Введение

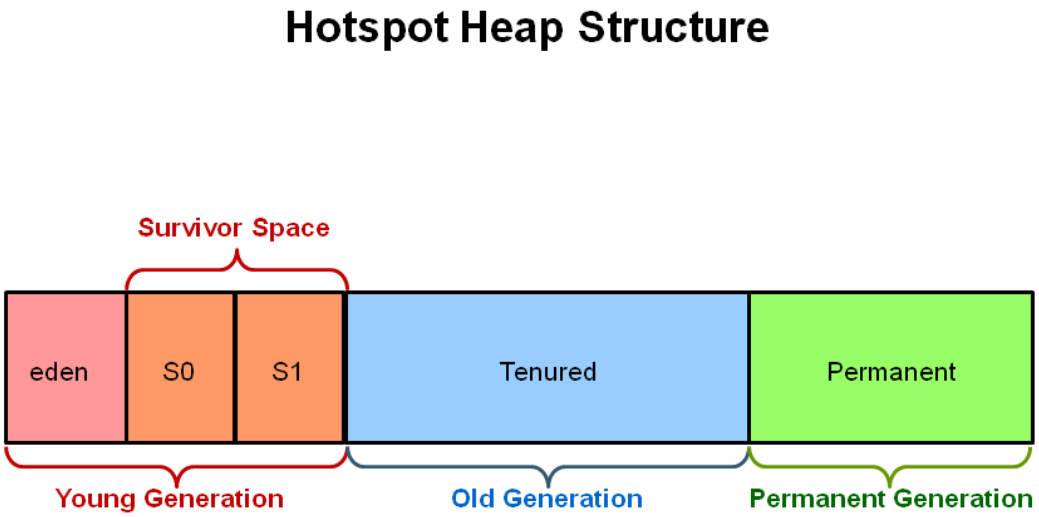
Коллектор Garbage-First (G1) - это конкурентный (фоновый, одновременный) и параллельный сборщик мусора. Он с высокой вероятностью удовлетворяет требованиям по времени паузы сборки мусора (GC), достигая при этом высокой пропускной способности. Можно конфигурировать количество и время пауз за промежуток времени. Это называется “мягким рантаймом”, так как G1 старается уложиться в эти ограничения, но не гарантирует их.

В первую очередь, он предназначен для приложений, которые:

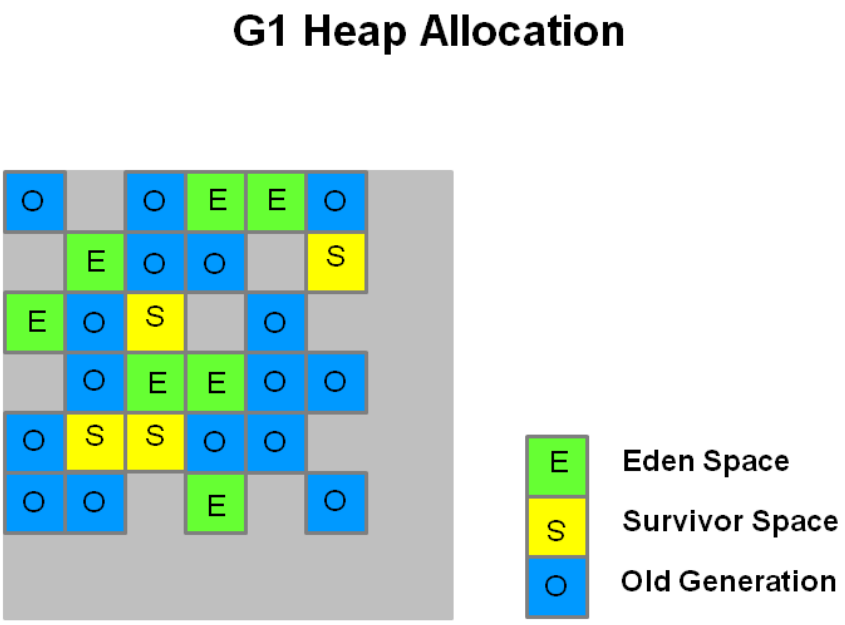
- могут работать одновременно с потоками приложений
- нужно компактное свободное пространство без длительных пауз, вызываемых GC.
- нуждаются в более предсказуемых длительностях пауз GC.
- не нужно жертвовать большой пропускной способностью.

