Extensible Markup Language

Simple API for XML

- SAX -

Introducción

- ¿Cómo maneja la información representada en XML un programa?
  - Requiere módulo analizador → Lee fichero y crea representación en memoria (objetos, variables, …)
- ¿Qué alternativas tenemos?
  - Crear nuestro propio analizador XML
    - Problemas → Consideraciones UNICODE, gestión de entidades, espacios de nombres…
  - Utilizar analizadores/*parsers* XML (xerces, saxon, msxml,…)
- Existen dos tendencias principales
  - Procesamiento en base a eventos
  - Procesamiento en árbol

Procesamiento en base a eventos

4

- El *parser* analiza el documento XML de forma secuencial
- Puede pararse para insertar ENTIDADES de DTD o ir validando el documento de acuerdo a DTD/Esquema
- Según lee el documento va generando eventos.
- Puede haber muchos eventos, no sólo apertura y cierre de un elemento
  - Definición de un nuevo espacio de nombres, atributo encontrado, comentario, instrucciones de procesamiento, etc.

Procesamiento en base a eventos

5

- Los analizadores orientados a eventos no mantienen memoria del documento
  - Sólo generan eventos, por tanto, el usuario de la API debe mantener el estado si lo necesita
  - Analizador sencillo → Aplicación compleja
- Limitaciones
  - Edición y construcción de documentos
  - Recorrido aleatorio del documento
- Ventajas
  - Simplicidad, Rapidez, y Consumo de memoria reducido
  - Ideales para leer documentos que no cambian → P.e. ficheros de configuración

Procesamiento en base a modelo árbol

6

- Los documentos XML describen estructuras en árbol, por tanto, du representación en memoria como un árbol es la alternativa más lógica.
- Existen diferentes modelos de árbol para los documentos XML ligeramente diferentes
  - XPath
  - DOM (Document Object Model) API del W3C (Representación de objetos en memoria)
  - XML Infoset también del W3C
  - Extensiones para aplicaciones particulares (SVG)
- El más extendido y conocido → DOM

Procesamiento en base a modelo árbol

7

- Ventajas modelos en árbol
  - El programa puede acceder de forma nativa a elementos XML
    - Objetos XML → Objetos Java, objetos C++, estructuras C, etc.
  - No hay limitaciones en cuanto al recorrido
    - Podemos volver hacia atrás, ir directamente al final, etc.
  - Sencillo modificar documento
    - Se añaden nuevos objetos al árbol cuyas ramas se construyen típicamente en base a listas → Modelo preferido para editores XML, browsers

Procesamiento en base a modelo árbol

8

- Inconvenientes
  - Grandes requerimientos de **memoria** → un documento XML en memoria puede ocupar varias veces el tamaño fichero XML.
    - P.e. consideremos elemento: <a/> (4 bytes UTF-8)
    - Representación Java podría traducirse a objeto Element (Name(String),AttributeList,NamespaceList) → Podría irse a 200 bytes
    - ¿Cuánto ocuparía la representación de un fichero de 120 MB?
  - Inviable para procesamiento de grandes volúmenes de información → Alternativas mixtas, trucos, etc.

APIs de mayor nivel

9

- Generalmente toda API de manejo información XML se basa internamente en el modelo basado en árbol o en el basado en eventos.
- Incluso las implementaciones basadas en árbol se suelen crear sobre APIs definidas en base a eventos.
  - Es habitual que ofrezcan las dos posibilidades.
- Casos como las APIs de manejo de documentos SVG (Scalable Vector Graphics) añaden extensiones propias a APIs DOM para adaptarse mejor a sus características concretas
- En la programación no tenemos por qué quedarnos con un único modelo
  - Es habitual combinar ambos.

¿Qué es SAX?

