最近在捣鼓一个仿简书的[开源项目](https://github.com/jiayisheji/jianshu)，从前端到后台，一战撸到底。就需要数据支持，最近mock数据，比较费劲。简书的很多数据都是后台渲染的，很难快速抓api请求数据，本人又比较懒，就想到用写个简易爬虫系统。

**项目初始化**

安装nodejs，[官网](https://nodejs.org/en/download/)， [中文网](http://nodejs.cn/)。根据自己系统安装，这里跳过，表示你已经安装了nodejs。

选择一款顺手拉风的编辑器，用来写代码。推荐webstorm最近版。

webstorm创建一个工程，起一个喜欢的名字。创建一个package.json文件，webstorm快捷创建package.json非常简单。还是用命令行创建，打开Terminal，默认当前项目根目录，npm init，一直下一步。

可以看这里[npm常用你应该懂的使用技巧](https://github.com/jiayisheji/blog/issues/5)

**主要技术栈**

* superagent 页面数据下载
* cheerio 页面数据解析

这是2个npm包，我们先下载在接着继续，下载需要时间的。

npm install superagent cheerio --save

接下啦简单说说这2个是啥东西

[**superagent**](http://visionmedia.github.io/superagent/)**页面数据下载**

superagent是nodejs里一个非常方便的客户端请求代码模块，superagent是一个轻量级的，渐进式的ajax API，可读性好，学习曲线低，内部依赖nodejs原生的请求API,适用于nodejs环境下。

**请求方式**

* get (默认)
* post
* put
* delete
* head

语法：request(RequestType, RequestUrl).end(callback(**err**, **res**));

写法：

request

.get('/login')

.end(function(err, res){

// code

});

**设置Content-Type**

* application/json (默认)
* form
* json
* png
* xml
* ...

设置方式：

1.

request

.get('/login')

.set('Content-Type', 'application/json');

2.

request

.get('/login')

.type('application/json');

3.

request

.get('/login')

.accept('application/json');

以上三种方效果一样。

**设置参数**

* query
* send

**query**

设置请求参数，可以写json对象或者字符串形式。

**json对象{key,value}**

可以写多组key,value

request

.get('/login')

.query({

username: 'jiayi',

password: '123456'

});

**字符串形式key=value**

可以写多组key=value，需要用&隔开

request

.get('/login')

.query('username=jiayi&password=123456');

**sned**

设置请求参数，可以写json对象或者字符串形式。

**json对象{key,value}**

可以写多组key,value

request

.get('/login')

.sned({

username: 'jiayi',

password: '123456'

});

**字符串形式key=value**

可以写多组key=value，需要用&隔开

request

.get('/login')

.sned('username=jiayi&password=123456');

上面两种方式可以使用在一起

request

.get('/login')

.query({

id: '100'

})

.sned({

username: 'jiayi',

password: '123456'

});

**响应属性Response**

**Response text**

Response.text包含未解析前的响应内容，一般只在mime类型能够匹配text/json、x-www-form-urlencoding的情况下，默认为nodejs客户端提供，这是为了节省内存，因为当响应以文件或者图片大内容的情况下影响性能。

**Response header fields**

Response.header包含解析之后的响应头数据，键值都是node处理成小写字母形式，比如res.header('content-length')。

**Response Content-Type**

Content-Type响应头字段是一个特列，服务器提供res.type来访问它，默认res.charset是空的，如果有的化，则自动填充，例如Content-Type值为text/html;charset=utf8，则res.type为text/html；res.charset为utf8。

**Response status**

[http响应规范](http://tool.oschina.net/commons?type=5)

[**cheerio**](https://cheerio.js.org/)**页面数据解析**

cheerio是一个node的库，可以理解为一个Node.js版本的jquery，用来从网页中以 css selector取数据，使用方式和jquery基本相同。

