# NodeJavaScript

## nodeJS概述

### NODEJS运行

#### 交互模式 node

exit 退出

Ctrl+C 两次 退出

#### 脚本模式

cd + C:/xampp..../nodejs/day01

node 01.js

shift + 桌面空白区域鼠标右键，选择Powershell窗口

### NodeJS的特点

#### 优势

1. 简单，避免过度设计
2. 单线程逻辑处理
3. 非阻塞的异步I/O处理
4. 事件驱动编程
5. 无锁机制，不会产生死锁
6. 支持数万个并发连接
7. 适合搭建以IO操作为主，响应速度快，易于扩展的网络应用
8. 基于社交网络的大规模Web应用

#### 劣势

1. 不适合CPU密集型应用
2. 深层次的嵌套和递归
3. 复杂加密和解密算法
4. 高可靠性运算
5. 严格内存管理
6. 数据挖掘和数据分析

总结：

NodeJS是服务器端语言，和Java，PHP，.net性质一样。

## NodeJS全局对象

全局作用域下的变量，可以通过全局对象来访问。

global是NODEJS下的全局对象

NODEJS每一个js文件都属于局部作用域，里边声明的变量属于局部变量，不能使用global来访问。

window是JS下的全局对象

JS下每一个js文件下都属于全局作用域，里边声明的变量属于全局变量，可以使用window来访问。

例：在浏览器中打开

var a = 1;

//使用js的全局对象window来访问

console.log(a); //1

console.log(window.a); //1

例：在node中打开

var a = 1;

console.log(a); //1

//a不是全局作用域下的变量，不能使用global访问

console.log(global.a); //undefined //a不是全局作用域下的变量 global,全局对象

### 全局对象

#### console对象

console.log() 打印消息

console.info() 打印消息

console.warn() 打印警告消息

console.error() 打印错误消息

例：//在node里面没区别，在浏览器中有明显区别

console.log(123); //123//log打印

console.info(123); //123//info信息

console.warn(123); //123//warn警告

console.error(123); //123//error错误

#### 计算程序的运行时间

console.time('**loop**'); //开始计时

检测的程序

console.timeEnd('**loop**');

例：

console.time('for循环运行时间');

var sum = 0;

for (var i = 0; i < 10; i++) {

sum += i;

}

console.log(sum);

console.timeEnd('for循环运行时间'); //可显示程序运行时间，括号内内容需完全相同

练习：比较while循环和do-while执行10万次，哪个时间更短

console.time('while时间');

var i = 0;

var sum = 0;

while (i < 100000) {

sum += i;

i++;

}

console.log(i);

console.timeEnd('while时间');

console.time('do-while时间');

var i = 0;

var sum = 0;

do {

sum += i;

i++;

}

while (i < 100000);

console.log(i);

console.timeEnd('do-while时间');

#### process对象（了解）

arch 查看CPU的架构类型

platform 查看操作系统类型

version 查看当前nodejs的版本号

env 查看当前计算机的环境变量有哪些

例：

process.arch //cup架构

'x64'

process.platform //系统

'win32'

process.version //node版本

'v8.11.1'

process.env //环境变量

#### Buffer对象

Buffer:缓冲区,值一块专用于存储数据的内存区域,可用于存储读写的文件数据、网络上要传输的数据等等。

Buffer对象的实例，可直接构造，也可以通过数据读写获得。

Buffer实例中，不仅能存储字符数据，也可以存储二进制的字节数据。

注意：不同版本的Node.js中，Buffer对象的API变化较大，请参考当前版本的Node.js手册。

### 全局函数

定时器

#### 一次性定时器

开启

var time=null;

timer=setTimeout(参数1,参数2);

参数1:回调函数，间隔的时间到了，执行的命令。

参数2:间隔的时间，单位毫秒

清除

clearTimeout(timer);

例：

//开启一次性定时器

var timer = null;

timer = setTimeout(function () {

console.log('嘀嘀嘀');

}, 3000);

//清除一次性定时器

clearTimeout(timer);

#### 周期性定时器

使用方式和一次性定时器完全一致。

setInterval()/clearInterval()

练习：查看周期性定时器函数的使用

var timer = null;

timer = setInterval(function () {

console.log('lalala');

}, 2000);

clearInterval(timer);

#### process.nextTick(回调函数)

例：

process.nextTick(function () {

console.log('滴答');

});

#### setImmdiate

开启

var timer=null;

setImmdiate(回调函数)

清除 clearImmdiate(timer);

例：

var timer = null;

timer = setImmediate(() => {

console.log('hello');

});

clearImmediate(timer);

#### 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，才会执行。——阻塞

异步：程序运行过程中，遇到一些耗时较久的程序，把这些程序放到整个程序的最后。——非阻塞

回调函数多为异步

parseInt/ parseFloat/ eval/ isNaN/ isFinite/ encodeURI...

