Node的设计与实现

By @朴灵

个人简介

- 朴灵@阿里巴巴数据平台
- JacksonTian@GitHub
- 资深开发工程师
- 《深入浅出Node.js》作者

“叩首问路，码梦为生。”

–Jackson Tian

带着问题听

- LinkedIn -90%的机器，20x性能提升
- groupon ~50,000rpm， RT↓↓↓

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

–nodejs.org

“Node.js is a platform built on **Chrome's JavaScript runtime** for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

–nodejs.org

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an **event-driven**, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

–nodejs.org

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, **non-blocking I/O model** that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

–nodejs.org

Ryan Dahl

Node之父

关于高性能服务器的思考

进程/用户

- process, 13MB
- 10GB / 13MB = ～787并发

线程/用户

- thread, 2MB
- 10GB / 2MB = ～5120并发

事件驱动

- 单线程/所有用户
- 资源消耗极小

什么是事件驱动?

什么是事件驱动?

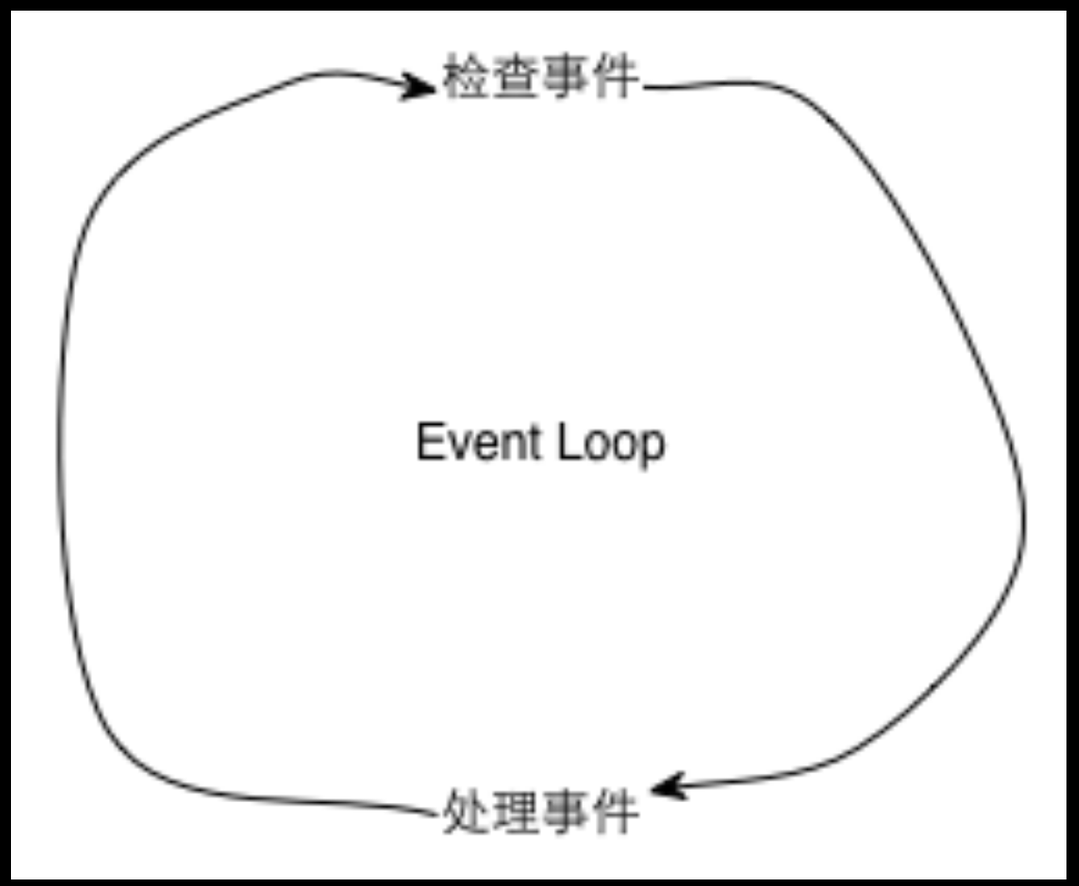
- 用户发起一个HTTP请求是一个事件
- 点击一次界面是一个事件
- 打开一个文件是一个事件
- etc.

