# 2020.10.31（ 第三方库目录 – 记录遇到的第三方库 ）

## fs | 读取本地文件库

fs | 读取本地文件库 | const fs = require(‘fs’);

## http | 基本网络模块

http | 基本网络模块 | const http = require(‘http’);

## url | 基本网络模块

url | 基本网络模块 | const url = require(‘url’);

## gzip | 配置GIZP压缩让JS文件更加的小(COMPRESSION – GIZP压缩)

gzip | 配置GIZP压缩让JS文件更加的小(COMPRESSION – GIZP压缩) | const compression = require('compression');

## slugify | 字符串加工库

SLUGIFY | 字符串加工库: 安装yarn add slugify

官方文档: 文档参考npm: <https://www.npmjs.com/package/slugify>

## nodemon | node.js文件代码发生改变自动重启node服务，方便开发

nodemon | NODE.JS文件代码发生改变自动重启NODE服务，方便开发: 安装 yarn add nodemon –dev

## crypto | node.js自带的加密库

crypto | 基本加密模块 | const crypto= require(‘crypto’);

# 2020.11.1（ node.js | log.md | nodemon ）

## Log.md

Udemy课程：Jonas Schmedtmann - <https://www.udemy.com/course/nodejs-express-mongodb-bootcamp/learn/lecture/15080910#overview>

    # 1 ~ # 4: 介绍Node.js

    # 5( 完成笔记 )

        a) 00:00 - 为什么要使用Node.js?

        b) 3:49 - I/O model是什么? ( 等待研究 )

        c) 4:57 - Node.js实话项目，以及不适合项目

            0. 适合: 构建，视频网站，聊天工具, API交互web程序

            1. 不适合: 图像处理, 视频转换, 文件压缩

                a) 原因: 因为node.js社区没有相关开发

                b) 推荐: Python3

    # 6( 完成笔记 )

        a) 00:00 - Node.js小试牛刀

        b) 2:29 - " node "命令, 可以进入node.js命令行模式, 书写一些js语法

            0. 退出命令: " .exit " | Ctrl+D

        c) 4:10 - node命令行模式下，"TAB"按键，显示全局可用变量

        d) 5:25 - 下划线的使用，下划线代表上一个返回的数据，如: 3\*8 --> 24 --> \_+6 --> 30

        e) 5:50 - 查看变量下的内容: 如 String，则在node命令行下输入: " String. " + 按2下TAB键

        f) 6:09 - 清除屏幕: MAC --> commd + k; Window --> console.clear()

    # 7( 完成笔记 )

        a) 00:00 - 核心模块

        b) 1:35 - 命令：node xxx.js 运行js文件

        c) 3:40 - node.js下索引第三方库的写法

            0. fs库: 文件读写函数库

        d) 5:20 - 学会查看阅读node.js官方文档: https://nodejs.org/docs/latest-v11.x/api/index.html

    # 8( 完成笔记 )

        a) 00:00 - fs库的使用

        b) 3:15 - fs库读取文件

        c) 6:48 - fs库写入文件

            0. Date.now(): 当前时间戳( 毫秒为单位 )

    # 9( 完成笔记 )

        a) 00:00 - 4个重要的概念

            0. 同步( 阻塞代码 )

            1. 异步( 非阻塞代码 )

            2. 阻塞: 影响性能

            3. 非阻塞: 性能良好，需要callBack写法

        b) 1:58 - 同步，异步代码，对比

        c) 7:04 - I/O代表: 输入/输出

            0. node.js就是围绕着callBack来处理请求

        d) 8:04 - 注意: callBack写法 !== 一定是异步

        e) 9:19 - 回调地狱callBack Hell: 错误写法示例

    # 10( 完成笔记 )

        a) 00:00 - 异步读写文件

        b) 10:32 - 用回调地狱的形式，读写文件( 只是示例, 实际开发不可用 )

    # 11( 完成笔记 )

        a) 00:00 - 建立一个简单的server

        b) 7:21 - 完整的server代码

        c) 8:09 - 访问server

        d) 9:49 - Ctrl+C退出程序，注意：在修改完新的代码后，要重启服务

    # 12( 完成笔记 )

        a) 00:00 - 路由

        b) 2:08 - 通常路由是借助express库的，但是这里是小项目，用原生构建路由即可

        c) 8:19 - res.url反馈的是当前请求的路由路径，我们可以根据这个来做一些，简单的路由操作。同时为做更复杂的路由时，暂且引用"url库"

        d) 12:21 - res.writeHead改写包头使用示范

            0. 包的状态码: 如404，200, 等....

            1. "Content-type"熟悉: 可以声明返回数据类型, "text/html" 返回html数据

            2. "my-xxx": "也可以自定义添加一些内容"

    # 13( 完成笔记 )

        a) 00:00 - 构建一个简单的API

        b) 5:35 - "\_\_dirname"代表当前目录，通常在node.js下使用配置文件路径( 索引库的路径时除外 )

        c) 9:51 - 发送json数据

        d) 13:41 - 改进API，让数据只读取一次，api接口直接反馈数据。( 利用JS代码的作用域，解决 )。

    # [14( 完成笔记 )](#_实战：node.js渲染简易站点项目(_slugify_|)

        a) 00:00 - HTML创建模板变量

    # [15( 完成笔记 )](#_实战：node.js渲染简易站点项目(_slugify_|)

        a) 00:00 - 加工HTML模板

        b) 8:26 - 正则替换模板变量

        c) 11:47 - 模板变量函数书写完毕

        d) 16:52 - node.js配合，将模板产品渲染

        e) 18:38 - 修复HTML产品模板错误

    # [16( 完成笔记 )](#_实战：node.js渲染简易站点项目(_slugify_|)

        a) 00:00 - 解析产品URL，并构建产品详情页

        b) 0:20 - 导入url库

        c) 2:46 - 使用url.parse函数解析当前url，获取url中的参数

        d) 4:46 - 解析url: 目的是将url与url中的参数分离，保证路由正常不受url中的参数影响访问

        e) 8:30 - 渲染产品详情页面代码

        f) 8:48 - 修正产品详情页HMTL，返回/overview页面按钮

    # 17( 完成笔记 )

        a) 00:00 - Node.js每一个文件就是一个模块

        b) 3:09 - Node.js的方式 module.exports 导出函数，使可以在全局调用

        c) 4:57 - Node.js的方式 导入本地模块

    # 18( 完成笔记 )

        a) 00:00 - npm

        b) 4:27 - npm init 初始化建立npm

    # [19( 完成笔记 )](#_Nodemon配置)

        a) 00:00 - npm安装第三方库

        b) 1:56 - npm安装slugify到生产分支: npm install slugify - 更加易读的URL

        c) 3:49 - npm安装nodemon到开发分支: npm install nodemon --save-dev - 代码改变自动重启node服务

        d) 7:13 - npm在全局安装nodemon这样就能在全局使用，在不同项目间无需重复安装: npm i nodemon --global ( i 代表 install )

        e) 9:05 - 使用nodemon监听文件: nodemon index.js

        f) 11:40 - npm在package.json快捷命令配置

    # [20( 完成笔记 )](#_slugify库:_字符串加工库)

        a) 00:00 - slugify库的使用

        b) 6:31 - 根据slugify官方文档知：此乃字符串加工库

        c) 7:44 - 任务：根据slugify官方文档，做一些测试

    # 21( 完成笔记 )

        a) 00:00 - 版本号，以及更新

        b) 00:00 - 版本号解析: 例如: 1.18.11 --> 主要版本号.次要版本号.补丁程序版本号

            0. 主要版本号: 出现重大改变时改变，可能不支持向下兼容

            1. 次要版本号: 改动不大增加新功能时使用，并且向下兼容

            2. 补丁程序版本号: 修复bug错误时使用

        c) 4:24 - npm更新package.json库的方式

            0. 查看可更新版本: npm outdated

            1. 更新指定版本例: npm install xxx@1.0.0

            2. 在package.json下:

                a) "^1.0.0": "^"代表接受，次要版本，以及补丁版本的更新，不接受主要版本更新。

                b) "~1.0.0": "~"代表仅接受, 补丁更新

                c) "*\*1.0.0": "\**"代表接受全部更新( 不推荐 )

            3. 直接更新版本: npm update xxx

        d) 7:56 - npm删除库: npm uninstall xxx

    # 22( 完成笔记 )

        a) 00:00 - vscode配置和插件介绍

        b) 1:16 - 插件

            0. image preview: 代码图片预览

            1. DotENV: 查看当前本地环境

            2. Pug beautiful：用于HTML模板浏览使更漂亮

            3. TabNine：让tab键智能写一些代码片段( 不推荐 )

            4. TODO Highlight: 参考2:10视频，TODO起高亮作用

        c) 3:55 - Prettier 格式化代码插件：功能开启 settings ---> 搜索format ---> Format On Save 打勾

        d) 5:32 - 自定义prettier, 单引号变为双引号, 更多功能请查阅官方文档

    # 23( 无需笔记 )

        a) 00:00 - 回顾

    # 24( 无需笔记 )

        a) 00:00 - 后端原理简介

    # [25( 完成笔记 - 需要根据视频截图做笔记 )](#_网络工作原理)

        a) 00:00 - 网络工作原理

        b) 11:49 - 完整的，客户访问服务器，流程图

        c) 13:13 - TCP/IP总体概述: 传递数据时，分解成多个小块，去发送，每个小块都有IP通过路由来到达访问目标。

            0. 分解成小块的原因是：尽量减少网络上的传输拥挤

    # [26( 完成笔记 - 需要根据视频截图做笔记 )](#_http实战)

        a) 00:00 - http通过调试器，看一些包的基本信息

        b) 0:56 - 浏览器设定为，无缓存状态, 方便调试

    # [27( 完成笔记 )](#_Web服务的组成)

        a) 00:00 - 前端和后端开发

        b) 0:48 - Front End: 前端工程师

        c) 3:46 - Back End: 后端工程师

            0. 存储文件 + http( 前端后端沟通的桥梁 ) + APP( 逻辑程序 ) + DataBase( 数据库 ) = 组建成WEB服务器

    #[28( 完成笔记 )](#_区分:_静态网站，动态网站，API)

        a) 00:00 - 区分: 静态网站，动态网站，API

        b) 3:58 - 区分图: 静态网站，动态网站( 服务端渲染 )

        c) 6:36 - Node.js非常适合构建API，当然也适合，服务端渲染的构建动态网站

        d) 6:52 - 区分图: 动态网站( 服务端渲染 )，API( 客户端渲染 )

        e) 9:08 - API的巨大优势( 跨平台 – 甚至有公司专门只提供API服务来收益 )

    # 29( 无需笔记 )

        a) 00:00 - 输入了解Node.js简介

    # [30( 完成笔记 - 需要根据视频截图做笔记 )](#_Node.js底层(_crypto_))

        a) 00:00 - Node.js依赖

        b) 1:54 - Node.js二大依赖

            0. Google V8

                a) 作用: 将node.js代码转为，计算机可以理解的机器码

                b) 底层: JS & C++

            1. libuv

                a) 作用:

                    0. 异步循环( 处理简单的事情 )，线程池( 处理复杂的事情 )

                    1. 帮助JavaScript: 访问，文件系统，网络系统，等...

                b) 底层: C++

            3. 其它依赖:

                a) http-parser库: 用于解析http

                b) c-ares库: 处理一些DNS请求的内容

                c) OpenSSL库: 记录

                d) zlib: 压缩

        c) 3:41 - Node.js底层依赖图

    # [31( 完成笔记 )](#_进程，线程，线程池)

        a) 00:00 - 进程，线程，线程池

        b) 02:29 - Node.js单线程执行

            0. 程序初始化，

            1. 顶级代码先执行

            2. 所需模块正常

            3. 注册回调事件: 向http server 监听端口等...