例：打印结果：1,2,123；与时间设定无关,是因为非阻塞

console.log(1);

setTimeout(() => {

console.log(123);

}, 1000);

console.log(2);

例：打印结果：1,2,4

console.log(1);

process.nextTick(() => {

console.log(4);

});

console.log(2);

例：打印结果1,2,3

console.log(1);

setImmediate(() => {

console.log(3);

});

console.log(2);

## 模块

### 模块的分类

nodejs下分为核心模块（nodejs提供）、第三方模块（个人组织写好的）、自定义模块（自己写的）。

### 模块封装

在nodejs下每一个模块文件的本质——是被一个构造函数所包含的

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

程序员写的代码

})

红色内容是nodejs自动为每一个文件添加的

### 相关函数

\_\_filename: 当前模块的完整路径和文件名

\_\_dirname: 当前模块的完整路径

require: 用于引入一个模块，返回一个对象

module: 指代当前的模块；

module.exports 当前模块导出的对象，可以供其他的模块使用。

exports: 等同于module.exports

|  |  |  |
| --- | --- | --- |
|  | 以路径开头 | 不以路径开头 |
| 文件模块 | require('./circle.js')  通常用于用户自定义的模块 | require('querystring')  用于引入官方提供的核心模块 |
| 目录模块 | require(./02\_1);  在目录下存储一个index.js文件，或者使用package.json文件来指定引入的文件 | require('2');  要求把目录2放入当前目录下的node\_modules中；  通常用于引入第三方模块 |

例：

var res = require('./06\_1.js'); //同级一定要加./，否则会报错

例：

console.log(\_\_filename); //文件的完整路径和文件的名称//C:\xampp\htdocs\sxs\03\_NODEJS\day01\06\06\_1.js

console.log(\_\_dirname); //文件的完整路径//C:\xampp\htdocs\sxs\03\_NODEJS\day01\06

例：

console.log(module.exports == exports);

module.exports = {

a: 1,

addNum: function (num1, num2) {

return num1 + num2;

}

}

例：

//核心模块

console.log('这是主模块');

//导入功能模块

//var res=require('./06\_1.js');

//console.log(res.a);

//console.log(res.addNum(3,5));

//需要计算长方形的周长和面积

var rect = require('./06\_2.js');

console.log(rect.getLength(3, 2));

console.log(rect.getArea(3, 2));

//功能模块06\_1

console.log('这是功能模块1');

var a = 1;

console.log(module.exports == exports);

module.exports = {

  a: 1,

  addNum: function (num1, num2) {

    return num1 + num2;

  }

}

//功能模块06\_2

//公开计算长方形的周长和面积两个方法

module.exports = {

  getLength: function (w, h) {

    return 2 \* (w + h);

  },

  getArea: function (w, h) {

    return w \* h;

  }

}

//导入主模块，可得

练习:创建主模块，创建一个功能模块，用于计算圆的周长和面积。

//主模块

console.log('这是主模块');

var rect = require('./01\_1.js');

console.log(rect.getLength(2));

console.log(rect.getArea(2));

//功能模块

//创建一个功能模块，用于计算圆的周长和面积。

module.exports = {

  getLength: function (r) {

    return 3.14 \* 2 \* r;

  },

  getArea: function (r) {

    return 3.14 \* Math.pow(r, 2);

  }

}

练习：创建一个03\_1.js，分别引入03\_2目录模块（有index.js），引入03\_3目录模块（有rect.js）

//创建03\_1.js文件

require('./03\_2');

require('./03\_3');

//在同文件内建立03\_2和03\_3文件夹

//在03\_2文件夹内建立index.js

console.log('./03\_2/index.js');

//在03\_3文件夹内建立rect.js和package.json文件

//在rect.js文件内

console.log('./03\_3/rect.js');

//在package.json文件内

{

  "main": "rect.js"

}

//导入03\_1.js文件可得

注：.jiso 文件要求属性名必须加双引号

例：

//带路径的模块

//建立main.js文件

require('2');

//在同文件内建立node\_modules文件夹

//在node\_modules文件夹内建立2文件夹

//在2文件夹内建立index.js文件

console.log('./node\_modules/2/index.js');

//导入main.js文件可得

练习：引入目录模块circle，里边文件名称circle\_1.js

//不带路径的模块

//建立新文件夹

//建立main.js文件

require('circle');

//在同文件内建立node\_modules文件夹

//在node\_modules文件夹内建立circle文件夹

//在circle文件夹内分别建立circle\_1.js和package.json文件

//在circle\_1.js文件内

console.log('circle\_1.js'); //输出文件

//在package.json文件内

{

  "main": "circle\_1.js" //关联circle\_1.js文件

}

//导入main.js文件可得

### 包和npm

（1）包：就是一个目录模块，里边含有多个文件，其中有一个命名为package.json的包说明文件。

（2）npm -> node package manage

cd切换到要下载的目录

npm install 包的名称

### 工具模块

#### querysting模块

查询字符串模块

'lid=20&name=tom'

parse 将查询字符串转为对象（服务器端解析传递的数组）

stringify 将对象转为查询字符串（浏览器发送数据）

例：

//nodejs提供一个模块

const querystring = require('querystring'); //一般核心模块就作为常量名称

var str = 'lid=5&name=tom&age=18';

//把查询字符串解析为对象

var obj = querystring.parse(str);

console.log(obj); //{ lid: '5', name: 'tom', age: '18' }

//把对象转换为查询字符串

var product = {

  pid: 25,

  name: 'dell',

  price: 4000

}

var str = querystring.stringify(product);

console.log(str)

//pid=25&name=dell&price=4000;

练习：把百度网址的查询字符串解析为对象；

ie = utf - 8 & f = 8 & rsv\_bp = 0 & rev\_idx = 1 & tn = baidu & wd = 手机

var baiduStr = 'ie=utf-8&f=8&rsv\_bp=0&rev\_idx=1&tn=baidu&wd=手机';

console.log(querystring.parse(baiduStr));

#### URL模块

parse 将路径解析为对象

protocol 协议

hostname 主机名（域名）

port 端口

pathname 文件在服务器的路径

query 查询字符串

format 将对象转换成路径

注意：query对应的是一个对象

一个网址包含：协议，域名，端口，路径，查询对象，这5个部分

例：

//引入url模块

const url = require('url');

var str = 'http://www.codeboy.com:80/course/1.html?lid=10&name=tom&age=18';

