DAY31 node 和模块化

2023年5月30日 16:41

NODE概述

一.NODE的学习目标

1.NODE对后期环境配置起到核心的作用。

2.协作的工具。

3.NODEJS编写后端，这个需求不够大。

4.理解前后端的交互行为。

5.根据上课强调的内容进行学习(了解，熟悉)

二.NODE概述

中文网站：https://nodejs.cn/

1.什么是 Node.js

Node.js是一个基于 Chrome V8 引擎的 JavaScript 运行环境。

2.这个是 node 官网的解释

其实 node 就是用 javascript 语言写后端

也就是说，我们的 javascript 有了 node 以后，不光是一个前端语言，也是一个后端语言

3.前端JavaScript和后端JavaScript的区别

3.1.前端javascript

三大核心

- ECMAScript
- DOM
- BOM

操作内容

- 浏览器
- 解决兼容问题

3.2.后端的javascript

核心

- ECMAScript

操作内容

- 后端代码
- 数据库

三.安装NODE软件

下载

安装

测试 node -v npm -v npm是node自带的包(插件，组件，模块)管理工具，安装node时自动安装npm

四.NODE初体验

1.直接在终端中书写 js 代码

启动命令行工具(DOS命令，LINUX命令)

node 回车

可以编写js代码，维护不方便

两次ctrl+c退出

2.在命令行运行一个 js 文件

创建js文件

编写对应的js代码

利用node命令执行对应的文件

DOS和LINUX命令

一.DOS和LINUX命令

window系统默认的DOS命令(磁盘操作系统)

mac系统默认LINUX命令 - 重点

其实就是在命令行使用指令来操作我们的电脑;

因为我们的 node 就是在命令行运行 js，所以我们要了解一些常用的命令行指令

注意：

如果你是mac电脑，直接使用linux命令,忽略DOS命令，

但是如果是window系统的电脑，必须安装git软件，学习linux命令和DOS命令

核心就是linux命令，了解dos命令

二.DOS命令

1.启动DOS命令窗口：win + r 弹出运行框 输入cmd 回车启动(快捷方式：通过盘符的路径进入对应的位置)

注意：进入对应的目录，直接在地址栏输入cmd，在当前位置启动

2.进入或者退出任意目录

- 进入对应的系统盘: 盘符: 例如：d: e:
- 进入对应的目录： cd 目录的路径 例如：cd phpstudy_pro/www/JS2207
- 退出目录：每退出一层使用一次：cd.. 例如：cd.. 退出一层，继续cd..
- 显示目录内容列表：dir
- 退出命令窗口： exit或者直接叉掉
- 查看本地的ip地址：ipconfig/(ipconfig/all)
- DOS窗口写过了很多命令，清屏，重新编写：cls

3.了解命令

- 创建目录： mkdir 目录名称
- 删除目录： rmdir 目录名称
- 创建文件并且写入内容：copy con 文件名和扩展名 - 回车写入内容- ctrl+z 创建完成
- 删除文件： del 文件名称

- 注意：删除的目录里面如果存在文件，先删文件，再删目录。
- 注意：如果当前的操作结束了，一定会出现盘符路径，方便下一次操作。

三.linux命令 - 重点关注

1.启动Linux命令行工具

- mac系统直接[终端]启动
- window系统需要安装git工具

2.安装git，推荐默认路径(c盘)，因为vscode里面终端直接编写DOS和Linux命令

- 一路next，直接安装完成
- 安装完成，任意目录位置右键启动 Git Bash Here

3.linux命令行学习

3.1.window启动linux命令

- 任意目录位置右键启动 Git Bash Here，弹出命令框，ctrl+滚轮控制字体大小。

3.2.进入或者退出任意目录

- 进入对应的系统盘: cd 盘符:　　例如：cd d:
- 进入对应的目录：　cd 目录的路径　例如：cd phpstudy_pro/www/JS2201
- 退出目录：每退出一层使用一次：cd .. 例如：cd .. 注意：cd和..之间多一个空格
- 显示目录内容列表：dir/ls　推荐使用：ls
- 退出命令窗口：　exit或者直接叉掉
- 查看本地的ip地址：ipconfig/ipconfig/all
- Linux窗口写过了很多命令，清屏，重新编写：clear