Одно из ключевых особенностей заключается в том, что G1 - это уплотняющий коллектор без длительных пауз. G1 достаточно компактен и для этого полагается на регионы. Это значительно упрощает отдельные части сборщика и в основном устраняет потенциальные проблемы фрагментации. Кроме того, G1 предлагает более предсказуемые паузы в сборке мусора, чем например сборщик CMS, и позволяет пользователям указывать желаемые цели паузы.

Все старые сборщики мусора (последовательный, параллельный, CMS) структурируют кучу на три секции: молодое поколение, старое поколение и постоянное поколение фиксированного объема памяти.
Все объекты памяти оказываются в одной из этих трех секций.



Коллектор G1 использует другой подход.



Куча разбивается на множество одинаковых по размеру регионов кучи, каждый из которых представляет собой непрерывный диапазон виртуальной памяти. Определенным наборам регионов присваиваются те же роли (eden, survivor, old), что и в старых коллекторах, но для них не устанавливается фиксированный размер. Это обеспечивает большую гибкость в использовании памяти.

Увеличивается и потребление памяти. По сравнению с коллекторами ParallelOldGC или CMS на G1 будет больший размер процесса JVM. При этом влияние на размер кучи составляет менее 5%.

При выполнении сборки мусора G1 выполняет параллельную фазу глобальной маркировки для определения "живости" объектов по всей куче. Это в значительной степени связано с "учетными" структурами данных, такими как Remembered Sets и Collection Sets.

Remembered Sets или RSets отслеживают ссылки на объекты в каждом регионе. На каждый регион в куче приходится один RSet. RSet позволяет параллельно и независимо собирать регион. RSets знает, какие регионы в основном состоят из мертвых объектов. G1 собирает мусор сначала в этих регионах, что обычно позволяет получить большое количество свободного места. Именно поэтому этот метод сборки мусора называется Garbage-First. Как

следует из названия, G1 концентрирует свою деятельность по сбору и уплотнению на тех областях кучи, которые, скорее всего, будут заполнены мусором. G1 использует модель прогнозирования паузы для достижения заданного времени паузы и выбирает количество областей для сбора на основе этого.

Collection Sets или CSet - набор регионов, которые будут собраны в GC. Все живые данные в наборе CSet эвакуируются (копируются/перемещаются) во время GC. Наборы регионов могут быть eden, survivor и/или old. Наборы CSets оказывают менее 1% влияния на размер JVM.

Из регионов, определенных G1, собирается мусор и производится эвакуация. То есть G1 копирует объекты из одного или нескольких регионов кучи в один регион кучи, при этом происходит уплотнение и освобождение памяти. Эта эвакуация выполняется параллельно на нескольких процессорах, чтобы уменьшить время паузы и увеличить пропускную способность. Таким образом, при каждой сборке мусора G1 непрерывно работает над уменьшением фрагментации, работая в пределах заданного пользователем времени паузы. Это выходит за рамки возможностей обоих предыдущих ГЦ. Сборщик мусора CMS (Concurrent Mark Sweep) не выполняет уплотнение. Сборщик мусора ParallelOld выполняет уплотнение только всей кучи, что приводит к значительным паузам.

Важно отметить, что G1 не является коллектором реального времени. Он выполняет заданное время паузы с высокой вероятностью, но не с абсолютной уверенностью. Основываясь на данных предыдущих сборов, G1 делает оценку того, сколько регионов может быть собрано в течение заданного целевого времени. Таким образом, у сборщика есть достаточно точная модель стоимости сбора регионов, и он использует эту модель для определения того, какие и сколько регионов нужно собрать, оставаясь в пределах заданного времени паузы.

