16 XML与JSON

XML和JSON是两种经常在网络使用的数据表示格式，本章我们介绍如何使用Java读写XML和JSON。

XML简介

XML是可扩展标记语言（eXtensible Markup Language）的缩写，它是是一种数据表示格式，可以描述非常复杂的数据结构，常用于传输和存储数据。

例如，一个描述书籍的XML文档可能如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE note SYSTEM "book.dtd">
<book id="1">
    <name>Java核心技术</name>
    <author>Cay S. Horstmann</author>
    <isbn lang="CN">1234567</isbn>
    <tags>
        <tag>Java</tag>
        <tag>Network</tag>
    </tags>
    <pubDate/>
</book>
```

XML有几个特点：一是纯文本，默认使用UTF-8编码，二是可嵌套，适合表示结构化数据。如果把XML内容存为文件，那么它就是一个XML文件，例如`book.xml`。此外，XML内容经常通过网络作为消息传输。

XML的结构

XML有固定的结构，首行必定是`<?xml version="1.0" encoding="UTF-8" standalone="no"?>`，可以加上可选的编码。紧接着，如果以类似`<!DOCTYPE note SYSTEM "Note.dtd">`声明的是文档定义类型（DTD：Document Type Definition），DTD是可选的。接下来是XML的文档内容，一个XML文档有且仅有一个根元素，根元素可以包含任意个子元素，元素可以包含属性，例如，`1234567`包含一个属性`lang="CN"`，且元素必须正确嵌套。如果是空元素，可以用``表示。

由于使用了`<`、`>`以及引号等标识符，如果内容出现了特殊符号，需要使用`&???;`表示转义。例如，`Java`必须写成：

```
<name>Java&lt;tm&gt;</name>
```

常见的特殊字符如下：

字符	表示
<	<
>	>
&	&
"	"
'	'

格式正确的XML（Well Formed）是指XML的格式是正确的，可以被解析器正常读取。而合法的XML是指，不但XML格式正确，而且它的数据结构可以被DTD或者XSD验证。

DTD文档可以指定一系列规则，例如：

- 根元素必须是`book`
- `book`元素必须包含`name`，`author`等指定元素
- `isbn`元素必须包含属性`lang`
- ...

如何验证XML文件的正确性呢？最简单的方式是通过浏览器验证。可以直接把XML文件拖拽到浏览器窗口，如果格式错误，浏览器会报错。

和结构类似的HTML不同，浏览器对HTML有一定的“容错性”，缺少关闭标签也可以被解析，但XML要求严格的格式，任何没有正确嵌套的标签都会导致错误。

XML是一个技术体系，除了我们经常用到的XML文档本身外，XML还支持：

- DTD和XSD：验证XML结构和数据是否有效；
- Namespace：XML节点和属性的名字空间；
- XSLT：把XML转化为另一种文本；
- XPath：一种XML节点查询语言；
- ...

实际上，XML的这些相关技术实现起来非常复杂，在实际应用中很少用到，通常了解一下就可以了。

小结

- XML使用嵌套结构的数据表示方式，支持格式验证；
- XML常用于配置文件、网络消息传输等。

使用DOM

因为XML是一种树形结构的文档，它有两种标准的解析API：

- DOM：一次性读取XML，并在内存中表示为树形结构；
- SAX：以流的形式读取XML，使用事件回调。

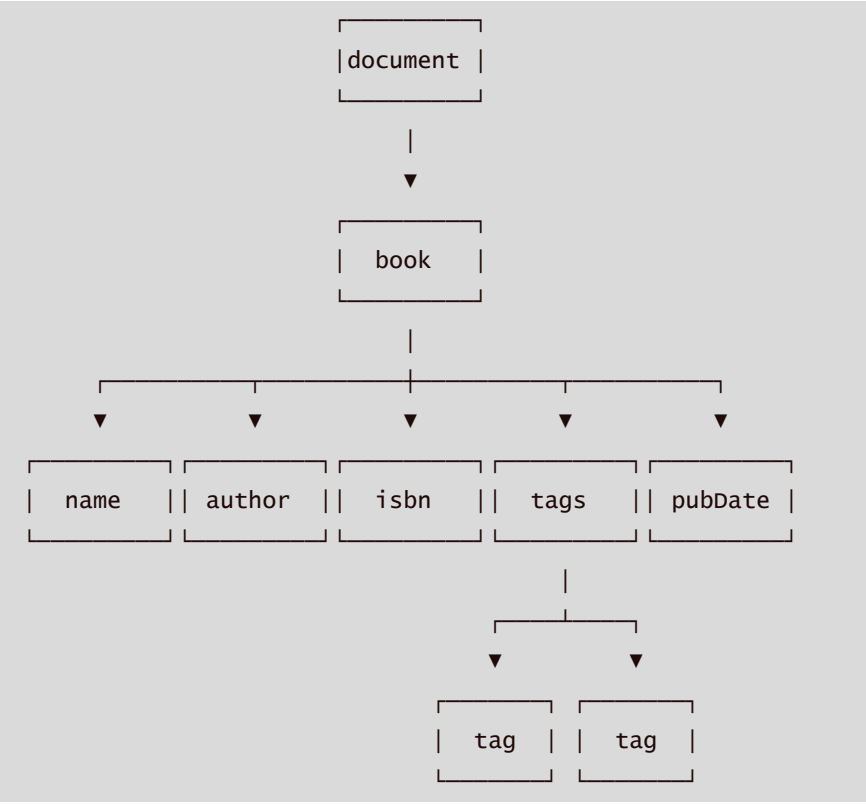
我们先来看如何使用DOM来读取XML。

DOM是Document Object Model的缩写，DOM模型就是把XML结构作为一个树形结构处理，从根节点开始，每个节点都可以包含任意个子节点。

我们以下面的XML为例：

