# 第十四章 Django博客开发-分页与vue分页

本节所讲内容：

14.1 分页介绍与逻辑

14.2 vue介绍

14.3 基于vue尝试分页

14.4 基于vue-resource尝试分页

## 14.1 分页介绍与逻辑

由于当前开发的需求当中各种管理平台的增多，各种各样的列表需求逐渐增多，而往往数据量有很多，很难实现也没有必要将所有的数据都罗列到一个前端页面上，用户用不到，也很浪费资源。所以我们有了分页的需求，将数据分页显示。

来看这样的一个例子

现在有1000条文章的数据需要展示在我们的计算机上。

1、可以讲1000条分为每10条一页，这样就形成了分页。看一下分页的逻辑

|  |  |
| --- | --- |
| 页码 | 数据 |
| 1 | 0 - 9 |
| 2 | 10 - 19 |
| 3 | 20 - 29 |
| …… | …… |
| n | （n-1）\*10 - n\*10-1 |

那么这种分页，需要我们定义当前页码还有单页数据长度。

2、这样的分页，每次请求，都需要对1000条数据进行遍历，也许浏览者只需要第20条数据，所以，我们是不是可以在上面的基础上，对使用者进行”欺诈”呢？

|  |  |  |
| --- | --- | --- |
| 分页码 | 页码 | 数据 |
| 0-29 | 1 | 0 - 9 |
| 2 | 10 - 19 |
| 3 | 20 - 29 |
| 30-59 | 4 | 30 - 39 |
| 5 | 40 - 49 |
| 6 | 50 - 59 |

这种逻辑，当用户请求数据，先不去查询所有数据，而是查询前30条数据，然后，对这30条数据进行分页，如果有人查询到第4页，我们就接着加载。

到这里，分页的逻辑并没有结束，还需要想明白前端除了分页的数据，还需要什么：

1. 分页的数据
2. 当前的页面 用来搭建上一页和下一页
3. 页码的序列

14.2 pure\_pagination 实现分页回顾  
进行分页数据查询，将接下来进行的就是我们的myArticle.html中的分页功能。

到这里我们可以使用django的分页模块Paginator进行数据查询分页，但是这个模块不太强大。

pip install django-pure-pagination==0.3.0

在使用之前先对paginator对象进行一下讲解，paginator通常使用两个参数进行分页，要分页的数据队列，和单页数据的个数。我们采用这两个参数形成分页对象

然后通过各个方法来查看分页的结果，我们常用的方法如下

总页数：paginator.num\_pages

总记录数： paginator.count

分页范围: paginator.page\_range, 比如 [1, 2, 3, 4]

分页数据(list)： page.object\_list（列表中是一个个对象）

是否有下页：page.has\_next(), True/False

是否有上页：page.has\_previous() previous [ˈpriːviəs]:先前的

下一页的页码：page.next\_page\_number()

上一页的页码：page.previous\_page\_number()

当前页第一个对象： page.start\_index()

当前页最后一个对象: page.end\_index()完成前端分页效果

获取页码参数：page.querystring 实现效果page=1

到这里我们开始进行分页设置

在Article/view.py中编写如下：

from django.http import HttpResponse  
from django.shortcuts import render\_to\_response  
from pure\_pagination import Paginator,PageNotAnInteger

#......

def myArticle(request):  
 *#获取所有后端的文章* articles = Article.objects.all()  
 *#获取页码，保证起始页码为1* try:  
 page = request.GET.get('page',1)  
 except PageNotAnInteger:  
 page = 1  
 *#实现分页，第一个参数是所有文章，第二个参数是每页文章个数* paginator = Paginator(articles,2)  
 *#获取分页数据，交给前端渲染* pageData = paginator.page(page)  
 return render(request,'myArticle.html',{'pageData':pageData})

前端调用参数（articleList.html部分分页代码，全部代码请文档下附录看articleList.html）

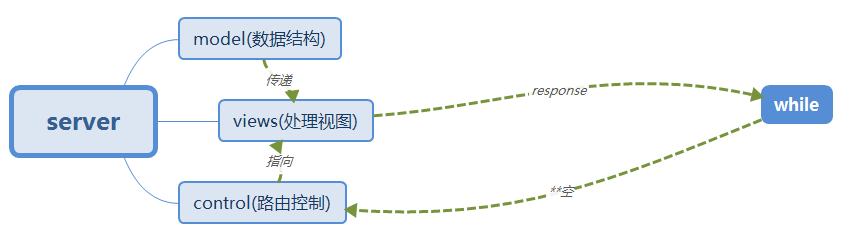
<div class="pageturn">  
 <ul class="pagelist" >  
 {% if pageData.has\_previous %}  
 <li class="long"><a href="?{{ pageData.previous\_page\_number.querystring }}">上一页</a></li>  
 {% endif %}  
  
 {% for page in pageData.pages %}  
 {% if page %}  
 {% ifequal page pageData.number %}  
 <li class="active"><a href="?{{ page.querystring }}">{{ page }}</a></li>  
 {% else %}  
 <li><a href="?{{ page.querystring }}" class="page">{{ page }}</a></li>  
 {% endifequal %}  
 {% else %}  
 <li class="none"><a href="">...</a></li>  
 {% endif %}  
 {% endfor %}  
 {% if pageData.has\_next %}  
 <li class="long"><a href="?{{ pageData.next\_page\_number.querystring }}">下一页</a></li>  
 {% endif %}  
 </ul>  
</div>

