




Are Your V8 GC Logs
Speaking to You?

About Me

- ❖ @joyeecheung (GitHub, Twitter)
- ❖ Chau Yee Cheung (Cantonese) Qiuyi Zhang /张秋怡 (Mandarin)
- ❖ Joyee
- ❖ Intern @  alinode ∈  AliCloud ∈  Alibaba Group



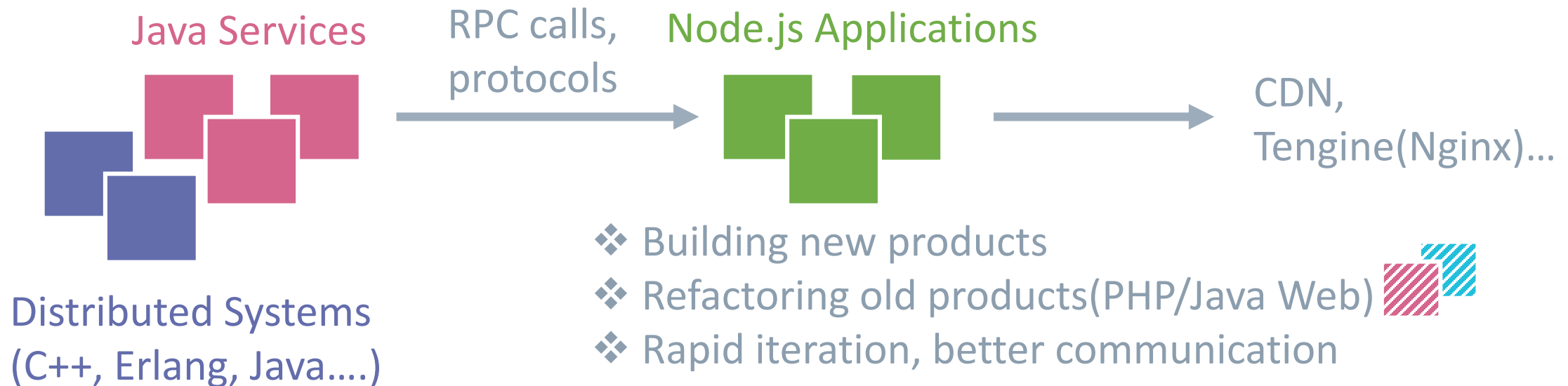
Background

- ❖ We provide **performance management services** to teams both inside and outside the group
- ❖ Alibaba is one of the largest companies in China that uses Node.js in production
 - ❖ Mostly driven by frontend devs

天猫 TMALL.COM

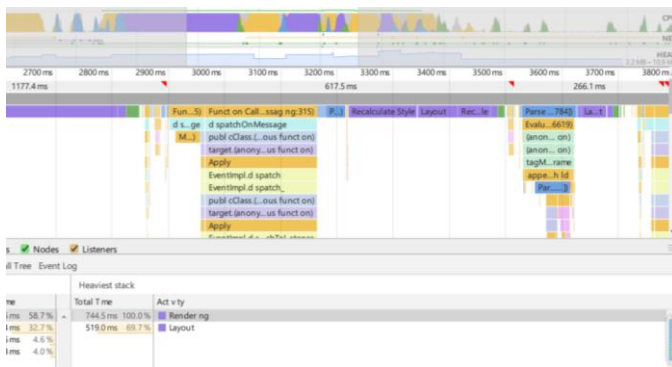
淘宝网
Taobao.com

支付宝
ALIPAY



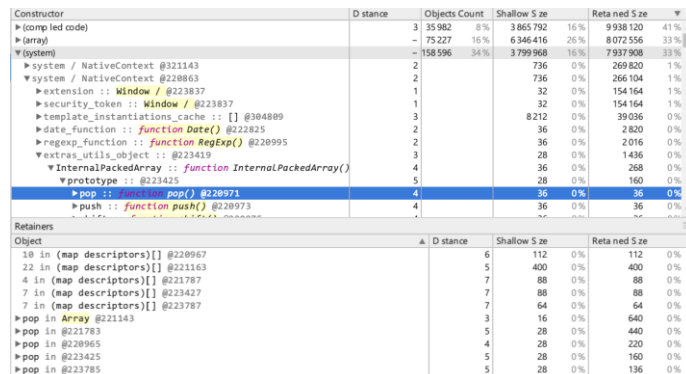
Background

- ❖ Java architects argue that Node.js is *not mature enough*
 - ❖ Lack of tooling for **monitoring and profiling**
- ❖ Java programmers *know* their VM and their GC. As for JS programmers, well...
 - ❖ Most come from a frontend dev background



CPU profile

The pursuit of 60fps



A screenshot of a HeapSnapshot from a web browser's developer tools. The top panel shows a tree of objects, including 'system / NativeContext @221143', 'extension :: Window / @223837', 'security_token :: Window / @223837', 'template_instantiations_cache :: [] @384889', 'date_function :: function Date() @222825', 'regex_function :: function RegExp() @228995', 'extras_utils_object :: @223419', 'InternalPackedArray :: function InternalPackedArray()', 'pop :: @222971', and 'push :: function push() @228973'. The bottom panel shows a table of retainers, listing objects and their memory usage.

Constructor	D stance	Objects Count	Shallow S ze	Retained S ze
▶ (compiled code)		3	35 982	8%
▶ (array)		75 227	6 346 416	26%
▶ (system)		158 596	3 799 968	16%
▶ system / NativeContext @221143		2	736	0%
▶ extension :: Window / @223837		1	32	0%
▶ security_token :: Window / @223837		1	32	0%
▶ template_instantiations_cache :: [] @384889		3	8 212	0%
▶ date_function :: function Date() @222825		2	36	0%
▶ regex_function :: function RegExp() @228995		2	36	0%
▶ extras_utils_object :: @223419		3	28	0%
▶ InternalPackedArray :: function InternalPackedArray()		4	36	0%
▶ pop :: @222971		4	36	0%
▶ push :: function push() @228973		4	36	0%

Object	D stance	Shallow S ze	Retained S ze
10 in (map descriptors)[] @228967		6	112
22 in (map descriptors)[] @221163		5	400
4 in (map descriptors)[] @221787		7	88
7 in (map descriptors)[] @223427		7	88
7 in (map descriptors)[] @223787		7	64
▶ pop in Array @221143		3	16
▶ pop in @221783		5	28
▶ pop in @228965		4	28
▶ pop in @223425		5	28
▶ pop in @223785		5	28

HeapSnapshots / HeapTimelines
you don't want to blow up your
users' RAM



Click this!