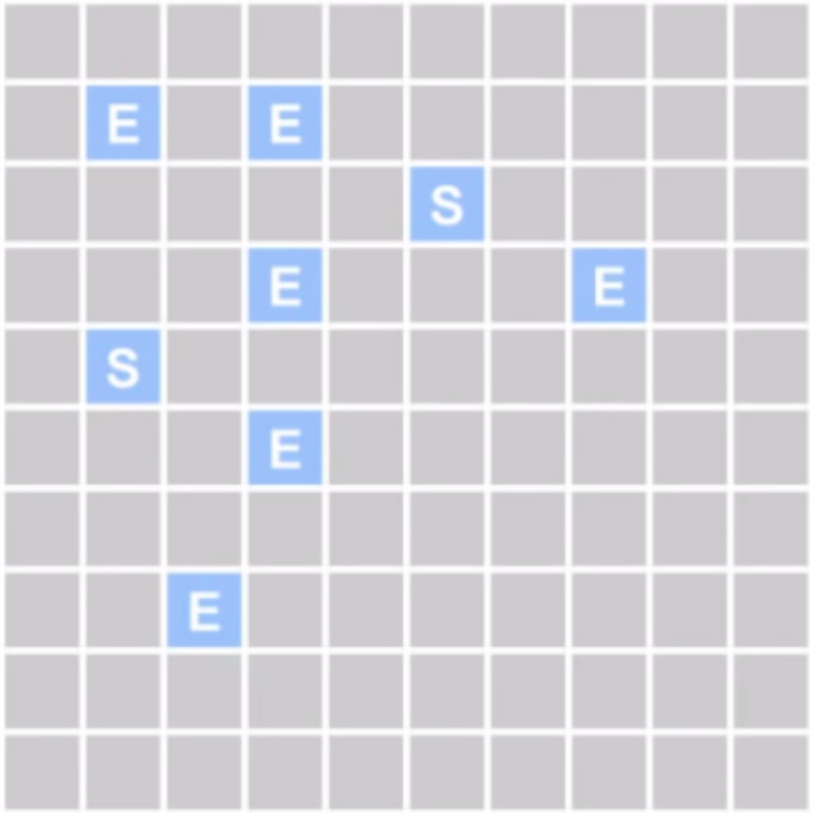
Примечание: G1 имеет как конкурентную (выполняется вместе с потоками приложения, например, уточнение, маркировка, очистка), так и параллельную (многопоточную, например, остановка мира) фазы. Полная сборка мусора по-прежнему выполняется в один поток, но при правильной настройке ваши приложения должны избегать полной GC.

Рассмотрим подробнее устройство G1

Куча

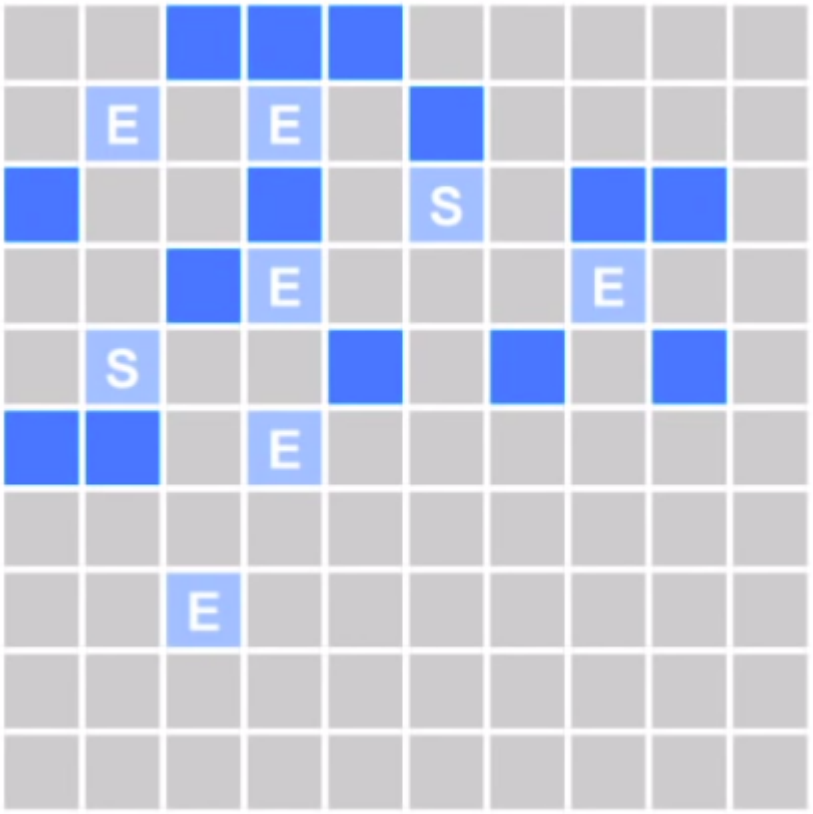
Куча - это одна область памяти, разбитая на множество областей фиксированного размера.
Размер региона выбирается JVM при запуске. Обычно JVM выбирает около 2000 регионов, размер которых варьируется от 1 до 32 Мб.

