Extensible Markup Language

Document Object Model

- DOM -

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

3

- El *parser* analiza el documento XML de forma secuencial
- Puede pararse para insertar ENTIDADES de DTD o ir validando 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, instrucción de procesamiento, etc.

Procesamiento en base a eventos

4

- Los analizadores orientados a eventos no mantienen memoria del documento
  - Sólo generan eventos → 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

5

- Los documentos XML describen estructuras en árbol, por tanto, la representación en memoria como un árbol el 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

6

- Ventajas modelos de 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

7

- Inconvenientes
  - Grandes requerimientos de **memoria** → un documento XML en memoria puede ocupar. Varias veces 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

8

- Generalmente toda API de manejo información XML se implementa haciendo uso del modelo basado en árbol o del basado en eventos.
- Incluso las implementaciones basadas en árbol se suelen crear sobre APIs 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 DOM?

9

- Es una API para el tratamiento de documentos XML y HTML independiente del lenguaje y basada en objetos.
  - Definida utilizando *Interface Definition Language* (IDL; definido en la especificación de Corba del *OMG*).
  - Permite a los programas y scripts construir documentos, navegar por su estructura, añadir, modificar o eliminar elementos y contenidos.
  - Proporciona la base para el desarrollo de aplicaciones de alto nivel para la consulta, filtrado, transformación y rendering de documentos XML.
- En contraste con "*Serial Access XML*" podemos pensar en"*Directly Obtainable in Memory*"

¿Qué es DOM?

11

- Nos permite proporcionar un acceso uniforme a documentos estructurados desde diferentes aplicaciones (analizadores, navegadores, editores, bases de datos…).
- Es una especificación del W3C.
  - http://www.w3c.org/DOM
  - La segunda en la familia de recomendaciones de XML
    - Level 1, W3C Rec, Oct. 1998
    - Level 2, W3C Rec, Nov. 2000
    - Level 3, W3C Working Draft (January 2002)

DOM Level 1

12

- Representación básica y manipulación de la estructura de los documentos y su contenido (no se proporciona acceso a los contenidos de un DTD).

I: DOM **Core Interfaces**

- **Fundamental interfaces**
  - Interfaces básicas para documentos estructurados
- Extended interfaces
  - Específicos de XML: CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction

II: DOM **HTML Interfaces**

- Acceso más conveniente a documentos HTML
- (no lo estudiamos)

DOM Level 2

13

- DOM Level 2 añade
  - Soporte para espacio de nombres
  - Acceso a los elementos en base al valor del atributo ID
  - Características adicionales
    - Interfaces a vistas de documentos y hojas de estilo.
    - Incluye un modelo de eventos.
    - Métodos para recorrer el árbol del documento y manipular regiones del mismo.
  - La carga y escritura de documentos no se especifica (-> Level 3)

DOM *estructura del modelo*

14

- Basado en conceptos de OO:
  - *métodos* (para acceder o cambiar el estado de los objetos)
  - *interfaces* (declaración de conjuntos de métodos)
  - *objetos* (encapsulación de datos y métodos)
- Muy similar al modelo de datos de XSLT/Xpath ≈ un árbol de análisis
  - La estructura similar a un árbol está implicita por las relaciones abstractas definidas en las interfaces. Éstas no reflejan necesariamente las estructuras de datos usadas para la implementacón (pero probablemente lo hacen).

DOM y JAVA

15

- 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 DOM en Java?
  - Directamente a través del driver
  - A través de la API DOM independiente del driver
  - A través de la API JAXP

¿Qué necesito para empezar?

16

- 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 analizador compatible DOM instalado en el *classpath*
    - Recomendable Xerces http://xml.apache.org/xerces-j/
  - La distribución JAXP para Java
- Opcionalmente
  - Entorno de desarrollo Java (en nuestro caso Eclipse)

Pasos de análisis con DOM

17

1. Indicar al sistema el analizador que queremos utilizar *DocumentBuilderFactory*.
2. Crear un JAXP *DocumentBuilder*.
3. Invocar al analizador y crear un objeto *Document* para representar el documento XML.
4. Recorrer el árbol generado realizando las operaciones oportunas.

