构建多线程的 Electron 应用和性能优化实践

仇浩俊

前端开发工程师

GTLC
GLOBAL
TECH LEADERSHIP
CONFERENCE

全球技术领导力峰会

Geekbang. 极客邦科技 | TGO 鲲鹏會

500+高端科技领导者与你一起探讨

技术、管理与商业那些事儿

2019年6月14–15日 | 上海圣诺亚皇冠假日酒店

扫码了解更多信息

自我介绍

毕业后一直从事前端开发的相关工作，参与过多种形态的 Web 产品研发，曾就职于唯品会等互联网企业。2017年加入广发证券信息技术部，目前负责机构交易终端的业务模块、基础架构工作。

目录

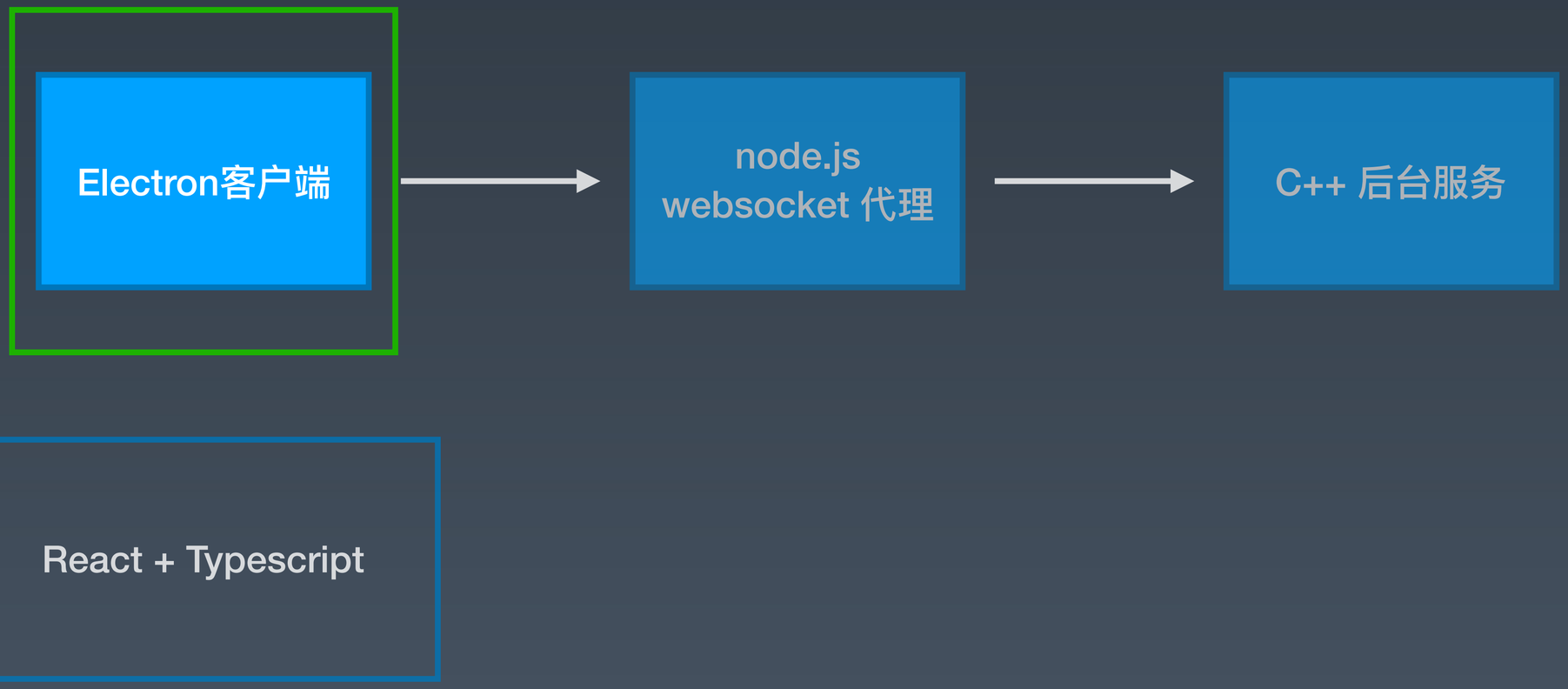
项目背景介绍

"广发投易通"是广发证券全自研的一款面向专业投资客户的极速交易终端产品，其基于Electron技术。产品在运行过程中需要频繁处理海量的实时行情数据和实时交易数据，持续完成数据渲染展示，同时保持对客户交易操作的快速响应。

