20 | 垃圾回收（一）：V8的两个垃圾回收器是如何工作的?

2020-04-30 李兵

《图解 Google V8》 课程介绍 >

讲述：李兵

时长 11:39 大小 10.69M

你好，我是李兵。

我们都知道，JavaScript 是一门自动垃圾回收的语言，也就是说，我们不需要去手动回收垃圾数据，这一切都交给 V8 的垃圾回收器来完成。V8 为了更高效地回收垃圾，引入了两个垃圾回收器，它们分别针对着不同的场景。

那这两个回收器究竟是如何工作的呢，这节课我们就来分析这个问题。

垃圾数据是怎么产生的?

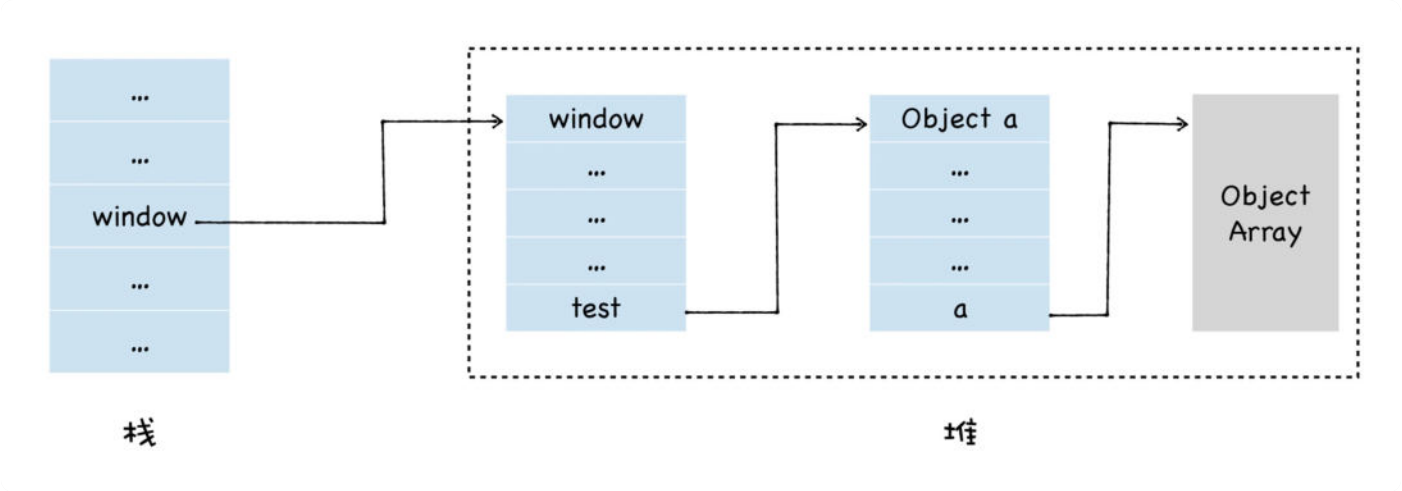
首先，我们看看垃圾数据是怎么产生的。

无论是使用什么语言，我们都会频繁地使用数据，这些数据会被存放到栈和堆中，通常的方式是在内存中创建一块空间，使用这块空间，在不需要的时候回收这块空间。

比如下面这样一句代码：

```
window.test = new Object()
window.test.a = new Uint16Array(100)
```

当 JavaScript 执行这段代码的时候，会先为 window 对象添加一个 test 属性，并在堆中创建了一个空对象，并将该对象的地址指向了 window.test 属性。随后又创建一个大小为 100 的数组，并将属性地址指向了 test.a 的属性值。此时的内存布局图如下所示：