## 14.2 vue介绍

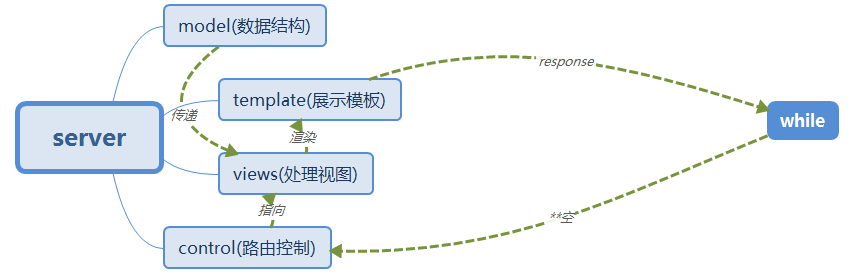
我们今天主要用ajax+vue实现分页的动态加载。

当然在这里先需要给大家介绍一下我们的vue.js, 是一套构建用户界面的渐进式框架。只关注视图层的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

MVC (Model – View - Control)



MTV (Model – Template - Views)



Vue.js采用的是MVVM(ModelView-viewModel)

强调前端的双向绑定逻辑



通俗的讲，vue.js可以将我们从后端传递过来的json数据传递到HTML当中，进行动态的更新。

首先可以从vue的官网（https://vuejs.org/）上下载vue的脚本，在这里，我们使用已经下载好的文件，将它们放入static下面。<https://vuejs.org/v2/guide/> 使用说明

新建一个vuejsExample.html编写如下代码

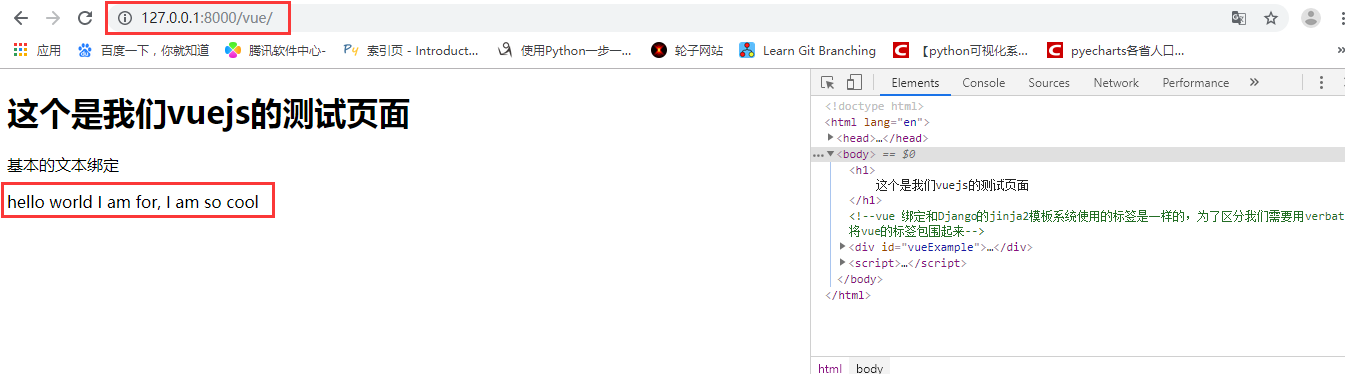
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>vuejsExample</title>  
 <script src = "/static/js/jquery.js"></script>  
 <script src = "/static/js/vue.min.js"></script>  
</head>  
<body>  
<h1>  
 这个是我们vuejs的测试页面  
</h1>  
*<!--vue 绑定和Django的jinja2模板系统使用的标签是一样的，为了区分我们需要用verbatim将vue的标签包围起来-->  
{#vue的渲染语法和Django默认的jinja2的渲染语法类似。直接使用会导致歧义。所以，我们在使用vue绑定数据之前，必须先讲绑定的部分用{% verbatim myblock %}% endverbatim myblock %}标签包围起来。#}*{% verbatim myblock %}  
 <div id = "vueExample">  
 <p> 基本的文本绑定 </p>  
 <p> hello world I am {{ name }}</p>  
 </div>  
{% endverbatim myblock %}  
*<!--vue对象在HTML当中调用，在js当中进行数据的声明，其中有一些固定的点和结构需要记忆  
首先需要声明vue对象  
指定绑定的HTML标签，选择器是jq选择器，这里用#veuExample来绑定  
具体的数据都放在一个data的地方-->*<script>  
 new Vue( *//实例化vue对象* {  
 el: "#vueExample", *//绑定的元素是一个id为vueExample的div* data:{  
 name: "for, I am so cool"  
 } *//可供绑定的数据设置* }  
 )  
</script>  
</body>  
</html>

然后编写Article/views.py

def vuetest(request):  
 return render(request,'vuejsexample.html')