Молодое поколение



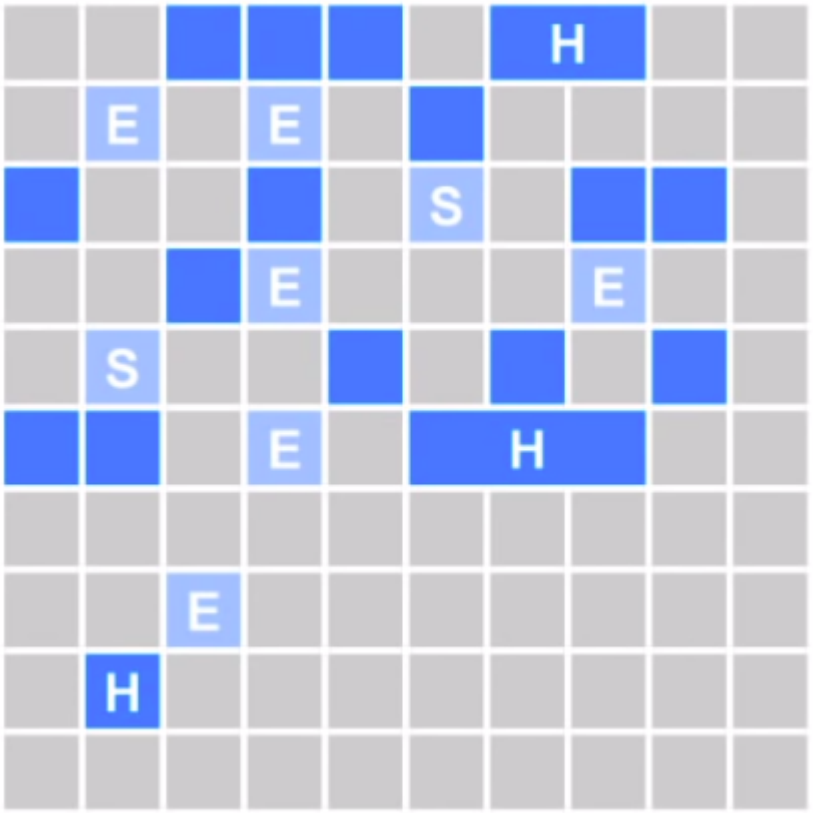
Молодые регионы отображаются в логические представления пространств eden, survivor. Это динамически выбираемые набор регионов, он меняется от одной сборки к другой.
Eden - место аллокации объектов.
Survivor - объекты, пережившие хотя бы одну сборку.

Старое поколение



Old - объекты, которые пережили много сборок мусора (тоже параметр ГЦ), выбираются динамически.

