QCon+ 案例研习社

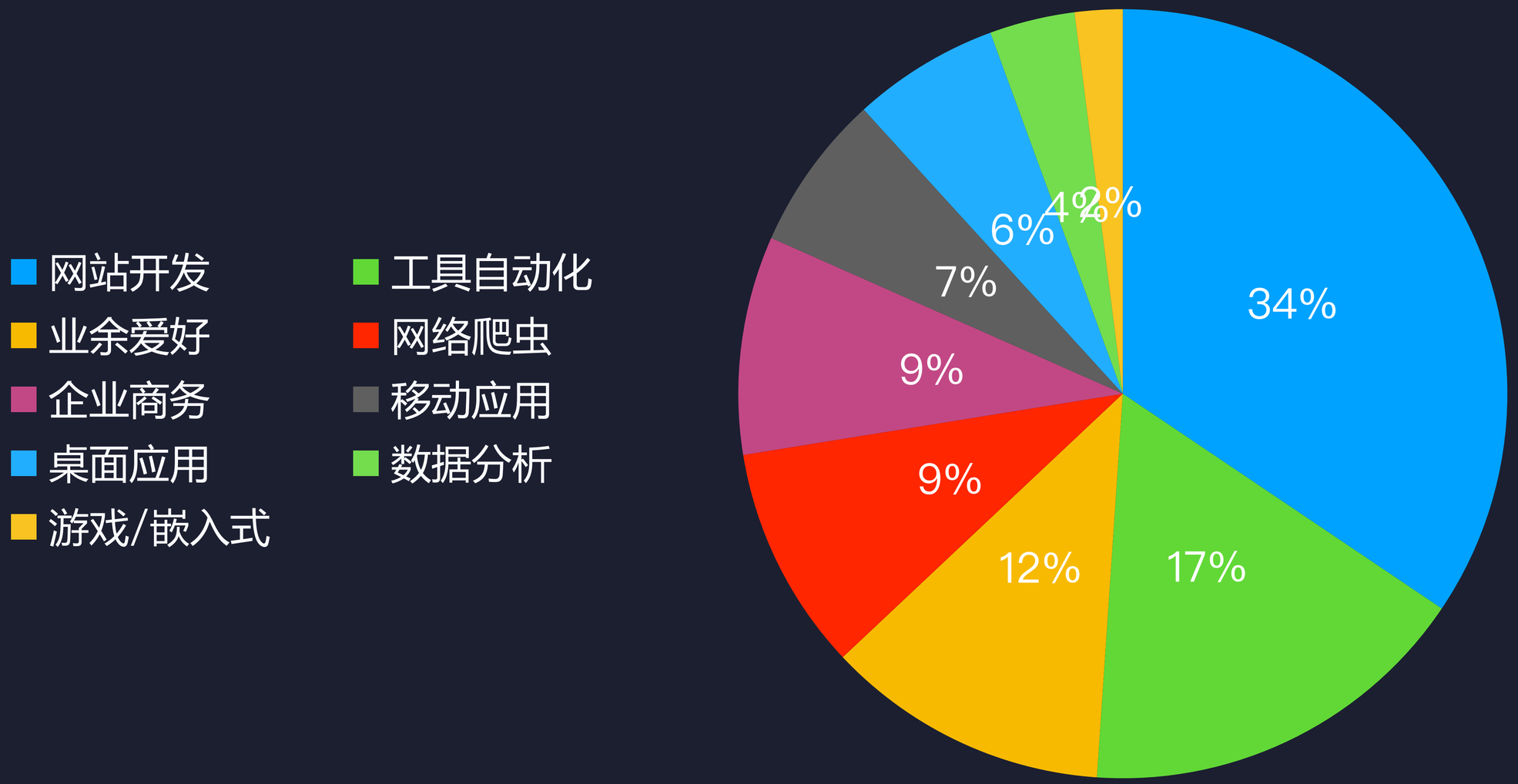
基于 Addon 的企业级 Node.js 性能监控解决方案

黄一君

黄一君

- 开源项目 Easy-Monitor 作者
- 前 AliNode 核心开发者
- 苏宁科技消费者平台研发中心

Node.js 的发展现状



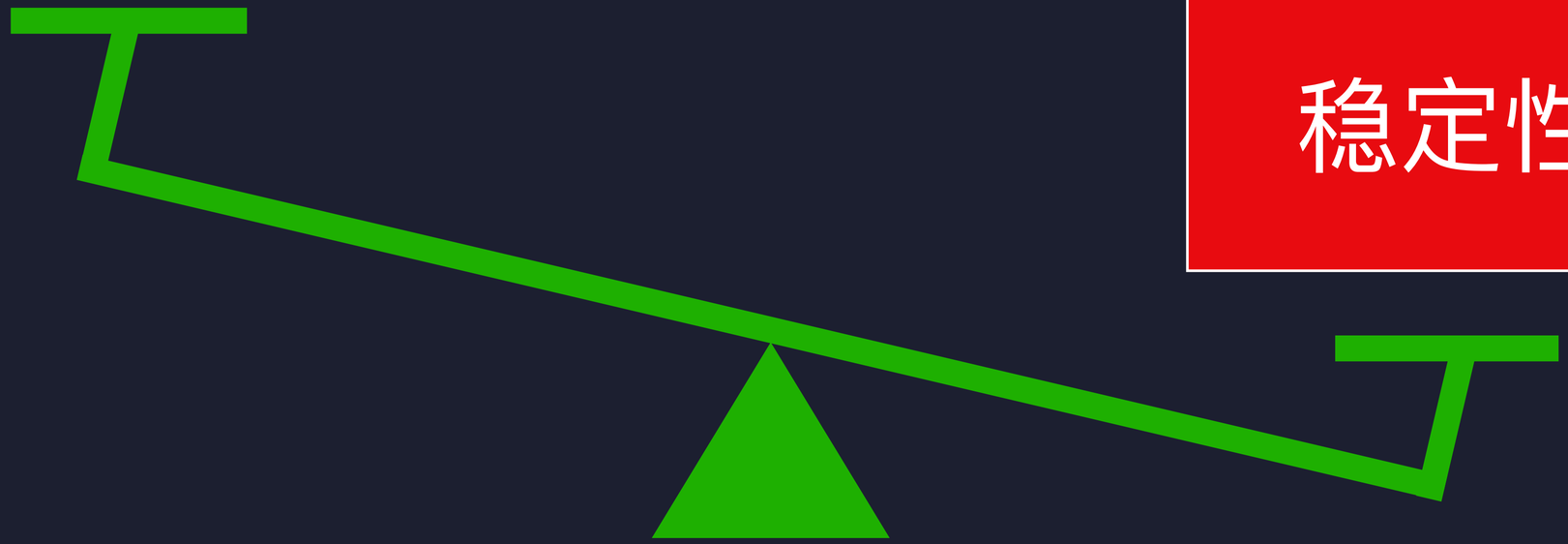
开发者面临的矛盾

快速迭代

Npm 生态

统一技术栈

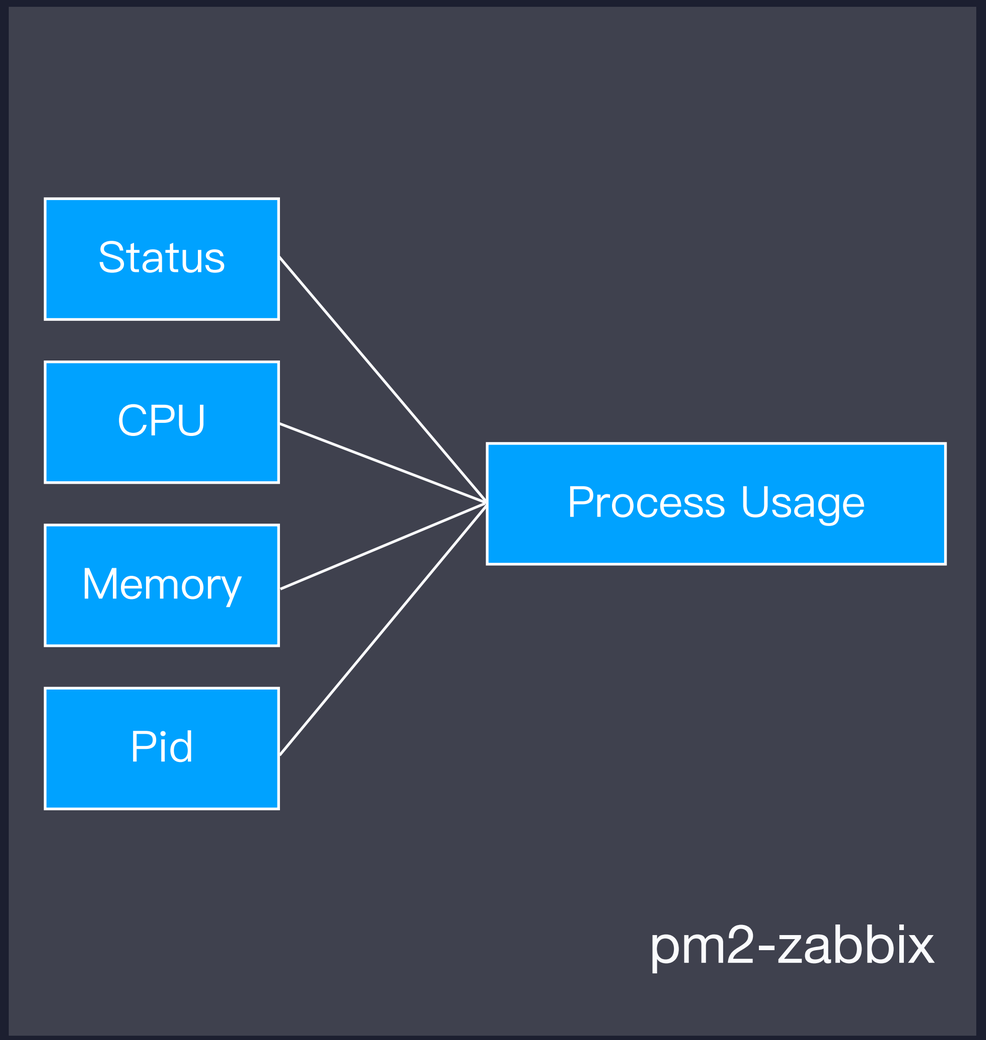
前端天花板

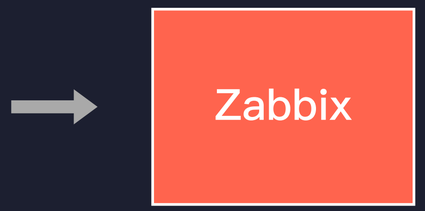


稳定性

目录

常规方案 —— Zabbix

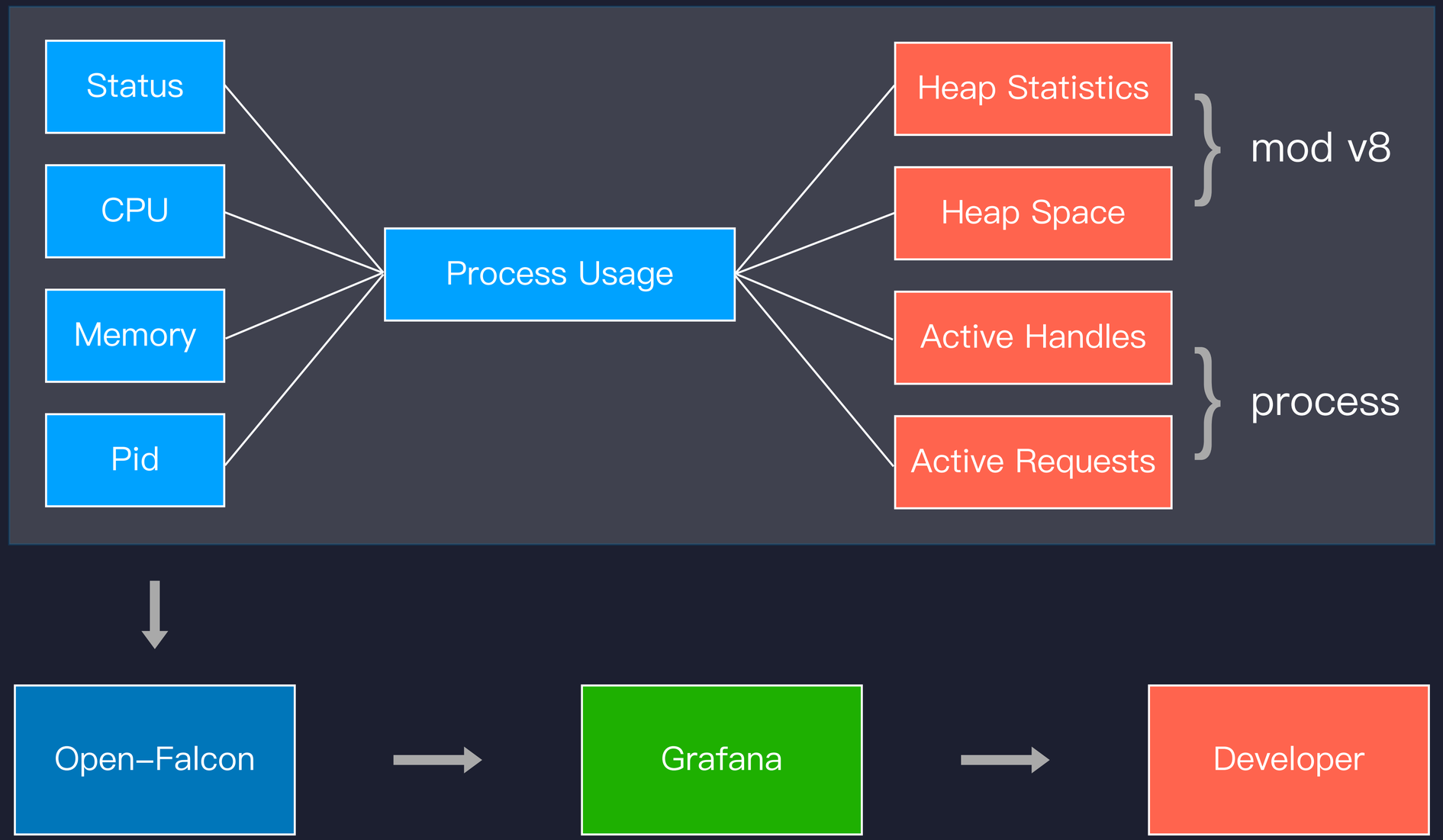

Status
CPU
Memory
Pid
Process Usage
pm2-zabbix


Zabbix


ZABBIX
监测中
资产记录
报表
配置
管理
仪表板
概述
Web监测
最新数据
触发器
事件
图形
聚合图形
拓扑图
图形
缩放: 5m 15m 30m 1h 2h 3h 6h 12h 1d 3d 7d 14d 1m 所有
«« 1m 7d 1d 12h 1h 5m | 5m 1h 12h 1d 7d 1m »»
线
1.2
1.0
0.8
0.6
0.4
0.2
0
10-12 08:43
08:46
08:48
08:50
08:52
08:54
08:56
08:58
09:00
09:02
09:04
PM2 Daemon CPU Usage [平均]
最新 0
最小 0
平均 0.0083
最大 1

这些 **基础的监控指标** 足够了吗?

扩展自定义指标


Status
CPU
Memory
Pid
Process Usage
Heap Statistics
Heap Space
Active Handles
Active Requests
mod v8
process
Open-Falcon
Grafana
Developer

Node.js 监控侧更多的痛点

Status

CPU

Memory

Pid

Heap Statistics

Heap Space

Active Handles

Active Requests

更多内核数据

1. 应用的 **GC 状态** 究竟是否健康？
2. Libuv 的每一个 **活跃的句柄详情** 是什么样的？
3. 在任意状况下都能对进程进行 **采样** / **导出快照**