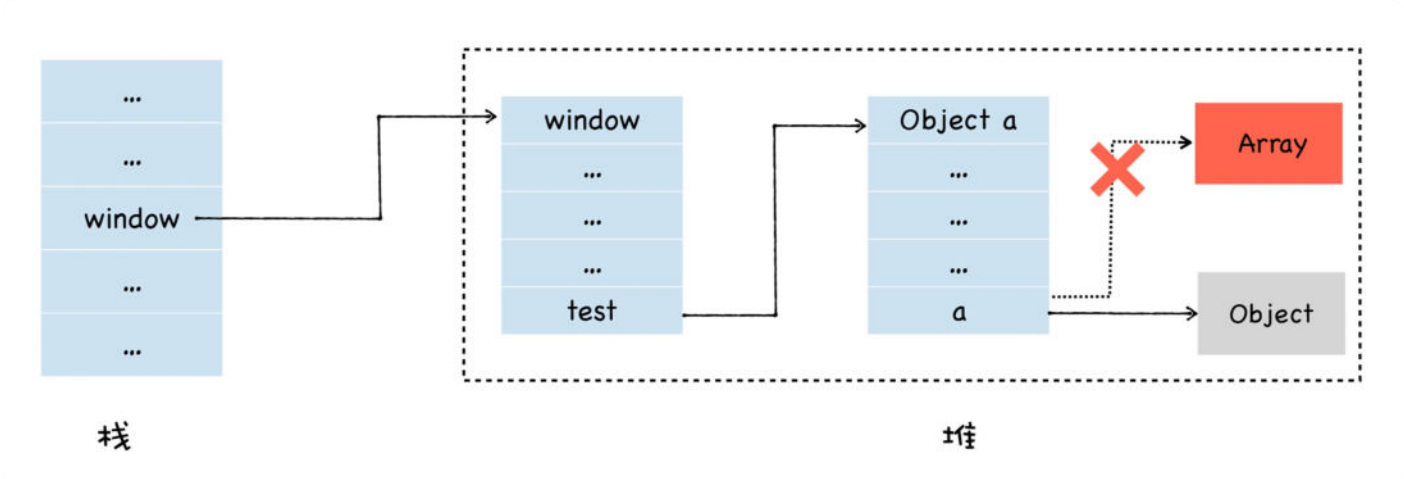


我们可以看到，栈中保存了指向 window 对象的指针，通过栈中 window 的地址，我们可以到达 window 对象，通过 window 对象可以到达 test 对象，通过 test 对象还可以到达 a 对象。

如果此时，我将另外一个对象赋给了 a 属性，代码如下所示：

```
window.test.a = new Object()
```

那么此时的内存布局如下所示：



我们可以看到，a 属性之前是指向堆中数组对象的，现在已经指向了另外一个空对象，那么此时堆中的数组对象就成为了垃圾数据，因为我们无法从一个根对象遍历到这个 Array 对象。

不过，你不用担心这个数组对象会一直占用内存空间，因为 V8 虚拟机中的垃圾回收器会帮你自动清理。

垃圾回收算法

那么垃圾回收是怎么实现的呢？大致可以分为以下几个步骤：

第一步，通过 GC Root 标记空间中**活动对象**和**非活动对象**。

目前 V8 采用的**可访问性（reachability）算法**来判断堆中的对象是否是活动对象。具体地讲，这个算法是将一些 **GC Root** 作为初始存活的对象的集合，从 GC Roots 对象出发，遍历 GC Root 中的所有对象：

- 通过 GC Root 遍历到的对象，我们就认为该对象是**可访问的（reachable）**，那么必须保证这些对象应该在内存中保留，我们也称可访问的对象为活动对象；
- 通过 GC Roots 没有遍历到的对象，则是**不可访问的（unreachable）**，那么这些不可访问的对象就可能被回收，我们称不可访问的对象为非活动对象。

在浏览器环境中，GC Root 有很多，通常包括了以下几种 (但是不止于这几种)：

- 全局的 window 对象（位于每个 iframe 中）；
- 文档 DOM 树，由可以通过遍历文档到达的所有原生 DOM 节点组成；

- 存放栈上变量。

第二步，回收非活动对象所占据的内存。其实就是在所有的标记完成之后，统一清理内存中所有被标记为可回收的对象。

第三步，做内存整理。一般来说，频繁回收对象后，内存中就会存在大量不连续空间，我们把这些不连续的内存空间称为**内存碎片**。当内存中出现了大量的内存碎片之后，如果需要分配较大的连续内存时，就有可能出现内存不足的情况，所以最后一步需要整理这些内存碎片。但这步其实是可选的，因为有的垃圾回收器不会产生内存碎片，比如接下来我们要介绍的副垃圾回收器。

以上就是大致的垃圾回收的流程。目前 V8 采用了两个垃圾回收器，**主垃圾回收器 –Major GC 和副垃圾回收器 –Minor GC (Scavenger)**。V8 之所以使用了两个垃圾回收器，主要是受到了**代际假说（The Generational Hypothesis）**的影响。

