# 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');

# 2020.11.1（ node.js ）

## \* 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>`);

    }

} );

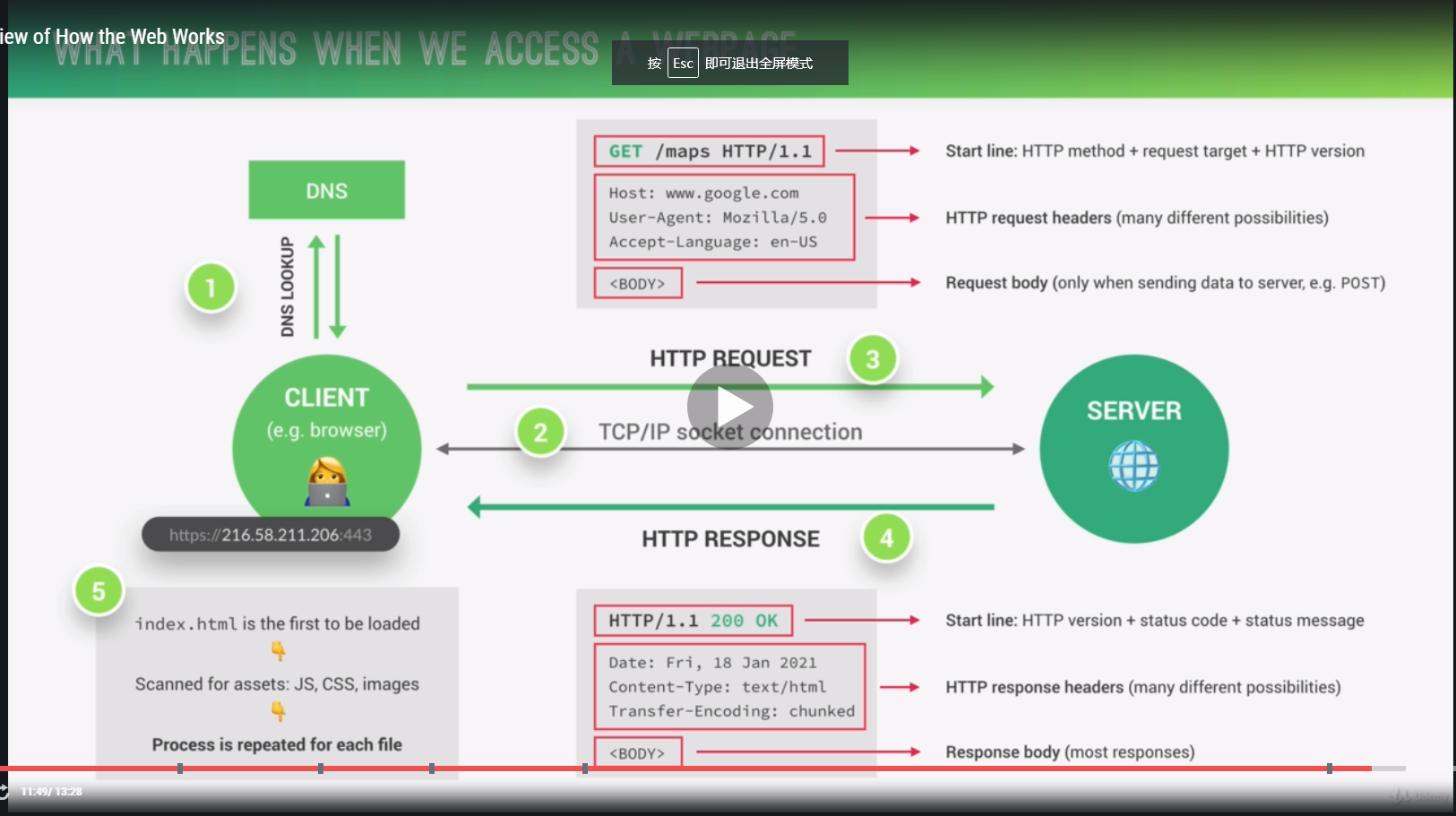
// c) 监听指定端口，促发server

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

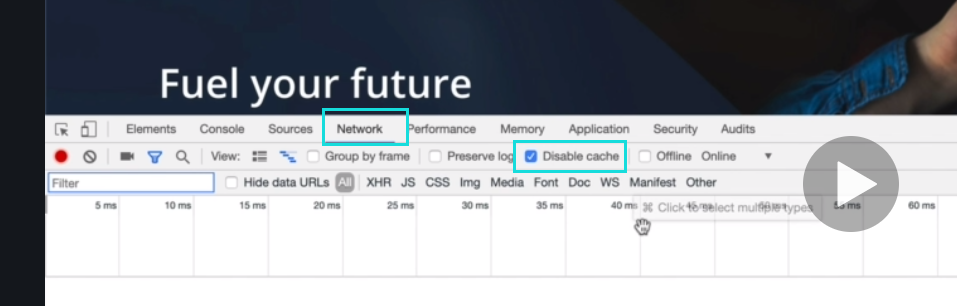
    console.log('监听端口3000');

} );