* 相似的语法:Cheerio 包括了 jQuery 核心的子集。Cheerio 从jQuery库中去除了所有 DOM不一致性和浏览器尴尬的部分，揭示了它真正优雅的API。
* 闪电般的块:Cheerio 工作在一个非常简单，一致的DOM模型之上。解析，操作，呈送都变得难以置信的高效。基础的端到端的基准测试显示Cheerio 大约比JSDOM快八倍(8x)。
* 巨灵活: Cheerio 封装了兼容的htmlparser。Cheerio 几乎能够解析任何的 HTML 和 XML document。

需要先loading一个需要加载html文档，后面就可以jQuery一样使用操作页面了。

const cheerio = require('cheerio');

const $ = cheerio.load('<ul id="fruits">...</ul>');

$('#fruits').addClass('newClass');

基本所有选择器基本和jQuery一样，就不一一列举。具体怎么使用看[官网](https://cheerio.js.org/)。

上面已经基本把我们要用到东西有了基本的了解了，我们用到比较简单，接下来就开始写代码了，爬数据了哦。

**抓取首页文章列表20条数据**

根目录创建一个app.js文件。

**实现思路步骤**

1. 引入依赖
2. 定义一个地址
3. 发起请求
4. 页面数据解析
5. 分析页面数据
6. 生成数据

**1. 引入依赖：**

const superagent = require('superagent');

const cheerio = require('cheerio');

**2. 定义一个地址**

const reptileUrl = "http://www.jianshu.com/";

**3. 发起请求**

superagent.get(reptileUrl).end(function (err, res) {

// 抛错拦截

if(err){

return throw Error(err);

}

// 等待 code

});

这个时候我们会向简书首页发一个请求，只要不抛错，走if，那么就可以继续往下看了。

**4. 页面数据解析**

superagent.get(reptileUrl).end(function (err, res) {

// 抛错拦截

if(err){

return throw Error(err);

}

/\*\*

\* res.text 包含未解析前的响应内容

\* 我们通过cheerio的load方法解析整个文档，就是html页面所有内容，可以通过console.log($.html());在控制台查看

\*/

let $ = cheerio.load(res.text);

});

注释已经说明这行代码的意思，就不在说明了。就下了就比较难了。

**5. 分析页面数据**

你需在浏览器打开[简书官网](http://www.jianshu.com/)，简书是后台渲染部分可见的数据，后续数据是通过ajax请求，使用js填充。我们爬数据，一般只能爬到后台渲染的部分，js渲染的是爬不到，如果ajax，你可以直接去爬api接口，那个日后再说。

言归正传，简书首页文章列表，默认会加载20条数据，这个已经够我用了，你每次刷新，如果有更新就会更新，最新的永远在最上面。

这20条数据存在页面一个类叫.note-list的ul里面，每条数据就是一个li，ul父级有一个id叫list-container，学过html的都知道id是唯一，保证不出错，我选择id往下查找。

$('#list-container .note-list li')

上面就是cheerio帮我们获取到说有需要的文章列表的li，是不是和jq写一样。我要获取li里面内容就需要遍历 Element.each(function(i, elem) {}) 也是和jq一样

$('#list-container .note-list li').each(function(i, elem) {

// 拿到当前li标签下所有的内容，开始干活了

});

以上都比较简单，复杂的是下面的，数据结构。我们需要怎么拼装数据，我大致看了一下页面，根据经验总结了一个结构，还算靠谱。

{

id： 每条文章id

slug：每条文章访问的id （加密的id）

title： 标题

abstract： 描述

thumbnails： 缩略图 （如果文章有图，就会抓第一张，如果没有图就没有这个字段）

collection\_tag：文集分类标签

reads\_count： 阅读计数

comments\_count： 评论计数

likes\_count：喜欢计数

author： { 作者信息

id：没有找到

slug： 每个用户访问的id （加密的id）

avatar：会员头像

nickname：会员昵称（注册填的那个）

sharedTime：发布日期

}

}

基本数据结构有了，先定义一个数组data，来存放拼装的数据，留给后面使用。

随便截取一条文章数据

<li id="note-12732916" data-note-id="12732916" class="have-img">

<a class="wrap-img" href="/p/b0ea2ac2d5c4" target="\_blank">

<img src="//upload-images.jianshu.io/upload\_images/1996705-7e00331b8f3dbc5d.jpg?imageMogr2/auto-orient/strip|imageView2/1/w/375/h/300" alt="300" />

