# 项目总结（11/2）

17:00

（注：文中的代码可以自行copy到pycharm中运行，如果遇到tab消失的情况，可自行根据pdf里的排版tab。此外，建议直接下载github里的项目以供）

## 前言

半个月前，我开始准备有关明年三月份春招的事情。为此搜到了牛客网，及其在知乎上的精彩回答。

以此为基础，我把自己的准备工作分为三个大的部分，项目，算法+知识点，面试经验。为此，我在10.16-11.02这段时间里，边看边练，写出了这个基础的Python网站项目。

期间我重温了Python的基础语法知识，了解了flask的框架结构，还有如何与数据库交互，以及flask-login，图片上传和云存贮，以及单元测试的内容。和git工具的使用，以及如何部署完成的项目到服务器。

此外，项目中对前端知识设计较少，一方面，你可以参照已有的东西来改。另一方面，可以大致学习bootstrap中css的内容，js就用jQuery。

## 1，项目地址：

最终版见github上的P1-address:

https://github.com/jing1900/p1-v3.git

## 2，环境搭建：

### 2.1,数据库环境：

a, 通过pycharm或pip，安装pymysql，然后安装sqlalchemy和libmysqlclient-dev

b,安装mariadb。http://blog.topspeedsnail.com/archives/6323，设置的mariadb的用户名密码为root：12345678

c,操作：http://www.jianshu.com/p/e59afa955a2d

C1, 登陆MariaDB命令行：$ sudo mysql -u root -p

C2,把数据库文件恢复到指定的数据库:先创建一个test数据库 create database test；
use xxx;
source xxx.sql #该sql在本目录下

d,遇到 access denied 的问题时，解决方案：

```
Sudo mysql -u root -p123456
Use mysql;
Update user set plugin='' where User ='root';
Flush privileges;
\q
```

### 2.2,Python 环境：

a，python3.5+anaconda+pycharm

### 2.3,Flask扩展：

a，pip install Python-flask  Flask-Script Flask-SQLAlchemy Flask-Login qiniu

### 2.4,七牛：

a, pip install qiniu

## 3, 技术总结：

http://tool.oschina.net/highlight ：在线代码调亮  Django样式

### 3.1，Python基础语法：（一句话，有需要就去看官方文档）(c1.py)

```
# -*- encoding=UTF-8 -*-
import requests
import random
import re
from bs4 import BeautifulSoup

'''糗事百科爬虫'''
def qiushibaike():
    #获取内容
    content = requests.get('http://www.qiushibaike.com').content
    #解析
    soup = BeautifulSoup(content, 'html.parser')
    #打印
    for div in soup.find_all('div', {'class': 'content'}):
        print(div.text.strip())

'''python string入门'''
def demo_string():
    stra = 'hello world'
    #变成首字母大写的标准形式
    print(stra.capitalize())
    #字符串替换
    print(stra.replace('world', 'nowcoder'))
    strb = '  \n\rhello nowcoder \r\n'
    #去除左边的空格回车等符号
    print(1, strb.lstrip())
    #去除右边的
    print(2, strb.rstrip())
    strc = 'hello w'
    #判断字符串是不是以这个开头结尾
    print(3, strc.startswith('hel'))
    print(4, strc.endswith('x'))
    #字符串连接
    print(5, stra + strb + strc)
    #字符串长度
    print(6, len(strc))
    #字符串连接
    print(7, '-'.join(['a', 'b', 'c']))
    #字符串连接
    print(8, strc.split(' '))
    #返回该字符串的首字母索引
    print(9, strc.find('ello'))

'''python 操作数入门'''
def demo_operation():
    print(1, 1 + 2, 5 / 2, 5 * 2, 5 - 2)
    print(2, True, not True)
    print(3, 1 < 2, 5 > 2)
    #位操作
    print(4, 2 << 3)
    #位操作
    print(5, 5 | 3, 5 & 3, 5 ^ 3)
    x = 2
```

```
    y = 3.3
    print(x, y, type(x), type(y))

'''Python行内函数入门'''
def demo_buildinfunction():
    #最大最小，绝对值，长度
    print(1, max(2, 1), min(5, 3))
    print(2, len('xxx'), len([1, 2, 3]))
    print(3, abs(-2))  # fabs,Math.fabs
    #python3,需要加list()
    print(4, list(range(1, 10, 3)))
    #dir，获取元素属性
    print(5, dir(list))
    x = 2
    #eval方法，将字符串str当做有效的表达式来求值，并返回计算结果
    print(6, eval('x + 3'))
    #ascii编码转换
    print(7, chr(65), ord('a'))
    #得到除数余数
    print(8, divmod(11, 3))

'''python 控制流入门'''
def demo_controlflow():
    score = 65
    #if
    if score > 99:
        print(1, 'A')
    elif score > 60:
        print(2, 'B')
    else:
        print(3, 'C')
    #while
    while score < 100:
        print(score)
        score += 10
    score = 65

    #for
    # for (int i = 0; i < 10; ++i)
    # continue ,break, pass
    for i in range(0, 10, 2):
        if i == 0:
            pass # do_special
            # print(3, i
        if i < 5:
            continue
        print(3, i)
        if i == 6:
            break

'''python list 入门'''
def demo_list():
    #定义
    lista = [1, 2, 3]  # vector<int> Arraylist
    print(1, lista)
    #一个list可包含不同类型
    listb = ['a', 1, 'c', 1.1]
    print(2, listb)
    #拓展
    lista.extend(listb)
    print(3, lista)
    #长度
    print(4, len(lista))
    #判断元素是否在
    print(5, 'a' in listb)
    lista = lista + listb
    print(6, lista)
    #插入元素
    listb.insert(0, 'www')
    print(7, listb)
    #弹出元素
    listb.pop(1)
    print(8, listb)
    #反转列表
    listb.reverse()
    print(9, listb)
    print(10, listb[0], listb[1])
    #排序，一般需要列表中的元素为同一种
```

```
    #listb.sort()
    print(11, listb)
    #listb.sort(reverse=True)
    print(12, listb)
    #元素长度扩大两倍
    print(13, listb * 2)
    print(14, [0] * 14 ) # memset(src, 0, len))
    tuplea = (1, 2, 3)
    listaa = [1, 2, 3]
    listaa.append(4)
    print(15, listaa)


def add(a, b):
    return a + b


def sub(a, b):
    return a - b

'''python 字典入门'''
def demo_dict():
    #定义
    dicta = {4: 16, 1: 1, 2: 4, 3: 9}
    print(1, dicta)
    #得到键，值
    print(2, dicta.keys(), dicta.values())
    #for key in dict:python 3
    #print(3, dicta.has_key(1), dicta.has_key('3'))
    # for map<int,int>::iterator it = x.begin(); it != x.end()
    for key, value in dicta.items():
        print('key-value:', key, value)
    #字典的值还可以是函数
    dictb = {'+': add, '-': sub}
    print(4, dictb['+'](1, 2))
    print(5, dictb.get('-')(15, 3))
    #字典赋值
    dictb['*'] = 'x'
    print(dictb)
    #弹出字典元素
    dicta.pop(4)
    print(6, dicta)
    #删除字典元素
    del dicta[1]
    print(7, dicta)

'''python 集合入门'''
def demo_set():
    lista = [1, 2, 3]
    #set定义
    seta = set(lista)
    setb = set((2, 3, 4))
    print(1, seta)
    #交
    print(3, seta.intersection(setb), seta & setb)
    #并
    print(4, seta | setb, seta.union(setb))
    #补
    print(5, seta - setb)
    #添加
    seta.add('x')
    print(6, seta)
    print(len(seta))
    #
    print(seta.isdisjoint(set((1, 2))))

'''python 封装 继承 多态'''
class User:
    type = 'USER'


    def __init__(self, name, uid):
        self.name = name
        self.uid = uid
```

```
    def __repr__(self):
        return 'im ' + self.name + ' ' + str(self.uid)


class Guest(User):
    def __repr__(self):
        return 'im guest:' + self.name + ' ' + str(self.uid)


class Admin(User):
    type = 'ADMIN'

    def __init__(self, name, uid, group):
        User.__init__(self, name, uid)
        self.group = group

    def __repr__(self):
        return 'im ' + self.name + ' ' + str(self.uid) + ' ' + self.group


def create_user(type):
    if type == 'USER':
        return User('u1', 1)
    elif type == 'ADMIN':
        return Admin('a1', 101, 'g1')
    else:
        return Guest('gu1', 201)
        # raise ValueError('error')

'''python 异常入门'''
def demo_exception():
    try:
        print(2 / 1)
        # print(2 / 0
        # if type == 'c':
        raise Exception('Raise Error', 'NowCoder')
    except Exception as e:
        print('error:', e)
    finally:
        print('clean up')

'''python 随机数入门'''
def demo_random():
    # 1 - 100
    # random.seed(1)
    # x = prex * 100007 % xxxx
    # prex  = x 幂等性
    #1-100随机数
    print(1, int(random.random() * 100))
    #随机整数
    print(2, random.randint(0, 200))
    #选1
    print(3, random.choice(range(0, 100, 10)))
    #选4个
    print(4, random.sample(range(0, 100), 4))
    a = [1, 2, 3, 4, 5]
    #随机打乱
    random.shuffle(a)
    print(5, a)

'''python 正则表达式'''
def demo_re():
    str = 'abc123def12gh15'
    #1个多个数字
    p1 = re.compile('[\d]+')
    #1个数字
    p2 = re.compile('\d')
    print(1, p1.findall(str))
    print(2, p2.findall(str))
    # \t\n
    str = 'a@163.com;b@gmail.com;c@qq.com;e0@163.com;z@qq.com'
```

```
    #163，qq邮箱
    p3 = re.compile('[\w]+@[163|qq]+\.com')
    print(3, p3.findall(str))
    #中间是非<的内容
    str = '<html><h>title</h><body>xxx</body></html>'
    p4 = re.compile('<h>[^<]+</h>')
    print(4, p4.findall(str))
    #只取中间的部分，返回不包含<h></h>
    p4 = re.compile('<h>([^<]+)</h><body>([^<]+)</body>')
    print(5, p4.findall(str))
    #时间格式匹配
    str = 'xx2016-06-11yy'
    p5 = re.compile('\d{4}-\d{2}-\d{2}')
    print(p5.findall(str))


if __name__ == '__main__':
    qiushibaike()
    user1 = User('u1', 1)
    print(user1)
    admin1 = Admin('a1', 101, 'g1')
    print(admin1)

    print(create_user('USERX'))

    demo_string()
    demo_operation()
    demo_buildinfunction()
    demo_controlflow()
    demo_list()
    demo_dict()
    demo_set()
    demo_exception()
    demo_random()
    demo_re()
```

## 3.2,Flask入门（有问题就去看文档：http://dormousehole.readthedocs.io/en/latest/（中文））：

### a，装饰器-decorator (decorator.py)

```
#-*- encoding=UTF-8 -*-
'''装饰器'''
def log(level, *args, **kvargs):
    def inner(func):
        '''
        * 无名字参数，如果没有user = user,则放在这个数组里
        ** 有名字参数，否则，会放在kvargs
        '''

        def wrapper(*args, **kvargs):
            print(level, 'before calling ', func.__name__)
            print(level, 'args', args, 'kvargs', kvargs)
            func(*args, **kvargs)
            print(level, 'end calling ', func.__name__)

        return wrapper
    return inner


@log(level='INFO')
def hello(name, age):
    print('hello', name, age)

if __name__ == '__main__':
    hello(name='nowcoder', age=2) #= log(hello())
```

