Linguagem de Definição de Dados

Análise e Desenvolvimento de Sistemas

Paulo Maurício Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

23 de agosto de 2023

Parte X

StAX – Java Streaming API for XML

Introdução I

- O objetivo principal da API StAX é dar "controle do parser para o programador expondo um iterador simples. Isto permite que o programador solicite o próximo evento (puxar o evento) e permite o estado a ser armazenado de forma procedural."
- SAX (*push parsing*): parser XML envia dados XML para o cliente quando encontra elementos constituintes da linguagem.
- StAX (*pull parsing*): aplicação chama métodos no parser XML quando ele precisa interagir com um elemento constituinte da linguagem.
- Vantagens
  - Precisa de pouca memória para executar, pois o documento é analisado sequencialmente.
  - Pode ser usado para criar documentos de tamanho arbitrário.
  - A aplicação informa o parser quando está pronto para receber informações.
- Desvantagens

Introdução II

- Não pode revisitar conteúdo.
- Validação: Como não lê todo o documento de uma vez, se o documento possuir um erro de sintaxe, StAX só o reportará quando o encontrar.

Formas de manipulação I

- StAX pode ser utilizado para manipular XML, tanto para leitura como para escrita.
- Podem seguir o modelo de fluxos ou eventos.
- Um leitor de fluxo extrai o próximo item de um fluxo de entrada via um cursor.
- Um escritor de fluxo escreve o próximo item para um fluxo de saída na posição de um cursor.
- O cursor só pode apontar para um item por vez, e sempre se move para frente.
- Leitores e escritores de fluxo são apropriados para ambientes com restrição de memória pois permite criar código menor e mais eficiente.

Formas de manipulação II

- Um leitor baseado em eventos extrai o próximo item de um fluxo obtendo um evento.
- Um escritor baseado em eventos escreve o próximo item em um fluxo adicionando um evento ao fluxo de saída.
- Leitores e escritores baseados em eventos não possuem o conceito de cursor.
- São apropriados para criar pipelines (sequências de componentes que transformam componentes prévios de entrada e passam a saída transformada para o próximo componente na sequência).

Lendo documentos XML

- O primeiro passo é obter uma instância da classe `XMLInputFactory`.

```
XMLInputFactory xmlif = XMLInputFactory.newFactory();
```

- Caso necessário, podemos passar um conjunto de parâmetros para definir o comportamento do parser.
- Os parâmetros estão definidos na própria classe.
- Se desejarmos que o espaço de nomes seja informado podemos informar assim:

```
xmlif = setProperty(XMLInputFactory.IS_NAMESPACE_AWARE, true);
```

Lendo XML I

Fluxos

- Para criar um leitor de XML usando fluxos, chamaremos um dos métodos `createXMLStreamReader()`.
- Por exemplo, para lermos um arquivo XML:

```
Reader reader = new FileReader("recipe.xml");
XMLStreamReader xmlsr = xmlif.createXMLStreamReader(reader);
```

- O método `boolean hasNext()` retorna `true` quando ainda existe um item a ser processado.
- O método `int next()` move o cursor um item e retorna o código do tipo de item encontrado.
- Ao invés de comparar os tipos com inteiros, podemos usar constantes:

Lendo XML II

Fluxos

```
while (xmlsr.hasNext()) {
  switch (xmlsr.next())  {
    case XMLStreamReader.START_ELEMENT:
    // Do something at element start.
      break;
    case XMLStreamReader.END_ELEMENT:
    // Do something at element end.
  }
}
```

- Ao final do processamento, é interessante chamar o método `void close()` para liberar recursos associados ao leitor de fluxo.
- Vejamos um exemplo com os itens mais comuns de XML:

Lendo XML III

Fluxos

```
XMLInputFactory xif = XMLInputFactory.newFactory();
XMLStreamReader xsr = xif.createXMLStreamReader(new FileReader("input.
    xml"));
while(xsr.hasNext()) {
  switch(xsr.next()) {
    case XMLStreamReader.START_DOCUMENT:
    case XMLStreamReader.START_ELEMENT:
      String qName = xsr.getLocalName();
      int numAttributes = xsr.getAttributeCount();
    case XMLStreamReader.CHARACTERS:
      String data = xsr.getText();
    case XMLStreamReader.END_ELEMENT:
      String qName = xsr.getLocalName();
    case XMLStreamReader.END_DOCUMENT:
  }
}
xsr.close();
```

Lendo XML I

Eventos

- Para criar um leitor de XML usando eventos, chamaremos um dos métodos `createXMLEventReader()`.
- Por exemplo, para lermos um arquivo XML:

```
Reader reader = new FileReader("recipe.xml");
XMLEventReader xmler = xmlif.createXMLEventReader(reader);
```

- O método `boolean hasNext()` retorna `true` quando ainda existe um evento a ser processado.
- O método `XMLEvent nextEvent()` retorna o próximo evento como um objeto.
- O método `int getEventType()` retorna o tipo de evento, similar ao visto anteriormente.