故障定位一体化

理想的性能监控方案

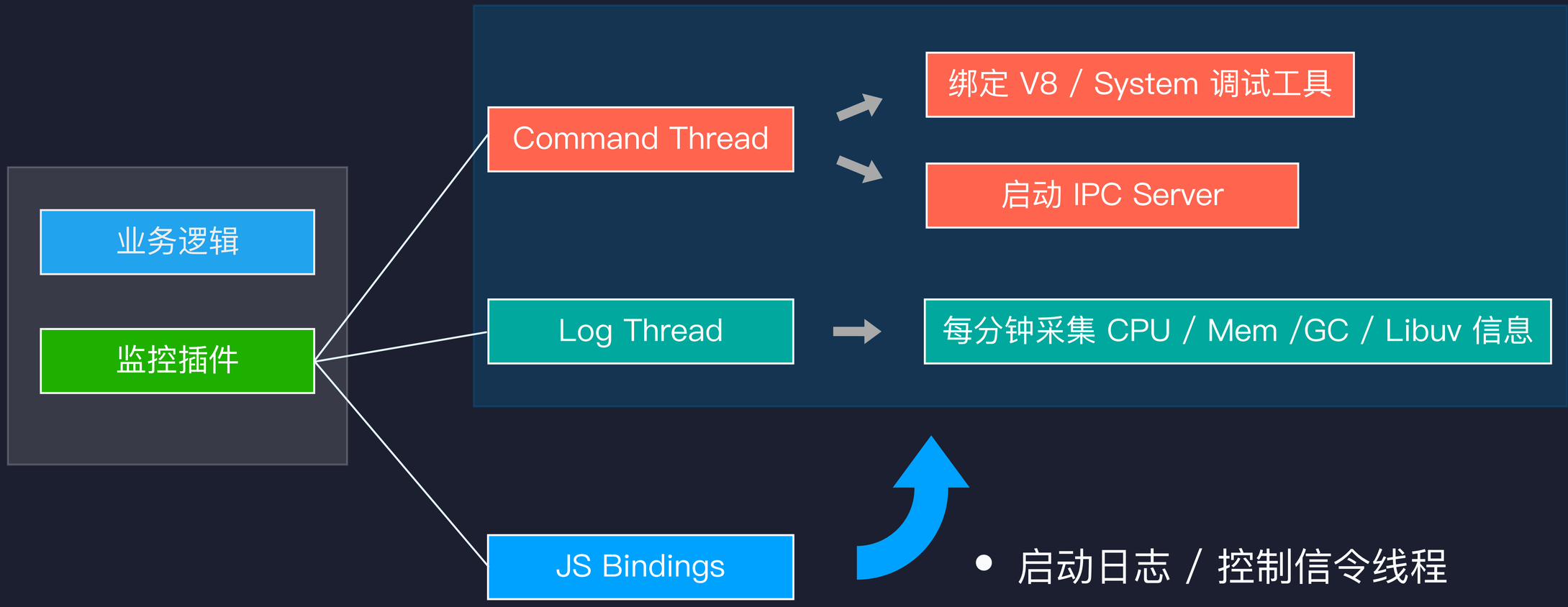
基本采集展示

异常感知机制

故障排查能力

目录

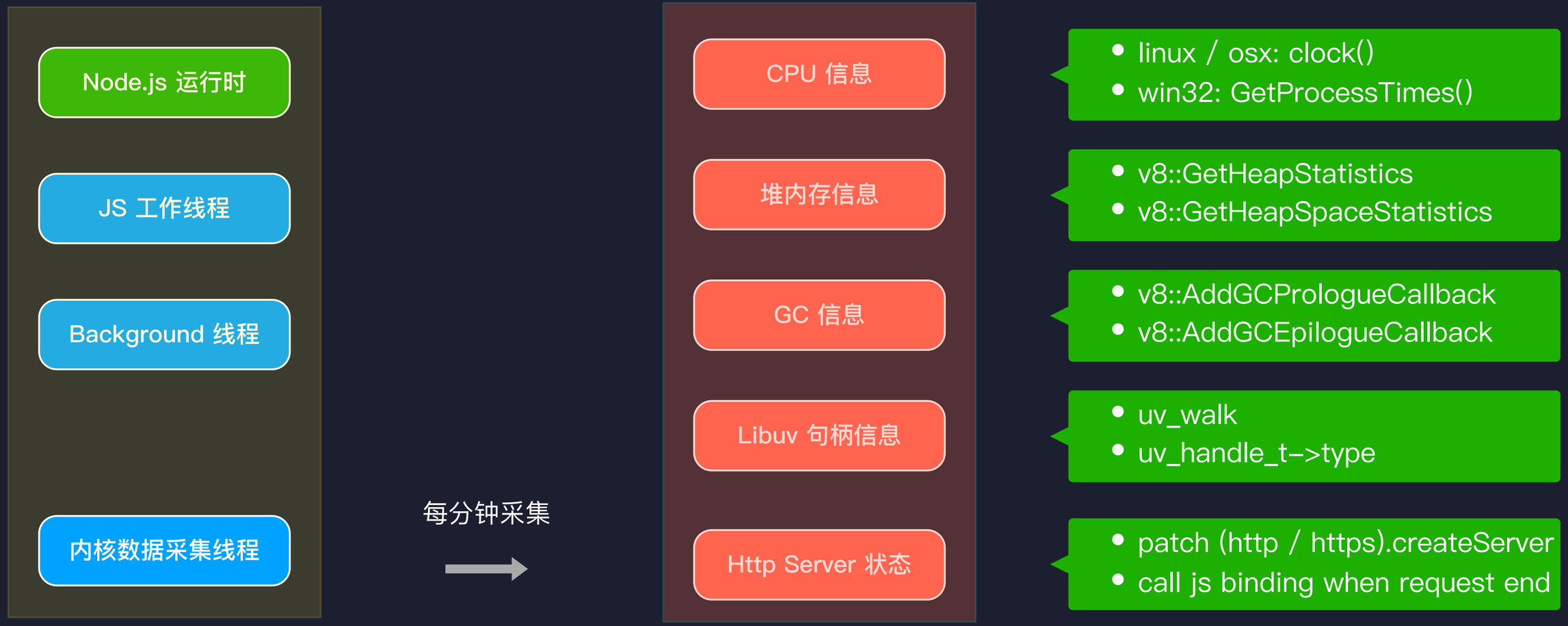
更强大的 Addon



- 启动日志 / 控制信令线程
- 根据配置添加致命错误钩子
- 根据配置 Monkey Patch

日志线程设计

基于 V8 / Libuv 暴露的内核数据 API



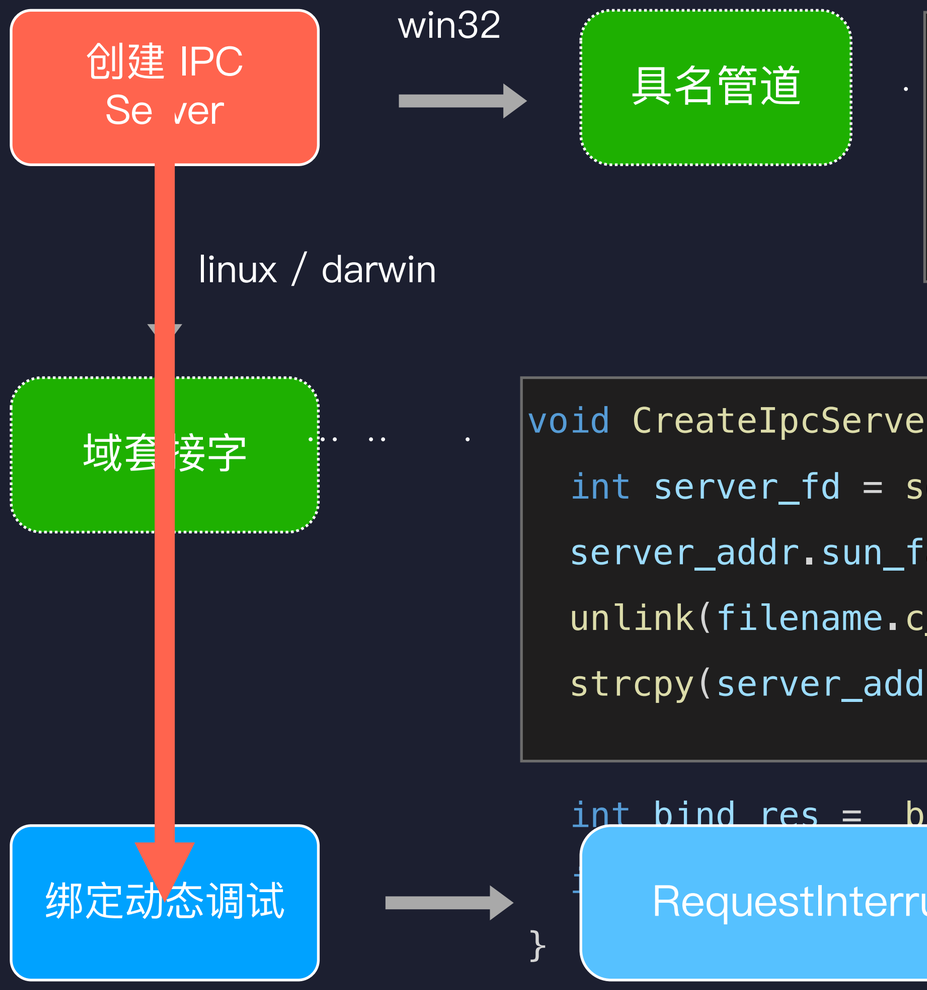
日志输出效果

```
[2020-07-19 13:10:32] [info] [cpu] [49176] [1.2.1] cpu_usage(%) cpu_now: 0.000000,
cpu_15: 0.000000, cpu_30: 0.000000, cpu_60: 0.000000

[2020-07-19 13:10:32] [info] [memory] [49176] [1.2.1] memory_usage(byte) rss: 2928640,
heap_used: 8178128, heap_available: 2190135496, heap_total: 10596352, heap_limit:
2197815296,, new_space_used: 91104, new_space_available: 956352

[2020-07-19 13:10:32] [info] [gc] [49176] [1.2.1] uptime: 870670, total_gc_times: 49,
total_gc_duration: 528, total_scavange_duration: 524, total_marksweep_duration: 4,
total_incremental_marking_duration: 0, gc_time_during_last_record: 0
```

信令线程设计

基于 V8 暴露的运行时调试 API

信令线程实现




```
#define CREATE_NAMED_PIPE                                                \
  named_pipe = CreateNamedPipeW(                                         \
      lp_name, PIPE_ACCESS_DUPLEX | FILE_FLAG_OVERLAPPED | WRITE_DAC,    \
      PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | PIPE_WAIT,                   \
      PIPE_UNLIMITED_INSTANCES, IN_AND_OUT_BUFFER_SIZE,                  \
```

```
void CreateIpcServer(void (*parsecmd)(char *)) {
  int server_fd = socket(AF_UNIX, SOCK_STREAM, 0);
  server_addr.sun_family = AF_UNIX;
  unlink(filename.c_str());
  strcpy(server_addr.sun_path, filename.c_str());

  int bind_res = bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr));
}
```

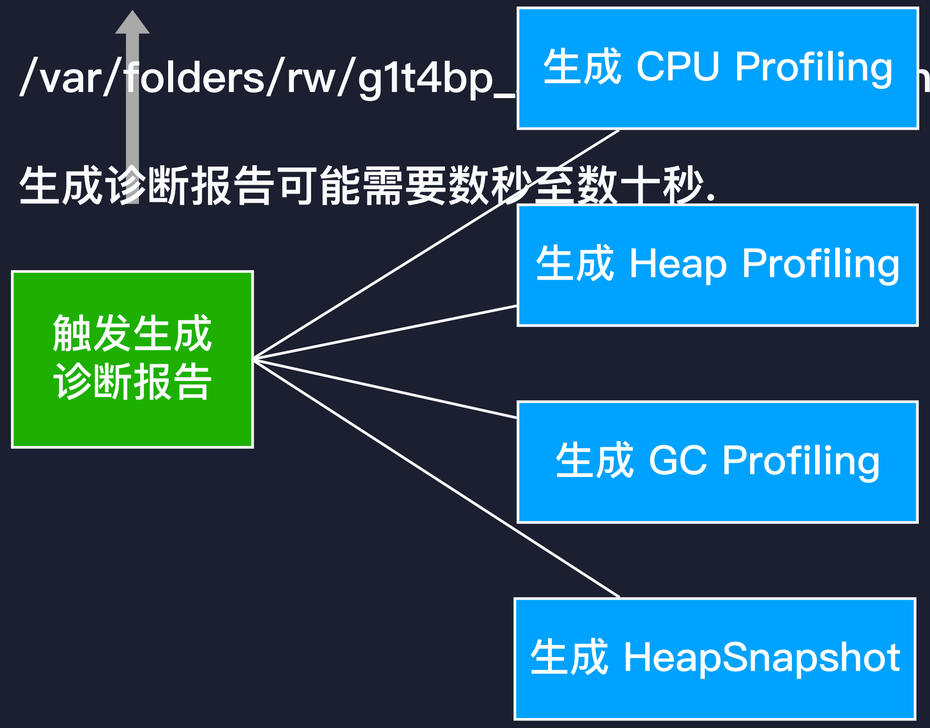
信令控制效果

```
hyj1991$ xprofctl diag_report –p 49176
```

