Web Technologies

XML data processing (II)

SAX (Simple API for XML)
XML document simplified processing
“Before asking new questions, think if you really want to know the response to them.”

Gene Wolfe
There are alternative ways to process the XML documents?
sax: intro

Goal:
access to XML/HTML documents without creating the tree of node-objects
Goal:
access to XML/HTML documents without creating the tree of node-objects

- the document is not entirely stored into the memory before processing
Providing a sequential (linear) event-oriented XML processing

initiator: David Megginson

www.megginson.com/downloads/SAX/
sax: features

Providing a sequential (linear) event-oriented XML processing

“SAX is a streaming interface – applications receive information from XML documents in a continuous stream with no backtracking or navigation allowed”
sax: features

Independent effort – aside from the W3C – to standardize the event-driven XML processing

www.saxproject.org
sax: features

Largely accepted as an industrial standard

**SAX 1.0** (1998)

reference implementation in Java language

`org.xml.sax`
sax: features

Largely accepted as an industrial standard

SAX 2.0 (2004)

supporting namespaces,
various configurations + extensions
**sax: processing**

For each type of XML construct,
– start tag, end tag, data (content),
processing instruction, comment,... – an event will be emitted and handled by a function/method
sax: processing

Handler functions/methods will be specified by the programmer, for each type of XML construct
sax: processing

The program *consumes* and *handles* events produced by the *SAX* processor
sax: processing

Minimally, we should define the following functions/methods:

- handle_start_tag (processor, tag, attributes)
- handle_end_tag (processor, tag)
- handle_character_data (processor, data)

contains the list of attributes attached to the start tag
sax: processing

For each event regarding the occurrence of start tag, end tag, and content-data, one of the handler functions will be attached:

set_element_handler
  (handle_start_tag, handle_end_tag)
set_character_data_handler
  (handle_character_data)
**sax: processing**

Client application

SAX processor

**Handler instantiation**

Notifying

**start processing:** `parse()`

**event occurred**

**start tag**

**calling handler**

**event occurred**

**end tag**

**calling handler**

**sending SAX events**

**Pro- cessing**

```xml
<projects>
<project class="S">
...
</project>
</projects>
```
sax: processing

Reference implementation (Java): org.xml.sax

details at
www.saxproject.org/apidoc/org/xml/sax/package-summary.html
Interfaces that can be implemented by our application:

**ContentHandler**

resolves event notifications regarding the types of XML constructs: start document, start tag, textual data, end tag, end document, etc.
sax: processing

Interfaces that can be implemented by our application:

Attributes

defines the list of attributes specified in the start tag
sax: processing

Interfaces that can be implemented by our application:

XMLReader

specifies how XML data is read by using event callbacks
Interfaces that can be implemented by our application:

**ErrorHandler**

specifies how (fatal) errors and warnings will be handled

exceptions like SAXException and SAXParseException could be thrown
sax: processing

Provided SAX class:

InputSource

encapsulates information about an input source providing XML data (e.g., a character stream)
Example: **XMLReader** interface (Apache Xerces)

// XML processing via events (reading data)
public interface XMLReader {
    // offers info about the document
    public ContentHandler getContentHandler ();
    public DTDHandler getDTDHandler ();
    public EntityResolver getEntityResolver ();
    public ErrorHandler getErrorHandler ();
    // setting various functionalities
    public void setContentHandler (ContentHandler contentHandler);
    public void setDTDHandler (DTDHandler dtdHandler);
    public void setEntityResolver (EntityResolver resolver);
    public void setErrorHandler (ErrorHandler errHandler);
    // actual processing
    public void parse (InputSource in)
        throws java.io.IOException, SAXException;
    public void parse (String uri)
        throws java.io.IOException, SAXException;
}
Example: **ContentHandler** interface (Apache Xerces)

```java
// used to process XML constructs
public interface ContentHandler {
    public void setDocumentLocator (Locator locator);
    public void startDocument () throws SAXException;
    public void endDocument () throws SAXException;

    // events
    public void startElement (String uri, String localName, String qName,
        Attributes attributes) throws SAXException;
    public void endElement (String uri, String localName, String qName)
        throws SAXException;
    public void characters (char buf[], int offset, int length)
        throws SAXException;

    // additional info
    public void ignorableWhitespace (char buf[], int offset, int length)
        throws SAXException;
    public void startPrefixMapping (String prefix, String uri)
        throws SAXException;
    public void endPrefixMapping (String prefix)
        throws SAXException;
}
```
Example: **Attributes** interface (Apache Xerces)

```java
// specifies the attributes attached to an XML element
public interface Attributes {
    public int getLength ();
    public String getType (int index);
    public String getValue (int index);
    // access to info regarding the attribute name
    public String getQName (int index);
    public String getLocalName (int index);
    public String getURI (int index);
    // access via XML namespaces
    public int getIndex (String uri, String localName);
    public String getType (String uri, String localName);
    public String getValue (String uri, String localName);
    // access by using qualified names (prefix:name)
    public int getIndex (String qName);
    public String getType (String qName);
    public String getValue (String qName);
}
```
**sax: implementations**

