# Flask

**文档**：<https://dormousehole.readthedocs.io/en/latest/>

扩展：https://pypi.org/search/?q=login&o=

框架：MTV/MVT

model 模型--->数据库

templates 模板--->页面

view 视图+路由route--->控制作用

虚拟环境(windown)：pip install virtualenvwrapper-win -i https://mirrors.aliyun.com/pypi/simple/

查看：workon 新建：mkvirtualenv 名称

进入：workon 名称 退出：deactivate

删除：rmvirtualenv 名称

# **Flask项目目录结构**

| - flask //程序包

| - static //静态文件夹(css,js,图片)

| - templates //模板（网页）

| - app.py

# **WSGI协议**

pass

from flask import Flask

app=Flask(\_\_name\_\_)

@app.route('/')  
def index():  
 return 'hello'  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run()

# 配置

配置路由：@app.route('/')

设置settings.py配置文件

ENV='development'  
DEBUG= True

导入配置文件

import settings

app.config.from\_object(settings) #传入对象方式 要import  
app.config.from\_pyfile('settings.py') #传入文件

# 路由

使用 **[route()](https://dormousehole.readthedocs.io/en/latest/api.html" \l "flask.Flask.route" \o "flask.Flask.route)** 装饰器来把函数绑定到 URL:

@app.route('/')  
def index():  
 return 'Index Page'

app.add\_url\_rule('/',view\_func=index)

@app.route('/',endpoint='')

# 变量规则

通过把 URL 的一部分标记为 <variable\_name> 就可以在 URL 中添加变量。标记的 部分会作为关键字参数传递给函数。通过使用 <converter:variable\_name> ，可以 选择性的加上一个转换器，为变量指定规则。

@app.route('/user/<username>')

def show\_user\_profile(username):

@app.route('/post/<int:post\_id>')

def show\_post(post\_id):

转换器类型：

|  |  |
| --- | --- |
| string | （缺省值） 接受任何不包含斜杠的文本 |
| int | 接受正整数 |
| float | 接受正浮点数 |
| path | 类似 string ，但可以包含斜杠 |
| uuid | 接受 UUID 字符串 |

# 响应上下文

## current\_app

## g

## 请求报文常用参数

method：请求的类型（GET/POST）

form：POST请求的数据dict

args：GET请求的数据dict

values：POST和GET数据的集合dict

files：上传的文件数据dict

cookies：请求的cookies

from flask import Flask,current\_app,request,g  
@app.route('/')  
def hello\_world():  
 args=request.args #获取get数据 http：0.0.0.0/?name=1&age=1

header=request.headers #请求头

ip=request.remote\_addr #远程IP地址  
 return 'a'

# 请求对象

只要有请求就会产生一个resquest对象

from flask import Flask,request

@app.route('/login', methods=['POST', 'GET'])  
def login():  
 if request.method == 'GET':  
 return render\_template()  
 else:  
 ...

# request.args['username'] #get请求  
# request.form['username'] #post请求

# request.remote\_addr #远程地址

# 响应

视图函数的返回值会自动转换为一个响应对象。如果返回值是一个字符串，那么会被 转换为一个包含作为响应体的字符串、一个 200 OK 出错代码 和一个 text/html 类型的响应对象。如果返回值是一个字典，那么会调用 **jsonify()** 来产生一个响应。

视图函数的返回值可以是：

1. string 将自动转成response对象
2. make\_response() 自己构建response对象，可以添加头部信息
3. jsonify() 转成json格式
4. render\_template()
5. redirect('/') 重定向 填路由
6. abort(404) 返回默认好的404错误信息，或搭配****@app.errorhandler(404)钩子函数自定义错误返回****

resp = make\_response(render\_template('error.html'), 404) #自己构建response  
resp.headers['X-Something'] = 'A value' #设置响应头  
return resp

# 钩子函数

before\_first\_request：第一次请求前执行

before\_request：每次请求前执行

after\_request:每次请求后执行，发生错误不执行

teardown\_request::每次请求后执行，发生错误也执行

errorhander,：发生异常时, 自动调用对应的钩子函数

teardown\_appcontext 当APP上下文被移除之后执行的函数, 可以进行数据库的提交或者回滚

template\_filter,：增加模板过滤器

context\_processor ：上下文处理器, 返回的字典可以在全部模板中使用

# 模板

反向解析地址：{{ url\_for('static',filename='image/aa.jpg') }}

查看路由映射：app.url\_map

## 模板变量

代码中传入字符串,列表,字典到模板中

@app.route('/')  
def index():  
 my\_str = 'Hello Word'  
 return render\_template('temp\_demo1.html',my\_str=my\_str )

