前言
node一小步 前端一大步

NODE

- 今天开始， 我们开始进行 `node` 的学习

什么是 node

- 第一个就要说什么是 `node.js`
    a. `node.js` 是一个基于 `Chrome V8` 的运行时环境

再浏览器里面，之所以能运行 js 代码，是因为浏览器都有一个 “引擎”，专门来解析 js 代码

而 node.js 就是把这个引擎单独拿出来了，当作一个 “软件” 给安装到我们的电脑里面

那么我们就不需要借助浏览器，就可以用我们的电脑来运行 js 代码

Chrome V8 是目前比较先进的解析引擎，也是 谷歌浏览器 使用的解析引擎

b. 使用 `JavaScript` 语言来进行后台开发

node.js 就是使用 JavaScript 这个语言来进行后端开发

换句话说，也就是使用 js 来书写服务端代码

c. `node.js` 是依靠事件驱动的

因为 js 是事件驱动语言，node.js 又是使用的 js 语言，所以 node.js 也是事件驱动的

我们的 js 有一个叫做 EventLoop（事件轮询）的机制，利用这个东西，可以高效而简介的进行开发

最简单的说 => 其实也就是我们的异步执行代码的机制

d. 非阻塞 I/O

I： input 表示输入流

O： output 表示输出流

非阻塞 I/O 也叫做 异步 I/O

就是利用了 js 的异步执行机制

把输入和输出变成一个异步行为，那么对于高并发的时候，是很好的一个处理方法

为什么要学习 node

- 接下来我们就来说一说为什么要学习 `node.js`

a. 了解后端开发的工作过程

学习完 node.js 以后，我们就能了解到前后台交互的整个过程

能理解在一个项目里面，前端都做了什么，后端都做了什么

在出现一些错误的时候，可以通过一些错误信息快速捕获到错误出现在哪里
我们自己的错误我们改，后端的错误，我们和后端交流让他们改
避免因为不了解错误出现在哪里，而耽误开发进度

b. 书写 `api` 接口

我们可以在自己写一些项目的时候，没有后端人员的情况下
自己书写后代代码

c. 全栈

我们想要进步成为一个 全栈工程师，那么必须要经历的一步就是后端的学习
之所以学习 node.js，就是因为他还是使用 js 语言
对于我们来说，相对学习成本就要低一些，不需要语法上的切换

在哪学习 node

官网地址 https://nodejs.org/en/
中文官网 http://nodejs.cn/

node 安装

- 去官网直接进行下载
- 下载好以后直接双击安装，一直 `next` 就可以了
- 安装完毕以后，我们打开命令行（终端）
  - `$ node --version`
  - 能出现版本号表示安装成功

node 相关版本介绍

- `node.js` 到目前为止一经有很多的版本了
- 偶数的版本，比如： `x.6.x` / `x.8.x` / `x.4.x` 表示稳定版本
- 奇数的版本，比如： `x.3.x` / `x.11.x` / `x.9.x` 表示非稳定版本

- LTS（Long Term Support）：表示长期支持的版

npm 安装

- npm 不需要我们单独安装，会在安装 node 的时候自动帮我们安装
- npm 是 node.js 的包管理工具，是全球最大的开源库生态系统
- 也就是帮我们下载第三方依赖使用的一个工具
- 打开命令行（终端）输入指令
  - $ npm --version
  - 出现版本号，表示 npm 安装成功，可以正常使用

npm 版本更新

- 安装 node 的时候，会默认安装 npm 的最新稳定版本
- 如果你以后想单独更新 npm 的时候，不需要从新安装 node
- 我们直接使用 npm 就可以把 npm 更新了
- 使用指令

```
# windows 操作系统
$ npm install --global npm
# OS 操作系统
$ sudo npm install --global npm
```

npm 插件官网

- npm 上有很多的插件
- 我们需要的时候可以直接下载就可以
- npm插件官网
- 对于某一个插件，你不是很了解他的用法，也可以直接去 npm 官网搜索
- 会有对应的说明文档
-

00 node特点（记住三句话）

1 天生单线程支持高并发

2 非阻塞IO模型 高并发性能数一数二（异步）

3 轻量和高效

补充 充分认识NODE的模块化特征 开发的工程化基础

+ node:

