Piazza信息提取与数据解析 技术文档

1. Piazza信息提取
2. 数据存放
3. 数据解析
4. 集成到Open edX平台

## 一、Piazza信息提取

从Piazza讨论区平台提取信息。提取的内容包括：

1. 所有的讨论记录。由于每一个讨论记录都对应一个唯一的cid，可以通过遍历所有的cid，向API发送POST请求从而获得所有讨论记录的json数据。
2. 所有标签的第一次筛选结果。先向API发送一个POST请求，从获得的json数据中提取出标签列表，根据标签列表再向API发送POST请求，获得按相应的标签筛选后的json数据，该数据作为以后实现多级筛选的基础。

提取信息的途径是通过piazza的API获得json数据。具体的实现思路如下：

首先，通过python脚本实现piazza平台的登录操作。分析浏览器登录piazza所需的请求头（header），根据分析的结果利用python脚本实现模拟登录piazza平台，并保存cookie，供以后直接访问piazza使用。

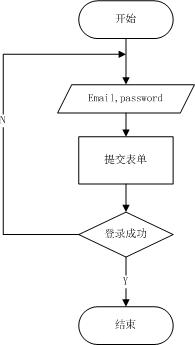
其次，通过API获得所需的json数据。利用python脚本实现模拟登录piazza平台之后，分析浏览器进行“点击某一条记录的快照，网页中显示该记录详细信息”这一行为所发送的POST请求。分析后发现：浏览器采用新的请求头向piazza的API发送POST请求，请求结果是返回json数据到浏览器端，浏览器端对json数据解析，实现讨论记录详细信息的显示。由此可以通过python脚本实现向piazza的API发送POST请求，实现json数据的获取。

最后对获得的json数据解压。

下面详细说明这三个过程。

### 1. python脚本实现piazza平台的登录操作

用流程图表示浏览器直接登录piazza以及python脚本模拟登录piazza的流程如下：



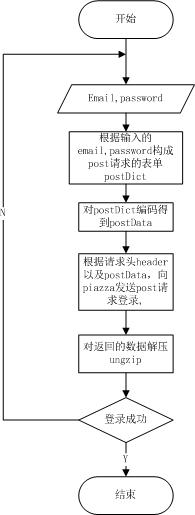


图1­-1 浏览器登录piazza 图1-2 利用python脚本登录piazza

下面对python脚本登录piazza所涉及的关键内容进行分析：

利用urllib.request, http.cookiejar两个库创建opener实现自动在发出的POST请求中加上自定义的请求头，以及自动处理使用opener过程中遇到的cookies的功能，创建opener以及利用opener发送post请求的代码如下：

//创建opener

cj = http.cookiejar.CookieJar()

pro = urllib.request.HTTPCookieProcessor(cj)

opener = urllib.request.build\_opener(pro)

urllib .request .install\_opener(opener)

//利用opener发送post请求，接收返回的数据并解压

req = urllib.request.Request (url,postData,header)

response=urllib .request .urlopen(req)

data=response.read()

data = ungzip(data)

urllib.request.Request (url,postData,header) 的参数说明：

“url”: 发送POST请求的目标url，url=” <https://piazza.com/class/>”,

“postData”: POST请求的参数，登录时所需的参数包括：email, password, from, remember四项，其中email, password即用户登录的邮箱和密码，from：“signup”, remember: ”on” 。这四项内容以json格式组合即：postDict={ 'from': '/signup','email':email ,'password': password , 'remember': 'on' }，然后要对postDict编码才能作为urllib.request.Request的参数。调用urllib.parse包中的urlencode方法对postDict编码。postData = urllib.parse.urlencode(postDict).encode()。

“header”: 自定义的请求头。Header通过分析浏览器行为获得，python脚本中发送POST请求时利用此header让服务器端认为是由浏览器发出的POST请求，以达到模拟浏览器操作的效果。