项目背景介绍

逻辑计算复杂，实时性强

项目背景介绍



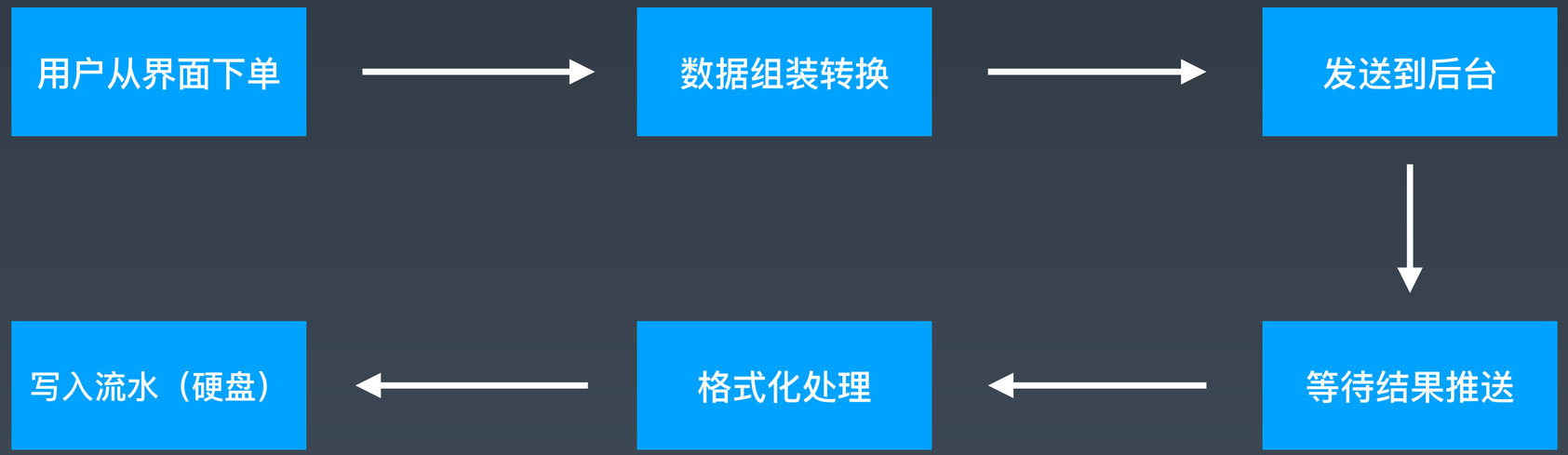
项目背景介绍

问题痛点

盘中的时候，客户下单回报时间非常慢（超过500ms），并且不稳定

项目背景介绍

优化成效



优化目标：减少用户下单的回报时间（以写入流水为准）。

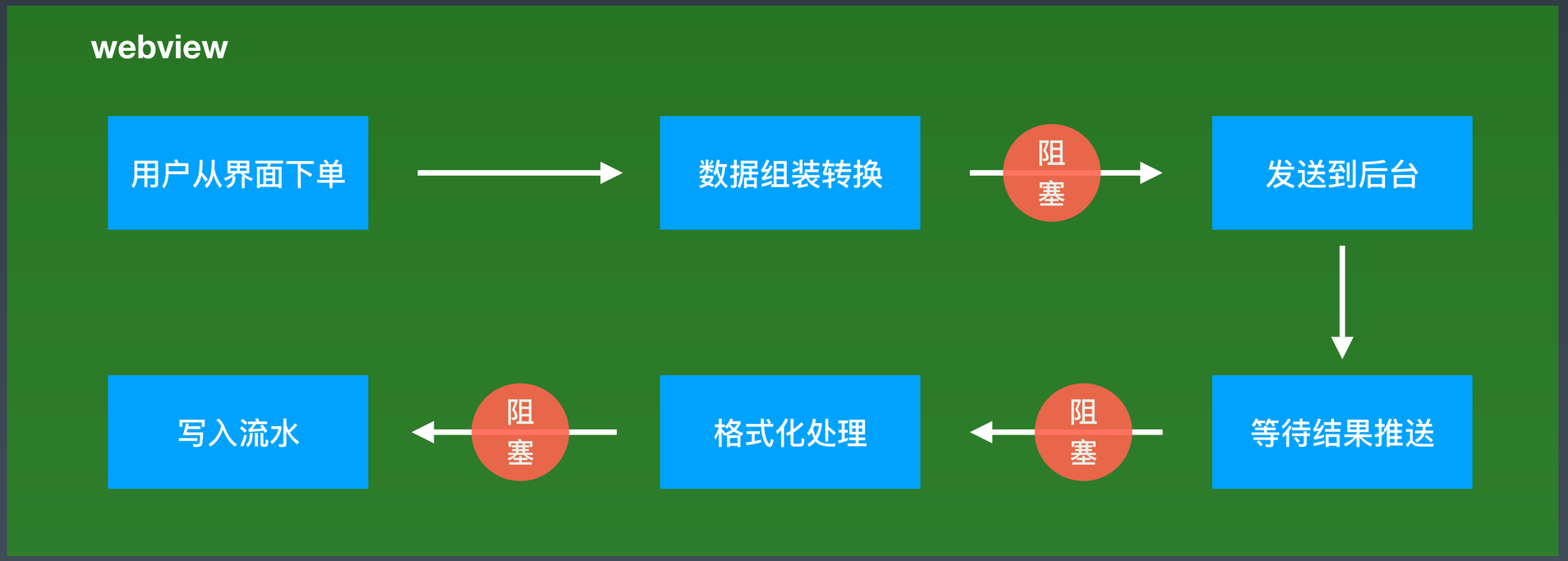
项目背景介绍

优化成效

版本 / 数量	1笔	10笔	20笔
优化前（约）	500ms	1s	1.5s
优化后（约）	120ms 实际更低	250ms	400ms
减少比例	76%	75%	73%

Main / Renderer Process: 多进程架构方案实践

问题分析



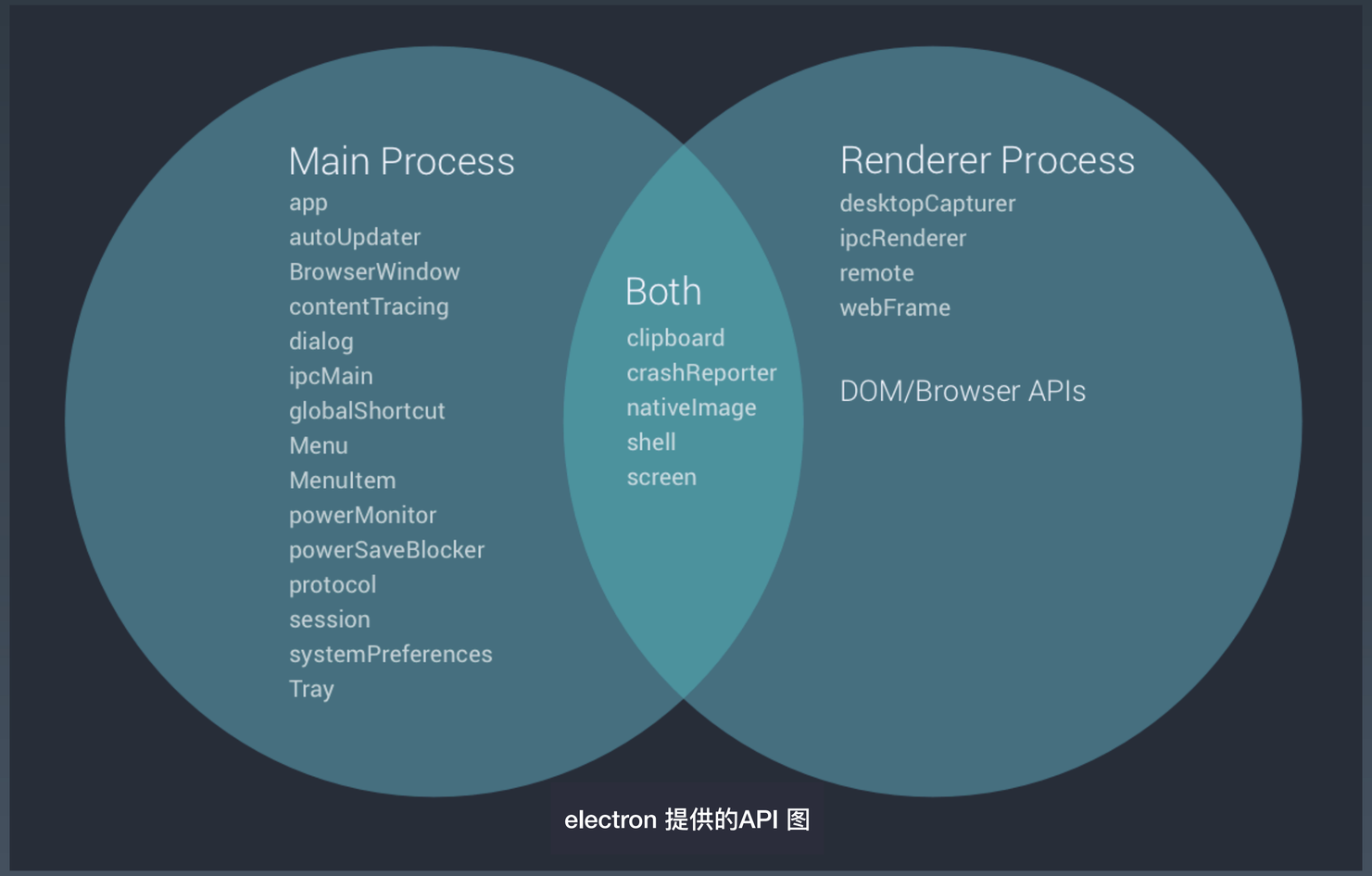
所有的操作都在同一进程中运行，不连续的动作都会马上被抢占CPU资源，导致用户下单回报时间变长

Main / Renderer Process: 多进程架构方案实践

多进程 / 多线程 ？

把交易接口数据和写入流水放到独立的进程中处理，不受渲染进程的计算干扰

Main / Renderer Process: 多进程架构方案实践



electron 提供的API 图

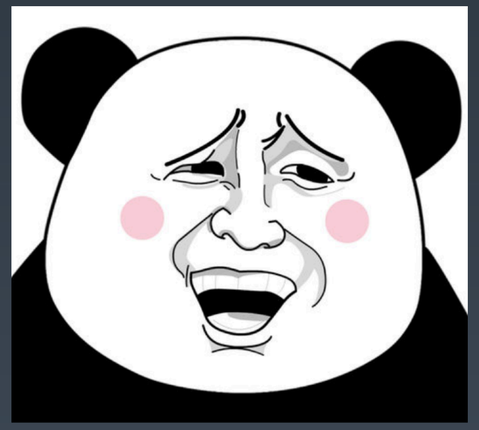
Main / Renderer Process: 多进程架构方案实践

Main / Renderer Process: 多进程架构方案实践

```
// In main process.
const { ipcMain } = require('electron')
ipcMain.on('asynchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.reply('asynchronous-reply', 'pong')
})

ipcMain.on('synchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.returnValue = 'pong'
})
```

```
// In renderer process (web page).
const { ipcRenderer } = require('electron')
console.log(ipcRenderer.sendSync('synchronous-message', 'ping')) // prints "pong"

ipcRenderer.on('asynchronous-reply', (event, arg) => {
  console.log(arg) // prints "pong"
})
ipcRenderer.send('asynchronous-message', 'ping')
```

