# 第十一章 django 请求与表单

本节所讲内容：

11.1 深入探索Request请求

11.2 form表单的定义

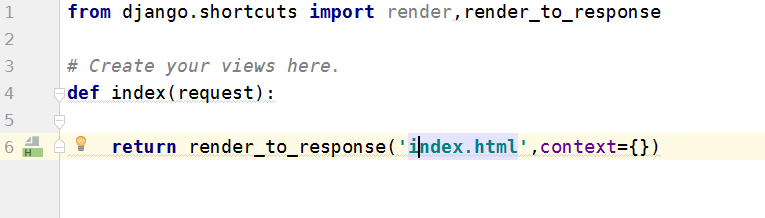
11.3 Django的form表单类

11.4 表单提交的案例

## 11.1 深入探索Request请求

Web开发是在http协议的基础之上处理各种各样的请求，用户的访问叫做请求(request),对用户的回复叫做响应(response).

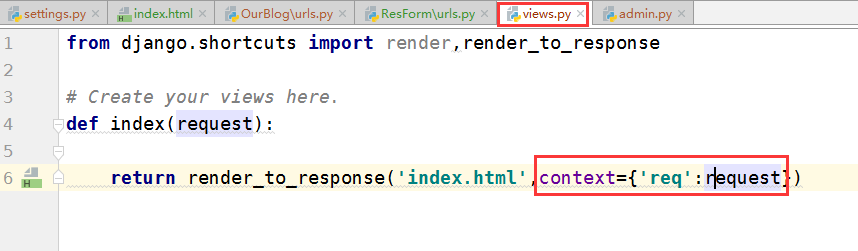
下面就来学习requests，在学习django的之初的例子，就介绍了django的MVC,其中的v指的是views文件，里面存放的是视图函数，而视图函数就是用来响应请求的，在这里，django给每一个函数都添加了一个参数用来接收请求，这个参数约定俗成，被大家称为request。



这个参数具体接收的样式是什么样的，使本章要学习的内容。首先通过模板渲染，将这个函数传递到前端index页面上，查看一下

Views.py

from django.shortcuts import render,render\_to\_response  
*# Create your views here.*def index(request):  
 return render\_to\_response('index.html',context={'req':request})



Index.html

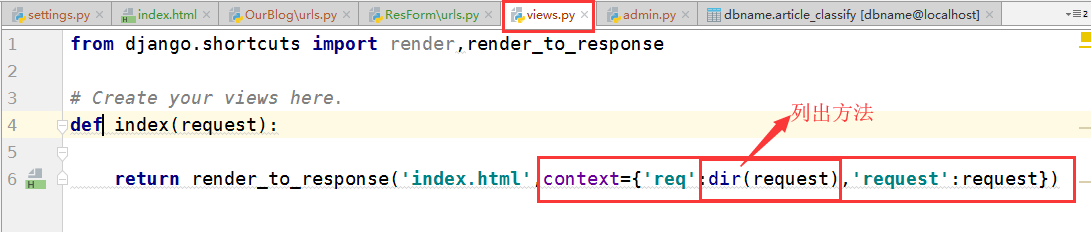


得到的结果如下



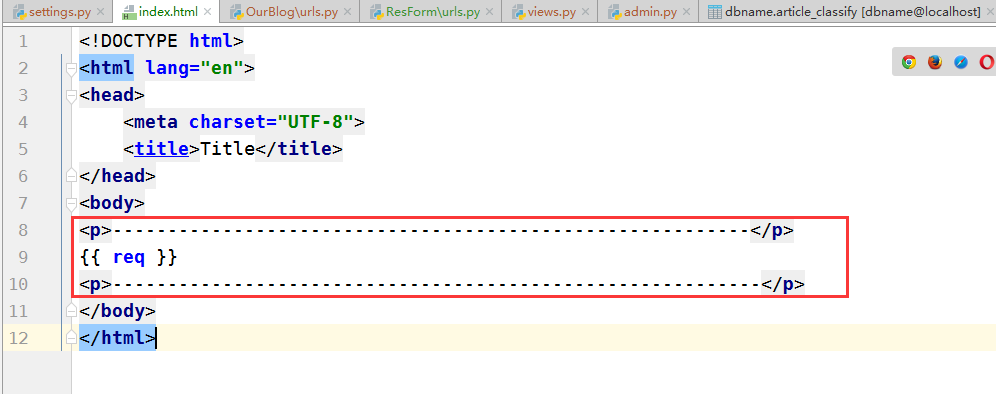
request是一个wsgi的请求对象，那么接下来我们来查看一下这个对象当中具体的属性

from django.shortcuts import render,render\_to\_response  
*# Create your views here.*def index(request):  
 return render\_to\_response('index.html',context={'req':dir(request),'request':request})

