Node.js

1.全局对象 Global Object

nodejs的全局对象 global.a

在脚本模式下，文件中声明的变量都是局部作用域下

js的全局对象 window.a

(1)console对象

console.log() 标准打印

console.info() 打印信息

console.warn() 打印警告

console.error() 打印错误

console.time(‘loop’)/console.timeEnd(‘loop’) 开始/结束计时，计算一段程序运行的时间，括号内的字符串必须一样

(2)process对象

process.arch 查看CPU架构类型 //x64

process.platform 查看操作系统的类型 //win32

process.version/ process.versions 查看nodejs的版本//v8.11.1

process.env 查看当前计算机环境变量

process.memoryUsage() 查看内存的使用情况

process.pid 查看当前进程的编号

process.kill() 杀死某个编号的进程

process.uptime() 查看nodejs的运行时间

process.nextTick(function(){})

(3)Buffer对象 缓冲区 默认存储的形式为Unicode编码，通过toString转换

global对象，用于将数据存储在缓冲区中，数据相对较小——Buffer数据

创建：Buffer.alloc(参数1，[参数2]); 参数1设置缓冲区的大小，参数2是缓冲区存储的值，格式是字符串

2.全局函数

定时器

(1)单次定时(一次性定时器)

开启：

setTimeout(function(){ },3000)/setTimeout(()=>{ },3000) 间隔3秒(以毫秒为单位)

清除：

clearTimeout(timer) //timer:变量名

例：var timer=null;

timer= setTimeout(()=>{

console.log(‘嘀嘀嘀嘀’);

},3000);

clearTimeout(timer);

(2)周期性定时器

开启：setInterval(function(){},3000)

清除：clearInterval(timer)

(3) process.nextTick(function(){}) 本次事件循环结束时立即执行的定时器 //nodejs的事件

(4) setImmediate(function(){})/clearImmediate() 下次事件循环立即执行的定时器

3.同步和异步

同步：程序按照顺序执行，后边的程序始终等待前边的程序执行完

异步：程序在执行的过程中，遇到异步处理函数，把这些操作放到整个程序的最后来执行。—— 异步非等待

特点：使用回调函数 ——回调次数过多会造成回调地狱(解决方法：Promise)

异步函数通常用于一些比较耗费时间、内存的代码。

例：var a=1;

SetTimeout(function(){

console.log(‘hello’);

},1000);

console.log(a); //1 hello 先打印a再打印hello

4.模块

(1)文件模块

任何一个文件模块，就是一个独立的作用域

Nodejs中模块文件的本质，是一个构造函数方法内的执行体，本质上被一个自动生成的函数包含

function(exports,require,module,\_\_filename,\_\_dirname){…}

exports：导出

require：导入模块 //require(‘./文件名’)

module：模块

\_\_filename：显示完整的路径和文件名称 // C:\xampp\htdocs\wxx\nodejs\_0817.js

\_\_dirname：显示完整的路径 // C:\xampp\htdocs\wxx

例：

//引入08\_1.js模块

var obj=require('./08\_1.js');

var res=obj.sum1(1,2);

console.log(res);

var res2=obj.cheng(3,5);

console.log(res2);

function sum(a,b){

return a+b;

}

function cheng(a,b){

return a\*b;

}

//公开sum——导出

module.exports={

sum1: sum,

cheng: cheng

}

1)带路径的文件模块

例：require('./08\_1.js') 通常用于用户自定义的模块

2)核心模块：不带路径的文件模块

例：require(‘querystring’) 用于官方定义的核心模块

3)带路径的目录模块 ——通常用于第三方模块

例：require(‘./2’) 2目录下存在index.js文件或者使用package.json声明main属性

引用目录模块会首先查找目录文件下的index.js文件；

如果目录下没有index.js文件，会查找package.json文件中main属性对应的文件。

如果index.js和package.json文件同时存在，使用package.json文件中main属性对应的文件

例：引入rect文件夹下的index.js文件 require('./rect'); //index.js

引入rect文件夹下的2.js文件 require('./rect'); //新建package.json文件 {"main":"2.js"}

4)不带路径的目录模块 ——通常用于第三方模块

例：require(‘3’) 要求把3目录放置node\_modules中，通常用于第三方模块

引入不带路径的文件 创建node\_modules文件夹

包和NPM(node package manage) npmjs.com

包：是一个目录模块，含有任意的js文件，含有一个命名为package.json包说明文件

淘宝NPM镜像(npm.taobao.org) 使用CNPM命令，需要单独安装 npm install cnpm --globa

改变目录：cd 路径

npm 包管理命令:

