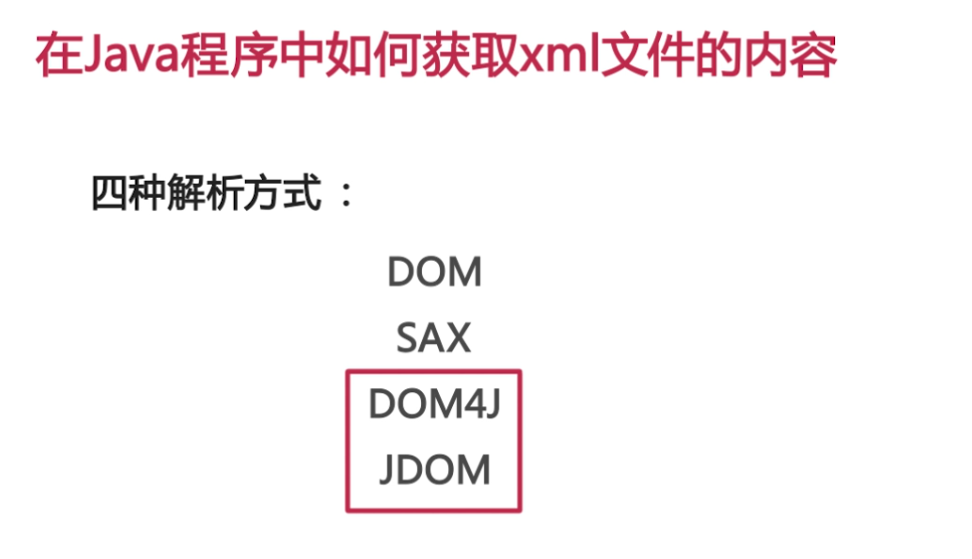
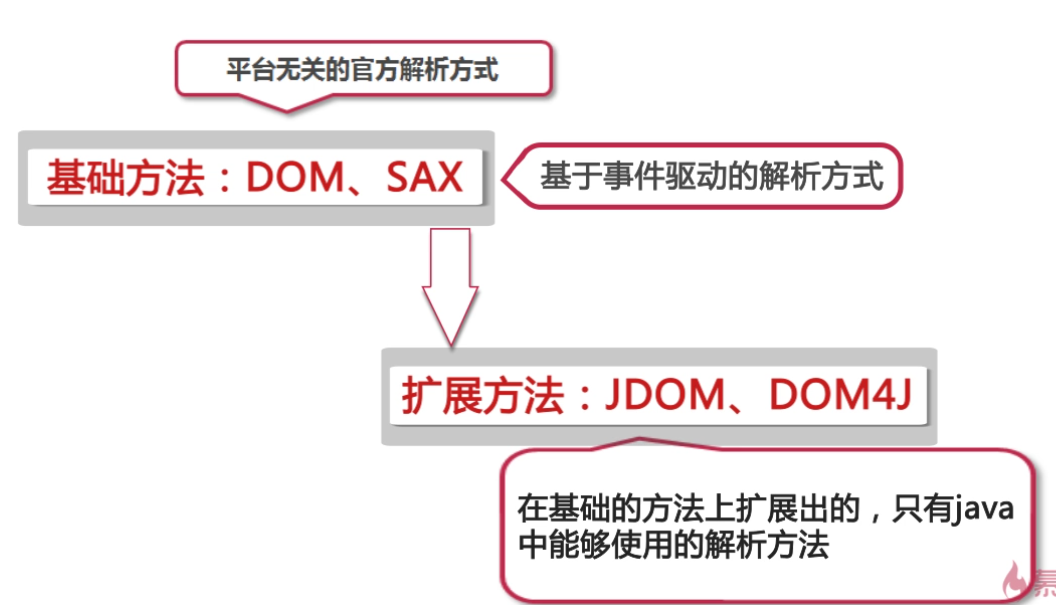
