Java XML parsing

Tree-based vs Event-based API

DOM SAX

Tree-based API

A tree-based API compiles an XML document into an internal tree structure. This makes it possible for an application program to navigate the tree to achieve its objective. The Document Object Model (DOM) working group at the W3C is developing a standard tree-based API for XML.

Event-based API

An event-based API reports parsing events (such as the start and end of elements) to the application using *callbacks*. The application implements and registers event handlers for the different events. Code in the event handlers is designed to achieve the objective of the application. The process is similar (but not identical) to creating and registering event listeners in the Java Delegation Event Model.

what is SAX?

DOM SAX

SAX is a set of interface definitions

For the most part, SAX is a set of interface definitions. They specify one of the ways that application programs can interact with XML documents.

(There are other ways for programs to interact with XML documents as well. Prominent among them is the Document Object Model, or DOM)

SAX is a standard interface for event-based XML parsing, developed collaboratively by the members of the XML-DEV mailing list. SAX 1.0 was released on Monday 11 May 1998, and is free for both commercial and noncommercial use.

The current version is SAX 2.0.1 (released on 29-January 2002)

See http://www.saxproject.org/

JAXP

DOM SAX

JAXP: Java API for XML Processing

This API provides a common interface for creating and using the standard SAX, DOM, and XSLT APIs in Java, regardless of which vendor's implementation is actually being used.

The main JAXP APIs are defined in the javax.xml.parsers package. That package contains two vendor-neutral factory classes: SAXParserFactory and DocumentBuilderFactory that give you a SAXParser and a DocumentBuilder, respectively. The DocumentBuilder, in turn, creates DOM-compliant Document object.

The actual binding to a DOM or SAX engine can be specified using the System properties (but a default is provided).

JAXP – other packages

org.xml.sax *Defines the basic SAX APIs.*

The "Simple API" for XML (SAX) is the event-driven, serial-access mechanism that does element-by-element processing. The API for this level reads and writes XML to a data repository or the Web.

org.w3c.dom *Defines the Document class (a DOM), as well as classes for all of the components of a DOM.*

The DOM API is generally an easier API to use. It provides a familiar tree structure of objects. You can use the DOM API to manipulate the hierarchy of application objects it encapsulates. The DOM API is ideal for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

On the other hand, constructing the DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU and memory intensive.

javax.xml.transform *Defines the XSLT APIs that let you transform XML into other forms.*

javax.xml.xpath *Defines theXPath APIs that let you query XML docs.*

JAXP in JDK

DOM SAX

The Java Platform, Standard Edition version 6.0 includes JAXP 1.4.

It uses Xerces 2.7 Xalan 2.6

For info on JAXP 1.4, see

http://jaxp.java.net/

http://download.oracle.com/javase/tutorial/jaxp/index.html

More docs

DOM SAX

API

http://download.oracle.com/javase/6/docs/api/javax/xml/parsers/package-summary.html

