## 慕课网 进击NODE.JS基础一

### 3-1 起一个web服务器

### 代码：

var http = require('http');

http.createServer(function(req,res){

res.writeHead(200,{'Content-Type':'text/plain'});

res.end('Hello World\n');

}).listen(1337,'127.0.0.1');

console.log('Server runing at http://127.0.0.1/');

#### 分析：

1. 第一行：加载了一个http模块，这个模块由js来编写的，它的职责就是创建一个web服务器及处理与http相关的任务
2. createServer创建了一个web服务器，然后通过listen让服务器在1337端口监听请求。服务器ready后能够收到任何来自端口的请求
3. 回调函数：只有当某个函数（更确切的说是函数的指针）被作为参数，被另一个函数调用时，它才是回调函数。createServer的参数就是一个回掉函数
4. res（请求对象,可以获得请求对象的一些信息）的来源：var res = new ServerResponse(req)

req（响应对象 ，收到请求后要做出的响应）的来源：来自parserOnIncoming的形参（具体不用深究）

1. http.createServer([requestListener])

requestListener   请求处理函数，自动添加到 request 事件，函数传递两个参数（req和res）

1. writeHead：给返回的请求头里写入状态码，200代表成功

### 3-2 命令行中体验

浏览器下和nodeJS下的全局变量是不一样的

浏览器中，打开developer Tools，在console中输入window（全局变量中有window）

但在NODEJS中，window不存在，但它有自己的全局变量，process

### 4-1Node.js的模块与common.js规范

没怎么听懂

### 4-2模块的分类

模块与文件是一一对应的，可以通过文件路径或者文件名来引用模块。如果用文件名来引用非核心模块，node会把模块名映射到对应的模块文件的路径。包含了核心函数的核心模块，会在Node启动时被预先加载

#### 核心模块

http、 fs、 path

#### 文件模块

var util = require(‘./util.js’)

#### 第三方模块

var promise = require(‘bluebird’) (通过npm加载的)

### 4-3 简单的NoDejs模块

可以使用require函数来加载模块，不用担心在代码中用require会污染全局命名空间，因为在node中没有全局命名空间这个概念。如果模块存在，且没有语法错误，那么require会返回这个模块对象。这个模块对象就是下面的exports。

#### 创建模块

teacher.js

function add(teacher){

console.log("add teacher:"+teacher);

}

#### 导出模块

1. exports.add = add;
2. module.exports = add;
3. module.exports.add = add;(与一同)

exports.add = function(){}或者 module.exports = function(){} （其中，add是自己定义的属性名）

如果希望模块是一个特殊的对象类型，则用module.exports；如果希望模块是一个传统的模块实例，则用exports。module.exports是真实存在的，而exports是module.exports的辅助方法。推荐使用第一种方式。

<https://cnodejs.org/topic/54e67b70629934692ea99f91>

链接里面对exports和module.exports讲的比较清楚。注意直接复制和添加属性的区别，如果是添加属性，那么两者是共同变化的，而直接赋值会切断另一个对同一个对象的引用；真正输出的是module.exports。

#### 加载模块

1. var teacher = require(‘./teacher.js’)
2. 上同
3. 上同

#### 使用模块

1. teacher.add(‘zhou’)
2. teacher (‘zhou’)
3. 与1同

### 5-2 URL网址解析的好帮手

<https://nodejs.org/docs/v0.10.35/> v0.10.35的API介绍

* [url.parse(urlStr, [parseQueryString], [slashesDenoteHost])](https://nodejs.org/docs/v0.10.35/api/url.html#url_url_parse_urlstr_parsequerystring_slashesdenotehost)
* [url.format(urlObj)](https://nodejs.org/docs/v0.10.35/api/url.html#url_url_format_urlobj)
* [url.resolve(from, to)](https://nodejs.org/docs/v0.10.35/api/url.html#url_url_resolve_from_to)

进入命令行，输入node，在node环境下输入url，可以看到API的使用方法。

### 5-3 Querystring参数处理小利器（querystring）

### 5-4 HTTP知识先填坑

流程：

1. http客户端发起请求，创建端口
2. http服务器在端口监听客户端请求
3. http服务器向客户端返回状态和内容

当输入网址按了回车或者按了F5以后，chrome和计算机其实做了很多事情（粗浅理解）：

1. 搜索自身的DNS缓存（在chrome浏览器输入chrome：//net-internals/#dns即可查看）

2. 搜索操作系统自身的DNS缓存（浏览器没有找到缓存或者缓存已经失效）

3.读取本地的HOST文件

4.浏览器发起一个DNS的一个系统调用

1. 宽带运营商服务器查看本身缓存

2.运营商服务器发起一个迭代的DNS解析的请求（找根域去问，逐层去问）

（根域->com域->imooc.com域）

1.运营商服务器把结果返回操作系统内核同时缓存起来（这里讲的生动搞笑）

2.操作系统内核把结果返回浏览器

3.最终浏览器拿到了[www.imooc.com](http://www.imooc.com)对应的地址

5.浏览器获得域名对应的IP地址后，发起HTTP“三次握手”

