StAX or SAX ? Push or Pull Parsing XML ?

Neeraj Bajaj

Sun Microsystems, inc.

http://www.sun.com

BoF 9778

Agenda

XML Parsing Models

StAX

- Cursor APIs : XMLStreamReader
- Event APIs : XMLEventReader

Cursor 2 Event

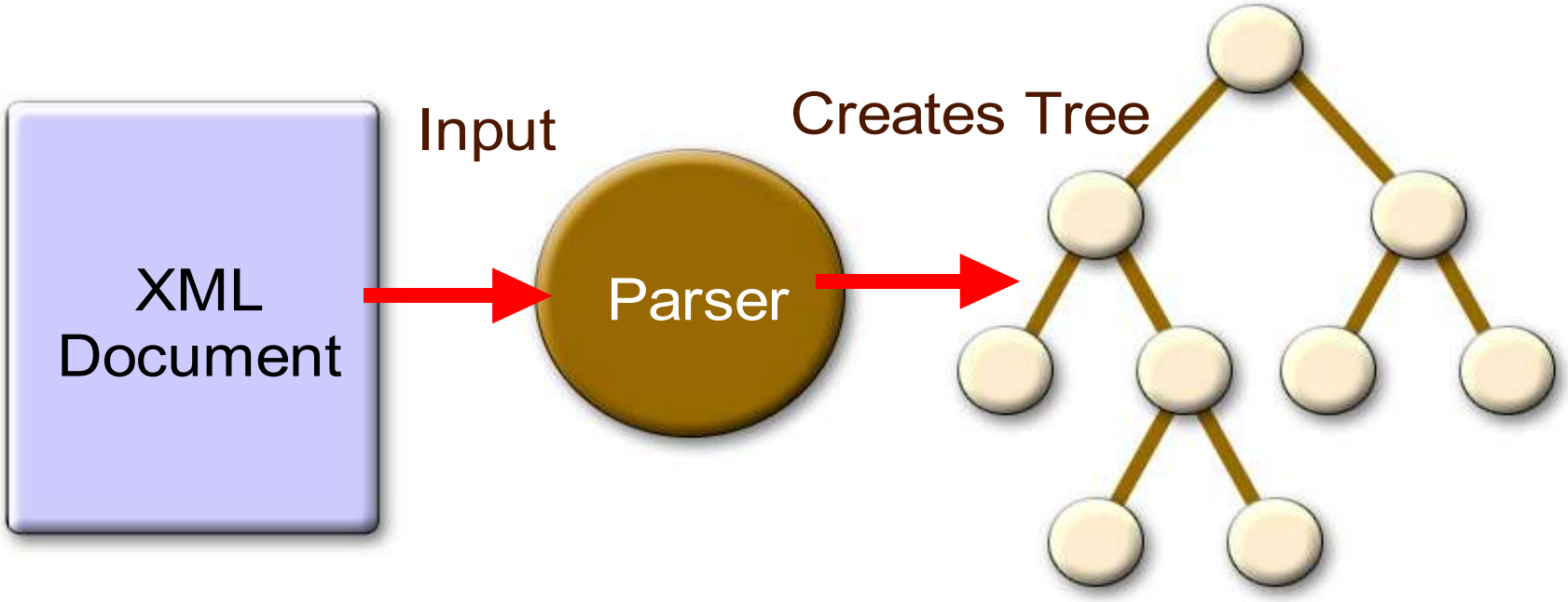
Stax Vs. SAX

Sun Java Streaming XML Parser (SJSXP)