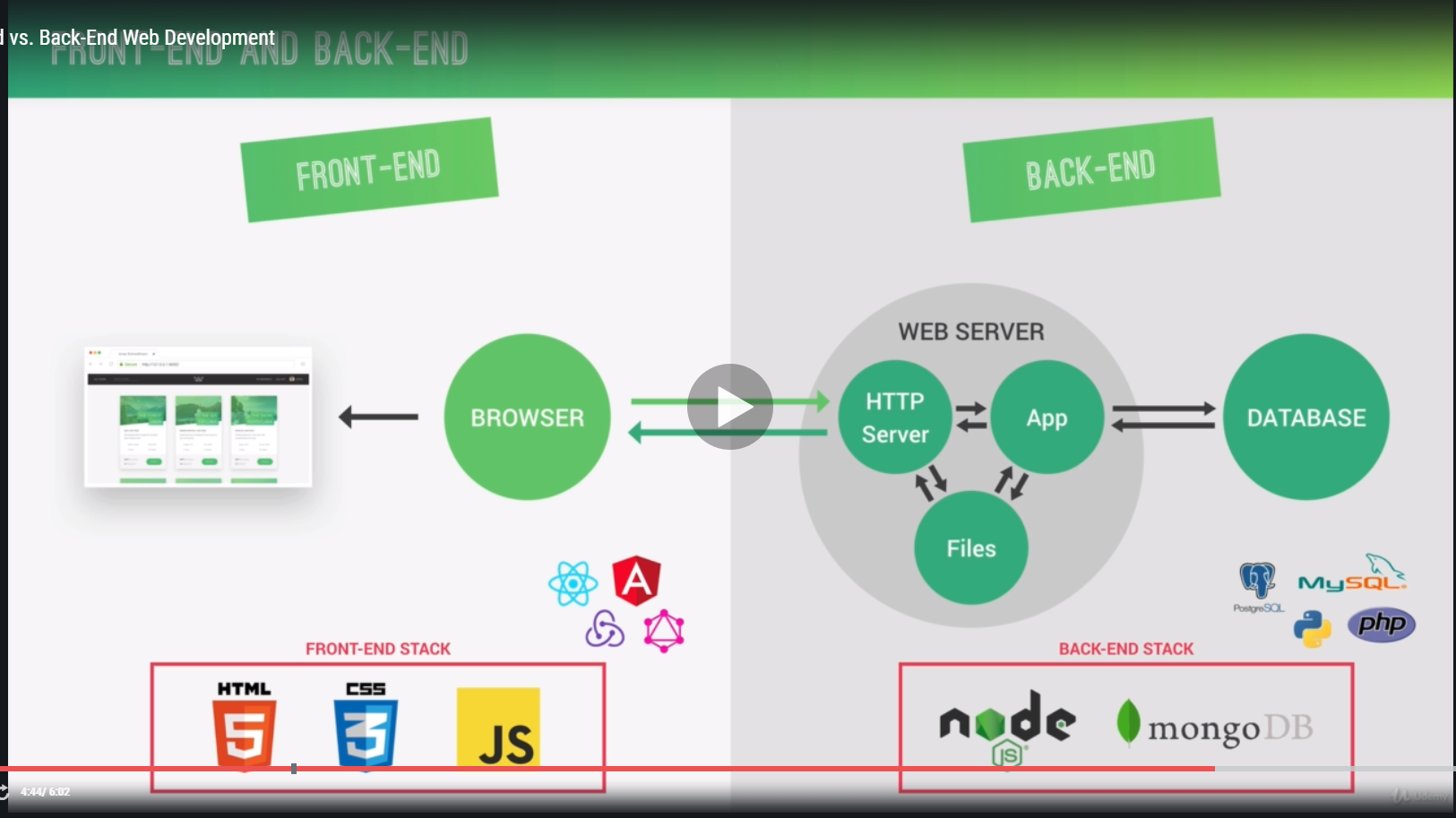
## 网络工作原理



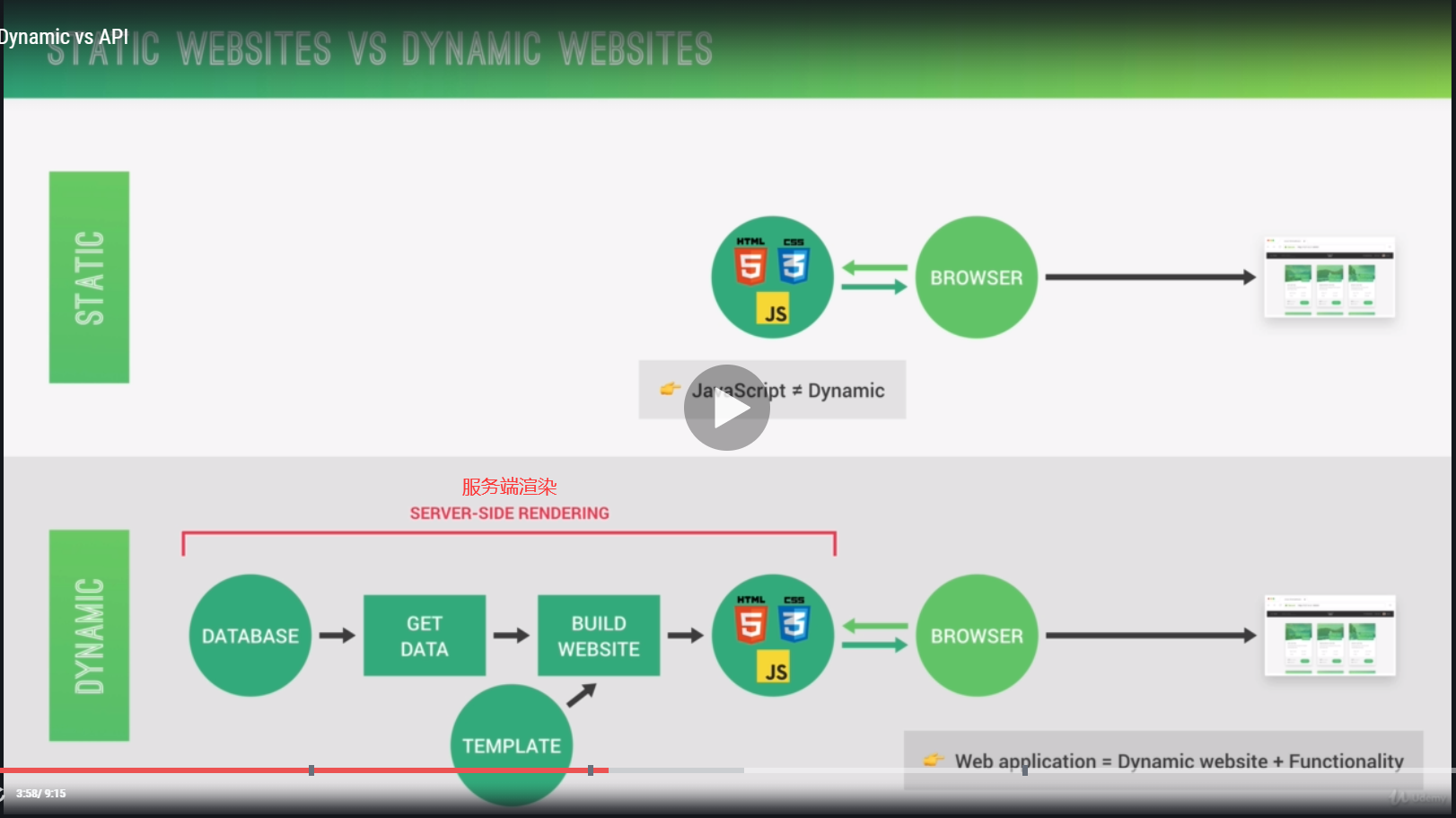
