V8引擎中的垃圾回收机制 GC

1. GC：垃圾回收机制（用于负责垃圾回收即清除不用或者用过的内存）
2. 常见的GC算法：
   - 标记清除
     - 垃圾收集器在运行时给内存中所以变量加上一个标记，假设所有对象都是垃圾，全部标记为0
     - 然后从各个根对象开始遍历，把不是垃圾的节点改为1
     - 清除所有标记为0的垃圾，销毁并回收他们所占用的内存空间
     - 最合，把所有内存中对象标记修改为0，等待下一轮垃圾回收

   优点： 实现简单，只需要进行标记即可

   缺点： 清除垃圾后，剩余的对象的位置是不变的，会导致空闲的内存不连续，出现内存碎片，引起内存分配问题，即出现 **内存碎片化 分配速度慢** 。使用**标记整理法**可以有效的解决内存碎片化的问题：他在标记结束后，将活着的对象向内存一端移动，最后清理掉边界的内存。

   - 引用计数法
     - 当声明了一个变量并且将一个引用类型赋值给该变量的时候这个值的引用次数就为1
     - 如果同一个值又被赋给另一个变量，那么这个引用数加1
     - 如果该变量的值被其他的值覆盖了，则引用次数减1

     当这个值的引用次数变为0的时候，说明没有变量在使用，这个值没法被访问了，回收空间，垃圾回收器会在运行的时候清理掉引用次数为0的值占用的内存

   优点： 相比于标记清除来说，在计数引用值为0的时候就可以被立即回收

   缺点：无法解决循环引用无法回收的问题
3. V8对GC的优化

   内存限制： 由于Javascript 单线程的执行机制 和 垃圾回收机制 的限制，导致了一旦进入到垃圾回收过程中，其他的所有逻辑都得停止，另一方面垃圾回收耗费时间也是一个非常耗费时间的操作

   分代式垃圾回收：

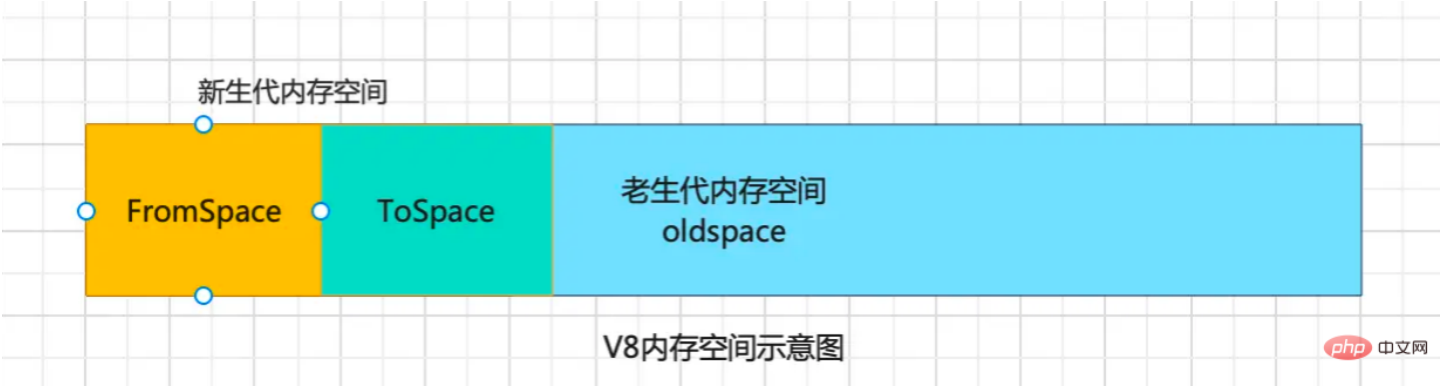
   V8基于分代式垃圾回收机制，将堆内存分为新生代和老生代区域

   - 新生代区域：新生代的对象主要为存活时间较短的对象，简单来说是新产生的对象，通常1-8M

- 老生代区域： 老生代为存活时间较长或者常驻内存的对象

新生代的垃圾回收：

这个算法中，主要采用了一种复制式法即Cheney算法。该算法将新生代一分为二，一个是处于使用状态的空间称之为使用区 ，另一部分处于闲置状态的称为 空闲区



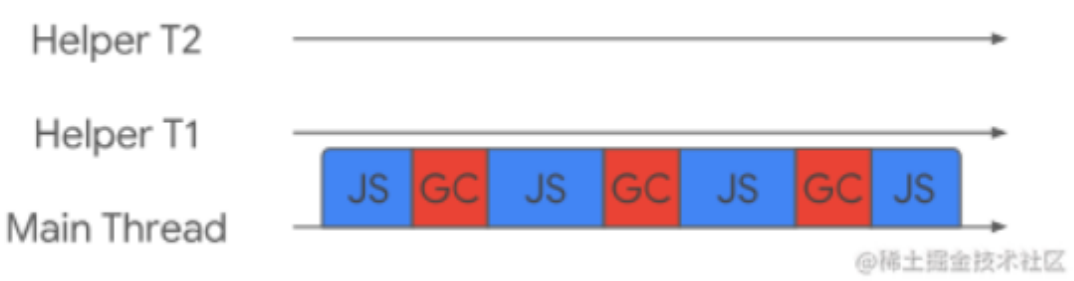
- 新加入的对象都放在使用区，当使用区快要写满时，就需要进行一次垃圾回收
- 开始回收时，新生代垃圾回收器会对使用区的活动对象做标记，标记完成后，将标记的活动对象复制进空闲区并进行排序。然后将非活动对象占用的空间清除掉。
- 随后进入角色互换，将原来的使用区变成空闲区，空闲区变为使用区
- 当一个对象经过多次复制还依然存活，它将被认为是生命周期较长的对象，随后会被移入老生代的内存中

老生代的垃圾回收

标记清除算法，清除阶段垃圾回收器会直接将非活动对象处理掉

优化

- 新生代对象空间使用**并行策略，**在执行垃圾回收时，会启动多个线程来负责新生代的垃圾清理操作，这些线程同时将对象空间的数据移入到空闲区域中，整个过程由于数据地址的改变，需要同步更新引用这些对象的指针
- 老生代采用**增量标记法，**增量就是将一次GC标记的过程分为很多小步，每次执行完一小步就让应用逻辑执行停一会儿，这样交替多次后完成GC标记



增量回收策略：**三色标记法 （暂停与重启）**

使用每个对象的两个标记位和一个标记工作表来实现标记，两个标记位编码三种颜色：白、灰、黑

- 白色指未被标记的对象
- 灰色是指被标记，成员变量（该对象的引用对象）未被标记
- 黑色是指自身和成员变量皆被标记