Node基础编程

1. 基于Chrome V8引擎，可处理高并发
2. 单线程
3. 支持js语法开发后端代码
4. 非阻塞的IO：基于事件驱动

# Common规范

定义：一个文件代表一个模块，导出，引用

node.js模块：内置模块，第三方、自定义模块

内置模块载入：https://nodejs.org/dist/latest-v12.x/docs/api/

const os = require('os');

console.log(os.hostname())

第三方模块：例request模块发送http请求

需要先安装(npm)：npm install –save request

发起get请求



发送post请求



form格式的post请求：



暴露接口：自定义模块

创建demo.js文件

写个方法:demo1

引出：export demo1 = demo1 or module.exports = hello

引入：const demo = require(./demo.js)

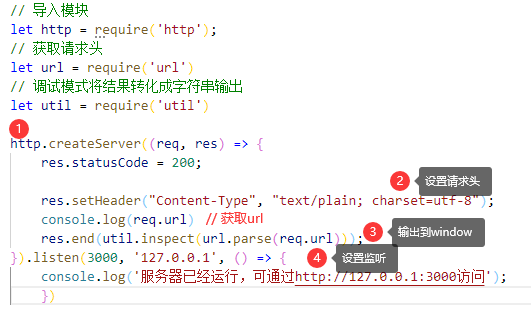
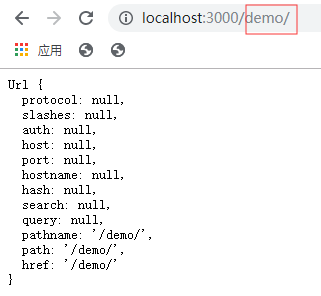
console.log(demo.demo1())

**exports 和 module.exports 的使用**

如果要对外暴露属性或方法，就用 ***exports*** 就行，要暴露对象(类似class，包含了很多属性和方法)，就用 ***module.exports***。

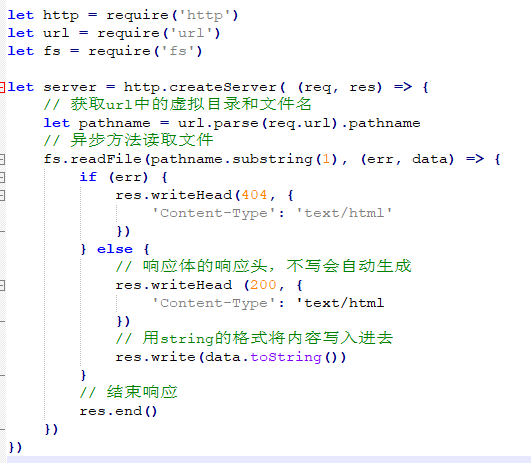
引用：

# 创建http server



# 服务端：通过node加载静态资源

首先需要获取url，通过url.parse模块对url进行解析，并获取pathname



里面用到了http模块、url模块和fs模块搭建一套简单的服务器

只能是静态的文件

# 客户端：可以调用第三方接口

爬虫

# 自定义模块(module.export和export)

module.export是真正的接口，export是辅助函数；最终返回给调用的是Module.exports而不是exports

若同时出现module.export和export，那么给export设置的属性和方法将不起作用

module.exports = 'ROCK IT!'; // rocker.js

exports.name = function() {

console.log('My name is Lemmy Kilmister');

};

调用：

使用require，使用其他框架也是用它

let my = require(‘./rocker.js’);

my.name() // 会报错

1. exports是module.exports的一个引用
2. require引用模块后，返回给调用者的是module.exports而不是exports
3. exports.xxx，相当于在导出对象上挂属性，该属性对调用模块直接可见
4. exports =相当于给exports对象重新赋值，调用模块不能访问exports对象及其属性
5. 如果此模块是一个类，就应该直接赋值module.exports，这样调用者就是一个类构造器，可以直接new实例

Module.exports的多种用法：

作为一个类：

module.export = function(name, age) { // demo.js

this.name = name

this.age = age

this.sayHello = function() {

console.log(this.name + 'is' + this.age + 'years old')

