终于来到了传说中的Nodejs了，好开心！

http://my.oschina.net/ximidao/blog/413101

Node.js是一个Javascript运行环境(runtime)。实际上它是对Google V8引擎进行了封装。V8引 擎执行Javascript的速度非常快，性能非常好。Node.js对一些特殊用例进行了优化，提供了替代的API，使得V8在非浏览器环境下运行得更好。

Node.js是一个基于Chrome JavaScript运行时建立的平台， 用于方便地搭建响应速度快、易于扩展的网络应用。Node.js 使用[事件驱动](http://baike.baidu.com/view/536048.htm)， 非阻塞[I/O](http://baike.baidu.com/subview/300881/11169495.htm) 模型而得以轻量和高效，非常适合在分布式设备上运行的数据密集型的实时应用。

2009年5月

**Node.js的优点**

nodejs作为一个新兴的后台语言，有很多吸引人的地方：

RESTful API

单线程

Node.js可以在不新增额外线程的情况下，依然可以对任务进行并行处理 —— Node.js是单线程的。它通过事件轮询（event loop）来实现并行操作，对此，我们应该要充分利用这一点 —— 尽可能的避免阻塞操作，取而代之，多使用非阻塞操作。

非阻塞IO

V8虚拟机

事件驱动[4]

Linux 下 Node.js的安装

下面介绍下Node.js的安装，首先在nodejs的网站上根据操作系统下载相关的安装包，对于ubuntu[5]  (linux)下的安装，可以如下进行：

sudo apt-get update

sudo apt-get install node

执行which node 看安装路径

shutdown -r now ： 表示现在重启计算机！

另一种重启方式就是输入： reboot   也表示重启！

安装npm:apt-get install npm（node包管理器）

查找模块

<http://search.npmjs.org/>

npm search irc

本地安装

npm install underscore

全局安装

Npm install –g express

查看文档

Npm docs [modulename]

统一安装模块

Npm install，通过package.json安装

<!--[if It IE 9]>

<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>

<![endif]-->

Ghost的配置：

https://segmentfault.com/a/1190000000372040

node.js、npm、express的安装及express使用命令的配置1

在node.js的官网https://nodejs.org/下载安装程,双击安装。

1. 2

安装后，打开运行命令窗口，输入cmd。输入node -v,测试是否安装成功，会输出版本信息。

新的node.js已经集成了npm，所以npm也一并装好了，输入npm -v测试，会输出版本信息。

1. 3

先配置npm的全局模块的存放路径以及cache路径：

npm config set prefix "D:\nodejs\node\_global"

npm config set cache "D:\nodejs\node\_cache"

1. 4

安装express模块。

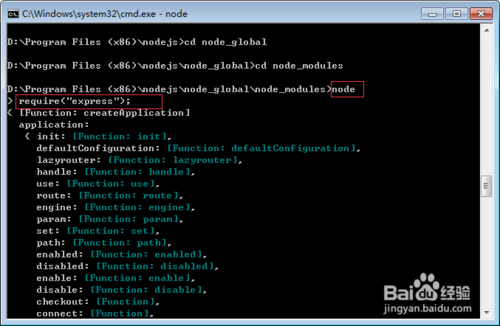
命令行里输入npm install express -g

-g 意思是装在global目录下

5

查看是否安装成功。进入到全局的D:\Program Files (x86)\nodejs\node\_global目录下，输入node 回车，输入require("express");出现以下信息即安装成功。

**ps:退出nodejs终端命令行，ctrl+D一次 或  ctrl+C两次 或输入“ .exit” 即可**

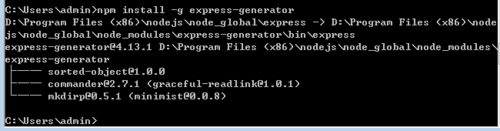
[](http://jingyan.baidu.com/album/7f41ecec0e3a25593d095c26.html?picindex=4)

