Simple API for XML (SAX)

Asst. Prof. Dr. Kanda Runapongsa
(krunapon@kku.ac.th)
Dept. of Computer Engineering
Khon Kaen University

Topics
- Parsing and application
- SAX event model
- SAX event handlers
- Apache Xerces
- JAXP
- When to use SAX

SAX API
- SAX API is based on an event-driven processing model where
  - The data elements are interpreted on a sequential basis
  - The callbacks are called based on selected constructs
- It uses a sequential read-only approach and does not support random access to the XML elements
**SAX Features**

- **Event-driven**
  - You provide event handlers
- **Fast and lightweight**
  - Document does not have to be entirely in memory
- **Sequential read access only**
- **One-time access**
- **Does not support modification of document**

**SAX Operational Model**

**SAX Programming**

- **Collection of Java interfaces and classes**
  - Package org.xml.sax
- **Interfaces**
  - Parser
    - XMLReader
  - Event handlers
    - ContentHandler
Topics
- Parsing and application
- SAX event model
- **SAX event handlers**
- Apache Xerces
- JAXP
- When to use SAX

SAX Event Handlers
- Interfaces
  - ContentHandler
  - ErrorHandler
  - DTDHandler
  - EntityResolver
  - Attributes
- Class
  - DefaultHandler

SAX Event Handlers

```
Interfaces
- ContentHandler
- ErrorHandler
- DTDHandler
- EntityResolver
- Attributes

Class
- DefaultHandler
```
**SAX Event-based**

```
<?xml version="1.0"?>
<nation>
  <name>Thailand</name>
  <location>Asia</location>
</nation>
```

- `startDocument`
- `startElement: nation`
- `startElement: name`
- `Characters: Thailand`
- `endElement: name`
- `startElement: location`
- `characters: Asia`
- `endElement: location`
- `endElement: nation`
- `endDocument`

---

**ContentHandler Interface**

```java
public interface ContentHandler{
    void startDocument() throws SAXException;
    void endDocument() throws SAXException;
    void startElement(String namespace, String name, String qName, Attributes atts) throws SAXException;
    void endElement(String namespace, String name, String qName) throws SAXException;
    void characters(char[] ch, int start, int length) throws SAXException;
    void ignorableWhiteSpace(char[] ch, int start, int length) throws SAXException;
    void processingInstruction(String target, String data) throws SAXException;
    void setDocumentLocator(Locator locator);
    void startPrefixMapping(String prefix, String uri) throws SAXException;
    void endPrefixMapping(String prefix) throws SAXException;
    void skippedEntity(String name) throws SAXException;
}
```

---

**DefaultHandler Class**

- **Helper class**
- **Implements**
  - ContentHandler
  - ErrorHandler
  - DTDHandler
  - EntityResolver
- **Just subclass and override the methods you are interested in**
ContentHandler and DefaultHandler

- There are eleven methods declared in the ContentHandler interface.
- Few SAX programs actually use all eleven methods.
- SAX includes the org.xml.sax.helpers.DefaultHandler class that implements the ContentHandler interface.
- By extending DefaultHandler, we only have to override methods we actually care about.

Most Frequently Used Methods (1/2)

- `startDocument()` This method is called only once at the start of the XML document.
  ```java
  public void startDocument()
  ```
- `endDocument()` This method is called when the parser reaches the end of the XML document.
  ```java
  public void endDocument()
  ```
- `characters()` This method is called from character data residing inside an element.
  ```java
  public void characters(char[] text, int start, int length)
  ```

Most Frequently Used Methods (2/2)

- `startElement()` This method is called every time a new opening tag of an element is encountered (for example, `<element>`).
  ```java
  public void startElement(String namespaceURI, String localName, String qualifiedName, Attributes atts)
  ```
- `endElement()` This method is called when an element ends (for example, `</element>`).
  ```java
  public void endElement(String namespaceURI, String localName, String qualifiedName)
  ```
What the ContentHandler Does not Tell You

- The type of quotes that surround attributes
- Whether empty elements are represented as
  - `<name>` or `<name>`
- Whether an attribute was specified in the instance document or defaulted in from the DTD or schema

