总结(Django+ 微信小程序)

- API设计--->restful

  一种设计风格,取代SOAP,成为web api 标准,标识资源在网络上的位置

  rest架构原则
  每个资源都有一个唯一的资源标识符同一种资源具有多种表现形式对资源的各种操作不会改变资源标识符
  所有操作都是无状态的符合rest原则的架构方式即可称为restful

  解决问题
  降低开发的复杂性提高系统的可伸缩性

  地址/资源类别/版本/资源的参数

- YAML

  专门用来写配置文件的语言

  基本语法规则:
  大小写敏感
  使用缩进表示层级关系
  缩进的空格数目不重要，只要相同层级的元素左侧对齐即可
  # 表示注释
  YAML 支持的数据结构:
  对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes） / 字典（dictionary）--->对象的一组键值对，使用冒号结构表示
  数组：一组按次序排列的值，又称为序列（sequence） / 列表（list）---->一组连词线开头的行，构成一个数组。
  纯量（scalars）：单个的、不可再分的值
  数值直接以字面量的形式表示
  布尔值用true和false表示
  null用~表示
  字符串默认不使用引号表示
  多行字符串可以使用|保留换行符，也可以使用>折叠换行

  读取YAML文件

```
def apps(request):
  filepath = r'F:\django_project\mysite\mysite\myappconfig.yaml'
  with open(filepath, 'r', encoding='utf8') as f:
    res = yaml.load(f, Loader=yaml.FullLoader)
  return JsonResponse(res, safe=False)
```

- 类视图

  views.py--->

```
from django.views import View  # 类视图
class CookieTest(View):
```

```
    def get(self, request):
        pass
http_method_names = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']
urls--->path('image1', blog.views.CookieTest.as_view()), # 视图类
```

- Mixin

```
# Mixin 给类提供某一种能力--->提供某些功能的类
class UtilMixin():
    @staticmethod
    def savepic(filename, content):
        with open(filename, 'wb') as f:
            f.write(content)
使用--->
from utils.responseutil import UtilMixin
class ImageView(View, UtilMixin):
    UtilMixin.savepic()
```

- 微信小程序

目录结构--->微信小程序项目结构主要有四个文件类型
WXML--->构建页面结构(视图)
WXSS--->用于描述 WXML 的组件样式
js--->逻辑处理
json --->配置文件

主要文件
app.json 微信框架把这个作为配置文件入口，整个小程序的全局配置。包括页面注册，网络设置，以及小程序的 window 背景色，配置导航条样式，配置默认标题
app.js --->监听并处理小程序的生命周期函数、声明全局变量

pages文件夹放置文件

使用第三方组件--->app.wxss--->@import 'thirdparty/weui.wxss';

发起网络请求--->wx.request(){}

使用全局变量实现数据传递
app.js--->globalData
调用 --->const app = getApp()--->app.globalData.name

小程序的生命周期函数
onLoad 页面加载时触发。一个页面只会调用一次，可以在 onLoad 的参数中获取打开当前页面路径中的参数
onShow() 页面显示/切入前台时触发
onReady() 页面初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进

行交互
onHide() 页面隐藏/切入后台时触发。 如 navigateTo 或底部 tab 切换到其他页面，小程序切入后台等
onUnload() 页面卸载时触发。如 redirectTo 或 navigateBack 到其他页面时
底边按钮--->app.json--->tabBar
工具类--->const cookieUtil = require('../../utils/util.js')
开启下拉刷新--->enablePullDownRefresh:true--->小写
事件绑定函数
binds事件类型--->binlongpress='长按'--->bindtap='点击'

小程序的用户体系的构建
小程序先去访问微信服务器
--> code
小程序--> django
---> code + appid + secretkey --> wx服务器
>>openid
--> django后台
给这个用户添加额外的描述信息(添加状态)
--> 有状态的首页的实现

无状态服务--->是指该服务运行的实例不会在本地存储需要持久化的数据，并且多个实例对于同一个请求响应的结果是完全一致的
有状态服务--->是指该服务的实例可以将一部分数据随时进行备份，并且在创建一个新的有状态服务时，可以通过备份恢复这些数据，以达到数据持久化的目的。

- Django模型层使用
  - 跨DB迁移--->sqlite3--->mysql

why--->
-->sqlite3是文件数据库,性能跟不上
--->MySQL是工业界常用(免费开源)

迁移过程--->重要的东西--->数据/表结构

how--->
--->1--->**数据备份**---> **python manage.py dumpdata (appname)>appname.json--->可以不指定应用名** --->2--->**表结构同步**
--->1.pip install mysqlclient
--->2.更改settings.py--->mysql创建好数据库--->blog

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'slave': {
        'ENGINE': 'django.db.backends.mysql',
```

```
        'NAME': 'blog',
        'USER': 'root',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

