

# A NODE.JS TALK

---

@SYSU

## 张秋怡

- ▶ 12 级软件工程
- ▶ 校招进入 alinode (阿里云)
  - ▶ <https://alinode.aliyun.com>
- ▶ 目前工作
  - ▶ Node.js 管理解决方案 (alinode) 的开发
  - ▶ 内外部客户的性能优化技术支持
- ▶ [joyeec9h3@gmail.com](mailto:joyeec9h3@gmail.com)

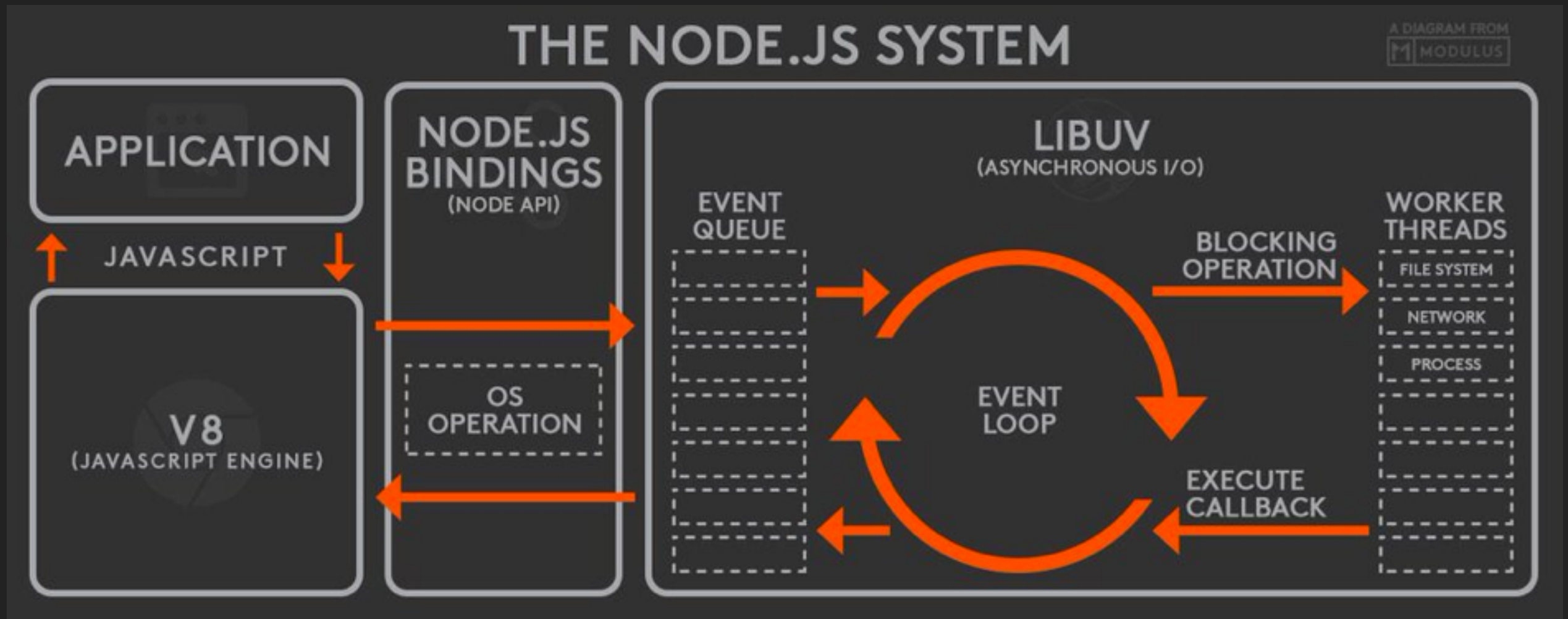
## 如果你对进入阿里工作感兴趣



- ▶ 不同 BU / 团队不同技术架构，切勿听说 A 用了 X 技术以为 B 也会用跑去投简历
- ▶ 招聘比上岗要提前很多的！！
- ▶ 校招（应届毕业生）大三暑假开始
- ▶ 实习大三上学期（3~4月）开始
- ▶ 内推
  - ▶ 推荐 — 筛简历 — 电话面试 — offer
  - ▶ 有认识的人在想去的团队可以直接联系，否则可能随机到不感兴趣的地方去
  - ▶ 一旦点开内推链接，你就不能随便换 BU 推了
- ▶ 网申
  - ▶ 网上申请 — 笔试 — 电话面试 — offer

# BEFORE WE START

## Node.js 是什么?



Credit: BusyRich @ Twitter

- ▶ 核心包括 libuv + cares (DNS) + OpenSSL(crypto) + V8(执行JavaScript) 等
- ▶ 本身主要由 C/C++ 组成，部分是 JavaScript 写的
- ▶ 用户在使用 Node.js 时，代码主要是 JavaScript 的，也可以写 C/C++ 的 addon，暴露 binding 和 JavaScript 互相调用