6.TCP/IP连接建立起来后，浏览器就可以向服务器发送HTTP请求了，比如说，用HTTP的GRT方

法请求一个根域里的一个域名，协议可以采用HTTP 1.0的一个协议

7.服务器端接收到了这个请求，根据路径参数，经过后端的一些处理之后，把处理后的一个结果

的数据返回给浏览器，如果是慕课网的页面就会把完整的HTML页面代码返回给浏览器

8.浏览器拿到了慕课网的完整的HTML页面代码，在解析和渲染这个页面的时候，里面的JS、

CSS、图片静态资源，他们同样也是一个个HTTP请求，都需要经过上面的主要的七个步骤

9.浏览器根据拿到的资源对页面进行渲染，最终把一个完整的页面呈现给用户

### [5-5 HTTP知识填坑之“以慕课网为例分析”](https://www.imooc.com/video/6713)

进入慕课网，刷新页面，进入到develop tools里面的network，点开里面的list，可以查看请求。

请求方法：GET（获取数据）、POST（提交数据）、PUT（更新）、DELETE（删除）、HEAD（与GET一样，只是服务器不传回本文的内容，不用传输全部内容）、TRACE、OPTIONS

状态码：1xx：指示信息，表示请求已经接受，继续处理

2xx：接受成功，处理结束。 200：成功

3xx：表示要完成这个请求需要更进一步的处理。

4xx：客户端错误。 400： 语法错误 401：没有授权 403：服务器拒绝服务 404：没找到

5xx：服务器端错误。500：服务器端发生了不可预测的错误 503：服务器当前不能处理这个请求

### [5-6 HTTP 事件回调进阶](https://www.imooc.com/video/7557)

回掉：单线程的nodejs通过回掉的方式实现了异步，实现非阻塞

异步：打电话给饭店老板问有没有位置，老板说我去查查，挂电话，查到了回电话

同步：打电话给饭店老板问有没有位置，老板说我去查查，不挂电话，要在电话这边等着

阻塞：同步状态下，因为老板去查结果，自己挂起等待，直到拿到结果

非阻塞：异步状态，挂了电话后随便去做啥

### [5-7 HTTP 源码解读之先了解作用域、上下文](https://www.imooc.com/video/7558)

function Pet(words) {

this.words = words;

this.speak = function (say) {

console.log(say + " " + words)

}

}

function Dog(words) {

Pet.apply(this,[words])

// Pet.call(this, words);

}

var dog = new Dog('tang')

dog.speak('zhou');

用apply 和call来实现的继承

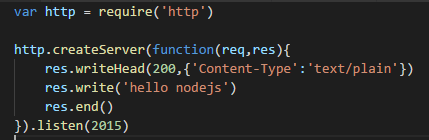
### [5-8 HTTP 源码解读](https://www.imooc.com/video/7963)

（卧槽，原来最开始讲的那个createServer会在这边讲）

在github中搜索node，在官方的仓库中的分支中选择版本。（小技巧：按t可以跳出搜索）

### [5-9 HTTP 性能测试](https://www.imooc.com/video/7964)

代码

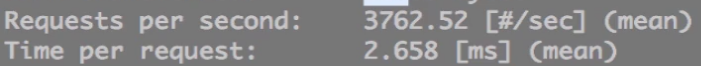


#### 性能测试（就是看网页执行需要多久，这样自己写完代码可以去测试一下是否需要做优化）

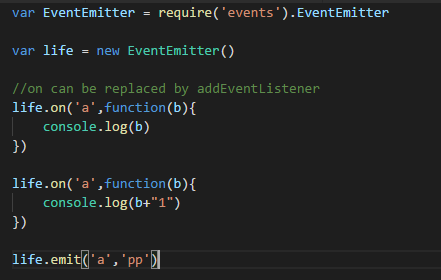
1. 输入node http.js（文件名）后，在浏览器中输入localhost:2015就能看到在页面上显示的hello nodejs
2. 新起一个命令窗口，输入ab –n1000 –c10 http://localhost:2015/ -n1000表示总的请求数为1000个（默认值 为1），-c10表示并发数位10（默认值为1）（在此说明一共发了1000个，一次发了10个，所以一共发了10次），除此之外-t表示测试的总时间，-p为post时的数据文件，-w是以html表格的形式输出结果。后面跟上测试的页面
3. **ab**命令是Apache的Web服务器的性能测试工具，它可以测试安装Web服务器每秒种处理的HTTP请求。<http://httpd.apache.org/download.cgi#apache24> <https://blog.csdn.net/qq_27682041/article/details/73897679>
4. 比较重要去关注：

Requests per second (每秒的事物数)

Time per request （平均请求的等待时间，相当于平均事物的响应时间）



### [5-11 事件模块小插曲](https://www.imooc.com/video/8525)



1. on:相当于addEventListener，可以为一个事件设置10个监听器，太多可能会造成内存泄漏。如果想要增加事件监听上限，可以 life.setMaxListeners(11)来设置，放在new之后以及添加监听事件以前。
2. emit:执行事件。
3. 如果给life的“a”事件有11个，再添加两个其他名字的事件，并不会占用“a”事件的事件数。说明是给每个事件分配11个最大额度。
4. life.emit函数会有一个bool的返回值，如果是true则说明该事件被监听过，如果是false,则没有被监听过。在上述代码中，并没有为c添加监听，所以这样pl的值为false
5. 如果想要移除事件监听。life.removeListener(‘a’,function(b){console.log(b)})是不行的。第二个参数必须是具名函数。如下图：



1. 如果想要看一个事件有多少监听函数，可以console.log(life.listeners(‘a’).length)来查看，或者通过console.log(EventEmitter.listenerCount(life,’a’))来查看

### [5-12 Node.js：request方法](https://www.imooc.com/video/8837)

1. get是对request的一个封装，get能做的事，request都能做。

## 慕课网 进击NODE.JS基础二

<https://www.imooc.com/learn/637>