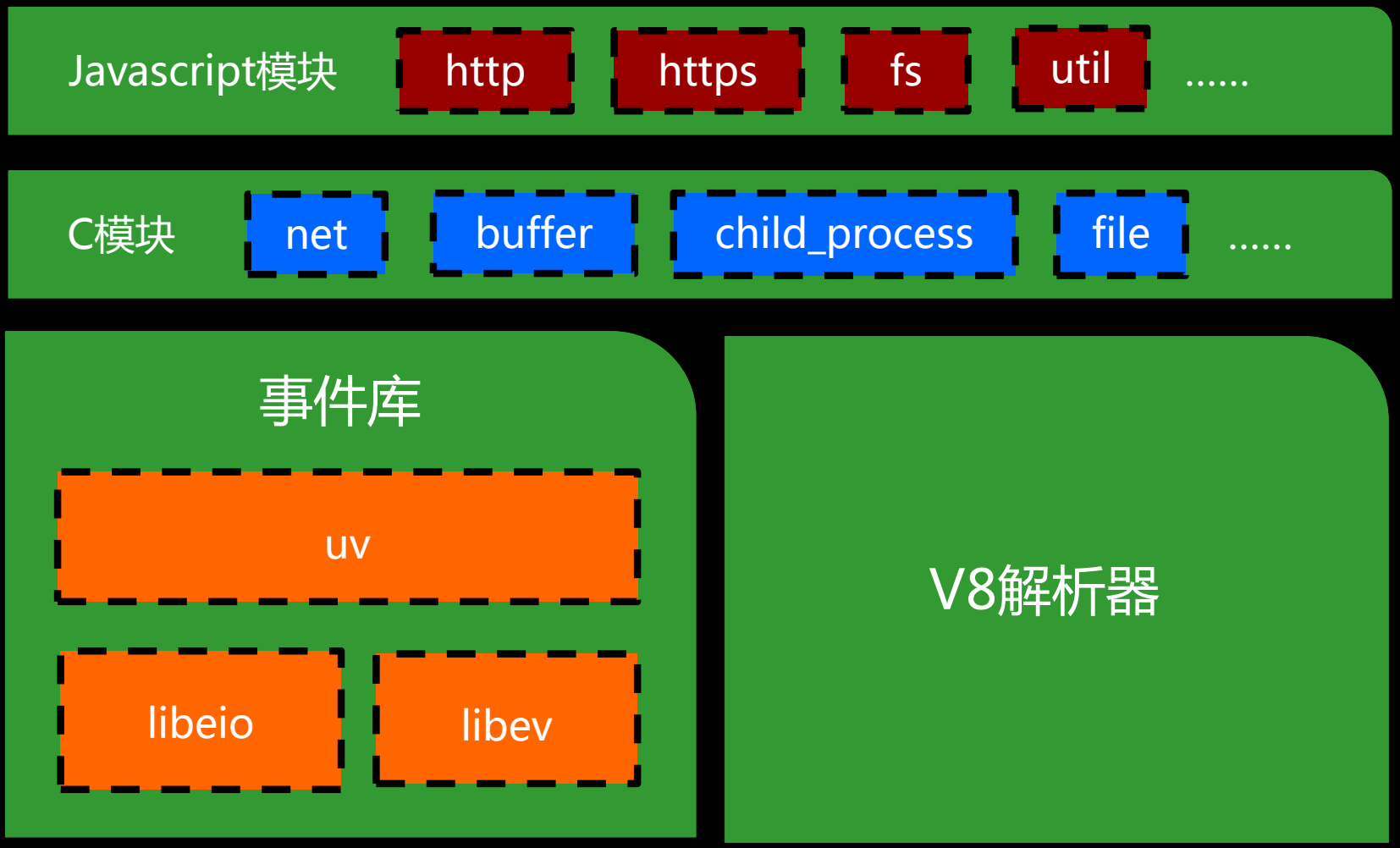
node.js异步：原理和缺陷

赵成
@小型笨蛋

node.js


Javascript模块
http
https
fs
util
.....
C模块
net
buffer
child_process
file
.....
事件库
uv
libeio
libev
V8解析器