事件驱动的场景

- Event Loop
- Node
- GUI: iOS
- Browser

Event Loop

while (true)



如何检查事件?

Watcher

- 向Watcher询问是否有事件需要处理
  - timer watcher
  - fs watcher
  - udp/req watcher
  - process watcher
  - etc.

Handles

- 由Watcher产生具体要处理的事件
- setTimeout
  - 当时间到达时，产生事件，执行handle
  - handle就是执行传入的回调函数

Event Loop -> Watcher -> Handles

Event Loop的退出

- 为什么console.log()执行完后就退出
- 为什么http.server()可以一直让进程执行
- 当Event Loop中没有Watcher的时候退出进程

Node中Event Driven实现

- Windows: IOCP
- Linux: epoll
- Mac: kqueue
- Solaris: event ports

Event Driven的问题

- 单线程的阻塞问题
- 一旦阻塞，事件的处理就变得低效

Non-Blocking I/O

数据访问

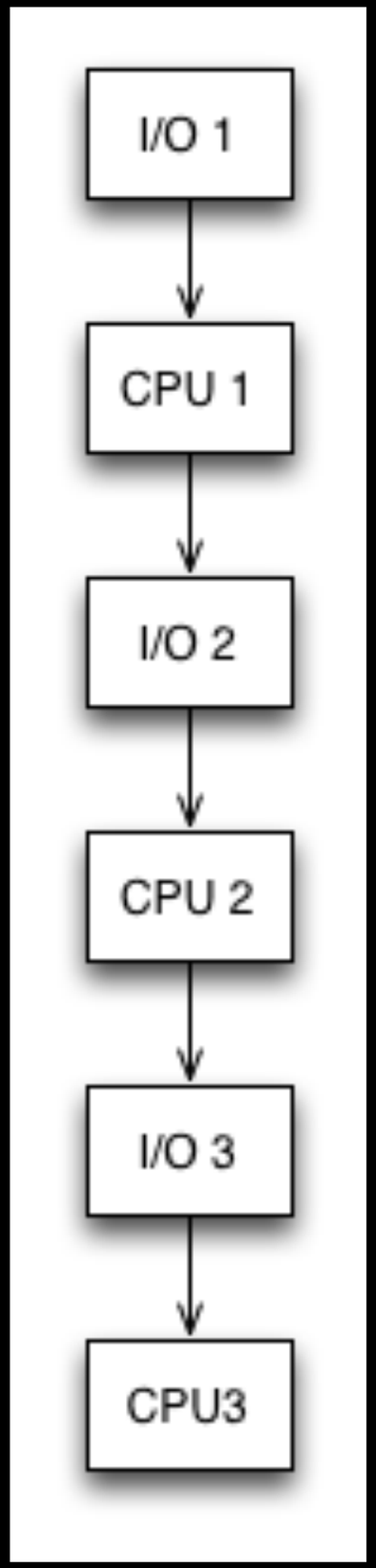
- CPU L1 Cache: 3 tick
- CPU L2 Cache: 14 tick
- Memory: 250 tick
- Disk: 41000000 tick
- Net: 240000000 tick

