唯快不破

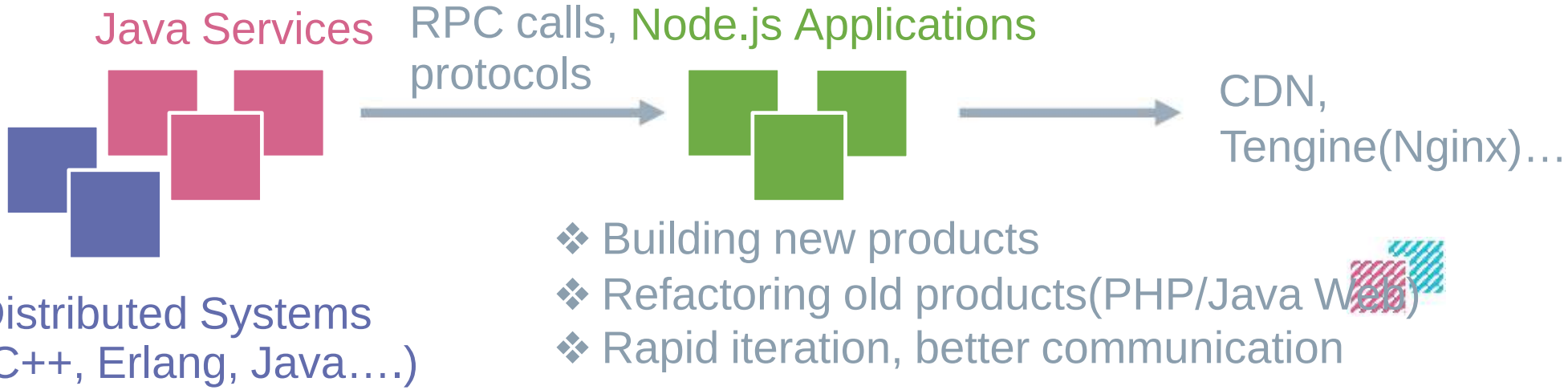
高效定位线上 Node.js 应用内存泄漏

关于我

- @hyj1991 (GitHub, CNode)
- @黄一君，*Easy-Monitor* 作者
- @阿里云计算有限公司，高级开发工程师，*Node.js* 性能平台

背景

- ❖作为中间层，前后端分离
- ❖长连接，纯服务端应用
- ❖NW.js、Electron 等构建跨平台客户端



探究 V8 GC 过程

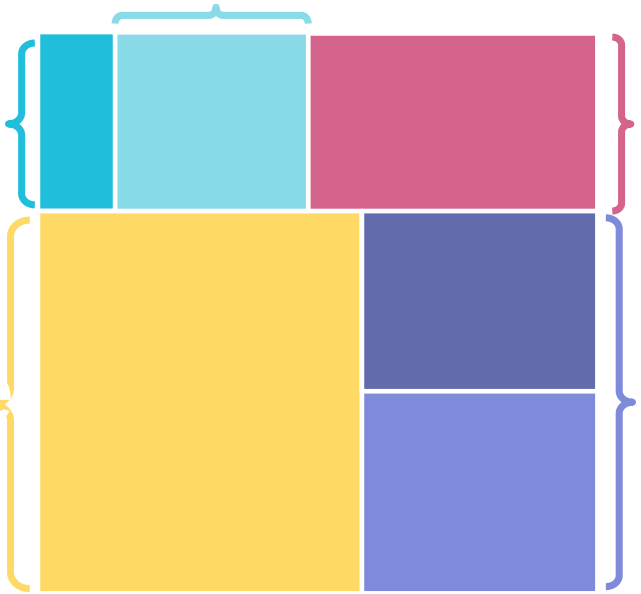
堆内内存划分

Code Spac

v8 编译后的可执行代码

Map Spac

Object 指向的
隐藏类元对象



Old Spac

在 new space 中经过两次 GC 依旧存活的对象晋升到 old space 中

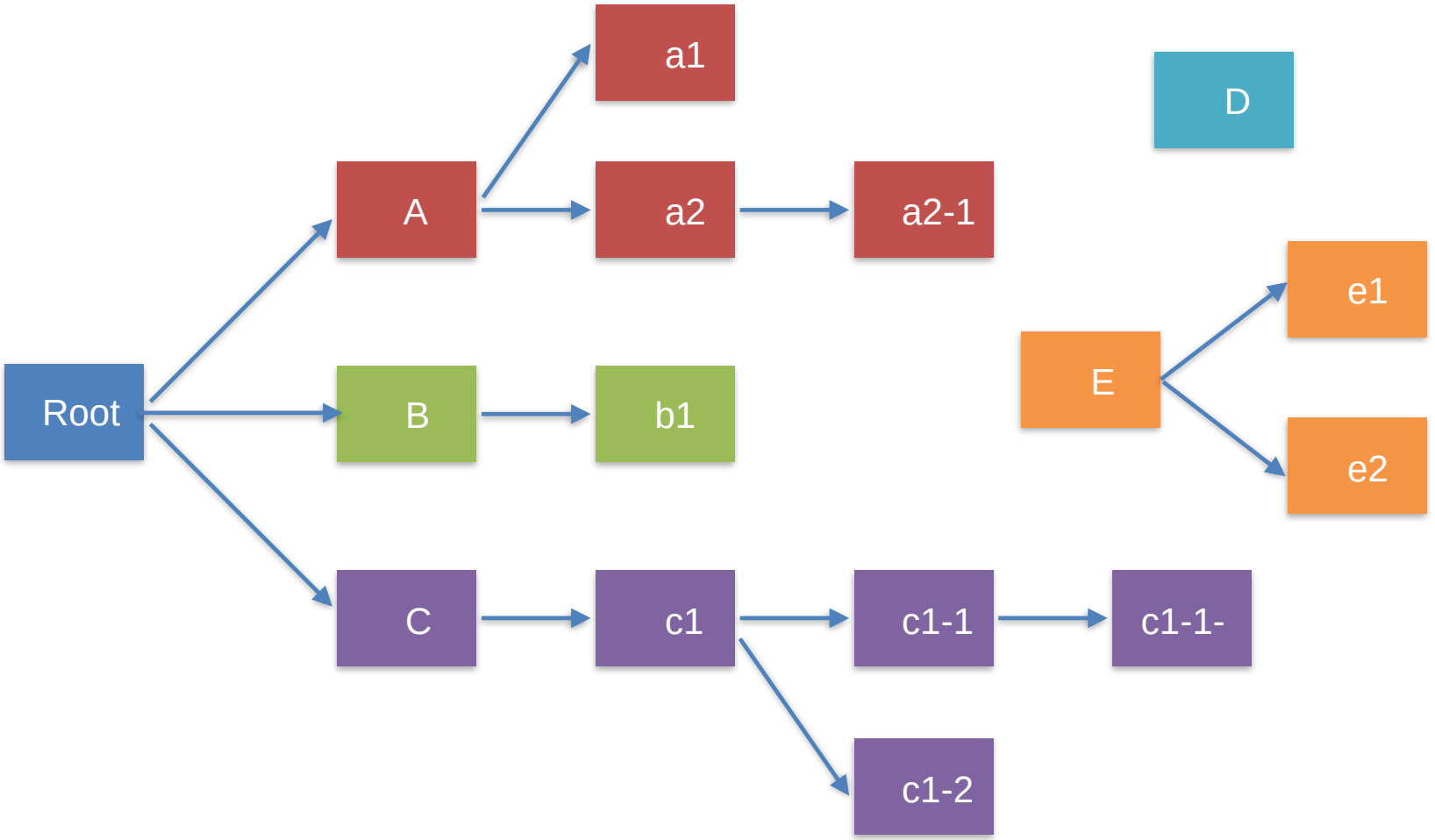
Large Object Spa

大对象 (大于 507136 byte)