模板运算,取值

<br/> my\_int + 10 的和为：{{ my\_int + 10 }}

<br/> my\_int + my\_array第0个值的和为：{{ my\_int + my\_array[0] }}

<br/> my\_array 第0个值为：{{ my\_array[0] }}

<br/> my\_array 第1个值为：{{ my\_array.1 }}

<br/> my\_dict 中 name 的值为：{{ my\_dict['name'] }}

<br/> my\_dict 中 age 的值为：{{ my\_dict.age }}

用{%%}定义的控制代码块，可以实现一些语言层次的功能，比如循环语句

{% if user %}

{% else %}

<ul>

{% for index in indexs %} #循环中的变量loop.first,loop.last

<li> {{ index }} </li>

{% endfor %}

</ul>

## 过滤器

过滤器由竖线符号（|）与变量分开，并且括号中可以包含可选参数。可以链接多个过滤器。

{{ name|striptags|title }}

**字符串：**

**default** 默认值,如果该值未定义{{ name|default('no name') }}

**title** 返回标题的版本。即单词以大写字母开头，其余字符为小写字母。

**capitalize** 第一个字符大写，其他字符小写。

**upper** 换为大写 **lower** 换为小写 **reverse** 反转对象 **trim** 去除开头和结尾空格

**format** 格式字符串{{ "%s, %s!"|format(greeting, name) }}

**safe** 将值标记为安全，在启用了自动转义的环境中，此变量将不会被转义。

块内不会自动转义{% autoescape flase %}...{% endautoescape %}

**数值：**

**abs** 参数的绝对值。

**round** 四舍五入到给定的精度（默认为0）。round（precision = 0，method ='common' ）

'common' 向上或向下取整 'ceil' 总是四舍五入 'floor' 总是四舍五入

**列表：**

**length** 返回容器中长度**。**别名count

**first**返回序列的第一项。 **last**返回序列的最后一项。

**join** 拼接成字符串{{ [1, 2, 3]|join('|') }}

**sort**（reverse = False，case\_sensitive = False，attribute = None ）排序可迭代对象

reverse –降序排列。

case\_sensitive –排序时，分别对大小写进行排序。

attribute –排序时，作为排序依据的属性或键。{{ users|sort(attribute="age,name") }}

**sum**（attribute = None，start = 0 ）求和加上参数“ start”的值（默认为0）。

# 模板继承

基本模板：{% block head %}{% endblock %}

子模板：{% extends "base.html" %}

{% block head %}{{ super() }}{% endblock %}

# 消息闪现

用flash要设置secret\_key

app.secret\_key='123'

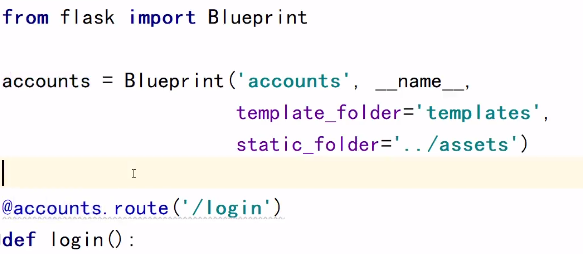
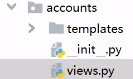
@app.route('/')  
def hello\_world():  
 flash('成功','success')  
 return render\_template('1.html')

# 控制结构清单

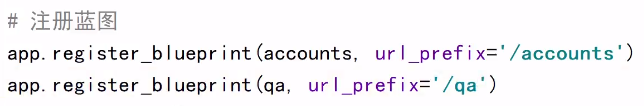
[https://jinja.palletsprojects.com/en/2.11.x/templates/#for](https://jinja.palletsprojects.com/en/2.11.x/templates/" \l "for)