10

- SAX significa *Simple API for XML* → API **basada en eventos**
  - Website oficial → http://www.saxprojet.org
  - Originariamente fue una especificación para Java
- Se ha convertido en un *estándar de facto*. Portada a multitud de plataformas:
  - Java, MSXML 3.0 (a través de objeto COM, Visual Basic, C, C++, …), Pascal, Perl, Python 2.0, C (Xerces-C), C++, …
- Xerces → API Parser Java y otros lenguajes del proyecto Apache. Soporta entre otros
  - Interfaces SAX version 2 (actual), DOM level 3, JAXP 1.2 → Java API for XML Processing y Validación XML Schema

SAX y JAVA

11

- SAX es una API estándar
  - No obstante cada implementación puede variar ligeramente
- Java proporciona la API JAXP → Java API for XML Processing
  - Proporciona interfaces comunes para utilizar: SAX, DOM y XSLT
  - Abstrae del fabricante.
  - Equivalente a JDBC o ADO (MS) para BBDD
- ¿Cómo uso entonces una API SAX en Java?
  - Directamente a través del driver
  - A través de la API SAX independiente del driver
  - A través de la API JAXP

¿Qué necesito para empezar?

12

- JAXP está incluido como paquete a partir de Java1.4
  - Paquetes: *javax.xml.**
- JAXP está disponible de forma separada para versiones anteriores a Java1.4. En este caso necesitaríamos:
  - Un parser compatible SAX2 instalado en el *classpath*
    - Recomendable Xerces http://xml.apache.org/xerces-j/
  - La distribución SAX2 para Java
- Opcionalmente
  - Entorno de desarrollo Java (en nuestro caso Eclipse)

Pasos de análisis con SAX

13

1. Indicar al sistema el analizador que queremos utilizar.
2. Crear una instancia del analizador.
3. Crear un manejador de contenidos para responder a los eventos que genera el analizador.
4. Invocar al analizador con el manejador de contenidos definido y el documento a analizar.

Pasos de análisis con SAX

14

1. Indicar al sistema el analizador que queremos utilizar
   - Fijar la propiedad del sistema:
     `javax.xml.parsers.SAXParserFactory`
   - Especificarlo en la máquina virtual:
     `jre_dir/lib/jaxp.properties`
   - A través de los servicios J2EE
     `META-INF/services`
   - Utilizar el analizador por defecto

Pasos de análisis con SAX

15

1. Indicar al sistema el analizador que queremos utilizar
   - Fijar la propiedad del sistema en línea de comandos

```
java -Djavax.xml.parser.SAXParserFactory=
     com.sun.xml.parser.SAXParserFactoryImpl ...
```

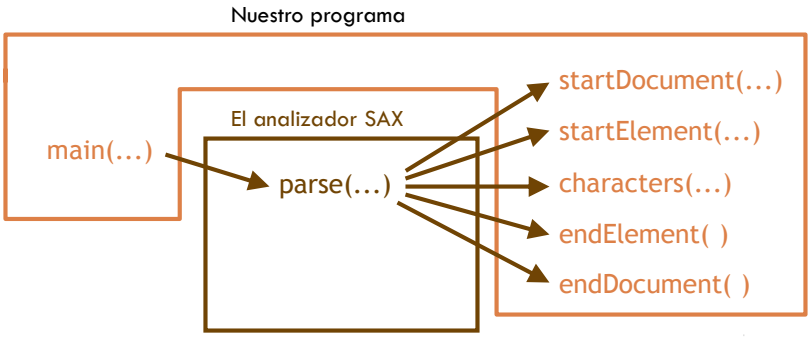
   - Fijar la propiedad del sistema por programa

```
String jaxpPropertyName =
  "javax.xml.parsers.SAXParserFactory";
if (System.getProperty(jaxpPropertyName) == null) {
  String apacheXercesPropertyValue =
    "org.apache.xerces.jaxp.SAXParserFactoryImpl";
  System.setProperty(jaxpPropertyName,
                     apacheXercesPropertyValue);
}
```

Funcionamiento de SAX: Callbacks

16

- SAX funciona mediante callbacks: nuestro programa invoca al analizador, y éste hace llamadas a los métodos que le proporcionamos.



