性能优化指北：通用流程和方法

陈熙旻@Rokid

提纲

1. 简介

性能优化二三事

2. 案例

YODAOS 核心语音链路性能优化

3. 流程

性能问题处理模板

4. 方法

性能问题的解决方式

1. 简介

WHAT

- 首屏速度/渲染效率/...
- QPS/响应时间/...
- ...

WHY

- 提升体验
- 成本控制

WHEN

- 持久战

终端：智能音箱

CPU：单核 900M

业务：唤醒

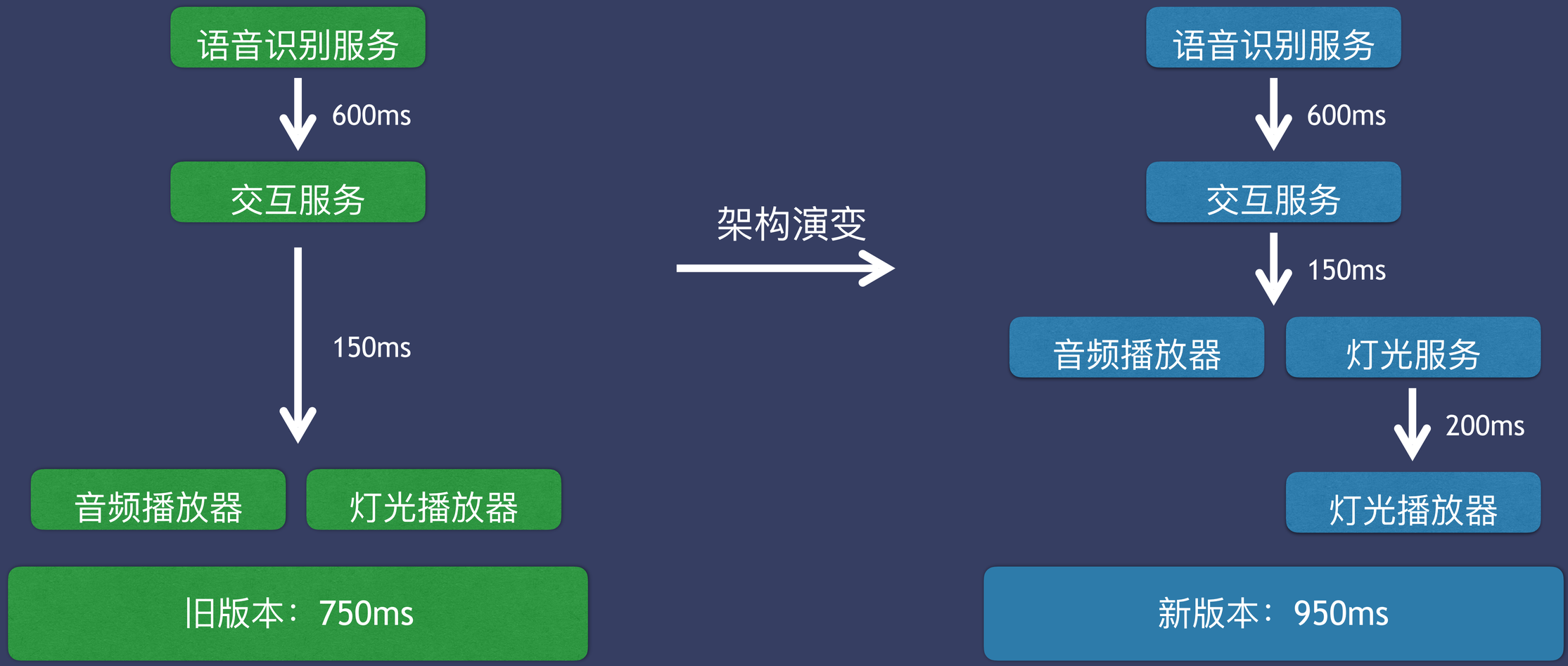
问题

唤醒速度变慢

目标

恢复到原有水平

2.3 案例 - 分解


语音识别服务
600ms
交互服务
150ms
音频播放器
灯光播放器
旧版本：750ms
架构演变
语音识别服务
600ms
交互服务
150ms
音频播放器
灯光服务
200ms
灯光播放器
新版本：950ms

2.4.1 案例 - 排查


文件预加载
优化调用


语音识别服务
600ms
交互服务
150ms
音频播放器
灯光服务
50ms (-150ms)
灯光播放器
新版本：800ms

系统 CPU

40% 波动 80%

交互进程 CPU

0% 波动 25%

GC?

