Javascript 异步编程技巧和实践

吕伟

v8 支持多线程吗?

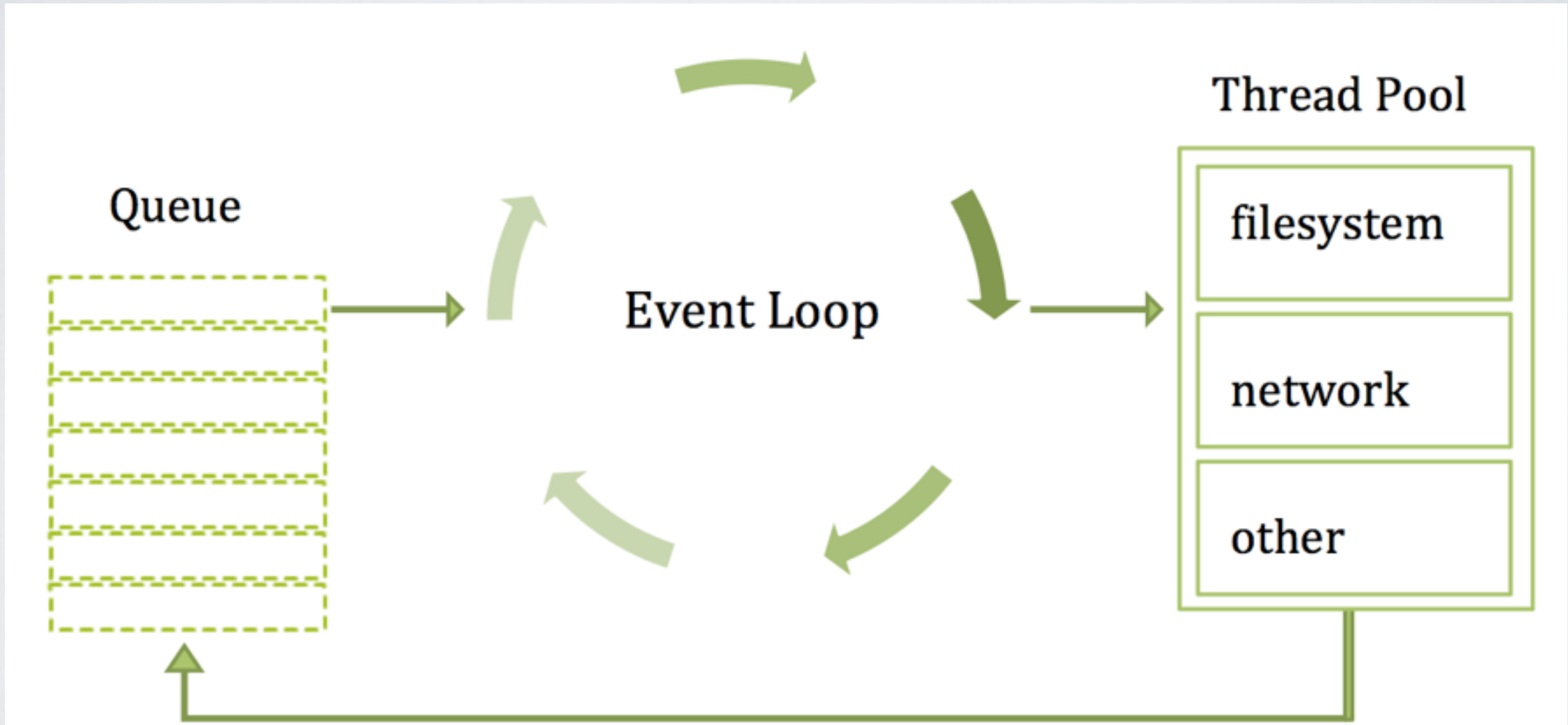
Blocking IO

- IO 耗时和程序 CPU 的耗时差距巨大
- 通常 UI 逻辑的计算耗时肉眼无法察觉
- 通常的网络 IO 延迟肉眼可以察觉

Non-blocking IO

- 这个技术我们并不陌生
- IE 是最早使用 non-blocking IO 技术的浏览器
- 事件驱动 和 Thread Pool 是最常见的解决方案
- 并不是真正的并发和多线程，web worker 才是

Event Loop



Non-blocking 技术更费电

- 维系 thread pool 和 event loop 是要耗费额外的 CPU 的
- 那 non-blocking 让 UI 变快难道是错觉?
- 计算量增加了，单减少了 CPU 的空闲时间

Javascript 的异步编程范式演进

- Callback
- Promise
- Generator
- Async Function

异步和函数式编程

- Pure Function
- Immutable Data Type
- 从上世纪起就被认为是解决异步问题的总要手段

Callback

- Callback 是如何在事件循环中工作的
- Callback Hell 是因为异步依赖
- async.js 是如何解决 callback hell 问题的
- async.js 的让函数难移组合

