S2306：Node.js 性能优化

https://rvspoc.org/s2306/

RVSPOC 组委会 – Ryan
2024 年 01 月 12 日

题目解析 S2306

项目描述：

在 Node.js 官方 RISC-V 版本的基础上，修复 bug 提升稳定性的同时，基于 SG2042/Pioneer Box 硬件实施优化，尽可能高的提升相关 Benchmark 的分数。

产出及评分要求：

1. 以 Node 源码中的 benchmark 测试集作为本次比赛测试、分析和优化的标准。
2. 测试并整理 Node.js v20.10.0 上 benchmark 测试结果，并与至少一种主流平台 (ARM64/X64) 上相同 Node 版本 benchmark 测试结果进行对比和分析。
3. 通过运行 node-benchmark-compare 工具来展示优化效果。
4. 最终将综合参赛者所提交的 (1) 测试分析报告；(2) 取得优化效果的项目数量；(3) 单项优化幅度；(4) 综合优化幅度来评价胜负。

验证平台：SG2042

提交仓库

- https://github.com/plctlab/rvspoc-s2306-node

知识产权及开源协议说明：

所有参赛结果要求开源，并提交至主办方指定仓库。参赛者（作者）持有作品的所有权。主办方鼓励参赛者将结果回馈贡献至 upstream。

有关 Node.js nodejs.org

- 开源的跨架构 JavaScript 运行环境
- 运行在 V8 JavaScript 引擎上
- 使用 JavaScript 以及 C++ 语言写成
- 采用能够异步 I/O 的事件驱动型架构（EDA）
- 让通过 JavaScript 来创建 Web 服务和网络相关的工具变得更加简单
- LTS 版本： 18.x 以及 **20.x（本次比赛声明版本）**
- 最新版本： 21.x

有关 Node.js

——依赖

- 引擎：V8
- 处理异步事件：libuv
- HTTP 支持：nghttp2
- 符合 WHATWG 规范的 URL 解析器：Ada
- Unicode 验证及转码：simdutf （19.5 版本以后）
- JSON 解析：simdjson （21.3 版本以后**）
- 其余见源码的 /deps 文件夹

