定位线上运行时错误

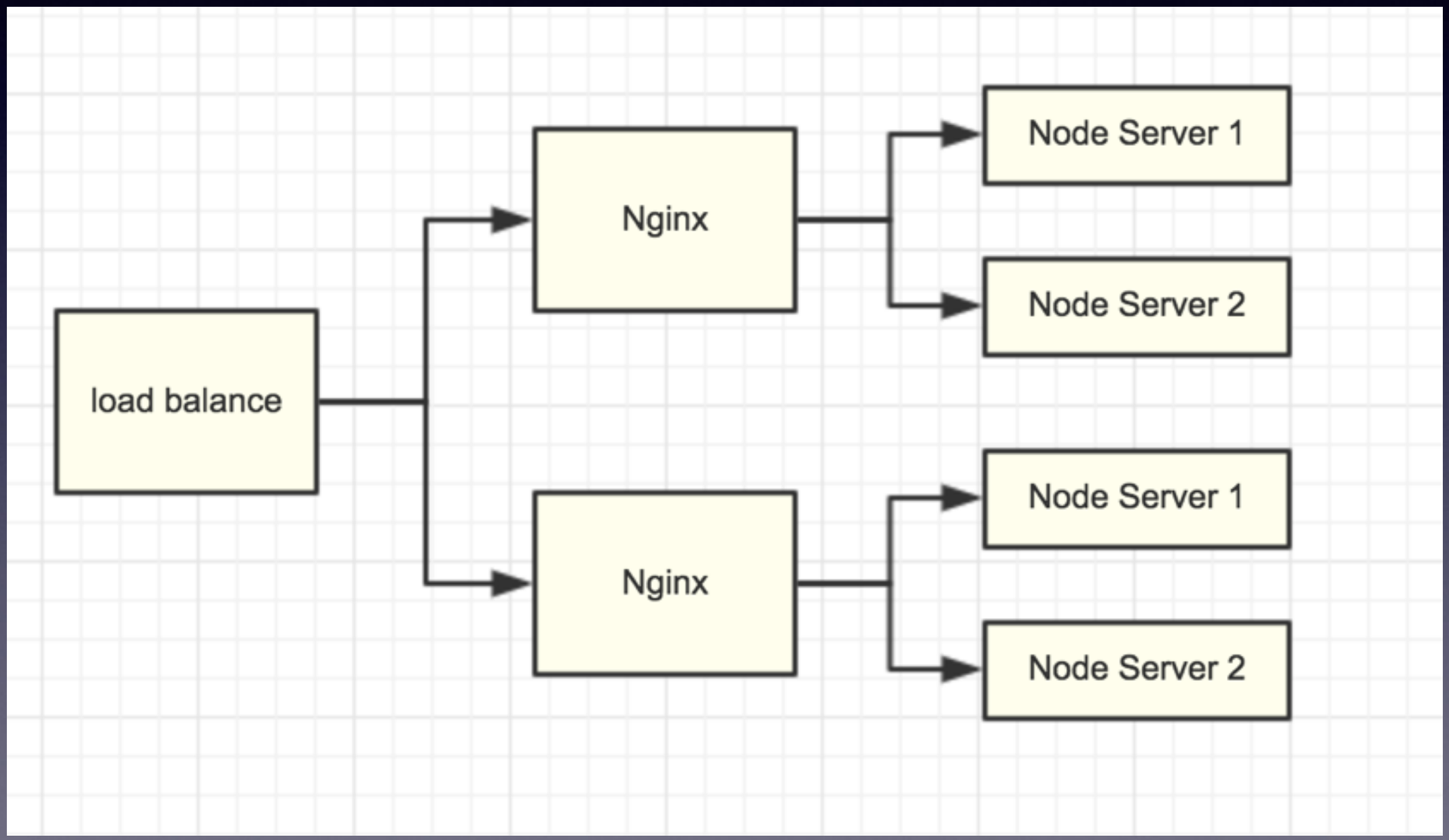
——@途牛旅游网 黄一君

一个生产环境遇到的案例

故障现象

- 开发环境测试通过
- 灰度机测试通过
- 生产环境构建完毕测试通过，但是每次都是间隔半小时到两三小时服务器就没反应了

线上环境


load balance
Nginx
Nginx
Node Server 1
Node Server 2
Node Server 1
Node Server 2

常规排查方式

- 1. Node Server启动时通过组装url测试所有的controller是好的，初步判断框架无问题。

- 线上运行一段时间后出现进程及被阻塞，猜测是某个特定参数组装成的url触发了异常

- Nginx记录了所有的url转发日志集合A，Node Server同样会记录access日志集合B，那么判断引起阻塞的url很可能是A-B中的某一个

```
