基于流的XML解析使用心得

基于指针流需要关注的类

- `XMLStreamConstants`
- `XMLStreamReader`
- `XMLInputFactory`
- `XMLOutputFactory`
- `XMLStreamWriter`（基于流的解析写的效率比较低，最好不要使用）

其中本次的使用主要用于解析，因此先重点关注前三个类的使用

基于指针流的读取解析过程

首先无论是基于指针流的读取还是基于事件流的读取，**都需要使用相应的事件类型（也就是事件代码、消息，以下简称事件），代表当前文档的读取状态，事件可以参看`XMLStreamConstants`类中的定义。**这里列出几个需要了解的：

- **START_DOCUMENT：代表xml文档开头**
- **START_ELEMENT：代表xml文档中任一元素的开始位置**
- **END_ELEMENT：代表xml文档中任一元素的结束位置**
- **END_DOCUMENT：代表xml文档末尾**
- CDATA：表示当前元素内容是CDATA部分
- COMMENT：表示注释
- **CHARACTERS：表示xml标签元素值**
- **ATTRIBUTE：表示元素的属性**
- **PROCESSING_INSTRUCTION：事件是处理指令**

创建`XMLStreamReader`类

要创建一个`XMLStreamReader`需要使用工厂类`XMLInputFactory`来创建，需要先创建一个`XMLInputFactory`对象，通过该工厂对象创建`XMLStreamReader`

可以参考下面代码：

```
XMLInputFactory factory = XMLInputFactory.newFactory();
// XMLInputFactory factory = XMLInputFactory.newInstance();// 这样创建也可以
XMLStreamReader xmlStreamReader = null;
xmlStreamReader =
factory.createXMLStreamReader(newFileReader("Launcher_backup.xml"));
```

使用`factory.createXMLStreamReader()`创建`XMLStreamReader`对象，参数可以接受`Source`、`Reader`、`InputStream`这三类对象。

处理事件类型（事件代码）

处理事件类型过程非常关键，也是解析XML文件的核心部分。一般使用循环来处理事件代码。

在创建完`XMLStreamReader`对象之后：

1. **首先调用`hasNext()`，判断是否有更多的解析事件代码，如果有则返回`true`，否则返回`false`**
2. **然后调用`next()`，来获取下一个解析事件代码。**

通过这两个方法来迭代解析过程，代码可以参考：

```
do{
 // 获取当前事件类型
 int i = xmlStreamReader.getEventType();
 ...
 if(xmlStreamReader.hasNext()){
    xmlStreamReader.next();
 }else{
    break;
 }
}while(true)
```

其中调用`xmlStreamReader.getEventType()`获取当前事件类型。中间的`...`部分就是处理事件类型的代码，参考：

```
if(XMLStreamConstants.START_ELEMENT == i){
    // 文档中标签元素开头处
    // 通常在这里使用getLocalName() | getName() 获取标签名
}
if(XMLStreamConstants.CHARACTERS == i && !xmlStreamReader.isWhiteSpace()){
    // 文档中各类元素值
    // 在这一块通常使用getText()获取元素值
    // isWhiteSpace()去掉空白区域
}
if(XMLStreamConstants.END_ELEMENT == i){
    // 文档中标签元素结尾处
    // 通常在这里使用getLocalName() | getName() 获取标签名
}
if(XMLStreamConstants.CDATA == i){
    ...
}
if(XMLStreamConstants.START_DOCUMENT == i){
    // 文档头
}
if(XMLStreamConstants.END_DOCUMENT == i){
    // 文档尾
}
```

第二段代码也可以使用`switch`。

可能有些人会对一开始的迭代`do...while`写法不太理解，这里解释下，

当我们调用：

```
xmlStreamReader =
factory.createXMLStreamReader(newFileReader("Launcher_backup.xml"));
```

创建一个`XmlStreamReader`对象的完成之后，指针会指向第一个事件类型，也就是文档开始事件（START_DOCUMENT），这个时候直接调用`getEventType()`，返回的就是`START_DOCUMENT`，然后后面的`hasNext()`，**实际上判断的是`START_DOCUMENT`事件之后还有没有事件！，如果有调用`next()`使指针指向下一个事件，然后再通过`getEventType()`获取。如果当前事件为`END_DOCUMENT`，则`hasNext()`一定返回`false`，这时候就可以退出循环关闭文档**

关闭文档

调用`close()`

```
xmlStreamReader.close();
```

`XmlStreamReader`一些方法注意

`getLocalName()` VS `getName()`

- `getName()`作用于START_ELEMENT 和 END_ELEMENT，**也就是说他的前置条件是当前事件类型是START_ELEMENT 和 END_ELEMENT的时候才能使用**
- `getLocalName()`作用于START_ELEMENT、END_ELEMENT 和 ENTITY_REFERENCE。**在处理xml实体引用的时候，只能用`getLocalName()`**

getElementText()

```
parser must be on START_ELEMENT to read next text
```

这个方法的用途可以参考这段关于它的源代码：

```
if(getEventType() != XMLStreamConstants.START_ELEMENT) {
     throw new XMLStreamException("parser must be on START_ELEMENT to read next
text", getLocation());
}

int eventType = next();
StringBuffer content = new StringBuffer();
while(eventType != XMLStreamConstants.END_ELEMENT) {
if(eventType == XMLStreamConstants.CHARACTERS
           || eventType == XMLStreamConstants.CDATA
           || eventType == XMLStreamConstants.SPACE
           || eventType == XMLStreamConstants.ENTITY_REFERENCE) {

         buf.append(getText());
  } else if(eventType == XMLStreamConstants.PROCESSING_INSTRUCTION
           || eventType == XMLStreamConstants.COMMENT) {

         // skipping

 } else if(eventType == XMLStreamConstants.END_DOCUMENT) {

         throw new XMLStreamException("unexpected end of document when reading
element text content", this);

 } else if(eventType == XMLStreamConstants.START_ELEMENT) {

         throw new XMLStreamException("element text content may not contain
START_ELEMENT", getLocation());

 } else {

         throw new XMLStreamException("Unexpected event type "+eventType,
getLocation());
```

```
 }
    eventType = next();
}
return buf.toString();
```

getName ()