代际假说是垃圾回收领域中一个重要的术语，它有以下两个特点：

- 第一个是大部分对象都是“朝生夕死”的，也就是说大部分对象在内存中存活的时间很短，比如函数内部声明的变量，或者块级作用域中的变量，当函数或者代码块执行结束时，作用域中定义的变量就会被销毁。因此这一类对象一经分配内存，很快就变得不可访问；
- 第二个是不死的对象，会活得更久，比如全局的 window、DOM、Web API 等对象。

其实这两个特点不仅仅适用于 JavaScript，同样适用于大多数的编程语言，如 Java、Python 等。

V8 的垃圾回收策略，就是建立在该假说的基础之上的。接下来，我们来分析下 V8 是如何实现垃圾回收的。

如果我们只使用一个垃圾回收器，在优化大多数新对象的同时，就很难优化到那些老对象，因此你需要权衡各种场景，根据对象生存周期的不同，而使用不同的算法，以便达到最好的效果。

所以，在 V8 中，会把堆分为新生代和老生代两个区域，**新生代中存放的是生存时间短的对象，老生代中存放生存时间久的对象**。

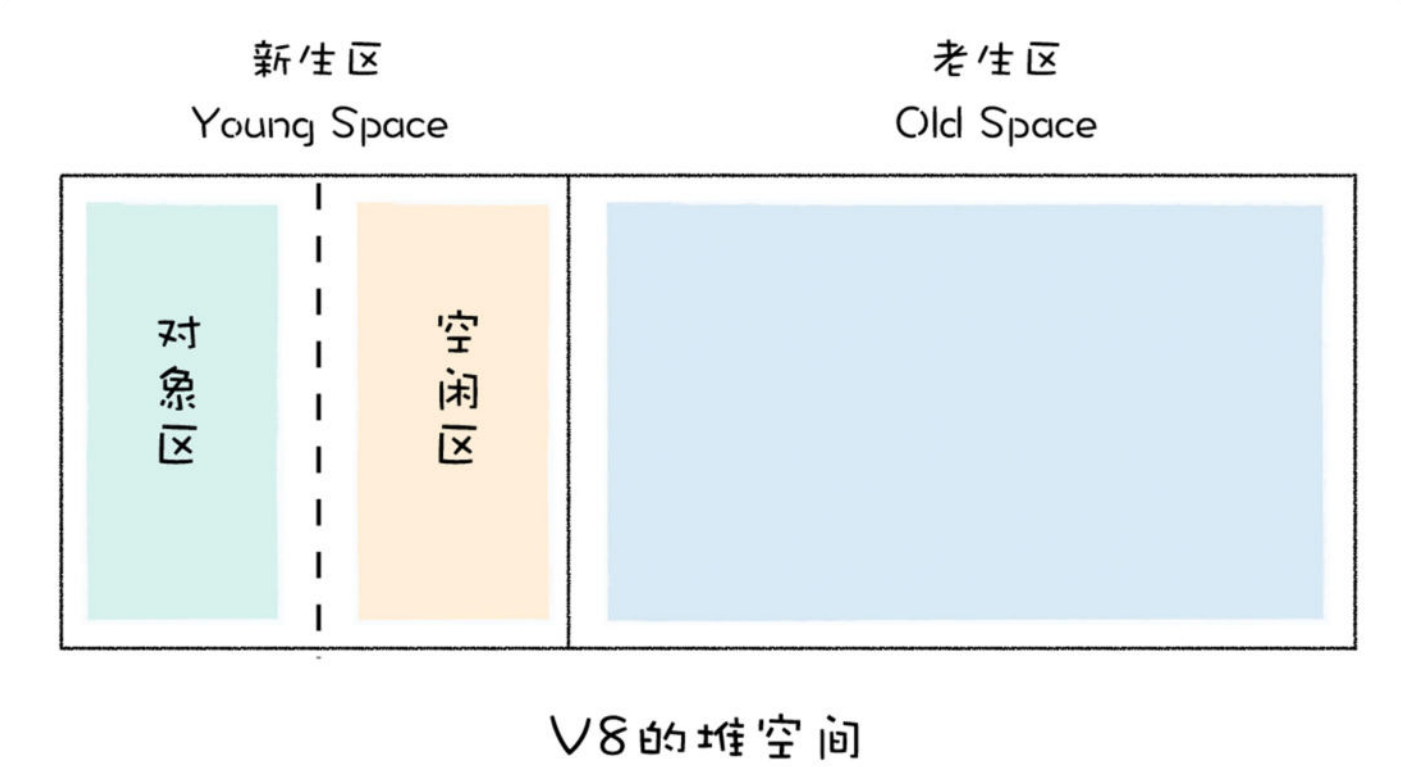
新生代通常只支持 1～8M 的容量，而老生代支持的容量就大很多了。对于这两块区域，V8 分别使用两个不同的垃圾回收器，以便更高效地实施垃圾回收。

