[问题 2](#_Toc2989)

[1. 堆内存不够的问题 2](#_Toc6593)

[2. 调用栈容量超出 2](#_Toc19132)

[2. 数据库和json文件读取 2](#_Toc4889)

[尚未解决的问题： 2](#_Toc8523)

[基本 3](#_Toc5444)

[正则 3](#_Toc8279)

[Node全局变量 3](#_Toc16919)

[v8 文档 3](#_Toc27679)

[性能优化篇 3](#_Toc30624)

[1. Lodash 4](#_Toc20988)

[2. Pg-promise 4](#_Toc16185)

[Sequelize 4](#_Toc27075)

[3. Bluebird 4](#_Toc7636)

[4. request 5](#_Toc1856)

[Request.jar 5](#_Toc6161)

[http认证 6](#_Toc9363)

[OAuth登陆 6](#_Toc3997)

[5. Follow-redirects 7](#_Toc29817)

[6. https/http 7](#_Toc3305)

[Follow-redirects 8](#_Toc959)

[Sequelize 8](#_Toc24380)

[基本 8](#_Toc19317)

[查询 8](#_Toc3754)

[mocha 8](#_Toc24991)

[Pm2 9](#_Toc13945)

[Cluster mode 9](#_Toc9324)

[Reload 9](#_Toc21500)

[Shutdown 9](#_Toc23622)

[Statelessify application 9](#_Toc13544)

[ESLint 9](#_Toc29490)

[Zone 9](#_Toc18863)

[DNode 9](#_Toc5209)

[Cluster 9](#_Toc875)

[Net 10](#_Toc23729)

[TLS 10](#_Toc25178)

[Events 10](#_Toc20002)

[Linebyline 11](#_Toc16632)

[Jshint 11](#_Toc151)

[Gulp---任务自动管理工具 11](#_Toc2369)

[Src() 11](#_Toc10464)

[分布式爬虫架构 12](#_Toc11421)

[request模块爬虫数据实例 13](#_Toc2892)

[参考文件： 15](#_Toc15484)

[Hotel-api程序问题 15](#_Toc24673)

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

数组的array

Spread syntax简化数组的concat，slice等操作。

node中的process.getuid()和process.getgid()在windows系统中提示不存在。但在linux中正常运行

# 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>

Const v8 = require(“v8”); //此包可以不引入（不确定）

使用 v8.getHeapStatistics()获取该程序的内存分配情况。

## 调用栈容量超出

Error: Maximum call back stack size exceeded.

出现的情况，使用监听函数，在监听函数内部出发了该监听函数。造成迭代调用。

具体的体现是：在迭代中，新的variable被创建在stack上，在v8尚未回收之前，导致变量迅速增加，导致stack不够用。

方法1：简单的处理办法为node --stack-size=16000 app.js

方法2：

setTimeout

setImmediate

Process.nextTick

## 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

# 尚未解决的问题：

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

编码风格

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。

# Node全局变量

<https://nodejs.org/api/globals.html>

## v8 文档

<https://v8docs.nodesource.com/>

Require 实现cache机制，当时可以手动删除该cache键值，

# 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

项目集成篇

Docker运行

配置文件:

Dockerfile

.gitlab-ci

.ssh

.npmrc:

参考： <https://docs.npmjs.com/private-modules/docker-and-private-modules>

若使用私有库，那么需要在该文件中注明：registry的地址以及相应的token。如：

//npm.l.jingli365.com/:\_authToken="Rkl2OKk7u6+am+KhFB4ZYy6N7J9vuKrumaNZWioq5+8="

.nrmrc

.gitconfig

写入了用户名和密码，同时可以为git command设置alias

[user]

name = yuchanglong

email = changlong.yu@jingli365.com

[gui]

[alias]

ls =status

co=checkout

ci=commit

[gui]

# 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

# Sequelize

参考：<http://docs.sequelizejs.com/manual/tutorial/raw-queries.html>

使用sequelize.query(sql) 函数返回数组类型结果 包含两个参数。其一为result array.数组。另一个为meta-data 定义

[anonymous: [], Result: {}]

Query(sql,{})，通过传递{}参数，指定获取某个类型数据，如若只要结果集数据，添加参数{type: sequelize.QueryTypes.SELECT}

