

# Lecture 0 (1 hour)

---

# SAX

## (Simple API for XML)

Sang Shin  
Java™ Technology Evangelist  
[sang.shin@sun.com](mailto:sang.shin@sun.com)

(You can use this material in any way you want,  
but if you can drop me an email when you do,  
that will be greatly appreciated.)

# Topics



- Parsing and application
- SAX historical background
- SAX event model
- Error handling
- Apache Xerces
- JAXP 1.1
- When to use SAX
- Demo and code review

# Parsing and Application



- Parsing
  - ◆ Well-formed'ness checking & Validating
  - ◆ Reading
- Application
  - ◆ Manipulating
  - ◆ Creating
  - ◆ Writing and Sending

# SAX Historical Background

---

- Simple API for XML
- Started as community-driven project
  - ◆ xml-dev mailing list
- Originally designed as Java API
  - ◆ Others (C++, Python, Perl) are now supported
- SAX 2
  - ◆ Namespaces

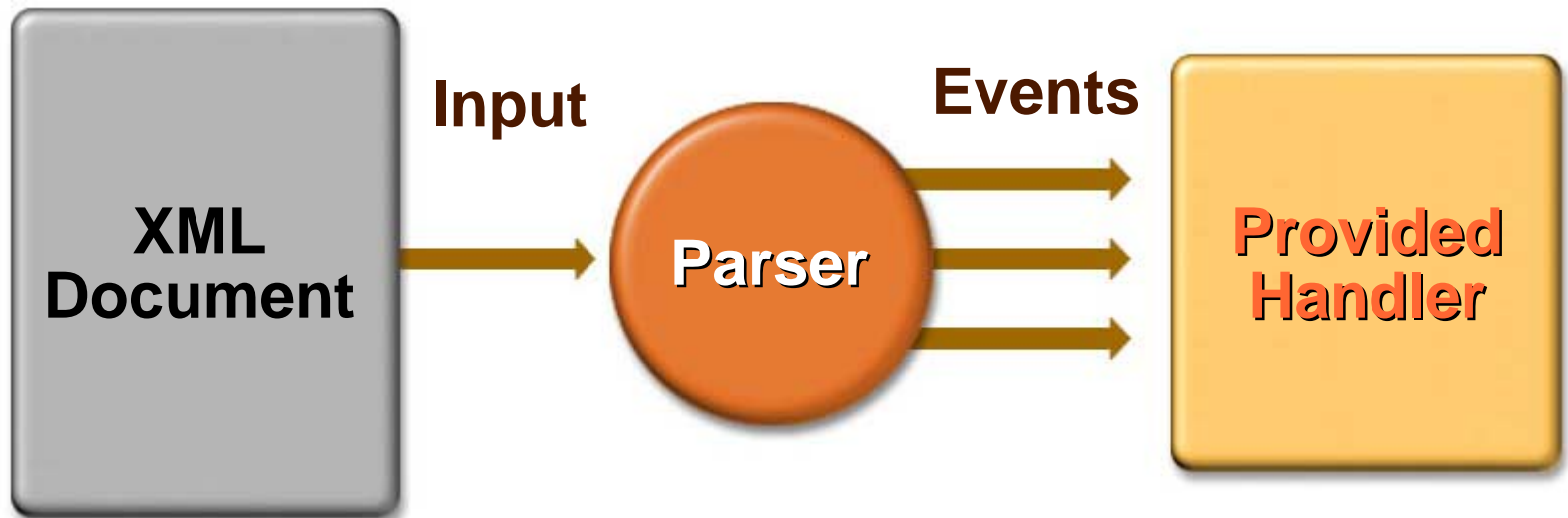
# SAX Features

---

- **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
  - ◆ `org.xml.sax` package
- Interfaces
  - ◆ Parser
    - *XMLReader*
  - ◆ Event handlers
    - *ContentHandler*

# Java Interface Type Primer



- Define methods
- Separation of how from what
  - ◆ Does say **what** the methods do
  - ◆ Does **NOT** say **how** the methods are implemented
- Java interface type is implemented by concrete implementation class
- Concrete implementation class is “bound” to interface during runtime