Lendo XML II

Eventos

```
while (xmler.hasNext()) {
  switch (xmler.nextEvent().getEventType()) {
    case XMLEvent.START_ELEMENT:
    // Do something at element start.
      break;
    case XMLEvent.END_ELEMENT:
    // Do something at element end.
  }
}
```

- Ao final do processamento, é interessante chamar o método `void close()` para liberar recursos associados ao leitor de fluxo.
- Vejamos um exemplo com os itens mais comuns de XML:

Lendo XML III

Eventos

```
XMLInputFactory xif = XMLInputFactory.newFactory();
XMLEventReader xsr = xif.createXMLEventReader(new FileReader("input.xml
    "));
while(xsr.hasNext()) {
  XMLEvent event = xsr.nextEvent();
  switch(eventType) {
    case XMLStreamConstants.START_DOCUMENT:
    case XMLStreamConstants.START_ELEMENT:
      StartElement startElement = event.asStartElement();
      String qName = startElement.getName().getLocalPart();
      Iterator<Attribute> attributes = startElement.getAttributes();
    case XMLStreamConstants.CHARACTERS:
      Characters characters = event.asCharacters();
      String data = characters.getData();
    case XMLStreamConstants.END_ELEMENT:
      EndElement endElement = event.asEndElement();
      String qName = endElement.getName().getLocalPart();
    case XMLStreamConstants.END_DOCUMENT:
  }
}
```

Escrevendo documentos XML

- O primeiro passo é obter uma instância da classe `XMLOutputFactory`.

```
XMLOutputFactory xmlof = XMLOutputFactory.newFactory();
```

- Caso necessário, podemos passar um conjunto de parâmetros para definir o comportamento do parser.
- Os parâmetros estão definidos na própria classe.
- Se desejarmos que escritor lide com o espaço de nomes com mínima intervenção da aplicação:

```
xmlif = setProperty(XMLInputFactory.IS_REPAIRING_NAMESPACES, true);
```

Escrevendo XML I

Fluxos

- Para criar um escritor de XML usando fluxos, chamaremos um dos métodos `createXMLStreamWriter()`.
- Por exemplo, para escrevermos em um arquivo XML:

```
Writer writer = new FileWriter("recipe.xml");
XMLStreamWriter xmlsw = xmlof.createXMLStreamWriter(writer);
```

- **void** `close()` fecha o fluxo de saída, liberando recursos associados.
- **void** `flush()` escreve quaisquer dados em cache na saída.
- **void** `setPrefix(String prefix, String uri)` associa o prefixo do espaço de nomes à URI.

Escrevendo XML II

Fluxos

- **void** `writeAttribute(String localName, String value)` escreve o atributo com seu valor na saída.
- **void** `writeCharacters(String text)` escreve o texto na saída.
- **void** `writeEndDocument()` fecha todas as tags de abertura e fecha o documento.
- **void** `writeEndElement()` fecha todas a tag de abertura do elemento corrente.
- **void** `writeStartDocument()` escreve a declaração XML na saída.
- **void** `writeStartElement(String namespaceURI, String localName)` escreve a tag de abertura na saída.

Escrevendo XML III

Fluxos

```
XMLOutputFactory xof = XMLOutputFactory.newInstance();
XMLStreamWriter xsw = new xof.createXMLStreamWriter(new FileWriter("
    results.xml"));
xsw.writeStartDocument();
xsw.writeStartElement("results");
xsw.writeStartElement("dude");
xsw.writeCharacters("texto");
xsw.writeEndElement();
xsw.writeEndElement();
xsw.writeEndDocument();
xsw.close();
```

Escrevendo XML I

Eventos

- Para criar um escritor de XML usando eventos, chamaremos um dos métodos `createXMLEventWriter()`.
- Por exemplo, para escrevermos em um arquivo XML:

```
Writer writer = new FileWriter("recipe.xml");
XMLEventWriter xmlew = xmlof.createXMLEventWriter(writer);
```

- O método `void add(XMLEvent event)` adiciona eventos que descreve os itens do XML na saída.
- A classe `EventFactory` permite criar objetos que representam o XML.

Escrevendo XML II

Eventos

```
XMLOutputFactory xmlof = XMLOutputFactory.newFactory();
XMLEventWriter xmlew = xmlof.createXMLEventWriter(System.out);
XMLEventFactory xmlef = XMLEventFactory.newFactory();
xmlew.add(xmlef.createStartDocument());
xmlew.add(xmlef.createStartElement("", null, "results"));
xmlew.add(xmlef.createStartElement("", null, "dude"));
xmlew.add(xmlef.createAttribute("asdas", "123"));
xmlew.add(xmlef.createCharacters("texto"));
xmlew.add(xmlef.createEndElement("", null, "dude"));
xmlew.add(xmlef.createEndElement("", null, "results"));
xmlew.add(xmlef.createEndDocument());
xmlew.flush();
xmlew.close();
```