- **副垃圾回收器 –Minor GC (Scavenger)，主要负责新生代的垃圾回收。**
- **主垃圾回收器 –Major GC，主要负责老生代的垃圾回收。**

副垃圾回收器

副垃圾回收器主要负责新生代的垃圾回收。通常情况下，大多数小的对象都会被分配到新生代，所以说这个区域虽然不大，但是垃圾回收还是比较频繁的。

新生代中的垃圾数据用 **Scavenge 算法**来处理。所谓 Scavenge 算法，是把新生代空间对半划分为两个区域，一半是**对象区域 (from–space)**，一半是**空闲区域 (to–space)**，如下图所示：

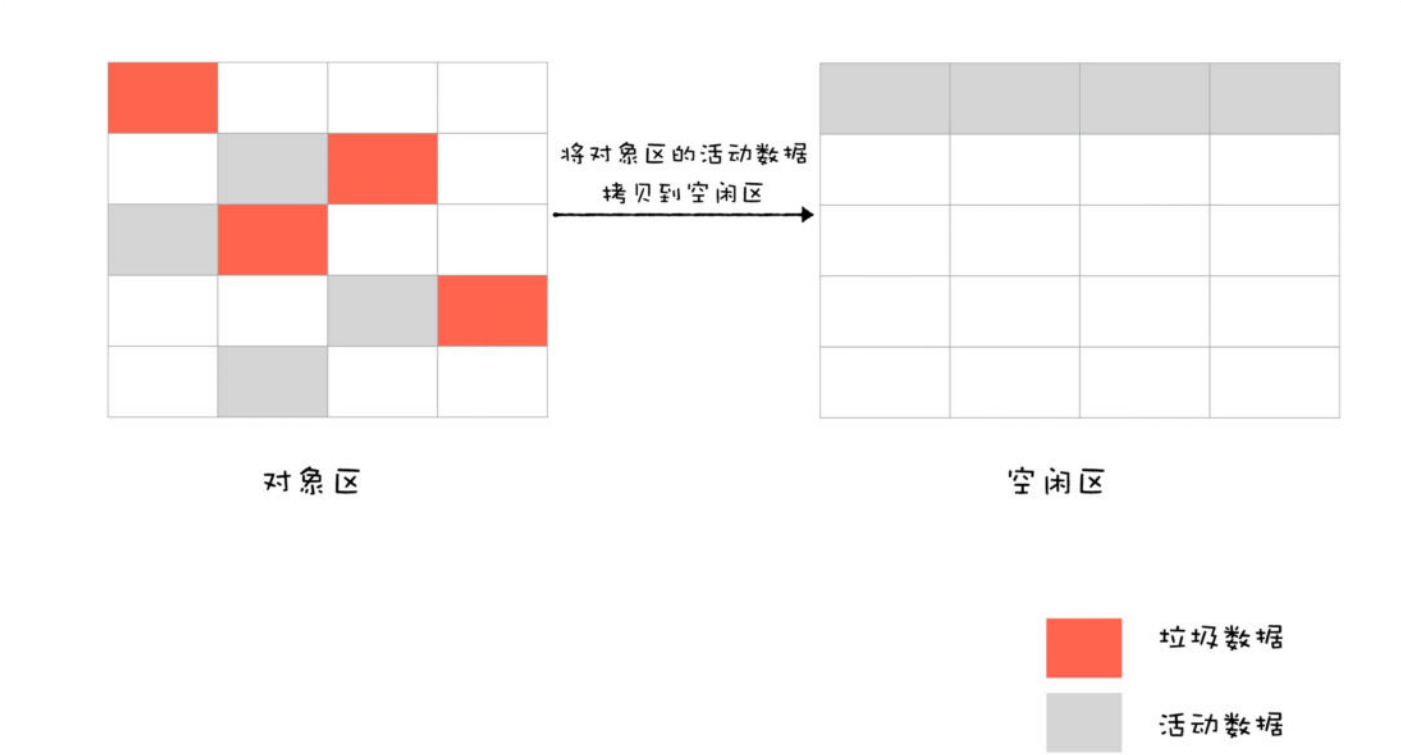