**配置路由  
 url(r'vue/',vuetest),**

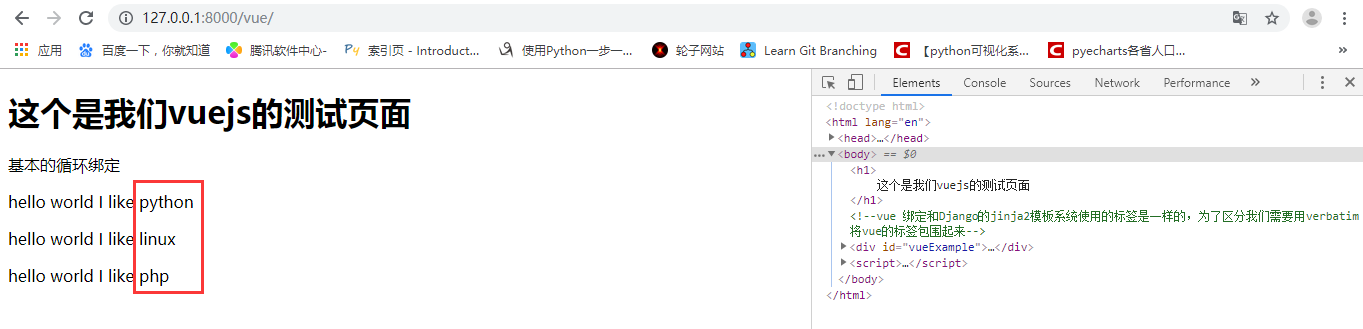
效果如下：



上面是一个最简单的文本绑定的例子，我们接下来再看几种绑定

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>vuejsExample</title>  
 <script src = "/static/js/jquery.js"></script>  
 <script src = "/static/js/vue.min.js"></script>  
</head>  
<body>  
<h1>  
 这个是我们vuejs的测试页面  
</h1>  
*<!--vue 绑定和Django的jinja2模板系统使用的标签是一样的，为了区分我们需要用verbatim将vue的标签包围起来-->*{% verbatim myblock %}  
 <div id = "vueExample">  
 <p> 基本的循环绑定 </p>  
 <p v-for = "p in project"> hello world I like {{ p }}</p> *<!--关注循环体-->* </div>  
{% endverbatim myblock %}  
<script>  
 new Vue( *//实例化vue对象* {  
 el: "#vueExample", *//绑定的元素是一个id为vueExample的div* data:{  
 name: "for, I am so cool",  
 project: ["python", "linux", "php"]  
 } *//可供绑定的数据设置* }  
 )  
</script>  
</body>  
</html>

效果如下：



带有属性和映射的循环绑定

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>vuejsExample</title>  
 <script src = "/static/js/jquery.js"></script>  
 <script src = "/static/js/vue.min.js"></script>  
</head>  
<body>  
<h1>  
 这个是我们vuejs的测试页面  
</h1>  
*<!--vue 绑定和Django的jinja2模板系统使用的标签是一样的，为了区分我们需要用verbatim将vue的标签包围起来-->*{% verbatim myblock %}  
 <div id = "vueExample">  
 <p> 带有属性和映射的循环绑定 </p>  
 <ul>  
 <li v-for = "p in projects">  
 <span v-bind:id = "p.color">  
 {{ p.name }}  
 </span>  
 </li>  
 </ul>  
 </div>  
{% endverbatim myblock %}  
<script>  
 new Vue( *//实例化vue对象* {  
 el: "#vueExample", *//绑定的元素是一个id为vueExample的div* data:{  
 name: "for, I am so cool",  
 project: ["python", "linux", "php"],  
 projects:[  
 {name: "python", color: "red"},  
 {name: "linux", color: "green"},  
 {name: "php", color: "blue"}  
 ]  
 } *//可供绑定的数据设置* }  
 )  
</script>  
</body>  
</html>

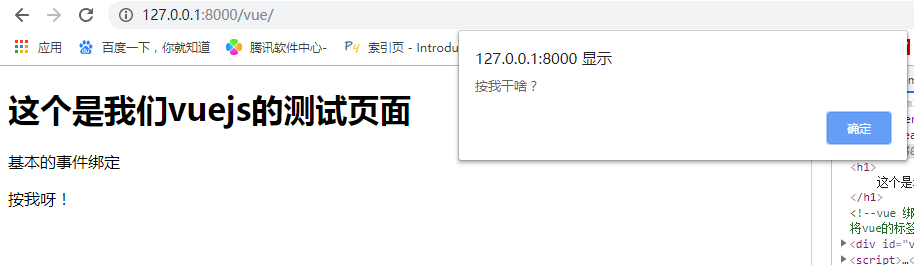
效果如下：



基本的事件绑定

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>vuejsExample</title>  
 <script src = "/static/js/jquery.js"></script>  
 <script src = "/static/js/vue.min.js"></script>  
</head>  
<body>  
<h1>  
 这个是我们vuejs的测试页面  
</h1>  
*<!--vue 绑定和Django的jinja2模板系统使用的标签是一样的，为了区分我们需要用verbatim将vue的标签包围起来-->*{% verbatim myblock %}  
 <div id = "vueExample">  
 <p> 基本的事件绑定 </p>  
 <span v-on:click = "alertthing">按我呀！</span>  
 </div>  
{% endverbatim myblock %}  
<script>  
 new Vue( *//实例化vue对象* {  
 el: "#vueExample", *//绑定的元素是一个id为vueExample的div* data:{  
 name: "while, I am so cool",  
 project: ["python", "linux", "php"],  
 projects:[  
 {name: "python", color: "red"},  
 {name: "linux", color: "green"},  
 {name: "php", color: "blue"}  
 ]  
 }, *//可供绑定的数据设置* methods: {  
 alertthing: function () {  
 *alert*("按我干啥？")  
 }  
 }  
 }  
 )  
</script>  
</body>  
</html>

效果如下



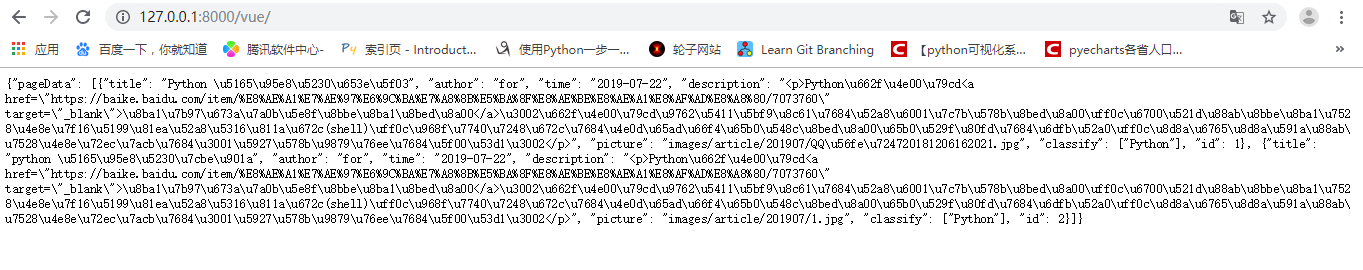
## 14.3 基于vue尝试分页

说到vue.js我们听过最多的就是前后端分离，既然是前后端分离，那服务器往往响应的就是一段json文本数据。如果我们想要基于vue做分页处理，后端编写json响应数据，前端编写ajax请求处理

