导航

博客园
首　页
新随笔
联　系
订　阅 XML
管　理

< 2019年3月 >

日	一	二	三	四	五	六
24	25	26	27	28	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

公告

昵称：大蛇Python
园龄：1年4个月
粉丝：16
关注：16
+加关注

搜索

找找看

Django 千锋培训的学习笔记（2）

Django 千锋培训读书笔记

https://www.bilibili.com/video/av17879644/?p=1

切换到创建项目的目录　cd C:\Users\admin\Desktop\DjangoProject

创建名为project的项目命令　django-admin startproject project

注：所有路径不要有中文

切换到目录cd C:\Users\admin\Desktop\DjangoProject\project

目录层级说明：manage.py　一个命令行工具，可以让我们用多种方式对Django项目进行交互

__init__.py 一个空文件，它告诉Python这个目录应该被看做一个包

settings.py 项目的配置文件（主要处理文件）

urls.py　　项目的url声明　（主要处理文件）

wsgi.py　　项目与WSGI兼容的Web服务器入口

配置数据库　Django默认使用SQLite数据库

在settings.py文件中通过DATABASES选项进行数据库配置

配置MySQL　Python3.x中安装的是PyMySQL

在__init__.py文件中写入两行代码import pymysql

pymysql.install_as_MySQLdb()

以数据库sunck为例进行示范：对settings.py中的DATABASES进行设置

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': "sunck",
        'USER': "root",
        'PASSWORD': "admin123",
        'HOST': "localhost",
        'PORT': "3306"
    }
}
```

创建应用--在一个项目中可以创建多个应用，每个应用进行一种业务处理

打开CMD,进入project(目录名)的目录下，输入命令创建名为myApp的app:

```
python manage.py startapp myAPP
```

myAPP目录说明

admin.py 进行站点配置

models.py 创建模型

views.py 创建视图

激活应用 在settings.py文件中，将myApp应用加入到INSTALLED_APPS选项中

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myApp',
]
```

定义模型 概述：有一个数据表就对应有一个模型

在models.py文件中定义模型

引入：`from django.db import models`

模型类要继承models.Model类

1. Re:Python头脑风暴4
服务业的最高境界是私人定制,有大佬说过的....
--大蛇Python
2. Re:Python头脑风暴3
互联网巨头下一步可能是驾校了...
--大蛇Python
3. Re:Python头脑风暴2
不知道淘宝与京东有无相关防御,本人没用过,本着说明的心态发此博客,胆子小没见过世面求别上加急名单
--大蛇Python
4. Re:Primer C++第五版读书笔记(一)
抄书
--鲦
5. Re:抽屉网点赞爬虫
404 Not FoundThe requested URL was not found on this server. Sorry for the inconvenience.Please repo.....
--allenfeng33

阅读排行榜

1. Django 千锋培训的学习笔记（1）(6651)
2. Django 千锋培训的学习笔记（2）(1443)
3. [原创]使用python对视频/音频文件进行详细信息采集,并进行去重操作(408)

示例：

```
class Grades(models.Model):
    gname = models.CharField(max_length=20)
    gdate = models.DateTimeField()
    ggirlnum = models.IntegerField()
    gboynum = models.IntegerField()
    isDelete = models.BooleanField(default=False)


class Students(models.Model):
    sname = models.CharField(max_length=20)
    sgender = models.BooleanField(default=True)
    sage = models.IntegerField()
    scontend = models.CharField(max_length=20)
    isDelete = models.BooleanField(default=False)
    sgrade = models.ForeignKey("Grades", on_delete=models.CASCADE,)
```

说明：

不需要定义主键，在生成时自动添加，并且值为自动增加

在数据库中生成数据表

生成迁移文件

执行 `python manage.py makemigrations` 在migrations目录下生成一个迁移文件，此时数据库中还没有生成数据表

执行迁移

执行 `python manage.py migrate` 相当于执行MySQL语句创建了数据表

测试数据操作

进入到python shell

执行 `python manage.py shell`

引入包

```
from myApp.models import Grades, Students
from django.utils import timezone
from datetime import *
```

4. 如何在微信中发送"相册"文件时有选择性地显示视频文件(274)
5. Python学习网站推荐(272)

评论排行榜

1. Python头脑风暴4(1)
2. Python头脑风暴3(1)
3. Python头脑风暴2(1)
4. Primer C++第五版 读书笔记(一)(1)
5. 抽屉网点赞爬虫(1)

推荐排行榜

1. Python 3.52官方文档翻译http://usyiyi.cn/translate/必看！(3)
2. JavaScript CSS 等前端推荐(1)
3. 分享一个编程学习网站:https://github.com/justprogramming-books-zh_C
4. Django 千锋培训的学习笔记（2）(1)
5. Python学习网站推荐(1)

查询所有数据

类名.objects.all()

示例： Grades.objects.all()

添加数据

本质：创建一个模型类的对象实例

示例：CMD窗口下：

```
grade1 = Grades()
grade1.gname = "python04"
grade1.gdate = datetime(year=2017, month=7, day=17)
grade1.ggirlnum = 3
grade1.gboynum = 70
grade1.save()
```

查看某个对象

类名.objects(pk=索引号)

示例：

```
Grades.objects.get(pk=2)
Grades.objects.all()
```

修改某个数据

模型对象属性 = 新值

示例：

```
grade2.gboynum = 60
grade2.save()
```

删除数据

模型对象.delete()

```
grade2.delete()
```

注意：这是物理删除，数据库中的相应数据被永久删除

关联对象

示例：

```
stu = Students()
stu.sname = "Xue Yanmei"
stu.sgender = False
```

```
        stu.sage = 20
        stu.scontend = "I am Xue Yanmei"
        stu.sgrade = grade1
        stu.save()
```

获得关联对象的集合

需求：猎取python04班级的所有学生

对象名.关联的类名小写_set.all()

示例：grade1.students_set.all()

需求:创建曾志伟，属于python04班级

示例：

```
stu3 = grade1.students_set.create(sname=u'Zhen Zhiwei',sgender=True,scontend=u"I am Zhen Zhiwei",sage=45)
```