新生区要划分为对象区域和空闲区域

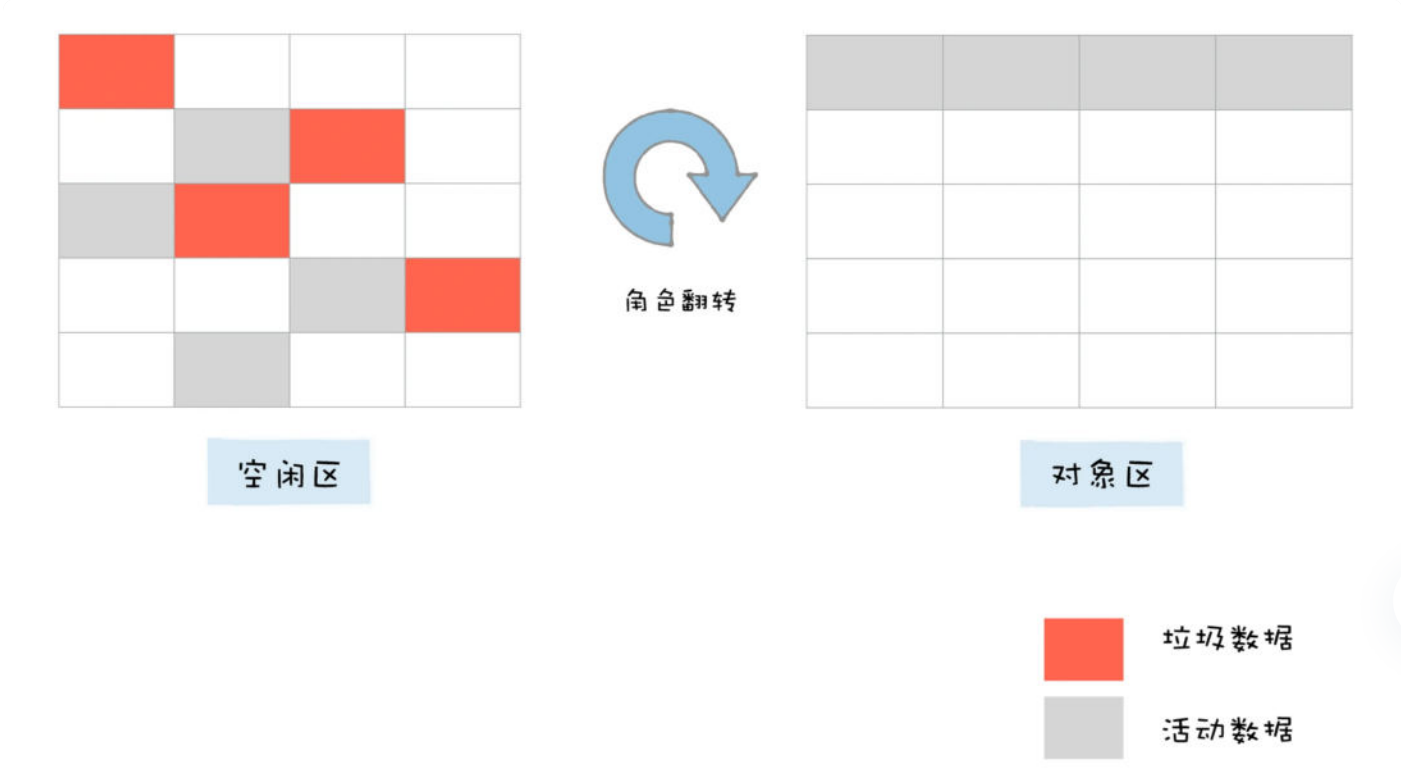
新加入的对象都会存放到对象区域，当对象区域快被写满时，就需要执行一次垃圾清理操作。

在垃圾回收过程中，首先要对对象区域中的垃圾做标记；标记完成之后，就进入垃圾清理阶段。副垃圾回收器会把这些存活的对象复制到空闲区域中，同时它还会把这些对象有序地排列

起来，所以这个复制过程，也就相当于完成了内存整理操作，复制后空闲区域就没有内存碎片了。



完成复制后，对象区域与空闲区域进行角色翻转，也就是原来的对象区域变成空闲区域，原来的空闲区域变成了对象区域。这样就完成了垃圾对象的回收操作，同时，**这种角色翻转的操作还能让新生代中的这两块区域无限重复使用下去**。