//使用URL模块下的方法解析路径

var obj = url.parse(str);

console.log(obj);

/\*

Url {

protocol: 'http:',

slashes: true,

auth: null,

host: 'www.codeboy.com:80',

port: '80',

hostname: 'www.codeboy.com',

hash: null,

search: '?lid=10&name=tom&age=18',

query: 'lid=10&name=tom&age=18',

pathname: '/course/1.html',

path: '/course/1.html?lid=10&name=tom&age=18',

href: 'http://www.codeboy.com:80/course/1.html?lid=10&name=tom&age=18' }

\*/

console.log(obj.query); //lid=10&name=tom&age=18

练习：将一个路径中查询字符串解析为对象

//http://www.tmooc.cn:8080/images/list.html>pid=10&pname=dell&price=5000

//用url模块解析出查询字符串

var obj = url.parse(str);

//通过url对象获取查询字符串

var qs = obj.query;

//通过querystring解析查询字符串

var obj2 = querystring.parse(qs);

console.log(obj2);

例：将一个对象再转换为路径

const url = require('url');

var obj2 = {

  protocol: 'http',

  hostname: 'www.baidu.com',

  port: 3000,

  pathname: '/products/list.html',

  //search: 'lid=5&name=tom'

  query: {

    lid: 5,

    name: 'tom'

  }

}

var res = url.format(obj2);

console.log(res);

例：引入url模块

const url = require('url');

const querystring = require('querystring');

var str = 'http://www.tmooc.cn:8080/images/list.html>pid=10&pname=dell&price=5000';

//用url模块解析出查询结果

var obj = url.parse(str);

//通过url对象获取查询字符串

var qs = obj.query;

console.log(qs);

var obj2 = querystring.parse(qs);

console.log(obj2);

练习：如何使用对象转成路径

//http://www.tmooc.cn:8080/images/list.html?pid=10&pname=dell&price=5000

const url = require('url');

var obj = {

  protocol: 'http',

  hostname: 'www.tmooc.cn',

  port: 8080,

  pathname: '/images/list.html',

  query: {

    pid: 10,

    pname: 'dell',

    price: 5000

  }

}

var str = url.format(obj);

console.log(str);

#### Buffer对象

buffer 缓冲器

创建buffer

Buffer.alloc(size,fill)

size，buffer空间的大小；fill，填充的内容。

buffer数据转为普通字符

toString()

例：//创建一个缓冲区

var buf = Buffer.alloc(5, 'abc');

console.log(buf); //<Buffer 61 62 63 61 62>

//将buffer数据转为普通字符

console.log(buf.toString()); //abcac

练习：创建一个buffer空间，存储'hello world';分别打印buffer数据和普通字符形式。

var buf = Buffer.alloc(11, 'hello world');

console.log(buf); //<Buffer 68 65 6c 6c 6f 20 77 6f 72 6c 64>

console.log(buf.toString()); //hello world

#### 文件系统模块

fs.stat (path,callback) 查看文件的状态

path 文件的路径；

callback 回调函数；

fs.startSync(path)

path 文件路径

sync表同步

##### 对比同步方法和异步方法的区别

同步方法会阻止后续的执行，是通过返回值来得到结果。

异步方法不会阻止后续的执行，是通过回调函数来得到结果

例：1. 查看文件的状态

//引入文件系统模块

const fs = require('fs');

//查看文件的状态

法1

fs.stat('05.js',function(err,stats){

//err，如果出现错误，会存到这个里边

if (err){

//抛出错误，一旦有错误，后续的代码将无法再执行

throw err;

}

console.log('执行成功');

//stats，文件的状态信息

//如何查看是否为 文件

console.log(stats.isFile());

//如何查看是否为 目录

console.log(stats.isDirectory());

});

console.log(2);

//为异步函数

法2

同步方法：是通过返回值来获取结果

const fs = require('fs');

var stats = fs.statSync('05.js'); //同步函数的方法是把变量放到返回值

console.log(stats.isFile());

console.log(stats.isDirectory());

console.log(2);

#### FS模块

fs模块，使用前需要引入fs模块（var fs= require(“fs”);）

fs.kldir 创建目录

fs.rmdir 删除目录

fs.readdir 读取目录，读取的数据是数组的形式

fs.readFile 读取文件，读取的文件类型是buffer形式数据

fs.writeFile(path,data,callbank) 写入文件，如果文件不存在，则会创建；如果文件已存在，清空里边的内容；然后写入；

data 是要写入的数据

fs.appendFile(path,data,callback) 写入文件，如果文件不存在，则会创建；如果文件已经存在，在文件的末尾写入数据。

data 是要写入的数据

fs.unlink 删除文件

fs.exists**Sync** 判断文件是否存在；存在true，不存在false

fs.rename(oldPath,newPath,[callback(err)]) 修改文件名称，可更改文件的存放路径

oldPath 原路径

newPath 新路径

callback 回调，传递一个err异常参数

例：2.创建目录

法1

const fs = require('fs');

fs.mkdir('files', function (err) {

  if (err) {

    throw err;

  }

  console.log('目录创建成功');

}); //创建名为'files'的文件夹

法2

const fs = require('fs');

var res = fs.mkdir('files2');

console.log(res);

例：3.删除目录

const fs = require('fs');

fs.rmdir('files', function (err) {

  if (err) throw err;

  console.log('删除成功');

});

例：4.读取目录

const fs = require('fs');

fs.readdir('03\_3', function (err, files) {

  if (err) throw err;

  console.log(files); //为对象形式

}); //[ 'package.json', 'rect.js' ]