### b,Routing + HTTP Method + request/response + 重定向/Error + Flash Message+ Log (c2.py)

```
# -*- encoding=UTF-8 -*-

from flask import Flask, render_template, request, make_response, redirect, flash, get_flashed_messages
import logging
from logging.handlers import RotatingFileHandler
```

```
#定义app
app = Flask(__name__)
#加了这一行之后，在templates里面的语法，可以直接#开头
app.jinja_env.line_statement_prefix = '#'
#设置session之间相互通信的身份标识
app.secret_key = 'nowcoder'

#路径与函数的映射，这里路径可以是多个，因此，很适合改版的情况
@app.route('/index/')
@app.route('/')
def index():
    res = ''
    #这里获取通过flash传递的消息，其中category可以作为标识
    for msg, category in get_flashed_messages(with_categories=True):
        res = res + category + msg + '<br>'
    res += 'hello'
    return res

#这里我们用了前后端分离的策略，即前端的大部分代码放在templates里，只有少部分需要修改的参数，在这里传进去：render_template
#<int:uid>这里获取地址栏的参数,同时我们可以设置访问的类型，大小写都行
@app.route('/profile/<int:uid>/', methods=['GET', 'post'])
def profile(uid):
    colors = ('red', 'green')
    infos = {'nowcoder': 'abc', 'google': 'def'}
    return render_template('profile.html', uid=uid, colors=colors, infos=infos)

#request，可以获取参数，如果没有就得到defaultkey
@app.route('/request')
def request_demo():
    key = request.args.get('key', 'defaultkey')
    res = request.args.get('key', 'defaultkey') + '<br>'
    res = res + request.url + '++' + request.path + '<br>'
    #看request方法有什么属性
    for property in dir(request):
        res = res + str(property) + '<br>' + str(eval('request.' + property)) + '<br>'
    #response可以设置一些东西
    response = make_response(res)
    response.set_cookie('nowcoderid', key)
    response.status = '404'
    response.headers['nowcoder'] = 'hello~~'
    return response

#重定向后要跳转的路径
@app.route('/newpath')
def newpath():
    return 'newpath'

#访问它需要重定向的路径
@app.route('/re/<int:code>')
def redirect_demo(code):
    #跳转到新的路径，允许状态码
    return redirect('/newpath', code=code)

#错误页
@app.errorhandler(400)
def exception_page(e):
    response = make_response('出错啦~')
    return response

#404页，这里还可以返回找不到的url值
@app.errorhandler(404)
def page_not_found(error):
    return render_template('not_found.html', url=request.url), 404

#测试
@app.route('/admin')
def admin():
    if request.args['key'] == 'admin':
        return 'hello admin'
    else:
```

```
        raise Exception()
#测试info log和flash传递消息
@app.route('/login')
def login():
    app.logger.info('log success')
    flash('登陆成功', 'info')
    return 'ok'
    # return redirect('/')

#在页面中显示log等级
@app.route('/log/<level>/<msg>/')
def log(level, msg):
    dict = {'warn': logging.WARN, 'error': logging.ERROR, 'info': logging.INFO}
    if dict.has_key(level):
        app.logger.log(dict[level], msg)
    return 'logged:' + msg

#设置log，其中error的会出现在error，info和warn里，warn的会出现在warn和info里，info的只会出现在info里
def set_logger():
    info_file_handler = RotatingFileHandler('info.txt')
    info_file_handler.setLevel(logging.INFO)
    app.logger.addHandler(info_file_handler)

    warn_file_handler = RotatingFileHandler('warn.txt')
    warn_file_handler.setLevel(logging.WARN)
    app.logger.addHandler(warn_file_handler)

    error_file_handler = RotatingFileHandler('error.txt')
    error_file_handler.setLevel(logging.ERROR)
    app.logger.addHandler(error_file_handler)


if __name__ == '__main__':
    set_logger()
    app.run(debug=True)
```

## c,静态和模板文件 +  Jinja2语法 (profile.html)

```
{#
注释部分：
for循环
filter endfilter，过滤器，比如可以将其中的内容都大写显示
macro endmacro 宏，就相当于函数
include|extend：模板继承

#}

<html>
<link rel="stylesheet" type="text/css" href="/static/c2.css"/>
<body>
<h>head</h>
<br>
profile : {{ uid }} <br>

{# 看不到我 ~#}
{% for i in range(0, 5): %}
profile: i <br>
{% endfor %}

# for color in colors:
{{ color }}<br>
# endfor

{% for color in colors: %}
{{ loop.index }} {{ color }} <br>
{% endfor %}

{% filter upper %}
{% for k, v in infos.items(): %}
{{ k }}, {{ v }} <br>
{% endfor %}
{% endfilter %}

{% macro render_color(color) %}
<div>This is color {{ color }}</div><br> {{ caller() }}
```

```
{% endmacro %}

{% for color in colors: %}
    {% call render_color(color) %}
        render_color_demo
    {% endcall%}
{% endfor %}

</body>
</html>
```

### d,Flask-Script (manage.py)

```
# -*- encoding=UTF-8 -*-
from flask_script import Manager
from c2 import app
'''这个文件存在的意义在于，当项目存在很多很多文件时，一一启动太麻烦，你可以在这里自己定义一些启动方式'''
manager = Manager(app)

'''这里可以放入参数，和参数的默认值'''
@manager.option('-n', '--name', dest='name', default='nowcoder')
def hello(name):
    print('hello', name)

@manager.command
def initialize_database():
    'initialize database'
    print('database ...')

if __name__ == '__main__':
    manager.run()

'''使用时，可在终端先python manager.py，然后python manager.py initialize_database

'''
```

## 3.3, 项目框架搭建+Flask-SQLAlchemy 配置与使用+模板继承

### a,项目框架：

```
application
| manage.py <-脚本数据
| runserver.py <-启动服务器，run这个
|
└──application <-web目录
| app.conf <-配置文件
| models.py <-数据模型
| views.py <- 视图
| __init__.py <-初始化
|
├──static <-静态文件
└──templates <-页面模板
base.html
index.html
login.html
```

### b，数据库配置与使用

#### B1:配置：App.conf文件里：

```
#参数依次是:数据库类型：//用户名：密码@地址：端口号/数据库的名字
SQLALCHEMY_DATABASE_URI = 'mysql://root:nowcoder@localhost:3306/test'
#SQLALCHEMY_DATABASE_URI = 'sqlite:///../nowstagram.db'
SQLALCHEMY_TRACK_MODIFICATIONS = True
SQLALCHEMY_ECHO = False
SQLALCHEMY_NATIVE_UNICODE = True
SQLALCHEMY_RECORD_QUERIES = False
```

## 然后__init__.py里：

```
from flask_sqlalchemy import SQLAlchemy
...
db = SQLAlchemy(app)
...
```

## B2:构建数据库表：ORM映射方式。一对多、多对多 ：（model.py)

```
#-*-encoding=UTF-8-*-
'''数据模型，有问题就去查官方文档，上面什么都有
真的没有比官方文档更好的东西了'''
#stagram是web目录
from stagram import db
from datetime import datetime
import random

#评论类
class Comment(db.Model):
    #评论id
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    #内容
    content = db.Column(db.String(1024))
    #评论是属于那张图片的
    image_id = db.Column(db.Integer,db.ForeignKey('image.id'))
    #评论是谁发的
    user_id = db.Column(db.Integer,db.ForeignKey('user.id'))
    #设计一个字段，来表明当前实体属于什么状态
    status = db.Column(db.Integer,default=0)#0,正常，1，被删除
    #将评论和用户关联起来
    user = db.relationship('User')

    def __init__(self, content, image_id, user_id):
        self.content = content
        self.image_id = image_id
        self.user_id = user_id

    def __repr__(self):
        return '<comment %d: %s>'%(self.id,self.content)

class Image(db.Model):
    #图片id
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    #url
    url = db.Column(db.String(512))
    #创建时间
    created_data = db.Column(db.DateTime)
    #图片是那个user id发的,这里的user_id是user的外键
    user_id = db.Column(db.Integer,db.ForeignKey('user.id'))
    #评论关联起来
    comments = db.relationship('Comment')

    def __init__(self,url,user_id):
        self.url = url
        self.created_data = datetime.now()
        self.user_id = user_id

    def __repr__(self):
        return '<Pic %d: %s>'%(self.id,self.url)

class User(db.Model):
    # __tablename__ = 'myuser' 指定表名字,不指定就默认类名小写
    '''这里类里的一个变量，就表示表中的一列，具体怎样跟数据库做交互见manage.py'''
    #user id,指明类型，是否主键和是否自动增长
    id = db.Column(db.Integer,primary_key=True,autoincrement=True)
    #用户名，指明类型，和非重复
    username = db.Column(db.String(80),unique=True)
    #密码
    password = db.Column(db.String(32))
```

```
    #头像
    head_url = db.Column(db.String(256))
    #这里我们怎么将每个人发的图片关联起来呢，1对多
    images = db.relationship('Image',backref='user',lazy='dynamic')

    '''定义构造函数'''
    def __init__(self,username,password):
        self.username = username
        self.password = password
        #这里头像先用牛客网给出的1000张图片之一，中间的变量是0-1000之间随机一个整数
        self.head_url = 'http://images.nowcoder.com/head/' + str(random.randint(0, 1000)) + 't.png'

    def __repr__(self):
        return '<User %d : %s>'%(self.id,self.username)
```

## B3：增删改查(manage.py里的init_database())

```
#-*- encoding=UTF-8 -*-
from stagram import app,db
from stagram.models import User
from stagram.models import Image,Comment
import random
from sqlalchemy import or_,and_
'''脚本'''
#导入manager
from flask_script import Manager


manager = Manager(app)

#从服务器上获取图片
def get_image():
    return 'http://images.nowcoder.com/head/' + str(random.randint(0, 1000)) + 'm.png'

@manager.command
def init_database():
    '''前面这两行是为了方便演示，正常情况下，这两行是要删掉的，只在第一次运行时创建表'''
    #先删除所有表
    db.drop_all()
    #再把所有表创建起来,这里是把所有定义好的数据类，根据类名和变量名创建好.可以在终端里面这样跑python manage.py init_database
    #在终端执行完这个，就可以用deraidb查看，test数据库里面已经被创建了一个名叫table的表。一切都很顺利
    db.create_all()
    '''增--先添加100个用户'''
    for i in range(1,100):
        db.session.add(User('User'+str(i),'pw'+str(i)))
        #为每个人添加三张图片
        for j in range(0, 3):  # 每人发三张图
            # 这里user_id应该为i，因为循环是从1开始的，而自增是从1开始的
            db.session.add(Image(get_image(), i ))
            #每张图片加三条评论
            for k in range(0,3):
                db.session.add(Comment('this is a comment'+str(k),3*(i-1)+j+1,i))

    db.session.commit()#数据库事务就是没提交时，啥都不是

    '''更新,50-100内的偶数用户'''
    for i in range(50,100,2):
        user = User.query.get(i)
        user.username = '[*]'+User.username
    #直接用update方式更新，update的参数是一个字典,更新51-100之间的奇数
    for i in range(0, 100, 10):
        # 通过update函数
        User.query.filter_by(id=i + 1).update({'username': '牛客新' + str(i)})

    db.session.commit()

    '''删'''
    #删除从50-100的奇数评论
    for i in range(50,100,2):
        comment = Comment.query.get(i+1)
        db.session.delete(comment)
    db.session.commit()

    '''查'''
```

```
    #查全部
    #print(User.query.all())
    #查第三个
    #print(User.query.get(3))
    #有条件的查询
    #print(User.query.filter_by(id = 5).first())
    #根据id将序取，然后偏移一下，取两个
    #print(User.query.order_by(User.id.desc()).offset(1).limit(2).all())
    #打印以0结尾的用户名
    #print(User.query.filter(User.username.endswith('0')).limit(3).all())
    #组合查询.这里如果去掉之后的all（），就会打印数据库查询语句
    #print(User.query.filter(or_(User.id == '88',User.id == '99')).all())
    #print(User.query.filter(and_(User.id > '88', User.id < '90')).all())
    #返回第一个或者报404错误
    #print(User.query.filter(and_(User.id > '88', User.id < '90')).first_or_404())
    #分页查询
    #print(User.query.paginate(page=1,per_page=10).items)
    #逆序后分页查询
    #print(User.query.order_by(User.id.desc()).paginate(page=1, per_page=10).items)
    #1对多的查询，这里以第一个用户为例
    #user1 = User.query.get(1)
    #print(user1.images)#这是因为我们已经把user和images表关联起来了，在image表中，可以容易的根据外键来查到对应的表
    #直接这样查询是没有结果的，我们需要在user里制定backref.images = db.relationship('Image',backref='user',lazy='dynamic')
    #image1 = Image.query.get(1)
    #print(image1.user)


if __name__ == '__main__':
    manager.run()
```

## c,模板继承：

### Base.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="description" content="">
    <meta name="Keywords" content="">

