

# Lecture 9

## Node.js



Play



# Dynamic Vs. Static

## \* Static Page

- \* Client/Consumer's Viewpoint: an url refers to an identical html file
- \* Server/Producer's Viewpoint: a file stored within or sub-within the root folder of a Web Server
- \* it is a html ...
- \* Can be display directly at a browser

## \* Dynamic Page

- \* Client/Consumer's Viewpoint: an url refers to a dynamic html (maybe vary each time requested)
- \* Server/Producer's Viewpoint: a program/script produces html
- \* it is NOT a html, but a program producing html(s)
- \* Can't be display directly at a browser

## \* Dynamic Web Page, Dynamic HTML(DHTML), what's the difference?

# Development for the Web....

- ❖ Traditional desktop applications have a user interface wired up with background logic
  - \* user interfaces are based on Windows Forms, Jswing, WPF, Gtk, Qt, etc. and dependant on operating system
- ❖ On the web user interfaces are standardized
  - \* HTML for markup
  - \* CSS for style
  - \* JavaScript for dynamic content
- ❖ In the past proprietary technologies existed on the web, e.g. Adobe Flash, ActiveX
  - \* They are slowly dying
- ❖ Data is generated on the server, transferred to the client and displayed by the browser
- ❖ Server Side technologies include PHP, JSP, ASP.net, Rails, Django, and yes, also node.js

# Server-Side Web Programming

- ❖ server-side pages are programs written using one of many web programming languages/frameworks
  - \* examples: PHP, Java/JSP, Ruby on Rails, ASP.NET, Python, Perl
- ❖ the web server contains software that allows it to run those programs and send back their output as responses to web requests
- ❖ each language/framework has its pros and cons

# PHP vs. Node.js

❄️ JavaScript everywhere

❄️ [https://www.infoworld.com/article/3166109/  
application-development/php-vs-nodejs-an-epic-  
battle-for-developer-mind-share.html](https://www.infoworld.com/article/3166109/application-development/php-vs-nodejs-an-epic-battle-for-developer-mind-share.html)

# What is node.js?

- ❖ Asynchronous I/O framework
- ❖ Core in c++ on top of v8
- ❖ Rest of it in javascript
- ❖ Swiss army knife for network Related stuffs
- ❖ Can handle thousands of Concurrent connections with Minimal overhead (cpu/memory) on a single process
- ❖ It's NOT a web framework, and it's also NOT a language

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.

— from [nodejs.org](http://nodejs.org)

# Node.js的诞生

- ❄️ Created by Ryan Dahl in 2009
- ❄️ Development && maintenance sponsored by Joyent(一家位于硅谷的创业公司), 注册了“Node.js”这个商标, 使用其相关内容需要得到法律授权。
- ❄️ 2014, 开源社区io.js
- ❄️ 2015, Node.js 基金会, 支持社区驱动的开放管理模式
- ❄️ 2019, Node.js 基金会与 JS 基金会合并, 成立 OpenJS 基金会
- ❄️ Licence MIT

# Node.js 现状

- ✿ 2018 年 5 月 31 日，Node.js 基金会发布的用户调查报告，显示学习 Node.js 看起来更容易了，少于 2 年 node 经验的用户中，有 43% 的觉得“容易”。绝大多数（85%）Node.js 用户用于网页开发，43% 参与一些企业级开发，13% 用于大数据分析，8% 用于嵌入式系统。



# Node.js 应用场景

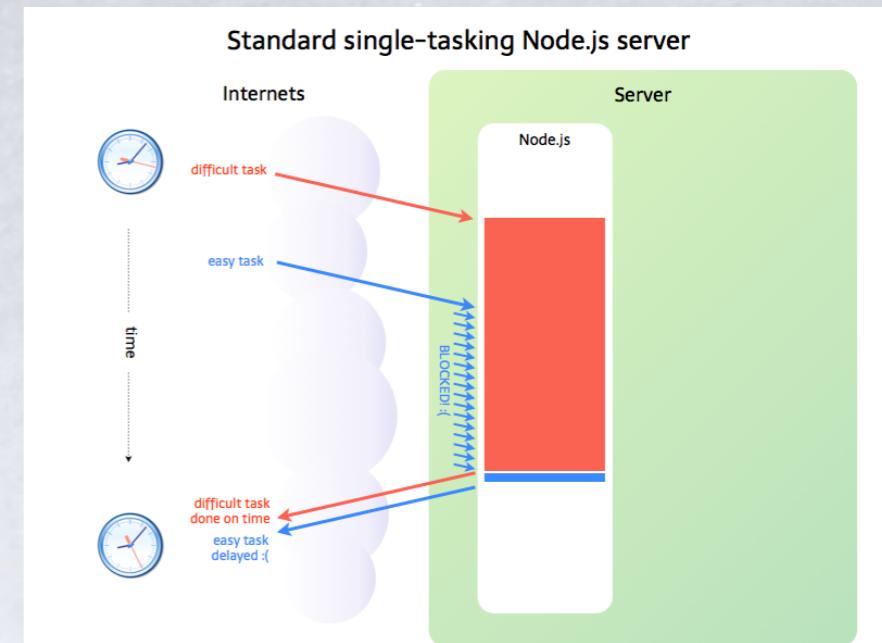
- ❄ 网站 (如express/koa等)
- ❄ im即时聊天(socket.io)
- ❄ api (移动端, pc, h5)
- ❄ HTTP Proxy (淘宝、Qunar、腾讯、百度都有)
- ❄ 前端构建工具(grunt/gulp/bower/webpack/fis3...)
- ❄ 写操作系统 (NodeOS)
- ❄ 跨平台打包工具 (PC端的electron、nw.js, 比如钉钉PC客户端、微信小程序IDE、微信客户端, 移动的cordova, 即老的Phonegap, 还有更加有名的一站式开发框架ionicframework)
- ❄ 命令行工具 (比如cordova、shell.js)
- ❄ 反向代理 (比如anyproxy, node-http-proxy)

# 何时使用

- ❄️ Node所针对的应用程序有一个专门的简称：DIRT，表示数据密集型实时（data-intensive real-time）程序。
- ❄️ 性能和I/O负载：Nodejs非常好的解决了IO密集的问题，通过异步IO来实现。
- ❄️ 大前端的基石。
- ❄️ 从脚手架、辅助前端开发（比如 SSR、PWA 等）的快速开发实践，到 API 中间层、代理层，到专业的后端开发都有非常成熟的经验。
- ❄️ 全栈。