**libxml** – open source API: C, C++, Haskell, Scala,...

**MSSAX** – SAX processing in C, C++, JavaScript; included in MSXML SDK (Software Development Kit)

**NSXMLParser** – Objective-C implementation (Apple)

**org.xml.sax** – reference API for Java

**REXML** – an XML processor for Ruby

**QSAX** – part of Qt development environment (C++)
sax implementations

sax-js – Node.js module; others: nodejsmodules.org/tags/sax

Xerces SAX API – XML platform for C++ and Java: http://xml.apache.org/

erlsom, xmerl_eventp – Erlang modules

xml – a Go package: https://golang.org/pkg/encoding/xml/

XML::Parser – a Perl module based on Expat processor


xml.sax – Python: docs.python.org/3/library/xml.sax.html
sax: demo
Cutting corners to meet arbitrary management deadlines

Essential

Copying and Pasting from Stack Overflow

The Practical Developer
@ThePracticalDev
sax vs. dom

When SAX could be used?

the necessity to process very large documents

necessity to cancel the XML processing (SAX processor could be stopped anytime)

extracting tiny amounts of information
sax vs. dom

When SAX could be used?

creating a new structure of an XML document

using in the context of limited computing resources
(low memory, narrow bandwidth,...)

example for Android:
developer.android.com/reference/javax/xml/parsers/SAXParser.html
demo source-code for iOS – SeismicXML:
developer.apple.com/library/ios/samplecode/SeismicXML/
sax vs. dom

When DOM could be used?

- random access to the data of an XML document
- complex processing
- complex data filtering via XPath
- performing XSL transformations
- XML data validation with DTD, XML Schema, etc.
sax vs. dom

When DOM could be used?

the necessity to modify and/or save XML documents in the context of processing XML/HTML data on the Web browser (including support for asynchronous data transfer via Ajax)
sax vs. dom

DOM requires loading into the memory the entire XML document in order to be processed as a tree
sax vs. dom

SAX gets small fragments from a document, performing a linear processing (event stream)
sax vs. dom

SAX could be used to generate DOM trees

Conversely, DOM trees could be traversed in order to emit SAX events

examples:
* `dom-js` module (Node.js), `lxml` library (Python)
sax vs. dom

In the case of complicated XML structures, the SAX processing could be improper.

SAX processing ignores the context of a certain element occurrence.
sax vs. dom: example

Consider the XML document structure, specified by the following DTD:

```xml
<!DOCTYPE catalog [ 
  <!ELEMENT catalog (categ+)>
  <!ELEMENT categ (#PCDATA | categ)*> ]>
```

What processing method will be suitable for a huge number of `<categ>` elements?
sax vs. dom

Certain SAX implementations provide support for XML validation and transformation.

commonly, both DOM and SAX are used.
There are any other XML processing ways?
XML document processing

alternatives:
XPP – XML Pull Parsing
XML data binding
simplified processing
alternatives: xml pull parsing

Styles of event-driven XML processing:

push versus pull
alternatives: xml pull parsing

Styles of event-driven XML processing:

**push** = XML processor reads XML data and notifies the application on occurred events (parsing events) – SAX
alternatives: xml pull parsing

Styles of event-driven XML processing:

