Utilisation de XML en Java

Utiliser XML en Java peut se faire de plusieurs façons en fonction de ce que vous souhaitez accomplir—analyser, créer ou manipuler des données XML. Voici un guide concis pour vous aider à démarrer avec quelques approches courantes :

1. Analyser XML avec DOM (Document Object Model)

Si vous souhaitez lire et manipuler XML sous forme d'une structure arborescente en mémoire, l'API DOM est un bon choix.

```java
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 {
            // Charger et analyser le fichier XML
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse("example.xml");

            // Normaliser la structure XML
            doc.getDocumentElement().normalize();

            // Accéder aux éléments (par exemple, toutes les balises "item")
            NodeList list = doc.getElementsByTagName("item");
            for (int i = 0; i < list.getLength(); i++) {
                Element element = (Element) list.item(i);
                System.out.println("Element: " + element.getNodeName());
                System.out.println("Value: " + element.getTextContent());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Exemple XML (example.xml):

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

- **Avantages** : Facile à naviguer et à modifier la structure XML.
- **Inconvénients** : Consommateur de mémoire pour les grands fichiers.

---

2. Analyser XML avec SAX (Simple API for XML)

Pour les grands fichiers XML où l'efficacité de la mémoire est importante, SAX est basé sur les événements et ne charge pas le fichier entier en mémoire.

```
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();
    }
  }
}
```

- **Avantages** : Léger et rapide pour les grands fichiers.
- **Inconvénients** : Pas d'accès aléatoire ; vous le traitez séquentiellement.

---

3. Créer XML avec DOM

Pour générer XML à partir de zéro :

```
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();

            // Élément racine
            Element root = doc.createElement("root");
            doc.appendChild(root);
```

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

            // Sauvegarder dans un fichier
            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("Fichier XML créé !");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Cela crée un fichier `output.xml` avec la structure montrée précédemment.

---

4. Utiliser JAXB (Java Architecture for XML Binding)

Si vous travaillez avec XML qui se mappe à des objets Java (par exemple, sérialisation/désérialisation), JAXB est une option de niveau supérieur. Vous devrez définir une classe avec des annotations.

```
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; }
}
```

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

public class JAXBExample {
    public static void main(String[] args) {
        try {
            // Marshalling (Java to 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 to Java)
            Unmarshaller unmarshaller = context.createUnmarshaller();
            Item loadedItem = (Item) unmarshaller.unmarshal(new File("item.xml"));
            System.out.println("Loaded item: " + loadedItem.getName());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Avantages** : Simplifie le mappage XML-objet.
- **Inconvénients** : Nécessite des classes annotées et n'est pas idéal pour des XML complexes et irréguliers.

Démarrer

- **Dépendances** : Ces exemples utilisent les bibliothèques intégrées de Java (comme `javax.xml.parsers`), donc aucune dépendance externe n'est nécessaire pour DOM ou SAX. Pour JAXB, assurez-vous d'utiliser une version de Java qui l'inclut (pré-Java 9) ou ajoutez-le via Maven/Gradle si vous utilisez une version plus récente.
- **Fichier** : Remplacez `"example.xml"` ou `"output.xml"` par vos chemins de fichiers réels.

L’approche qui vous convient dépend de vos besoins—DOM pour la flexibilité, SAX pour l’efficacité ou JAXB pour le mappage d’objets. Voulez-vous que je plonge plus profondément dans l’une de ces approches ?