例：5.读取文件

const fs = require('fs');

fs.readFile('05.js', function (err, data) {

  if (err) throw err;

  console.log(data);

});

例：6.写入文件——清空写入

const fs = require('fs');

fs.writeFile('num.txt', 'hello', function (err) {

  if (err) throw err;

  console.log('写入成功');

});

例：7.写入文件——追加写入

const fs = require('fs');

fs.appendFile('num.txt', 'world\n', function (err) {

  if (err) throw err

  console.log('ok');

}); //不能用记事本打开，用编辑器打开才有换行的效果

例：8.删除文件

const fs = require('fs');

fs.unlink('num.txt', function (err) {

  if (err) throw err;

  console.log('ok');

});

例：9.判断文件是否存在————同步方法

const fs = require('fs');

var res = fs.existsSync('num.txt');

console.log(res);

例:10.删除文件————判断文件num.txt是否存在，如果存在就删除，否则打印“此文件不存在”。

const fs = require('fs');

if (fs.existsSync('num.txt')) {

  fs.unlink('num.txt', function (err) {

    if (err) throw err;

    console.log('删除成功');

  });

} else {

  console.log('此文件不存在');

}

练习：计数器：把数字保存在num.txt中，每次运行读取文件中的数字。

1. 判断文件是否存在如果不存在创建num.txt，初始值为0；
2. 读取num.txt中的内容，将数字加1（num++），并打印出来
3. 把加1的结果再次写入到num.txt中

const fs = require('fs');

if (fs.existsSync('num.txt')) {

  fs.readFile('num.txt', function (err, file) {

    if (err) throw err;

    fs.writeFile('num.txt', parseInt(file.toString()) + 1, function (err) {

      if (err) throw err;

    });

    console.log(Number(file.toString('utf8')) + 1);

  });

} else {

  fs.writeFile('num.txt', '0', function (err) {

    if (err) throw err;

    console.log('写入成功');

  });

}

推荐答案

const fs = require('fs');

//计数器

//1.初始化保存数字的文件——0

//用到同步

//判断是否有num.txt，如果没有则创建文件，初始值0

var res = fs.existsSync('num.txt');

if (!res) { //!false

  fs.writeFileSync('num.txt', 0); //必须用同步方法

}

//2.读取文件中的数字，打印出来

//把数字加1，再次保存到文件中

fs.readFile('num.txt', function (err, data) {

  var num = data.toString();

  num = parseInt(num);

  num++;

  console.log(num);

  //把num写入到文件中，先清除之前的内容再写入

  fs.writeFile('num.txt', num, function (err, data) {

    if (err) throw err;

  });

});

### HTTP

#### HTTP协议

HTTPS协议是在HTTP协议的基础上就安全进行了升级。

HTTP：用于浏览器和Web服务器之间的通信协议。翻译为，超文本传输协议。

##### 通用头部General

Request URL 请求的URL

Request Method 请求的方法

get 用于从服务器端获取内容（浏览网页...）

post 向服务器端发送内容（注册、登录...）

Status Code 响应的状态码

1\*\*系列 请求-响应继续进行

2\*\*系列 成功的响应

3\*\*系列 响应的重定向

4\*\*系列 客户端错误

5\*\*系列 服务器端错误

Remote Address 请求的远程地址和端口

##### 请求的头信息Request headers

Accept 接受的文件类型

Connection：keep-alive 告诉浏览器持久连接

User-agent 使用的浏览器类型

Accept-Language 浏览器所接受的语言

##### 响应的头信息Reponse headers

Content-Type 响应主体类型是什么

text/html、text/css...

Connection：keep-alive 开启持久连接

#### HTTP模块

既可以模拟浏览器向服务器发请求，也可以创建web服务器。

##### 作为客户端浏览器

http.get(url,callback)

get 以get方法发请求

url 请求的url

callback 以回调函数形式获取服务器的响应

res.statusCode 获取相应的状态码

res.on('data', function (buf) { })

使用事件来获取服务端响应的数据

数据是分段传输，格式为buffer数据

例：

//1.使用http模块，模拟浏览器

const http = require('http');

//使用get方法向服务器发请求

http.get('http://www.codebody.com', function (res) {

  //res 服务器端做出的响应————对象

  console.log(res.statusCode); //statusCode，响应的状态码

  //获取响应的内容

  //通过一个事件

  //on('data')//当有数据传输时       //on，表示当有事件时

  res.on('data', function (buf) {

    console.log(buf.toString());

  });

});

练习：使用http模块模拟浏览器向以下url发请求，并打印响应的内容。

http: //www.codeboy.com/products.html

  const http = require('http');

http.get('http://www.codeboy.com/products.html', function (res) {

  console.log(res.statusCode);

  res.on('data', function (buf) {

    console.log(buf.toString());

  });

});

##### 创建web服务器

createServer 创建web服务器

listen 分配端口，监听端口的变化

server.on('request',function(req,res){ })

当有浏览器发起请求，调用事件

req 请求的对象

method 请求的方法 get/psot

url 请求URL，显示端口后边的路径

http://127.0.0.1/login

res 响应的对象

writeHead(statusCode,对象) 设置响应的头信息

对象中Location实现跳转

write（文本） 响应文本内容（显示在浏览器页面上）

end() 结束响应

练习：使用http创建web服务器，端口是3001；使用事件获取客户端的请求

const http = require('http');

var server = http.createServer();

server.listen(3000, function () {

  console.log('服务器创建成功');

});