Большие объекты

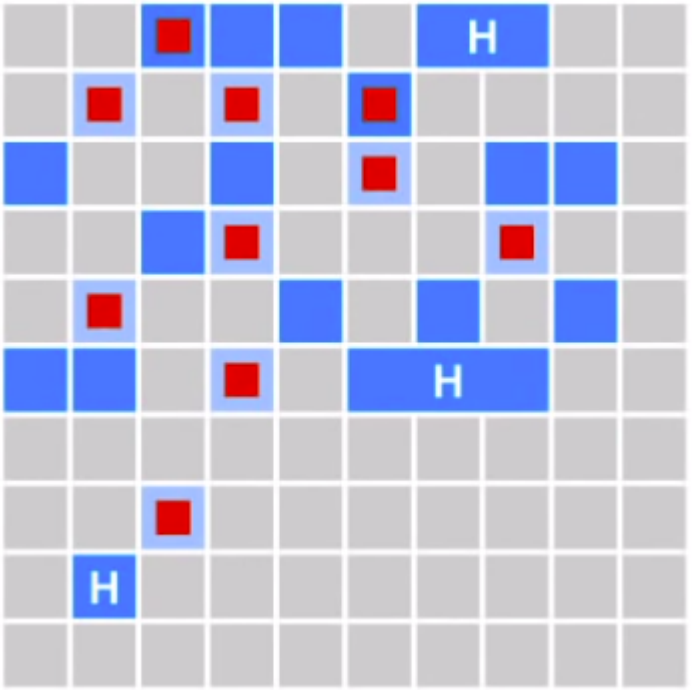


Существует тип регионов, известный как humongous. Они предназначены для хранения одноименных объектов, размер которых больше одного региона (1 - 32 МБ). Тогда они хранятся как набор смежных регионов. ГЦ работает с ними по-особенному, что тормозит его работу. Поэтому следует избегать создания объектов такого размера.

Пустые регионы

Наконец, последний тип регионов - это неиспользуемые области кучи.

Collection sets

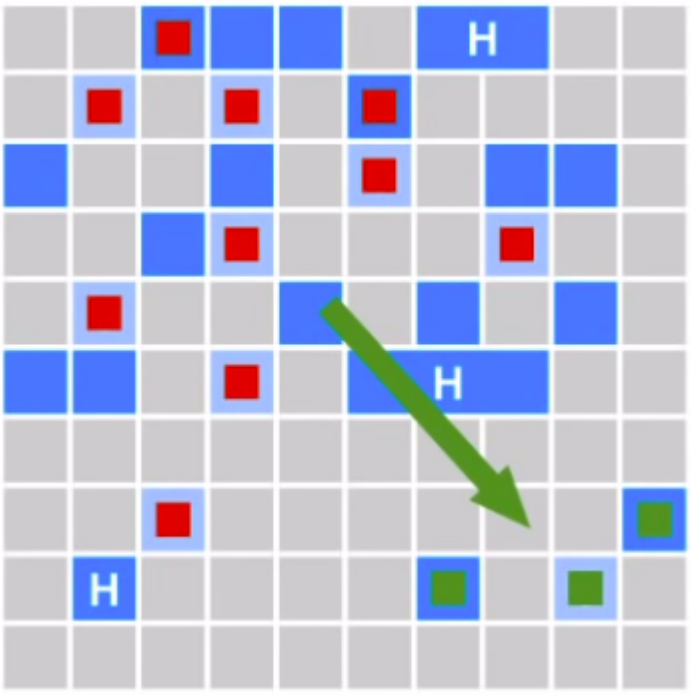


При сборке мусора выбираются регионы, в которых она будет происходить. Collection set должен быть существенно меньше, чем общий размер heap для удовлетворения времени паузы. Иначе мы уходим в full GC и время пауз не соответствует требованиям.

В collection set всегда попадает все молодое поколение и возможно часть старого поколения. В этом проявляется инкрементальность.

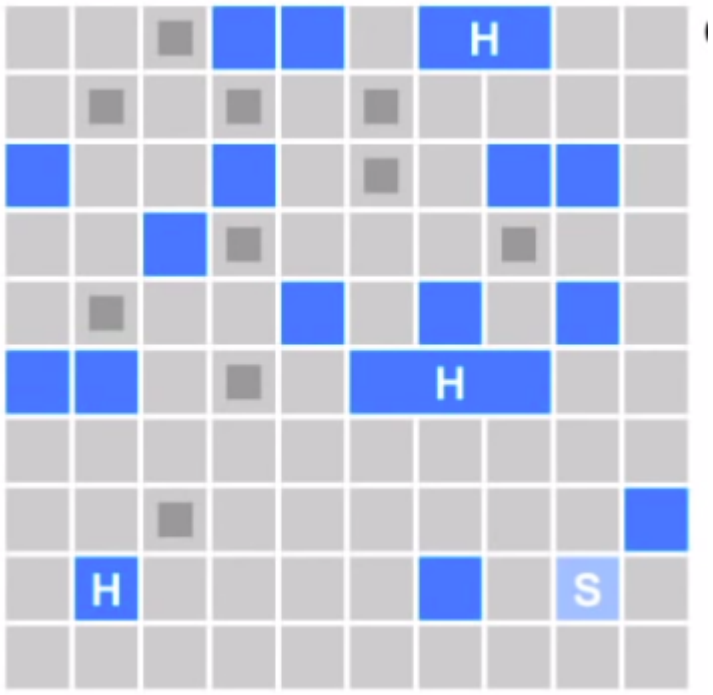
Поэтому выделяют три вида сборки мусора G1: молодая (young) сборка (только молодые регионы), смешанная (mixed) (молодые + некоторые старые) и полная (full) (когда collection set = heap)

