# 在nodejs中异步编程的解决方案

1. 回调。
2. Promise包。但是这种方式需要增加一层抽象。
3. async包。这种方法最简单，方便，快捷。

# asycn.each

**定义**

async.each(arr,iterator,[callback])

**参数说明：**

* arr：数组
* iterator(item,callback)
  + 遍历数组arr中的每一个元素
  + 当遍历结束之后必须调用callback，如果没有发生错误，则callback的没有参数或者参数为null
  + 如果你希望使用数组的index，请使用forEachOf方法
* callback(err):
  + 可选参数
  + 调用时间：当遍历结束之后，或者是遍历的时候有错误发生的时候

**作用：**

并行的遍历数组arr中的item

**适用的场景：**

当一个函数被多次调用，调用的参数不同，在不考虑调用返回值的情形下，检查是否有调用出错。

备注

下面黄色底纹的地方，如果不写的话，虽然写没有错误，但是并不是一种好的编程风格。

**var *async***=*require*(**'async'**);  
**var *http***=*require*(**'http'**);  
  
**var *arr***=[**'http://www.baidu.com'**,**'http://www.qq.com'**];  
  
***async***.each(***arr***,**function**(item,done){  
 **var** req=***http***.get(item,**function**(res){  
 **var** result=[];  
 res.on(**'data'**,**function**(chunked){  
 result.push(chunked);  
 });  
 res.on(**'end'**,**function**(){  
 ***console***.log(Buffer.concat(result).toString());  
 done();  
 });  
  
 res.on(**'error'**,**function**(err){  
 done(err);  
 });  
 });  
 req.end();  
  
},**function**(err){  
 ***console***.log(err.**message**);  
});

# async-map

**定义：**  
map(arr,iterator,[callbck])

**参数：**

* arr---数组
* iterator(item,callback)
  + 一个应用到数组中每个元素的方法
  + callback(err,transformed)当有错误发生的时候必须要被调用。transformed代表一个被item所转换的变量
* callback(err,results)
  + **执行时间**：当所有遍历结束完成之后调用，或者是有错误发生的时候调用
  + results:是一个经过arr转变之后的数组
  + 备注：遍历数组中的元素是并发访问的，并不能保证遍历函数的完成是按照顺序的。但是结果集results将和数组arr中元素的次序保持一致。

**async.each问题**

异步遍历的结果集丢失。

*/\*\*  
 \* Created by John on 2015-08-25.  
 \*/***var *async***=*require*(**'async'**);  
**var *http***=*require*(**'http'**);  
  
**var *arr***=[**'http://www.baidu.com'**,**'http://www.qq.com'**];  
  
***async***.**map**(***arr***,**function**(item,done){  
 **var** req=***http***.get(item,**function**(res){  
 **var** result=[];  
 res.on(**'data'**,**function**(chunked){  
 result.push(chunked);  
 });  
  
 res.on(**'end'**,**function**(){  
 done(**null**,Buffer.concat(result).toString().slice(0,1000));  
 });  
  
 res.on(**'error'**,**function**(err){  
 done(err,**null**);  
 });  
 });  
 req.end();  
},**function**(err,results){  
 **if**(err){  
 **return *console***.log(err);  
 }  
 ***console***.log(results);  
});

备注：

在异步编程的时候一定要注意错误的处理，这是一种好的编程习惯。

# async.reduce

**定义：**

reduce(arr,memo,iterator,[callbck])

**参数**

* arr—数组
* memo—累加器的初始状态
* iterator(memo,item,callback)
  + 在累加器中应用于产生下一步的函数
  + callback(err,reduction)第二个参数是reduction的状态，如果传递了一个err，则这个reduction停止，并且主回调会被立即调用。
* callback(err,result)
  + 执行时间：所有遍历结束之后被调用或者出错的时候被调用
  + result是最后一个回调的结果集

**使用场景**

累加的每一步都需要使用到异步。

*/\*\*  
 \* Created by John on 2015-08-25.  
 \*/***var *http***=*require*(**'http'**);  
**var *async***=*require*(**'async'**)  
  