## 2. 通过API获得所需的json数据

流程图表示浏览器获得某条讨论记录的详细信息以及通过python脚本获得讨论记录详细信息：

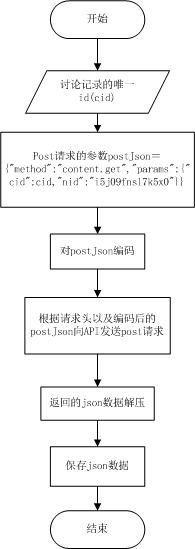


图1-3 浏览器获得某一条讨论记录详细信息流程图 图1-4 python脚本获得某一条讨论记录详细信息流程图

为了便于后续内容的理解,选对piazza页面组织结构作简要分析。如图1-5所示piazza的页面主要由三部分构成: 第一部分，导航栏。导航栏位于整个页面上方,这里列出了一些受欢迎的标签,点击标签可以实现快速筛选,筛选的结果会显示在第二部分。第二部分，讨论记录快照列表。讨论记录快照列表位于页面的左边，页面初始加载时列出的是按时间倒序排序的讨论记录快照列表，当点击导航栏处受欢迎的标签时，按标签筛选的结果会显示在这里。第三部分，讨论记录详细信息。讨论记录详细信息位于第三部分的右边，当点击讨论记录快照列表中的某一项时，第三部分会显示讨论记录的详细信息。

|  |  |
| --- | --- |
| 第一部分：导航栏 | |
| 第二部分  讨论记录快照列表 | 第三部分  讨论记录详细信息 |

图1-5 piazza页面组织结构简图

从获得某一个讨论记录详细信息的json数据出发，然后把方法推广到获得所有讨论记录详细信息数据，以及获得按标签筛选后的讨论记录快照列表的数据。

通过在浏览器的Piazza页面上进行”点击讨论记录快照列表中某一项”这一操作,分析浏览器处理这该动作所涉及到的网络请求,得到如下的信息:当点击讨论记录快照表表中某一项时,浏览器会向服务器端发送一个POST请求,请求的url为: <https://piazza.com/logic/api/>. 另外，POST请求的请求头header也和登录时不一样, 请求的参数包括两项内容”method”和”params”.参数格式为{"method":"content.get","params":{"cid":nr,"nid":"i5j09fnsl7k5x0"}}.其中,”method”:”content.get”表示的是向API请求讨论的详细内容,如果向API请求的是讨论记录快照则”method”:”network.get\_my\_feed”. ”params”是相应的参数,”cid”:表示的是点击的讨论记录快照所对应的ID, nid表示的是课程id.

接下来用Python实现上述操作。通过API向服务器端获得json数据实质上也是向服务器端发送POST请求，因此基本的代码和登录piazza时一样，只是改变了urllib.request.Request(url,postData,header\_new)中的参数的值。此时的url变为：”https://piazza.com/logic/api/”，header\_new是新的请求头可以通过浏览器行为分析中获得，postData是编码后的postJson ， postJson={"method":"content.get","params":{"cid":nr,"nid":"i5j09fnsl7k5x0"}}，对postJson编码调用的是json.dumps(postJson).encode()方法。POST请求的返回结果需要进行预处理以便于下一步的操作。预处理时需要对返回结果进行解压，解码，然后处理中文。关键的代码如下。

data是POST请求的返回结果，首先调用ungzip()对其解压，然后解码，解码后得到的数据中文不能正常显示，通过json.loads()以及json.dumps()处理中文。最后得到的newjson即为预处理后的json数据。

data = ungzip(data)

data=data.decode()