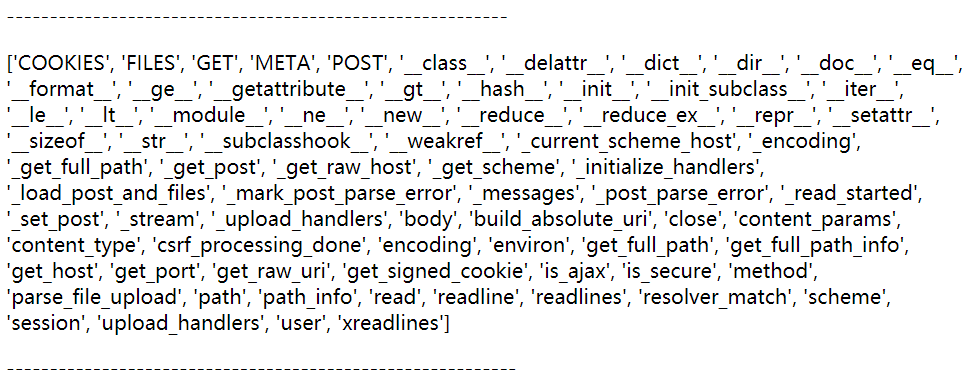


首先查看一下request对象当中的方法，使用了dir方法，得到如下效果

Index.html



运行结果如下：

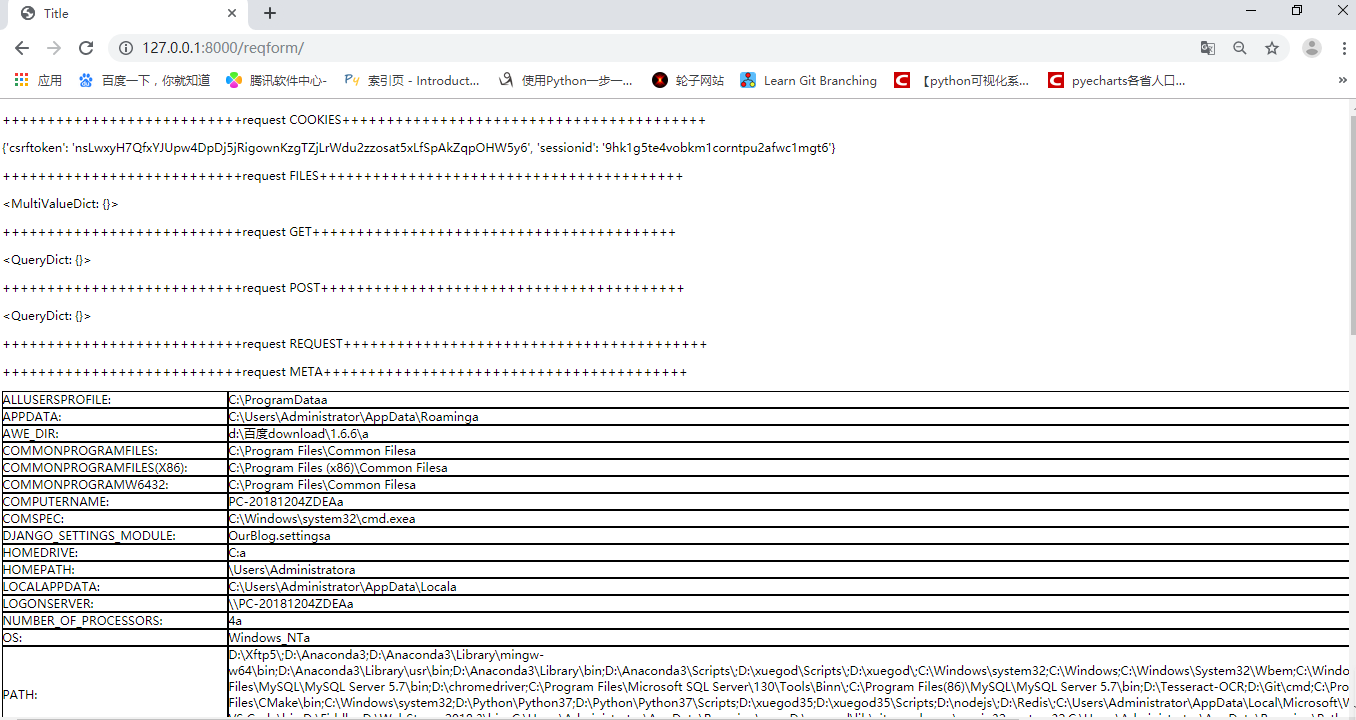


这里展示出来的是所有的request的方法，我们先认识一下前面的几个，视图函数当中不用修改，直接在index.html当中渲染。

为了更好的给大家展示我们对前端index.html进行具体修改，前端代码如下：

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
<p>+++++++++++++++++++++++++++request COOKIES+++++++++++++++++++++++++++++++++++++++++</p>  
{{ request.COOKIES }}  
<p>+++++++++++++++++++++++++++request FILES+++++++++++++++++++++++++++++++++++++++++</p>  
{{ request.FILES }}  
<p>+++++++++++++++++++++++++++request GET+++++++++++++++++++++++++++++++++++++++++</p>  
{{ request.GET }}  
<p>+++++++++++++++++++++++++++request POST+++++++++++++++++++++++++++++++++++++++++</p>  
{{ request.POST }}  
<p>+++++++++++++++++++++++++++request REQUEST+++++++++++++++++++++++++++++++++++++++++</p>  
{{ request.REQUEST }}  
<p>+++++++++++++++++++++++++++request META+++++++++++++++++++++++++++++++++++++++++</p>  
<table border="1" bordercolor="#000000" width="1px" cellpadding = "0" cellspacing = "0">  
 {% for keys,values in request.META.items %}  
 <tr>  
 <td>{{ keys }}:</td>  
 <td>{{ values }}a</td>  
 </tr>  
 {% endfor %}  
</table>  
</body>  
</html>

运行效果如下：



发现好多没有值的参数，在这里，给大家介绍一下

request.cookies是用来存储请求的cookie的，通常我们在编写网站的时候进行cookie的校验，就是从这里校验

request.FILES 是用来接收客户端提交的文件的数据的，比如用户上传的图片和文件.

request.GET是用来接收客户端提交的get数据

request.POST是用来接收客户端提交的post数据

request.REQUEST是用来接收存储跳转的容器，（通常不用，后面CMDB项目上讲）

当然，还有一个值超长的对象，META，这个是一个字典对象，所以在这里对它进行单独的渲染

而下面小写的方法，我们在后面请求过程当中会逐渐用到。

## 11.2 form表单的定义

上面课程的学习，我们已经学会了如何创建Django项目、完成网站展示功能、如何定义数据库、利用Django自带的admin讲数据保存到数据库当中。但是现在并没有学习到如何自己向服务器传递数据，

在我们工作的过程当中，向服务器提交数据是必不可少的，而最常用的方式就是前端的form表单提交数据到服务器。

Django开发过程当中的表单数据提交分为三个部分。

11.2.1表单的定义

说道表单的定义，我们不得不回顾一下之前学习过的前端知识，因为form表单，本身就是来源于前端的form标签。

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
<form action="/reqform/" method="post">  
 {% csrf\_token %}  
 姓名:<input type="text" name="username"><br>  
 密码:<input type="password" name="password"><br>  
 邮箱:<input type="email" name="email"><br>  
 电话:<input type="text" name="phone"><br>  
 <input type="submit" value="提交">  
</form>  
</body>  
</html>

界面效果如下：



* + 1. form表单的属性

action 提交的路径，我们要把数据提交到哪里，如果不行，默认提交到当前url指定的视图上method 提交的方式，request常见的方式

get GET请求会向数据库发索取数据的请求，从而来获取信息，该请求就像数据库的select操作一样，只是用来查询一下数据，不会修改、增加数据，不会影响资源的内容，即该请求不会产生副作用。无论进行多少次操作，结果都是一样的。