# Nodejs不适合的领域

- ❄ 计算密集型应用
- ❄ 内存控制
- ❄ 大内存的应用，由于V8引擎有内存设计的限制，32位环境中最大堆是1G，64位环境中最大堆也不到2G，如果要一次读入10G数据，对于Nodejs来说也无法实现。
- ❄ 静态服务器，虽然Nodejs的优势在IO密集型应用，但是和Nginx的处理静态资源还是有很大的差距。
- ❄ 不需要异步的应用：比如系统管理，自动化脚本等，Nodejs的异步调用可能会给编程带来一些麻烦。



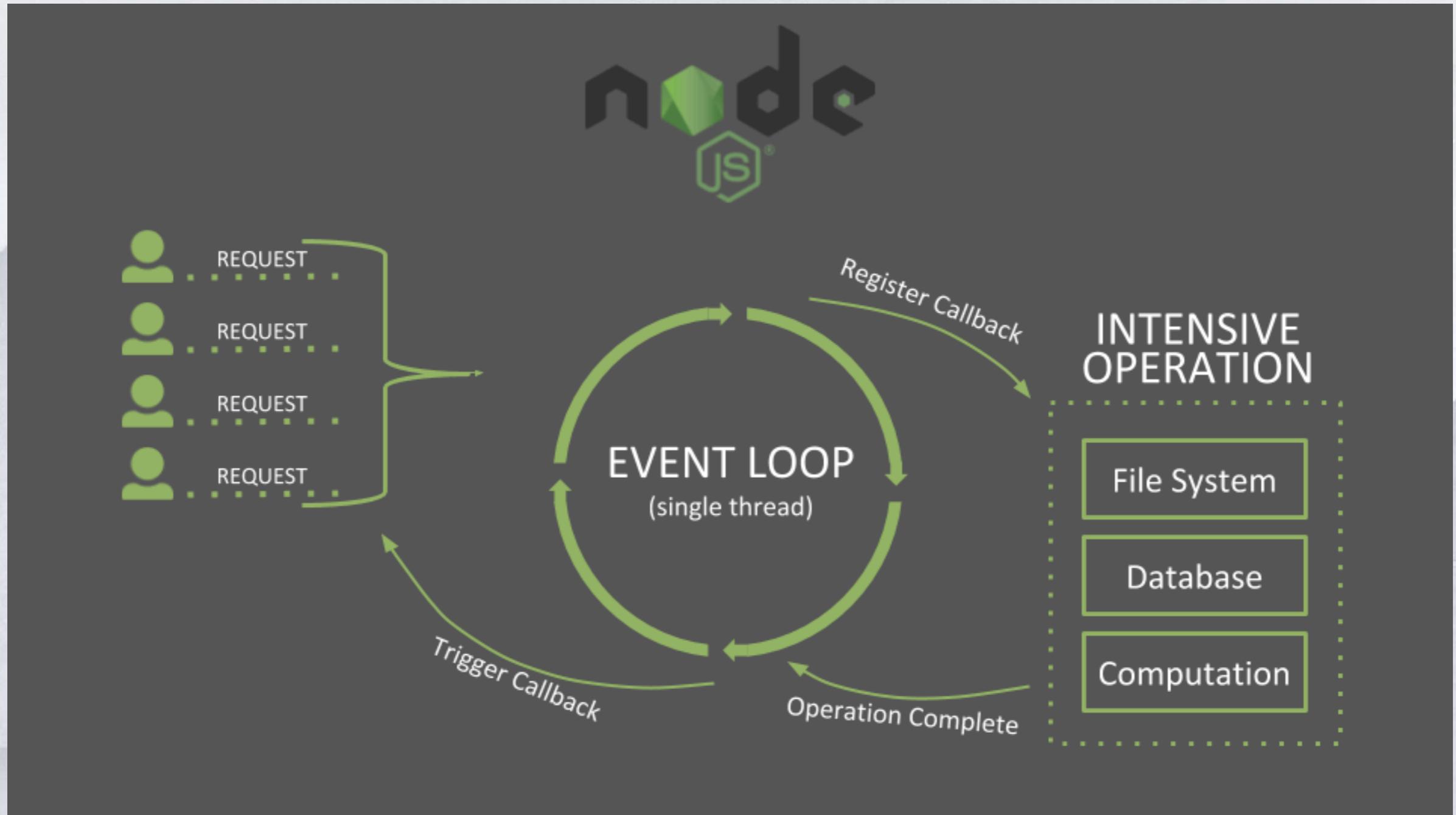
# Why node.js ?

- ❄ Non Blocking I/O
- ❄ V8 Javascript Engine
- ❄ Single Thread with Event Loop
- ❄ 超过100万 modules
- ❄ Windows, Linux, Mac
- ❄ 1 Language for Frontend and Backend
- ❄ Active community

# The idea behind node.js....

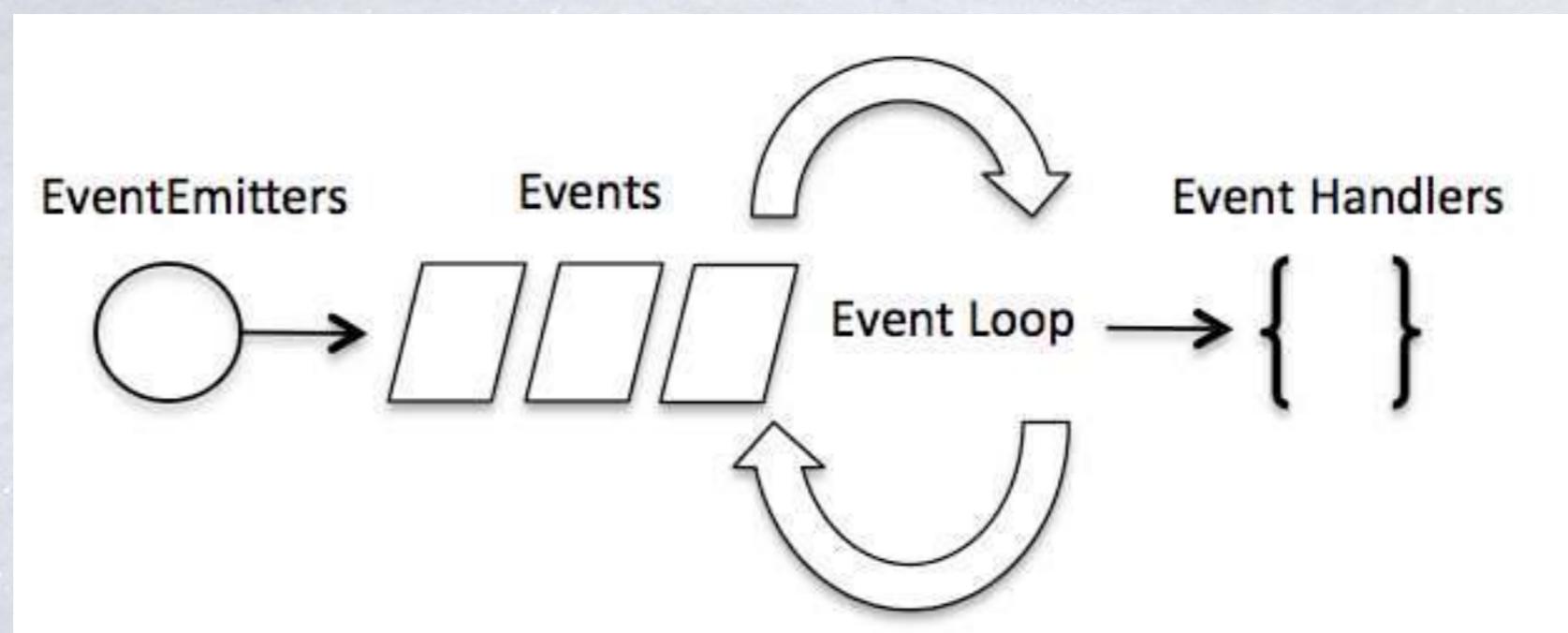
- ❖ Perform asynchronous processing on single thread instead of classical multithread processing, minimize overhead & latency, maximize scalability
- ❖ Scale horizontally instead of vertically
- ❖ Ideal for applications that serve a lot of requests but don't use/need lots of computational power per request
- ❖ Not so ideal for heavy calculations, e.g. massive parallel computing
- ❖ Also: Less problems with concurrency