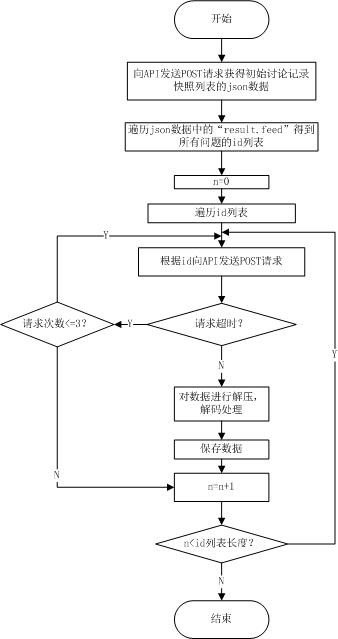
#处理data的中文

myjson=json.loads(data)

newjson=json.dumps(myjson,ensure\_ascii=False)

有了上述获得某一条指定的讨论记录json数据这一核心的方法后，就可以推而广之来获得所有的讨论记录json数据。首先需要得到一个所讨论记录的id列表，然后遍历这个列表，向API发送POST请求，获得讨论记录的详细信息数据。为了获得所有讨论记录的id列表，先向API发送POST请求获得初始时的讨论记录快照列表数据，数据中包含了所有讨论记录的快照，对数据解析可以得到id列表。另外，数据请求时有时因为网络问题请求时间太长，这里通过设定一个默认的请求时限，超过时限没有响应则显示超过再次发起本次请求，如果连续三次请求超时则跳过该请求。请求时限的设定由socket.setdefaulttimeout()实现，脚本中设定的值为20s。具体的流程见图1-5。

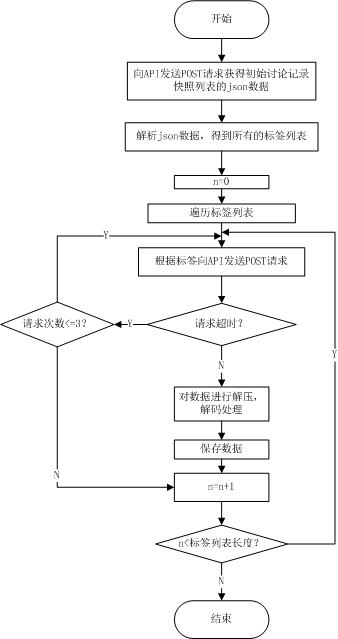
图1-5 获得所有讨论记录详细信息流程图



下面讨论获得按标签筛选后的讨论记录快照列表的json数据的方法。仍然通过分析在浏览器piazza页面进行“点击某一个受欢迎的标签列表中的标签”这一动作，浏览器的所涉及的操作。分析后得知，进行这一操作时浏览器仍然向同一个url发送post请求，只是请求时的参数不一样了。与”点击讨论记录快照列表中某一项”这一操作的区别在于postJson的构成不同。例如要获得按标签”lab1”筛选后的讨论记录快照列表，则postJson={"method":"network.filter\_feed","params":{"nid":"i5j09fnsl7k5x0","sort":"updated","filter\_folder":"lab1","folder":1}}。”filter\_folder” 的值即为标签的名称。

通过上述分析可以知道，获得json数据的实质是向“<https://piazza.com/logic/api/>”这个url发送POST请求，根据请求的参数不同，会返回不同的数据，然后对数据进行解压，解码处理得到json数据。具体的流程图如图1-6所示。

图1-6 获得所有标签筛选后讨论记录快照流程图



通过上面的方法已经得到了从piazza平台提取的数据。下面介绍的是数据的存储问题。

## 二、数据存放

这些数据都是以json字符串的形式保存的。由于数据量不是特别大，为了节省开支不需要存放在一个特点的数据库中，以文件的格式保存即可，为了便于后续在Open edX平台上的集成，以及数据的存取方便，把这些数据保存在github的一个仓库中。

## 三、json数据解析

对json数据进行解析是最重要的一部分内容。首先对所获得的json数据的结构进行分析。然后根据json数据的结构，以json数据为数据源利用js模板引擎――handlebar进行数据解析，生成js模板，并利用css渲染，最后以html页面的形式展现出来。