不过，副垃圾回收器每次执行清理操作时，都需要将存活的对象从对象区域复制到空闲区域，复制操作需要时间成本，如果新生区空间设置得太大了，那么每次清理的时间就会过久，所以**为了执行效率，一般新生区的空间会被设置得比较小**。

也正是因为新生区的空间不大，所以很容易被存活的对象装满整个区域，副垃圾回收器一旦监控对象装满了，便执行垃圾回收。同时，副垃圾回收器还会采用**对象晋升策略**，也就是移动那些经过两次垃圾回收依然还存活的对象到老生代中。

主垃圾回收器

主垃圾回收器主要负责老生代中的垃圾回收。除了新生代中晋升的对象，一些大的对象会直接被分配到老生代里。因此，老生代中的对象有两个特点：

- 一个是对象占用空间大；
- 另一个是对象存活时间长。

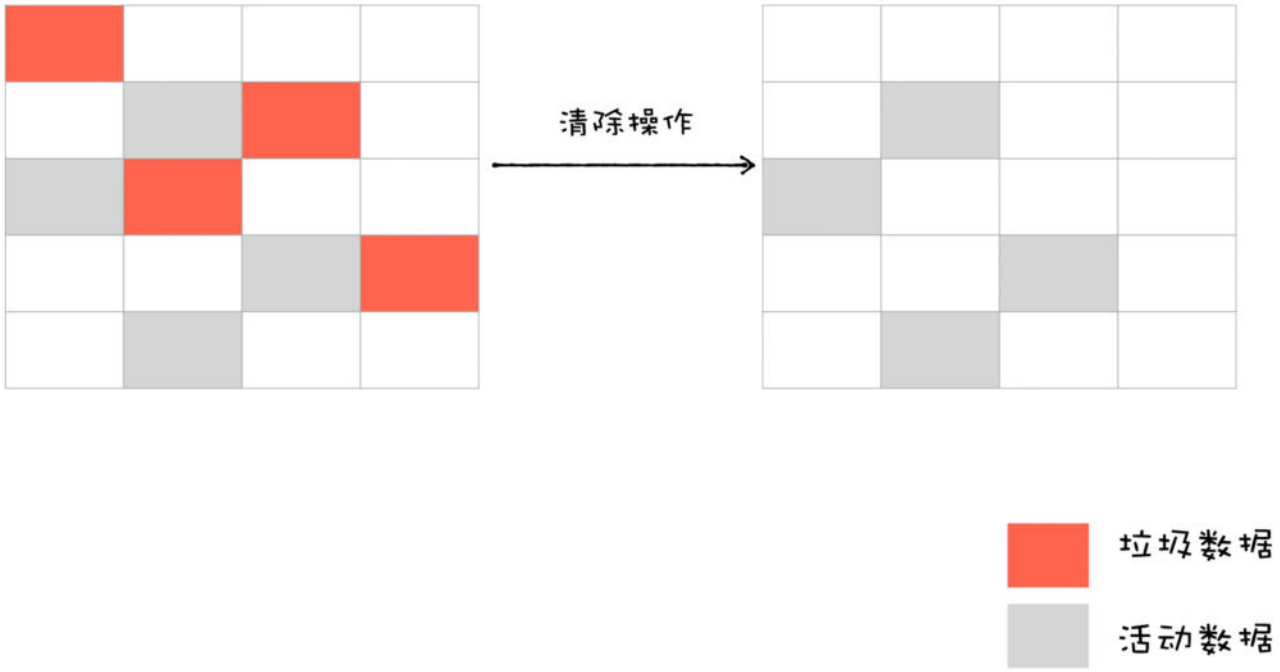
由于老生代的对象比较大，若要在老生代中使用 Scavenge 算法进行垃圾回收，复制这些大的对象将会花费比较多的时间，从而导致回收执行效率不高，同时还会浪费一半的空间。所以，主垃圾回收器是采用**标记 – 清除（Mark–Sweep）**的算法进行垃圾回收的。

那么，标记 – 清除算法是如何工作的呢?

