Ajax

> `ajax`可以使当前浏览器不需要整个重新加载，只是局部刷新，给用户的体验良好，也因为只是刷新局部页面，相对而言效率更高一些
>
> 同步交互：客户端发出一个请求后，需要等待服务器相应结束后，才可以发起第二个请求
>
> 异步交互：客户端发出一个请求后，无需等待该次服务器的相应，即可发起第二个请求

json

- 数据在键值对中
- 数据由逗号分隔
- 花括号存储数据
- 方括号保存数组

```
[
    { "name":"Bill", "age":1 },
    { "name":"George", "age":2 },
    { "name":"Thomas", "age": 3 }
];
```

`jQuery-Ajax`

> 使用`ajax`进行`django`后台数据的异步获取，`django`只是提供的数据，并不承担前端页面的渲染工程
>
> 这里使用`jQuery`所提供的`ajax`方法进行异步通信

- 首先测试数据库中模型类定义如下：

```
class Article(models.Model):
    title = models.CharField(max_length=50,verbose_name="标题")
    author = models.CharField(max_length=20,verbose_name="作者")
    date = models.DateField(auto_now_add=True,verbose_name="发表日期")
    content = models.TextField(verbose_name="文章内容")

    def __str__(self):
        return self.title
```

> 测试数据可由用户自行添加，非常简单

- 编写主页视图函数，返回所有数据库中内容

```
def index(request):
    articles = models.Article.objects.all()
    return render(request,'ajax/index.html',locals())
```

> 此处的`index.html`页面不光承担所有数据的渲染工作

> 还将负责未来`ajax`异步请求，获取对应文章的详细内容

- `index.html`页面代码

```
<style>
    label{
        border: 5px outset gray;
        width: 150px;
        margin-top: 10px;
    }
</style>
```

```
<head>
    {% load staticfiles %}
    <meta charset="utf-8">
    <title>Ajax测试</title>
    <script type="text/javascript" src="{% static 'js/jquery-1.10.2.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'js/jquery.cookie.js' %}"></script>
    <!-- 该js文件用来引入jquery所提供的获取cookie值的库 为了提取对应csrf_token-->
</head>

<body>
    <h1>这是一个ajax的请求测试</h1>
        {% for article in articles %}
            <label class="{{ article.id }}">{{ article.author }}:{{ article.title }}
</label>
        {% endfor %}
    <p class="content"></p>
</body>
```

```
<script type="text/javascript">
    $(document).ready(function () {
        $("label").click(function () {
            $.ajax({
                url: '/article/', // 请求地址，对应Django某个路由映射
                type: 'POST', // 请求方式 post
                data: {
                    'csrfmiddlewaretoken': $.cookie('csrftoken'),
                    // 提交数据需有当前csrf_token 防跨站请求伪造令牌
                    'id_': $(this).attr('class'),
                    // 获取当前的id值 传递到视图后台
                },
                success: function (result) {
                    var data = JSON.parse(result)
                    // 解析获得实际字符串
                    $('.content').html(data)
                    // 将内容以html形式显示到对应的p标签上
                }
            })
        })
    })
</script>
```

> 有了前端页面，并且`ajax`的请求地址为`/article/`，那么就需要我们定义一个视图函数返回对应的`json`数据，并且设置路由为`/article/`

```
#urls.py
path('ajax/',ajaxviews.index), # 首页路由
path('article/',ajaxviews.article) # ajax请求路由
```

```
#views.py
def article(request):
    if request.is_ajax(): # 判断是否为ajax请求
        if request.method == "POST": # 为ajax的post方式请求
            id_ = request.POST.get('id_')
            if id_:
                try:


                    content =
models.Article.objects.get(id=id_).content.replace('\r\n','<br>')
                # 这里还将获取到的文章字符串内容中的换行替换为HTML的换行标签
                except models.Article.DoesNotExist:
                    raise Http404
                else:
                    data = json.dumps(content,ensure_ascii=False,cls=JsonEncoder)
                    # 返回get对应取到的实际属性
                    return HttpResponse(data)
    raise Http404
```

> 这里要注意的是，后端返回的数据得是序列化之后的才可以被前端`js`所解析，直接返回一个`django model`数据实例是不行的。所以需要我们视图函数对需要返回的数据进行序列化操作

> 对于数据的序列化操作主要有以下两种

json序列化

> 普通`Python`数据直接使用`json`模块进行序列化

```
content = models.Article.objects.get(id=id_).content.replace('\r\n','<br>')
#这里将文章内容对应返回，之所以有replace函数，是因为文章数据是通过admin后台复制添加，需要将其中的\r\n换行转换为HTML可以解析的<br>标识符
data = json.dumps(content,ensure_ascii=False)
# 第二个参数是因为序列化时对中文默认使用的ascii编码，此时需要将该值设置为False，这样前端接收到时才是一个正常中文结果
return HttpResponse(data)
```