诊断报告文件路径:

/var/folders/rw/g1t4bp_ h0000gn/T/x-diagreport-491

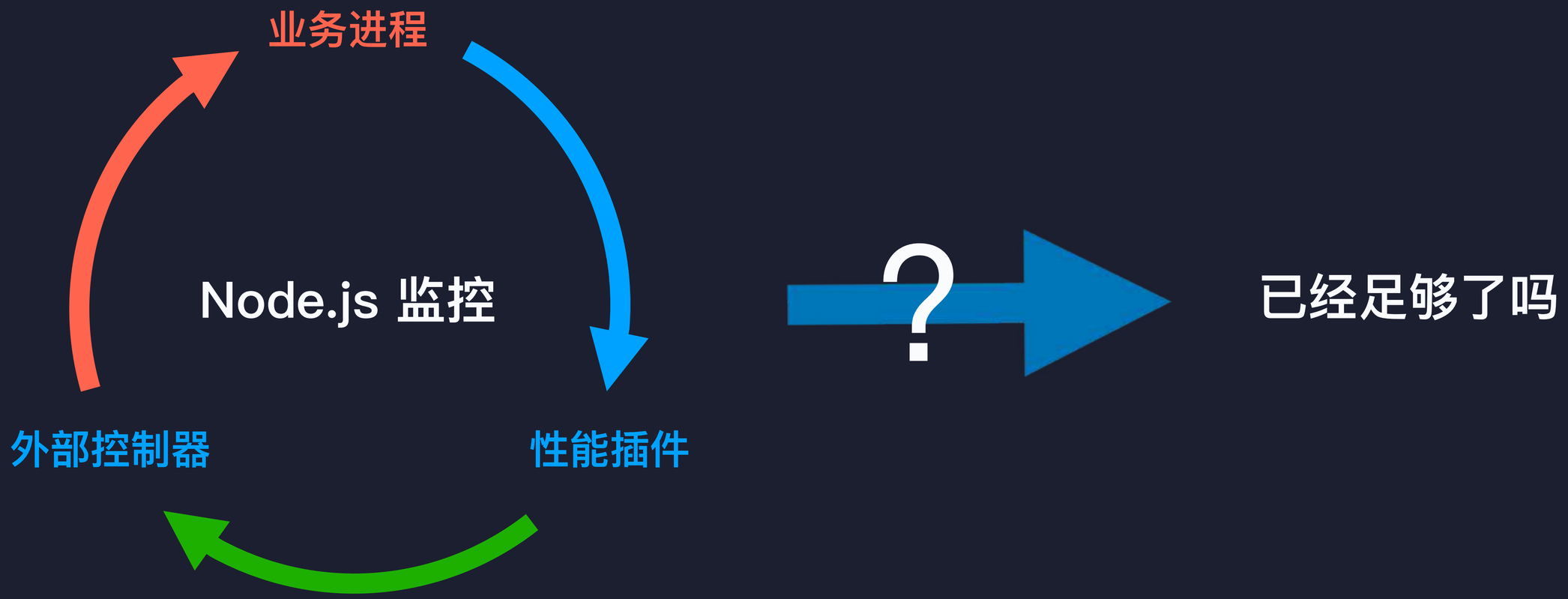
生成诊断报告可能需要数秒至数十秒.




```
{
  "pid": 49176,
  "location": "Active Dump",
  "message": "Active Dump",
  "nodeVersion": "v12.18.0",
  "osVersion": "Darwin Kernel 19.5.0",
  "loadTime": "2020-07-09 11:19:22",
  "dumpTime": "2020-07-19 21:45:31",
  "vmState": "EXTERNAL",
  "jsStacks": [],
  "nativeStacks": [],
  "heapStatistics": {},
  "heapSpaceStatistics": [],
  "libuvHandles": [],
```

方案对比

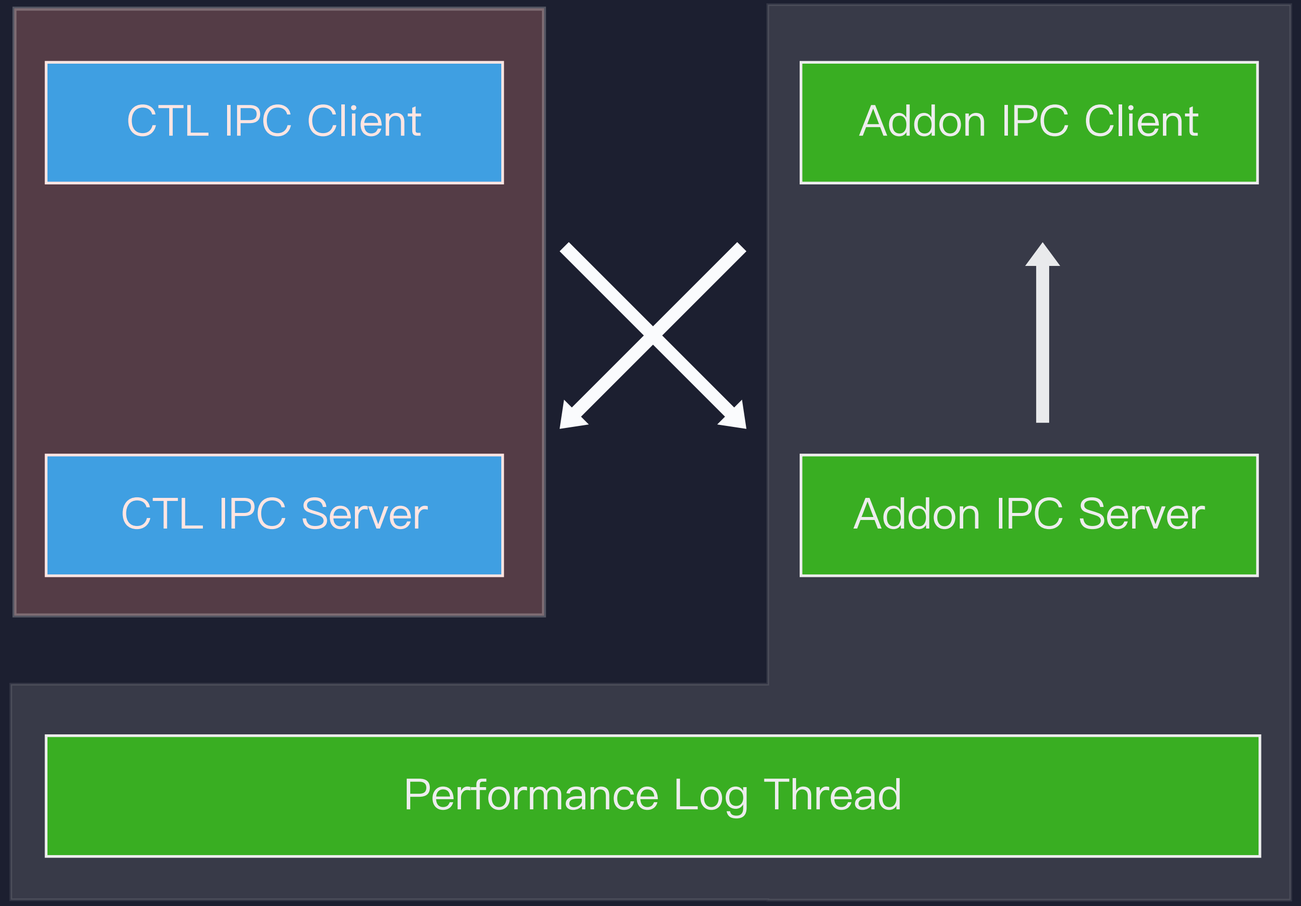
Runtime OR **Addon** ?


