**1.模块**

定义：一个独立的功能体。 用文件封装

在NodeJS下，模块分为： 自定义模块，核心模块，第三方模块

自定义模块

在NodeJS中 每个文件都是一个模块。

NodeJS自动为每个文件添加了构造函数，所有代码被构造函数所包含

|  |
| --- |
| (function (exports, require, module, \_\_filename, \_\_dirname){  程序员写代码处  }) |
| require() 用于引入模块，返回一个对象。同一级目录在名字前写路径 ./ |
| module 模块，指代当前的模块对象 |
| module.exports 当前模块的导出对象，公开的属性和方法 |
| exports 指代module.exports exports===module.exports对象指代的地址一模一样 |
| \_\_filename 当前模块的绝对目录和模块名称（当前文件所在目录及名字） |
| \_\_dirname 当前模块的绝对目录 （当前文件所在地目录） |

exports === module.exports 对象指代的地址一模一样

例 引入模块

main文件

var r = 4; //↓若为.js文件则js可省略

var yuan = require('./circle.js'); //↓取两位小数

console.log(yuan.mj(r).toFixed(2) , yuan.zc(r).toFixed(2));

console.log(\_\_filename);

circle文件

function mj(r){

var mianji = Math.pow(r,2) \* Math.PI;

return mianji;

};

module.exports.mj = mj;

module.exports.zc = function zc(r){

var zhouchang = 2\*r\*Math.PI;

return zhouchang;

}; //也可以这样写

/\*直接导出一个对象，相当于在堆内存创建了一个新的内存，exports和module.exports就不再指向同一个对象

module.exports={

ename:'xiaoming',

sex:2

};

console.log( module.exports ===exports ) ==> false

1. **模块引入**

|  |  |  |
| --- | --- | --- |
|  | 以路径开头（./） | 不以路径开头 |
| 文件模块 | require('./circle.js')  常用与引入自定义的模块，后缀名.js可省略 | require('url')  直接引入，常用于官方提供的核心模块 |
| 目录模块 | require('./mulu')  到mulu文件夹下找package.json文件中main属性对应的文件名称，如果找不到才会引入名为index.js的文件，其他的不会引用 | require('mulu2')  首先会在当前目录寻找node\_modules文件夹，再找其中的mulu2，如果找不到往上级查找直到顶级目录，找不到才会报错，常用与第三方模块 |

package.json 看做一个对象

{

"main":"2333.js" // 属性名要双引号括起来，字符串的方法也是

}

自我总结：如果在一个目录（以路径开头），直接'./文件名'导入，导入目录下模块，如在同级别，输入'./目录名'，系统会优先在目录下找文件package.json，其中main对应的文件，如果没有package则默认打开index.js文件，否则无法引用。如果不在同一个目录下在，在当前级别node\_modules目录下去找，最高同盘顶级目录，其余和前面一样。

1. 包和npm

CommonJS是一种规范，JS向服务器端发展就是遵循了这个规范。模块概念的规范属于其中。

包（package）就是一个目录模块 （第三方模块）

npm：用于管理 包 的工具

www.npmjs.com

**切换目录**

cd 完整路径（记得先换盘）

在指定的目录空白区域，按住shift单击右键，在此处打开power

shell窗口

**使用npm下载**

npm install 包名称 回车

会自动安装到所在目录下的node\_modules中。

npm init -y 回车

安装过的包相关信息会自动存储到与node\_modules同级的package.json 文件中，

npm install 回车

自动安装package.json中记录的包名称及版本号

1. querystring 模块（查询字符串）

查询字符串：浏览器向服务器发送请求，传递数据的一种方式

Search?keyword=手机&enc=utf-8.....

parse( ) 将查询字符串解析为对象 （算属性吗？）

stringify( ) 将对象格式化为查询字符串

//引入查询字符串模块

const querystring =require('querystring');

//查询字符串

var str = 'keyword=手机&enc=utf8';

//把查询字符串str解析为对象

var ob = querystring.parse(str);

console.log(ob); //{ keyword: '手机', enc: 'utf8' }

//把对象格式化为查询字符串

console.log(querystring.stringify(ob)); //keyword=%E6%89%8B%E6%9C%BA&enc=utf8

1. URL模块

http://www.tmooc.cn:80/detail.html?lid=5

parse( ) 将URL解析为对象 （和上面比对象不同）

protocal 协议 http:

hostname 主机名 域名/ip地址 www.tmooc.cn

port 端口号 80

pathname 请求的文件路径名称 /detail.html

query 查询字符串 lid=5

format( ) 将对象格式化为URL

注意：query项中对应的是一个对象

例：//引入url模块

const url = require('url');

//处理和解析url

var str = 'http://www.tmooc.cn:80/detail.html?lid=5';

//把url解析为对象

var obj = url.parse(str);

console.log(obj);

var obj2 = {

protocol:'http:',

hostname:'www.tmooc.cn',

port:8080,

pathname:'detail.html',

query:{

lid:5,

pname:'dell'}

};

//把对象格式化为URL

var str2 = url.format(obj2);

console.log(str2); //http://www.tmooc.cn:8080/detail.html?lid=5&pname=dell

1. 全局函数

可以借助global调用的函数

回调函数会放入到事件队列中，待主程序执行完才会执行

1. 一次性定时器

|  |
| --- |
| 开启  var timer = setTimeout( 回调函数 , 间隔时间 );  当间隔时间到，执行一次回调函数，时间单位是毫秒  清除（把开启存到一个函数中）  clearTimeout( timer ) |

var timer = global.setTimeout(()=>{console.log('boom!');},1000);

//clearTimeout(timer);

1. 周期性定时器

|  |
| --- |
| 开启  var timer = setInterval( 回调函数 , 间隔时间 );  没隔一个间隔时间执行一次回调函数  清除  clearInterval(timer) |

1. 立即执行定时器（事件队列最前执行）

|  |
| --- |
| 开启  var timer = setImmediate(回调函数);  清除  clearImmediate(timer) |

1. 立即执行

将回调函数放到主程序最后一个执行，执行顺序在事件队列之前，且没有清除功能

|  |
| --- |
| process.nextTick( 回调函数 ); |