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 += '                                                ' +
        '                                                ' +
        '       根据船班时间，自行前往暹粒机场，返回中国。<br/>';
    str += '如需送机服务，需增加280/每单。<br/>';

let r = String(str).replace(/(^(\s*?<br[\s\/]*?>\*?)+|(\s*?<br[\s\/]*?>\s*?)+?$)/igm, '');
```

正则的灾难性回溯引起的阻塞

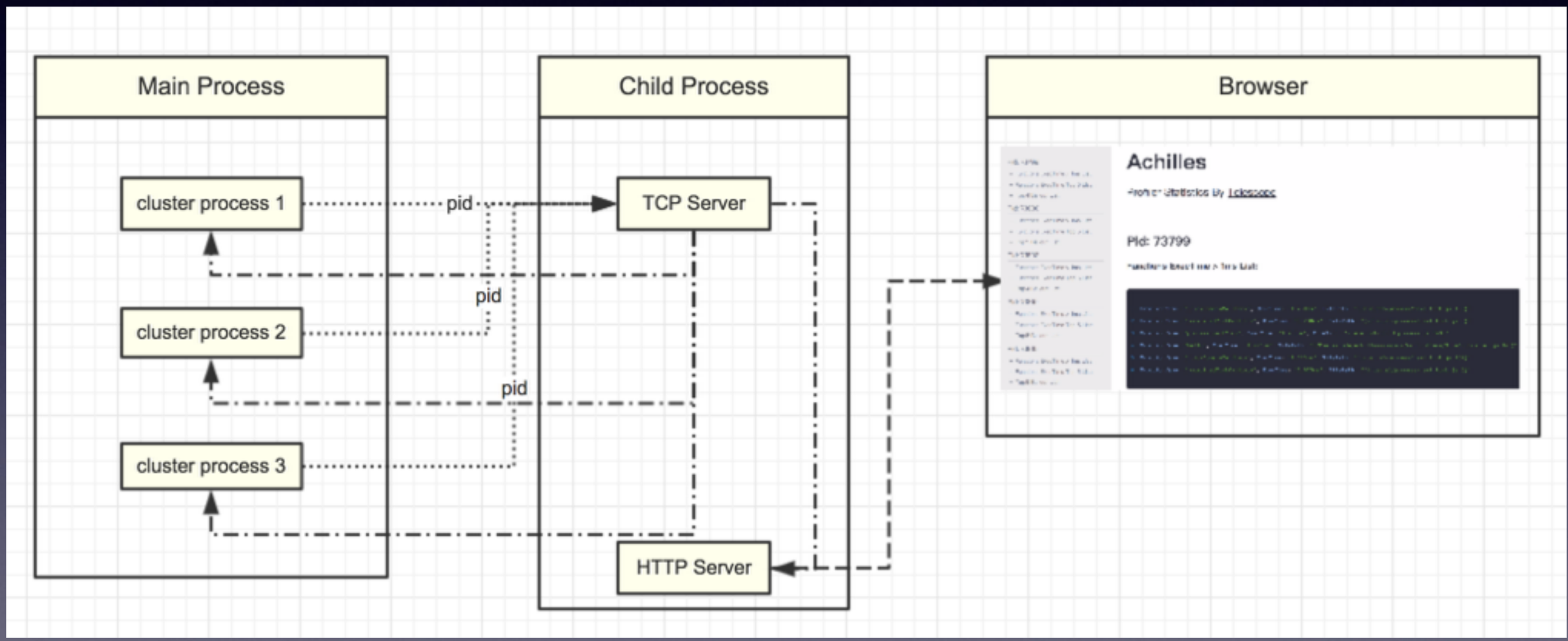
更好的工具

- 线上进程出现阻塞时，能感知到阻塞的函数点。
- 线上进程出现内存泄漏时，能获取到堆内内存的结构引力图。
- JIT中监视器记录为“hot”的热点代码优化程度以及去优化的原因。