Post 向服务器端发送数据的，但是该请求会改变数据的种类等资源，就像数据库的insert操作一样，会创建新的内容。几乎目前所有的提交操作都是用POST请求的。

Put PUT请求是向服务器端发送数据的，从而改变信息，该请求就像数据库的update操作一样，用来修改数据的内容，但是不会增加数据的种类等，也就是说无论进行多少次PUT操作，其结果并没有不同。

Delete 请求顾名思义，就是用来删除某一个资源的，该请求就像数据库的delete操作。

* + 1. form表单的input类型和关键参数

input类型通常有以下几种

button 按钮；checkbox 选择框；file 文件；hidden 隐藏域；image 图片；password 密码域  
radio 单选框；submit 提交按钮；text 文本域

数据提交过程当中，对于input最为重要的参数

name 决定提交数据的结构；value 提交数据的值

当然submit是最为传统的一种提交方式，它会将表单当中所有的input提交到action指定的路径上。

1. 数据的发送

数据的发送我们通常通过上面讲的HTML的submit提交进行发送

1. 数据的接收

前端数据提交之后，我们可以通过视图函数的对应方法进行数据的接收和判断，之前我们查看了request所有的方法。那么我们回忆一下在数据接收部分常用到的request方法。

|  |  |
| --- | --- |
| request.GET | Get请求的数据字典 |
| request.POST | POST请求的数据字典 |
| request.method | 请求的方法 |

在这里要注意的是POST和GET方法在我们开发的过程当中使用的相当频繁，所有Django创建了request.GET和request.POST方法，用两个字典接收数据至于put和delete方法就需要我们定义视图类进行重写了。在这里要注意的是，在Django当中凡是能够对数据发生修改的请求方法（post、put、delete），都需要通过csrf校验。

知识补充：

Csrf（Cross-site request forgery）跨站请求伪造，也被称为“One Click Attack”或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。

攻击通过在授权用户访问的页面中包含链接或者脚本的方式工作

例如：一个网站用户Bob可能正在浏览聊天论坛，而同时另一个用户Alice也在此论坛中，并且Alice刚刚发布了一个具有Bob银行链接的图片消息。设想一下，Alice编写了一个在Bob的银行站点上进行取款的form提交的链接，并将此链接作为图片src。如果Bob的银行在cookie中保存他的授权信息，并且此cookie没有过期，那么当Bob的浏览器尝试装载图片时将提交这个取款form和他的cookie，这样在没经Bob同意的情况下便授权了这次事务。

那么我们接下来看一个表单注册的实例。

首先我们确定我们要编写的功能，一个提供给博客的访客发送自己的联系方式的功能。并且搭建相对应的数据库

接收表单数据，存入数据库

接下来让我们用两个实例给大家操作下：

1. Form的get请求

填写数据发送的网址示例:

<http://127.0.0.1:8000/reqform/?username=root&password=123&email=1286211699%40qq.com&phone=17611100746>

网站地址参数详解：

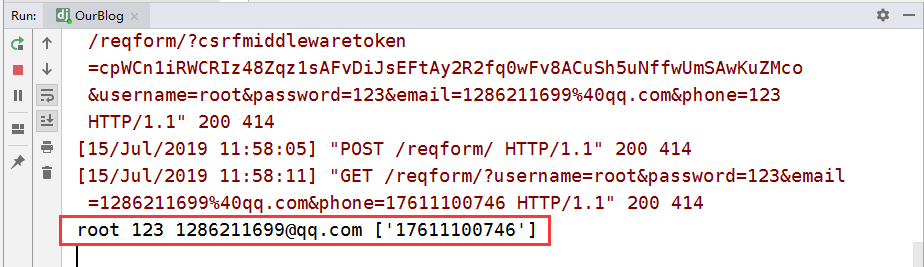
？ 网站地址和传递参数分割符

& 参数的链接符

紧接着views中进行接收

def reqform(request):  
 if request.method == 'GET' and request.GET:  
 try:  
 username = request.GET.get('username')  
 password = request.GET.get('password')  
 email = request.GET.get('email')  
 phone = request.GET.get('phone')  
 except:  
 pass  
 print(username,password,email,phone)  
 return render(request,'reqform.html')

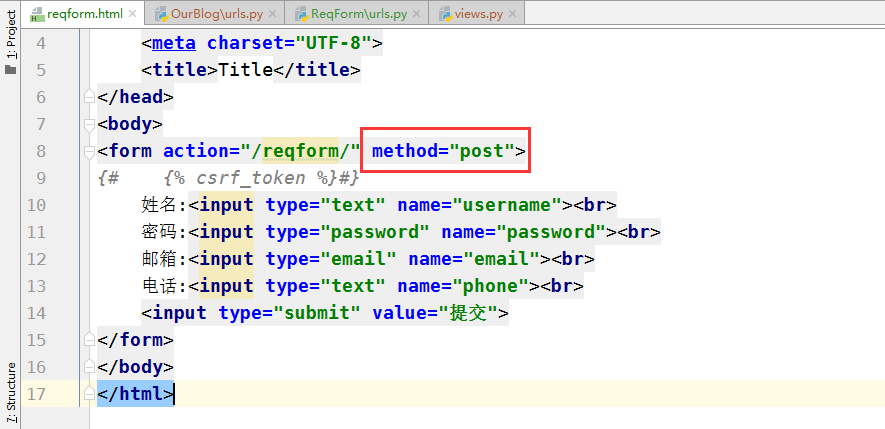
运行结果如下：



注意：如果前端返回是一个列表可用getlist()方法，例如：phone = request.GET.getlist('phone')

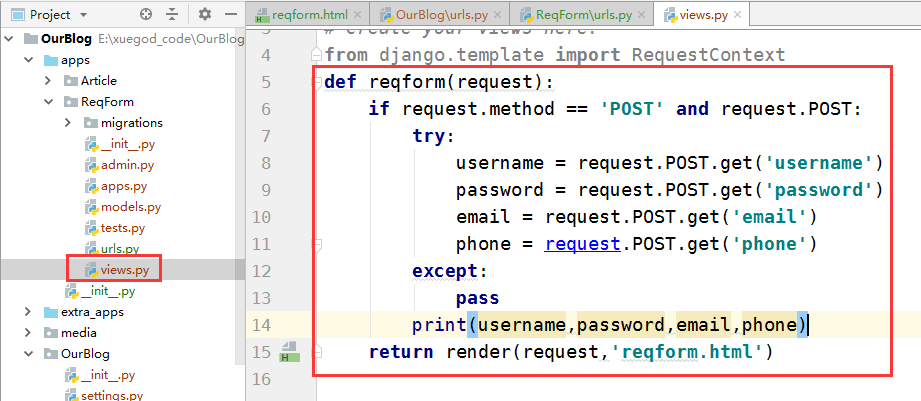
很明显上面的代码中我们的password是明文，在url中直接展示的不太符合我们的保密要求，接下2、form表单的get请求

