操作步骤

1. 创建命令：django-admin startproject [xx]
2. 创建一个应用：

python manage.py startapp [xx]

1. 运行命令：python manage.py runserver
2. 访问：输入网址：127.0.0.1：8000
3. 通过模型操作数据表：python

manage.py shell

模型类的对象x ， 对X进行操作想表中添加数据

From 模型类导入模型

1. 创建超级用户：python manage.py createsuperuser
2. 为了方便管理：admin 定义一个模型管理类
3. 数据迁移：python manage.py migrate 作用数据库

Python mange.py makemigrations 生成迁移文件

1. setting 中在ALLOWED\_HOSTS,添加部署的域名或者主机，允许使用的地址
2. ROOT\_URLCONF总体的路由控制
3. Templates 使用默认的方式查找APP = TRUE
4. Render 方法使用： 传递的请求， 模板， 内容；
5. Render里面的context变量是一个字典，在传递到的html文件中可以通过查找键的方式，查找出值。
6. Render\_to\_response不需要传递request参数
7. locals参数：可以将函数的所有参数传递到模板。
8. \*#\*#6485#\*#\* 查看手机充电状态--手机充电状态查看
9. 自定义一个类管理，代替objects: 使用方式，关键字传递参数，这个管理类也可以添加函数对数据库进行操作。
10. 指定一个表名使用元类即mate类，定义数据库写数据的时候，参数值。
11. 上传图片两种方式：
12. 使用imagefield进行上传
13. 用户自定义上传

常用的模板标签

1. {% if %}-----{% endif %} 假设
2. {% for a in s %}------{% endfor %} 遍历对象 反向遍历在s后面加一个reversed
3. “.”，字典的话对应的键值，如果是类的话则调用的是相应的对象。
4. 遍历字典需要加上键值对：For key, value in person.items

通用如果遍历的对象没有值，那么需要 ｛% empty%｝

1. Forloop.counter（可以添加多个值） 循环的下表以1为起始值。
2. With 把一个复杂的变量缓存到简单的变量上面，而且赋值的时候，不能在等号的左右留空格。
3. Url href属性定义的可以使用------

在传递参数的时候，位置参数和关键字参数两者只能用一。

如果想要查询必须在后面添加，并且使用？相连接：



多个参数传递，只需要在每个参数中间加空格即可

1. spaceless 移除标签中的空白字符，包括空格，tab和换行。
2. Autoescape 标签 ----{% autoescape on %}-------{% endautoscape %} 自动转义
3. Verbatim 标签------DTL解析关闭？

常用过滤器

1. Add-------｛value|add:”x”｝ 使用”|”代表过滤的意思，后面可以加上一些使用的过滤器方法，add 为加数字加法，字符串为添加在后面。
2. Cut ------移除指定的字符串 ｛value|cut:””｝
3. Date-----将日期转换成指定的格式 ｛birthday|data:”Y/m/d”｝
4. Default-----在判断中，使用这个的话，出现false会使用默认值。
5. Defaut-if-none 在值为nothing
6. First ---返回列表/元祖/字符串的第一个字符

{{value|first}} 如果这个值就是value是等于[] ,

则会取出其中的第一个值。

1. 相对来说last 为返回的最后一个元素。
2. Join 将列表/元祖/字符串 用指定的字符进行拼接。

{{value|join:”/”}} 如果value=[‘x’,’y’] , 输出x/y

1. length 长度
2. Random 随机从中选择一个数据
3. slice切片---对列表进行切片 {{value|slice:‘1’}}

将以上的列表进行切片，从第一个数字开始切片

**自定义函数封装：**

除了以上的封装好的函数之外，还可以将自定义一个函数封装成过滤器将数据封装进来。

例如： from django import template(自定义的模板在哪里就从哪里获得数据)

Re = template.Library()

Def mycut(value, mystr):

Return value.replace(mystr)

Re.filter(“mycut”, mycut)

