Java G1GC - Card Table (CT) vs Remembered Set (RS)

Java G1GC - 卡片表 （CT） 与记忆集 （RS）

Asked 4 years ago Modified 9 months ago Viewed 2k times

3

Why g1 needs both of these data structures ?

为什么 g1 需要这两种数据结构?

My understanding is: 我的理解是：

1. CT holds the information about references' actual location in old generation.

   CT 保存有关引用在老一代中实际位置的信息。

2. RS is specific to each region, each region has one RS associated with it, it stores info about outside references which points to objects in this region. So while navigating RS, actual location of reference can be found using CT.

   RS 特定于每个区域，每个区域都有一个与之关联的 RS，它存储有关指向该区域中对象的外部引用的信息。因此，在导航 RS 时，可以使用 CT 找到参考的实际位置。

Why can't RS hold all of the information instead of using CT ?

为什么 RS 不能保存所有信息而不是使用 CT ?

Is it because otherwise there would be too much of duplicate data stored in different RSs ? For example - Young generation has regions A and B, objects in both these regions have same outside reference from old generation. In this case both RSs associated with region A and B will store this information which is redundant, is this explanation correct ?

是因为否则不同的 RS 中会存储太多的重复数据吗？例如，新生代具有区域 A 和 B，这两个区域中的对象与老代具有相同的外部引用。在这种情况下，与区域 A 和 B 关联的 RS 都将存储此信息，这是多余的，这种解释正确吗?

java garbage-collection jvm g1gc

Share Improve this question Follow

asked Dec 28, 2020 at 9:32
user10916892
977 ●17 ●43

2 Answers

Sorted by: Highest score (default)

7

Let's first put some things in correct order. A `Card Table` shows were there *might* be incoming references into this region. It does sort of a "hash". For a single byte in the card table - there are *many* bytes in the old region. It's like saying that if (in theory) you have a card table that looks like this (a single card is marked as "dirty")

让我们首先把一些事情按正确的顺序排列。`Card Table` 显示*是否有*传入引用进入此区域。它确实有点像 “哈希”。对于 card 表中的单个字节 - 旧区域中*有许多*字节。这就像说，如果（理论上）你有一个看起来像这样的牌桌（一张牌被标记为 “dirty”）

```
0 1 0 0 0 0
```

For that `1` (dirty card) there is a certain mapping in the old region that needs to be scanned also, when scanning the young regions.

对于那 `1` （脏卡），在扫描年轻区域时，也需要扫描旧区域中的某个映射。

```
   0          1      0    .....

 0 - 512   512-1024  ...........
```

So that dirty card corresponds to a certain portion (from 512 to 1024 bytes) in the old generation, that will be scanned also, as part of the young generation scanning.

因此，该脏卡对应于老一代中的某个部分（从 512 到 1024 字节），作为新生代扫描的一部分，该部分也将被扫描。

G1 has regions and now you need to understand how `CT` and `RS` work in tandem. Let's say GC scans `Region1` at this point in time, it takes everything that is alive from this region and copies into `Region2`. At the same time `Region2` has references to `Region3`. If we use `CT`, `Region2` will be marked as "dirty" (by placing the specific card in the card table).

G1 有区域，现在您需要了解 `CT` 和 `RS` 如何协同工作。假设 GC 此时扫描 `Region1`，它会从该区域获取所有存活的内容并复制到 `Region2` 中。同时，`Region2` 引用了 `Region3`。如果我们使用 `CT，Region2` 将被标记为 “dirty” （通过将特定卡放在 card 表中） 。

Next cycle wants to scan `Region3`. How does `Region3` knows if there are *other* regions that might point into it? And indeed there are such cases: `Region2` has references into `Region3`. Well, it could look into the `CT` and check every single dirty card (and thus every region that these dirty cards corresponds to) and see if there are references from any of those regions in here. Think about it: in order to clear `Region3`, `G1` has to look at the *entire* `CT`. In the worst case scenario, it should be scanning the entire heap - for a single region.

下一个周期想要扫描 `Region3`。`Region3` 如何知道是否有*其他*区域可能指向它？确实存在这样的情况：`Region2` 引用了 `Region3`。嗯，它可以查看 `CT` 并检查每张脏卡（以及这些脏卡对应的每个区域），并查看此处是否有来自这些区域的引用。想想看：为了清除 `Region3，G1` 必须查看*整个* `CT。` 在最坏的情况下，它应该扫描整个堆 - 针对单个区域。

Thus: `Remembered Sets`. These data structures are populated based on what `CT` knows about, by an asynchronous thread. When `Region2` is marked as dirty, an asynchronous thread will start computing its `RS`. When `Region3` needs to be scanned, its `RS` will only have an entry with `Region2`.