# 蓝图

1,在包里配置蓝图，路由app.route变蓝图名



2,在app.py里注册蓝图，就可



| - flask

app.py

settings.py

| - templates

| - apps

| - views

\_\_init\_\_.py

| - \_\_init\_\_.py

| - blog.py

创建蓝图 \_\_init\_\_.py

from flask import Flask  
from settings import DevelopmentConfig  
from apps.views.blog\_view import blog\_bp  
def create\_app():  
 app = Flask(\_\_name\_\_,template\_folder='../templates')  
 #加载配置  
 app.config.from\_object(DevelopmentConfig)  
 # 注册南图  
 app.register\_blueprint(blog\_bp)  
 return app

引用蓝图blog\_view.py

from flask import Blueprint  
blog\_bp=Blueprint('blog',\_\_name\_\_,url\_prefix='/blog')  
@blog\_bp.route('/',endpoint='index')  
def blog\_index():  
 return 'blog'

app.py

from apps import create\_app  
app=create\_app()  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run()

settings.py

class Config:  
 DEBUG = False  
 TESTING = False  
 DATABASE\_URI = ''  
# 开发环境配置  
class DevelopmentConfig(Config):  
 ENV = 'development'  
 DEBUG = True  
# 生产环境配置  
class ProductionConfig(Config):  
 DATABASE\_URI = ''  
# 测试环境配置  
class TestingConfig(Config):  
 TESTING = True

index.html

<body>  
<a href="{{ url\_for('blog.index') }}">博客</a>  
</body>

# 数据库

安装：pip install -U flask-sqlalchemy

安装：pip install mysqlclient

# SQLAlchemy(Mysql数据库)

pip install pymysql pip install flask-sqlalchemy pip install flask-script

pip install flask-migrate

Flask: app=Flask(\_\_name\_\_)

app.run()

蓝图:blog\_bp = Blueprint('blog', \_\_name\_\_, url\_prefix='/blog')

加载蓝图： app.register\_blueprint(blog\_bp)

SQLAlchemy：db =SQLAlchemy() / db=SQLAlchemy(app)

加载到app：db.init\_app(app)

flask\_script，flask\_migrate：

from flask\_script import Manager  
from flask\_migrate import Migrate, MigrateCommand  
  
from apps.models.blog\_model import User  
from apps import create\_app  
from exts import db  
  
app = create\_app()  
manager = Manager(app)  
  
# 给manager添加db命令  
migrate = Migrate(app=app, db=db)  
manager.add\_command('db', MigrateCommand)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 manager.run()

## 数据库model

from datetime import datetime  
  
from exts import db  
  
  
class User(db.Model):

\_\_tablename\_\_ = 'user'  
 id = db.Column(db.Integer, primary\_key=True, autoincrement=True)  
 username = db.Column(db.String(12), nullable=False, unique=True)  
 password = db.Column(db.String(64), nullable=False)  
 phone = db.Column(db.String(11))  
 isdelete = db.Column(db.Boolean,default=False)  
 rdatetime = db.Column(db.DateTime, default=datetime.now)

迁移与同步：python manage.py db \_\_init\_\_

python manage.py db migrate

python manage.py db upgrade



# 文件

链接: https://pan.baidu.com/s/1Kr9OnlFuxBkr32Q2i7PEww 提取码: 289c

# 数据库URI：

安装：pip install flask-sqlalchemy

pip install pymysql

from flask\_sqlalchemy import SQLAlchemy

app.config['SQLALCHEMY\_DATABASE\_URI']='mysql://root:1234@127.0.0.1:3306/test\_flask'

db=SQLAlchemy(app)

URI参数格式：

Postgres:

postgresql://root:123456@127.0.0.1/test\_flask

MySQL:

mysql://root:123456@127.0.0.1/test\_flask

Oracle:

oracle://root:123456@127.0.0.1/test\_flask

SQLite:

sqlite:////absolute/path/to/foo.db

多个数据库支持

SQLALCHEMY\_BINDS = {

‘db1': 'mysqldb://localhost/user',

'db2': 'sqlite:////path/to/appmeta.db'

}

# 数据库模型：

class User(db.Model):  
 \_\_tablename\_\_='weibo\_user'  
 id=db.Column(db.Integer,primary\_key=True)