Callback

```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

```
async.serise([
    function step1 (callback) {
        callback(null, value1);
    },
    function step2 (callback) {
        callback(null, value2);
    },
    function step3 (callback) {
        callback(null, value3);
    },
    function step4 (callback) {
        callback(null, value4);
    }
]);
```

Promise

- Promise 抽象了 box， unbox 以及 continuous
- Promise 更像是一种运算符（monad）
- 因为是运算符，所以一切变得更 composable，包括异常处理
- Promise 并没有消除 callback
- Promise 的 box-unbox 显然会让性能下降，尤其是会引起一些不良 closure 使用习惯

Promise

```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```



```
Promise.resolve(promiseStep1)
.then(promiseStep2)
.then(promiseStep3)
.then(promiseStep4);
```

Generator

- 真正让代码脱离的 callback
- yield 是 switch case 的语法糖
- Generator 是同步的
- 理论性能高于 Promise
- Generator 本意并不是为了解决异步书写问题
- 需要依赖三方的库让 Generator 处理异步流程

Generator async helper 常见库

- Co：https://github.com/tj/co
- Bluebird：https://github.com/petkaantonov/bluebird
- Yaku：https://github.com/ysmood/yaku

```
Node v5.6.0
OS   darwin
Arch x64
CPU  Intel(R) Core(TM) i7-4770HQ CPU @ 2.20GHz

yaku: 117ms
co: 283ms
bluebird: 643ms
```

https://github.com/ysmood/yaku#asyncawait-generator-wrapper

Generator

```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

```
async(function * () {
    yield promiseStep1();
    yield promiseStep2();
    yield promiseStep3();
    yield promiseStep4();
});
```

扩展阅读

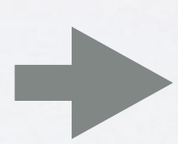
- Generator
- 从 Concurrent 和 Parallel 的区别来理解 Generator 价值
- 懒执行表与里

Async Function

- 只是 Generator 的语法糖
- 比 yield 更直觉的运算优先级
- 省去了三方库的依赖

Async Function

```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

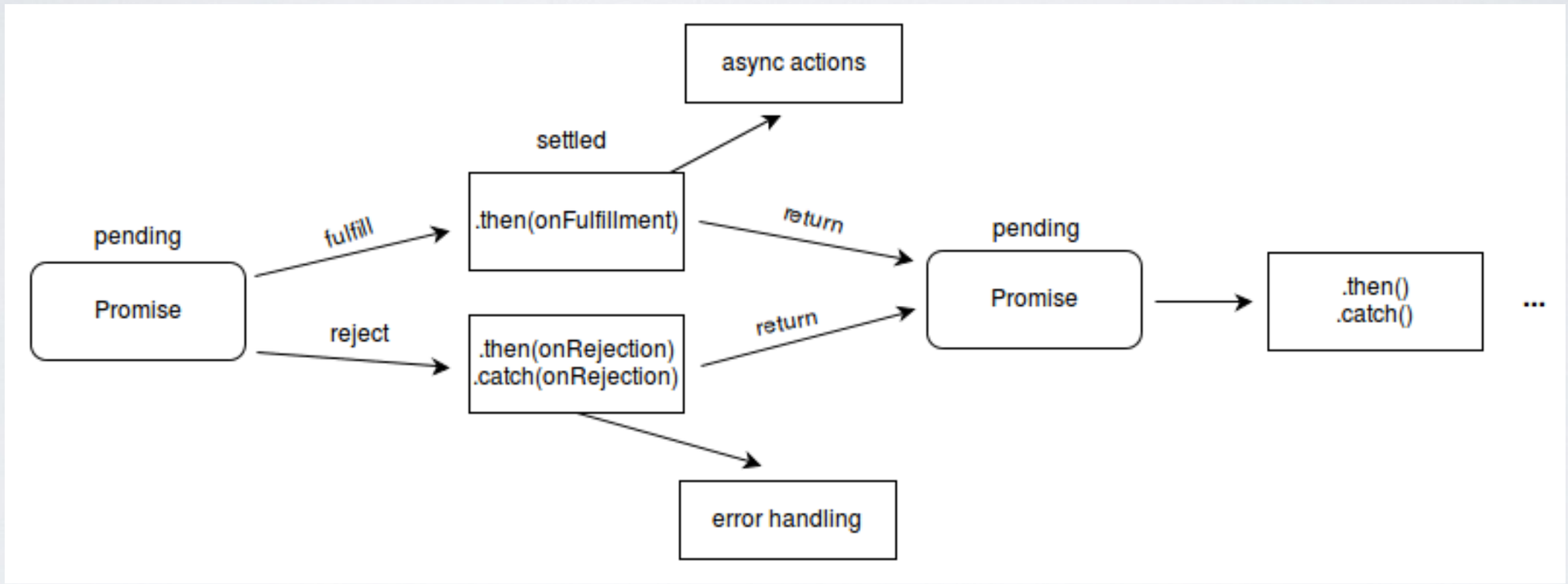


```
async function () {
    await promiseStep1();
    await promiseStep2();
    await promiseStep3();
    await promiseStep4();
}
```