业务进程
Node.js 监控
外部控制器
性能插件
?
已经足够了吗

目录

完整的插件包

JS 控制器

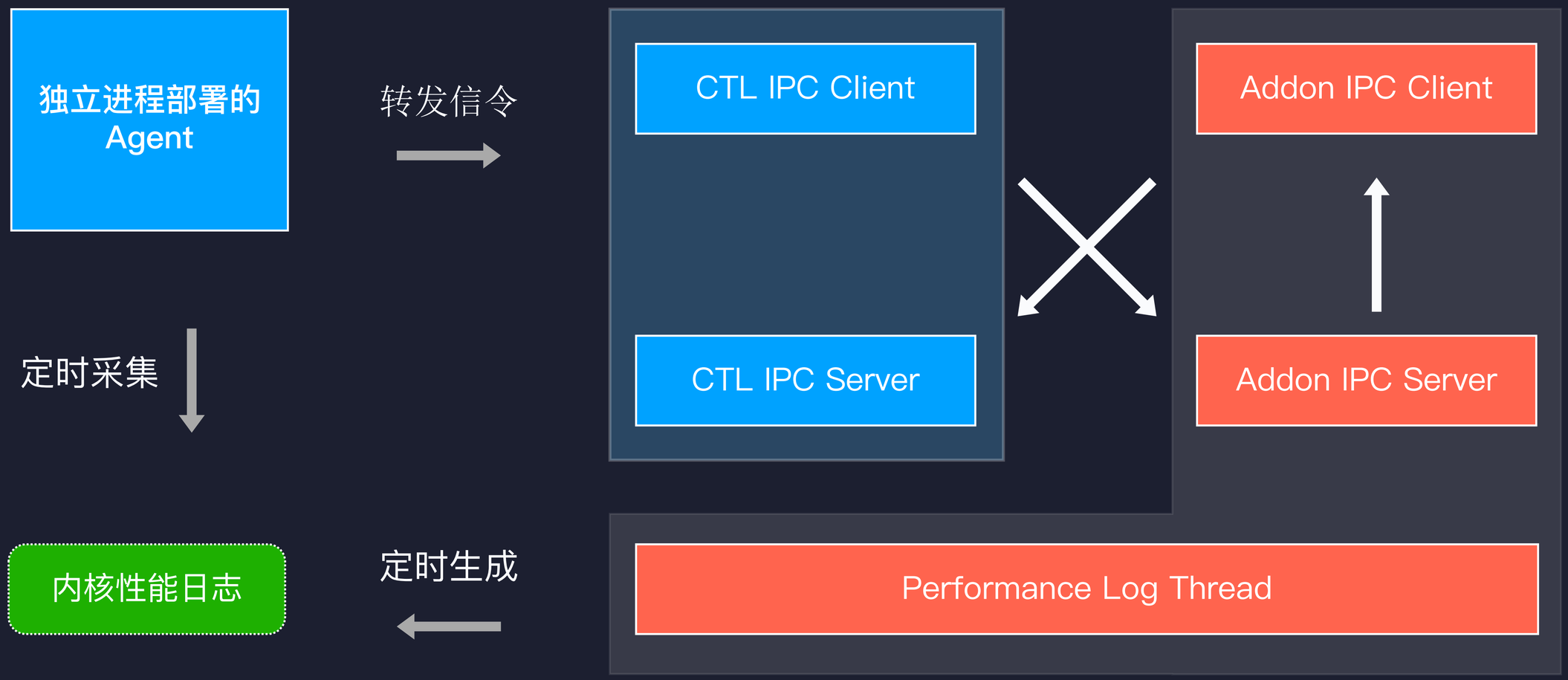


Node.js Addon

方案对比

直接上报 VS 采集 / 控制分离？

日志采集 / 信令转发


独立进程部署的 Agent
转发信令
定时采集
内核性能日志
定时生成
CTL IPC Client
CTL IPC Server
Addon IPC Client
Addon IPC Server
Performance Log Thread

数据采集与处理服务

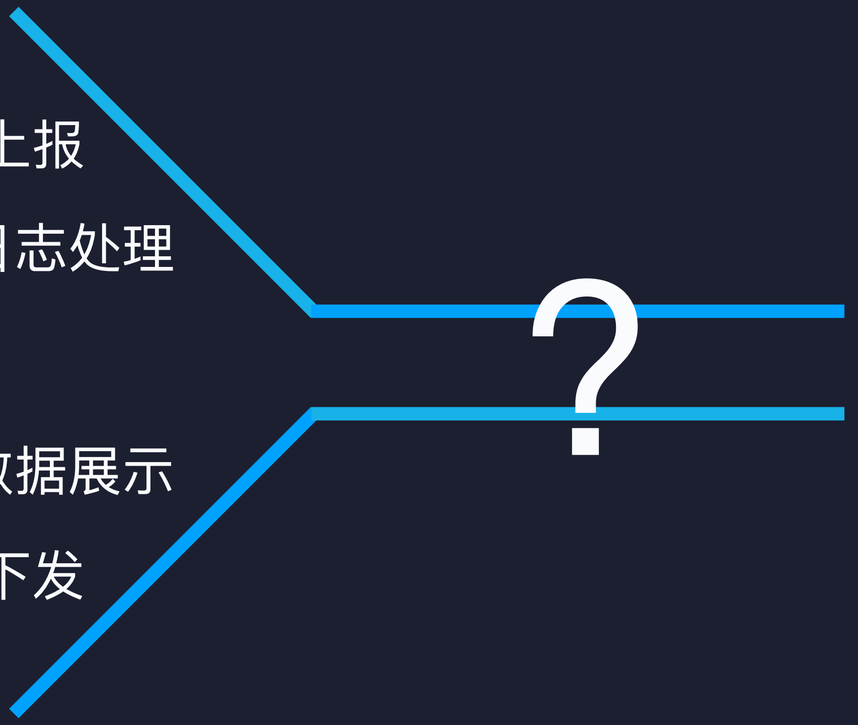

日志数据存储存储
钉钉机器人
邮件告警
企业微信
日志处理与存储服务
http
Agent 长连接服务
ws
独立进程部署的Agent

用户控制台

- 查看进程内核 **数据趋势**
- 下发 **Profiling** / **Snapshot** 指令

业务进程

- 插件上报
  - 日志处理
  - 数据展示
- 指令下发



已经足够了吗

完善控制台 —— 团队协作

1. 新增团队成员

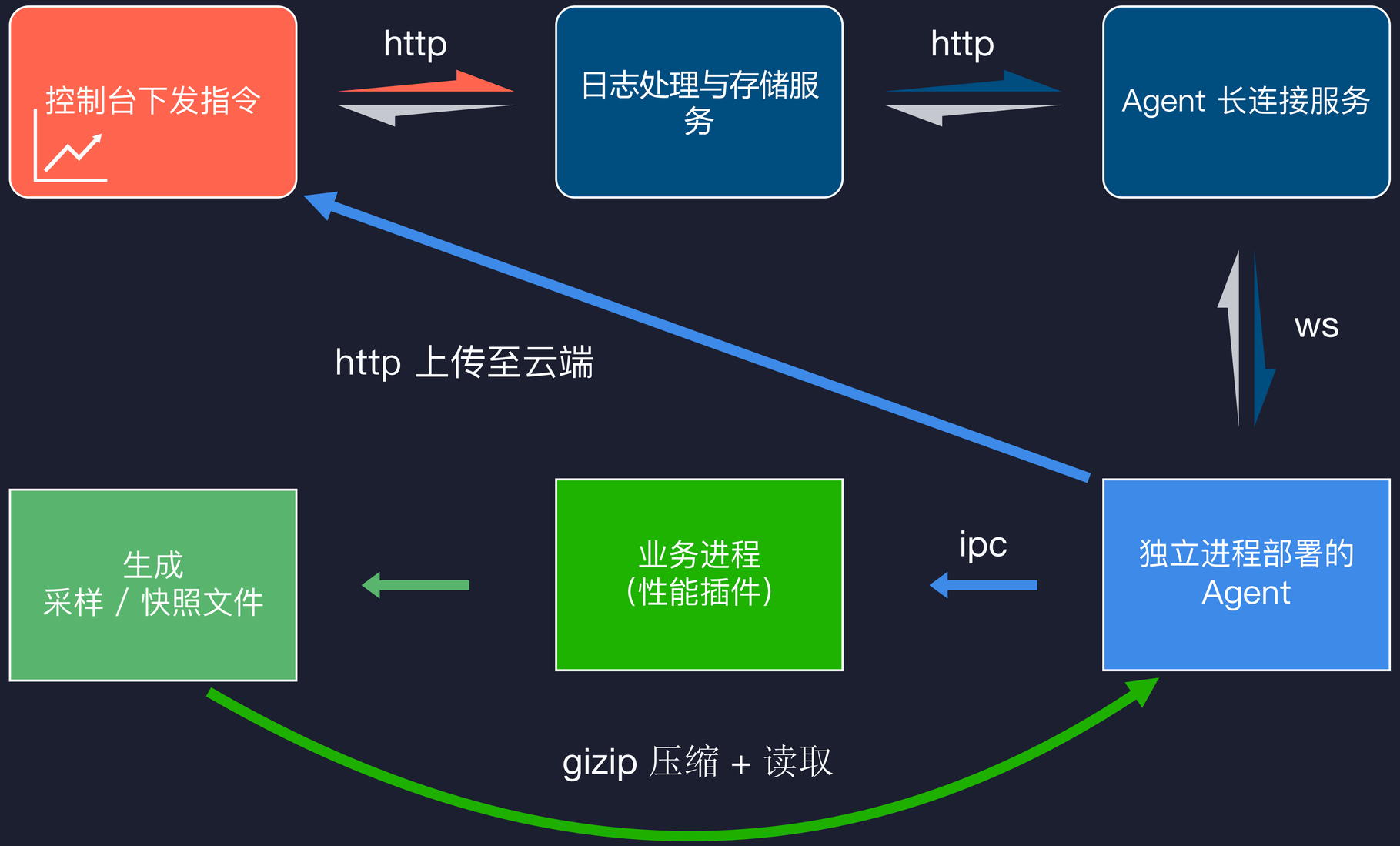
2. 移除团队成员

3. 管理团队成员

成员信息	当前状态	邀请 / 加入时间	团队操作
奕钧	管理员	2020-06-18 09:17:52	
穆客	邀请中	2020-07-20 09:17:52	撤回邀请
朴灵	邀请中	2020-07-21 09:17:52	撤回邀请
TZ | 天猪	已加入	2020-06-26 09:17:52	转交应用 移除成员