**push** = XML processor reads XML data and notifies the application on occurred events (parsing events) – **SAX**

the programul can not request events

the events appear as they are sent by the processor (push)
alternatives: xml pull parsing

Styles of event-driven XML processing:

pull = the application controls the processing and can request (pull) the processor to send the following event

XPP – XML Pull Parsing

www.xmlpull.org
alternatives: xml pull parsing

Styles of event-driven XML processing:

**pull** = the application controls the processing and can request (pull) the processor to send the following event

XPP – XML Pull Parsing

the program’s source-code structure reflects the structure of the processed XML document
## alternatives: xml pull parsing

<table>
<thead>
<tr>
<th>Push APIs</th>
<th>Pull APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-only processing</td>
<td>Inheriting the advantages of push APIs</td>
</tr>
<tr>
<td>Fast processing via streams</td>
<td>Events are consumed according to the needs</td>
</tr>
<tr>
<td>Source-code could be difficult to be understood</td>
<td>Programs have a clearer structure</td>
</tr>
</tbody>
</table>
xml pull parsing – implementations

StAX – Streaming API for XML (Java) – JSR 173

examples of implementations:
Javolution – focused on performance: http://javolution.org/

Oracle StAX – included in XDK (XML Developer’s Kit)
https://docs.oracle.com/javase/tutorial/jaxp/stax/

Woodstox – https://github.com/FasterXML/woodstox
xml pull parsing – implementations

irrXML – initially, part of Irrlicht 3D Engine (C++)
pull – Scala package for XPP processing
QXmlStreamReader, QXmlStreamWriter from Qt (C++)
saxpath – Node.js modules able to evaluate XPath expressions for a SAX event stream
xml.dom.pulldom – a Python solution
XmlPullParser – Java interface for Android
A classification of XML processing approaches

access mode: **sequential** *vs.* **direct** (random)

control of the event stream: **pull** *vs.* **push**

tree management: **hierarchic** *vs.* **nested**
alternatives

DOM
random access, using pull style

SAX
sequential access, using push

XPP and .NET XmlTextReader
sequential access, adopting pull
alternatives

“Tying up” XML data to other data sources (XML binding)

databases: XML infoset ↔ dataset
alternatives

XML binding

databases: XML infoset ↔ dataset

SQL/XML specification – see SQL:2016-14 standard

aspects of interest:
XML type for table columns’ values
using XML-specific predicates + functions
databases: XML infoset ↔ dataset

existing implementations:

Oracle XML DB
docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb01int.htm

pureXML (IBM DB2)

SQLXML (Microsoft SQL Server)
msdn.microsoft.com/library/aa286527.aspx

XML Functions (PostgreSQL)
www.postgresql.org/docs/current/static/functions-xml.html
alternatives

XML binding

object-oriented approach:
XML data ↔ classes created “on the fly”
(serialization, marshalling)
object-oriented approach: XML data ↔ classes created “on the fly”

eamples:

C++ – cereal: http://uscilab.github.io/cereal/
C++, C#, Go, Java, Python – Protocol Buffers
developers.google.com/protocol-buffers/
Java – Digester: commons.apache.org/proper/commons-digester/
JS – node-xml2js: github.com/Leonidas-from-XIV/node-xml2js
.NET (C# et al.) – XmlSerializer class
Python – Untangle: github.com/stchris/untangle
Scala – scalaxb: http://scalaxb.org/
alternatives

XML binding

performing queries on XML data directly from the programing language

LINQ (Language INtegrated Query) – .NET Framework

docs.microsoft.com/dotnet/articles/csharp/programming-guide/concepts/linq/linq-to-xml
XDocument projects; // XDocument is a .NET class

projects = XDocument.Load("projects.xml");

var projectsS =
  // via a LINQ expression, getting all projects
  from p in projects.Descendants("project")
  // by choosing those of 'S' class
  where (String) p.Attribute("class") == "S"
  // sorted by student number
  orderby (String) p.Element("stud")
  // and selecting only their title
  select (String) p.Element("title");

// showing the title of 'S' class projects
foreach (var project in projectsS) {
  Console.WriteLine(project);
}

// same result, by using XPath
var projectsS2 = (IEnumerable) proiecte.XPathEvaluate("//project[@class='S']/title");
alternatives