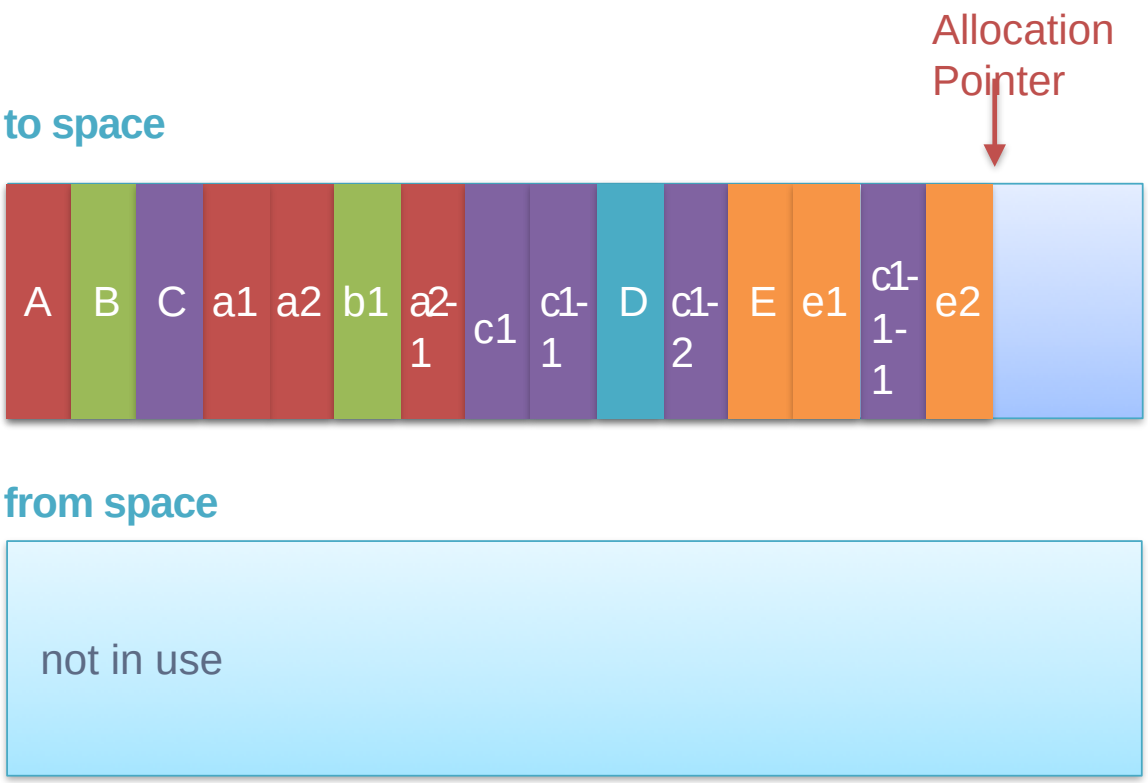
New Space

对半分割为两个部分，同一时刻只使用其中的一半，绝大部分对象的创建和销毁都在这里发生

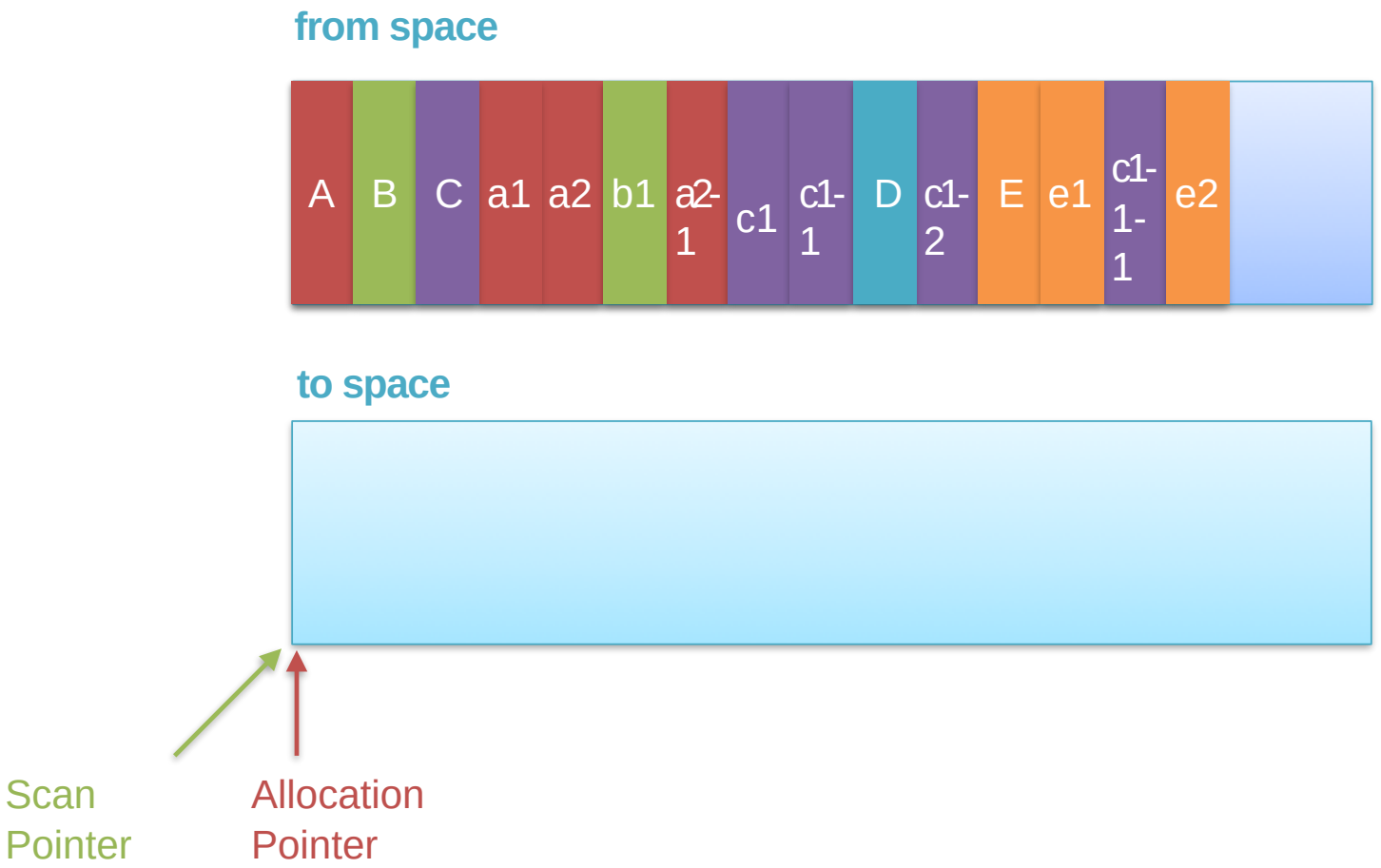
新生代（Scavenge 算法）



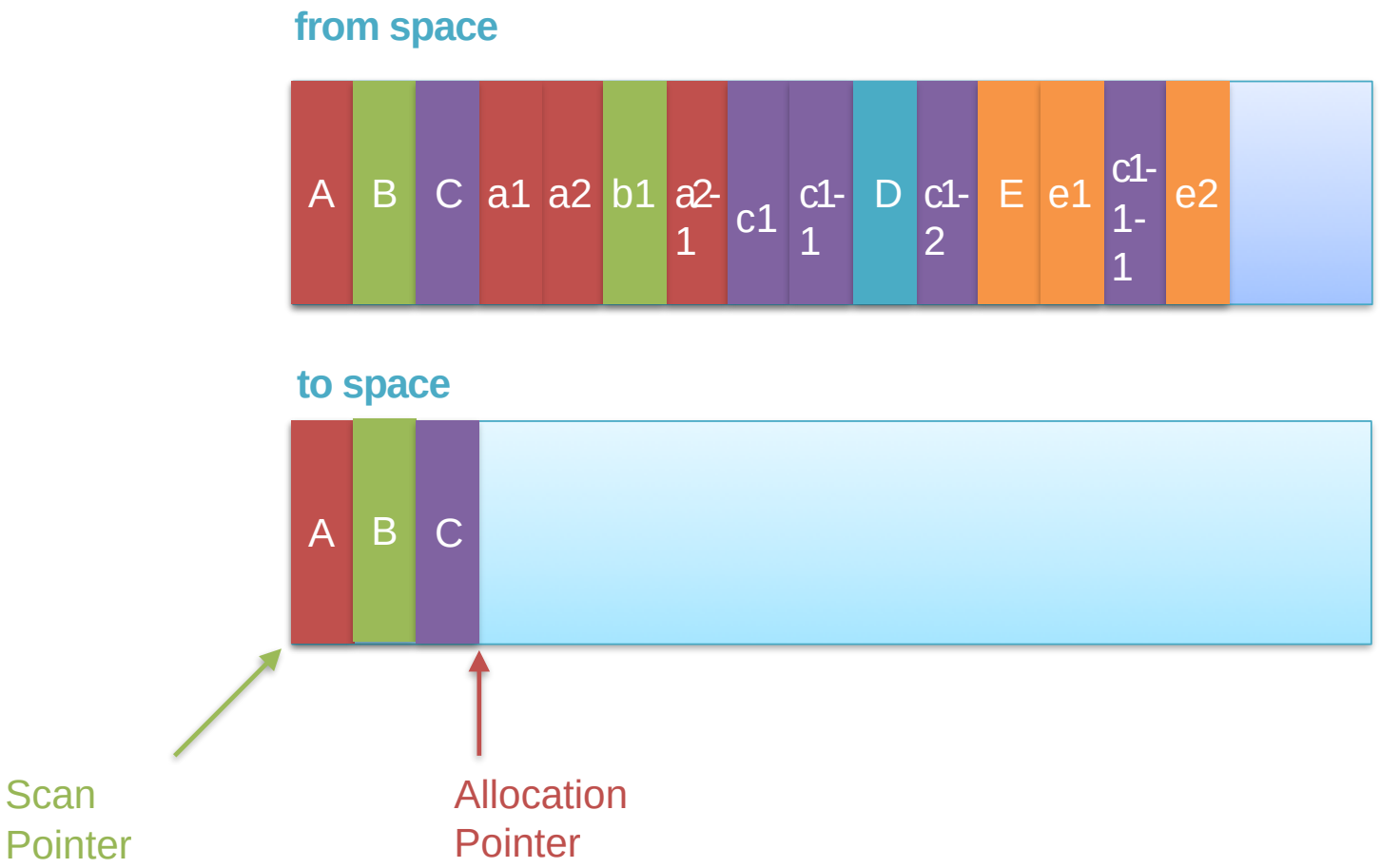
新生代（Scavenge 算法）



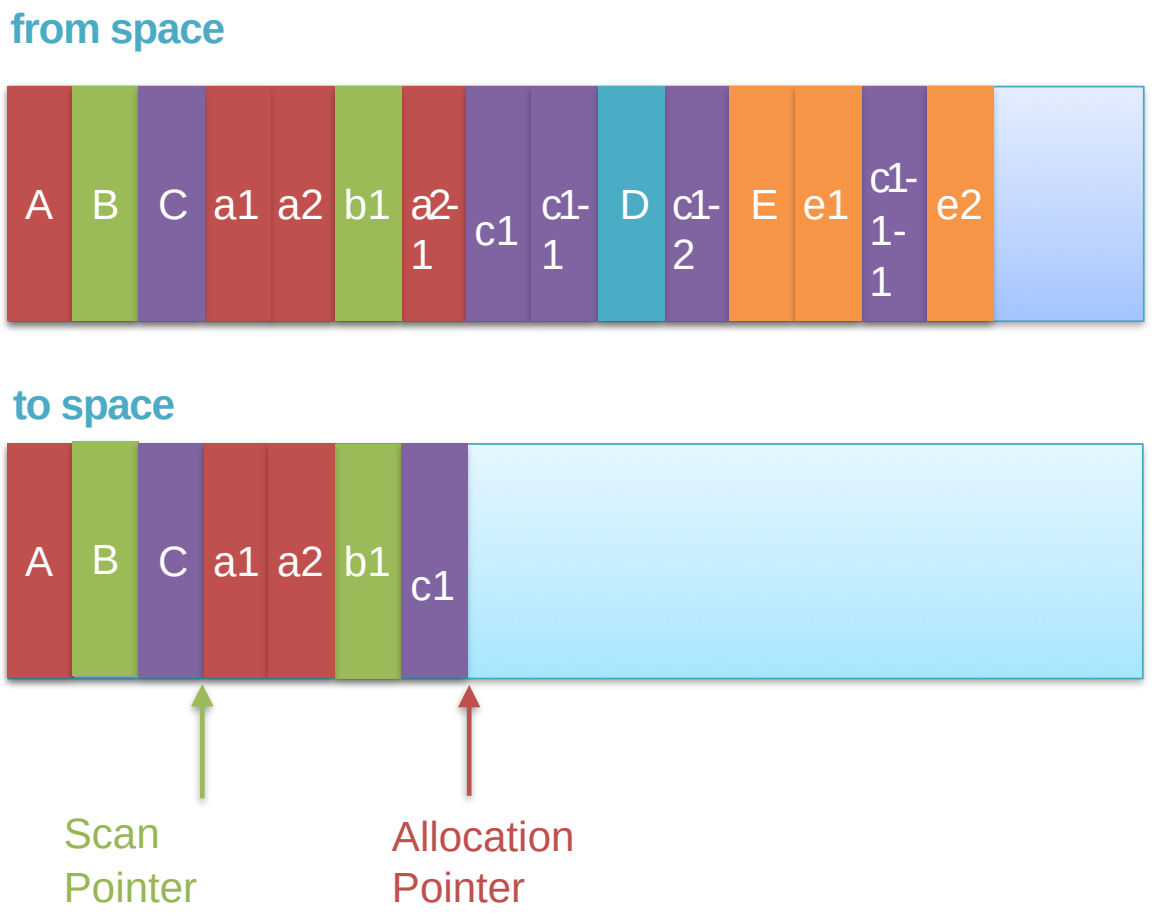
新生代（Flip）


from space
A
B
C
a1
a2
b1
a2-1
c1
c1-1
D
c1-2
E
e1
c1-1-1
e2
to space
Scan Pointer
Allocation Pointer

