模板的使用

Flask Template (模板学习)
学习目标

```
基本使用
过滤器&自定义过滤器
控制代码块
宏、继承、包含
Flask 的模板中特有变量和方法
CSRF
```

Jinja2模板引擎简介(template)

模板

Jinja2模板引擎简介(template)
模板

视图函数的主要作用是生成请求的响应，这是最简单的请求。实际上，视图函数有两个作用：处理业务逻辑和返回响应内容。在大型应用中，把业务逻辑和表现内容放在一起，会增加代码的复杂度和维护成本。本节学到的模板，它的作用即是承担视图函数的另一个作用，即返回响应内容。

```
模板其实是一个包含响应文本的文件，其中用占位符(变量)表示动态部分，告诉模板引擎其具体的值需要从使用的数据中获取
使用真实值替换变量，再返回最终得到的字符串，这个过程称为“渲染”
Flask是使用 Jinja2 这个模板引擎来渲染模板
```

使用模板的好处：

```
视图函数只负责业务逻辑和数据处理(业务逻辑方面)
而模板则取到视图函数的数据结果进行展示(视图展示方面)
代码结构清晰，耦合度低
```

Jinja2
两个概念：

```
Jinja2：是 Python 下一个被广泛应用的模板引擎，是由Python实现的模板语言，他的设计思想来源于 Django 的模板引擎，并扩展了其语法和一系列强大的功能，其是Flask内置的模板语言。
模板语言：是一种被设计来自动生成文档的简单文本格式，在模板语言中，一般都会把一些变量传给模板，替换模板的特定位置上预先定义好的占位变量名。
官方文档
```

渲染模版函数

```
Flask提供的 render_template 函数封装了该模板引擎
render_template 函数的第一个参数是模板的文件名，后面的参数都是键值对，表示模板中变量对应的真实值。
```

使用

```
{{}} 来表示变量名，这种 {{}} 语法叫做变量代码块
<h1>{{ post.title }}</h1>
```

Jinja2 模版中的变量代码块可以是任意 Python 类型或者对象，只要它能够被 Python 的 str() 方法转换为一个字符串就可以，比如，可以通过下面的方式显示一个字典或者列表中的某个元素:

```
{{your_dict['key']}}
{{your_list[0]}}
```

```
用 {%%} 定义的控制代码块，可以实现一些语言层次的功能，比如循环或者if语句
```

```
{% if user %}
    {{ user }}
{% else %}
    hello!
<ul>
    {% for index in indexs %}
    <li> {{ index }} </li>
    {% endfor %}
</ul>
```

注释

```
使用 {# #} 进行注释，注释的内容不会在html中被渲染出来
{# {{ name }} #}
```

模板使用

```
#.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>{{title | reverse | upper}}</h1>
    <br>
    {{list2 | listreverse}}
    <br>
    <ul>
        {% for item in my_list %}
        <li>{{item.id}}----{{item.value}}</li>
        {% endfor %}
    </ul>


    {% for item in my_list %}
        {% if loop.index==1 %}
            <li style="background-color: red;">{{ loop.index }}--{{
item.get('value') }}</li>
        {% elif loop.index==2 %}
            <li style="background-color: blue;">{{ loop.index }}--{{
item.get('value') }}</li>
```

```
        {% elif loop.index==3 %}
            <li style="background-color: green;">{{ loop.index }}--{{
item.get('value') }}</li>
        {% else %}
            <li style="background-color: yellow;">{{ loop.index }}--{{
item.get('value') }}</li>
        {% endif %}
    {% endfor %}

</body>
</html>
```

```
#.py
from flask import Flask
from flask import render_template

app = Flask(__name__)
@app.route('/')
def index():
    list1 = list(range(10))
    my_list = [{"id": 1, "value": "我爱工作"},
               {"id": 2, "value": "工作使人快乐"},
               {"id": 3, "value": "沉迷于工作无法自拔"},
               {"id": 4, "value": "日渐消瘦"},
               {"id": 5, "value": "以梦为马，越骑越傻"}]
    return render_template(
        # 渲染模板语言
        'index.html',
        title='hello world',
        list2=list1,
        my_list=my_list
        )
# step1 定义过滤器
def do_listreverse(li):
    temp_li = list(li)
    temp_li.reverse()
    return temp_li
# step2 添加自定义过滤器
app.add_template_filter(do_listreverse, 'listreverse')
if __name__ == '__main__':
    app.run(debug=True)
```

