1.nodejs是什么

后端语言: 它是通过js代码编写，调用的api封装是好了的底层C++代码，并不是通过js代码转换成了C++；

> nodejs是对是对Google V8引擎进行了封装。V8引 擎执行Javascript的速度非常快，性能非常好。Node.js对一些特殊用例进行了优化，提供了替代的API，使得V8在非浏览器环境下运行得更好。

异步编程：执行一个命令时不会影响后面的程序执行，大量的异步编程使效率大大提升。

2. 终端常用的命令

- cd 进入目录 （按tab可以自动补全目录）
- 换盘直接 盘符: （比如 D:）
- cls 清空当前终端的内容
- （powsershell） clear 快捷键ctrl+L 清空当前终端的内容
- dir 查看当前目录中的内容
- （powsershell） ls 查看当前目录中的内容
- 方向键上下查看历史命令
- history：查看所有历史敲过的命令

3.模块系统

```
console.log("hello world!");
//node 01 ：  通过node运行01.js文件
```

1. 包里面有很多模块
2. 引入：require()

3. 导出：module.exports = 数据
4. 没有加./ 找的是原生的模块
5. 可以省略js后缀
6. 一个模块里面导入可以有多个，导出只能一个
7. 每个js文件都是一个模块

```
//声明全局的变量(尽量不采用这种写法)
global.name = name;
```

示例：

05导出1.js：

```
let goudan = (x,y) => x + y;

// module.exports = 10;
module.exports = goudan;
```

06导出2.js：

```
const value =  require("./05导出1");
// require()的返回值就是路径中模块导出的内容

//console.log(value); // 10

console.log(value(4, 5));

// 一个模块里面导入可以有多个，导出只能一个
//每个js文件都是一个模块
```

进阶：

1. 导入的模块没有导出数据的时候，数据默认是一个空对象
2. 导出的本质是返回module.exports 这个数据
3. module.exports 默认是一个空对象
4. 默认情况下，存在 exports = module.exports，exports 和 module.exports 是引用关系

引用关系分析：

```
let b ={};  // 声明一个引用类型的对象数据
let a = b;  // 将b的地址给了a ，  a 和 b 指向同一个引用

a.name = '11'; // a 的变化会影响到 b
a.age = 18;    // a 的变化会影响到 b

a = {name:'22',age:20};  // 给a 了一个新地址，让a指向了一个新的引用，使其与b断开引用关系
console.log(b); // { name: '11', age: 18 }  // b 还是原来的，a再不会影响到b
```

案例:

第一个js文件:

```
let fn = function () {
    return '导出~~~';
};

// module.exports = fn;
/*
    默认情况下，存在  exports = module.exports
    exports 和 module.exports  是引用关系
 */
exports.name = "hjh";

exports.age = 18;

exports.address = "北京";

exports = {
  id : 1,
  name : "afei",
  age : 30
};
```

第二个js文件:

```
const val = require("./exports");
// console.log(val());
//console.log(val); // 导入的模块没有导出数据的时候，数据默认是一个空对象
/*
    导出的本质是返回module.exports 这个数据
    module.exports 默认是一个空对象
 */
//console.log(module.exports);
console.log(val); //{ name: 'hjh' }
console.log(val); //{ name: 'hjh', age: 18, address: '北京' }
console.log(val); //{ name: 'hjh', age: 18, address: '北京' }'
```

02与03.js

02.js:

```
module.exports = {
    name : 'hjh',
    age : 22
};
console.log(module.exports); // { name: 'hjh', age: 22 }
```

03.js:

```
const  d = require('./02');
d.sex = "男"; // 不会改变被导入模块中的module.exports中的值
console.log(d);
```

path模块

```
1. 原生模块/npm下载的模块 引入
/**
 *
 *       .join()    传入路径，返回拼接好的路径
 *       __dirname : 返回绝对路径
 *       .resolve() : 前面补上绝对路径
 *       .relative( 参数一，参数二 ) : 从参数一到参数二如何去(返回的是相对路径)
 *        .parse() : 解析路径的
 */
```

```
/*
   {
     root: 'c:/',  // 根路径
     dir: 'c:/user/note', // 目录
     base: 'app.js', // 当前的文件
     ext: '.js',      // 后缀名（扩展名）
     name: 'app'      // 文件名
    }
*/
```

路径模块1：

```
/*
    原生模块/npm下载的模块 引入
 */
const  path = require('path'); // 自动的去我们的安装目录里面找
console.log(path);

/**
 *
 *       .join()    传入路径，返回拼接好的路径
 *       __dirname : 返回绝对路径
 *       .resolve() : 前面补上绝对路径
 *       .relative( 参数一，参数二 ) : 从参数一到参数二如何去(返回的是相对路径)
 *
 */
console.log(path.join('./a','/b')); // a\b
console.log(path.join('C:/a','/b')); // C:\a\b
console.log(path.join('C:/a','b','c')); // C:\a\b\c

console.log(__dirname); // E:\node\node3

// windows 和 linus 下面都可以使用(通过path拼接的路径)
console.log( path.join( __dirname,"./text/1.txt" ) ); //
E:\node\node3\text\1.txt

console.log(path.resolve('a', 'b')); // E:\node\node3\a\b

console.log(path.relative('a/b', 'a/b/c')); // c
console.log(path.relative('a/b/c', 'a/')); // ..\..
```

