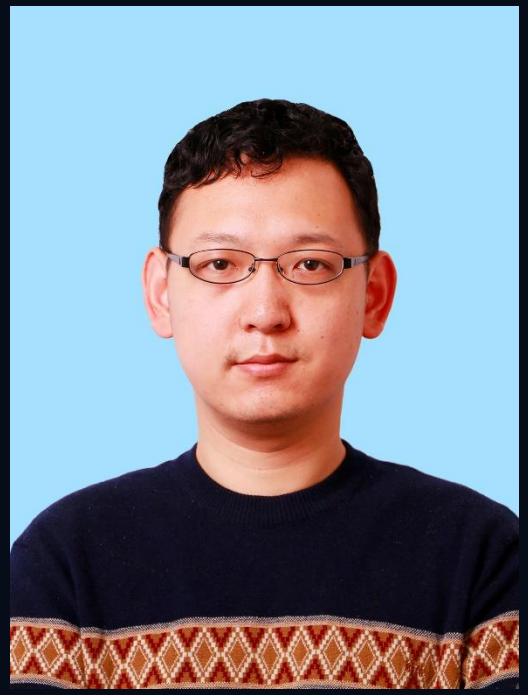
KNode
– 打造快手高性能Node.js应用

张屾
平台研发部工程师

自我介绍



张屾

个人经历：

2011 ~ 2015 搜狗

2015 ~ 2018 百度

2018 ~ 至今 快手

目录

1. KNode是什么？产生的背景？
2. Node.js基础监控能力建设
3. KNode展望未来

KNode是什么

狭义上：KNode是一个定制的Node.js运行时

广义上：预期做成快手一站式的Node.js解决方案

KNode产生的背景

1. 快手各业务线都有Node.js的身影

KNode产生的背景

express

BFF

点播

主站

音视频

eggjs

BigPipe

海外

短视频

商业化

直播

koa

GraphQL

电商

AjaxPipe

中台

KNode产生的背景

1. 快手各业务线都有Node.js的身影
2. 春晚一战暴露了一些问题
   - 基础监控粒度较粗，排障能力不足
   - 各业务团队方案不一，有简有繁
3. 技术团队想在Node.js领域深耕

Node.js基础监控能力建设

如何做好监控、提升排障能力？

Node.js基础监控能力建设

如何做好监控、提升排障能力？

数据生产

Node.js基础监控能力建设

如何做好监控、提升排障能力？

数据生产

- 常规日志：服务启动后间隔输出运行情况
- 按需日志：服务出现不符预期的时候触发

Node.js基础监控能力建设

如何做好监控、提升排障能力？

数据生产

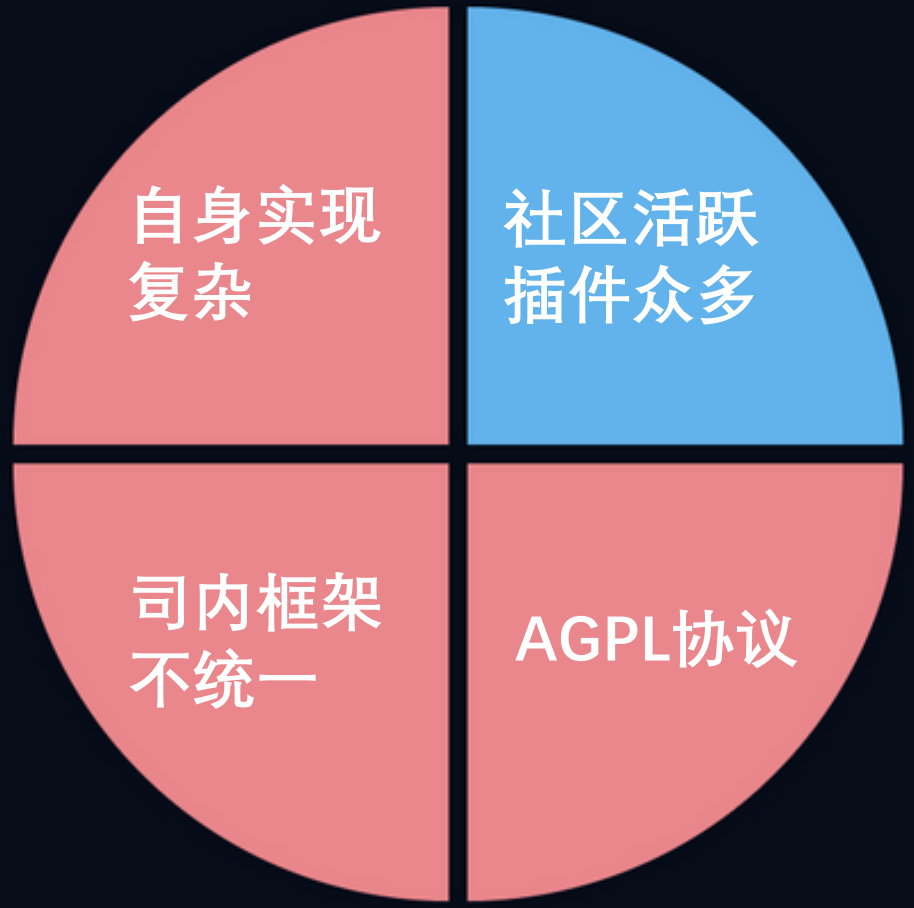
- 通过pm2的监控插件和日志
- 通过Node.js扩展或js模块的方式
- 通过定制Node.js源代码