GC logs

Servers are long-running,
clients are not

- ❖ V8 garbage collection logs are the least documented tools, not even available in devtools

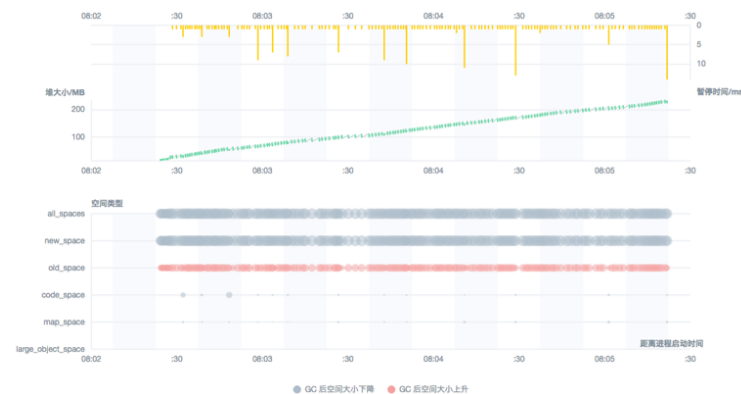
What are V8 Garbage Collection logs good for?

- ❖ Combined with heap profiles, they can help you catch the culprit for **memory leaks**

Constructor	Distance	Objects Count	Shallow Size	Retained Size
▶ (compiled code)	3	35 982	8%	3 865 702
▶ (array)	~	75 227	16%	6 346 416
▼ (system)	~	158 596	34%	3 799 968
▶ system / NativeContext @221143	2	736	0%	269 820
▼ system / NativeContext @220863	2	736	0%	266 104
▶ extension :: Window / @223837	1	32	0%	154 164
▶ security_token :: Window / @223837	1	32	0%	154 164
▶ template_instantiations_cache :: [] @304809	3	8212	0%	39 036
▶ date_function :: function Date() @222825	2	36	0%	2 820
▶ regexp_function :: function RegExp() @220995	2	36	0%	2 016
▼ extras_utils_object :: @223439	3	28	0%	1 436
▶ InternalPackedArray :: function InternalPackedArray()	4	36	0%	268
▼ prototype :: @223425	5	28	0%	160
▶ pop :: function pop() @220971	4	36	0%	36
▶ push :: function push() @220973	4	36	0%	36
Retainers	Distance	Shallow Size	Retained Size	
Object				
18 in (map descriptors)[] @220967	6	112	0%	112
22 in (map descriptors)[] @221163	5	400	0%	400
4 in (map descriptors)[] @221787	7	88	0%	88
7 in (map descriptors)[] @223427	7	88	0%	88
7 in (map descriptors)[] @223787	7	64	0%	64
▶ pop in Array @221143	3	16	0%	640
▶ pop in @221783	5	28	0%	440
▶ pop in @220965	4	28	0%	220
▶ pop in @223425	5	28	0%	160
▶ pop in @223785	5	28	0%	136

Heapdump can tell you **what is leaking**, but they are sometimes too specific

GC logs can give you a bigger picture about what's going on, so you know **where to look**



- ❖ Help you **verify** that your fixes do fix your leaks

An introduction to V8 GC

- ❖ What is Garbage Collection(GC)?
 - ❖ **Collect** and free your **objects(garbage)** when you don't need them anymore. That's why you don't need to manage the memory yourself in languages like JavaScript, Java, Ruby...
 - ❖ For ECMAScript/JavaScript, there are no specifications about it, everything is up to the implementation
- ❖ This talk is based on **V8 4.5.103.35** (the one used by Node v4.x)
 - ❖ Node v6.x will use V8 5.x, which introduces a few improvements(mostly parallel stuff) and changes the GC log format a bit, but most of this talk still apply

An introduction to V8 GC

- ❖ Generational hypothesis
 - ❖ Most objects die young, others tend to live forever
- ❖ V8 GC Strategy
 - ❖ **Generational**: Young and Old
 - ❖ **Accurate**: Given a random word(always at least 4-byte aligned) in the memory, V8 can tell you if it's a pointer or some data by looking at the last bit reserved as a **tag**
 - ❖ So V8 doesn't have to scan the memory to be sure about this, which makes updating the pointers during GC fairly sufficient.
- ❖ GC is triggered by allocations(**new** or **var** most of the time) , when the memory allocated for a space is not enough
- ❖ In V8, Objects live on the heap, and the heap is divided into **spaces**

An introduction to V8 GC

Code Space

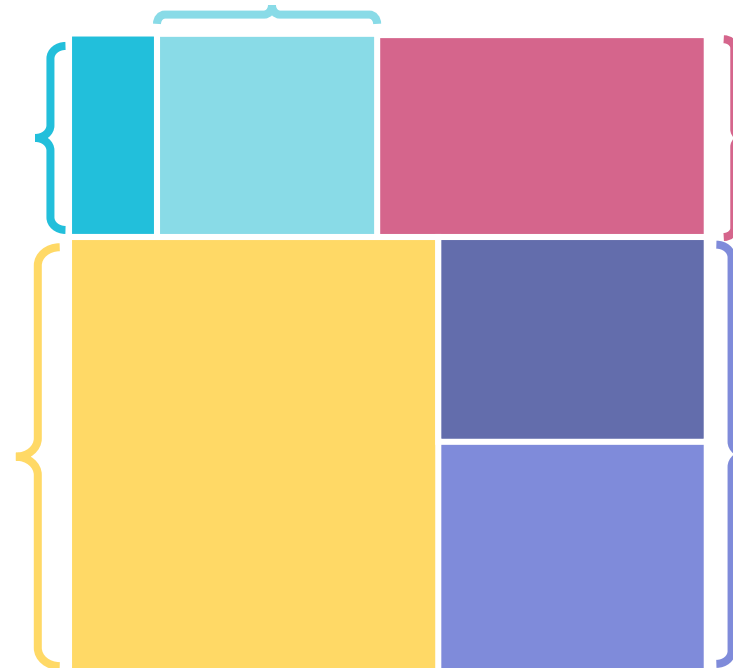
Executable code compiled by V8

Map Space

Metadata of hidden classes, pointed by objects

Large Object Space

Objects too large to fit into any other spaces.