```
<?xml version="1.0" encoding="UTF-8" ?>
<book id="1">
    <name>Java核心技术</name>
    <author>Cay S. Horstmann</author>
    <isbn lang="CN">1234567</isbn>
    <tags>
        <tag>Java</tag>
        <tag>Network</tag>
    </tags>
    <pubDate/>
</book>
```

如果解析为DOM结构，它大概长这样：



注意到最顶层的document代表XML文档，它是真正的“根”，而`虽然是根元素，但它是document`的一个子节点。

Java提供了DOM API来解析XML，它使用下面的对象来表示XML的内容：

- Document：代表整个XML文档；
- Element：代表一个XML元素；
- Attribute：代表一个元素的某个属性。

使用DOM API解析一个XML文档的代码如下：

```
InputStream input =
Main.class.getResourceAsStream("/book.xml");
DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(input);
```

`DocumentBuilder.parse()`用于解析一个XML，它可以接收InputStream，File或者URL，如果解析无误，我们将获得一个Document对象，这个对象代表了整个XML文档的树形结构，需要遍历以便读取指定元素的值：

```
void printNode(Node n, int indent) {
    for (int i = 0; i < indent; i++) {
        System.out.print(' ');
    }
    switch (n.getNodeType()) {
    case Node.DOCUMENT_NODE: // Document节点
        System.out.println("Document: " +
n.getNodeName());
        break;
    case Node.ELEMENT_NODE: // 元素节点
        System.out.println("Element: " + n.getNodeName());
        break;
    case Node.TEXT_NODE: // 文本
        System.out.println("Text: " + n.getNodeName() + "
= " + n.getNodeValue());
        break;
    case Node.ATTRIBUTE_NODE: // 属性
        System.out.println("Attr: " + n.getNodeName() + "
= " + n.getNodeValue());
        break;
    default: // 其他
        System.out.println("NodeType: " + n.getNodeType()
+ ", NodeName: " + n.getNodeName());
    }
    for (Node child = n.getFirstChild(); child != null;
child = child.getNextSibling()) {
        printNode(child, indent + 1);
    }
}
```

解析结构如下：

```
Document: #document
 Element: book
  Text: #text =

  Element: name
   Text: #text = Java核心技术
  Text: #text =

  Element: author
   Text: #text = Cay S. Horstmann
  Text: #text =
  ...
```

对于DOM API解析出来的结构，我们从根节点Document出发，可以遍历所有子节点，获取所有元素、属性、文本数据，还可以包括注释，这些节点被统称为Node，每个Node都有自己的Type，根据Type来区分一个Node到底是元素，还是属性，还是文本，等等。

使用DOM API时，如果要读取某个元素的文本，需要访问它的Text类型的子节点，所以使用起来还是比较繁琐的。

练习

下载练习：使用DOM解析XML（推荐使用IDE练习插件快速下载）

小结

- Java提供的DOM API可以将XML解析为DOM结构，以Document对象表示；
- DOM可在内存中完整表示XML数据结构；
- DOM解析速度慢，内存占用大。

使用SAX

使用DOM解析XML的优点是用起来省事，但它的主要缺点是内存占用太大。

另一种解析XML的方式是SAX。SAX是Simple API for XML的缩写，它是一种基于流的解析方式，边读取XML边解析，并以事件回调的方式让调用者获取数据。因为是一边读一边解析，所以无论XML有多大，占用的内存都很小。

SAX解析会触发一系列事件：

- startDocument：开始读取XML文档；
- startElement：读取到了一个元素，例如``；
- characters：读取到了字符；
- endElement：读取到了一个结束的元素，例如``；
- endDocument：读取XML文档结束。

如果我们用SAX API解析XML，Java代码如下：