JAXP Tutorial

http://docs.oracle.com/javase/tutorial/jaxp/index.html

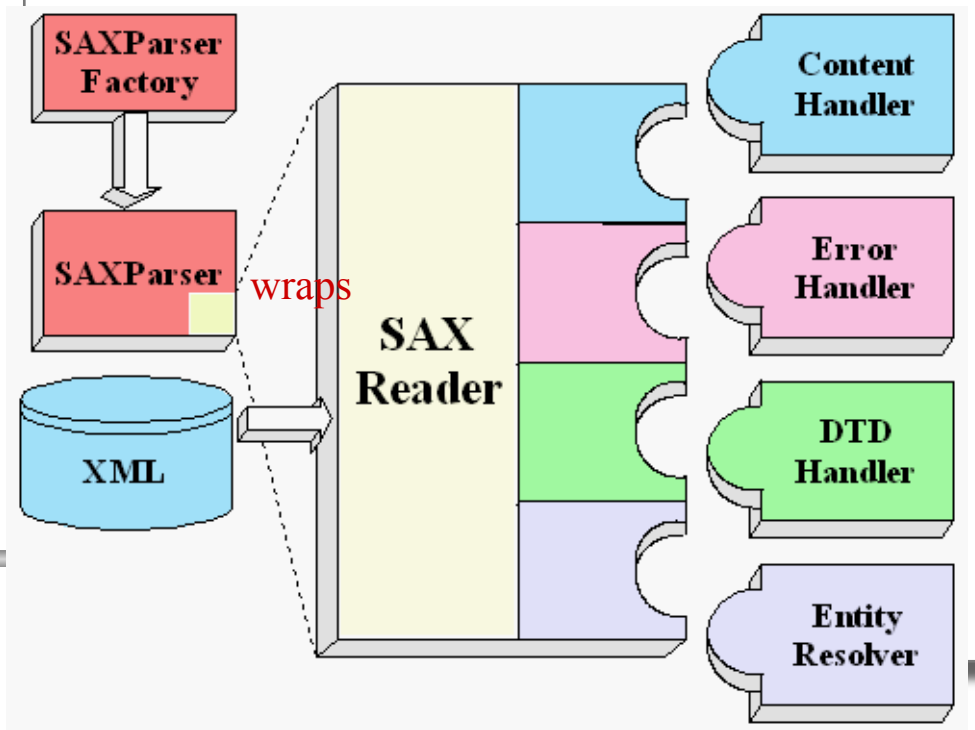
SAX architecture

DOM SAX


```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true); //optional - default is non-validating
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(File f, DefaultHandler-subclass h)
```

File containing input XML

Default-handler (classe che implementa le callback)



Interfaces implemented by DefaultHandler class

SAX packages

DOM SAX

Package	*Description*
org.xml.sax	Defines the SAX interfaces. The name "org.xml" is the package prefix that was settled on by the group that defined the SAX API.
org.xml.sax.ext	Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definitions (DTD) or to see the detailed syntax for a file.
org.xml.sax.helpers	Contains helper classes that make it easier to use SAX -- for example, by defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.
javax.xml.parsers	Defines the SAXParserFactory class which returns the SAXParser. Also defines exception classes for reporting errors.

SAX callbacks

DOM SAX

```
// ----------------------------  ContentHandler methods
void characters(char[] ch, int start, int length)
void startDocument()
void startElement(String name, AttributeList attrs)
void endElement(String name)
void endDocument()
void processingInstruction(String target,String data)
```

SAX example

DOM SAX

```
package jaxp_demo;
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
public class IndentXML extends DefaultHandler {
  public static void main(String argv[]) throws Exception {
    if (argv.length != 1) {
      System.err.println("Usage: cmd filename");
      System.exit(1);
    }
    // JAXP methods
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser saxParser = factory.newSAXParser();
    saxParser.parse(new File(argv[0]), new MySaxHandler());
   }
   MySaxHandler () {…} …implementation of the other methods…
```

Obtain a SAX parser,
Parse the file

SAX example 1

DOM SAX

```
package jaxp_demo;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.*;
import java.io.*;
public class MySaxHandler extends DefaultHandler {
  int indentCount=0;
  String indentString="   ";
  private PrintStream out = System.out;
  private void emit(String s) {
    out.print(s);
    out.flush();
  }
  private void nl() {
    String lineEnd = System.getProperty("line.separator");
    out.print(lineEnd);
  }
  private void indent(){
     String s="";
     for (int i=1;i<=indentCount;i++) s=s+indentString;
     out.print(s);
  }
```

Utility methods

SAX example 1

DOM SAX

```
//=============================================================
// SAX DocumentHandler methods
//=============================================================

  public void startDocument() throws SAXException {
    emit("<?xml version='1.0' encoding='UTF-8'?>");
    nl();
  }

  public void endDocument() throws SAXException {
    nl();
    out.flush();
  }
```

SAX example 1

DOM SAX

```
public void startElement(String namespaceURI,
                         String lName, // local name
                         String qName, // qualified name
                         Attributes attrs) throws SAXException {
    String eName = lName; // element name
    if ("".equals(eName)) eName = qName;
    indent();
    emit("<" + eName);
    if (attrs != null) {
      for (int i = 0; i < attrs.getLength(); i++) {
        String aName = attrs.getLocalName(i); // Attr name
        if ("".equals(aName)) aName = attrs.getQName(i);
        emit(" ");
        emit(aName + "=\"" + attrs.getValue(i) + "\"");
      }
    }
    emit(">");
    nl();
    indentCount++;
  }
```

SAX example 1

DOM SAX

```
public void endElement(String namespaceURI,
                       String sName, // simple name
                       String qName // qualified name
                       ) throws SAXException {
    indentCount--;
    indent();
    emit("</" + qName + ">");
    nl();
  }

  public void characters(char buf[], int offset, int len)
         throws SAXException {
         String s = new String(buf, offset, len);
         StringBuffer buffer = new StringBuffer(s);
         buffer.reverse();
         s=buffer.toString();
         emit(s);
  }
}
```

SAX references

JAXP tutorial

http://docs.oracle.com/javaee/1.4/tutorial/doc/#wp65584

SAX Tutorial

http://docs.oracle.com/javaee/1.4/tutorial/doc/#wp65584