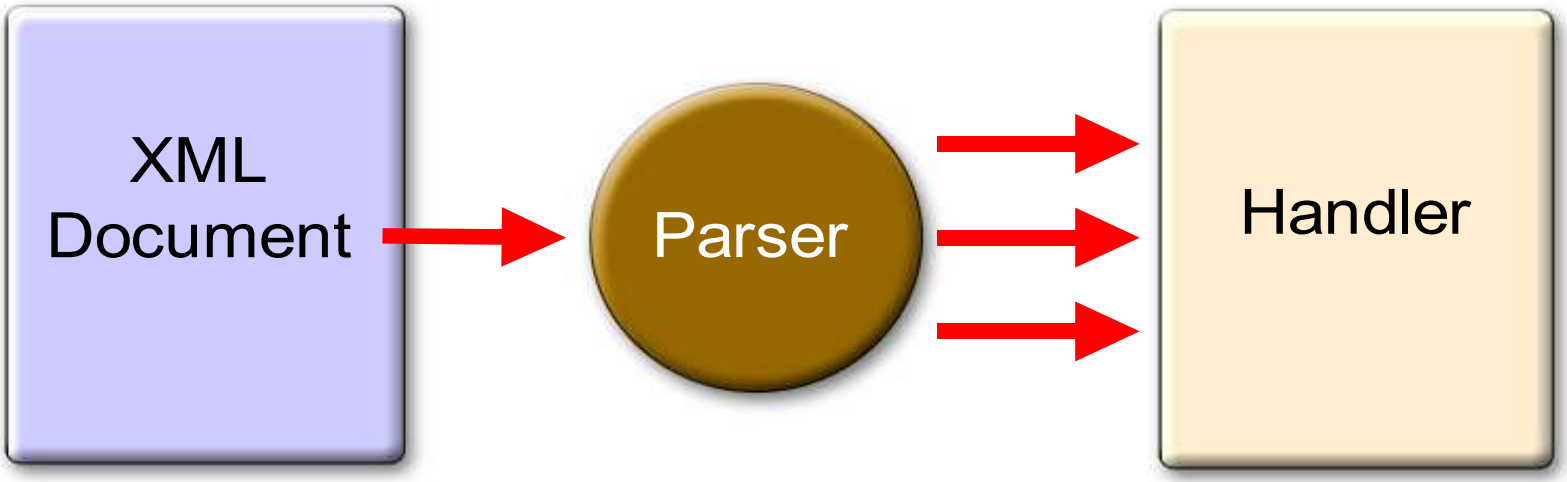
XML Parsing Models

- Object (DOM, JDOM etc.)
- Push (SAX)
- Pull (StAX) new

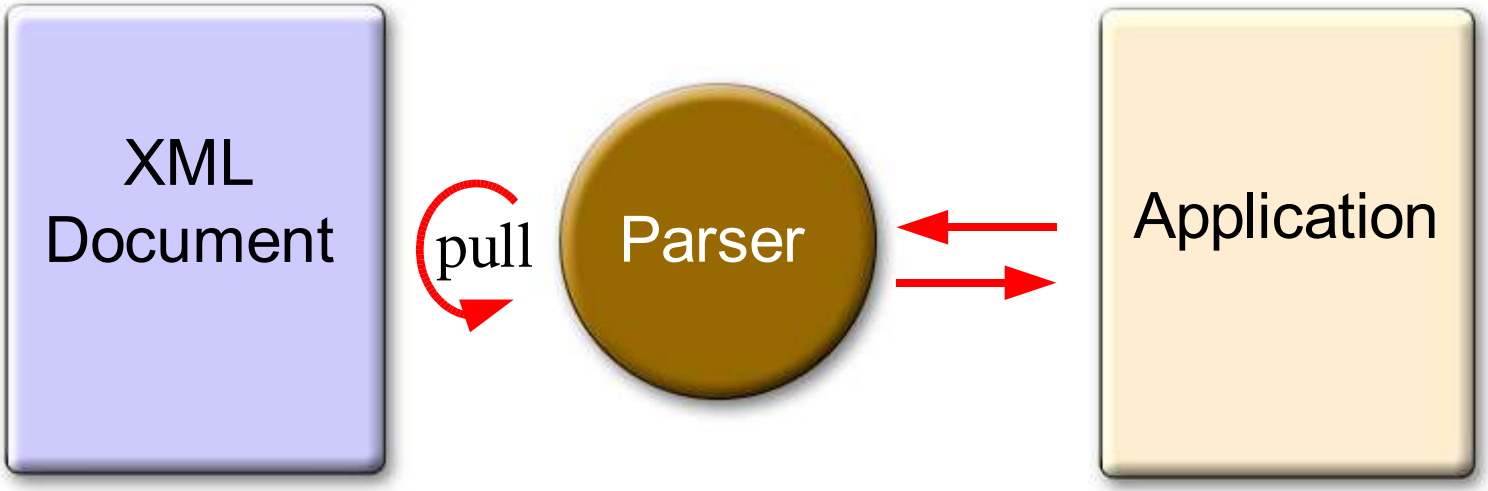
Tree Model, DOM



Push Model, SAX



Pull Model, StAX



StAX Pull Model

Two approach..

1. Cursor APIs: XMLStreamReader

2. Event APIs: XMLEventReader

StAX Events

1) Namespace
2) StartDocument
3) EndDocument
4) StartElement
5) EndElement
6) Attribute
7) EntityDeclaration
8) EntityReference
9) Notation
10) PI
11) DTD
12) Characters
13) Comment

Agenda

XML Parsing Models

StAX

- **Cursor APIs : XMLStreamReader**
- Event APIs : XMLEventReader

Cursor 2 Event

Stax Vs. SAX

Sun Java Streaming XML Parser (SJSXP)

XMLStreamReader

- Based on “iterator” pattern
  - hasNext()
  - Next()
- Most efficient way to read XML data
- Consumes less memory
- Represents a *cursor* moving forward

Creating XMLStreamReader

```
//create XMLInputFactory
XMLInputFactory factory = XMLInputFactory.newInstance();

//configure factory
factory.setXMLReporter(myXMLReporter);
factory.setXMLResolver(myXMLResolver);
factory.setProperty(..);

//create XMLStreamReader
XMLStreamReader reader =
factory.createXMLStreamReader(..);
```

Reading XML using XMLStreamReader

```
//create XMLStreamReader
XMLStreamReader reader =
factory.createXMLStreamReader(..);
int eventType = reader.getEventType();

//continue reading until there are more events
while(reader.hasNext()){
   //move to the next event in the XML stream