将reqform.html中的method改成post如下图所示：

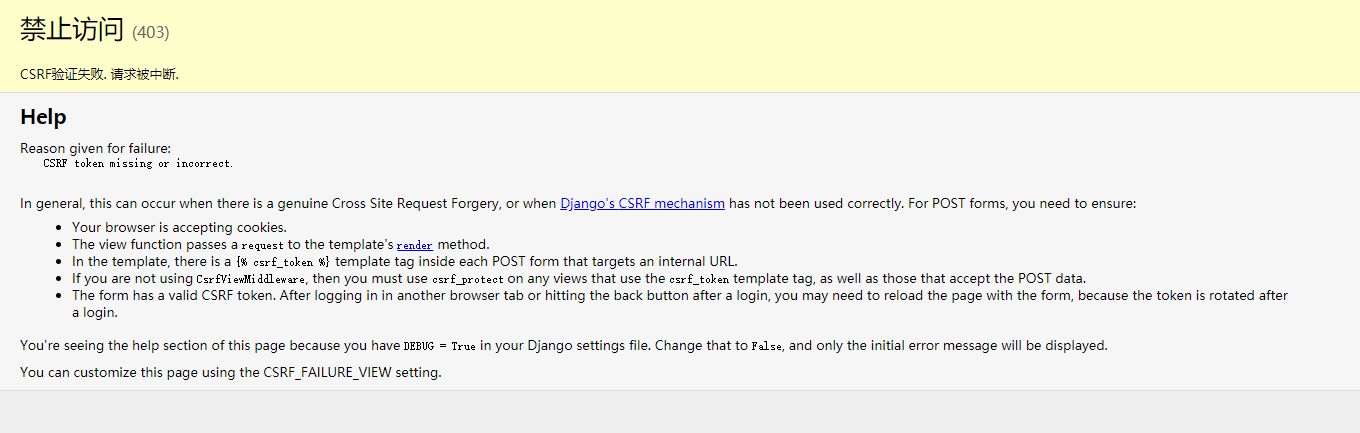


**然后在ReqForm/views.py中修改先前的get字段  
def reqform(request):  
 if request.method == 'POST' and request.POST:  
 try:  
 username = request.POST.get('username')  
 password = request.POST.get('password')  
 email = request.POST.get('email')  
 phone = request.POST.get('phone')  
 except:  
 pass  
 print(username,password,email,phone)  
 return render(request,'reqform.html')**

代码图片展示



运行项目进行表单提交出现如下错误



这是跨域请求的错误，解决这个错误的方法有多种

第一种：

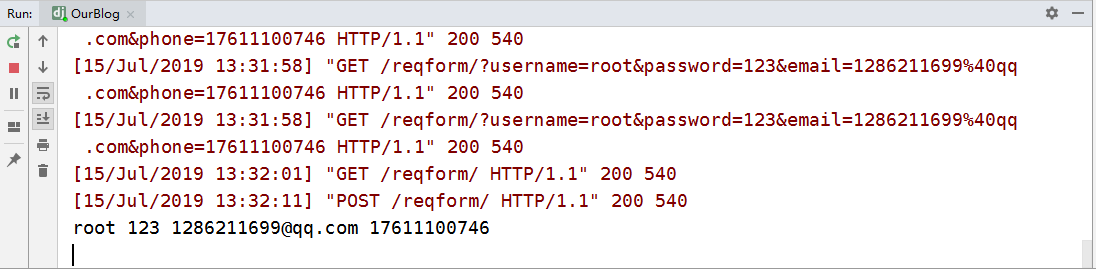
在reqform.html的form表单中写入{% csrf\_token%}，在ReqForm/views.py中render进行响应处理，如下代码图片展示：