手动创建数据库（在数据库中手动创建了test\_flask逻辑库）（cmd）

创建表

db.create\_all(bind='db1')（支持多个数据库，但是只连接了一个数据库可以不加bind='db1'）

删除表

db.drop\_all()

数据库模型设计

ORM模型字段类型支持

一对多关系，外键关联

user\_id = db.Column(db.Integer,db.ForeignKey('user.id'), nullable=False)

反向引用

user = db.relationship('User', backref=db.backref('address', lazy=True))

# 新增/修改数据

1、构造ORM模型对象（cmd）

user = User('admin', 'admin@example.com')

2、添加到db.session（备注：可添加多个对象）

db.session.add(user)

3、提交到数据库

db.session.commit()

修改字段的操作

>>> user.password = '123321'

>>> db.session.add(user)

>>>db.session.delete(user)

>>> db.session.commit()

# ORM查询

From flasker import db,User (cmd)

返回结果集（list）

查询所有数据：User.query.all()

按条件查询：User.query.filter\_by(username='zhangsan')

高级查，名字以n结束的:User.query.filter(User.username.endswith('n')).all()

排序：User.query.order\_by(User.username)

查询前10条：User.query.limit(10).all()

返回单个ORM对象

根据主键查询:User.query.get(1)

获取第一条记录:User.query.first()

视图快捷函数：有则返回，无则返回404

User.query.first\_or\_404()

User.query.get\_or\_404()

多表关联查询

db.session.query(User).join(Address)

User.query.join(Address)

# 分页

@app.route('/user/<int:page>')  
def page\_user(page):  
 page\_data=User.query.paginate(page=page,per\_page=10)  
 return render\_template('page\_user.html',page\_data=page\_data)

---------------------------------------------------------------

{% for i in page\_data.items %}  
<li>{{i.username}}:{{i.password}}</li>

has\_prev/has\_next——是否有下一页/下一页

iter\_pages()——当前页的数据列表

prev\_prev/next\_num——上一页/下一页的页码

total——总记录数

pages——总页数

# 表单：

安装：pip install Flask-WTF

配置：

配置了SECRET\_KEY，可忽略

app.config['WTF\_SCRF\_SECRET\_KEY'] = '1234'

## 表单模型：

from flask\_wtf import FlaskForm  
from wtforms import StringField

class LoginForm(FlaskForm):  
 username=StringField(label='用户名',validators=[Phone\_required])

## 添加class等字段

render\_kw

class RegisterForm(FlaskForm):  
 # 用户注册  
 username = StringField(label='用户名', render\_kw={  
 'class': "form-control input-lg",  
 'placeholder': "用户名"  
 }, validators=[DataRequired('请输入用户名'), Phone\_required])  
 nickname = StringField(label='昵称', render\_kw={  
 'class': "form-control input-lg",  
 'placeholder': "昵称"  
 }, validators=[DataRequired('请输入昵称'),  
 Length(min=2, max=10, message='长度为2~10个字符')])  
 password = PasswordField(label='密码', render\_kw={  
 'class': "form-control input-lg",  
 'placeholder': "请输入密码"  
 }, validators=[DataRequired('请输入密码'),  
 Length(min=2, max=10, message='密码长度为2~10个字符')])  
 confirm\_password = PasswordField(label='确认密码', render\_kw={  
 'class': "form-control input-lg",  
 'placeholder': "请重复密码"  
 }, validators=[DataRequired('请重复密码'),  
 EqualTo(fieldname='password',message='两次密码不一致')])

# wtforms内置的校验器

DataRequired：检测是否输入了数值

Email：验证Email格式

EqualTo：两个数据是否相等

Length：验证字符串的长度

NumberRange：在最小值与最大值之间

Regexp：正则表达式验证

class RegisterFrom(FlaskForm):  
 username=StringField(validators=[Length(max=6,min=2,message='2位以上'),DataRequired('请输入')])  
 password=PasswordField(validators=[EqualTo('username',message='不一样')])

自定义验证

def username\_required(form, field):  
 username = field.data  
 if re.search('^1[0**-**9]{10}$', username):  
 return username  
 raise ValueError('手机号不正确')

## 表单保存数据：

<form action="{{ url\_for('page\_form') }}" method="post">

{{ form.csrf\_token }}

--------------------------------------------------------------

@app.route('/form',methods=["GET","POST"])  
def page\_form():  
 form = LoginForm()  
 if form.validate\_on\_submit():  
 username=form.username.data

