índice

Tema 3 Trabajo con ficheros XML

Acceso a datos con DOM y SAX

DOM Y SAX: Son herramientas que nos ofrecen la posibilidad de leer ficheros XML. Estas herramientas se dedican a verificar si sintácticamente son ficheros válidos. Son los llamados "parsers" o analizadores.

DOM: carga el xml completo, obtiene el árbol de nodos ya formado, más lento y menos versátil.

SAX: tiene en memoria solo una parte del nodo, más rápido pero menos potente más funcional y versátil.

SAX	DOM
Basado en eventos	Carga el fichero completo
Va analizando nodo por nodo	Búsqueda de tags hacia delante y hacia atrás
No carga en totalidad el fichero	Estructuras de árbol
Rapidez en tiempo de ejecución	Más lento en tiempo de ejecución
Es sólo lectura	Se pueden insertar o eliminar nodos
Es sólo lectura	DataInputStream DataOutputStream, PrintStream

Conversión de ficheros XML

Existen muchas librerías que se utilizan como parsers de ficheros XML ejemplo: javax.xml.parsers

previamente tenemos que tener el xml previamente realizado y validado con DTD

parser DOM

Todo se tiene que hacer dentro de Try-catch.

1. Crear una instancia de DocumentBuilderFactory

- setValidating(true) nos aseguramos de que se valide el XML con su DTD

- setIgnoringElementContentWhitespace(true) que se ignore los espacios en blanco

2. Crear un objeto DocumentBuilder a partir de la factory creada

3. Ruta del xml con File

4. Parsear el XML en un objeto Document

5. Crear un objeto Xpath para hacer consultas en el XML (Explicado en el apartado xpath)

```java
public class parserDOM {
    public static void main(String[] args) {
        try {
            // Crear una instancia de DocumentBuilderFactory
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            factory.setValidating(true);
            factory.setIgnoringElementContentWhitespace(true);

            // Crear un objeto DocumentBuilder
            DocumentBuilder builder = factory.newDocumentBuilder();

            // Ruta del XML que queremos analizar
            File file = new File("playlist.xml");

            // Parsear el XML y obtener un objeto Document
            Document doc = builder.parse(file);


            // Crear un objeto XPATH para consultar el documento XML
            XPath xPath = XPathFactory.newInstance().newXPath();
            // Definir la expresión XPath para obtener la lista de nodos
            String consulta = "/playlist/cancion";
            NodeList listaNodos = (NodeList) xPath.compile(consulta).evaluate(doc, XPathConstants.NODESET);

            // Iterar la lista
            for (int i = 0; i < listaNodos.getLength(); i++) {
                Node nNode = listaNodos.item(i);
                System.out.println("\nElemento Actual: " + nNode.getNodeName());

                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    System.out.println("Título: " + eElement.getElementsByTagName("titulo").item(0).getTextContent());

                    // Ahora accedemos al nombre del artista correctamente
                    Element artistaElement = (Element) eElement.getElementsByTagName("artista").item(0);
                    String nombreArtista = artistaElement.getElementsByTagName("nombre").item(0).getTextContent();
                    System.out.println("Artista: " + nombreArtista);

                    // Si necesitas la nacionalidad
                    String nacionalidadArtista = artistaElement.getElementsByTagName("nacionalidad").item(0)
                            .getTextContent();
                    System.out.println("Nacionalidad: " + nacionalidadArtista);

                    System.out.println("=========="); // Separador
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Parser SAX

Todo se tiene que hacer dentro de Try-catch.

1. **Configurar el analizador SAX** crear el SAXParserFactory y el SAXParser

2. **Definir la clase interna que herede de DefaultHander**

3. **Crear dentro de la clase las variables** para ver el estado de los elementos a procesar

4. **Método startElement** se llama a este método cuando el analizador encuentra un elementos

5. **Método characters** este método se llama para el contenido de los elementos extrae y muestra el contenido

6. **Método endElement** se llama cuando se encuentra un elemento de cierre

7. **Analizar el archivo XML** con parse

