PRESA. UNIDAD 1. ACCESO A EXPEDIENTES PARTE 2. EXPEDIENTES. XML y AMP; XLS

DAM. Acceso a Datos (ADA) (a distancia en inglés)

Unidad 1. ACCESO A LOS EXPEDIENTES

Parte 2. Archivos. XML y XSL

Abelardo Martínez
Año 2023-2024

1. Introducción

El uso de formatos binarios independientes va más allá de la barrera del lenguaje Java, permitiendo que otros lenguajes lean archivos que siguen el formato especificado. Sin embargo, los formatos binarios aún presentan cierta incompatibilidad si los sistemas operativos en los que está trabajando utilizan diferentes sistemas de representación de datos binarios. Como sabes, no todos los sistemas representan los datos de la misma forma. Algunos usan BCD para representar números, otros usan el complemento a 2. Cuando un dato ocupa más de un byte, se puede leer empezando por el byte más pequeño (Little Endian) o desde el más grande (Big Endian), etc.

En otras palabras, los formatos binarios no garantizan la compatibilidad de datos entre sistemas. Por lo tanto, cuando se desea almacenar datos que deben ser leídos por aplicaciones que se ejecutan en múltiples plataformas, es necesario recurrir a formatos más estandarizados, como los lenguajes de marcado. Estos lenguajes de marcado, como XML, se utilizan para resolver este problema.

2. lenguaje XML

Los documentos XML estructuran la información intercalando una serie de etiquetas llamadas hashtags. En XML, los hashtags tienen cierta similitud con un contenedor de información. Así, un hashtag puede contener otros hashtags o información textual. De esta manera podremos subdividir la información estructurándola para que pueda ser información estructurándola de manera que pueda ser fácilmente interpretada.

Como toda la información es textual, no hay problema en representar los datos de diferentes formas. Cualquier dato, ya sea numérico o booleano, deberá transcribirse en modo texto, de modo que sea cual sea el sistema de representación de datos, será posible leer e interpretar correctamente la información contenida en un archivo XML.

Es cierto que los caracteres también se pueden escribir utilizando diferentes sistemas de codificación, pero XML ofrece varias técnicas para evitar que esto sea un problema. Por ejemplo, es posible incluir en el encabezado del archivo qué codificación se ha utilizado durante el almacenamiento, o también es posible escribir caracteres de código ASCII mayores a 127, utilizando entidades de caracteres, una forma universal de codificar cualquier símbolo.

XML es capaz de estructurar cualquier tipo de información jerárquica. Se puede establecer cierta similitud entre la forma en que se almacena la información en los objetos de una aplicación y la forma en que se almacenaría en un documento XML.

La información, en aplicaciones orientadas a objetos, está estructurada, agrupada y jerarquizada en clases, y en documentos XML está estructurada, organizada y jerarquizada en hashtags contenidos unos dentro de otros y en atributos de los hashtags.

2.1. ¿Qué es XML?

XML (lenguaje de marcado extensible) es un lenguaje de marcado similar al HTML, pero sin etiquetas predefinidas para usar. En su lugar, usted define sus propias etiquetas diseñadas específicamente para sus necesidades. Esta es una forma poderosa de almacenar datos en un formato que se puede almacenar, buscar y compartir. Lo más importante es que, dado que el formato fundamental de XML está estandarizado, si comparte o transmite XML entre sistemas o plataformas, ya sea localmente o a través de Internet, el destinatario aún puede analizar los datos gracias a la sintaxis XML estandarizada.

Hay muchos lenguajes basados en XML, incluido XHTML, MatemáticasML, SVG, RSS, y RDF. También puedes definir el tuyo propio.