完善控制台 —— 自定义告警

完善控制台 —— 文件管理

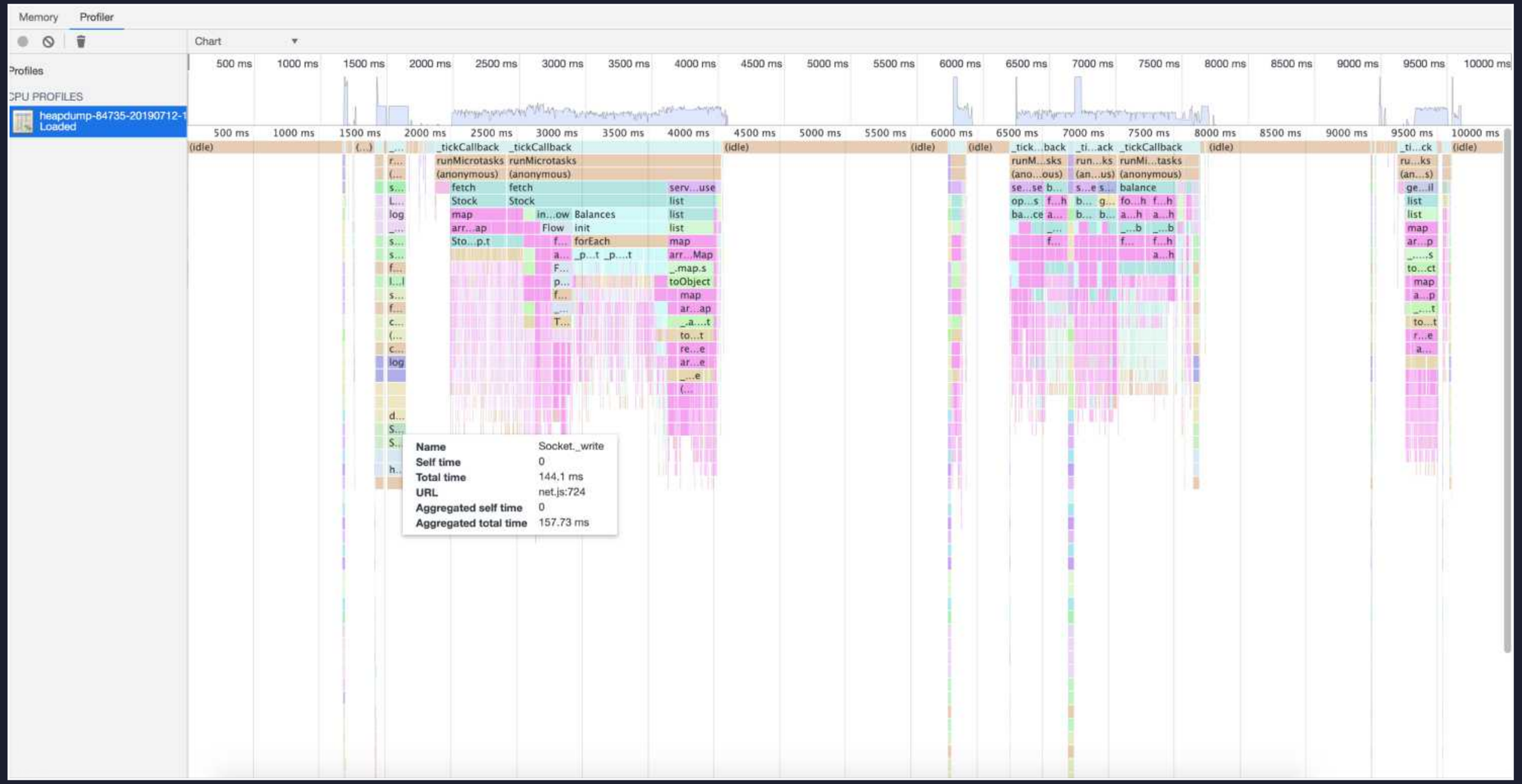

控制台下发指令
http
日志处理与存储服务
http
Agent 长连接服务
ws
http 上传至云端
生成
采样 / 快照文件
业务进程
(性能插件)
ipc
独立进程部署的
Agent
gizip 压缩 + 读取

完善控制台 —— 服务化分析

- CPU Profile
- GC Profile
- NodeReport

完善控制台 —— 服务化分析

- CPU Profile
- Heap Profile
- HeapSnapshot



完整的应用故障定位步骤

根据告警类型执行
Profiling / Snapshot

问题应用
生成性能分析文件

性能分析文件转储
至监控服务端

根据服务化分析结果
定位故障点

小结：复杂丢给设计，简洁留给用户

用户简单易用

- 仅需在 Node.js 应用入口引入插件模块
- 低侵入性，无需更改现有业务代码

监控功能完备

- 针对 Node.js 应用指标的性能监控
- 灵活可配置的自定义运维阈值告警
- 可实时导出的 Runtime 采样与快照

存量应用逻辑不变

无痛接入

前述 Node.js 性能监控方案已开源

插件模块：https://github.com/X-Profiler/xprofiler

项目地址：https://github.com/hyj1991/easy-monitor

目录

案例一：内存泄露

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

// 业务处理理...
a, b, c
```

```
delete require.cache[require.resolve('a')];
delete require.cache[require.resolve('b')];
delete require.cache[require.resolve('c')];
```

```
'use strict';

const Controller = require('egg').Controller;

class HomeController extends Controller {
  async index() {
    const { ctx } = this;

    require('./Operator');
    require('./School');
    require('./User');
    require('./OperatorMenu');

    // 业务逻辑...
    ctx.body = 'hi, egg';

