XML PARSER - SAX

UNIT-II

What is SAX?

- Expands as Simple API for XML
- SAX framework defines event listeners, or *handlers*
- *Handlers are* written by developers for parsing documents with a known structure
- Handlers contain numerous methods that are invoked in response to these events

Contd…

- Handlers registered with SAX framework to receive events
- Events include start of document, start of element, end of element, and so on
- Once the handlers are defined and registered, an input source can be specified and parsing can begin

Where and When SAX used

- To pull out the text from a document
- Look for attributes of specific tags
- To do some of the work using a tool such as XSLT
- Traditional "properties" are replaced with XML due to uniformity and richness of expression

- To carry out all these, instead of writing our own standalone program, SAX parser allows to do that

- SAX is also a validating parser

SAX vs DOM

- DOM in-memory tree structure, takes lots of memory
- SAX is event – based, read document and notify about interested data found
- SAX faster, not needed to load entire doc, so saves network cost
- SAX allows to build our own object model of doc efficiently
- AX contains simple API and smaller than DOM Implementation

SAX2 Packages

- SAX 2.0 API comprised of two standard packages and one extension package
- Standard packages are org.xml.sax and org.xml.helpers
- org.xml.sax package - contains basic classes, interfaces, and exceptions needed for parsing documents
- org.xml.sax.helpers package - contains additional classes that can simplify some of your coding and make it more portable (adapter classes, factory classes)
- The org.xml.sax.ext package is an extension that is not shipped with all implementations (contains two handler interfaces for capturing declaration and lexical events)

org.xml.sax Package

- Attributes - Interface for a list of XML attributes
- ContentHandler - Receives notification of the logical content of a document
- DTDHandler - Receives notification of basic DTD-related events
- EntityResolver - Basic interface for resolving entities
- ErrorHandler - Basic interface for SAX error handlers
- Locator - Interface for associating a SAX event with a document location

Contd…

- Parser - Deprecated. This interface has been replaced by the SAX2 XMLReader interface, which includes namespace support
- XMLFilter - Interface for an XML filter
- XMLReader - Interface for reading an XML document using callbacks
- **Classes**
- InputSource - single input source for an XML entity

Contd…

- **Exceptions**

- SAXException, SAXParseException - Encapsulates a general SAX error or warning / Encapsulates parse error

- SAXNotRecognizedException, SAXNotSupportedException - Exception classes for an unrecognized identifier and unsupported operation

Steps in parsing using SAX

- Write a *content handler* creating a Java class that implements the ContentHandler interface in the org.xml.sax Package
- Convenience adapters are available to simplify this
- Register content handler with a SAX XMLReader, set up the input source, and start the parser
- Methods in your content handler will be called when the parser encounters elements, text, and other data

Sample i/p and o/p

Input:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<fiction>
<book author=”Herman Melville”>Moby Dick</book>
</fiction>
```

Output:

```
start document
start element: fiction
start element: book (including attributes)
characters: Moby Dick
end element: book
end element: fiction
end document
```

Namespace URI, local name and qualified name

- <?xml version="1.0"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> <xs:element name="note">
- <xs:complexType>
- <xs:sequence>
- <xs:element name="to" type="xs:string"/>
- <xs:element name="from" type="xs:string"/>
- <xs:element name="heading" type="xs:string"/>
- <xs:element name="body" type="xs:string"/>
- </xs:sequence>
- </xs:complexType>
- </xs:element>
- </xs:schema>

- A namespace is the logical container in which an element is defined
- Example: XML Schema namespace (with uri: http://www.w3.org/2001/XMLSchema)

Contd…

- XML parser may be either namespace-aware or not
- But documents using namespaces need to be parsed by namespace-aware parsers
- Namespaces are defined so that
  - a) they can be catalogued by the parser
  - b) elements with the same name in different namespaces can exist in the same document unambiguously-defined

Contd…

- Prefix
  - A prefix is the short-hand key used to refer to a namespace
  - Example, xs is used to refer to the XML Schema namespace

- Local Name
  - An element in a document has a name as it is defined in the namespace
  - Example: schema, element, complexType, sequence in the given example
  - Local names can be ambiguous if you have multiple namespaces referenced in your document

Contd...

- **Qualified Name (qName)**

  - A qualified name consists of the prefix for the namespace followed by a :, followed by the element's local name
  - Example: xs:schema, xs:element, xs:complexType, xs:sequence, and xs:element
  - qnames are unambiguous, and can be processed by the parser and validated

Contd…

- SAX2 adds XML Namespace support, which is required for higher-level standards
- SAX2 XMLReader interface supports Namespace processing in its default state
- Many XML readers allow Namespace processing to be modified or disabled
- Explicitly required in older version of SAX

Example:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setNamespaceAware(false);
    SAXParser parser = factory.newSAXParser();
    ……………………….
```

Detail Example Code

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

public class SAXDemo extends DefaultHandler {
public void startDocument() {
System.out.println("***Start of Document***");
}

public void endDocument() {
System.out.println("***End of Document***");
}
```

Contd…

```
public void characters(char[] ch, int start, int length) {
System.out.println(new String(ch, start, length).trim());
}
public void endElement(String namespaceURI, String
   localName,
String qName) throws SAXException {
System.out.println(“ ” + qName + “ ”);
}
```

Contd…

```
public void startElement(String uri, String localName,
String qName, Attributes attributes) {
System.out.println(“ ” + qName + “ ”);
int n = attributes.getLength();
for (int i=0; i<n; i+=1) {
System.out.println(“ “ + attributes.getQName(i) +
“=’” + attributes.getValue(i) + “‘“);
}
}
```

Contd...

```
public static void main(String args[]) throws Exception {
if (args.length != 1) {
System.err.println(“Usage: java SAXDemo <xml-file>”);
System.exit(1);
}
SAXDemo handler = new SAXDemo();
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
parser.parse(new File(args[0]), handler);
}
}
```

Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<fiction>
<book author="Herman Melville">Moby Dick</book>
</fiction>
***Start of Document***
 fiction
 book
 author Herman Melville
 Moby Dick
 book
 fiction
***End of Document***
```