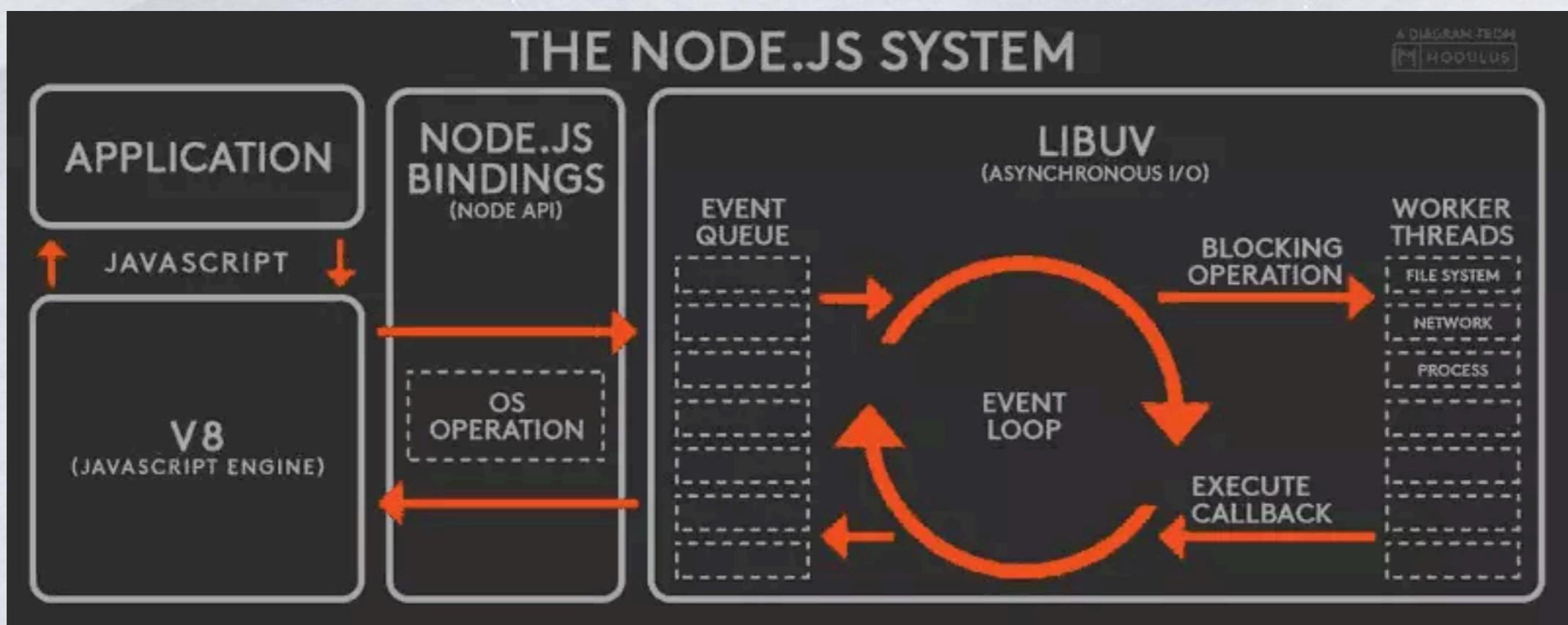
# Node.js Event Loop



There are a couple of implications of this apparently very simple and basic model  
Avoid synchronous code at all costs because it blocks the event loop  
Which means: callbacks, callbacks, and more callbacks

# 事件驱动模型





Javascript

C/C++

## Node standard library

http,net,stream,fs,events,buffer...

## Node bindings

**V8**

Javascript VM

**libuv**

Thread pool  
Event pool  
Async I/O

**C-ares**

Async DNS

http\_parser  
OpenSSL  
zlib,etc...

# libuv

## Network I/O

TCP

UDP

TTY

Pipe

...

File  
I/O

DNS  
Ops.

