[问题 1](#_Toc16937)

[1. 堆内存不够的问题 1](#_Toc30247)

[问题： 2](#_Toc6346)

[基本 2](#_Toc5818)

[正则 2](#_Toc17996)

[性能优化篇 2](#_Toc27296)

[1. Lodash 2](#_Toc5880)

[2. Pg-promise 3](#_Toc28329)

[3. Bluebird 3](#_Toc5980)

[4. request 3](#_Toc31776)

[Request.jar 3](#_Toc9897)

[http认证 5](#_Toc31811)

[OAuth登陆 5](#_Toc7369)

[5. Follow-redirects 5](#_Toc13413)

[6. https/http 5](#_Toc10867)

[Follow-redirects 6](#_Toc22612)

[Sequelize 6](#_Toc2690)

[mocha 6](#_Toc15542)

[Pm2 6](#_Toc5974)

[Cluster mode 6](#_Toc4467)

[Reload 6](#_Toc11285)

[Shutdown 7](#_Toc3187)

[Statelessify application 7](#_Toc14586)

[ESLint 7](#_Toc28465)

[Zone 7](#_Toc27415)

[DNode 7](#_Toc26096)

[Net 7](#_Toc2872)

[TLS 8](#_Toc30103)

[Events 8](#_Toc5318)

[Linebyline 8](#_Toc5507)

[分布式爬虫架构 9](#_Toc19676)

[request模块爬虫数据实例 10](#_Toc15568)

[参考文件： 12](#_Toc8941)

[Hotel-api程序问题 12](#_Toc9948)

**alter table** place.stations **add column** isAbroad boolean;

# 问题

## 1. 堆内存不够的问题

运行node程序，出现FATAL ERROR: CALL\_AND\_RETRY\_LAST Allocation failed - JavaScript heap out of memory

添加命令参数

node --max\_old\_space\_size=4096 yourFile.js

Reference: <http://jayconrod.com/posts/55/a-tour-of-v8-garbage-collection>

如何在字符串中动态地添加某变量。

Var name=”jack”;

Var prompt=`My name is ${name}`;

Console.log(prompt); //My name is jack

使用require(“”)来获取某个json文件，此时直接已经解析成变量。

## 2. 数据库和json文件读取

当数据量达到10k以上时，使用文件一次性读取，node性能消耗非常之大。此时建议使用数据库。

例如处理2800k数据量时，无法使用

# 问题：

函数以参数形式传递和使用export的区别

Js中的null和undefined是两种情况，其中if(param) 只能判断当其为null的情况

# 基本

href="javascript:void(0)"

# 正则

对于使用RegExp.$1语法选中捕获组，只能够使用1-9九个捕获组

使用new RegExp();后，使用let result=regex.exec(); //获取捕获组的数组

# 性能优化篇

<https://strongloop.com/strongblog/performance-node-js-v-0-12-whats-new/>

对于net的可写流，添加了corked mode。在corked模式下，data会顺序写入stream中，减少system call和tcp roundtrips

GC strain

在V8堆中使用强引用，对于经常使用的的string，object prototypes，使用强引用。但是overhead随着handles的数量线性发展。

对于multi-context的cleanup工作，开始使用eternal handles

Round robin

轮询调度

# Lodash

函数：defaultsDeep(object, [sources])

对对象进行深度复制，对于sources中有object中不存在的，则复制到object中。

# Pg-promise

Node访问postgresql的模块

# 3. Bluebird

Promise.config()

u1.c1566.g1567.k1438553095041.pz

# 4. request

参考：<https://www.npmjs.com/package/request>

提交请求时，需要注意header中的content-type字段。不同的content-type，其做request请求时提交的请求参数和参数的组织形式不同。

Application/json

## Request.jar

管理是否使用cookie，可以不使用全局cookie，定制自己的cookie

对于post,get, delete, head,等函数，其接受的参数的形式为两种：

(url, options, callback)

或者：

(options,callback)

对于JSONObject中的参数有：

form: 该参数为json对象，不能使用JSON.stringify

formData: 同上

json:

body: 该参数必须是string或者buffer

RequestBodyStream：

headers:

proxy:

qs:

aws:

hawk:

httpSignature:

auth:为json对象。{user:””, pass:””, sendImmediately:””, bearer:””}

multipart:

time:

oauth:

ca:

agentClass:

agentOptions

forever:

requestBodyStream: 已经弃用，需要将request object传给stream.pipe

注意当request收到来自服务器的应答后会触发response事件，response的arguments是http.IncomingMessage的一个实例

Get

Put

Post

表单数据时，content-type支持application/x-www-form-urlencoded 和multipart/form-data、multipart/related

