1.使用coffeescript编写，我已编译成js项目mygo

2.启动文件index.js

引入coffeescript框架和后续的启动文件：

update.js

3.启动检查文件update.js

3.1 初始化配置和引入公共库

引入公用库：

request 用于发起客户端http请求

cheerio 用于服务器端模拟dom

引入配置文件

Config.json

常量：

Baseurl:

<http://qa.developer.163.com/component/server>

应该是内部的测试服务器

通过config文件里的check\_update进行判断，如果需要更新的话，就执行

check\_update

通过request库发送本地请求，到服务器的/update目录下获取更新的文件列表，获取一个文件list，再根据list一个个的request请求文件，获得的文件同步写（覆盖）到当前目录下。当所有待更新文件都写完了，调用开始程序：

run\_server：

调用/lib/server.coffee进行启动：

4.核心启动文件/lib/server.coffe

使用connect中间件框架，注册中间件顺序如下：

1.proxy(CONFIG.proxy)

2.controller(\_\_dirname)

3.一个匿名函数

4.require('./static')((CONFIG.domains ? tools.domain\_root : CONFIG.dir), {

index: CONFIG.index

})

5.require('./directory')(CONFIG.domains ? tools.domain\_root : CONFIG.dir)

然后exports两个方法：

1. run

使用http模块的createServer创建服务

Server.on(“error”)注册错误事件

1. close关闭server实例

5. 中间件：

1. proxy.js

一个代理服务

首先配置config.json中的proxy

Key是你的代理服务器地址，value是你想要的结果。

作用就是：你的代理服务器访问（或者设置代理服务器的host）访问gooda时候，gooda会作为踏板访问你设置的value地址。

实际就是：

通过request(value)获得了一个读，写流

然后往这个流里pipe request请求，这个流获得的返回流再pipe到response中。

Req.pipe(request(value)).pipe(res);

1. controller.js
2. 通过全局变量\_\_dirname获得controller.js的目录，然后遍历目录，找到所有后缀是.controller.coffee的文件。
3. 设置url\_map对象，把第一步找到的所有文件通过文件名变量的方式require进map对象：

url\_map = {

Js.controller: require(“js.controller.coffee”)

}

当前就三个：

Csslint.controller.js

引入cssLint模块进行css语法检查，

1）添加3个自定义校验规则：

类名里不用大写字母、下划线和描述颜色的字母。

2）处理请求方法：

判断，如果没有csslint模块或者没有refer头或者是ie的话，就不校验。

根据url读取文件，保存到变量

调用csslint.verify根据ruleset和读取的文件进行检查。

把报错的message信息返回到页面。

jslint.controller.js

引入jslint模块进行js语法检查

当js长度小于50000并且平均每行不超过120个字符时候进行检查：

jshint(content, jsRules, globalVars);

content是内容，jsRules是检查语法，globalVars是全局变量

node.controller.js

url方法：直接获得请求url的后缀，如果是.node，返回true。

处理请求方法：

.node文件的实际处理文件是同名的无后缀文件，如果没有的话，抛出错误。

如果有的话，先delete require.cache[path.resolve]清空缓存，然后再require一次，然后该文件中对应url的method小写方法进行处理。

作用的话，就是一个自定义的中间件处理方法？

1. 定义该中间件的回调函数：

遍历url\_map的value（也就是require的上边三个文件），执行.url方法，如果true，

那么就调用该controller处理请求。

1. 匿名函数中间件：

1）首先设置跨域返回头：

（跨域的问题就是，浏览器发出请求了，也收到请求，但如果请求方的域名和当前浏览的域名不一致的话，浏览器会把接受的请求扔了）

res.setHeader("Access-Control-Allow-Origin", "\*");

res.setHeader("Access-Control-Allow-Headers", "content-type");

res.setHeader("Access-Control-Allow-Methods", "GET,POST,DELETE,OPTIONS");

Access-Control-Allow-Origin设置一个安全的域，如果请求header的origin和Access-Control-Allow-Origin设置的一致，那么浏览器就可以接收

2）对.html,.shtml和.vm做处理，调用

**render\_local**

读取请求的文件，做如下处理

第一步

处理#parse(“url”)语法，取出url替换成对应的文件字符串

循环判断，如果有#parse匹配的话，执行下面的（这样就算下面引用的文件中有#parse也可以不漏掉的替换）

通过s.replace(//g,function() {})进行全局替换#parse（url）中的url，

1）把前缀去掉：Zcs\_123 => 123

2）把后缀是\_test.vm去掉：Zcs\_test.vm => zcs.shtml

3）组装文件地址

4）通过fs. readFileSync读取文件，替换原来的#parse(“url”)