**var *arr***=[2,4];  
***async***.reduce(***arr***,0,**function**(memo,item,done){  
 **var** options=**'http://localhost:8012/?num='**+item;  
 **var** req=***http***.get(options,**function**(res){  
 **var** reduction=**''**;  
 res.on(**'data'**,**function**(chunked){  
 reduction+=chunked;  
 });  
  
 res.on(**'end'**,**function**(){  
 done(**null**,Number(reduction)+memo);  
 });  
  
 res.on(**'error'**,**function**(err){  
 done(err,**null**);  
 });  
  
 });  
 req.end();  
},**function**(err,result){  
 **if**(err){  
 **return *console***.log(err);  
 }  
   
 ***console***.log(result);  
  
});

# async-series

**定义：**

series(task,[callback])

**参数：**

* task
  + 一个包含function的数组或者对象
  + 每个函数 function(callback)中必须传递一个回调函数,在每个函数返回之前必须调用该回调函数callback(err,result).
  + 如果回调函数中没有错误的，则传递null。如果有错误发生，则数组或者对象后续的函数将停止调用，并且立即调用最后一个回调函数，并将错误传给该回到函数。
  + 在ECMAscript中并没有要求对象属性的次序，如果你希望最终的结果是有次序的，这个时候建议使用数组作为task的参数。
* callback(err,results)
  + 可选参数
  + 执行时间：当所有函数返回的时候调用，或者是有错误发生的时候调用
  + results数组或者对象中包含task回调函数中的所有结果集。

**使用场合：**

串行连接

**问题：**

Write a program that will receive two URLs as the first and second command-line arguments.

Using `http.get`, create a GET request to these URLs and pass the response body

to the callback.

Pass in an object of task functions, using the property names `requestOne` and

`requestTwo`, to `async.series`.

`console.log` the results in the callback for series when all the task functions

have completed.

答案：

*/\*\*  
 \* Created by John on 2015-08-26.*

*自己的答案  
 \*/***var *http***=*require*(**'http'**);  
**var *async***=*require*(**'async'**);  
**var *iconv***=*require*(**'iconv-lite'**);  
  
***async***.series([**function**(callback){  
 **var** req= ***http***.get(**'http://www.baidu.com'**,**function**(res){  
 **var** result=[];  
 res.on(**'data'**,**function**(chunked){  
 result.push(chunked);  
 });  
  
 res.on(**'end'**,**function**(){  
 callback(**null**,Buffer.concat(result).toString().slice(0,1000));  
 });  
  
 res.on(**'error'**,**function**(err){  
 callback(err,**null**);  
 });  
 });  
 req.end();  
},**function**(callback){  
 **var** req= ***http***.get(**'http://www.qq.com'**,**function**(res){  
 **var** result=[];  
 res.on(**'data'**,**function**(chunked){  
 result.push(chunked);  
 });  
 res.on(**'end'**,**function**(){  
 callback(**null**,***iconv***.decode(Buffer.concat(result),**'gb2312'**).slice(0,1000));  
 });  
 res.on(**'error'**,**function**(err){  
 callback(err,**null**);  
 });  
 });  
 req.end();  
  
}],**function**(err,results){  
 **if**(err){  
 **return *console***.log(err);  
 }  
 ***console***.log(results);  
});

修正之后的答案

*/\*\*  
 \* Created by John on 2015-08-26.  
 \*/***var** http=*require*(**'http'**);  
**var *async***=*require*(**'async'**);  
**var *iconv***=*require*(**'iconv-lite'**);  
  
***async***.series({requestOne:**function**(callback){  
 *fetchURL*(**'http://www.baidu.com'**,callback,**'utf-8'**);  
},requestTwo:**function**(callback){  
 *fetchURL*(**'http://www.qq.com'**,callback,**'gb2312'**);  
}},**function**(err,results){  
 **if**(err){  
 **return *console***.log(err);  
 }  
 ***console***.log(results);  
});  
  
**function** *fetchURL*(url,callback,encoding){  
 **var** req= http.get(url,**function**(res){  
 **var** result=[];  
 res.on(**'data'**,**function**(chunked){  
 result.push(chunked);  
 });  
 res.on(**'end'**,**function**(){  
 callback(**null**,***iconv***.decode(Buffer.concat(result),encoding).slice(0,1000));  
 });  
 res.on(**'error'**,**function**(err){  
 callback(err,**null**);  
 });  
 });  
 req.end();  
}

总结：

1. 重复的代码最好写成一个方法。代码行数一下子减少了许多。
2. 注意题目要求（希望task的参数是一个对象，而不是数组）

# async-times

**语法：**

times(n,iterator,[callback])