Content-type支持application/x-www-form-urlencoded 时：

request.post('http://service.com/upload', {form:{key:'value'}})

request.post('http://service.com/upload').form({key:'value'})

request.post({url:'http://service.com/upload', form: {key:'value'}}, function(err,httpResponse,body){ //处理响应})

'X-Requested-With': 'XMLHttpRequest',？

对象的toString()和JSON.stringify()区别

'Accept-Encoding':'gzip, deflate, sdch'

注意：

Ajax.Request

Post

Request.post(“url”,{form:formJsonData,},function(err, res, body){});

Request.post({url:””, body:bodyData,},function(err, res, body){});

Request.post({headers:headers, url:””, body:bodyData,},function(err, res, body){});

Body:”mes=heydude”

Form:{mes:”heydude”}

## http认证

<http://www.open-open.com/lib/view/open1435301679966.html>

## OAuth登陆

# 5. Follow-redirects

对于爬取大量网站的信息，由于请求引发的情况的多样性，适合使用该包，提供处理redirect的情况，同时获取数据后，对于response对象提供监听事件，可以动态的处理数据。如：

Http.request(url,function(response){

Response.on(“data”,function(){});

}).on(“error”,function(){});

# 6. https/http

7. Buffer

编码

8. BufferHelper

# Follow-redirects

该模块封装处理了转发

# Sequelize

去掉系统默认的id字段。在定义model时，需要人为指定primaryKey字段，否则系统会自动生成id字段。

Sequelize.define(“user”{

Firstname:{

Type: Sequelize.STRING,

primaryKey:true

}

},{});

对于系统自动生成的createdAt和updatedAt字段，通过define时，提供createdAt:false来限制其自动生成

通过define或者import，获取映射某个表的Model，对此可以进行各种操作。

如

---findAll

---findOne

# mocha

Nodejs中的測試框架

https://cnodejs.org/topic/516526766d38277306c7d277

# Pm2

优点：

Boost applications and handle a ton of traffic

支持的语言：coffee、php、python、shell、rb

## Cluster mode

对于https/tcp/udp server，能够利用cpu进行提升perfermance and reliability

使用方法：

Pm2 start server.js -i 0

0表示将app延伸到all CPUS

-1 表示将app延伸到all-1 个cpus

### Reload

### Shutdown

### Statelessify application

无状态指：无local data 被保存在process中，即无sessions/websocket connections，session-memory 等。

# ESLint

QA工具，避免低级错误和统一代码风格。

# Zone

# DNode

nodejs的rpc解决方法。

创建dnode时，服务器端和客户端都有remote事件，可以根据remote事件进行双向主动沟通交流。使用时可以使用zone(npm)将回掉函数中的对象记录，实现代码的扁平化

# Net

参考：<https://nodejs.org/api/net.html>

<http://stackoverflow.com/questions/20085513/using-pipe-in-node-js-net>

基于tcp链接

异步网络包装

函数

Let server =net.createServer(function(socket){});

Server.listen(callback); //若不指定端口，则自由分配

Io.socket

Net：tcp协议

Websocket

Dnode

http

创建server时，callback函数中有socket对象，存在属性：

remoteAddress，remotePort，localAddress，localPort

Net中的socket绑定了ip和port，服务器的port固定，客户端则随机生成一个port。

问题：

使用net的socket的data监听事件，正常同一次读写，或进行多次。因为data event 可能会被emitted多次，当接收到response的几个chunk时。

以net为主导：

创建server后，返回stream对象，进行stream.pipe(d).pipe(stream);

以dnode为主导

创建server后，返回stream对象，进行d.pipe(stream).pipe(d);

# TLS

基于OpenSSL实现了运输层和secure socket layer协议。

参考：<https://nodejs.org/api/tls.html>

# Events

事件的订阅/触发：注意监听的事件存储与内存中。每个对象有监听事件个数的限制，同一对象的监听事件可以被注册多次，故需要时可以根据emitter.listenerCount(eventName)来获取某个事件是否被注册，若是则不再注册。

