# http.get

1:问题描述

编写一个程序来发起一个 **HTTP GET 请求**，所请求的 URL 为**命令行参数的第一个**。然后将\*\*每一个\*\* **"data" 事件**所得的数据，以字符串形式在终端（标准输出 stdout）的新的一行打印出来。

2:思路分析

http.get(options, [callback])[**#**](file:///D:\WebStorm9.0.2\learnyounode\learnyounode-master\node_apidoc\http.html#http_http_get_options_callback)

向服务器发起一个get请求。

由于大多数请求是没有请求体的get请求，这个方法是node提供的关于get请求的一个简单的方法。

http.get方法与http.request方法的区别是

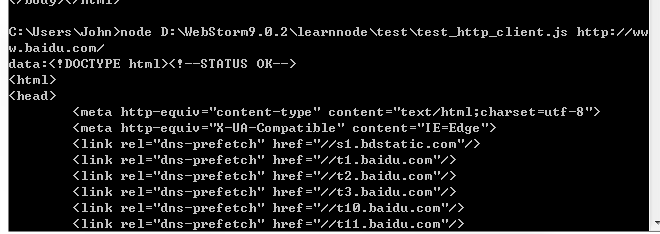
1. 不需要在option中设置这个method是get请求。
2. 不需要在callback的req对象中设置req.end();

命令行的第一个参数

process.argv[2]

3:解决方法

**var *http***=*require*(**'http'**);  
***http***.get(***process***.**argv**[2],**function**(res){  
  
 res.setEncoding(**'utf8'**)  
 res.on(**'data'**,**function**(chunk){  
 ***console***.log(**"data:"**+chunk);  
 })  
});



# http.IncomingMessage

**1:问题描述**

编写一个程序，发起一个 **HTTP GET 请求**，请求的 URL 为所提供给你的命令行参数的第一个。收集\*\*所有\*\*服务器所返回的数据（不仅仅包括 "data" 事件）然后在终端（标准输出 stdout）用两行打印出来。

你所打印的内容，第一行应该是一个整数，用来表示你所收到的字符串内容长度，第二行则是服务器返回给你的完整的字符串结果。

**2：思路分析**

**2.1setEncoding**

`http.get()` 所获得的 `response` 对象/Stream 还有一个 `setEncoding()` 的方法。如果你调用这个方法，并为其指定参数为 `utf8`，那么 `data` 事件中会传递字符串，而不是标准的 Node `Buffer` 对象，这样，你也不用再手动将 `Buffer` 对象转换成字符串了。

**2.2第三方的模块—bl**

bl是一个用于收集Node Buffers的存储对象。并且可以作为一个流，收集冲流来的buffers，把这些buffers释放在流中，从而你可以使用它。

**2.3buffer的问题所在。**

1：data事件中回调函数的参数的数据类型？

**问题分析：**

http.get()中的回调事件会触发response事情，回调函数中的参数res，它是httpIncomingMessage类的实例对象。

httpIncomingMessage类继承stream.readable.

stream.readable它是**\_stream\_readable.js的对象。**

这个可读流中会触发data事件。

stream.emit('data', chunk);

chunk = state.decoder.write(chunk);

chunk = new Buffer(chunk, encoding);

**解答：**

可以看到chunk返回的可能是buffer对象，也可能是一个buffer.toString(this.encoding, 0, end);

**总结：**

* http.get() 它是http.ClientRequest类的实例对象。
* 并且为该请求对象增加一个response事件监听。
* 当有response头返回的实收response事件触发。回调函数的参数是http.IncomingMessage 类的实例对象。
* http.IncomingMessgae类继承stream.readable,在stream.readable中会触发data事件。data事件的回调函数的参数chunk的数据类型可能是buffer类型，要可能是Buffer转换成的字符串。

3.2解决方法

**var *http***=*require*(**'http'**);  
**const *BufferList*** = *require*(**'bl'**);  
**var *bl*** = **new *BufferList***();  
  