   eventType = reader.next();
   //pass the reader
   process(reader);
}
```

next() function

- drives the parser to read the next event on input stream.
- returns the integer code corresponding to the current parse event.
- After next() control is back with application, has the option to
  - Stop parsing
  - Call next() to go to next event
  - Read the information associated with the event

Reading XML decl.

```
<?xml version="1.0" encoding="UTF-8"?>
<Books xmlns="foo">10
   <Book>
      <Name>Stax</Name>
      <Author>ABC</Author>
   </Book>
</Books>
```

reader.getVersion()
reader.getEncoding()
reader.standAloneSet()

Navigation

```
<?xml version="1.0" encoding="UTF-8"?>
<Books xmlns="foo">10
   <Book>
      <Name>Stax</Name>
      <Author>ABC</Author>
   </Book>
</Books>
```

reader.next()*

* No new line character after xml declaration

Reading Element /Attributes

```
<?xml version="1.0" ...>
<Books xmlns="foo">10

  <Book>
    <Name> Stax </Name>
    <Author> ABC</Author>
  </Book>
</Books>
```

```
reader.next() == ELEMENT

reader.getName():QName
reader.getLocalName():String
reader.getPrefix():String
reader.isAttributeSpecified()

reader.getAttributeName(index):Q
reader.getAttributePrefix(index)
........
reader.getNamespaceURI()
reader.getNamespaceContext()
........
```

Navigation

```
<?xml version="1.0" encoding="UTF-8"?>
<Books xmlns="foo">10
   <Book>

      <Name>Stax</Name>
      <Author>ABC</Author>
   </Book>
</Books>
```

reader.next()