表示查看访问 百度 网站的速度： ping www.baidu.com
表示查看当前电脑的信息： systeminfo
表示在当前目录下创建一个叫做 test 的文件夹： mkdir test

4.了解命令

- 创建目录：　mkdir 目录名称
- 创建文件：　touch 文件名称
- 打开文件：　cat 文件名称
- 删除目录和文件：　rm -rf 目录或者文件名称
- 注意：删除的目录里面如果存在文件，先删文件，再删目录。

四.终端 - vscode终端

1.利用vscode开发工具里面的终端进行命令的编写。
2.利用终端配置选择命令

- git bash - linux命令
- powershell - 有一些环境需要配置
- command prompt - dos命令

3.终端可以同时开启多个，可以删除

4.左侧面板的任意文件都可以右键终端打开，方便操作。

五.注意事项：

1.编写的命令通过方向键回滚(继续通过方向键让他显示)。

2.tab键有自动补全功能。

3.目录位置启动命令。

4.所有的电脑都需要学习linux命令

规范

模块化语法？

1.CommonJS

CommonJS:2009年诞生最早的模块规范，基于服务器端规范，NodeJS实现了这一规范.

CommonJS约定的规则：

一个js文件就是一个独立模块

定义模块

调用模块

配置模块

2.AMD

AMD:异步的模块定义，基于commonJS的讨论诞生的，浏览器端的异步规范，require.js实现了这个规范

3.CMD

CMD:通用的模块定义，国内的规范，sea.js实现了这个规范

4.ES6 Module

ES6在语言标准的层面上，实现了模块功能，而且实现得相当简单，完全可以取代 CommonJS 和 AMD 规范，成为浏览器和服务器通用的模块解决方案。

export export default 导出模块 定义模块

import 导入模块，调用模块

注意：模块化的意义就是解决冲突和依赖

注意：包，模块，插件 都是类似的意义

Node模块化开发

ConmmonJS规范 - node模块化

1.模块化

在 node 的开发过程中

我们是把每一个功能独立做成一个模块（一个文件就是一个模块）

然后在使用 导入导出（定义和调用） 的方式把他们关联在一起

利于维护

准确定位

2.CommonJS约定的规则：

一个js文件就是一个独立模块

定义模块

调用模块

配置模块

3.我们一般把模块分为三种

自定义模块：我们自己写的文件

内置模块：node天生就带有的模块

第三方模块：从网上下载的别人写好的模块

模块的写法

一.自定义模块开发方式

1.定义模块，导出模块

2.调用模块，导入模块

3.配置模块

每一个 js 文件都会有一个对象叫做 module

在 module 里面有一个成员，叫做 exports

每一个 js 文件会默认把 module.exports 导出

也就是说，我们向 module.exports 中添加什么内容

那么就会导出什么内容

对比ES6模块写法

1.ES6

1.1.导出模块

export导出模块 利用import导入模块必须知道导出内容相关名称

define.js

```
const obj = { a: 1 };
const arr = [1, 2, 3];
export { obj, arr };
```

export default导出一个变量或者函数，调用的时候可以自定义名称，一个模块只能有一个默认输出。

define1.js

```
const res = { a: 1 };
export default res;
```

1.2.导入模块

use.js

```
import { obj, arr } from "./define.js";
import hehe from './define1.js'
```

2.nodejs

1.1.导出模块

逐个导出

```
module.exports.name = "zhangsan";
module.exports.age = 18;
module.exports.sex = "男";
```

整体导出

```
module.exports = {
  name: "lisi",
  age: 20,
  sex: "女",
};
```

1.2.导入模块

```
const obj = require("./define");  //省略js扩展名
```

二.内置模块：Nodejs天生具有的模块

1.直接导入即可使用。

2.学习模块的api(说明文档)