注意：这样创建的数据直接被添加到了数据库当中。

启动服务器：

格式：python manage.py runserver ip:port

注意：ip可以不写，不写代表本机ip

端口号默认是8000

```
python manage.py runserver
```

说明：

这是一个纯python编写的轻量级web服务器，仅仅在开发测试中使用这个

Admin站点管理：

概述：

内容发布：负责添加，修改，删除内容的

公告访问

配置Admin应用：

在settings.py文件中的INSTALLED_APPS中添加'django.contrib.admin',

这条默认是添加好的。

创建管理员用户：

在项目目录下执行 python manage.py createsuperuser

依次输入账号名，邮箱，密码即可完成用户创建

登陆：

http://127.0.0.1:8000/admin/

汉化：

把project\settings.py

中作如下设定：LANGUAGE_CODE = 'zh-Hans'

TIME_ZONE = 'Asia/Shanghai'

管理数据表：

修改 myAPP\admin.py 如下：

```
from django.contrib import admin
# Register your models here.
from .models import Grades, Students
# 注册
admin.site.register(Grades)
admin.site.register(Students)
```

自定义管理页面：

属性说明

```
# 列表页属性
list_display = [] # 显示字段设置
list_filter = [] # 过滤字段设置
search_fields = [] # 搜索字段设置
list_per_page = [] # 分页设置
# 添加，修改页属性
fields = [] # 规定属性的先后顺序
fieldsets = [] # 给属性分组 注意：fields与fieldsets不能同时使用
```

属性示例：

```
# 列表页属性
list_display = ['pk', 'gname', 'gdate', 'ggirlnum', 'gboynum', 'isDelete']
list_filter = ['gname']
search_fields = ['gname']
list_per_page = 5
# 添加，修改页属性
# fields = ['ggirlnum', 'gboynum', 'gname', 'gdate', 'isDelete']
```

```
    fieldsets = [
        ("num",{"fields":['ggirlnum', 'gboynum']}),
        ("base", {"fields":["gname", "gdate", "isDelete"]}),
    ]
```

关联对象：需求：在创建一个班级时可以直接添加几个学生

```
    class StudentsInfo(admin.TabularInline):# 可选参数admin.StackedInline
        model = Students
        extra = 2
    class GradesAdmin(admin.ModelAdmin):
        inlines = [StudentsInfo]
```

布尔值显示问题示例：

```
    class StudentsAdmin(admin.ModelAdmin):
        def gender(self):
            if self.sgender:
                return "男"
            else:
                return "女"
        # 设置页面列的名称
        gender.short_description = "性别"
        list_display = ['pk', 'sname', 'sage', gender,
                        'scontend', 'sgrade', 'isDelete']
        list_per_page = 10
    admin.site.register(Students, StudentsAdmin)
```

执行按钮位置：

```
    class StudentsAdmin(admin.ModelAdmin):
        ...snip...
        actions_on_top = False
        actions_on_bottom = True
    admin.site.register(Students, StudentsAdmin)
```

使用装饰器完成注册：

```
    @admin.register(Students)
```

```
class StudentsAdmin(admin.ModelAdmin):
    def gender(self):
    ...snip...
    actions_on_top = False
    actions_on_bottom = True
```

视图的基本使用

概述：

在Django中，视图是对web请求进行回应

视图就是一个python函数，在views.py文件中定义。

定义视图：

示例：在myApp\views.py中写入

```
from django.shortcuts import render
# Create your views here.
from django.http import HttpResponse
def index(request):
    return HttpResponse("Sunck is a good man")
```

配置url：方法一：path方法：

修改project目录下的urls.py文件：

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myApp.urls')),
]
```

在myApp应用目录下创建urls.py文件：

```
from django.urls import path, include
from . import views
urlpatterns = [
    path('',views.index),
]
```

配置url：方法二：url方法：

修改project目录下的urls.py文件：

```
from django.contrib import admin
from django.conf.urls import url,include
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^', include('myApp.urls')),
]
```

在myApp应用目录下创建urls.py文件：

```
from django.conf.urls import url
from . import views
urlpatterns = [
    url(r'^$', views.index),
]
```

模板的基本使用：

概述：模板是HTML页面，可以根据视图中传递过来的数据进行填充

创建模板：

创建templates目录，在目录下创建对应项目的模板目录（project/templates/myApp）

配置模板路径：

修改settings.py文件下的TEMPLATES下的'DIRS'为'DIRS': [os.path.join(BASE_DIR, 'templates')],

定义grades.html与students.html模板：

在templates\myApp\目录下创建grades.html与students.html模板文件

模板语法：

{{输出值，可以是变量，也可以是对象，属性}}

{%执行代码段%}

http://127.0.0.1:8000/grades

写grades.html模板：

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
                <meta name="viewport"
                      content="width=device-width, user-scalable=no, initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0">
                <meta http-equiv="X-UA-Compatible" content="ie=edge">
                <title>班级信息</title>
            </head>
            <body>
                <h1>班级信息列表</h1>
                <ul>
                    <!--[python04, python05, python06]-->
                    {%for grade in grades%}
                    <li>
                        <a href="#">{{grade.gname}}</a>
                    </li>
                    {%endfor%}
                </ul>
            </body>
            </html>
```

定义视图：myApp\views.py

```
    from .models import Grades
    def grades(request):
        # 去模板里取数据
        gradesList = Grades.objects.all()
        # 将数据传递给模板,模板再渲染页面，将渲染好的页面返回给浏览器
        return render(request, 'myApp/grades.html', {"grades": gradesList})
```

配置url:myApp\urls.py

```
urlpatterns = [
    url(r'^$', views.index),
    url(r'^(\d+)/(\d+)$', views.detail),
    url(r'^grades/', views.grades)
]
```

http://127.0.0.1:8000/students

写students.html模板

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>学生页面</title>
</head>
<body>
    <h1>学生信息列表</h1>
    <ul>
        {%for student in students%}
        <li>
            {{student.sname}}--{{student.scontend}}
        </li>
        {%endfor%}
    </ul>
</body>
</html>
```

定义视图：myApp\views.py

```
from .models import Students
def students(request):
    studentsList = Students.objects.all()
    return render(request, 'myApp/students.html', {"students": studentsList})
```

配置url:myApp\urls.py