Node.js基础监控能力建设

如何做好监控、提升排障能力？

数据生产

- 通过pm2的监控插件和日志
- 通过Node.js扩展或js模块的方式
- 通过定制Node.js源代码

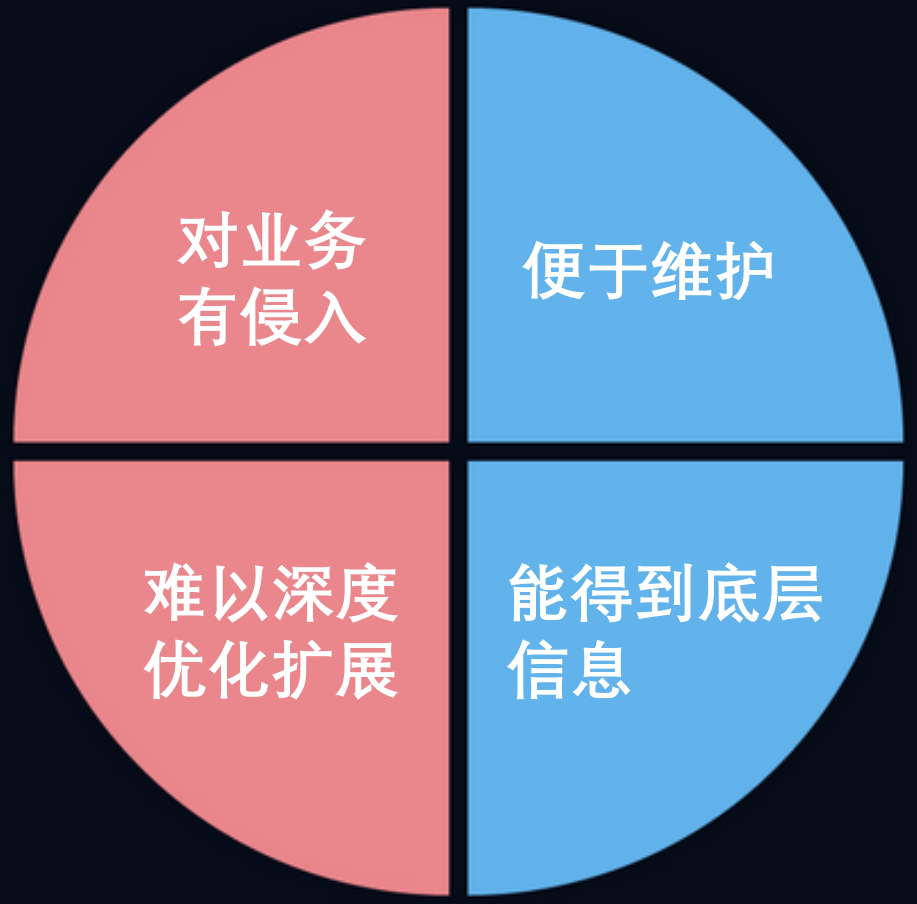


Node.js基础监控能力建设

如何做好监控、提升排障能力？

数据生产

- 通过pm2的监控插件和日志
- 通过Node.js扩展或js模块的方式
- 通过定制Node.js源代码

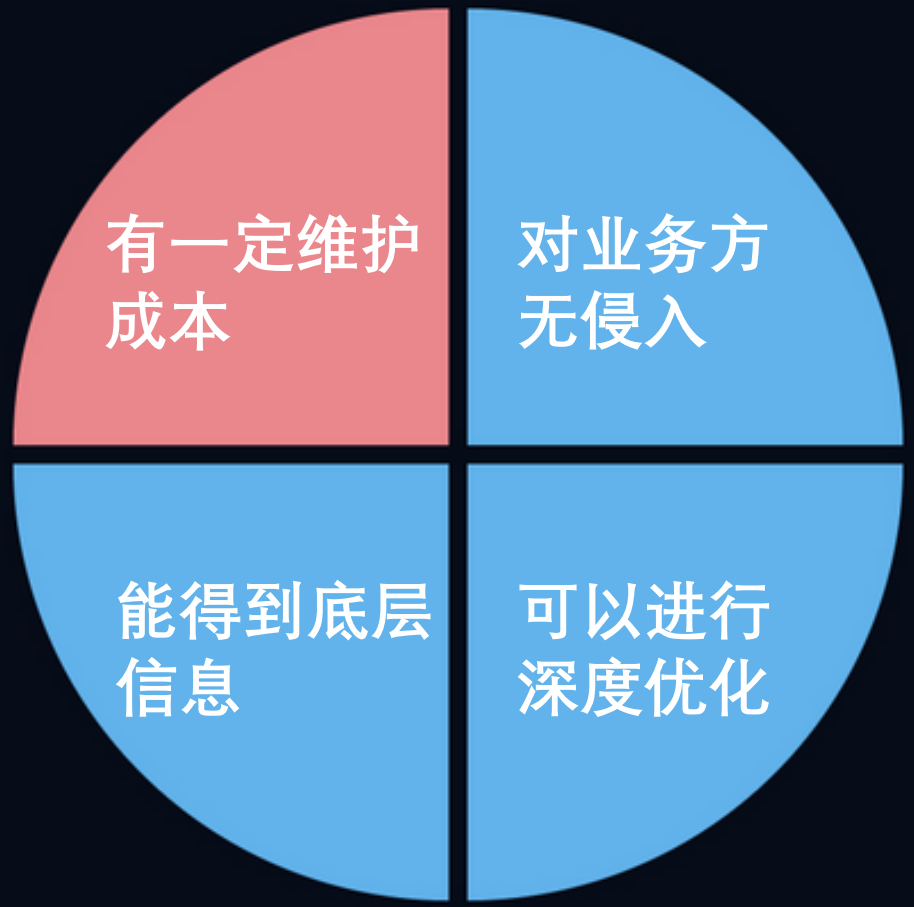


Node.js基础监控能力建设

