Django阶段性总结

**1、Django简介**

Django是一个开放源代码的Web应用框架，由Python写成。采用了MTV的框架模式，即模型M，模板T和视图V。它最初是被开发来用于管理劳伦斯出版集团旗下的一些以新闻内容为主的网站的，即是CMS（内容管理系统）软件。并于2005年7月在BSD许可证下发布。这套框架是以比利时的吉普赛爵士吉他手Django Reinhardt来命名的。

**2、Django使用框架模式(MTV)**

**M:Model** --> 负责数据库的CRUD，表结构的定义；

**T:Template** --> 负责把渲染的页面及特定的内容展示给用户；

**V:View** --> 负责业务逻辑处理，Model与Template间的桥梁。Django中还有一个url分发器，也叫作路由。主要用于将url请求发送给不同的View并进行相关的业务逻辑处理。

**3、Django的常规操作**

1）**pip install virtualenv**: 安装virtualenv(虚拟环境)（注：创建pycharm项目时需要指定虚拟环境，可以将不同版本的项目隔离开来，项目上线时要将环境和项目分离开）；

2）**pip install django==1.11**: 安装django环境；

3）**virtualenv --no-site-packages xxx** : 安装env环境；

xxx --> 指定python版本所在的路径。没有则使用默认版本的python解释器--no-site-packages –> 纯净的虚拟环境，不包括外界的包/模块(Pycharm创建的虚拟环境自带多种包/模块)

4）**cd envname/Scripts --> activate | deactivate**： 进入|退出env；

5）**django-admin startproject 项目名**: 创建django项目，项目文件夹名及其中的工程名一致；

Project(工程内部文件简介)：

\_\_init\_\_.py: 初始化，配置pymysql连接的地方

setting.py: 配置文件，具体配置项目如下：

-- 在INSTALLED\_APPS下添加包/模块，第三方库（如：rest\_framework）

-- 在MIDDLEWARE添加再定义中间件(如：访问前登录),在POST提交时注意crsf中间件的存在

-- 在TEMPLATES中配置承载前端页面的templates路径，并在工程同级处创建templates文件；

-- 在DATABASES中配置数据库相关参数，同时在项目外创建相应utf8编码类型的数据库（数据库名称，用户，密码等参数需保持一致）

-- LANGUAGE\_CODE：配置语言(如:zh-hans)，TIME\_ZONE：配置时区 ('Asia/Shanghai')

-- 在STATIC\_URL，STATICFILES\_DIRS中配置静态文件路径，用于存放CSS、JS、IMAGE,fonts相关文件

-- MEDIA\_URL、MEDIA\_ROOT中配置上传文件(如:头像)的路径，使用前先转为动态(static)文件并加至路由(urlpatterns)中

-- 在REST\_FRAMEWORK中配置restframework相关信息，如分页等

urls.py: 配置路由信息，可设置命名空间namespace

wsgi.py: 网关

6）**python manage.py startapp hello\_app**: 创建app(模块)。该命令是在工程下创建一个名为hello\_app的app，并与工程同级

app(模块内部文件简介):

urls.py(自创): url路由，给每个请求寻找对应的方法

admin.py: 管理后台、注册模型

apps.py: settings.py里面注册app的时候需要使用到，一般不推荐直接使用

models.py:存放模型的地方，定义数据库中的表结构

views.py:写处理业务逻辑的地方

7）**settings**

7.1）在settings.py文件中INSTALLED\_APPS中写入app name

7.2）修改databases

7.3）修改templates -- os.path.join(BASE\_DIR, 'templates')

7.4）设置语言 -- LANGUAGE\_CODE = 'zh-hans'

7.5）修改时区 -- TIME\_ZONE = 'Asia/Shanghai'

8）**建立模型 --** 定义的模型继承于models.Model

A 模型字段：

-- CharField: 字符串

-- IntegerField: 整数

-- FloatField: 浮点数

-- BooleanField: 布尔类型(1,0)

-- DateField(auto\_now\_add=True/auto\_now=True): 存放日期（年月日）

auto\_now\_add=True:第一次创建数据时设置时间(1次)

auto\_now=True:每次登录、修改操作都会更新到最新时间(多次)

-- DateTimeField:年月日 时分秒

auto\_now\_add：同上

auto\_now：同上

-- AutoField: 自动增长

-- DecimalField:

models.DecimalField(max\_digits=3,decimal\_places=1) 最大值99.9

max\_digits: 总位数

decimal\_places: 小数点后的位数

-- TextField: 存长文本信息，页面等

-- FileField: 文件上传字段

-- ImageField(upload\_to=''): 上传图片

upload\_to="xxx":指定上传图片的路径

B 模型参数:

-- default='xxx':设置默认值

null=True: 设置是否为空，针对数据库中该字段是否可以为空

blank=True:设置是否为空，针对表单提交中该字段是否可以为空

-- primary\_key=True:创建主键

-- unique:唯一

-- max\_length: 最大长度

1. -- choices,示例如下：

在student模型中添加就读状态属性，使用choices字段使得值只能从STATUS列表中选取(注:STATUS为规定关键字，其类型只能为列表或元组,可以强制转换为字典从而采用键值对的方式进行操作)