//接收请求

server.on('request', function (req, res) {

  console.log('收到1个请求');

  //响应的对象——res

  /\*

  res.write('this is web 1808');

  res.write('hello world');

  \*/

  //重定向到其他的URL

  //跳转到学子商城的首页

  res.writeHead(302, {

    Location: 'http://www.codeboy.com',

    'Content-Type': 'text/html;charset=utf-8'

  });

  //响应结束

  res.end();

});

练习：使用http创建web服务器，根据浏览器请求的URL执行不同的操作。

/login  响应一行文本  'this is login page'  到浏览器 /

reg 响应一行文本 'this is reg page'

到浏览器

  /

  home 转到学子商城首页， http: //www.codeboy.com

  const http = require('http');

//创建web服务器

var server = http.createServer();

server.listen(3000);

//接收请求

server.on('request', function (req, res) {

  //console.log('收到请求');

  //获取请求的URL

  var url = req.url;

  console.log(url);

  //根据url的不同，来响应不同的内容——路由

  switch (url) {

    case '/':

      res.write('this is home page');

      break;

    case '/login':

      res.write('this is login page');

      break;

    case '/reg':

      res.write('this is reg page');

      break;

    case '/home':

      res.writeHead(302, {

        Location: 'http://www.codeboy.com'

      });

      break;

    default:

      res.write('404  NOT FOUND');

  }

  //结束响应

  res.end();

});

练习：在node中创建服务器

之后在浏览器内输入127.0.0.1:3000

输入127.0.0.1:3000/ //this is home page

输入127.0.0.1:3000/login //this is login page

输入127.0.0.1:3000/reg //this is reg page

输入127.0.0.1:3000/home //跳转到http://www.codeboy.com（学子商城）

输入127.0.0.1:3000/\* //404 NOT FOUND

//node会显示favicon.ico，其表示默认搜索小图标

const http = require('http');

var server = http.createServer();

server.listen(3000);

server.on('request', function (req, res) {

  var url = req.url;

  console.log(url);

  switch (url) {

    case '/':

      res.write('this is home page');

      break;

    case '/login':

      res.write('this is login page');

      break;

    case '/reg':

      res.write('this is reg page');

      break;

    case '/home':

      res.writeHead(302, {

        Location: 'http://www.codeboy.com'

      });

      break;

    default:

      res.write('404 NOT FOUND');

  }

  res.end();

});

#### Express

##### Express概述

Express是基于nodejs，用于构建web服务器的框架。

官网：www.expressjs.com.cn

注：包，如果下载时不经过特殊处理，若移动位置，会造成包无法使用

##### 使用Express创建服务器

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('服务器创建成功');

});

##### 路由

客户端发来请求，服务器程序会根据请求的URL和请求的方法（Method）来确定响应的内容。

路由：请求的URL，请求的方法，响应的内容

（浏览器端进行请求；服务器端负责响应）

###### 请求的对象req

req.method 请求的方法 get/post...

req.url 请求的URL /login...

req.headers 请求的头信息

JSON.stringify() 对象转换为字符串

JSON.parse() 将对象字符串解析为对象

req.query 请求的查询字符串，格式为对象

例：

//使用express创建web服务器

const express = require('express');

var app = express(); //创建

app.listen(3000, function () {

  console.log('服务器创建成功');

});

//使用get方法，请求url为/login

//只有两个条件都满足，才会做出以下的响应

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

  //app:代表服务器端

  //get：代表请求法方法

  //login：代表请求的url

  console.log('接收到了请求为get，url为/login');

});

练习：创建请求方法为get，url为/reg的路由

创建请求方法为get，url为/home的路由

创建请求方法为post，url为/buy的路由

const express = require('express');

var app = express();

app.listen('3000', function () {

  console.log('服务器创建成功');

});

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

  console.log('接收到了get，url为/reg');

});

app.get('/home', function (req, res) {

  console.log('接收到了get，url为/home');

});

app.post('/buy', function (req, res) {

  console.log('接收到了get，url为/bug');

});

例：

//创建服务器

const express = require('express');

var app = express();

app.listen(3000);

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

  //console.log('这是login');

  //请求的方法，请求的url

  console.log('请求的方法：' + req.method);

  console.log('请求的URL：' + req.url);

  //console.log('请求的头部信息'+JSON.stringify(req.headers));

  console.log('请求中的查询字符串：' + JSON.stringify(req.query));

});

//JSON.stringify  将json对象转为字符串

//JSON.parse  将json字符串解析为对象

app.get('/list', function (req, res) {

  //响应一行文本

  //不允许出现多个send

  //  res.send('这是商品的列表页面');//发送一行文本

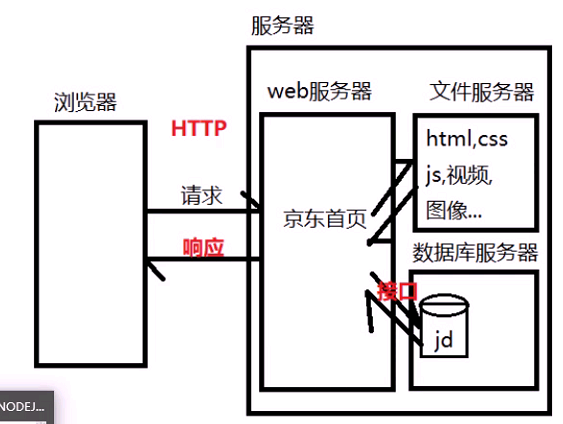
  //响应文件

  //当前文件的完整目录 \_\_dirname

  //console.log(\_\_dirname);

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

});



