# 理解NodeJS 实现高并发原理

讲者: 子 非

# 内容

- Node.js的诞生以及简介
- Node.js单线程实现高并发原理：
  - ✓ 单线程
  - ✓ 非阻塞I/O （non-blocking I/O）
  - ✓ 事件驱动/事件循环
- Node.js适用场景

# Node.js的诞生

# Node.js的诞生

- Ryan Dahl为了解决Web 服务器的高并发性能问题，他认为通过事件驱动和异步I/O来达成目的是问题的关键。

# Node.js的诞生

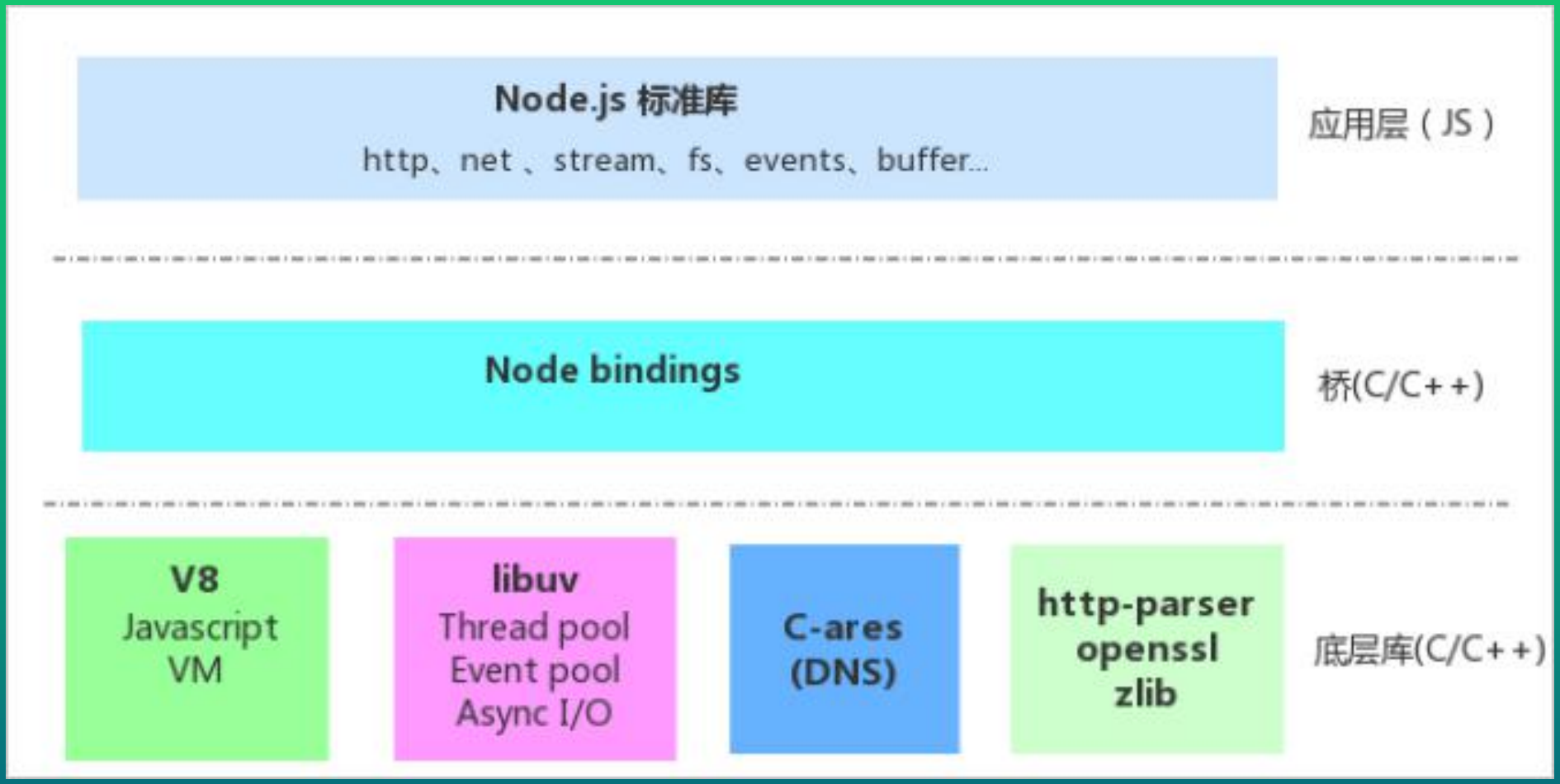
- 2008年Google发明了Chrome浏览器，使用V8引擎来解析JS程序满足了他的想象。
  - ✓ 历史遗留问题少，都是异步I/O
  - ✓ 强大的编译和快速执行效率（通过运用大量算法和技巧）远超Python和ruby等脚本语言
  - ✓ JavaScript语言的闭包特性非常方便
  - ✓ 利用事件驱动机制
- Rydan Dahl 就把V8移植到了服务器端，2009年底，Ryan Dahl JSConf EU会议上发表关于Node.js的演讲，之后Node.js逐渐流行于世。