Old Space

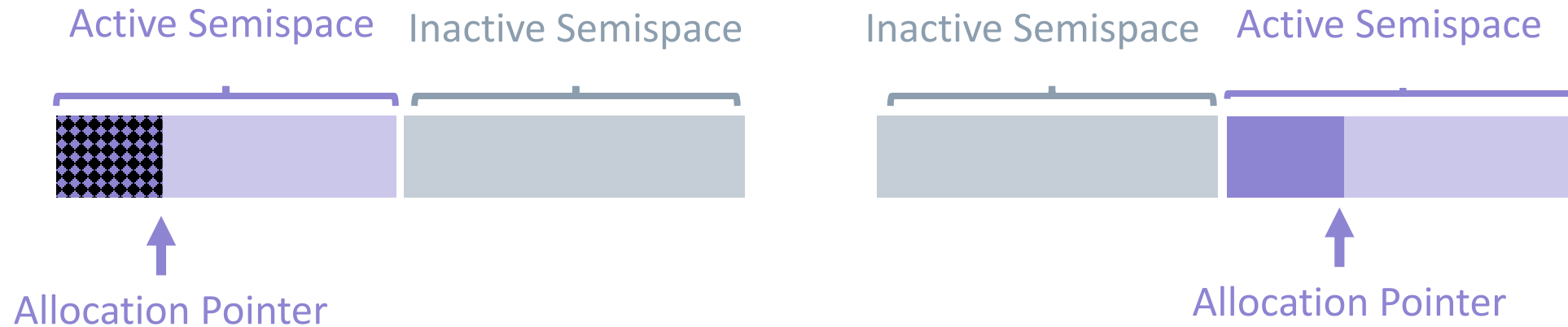
Objects survive more than two GCs in the new space

New Space

Divided in half, only one is active at a time.
Most objects start and end their lives here.

An introduction to V8 GC

- ❖ New space: Scavenger
 - ❖ Implementation of Cheney's algorithm



- ❖ Old space
 - ❖ Mark-sweep/mark-compact
- ❖ *Oddly* similar to the early HotSpot JVM(designed by the same person)
- ❖ Further reading
 - ❖ <http://alinode.aliyun.com/blog/14> (Chinese)
 - ❖ <http://jayconrod.com/posts/55/a-tour-of-v8-garbage-collection> (English)

Getting started with V8 GC logs

- ❖ Plenty of options(`node --v8-options | grep gc`)
- ❖ We'll be focusing on
 - ❖ `--trace_gc`
 - ❖ `--trace_gc_nvp`
- ❖ Obviously you can't use these in production, even though it would be difficult to reproduce your problems offline
 - ❖ `v8.setFlagsFromString('--trace_gc')`
 - ❖ `v8.setFlagsFromString('--notrace_gc')`
 - ❖ Or use alinode to get them by clicking a button on our SAAS platform(wink wink)
- ❖ In case you do want to read the code
 - ❖ `deps/v8/src/heap`, start with `gc-tracer.cc`

Demo Time

Leaks caused by closures

- ❖ Closures are created when **declared**, not when executed
- ❖ In V8, once there is **any** closure in the context, the context will be attached to **every function**, even for those who don't reference the context at all

```
const express = require('express');
const app = express();

var GIANT;

function leak() {
  var HUGE = GIANT;

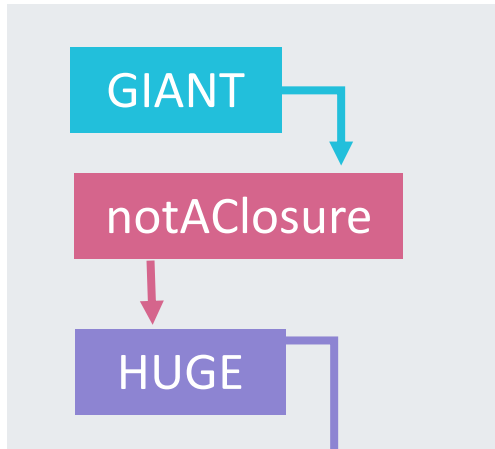
  function unusedClosure() {
    HUGE.slice(1);
  }

  GIANT = {
    willBeLeaked: new Array(1e5).join('.'),
    notAClosure: function notAClosure() {
      return 1;
    }
  }
}

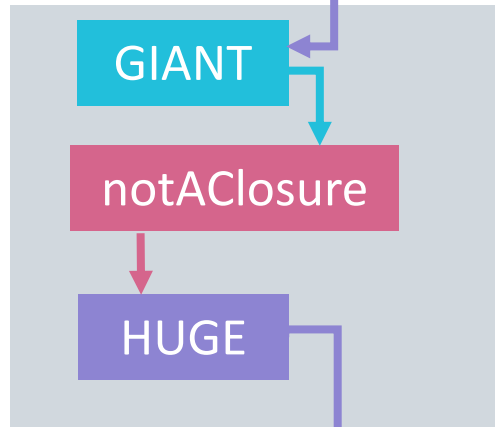
app.get('/', function handler(req, res) {
  leak();
  console.log(new Date());
  res.send('Hello World!');
});

app.listen(3000, function startApp() {
  console.log('Example app listening on port 3000!');
});
```

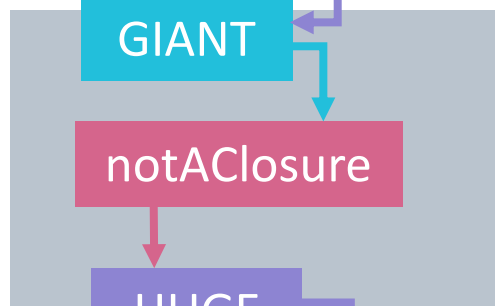
Leaks caused by closures



Created in this call



Created in last call, but can't be GC'ed because they are referenced by the new ones



```
const express = require('express');
const app = express();

var GIANT;

function leak() {
  var HUGE = GIANT;

  function unusedClosure() {
    HUGE.slice(1);
  }

  GIANT = {
    willBeLeaked: new Array(1e5).join('.'),
    notAClosure: function notAClosure() {
      return 1;
    }
  }

  app.get('/', function handler(req, res) {
    leak();
    console.log(new Date());
    res.send('Hello World!');
  });

  app.listen(3000, function startApp() {
    console.log('Example app listening on port 3000!');
  });
}
```

This shouldn't have happened!