# Java Interface Type Primer

---

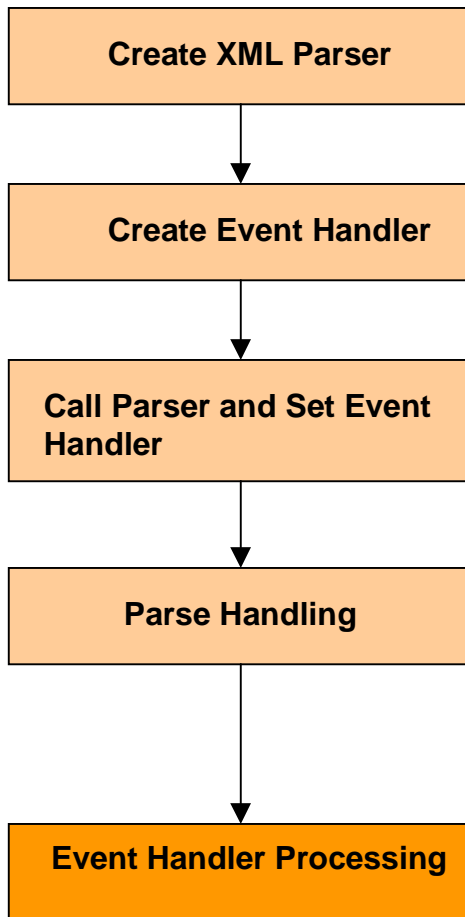
- Key feature of Java
- Most of Java APIs are defined as Java interface types
  - ◆ Multiple implementations
  - ◆ Implementations can evolve without affecting applications
- Supports polymorphism
- De-couple applications from underlying services

# Java Interface Type Primer

- Example - XMLReader

```
public interface XMLReader{  
    ...  
    public void parse (InputSource input)  
        throws IOException, SAXException;  
    public void parse (String systemId)  
        throws IOException, SAXException;  
    ...  
    public void setFeature (String name, boolean value)  
        throws SAXNotRecognizedException,  
        SAXNotSupportedException;  
    ...  
}
```

# SAX Programming Procedures



```
XMLReader parser =  
XMLReaderFactory.createXMLReader();
```

```
myHandler handler = new myHandler();
```

```
parser.parse(args[0]);
```

SAX parser calls methods on the event handler

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



# XMLReader Interface

---

- Represents SAX parser
  - ◆ SAX2 parser implementation has to implement this interface
- Methods for
  - ◆ Registering event handlers
  - ◆ Initiating parsing
  - ◆ Configuring features and properties
    - Validation on and off

# XMLReader Interface

```
public interface XMLReader{
    public boolean getFeature (String name)
        throws SAXNotRecognizedException, SAXNotSupportedException;
    public void setFeature (String name, boolean value)
        throws SAXNotRecognizedException, SAXNotSupportedException;
    public Object getProperty (String name)
        throws SAXNotRecognizedException, SAXNotSupportedException;
    public void setProperty (String name, Object value)
        throws SAXNotRecognizedException, SAXNotSupportedException;
    public void setEntityResolver (EntityResolver resolver);
    public EntityResolver getEntityResolver ();
    public void setDTDHandler (DTDHandler handler);
    public DTDHandler getDTDHandler ();
    public void setContentHandler (ContentHandler handler);
    public ContentHandler getContentHandler ();
    public void setErrorHandler (ErrorHandler handler);
    public ErrorHandler getErrorHandler ();
    public void parse (InputStream input) throws IOException, SAXException;
    public void parse (String systemId) throws IOException, SAXException;
}
```

# 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

---

```
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

```
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
}
```

# XMLReader Example 3

```
XMLReader parser = null;
try {
    // Create an instance of Apache's Crimson SAX parser
    parser =
    XMLReaderFactory.createXMLReader("org.apache.crimson.parser.XMLReaderImpl");

    // Parse the document

}catch(SAXException ex){
    // Couldn't create XMLReader maybe because
    // org.apache.xerces.parsers.SAXParser class is
    // not in classpath
}
```

# XMLReader Example 4

```
XMLReader parser = null;
try {
    // Create an instance of Apache's Crimson SAX parser
    // without using XMLReaderFactory helper class
    parser =
        (XMLReader)Class.forName("org.apache.crimson.parser.
XMLReaderImpl").newInstance();

    // Parse the document

}catch(SAXException ex){
}
```

# Setting Features

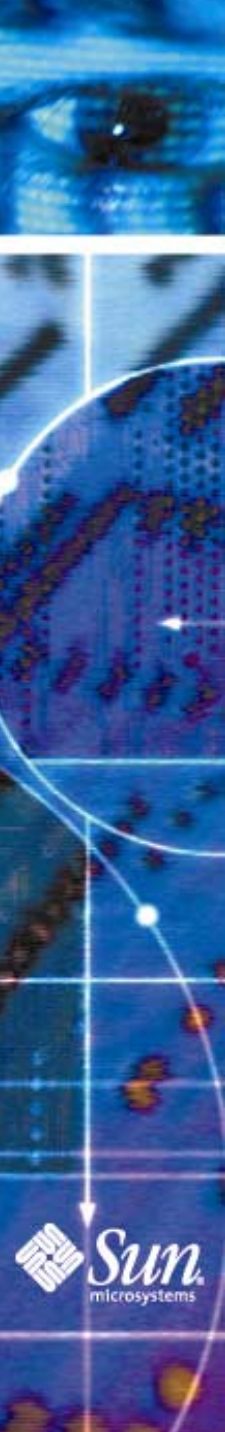
---

- *setFeature(String, boolean)* method of *XMLReader* interface
- Features
  - ◆ General features
  - ◆ SAX features
  - ◆ DOM features

# Features

---

- General Features
  - ◆ <http://xml.org/sax/features/validation>
  - ◆ <http://xml.org/sax/features/namespace>
  - ◆ <http://apache.org/xml/features/validation/schema>
  - ◆ <http://apache.org/xml/features/continue-after-fatal-error>
- SAX Features
  - ◆ <http://xml.org/sax/features/namespace-prefixes>



# Example

---

