Streaming API for XML

Asst. Prof. Dr. Kanda Runapongsa Saikaew

(krunapon@kku.ac.th)

Dept. of Computer Engineering

Khon Kaen University

Agenda

- What is StAX?
- Why StAX?
- StAX API
- Using StAX
- Sun's Streaming Parser Implementation

What is StAX? (1/2)

- StAX stands for Streaming API for XML (StAX)
- A streaming Java-based, event-driven, pull-parsing API for reading and writing XML documents
- StAX enables you to create bidirectional XML parsers that are fast, relatively easy to program, and have a light memory footprint

What is StAX? (2/2)

- StAX provides a standard, bidirectional pull parser interface for streaming XML processing
- Offer a simpler programming model than SAX
- Process with more efficient memory management than DOM
- Enable developers to parse and modify XML streams as events

Push APIs

- The common streaming APIs like SAX are all push APIs
- Feed the content of the document to the application as soon as they see it
- Does not pay attention to whether the application is ready to receive that data or not
- Cause patterns that are unfamiliar and uncomfortable to many developers

Pull APIs vs. Push APIs

- In a pull API, the client program asks the parser for the next piece of information
  - Not the parser tell the client program when the next datum is available
- In a pull API the client program drives the parser
- In a push API the parser drives the client

Pull Parsing vs. Push Parsing (1/2)

- Streaming pull parsing
  - The client only gets (pulls) XML data when it explicitly asks for it
  - The client controls the application thread
- Streaming push parsing
  - The parser sends the data whether or not the client is ready to use it at that time
  - The parser controls the application thread

Pull Parsing vs. Push Parsing (2/2)

- Pull parsing libraries can be much smaller
- Pull clients can read multiple documents at one time with a single thread
- Pull parser can filter XML documents such that elements unnecessary to the client can be ignored

Why StAX?

- The primary goal of the StAX API is to give “parsing control to the programming by exposing a simple iterator based API
- This allows the programmer to ask for the next event (pull the event) and allow state to be stored in procedural fashion
- StAX was created to address limitations in the two prevalent parsing APIs, SAX and DOM

StAX Use Cases (1/2)

- Data binding
  - Unmarshalling an XML document
  - Marshalling an XML document
  - Parallel document processing
  - Wireless communication
- SOAP message processing
  - Parsing simple predictable structures
  - Parsing graph representations with forward references
  - Parsing WSDL

StAX Use Cases (2/2)

- Virtual data sources
  - Viewing as XML data stored in databases
  - Viewing data in Java objects created by XML data binding
  - Navigating a DOM tree as a stream of events
- Parsing specific XML vocabularies
- Pipelined XML processing

StAX vs. SAX

- StAX-enabled clients are generally easier to code than SAX clients
- StAX is a bidirectional API
  - It can both read and write XML documents
  - SAX is read only
- SAX is a push API whereas StAX is pull

XML Parser API Feature Summary (1/2)

Feature	StAX	SAX	DOM	TrAX
API Type	Pull, streaming	Push, streaming	In memory tree	XSLT rule
Ease of use	High	Medium	High	Medium
XPath Capability	No	No	Yes	Yes
CPU and Memory Efficiency	Good	Good	Varies	Varies

XML Parser API Feature Summary (2/2)

Feature	StAX	SAX	DOM	TrAX
Forward Only	Yes	Yes	No	No
Read XML	Yes	Yes	Yes	Yes
Write XML	Yes	No	Yes	Yes
Create, Read, Update, Delete	No	No	Yes	No

StAX API

- The StAX API exposes methods for iterative, event-based processing of XML documents
- The StAX API is really two distinct API sets
  - A cursor API
  - An iterator API

Using StAX

In general, StAX programmers create XML stream readers, writers, and events by using classes

- XMLInputFactory
- XMLOutputFactory
- XMLEventFactory

Cursor API

- The StAX cursor API represents a cursor with which you can walk an XML document from beginning to end
- This cursor can point to one thing at a time
- It always moves forward, never backward, usually one infoset element at a time

Cursor Interfaces

- The two main cursor interfaces are XMLStreamReader and XMLStreamWriter
- XMLStreamReader includes accessor methods for all possible information retrievable from the XML information model
- XMLStreamWriter provides methods that corresponds to StartElement and EndElement event types

XMLStreamReader

```
public interface XMLStreamReader {
    public int next() throws
    XMLStreamException;
    public boolean hasNext() throws
    XMLStreamException;
    public String getText();
    public String getLocalName();
    public String getNamespaceURI();
    // ... other methods not shown }
```