    <title>{%block title%}{%endblock%}</title>
    <link rel="stylesheet" href="{%block css%} {%endblock%}">
</head>
<body>
    <div class="container">
        <section>
            <header class="header">
                <div class="header-cont">
                    <a class="logo" href="/">logo</a>
                    <div class="web-menu">
                        <a class="profile-ico">个人主页</a>
                    </div>
                </div>
            </header>

            {%block content%}{%endblock%}

            <footer class="footer">
                <div class="footer-cont" style="max-width:935px;">
                    <nav>
                        <ul class="footer-items">
                            <li>
                                <a href="/about/us/">关于我们</a>
                            </li>
                            <li>
                                <a href="https://help.nowcoder.com/">支持</a>
                            </li>
                            <li>
                                <a href="http://blog.nowcoder.com/">博客</a>
                            </li>
                            <li>
                                <a href="/press/">新闻中心</a>
                            </li>
                            <li>
                                <a href="/developer/">API</a>
                            </li>
                            <li>
                                <a href="/about/jobs/">工作信息</a>
                            </li>
```

```
                        <li>
                            <a href="/legal/privacy/">隐私</a>
                        </li>
                        <li>
                            <a href="/legal/terms/">条款</a>
                        </li>
                        <li>
                            <span class="language-wrapper">
                                <span>语言</span>
                                <select class="ui-select">
                                    <option value="af">南非荷兰语</option>
                                    <option value="cs">捷克语</option>
                                    <option value="da">丹麦语</option>
                                    <option value="de">德语</option>
                                    <option value="el">希腊语</option>
                                    <option value="en">英语</option>
                                    <option value="es">西班牙语</option>
                                    <option value="fi">芬兰语</option>
                                    <option value="fr">法语</option>
                                    <option value="hi">印地语</option>
                                    <option value="id">印度尼西亚语</option>
                                    <option value="it">意大利语</option>
                                    <option value="ja">日语</option>
                                    <option value="ko">韩语</option>
                                    <option value="ms">马来语</option>
                                    <option value="nb">挪威语</option>
                                    <option value="nl">荷兰语</option>
                                    <option value="pl">波兰语</option>
                                    <option value="pt">葡萄牙语（葡萄牙）</option>
                                    <option value="pt-br">葡萄牙语</option>
                                    <option value="ru">俄语</option>
                                    <option value="sv">瑞典语</option>
                                    <option value="th">泰语</option>
                                    <option value="tl">塔加洛语/菲律宾语</option>
                                    <option value="tr">土耳其语</option>
                                    <option selected="" value="zh-cn">中文（简体）</option>
                                    <option value="zh-tw">中文（繁体）</option>
                                </select>
                            </span>
                        </li>
                    </ul>
                </nav>
                <span class="copy-right">© 2016 nowcoder</span>
            </div>
        </footer>
    </section>
</div>
</body>
</html>
```

## Index.html（继承自base.html)

```
{%extends 'base.html'%}
{%block title%}首页{%endblock%}
{%block css%}/static/styles/pages/index.css {%endblock%}

{%block content%}
<div class="page-main clearfix">
    {%for image in images%}
    <article class="mod">
            <header class="mod-hd">
                <time class="time">{{image.created_data}}</time>
                <a href="/profile/{{image.user.id}}" class="avatar">
                    <img src="{{image.user.head_url}}">
                </a>
                <div class="profile-info">
                    <a title="{{image.user.username}}" href="/profile/{{image.user.id}}">{{image.user.username}}</a>
                </div>
            </header>
            <div class="mod-bd">
```

```
                <div class="img-box">
                    <a href="/image/{{image.id}}">
                        <img src="{{image.url}}">
                    </a>
                </div>
            </div>
            <div class="mod-ft">
                <!--<section class="times">
                    <span></span>
                    <span>6.2百万</span>
                    <span> 次播放</span>
                </section>-->
                <ul class="discuss-list">
                    <li class="more-discuss">
                        <a>
                            <span>全部 </span><span class="">{{image.comments|length}}</span>
                            <span> 条评论</span></a>
                    </li>
                    {%for comment in image.comments%}
                    {#这里设置最多显示两条评论,直接运行会报错，因为这里不支持break语言，要支持，需要在init里加上
app.jinja_env.add_extension('jinja2.ext.loopcontrols')#}
                    {%if loop.index >2%}
                        {%break%}
                    {%endif%}
                        <li>
                            <!--<a class=" icon-remove" title="删除评论"></a>-->
                            <a class="_4zhc5 _iqaka" title="{{comment.user.username}}" href="/profile/{{comment.user.id}}" data-
reactid=".0.1.0.0.0.2.1.2:$comment-17856951190001917.1">{{comment.user.username}}</a>
                            <span>
                                <span>{{comment.content}}</span>
                            </span>
                        </li>
                    {%endfor%}


                </ul>
                <section class="discuss-edit">
                    <a class="icon-heart"></a>
                    <form>
                        <input placeholder="添加评论..." type="text">
                    </form>
                    <button class="more-info">更多选项</button>
                </section>
            </div>
        </article>
    {%endfor%}
</div>
{%endblock%}
```

## 3.4,注册登录Flask-login和ajax异步刷新

### a，注册 登录 页面访问

注册时：

用户名：需要有格式限制，过滤敏感词（管理员等），重复，和一些特殊字符（比如一些恶意的html语言）

密码：Md5已经太弱了，很大部分都是可以被破解的。因此我们引入了密码加盐的概念，将盐也作为一个字段存入数据库。

验证码：则是为了避免大量恶意注册，不过现在最好的验证码是基于人工行为的：分发key，页面自动生成图片，后台临时记录比对

登录过程：

首先输入用户名，密码，服务器然后进行登录校验，校验成功，服务器会下发一个值token（数据库也会将token和userid关联起来），浏览器就会将这个下发的token和userid关联起来，表明这个人登录成功。下发的这个token，客户端和浏览器都要存，客户端存本地，每次你请求时，都要带这个token，来告诉服务器你是谁。浏览器存cookie里。服务器和客户端还会设置这个token的有效期，比如记住登录。如果不设置的话，默认浏览器一关闭，有效期就结束了

登出的过程：

就是服务器和客户端都将token删掉

页面访问的流程：

客户端发带token的http请求，服务器来解析这个token，得到用户id，再做一系列的页面渲染，跳转，权限处理等操作。

### A1，注册登录页面：login.html

```
{%extends 'base.html'%}
{%block title%}注册登录页{%endblock%}
{%block css%}/static/styles/pages/login.css {%endblock%}
{%block content%}
<main class="main login-main" role="main">
    <article class="login-cont clearfix">
```

```
            <div class="login-pic-box">
                <img src="/static/images/res/nkFamily.jpg">
            </div>
            <div class="login-wrapper">
                <div class="login-box">
                    <form method ="post",id="reg_login_form">
                        <h2 class="login-hd">
                            {%if (msg|length) >0%}
                                <b>{{msg}}</b>
                            {%else%}
                                注册 牛客
                            {%endif%}</h2>
                        <div class="form-item">
                            <input class="input-txt" aria-label="用户名" aria-
    required="true" autocapitalize="off" autocorrect="off" maxlength="30" name="username" placeholder="用户名" value="" type="text" data-
    reactid=".0.1.0.1.0.1.0.5.0">
                        </div>
                        <div class="form-item">
                            <input class="input-txt" aria-describedby="" aria-label="密码" aria-
    required="true" autocapitalize="off" autocorrect="off" name="password" placeholder="密码" type="password" value="" data-
    reactid=".0.1.0.1.0.1.0.6.0">
                        </div>
                        <input type="hidden" name="neuxt" value="{{next}}"/>
                        <div class="btn-wrapper">
                            <button class="btn-primary" onclick="document.getElementById('reg_login_form',form.action='/reg/')">注册</button>
                            <button class="btn-primary" onclick="document.getElementById('reg_login_form',form.action='/login/')">登录</button>
                        </div>
                        <p class="agreenment">
                            <span>注册即表示你同意我们的 </span>
                            <a class="agreenment-link" href="/legal/terms/" target="_blank">条款</a>
                            <span data-reactid=".0.1.0.1.0.1.0.9.2"> 和 </span>
                            <a class="agreenment-link" href="/legal/privacy/" target="_blank">隐私权政策</a>
                            <span> 。</span>
                        </p>
                    </form>
                </div>
            </div>
        </article>
    </main>
    {%endblock%}
```

## A2，注册登录视图，注册，登入，登出函数（view.py）

```
@app.route('/reg_login_page/')
def reg_login():
    #这里用来获取注册时反馈的flash
    msg =''
    for m in get_flashed_messages(with_categories=False,category_filter=['reg_log']):
        msg = msg+m
    #这里我们增加一个字段next，来记录登录后想跳转到的界面.同时，我们需要在login界面里面加一个隐藏字段，来获取next
    #再同时，我们需要在login函数里面做一个判断，如果有next字段，则跳转至next
    return render_template('login.html',msg=msg,next = request.values.get('next'))


@app.route('/reg/',methods={'post','get'})
def reg():
    #request.args:url里面的参数
    #request.form:body里面的数据，也可以用value
    username = request.values.get('username').strip()
    #这里为了加强密码，这里我们在model的user里面加了盐.改完记得把数据库重新初始化一下
    password = request.values.get('password').strip()

    #判断是否为空
    if username=='' or password =='':
        return redirect_with_msg('/reg_login_page/',u'用户名或密码为空',category='reg_log')
    #判断是否重复,重复的话，flash一个消息过去。因为以后很多地方redirect的时候要带flash,所以我们特意打包一个函数在view的最上面
    user = User.query.filter_by(username = username).first()
    if user != None:
        return redirect_with_msg('/reg_login_page/',u'用户名已注册',category='reg_log')

    #生成盐
    salt = '.'.join(random.sample('0123456789abcdefgABCDEFG',10))
    #加密
    m = hashlib.md5()
    #这里改版之后，需要加encode
    m.update(password.encode("utf8")+salt.encode("utf8"))
    #加密之后的16进制字符串作为密码
    password = m.hexdigest()
```

```
    ##更多判断做完之后，将用户插入数据库
    user = User(username,password,salt)
    db.session.add(user)
    db.session.commit()
    # 注册完之后自动登录
    login_user(user)

    # 判断页面是否有next字段传入
    next = request.values.get('next')
    if next != None and next.startswith('/') > 0:
        return redirect(next)

    #没有next就跳转至首页
    return redirect('/')

@app.route('/login/',methods={'post','get'})
def login():
    username = request.values.get('username').strip()
    password = request.values.get('password').strip()
    # 判断是否为空
    if username == '' or password == '':
        return redirect_with_msg('/reg_login_page/', u'用户名或密码为空', category='reg_log')
    #看用户是否存在
    user = User.query.filter_by(username = username).first()
    if user == None:
        return redirect_with_msg('/reg_login_page/', u'用户不存在', category='reg_log')
    #验证密码
    m =hashlib.md5()
    #重新加密
    m.update(password.encode("utf8")+user.salt.encode("utf8"))
    #加密后的与数据库里的做比对
    if m.hexdigest() != user.password:
        return redirect_with_msg('/reg_login_page/', u'密码错误', category='reg_log')

    login_user(user)

    # 判断页面是否有next字段传入
    next = request.values.get('next')
    if next != None and next.startswith('/') >0:
        return redirect(next)
    #没有next就跳转至首页
    return redirect('/')


@app.route('/logout/')
def logout():
    logout_user()
    return redirect('/')
```

## b，用户数据安全性设置：

1. HTTPS注册页
2. 公钥加密私钥解密，支付宝h5页面的支付密码加密
3. 用户密码salt防止破解（CSDN，网易邮箱未加密密码泄漏）
4. token有效期
5. 单一平台的单点登陆，登陆IP异常检验
6. 用户状态的权限判断
7. 添加验证码机制，防止爆破和批量注册

## c，Flask-login

Flask-login集成了大部分功能，比如session，token下发验证都给做了。

### B1：Flask-login介绍：

（1），首先需要做一个实例登记：login_manager = LoginManager(app)。

（2），然后回调：

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(user_id)
```