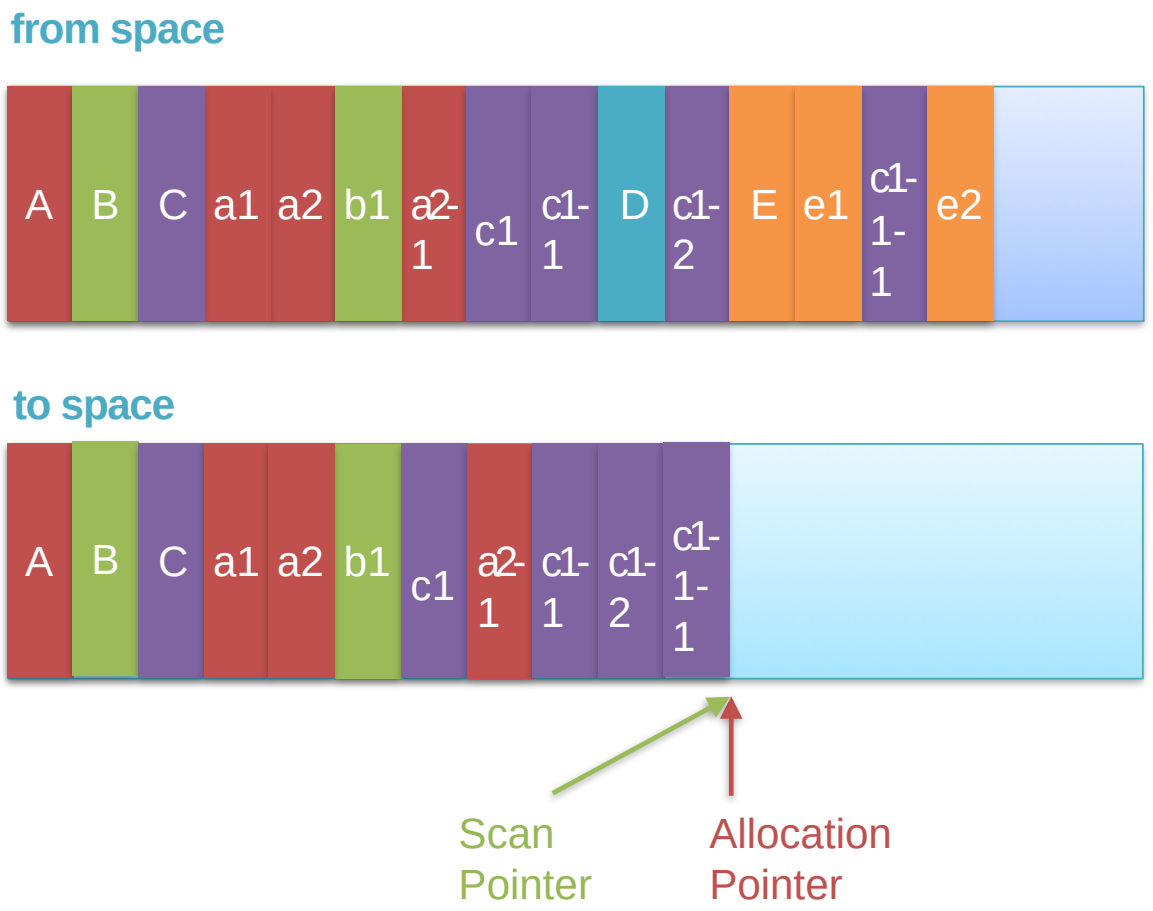
新生代（Copy Roots）



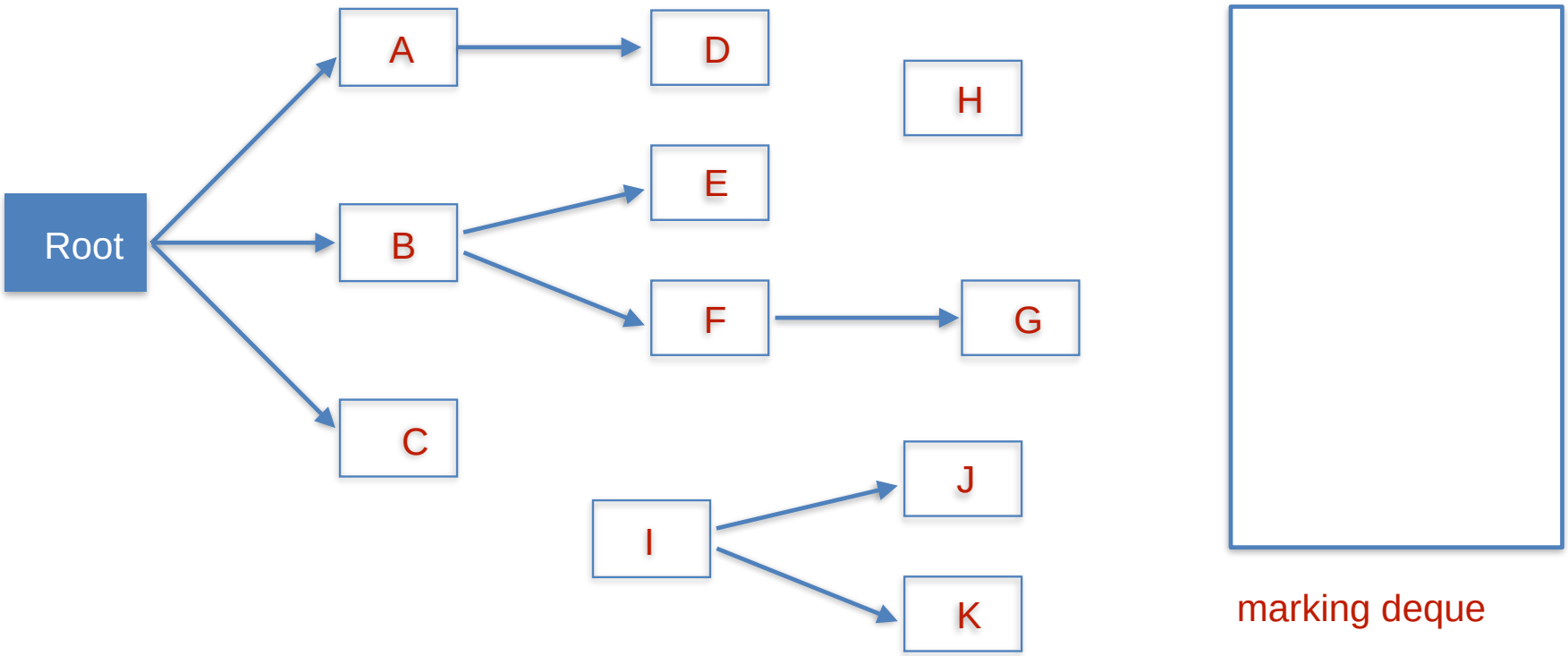
新生代（BFS）


from space
A
B
C
a1
a2
b1
a2-1
c1
c1-1
D
c1-2
E
e1
c1-1-1
e2
to space
A
B
C
a1
a2
b1
c1
Scan Pointer
Allocation Pointer

