SAX & DOM

CPS 196.3
Introduction to Database Systems

Announcements

- Homework #3 assigned today
  - Due next Friday (November 7)
  - If you wish, you can probably do this homework on your own machine
    - Downloads for QuiP and other Java-based XML tools are free
- Course project milestone 2 due on November 12

Review

- XML: tree (or graph)-structured data
- DTD: simple schema for XML
  - Well-formed XML: syntactically correct
  - Valid XML: well-formed and conforms to a DTD
- XPath: path expression language for XML
  - An XPath expression selects a list of nodes in an XML document
  - Used in other languages
- XQuery: SQL-like query language for XML
  - FLWR expression, quantified expression, sort-by expression, etc.
- XSLT: stylesheet language for XML, in XML
  - Transforms input XML by applying template rules recursively on the structure of input XML

SAX & DOM

- Both are API’s for XML processing
- SAX (Simple API for XML)
  - Started out as a Java API, but now exists for other languages too
- DOM (Document Object Model)
  - Language-neutral API with implementations in Java, C++, etc.
  - JAXP (Java API for XML Processing)
    - Bundled with standard JDK
    - Includes SAX, DOM parsers and XSLT transformers

SAX processing model

- Serial access
  - XML document is processed as a stream
  - Only one look at the data
  - Cannot go back to an early portion of the document
- Event-driven
  - A parser generates events as it goes through the document (e.g., start of the document, end of an element, etc.)
  - Application defines event handlers that get invoked when events are generated

SAX events

Most frequently used events:

- startDocument
- endDocument
- startElement
- endElement
- character

Whitespace may come up as characters or ignorable whitespace, depending on whether a DTD is present

Warning: The parser may generate multiple characters events for one piece of text
A simple SAX example

- Print out text contents of title elements

```java
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.*;

public class SaxExample extends DefaultHandler {
    public static void main(String[] argv) throws Exception {
        String fileName = argv[0];
        // Create a SAX parser:
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();
        // Parse the document with this event handler:
        DefaultHandler handler = new SaxExample();
        saxParser.parse(new File(fileName), handler);
    }
}
```

```java
private StringBuffer titleStringBuffer = null;
public void startElement(String uri, String localName, String qName, Attributes attributes) {
    if (qName.equals("title"))
        titleStringBuffer = new StringBuffer();
}

public void endElement(String uri, String localName, String qName) {
    if (qName.equals("title")) {
        System.out.println(titleStringBuffer.toString);
        titleStringBuffer = null;
    }
}

public void characters(char[] ch, int start, int length) {
    if (titleStringBuffer != null)
        titleStringBuffer.append(ch, start, length);
}
```

Warning: This code does not handle data with title/title pattern

Only relevant when namespace is involved

Assuming no namespace processing, qname is tag name

Assuming no namespace

Assuming no namespace, qname is tag name

Assuming no namespace, qname is tag name

Assuming no namespace, qname is tag name

A common mistake

What is wrong with the following?

```java
private String titleString = null;
public void endElement(String uri, String localName, String qName) {
    // Print the last chunk of characters seen before </title>:
    if (qName.equals("title"))
        System.out.println(titleString);
}

public void characters(char[] ch, int start, int length) {
    titleString = new String(ch, start, length);
}
```

- Cannot handle the case where other tags appear within a title element
- It is possible that characters() are called multiple times for one piece of text; this code only prints out the last part

A more complex SAX example

- Print out the text contents of top-level section titles in books, i.e., //book/section/title
- Old code would print out all titles, e.g., //book/title, //book/section/title
- For simplicity, assume that if we have the pattern //book/section/title//book/section/title, we print the higher-level title element
- Idea: maintain as state the path from the root

```java
private ArrayList path = new ArrayList();
private int pathLengthWhenOutputIsActivated;
```

```java
public void startElement(String uri, String localName, String qName, Attributes attributes) {
    path.add(qName); // Maintain the path.
    if (path.size() >= 3 &&
        ((String)(path.get(path.size()-1))).equals("title") &&
        ((String)(path.get(path.size()-2))).equals("section") &&
        ((String)(path.get(path.size()-3))).equals("book")) {
        // path matches //book/section/title:
        if (titleStringBuffer == null) {
            pathLengthWhenOutputIsActivated = path.size();
            titleStringBuffer = new StringBuffer();
        }
    }
}
```

```java
public void endElement(String uri, String localName, String qName) {
    if (titleStringBuffer != null &&
        path.size() == pathLengthWhenOutputIsActivated) {
        // Closing the element that activated output buffering:
        System.out.println(titleStringBuffer.toString());
        titleStringBuffer = null;
    }
    path.remove(path.size()-1); // Maintain the path.
}

public void characters(char[] ch, int start, int length) {
    if (titleStringBuffer != null)
        titleStringBuffer.append(ch, start, length);
}
```

This check prevents premature output in case that title has subelements

Would it work if we change this check to qName.equals("title")?
DOM processing model

- XML is parsed by a parser and converted into an in-memory DOM tree
- DOM API allows an application to
  - Construct a DOM tree from an XML document
  - Traverse and read a DOM tree
  - Construct a new, empty DOM tree from scratch
  - Modify an existing DOM tree
  - Copy subtrees from one DOM tree to another
  etc.

