使用 SAX（簡單 XML 應用程式介面）

使用 `org.xml.sax` 包在 Java 中解析 XML 文檔，可以按照以下簡單的過程使用 SAX（Simple API for XML）解析方法。SAX 是一種事件驅動的解析器，它按順序讀取 XML 文檔，並在遇到文檔的各個部分（如元素或文本內容）時觸發回調方法。這使得它在處理大型 XML 文件時非常高效，因為它不像 DOM 那樣將整個文檔加載到記憶體中。以下是使用 `org.xml.sax` 的逐步指南。

第一步：設置 SAX 解析器

要開始使用 SAX 解析 XML，需要使用 `SAXParserFactory` 類創建一個 `SAXParser` 實例。這個工廠允許在創建解析器之前配置解析器的行為。

以下是如何創建一個 `SAXParser`：

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

public class SAXExample {
    public static void main(String[] args) {
        try {
            // 創建一個 SAXParserFactory 實例
            SAXParserFactory factory = SAXParserFactory.newInstance();

            // 可選：配置工廠（例如，啟用命名空間感知）
            factory.setNamespaceAware(true); // 如果您的 XML 使用命名空間，請設置為 true

            // 創建一個 SAXParser
            SAXParser parser = factory.newSAXParser();

            // 下一步將在此處進行
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- `SAXParserFactory.newInstance()`：創建一個新的工廠實例。
- `factory.setNamespaceAware(true)`：可選。如果您的 XML 使用命名空間，啟用命名空間處理。默認情況下為 `false`。

- `factory.newSAXParser()`：創建一個 `SAXParser`。如果無法配置或初始化解析器，則可能會拋出 `ParserConfigurationException` 或 `SAXException`。

第二步：創建 XML 事件處理器

SAX 解析依賴於處理器來處理它遇到的 XML 事件，例如元素的開始、結束或文本內容。這個主要接口是 `ContentHandler`，但為了簡化，您可以擴展 `DefaultHandler` 類，它為 `ContentHandler` 和其他處理器接口（例如 `ErrorHandler`、`DTDHandler` 等）提供了空的實現。您只需覆蓋您關心的方法。

以下是自定義處理器的示例：

```
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;

public class MyHandler extends DefaultHandler {
    // 變量以累積文本內容
    private StringBuilder content = new StringBuilder();

    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes) {
        // 為新元素清除內容緩衝區
        content.setLength(0);
        System.out.println(" 開始元素: " + qName);

        // 如果元素有屬性，則處理它們
        for (int i = 0; i < attributes.getLength(); i++) {
            System.out.println(" 屬性: " + attributes.getQName(i) + " = " + attributes.getValue(i));
        }
    }

    @Override
    public void characters(char[] ch, int start, int length) {
        // 累積文本內容（可能會在單個元素的內容中多次調用）
        content.append(ch, start, length);
    }

    @Override
    public void endElement(String uri, String localName, String qName) {
        // 處理累積的文本內容
```

```
        String text = content.toString().trim();
        if (!text.isEmpty()) {
            System.out.println(" 內容: " + text);
        }
        System.out.println(" 結束元素: " + qName);
        content.setLength(0); // 重置以供下一個元素使用
    }

    @Override
    public void startDocument() {
        System.out.println(" 文檔解析開始");
    }

    @Override
    public void endDocument() {
        System.out.println(" 文檔解析完成");
    }
}
```

- `startElement`：當解析器遇到元素的開始時調用。參數包括：
  - `uri`：命名空間 URI（如果啟用命名空間感知）。
  - `localName`：元素的本地名稱（如果啟用命名空間感知，則不包含前綴）。
  - `qName`：限定名稱（如果有前綴，則包含前綴）。
  - `attributes`：元素的屬性列表。
- `characters`：在元素內調用文本內容。請注意，這可能會在單個元素的內容中多次調用，因此使用 `StringBuilder` 來累積它。
- `endElement`：當元素結束時調用。
- `startDocument` **和** `endDocument`：在 XML 文檔的開始和結束時調用。