            4. 事件循环开始: Start Event Loop ( 防止阻塞 | Node体系的核心 )

        c) 04:00 - 线程池: 帮助单线程的Node.js解决太沉重的任务，防止阻塞事件循环

            0. Libuv提供线程池

            1. 线程池: 提供额外的4个线程, 用于处理沉重的任务

                a) 最大理论可配置128线程，但大多情况4个线程已经够用

                b) 沉重任务交给线程，由底层决定，开发人员无法操控

            2. 线程池: 常用来解决

                a) 文件系统访问

                b) 加密相关，如缓存密码

                c) 压缩内容

                d) DNS查找

    # [32( 完成笔记 )](#_事件循环，队列顺序)

        a) 00:00 - Node.js架构核心 事件循环 | Event Loop | Node.js防止阻塞规则( 核心 )

        b) 4:09 - Event Loop | 事件循环4个队列阶段顺序:

            0. timer相关的callback队列：如settimeout

            1. I/O事件相关callback队列: 如，异步的文件系统访问，网络相关异步

            2. setImmediate callBack队列: 是特殊的计时器，更高级的用例中很重要

            3. Close callBack | 关闭回调队列: 如，web server停止运行

        c) 6:00 - 2个特殊队列

            0. nextTick()队列

            1. 微任务队列

            2，目的解决: somePromise异步函数

        d) 9:15 - Node.js单线程的优缺点 | Node.js防止阻塞规则( 核心 )

            0. 优点:

                a) 因为是单线程处理，主线程只处理简单的事件循环，繁杂交给线程池，因此资源占用小

                b) Node.js轻巧且易扩展

            1. 缺点:

                a) 单线程非常危险，因为一旦单线程被阻塞，程序将崩溃

            2. PHP优缺点:

                a) 优点: 每个用户对应一个线程

                b) 缺点: 占用资源高

            3. Node.js防止阻塞规则( 核心 )

                a) fs,crypto,zlib( 文件|加密|网络相关 )等库，不要使用非异步callback函数

                b) 事件循环下: 不要进行复杂的计算

                c) 尽量减少小心，大型的json字符串解析

                d) 不要使用过于复杂的，正则表达式

    # [33( 完成笔记 )](#_事件循环，队列顺序)

        a) 00:00 - 实践事件循环体系( 核心 )

        b) 4:38 - 异步callback函数测试代码

        c) 9:49 - I/O下的异步顺序代码测试

        d) 12:12 - 加密库: crypto

            0. 异步加密函数:

                a) 模型: crypto.pbkdf2( "密码", "( 未知 )", "密码迭代次数", "密码长度", "加密算法", "回调函数" )

                b) 例如: crypto.pbkdf2( "password", "salt", 100000, 1024, "sha512", () => {} )

        e) 13:57 - 加密时间统计, 以此方便判断，循环事件是否被阻塞

        f) 15:22 - 配置线程池数量( 最大128 ), 在配合时间统计，以此判断线程池对事件的影响

        g) 17:00 - 同步加密函数, 配合时间统计，用于测试被阻塞时的状态

    # 34( 等待笔记 )

        a) 00:00 -

## \* node.js获取第三方库写法

## \* 1. fs库: 读写文件

## \* 2. Node.js理论知识

*/\*\**

\* node.js获取第三方库写法

*\*      a) fs库：为读写文件的库*

*\*/*

const fs = require('fs');

*/\*\**

\* 0. hello word 练习

*\*/*

*// const hello = " Hello World !!";*

*// console.log( hello );*

*/\*\**

\* 1. fs库: 读写文件

*\*      a) 同步*

*\*          0. fs.readFileSync函数: 同步读取文件*

*\*              a) 模型: fs.readFileSync( "文件路径", "编码" )*

*\*          1. fs.writeFileSync函数: 同步写入文件*

*\*              a) 模型: fs.writeFileSync( "文件路径", "写入内容" )*

*\*      b) 非同步*

*\*          0. fs.readFile函数: 异步读取文件内容*

*\*              a) 模型: fs.readFile( "文件路径", "utf-8", "回调函数" )*

*\*          1. fs.writeFile函数: 异步书写内容*

*\*              a) 模型: fs.writeFile( "文件路径", "书写内容", "utf-8", "回调函数" );*

\* 2. \_\_dirname意思: 路径 - 当前目录下

*\*      a) 使用范围: 除require读取第三方库时不适用他, 其它关于路径的基本都使用他*

*\*/*

const textIn = fs.readFileSync( `${\_\_dirname}/txt/input.txt`,"utf-8" ); *// fs.readFileSync函数: 同步读取文件*

console.log('textIn', textIn);

const textOut = `${textIn} \n--->写入文件时间: ${Date.now()}`;

fs.writeFileSync(`${\_\_dirname}/txt/outInput.txt`,textOut);              *// fs.writeFileSync函数: 同步写入文件*

*/\*\**

\* 3. Node.js理论知识

*\*      a) Node.js的作用:*

*\*          0. 适合: 构建，视频网站，聊天工具, API交互web程序*

*1. 不适合: 图像处理, 视频转换, 文件压缩*

*a) 原因: 因为node.js社区没有相关开发*

*b) 推荐: Python3*

*\*      b) 4个重要的概念:*

*\*          0. 同步( 阻塞代码 )*

*\*          1. 异步( 非阻塞代码 )*

*\*          2. 阻塞: 影响性能*

*\*          3. 非阻塞: 性能良好，需要callBack写法*

*\*      c) I/O代表: 输入/输出*

*\*      d) node.js核心: 围绕着callBack来处理请求*

*\*          0. 注意: callBack写法 !== 一定是异步*

*\*          1. 回调地狱callBack Hell: 错误写法示例*

*\*/*

// a) 回调地狱写法示例( 注意: 切换在生产环境使用 )：异步读取文件/异步书写文件

*//      0. 逻辑步骤:*

*//          a) R - start.txt -->*

*//          b) R - 以上一次读取内容为名字.txt -->*

*//          c) R - append.txt -->*

*//          d) W - final.txt - 读取文件的内容整和*

fs.readFile( "./txt/start.txt","utf-8",( *err*, *data* )=>{

    fs.readFile( `./txt/${ data }.txt`, "utf-8", ( *err*, *data\_1* ) => {

        fs.readFile("./txt/append.txt", "utf-8", ( *err*, *data\_2* ) => {

          // fs.writeFile函数: 异步书写函数

            fs.writeFile(

                "./txt/final.txt",

                `${data\_1}\n${data\_2}\n书写时间: ${Date.now()}`,

                "utf-8",

                ( *err* ) => {s

                 console.log("书写完毕!");

             } );

        })

    } );

} );

## 回调地狱callBack Hell: 错误写法示例



## \* node.js获取第三方库写法

## \* 0. 建立一个简单的server

## \* 1. 构建简易的路由

## \* 2. http库: server相关库

## \* 3. 读取文件发送数据的二种写法

*/\*\**

\* node.js获取第三方库写法

*\*      a) fs库：为读写文件的库*

*\*      b) http库: server相关*

*\*/*

const http = require("http");

const fs = require("fs");

*/\*\**

\* 0. 建立一个简单的server

\* 1. 构建简易的路由

*\*      a) 注意: 生产环境使用express，当前为小测试故用基础函数来写路由*

\* 2. http库: server相关库

*\*      a) 创建服务: http.createServer( ( req, res ) => {} )*

*\*          0. 回调函数:*

*\*              a) req: 接受类参数*

*\*                  0. req.url属性: 当前访问的url路径*

*\*              b) res：发送类参数*

*\*                  0. res.end函数: 发送数据*

*\*                  1. res.writeHead函数: 改写"包头"*

*\*                      a) 模型: res.writeHead( "状态码",{ 改写属性 } )*

*\*                      b) 状态码:*

*\*                          0. 404: 包不存在*

*\*                          1. 200: 包正常*

*\*                      c) 改写属性相关:*

*\*                          0. "Content-type"指定发送数据类型:*

*\*                              a) 发送JSON数据: "Content-type": "application/json"*

*\*                              b) 发送的HTML数据:  "Content-type":"text/html"*

*\*      b) 监听端口服务: xxxServer.listen()*

*\*          0. 模型: xxxServer.listen( "监听的端口", "监听的IP地址", "监听时促发的回调函数" )*

*\*          1. 监听的ip地址: 127.0.0.1 - 代表监听本地*

\* 3. 读取文件发送数据的二种写法

*\*      a) server下读取文件发送*

*\*          0. 异步读取JSON文件发送数据( 未优化写法 - 缺陷: 每一次访问API都要读取文件, 资源浪费 )*

*\*      b) server外读取文件发送*

*\*          1. server外非异步读取JSON文件发送数据( 优化写法 - 优点: 每一次访问API无需重复读取文件 )*

*\*          2. 注意: 数据直接在没有启动server前，写入非异步读取好文件，这样server用到时直接发送数据*

*\*/*

// a) server外非异步读取JSON文件发送数据( 优化写法 - 优点: 每一次访问API无需重复读取文件 )

const jsonData = fs.readFileSync( `${\_\_dirname}/dev-data/data.json`,"utf-8" );

// b) 建立server

const server = http.createServer( ( *req*, *res* ) => {

    const pathUrl = req.url;                                        // req.url属性: 当前访问的url路径名称

    if( pathUrl === "/" ){

        res.end(`<h1>Hello World - Node.js Server - \_\_OO7\_\_<h1>`);  // res.end函数: 返回前端数据

    }else if ( pathUrl === "/pro" ){

        res.end(" this pro page ");

    }else if ( pathUrl === "/dataApi" ){

        // 异步读取JSON文件发送数据( 未优化写法 - 缺陷: 每一次访问API都要读取文件, 资源浪费 )

        fs.readFile( `${\_\_dirname}/dev-data/data.json`,"utf-8",( *err*, *data* ) => {

            res.writeHead(200,{

                "Content-type": "application/json",

            });

            res.end( data );