路径模块2：

```
const path = require('path');
/*
    .parse() : 解析路径的
 */
let a = path.parse("https://www.tmall.com/?
ali_trackid=2:mm_26632258_3504122_48284354:1580968197_124_1482015
932&clk1=f779f476078e140f1c7333484fb0f250&upsid=f779f476078e140f1
c7333484fb0f250");
//console.log(a);

let b = path.parse("c:/user/note/app.js");
console.log(b);
/*
    {
      root: 'c:/',  // 根路径
      dir: 'c:/user/note', // 目录
      base: 'app.js', // 当前的文件
      ext: '.js',     // 后缀名（扩展名）
      name: 'app'     // 文件名
     }
 */
```

*** 注意：在path模块中常用的有：.join()、__dirname、.parse()**

url模块：

```
//const url = require('url');
//console.log(url);
/*
    { Url: [Function: Url],
      parse: [Function: urlParse],
      resolve: [Function: urlResolve],
      resolveObject: [Function: urlResolveObject],
      format: [Function: urlFormat],
      URL: [Function: URL],
      URLSearchParams: [Function: URLSearchParams],
      domainToASCII: [Function: domainToASCII],
      domainToUnicode: [Function: domainToUnicode],
      pathToFileURL: [Function: pathToFileURL],
      fileURLToPath: [Function: fileURLToPath] }

      以上重点学习的是URL，其他大部分都是遗留的，不推荐使用

 */
```

```
/*console.log(url.URL); // [Function: URL]
let u = new url.URL("https://www.tmall.com/?
ali_trackid=2:mm_26632258_3504122_48284354:1580968197_124_1482015
932&clk1=f779f476078e140f1c7333484fb0f250&upsid=f779f476078e140f1
c7333484fb0f250");
console.log(u);*/

// 改进
const {URL} = require('url');
let u = new URL("https://www.tmall.com/?
ali_trackid=2:mm_26632258_3504122_48284354:1580968197_124_1482015
932&clk1=f779f476078e140f1c7333484fb0f250&upsid=f779f476078e140f1
c7333484fb0f250");
console.log(u);


/*
    URL {
  href:
   'https://www.tmall.com/?
ali_trackid=2:mm_26632258_3504122_48284354:1580968197_124_1482015
932&clk1=f779f476078e140f1c7333484fb0f250&upsid=f779
f476078e140f1c7333484fb0f250',
  origin: 'https://www.tmall.com',  // 源
  protocol: 'https:', // 协议
  username: '', // 用户名
  password: '', // 密码
  host: 'www.tmall.com', // 主域名
  hostname: 'www.tmall.com',  // 主域名
  port: '', // 端口
  pathname: '/', //路由
  search:
   '?
ali_trackid=2:mm_26632258_3504122_48284354:1580968197_124_1482015
932&clk1=f779f476078e140f1c7333484fb0f250&upsid=f779f476078e140f1
c7333484f
b0f250', // get方式传的数据
  searchParams:
   URLSearchParams {
  'ali_trackid' =>
'2:mm_26632258_3504122_48284354:1580968197_124_1482015932',
  'clk1' => 'f779f476078e140f1c7333484fb0f250',
  'upsid' => 'f779f476078e140f1c7333484fb0f250' },
  hash: '' } // 整理后的search ，  通过map格式
```

```
 */


let x = {a:1,b:2,c:3};
// let y = x.a;
/*let {b} = x;
console.log(b); // 2
console.log(a); // a is not defined*/

let {b:hjh} = x;
console.log(hjh); // 2
```

*** 注意：主要学习url模块中的URL,URL是构造函数，获取时采用解构赋值。**

querystring模块

querystring也是处理路径的

主要有两个方法：

1. parse() : 将查询字符串转换成对象
2. stringify ： 将对象拼接成需要查询的字符串

```
const querystring = require('querystring');
//console.log(querystring);

let a =
querystring.parse('ali_trackid=2:mm_26632258_3504122_48284354:158
0968197_124_1482015932&clk1=f779f476078e140f1c7333484fb0f250&upsi
d=f779f476078e140f1c7333484f\n' +
    'b0f250');

console.log(a);
/*
    [Object: null prototype] {
          ali_trackid:
'2:mm_26632258_3504122_48284354:1580968197_124_1482015932',
          clk1: 'f779f476078e140f1c7333484fb0f250',
          upsid: 'f779f476078e140f1c7333484f\nb0f250' }
 */
```

```
const querystring = require('querystring');
//console.log(querystring);
/*
    .stringify（） ：  将对象拼接成需要查询的字符串
 */

let a = querystring.stringify({name:'hjh',age:22,hobby: '乒乓球'},";",":");
console.log(a); // name=hjh&age=22
```