过滤器

过滤器的本质就是函数。有时候我们不仅仅只是需要输出变量的值，我们还需要修改变量的显示，甚至格式化、运算等等，而在模板中是不能直接调用 Python 中的某些方法，那么这就用到了过滤器。

链式调用

```
{{ "hello world" | reverse | upper }}
```

常见内建过滤器

字符串操作

```
first：取第一个元素
```

```
safe：禁用转义
<p>{{ '<em>hello</em>' | safe }}</p>
capitalize：把变量值的首字母转成大写，其余字母转小写
<p>{{ 'hello' | capitalize }}</p>
lower：把值转成小写
<p>{{ 'HELLO' | lower }}</p>
upper：把值转成大写
<p>{{ 'hello' | upper }}</p>
title：把值中的每个单词的首字母都转成大写
<p>{{ 'hello' | title }}</p>
reverse：字符串反转
<p>{{ 'olleh' | reverse }}</p>
format：格式化输出
<p>{{ '%s is %d' | format('name',17) }}</p>
striptags：渲染之前把值中所有的HTML标签都删掉
<p>{{ '<em>hello</em>' | striptags }}</p>
truncate: 字符串截断
<p>{{ 'hello every one' | truncate(9)}}</p>
```

列表操作

```
first：取第一个元素
列表操作
first：取第一个元素
<p>{{ [1,2,3,4,5,6] | first }}</p>
last：取最后一个元素
<p>{{ [1,2,3,4,5,6] | last }}</p>
length：获取列表长度
<p>{{ [1,2,3,4,5,6] | length }}</p>
sum：列表求和
<p>{{ [1,2,3,4,5,6] | sum }}</p>
sort：列表排序
<p>{{ [6,2,3,1,5,4] | sort }}</p>
```

语句块操作

```
{% filter upper %}
    #一大堆文字#
{% endfilter %}
```

自定义过滤器

```
from flask import Flask
from flask import render_template

app = Flask(__name__)
@app.route('/')
def index():
    list1 = list(range(10))
    my_list = [{"id": 1, "value": "我爱工作"},
               {"id": 2, "value": "工作使人快乐"},
               {"id": 3, "value": "沉迷于工作无法自拔"},
               {"id": 4, "value": "日渐消瘦"},
               {"id": 5, "value": "以梦为马，越骑越傻"}]
    return render_template(
        # 渲染模板语言
```

```
        'index.html',
        title='hello world',
        list2=list1,
        my_list=my_list
        )
# step1 定义过滤器
def do_listreverse(li):
    temp_li = list(li)
    temp_li.reverse()
    return temp_li
# step2 添加自定义过滤器
app.add_template_filter(do_listreverse, 'listreverse')
if __name__ == '__main__':
    app.run(debug=True)
```

控制代码块

控制代码块主要包含两个:

- ```
  - if/else if /else / endif
  - for / endfor
  ```

  条件语句

  ```
  {% if comments | length > 0 %}
    There are {{ comments | length }} comments
  {% else %}
    There are no comments
  {% endif %}
  ```

  循环语句

  ```
  {% for post in posts %}

        <h1>{{ post.title }}</h1>
        {{ post.text | safe }}

  {% endfor %}
  ```

  结合使用
  模拟continue功能

```
{% for post in posts if post.text %}

        <h1>{{ post.title }}</h1>
        {{ post.text | safe }}

{% endfor %}
```

代码复用
宏 —- 类似python中的函数

```
创建:
        {% macro 标签名(key=value)%} {% end macro %}
```
```

```
# d8_macro1.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<!-- step1 宏导入 import filename as xx-->
{% import 'd8_macro2.html' as fun %}
<!-- step2 宏调用 类似于python函数调用 -->
{{ fun.input('button','zhuce') }}
</body>
</html>

# d8_macro2.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<!-- step1 宏导入 import filename as xx-->
{% import 'd8_macro2.html' as fun %}
<!-- step2 宏调用 类似于python函数调用 -->
{{ fun.input('button','zhuce') }}
</body>
</html>
```

继承

关键字: block extends

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>top123 {{ title }}</h1><br>

<!-- 预留空间 给继承的html -->
{% block body %}{% endblock %}

<br><h1>bottom</h1>
</body>
</html>
```

```
{% extends 'd1_base.html' %}

{% block body %}
<h2>detail</h2>
{% endblock %}
```