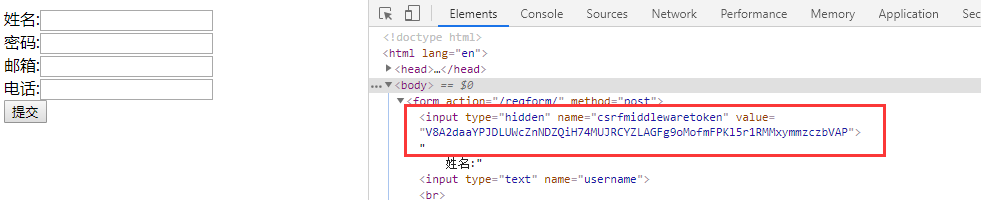
Views.py



运行结果如下：

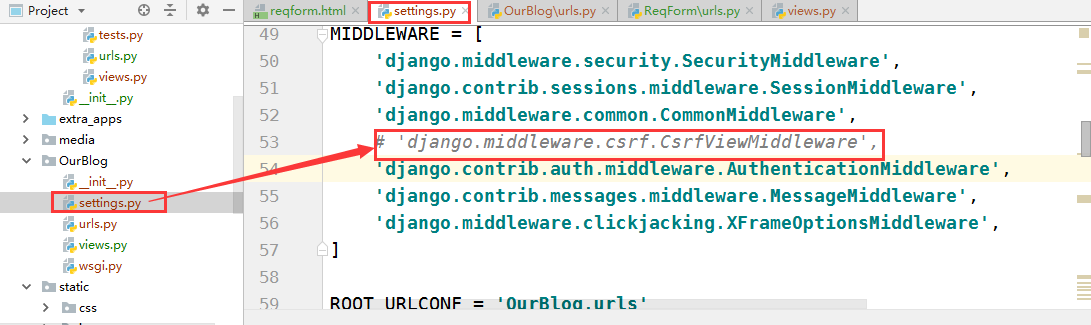


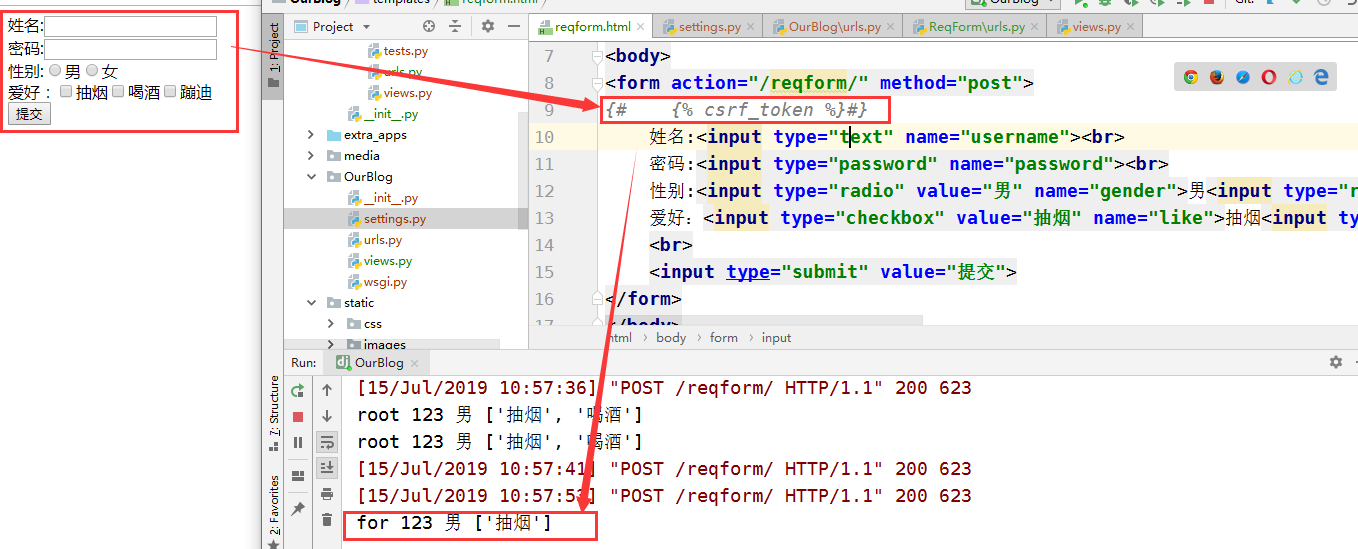
可看到网站中的标识



第二种方法：

在settings.py中屏蔽csrf中间件（这种方法不推荐不安全）

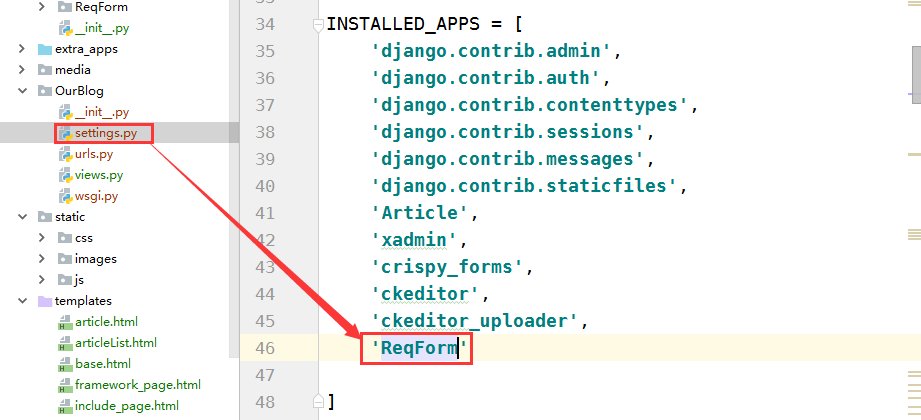




* + 1. 数据库交互

上面我们Reqform/views.py 下已经接收到数据了，如果我们想要把书保存到数据库中，需要在ReqForm/models.py中 建表。

在settings.py中配置INSTALL\_APPS



创建ReqForm/models.py

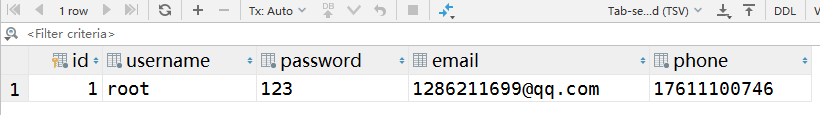
from django.db import models  
*# Create your models here.*class User(models.Model):  
 username = models.CharField(max\_length=20,verbose\_name='用户名')  
 password = models.CharField(max\_length=20,verbose\_name='密码')  
 email = models.EmailField(max\_length=20,verbose\_name='邮箱',null=True)  
 phone = models.CharField(max\_length=20,verbose\_name='电话')  
  
 class Meta:  
 verbose\_name = '用户'  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return '用户:%s'%self.username

编写ReqFoem/views.py

from django.http import JsonResponse  
from django.shortcuts import render,render\_to\_response  
  
*# Create your views here.*from django.template import RequestContext  
from .models import User  
  
def reqform(request):  
 if request.method == 'POST' and request.POST:  
 username = request.POST.get('username')  
 password = request.POST.get('password')  
 email = request.POST.get('email')  
 phone = request.POST.get('phone')  
 if username != None:  
 user = User.objects.create(  
 username=username,  
 password=password,  
 email=email,  
 phone=phone,  
 )  
 user.save()  
 return JsonResponse({'status':'success'})  
 else:  
 return JsonResponse({'status':'error'})  
 return render(request,'reqform.html')

结果运行如下：





## 11.3 Django的form表单类

上面的例子当中，实现了用户信息的提交入库，但是在Django当中，除了用上面的方法来定义Django的表单之外，Django本身也有一个叫做form的类，用来定义form表单。

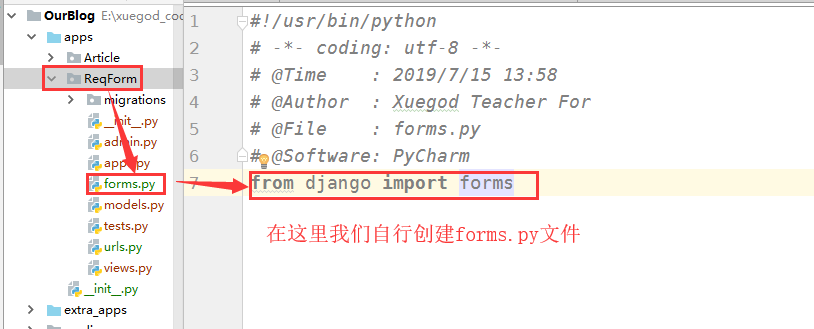
Form表单的主要作用：提供了完善的前后端封装，免去了开发人员进行js校验，数据整合，报错等的编写，让我们的编程更加快捷。

