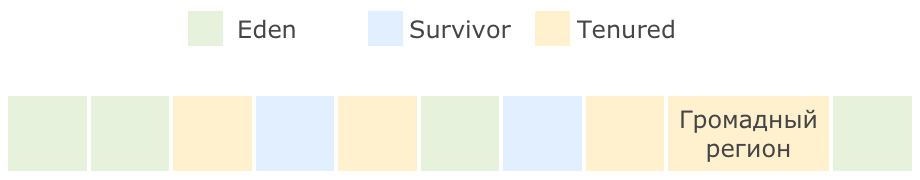
***25. Характеристики алгоритма G1. (Garbage First)***

Взято остюда:

<https://habrahabr.ru/post/269863/>

Здесь память разбивается на множество регионов одинакового размера. Размер этих регионов зависит от общего размера кучи и по умолчанию выбирается так, чтобы их было не больше 2048, обычно получается от 1 до 32 МБ. Исключение составляют только так называемые громадные (humongous) регионы, которые создаются объединением обычных регионов для размещения очень больших объектов.

Разделение регионов на Eden, Survivor и Tenured в данном случае логическое, регионы одного поколения не обязаны идти подряд и даже могут менять свою принадлежность к тому или иному поколению. Пример разделения кучи на регионы может выглядеть следующим образом (количество регионов сильно приуменьшено):

Малые сборки выполняются периодически для очистки младшего поколения и переноса объектов в регионы Survivor, либо их повышения до старшего поколения с переносом в Tenured. Над переносом объектов трудятся несколько потоков, и на время этого процесса работа основного приложения останавливается. Это уже знакомый нам подход из рассмотренных ранее сборщиков, но отличие состоит в том, что очистка выполняется не на всем поколении, а только на части регионов, которые сборщик сможет очистить не превышая желаемого времени. При этом он выбирает для очистки те регионы, в которых, по его мнению, скопилось наибольшее количество мусора и очистка которых принесет наибольший результат. Отсюда как раз название Garbage First — мусор в первую очередь.

А с полной сборкой (точнее, здесь она называется смешанной (mixed)) все немного хитроумнее, чем в рассмотренных ранее сборщиках. В G1 существует процесс, называемый циклом пометки (marking cycle), который работает параллельно с основным приложением и составляет список живых объектов. За исключением последнего пункта, этот процесс выглядит уже знакомо для нас:

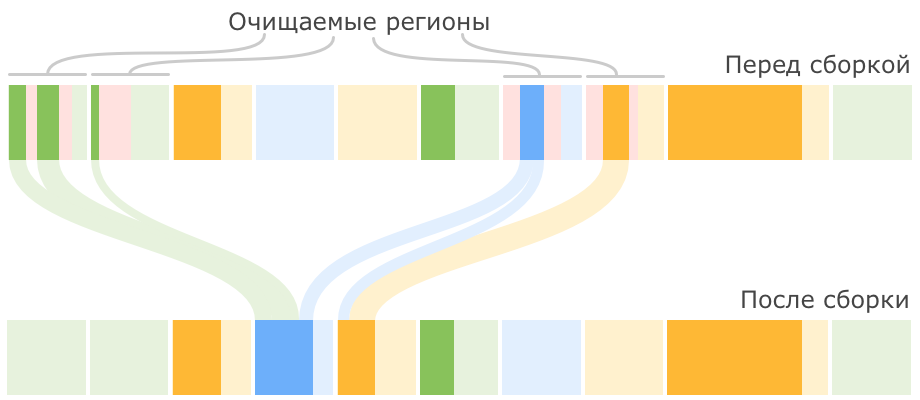
* Initial mark. Пометка корней (с остановкой основного приложения) с использованием информации, полученной из малых сборок.
* Concurrent marking. Пометка всех живых объектов в куче в нескольких потоках, параллельно с работой основного приложения.
* Remark. Дополнительный поиск не учтенных ранее живых объектов (с остановкой основного приложения).
* Cleanup. Очистка вспомогательных структур учета ссылок на объекты и поиск пустых регионов, которые уже можно использовать для размещения новых объектов. Первая часть этого шага выполняется при остановленном основном приложении.

Следует иметь в виду, что для получения списка живых объектов G1 использует алгоритм Snapshot-At-The-Beginning (SATB), то есть в список живых попадают все объекты, которые были таковыми на момент начала работы алгоритма, плюс все объекты, созданные за время его выполнения. Это, в частности, означает, что G1 допускает наличие плавающего мусора, с которым мы познакомились при рассмотрении сборщика CMS.

После окончания цикла пометки G1 переключается на выполнение смешанных сборок. Это значит, что при каждой сборке к набору регионов младшего поколения, подлежащих очистке, добавляется некоторое количество регионов старшего поколения. Количество таких сборок и количество очищаемых регионов старшего поколения выбирается исходя из имеющейся у сборщика статистики о предыдущих сборках таким образом, чтобы не выходить за требуемое время сборки. Как только сборщик очистил достаточно памяти, он переключается обратно в режим малых сборок.

Очередной цикл пометки и, как следствие, очередные смешанные сборки будут запущены тогда, когда заполненность кучи превысит определенный порог.

Смешанная сборка мусора в приведенном выше примере кучи может пройти вот так:

 Может оказаться так, что в процессе очистки памяти в куче не остается свободных регионов, в которые можно было бы копировать выжившие объекты. Это приводит к возникновению ситуации allocation (evacuation) failure, подобие которой мы видели в CMS. В таком случае сборщик выполняет полную сборку мусора по всей куче при остановленных основных потоках приложения.

Опираясь на уже упомянутую статистику о предыдущих сборках, G1 может менять количество регионов, закрепленных за определенным поколением, для оптимизации будущих сборок.

Гиганты

В начале рассказа о G1 я упомянул о существовании громадных регионов, в которых хранятся так называемые громадные объекты (humongous objects). С точки зрения JVM любой объект размером больше половины региона считается громадным и обрабатывается специальным образом:

* Он никогда не перемещается между регионами.
* Он может удаляться в рамках цикла пометки или полной сборки мусора.
* В регион, занятый громадным объектом, больше никого не подселяют, даже если в нем остается свободное место.

Вообще, эти пункты иногда имеют далеко идущие последствия. Объекты большого размера, особенно короткоживущие, могут доставлять много неудобств всем типам сборщиков, так как не удаляются при малых сборках, а занимают драгоценное пространство в регионах старшего поколения. Но G1 оказывается более уязвимым к их негативному влиянию в силу того, что для него даже объект в несколько мегабайт (а в некоторых случаях и 500 КБ) уже является громадным. В комментарии к предыдущей статье как раз приводится пример такой проблемы у Solr.

В продолжении данного цикла статей мы посмотрим, как с этим можно бороться.

Ситуации STW

Если резюмировать, то у G1 мы получаем STW в следующих случаях:

* Процессы переноса объектов между поколениями. Для минимизации таких пауз G1 использует несколько потоков.
* Короткая фаза начальной пометки корней в рамках цикла пометки.
* Более длинная пауза в конце фазы remark и в начале фазы cleanup цикла пометки.