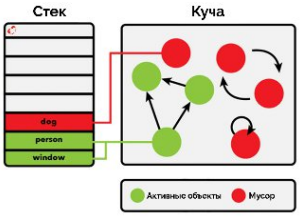
Что такое сборщик мусора? (Garbage collector)

В Java используется автоматическое управление памятью. Программист выделяет память, а за освобождение отвечает JVM. Когда программа больше не ссылается на объект (прямые или косвенные ссылки), то объект удаляется, а память пере-используется. **Сборщик мусора – это демон-поток,** который выполняет две задачи: **поиск и очистка мусора.**

Разбираемся на примере **HotSpot (**популярная реализация JVM). Все объекты, которые явно или неявно создаются Java-приложением, размещаются в куче.

Что такое мусор, как определить, что объекты мёртвые?
Для обнаружения мусора есть ДВА основных подхода: **учёт ссылок и трассировка.**

Учет ссылок (Reference counting).
Если объект не имеет ссылок, он считается мусором.
Проблема: невозможно выявить циклические ссылки (когда два объекта не имеют внешних ссылок, но ссылаются друг на друга -> приводит к утечке памяти)

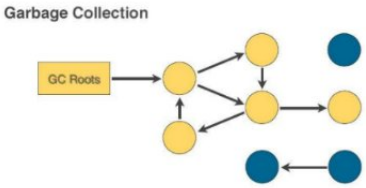


Трассировка (Tracing), используется в HotSpot (популярная разновидность JVM).

До объекта можно добраться из корневых точек (GC root).
Если до чего-то добраться нельзя, то это мусор.
Всё, что доступно из «живого» объекта, также является «живым».

Типы корневых точек **GC Roots** java приложения:

- объекты в статических полях классов
- объекты, доступные из стека потоков
- объекты из JNI (Java Native Interface) ссылок в native методах



Сборщики мусора бывают разные:

Java **HotSpot VM** предоставляет разработчикам на выбор СЕМЬ различных сборщика мусора:

Serial (последовательный) — самый простой вариант для приложений с небольшим объемом данных и не требовательных к задержкам. Редко когда используется, но на слабых компьютерах может быть выбран виртуальной машиной в качестве сборщика по умолчанию.

Parallel (параллельный) — наследует подходы к сборке от последовательного сборщика, но добавляет параллелизм в некоторые операции, а также возможности по автоматической подстройке под требуемые параметры производительности.

Concurrent Mark Sweep (CMS) — нацелен на снижение максимальных задержек путем выполнения части работ по сборке мусора параллельно с основными потоками приложения. Подходит для работы с относительно большими объемами данных в памяти. Использует инкрементальный алгоритм сборки (см. дальше).

Garbage-First (G1, см. ниже) — создан для постепенной замены CMS, особенно в серверных приложениях, работающих на многопроцессорных серверах и оперирующих большими объемами данных.

Epsilon GC — разработан для случаев, когда сборка мусора вообще не нужна.

Epsilon (2017, Алексей Шипилёв) выглядит как любой GC для OpenJDK, подключенный с помощью -XX:+UnlockExperimentalVMOptions -XX:+UseEpsilonGC.

Epsilon GC занимается только аллокацией памяти и ничего не делает для её освобождения.
При достижении лимита памяти виртуальная машина Java остановится.

Зачем? тестирование производительности, анализ накладных расходов других сборщиков мусора, облегчение разработки виртуальной машины.

ZGC — пытается удерживать паузы на субмиллисекундном уровне, даже при работе с очень большими кучами.

Shenandoah GC — еще один сборщик, нацеленный на ультракороткие паузы независимо от размера кучи.

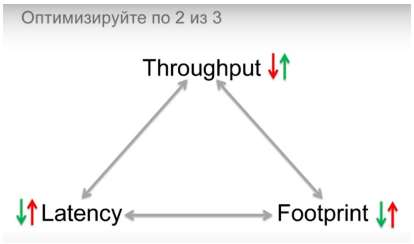
Как определяется производительность для сборщика мусора?