Reading Characters

```
<?xml version="1.0" ...>
<Books xmlns="foo">10

  <Book>
    <Name> Stax </Name>
    <Author> ABC</Author>
  </Book>
</Books>
```

reader.next() == CHARACTERS

reader.getText():String

reader.getTextCharacters():Char[]
reader.getTextStart():int
reader.getTextLength():int

Reading characters

- Don't assume next() has read all character data !
- Text data may be split into several calls
- Set “isCoalesce” property to true to receive all the adjacent character data as one event.

XMLStreamReader

- Behavior is function of its state.
- Need to understand which functions are valid for a particular state (or event)
- Certain functions are valid only for a particular state. For example:
  - illegal to call getPI() when next() returns 'Element' event

Agenda

XML Parsing Models

StAX

- Cursor APIs : XMLStreamReader
- **Event APIs : XMLEventReader**

Cursor 2 Event

Stax Vs. SAX

Sun Java Streaming XML Parser (SJSXP)

XMLEventReader

- Easy to use
- Flexible
- Easy pipelining
- Data returned as immutable XMLEvents

Creating XMLEventReader

```
//create XMLInputFactory
XMLInputFactory factory = XMLInputFactory.newInstance();

//configure factory
factory.setXMLReporter(myXMLReporter);
factory.setXMLResolver(myXMLResolver);
factory.setProperty(..);

//create XMLEventReader
XMLEventReader reader =
factory.createXMLEventReader(..);
```

Reading XML using XMLEventReader

```
//create XMLEventReader
XMLEventReader reader =
factory.createXMLEventReader(..);

//continue reading until there are more events
while(reader.hasNext()){
   //move to the next event in the XML stream
   XMLEvent event = reader.next();
   if(event.isStartElement()){
      StartElement se = event.asStartElement();
      QName  name = se.getName();
      Iterator attributes = se.getAttributes();
}
```

Agenda

XML Parsing Models

StAX

- Cursor APIs : XMLStreamReader
- Event APIs : XMLEventReader

Cursor 2 Event

Stax Vs. SAX

Cursor 2 Event

- You can switch from Cursor to Event while reading same XML document.
- Two ways
  - Create XMLEventReader wrapping XMLStreamReader
  - Allocate event using XMLEventAllocator

Create XMLEventReader Wrapping XMLStreamReader

```
XMLStreamReader sr = //get XMLStreamReader

if(sr.isStartElement() &&
    sr.getLocalName().equals("Book")){

//create XMLEventReader wrapping XMLStreamReader
XMLEventReader reader =
factory.createXMLEventReader(sr);

while(reader.hasNext()){
   //move to the next event in the XML stream
   XMLEvent event = reader.next();
}
}
```

Using XMLEventAllocator

```
XMLStreamReader sr = //get XMLStreamReader
XMLEventAllocator allocator =
          new MyXMLEventAllocator();

if(sr.isStartElement() && sr.getLocalName().
equals("Book")){

    StartElement se =
    allocator.allocate(sr).asStartElement();

}
```

DEMO

Cursor 2 Event

Agenda

XML Parsing Models

StAX

- Cursor APIs : XMLStreamReader
- Event APIs : XMLEventReader

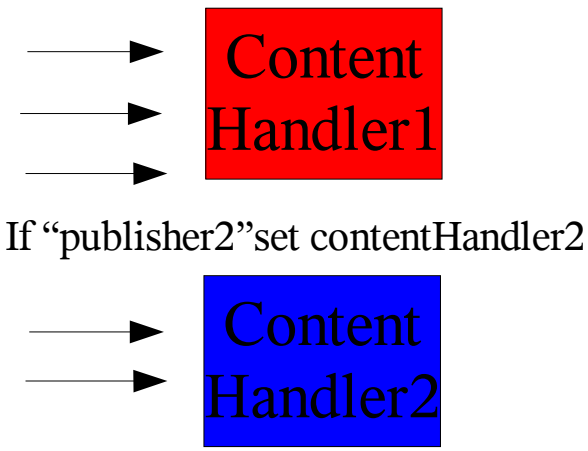
Cursor 2 Event

Stax Vs. SAX

Sun Java Streaming XML Parser (SJSXP)