=> 每一个 js 文件都是相对独立的

=> 互相之间没有任何关系, 我们没有一个统一调配的东西

=> 我们只能靠模块化开发

模块化分类

1. 什么是模块

=> 一个 js 文件就是一个模块

=> 我们把一类方法放在一个 js 文件里面, 这个 js 文件就变成了一个模块

=> 再需要哪一类方法的时候, 引入这个 js 文件就好了

2. 什么是模块化

=> 再开发的过程中, 尽可能把开发方式趋近于模块的方式

=> 把我们所有的内容都按照类别分好文件

=> 按需引入

3. 模块化的分类

3-1. 自定义模块

-> 我们自己按照模块化的语法, 开发的 js 文件

-> 自己定义模块, 自己再需要的时候导入模块使用

-> 自己决定模块内哪些内容向外暴露

3-2. 内置模块

-> node 这个环境天生自带的一些模块("一个一个的 js 文件")

-> 你需要使用的时候, 直接引入使用就好了

3-3. 第三方模块

-> 其他人把一些常用的功能直接封装好

-> 做了一个开源

-> 我们使用的时候先下载下来, 直接导入按照人家的规则使用

01 深入了解NODE的模块化

```
// console.log(module)
/*
 Module {
 id: '.',
 //文件夹路径
 path: 'E:\\BK_GP_19\\01_第一周\\01_DAY\\01_代码\\02_自定义模块',
 // 我这个模块向外导出了哪些内容, 我这个文件里面哪些方法别的文件可以使用
 exports: {},
```

```
// 我被哪些模块导入了，谁再使用我的内容
 parent: null,
 //文件名路径
 filename: 'E:\\BK_GP_19\\01_第一周\\01_DAY\\01_代码\\02_自定义模块\\01_定义模块.js',
 //是否被引用
 loaded: false,
 // 我引用着哪些模块
 children: [],
 // 查找第三方模块的时候的顺序
 paths: [
 'E:\\BK_GP_19\\01_第一周\\01_DAY\\01_代码\\02_自定义模块\\node_modules',
 'E:\\BK_GP_19\\01_第一周\\01_DAY\\01_代码\\node_modules',
 'E:\\BK_GP_19\\01_第一周\\01_DAY\\node_modules',
 'E:\\BK_GP_19\\01_第一周\\node_modules',
 'E:\\BK_GP_19\\node_modules',
 'E:\\node_modules'
 ]
 }
*/
```

接下来我们了解下 三种导出方式

1. 标准导出

```

// 如果我想让 fn1 方法别人也可以使用
const fn1 = ()=>{
```

```
    console.log('我是fn1')
}
const num = 100
module.exports.fn1 = fn1
module.exports.num = num


 console.log(module.exports)

```

2. 给 module.exports 从新赋值

```
 const fn1 = ()=>{
    console.log('我是fn1')
}
const num = 100
 module.exports = {
 fn1: fn1,
 num: num
 }

  console.log(module.exports)

```

3. 为了让你少写单词 NODE 自带一句代码 var exports = module.exports 没方便多少说实话 程序员不怕多写 但是坑多了 面试也容易考 用的时候注意坑

```
 const fn1 = ()=>{
    console.log('我是fn1')
}
const num = 100
 exports.fn1 = fn1
 exports.num = num

  console.log(module.exports)

```

```
10
11
12
13
14
15  第二种就是最大的坑
16
17  const fn1 = ()=>{
18      console.log('我是fn1')
19  }
20  const num = 100
21
22
23  exports = {
24   fn1: fn1,
25   num: 100,
26
27   }
28
29  console.log(module)
30
31
```

讲了module.exports 这么多使用方式 和最大的坑 那么我们来实战导入导出 使用module.exports

```
1  //创建按a.js
2
3  const iceBox = '我是冰箱'
4
5  const tv = '我是电视'
6
7
8  function computer() {
9      console.log('我是一台电脑')
10 }
11
12
```

```
13
14 module.exports = {
15     iceBox:iceBox,
16     tv:tv,
17     computer:computer
18 }
19
20
21
22
23 //b .js
24
25 const a = require('./a')
26
27 console.log(a.tv)
28 console.log(a.computer)
29 console.log(a.iceBox)
```