阻塞问题

- 磁盘I/O和网络I/O的访问阻塞CPU的执行
- 进行磁盘I/O和网络I/O时，CPU浪费
- 后续计算无法进行

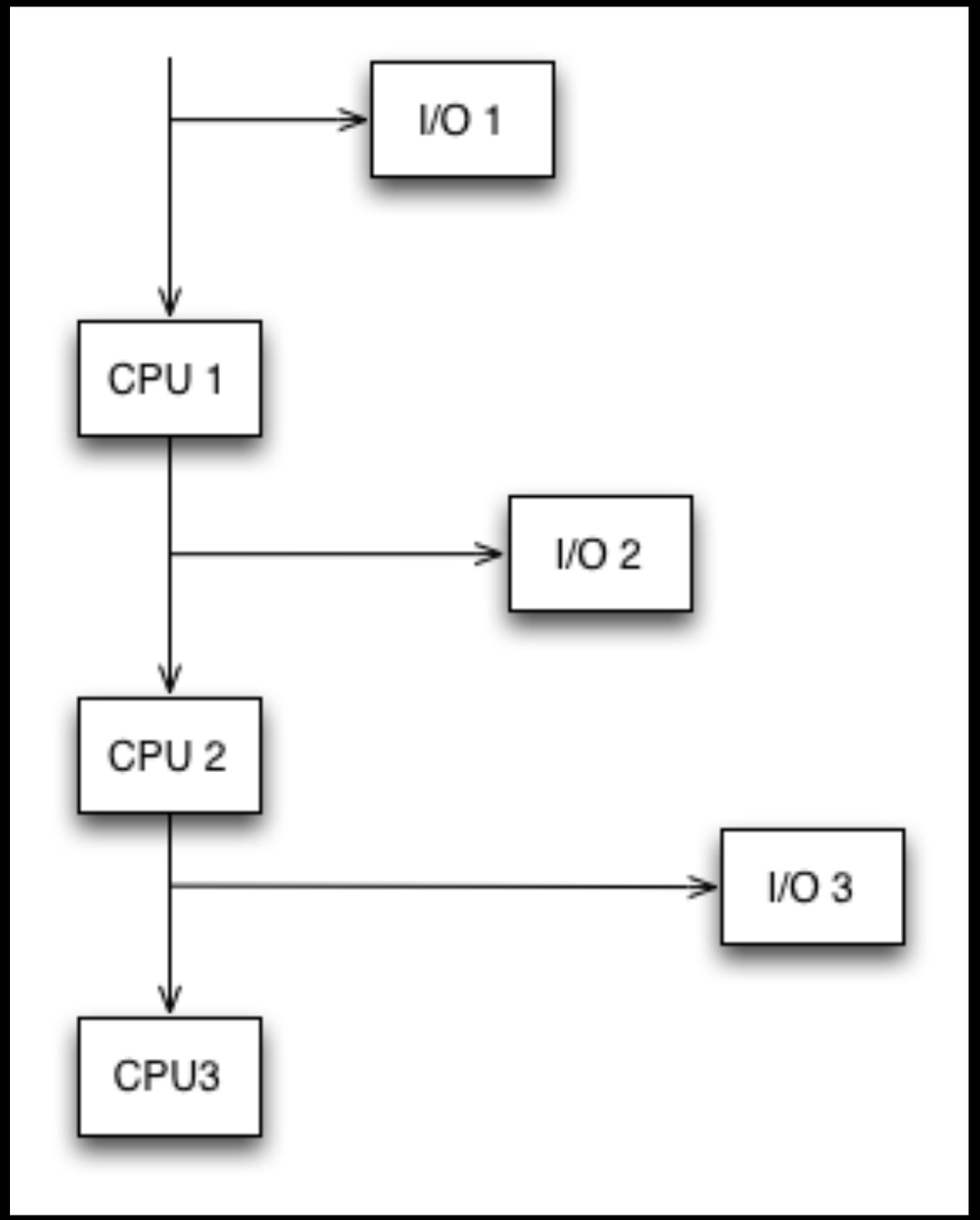
阻塞执行

CPU与I/O互相阻塞
导致Event Loop执行效率低下



非阻塞执行

CPU与I/O并行执行
计算设备与I/O设备互不干扰



非阻塞I/O的实现