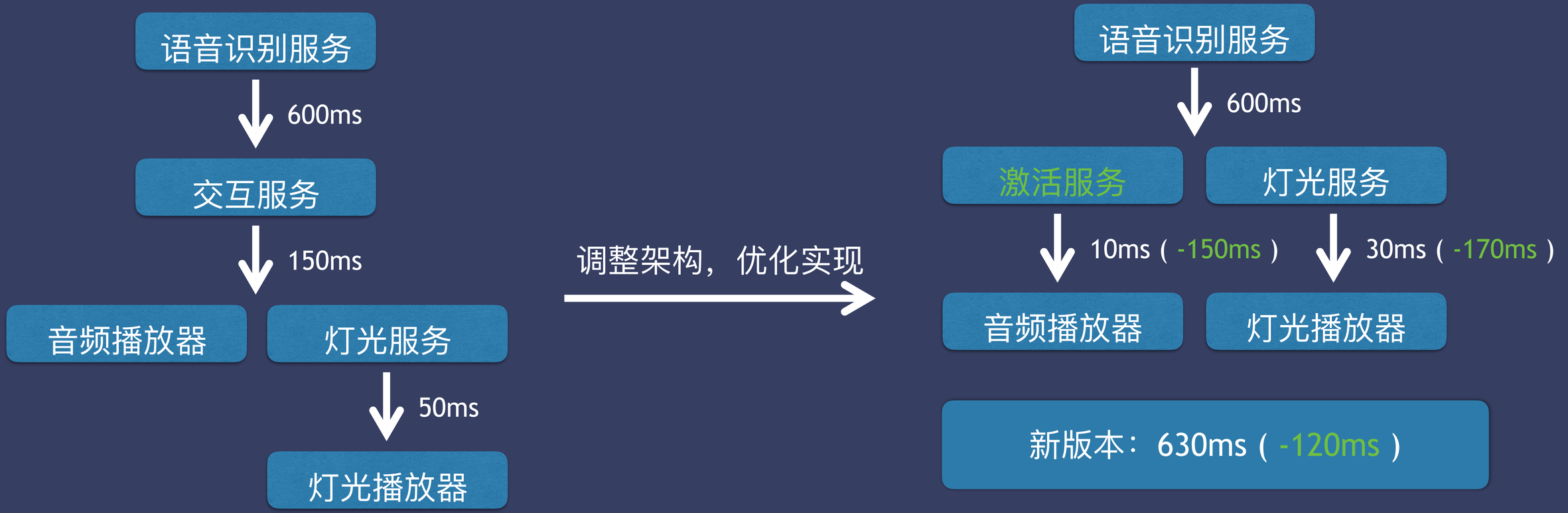
Percent of CPU Usage

```
1 void print_cpu_usage () {
2    sleep(1000)
3    total_cpu = usr + sys + idle + iowait + ...
4    per = (total_cpu - idle - iowait) / total_cpu
5    pcpu = pusr + psys + pidle + piowait + ...
6    pper = ( pcpu / sum(pcpu) ) * per
7    print("total: %f\%, process: %f\%", per, pper)
8 }
```

25% Timeline

Test `$ nice -n -20 node ./app.js`

2.5 案例 - 解决方案


语音识别服务
600ms
交互服务
150ms
音频播放器
灯光服务
50ms
灯光播放器
调整架构，优化实现
语音识别服务
600ms
激活服务
灯光服务
10ms（-150ms）
30ms（-170ms）
音频播放器
灯光播放器
新版本：630ms（-120ms）

分解问题

场景

问题发生在什么场景?

链路

这个场景的链路是什么样的?