}

}

应用：let Person = require(‘./demo.js’)

let a = new Person(‘jane’, 10)

a.sayHello()

作为一个数组：

module.export = [‘a’, ‘b’, ‘c’]

引用：let arr = require(‘./demo.js’)

arr[1]

# Node.js常用模块介绍

## url

### url.parse(urlStr, [parseQueryString], [slashesDenoteHost])

url 模块中的 parse 函数可以用于解析url中的参数。

参数详解：

**urlStr** - 需要接收的url字符串。

**parseQueryString** - 为true时将使用查询模块分析查询字符串，默认为false。

主要针对query中的内容分析

**DenoteHost**

         -默认为false，//foo/bar 形式的字符串将被解释成 { pathname: ‘//foo/bar' }

         -如果设置成true，//foo/bar 形式的字符串将被解释成  { host: ‘foo', pathname: ‘/bar' }

例：解析博客地址https://www.cnblogs.com/1314-/p/6517740.html

const url = require('url')

const myURL =  url.parse('https://www.baidu.com/s?tn=80035161\_2\_dg&wd=%E6%98%9F%E7%A9%BA')

console.log(myURL)

**parseQueryString** 为false时：

query: 'tn=80035161\_2\_dg&wd=%E6%98%9F%E7%A9%BA',



### url.format()

### url.resolve()

## querystring（查询字符串）

**语法：querystring.stringify(obj[, sep[, eq[, options]]])**

提供用于解析和格式化 URL 查询字符串的实用工具，一般是对**http请求所带的数据**进行解析

调用：const querystring = require('querystring');

方法：

### querystring.parse(str,separator,eq,options)

parse这个方法是将一个字符串反序列化为一个**对象**

separator：默认分隔符为&

eq: 默认分割=，用于划分键值对

options： 默认该参数是对象



**[Object: null prototype] { name: 'whitemu', sex: [ 'man', 'women' ] }**

### querystring.stringify(obj,separator,eq,options)

将一个对象序列化成一个**字符串**，与querystring.parse相对。

var b = querystring.stringify({ name: 'whitemu', sex: [ 'man', 'women' ]})

console.log(b)

**name=whitemu&sex=man&sex=women**

### querystring.escape(str)

escape可使传入的字符串进行**编码**

var c = querystring.escape('name=唐高峰')

console.log(c)

name%3D%E5%94%90%E9%AB%98%E5%B3%B0

### querystring.unescape(str)

unescape方法可将**含有%的字符串**进行**解码**

demoUrl = 'https://www.baidu.com:8080/s?tn=80035161\_2\_dg&wd=%E6%98%9F%E7%A9%BA';

var d = querystring.unescape(demoUrl)

console.log(d)

https://www.baidu.com:8080/s?tn=80035161\_2\_dg&wd=星空

## HTTP / HTTPS模块

调用：const http = require('http')



createServer方法本质上是为http.Server对象添加了一个request事件监听

var server=new http.Server();

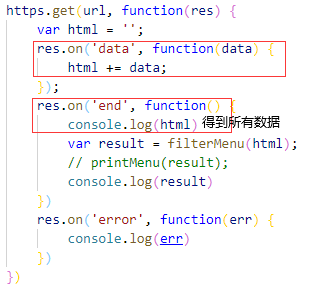
server.on("request", (req,res) => {}).listen(3000);

req是请求消息，res是响应消息

req是http发送请求后返回的一个对象，req.query里面有你传递的数据

res.on等同于监听事件addListener

res.end; res 是个writable的流，当writable流接收到 null 的时候会触发 end 事件这里的确切含义就是给客户端的东西写完了，可以发送给客户端了