如何做好监控、提升排障能力？

数据生产

- 通过pm2的监控插件和日志
- 通过Node.js扩展或js模块的方式
- 通过定制Node.js源代码

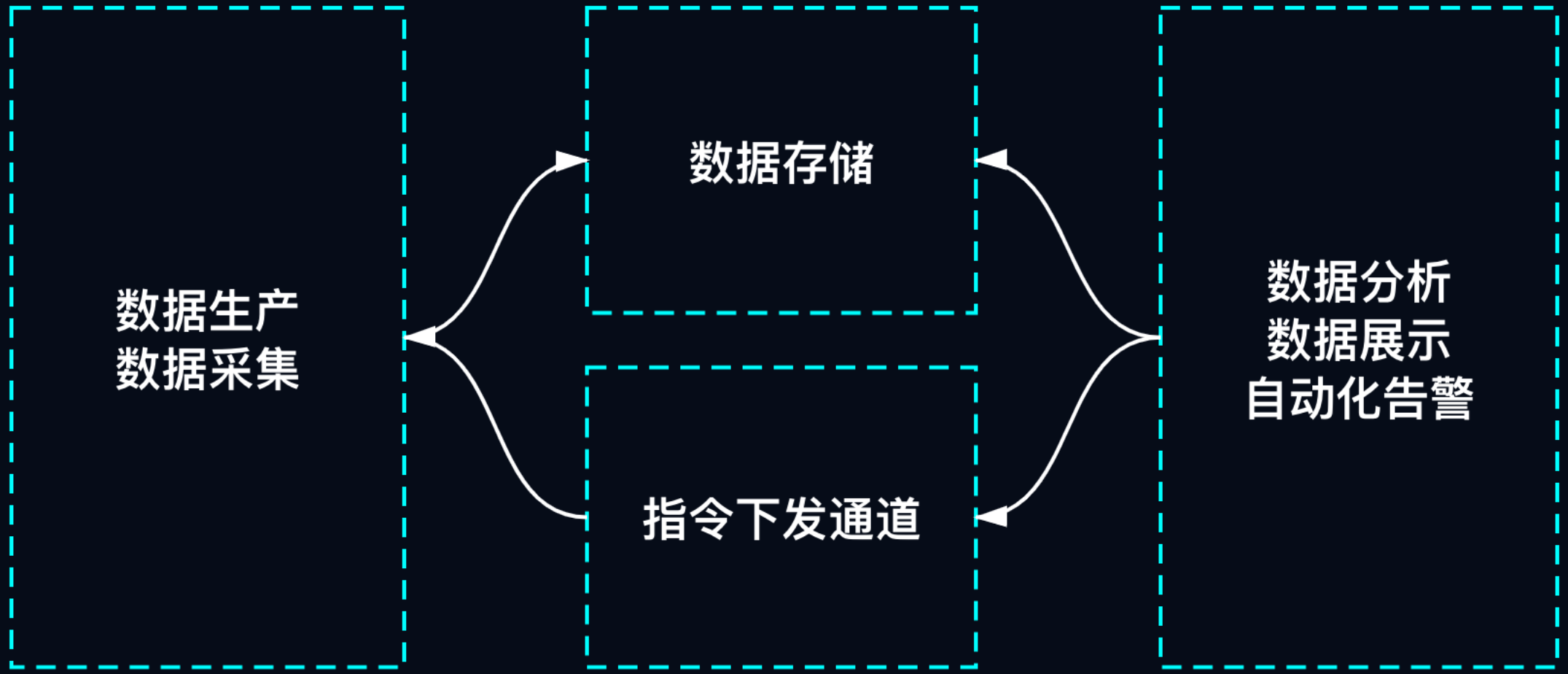


Node.js基础监控能力建设

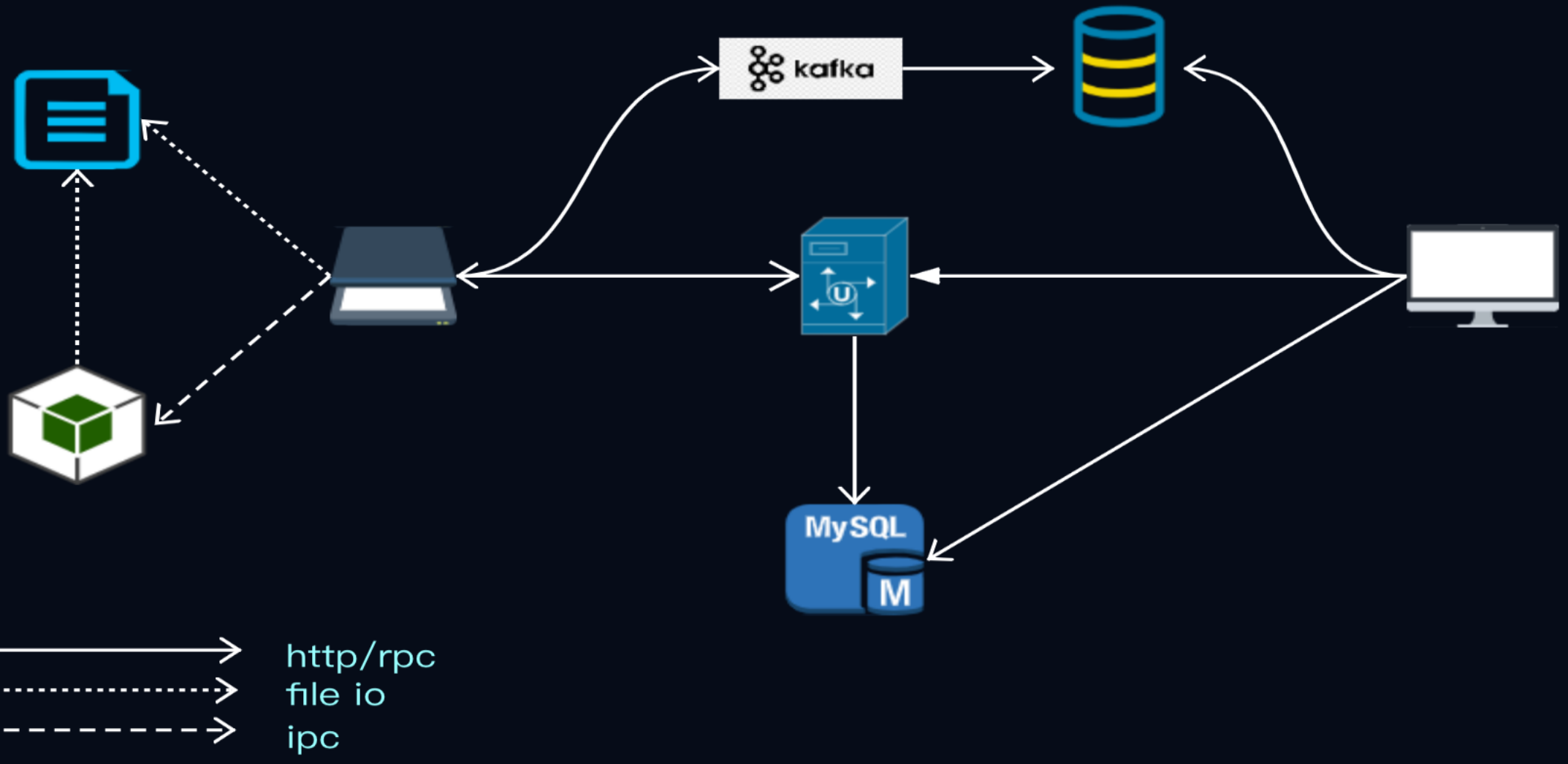
如何做好监控、提升排障能力？

- 数据生产 → 定制运行时
- 数据采集 → 容器环境统一日志收集器
- 数据展现与分析 → 提供平台，集成dashboard和devtool
- 自动化告警 → 平台打通司内基于prometheus的告警系统

