Usando XML en Java

Usar XML en Java se puede hacer de varias maneras dependiendo de lo que quieras lograr: analizar, crear o manipular datos XML. Aquí tienes una guía concisa para empezar con algunos enfoques comunes:

1. Analizar XML con DOM (Document Object Model)

Si quieres leer y manipular XML como una estructura de árbol en memoria, la API DOM es una buena opción.

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class XMLParser {
    public static void main(String[] args) {
        try {
            // Cargar y analizar el archivo XML
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse("example.xml");

            // Normalizar la estructura XML
            doc.getDocumentElement().normalize();

            // Acceder a elementos (por ejemplo, todas las etiquetas "item")
            NodeList list = doc.getElementsByTagName("item");
            for (int i = 0; i < list.getLength(); i++) {
                Element element = (Element) list.item(i);
                System.out.println("Elemento: " + element.getNodeName());
                System.out.println("Valor: " + element.getTextContent());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ejemplo XML (example.xml):

```
<root>
    <item>Item 1</item>
    <item>Item 2</item>
</root>
```

- **Pros**: Fácil de navegar y modificar la estructura XML.
- **Cons**: Consumidor de memoria para archivos grandes.

---

2. Analizar XML con SAX (Simple API for XML)

Para archivos XML grandes donde la eficiencia de la memoria importa, SAX es impulsado por eventos y no carga todo el archivo en memoria.

```
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

public class SAXExample {
    public static void main(String[] args) {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();

            DefaultHandler handler = new DefaultHandler() {
                boolean inItem = false;

                @Override
                public void startElement(String uri, String localName, String qName, Attributes attributes) {
                    if (qName.equalsIgnoreCase("item")) {
                        inItem = true;
                    }
                }

                @Override
                public void characters(char[] ch, int start, int length) {
                    if (inItem) {
                        System.out.println("Item: " + new String(ch, start, length));
                        inItem = false;
```

```
                    }
                }
            };

            saxParser.parse("example.xml", handler);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Pros**: Ligero y rápido para archivos grandes.
- **Cons**: No hay acceso aleatorio; lo procesas secuencialmente.

---

3. Crear XML con DOM

Para generar XML desde cero:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class XMLCreator {
    public static void main(String[] args) {
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();

            // Elemento raíz
            Element root = doc.createElement("root");
            doc.appendChild(root);

            // Elemento hijo
```

```
            Element item = doc.createElement("item");
            item.appendChild(doc.createTextNode("Item 1"));
            root.appendChild(item);

            // Guardar en archivo
            TransformerFactory transformerFactory = TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult("output.xml");
            transformer.transform(source, result);

            System.out.println("Archivo XML creado!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Esto crea un archivo `output.xml` con la estructura mostrada anteriormente.

---

4. Usar JAXB (Java Architecture for XML Binding)

Si estás trabajando con XML que se mapea a objetos Java (por ejemplo, serialización/deserialización), JAXB es una opción de nivel superior. Necesitarás definir una clase con anotaciones.

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
class Item {
    private String name;

    @XmlElement
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}

import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
```

```java
import javax.xml.bind.Unmarshaller;
import java.io.File;

public class JAXBExample {
    public static void main(String[] args) {
        try {
            // Marshalling (Java a XML)
            JAXBContext context = JAXBContext.newInstance(Item.class);
            Marshaller marshaller = context.createMarshaller();
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

            Item item = new Item();
            item.setName("Item 1");
            marshaller.marshal(item, new File("item.xml"));

            // Unmarshalling (XML a Java)
            Unmarshaller unmarshaller = context.createUnmarshaller();
            Item loadedItem = (Item) unmarshaller.unmarshal(new File("item.xml"));
            System.out.println("Item cargado: " + loadedItem.getName());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Pros**: Simplifica el mapeo XML-a-objeto.
- **Cons**: Requiere clases anotadas y no es ideal para XML complejo e irregular.

Empezar

- **Dependencias**: Estos ejemplos usan las bibliotecas integradas de Java (como `javax.xml.parsers`), por lo que no se necesitan dependencias externas para DOM o SAX. Para JAXB, asegúrate de estar usando una versión de Java que lo incluya (pre-Java 9) o agrégalo a través de Maven/Gradle si usas una versión más nueva.
- **Archivo**: Reemplaza `"example.xml"` o `"output.xml"` con las rutas de tus archivos reales.

El enfoque que te conviene depende de tus necesidades: DOM para flexibilidad, SAX para eficiencia, o JAXB para mapeo de objetos. ¿Quieres que profundice en alguno de estos?