ALTAMIRA (/)

Performance: LINQ to XML vs XmlDocument vs XmlReader

05.01.08 | JOE FERNER (/profiles/jferner) | Employee Post (/blog/employee-posts)

I recently had a project where I needed to ingest large XML documents using C# so I was curious which XML reader technology would be the fastest. So I coded up a quick benchmark that would compare LINQ to XML, XmlDocument.Load, and XmlReader against each other.

The Test Data

I generated a very simple XML file before each run of a test. The id's were random and the number of "child" nodes varied based on the run. The following is an example of the test data I used.

```
<span class="kwrd">&lt;</span><span class="html">root</span><span class="kwrd">&gt;</s
pan>
  <span class="kwrd">&lt;</span><span class="html">child</span> <span class="attr">id<
/span><span class="kwrd">='123'</span><span class="kwrd">/&gt;</span>
  <span class="kwrd">&lt;</span><span class="html">child</span> <span class="attr">id<
/span><span class="kwrd">='234'</span><span class="kwrd">/&gt;</span>
  ...
<span class="kwrd">&lt;/</span><span class="html">root</span><span class="kwrd">&gt;</
span>
```

The Test

As I said before I wanted to compare LINQ to XML, XmlDocument.Load, and XmlReader against each other. I ran each of these technologies using 1, 10, 100, 1000, 10,000, 100,000 "child" nodes. I also ran each against a XML document using UTF-8, ASCII, and UTF-32 encodings. Each iteration was run 100 times to reduce anomalies. In each of the tests I call the method "ProcessId" which simulates the processing of the "id" attribute.

XmlDocument.Load

I thought the code for XmlDocument.Load was the cleanest and easiest to understand, although I must admit I like XPath. XmlDocument does have some security concerns but that's another post. Here is the code I used to load and search the document:

```
<span class="kwrd">private</span> <span class="kwrd">static</span> <span class="kwrd">
void</span> XmlDocumentReader(<span class="kwrd">string</span> fileName) {
    XmlDocument doc = <span class="kwrd">new</span> XmlDocument();
    doc.Load(fileName);
    XmlNodeList nodes = doc.SelectNodes(<span class="str">"//child"</span>);
    <span class="kwrd">if</span> (nodes == <span class="kwrd">null</span>) {
        <span class="kwrd">throw</span> <span class="kwrd">new</span> ApplicationExcep
tion(<span class="str">"invalid data"</span>);
    }
    <span class="kwrd">foreach</span> (XmlNode node <span class="kwrd">in</span> nodes
) {
        <span class="kwrd">string</span> id = node.Attributes[<span class="str">"id"</
span>].Value;
        ProcessId(id);
    }
}
```

LINQ to XML

LINQ to XML was also very easy to read and understand code. I did find that even though LINQ to XML is supposed to use XmlReaders under the covers calling XDocument.Load does read the whole document into memory before returning. So if you are looking for data at the top of middle of a very large document this could be a concern. Here is the code I used to load and search the document:

```
<span class="kwrd">private</span> <span class="kwrd">static</span> <span class="kwrd">
void</span> XDocumentReader(<span class="kwrd">string</span> fileName) {
    XDocument doc = XDocument.Load(fileName);
    <span class="kwrd">if</span> (doc == <span class="kwrd">null</span> | doc.Root ==
<span class="kwrd">null</span>) {
        <span class="kwrd">throw</span> <span class="kwrd">new</span> ApplicationExcep
tion(<span class="str">"invalid data"</span>);
    }
    <span class="kwrd">foreach</span> (XElement child <span class="kwrd">in</span> doc
.Root.Elements(<span class="str">"child"</span>)) {
        XAttribute attr = child.Attribute(<span class="str">"id"</span>);
        <span class="kwrd">if</span> (attr == <span class="kwrd">null</span>) {
            <span class="kwrd">throw</span> <span class="kwrd">new</span> ApplicationE
xception(<span class="str">"invalid data"</span>);
        }
        <span class="kwrd">string</span> id = attr.Value;
        ProcessId(id);
    }
```

```
}
```

XmlReader

XmlReader, specifically XmlTextReader was the hardest to write and understand. With it's quirks of being a forward only reader you need to take what you need while you have it because you can't rewind.

```
<span class="kwrd">private</span> <span class="kwrd">static</span> <span class="kwrd">
void</span> XmlReaderReader(<span class="kwrd">string</span> fileName) {
    <span class="kwrd">using</span> (XmlReader reader = <span class="kwrd">new</span>
XmlTextReader(fileName)) {
        <span class="kwrd">while</span> (reader.Read()) {
            <span class="kwrd">if</span> (reader.NodeType == XmlNodeType.Element) {
                <span class="kwrd">if</span> (reader.Name == <span class="str">"child"
</span>) {
                    reader.MoveToAttribute(<span class="str">"id"</span>);
                    <span class="kwrd">string</span> id = reader.Value;
                    ProcessId(id);
                }
            }
        }
    }
}
```

The Results

The following results are in milliseconds for each run. I took the total time to run and divided it by 100.