Main / Renderer Process: 多进程架构方案实践



全部都搞定啦？

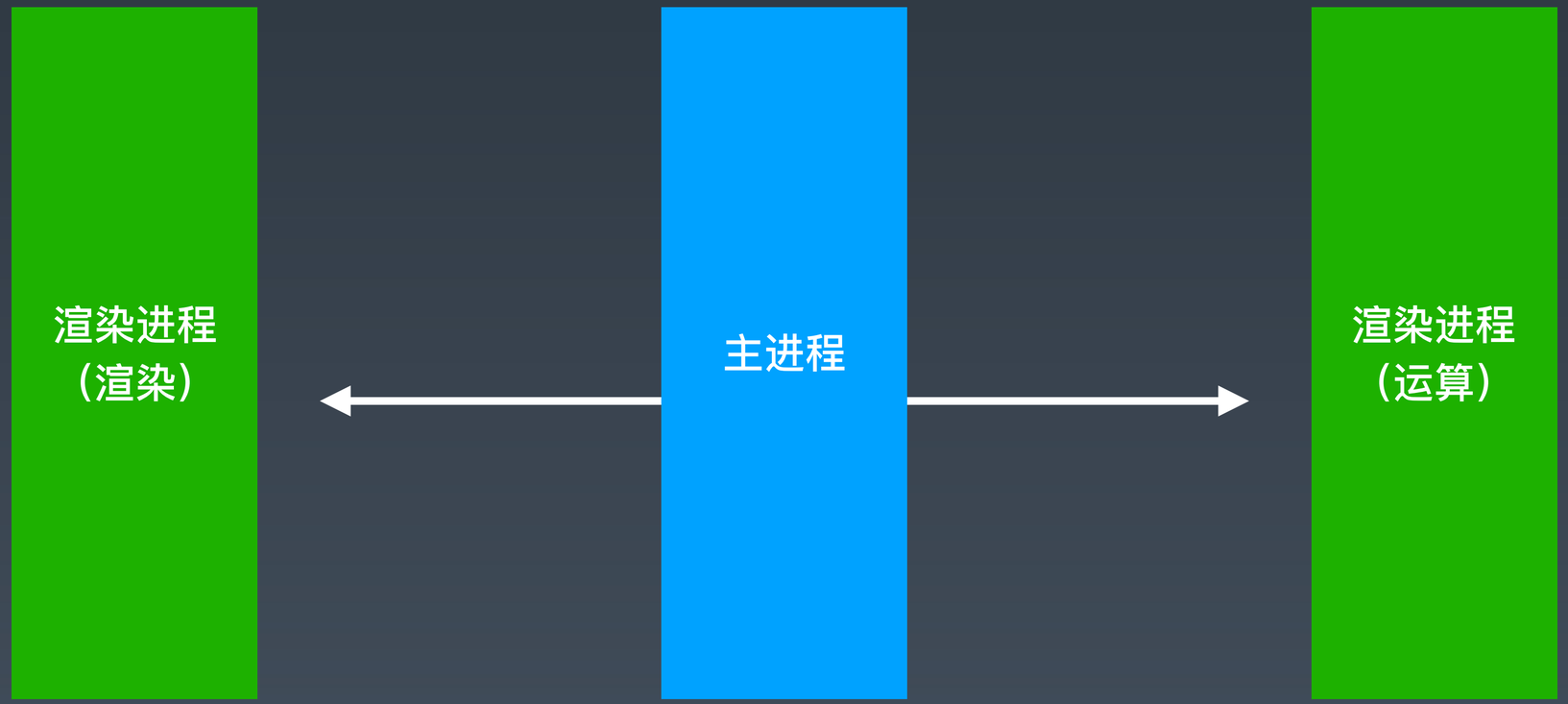
实际上应用的速度没有变快，交互甚至出现了莫名其妙的卡顿。

主进程与渲染进程之间并非相互独立的关系

主进程的大量运算会引起渲染进程的卡顿

主进程发送大量的数据会引起渲染进程的卡顿

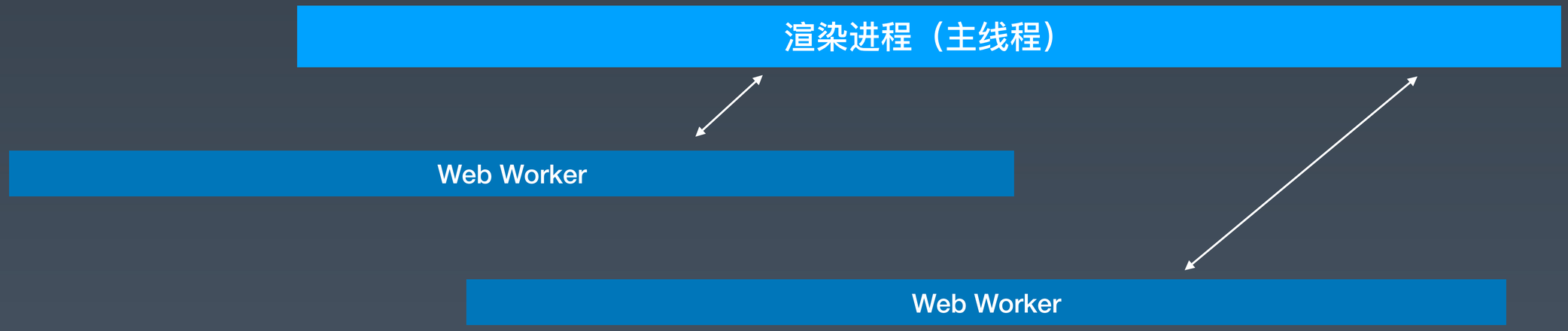
Main / Renderer Process: 多进程架构方案实践



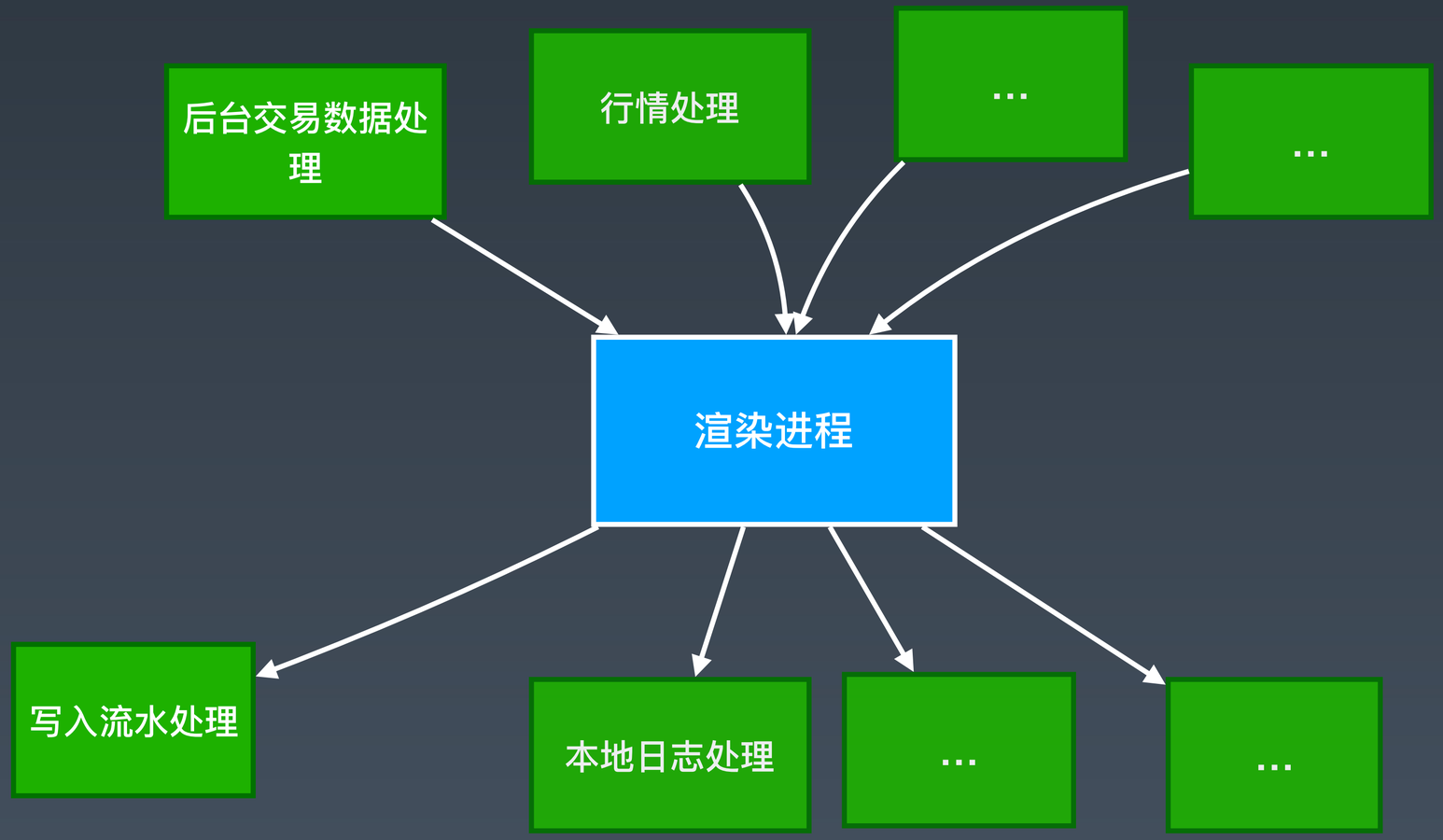
数据通信始终需要经过主进程进行转发

Web Worker: 多线程架构方案实践

Web Worker 为 JavaScript 创造多线程环境，允许主线程创建 Worker 线程，将一些任务分配给后者运行。在主线程运行的同时，Worker 线程在后台运行，两者互不干扰。



