EDIT - bug #9147

Use StringBuilder in CdmUtils.concat() to avoid performance penalties

07/13/2020 02:03 PM - Andreas Kohlbecker

Status:	Closed	**Start date:**	
Priority:	New	**Due date:**	
Assignee:	Andreas Müller	**% Done:**	100%
Category:	cdmlib	**Estimated time:**	0:00 hour
Target version:	Release 5.17		
Severity:	normal	**Found in Version:**	

Description

Use String.join() or one of the org.apache.commons.lang3.StringUtils.join(..) methods instead. These methods are making explicit use of the StringBuffer class. CdmUtils.concat() can not be optimized by the jit, so for each string item the new String object needs to be extended to new length, which is known to be a performance penalty.

~~The CdmUtils.concat() methods have been deprecated now.~~

Associated revisions

Revision 11c127bf - 07/13/2020 02:05 PM - Andreas Kohlbecker

ref #9147 deprecating and commenting CdmUtils.concat()

Revision 8ab5e47c - 07/14/2020 09:46 AM - Andreas Müller

fix #9147 use StringBuffer in concat()

Revision 63b2dbdb - 07/14/2020 03:20 PM - Andreas Müller

fix #9147 use StringBuilder instead of StringBuffer

History

#1 - 07/13/2020 02:05 PM - Andreas Kohlbecker

- Description updated

#2 - 07/13/2020 02:17 PM - Andreas Müller

- Status changed from New to Feedback

- Assignee changed from Andreas Müller to Andreas Kohlbecker

- Target version changed from Unassigned CDM tickets to Release 5.18

The semantics of String.join() and StringUtils.join() is different to the one of CdmUtils.concat() especially in terms of handling null values therefore I disagree to setting it to @deprecated.
The current solution is very often used and has exactly the semantics that is needed for those cases.

If we want to improve performance solution would be to use StringBuffer within CdmUtils.concat()

#3 - 07/14/2020 09:46 AM - Andreas Müller

- Status changed from Feedback to Resolved

- % Done changed from 0 to 50

Applied in changeset cdmlib|8ab5e47c6a6ca132f2e495900c3bdfc7743d69c8.

#4 - 07/14/2020 09:52 AM - Andreas Müller

- Description updated

I changed the method to use StringBuffer now.

However, I tested the performance difference and it is true that using StringBuffer is a bit faster (at least for large amounts of data). However the difference is small. Concatenating 1 Mio Strings of length 50 saved 200-300 ms, which is 0.0002 ms per concatenation. In most contexts this is not a relevant issue compared to other operations. Only when concatenating huge numbers of strings it should be considered.

Please review.

#5 - 07/14/2020 12:54 PM - Andreas Kohlbecker

- Status changed from Resolved to Feedback

- Assignee changed from Andreas Kohlbecker to Andreas Müller

Andreas Müller wrote:

> I changed the method to use StringBuffer now.
>
> However, I tested the performance difference and it is true that using StringBuffer is a bit faster (at least for large amounts of data). However the difference is small. Concatenating 1 Mio Strings of length 50 saved 200-300 ms, which is 0.0002 ms per concatenation. In most contexts this is not a relevant issue compared to other operations. Only when concatenating huge numbers of strings it should be considered.
>
> Please review.

Since jdk1.5 StringBuilder should be used instead of StringBuffer in situations which are guaranteed single threaded operations on the strings to be created.

When writing tests to compare the performance of + with StringBuilder or StringBuffer you need to write the code in a way which prevents the compiler from optimizing the + operator. Otherwise Java will internally translate it to StringBuilder. Please see for example https://www.java67.com/2015/05/4-ways-to-concatenate-strings-in-java.html for a comparison of several ways to concatenate strings: "You can clearly see that given everything same, StringBuilder outperform all others. It's almost **3000 times faster** than + operator."

#6 - 07/14/2020 03:20 PM - Andreas Müller

- Status changed from Feedback to Resolved

Applied in changeset cdmlib|63b2dbdb54bc105a8fa615e36eb754eeed6d3596.

#7 - 07/14/2020 03:22 PM - Andreas Müller

- Assignee changed from Andreas Müller to Andreas Kohlbecker

Interesting. I didn't know about the differences between StringBuilder and StringBuffer. I adapted the code accordingly.

#8 - 07/14/2020 03:38 PM - Andreas Müller

Andreas Kohlbecker wrote:

> When writing tests to compare the performance of + with StringBuilder or StringBuffer you need to write the code in a way which prevents the compiler from optimizing the + operator. Otherwise Java will internally translate it to StringBuilder. Please see for example https://www.java67.com/2015/05/4-ways-to-concatenate-strings-in-java.html for a comparison of several ways to concatenate strings: "You can clearly see that given everything same, StringBuilder outperform all others. It's almost **3000 times faster** than + operator."

There is an important mistake in the text of the above link. It is not the +operator which is expensive but the assignment(=) operator. And this is especially the case if you assign very large strings as the data has to be copied from one place to another in memory then. The +operator itself is handled like StringBuilder internally so it does not cost much more. You can test this by replacing in the given test class buffer.append(i); by s = StringUtils.join(...) which internally uses StringBuilder. The result is about the same size as for the +operator so using StringBuilder has no result, but because the code then also has an assignments per iteration it becomes slow.
Also you can see that if you use small number of iterations which results in smaller Strings to be copied the factor (which you say is 3000) is much smaller (see last example on the given page). This is because the assignments are done only on small (realistically sized) strings for which the difference is not so big.

So my conclusion is: there is no real problem with +operator but you need to be carefull with assignments (=) especially if you handle large Strings. So it is good that we updated the method as it was also using assignment operators.

(Note: there is another error on the given test class. Before testing s.concat(Integer.toString(i)); they forgot to set parameter s back to empty string. Concat has a much better performance than said on the page.)

#9 - 07/15/2020 02:58 PM - Andreas Müller

- Subject changed from avoid using CdmUtils.concat() which causes performance penalties to Use StringBuilder in CdmUtils.concat() to avoid performance penalties

#10 - 07/16/2020 08:48 PM - Andreas Kohlbecker

- Status changed from Resolved to Closed

- Assignee changed from Andreas Kohlbecker to Andreas Müller

- % Done changed from 50 to 100

With the final adaption of its subject this ticket is fully solved.

Code is perfect.

#11 - 08/19/2020 08:41 PM - Andreas Müller

- Target version changed from Release 5.18 to Release 5.17