- 没有完美的非阻塞I/O
- 通过线程池结合事件驱动实现

选择JavaScript的原因

- 成熟的事件驱动模式（浏览器中）
- 没有I/O库，没有历史包袱，利于构建非阻塞IO库

V8的性能

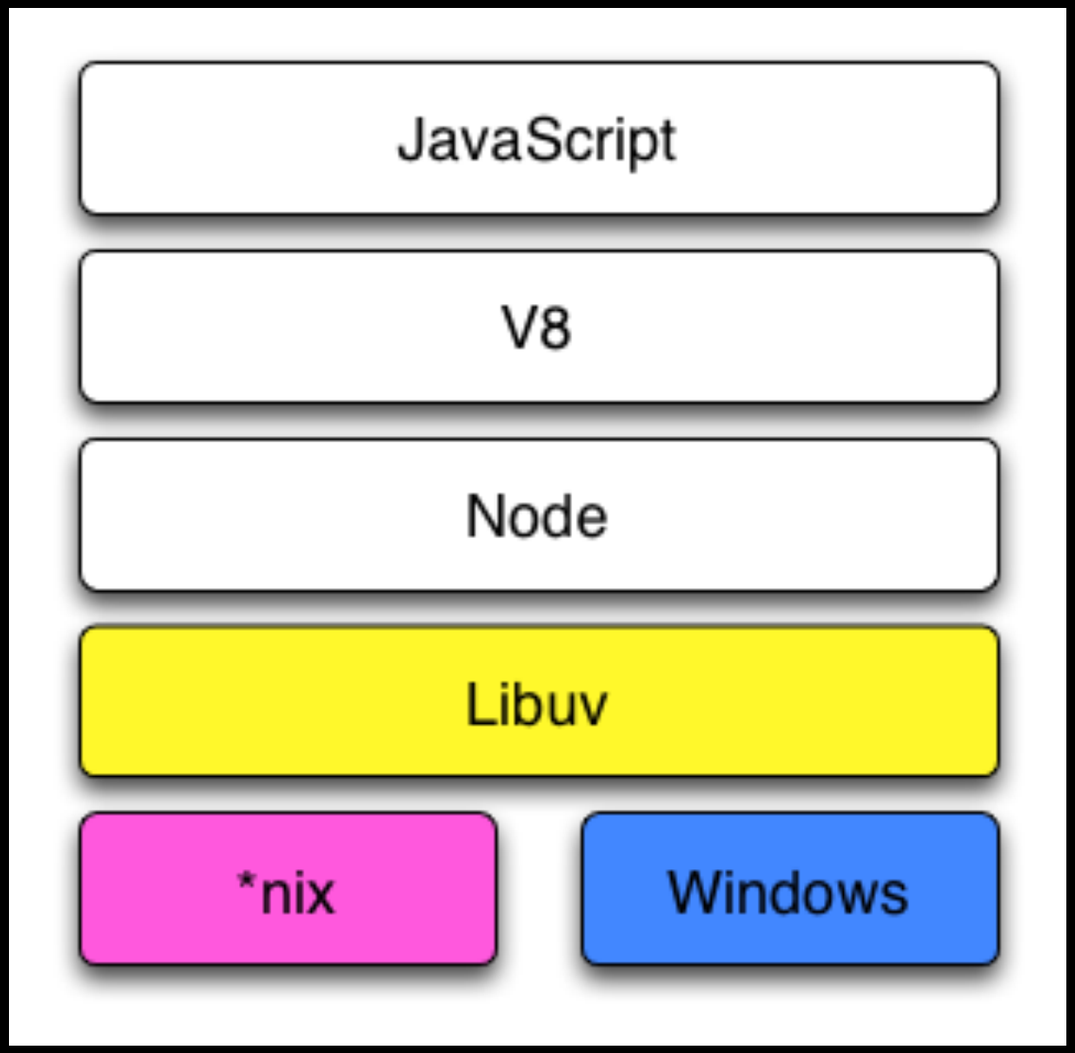
- V8作为JavaScript引擎，一改JavaScript执行缓慢的形象
- 直接生成机器码
- 分代式GC
- 优化