# BEFORE WE START

---

## Node.js 是什么?

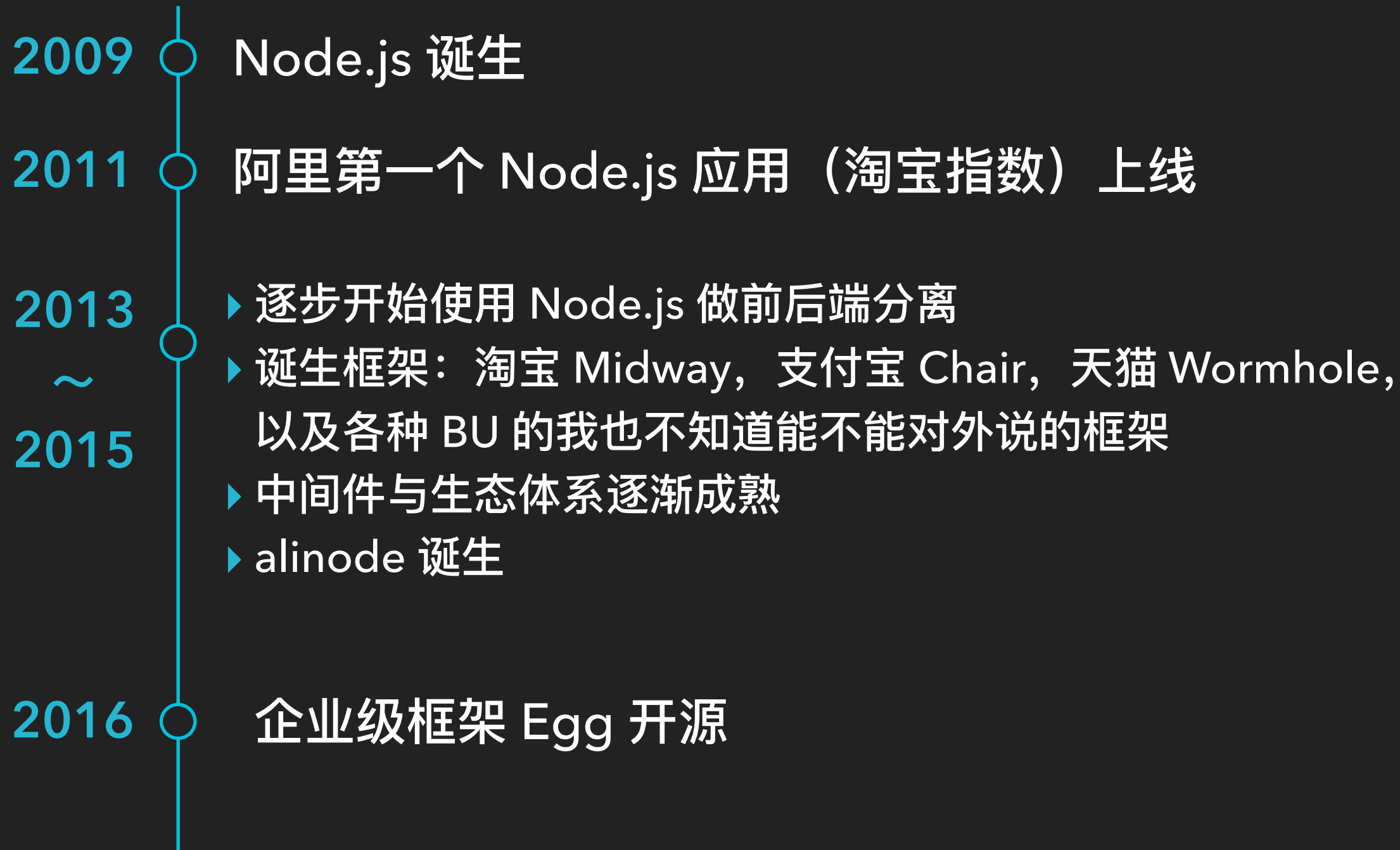
```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

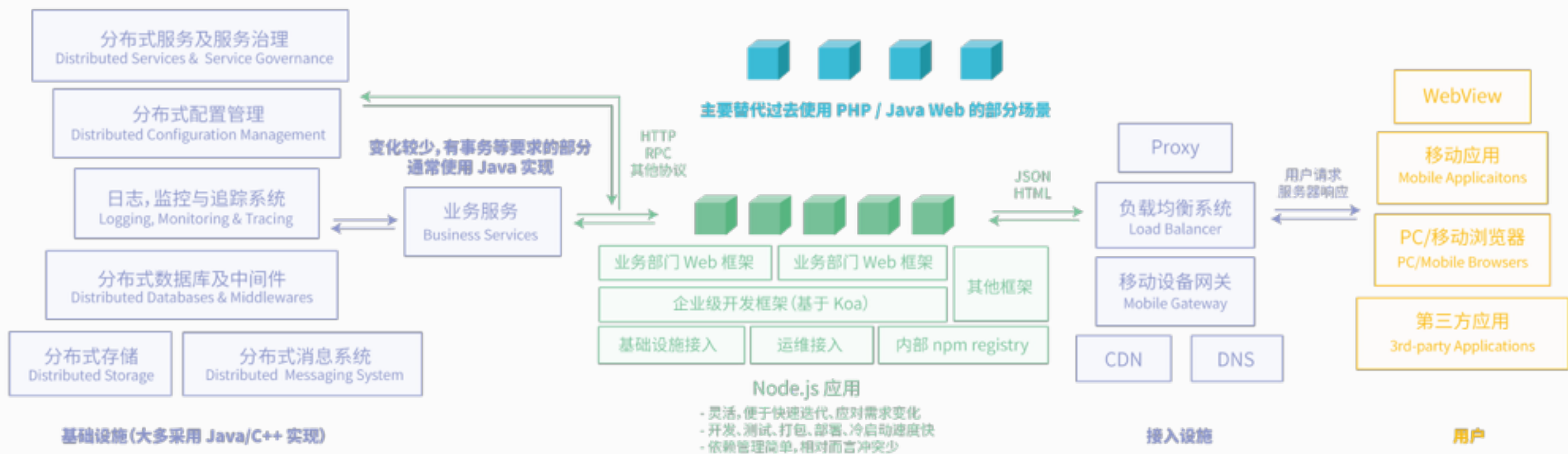
## Node.js @ Alibaba



# BACKGROUND

## Node.js @ Alibaba

### 阿里的 Node.js 应用场景



- ▶ 这个架构的前提是左边（紫色）的基础设施，复杂业务逻辑服务化的支持，和右边（紫色）的接入设施
- ▶ Netflix, Paypal, ...etc 也在进行类似的改造
- ▶ 没有银弹，不要到处说 Node.js FTW，没人可以 FTW.....

## Node.js @ Alibaba

WHY?

### ▶ 相比于 Java Web

- ▶ 本地开发方便，测试打包部署冷启动快，依赖冲突少，仪式型的代码少
- ▶ 频繁变化的后端 View 和 API 可以交给前端去写，少为工作量扯皮
- ▶ Web 应用在今天的系统里只是冰山一角，封装和约束反而碍手碍脚
- ▶ 需要灵活的语法，快速迭代，应对运营和业务的变化
- ▶ 原本是前端在推，现在也有 Java 后端加入



## Node.js @ Alibaba

### WHY?

- ▶ Taobao FED: 前后端分离的思考与实践（一共六篇）
  - ▶ （不知道为什么，原来的博客删掉了，看转载吧.....）
  - ▶ <http://blog.jobbole.com/?s=前后端分离的思考与实践>
  - ▶ <http://2014.jsconf.cn/slides/herman-taobaoweb/>
- ▶ 天猫双11前端分享系列（四）：大规模 Node.js 应用
  - ▶ <https://github.com/tmallfe/tmallfe.github.io/issues/28>
  - ▶ <https://github.com/tmallfe/tmallfe.github.io/issues/30>

## Node.js @ Alibaba

WHY?

### ▶ 相比于 PHP

- ▶ 不是 JavaScript
- ▶ 同步 + 多线程模型，高并发不好 Scale
  - ▶ 一个链接一个线程 v.s. 一个链接一小块堆上内存
- ▶ 没有 JIT（最近才加上.....）

# BACKGROUND

---

## Node.js @ Alibaba

### DISADVANTAGE

- ▶ JavaScript 是为客户端而生的语言
  - ▶ 客户端运行时间一般最多在天级别，服务端可能跑几个月
  - ▶ 客户端可以刷新，服务端不能乱重启
  - ▶ 设计只花了10天，为了兼容，语法太乱，类型系统还不如叉烧
- ▶ 解决
  - ▶ 单元测试
  - ▶ 压力测试
  - ▶ Linter 检查代码，提前抓 bug

