1. 事件触发器(events)模块：用户自己定义[注册]并触发事件

1、注册事件：

myEvent.on('事件名',回调函数);

myEvent.once('事件名', 回调函数); once()注册的事件只会被触发一次

1. 触发事件: myEvent.emit('事件名')：

//events事件触发器模块：用户自己注册并触发事件

const eventsEmiter = require('events');

const events = new eventsEmiter();

//注册事件： events.on('事件名',回调函数);

//          events.once('事件名',回调函数);  //只能触发一次

//触发事件： events.emit('事件名');

events.on('cctv1', () => {

    console.log('cctv1...');

});

function mydemo() {

    setTimeout(() => {

        console.log('Hello web');

    }, 2000);

}

events.on('tv', mydemo);

events.once('tv13', () => {

    console.log('cctv13...');

});

//触发事件：

events.emit('cctv1');

events.emit('cctv1');

events.emit('cctv1');

events.emit('tv');

events.emit('tv13');

events.emit('tv13');

1. 完整的url地址格式：

┌────────────────────────────────────────────────────────────────────────────────────────────────┐

│ href │

├──────────┬──┬─────────────────────┬────────────────────────┬───────────────────────────┬───────┤

│ protocol │ │ auth │ host │ path │ hash │

│ │ │ ├─────────────────┬──────┼──────────┬────────────────┤ │

│ │ │ │ hostname │ port │ pathname │ search │ │

│ │ │ │ │ │ ├─┬──────────────┤ │

│ │ │ │ │ │ │ │ query │ │

" https: // user : pass @ sub.example.com : 8080 /p/a/t/h ? query=string #hash "

│ │ │ │ │ hostname │ port │ │ │ │

│ │ │ │ ├─────────────────┴──────┤ │ │ │

│ protocol │ │ username │ password │ host │ │ │ │

├──────────┴──┼──────────┴──────────┼────────────────────────┤ │ │ │

│ origin │ │ origin │ pathname │ search │ hash │

├─────────────┴─────────────────────┴────────────────────────┴──────────┴────────────────┴───────┤

│ href │

└────────────────────────────────────────────────────────────────────────────────────────────────┘

1. nodejs模块分为三种：内置模块、自定义模块、第三方模块(包)

内置模块: url、path

//内置模块：

const path = require('path');

const url = require('url');

let cururl = "http://www.baidu.com:3000/mydemo/test/index.php?uid=21&usr=lisi";

let obj = url.parse(cururl);

// console.log(obj, obj.pathname);

//获取文件扩展名

let ext = path.extname(obj.pathname);

console.log(ext);

// path.join(): 将路径片段拼装在一起， 需要注意的是使用当前平台中的分隔符来拼装， window使用的是: \(反斜杠), linux使用的是: /(斜杠)

let p = `${\_\_dirname}mydemo/123.html`;

let curpath = path.resolve(\_\_dirname, './mydemo/123.html');

console.log(curpath);

curpath = path.resolve(\_\_dirname, '../mydemo/123.html');

console.log(curpath);

curpath = path.resolve(\_\_dirname, './mydemo/yesok/', '../test/123');

console.log(curpath);

1. 自定义模块： 在nodejs中每个.js文件都是一个模块，而每个模块都是封闭的空间，因为每个模块文件被一个匿名函数所包裹，如下：

// function (exports, require, module, \_\_filename, \_\_dirname) { var age = 20;

// exports = module.exports;   //自己添加的

//     function fn() {

//         console.log('fn在4-module.js文件中...');

//     }

//     console.log(arguments.callee.toString());

//     }

//  module.exports = {};  //自己添加的

//  return module.exports;  //自己添加的

// }

1自定义模块的语法：

// 1)、单个暴露：

//  (1)使用 exports:

//      exports.属性 = 属性值;

//      exports.方法 = 函数;

//  (2)使用module.exports:

//      module.exports.属性 = 属性值;

//      module.exports.方法 = 函数;