User  
code

`uv__io_t`

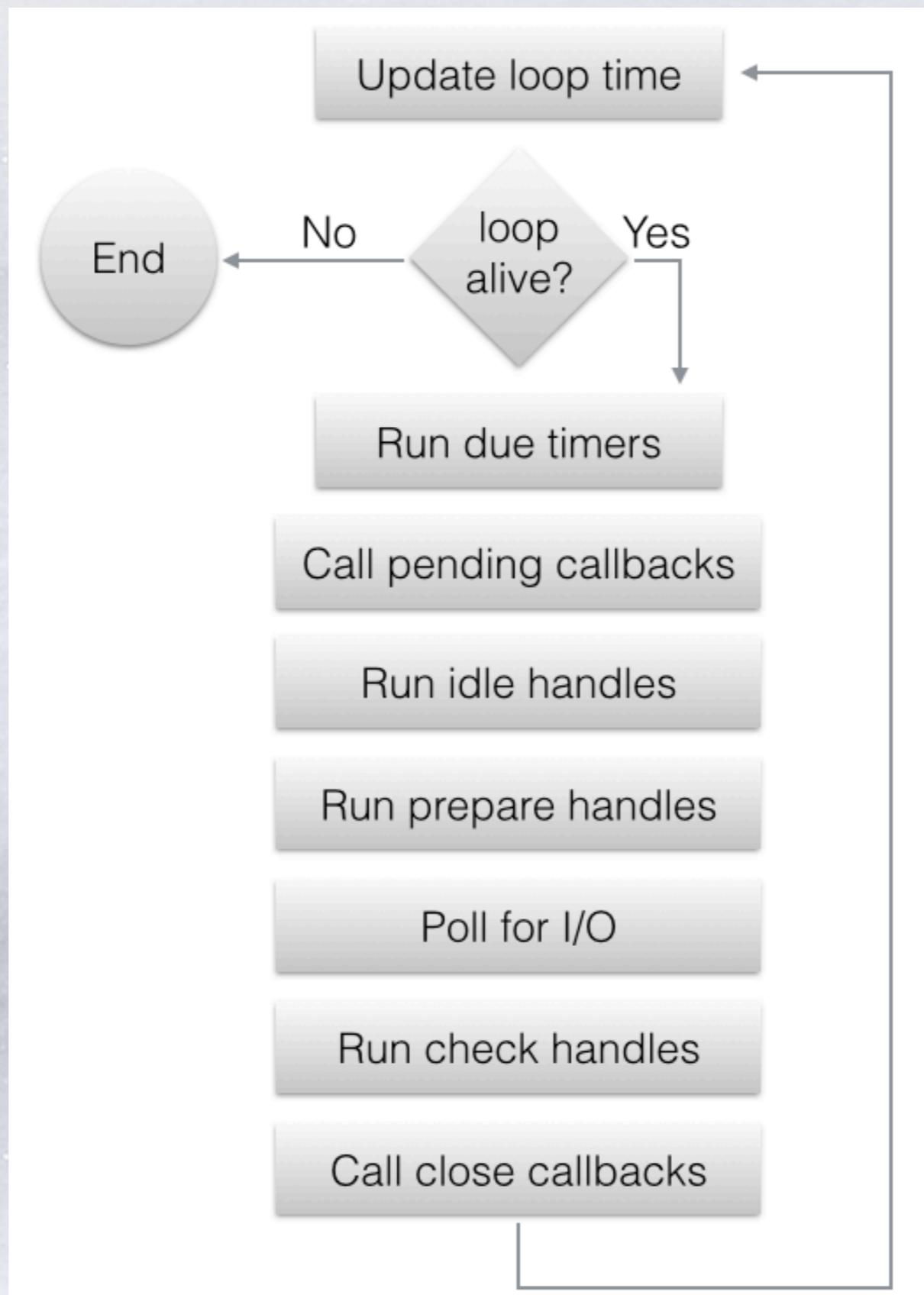
epoll

kqueue

event ports

IOCP

Thread Pool



# Node.js 支持 JavaScript。

❄️ 这是 Node.js 能够发展壮大的一个非常重要的间接原因。

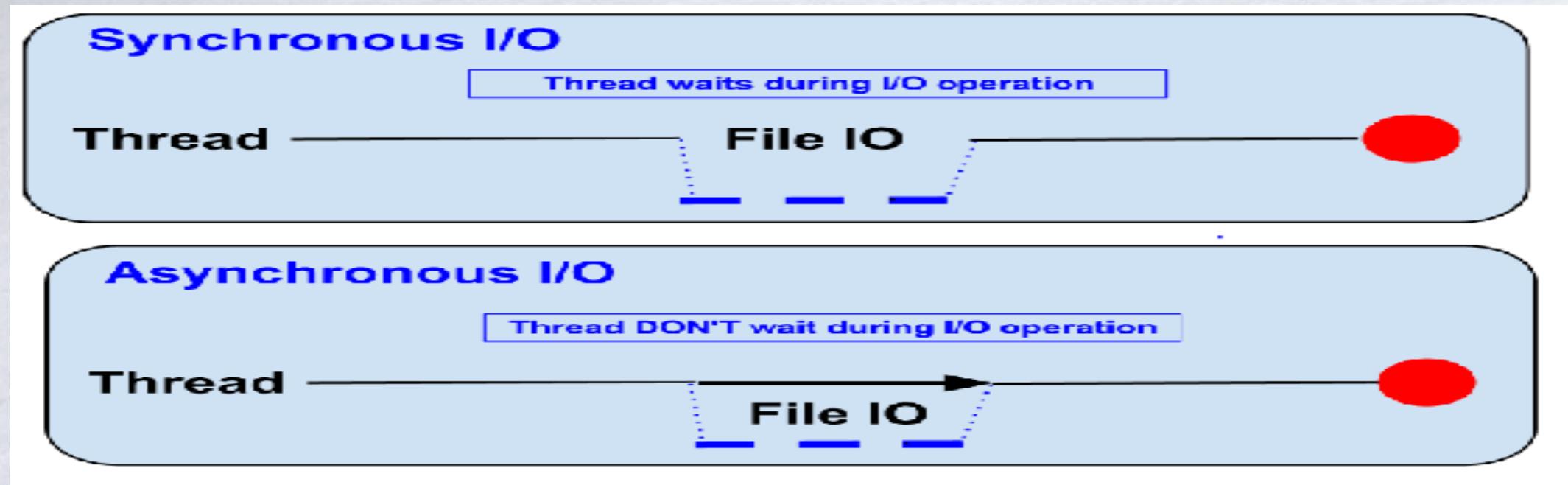
- \* 首先，Javascript 作为前端工程师的主力语言，在技术社区中有相当的号召力。而且，随着 Web 技术的不断发展，特别是前端的重要性增加，不少前端工程师开始试水“后台应用”，在许多采用 Node.js 的企业中，工程师都表示因为习惯了 Javascript，所以选择 Node.js。
- \* 其次，Javascript 的匿名函数和闭包特性非常适合事件驱动、异步编程。
- \* 有 Google V8 引擎的加持，Node.js 的性能也是受益其中。

# Synchronous vs Asynchronous

Synchronous	Asynchronous
Waits for each operation to complete, after that executes next operation.	Never waits each operation to complete, executes all at one time. Results handled as available.
Step by Step execution	Callbacks used to handle results

# Blocking vs Non-Blocking.....

\* Example :: Read data from file and show data



# 同步式 I/O 和异步式 I/O 的特点

同步式 I/O(阻塞式)	异步式 I/O(非阻塞式)
利用多线程提供吞吐量	单线程即可实现高吞吐量
通过事件片分割和线程调度利用多核CPU	通过功能划分利用多核CPU
需要由操作系统调度多线程使用多核 CPU	可以将单进程绑定到单核 CPU
难以充分利用 CPU 资源	可以充分利用 CPU 资源
内存轨迹大， 数据局部性弱	内存轨迹小， 数据局部性强
符合线性的编程思维	不符合传统编程思维