首先是标记过程阶段。标记阶段就是从一组根元素开始，递归遍历这组根元素，在这个遍历过程中，能到达的元素称为活动对象，没有到达的元素就可以判断为垃圾数据。

接下来就是垃圾的清除过程。它和副垃圾回收器的垃圾清除过程完全不同，主垃圾回收器会直接将标记为垃圾的数据清理掉。

你可以理解这个过程是清除掉下图中红色标记数据的过程，你可参考下图大致理解下其清除过程：



标记清除过程

对垃圾数据进行标记，然后清除，这就是**标记 – 清除算法**，不过对一块内存多次执行标记 – 清除算法后，会产生大量不连续的内存碎片。而碎片过多会导致大对象无法分配到足够的连续内存，于是又引入了另外一种算法——**标记 – 整理（Mark-Compact）**。

这个算法的标记过程仍然与标记 – 清除算法里的是一样的，先标记可回收对象，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉这一端之外的内存。你可以参考下图：

标记整理过程

总结

今天，我们先分析了什么是垃圾数据，从“GC Roots”对象出发，遍历 GC Root 中的所有对象，如果通过 GC Roots 没有遍历到的对象，则这些对象便是垃圾数据。V8 会有专门的垃圾回收器来回收这些垃圾数据。

V8 依据代际假说，将堆内存划分为新生代和老生代两个区域，新生代中存放的是生存时间短的对象，老生代中存放生存时间久的对象。为了提升垃圾回收的效率，V8 设置了两个垃圾回收器，主垃圾回收器和副垃圾回收器。主垃圾回收器负责收集老生代中的垃圾数据，副垃圾回收器负责收集新生代中的垃圾数据。

副垃圾回收器采用了 **Scavenge 算法**，是把新生代空间对半划分为两个区域，一半是**对象区域**，一半是**空闲区域**。新的数据都分配在对象区域，等待对象区域快分配满的时候，垃圾回收器便执行垃圾回收操作，之后将存活的对象从对象区域拷贝到空闲区域，并将两个区域互换。主垃圾回收器回收器主要负责老生代中的垃圾数据的回收操作，会经历标记、清除和整理过程。

思考题

观察下面这段代码：

复制代码

```
function strToArray(str) {
  let i = 0
  const len = str.length
  let arr = new Uint16Array(str.length)
  for (; i < len; ++i) {
    arr[i] = str.charCodeAt(i)
  }
  return arr;
}


function foo() {
  let i = 0
  let str = 'test V8 GC'
  while (i++ < 1e5) {
    strToArray(str);
  }
}


foo()
```

课后请你想一想，V8 执行这段代码的过程中，产生了哪些垃圾数据，以及 V8 又是如何回收这些垃圾的数据的， 最后站在内存空间和主线程资源的角度来分析，如何优化这段代码。欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

上一篇　19 | 异步编程（二）：V8是如何实现async/await的?

下一篇　21 | 垃圾回收（二）：V8是如何优化垃圾回收器执行效率的?

精选留言 (24)

写留言

sugar 置顶

2020-04-30

分享一篇好几年前我刚开始研读v8源码时，参考的GC相关资料：https://www.dynatrace.com/news/blog/understanding-garbage-collection-and-hunting-memory-leaks-in-node-js/
还有这个github仓库：https://github.com/yjhjstz/deep-into-node
如今与老师的这篇

作者回复: 赞

共 4 条评论 >

16

一步

2020-04-30

上面那段代码最大的问题就是 strToArray 函数中的 let arr = new Uint16Array(str.length)，在 foo 函数每次循环执行都会在堆中生成新的数组，这里可以在 foo 函数声明一个数组，然后每次调用 strToArray 函数的时候传递 进去就可以了

作者回复: 是的

9

子云

2020-06-08

老师我有两个疑惑，我看到其他的文章中说新生代是 64位操作系统有32MB，32位系统有16MB，而您的文章中说的是8MB，那么是怎么回事呢。。
还有其他的文章中说的新生代提升到老生带是
* 已经经历过1次 Scavenge 回收。
* To（闲置）空间的内存占用超过25%。
似乎也和您的说法不一样呀

共 1 条评论 >

5

Z_CHP

2020-04-30

有个疑惑，没有被GC Root遍历触达的对象就是垃圾数据。那在垃圾清理的过程中，没有被遍历到的对象是如何被标记为垃圾数据的呢?

作者回复: 比如在标记之前，将所有对象设置为白色，然后遍历到的设置为黑色，最终白色的都视为垃圾数据

共 3 条评论> 4

jyzhang
2020-07-28

新生代是在对象空间快要占满的时候触发垃圾回收，老生代区域的垃圾回收是什么时候触发的?