1. 6

express命令的使用，查看版本信息。输入express -V ,提示express不是内部或外部命令。

因为express4.x中将命令工具分离出来了，所有需要先装express-generator。

命令行输入：npm install -g express-generator，即安装成功。

[](http://jingyan.baidu.com/album/7f41ecec0e3a25593d095c26.html?picindex=5)

1. 7

使用express命令前，还需要配置下环境变量。

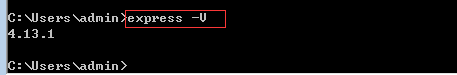
打开计算机，系统属性——高级系统设置——环境变量；在系统变量下，新建NODE\_PATH，值为D:\Program Files (x86)\nodejs\node\_global（全局路径），将“%NODE\_PATH%”到path变量值后面即可。

1. 8

测试express命令。命令行下输入express -V

显示版本信息，即安装成功。

PS：express -V 中的V 必须是大写的。

[](http://jingyan.baidu.com/album/7f41ecec0e3a25593d095c26.html?picindex=11)

app.set('port', 3000);

app.listen(app.get('port'), function () {

console.log('Express server listening on port ' + app.get('port'));

});

http头

http://kb.cnblogs.com/page/92320/

**Node的小基友supervisor**

　　每次修改代码后会自动重启。懒程序员就指望这种省事省力的工具活着了:)

　　安装：**npm install -g supervisor**

　　执行：**supervisor app.js**

http://www.searchsoa.com.cn/showcontent\_79099.htm

部署方式1：(阿里云)Nginx, nodejs, Express, forever

http://cnodejs.org/topic/5059ce39fd37ea6b2f07e1a3

部署方式2：PM2

<http://yijiebuyi.com/blog/77db2532be0afae2959cd918331d9dda.html>

不错的博文<http://www.cnblogs.com/Darren_code/p/node_express.html>

Mongoose

http://www.nodeclass.com/api/mongoose.html

http://cnodejs.org/topic/504b4924e2b84515770103dd

Nodejs实践：http://blog.csdn.net/yanghua\_kobe/article/details/17199417

Nodejs同步代码

  众所周知，异步是nodejs中得天独厚的特点和优势，但同时在程序中同步的需求（比如控制程序的执行顺序为：func1 -> func2 ->func3 ）也是很常见的。本文就是对这个问题记录自己的一些想法。

     需要执行的函数：

[复制代码](javascript:void(0);)

var func1 = function(req,res,callback){  
 setTimeout(function(){  
 console.log('in func1');  
 callback(req,res,1);   
 },13000);  
}  
var func2 = function(req,res,callback){   
 setTimeout(function(){  
 console.log('in func2');  
 callback(req,res,2);  
 },5000);  
}  
  
var func3 = function(req,res,callback){  
 setTimeout(function(){  
 console.log('in func3');  
 callback(req,res,3);   
 },1000);  
}

[复制代码](javascript:void(0);)

      可以看出在func1，func2和func3中都是用了setTimeout函数，执行的时间分别为13秒，5秒和1秒。由于nodejs异步的特性，如果使用普通的函数调用方法：

[复制代码](javascript:void(0);)

var req = null;  
var res = null;  
var callback = function(){};  
func1(req,res,callback);  
func2(req,res,callback);  
func3(req,res,callback);

[复制代码](javascript:void(0);)

     输出内容：

in func3  
in func2  
in func1

     原因是因为nodejs是异步的，func2不会等func1执行完毕后再执行，而是立即执行（func3也是如此）。由于func3的运行时间最短而率先结束，func2次之，func1最后。但这明显不是我们想要的结果。怎么办？

**解决办法一：callback**

[复制代码](javascript:void(0);)

//深层嵌套  
var req = null;  
var res = null;  
  
func1(req,res,function(){  
 func2(req,res,function(){  
 func3(req,res,function(){  
 process.exit(0);   
 })   
 });   
});

[复制代码](javascript:void(0);)

