表单

一、原生表单

- 准备模板文件`login.html`：

```
<form method="post" action="/check/">
    用户名：<input name="username" /><br />
    <input type="submit" />
</form>
```

- 添加视图函数，并渲染模板文件：

```
@app.route('/login/')
def login():
    return render_template('login.html')
```

- 添加检验视图函数：

```
@app.route('/check/', methods=['POST'])
def check():
    return '登录成功'
```

- 一个路由接收两种请求：

```
@app.route('/login/', methods=['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template('login.html')
    else:
        # request.form存放了所有的POST请求数据
        # return request.form.get('username')
        # request.values存放了所有GET/POST请求数据
        return request.values.get('username')
```

二、flask-wtf

- 说明：表单处理的扩展库，提供了CSRF、字段校验等功能，使用非常方便

- 安装：`pip install flask-wtf`
- 使用：
  - 创建注册表单类

```
from flask_wtf import FlaskForm #导入表单基类
from wtforms import StringField,PasswordField,SubmitField
from wtforms.validators import DataRequired,Length,EqualTo,Email

app = Flask(__name__)
app.config['SECRET_KEY'] = 'SECRET_KEY'
maanger = Manager(app)

#自定义表单注册类
class Register(FlaskForm):
    #username为当前标签的name值 用户名为到那个前标签展示的左侧的label标签 点击用户名 出发当前标签选中节点
    username = StringField('用户名',validators=[DataRequired('请输入用户名'),Length(min=6,max=12,message='用户名长度范围在6~12位之间')])
    userpass = PasswordField('密码',validators=[DataRequired('请输入密码'),Length(min=6,max=12,message='密码长度范围在6~12位之间')])
    confirm = PasswordField('确认密码',validators=[DataRequired('请输入确认密码'),EqualTo('userpass',message='密码和确认密码不一致')])
    email = StringField('邮箱',validators=[DataRequired('请输入邮箱地址'),Email(message='请输入正确的邮箱')])
    submit = SubmitField('注册')
```

  - 添加视图函数，创建表单对象，并渲染模板文件：

```
#注册
@app.route('/register/',methods=['GET','POST'])
def register():
    form = Register() #实例化表单类
    #这个方法是实现表单校验功能的 csrf, 数据正确性 都通过了 则为真 否则为假
    if form.validate_on_submit():
        # print(request.form)
        print(form.username) #拿到username的整个标签
        print(form.username.data) #取出username里面的value值
        return '数据提交过来了'
    return render_template('register.html',form=form)
```

- 原生渲染表单

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>flask-wtf的表单类</h2>
<form action="{{ url_for('register') }}" method="post">
    {# csrf #}
    {{ form.csrf_token }}
    {{ form.username.label }}
    {{ form.username(style="color:red;",class='myself',placeholder='请输入用户名...') }} #给当前的标签添加属性和值 关键字参数
    #循环迭代取出验证失败的错误信息（也就是你在验证器里的属性message的值）
    {% for error in form.username.errors %}
        <span style="color:red;">{{ error }}</span>
    {% endfor %}
    <br>
    {{ form.userpass.label }}
    {{ form.userpass }}
    {% for error in form.userpass.errors %}
        <span style="color:red;">{{ error }}</span>
    {% endfor %}
    <br>
```

```
    {{ form.confirm.label }}
    {{ form.confirm() }}
    {% for error in form.confirm.errors %}
        <span style="color:red;">{{ error }}</span>
    {% endfor %}
    <br>
    {{ form.email.label }}
    {{ form.email() }}
    {% for error in form.email.errors %}
        <span style="color:red;">{{ error }}</span>
    {% endfor %}
    <br>
    {{ form.submit() }}
</form>
</body>
</html>
```

- 常见字段类型

字段类型	说明
StringField	普通文本字段
Submit	提交按钮
PasswordField	密文字段
HiddenField	隐藏字段
RadioField	单选框
BooleanField	复选框
FileField	文件上传
SelectField	下拉框
TextAreaField	文本域
IntegerField	文本字段，值为整数
FloatField	文本字段，值为浮点数
DateField	datetime.date类型
DateTimeField	datetime.datetime类型

- 常见验证器

验证器	说明
Length	规定字符长度
DataRequired	确保字段有值(提示信息与所写的不一致)
Email	邮箱地址
IPAddress	IP地址
NumberRange	数值的范围
URL	统一资源定位符
EqualTo	验证两个字段的一致性
Regexp	正则验证

- 自定义验证函数

