GC

Conservative

保守方式：无法借助compiler，将pointer的内存分配呈线性排列方式。有些data将会被视为pointer，造成内存泄露漏。如Integer对象成功被判断成data，将会保存subgraph of objects.

Compiler hints

静态语言中，编译器会告知每个class中的pointers。Java采取这种方式。

Tagged pointer

对于每个word将会有一位bit用来表示pointer或者data。V8 engine使用的使这种方法。

General collection

对于大部分的程序，GC主要将heap分成两代：old generation & new generation。新生代的space size很小，分配消耗小。当space被用完时，会自动出发GC机制(Scavenge)。 对于变量成功活过两次回收周期的，成功晋级old generation。对于老生代，其GC当一定的memory被存入old space后进行触发。将根据程序的行为和old space的大小决定其threshold

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。

# 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

# 函数以参数形式传递和使用export的区别

# 3. Bluebird

Promise.config()

# 4. request

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

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

对于post,get, delete, head, get, put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

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

request提交请求后该request在收到response数据后将触发response事件，该response的参数http.IncomingMessage的一个实例，IncomingMessage是stream.Readable的子类。具有data，response，end监听事件。

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。

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})

问题：

'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”}

# Follow-redirects

该模块封装处理了转发