3.学习常用的一些模块

fs模块

path模块

url模块

http模块

querystring模块

三.第三方模块

第三方模块就是别人或者别的公司写的模块

必须先安装(下载)，再使用

利用npm 命令安装第三方模块　比如：npm i request

默认生成一个node_modules目录，里面安装的第三方模块

默认生成一个配置文件package.json，记录项目的一切信息。

request模块，加载第三方接口的数据

md5

nodemailer

ws

Node内置模块-Path

一.PATH模块

node自带的模块,操作路径

1.路径的划分

绝对路径：从根目录开始，也是说是一段完整的域名。

相对路径：./当前目录，../上一级目录，/根目录

物理路径：从根目录开始，找到对应文件的路径

2.前端操作路径 - 重点

- 基本都是绝对路径操作
- 如果是相对路径，必须采用了拼接的方式(域名+相对路径)。

```
// 3.案例:
// http://localhost:80/vmall/src/index.html
// ./img/vmall_logo.png
// 结果: http://localhost:80/vmall/src/img/vmall_logo.png
// http://localhost:80/vmall/src/index.html
// ../img/vmall_logo.png
// 结果: http://localhost:80/vmall/img/vmall_logo.png
// http://localhost:80/vmall/src/index.html
// /img/vmall_logo.png
// 结果: http://localhost:80/img/vmall_logo.png
// http://localhost:80/vmall/src/index.html
// ../.././img/vmall_logo.png
// 结果: http://localhost:80/img/vmall_logo.png
```

二.path模块

const path = require("path");

1.path.join() 方法使用平台特定的分隔符把全部给定的 path 片段连接到一起，并规范化生成的路径 - 重点

```
console.log(path.join(__dirname)); // 获取当前文件的物理路径
D:\phpstudy_pro\WWW\JS2303\week07\DAY31\code
console.log(path.join(__dirname, "hehe", "xixi", "haha", "index.html"));
绝对路径
D:\phpstudy_pro\WWW\JS2303\week07\DAY31\code\hehe\xixi\haha\index.html
console.log(path.join("hehe", "xixi", "haha", "index.html"));
拼接相对路径
hehe\xixi\haha\index.html
```

2.path.resolve():将参数拼接成一个完整的路径

```
console.log(path.resolve("haha", "index.html"));
D:\phpstudy_pro\WWW\JS2303\week07\DAY31\code\haha\index.html
```

3.path.parse():解析路径 - 重点

```
const url = "http://127.0.0.1:8888/goods/list/index.html";
console.log(path.parse(url));

// {
//   root: '',
//   dir: 'http://127.0.0.1:8888/goods/list',
//   base: 'index.html', // 文件
//   ext: '.html', // 文件的扩展名
//   name: 'index' // 文件名
// }

let { ext } = path.parse(url);
console.log(ext);
```

Node内置模块-fs

fs:fileSystem 文件系统，操作文件以及文件夹

1.引入内置模块

```
const fs = require("fs");
const path = require("path");
```

2.writeFile:异步的向某一个文件中写入内容,如果文件不存在，创建再写入内容。

```
const obj = {
  a: 1,
  b: 2,
  c: 3,
};
```

创建目录(异步创建,添加回调进行是否成功创建的约定)

```
fs.mkdir("fs", (err) => {
  if (err) console.log(err);
  console.log("创建成功");
});
```

目录里面异步创建index.json

```
fs.writeFile(
  path.join(__dirname, "fs/index.json"),
  JSON.stringify(obj),
  (err) => {
    if (err) console.log(err);
    console.log("创建文件成功");
  }
);
console.log(1);
```

同步创建

```
fs.writeFileSync("fs/index.html", "<body>我是一段文字</body>");
```

异步追加内容

```
fs.writeFile(
  "fs/index.html",
  "<h2>这是二标题</h2>",
  {
    flag: "a", //w:修改   a:追加
  },
  (err) => {
    if (err) console.log(err);
    console.log("修改文件成功");
  }
);
```

3.readFile:异步获取文件里面的内容

```
fs.readFile("./fs/index.json", (err, docs) => {
  if (err) console.log(err);
  console.log(docs.toString()); //{"a":1,"b":2,"c":3}
});
fs.readFile("./fs/index.json", "utf-8", (err, docs) => {
  if (err) console.log(err);
  console.log(docs); //{"a":1,"b":2,"c":3}
});
```

4.readFileSync:同步获取文件里面的内容

```
console.log(fs.readFileSync("./fs/index.html", "utf-8"));
```