Reading with SAX

```
<Book cover="hard" xmlns="publisher1">
   <Name>Stax</Name>
   <Author>ABC</Author>
</Book>
<Book cover="soft" xmlns="publisher2">
   <Name>SAX</Name>
   <Author>XYZ</Author>
</Book>
```



Problem

- ContentHandler2 will not have details about 2nd Book element.
- Application has to do special handling to localize the processing of Book element in 'publisher2' domain.
- Less control over reading data.

Reading with StAX

```
<Book cover="hard" xmlns="publisher1">
   <Name>Stax</Name>
   <Author>ABC</Author>
</Book>
<Book cover="soft" xmlns="publisher2">
   <Name>SAX</Name>
   <Author>XYZ</Author>
</Book>
```

```
p1(XMLStreamReader reader){
  If "publisher2"{
    call p2(reader)
}
```

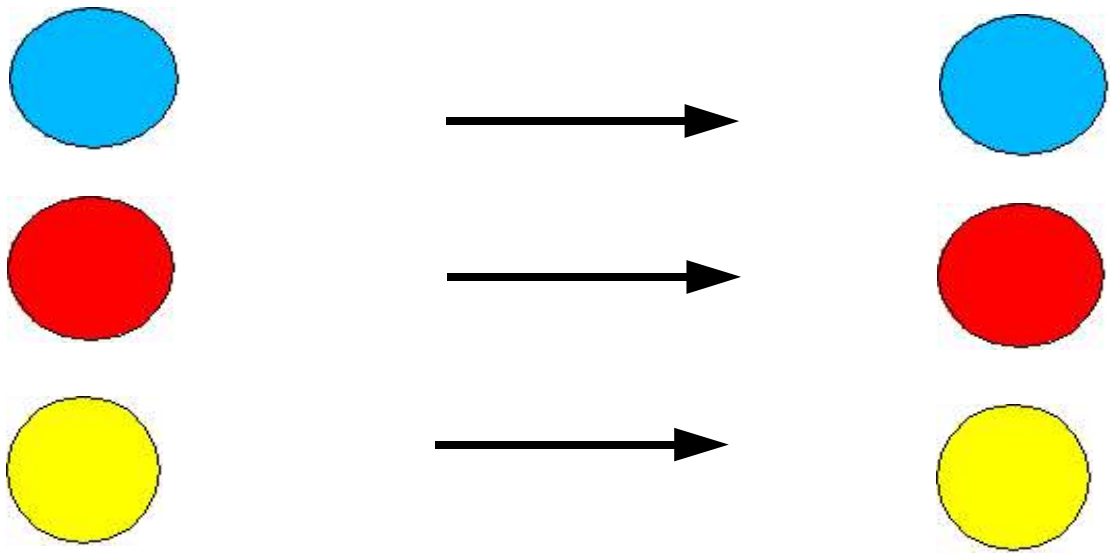
```
p2(XMLStreamReader reader){

}
```

Advantage

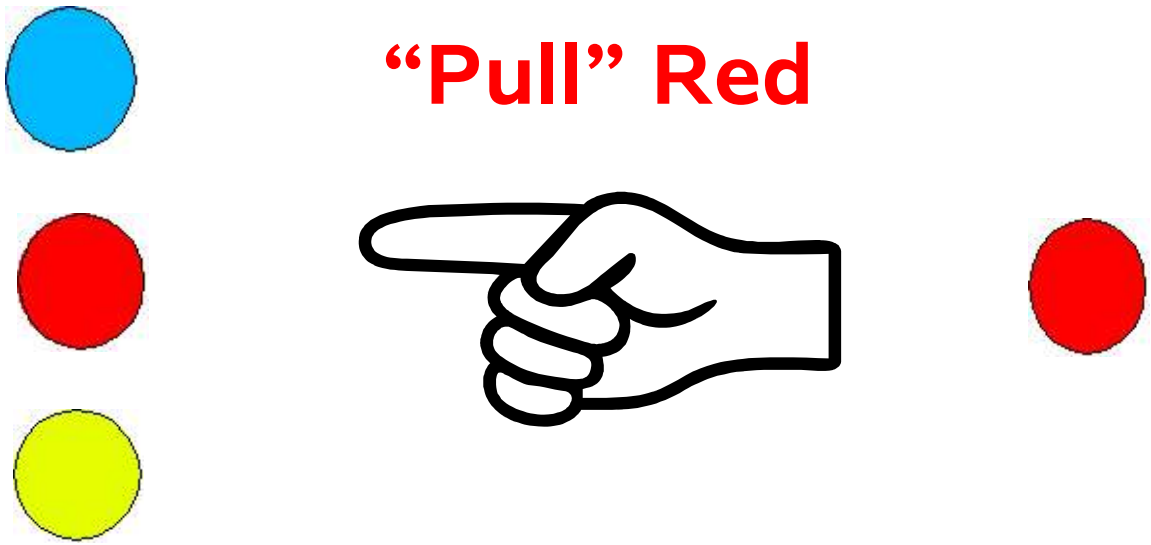
- Processing of Book elements can be done separately.
- Control over reading the data.
- Modular code.
- Easy to pass StreamReader to different parts of code.