第二步

替换变量：

处理文件中所有的

{{props.key||"defvalue"}}  
 取出请求url中的请求参数（?key=value$key1=value…）组成对象a，替换的时候判断是否a存在上边的key属性，如果存在替换a[key]否则替换defvalue

第三步，调用本地Vmrender.js处理

1. 替换[英文，数字和中文]的字符串，如果在REPLACE\_MAP里有定义，就替换为：

[链接] => ${ one.get(\"url"\) }

1. 去掉vm注释：replace(/##.+\r/g, ‘’)
2. 去掉<!--(ThePills) 和(ThePills)--> ，让这种特殊注释取消，内容显露出来
3. 替换#parse("/0080/n/0080nph\_2.vm")', fs.readFileSync(\_\_dirname + '/photo.inc'))
4. 再次解析#parse，只不过之前是替换文件，这次的根据#parse(key)中的key，匹配parse.json中的内容，如果存在key，返回parse.json的value
5. 调用macro\_replace：
6. 找到所有/#macro\s\*?\(\s\*?(\S\*?)\s\*\)([\s\S]\*?)/g的位置
7. 找到每个#macro真正的结束#end（通过exec(#if|#foreach|#end），判断位置和

计算数量的方式（if,foreach 数量加一，end数量减1，当遇到end而且数量是0的时候，就是macro的最终end）

1. 替换所有文件中调用的代码为步骤2找到的代码。
2. 最后使用npm模块velocity对文件内容进行处理。

处理之后通过setTimeout(0)的方式，先return 处理的文本，然后通过delete obj[property] 清除缓存。

**resolve\_include**

4个参数：

第一个是待处理的文件内容，第二个是待处理的文件路径 ，第三个是req请求（一样），第四个是设置的替换回调函数（只有根节点的处理不一样，有htmlhint和res.end等）。

先正则匹配include，如果满足条件/<!--#include (\S+)="(\S+)"\s\*-->/g：

len数目加1（len记录了该文件有多少个include待替换文件），获取匹配的include url内容：

1. 获得所有的include url，判断url是相对、绝对还是开始带.的路径，然后组装成本地路径。
2. 然后调用**render\_local处理该文件**

如果处理成功，说明有本地文件，如果没成功，就得通过下面方法http客户端方式获得文件：

1. 查找/meta name="cms\_id" content="(\d{4})\S\*"/ 前4个数字，作为key对应domain\_map的url
2. 如果还没有，查找include里url的前4个数字

总之就是组装一个远端地址，然后发客户端请求，获取binary格式后通过decode转码成utf-8，然后调用vmrender把这部分渲染了

然后把这个include对应的文件内容作为第一个参数，进行递归

如果没有匹配到正则，是节点的时候，直接调用父亲的cb函数。（这个cb的作用就是把父亲match的include标记替换成我传入的include文件内容参数ins，当通过len判断这个节点是最后一个兄弟节点的时候，再调用父亲的父亲的cb函数）。这个cb里是一个异步执行的处理函数，通过nextStick方法，依次压栈，等到递归结束之后，再出栈执行。这个函数会保存父亲的4个变量待用

文件内容：html （共同的父节点公用）

匹配的includeurl：match（每个不同）

替换include函数cb，参数就是要替换的include文件内容：cb

父节点的include数目：len

等所有的处理完成后再顺序出栈执行。这个cb的作用就是把父亲match的include标记替换成我传入的参数ins。也就是把父亲看做一个待处理的include文件，然后按照入栈的顺序执行。

4. static.js中间件：

设置文件maxage时间，默认0

设置root等变量。

核心是send.js中的pipe方法：

会对路径做一系列处理：

重新设置

.

..

相对，绝对路径等内容，如果对应请求路径的文件能找到，那么就做如下操作：

1. 设置响应头

'Accept-Ranges' ： 'bytes

该响应头表明服务器支持Range请求,以及服务器所支持的单位是字节

Etag请求文件的md5值

Date 请求发送的日期时间

Cache-Control', 'public, max-age=设置的maxage时间

Last-Modified 文件的修改日期

Content-type 通过mime模块设置content-type

判断是否有range标签，根据range标签的开始和结束位置读文件pipe给res响应头。

5. directory.js中间件：

获取请求url对应目录的所有文件，读取directory.html模板，把相关的变量替换，然后在directory.html中生成文件列表，最后返回。