### http实战



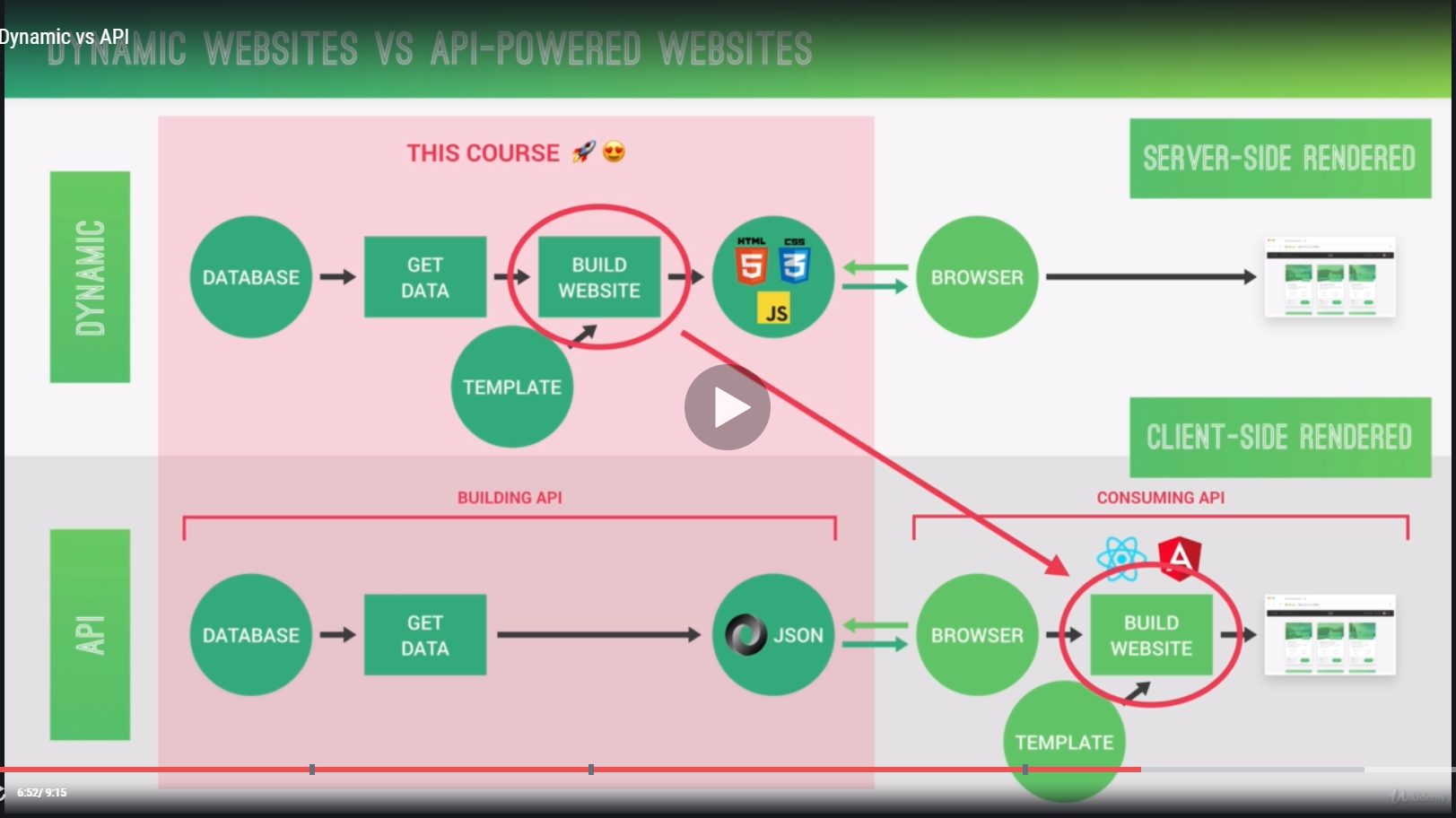