有关 Node.js

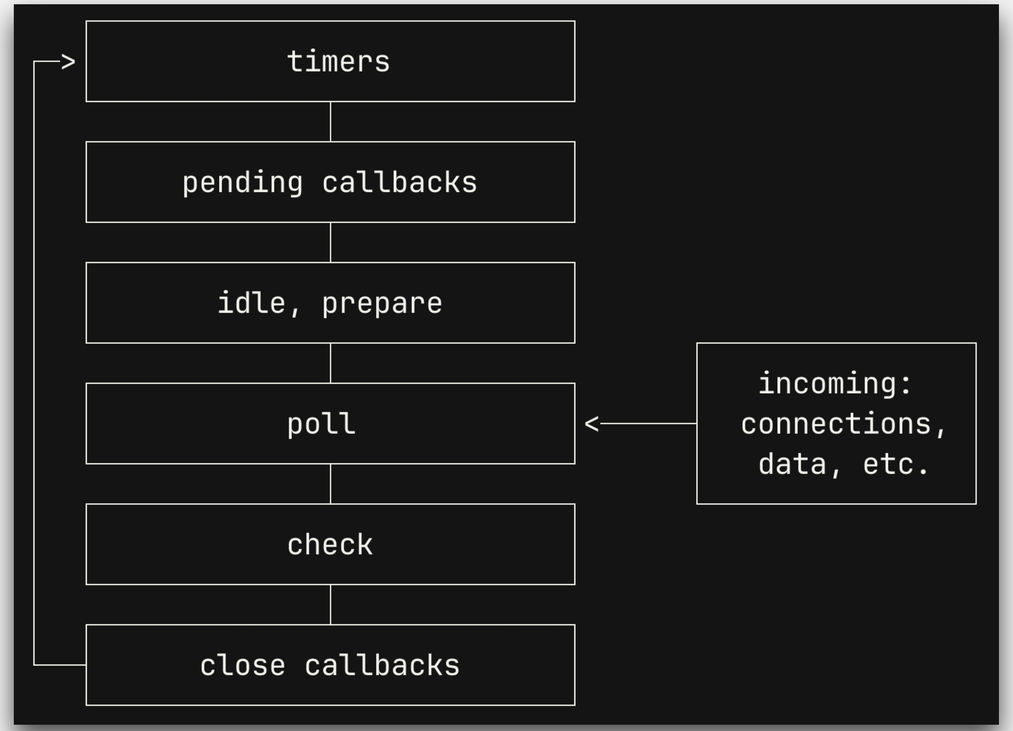
——源码目录结构

- deps/：依赖，包括 V8、libuv、nghttp2、Ada 等等
- doc/：包含文档，各组件 API 调用说明等
- lib/：包含核心的 Node.js 模组 (child_process.js, cluster.js, crypto.js, dns.js, events.js, fs.js, http.js, https.js, net.js, os.js, path.js, process.js, querystring.js, readline.js, stream.js, string_decoder.js, timers.js, tls.js, tty.js, url.js, util.js, vm.js, zlib.js…)
- src/：支撑 Node.js 的 C++ 代码，包含了将 Node.js 与 V8/libuv 连接起来的代码
- test/：包含了单元测试代码
- tools/：包含了构建和开发工具

有关 Node.js

——Event Loop 概览

- https://nodejs.org/en/guides/event-loop-timers-and-nexttick
- 达成能够执行非阻塞的 I/O 操作
  (尽管 JS 是单线程的)
- 什么是单线程，非阻塞
- 很多性能问题归结于此



每个阶段的回调都以 FIFO 原则执行

有关 Node.js

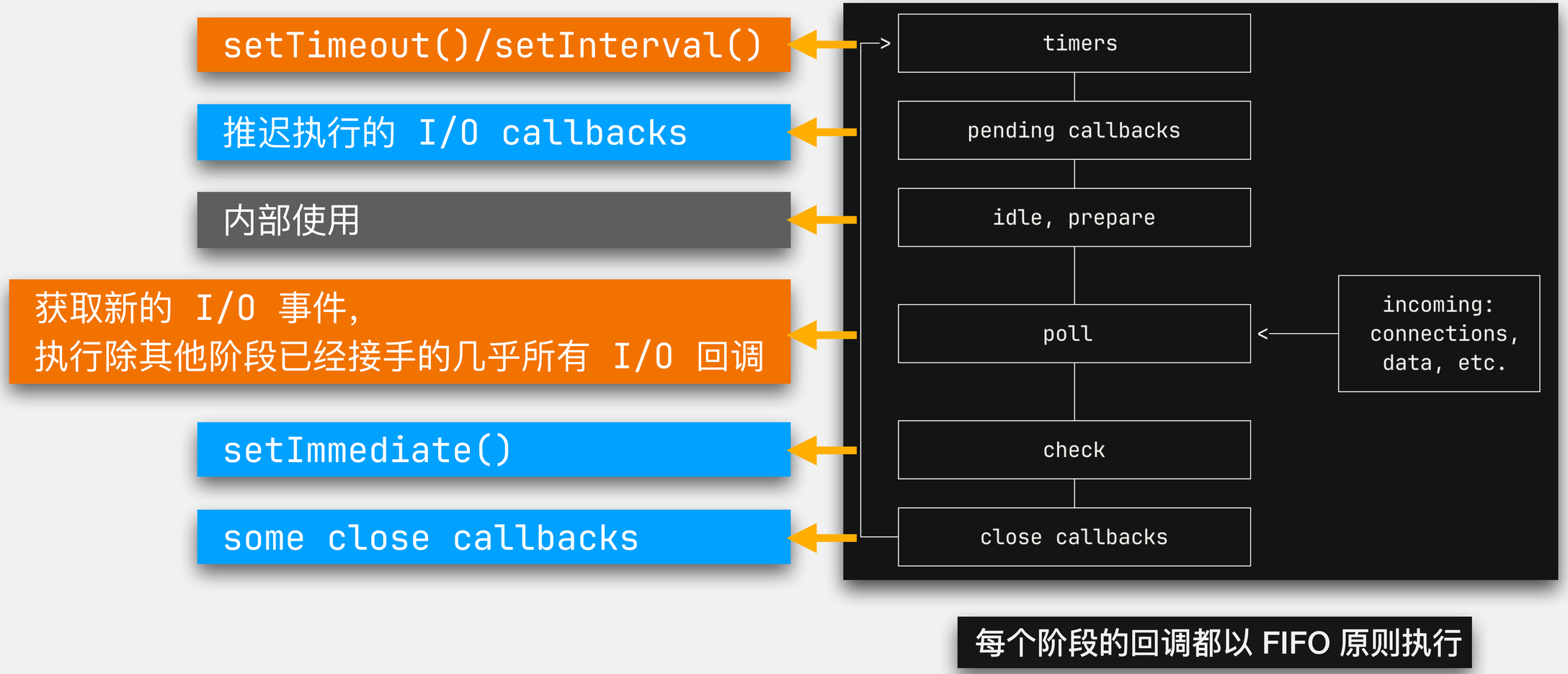
——Event Loop 概览

Event Loop Explained

When Node.js starts, it initializes the event loop, processes the provided input script (or drops into the REPL, which is not covered in this document) which may make async API calls, schedule timers, or call `process.nextTick()`, then begins processing the event loop.