3.2 流程 - 明确目标

定义指标

需要关注哪些指标？这些指标需要谁来定义？

预期结果

优化后最终希望达到什么样的目标？

安排周期

是否需要分阶段进行？每个阶段需要完成什么目标？

3.3 流程 - 善用工具

3.5 流程 - 完成

解决

皆大欢喜

规避

查明原因但无法解决，通过某些方式避免

兜底

查明或未查明原因时都无法解决，出现问题时自动订正

搁置

可承受范围内，暂时先不管

评估指标

规划目标

4.2 方法 - 思路

批量

缓存

惰性&预加载

延迟

并发

4.2.1 方法 - 思路 - 缓存

```
1 console.time('cache-test')
2 function handleArray() {}
3 var arr = new Array(100000)
4 for (var i = 0; i < arr.length; ++i) {
5   handleArray(arr[i], function handler() {})
6 }
7 console.timeEnd('cache-test')
```

750ms

```
1 console.time('cache-test')
2 function handleArray() {}
3 var arr = new Array(100000)
4 var length = arr.length
5 var handler = function() {}
6 for (var i = 0; i < length; ++i) {
7   handleArray(arr[i], handler)
8 }
9 console.timeEnd('cache-test')
```

80ms

4.2.1 方法 - 思路 - 延迟

```
function sendStatistic() {
  /*
   * send statistic data
   */
}

function onRequest() {
  /*
   * event logic
   */
  sendStatistic()
}
```

```
function sendStatistic() {
  /*
   * send data
   */
}

function onRequest() {
  /*
   * event logic
   */
  delay(sendStatistic, 1000)
}
```

4.2.1 方法 - 思路 - 批量

```
var sendQueue = []

function sendStatistic() {
  sendQueue.push(...)
  if (sendQueue.length >= MAX_QUEUE_SIZE) {
    /**
     * send data
     */
    sendQueue = []
  }
}

function onRequest() {
  /*
   * event logic
   */
  setImmediate(sendStatistic)
}
```

Preload

```
var data = []
/**
 * init all data
 */

function onRequest(i) {
  return data[i]
}
```

Lazy Load

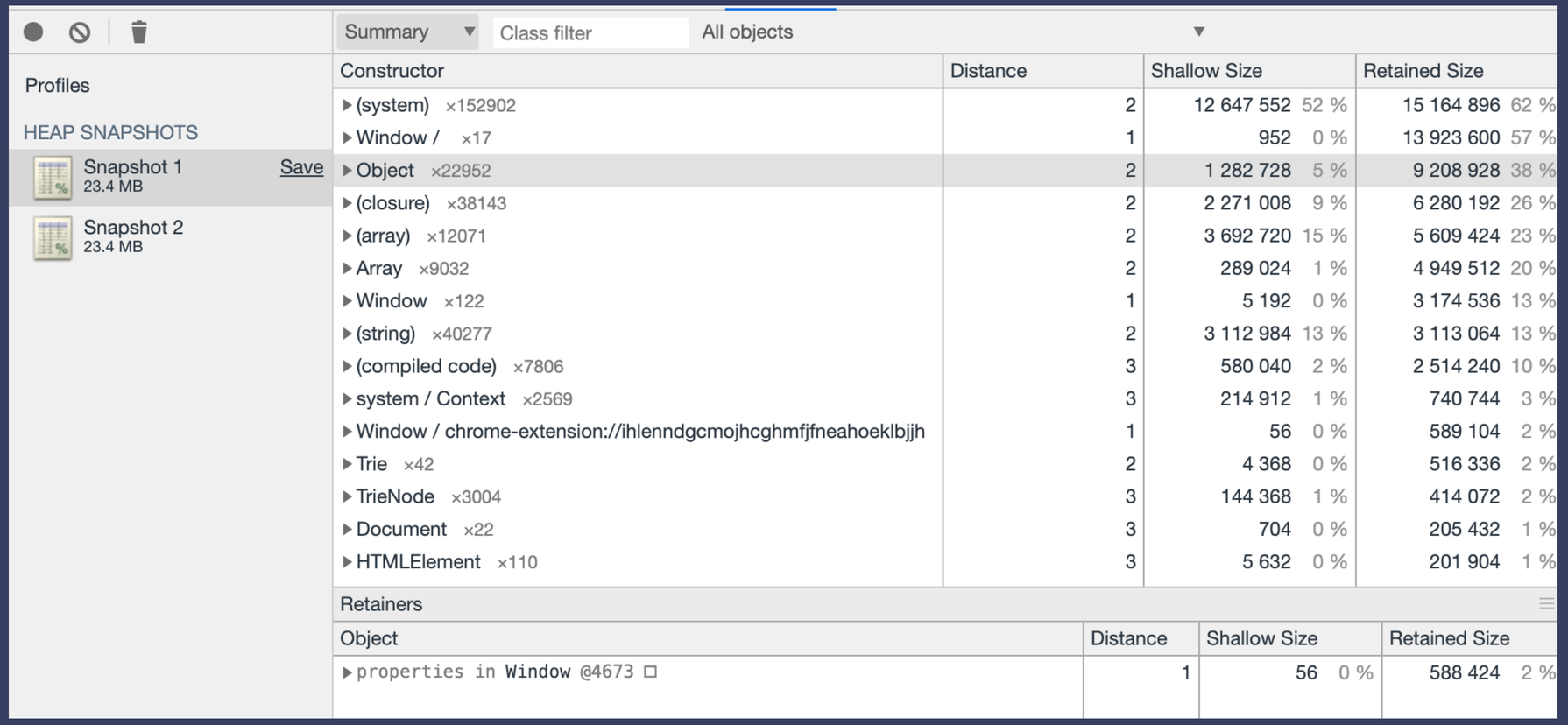
```
var data = []

function onEvent(i) {
  if (!data[i]) {
    /**
     * init data[i]
     */
  }
  return data[i]
}
```