我们正式用node启一个服务器

```
1  //1引入http模块
2  const http = require('http')
3
4  //2 创建web服务 返回http对象
5  const app = http.createServer((request,response)=>{
6   console.log('有人进来了')
7   response.write('hello world')
8   response.end()
9  })
10
11
12 //3监听端口
13 端口号写1-65535都可以 1024以下频繁被系统占用
14
15 app.listen(3000,'localhost',()=>{
```

```
16  console.log('3000端口开启了')
17 })
```

概念

web的内容都是存储在web服务器上面的。而web服务器通常使用的是HTTP协议，因此web服务器也被称为HTTP服务器。 客户端和web服务器之间进行通信，通常需要经历请求和响应两个过程，简单说，就是客户端向服务器发送一个请求索要数据，而服务器端需要对请求作出响应，即把客户索要的数据返回。

具体流程是 描述就是

1 按下回车时浏览器根据输入的URL地址发送请求报文给放服务器

2 服务器接收请求报文，会对请求报文进行处理

3 服务器将处理完的结果通过响应报文返回给浏览器

4 浏览器解析服务器返回的结果，将结果显示出来

以前web服务器第一选择是PHP 现在是NODE 因为曾经JS不能运行后端 现在可以了 抢占了PHP 的地位 PHP 开发过FACEBOOK 官网 。

02 安装自动重启工具nodemon

```
1 npm i nodemon 可以自动重启代码 省去繁琐来回重启
```

03 解决node服务器中文乱码的问题

```
1 解决中文乱码问题就必须要求前端 用utf8解析
2
3 //1引入http模块
4 const http = require('http')
5
6 //2 创建http服务对象
7 const app = http.createServer((req,res)=>{
```

```
// 状态码 和响应头
res.writeHead(200,{"Content-Type":"text/html;charset=utf8"})
 res.write('hello world11')
 res.write('中国')
 res.end()
})


//3监听端口

app.listen(3000,()=>{
 console.log('3000端口开启了')
})


```

04 解决node服务器中文乱码的问题2 细节问题

重点概要

状态码 以及HTTP状态的深度理解

Response 消息中的第一行叫做状态行 ， 由HTTP协议版本号，状态码，状态消息三部分组成

状态码用来告诉HTTP客户端 HTTP服务器是否产生了预期的响应，处理结果的代号

1。 1xx 提示信息 表示请求已经被成功接收 继续处理

2。 2xx 表示成功接收 理解 接受 本案例 200 是一切ok 请求成功完全 请求资源被发送回客户端

3。 3xx 重定向 要完成请求必须进一步的处理

4。 4xx 客户端错误 请求语法错误或者无法实现

Content-Type是HTTP协议 header中一个重要的参数，它用于标识发送或接收到的数据的类型，浏览器根据该参数来决定数据的打开方式。
为了支持多媒体数据类型，HTTP协议中就使用了附加在文档之前的MIME数据类型信息来标识数据类型，它使得HTTP传输的不再是普通的文本，让网页内容变得丰富多彩。
“主类型”（type）主要有以下几种：
\1. text：用于标准化地表示的文本信息，文本消息可以是多种字符集和或者多种格式的； 默认是text/plain；
\2. multipart：用于连接消息体的多个部分构成一个消息，这些部分可以是不同类型的数据； 默认是multipart/mixed；
\3. application：用于传输应用程序数据或者二进制数据； 默认是application/octet-stream；
\4. message：用于包装一个E-mail消息；
\5. image：用于传输静态图片数据；
\6. audio：用于传输音频或者音声数据；
\7. video：用于传输动态影像数据，可以是与音频编辑在一起的视频数据格式。
\8. drawing：--未整理