```
from wtforms.validators import ValidationError

class NameForm(FlaskForm):
    。。。
    def validate_name(self, field):
        if len(field.data) < 6:
            raise ValidationError('用户名不能少于6个字符')
```

> 总结：写字段验证函数，就是写一个'validate_字段名'的函数

三、flask-moment

- 说明：专门负责时间本地化显示的扩展库，使用非常方便
- 安装：`pip install flask-moment`
- 官网：https://momentjs.com/docs/#/displaying/format/
- 使用：
  - python代码

```
from flask_moment import Moment

moment = Moment(app)

@app.route('/mom/')
def mom():
    from datetime import datetime, timedelta
    current_time = datetime.utcnow() + timedelta(seconds=-60)
    return render_template('mom.html', current_time=current_time)
```

  - 模板文件

```
{% block scripts %}
{{ moment.include_jquery() }}
{{ moment.include_moment() }}
{% endblock %}
{# 加载jQuery #}


{# 设置语言 #}
```

```
{{ moment.locale('zh-CN') }}

{# 简单的格式化时间显示 #}
<div>时间：{{ moment(current_time).format('LLLL') }}</div>
<div>时间：{{ moment(current_time).format('LLL') }}</div>
<div>时间：{{ moment(current_time).format('LL') }}</div>
<div>时间：{{ moment(current_time).format('L') }}</div>

{# 自定义格式化显示 #}
<div>自定义：{{ moment(current_time).format('YYYY-MM-DD HH:mm:ss')
}}</div>

{# 显示时间差 #}
<div>发表于：{{ moment(current_time).fromNow() }}</div>
```

四、文件上传

原生实现

- 模板文件

```
<form method="post" enctype="multipart/form-data">
    <input type="file" name="photo" /><br />
    <input type="submit" value="上传" />
</form>
```

- 视图函数

```
import os

# 配置上传文件保存目录
app.config['UPLOADED_FOLDER'] = os.path.join(os.getcwd(),
'static/upload')

@app.route('/upload/', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # 获取上传对象
        photo = request.files.get('photo')
        if photo:
            # 拼接保存路径名
```

```
            pathname =
os.path.join(app.config['UPLOADED_FOLDER'], photo.filename)
            # 保存上传文件
            photo.save(pathname)
            return '上传成功'
        else:
            return '上传失败'
    return render_template('upload.html')
```

flask-uploads

- 说明：极大的的简化了文件上传相关的操作，使用非常方面。
- 安装：`pip install flask-uploads`
- 使用：
  - 配置

```
from flask_uploads import UploadSet, IMAGES
from flask_uploads import configure_uploads
from flask_uploads import patch_request_class
import os

app.config['UPLOADED_PHOTOS_DEST'] = os.getcwd()
app.config['MAX_CONTENT_LENGTH'] = 8 * 1024 * 1024
# 创建上传对象
photos = UploadSet('photos', IMAGES)
# 配置上传对象
configure_uploads(app, photos)
# 配置上传文件大小，默认为64M,
# 若设置为None，则以MAX_CONTENT_LENGTH配置为准
patch_request_class(app, size=None)
```

  - 视图函数

```
@app.route('/upload/', methods=['GET', 'POST'])
def upload():
    img_url = None
    if request.method == 'POST':
        # 获取上传对象
        photo = request.files.get('photo')
        if photo:
            # 保存上传文件，返回文件名
            filename = photos.save(photo)
            # 根据文件名获取上传文件的URL
            img_url = photos.url(filename)
    return render_template('upload.html', img_url=img_url)
```

综合使用

- 要求：结合flask-bootstrap、flask-wtf、flask-uploads等完成文件上传
- 使用：
  - 配置