Сборка мусора для молодого и старшего поколения



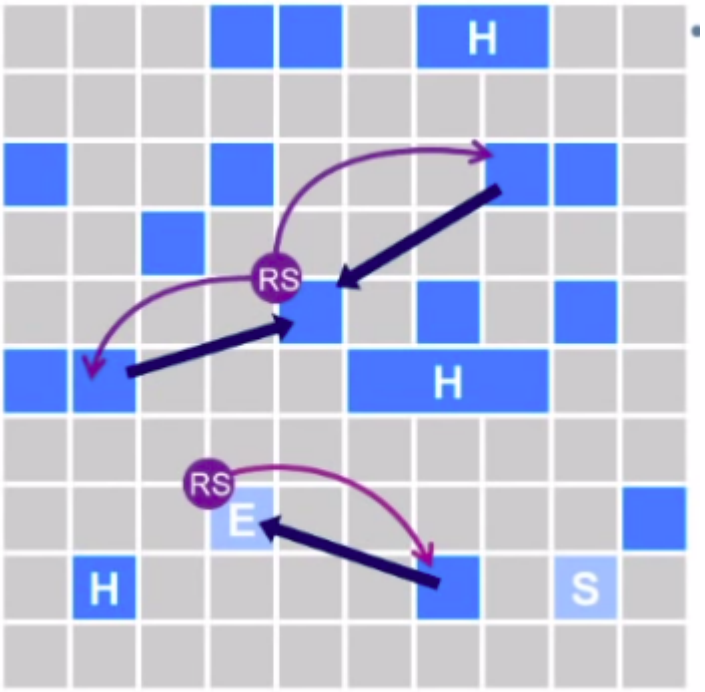
Молодая и смешанная сборка мусора происходят с помощью копирования живых объектов из collection set в пустые регионы. Collection set в данном случае и есть FROM-пространство, а новые регионы составляют часть TO-пространства. Так появляются survivor и old регионы, которые пережили некоторое число сборок мусора.

Освобождение памяти



Когда все живые объекты эвакуированы, G1 очищает регионы из collection set. Для эффективной работы, объем освобожденного места должен быть больше, чем скопированных живых объектов. В этом и проявляется уплотнение.

Remembered sets



Чтобы иметь возможность независимо собирать регионы и распараллелить эту задачу, поддерживаются Remembered Sets. Это хеш-таблицы с информацией о регионах, которые ссылаются на объекты из данного региона. Они создаются только между регионами молодого и старого поколения, а также внутри старого поколения.

Идея в том, что когда мы хотим определить, какие объекты достижимы в данном регионе, мы просматриваем только Remembered Set, а не весь heap.

А теперь вопрос: почему мы не храним ссылки в RSets в молодом поколении?

- Так как каждая сборка мусора гарантированно собирает молодые объекты. Это следствие слабой гипотезы о поколениях.

Барьер на запись

Чтобы поддерживать RSets в консистентном состоянии, G1 широко использует барьер на запись. Это дополнительный код, исполняемый при изменении значения поля объекта с указателем. К примеру, в G1 он используется когда нужно обновить ссылки в RSets на объекты, местоположение которых изменилось в ходе сборки мусора. Вот пример барьера из реализации RSets.

```
a.f = x;
```

```
a.f = x;
card_table[index_for(&a.f)] = DIRTY_VALUE;
```

Типы барьеров G1

Для корректной работы, G1 требует два вида барьеров: pre- и post-barrier.

Pre-barrier выполняется до изменения указателя, он сохраняет старое значение указателя. Это необходимо для правильной работы фоновой маркировки, которая выполняется по принципу SATB.