# Node.js简介

# NodeJS简介

- 构建在Chrome浏览器V8引擎上的JavaScript 运行环境
- 非阻塞I/O模型
- 事件驱动
- 花最小的硬件成本，追求更高的并发，更高的处理性能

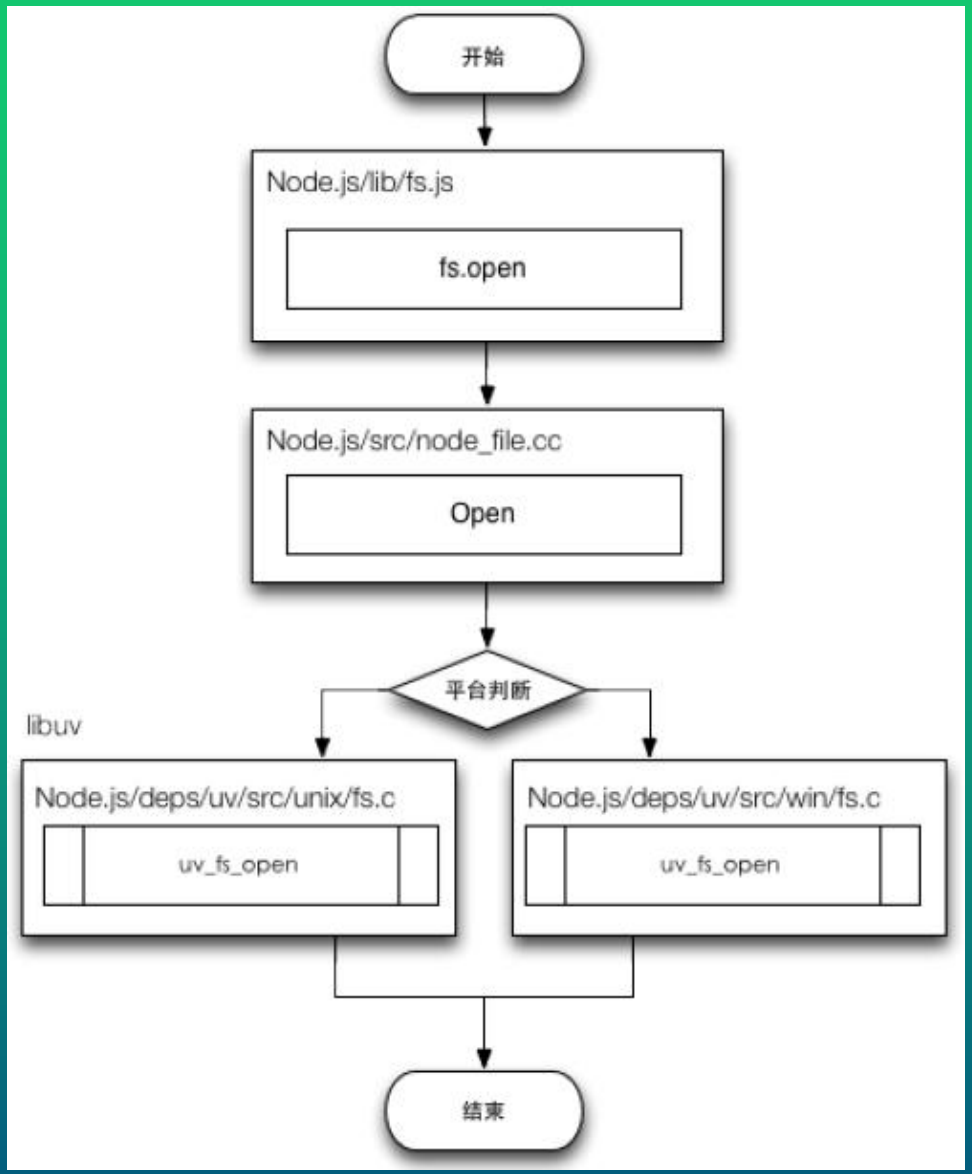
# NodeJS简介


Node.js 标准库
http、net 、stream、fs、events、buffer...
应用层（JS）
Node bindings
桥(C/C++)
V8
Javascript
VM
libuv
Thread pool
Event pool
Async I/O
C-ares
(DNS)
http-parser
openssl
zlib
底层库(C/C++)