```
urlpatterns = [
url(r'^$', views.index),
```

```
            url(r'^(\d+)/(\d+)$', views.detail),
            url(r'^grades/', views.grades),
            url(r'^students/', views.students),
        ]
```

需求:点击班级，显示对应班级的学生名字

运行不正常https://www.bilibili.com/video/av17879644/?p=12

创建项目流程

Django流程梳理

创建工程：执行 django-admin startproject 工程名

创建项目：执行 python manage.py startapp 项目名称

激活项目：修改 settings.py中的INSTALLED_APPS

配置数据库：

修改__init__.py文件

修改settings.py文件中的DATABASES

创建模型类：在项目目录下的models.py文件中创建

生成迁移文件：执行python manage.py makemigrations

执行迁移：执行python manage.py migrate

配置站点：略

创建模板目录/项目模板目录

在settings.py中的TEMPLATES添加templates路径

在工程目录下（project）修改urls.py

在项目目录下创建urls.py

使用他人Django代码需要的简易修改：

1.在settings.py中修改数据库名

模型

2.在settings.py中修改数据库密码
3.删除由内向外文件（在对应目录里鼠标右键删除）
4.在数据库中创建对应第一步的数据库(自己在SQL中创建)
5.执行生成迁移文件
6.执行迁移
7.启动服务
8.浏览器测试

Django模型

Django对各种数据库提供了很好的支持，Django为这些数据库提供了统一的调用API
我们可以根据不同的业务需求选择不同的数据库。
配置数据库
修改工程目录下的__init__.py文件

```
import pymysql
pymysql.install_ad_MySQLdb()
```

修改settings.py文件中的DATABASES
开发流程
配置数据库
定义模型类：一个模型都在数据库中对应一张数据库表
生成迁移文件
执行迁移生成数据表
使用模型类进行增删改查
ORM
概述：对象-关系-映射
任务：
根据对象的类型生成表结构
将对象，列表的操作转换成SQL语句
将SQL语句查询到的结果转换为对象，列表
优点：
极大的减轻了开发人员的工作量，不需要面对因数据库的变更而修改代码的问题

定义模型

模型，属性，表，字段之间的关系

一个模型类在数据库中对应一张表，在模型类中定义的属性，对应该模型对照表中的一个字段

定义属性：见下文

创建模型类

元选项

在模型类中定义Meta类，用于设置元信息

示例：

```
class Meta:
    db_table = "students"
    ordering = ['id']
```

db_table

定义数据表名，推荐用小写字母，数据表名默认为项目名小写_类名小写

ordering

对象的默认排序字段，获取对象的列表时使用

示例：

```
ordering['id'] id按升序排列
ordering['-id'] id按降序排列
```

注意：排序会增加数据库开销

模型成员

类属性

隐藏类属性objects：

是Manager类型的一个对象，作用是与数据库进行交互

当定义模型类时没有指定管理器，则Django为模型创建一个名为objects的管理器

自定义管理器示例：

定义stuObj管理器：

```
stuObj = models.Manager()
```

当为模型指定模型管理器，Django就不再为模型类生成objects模型管理器了。

自定义管理器Manager类

模型管理器是Django的模型进行与数据库交互的窗口，一个模型可以有多个模型管理器

作用：

向管理器类中添加额外的方法

修改管理器返回的原始查询集

通常会重写get_queryset()方法

代码示例：

```
class StudentsManager(models.Manager):
    def get_queryset(self):
        return super(StudentsManger,
self).get_queryset().filter(isDelete=False)

class Students(model.Moder):
    # 自定义模型管理器
    # 当自定义模型管理器，objects就不存在了
    stuObj = models.Manger()
    stuObj2 = StudentsManager()
```

创建对象

目的：向数据库中添加数据

当创建对象时，django不会对数据库进行读写操作，当调用save()方法时才与数据库交互，将对象保存在数据库表中。

注意：

__init__方法已经在父类models.Model中使用，在自定义的模型中无法使用。

方法：

在模型类中增加一个类方法，示例如下：

```
class Students(model.Moder):
    ...snip...
     @classmethod
    def createStudent(cls, name, age, gender, contend,
                      grade,lastT, createT, isD=False):
        stu = cls(sname=name, sage=age, sgender=gender,
                  scontend=contend, sgrade=grade, lastTime=lastT,
createTime=createT,
```

```
                                        isDelete=isD)
                        return stu
```

在自定义管理器中添加一个方法,示例如下:

```
            class StudentsManager(models.Manager):
                def get_queryset(self):
                    return super(StudentsManager,
self).get_queryset().filter(isDelete=False)
                def createStudent(self, name, age, gender, contend, grade, lastT,
createT, isD=False):
                    stu = self.model()
                    # print(type(grade))
                    stu.sname = name
                    stu.sage = age
                    stu.sgender = gender
                    stu.scontend = contend
                    stu.sgrade = grade
                    stu.lastTime = lastT
                    stu.createTime = createT
                    return stu
```

模型查询

概述

查询集表示从数据库获取的对象的集合

查询集可以有多个过滤器

过滤器就是一个函数，基于所给的参数限制查询集结果

从SQL角度来说，查询集和select语句等价，过滤器就像where条件

查询集

在管理器上调用过滤器方法返回查询集

查询集经过过滤器筛选后返回新的查询集，所以可以写成链式调用

惰性执行

创建查询集不会带来任何数据库的访问，直到调用数据库时，才会访问数据

直接访问数据的情况:

迭代

序列化

与if合用

返回查询集的方法称为过滤器

all()：返回查询集中的所有数据

filter()：保留符合条件的数据

filter(键=值)

filter(键=值，键=值)

filter(键=值).filter(键=值)　且的关系

exclude()：过滤掉符合条件的

order_by()：排序

values():一条数据就是一个字典，返回一个列表

get()

返回一个满足条件的对象

注意：

如果没有找到符合条件的对象，会引发模型类.DoesNotExist异常

如果找到多个对象，会引发模型类MultipleObjectsReturned异常

count()：返回查询集中对象的个数

first()：返回查询集中第一个对象

last():返回查询集中最后一个对象

exits()：判断查询集中是否有数据，如果有数据返回 True,否则返回 False.

限制查询集

查询集返回列表，可以使用下标的方法进行限制，等同于sql中的limit语句

注意：下标不能是负数

示例：studentsList = Students.stuObj2.all()[0:5]

查询集的缓存

概述：

每个查询集都包含一个缓存，来最小化对数据库的访问

在新建的查询集中，缓存首次为空，第一次对查询集求值，会发生数据缓存，Django会将查询出来的数据做一个缓存，并返回查询结果。

以后的查询直接使用查询集的缓存

字段查询

概述

实现了sql中的where语句，作为方法filter(),exclude(),get()的参数

语法：属性名称__比较运算符=值

外键：属性名称_id

转义：类似sql中的like语句

like有关情况看我哥他%是为了匹配点位，匹配数据中的%使用（where like "\%"）

filter(sname__contains="%")

比较运算符

exact:判断，大小写敏感

filter(isDelete=False)

contains：是否包含，大小写敏感