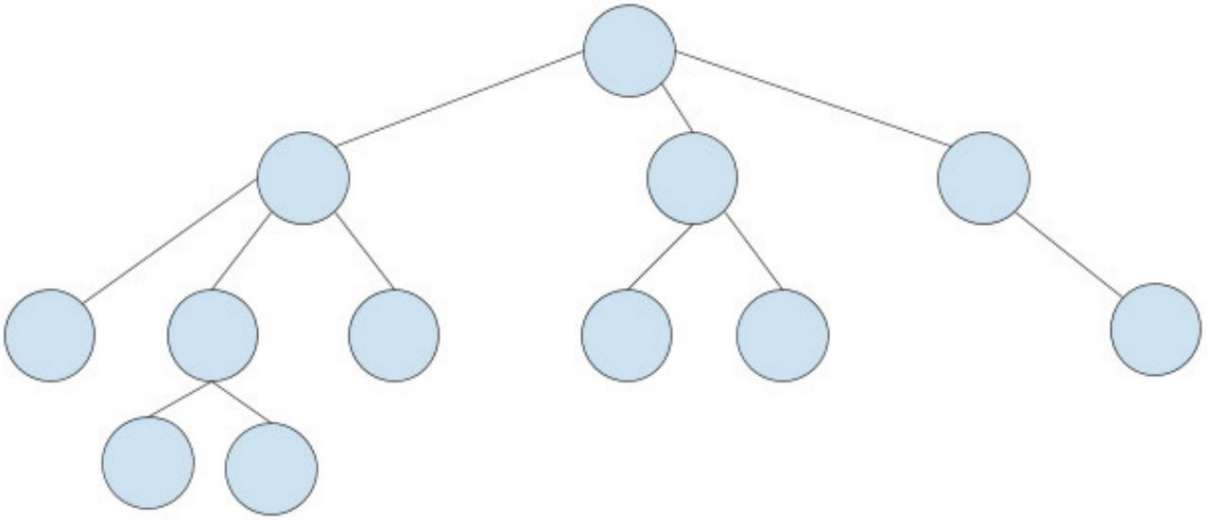
2.2. Estructura de un documento XML

XML es un metalenguaje (lenguaje para la definición de lenguajes de marcado) que nos permite jerarquizar y describir los contenidos dentro del propio documento. Los archivos XML son archivos donde

la información se organiza de forma secuencial y en orden jerárquico.

En un documento XML, la información se organiza de forma jerárquica, de modo que los elementos están relacionados entre sí a través de relaciones de padre, hijo, hermano, parentesco, descendencia, etc. Los documentos XML forman una estructura de árbol que comienza en "la raíz" y ramas a "el

hojas".



2.2.1. Reglas de diseño correctas.

Para que un documento XML sea correcto, se deben cumplir las siguientes condiciones:

- El documento debe estar bien formado.
- El documento debe cumplir con todas las reglas de sintaxis XML.
- El documento debe ajustarse a reglas semánticas, que generalmente se establecen en un esquema XML o en un DTD (Definición de tipo de documento).

2.2.2. sintaxis XML

Un documento XML es una cadena de caracteres. Cada Unicode legal Un carácter (excepto nulo) puede aparecer en un documento XML (1.1) (aunque algunos no se recomiendan).

a) Marcado y contenido

Los caracteres que componen un documento XML se dividen en marcado y contenido, que pueden distinguirse mediante la aplicación de reglas sintácticas simples. Generalmente, las cadenas que constituyen marcado comienzan con el carácter < y terminan con >, o comienzan con el carácter & y terminan con ;. Las cadenas de caracteres que no están marcados son contenido.
Sin embargo, en un CDATA sección, los delimitadores <![CDATA[y]]> se clasifican como marcado, mientras que el texto entre ellos se clasifica como contenido. Además, los espacios en blanco antes y después del elemento más externo se clasifican como marcado.

b) Etiqueta

Una etiqueta es una construcción de marcado que comienza con < y termina con >. Hay tres tipos de etiquetas:

- etiqueta de inicio, como <sección>
- etiqueta final, como </section>
- etiqueta de elemento vacío, como <salto de línea />

c) Elemento

Un elemento es un componente lógico de un documento que comienza con una etiqueta de inicio y termina con una etiqueta de finalización coincidente o que consta únicamente de una etiqueta de elemento vacío. Los caracteres entre la etiqueta inicial y la etiqueta final, si los hay, son el contenido del elemento y pueden contener marcas, incluidos otros elementos, que se denominan elementos secundarios.

Ejemplos:

```
<greeting>¡Hola mundo!</greeting>

<salto de línea />
```

d) Atributo

Un atributo es una construcción de marcado que consta de un par nombre-valor que existe dentro de una etiqueta de inicio o una etiqueta de elemento vacío.

Ejemplos:

```
<img src="valentia.jpg" alt="Valentia" /> <!-- donde están los nombres de los atributos

<step number="3">Conecte A con B.</step> <!-- donde el nombre del atributo es "numb
```

Un atributo XML solo puede tener un valor único y cada atributo puede aparecer como máximo una vez en cada elemento. En la situación común en la que se desea una lista de múltiples valores, esto se debe hacer codificando la lista en un atributo XML bien formado con algún formato más allá de lo que XML se define a sí mismo. Por lo general, se trata de una lista delimitada por comas o punto y coma o, si se sabe que los valores individuales no contienen espacios, se puede utilizar una lista delimitada por espacios.

e) declaración XML

Los documentos XML pueden comenzar con una declaración XML que describe cierta información sobre ellos mismos. La declaración XML no es una etiqueta. Se utiliza para la transmisión de los metadatos de un documento.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?> <!-- donde versión significa versión XML utilizada en t
```

f) Comentarios

Lógicamente, XML permite insertar comentarios dentro de la estructura del documento. Los comentarios pueden aparecer en cualquier parte de un documento fuera de otras marcas. Los comentarios no pueden aparecer antes de la declaración XML.

Ejemplo:

```
<!-- Comentario -->
```

2.2.3. Entidades

Al igual que HTML, XML ofrece métodos (llamados entidades) para hacer referencia a algunas funciones reservadas especiales. caracteres (como un signo mayor que que se utiliza para etiquetas). Hay cinco de estos personajes que debes conocer:

Entidad	Personaje	Descripción
&lt;	<	signo menor que
>	>	signo mayor que
&amperio;	&	ampersand
"	"	Una comilla doble
&apos;	'	Un apóstrofe (o comillas simples)

Aunque solo hay 5 entidades declaradas, se pueden agregar más usando el documento Definición del tipo de documento. También puede utilizar referencias de caracteres numéricos para especificar caracteres especiales; por ejemplo, © es el símbolo "©".

2.2.4. Mostrando XML

XML se utiliza normalmente con fines descriptivos, pero existen formas de mostrar datos XML. Si no define una forma específica para que se represente el XML, el XML sin formato se muestra en el navegador.

Una forma de aplicar estilo a la salida XML es especificar CSS para aplicar al documento utilizando la instrucción de procesamiento de hoja de estilo xml.

Ejemplo:

```
<?xml-stylesheet type="text/css" href="stylesheet.css"?>
```

También existe otra forma más poderosa de mostrar XML: las Transformaciones de lenguaje de hoja de estilo extensible (XSLT) que se puede utilizar para transformar XML a otros lenguajes como HTML. Esto hace que XML sea increíblemente versátil.

Ejemplo:

```
<?xml-stylesheet type="text/xsl" href="transform.xsl"?>
```

2.2.5. Ejemplos

Ejemplo 1. XML incorrecto (mal formado)

```
<?xml versión="1.0" codificación="UTF-8"?>
<mensaje>
<advertencia>
Hola Mundo
<!--faltante </advertencia> --> </
message>
```

Ejemplo 2. XML válido (bien formado)

Ahora veamos una versión corregida de ese mismo documento:

```
<?xml versión="1.0" codificación="UTF-8"?>
<mensaje>
<advertencia>
Hola Mundo
</advertencia>
</mensaje>
```

Un documento que contiene una etiqueta indefinida no es válido. Por ejemplo, si nunca definimos la etiqueta <warning>, el documento anterior no sería válido.

La mayoría de los navegadores ofrecen un depurador que puede identificar documentos XML mal formados.

3. Trabajar con archivos XML

Los archivos XML se pueden utilizar para:

- Proporcionar datos a una BD.
- Almacenar información en bases de datos especiales.
- Archivos de configuración.
- Intercambio de información en entornos web (SOAP).
- Ejecución de comandos en servidores remotos.
- Etc.

Para realizar cualquiera de estas operaciones es necesario un lenguaje de programación que proporcione funcionalidad a XML (XML no es un lenguaje de Turing completo).

3.1. Procesador o analizador XML

El procesador analiza el marcado y pasa información estructurada a una aplicación.
La especificación impone requisitos sobre lo que un procesador XML debe hacer y no hacer, pero la aplicación está fuera de su alcance. El procesador (como lo llama la especificación) a menudo se denomina coloquialmente analizador XML.

Un XML Parser es una clase que tiene como objetivo analizar y clasificar el contenido de un archivo XML extrayendo la información contenida en cada uno de los hashtags y relacionándola según su posición dentro de la jerarquía.

3.1.1. Analizadores secuenciales

Los analizadores secuenciales, que permiten extraer contenido a medida que se descubren los hashtags de apertura y cierre, se denominan analizadores sintácticos. Son analizadores muy rápidos, pero tienen el problema de que cada vez que necesitas acceder a un contenido tienes que volver a leer todo el documento de arriba a abajo.

En Java, el analizador más popular se llama SAX, que significa API simple para XML. Es un analizador ampliamente utilizado en varias bibliotecas de procesamiento de datos XML, pero no se utiliza con frecuencia en aplicaciones finales.

- Ventajas
  - Son muy rápidos.
- Desventajas
  - Deben leer el documento completo en cada consulta.

3.1.2. Analizadores jerárquicos

Generalmente, las aplicaciones finales que tienen que trabajar con datos XML suelen utilizar analizadores jerárquicos, porque además de realizar un análisis secuencial que les permite clasificar el contenido, se almacenan en la RAM siguiendo la estructura jerárquica detectada en el documento. Esto hace que sea mucho más fácil realizar consultas.

Esto facilita enormemente las consultas que deben repetirse varias veces, ya que las estructuras jerárquicas en la RAM tienen un rendimiento de acceso parcial a datos muy eficiente.

El formato de la estructura donde se almacena la información en la RAM ha sido especificado por el organismo internacional W3C (World Wide Web Consortium) y se conoce comúnmente como DOM (Document Object Model). Es una estructura que HTML y JavaScript han hecho muy popular y es una especificación que Java materializa en forma de interfaces. El principal se llama Documento y representa un documento XML completo. Como es una interfaz, puede ser implementada por varias clases.

- Ventajas
  - Son ideales para aplicaciones que requieren consulta continua de datos.
- Desventajas
  - Almacenan todos los datos del documento XML en la memoria (RAM) dentro de una estructura jerárquica.

4. Analizador SAX

El grupo de correo XML-DEV desarrolló una API simple para XML, también llamada SAX, que es un algoritmo en línea controlado por eventos para analizar documentos XML. SAX (API simple para XML) es un conjunto de clases e interfaces que proporcionan una herramienta para procesar documentos XML. La API SAX está completamente incluida en el JRE.

SAX es una forma de leer datos de un documento XML que es una alternativa al mecanismo del Modelo de objetos de documento (DOM). Mientras que el DOM trabaja en el documento como un todo, creando todo el árbol de sintaxis abstracta de un documento XML para comodidad del usuario, los analizadores SAX trabajan en cada elemento del documento XML de forma secuencial, emitiendo eventos de análisis mientras pasan por el flujo de entrada en un único aprobar. A diferencia de DOM, SAX no tiene una especificación formal.

SAX es una interfaz de programación para procesar archivos XML basados en eventos. La contraparte del DOM, SAX, tiene una forma muy diferente de leer el código XML. La implementación Java de SAX se considera el estándar de facto. SAX procesa el estado de los documentos de forma independiente, a diferencia de DOM, que se utiliza para el procesamiento de documentos XML dependiente del estado.

En pocas palabras, SAX es más complejo de programar que DOM, sin embargo, ofrece un menor consumo de memoria, lo que lo hace más adecuado para procesar archivos XML de gran tamaño.

Ventajas

- Analiza el archivo XML como una secuencia en lugar de asignar RAM para el archivo completo.
- Desde entonces, utiliza menos memoria y es más rápido que el DOM Parser porque el archivo completo no se almacena en la memoria.
- Por lo tanto, se considera útil para analizar archivos XML de gran tamaño.

Desventajas

- No hay métodos de actualización en SAX Parser. Dado que el archivo completo no se guarda en la memoria, es posible acceder a los elementos sólo de forma secuencial y no se puede acceder a los elementos de forma aleatoria.

4.1. Leer archivos XML con Java

SAX significa "API simple para XML". A diferencia de un analizador DOM que carga documentos en la memoria, SAX es un análisis basado en eventos. Funciona en los eventos cuando ocurre un evento y llama a algunos métodos de devolución de llamada. El método parse() de SAXParser iniciará el procesamiento XML.

El analizador SAX funciona en los siguientes eventos:

• iniciar documento

• elemento de inicio

• caracteres

• comentarios

• instrucciones de procesamiento

• elemento final

• documento final

La lectura de un XML con SAX produce eventos que provocan que se llamen a diferentes métodos.
Métodos comúnmente utilizados de SAX XML Parser:

Método	Descripción
iniciarDocumento()	Se llama al principio del documento xml.
documento final()	Se llama al final del documento xml.
startElement(String uri, String localName, String qName, Atributos atributos)	Se llama al comienzo de un elemento.
endElement(cadena uri, cadena nombre local, cadena qnombre)	Se llama al final de un elemento.
caracteres(char[] ch, int inicio, int longitud)	Se llama cuando se encuentran datos de texto entre las etiquetas de inicio y fin de un elemento.

Todos estos métodos están definidos en DefaultHandler. Los siguientes son los pasos para leer un archivo usando SAX:

1. Cree una clase para extender org.xml.sax.helpers.DefaultHandler y anule el
   Métodos startElement, endElement y caracteres para imprimir todos los elementos, atributos, comentarios y textos XML.

2. Crear una nueva clase que se encargará de la ejecución principal de nuestra aplicación (MAIN):
    1. Cree una instancia de SAXParserFactory, según lo determine la configuración de la propiedad del sistema javax.xml.parsers.SAXParserFactory. La fábrica que se creará se configura para admitir espacios de nombres XML configurando setNamespaceAware en verdadero y luego se obtiene una instancia de SAXParser de la fábrica invocando su método newSAXParser().
    2. Cree un objeto de la clase DefaultHander que acabamos de crear.
    3. Utilice el método de análisis de nuestro SAXParserFactory para leer el archivo. Pasamos como parámetros el nombre del archivo XML y el objeto de la clase DefaultHander.

4.2. ejemplo de lectura

Veamos un pequeño ejemplo en Java.

1) Lo primero que haremos será importar todas las clases e interfaces necesarias.

```
importar org.xml.sax.Attributes; importar
org.xml.sax.helpers.DefaultHandler; importar
org.xml.sax.SAXException;
```

2) Luego creamos una clase que extiende DefaultHandler para manejar eventos de lectura. Esta clase proporciona la implementación predeterminada para todos sus métodos, el programador debe definir esos métodos que serán utilizados por la aplicación. Esta es la clase que ampliaremos para crear nuestro analizador XML.