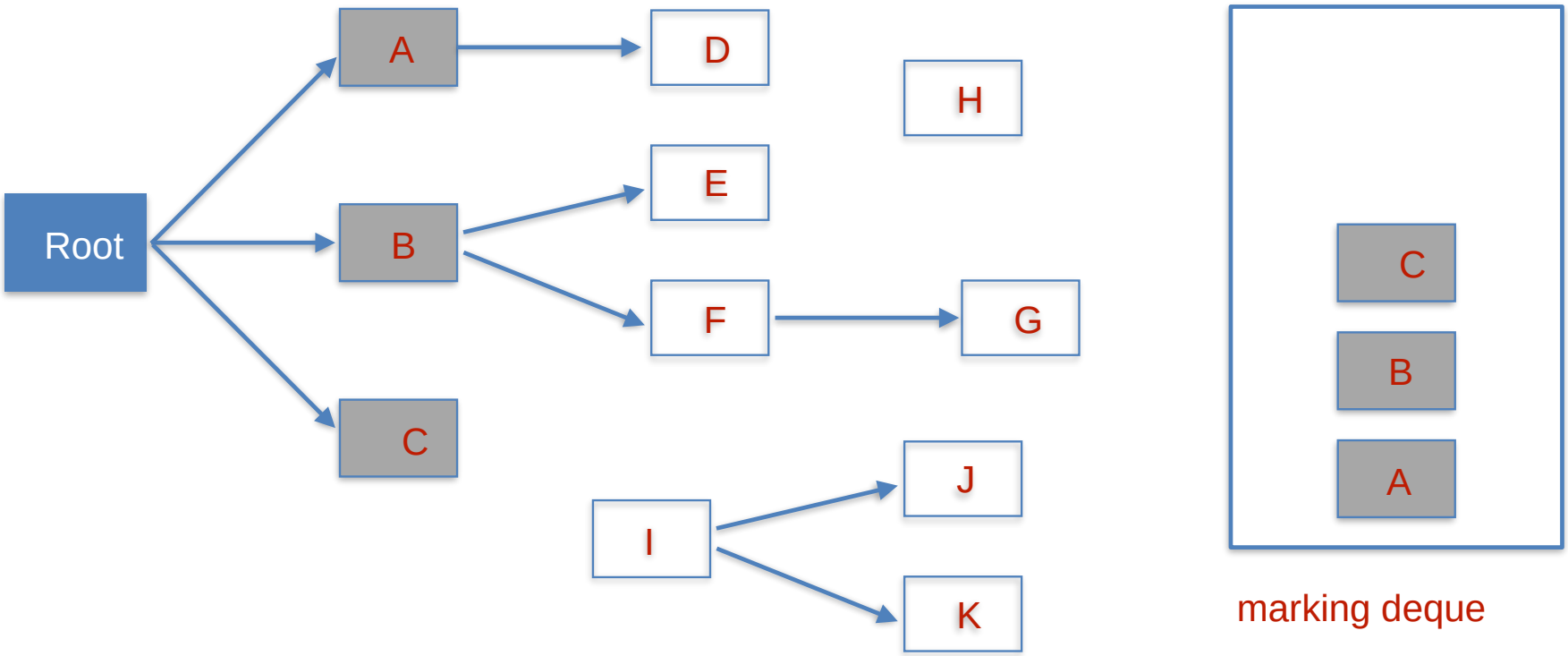
新生代（Scan 指针和 Allocation 指针重合）



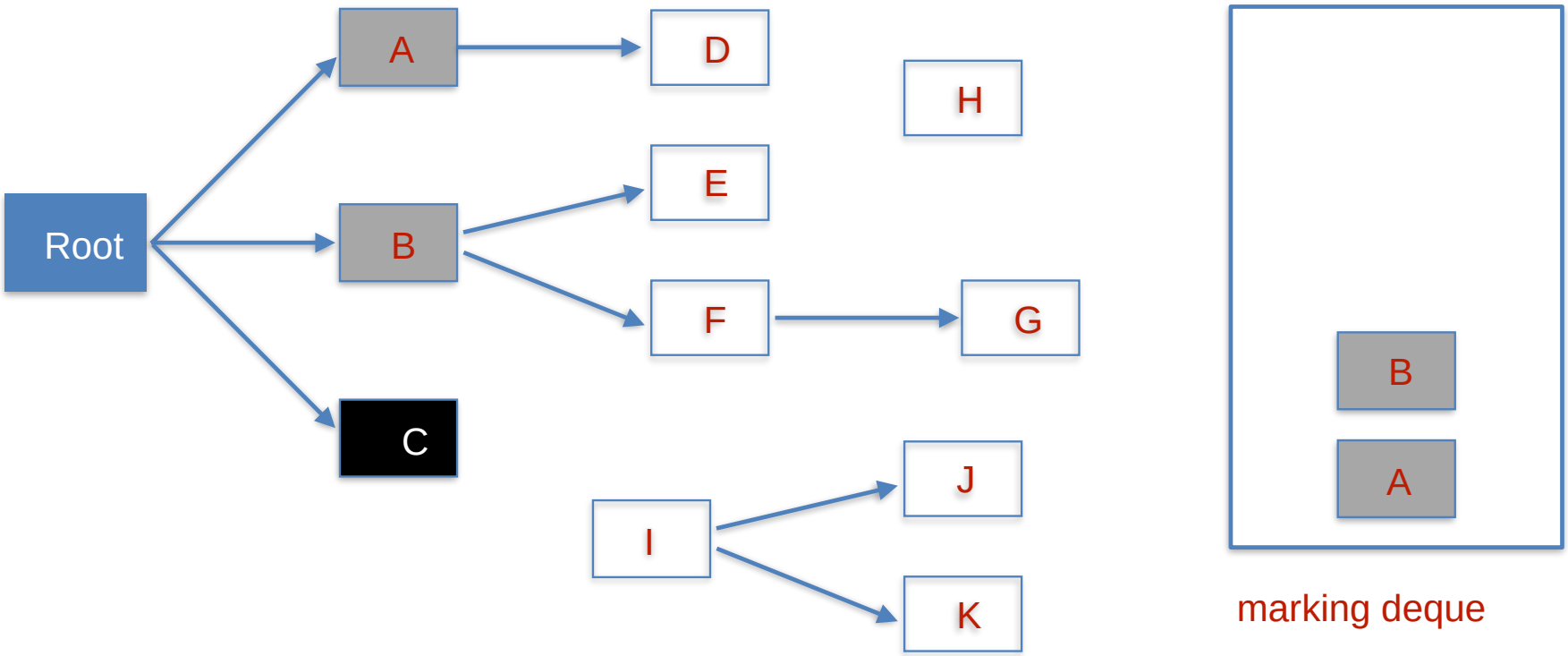
老生代（Mark-Sweep / Mark-Compact 算法）



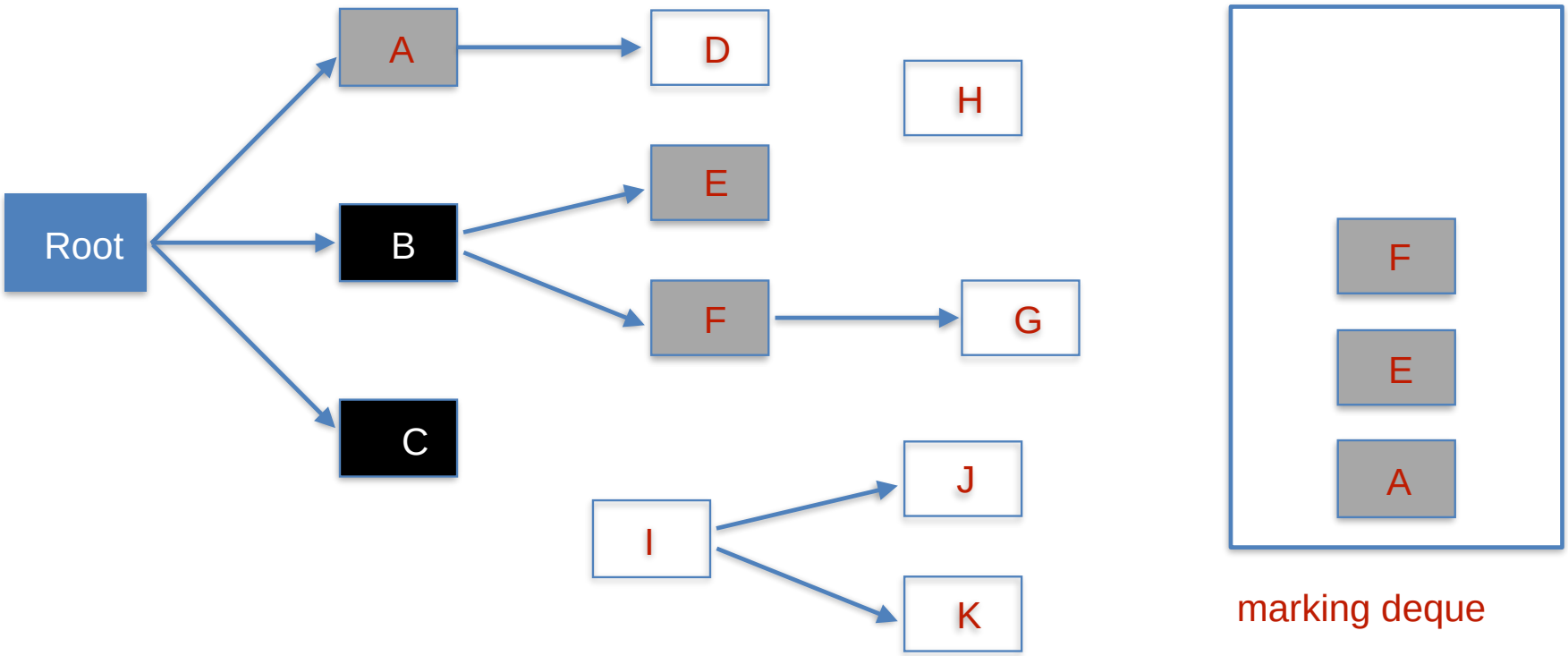
老生代（Mark-Sweep / Mark-Compact 算法）



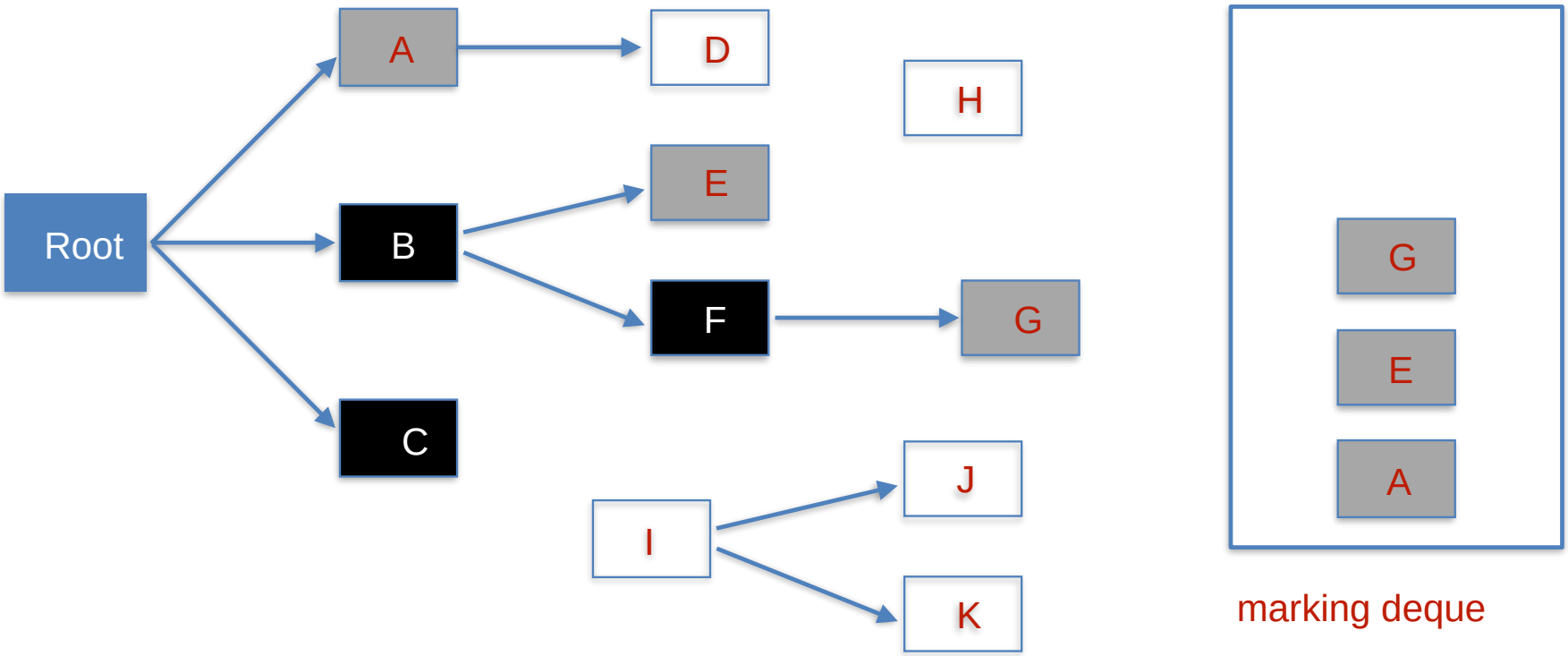
老生代（Mark-Sweep / Mark-Compact 算法）



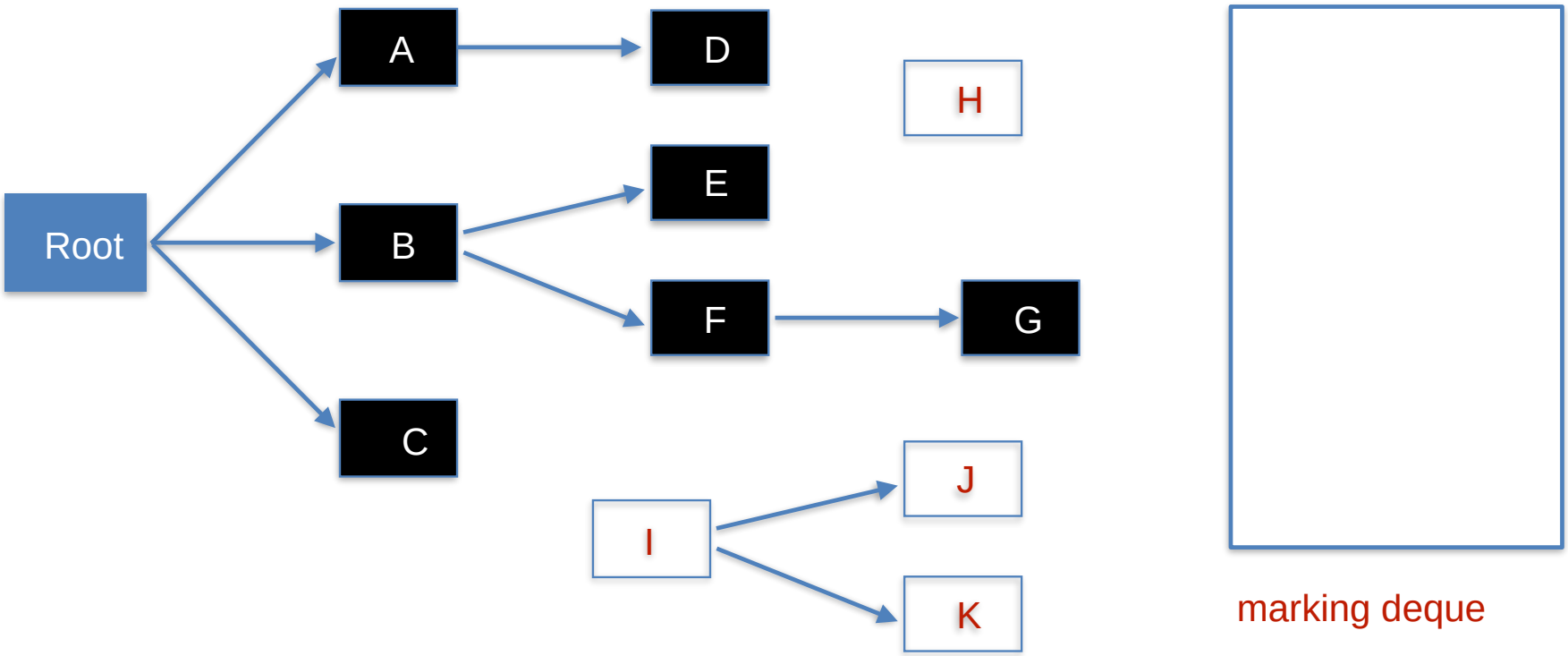
老生代（Mark-Sweep / Mark-Compact 算法）



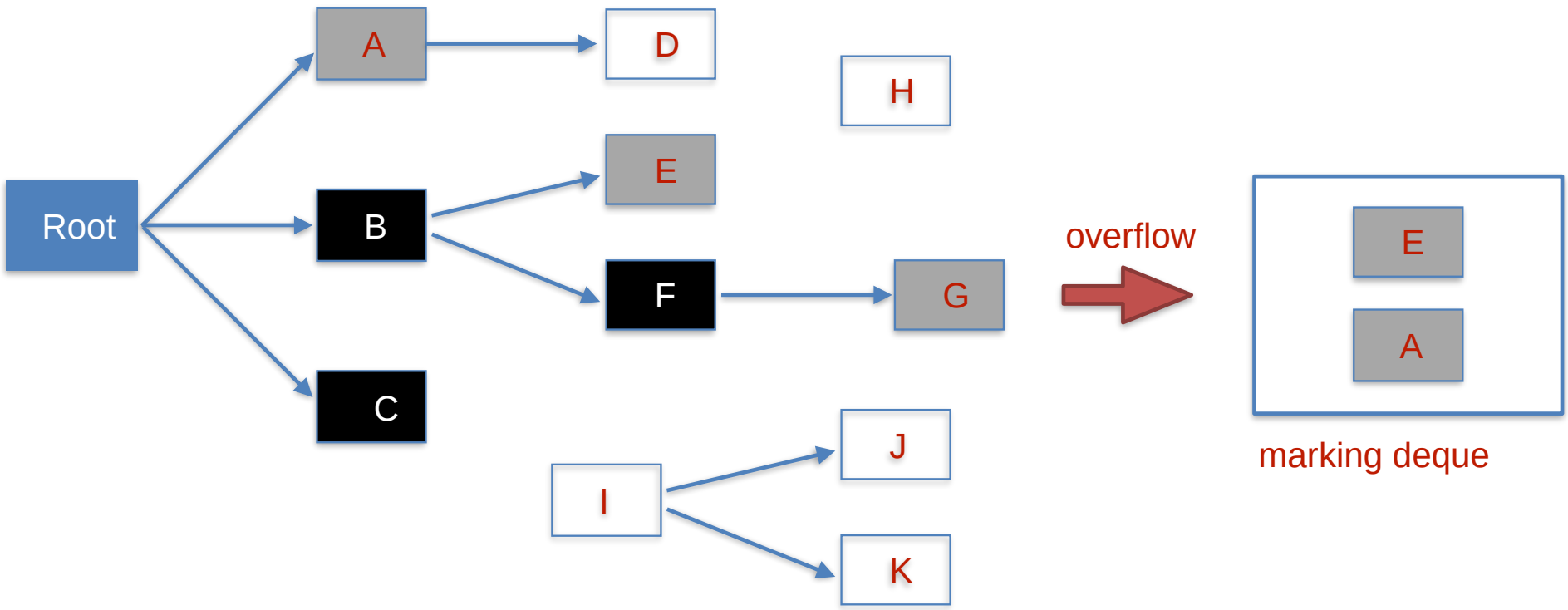
老生代（Mark-Sweep / Mark-Compact 算法）



老生代（Mark-Sweep / Mark-Compact 算法）



老生代（overflow）



先将 G 标记为灰色，但是不放入 marking queue，那么从 E 开始 pop，很快 marking queue 就会被清空；此时再遍历整个堆，找到灰色的对象放入 marking queue，继续原样标记执行

增量式标记

- 每次 Mark-Sweep 需要全量扫描整个堆，开销过大
- 堆达到一定大小时，执行增量标记 (incremental_marking)

探究 Heapsnapsho

什么是 Heapsnapshot

- Root 到应用运行生成的各个对象间的引用关系

▶2 :: global @1279	1	40	0 %	4 462 704 79 %
▼[1] :: (GC roots) @3	–	0	0 %	1 149 600 20 %
▶[3] :: (Strong roots) @9	–	0	0 %	331 112 6 %
▶[11] :: (Dispatch table) @25	–	0	0 %	305 496 5 %
▶[1] :: (Internalized strings) @5	–	0	0 %	136 744 2 %
▶[9] :: (Compilation cache) @21	–	0	0 %	79 624 1 %
▶[12] :: (Builtins) @27	–	0	0 %	79 280 1 %
▶[13] :: (Global handles) @29	–	0	0 %	28 264 1 %
▶[10] :: (Handle scope) @23	–	0	0 %	0 0 %
▶[14] :: (Eternal handles) @31	–	0	0 %	0 0 %
[15] :: (Thread manager) @33	–	0	0 %	0 0 %
[16] :: (Strong roots) @35	–	0	0 %	0 0 %
▶[17] :: (Extensions) @37	–	0	0 %	0 0 %
▶[2] :: (External strings) @7	–	0	0 %	0 0 %
[4] :: (Smi roots) @11	–	0	0 %	0 0 %
[5] :: (Bootstrapper) @13	–	0	0 %	0 0 %
▶[6] :: (Isolate) @15	–	0	0 %	0 0 %