        } );

    }

    else if ( pathUrl === "/newDataApi" ){

        // server外非异步s读取JSON文件发送数据( 优化写法 - 优点: 每一次访问API无需重复读取文件 )

*//      a) 注意: 数据直接在没有启动server前，写入非异步读取好文件，这样server用到时直接发送数据*

        res.writeHead(200,{

            "Content-type": "application/json",

        });

        res.end( jsonData );

    }

    else{

        res.writeHead( 404, {                                       // res.writeHead函数: 自定义"包头"

            "Content-type":"text/html",

            "div-props": "xxx",

        });

        res.end(`<h1>404</h1>`);

    }

} );

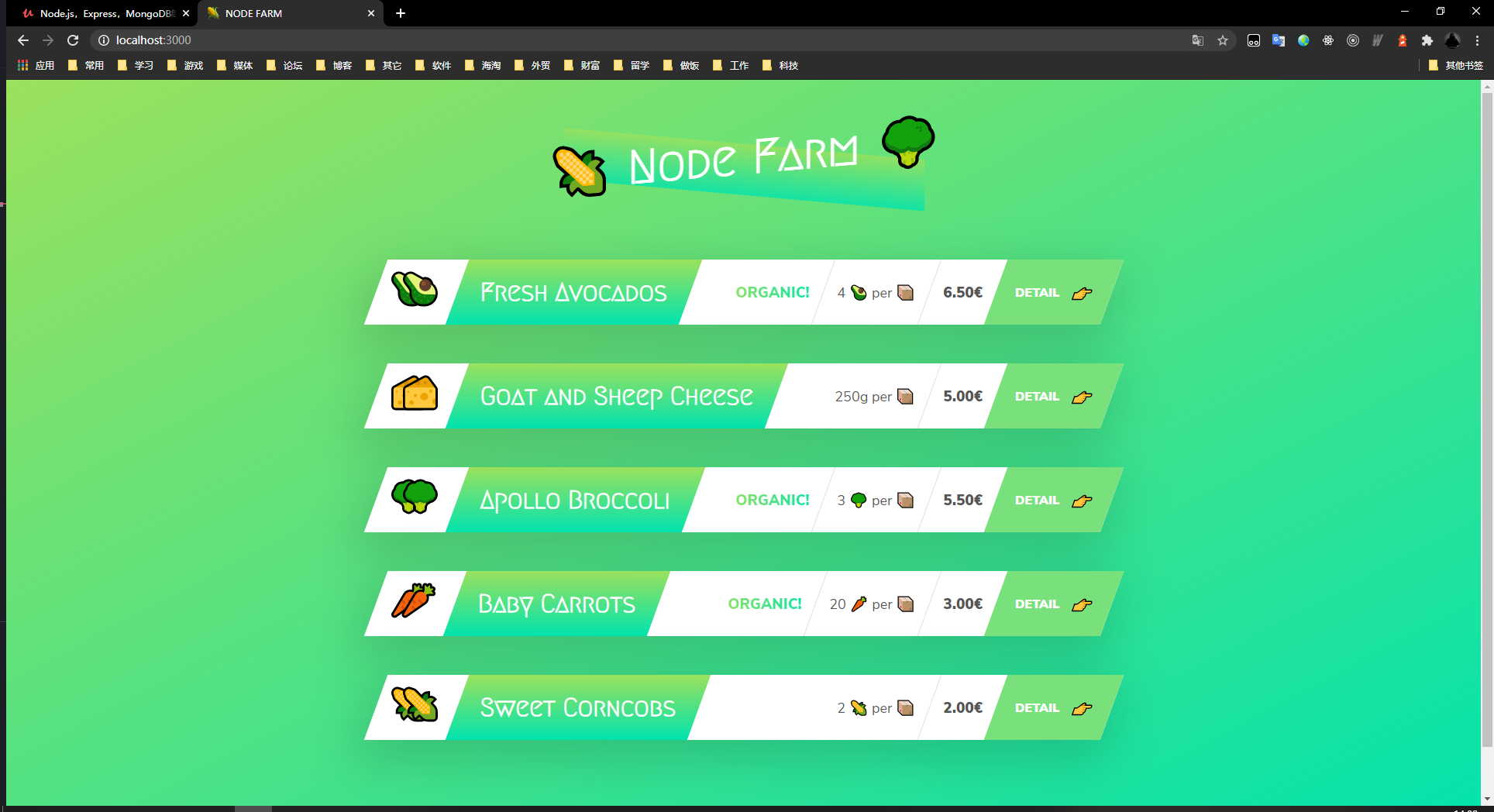
// c) 监听指定端口，促发server

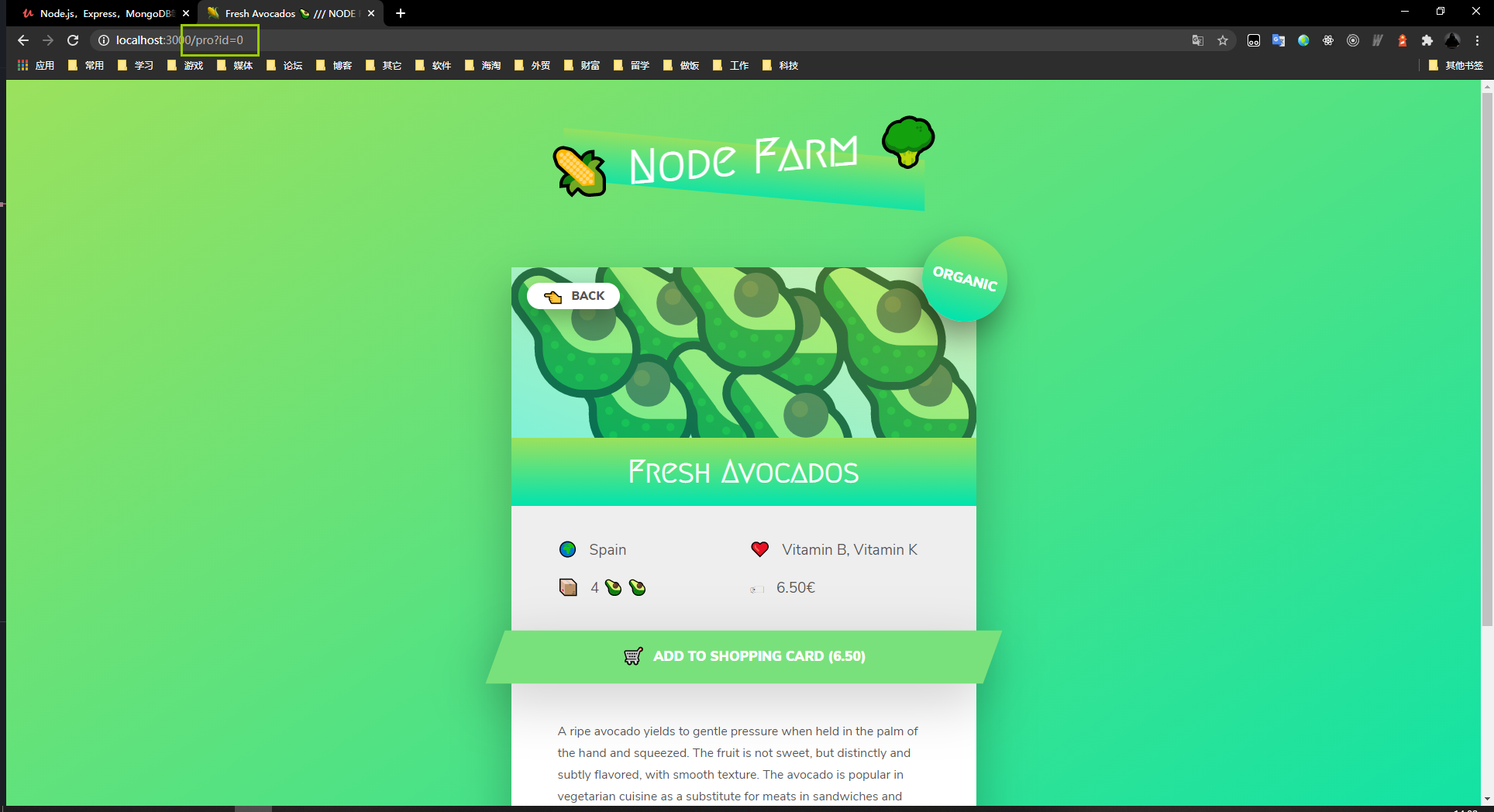
server.listen( 3000, "127.0.0.1", () => {

    console.log('监听端口3000');

} );