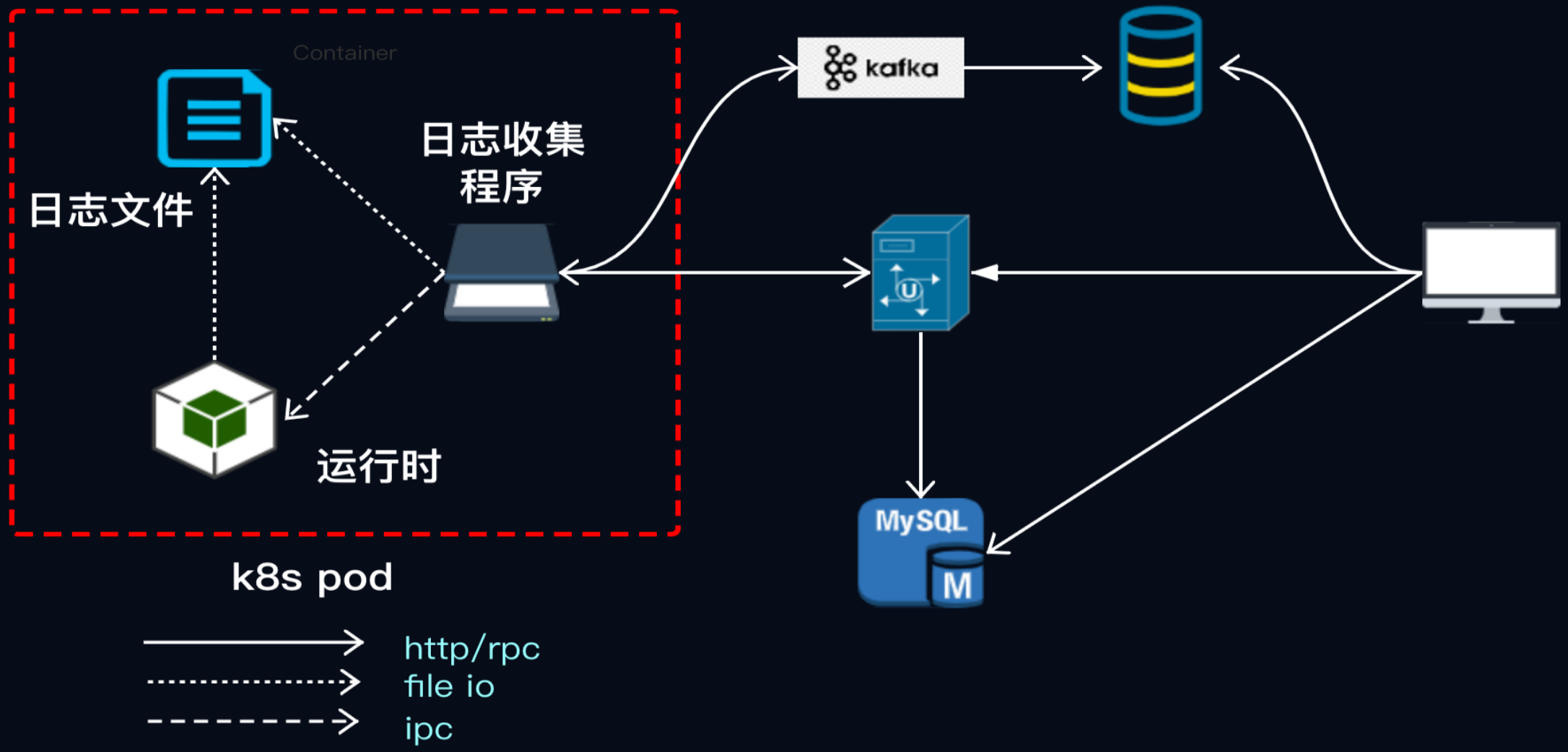
Node.js基础监控能力建设



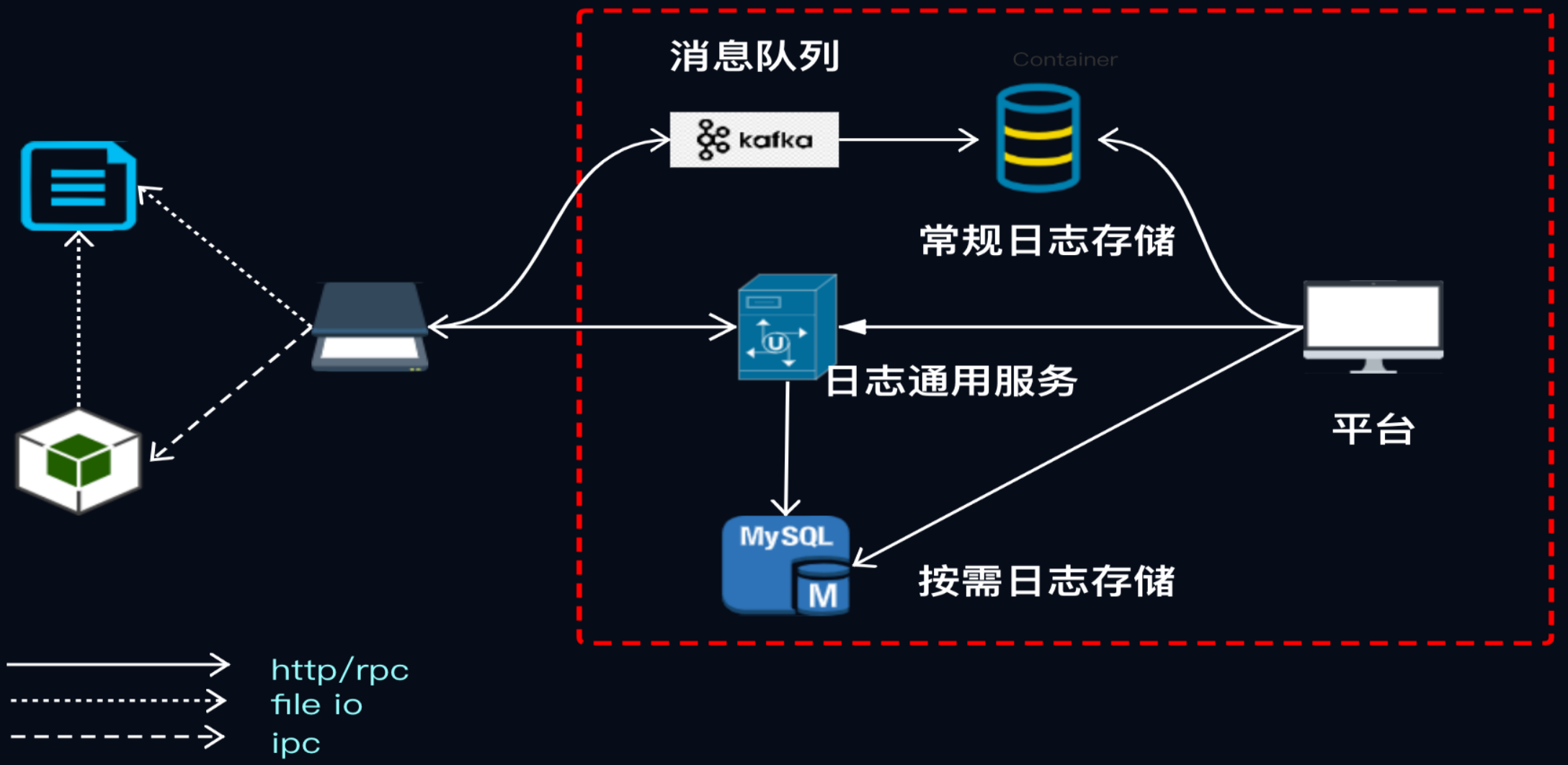
Node.js基础监控能力建设



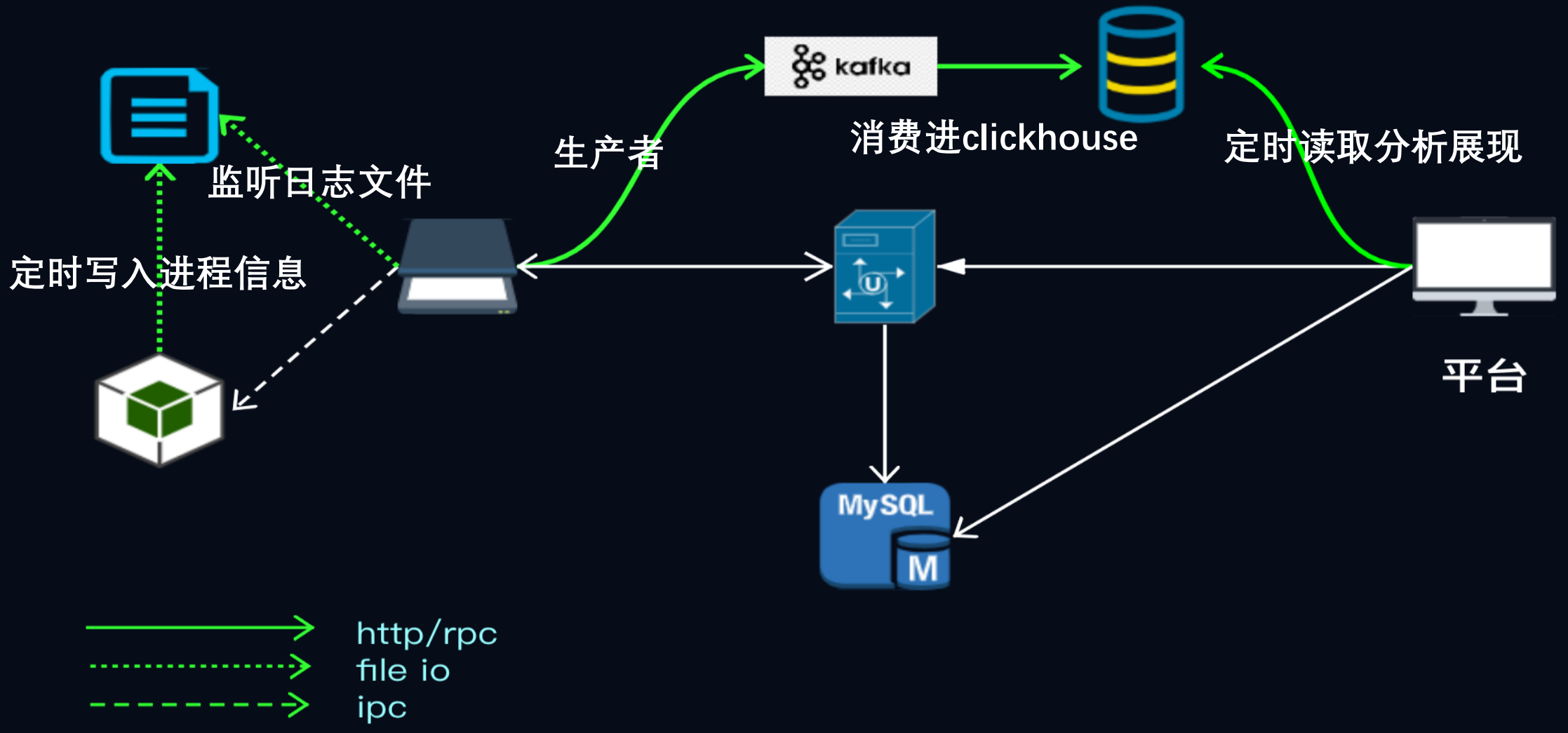
Node.js基础监控能力建设 – 前端



Node.js基础监控能力建设 – 后端



Node.js基础监控能力建设 – 常规日志



Node.js基础监控能力建设 – 常规日志

heap：

JavaScript

```
const v8 = require('v8');
v8.getHeapStatistics();
```

C++

```
v8::HeapStatistics v8_heap_stats;
isolate->GetHeapStatistics(&v8_heap_stats);
```

Node.js基础监控能力建设 – 常规日志

heap space：

JavaScript

```
const v8 = require('v8');
v8.getHeapSpaceStatistics();
```

C++

```
v8::HeapSpaceStatistics v8_heap_spaces;
isolate->GetHeapSpaceStatistics(&v8_heap_spaces, index);
```

Node.js基础监控能力建设 – 常规日志

cpu：

JavaScript

```
process.cpuUsage(prevValue);
```

C++

```
uv_rusage_t rusage;
int result = uv_getrusage(&rusage);

now_cpu_usage = clock();
now_time = uv_hrtime();
```