通过session里面的id获取是哪个用户，

（3），之后是用户权限的验证：是否认证啊，激活啊，或者匿名

（4），最后最核心的就四个函数：

1，登入用户，将状态设置为登入

2，登出用户，删除session和登入状态

3，可以指定哪些页面是需要先登录的才能访问

4，当前用户的属性，如果是登入的，那么久可以用这个来访问当前用户的信息

B2：flask-login流程:先实例化登记，然后model.py里设置回调函数，然后在user class中添加四个用户接口函数。就可以使用四个核心函数了

（1）实例化登记：（init.py里）

```
login_manager = LoginManager(app)
```

（2）Model.py里设置回调函数

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(user_id)
```

（3）Class user里添加用户接口函数

```
  # 认证
    def is_authenticated(self):
        print('is_authenticated')
        return True
    #激活
    def is_active(self):
        print('is_active')
        return True
    #匿名
    def is_anonymous(self):
        print('is_anonymous')
        return False

    #这里不能加@property
    def get_id(self):
        print('get_id')
        return self.id
```

（4）核心函数使用：

```
@app.route('/addcomment/',methods={'post'})
@login_required
def add_comment():
```

或

```
<div class="web-menu">
                        {%if current_user.confirmed and current_user.is_authenticated%}
                            <a class="profile-ico" href="/profile/{{current_user.id}}">{{current_user.username}}</a>
                        {%elif current_user.confirmed == 0%}

                            <a class="profile-ico" href="/unconfirmed/">{{current_user.username}}</a>
                        {%else%}
                            <a class="profile-ico" href="/reg_login_page/">登录注册</a>
                        {%endif%}
</div>
```

## d, ajax异步刷新（主页）

### C1：index.html见3.5的c3

### C2：view.py里的异步加载更多图片

```
@app.route('/index/images/<int:page>/<int:per_page>/')
def index_images(page, per_page):
    #逆序读取分页
    paginate = Image.query.order_by(db.desc(Image.id)).paginate(page=page, per_page=per_page, error_out=False)
    #返回map
    map = {'has_next': paginate.has_next}
    images = []
    #遍历分页中的图片
    for image in paginate.items:
        comment_user_username = []
        comment_user_id = []
        comment_content = []
        for comments_i in image.comments:
            comment_user_username.append(comments_i.user.username)
            comment_user_id.append(comments_i.user.id)
            comment_content.append(comments_i.content)
```

```
        imgvo = {'id': image.id,
                 'url': image.url,
                 'imageusername': image.user.username,
                 'comment_count': len(image.comments),
                 'user_id': image.user_id,
                 'head_url': image.user.head_url,
                 'created_date': str(image.created_data),
                 'comment_user_username': comment_user_username,
                 'comment_user_id': comment_user_id,
                 'comment_content': comment_content}
        images.append(imgvo)


    map['images'] = images

    return json.dumps(map)
```

## C3，index_profile.js

```
$(function () {
    var oExports = {
        initialize: fInitialize,
        // 渲染更多数据
        renderMore: fRenderMore,
        // 请求数据
        requestData: fRequestData,
        // 简单的模板替换
        tpl: fTpl
    };
    // 初始化页面脚本
    oExports.initialize();


    function fInitialize() {
        var that = this;
        // 常用元素
        that.listEl = $('div.js-image-list');
        // 初始化数据
        that.page = 1;
        that.pageSize = 10;
        that.listHasNext = true;
        // 绑定事件
        $('.js-load-more').on('click', function (oEvent) {
            //alert('执行onclick')
            var oEl = $(oEvent.currentTarget);
            var sAttName = 'data-load';
            // 正在请求数据中，忽略点击事件
            if (oEl.attr(sAttName) === '1') {
                return;
            }
            // 增加标记，避免请求过程中的频繁点击
            oEl.attr(sAttName, '1');
            that.renderMore(function () {
                // 取消点击标记位，可以进行下一次加载
                oEl.removeAttr(sAttName);
                // 没有数据隐藏加载更多按钮
                !that.listHasNext && oEl.hide();
            });
        });
    }


    function fRenderMore(fCb) {
        //alert('执行rendermore')
        var that = this;
        // 没有更多数据，不处理
        if (!that.listHasNext) {
            return;
        }
        that.requestData({
            page: that.page + 1,
            pageSize: that.pageSize,
            call: function (oResult) {
                //alert(oResult)
                // 是否有更多数据
                that.listHasNext = !!oResult.has_next && (oResult.images || []).length > 0;
                // 更新当前页面
                that.page++;
                // 渲染数据
                var sHtml = '';
                $.each(oResult.images, function (nIndex, oImage) {
                    sHtml_1 = that.tpl([
                        '<article class="mod">',
                            '<header class="mod-hd">',
                                '<time class="time">#{created_date}</time>',
```

```
                                '<a href="/profile/#{user_id}" class="avatar">',
                                    '<img src="#{head_url}">',
                                '</a>',
                                '<div class="profile-info">',
                                    '<a title="#{imageusername}" href="/profile/#{user_id}">#{imageusername}</a>',
                                '</div>',
                            '</header>',
                            '<div class="mod-bd">',
                                '<div class="img-box">',
                                    '<a href="/image/#{id}">',
                                        '<img src="#{url}">',
                                    '</a>',
                                '</div>',
                            '</div>',
                            '<div class="mod-ft">',
                                '<ul class="discuss-list js-discuss-list" id="ul#{id}">',
                                    '<li class="more-discuss">',
                                        '<a>',
                                            '<span>全部 </span><span class="">#{comment_count}</span>',
                                            '<span> 条评论</span></a>',
                                    '</li>'].join(''), oImage);
                        //alert(sHtml_1)
                        sHtml_2 = ' ';
                        for (var ni = 0; ni < oImage.comment_count; ni++){
                            //alert(ni)
                            dict = {'comment_user_username':oImage.comment_user_username[ni], 'comment_user_id':oImage.comment_user_id[ni],
                                'comment_content':oImage.comment_content[ni] };
                            //alert(dict)
                            sHtml_2 += that.tpl([
                                '<li>',
                                '<a class="_4zhc5 _iqaka" title="#{comment_user_username}" href="/profile/#{comment_user_id}" data-
reactid=".0.1.0.0.0.2.1.2:$comment-17856951190001917.1">#{comment_user_username}</a>',
                                '<span>',
                                '<span>#{comment_content}</span>',
                                '</span>',
                                '</li>'].join(''), dict);


                        }
                        //alert(sHtml_2)
                        sHtml_3 = that.tpl([
                                    '</ul>',
                                    '<section class="discuss-edit">',
                                        '<a class="icon-heart-empty"></a>',
                                            '<form>',
                                                '<input placeholder="添加评论..." id="jsCmt#{id}" type="text">',
                                            '</form>',
                                            '<button class="more-info" id="jsSubmit#{id}" onclick="mao(#{id})">更多选项</button>',
                                    '</section>',
                                '</div>',
                            '</article>'].join(''), oImage);
                        //alert(sHtml_3)
                        sHtml += sHtml_1  +sHtml_2+ sHtml_3;
                    });
                    //alert(sHtml)
                    sHtml && that.listEl.append(sHtml);
                },
                error: function () {
                    alert('出现错误，请稍后重试');
                },
                always: fCb
            });
        }


        function fRequestData(oConf) {
            //alert('执行frequest')
            var that = this;
            var sUrl = '/index/images/' + oConf.page + '/' + oConf.pageSize + '/';
            //alert(sUrl)
            $.ajax({url: sUrl, dataType: 'json'}).done(oConf.call).fail(oConf.error).always(oConf.always);

        }


        function fTpl(sTpl, oData) {
            var that = this;
            sTpl = $.trim(sTpl);
            return sTpl.replace(/#{(.*?)}/g, function (sStr, sName) {
                return oData[sName] === undefined || oData[sName] === null ? '' : oData[sName];
            });
        }
    });
```

## e，next跳转优化

这里我们还引入了next来记录用户登录之前在访问的页面，然后登录成功后跳转回去，来做用户体验的优化

#### D1：Login.html里面：增加一个隐藏字段

```
<input type="hidden" name="next" value="{{next}}"/>
```

#### D2：View.py里面

```
@app.route('/reg_login_page/')
def reg_login():
    #这里用来获取注册时反馈的flash
    msg =''
    for m in get_flashed_messages(with_categories=False,category_filter=['reg_log']):
        msg =msg+m
    #这里我们增加一个字段next，来记录登录后想跳转到的界面.同时，我们需要在login界面里面加一个隐藏字段，来获取next
    #再同时，我们需要在login函数里面做一个判断，如果有next字段，则跳转至next
    return render_template('login.html',msg=msg,next =request.values.get('next'))
```

## 3.5，图片上传+七牛+评论

### a，图片上传和显示

#### A1：首先app.conf设置如下：

```
ALLOWED_EXT =set(['png','jpg','jpeg','bmp','gif'])
UPLOAD_DIR ='/home/j/upload'
```

#### A2：然后view.py视图里：

```
def save_to_local(file,filename):
    file_dir =app.config['UPLOAD_DIR']
    #存贮文件
    file.save(os.path.join(file_dir,filename))
    #返回访问地址,可以通过这个地址从浏览器访问该存贮文件
    return '/image/'+filename


#显示图片,根据上面函数的返回地址定义
@app.route('/image/<filename>/')
def show_image(filename):
    return send_from_directory(app.config['UPLOAD_DIR'],filename)


#这里必须用post方法
@app.route('/upload/',methods={'post'})
def upload():
    #获取上传文件的信息
    file =request.files['file']
    #获取多张图片
    #file1 =request.files['file1']

    #上传至服务器
    #后缀名验证（放在app.conf里）
    file_ext =''
    if file.filename.find('.') >0:
        file_ext =file.filename.rsplit('.',1)[1].strip().lower()


    if file_ext in app.config['ALLOWED_EXT']:
        #保存文件，重新定义文件名，避免不规范
        filename =str(uuid.uuid1()).replace('-','')+'.'+file_ext
        #再调用自己定义的函数,存贮至本地
        url =save_to_local(file,filename)
        #url =qiniu_update_file(file,filename)
        #入数据库
        if url !=None:
            db.session.add(Image(url,current_user.id))
            db.session.commit()


    #上传完返回用户首页
    return redirect('/profile/%d'%current_user.id)
```

#### A3，profile.html里

```
<span class="_jxp6f _e6l6g" style="display:inline-block;position:relative;">
                        <form method="post" action="/upload/" enctype="multipart/form-data">
                        <button class="btn-success">上传图片</button>
                        <input name="file" type="file" onchange="this.parentNode.submit()" style="opacity:0;position:absolute;top:0;left:0;display:blo
ck;width:100%;height:100%;">
                        </form>
</span>
```

#### A4，profile.js(可以异步显示更多数据）

```
$(function () {
    var oExports = {
        initialize: fInitialize,
        // 渲染更多数据
        renderMore: fRenderMore,
        // 请求数据
        requestData: fRequestData,
        // 简单的模板替换
        tpl: fTpl
    };
    // 初始化页面脚本
    oExports.initialize();

    function fInitialize() {
        var that = this;
        // 常用元素
        that.listEl = $('div.js-image-list');
        // 初始化数据
        that.uid = window.uid;
        that.page = 1;
        that.pageSize = 3;
        that.listHasNext = true;
        // 绑定事件
        $('.js-load-more').on('click', function (oEvent) {
            var oEl = $(oEvent.currentTarget);
            var sAttName = 'data-load';
            // 正在请求数据中，忽略点击事件
            if (oEl.attr(sAttName) === '1') {
                return;
            }
            // 增加标记，避免请求过程中的频繁点击
            oEl.attr(sAttName, '1');
            that.renderMore(function () {
                // 取消点击标记位，可以进行下一次加载
                oEl.removeAttr(sAttName);
                // 没有数据隐藏加载更多按钮
                !that.listHasNext && oEl.hide();
            });
        });
    }

    function fRenderMore(fCb) {
        var that = this;
        // 没有更多数据，不处理
        if (!that.listHasNext) {
            return;
        }
        that.requestData({
            uid: that.uid,
            page: that.page + 1,
            pageSize: that.pageSize,
            call: function (oResult) {
                // 是否有更多数据
                that.listHasNext = !!oResult.has_next && (oResult.images || []).length > 0;
                // 更新当前页面
                that.page++;
                // 渲染数据
                var sHtml = '';
                $.each(oResult.images, function (nIndex, oImage) {
                    sHtml += that.tpl([
                        '<a class="item" href="/image/#{id}">',
                            '<div class="img-box">',
                                '<img src="#{url}">',
                            '</div>',
                            '<div class="img-mask"></div>',
                            '<div class="interaction-wrap">',
                                '<div class="interaction-item"><i class="icon-comment"></i>#{comment_count}</div>',
                            '</div>',
                        '</a>'].join(''), oImage);
                });
                sHtml && that.listEl.append(sHtml);
            },
            error: function () {
                alert('出现错误，请稍后重试');
            },
```