Un ejemplo SAX sencillo

17

- El programa consiste en dos clases:
  - Principal – Esta clase contiene el método main que se encarga de
    - Obtiene una factoría de analizadores
    - Obtiene un analizador de la factoría
    - Crea un objeto de Manejador para manejar las llamadas desde el analizador
    - Le dice al intérprete a qué manejador enviar sus devoluciones de llamada
    - Lee y analiza el archivo de entrada XML
  - Manejador -- esta clase contiene manejadores para tres tipos de callbacks:
    - startElement callbacks, generados cuando aparece una etiqueta de inicio
    - endElement callbacks, generados cuando aparece una etiqueta de cierre
    - characters callbacks, generados por los contenidos de un elemento

La clase Principal, I

18

```
import javax.xml.parsers.*; // para SAX y DOM
import org.xml.sax.*;

// por simplicidad no vamos a manejar la excepciones
// aunque esta no es una buena práctica de programación
public class Principal{
   public static void main(String args[]) throws Exception {
      // Creamos una factoría de analizadores
      SAXParserFactory factory = SAXParserFactory.newInstance();
      // el analizador debe entender los espacios de nombres
      factory.setNamespaceAware(true);
      // Creamos el analizador
       SAXParser saxParser = factory.newSAXParser();
       XMLReader reader = saxParser.getXMLReader();
```

La clase Principal, II

19

- Continuación…

```
      // creamos el manejador
      Manejador handler = new Manejador();
      // le decimos al analizador que utilice este manejador
      reader.setContentHandler(handler);
      // finalmente leemos y analizamos el documento XML
      reader.parse(”saludos.xml");
   } // final de la clase Principal
```

- Necesitamos crear un documento saludos.xml :
  - En el mismo directorio, si ejecutamos el programa desde la línea de comandos
  - O dónde pueda ser encontrado por el IDE que utilicemos.

La clase Manejador, I

20

- public class Manejador extends DefaultHandler {
  - DefaultHandler es una clase adaptador que define éstos y otros métodos como métodos vacios que deberán ser sobreescritos cuando se deseen utilizar.
  - Vamos a definir tres métodos muy similares (simplemente escriben una línea) para manejar (1) las etiquetas de inicio, (2) los contenidos, y (3) las etiquetas de cierre.
  - Cada uno de estos métodos lanzará una excepción de tipo SAXException

```
// SAX llama a este método cuando encuentra una etiqueta de inicio
public void startElement(String namespaceURI,
                         String localName,
                         String qualifiedName,
                         Attributes attributes)
    throws SAXException {
  System.out.println("startElement: " + qualifiedName);
}
```

La clase Manejador, II

21

```
// SAX llama a este método en los contenidos de los elementos
public void characters(char ch[], int start, int length)
    throws SAXException {
  System.out.println("characters: \"" +
                     new String(ch, start, length) + "\"");
}

// SAX llama a este método cuando encuentra una etiqueta de cierre
public void endElement(String namespaceURI,
                       String localName,
                       String qualifiedName)
    throws SAXException {
  System.out.println("Element:  /" + qualifiedName);
}

} // fin de la clase Manejador
```

Resultados

22

- Si el fichero saludos.xml contiene:

```
<?xml version="1.0"?>
<saludos>Hola Mundo!</saludos>
```

- La salida de la ejecución será:

```
startElement: saludos
characters: "Hola Mundo!"
Element:  /saludos
```

Más Resultados

23

- Ahora supón que el fichero saludos.xml contiene:

```
<?xml version="1.0"?>
<saludos>
   <i>Hola</i> Mundo!
</saludos>
```