1. npm init –y 初始化package.json文件(记录包的安装信息)
2. 下载包：npm install 包名称 [--save(把包以运行依赖安装)]
3. npm install 包名称 –-save-dev把包以开发依赖安装
4. npm install 自动安装package.json文件中所保存的包信息(拷贝package.json文件)

(2)核心模块：

1)querystring模块 ——查询字符串模块

const querystring=require(‘querystring’);

//引入模块

var str=’lid=10&pname=apple&price=6250’; //解析一个查询字符串：在url中问号(?)后面的部分

var obj=querystring.parse(str); //对查询字符串解析，结果是对象 {lid:10,panme:apple,price:6250}

把对象转换为查询字符串：querystring.stringify()

2)url模块

得到一个url中的查询字符串

const url=require(‘url’);

var str=’http://www.codeboy.com/product\_details.html?...’;

var obj=url.parse(‘str’); //将url转换为对象

console.log(obj.query); //获取一个url中的查询字符串

例：将一个对象转换为url：

const url=require(‘url’);

var urlObj={

protocol:’https:’, //协议

hostname:‘www.tmooc.cn’， //主机名（域名）

pathname:’/course/100009.shtml’, //文件的路径

search:’?lid=10&pname=dell’

}

var str=url.format(urlObj);

console.log(str); // https://www.tmooc.cn/course/100009.shtml?lid=10&pname=dell

3)fs模块——文件系统模块

可以实现对文件操作(创建、写入、删除、读取…)

异步函数：非阻塞模式，通过回调函数来获取结果 fs.stat(path,callback)

同步函数：阻塞模式，通过函数的返回值来获取结果 fs.statSync(path)/fs.existsSync()

例1：查看文件状态

const fs=require('fs');

fs.stat('./1.txt',(err,stats)=>{ //err：查看失败显示的信息 stats：把查到的信息放入到stats中

if(err){ //如果有错误，抛出错误，程序不会往后进行

throw err;

}

console.log(stats);

console.log(stats.isFile()); //查看1.txt是否为文件

console.log(stats.isDirectory()); //查看1.txt是否为目录

});

例2：创建目录：fs.mkdir(path,callback) fs.mkdirSync(path)

const fs=require('fs');

fs.mkdir('./myDir',(err)=>{

if(err) throw err;

console.log('目录创建成功');

});

删除目录：fs.rmdir(path,callback)

练习：const fs=require('fs');

fs.stat('./myDir',(err,stats)=>{

if(err){

fs.mkdir('./myDir',(err)=>{

if(err) throw err;

console.log('目录创建成功');

});

}

})

例3：读取目录/文件 fs.readdir()/fs.readFile

const fs=require('fs');

fs.readdir('./myDir',(err,files)=>{

if(err) throw err;

console.log(files); //结果是数组形式

});

fs.readFile(‘./1.txt’,(err,data)=>{

if(err) throw err;

console.log(data.toString()); //data是一个Buffer数据，Unicode码

})

例4：文件创建1——借助写入文件的方法 fs.writeFile(file,data,callback)

如果文件不存在，会创建文件；如果文件已经存在，会清空文件的内容，然后再写入

const fs=require('fs');

fs.writeFile('./data.txt','hello',(err)=>{

if(err) throw err;

console.log('写入成功');

});

文件创建2——追加写入 fs.appendFile(file,data,callback)

如果文件不存在，会创建文件；如果文件已经存在，会在末尾写入

例5：删除文件 fs.unlink(file,callback)

const fs=require('fs');

fs.unlink ('./data.txt',(err)=>{

if(err) throw err;

console.log('删除成功');

});

练习：实现计数器功能，每运行1次，数字加1

const fs=require('fs');

var fn='./num.txt';

var res=fs.existsSync(fn);

if(!res){

fs.writeFileSync(fn,1);

}else{

fs.readFile(fn,(err,data)=>{

var num=data.toString();

num++;

console.log(num);

fs.writeFileSync(fn,num);

});