Pasos de análisis con DOM

18

1. Indicar al sistema el analizador que queremos utilizar
   - Fijar la propiedad del sistema: `javax.xml.parsers.DocumentBuilder-Factory`
   - 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 DOM

19

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

```
java -Djavax.xml.parser.DocumentBuilderFactory =
     com.sun.xml.parser.DocumentBuilderFactoryImpl ...
```

   - Fijar la propiedad del sistema por programa

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

DOM un programa sencillo, I

20

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class EjemploDOM {
    public static void main(String args[]) {
        try {
            ...cuerpo principal del programa...
        } catch (Exception e) {
            System.out.println("Error: "+e.getMessage());
        }
    }
}
```

DOM un programa sencillo, II

21

- Primero necesitamos crear un analizdor DOM, llamado "*DocumentBuilder*"
- El analizador se crea haciendo uso de una factoría.
  - Esta es una técnica común en programación avanzada en Java. El uso de una factorñia facilita el cambio posterior a otro tipo de analizador, sin necesidad de cambiar la aplicación.

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();

DocumentBuilder builder =
    factory.newDocumentBuilder();
```

DOM un programa sencillo, III

22

- El siguiente paso es cargar el documento XML. El contenido del fichero saludos.xml es:

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

- Para poder leer el fichero necesitamos aañdir la siguiente línea al programa:

```
Document document = builder.parse("saldos.xml");
```

- Notas:
  - document contiene todo el documento XML (como un árbol); este es el Document Object Model
  - Si se ejecuta el programa desde la línea de comandos, el fichero deberá estar en el mismo directorio que el programa.
  - Si se utiliza un IDE es posible que sea necesario ponerlo en un directorio distinto; si no se encuentra es posible que se produzca una java.io.FileNotFoundException.

DOM un programa sencillo, IV

23

- El siguiente fragmento de código encuentra el contenido del elemento raíz y lo imprime.

```
Element root = document.getDocumentElement();
Node textNode = root.getFirstChild();
System.out.println(textNode.getNodeValue());
```

- Este código debería ser suficientemente autoexplicativo, pero entraremos en detalle pronto.
- La salida del programa es: Hola Mundo!

Leyendo en el árbol

24

- El método parse lee el documento XML completo y lo representa como un árbol en memoria.
  - Cuando el documento es grande, el análisis puede necesitar bastante tiempo.
  - Para que el programa pueda interactuar mientras se está analizando, es necesario analizar en un hilo separado.
    - Una vez que el análisis comienza no puede ser interrumpido o parado.
    - No se puede acceder al árbol de análisis mientras el análisis se está realizando.
- Un árbol de análisis XML puede requerir hasta diez veces más recursos de memoria que el documento XML original.
  - En los programas con mucha manipulación del árbol es preferible DOM a SAX, en otro caso hay que considerar el uso de SAX.

Estructura del árbol DOM

25

- El árbol DOM está compuesto de nodos, objetos Node
- Node es un interface
  - Algunos de sus subinterfaces más importantes son Element, Attr, y Text
    - Un nodo Element puede tener hijos
    - Los nodos Attr y Text son hojas
  - Otros tipos adicionales son Document, ProcessingInstruction, Comment, Entity, CDATASection y varios más.
- Por tanto, el árbol DOM está compuesto enteramente por objetos de tipo Node, pero estos objetos Node pueden realizar un *downcast* en tipos más específicos según sea necesario.

Operaciones sobre Node, I

26

- Los resultados devueltos por getNodeName(), getNodeValue(), getNodeType() y getAttributes() dependen del subtipo de nodo tal y como se muestra en la tabla:

	Element	Text	Attr
getNodeName()	*tag name*	"#text"	*name of attribute*
getNodeValue()	null	*text contents*	*value of attribute*
getNodeType()	ELEMENT_NODE	TEXT_NODE	ATTRIBUTE_NODE
getAttributes()	NamedNodeMap	null	null

Diferenciando los tipos de Nodos

27

- El siguiente código muestra una manera sencilla de poder trabajar con los diferentes tipos de nodos:

```
switch(node.getNodeType()) {
   case Node.ELEMENT_NODE:
        Element element = (Element)node;
        ...;
        break;
   case Node.TEXT_NODE:
        Text text = (Text)node;
        ...
        break;
   case Node.ATTRIBUTE_NODE:
        Attr attr = (Attr)node;
        ...
        break;
   default: ...
}
```

Operaciones sobre Node, II

28

- Operaciones de recorrido que devuelven un Node:
  - getParentNode()
  - getFirstChild()
  - getNextSibling()
  - getPreviousSibling()
  - getLastChild()
- Comprobaciones que devuelven un boolean:
  - hasAttributes()
  - hasChildNodes()

Operaciones sobre Element

29

- String getTagName()
  - Devuelve el nombre de la etiqueta
- boolean hasAttribute(String name)
  - Devuelve true si el Element tiene el atributo indicado
- String getAttribute(String name)
  - Devuelve el valor (String) del atributo indicado
- boolean hasAttributes()
  - Devuelve true si el Element tiene al menos un atributo
  - Este método se hereda de Node
    - Devuelve false si se aplica a un Node que no sea un Element
- NamedNodeMap getAttributes()
  - Devuelve un NamedNodeMap con todos los atributos del Element
  - Este método se hereda de Node
    - Devuelve false si se aplica a un Node que no sea un Element

