XML

Modelo de procesamiento DOM

Aplicaciones Distribuidas

Curso 2025/26

DIS
Departamento de
Informática y Sistemas

UNIVERSIDAD
DE MURCIA

Procesamiento XML

- **De entrada**:
  - Extraer información de un documento.
  - Habitualmente también interesa comprobar previamente si el documento es válido.
- **De salida**:
  - Generar un documento XML.

- Procesamiento **mixto**:
  - Actualizar un documento existente.
  - Transformar un documento de entrada en otro de salida (con distinto esquema).

API JAXP

- **JAXP** (*Java API for XML Processing*)
- Incluida en la versión estándar de Java

- Modelos de procesamiento:
  - **SAX:** basado en eventos.
  - **DOM**: representación del documento como un árbol.
  - **StAX**: procesamiento basado en un cursor (iterador).

- El API también incorpora los estándares:
  - **XSLT** para la transformación de documentos XML.
  - **XPath** para consultas.

Procesamiento DOM

- **DOM** *(Document Object Model)*
- El procesamiento DOM construye la representación en forma de árbol de un documento XML.
- Ofrece la funcionalidad para:
  - Consultar, recorrer y **modificar** el árbol.
  - Construcción de un nuevo árbol DOM.
- Inconvenientes:
  - Si queremos procesar sólo una pequeña parte del documento, es ineficiente construir el árbol completo.

DOM – Construcción del analizador

- La construcción del analizador sigue la estrategia de construcción de JAXP:

```
// 1. Obtener una factoría

DocumentBuilderFactory factoria =
                    DocumentBuilderFactory.newInstance();

// 2. Pedir a la factoría la construcción del analizador

DocumentBuilder analizador = factoria.newDocumentBuilder();

// 3. Analizar el documento

Document documento = analizador.parse("documento.xml");
```

- El resultado del análisis es un objeto (**`Document`**) que representa el documento XML.

DOM – Nodos del árbol

- Tipos de nodos del árbol DOM:
  - `Document`: representa el documento completo.
    - Ofrece acceso al elemento raíz (`getDocumentElement`).
  - `Element`: representa un elemento.
    - Ofrece acceso a sus atributos.
  - `Attr`: nodo atributo.
  - `Text`: nodo que contiene el texto de un elemento.
  - …
- En general, ofrece tipos para todas las declaraciones de XML → `ProcessingInstruction`, etc.

DOM – Elementos y atributos

- Tipo **`Element`**:
  - **`getTagName`**, `getLocalName`, `getNamespaceURI`: nombre del elemento con prefijo, sin prefijo y su espacio de nombres (depende del tipo de procesamiento).
  - `getTextContent`: El contenido textual del elemento **y de todos sus descendientes**.
  - Acceso a los atributos por nombre:
    - `getAttribute(`*`nombre`*`)`: retorna su contenido textual.
    - `getAttributeNode(`*`nombre`*`)`: retorna su nodo (tipo `Attr`).
    - `getAttributes()`: retorna todos los atributos.
- Tipo **`Attr`**:
  - `getName()`: nombre.
  - `getValue()`: contenido textual.

DOM – Consultas

- **En cualquier nodo del árbol DOM**, obtenemos los elementos que contiene de un cierto tipo **`getElementsByTagName`**

```
NodeList elementos = nodo.getElementsByTagName("nif");

for (int i = 0; i < elementos.getLength() ; i++) {

   Element elemento = (Element) elementos.item(i);
   // ...
}
```

- **Nota**: a pesar de que la consulta es específica para elementos, el tipo de retorno es una lista de nodos (`NodeList`). Por tanto, al recuperar el elemento hay que hacer un **casting**.

DOM – Consultas

- **Recorrido del árbol**:
  - `getFirstChild`, `getLastChild`, `getNextSibling`, `getPreviousSibling`, `getParentNode`, `getChildNodes`.
  - Los más útiles son las que ofrecen acceso al nodo padre (`getParentNode`) y a los hijos (`getChildNodes`).

- **Nota 1:** las operaciones de recorrido devuelven objetos de tipo `Node`. Este tipo ofrece el método `getNodeType()` y las constantes `ELEMENT_NODE`, `ATTRIBUTE_NODE`, etc. para consultar el tipo y hacer un **casting** al tipo correspondiente.

