Java对XML解析有四种方法：

## 1. DOM生成和解析XML文档

解析器读入整个文档，在内存中构建一个树结构，通过操作这个树结构来获取相应的内容。优点：整个文档树在内存中，便于操作，支持增删改查等功能；缺点：在内存中存储整个文档，浪费内存。建议将XML数据转换成Java对象存取。

// 获取Document对象: javax.xml.parsers.DocumentBuilderFactory

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

// 使用当前配置的参数创建一个新的 DocumentBuilder 实例

DocumentBuilder builder = null;

try {

   builder = factory.newDocumentBuilder();

} catch (ParserConfigurationException e) {

   e.printStackTrace();

}

// 获取 DOM Document 对象的一个新实例来生成一个 DOM 树

this.document = builder.newDocument();

## 2. SAX解析XML文档

SAX是事件驱动的XML解析器。当解析器发现元素开始、元素结束、文本、文档的开始或结束时，向编写的事件响应代码中发送事件，在响应代码中获取相应的数据。优点：采用流的方式读取数据，占用资源少；缺点：非持久型数据读取方式，读取一遍，没有处理的数据不会保存。

package cn.main.sax;  
  
import java.io.File;  
  
import javax.xml.parsers.SAXParser;  
import javax.xml.parsers.SAXParserFactory;  
  
import org.xml.sax.Attributes;  
import org.xml.sax.SAXException;  
import org.xml.sax.helpers.DefaultHandler;  
  
*/\*\*  
 \* SAX解析XML文档  
 \*  
 \* startDocument()，endDocument(),startElement(),endElement(),characters()  
 \*  
 \** ***@author*** *wanglp 2012-2-21  
 \*/*public class SAXParseDemo extends DefaultHandler {  
  
 private String tagValue; // 标签值  
  
 // 开始解析XML文件  
 public void startDocument() throws SAXException {  
 System.*out*.println("开始解析");  
 }  
  
 // 结束解析XML文件  
 public void endDocument() throws SAXException {  
 System.*out*.println("结束解析");  
 }  
  
 // 解析元素  
 */\*\*  
 \* 开始解析一个元素  
 \** ***@param*** *qName 标签名  
 \** ***@param*** *attributes 属性  
 \*/* @Override  
 public void startElement(String uri, String localName, String qName, Attributes attributes)  
 throws SAXException {  
 System.*out*.println(qName + "开始");  
 // 属性  
 if (attributes != null && attributes.getLength() != 0) {  
 System.*out*.println("属性：");  
 for (int i = 0; i < attributes.getLength(); i++) {  
 System.*out*.print(attributes.getQName(i) + "="); // 属性名  
 System.*out*.print(attributes.getValue(i) + " "); // 属性值  
 }  
 System.*out*.println();  
 }  
 }  
  
 */\*\*  
 \* 结束一个元素的解析 遇到结束标签时调用此方法 通常在此方法对标签取值并处理  
 \*/* @Override  
 public void endElement(String uri, String localName, String qName) throws SAXException {  
 System.*out*.println(qName + "标签值：" + tagValue);  
 System.*out*.println(qName + "结束");  
 }  
  
 // 所有xml文件中的字符都会放到ch[]中  
 public void characters(char ch[], int start, int length) throws SAXException {  
 tagValue = new String(ch, start, length).trim();  
 }  
  
 public static void main(String[] args) {  
 File file = new File("src/cn/main/example/demo.xml");  
 SAXParserFactory saxParFac = SAXParserFactory.*newInstance*();  
 try {  
 SAXParser saxParser = saxParFac.newSAXParser();  
 saxParser.parse(file, new SAXParseDemo());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
}

## 3. DOM4j

DOM4j具有强大的功能和优异的性能。可以使用XPath解析xml。

## 4. 将XML转换成Java对象

JDK中自带的JAXB可以将XML转换成Java对象。该方法采用Annotation的方式，将XML数据注入到对象中，该方法有编码少，效率高。适用于持久化存储。

JDK中JAXB相关的Class和Interface：

* JAXBContext类，是应用的入口，用于管理XML/Java绑定信息
* Marshaller接口，将Java对象序列化为XML数据。
* Unmarshaller接口，将XML数据反序列化为Java对象。

相应的ANnotation：

* @XmlType，将Java类或枚举类型映射到XML模式类型
* @XmlAccessorType(XmlAccessType.FIELD) ，控制字段或属性的序列化。FIELD表示JAXB将自动绑定Java类中的每个非静态的（static）、非瞬态的（由@XmlTransient标注）字段到XML。其他值还有XmlAccessType.PROPERTY和XmlAccessType.NONE。
* @XmlAccessorOrder，控制JAXB 绑定类中属性和字段的排序。
* @XmlJavaTypeAdapter，使用定制的适配器（即扩展抽象类XmlAdapter并覆盖marshal()和unmarshal()方法），以序列化Java类为XML。
* @XmlElementWrapper ，对于数组或集合（即包含多个元素的成员变量），生成一个包装该数组或集合的XML元素（称为包装器）。
* @XmlRootElement，将Java类或枚举类型映射到XML元素。
* @XmlElement，将Java类的一个属性映射到与属性同名的一个XML元素。
* @XmlAttribute，将Java类的一个属性映射到与属性同名的一个XML属性。