Edit - bug #7900

Excessive heap consuption in RegistrationWorkingsetEditor with big workingsets

11/09/2018 12:34 PM - Andreas Kohlbecker

Status:	Closed	**Start date:**	11/09/2018
Priority:	Highest	**Due date:**	
Assignee:	Andreas Kohlbecker	**% Done:**	100%
Category:	cdm-vaadin	**Estimated time:**	0.00 hour
Target version:	Release 5.5		
Severity:	blocker	**Found in Version:**	

Description

Adding a new name to the registration workingset *Kulikovskiy, M., Lange-Bertalot, H., Metzeltin, D. & al. - Lake Baikal: Hotspot of endemic diatoms I. in Taxonomy - biogeography - diversity. 23.* (http://api.cybertaxonomy.org/phycobank/app/registration#!workingset/5d29277e-a527-4a26-8685-1a7a6cb2e3de) fails on the production server due to excessive heap consumption after pressing the save button in the name editor.

The problem happens after the name is saved during the reloading of the workingset:

A differential comparison of the heap before and during the reloading of the workingset shows that especially the amount Team objects is increasing excessively.

1 minute after saving +25.000 Team objects

mem-diff-1.png

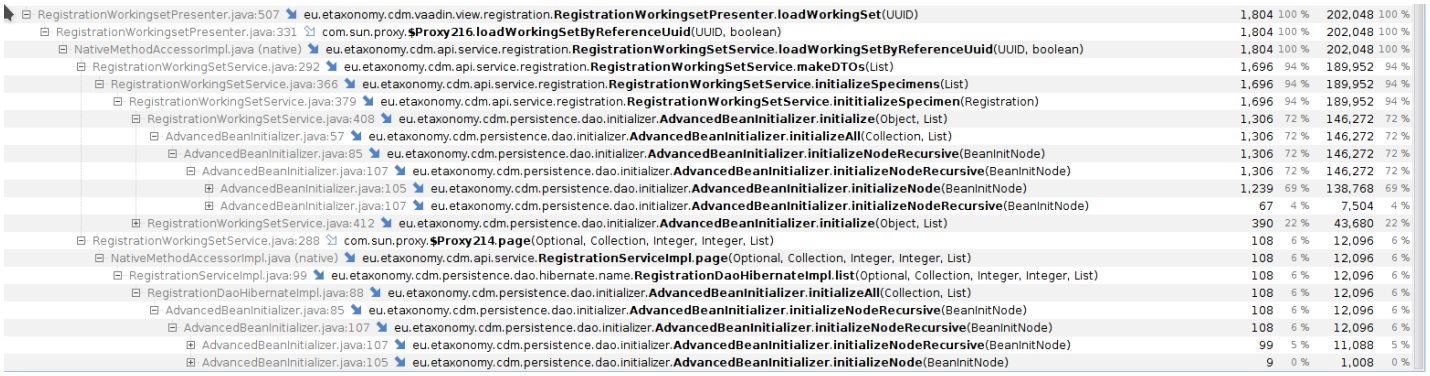
3 minutes after saving +109.000 Team objects

mem-diff-2.png

Comparing the objects in memory from 3 minutes and 1 minute after saving reveals that **only the amount of Team objects is increasing that much!**

mem-diff-1-2.png

The allocation analysis shows that all Team Objects have been created in the call to the RegistrationWorkinsetService



There are **423** Teams in the database but in the test above more than 100.000 Team objects have been created evene after 3 minutes.

Related issues:

Copied to Edit - bug #7904: DerivedUnitFacadeFieldUnitCacheStrategy.getCollec...	**Feedback**	**11/13/2018**

Associated revisions

Revision 07bbd695 - 11/12/2018 04:46 PM - Andreas Kohlbecker

ref #7900 comitting writable transaction before loading the registration working set

Revision 4d18f3c4 - 11/12/2018 05:07 PM - Andreas Kohlbecker

ref #7900 read-only transactions in DTO service with Propagation.REQUIRES_NEW

Revision 923fd553 - 11/13/2018 09:19 AM - Andreas Kohlbecker

ref #7900 reverting : read-only transactions in DTO service with Propagation.REQUIRES_NEW