能够在我们需要时反馈当前进程的详细状态。

Telescope轻量级监控工具

设计架构


Main Process
cluster process 1
cluster process 2
cluster process 3
pid
pid
pid
Child Process
TCP Server
HTTP Server
Browser
Achilles
Pid: 73799

v8-profiler模块

```
function doCpuProfilingP(options, cb){
    if (doingCpuProfiling)  return;
    doingCpuProfiling = true;
    v8Profiler.startProfiling('easy_monitor', true);
    cTimer.addTimer(() => {
        let profile = v8Profiler.stopProfiling('easy_monitor');
        cb(profile);
        profile.delete();
        doingCpuProfiling = false;
    }, options.CPU_PROFILING_TIME);
}
```

v8-profiler日志解析

日志JSON结构概览

```
{
    "typeId": "CPU",
    "uid": "1",
    "title": "title",
    "head": CpuProfileNode,
    "startTime": 479,
    "endTime": 489,
    "samples": [id1, id2, id3, …],
    "timestamps": [t1, t2, t3, …]
}
```

日志数据详细解析

- startTime和endTime代表本次的Profiler的起始时间戳和结束时间戳，单位是：秒(s)。

- head对应的值为一个对象：可以理解成根节点，是调用 node->`GetTopDownRoot()` 得到的值封装成js对象，并且此对象包含了 `children` 数组，用于指向子节点，后面会详细解析。

- samples的值为一个数组，里面的id其实可以认为是一个key值，该key值为对head根节点开始进行一次前序遍历，每遍历一个节点记录当前的id值得到的，所以这个id对应了当前正在执行的函数栈。

- timestamps的值同样为一个数组，记录的是时间戳，其中每一个时间戳和samples中的id是一一对应的。

head节点


```
{
    "functionName": "(root)",
    "url": "",
    "lineNumber": 0,
    "callUID": 14,
    "bailoutReason": "",
    "id": 1,
    "scriptId": 0,
    "hitCount": 0,
    "children": [child1, child2, ...]
}
```

head节点解析

- functionName：函数名字
- url：函数代码路径
- lineNumber：函数位于代码行数
- callUID：函数入口ID
- bailoutReason：引擎去优化原因
- id：即前面提到的前序遍历得到的key，整棵树唯一
- children：包含子节点的数组，每一个子节点格式和head节点一样

V8-CPU-Analysis

https://www.npmjs.com/package/v8-cpu-analysis

进程阻塞时依旧能运行的扩展模块

- block_timer:基于信号量的定时器，底层使用了signal信号量，js层参考了核心模块中的_linkList.js封装了一个双向循环链表。
- block_socket:建立tcp链接和发包采用了linux下原生的套接字，收包则利用libuv提供的uv_queue_work多线程API，构造了一个while(true)循环进行收包。

以上部分组合起来，就是我们目前试图去还原Node进程运行时状态的轻量级监控工具。

一个测试的例子

```
http.createServer(function (req, res) {
    //测试函数，使用了try catch导致引擎无法优化
    if (req.url === '/tryCatchStatement') {
        return testTryCatch(req, res);}
    //测试函数，while轻微阻塞300ms后返回
    if (req.url === '/sleep') {
        return testSleep(req, res);}
    //测试函数，while无限循环阻塞
    if (req.url === '/long') {
        return testLongLoop(req, res);}
    //测试函数，死亡正则阻塞
    if (req.url === '/regexp') {
        return testRegexp(req, res);}
    res.end('404');
}).listen(8081);
```