Web Worker: 多线程架构方案实践

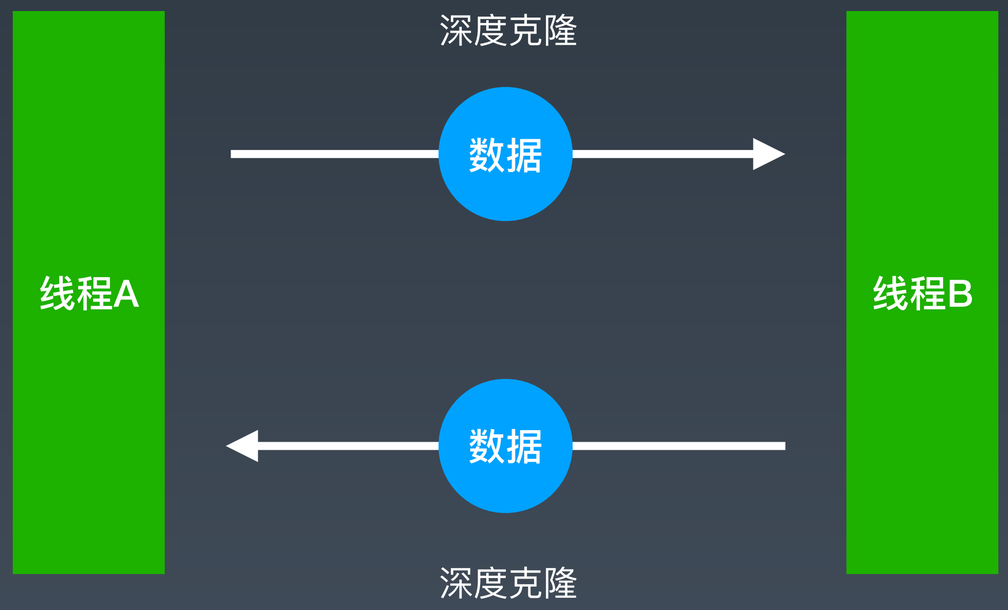


行情信息处理 ：行情订阅推送序列化，数据去重等；

后台交易处理：序列化转换、数据乱序处理等；

写入流水处理：数据结构组织，写入硬盘操作等；

线程间通信的优化



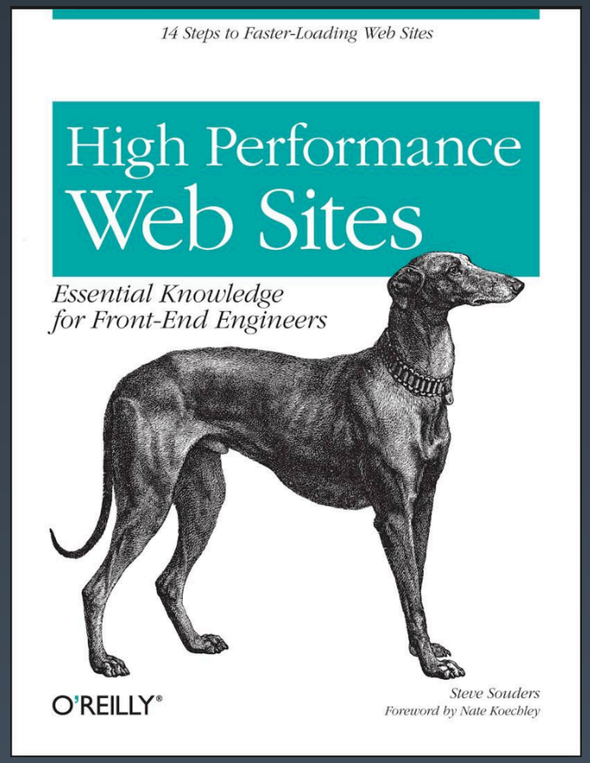
线程间值传递的方式

线程间数据相互独立，避免数据污染

存在数据传输性能耗损，传输大量数据会阻塞发送方

线程间通信的优化

优化思路



从 YUI14 得出的灵感

1. **压缩网络请求大小**
2. **加快网络请求速度**
3. **减少网络请求数量**

线程间通信的优化

第一步 - 压缩数据体积

线程间通信的优化

数据压缩

```
[{
    "name": "John",
    "age": 10
}, {
    "name": "Mary",
    "age": 20
}, {
    "name": "Xiaoming",
    "age": 30
}, {
    "name": "Sophie",
    "age": 40
}]
```

原始数据

```
{
    "field": ["name", "age"],
    "data": [
        ["John", 10],
        ["Mary", 20],
        ["Xiaoming", 30],
        ["Sophie", 40]
    ]
}
```

优化数据结构

```
{
    "f": ["1N", "8C"],
    "d": [
        ["John", 10],
        ["Mary", 20],
        ["Xiaoming", 30],
        ["Sophie", 40]
    ]
}
```