**参数：**

* n：运行次数
* iterator：被调用n次的函数function(n,next)
* callback:与map中的相同

**使用场景**

一个函数被多次调用，需要知道与次数有关的参数的时候可以使用该函数。

# aync-waterfall

**定义：**

waterfall(tasks,[callback])

**参数：**

* tasks:由函数组成的数组
  + 在函数调用结束之前，必须传递callback(err,result1,result2….).如果没有错误的时候第一个参数传递为null，其他参数会作为下一个数组函数中的形参
* callback(err,[results])
  + 可选参数
  + 执行时机：数组中所有的函数都执行完毕之后，或者tasks中的回调函数中有错误的时候，最后一个回到函数会被立即调用，并且数组中剩下的回到函数将不会被执行。
  + results.它是tasks回调函数中的最后一个值。

**问题描述：**

In this problem you will need to write a program that first reads the contents

of a file.

The path will be provided as the first command-line argument to your program

(i.e. `process.argv[2]`).

The file will contain a single URL. Using `http.get`, create a GET request to

this URL and `console.log` the response body.

**解答：**

*/\*\*  
 \* Created by John on 2015-08-26.  
 \*/***var *fs***=*require*(**'fs'**);  
**var *http***=*require*(**'http'**);  
**var *async***=*require*(**'async'**);  
**var *path***=*require*(**'path'**);  
  
***async***.waterfall([**function**(callback){  
 **var** filename=***path***.join(**'../public'**,**'async\_waterfall.txt'**);  
 *//console.log(filename);* **var** url=***fs***.readFileSync(filename);  
 *// console.log(url.toString());* callback(**null**,url.toString());  
},**function**(url,callback){  
 **var** req=***http***.get(url,**function**(res){  
 **var** result=[];  
 res.on(**'data'**,**function**(chunked){  
 result.push(chunked);  
 });  
 res.on(**'end'**,**function**(){  
 callback(**null**,Buffer.concat(result).toString().slice(0,1000));  
 });  
  
 res.on(**'error'**,**function**(err){  
 **if**(err){  
 **return *console***.log(err,**message**);  
 }  
 });  
 });  
  
}],**function**(err,results){  
  
 ***console***.log(results);  
});

备注：

fs.readFileSync()得到的数组为buffer类型。

# async-whilst

whilst是有时的意思。

**定义：**

whilst(test,fn,callback)

**参数：**

* test()---在执行fn之前的检查
* fn(callback)
  + 通过test()之后的执行函数。
  + callback(err),必须被执行，当回调函数中有错误的时候
* callback(err)
  + 执行时机：当fn中的回调函数有错误的时候，或者是test()中返回false的时候

问题：

Write a program that will receive a single command line argument to a URL.

Using `async.whilst` and `http.get`, send GET requests to this URL until

the response body contains the string `"meerkat"`.

`console.log` the amount of GET requests needed to retrieve the "meerkat" string.

解答：

*/\*\*  
 \* Created by John on 2015-08-26.  
 \*/* **var *http***=*require*(**'http'**);  
**var *async***=*require*(**'async'**);  
  
**var *result***=**""**;  
**var *count***=0;  
  
***async***.whilst(**function**(){  
 **var** pattern =/.\*meerkat.\*/;  
 **if**(pattern.test(***result***)){  
 **return false**;  
 }  
 **else  
 return true**;  
 },  
**function**(callback){  
  
 **var** url=**"http://localhost:8014/?count="**+***count***;  
 **var** req=***http***.get(url,**function**(res){  
  
 **var** res\_data=**""**;  
  
 res.on(**'data'**,**function**(chunked){  
 res\_data+=chunked;  
 });  
 res.on(**'end'**,**function**(){  
 ***console***.log(url);  
 ***result***=res\_data;  
 ***count***++;  
 setTimeout(callback, 1000);  
 });  
 res.on(**'error'**,**function**(err){  
 callback(err);  
 });  
 });  
 req.end();  
},  
**function**(err){  
 **if**(err){  
 ***console***.log(err.**message**);  
 }  
 *//记录需要调用的请求数目* ***console***.log(***count***);  
});

备注：

如果自己没有上面的那段代码的话setTimeout(callback, 1000);，那么请求只会有一次。