```
XMLReader parser = null;
try {
    // Create an instance of Apache's Crimson SAX parser
    parser = XMLReaderFactory.createXMLReader();

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

    // Parse the document

} catch(SAXException ex){
}
```

# Parse Methods

---

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

# Example

---

```
XMLReader parser = null;
try {
    parser = XMLReaderFactory.createXMLReader();

    // Parse the document
    parser.parse("http://www slashdot.org/slashdot.xml");

    // Capture SAX events

} catch (SAXException ex) {
    // exception occurs maybe because document
    // is malformed
}
```

# Example

---

```
XMLReader parser = null;
try {
    parser = XMLReaderFactory.createXMLReader();

    // Parse the document in File URI form
    parser.parse("file:/tmp/people.xml");

    // Capture SAX events

} catch (SAXException ex) {
    // exception occurs maybe because document
    // is malformed
}
```

# InputSource Objects

---

- Encapsulates information about an input source as a single object
- Passed to the *XMLReader.parse()* method
- May include the following:
  - ◆ Public identifier
  - ◆ System identifier
  - ◆ A byte stream
  - ◆ A character stream

# Parser Behavior on InputSource

---

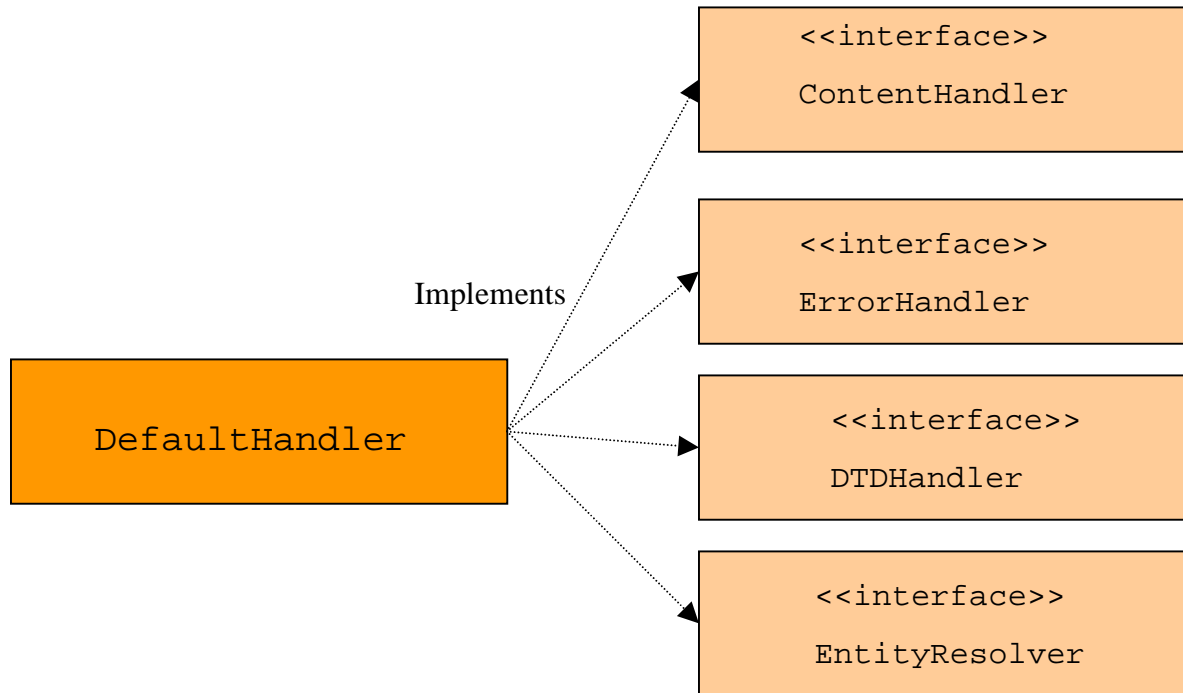
- If character stream is available, parser will read that stream directly
- Else if byte stream is available, parser will read byte stream
- Else open URI connection to resource specified in System identifier

# SAX Event Handlers



- Interfaces
  - ◆ ContentHandler
  - ◆ ErrorHandler
  - ◆ DTDHandler
  - ◆ EntityResolver
  - ◆ Attributes
- Class
  - ◆ DefaultHandler

# SAX Event Handlers



# ContentHandler Interface

