使用 XML 於 Java

在 Java 中使用 XML 可以通過多種方式來實現，具體取決於你想要達成的目標——解析、創建或操作 XML 數據。以下是一些常見方法的簡要指南：

1. 使用 DOM (文檔對象模型) 解析 XML

如果你想要在記憶體中以樹狀結構讀取和操作 XML，DOM API 是一個不錯的選擇。

```
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 {
            // 載入並解析 XML 文件
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse("example.xml");

            // 規範化 XML 結構
            doc.getDocumentElement().normalize();

            // 訪問元素（例如，所有的 "item" 標籤）
            NodeList list = doc.getElementsByTagName("item");
            for (int i = 0; i < list.getLength(); i++) {
                Element element = (Element) list.item(i);
                System.out.println(" 元素: " + element.getNodeName());
                System.out.println(" 值: " + element.getTextContent());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

範例 XML (example.xml):

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

- **優點**: 易於導航和修改 XML 結構。
- **缺點**: 對於大文件來說記憶體消耗大。

---

2. 使用 SAX (簡單 API for XML) 解析 XML

對於大 XML 文件，記憶體效率很重要，SAX 是事件驅動的，不會將整個文件加載到記憶體中。

```
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(" 項目: " + new String(ch, start, length));
                        inItem = false;
                    }
```

```
            }
        };

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

- **優點**: 對於大文件來說輕量且快速。
- **缺點**: 無隨機訪問；你須順序處理。

---

3. 使用 DOM 創建 XML

從頭開始生成 XML：

```
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();

            // 根元素
            Element root = doc.createElement("root");
            doc.appendChild(root);

            // 子元素
            Element item = doc.createElement("item");
```

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

            // 保存到文件
            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 文件已創建!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

這將創建一個具有前面所示結構的 `output.xml` 文件。

---

4. 使用 JAXB (Java Architecture for XML Binding)

如果你正在處理與 Java 對象映射的 XML（例如序列化/反序列化），JAXB 是一個更高層次的選擇。你需要定義一個帶有註解的類。

```
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;
import javax.xml.bind.Unmarshaller;
```

```java
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(" 已加載項目: " + loadedItem.getName());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **優點**: 簡化 XML 到對象的映射。
- **缺點**: 需要帶有註解的類，對於複雜、不規則的 XML 並不理想。

開始使用

- **依賴**: 這些範例使用 Java 的內建庫（如 `javax.xml.parsers`），因此 DOM 或 SAX 不需要外部依賴。對於 JAXB，請確保你使用的是包含它的 Java 版本（Java 9 之前）或通過 Maven/Gradle 添加它，如果使用的是較新版本。
- **文件**: 將 `"example.xml"` 或 `"output.xml"` 替換為你的實際文件路徑。

哪種方法適合你取決於你的需求——DOM 靈活，SAX 高效，JAXB 對象映射。想要我深入探討其中任何一種嗎？