```
            always: fCb
        });
    }

    function fRequestData(oConf) {
        var that = this;
        var sUrl = '/profile/images/' + oConf.uid + '/' + oConf.page + '/' + oConf.pageSize + '/';
        //alert(sUrl)
        $.ajax({url: sUrl, dataType: 'json'}).done(oConf.call).fail(oConf.error).always(oConf.always);
    }

    function fTpl(sTpl, oData) {
        var that = this;
        sTpl = $.trim(sTpl);
        return sTpl.replace(/#{(.*?)}/g, function (sStr, sName) {
            return oData[sName] === undefined || oData[sName] === null ? '' : oData[sName];
        });
    }
});
```

## b，七牛云存贮

将图片放其他平台存贮的好处是，无需让大量静态的图片去占用太多的带宽，此外，无需将图像冗余备份到每台机子。最后缩图的话，可以直接根据规范来调整云链接中的参数即可。

### B1:注册七牛账户（我自己的：2515418348@qq.com，pw：hanguzuyin），并pip install qiniu

### B2：app.conf里的设置，这些参数都是在七牛网注册后得到的

```
QINIU_ACCESS_KEY='tTi-LQws8Wt-eot-zATn24rs5r19AAEuP-EtWpjZ'
QINIU_SECRET_KEY='3tJstPV1992pV5s1I1iLQZhsFsnMk_BpkZOYZAob'
QINIU_BUCKET_NAME = 'jing'
QINIU_DOMAIN='oyf8hop5q.bkt.clouddn.com'
```

### B3，新建文件qiniusdk.py

```
#-*-encoding=UTF-8-*-
from qiniu import Auth,put_data
from stagram import app

#设置ak，sk，bucket_name
q = Auth(app.config['QINIU_ACCESS_KEY'], app.config['QINIU_SECRET_KEY'])
#要上传的空间
bucket_name = app.config['QINIU_BUCKET_NAME']
#domain
domain_prefix = app.config['QINIU_DOMAIN']


def qiniu_update_file(source_file,save_file_name):
    # 生成上传 Token，可以指定过期时间等
    token = q.upload_token(bucket_name, save_file_name)
    #print(source_file))
    localfile = '/home/j/桌面/cb6011c5b7274c7c375f011ca93c4a2f.jpg'
    #ret, info = put_file(token, save_file_name, localfile)
    #source_file.read_into(f)
    #这个七牛网的接口，适合于python2.7.3.5的版本有问题
    ret, info = put_data(token, save_file_name, source_file.stream)
    #上传成功，返回可访问的地址
    if info.status_code == 200:
        return domain_prefix + save_file_name
    return None
```

### B4，将3.5的A2里面的换成下面这行即可，

```
 #再调用自己定义的函数,存贮至本地
        url = save_to_local(file,filename)
        #url = qiniu_update_file(file,filename)
```

## c，评论功能

### C1,model.py里增加comment表和与其他表之间的外键关系，然后重新初始化数据库

```
#评论类
class Comment(db.Model):
    #评论id
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    #内容
    content = db.Column(db.String(1024))
    #评论是属于那张图片的
    image_id = db.Column(db.Integer,db.ForeignKey('image.id'))
```

```
    #评论是谁发的
    user_id = db.Column(db.Integer,db.ForeignKey('user.id'))
    #设计一个字段，来表明当前实体属于什么状态
    status = db.Column(db.Integer,default=0)#0,正常，1，被删除
    #将评论和用户关联起来
    user = db.relationship('User')


    def __init__(self, content, image_id, user_id):
        self.content = content
        self.image_id = image_id
        self.user_id = user_id


    def __repr__(self):
        return '<comment %d: %s>'%(self.id,self.content)
```

## C2，在view.py里添加为图片详情页和主页添加评论的函数：

```
@app.route('/addcomment/',methods={'post'})
@login_required
def add_comment():


    image_id = int(request.values['image_id'])
    content = request.values['content']
    comment = Comment(content,image_id,current_user.id)
    db.session.add(comment)
    db.session.commit()
    dic = {'code':0,'id': comment.id,'content': comment.content,'username': comment.user.username,'user_id': comment.user_id}
    return json.dumps(dic)


@app.route('/addindexcomment/',methods={'post'})
@login_required
def add_index_comment():
    image_id = int(request.values['image_id'])
    content = request.values['content']
    comment = Comment(content,image_id,current_user.id)
    db.session.add(comment)
    db.session.commit()
    dic = {'code':0,'id': comment.id,'content': comment.content,'username': comment.user.username,'user_id': comment.user_id,'image_id':comment.image
_id}
    return json.dumps(dic)
```

## C3，首页和图片详情页的html

首页index.html：

```
{%extends 'base.html'%}
{%block title%}首页-{{current_user.username}}{%endblock%}
{%block css%}/static/styles/pages/index.css {%endblock%}

{%block content%}
<div class="page-main ">
    <div class="list clearfix js-image-list">
        {%for image in images%}


        <article class="mod clearfix">
                <header class="mod-hd clearfix">
                    <time class="time">{{image.created_data}}</time>
                    <a href="/profile/{{image.user.id}}" class="avatar">
                        <img src="{{image.user.head_url}}">
                    </a>
                    <div class="profile-info">
                        <a title="{{image.user.username}}" href="/profile/{{image.user.id}}">{{image.user.username}}</a>
                    </div>
                </header>
                <div class="mod-bd clearfix">
                    <div class="img-box">
                        <a href="/image/{{image.id}}">
                            <img src="{{image.url}}">
                        </a>
                    </div>
                </div>
                <div class="mod-ft clearfix">
                    <!--<section class="times">
```

```
                        <span></span>
                        <span>6.2百万</span>
                        <span> 次播放</span>
                    </section>-->
                    <ul class="discuss-list js-discuss-list" id="ul{{image.id}}">
                        <li class="more-discuss">
                            <a>
                                <span>全部 </span><span class="">{{image.comments|length}}</span>
                                <span> 条评论</span></a>
                        </li>
                        {%for comment in image.comments%}
                        {#这里设置最多显示两条评论,直接运行会报错，因为这里不支持break语言，要支持，需要在init里加上
app.jinja_env.add_extension('jinja2.ext.loopcontrols')#}
                        {%if loop.index >2%}
                            {%break%}
                        {%endif%}
                            <li>
                                <!--<a class=" icon-remove" title="删除评论"></a>-->
                                <a class="_4zhc5 _iqaka" title="{{comment.user.username}}" href="/profile/{{comment.user.id}}" data-
reactid=".0.1.0.0.0.2.1.2:$comment-17856951190001917.1">{{comment.user.username}}</a>
                                <span>
                                    <span>{{comment.content}}</span>
                                </span>
                            </li>
                        {%endfor%}

                    </ul>

                    <section class="discuss-edit">
                        <a class="icon-heart-empty"></a>
                            <form>

                                <input placeholder="添加评论..." id="jsCmt{{image.id}}" type="text">

                            </form>
                            {%if current_user.is_authenticated%}
                            <button class="more-info" id="jsSubmit{{image.id}}" onclick="mao('{{image.id}}','yes')">更多选项</button>
                            {%else%}
                            <button class="more-info" id="jsSubmit{{image.id}}" onclick="mao('{{image.id}}','no')">更多选项</button>
                            {%endif%}

                    </section>

                </div>
            </article>
        {%endfor%}
    </div>
{% if has_next %}
    <div class="more-content js-load-more">
            <a class="_oidfu" href="javascript:void(0);">更多</a>
    </div>
{% endif %}
</div>



{%endblock%}
{% block js %}
<script type="text/javascript" src="/static/js/jquery.js"></script>
<script type="text/javascript" src="/static/js/index_detail.js"></script>
<script type="text/javascript" src="/static/js/index_profile.js"></script>
{% endblock%}
```

图片详情页pageDetail.html：

```
{%extends 'base.html'%}
{%block title%}图片页{%endblock%}
{%block css%}/static/styles/pages/detail.css {%endblock%}
{%block content%}
<div class="page-main clearfix">
    <article>
        <div class="pic-wrapper">
            <img src="{{image.url}}">
        </div>
        <div class="pic-discuss">
            <header class="discuss-hd">
                <a href="/profile/{{image.user.id}}" class="avatar">
                    <img src="{{image.user.head_url}}">
```

```
                </a>
                <div class="profile-info">
                    <a title="{{image.user.username}}" href="/profile/{{image.user.id}}">{{image.user.username}}</a>
                </div>
            </header>

            <ul class="discuss-list js-discuss-list">

                {%for comment in image.comments%}
                {%if loop.index >2%}
                        {%break%}
                {%endif%}
                <li>
                    <!--<a class=" icon-remove" title="删除评论"></a>-->
                    <a class="_4zhc5 _iqaka" title="{{comment.user.username}}" href="/profile/{{comment.user.id}}" data-reactid=".0.1.0.0.0.2.1.2:
$comment-17856951190001917.1">{{comment.user.username}}</a>
                    <span>
                        <span>{{comment.content}}</span>
                    </span>
                </li>
                {%endfor%}

            </ul>
            <section class="discuss-edit">
                <a class="icon-heart-empty"></a>
                <form>
                    <input placeholder="添加评论..." id="jsCmt" type="text">
                </form>
                <button class="more-info" id="jsSubmit">更多选项</button>
            </section>


        </div>
    </article>
</div>
{%endblock%}
{% block js %}
<script type="text/javascript">
    window.imageId ={{image.id}};
</script>
<script type="text/javascript" src="/static/js/jquery.js"></script>
<script type="text/javascript" src="/static/js/detail.js"></script>
{% endblock%}
```

## C4,首页和图片详情页的js

Index_detail.js

```
function mao(fff,t){
    if(t == 'no'){
        alert('请登录后评论')
    }

    var oExports = {
        initialize: fInitialize(fff),
        encode: fEncode
    };
    oExports.initialize();

    function fInitialize(fff) {

        //alert(image_id)
        var that = this;
        //var test = new RegExp("jsSubmit")
        var sImageId =fff;

        var oCmtIpt = $('#jsCmt'+fff);
        //var oListDv = $('ul.discuss-list-js-discuss-list'+fff);
        var oListDv = $('#ul'+fff);
        var s = 'jsSubmit' + sImageId.toString()

        // 点击添加评论
        var bSubmit = false;
        var id = sImageId;
        //alert(id)
            var sCmt = $.trim(oCmtIpt.val());
            // 评论为空不能提交
```

```
            if (!sCmt) {
                return alert('评论不能为空');
            }
            // 上一个提交没结束之前，不再提交新的评论
            if (bSubmit) {
                return;
            }
            bSubmit = true;
            $.ajax({
                url: '/addindexcomment/',
                type: 'post',
                data: {image_id: sImageId, content: sCmt}
            }).done(function (oResult) {
                oResult = eval('(' + oResult + ')')
                //alert(oResult.code)
                if (oResult.code !== 0) {


                    return alert(oResult.msg || '提交失败1，请重试');
                }
                // 清空输入框
                oCmtIpt.val('');
                // 渲染新的评论
                if (oResult.image_id == sImageId){
                    //alert(fEncode(oResult.username))
                    //alert((oResult.username))
                    var sHtml = [
                        '<li>',
                        '<a class="_
4zhc5 _iqaka" title="', fEncode(oResult.username), '" href="/profile/', oResult.user_id, '">', fEncode(oResult.username), '</a> ',
                        '<span><span>', fEncode(sCmt), '</span></span>',
                        '</li>'].join('');
                    oListDv.prepend(sHtml);
                }