```
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;
}
```

# Simple SAX Example: Parser

---

```
XMLReader parser = null;
try {
    // Create XML (non-validating) parser
    parser = XMLReaderFactory.createXMLReader();
    // 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 SAX Example: Event Handler

```
class myContentHandler implements ContentHandler {  
  
    // 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));  
    }  
}
```

# Character Data

---

- Character data
  - ◆ void `characters(char [ ] ch, int start, int length)`  
throws `SAXException`
- Parsers are allowed to break up character data any way desired
- Character data are in Unicode regardless of encoding scheme specified in XML document

# White Space

---

- White space
  - ◆ Nonvalidating parser
    - void **characters**(char [ ] ch, int start, int length) throws SAXException
  - ◆ Validating parser
    - void **ignorableWhiteSpace**(char [ ]ch, int start, int length) throws SAXException
- “Ignorable white space” includes spaces, tabs and newlines

# Attributes Interface

```
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)
}
```

# Locator Interface

---

- Tells application where events occurred

```
public interface Locator{  
    public int getLineNumber();  
    public int getColumnNumber();  
    public String getPublicId();  
    public String getSystemId();  
}
```

# Locator Interface

---

- SAX parser passes implementation instance of Locator interface to the **ContentHandler.setDocumentLocator()**
  - ◆ It should be saved to a local reference if the application needs it

# Locator Example

---

```
Locator loc;
```

```
public void setDocumentLocator(Locator loc){  
    this.loc = loc;  
}
```

```
public void startElement(String namespace, String  
    name, String qName, Attributes a){  
    System.out.println(name);  
    System.out.println(" line: " +  
        loc.getLineNumber());  
    System.out.println(" ID: " + loc.getSystemId());  
}
```

# ErrorHandler Interface

---

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

# DTDHandler Methods

---

```
Public interface DTDHandler{  
    void notationDecl(String name, String publicID,  
        String systemID) throws SAXException  
    void unparsedEntityDecl(String name, String  
        publicID, String systemID, String notationName)  
        throws SAXException  
}
```

# EntityResolver Interface

---

```
Public interface EntityResolver{  
    InputSource resolveEntity(String publicID, String  
        systemID) throws SAXException  
}
```

- This method is used to return an *InputSource* so the content of the external entity can be read
- An external entity
  - ◆ External Parsed Entities
  - ◆ Unparsed Entities

# DefaultHandler Class



- Helper class
- Implements
  - ◆ **ContentHandler**
  - ◆ **ErrorHandler**
  - ◆ **DTDHandler**
  - ◆ **EntityResolver**
- Just subclass and override the methods you are interested in.
- Completely optional for use

# Apache Xerces 1.3

---

- XML 1.0
- SAX Version 1 and 2
- DOM Level 1 and 2
- XML Schema

# Demo and Code Review



- Code review
  - ◆ SAX2Count (Xerces)
  - ◆ SAX2Writer (Xerces)
  - ◆ DocumentTracer (Xerces)
  - ◆ DTDReader (Xerces)
- Demo from command line
- Demo from Forte for Java

# JAXP 1.1



- A thin and lightweight Java API for **parsing** and **transforming** XML documents
- Allows for **pluggable** parsers and transformers
- Allows parsing of XML document using:
  - ◆ Event-driven (SAX 2.0)
  - ◆ Tree based (DOM Level 2)

# JAXP: Pluggable Framework for Parsers and Transformers

---

User Application

JAXP Interfaces

Reference Parser

Other Parser

# JAXP/SAX Code Sample

---

```
01 import javax.xml.parsers.*;
02 import org.xml.sax.*;
03
04 SAXParserFactory factory =
05     SAXParserFactory.newInstance();
06
07 factory.setValidating(true);
08 SAXParser parser = factory.newSAXParser();
09
10 // can also parse InputStreams, Files, and
11 // SAX input sources
12 parser.parse("http://foo.com/bar.xml",
13             mySAXEventHandler);
14 ..
```

# 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
- No random access to documents
- Searching of documents is not easy

# Summary

---

- Parsing and application
- SAX historical background
- SAX event model
- Error handling
- Apache Xerces
- JAXP 1.1
- When to use SAX

# References

---

- “Java and XML” written by Brett McLaughlin, O’Reilly, June 2000 (First edition), Chapter 3 “Parsing XML”
- “XML in a Nutshell” written by Elliotte Rusty Harold & W. Scott Means, O’Reilly, Jan. 2001 (First Edition), Chapter 17 “SAX”