缩短字段名

线程间通信的优化

数据压缩

换一种数据格式？

Protocol-Buffers

常用于前后端的数据交换

传输通过二进制转换，安全性更好

压缩效率高，只传输数据内容不传输格式

JSON
（1.6KB）

下单数据通过 protobuf 转换

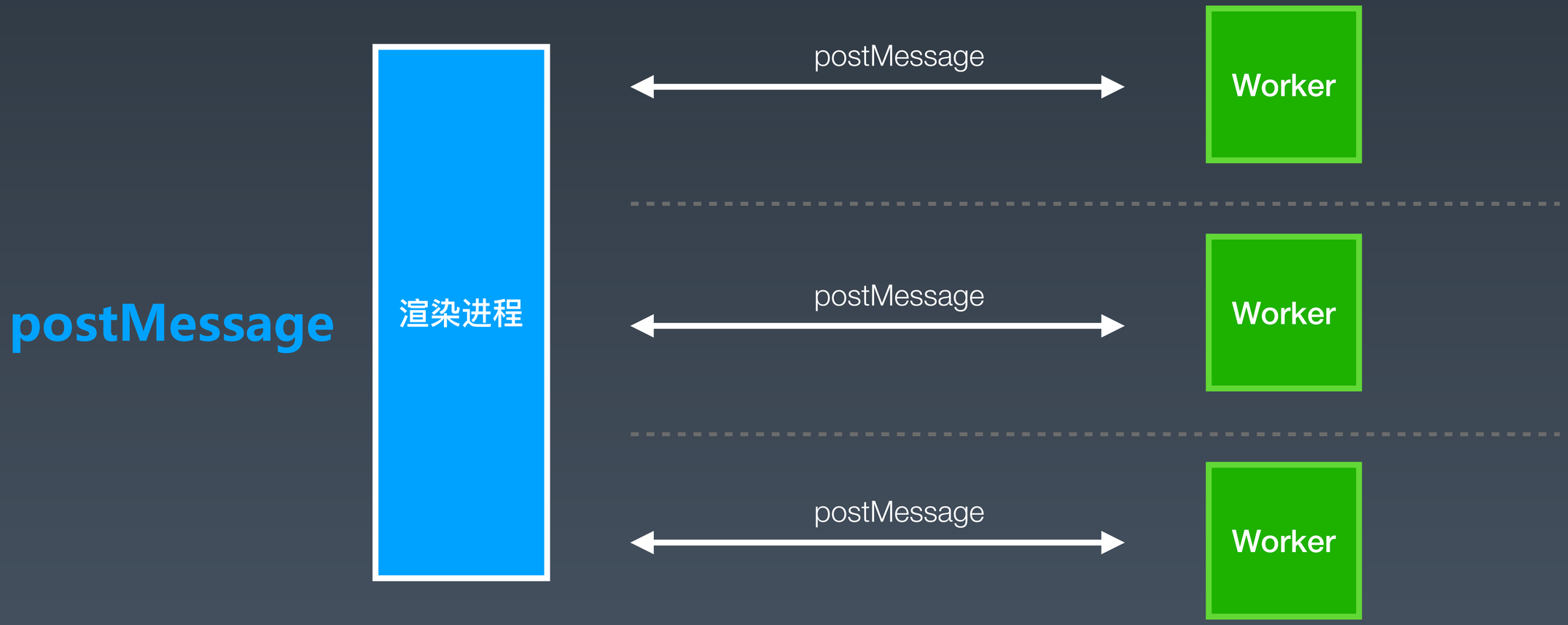
减少 37.5% 的体积

二进制
（1KB）

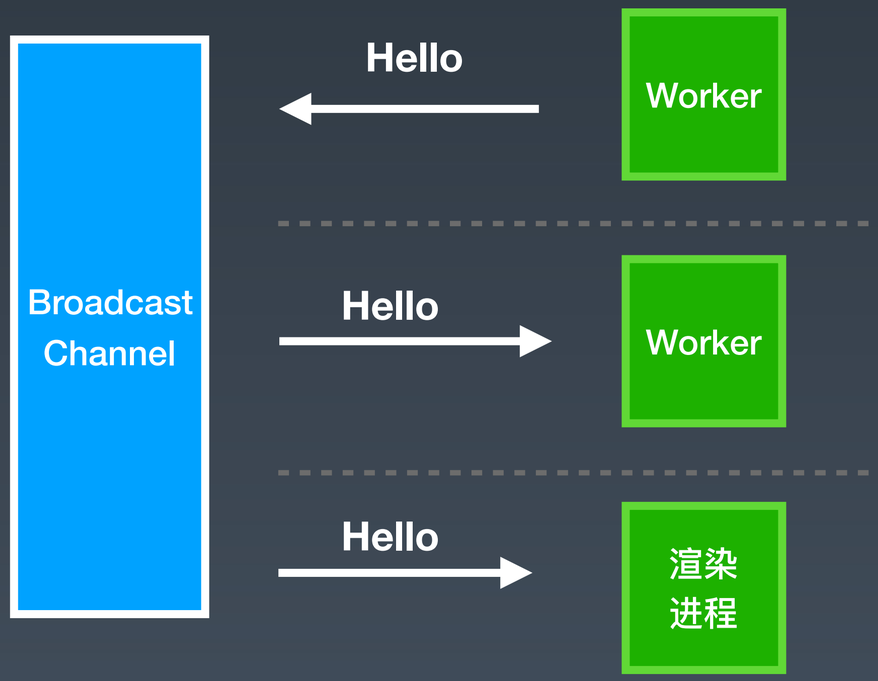
线程间通信的优化

第二步 - 加快通信速度

线程间通信的优化


postMessage
渲染进程
postMessage
Worker
postMessage
Worker
postMessage
Worker

线程间通信的优化

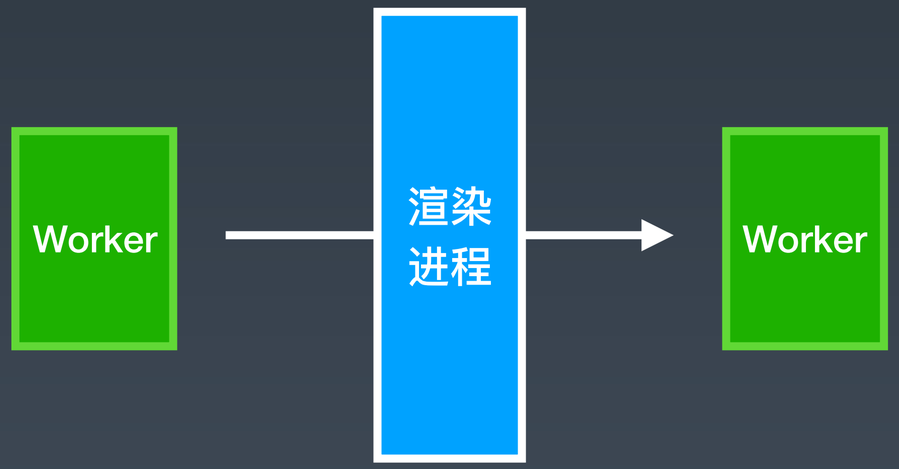


BroadcastChannel

The Broadcast Channel API allows simple communication between browsing contexts (that is windows, tabs, frames, or iframes) with the same origin (usually pages from the same site).

```
// 线程A，加入为global的BroadcastChannel
const channelA = new BroadcastChannel('global');
channelA.onmessage = function (ev) { console.log(ev); }


// 线程B，加入为global的BroadcastChannel
const channelB = new BroadcastChannel('global');
channelB.postMessage('This is a test message.');
```

线程间通信的优化



MessageChannel

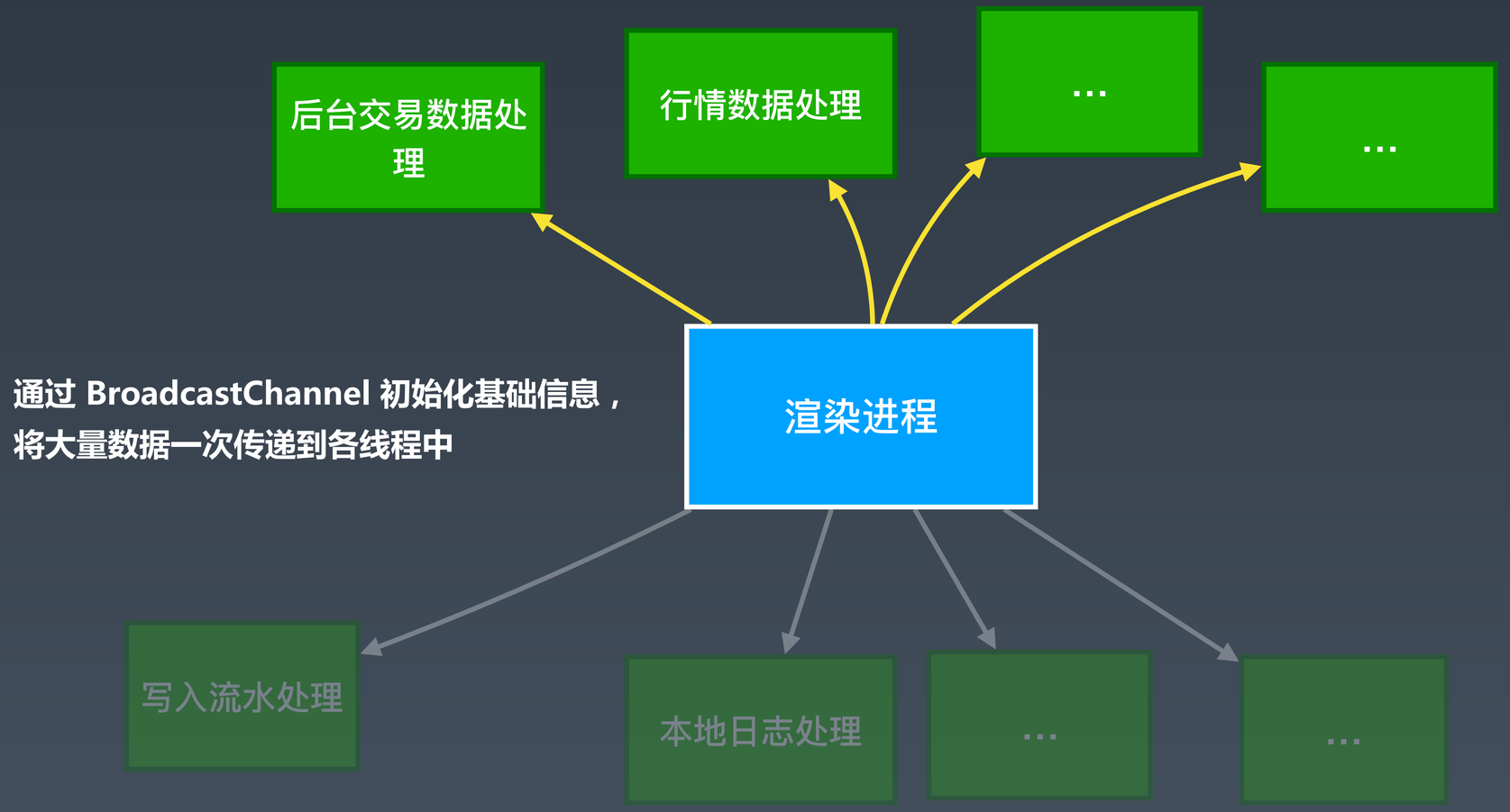
(线程直通，单方向独立管道)

```
const channel = createMessageChannel();

// 两个Worker初始化时传入MessageChannel
initWorkerA(channel.in);
initWorkerB(channel.out);


// WokerA 发送 hello
channel.in.post('hello');

// WokerB 收到 hello
channel.out.listen(data => console.log(data));
```

线程间通信的优化 效率对比

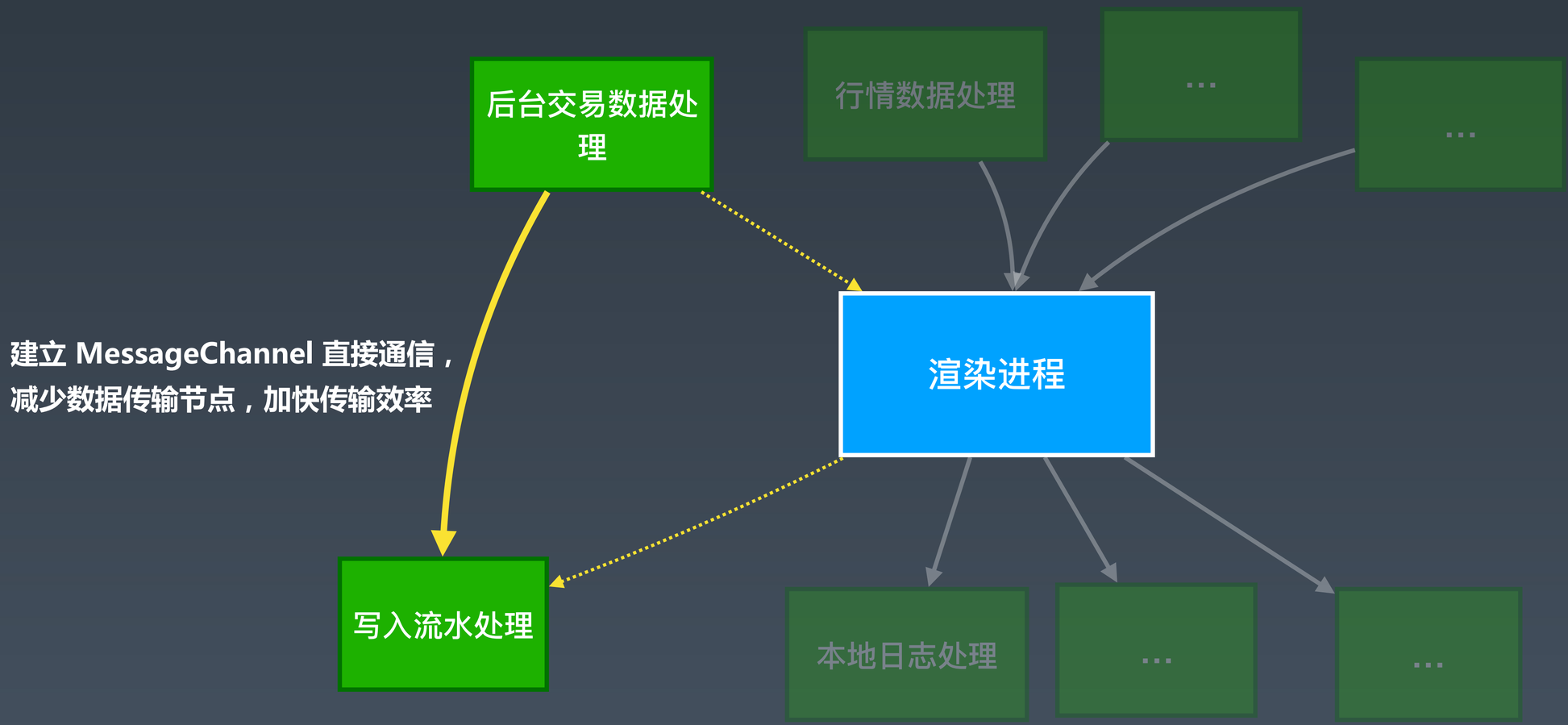
项目 / 类型	postMessage	BroadcastChannel	MessageChannel
传输耗时 （系数，越大越慢）	1	4	1