</a>

<div class="content">

<div class="author">

<a class="avatar" target="\_blank" href="/u/652fbdd1e7b3">

<img src="//upload.jianshu.io/users/upload\_avatars/1996705/738ba2908445?imageMogr2/auto-orient/strip|imageView2/1/w/96/h/96" alt="96" />

</a> <div class="name">

<a class="blue-link" target="\_blank" href="/u/652fbdd1e7b3">xxx</a>

<span class="time" data-shared-at="2017-05-24T08:05:12+08:00"></span>

</div>

</div>

<a class="title" target="\_blank" href="/p/b0ea2ac2d5c4">xxxxxxx</a>

<p class="abstract">

xxxxxxxxx...

</p>

<div class="meta">

<a class="collection-tag" target="\_blank" href="/c/8c92f845cd4d">xxxx</a>

<a target="\_blank" href="/p/b0ea2ac2d5c4">

<i class="iconfont ic-list-read"></i> 414

</a> <a target="\_blank" href="/p/b0ea2ac2d5c4#comments">

<i class="iconfont ic-list-comments"></i> 2

</a> <span><i class="iconfont ic-list-like"></i> 16</span>

<span><i class="iconfont ic-list-money"></i> 1</span>

</div>

</div>

</li>

我们就拿定义的数据结构和实际的页面dom去一一比对，去获取我们想要的数据。

id： 每条文章id

li上有一个 data-note-id="12732916"这个东西就是文章的id，  
怎么获取：$(elem).attr('data-note-id')，这样就完事了

slug：每条文章访问的id （加密的id）

如果你点文章标题，或者带缩略图的位置，都会跳转一个新页面 http://www.jianshu.com/p/xxxxxx 这样的格式。标题是一个a链接，链接上有一个href属性，里面有一段 /p/xxxxxx 这样的 /p/是文章详情一个标识，xxxxxx是标识哪片文章。而我们slug就是这个xxxxxx，就需要处理一下。$(elem).find('.title').attr('href').replace(//p//, "")，这样就可以得到xxxxxx了。

title： 标题

这个简单，$(elem).find('.title').text()就好了。

abstract： 描述

这个简单，$(elem).find('.abstract').text()就好了。

thumbnails： 缩略图 （如果文章有图，就会抓第一张，如果没有图就没有这个字段）

这个存在.wrap-img这a标签里面img里，如果没有就不显示，$(elem).find('.wrap-img img').attr('src')，如果取不到就是一个undefined，那正合我意。

下面4个都在.meta的div里面 （我没有去打赏的数据，因为我不需要这个数据）

collection\_tag：文集分类标签

有对应的class，$(elem).find('.collection-tag').text()

reads\_count： 阅读计数

这个就比较麻烦了，它的结构是这样的

<a target="\_blank" href="/p/b0ea2ac2d5c4">

<i class="iconfont ic-list-read"></i> 414

</a>

还要有一个字体图标的class可以使用，不然还真不好玩，那需要怎么获取了，$(elem).find('.ic-list-read').parent().text()，先去查找这个字体图标i标签，然后去找它的父级a标签，获取里面text文本，标签就不被获取了，只剩下数字。

接下来2个一样处理的。

comments\_count： 评论计数

$(elem).find('.ic-list-comments').parent().text()

likes\_count：喜欢计数

$(elem).find('.ic-list-like').parent().text()

接来就是会员信息，全部都在.author这个div里面

id：没有找到

slug： 每个用户访问的id （加密的id）

这个处理方式和文章slug一样，$(elem).find('.avatar').attr('href').replace(//u//, "")，唯一不同的需要吧p换成u。

avatar：会员头像

$(elem).find('.avatar img').attr('src')

nickname：会员昵称（注册填的那个）

昵称存在一个叫.blue-link标签里面，$(elem).find('.blue-link').text()

sharedTime：发布日期

这个发布日期，你看到页面是个性化时间，xx小时前啥的，如果直接取就是一个坑爹的事了，在.time的span上有一个data-shared-at="2017-05-24T08:05:12+08:00"这个才是正真的时间，你会发现它一上来是空的，是js来格式化的。$(elem).find('.time').attr('data-shared-at')