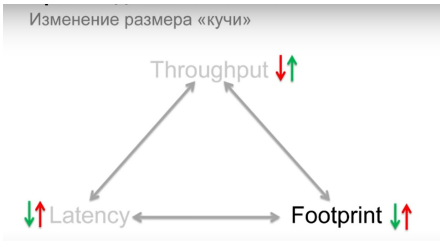
- **Throughput** (пропускная способность) – объём вычислительных ресурсов, затрачиваемых GC
- **Latency** (задержка) – на какое время прерывается работа приложения
- **Footprint** (след) – объём используемой памяти

Можно и нужно выбирать оптимальную стратегию, исходя из целей разработки.

Варианты:

- Оптимизируем 2 из 3 компонентов (от чего и во имя чего можем отказаться)
- Изменяем размер кучи: ячейки имеют размер от 1мб по дефолту до 32 мб (влияем)





Сборщик мусора выбирается в зависимости от цели и задач приложения.

Два подхода к сборке мусора, два алгоритма работы:

STW «stop-the-world» - остановка приложения на период уборки (остановка нужна, чтоб никакие новые изменения не произошли в программе, новые объекты не появились и т.д.)

- проще определять достижимость объектов «граф объектов заморожен»)
- проще перемещать объекты в куче (в процессе сборки куча может находиться в некорректном состоянии)

НО!

- приложение останавливается на время сборки мусора
- зависит от размера кучи (объёма памяти живых объектов)

Инкрементальная сборка

- Попытка уменьшить паузы, вызванные GC (за счёт большого количества коротких пауз и фоновой сборки)
- Требуется синхронизировать работу GC с приложением (барьеры на чтение/запись)

Такой алгоритм занимает больше времени.

Универсального «UBER» GC не существует =)

Красными границами обозначено **время остановки программы STW** «stop-the-world»

GC в Hotspot JVM

Взгляд извне

- **SerialGC**
  - последовательная сборка молодого и старого поколений
- **ParallelGC**
  - максимальный throughput
  - параллельная сборка молодого и старого поколений
- **CMS**
  - предсказуемость
  - по возможности, сборка мусора в фоновом режиме
- **G1**
  - предсказуемость
  - слабо подвержен фрагментации

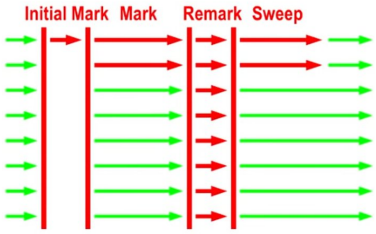
SerialGC

- Молодое поколение: последовательный копирующий GC
- Старшее поколение: последовательный Mark-Sweep-Compact
  - Аллокация: линейная
- -XX:+UseSerialGC

ParallelGC

- Молодое поколение: параллельный копирующий
- Старшее поколение: параллельный Mark-Com
  - Аллокация: линейная
- -XX:+UseParallelGC -XX:+UseParallelOldGC

CMS



- Молодое поколение: параллельный копирующий GC
- Старшее поколение: фоновый Mark-Sweep GC
  - Аллокация: free-листы
  - Компактификация только при FullGC
  - K:+UseConcMarkSweepGC

Сборка мусора с поколениями G1 или Garbage-First GC https://youtu.be/iGRfyhE02lA

1) Слабая гипотеза о поколениях («слабая» — это значит, что не обязательно так, но если в приложении это так устроено, то такая сборка будет наиболее эффективна)
   - большинство объектов временные
   - старые объекты редко ссылаются на молодые

2) Молодые и старые объекты содержатся отдельно
   - в «пространствах», называемых «поколения» (generations)
   - возможны разные алгоритмы для молодого и старого поколения
   - молодое поколение можно собирать отдельно от старого

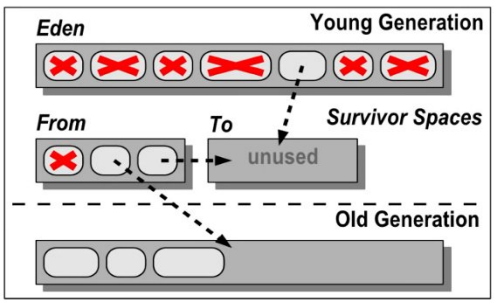
Процессы сборки мусора разделяются на несколько видов: Minor GC, Major GC, Full GC. Сначала происходит малая сборка мусора в области непостоянных объектов (Young Gen)

1) Minor GC (малая)

Частый и быстрый, работает только с областью памяти "young generation".