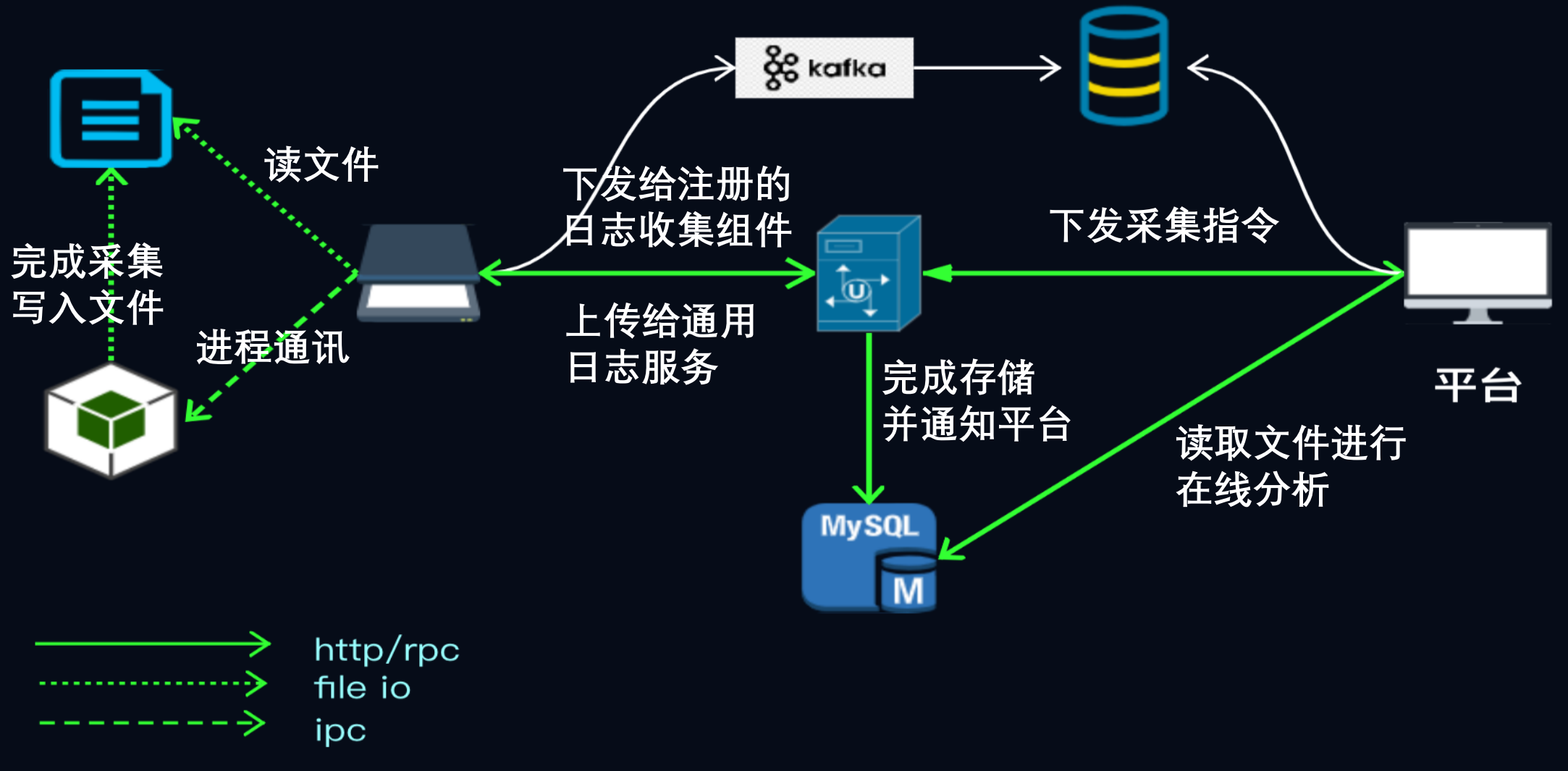
Node.js基础监控能力建设 – 常规日志

gc:

C++

```
void GCPrologueCallback(
  v8::Isolate* isolate,
  v8::GCType type,
  v8::GCCallbackFlags flags);

void GCEpilogueCallback(
  v8::Isolate* isolate,
  v8::GCType type,
  v8::GCCallbackFlags flags);

isolate->AddGCPrologueCallback(GCPrologueCallback);
isolate->AddGCEpilogueCallback(GCEpilogueCallback);
```

Node.js基础监控能力建设 – 按需日志



Node.js基础监控能力建设 – 按需日志

heapdump：

C++

```
v8::HeapSnapshot* snapshot = isolate->GetHeapProfiler()->TakeHeapSnapshot();
ks::helper::FileOutputStream stream(fp);
snapshot->Serialize(&stream, v8::HeapSnapshot::kJSON);
```

Node.js基础监控能力建设 – 按需日志

cpu profile:

C++

```
cpu_profiler = new ks::CPUProfiler();
cpu_profiler->StartProfiling(title, true);
std::string result = cpu_profiler->StopProfiling(title);
```

Node.js基础监控能力建设 – 按需日志

others：

C++

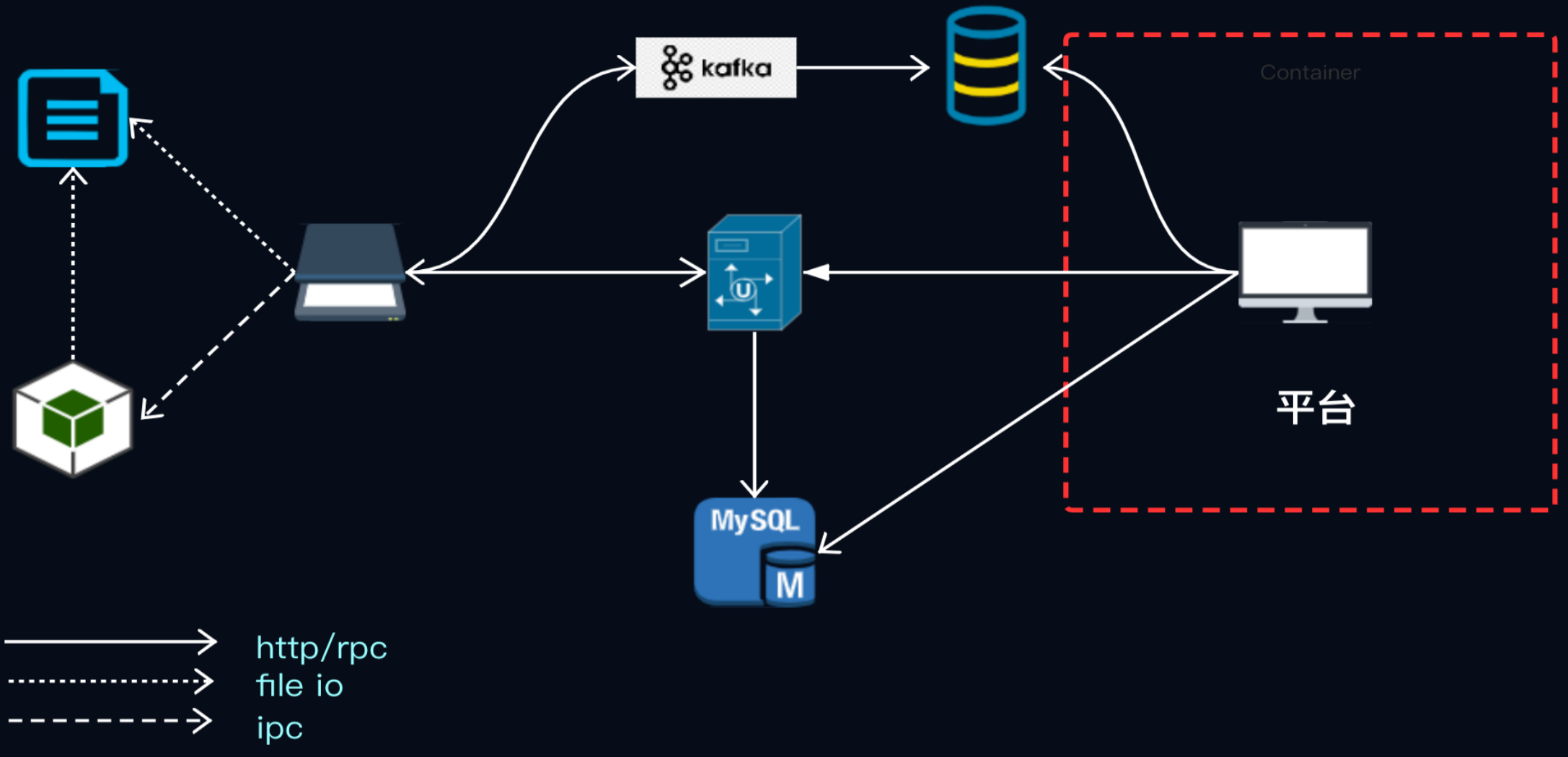
```
heap_profiler->StartSamplingHeapProfiler();
v8::AllocationProfile* profile = heap_profiler->GetAllocationProfile();
heap_profiler->StopSamplingHeapProfiler();

heap_profiler->StartTrackingHeapObjects(true);
heap_profiler->StopTrackingHeapObjects();

isolate->LowMemoryNotification();

isolate->GetHeapCodeAndMetadataStatistics();

isolate->GetHeapObjectStatisticsAtLastGC()
```