UTF8Encoding

	1	10	100	1,000	10,000	100,000
XmlDocument	0.1567800	0.1713450	0.3888620	1.9816480	22.8049260	459.8570340
XmlReader	0.1467460	0.1439580	0.2300500	0.8534400	7.5771640	76.8635690
LINQ to XML	0.1499530	0.1500640	0.2778720	1.4616730	15.7719020	208.9360300

ASCIIEncoding

	1	10	100	1,000	10,000	100,000
XmlDocument	0.1659350	0.1922080	0.3433140	1.9846330	22.5484690	482.8699720
XmlReader	0.1376840	0.1453730	0.2199810	0.8768260	7.9187380	77.7760560
LINQ to XML	0.1345900	0.1573340	0.2848420	1.4889930	15.1504500	214.9338990

UTF32Encoding

	1	10	100	1,000	10,000	100,000
XmlDocument	0.1672370	0.1799780	0.4156250	2.7188370	30.6423960	543.4604540
XmlReader	0.1386820	0.1503870	0.2867400	1.4981070	14.4428430	152.7660780
LINQ to XML	0.1317060	0.1866610	0.5385940	2.3631290	21.4566290	274.3280280

Conclusion

XmlReader beats LINQ to XML in almost every run except for very small XML documents. What's interesting is how the numbers scale between the encodings. XmlReader is over twice as slow when reading UTF-32 documents verse UTF-8 or ASCII encoded XML, yet LINQ to XML and XmlDocument slowed down by a much smaller amount. If you need speed when reading XML documents stick with XmlReader. If you need readability and maintainability of your code go with LINQ to SQL or XmlDocument.
Updated on: 06.13.2013

TAGS: .NET (/blog/tag/.net)

COMMENTS

Andrew said:

Very interesting article.

Regarding your comment about utf32 encoding.

The percentage change of the numbers don't make sense, but if you look at the absolute change, the numbers are much closer.

For instance. For 100,000 nodes, comparing ascii to utf32.

XmlDocument -> 61s difference

XmlReader -> 75s difference

LINQ to XML -> 60s differenceI'm guessing that XmlReader spends most of it's time reading the document(which is 4 times larger), as opposed to processing the document.

05.01.08 Reply

Anders said:

You can make the LINQ to XML code substantially simpler. Two lines should do it:var ids = XElement.Load(fileName).Elements("child").Attributes("id");

foreach (XAttribute a in ids) ProcessId((string)id);It is not surprising the XmlReader code is faster since both the XML DOM (XmlDocument) and LINQ to XML use XmlReader to do their reading.Anders

05.01.08 Reply

james osburn said:

what is the fast way to generate an xml document?

from say web service?

regards

james

07.01.09 Reply

thorn said:

Didn't you forget about XPathDocument?

10.20.09 Reply

Eduardo said:

I'm interested on how did you test them. Cuz you know, a good test involves warm up, processor afinity/priority change, RELEASE config and a Run Without Debug start. Could you show us the test code? I'd aprreciate if you could send it to my personal e-mail

05.26.10 Reply

Raj Aththanayake said:

Thanks for sharing the info.

06.23.10 Reply

Walkincg said:

this is the diffrence in time taking by diffrent XMl technique . But what is the mai diff between them in functionality wise.

07.15.10 Reply

Alan Yost said:

Thanks for the sharing the effort.

02.09.11 Reply

Schalk said:

You should try combining XmlReader and LINQ to XML as described in this post:

http://blogs.msdn.com/b/xmltea... (http://blogs.msdn.com/b/xmlteam/archive/2007/03/24/streaming-with-linq-to-xml-part-2.aspx)Should be a interesting comparison.

06.16.11 Reply

Satya said:

Thats Great, Joe.

I found out a great difference when I moved from LINQ to XML to XMLReader.

Thanks for your article. Definately be useful to many.

