Устройство памяти и GC в JAVA.

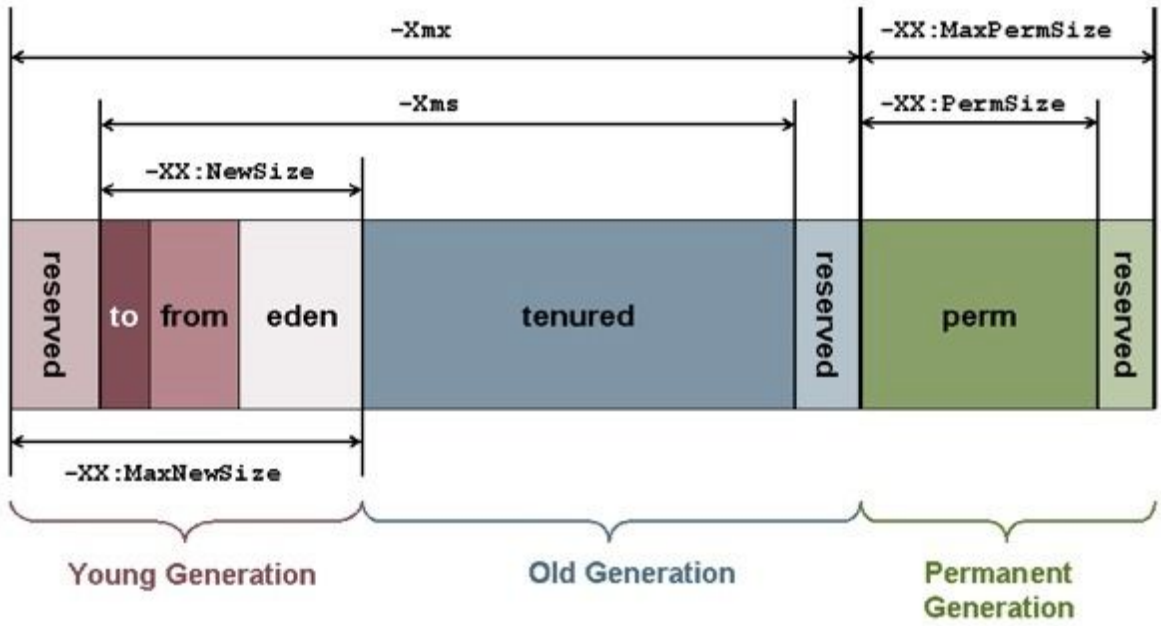
Общая картина.

HEAP:

- **NEW GENERATION**
  - **Eden Space** (все объекты сначала создаются здесь)
  - **Survivor space** (S0, S1) (from space, to space) (выжившие, кандидаты на долгоживущие)
- **OLD GENERATION**

NON-HEAP:

- **Permanent Generation** (метаинформация, используемая JVM (используемые классы, методы и т.п.)
- **Code Cache**



Было замечено, что большинство приложений удовлетворяют двум правилам (**weak generational hypothesis**):

- Большинство аллоцированых объектов быстро становятся мусором.
- Существует мало связей между объектами, которые были созданы в прошлом и только что аллоцированными объектами.

Память делится на три пространства:

- **Young generation**. Объекты аллоцируются в этом участке. Обычно имеет сравнительно не большой размер. Очищается часто. Предполагается, что

количество объектов переживших сборку будет мало (основывая на "weak generational hypothesis"). Сборку мусора в этом участке называют "**minor garbage collection**". В общем, "minor garbage collection" проходит часто, быстро и уничтожает кучу мусора, так как происходит на сравнительно не большом участке памяти который скорее всего содержит много мусора.

- **The old generation**. Объекты которые переживают "minor collection" перемещаются в участок памяти называемый "old generation". Обычно "old generation" больше чем "young generation". Заполняется этот участок сильно медленней, так как большинство объектов живут не долго. В итоге, сборка мусора в "old generation" (major garbage collection) происходит не часто, но когда происходит, занимает много времени.
- **Permanent generation**. Тут хранятся метаданные, классы, интернированные строки, и т.д.

Новые объекты создаются в "**young generation**", пережившие сборку мусора попадают в "**old generation**". Существует "**minor GC**" и "**major GC**".

- "**minor GC**" - проходит часто и быстро, в основном работает с "young generation".
- "**major GC**" - проходит редко и долго, в основном работает с "old generation".

Алгоритмы очистки.

Java в основном использует 2 алгоритма очистки.

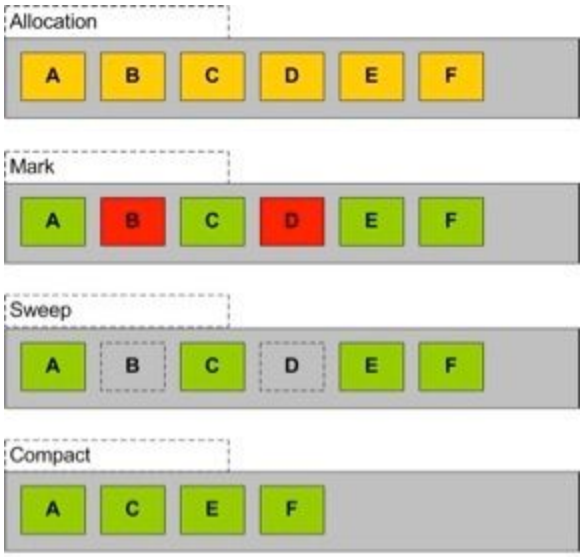
Mark-Sweep-Compact (используется в основном для Major Collection, Old Generation)

***«Mark»**: помечаются неиспользуемые объекты.*

***«Sweep»**: эти объекты удаляются из памяти.*

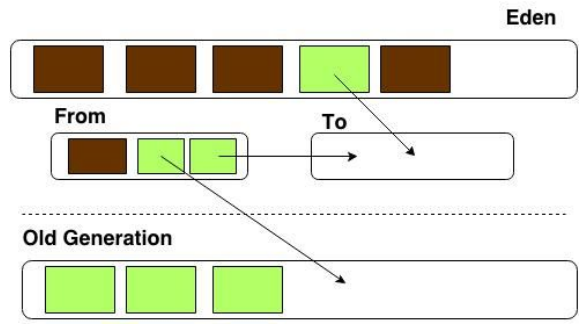
***«Compact»**: объекты «уплотняются», занимая свободные слоты, что освобождает пространство на тот случай, если потребуется создать «большой» объект.*

*На протяжении всех трех шагов действует **«stop-the-world»** пауза.*



Copying Collector (используется в основном при Minor GC, в Young Generation)

Выполняет все три действия за один проход, используя области “from” и “to”. Происходит это следующим образом:
в начале работы одно из survivor spaces, “To”, является пустым, а другое, “From”, содержит объекты, пережившие предыдущие сборки.
Сборщик мусора ищет живые объекты в Eden и копирует их в To space, а затем копирует туда же и живые «молодые» (то есть не пережившие еще заданное число сборок мусора, tenuring threshold) объекты из From space.
Старые объекты из From space перемещаются в Old generation. После сборки From space и To space меняются ролями, область Eden становится пустой, а число объектов в Old generation увеличивается. Если в процессе копирования живых объектов To space переполняется, то оставшиеся живые объекты из Eden и From space, которым не хватило места в To space, будут перемещены в Old generation, независимо от того, сколько сборок мусора они пережили.

До	После
	

“

Использовал:

- https://habrahabr.ru/post/84165/
- https://habrahabr.ru/post/148322/
- https://habrahabr.ru/post/112676/
- http://ggenikus.github.io/blog/2014/05/04/gc/