Processing of structured documents

Spring 2003, Part 5
Helena Ahonen-Myka

Programming interfaces

- event-based approach: SAX
- tree-walking approach: DOM

XML processing model

- XML processor is used to read XML documents and provide access to their content and structure
- XML processor works for some application
- the XML specification defines which information the processor should provide to the application

Parsing

- input: an XML document
- basic task: is the document well-formed?
- validating parsers additionally: is the document valid?

Parsing

- parsers produce data structures, which other tools and applications can use
- two kind of APIs: tree-based and event-based

Tree-based API

- compiles an XML document into an internal tree structure
- allows an application to navigate the tree
- Document Object Model (DOM) is a tree-based API for XML and HTML documents

Event-based API

- reports parsing events (such as start and end of elements) directly to the application
- the application implements handlers to deal with the different events
- Simple API for XML (SAX)

7

Example

```
<?xml version="1.0">
 <doc>
    <para>Hello, world!</para>
 </doc>
```

- Events:

```
start document
start element: doc
start element: para
characters: Hello, world!
end element: para
end element: doc
end document
```

8

Example (cont.)

- an application handles these events just as it would handle events from a graphical user interface (mouse clicks, etc) as the events occur
- no need to cache the entire document in memory or secondary storage

9

Tree-based vs. event-based

- tree-based APIs are useful for a wide range of applications, but they may need a lot of resources (if the document is large)
- some applications may need to build their own tree structures, and it is very inefficient to build a parse tree only to map it to another tree

10

Tree-based vs. event-based

- an event-based API is simpler, lower-level access to an XML document
- as document is processed sequentially, one can parse documents much larger than the available system memory
- own data structures can be constructed using own callback event handlers

11

SAX

- A parser is needed
  - e.g. Apache Xerces: http://xml.apache.org
- and SAX classes
  - www.saxproject.org
  - often the SAX classes come bundled to the parser distribution

12

Starting a SAX parser

```
import org.xml.sax.XMLReader;
import org.apache.xerces.parsers.SAXParser;

XMLReader parser = new SAXParser();
parser.parse(uri);
```

13

Content handlers

- In order to let the application do something useful with XML data as it is being parsed, we must register **handlers** with the SAX parser
- handler is a set of callbacks: application code can be run at important events within a document's parsing

14

Core handler interfaces in SAX

- org.xml.sax.ContentHandler
- org.xml.sax.ErrorHandler
- org.xml.sax.DTDHandler
- org.xml.sax.EntityResolver

15

Custom application classes

- custom application classes that perform specific actions within the parsing process can implement each of the core interfaces
- implementation classes can be registered with the parser with the methods setContentHandler(), etc.

16

Example: content handlers

```
class MyContentHandler implements ContentHandler {

  public void startDocument() {
      System.out.println("Parsing begins…");
  }

  public void  endDocument() {
      System.out.println("...Parsing ends.");
  }
```

17

Element handlers

```
public void startElement (String namespaceURI,
                         String localName,
                         String rawName,
                         Attributes atts)  {

System.out.print("startElement: " +  localName);
if (!namespaceURI.equals("")) {
   System.out.println(" in namespace " + namespaceURI +
                      " (" + rawname + ")");
} else {
   System.out.println(" has no associated namespace"); }

for (int i=0; i<atts.getLength(); i++) {
    System.out.println("  Attribute: " + atts.getLocalName(i)
                       +  "=" + atts.getValue(i)); }}
```

18

endElement

```
public void endElement(String namespaceURI,
                       String localName,
                       String rawName) {

    System.out.println("endElement: " + localName + "\n");
}
```

19

Character data

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

    String s = new String(ch, start, end);
    System.out.println("characters: " + s);
}
```

- parser may return all contiguous character data at once, or split the data up into multiple method invocations

20

Processing instructions

- XML documents may contain **processing instructions** (PIs)
- a processing instruction tells an application to perform some specific task
- form: <?target instructions?>

21

Handlers for PIs

```
public void processingInstruction (String target,
                                   String data){

    System.out.println("PI: Target:" + target
                       + " and Data:" + data);
}
```

- Application could receive instructions and set variables or execute methods to perform application-specific processing

22

Validation

- some parsers are validating, some non-validating
- some parsers can do both
- SAX method to turn validation on:

```
parser.setFeature ("http://xml.org/sax/features/validation",
                   true);
```

23

Ignorable whitespace

- parsers have to pass all the characters to the application
- validating parser can tell which whitespace characters can be ignored
- for a non-validating parser, all whitespace is just characters
- content handler:

```
public void  ignorableWhitespace (char ch[], int start,
                                  int end) { … }
```

24

Document Object Model (DOM)

- In transforming documents, random access to a document is needed
- SAX cannot look backward or forward
- difficult to locate siblings and children
- DOM: access to any part of the tree
  - www.w3.org/DOM/

25

DOM

- Level 1: navigation of content within a document (W3C recommendation)
- Level 2: modules and options for specific content models, such as XML, HTML, and CSS; events (W3C recommendation)
- Level 3: document loading and saving; access of schemas (W3C working draft)

26

Some requirements

- All document content, including elements and attributes, will be programmatically accessible and manipulable
- Navigation from any element to any other element will be possible
- There will be a way to add, remove, and change elements/attributes in the document structure

27

DOM

- XML documents are treated as a tree of nodes
- two views:
  - higher level XML DOM objects
  - "everything is a node"

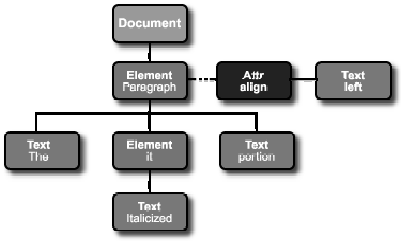
28

Higher level XML DOM objects

- Element
- Attr
- Text
- CDATAsection
- EntityReference
- Entity
- Document
- ...

29

<paragraph align="left"> The <it>Italicized</it> portion. </paragraph>



30

"Everything is a node"

- Node
  - a single node in the document tree
- NodeList
  - a list of node objects (e.g. children)
- NamedNodeMap
  - allows access by name (e.g. to the collection of attributes)
  - names are unique in the map

31

DOM Java bindings

- DOM is language-neutral
- bindings for several languages, e.g. for Java
  - Interfaces and classes that define and implement the DOM
  - bindings often included in the parser implementations (the parser generates a DOM tree)

32

Parsing using a DOM parser