It happens because of this

Leaks caused by closures

Constructor	Distance	Objects C...	Shallow Size	Retained Size ▼
▼(string)	3	723 27%	031 696 98%	031 696 98%
"....."	6172		100 024 0%	100 024 0%
"....."	6175		100 024 0%	100 024 0%
"....."	3988		100 024 0%	100 024 0%
"....."	4372		100 024 0%	100 024 0%
"....."	4375		100 024 0%	100 024 0%
"....."	3985		100 024 0%	100 024 0%
"....."	3619		100 024 0%	100 024 0%
"....."	3622		100 024 0%	100 024 0%
"....."	2806		100 024 0%	100 024 0%
"....."	2809		100 024 0%	100 024 0%
Retainers				
Object	Distance ▲	Shallow Size	Retained Size	
▼willBeLeaked in @13957	6171	40 0%	40 0%	
▼HUGE in system / Context @100513	6170	56 0%	56 0%	
▼context in function() @100511	6169	72 0%	72 0%	
▼notAClosure in @100507	6168	40 0%	40 0%	
▼HUGE in system / Context @100505	6167	56 0%	56 0%	
▼context in function() @100503	6166	72 0%	72 0%	
▼notAClosure in @100499	6165	40 0%	40 0%	
▼HUGE in system / Context @100497	6164	56 0%	56 0%	
▼context in function() @100495	6163	72 0%	72 0%	
▼notAClosure in @100491	6162	40 0%	40 0%	
▼HUGE in system / Context @100489	6161	56 0%	56 0%	
▼context in function() @100487	6160	72 0%	72 0%	
▼notAClosure in @100483	6159	40 0%	40 0%	
▼HUGE in system / Context @100481	6158	56 0%	56 0%	
▼context in function() @100479	6157	72 0%	72 0%	
▼notAClosure in @100475	6156	40 0%	40 0%	

Leaks caused by closures

❖ --trace_gc (print one trace line following each garbage collection)

Spent on external memory

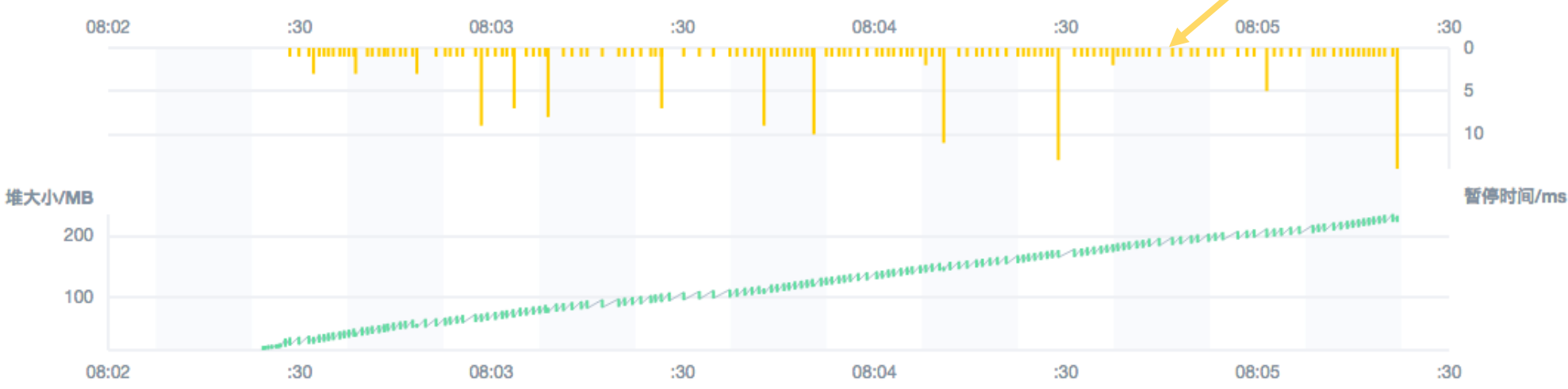
PID:Address	The time since you start this VM	Type of GC	Size of all objects	Size of memory allocated from OS	Pause		
19642:0x102004a00	611604 ms	Scavenge	1153.6	1457.9	-> 1152.9	1457.9 MB, 0.6 / 0 ms	[allocation failure].
[19642:0x102004a00]	611774 ms	Scavenge	1153.7	(1457.9)	-> 1152.9	(1457.9) MB, 0.6 / 0 ms	[allocation failure].
[19642:0x102004a00]	611775 ms	Scavenge	1153.7	(1457.9)	-> 1153.0	(1457.9) MB, 0.6 / 0 ms	[allocation failure].
[19642:0x102004a00]	611880 ms	Scavenge	1153.1	(1457.9)	-> 1153.1	(1457.9) MB, 0.5 / 0 ms	[allocation failure].
[19642:0x102004a00]	611881 ms	Scavenge	1153.1	(1457.9)	-> 1153.1	(1457.9) MB, 0.5 / 0 ms	[allocation failure].
[19642:0x102004a00]	611916 ms	Mark-sweep	1153.1	(1457.9)	-> 1152.4	(1449.9) MB, 35.0 / 0 ms	[last resort gc].
[19642:0x102004a00]	611951 ms	Mark-sweep	1152.4	(1449.9)	-> 1152.3	(1442.9) MB, 34.8 / 0 ms	[last resort gc].

Type of GC

Size of all objects

Pause

Because the GC is triggered when there's not enough memory



Leaks caused by closures

- ❖ `--trace_gc_nvp` (print one detailed trace line in name=value format after each garbage collection)