主要内容

- 异步接口的实现
- uv，libev，以及libeio
- 存在的问题

异步的实现方式

- Synchronous I/O Multiplexing
  select，pselect，poll，epoll，kqueue，libev
- 线程模拟
  glibc aio，libeio
- Kernel Native AIO，以及Windows Overlapped I/O
  前者问题多多，比如仅支持 O_DIRECT 方式来对磁盘读写
  后者用于在uv中实现Windows的异步I/O

在Linux下，node.js靠libev和libeio配合使用来实现异步I/O

事件驱动的一个例子

```
var net = require('net');

var server = net.createServer(function (socket) {
  socket.write("Echo server\r\n");
  socket.pipe(socket);
});

server.listen(1337, "127.0.0.1");
```

```
static Handle<Value> Connect(const Arguments&) {
  ...
  uv_tcp_connect(..., AfterConnect);
  ...
}
```

uv

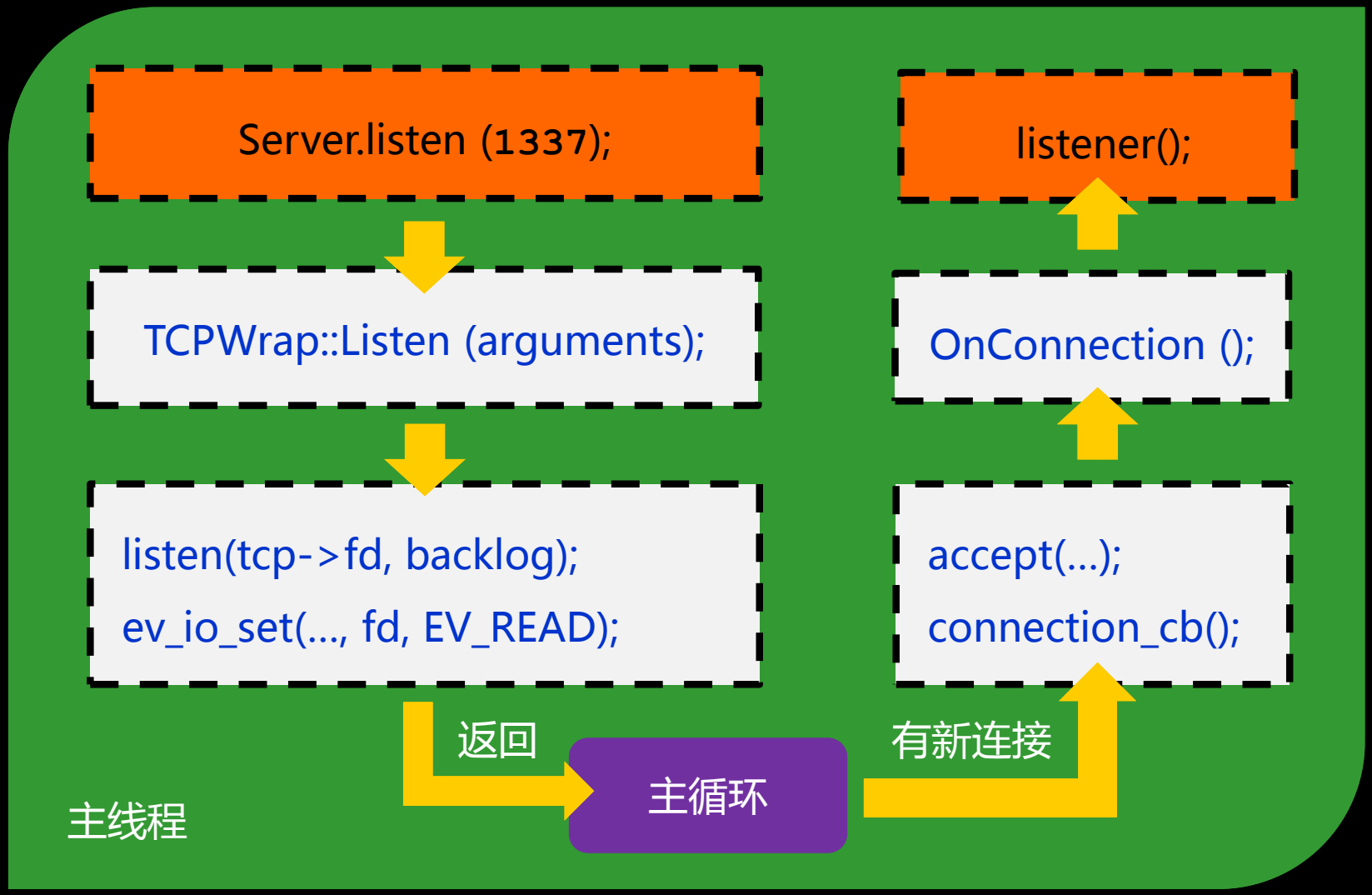
This is the new networking layer for Node. Its purpose is to abstract IOCP on windows and libev on Unix systems. We intend to eventually contain all platform differences in this library