继承:常常用于上下部分不做修改的网址

包含

包含: 将一个模板加载到另一个模板里面种

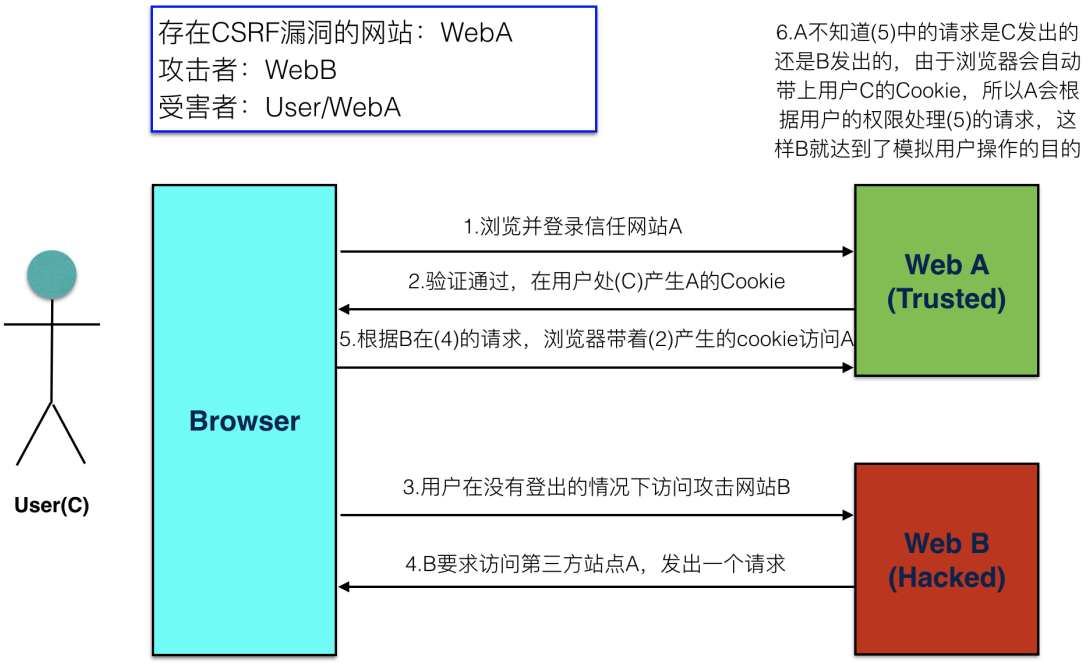
```
<h1>这是一个多页面共同的内容</h1>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<!-- 特殊的自定义 函数 -->
<h1>{{ request.url }}</h1>
<h2>{{ g.title }}</h2>
index
<br>
<!-- include 调用html文件 -->
{% include 'd2_include.html' %}
<hr>
<!-- 已经内置的特殊变量 -->
{{ url_for('detail') }}<br>
<a href="{{ url_for('detail') }}">详细页面</a>
</body>
</html>
```

总结:

```
宏(Macro)、继承(Block)、包含(include)均能实现代码的复用。
继承(Block)的本质是代码替换，一般用来实现多个页面中重复不变的区域。
宏(Macro)的功能类似函数，可以传入参数，需要定义、调用。
包含(include)是直接将目标模板文件整个渲染出来。
```

crsf 跨域请求伪造



1.能够描述出这是一种什么样的攻击方式

```
a. 访问A时进行了登录
b. A进行了状态保持
c. 访问网站B
d. 返回攻击代码
e. 向网站A发起请求 在用户未意识到的情况下
```

2.如何防止这种攻击?

```
1.修改操作 由get操作 改为 post方式
数据修改 get ==》 post
跨网站 借用用户信息 访问目标网站
2.口令验证 随机产生
随机产生一个口令 拿这个口令做对比
安装包 pip install flask-wtf
```

3.作为开发人员如何防止这样的攻击

```
1.设置加密字段
app.secret_key('fad')
2.引入类
from flask_wtf.csrf import CRSFProtect
3.创建对象
CRSFProtect(app)
4.在模板的form中生成一个随机的口令值
<input type="hidden" name="crsf_token" value="{{csrf_token()}}">
```