以上就是所有字段来源的。接下来要说一个坑爹的事，text()获取出来的，有回车符/n和空格符/s。所以需要写一个方法把它们去掉。

function replaceText(text){

return text.replace(/\n/g, "").replace(/\s/g, "");

}

组装起来的数据代码：

let data = [];

// 下面就是和jQuery一样获取元素，遍历，组装我们需要数据，添加到数组里面

$('#list-container .note-list li').each(function(i, elem) {

let \_this = $(elem);

data.push({

id: \_this.attr('data-note-id'),

slug: \_this.find('.title').attr('href').replace(/\/p\//, ""),

author: {

slug: \_this.find('.avatar').attr('href').replace(/\/u\//, ""),

avatar: \_this.find('.avatar img').attr('src'),

nickname: replaceText(\_this.find('.blue-link').text()),

sharedTime: \_this.find('.time').attr('data-shared-at')

},

title: replaceText(\_this.find('.title').text()),

abstract: replaceText(\_this.find('.abstract').text()),

thumbnails: \_this.find('.wrap-img img').attr('src'),

collection\_tag: replaceText(\_this.find('.collection-tag').text()),

reads\_count: replaceText(\_this.find('.ic-list-read').parent().text()) \* 1,

comments\_count: replaceText(\_this.find('.ic-list-comments').parent().text()) \* 1,

likes\_count: replaceText(\_this.find('.ic-list-like').parent().text()) \* 1

});

});

let \_this = $(elem); 先把$(elem);存到一个变量里面，jq写习惯了。

有几个\*1是吧数字字符串转成数字，js小技巧，不解释。

**6. 生成数据**

数据已经可以获取了，都存在data这个数据里面，现在是20条数据，我们理想的数据，那么放在node里面，我们还是拿不到，怎么办，一个存在数据库（还没有弄到哪里，我都还没有想好怎么建数据库表设计），一个就存在本地json文件。

那就存在本地json文件。nodejs是一个服务端语言，就说可以访问本地磁盘，添加文件和访问文件。需要引入nodejs内置的包fs。

const fs = require('fs');

它的其他用法不解释了，只说一个创建一个文件，并且在里面写内容

这是写文件的方法：

fs.writeFile(filename,data,[options],callback);

/\*\*

\* filename, 必选参数，文件名

\* data, 写入的数据，可以字符或一个Buffer对象

\* [options],flag 默认‘2’,mode(权限) 默认‘0o666’,encoding 默认‘utf8’

\* callback 回调函数，回调函数只包含错误信息参数(err)，在写入失败时返回。

\*/

我们需要这样来写了：

// 写入数据, 文件不存在会自动创建

fs.writeFile(\_\_dirname + '/data/article.json', JSON.stringify({

status: 0,

data: data

}), function (err) {

if (err) throw err;

console.log('写入完成');

});

注意事项

1. 我方便管理数据，放在data文件夹，如果你也是这样，记得一定先要在根目录建一个data文件夹不然就会报错
2. 默认utf-8编码;
3. 写json文件一定要JSON.stringify()处理，不然就是[object Object]这货了。
4. 如果是文件名可以直接article.json会自动生成到当前项目根目录里，如果要放到某个文件里，例如data，一定要加上\_\_dirname + '/data/article.json'。千万不能写成3. 如果是文件名可以直接article.json会自动生成到当前项目根目录里，如果要放到某个文件里，例如data，一定要加上\_\_dirname + '/data/article.json'。千万不能写成'/data/article.json'不然就会抛错，找不到文件夹，因为文件夹在你所在的项目的盘符里。例如G:/data/article.json。

以上基本就完成一个列表页面的抓取。看下完整代码：

/\*\*

\* 获取依赖

\* @type {\*}

\*/

const superagent = require('superagent');

const cheerio = require('cheerio');

const fs = require('fs');