XML binding

JAXB – Java Architecture for XML Binding (JSR-222)

reference implementation: https://jaxb.java.net/

also, experiment EclipseLink: www.eclipse.org/eclipselink/
alternatives

XML binding

interoperability with other formats: XML ↔ JSON

there is no standardized approach

examples of tools and libraries:
Apache Camel (Java), js2xmlparser (Node.js), JSON-lib (Java), ruby-xml-to-json, x2js (JavaScript), xml2json (Node.js), xml-to-json (Haskell), xmlutils.py
alternatives

Simplified XML processing

goal:
processing a (small) XML document directly into the memory, by using an object-oriented approach different from DOM
alternatives

Simplified XML processing

usually, the XPP (XML Pull Parsing) is adopted
alternatives

Simplified XML processing

for each XML element, an object property is available

the attributes attached to the XML elements are modeled by using a data structure – e.g., a hash table
alternatives

Simplified XML processing

various examples:

libxml (C, C++, and other languages)
XML::Simple + XML::Writer (Perl)
XmlSimple (Ruby)
XmlTextReader + XmlTextWriter (.NET)
// loading the XML document
$xml = simplexml_load_file('http://web.info/projects.xml');
// showing the descriptions of class S projects
foreach ($xml->project as $proj) {
    if ($proj['class'] == 'S') {
        echo '<p>' . $proj->desc . '</p>';
    }
}

// similarly, but using XPath
foreach ($xml->xpath("//project[@class='S']") as $proj) {
    echo '<p>' . $proj->desc . '</p>';
}

consult the examples from archive
Simplified XML processing
for data access, a “reader” is used: XMLReader

examples:

xmlreader module for Node.js
xmlReader provided by libxml library (C et al.)
XmlReader class from .NET (C# et al.)
alternatives

Simplified XML processing to generate data, a “writer” is used: **XMLWriter**

examples:

**XmlWriter** class for .NET

**xmlWriter** provided by **libxml** library (C et al.)


**xml-writer** module – Node.js
How HTML documents could be processed?
html processing

Aspect of interest: ignoring the syntax errors

well formed documents versus valid documents
html processing

Aspect of interest:
ignoring the syntax errors

malformed markup

there are relatively few cases when the HTML documents are written/generated correctly
html processing

A common used technique – not recommended

Web scraping

extracting the data of interest by processing – in an empirical manner, usually – the HTML markups
html processing

Using a specific HTML/XML processor

important goals:
traversing (processing) a Web page – e.g., via DOM
+ detecting & fixing the syntactic errors (HTML clean)

see previous lectures
**html processing: tools**

Beautiful Soup – a Python library
www.crummy.com/software/BeautifulSoup/

goquery – a Go library
silviosimunic.com/blog/web-scraping-with-go/

Gumbo – a HTML5 processor implemented in C (Google)
github.com/google/gumbo-parser

html5lib – HTML processing + serialization for Python
github.com/html5lib
html processing: tools

HTML::Gumbo, HTML::Parser – Perl modules
github.com/gisle/html-parser

Html Agility Pack – a .NET library
htmlagilitypack.codeplex.com

Hubbub – HTML5 markup processing in C
www.netsurf-browser.org/projects/hubbub/

Jericho HTML Parser – a Java library
jericho.htmlparser.net
html processing: tools

jsoup – a Java library for HTML5 processing
jsoup.org

Masterminds HTML5-PHP – a HTML5 processor (PHP)
masterminds.github.io/html5-php/

Nokogiri – a Ruby package
www.nokogiri.org

Parse5 – a Node.js module
inikulin.github.io/parse5/
html processing: tools

HtmlCleaner – Java tool for fixing bad HTML markup

HTML Purifier – verifies and filters HTML markups (including those used for XSS – *Cross Site Scripting* attacks) implementations in PHP and Objective-C

NekoHTML – Java processor for HTML based on Xerces having support for resolving syntax errors

Validator.nu – Java processor using DOM or SAX in order to report the HTML5 syntax errors
“conclusions”

XML processing: from SAX to XPP & Simple XML HTML document processing tools
next episode: **Web services** with SOAP