SAX



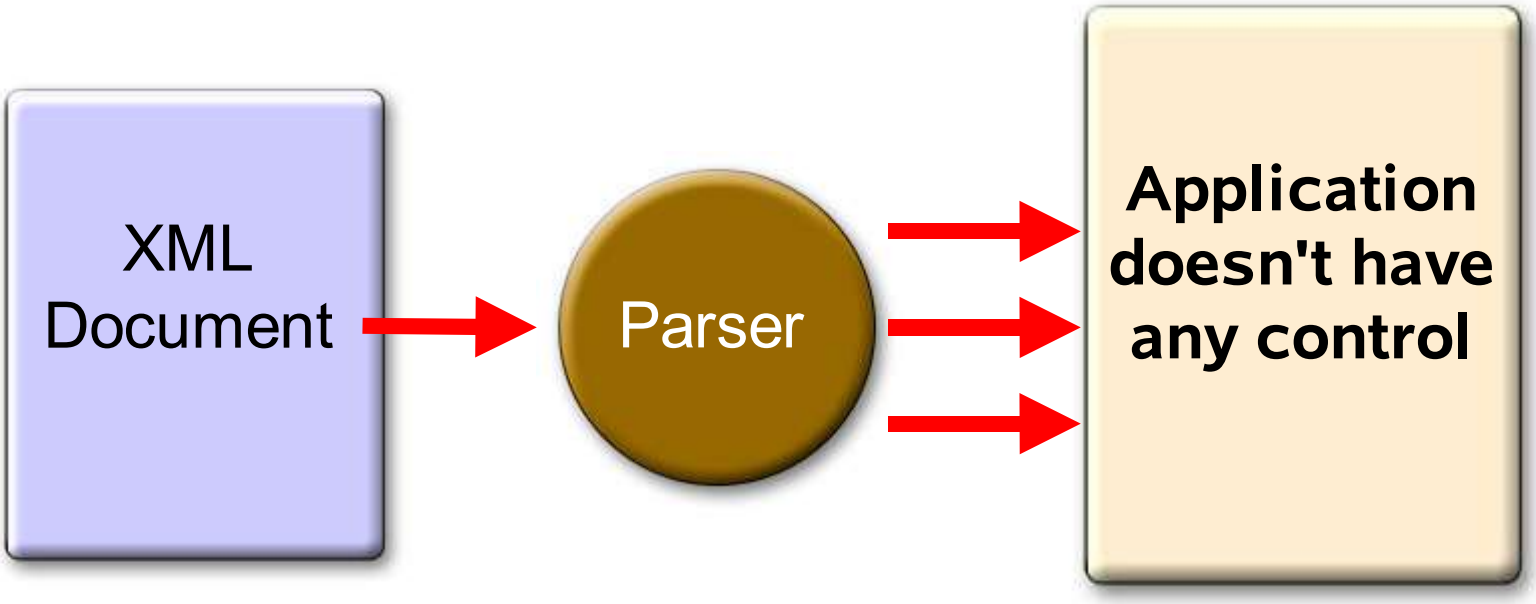
All the data is pushed

StAX



Allows selective reading

SAX Parsing



- It's like uncontrolled reaction of throwing events until all the data is consumed.
- Throw exception to “stop” parsing before endDocument callback

StAX Vs. SAX

- Application controls when and how much data to read
- easy to write code for parsing complex XML documents
- "Iterator" based

- Application has no control over parsing
- State machine becomes complex with complexity of XML doc.
- "Observer" based

StAX Vs. SAX

- Allows “selective” reading of data
- Allows computing of data lazily
- Easy and efficient to build 'push' layer on top of 'pull' layer.

- All data is pushed to application
- Values need to be calculated before pushing data
- Difficult to write efficient 'pull' layer on top of 'push' layer

StAX Vs. SAX

- Easy to read multiple docs. at a time with just a single thread.
- Easy to skip non-relevant parts of XML document.
- Very easy to stop, just do nothing.
- Read and Write APIs

- Very difficult to read multiple docs. at a time with single thread.
- Doesn't allow skipping parts of XML document.
- Throw exception to stop parsing.
- Need separate APIs to write

StAX Vs SAX

- ErrorHandler can't be changed during parse
- EntityHandler can't be changed during parse
- Modifying stream is not easy

- ErrorHandler can be changed during parse
- EntityHandler can be changed during parse
- Modifying stream is easy

Agenda

XML Parsing Models

StAX

- Cursor APIs : XMLStreamReader
- Event APIs : XMLEventReader

Cursor 2 Event

Stax Vs. SAX

Sun Java Streaming XML Parser (SJSXP)