回到 Promise

- 我们发现不论是 Generator 还是 Async Function 它们的核心都是 Promise
- 接下来着重介绍下 Promise

Promise

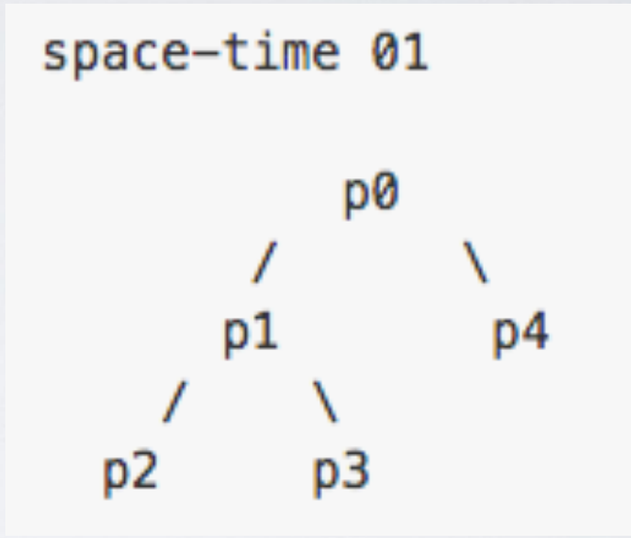

async actions
settled
pending
Promise
fulfill
reject
.then(onFulfillment)
.then(onRejection)
.catch(onRejection)
return
return
pending
Promise
.then()
.catch()
...
error handling

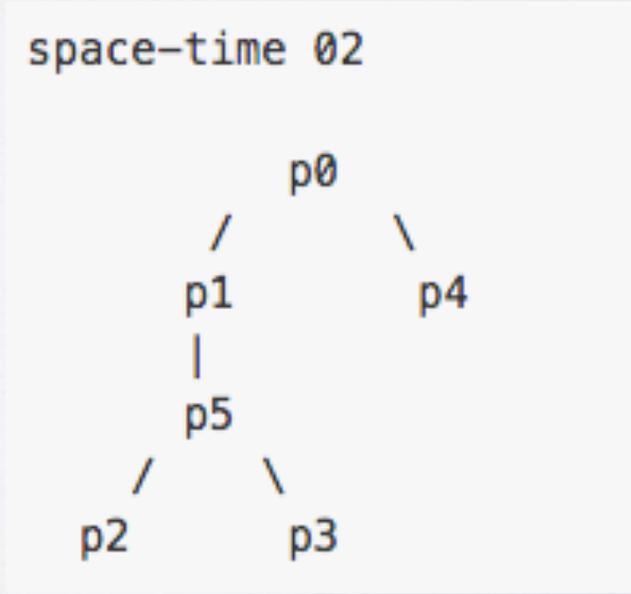
Promise 的数据结构

- Promise 从过程角度看，是一种 Lazy Tree
- 树的结构会在运行时动态变化
- 树的结构在某一时刻一定是确定的
- 在确定时刻之前的树结构不会再随时间改变

Promise 的异常和数据结构

- Uncaught Rejection
- Long Stack Trace





常见 Promise 库对比

name	unit tests	1ms async task	optional helpers	helpers	min js
yaku@0.15.2	✓	357ms / 108MB	✓	32	3.9KB
bluebird@3.3.5	x (27 failing)	283ms / 89MB	partial	100	52.7KB
es6-promise@3.1.2	x (27 failing)	403ms / 113MB	x	10	6.3KB
native@0.15.3	x (9 failing)	572ms / 168MB	x	13	0KB
core-js@2.3.0	x (3 failing)	826ms / 197MB	x	11	12.3KB
es6-shim@0.35.0	x (1 failing)	993ms / 85MB	x	12	55KB
q@1.4.1	x (67 failing)	1555ms / 425MB	x	74	15.4KB
my-promise@1.1.0	x (1 failing)	922ms / 223MB	x	10	8.4KB

https://github.com/ysmood/yaku#compare-to-other-promise-libs

其它

- Web Worker
- Fork
- Cluster

问答

谢谢