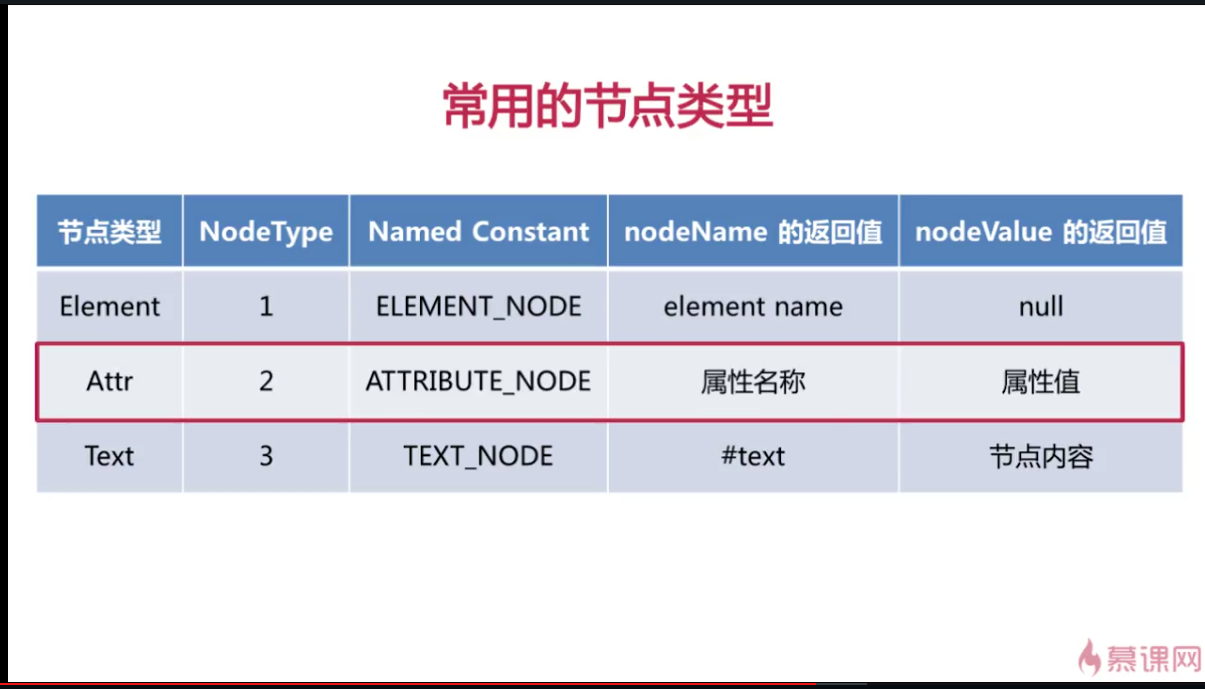
# XML解析



