SAX & DOM

CPS 116

Introduction to Database Systems

Announcements (October 27)

- Homework #3 due next Tuesday
- Project milestone #2 due Nov. 10

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`
- `characters`
  - Whenever the parser has processed a chunk of character data (without generating other kinds of events)
  - Warning: The parser may generate multiple `characters` events for one piece of text

```
<?xml version="1.0"> ⟶ startDocument
<bibliography> ⟶ startElement
  <book ISBN="ISBN-10" price="80.00"> ⟶ startElement
    <title>Foundations of Databases</title>
          ↳ startElement          ↓          ↓
  </book> ⟶ endElement   characters  endElement
</bibliography> ⟶ endElement
                ↳ endDocument
```

Whitespace may come up as `characters` or `ignorableWhitespace`, depending on whether a DTD is present

A simple SAX example

- Print out text contents of `title` elements

```
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);
    return;
  }
  ...
```

A simple SAX example (cont'd)

```
  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);
  }
```

Only relevant when namespace is involved

Assuming no namespace processing, qname is tag name

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

A common mistake

What is wrong with the following?

```
  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);
  }
```

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

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