--->3.**python manage.py migrate --run-syncdb --database slave** --->**将表结构导入mysql**

--->4. 将settings.py---->设置为

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

--->5. **python manage.py loaddata blog.json** --->**将数据加载到MySQL**

- 数据库索引

加快检索数据的速度
降低插入删除更新的速度
排序/比较/过滤--->的字段应该被索引

索引底层结构: B+树

添加索引两种方法

1.db_index=True---->给字段添加索引

2. 添加Meta类

```
cclass User(models.Model):
    openid = models.CharField(max_length=64, unique=True)
    nickname = models.CharField(max_length=64)
    # nickname = models.CharField(max_length=64, db_index=True)
    def __str__(self):
        return self.nickname
    class Meta:
        """
        元: 描绘本身
        """
        # db_table = 'abc'  # 该变表名
        # app_label = 'User'  # 定义模型类属于哪一个应用
        indexes = [
            models.Index(fields=['nickname'], name='nickname'),
        ]
```

3.控制台输入

--->1.python manage.py makemigrations

--->2.python manage.py migrate

查看表的索引-->show index from tablename;

- 数据库操作函数

```
import django
import os
import random
os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'
django.setup()  # 环境配置

def ranstr(length):
    CHS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
    salt = ''
    for i in range(length):
        salt += random.choice(CHS)
    return salt
from blog.models import User
from django.db.models import Q  #  使用 Q 对象构建复杂的查询语句
# 查询
res1 = User.objects.filter(nickname__contains='张')  # 模糊查询
# res = User.objects.get(nickname='1')  # 精确查询
users = User.objects.filter(open_id__contains='test_').order_by('open_id')  # 连锁查询
res2 = User.objects.get(Q(openid='test_openid') | Q(nickname='test_nickname'))
res3 = User.objects.filter(Q(openid='test_openid') & Q(nickname='test_nickname'))
print(res1)
print(users)
# 增加
def add_one():
    user = User(openid='test_openid', nickname='test_nickname')  # 第一种
    # User.objects.create(openid='test_openid2', nickname='test_nickname2')  # 第二种
    user.save()
# add_one()
# 批量添加
def add_batch():
    new_user_list = []
    for i in range(10):
        open_id = ranstr(32)
        nickname = ranstr(10)
        user = User(open_id=open_id, nickname=nickname)
        new_user_list.append(user)
    User.objects.bulk_create(new_user_list)
# add_batch()
# 修改一个
def modify_one():
    user = User.objects.get(open_id='test_open_id')
    user.nickname = 'modify_username'
```

```
    user.save()
# modify_one()
# 批量改
def modify_batch():
    User.objects.filter(open_id__contains='test_').update(nickname='modify_uname')
# 删一个
def delete_one():
    User.objects.get(open_id='test_open_id').delete()
# 批量删除
def delete_batch():
    User.objects.filter(open_id__contains='test_').delete()
# 全部删除
def delete_all():
    User.objects.all().delete()
    # User.objects.delete()
# ---------数据库函数----------
# 数据库函数
# 字符串拼接：Concat
from django.db.models import Value
from django.db.models.functions import Concat
# annotate创建对象的一个属性, Value,如果不是对象中原有属性
def concat_function():
    user = User.objects.filter(open_id='test_open_id').annotate(
        # open_id=(open_id), nickname=(nickname)
        screen_name=Concat(
            Value('open_id='),
            'open_id',
            Value(', '),
            Value('nickname='),
            'nickname')
    )[0]
    print('screen_name = ', user.screen_name)
# 字符串长度： Length
from django.db.models.functions import Length
def length_function():
    user = User.objects.filter(open_id='test_open_id').annotate(
        open_id_length=Length('open_id'))[0]
    print(user.open_id_length)
# 大小写函数
from django.db.models.functions import Upper, Lower
def case_function():
    user = User.objects.filter(open_id='test_open_id').annotate(
        upper_open_id=Upper('open_id'),
        lower_open_id=Lower('open_id')
    )[0]
    print('upper_open_id:', user.upper_open_id, ', lower_open_id:', user.lower_open_id)
    pass
```

```
# 日期处理函数
# Now()
from blog.models import Article
from django.db.models.functions import Now
from datetime import datetime
dt = datetime(day=1,year=2020,month=3)
# print(dt)
def now_function():
    # 当前日期之前发布的所有应用
    apps = Article.objects.filter(publish_date__lte=Now())
    for app in apps:
        print(app)
# 时间截断函数
# Trunc
from django.db.models import Count
from django.db.models.functions import Trunc
def trunc_function():
    # 打印每一天发布的应用数量
    article_per_day = Article.objects.annotate(publish_day=Trunc('publish_date', 'month')) \
        .values('publish_day') \
        .annotate(publish_num=Count('article_id'))
    for article in article_per_day:
        print('date:', article['publish_day'], ', publish num:', article['publish_num'])
```