首页显示所有的项目列表

TeleScope

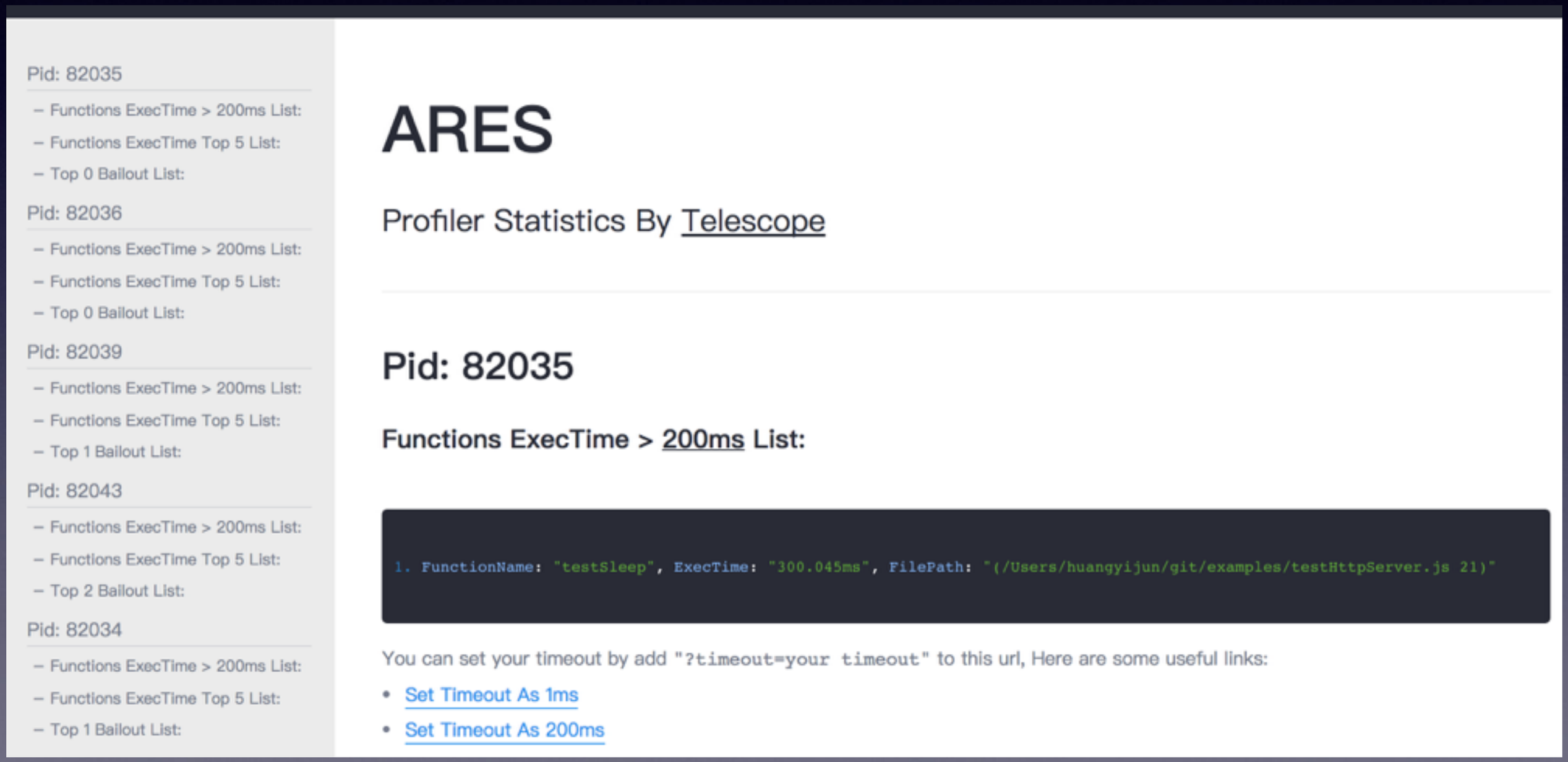
Process Name: ARES

Pid: 82035 Pid: 82036 Pid: 82039 Pid: 82043 Pid: 82034 Pid: 82040 Pid: 82033 Pid: 82042

Process Name: Achilles

Pid: 82067 Pid: 82068 Pid: 82073 Pid: 82078 Pid: 82083 Pid: 82093 Pid: 82088 Pid: 82098