## 实战：node.js渲染简易站点项目( slugify | url )

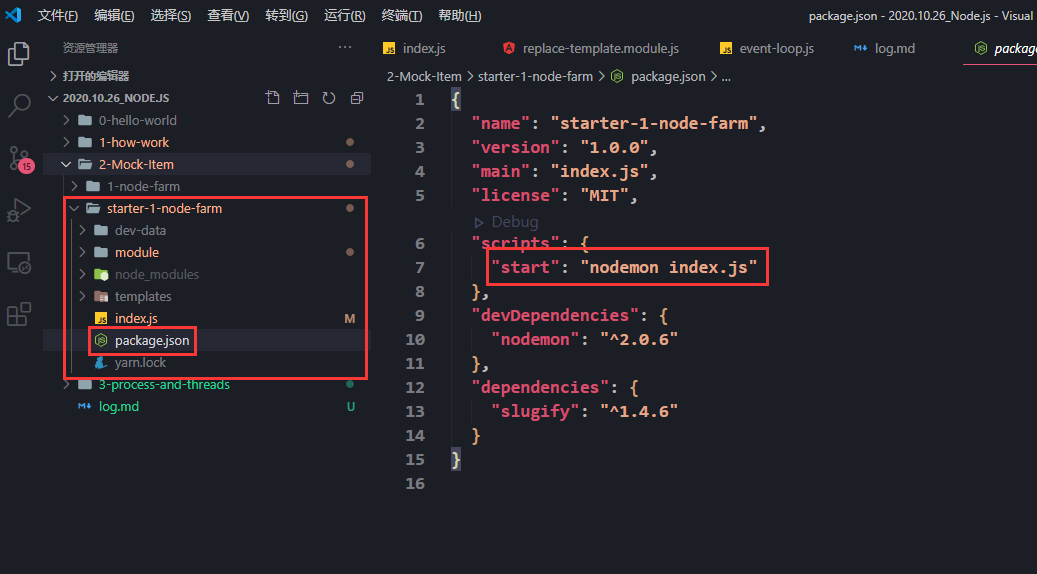




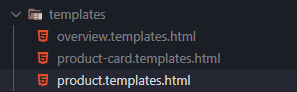
### 目录结构



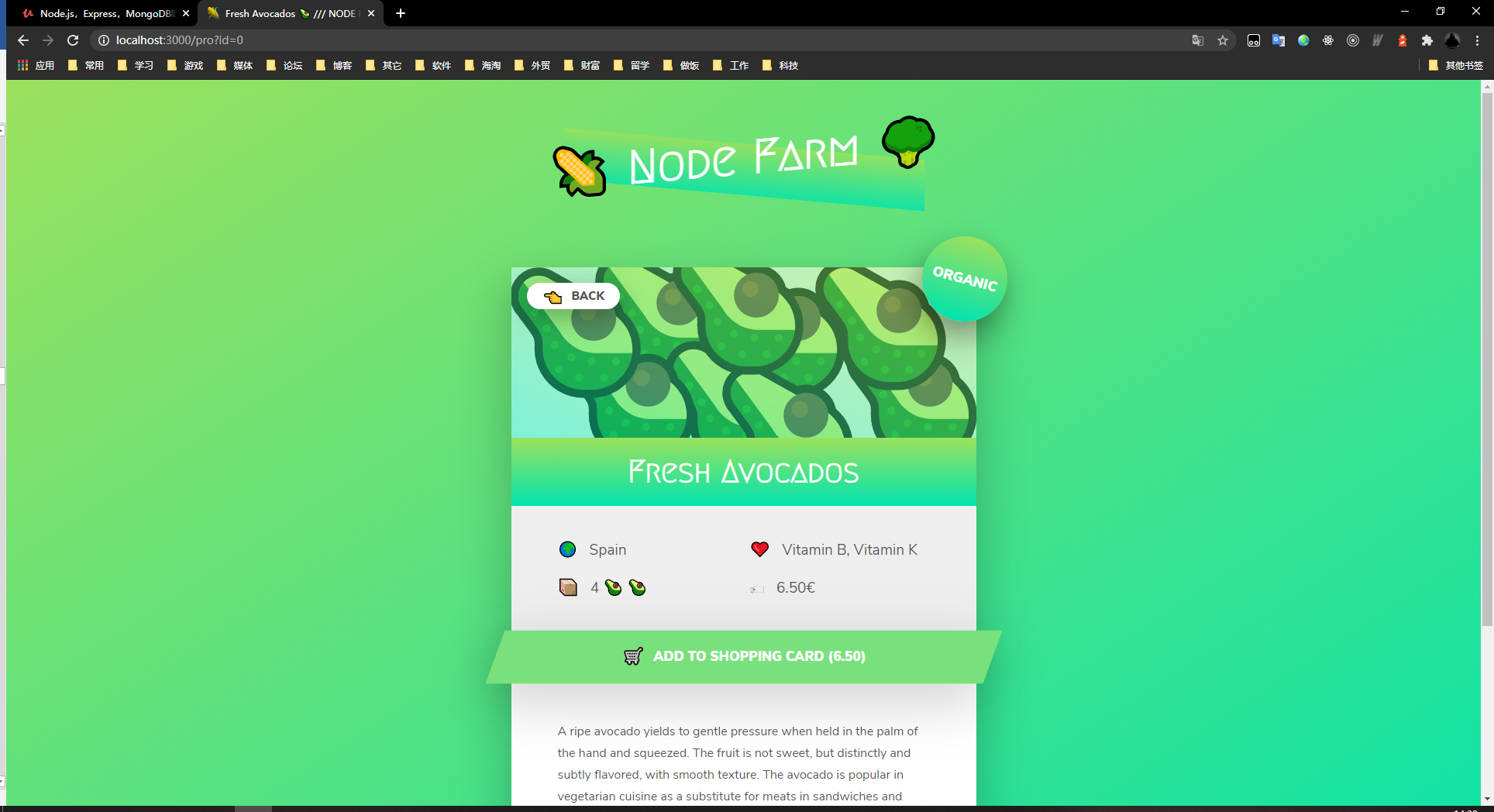
### Nodemon配置



### Templates/xxx.html | H5模板构建 | （ 核心：原生js项目开发思维 )



#### H5 | 产品页（ 主页类似，就不做演示了 | 已删除css3代码 ）



<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <link

      href="https://fonts.googleapis.com/css?family=Megrim|Nunito+Sans:400,900"

      rel="stylesheet"

    />

    <link

      rel="icon"

      href="https://emojipedia-us.s3.dualstack.us-west-1.amazonaws.com/thumbs/240/apple/155/ear-of-maize\_1f33d.png"

    />

    <title>Fresh Avocados 🥑 /// NODE FARM</title>

  </head>

  <body>

    <div class="container">

      <h1>🌽 Node Farm 🥦</h1>

      <figure class="product">

        <div class="product\_\_organic"><h5>Organic</h5></div>

        <a href="/" class="product\_\_back">

          <span class="emoji-left">👈</span>Back

        </a>

        <div class="product\_\_hero">

          <span class="product\_\_emoji product\_\_emoji--1">{%IMAGE%}</span> <!--用于配合js函数替换、其实就是原生开发思想-->

          <span class="product\_\_emoji product\_\_emoji--2">{%IMAGE%}</span>

          <span class="product\_\_emoji product\_\_emoji--3">{%IMAGE%}</span>

          <span class="product\_\_emoji product\_\_emoji--4">{%IMAGE%}</span>

          <span class="product\_\_emoji product\_\_emoji--5">{%IMAGE%}</span>

          <span class="product\_\_emoji product\_\_emoji--6">{%IMAGE%}</span>

          <span class="product\_\_emoji product\_\_emoji--7">{%IMAGE%}</span>

          <span class="product\_\_emoji product\_\_emoji--8">{%IMAGE%}</span>

          <span class="product\_\_emoji product\_\_emoji--9">{%IMAGE%}</span>

        </div>

        <h2 class="product\_\_name">{%PRODUCT\_NAME%}</h2>

        <div class="product\_\_details">

          <p><span class="emoji-left">🌍</span> {%FROM%}</p>

          <p><span class="emoji-left">❤️</span> {%NUTRIENTS%}</p>

          <p><span class="emoji-left">📦</span> {%QUANTITY%} {%IMAGE%}</p>

          <p><span class="emoji-left">🏷</span> {%PRICE%}€</p>

        </div>

        <a href="#" class="product\_\_link">

          <span class="emoji-left">🛒</span>

          <span>Add to shopping card ({%PRICE%})</span>

        </a>

        <p class="product\_\_description">

          {%DESCRIPTION%}

        </p>

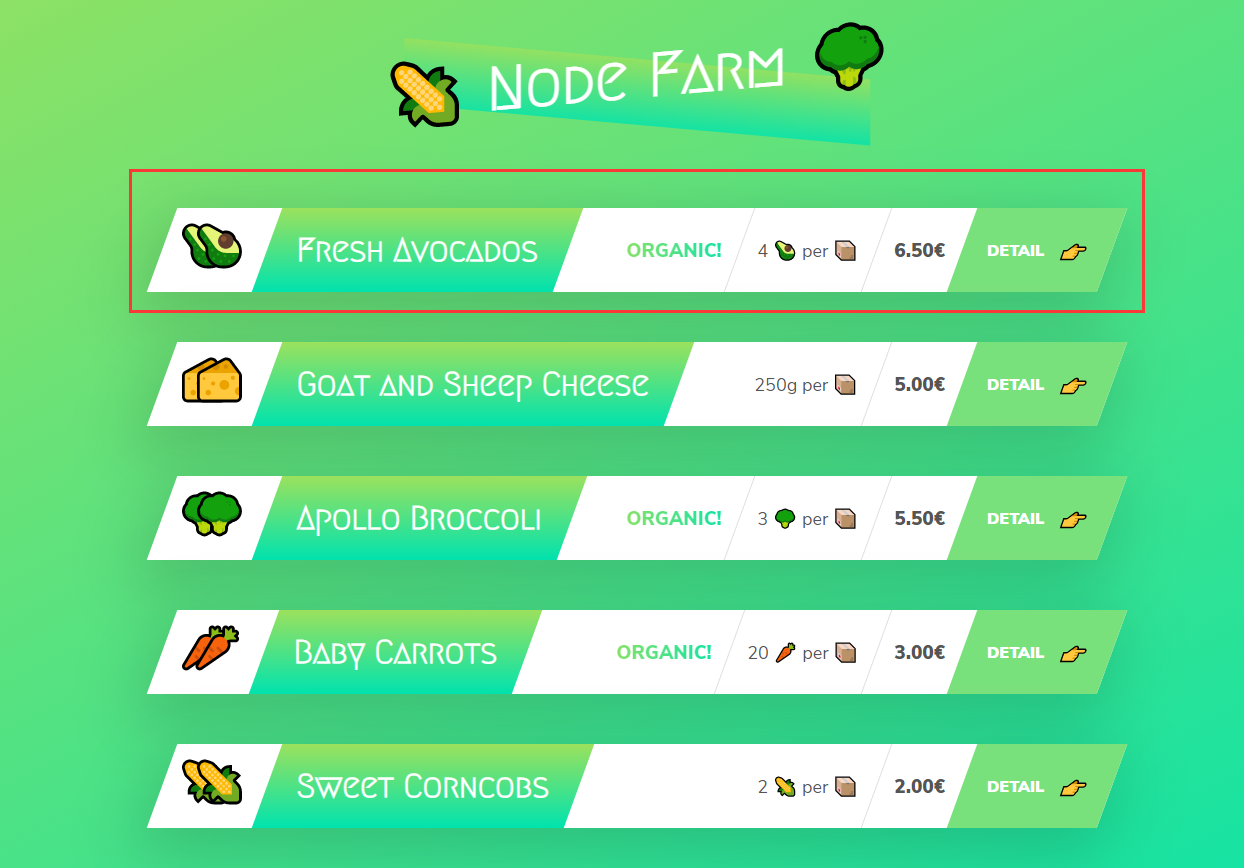
      </figure>

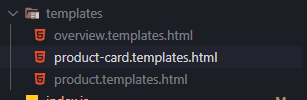
    </div>

  </body>

</html>

#### H5 | 单个产品，用于迭代渲染





<figure class="card">

    <div class="card\_\_emoji">{%IMAGE%}{%IMAGE%}</div>

    <div class="card\_\_title-box">

        <h2 class="card\_\_title">{%PRODUCT\_NAME%}</h2>

    </div>

    <div class="card\_\_details">

    <div class="card\_\_detail-box {%NOT\_ORGANIC%} ">

        <h6 class="card\_\_detail card\_\_detail--organic">Organic!</h6>

    </div>

    <div class="card\_\_detail-box">

        <h6 class="card\_\_detail">{%QUANTITY%} per 📦</h6>

    </div>

    <div class="card\_\_detail-box">

        <h6 class="card\_\_detail card\_\_detail--price">{%PRICE%}€</h6>

    </div>

    </div>

    <a class="card\_\_link" href="/pro?id={%ID%}">

    <span>Detail <i class="emoji-right">👉</i></span>

    </a>

</figure>

### module/xxx.js | 模块函数构建，用于迭代渲染h5模板

#### MODULE.EXPORTS导出匿名函数 | 使可在全局索引使用



*/\*\**

*\**module.exports导出匿名函数 | 使可在全局索引使用

*\*/*

module.exports = ( *temp*, *proItemData* ) => {

    let result = temp;

    result = result.replace( /{%PRODUCT\_NAME%}/g, proItemData.productName );

    result = result.replace( /{%IMAGE%}/g, proItemData.image );

    result = result.replace( /{%QUANTITY%}/g, proItemData.quantity );

    result = result.replace( /{%PRICE%}/g, proItemData.price );

    result = result.replace( /{%ID%}/g, proItemData.id );

    result = result.replace( /{%NUTRIENTS%}/g, proItemData.nutrients );

    result = result.replace( /{%FROM%}/g, proItemData.from );

    result = result.replace( /{%DESCRIPTION%}/g, proItemData.description );