```
studentsList = Students.stuObj2.filter(sname__contains="孙")
```

startswith,endswith:以value开头或结尾，大小写敏感

以上四个在前面加上i，就表示不区分大小写

iexact,icontains,istartswith,iendswith

isnull,isnotnull

是否为空

filter(sname__isnull=False)

in:是否包含在范围内

gt大于,gte大于等于,lt小于,lte小于等于

year,month,day,week_day,hour,minute,second

```
studentsList = Students.stuObj2.filter(lastTime__year=2017)
```

跨关联查询

处理join查询

语法：

模型类名__属性名__比较运算符

```
# 描述中带有‘薛延美’这三个字的数据是属于哪个班级的
grade = Grades.objects.filter(students__scontend__contains='薛延美')
```

```
                print(grade)
```

查询快捷pk代表的主键

聚合函数

使用aggregate函数返回聚合函数的值

Avg

Count

Max

```
maxAge = Student.stuObj2.aggregate(Max('sage'))
```

maxAge为最大的sage。

Min

Sum

F对象

可以使用模型的A属性与B属性进行比较

```
from django.db.models import F,Q
def grades1(request):
    g = Grades.objects.filter(ggirlnum__gt=F('gboynum'))
    print(g)
    # [<Grades: python02>,<Grades: python03>]
    return HttpResponse("OOOOOOOo")
```

支持F对象的算术运算

```
    g = Grades.objects.filter(ggirlnum__gt=F('gboynum')+20)
```

Q对象

概述：过滤器的方法的关键字参数，条件为And模式

需求：进行or查询

解决：使用Q对象

```
    def students4(request):
        studentsList = Students.stuObj2.filter(Q(pk__lte=3) |
Q(sage__gt=50))
        return render(request, 'myApp/students.html',
{"students": studentsList})
```

只有一个Q对象的时候，就是用于正常匹配条件

```
studentsList = Students.stuObj2.filter(~Q(pk__lte=3))
```

~Q是取反

定义属性

概述：

django根据属性的类型确定以下信息

当前选择的数据库支持字段的类型

渲染管理表单时使用的默认html控件

在管理站点最低限度的验证

django会为表增加自动增长的主键列，每个模型只能有一个主键列，如果使用选项设置某属性为主键列后，则django不会再生成默认的主键列

属性命名限制

遵循标识符规则，且变量不能与Python保留字相同

由于django的查询方式，不允许使用连续的下划线

库

定义属性时，需要字段类型，字段类型被定义在django.db.models.fields目录下，为了方便使用，被导入到django.db.models中

使用方式

导入：`from django.db import models`

通过 models.Field创建字段类型的对象，赋值给属性

逻辑删除

对于重要类型都做逻辑删除，不做物理删除，实现方法是定义idDelete属性，类型为BooleanField,默认值为False

字段类型

字段类型

autoField

一个根据实际ID自动增长的IntegerField,通常不指定,

如果不指定，一个主键字段将自动添加到模型中

CharField(max_length=字符长度)

字符串，默认的表彰样式是TextInput

TextField

大文本字段，一般超过4000时使用，默认的表单控件是Textarea

IntegerField

整数

DecimalField(max_digits=None, decimal_places=None)

使用Python的Decimal实例表示的十进制浮点数

参数说明

DecimalField.max_digits

位数总数

DecimalField.decimal_places

小数点后的数字位置

FloatField

使用Python的float实例来表示的浮点数

BooleanField

True/False 字段，此字段的默认表彰控制是CheckboxInput

NullBooleanField

支持 Null, True, False 三种值

DateField([auto_now=False, auto_now_add=False])

使用Python的datetime.date实例表示的日期

参数说明：

DateField.auto_now

每次保存对象时，自动设置该字段为当前时间，用于"最后一次修改"的时间戳，它总是使用当前日期，默认为 False

DateField.auto_now_add

当前对象第一次被创建时自动设置当前时间，用于创建的时间戳，它总是使用当前日期，默认为 False

说明

该字段默认对应的表单控件是一个TextInput.在管理员站点添加了一个JavaScript写的日历控件，和一个"Today"的快捷按钮，包含了一个额外的invalid_date错误消息键

注意

auto_now_add, auto_now, and default 这些设置是相互排斥的，他们之间的任何组合将会发生错误的结果

TimeField

使用Python的datetime.time实例表示的时间，参数同DateField

DateTimeField

使用Python的datetime

datetime实例表示的日期和时间，参数同DateField

FileField

一个上传文件的字段

ImageField

继承了FileField的所有属性和方法，但对上传的对象进行校验，确保它是一个有效的image

字段选项

概述

通过字段选项，可以实现对字段的约束

在字段对象中通过关键字参数指定

null

如果为True,Django将空值以NULL存储在数据库中，默认值为 False

blanke

如果为True,则该字段允许为空白，默认值为 False

注意

null是数据库范畴的概念，blank是表彰验证范畴的概念

db_column

字段的名称，如果未指定，则使用属性的名称

db_index

若值为 True,则在表中会为此字段创建索引

default

默认值

primary_key

若为 True,则该字段会成为模型的主键字段

unique

如果为 True,这个字段在表中必须有唯一值

关系

分类

ForeignKey:一对多，将字段定义在多的端中

ManyToManyField:多对多，将字段定义在两端中

OneToOneField：一对一，将字段定义在任意一端中

用一访问多

格式

对象.模型类小写_set

示例

grade.students_set

用一访问一

格式

对象.模型类小写

示例

grade.studnets

访问id

格式

对象.属性_id

示例

student.sgrade_id

视图

视图

概述：

作用：视图接收web请求，并响应web请求

本质：视图就是python中的一个函数

响应：

响应过程：

用户在浏览器中输入网址www.sunck.wang/sunck/index.html

---网址--->

django获取网址信息，去掉IP与端口号，网址变成：sunck/index.html

---虚拟路径与文件名--->

url管理器逐个匹配urlconf,记录视图函数

---视图函数名--->

视图管理，找到对应的视图去执行，返回结果给浏览器

---响应的数据--->

返回第一步：用户在浏览器中输入网址

网页

重定向

错误视图

404视图：找不到网页（url匹配不成功时返回）时返回

在templates目录下定义404.html

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>页面丢失</h1>
    <h2>{{request_path}}</h2>
</body>
</html>
```

request_path:导致错误的网址

配置settings.py

DEBUG

如果为 True,永远不会调用404页面，需要调整为 False 才会显示

ALLOWED_HOSTS = ['*']