展示出执行时长超过预期的函数列表



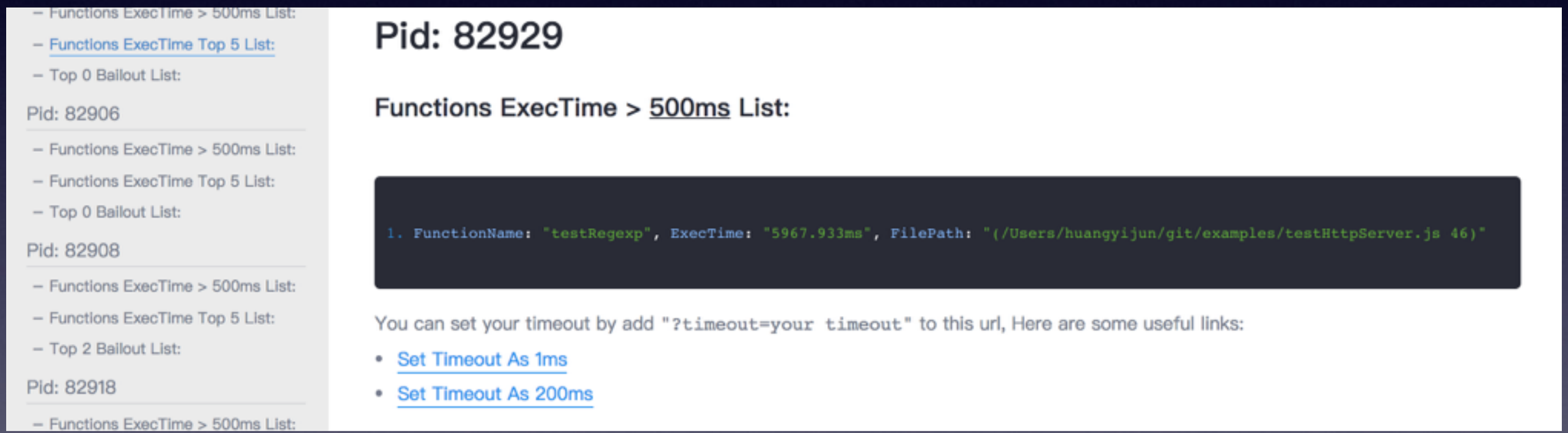
展示运行时V8引擎优化失败的函数

Top 2 Bailout List:

```
1. FunctionName: "testTryCatch", BailoutReason: "TryCatchStatement", HitTimes:1, FilePath: "(/Users/huangyijun/git/examples/
2. FunctionName: "ipcSend", BailoutReason: "TryCatchStatement", HitTimes:1, FilePath: "(/usr/local/lib/node_modules/pm2/node
```

测试while无限循环造成的阻塞

测试死亡正则造成的阻塞



尚待解决的问题

- v8-profiler依赖于isolate->GetCpuProfiler，v8引擎提供的此api在长阻塞模式下不稳定。Node版本的问题，必须小于V7。
- 利用信号量构建的阻塞时依旧能运行的定时器，对于主进程的影响如何暂未理清。
- 编写的c++扩展稳定性待进一步验证。

更多的可以做的：

MemProfiler

题外话:apm监控原理

——运行时侵入库函数

http.createrServer例子

```
wrapMethod(http.Server.prototype, 'http.Server.prototype',
    ['on', 'addListener'], function cb_wrapMethod(addListener) {
        return function cls_wrapMethod(type, listener) {
            if (type === 'request' && typeof listener === 'function') {
                return addListener.call(this, type, wrapListener(agent, listener));
            }
            else {
                return addListener.apply(this, arguments);
            }
        };
});
```

```
//侵入的公共方法
wrapMethod(nodule, noduleName, methods, wrapper) {
        methods = Array.isArray(methods) && methods || [methods];
        methods.forEach(item => {
            nodule[item] = wrapper(nodule[item]);
        });
}
```

```
//http服务器请求处理句柄侵入，收集相关信息
function wrapListener(agent, listener) {
    return function innerListener(request, response) {
        listener(request, response);
        let ctrObj = {request, response};
        let collector = new agent.Colle(agent.config, ctrObj);
        response.once('finish', function finishRequest() {
            collector.requestEnd(agent);
        });
}}
```

Q&A

Thank You

END