### Web服务的组成



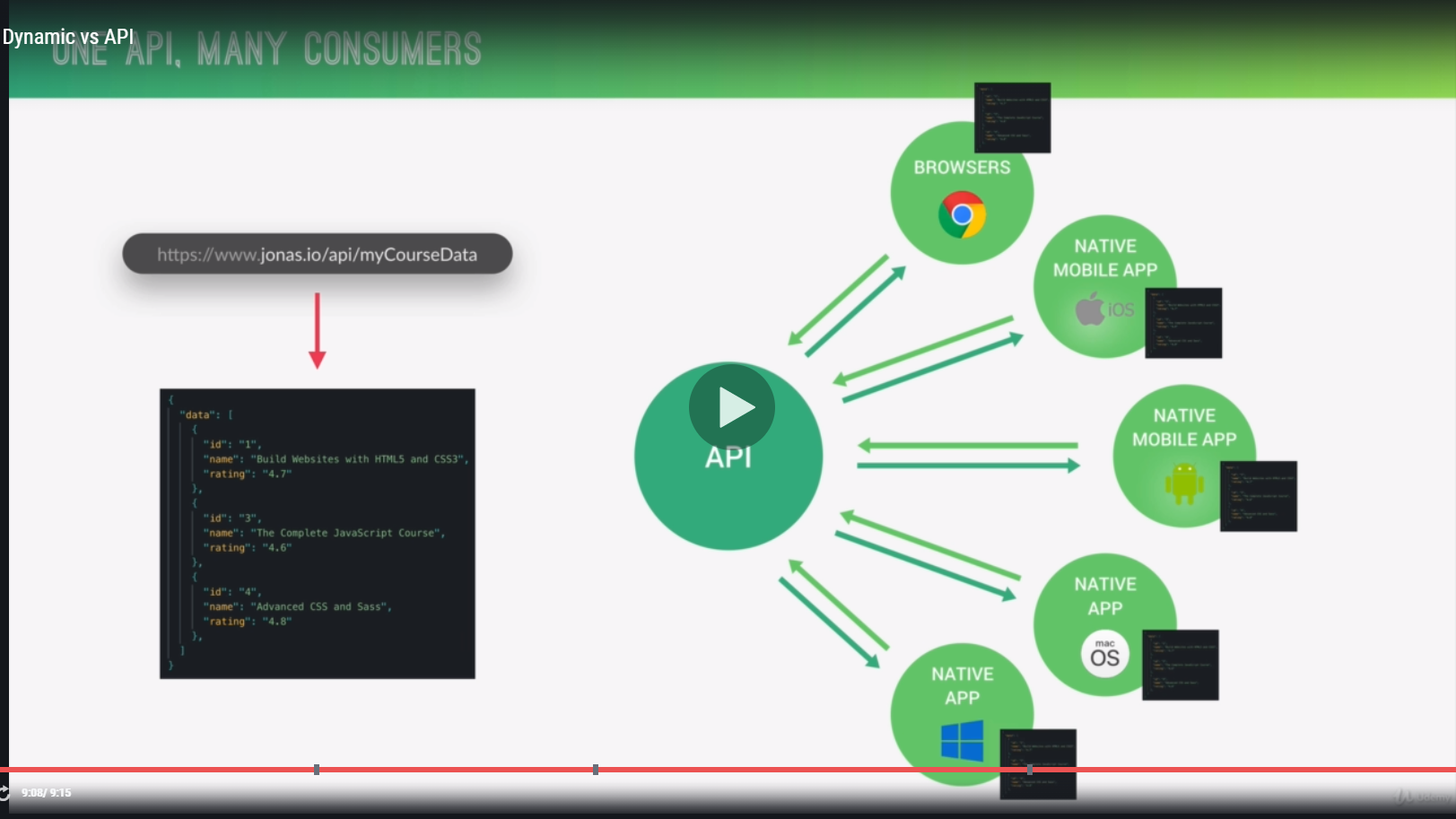
### 区分: 静态网站，动态网站，API