[7] :: (Relocatable) @17	–	0	0 %	0 0 %
[8] :: (Debugger) @19	–	0	0 %	0 0 %

获取 Heapsnapshot (heapdump)

- 使用 writeSnapshot 按需获取堆快照
- 使用 kill -USR2 <pid> 按需获取堆快照

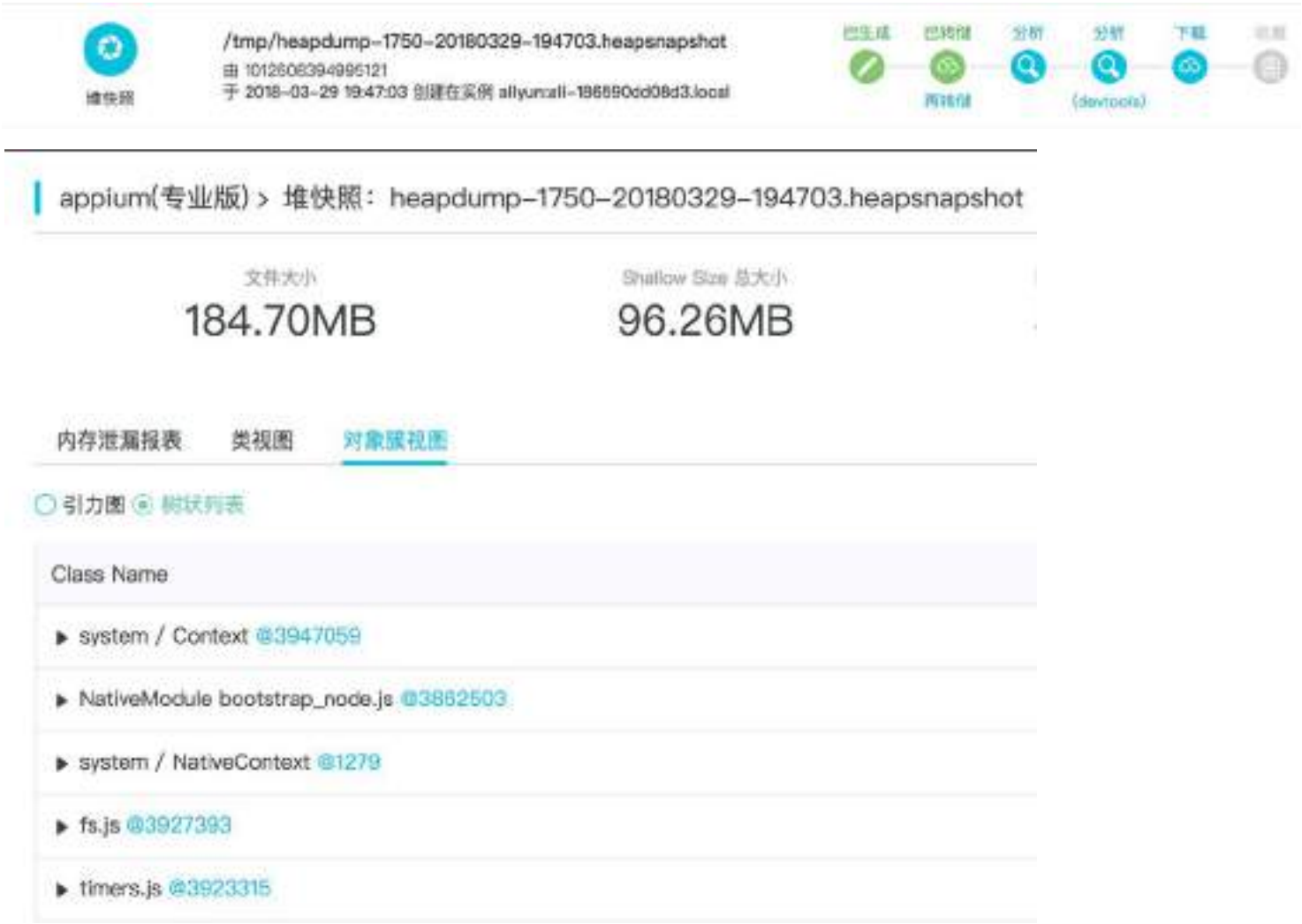
```
'use strict';
const path = require('path');
const heapdump = require('heapdump');
setTimeout(function () {
  heapdump.writeSnapshot(path.join(__dirname, './')
    + Date.now() + '.heapsnapshot');
}, 30 * 1000);
```

获取 Heapsnapshot (v8-profiler)

- 传入回调获取完整序列化堆快照
- 不传回调返回 transform 流式获取堆快照

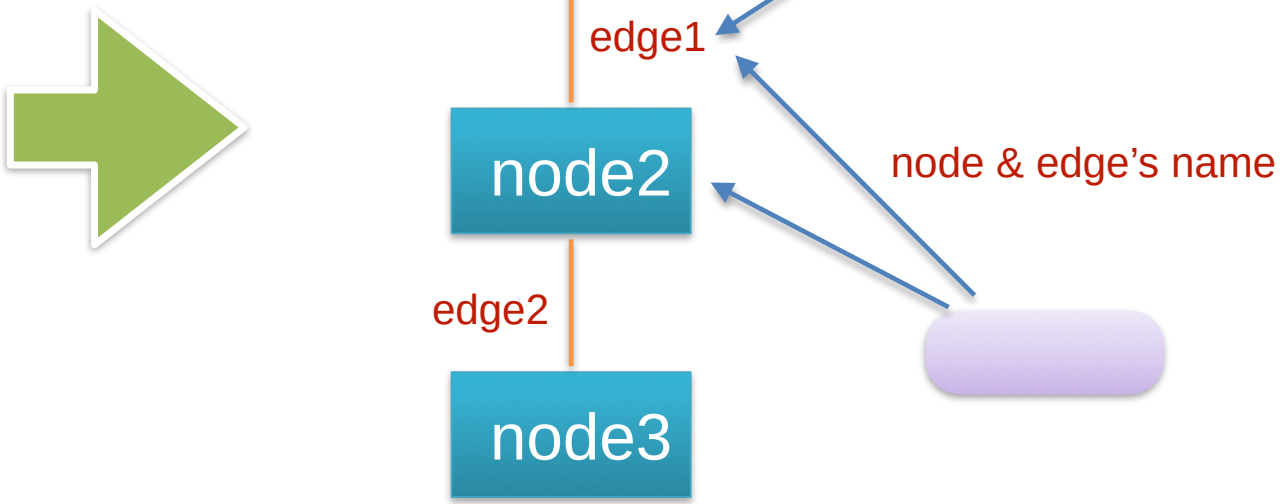
```
'use strict';
const path = require('path');
const v8Profiler = require('v8-profiler');

// 1. 直接获取序列化后的 snapshot
setTimeout(function () {
  const snapshot = v8Profiler.takeSnapshot();
  snapshot.export(function (error, result) {
    console.log(result);
    snapshot.delete();
  });
}, 30 * 1000);

// 2. 流式获取序列化 snapshot
setTimeout(function () {
  const snapshot = v8Profiler.takeSnapshot();
  const transform = snapshot.export();
  transform.on('data', data => console.log(data));
  //结束后删除
  transform.on('finish', snapshot.delete.bind(snapshot));
}, 30 * 1000);
```

获取 Heapsnapshot (Node.js 性能平台)



Heapsnapshot 数据结构详解

```
{
  snapshot: {}
  nodes: []
  edges: []
  strings: []
}
```



Heapsnapshot 数据结构详解 (snapshot)

❖ meta.node_fields: 长度为一个 node 实际长度，每一个元素代表其含

```
node_fields: [ 'type', 'name', 'id', 'self_size', 'edge_count', 'trace_node_id' ]
```

❖ meta.node_types: 每一个 node 中每一位的类型，第一位 type 是一个数

```
node_types: [ [Array], 'string', 'number', 'number', 'number', 'number', 'number' ]
```