Node内置模块-url

url:解析地址

1.引入内置模块

```
const url = require("url");
```

2.url.parse()

```
console.log(url.parse(pathUrl));

// {
//   protocol: 'http:',  //协议
//   slashes: true,  //包含双斜杠
//   auth: null, //作者
```

```
//   host: 'localhost:8888',//主机
//   port: '8888',//端口
//   hostname: 'localhost',//主机名
//   hash: '#about',//哈希值
//   search: '?a=1&b=2&c=3&d=4',//携带的参数的完整格式
//   query: 'a=1&b=2&c=3&d=4',//携带的参数
//   pathname: '/goods/list/index.html',//路径名
//   path: '/goods/list/index.html?a=1&b=2&c=3&d=4',//路径
//   href: 'http://localhost:8888/goods/list/index.html?a=1&b=2&c=3&d=4#about' //完整的url
// }

console.log(url.parse(pathUrl, true)); //深度解析参数

// {
//   protocol: 'http:',
//   slashes: true,
//   auth: null,
//   host: 'localhost:8888',
//   port: '8888',
//   hostname: 'localhost',
//   hash: '#about',
//   search: '?a=1&b=2&c=3&d=4',
//   query: [Object: null prototype] { a: '1', b: '2', c: '3', d: '4' },
//   pathname: '/goods/list/index.html',
//   path: '/goods/list/index.html?a=1&b=2&c=3&d=4',
//   href: 'http://localhost:8888/goods/list/index.html?a=1&b=2&c=3&d=4#about'
// }

const { query, pathname } = url.parse(pathUrl, true);
console.log(query); //{ a: '1', b: '2', c: '3', d: '4' }
console.log(pathname); ///goods/list/index.html
```

3.url.format(url):url.parse逆运算

```
const obj = {
  protocol: "http:",
  slashes: true,
  auth: null,
  host: "localhost:8888",
  port: "8888",
  hostname: "localhost",
  hash: "#about",
  search: "?a=1&b=2&c=3&d=4",
  query: "a=1&b=2&c=3&d=4",
  pathname: "/goods/list/index.html",
  path: "/goods/list/index.html?a=1&b=2&c=3&d=4",
  href: "http://localhost:8888/goods/list/index.html?a=1&b=2&c=3&d=4#about",
};
console.log(url.format(obj)); //http://localhost:8888/goods/list/index.html?a=1&b=2&c=3&d=4#about
```

Node内置模块-querystring

querystring - 查询字符串

```
const qs = require("querystring");
```

1.querystring.parse()解析字符串，将字符串按照特定的参数进行分解。

```
const str = "name=zhangsan&age=18&sex=男";
console.log(qs.parse(str)); //{ name: 'zhangsan', age: '18', sex: '男' }
```

自由设定拆分的参数,默认是&和=

```
const str1 = "name:zhangsan%age:18%sex:男";
console.log(qs.parse(str1, "%", ":")); //{ name: 'zhangsan', age: '18', sex: '男' }
```

2.querystring.stringify() 将对象拼接成字符串

```
const obj = { name: "zhangsan", age: "18", sex: "男" };
console.log(qs.stringify(obj)); //name=zhangsan&age=18&sex=%E7%94%B7
console.log(qs.stringify(obj, "%", ":")); //name:zhangsan%age:18%sex:%E7%94%B7
```

Node内置模块-http

http模块

- 因为 node 是一个服务端语言
- 所以 node 一定也可以开启一个服务器，开启一个服务
- http 这个模块就是专门用来开启服务，并且接受请求，返回响应的
- http 也是一个内置模块，直接导入使用就行

注意：只要服务器里面的代码发生改变，必须重启服务器(ctrl+c关闭，重新开启)

1：引入内置模块http

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

2：利用createServer创建服务器

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

2.1.req:请求的意义，前端给后端 request

req.url:获取前端的接口地址

req.method:获取前端的请求方式(GET/POST)

2.2.res:响应的意义，后端给前端 response

res.writeHead()配置响应头

res.setHeader()设置响应头

res.end():结束响应，返回括号里面的数据。

```
res.setHeader("Access-Control-Allow-Origin", "*"); //解决跨域访问
// res.end(
```

```
    //   "哇塞，你已经成功使用 ajax 发送给我了一个请求，这是我给你的回应，我是一个字符串类型
  ^_^ !"
    // );
    res.end(
      JSON.stringify({
        message: "我已经接受到了你的请求，这是我给你的回应，我是一个 json 格式",
        tips: "后端返回给前端的数据",
        code: 1,
        age: 18,
      })
    );
  });
```

3: 监听的端口

```
  server.listen(3000, () => {
    console.log("服务器启动成功，你目前在监听3000端口");
  });
```