*// 非有机物加入 not-organic 加入css3样式*

    if( !proItemData.organic ) result = result.replace( /{%NOT\_ORGANIC%}/g, 'not-organic' );

    return result;

};

### index.js | 简单的node.js页面项目示例

#### 配合module.exports导入匿名函数使用

#### slugify库: 字符串加工库

#### url库：解析url内容

*/\*\**

*\**简单的node.js页面项目示例

*\*      a) 核心: 替换字段页面模板*

*\*/*

const http = require("http");

const fs = require("fs");

const url = require("url");

// 配合module.exports导入匿名函数使用

const replaceTemplates = require("./module/replace-template.module");

*/\*\**

*\**slugify库: 字符串加工库

*\*      a) 文档参考npm:*[*https://www.npmjs.com/package/slugify*](https://www.npmjs.com/package/slugify)

*\*/*

const slugify = require("slugify");

// a) 获取数据

const proData = fs.readFileSync( `${\_\_dirname}/dev-data/data.json`, "utf-8" );

const proDataObj = JSON.parse( proData );

// b) 获取h5模板

const over\_templates = fs.readFileSync( `${\_\_dirname}/templates/overview.templates.html`, "utf-8" );

const pro\_templates = fs.readFileSync( `${\_\_dirname}/templates/product.templates.html`, "utf-8" );

const proCard\_templates = fs.readFileSync( `${\_\_dirname}/templates/product-card.templates.html`, "utf-8" );

// c) 创建http服务

const server = http.createServer( ( *req*, *res* ) => {

*/\*\**

*\**url库：解析url内容

*\*      0. 导入: const url = require("url");*

*\*      1. url.parse: 解析url内容*

*\*          a) 模型:  url.parse( "访问的url", true )*

*\*          b) 返回对象参数:*

*\*              0. pathname属性: 当前url，不包含参数*

*\*              1. query属性: url中的参数，对象类型数据*

*\*          c) 例如:*

*\*              0. url.parse( "/pro?id=1", true )*

*\*              1. { ...other, pathname: "/pro", query: { id: "1" } }*

*\*/*

    const { pathname, query } = url.parse( req.url, true );

    if( pathname === "/" || pathname === "/over" ){

*/\*\**

\* 主页:

*\*/*

        // a) 构建迭代产品HTML

        const proCardHtml = proDataObj.map( *itemData* => replaceTemplates( proCard\_templates, itemData ) ).join("");

        // b) 模板内容替换修改

        const output = over\_templates.replace( "{%PRODUCT\_CARDS%}", proCardHtml );

        res.writeHead( 200, {

            "Content-type": "text/html",

        } );

        res.end( output );

    }

    else if( pathname === "/pro" ) {

*/\*\**

\* 产品详情页:

*\*/*

        // a) 根据url参数查询数据

        const proItemData = proDataObj[ parseInt( query.id ? query.id : 0 ) ]

        // b) 模板内容替换修改

        const output = replaceTemplates( pro\_templates, proItemData );

        res.writeHead( 200, {

            "Content-type": "text/html",

        } );

        res.end( output );

    }

    else if( pathname === "/proData" ){

*/\*\**

\* JSON数据:

*\*/*

        res.writeHead( 200, {

            "Content-type": "application/json",

        } );

        res.end( proData );

    }

    else{

*/\*\**

 \* 404页面:

*\*/*

        res.writeHead( 404, {

            "Content-type": "text/html",

        } );

        res.end(`<h1> 404 </h1>`);

    };

} );

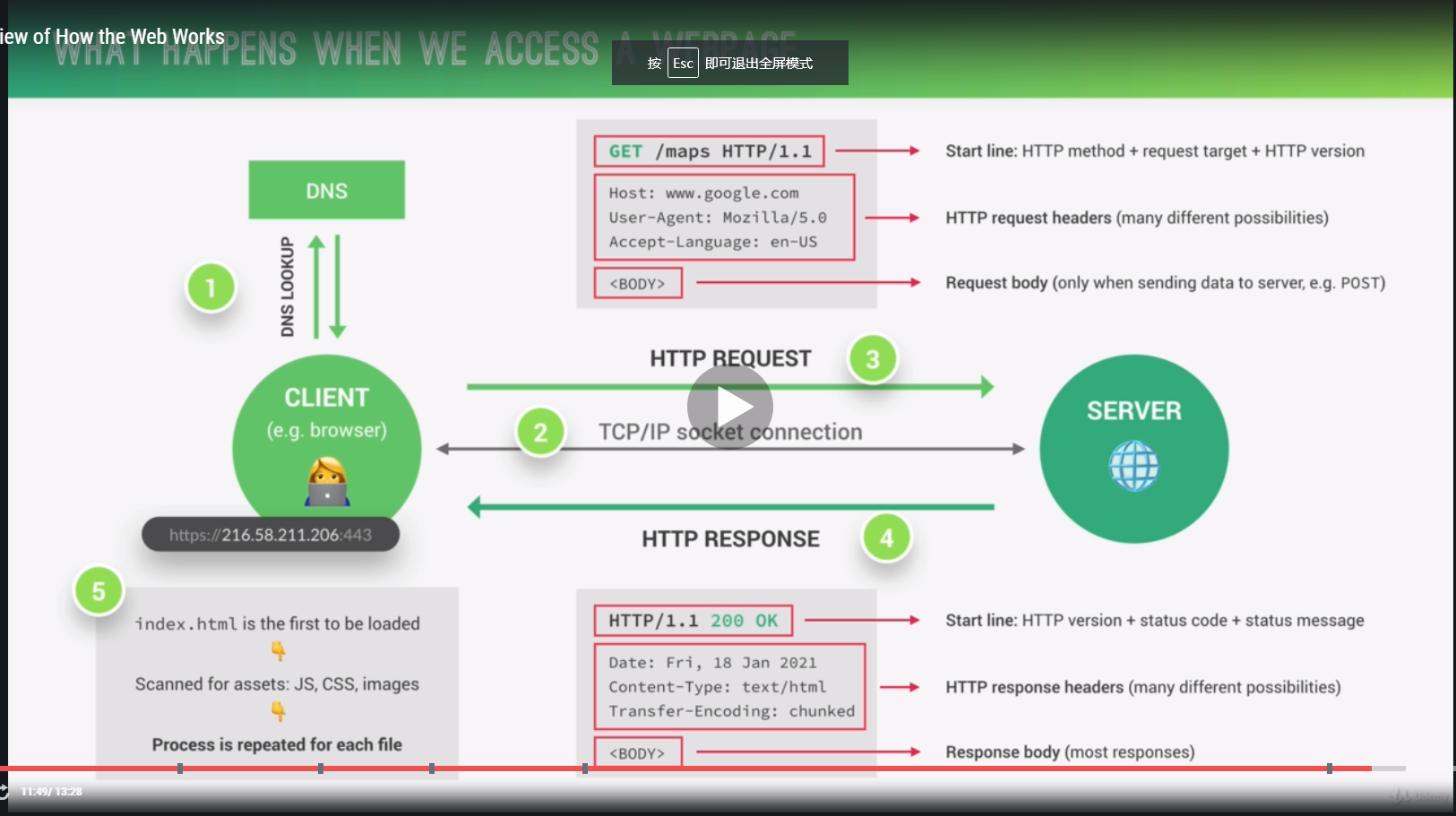
// d) 监听端口

server.listen( 3000, "127.0.0.1", () => {

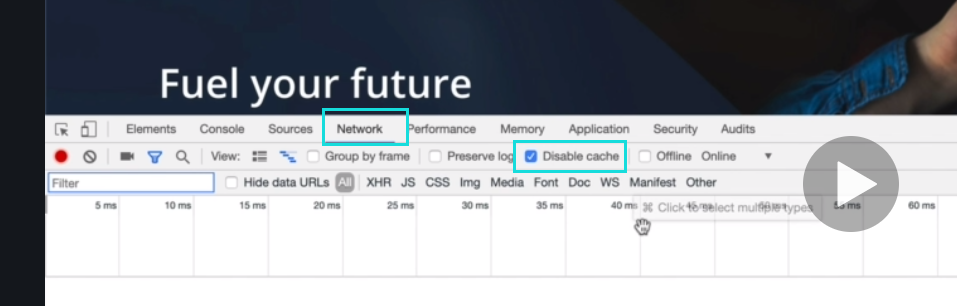
    console.log("http://localhost:3000/");

} );