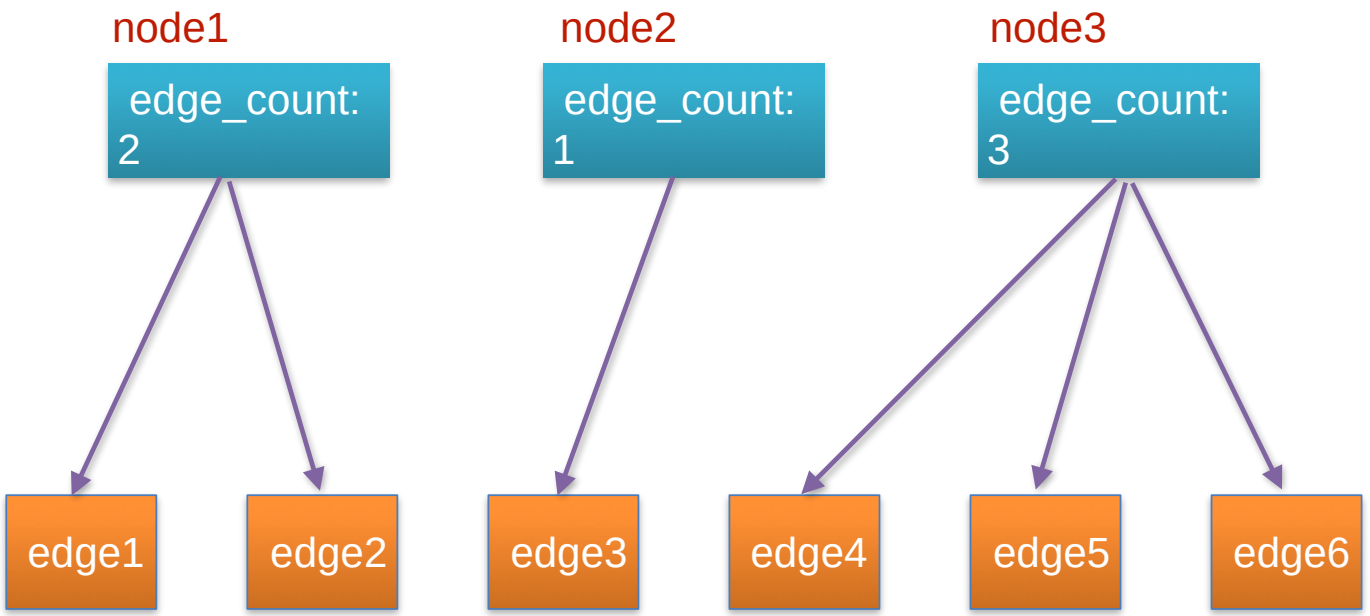
❖ meta.edge_fields: 长度为一个 edge 实际长度，每一个元素代表其含义

```
edge_fields: [ 'type', 'name_or_index', 'to_node' ]
```

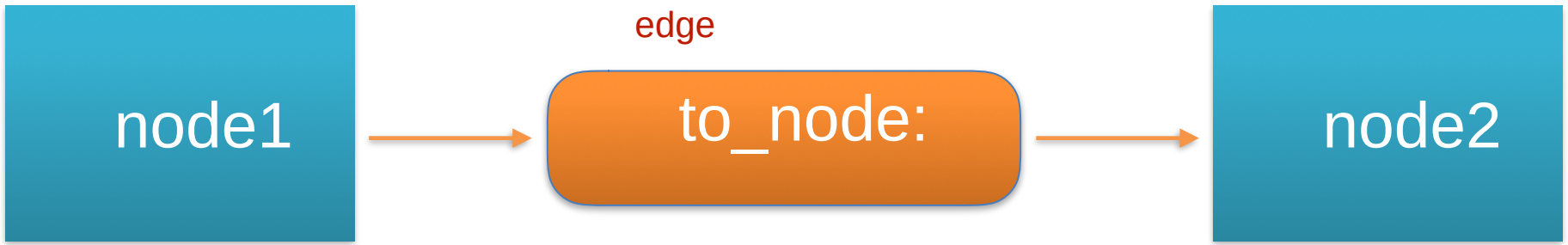
❖ meta.edge_types: 每一个 edge 中每一位的类型，第一位 type 是一个数

```
edge_types: [ [Array], 'string_or_number', 'node' ]
```

Heapsnapshot 数据结构详解（node 和 edge 的对应关系）



Heapsnapshot 数据结构详解 (node 和 node 的引用关系)



定位泄漏点（内存图）

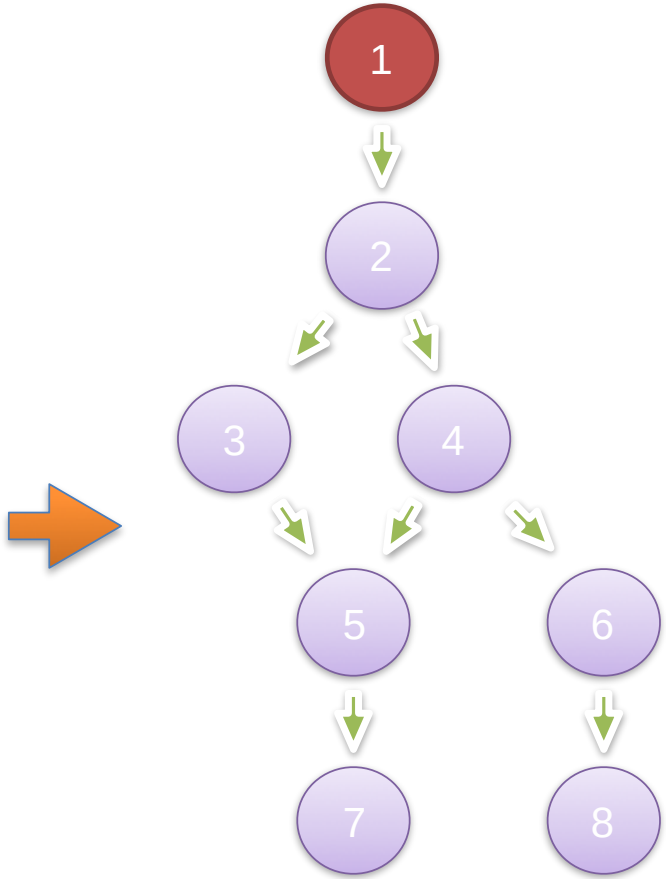
please input node address

Edges:

- SYNTTETICROOT @1 (type: synthetic, self_size: 0.00 Bytes)
  - [1] :: (GC roots) @3 (type: synthetic, self_size: 0.00 Bytes)
  - 2 :: global @1223 (type: object, self_size: 40.00 Bytes)
  - [3] :: SIGNALWRAP @2389960620 (type: synthetic, self_size: 0.00 Bytes)
  - [4] :: TTYWRAP @3958813572 (type: synthetic, self_size: 0.00 Bytes)
  - [5] :: HTTPPARSER @1050238604 (type: synthetic, self_size: 0.00 Bytes)
  - [6] :: TCPWRAP @689006112 (type: synthetic, self_size: 0.00 Bytes)

Retainers:

- SYNTTETICROOT @1 (type: synthetic, self_size: 0.00 Bytes)



定位泄漏点（支配树）

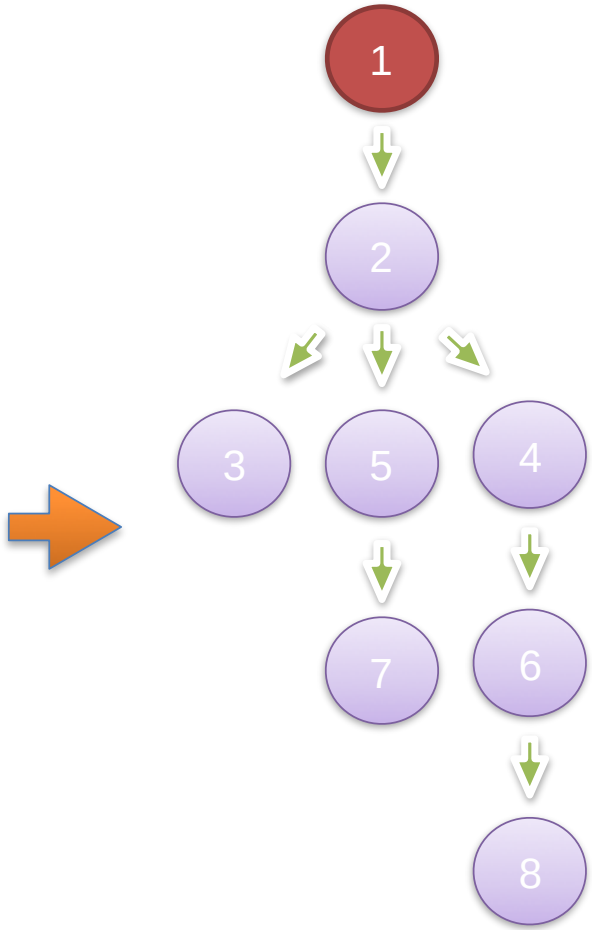
please input node address

Edges:

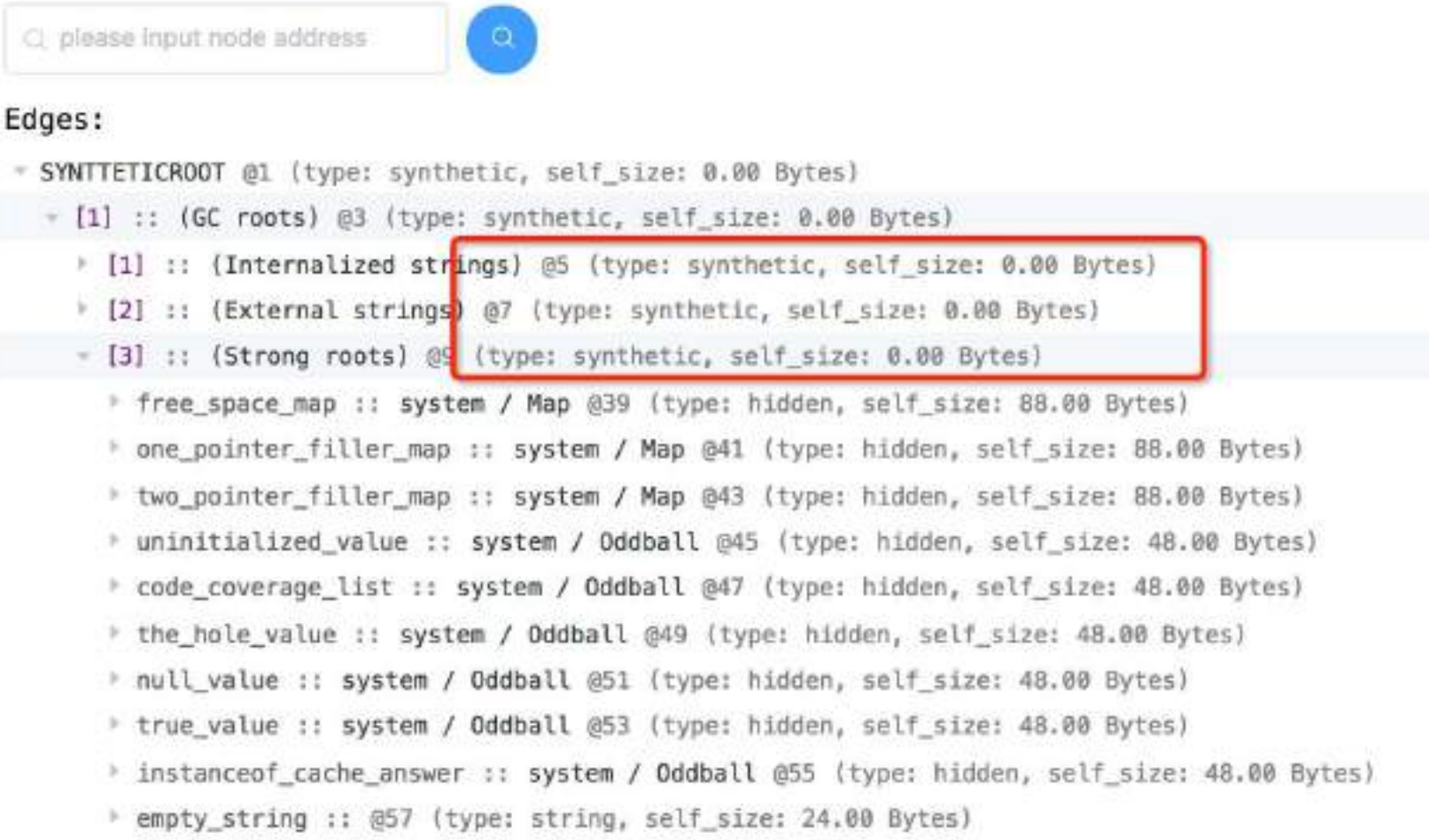
- SYNTTETICROOT @1 (type: synthetic, self_size: 0.00 Bytes)
  - [1] :: (GC roots) @3 (type: synthetic, self_size: 0.00 Bytes)
  - 2 :: global @1223 (type: object, self_size: 40.00 Bytes)
  - [3] :: SIGNALWRAP @2389960620 (type: synthetic, self_size: 0.00 Bytes)
  - [4] :: TTYWRAP @3958813572 (type: synthetic, self_size: 0.00 Bytes)
  - [5] :: HTTPPARSER @1050238604 (type: synthetic, self_size: 0.00 Bytes)
  - [6] :: TCPWRAP @689006112 (type: synthetic, self_size: 0.00 Bytes)