***http***.get(***process***.**argv**[2],**function**(res){  
  
 res.on(**'data'**,**function**(chunk){  
 ***bl***.append(chunk);  
 })  
  
 res.on(**'end'**,**function**(){  
 ***console***.log(***bl***.toString());  
 });  
  
});

# http.createServer

1:问题描述

编写一个 HTTP 文件 \*\*服务器\*\*，它用于将每次所请求的文件返回给客户端。

你的服务器需要监听所提供给你的第一个命令行参数所制定的端口。

同时，第二个会提供给你的程序的参数则是所需要响应的文本文件的位置。在这一题中，你\*\*必须\*\*使用 `fs.createReadStream()` 方法以 stream 的形式作出请求相应。

2：思路分析

http.createServer(requestListener)

它返回http.Server 类的实例对象，在回调函数中增加request事件的监听。

fs.createReadStream()

返回一个可读流对象。

可读流中的pipe()的作用？

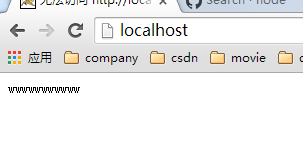
readable.pipe(destination, [options])[**#**](file:///D:\WebStorm9.0.2\learnyounode\learnyounode-master\node_apidoc\stream.html#stream_readable_pipe_destination_options)

* destination [**Writable**](file:///D:\WebStorm9.0.2\learnyounode\learnyounode-master\node_apidoc\stream.html#stream_class_stream_writable) Stream The destination for writing data
* options Object Pipe options
* end Boolean End the writer when the reader ends. Default = true

将可读流写入可写流，并且在读写完毕之后，关闭可写流，让它不能再写入。

3:解决方案

**var *http*** = *require*(**'http'**)  
**var *fs*** = *require*(**'fs'**)  
  
**var *server*** = ***http***.createServer(**function** (req, res) {  
 res.writeHead(200, { **'content-type'**: **'text/plain'** })  
  
 ***fs***.createReadStream(**'D:\WebStorm9.0.2\\learnnode\\public\\txt\\test\_http\_file\_server.txt'**).pipe(res)  
  
})  
  
***server***.listen(80)



nodejs中如何写入路劲？

通过path模块来控制，更加的方便。可扩展性，以及可读性都更加的好。

**var *http*** = *require*(**'http'**)  
**var *fs*** = *require*(**'fs'**);  
**var *path***=*require*(**'path'**);  
  
**var *server*** = ***http***.createServer(**function** (req, res) {  
 res.writeHead(200, { **'content-type'**: **'text/plain'** })  
  
 *//fs.createReadStream('D:\WebStorm9.0.2\\learnnode\\public\\txt\\test\_http\_file\_server.txt').pipe(res)* ***fs***.createReadStream(***path***.join(**'D:/WebStorm9.0.2/learnnode/public/txt'**,**'test\_http\_file\_server.txt'**)).pipe(res);  
  
})  
***server***.listen(80)

# 从服务器端返回json类型的数据

问题描述

编写一个 HTTP \*\*服务器\*\*，每当接收到一个路径为 '/api/parsetime' 的 GET 请求的时候，响应一些 JSON 数据。我们期望请求会包含一个查询参数（query string），key 是 "iso"，值是 ISO 格式的时间。

如:

/api/parsetime?iso=2013-08-10T12:10:15.474Z

所响应的 JSON 应该只包含三个属性：'hour'，'minute' 和 'second'。例如：

```json

{

"hour": 14,

"minute": 23,

"second": 15

}

```

然后增再加一个接口，路径为 '/api/unixtime'，它可以接收相同的查询参数（query string），但是它的返回会包含一个属性：'unixtime'，相应值是一个 UNIX 时间戳。例如:

```json

{ "unixtime": 1376136615474 }

```

你的服务器需要监听第一个命令行参数所指定的端口。

response.end()方法一定需要吗？那些情况下是一定需要response.end(),那些情形下不需要这个方法也可以的？

The method, response.end(), MUST be called on each response.