### json数据结构分析。

提取到的数据主要有两种：一种是讨论记录的详细信息，另一种是按标签筛选后的讨论记录快照。这两种json数据的结构存在差别，而每一种数据各自的结构是相同的。

讨论记录的详细信息。其结构如图3-1所示。

图3-1讨论记录详细信息数据结构简图

“result”：

“error”

“aid”

“tags”：

“id”：

“nr”：

“type”：

“history”：

“children”：

“subject”：

“content”：

“created”：

“ type”：”i\_answer”

“history”：

“subject”：

“content”：

“created”：

“ type”：”s\_answer”

“history”：

“subject”：

“content”：

“created”：

“ type”：”follow up”

“children”：

“ subject”：

“ created”：

“ type”：“feedback”

“created” ：

“ subject”：

一个讨论记录详细信息数据以json字符串的形式保存在文件中，进行解析的时候先把json字符串转化为json对象，每一个json对象的格式如图3-1所示，注意图3-1中只列出了有用信息的名称，从图中可以清楚的了解到json对象的格式。所需的信息存在于“result.history”以及”result.children”这两个字面量中。

Result.history是一个对象数组，这个数组又由对象字面量组成，每一个{}所代表的是一个对象字面量。对象字面量中存储的是post的内容，每当post的发表者进行修改的时候，数组中会相应的增加一个对象字面量。在进行json解析时，只需要读出数组中第一个对象字面量的内容即可，第一个对象字面量的内容就是post内容的最新版本。

Result.children是一个对象数组。它包括三种对象，这三种对象按”type”的值不同来区分，”type”的值为” i\_answer”时该对象存储的是instructor针对该post的回答。”type”的值为” s\_answer”时该对象存储的是student针对该post的回答。”type”的值为” followup”时该对象存储的是后续的讨论。当”type”的值为” followup”时，对象中又包含数组，后续的讨论中会有回复，回复的内容会存储在数组中。

按标签筛选后的讨论记录快照，其结构如图3-2所示。

“result”：

“error”：

“aid”：

“sort”：

“feed”：

“tags”：

“id”：

“nr”：

“updated”：

“modified”：

“type”：

“content\_snipet”：

“subject”：

“tags”：

“id”：

“nr”：

“modified”：

“updated”：

“subject”：

“content\_snipet”：

……

“more”：false

图3-2 按标签筛选后讨论记录快照json数据结构简图

标签筛选后讨论记录快照的json对象的格式则简单得多，每一个快照以对象数组”result.feed”中的一个对象的形式存在。每一个对象包含一系列的“名称－值”。对这些“名称”的含义作说明。”tags”的值表示的是该快照所包含的标签，是一个数组；”id”的值表示的是该快照的id，由14个字符的字符串表示；”nr”的值表示的是该快照的id，是一个整数；”modified”的值表示的是修改时间；”updated”:的值表示的是更新时间；”subject”的值表示的是快照的主题，与图3-1中“result.history.subject”的值相同；”content\_snipet”的值表示的是快照的内容，是图3-2中“result.history.content”中的一部分内容。

### handlebars模板引擎

清楚了json数据的格式之后，下面需要做的是把json数据对象的内容展示在页面上。

关于页面的展示采用的是”json+ajax+html”的方式。这种方法是基于ajax请求，要求服务器端返回一个json类型的json字符串，这个json串中包含了界面所需的所有信息，界面拿到json串后，构造出html。

完成界面展示利用javascript模板引擎可以有效的组织和分离前端页面代码中的显示层和数据层两个部分。Javascript模板引擎有很多种，本文中使用的是handlebars模板引擎。handlebars以json对象为数据源，支持逻辑判断、循环等操作，同时具有非常好的扩展性。

下面介绍利用handlebars模板引擎完成数据的展示的方法。

1. 创建html页面

①在<head>部分添加js库，包括两个js库，一个是jquery的js 库（jquery-1.11.3.min.js）；另一个是handlebars的js 库(handlebars-v3.0.3.js)。

