PB138 — XML Processing in general programming languages (XML APIs)

API for XML Processing (to repeat)

- APIs offer simple standardized XML access.
- APIs connect application to the parser and applications together.
- APIs allow XML processing without knowledge of physical document structure (entities).
- APIs optimize XML processing.

XML APIs Fundamental Types

***Tree-based* API**

tree representation in constructed and processed

***Event-based* API**

events are produced and handled

***Pull* API**

events are pulled off the document

Tree-based API

- They map an XML document to a memory-based tree structure.
- It allows to traverse the entire DOM tree.
- Best-known is *Document Object Model* (DOM) from W3C, http://www.w3.org/DOM)

Programming Language Specific Models

- Java JDOM - http://jdom.org
- Java dom4j - http://dom4j.github.io
- Java XOM - http://www.xom.nu
- Python 4Suite - http://4suite.org
- PHP SimpleXML - http://www.php.net/simplexml

Document Object Model (DOM)

- Basic interface to process and access the tree representation of XML data

- Three versions of DOM: DOM Level 1, 2, 3
- DOM - does not depend on the XML parsing.
- Described using IDL + API descriptions for particular programming languages (C++, Java, etc.)

DOM Levels

- DOM Level 1 — provides low-level set of fundamental interfaces as well as extended interfaces those can represent any structured document (Document, Element, DocumentFragment, etc, see DOM Level 1 Specification).
- DOM Level 2 — defines platform- and language neutral interface that allow to dynamically access and update the content and structure of documents (see DOM Level 2 Specification)
- DOM Level 3 — enhances DOM Level 2 by completing mapping between DOM and XML Information Set, by including support for XML Base, allows to attach user information to DOM Nodes, etc (see DOM Level 3 Specification)

HTML Documents Speci□c DOM

- The HTML Core DOM is more less consolidated with the XML DOM
- Designated to CSS
- Used for dynamic HTML programming (scripting using VB Script, JavaScript, etc)
- Contains the browser environment (windows, history, etc) besides the document model itself.

DOM references

- JAXP Tutorial, part dedicated to the DOM Part III: XML and the Document Object Model (DOM) (http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/dom/index.html)
- Portal dedicated to the DOM http://www.oasis-open.org/cover/dom.html
- DOM 1 Interface visual overview http://www.xml.com/pub/a/1999/07/dom/index.html
- Tutorial ”Understanding DOM (Level 2)” available at https://www.ibm.com/developerworks/xml/

Using DOM in Java

- Native DOM support in the new Java versions (JDK and JRE) - no need of additional library.
- Applications need to import needed symbols (interfaces, classes, etc.) mostly from package `org.w3c.dom`.

What we frequently need

Most often used interfaces are:

- `Element` corresponds to the element in a logical document structure. It allows us to access name of the element, names of attributes, child nodes (including textual ones). Useful methods:
- `Node getParentNode()` - returns the parent node
- `String getTextContent()` - returns textual content of the element.
- `NodeList getElementsByTagName(String name)` - returns the list of ancestors (child nodes and their ancestors) with the given name.

What we frequently need (2)

- `Node` super interface of `Element`, corresponds to the general node in a logical document structure, may contain element, textual node, comment, etc.
- `NodeList` a list of nodes (a result of calling `getElementsByTagName` for example). It o□ers the following methods for its processing:
- `int getLength()` - returns the number of nodes in a list
- `Node item(int index)` - returns the node at position index
- `Document` corresponds to the document node (its a parent of a root element)

Example 1 - creating DOM tree from file

```
public class Task1 {
  public Task1(URL url) throws SAXException,
    ParserConfigurationException, IOException {
    // We create new instance of factory class
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    // We get new instance of DocumentBuilder using the factory class.
    DocumentBuilder builder = factory.newDocumentBuilder();
    // We utilize the DocumentBuilder to process an XML document
    // and we get document model in form of W3C DOM
    Document doc = builder.parse(url.toString());
  }
}
```

Example 2 - DOM tree modification

```
public class Task1 {
  private Document doc;
  public void adjustSalary(double minimum) {
    NodeList salaries = doc.getElementsByTagName("salary");
    for (int i = 0; i < salaries.getLength(); i++) {
      Element salaryElement = (Element) salaries.item(i);
      double salary = Double.parseDouble(
          salaryElement.getTextContent());
```

```
      if (salary < minimum) {
        salaryElement.setTextContent(String.valueOf(minimum));
      }
    }
  }
}
```

Example 3 - storing a DOM tree into an XML file

Example of the method storing a DOM tree into a file (see Homework 1). The procedure utilizes a transformation we do not know yet. Let use it as a black box.

```
public class Task1 {
    private Document doc;
    public void serializetoXML(File output) throws IOException,
    TransformerConfigurationException {
      TransformerFactory factory
        = TransformerFactory.newInstance();
      Transformer transformer
        = factory.newTransformer();
      DOMSource source = new DOMSource(doc);
      StreamResult result = new StreamResult(output);
      transformer.transform(source, result);
    }
}
```

Event-based API

- Generates Sequence of Events while parsing the Document.
- Technical implementation: using *callback methods* [1]
- Application implements *handlers* (which process generated events).
- Works on lower-level than tree-based.
- Application should do further processing.
- It saves memory - does not itself create any persistent objects.

