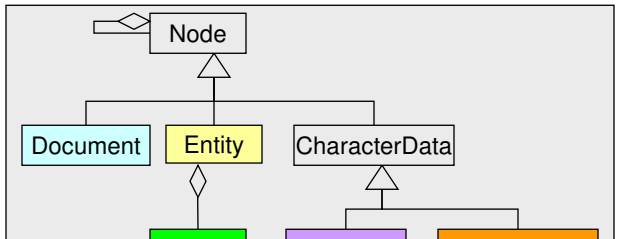
Marco Ronchetti - ©2005

public interface Document extends Node

The **Document interface** represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data. Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have a ownerDocument attribute which associates them with the Document within whose context they were created.

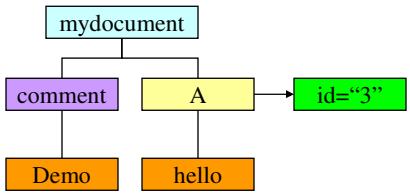
The Node hierarchy

DOM SAX



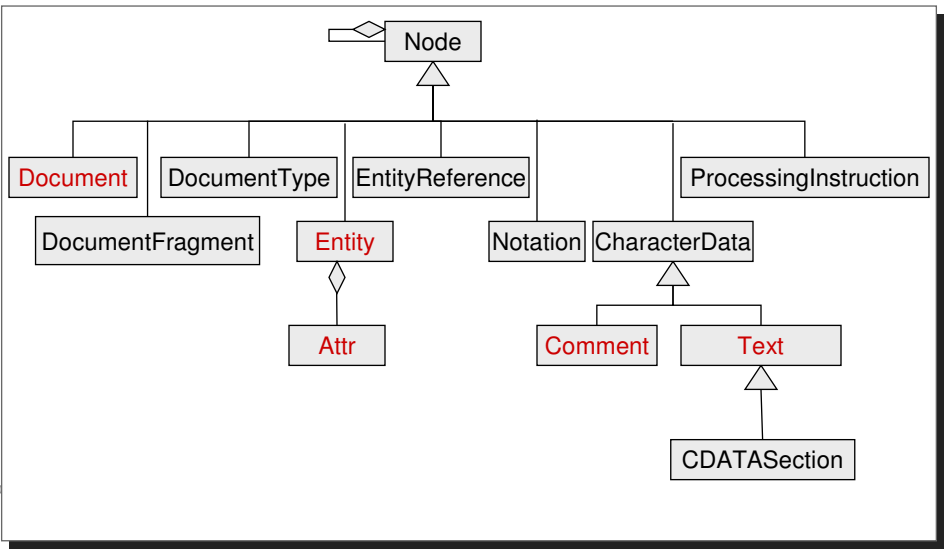
Marco Ronchetti - ©2005


```
<!-- Demo -->
<A id="3">hello</A>
```



The Node hierarchy

DOM SAX



Marco Ronchetti - ©2005

Node: WARNING!

DOM SAX

Marco Ronchetti - ©2005

The implied semantic of this model is WRONG!

You might deduce that a comment might contain another comment, or a document, or any other node!

The integrity is delegated to a series of Node's attributes, that the programmer should check.

Node: main methods

DOM SAX

Marco Ronchetti - ©2005

NAVIGATION

Node getParentNode()	The parent of this node.
NodeList getChildNodes()	A NodeList that contains all children of this node.
Node getFirstChild()	The first child of this node.
Node getLastChild()	The last child of this node.
Node getNextSibling()	The node immediately following this node
Node getPreviousSibling()	The node immediately preceding this node.

The Node interface

DOM SAX

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	"#cdata-section"	content of the CDATA Section	null
Comment	"#comment"	content of the comment	null
Document	"#document"	null	null
DocumentFragment	"#document-fragment"	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	"#text"	content of the text node	null

Node: main methods

DOM SAX

INSPECTION

int getNodeType()
: The type of this node

java.lang.String getNodeName()
: The name of this node, depending on its type; see table.

Short getNodeType()
: A code representing the type of the underlying object.

java.lang.String getNodeValue()
: The value of this node, depending on its type; see the table.

Document getOwnerDocument()
: The Document object associated with this node.

Boolean hasAttributes()
: Returns whether this node (if it is an element) has any attributes.

Boolean hasChildNodes()
: Returns whether this node has any children.

Node: main methods

DOM SAX

Marco Ronchetti - ©2005

EDITING NODES

Node cloneNode(boolean deep)

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.

void setNodeValue(java.lang.String nodeValue)

The value of this node, depending on its type; see the table.

Node: main methods

DOM SAX

Marco Ronchetti - ©2005

EDITING STRUCTURE

Node appendChild(Node newChild)

Adds the node newChild to the end of the list of children of this node.

Node removeChild(Node oldChild)

Removes the child node indicated by oldChild from the list of children, and returns it.

Node replaceChild(Node newChild, Node oldChild)

Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.

Node insertBefore(Node newChild, Node refChild)

Inserts the node newChild before the existing child node refChild.

void normalize()

Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes.