- uv是node的网络实现层
- *nix下uv是对libev的封装
- 主要目的是实现Windows版本的node

什么是libev

- libev是一个事件驱动库，提供高性能事件循环
- 主要用于事件驱动的网络编程

调用流程


Server.listen (1337);
TCPWrap::Listen (arguments);
listen(tcp->fd, backlog);
ev_io_set(..., fd, EV_READ);
返回
主循环
主线程
有新连接
accept(...);
connection_cb();
OnConnection ();
listener();

另一个例子：fs.close (fd)

```
static Handle<Value> Close(const Arguments& args) {
  int fd = args[0]->Int32Value();

  if (args[1]->IsFunction()) {
    ASYNC_CALL(close, args[1], fd)
  }
}
```

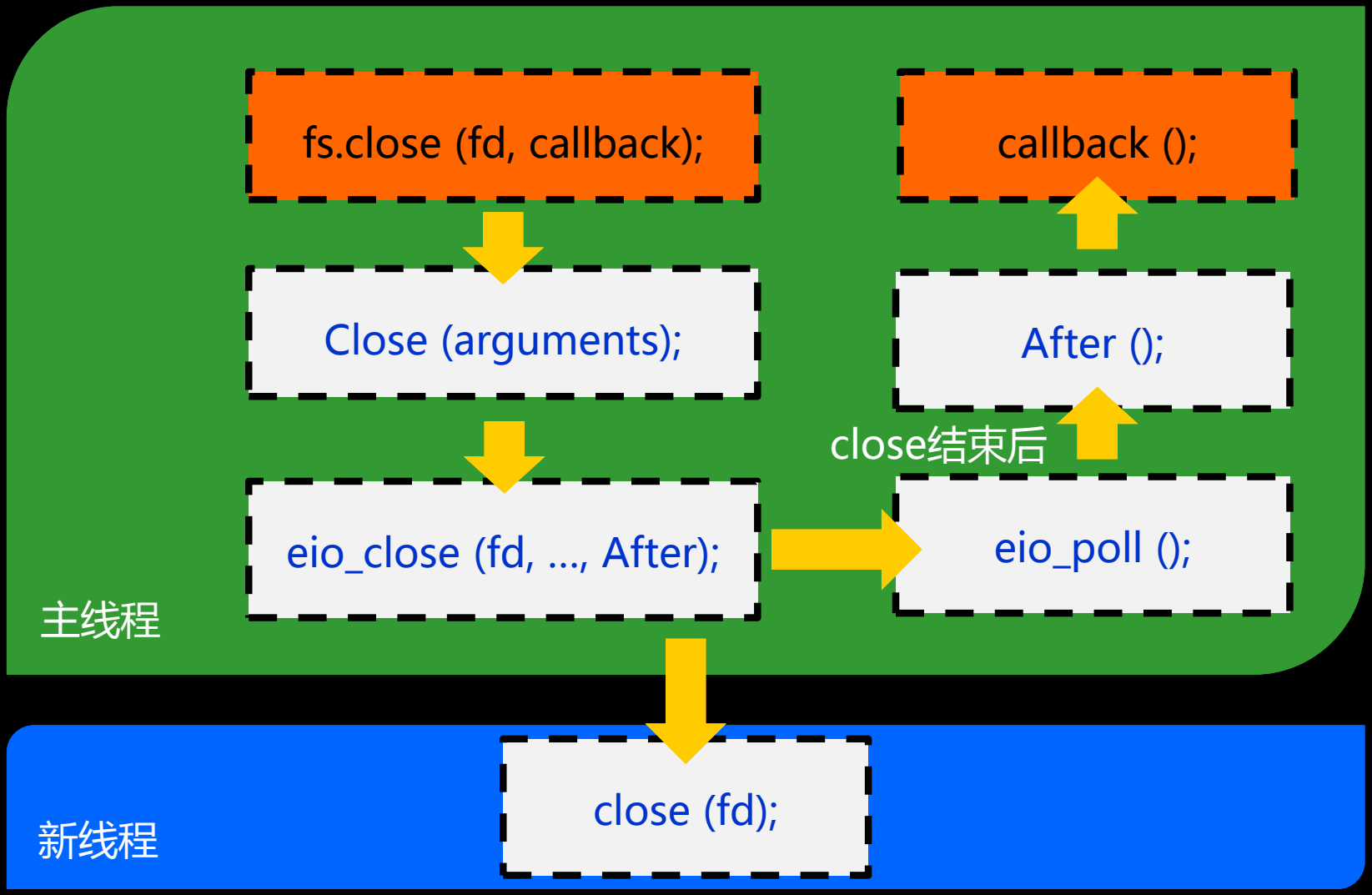
```
#define ASYNC_CALL(func, callback, ...) \
    eio_##func(After, cb_persist(callback));
```