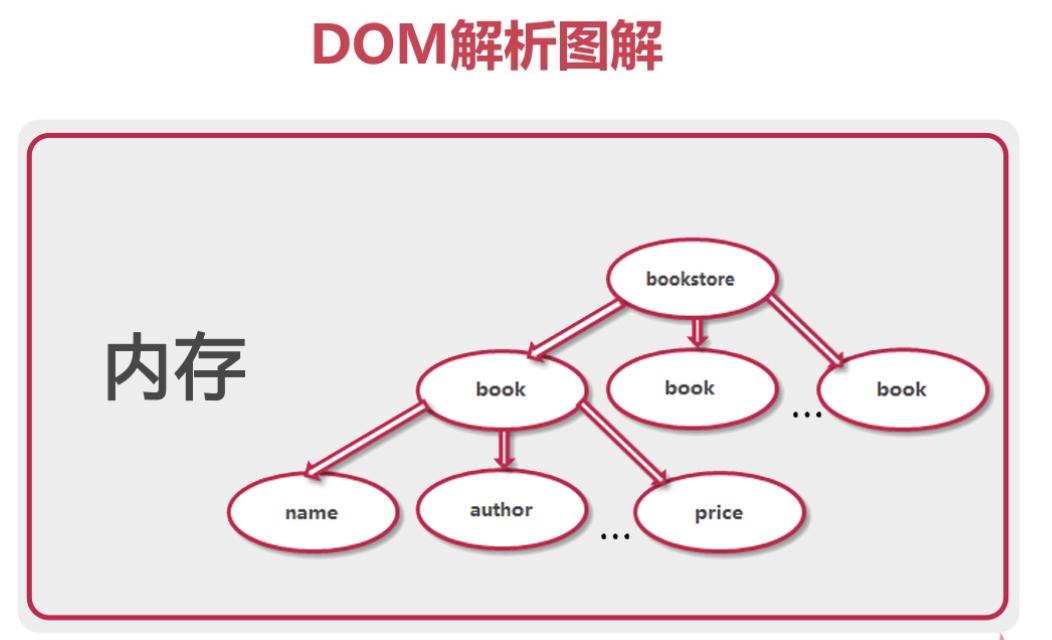
Dom、SAX解析需要不需要导入jar包，DOM4J、JDOM需要导入jar包

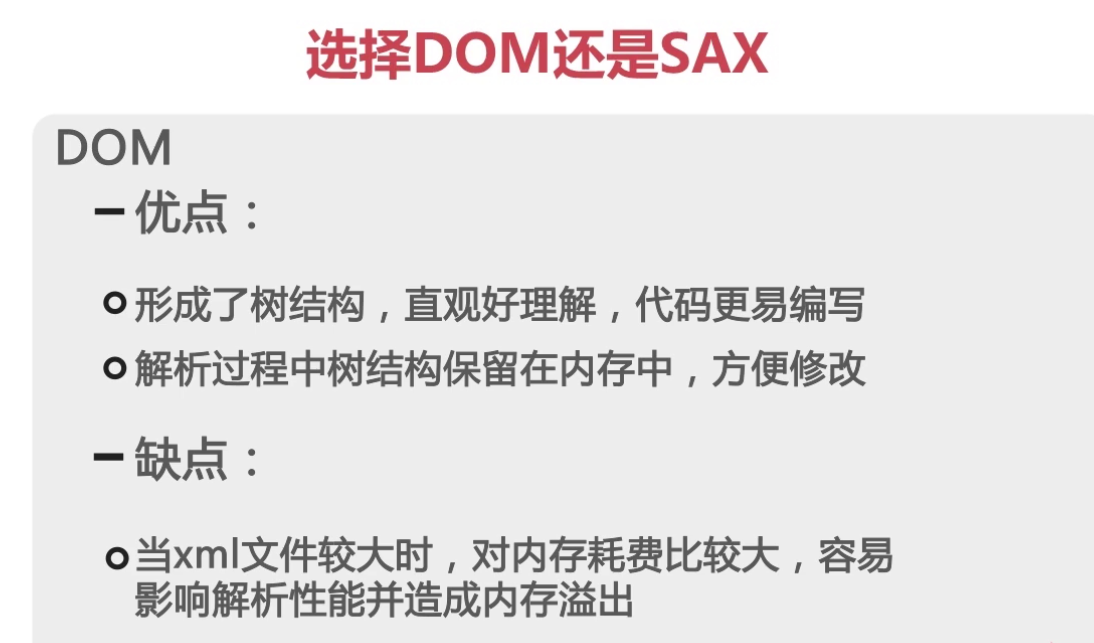


## DOM解析



Dom解析需要将xml全部加载到内存中





Book XML示例:



### 获取Dom对象

//创建DocumentBuilderFactery对象

DocumentBuilderFactory dbf = DocumentBuilderFactory.*newInstance*();

//创建DocumentBuilder对象

DocumentBuilder db = dbf.newDocumentBuilder();

//创建DocumentBuilder对象的parse方法加载books.xml文件到当前目录下

Document document = db.parse("xml/books.xml");