## 表单格式：

from wtforms.validators import DataRequired,ValidationError

系统自带：

password=PasswordField(label='密码',validators=[DataRequired('请输入密码')])

自定义：

def Phone\_required(self, field):  
 username = field.data  
 pattern = r'^1[0-9]{10}$'  
 if not re.search(pattern, username):  
 raise ValidationError('请输入手机号')  
 return field

html，错误消息展示

{% if form.username.errors %}  
<ul class="list-unstyled text-danger">  
 {% for err in form.username.errors %}  
 <li>{{ err }}</li>  
 {% endfor %}  
</ul>  
{% endif %}

## 图片上传：

<form action="{{ url\_for('img\_upload') }}" method="post" enctype="multipart/form-data">  
 <input type="file" name="file1">  
 <input type="submit" value="上传">

---------------------------------------------------------------

@app.route('/up',methods=["GET","POST"])  
def img\_upload():  
 if request.method=='POST':  
 files=request.files  
 file1=files['file1']  
 if file1:

f\_name=secure\_filename(file1.filename)  
 file1.save(f\_name)  
 return render\_template('img\_upload.html')

用form实现

form

class UserAvatarForm(FlaskForm):  
 *""" 用户头像上传 """* avatar = FileField(label='上传头像', validators=[  
 FileRequired('请选择头像文件'),  
 FileAllowed(['png'], '仅支持PNG图片上传')  
 ])

view

def avatar\_upload():  
 *""" 头像上传 """* form = UserAvatarForm()  
 if form.validate\_on\_submit():  
 # 获取图片对象  
 img = form.avatar.data  
 f\_name = secure\_filename(img.filename)  
 img.save(f\_name)  
 return redirect('/')  
 else:  
 print(form.errors)  
 return render\_template('avatar\_upload.html', form=form)

或者

img = self.img.data  
if img:  
 img\_name=secure\_filename(img.filename)  
 img\_path=os.path.join(current\_app.config['MEDIA\_ROOT'],img\_name)  
 img.save(img\_path)

# API

pip install flask-restful

# 宏

{% macro form\_field\_errors(err\_list) %}  
 {% if err\_list %}  
 <ul class="list-unstyled text-danger">  
 {% for err in err\_list %}  
 <li>{{ err }}</li>  
 {% endfor %}  
 </ul>  
 {% endif %}  
{% endmacro %}

导入

{% from 'maco/form\_errors.html' import form\_field\_errors %}

# 模块

## 验证

数据库用户名唯一

def validate\_username(self,field):  
 # 检查用户名唯一  
 user=User.query.filter\_by(username=field.data).first()  
 if user:  
 raise ValidationError('手机号已存在')  
 return field

手机号

def Phone\_required(self, field):  
 username = field.data  
 pattern = r'^1[0-9]{10}$'  
 if not re.search(pattern, username):  
 raise ValidationError('请输入手机号')  
 return field

异步分页

view

@qa.route('/follow')  
def follow():  
 return render\_template('follow.html')  
  
@qa.route('/que\_list')  
def que\_list():  
 per\_page = 2  
 page = int(request.args.get('page', 1))  
 page\_data = Question.query.filter\_by(is\_valid=True).paginate(page=page, per\_page=per\_page)  
 return render\_template('que\_list.html', page\_data=page\_data)

html

<script>  
 $(function(){  
 var container=$('#que\_list');  
 var page=1;  
 $('#more').click(function(data){  
 $.get('{{url\_for("qa.que\_list")}}',{page:page},function(data){  
 var html=$(data);  
 // 点击，查看更多，把详细的那个p标签展示出来，把自己隐藏

//这里网页原js，加上html表示应用范围 $('.show-desc',html).click(function () {  
 $(this).addClass('hidden');  
 $(this).next().removeClass('hidden')  
 })  
 $('.show-all',html).click(function () {  
 $(this).addClass('hidden');  
 $(this).prev().removeClass('hidden')  
 })  
  
 container.append(html)  
 page+=1  
 })  
 })  
 })  
</script>

# ps

{% autoescape false %}  
{{question.content|safe}}  
{% endautoescape %}