# BACKGROUND

---

## Node.js @ Alibaba

### DISADVANTAGE

#### ▶ Callback Hell

- ▶ libuv 的 API 本来就是回调风格

#### ▶ 解决

- ▶ generator/promise/thunk, 阿里目前的 Node.js 应用大多基于 Koa
- ▶ 现在没有 coroutine/fiber, 不知道以后能不能有
- ▶ ES2016 async/await

## Node.js @ Alibaba

### DISADVANTAGE

#### ▶ 服务端调优工具和经验不足

- ▶ Java 程序员面试经典问题: JVM GC

- ▶ 常见问题

- ▶ 内存泄漏, 随着请求暴涨, 进程 out of memory 挂掉

- ▶ 频繁分配临时对象, GC 暂停严重, 占用了实际执行程序的时间, 响应效率下降

- ▶ 部分代码引起 CPU 飙高, 由于 JS 执行是单线程, 影响整体响应效率

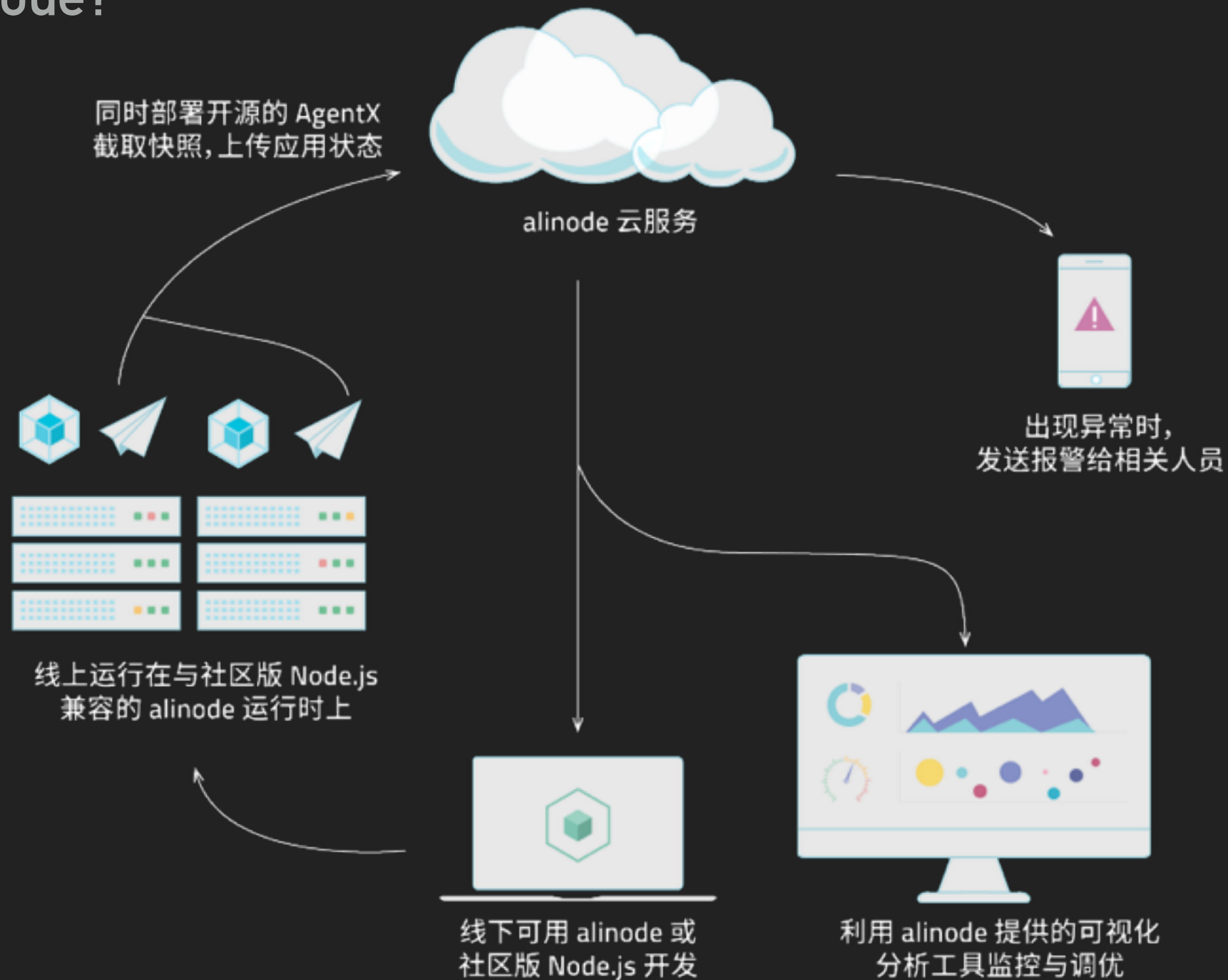
#### ▶ 解决

- ▶ alinode

# BACKGROUND

## Node.js @ Alibaba

什么是 alinode?



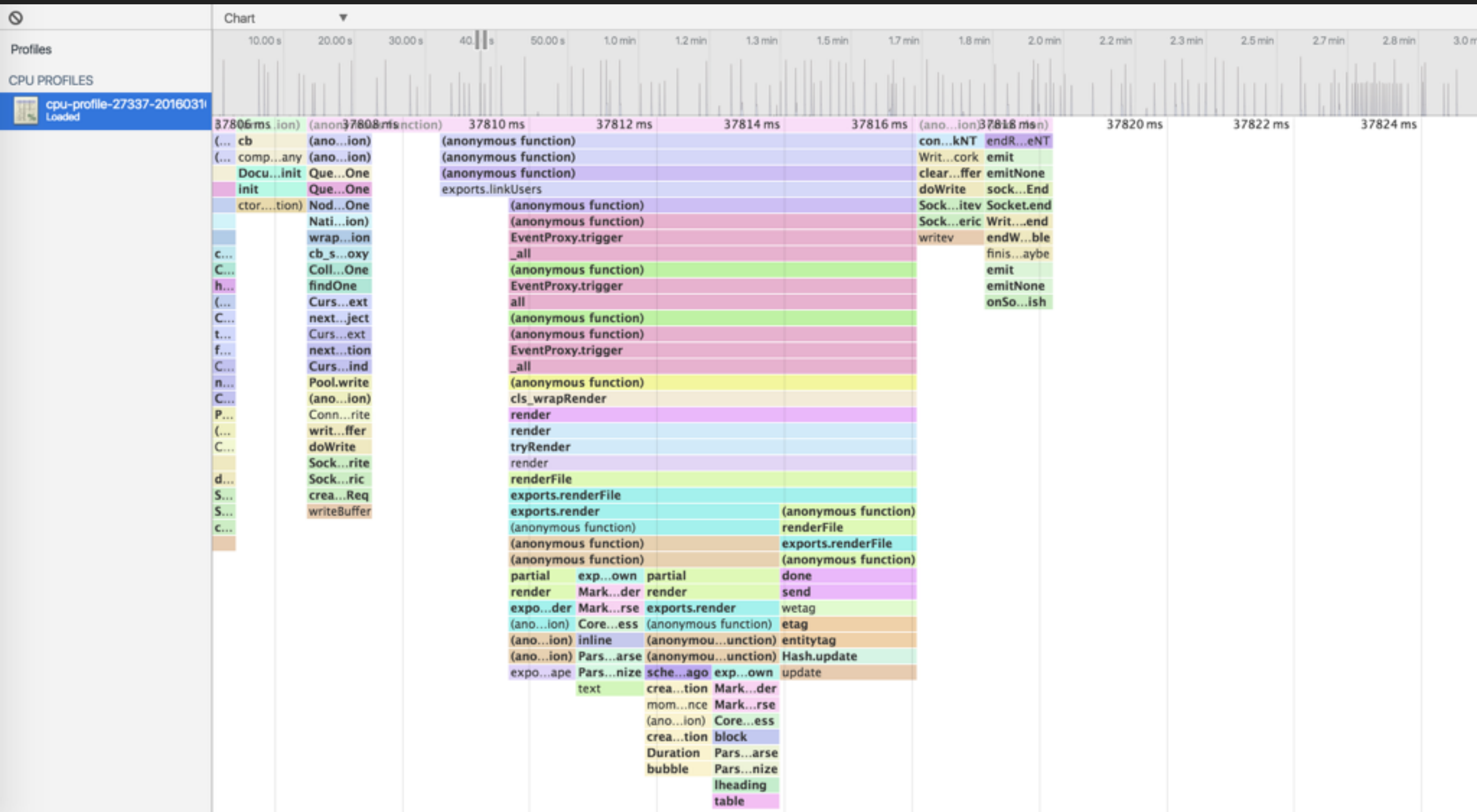
## 性能调优：JavaScript

CPU: V8 CPU PROFILE

- ▶ 没有 profile 的优化都是刷流氓
  - ▶ 如果这段代码执行时间占整体的不到 1%，算了吧
  - ▶ 如果这个程序执行时间只有几秒种，算了吧
- ▶ 先找到热点，消耗用时最多的地方，逐个击破

# 性能调优：JavaScript

## CPU: V8 CPU PROFILE





## 性能调优：JavaScript

### CPU: V8 CPU PROFILE

- ▶ 记录函数调用关系和耗时
- ▶ 怎么读懂：
  - ▶ <https://developers.google.com/web/tools/chrome-devtools/profile/rendering-tools/js-execution?hl=en>
- ▶ 怎么获取：
  - ▶ 在代码里调用 API: <https://github.com/node-inspector/v8-profiler>
    - ▶ 在 Chrome Devtools 的 profile 面板看
  - ▶ 用 Chrome Devtools 截取：
    - ▶ <https://github.com/node-inspector/node-inspector>
    - ▶ Node.js v6.x 自带支持: <https://github.com/nodejs/node/pull/6792>
  - ▶ 开 flag: ``node --prof --log_timer_events --logfile=v8.log``
    - ▶ 拖进 Chrome 的 `chrome://tracing` 看