第三步：解析 XML

準備好解析器和處理器後，可以通過在 `SAXParser` 上調用 `parse` 方法並傳遞您的處理器來解析 XML 源（例如文件、`InputStream` 或字符串）。

以下是如何解析 XML 文件：

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import java.io.File;
```

```
public class SAXExample {
    public static void main(String[] args) {
        try {
            // 創建解析器
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();

            // 創建處理器
            MyHandler handler = new MyHandler();

            // 解析 XML 文件
            File xmlFile = new File("example.xml");
            parser.parse(xmlFile, handler);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- `parser.parse`：接受 XML 源（例如 `File`）和 `DefaultHandler`。它可能會拋出 `IOException`（用於輸入錯誤）或 `SAXException`（用於解析錯誤）。

對於 XML 字符串而不是文件，請使用 `InputSource`：

```
import org.xml.sax.InputSource;
import java.io.StringReader;

// 在 main 方法內
String xml = "<book><title>XML 解析 </title></book>";
InputSource inputSource = new InputSource(new StringReader(xml));
parser.parse(inputSource, handler);
```

第四步：處理異常

SAX 解析涉及幾個異常，您應該處理：

- `ParserConfigurationException`：由 `newSAXParser()` 拋出，如果無法配置解析器。
- `SAXException`：由 `newSAXParser()` 或 `parse()` 拋出，用於一般解析錯誤。

- `IOException`：由 `parse()` 拋出，如果讀取輸入源時出現問題。

將代碼包裹在 `try-catch` 塊中：

```
try {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser parser = factory.newSAXParser();
    MyHandler handler = new MyHandler();
    parser.parse(new File("example.xml"), handler);
} catch (ParserConfigurationException e) {
    System.err.println(" 解析器配置錯誤: " + e.getMessage());
} catch (SAXException e) {
    System.err.println("SAX 錯誤: " + e.getMessage());
} catch (IOException e) {
    System.err.println("IO 錯誤: " + e.getMessage());
}
```

第五步：可選功能

錯誤處理 `DefaultHandler` 也實現了 `ErrorHandler`。覆蓋其方法以處理解析錯誤：

```
public class MyHandler extends DefaultHandler {
    @Override
    public void error(SAXParseException e) {
        System.err.println(" 解析錯誤: " + e.getMessage());
    }

    @Override
    public void fatalError(SAXParseException e) throws SAXException {
        System.err.println(" 致命錯誤: " + e.getMessage());
        throw e; // 停止解析
    }

    @Override
    public void warning(SAXParseException e) {
        System.out.println(" 警告: " + e.getMessage());
    }
    // 其他方法如上
}
```

命名空間處理　如果您的 XML 使用命名空間，請設置 `factory.setNamespaceAware(true)`，並在處理器方法中使用 `uri` 和 `localName` 而不是 `qName` 來正確識別元素。

驗證　要將 XML 驗證為 DTD 或架構，請設置 `factory.setValidating(true)`。您可能還需要一個 `ErrorHandler` 來管理驗證錯誤。

示例 XML 和輸出

對於 XML 文件 `example.xml`：

```
<book id="123">
    <title>XML 解析 </title>
    <author>John Doe</author>
</book>
```

運行上述代碼可能會輸出：

```
文檔解析開始
開始元素: book
屬性: id = 123
開始元素: title
內容: XML 解析
結束元素: title
開始元素: author
內容: John Doe
結束元素: author
結束元素: book
文檔解析完成
```

摘要

要使用 `org.xml.sax`：1. 使用 `SAXParserFactory` 創建一個 `SAXParser`。2. 通過擴展 `DefaultHandler` 並覆蓋方法（如 `startElement`、`endElement` 和 `characters`）來實現處理器。3. 通過調用 `parser.parse()` 並傳遞您的處理器和 XML 源來解析 XML。4. 處理異常（`SAXException`、`IOException`、`ParserConfigurationException`）。5. 可選配置命名空間感知、驗證或錯誤處理。

這種方法適合高效解析大型 XML 文檔，當您只需要提取特定數據而不構建內存樹時。