Node.js

Node.js是 Javascript 在服务器端运行环境==相对于后台

浏览器中的 js 组成部分：ECMAScript 核心 + DOM + BOM

Node中的 js 组成部分： ECMAScript 核心 + 全局成员 + 核心API成员,====>异步单线程非堵塞I/O--kfc

打开终端三种方式/window+r+cmd 所在文件夹输入cmd shift+右键

Node -v:表示查找应用程序为node的文件通过终端启动软件并且对其操作，先在当前目录找再去path环境变量找：高级设置查看path环境变量，可以删除和增加环境变量。从新查找要重新启动cmd查找

执行桌面上js文件： Node ./文件名 上箭头执行上一次命令。Tab快速补全路径。cls清空终端。 cd切换文件夹目录，cd ../ =退出在进入另一个 。Ctrl+~==》调出终端返回当前文件的所在目录

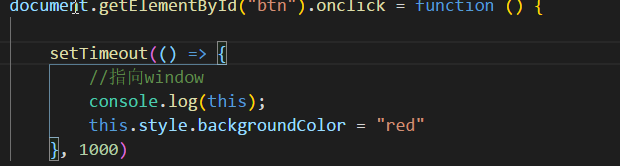
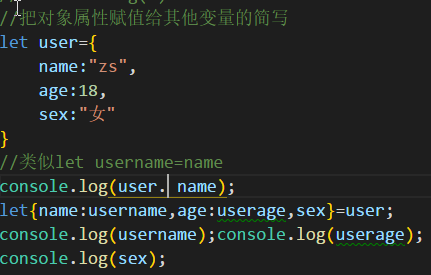
Node.js环境中有两种方式可以允许js代码：1node命令调出js文件：Node ./文件名 2REPL环境直接执行 ctrl+c 退出环境

Let语法解决var带来的变量提升问题，和{}提供了一个块级作用域（原生js函数产生块级作用域 里面的var变量外面访问不到），解决了变量污染问题 var 在for循环遍历只能执行最后一次结果==可以使用闭包匿名函数解决。每循环一次就开辟一次空间，当触发事件就可以正确执行。Let在for循环每一次就相当于函数开辟一次空间，把变量存在里面 for switch if while...等语句都都可。对象可以。不存在设置变量的问题

Const:定义常量，最大特性不能重新赋值，而且跟let一样只能先定义在使用，否则报错。其他和let也相似

声明对象时重新改变对象的属性：如果声明多个变量把他们写成对象。Var obj ={name,age.sayhi}

函数：（行参）{函数体,函数体里面的this指向当前整个函数外部的this对象}，（x,10）=》{return x+10}==>如果左边只有一个参数，右边只有一个语句，可以把两个括号省略掉。Return也可以省略。X=>10+x



A）:调用核心api成员方法：requre(“核心成员名称”):fs模块，相当于方法集合

//路径是相对路径，fs直接去拼接端口目录，node之前的目录

（E:\黑马培训\上课资料\NodeJS\作业demo> node 03demo.js）

（E:\黑马培训\上课资料\NodeJS(123情况下读取的文件直接拼接这个目录)> node 作业demo\03demo.js），$filrname起到拼接字符串作用

当我们文件刚好不再当前端口的所在目录中在去拼接就会出错，用一个方法解决。./表示当前文件下的某个文件==》\_\_dirname（能返回当前文件真正磁盘目录，不包含当前执行的文件名）+”拼接字符串，我们需要执行文件名”。\_\_filename(能返回包含当前执行文件名的完整磁盘路径，不用拼接)

1：读取文件

fs.readFile(“文件路径”，”编码格式，默认为null”,回调函数callback=function(error,data)用来处理读取结果.如果error=null代表读取成功，data是文件的内容，error.message是它一个属性)

2：写入文件

fs.writeFile（“文件路径.可以在某个路径新增文件”，“写入的字符串”，“编码格式默认utf8”，function（error）{}）===如果写入的文件路径不存在会新增，如果存在就覆盖以前的内容

3：追加文件

fs.appendFile(“存在的文件路径”，“追加的具体内容”，“编码格式默认utf8”,追加完成的回调function（error）{返回错误结果})，js控制换行\n====如果写入的文件路径不存在会新增，如果存在不会覆盖只会追加

4:读取文件信息：

fs.stat(\_\_dirname，(error,stats)=>{以便在返回函数中得到文件类型（size:单位字节）/文件创建时间（birthtime）/文件的磁盘目录(isDirectory())/是目录还是单一文件(isFile())

})

5:复制文件

fs.copyFile(“复制文件路径”，”目标文件路径” ，如果目标路径为存在传了非0的整数。会报错，function（error）{})==如果文件存在就覆盖以前内容

Arr.forEach((item,index)=>{}) 方法用于调用数组的每个元素，并将元素传递给回调函数直接执行。没有返回值。、

1. path模块const path = require('path')