# NodeJS简介

```
var fs = require('fs');
fs.open('./test.txt', "w", function(err, fd) {
    //..do something
});
```

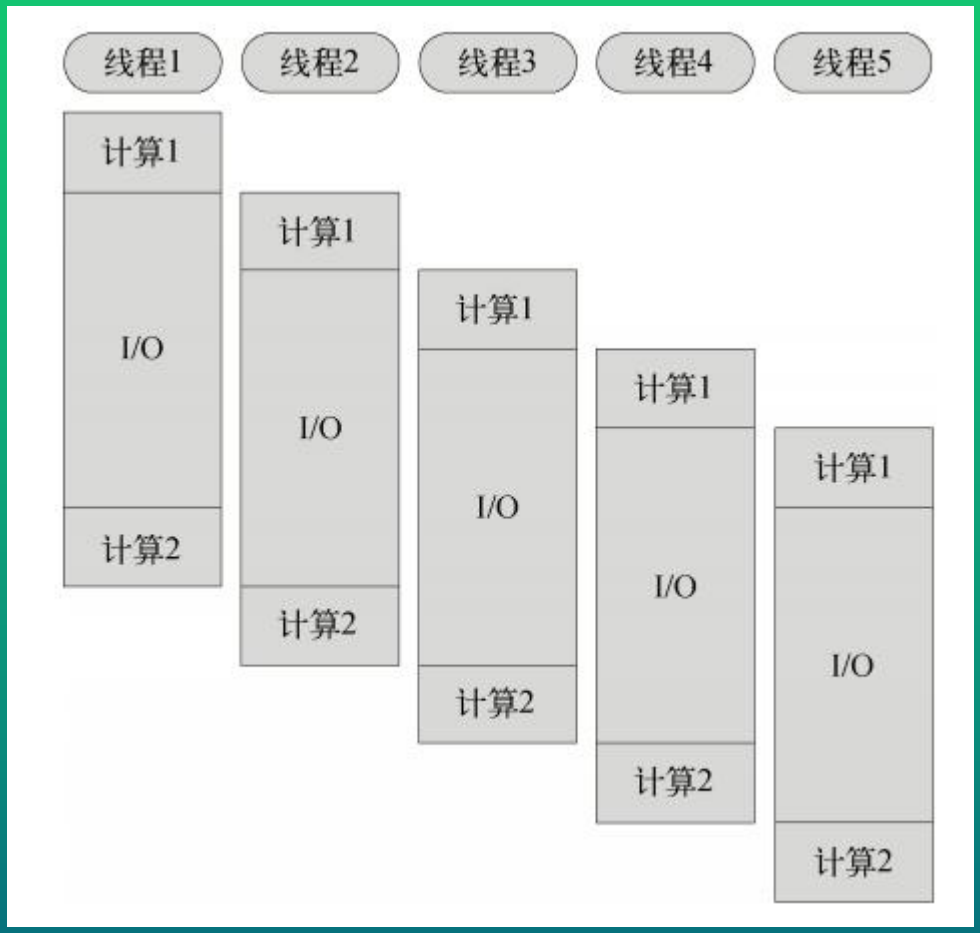
# NodeJS简介

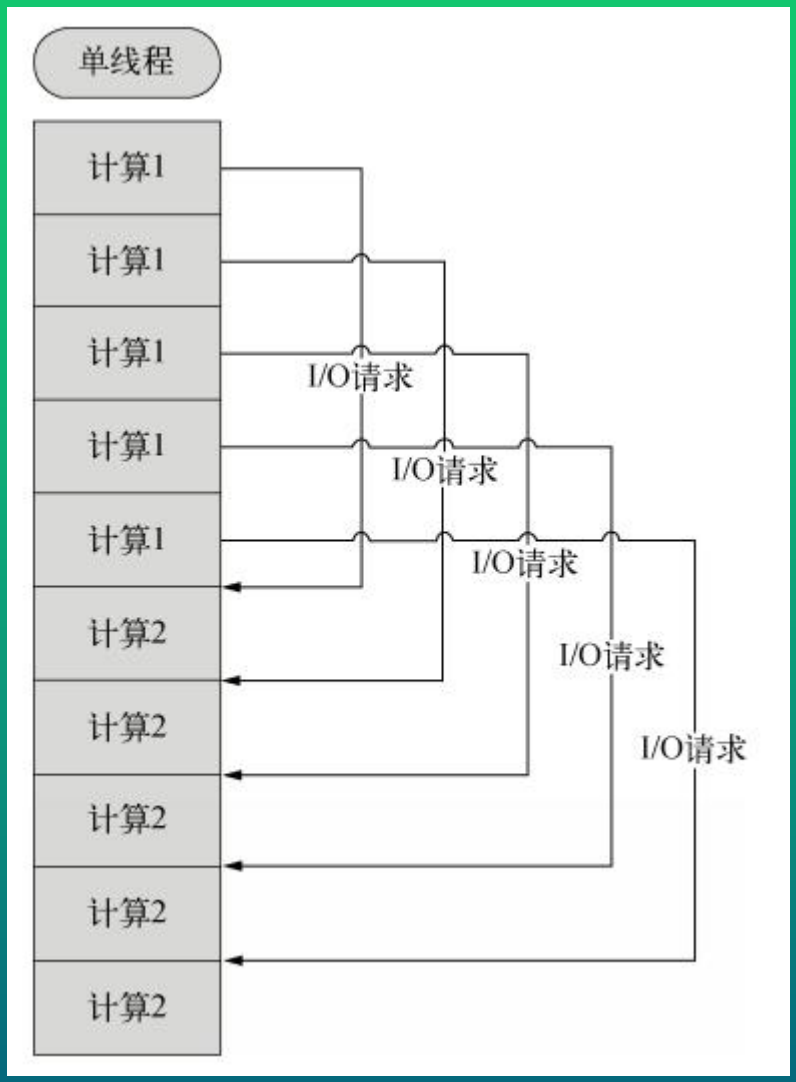

开始
Node.js/lib/fs.js
fs.open
Node.js/src/node_file.cc
Open
平台判断
libuv
Node.js/deps/uv/src/unix/fs.c
uv_fs_open
Node.js/deps/uv/src/win/fs.c
uv_fs_open
结束

# Node.js 单线程

- 在传统web 服务模型中，大多都使用多线程来解决并发的问题。而每一个客户端连接创建一个线程，需要耗费2MB的内存。也就是说。理论上一个8GB的服务器可以同时连接用户数为4000个左右。
- Node.js 使用一个线程（thread），利用非阻塞IO，事件驱动，理论上，一个8G内存的服务器，可以同时容纳3到4万用户的连接。