### 区分: 动态网站( 服务端渲染 )，API( 客户端渲染 )

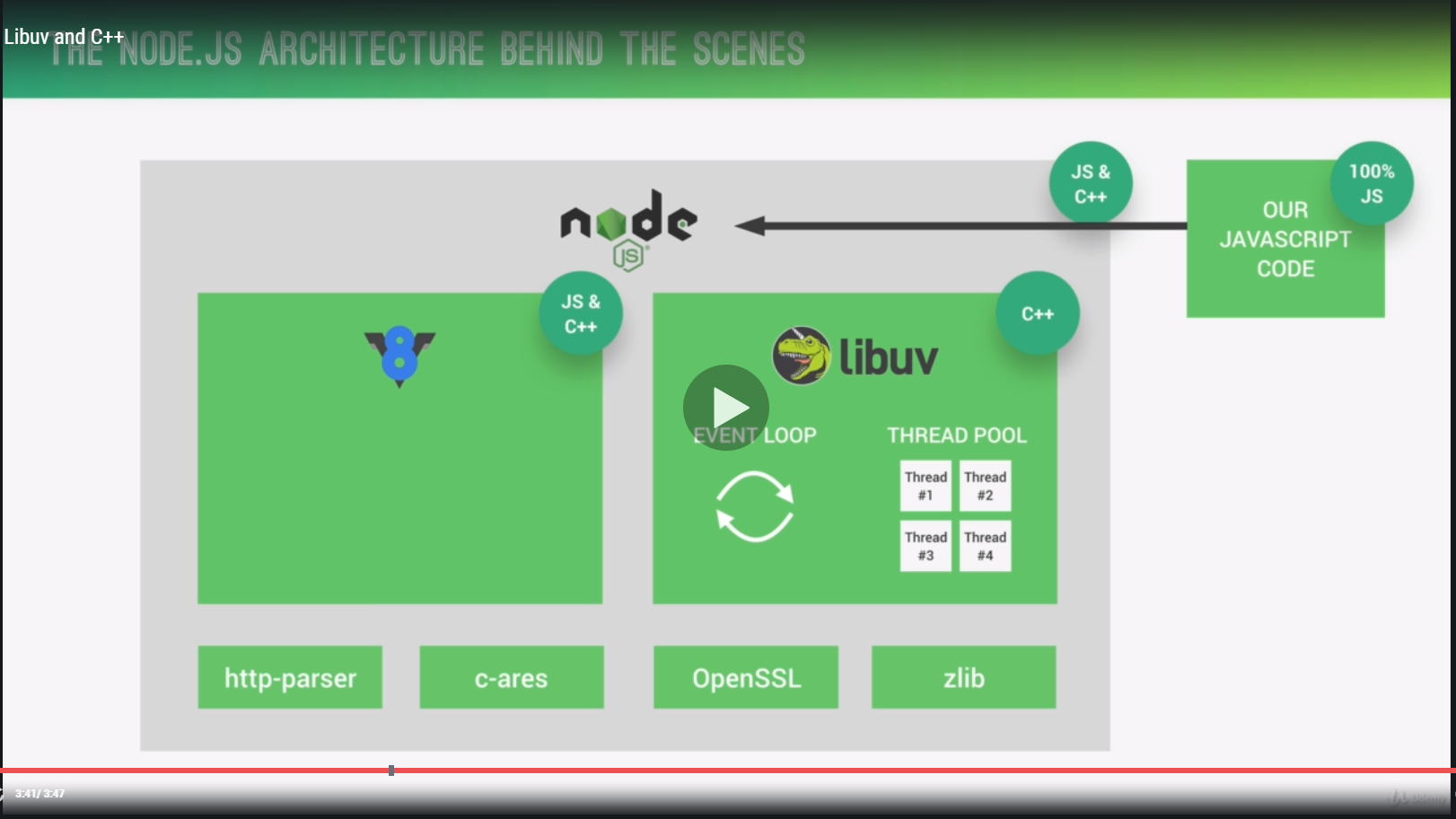