```
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed, FileRequired
from wtforms import SubmitField
from flask_uploads import UploadSet, IMAGES
from flask_uploads import configure_uploads
from flask_uploads import patch_request_class
from flask_bootstrap import Bootstrap
import os

bootstrap = Bootstrap(app)

app.config['SECRET_KEY'] = '123456'
app.config['MAX_CONTENT_LENGTH'] = 8 * 1024 * 1024
app.config['UPLOADED_PHOTOS_DEST'] = os.path.join(os.getcwd(),
'static/upload')

photos = UploadSet('photos', IMAGES)
configure_uploads(app, photos)
patch_request_class(app, size=None)

class UploadForm(FlaskForm):
```

```
    photo = FileField('头像', validators=[FileRequired(message='请
选择文件'),
                                          FileAllowed(photos,
message='只能上传图片文件')])
    submit = SubmitField('上传')
```

- 视图函数

```
@app.route('/upload/', methods=['GET', 'POST'])
def upload():
    img_url = None
    form = UploadForm()
    if form.validate_on_submit():
        photo = form.photo.data
        filename = photos.save(photo)
        img_url = photos.url(filename)
    return render_template('upload.html', form=form,
img_url=img_url)
```

- 模板文件

```
{% extends 'bootstrap/base.html' %}

{% from 'bootstrap/wtf.html' import quick_form %}

{% block title %}完整的文件上传{% endblock %}

{% block content %}
    <div class="container">
        {% if img_url %}
            <img src="{{ img_url }}">
        {% endif %}
        {{ quick_form(form) }}
    </div>
{% endblock %}
```

- 生成随机文件名

```
def random_string(length=32):
    import random
    base_str = 'abcdefghijklmnopqrstuvwxyz1234567890'
```

```
    return ''.join(random.choice(base_str) for i in
range(length))

@app.route('/upload/', methods=['GET', 'POST'])
def upload():
    。。。
        # 提取文件后缀
        suffix = os.path.splitext(photo.filename)[1]
        # 生成随机文件名
        filename = random_string() + suffix
        # 保存文件
        photos.save(photo, name=filename)
    。。。
```

- 生成缩略图：PIL模块(只支持py2，要支持py3需要安装pillow)

```
from PIL import Image

@app.route('/upload/', methods=['GET', 'POST'])
def upload():
    ...
        # 拼接完整文件路径名
        pathname =
os.path.join(app.config['UPLOADED_PHOTOS_DEST'], filename)
        # 打开文件
        img = Image.open(pathname)
        # 设置大小
        img.thumbnail((64, 64))
        # 保存图片
        img.save(pathname)
    ...
```

五.富文本编辑器

UEditor是由百度开发的所见即所得富文本web编辑器，具有轻量，可定制，注重用户体验等特点，开源基于MIT协议，允许自由使用和修改代码。帮助文档:http://fex-team.github.io/ueditor/

- 下载(https://ueditor.baidu.com/website/download.html#ueditor)

  解压后,把ueditor放到static目录下

```
| static/
| | ueditor/
| | |+dialogs/
| | |+lang/
| | |+php/
| | |+themes/
| | |+third-party/
| | |-config.json
| | |-index.html
| | |-ueditor.all.js
| | |-ueditor.all.min.js
| | |-ueditor.config.js
| | |-ueditor.parse.js
| | |-ueditor.parse.min.js
```

- 编辑模板文件

  在head标签加入下面几行：