# Blocking.....

- ❖ • Read data from file
- ❖ • Show data
- ❖ • Do other tasks
- ❖

```
var data = fs.readFileSync( "test.txt" );
console.log( data );
console.log( "Do other tasks" );
```

# Non-Blocking.....

- ❖ Read data from file
  - \* When read data completed, show data
- ❖ Do other tasks

```
fs.readFile( "test.txt", function( err, data ) {  
  console.log(data);  
});
```

Callback

# Getting Started.....

✿ <https://nodejs.org/en/download/>

LTS Recommended For Most Users	Current Latest Features
 Windows Installer node-v8.9.1-x86.msi	 Macintosh Installer node-v8.9.1.pkg
<a href="#">Windows Installer (.msi)</a>	32-bit      64-bit
<a href="#">Windows Binary (.zip)</a>	32-bit      64-bit
<a href="#">macOS Installer (.pkg)</a>	64-bit
<a href="#">macOS Binaries (.tar.gz)</a>	64-bit
<a href="#">Linux Binaries (x86/x64)</a>	32-bit      64-bit
<a href="#">Linux Binaries (ARM)</a>	ARMv6      ARMv7      ARMv8
<a href="#">Source Code</a>	node-v8.9.1.tar.gz
Additional Platforms	
<a href="#">SunOS Binaries</a>	32-bit      64-bit
<a href="#">Docker Image</a>	Official Node.js Docker Image
<a href="#">Linux on Power Systems</a>	64-bit le
<a href="#">Linux on System z</a>	64-bit
<a href="#">AIX on Power Systems</a>	64-bit

# Node Package Manager (NPM)

 [www.npmjs.org](http://www.npmjs.org)

 1450.000+ modules

 package.json file (npm init)

 npm install <module.name> --save



# File package.json.....

## Project informations

- \* **name** 项目名称
- \* **version** 项目的版本号
- \* **description** 项目的描述信息
- \* **entry point** 项目的入口文件
- \* **test command** 项目启动时脚本命令
- \* **git repository** 如果你有 Git 地址，可以将这个项目放到你的 Git 仓库里
- \* **keywords** 关键词
- \* **author** 作者
- \* **license** 项目要发行的时候需要的证书
- \*

# Install module.....

 \$npm install <module name>

# In NodeJS

Standard JavaScript with

- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Console
- Crypto
- Debugger
- DNS
- Domain
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- VM
- ZLIB

... but without DOM manipulation

# Node.js流行模块

- ✿ Express.js 由核心 Node 项目团队的成员之一 TJ Holowaychuk 构建。大型社区支持此框架，因此具有不断更新和改革所有核心功能的优势。这是一个极简主义的框架，用于构建 mobile 应用程序和 API。
- ✿ Koa 由创建 Express.js 的同一团队开发，通常被称为下一代 NodeJS 框架。Koa 的独特之处在于它使用了一些非常酷的 ECMAScript（ES6）方法，使你无需回调即可工作，同时极大地扩展了错误处理。
- ✿ Hapi 是一个强大且健壮的框架，用于开发API。完善的插件系统和各种关键功能（例如输入验证、基于配置的功能、实现缓存、错误处理、日志记录等）使 Hapi 成为最受欢迎的框架之一。它用于构建有用的应用，并通为 PayPal, Disney 等多个大型网站提供技术解决方案。Hapi 以最小的开销构建安全、强大、可扩展的开箱即用的功能。
- ✿ Socket.io 用于构建实时 Web 应用。这是一个 Javascript 库，可在 Web 客户端和服务器之间进行双向数据通信。异步数据 I/O、二进制流和即时消息传递是此框架最重要的功能。
- ✿ Meteor.JS 是最常用的 NodeJS 框架之一。NodeJS 的全栈框架，允许用户构建实时应用程序。它用于创建基于移动和基于 Web 的 javascript 应用。

# 模块

## ❄ 两种模块，不兼容

- \* ES6 模块，简称 ESM
- \* Node.js 专用的 CommonJS 模块，简称 CJS

# ES6 模块与 CommonJS 模块的差异

## 语法

- \* CommonJS 模块使用 `require()` 加载和 `module.exports` 输出
- \* ES6 模块使用 `import` 和 `export`

\*

```
var http = require('http');
var fs = require('fs');
var express = require('express');
```

```
// profile.js
var firstName = 'Michael';
var lastName = 'Jackson';
var year = 1958;

export {firstName, lastName, year};

// main.js
import {firstName, lastName, year} from './profile';
console.log(firstName, lastName) // Michael Jackson
```

# ES6 模块与 CommonJS 模块的差异

- \* CommonJS 模块输出的是一个值的拷贝，ES6 模块输出的是值的引用。
- \* CommonJS 模块是运行时加载，ES6 模块是编译时输出接口。
  - \* CommonJS 加载的是一个对象（即`module.exports`属性），该对象只有在脚本运行完才会生成。
  - \* ES6 模块不是对象，它的对外接口只是一种静态定义，在代码静态解析阶段就会生成。
- \*

# 模块加载的实质

```
// lib.js
var counter = 3;
function incCounter() {
    counter++;
}
module.exports = {
    counter: counter,
    incCounter: incCounter,
};

// main.js
var mod = require('./lib');

console.log(mod.counter); // 3
mod.incCounter();
console.log(mod.counter); // 3
```

# ES6

```
// lib.mjs
export let counter = 3;
export function incCounter() {
  counter++;
}

// main.mjs
import { counter, incCounter } from './lib.mjs';
console.log(counter); // 3
incCounter();
console.log(counter); // 4
```

# Node.js 的区分

## ❄ Node.js 要求 ES6 模块采用.mjs后缀文件名。

- \* 只要脚本文件里面使用import或者export命令，那么就必须采用.mjs后缀名。Node.js 遇到.mjs文件，就认为是 ES6 模块，默认启用严格模式，不必在每个模块文件顶部指定"use strict"。
- \* 如果不希望将后缀名改成.mjs，可以在项目的package.json文件中，指定type字段为module。

## ❄ .cjs文件总是以 CommonJS 模块加载

- \* 如果没有type字段，或者type字段为commonjs，则.js脚本会被解释成CommonJS 模块。

## ❄ .js文件的加载取决于package.json里面type字段的设置。

## ❄ 注意，ES6 模块与 CommonJS 模块尽量不要混用！！！

- \* require命令不能加载.mjs文件，会报错，只有import命令才可以加载.mjs文件。
- \* 反之，.mjs文件里面也不能使用require命令，必须使用import。