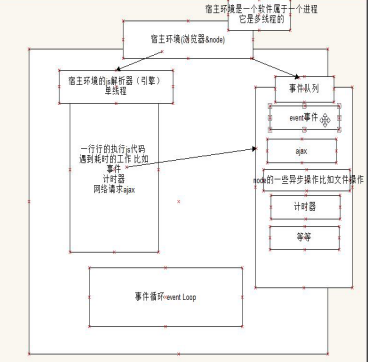
Path.join(“”，””，””)：拼接路径：遇到../翻到当前上一级，./进入当前下一级。

path.basename（）：文件名

path.dirname（）：文件所在磁盘路径

path.extname（）：获取文件的扩展名

单线程和异步



Commom.js规范：

模块化=》是一种编程思想，

引入模块，requre（“引入模块，每个js文件也是一个模块”）

核心模块：fs/path

第三方模块。直接下载，包管理工具npm下载，install jquery 。requre(“文件名”)

用户自定义模块：requre(“路径标识符”)

全局作用域，一个文件先把作为属性/方法挂在global上。另一个文件用requre引入文件在用global调用

向外暴露自己的模块：module.exports==两个都是一个对象。Module是（模块标识）

暴露成员exports：是一个对象，exports.add=10向外暴露成员。在外界引入这个文件后，可以直接用变量接收到文件就可以接收了。var m = requre(“文件1”)

Exports（系统默认经行了一个赋值操作把module的属性exporets也当成了一个变量，所以相等，如果它被重新赋值就不能直接暴露了）===Module.exports ==》默认都是指向统一空间（空间向外暴露）的空对象。都可以以对象方式向外暴露成员，但永远以Module.exports为准

全局包：在npm安装全局包（全局包安装在了npm中。）可以在电脑任何目录下随时可以直接调用,工具性包，比如文件类型转。换适合这种方式安装：npm install 包名 -g

卸载：npm uninstall 包名 -g（安装到包npm的全局目录中）===》

本地包：安装卸载本地包自己创建文件初始化后产生文件的node\_modules下面：在一个空文件夹调出端口进行初始化产生pacjage.json,里面是记录每个安装包的信息，比如名字版本，描述。

初始化：npm init -y---》在自定义文件夹生成一些自带的文件。比如node\_mudlea文件夹，package.json文件记录每次下载在本地包的信息--本地包也可以直接调用名字。全局包是用来直接操作使用的工具

如果是别人给的包用：npm/cnpm i 下载依赖项

--save 的缩写是 -S

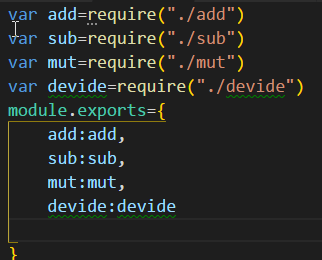
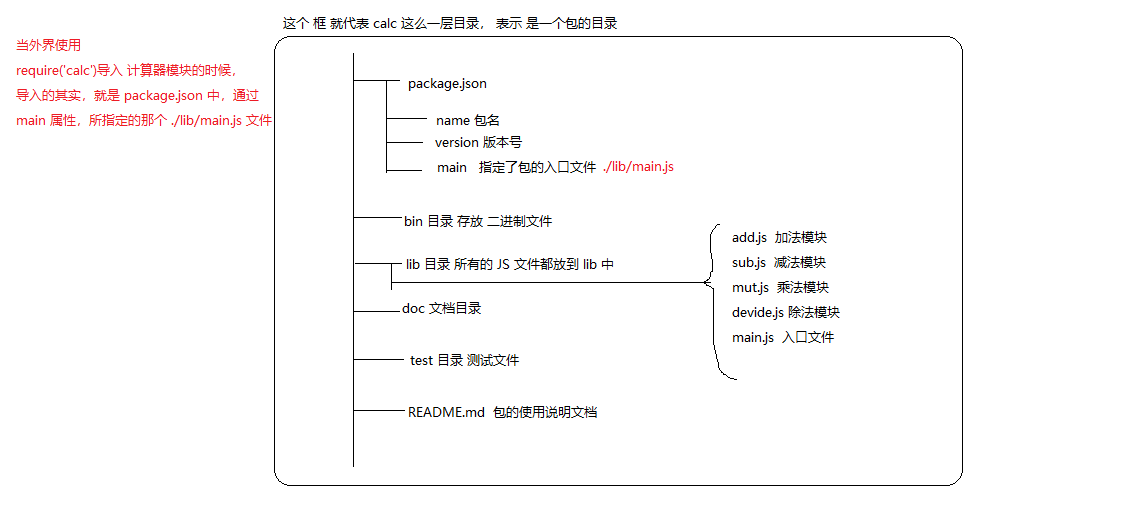
--save-dev的缩写是 -D

install 的缩写是 i

安装：npm i 包名 --save/-S -D/-d--save

删除：npm uninstall/remove 包名 ====下载cnpm解决下载包速度慢的问题

自定义包：cslc



http核心模块创建服务器：重新启动ctrl+c

1，const http = require(“http”)