有关 Node.js

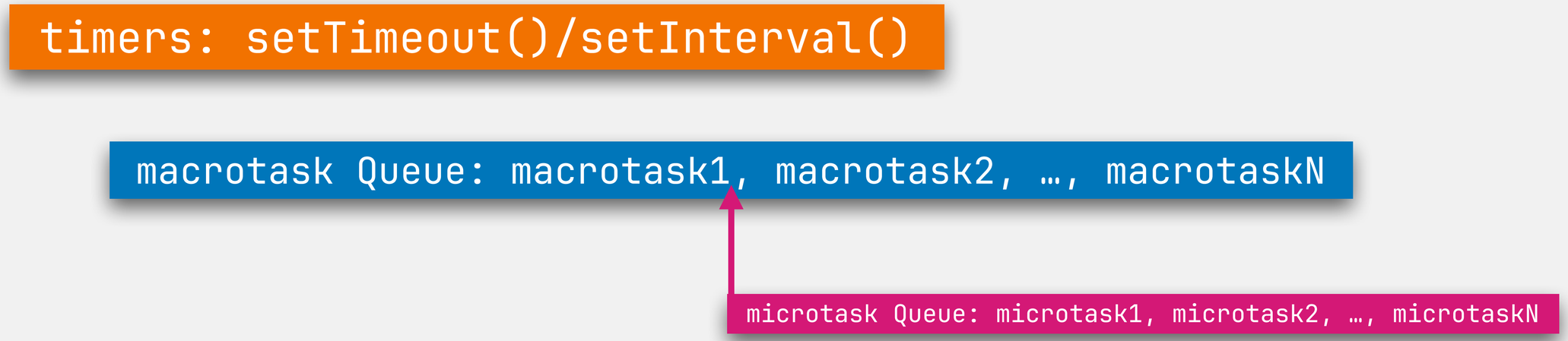
——Event Loop 概览



每个阶段的回调都以 FIFO 原则执行

有关 Node.js

——Event Loop 概览



有关 Node.js

——Event Loop 示例

```
/tmp/xT/tmp.8xdgOddNO5 ➤ bat el3.js
File: el3.js
1   console.log("1")
2   setTimeout(() ⇒ {
3       console.log("2 timeout 0");
4   },0)
5   process.nextTick(() ⇒ {
6       console.log("3 nextTick")
7   });
8   console.log("4")
.025460

/tmp/xT/tmp.8xdgOddNO5 ➤ node el3.js
1
4
3 nextTick
2 timeout 0
.044855 +.010
```

```
/tmp/xT/tmp.8xdgOddNO5 ➤ bat el4.js
File: el4.js
1   console.log("1")
2   setTimeout(() ⇒ {
3       console.log("2 timeout 0");
4       process.nextTick(() ⇒ {
5           console.log("3 nextTick")
6       });
7   },0)
8   process.nextTick(() ⇒ {
9       console.log("4 nextTick")
10  });
11  console.log("5")
.023723 +.011

/tmp/xT/tmp.8xdgOddNO5 ➤ node el4.js
1
5
4 nextTick
2 timeout 0
3 nextTick
.048464
```

```
/tmp/xT/tmp.8xdgOddNO5 ➤ bat el5.js
File: el5.js
1   const fs = require("fs");
2
3   console.log("1");
4   setTimeout(() ⇒ {
5       console.log("2 timeout 0");
6   },0)
7   fs.readFile("./file", () ⇒ {
8       console.log("3 readFile");
9   })
10  console.log("4");
.025243

/tmp/xT/tmp.8xdgOddNO5 ➤ node el5.js
1
4
2 timeout 0
3 readFile
.051439 +.010
```

有关 Node.js

——Event Loop 实现概览

- V8 JavaScript 引擎：
  - 用于执行 JS 代码
  - 内存管理
  - 函数调用
  - 以及其他同步操作
- libuv:
  - 跨平台的 C 语言写的库：
    - 异步的 I/O 操作（文件系统，网络相关等）
    - 给像 DNS 查询这种任务用的线程池，（UV_THREADPOOL_SIZE）
    - 直接与操作系统的内核进行通讯，以提供更高效的 I/O 操作

