# 一 http模块

## 1 简单web程序

let http = require('http');  
let server = http.createServer(function (req,res) {  
 res.writeHead(200,{  
 'Content-Type':'text/plain;charset=UTF8'  
 });  
 res.end('hello world');  
}).listen(8000);

http.createServer()方法返回的是http模块封装的一个基于事件的http服务器。http.request是其封装的一个http客户端工具，可以用来向http服务器发送请求。req，res分别是http.IncomingMessage和http.ServerResponse的实例。

http.Server的主要事件有：

request：客户端发起请求时，被处罚，提供req，res参数

connection：TCP建立连接时候处罚，提供一个scoket参数，是net.Socket的实例。

close：服务器关闭时，触发。

http.createServer()方法其实就是添加了一个Reuqest事件监听，如下所示：

|  |
| --- |
| **var http** = *require*(**'http'**); **var** server = **http**.*createServer*(); server.on(**'error'**,**function** (err) {  ***console***.log(err); }); server.on(**'request'**,**function** (req,res) {  ***console***.log(**'有用户请求了'**);  ***console***.log(req); }); server.**listen**(8081,**'localhost'**); |

如果想修改程序，必须中断当前运行的服务器，重新node一次，刷新，才行。**ctrl+c，就可以打断挂起的服务器程序。**

Node.js没有根目录的概念，因为它根本没有任何的web容器！

让node.js提供一个静态服务，都非常难！也就是说，node.js中，如果看见一个网址是

|  |
| --- |
| 1. 127.0.0.1:3000/fang |

别再去想，一定有一个文件夹，叫做fang了。可能/fang的物理文件，是同目录的test.html

URL和真实物理文件，是没有关系的。URL是通过了Node的顶层路由设计，呈递某一个静态文件的。

http.IncomingMessage是http请求信息，提供了3个事件：

data：当请求数据到来时触发；

end：当请求体数据传输完毕时候触发；

close：当用户请求结束时候触发。

http.IncomingMessage提供的属性有：

method：请求方式

headers：请求头

url：请求路径

httpVersion：http版本

如：req.on(‘data’,function(chunk){

let data = ‘’;

data = data + chunk;

});

http.ServerResponse是返回客户端的信息，主要方法有：

res.writeHead(statusCode,[headers]; 向请求的客户端发送响应头

res.write(data,[encoding]); 向请求发送内容

res.end([data],[encoding); 结束请求

## 2 http与https

二者创建服务器的接口都是一样的，却别是https多了一个参数opts，参数中保存了key和cert信息。

const https = require('https');  
const fs = require('fs');  
let options = {  
 key:fs.readFileSync('keys/agent2-key.pem'),  
 cert:fs.readFileSync('keyas/agent2-key.pem')  
};  
https.createServer(options,function (req,res) {  
 res.writeHead(200);  
 res.end('hi');  
}).listen(80);

# 二 url模块

req.url属性，表示用户的请求URL地址。所有的路由设计，都是通过req.url来实现的。识别URL，用到两个新模块，第一个就是url模块，第二个就是querystring模块

|  |
| --- |
| **var url** = *require*(**'url'**); **var** urlStr = **url**.*parse*(**'http://www.baidu.com/a/index.html?name=2'**); ***console***.log(urlStr); |

输出如下所示的结果：

Url {

protocol: 'http:',

slashes: true,

auth: null,

host: 'www',

port: null,

hostname: 'www',

hash: null,

search: '?name=2',

query: 'name=2',

pathname: '/baidu/com/a/index.html/',

path: '/baidu/com/a/index.html/?name=2',

href: 'http://www/baidu/com/a/index.html/?name=2'

}

如上所示，如果直接输出：

console.log(urlStr.query);

这是一个字符串：name=2，如果需要直接获取一个键值对结果，那么需要额外传入true参数：

let url = require('url');  
let urlObj= url.parse('http://www/baidu/com/a/index.html/?name=2',true);  
console.log(urlObj.query);

输出的结果为：

{ name: '2' }

这时候就方便实际项目开发的使用了。

url.formate(urlObj) 接收一个url对象（注意是对象），返回完整的url地址

url.resolve(from,to) 接收一个base url对象和一个href url对象，像浏览器那样解析，返回一个完整的地址。

const url = require('url');  
let urlObj = {  
 'host':'www.baidu.com',  
 'port':80,  
 'protocol':'https',  
 'search':'?q=test',  
 'query':'?q=test',  
 'path':'/'  
};  
console.log(url.format(urlObj));  
*//https://www.baidu.com?q=test*console.log(url.resolve('http://www.a.com','/image'));  
*//http://www.a.com/image*

# 三 querystring 模块

主要方法：

querystring.parse(); 将查询字符串反序列化为一个对象，类似JSON.parse()

querystring.stringify(); 将一个对象序列化为一个字符串对象，类似JSON.stringify()

const querstring = require('querystring');  
  
let str1 = 'username=lisi&password=123';  
let obj1 = querstring.parse(str1); *//转换为了对象*let obj2 = {  
 username:'zs',  
 password:'456'  
};  
let str2 = querstring.stringify(obj2); *//重新转换为字符串*

用户提交的网址一般包含一定的数据，比如 username=lisi 这样的参数，我们可以通过url.query获取到这些字符串数据，Node提供了querystring对象来对这个字符串数据进行实例化处理：

var querystring = require(‘querystring’);

urlStr = url.parse(req.url);

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

# 四 简单的路由

|  |
| --- |
| **var http** = *require*(**'http'**); **var url** = *require*(**'url'**); **var** fs = *require*(**'fs'**);  **var** server = **http**.*createServer*(); server.**on**(**'request'**,**function** (req,res) {  ***console***.log(**'有用户请求了'**);  **var** urlStr = **url**.*parse*(req.**url**);  **switch** (urlStr.**pathname**) {  **case '/'**:  res.writeHead(200,{  **'content-type'**:**'text/html;charset=utf-8'** });  res.end(**'<h1>这是首页</h1>'**);  **break**; *//首页* **case '/user'**:  *sendData*(\_\_dirname + **'/html/'** + **'1.html'**,req,res);  **break**; *//用户首页* **default**:  res.writeHead(404,{  **'content-type'**:**'text/html;charset=utf-8'** });  res.end(**'<h1>页面不存在</h1>'**);  **break**;  } });  **function** *sendData*(filePath,req,res) {  fs.readFile(filePath,**function** (err,data) {  **if**(err){  res.writeHead(404,{  **'content-type'**:**'text/html;charset=utf-8'** });  res.end(**'<h1>页面不存在</h1>'**);  }**else** {  res.writeHead(200,{  **'content-type'**:**'text/html;charset=utf-8'** });  ***console***.log(String(data));  res.end(data);  }  }); }  server.**listen**(8081,**'localhost'**); |

# 五 path模块

const path = require('path');  
let myurl = path.join(\_\_dirname,'username','123');  
console.log(myurl); *//E:\ProjectWeb\Test\username\123*

path.extname(); 返回路径参数的扩展名，如果没有扩展民返回空字符串；

path.parse(); 将路径解析为对象

path.format(); 接收一个路径对象，返回一个完整的路径地址

paths.normalize(); 将传入的路径字符串中多余 / 等去除，规范化路径