    delete require.cache[require.resolve('./Operator')];
```

热更新逻辑

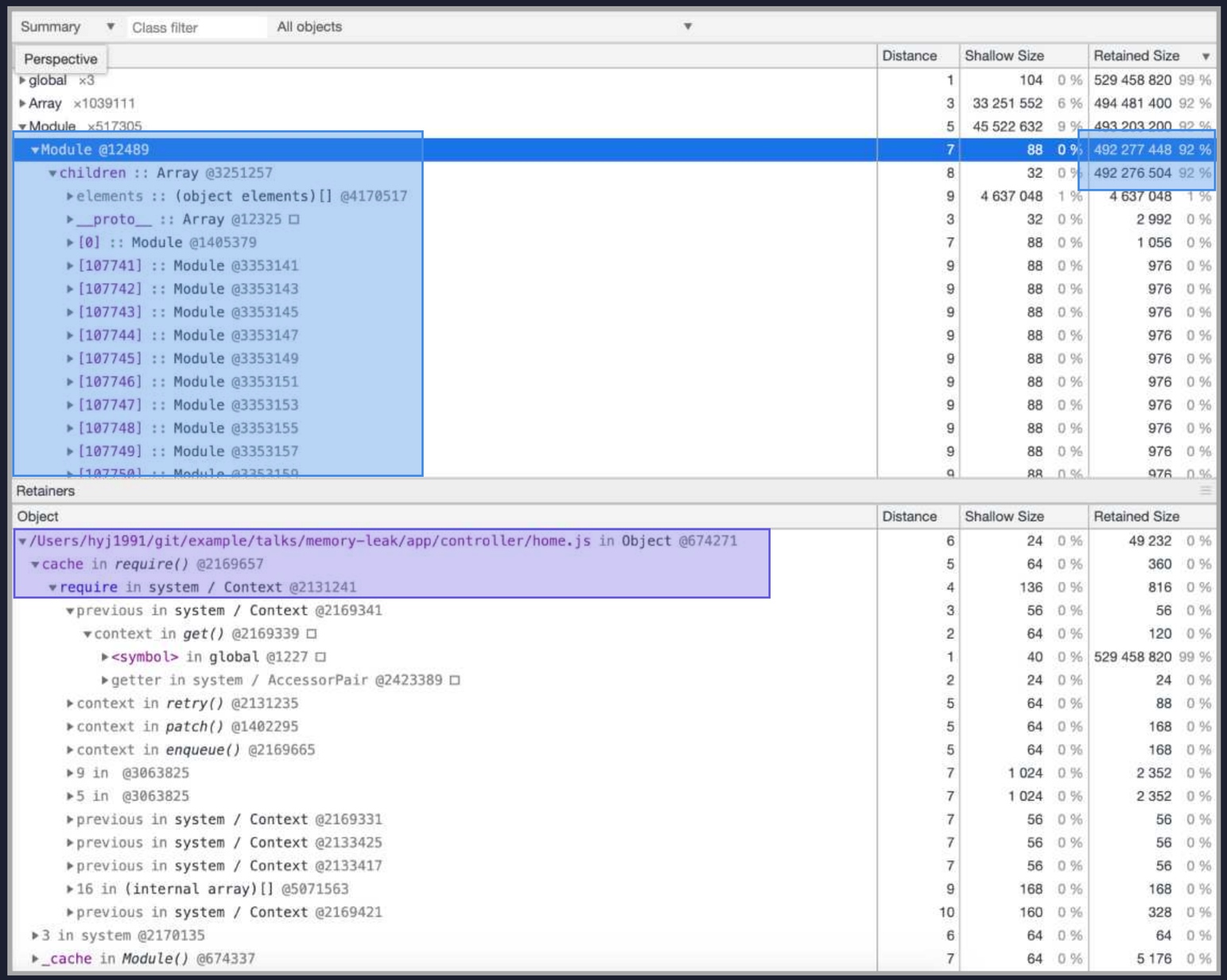

模块 A 的 V1 版本
模块 B 的 V1 版本
delete require.cache['模块 A']
require('模块 A')
模块 A 的 V2 版本
模块 B 的 V1 版本
Node.js 应用
Node.js 应用

堆内存趋势

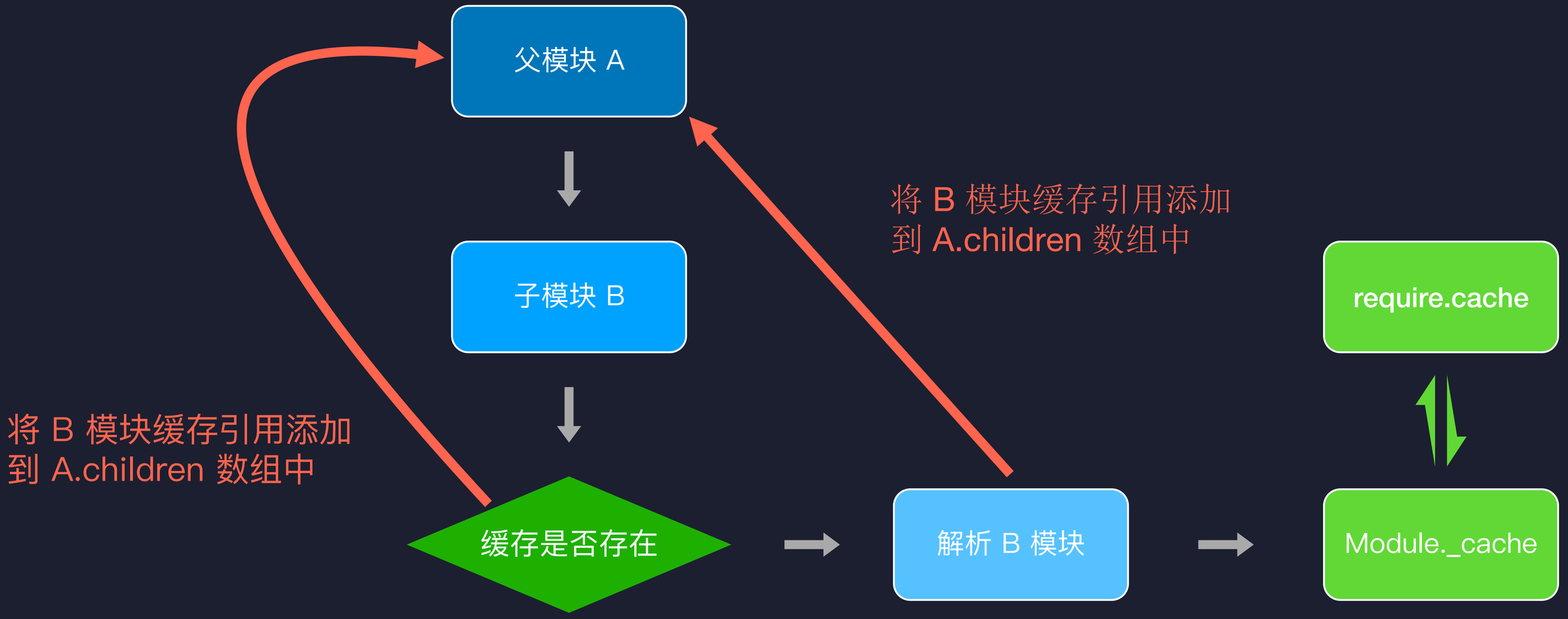
项目发布后内存
快速增长

定位内存泄露点

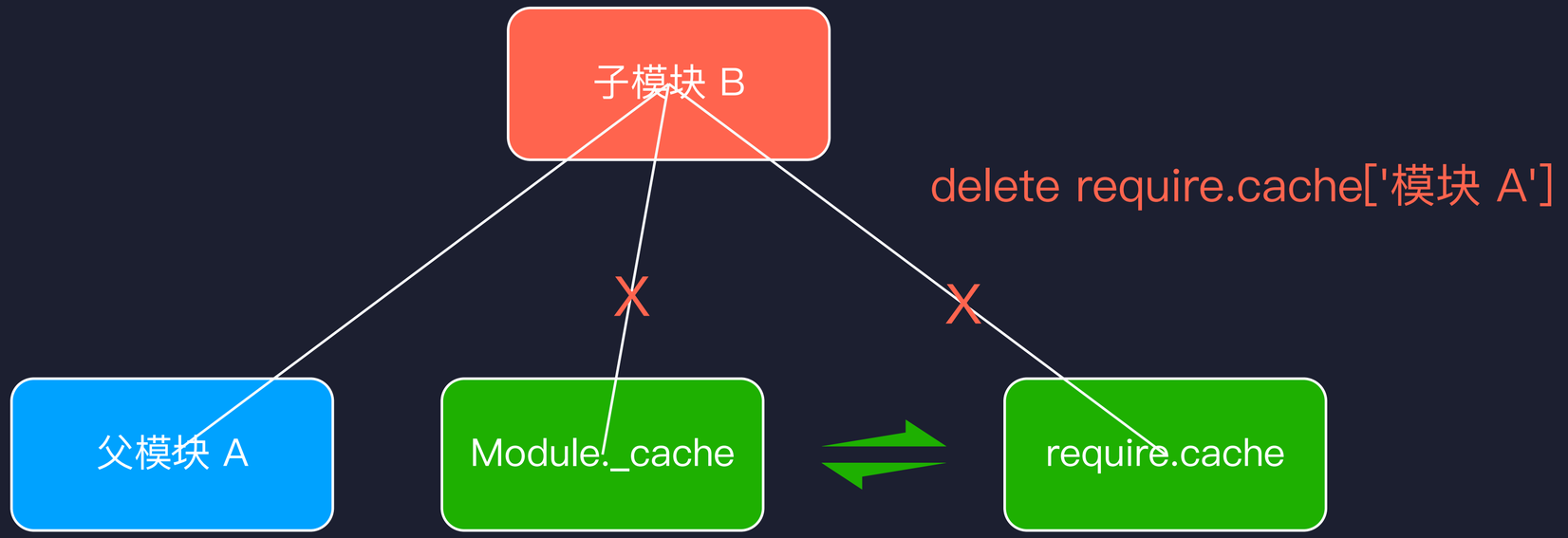

进程详细信息
查看 X-Profiler 插件状态
数据趋势
保存数据
抓取性能数据
CPU Profile
堆快照
Heap Profile
GC 追踪
Node.js 实时诊断


Summary
Class filter
All objects
Perspective
Distance
Shallow Size
Retained Size
global ×3 1 104 0 % 529 458 820 99 %
Array ×1039111 3 33 251 552 6 % 494 481 400 92 %
Module ×517305 5 45 522 632 9 % 493 203 200 92 %
Module @12489 7 88 0 % 492 277 448 92 %
children :: Array @3251257 8 32 0 % 492 276 504 92 %
elements :: (object elements)[] @4170517 9 4 637 048 1 % 4 637 048 1 %
__proto__ :: Array @12325 3 32 0 % 2 992 0 %
[0] :: Module @1405379 7 88 0 % 1 056 0 %
[107741] :: Module @3353141 9 88 0 % 976 0 %
[107742] :: Module @3353143 9 88 0 % 976 0 %
[107743] :: Module @3353145 9 88 0 % 976 0 %
[107744] :: Module @3353147 9 88 0 % 976 0 %
[107745] :: Module @3353149 9 88 0 % 976 0 %
[107746] :: Module @3353151 9 88 0 % 976 0 %
[107747] :: Module @3353153 9 88 0 % 976 0 %
[107748] :: Module @3353155 9 88 0 % 976 0 %
[107749] :: Module @3353157 9 88 0 % 976 0 %
Retainers
Object
Distance
Shallow Size
Retained Size
/Users/hyj1991/git/example/talks/memory-leak/app/controller/home.js in Object @674271 6 24 0 % 49 232 0 %
cache in require() @2169657 5 64 0 % 360 0 %
require in system / Context @2131241 4 136 0 % 816 0 %
previous in system / Context @2169341 3 56 0 % 56 0 %
context in get() @2169339 2 64 0 % 120 0 %
<symbol> in global @1227 1 40 0 % 529 458 820 99 %
getter in system / AccessorPair @2423389 2 24 0 % 24 0 %
context in retry() @2131235 5 64 0 % 88 0 %
context in patch() @1402295 5 64 0 % 168 0 %
context in enqueue() @2169665 5 64 0 % 168 0 %
9 in @3063825 7 1 024 0 % 2 352 0 %
5 in @3063825 7 1 024 0 % 2 352 0 %
previous in system / Context @2169331 7 56 0 % 56 0 %
previous in system / Context @2133425 7 56 0 % 56 0 %
previous in system / Context @2133417 7 56 0 % 56 0 %
16 in (internal array)[] @5071563 9 168 0 % 168 0 %
previous in system / Context @2169421 10 160 0 % 328 0 %
3 in system @2170135 6 64 0 % 64 0 %
_cache in Module() @674337 7 64 0 % 5 176 0 %

模块加载机制


父模块 A
子模块 B
缓存是否存在
解析 B 模块
Module._cache
require.cache
将 B 模块缓存引用添加
到 A.children 数组中
将 B 模块缓存引用添加
到 A.children 数组中

残留的引用


子模块 B
delete require.cache['模块 A']
X
X
父模块 A
Module._cache
require.cache

热更新引发的内存泄漏小结

- 热更新需要清除 **require.cache** 和 **所有父模块引用**
- 热更新模块中创建的 **定时器**、**Socket 连接** 等资源需要手动释放
- **不建议** 对线上 Node.js 进程做模块的热加载

基本上 100%
内存泄露

案例二：JS 主线程无限阻塞