Retainers:

- SYNTTETICROOT @1 (type: synthetic, self_size: 0.00 Bytes)



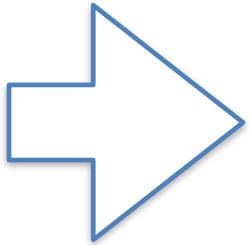
定位泄漏点（GC roots）



经典案例实战

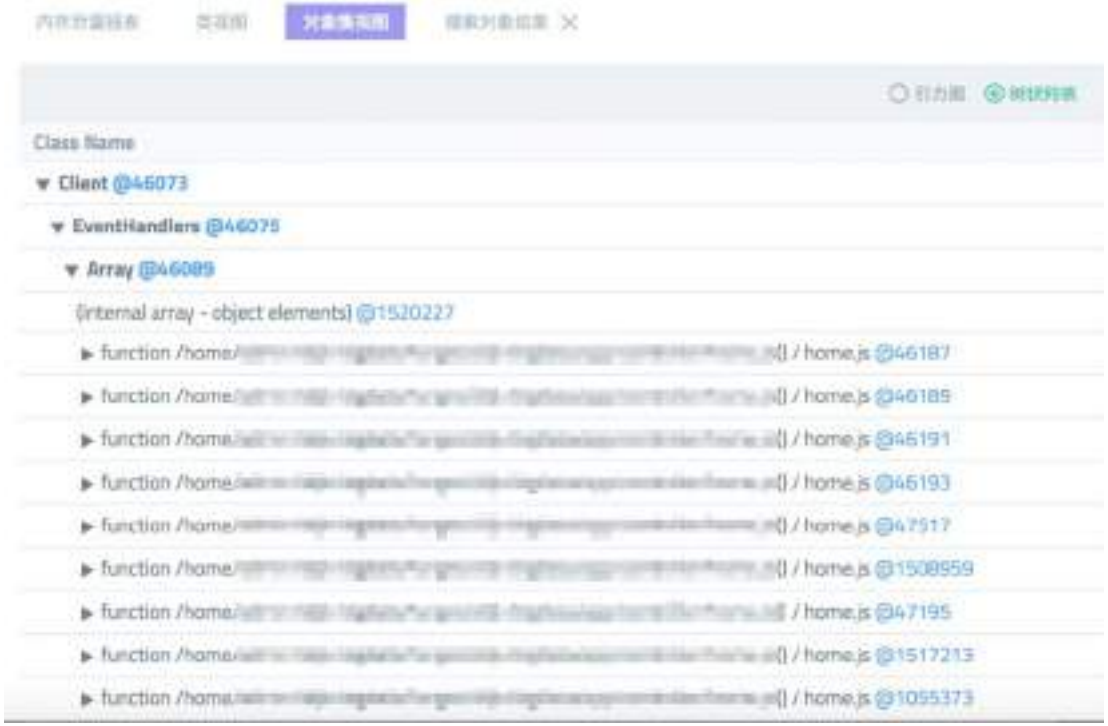
EventHandle

```
if (!client) {
  client = Client.create({
    refreshInterval: 30000,
    requestTimeout: 5000,
    urllib: urllib
  });
}

client.on('error', err => {
  // error 处理...
});
```



```
'use strict';
const Client = require('@my/Client')

module.exports = app => {
  class HomeController extends app.Controller {
    * demo() {
      if (ENV === DEVELOPMENT) {
        //开发环境下操作...
      } else {
        if (!client) {
          client = Client.create({
            refreshInterval: 30000,
            requestTimeout: 5000,
            urllib: urllib
          });
        }
        client.on('error', err => {
          //error 处理...
        })
        //其余逻辑处理...
      }
    }
  }

  return HomeController;
};
```

EventHandle



内存泄漏报表　类视图　对象族视图　搜索对象结果 ×

可疑点 1

"Client"实例"@46073"占用了204,100,080 (86.52%)字节.其内存主要积累在"Array"实例"@46089".

关键字
Array
Client

EventHandle

MyEvent extends events.EventEmitter

myEvent.on(‘some’, listener)

myEvent._events[‘some’] = [listener1, listener2, listener3, …]

EventHandle

home.js()@3454

↓ context

Client@4607

↓ _events

EventHandlers

↓ erro

Array

↓ array

listener1,

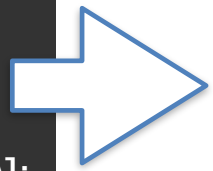
```
'use strict';
const Client = require('@my/Client')

module.exports = app => {
  class HomeController extends app.Controller {
    * demo() {
      if (ENV === DEVELOPMENT) {
        //开发环境下操作...
      } else {
        if (!client) {
          client = Client.create({
            refreshInterval: 30000,
            requestTimeout: 5000,
            urllib: urllib
          });
          每次都会给 client._events.error 数组
          增加一个错误处理侦听器
        }
        client.on('error', err => {
          //error 处理...
        })
        //其他逻辑处理...
      }
    }
  }

  return HomeController;
};
```

EventHandle —— 总结

- 设置 max listeners (setMaxListeners)
- 调用 on 方法时记得留意下是否会重复执行

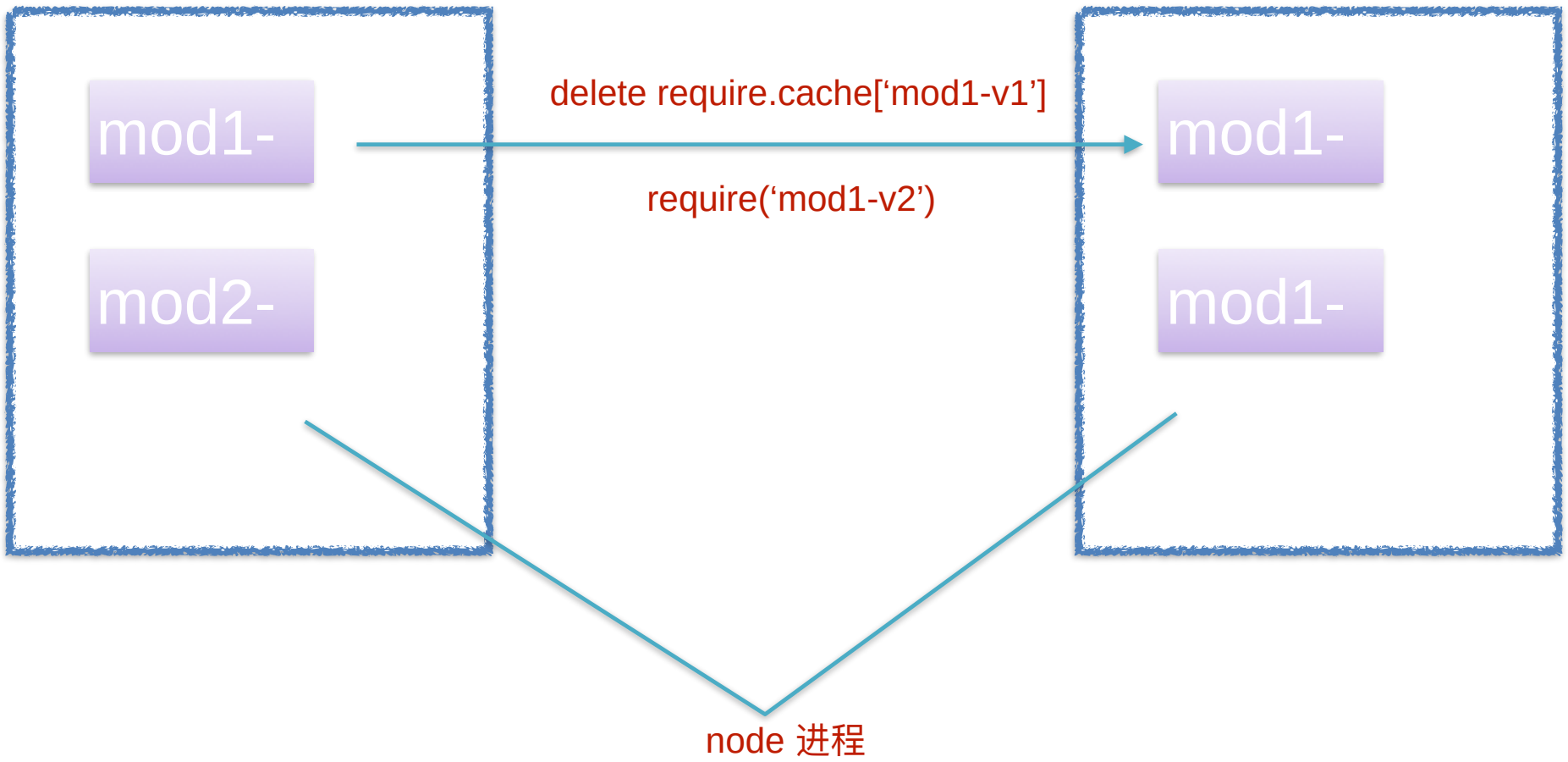
动态更新模块

```
const a = require('a');
const b = require('b');
const c = require('c');

// 业务处理...
delete require.cache[require.resolve('a')];
delete require.cache[require.resolve('b')];
delete require.cache[require.resolve('c')];
```

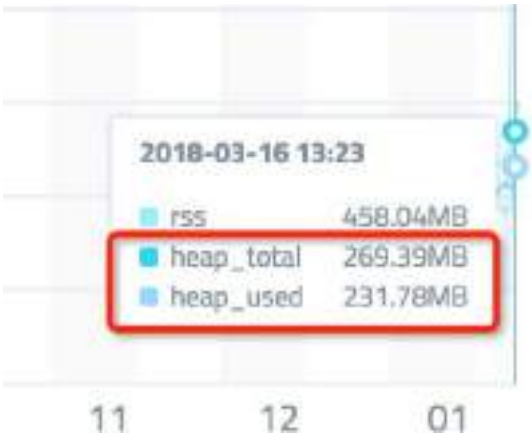


```
'use strict';
const Operator = require('./Operator.js');
const School = require('./School.js');
const User = require('./User.js');
const OperatorMenu = require('./OperatorMenu.js');

//...
Model.prototype.getSchoolList = function () {
  //...
  let where = [
    ['service', '=', service],
    ['userid', '=', userid]
  ]
  return school.find().where(where);
}

delete require.cache[require.resolve('./Operator.js')];
delete require.cache[require.resolve('./School.js')];
delete require.cache[require.resolve('./User.js')];
delete require.cache[require.resolve('./OperatorMenu')];
```

