PERFORMANCE ANALYSIS OF JAVA APIs FOR XML PROCESSING

Bruno Oliveira¹, Vasco Santos¹ and Orlando Belo²
¹CIICESI, School of Management and Technology, Polytechnic of Porto
Felgueiras, PORTUGAL
{bmo,vsantos}@estgf.ipp.pt
²Algoritmi R&D Centre, University of Minho
4710-057 Braga, PORTUGAL
obelo@di.uminho.pt

ABSTRACT

Over time, XML markup language has acquired a considerable importance in applications development, standards definition and in the representation of large volumes of data, such as databases. Today, processing XML documents in a short period of time is a critical activity in a large range of applications, which imposes choosing the most appropriate mechanism to parse XML documents quickly and efficiently. When using a programming language for XML processing, such as Java, it becomes necessary to use effective mechanisms, e.g. APIs, which allow reading and processing of large documents in appropriated manners. This paper presents a performance study of the main existing Java APIs that deal with XML documents, in order to identify the most suitable one for processing large XML files.

KEYWORDS

XML, XML Markup languages, XML Documents, Java API, Performance Analysis.

1 INTRODUCTION

Due to the simplicity of its hierarchical structure, XML (Extensible Markup Language) is widely used for data representation in many applications. As a result of its portability, XML is used to ensure data interchanging among systems with high heterogeneous natures, facilitating data communication and sharing, it’s platform independent, which makes it quite attractive for the majority of applications. Associated with the XML format there are other languages that complement the application area of this format, such as XSD, XSLT or XQuery. Currently, XML format is used in the development of several types of software, including web pages, web services, network applications, and fully based XML databases.

Access and modification operations are essential to XML files manipulation once they are affected by any increasing amount of data, by the complexity of those operations, and by shorter periods of time needed to process them. Coupled with this data growing, XML documents can reach large number of megabytes (or even gigabytes), limiting and conditioning the technology used for development of applications appealing for XML data processing. Also coupled with the concept of portability, Java programming language provides a set of interfaces allowing for the manipulation of structured documents according to the XML format. Due to their portability, Java and XML are commonly used in application development.

The main focus of this paper was to conduct a study of the various parsing models and APIs (Application Programming Interface) for XML processing using Java programming language, with the purpose to supply a refresh benchmark to the available representation models, identifying which is the most suitable for access and transformation of large XML documents. We also refer the main advantages identified for each representation model, always keeping the performance factor in mind. The next section (2) examines some memory and streaming representation models, identifying how documents are processed according to each parsing model. Section 3 and 4 present some memory-based APIs and streaming-based APIs and their features. Section 5 compares performance and memory consumption of
SAX and XOM. This kind of APIs do not require an additional effort to dominate the memory, proved being far superior in relation to other memory-based APIs. Memory usage in streaming APIs is much more accessible and quick, since for them modifications are allowed, while in the two approaches studied for processing XML documents, dependencies are not possible. Results show a clear superiority of VTD for temporary buffers to keep information in memory.

Due to its complexity and importance, the parsing process is the most critical operation in XML processing, directly conditioning processing time and memory consumption. Several studies [1–6] have been conducted with the goal to test and improve representation models and APIs in XML processing [7]. However, some of them did not focus its context on Java programming and others are outdated. This is mainly caused by miscellaneous updates and improvements in the execution environment, particularly in the Java Virtual Machine, which affects, as we know, runtime and effectiveness of the operations.

In [8] the process of handling XML documents was described in four phases: Parsing that is considered a critical step in performance, Access, Modification and Serialization, whose performance is directly affected by the parsing models. As the most critical factor of performance, parsing is characterized by the conversion of characters, mainly related to the conversion of characters into a format that a programming language understands, lexical analysis which is the process that identifies XML elements, e.g. start node, end node or characters, applying regular expressions defined by World Wide Web Consortium (W3C). The last step of the parsing phase is the syntactic analysis of the document, where it is checked if the document complies with the rules of construction of an XML document. Finally, the API implements access and modification operations on the data resulted from the parsing process.

2.1 Memory-based representation models

Most memory-based APIs use a common model in data processing, where XML documents are entirely stored in memory in a tree format with multiple nodes, descending all from a single node representing the root of the tree. This kind of schema allows the use of different methods to locate and manipulate data contained inside the nodes. For each search, or other kind of manipulation, it is necessary to start the processing by the root element continuing in the structure hierarchy to access the remaining data (figure 1). Since all the information is available in memory, we can traverse the tree in random order, changing the positioning of the nodes and performing data transformations in a very simple and accessible way. Considering its memory structure.

1http://www.w3.org/
representation, these APIs facilitate application development, providing a wide range of search methods that allow you to easily perform operations on the constituent nodes of the tree. However memory-based APIs consume, in average, four to five times more memory than the document’s size. For example, a 20 megabytes document needs, depending on the representation model, approximately 100 megabytes in order to be stored in memory, which may represent a problem in processing large documents.

2.2 Streaming-based models

Streaming-based APIs perform a sequential scan of the document using minimum memory resources. Typically, this type of APIs use the depth of the XML document (number of nested elements) and the maximum data stored in XML attributes on a single XML element. Both of these are always smaller than the size of the memory-based parsing tree approach. Then, a small portion of the document is extracted sequentially without the need to load the whole document structure. Usually, the parser reads the XML document calling a specific method for each type of event to process its object. Figure 2 presents the SAX (Simple API for XML Processing)\(^2\) conceptual model for XML processing, which is similar to other streaming-based APIs.

The parser is configured as an input source, which is associated with a set of content management methods that identify, for example, the beginning or the end of the document and elements of data that might contain errors that occurred during the parsing step. When the parser runs, event triggers are captured by content management methods. Each time the parser detects an important part of the XML document it triggers the appropriate method in order to read the respective data block. Streaming-based APIs are more suitable for processing large XML documents, because, in theory, they can process documents of infinite size.

3 MEMORY-BASED APIs

In order to get a memory-based API overview for XML processing in Java, we conducted a specific study to determine which APIs are more efficient in memory management and which is faster in XML processing. We covered the following APIs: DOM\(^3\), XOM\(^4\), OJXQI\(^5\), VTD\(^6\), JDOM\(^7\), dom4j\(^8\) and Xerces2\(^9\). Included in JAXP package, DOM API is a collection of classes that has a set of Java methods that allows XML processing in memory with a structure similar to figure 1. In several cases, the DOM API is the basis for the construction of new APIs that revise some of its characteristics, with the aim of serving specific requirements. For instance, the JDOM API allows the manipulation of XML documents with Java via a tree structure representation, thus being similar to DOM. However, this API has been developed specifically for Java language, making it much more intuitive for a typical Java programmer. For example, there is no Text class [9], since Java programming language provides its own class (String class). JDOM takes advantage of Java features such as: creating methods with the same name, reflection\(^10\), weak references\(^11\), and the use

---

\(^2\)http://www.saxproject.org

---

3 http://download.oracle.com/javase/6/docs/technotes/guides/xml/
4 http://www.xom.nu
5 http://www.oracle.com/technetwork/database/features/xmldb/index-087544.html
6 http://vtd.xml.sourceforge.net
7 http://www.jdom.org
8 http://dom4j.sourceforge.net/dom4j-1.6.1
9 http://xerces.apache.org/xerces2-j
10 http://java.sun.com/developer/technicalArticles/ALT/Reflection/