2017年7月24日 16:19

require(X)
如果x看起来像一个路径（以 ./、../、 /开头），直接加载路径对应的文件，如当前目录下 require("./a.js")
如果x看起来不像一个路径
    如果x是一个内置模块的名字，直接返回内置模块
    如果x不是一个内置模块的名字
        在当前文件夹的node_modules文件夹里找名为x的文件夹y(即路径为node_modules/x/)
        如果y文件夹里有package.json,则加载main字段指向文件
        如果不存在package.json，则直接加载Y文件夹里的index文件
    如果在当前文件夹的node_modules里找不到名为x的文件夹
        则往当前文件夹的父文件夹里找node_modules

’..’表示父文件夹
'.' 表示当前文件夹
'/' 表示从根目录取

npm i -g aaa
-g 即--global
全局安装，安装在全局的目录，在全局都可以直接用该命令
一般是会在安装完成后为系统增加一个或多个命令行工具
没有-g则是把模块安装到当前文件夹下的node_modules文件夹里

交互式命令行下的变量是放在全局里的，模块里的变量是放在一个函数里的

交互式命令下，内部模块都为全局变量，不需要require就可以直接使用，如fs

2017年7月25日　　11:39

Buffer 同ES6中的 `TypedArray`

字节序　解码一段内存时，是高位地址在前还是低位地址在前
TypeArray默认LE
Buffer默认BE

a=Buffer.alloc(10,"你我他") 给buffer分配10个字节
Buffer.from("5oiR","base64")　==>我
Buffer.from("你我","utif8") ==>出来你我的utif8编码的十六进制
a.writeDoubleBE(NaN,0) 从0位置开始写入浮点数

BE/LE代表字节序 bigending，BE从小到大，LE从大到小
buf.writeUInt16BE(value, offset[, noAssert]) 大的字节放前面，默认
buf.writeUInt16LE(value, offset[, noAssert]) 小的字节放前面

buf.swap32　四个字节换位，反序

IEE754

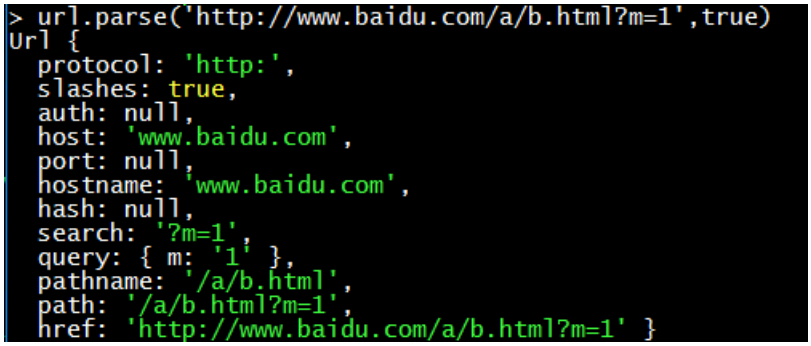
double精度64位，8个字节

url
url.parse(url,[true]) 解析url各个部分

```
> url.parse('http://www.baidu.com/a/b.html?m=1')
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'www.baidu.com',
  port: null,
  hostname: 'www.baidu.com',
  hash: null,
  search: '?m=1',
  query: 'm=1',
  pathname: '/a/b.html',
  path: '/a/b.html?m=1',
  href: 'http://www.baidu.com/a/b.html?m=1' }
>
```

传入第二个参数true，多出query对象

```
> url.parse('http://www.baidu.com/a/b.html?m=1',true)
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'www.baidu.com',
  port: null,
  hostname: 'www.baidu.com',
  hash: null,
  search: '?m=1',
  query: { m: '1' },
```



url.resolve('http://example.com/', '/one');
// 'http://example.com/one'
url.resolve("https://a/b/c/d.html","/b/e.html")
//'https://a/b/e.html'

path
path.join('/foo', 'bar', 'baz/asdf', 'quux', '..'); 路径字符串拼接，例子中..为上一文件夹，所以抵消
// Returns: '/foo/bar/baz/asdf'

path.resolve([...path])
path.format(pathObject)

```
path.format({
  root: '/',
  base: 'file.txt',
  ext: 'ignored'
});
// Returns: '/file.txt'
```

path.basename(path[,ext]) 获取文件的名称
path.basename('/foo/bar/baz/asdf/quux.html');
// Returns: 'quux.html'
path.basename('/foo/bar/baz/asdf/quux.html', '.html');
// Returns: 'quux'

path.extname(path) 获取文件扩展名

path.isAbsolute('C:\\foo\\..'); // true 是否为绝对路径
path.isAbsolute('bar\\baz'); // false

path.normalize('/foo/./bar/za/../c.htm') 化简路径
'\\foo\\bar\\c.htm'

path.parse("/a/b/c.html") 路径转换为对象
{ root: '/', dir: '/a/b', base: 'c.html', ext: '.html', name: 'c' }

time

timeout.unref()

process

process.cwd() 当前工作目录 current working dir

process.chdir() 改变当前目录

process.title 命令窗口的标题

process.nextTick(callback[,...args]) 事件循环结束后立刻执行

util

util.promisify(fn) 把异步函数转换为返回promise的函数

2017年7月26日 15:18

request 可读流，所有可读流都有data、end事件
response 可写流，所有可写流都有write、end方法

所有流都是EventEmitter的实例，即都有on data,on end事件

process.stdin 为可读流
process.stdout 为可写流
所以可以：
process.stdin.pipe(process.stdout)

pipe 可读流传到可写流
readStream.pipe(writeStream)
同on("data")，on("end"),但是没发出之前就不再读数据，所以比on data快
每收到64kb数据就触发on("data",fn)