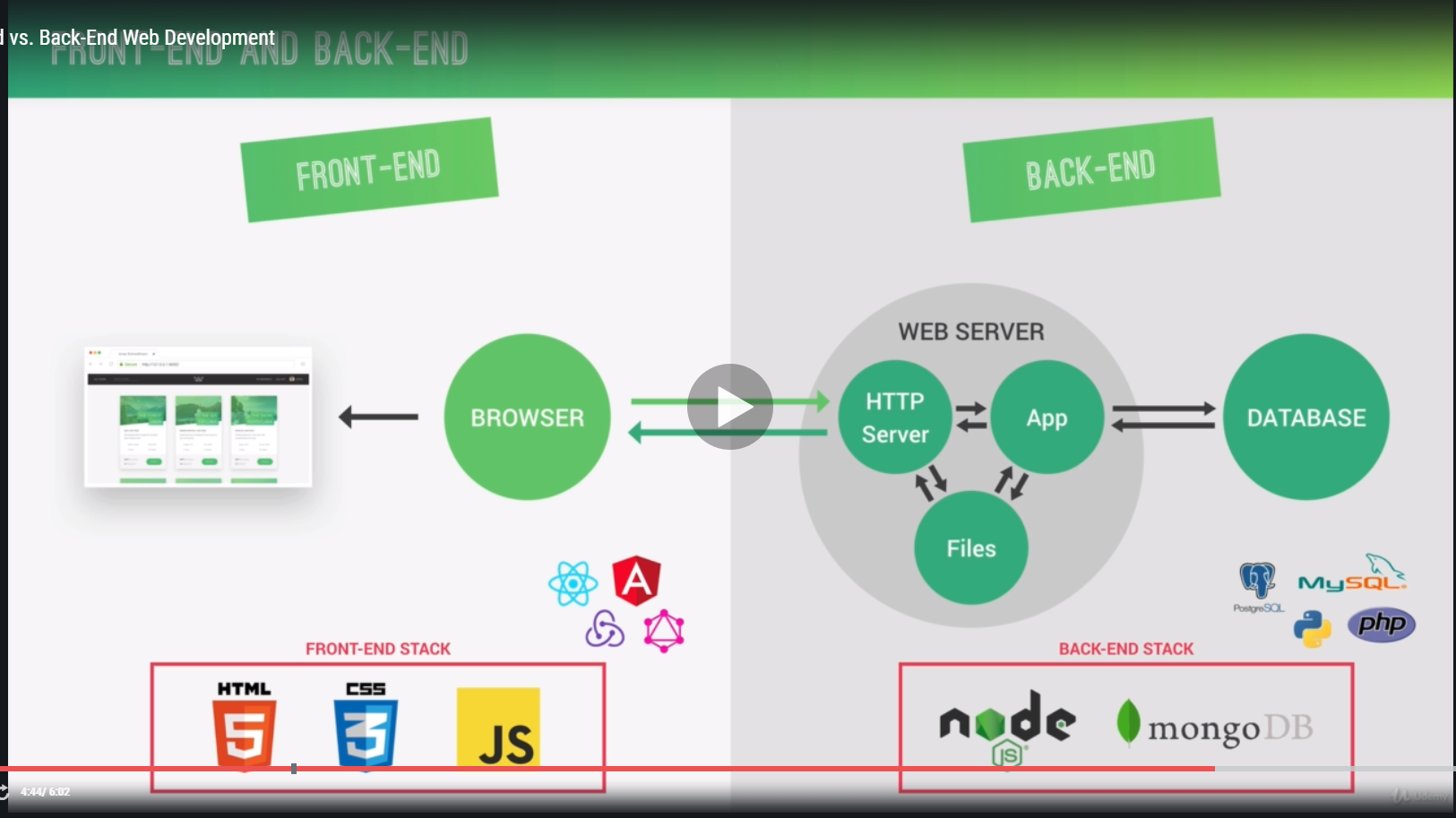
## 网络工作原理



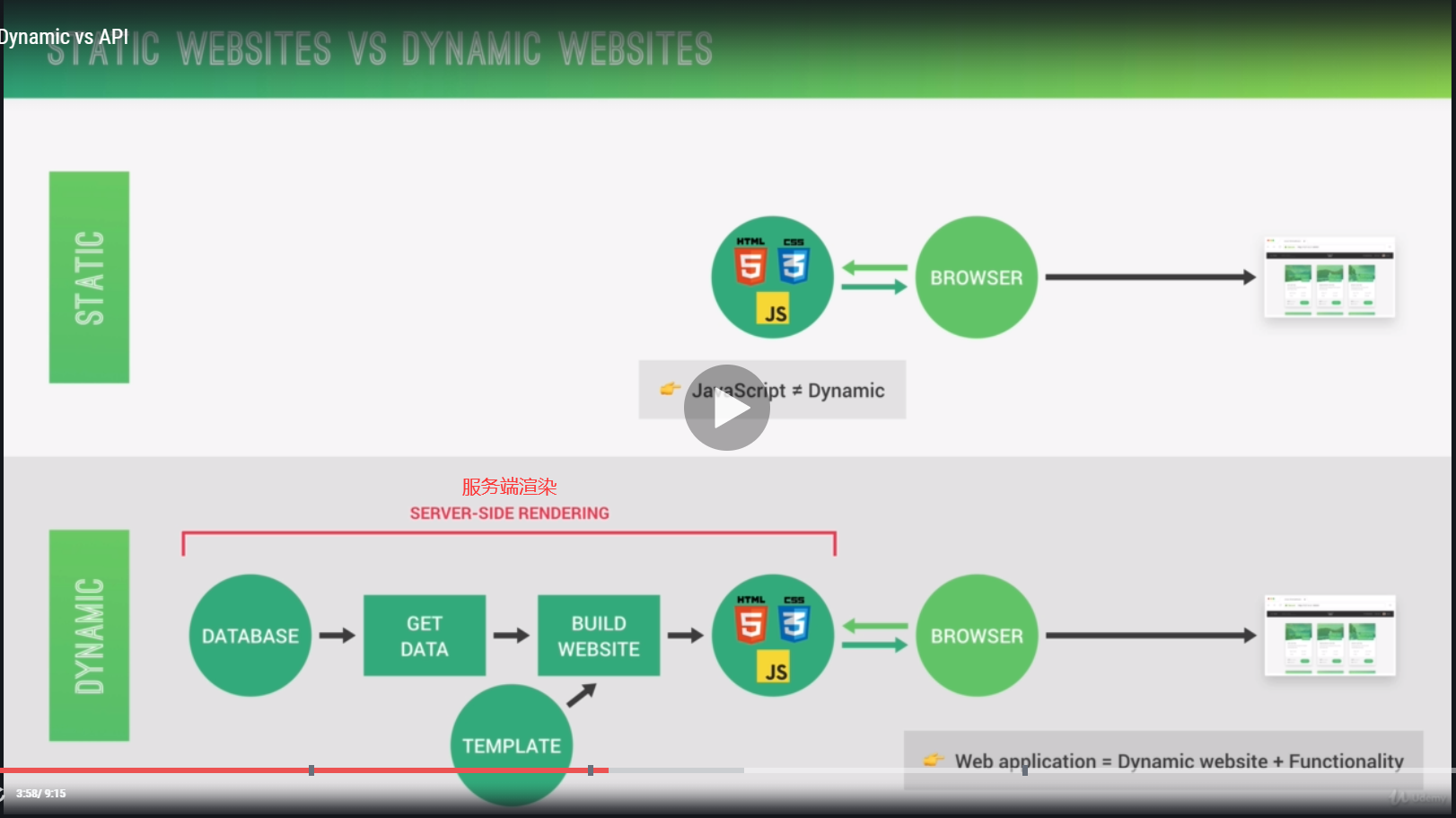
### http实战



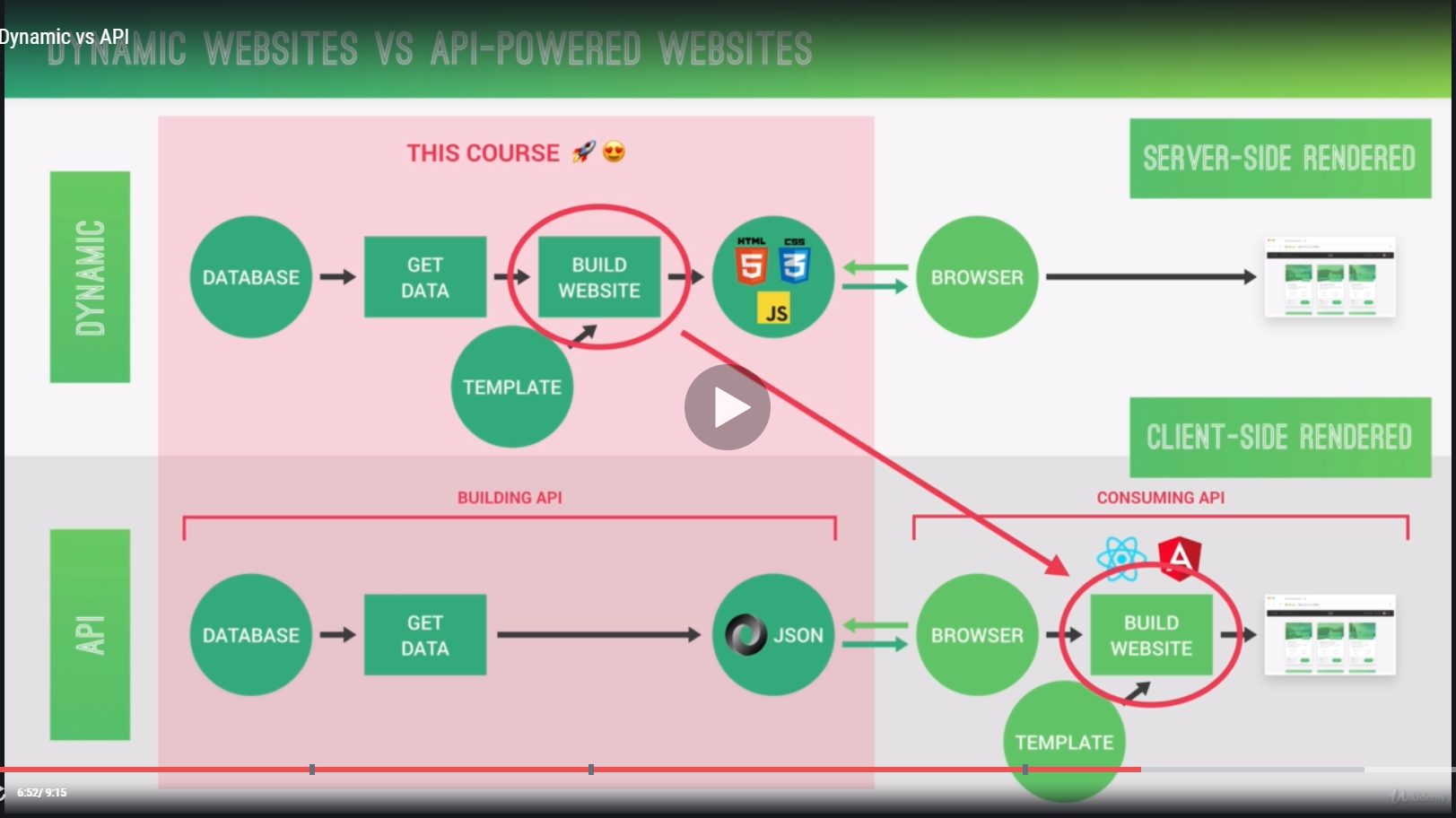
### Web服务的组成



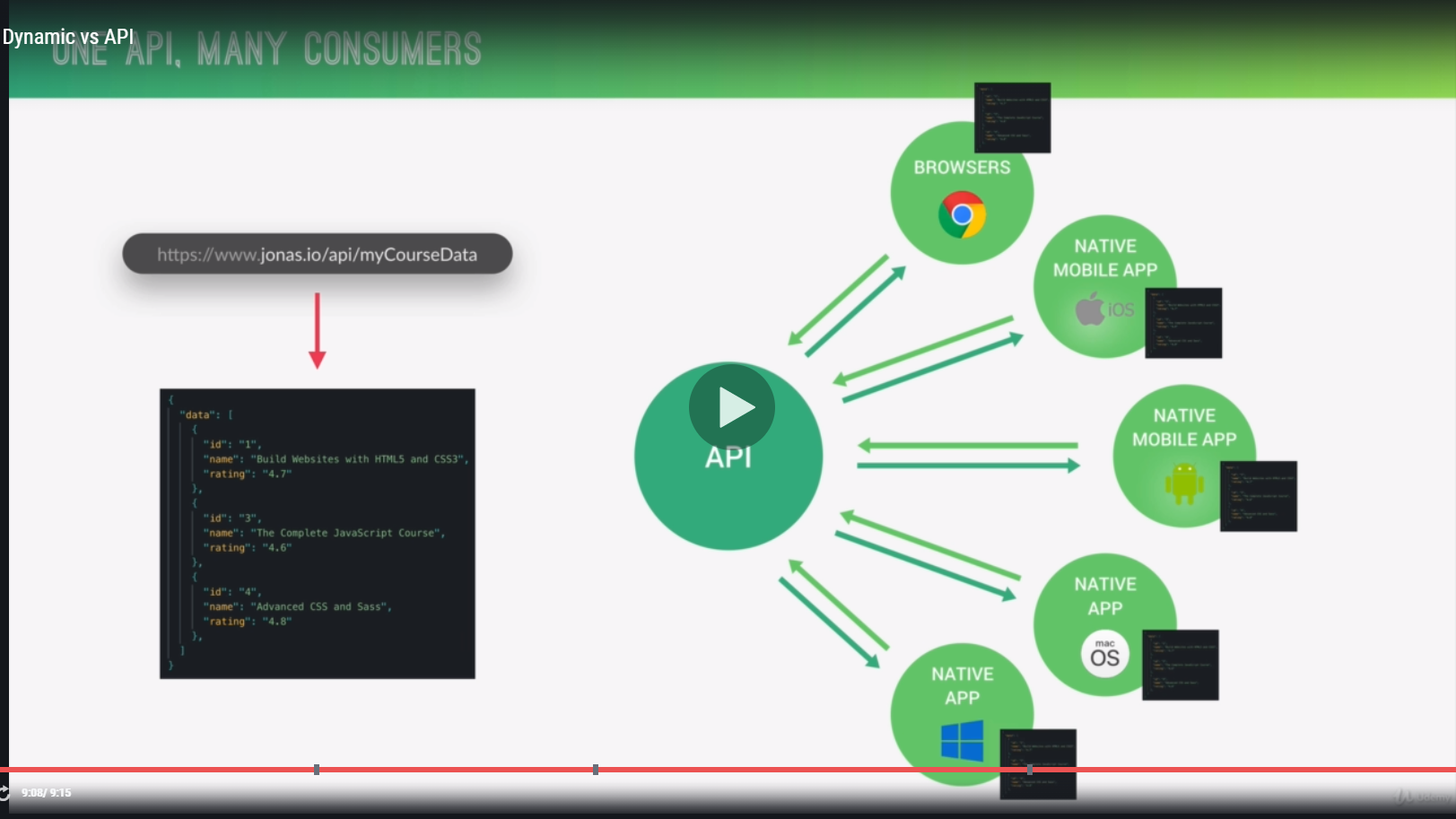
### 区分: 静态网站，动态网站，API



### 区分: 动态网站( 服务端渲染 )，API( 客户端渲染 )

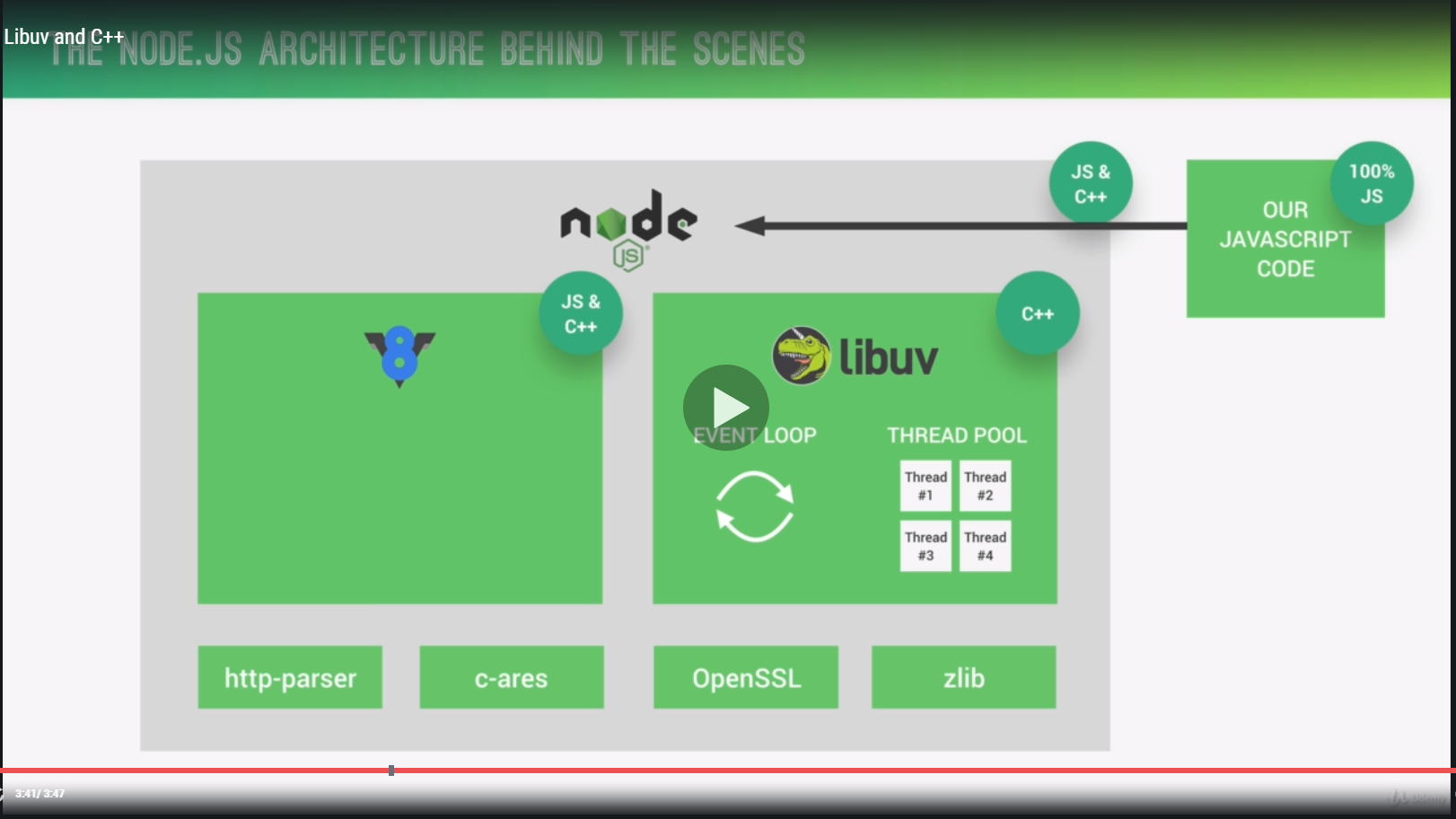


### API的巨大优势( 跨平台 – 甚至有公司专门只提供API服务来收益 )

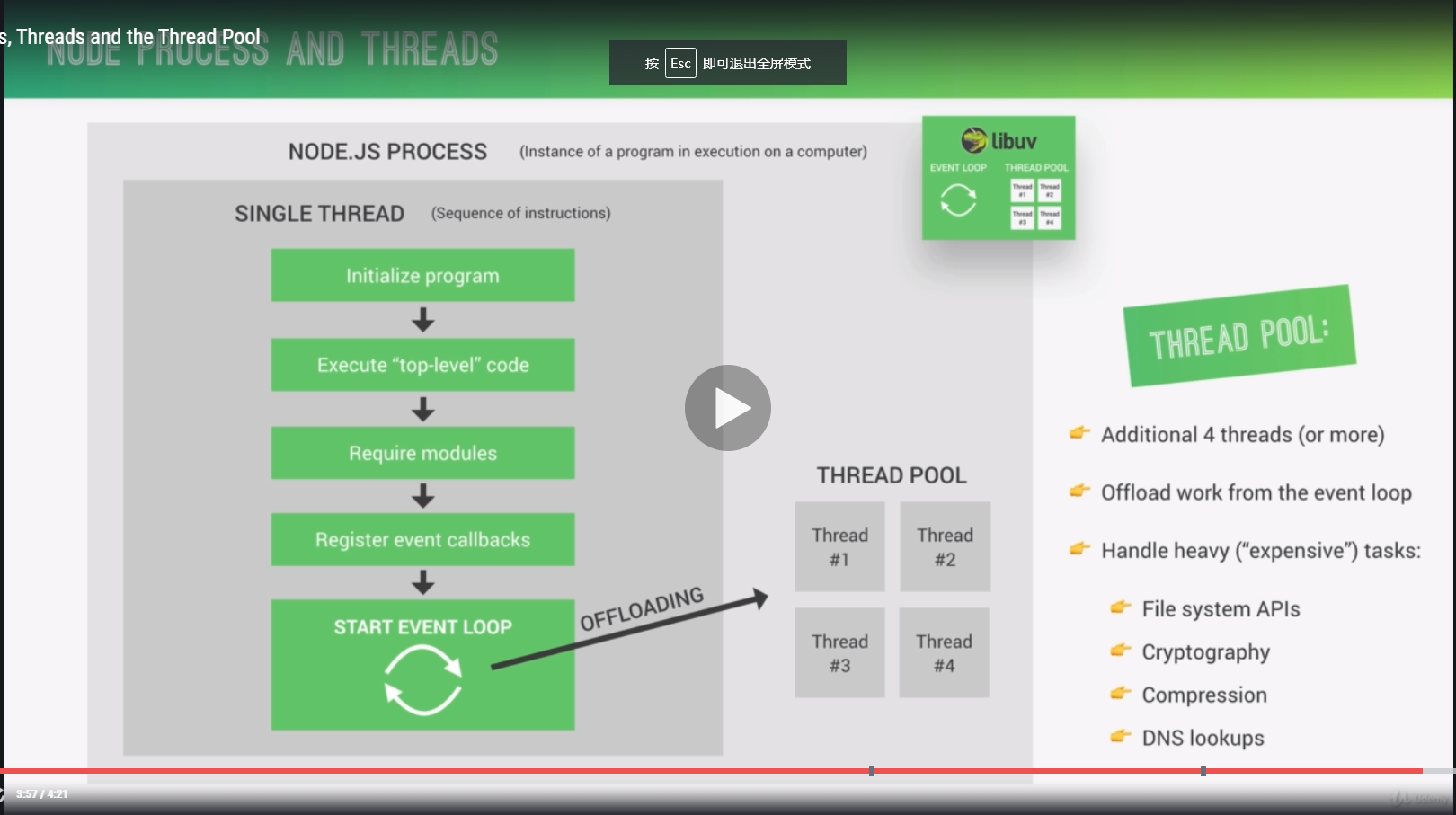


## Node.js底层( crypto )

### NODE.JS底层依赖



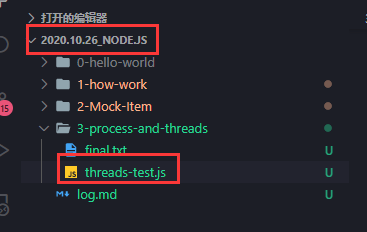
### 进程，线程，线程池



#### 改变线程数量

#### \* 事件循环细节

#### \* 线程测试



const fs = require("fs");

const crypto = require("crypto");

const start = Date.now();   // 方便统计时间

*/\*\**

\* 改变线程数量

*\*/*

*// a) 仅限mac平台使用:*

process.env.UV\_THREADPOOL\_SIZE = 1;