// 2)、批量暴露：

//  module.exports = {}

//\*\*\*\*注意：单个暴露与批量暴露不能一起使用

1. 自定义模块中的exports与module.exports的区别？

因为exports是module.exports的引用，而module.exports是真实存在的，最终暴露的module.exports，如下：

// module = { exports: {} };

// exports = module.exports;

// console.log(exports, module.exports, 111);

// module.exports = { d: 11, dem: function() {} };

// console.log(exports, module.exports, 2222);

作业：在自定义模块中实现计算器功能(加、减、乘、除)

1. 第三方模块[包]： <https://www.npmjs.com>
2. 使用第三模块时要先安装: npm i/install 包名

**将npm服务器切到国内淘宝服务器(镜像服务器)，在dos小黑窗中分别执行下面的命令**：

npm config set registry https://registry.npm.taobao.org --global

npm config set disturl https://npm.taobao.org/dist --global

2、在安装第三方模块时要注意两点：

(1)先创建我们项目自己的package.json文件：

npm init 或 npm init -y (**使用-y参数时要注意项目名称不能含有中文**)

为什么要创建我们项目自己的package.json文件？

Package.json文件中配置项”dependencies”会记录我们依赖的所有第三方模块，当我们要把项目发给别人或要上传到服务器时，可以把我们项目安装的所有第三方模块删除掉，然后再发给别人或上传到服务器，当别人拿到项目或上传到服务后我们再使用npm install就可以把项目所有依赖的第三方模块再安装上。

(2)所有第三方模块最好安装在我们项目的根目录下.

为什么第三方模块最好安装在我们项目的根目录下？

将第三方模块安装在项目根目录下可以在项目的任何目录都使用该第三方模块，因为当我们使用require(‘包名’)引入第三方模块时会在当前目录中查找访模块，如果找不到则返回到上一级目录中查找，一直到某个盘符的根目录中还是找不到则报错。

3、常用的npm命令：

npm -v 查看npm的版本

npm version 查看所有模块的版本

npm init 初始化项目（创建package.json），创建package.json文件作用：记录我们当前node项目中依赖哪些第三方的包，当我们把node项目发给别人或发布时，就可以删除node项目中已安装的第三包文件。

npm i/install 包名 安装指定的包 @版本号，不加版本默认为最新

npm i/install 包名 --save 安装指定的包并添加依赖

--save-dev的功能： 开发依赖（开发阶段需要，上线不需要[因为已转换好了]）

--save的功能：之前旧的npm命令如果不带--save参数，则在.json文件中不产生依赖项(dependencies)

npm i/install 包名 -g 全局安装（一般都是一些工具）

npm i/install 安装当前项目所依赖的包

npm s/search 包名 搜索包

npm r/remove 包名 删除一个包

npm uninstall 包名 删除

注意：(1)、npm 老的版本(2，3版本在)安装包的时候，默认不加自动添加的denpendcies里面npm install 包名 --save 才会保存到依赖里面，但是现在npm升级好多次了，自动会回到package.json项目依赖当中。

1. 使用第三方模块(mime)：

1、在当前项目根目录下安装mime: npm i mime

2、使用mime模块，代码如下：

const mime = require('mime');

let curtypes = mime.getType('json');

console.log(curtypes);

curtypes = mime.getType('js');

console.log(curtypes);

let extname = mime.getExtension('image/png');

console.log(extname);

1. 什么是MIME?

MIME的全称是Multipurpose Internet Mail Extensions，即多用途互联网邮件扩展类型，在http协议用来标识要返回数据在类型，客户端会根据不同数据类型启动不同应用程序进行处理。

1. fs模块操作一般文件常用的方法有：

读取文件：fs.readFile()     fs.readFileSync()

写文件：  fs.writeFile()    fs.writeFileSync();

const fs = require('fs');

const path = require('path');

//对文件有哪些常用操作[了解]？

//打开文件： fs.open()      fs.openSync()

//读文件：  fs.readFile()   fs.readFileSync()

//写文件：  fs.writeFile()  fs.writeFileSync()

//关闭文件： fs.close()     fs.closeSync()