}

练习：获取url中查询字符串的值，并且把这些值写入str.txt中(如果不存在创建)

const url=require('url');

const querystring=require('querystring');

const fs=require('fs');

var str='http://www.codeboy.com/list.html?lid=10&pname=apple&price=5000';

var obj1=url.parse(str);

var oq=obj1.query;

var obj2=querystring.parse(oq);

for(var k in obj2){

fs.appendFileSync('./str.txt',obj2[k]);

}

5.HTTP模块

(1)HTTP协议：

web服务器和浏览器之间通信所遵循的协议。

浏览器发出请求，web服务器响应

请求方法：get 从服务器端获取内容(默认) post 向服务器端发送内容(注册、登录等)

请求的状态码：1\*\* 正在响应，还没有结束

2\*\* 响应成功

3\*\* 响应重定向到其它的url

4\*\* 客户端请求错误

5\*\* 服务器端运行错误

(2)http

例1：http模块

const http=require('http');

http.get('http://www.tmooc.cn/',(res)=>{ //res是响应的对象

console.log(res.statusCode); //状态码

console.log(res.headers); //响应的头信息

res.on('data',(buf)=>{ //响应的数据 Buffer格式

console.log(buf.toString()); //如果数据比较大，分多次返回

});

});

例2：构建web服务器

const http=require('http');

var server=http.createServer();

server.listen(3000,()=>{ //监听端口 127.0.0.1:3000

//console.log('web服务器构建成功');

});

//接收客户端的请求

server.on('request',(req,res)=>{ //事件：当有浏览器发送请求的时候，触发这个事件

console.log('接收到请求');

console.log(req.method); //请求方法

console.log(req.url); //请求的url

console.log(req.headers); //请求的头信息

res.writeHead(302,{ //响应的头信息

//'Content-type':'text/html',

//'Server':'web1807'

'Location':'http://www.tmooc.cn/'

});

res.write('this is web1807 homepage'); //发送到浏览器页面上

res.end(); //响应结束

});

6.express框架 ——构建web服务器的框架

基于node.js平台，快速、开放、极简的web开放框架。

中文官网：<http://www.expressjs.com.cn/> 下载安装：npm install express --sav-dev

路由

http客户端发来请求，服务器端根据请求的url，响应对应的内容

路由分为三部分:1) 请求的方法; 2）请求的url; 3) 响应的方法（回调函数）

例：

const express=require('express');

var app=express();

app.listen(3000,()=>{});

app.get('/login',(req,res)=>{

console.log('get responses');

console.log(req.headers); //请求的头信息

console.log(req.url); //请求的url：/login

console.log(req.method); //请求的方法：get

console.log(req.query); //请求url的查询字符串：{…}

res.send('this is web1807'); //服务器向浏览器发送响应的内容——状态码/string/buffer

res.sendFile(\_\_dirname+'/login.html'); //把登录的login.html文件发送给浏览器,绝对路径

res.redirect('http://www.tmooc.cn/'); //响应的重定向

});

注：send/sendFile只能使用一次

**get/post**

get 可以通过表单提交，也可以在浏览器地址栏直接输入，会暴露数据

post只能使用表单提交(现阶段)，不会在地址栏暴露输入的数据，相对安全，执行速度比get慢

例：

const express=require('express');

const querystring=require('querystring');

var app=express();

app.listen(3000,()=>{

console.log('服务器创建成功');

});

app.get('/login',(req,res)=>{

res.sendFile(\_\_dirname+'/login.html');

});

app.post('/member',(req,res)=>{

res.send('登录成功');

req.on('data',(buf)=>{

var str=buf.toString(),

obj=querystring.parse(str)

console.log(obj);

});

});

app.get('/reg',(req,res)=>{

res.sendFile(\_\_dirname+'/register.html');

});

app.get('/index',(req,res)=>{

res.send('注册成功');

});

路由中的参数

服务器端配置路由参数

例：

const express=require('express');

var app=express();

app.listen(3000,()=>{});

app.get('/list/:lid/:name',(req,res)=>{ //形参->地址:127.0.0.1:3000/list/13/dell

res.send('这是商品的详细信息');

console.log(req.params); //获取传递参数对象 { lid: '13', name: 'dell' }

});