DOM example

```xml
<?xml version="1.0">
<!DOCTYPE ...>
<bibliography>
<book ISBN="ISBN-10" price="80.00">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
...</book>
...</book>
...
</bibliography>
```

```
Document
  DocumentType
  Element
  Attr
  Text
  Element
  Attr
  Text
  Element
  Attr
  Text
  ...  
  Text
  Element
  Attr
```

Whitespace between tags are also parsed as Text

Constructing DOM from XML

```java
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;

public class DomExample {
    public static void main(String[] args) throws Exception {
        // Parse input XML into a DOM Document:
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(new File(args[0]));
        // Use the default (identity) Transformer to print the DOM Document:
        TransformerFactory tFactory = TransformerFactory.newInstance();
        Transformer transformer = tFactory.newTransformer();
        transformer.transform(new DOMSource(document), new StreamResult(System.out));
    }
}
```

Node interface

n.getNodeType() returns the type of Node
n.getChildNodes() returns a NodeList containing Node n's children
  - For example, subelements are children of an Element; DocumentType is a child of the Document

d.getDocumentElement() returns the root Element of Document d
e.getNodeName() returns the tag name of Element e
e.getElementsByTagName() returns a NamedNodeMap (hash table) containing the attributes of Element e
  - Attributes are not considered children!
a.getNodeName() returns the name of Attr a
t.getNodeValue() returns the content of Text t
For convenience: n.getParentNode(), n.getPreviousSibling(), n.getNextSibling(), n.getOwnerDocument(), etc.

Traversing DOM

- Compute the string value of an XML node
  ```java
  public static String convertNodeToString(Node n) {
      // String value of a Text Node is just its content:
      if (n.getNodeType() == Node.TEXT_NODE)
          return n.getNodeValue();
      // String value of a Node of another type is the concatenation
      // of its children's string values:
      String text = "";
      NodeList children = n.getChildNodes();
      for (int i=0; i<children.getLength(); i++) {
          Node child = children.item(i);
          text = text + convertNodeToString(child);
      }
      return text;
  }
  ```

- In general, you can use an XSLT Transformer instead

DOM Node's

- A DOM tree is made up of Node's
- Most frequently used types of Node's:
  - Document: root of the DOM tree
    - Not the same as the root element of XML
  - DocumentType: corresponds to the DOCTYPE declaration in an XML document
  - Element: corresponds to an XML element
  - Attr: corresponds to an attribute of an XML element
  - Text: corresponds to chunk of text
Traversing DOM (cont’d)

- Print out text contents of `title` elements
  ```java
  public static void outputTitle(Node n) {
    if (n.getNodeType() == Node.ELEMENT_NODE &&
        n.getNodeName().equals("title")) {
      // This is a title Element; output it:
      System.out.println(convertNodeToString(n));
    } else {
      // Recurse down the tree and look for titles to output:
      NodeList children = n.getChildNodes();
      for (int i=0; i<children.getLength(); i++) {
        Node child = children.item(i);
        outputTitle(child);
      }
    }
  }
  ```
- How would you print out just //book/section/title?
  - Use `getParentNode()` to check for section parent and book grandparent.

Constructing DOM from scratch

- Construct a DOM Document showing all titles as follows:
  ```xml
  <result>
    <title text="title1"/>
    <title text="title2"/>
  </result>
  ```
  ```java
  public static Document newDocWithTitles(Document inputDoc) throws Exception {
    // Create a new Document:
    DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
    DocumentBuilder builder=factory.newDocumentBuilder();
    Document newDoc=builder.newDocument();
    // Create the root Element:
    Element newElement=newDoc.createElement("result");
    newDoc.appendChild(newElement);
    // Add titles:
    addTitlesToNewDoc(newDoc, inputDoc);
    return newDoc;
  }
  ```

Copying subtrees in DOM

- Construct a DOM Document showing all title elements from the input XML
  ```java
  public static Document newDocWithTitles2(Document inputDoc) throws Exception {
    // Add titles:
    addTitlesToNewDoc2(newDoc, inputDoc);
  }
  ```
  ```java
  public static void addTitlesToNewDoc2(Document newDoc, Node n)
  throws Exception {
    if (n.getNodeType() == Node.ELEMENT_NODE &&
        n.getNodeName().equals("title")) {
      Node newNode = newDoc.importNode(n, true);
      newDoc.getDocumentElement().appendChild(newNode);
    } else {
    }
  }
  ```
- A Document can import (copy) a Node from another element.
  - The second argument specifies whether to copy recursively or not.

Summary: SAX versus DOM

- **SAX**
  - Because of one-pass processing, a SAX parser is fast, consumes very little memory, and scales to very large XML documents
  - Applications are responsible for keeping necessary state in memory, and are therefore more difficult to code
- **DOM**
  - Because the input XML needs to be converted to an in-memory DOM tree representation, a DOM parser consumes more memory, and does not scale as well as a SAX parser
  - Applications are easier to develop because of the powerful DOM interface