- 500视图：在视图代码中出现错误（服务器代码错误）
- 400视图：错误出现在客户的操作

JSON数据

url配置

配置流程：

制定根级url配置文件

settings.py文件中的ROOT_URLCONF

```
ROOT_URLCONF = 'project.urls'
```

默认实现了

urlpatterns

一个url实例的列表

url对象

正则表达式

视图名称

名称

url匹配正则的注意事项

如果想要从url中获取一个值，需要对正则加小括号

匹配正则前方不需要加'/'

正则前需要加'r'表示字符串不转义

引入其他url配置

在应用中创建urls.py文件，定义本应用的url配置，在工程urls.py中使用include方法

project\urls.py

```
from django.contrib import admin
from django.conf.urls import url,include
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^', include('myApp.urls', namespace="myAPP")),
]
```

myApp\urls.py

```
from django.urls import path, include
from django.conf.urls import url
```

HttpRequest

```
from . import views
urlpatterns = [
    url(r'^$', views.index, name="index"),
]
```

- url的反向解析
  - 概述：如果在视图，模板中使用了硬编码链接，在url配置发生改变时，动态生成链接的地址
  - 解决：在使用链接时，通过url配置的名称，动态生成url地址
  - 作用：使用url模板

- 视图函数
  - 定义视图：
    - 本质：一个函数
    - 视图参数：
      - 一个HttpRequest的实例
      - 通过正则表达式获取的参数
    - 位置：一般在views.py文件下定义

- HttpRequest对象
  - 概述：
    - 服务器接收http请求后，会根据报文创建HttpRequest对象
    - 视图的第一个参数就是HttpRequest对象
    - django创建的，之后调用视图时传递给视图
  - 属性
    - path:请求的完整路径（不包括域名和端口）
    - method:表示请求的方式，常用的有GET,POST
    - encoding:表示浏览器提交的数据的编码方式，一般为utf-8
    - GET：类似于字典的对象，包含了get请求的所有参数
    - POST:类似于字典的对象，包含了post请求的所有参数
    - FILES:类似字典的对象，包含了所有上传的文件
    - COOKIES:字典，包含所有的cookie
    - session:类似字典的对象，表示当前会话
  - 方法
    - is_ajax():如果是通过XMLHttpRequest发起的，返回 True

QueryDict对象

request对象中的GET,POST都属于QueryDict对象

方法:

get():

根据键获取值，只能获取一个值

www.sunck.wang/abc?a=1&b=2&c=3

getlist()

将键的值以列表的形式返回

可以获取多个值

www.sunck.wang/abc?a=1&b=2&c=3

GET属性

获取浏览器传递过来数据

www.sunck.wang/abc?a=1&b=2&c=3

urls.py

```
url(r'^get1', views.get1),    #结尾不能加$,否则无法匹配
```

views.py

```
def get1(request):
    a = request.GET.get('a')
    b = request.GET.get('b')
    c = request.GET.get('c')
    return HttpResponse(a + " " + b + " " + c)
```

www.sunck.wang/abc?a=1&a=2&c=3

urls.py

```
url(r'^get2', views.get2),
```

views.py

```
def get2(request):
    a = request.GET.getlist('a')
    a1 = a[0]
    a2 = a[1]
    c = request.GET.get('c')
```

```
        return HttpResponse(a1 + " " + a2 + " " + c)
```

POST属性

使用表单模拟POST请求

关闭CSRF:project\project\settings.py

```
    MIDDLEWARE = [
        'django.middleware.security.SecurityMiddleware',
        'django.contrib.sessions.middleware.SessionMiddleware',
        'django.middleware.common.CommonMiddleware',
        # 'django.middleware.csrf.CsrfViewMiddleware',
        'django.contrib.auth.middleware.AuthenticationMiddleware',
        'django.contrib.messages.middleware.MessageMiddleware',
        'django.middleware.clickjacking.XFrameOptionsMiddleware',
    ]
```

示例:

```
    def showregist(request):
        return render(request, 'myApp/regist.html',)


    def regist(request):
        name = request.POST.get("name")
        gender = request.POST.get("gender")
        age = request.POST.get("age")
        hobby = request.POST.getlist("hobby")
        print(name)
        print(gender)
        print(age)
        print(hobby)
        return HttpResponse("regist")
```

路径:

```
    url(r'^showregist/$', views.showregist),
        url(r'^showregist/regist/$', views.regist),
```

页面:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>注册</title>
</head>
<body>
    <form action="regist/" method="post">
        姓名：<input type="text" name="name" value=""/>
        <hr>
        性别：<input type="radio" name="gender" value="1">男<input type="radio" name="gender" value="0">女
        <hr>
        爱好：<input type="checkbox" name="hobby" value="power"/>权利 <input type="checkbox" name="hobby" value="money">金钱<input type="checkbox" name="hobby" value="beauty">美女<input type="checkbox" name="hobby" value="Tesla">Tesla
        <hr>
        <input type="submit" value="注册">
    </form>
</body>
</html>
```

HttpResponse

HttpResponse对象

概述：

作用：给浏览器返回数据

HttpRequest对象是由Django创建的，HttpResponse对象是由程序员创建

用法：

不用模板，直接返回数据

语句示例：`return HttpResponse("Sunck is a good man")`

调用模板

使用render方法

原型：`render(request, templateName[, context])`

作用:结合数据和模板，返回一个完整的HTML页面

参数:

`request`：请求体对象

`templateName`：模板路径

`context`：传递给需要渲染在模板上的数据

属性

`content`:表示返回内容

`charset`:编码格式

`status_code`:响应状态码

200

304

404

400

`content-type`:指定输出的MIME类型

方法

`init`:使用页面内容实例化HttpResponse对象

`write(content)`:以文件的形式写入

`flush()`:以文件的形式输出缓冲区

`set_cookie(key, value, maxAge=None, exprise=None)`

`delete_cookie(key)`:

删除cookie

如果删除一个不存在的cookie，就当什么都没发生

子类HttpResponseRedirect

功能：重定向，服务器端的跳转

简写

`redirect(to)`

to推荐使用反向解析

示例：