使用 {% load my\_filter% } 定义的.py文件

**表单数据结构的应用和原理**

1. **其主要的作用：向服务器中添加数据，配合两个标签进行使用：**

**这两个标签分别是 input form ，form被Input所包含**

1. **在django中表单的主要作用是渲染模板，另外一方面会验证表单的数据是否合法。**

**根据请求的类型进行不同的处理：以下是一个表单类**

# forms.pyclass MessageBoardForm(forms.Form):

title = forms.CharField(max\_length=3,label='标题',min\_length=2,error\_messages={"min\_length":'标题字符段不符合要求！'})

content = forms.CharField(widget=forms.Textarea,label='内容')

email = forms.EmailField(label='邮箱')

reply = forms.BooleanField(required=False,label='回复')

**Get请求，往后台提交的是一个空的表单类。**

**Post请求，那么是将数据进行矫正和检查。**

# views.pyclass IndexView(View):

def get(self,request):

form = MessageBoardForm()

return render(request,'index.html',{'form':form})

def post(self,request):

form = MessageBoardForm(request.POST)

if form.is\_valid():

title = form.cleaned\_data.get('title')

content = form.cleaned\_data.get('content')

email = form.cleaned\_data.get('email')

reply = form.cleaned\_data.get('reply')

return HttpResponse('success')

else:

print(form.errors)

return HttpResponse('fail')

**使用get的时候，将一个空的form给模板，post会将提交的表单进行验证，如果验证通过的话，那么会获得里面的数据**

**差距在于，一个是直接传递到后台模板，另外一个是验证后传入响应的数据，一个全部传入，一个分类传入。**

**<form action="" method="post">**

**<table>**

**<tr>**

**<td></td>**

**<td><input type="submit" value="提交"></td>**

**</tr>**

**</table></form>**

**上面编写的时候，在最外层编写一个form标签，使用table可以将表格进行美化，所谓的h5语法更多的就是标签的语法。**

**下面是常见的几种表单验证的参数，类似于model类型的定义，**

**数据是需要和数据库的数据进行匹配的。**

**表单的验证文件： 根据提交的数据类型来判断使用什么样的field**

**Charfield 接受文本 属性：max\_length min\_length required error\_messages---错误信息报错**

**EmailField: 用来接受邮件，并且进行判断**

**floatField 接受浮点型数据，并且将参数进行转换成浮点型**

**URLField： 用来接受url格式的字符串 常见的信息有required , invaild**

**常见的几种验证器--------即以上验证参数的内嵌函数**

1. MaxValueValidator：验证最大值。
2. MinValueValidator：验证最小值。
3. MinLengthValidator：验证最小长度。
4. MaxLengthValidator：验证最大长度。
5. EmailValidator：验证是否是邮箱格式。
6. URLValidator：验证是否是URL格式。
7. RegexValidator：如果还需要更加复杂的验证，那么我们可以通过正则表达式的验证器：RegexValidator。比如现在要验证手机号码是否合格，那么我们可以通过以下代码实现：

**表单的参数进行，定义一个validators参数用来指定验证器**

class MyForm(forms.Form):

telephone = forms.CharField(**validators**=[validators.RegexValidator("1[345678]\d{9}",message='请输入正确格式的手机号码！')])

**自定义进行表单的验证：**

**对某个字段进行自定义验证的方式 。**

**在表单提交进行自定义的函数验证，如下：**

**其中方法命名有自己的规则 使用的clean\_fieldname**

class MyForm(forms.Form):

telephone = forms.CharField(validators=[validators.RegexValidator("1[345678]\d{9}",message='请输入正确格式的手机号码！')])

def clean\_telephone(self):

telephone = self.cleaned\_data.get('telephone')

exists = User.objects.filter(telephone=telephone).exists()

if exists:

raise forms.ValidationError("手机号码已经存在！")

return telephone

Clean\_telephone就是指定的函数。

其中exists封装好的函数方法。

clean方法可以自己进行定义：

class MyForm(forms.Form):

telephone = forms.CharField(validators=[validators.RegexValidator("1[345678]\d{9}",message='请输入正确格式的手机号码！')])

pwd1 = forms.CharField(max\_length=12)

pwd2 = forms.CharField(max\_length=12)

def clean(self):

cleaned\_data = super().clean()------这个clean()方法不知道是不是自己进行返回。

pwd1 = cleaned\_data.get('pwd1')

pwd2 = cleaned\_data.get('pwd2')

if pwd1 != pwd2:

raise forms.ValidationError('两个密码不一致！')

