

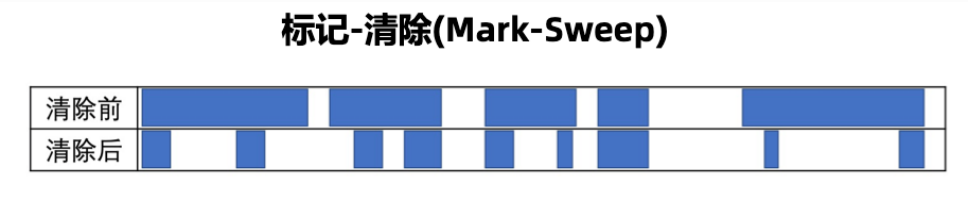
# GC总结

## 1.内存回收算法

### 1.1 标记-清除( Mark-Sweep )

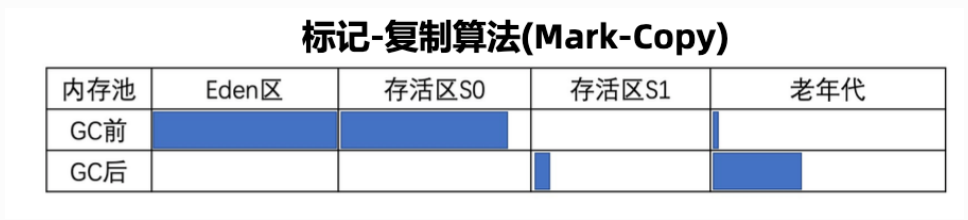
首先标记处所有需要清除的对象，然后统一清除掉所有被标记的对象。存在两个缺点：

- 若存在大量对象需要清除，则需要大量的标记清除动作
- 标记清除后，存在内存空间碎片化的问题



### 1.2 标记-复制算法( Mark-Copy )

为解决标记-清楚在面对大量可收回对象时执行效率低的问题，提出“半区复制，将内存空间分为大小相等的两块。每次只要一块内存区有对象，而在标记-复制时，将需要收回的对象，按照顺序复制到另一个空的内存区。优点是执行效率高，缺点是只能使用一半的内存空间，空间被浪费。



### 1.3 标记-清除-整理算法( Mark-Sweep-Compact )



# 2. 垃圾回收器

## 2.1 串行GC

### 2.1.1 参数配置

```
java -XX:+UseSerialGC -Xmx512m -Xms512m -XX:+PrintGCDetails -Xloggc:gc.serial.log -
XX:MaxTenuringThreshold=8 -XX:+PrintCommandLineFlags -XX:+PrintGCDateStamps GCLogAnalysis
```

### 2.2.2 特点

- 对于年轻代采用 `Mark-Copy` ；对于老年代采用 `Mark-Sweep-Compact`
- 均采用单线程进行垃圾回收，且都会触发STW



### 2.2.3 两种GC事件解读

- Minor GC(Young GC)
  - 采用 `Mark-Copy` 算法，只对年轻代区中的Eden区和一个存活区进行GC操作。一部分对象复制到另一个存活区，一部分对象复制到老年代区，将原有的数据清除。可以采用 `-XX:MaxTenuringThreshold` 设定，对象在年轻区复制次数，当超过该次数时，则会被复制到老年代



- Full GC
  - 采用 `Mark-Sweep-Compact` 算法,主要集中对老年代区和Metaspace区进行GC

# Full GC

内存池	Eden区	存活区S0	存活区S1	老年代
GC前	?	?		
GC后	?	?		

## 2.2 并行GC

### 2.2.1 参数配置

```
-XX:+UseParallelGC -Xmx512m -Xms512m -XX:+PrintGCDetails -Xloggc:gc.parallel.log -  
XX:MaxTenuringThreshold=8 -XX:ParallelGCThreads=4 -XX:+PrintCommandLineFlags -  
XX:+PrintGCDateStamps GCLogAnalysis
```





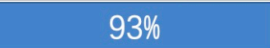
### 2.2.2 特点

- 同上面的串行GC，唯一不同之处是采用多线程并行处理GC，所以相比于串行GC，总的STW时间更短
- 有利于增加系统吞吐量（即降低GC总体消耗时间）

### 2.2.3 两种GC事件解读

- Minor GC(Young GC)：同串行GC，只不过这里是并行进行处理，采用 [Mark-Copy](#) 算法

## Parallel: 年轻代GC

内存池	Eden区	存活区S0	存活区S1	老年代
GC前				 79%
GC后				 93%

- Full GC：与上述串行GC不同，在这里Young区会被清除，一部分对象会晋升到Old区；同时对Old区和Metaspace区进行 [Mark-Sweep-Compact](#)

# Parallel: Full GC

内存池	Eden区	存活区S0	存活区S1	老年代
GC前				
GC后				

## 2.3 CMS GC

### 2.3.1 参数配置

```
java -XX:+UseConcMarkSweepGC -Xmx256m -Xms256m -XX:+PrintGCDetails -Xloggc:gc.CMSGC.log -XX:+PrintCommandLineFlags GCLogAnalysis
```

### 2.3.2 特点

- 对Young区进行 并行STW 的 Mark-Copy ，对old区采用 并发方式 使用 Mark-Sweep
- 目的：避免在old区进行垃圾收集时，出现长时间卡顿，两种方式达成这个目的
  - 不对老年代进行整理，采用空闲列表（free-list）管理内存空间的收回
  - Mark-Sweep 的回收线程与业务线程是并发执行的
- 六个阶段
  - Initial Mark(存在 STW )
  - Concurrent Mark
  - Concurrent Preclean
  - Concurrent Abortable Preclean
  - Final Remark(存在 STW )
  - Concurrent Sweep
  - Concurrent Reset

### 2.3.3 两种GC事件解读

- **Minor GC:** Young 区采用 **ParNew**，即并行GC，也是采用 **Mark-Copy** 算法
- **Full GC:** 在old区进行并发的GC时（只会进行对Old进行标记-清除，不会进行整理），中间可能存在多次的对Young区进行的并行GC
- CMS的设计，完全为了对Old区进行回收的，所以使用时配合使用UseParNewGC

## 2.4 G1 GC

### 2.4.1 参数配置（XX:MaxGCPauseMillis=50 设置最大GC暂停时间，并不是一定会小于50）

```
java -XX:+UseG1GC -XX:MaxGCPauseMillis=50 -XX:+PrintGCDetails -Xloggc:gc.G1GC.log -XX:+PrintCommandLineFlags GCLogAnalyser
```

### 2.4.2 特点

- **G1(Garbage-First)**, 不再划分年轻代和老年代，而是划分多个存放对象的小块堆区域(smaller heap regions)，其中每一块都有可能被定义为Eden区/Survivor区/Old区
- 采用增量的方式（每次不需要将所有的垃圾收回，符合实际情况，因为垃圾收回的线程不可能是主角；业务线程执行才是主体）处理，每次GC暂停会收集所有年轻代和部分老年代内存；

### 2.4.3 两种GC事件

- **单个模式GC:** 只会对年轻代进行GC（标记-清除），
- **混合模式GC:** 当老年代的使用超过 **XX:initialHeapSize** 时，需要对年轻代和老年代都进行GC