# 同时支持两种格式的模块

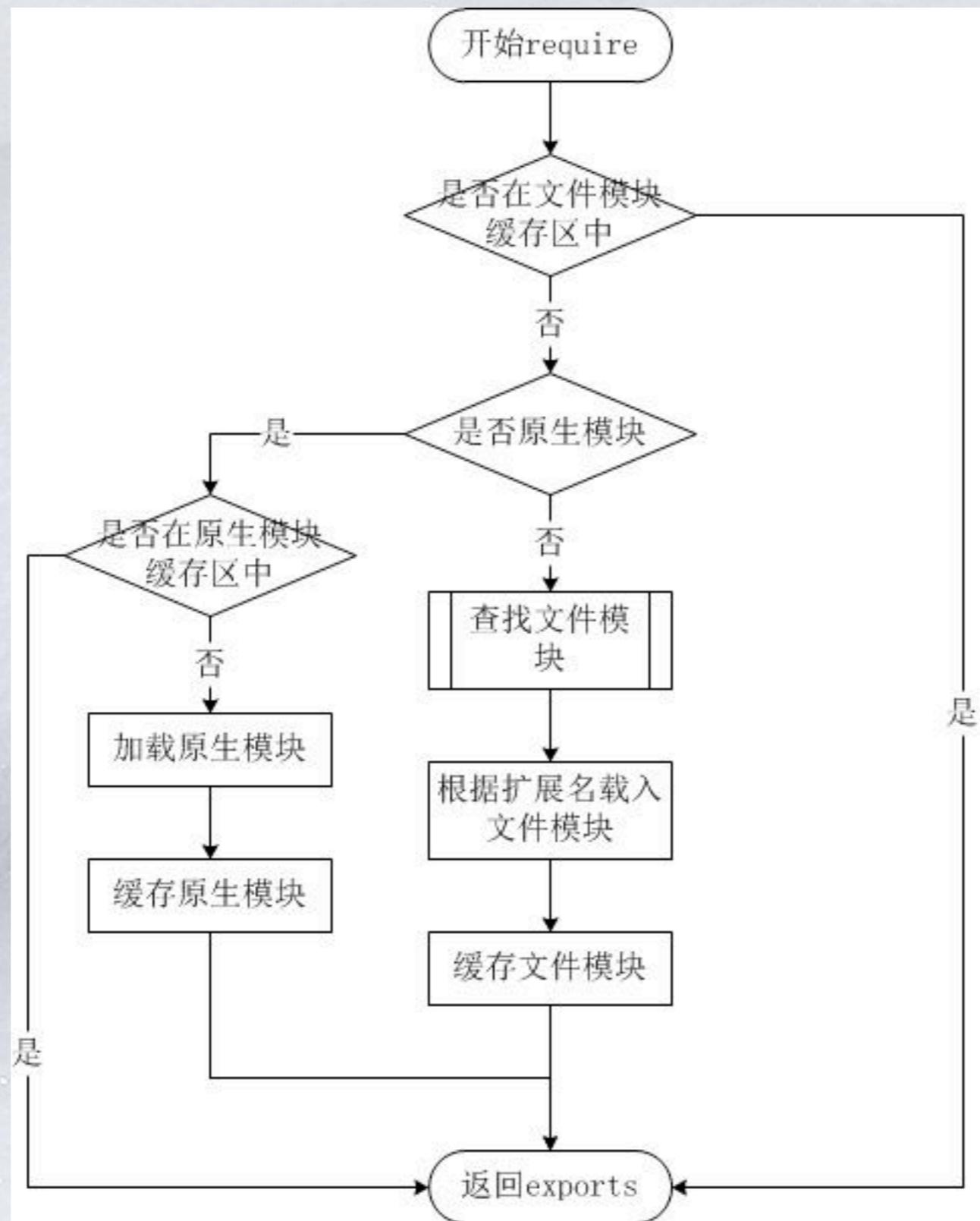
- ❄️ 如果原始模块是 ES6 格式，那么需要给出一个整体输出接口，比如 `export default obj`，使得 CommonJS 可以用`import()`进行加载。
- ❄️ 如果原始模块是 CommonJS 格式，那么可以加一个包装层。

```
import cjsModule from '../index.js';
export const foo = cjsModule.foo;
```

- ❄️ 可以把这个文件的后缀名改为`.mjs`，或者放在一个子目录，再在这个子目录里面放一个单独的`package.json`文件，指明`{ type: "module" }`。
- ❄️ 另一种做法是在`package.json`文件的`exports`字段，指明两种格式模块各自的加载入口。

```
"exports": {
  "require": "./index.js",
  "import": "./esm/wrapper.js"
}
```

# require方法查找策略



# A simple file server

```
var http = require("http");

var server = http.createServer(function(req, res){
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n')
});

server.listen(3000);

console.log('Server running at http://localhost:3000/');
```

node server.js

```
sudo npm install -g supervisor
supervisor server.js
```

# as

```
//main.js
import { lastName as surname } from './profile';
console.log(surname); // Jackson
```

```
//profile.js
export {firstName as name}
```

# export default

```
//default.js
function add(a, b){
  return a + b;
}
export default add;
// 实际上
export {add as default};

// main.js
import add from './default'
//实际上 add 名字可以随便起
import {default as add} from './default'
```

# 循环加载

```
//a.js
exports.done = false;
var b = require('./b.js');
console.log('在 a.js 之中, b.done = %j', b.done);
exports.done = true;
console.log('a.js 执行完毕');
```

```
//b.js
exports.done = false;
var a = require('./a.js');
console.log('在 b.js 之中, a.done = %j', a.done);
exports.done = true;
console.log('b.js 执行完毕');
```

```
//main.js
var a = require('./a.js');
var b = require('./b.js');
console.log('在 main.js 之中, a.done=%j, b.done=%j',
a.done, b.done);
```

在 b.js 之中, a.done = false  
b.js 执行完毕  
在 a.js 之中, b.done = true  
a.js 执行完毕  
在 main.js 之中, a.done=true, b.done=true

# ES6 循环加载-1

```
// a.mjs如下
import {bar} from './b.mjs';
console.log('a.mjs');
console.log(bar);
export let foo = 'foo';

// b.mjs
import {foo} from './a.mjs';
console.log('b.mjs');
console.log(foo);
export let bar = 'bar';
```

```
$ node --experimental-modules a.mjs
b.mjs
ReferenceError: Cannot access 'foo' before
initialization
```