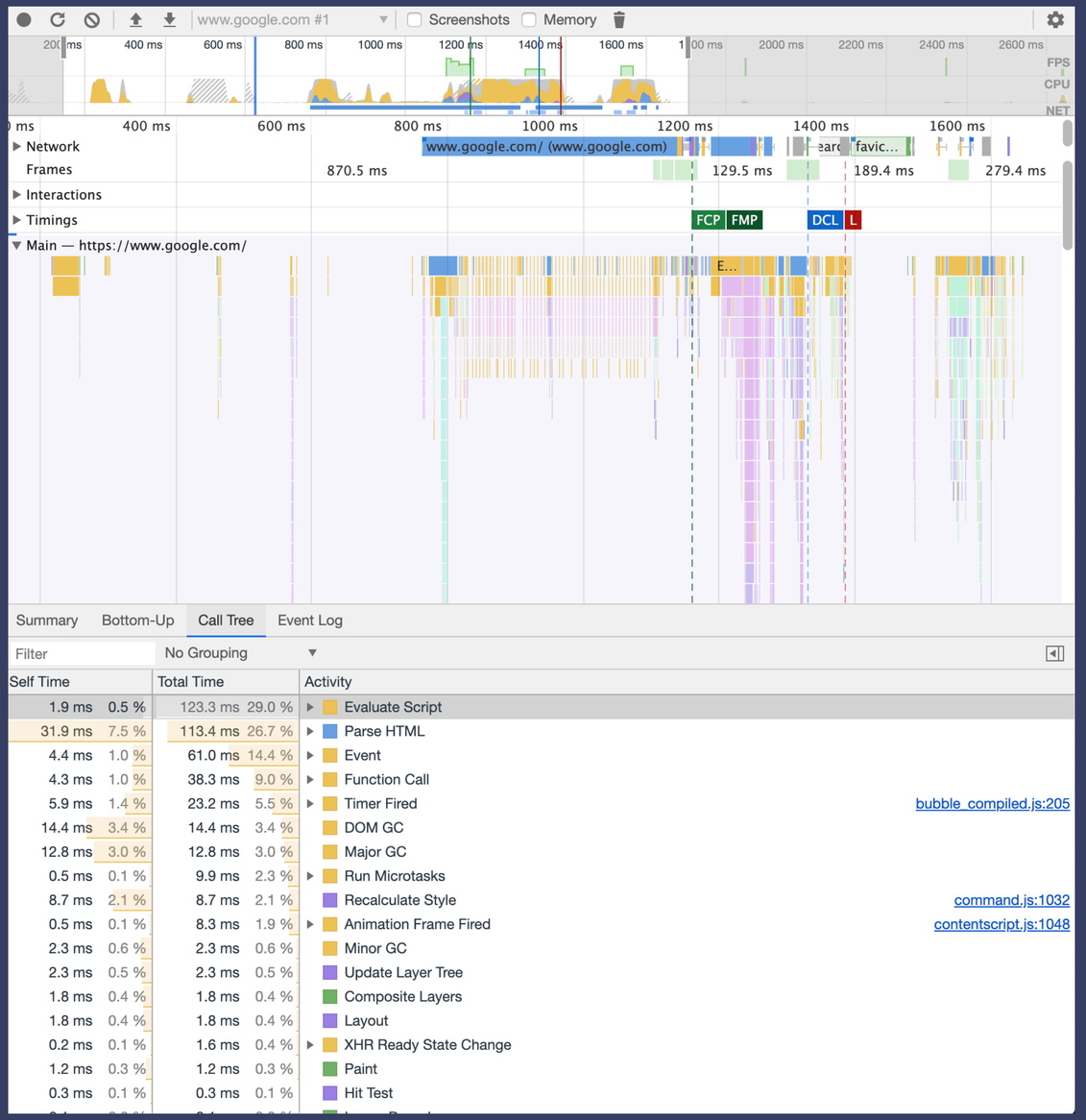
```
/****************** master ******************/
var worker = new Worker('./worker_file', {
  workerData: data
})
worker.on('message', function (message) {
  console.log('worker message: ', message)
})
```

```
/****************** worker ******************/
var result = handleWork(workerData)
masterPort.postMessage(result)
```