> 但如果要序列化的数据中包含时间类型`date`或`datetime`时，这种办法就会报错啦

> `TypeError: Object of type date is not JSON serializable`

```
class JsonEncoder(json.JSONEncoder):
    # 自定义json处理器
    def default(self, obj):
        if isinstance(obj, datetime):
            # 如果判断到类型为datetime格式
            return obj.strftime('%Y-%m-%d %H:%M:%S')
            # 处理为字符串类型的 (年-月-日 时:分:秒)
        elif isinstance(obj, date):
            # 如果判断到json处理数据为date类型
            return obj.strftime('%Y-%m-%d')
        else:
            return json.JSONEncoder.default(self,obj)
            # 其他数据类型按照默认的序列化方式处理即可
```

> 使用`cls`指定序列化方式，即可轻松解决特殊格式没有办法被`json`序列化的问题

```
content = models.Article.objects.get(id=id_).date
data = json.dumps(content,ensure_ascii=False,cls=JsonEncoder)
# 通过json.dumps的cls参数指明所使用的自定义序列化类
return HttpResponse(data)
```

> 对应前端接收展示

```
var data = JSON.parse(result) // 普通json传输方式
$('.content').html(data)
```

> 如果返回的数据并不是一个单独的数据属性，那么也可以通过`json`进行处理，以一个数据列表的形式返回

```
content = models.Article.objects.filter(id=id_).values()
# ------------------------------------------------
# content = models.Article.objects.all().values()
# ------------------------------------------------
data = json.dumps(list(content),ensure_ascii=False,cls=JsonEncoder)
return HttpResponse(data)
```

> 对应前端接收展示

```
<div class="content">
    <!-- 这里用到的不是之前的p标签 而是一个div容器 -->
</div>
```

```
success: function (result) {
    var data = JSON.parse(result)[0]['content']
    $('.content').html(data.replace(/\r\n/g,"<br>"))
}
// ------------------------------------------------
// 如果需要展示的是所有的结果，可以通过js的for循环
success: function (result) {
    var data = JSON.parse(result)
```

```
    var tag = ''
    for (var i = 0, len = data.length; i < len; i++) {
        tag += '<p>' + data[i]['content'].replace(/\r\n/g, "<br>") + '</p>'
        tag += '<hr>'
    }
    $('.content').html(tag)
}
// ---------------------------------------------
```

serializer序列化

> `serializer`是由`django`所提供的一个专门用来处理`django`数据对象`(django model)`变为序列化数据的框架
>
> 并且`Django`的序列化不支持**单个对象**，比如像`objects.get`获取到的数据，或是`Python`中的`str`等数据类型
>
> 该序列化框架所提供的功能类位于`django.core.serializers`

```
#views.py
from django.core import serializers
content = models.Article.objects.filter(id=id_)
data = serializers.serialize('json',content,ensure_ascii=False)
return HttpResponse(data)
```

```
var data = JSON.parse(result)[0]['fields']['content'] // 序列化传输方式
$('.content').html(data.replace(/\r\n/g,"<br>"))
console.log(data)
```

> **总结**：通过管理器的`get`方法获取到的是一个独立的结果，并不是一个`QuerySet`数据对象，也不是一个普通`Python`数据类型；只能对数据其中的某条属性进行`json`格式的处理或是将其变为列表等序列数据类型之后再进行序列化处理

serializer反序列化

> 序列化：`serializers.serialize`
>
> 反序列化：`serializers.serialize`

```
from django.core import serializers
content = models.Article.objects.filter(id=id_) # QuerySet
data = serializers.serialize('json',content,ensure_ascii=False) # str
content = serializers.deserialize("json", data)
return HttpResponse(data)
```

`Vue-Axios`

> 除去`jQuery`所提供的异步通信`ajax`方法
>
> 在`Vue`中也提供了`ajax`的异步通信方法，叫做`Axios`
>
> `Axios`会自动转换`json`数据

> 简单的来编写一个视图函数
>
> `get`：返回当前页面
>
> `post`：返回一条`json`数据

```
window.onload = function () {
    new Vue({
        el: '#content', // Vue接管的区域
        data: {
            message: '这个是表单内容',
        },
        methods: {
            getajax() {
                axios.get('/get_ajax/', {
                    params: { // 这部分为get方式进行传参时使用的
                        id: 123
                    }
                })
                    .then(function (response) {
                    console.log(response) // 打印输出get方式进行ajax请求时获取到的数据
                })
                    .catch(function (error) {
                    console.log(error) // 当get方式ajax请求报错时，会进入该函数
                })
            }
        },
    })
}
```