- El elemento raíz, <saludos>, ahora contiene un elemento anidado <i> y algunos espacios en blanco (incluyendo nuevas líneas)
- El resultado sería el mostrado a la derecha:

```
startElement: saludos
characters: ""  // cadena vacía
characters: "
"       // nueva línea
characters: "    "  // espacios
startElement: i
characters: "Hola"
endElement: /i
characters:  "Mundo!"
characters: "
"       // otra nueva línea
endElement: /saludos
```

Factorías de analizadores

24

- Una factoría es una alternativa a un constructor
- Para crear una factoría de analizadores SAX, hay que llamar al método:
  SAXParserFactory.newInstance()
  - Quie devuelve un objeto del tipo SAXParserFactory
  - Y puede lanzar una excepción de tipo FactoryConfigurationError
- Podemos fijar el tipo de analizador que deseamos:
  - public void setNamespaceAware(boolean awareness)
    - Llamamos al método con true si queremos utilizar espacios de nombres
    - Por defecto (si no se llama al método) el valor es false
  - public void setValidating(boolean validating)
    - Llamamos al método con true si queremos validar el documento con un DTD
    - Por defecto (si no se llama al método) el valor es false
    - La validación puede dar error si no existe DTD

Obteniendo un analizador

25

- Una vez tenemos una SAXParserFactory podemos crear un analizador con:

```
SAXParser saxParser = factory.newSAXParser();
XMLReader reader = saxParser.getXMLReader();
```

- Nota: en algunos códigos se puede utilizar Parser en lugar de XMLReader
  - Parser es SAX1, no SAX2, y ahora está obsoleto
  - SAX2 soporta espacios de nombres y nuevas funcionalidades de los analizadores
- Nota: SAXParser no es *thread-safe*; para usarlo con múltiples hilos hay que crear un SAXParser por cada uno de ellos.
  - Esto es poco probable que sea un problema en los proyectos.

Indicando el manejador a utilizar

26

- Antes de que el analizador SAX llame a nuestros métodos, necesitamos proporcionarle dichos métodos
- En nuestro ejemplo los proporcionamos en la clase, Manejador
- Necesitamos indicar al analizador cómo encontrar los métodos:

```
Manejador handler = new Manejador();
reader.setContentHandler(handler);
```

- Estas sentencias pueden combinarse:

```
reader.setContentHandler(new Manejador());
```

- Finalmente llamamos al analizador y le proporcionamos el documento:

```
reader.parse("hello.xml");
```

- El resto de las acciones se llevan a cabo en los métodos.

Manejadores SAX

27

- Un manejador de *callbacks* SAX debe implementar estos cuatro interfaces:
  - interface ContentHandler
    - Es el interfaz más importante -- maneja los callbacks básicos, cómo los inicios y finales de los elementos.
  - interface DTDHandler
    - Maneja solamente las declaraciones de las entidades de tipo notación y no analizadas.
  - interface EntityResolver
    - Maneja las entidades externas
  - interface ErrorHandler
    - Debe ser implementada o los errores de análisis serán ignorados.
- Se pueden implementar todos los interfaces, pero es más sencillo utilizar una clase adaptadora.

Clase DefaultHandler

28

- DefaultHandler está en el paquete org.xml.sax.helpers
- DefaultHandler implementa ContentHandler, DTDHandler, EntityResolver, y ErrorHandler
- DefaultHandler es una clase adaptadora que proporciona métodos vacíos para cada uno de los métodos declarados en cada uno de los cuatro interfaces.
  - Los métodos vacíos no hacen nada.
- Para usar esta clase, hay que extenderla y sobreescribir los métodos que son importantes para nuestra aplicación.
  - En nuestro ejemplo reescribiremos algunos de los métodos de las interfaces ContentHandler y ErrorHandler

ContentHandler métodos, I

29

- public void setDocumentLocator(Locator loc)
  - Este método se llama una única vez cuando comienza el análisis.
  - El Locator contiene una URL o un URN, o ambas, que especifican la localización del documento.
  - Esta información puede ser necesaria si necesitamos encontrar un documento cuya posición sea relativa al documento XMl que se analiza.
  - Locator incluye los métodos:
    - public String getPublicId() devuelve el identificador público del documento actual
    - public String getSystemId() devuelve el identificador de sistema del documento actual
  - Cada método de ContentHandler excepto este puede lanzar una excepción de tipo SAXException