https.get(url, () => {})可用于爬虫：

cheerio: node版jQuery

使用：var $ = cheerio.load('html' // html可以是用https.get爬取到的整个页面

之后就可以进行$(‘’).find().text()等操作

### http.request(GET & POST)

简化的CURL模块，建立各种HTTP请求，支持授权验证，其实功能还是非常多的，基本满足绝大部分需求

var options = {

hostname: 'spark.appublisher.com',

port: 443,

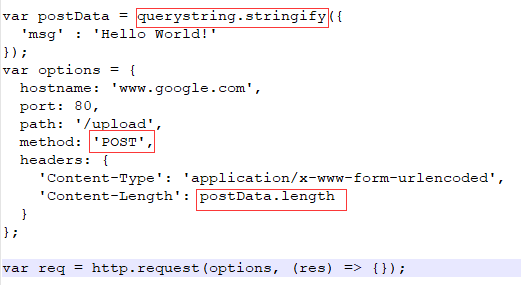
method: 'GET',

path: '/course/get\_open\_class'

}

var request = https.request(options, (response) => {})

post：



我们可以利用这个http的request来提交一下评论，我们可以获取网站的一些评论接口，通过上面options，我们可以配置请求的headers信息，进行网站的灌水评论！

## Events

## fs

## Stream

Stream 是一个抽象接口，Node 中有很多对象实现了这个接口。例如，对http 服务器发起请求的request 对象就是一个 Stream，还有stdout（标准输出）。

Stream 有四种流类型：

* **Readable** - 可读操作。
* **Writable** - 可写操作。
* **Duplex** - 可读可写操作.
* **Transform** - 操作被写入数据，然后读出结果。

所有的 Stream 对象都是 EventEmitter 的实例。常用的事件有：

* **data** - 当有数据可读时触发。
* **end** - 没有更多的数据可读时触发。
* **error** - 在接收和写入过程中发生错误时触发。
* **finish** - 所有数据已被写入到底层系统时触发。

## 后端路由

## 请求参数

## 同步异步编码async

## socket

## MongoDB

cnpm install mongodb

var MongoClient = require('mongodb').MongoClient;

var url = "mongodb://localhost:27017/runoob";

MongoClient.connect(url, { useNewUrlParser: true }, function(err, db) {

if (err) throw err;

console.log("数据库已创建!");

db.close();

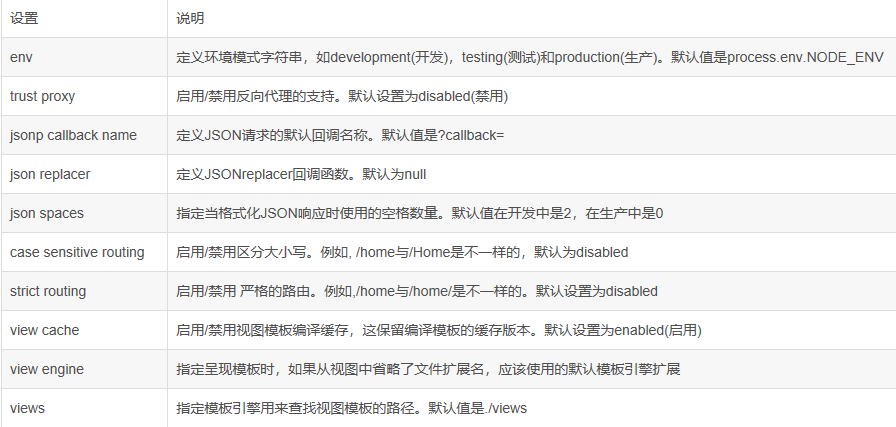
});

## express

Express 是轻量灵活的Nodejs Web应用框架，它可以快速地搭建网站。Express框架建立在Nodejs内置的Http模块上，并对Http模块再包装，从而实际Web请求处理的 功能。它支持多种前端模板，如Jade, EJS等。它是T.J大神的作品，不过已经交由其它团队维护了。

* 1. 安装：npm install express
  2. 引入和创建实例：let express = require(‘express’);

let app = express()



以上是创建实例后可以进行的一些设置 ，例：

app.enable('trust proxy');

app.disable('strict routing');

app.set('view engine','jade');

* 1. 开启监听

let express = require(‘express’);

let app = express()

app.listen(8080)

注：express()返回的值实际上是一个回调函数，它映射了传递到http.createServer()和https.createServer()方法的回调函数



* 1. 配置路由

为跟目录时

可以配置多个