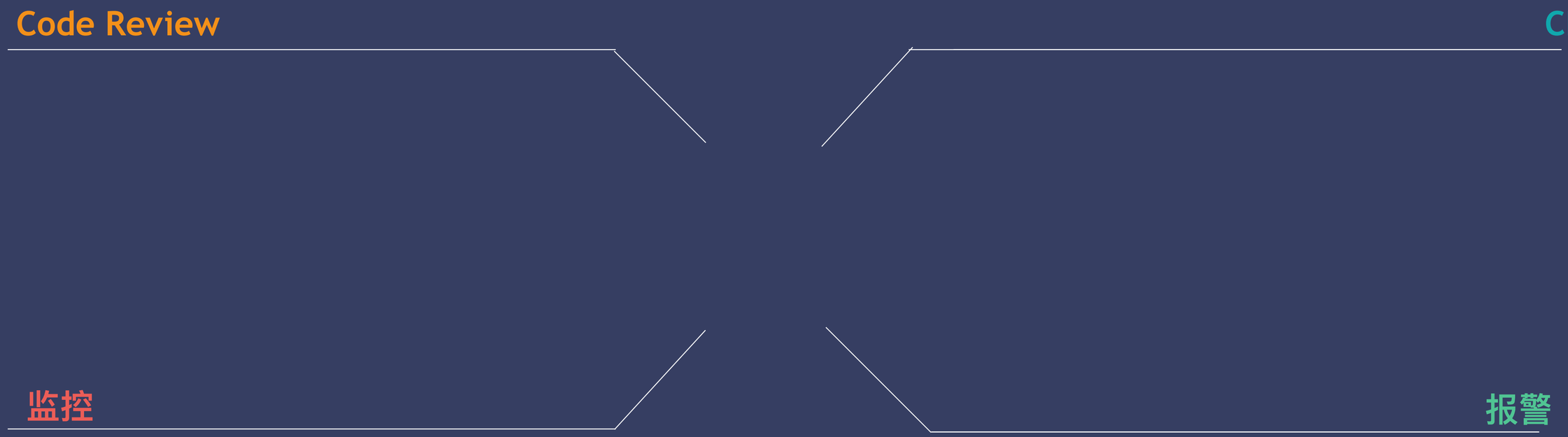
4.3.1 方法 - 工具 - Heap Snapshot



4.3.2 方法 - 工具 - Profiler



4.5 方法 - 持久战



QA