但是我写的前面一个demo中就没有使用response.end().这是为什么了？文档中又说每个response都必须使用response.end();

readstream(路径).pipe(writestream)

这是由于pipe()方法会关闭可写流，因此也就不需要response.end()

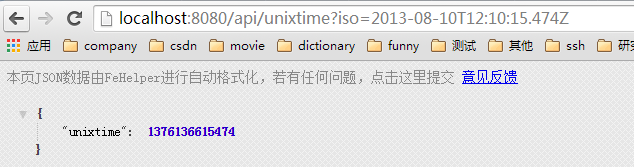
在nodejs中我如何把json数据传到服务器端？

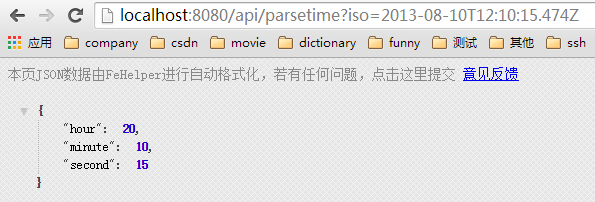
response.write()方法的第一个参数必须是string类型或者是buffer类型的。难道说我要把一个json对象

1. 设置content-type为application/json类型
2. JSON.stringify()的使用。

3：解决方法

**var *http***=*require*(**'http'**);  
**var *url***=*require*(**'url'**);  
**var *path***=*require*(**'path'**);  
  
**var *server***=***http***.createServer(**function**(req,res){  
 res.writeHead(200, { **'content-type'**: **'application/json'** });  
  
  
 **var** url\_parese=***url***.parse(req.**url**,**true**);  
 **var** path\_basename=***path***.basename(url\_parese.**pathname**);  
  
 **var** iso = **new** Date(url\_parese.**query**[**'iso'**]);  
 **var** hour = iso.getHours();  
 **var** minute = iso.getMinutes();  
 **var** second = iso.getSeconds();  
 **var** unixtime=iso.getTime();  
  
 **var** result\_parse = {  
 **"hour"**: hour,  
 **"minute"**: minute,  
 **"second"**: second  
 };  
 **var** result\_unix={  
 **"unixtime"**:unixtime  
 }  
  
 **if**(path\_basename==**'parsetime'**) {  
 res.write(**JSON**.stringify(result\_parse));  
 }  
 **if**(path\_basename==**'unixtime'**){  
 res.write(**JSON**.stringify(result\_unix));  
  
  
 }  
 res.end();  
  
});  
  
***server***.listen(8080);





问题描述：

编写一个 HTTP \*\*服务器\*\*，它只接受 POST 形式的请求，并且将 POST 请求主体（body）所带的字符转换成大写形式，然后返回给客户端。

你的服务器需要监听由第一个命令行参数所指定的端口。

思路分析：

如何获取post请求体中的数据？

通过req.body可以访问，不清楚为什么在文档中没有对应的说明。

遍历一个对象，并修改属性值？

**var** receiver\_data=req.**body**;  
**var** send\_data={};  
  
**for**(**var** p **in** receiver\_data){  
 send\_data[p]=receiver\_data[p].toUpperCase();  
}

标准答案中是使用了一个第三方的模块

through2-map

为什么在http.createServer(requestLisenter)中的回调函数中的request事件中没有监听data事件。在文档中说这个request对象时http.IncomingMessage 类的实例对象，http.get(responselisterner).的回调函数的参数也是http.IncomingMessage类的实例对象，但是它却可以监听到data事件，这个为什么？

这是由于在express中对于req对象做了封装，如果要获取请求体的话，使用req.body就好了，但是在http.createServer中的req对象是可以监听data事件的。

接收post请求

* 自己想到的是使用表单发送一个post请求，然后再进行路由的控制等等，
* 其实http.request()方法就可以发送post请求。

解决方案：

**var *http***=*require*(**'http'**);  
**var *querystring***=*require*(**'querystring'**);  
  
**var *server***=***http***.createServer(**function**(req,res){  
 */\*���󷽷���req.method �õ���������  
 \* \*/  
 /\*�������壺  
 ����data�¼����м���  
 \* \*/* **var** recive\_data=**""**;  
 **var** send\_data={};  
  
  
 req.setEncoding(**'utf-8'**);  
 req.on(**'data'**, **function** (chunk) {  
  
 recive\_data=***querystring***.parse(chunk);  
  