示例：

**public** **class** DOMXMLAnalysis {

**public** **static** **void** main(String[] args) {

**try** {

//创建DocumentBuilderFactery对象

DocumentBuilderFactory dbf = DocumentBuilderFactory.*newInstance*();

//创建DocumentBuilder对象

DocumentBuilder db = dbf.newDocumentBuilder();

//创建DocumentBuilder对象的parse方法加载books.xml文件到当前目录下

Document document = db.parse("xml/books.xml");

//获取book集合

NodeList nodeList = document.getElementsByTagName("book");

System.***out***.println("共有" + nodeList.getLength() + "本书");

//遍历book集合

**for** (**int** i = 0; i < nodeList.getLength(); i++) {

System.***out***.println("==========开始遍历第" + (i+1) + "本书的内容==========");

/\*\*

\* 获取属性方法一

\*/

//通过NodeList的item(index)获取Node

Node book = nodeList.item(i);

//获取Node的属性值

NamedNodeMap attrs = book.getAttributes();

System.***out***.println("第" + (i + 1) + "本书共有" + attrs.getLength() + "个属性");

//遍历NamedNodeMap属性

**for** (**int** j = 0; j < attrs.getLength(); j++) {

//通过item(index)获取属性

Node attr = attrs.item(j);

System.***out***.print("属性名：" + attr.getNodeName());

System.***out***.println("--属性值：" + attr.getNodeValue());

}

/\*\*

\* 获取属性方法二

\*/

//通过强制类型转换获取Element对象（标签对象）

// Element bookElement = (Element) nodeList.item(i);

//获取属性值

// String elementValue = bookElement.getAttribute("id");

// System.out.println("id属性的属性值为" + elementValue);

/\*\*

\* 通过getChildNodes()获取子节点

\*/

NodeList childNode = book.getChildNodes();

System.***out***.println("第" + (i + 1) + "共有" + childNode.getLength() + "个子节点");

//遍历子节点内容

**for** (**int** j = 0; j < childNode.getLength(); j++) {

//获取子节点

Node node = childNode.item(j);

//判断子节点属性 ELEMENT\_NODE(ElEMENT) ATTRIBUTE\_NODE(Attr) TEXT\_NODE(Text)

**if**(node.getNodeType() == Node.***ELEMENT\_NODE***) {

//获取Element节点的名称

System.***out***.print("第" + (j + 1) + "个节点名：" + node.getNodeName());

//获取Element节点的节点值

// System.out.println("--节点值：" + node.getFirstChild().getNodeValue());

System.***out***.println("--节点值：" + node.getTextContent());

}

}

System.***out***.println("==========结束遍历第" + (i+1) + "本书的内容==========");

}

} **catch** (ParserConfigurationException e) {

e.printStackTrace();

} **catch** (SAXException e) {

e.printStackTrace();