Model.py

STATUS =[

('NONE','正常'),

('NEXT\_SCH','留级'),

('DROP\_SCH','退学'),

('LEAVE\_SCH','休学')

]

s\_status = models.CharField(choices=STATUS,max\_length=20,null=True)

Views.py

status = request.GET.get("status") # 状态

status\_dict = dict(Student.STATUS) --- {‘NONE’:’正常’ ......}

for key,value in status\_dict.items():

if value == status:

stus = Student.objects.filter(s\_status=key)

9）**加载数据库**

import pymysql

pymysql.install\_as\_MySQLdb()：当成是mysqldb一样使用，当然也可以不写这句，那就按照pymysql的方式

10）**数据库迁移**

python manage.py makemigrations

python manage.py migrate

python manage.py makemigrations stu:强制找到stu模块进行迁移

Navicat中指定数据库中的表django\_migrations记录了所有迁移记录

注:Django里面自带user -- request里的user默认为AnonymousUser,其中的id,username等属性为空

11）**配置DEBUG**

Run – Debug – Edit Configuration – + – Python – 配置Script path(manage.py) Parameters(runserver 8000) -- Apply

12）**启动项目**

python manage.py runserver [ip:](默认127.0.0.1)端口号(默认8000端口):启动django项目，端口号可以不用写，启动的时候会默认随机创建一个可以使用的端口

13）（\*暂时可有可无）**创建超级管理员的帐号和密码**

python manage.py createsuperuser

1. **Django过滤**

**科普知识：ORM(objects relational mapping) -- 对象关系映射，翻译机**

1）通过modelname.objects来实现数据的增删查改(CRUD)操作

.all() – 获取全部信息

.filter() – 获取指定信息(.first():获取过滤后的第一条信息，.last():… 最后一条 …

.create() – 创建指定信息

.get() -- 只能获取一条数据，获取不到或数据有多条则会报错，不建议使用

2）F()和Q()函数

F() -- F()允许Django在未实际链接数据的情况下具有对数据库字段的值的引用。通常情况下我们在更新数据时需要先从数据库里将原数据取出后方在内存里，然后编辑某些属性，最后提交。

例1

常规操作：

order = Order.objects.get(orderid='123456789')

order.amount += 1

order.save()

使用F()方法：

from django.db.models import F

from core.models import Order

order = Order.objects.get(orderid='123456789')

order.amount = F('amount') - 1

order.save()

例2

有一张表,保存着公司员工的工资,公司普涨工资，代码如下：

from django.db.models import F

UserInfo.objects.filter().update(salary=F('salary')+500)

Q() -- Q对象(django.db.models.Q)可以对关键字参数进行封装，从而更好地应用多个查询。可以组合使用 &（and）,|（or），~（not）操作符，当一个操作符是用于两个Q的对象,它产生一个新的Q对象。

例

代码

Order.objects.get(

Q(desc\_\_startswith='Who'),

Q(create\_time=date(2016, 10, 2)) | Q(create\_time=date(2016, 10, 6))

)

相应的SQL语句

SELECT \* from core\_order WHERE desc LIKE 'Who%' AND (create\_time = '2016-10-02' OR create\_time = '2016-10-06')

注：Q对象可以与关键字参数查询一起使用，不过一定要把Q对象放在关键字参数查询的前面

例

正确写法

Order.objects.get(

Q(create\_time=date(2016, 10, 2)) | Q(create\_time=date(2016, 10, 6))

desc\_\_startswith='Who',

)

错误写法

Order.objects.get(

desc\_\_startswith='Who',

Q(create\_time=date(2016, 10, 2)) | Q(create\_time=date(2016, 10, 6))

)

**5、关联**

1） **1:1 OneToOneField**: 主键和外键是一对一关系，在关联表中，只能关联一个主表的id (一般添加到扩展表，关联到主表)

拓展表找主表 -- 拓展信息对象.关联字段

主表找拓展表 -- 主表对象.关联表的modelname(小写)

2）1:N OneToManyField(用户、订单)

拓展表找主表 -- 拓展信息对象.关联字段

主表找拓展表 -- 主表对象.关联表的modelname(小写)\_set

3) N:N MangToManyField

A在数据库表中的多对多,有两种方式:

方式1 – 自定义第三张表

class B2G(models.Model):

b\_id = models.ForeignKey('Boy')

g\_id = models.ForeignKey('Girl')

class Boy(models.Model):

name = models.CharField(max\_length=16)

class Girl(models.Model):

name = models.CharField(max\_length=16)

方式2 –使用models中自带的ManytoManyField自动创建第三张表

class Boy(models.Model):

name = models.CharField(max\_length=16)

class Girl(models.Model):

name = models.CharField(max\_length=16)

b = models.ManyToManyField('Boy')

注：使用多对多自动创建后,会创建第三张表,第三张表中会将操作的前两张表中的ID做对应

B 多对多查询

#正向查询

#获取一个女孩对象

g1 = Girl.objects.get(id=1)

#获取和当前女孩有关联的所有男孩

g1.b.all() #获取全部

#反向查询

b1 = Boy.objects.get(id=1)

b1.girl\_set.all() #获取全部

#连表查询

#正向连表

Girl.objects.all().values('id','name','b\_\_username')

#反向连表

Boy.objects.all().values('id','name','girl\_\_name')

#注意此处为girl\_\_name,并非girl\_set\_\_name.