Node.js基础监控能力建设

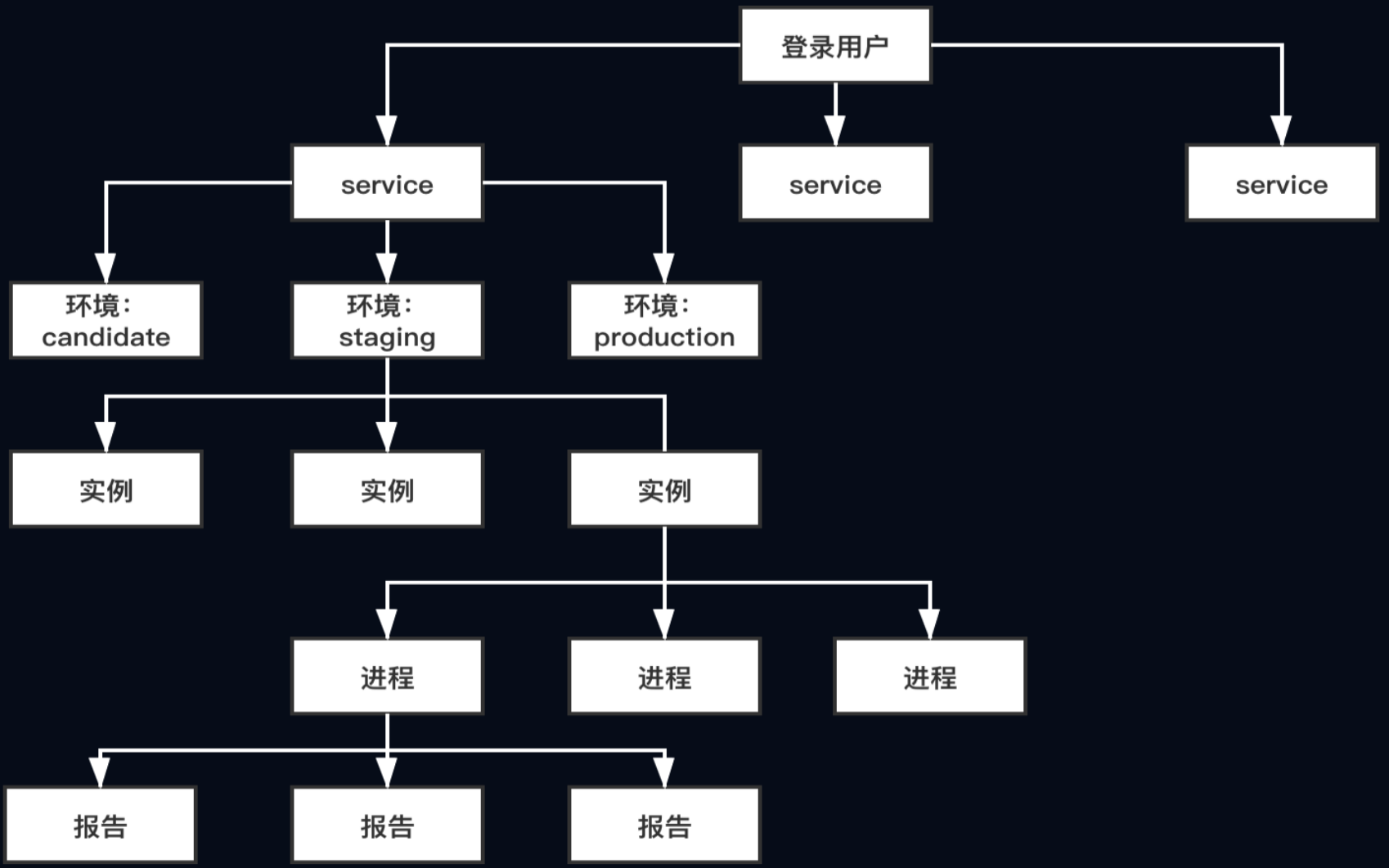
注意事项:

1. 常规日志系统写方法、日志切割
2. 容器环境IPC的限制
3. 多线程处理
4. 其他容错处理

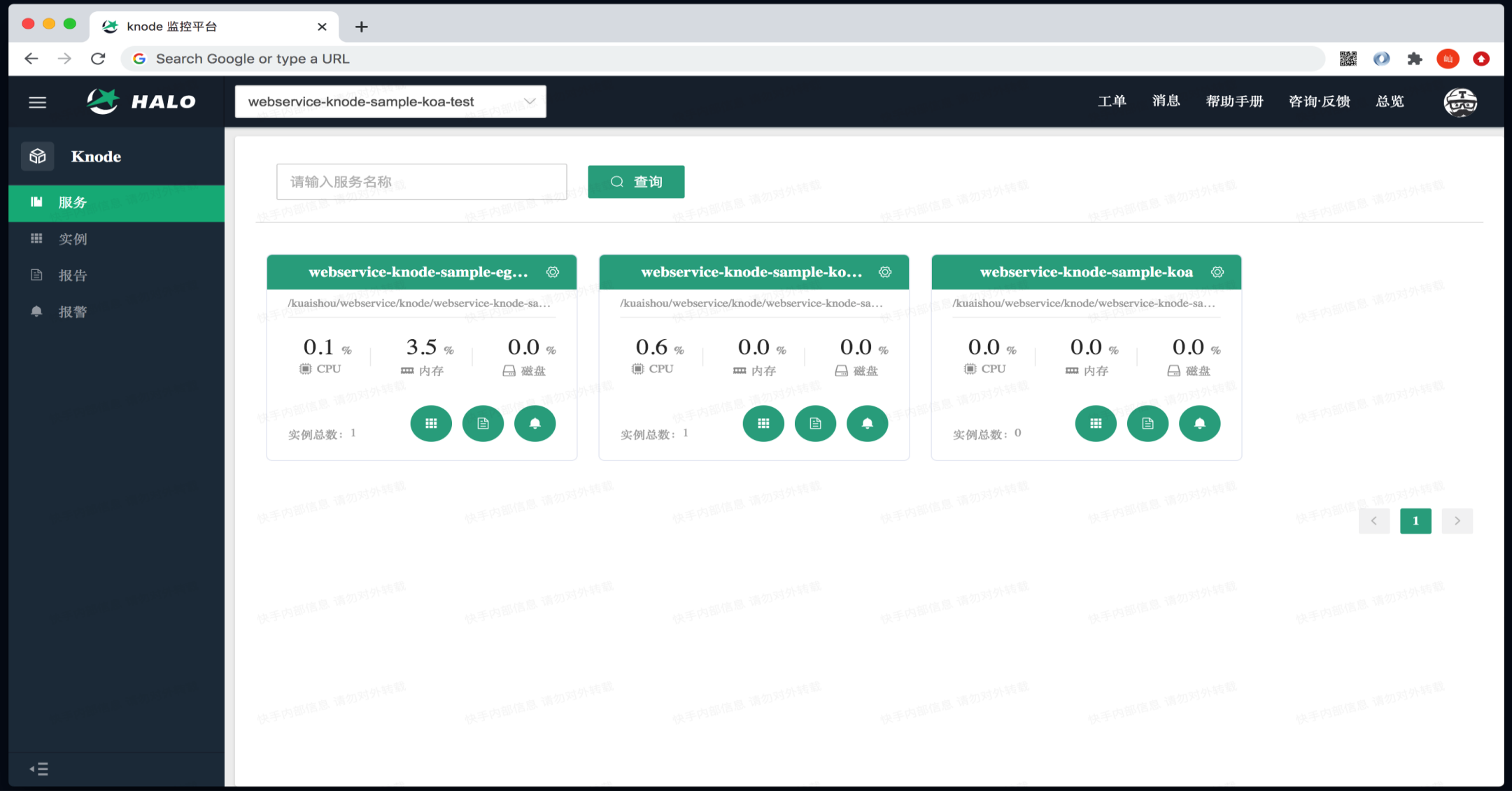
Node.js基础监控能力建设 – 平台



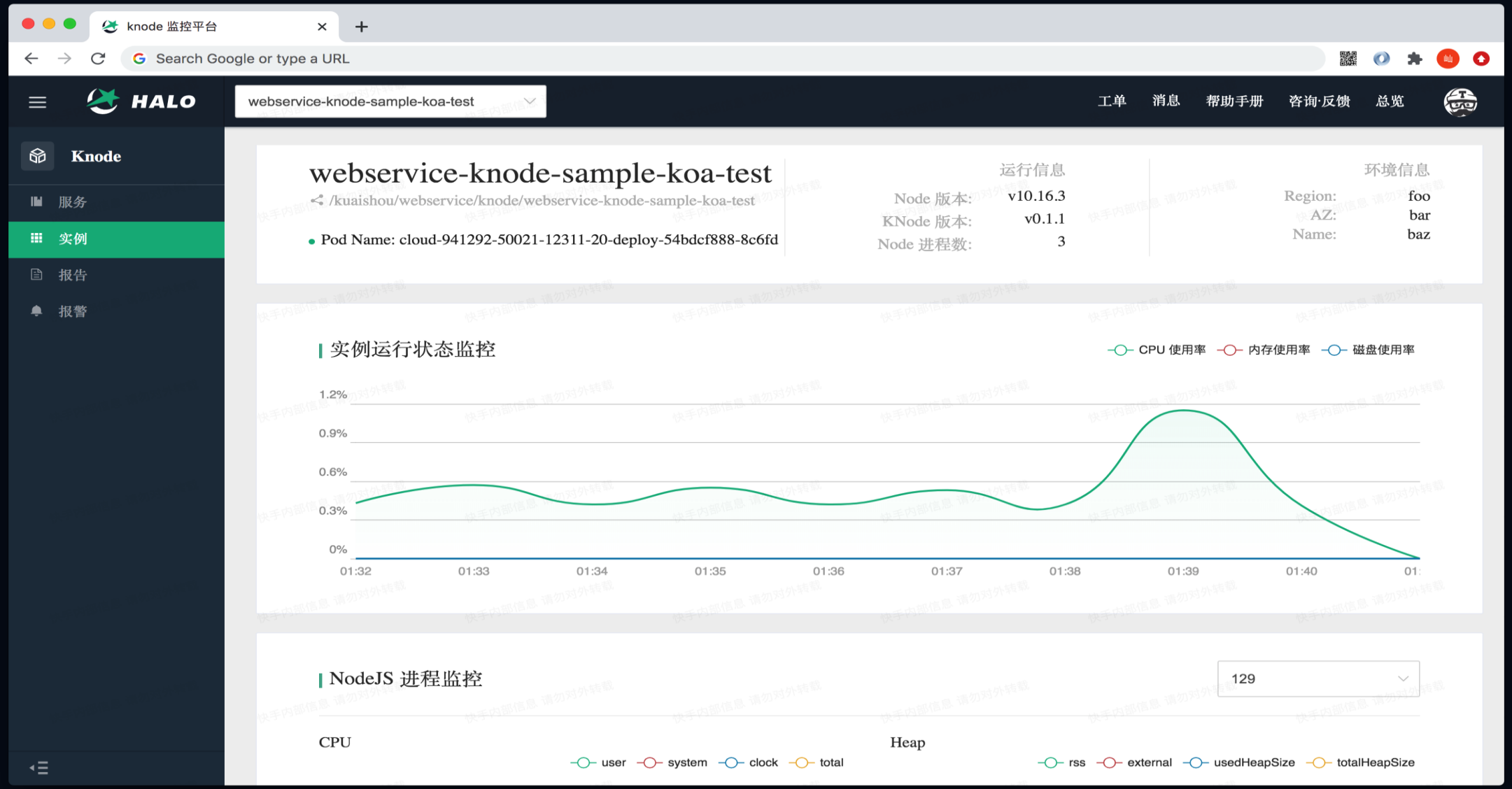
Node.js基础监控能力建设 – 平台



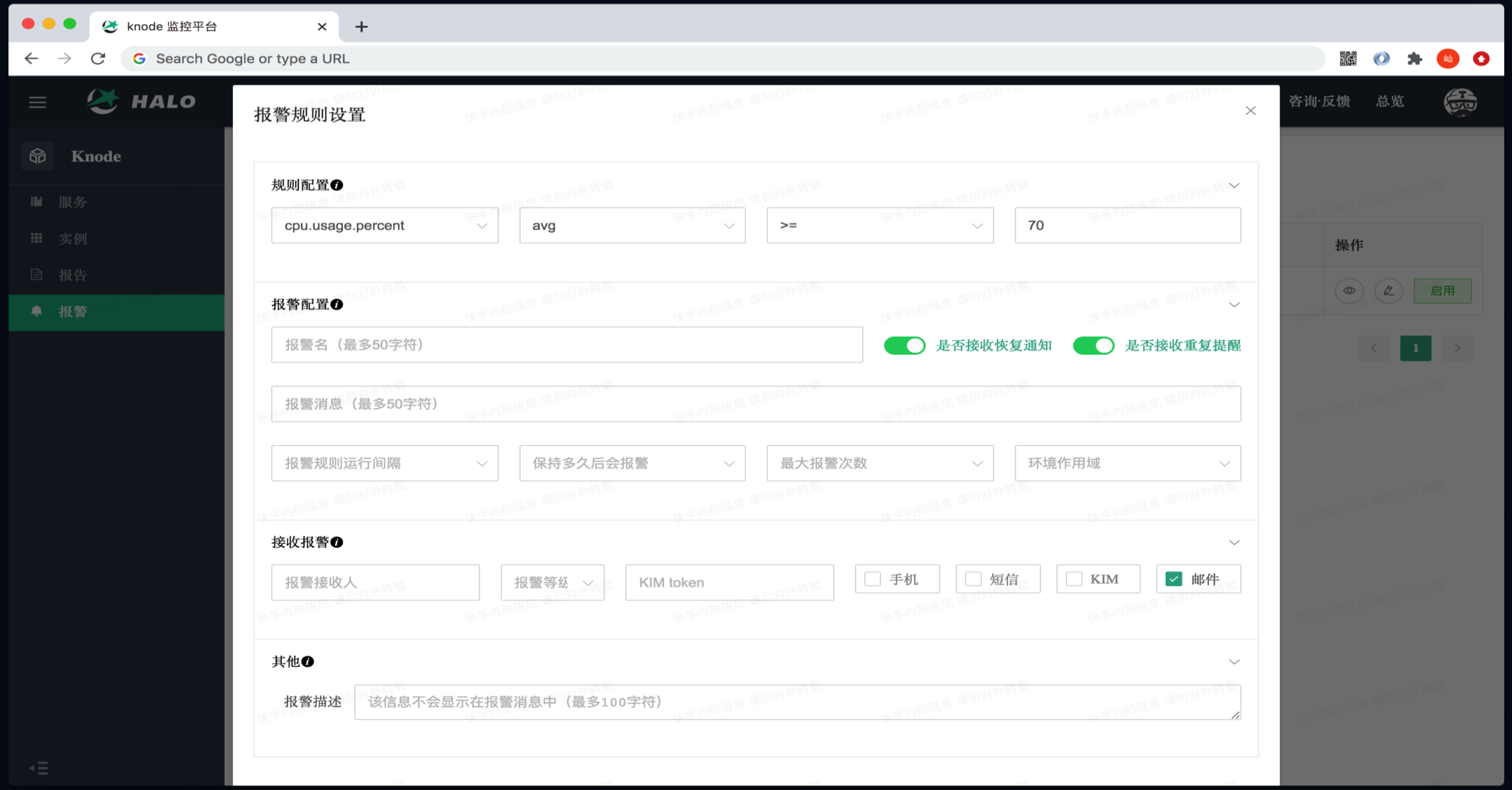
Node.js基础监控能力建设 – 平台