07.08.11 Reply

Jeremy Child said:

Typo there in Conclusion section:"If you need readability and maintainability of your code go with LINQ to SQL or XmlDocument."Should be LINQ to XML.

07.13.11 Reply

null said:

Thanks for the post, it was quite informative.

But what did you mean by saying "XmlDocument does have some security concerns"?

Did you mean something serious (similar to XmlSerializer security concerns - having to run under administrator and have access to TEMP path) or something less horrible as:

"Exceptions raised as a result of using the XmlDocument class, such as the XmlException class may contain sensitive information that should not be exposed in untrusted scenarios."?

10.21.11 Reply

Jeff Davis said:

Dog, great read. Keep up the good work.

12.27.11 Reply

The Dag said:

Interesting test, but would have been so much more so if the data used wasn't so ridiculously contrived. Why not just search up a collection of XML files on your local hard disk, pick a large set with varying sizes, and write the code so it processes every element and attribute?The "test" as it stands is near worthless in my opinion, as there is no way to know if the results are due to the very particular nature of the XML data used. It could well be that the situation would be the opposite for a document that has a very deep structure, or even one with three or four levels.Using real-world data would have made me much more confident that these are real-world results.

03.27.12 Reply

Joe Ferner said:

@The Dag: We ended up trying both methods on our project and found the results to be about the same. XmlReader always came up on top.

04.11.12 Reply

Alan8 said:

Useful info; thanks for sharing!

04.12.12 Reply

Bharat Ram said:

Thanks for the very nice comparison

- Bharat

11.27.12 Reply

hoangedward said:

Thanks for your nice analyzation.But I saw that your C# code is mixing with HTML codeCould you please to remove all redundant code for readable

04.09.13 Reply

Sabby said:

Really appreciate your sharing of the work. Thanks

10.02.13 Reply

New Comment

Author

Body

Markdown Supported

Comment Toggle Preview

SEARCH

(/blog/search?query=)

ANALYTICS (/blog/analytics)

VISUALIZATION (/blog/visualization)

COMMS & SENSORS (/blog/comms-sensors)

CYBER (/blog/cyber)

MOBILITY (/blog/mobility)

BIG DATA (/blog/big-data)

Join Us

All Source Targeting Intel Trainer (https://timesheets.altamiracorp.com/job/1437)

Fort Bragg, NC

Responsibilities: The Trainer position must have the ability to train analysts possess ...
(https://timesheets.altamiracorp.com/job/1437)

Multi-Layer Targeting Analyst (https://www.altamiracorp.com/job/1445)

Washington, DC

Responsibilities: These analysts possess strong organizational skill, analytical ...
(https://www.altamiracorp.com/job/1445)

View All (/jobs)

More By Joe

Microprocessor: Arduino (/blog/tech-talks/Microprocessor_Arduino)

Joe Ferner (https://www.altamiracorp.com/profiles/jferner) on 04.12.12

How to use libwireshark to dissect a packet (/blog/employee-posts/how-to-use-libwireshark-to-dis)

Joe Ferner (https://www.altamiracorp.com/profiles/jferner) on 01.10.12

Type-Safe Entity Framework Include (/blog/employee-posts/type-safe-entity-framework-inc)

Joe Ferner (https://www.altamiracorp.com/profiles/jferner) on 11.18.10

Related by Tags

Google Play vs Windows Store: A Stark Contrast (/blog/employee-posts/google-play-vs-windows-store-a-stark-contrast)

Lee Richardson (https://www.altamiracorp.com/profiles/lrichard) on 01.22.13

ASP .NET MVC Windows Server Setup (/blog/employee-posts/asp-net-mvc-windows-server-set)

Sean Howell (https://www.altamiracorp.com/profiles/showell) on 06.04.12

Integrating JavaScript and C# with Script# (/blog/employee-posts/integrating-javascript-and-c-w)

Lee Richardson (https://www.altamiracorp.com/profiles/lrichard) on 08.24.11

Twitter (https://twitter.com/explorealtamira)

Facebook (https://www.facebook.com/AltamiraTechnologiesCorporation)

YouTube (https://www.youtube.com/user/nearinfinity)

LinkedIn (https://www.linkedin.com/company/altamira-corporation)

CONTACT

(/contact-us)

Headquarters (/contact-us)
8201 Greensboro Drive
Suite 800
McLean, Va 22102

703.813.2100

703.813.1740

info@altamiracorp.com (mailto:info@altamiracorp.com)

amazon web services | Partner Network

CONSULTING PARTNER