nodejs

zepto 移动端版jq

nodejs是一个系统框架，专门实现把JS跑在浏览器以外的地方，nodejs其实就是用JS的语法

模块

fs(文件系统) http url mysql querystring express body-parser(用于处理post过来的数据) multer(用于解析图片form-data) jade cheerio(nodejs JQ模块)

# 协议，域名，端口，路由，哈希值

<http://localhost:3000/route?name=laoxie>

协议:http 域名:localhost 端口:3000 路由:/route 哈希值: ?name=laoxie

# 模块系统

1. \*\*自定义模块\*\* 自己封装的方法

2. \*\*第三方模块\*\* 别人写好的模块(就是从NPM下载的模块) 比如gulp

3. \*\*内置模块\*\* nodejs自带的模块(不用下载，就是存在nodejs内部的模块)

module.export.名字 = 数据类型/函数;

使用的时候

var a = require(‘改模块路径’)；

a．名字…

# 接收ajax请求数据的方式

## get

原生用于接收get请求的

var http = require(‘http’);

http.createServer(function(request,response){

var url = require(‘url’);

var querystring = require(‘querystring’);

//获取get的数据

var params = url.parse(request.url).query;

//将get数据变成对象

var par = querystring.parse(params);

response.setHeader(‘Access-Control-Allow-Origin’, ‘\*’);

//返回前端的内容

response.end(‘返回的内容’)=>php echo

}).listen(‘端口’)

express模块

var express = require(‘express’);

var app = express();

app.get(‘/路由’,function(req,res){

//获取get的数据

var params = req.query;(对象)

res.append(‘Access-Control-Allow-Origin’, ‘\*’)

//返回前端的内容

res.send(‘返回的内容’);

})

var server = app.listen(端口号);

## Post

原生用于接收post请求的

var http = require(‘http’);

http.createServer(function(request,response){

var querystring = require(‘querystring’);

//获取post的数据

var post = ‘’;

request.on(‘data’,function(chunk){

post += chuck;

})

request.on(‘end’,function(){

//将post数据变成对象

var params = querystring.parse(post);

response.setHeader(‘Access-Control-Allow-Origin’, ‘\*’);

//返回前端的内容

response.end(‘返回的内容’)=>php echo

})

}).listen(‘端口’)

express模块

var express = require(‘express’);

var bodyParser = require(‘body-parser’);

var app = express();

app.use(bodyParser,urlencoded({extended:false}));

app.use(bodyParesr.json());

app.post(‘/路由’,function(req,res){

//获取post的数据(对象)

var params = req.body;

res.append(‘Access-Control-Allow-Origin’, ‘\*’)

//返回前端的内容

res.send(‘返回的内容’);

})

var server = app.listen(端口号);

## Jsonp

ajax前端写法

$.ajax({

[url:"路径](url:%22路径)",

type:"get",

dataType:"jsonp",

jsonpCallback:"回调函数name",

success:function(data){

console.log(data)

}

})

后端返回写法

http.createServer(function(request,response){

//把我们参数部分截取出来

var paramStr = url.parse(request.url).query;

var param = querystring.parse(paramStr);

//解决跨域

response.setHeader("Access-Control-Allow-Origin","\*");

//相应结果显示浏览器上,返回回调函数加数据

response.end(param["callback"]+数据);

}).listen(12345)

# Mysql数据库链接

var mysql = require(‘mysql’);

var connection = mysql.createConnection({

host:’域名’,

user:’用户名’,

password:’密码’，

database:’数据库名’

});

connection.connect();

//sql语句

connection.query(‘sql语句’，function(error,request,field){

if(error) throw error;

})

connection.end();

## 拓展sql语句增删查改

增 insert into 表名 (表头) values (值)

删 delete from 表名 where id = 值…

查 select \* from 表名 limit 值,值…

改 update 表名 set name = ‘值’where id = ‘值’…

清空 truncate table 表名

# 爬虫(后端链接后端)

第一种

http.request({

hostname:’域名’,

port:’端口’,

path:’路由+哈希值’,

method:’GET’(一般用get),

},function(res){

var data = ‘’;

res.on(‘data’,function(chunk){

data += chunk;

})

res.on(‘end’,function(){})

}). on('error', function(e) {

console.log('problem with request: ' + e.message);

}).end();

第二种

http.get(‘网址’，function(res){

var data = ‘’;

res.on(‘data’,function(chunk){

data += chunk;

})

res.on(‘end’,function(){})

})

# 下载

http.get(‘网址’，function(res){

var data = ‘’;

res.on(‘data’,function(chunk){

data += chunk;

})

res.on(‘end’,function(){

//res就是返回的流数据，整段html结构，然后按照常规方法获取标签处理数据

var imgArr = [];

//把爬到的地址链接push进imgArr

imgArr.push(‘链接’)

//执行下载函数

down(imgArr);

})

})

var i = 0;

function down(imgArr){

var length = imgArr.length;

http.get(imgArr[i],function(res){

var writeStream = fs.createWriteStream('./'+i+Date.now()+'.jpg');

//递归

if(i<=length){

i++;

down(imgArr);

console.log('当前'+ i + '总共' + length)

}

res.pipe(writeStream);

})

}

# 文件系统

var fs = require(‘fs’);

fs.readFile(‘路径’,function(err,data){})

fs.writeFile(‘路径’,data,function(err,data){})

# Stream流数据

var readerStream = fs.createReadStream(‘路径’)；

var writeStream = fs.createWriteStream(‘路径’);

readerStream.pipe(writeStream);

前面部分只要是流数据皆可替换，例如爬虫技术返回的res就是流数据

# 静态文件夹

app.use(express.static(‘路径’))

路由没有后缀，进入接口；如果有后缀，返回该文件。如果没有匹配路由，就去静态文件夹里面找文件，然后返回到浏览器上。

# 上传

**详见自己的upload文件**

# websocket

socket.on/socket.emit(监听/发送给置顶客户端)

io.on/io.emit(监听/发送所有客户端)

前端写法

先引入前端用的socket.js

<script src=’socket.js’></script>

var socket = io(‘connection’,’地址’);

//监听后端返回的消息

socket.on(‘发送消息的名字(必须与后端一致)’,function(data){})

//发送消息

socket.emit(‘发送消息的名字(必须与后端一致)’,’内容’)

后端写法

var http = require(‘http)’;

var ioFn = require(‘socket.io’);

//创建服务器

var app = http.createServer(function(req,res){})

//实例化服务器，让服务器支持websocket

var io = ioFn(app)

//前后端建立联系

io.on(‘connection’,function(socket){

socket.on(‘发送消息的名字(必须与前端一致)’,function(data){

//socket.emit(‘发送消息的名字(必须与前端一致)’,’内容’)

})

})

app.listen(‘端口号’)