     这种方法虽然能快速的解决，但暴露的问题也很明显，一是代码维护不方面，二是代码的深层嵌套看起来很不舒服。这种方法并不可取。

**解决方法二：递归调用**

[复制代码](javascript:void(0);)

function executeFunc(funcs,count,sum,req,res){  
 if(count == sum){  
 return ;   
 }  
 else{  
 funcs[count](req,req,function(){  
 count++;  
 executeFunc(funcs,count,sum,req,res);  
 });  
 }   
}  
  
//同步调用  
var req = null;  
var res = null;  
var funcs = [func1,func2,func3];  
var len = funcs.length;  
executeFunc(funcs,0,len,req,res);

[复制代码](javascript:void(0);)

        先将多个函数组成一个数组。再可以利用递归函数的特性，使程序按照一定的顺序执行。

**解决方法三：调用类库**

       随着nodejs的发展，响应的类库也越来越多。[Step](https://github.com/creationix/step)和async 就是其中不错的。

       1. Step的调用相对比较清爽：

[复制代码](javascript:void(0);)

Step(  
 function thefunc1(){  
 func1(this);  
 },  
 function thefunc2(finishFlag){  
 console.log(finishFlag);  
 func2(this);  
 },  
 function thefunc3(finishFlag){  
 console.log(finishFlag);  
 }  
);

[复制代码](javascript:void(0);)

       2.async 的 [series](https://github.com/caolan/async#series)方法，就本例而言，它的调用方法：

[复制代码](javascript:void(0);)

var req = null;  
var res = null;  
var callback = function(){};  
  
async.series(  
 [  
 function(callback){  
 func1(req,res,callback);  
 },   
 function(callback){  
 func2(req,res,callback);  
 },  
 function(callback){  
 func3(req,res,callback);   
 }   
 ]  
);

[复制代码](javascript:void(0);)

下面是我们使用Node.js时遵循的10个性能规则：

1. 避免使用同步代码

在设计上，Node.js是单线程的。为了能让一个单线程处理许多并发的请求，你可以永远不要让线程等待阻塞，同步或长时间运行的操作。Node.js的一个显著特征是：它从上到下的设计和实现都是为了实现异步。这让它非常适合用于事件型程序。

不幸的是，还是有可能会发生同步/阻塞的调用。例如，许多文件系统操作同时拥有同步和异步的版本，比如writeFile和writeFileSync。即使你用代码来控制同步方法，但还是有可能不注意地用到阻塞调用的外部函数库。当你这么做时，对性能的影响是极大的。

[?](http://www.jb51.net/article/52187.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | // Good: write files asynchronously  fs.writeFile('message.txt', 'Hello Node', function (err) {   console.log("It's saved and the server remains responsive!");  });    // BAD: write files synchronously  fs.writeFileSync('message.txt', 'Hello Node');  console.log("It's saved, but you just blocked ALL requests!"); |

我们的初始化log在实现时无意地包含了一个同步调用来将内容写入磁盘。如果我们不做性能测试那么就会很容易忽略这个问题。当以developer box中一个node.js实例来作为标准测试，这个同步调用将导致性能从每秒上千次的请求降至只有几十个。

2.关闭套接字池

Node.js的http客户端会自动地使用套接字池：默认地，它会限制每台主机只能有5个套接字。虽然套接字的重复使用可能会让资源的增加在控制之下，但如果你需要处理许多数据来自于同一主机的并发请求时，将会导致一系列的瓶颈。在这种情况下，增大maxSockets 的值或关闭套接字池是个好主意：

[?](http://www.jb51.net/article/52187.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | // Disable socket pooling    var http = require('http');  var options = {.....};  options.agent = false;  var req = http.request(options) |

3.不要让静态资源使用Node.js

对于css和图片等静态资源，用标准的WebServer而不是Node.js。例如，领英移动使用的是nginx。我们同时还利用内容传递网络（CDNs），它能将世界范围内的静态资拷贝到服务器上。这有两个好处：（1）能减少我们node.js服务器的负载量（2）CDNs可以让静态内容在离用户较近的服务器上传递，以此来减少等待时间。