什么是libeio

> Libeio is a full-featured asynchronous I/O library for C, modelled in similar style and spirit as libev. Features include: asynchronous read, write, open, close, stat, unlink, fdatasync, mknod, readdir etc.

- libeio为C提供异步版本的POSIX API
- 主要提供文件I/O操作
- 异步操作通过线程实现
- libeio仅依赖pthread，跨平台能力非常好
- 可以和任何事件库配合使用，比如libev

libeio做了什么


fs.close (fd, callback);
Close (arguments);
eio_close (fd, ..., After);
主线程
close (fd);
新线程
eio_poll ();
close结束后
After ();
callback ();

为什么不用libev实现异步文件操作？

@爱多

对于Regular File 来说，是不能够用采用 poll/epoll 的，即 O_NOBLOCK 方式对于传统文件句柄是无效的，也就是说我们的 open ,read, mkdir 之类的Regular File操作必定会导致阻塞

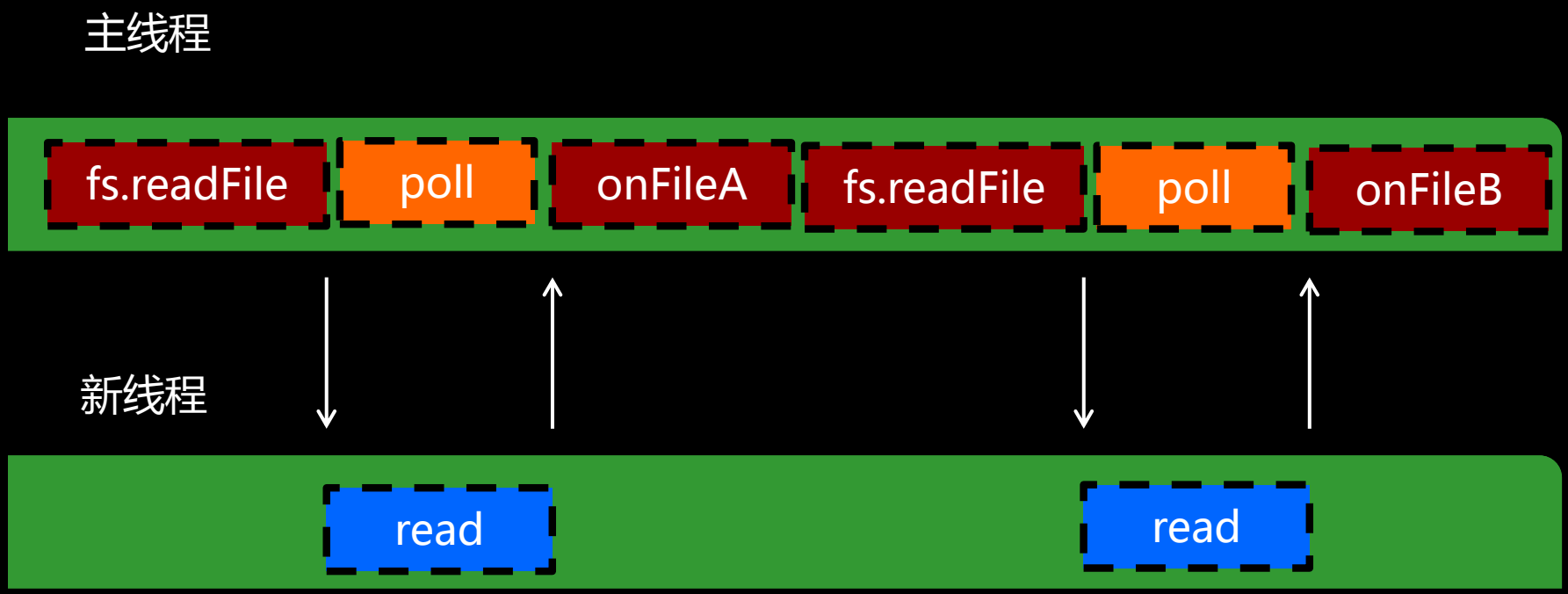
BAD CASE

```
function onFileB(err) {
  fs.readFile ("c");
}

function onFileA(err, data) {
  data += "blabla";
  fs.writeFile ("b", data, onFileB);
}

fs.readFile ("a", onFileA);
```

调用过程


主线程
fs.readFile
poll
onFileA
fs.readFile
poll
onFileB
新线程
read
read

代价

- 代码变得非常不直观
- 每个fs.readFile都发起一个线程