```
from django.http import HttpResponseRedirect
from django.shortcuts import redirect
def redirect1(request):
    # return HttpResponseRedirect('/redirect2')
    return redirect('/redirect2')
def redirect2(request):
    return HttpResponse("我是重定向后的视图")
```

子类JsonResponse

返回Json数据，一般用于异步请求

__init__(self.data)

data 字典

注意：Content-type类型为application/json

状态保持

http协议是无状态的，每次请求都是一次新的请求，它不记得之前的请求。

客户端与服务器的一次通信就是一次会话

实现状态的保持，在客户端或服务端存储有关会话的数据

存储的方式

cookie:所有数据存储在客户端，不要存储敏感的数据

session:所有的数据存储在服务端，在客户端用cookie存储session_id

状态保持的目的：

在一段时间内跟踪请求者的状态，可以实现跨页面访问当前的请求者的数据

注意：不同的请求者之间不会共享这个数据，与请求者一一对应

启用session:project\project\settings.py

```
INSTALLED_APPS    'django.contrib.sessions',
MIDDLEWARE    'django.contrib.sessions.middleware.SessionMiddleware',
```

使用session

启用session后，每个httpRequest对象都有一个session属性

get[key, default=None]   根据键获取session值

clear()   清空所有会话

flush()   删除当前会话并删除会话的cookie

示例：

```
# session
def main(request):
    # 取session
    username = request.session.get('name', '游客')
    print(username)
    return render(request, 'myApp/main.html', {'username': username})

def login(request):
    return render(request, 'myApp/login.html')

def showmain(request):
    print("*****************")
    username = request.POST.get('username')
    # 存储session
    request.session['name'] = username
    return redirect('/main/')

from django.contrib.auth import logout
def quit(request):
    # 清除session
    logout(request) # 方法1，推荐
    # request.session.clear() # 方法2
    request.session.flush() # 方法3
    return redirect('/main/')
```

设置session过期时间

```
set_expiry(value)
request.session.set_expiry(10)  设置为10秒后过期
```

如果不设置，2个星期后过期

value设置为0代表关闭浏览器时过期

value设置为None代表设置永不过期，不推荐

模板

Redis使用：略

****
**********第二篇笔记从这里开始
****
****

模板
    定义模板
        变量
            变量传递给模板的数据
            要遵守标识符规则
            语法 {{ var }}
            注意：如果使用的变量不存在，则插入的是空字符串
            在模板中使用点语法
                字典查询
                属性或者方法
                数字索引
            在模板中调用对象的方法
                注意：在模板里定义的函数不能传递self以外的参数
        标签:语法 {% tag %}
            作用
                在输出中创建文本
                控制逻辑和循环
        标签示例:
            if 格式

```
        {% if 表达式 %}
            语句
        {% endif %}
if-else 格式
            {% if 表达式 %}
                语句1
            {% else %}
                语句else
            {% endif %}
if-elif-else 格式
            {% if 表达式 %}
                语句1
            {% elif 表达式 %}
                语句2
            ...

            {% else %}
                语句else
            {% endif %}
for 格式
        {% for 变量 in 列表 %}
            语句
        {% endfor %}
    格式2
        {% for 变量 in 列表 %}
            语句
        {% empty %}  # 注意：列表为空或者列表不存在时执行语句2
            语句2
        {% endfor %}
    格式3
        {{ forloop.counter }}
```

示例：

```
<ul>
    {% for stu in students %}
        <li>
            {{forloop.counter}}--{{stu.sname}}--{{stu.sgrade}}
        </li>
    {% empty %}
        <li>目前没有学生</li>
    {% endfor %}
```

commnet 格式

```
{% commnet %}
    被注释的内容
{% endcomment %}
```

作用：相当于多行注释，被注释的内容不再执行

ifequal/ifnotequal 作用 判断是否相等或者不相等

格式

```
{% ifequal 值1 值2 %}
    语句1
{% endifequal %}  # 如果值1等于值2，执行语句1，否则不执行语句1
```

include

作用：加载模板并以标签内的参数渲染

格式：`{% include '模板目录' 参数1 参数2 %}`

url

作用：反射解析

格式：`{% url 'namespace: name' p1 p2 %}`

csrf_token

作用：用于跨站请求伪造保护

格式：`{% csrf_token %}`

block, extends

作用：用于模板的继承

autoescape

作用：用于HTML转义

过滤器

语法 {{ var|过滤器 }}

作用：在变量被显示前修改它，只是加一个效果，对变量不会造成影响

示例：

```
lower
upper
```

过滤器可以传递参数，参数用引号引起来

```
join 格式 列表|join:"#"
```

示例：{{list1|join:"#"}}

如果一个变量没有被提供，或者值为false,空，我们可以通过 default 语法使用默认值

格式： {{str1|default:"没有"}}

根据给定格式转换日期为字符串：date

格式： {{dateVal|date:'y-m-d'}}

HTML转义：escape

加减乘除示例：

```
<h1>num = {{num|add:10}}</h1>
<h1>num = {{num|add:-10}}</h1>
<h1>num = {% num widthratio num 1 5%}</h1>
<h1>num = {% num widthratio num 5 1%}</h1>
```

注释

单行注释：语法： {# 被注释的内容 #}

多行注释

```
{% commnet %}
    被注释的内容
{% endcomment %}
```

反射解析

示例：

```
project/project/urls.py
url(r'^', include('myApp.urls', namespace='app')),
```

project/myApp/urls.py

```
url(r'^good/(\d+)$', views.good, name="good")
```

templates/good.html

```
<a href={% url 'app:good' 1 %}>链接</a>
```

模板继承

作用：模板继承可以减少页面的重复定义，实现页面的重用

block标签：在父模板中预留区域 ，子模板去填充

语法 ：

```
{% block 标签名 %}

{% endblock 标签名 %}
```

extends标签：继承模板，需要写在模板文件的第一行

语法 ：

```
{% extends 'myApp/base.html' %}
{% block main %}
    内容
{% endblock 标签名 %}
```

示例：

定义父模板

body标签中

```
{% block main %}

{% endblock main %}

{% block main %}

{% endblock main2 %}
```

定义子模板

```
{% extends 'myApp/base.html' %}
{% block main %}
    <h1>sunck is a good man</h1>
{% endblock main %}
```