练习：使用express创建web服务器，创建以下路由

get /index 发送文本 'this is index page'

get /shopping 发送文件

post /login 发送文本 '登录成功'

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

});

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

  res.send('this is index page');

});

app.get('/shopping', function (req, res) {

  res.sendFile(\_\_dirname + '/01\_homework.html')

});

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

  res.send('登陆成功');

});

###### 响应的对象res

res.send 响应文本；注意：只能用有一个send

res.sendFie 响应文件；注意：使用绝对路径\_\_dirname

res.send() 响应文本 里边的参数若为数字，表示状态码；若状态码不存在，则报错

res.redirect(url) 响应的重定向 url可以跳转到其他的地址

##### 获取浏览器端请求的数据

###### POST请求方法的数据

req.on('data',function(buf){

//获取的数据是bufer形式

buf.toString()

//转换后的数据是查询字符串，需要借助查询字符串模块，解析为对象

});

###### GET请求方法的数据

以查询字符串的形式发送服务器

req.query //结果自动将查询字符串转为对象

###### 对比POST和GET方法

POST请求安全程度相对比较高，但速度较慢，常用于注册、登录等；

GET请求安全程度相对较低，但速度较快，常用于搜索、检索等。

例：//建立02\_post.js

//使用express创建web服务器

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

})

//创建路由

//请求方法:get;url:/login;

//响应内容:login.html

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

  res.sendFile(\_\_dirname + '/02\_login.html');

});

//因为浏览器发送了请求方法为post，

//请求URL为/mylogin

app.post('/mylogin', function (req, res) {

  res.send('登陆成功');

});

//建立02\_login.html

<h2> 登录页面 </h2> <!-- 在form表单中， 设置请求的方法和请求的url method： 请求的方法 action： 请求的url -->

<form method="post" action="/mylogin">

用户： <input type="text" name="uname">

<br>

<!--uname可以自定义命名-->

密码： <input type="text" name="upwd">

<br>

<input type="submit">

</form>

练习：

//使用express创建web服务器

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

})

//创建路由

//请求方法:get;url:/login;

//响应内容:login.html

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

  res.sendFile(\_\_dirname + '/02\_login.html');

});

//因为浏览器发送了请求方法为post，

//请求URL为/mylogin

app.post('/mylogin', function (req, res) {

  res.send('登陆成功');

});

//路由：请求的方法：get；请求的url：/reg

//请求的内容：/reg.html

//在reg.html中发送post的请求（用户、密码、邮箱、手机），请求的URL：/nyreg

//要求点击按钮，响应的内容是注册成功

//在02\_post.js中

const querystring = require('querystring');

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

});

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

  res.sendFile(\_\_dirname + '/02\_1\_reg.html');

});

app.post('/myreg', function (req, res) {

  //获取浏览器请求的数据

  //事件:当有数据从浏览器传输到服务器的时候会触发事件

  req.on('data', function (buf) {

    var body = buf.toString();

    //转化后的数据是查询字符串形式

    var obj = querystring.parse(body);

    console.log(obj);

  });

  res.send('注册成功');

});

//建立02\_1\_reg.html

<h2>注册页面</h2>

<form method="post" action="/myreg">

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：<input type="text" name="phone"><br>

<input type="submit">

</form>

例：

//在02\_post.js中，增加

//路由：请求方法get，请求url：/myreg2

app.get('/myreg2', function (req, res) {

  //获取请求的查询字符串，并自动解析为对象

  console.log(req.query);

  res.send('使用get注册成功');

}); //数据会记录在缓存中，可在安全要求不高的地方使用

##### 路由传递参数

###### 配置路由

app.get('/detail/:lid',function(){

  //lid是用于接收的参数名称

  req.params  //获取传递参数，是对象形式的数据

  });

###### 请求URL

http://127.0.0.1/detail/30

30表示传递的参数

例：

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

});

//路由

//接受浏览器请求的参数

app.get('/detail/:lid/:pname', function (req, res) {

  //把所有要传递的参数都放入到req.params下

  console.log(req.params);

  res.send('这是详情');

});

练习：创建路由，请求的方法：get，请求的URL：/book,传递图书的价格和图书的名称，把两个值响应在浏览器中。

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

});

app.get('/book/:price/:bname', function (req, res) {

  res.send(`

    图书的价格为：${req.params.price}

    <br>

    图书的名称为：${req.params.bname}

  `);

});

##### 路由器

把同一模块下的路由归纳到一起进行管理。

###### 使用步骤

1. 创建路由器模块（js文件）
2. 引入express模块
3. 调用express下的路由器功能，创建路由器
4. 在路由器下添加所有的路由
5. 导出路由器
6. 在服务器模块中，引入路由器模块
7. 把路由器挂载到具体某个URL

**app.use('/product',productRouter)**

访问形式 /product/delete

例:

//创建文件app.js作为服务器模块

//创建服务器

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

});

//引入创建好的路由器

const productRouter = require('./routes/product.js');

//路由器只能用于web服务器

//把导入的路由放到(挂载)/product下

//最终的访问形式 /product/list product/delete

app.use('/product', productRouter);

//创建routes文件夹作为所有路由器的集合

//创建product.js文件作为产品路由器模块

//商品路由器——所有商品模块下的路由

//属于express下的一个功能

const express = require('express');

//使用路由器功能——默认是一个空路由

var router = express.Router();

//在路由器中添加路由

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

  res.send('这是商品列表');

});

router.get('/delete', function (req, res) {

  res.send('这是删除商品');

});

//创建好的路由器公开出去——让服务器调用

//导出

module.exports = router;