```java
public class parserSAX {
    public static void main(String[] args) {
        try {

            // Configura la fábrica del analizador SAX
            SAXParserFactory factory = SAXParserFactory.newInstance();
            // Crea una instancia del analizador SAX
            SAXParser saxParser = factory.newSAXParser();
            // Crea un manejador personalizado
            SAXHandler handler = new SAXHandler();
            // Analiza el archivo XML con el manejador
            saxParser.parse(new File("T3\\Tarea_opcional_XML_con_SAX\\playlist.xml"), handler);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Clase interna que actúa como un manejador personalizado para eventos SAX
    public static class SAXHandler extends DefaultHandler {
        // Controla si estamos dentro de un elemento 'playlist'
        private boolean inPlaylist = false;
        // Controla si estamos dentro de un elemento 'cancion'
        private boolean inCancion = false;
        // Almacena el nombre del elemento actual
        private String currentElement;


        /*El método startElement se llama cuando el analizador encuentra un elemento de apertura en el documento XML.
        En este método, se almacena el nombre del elemento actual en la variable currentElement. */
        public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
             // Guarda el nombre del elemento actual
            currentElement = qName;
            // Indica que estamos dentro de un elemento se cambia a true
            if ("playlist".equals(qName)) {
                inPlaylist = true;
                System.out.println("\nElemento Actual: " + qName);

            } else if ("cancion".equals(qName)) {
                inCancion = true;
                System.out.println("-----");
            }
        }

        /* El método characters se llama cuando el analizador encuentra el contenido dentro de un elemento.
        En este método, se obtiene el valor del contenido y se imprime. */
        public void characters(char[] ch, int start, int length) throws SAXException {
            String value = new String(ch, start, length).trim();
            // Imprime el contenido del elemento
            if (!value.isEmpty()) {
                System.out.println(currentElement + ": " + value);
            }
        }

        /*El método endElement se llama cuando el analizador encuentra un elemento de cierre en el documento XML.
        En este método, se indica que se ha salido del elemento actual. */
        public void endElement(String uri, String localName, String qName) throws SAXException {
            // Indica que hemos salido del elemento
            if ("library".equals(qName)) {
                inPlaylist = false;
                System.out.println("==========");

            } else if ("book".equals(qName)) {
                inCancion = false;
            }
        }
    }
}
```

Procesamiento XML: Path

1. Crear un objeto Xpath y una expresión (consulta)

2. Realizar una compilación con .compile() y después evaluamos la expresión con .evaluate()

3. Realizaremos una iteración (recorrer los nodos NodeList)

4. Verificar tipo de nodo y extraer información

5. Extraer información del nodo

Ejemplo de XPath

```
// Crear un objeto XPATH para consultar el documento XML
            XPath xPath = XPathFactory.newInstance().newXPath();
            // Definir la expresión XPath para obtener la lista de nodos
            String consulta = "/playlist/cancion";
            NodeList listaNodos = (NodeList) xPath.compile(consulta).evaluate(doc, XPathConstants.NODESET);

            // Iterar la lista
            for (int i = 0; i < listaNodos.getLength(); i++) {
                Node nNode = listaNodos.item(i);
                System.out.println("\nElemento Actual: " + nNode.getNodeName());

                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    System.out.println("Título: " + eElement.getElementsByTagName("titulo").item(0).getTextContent());

                    // Ahora accedemos al nombre del artista correctamente
                    Element artistaElement = (Element) eElement.getElementsByTagName("artista").item(0);
                    String nombreArtista = artistaElement.getElementsByTagName("nombre").item(0).getTextContent();
                    System.out.println("Artista: " + nombreArtista);

                    // Si necesitas la nacionalidad
                    String nacionalidadArtista = artistaElement.getElementsByTagName("nacionalidad").item(0)
                            .getTextContent();
                    System.out.println("Nacionalidad: " + nacionalidadArtista);