/\*\*

\* 定义请求地址

\* @type {\*}

\*/

const reptileUrl = "http://www.jianshu.com/";

/\*\*

\* 处理空格和回车

\* @param text

\* @returns {string}

\*/

function replaceText(text) {

return text.replace(/\n/g, "").replace(/\s/g, "");

}

/\*\*

\* 核心业务

\* 发请求，解析数据，生成数据

\*/

superagent.get(reptileUrl).end(function (err, res) {

// 抛错拦截

if (err) {

return throw Error(err);

}

// 解析数据

let $ = cheerio.load(res.text);

/\*\*

\* 存放数据容器

\* @type {Array}

\*/

let data = [];

// 获取数据

$('#list-container .note-list li').each(function (i, elem) {

let \_this = $(elem);

data.push({

id: \_this.attr('data-note-id'),

slug: \_this.find('.title').attr('href').replace(/\/p\//, ""),

author: {

slug: \_this.find('.avatar').attr('href').replace(/\/u\//, ""),

avatar: \_this.find('.avatar img').attr('src'),

nickname: replaceText(\_this.find('.blue-link').text()),

sharedTime: \_this.find('.time').attr('data-shared-at')

},

title: replaceText(\_this.find('.title').text()),

abstract: replaceText(\_this.find('.abstract').text()),

thumbnails: \_this.find('.wrap-img img').attr('src'),

collection\_tag: replaceText(\_this.find('.collection-tag').text()),

reads\_count: replaceText(\_this.find('.ic-list-read').parent().text()) \* 1,

comments\_count: replaceText(\_this.find('.ic-list-comments').parent().text()) \* 1,

likes\_count: replaceText(\_this.find('.ic-list-like').parent().text()) \* 1

});

});

// 生成数据

// 写入数据, 文件不存在会自动创建

fs.writeFile(\_\_dirname + '/data/article.json', JSON.stringify({

status: 0,

data: data

}), function (err) {

if (err) throw err;

console.log('写入完成');

});

});

一个简书首页文章列表的爬虫就大工告成了，运行代码，打开Terminal运行node app.js或者node app都行。或者在package.json的scripts对象下添加一个"dev": "node app.js"，然后用webstorm的npm面板运行。

有文章列表就有对应的详情页面，后面继续讲解怎么爬详情。

**抓取首页文章列表对应的20条详情数据**

有了上面抓取文章列表的经验，接下来就好办多了，完事开头难。

**实现思路步骤**

1. 引入依赖
2. 定义一个地址
3. 发起请求
4. 页面数据解析
5. 分析页面数据
6. 生成数据

**1. 引入依赖**

这个就不用引入，在一个文件里面，因为比较简单的，代码不多，懒得分文件写。导入导出模块麻烦，人懒就这样的。

但我们需要写一个函数，来处理爬详情的方法。

function getArticle(item){

// 等待code

}

**2. 定义一个地址**

注意这个地址，是有规律的，不是随便的地址，随便点开一篇文章就可以看到地址栏，http://www.jianshu.com/p/xxxxxx， 我们定义的reptileUrl = "http://www.jianshu.com/";那么就需要拼地址了，还记得xxxxxx我们存在哪里吗，存在slug里面。请求地址：reptileUrl + 'p/' + item.slug。

**3. 发起请求**

superagent.get(reptileUrl + 'p/' + item.slug).end(function (err, res) {

// 抛错拦截

if(err){

return throw Error(err);

}

});

你懂的

**4. 页面数据解析**

superagent.get(reptileUrl + 'p/' + item.slug).end(function (err, res) {

// 抛错拦截

if(err){

return throw Error(err);

}

/\*\*

\* res.text 包含未解析前的响应内容

\* 我们通过cheerio的load方法解析整个文档，就是html页面所有内容，可以通过console.log($.html());在控制台查看

\*/

let $ = cheerio.load(res.text);

});