- приложение приостанавливается на начало сборки мусора (такие остановки называются stop-the-world)
- «живые» объекты из Eden (тут создаются new Object) перемещаются в область памяти «To»
- «живые» объекты из «From space» перемещаются в «To space» или в «Old generation», если они достаточно «старые»
- Eden и «From» очищаются от мусора
- «To space» и «From space» меняются местами
- «очищенное» приложение возобновляет работу

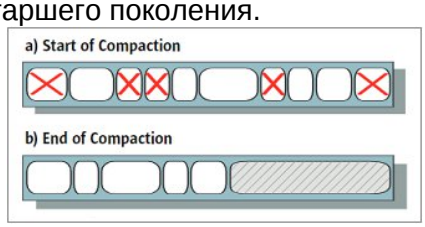
Молодое поколение: сборка мусора



Когда место заканчивается в области непостоянных объектов, тогда запускается полная или старшая сборка мусора. И она работает сразу в обоих поколениях (Young & Old).

2) **Major GC (старшая)**
   Редкий и более длительный, затрагивает объекты старшего поколения.

В принцип работы «major GC» добавляется процедура «уплотнения», позволяющая более эффективно использовать память. В процедуре живые объекты перемещаются в начало. Таким образом, мусор остается в конце памяти.



3) **Full Garbage Collection (полная)**
   Полный сборщик мусора сначала запускает Minor, а затем Major.

Хотя порядок может быть изменен, если старое поколение заполнено, и в этом случае он освобождается первым, чтобы позволить ему получать объекты от молодого поколения. Сборка отработала, потом происходит дефрагментация объектов памяти и цикл начинается сначала.

Garbage-First, G1 GC:

- фоновый и параллельный
- высокая предсказуемость работы (паузы малЫ)
- сборщик мусора с поколениями, НО…

Куча состоит из регионов. Нет ФИЗИЧЕСКОГО разделения между молодым и старым поколением. Принадлежность к регионам определяется динамически. Для каждого региона известно, где находятся объекты, ссылающиеся на него.

Максимальное количество «регионов» в куче G1 GC – это 2^{11} (два в одиннадцатой). Каждый регион фиксированного размера от 1 мб (по дефолту) до 32 мб.

Справка: Большие объекты еще называют **Акселератами** (иногда их называют гигантскими, огромными или «humongous»).

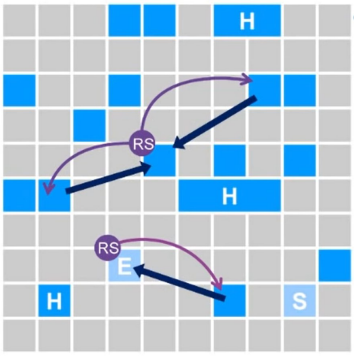
Объекты-акселераты, размер которых настолько велик, что создавать их в Eden, а потом таскать за собой по Survivor'ам слишком накладно. **В этом случае они сразу размещаются в Tenured (старшее поколение).**

RSet хранит инфо о местонахождении ссылок на OLD объекты из региона. Почему old? Хранить ссылки из старшего поколения на младшее нет смысла, т.к. молодое УЖЕ собрано на первом этапе. В молодом поколении % мёртвых объектов максимальный.

Справа полезный плагин, визуализирующий процесс сборки мусора.

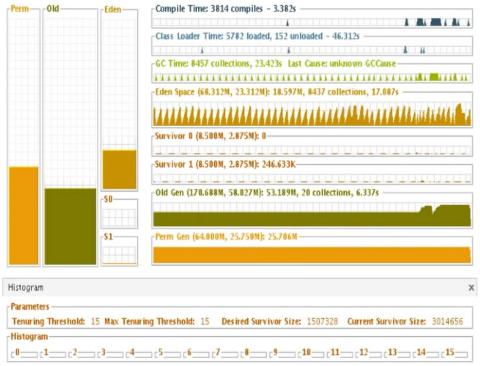
G1: Структура «кучи»

Освобождение памяти : компактификация за счёт копирования



- **RSet == Remembered Set**
  - Информация о местонахождении ссылок на объекты из региона
  - Позволяет собирать регионы независимо
  - RSet поддерживается
    - Из старого в молодое поколение
    - Между регионами в старом поколении

VisualVM / VisualGC



Утечки памяти и сборка мусора https://habr.com/ru/company/otus/blog/589321/