Node.js 的编译

- https://github.com/nodejs/node/blob/main/BUILDING.md
- 没有麻烦的地方
- 依赖 gcc 和 g++ 命令（版本 >= 10.1）
- 依赖 make 命令
- 依赖 Python3 （>= 3.6）
- 可选依赖 ninja

Node.js 的 Benchmark

```
~/Git/node ➤ cat ~/cc.txt| Rscript ./benchmark/compare.R                                                    tags/v20.11.0
                                                                                confidence improvement accuracy (*)   (**)  (***)
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=0 strict=0 len=100 n=20000             ***      7.89 %       ±1.30% ±1.73% ±2.25%
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=0 strict=0 len=1000 n=20000            ***      6.18 %       ±1.33% ±1.77% ±2.30%
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=0 strict=1 len=100 n=20000             ***      7.60 %       ±0.81% ±1.08% ±1.41%
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=0 strict=1 len=1000 n=20000            ***      5.52 %       ±1.01% ±1.35% ±1.75%
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=1 strict=0 len=100 n=20000             ***     12.03 %       ±1.64% ±2.20% ±2.90%
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=1 strict=0 len=1000 n=20000            ***      9.60 %       ±1.46% ±1.96% ±2.56%
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=1 strict=1 len=100 n=20000             ***      8.58 %       ±1.05% ±1.40% ±1.83%
assert/deepequal-buffer.js method='deepEqual' arrayBuffer=1 strict=1 len=1000 n=20000            ***      9.45 %       ±1.72% ±2.31% ±3.04%
assert/deepequal-buffer.js method='notDeepEqual' arrayBuffer=0 strict=1 len=100 n=20000          ***      6.42 %       ±0.75% ±1.00% ±1.31%
assert/deepequal-buffer.js method='notDeepEqual' arrayBuffer=0 strict=1 len=1000 n=20000         ***      5.69 %       ±1.25% ±1.67% ±2.19%
assert/deepequal-buffer.js method='notDeepEqual' arrayBuffer=1 strict=1 len=100 n=20000          ***     11.29 %       ±1.16% ±1.55% ±2.04%
assert/deepequal-buffer.js method='notDeepEqual' arrayBuffer=1 strict=1 len=1000 n=20000         ***      9.25 %       ±1.32% ±1.76% ±2.29%
assert/deepequal-buffer.js method='unequal_length' arrayBuffer=0 strict=1 len=100 n=20000        ***      5.16 %       ±0.95% ±1.27% ±1.65%
assert/deepequal-buffer.js method='unequal_length' arrayBuffer=0 strict=1 len=1000 n=20000        **      3.33 %       ±1.93% ±2.58% ±3.38%
assert/deepequal-buffer.js method='unequal_length' arrayBuffer=1 strict=1 len=100 n=20000         ***     13.36 %       ±1.34% ±1.80% ±2.38%
assert/deepequal-buffer.js method='unequal_length' arrayBuffer=1 strict=1 len=1000 n=20000        ***     10.93 %       ±2.03% ±2.71% ±3.55%

Be aware that when doing many comparisons the risk of a false-positive
result increases. In this case, there are 16 comparisons, you can thus
expect the following amount of false-positive results:
  0.80 false positives, when considering a   5% risk acceptance (*, **, ***),
  0.16 false positives, when considering a   1% risk acceptance (**, ***),
  0.02 false positives, when considering a 0.1% risk acceptance (***)
  .427986 +.044
```

Node.js 的 Benchmark

——node-benchmark-compare 的安装

- https://github.com/targos/node-benchmark-compare
- README.md 已经说明了如何全局安装
- 如何不进行全局安装也可以使用?
  - `git clone https://github.com/targos/node-benchmark-compare /path/to/dir`
  - `cd /path/to/dir`
  - `npm install`
  - `./bin/node-benchmark-compare /path/to/result.txt`

Node.js 的 Benchmark

——如何使用 compare.R 以达到和 node-benchmark-compare 一样的效果

- Node.js 自带了一个用于比较由 compare.js 所生成结果的 R 语言脚本
- 经过测试，compare.R 与 node-benchmark-compare 的效果一致
- 如何使用这个脚本，在它的源码里面已经有所体现：
  - `cat /path/to/result.txt | Rscript ./compare.R`
- 但安装好 R 环境之后，还需要额外安装两个库，ggplot2 & plyr，方法也很简单：
  - `R #运行进入 R 交互系统`
  - `> install.packages(“ggplot2”) #运行后会提示选择一个镜像站用于下载，比如选择 15 号北京站 2`
  - `> install.packages(“plyr”)`

感谢观看

RVSPOC 组委会 – Ryan
2024 年 01 月 12 日