} **catch** (IOException e) {

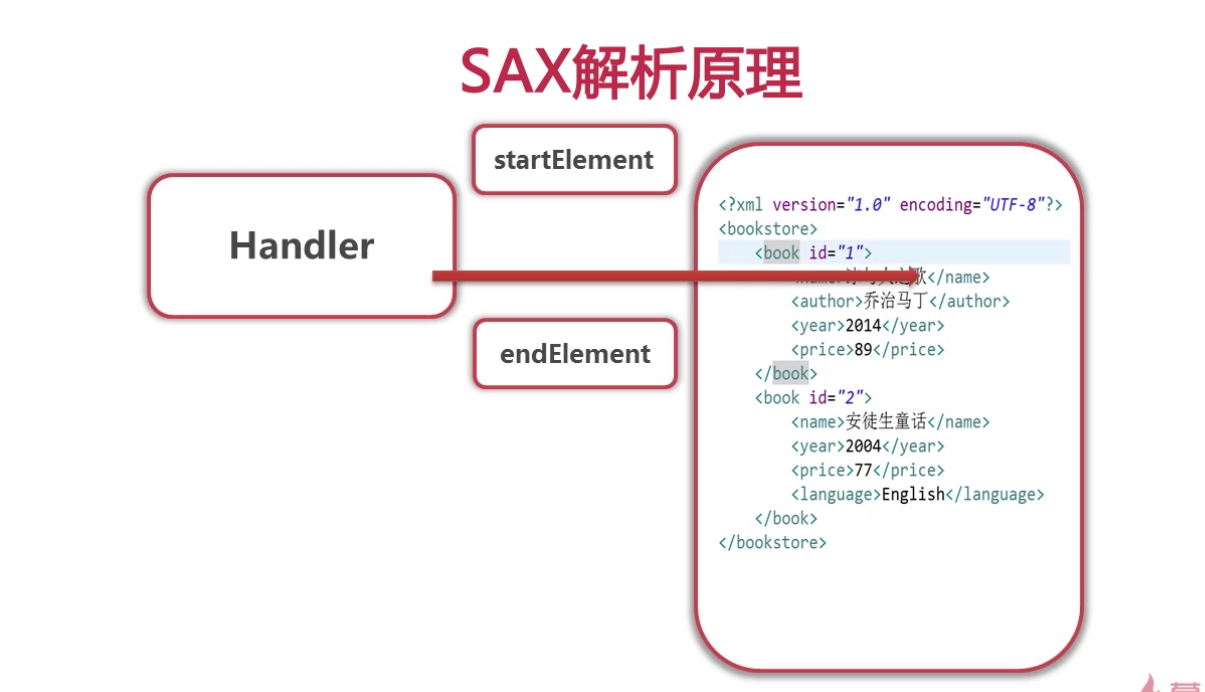
e.printStackTrace();

}

}

}

## SAXj解析





**public** **class** SAXTest {

**public** **static** **void** main(String[] args) {

//使用SAXParserFactory.newInstance()构建SAXParserFactory对象

SAXParserFactory sf = SAXParserFactory.*newInstance*();

//通过SAXParserFactory创建SAXParser对象

**try** {

SAXParser saxParser = sf.newSAXParser();

//创建SAXPaserHandler对象

SAXPaserHandler handler = **new** SAXPaserHandler();

saxParser.parse("xml/books.xml", handler);

} **catch** (ParserConfigurationException e) {

e.printStackTrace();

} **catch** (SAXException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

public class SAXPaserHandler extends DefaultHandler {

/\*\*

\* 用来标识开始标签

\*/

@Override

public void startElement(String uri, String localName, String qName,

Attributes attributes) throws SAXException {

//开始解析book元素的属性

if("book".equals(qName)) {

System.out.println("=====================开始解析某一本书====================");

//1.已知book元素下的属性名称，获取属性

// String value = attributes.getValue("id");

// System.out.println("book的属性值：" + value);

//2.不知道book元素下的属性名称及属性个数

int length = attributes.getLength();

for (int i = 0; i < length; i++) {

//获取属性名

System.out.print("book的第" + (i + 1) + "个属性名："

+ attributes.getQName(i));

//属性值

System.out.println(" --属性值：" + attributes.getValue(i));

}

} else if(!"bookstore".equals(qName)) {

System.out.print("节点名：" + qName);

}

}

/\*\*

\* 用来标识结束标签

\*/

@Override

public void endElement(String uri, String localName, String qName)

throws SAXException {

//结束简析book元素

if("book".equals(qName)) {

System.out.println("=====================结束解析某一本书====================");

}

}

/\*\*

\* 获取节点值

\*/

@Override

public void characters(char[] ch, int start, int length)

throws SAXException {

//节点值

String value = new String(ch, start, length);

if(!value.trim().equals("")) {

System.out.println(" --节点值：" + value);

}

}

/\*\*

\* 开始SAX解析

\*/

@Override

public void startDocument() throws SAXException {

System.out.println("开始SAX解析");

}

/\*\*

\* 结束SAX解析

\*/

@Override

public void endDocument() throws SAXException {

System.out.println("结束SAX解析");