 **for**(**var** p **in** recive\_data){  
 send\_data[p]=recive\_data[p].toUpperCase();  
 }  
 }) ;  
 req.on(**'end'**, **function** () {  
 res.writeHead(200,{**'content-type'**:**'text/plain;charset=UTF-8'**});  
 res.write(**JSON**.stringify(send\_data));  
 res.end();  
  
 });  
  
  
});  
  
***server***.listen(8080);

测试：

*/\*\*  
 \* Created by John on 2015-08-12.  
 \*/***var *http***=*require*(**'http'**);  
**var *querystring***=*require*(**'querystring'**);  
  
**var *postData*** = ***querystring***.stringify({  
 **content**:**'tom赞！'**,  
 **mid**:8837  
});  
  
**var *optinos***={  
 **hostname**: **'localhost'**,  
 **port**: 8080,  
 **method**: **'POST'**,  
 **'Content-Type'**:**'charset=UTF-8'**};  
  
**var *req***=***http***.**request**(***optinos***, **function** (res) {  
 res.setEncoding(**'utf-8'**);  
 res.on(**'data'**,**function**(chunked){  
 ***console***.log(chunked);  
 });  
  
});  
  
***req***.on(**'error'**, **function**(e) {  
 ***console***.log(**'problem with request: '** + e.**message**);  
});  
  
*// write data to request body****req***.write(***postData***);  
***req***.end();

# get读取数据

不乱码方式  
**var *http***=*require*(**'http'**);  
**const *BufferList*** = *require*(**'bl'**);  
**var *bl*** = **new *BufferList***();  
**var *result***=**""**;  
*//process.argv[2]****http***.get(**'http://www.baidu.com'**,**function**(res){  
 res.setEncoding(**'utf-8'**);  
  
 res.on(**'data'**,**function**(chunk){  
 ***result***+=chunk;  
 });  
  
 res.on(**'end'**,**function**(){  
 ***console***.log(***result***);  
 });  
});

## 乱码方式—setEncoding的缺陷

**var *http***=*require*(**'http'**);  
**const *BufferList*** = *require*(**'bl'**);  
**var *bl*** = **new *BufferList***();  
**var *result***=**""**;  
*//process.argv[2]****http***.get(**'http://www.qq.com'**,**function**(res){  
 res.setEncoding(**'utf-8'**);  
  
 res.on(**'data'**,**function**(chunk){  
 ***console***.log(chunk)  
 });  
  
});

**setEncoding()作用？**

* 可以让data事件中传递的不再是一个buffer对象，而是编码后的字符串。
* 但是setEncoding()只能处理UTF-8 , Base64和UCS-2/UTF-16LE这3种编码。

**setEncoding()的局限？  
var *http***=*require*(**'http'**);  
**const *BufferList*** = *require*(**'bl'**);  
**var *bl*** = **new *BufferList***();  
**var *result***=**""**;  
*//process.argv[2]****http***.get(**'http://www.qq.com'**,**function**(res){  
 res.setEncoding(**'utf-8'**);  
  
 res.on(**'data'**,**function**(chunk){  
 ***console***.log(chunk)  
 });  
  
});

由于qq页面的字符编码给是charset=gb2312。上面的代码就会出现乱码。

<!DOCTYPE html>

<html lang="zh-CN">

<head>

<meta content="text/html; charset=gb2312" http-equiv="Content-Type">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>��Ѷ��ҳ</title>

<script type="text/javascript">

**Buffer转字符串**

buf.toStirng([encoding],[start],[end])

但是中国常用的GBK GB2312 BIG-5都不支持

可以通过使用i**conv-lite**或者是**icon**模块达到效果。