XHTMLOutliner (1/7)

```
package stax_parser;
import javax.xml.stream.*;
import java.net.URL;
import java.io.*;
import java.util.Properties;
public class XHTMLOutliner {
public static void main(String[] args) {
    if (args.length == 0) {
        System.err.println("Usage: java XHTMLOutliner url");
        return;
    }
    String input = args[0];
```

XHTMLOutliner (2/7)

```
try {
    setProxy();
    URL u = new URL(input);
    InputStream in = u.openStream();
    XMLInputFactory factory =
        XMLInputFactory.newInstance();
    XMLStreamReader parser =
        factory.createXMLStreamReader(in);
    int inHeader = 0;
    for (int event = parser.next();
        event != XMLStreamConstants.END_DOCUMENT;
        event = parser.next()) {
```

XHTMLOutliner (3/7)

```
switch (event) {
    case XMLStreamConstants.START_ELEMENT:
        if (isHeader(parser.getLocalName())) {
                inHeader++;
        }
        break;
    case XMLStreamConstants.END_ELEMENT:
        if (isHeader(parser.getLocalName())) {
                inHeader--;
                if (inHeader == 0) System.out.println();
        }
        break;
```

XHTMLOutliner (4/7)

```
case XMLStreamConstants.CHARACTERS:
    if (inHeader > 0)
        System.out.print(parser.getText());
    break;
case XMLStreamConstants.CDATA:
    if (inHeader > 0)
        System.out.print(parser.getText());
        break;
    } // end switch
} // end for
```

XHTMLOutliner (5/7)

```
            parser.close();
            System.out.println("Done processing");
        } catch (XMLStreamException ex) {
            System.out.println(ex);
        } catch (IOException ex) {
            System.out.println("IOException while
                    parsing " + input);
        } // end try-catch
    } // end main
```

XHTMLOutliner (6/7)

```
private static boolean isHeader(String
    name) {
    if (name.equals("h1")) return true;
    if (name.equals("h2")) return true;
    if (name.equals("h3")) return true;
    if (name.equals("h4")) return true;
    if (name.equals("h5")) return true;
     if (name.equals("h6")) return true;
    return false;
}
```

XHTMLOutliner (7/7)

```
  private static void setProxy(){
    Properties systemSettings =
   System.getProperties();
        systemSettings.put("proxySet", "true");
        systemSettings.put("http.proxyHost","2
   02.12.97.116") ;
        systemSettings.put("http.proxyPort",
   "8088") ;
}
```

XHTMLOutliner: Sample Input

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head> <title>I Love HTML</title>
<meta http-equiv="Content-Language" content="en-us“
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
    1" /> </head>
<body>
    <h1>Top 10 Strategic Technologies for 2008</h1> <h2>By
    Gartner</h2>
    <h3>Green IT</h3>
    <h4>Scheduling decisions for workloads on servers will begin to
    consider power efficiency as a key placement attribute.</h4>
</body>
</html>
```

XHTMLOutliner: Sample Output

Top 10 Strategic Technologies for 2008

By Gartner

Green IT

Scheduling decisions for workloads on servers will begin to consider power efficiency as a key placement attribute.

XMLStreamWriter

```
public interface XMLStreamWriter {
   public void writeStartElement(String
   localName) \     throws
   XMLStreamException;
   public void writeEndElement() \     throws
   XMLStreamException;
   public void writeCharacters(String text)
   \     throws   XMLStreamException;
   // ... other methods not shown
}
```

Writer1 (1/4)

```
package staxtutorial;

import java.io.*;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamWriter;

public class Writer1 {
    public static void main(String[] args) {
        try {
            // output file name
            String fileName = "nation.xml";
```

Writer1 (2/4)

```
// write an output factory
    XMLOutputFactory xof =
        XMLOutputFactory.newInstance();

// write an xml stream writer
    XMLStreamWriter xtw =
        xof.createXMLStreamWriter(new
        FileWriter(fileName));

  // xml declaration with encoding setting to tis-620
    xtw.writeStartDocument("tis-620", "1.0");
```

Writer1 (3/4)

```
xtw.writeStartElement("nation");
xtw.writeStartElement("name");
xtw.writeCharacters("ประเทศไทย");
xtw.writeEndElement(); // end name element
xtw.writeStartElement("location");
xtw.writeCharacters("Southeast Asia");
xtw.writeEndElement(); // end location element
xtw.writeEndElement();  // end nation element
xtw.writeEndDocument();
```

Writer1 (4/4)