```
/*
 *   ----------------------------
 * Manejador de SAX para estudiantes
 *   ----------------------------
 */
la clase pública PrintAllHandlerSax extiende DefaultHandler{

    //Esta es la lista que se completará al analizar el XML private StringBuilder
    sbCurrentValue = new StringBuilder();

    @Anular
    documento de inicio público vacío () {
        System.out.println("Iniciar documento");
    }

    @Override
    public void endDocument() lanza SAXException {
```

```
            System.out.println(" Documento final");
        }

        @Override
        public void startElement(String uri, String localName, String qName, Atributos a
            //restablecer el valor de la
            etiqueta sbCurrentValue.setLength(0);
            System.out.printf(" Elemento de inicio: %s%n", qName);
        }

        @Anular
        endElement público vacío (String uri, String localName, String qName) lanza SAXExce
            // maneja el valor según el elemento al que pertenece if
            (qName.equalsIgnoreCase("studentID")) {
                System.out.printf("              ID: %s%n", sbCurrentValue.toString());
            }
            System.out.printf(" Elemento final: %s%n", qName);
        }

        /*
         * Esto se llamará cada vez que el analizador encuentre un nodo de valor */

        @Override
        caracteres públicos vacíos (char[] ch, int start, int length) lanza SAXException
            // El método caracteres() se puede llamar varias veces para un solo guiño de texto // Es posible
            que falten algunos valores si se asignan a una nueva cadena
            sbCurrentValue.append(ch, start, length);
        }

    }
```

3) Una vez que tengamos nuestro propio handler, creamos una nueva clase que se encargará de la ejecución principal de nuestra aplicación (MAIN)

- Cree una instancia de SAXParserFactory, según lo determine la configuración de la propiedad del sistema javax.xml.parsers.SAXParserFactory. La fábrica a crear es

se configura para admitir espacios de nombres XML estableciendo setNamespaceAware en verdadero, y luego se obtiene una instancia de SAXParser de fábrica invocando su método newSAXParser().

```
SAXParser saxParser = saxFactory.newSAXParser();
```

• Cree un objeto de la clase DefaultHander que acabamos de crear.

```
PrintAllHandlerSax saxHandler = nuevo PrintAllHandlerSax();
```

• Debemos decirle a nuestro analizador XML que use una instancia del mismo para poder leer el archivo.
Una vez que tengamos el código fuente creado y el ContentHandler asignado procederemos a leer el archivo simplemente usando el método Parse de nuestro procesador XML.

```
saxParser.parse(XML_FILENAME, saxHandler);
```

Código completo de la clase principal:

```
importar java.io.FileNotFoundException; importar
java.io.IOException;

importar org.xml.sax.SAXException;
importar javax.xml.parsers.ParserConfigurationException; importar
javax.xml.parsers.SAXParser; importar
javax.xml.parsers.SAXParserFactory;

/*
 *   ----------------------------------
 * Lectura de un archivo XML con SAX
```

```
 *   ----------------------------------------
 */

clase pública StudentReaderSAX {

    /**
     *   ----------------------------------------
     * VARIABLES Y CONSTANTES GLOBALES
     *   ----------------------------------------
     */
    //Constante de nombre de archivo. Suponemos que nuestro archivo está ubicado en la carpeta
    del proyecto static final String XML_FILENAME = "students.xml";

    /**
     *   ----------------------------------------
     * PROGRAMA PRINCIPAL
     *   ----------------------------------------
     */
    principal vacío estático público (cadena [] stArgs) {

        System.out.println("");
        SAXParserFactory saxFactory = SAXParserFactory.newInstance(); intente

        { SAXParser saxParser = saxFactory.newSAXParser();
            PrintAllHandlerSax saxHandler = nuevo PrintAllHandlerSax();
            saxParser.parse(XML_FILENAME, saxHandler);
        /*
          * NO OLVIDE ESTABLECER AQUÍ EL MANEJO DE EXCEPCIONES ADECUADO
          */
        } catch (ParserConfigurationException | SAXException saxo) {
            System.out.println("Archivo no encontrado");
        } catch (FileNotFoundException fina) {
            System.out.println("Archivo no encontrado"); }
        catch (IOException ioe)
            { System.out.println(" Error de IO");
        }
```