4.知道实现代码 背后做了什么

```
注册了请求勾子before_request 这个函数会在视图函数前执行 进行口令验证
如何验证? 接收表单传递的口令 与session的口令对比

CRSFProtect === init === init_app ==== crsf_protect === protect ===
saft_str_cmp(session[filed_name], token)
session中的值是什么时候写  在调用csrf_token()函数时
```

```
from flask import Flask, render_template
app = Flask(__name__)
# 1 设置加密字符串
app.secret_key = 'python'
# 2. 引入类
from flask_wtf.csrf import CSRFProtect
# 3. 创建对象
CSRFProtect(app)
@app.route("/")
def index():
    return render_template('d3_index.html')
@app.route("/detail", methods=['POST'])
def detail():
    return 'ok'
if __name__ == '__main__':
    app.run()
```

模板

在前面的示例中，视图函数的主要作用是生成请求的响应，这是最简单的请求。实际上，视图函数有两个作用：处理业务逻辑和返回响应内容。在大型应用中，把业务逻辑和表现内容放在一起，会增加代码的复杂度和维护成本。本节学到的模板，它的作用即是承担视图函数的另一个作用，即返回响应内容。模板其实是一个包含响应文本的文件，其中用占位符（变量）表示动态部分，告诉模板引擎其具体值需要从使用的数据中获取。使用真实值替换变量，再返回最终得到的字符串，这个过程称为“渲染”。Flask使用Jinja2这个模板引擎来渲染模板。Jinja2能识别所有类型的变量，包括{}。 Jinja2模板引擎，Flask提供的render_template函数封装了该模板引擎，render_template函数的第一个参数是模板的文件名，后面的参数都是键值对，表示模板中变量对应的真实值。

Jinja2官方文档（http://docs.jinkan.org/docs/jinja2/）

我们先来认识下模板的基本语法：

```
{% if user %}
  {{ user }}
{% else %}
  hello!
<ul>
  {% for index in indexs %}
  <li> {{ index }} </li>
</ul>
```

通过修改一下前面的示例，来学习下模板的简单使用：

```
@app.route('/')
def hello_itcast():
  return render_template('index.html')
@app.route('/user/<name>')
def hello_user(name):
  return render_template('index.html',name=name)
```

变量

在模板中{{ variable }}结构表示变量，是一种特殊的占位符，告诉模板引擎这个位置的值，从渲染模板时使用的数据中获取；Jinja2除了能识别基本类型的变量，还能识别{}；

```
<span>{{mydict['key']}}</span>
<br/>
<span>{{mylist[1]}}</span>
<br/>
<span>{{mylist[myvariable]}}</span>
```

```
from flask import Flask,render_template
app = Flask(__name__)
@app.route('/')
def index():
  mydict = {'key':'silence is gold'}
  mylist = ['Speech', 'is','silver']
  myintvar = 0
  return render_template('vars.html',
              mydict=mydict,
              mylist=mylist,
              myintvar=myintvar
              )
if __name__ == '__main__':
  app.run(debug=True)
```

反向路由： Flask提供了url_for()辅助函数，可以使用程序URL映射中保存的信息生成URL；url_for()接收视图函数名作为参数，返回对应的URL；

如调用url_for('index',_external=True)返回的是绝对地址，在下面这个示例中是 http://localhost:5000/。

如调用url_for('index',name='apple',_external=True)返回的是：http://localhost:5000/index/apple：

```
@app.route('/')
def hello_itcast():
  return render_template('index.html')
@app.route('/user/<name>')
def hello_user(name):
  return url_for('hello_itcast',_external=True)
```

自定义错误页面：

```
from flask import Flask,render_template
@app.errorhandler(404)
def page_not_found(e):
  return render_template('404.html'), 404
```