Revision f277df8a - 11/13/2018 03:53 PM - Andreas Kohlbecker

ref #7900 removing remains from old code

History

#1 - 11/09/2018 12:36 PM - Andreas Kohlbecker

- File mem-diff-2.png added

#2 - 11/09/2018 12:41 PM - Andreas Kohlbecker

- File mem-diff-1.png added

- Description updated

#3 - 11/09/2018 12:44 PM - Andreas Kohlbecker

- File mem-diff-1-2.png added

#4 - 11/09/2018 12:45 PM - Andreas Kohlbecker

- Description updated

#5 - 11/12/2018 11:34 AM - Andreas Kohlbecker

- File Allocations-of-Teams.png added

- Description updated

#6 - 11/12/2018 11:35 AM - Andreas Kohlbecker

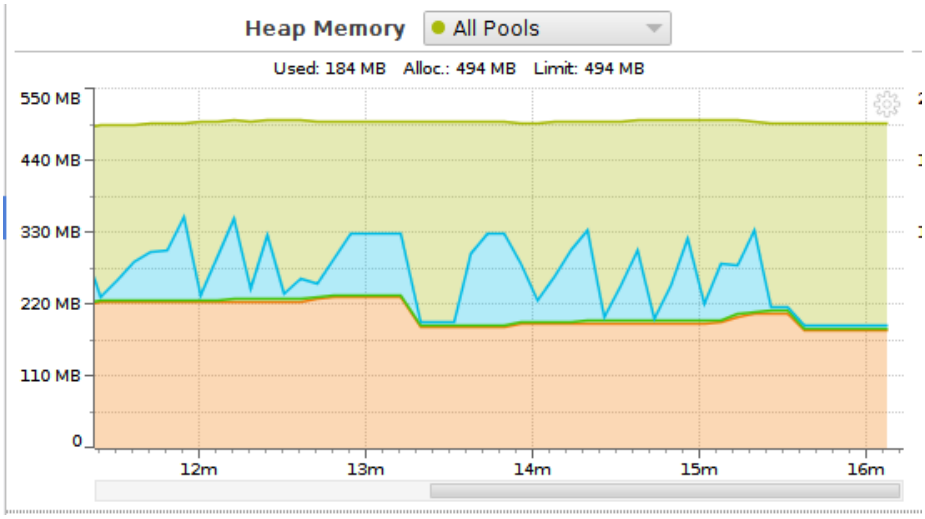
- Description updated

#7 - 11/12/2018 02:12 PM - Andreas Kohlbecker

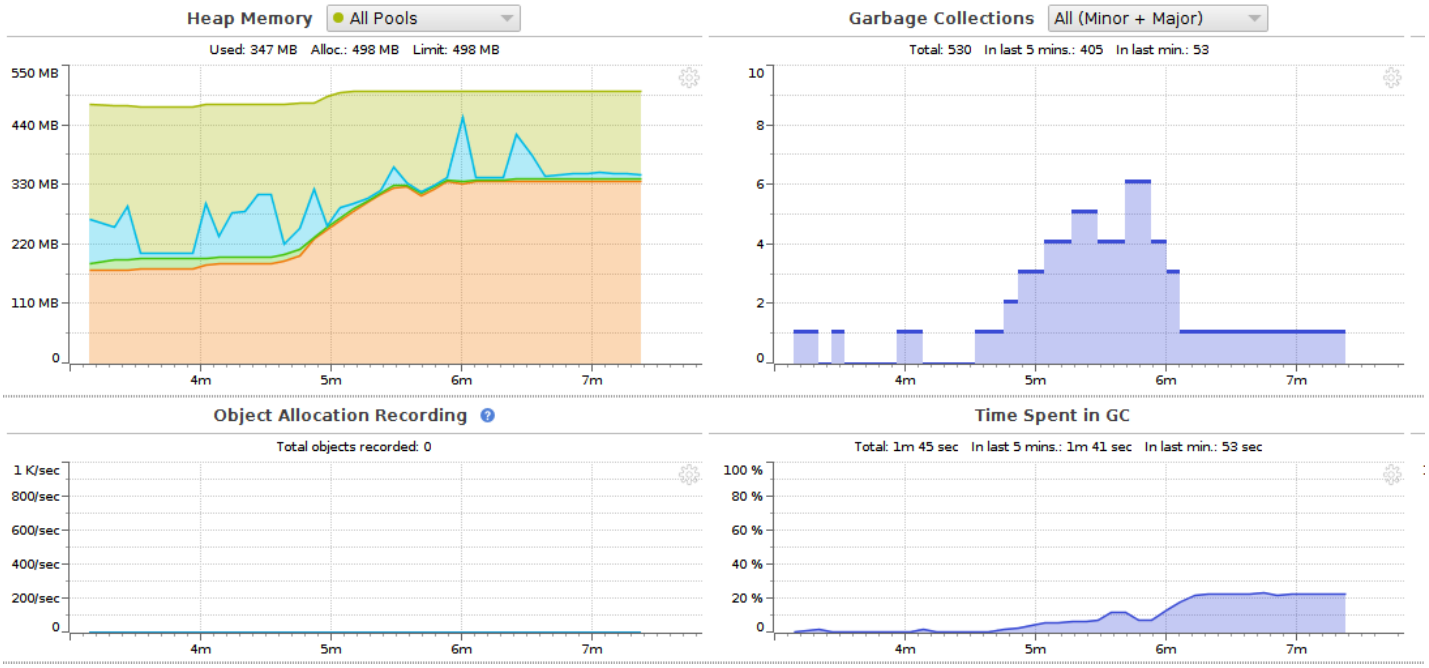
- File mem-getCollectorAndFieldNumber.png added