```
<script type="text/javascript" charset="utf-8" src="{{
url_for('static', filename='ueditor/ueditor.config.js') }}">
</script>
<script type="text/javascript" charset="utf-8" src="{{
url_for('static', filename='ueditor/ueditor.all.min.js') }}">
</script>
<!--建议手动加在语言，避免在ie下有时因为加载语言失败导致编辑器加载失败-
->
<!--这里加载的语言文件会覆盖你在配置项目里添加的语言类型，比如你在配置
项目里配置的是英文，这里加载的中文，那最后就是中文-->
<script type="text/javascript" charset="utf-8" src="{{
url_for('static', filename='ueditor/lang/zh-cn/zh-cn.js') }}">
</script>
```

  在body中加入:

```
<script id="editor" type="text/plain"></script>

<script type="text/javascript">
    //实例化编辑器
    //建议使用工厂方法getEditor创建和引用编辑器实例，如果在某个闭包下引用该编辑器，直接调用UE.getEditor('editor')就能拿到相关的实例
    var ue = UE.getEditor('editor', {
        serverUrl: "/upload/"
    });
</script>
```

六、图形验证码

6.1 安装Pillow库

PIL：Python Imaging Library，已经是Python平台事实上的图像处理标准库了。PIL功能非常强大，但API却非常简单易用。

由于PIL仅支持到Python 2.7，加上年久失修，于是一群志愿者在PIL的基础上创建了兼容的版本，名字叫Pillow，支持最新Python 3.x，又加入了许多新特性，因此，我们可以直接安装使用Pillow。

```
$ pip install pillow
```

6.2 创建验证码步骤

1)、创建画布 2)、生成验证码字符串 3)、画验证码 4)、画干扰点 5)、画干扰线 6)、返回验证码图片

6.3 常用方法

方法名	说明
Image.new()	创建图像
ImageDraw.Draw	创建画笔
ImageDraw.point	画点
ImageDraw.line	画线
ImageDraw.text	画文字
ImageFont.truetype	获取字体

6.4 实现

```
from io import BytesIO
from random import randint,sample
import string

from PIL import Image,ImageFont,ImageDraw

class VerifyCode:
    def __init__(self,width=100, height=40,size=4):
        self.width = width
        self.height = height
        self.size = size    #验证码长度
        self.__code = None  #保存验证码字符串
        self.pen = None     #画笔
    @property
    def code(self):  #获取验证码字符串的方法
        return self.__code

    def output(self):
        # 1 image pen
        im = Image.new('RGB', (self.width,
self.height),self.__rand_color(160,255))
        self.pen = ImageDraw.Draw(im)

        # 2code string
        self.__code = self.rand_string()

        # 3 draw string
```

```
        self.__draw_string()
        # 4 point
        self.__disturb_point()
        # 5 line
        self.__draw_line()
        # 6 return图片的二级制
        # im.save('yzm.png','PNG')  #保存图片
        buf = BytesIO()
        im.save(buf, 'PNG')
        res = buf.getvalue()
        buf.close()
        return res

    def __draw_line(self):
        for i in range(5):
            start_point = (randint(1,self.width -
1),randint(1,self.height - 1))
            end_point = (randint(1,self.width -
1),randint(1,self.height - 1))

self.pen.line([start_point,end_point],fill=self.__rand_color(50,100))
    def __disturb_point(self):
        for i in range(300):
            x = randint(1,self.width - 1)
            y = randint(1,self.height - 1)
            self.pen.point([(x,y)],fill=self.__rand_color(60,120))

    def __draw_string(self):
        font1 = ImageFont.truetype('SIMLI.TTF',size=20,encoding='utf-
8')
        width = (self.width - 20) / self.size
        for i in range(len(self.__code)):
            x = 13 + i * width
            y = randint(5,20)
            self.pen.text((x,
y),self.__code[i],fill='black',font=font1)

    def rand_string(self):
        # 数字验证码
        return str(randint(1000,pow(10,self.size)) - 1)

    def __rand_color(self,low,high):
```

```
            return randint(low,high),randint(low,high),randint(low,high)
```

在flask中使用：

```
from VerifyCode import VerifyCode
def yzm():
    vc = VerifyCode()
    result = vc.output()
    session['code'] = image.code
    return HttpResponse(result,'image/png')
```