```
        }
    }
```

5. analizador DOM

El DOM define un estándar para acceder y manipular documentos: "El Modelo de Objetos de Documento (DOM) del W3C es una plataforma y una interfaz neutral en cuanto al idioma que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento".

XML DOM define una forma estándar de acceder y manipular documentos XML.
DOM almacena todos los datos del documento XML en la memoria en una estructura de árbol jerárquica.

Ventajas:

- En pocas palabras, un analizador DOM trabaja en todo el documento XML.
- Es ideal para aplicaciones que requieren consulta continua de los datos.

Desventajas:

- Carga todo el documento en la memoria. Este tipo de procesamiento requiere más memoria. recursos y tiempo que SAX.
- No apto para documentos grandes.

5.1. La estructura DOM

La estructura DOM toma la forma de un árbol, donde cada parte del XML se representa como un nodo. Dependiendo de la posición en el documento XML hablaremos de diferentes tipos de nodos. El nodo principal que representa todo el XML se llama documento, y los distintos hashtags, incluida la etiqueta raíz, se conocen como nodos de elementos. El contenido textual de un hashtag se instancia como un nodo de tipo TextElement y los atributos como nodos de tipo Attribute.

Cada nodo específico tiene métodos para acceder a sus datos específicos (nombre, valor, nodos secundarios, nodos pares, etc.).

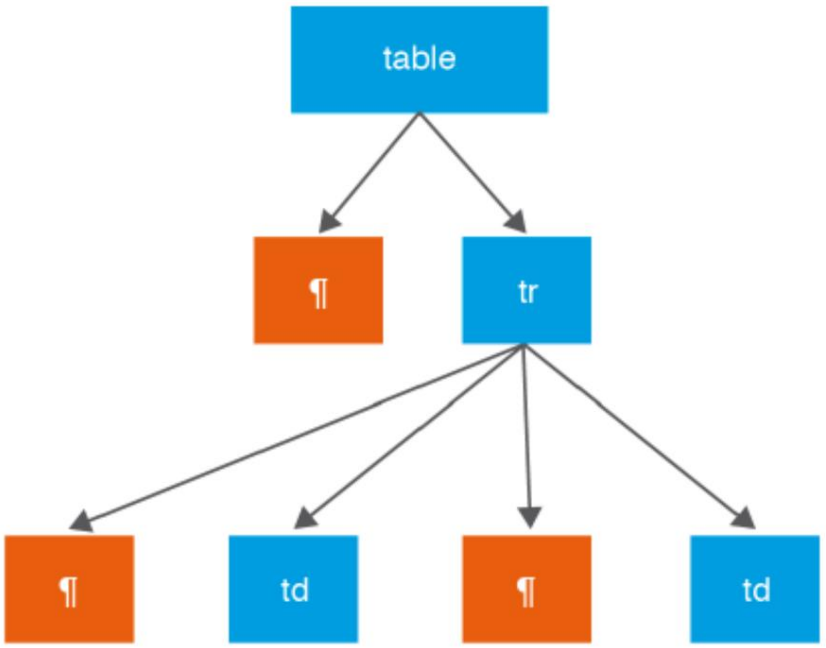
El DOM resultante obtenido de un XML acaba siendo una copia exacta del archivo, pero organizado de otra forma. Tanto en el XML como en el DOM habrá información no visible, como retornos de carro, que hay que tener en cuenta para saber procesar correctamente el contenido y evitar sorpresas ininteligibles.

a) Representación de un objeto DOM con retornos de carro

Imaginemos que tenemos un documento XML con el siguiente contenido:

```
<tabla>
<tr>
<td></td>
<td></td>
</tr>
</table>
```

La siguiente figura muestra la representación que tendría el objeto DOM, una vez copiado en memoria. Cabe señalar que el elemento de la tabla tendrá dos hijos. Se almacenará el retorno de carro que coloca el hashtag tr en la siguiente línea; en el otro encontraremos el hashtag tr. Lo mismo ocurre con los hijos de tr, antes de cada nodo td encontraremos un retorno de carro.

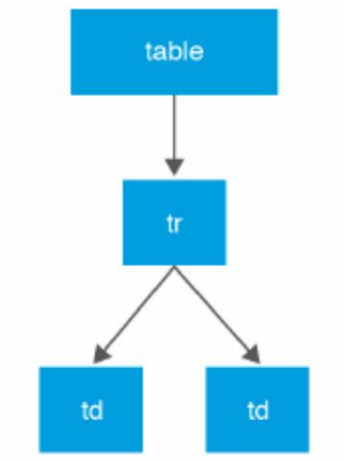


b) Representación de un objeto DOM sin retornos de carro

Por otro lado, si hubiéramos partido de un XML equivalente sin retornos de carro, el resultado también habría sido diferente.

```
<tabla><tr><td></td><td></td></tr></table>
```

El documento XML anterior, sin retornos de carro, daría la representación del objeto DOM que puede ver en la figura siguiente.



La ausencia de retornos de carro en el fichero implica también la ausencia de nodos que contengan el

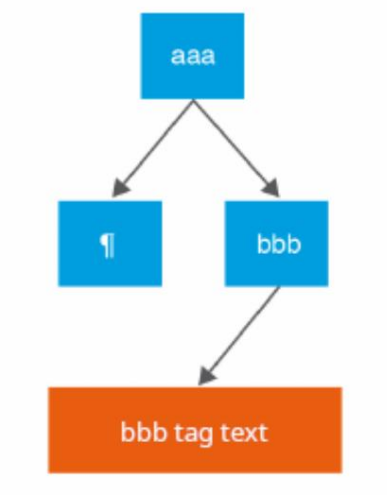
retornos de carro en la estructura DOM.

c) Representación DOM de un hashtag con texto

Otro aspecto a tener en cuenta al mapear archivos XML es que el contenido textual de los hashtags se captura en el DOM como un nodo secundario del contenedor de hashtags. En otras palabras, para obtener el texto de un hashtag debes obtener el primer hijo del hashtag.