认识form类需要从以下三个方面开始

1、form表单的定义

form表单的定义通过Django下面的form类进行定义，过程和Django的模型定义类似。

无论下面的那种定义方法，建议各位同学首先在app目录当中创建form定义的专属文件form.py，然后开始我们的编写。



常规定义，我们按照与model模型定义类似的定义方法定义表单

from django import forms  
class UserForm(forms.Form):  
 username = forms.CharField(label='联系人姓名')  
 password = forms.CharField(min\_length=6,label='联系人密码')  
 email = forms.CharField(max\_length=32,label='联系人邮箱')  
 phone = forms.CharField(min\_length=11,label='联系人电话')

这样我们就定义了一个表单，这个表单和上面讲课用的联系人表单的效果是一模一样的。

当然惯用form字段里面常用的字段和对应的参数，也给大家列出来常用的字段:

1、charfield  对应单行输入框

2、booleanfield  对应checkbox选择框

3、choicefield 对应下拉菜单选择框   重要参数choice

4、datafield  对应一个单行输入框，但是会自动转化为日期类型

5、emailfield  对应输入框，自动验证是否是邮件地址

6、filefield  对应文件上传选项

7、filepathfiled 对应一个（文件组成的下拉菜单）选择，必须参数path=''.选项为这个地址里边所有的文件。可选参数recursive=True是否包含子文件夹里的文件

8、imagefield  图片上传按钮。需要pillow模块

9、urlfield  对应输入框 自动验证是否为网址格式

字段常用的参数

1、rqeuired    字段是否为必填 默认为True

2、label  类似于输入框前边的提示信息

3、initial 初始值（占位符）就是给出一个默认值

4、help\_text  字段的辅助描述

5、error\_message={}  覆盖｛｛form.name.error｝｝信息

6、disable  字段是否可以修改

7、widget 负责渲染网页上HTML表单的输入元素和提取提交的原始数据。

8、max\_length 最大长度

9、min\_length 最小长度

2、对应Django模型定义

为了保证我们定义的表单模型和数据库提交模型的一致性，我们可以通过Form表单类当中嵌套元类Meta来实现通过模型快速的定义表单

from django import forms  
from .models import User *#首先导入要对接的数据库*class UserForm(forms.ModelForm): *#注意，我们现在继承的是ModelForm而不是Forms* class Meta: *#这是一个元类* model = User*#指定models对应的模型* fields = ["username","password","email"] *#指定显示的字段*

当然，这么简单的代码只能完成最简单的效果，我们还需要一些常用的参数

fields 要展示的字段，接收的是一个列表，如果不写默认是models的所有字段  
exclude 不要展示的字段，接收的是一个列表，排除的字段，默认没有

widgets 是一个模型字段的名称对应的样式的字典  
labels 是一个模型字段的名称对应的标签的字典  
help\_texts 是一个模型字段的名称对应的帮助文档的字典  
error\_messages是一个模型字段的名称对应的错误提示的字典

### 11.3 form表单的调用和修饰

上面我们讲了如何来定义form表单，接下来学习form表单的使用。使用分为下面的两个步骤

表单的调用：

来定义一个基础的前端页面来作为表单的基本调用页面fromtest.html,用作表单测试

首先来看最常规的调用方法

from .forms import UserForm  
def formTest(request):  
 obj = UserForm()  
 return render(request,'fromtest.html',{'obj':obj})

然后在前端调用

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FromTest</title>

</head>

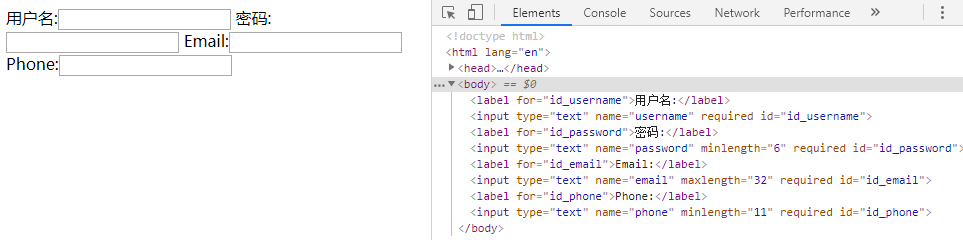
<body>

{{ obj }}<!--变量调用-->

</body>

</html>

结果如下



上面是最简单的调用，在form表单调用之初，有三种常用的格式，

as\_p用p标签来修饰表格的样式，前端调用如下

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FromTest</title>

</head>

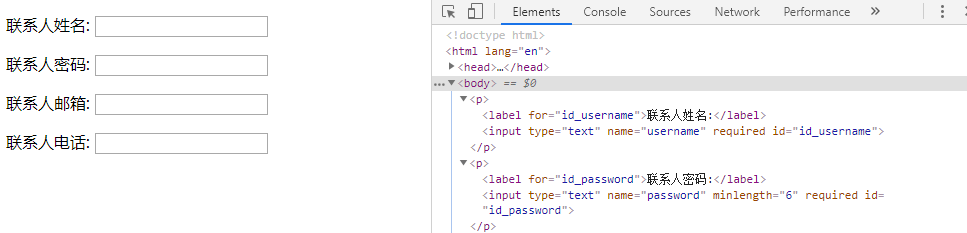
<body>

{{ obj.as\_p }}<!--变量调用-->

</body>

</html>

效果如下：



as\_ul 用li标签来修饰表格的样式,前端调用如下

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FromTest</title>

</head>

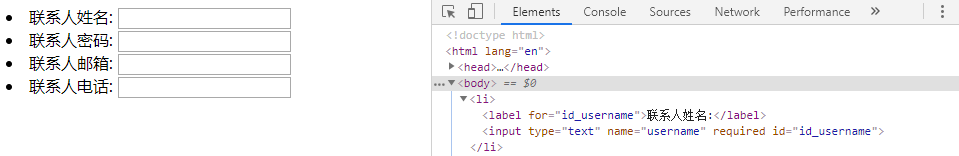
<body>

{{ obj.as\_ul }}<!--变量调用-->

</body>

</html>

效果如下



as\_table 在老的版本还是有很棒的效果的，就是用tr、td、th来修饰我们的表格，但是在1.6版本之后没有了效果，就如同我们直接调用obj一样

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FromTest</title>

</head>

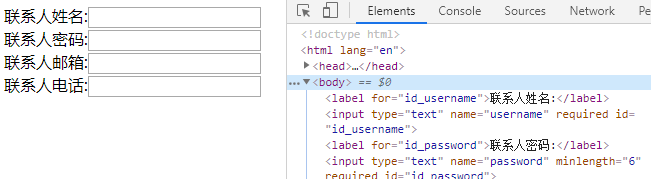
<body>

{{ obj.as\_table }}<!--变量调用-->

</body>

</html>

效果如下



上面粗略的调用了表单，首先要关注一个点，上面生成的表单，无论是任何格式，所有的标签都有id，而且id很有规律，所以，我们可以通过id捕获使用标签，当然，也留下了很多问题。我们来思考一下下面的两个问题