Handling Attributes

- Attributes are not reported through separate callbacks
- Instead an Attributes object containing all the attributes of an element is passed to the `startElement()` method

Attributes Interface

```java
public interface Attributes {
    public abstract int getLength();
    public abstract int getIndex(String qName);
    public abstract int getIndex(String namespace, String name);
    public abstract String getLocalName(int index);
    public abstract String getQName(int index);
    public abstract String getType(int index);
    public abstract String getType(String qName);
    public abstract String getType(String namespace, String name);
    public abstract String getValue(String qName);
    public abstract String getValue(String namespace, String name);
    public abstract String getValue(int index);
    public abstract String getURI(int index);
}
```
Handling Attributes Example

```java
public void startElement(String namespaceURI, String localName, String qName, Attributes atts)
{
    int length = atts.getLength();
    if (length > 0)
    {
        System.out.println("Element <"+qName+"> has the following attributes");
        for (int i = 0; i < length; i++)
        {
            System.out.println(" " + atts.getQName(i) + " = " + atts.getValue(i));
        }
    }
}
```

ErrorHandler Interface

```java
public interface ErrorHandler{
    void error(SAXParserException e) throws SAXException
    void fatalError(SAXParseException e) throws SAXException
    void warning(SAXParseException e) throws SAXException
}
```

ErrorHandler Example

```java
public class SAXWithErrorHandler extends DefaultHandler {
    public void fatalError(SAXParseException exception)
    {
        System.err.println("FATAL ERROR! " + exception.getMessage());
    }
    ...
    public static void main(String[] args)
    throws SAXException, IOException {
        parser.setContentHandler(handler);
        parser.setErrorHandler(handler);
        ...
    }
```
### Topics
- Parsing and application
- SAX event model
- SAX event handlers
- **Apache Xerces**
- JAXP
- When to use SAX

### SAX Programming Procedures (Using Xerces)

```
XMLReader parser = XMLReaderFactory.createXMLReader();
myHandler handler = new myHandler();
parser.parse(args[0]);
parser.setContentHandler(handler);
```

SAX parser calls methods on the event handler

```
public void startDocument() {
    System.out.println("XML Document Start");
}
```

### XMLReader Instance
- Concrete implementation instance "bound" to XMLReader interface
- Has to be created before parsing
- Gets created by using static method of createXMLReader() method of helper class XMLReaderFactory
**XMLReader Example 1**

```java
XMLReader parser = null;
try {
    // Get SAX parser instance reading
    // org.xml.sax.driver system property
    parser = XMLReaderFactory.createXMLReader();
    // Parse the document
} catch (SAXException ex) {
    // Couldn’t create XMLReader
    // either because org.xml.sax.driver system
    // property
    // was not set or set incorrectly
}
```

**XMLReader Example 2**

```java
XMLReader parser = null;
try {
    // Create an instance of Apache’s Xerces
    // SAX parser
    parser = XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
    // Parse the document
} catch (SAXException ex) {
    // Couldn’t create XMLReader maybe because
    // org.apache.xerces.parsers.SAXParser class is
    // not in classpath
}
```

**Setting Features**

- `setFeature(String, boolean)` method of `XMLReader` interface
  - The first argument is feature name
  - The second argument is the boolean value. Set to `true` if we want to activate that feature

**Sample Feature Names**

- **Validation**: [http://xml.org/sax/features/validation](http://xml.org/sax/features/validation)
- **Namespace Awareness**: [http://xml.org/sax/features/namespaces](http://xml.org/sax/features/namespaces)
- **Schema Validation**: [http://apache.org/xml/features/validation/schema](http://apache.org/xml/features/validation/schema)
Simple API for XML (SAX)

Parse Methods
- void parse(String uri) throws SAXException, IOException
- void parse(InputSource source) throws SAXException, IOException

Example:
XMLReader parser =
XMLReaderFactory.createXMLReader()
parser.parse("c:/employee.xml")

XMLReader Example3
XMLReader parser = null;
try {
    // Create an instance of Apache's Xerces // SAX parser
    parser =
    XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
    // Parse the document
    parser.parse("c:/employee.xml");
} catch (SAXException ex) {
    // Catch any exception during parsing
}