```
<aaa>
<bbb>
texto de la etiqueta
bbb </
bbb> </aaa>
```

La siguiente figura ilustra la representación DOM de un hashtag que contiene texto.



La interfaz del Documento proporciona un conjunto de métodos para seleccionar diferentes partes del árbol a partir del nombre del hashtag o un atributo de identificación. Las partes del árbol se devuelven como objetos Elemento, que representan un nodo y todos sus hijos. De esta manera, podemos explorar partes del árbol sin tener que pasar por todos los nodos.

5.2. Administrar archivos XML con Java

Para trabajar con DOM en Java haremos uso de los paquetes

• org.w3c.dom (Contenido en el JSDK)

• javax.xml.parsers (De la API estándar de Java)

Proporcionan dos clases abstractas que debemos ampliar para trabajar con DOM.

• Fábrica de documentos

• Generador de documentos

DOM no define ningún mecanismo para generar un archivo XML a partir de un árbol DOM. Para esto nosotros utilizará el paquete javax.xml.transform. Este paquete permite especificar una fuente y un resultado:

• Archivos

• Flujos de datos

• Nodos DOM

• Etc.

Los programas Java que utilizan DOM necesitan utilizar las siguientes interfaces:

Interfaz	Descripción
Documento	Es un objeto que representa el documento XML. Permite la creación de nuevos nodos.
Elemento	Representa cada uno de los elementos del documento.
Nodo	Representa cualquier nodo del documento.
Lista de nodos	Contiene una lista de nodos secundarios de un nodo determinado.
Atributo	Permite el acceso a los atributos de un nodo.
Texto	Representa los datos de caracteres de un nodo.
Datos de carácter	Representa los datos de caracteres presentes en el documento.
Tipo de Documento	Proporciona información contenida en la etiqueta <!DOCTYPE>.

5.3. ejemplo de lectura

Para leer un documento XML usando DOM debemos crear una instancia de DocumentBuilderFactory para construir el analizador y, a través de él, cargar el documento.

```
//https://howtodoinjava.com/java/xml/read-xml-dom-parser-example/ //Creamos el
DocumentBuilder para poder obtener el Documento DocumentBuilderFactory
dbfEmployee= DocumentBuilderFactory.newInstance(); DocumentBuilder
dbEmployee=dbfEmployee.newDocumentBuilder(); //Leemos el Documento del
archivo Documento docEmployee=
dbEmployee.parse(new File("Employees.xml")); //Estandarizamos el documento para
evitar errores de lectura docEmployee.getDocumentElement().normalize();
```

Luego creamos una lista con todos los elementos utilizados usando la clase NodeList.

```
//Muestra el nombre del elemento raíz
System.out.println("El elemento raíz es "+
docEmployee.getDocumentElement().getNodeName());

//Creamos una lista de todos los nodos Employee NodeList
nlstEmployee= docEmployee.getElementsByTagName("employee"); //Mostramos el
número de elementos Empleado que hemos encontrado System.out.println("
Se ha encontrado lo siguiente "+nlstEmployee.getLength()+" emplear
```

Finalmente, usando un bucle, recorremos NodeList y mostramos su contenido.

```
//Recorremos la lista for(int ii=0;
ii<nlstEmployee.getLength();ii++) {
```

```
        //Obtenemos el primer nodo de la lista
        Node nodeEmployee= nlstEmployee.item(ii); //En
        caso de que ese nodo sea un Elemento
        if(nodeEmployee.getNodeType()==Node.ELEMENT_NODE) {
            //Creamos el elemento empleado y leemos su información Element
            objEmployee= (Element)nodeEmployee;
            System.out.print("ID: " + objEmployee.getElementsByTagName("id").
                    item(0).getTextContent() );
            System.out.print("\tName: " + objEmployee.getElementsByTagName("nombre")
                    elemento(0).getTextContent() );
            System.out.println("\tApellido: "          + objEmployee.getElementsByTagName("apellido"
                    elemento(0).getTextContent() );
        }
    }
```

5.4. ejemplo de escritura

A continuación vamos a crear un archivo XML de empleados.

Lo primero que debe hacer es importar los paquetes necesarios.

```
importar org.w3c.dom.*;
importar javax.xml.transform.*;
importar javax.xml.transform.dom.*;
importar javax.xml.transform.stream.*;
importar javax.xml.parsers.*;
importar java.io.*;
```

Luego crearemos el Documento en el que vamos a insertar a nuestros empleados.

```
//Creamos un DocumentBuilder usando DocumentBuilderFactory
DocumentBuilderFactory dbfEmployee= DocumentBuilderFactory.newInstance();
DocumentBuilder dbEmployee=dbfEmployee.newDocumentBuilder();

/* Creamos un documento vacío
 * Nombre --> Registro de Empleado
 * Nodo raíz --> Empleados */

DOMImplementación domImplement= dbEmployee.getDOMImplementation();
Documento docEmployee = domImplement.createDocument(null, "empleados", null); //Asignamos
la versión XML
docEmployee.setXmlVersion("1.0");
```

Y crearemos a cada uno de los empleados con sus datos.