## 性能调优：JavaScript

CPU: V8 CPU PROFILE

### ▶ 问题

- ▶ CPU profile 会影响性能，所以上面这些都不能在线上打开，只能在线下用，如果非得在线上才能重现出性能问题怎么办？
- ▶ 上面的方法有的要侵入代码，有的只能在启动的时候开启，随进程退出而关闭

### ▶ alinode 的工作

- ▶ 点点按钮，随时在线上打开，截取3分钟后自动关闭，不影响其他时候的性能
- ▶ 一键转储到阿里云上，在浏览器里打开分析，不用自己登录到服务器上拉下来
- ▶ TODO：魔改 Chrome Devtools，加一些额外的功能（展开 GC 触发统计等）

# NODE.JS OPTIMIZATION

## 性能调优：JavaScript
























CPU: V8 CPU PROFILE



# NODE.JS OPTIMIZATION

## 性能调优：JavaScript

### CPU: V8 CPU PROFILE

 堆快照	<b>/tmp/heapdump-27282-20160712-181830.heapsnapshot</b> 由 foo@bar.com 于 2016-07-12 18:18:31 创建在实例 i104jlo10JF	已生成 	已转储 再转储 	分析  (allnode)	分析  (devtools)	下载 
 堆快照	<b>/tmp/heapdump-27282-20160712-181740.heapsnapshot</b> 由 foo@bar.com 于 2016-07-12 18:17:41 创建在实例 i104jlo10JF	已生成 	转储 	分析  (allnode)	分析  (devtools)	下载 
 GC Trace	<b>/tmp/gc-log-27282-20160712-181656.txt</b> 由 foo@bar.com 于 2016-07-12 18:16:57 创建在实例 i104jlo10JF	生成中 	转储 	分析  (allnode)		下载 
 CPU Profile	<b>/tmp/cpu-profile-27282-20160712-181650.cpubprofile</b> 由 foo@bar.com 于 2016-07-12 18:16:50 创建在实例 i104jlo10JF	生成中 	转储 	分析  (devtools)		下载 
 堆快照	<b>/tmp/heapdump-27282-20160712-181646.heapsnapshot</b> 由 foo@bar.com 于 2016-07-12 18:16:47 创建在实例 i104jlo10JF	已生成 	转储 	分析  (allnode)	分析  (devtools)	下载 

## 性能调优：JavaScript

CPU: V8 CPU PROFILE

### ▶ TIPS:

- ▶ 热点地带不要写匿名函数，不然 profile 里只能看到 `(anonymous function)`