//在node中导入app.js文件,启动服务器;在在浏览器中输入127.0.0.1:3000/product/list和127.0.0.1:3000/product/delete可分别显示

练习: 创建用户模块路由器(user.js)

//在app.js文件输入以下内容

const userRouter = require('./routes/user.js');

//把导入的用户路由器挂在到user下

//  /user/list  /user/update

app.use('/user', userRouter);

//在routes文件夹中建立user.js文件作为用户路由器

const express = require('express');

//创建了一个空路由器

var router = express.Router();

//往空路由器中添加路由

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

  res.send('这是用户列表页面');

});

router.get('/update', function (req, res) {

  res.send('这是用户修改的页面');

});

//导出路由器router

module.exports = router;

//在node中导入app.js文件,启动服务器;在在浏览器中输入127.0.0.1:3000/user/list和127.0.0.1:3000/user/update可分别显示

##### 引入第三方模块

require('express');

首先在当前目录下查找node\_module中是否有express。

若没有就会到上一级目录查找，直到顶级目录找不到才会报错。

##### express中间件

###### 中间件概述

中间件的作用是为主要的业务逻辑所服务，就是express下的函数。

必须要配合其他的中间件或者路由共同实现一个业务逻辑。

app.use(fn) 应用于(拦截)所有的路由

app.use('/list',fn) 应用于(拦截)特定的路由

例:

const express = require('express');

var app = express();

app.listen(3000);

//创建了一个中间件函数

function fn(req, res, next) {

  console.log('执行了验证');

  //res.send('中间件拦截了请求');//send只能出现一次,此次只是测试

  //继续调用下一个中间件

  next();

}

function fn1(req, res, next) {

  console.log('查看了权限');

  next();

}

//在中间件中使用函数——把函数名称传递进去

app.use(fn);

//创建中间件——查看权限

app.use(fn1);

//路由

app.get('/reg', function (req, res, next) {

  res.send('注册成功');

  next();

});

app.get('/login', function (req, res, next) {

  res.send('登录成功');

  next();

});

//后置中间件

app.use(function (req, res, next) {

  console.log('打印的日志')

  next();

});

例:

const express = require('express');

var app = express();

app.listen(3000);

//应用于指定的路由

//给/list添加中间件

app.use('/list', function (req, res, next) {

  console.log('列表的前置中间件');

  next();

});

app.get('/detail', function (req, res) {

  res.send('这是商品详情');

});

练习,给/detail分别添加前置和后置中间件

//在上述例子的/detail处添加

app.use('/detail', function (req, res, next) {

  console.log('详情前置中间件');

  next();

});

app.get('/detail', function (req, res, next) {

  res.send('这是商品详情');

  next();

});

app.use('/detail', function (req, res) {

  console.log('详情后置中间件');

});

练习:使用Web服务器创建网页计数器。

a.使用中间件初始化num为0

b.在路由中,nun加1,把num的值响应到浏览器

提示:发送式,数字转字符串

const express = require('express');

var app = express();

app.listen(3000, function () {

  console.log('ok');

});

var num = 0;

app.use('/num', function (req, res, next) {

  num++;

  console.log(num);

  next();

});

app.get('/num', function (req, res) {

  res.send('计数:' + num.toString());

});

###### 中间件分类

**应用级的中间件** 拦截路由 app.use（'/list',fn）;

**路由级的中间件** 用于调用路由器;

**内置的中间件** 托管静态文件到某一个目录下;

app.use(express.static('要托管的目录'));

可以托管多个目录,按照托管的顺序来查找文件,如果第一没有,会继续往后查找

###### **第三方的中间件**

先下载安装模块

body-parser 中间件使用,作用:将post请求的数据解析为对象

app.use(bodyParser.urlencoded({

extended:false //使用querystring解析数据

}));

获取数据: req.body

###### 错误级的中间件

例://内置中间件

const express = require('express');

var app = express();

app.listen(3000);

//把所有的静态文件(html、css、浏览器端的js、图像)托管到public目录下

//如果浏览器请求某一个静态资源,不需要再使用路由,而是自动到public下查找

app.use(express.static('public')); //express.static,使用express包下的static表托管静态文件

//浏览器端输入http:       //127.0.0.1:3000/login.html,可直接登录网页

练习: 把静态文件托管到files目录下, 查看如果和public下存在相同名称的文件, 最终显示哪一个

//接上个程序

app.use(express.static('files'));

//显示哪一个与程序的书写顺序有关,当第一个文件中没有时,才会访问第二个文件

练习:创建Web服务器,托管静态文件到public目录下;在login.html中实现用户登录(用户名,密码,提交按钮)

//创建03\_bodyParser.js文件

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

const express = require('express');

var app = express();

app.listen(3000);

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

//使用中间件body-parser,解析post请求的数据

//urlencoded 将post请求数据解析为对象

app.use(bodyParser.urlencoded({

  extended: false

  //false 使用querystring模块解析为对象

  //true 使用第三方模块qs解析为对象

}));

//路由

app.post('/mylogin', function (req, res, next) {

  //获取get请求的数据

  //req.query

  //获取post请求的数据——数据

  /\*

  req.on('data',function(buf){

  console.log(buf);

  console.log(buf.toString());

  //使用查询字符串模块将数据解析为对象

  });

  \*/

  //若是使用了第三方的中间件,获取数据

  console.log(req.body);

  res.send('登陆成功');

});

//在login.html中输入以下代码

<form method="post" action="/mylogin">

用户名:<input type="text" name="uname"><br>

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

<input type="submit">

</form>

###### npm使用

npm init -y //查看安装npm包的日志

npm install 包名 //安装npm包

例:npm install express

用npm包的时候现装现用,正常的不得拷贝不得移动位置

拷贝package.json文件

在node中转入package.json文件所在文件夹,输入npm install,可自动安装

###### req对象

req.method 请求的方法

req.url 请求的URL

req.query 获取GET请求的查询字符串,并解析为对象

req.params 获取路由中传递参数,解析为对象

req.body 获取POST请求的数据,并解析为对象

### MySQL模块

增 INSERT INTO emp VALUES(NULL...)