**5. 分析页面数据**

你可能会按上面的方法，打开一个页面，然后就去获取标签上面的class，id。我开始也在这个上面遇到一个坑，页面上有阅读 ，评论 ，喜欢 这三个数据，我一开始以为都是直接load页面就有数据，在获取时候，并没有数据，是一个空。我就奇怪，然后我就按了几次f5刷新，发现问题了，这几个数据的是页面加载完成以后才显示出来的，那么就是说这个有可能是js渲染填充的。那就说明的我写的代码没有错。

有问题要解决呀，如果是js渲染，要么会有网络加载，刷新几次，没有这个数据，那就只能存在页面里，写的内联的script标签里面了，右键查看源码，ctrl+f搜索，把阅读 ，评论 ，喜欢的数字，随便挑一个，找到了最底部data-name="page-data"的script标签里面，有一个json对象，里面有些字段，和我文章列表定义很像，就是这个。有了这个就好办了，省的我去截取一大堆操作。

解析script数据

let note = JSON.parse($('script[data-name=page-data]').text());

script里面数据

{"user\_signed\_in":false,"locale":"zh-CN","os":"windows","read\_mode":"day","read\_font":"font2","note\_show":{"is\_author":false,"is\_following\_author":false,"is\_liked\_note":false,"uuid":"7219e299-034d-4051-b995-a6a4344038ef"},"note":{"id":12741121,"slug":"b746f17a8d90","user\_id":6126137,"notebook\_id":12749292,"commentable":true,"likes\_count":59,"views\_count":2092,"public\_wordage":1300,"comments\_count":29,"author":{"total\_wordage":37289,"followers\_count":221,"total\_likes\_count":639}}}

把script里面内容都获取出来，然后用 JSON方法，字符串转对象。

接下来依旧是要定义数据结构：

article: { 文章信息

id: 文章id

slug: 每条文章访问的id （加密的id）

title: 标题

content: 正文（记得要带html标签的）

publishTime: 更新时间

wordage: 字数

views\_count: 阅读计数

comments\_count: 评论计数

likes\_count: 喜欢计数

},

author: {

id: 用户id

slug: 每个用户访问的id （加密的id）

avatar: 会员头像

nickname: 会员昵称（注册填的那个）

signature: 会员昵称签名

total\_wordage: 总字数

followers\_count: 总关注计数

total\_likes\_count: 总喜欢计数

}

还要专题分类和评论列表我没有累出来，有兴趣可以自己去看看怎么爬出来。它们是单独api接口，数据结构就不需要了。

因为有了note 这个对象很多数据都简单了，还是一个一个说明来源

article 文章信息

id: 文章id

主要信息都存在note.note里面，文章id就是note.note.id,

slug: 每条文章访问的id （加密的id）

note.note.slug

title: 标题  
所有的正文都存在.post下的.article里，那么获取title就是$('div.post').find('.article .title').text()

content: 正文（记得要带html标签的）

注意正文不是获取text文本是要获取html标签，需要用到html来获取而不是text，$('div.post').find('.article .show-content').html() 返回都是转义字符。到时候前端需要处理就会显示了。虽然我们看不懂，浏览器看得懂就行了。

publishTime: 更新时间

这时间直接显示出来了，不是个性化时间，直接取就好了$('div.post').find('.article .publish-time').text()

wordage: 字数

这个是一个标签里面<字数 1230>这样的，我们肯定不能要这样的，需要吧数字提取出来，$('div.post').find('.article .wordage').text().match(/d+/g)[0]\*1 用正则获取数字字符串，然后转成数字。

views\_count: 阅读计数

note.note.views\_count

comments\_count: 评论计数

note.note.comments\_count

likes\_count: 喜欢计数

note.note.likes\_count

author 用户信息

id: 用户id

前面的文章列表我们并没有拿到用户id，note.note发现了一个user\_id，反正不管是不是先存了再说，别空着，note.note.user\_id

slug: 每个用户访问的id （加密的id）

文章列表怎么获取，这个就怎么获取$('div.post').find('.avatar').attr('href').replace(//u//, "")

avatar: 会员头像

$('div.post').find('.avatar img').attr('src')

nickname: 会员昵称（注册填的那个）

$('div.post').find('.author .name a').text()