共 1 条评论> 2

断线人偶
2020-05-24

主垃圾回收器的两种算法是同时使用还是有个策略？还是说早期用的是标记 – 清除算法，现在都是使用标记 – 整理算法?

作者回复: 同时使用的，并不是所有时候都需要做内存整理

2

刘长锋
2020-05-10

老师好：

“其实这两个特点不仅仅适用于 JavaScript，同样适用于大多数的动态语言，如 Java、Python 等。”

这句话，是不是应该写成适合于大多数编程语言?

为什么适合于动态语言呢?

java 是典型的静态语言吧?

作者回复: 嗯，我修改下

共 2 条评论> 2

大力
2020-05-02

麻烦老师能不能讲讲 Scavenge 与 Mark-Compact 算法的不同点?

共 1 条评论>

2

Miracle

2020-04-30

有一个疑问：新生代在进行垃圾回收的时候，复制整理后的空闲区域和对象区域进行反转的时候，对象区域是整体清空吗，还是只清除标记的垃圾数据

作者回复: 全部清空

共 2 条评论>

2

周振

2021-06-25

什么线程在监听堆的内存空间大小，从而让渲染线程启动垃圾回收的呢

1

划水摸鱼小能手

2020-11-08

副垃圾回收器和主垃圾回收器的回收阶段没有讲清楚。垃圾回收器首先会对所有对象进行标记，有被引用到的会清楚标记，垃圾回收过程中，带有标记的变量会被回收掉，这是红宝书里讲的标记清除过程，假设副垃圾回收器是这样进行标记清除的（因为文中副垃圾回收器相关内容只讲了个标记，没有深入讲解），主垃圾回收器的标记清除算法从根元素开始递归根元素遍历，那主垃圾回收器中还得是个树结构?

1

蹦哒

2020-05-12

请教老师：引用计数的垃圾回收机制，目前只看到iOS中使用，想知道苹果为啥选择引用计数，而其他很多语言都采用GC呢？因为GC有stop-the-world的问题，而引用计数方法貌似主要是会有循环引用问题，但是循环引用问题写代码时注意就不会有问题了，而stop-the-world是无法避免的。所以我个人更倾向于引用计数的方案，不知道老师怎么认为的呢?

作者回复: 引用计数存在无法回收循环引用的问题，以前ie的js引擎就采用了循环引用的方式，现在都切换回来了

1

yunplane

2020-05-06

栈里的数据不需要清除吗？怎么清除?

作者回复: 滑动下栈指针就清除了，速度非常快

共 2 条评论> 1

luckyone

2020-04-30

函数里new array 会导致新生代不停触发gc，导致影响性能，改进方案可以吧array 提取出函数调用的环境这样可以放到老生代不会平凡gc

作者回复: 没问题

1

咪呐！哦哈哟嘶！٩(...

2020-04-30

循环引用的怎么做垃圾回收?

作者回复: 采用引用计数的垃圾回收器不好解决循环引用的问题，但是目前v8的标记清理方式可以解决循环引用的问题

共 4 条评论> 1

浩东

2021-07-05

Scavenge算法也要标记吗

马里奥JOSEPH

2021-06-01

对于标记整理的解释："这个算法的标记过程仍然与标记 – 清除算法里的是一样的，先标记可回收对象，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉这一端之外的内存。" 如果先让存活的对象向一端移动，没有先对回收对象进行清理，移动后的内存还是不连续的；如果“直接对可回收对象进行清理”，为何最后还要“直接清理掉这一端之外的内存”？这样不是多此一举吗?

hao

2021-05-16

李兵老师，想请教一下

1. 老生代区域同时采取Mark–Sweep策略和Mark–Compact策略，那么什么时候会去使用Mar

k-Compact策略呢?
个人猜测：是老生区的内存碎片过多，好比如有个MAX值来约束它，超过它了就采取Mark-Compact策略

2. 老生区采取Mark-Compact策略后，存活对象移动到老生区的一端中，如果此时老生区中前面的一端有标记为垃圾的对象，那它们是如何进行交换的呢？（想贴图的，但留言不允许😂，可能描述的不是很清晰）


雪飞鸿
2020-12-27

标记-清除的过程，是不是标记的时候遍历一次内存中的对象，清除的时候再遍历一次内存中的对象?


Jerryz
2020-11-06

1.从优化栈调用的角度，可以内联str2Array的逻辑。
2.从优化对空间的角度，可以复用new Object（）出来的对象。