Emitter.eventNames()来获取某个对象的全部注册的事件

setTimeout

setInterval

# Linebyline

Reference：<https://www.npmjs.com/package/readline>

原型为nodejs的readline，只是换了名字

为event类型，有监听事件line和close等

# 分布式爬虫架构

欠缺：

---dnode的具体实现

---net的流问题

---zone使用问题

---tcp/http/websocket/socketjs/net/等

# request模块爬虫数据实例

爬取数据实例：

介绍：爬取携程国际火车票（欧铁），相关网址：<http://rails.ctrip.com/international/PassFamilyIndex.aspx>

提交表单后跳转到实际页面，举例如下。

<http://rails.ctrip.com/international/OutiePTPList.aspx?departureDate=2017-3-22&starttime=&adult=1&child=0&youth=0&seniors=0&searchType=0&pageStatus=0&passHolders=0&from=FRPAR&to=ITFLR&arriveDate=>

此时可以查看返回的文件：<http://webresource.ctrip.com/ResTrainOnline/R9/Outie/JS/OutiePtpList.js?2017_3_9_16_40_48.js>。该文件中存在ajax请求。

$.ajax('./Ajax/QueryOutiePTPProd.ashx', {

method: 'POST',

async: true,

context: {

QueryParam: $.stringifyJSON(OutiePTPProcess.packetqueryparam())

},

onsuccess: function (result) {//实现省略}

Form表单提交的数据(即上述ajax请求中的OutiePTPProcess.packetqueryparam()返回的json字段。：

var QueryParam={

StartTime:"06:00",

BackTime:"06:00",

StartDate:"2017-3-23",

BackDate:"2017-03-26",

StartCityCode:"FRPAR",

ArriveCityCode:"ITFLR",

PassengerType:{

AdultCount:"2",

YouthCount:"0",

ChildCount:"0",

OldCount:"0"

},

PassHolders:"0",

LastStartDate:"",

StartCityName:"巴黎",

ArrivalCityName:"佛罗伦萨",

TrvalType:"1",

PageLoadGUID:data

};

使用nodejs的模块request，进行模拟ajax请求。

Var request=require(“request”);

Var headers={

User-Agent:””,

Cookie:””

};

var context={QueryParam: JSON.stringify(QueryParam)};

Request.post(url,{headers: headers, form: context, gzip: true}, function(err, res, body){

console.log(body); //此时的body即是所要的json数据。

});

注意：

提交请求时，提交的表单信息格式要根据其原网站设计传递的参数类型来传递，因为原系统提交的信息为JSON.stringify参数，故模拟传递时也要传stringify格式数据。

爬取数据时，产生输入图中验证码

南航：

正常请求网页，获取json数据，网页地址为http://b2c.csair.com/B2C40/query/jaxb/direct/query.ao

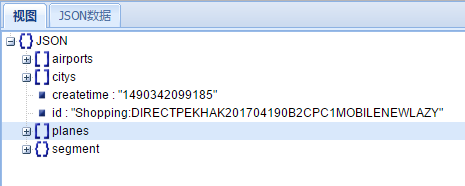
提交的参数为

json:{"depcity":"PEK", "arrcity":"DLC", "flightdate":"20170413", "adultnum":"3", "childnum":"0", "infantnum":"0", "cabinorder":"0", "airline":"1", "flytype":"0", "international":"0", "action":"0", "segtype":"1", "cache":"1", "preUrl":"", "isMember":""}

提交的参数中：flytype并非常量，不同的城市不同，暂未找到该值的来源。

提交的城市码信息中，北京市的城市码为PEK,而不是BJS

正常返回的数据格式



异常情况，其返回需要验证的response，格式如下

{"needverify": "true"}

此时浏览器发起另外的请求http://b2c.csair.com/B2C40/GenerateVerifyCodeServlet.do?r=0.14767951674547386，该请求为图片验证，

参数r的值为Math.random()生成。

## 参考文件：

Google的re-captcha

Google使用用户的上网行为来进行辨别

Github项目：<https://github.com/andelf/fuck12306>

注意图片验证请求，其响应位于客户端中，若错误，则生成新的图片验证，否则再次提交请求。此时的请求相当于新的请求。

问题：

如何

当正常请求时，返回需要验证的数据，如图片验证，此时进行 提交后，

## Hotel-api程序问题

问题描述：

创建了两个worker程序，四大航空公司每个worker请求一次的时间间隔为60s。两个worker的创建时间相近，可能存在两个worker进行同一个网站的请求的时间间隔在5s内，可能造成请求频繁，此时需要将做些改动：

方法1. 两个worker进程请求同一个网站的时间限制间隔为30s

方法2. 创建两个worker进程的时间相隔30s