### API的巨大优势( 跨平台 – 甚至有公司专门只提供API服务来收益 )

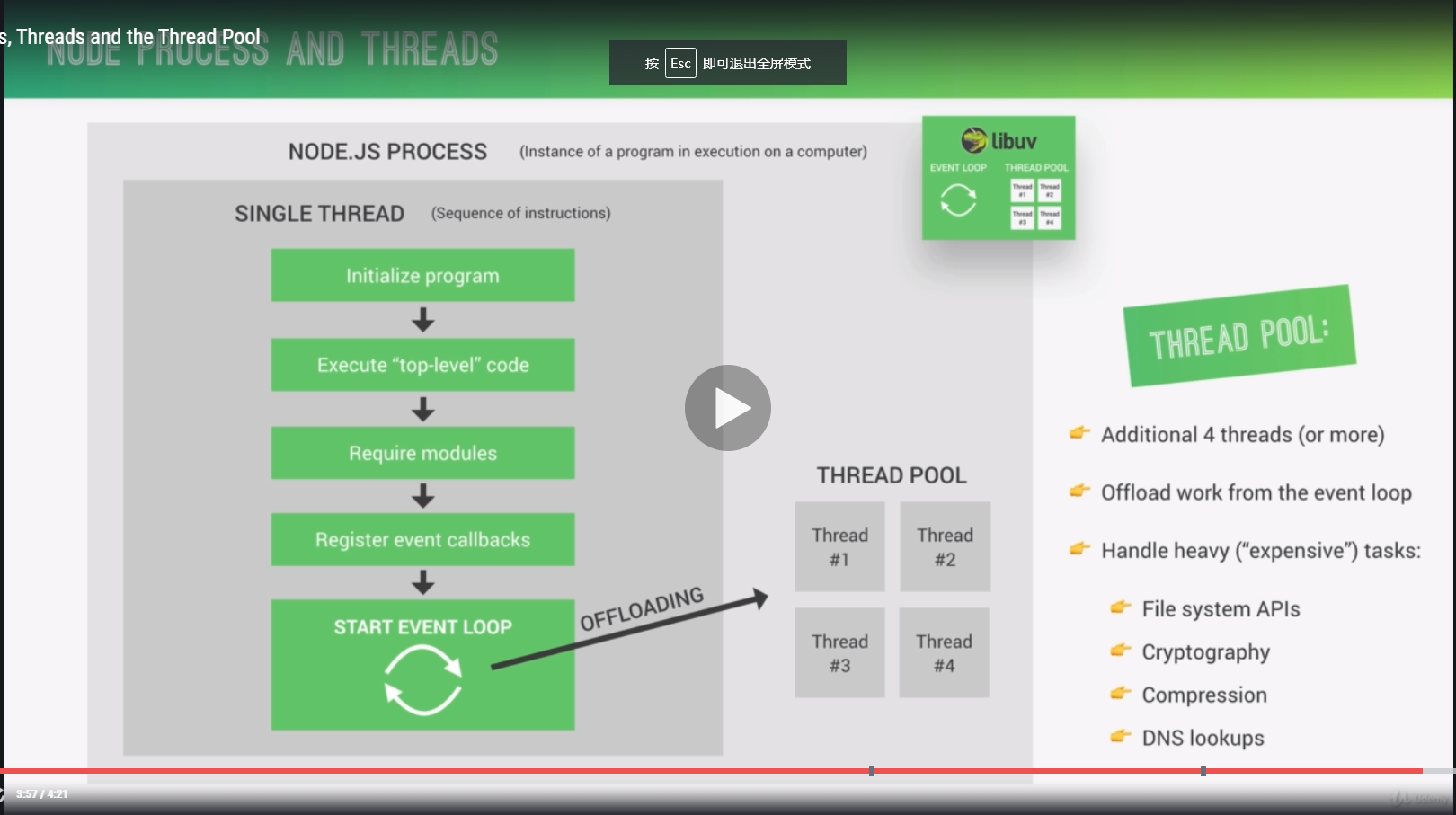


## Node.js底层

### NODE.JS底层依赖



### 进程，线程，线程池



#### 事件循环，队列顺序

