XML in Java verwenden

Verwenden von XML in Java kann auf verschiedene Weisen erfolgen, je nachdem, was Sie erreichen möchten –Parsen, Erstellen oder Manipulieren von XML-Daten. Hier ist eine kurze Anleitung, um Sie mit einigen gängigen Ansätzen zu beginnen:

1. XML mit DOM (Document Object Model) parsen

Wenn Sie XML als Baumstruktur im Speicher lesen und manipulieren möchten, ist die DOM-API eine gute Wahl.

```
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 {
            // XML-Datei laden und parsen
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse("example.xml");

            // XML-Struktur normalisieren
            doc.getDocumentElement().normalize();

            // Elemente zugreifen (z.B. alle "item"-Tags)
            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("Wert: " + element.getTextContent());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Beispiel XML (example.xml):

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

- **Vorteile**: Einfach zu navigieren und zu ändern.
- **Nachteile**: Speicherintensiv für große Dateien.

---

2. XML mit SAX (Simple API for XML) parsen

Für große XML-Dateien, bei denen die Speichereffizienz wichtig ist, ist SAX ereignisgesteuert und lädt die gesamte Datei nicht in den Speicher.

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

- **Vorteile**: Leichtgewichtig und schnell für große Dateien.
- **Nachteile**: Kein direkter Zugriff; Sie verarbeiten es sequenziell.

---

3. XML mit DOM erstellen

Um XML von Grund auf neu zu erstellen:

```
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();

            // Root-Element
            Element root = doc.createElement("root");
            doc.appendChild(root);
```

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

            // In Datei speichern
            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("XML-Datei erstellt!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Dies erstellt eine `output.xml`-Datei mit der zuvor gezeigten Struktur.

---

4. Verwenden von JAXB (Java Architecture for XML Binding)

Wenn Sie mit XML arbeiten, das Java-Objekten entspricht (z.B. Serialisierung/Deserialisierung), ist JAXB eine höherstufige Option. Sie müssen eine Klasse mit Annotationen definieren.

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

- **Vorteile**: Vereinfacht die XML-zu-Objekt-Zuordnung.
- **Nachteile**: Erfordert annotierte Klassen und ist nicht ideal für komplexe, unregelmäßige XML.

Loslegen

- **Abhängigkeiten**: Diese Beispiele verwenden Java-integrierte Bibliotheken (wie `javax.xml.parsers`), sodass keine externen Abhängigkeiten für DOM oder SAX erforderlich sind. Für JAXB stellen Sie sicher, dass Sie eine Java-Version verwenden, die es enthält (vor Java 9) oder fügen Sie es über Maven/Gradle hinzu, wenn Sie eine neuere Version verwenden.
- **Datei**: Ersetzen Sie `"example.xml"` oder `"output.xml"` durch Ihre tatsächlichen Dateipfade.

Welcher Ansatz für Sie am besten geeignet ist, hängt von Ihren Anforderungen ab –DOM für Flexibilität, SAX für Effizienz oder JAXB für Objektzuordnung. Möchten Sie, dass ich tiefer in eines dieser Themen eintauche?