4.在客户端渲染

让我们快速比较一下服务器渲染和客户端渲染的区别。如果我们用node.js在服务器端渲染，对于每个请求我们都会回送像下面这样的HTML页面：

[?](http://www.jb51.net/article/52187.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <!-- An example of a simple webpage rendered entirely server side -->    <!DOCTYPE html>  <html>   <head>    <title>LinkedIn Mobile</title>   </head>   <body>    <div class="header">     <img src="<http://mobile-cdn.linkedin.com/images/linkedin.png>" alt="LinkedIn"/>    </div>    <div class="body">     Hello John!    </div>   </body>  </html> |

请注意观察这个页面所有的内容，除了用户的名字，其余都是静态内容：对于每个用户和页面重载内容都是一样的。因此更有效的作法是让Node.js仅以JSON形式返回页面需要的动态内容。

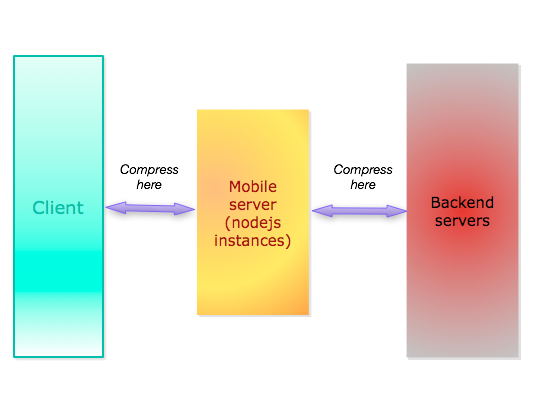
{"name": "John"}  
页面的其余部分—所有静态的HTML标记－能放在JavaScript模板中（比如underscore.js模板）：

[?](http://www.jb51.net/article/52187.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <!-- An example of a JavaScript template that can be rendered client side -->    <!DOCTYPE html>  <html>   <head>    <title>LinkedIn Mobile</title>   </head>   <body>    <div class="header">     <img src="<http://mobile-cdn.linkedin.com/images/linkedin.png>" alt="LinkedIn"/>    </div>    <div class="body">     Hello <%= name %>!    </div>   </body>  </html> |

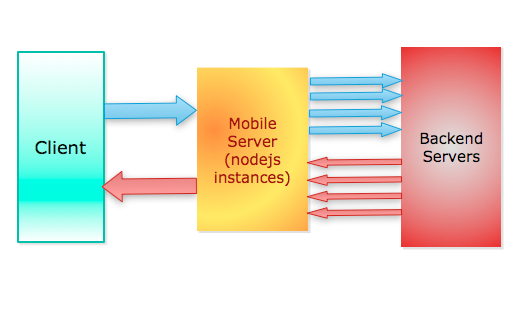
性能的提升来自于这些地方：如第三点所说，静态JavaScript模板能通过webserver（比如nginx）在服务器端提供，或者用更好的CDN来实现。此外，JavaScript模板能缓存在浏览器中或存储在本地，所有初始页面加载以后，唯一需要发送给客户端的数据就是JSON，这将是最有效果的。这个方法能极大性地减少CPU，IO，和Node.js的负载量。

5.使用gzip



许多服务器和客户端支持gzip来压缩请求和应答。无论是应答客户端还是向远程服务器发送请求，请确保充分使用它。

6.并行化



试着让你所有的阻塞操作－向远程服务发送请求，DB调用，文件系统访问并行化。这将能减少最慢的阻塞操作的等待时间，而不是所有阻塞操作的等待时间。为了保持回调和错误处理的干净，我们使用Step来控制流量。