```
func(function () {  
  return 1;  
});
```

```
// OR  
func(() => 1);
```

```
func(function callback() {  
  return 1;  
});
```

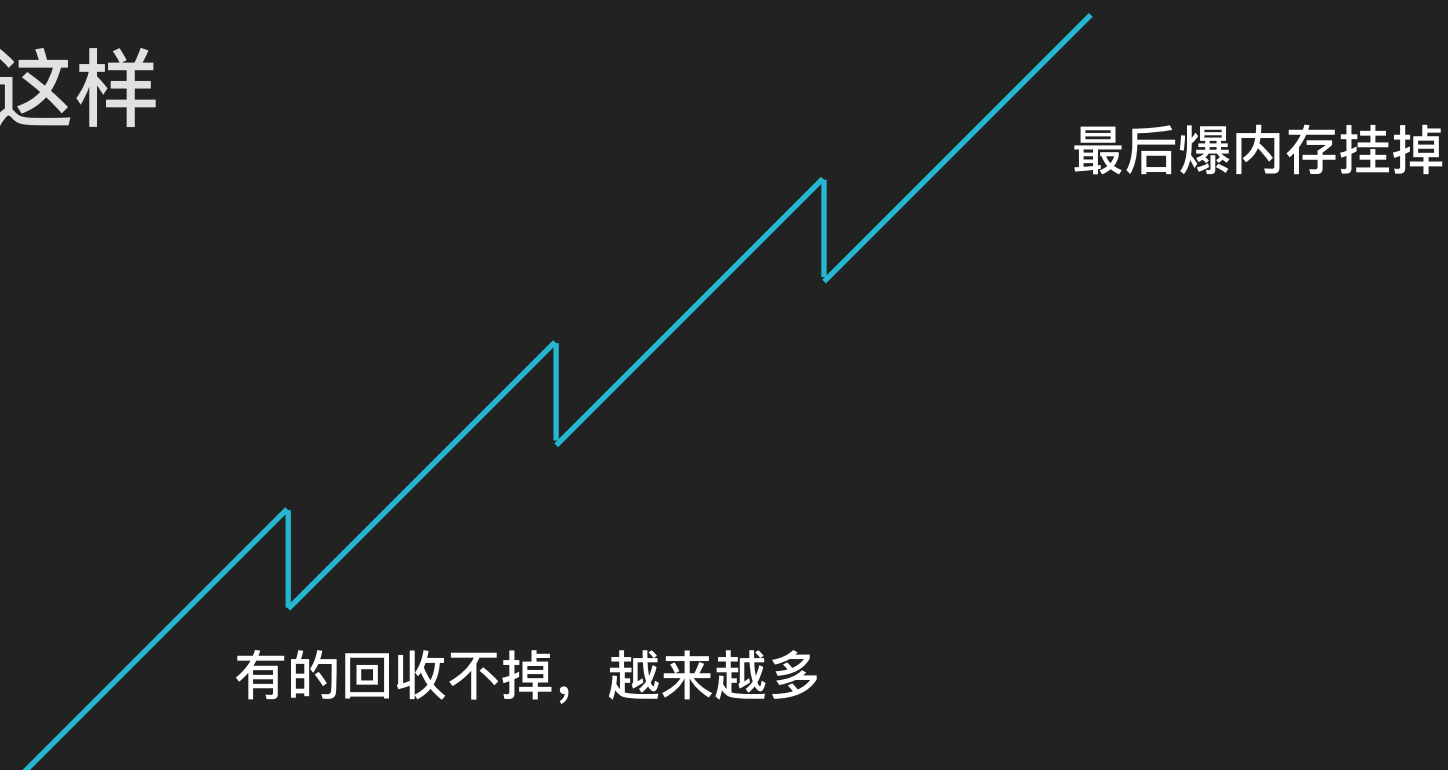
## 性能调优：JavaScript

内存：V8 HEAP SNAPSHOT

- ▶ 普通的内存消耗长这样



- ▶ 内存泄漏长这样



# 内存: V8 HEAP SNAPSHOT

	Summary	Class filter	All objects							
Profiles	Constructor					Distance	Objects Count	Shallow Size	Retained Size	
	▼ smalloc					5	17 0%	17 939 304 28%	17 939 304 28%	
HEAP SNAPSHOTS	▶ smalloc @1979291716					11		17 825 792 28%	17 825 792 28%	
	▼ smalloc @48792100					11		25 210 0%	25 210 0%	
heapdump-15120-20150911 60.4 MB	▶ [1] :: NativeBuffer @759					10		40 0%	40 0%	
	▶ smalloc @42558948					12		8 192 0%	8 192 0%	
	▶ smalloc @42762436					12		8 192 0%	8 192 0%	
	▶ smalloc @48691268					14		8 192 0%	8 192 0%	
	▶ smalloc @48728804					14		8 192 0%	8 192 0%	
	▶ smalloc @48756164					12		8 192 0%	8 192 0%	
	▶ smalloc @49171620					12		8 192 0%	8 192 0%	
	▶ smalloc @50115620					5		8 192 0%	8 192 0%	
	▶ smalloc @50279620					14		8 192 0%	8 192 0%	
	▶ smalloc @51785764					8		8 192 0%	8 192 0%	
	▶ smalloc @53467556					12		8 192 0%	8 192 0%	
	▶ smalloc @51450788					12		1 643 0%	1 643 0%	
	▶ smalloc @42245636					12		1 620 0%	1 620 0%	
	▶ smalloc @42778948					12		1 616 0%	1 616 0%	
	▶ smalloc @42256708					12		1 498 0%	1 498 0%	
	▶ smalloc @43362884					15		5 0%	5 0%	
	▶ (string)					3	80 932 25%	15 083 480 24%	15 083 480 24%	
	▶ (array)					-	75 828 23%	13 258 896 21%	13 258 896 21%	
	▶ (compiled code)					-	34 088 11%	8 863 776 14%	8 863 776 14%	
	▶ (system)					-	49 240 15%	2 108 632 3%	2 108 632 3%	
	▶ (closure)					-	18 665 6%	1 343 880 2%	1 343 880 2%	
	▶ Object					1	18 884 6%	1 070 320 2%	1 070 320 2%	
	▶ Array					3	19 316 6%	618 112 1%	618 112 1%	
	▶ (concatenated string)					4	10 498 3%	419 920 1%	419 920 1%	
	▶ system / Context					3	4 708 1%	333 736 1%	333 736 1%	
	▶ Module					3	1 247 0%	99 704 0%	99 704 0%	
	▶ (regexp)					2	1 176 0%	75 264 0%	75 264 0%	
	▶ CallSite					8	636 0%	35 616 0%	35 616 0%	
	▶ Layer					7	195 0%	15 600 0%	15 600 0%	
	▶ system / JSArrayBufferData					5	19 0%	14 912 0%	14 912 0%	