上面的clean方法，会将自己定义的函数重新调用，进行对比。

**关于验证失败：**

验证失败的时候需要定义下面的几个属性：

1. form.errors 2.form.erros.get\_json\_data()
2. form.as\_json

将表单返回的值按照指定的数据类型显示出来

表单的field和模型中的field类似，所以可以将两者进行绑定，直接在表单里面进行编写：

from django import forms

class MyForm(forms.ModelForm):

class Meta:

model = Article

fields = "\_\_all\_\_"

上面中，Article是从模型类里面导入的函数

因为fields为all所以讲整个模型中的字段赋值过来

同样如果想要获取几个字段，那么可以把它写成一个列表

Field = [‘title’ , ‘name’]

Exclude= [‘title’]==========将除了title的所有字段全部代入表单

在表单中可以自定义错误抛出的信息：class MyForm(forms.ModelForm):

class Meta:

model = Article

exclude = ['category']

error\_messages ={

'title':{

'max\_length': '最多不能超过10个字符！',

'min\_length': '最少不能少于3个字符！'

},

'content': {

'required': '必须输入content！',

}

}

表单进行调用save方法的时候，需要使用clean进行验证，如果成功之后，之后会将数据写入数据库，clean方法，需要进行测试。

form标签中的action属性是指定表单的数据的提交方向

前端中进行数据的上传的时候需要在form标签中添加一个属性----enctype=’multipart/form-data’

<form action="" method="post" enctype="multipart/form-data">

<input type="file" name="myfile"></form>

表单的提交使用的是post属性。

后端代码的实现：后端工作是进行接收文件，并且接收的方式和post是一样的，不过需要使用files来实现：

# models.pyclass Article(models.Model):

title = models.CharField(max\_length=100)

content = models.TextField()

thumbnail = models.FileField(upload\_to="files")

# views.pydef index(request):

if request.method == 'GET':

return render(request,'index.html')

else:

title = request.POST.get('title')

content = request.POST.get('content')

thumbnail = request.FILES.get('thumbnail')

article = Article(title=title, content=content, thumbnail=thumbnail)

article.save()

return HttpResponse('success')

Article.save()方法使用之后，会将文件保存在files下面，并且会把这个文件的路径存到数据库中。

可以指定media\_root和media\_url进行文件的指定存储，

使用upload\_to来指定上传的文件目录，首先需要在setting中配置media的路径：

from django.urls import pathfrom front import viewsfrom

django.conf.urls.static import staticfrom

django.conf import settings

urlpatterns = [

path('', views.index),

] + static(settings.MEDIA\_URL,document\_root=settings.MEDIA\_ROOT)

如果我们同时指定MEDIA\_ROOT和upload\_to，那么会将文件上传到MEDIA\_ROOT下的upload\_to文件夹中。示例代码如下：

class Article(models.Model):

title = models.CharField(max\_length=100)

content = models.TextField()

thumbnail = models.FileField(upload\_to="%Y/%m/%d/")

上面的代码实在model类中进行编写时候使用。

可以使用form对上传的文件扩展名进行限制：

# forms.pyclass ArticleForm(forms.ModelForm):

class Meta:

model = Article 主要使用的就是Modelform方法

fields = "\_\_all\_\_"

上传图片的原理和普通文件的上传是同一个类型的，不同的是就是上传的时候，jango会判断是不是图片。

因此在上传文件之前，需要首先定义一个imagefield模型，

thumbnail = models.ImageField(upload\_to="%Y/%m/%d/")

写在一个模型类的里面。

还要使用表单类进行验证，同样需要继承modelform类。

通过数据库的表执行此命令，生成对应的model

python manage.py inspectdb

Restful 接口风格：

不论什么样的资源都能通过相同接口进行资源访问

资源对应一个独特的url，想要获取此资源，直接访问URL即可；

状态转换：get post put delete

动词设计需要和状态转换中的定义的相同

Pymysql连接数据库，通过对数据库读取

1. **扩展名**，比如.json表示我要JSON格式数据、.xml表示我要xml格式数据
2. **请求参数**：默认是“format”
3. **请求头设置Accept参数**，比如设置Accept为application/json表示要JSON格式数据

基本访问方法都是GET,POST,PUT