`getName ()`方法必须使用在以下场景：START_ELEMENT, END_ELEMENT

```
java.lang.IllegalStateException: Illegal to call getName() when event type is
CHARACTERS. Valid states are START_ELEMENT, END_ELEMENT
```

nextTag()

`nextTag()`可以跳过任何的空白部分（`isWhiteSpace()` == true） 的情况，理论上是`next()`的方便版本，**但这个方法只能返回两个标志START_ELEMENT, END_ELEMENT**，也就是说他的后置条件必须是**START_ELEMENT, END_ELEMENT**，**如果你当前的标志是CHARACTERS且`isWhiteSpace()`==false，也就是说该标签元素上有有效的内容，不为空白，则会抛出异常！**，具体查看他的源代码：

```
int eventType = next();
 while((eventType == XMLStreamConstants.CHARACTERS && isWhiteSpace()) // skip
whitespace
 || (eventType == XMLStreamConstants.CDATA && isWhiteSpace())         // skip
whitespace
 || eventType == XMLStreamConstants.SPACE
 || eventType == XMLStreamConstants.PROCESSING_INSTRUCTION
 || eventType == XMLStreamConstants.COMMENT
 )
 {
    eventType = next();
 }
 // !!!!重点在这里
 if (eventType != XMLStreamConstants.START_ELEMENT
     && eventType != XMLStreamConstants.END_ELEMENT)
 {
    throw new String XMLStreamException("expected start or end tag",
getLocation());
 }
 return eventType;
```

基于事件流需要关注的类

- `XMLEvent`

- `XMLEventReader`
- `XMLInputFactory`
- `XMLOutputFactory`
- `XMLEventWriter`（基于流的解析写的效率比较低，最好不要使用）

同样重点关注前三个类。

基于事件流的读取解析过程

创建`XMLEventReader`类

要创建一个`XMLStreamReader`需要使用工厂类`XMLInputFactory`来创建，需要先创建一个`XMLInputFactory`对象，通过该工厂对象创建`XMLStreamReader`

可以参考下面代码：

```
XMLInputFactory factory = XMLInputFactory.newFactory();
// XMLInputFactory factory = XMLInputFactory.newInstance();// 这样创建也可以
XMLEventReader xmlEventReader = null;
xmlEventReader =
factory.createXMLEventReader(newFileReader("Launcher_backup.xml"));
```

使用`factory.createXMLStreamReader()`创建`XMLStreamReader`对象，参数可以接受`Source`、`Reader`、`InputStream`这三类对象。

基于事件流的解析的使用的方式与基于指针流的使用方式非常不同。**`XMLStreamReader`初始化的时候就指向`START_DOCUMENT`事件，而`XMLEventReader`初始化的时候需要调用`nextEvent()`方法才会指向`START_DOCUMENT`事件。**也就是说当`XMLEventReader`初始化的时候，第一次调用`hasNext()`判断的是有没有`START_DOCUMENT`事件。

处理事件类型（消息）

与指针流的处理不同，事件流的迭代代码非常简单：

```
while (reader.hasNext()) {
    XMLEvent event = reader.nextEvent();
    ...
}
```

`XMLEvent`类是一个关键处理类，事件流的解析简化了`XMLEventReader`接口的方法，使接口只关心返回消息（`XMLEvent`）对象，具体的解析内容，交给消息对象来处理，因此首先，需**要调用`XMLEventReader`接口的`nextEvent()`，获取下一个消息，同时这个消息出队。如果只想获取下一个消息，但不想这个消息出队，可以使用`peek()`**

事件流的消息处理和指针流的差不多，可以参考下面代码：

```
while (reader.hasNext()) {
    XMLEvent event = reader.nextEvent();

    int i = event.getEventType();
    if(XMLStreamConstants.START_ELEMENT == i){
          StartElement element = event.asStartElement();
    }
    if(XMLStreamConstants.CHARACTERS == i){
```

```
            Characters characters = event.asCharacters();
        }
        if(XMLStreamConstants.END_ELEMENT == i){
            EndElement endElement = event.asEndElement();
        }
        if(XMLStreamConstants.CDATA == i){

        }
        if(XMLStreamConstants.START_DOCUMENT == i){

        }
        if(XMLStreamConstants.END_DOCUMENT == i){

        }
    }
```

重点看这段代码

```
int i = event.getEventType();
        if(XMLStreamConstants.START_ELEMENT == i){
            StartElement element = event.asStartElement();
        }
        if(XMLStreamConstants.CHARACTERS == i){
            Characters characters = event.asCharacters();
        }
        if(XMLStreamConstants.END_ELEMENT == i){
            EndElement endElement = event.asEndElement();
        }
```

首先`XMLEvent`的对象`event`，通过调用`event.getEventType()`获取事件类型，然后通过判断事件类型，将类型转为具体的类型，事件流具体的类型定义只有三种，也是最需要关注的三种：`StartElement`、`Characters`、`EndElement`。

`StartElement`、`EndElement`

这个类需要重点关注一个方法：`getName()`，这个方法返回一个`QName`对象，代表标签名，使用`QName`对象的`toString()`可以返回标签名的字符串值。

`Characters`

这个类需要重点关注一个方法：`getData()`，这个方法返回一个`String`对象，代表标签值。

关闭文档

调用`close()`

```
xmlStreamReader.close();
```