```
InputStream input =
Main.class.getResourceAsStream("/book.xml");
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser saxParser = spf.newSAXParser();
saxParser.parse(input, new MyHandler());
```

关键代码`SAXParser.parse()`除了需要传入一个`InputStream`外，还需要传入一个回调对象，这个对象要继承自`DefaultHandler`：

```
class MyHandler extends DefaultHandler {
    public void startDocument() throws SAXException {
        print("start document");
    }

    public void endDocument() throws SAXException {
        print("end document");
    }

    public void startElement(String uri, String localName,
String qName, Attributes attributes) throws SAXException {
        print("start element:", localName, qName);
    }

    public void endElement(String uri, String localName,
String qName) throws SAXException {
        print("end element:", localName, qName);
    }

    public void characters(char[] ch, int start, int
length) throws SAXException {
        print("characters:", new String(ch, start,
length));
    }

    public void error(SAXParseException e) throws
SAXException {
        print("error:", e);
    }

    void print(Object... objs) {
        for (Object obj : objs) {
            System.out.print(obj);
            System.out.print(" ");
        }
        System.out.println();
    }
}
```

运行SAX解析代码，可以打印出下面的结果：

```
start document
start element:  book
characters:

start element:  name
characters: Java核心技术
end element:  name
characters:

start element:  author
...
```

如果要读取`节点的文本，我们就必须在解析过程中根据startElement()和endElement()定位当前正在读取的节点，可以使用栈结构保存，每遇到一个startElement()入栈，每遇到一个endElement()出栈，这样，读到characters()`时我们才知道当前读取的文本是哪个节点的。可见，使用SAX API仍然比较麻烦。

练习

下载练习：使用SAX解析XML（推荐使用IDE练习插件快速下载）

小结

- SAX是一种流式解析XML的API；
- SAX通过事件触发，读取速度快，消耗内存少；
- 调用方必须通过回调方法获得解析过程中的数据。

使用Jackson

前面我们介绍了DOM和SAX两种解析XML的标准接口。但是，无论是DOM还是SAX，使用起来都不直观。

观察XML文档的结构：

```
<?xml version="1.0" encoding="UTF-8" ?>
<book id="1">
    <name>Java核心技术</name>
    <author>Cay S. Horstmann</author>
    <isbn lang="CN">1234567</isbn>
    <tags>
        <tag>Java</tag>
        <tag>Network</tag>
    </tags>
    <pubDate/>
</book>
```

我们发现，它完全可以对应到一个定义好的JavaBean中：

```
public class Book {
    public long id;
    public String name;
    public String author;
    public String isbn;
    public List<String> tags;
    public String pubDate;
}
```

如果能直接从XML文档解析成一个JavaBean，那比DOM或者SAX不知道容易到哪里去了。

幸运的是，一个名叫Jackson的开源的第三方库可以轻松做到XML到JavaBean的转换。我们要使用Jackson，先添加两个Maven的依赖：

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.10.1</version>
</dependency>
<dependency>
    <groupId>org.codehaus.woodstox</groupId>
    <artifactId>woodstox-core-asl</artifactId>
    <version>4.4.1</version>
</dependency>
```

然后，定义好JavaBean，就可以用下面几行代码解析：

```
InputStream input =
Main.class.getResourceAsStream("/book.xml");
JacksonXmlModule module = new JacksonXmlModule();
XmlMapper mapper = new XmlMapper(module);
Book book = mapper.readValue(input, Book.class);
System.out.println(book.id);
System.out.println(book.name);
System.out.println(book.author);
System.out.println(book.isbn);
System.out.println(book.tags);
System.out.println(book.pubDate);
```

注意到`XmlMapper`就是我们需要创建的核心对象，可以用`readValue(InputStream, Class)`直接读取XML并返回一个JavaBean。运行上述代码，就可以直接从Book对象中拿到数据：

```
1
Java核心技术
Cay S. Horstmann
1234567
[Java, Network]
null
```

如果要解析的数据格式不是Jackson内置的标准格式，那么需要编写一点额外的扩展来告诉Jackson如何自定义解析。这里我们不做深入讨论，可以参考Jackson的官方文档。

练习

下载练习：使用Jackson解析XML（推荐使用IDE练习插件快速下载）

使用JSON

前面我们讨论了XML这种数据格式。XML的特点是功能全面，但标签繁琐，格式复杂。在Web上使用XML现在越来越少，取而代之的是JSON这种数据结构。

JSON是JavaScript Object Notation的缩写，它去除了所有JavaScript执行代码，只保留JavaScript的对象格式。一个典型的JSON如下：

```
{
    "id": 1,
    "name": "Java核心技术",
    "author": {
        "firstName": "Abc",
        "lastName": "Xyz"
    },
    "isbn": "1234567",
    "tags": ["Java", "Network"]
}
```