线程间通信的优化



通过 BroadcastChannel 初始化基础信息，
将大量数据一次传递到各线程中

线程间通信的优化


后台交易数据处理
行情数据处理
...
...
渲染进程
建立 MessageChannel 直接通信，
减少数据传输节点，加快传输效率
写入流水处理
本地日志处理
...
...

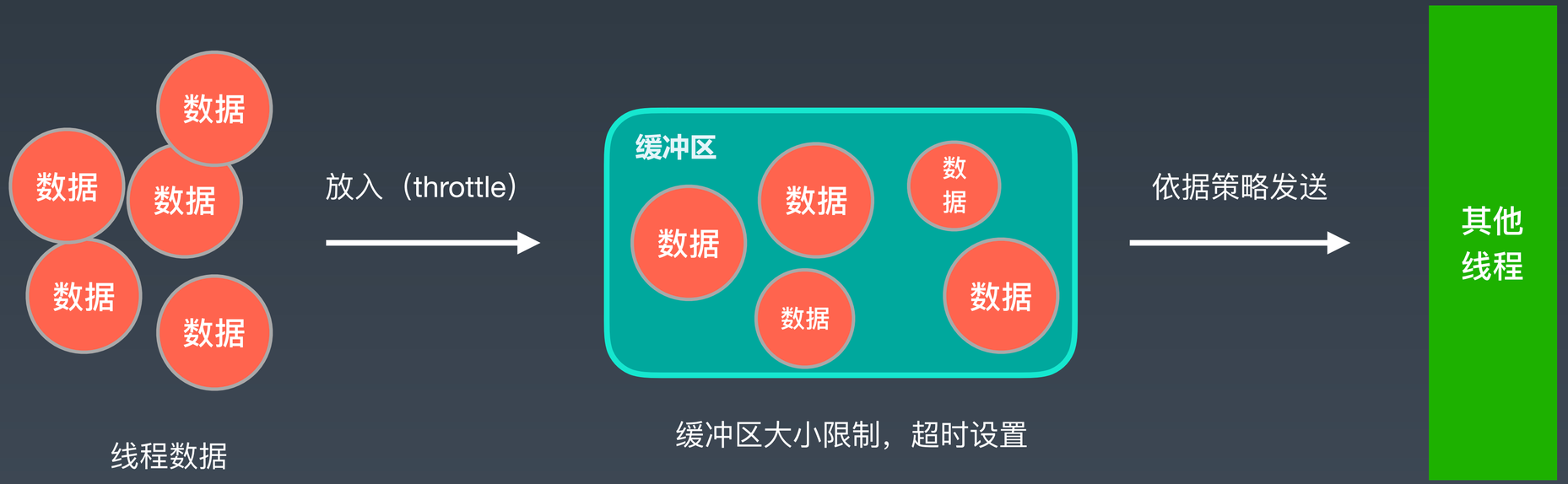
线程间通信的优化 频率控制

传输次数 \ 数据体积	10倍	100倍	1000倍	10000倍
10次	-	12	99	906
100次	18	122	936	8501
1000次	239	1060	8845	-
10000次	2947	10518	-	-

在传输相同数据量大小（大）的情况下，减少传输次数可以有效提高传输效率

线程间通信的优化

频率控制



频率控制会产生以下的问题：

1. 降低了一点数据实时性（业务数据）

2. 线程传输过程产生冗余信息（行情数据）

线程间通信的优化 数据分级

场景（提高实时性）：

用户下单会依次生成指令，委托，成交 三种类型的流水数据，这些数据会快速地从后台线程推送回来，而对于指令数据的实时性要求最高，因为下单响应以收到指令数据为准。

后台推送的数据序列

线程间通信的优化

数据分级

```
[
  {"type": "a", "status": 1},
  {"type": "b", "status": 1},
  {"type": "c", "status": 1},

  {"type": "a", "status": 2},
  {"type": "b", "status": 2},
  {"type": "c", "status": 2},

  {"type": "a", "status": 3},
  {"type": "b", "status": 3},
  {"type": "c", "status": 3}
]
```

a代表指令，b、c分别代表委托、成交

```
[
  {"type": "a", "status": 1},
  {"type": "a", "status": 2},
  {"type": "a", "status": 3},
```

```
  {"type": "b", "status": 1},
  {"type": "c", "status": 1},
  {"type": "b", "status": 2},
  {"type": "c", "status": 2},
  {"type": "b", "status": 3},
  {"type": "c", "status": 3}
]
```

数据根据优先级分组，分两次传输

线程间通信的优化

第三步 - 减少通信数据量

线程间通信的优化

数据去重

场景（减低冗余）：

交易数据的变化非常快，假如时间窗口内一只股票成交了N笔，那么后台就会推送N条流水数据到前端，但对于UI只需要展示5笔。

广发证券 13.36:200	广发证券 13.40:100	广发证券 13.39:150	……	广发证券 13.36:100	广发证券 13.40:500

后台推送的数据序列

线程间通信的优化 数据去重

```
[
    {code: '000776', n: '200', d: 'b'},
    {code: '000776', n: '100', d: 's'},
    {code: '000776', n: '100', d: 's'},
    {code: '000776', n: '150', d: 'b'},
    // ... 很多条数据
    {code: '000776', n: '100', d: 'b'},
    {code: '000776', n: '150', d: 'b'},
    {code: '000776', n: '500', d: 's'},
    {code: '000776', n: '500', d: 's'},
]
```

大量交易流水数据

```
[
    {code: '000776', n: '200', d: 'b'},
    {code: '000776', n: '100', d: 's'},
    {code: '000776', n: '100', d: 'b'},
    {code: '000776', n: '500', d: 's'},
    {code: '000776', n: '500', d: 's'},
]
```

在缓冲区发送前，只选取最新的前5条流水

线程间通信的优化

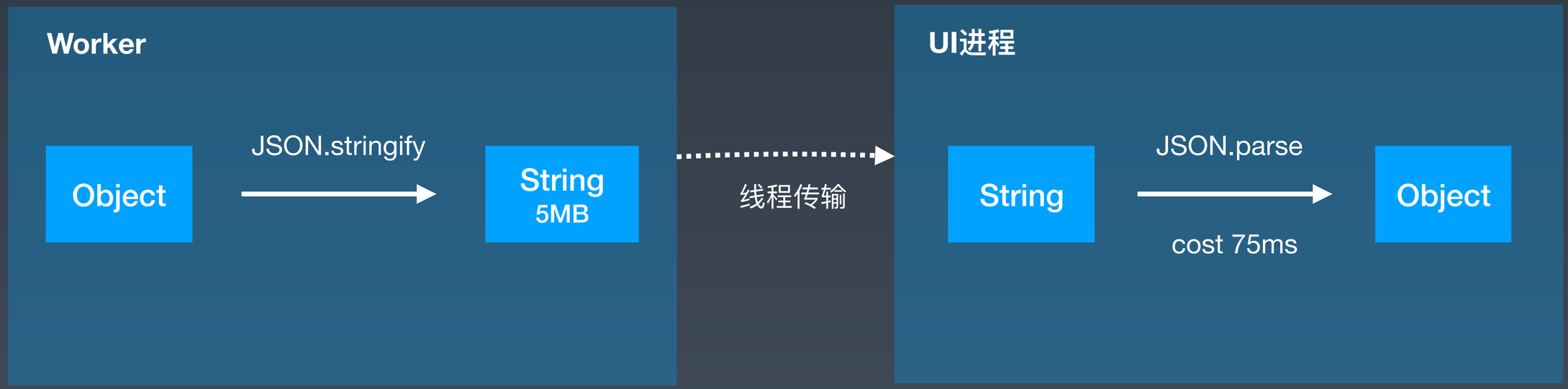
序列化转换

体积 / 类型	对象	字符串传输	字符串（反序列化）
10KB	13.54ms	3.88ms	0.1ms
500KB	375.34ms	27.78ms	8ms
5MB	3590ms	255.02ms	75ms