- Event Driven
- Non-Blocking I/O
- JavaScript/V8

跨平台

实现跨平台

- 分层架构
- 引入Libuv层
- 分别实现Windows和*nix平台的功能
- 分别编译



单线程问题

单线程问题

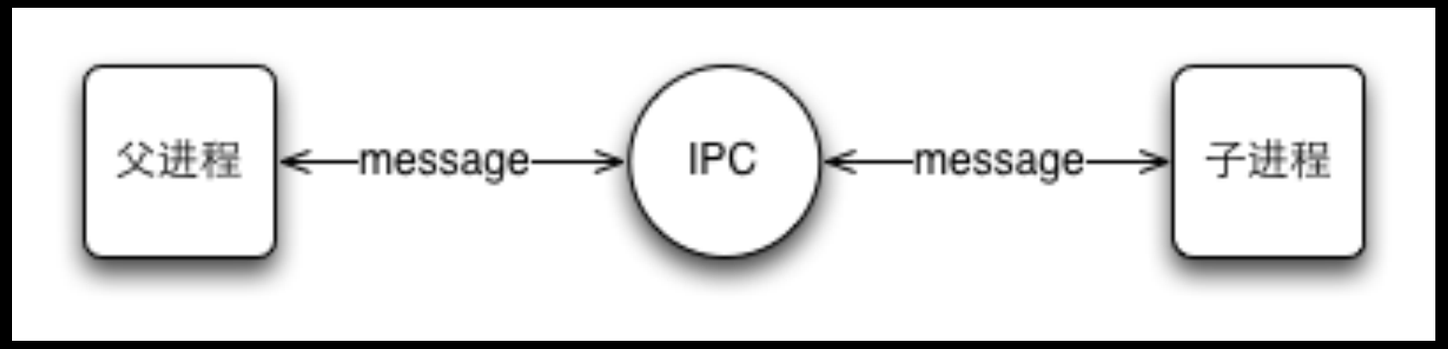
- 单线程服务大量用户请求
- 异常导致进程退出时会丢失大量用户请求
  - 谁让能力越大责任越大呢