*/\*\**

\* 事件循环细节

*\*      a) setImmediate特殊的异步计时器( 等待研究 )*

*\*/*

setImmediate( () => console.log('setImmediate - 2') );

setTimeout( () => console.log('setTimeout - 1') );

fs.readFile( "./final.txt",() => {

    console.log(' 异步读取文件 - 4 ');

    console.log('-----');

    setTimeout( () => console.log('setTimeout - 5'));

    setImmediate( () => console.log('setImmediate - 6') );

    process.nextTick( ()=> console.log('nextTick - 7') );   // 微任务队列: 优先执行~

*/\*\**

\* 线程测试

*\*      a) 注意: crypto为加密库, 这里只用于测试异步*

*\*/*

    crypto.pbkdf2( "password", "salt", 100000, 1024, "sha512", () => {

        console.log(Date.now() - start ,' 测试线程 - 1 ')

    } );

    crypto.pbkdf2( "password", "salt", 100000, 1024, "sha512", () => {

        console.log(Date.now() - start ,' 测试线程 - 2 ')

    } );

    crypto.pbkdf2( "password", "salt", 100000, 1024, "sha512", () => {

        console.log(Date.now() - start ,' 测试线程 - 3 ')

    } );

    crypto.pbkdf2( "password", "salt", 100000, 1024, "sha512", () => {

        console.log(Date.now() - start ,' 测试线程 - 4 ')

    } );

    crypto.pbkdf2( "password", "salt", 100000, 1024, "sha512", () => {

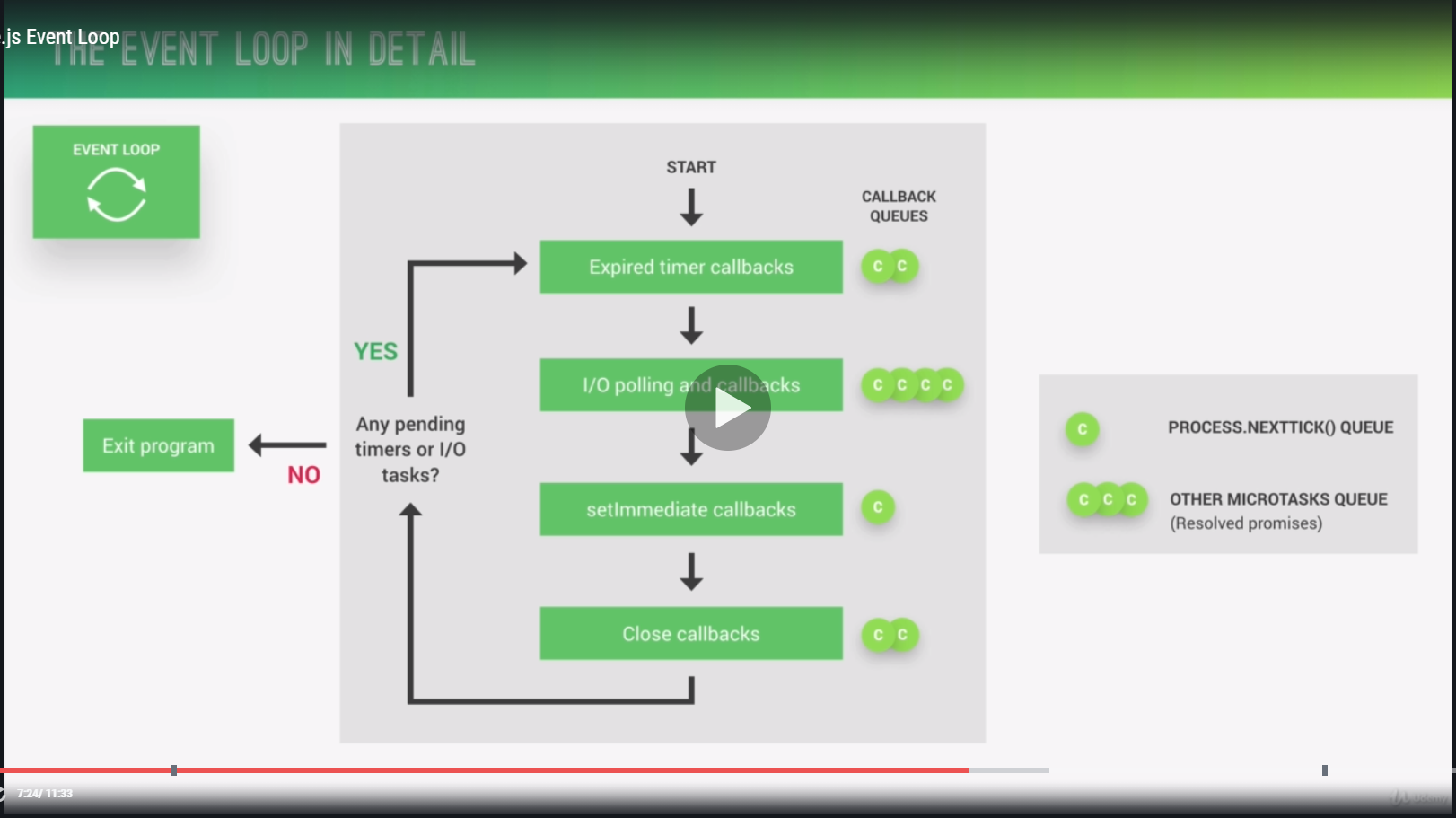
        console.log(Date.now() - start ,' 测试线程 - 4 ')

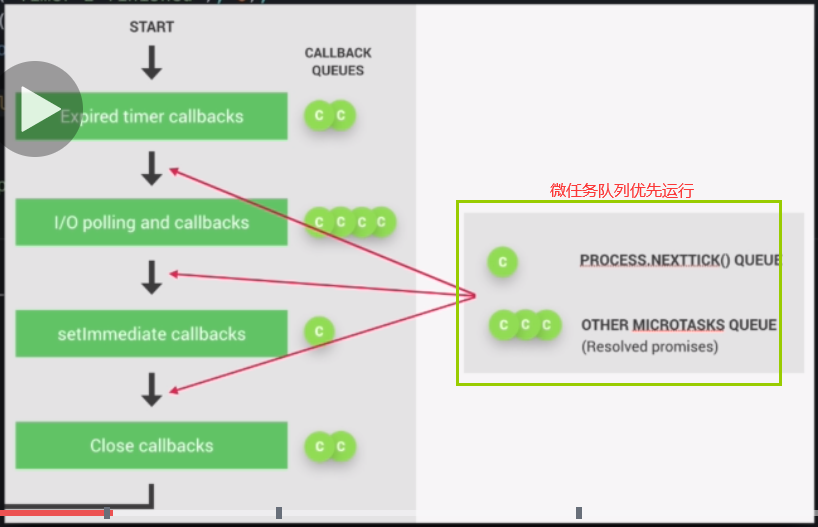
    } );

} );

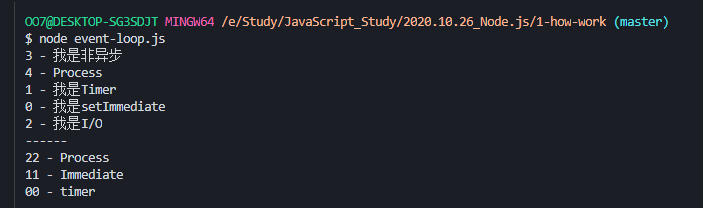
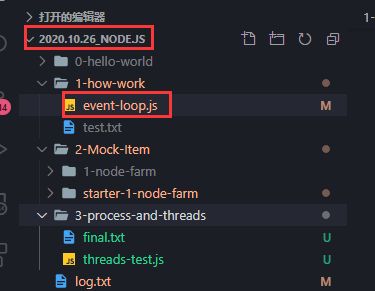
console.log("非异步 - 3");

### 事件循环，队列顺序





#### 异步实验



const fs = require("fs");

const { setImmediate } = require("timers");

*/\*\**

\* 异步实验

*\*      a) 默认异步顺序队列*

*\*          0. timer队列*

*\*          1. I/O队列( 虽然他位居第二，但是通常执行较为缓慢，实际情况要比Immediate慢 )*

*\*          2. Immediate队列*

*\*          3. Close Callback队列*

*\*          4. 判断是否存在，I/O, timer*

*\*          5. 不存在: 终止循环，退出程序*

*\*          6. 存在: 则继续循环*

*\*      b) 2个特殊队列( 要比默认队列，先执行 )*

*\*          0. process.nextTick()队列*

*\*          1. 微任务队列( somePromise异步函数 )*

*\*/*

setTimeout( ()=>{ console.log("1 - 我是Timer"); },0 );

setImmediate( () => { console.log('0 - 我是setImmediate') } );

fs.readFile("test.txt",()=>{

    console.log('2 - 我是I/O');

    // 注意: 此函数内乃是I/O的地盘

*//      a) 根据图表: ( 在I/O地盘时的步骤 )*

*//          0. I/O -->*

*//          1. Immediate -->*

*//          2. 结束无用的callback -->*

*//          3. 继续监听timer或者I/O -->*

*//          4. timer类型callback -->*

*//          5. I/O --> 由此循环*

    console.log('------');

    setTimeout(()=>{ console.log('00 - timer') },0);

    setImmediate(()=>{ console.log('11 - Immediate') });

    process.nextTick( () => { console.log("22 - Process") } );

});

console.log('3 - 我是非异步')

process.nextTick( () => { console.log("4 - Process") } );

// 结果是: 3,4,1,0,2,22,11,00