双工流 Duplex 可写可读
如TCP net.Socket

转换流 Transform 如压缩流，zlib.createGzip()

2017年8月4日 11:40

ORM Object Relation Model
用编程语言里的类与类的实例，来表达数据库的表与表中的每一行

LeanCloud 前端数据库,不用搭服务器

CRUD create r update delete 增删改查
对应http四种method
对应数据库 insert delete update search

2017年8月15日 9:58

npm link //bin内的设置项，创建cmd文件到path路径下，
可以在全局执行
如 uglifyjs 为
"bin": {
"uglifyjs": "bin/uglifyjs" //该命令执行时启动的文件
},

符号链接

npm run test //scripts内的配置项
"scripts": {
"test": "node test/run-tests.js",
"server":"anywhere", //先在当前目录的node_modules下的bin找该命令,
"start":"echo start" ,
},

npm start
npm test
只有这两个命令可以不用run

//dependencies 运行阶段需要的包
npm i --save anywhere 保存到dependencies配置项，现在已经默认，不需要再写save
npm i 根据dependencies内的项全部下载安装

"dependencies": {
"optimist": "~0.3.5",
"source-map": "~0.1.7"
}
version
version Must match version exactly
>version Must be greater than version
>=version etc
<version
<=version
~version "Approximately equivalent to version" See semver，近似

^version "Compatible with version" See semver ,可以兼容

来自 <https://docs.npmjs.com/files/package.json>

//devDependence 开发或维护阶段的依赖包
npm i -D webpack　保存到devDependence项

将包发到npm
npm login
npm publish

监视文件变化重启
nodemon a.js

2017年8月15日 16:25

模块化：为了不用一堆js标签且还要根据依赖关系关心顺序

CommonJS 适用于后端 可加载动态模块 node中模块化加载
不涉及同步异步，类同步，即模块名字可以在代码运行的时候计算出来，模块直接从硬盘获取，一require就获取文件即执行
（书上的require，自己实现的require）
AMD Async Module Definition 适用于前端，异步获取，防止阻塞页面加载 即 requirejs 可加载动态模块
async异步执行，模块用到时才加载，加载到之后立即执行
写法上跟CommonJS不同
CMD Common Module Definition 即 seajs 不可加载动态模块
懒加载 lazy evalution
从入口模块开始所有依赖都会在入口模块运行之前加载好，但并未运行，直到require才执行
写法上跟CommonJS几乎相同

浏览器端的模块化加载资源，但还是流线型下载，跟原来放标签的效果一样，下载太慢
seajs CMD
模块中的代码放在define中为了防止文件因为跨域问题而获取不到，
所以seajs实现了一个define（把传给他的参数放到一个全局变量以便后续执行完后在script标签的onload事件里放入缓存以及继续加载他的依赖项）方法以及load（通过创建srcipt标签加载依赖文件）方法，通过最开始的入口文件加载完成之后，立即执行了define函数即得到传给他的参数（即模块内容），然后通过调用script标签的onload事件（现代浏览器script的onload事件会在脚本执行完后立即调用），通过将函数变成字符串得到函数内部依赖的文件，从而继续load这些文件
所以模块中的依赖文件的路径也只能写成静态的，不能有运算符之类的，否则无法解析依赖文件
requirejs

文件打包工具，打包成一个文件
browserify
只能打包js，以及不能做代码的拆分等
webpack
原理：通过node依次读取依赖文件内容，把代码都放到一个文件，然后生成一个require函数
默认调用入口文件

webpack -w 监视文件

有插件系统，可以加载 js/css/less/sass/jpg/jsX/vue，原理将所有文件都转换为js,图片转成base64或一个url来require,css用createElement("style")来实现

loader 处理require除js格式以外的其他类型的资源，将某种类型的文件转换为js文件以供webpack打包
plugin 处理webpack最终生成出来的文件，扩展webpack功能

vue-cli 生成一个webpack项目
vue

babel
一个js的编译器
jsx,ts,es6/7/8
一开始只是用于把es6写的代码转换为es5
其插件机制可以方便的开发各种其他功能
babel也是做webpack的loader

gulp /grunt task runner
其仅为一个任务运行器
提供了强大的流式任务处理和任务间的依赖关系管理
 es6->babel->uglifyjs->app.js

代码分离：
1、常用模块与不常用模块分开打包，如常用api与应用分开打包
2、按需加载模块，减少原始只打包成一个js的内容，增加加载速度

2017年8月17日 9:22

n

npm uuid 类似guid

表单提交类库npm
multer
formidable

Electron node+chrome的集成环境
fiddler

KCP 翻墙

```
ssh -R rport:localip:lport root@vpsip -p port -N
通过本机与vpsip机器的ssh连接，
让vpsip的rport端口相当于localip机器的lport端口
ssh 连接的接口通过 -p port 来指定
-N代表不执行shell

ssh -L localip:localport:remoteip:remoteport root@vpsip -p port -N
通过本机与vpsip机器的ssh连接，让localip的localport相当于remoteip的remoteport

ssh -D localport user@vpsip -p sshport
在本地localport开一个socks代理，通过ssh连接让vpsip机器来转发


//kcp
vps server
kcpserver -t targetip:targetport -l :kcpserverport

local pc
kcpclient -r vpsip:kcpserverport -l :localport

本机的 localport 就相当于 targetip:targetport
```