因此：`Remembered Sets`。这些数据结构是根据 `CT` 所了解的，由异步线程填充。当 `Region2` 被标记为 dirty 时，异步线程将开始计算其 `RS`。当需要扫描 `Region3` 时，其 `RS` 将仅包含带有 `Region2` 的条目。

Thus, in order to scan a single region, `G1` *only* needs to look into that particular `RS`.

因此，为了扫描单个区域，`G1` *只需要*查看该特定 `RS`。

Share Improve this answer Follow

answered Dec 28, 2020 at 20:08
Eugene
121k ● 17 ● 210 ● 328

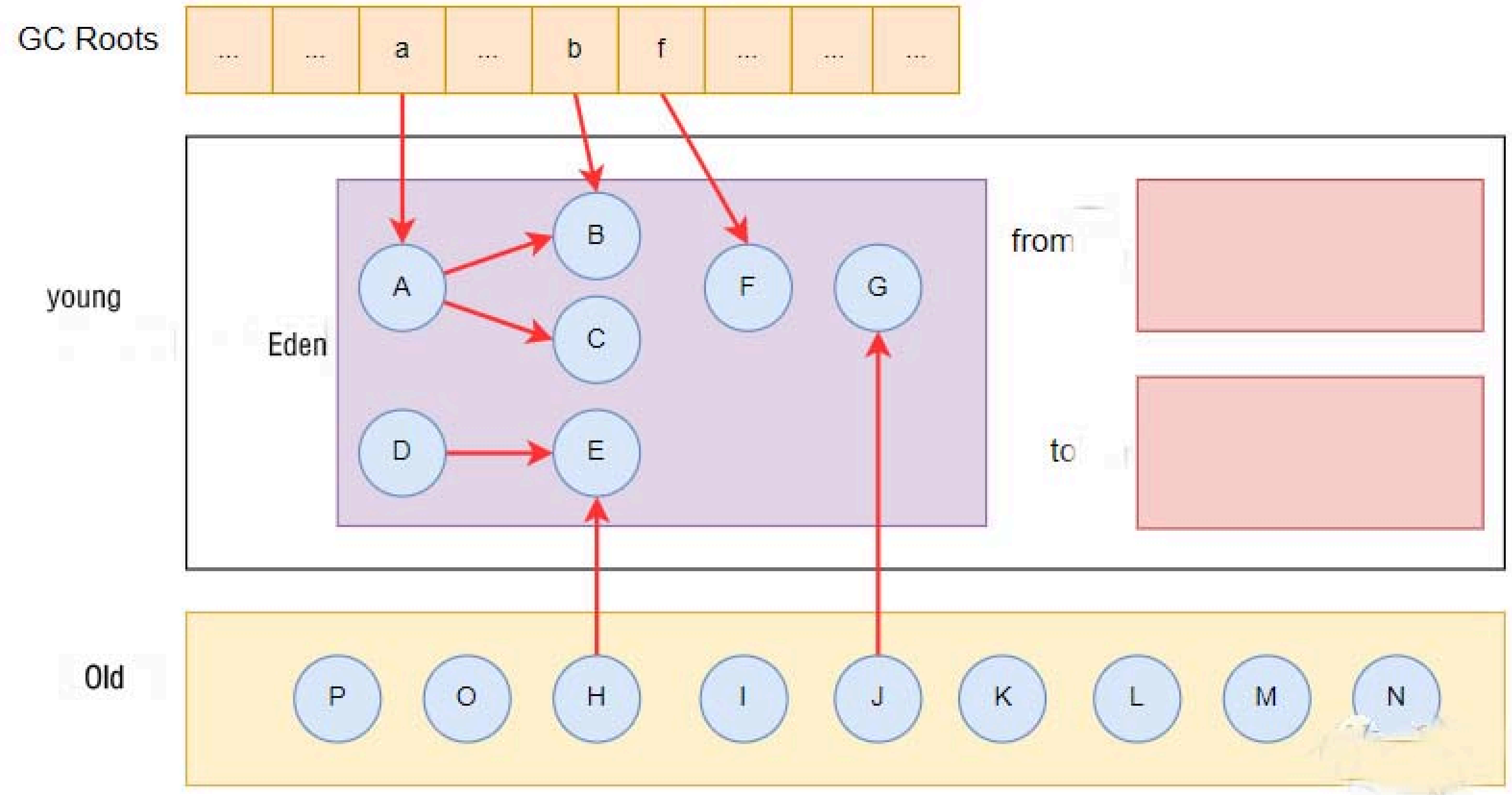
Thanks, this is kind of what i was looking for. Quick question - So when does RS of Region3 gets updated/populated, is it when Region2 is marked dirty ?
谢谢，这就是我一直在找的。快速问题 - 那么 Region3 的 RS 何时更新/填充，是 Region2 被标记为脏的时候吗？ – user10916892 Dec 29, 2020 at 11:33

@user10916892 I do not remember that, and this might have changed in recent versions. What is important, is that it does.