                    System.out.println("==========="); // Separador
                }
            }
```

Excepciones

El código propenso a excepciones se coloca dentro del bloque **try**. Cuando salta una excepción es capturada por el bloque **catch**, el bloque **finally** lo encontraremos siempre al final, se ejecutará siempre independientemente de saltar una excepción o no. Suele usarse para cerrar como close() o limpiar recursos de memoria.

Dos formas principales de manejar excepciones en Java

- Añadir la excepción en la definición del método

Lo que hacemos es lanzar una excepción a un nivel superior. Indicamos que este método puede lanzar una excepción. Ejemplo:

```
public static void main(String[] args) throws ParserConfigurationException {
```

- Rodear con sentencia try/catch

Rodearíamos el código con la estructura try-catch

ejemplo:

```
try {
dBuilder = dbFactory.newDocumentBuilder();
} catch (ParserConfigurationException e) {
e.printStackTrace();
}
```

Tipos de excepciones

- Excepciones con chequeo (chequed exceptions)

Las excepciones que son notificadas por el compilador, no pueden ser ignoradas. Obligan al programador a manejarlas si no no se compila. (no deja ejecutar) ejemplo: SQLException, IOException

- Excepciones sin chequeo (unchequed exceptions)

Estas excepciones no son verificadas por el compilador. Ocurren cuando se ejecuta el código y como resultado dan errores de programación. No son notificadas por el compilador. No es necesario manejarlas ejemplo: ArrayIndexOutOfBoundsExceptio, NullPointerException

- Errores

  no son excepciones, escapan del control del usuario y del programador. Errores ignorados en tiempo de compilación. Ejemplo Stack overflow(Error recursivo. (pararlo con ctrl+c). Un error que no para de repetirse)

Excepciones personalizadas Crear una excepción con throws

```java
public class ExcepcionPersonalizada {
    public static void main(String[] args) {

        try {
            // Llama al método que puede lanzar una excepción personalizada
            metodoQueLanzaExcepcion();
        } catch (MiExcepcion e) {
            // Captura y maneja la excepción personalizada
            System.out.println("Excepción capturada: " + e.getMessage());
        }

    }

    // Método que lanza la excepción
    public static void metodoQueLanzaExcepcion() throws MiExcepcion {
        throw new MiExcepcion("Esto es una excepción personalizada lanzada en la definición del método");
    }

    // Clase estática que define una excepción personalizada llamada MiExcepcion
    public static class MiExcepcion extends Exception {
        public MiExcepcion(String mensaje) {
            // Constructor que recibe un mensaje para la excepción
            super(mensaje);
        }
    }
}
```

Excepciones asociadas a clases por ejemplo XpathExpressionException asociada a XPath

Métodos para manejar excepciones e información detallada de los errores

- GetMessage(); mensaje detallado sobre la excepción
- getCause(): devuelve la causa de la excepción
- toStriing(); devuelve el nombre de la clase de la excepción
- printStackTrace(); imprime el resultado del método toString muestra el camino que siguió el código hasta la excepción
- getStackTrace(); Devuelve un array que representa cada uno de los elementos de la pila en el momento de la excepción
- fillInStackTrace(); se utiliza para capturar el estado de la pila en el momento en que se invoca, generalmente dentro del constructor de una excepción personalizada

Pruebas y documentación Junit

Un test: es una pieza de software que valida si el resultado de otro código es el resultado esperado. Junit es un framework que utiliza anotaciones para identificar test.

Una prueba unitaria: Es un método dentro de una clase llamada Test class, @Test

Para usar JUnit se puede añadir la anotación @Test, asi el ide importa las librerías necesarias.

Anotaciones:

@Before: Define un método que se ejecuta antes de cada test, para instanciar variables necesarias

@After: Define un método que se ejecuta después de cada test