1. 为啥form表单的form标签和提交按钮，以及as\_ul当中的ul不存在

第一个问题是由于forms需要给开发者一个自由开发的空间，form标签和标签的action、method属性都交给开发者自己定义，至于提交按钮，可以让开发者自己定义submit常规提交还是js事件

同样ul也是为了灵活度，大家可以通过自定义ul获得通过ul给下面Django 生成的form元素添加css样式的机会。当然form标签你也可以这样做。

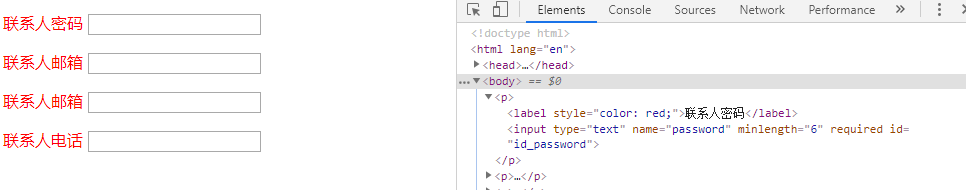
2. 如果想加入样式，该咋办

上面第一点已经给了大家定义表单样式的一种方法，但是这种方法太过繁复，其实我们还可以从两个角度来着手修改表单的样式：

form表单是可以单独调用具体的字段和标签的

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>FromTest</title>  
</head>  
<body>  
<p>  
 <label style = "color: red;">{{ obj.password.label }}</label>  
 {{ obj.password }}  
</p>  
<p>  
 <label style = "color: red;">{{ obj.email.label }}</label>  
 {{ obj.email }}  
</p>  
<p>  
 <label style = "color: red;">{{ obj.email.label }}</label>  
 {{ obj.email }}  
</p>  
<p>  
 <label style = "color: red;">{{ obj.phone.label }}</label>  
 {{ obj.phone }}  
</p>  
  
</body>  
</html>

效果如下



当然，如果你前面的知识学的很扎实，还可以对这段笨拙的代码进行调整

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FromTest</title>

</head>

<body>

{#……#}

{% for field in obj %}

<p>

<label style = "color: red;">{{ field.label }}</label>

{{ field }}

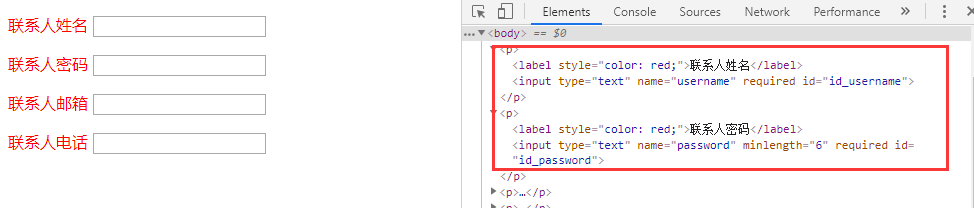
</p>

{% endfor %}

</body>

</html>

效果如下



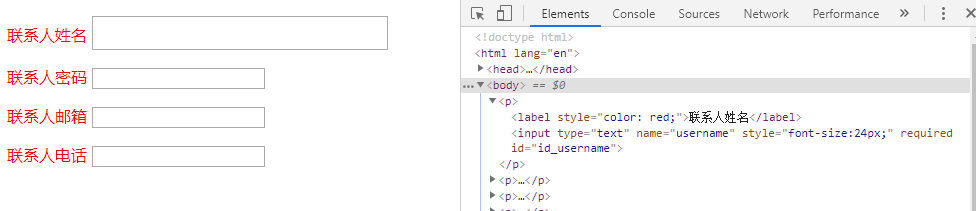
2、我们在定义forms表单的时候，通过weight属性也是可以定义样式的。

首先修改forms当中的定义

class UserForm(forms.Form):  
 username = forms.CharField(label='联系人姓名',widget=forms.TextInput(attrs = {"style":"font-size:24px;"}))

#......

效果如下



当然用models对应定义的方法也可以这么用

#coding:utf-8

from django import forms

from.models import User#首先导入要对接的数据库

class UserForm(forms.ModelForm): #注意，我们现在继承的是ModelForm而不是Forms

class Meta: #这是一个元类

model = User #指定models对应的模型

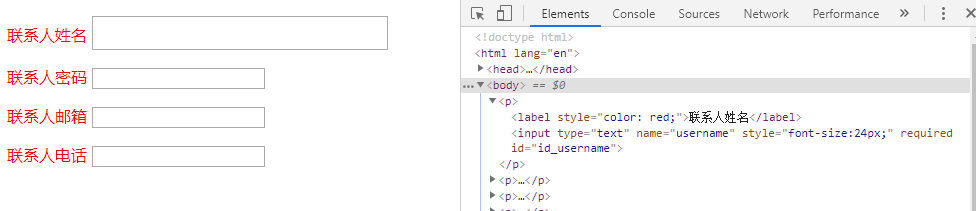
fields = ["username","phone","email"] #指定显示的字段

widgets = {

"con\_name": forms.TextInput(attrs = {"style":"font-size:24px;"})

} #指定对应标签的样式

效果如下



### 11.2.4 form表单校验

上面讲了表单的定义，那么下面来学习表单数据的提交校验，Django的form表单在刚刚拿出来的时候，好多同学认为只是对提供了方便的前端操作，实际上Django的form表单的标签元素本身自带校验给开发者提供了极大的便利，现在我们将我们的表单补充完整，进行提交。

在这里还是先强调，我们上述的两种表单定义都可以完成这样的共能，我们先使用对应Django模型定义的表单

from django import forms  
from .models import User  
class UserForm(forms.ModelForm):  
 class Meta:  
 model = User *#指定models对应的模型* fields = ["username","password","email",'phone'] *#指定显示的字段*

from表单前端补全，要注意post请求的csrf设置

<form action="/reqform/formtest/" method="post">  
 {% csrf\_token %}  
 {% for field in obj %}  
 <p>  
 <label style = "color: red;">{{ field.label }}</label>  
 {{ field }}  
 </p>  
 {% endfor %}  
 <p>  
 <input type = "submit" value = "提交">  
 </p>  
</form>