//打开文件：

//a:append

// fs.open(path.resolve(\_\_dirname, './demo.txt'), 'w+', (err, hd) => {

//     // fs.open(path.resolve(\_\_dirname, './demo.txt'), 'a+', (err, hd) => {

//     //注意：如果成功打开文件则hd代表当前打开的文件

//     //  console.log(err, hd);

//     //写文件：

//     fs.writeFile(hd, '12345', (err) => {

//         //console.log(err);

//         //关闭文件：

//         fs.close(hd, (err) => {

//             console.log(err);

//         });

//     });

// });

//fs模块中操作一般文件常用的方法有：

//读取文件：fs.readFile()     fs.readFileSync()

//写文件：  fs.writeFile()    fs.writeFileSync();

//let rtn = fs.writeFileSync(path.resolve(\_\_dirname, 'demo.txt'), 'Hi web');

let rtn = fs.writeFileSync(path.resolve(\_\_dirname, 'demo.txt'), 'Hello...', { flag: "a+" });

console.log(rtn);

1. 操作大文件时要使用文件流方式：

fs.createReadStream():创建可读文件流

fs.createWriteStream():创建可写文件流

readerStream.pipe(writerStream)：管道流读写操作

const fs = require('fs');

const path = require('path');

//fs操作大文件时要使用文件流方式：

//创建可读文件流：

let rs = fs.createReadStream(path.resolve(\_\_dirname, 'aipin.mp3'));

//创建可写文件流：

let wr = fs.createWriteStream(path.resolve(\_\_dirname, '123.mp3'));

//创建可写文件流：

let wr2 = fs.createWriteStream(path.resolve(\_\_dirname, '456.mp3'));

//管道流方法：

rs.pipe(wr2);

rs.on('open', () => {

    console.log('open...');

});

rs.on('ready', () => {

    console.log('ready...');

});

//读取(分批读取文件内容)

let i = 0;

// rs.on('data', (d) => {

//     // console.log(d, 111);

//     // i++

//     // console.log(i);

//     wr.write(d);

// })

1. fs模块操作目录常用的方法有：

创建目录：fs.mkdir()、fs.mkdirSync()

删除文件：fs.unlink fs.unlinkSync()

删除目录：fs.rmdir()、fs.rmdirSync()

目录或文件重命名：fs.rename()、fs.renameSync()

读取目录：fs.readdir()、fs.readdirSync()

获取文件状态: fs.stat() fs.statSync()

判断目录或文件是否存在 fs.existsSync()

const fs = require('fs');

const path = require('path');

//创建目录 : fs.mkdir()   fs.mkdirSync()

//当目录不存在时再创建

if (!fs.existsSync(path.resolve(\_\_dirname, 'test'))) {

    let rt = fs.mkdirSync(path.resolve(\_\_dirname, 'test'));

    console.log(rt);

}

//删除目录:fs.rmdir()   fs.rmdirSync()

//注意：rmdir()不能删除非空目录

//fs.rmdirSync(path.resolve(\_\_dirname, 'mydemo'));

//删除文件：fs.unlink()  fs.unlinkSync()

// fs.unlinkSync(path.resolve(\_\_dirname, '123.mp3'));

//更改文件或目录的名称：fs.rename()  fs.renameSync()

fs.renameSync(path.resolve(\_\_dirname, '456.mp3'), path.resolve(\_\_dirname, '789.mp3'))

//读取目录:fs.readdir()   fs.readdirSync()

let arr = fs.readdirSync(\_\_dirname);

//console.log(arr);

arr.forEach((item) => {

    //获取文件状态：fs.stat()   fs.statSync()

    let fstat = fs.statSync(path.resolve(\_\_dirname, item));

    //判断当前文件是否为目录：fstat.isDirectory()

    //判断当前文件是否为文件：fstat.isFile()

    if (fstat.isDirectory()) { //为目录

        console.log(`目录：${item}`);

    } else { //为文件

        console.log(`文件：${item}`);

    }

})

**作业**：读取某个目录及所有子目录中的文件