```
Import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;

DOMParser  parser = new DOMParser();
parser.parse(uri);
```

33

Output tree

- the entire document is parsed and added into the output tree, before any processing takes place
- handle: org.w3c.dom.Document object = one level above the root element in the document

```
parser.parse(uri);
Document doc = parser.getDocument();
```

34

Example: printing a document

- at each node of the document tree, the current node is processed and then recursively the children of that node
- starting from the document node

```
parser.parse(uri);
Document doc = parser.getDocument();
printTree(doc);
```

35

Printing a document: general view

```
Private static void printTree(Node node) {
    switch (node.getNodeType()) {
        case Node.DOCUMENT_NODE:
          // Print the contents of the Document object
         break;

        case Node.ELEMENT_NODE:
          // Print the element and its attributes
         break;

        case Node.TEXT_NODE:  ...
```

36

...the Document node

```
Case Node.DOCUMENT_NODE:
    System.out.println("<xml version=\"1.0\">\n");
    Document doc = (Document)node;
    printTree(doc.getDocumentElement());
    break;
```


... elements

```
Case Node.ELEMENT_NODE:
    String name= node.getNodeName();
    System.out.print("<" + name);
    // Print out attributes…  (see next slide…)
    System.out.println(">");

    // recurse on each child:
    NodeList children = node.getChildNodes();
    if (children != null) {
       for (int i=0; i<children.getLength(); i++) {
          printTree(children.item(i));
       }
    }
    System.out.println("</" + name + ">");
```


... and their attributes

```
case Node.ELEMENT_NODE:
    String name = node.getNodeName();
    System.out.print("<" + name);
    NamedNodeMap attributes = node.getAttributes();
    for (int i=0; i<attributes.getLength(); i++) {
        Node current = attributes.item(i);
        System.out.print(" " + current.getNodeName() +
                "=\"" + current.getNodeValue() +
                "\"");
    }
    System.out.println(">");
    …
```


...textual nodes

```
case Node.TEXT_NODE:
case Node.CDATA_SECTION_NODE:
    System.out.print(node.getNodeValue());
  break;
```


Document interface methods

- Attr createAttribute(String name)
- Element createElement(String tagName)
- Text createTextNode(String data)
- Element getDocumentElement()
- Element getElementById(String elementID)
- NodeList getElementsByTagName(String tagName)

Node interface methods

- NamedNodeMap getAttributes()
- NodeList getChildNodes()
- String getLocalName()
- String getNodeName()
- String getNodeValue()
- Node getParentNode()
- short getNodeType()
- appendChild(Node newChild), removeChild(Node oldChild), insertBefore(Node newChild, Node refChild)

Node types

- static short ATTRIBUTE_NODE
- static short ELEMENT_NODE
- static short TEXT_NODE
- static short DOCUMENT_NODE
- static short COMMENT_NODE
- ...

43

Node interface

- some methods are not allowed for all the nodetypes, e.g.
  - appendChild for text nodes
- some methods return "null" for some nodetypes, e.g.
  - nodeValue for Element
  - attributes for Comment

44

NodeList interface methods

- int getLength()
  - gets the number of nodes in this list
- Node item(int index)
  - gets the item at the specified index value in the collection

45

NamedNodeMap interface methods

- Int getLength()
  - returns the number of nodes in this map
- Node getNamedItem(String name)
  - gets a node indicated by name
- Node item(int index)
  - gets an item in the map by index

46

Element interface methods

- String getAttribute()
  - returns an attribute's value
- String getTagName()
  - return an element's name
- NodeList getElementsByTagName(String tagName)
  - returns the descendant elements with given name
- removeAttribute(String name)
  - removes an element's attribute
- setAttribute(String name, String value)
  - set an attribute's value

47

Attr interface methods

- String getName()
  - gets the name of this attribute
- Element getOwnerElement()
  - gets the Element node to which this attribute is attached
- String getValue()
  - gets the value of the attribute as a string

48

NodeList and NamedNodeMap objects are "live"

- changes to the underlying document structure are reflected in all relevant NodeList and NamedNodeMap objects
- for instance,
  - if a DOM user gets a NodeList object containing the children of an Element object,
  - and then he/she subsequently adds more children to that element (or removes/modifies), those changes are automatically reflected in the NodeList

49

Common sources of problems with DOM programming

- whitespace
  - text nodes containing whitespace may appear in places where the programmer does not expect them to be
- value of an element
  - <someTag>something</someTag>
  - value of the element is actually the value of the text node that is a child
  - text can also be distributed into several text nodes: all child nodes have to be checked to collect the entire text content (value) of the element

50

Some efficiency etc. issues

- if the document tree can be kept in memory, updates of small parts of the tree are efficient
  - assumed that we do not want to serialize the tree (= write it back to a text file) after each update
- complicated transformations are easier with DOM than with SAX
  - or should one use XSLT?

51