ContentHandler métodos, II

30

- public void processingInstruction(String target, String data) throws SAXException
- Este método es llamado cada vez que se encuentra una intrucción de procesamiento (PI)
- La PI se representa con dos cadenas: <?*target* *data*?>
- De acuerdo con las reglas XML, las PI's pueden aparecer en cualquier lugar del documento después de la declaración inicial <?xml ...?>
  - Esto significa que las llamadas a processingInstruction no tienen porqué ocurrir antes de que startElement sea llamado con la raíz del documento, puede ocurrir después.

ContentHandler métodos, III

31

- public void startDocument() throws SAXException
  - Sólo se llama una vez al comienzo del análisis.
- public void endDocument() throws SAXException
  - Sólo se llama una vez, y es el último método que es llamado en el análisis.
- Nota: cuando se reescribe un método se pueden lanzar algunos tipos de excepciones pero no de cualquier clase. En otras palabras:
  - los métodos no tienen por qué lanzar una excepción de tipo SAXException
  - Pero si lanzan una excepción, sólo podrá ser de tipo SAXException

ContentHandler métodos, IV

32

- public void startElement(String namespaceURI,
  String localName,
  String qualifiedName,
  Attributes atts)
  throws SAXException
- Este método se llama en el inicio de cada elemento.
- Si el analizador acepta espacios de nombres,
  - namespaceURI almacenará el prefijo (antes de los ":")
  - localName almacenará el nombre del elemento (sin prefijo)
  - qualifiedName será una cadena vacía
- Si el analizador no utiliza espacios de nombres,
  - qualifiedName almacenará el nombre del elemento (posiblemente con prefijo)
  - namespaceURI y localName serán cadenas vacías

Attributes, I

33

- Cuando SAX llama al método startElement, le pasa un parámetro de tipo Attributes
- Attributes es una interfaz que define un conjunto de métodos muy útiles para trabajar con los atributos de los elementos. Algunos de ellos son:
  - getLength() devuelve el número de atributos
  - getLocalName(index) devuelve el nombre local de un atributo
  - getQName(index) devuelve el nombre cualificado de un atributo
  - getValue(index) devuelve el valor de un atributo
  - getType(index) devuelve el tipo de un atributo, que será algunas de las siguientes cadenas "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", o "NOTATION"
- Al igual que ocurre con los elementos, si el nombre local es una cadena vacía, el nombre del atributo estará almacenado en el nombre cualificado.

Attributes, II

34

- SAX no garantiza que los atributos sean devueltos en el mismo orden en que fueron escritos.
  - Después de todo, el orden de los atributos es irrelevante en XML.
- Los siguientes métodos manejan los atributos en base a su nombre y no por su índice de aparición:
  - public int getIndex(String qualifiedName)
  - public int getIndex(String uri, String localName)
  - public String getValue(String qualifiedName)
  - public String getValue(String uri, String localName)
- Un objeto Attributes sólo es válido durante la llamada a startElement
  - Si es necesario mantener los atributos más allá de la llamada, hay que utilizar:
    AttributesImpl attrImpl = new AttributesImpl(attributes);

ContentHandler métodos, V

35

- endElement(String namespaceURI,
  String localName,
  String qualifiedName)
  throws SAXException
- Los parámetros de endElement son los mismos que en startElement, excepto que el parámetro Attributes se omite.