- File mem-getCollectorAndFieldNumberXX.png added

It seems as if the DerivedUnitFacadeFieldUnitCacheStrategy.getCollectorAndFieldNumber() is causing the massive memory usage. In this method new Team instances are created. Replacing this method by a empy implementation which only returns an empty string solves at least the memory issue:



In contrast to the above graph the getCollectorAndFieldNumber() will create a massive load of Teams which can not be garbage collected:



#8 - 11/12/2018 04:45 PM - Andreas Kohlbecker

- % Done changed from 0 to 20

The root cause for the problem is the RegistrationWorkingsetPresenter.onDoneWithTaxonnameEditor() method in which a write enabled transaction is committed at the wrong time. The commit takes place after reloading the registration workingset which causes the whole workingset to be reloaded in the writable transaction since the RegistrationWorkingSetService misses the transaction propagation type being set to Propagation.REQUIRES_NEW in @Transactional(readOnly=true). With REQUIRES_NEW a new read-only transaction would be created even if the service is called in a writable transaction.

Despite having pinned down the root cause to these two findings it is still quite irritating that the DerivedUnitFacadeFieldUnitCacheStrategy.getCollectorAndFieldNumber() can impose such a massive load on the heap even if it is only returning plain strings.

#9 - 11/12/2018 05:08 PM - Andreas Kohlbecker

- % Done changed from 20 to 50

#10 - 11/12/2018 05:33 PM - Andreas Kohlbecker

- Description updated

#11 - 11/12/2018 06:21 PM - Andreas Kohlbecker

- Status changed from New to Resolved

#12 - 11/13/2018 02:11 PM - Andreas Kohlbecker

- Copied to bug #7904: DerivedUnitFacadeFieldUnitCacheStrategy.getCollectorAndFieldNumber() creates temporary Team objects which can not be garbage collected. added

#13 - 11/13/2018 02:11 PM - Andreas Kohlbecker

The DerivedUnitFacadeFieldUnitCacheStrategy.getCollectorAndFieldNumber() is actually the root cause for the massive memory allocation as pointed out in #7904

#14 - 01/28/2019 02:54 PM - Andreas Kohlbecker

- Status changed from Resolved to Closed

- % Done changed from 50 to 100

the symptom as described in this issue is no longer reproducible.

Files

stacktrace.png	99.8 KB	11/09/2018	Andreas Kohlbecker
mem-diff-2.png	47.2 KB	11/09/2018	Andreas Kohlbecker
mem-diff-1.png	52.7 KB	11/09/2018	Andreas Kohlbecker
mem-diff-1-2.png	45.7 KB	11/09/2018	Andreas Kohlbecker
Allocations-of-Teams.png	92.9 KB	11/12/2018	Andreas Kohlbecker
mem-getCollectorAndFieldNumber.png	33.5 KB	11/12/2018	Andreas Kohlbecker
mem-getCollectorAndFieldNumberXX.png	20.1 KB	11/12/2018	Andreas Kohlbecker