NODE: determining the type

DOM SAX


```
Switch (node.getNodeType()) {
        case Node.ELEMENT_NODE;  …; break;
        case Node.ATTRIBUTE_NODE;  …; break;
        case Node.TEXT_NODE;  …; break;
        case Node.CDATA_SECTION_NODE;  …; break;
        case Node.ENTITY_REFERENCE_NODE;  …; break;
        case Node.PROCESSING_INSTRUCTION;  …; break;
        case Node.COMMENT_NODE;  …; break;
        case Node.DOCUMENT_NODE;  …; break;
        case Node.DOCUMENT_TYPE_NODE;  …; break;
        case Node.DOCUMENT_FRAGMENT_NODE;  …; break;
        case Node.NOTATION_NODE;  …; break;
        default: throw (new Exception());
}
```


DOM example

DOM SAX


```
import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;   // parser uses SAX methods to build DOM object
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

public class CountDom {
  public static void main(String[ ] arg) throws Exception {
    if (arg.length != 1) {
      System.err.println("Usage: cmd filename (file must exist)");
      System.exit(1);
    }
     Node node = readFile(new File(arg[0]));
     System.out.println(arg + " elementCount: " + getElementCount(node));
    }
  }
```

DOM example

DOM SAX

Marco Ronchetti - ©2005

```
public static Document readFile(File file) throws Exception {
   Document doc;
   try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setValidating(true);
    DocumentBuilder db = dbf.newDocumentBuilder();
    doc = db.parse(file);
    return doc;
   } catch (SAXParseException ex) {
    throw (ex);
   } catch (SAXException ex) {
    Exception x = ex.getException();   // get underlying Exception
    throw ((x == null) ? ex : x);
   }
  }
```

Parse File,
Return Document

DOM example

DOM SAX

Marco Ronchetti - ©2005

```
public static int getElementCount(Node node) {
  if (null == node)      return 0;
  int sum = 0;
  boolean isElement = (node.getNodeType() == Node.ELEMENT_NODE);
  if (isElement)  sum = 1;
  NodeList children = node.getChildNodes();
  if (null == children)      return sum;

  for (int i = 0; i < children.getLength(); i++) {
    sum += getElementCount(children.item(i));   // recursive call
  }
  return sum;
 }
}
```

use DOM methods to count elements:
for each subtree if the root is an Element,
 set sum to 1, else to 0;
 add element count of all children of the root to sun

Alternatives to DOM

"Build a better mousetrap, and the world will beat a path to your door."
--Emerson

Alternatives to DOM

JDOM: Java DOM (see http://www.jdom.org).

The standard DOM is a very simple data structure that intermixes text nodes, element nodes, processing instruction nodes, CDATA nodes, entity references, and several other kinds of nodes. That makes it difficult to work with in practice, because you are always sifting through collections of nodes, discarding the ones you don't need into order to process the ones you are interested in. JDOM, on the other hand, creates a tree of *objects* from an XML structure. The resulting tree is much easier to use, and it can be created from an XML structure without a compilation step.

DOM4J: DOM for Java (see http://www.dom4j.org/)

***dom4j* is an easy to use, open source library for working with XML, XPath and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX and JAXP.**

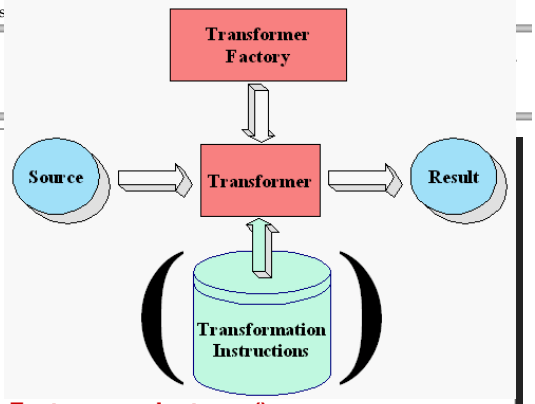
Transformations

DOM SAX

Using XSLT from Java

Marco Ronchetti - ©2005

TrAX




```
TransformerFactory tf = TransformerFactory .newInstance();
StreamSource xslSS=new StreamSource("source.xsl");
StreamSource xmlSS=new StreamSource("source.xml");
Transformer t=tf.newTrasformer(xslSS);
t.transform(xmlSS,new StreamResult(new
        FileOutputStream("out.html");
```

```
java –Djavax.xml.transform.TransformerFactory=
org.apache.xalan.processor.TrasformerFactoryImpl MyClass
```

Marco Ronchetti - ©2005

xml.transform packages

DOM SAX

Package	*Description*
javax.xml.transform	Defines the TransformerFactory and Transformer classes, which you use to get a object capable of doing transformations. After creating a transformer object, you invoke its transform() method, providing it with an input (source) and output (result).
javax.xml.transform.dom	Classes to create input (source) and output (result) objects from a DOM.
javax.xml.transform.sax	Classes to create input (source) from a SAX parser and output (result) objects from a SAX event handler.
javax.xml.transform.stream	Classes to create input (source) and output (result) objects from an I/O stream.

TrAX main classes

DOM SAX

javax.xml.transform.Transformer
transform(Source xmls, Result output)

javax.xml.transform.sax.SAXResult implements Result
javax.xml.transform.sax.SAXSource implements Source

javax.xml.transform.stream.StreamResult implements Result
javax.xml.transform.stream.StreamSource implements Source

javax.xml.transform.dom.DOMResult implements Result
javax.xml.transform. dom.DOMSource implements Source

Other Java-XML APIs

Marco Ronchetti - ©2005

Java Architecture for XML Binding (JAXB) provides a convenient way to bind an XML schema to a representation in Java code.

See also:

- JAX-WS
- JAX-SWA
- JAX- RPC
- SAAJ
- XML –Digital Signatures
- ecc.

20