```
//Creamos un elemento de nodo
Empleado objEmployee= docEmployee.createElement("employee"); //Lo
agregamos como hijo de los empleados
docEmployee.getDocumentElement().appendChild(objEmployee); //
Creamos el elemento ID del
nodo eNodeID= docEmployee.createElement("id"); //Crea el
nodo de texto con el valor de ID Text txtNode=
docEmployee.createTextNode("01"); //Agregamos el valor
al nodo ID eNodeID.appendChild(txtNode); //
Agregamos el nodo ID a Empleado
objEmployee.appendChild(eNodeID);
Elemento eNodeName=
docEmployee.createElement("nombre"); txtNode=
docEmpleado.createTextNode("Pepe");
eNodeName.appendChild(txtNode);
objEmployee.appendChild(eNodeName);
Elemento eNodoApellido= docEmpleado.createElement("apellido");
txtNode= docEmpleado.createTextNode("García");
eNodeApellido.appendChild(txtNode);
objEmployee.appendChild(eNodoApellido);
```

Por último, debemos escribir nuestro Documento en el disco.

```
/*
 *Por último, para guardar el documento en disco debemos:
 *
 * 1. Crear la fuente de datos (Nuestro Documento)
 * 2. Cree el resultado (el archivo de destino)
 * 3. Crear una TransformerFactory
 * 4. Realizar la transformación
 */
Fuente srcEmployee= new DOMSource(docEmpleado);
Resultado resultFile= new StreamResult(nuevo archivo ("Empleados.xml"));
Transformador transfEmployee = TransformerFactory.newInstance().newTransformer()
```

```
transfEmployee.transform(srcEmployee, resultFile);
```

De la misma manera que usamos la clase Transform para enviar nuestro Documento a un archivo, podemos usarlo para mostrarlo a través de la salida estándar.

```
/*
 * Mostramos el resultado por la salida estándar */

Resultado resultStdOutput = nuevo StreamResult(System.out);
transfEmployee.transform(srcEmployee, resultStdOutput);
```

6. lenguaje XSL

XSL (que significa eXtensible Stylesheet Language), es una familia de recomendaciones para definir la transformación y presentación de documentos XML. Es decir, XSL es un lenguaje de estilo para XML.

Ha sido desarrollado por el W3C. Consta de tres partes:

- Transformaciones XSL (XSLT). Un lenguaje para transformar XML;
- El lenguaje de ruta XML (XPath). Un lenguaje de expresión utilizado por XSLT (y muchos otros lenguajes) para acceder o hacer referencia a partes de un documento XML.
- Objetos de formato XSL (XSL-FO). Un vocabulario XML para especificar el formato. semántica.

6.1. ¿Qué es XSLT?

XSLT (eXtensible Stylesheet Language Transformations) es un lenguaje basado en XML que se utiliza, junto con software de procesamiento especializado, para la transformación de documentos XML.

Aunque el proceso se denomina "transformación", el documento original no se modifica; más bien, se crea un nuevo documento XML basado en el contenido de un documento existente. Luego, el procesador puede serializar (generar) el nuevo documento en sintaxis XML estándar o en otro formato, como HTML o texto sin formato.

XSLT se utiliza con mayor frecuencia para convertir datos entre diferentes esquemas XML o para convertir XML datos en páginas web o documentos PDF.

Para obtener más información, consulte: Referencia de elementos XSLT.

7. Bibliografía

Fuentes

•Wikipedia. XML. https://en.wikipedia.org/wiki/XML

• Desarrollador Mozilla. Introducción a XML. https://developer.mozilla.org/en-US/docs/Web/XML/XML_introducción

• Escuelas W3. Introducción a XML. https://www.w3schools.com/XML/xml_whatis.asp

• Josep Cañellas Bornas, Isidre Guixà Miranda. Acceso a datos. Desarrollo de aplicaciones multiplataforma. Creative Commons. Departamento de Enseñanza, Institut Obert de Catalunya. Depósito legal: B. 29430-2013. https://ioc.xtec.cat/educacio/recursos

• Alberto Oliva Molina. Acceso a datos. UD 1. Trabajo con ficheros XML. IES Tubalcaín. Tarazona (Zaragoza, España).

• Documentación de Oracle. SAXÓFONO. https://docs.oracle.com/javase/tutorial/jaxp/sax/index.html

• Geeksforgeeks. ¿Qué es SAX en XML? https://www.geeksforgeeks.org/what-is-sax-in-xml/

• Escuelas W3. Analizador SAX XML en Java. https://www.w3schools.blog/sax-xml-parser-in-java ejemplo-tutorial

• Documentación de Oracle. DOM. https://docs.oracle.com/javase/tutorial/jaxp/dom/index.html

• Escuelas W3. Introducción a XSLT. https://www.w3schools.com/xml/xsl_intro.asp