            }).fail(function (oResult) {
                alert(oResult.msg || '请登录后评论');
            }).always(function () {
                bSubmit = false;
            });
    }


    function fEncode(sStr, bDecode) {
        var aReplace =["&#39;", "'", "&quot;", '"', " ", " ", "&gt;", ">", "&lt;", "<", "&amp;", "&", "&yen;", "¥"];
        !bDecode && aReplace.reverse();
        for (var i = 0, l = aReplace.length; i < l; i += 2) {
             sStr = sStr.replace(new RegExp(aReplace[i],'g'), aReplace[i+1]);
        }
        return sStr;
    };

}
```

Detail.js(图片详情页的 )

```
$(function () {
    var oExports = {
        initialize: fInitialize,
        encode: fEncode
    };
    oExports.initialize();


    function fInitialize() {
        var that = this;
        var sImageId = window.imageId;
        var oCmtIpt = $('#jsCmt');
        var oListDv = $('ul.js-discuss-list');

        // 点击添加评论
        var bSubmit = false;
        $('#jsSubmit').on('click', function () {
            var sCmt = $.trim(oCmtIpt.val());
            // 评论为空不能提交
            if (!sCmt) {
                return alert('评论不能为空');
            }
```

```
                // 上一个提交没结束之前，不再提交新的评论
                if (bSubmit) {
                    return;
                }
                bSubmit = true;
                $.ajax({
                    url: '/addcomment/',
                    type: 'post',
                    data: {image_id: sImageId, content: sCmt}
                }).done(function (oResult) {
                    oResult = eval('(' + oResult + ')')


                    if (oResult.code !== 0) {


                        return alert(oResult.msg || '提交失败1，请重试');
                    }
                    // 清空输入框
                    oCmtIpt.val('');
                    // 渲染新的评论
                    var sHtml = [
                        '<li>',
                            '<a class="_
4zhc5 _iqaka" title="', that.encode(oResult.username), '" href="/profile/', oResult.user_id, '">', that.encode(oResult.username), '</a> ',
                            '<span><span>', that.encode(sCmt), '</span></span>',
                        '</li>'].join('');
                    oListDv.prepend(sHtml);
                }).fail(function (oResult) {
                    alert(oResult.msg || '提交失败2，请重试');
                }).always(function () {
                    bSubmit = false;
                });
            });
        }


        function fEncode(sStr, bDecode) {
            var aReplace =["&#39;", "'", "&quot;", '"', " ", " ", "&gt;", ">", "&lt;", "<", "&amp;", "&", "&yen;", "¥"];
            !bDecode && aReplace.reverse();
            for (var i = 0, l = aReplace.length; i < l; i += 2) {
                sStr = sStr.replace(new RegExp(aReplace[i],'g'), aReplace[i+1]);
            }
            return sStr;
        };

    });
```

## 3.6，邮箱激活功能，Flask-Mail（参考链接：http://python.jobbole.com/81410/。git地址：https://github.com/jing1900/p1__email.git）

### a，先pip install Flask-Mail，然后更新login.html页面

```
{%extends 'base.html'%}
{%block title%}注册登录页{%endblock%}
{%block css%}/static/styles/pages/login.css {%endblock%}
{%block content%}
<main class="main login-main" role="main">
    <article class="login-cont clearfix">
        <div class="login-pic-box">
            <img src="/static/images/res/nkFamily.jpg">
        </div>
        <div class="login-wrapper">
            <div class="login-box">
                <form method ="post",id="reg_login_form">
                    <h2 class="login-hd">
                        {%if (msg|length) >0%}
                            <b>{{msg}}</b>
                        {%else%}
                            注册 instagram
                        {%endif%}</h2>
                    <div class="form-item">
                        <input class="input-txt" aria-label="用户名" aria-
required="true" autocapitalize="off" autocorrect="off" maxlength="30" name="username" placeholder="用户名" value="" type="text" data-
reactid=".0.1.0.1.0.1.0.5.0">
                    </div>
                    {%if msg =='验证邮件已发送'%}
                    {%else%}
                    <div class="form-item">
```

```
                    <input class="input-txt" aria-label="邮箱" aria-
required="true" autocapitalize="off" autocorrect="off" maxlength="30" name="email" placeholder="邮箱" value="" type="text" data-
reactid=".0.1.0.1.0.1.0.5.0">
                </div>
                {%endif%}
                <div class="form-item">
                    <input class="input-txt" aria-describedby="" aria-label="密码" aria-
required="true" autocapitalize="off" autocorrect="off" name="password" placeholder="密码" type="password" value="" data-reactid=".0.1.0.1.0.1.0.6.0">
                </div>
                <input type="hidden" name="next" value="{{next}}"/>
                <div class="btn-wrapper">
                    <button class="btn-primary" onclick="document.getElementById('reg_login_form',form.action='/reg/')">注册</button>
                    <button class="btn-primary" onclick="document.getElementById('reg_login_form',form.action='/login/')">登录</button>
                </div>
                <p class="agreenment">
                    <span>注册即表示你同意我们的 </span>
                    <a class="agreenment-link" href="/legal/terms/" target="_blank">条款</a>
                    <span data-reactid=".0.1.0.1.0.1.0.9.2"> 和 </span>
                    <a class="agreenment-link" href="/legal/privacy/" target="_blank">隐私权政策</a>
                    <span> 。</span>
                </p>
            </form>
        </div>
    </div>
</article>
</main>
{%endblock%}
```

## b，先在model.py的User里增加两个字段，（是否邮箱确认和确认时间），接下来重新init_database，以使数据库生效

```
class User(db.Model):
    # __tablename__ = 'myuser' 指定表名字,不指定就默认类名小写
    '''这里类里的一个变量，就表示表中的一列，具体怎样跟数据库做交互见manage.py'''
    #user id,指明类型，是否主键和是否自动增长
    id = db.Column(db.Integer,primary_key=True,autoincrement=True)
    #用户名，指明类型，和非重复
    username = db.Column(db.String(80),unique=True)
    #邮箱
    #email = db.Column(db.String(80))
    email = db.Column(db.String(80), unique=True, nullable=False)
    #密码
    password = db.Column(db.String(32))

    #头像
    head_url = db.Column(db.String(256))
    #是否邮箱确认
    confirmed = db.Column(db.Boolean, nullable=False, default=False)
    #邮箱确认时间
    confirmed_on = db.Column(db.DateTime, nullable=True)
    # 盐
    salt = db.Column(db.String(32))

    #这里我们怎么将每个人发的图片关联起来呢
    images = db.relationship('Image',backref='user',lazy='dynamic')

    '''定义构造函数'''
    def __init__(self,username,email,password,confirmed,confirmed_on=None,salt =''):
        self.username = username
        self.email = email
        self.password = password
        self.confirmed = confirmed
        self.confirmed_on = confirmed_on
        self.salt = salt
        #这里头像先用牛客网给出的1000张图片之一，中间的变量是0-1000之间随机一个整数
        self.head_url = 'http://images.nowcoder.com/head/' + str(random.randint(0, 1000)) + 't.png'
```

## c，app.conf里设置，这里我们使用qq邮箱作为发送邮箱

```
SECRET_KEY ='jing'
MAIL_SERVER='smtp.qq.com'
MAIL_PORT = 465
MAIL_USE_SSL= True
MAIL_USERNAME ='2515418348@qq.com'
''授权码'''
MAIL_PASSWORD ='uidzxmtzynltdhie'
```

```
SECURITY_PASSWORD_SALT = 'ilove'
```

授权码获取：登录qq邮箱，进入设置页面，切换到账户选项，开启POP3/IMAP服务，获取授权码

## d，添加token.py，用来生成和验证token

```
'''生成token'''
from itsdangerous import URLSafeTimedSerializer


from stagram import app


def generate_confirmation_token(email):
    # 过URLSafeTimedSerializer用在用户注册时得到的email地址生成一个令牌。
    serializer = URLSafeTimedSerializer(app.config['SECRET_KEY'])
    return serializer.dumps(email, salt=app.config['SECURITY_PASSWORD_SALT'])


#确认令牌之后，在confirm_token()函数中，我们可以用loads()方法，
# 它接管令牌和其过期时间——一个小时（3600秒）内有效——作为参数。
# 只要令牌没过期，那它就会返回一个email。
def confirm_token(token, expiration=3600):
    serializer = URLSafeTimedSerializer(app.config['SECRET_KEY'])
    try:
        email = serializer.loads(
            token,
            salt=app.config['SECURITY_PASSWORD_SALT'],
            max_age=expiration
        )
    except:
        return False
    return email
```

## e,更新用户注册函数如下：（view.py里，修改完数据库后，插入的代码也要对应修改下）

### E1：在此之前，写一个active.html确认邮件页面：

```
<p>Welcome! Thanks for signing up. Please follow this link to activate your account:</p>
<p><a href="{{ confirm_url }}">{{ confirm_url }}</a></p>
<br>
<p>Cheers!</p>
```

### E2：注册函数（发邮件，然后转入未激活状态页面）

```
@app.route('/reg/',methods={'post','get'})
def reg():
    #request.args:url里面的参数
    #request.form:body里面的数据，也可以用value
    username = request.values.get('username').strip()
    email = request.values.get('email').strip()
    #注册时设置是否确认邮箱为false，即在插入数据库表时，该值为0
    confirmed = False
    #这里为了加强密码，这里我们在model的user里面加了盐.改完记得把数据库重新初始化一下
    password = request.values.get('password').strip()

    #判断是否为空
    if username=='' or email =='' or password =='' :
        return redirect_with_msg('/reg_login_page/',u'用户名或邮箱或密码为空',category='reg_log')
    #判断是否是邮箱格式
    if re.match("[a-zA-Z0-9]+\@+[a-zA-Z0-9]+\.+[a-zA-Z]", email) == None:
        return redirect_with_msg('/reg_login_page/',u'邮箱格式不正确',category='reg_log')

    #判断是否重复,重复的话，flash一个消息过去。因为以后很多地方redirect的时候要带flash,所以我们特意打包一个函数在view的最上面
    user = User.query.filter_by(username = username).first()
    if user != None:
        return redirect_with_msg('/reg_login_page/',u'用户名已注册',category='reg_log')
    #判断邮箱是否已注册
    user = User.query.filter_by(email=email).first()
    if user != None:
        return redirect_with_msg('/reg_login_page/', u'邮箱已注册', category='reg_log')

    #生成盐
    salt = '.'.join(random.sample('0123456789abcdefgABCDEFG',10))
    #加密
    m = hashlib.md5()
```

```
    #这里改版之后，需要加encode
    m.update(password.encode("utf8")+salt.encode("utf8"))
    #加密之后的16进制字符串作为密码
    password = m.hexdigest()
    #print(salt)

    ##更多判断做完之后，将用户插入数据库,默认确认状态为0
    user = User(username,email,password,0,0,salt)


    #def __init__(self, username, email, password, confirmed, confirmed_on=None, salt=''):
    db.session.add(user)
    db.session.commit()

    #获取token
    token = generate_confirmation_token(email)
    confirm_url = url_for('confirm_email', token=token, _external=True)
    html = render_template('active.html', confirm_url=confirm_url)

    #这里发验证邮件，传入user参数
    msg = Message('Confirm Your Account', sender='2515418348@qq.com', recipients=[str(email)])
    msg.html = html
    mail.send(msg)

    #登入用户，但转入未邮件验证的页面
    login_user(user)
    return redirect('/unconfirmed/')

#待确认页面
@app.route('/unconfirmed/')
@login_required
def unconfirmed():
    if current_user.confirmed:
        return redirect('/')
    flash('Please confirm your account!', 'warning')
    return render_template('/unconfirmed.html')
```

### E3：待确认页面unconfirmed.html：

```
{%extends 'base.html'%}
{%block css%}/static/styles/pages/index.css {%endblock%}
{% block content %}
<br><br><br>
<p align="center" >
You have not confirmed your account. Please check your inbox (and your spam folder) - you should have received an email with a confirmation link.
</p>
<br><br><br>
<p align="center">Didn't get the email? <a href="{{ confirm_url }}">Resend</a></p>
{% endblock %}
```

### f，添加新的视图处理邮件确认：（view.py里），用户点击邮箱里的链接，会激活该函数，从而更改数据库中的激活状态。

