Simple API for XML

SAX (**Simple API for XML**) is an event-driven online algorithm for parsing XML documents, with an API developed by the XML-DEV mailing list.[1] SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

1 Definition

Unlike DOM, there is no *formal* specification for SAX. The Java implementation of SAX is considered to be normative.[2] SAX processes documents state-independently, in contrast to DOM which is used for state-dependent processing of XML documents.[3] SAX parsers contrast with DOM-style parsers in a similar way single-pass compilers contrast with multi-pass ones.

2 Benefits

A SAX parser only needs to report each parsing event as it happens, and normally discards almost all of that information once reported (it does, however, keep some things, for example a list of all elements that have not been closed yet, in order to catch later errors such as end-tags in the wrong order). Thus, the minimum memory required for a SAX parser is proportional to the maximum depth of the XML file (i.e., of the XML tree) and the maximum data involved in a single XML event (such as the name and attributes of a single start-tag, or the content of a processing instruction, etc.).

This much memory is usually considered negligible. A DOM parser, in contrast, has to build a tree representation of the entire document in memory to begin with, thus using memory that increases with the entire document length. This takes considerable time and space for large documents (memory allocation and data-structure construction take time). The compensating advantage, of course, is that once loaded *any* part of the document can be accessed in any order.

Because of the event-driven nature of SAX, processing documents is generally far faster than DOM-style parsers, *so long as* the processing can be done in a start-to-end pass. Many tasks, such as indexing, conversion to other formats, very simple formatting, and the like, can be done that way. Other tasks, such as sorting, rearranging sections, getting from a link to its target, looking up information on one element to help process a later one, and the like, require accessing the document structure in complex orders and will be much faster with DOM than with multiple SAX passes.

Some implementations do not neatly fit either category: a DOM approach can keep its persistent data on disk, cleverly organized for speed (editors such as SoftQuad Author/Editor and large-document browser/indexers such as DynaText do this); while a SAX approach can cleverly cache information for later use (any validating SAX parser keeps more information than described above). Such implementations blur the DOM/SAX tradeoffs, but are often very effective in practice.

Due to the nature of DOM, streamed reading from disk requires techniques such as lazy evaluation, caches, virtual memory, persistent data structures, or other techniques (one such technique is disclosed in [4]). Processing XML documents larger than main memory is sometimes thought impossible because some DOM parsers do not allow it. However, it is no less possible than sorting a dataset larger than main memory using disk space as memory to sidestep this limitation.[5]

3 Drawbacks

The event-driven model of SAX is useful for XML parsing, but it does have certain drawbacks.

Virtually any kind of XML validation requires access to the document in full. The most trivial example is that an attribute declared in the DTD to be of type IDREF, requires that there be an element in the document that uses the same value for an ID attribute. To validate this in a SAX parser, one must keep track of all ID attributes (any one of them *might* end up being referenced by an IDREF attribute at the very end); as well as every IDREF attribute until it is resolved. Similarly, to validate that each element has an acceptable sequence of child elements, information about what child elements have been seen for each parent, must be kept until the parent closes.

Additionally, some kinds of XML processing simply require having access to the entire document. XSLT and XPath, for example, need to be able to access any node at any time in the parsed XML tree. Editors and browsers likewise need to be able to display, modify, and perhaps re-validate at any time. While a SAX parser may well be used to construct such a tree initially, SAX provides no help for such processing as a whole.

4 XML processing with SAX

A parser that implements SAX (i.e., *a SAX Parser*) functions as a stream parser, with an event-driven API.[1] The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include (among others):

- XML Text nodes
- XML Element Starts and Ends
- XML Processing Instructions
- XML Comments

Some events correspond to XML objects that are easily returned all at once, such as comments. However, XML *elements* can contain many other XML objects, and so SAX represents them as does XML itself: by one event at the beginning, and another at the end. Properly speaking, the SAX interface does not deal in *elements*, but in *events* that largely correspond to *tags*. SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

There are many SAX-like implementations in existence. In practice, details vary, but the overall model is the same. For example, XML attributes are typically provided as name and value arguments passed to element events, but can also be provided as separate events, or via a hash or similar collection of all the attributes. For another, some implementations provide “Init” and “Fin” callbacks for the very start and end of parsing; others don't. The exact names for given event types also vary slightly between implementations.