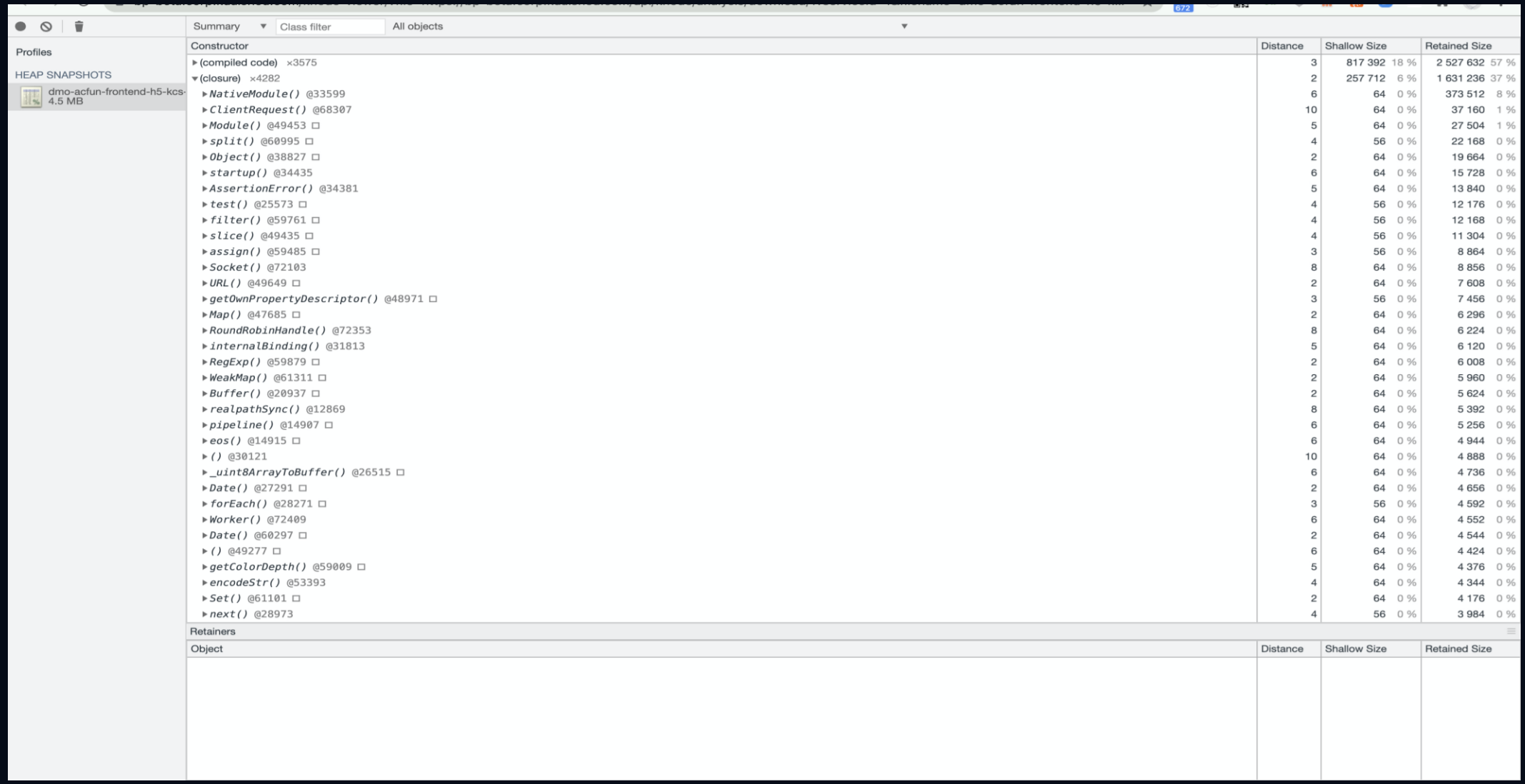
Node.js基础监控能力建设 – 平台



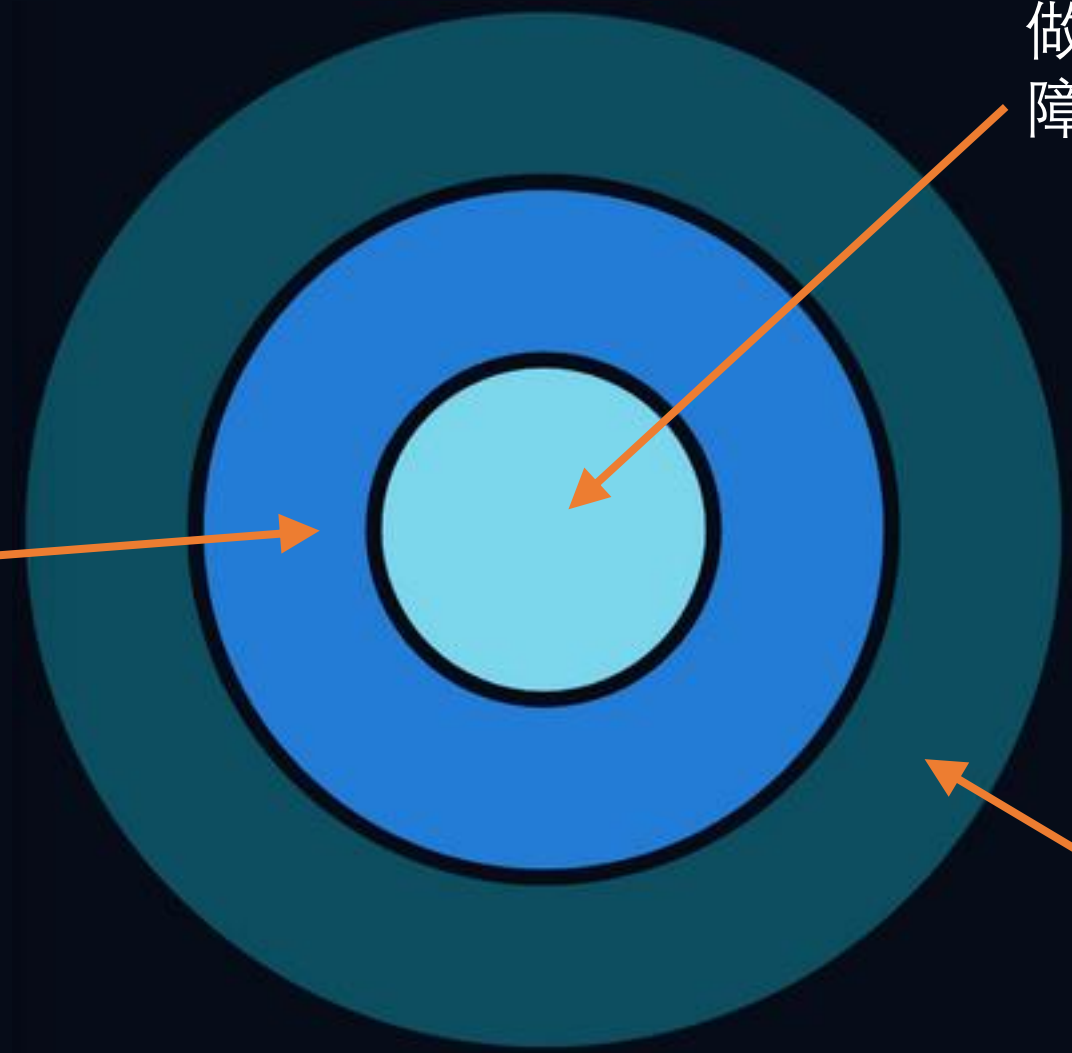
Node.js基础监控能力建设 – 平台



Node.js基础监控能力建设 – 平台



展望未来



做好基础监控、提升排障能力

加速Node.js应用的运行时性能，优化内存管理

一站式Node.js应用解决方案

快手大前端
技术交流会 2020

THANKS