signature: 会员昵称签名

这个签名在上面位置了，就在文章正文下面，评论和打赏上面，有个很大关注按钮那个灰色框框里面，最先一段文字。$('div.post').find('.signature').text()

total\_wordage: 总字数

note.note.author.total\_wordage

followers\_count: 总关注计数

note.note.author.followers\_count

total\_likes\_count: 总喜欢计数

note.note.author.total\_likes\_count

有些字段命名就是从note.note这个json对象里面获取的，一开始我也不知道取什么名字。

最终拼接的数据

/\*\*

\* 存放数据容器

\* @type {Array}

\*/

let data = {

article: {

id: note.note.id,

slug: note.note.slug,

title: replaceText($post.find('.article .title').text()),

content: replaceText($post.find('.article .show-content').html()),

publishTime: replaceText($post.find('.article .publish-time').text()),

wordage: $post.find('.article .wordage').text().match(/\d+/g)[0]\*1,

views\_count: note.note.views\_count,

comments\_count: note.note.comments\_count,

likes\_count: note.note.likes\_count

},

author: {

id: note.note.user\_id,

slug: $post.find('.avatar').attr('href').replace(/\/u\//, ""),

avatar: $post.find('.avatar img').attr('src'),

nickname: replaceText($post.find('.author .name a').text()),

signature: replaceText($post.find('.signature').text()),

total\_wordage: note.note.author.total\_wordage,

followers\_count: note.note.author.followers\_count,

total\_likes\_count: note.note.author.total\_likes\_count

}

};

**6. 生成数据**

和列表生成数据基本一样，有一个区别。文件需要加一个标识，article\_+ item.slug（文章访问的id）

// 写入数据, 文件不存在会自动创建

fs.writeFile(\_\_dirname + '/data/article\_' + item.slug + '.json', JSON.stringify({

status: 0,

data: data

}), function (err) {

if (err) throw err;

console.log('写入完成');

});

基本就撸完了，看获取详情的完整代码：

function getArticle(item) {

// 拼接请求地址

let url = reptileUrl + '/p/' + item.slug;

/\*\*

\* 核心业务

\* 发请求，解析数据，生成数据

\*/

superagent.get(url).end(function (err, res) {

// 抛错拦截

if (err) {

return throw Error(err);

}

// 解析数据

let $ = cheerio.load(res.text);

// 获取容器，存放在变量里，方便获取

let $post = $('div.post');

// 获取script里的json数据

let note = JSON.parse($('script[data-name=page-data]').text());

/\*\*

\* 存放数据容器

\* @type {Array}

\*/

let data = {

article: {

id: note.note.id,

slug: note.note.slug,

title: replaceText($post.find('.article .title').text()),

content: replaceText($post.find('.article .show-content').html()),

publishTime: replaceText($post.find('.article .publish-time').text()),

wordage: $post.find('.article .wordage').text().match(/\d+/g)[0]\*1,

views\_count: note.note.views\_count,

comments\_count: note.note.comments\_count,

likes\_count: note.note.likes\_count

},

author: {

id: note.note.user\_id,

slug: $post.find('.avatar').attr('href').replace(/\/u\//, ""),

avatar: $post.find('.avatar img').attr('src'),

nickname: replaceText($post.find('.author .name a').text()),

signature: replaceText($post.find('.signature').text()),

total\_wordage: note.note.author.total\_wordage,

followers\_count: note.note.author.followers\_count,

total\_likes\_count: note.note.author.total\_likes\_count

}

};

// 生成数据

// 写入数据, 文件不存在会自动创建

fs.writeFile(\_\_dirname + '/data/article\_' + item.slug + '.json', JSON.stringify({

status: 0,

data: data

}), function (err) {

if (err) throw err;

console.log('写入完成');

});

});

}

你肯定要问了，在哪里调用了，  
在上面获取文章列表的请求end里面底部随便找个位置加上:

data.forEach(function (item) {

getArticle(item);

});

运行，你就会在data文件夹里看到21个json文件。[源文件](https://github.com/jiayisheji/jianshu/tree/master/reptile)，欢迎指正Bug。