```
            // write any cached data to the underlying
          // output stream
          xtw.flush();
            xtw.close();
        } catch (Exception ex) {
            System.err.println("Exception occurred
    while running writer1");
            ex.printStackTrace();
        }
        System.out.println("Done");
    }
}
```

File nation.xml (Output of Writer1)

```
<?xml version="1.0" encoding="tis-620"?>
<nation>
    <name>ประเทศไทย</name>
    <location>Southeast Asia</location>
</nation>
```

Writer2 (with Namespaces) (1/6)

```
package staxtutorial;

import java.io.*;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamWriter;

public class Writer2 {

    // Namespaces
   private static final String BOOK =
    "http://www.kku.ac.th/bookstore";
   private static final String XHTML =
    "http://www.w3.org/1999/xhtml";
```

Writer2 (with Namespaces) (2/6)

```
public static void main(String[] args) {
        try {
            String fileName = "book.xml";
            // Create an output factory
            XMLOutputFactory xof =
                    XMLOutputFactory.newInstance();
            // Create an XML stream writer
            XMLStreamWriter xtw =
    xof.createXMLStreamWriter(new FileWriter(fileName));

            // Write XML prologue
            xtw.writeStartDocument();
```

Writer2 (with Namespaces) (3/6)

```
// Now start with the root element
// Declare XHTML prefix
xtw.setPrefix("h",XHTML);

xtw.writeStartElement(XHTML,"html");

// Declare XHTML namespace in
// the scope of the html element
xtw.writeNamespace("h",XHTML);
```

Writer2 (with Namespaces) (4/6)

```
xtw.writeStartElement("book");
xtw.setDefaultNamespace(BOOK);
xtw.writeNamespace("", BOOK);
xtw.writeStartElement("name");
xtw.writeAttribute("isbn",
        "123-456-7890");
xtw.writeCharacters("XML");
xtw.writeEndElement(); // end name
```

Writer2 (with Namespaces) (5/6)

```
xtw.writeStartElement("chapters");
xtw.writeStartElement("chapter");
xtw.writeCharacters("Intro to XML");
xtw.writeEndElement(); // end chapter

xtw.writeStartElement("chapter");
xtw.writeCharacters("XML Schema");
xtw.writeEndElement(); // end chapter
```

Writer2 (with Namespaces) (6/6)

```
            xtw.writeEndElement(); // end chapters
            xtw.writeEndElement(); // end book
            xtw.writeEndDocument();
            xtw.flush();
            xtw.close();
        } catch (Exception ex) {
            System.err.println("Exception occurred
while running Writer2");
            ex.printStackTrace();
        }
        System.out.println("Done");
    }
```

File book.xml (Output of Writer2)

```
<?xml version="1.0" ?>
<h:html xmlns:h="http://www.w3.org/1999/xhtml">
   <book xmlns="http://www.kku.ac.th/bookstore">
      <name isbn="123-456-7890">XML
      </name>
      <chapters>
         <chapter>Intro to XML
         </chapter>
         <chapter>XML Schema
         </chapter>
      </chapters>
   </book>
</h:html>
```

Cursor API vs. SAX

- The cursor API mirrors SAX in many ways
- Methods are available for directly accessing string and character information
- Integer indexes can be used to access attribute and namespace information
- Cursor API methods return XML information as string which minimizes object allocation requirements

Iterator API

- The StAX iterator API represents an XML document stream as a set of discrete event objects
- The base iterator interface is called XMLEvent
- The primary parser interface for reading iterator events is XMLEventReader
- The primary parser interface for writing iterator events is XMLEventWriter

XMLIterator

```
public interface XMLIterator {
   // Check if there are more events.
   boolean hasNext();

   // Get the next XMLEvent
   XMLEvent next();
}
```

XMLEventReader

```
public interface XMLEventReader extends
    XMLIterator {
    // Reads the content of a text-only element
    String getElementText();
    // Skip any insignificant space events until a
    // START_ELEMENT or END_ELEMENT is
    // reached.
    XMLEvent nextTag();
    // Check the next XMLEvent without reading it
    // from the stream.
    XMLEvent peek();
}
```

EventReader (1/5)

```
package staxprogramming;
import java.io.*;
import javax.xml.stream.*;
import javax.xml.stream.events.*;
import java.util.Iterator;
public class EventReader {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: java EventReader <xml
    file>");
            System.exit(1);
        }
```

EventReader (2/5)

```
// Create object in class XMLInputFactory
XMLInputFactory factory =
    XMLInputFactory.newInstance();

// Create parser object in class XMLEventReader
        XMLEventReader r =
    factory.createXMLEventReader(args[0],
                new FileInputStream(args[0]));
```

EventReader (3/5)

