Removing objects from Java Collections

Asked 15 years, 11 months ago Modified 15 years, 11 months ago

Viewed 3k times

0

I have a `HashMap` (although I guess this question applies to other collections) of objects. From what I understand, when the documentation talks about removing mappings, then it is removing the entry from the hashtable, i.e. not necessarily destroying the actual object. If the only remaining reference to the object is in this table, then will the object get garbage collected?

If I do `map.clear()` and those objects that were in the table are not referenced anywhere else, will they get garbage collected?

What is the fastest way, to actually remove all entries from the table, but also destroy those objects.

java collections

Share

Improve this question

Follow

asked Jan 13, 2009 at 16:42

Miles D

8,030 ● 5 ● 35 ● 36

6 Answers

Sorted by: Highest score (default)

9

Yes, if the collection is the last place these objects are referenced they are eligible for garbage collection after they have been removed from the collection. And no, you can not destroy these objects forcefully. The garbage collector will handle them when it feels like it.

Share Improve this answer

Follow

answered Jan 13, 2009 at 16:43

Bombe

83.7k • 20 • 127 • 127

2

> If the only remaining reference to the object is in this table, then will the object get garbage collected?

If there are no other references to an object, then the object will be garbage collected sometime in the future.

You should not have to force destruction of the objects. If they are extremely heavyweight objects (or you have too many objects to fit in memory), this points to a more fundamental problem with your code.

If you really must, then you can call `System.gc()`, although this is not good practice, and will always be a bellwether of underlying problems in your code.

Share Improve this answer Follow

answered Jan 13, 2009 at 16:48

James Van Huis

5,561 ● 1 ● 28 ● 25

2

Generally speaking, you have no strong control over when an object is specifically destroyed. Any object is eligible for garbage collection when there are no more (strong) reference to it - but there are no guarantees about when it will be garbage collected or in fact if it ever will be. Even calling System.gc() or Runtime.gc() provides no guarantees about actually doing anything, it's merely a hint to the JVM that it might want to consider garbage collecting now. I believe the only guarantee you get is that if an OutOfMemoryError is thrown, all potential garbage collections were done before the error was thrown.

There are implications here for handling sensitive information such as passwords. Since Strings cannot be programatically cleared, you ideally don't want to store the password as such. If you instead store it as an array of characters, you can then use `Arrays.fill(' ')` to overwrite the password and guarantee it is no longer resident in memory from that point.

Back back on topic - you are right that both operations will make the object eligible for garbage collection if it is not being referenced elsewhere. Collection.clear() is indeed the fastest way to drop references to all the objects in a collection at once.

Share Improve this answer

answered Jan 13, 2009 at 16:52

Follow

Andrzej Doyle
104k ● 33 ● 191 ● 231

Overwriting date does not guarantee that it is no longer in data. The JVM often moves data about, so perhaps the old value is still present. Worse swapping or hibernation may leave it on the disc, even after your machine has been rebooted many times. – Tom Hawtin - tackline Jan 13, 2009 at 17:03

Mmm, good point. It does at least mean that you can programmatically delete your variable, even if copies/caches of it may still be held by the OS (which, until you can flag memory as sensitive, is unavoidable regardless of the technique). At least this way your password will not be interned. – Andrzej Doyle Jan 13, 2009 at 17:52

2

Note that WeakHashMap allows you to place objects in it and have them eligible for garbage collection as soon as there are no more references to the key (not the value) outside the map - the map entry will disappear at this point.

In general you should not worry about *when* objects are garbage collected - the JVM decides this, and it knows a lot more about its memory needs and possible delays than you. What you *should* worry about is to make sure that objects you don't need anymore are *eligible* for garbage collection.

Share Improve this answer

Follow

edited Jan 13, 2009 at 17:10

answered Jan 13, 2009 at 16:48

Michael Borgwardt

346k ● 80 ● 486 ● 723

Do you mean "eligible" for garbage collection? :) – MetroidFan2002 Jan 13, 2009 at 16:57

MetroidFan2002, ve r prgrmurz, we r'nt suppozed to bee able to spell. ;) – WolfmanDragon Jan 13, 2009 at 17:00

Well, whaddayaknow: I actually had "egligible" fixed in my mind as the correct spelling - thanks! – Michael Borgwardt Jan 13, 2009 at 17:12

1

To have something truly garbage collected there can be no Strong references to the object. Objects with weakReference's may be garbage collected. Use WeakHashMap to make sure they are garbage collected, as that in a HashMap there is still references to the object.

Share Improve this answer

Follow

edited Jan 13, 2009 at 17:05

answered Jan 13, 2009 at 16:50

WolfmanDragon

7,952 ● 14 ● 51 ● 61

You can initiate a call to System.gc() after you clearyour map, but it's generally not a good idea.

0

Share Improve this answer

Follow

answered Jan 13, 2009 at 16:46

Omry Yadan

33.5k ● 19 ● 71 ● 89

-1 for System.gc(), +1 for pointing out it’s not a good idea. :)
– Bombe Jan 13, 2009 at 16:50