Vue后台数据返回Aritcles/views.py

def vuetest(request):  
 *"""  
 提供分页的ajax数据  
 """* if request.method == "GET": *#如果是get请求* try:  
 page = request.GET.get('page',1)  
 except:  
 page = 1  
 aritcles = Article.objects.all() *#查询所有的数据* paginator = Paginator(aritcles,2) *#对数据进行分页，每页三条* pageData = paginator.page(page) *#获取具体页的数据* print(pageData)  
 page\_data = [] *#对数据进行json结构化，json只接受字典对象[{},{}]* for data in pageData.object\_list:  
 classify = data.classify.all() *#多对多字段需要首先查询出所有对应的字段,查询出来还是数据库对象* if classify:  
 classify = [i.label for i in classify] *#对字段取指定的lable* else:  
 classify = "" *#空类表不可以建json序列，所以，我们改完字符串* page\_data.append(  
 {  
 "title": data.title,  
 "author": data.author.name, *#w外键，必须调用具体的字段* "time": data.time,  
 "description": data.description,  
 "picture": data.picture.name, *#这里的name是由于文件对象有name属性* "classify": classify,  
 "id": data.id  
 }  
 )  
 result = {  
 "pageData": page\_data  
 }  
 return JsonResponse(result)

效果如下：



Html前端进行调用template/myArticle.html

{% extends "base.html" %}  
  
{% block title %}  
 我的博客  
{% endblock %}  
  
{% block style %}  
 <link href="/static/css/main.css" rel="stylesheet">  
 <script src="/static/js/jquery.js"></script>  
 <script src="/static/js/vue.min.js"></script>  
{% endblock %}  
  
{% block content %}  
 <div class="container">  
 <div class="con\_content">  
 <div class="about\_box">  
 <h2 class="nh1"><span>您现在的位置是：<a href="/" target="\_blank">网站首页</a>>><a href="#" target="\_blank">个人日记</a></span><b>个人日记</b></h2>  
 {% verbatim myblock %}  
 <div class="dtxw box" id = "dataList">  
 <li v-for = "page in page\_data">  
 <div class="dttext f\_l">  
 <ul>  
 <h2><a href="/">{{ page.title }}</a></h2>  
 <p v-html="page.description">{{ page.description }}</p> *<!-- 注意和Django的safe标签的区分 v-html == safe -->* <span>{{ page.time }}</span>  
 </ul>  
 </div>  
 <div class="xwpic f\_r"><a href="/"><img v-bind:src="['/static/'+page.picture]"></a></div> *<!-- 注意绑定数据的拼接 -->* </li>  
 </div>  
 {% endverbatim myblock %}  
  
 <div class="pagelist">页次：1/1 每页25 总数10<a href="/">首页</a><a href="/">上一页</a><a href="/">下一页</a><a href="/">尾页</a></div>  
 </div>  
 </div>  
 <div class="blank"></div>  
 *<!-- container代码 结束 -->* </div>  
 <script>  
 $(  
 function () {  
 $.ajax(  
 {  
 url: "/vue/",  
 type: "GET",  
 data: "",  
 dataType: "json", *//我们需要抢到我们的ajax请求需求数据是json* success: function (data) {  
 new Vue( *//注意new地下和var* {  
 el:"#dataList", *//绑定指定元素，注意选择器，注意#* data: {  
 page\_data: data["pageData"] }  
 }  
 );  
 },  
 error: function (error) {  
 *console*.log(error)  
 }  
 }  
 )  
 }  
 );  
 </script>  
{% endblock %}

运行结果如下：



## 14.4 基于vue-resource尝试分页

上面操作我们已经实现vue逻辑的前端展示，更加符合我们项目中的前后端分离，但是有个明星的缺点就是我们无非实现分页。所以说要想实现前后端分离的分页，你需要另写个逻辑方便处理

14.4.1 分页逻辑展示

这样的分页，每次请求，都需要对1000条数据进行遍历，也许浏览者只需要第20条数据，所以，我们是不是可以在上面的基础上，对使用者进行”欺诈”呢？

|  |  |  |
| --- | --- | --- |
| 分页总数据 | 页码 | 数据 |
| 0-29 | 1 | 0 - 9 |
| 2 | 10 - 19 |
| 3 | 20 - 29 |
| 30-59 | 4 | 30 - 39 |
| 5 | 40 - 49 |
| 6 | 50 - 59 |