路由器

把相关的路由归纳到一起，便于管理，最终应用于web服务器

用户模块：注册，登录，列表，修改，删除，检索……

商品模块：列表，详情，修改，删除

const express=require('express');

var router=express.Router();//调用

router.get('/list',(req,res)=>{

res.send('这是商品列表页面');

});

router.get('/details',(req,res)=>{

res.send('这是商品详情页面');

});

router.post('/add',(req,res)=>{

res.send('这是添加商品页面');

});

module.exports=router;

const express=require('express');

const product=require('./routers/products.js'); //导入路由器

var app=express();

app.listen(3000,()=>{

console.log('服务器创建成功');

});

app.use('/products',product); //将路由挂载到文件中

Require引入第三方模块机制：

首先查找当前目录下是否有node\_modules，如果没有，就往上一级目录查找，直到根目录

**中间件**

本质上，一个express应用就是在调用各种中间件

中间件就是express的函数，作用是为主逻辑服务，中间件要配合其他的中间件或路由来实现完整的业务逻辑

中间件拦截路由：

app.use(fn) //拦截所有的路由

app.use(‘/list’,fn) //拦截特定的路由(/list)

分类：应用级中间件 app.use((req,res,next)=>{…next();});

路由级中间件——路由器的使用 app.use(‘/user’,user);

错误处理中间件

内置中间件

第三方中间件

(1)内置中间件 ——**static**托管静态资源

sendFile只有在请求的时候，才会发送文件到浏览器。

app.use(express.static(‘目录路径’))

static 可以托管多个静态目录，如果两个目录中有相同的文件名称，显示第一个目录下的文件。

(2)第三方中间件

**body-parser** 对post请求的数据进行获取

register.html：

<form method='post' action='/reg'>

用户：<input type='text' name='uname'><br>

密码：<input type='text' name='upwd'><br>

邮箱：<input type='text' name='email'><br>

手机号：<input type='text' name='phone'><br>

<input type='submit'>

</form>

const express=require('express');

const bodyParser=require('body-parser');

var app=express();

app.listen(3000);

app.use(express.static('./public'));

app.use(bodyParser.urlencoded({

extended:false

}));

app.post('/reg',(req,res)=>{

console.log(req.body);

res.send('注册成功，你好'+res.body.uname);

});

**mysql**

1.普通连接：

const mysql=require('mysql');

var con=mysql.createConnection({

host:'127.0.0.1',

port:3306,

user:'root',

password:'',

database:'tedu'

});

con.connect();

con.query('select\*from emp',(err,result)=>{

if(err) throw err;

console.log(result[0]);

});

2.连接池连接：

const mysql=require('mysql');

var pool=mysql.createPool({

host:'127.0.0.1', port:3306, user:'root', password:'',

database:'tedu', connectionLimit:20 //连接池的大小

});

pool.query('SELECT\*FROM dept',(err,result,fields)=>{

if(err) throw err;

//console.log(result);

console.log(fields);

});

pool.query('DELETE FROM emp WHERE eid=?and sex=?',

[10,0],(err,result)=>{ //?占位符，防止sql注入

if(err) throw err;

console.log(result);

});

pool.query('UPDATE emp SET salary=? WHERE eid=?',[3000,5],(err,result)=>{ //?占位符，防止sql注入

if(err) throw err;

console.log(result);

});

var obj={dname:'后勤部',did:60}; //60 后勤部

pool.query(`INSERT INTO dept SET ?`,

obj,(err,result)=>{ //?占位符，防止sql注入

if(err) throw err;

console.log(result);

});

const express=require('express');

const bodyParser=require('body-parser');

const mysql=require('mysql');

var app=express();

app.listen(4000,()=>{});

app.use(express.static('./public'));

app.use(bodyParser.urlencoded({

extended:false

}));

var pool=mysql.createPool({

host:'127.0.0.1',

port:3306,

user:'root',

password:'',

database:'tedu',

connectionLimit:20

});

app.post('/dept',(req,res)=>{

var obj=req.body;

console.log(obj);

pool.query(`INSERT INTO dept SET ?`,obj,(err,result)=>{

if(err) throw err;

res.send('数据添加成功');

});

});

<form method='post' action='/dept'>

部门编号： <input type='text' name='did'><br>

部门名称： <input type='text' name='dname'><br>

<input type='submit'>

</form>