Name-value

Type of GC, ms=Mark-sweep/Mark-compact, s=Scavenger

```
[12543:0x102004a00] [I:0x102004a00] 152163 ms: pause=3.2 mutator=729.4 gc=ms external=0.0 mark=0.3 sweep=2.27
sweepns=1.53 sweepos=0.15 sweepcode=0.18 sweepcell=0.00 sweepmap=0.05 evacuate=0.0 new_new=0.0 root_new=0.0
old_new=0.0 compaction_ptrs=0.0 intracompaction_ptrs=0.0 misc_compaction=0.3 weak_closure=0.0 inc_weak_closure=0.0
weakcollection_process=0.0 weakcollection_clear=0.0 weakcollection_abort=0.0 total_size_before=37492960
total_size_after=25744424 holes_size_before=2600536 holes_size_after=3375144 allocated=11756832 promoted=1407928
semi_space_copied=1119104 nodes_died_in_new=12 nodes_copied_in_new=1 nodes_promoted=0 promotion_ratio=10.6%
average_survival_ratio=17.5% promotion_rate=97.4% semi_space_copy_rate=8.4% new_space_allocation_throughput=14433
context_disposal_rate=0.0 steps_count=51 steps_took=7.2 longest_step=0.9 incremental_marking_throughput=2560955
```

```
[12543:0x102004a00] Memory allocator, used: 65476 KB, available: 1433660 KB
[12543:0x102004a00] New space,      used:   1105 KB, available:  15018 KB, committed:   32248 KB
[12543:0x102004a00] Old space,      used:  22174 KB, available:   2622 KB, committed:  25049 KB
[12543:0x102004a00] Code space,     used:   1617 KB, available:    622 KB, committed:   2266 KB
[12543:0x102004a00] Map space,      used:    243 KB, available:   810 KB, committed:   1070 KB
[12543:0x102004a00] Large object space, used:     0 KB, available: 1432619 KB, committed:     0 KB
[12543:0x102004a00] All spaces,      used:  25141 KB, available: 1451694 KB, committed:   60634 KB
[12543:0x102004a00] External memory reported:    16 KB
[12543:0x102004a00] Total time spent in GC : 32.5 ms
```

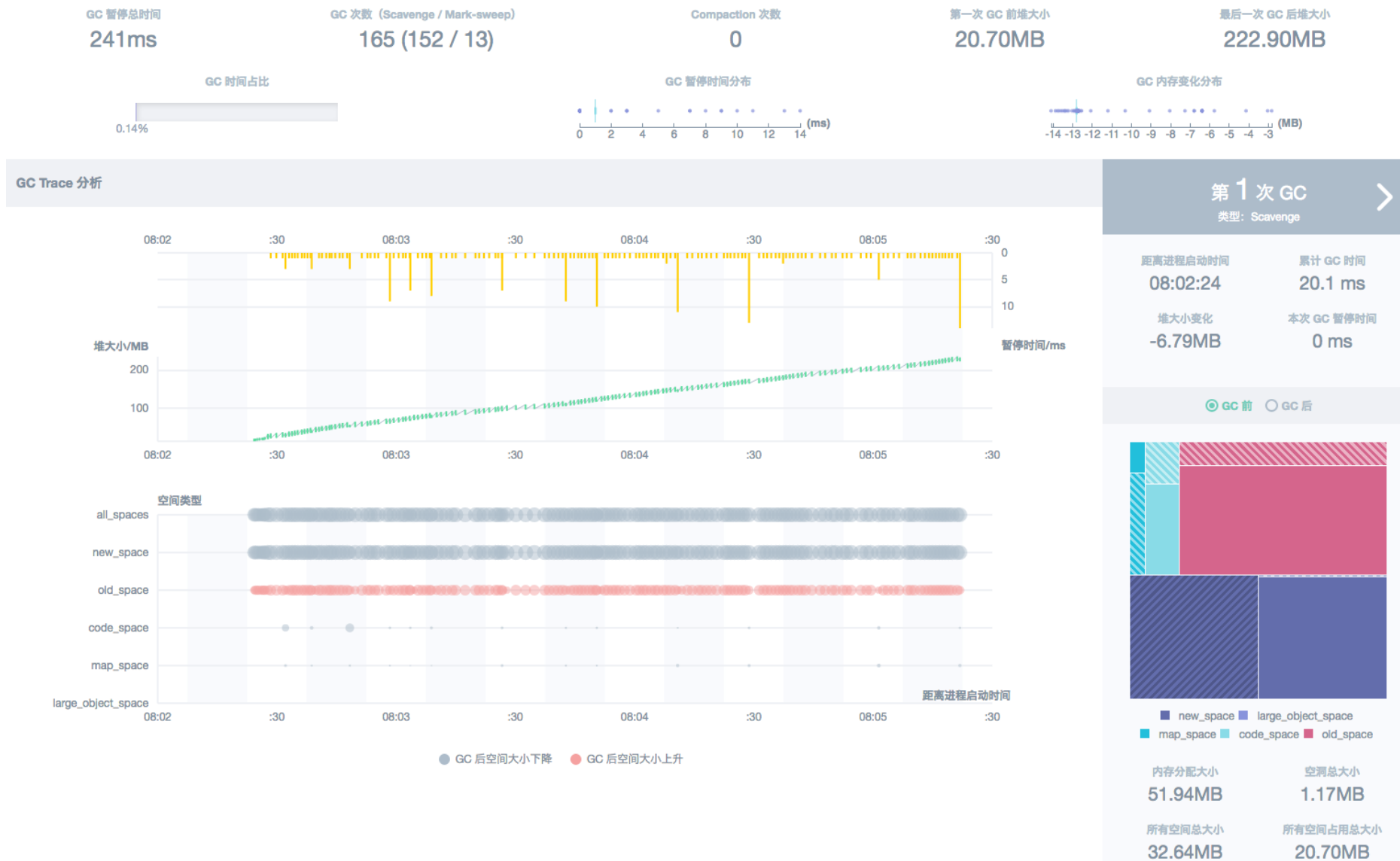
Allocatable memory and page headers

Memory left in the allocator

$\text{committed} = (\text{available} + \text{used})(*2 \text{ if it's new space})$