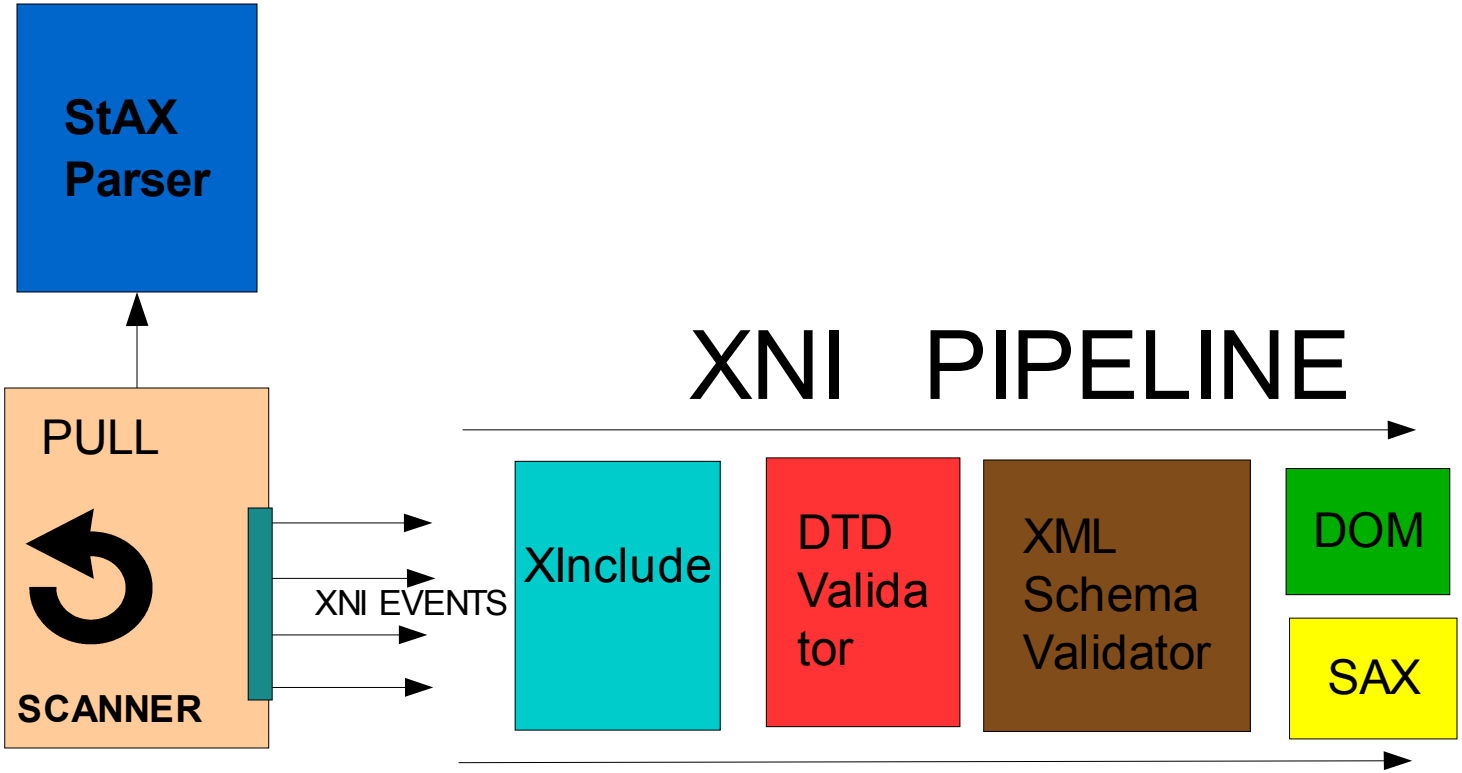
Sun Java™ Streaming XML Parser (SJSXP)

- Sun's implementation of StAX, JSR 173.
- Highly performant.
- Built on Xerces2 code base
  - XML scanner has been "re-designed" to behave in pull fashion
  - Lot of performance improvements
- Fully compliant to W3C XML 1.0, Namespace 1.0
- Non Validating

Sun Java™ Streaming XML Parser (SJSXP)

- Binaries available on https://sjsxp.dev.java.net
- Stax impl. merged with JAXP impl. (Xerces) in JAXP 1.4
  - Development happens at http://jaxp.dev.java.net
  - Sources available at http://jaxp-sources.dev.java.net
- Will be part of Java™ platform (JDK 6.0)
- Bundled with Java™ Web Services Developer Pack 1.5, 1.6

StAX impl. & JAXP impl. merge



Sun Java™ Streaming XML Parser Road Map

- Add DTD validation support
- Add support for XML 1.1
- Add XInclude support

References

- JSR 173, http://www.jcp.org/en/jsr/detail?id=173
- JAXP 1.4 (with StAX), https://jaxp.dev.java.net
- Source code at https://jaxp-sources.dev.java.net
- Java Web Services Developer Pack 1.5/1.6
  http://java.sun.com/webservices/jwsdp/index.jsp
- Sun Stax implementation (SJSXP)
  https://sjsxp.dev.java.net
- StAX RI, http://dev2dev.bea.com/technologies/stax/index.jsp

Thank You

StAX or SAX ? Push or Pull Parsing XML ?

Neeraj Bajaj

Sun Microsystems, inc.

http://www.sun.com

BoF 9778