> 对应的`HTML`页面

```
<body>
    <div id="content">
        <button @click='getajax'>点我发送ajax的get请求</button>
    </div>
</body>
```

> 后台视图函数

```
if request.method == 'GET':
    message = request.GET.get('message')
    print(message)
    return render(request,'axios/index.html')
```

> 当使用的是`post`形式获取服务端数据时，首先要注意，`axios`默认的提交`post`数据不是普通的`form-data`
>
> `axios`的`post`使用的是`request payload`方式，参数格式是`application/json;charset=UTF-8`
>
> 而我们之前的表单提交数据的类型都是`application/x-www-form-urlencoded`，所以直接再`django`后台通过`request.POST.get`是获取不到任何数据的

> 解决办法，需要我们在`axios`提交数据时，指明提交时的头部信息

```
window.onload = function () {
    new Vue({
        el: '#content', // Vue接管的区域
        data: {
            message: '这个是表单内容',
        },
        methods: {
            getajax() {
                axios({
                    method: 'post',
                    url: '/get_ajax/',
                    data: {
                        message:this.message,
                        name: '张三'
                    },
                    headers: {
                        'Content-Type':'application/x-www-form-urlencoded',
                    },
                }).then((response) => {
                    console.log(response.data)
                    this.message = response.data
                })
            }
        },
    })
}
```

> 虽然通过添加头部信息，可以让`axios`发送的数据被`django`后台所接收到，但是此时的数据还是有问题的
>
> 获取到的`POST`提交的数据被`django`打包成了一个`QueryDict`中的key值，`value`为空数组
>
> 导致后台按照平时的解析方式是获取不到的
>
> 解决办法也很简单，把`QueryDict`单独处理为一个字典

```
if request.method == 'POST':
    data = eval(list(request.POST.keys())[0]) # 将获取到的数据转换为字典
    message = data.get('message')
    data = json.dumps(message + '我被服务端后台修改过')
    return HttpResponse(data)
```

> 接下来，当用户点击按钮时，`post`提交表单数据，给到`django`后台，后台追加字符串并返回，返回的数据被`then`回调函数所接收到，重新赋值给绑定的表单变量中

Ajax跨域

> 浏览器有一个很重要的概念：同源策略`(Same-Origin Policy)`
>
> **所谓同源是指，域名，协议，端口相同。不同源的客户端脚本`javascript、ActionScript`在没明确授权的情况下，不能读写对方的资源**

> 同源：请求资源的地址与请求的发起方都属于同一域名下

jQuery-JSONP

> `JSONP`是`JSON with padding`（填充式`JSON`或参数式`JSON`）的简写
>
> **`JSONP`实现跨域请求的原理简单的说，就是动态创建`<script>`标签，然后利用`<script>`的`src`不受同源策略约束来跨域获取数据。**
>
> `JSONP`由两部分组成：回调函数和数据
>
> 回调函数是当响应到来时应该在页面中调用的函数；回调函数的名字一般是在请求中指定的，而数据就是传入回调函数中的参数

> 注意：`JSONP`方式解决`AJAX`跨域，必须使用`get`方式，并且该方式常在一些数据量级比较小的情况下，因为需要服务端后台构建回调函数带参数的字符串，像是下面这样

```
def index(request):
    name = request.GET.get('name') + '哈哈哈哈哈'
    callback = request.GET.get('callback')
    data = '%s("%s")' % (callback,name)
    # 这里以前端生成的回调函数名作为函数名，待返回数据作为参数返回
    return HttpResponse(data)
```

- 前端代码：点击按钮传送表单的值到后台，并由后台处理后追加内容返回，返回的结果展示再`p`标签处

```
<input type='text' id='ajax_data'>
<button>
    按钮
</button>
<p id="content"></p>
```

- `Ajax`代码，获取当前表单数据，并使用`get`方式传递到服务端

```
$(document).ready(function () {
    $("button").click(function () {
        $.ajax({
            url: 'http://127.0.0.1:8000/axios/', // 请求地址，对应Django某个路由映射
            type: 'get', // 请求方式 post
            dataType: "jsonp", // 指定服务端返回的数据为jsonp格式
            data: {
                'name': $('#ajax_data').val(),
            },
            success: function (result) {
                console.log(result)
                $('#content').html(result)
            }
        })
    })
})
```

1. `ajax`发起请求，并指定服务端返回数据类型为`jsonp`格式

2. 服务端构建函数包含参数的字符串，为`jsonp`请求发起时，给定的回调参数名，参数为要返回的数据
3. 客户端先会调用回调函数，然后会调用`success`回调函数可以接收处理服务端返回的数据
    - `success`回调函数是成功返回数据后必定会调用的函数