```
import datetime
#邮件确认
'''现在我们通过令牌调用confirm_token()函数。
如果成功，我们更新用户，把email_confirmed属性改成True， 设置datetime为验证发生的时间。
还有，要是用户已经进行过一遍验证过程了——而且已经验证了——我们要提醒用户这点。'''
@app.route('/confirm/<token>')
@login_required
def confirm_email(token):
    #print('run')
    try:
        email = confirm_token(token)
        #print(email)
    except:
        flash('The confirmation link is invalid or has expired.', 'danger')
    user = User.query.filter_by(email=email).first_or_404()
    #print(user)
    if user.confirmed:
        flash('Account already confirmed. Please login.', 'success')
    else:
        #print(user)
        user.confirmed = True
        user.confirmed_on = datetime.datetime.now()
        db.session.add(user)
        db.session.commit()
```

```
            flash('You have confirmed your account. Thanks!', 'success')
        return redirect('/')
```

g,创建装饰器来判断用户是否已激活，否则则进入待确认页面（decorators.py）：

G1：装饰器

```
from functools import wraps


from flask import flash, redirect, url_for
from flask_login import current_user
'''装饰器，验证是否已确认邮件'''
def check_confirmed(func):
    @wraps(func)
    def decorated_function(*args, **kwargs):
        if current_user.confirmed is False:
            flash('Please confirm your account!', 'warning')
            return redirect('/unconfirmed/')
        return func(*args, **kwargs)


    return decorated_function
```

G2：然后对profile视图添加装饰器：

```
@app.route('/profile/<int:user_id>/')
@login_required#访问权限设置
@check_confirmed
def user_detail(user_id):
```

h,更新base.html的代码，header部分如下，以根据激活状态显示不同的页面

```
<header class="header">
            <div class="header-cont">
                <a class="logo" href="/">logo</a>
                <div class="web-menu">
                    {%if current_user.confirmed and current_user.is_authenticated%}
                        <a class="profile-ico" href="/profile/{{current_user.id}}">{{current_user.username}}</a>
                    {%elif current_user.confirmed == 0%}


                        <a class="profile-ico" href="/unconfirmed/">{{current_user.username}}</a>
                    {%else%}
                        <a class="profile-ico" href="/reg_login_page/">登录注册</a>
                    {%endif%}
                </div>
            </div>
    </header>
```

## 3.7，淘宝女装爬虫（https://github.com/jing1900/spider.git）

a,运行前的准备工作：先在model.py里，添加下面的内容，然后重新在terminal里面运行init_database()

```
from stagram import db,login_manager
from datetime import datetime
import random
from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
from stagram import app
#服装类
class Address(db.Model):
    goods_id = db.Column(db.String(32), primary_key=True,nullable=False)
    shop_id = db.Column(db.String(80))
    shop_loc = db.Column(db.String(80))
    shop_name = db.Column(db.String(80))
    goods_title = db.Column(db.String(80))
    view_sales = db.Column(db.String(80))
    view_price = db.Column(db.String(80))
    comment_url = db.Column(db.String(1000))
    pic_url = db.Column(db.String(200))
    sale_counts = db.Column(db.Integer)
    def __init__(self):
        pass
    def __repr__(self):
        pass
```

## b，运行spider项目代码里的__init__.py

这里如果需要爬其他类目的，只需更改spider项目里的dict.py文件

### B1，dict.py

```
# coding=utf-8
# usr/bin/eny python


urldict = {
    'https://s.taobao.com/search?spm=a230r.1.0.0.aCclFK&q=%E7%9C%9F%E4%B8%9D%E8%BF%9E%E8%A1%A3%E8%A3%99':'真丝连衣裙',
    'https://s.taobao.com/search?spm=a230r.1.0.0.ijf5DJ&q=%E5%8D%8A%E8%BA%AB%E8%A3%99':'半身裙',
    'https://s.taobao.com/search?spm=a230r.1.0.0.baKYzv&q=%E6%A3%89%E9%BA%BB%E8%BF%9E%E8%A1%A3%E8%A3%99':'棉麻连衣裙',
    'https://s.taobao.com/search?spm=a230r.1.0.0.0pc0bG&q=%E9%95%BF%E8%A2%96%E8%BF%9E%E8%A1%A3%E8%A3%99':'长袖连衣裙',
    'https://s.taobao.com/search?spm=a230r.1.0.0.uxnjgr&q=%E8%95%BE%E4%B8%9D%E8%BF%9E%E8%A1%A3%E8%A3%99':'蕾丝连衣裙',
    'https://s.taobao.com/search?spm=a230r.1.0.0.POYxKd&q=%E9%9B%AA%E7%BA%BA%E8%BF%9E%E8%A1%A3%E8%A3%99':'雪纺连衣裙',
    'https://s.taobao.com/search?spm=a230r.1.0.0.8WLuAa&q=%E9%92%88%E7%BB%87%E8%BF%9E%E8%A1%A3%E8%A3%99':'针织连衣裙',
    'https://s.taobao.com/search?spm=a230r.1.0.0.yg1UkO&q=%E5%8D%8A%E8%BA%AB%E8%A3%99+%E9%9B%AA%E7%BA%BA':'半身裙雪纺',
    'https://s.taobao.com/search?spm=a230r.1.0.0.AZ9WmW&q=%E5%8D%8A%E8%BA%AB%E8%A3%99%E7%A7%8B%E5%86%AC':'半身裙秋冬',
    'https://s.taobao.com/search?spm=a230r.1.0.0.uMwWcB&q=%E8%BF%9E%E8%A1%A3%E8%A3%99%E5%A4%8F':'连衣裙夏',
    'https://s.taobao.com/search?spm=a230r.1.0.0.zN7Oz8&q=%E8%BF%9E%E8%A1%A3%E8%A3%99%E5%A4%8F%E5%B0%8F%E6%B8%85%E6%96%B0':'连衣裙夏小清新',
    'https://s.taobao.com/search?spm=a230r.1.0.0.nTcqpQ&q=%E8%BF%9E%E8%A1%A3%E8%A3%99%E7%A7%8B':'连衣裙秋',
    'https://s.taobao.com/search?spm=a230r.1.0.0.vaMtq1&q=%E8%BF%9E%E8%A1%A3%E8%A3%99%E7%A7%8B%E5%86%AC':'连衣裙秋冬',
    'https://s.taobao.com/search?spm=a230r.1.0.0.prsQXH&q=%E7%A7%8B%E5%86%AC%E8%BF%9E%E8%A1%A3%E8%A3%992016%E6%96%B0%E6%AC%BE':'秋冬连衣裙2016新款',
    'https://s.taobao.com/search?spm=a230r.1.0.0.6OOB8Q&q=%E9%95%BF%E8%A2%96%E8%BF%9E%E8%A1%A3%E8%A3%99%E6%98%A5%E7%A7%8B':'长袖连衣裙春秋',
    'https://s.taobao.com/search?spm=a230r.1.0.0.VnwiMC&q=%E8%BF%9E%E8%A1%A3%E8%A3%99%E5%A4%8F%E5%B0%8F%E6%B8%85%E6%96%B0%E9%9F%A9%E5%9B%BD':'连衣裙夏小清新韩国',
}
```

### B2，__init__.py

```
# coding=utf-8
# usr/bin/eny python


__author__ = 'HunterChao'


__all__ = ['Taobao']


from dict import urldict
from multiprocessing import Pool
from api import Taobao


if __name__ == '__main__':
    dics = list(urldict.values())
    pool = Pool(processes=4)
    for dic in dics:
        print(dic)
        tao = Taobao(dic)
        pool.apply_async(tao.run())
    pool.close()
    pool.join()

    print('-----*------ 结束 -----*------')
```

### B3,api.py

```
# coding=utf-8
# usr/bin/eny python


import requests
import re
import json
import time
import urllib.request
import pymysql.cursors


class Taobao(object):
    '    SQLALCHEMY_DATABASE_URI = 'mysql://root:12345678@localhost:3306/test'
    def __init__(self,theme):
        self.theme = urllib.request.quote(theme)    # 连衣裙类型
        self.conn = pymysql.connect(host = 'localhost',port = 3306,user = 'root',
                                    passwd = '12345678',db = 'test',charset = 'utf8')
        self.cursor = self.conn.cursor()
        self.info_sql = "INSERT IGNORE INTO `address` VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"



    def sovl_dict(self,contents):
```

```
'''
解析字典
'''
for content in contents:
    print(content)

    if 'itemId' in content:
        goods_id = content['itemId']
    elif 'nid' in content:
        goods_id = content['nid']
    else:
        goods_id = u" "

    if 'sellerId' in content:
        shop_id = content['sellerId']
    elif 'user_id' in content:
        shop_id = content['user_id']
    else:
        shop_id = u" "

    if 'item_loc' in content:
        shop_loc = content['item_loc']
    else:
        shop_loc = u" "

    if 'nick' in content:
        shop_name = content['nick']
    elif 'shopName' in content:
        shop_name = content['shopName']
        print(shop_name)
    else:
        shop_name = u" "

    if 'title' in content:
        goods_title = content['title']
    elif 'raw_title' in content:
        goods_title = content['raw_title']
    else:
        goods_title = u" "

    if 'recommendReason' in content:
        view_sales = content['recommendReason']
    elif 'view_sales' in content:
        view_sales = content['view_sales']
    else:
        view_sales = u" "

    if 'salePrice' in content:
        view_price = content['salePrice']
    elif 'view_price' in content:
        view_price = content['view_price']
    else:
        view_price = u" "

    if 'url' in content:
        comment_url = 'https:'+content['url']
    elif 'comment_url' in content:
        comment_url = content['comment_url']
        #print(comment_url)
    else:
        comment_url = u" "
    #comment_url = 'https:' + comment_url

    if 'pic' in content:
        pic_url = 'https:'+content['pic']
    elif 'pic_url' in content:
```

```
            pic_url = 'https:'+content['pic_url']
        elif 'picUrl' in content:
            pic_url = 'https:' + content['picUrl']
        else:
            pic_url = u" "


        count_list = re.findall('(\w*[0-9]+)\w*',view_sales)
        count = count_list[0]
        #print(type(count))

        #print(pic_url)
        #print("{0}::{1}::{2}::{3}::{4}::{5}::{6}".format(goods_id, shop_id, goods_title, \
                                                         #view_sales, view_price, comment_url,pic_url))
        self.cursor.execute(self.info_sql, (str(goods_id), str(shop_id), shop_loc, shop_name, goods_title, \
                                            str(view_sales), str(view_price), comment_url,pic_url,count))
        self.conn.commit()
        return goods_id,shop_id



def first_content(self,first_url):
    '''
    首页商品信息页
    '''
    s = requests.get(first_url)
    contents = s.content.decode('utf-8')
    regex = 'g_page_config = (.+)'
    items = re.findall(regex, contents)
    items = items.pop().strip()
    items = items[0:-1]
    items = json.loads(items)
    items = items['mods']['itemlist']['data']['auctions']
    if items == []:
        return
    else:
        goods_id,shop_id = self.sovl_dict(items)
        self.second_content(goods_id, shop_id)    # 爬取二级页面
        # time.sleep(1)



def second_content(self,goods_id,shop_id):
    '''
    二级页面的商品信息
    '''
    print('——*———— 二级页面 ————*—— ')
    second_url = 'https://tui.taobao.com/recommend?itemid={0}&sellerid={1}&_ksTS=&callback=jsonp&appid=3066'\
                                                                                    .format(goods_id,shop_id)
    #print(second_url)
    s = requests.get(second_url)
    contents = s.content.decode('gbk')
    regex = 'jsonp(.+)'
    items = re.findall(regex, contents)
    items = items.pop()
    items = items[1:-2]
    items = json.loads(items)
    items = items['result']
    if items == []:
        return
    else:
        self.sovl_dict(items)



def run(self):
    '''
    运行
    '''
    try:
        for i in range(100):
            first_url = 'https://s.taobao.com/search?spm=&q={0}&bcoffset=&ntoffset&p4ppushleft=1%2C48&s={1}'\
                                                                                    .format(self.theme, (i+1)*44)
            self.first_content(first_url)
    except Exception as e:
```

```
            print(e)
```

## 3.8，女装展示(按销量排行，git地址：https://github.com/jing1900/p1-address.git)

### Address.html

```
{%extends 'base.html'%}
{%block title%}商品页{%endblock%}
{%block css%}/static/styles/pages/index.css {%endblock%}