这种逻辑，当用户请求数据，先不去查询所有数据，而是查询前30条数据，然后对这30条数据进行分页，如果有人查询到第4页，我们就接着加载。

先写个逻辑方便理解

每次请求完毕 查询3页

每页 查询3条

首先完成一条逻辑

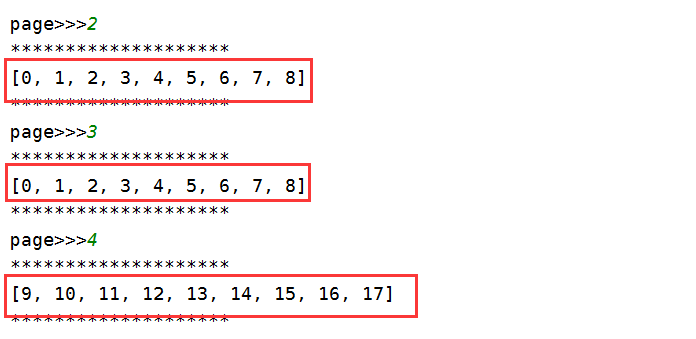
1,2,3 🡪 1

4,5,6 🡪 2

为了让大家更好理解，请看如下代码实现

*#!/usr/bin/python  
# -\*- coding: utf-8 -\*-  
# @Time : 2019/7/24 9:38  
# @Author : Xuegod Teacher For  
# @File : page\_test.py  
# @Software: PyCharm*datas = list(range(100))  
one\_page\_num = 3*#每页显示3条数据*one\_time\_num = 3*#每次查询分为3页*while True:  
 page = int(input('page>>>'))  
 *#page =1 findindex = 1 int(1/3)+1  
 #page =2 findindex = 1 int(2/3)+1  
 #page =3 findindex = 1 int(3/3)  
 #page =4 findindex = 2 int(4/3)+1  
 #page =5 findindex = 2  
 #page =6 findindex = 2  
 #判断大于* if page/ one\_time\_num > int(page/ one\_time\_num):  
 *#生成查询页码* find\_index = int(page/ one\_time\_num)+1  
 else:  
 *#当等于的时候我们就直接输出查询页码* find\_index = int(page/ one\_time\_num)  
 *#每次请求vue这个url查询的总数据#9 个* select\_num = one\_page\_num \* one\_time\_num  
 *#数据起始下标* start = (find\_index-1)\*select\_num  
 *#数据结束下标* end = find\_index\*select\_num  
 *#截取数据* data = datas[start:end]  
 *#处理数据* print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 print(data)  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

运行结果如下：

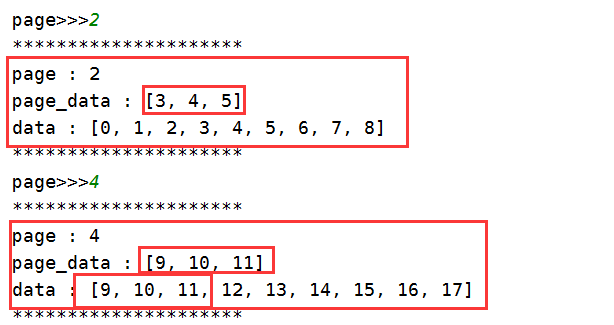


当数据超出3页的范围就会重新加载新的9条数据

进一步操作，我们实现每个页码中对应的3条数据

*#!/usr/bin/python  
# -\*- coding: utf-8 -\*-  
# @Time : 2019/7/24 9:38  
# @Author : Xuegod Teacher For  
# @File : page\_test.py  
# @Software: PyCharm*datas = list(range(100))  
one\_page\_num = 3*#每页显示3条数据*one\_time\_num = 3*#每次查询分为3页*while True:  
 page = int(input('page>>>'))  
 *#page =1 findindex = 1 int(1/3)+1  
 #page =2 findindex = 1 int(2/3)+1  
 #page =3 findindex = 1 int(3/3)  
 #page =4 findindex = 2 int(4/3)+1  
 #page =5 findindex = 2  
 #page =6 findindex = 2  
 #判断大于* if page/ one\_time\_num > int(page/ one\_time\_num):  
 *#生成查询页码* find\_index = int(page/ one\_time\_num)+1  
 else:  
 *#当等于的时候我们就直接输出查询页码* find\_index = int(page/ one\_time\_num)  
 *#每次请求vue这个url查询的总数据#9 个* select\_num = one\_page\_num \* one\_time\_num  
 *#数据起始下标* start = (find\_index-1)\*select\_num  
 *#数据结束下标* end = find\_index\*select\_num  
 *#截取数据* data = datas[start:end]  
 *#处理数据  
 # print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 # print(data)  
 # print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
  
 #现在1,2,3页对应的数据[0, 1, 2, 3, 4, 5, 6, 7, 8]  
 #我想要获取这样的数据  
 #find\_index = 1  
 #page = 1 [0,1,3] pageindex =1 now\_index = 1-(1-1)\*3  
 #page = 2 [3,4,5] pageindex =2  
 #page = 3 [6,7,8] pageindex =3  
 #find\_index = 2  
 #page = 4 pageindex =1 now\_index = 4-(2-1)\*3  
 #page = 5 pageindex =2  
 #page = 6 pageindex =3  
 #精确获取每个页码的对应的数据* now\_index = page-(find\_index-1)\*one\_page\_num  
  
 page\_start = (now\_index-1)\*one\_page\_num  
 page\_end = now\_index\*one\_page\_num  
 *#获取* page\_data = data[page\_start:page\_end]  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 print('page :',page)  
 print('page\_data :',page\_data)  
 print('data :',data)  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

运行结果如下：



编写Article/views.py

def setpage(page,one\_page\_num,one\_time\_num,datas):  
 *#判断大于* if page/one\_time\_num > int(page/one\_time\_num):  
 *#生成查询页码* find\_index = int(page/one\_time\_num)+1  
 else:  
 *#当等于的时候我们就直接输出查询页码* find\_index = int(page/one\_time\_num)  
 *#每次请求vue这个url查询的总数据#9 个* select\_num = one\_page\_num \* one\_time\_num  
 *#数据起始下标* start = (find\_index-1)\*select\_num  
 *#数据结束下标* end = find\_index\*select\_num  
 *#截取数据* data = datas[start:end]  
 now\_index = page-(find\_index-1)\*one\_page\_num  
 page\_start = (now\_index-1)\*one\_page\_num  
 page\_end = now\_index\*one\_page\_num  
 *#获取* page\_data = data[page\_start:page\_end]  
  