7.Session自由化

领英移动使用Express框架来管理请求/应答周期。许多express的例子都包含如下的配置：

app.use(express.session({ secret: "keyboard cat" }));  
默认地，session数据是存储在内存中的，这会给服务器增加巨大的开销，特别是随着用户量的增长。你可以使用一个外部session存储，比如MongoDB或Redis，不过每一个请求将会导致远程调用来取得session数据的开销。在可能的情况下，最好的选择就是在服务器端存储所有的无状态数据。通过不包含上述express配置让session自由化，你会看到更好的性能。

8.使用二进制模块

如果可能，用二进制模块取代JavaScript模块。例如，当我们从用JavaScript写的SHA模块转换到Node.js的编译版本，我们会看到性能的一个大跃进：

[?](http://www.jb51.net/article/52187.htm)

|  |  |
| --- | --- |
| 1  2  3 | // Use built in or binary modules  var crypto = require('crypto');  var hash = crypto.createHmac("sha1",key).update(signatureBase).digest("base64"); |

9.用标准的 V8 JavaScript 取代客户端库

许多JavaScript库都是为了在web浏览器上使用而创建的，因为在JavaScript环境不同时：比如，一些浏览器支持forEach，map和reduce这样的函数，但有些浏览器不支持。因此客户端库通常用许多低效的代码来克服浏览器的差异。另一方面，在Node.js中，你能确切地知道哪些JavaScript方法是有效的：V8 JavaScript引擎支撑Node.js实现ECMA-262第五版中指定的ECMAScript。直接用标准的V8 JavaScript函数替代客户端库，你会发现性能得到显著的提高。

10.让你的代码保持小且轻

使用移动设备会让访问速度慢且延迟高，这告诉我们要让我们的代码保持小且轻。对于服务器代码也保持同样的理念。偶尔回头看看你的决定且问自己像这样的问题：“我们真的需要这个模块吗？”，“我们为什么用这个框架，它的开销值得我们使用吗？”，“我们能用简便的方法实现它吗？”。小轻且的代码通常更高效、快速。

试试看

我们很努力地让自己的移动应用变得快速。在IPhone应用，Android应用和HTML5移动版本这些平台上尝试一下，让我们知道自己做得怎么样。

**node-xlsx**

1 **var** xlsx **=** require('**node-xlsx**');

2 **var** fs **=** require('fs');

3 **var** obj **=** {"worksheets"**:**[{"data"**:**[["姓名","性别","年龄"],["李晓龙","男","24"]]}]};

4 **var** file **=** xlsx.build(obj);

5 fs.writeFileSync('user.xlsx', file, 'binary');

*//node.io版*

1 **var** nodeio **=** require('node.io');

2 **var** xlsx **=** require('node-xlsx');

3 **var** fs **=** require('fs');

4 **var** jsonData **=** [{name**:**'lixiaodong',age**:**24,gender**:**'m'},{name**:**'zhaoge',age**:**24,gender**:**'m'}];

5

6 exports.job **=** **new** nodeio.Job({

7 input**:** **false**,*//从服务器获取数据*

8 run**:** **function** (row) {

9 **var** keyNames **=** [];

10 keyNames[0] **=** [];

11 **var** value **=** [];

12 **if**(jsonData.length){

13 **for**(**var** key **in** jsonData[0]){

14 keyNames[0].push(key);

15 }

16 **for**(**var** i**=**0;i**<**jsonData.length;i**++**){

17 value[i] **=** [];

18 **for**(**var** key **in** jsonData[i]){

19 value[i].push(jsonData[i][key]);

20 }

21 }

22 }

23 **for**(**var** i**=**0;i**<**value.length;i**++**){

24 keyNames[keyNames.length] **=** value[i];

25 }

26 console.log('value:',value);

27 console.log('keyNames:',keyNames);

28 **this**.emit(keyNames);

29 },

30 output**:** **function**(data){

31 console.log('data:',data);

32 **var** obj **=** {"worksheets"**:**[{"data"**:**data}]};

33 **var** file **=** xlsx.build(obj);

34 fs.writeFileSync('user.xlsx', file, 'binary');

35 }

36 });

博客Ghost

https://segmentfault.com/a/1190000000372040