ES6补充:

Set:

```
/*
    1、Set ： 自动去重
 */

let s = new Set([11,22,33,44,33,22,11]);
console.log(s); // Set { 1, 2, 3, 4 }

//取值
for (let k of s){
    console.log(k);
    /*
        11
        22
        33
        44
     */
}

// 将Set数据变成数组
console.log( [...s] );
// 将Set数据变成数组[ 11, 22, 33, 44 ]
// 去重后的数组


// 添加数据
s.add("hjh");
console.log(s);

// 删除数据
s.delete(11);
console.log(s);
```

```
// 判断有无
console.log(s.has(22)); // true
```

Map:

```
let a = {
    // name : 'hjh'
    "name" : "hjh2",
    "a b c d" : "sherry"
};
//console.log(a); // { name: 'hjh2', 'a b c d': 'sherry' }

/*
    本质：
        let a = {
            "name" : "hjh"
        };
    注意：  键值不能是对象
 */

/*
    1、Map : 存储的数据可以是 key  value 的对应
              只不过key不仅仅是字符串了，可以是任意的数据类型
 */
let map = new Map();
map.set("name","hjh");
map.set("age",22);

console.log(map); // Map { 'name' => 'hjh', 'age' => 22 }

let k = {};
map.set(k,10);
console.log(map); // Map { 'name' => 'hjh', 'age' => 22, {} => 10
}

//取值
console.log(map.get("name")); // hjh
console.log(map.get(k)); // 10

/*map.forEach(v=>{
    console.log(v);
    /!*
        hjh
```

```
        22
        10
     *!/
});*/
console.log(map.entries()); // [Map Iterator] { [ 'name', 'hjh'
], [ 'age', 22 ], [ {}, 10 ] }

// 第二种遍历
for(let [key,value] of map.entries()){
    console.log(key,value);
    /*
        name hjh
        age 22
        {} 10
     */
}
```

***注意：Set数据主要用于去重，Map可以设置任意类型的键值**

fs模块

nodejs所有异步操作api都必须有回调函数，并且回调函数的第一个形参是错误对象(没有错误是null，有错误是对象)
fs.readFile(文件路径，文件格式，回调函数);

```
/*
    fs
        file system
 */
const fs = require('fs');
//console.log(fs);

// 读文件
/*
    位置
 */
//fs.readFile("./text/1.txt"); // 报错
/*
    fs.js:128
  throw new ERR_INVALID_CALLBACK();
  ^

TypeError [ERR_INVALID_CALLBACK]: Callback must be a function
// 缺少回调函数
```

```
    at maybeCallback (fs.js:128:9)
    at Object.readFile (fs.js:277:14)
    at Object.<anonymous> (E:\node\node3\09fs文件系统.js:12:4)
    at Module._compile (internal/modules/cjs/loader.js:701:30)
    at Object.Module._extensions..js
(internal/modules/cjs/loader.js:712:10)
    at Module.load (internal/modules/cjs/loader.js:600:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:539:12)
    at Function.Module._load
(internal/modules/cjs/loader.js:531:3)
    at Function.Module.runMain
(internal/modules/cjs/loader.js:754:12)
    at startup (internal/bootstrap/node.js:283:19)
 */
// 读取文件是异步操作，必须有一个回调函数
// 正确写法如下
fs.readFile("./text/1.txt",(err) => {
    // 读取完成之后执行这个函数
    // 所有的异步回调，第一个参数都是 err
    //console.log(err);  // 成功的时候打印 null

    if(err){
        console.log("异步操作失败");
    }else{
        console.log("异步操作成功！");
    }
});
console.log(1);
// 1先打印

/*
    nodejs所有异步操作api都必须有回调函数，并且回调函数的第一个形参是错误对象
(没有错误是null，有错误是对象)
 */
```

```
const fs = require('fs');

fs.readFile("./text/1.txt","utf8",(err,body) => {
    if(err) return;
    // 读取成功之后，才会执行下面的
    //console.log(body); // <Buffer 68 65 6c 6c 6f 20 77 6f 72 6c
64 21 0d 0a>
    // 读取的内容是buffer数据 ，  所有读取的文本，视频等都是流文件

    //console.log(body.toString()); // hello world!
```

```
    //  body.toString(),转换成utf8格式，一般不采取这种方式
    // 通常直接在回调函数之前指定文件格式

    console.log(body); // hello world!

});

//fs.readFile(文件路径，文件格式，回调函数);
```

***注意：读取文件通常传入第二参数(utf8)**