Event Examples

- `start document`, `end document`
- `start element` - contains the attributes as well, `end element`.
- `processing instruction`

- `comment`
- `entity reference`
- Best-known event-based API: SAX http://www.saxproject.org

SAX - Document Analysis Example

```
<?xml version="1.0"?>
<doc>
<para>Hello, world!</para>
<!-- that□s all folks -->
<hr/>
</doc>
```

SAX - Document Analysis Example

It generates following events:

```
start document start element: doc
list of attributes: empty
start element: para
list of attributes: empty
characters: Hello, world!
```

SAX - Document Analysis Example (2)

```
end element: para
comment: that□s all folks
start element: hr
end element: hr
end element: doc
end document
```

When to use event-based API?

- Easier to parser programmer, more difficult to application programmer.
- No complete document available to application programmer.
- Programmers must keep the state of analysis themself.
- Suitable for tasks, that can be solved without the need of entire document.
- The fastest possible processing usually.

- Difficulties while writing applications can be solved using extensions like *Streaming Transformations for XML (STX)*, http://stx.sourceforge.net

Optional SAX Parser Features

- The SAX parser behavior can be controlled using so called features a properties.
- For optional SAX parser's features see http://www.saxproject.org/?selected=get-set
- For more details on properties and features see Use properties and features in SAX parsers (IBM DeveloperWorks/XML).

SAX filters

- The SAX filters (implementation of `org.xml.sax.XMLFilter` interface) can be programmed using the SAX API.
- Such a class instance accepts input events, process them and sends them to the output.
- For more information on event filtering see *Change the events output by a SAX stream* http://www.ibm.com/developerworks/xml/library/x-tipsaxfilter/ (IBM DeveloperWorks/XML) for example.

Additional SAX References

- Primary source: http://www.saxproject.org
- SAX Tutorial on JAXP: http://java.sun.com/webservices/reference/tutorials/jaxp/html/sax.html

Pull-based APIs

- Application does not process incoming events, but it pulls data from the processed file.
- Can be used when programmer knows the structure of an input data and she can pull them off the file.
- As opposite to event-based API.
- Very comfortable to an application programmer, but implementations are usually slower the push event-based APIs.

Java Pull-based APIs

- Java offers the XML-PULL parser API - see *Common API for XML Pull Parsing* http://www.xmlpull.org/ and also
- newly develop API - *Streaming API for XML (StAX)* http://www.jcp.org/en/jsr/detail?id=173 developed like a product of JCP (Java Community Process).

Streaming API for XML (StAX)

- The API may become the part of the *Java API for XML Processing* (JAXP) in the future.
- It offers two ways to pull-based processing:
- pulling the events using iterator - more comfortable
- low-level access using so called cursor - it is faster.

StAX - an Iterator Example

- from Oracle Java Tutorials http://docs.oracle.com/javase/tutorial/jaxp/stax/example.html
- In this example, the client application pulls the next event in the XML stream by calling the next method on the parser.

StAX - source XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<BookCatalogue xmlns="http://www.publishing.org">
  <Book>
    <Title>Yogasana Vijnana: the Science of Yoga</Title>
    <author>Dhirendra Brahmachari</Author>
    <Date>1966</Date>
    <ISBN>81-40-34319-4</ISBN>
    <Publisher>Dhirendra Yoga Publications</Publisher>
    <Cost currency="INR">11.50</Cost>
  </Book>
```

StAX - source XML document

```
  <Book>
    <Title>The First and Last Freedom</Title>
    <Author>J. Krishnamurti</Author>
    <Date>1954</Date>
    <ISBN>0-06-064831-7</ISBN>
    <Publisher>Harper &amp;amp; Row</Publisher>
    <Cost currency="USD">2.95</Cost>
  </Book>
</BookCatalogue>
```

StAX - Java code

```
try {
```

```
    for (int i = 0 ; i < count ; i++) {
       // pass the file name.. all relative entity
       // references will be resolved against this
       // as base URI.
       XMLStreamReader xmlr = xmlif.createXMLStreamReader(filename,
          new FileInputStream(filename));
          // when XMLStreamReader is created,
          // it is positioned at START_DOCUMENT event.
          int eventType = xmlr.getEventType();
          printEventType(eventType);
          printStartDocument(xmlr);
          // check if there are more events
          // in the input stream
```

StAX - Java code

```
          while(xmlr.hasNext()) {
             eventType = xmlr.next();
             printEventType(eventType);
             // these functions print the information
             // about the particular event by calling
             // the relevant function
             printStartElement(xmlr);
             printEndElement(xmlr);
             printText(xmlr);
             printPIData(xmlr);
             printComment(xmlr);
          }
       }
    }
```

Tree and event-based access combinations

- Events → tree
- Tree → events

Events → tree

- Allow us either to skip or to filter out the ”uninteresting” document part using the event monitoring and then
- create memory-based tree from the ”interesting” part of a document only and that part process.

Tree → events

- We create an entire document tree (and process it) and
- we go through the tree than and we generate events like while reading the XML file.
- It allows us easy integration of both processing types in a single application.

Virtual object models

- Document DOM model is not memory places, but is created on-demand while accessing particular nodes.
- combines event-based and tree-based processing advantages (speed and comfort)
- There is an implementation: the *Sablotron* processor, http://www.xml.com/pub/a/2002/03/13/sablotron.html

Alternative tree-based models

- XML Object Model (XOM)
- DOM4J

XML Object Model (XOM)

- XOM (XML Object Model) created as an one man project (author Elliote Rusty Harold).
- It is an interface that strictly respect XML data logical model.
- For motivation and specification see the XOM home page (http://www.xom.nu).
- You can get there the open-sourceXOM implementation and
- the API documentation, too.

DOM4J - practically usable tree-based model

- comfortable, fast and memory efficient tree-oriented interface
- designed and optimized for Java
- available as open-source at http://dom4j.github.io
- perfect ”cookbook” available

[1] The Hollywood Principle: Do not call us, we will call you!