 result = {  
 'page':page,  
 'page\_data':page\_data,  
 }  
 return result

def vuePageData(request):  
 *'''  
 分页逻辑* :param *reuqest:* :return*: json  
 '''* if request.method == "GET":  
 page = request.GET.get('page')  
 if page and int(page) > 1:  
 page = int(page)  
 else:  
 page =1  
 one\_page\_num = 3  
 one\_time\_num = 3  
 datas = Article.objects.all()  
 result\_list = []*#发送响应信息* page\_data = setpage(page,one\_page\_num,one\_time\_num,datas)  
 datas = page\_data.get('page\_data')  
 for data in datas:  
 classify = data.classify.all()  
 if classify:  
 classify = [i.label for i in classify]  
 else:  
 classify = ''  
 result\_list.append({  
 'title':data.title,  
 'author':data.author.name,  
 'time':data.time,  
 'description':data.description,  
 'picture':data.picture.name,  
 'classify':classify,  
 'id':data.id  
 })  
 *#覆盖到原有的数据* page\_data['page\_data'] = result\_list  
 return JsonResponse(page\_data)

修改urls.py

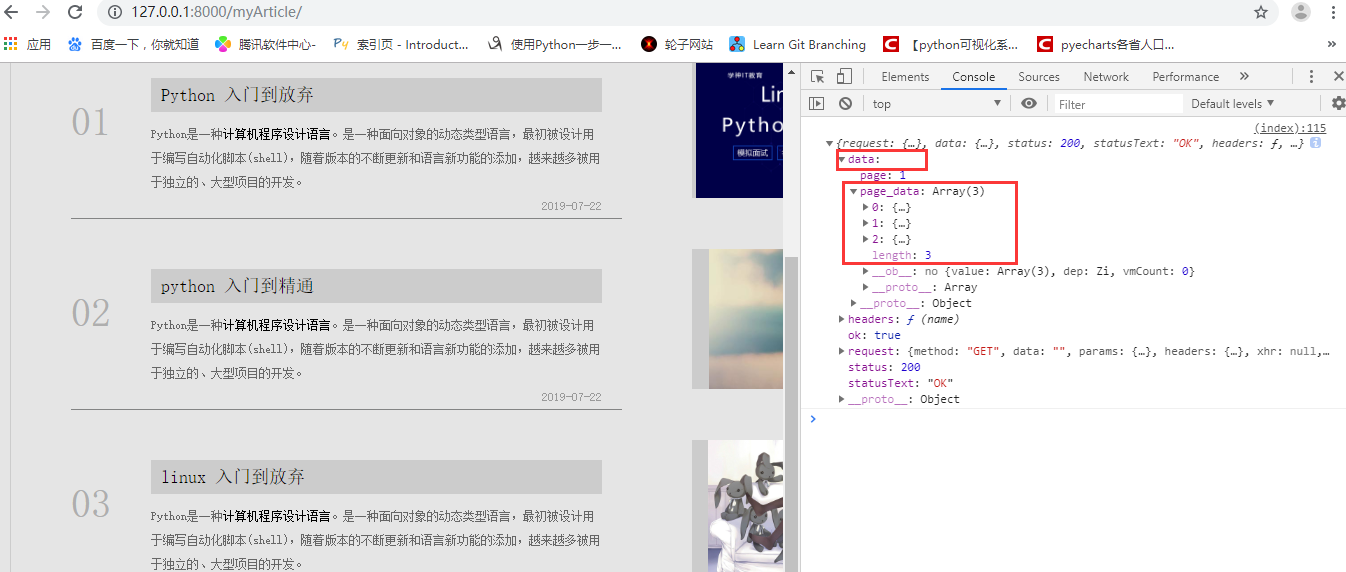
urlpatterns += [  
 url(r'vue/',vuePageData)  
]

然后修改myArticle.html前端

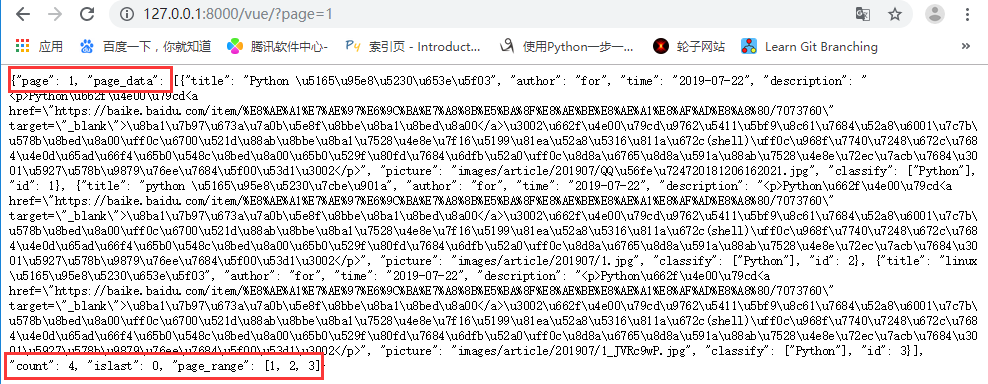
<script src = '/static/js/vue-resource.js'></script>

<script>  
 Vue.use(VueResource);  
 new Vue({  
 el:'#dataList',  
 data:{  
 page\_data:''  
 },  
 *//created方法在初始化调用的时候 相当于我们第一页 就是 page=1  
 //相当于ajax请求地址并响应* created:function () {  
 var url = '/vue/';  
 this.$http.get(url).then(  
 function (data) {  
 *console*.log(data);  
 this.page\_data = data['data']['page\_data']  
 },  
 function (error) {  
 *console*.error(error)  
 }  
 )  
 }  
 })  
</script>