动态更新模块


mod1-
mod2-
delete require.cache[‘mod1-v1’]
require(‘mod1-v2’)
mod1-
mod1-
node 进程

动态更新模块



▾Module	3	40 900 2 %	3 272 000 2 %	177 671 232 84 %
▾Module @14727	6		80 0 %	172 341 688 81 %
▾children :: Array @2016689	7		32 0 %	172 341 096 81 %
▾[196] :: Module @1477865	8		80 0 %	1 808 720 1 %
▸parent :: Module @14727	8		80 0 %	172 341 688 81 %
▸children :: Array @4200727	9		32 0 %	406 296 0 %
▸exports :: Model /.../common/models/Operator.js() @5352717	9		72 0 %	8 160 0 %
▸paths :: Array @4200729	9		32 0 %	584 0 %
▸__proto__ :: @7263	4		56 0 %	528 0 %
▸map :: system / Map @7265	4		88 0 %	296 0 %
▸filename :: "/.../common/models/Operator.js" @135655	7		64 0 %	64 0 %
▸id :: "/.../common/models/Operator.js" @135655	7		64 0 %	64 0 %
▸loaded :: system / Oddball @53	3		48 0 %	48 0 %
▾[144] :: Module @1319581	8		80 0 %	1 806 256 1 %
▸parent :: Module @14727	6		80 0 %	172 341 688 81 %
▸children :: Array @1948867	9		32 0 %	403 616 0 %
▸exports :: Model /.../common/models/Operator.js() @1948865	9		72 0 %	8 280 0 %
▸paths :: Array @1948869	9		32 0 %	584 0 %
▸__proto__ :: @7263	4		56 0 %	528 0 %
▸map :: system / Map @7265	4		88 0 %	296 0 %
▸filename :: "/.../common/models/Operator.js" @135655	7		64 0 %	64 0 %
▸id :: "/.../common/models/Operator.js" @135655	7		64 0 %	64 0 %
▸loaded :: system / Oddball @53	3		48 0 %	48 0 %
▸[128] :: Module @1685857	8		80 0 %	1 806 080 1 %

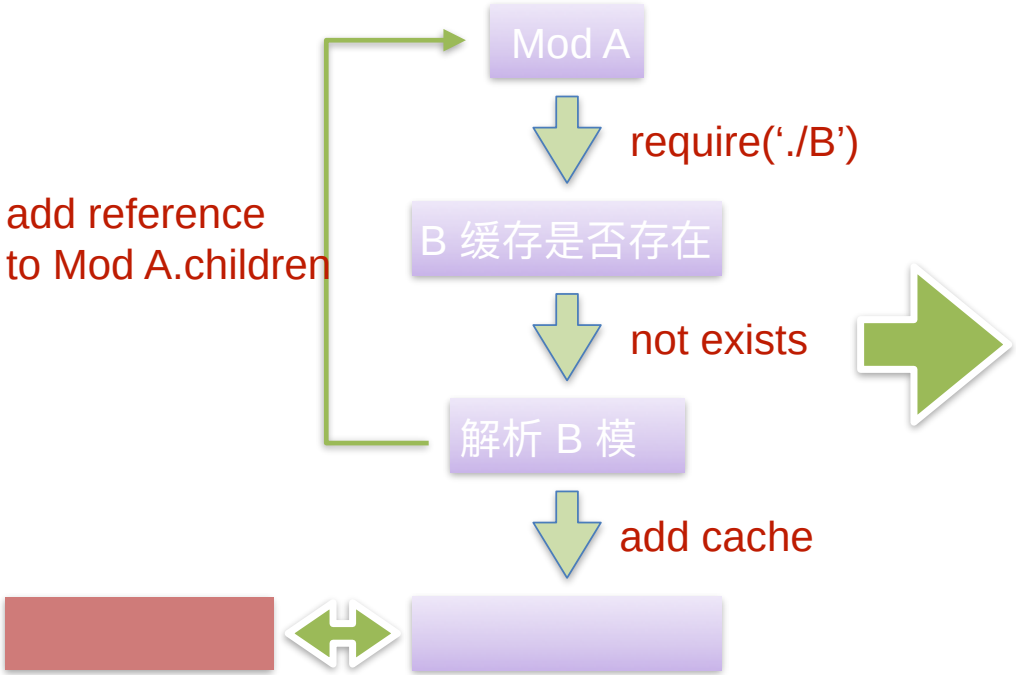
内存泄漏报表　类视图　对象簇视图

可疑点 1

"Module"实例"@14727"占用了172321048(81.46%)字节.其内存主要积累在"Array"实例"@2016689".

关键字
Module
Array

动态更新模块

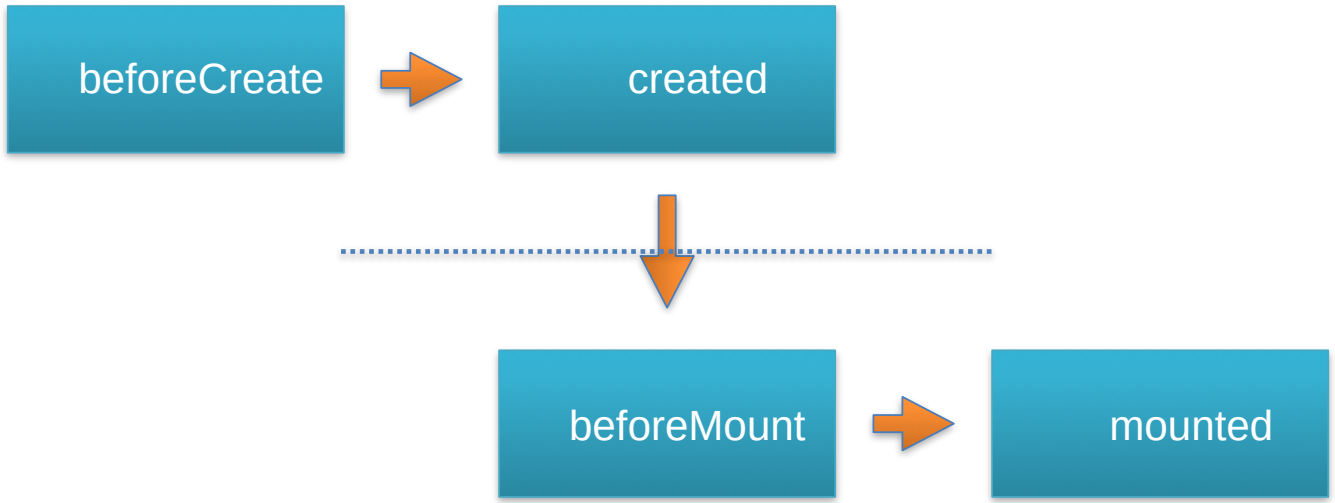



```
'use strict';
const Operator = require('./Operator.js');
const School = require('./School.js');
const User = require('./User.js');
const OperatorMenu = require('./OperatorMenu.js');

//...
Model.prototype.getSchoolList = function () {
  //...
  let where = [
    ['service', '=', service],
    ['userid', '=', userid]
  ]
  return school.find().where(where);
}

delete require.cache[require.resolve('./Operator.js')];
delete require.cache[require.resolve('./School.js')];
delete require.cache[require.resolve('./User.js')];
delete require.cache[require.resolve('./OperatorMenu')];
```

动态更新模块 —— 总结

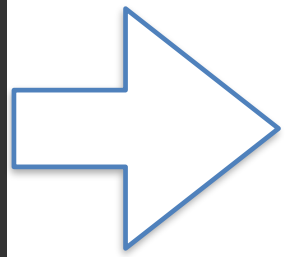
- 清除模块的缓存需要清除 require.cache 和所有父引用
- 清除的模块中包含定时器、socket 连接等资源的需要手动释放
- 不建议在线上对 Node.js 进程做模块的热加载

网红 Vue



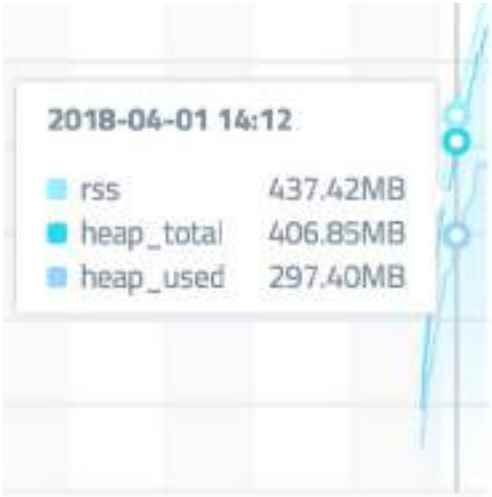
网红 Vue

```
const app = new Vue({
  created() {
    setInterval(function () {
      // 业务处理..
    });
  },
  template: `<xxx></xxx>`
});

render.renderToString(app, (err, html) => {
  // 业务处理..
});
```



```
'use strict';
const Vue = require('vue');
const server = require('express')();
const renderer = require('vue-server-renderer')
  .createRenderer();
Vue.component('public-component', {
  props: ['message'],
  template: `<div>{{ message }}</div>`
});
server.get('*', (req, res) => {
  const app = new Vue({
    created() { setInterval(function () { }, 10); },
    template: `<public-component :message="message">
    </public-component>`,
    computed: { message() { return 'Hello World'; } }
  })
  renderer.renderToString(app, (err, html) => {
    if (err) {
      console.log(err)
      res.status(500).end('Internal Server Error')
      return
    }
    res.end(html);
  })
})
server.listen(8080, () => console.log(process.pid));
```

网红 Vue



可疑点 1

"TimersList"实例"@154913"占用了54372888(78.4%)字节.

关键字
TimersList

Class Name

- ▼ TimersList @154913
  - ▼ Timeout @154907
    - ▶ function [redacted]/index.js() / index.js @154899
  - ▼ Timeout @103851
    - ▶ function [redacted]/index.js() / index.js @103855
  - ▼ Timeout @881673
    - ▶ function [redacted]/index.js() / index.js @881681
  - ▶ Timeout @154909
  - ▶ Timeout @1026315
  - ▶ Timeout @1220567
  - ▶ Timeout @1021349
  - ▶ Timeout @1021351
  - ▶ Timeout @1021353

网红 Vue

created 会在 ssr 时执行，这里创建的 timer 等资源属性的元素会在服务端累加

```
'use strict';
const Vue = require('vue');
const server = require('express')();
const renderer = require('vue-server-renderer')
  .createRenderer();
Vue.component('public-component', {
  props: ['message'],
  template: `<div>{{ message }}</div>`
});
server.get('*', (req, res) => {
  const app = new Vue({
    created() { setInterval(function () { }, 10); },
    template: `<public-component :message="message">
    </public-component>`,
    computed: { message() { return 'Hello World'; } }
  })
  renderer.renderToString(app, (err, html) => {
    if (err) {
      console.log(err)
      res.status(500).end('Internal Server Error')
      return
    }
    res.end(html);
  })
})
server.listen(8080, () => console.log(process.pid));
```

网红 Vue —— 总结

- 加强 SSR 生命周期的理解
- SSR 中处于服务端执行的生命周期钩子中不添加资源属性元素

总结与思考

线上诊断两大难题

- Out of Memory
- 进程仍存活但无故不响应