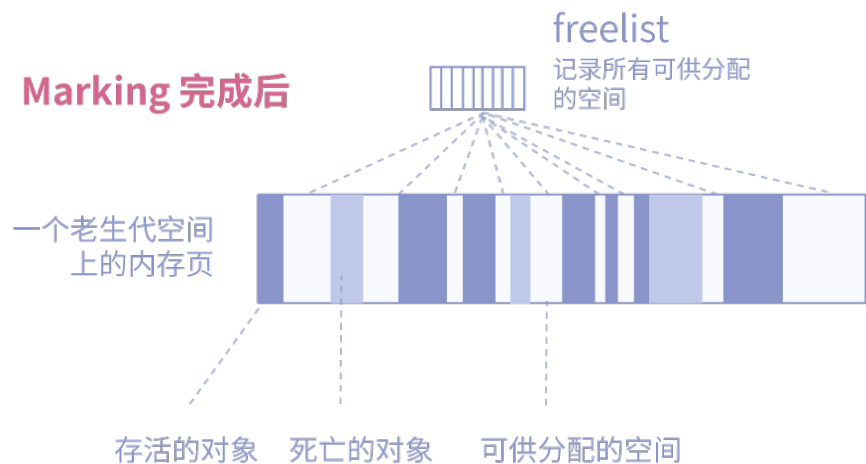
Statistics of each space

Leaks caused by closures

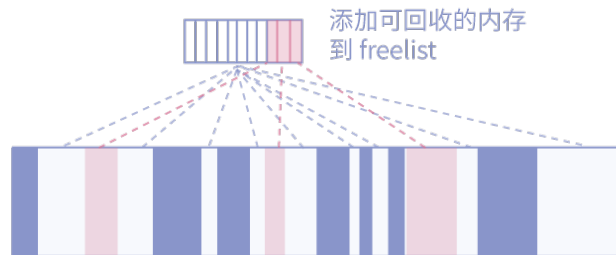


Sweeping 与 Compacting

Marking 完成后

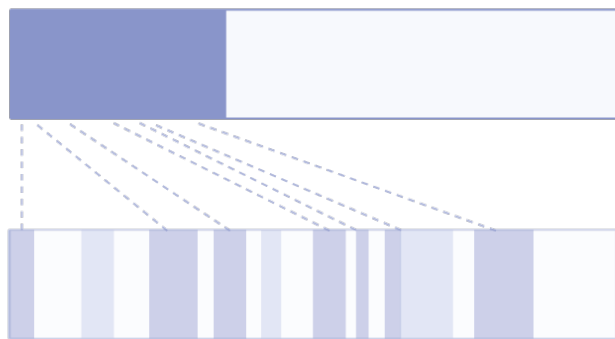


Sweeping

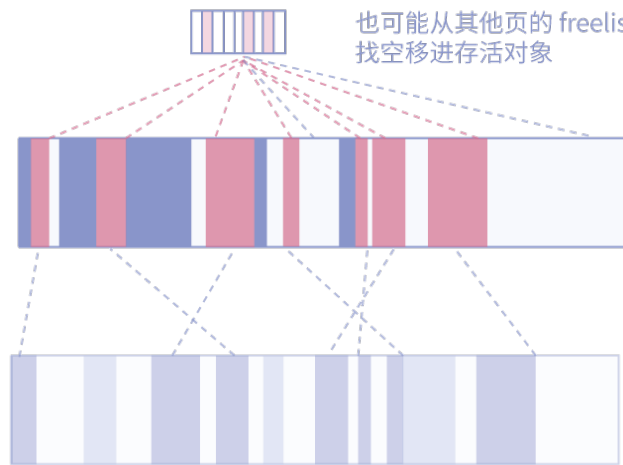


Compacting

可能移动存活对象到另一页

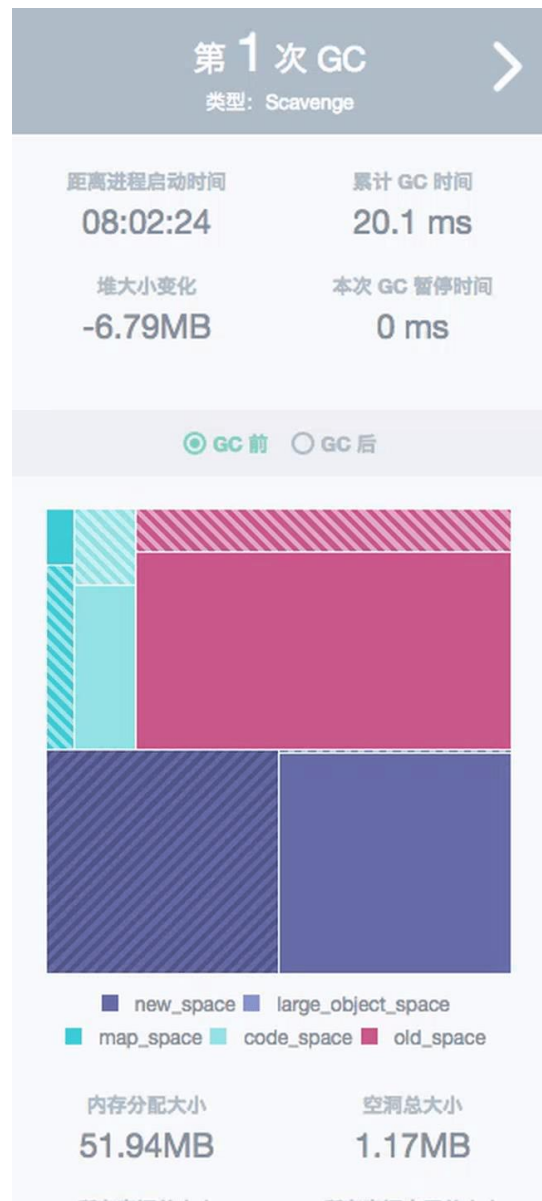


也可能从其他页的 freelist 找空移进存活对象



清空后的内存页可以还给操作系统

Leaks caused by closures



Leaks caused by closures

```
FATAL ERROR: CALL_AND_RETRY_LAST Allocation failed - process out of memory
Abort trap: 6
```

❖ deps/v8/src/api.cc: ResourceConstraints::ConfigureDefaults

```
const uint64_t low_limit = 512ul * i::MB;
const uint64_t medium_limit = 768ul * i::MB;
const uint64_t high_limit = 1ul * i::GB;
```

❖ deps/v8/src/heap/heap.h

```
// The old space size has to be a multiple of Page::kPageSize.
// Sizes are in MB.
static const int kMaxOldSpaceSizeLowMemoryDevice = 128 * kPointerMultiplier;
static const int kMaxOldSpaceSizeMediumMemoryDevice =
    ....256 * kPointerMultiplier;
static const int kMaxOldSpaceSizeHighMemoryDevice = 512 * kPointerMultiplier;
static const int kMaxOldSpaceSizeHugeMemoryDevice = 700 * kPointerMultiplier;

static const int kPointerMultiplier = i::kPointerSize / 4;

const int kPointerSize = sizeof(void*); // NOLINT
```

Leaks caused by closures

GC 暂停总时间
821ms

GC 次数 (Scavenge / Mark-sweep)
1705 (1684 / 21)