结果如下所示：



我们想要对vue响应的界面分页还需要对后端响应的数据进一步处理实现如下效果



再次修改Article/views.py/setPage

def setpage(page,one\_page\_num,one\_time\_num,datas):  
 *#判断大于* if page/one\_time\_num > int(page/one\_time\_num):  
 *#生成查询页码* find\_index = int(page/one\_time\_num)+1  
 else:  
 *#当等于的时候我们就直接输出查询页码* find\_index = int(page/ one\_time\_num)  
 *#每次请求vue这个url查询的总数据#9 个* select\_num = one\_page\_num \* one\_time\_num  
 *#数据起始下标* start = (find\_index-1)\*select\_num  
 *#数据结束下标* end = find\_index\*select\_num  
 *#截取数据* data = datas[start:end]  
 now\_index = page-(find\_index-1)\*one\_page\_num  
 page\_start = (now\_index-1)\*one\_page\_num  
 page\_end = now\_index\*one\_page\_num  
 *#获取* page\_data = data[page\_start:page\_end]  
 *#获取数据总量* count = len(datas)  
 *#获取总页数为2页* final\_page = count/one\_page\_num  
 if final\_page != int(final\_page):  
 final\_page += 1  
 final\_page = int(final\_page)  
 *#判断页码数* islast = 0  
 if page >= final\_page:  
 islast = 1  
 if page in [1,2]:  
 page\_range\_start = 1  
 page\_range\_end = 4  
 elif page >= 3:  
 page\_range\_start = page - 1  
 page\_range\_end = page + 2  
 page\_range = list(range(page\_range\_start,page\_range\_end))  
 result = {  
 'page':page,  
 'page\_data':page\_data,  
 }  
 result['count'] = count  
 result['islast'] = islast  
 result['page\_range'] = page\_range  
 return result

然后交给myArticle.html加载

*//created方法在初始化调用的时候 相当于我们第一页 就是 page=1  
//相当于ajax请求地址并响应*created:function () {  
 var url = '/vue/';  
 this.$http.get(url).then(  
 function (data) {  
 *console*.log(data);  
 this.page\_data = data['data']['page\_data'];  
 this.page\_range = data['data']['page\_range']  
 },  
 function (error) {  
 *console*.error(error)  
 }  
 )  
},

编写myArticle.html中的分页展示

<div class="pagelist">

<a v-for="p in page\_range"">{{ p }}</a>  
</div>

完善代码 实现点击跳转

<div class="pagelist">  
<a v-for="p in page\_range" v-on:click="getPage(p)">{{ p }}</a>

</div>

<script>  
 Vue.use(VueResource);  
 new Vue({  
 el:'#dataList',  
 data:{  
 page\_data:'',  
 page\_range:''  
 },  
 *//created方法在初始化调用的时候 相当于我们第一页 就是 page=1  
 //相当于ajax请求地址并响应* created:function () {  
 var url = '/vue/';  
 this.$http.get(url).then(  
 function (data) {  
 *console*.log(data);  
 this.page\_data = data['data']['page\_data'];  
 this.page\_range = data['data']['page\_range']  
 },  
 function (error) {  
 *console*.error(error)  
 }  
 )  
 },  
 methods: {  
 getPage:function (page) {  
 *{#alert(page);#}* var url='/vue/?page='+page;  
 this.$http.get(url).then(  
 function (data) {  
 this.page\_data = data['data']['page\_data'];  
 this.page\_range = data['data']['page\_range']  
 }  
 )  
  
 }  
 }  
 })  
</script>

总结：

14.1 分页介绍与逻辑

14.2 vue介绍

14.3 基于vue尝试分页

附录：

myArticle.html：

{% extends "base.html" %}  
{% block title %}  
 我的博客  
{% endblock %}  
  
{% block style %}  
 <link href="/static/css/main.css" rel="stylesheet">  
 <script src="/static/js/jquery.js"></script>  
 <script src="/static/js/vue.min.js"></script>  
 <script src = '/static/js/vue-resource.js'></script>  
{% endblock %}  
  
{% block content %}  
<div class="container">  
 <div class="con\_content">  
 <div class="about\_box">  
 <h2 class="nh1"><span>您现在的位置是：<a href="/" target="\_blank">网站首页</a>>><a href="#" target="\_blank">个人日记</a></span><b>个人日记</b></h2>  
 {% verbatim myblock %}  
 <div class="dtxw box" id = "dataList">  
 <li v-for = "page in page\_data">  
 <div class="dttext f\_l">  
 <ul>  
 <h2><a href="/">{{ page.title }}</a></h2>  
 <p v-html="page.description">{{ page.description }}</p> *<!-- 注意和Django的safe标签的区分 v-html == safe -->* <span>{{ page.time }}</span>  
 </ul>  
 </div>  
 <div class="xwpic f\_r"><a href="/"><img v-bind:src="['/media/'+page.picture]"></a></div> *<!-- 注意绑定数据的拼接 -->* </li>  
 <div class="pagelist">  
 <a v-for="p in page\_range" v-on:click="getPage(p)">{{ p }}</a>  
*{# <a v-for="p in page\_range"">{{ p }}</a>#}* </div>  
 </div>  
 {% endverbatim myblock %}  
  