# ES6 循环加载-2

```
// a.mjs
import {bar} from './b.mjs';
console.log('a.mjs');
console.log(bar());
function foo() { return 'foo' }
export {foo};
```

```
// b.mjs
import {foo} from './a.mjs';
console.log('b.mjs');
console.log(foo());
function bar() { return 'bar' }
export {bar};
```

```
$ node --experimental-modules a.mjs
b.mjs
foo
a.mjs
bar
```

# Nodejs版本

## \* 版本帝

### \* LTS和Current其实并不是版本，而是同一个主版本号的不同阶段

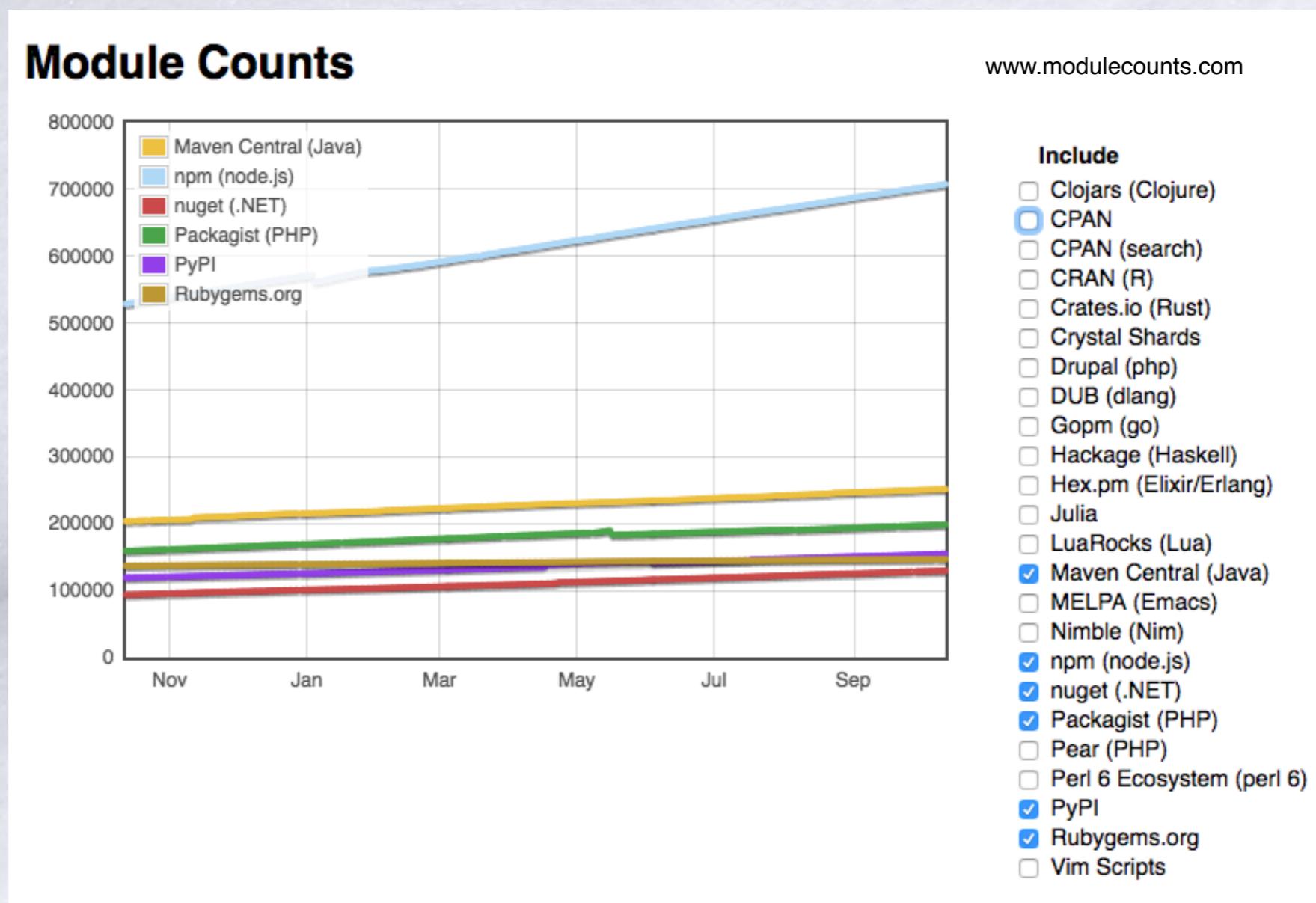
- \* LTS: Long Term Support。该版本进入了漫长的维护期。它又分为两个阶段：Active LTS和Maintenance LTS。从以往的发布历史看，LTS至少会被跟进2年时间，按照最新的官方网站的说法，Active LTS持续12个月，Maintenance LTS将会被持续维护18个月的时间。Nodejs 12之前，active阶段持续18个月，maintenance阶段持续12个月。
- \* Current: 一个新主版本号release后，先进入Current阶段，该阶段持续6个月，目的是给各个库(library)的作者时间来支持新版。偶数版本在Current阶段后进入LTS阶段，而奇数版本则终结不再维护

## \* 奇偶版本号

- \* Nodejs主版本号 (semver-major) 奇数版本和偶数版本有不同的生命周期。
- \* 每隔6个月，社区会从Nodejs master分支拉出一个分支作为主版本的release。偶数版本在4月发版，奇数版本则在十月。
- \* 奇数版本发版时，上一个偶数版本会进入LTS阶段，而奇数版本则只持续6个月的时间，则终结不再维护。