5 Example

Given the following XML document:

<?xml version="1.0” encoding="UTF-8"?> <DocumentElement param="value"> <FirstElement> &#xb6; Some Text </FirstElement> <?some_pi some_attr="some_value"?> <SecondElement param2="something"> Pre-Text <Inline>Inlined text</Inline> Post-text. </SecondElement> </DocumentElement>

This XML document, when passed through a SAX parser, will generate a sequence of events like the following:

- XML Element start, named *DocumentElement*, with an attribute *param* equal to “value”
- XML Element start, named *FirstElement*
- XML Text node, with data equal to "&#xb6; Some Text” (note: certain white spaces can be changed)
- XML Element end, named *FirstElement*
- Processing Instruction event, with the target *some_pi* and data *some_attr="some_value”* (the content after the target is just text; however, it is very common to imitate the syntax of XML attributes, as in this example)
- XML Element start, named *SecondElement*, with an attribute *param2* equal to “something”
- XML Text node, with data equal to “Pre-Text”
- XML Element start, named *Inline*
- XML Text node, with data equal to “Inlined text”
- XML Element end, named *Inline*
- XML Text node, with data equal to “Post-text.”
- XML Element end, named *SecondElement*
- XML Element end, named *DocumentElement*

Note that the first line of the sample above is the XML Declaration and not a processing instruction; as such it will not be reported as a processing instruction event (although some SAX implementations provide a separate event just for the XML declaration).

The result above may vary: the SAX specification deliberately states that a given section of text may be reported as multiple sequential text events. Many parsers, for example, return separate text events for numeric character references. Thus in the example above, a SAX parser may generate a different series of events, part of which might include:

- XML Element start, named *FirstElement*
- XML Text node, with data equal to "&#xb6;" (the Unicode character U+00b6)
- XML Text node, with data equal to " Some Text”
- XML Element end, named *FirstElement*

6 See also

- Document Object Model
- Expat (XML)
- Java API for XML Processing

- LibXML
- List of XML markup languages
- List of XML schemas
- MSXML
- StAX
- Streaming XML
- VTD-XML
- Xerces
- XQuery API for Java
- XSLT (Extensible Stylesheet Language Transformation)

7 References

[1] "SAX". http://www.webopedia.com/: WEBOPEDIA. Retrieved 2011-05-02. Short for Simple API for XML, an event-based API that, as an alternative to DOM, allows someone to access the contents of an XML document. SAX was originally a Java-only API. The current version supports several programming language environments other then Java. SAX was developed by the members of the XML-DEV mailing list.

[2] saxproject.org

[3] "Simple API for XML". http://oracle.com/: ORACLE. Retrieved 2011-05-02. Note: In a nutshell, SAX is oriented towards state independent processing, where the handling of an element does not depend on the elements that came before. StAX, on the other hand, is oriented towards state dependent processing. For a more detailed comparison, see SAX and StAX in Basic Standards and When to Use SAX.

[4] US patent 5557722

[5] "XML Parsers: DOM and SAX Put to the Test". http://www.devx.com/xml/article/16922/1954: devX. Although these tests do not show it, SAX parsers typically are faster for very large documents where the DOM model hits virtual memory or consumes all available memory.

8 Further reading

- David Brownell: *SAX2*, O'Reilly, ISBN 0-596-00237-8
- W. Scott Means, Michael A. Bodie: *The Book of SAX*, No Starch Press, ISBN 1-886411-77-8

9 External links

- SAX home page
- SAX at Cafe Con Leche
- Top Ten SAX2 Tips
- See SAX in action with a javascript API
- SAX parser in Python
- Saxon parser in Java

10 Text and image sources, contributors, and licenses

10.1 Text

10.2 Images

10.3 Content license