字符串的传输速度比对象快很多

线程间通信的优化

序列化转换



字符串的传输速度比对象快很多，但是反序列化操作会占用接收的线程资源，

频繁发送到渲染进程会引起卡顿

线程间通信的优化

序列化转换

```
const book = () => {
    return {
        name: '我爱写代码',
        price: 12.5,
    }
};

const data = [
    {a: book()},
    {b: book()},
    {c: book()},
    {d: book()},
]
```

```
const book = {
    name: '我爱写代码',
    price: 12.5,
};

const data = [
    {a: book},
    {b: book},
    {c: book},
    {d: book},
]
```

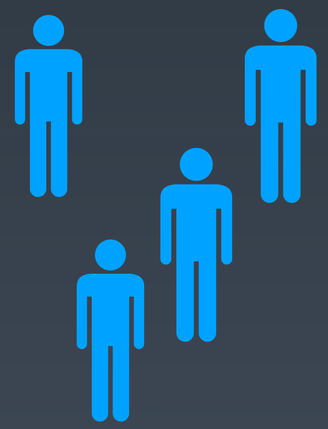
对需要传输的数据中，数据保持相同的引用值越多，实际体积在某种程度上越小，传输的速度越快。

以左图的数据结构为例，大量引用相同的数据结构可以减少约15% ~ 20%的传输时间。

结构化克隆算法：https://developer.mozilla.org/zh-CN/docs/Web/Guide/API/DOM/The_structured_clone_algorithm

建立极速的索引机制

优化计算



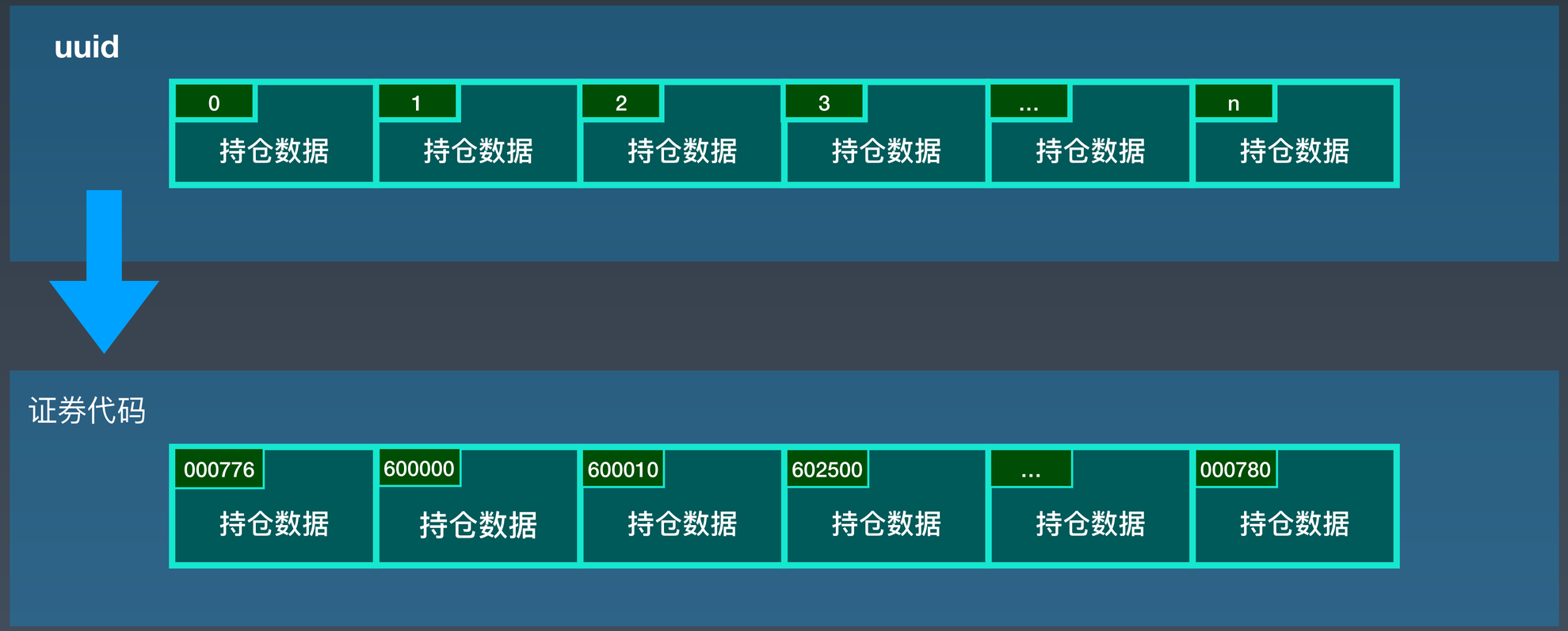
小明:平安银行:600	小花:海南航空:1000	小明:广发证券:3000	小春:金字火腿:500	小张:东阳光:2000	小春:三六零:800

持仓是键为uuid对象结构，每项值记录用户的股票持仓数量

1. 多账户（4）批量卖出500只成分股
2. 交易规则校验
3. 查找每个用户的持仓信息
4. 计算可卖的股票数量
5. 略

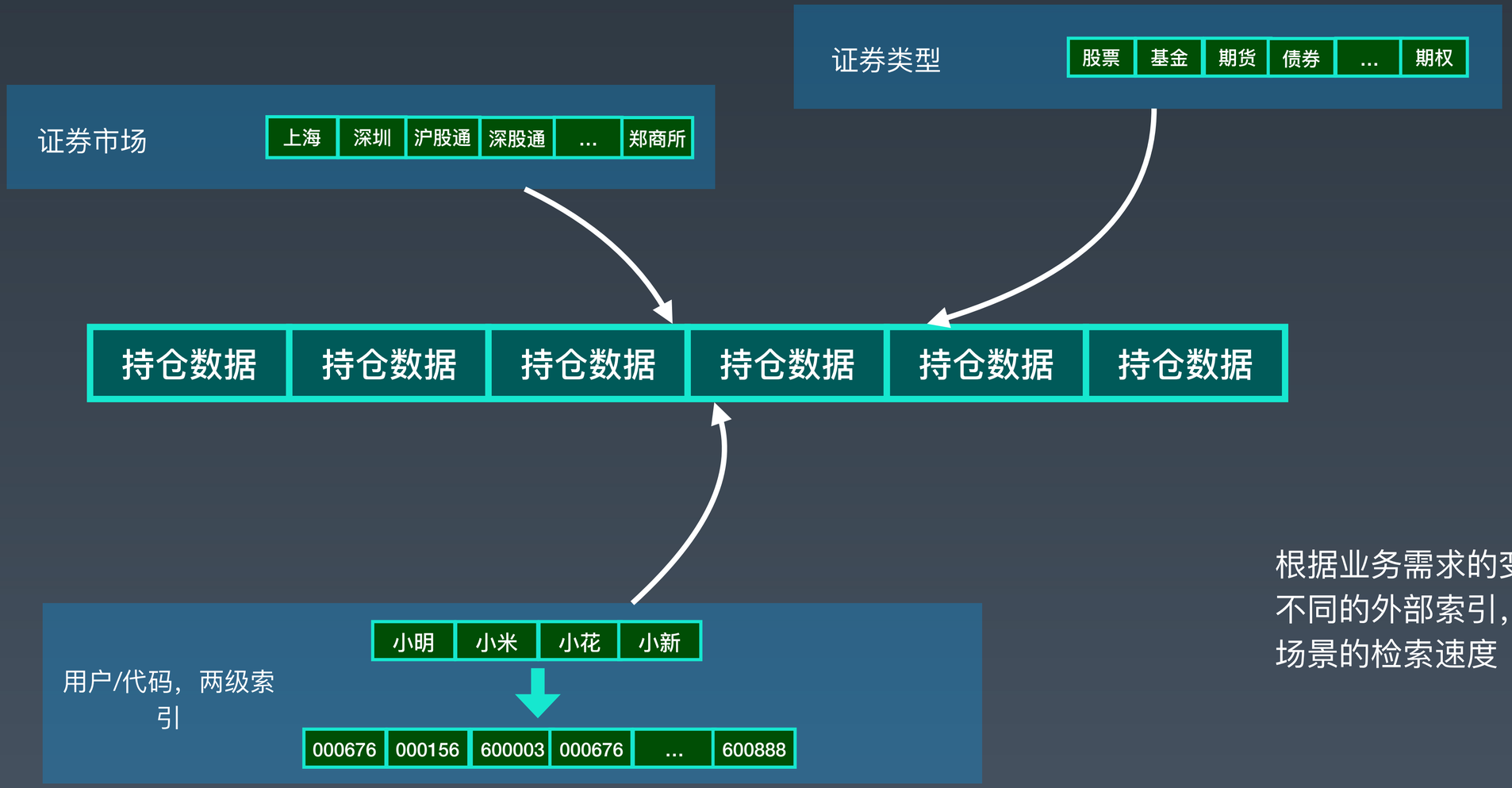
假设单个用户有2000条股票持仓，4个用户共有8000条记录（没有区分用户），那么单次查找500只股票的信息耗时约为500ms，那么四次查找持仓信息共耗时为2s。