# 已无性能优势?

- ❄ 实现成本
- ❄ 调优成本
- ❄ 学习成本



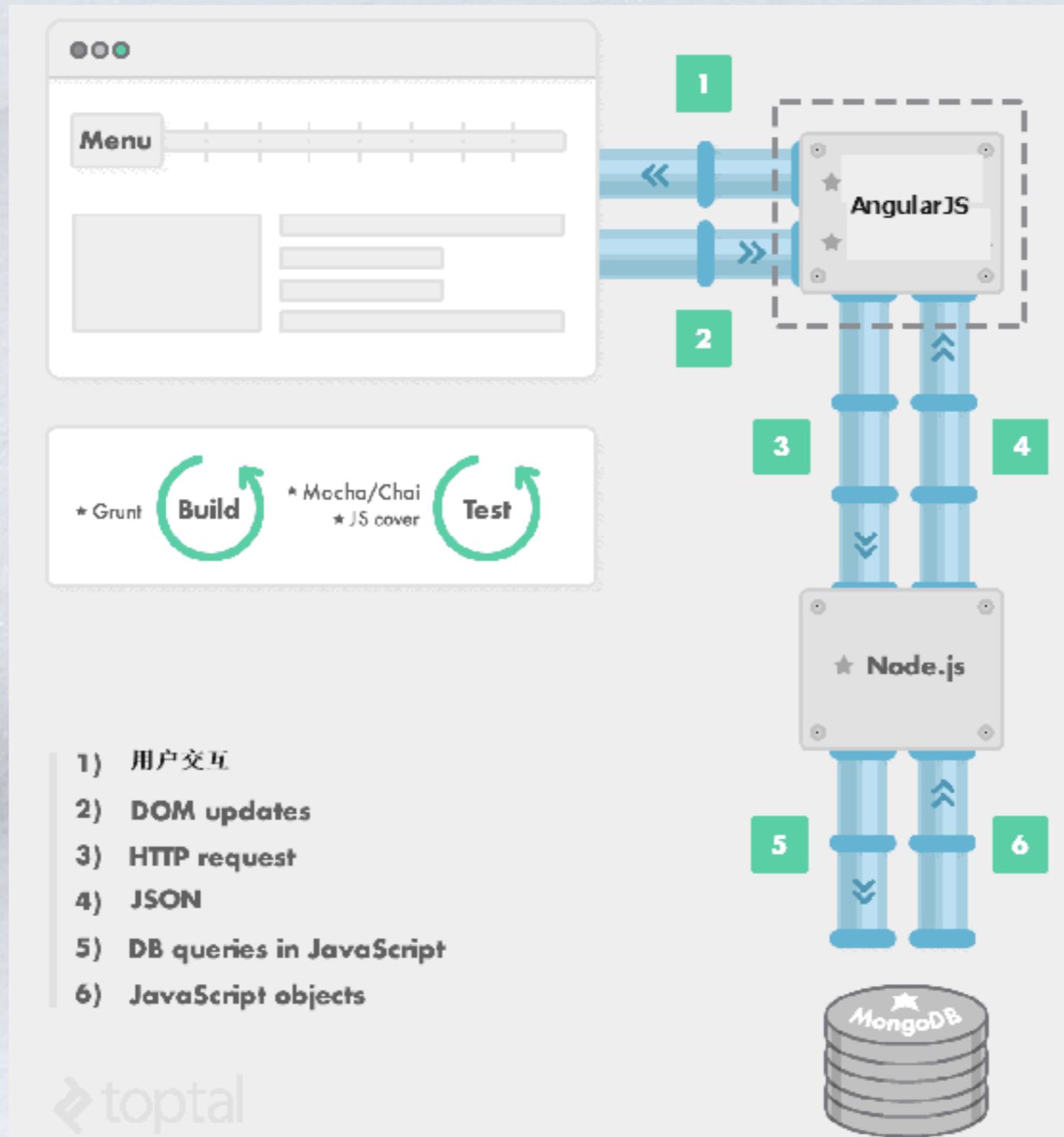
# MEAN

MEAN是一个Javascript平台的现代Web开发框架总称

- \* MongoDB是一个使用JSON风格存储的数据库，非常适合javascript。(JSON是JS数据格式)
- \* ExpressJS是一个Web应用框架，提供有帮助的组件和模块帮助建立一个网站应用。
- \* AngularJS是一个前端MVC框架。
- \* Node.js是一个并发 异步 事件驱动的Javascript服务器后端开发平台。



# MEAN架构原理



# Web框架

框架名称	特性	点评
Express	简单、实用，路由中间件等五脏俱全	最著名的Web框架
Derby.js && Meteor	同构	前后端都放到一起，模糊了开发便捷，看上去更简单，实际上对开发来说要求更高
Sails、Total	面向其他语言，Ruby、PHP等	借鉴业界优秀实现，也是Node.js成熟的一个标志
MEAN.js	面向架构	类似于脚手架，又期望同构，结果只是蹭了热点
Hapi和Restfy	面向Api && 微服务	移动互联网时代Api的作用被放大，故而独立分类。尤其是对于微服务开发更是利器
ThinkJS	面向新特性	借鉴ThinkPHP，并慢慢走出自己的一条路，对于Async函数等新特性支持，无出其右
Koa	专注于异步流程改进	下一代Web框架

# NodeJS 框架的优势

现在 NodeJS 框架正在成为最常用的构建 Web 应用前端的开发框架。这是自定义 Web 开发的首选环境。让我们检查一些主要的NodeJS框架的优点：

- \* 实时工作环境
- \* 简单的编码经验
- \* 无缝数据流
- \* 在整个开发过程中使用相同的代码模式
- \* 方便易用的



# 框架选型

- ❄ 业务场景、特点
- ❄ 自身团队能力、喜好，有时候技术选型决定团队氛围的，需要平衡激进与稳定
- ❄ 熟悉程度
- ❄ 个人学习求新，企业架构求稳，无非喜好与场景而已

# 预处理器

名称	实现	描述
模板引擎	art\mustache\ejs\hbs\jade ...	上百种之多，自定义默认，编译成html，继而完成更多操作
css预处理器	less\sass\scss\rework\postcss	自定义语法规则，编译成css
js友好语言	coffeescript、typescript	自定义语法规则、编译成js

# 跨平台

平台	实现	点评
Web/ H5	纯前端	
PC客 户端	nw.js和electron	尤其是atom和vscode编辑器最为著名，像钉钉PC端，微信客户端、微信小程序IDE等都是这样的，通过web技术来打包成PC客户端
移动端	cordova（旧称PhoneGap），基于cordova的ionicframework	这种采用h5开发，打包成ipa或apk的应用，称为Hybrid开发（混搭），通过webview实现所谓的跨平台，应用的还是非常广泛的

# 其他

用途	说明	前景
爬虫	抢了不少Python的份额，整体来说简单，实用	看涨
命令行工具	写工具、提高效率，node+npm	看涨
微服务与RPC	Node做纯后端不好做，但在新项目和微服务架构下，必有一席之地	看涨
微信公众号开发	已经火了2年多了，尤其是付费阅读领域，还会继续火下去，gitchat就是使用Node.js做的	看涨
反向代理	Node.js可以作为nginx这样的反向代理，比如node-http-proxy和anyproxy等，其实使用Node.js做这种请求转发是非常简单的	看涨

# Ref

- ❄ [cnodejs.org](http://cnodejs.org)
- ❄ [https://github.com/ElemeFE/node-interview/tree/  
master/sections/zh-cn](https://github.com/ElemeFE/node-interview/tree/master/sections/zh-cn)
- ❄ <https://www.bookstack.cn/read/es6-3rd/sidebar.md>

# Thanks!!!