JSON作为数据传输的格式，有几个显著的优点：

- JSON只允许使用UTF-8编码，不存在编码问题；
- JSON只允许使用双引号作为key，特殊字符用`\`转义，格式简单；
- 浏览器内置JSON支持，如果把数据用JSON发送给浏览器，可以用JavaScript直接处理。

因此，JSON适合表示层次结构，因为它格式简单，仅支持以下几种数据类型：

- 键值对：`{"key": value}`
- 数组：`[1, 2, 3]`
- 字符串：`"abc"`
- 数值（整数和浮点数）：`12.34`
- 布尔值：`true`或`false`
- 空值：`null`

浏览器直接支持使用JavaScript对JSON进行读写：

```
// JSON string to JavaScript object:
jsObj = JSON.parse(jsonStr);

// JavaScript object to JSON string:
jsonStr = JSON.stringify(jsObj);
```

所以，开发Web应用的时候，使用JSON作为数据传输，在浏览器端非常方便。因为JSON天生适合JavaScript处理，所以，绝大多数REST API都选择JSON作为数据传输格式。

现在问题来了：使用Java如何对JSON进行读写？

在Java中，针对JSON也有标准的JSR 353 API，但是我们在前面讲XML的时候发现，如果能直接在XML和JavaBean之间互相转换是最好的。类似的，如果能直接在JSON和JavaBean之间转换，那么用起来就简单多了。

常用的用于解析JSON的第三方库有：

- Jackson
- Gson
- Fastjson
- ...

注意到上一节提到的那个可以解析XML的浓眉大眼的Jackson也可以解析JSON！因此我们只需要引入以下Maven依赖：

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.10.0</version>
</dependency>
```

就可以使用下面的代码解析一个JSON文件：

```
InputStream input =
Main.class.getResourceAsStream("/book.json");
ObjectMapper mapper = new ObjectMapper();
// 反序列化时忽略不存在的JavaBean属性:
mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PR
OPERTIES, false);
Book book = mapper.readValue(input, Book.class);
```

核心代码是创建一个`ObjectMapper`对象。关闭`DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES`功能使得解析时如果JavaBean不存在该属性时解析不会报错。

把JSON解析为JavaBean的过程称为反序列化。如果把JavaBean变为JSON，那就是序列化。要实现JavaBean到JSON的序列化，只需要一行代码：

```
String json = mapper.writeValueAsString(book);
```

要把JSON的某些值解析为特定的Java对象，例如`LocalDate`，也是完全可以的。例如：

```
{
    "name": "Java核心技术",
    "pubDate": "2016-09-01"
}
```

要解析为：

```
public class Book {
    public String name;
    public LocalDate pubDate;
}
```

只需要引入标准的JSR 310关于JavaTime的数据格式定义：

```
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>2.10.0</version>
</dependency>
```

然后，在创建`ObjectMapper`时，注册一个新的`JavaTimeModule`：

```
ObjectMapper mapper = new
ObjectMapper().registerModule(new JavaTimeModule());
```

有些时候，内置的解析规则和扩展的解析规则如果都不满足我们的需求，还可以自定义解析。

举个例子，假设`Book`类的`isbn`是一个`BigInteger`：

```
public class Book {
    public String name;
    public BigInteger isbn;
}
```

但JSON数据并不是标准的整形格式：

```
{
    "name": "Java核心技术",
    "isbn": "978-7-111-54742-6"
}
```

直接解析，肯定报错。这时，我们需要自定义一个`IsbnDeserializer`，用于解析含有非数字的字符串：

```
public class IsbnDeserializer extends
JsonDeserializer<BigInteger> {

```

```
    public BigInteger deserialize(JsonParser p,
DeserializationContext ctxt) throws IOException,
JsonProcessingException {
        // 读取原始的JSON字符串内容:
        String s = p.getValueAsString();
        if (s != null) {
            try {
                return new BigInteger(s.replace("-", ""));
            } catch (NumberFormatException e) {
                throw new JsonParseException(p, s, e);
            }
        }
        return null;
    }
}
```

然后，在`Book`类中使用注解标注：

```
public class Book {
    public String name;
    // 表示反序列化isbn时使用自定义的IsbnDeserializer:
    @JsonDeserialize(using = IsbnDeserializer.class)
    public BigInteger isbn;
}
```

类似的，自定义序列化时我们需要自定义一个`IsbnSerializer`，然后在`Book`类中标注`@JsonSerialize(using = ...)`即可。

练习

下载练习：使用Jackson解析JSON（推荐使用IDE练习插件快速下载）

小结

- JSON是轻量级的数据表示方式，常用于Web应用；
- Jackson可以实现JavaBean和JSON之间的转换；
- 可以通过Module扩展Jackson能处理的数据类型；
- 可以自定义`JsonSerializer`和`JsonDeserializer`来定制序列化和反序列化。