1. **前端操作**

**科普知识：Jinja2 --> Python下一个被广泛应用的模版引擎，他的设计思想来源于Django的模板引擎，并扩展了其语法和一系列强大的功能。其中最显著的一个是增加了沙箱执行功能和可选的自动转义功能，这对大多应用的安全性来说是非常重要的。它基于unicode并能在python2.4之后的版本运行，包括python3**

过滤器（|）:在变量显示前修改

example: 语文成绩：{{stu.stu\_chinese | add:10}} -- add:加法，增加值

时间 date:（在数据库中时间会比页面显示中少8个小时，操作时需要注意）

Y-完整年份(四位年) m:月

y-简写年份(两位年） d:日

H-时 24小时制 m:分

h-时 12小时制 s:秒

example: 创建时间：{{ stu.stu\_create\_time | date:'Y-m-d H:m:s' }}

注释

{# #}:单行注释(<!-- -->)

{% comment %}{% endcommet %}:多行注释

大小写

upper | lower

example : 姓名：{{ stu.stu\_name | upper}}

分数乘法运算

widthratio 数 分母 分子

example: 语文成绩乘以10：{% widthratio stu.stu\_chinese 1 10 %}

<!-- stu.stu\_chinese \* 10(分子) / 1（分母）-->

整除运算

divisibleby:2 整除2，返回True,否则返回False

example: 数学成绩：{{ stu.stu\_math | divisibleby:2}}

<!-- 判断数学成绩是否能整除2 -->

命名空间

{% url 'namespace:name' value %}

工程url中定义namespace

模块url中定义name

静态资源加载

1）<img src="/static/images/xxx.png">

2） {% load static %}

<img src="{% static 'images/enemy1.png' %}">

for

{% for stu in stus %}

{% empty %} # for中内容为空时执行的操作

{% endfor %}

if

{% if xxx %}

{% else %}

{% endif %}

ifequal -- 如果相等时的操作

{% ifequal xxx 1 %}

{% else %}

{% endifequal %}

forloop

计数从0开始: {{ forloop.counter0 }}

计数从1开始: {{ forloop.counter }}

计数从最后开始，到1停: {{ forloop.revcounter }}

计数从最后开始，到0停: {{ forloop.revcounter0 }}

**7、REQUEST处理**

post -- 提交数据隐藏了

get -- 提交数据在url上, ?后跟参数，&用来连接多个参数，但是对参数的数量有限制，每个浏览器的限制不同

put 更新全部数据

patch 更新局部数据

delete:删除数据

example: stu\_id = request.GET.get('stu\_id')

Student.objects.filter(id=stu\_id).delete()

**8、COOKIE**

cookie: 随着url移动，在浏览器

session: 在数据库

set\_cookie(key,value,seconds): 设置cookie，并为其绑定令牌，令牌有过期时间，可以设置并存放在mysql或mongodb

1）绑定令牌到cookie里面，并将cookie存在前端

ticket = 'lalala'

response = HttpResponse()

response.set\_cookie('ticket',ticket)

return response

2）cookie存在数据库中

user.u\_ticket = ticket

user.save()

3）从浏览器及数据库中删除cookie

response = HttpResponseRedirect("/shopapp/login/")

ticket = request.COOKIES.get("ticket")

response.delete\_cookie("ticket")

UserSession.objects.get(session\_data=ticket).delete()

return response

1. **Ajax**

前端JS

function add(){

csrf = $('input[name="csrfmiddlewaretoken"]').val();

s\_name = $('#s\_name').val();

s\_tel = $('#s\_tel').val();

$.ajax({

url:'/stuapp/student/',

type:'POST',

data:{'s\_name':s\_name,'s\_tel':s\_tel}, //向后端传递数据，以字典的形式

dataType:'json',

headers:{'X-CSRFToken':csrf},

success:function (msg) {

alert('添加成功')

},

error:function () {

alert('添加失败')

}

});

}

前端HTML页面

{% csrf\_token %}

后端逻辑处理函数

JsonResponse(data) data不能是QuerySet

-- form表单提交(POST) 删除(DELETE) 局部更新(PATCH)

1. 开启中间件django.middleware.csrf.CsrfViewMiddleware
2. {% csrf\_token %},等同如下:

<input type='hidden' name='csrfmiddlewaretoken' value='duRoaLbyewAO7aLxV0HZyOWB9npPqLs01a27Y3nnqg2h8lTJjkcQdeiKzfpZJdX0' />

1. (ajax之外用不到

)csrf = $('input[name="csrfmiddlewaretoken"]').val()

headers:{'X-CSRFToken': csrf}

上述部分缺失会出现403 Forbidden错误