```
{% block main2 %}
    <h1>kaige is a good man</h1>
{% endblock main2 %}
```

HTML转义

问题：`return render(request, 'myApp/index.html', {"code": "<h1>sunck is a very good man</h1>"})`中的`{{code}}`

`{{code}}`里的code被当作`<h1>sunck is a very good man</h1>`显示，未经过渲染

解决方法：

```
{{code|safe}}
```

或

```
{% autoescape off %}
    {{code}}
{% endautoescape %}  # 这个可以一口气解决一堆
```

CSRF：

跨站请求伪造

某些恶意网站包含链接，表单，按钮，js，利用登录用户在浏览器中认证，从而攻击服务

防止CSRF

在settings.py文件的MIDDLEWARE增加'django.middleware.csrf.CsrfViewMiddleware'

```
{% csrf_token %}
```

验证码

作用

在用户注册，登录页面的时候使用，为了防止暴力请求，减轻服务器的压力

是防止CSRF的一种方式

验证码代码示例：

写在views.py里面

==========

验证码源码

```
def verifycode(request):
    # 引入绘图模块
    from PIL import Image, ImageDraw, ImageFont
    # 引入随机函数模块
    import random
    # 定义变量，用于画面的背景色，宽，高
    bgcolor = (random.randrange(20, 100), random.randrange(20, 100), random.randrange(20,
100))
    width = 100
    height = 50
    # 创建画面对象
    im = Image.new('RGB', (width, height), bgcolor)
    # 创建画面对象
    draw = ImageDraw.Draw(im)
    # 调用画笔的point()函数绘制噪点
    for i in range(0, 100):
        xy = (random.randrange(0, width), random.randrange(0, height))
        fill = (random.randrange(0, 255), 255, random.randrange(0, 255))
        draw.point(xy, fill=fill)
    # 定义验证码的备选值
    str = '1234567890QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm'
    # 随机选取4个值作为验证码
    rand_str = ''
    for i in range(0, 4):
        rand_str += str[random.randrange(0, len(str))]
    # 构造字体对象
    font = ImageFont.truetype(r'C:\Windows\Fonts\AdobeArabic-Bold.otf', 40)
    # 构造字体颜色
    fontcolor1 = (255, random.randrange(0, 255), random.randrange(0, 255))
    fontcolor2 = (255, random.randrange(0, 255), random.randrange(0, 255))
    fontcolor3 = (255, random.randrange(0, 255), random.randrange(0, 255))
```

```
        fontcolor4 = (255, random.randrange(0, 255), random.randrange(0, 255))
        # 绘制4个字
        draw.text((5, 2), rand_str[0], font=font, fill=fontcolor1)
        draw.text((25, 2), rand_str[1], font=font, fill=fontcolor2)
        draw.text((50, 2), rand_str[2], font=font, fill=fontcolor3)
        draw.text((75, 2), rand_str[3], font=font, fill=fontcolor4)
        # 释放画笔
        del draw
        # 存入session,用于做进一步的验证
        request.session['verifycode'] = rand_str
        # 内存文件操作
        import io
        buf = io.BytesIO()
        # 将图片保存在内存中，文件类型为png
        im.save(buf, 'png')
        # 将内存中的图片数据返回给客户端，MIME类型为图片png
        return HttpResponse(buf.getvalue(), 'image/png')


    from django.shortcuts import render, redirect
    def verifycodefile(request):
        f = request.session["falg"]
        str = ""
        if f == False:
            str = "请重新输入！"
        request.session.clear()
        return render(request, 'myApp/verifycodefile.html', {"flag":str})

    def verifycodecheck(request):
        code1 = request.POST.get("verifycode").upper()
        code2 = request.session["verify"].upper()
```

高级拓展

```
    if code1 == code2:
        return render(request, 'myApp/success.html')
    else:
        request.session["flag"] = False
        return redirect('/verifycodefile')
```

==========

写在verifycodefile.html的<body>标签中

```
<body>
    <form method="post" action="/verifycodecheck/">
        {%csrf_token%}
        <input type="text" name="verifycode"/>
        <img src="/verifycode">
        <input type="submit" name="登录">
        <span>{{flag}}</span>
    </form>
</body>
```

==========

Django高级扩展

静态文件

css,js,图片,json文件,字体文件等

配置settings.py

```
STATIC_URL = '/static'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]
```

中间件

概述：一个轻量级，底层的插件，可以介入Django的请求和响应。

本质：一个Python类

方法：

__init__

不需要传参数，服务器响应第一个请求的时候自动调用，用于确定是否启用该中间件

process_request(self, request)

在执行视图之前被调用(分配url匹配视图之前)，每个请求都会调用，返回None或者HttpResponse对象

process_view(self, request, view_func. view_args, view_kwargs)

调用视图之前执行，每个请求都会调用，返回None或者HttpResponse对象

process_template_response(self, request, response)

在视图刚好执行完后调用，每个请求都会调用，返回None或者HttpResponse对象

使用render

process_response(self, request, response)

所有响应返回浏览器之前调用，每个请求都会调用，返回HttpResponse对象

process_exception(self, request, exception)

当视图抛出异常时调用，返回HttpResponse对象

执行过程：

__init__ --> process_request--> url --> process_view --> view --> process_template_response --> template --> process_response -->返回开头部分

执行位置：

自定义中间件

在工程目录下的middleware目录下创建myApp

创建一个python文件

```
from django.utils.deprecation import middlewareMixin
class MyMiddle(middlewareMixin):
    def process_request(self, request):
        print("get参数为: ", request.GET.get("a"))
```

使用自定义中间件

配置settings.py文件 在MIDDLEWARE中添加 'middleware.myApp.MyMiddle.MyMiddle',

上传图片

概述：

文件上传时，文件数据request.FILES属性中.

注意：form表单要上传文件需要加enctype="multipart/form-data"

注意：上传文件必须用post请求

存储路径：

在static目录下创建upfile目录用于存储上传的文件

配置settings.py文件 MDEIA_ROOT = os.path.join(BASE_DIR, r'static\upfile')

views.py内容

```
================================================================================
def upfile(request):
    return render(request, 'myApp/upfile.html')

import os
from django.conf import settings
def savefile(request):
    if request.method == "POST":
        f = request.FILES["file"]
        # 文件在服务器端的路径
        filePath = os.pasth.join(settings.MDEIA_ROOT, f.name)
        with open(filePath, 'wb') as fp:
            for info in f.chunks():
                fp.write(info)
        return HttpResponse("上传成功。")
    else:
        return HttpResponse("上传失败。")
================================================================================
```

upfile.html中<body>里的内容

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

分页