②添加handlebars模板。

<script id=”page-center-template” type=”text/x-handlebars-template”>

　　　　　　　　　　　Template content　//对json数据的解析

</script>

1. 创建js

使用handlebars.compile方法将模板编译为函数，生成的执行函数接受json数据作为参数，用来渲染模板

//获得html中定义的模板

var source = $("#page-center-template").html();

//将模板编译为函数

var template = Handlebars.compile(source);

//生成的执行函数接受context作为参数

var html = template(data\_json);

//渲染模板

$("#page\_center").html(html);

下面以第二部分——讨论记录快照列表模板的生成为例详细介绍如何利用handlebar模板引擎生成js模板实现动态加载html页面。

1. 文件结构

　主要包括三类文件：.html，.js，.css

|  |  |
| --- | --- |
| 文件名 | 文件介绍 |
| example.html | 实现讨论记录快照列表的html页面 |
| handlebars-v3.0.3js和jquery-1.11.3.min.js | 利用handlebars模板所需的js脚本 |
| piazza\_feed.js | 对模板进行编译以及其它函数功能实现，利用ajax获取json数据的js脚本 |
| dashboard\_feed.js | 渲染html页面的css文件 |

1. 具体实现
2. 创建html页面

①在<head>部分添加js库，包括两个js库，一个是jquery的js 库（jquery-1.11.3.min.js）；另一个是handlebars的js 库(handlebars-v3.0.3.js)。

②页面框架

　<div id="page\_main" class="has\_popular\_tags\_bar">

<div class ="views" id="views">

<div id ="feed" class="page\_feed">

</div>

</div>

</div>

　　　　　　　　讨论记录快照列表的内容显示在<div id ="feed" class="page\_feed"></div>之内，具体的显示格式定义在下面的handlebars模板中。

③添加handlebars模板。

<script id=” feed-template” type=”text/x-handlebars-template”>

　　　　　　　　　　　 //template content

</script>

模板内容的编写是实现的重点部分之一，利用的是handlebars的语法规则，解析json数据，获取所需的内容使其在模板的对应位置显示。内容用{{{　}}}

获得。其中进入到属性上下文的方法为{{#with }}{{/with}}，遍历数组的方法为{{#each}}{{/each}}。例如：获得快照相对应的时间，快照的标题，快照的内容：

　　 {{#with result}}

　　　{{!-- 按组显示（遍历所有组） --}}

{{#each feed}}

　　　　　　　　{{!--日期 --}}

　　　　　　　　{{{modified}}}

　　　　　　　　{{!--标题 --}}

　　　　　　　　{{{subject}}}

　　　　　　　　{{!--内容片段 --}}

　　　　　　　　{{{content\_snipet}}}

　　　　　　{{/each}}

　　　　{{/with}}

另外，当需要进行条件判断时，需要注册helper来实现。例如定义比较大小的helper 则可以在html进行使用，使用方法如下：

{{#compare @index 0}}

….

{{/compare}}

可以判断数组的下标是否为0,若为0则执行相应的操作。

该比较大小的helper的注册在js脚本中实现，实现方法为：

//注册一个比较大小的Helper,判断v1是否等于于v2

Handlebars.registerHelper("compare",function(v1,v2,options){

if(v1==v2){

//满足添加继续执行

return options.fn(this);

}

else{

//不满足条件执行{{else}}部分

return options.inverse(this);

}

});

1. 创建js

页面加载时的初始化操作包括：导航栏显示“popular tags”以及生成多级筛选的第一级下拉菜单；讨论记录快照列表部分显示的是按时间倒序排列的快照列表；注册一个比较大小的helper。前两项内容的显示都涉及到数据的动态获取以及显示的问题，通过ajax实现从github获得json数据，然后利用handlebar模板引擎实现json数据的解析与显示。

当在前台的页面进行相应的操作时，页面需要根据操作进行响应，通过js中的函数实现对页面的响应。下面显示的是函数的调用关系图。