  
 </div>  
 </div>  
 <div class="blank"></div>  
 *<!-- container代码 结束 -->*</div>  
<script>  
 Vue.use(VueResource);  
 new Vue({  
 el:'#dataList',  
 data:{  
 page\_data:'',  
 page\_range:''  
 },  
 *//created方法在初始化调用的时候 相当于我们第一页 就是 page=1  
 //相当于ajax请求地址并响应* created:function () {  
 var url = '/vue/';  
 this.$http.get(url).then(  
 function (data) {  
 *console*.log(data);  
 this.page\_data = data['data']['page\_data'];  
 this.page\_range = data['data']['page\_range']  
 },  
 function (error) {  
 *console*.error(error)  
 }  
 )  
 },  
 methods: {  
 getPage:function (page) {  
 *{#alert(page);#}* var url='/vue/?page='+page;  
 this.$http.get(url).then(  
 function (data) {  
 this.page\_data = data['data']['page\_data'];  
 this.page\_range = data['data']['page\_range']  
 }  
 )  
  
 }  
 }  
 })  
</script>  
{% endblock %}

Article/views.py

**from django.http import JsonResponse  
from django.shortcuts import render  
  
*# Create your views here.  
# def base(request):  
# # return render(request,'base.html')*from .models import Article  
from pure\_pagination import Paginator  
def index(request):  
 return render(request,'index.html')  
*#文章*def myArticle(request):  
 *# articles = Article.objects.all()  
 # try:  
 # page = request.GET.get('page',1)  
 # except:  
 # page = 1  
 # paginator = Paginator(articles,2)  
 # pageData = paginator.page(page)* return render(request,'myArticle.html')  
def aboutMe(request):  
 return render(request,'aboutMe.html')  
def myPicture(request):  
 return render(request,'myPicture.html')  
  
def vuetest(request):  
 *'''* :param *request:* :return*:  
 提供json数据  
 '''* if request.method == 'GET':  
 try:  
 page = request.GET.get('page',1)  
 except:  
 page = 1  
 articles = Article.objects.all()  
 paginator = Paginator(articles,3)  
 pageData = paginator.page(page)  
  
 page\_data = []  
  
 for data in pageData.object\_list:  
 classify = data.classify.all()  
 if classify:  
 classify = [i.label for i in classify]  
 else:  
 classify = ''  
 page\_data.append({  
 'title':data.title,  
 'author':data.author.name,  
 'time':data.time,  
 'description':data.description,  
 'picture':data.picture.name,  
 'classify':classify,  
 'id':data.id  
 })  
 result = {  
 'pageData':page\_data  
 }  
 return JsonResponse(result)  
  
def setpage(page,one\_page\_num,one\_time\_num,datas):  
 *#判断大于* if page/one\_time\_num > int(page/one\_time\_num):  
 *#生成查询页码* find\_index = int(page/one\_time\_num)+1  
 else:  
 *#当等于的时候我们就直接输出查询页码* find\_index = int(page/one\_time\_num)  
 *#每次请求vue这个url查询的总数据#9 个* select\_num = one\_page\_num \* one\_time\_num  
 *#数据起始下标* start = (find\_index-1)\*select\_num  
 *#数据结束下标* end = find\_index\*select\_num  
 *#截取数据* data = datas[start:end]  
 now\_index = page-(find\_index-1)\*one\_page\_num  
 page\_start = (now\_index-1)\*one\_page\_num  
 page\_end = now\_index\*one\_page\_num  
 *#获取* page\_data = data[page\_start:page\_end]  
 *#获取数据总量* count = len(datas)  
  
 *#获取总页数为2页* final\_page = count/one\_page\_num  
 if final\_page != int(final\_page):  
 final\_page += 1  
 final\_page = int(final\_page)  
 *#判断页码数* islast = 0  
 if page >= final\_page:  
 islast = 1  
 if page in [1,2]:  
 page\_range\_start = 1  
 page\_range\_end = 4  
 elif page >= 3:  
 page\_range\_start = page - 1  
 page\_range\_end = page + 2  
 page\_range = list(range(page\_range\_start,page\_range\_end))  
 result = {  
 'page':page,  
 'page\_data':page\_data,  
 }  
 result['count'] = count  
 result['islast'] = islast  
 result['page\_range'] = page\_range  
 return result  
  
def vuePageData(request):  
 *'''  
 分页逻辑* :param *reuqest:* :return*: json  
 '''* if request.method == "GET":  
 page = request.GET.get('page')  
 if page and int(page) > 1:  
 page = int(page)  
 else:  
 page = 1  
 one\_page\_num = 3  
 one\_time\_num = 3  
 datas = Article.objects.all()  
 result\_list = []*#发送响应信息* page\_data = setpage(page,one\_page\_num,one\_time\_num,datas)  
 datas = page\_data.get('page\_data')  
 for data in datas:  
 classify = data.classify.all()  
 if classify:  
 classify = [i.label for i in classify]  
 else:  
 classify = ''  
 result\_list.append({  
 'title':data.title,  
 'author':data.author.name,  
 'time':data.time,  
 'description':data.description,  
 'picture':data.picture.name,  
 'classify':classify,  
 'id':data.id  
 })  
 *#覆盖到原有的数据* page\_data['page\_data'] = result\_list  
  
 return JsonResponse(page\_data)  
 else:  
 return JsonResponse({'error':'no data'})  
  
Ourblog/urls  
from django.contrib import admin  
from django.urls import path, include  
from django.conf.urls import url  
from django.views.static import serve  
from OurBlog.settings import MEDIA\_ROOT  
import xadmin  
from Article.views import \*  
urlpatterns = [  
 path('xadmin/', xadmin.site.urls),  
 url(r'cheditor/',include('ckeditor\_uploader.urls')),  
 url(r'^media/(?P<path>.\*)',serve,{'document\_root':MEDIA\_ROOT}),  
 url(r'^$',index),  
 url(r'index/',index),  
 url(r'myArticle/',myArticle),  
 url(r'myPicture/',myPicture),  
 url(r'aboutMe/',aboutMe),  
]  
  
urlpatterns += [  
 url(r'vue/',vuePageData)  
]**