	Retainers									
	Object					Distance	▲	Shallow Size	Retained Size	
	▼ native in NativeBuffer @873					11		40 0%	40 0%	
	▼ collector-124.newrelic.com:443::: in @104741					10		56 0%	56 0%	
	▼ map in @103593					9		40 0%	40 0%	
	▼ _sessionCache in Agent @103165					8		128 0%	128 0%	
	▼ globalAgent in @99019					7		56 0%	56 0%	
	▼ exports in NativeModule @47371					6		56 0%	56 0%	
	▼ https in @15953					5		24 0%	24 0%	
	▼ _cache in function NativeModule() @15949					4		72 0%	72 0%	
	▼ NativeModule in system / Context @14267					3		128 0%	128 0%	

## 性能调优：JavaScript

### 内存：V8 HEAP SNAPSHOT

- ▶ 遍历整个堆，记录所有对象（大小、隐藏类等）和引用关系
- ▶ 怎么读懂：
  - ▶ <https://developers.google.com/web/tools/chrome-devtools/profile/memory-problems/heap-snapshots>
- ▶ 怎么获取：
  - ▶ 在代码里调用 API: <https://github.com/node-inspector/v8-profiler>
    - ▶ 在 Chrome Devtools 的 profile 面板看
  - ▶ 用 Chrome Devtools 截取：
    - ▶ <https://github.com/node-inspector/node-inspector>
    - ▶ Node.js v6.x 自带支持: <https://github.com/nodejs/node/pull/6792>
  - ▶ 发信号 SIGUSR2 给 node 进程: `kill -USR2 <PID>`
    - ▶ 生成到 current working directory



## 性能调优：JavaScript

内存：V8 HEAP SNAPSHOT

### ▶ 问题

- ▶ 同样，有的要侵入代码，有的不能在线上用，出于安全考虑不能乱接受远程信号，需要登录到服务器上
- ▶ 有问题的代码可能会产生非常大（上GB）的堆，不方便传输和加载到 Chrome Devtools 查看

### ▶ alinode 的工作

- ▶ 点点按钮，随时在线上触发
- ▶ 一键转储到阿里云上，在浏览器里打开分析，不用自己登录到服务器上拉下来
- ▶ 除了 Chrome Devtools 外还提供另一种分析服务，可以异步加载并可视化，不用将整个文件加载到浏览器里，还能自动查找内存泄漏可疑点

# NODE.JS OPTIMIZATION

## 性能调优：JavaScript

内存：V8 HEAP SNAPSHOT

cnode(专业版) > 堆快照: heapdump-26963-20150910-192030.heapsnapshot

按类名 输入关键字 搜索

数据采集时间: 2016-09-04 23:26:06

28.91MB  
文件大小

61.76MB  
Shallow Size 总大小

16386  
隐藏类个数

331870  
对象个数

874  
GC Roots 个数

内存泄漏报表

类视图

对象簇视图

可疑点 1

"(context)"实例"@126367"占用了17,827,128 (27.53%)字节. 其内存主要积累在"Buffer"3

关键字  
Buffer  
(context)

@305

Shallow Size  
32,808 B

Retained Size  
565,736 B

Name  
(internal array) @305

查询对象 查看 GC Root

依赖路径 引力图 树状列表

可疑点 1

可疑点 2

可疑点 3

可疑点 4

Remainder

Retained size 比例

显示箭头

## 性能调优：JavaScript

内存：V8 HEAP SNAPSHOT

### ▶ TIPS:

- ▶ 会大量出现的对象，尽量用“构造函数”模式创建，不然 Snapshot 里可能只有一个 ID，根本找不到是什么代码创建的