Simple SAX Example: Parser
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.IOException;
public class SimpleSAX {
    public static void main(String[] args) {
        XMLReader parser = null;
        try {
            // Create XML (non-validating) parser
            parser =
            XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
            /*  Create event handler */
            myContentHandler handler = new
            myContentHandler();
            parser.setContentHandler(handler);
            // Call parsing method
            parser.parse(args[0]);
        } catch (SAXException ex) {
            System.err.println(ex.getMessage());
        } catch (Exception ex) {
            System.err.println(ex.getMessage());
        }}}}
Simple API for XML (SAX)

Event Handler Example: myContentHandler

```java
class myContentHandler extends DefaultHandler {

    // ContentHandler methods
    public void startDocument() {
        System.out.println("XML Document START");
    }
    public void endDocument() {
        System.out.println("XML Document END");
    }
    public void startElement(String namespace, String name, String qName, Attributes atts) {
        System.out.println("<" + qName + ">");
    }
    public void endElement(String namespace, String name, String qName) {
        System.out.println("</" + qName + ">");
    }
    public void characters(char[] chars, int start, int length) {
        System.out.println(new String(chars, start, length));
    }
}
```

Receiving Documents

- The parser invokes `startDocument()` as soon as it begins parsing a new document before it invokes any other methods in ContentHandler.
- It calls `endDocument()` after it's finished parsing the document and will not report any further content from that document.

Receiving Elements

- When the parser encounters a start tag, it calls the `startElement()` method.
- When the parser encounters an end tag, it calls the `endElement()` method.
- When the parser encounters an empty-element tag, it calls the `startElement()` method and then the `endElement()` method.
Receiving Characters
- When the parser reads #PCDATA, it passes this text to the characters() method as an array of chars
- Parsers are allowed to break up character data any way desired
- Character data are in Unicode regardless of encoding scheme specified in XML documents

Topics
- Parsing and application
- SAX event model
- SAX event handlers
- Apache Xerces
- JAXP
- When to use SAX

Processing XML with JAXP SAX
- Major steps for parsing using JAXP:
  - Getting Factory and Parser classes to perform XML parsing
  - Setting options such as namespaces, validation, and features
  - Creating a defaultHandler implementation class
Getting a Factory Class

- Obtain a factory class using the SAXParserFactory’s static newInstance() method
  - SAXParserFactory factory = SAXParserFactory.newInstance();

Getting and Using a SAXParser Class

- Obtain the SAX parser class from the factory by calling the newSAXParser() static method
  - SAXParser parser = factory.newSAXParser();

- Parse the XML data by calling the parse method
  - parser.parse("methodCall.xml", handler);
  - The second argument is the handler with type ContentHandler

JAXP/SAX Code Sample

```java
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class JAXPEcho extends DefaultHandler {
    // Use an instance of ourselves as the SAX event handler
    DefaultHandler handler = new JAXPEcho();
    // Use the default (non-validating) parser
    SAXParserFactory factory = SAXParserFactory.newInstance();
    // Parse the input
    SAXParser saxParser = factory.newSAXParser();
    saxParser.parse(new File("employee.xml"), handler);
    ...
}
```
Benefits of SAX

- It is very simple
- It is very fast
- Useful when custom data structures are needed to model the XML document
- Can parse files of any size without impacting memory usage
- Can be used to gather a subset of a document’s information

Drawbacks of SAX

- SAX provides read-only access
- Developers need to model the documents themselves
  - Design the model
  - Write appropriate event handlers
  - Create data structures in programs
- No random access to documents

When to Use SAX

- When we want to only read XML documents
- When we need to process very large XML documents
- When we have a limited amount of memory
- When we want to have a very fast and efficient parser
Summary

- SAX API is based on an event-driven processing model
- SAX Parser usually extends DefaultHandler to override only interested methods
- When using Xerces, use XMLReader
- When using JAXP, use SAXParserFactory and SAXParser
- SAX is fast and simple, but it provides read-only access in a sequential fashion

References

- "Java and XML" written by Brett McLaughlin, O'Reilly
- "Processing XML with Java" written Elliotte Rusty Harold
  http://www.cafeconleche.org/books/xmljava/
- Sang Shin “SAX (Simple API for XML)”,
  http://www.javapassion.com/xml