# Node.js 单线程



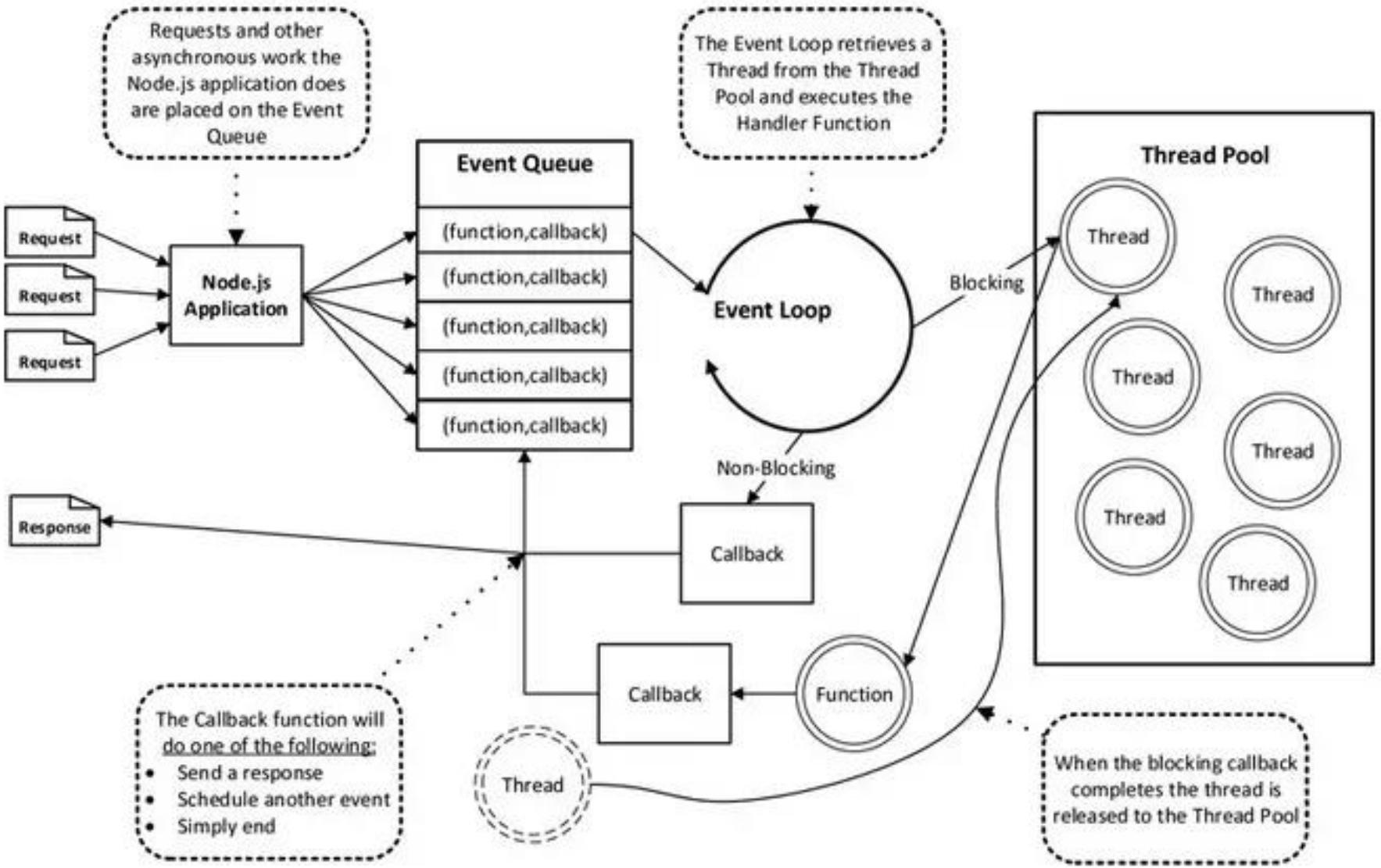



# 非阻塞I/O （non-blocking I/O）

# 非阻塞I/O （non-blocking I/O）

- 在传统的单线程处理机制中，I/O阻塞了代码的执行；
- Node.js中采用了非阻塞型I/O机制；
- 当某个I/O执行完毕时，将以事件的形式通知执行I/O操作的线程，线程执行这个事件的回调函数；
- 而非阻塞模式下，一个线程永远在执行计算操作，这个线程的CPU核心利用率永远是100%；

# 事件驱动/事件循环


Requests and other asynchronous work the Node.js application does are placed on the Event Queue
The Event Loop retrieves a Thread from the Thread Pool and executes the Handler Function
Event Queue
(function,callback)
(function,callback)
(function,callback)
(function,callback)
(function,callback)
Request
Request
Request
Node.js Application
Event Loop
Blocking
Non-Blocking
Thread Pool
Thread
Thread
Thread
Thread
Thread
Thread
Response
Callback
Callback
Function
Thread
The Callback function will do one of the following:
Send a response
Schedule another event
Simply end
When the blocking callback completes the thread is released to the Thread Pool

# 事件驱动/事件循环

- node.js单线程只是一个js主线程，本质上的异步操作还是由线程池完成的
- Nodejs之所以单线程可以处理高并发的原因，得益于libuv层的事件循环机制，和底层线程池实现

# 优缺点

- node的优点：I/O密集型处理是node的强项，因为node的I/O请求都是异步的（如：sql查询请求、文件流操作操作请求、http请求...）
- node的缺点：不擅长cpu密集型的操作

```
for (let i = 0; i < 1000000; i++) {
    console.log(i);
}
```

# 适用场景

- RESTful API: 请求和响应只需少量文本，并且不需要大量逻辑处理， 因此可以并发处理数万条连接。
- 聊天服务: 轻量级、高流量，没有复杂的计算逻辑。

# 下一节

基于koa2的前端工程化实践