```
var a = {  
  foo: 'bar'  
};
```

```
function A() {  
  this.foo = 'bar';  
}
```

```
var a = new A();
```

## 性能调优：JavaScript

内存：V8 HEAP SNAPSHOT

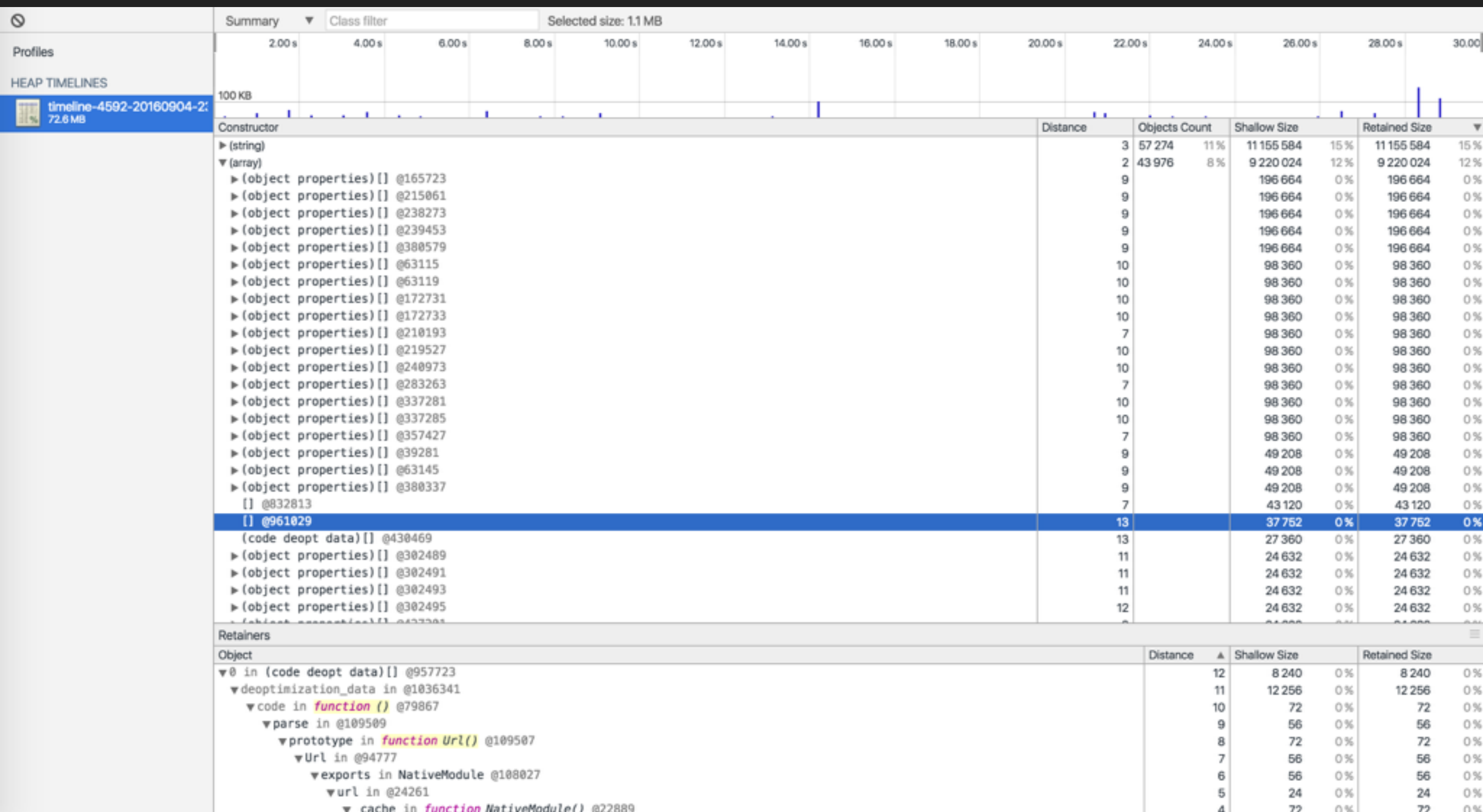
### ▶ TIPS:

- ▶ 不要在热点代码乱用闭包
- ▶ 你知道 V8 里只要作用域代码出现了闭包声明，不管最后有没有用，整个上下文都会包起来，保留引用没法被 GC 么？
  - ▶ 莫名其妙的内存泄漏
  - ▶ 闭包上下文（context）必须从内存加载，但参数经过逃逸分析可以从栈上加载，分配到寄存器，速度++

# NODE.JS OPTIMIZATION

## 性能调优：JavaScript

### 内存：V8 HEAP TIMELINE



## 性能调优：JavaScript

### 内存：V8 HEAP TIMELINE

- ▶ 记录一段时间内的对象存活状况，相当于长时间的 Heap Snapshot
- ▶ 怎么读懂：
  - ▶ <https://developers.google.com/web/tools/chrome-devtools/profile/memory-problems/allocation-profiler?hl=en>
- ▶ 怎么获取：
  - ▶ 用 Chrome Devtools 截取：
    - ▶ <https://github.com/node-inspector/node-inspector>
    - ▶ Node.js v6.x 自带支持： <https://github.com/nodejs/node/pull/6792>
  - ▶ 开 flag：`node --track\_heap\_objects`
    - ▶ 在 Chrome Devtools 的 profile 面板看

## 性能调优：JavaScript

内存：V8 HEAP SNAPSHOT

### ▶ 问题

- ▶ 影响性能，不能在线上一直开着，需要登录到服务器上获取

### ▶ alinode 的工作

- ▶ 点点按钮，随时在线上触发，30秒后自动关闭
- ▶ 一键转储到阿里云上，在浏览器里打开分析，不用自己登录到服务器上拉下来

## 性能调优：JavaScript

### GC: GC LOG

- ▶ GC: 为什么你在写 JavaScript 的时候不像写 C/C++ 需要自己管理内存 (new/free/malloc/delete)
- ▶ GC 回收内存有时候需要暂停 (给飞行中的飞机换轮子), 停的太多 (10%+), 时间都拿去回收内存了, 影响正事
- ▶ Heap Snapshot 和 Heap Timeline 太微观, 一个大型代码库截取出来上 G, 看得眼花。GC Log 可以先给一个宏观的诊断, 而且开销小。
- ▶ V8 的 GC (垃圾回收) 日志, 记录每次 GC 的状况
- ▶ 怎么读懂:
  - ▶ <https://github.com/joyeecheung/v8-gc-talk>
- ▶ 怎么获取:
  - ▶ ``v8.setFlagsFromString('--trace_gc_verbose')``
  - ▶ ``v8.setFlagsFromString('--trace_gc_nvp')``



## 性能调优：JavaScript

GC: GC LOG

### ▶ 问题

- ▶ 这个 flag 显然不能开到线上
- ▶ 记录下来的日志需要到服务器上拿
- ▶ 没有文档，没有可视化工具，看不懂

### ▶ alinode 的工作

- ▶ 点点按钮，随时在线上触发，3分钟后自动关闭，不影响应用
- ▶ 一键转储到阿里云上，在浏览器里打开分析，不用自己登录到服务器上拉下来
- ▶ 提供可视化工具，帮助理解 GC 日志，协助定位 GC 问题出现在哪一种对象上

# NODE.JS OPTIMIZATION

## 性能调优：JavaScript

GC: GC LOG

GC 暂停总时间  
2795ms

GC 次数 (Scavenge / Mark-sweep)  
512 (494 / 18)

Compaction 次数  
18

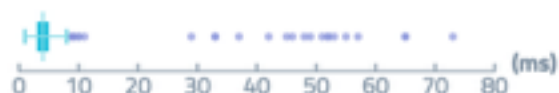
第一次 GC 前堆大小  
189.20MB

最后一次 GC 后堆大小  
219.14MB

GC 时间占比

1.56%

GC 暂停时间分布



GC 内存变化分布



GC Trace 分析



第 1 次 GC

类型: Mark-sweep

距离进程启动时间

12:16:36:41

累计 GC 时间

10307716.8 ms

堆大小变化

-143.60MB

本次 GC 暂停时间

52 ms

GC 前 GC 后



## 性能调优： JavaScript

GC: GC LOG

### ► TIPS:

- 在热点代码里，尽量复用已经分配的对象