```
'use strict';
const Controller = require('egg').Controller;

class RegexpController extends Controller {
  async index() {
    const { ctx } = this;
   let str = '<br/>                                                ' +
      '            早餐后自由活动，于指定时间集合自行办理退房手续。';
    str += '<br/>                                                <br/>' +
      '                                              <br/>            ' +
      '                           <br/>';
    str += '                                            <br/>' +
      '                                                              ' +
      '                                                              ' +
      '        <br/>';
    str += '                                                    <br/>
<br/>';
    str += '                                                          ' +
```

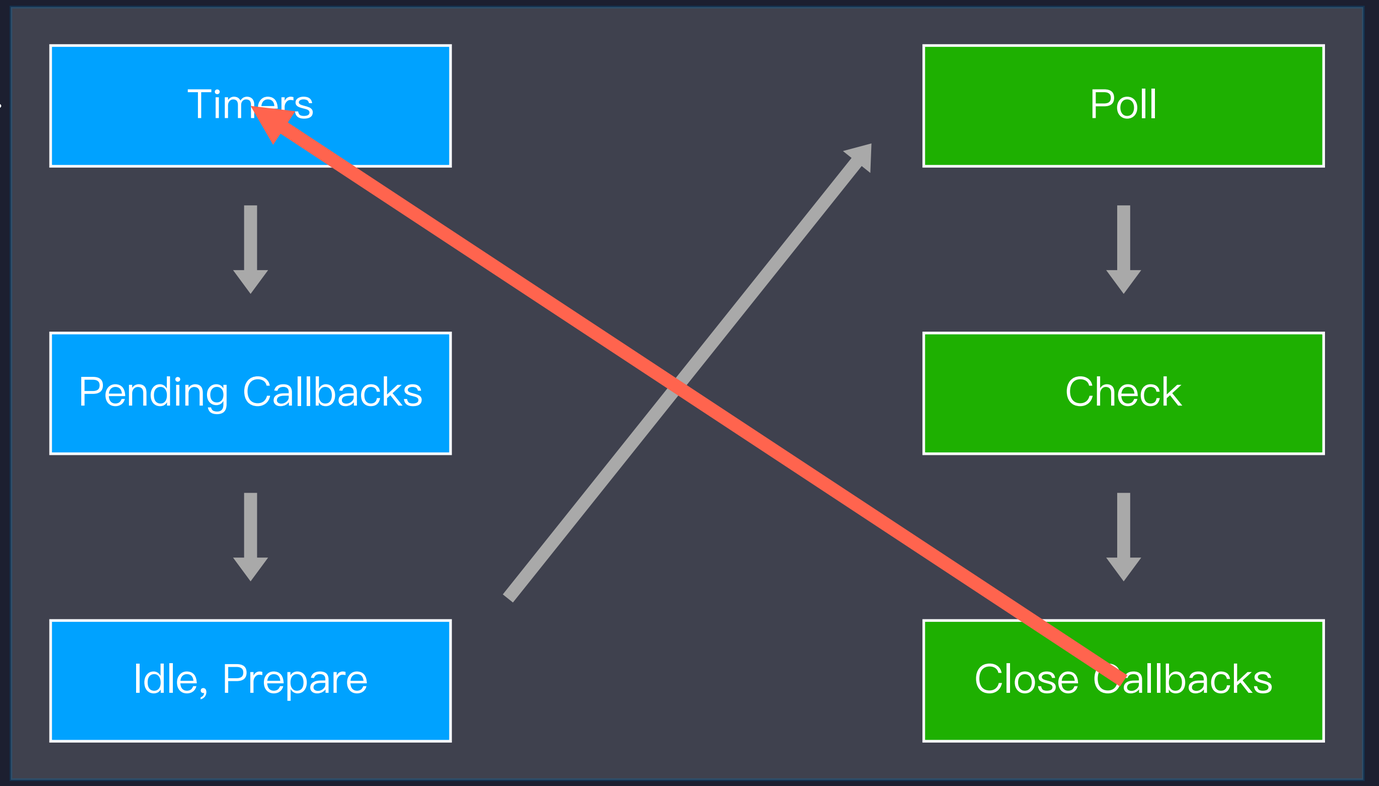
CPU 趋势

用户访问此页面

CPU 打满

事件循环

setTimeout,
setInterval



onnections,
incoming data,
etc.

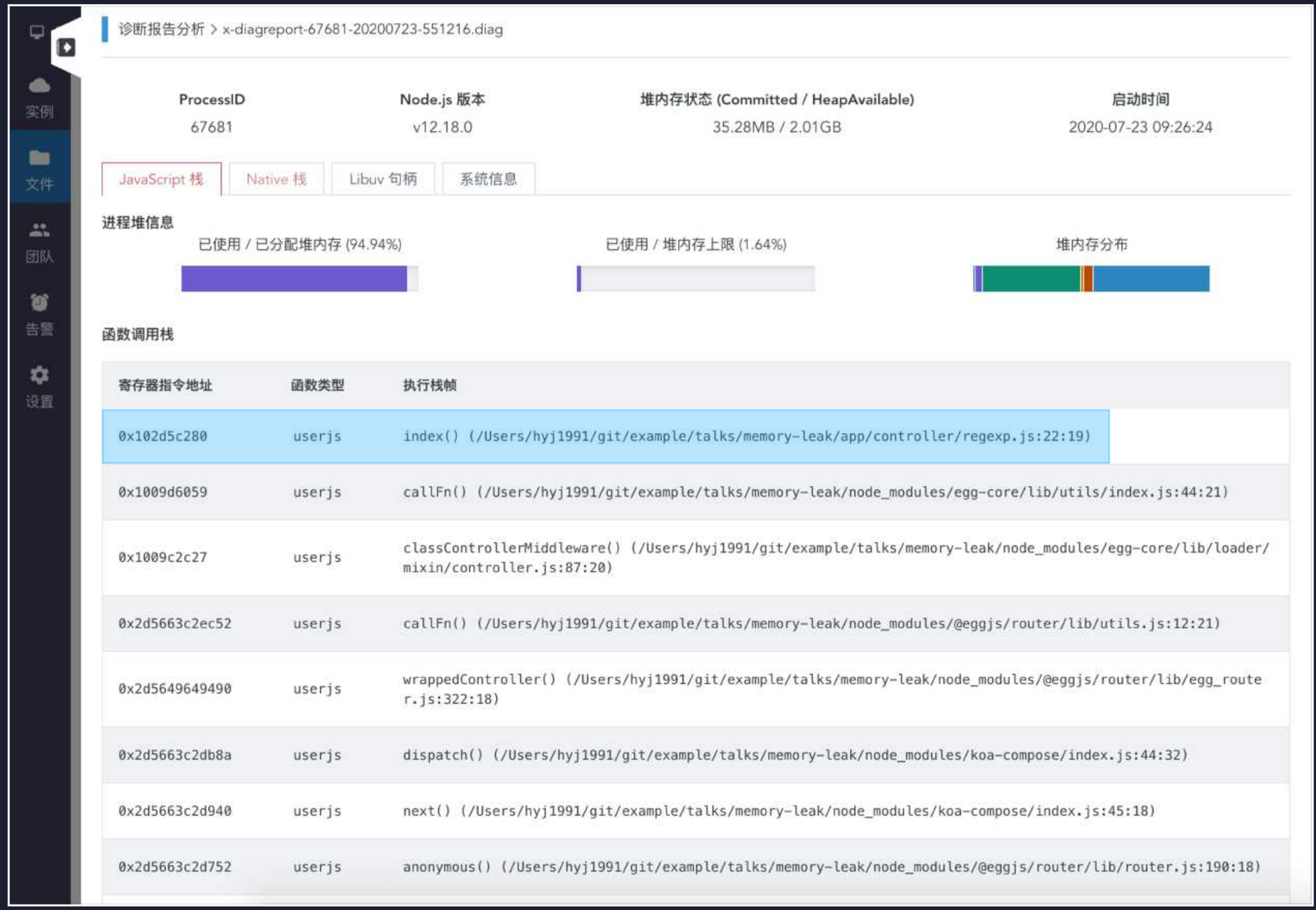
setImmediate

socket.on('close', ...)

Main Thread

定位业务阻塞点


进程详细信息
查看 X-Profiler 插件状态
数据趋势
保存数据
抓取性能数据
CPU Profile
堆快照
Heap Profile
GC 追踪
Node.js 实时诊断


实例
文件
团队
告警
设置
诊断报告分析 > x-diagreport-67681-20200723-551216.diag
ProcessID
67681
Node.js 版本
v12.18.0
堆内存状态 (Committed / HeapAvailable)
35.28MB / 2.01GB
启动时间
2020-07-23 09:26:24
JavaScript 栈
Native 栈
Libuv 句柄
系统信息
进程堆信息
已使用 / 已分配堆内存 (94.94%)
已使用 / 堆内存上限 (1.64%)
堆内存分布
函数调用栈
寄存器指令地址 函数类型 执行栈帧
0x102d5c280 userjs index() (/Users/hyj1991/git/example/talks/memory-leak/app/controller/regexp.js:22:19)
0x1009d6059 userjs callFn() (/Users/hyj1991/git/example/talks/memory-leak/node_modules/egg-core/lib/utils/index.js:44:21)
0x1009c2c27 userjs classControllerMiddleware() (/Users/hyj1991/git/example/talks/memory-leak/node_modules/egg-core/lib/loader/mixin/controller.js:87:20)
0x2d5663c2ec52 userjs callFn() (/Users/hyj1991/git/example/talks/memory-leak/node_modules/@eggjs/router/lib/utils.js:12:21)
0x2d5649649490 userjs wrappedController() (/Users/hyj1991/git/example/talks/memory-leak/node_modules/@eggjs/router/lib/egg_router.js:322:18)
0x2d5663c2db8a userjs dispatch() (/Users/hyj1991/git/example/talks/memory-leak/node_modules/koa-compose/index.js:44:32)
0x2d5663c2d940 userjs next() (/Users/hyj1991/git/example/talks/memory-leak/node_modules/koa-compose/index.js:45:18)
0x2d5663c2d752 userjs anonymous() (/Users/hyj1991/git/example/talks/memory-leak/node_modules/@eggjs/router/lib/router.js:190:18)

WE ARE JUST  THE WAY

THANKS

QCon+ 案例研习社