# 3. Bluebird

Promise.config() //bluebird.config({})

可配置项有： cancellation，取值true，false，表明promise的触发abort操作的时候，callback函数是否会被执行。默认关闭。

u1.c1566.g1567.k1438553095041.pz

# express

## Related Middleware

### Body-parser

支持json类型的body、raw类型的body、text类型的body、url-encoded form类型的body。不支持multipart类型的body。

Body-parser会使用工厂产生中间件，当请求的content-type满足type的类型时，中间件会产生req.body的属性。

依托http包，创建http服务器

注意：

---使用connect-timeout：注册请求的响应时间

对于创建express对象，使用中间件注册时，获取的req，res，next可以使用req.clearTimeout()消除创建express对象时设置的超时时间。

# 4. request

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

## 说明

request进行请求时，需要认为替代浏览器做些编码/解码的操作。使用浏览器，浏览器会将请求的url进行一些编码处理。

此时使用request时，需要进行些处理，一般使用querystring或者qs参数。

## 返回结果分析

类型：

----request

----response

----redirect：返回的结果中的statusCode为3xx， 同时data对象中包含headers, uri(下一个请求的地址)。

----auth：报401未授权错误，会视为redirect，返回相同的数据信息。如statusCode, headers,

## Request 源码分析:

直接使用request()函数时，需要在参数中知名method字段，若使用request.post等，无需添加，源码将一系列参数进行铰接。

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

## CORS 跨域问题

使用Cors(Cross-origin resource sharing) 解决跨域问题

# 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

http模块调用方法createServer后，调用listen时，若不显示地提供host，则系统根据是否支持IPv6，则监听IPv6的地址，否则监听IPv4地址。

在服务器环境下，0.0.0.0表示本地机器的所有IPv4地址。如果host存在两个内网地址，那么当server监听地址0.0.0.0，那么对于两个ip地址都可达。

在路由器环境下，0.0.0.0表示默认路由，the route which leads to "the rest of" the internet instead of somewhere on the local network

# 7. Buffer

编码

# 8. BufferHelper

# Follow-redirects

该模块封装处理了转发

# Sequelize

### 基本

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

### 查询

Sequelize提供两种类型的方法进行表查询：

Model：

通过使用sequelize进行define后，可对相应的表进行查询。缺点是此时只能进行一张表的查询。

该方法可以做aggregations。

使用model的一系列方法，其语法相当于mongo等数据库的语法。即条件使用json形式。

其中需要使用的关键字：

$and、$or、$gt、$gte、$lt、$lte、$like、$notLike、$iLike（pg only）

同时findAll()等操作的参数中，可以传入{limit: number, offset: number}，进行数量限制，同时可以跳过一些rows

JONB格式的数据查询三种方式：

Sequelize.query

无需在使用sequelize.define进行表结构的定义。此时直接使用进行sql的直接查询。

# 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

# Cluster

Process.send 只有当子进程运行时，才会存在。对于主进程，process并不存在send函数。

Process.send用于worker给master发送消息

# 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

# Jshint

用于detect errors and potential problems in js code

# Gulp---任务自动管理工具

## Src()

将参数中的文件转换成数据流，参数中可以使用正则表示多个文件。

Desc()

将数据流写入某个文件。如pipe(gulp.dest(“build”)) 将会写入build.js文件中，扩展名省略。

Task()

指定一系列的任务，参数1为任务名，参数2的函数为任务的处理逻辑。若参数2为数组，则为依赖的任务列表。注意依赖任务为异步执行。

# babel-polyfill

垫片，对于低版本的浏览器或者执行环境，当需要调用最新的框架的函数时，可以使用polyfill去提高兼容性

# karma/jasmine

angular框架中用来进行单元测试

**rxjs**

# 9. npm包工作机制

## 9.1 dependencies、devdependencies、peer dependencies

### 9.1.1 peer dependencies

对于只向外提供公用（静态）函数的包，则为non-peer dependencies

# 10. 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

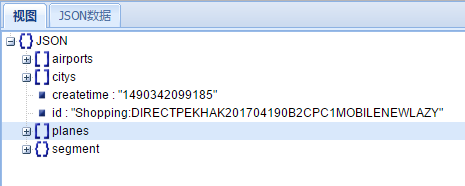
提交的参数为

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