- 线程间context switch时代价很大
- 如果传递的是匿名函数，那么在执行前，函数的context会一直保存在内存中
  回调函数执行前资源（比如说数据库连接）不会被释放

BAD CASE 2

```
var conn = db.connect(...);
conn.query('SELECT * FROM table', function(err) {
  ...
  conn.close();
});
```

- conn.close();只能在主线程中执行
- conn.query();中有很多层嵌套时conn.close();将会推迟很久才执行
- 大量连接并发时会有很多数据库连接阻塞

Solution?

如果能够并行

```
var filename = “a”;
async (function(){
  var data = fs.readFileSync (filename);
  data += "blabla";
  fs.writeFileSync ("b", data);
  fs.readFileSync ("c");
});
console.log(“Main thread is in parallel”);
```

怎样实现async？

fork ();

```
if (fork () == 0) { /* child process */
   ...
} else { /* parent process */
   ...
}
```

- fork之后的父、子进程拥有完全相同的context
- 子进程的运行结果通过pipe传输

node-fork https://github.com/zcbenz/node-fork

使用fork的问题

- 新进程相对昂贵
- 进程间传递javascript对象需要额外开销
- 不能传递函数
- file destriptor会被复制，如果fork前程序已经在监听端口，子进程也将会监听同一个端口，产生竞争

V8引擎不允许多个线程同时使用V8的接口，所以不可能把多线程引入node

Thank you

赵成
@小型笨蛋
zcbenz@gmail.com