删 DELETE FROM emp WHERE eid=3;

改 UPDATE emp SET sex=1,salary=800 WHERE eid=10;

查 SELECT \* FROM emp;

#### 普通连接

var connection=mysql.createConnect({ }) 创建连接

connection.connect() 执行连接

connection.query(sql.callback) 执行SQL语句,通过回调函数来获取SQL语句的执行结果。

例://使用MySQL模块连接数据库

//引入MySQL模块

const mysql = require('mysql');

//mysql -h127.0.0.1 -p3306 -uroot -p

//使用数据库 use tedu;

//连接:主机,端口,用户名,密码

var connection = mysql.createConnection({

  host: '127.0.0.1',

  port: '3306',

  user: 'root',

  password: '',

  database: 'tedu', //连接成功后所使用的数据库

});

//执行连接

connection.connect();

//执行SQL语句

connection.query('select\*from emp', function (err, result) {

  //err 如果查询失败

  //result  查询的结果

  if (err) throw err;

  console.log(result);

});

#### 使用连接池

var pool=mysql.createPool() 创建连接池,使用connectionLimit设置连接池的数量。

pool.query(sql,callback) 执行SQL语句,通过回调函数来获取SQL语句的执行结果。

SQL语句中可以使用占位符（？），可以防止SQL注入。

例://使用连接池——在创建MySQL连接的时候，设置连接的个数

const mysql = require('mysql');

//直接连接

var pool = mysql.createPool({

  host: '127.0.0.1',

  port: '3306',

  user: 'root',

  password: '',

  database: 'tedu',

  connectionLimit: 15 //设置连接池的数量

});

//执行SQL语句

pool.query('select \* from dept', function (err, result) {

  if (err) throw err;

  console.log(result);

});

练习:往部门表中插入2条记录,最后打印插入的结果

const mysql = require('mysql');

//直接连接

var pool = mysql.createPool({

  host: '127.0.0.1',

  port: '3306',

  user: 'root',

  password: '',

  database: 'tedu',

  connectionLimit: 15 //设置连接池的数量

});

pool.query('insert into dept values (50,"人事部"),(60,"行政部") ', function (err, result) {

  if (err) throw err;

  console.log(result);

});

var obj = {

  did: 70,

  dname: '后勤部'

}

pool.query('insert into dept SET ? ', obj, function (err, result) {

  if (err) throw err;

  console.log(result);

});

练习:使用对象的形式往员工表插入1条记录

var obj = {

  eid: null,

  birthday: '1995-3-15',

  salary: 7500,

  ename: 'zhangsan',

  sex: 1,

  deptId: 20

}

pool.query('insert into emp SET ? ', obj, function (err, result) {

  if (err) throw err;

  console.log(result);

});

练习:查询工资大于5000,性别为男的员工

var s = 5000;

var n = 1;

pool.query(`select \* from emp where salary>? and sex=?`, [s, n], function (err, result) { //?表示占位符,可以防攻击

  if (err) throw err;

  console.log(result);

});

练习:删除员工表中编号为5的记录。更改员工编号为6的,工资8888,生日为2000-1-1

//删除编号为5的记录

pool.query('DELETE FROM emp WHERE eid=?', 5, function (err, result) {

  if (err) throw err;

  console.log(result);

});

//更改编号为6的记录

pool.query('UPDATE emp SET salary=?,birthday=? WHERE eid=?', [8888, '2000-1-1', 6], function (err, result) {

  if (err) throw err;

  console.log(result);

});

练习:

//1.创建Web服务器

const mysql = require('mysql');

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

const express = require('express');

var app = express();

app.listen(3000);

//连接mysql数据库

var pool = mysql.createPool({

  host: '127.0.0.1',

  port: 3306,

  user: 'root',

  password: '',

  database: 'tedu',

  connectionLimit: 15

});

//2.托管静态文件到public下

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

//使用body-parser中间件,解析post请求的数据

app.use(bodyParser.urlencoded({

  extended: false

}));

//3.在public下,创建add.html

//4.路由:请求的方法post 请求的URL:/dept

//获取浏览器post请求的数据,格式:对象  body-parser

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

  //console.log(req.body);          //测试代码

  //把post请求的数据,插入到MySQL数据库中

  //执行SQL语句,如果插入成功,响应给浏览器"部门添加成功"

  pool.query('insert into dept set ?', req.body, function (err, result) {

    if (err) throw err;

    //console.log(result);          //会出affectedRows的值

    //如果结构中affectedRows大于0,说明插入成功,否则插入失败

    if (result.affectedRows > 0) {

      res.send('部门添加成功');

    } else {

      res.send('部门添加失败');

    }

  });

});

//add.html中的程序

<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>

## 连接错误

1. connect ECONNREFUSED 127.0.0.1:3306 没有启动MySQL服务器
2. You have an error in your SQL syntax SQL中存在语法错误
3. Access denied for user ''@' localhost' 连接错误,用户名或密码错误
4. Duplicate entry '50' for key 'PRIMARY' 插入的记录出现了重复内容