---

05 NODE也可以返回模板

```
//1引入http模块
const http = require('http')

//2 创建http服务对象
const app = http.createServer((request,response)=>{
 console.log('有人进来了')
 //这句话是什么意思 1 给前端返回http200状态啊 表示正确返回 2 让前端以charset=utf8解析
```

```
response.writeHead(200,{"Content-Type":"text/html;charset=utf8"})
response.write(`<!DOCTYPE html>
 <html lang="en">
 <head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
 </head>
 <body>
 <h1>你好 世界</h1>
 </body>
 </html>`)
 response.end()
})


//3监听端口

app.listen(3000,()=>{
 console.log('3000端口开启了')
})

```

06 fs模块

因为 node的全局是电脑上的 比浏览器级别高 所以可以用来磁盘操作 文件操作

node内置模块

```
fs.readFile('文件路径',[编码方式],(err,data)=>{})

例子

fs.readFile("./html/index.html","utf8",(err,data)=>{
console.log('err',err) //有错err不为null
```

```
 console.log('data',data) //默认二进制 buffer 必须加 utf8
})
```

1 fs异步代码报错后面会发生什么（这个要了解清楚涉及到NODE的设计理念和模式）

```
fs.readFile("./html/index.html","utf8",(err,data)=>{
 if(err) {
 console.log(err)
 }
 console.log(data)

})


console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
```

2 fs同步代码报错后面会发生什么（这个要了解清楚涉及到NODE的设计理念和模式）

```
 let data = fs.readFileSync('./html/index.html',"utf8")
 console.log('data',data)
 console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
```

```
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')
```

3 fs同步代码报错后 如何处理 如何让程序变得更健壮

```
const fs = require('fs')
try {
    let data = fs.readFileSync('./html/index.html',"utf8")
    console.log('data',data)
} catch (error) {
    console.log(error)
}


 console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')
 console.log('我还在继续执行噢')

```

07 fs其他 方法

```
//改名
fs.renameSync("./html/index.html","./html/index2.html") //同步

```

```
fs.rename("./html/index2.html","./html/index3.html",err=>{ //异步

 })



 3. writeFile()
 => 异步写入文件
 => 语法: fs.writeFile(路径, 要写入的内容, 回调函数)
 -> 路径: 你要写入文件的路径, 如果路径文件不存在, 会创建一个这个文件再写入
 -> 写入的内容: 你自己定义
 -> 回调函数: 写入成功以后执行的函数, 必填
 => 注意: 完全覆盖式的写入

 // 3. writeFile()
 // fs.writeFile('./test.txt', '你好 世界', function () {
 // console.log('写入完成')
 // })
 4. writeFileSync()
 => 同步写入文件
 => 语法: fs.writeFileSync(路径, 要写入的内容)
 => 注意: 完全覆盖式的写入

 // 4. writeFileSync()
 // fs.writeFileSync('./test.js', 'hello node')
 // console.log('写入完成')
 5. appendFile()
 => 异步追加内容
 => 语法: fs.appendFile(路径, 追加的内容, 回调函数)
 -> 路径: 写入文件的路径, 没有该路径, 自己创建
 -> 追加的内容
 -> 回调函数: 必填

 // 5. appendFile()
 // fs.appendFile('./data1.txt','hello\n',(err)=>{
 // if(err) return console.log(err);
 // console.log('写入成功');
 // });
 6. appendFileSync()
 => 同步追加内容
 => 语法: fs.appendFileSync(路径, 追加的内容)
```

```
// 6. appendFileSync()
// fs.appendFileSync('./test1.txt', '你好 世界')
7. readdir()
=> 异步读取文件夹
=> 语法: fs.readdir(路径, 回调函数)

// 7. readdir()
// fs.readdir('../02_自定义模块', (err, data) => {
// if (err) return console.log(err)

// console.log('读取文件夹成功')
// console.log(data)
// })

8. readdirSync()
=> 同步读取文件夹
=> 语法: fs.readdirSync(路径)

// 8. readdirSync()
// const res = fs.readdirSync('../02_自定义模块')
// console.log(res)
9. mkdir()
=> 异步创建文件夹
=> 语法: fs.mkdir(路径, 回调函数)

// 9. mkdir()
// fs.mkdir('./a', (err) => {
// if (err) return console.log(err)

// console.log('创建文件夹成功')
// })

10. mkdirSync()
=> 同步创建文件夹
=> 语法: fs.mkdirSync(路径)

// 10. mkdirSync()
// fs.mkdirSync('./b')
11. rmdir()
=> 异步删除文件夹
=> 语法: fs.rmdir(路径, 回调函数)
```

```

// 11. rmdir()
// fs.rmdir('./a', err => {
// if (err) return console.log(err)
// })
12. rmdirSync()
=> 同步删除文件夹
=> 语法: fs.rmdirSync(路径)
// 12. rmdirSync()
// fs.rmdirSync('./b')
13. unlink()
=> 异步删除文件
=> 语法: fs.unlink(路径, 回调函数)

 // 13. unlink()
 // fs.unlink('./test1.txt', err => {
 // if (err) return console.log(err)
 // })

 14. unlinSync()
 => 同步删除文件
 => 语法: fs.unlinkSync(路径)

 // 14. unlinSync()
 // fs.unlinkSync('./test123.txt')