Operaciones sobre Document

30

- El interfaz Document representa el documento HTML o XML completo. Conceptualmente es la raíz del árbol asociado al documento y proporciona el acceso al mismo.
- Contiene métodos para obtener información y manipular los documentos, en especial contiene todos los métodos necesarios para poder crear los diferentes tipos de nodos.
- Caben destacar los métodos:
  - getDocumentElement() que devuelve el elemento raíz del documento.
  - getElementByID(String elementID) que devuelve un elemento en particular, el identificado por elementID.
  - getElementsByTagName(String tagname) que permiten recuperar directamente un conjunto de nodos (NodeList) asociados a la etiqueta especificada en tagname.

NodeList

31

- Algunos métodos devuelven como resultado una lista de nodos. Esta interfaz (NodeList) proporciona una abstracción de una colección ordenadas de nodos.
- Los *items* de esta lista son accesibles mediante un índice integral que comienza en 0.
- Sus dos métodos permiten una fácil manipulación de los nodos que contiene:
  - getLength() devulelve el número (un int) de nodos (Node) en el NodeList.
  - item(int index) devuelve el item (un Node) de posición fijada por el parámetro *index*.

NamedNodeMap

32

- El método *node*.getAttributes() devuelve un NamedNodeMap
  - Como NamedNodeMap se devuelve también en la llamada a otros métodos, sus contenidos son de tipo Node, y no específicamente como componentes de tipo Attr
- Algunas operaciones sobre NamedNodeMap son:
  - getNamedItem(String name) devuelve (un Node) el atributo con el nombre dado
  - getLength() devulelve (un int) el número de Nodes en el NamedNodeMap
  - item(int index) devuelve (un Node) el item de posición *index*
    - Esta operación permite un recorrido por los contenidos del NamedNodeMap
    - Java *no garantiza el orden en que son devueltos los nodos*

Operaciones sobre Text

33

- Text es una subinterface de CharacterData y hereda las siguientes operaciones (entre otras):
  - public String getData() throws DOMException
    - Devuelve el texto contenido en el nodo Text
  - public int getLength()
    - Devuelve el número de caracteres Unicode del texto
  - public String substringData(int offset, int count) throws DOMException
    - Devuelve una subcadena del contendo del texto

Operaciones sobre Attr

34

- String getName()
  - Devuelve el nombre del atributo.
- Element getOwnerElement()
  - Devuelve el nodo Element en el que está definido el atributo o null si el atributo no está en uso.
- boolean getSpecified()
  - Devuelve true si al atributo se le ha dado valor explícitamente en el documento original.
- String getValue()
  - Devuelve el valor del atributo como un String

Recorrido en preorden

35

- El análisis DOM es almacenado en memoria como un árbol.
- Una forma sencilla de recorrerlo es en preorden.
- El modo general de un recorrido en preorden es el siguiente:
  - Visitar la raíz
  - Recorrer cada subárbol en orden

Recorrido en preorden en Java

36

```
static void simplePreorderPrint(String indent, Node node) {
        printNode(indent, node);
        if(node.hasChildNodes()) {
            Node child = node.getFirstChild();
            while (child != null) {
                simplePreorderPrint(indent + "  ", child);
                child = child.getNextSibling();
            }
        }
    }
static void printNode(String indent, Node node) {
        System.out.print(indent);
        System.out.print(node.getNodeType() + " ");
        System.out.print(node.getNodeName() + " ");
        System.out.print(node.getNodeValue() + " ");
        System.out.println(node.getAttributes());
    }
```

Recorrido en preorden en Java

37

Input:

```
<?xml version="1.0"?>
<novel>
  <chapter num="1">The Beginning</chapter>
  <chapter num="2">The Middle</chapter>
  <chapter num="3">The End</chapter>
</novel>
```

Output:

```
1 novel null
  3 #text
   null
  1 chapter null  num="1“
    3 #text The Beginning
null
  3 #text
   null
  1 chapter null  num="2“
    3 #text The Middle null
  3 #text
   null
  1 chapter null  num="3“
    3 #text The End null
  3 #text
 null
```

Modificando el árbol DOM

38

- Existen funciones que permiten modificar el árbol DOM, por ejemplo:
  - setNodeValue(String *nodeValue*)
  - insertBefore(Node *newChild*, Node *refChild*)
- Java proporciona muchas operaciones destinadas a modificar el árbol DOM.
- Estas operaciones no son parte de las especificaciones del W3C
- No existe un modo estandarizado de crear un documento XML a partir del árbol DOM.
  - El ejemplo anterior es una forma sencilla de hacerlo.