建立极速的索引机制



转换数据结构，将证券代码作为键值

对业务数据建立极速的索引机制



根据业务需求的变化，建立不同的外部索引，提高各种场景的检索速度

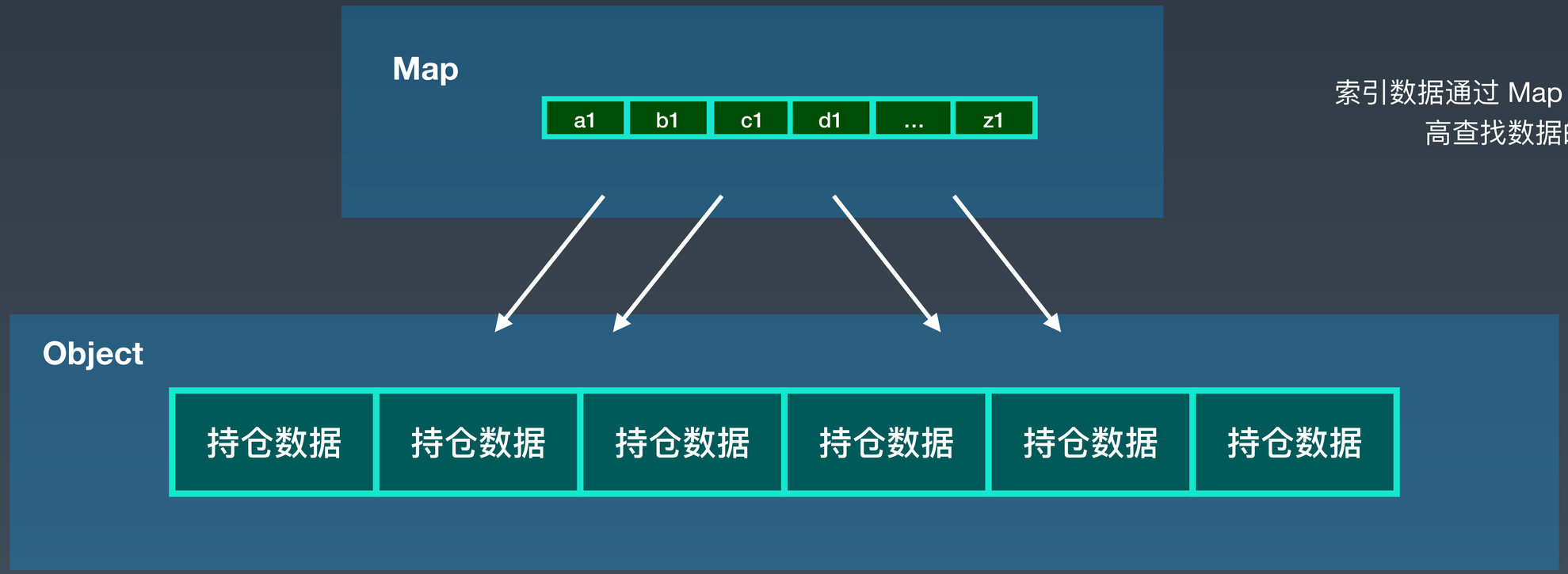
建立极速的索引机制

map lookup	`map.get('test1')`	800,163,648 ±0.85% fastest
obj lookup	`obj["test1"];`	108,547,562 ±5.26% 87% slower
obj set	`obj["test2"] = "hmmmmm";`	102,875,944 ±5.44% 88% slower
map set	`map.set("test2","hmmmmm");`	78,575,796 ±1.87% 90% slower

在大量数据的读写场景下，Map 的读取速度约为 Object 的8倍，但前者的写入速度较后者减少了20%

https://jsperf.com/map-vs-object-property-access/14

建立极速的索引机制



索引数据通过 Map 结构存储，提高查找数据的效率

业务数据通过 Object 结构存储，提高修改数据的效率

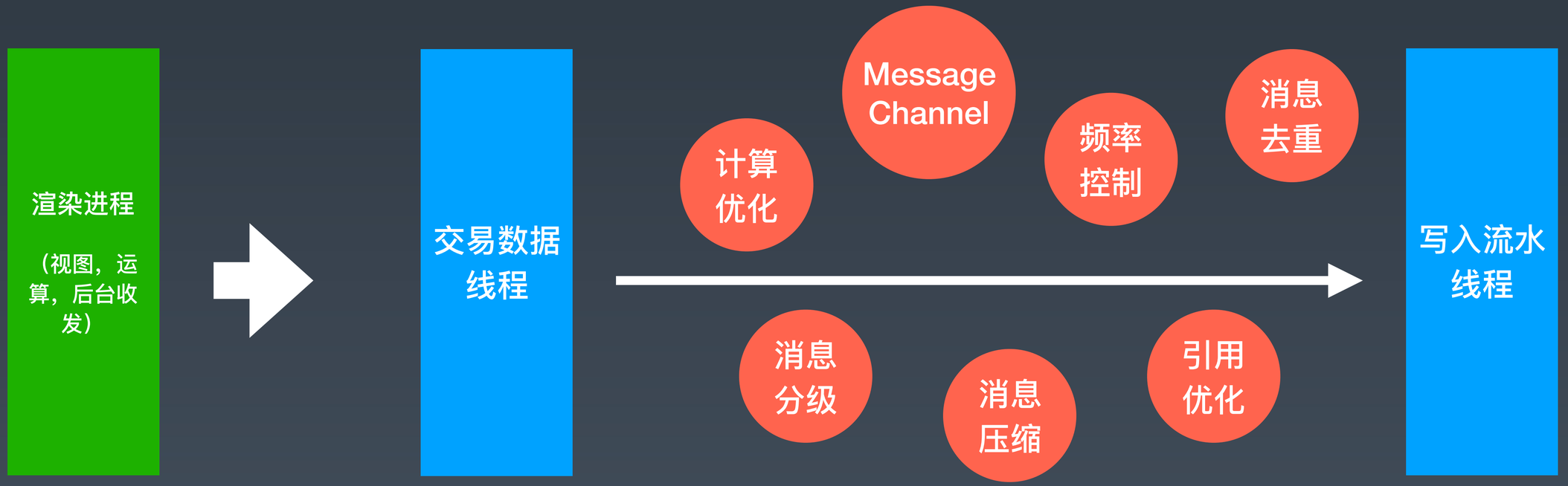
建立极速的索引机制

优化成效

耗时 / 持仓数	100	500	2000	5000
优化前	8ms	22ms	130ms	350ms
优化后	-	-	0.14ms	0.232ms

对500只股票查找用户持仓数据的速度提升

总结


渲染进程
（视图，运算，后台收发）
交易数据线程
Message Channel
计算优化
频率控制
消息去重
消息分级
消息压缩
引用优化
写入流水线程

总结

版本 / 数量	1笔	10笔	20笔
优化前（约）	500ms	1s	1.5s
优化后（约）	120ms 实际更低	250ms	400ms
减少比例	76%	75%	73%

用户下单到成功回报的耗时减少了70%以上

极客时间 | 企业服务

这是一封正经“福利”贴

想做团队的领跑者 需要迈过这些“槛”

成长型企业，易忽视人才体系化培养 企业转型加快，团队能力又跟不上	VS	从基础到进阶，超100+一线实战 技术专家带你系统化学习成长
团队成员技能水平不一， 难以一“敌”百人需求	VS	解决从小白到资深技术人所遇到 80%的问题
寻求外部培训，奈何价更高且 集中式学习	VS	多样、灵活的学习方式，包括 音频、图文 和视频
学习效果难以统计，产生不良循环	VS	获取员工学习报告，查看学习 进度，形成闭环

课程顾问「橘子」

回复「QCon」
免费获取
学习解决方案

 # 极客时间企业账号 # 解决技术人成长路上的学习问题

THANKS!

QCon 10th