}

}

### 使用java对象保存xml内容和结构

根据xml标签创建实体类

**public** **class** SAXTest {

**public** **static** **void** main(String[] args) {

//使用SAXParserFactory.newInstance()构建SAXParserFactory对象

SAXParserFactory sf = SAXParserFactory.*newInstance*();

//通过SAXParserFactory创建SAXParser对象

**try** {

SAXParser saxParser = sf.newSAXParser();

//创建SAXPaserHandler对象

SAXPaserHandler handler = **new** SAXPaserHandler();

saxParser.parse("xml/books.xml", handler);

List<Book> lsBook = handler.getLsBook();

**for** (Book book : lsBook) {

System.***out***.println(book.toString());

}

} **catch** (ParserConfigurationException e) {

e.printStackTrace();

} **catch** (SAXException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 重写SAX解析Handler

\* **@author** qinbo

\*

\*/

**public** **class** SAXPaserHandler **extends** DefaultHandler {

/\*\*

\* 节点内容

\*/

**private** String value = **null**;

/\*\*

\* 声明Book对象

\*/

**private** Book book = **null**;

/\*\*

\* 使用List存储Book对象

\*/

**private** List<Book> lsBook = **new** ArrayList<Book>();

**public** List<Book> getLsBook() {

**return** lsBook;

}

**public** **void** setLsBook(List<Book> lsBook) {

**this**.lsBook = lsBook;

}

/\*\*

\* 用来标识开始标签

\*/

@Override

**public** **void** startElement(String uri, String localName, String qName,

Attributes attributes) **throws** SAXException {

//开始解析book元素的属性

**if**("book".equals(qName)) {

//实例化Book对象

book = **new** Book();

System.***out***.println("=====================开始解析某一本书====================");

//1.已知book元素下的属性名称，获取属性

// String value = attributes.getValue("id");

// System.out.println("book的属性值：" + value);

//2.不知道book元素下的属性名称及属性个数

**int** length = attributes.getLength();

**for** (**int** i = 0; i < length; i++) {

//获取id属性

**if**("id".equals(attributes.getQName(i))) {

book.setId(attributes.getValue(i));

}

//获取属性名

System.***out***.print("book的第" + (i + 1) + "个属性名："

+ attributes.getQName(i));

//属性值

System.***out***.println(" --属性值：" + attributes.getValue(i));

}

} **else** **if**(!"bookstore".equals(qName)) {

System.***out***.print("节点名：" + qName);

}

}

/\*\*

\* 用来标识结束标签

\*/

@Override

**public** **void** endElement(String uri, String localName, String qName)

**throws** SAXException {

//结束简析book元素

**if**("book".equals(qName)) {

lsBook.add(book);

book = **null**;

System.***out***.println("=====================结束解析某一本书====================");

} **else** **if**("name".equals(qName)) {

book.setName(value);

} **else** **if**("author".equals(qName)) {

book.setAuthor(value);

} **else** **if**("year".equals(qName)) {

book.setYear(value);

} **else** **if**("price".equals(qName)) {

book.setPrice(value);

}

}

/\*\*

\* 获取节点值

\*/

@Override

**public** **void** characters(**char**[] ch, **int** start, **int** length)

**throws** SAXException {

//节点值

value = **new** String(ch, start, length);

**if**(!value.trim().equals("")) {

System.***out***.println(" --节点值：" + value);

}

}

/\*\*

\* 开始SAX解析

\*/

@Override

**public** **void** startDocument() **throws** SAXException {

System.***out***.println("开始SAX解析");

}

/\*\*

\* 结束SAX解析

\*/

@Override

**public** **void** endDocument() **throws** SAXException {

System.***out***.println("结束SAX解析");