2:创建服务器对象：const Server = http.creatServer()

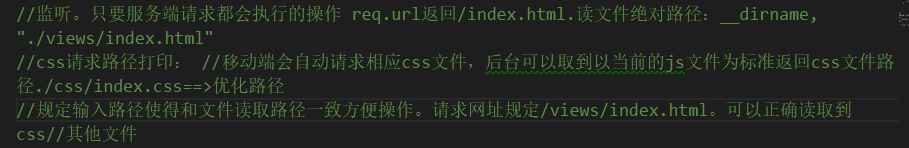
3:给对象绑定监听事件：server.on(“request”,(req,res)=>{})

Res.url=>返回客户端请求的路径/res.end(‘如果返回到客户端的字符串/二进制=》在前面设置到响应头，普通字符串解析是text/plain。Utf-8是防止中文乱码:res.writeHeader(200,{"Content-Type":"text/plain;charset=utf-8"})

如果不设置默认text/html解析’在这个里面可以去读取html文件fs.readfile()在把字符串结果结果传到服务器)

4：启动服务器：server.listen(3000,"127.0.0.1",function()

{ console.log("12");服务器启动后的执行事件 });



//下载全局包：Nodemon可以检测js文件改变不用手动重新启动，直接以nodemon “../”方式启动

//下载本地包。express/artTemplate/ejs==可以直接用require()引用后使用

//如果别人给你文件没有本地包，直接在此文件夹打开命令窗口 npm 环境输入node install

把所有相关联的包下载下来

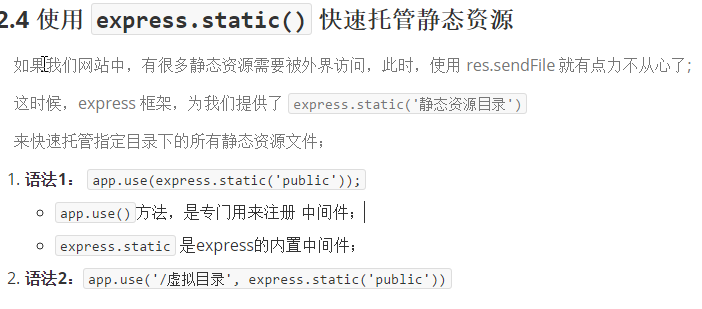
artTemplate模板引擎：可以在html文件<%....%>构建模板。在nodejs拿到html文件路径和渲染的数据返回数据到客户端页面：let reshtml = artTemplate( path.join(\_\_dirname,"./views/index.html"),{userData:arr})

res.end(reshtml);

Express:所有操作都是基于以下步骤，无论是原生还是引入的其他模板：1建服务器对象，监听对象，2当前台请求时触发监听事件，服务器在对象里面获取文件/读取文件，把文件拿到结合数据把页面渲染后返回给前台

Buffer:前台以文件下载的形式显示，sendfile:后台像前台发送静态页面。App.use(注册中间件)







配置模板引擎：使用 app.set('view engine', '模板引擎的名称')

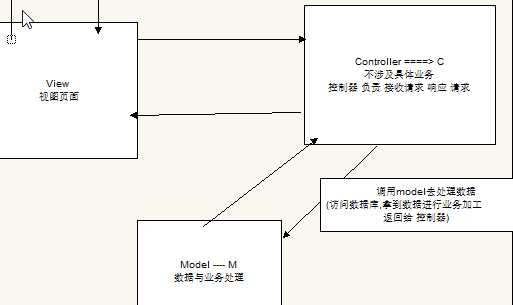
1. 设置模板页面的默认存放路径 app.set('views', '模板页面的具体存放路径目录名')

注意：如果想要调用 res.render（“模板文件后缀名跟模板引擎一致”，渲染的数据对象{}） 函数来渲染页面，必须先配置模板引擎

// 使用 res.redirect 方法，可以让 客户端重新访问 指定的页面

//fs.readdir能读到磁盘路径里面的自文件，以数组形式返回文件名filename

Mvc:后台路由分层



Cookie：

//服务器可以打印出/拿到第一次返回（客户端请求后服务端就再返回结果的响应头设置：'Set-Cookie'：【】，客户端第一子请求后再请求体保存一个 cookie：【】）给客户端的cookie值，给用户一个标识，可是客户端拿到的只一个key=value字符串，要解析成对象是因为后面方便通过属性判断判断，所有转化为对象，第一次请求客户端没有，req请求体没有值，cookie服务器有。但是第一次请求完毕客户端已经拿到并且存储起来。第二次请求客户端也携带了cooike值，但是服务端不用再返回cookie，只需要判断客户端传过来的标识

退出再服务端销毁cookie。即使再主页点击退出后cookie还存在但是没作用，下次登陆或者重新刷新后台不认识了，还是需要重新登陆

Session：