单线程问题

- 多核CPU利用问题
- 不应当浪费服务器资源

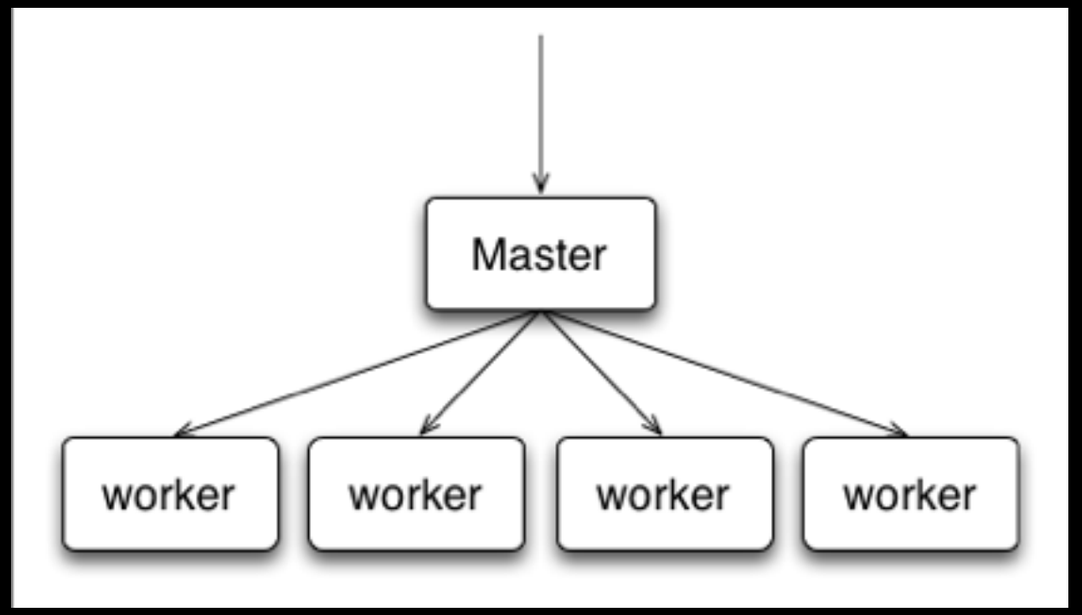
进程间消息传递

- 进程间不共享数据
- 通过消息传递



子进程/Cluster

- 分离监控和业务
- 充分利用硬件资源



用户使用层面：模块

模块

- 基于CommonJS Module规范构建
- 公共模块平台：NPM

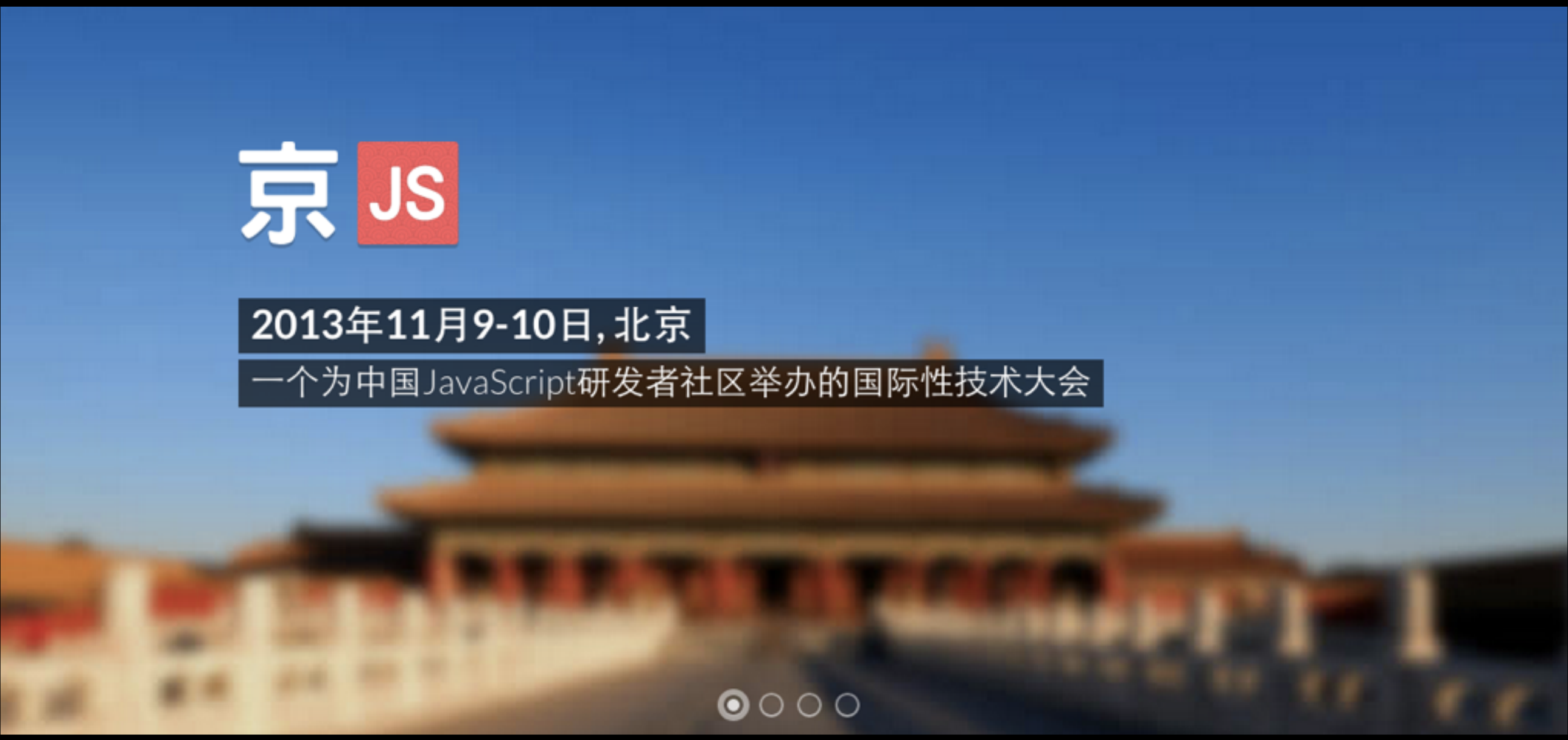
异步编程问题

- Promise（when、Q）
- EventProxy
- Async/Step

Harmony

- generators
- coroutines

Thanks/Q&A



http://jingjs.org/