然后，是views视图接收form表单的数据。

from django.shortcuts import render

from Article.models import Connection #导入数据模型，准备存入数据

#......

def formExample(request):

obj = ConnectionForm()

if request.method == "POST" and request.POST: #检测请求的方式和数据

formdata = ConnectionForm(request.POST) #直接讲request.post的获取到的form表单数据放入表单类进行校验

if formdata.is\_valid(): #如果校验完全通过

data = formdata.cleaned\_data #返回字典类型的校验后的纯粹form表单提交的数据

return render(request,"formTest.html",{"obj": obj,"data": data,"statue":"sucess","error":{}})

else:

return render(request,"formTest.html",{"obj": obj,"data": None,"statue":"valid Error","error":formdata.errors})

else:

return render(request, "formTest.html", {"obj": obj,"data": None,"statue": "request error","error":{}})

在这里与要强调几个点：

1. 在处理请求数据之前一定要校验请求的类型和是否为空
2. 讲请求的数据直接放入表单类可以进行校验
3. is\_valid方法是判断表单校验的结果
4. clean\_data是返回判断后的数据
5. 大家在编写返回的时候，为了方便前端调用和前端调用的完整性，请返回统一的数据结构。

接着我们前端查看表单校验

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FromTest</title>

</head>

<body>

{#......#}

<form action="" method="post"> {# form标签 #}

{% csrf\_token %} {# csrf token #}

{% for field in obj %}

<p>

<label style = "color: red;">{{ field.label }}</label>

{{ field }}

</p>

{% endfor %}

<p>

<input type = "submit" value = "提交"> {# 提交按钮 #}

</p>

</form>

<p>

{{ statue }} {# 后端提交过来的状态 #}

<span>

{{ error }} {#后端提交过来的错误 #}

</span>

</p>

</body>

</html>

效果如下：

什么都不填写，触发错误的效果如下：



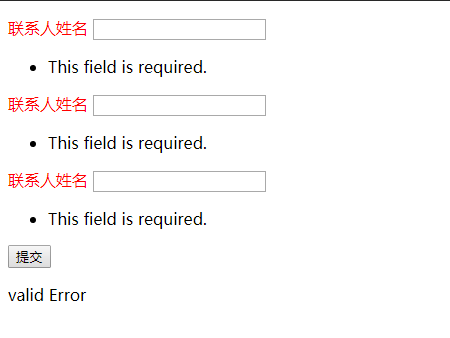


当然使用限制的知识需要“费点力气”，把报错的信息提交到form表单当中。

其他不变

Html代码如下

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>FromTest</title>  
</head>  
<body>  
*{#......#}*<form action="/reqform/formtest/" method="post"> *{# form标签 #}* {% csrf\_token %} *{# csrf token #}* <p>  
 <label style = "color: red;">{{ obj.username.label }}</label>  
 {{ obj.username }}  
 <span>  
 {{ error.username }} *{# 错误标签 #}* </span>  
 </p>  
 <p>  
 <label style = "color: red;">{{ obj.password.label }}</label>  
 {{ obj.password }}  
 <span>  
 {{ error.password }} *{# 错误标签 #}* </span>  
 </p>  
 <p>  
 <label style = "color: red;">{{ obj.phone.label }}</label>  
 {{ obj.phone }}  
 <span>  
 {{ error.phone }}  
 </span>  
 </p>  
 <p>  
 <label style = "color: red;">{{ obj.email.label }}</label>  
 {{ obj.email }}  
 <span>  
 {{ error.email }}  
 </span>  
 </p>  
 <p>  
 <input type = "submit" value = "提交"> *{# 提交按钮 #}* </p>  
</form>  
<p>  
 {{ statue }}  
</p>  
</body>  
</html>



## 11.4 表单提交的案例

到上面，我们完成了基础的form设置，那么，我们将我们学到的知识利用到我们刚刚开头举的那个表单的例子当中.

from django import forms  
class UserForm(forms.Form):  
 username = forms.CharField(label='联系人姓名')  
 password = forms.CharField(min\_length=6,label='联系人密码')  
 email = forms.CharField(max\_length=32,label='联系人邮箱')  
 phone = forms.CharField(min\_length=11,label='联系人电话')

views.py中编写：

from .models import User  
from .forms import UserForm  
def reqform(request):  
 if request.method == 'POST' and request.POST:  
 formdata = UserForm(request.POST)  
 if formdata.is\_valid():  
 data = formdata.cleaned\_data  
 username = data.get('username')  
 password = data.get('password')  
 email = data.get('email')  
 phone = data.get('phone')  
 User.objects.create(  
 username=username,  
 password=password,  
 email=email,  
 phone=phone  
 )  
 return JsonResponse({'status':'success'})  
 else:  
 error = formdata.errors  
 return render(request,'reqform.html',{'error':error})  
 else:  
 forms = UserForm()  
 return render(request,'reqform.html',locals())

reqform.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>FromTest</title>  
</head>  
<body>  
*{#......#}*<form action="/reqform/" method="post"> *{# form标签 #}* {% csrf\_token %} *{# csrf token #}* <p>  
 <label style = "color: red;">{{ forms.username.label }}</label>  
 {{ forms.username }}  
 <span>  
 {{ error.username }} *{# 错误标签 #}* </span>  
 </p>  
 <p>  
 <label style = "color: red;">{{ forms.password.label }}</label>  
 {{ forms.password }}  
 <span>  
 {{ error.password }} *{# 错误标签 #}* </span>  
 </p>  
 <p>  
 <label style = "color: red;">{{ forms.phone.label }}</label>  
 {{ forms.phone }}  
 <span>  
 {{ error.phone }}  
 </span>  
 </p>  
 <p>  
 <label style = "color: red;">{{ forms.email.label }}</label>  
 {{ forms.email }}  
 <span>  
 {{ error.email }}  
 </span>  
 </p>  
 <p>  
 <input type = "submit" value = "提交"> *{# 提交按钮 #}* </p>  
</form>  
<p>  
 {{ statue }}  
</p>  
</body>  
</html>

总结：

11.1 深入探索Request请求

11.2 form表单的定义

11.3 Django的form表单类

11.4 表单提交的案例