```
var aBigArray = new Array(1e5).fill(1);
// some code
function dumb() {
  // the old Array will be GC'ed
  aBigArray = new Array(1e5).fill(1);

  use(aBigArray);
}
```

```
var aBigArray = new Array(1e5).fill(1);
function smart() {
  // reuse the memory
  aBigArray.fill(1);

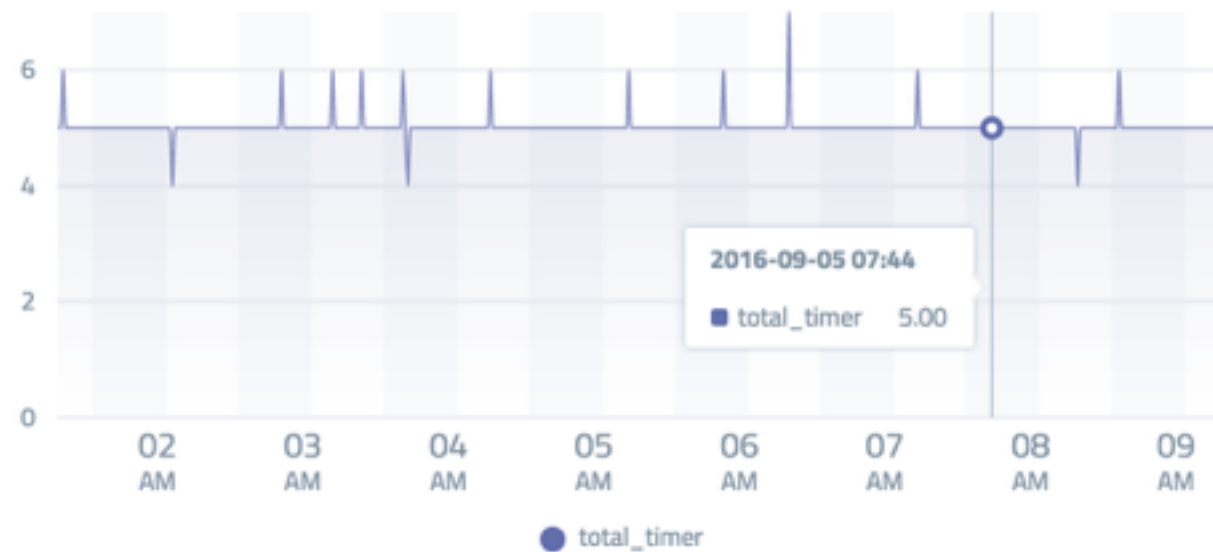
  use(aBigArray);
}
```

# NODE.JS OPTIMIZATION

## 性能调优: libuv

- ▶ 一般有问题会直接抛错
- ▶ alinode 的工作:

Timer 趋势



libuv 句柄趋势



- ▶ 更多调优工具开发中

## 其他的工作

- ▶ 异常日志的收集，不需要登录服务器看
  - ▶ 以后会加入聚合、可视化、针对 Node.js 代码的智能 bug 定位
- ▶ 慢 HTTP 日志的收集（同上）
- ▶ 分析依赖和 Node.js 版本，提示安全风险和更新



- ▶ 可以探测堆泄漏和 CPU 异常等问题
  - ▶ （没空截其他状态的图了.....）
  - ▶ 机器学习预测
    - ▶ 用的技术攀岩的课都有教

## 相关课程

- ▶ 操作系统：理解 libuv 和 V8 底层，文件系统，编译链接，进程线程, .etc
  - ▶ 课程没有讲 coroutine?
  - ▶ 建议做 PINTOS，有的老师会当作业
- ▶ 编译原理：理解 V8
  - ▶ 课程没有讲 JIT 编译和 GC？（可能后端一带而过）
  - ▶ 推荐选法师，推荐看斯坦福 Alex Aiken 的视频
    - ▶ 你知道 FJL 给 12 计应出的期末试卷就是斯坦福的试卷改改数据么？
- ▶ Web 2.0/软件过程改进？
  - ▶ 好像有教 Node.js（错了别打我）
  - ▶ 到生产环境还要涉及缓存、日志、监控、CDN等东西
- ▶ 服务计算
  - ▶ 如果是雪姨.....当我没说
  - ▶ 可能跟 SOA 相关，我没上过别打我

## 相关课程

- ▶ 函数式编程：JavaScript（大雾）
  - ▶ 我只是开个玩笑，教的其实是 Haskell
  - ▶ 如果只写过 C/C++/Java，可以帮助你了解基本的 FP 知识
  - ▶ Promise 其实是个 monad（大雾）
  - ▶ 闭包、高阶函数（这年头是个语言就有了其实）
  - ▶ “你们对力（lei）量（xing）一无所知 ♂”
- ▶ 计算机体系结构 / 计算机组成原理：汇编，理解 V8 底层优化
  - ▶ 如果老师还在教奇怪的古老内容，建议阅读 CSAPP / CAAQA
    - ▶ CAAQA 好像是体系结构的课本？
- ▶ 人工智能 / 数据挖掘：基于统计的算命（大雾）
  - ▶ 前面说了怎么用的，其实只是边角
  - ▶ 其实计算机视觉与模式识别也有很多重叠只不过用到图像（大矩阵，特征）上面去了

## 相关课程

- ▶ 计算机网络: libuv, Node.js 底层
  - ▶ DNS
  - ▶ 建议课后自己写一个可以 CGI 的 HTTP Server 玩
    - ▶ <http://tinyhttpd.sourceforge.net/> 代码风格诡异, 凑合看看
    - ▶ HTTP 大部分就是个解析字符串的活, 还特别暴力
    - ▶ 然后你就知道 Node.js 依赖的 http\_parser 是干嘛的了
  - ▶ 进阶: 用 epoll/IOCP 之类配合 Socket (UDP/TCP) 和文件系统改进你的 server, 然后你就知道 libuv 是干嘛的了
- ▶ Web 安全: crypto, OpenSSL
  - ▶ 把老蔡布置的所有作业都独立做掉, 你就入门了
  - ▶ 上完你就知道 Node.js 的 crypto API 是干嘛用的, 依赖的 OpenSSL 是干啥的了
  - ▶ 其实利用 buffer overflow 攻击写的 shell code 和 JIT 编译异曲同工



## 相关课程

- ▶ 如果新数计院的课程大改了.....都当我没说
- ▶ 好像有落下的.....? ?

Q&A