}

}

/\*\*

\* 使用Book实体类保存xml内容和结构

\* **@author** qinbo

\*

\*/

**public** **class** Book {

/\*\*

\* id属性

\*/

**private** String id = **null**;

/\*\*

\* name标签

\*/

**private** String name = **null**;

/\*\*

\* author标签

\*/

**private** String author = **null**;

/\*\*

\* year标签

\*/

**private** String year = **null**;

/\*\*

\* price标签

\*/

**private** String price = **null**;

**public** Book() {}

**public** Book(String id, String name, String author, String year) {

**this**.id = id;

**this**.name = name;

**this**.author = author;

**this**.year = year;

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getAuthor() {

**return** author;

}

**public** **void** setAuthor(String author) {

**this**.author = author;

}

**public** String getYear() {

**return** year;

}

**public** **void** setYear(String year) {

**this**.year = year;

}

**public** String getPrice() {

**return** price;

}

**public** **void** setPrice(String price) {

**this**.price = price;

}

@Override

**public** String toString() {

**return** "Book [id=" + id + ", name=" + name + ", author=" + author

+ ", year=" + year + ", price=" + price + "]";

}

}

## JDOM解析

使用JDOM解析xml需要添加jdom的jar包

/\*\*

\* JDom解析xml

\* **@author** qinbo

\*

\*/

**public** **class** JDomTest {

**public** **static** **void** main(String[] args) {

//实例化SAXBuilder对象

SAXBuilder builder = **new** SAXBuilder();

**try** {

//使用InputStream加载xml

InputStream in = **new** FileInputStream("xml/books.xml");

//实例化Document对象

Document document = builder.build(in);

//获取根节点

Element books = document.getRootElement();

//获取子节点及属性

List<Element> lsElement = books.getChildren();

**for** (Element book : lsElement) {

System.***out***.println("===========开始解析第" + (lsElement.indexOf(book) + 1) + "书============");

//判断节点是否有属性

**if**(book.hasAttributes()) {

//获取节点的属性

List<Attribute> lsAttr = book.getAttributes();

**for** (Attribute attribute : lsAttr) {

System.***out***.print("属性名：" + attribute.getName());

System.***out***.println(" ---属性值：" + attribute.getValue());

}

}

//获取book的子节点

List<Element> lsBooks = book.getChildren();

**for** (Element element : lsBooks) {

//获取标签名称

String name = element.getName();

System.***out***.println("节点名：" + name + " ---节点值：" + element.getText());

}

System.***out***.println("===========结束解析第" + (lsElement.indexOf(book) + 1) + "书============");

}

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (JDOMException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

### JDOM处理乱码问题

方法一：

如果解析xml显示乱码，修改 xml中首行encoding编码方式

<?xml version="1.0" encoding="UTF-8"?>

方法二：

使用InputStreamReader流转换编码方式

InputStreamReader ins = new InputStreamReader(in, "UTF-8");

## DOM4J解析



使用DOM4J解析xml需要获取

/\*\*

\* Dom4j解析xml

\* **@author** qinbo

\*

\*/

**public** **class** Dom4jTest {

**public** **static** **void** main(String[] args) {

//实例化SAXReader对象

SAXReader saxReader = **new** SAXReader();

**try** {

//加载xml

Document document = saxReader.read(**new** File("xml/books.xml"));

//获取根节点

Element bookStore = document.getRootElement();

//遍历根节点

Iterator<Element> iterator = bookStore.elementIterator();

**while**(iterator.hasNext()) {

System.***out***.println("============开始解析某一本书========");

//获取子节点

Element book = iterator.next();

//判断子节点是否有属性

**if**(book.attributeCount() > 0) {

//获取属性

List<Attribute> lsAttributes = book.attributes();

**for** (Attribute attribute : lsAttributes) {

System.***out***.println("属性名：" + attribute.getName()

+ " ----属性值：" + attribute.getValue());

}

}

//获取book的子节点

Iterator<Element> iteratorBook = book.elementIterator();

**while**(iteratorBook.hasNext()) {

Element node = iteratorBook.next();

System.***out***.println("节点名：" + node.getName()

+ " ----节点值：" + node.getText());

}

System.***out***.println("============结束解析某一本书========");

}

} **catch** (DocumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}