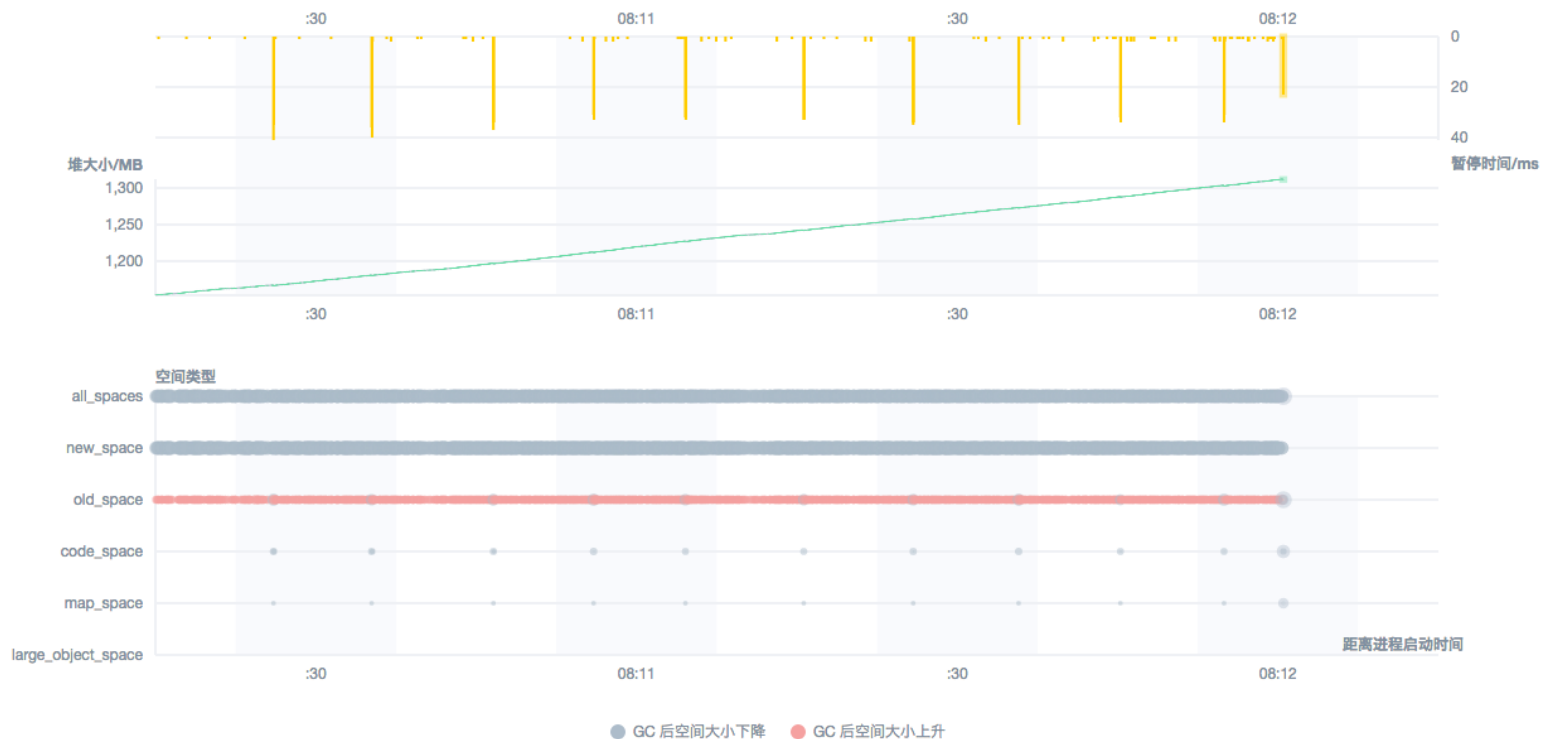
Compaction 次数
1

第一次 GC 前堆大小
1.13GB

最后一次 GC 后堆大小
1.28GB



GC Trace 分析



< 第 1705 次 GC

类型: Mark-sweep

距离进程启动时间
08:12:00

累计 GC 时间
4945.6 ms

堆大小变化
-436.70KB

本次 GC 暂停时间
23 ms

GC 前 GC 后



new_space large_object_space

map_space code_space old_space

内存分配大小
1.42GB

空闲总大小
87.10MB

所有空间总大小
1.37GB

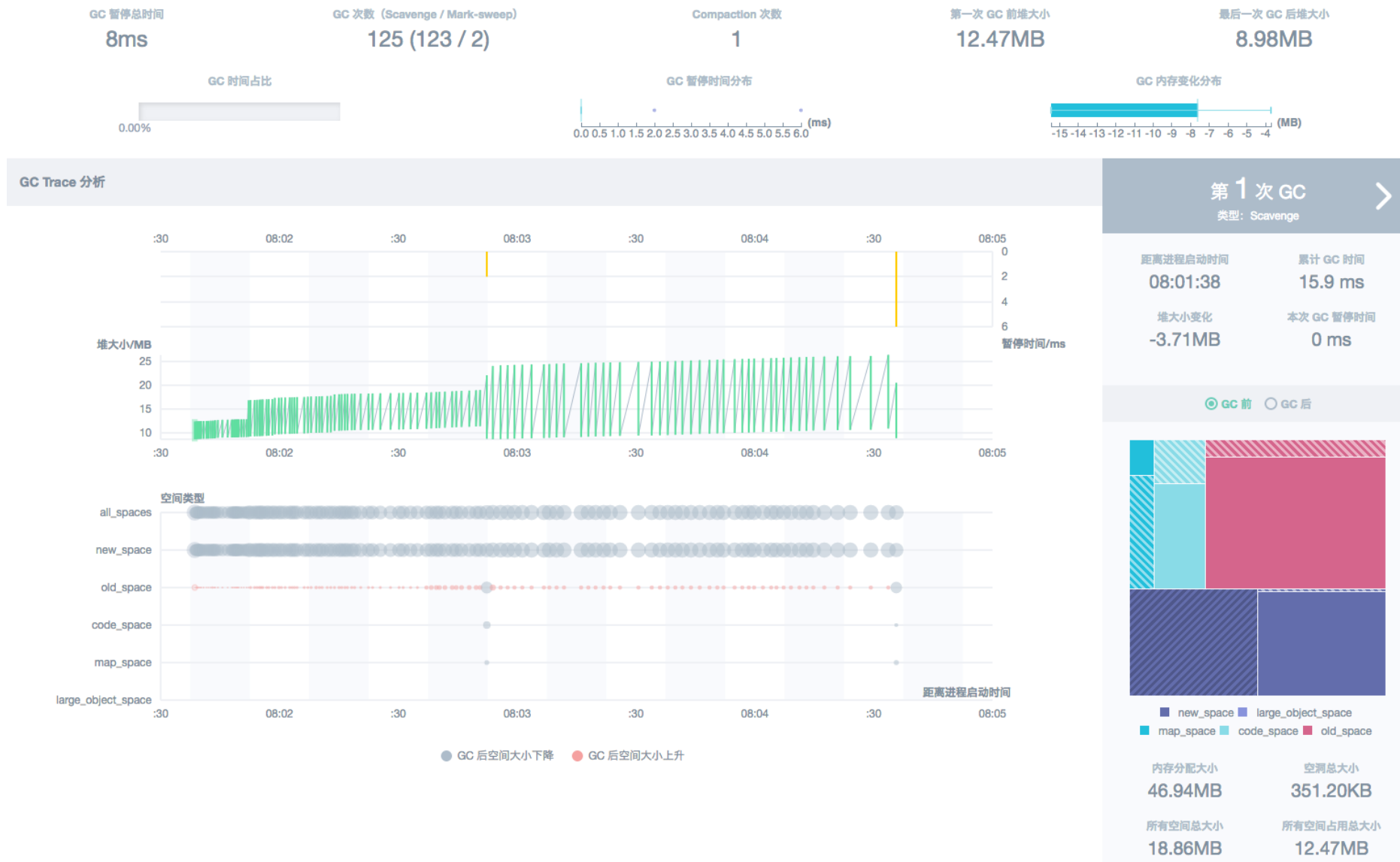
所有空间占用总大小
1.28GB

Solution #1: nullify your reference

- ❖ Cut off the reference to the old GIANT when you don't need it anymore

```
var GIANT;  
  
function leak() {  
  var HUGE = GIANT;  
  
  function unusedClosure() {  
    HUGE.slice(1);  
  }  
  
  GIANT = {  
    willBeLeaked: new Array(1e5).join('.'),  