@user10916892我不记得了，这在最近的版本中可能已经改变了。重要的是，它确实如此。 – Eugene Dec 29, 2020 at 13:03

4

Let's look at the below picture:



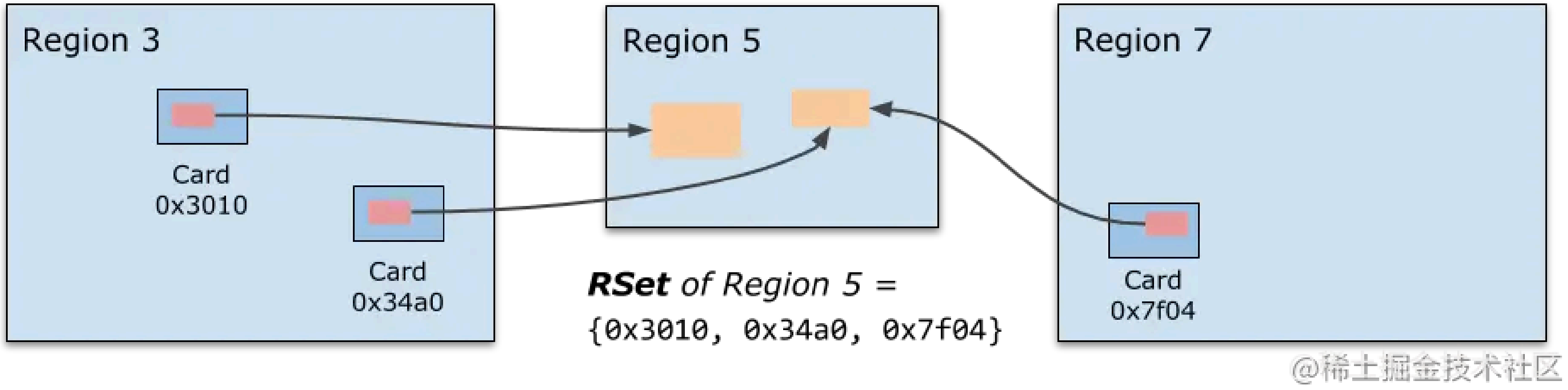
让我们看看下图：

During the young GC, the E and G objects are not reachable from GC Roots. So they might be remarked as 'dead' objects. If that happened, it will cause unpredictable problem.

在年轻的 GC 期间，无法从 GC 根访问 E 和 G 对象。因此，它们可能会被标记为“死”对象。如果发生这种情况，将导致不可预测的问题。

Considering scanning the whole heap is too time-consuming, JVM use rememberSet (RSet) to store all the locations which has a pointers to objects within this region. The follow-up image show a example RSet of Region 5.

考虑到扫描整个堆太耗时，JVM 使用 rememberSet （RSet） 来存储所有具有指向该区域内对象的指针的位置。后续图像显示了区域 5 的示例 RSet。



There're different implemtations dependig on various granularity of the RSet. For simplicity, if the RSet just record the reference Region index. RSet of Region 5 = {Region7, Region3}.

根据 RSet 的各种粒度，有不同的实现。为简单起见，如果 RSet 只记录引用 Region 索引。区域 5 的 RSet = {Region7， Region3}。

To check object liveness, GC is still need to scan two entire Region. The easiest optimization should be that GC divide a Region into serveral small areas/pages. Every areas has a flag in card table(CT). Assuming the area contains objects pointer to outside region, the corresponding flag is marked as *dirty* value.

要检查对象活动性，GC 仍然需要扫描两个完整的 Region。最简单的优化应该是 GC 将一个 Region 划分为几个小区域/页面。每个区域在卡片表中都有一个标志 （CT） 。假设该区域包含指向外部区域的对象指针，则相应的标志将标记为 *dirty* 值。

So card table is used to indicate whether it contains a pointer from old generation to young generation. It's particular type of RSet.

所以 card table 用于指示它是否包含从老一代到年轻一代的指针。这是 RSet 的特殊类型。

Share Improve this answer Follow edited Apr 6, 2024 at 9:09 answered May 1, 2023 at 7:12

dknight
679 ● 6 ● 7

Thank you. This was very helpful. In my opinion, this should have been marked as the answer.
谢谢。这非常有帮助。在我看来，这应该被标记为答案。 – Sony Antony Nov 21, 2023 at 7:32