SATB строит полный snapshot или view heap на момент начала фоновой (конкурентной) маркировки. Таким образом, приложение продолжает работать, его граф объектов меняется, а мы пока определяем достижимости по слепку.

Post-barrier используется когда нужно обновить ссылки в RSets на объекты, местоположение которых изменилось в ходе сборки мусора. То есть они

работают для ссылок в RSets между регионами старшего поколения, а также между старшими и молодыми регионами.

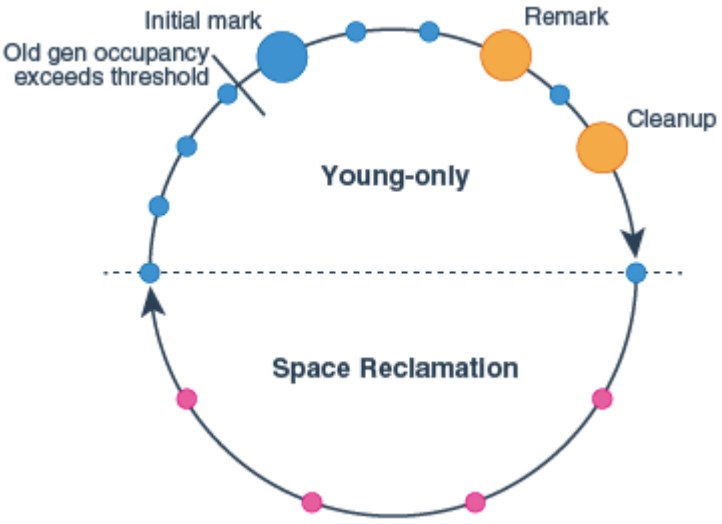
Фоновая маркировка

Чтобы выбирать регионы старшего поколения для сборки наиболее эффективно, рассчитывая цель времени паузы, в G1 используется фоновая и параллельная маркировка кучи.

По достижении порогового значения занятости кучи, начинается в конкурентном формате маркировка всех живых объектов в куче по принципу SATB. При этом каждая хеш-таблица может обрабатываться параллельно. Затем обновляется информация по достижимости в регионах в RSets, удаляются регионы без живых объектов. Разбиваются циклические зависимости среди недостижимых объектов.

Весь алгоритм

Ещё раз для целостной картины поэтапно, но уже более абстрактно.



Фаза young-only начинается с нескольких сборок young-only, которые продвигают объекты в старое поколение. Переход между фазой "только для молодых" и фазой освобождения пространства начинается, когда занятость старого поколения достигает определенного порога Initiating Heap Occupancy. В это время G1 планирует фоновую маркировку вместо обычной young-only сборки.

Фоновая маркировка начинается в Stop-The-World (STW) режиме и быстро переходит в конкурентный и параллельный режим. Она может быть прервана young-only сборкой и SATB придется делать заново. Маркировка завершается двумя специальными паузами с остановкой мира: Remark и Cleanup.

Remark завершает маркировку живых объекта в куче с помощью STW. Этот этап также выполняется параллельно, чтобы минимизировать паузу. Как мы помним, snapshot-at-the-beginning (SATB) делает виртуальный снимок кучи во время паузы в начале фоновой маркировки. Это означает, что объекты, которые становятся мертвыми во время маркировки, все еще считаются живыми. Это может привести к ошибочному копированию и лишнему расходу памяти. Так вот Remark подчищает такие объекты и возвращает актуальные данные по достижимости в куче.

Cleanup сначала производит STW и затем выполняется в конкурентном режиме.

- В STW режиме он выполняет учет на живых объектов, полностью свободных областей и рассчитывает время паузы.
- В STW режиме он выполняет эвакуацию живых объектов в новые регионы. Это может быть сделано с регионами только молодого поколения или молодого и старого поколения вместе.
- В STW режиме очищает collection set.
- В конкурентном режиме сбрасывает пустые регионы и возвращает их в список свободных.

В качестве резервного варианта, если у приложения закончилась память во время сбора информации о достижимости, G1 выполняет в STW полную сборку и уплотнение кучи (Full GC), как и другие сборщики.