ContentHandler métodos, VI

36

- public void characters(char[] ch,
  int start,
  int length) throws SAXException
- ch es un array de caracteres
  - Sólamente length caracters, comenzando desde ch[start], son los contenidos del elemento.
- El constructor de la clase String new String(ch, start, length) es un método muy sencillo para extraer la información relevante del array de caracteres.
- characters puede ser llamado múltiples veces para un mismo elemento.
  - Las nuevas líneas y las entidades implican llamadas separadas.
  - characters puede ser llamado con length = 0
  - Todos los datos de tipo caracter de un elemento pueden ser proporcionados por characters

Ejemplo

37

- Si saludos.xml contiene:
  - <?xml version="1.0"?>
    <saludos>
    Hola Mundo!
    </saludos>
- El programa generaría como salida:
  - startElement: saludos
    characters: <-- cadena de longitud 0
    characters: <-- caracter LF (ASCII 10)

    characters: Hola Mundo! <-- los espacios son preservados
    characters: <-- caracter LF (ASCII 10)

    Element: /saludos

Espacios en blanco, I

38

- Los espacios en blanco una incoveniencia considerable:
  - Los espacios en blanco son caracteres, por tanto, PCDATA
  - Si estamos validando, el analizador ignorará los espacios en blanco en dónde el DTD no permite que exita PCDATA.
  - Si no estamos validando, el analizador no puede ignorar los espacios en blanco.
  - Si ignoramos los espacios en blanco podemos perder la identación.
- Para ignorar los espacios en blanco cuando no estamos validando:
  - Podemos utilizar el método trim() de la clase String.

Espacios en blanco, II

39

- Un analizador no validador no puede ignorar los espacios en blanco porque no puede distinguirlos de los datos reales.
- Un analizador validador puede, y lo hace, ignora los espacios en blanco cuando los datos de tipo caracter no están permitidos.
  - Esto suele ser lo deseado cuando procesamos documentos XML
  - Sin embargo, si que queremos generar XML, descartar los espacios en blanco puede arruinar la identación del documento resultado.
  - Para capturar los espacios en blanco ignorables podemos reescribir el método (definido en DefaultHandler):

```
public void ignorableWhitespace(char[] ch,
                                int start,
                                int length)
                    throws SAXException
```

    - Los parámetros son los mismos que en el método characters

Manejo de Errores, I

40

- La mayoría de los errores se ignoran a no ser que se defina y registre un manejador de errores (org.xml.sax.ErrorHandler)
  - Ignorar los errores puede generar comportamientos no deseados.
  - No proporcionar un manejador de errores es inapropiado.
- La interfaz ErrorHandler declara:
  - public void fatalError (SAXParseException exception)
    throws SAXException // XML mal estructurado
  - public void error (SAXParseException exception)
    throws SAXException // error de validación XML
  - public void warning (SAXParseException exception)
    throws SAXException // problema menor

Manejo de Errores, II

41

- Si estamos extendiendo DefaultHandler, éste implementa y registra ErrorHandler
  - La versión del método fatalError() de DefaultHandler lanza una excepción de tipo SAXException, pero...
  - Sus métodos error() y warning() no hacen nada. Debemos sobreescribir estos métodos
- Nota: el único tipo de excepción que pueden lanzar los métodos sobreescritos es SAXException
  - Cuando sobreeescribimos un método no podemos añadir nuevos tipos de excepciones.
  - Si necesitamos lanzar otro tipo de excepción, por ejemplo una IOException, debemos encapsularla en una SAXException:

```
catch (IOException ioException) {
    throw new SAXException("I/O error: ", ioException)
}
```

Manejo de Errores, III

42

- Si no estamos extendiendo DefaultHandler:
  - Creamos una nueva clase (por ejemplo, ManejadorErrores) que implemente ErrorHandler (proporcionando los tres métodos fatalError, error, y warning)
  - Instanciamos un objeto de esta clase
  - Indicamos a nuestro objeto XMLReader el manejador de errores a utilizar enviándo el siguiente mensaje:
    setErrorHandler(ErrorHandler handler)
- Ejemplo:

```
XMLReader reader = saxParser.getXMLReader();
reader.setErrorHandler(new ManejadorErrores());
```