    notAClosure: function notAClosure() {  
      return 1;  
    }  
  }  
  
  HUGE = null; /* not used anymore! */  
}
```

Solution #1: nullify your reference

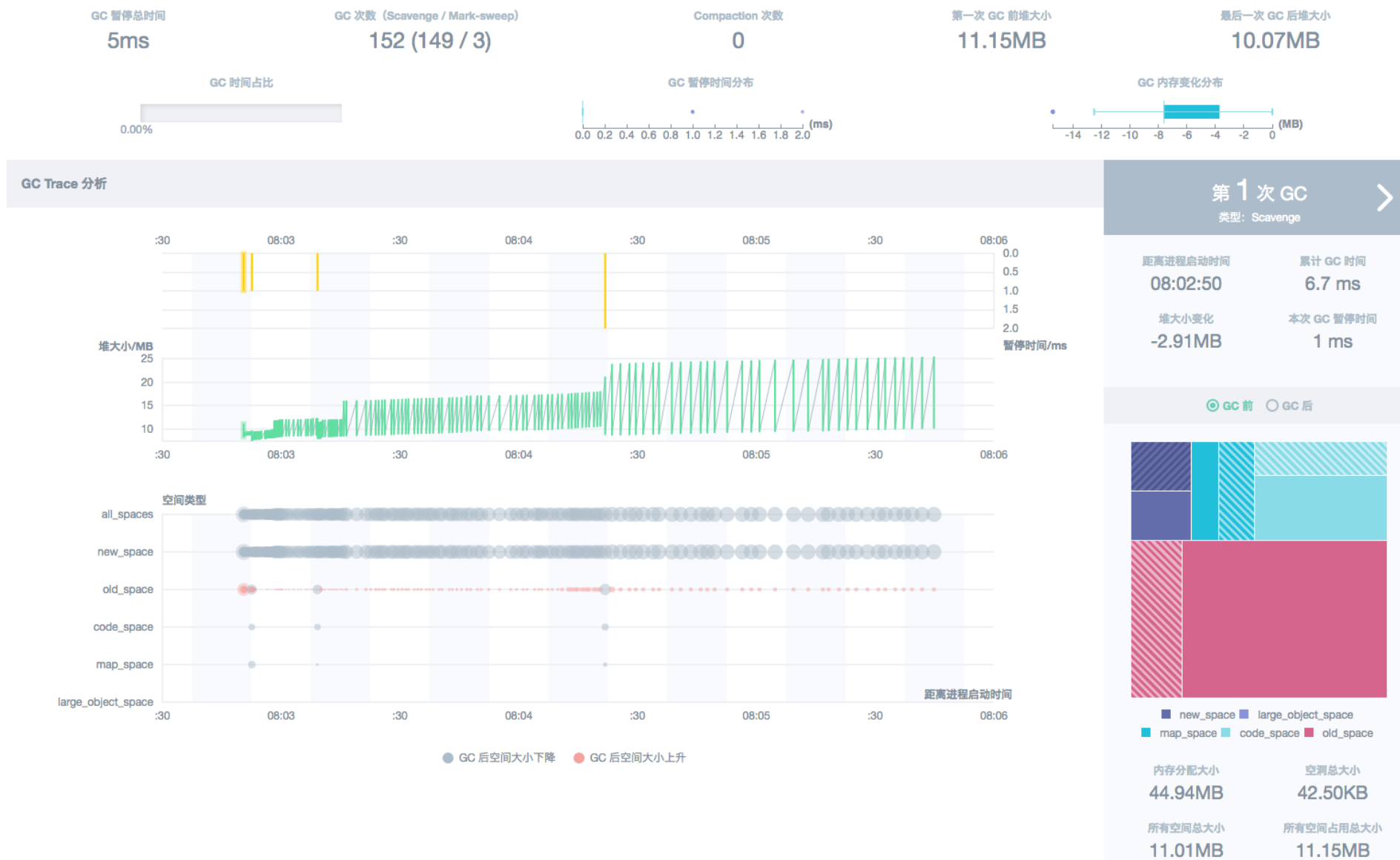


Solution #2: passing parameters

- ❖ Avoid closure like a plague

```
var GIANT;  
  
function leak() {  
    var HUGE = GIANT;  
  
    function notAClosureAnymore(HUGE_ARG) {  
        HUGE_ARG.slice(1);  
    }  
  
    GIANT = {  
        willBeLeaked: new Array(1e5).join('.'),  
        notAClosure: function notAClosure() {  
            return 1;  
        }  
    }  
}
```


Solution #2: passing parameters



Questions?



<https://alinode.aliyun.com>
qiuyi.zqy@alibaba-inc.com
@joyeecheung (GitHub, Twitter)