- **Nota 2**: la operación `getElementsByTagName` es un modo sencillo de recuperar elementos y a partir de un elemento sus atributos. Suele ser preferible a recorrer el árbol.

DOM – Construcción

- El tipo **`Document`** ofrece **métodos factoría** para construir todos los tipos de nodos. Cabe destacar:
  - El contenido textual de un elemento se añade como hijo, aunque hay un método que evita manejar los nodos textuales.
  - Los nodos atributos se establecen con `setAttribute` sobre el elemento.

- **Nota**: utilizando el analizador DOM (`DocumentBuilder`) también es posible construir un árbol DOM vacío.

DOM – Construcción

- Ejemplo:
  - `<nombre nif='23456789'>Pepe</nombre>`

```
// 1. Crea los nodos
Element elementoNombre = documento.createElement("nombre");
Text textoNombre = documento.createTextNode("Pepe");

// 2. Los enlaza
elementoNombre.appendChild(textoNombre);

// 3. Establece los atributos del elemento
elementoNombre.setAttribute("nif", "23456789");
```

  - **Nota**: también están disponibles los métodos `getTextContent` y `setTextContent` que permiten acceder y establecer el contenido textual de un elemento sin necesidad de manejar el nodo de texto.

DOM – Modificación

- Para que el nuevo elemento forme parte del documento hay que **situar el nodo en el árbol**:
  - Establecerlo como último hijo de algún nodo (`appendChild`)
  - Si se requiere situar en relación a otro, método `insertBefore(`*`nuevo, referencia`*`)` sobre el nodo padre.

- La **eliminación** de un nodo es solicitada a su padre:

  `removeChild(`*`nodo`*`)`

- **Sustituir un nodo** por otro, aplicado sobre el nodo padre:

  `replaceChild(`*`nuevo, antiguo`*`)`

DOM – Ejemplo

- Tomando como referencia el caso de estudio del acta, el siguiente ejemplo muestra cómo **añadir una calificación**.
  - En primer lugar, creamos la calificación

```
Element calificacion = documento.createElement("calificacion");

nif = documento.createElement("nif");
nif.setTextContent("22334312C");

nota = documento.createElement("nota");
nota.setTextContent("9");

calificacion.appendChild(nif);
calificacion.appendChild(nota);
```

DOM – Ejemplo

- Ejemplo: añadir la calificación al documento
  - A continuación, la situamos en el documento

```
// La situamos en el documento antes de las diligencias.

NodeList diligencias = documento.getElementsByTagName("diligencia");

// Si no hay diligencias, se puede colocar al final del documento
if (diligencias.getLength() == 0) {

    documento.getDocumentElement().appendChild(calificacion);
}
else {

    // Obtener como referencia la primera diligencia
    Element diligenciaReferencia = (Element) diligencias.item(0);
    Element padre = (Element) diligenciaReferencia.getParentNode();

    padre.insertBefore(calificacion, diligenciaReferencia);
}
```

DOM – Ejemplo

- Ejemplo 2: crear una diligencia
  - En el ejemplo es más sencillo situar una diligencia que una calificación

```
Element diligencia = documento.createElement("diligencia");
diligencia.setAttribute("fecha", "2023-09-01");

Element nif = documento.createElement("nif");
nif.setTextContent("22334312C");

Element nota = documento.createElement("nota");
nota.setTextContent("10");

diligencia.appendChild(nif);
diligencia.appendChild(nota);

// Situarla en el árbol como último elemento del documento

documento.getDocumentElement().appendChild(diligencia);
```

DOM – Almacenar en fichero

- El API DOM no ofrece funcionalidad para almacenar un árbol DOM en un fichero.
- Se utiliza un **transformador** (paq. `javax.xml.transform`):

```
// 1. Construye la factoría de transformación y obtiene un
// transformador

TransformerFactory tFactoria = TransformerFactory.newInstance();
Transformer transformacion = tFactoria.newTransformer();

// 2. Establece la entrada, como un árbol DOM
Source input = new DOMSource(documento);

// 3. Establece la salida, un fichero en disco
Result output = new StreamResult(fichero);

// 4. Aplica la transformación
transformacion.transform(input, output);
```

Referencias

- Documentación oficial de Oracle:
  - http://docs.oracle.com/javase/tutorial/jaxp/
  - http://docs.oracle.com/javase/tutorial/jaxp/dom/

- Tutoriales básicos:
  - http://www.mkyong.com/tutorials/java-xml-tutorials