{%block content%}
<div class="page-main ">
    <div class="list clearfix js-image-list">
        {%for address in address%}



        <article class="mod clearfix">
                <header class="mod-hd clearfix">
                    <time class="time">{{address.view_price}}￥</time>


                    <div class="profile-info">
                        <a title="{{address.shop_name}}" href="">{{address.shop_name}}</a>
                    </div>
                </header>
                <div class="mod-bd clearfix">
                    <div class="img-box">
                        <a href="{{address.comment_url}}">
                            <img src="{{address.pic_url}}">
                        </a>
                    </div>
                </div>
                <div class="mod-ft clearfix">
                    <ul class="discuss-list js-discuss-list" id="ul">


                        <li class="more-discuss">
                            <a href="{{address.comment_url}}">
                                <span>{{address.goods_title}}</span></a><br>
                            <span>{{address.view_sales}}</span>
                        </li>


                    </ul>


                </div>
            </article>
        {%endfor%}
    </div>
{% if has_next %}
    <div class="more-content js-load-more">
            <a class="_oidfu" href="javascript:void(0);">更多</a>
    </div>
{% endif %}
</div>



{%endblock%}
{% block js %}


<script type="text/javascript" src="/static/js/jquery.js"></script>
<script type="text/javascript" src="/static/js/address_profile.js"></script>
{% endblock%}
```

### View.py

```
#/address/页
@app.route('/address/')
def address():
    #paginate = Address.query.filter(Address.pic_url != "").order_by(abs(Address.sale_counts)).paginate(page=1,per_page=10,error_out=False)
    paginate = Address.query.filter(Address.pic_url != "").order_by(db.desc(Address.sale_counts)).paginate(page=1,per_page=10,error_out=False)
    #images = Image.query.order_by('id desc').limit(10).all()
    return render_template('address.html',has_next = paginate.has_next,address = paginate.items)

 #ajax异步加载address
@app.route('/address/items/<int:page>/<int:per_page>/')
def address_items(page, per_page):
```

```
    #逆序读取分页
    paginate = Address.query.filter(Address.pic_url != "").order_by(db.desc(Address.sale_counts)).paginate(page=page, per_page=per_page, error_out=False)
    #返回map
    map = {'has_next': paginate.has_next}
    address = []
    #遍历分页中的图片
    for a in paginate.items:
        addressvo = {
            'view_price': a.view_price,
            'shop_name': a.shop_name,
            'comment_url': a.comment_url,
            'pic_url': a.pic_url,
            'goods_title': a.goods_title,
            'view_sales': a.view_sales
        }
        address.append(addressvo)

    map['address'] = address
    return json.dumps(map)
```

## Address_profile.js

```
$(function () {
    var oExports = {
        initialize: fInitialize,
        // 渲染更多数据
        renderMore: fRenderMore,
        // 请求数据
        requestData: fRequestData,
        // 简单的模板替换
        tpl: fTpl
    };
    // 初始化页面脚本
    oExports.initialize();

    function fInitialize() {
        var that = this;
        // 常用元素
        that.listEl = $('div.js-image-list');
        // 初始化数据
        that.page = 1;
        that.pageSize = 10;
        that.listHasNext = true;
        // 绑定事件
        $('.js-load-more').on('click', function (oEvent) {
            //alert('执行onclick')
            var oEl = $(oEvent.currentTarget);
            var sAttName = 'data-load';
            // 正在请求数据中，忽略点击事件
            if (oEl.attr(sAttName) === '1') {
                return;
            }
            // 增加标记，避免请求过程中的频繁点击
            oEl.attr(sAttName, '1');
            that.renderMore(function () {
                // 取消点击标记位，可以进行下一次加载
                oEl.removeAttr(sAttName);
                // 没有数据隐藏加载更多按钮
                !that.listHasNext && oEl.hide();
            });
        });
    }

    function fRenderMore(fCb) {
        //alert('执行rendermore')
        var that = this;
        // 没有更多数据，不处理
        if (!that.listHasNext) {
            return;
        }
        that.requestData({
            page: that.page + 1,
            pageSize: that.pageSize,
            call: function (oResult) {
                //alert(oResult)
                // 是否有更多数据
                that.listHasNext = !!oResult.has_next && (oResult.address || []).length > 0;
                // 更新当前页面
                that.page++;
```

```
                // 渲染数据
                var sHtml = '';
                $.each(oResult.address, function (nIndex, oImage) {
                    sHtml_1 = that.tpl([
                        '<article class="mod">',
                            '<header class="mod-hd">',
                                '<time class="time">#{view_price}￥</time>',
                                '<div class="profile-info">',
                                    '<a title="#{shop_name}" href="">#{shop_name}</a>',
                                '</div>',
                            '</header>',
                            '<div class="mod-bd">',
                                '<div class="img-box">',
                                    '<a href="#{comment_url}">',
                                        '<img src="#{pic_url}">',
                                    '</a>',
                                '</div>',
                            '</div>',
                            '<div class="mod-ft clearfix">',
                                '<ul class="discuss-list js-discuss-list" id="ul">',
                                    '<li  class="more-discuss">',
                                        '<a href="#{comment_url}">',
                                            '<span>#{goods_title}</span></a>',
                                            '<br>',
                                            '<span>#{view_sales}</span>',
                                    '</li>',
                                '</ul>',
                            '</div>',
                        '</article>'].join(''), oImage);
                    sHtml += sHtml_1;
                });
                //alert(sHtml)
                sHtml && that.listEl.append(sHtml);
            },
            error: function () {
                alert('出现错误，请稍后重试');
            },
            always: fCb
        });
    }

    function fRequestData(oConf) {
        //alert('执行frequest')
        var that = this;
        var sUrl = '/address/items/' + oConf.page + '/' + oConf.pageSize + '/';
        //alert(sUrl)
        $.ajax({url: sUrl, dataType: 'json'}).done(oConf.call).fail(oConf.error).always(oConf.always);

    }

    function fTpl(sTpl, oData) {
        var that = this;
        sTpl = $.trim(sTpl);
        return sTpl.replace(/#{(.*?)}/g, function (sStr, sName) {
            return oData[sName] === undefined || oData[sName] === null ? '' : oData[sName];
        });
    }
});
```

## 3.9，单元测试+服务器部署（git：https://github.com/jing1900/p1_unittest.git)

### a，单元测试

#### A1：这里测试一般直接写在工程项目的外面，即项目的根目录，以test开头。

Test.py

```
import unittest
#导入工程
from stagram import app


'''单元测试类,继承unittest的TestCase
一旦run这个测试用例，就会自动把里面以test开头的方法作为测试用例跑
其执行顺序是，setup-test1-teardown,setup-test2-teardown

因此：
1，初始化数据，这个过程在setup里执行
2，执行测试业务，自行写执行函数
3，验证测试数据（assert，断言），这个过程在test函数里
3,清理数据，这个过程放在teardown里'''
```

```
'''这里我们对首页做个测试'''
class stagramTest(unittest.TestCase):

    #每次跑单元测试时，它都会跑
    def setUp(self):
        print('setup')
        #测试模式
        app.config['TESTING'] = True
        #把app保存下来
        self.app = app.test_client()



    def tearDown(self):
        print('tearDown')


    def register(self,username,password):
        return self.app.post('/reg/',data={'username':username,'password':password},follow_redirects=True)


    def login(self,username,password):
        return self.app.post('/login/',data={"username": username, "password": password},follow_redirects=True)


    def logout(self):
        return self.app.get('/logout/')

    #测试注册，登录，测试前，先把注册登录都写好，在上面
    def test_reg_login_logout(self):
        #测试注册这里执行成功，会返回一个http response,故可以用状态码来验证
        assert self.register('jing1','1').status_code ==200
        #注册成功后，判断用户名是否在首页的标题上
        assert bytes('-jing1',encoding="utf8") in self.app.open('/').data
        #登出
        self.logout()
        #再次测试，这里应该不在
        assert bytes('-jing1',encoding="utf8") not in self.app.open('/').data
        #登入
        self.login("jing1", "1")
        #再判断
        assert bytes('-jing1',encoding="utf8") in self.app.open('/').data



    #测试profile
    def test_profile(self):
        #这里由于没登录，所以一定会跳转到登录注册界面，因此会response
        r = self.app.open('/profile/3/',follow_redirects=True)
        #判断response状态码
        assert r.status_code == 200
        #查看页面元素里有没有password这个关键词
        assert bytes('password',encoding='utf8') in r.data
        #注册后，再进行判断
        assert self.register('jing2','2')
        #再打开一个用户,判断用户名是否在这个页面，也就是不进行跳转到登录注册页面了
        assert bytes('jing2',encoding='utf8') in self.app.open('/profile/1/', follow_redirects=True).data
```

## A2：在脚本里运行测试用例：（manage.py)

```
#-*- encoding=UTF-8 -*-
from stagram import app,db
from stagram.models import User
from stagram.models import Image,Comment
import random
import unittest
from sqlalchemy import or_,and_
'''脚本'''
#导入manager
from flask_script import Manager


manager = Manager(app)
@manager.command
def run_test():
    #每次跑之前，清空下数据库
    db.drop_all()
    db.create_all()
```

```
#让其自行从目录里找测试用例，该目录下以test开头的
tests = unittest.TestLoader().discover('./')
#跑这个测试用例
unittest.TextTestRunner().run(tests)
```

### b，部署服务器

B1，配置服务器环境：安装nginx，mysql-server。gunicorn等，还要装一些扩展包：

服务器：apt-get install nginx mysql-server gunicorn python-flask libmysqlclient-dev python-dev

依赖包：pip install Flask-Script Flask-SQLAlchemy Flask-Login qiniu Flask-MySQLdb

B2，设置nginx分发代理，见下图（请求-经nginx分发到-不同的服务器-再执行服务器上的后端程序，这样以来，当一台机子出问题时，nginx可以立马把流量导到其他机子上，使服务更稳健）：

Nginx 配置 /etc/nginx/sites-enabled/c1

```
server {
    listen 80;
    server_name c1.nowcoder.com;
    location / {
        proxy_pass http://127.0.0.1:8000;
    }
}
```

B3，启动服务器（这行命令中-w2，是指有两个线程，-d指在后台运行这个程序，-b，要绑定的端口和地址。然后默认运行在127.0.0.1:8080这个端口，项目文件名为nowstagram）：

gunicorn –D –w 2*core+1 –b 127.0.0.1:8000 nowstagram:app

## 4, 面试时可以讲什么：

先大体介绍实现的功能：
登录注册，邮箱激活验证，图片上传和云存贮，爬虫，页面展示，评论和评论的实时加载，页面异步刷新。
然后跟着功能逐个深入技术：

1，首先数据库，底层采用的是mysql，由于项目是python写的，所有我们用一个flask-sqlalchemy的框架来读取数据库。这个框架实际是一个orm的思想，它把数据库中的数据和开发过程中的对象关联起来，可以简化开发的过程。
通过这个以后，我理解了orm的思想，在开发流程确定之后，不需要写一些重复的代码，代码的model和数据库的model是一一匹配的。通过这种orm映射，可以将数据库里的对象和代码里的对象一一关联起来。这样可以大幅提高开发的效率。此外，这个sqlalchemy里面还有一对多，多对多和lazy加载，分别是怎么做的

2，mvc设计模式：
前后端分离，模板。

model放在一起，view放在一起，html放在一起

视图的渲染，数据和页面是分离的。通过render_template这个模式，将两者联系起来，然后这个变量是如何传递的，然后这个页面又是如何渲染的。然后又怎么可以通过ajax优化

此外，很多页面，为了增加其复用性，页面都是通过嵌套和继承来写的。

3，单元测试
开发规范，大致流程。

4，云sdk：

快速学习的能力，服务刚好匹配。接入的时候是怎么学的，官方代码，功能，快速切入。

最好了解下七牛是具体怎么做的。内部大体是怎么实现的

5，ajax-设计的思路：为了体验更好，json数据格式定义。ajax返回json串，js先解析，再展现

6，flask框架好在哪里，框架结构，把网页开发中核心的几个点，都包装成独立的模块

然后用路径映射的方式，跟后端进行数据传递

7，git，熟悉这个工具，然后备份代码