```
// Iterate until there is no more data to read
        while (r.hasNext()) {
            XMLEvent e = r.nextEvent();

            // if this part of data is characters section
            if (e.getEventType() == e.CHARACTERS) {
                Characters chars = e.asCharacters();
                System.out.print("Characters: " +
    chars.getData());
            }
```

EventReader (4/5)

```
// if this part of data is the start tag
        if (e.getEventType() ==
e.START_ELEMENT) {
            StartElement startE =
e.asStartElement();
            System.out.println("StartElement:" +
startE.getName());

// retrieve attributes
            Iterator it = startE.getAttributes();
```

EventReader (5/5)

```
        // Read each attribute then print its name
        // and its value
        while (it.hasNext()) {
                Attribute attr = (Attribute) it.next();
                System.out.println("Attribute: " + attr.getName()
    + " = "
                        + attr.getValue());
            }
        }
    }
}
```

EventReader: Sample Input

```
<?xml version="1.0" ?>
<p:nation
  xmlns:p="http://coeservice.en.kku.ac.t
  h" id="th">
   <p:name>Thailand</p:name>
   <p:location>Southeast
  Asia</p:location>
</p:nation>
```

EventReader: Sample Output

StartElement:{http://coeservice.en.kku.ac.th}nation

Attribute: id = th

Characters:

StartElement:{http://coeservice.en.kku.ac.th}name

Characters: ThailandCharacters:

StartElement:{http://coeservice.en.kku.ac.th}location

Characters: Southeast AsiaCharacters:

XMLEventWriter

```
public interface XMLEventWriter
extends XMLEventConsumer {
   void add(XMLEvent event);
   void setDeafultNamespace(
        java.lang.String uri);
   String getPrefix(String uri);
   void setPrefix(String prefix, String uri);
   …
}
```

EventWriter (1/4)

```
package staxprogramming;
import javax.xml.stream.*;
import javax.xml.stream.events.*;
import javax.xml.namespace.QName;
import java.util.*;
public class EventWriter {
    public static void main(String args[]) {
        try {
            XMLEventFactory eventFactory =
    XMLEventFactory.newInstance();
            XMLOutputFactory output =
    XMLOutputFactory.newInstance();
            XMLEventWriter xmlwriter =
                    output.createXMLEventWriter(System.out);
```

EventWriter (2/4)

```
xmlwriter.add(eventFactory.createStartDocument("UTF-8", "1.0"));
// create an attribute
Attribute att = eventFactory.createAttribute("id", "th");
ArrayList attArr = new ArrayList();
attArr.add(att);
// create namespace
Namespace namespace =
        eventFactory.createNamespace("p", "http://campus.en.kku.ac.th");
ArrayList nameArr = new ArrayList();
nameArr.add(namespace);
```

EventWriter (3/4)

```
// Declare qualified name
 QName qname = new
 QName("http://campus.en.kku.ac.th", "nation", "p");
// Create start tag with attributes
xmlwriter.add(eventFactory.createStartElement(
qname, attArr.iterator(), nameArr.iterator()));
 xmlwriter.add(eventFactory.createStartElement(
"p", "http://campus.en.kku.ac.th", "name"));
 // Create element content
 xmlwriter.add(
 eventFactory.createCharacters("Thailand"));
```

EventWriter (4/4)

```
            // Create end tag
            xmlwriter.add(eventFactory.createEndElement("p",
    "http://campus.en.kku.ac.th", "name"));
            xmlwriter.add(eventFactory.createEndElement(
                    qname, nameArr.iterator()));
            xmlwriter.add(eventFactory.createEndDocument());
            xmlwriter.flush();
            xmlwriter.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

EventWriter: Output

```
<?xml version="1.0"?><p:nation
  xmlns:p="http://campus.en.kku.ac.th"
  id="th"><p:name>Thailand</p:name>
  </p:nation>
```

Making Choices between Iterator API and Cursor API (1/2)

- In a memory-constrained environment, like J2ME, you can make smaller, more efficient code with the cursor API
- If performance is your highest priority, the cursor API is more efficient
- If you want to create XML processing pipelines, use the iterator API

Making Choices between Iterator API and Cursor API (2/2)

- If you want to modify the event stream, use the iterator API
- If you want your application to be able to handle pluggable processing of the event stream, use the iterator API
- In general, use the iterator API if you are not concerned about performance and memory because it is more flexible and extensible

References

- http://java.sun.com/webservices/docs/1.6/tutorial/doc/
- http://www.xml.com/pub/a/2003/09/17/stax.html
- http://www.oracle.com/technology/oramag/oracle/03-sep/o53devxml.html