```
        {%csrf_token%}
        <input type="file" name="file"/>
        <input type="submit" value="上传"/>
    </form>
</body>
```

==========

- 分页
  - Paginator对象
    - 创建对象
      - 格式 Paginator(列表,整数)
      - 返回值 返回的分页对象
    - 属性
      - count 对象总数
      - num_pages 页面总数
      - page_range
        - 页码列表
        - [1,2,3,4,5]
        - 页码从1开始
    - 方法
      - page(num) 获得一个Page对象，如果提供的页码不存在会抛出"InvalidPage"异常
    - 异常
      - InvalidPage:当向 page()传递的是一个无效的页码时抛出
      - PageNotAnInteger:当向 page()传递的不是一个整数时抛出
      - EmptyPage:当向 page()传递一个有效值，但是该页面里没有数据时抛出
  - Page对象
    - 创建对象
      - Paginator对象的 page()方法返回得到Page对象
      - 不需要手动创建
    - 属性
      - object_list:当前页上所有数据(对象)列表

number:当前页面的页码值

paginator:当前page对象关联的paginator对象

方法

has_next() 判断是否有下一页，如果有返回 True

has_previous() 判断是否有上一页，如果有返回 True

has_other_pages() 判断是否有上一页或者下一页，如果有返回 True

next_page_number() 返回下一页的页码，如果下一页不存在抛出InvalidPage异常

previous_page_number() 返回上一页的页码，如果上一页不存在，抛出InvalidPage异常

len() 返回当前页的数据（对象）个数

Paginator与Page对象关系(略)

代码示例:

配置路由: url(r'^studentpage/(\d+)/$', views.studentpage),

配置视图:

```
from .models import Students
from django.core.paginator import Paginator
def studentpage(request, pageid):
    # 所有学生列表
    allList = Students.objects.all()
    paginator = Paginator(allList, 6)
    page = paginator.page(pageid)
    return render(request, 'myApp/studentpage.html', {"students": page})
```

配置html

studentpage.html中body标签中的内容:

```
<body>
    <ul>
        {% for stu in students %}
        <li>
            {{stu.sname}}--{{stu.sgrade}}
        </li>
        {% endfor %}
```

Ajax

```
                </ul>
                <ul>
                    {% for index in students.paginator.page_range %}
                        {% if index == students.number %}
                            <li>
                                {{index}}
                            </li>
                        {% else %}
                            <li>
                                <a href="/sunck/studentpage/{{index}}">{{index}}</a>
                            </li>
                        {% endif %}
                    {% endif%}
                </ul>
            </body>
```

==========

ajax

需要动态生成，请求JSON数据

代码示例

ajaxstudents.html页面示例

```
        <script type="text/javascript" src="/static/myApp/js/jquery-3.1.1.min.js">
</script>
        <body>
            <h1>学生信息列表</h1>
            <button id="btn">显示学生信息</button>
            <script type="text/javascript" src="/static/myApp/js/sunck.js"></script>
        </body>
```

sunck.js代码示例

```
        $(document).ready(function (){
```

```
            document.getElementById("btn").onclick = function (){
                $.ajax({
                    type:"get",
                    url:"/studentsinfo/",
                    dataType:"json",
                    success:function(data, status){
                        console.log(data)
                        var d = data["data"]
                        for(var i=0; i<d.length; i++){
                            document.write('<p>' + d[i][0] + '</p>')
                        }
                    }
                })
            }
        })
```

views.py代码示例

```
        def ajaxstudents(request):
            return render(request, 'myApp/ajaxstudents.html')

        from django.http import JsonResponse
        def studentsinfo(request):
            stus = Students.objects.all()
            list = []
            for stu in stus:
                list.append([stu.sname, stu.sage])
            return JsonResponse({"data":list})
```

==========

富文本

```
    pip install django-tinymce
```

富文本

在站点中使用

配置settings.py文件

INSTALLED_APPS 列表中添加 'tinymce',

增加

```
# 富文本
TINYMCE_DEFAULT_CONFIG = {
    'theme':'advanced',
    'width':600,
    'height':400,
    }
```

创建一个模型类:

在models.py文件中增加

```
from tinymce.models import HTMLField
class Text(models.Model):
    str = HTMLField()
```

配置站点:

```
from .models import Text
admin.site.register(Text)
```

自定义视图使用

```
<head>
    <meta charset="UTF-8">
    <title>富文本</title>
    <script type="text/javascript" src="/static/tiny_mce/tiny_mce.js">
</script>
    <script type="text/javascript">
        tinyMCE.init({
            'mode':'textareas',
            'theme':'advanced',
            'width':'800',
            'height':'600',
```

```
                })
            </script>
        </head>
        <body>
            <form action="/saveedit/" method="post">
                <textarea name="str">sunck is a good man</textarea>
                <input type="submit" value="提交"/>
            </form>
        </body>
```

Celery

http://docs.jinkan.org/docs/celery/

问题:

用户发起request,并且要等待response返回，但在视图中有一些耗时的操作，
导致用户可能会等待很长时间才能接收response，这样用户体验很差
网站每隔一段时间要同步一次数据，但是http请求是需要触发的

解决:

celery来解决

将耗时的操作放在celery中执行
使用celery定时执行

celery:

任务task

本质是一个Python函数，将耗时的操作封装成一个函数

队列queue

将要执行的任务放在队列里

工人worker

负责执行对列中的任务

代理broker

负责高度，在部署环境中使用redis

安装:

```
pip install celery
```

```
    pip install celery-with-redis
    pip install django-celery
配置settings.py
    在INSTALLED_APPS 列表中添加 'djcelery',
    # Celery
    import djcelery
    djcelery.setup_loader() # 初始化
    BROKER_URL='redis://:sunck@127.0.0.1:6379/0'
    CELERY_IMPORTS=('myApp.task')
在应用目录下创建task.py文件
迁移生成celery需要的数据库表：python manage.py migrate
在工程目录下的project目录下创建celery.py文件
    celery.py文件全部内容
    from __future__ import absolute_import

    import os
    from celery import Celery
    from django.conf import settings

    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'whthas_home.settings')

    app = Celery('portal')

    app.config_from_object('django.conf:settings')
    app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)



    @app.task(bind=True)
    def debug_task(self):
        print('request: {0!r}'.format(self.request))
在工程目录下的 project目录下的 __init__.py文件中添加
```