```

URL模块的学习

```
//一共里面也没有几个方法



/*
 url 内置模块
 + node 自带的一个内置模块
```

```
 + 里面封装的都是和 url 地址栏相关的方法
 + 使用的时候直接导入使用

 1. parse()
 + 解析 url 地址的方法
 + 语法: url.parse(url 地址, 是否解析查询字符串)
 -> 地址: 要解析的地址
 -> 是否解析查询字符串: 默认是 false, 选填 true
 => 会把地址里面的查询字符串解析成一个对象的形式
 + 返回值: 是一个对象, 里面包含整个 url 地址的所有信息
*/
/*
 url 内置模块
 + node 自带的一个内置模块
 + 里面封装的都是和 url 地址栏相关的方法
 + 使用的时候直接导入使用

 1. parse()
 + 解析 url 地址的方法
 + 语法: url.parse(url 地址, 是否解析查询字符串)
 -> 地址: 要解析的地址
 -> 是否解析查询字符串: 默认是 false, 选填 true
 => 会把地址里面的查询字符串解析成一个对象的形式
 + 返回值: 是一个对象, 里面包含整个 url 地址的所有信息
*/









// 0. 导入 url 模块
const url = require('url')

// 1. parse()
// const res = url.parse('http://www.name.com:8080/a/b/c/hello.html?a=100&b=200#abc', true) //解析查询字符串
// const res = url.parse('http://www.name.com:8080/a/b/c/hello.html?a=100&b=200#abc') //不解析查询字符串
// console.log(res)
```

```
/*
 {
 //传输协议
 protocol: 'http:',
 //表示是否有斜线:true
 slashes: true,
 // 作者无
 auth: null,
 //表示主机:www.name.com:8080
 host: 'www.name.com:8080',
 //端口号
 port: '8080',
 //表示主机name
 hostname: 'www.name.com',
 // 哈希值
 hash: '#abc',
 //指的是后#前的内容，包含？:?query=string
 search: '?a=100&b=200',
 //查询字符串 少了?比search
 query: 'a=100&b=200',
 //路径名
 pathname: '/a/b/c/hello.html',
 // 路径标识符，前后端配套的一个标识符(暗号)
 path: '/a/b/c/hello.html?a=100&b=200',
 //代表未解析的url地址
 href: 'http://www.guoxiang.com:8080/a/b/c/hello.html?a=100&b=200#abc'
 }
*/

```

08 了解express 框架

官网 https://www.expressjs.com.cn/

```
了解 express 框架
 + 后端框架之一
 + 框架 库 和 插件
 => 插件：为了实现某一类功能而封装的代码
```

```
=> 库：把所有的基础操作全部封装，需要任何东西需要自己组装
=> 框架：一套完整的自己的生态体系，能帮你把大部分事情全部做完了
+ node 框架
=> 自己把所有的后端需要做的事情给你准备好
=> 把大部分的行为都封装成了方法(服务)
 => 预留了一个接口位置，你可以把很多的插件直接注册进去

 npm install express --save
```

09 expess 创建服务器 精炼代码

```
//express框架就像精装修 智能化的房子 让你很舒服 让你专注于业务逻辑

//1 导入框架使用
const express = require('express')


// 2 创建服务

const app = express()


//监听端口

app.listen(8080,()=>{
 console.log('running at port 8080')
})

```