过滤器：

过滤器的本质就是函数。有时候我们不仅仅只是需要输出变量的值，我们还需要修改变量的显示，甚至格式化、运算等等，这就用到了过滤器。 过滤器的使用方式为：变量名 | 过滤器。 过滤器名写在变量名后面，中间用 | 分隔。如：{{variable | capitalize}}，这个过滤器的作用：把变量variable的值的首字母转换为大写，其他字母转换为小写。 其他常用过滤器如下：

safe：禁用转义；

```
<p>{{ '<em>hello</em>' | safe }}</p>
```

capitalize：把变量值的首字母转成大写，其余字母转小写；

```
<p>{{ 'hello' | capitalize }}</p>
```

lower：把值转成小写；

```
<p>{{ 'HELLO' | lower }}</p>
```

upper：把值转成大写；

```
<p>{{ 'hello' | upper }}</p>
```

title：把值中的每个单词的首字母都转成大写；

```
<p>{{ 'hello' | title }}</p>
```

trim：把值的首尾空格去掉；

```
<p>{{ ' hello world ' | trim }}</p>
```

reverse:字符串反转；

```
<p>{{ 'olleh' | reverse }}</p>
```

format:格式化输出；

```
<p>{{ '%s is %d' | format('name',17) }}</p>
```

striptags：渲染之前把值中所有的HTML标签都删掉；

```
<p>{{ '<em>hello</em>' | striptags }}</p>
```

语句块过滤(不常用)：

```
{% filter upper %}
  this is a Flask Jinja2 introduction
{% endfilter %}
```

自定义过滤器：

通过Flask应用对象的add_template_filter方法，函数的第一个参数是过滤器函数，第二个参数是过滤器名称。然后，在模板中就可以使用自定义的过滤器。

```
def filter_double_sort(ls):
  return ls[::2]
app.add_template_filter(filter_double_sort,'double_2')
```

Web表单：

web表单是web应用程序的基本功能。

它是HTML页面中负责数据采集的部件。表单有三个部分组成：表单标签、表单域、表单按钮。表单允许用户输入数据，负责HTML页面数据采集，通过表单将用户输入的数据提交给服务器。

在Flask中，为了处理web表单，我们一般使用Flask-WTF扩展，它封装了WTForms，并且它有验证表单数据的功能。

WTForms支持的HTML标准字段

字段对象	说明
StringField	文本字段
TextAreaField	多行文本字段
PasswordField	密码文本字段
HiddenField	隐藏文本字段
DateField	文本字段,值为 datetime.date 格式
DateTimeField	文本字段,值为 datetime.datetime 格式
IntegerField	文本字段,值为整数
DecimalField	文本字段,值为 decimal.Decimal
FloatField	文本字段,值为浮点数
BooleanField	复选框,值为 True 和 False
RadioField	一组单选框
SelectField	下拉列表
SelectMultipleField	下拉列表,可选择多个值
FileField	文件上传字段
SubmitField	表单提交按钮
FormField	把表单作为字段嵌入另一个表单
FieldList	一组指定类型的字段

WTForms常用验证函数

验证函数	说明
DataRequired	确保字段中有数据
EqualTo	比较两个字段的值，常用于比对两次密码输入
Length	验证输入的字符串长度
NumberRange	验证输入的值在数字范围内
URL	验证URL
AnyOf	验证输入值在可选列表中
NoneOf	验证输入值不在可选列表中

使用Flask-WTF需要配置参数SECRET_KEY。

CSRF_ENABLED是为了CSRF（跨站请求伪造）保护。 SECRET_KEY用来生成加密令牌，当CSRF激活的时候，该设置会根据设置的密匙生成加密令牌。

```
<form method='post'>
  <input type="text" name="username" placeholder='Username'>
  <input type="password" name="password" placeholder='password'>
  <input type="submit">
</form>
```

```
from flask import Flask,render_template
@app.route('/login',methods=['GET','POST'])
def login():
  if request.method == 'POST':
    username = request.form['username']
    password = request.form['password']
    print username,password
  return render_template('login.html',method=request.method)
```

配置参数:

```
app.config['SECRET_KEY'] = 'silents is gold'
{{ form.username.label }}
{{ form.username() }}
{{ form.password.label }}
{{ form.password() }}
{{ form.submit() }}
```

我们通过登录页面来演示表单的使用。

```
#coding=utf-8
from flask import Flask,render_template,\
  flash,redirect,url_for,session
#导入WTF扩展包的Form基类
from flask_wtf import Form
from wtforms.validators import DataRequired,EqualTo
from wtforms import StringField,PasswordField,SubmitField
app = Flask(__name__)
#设置secret_key,防止跨站请求攻击
app.config['SECRET_KEY'] = '2017'
#自定义表单类，继承Form
class Login(Form):
  us = StringField(validators=[DataRequired()])
  ps = PasswordField(validators=[DataRequired(),EqualTo('ps2','error')])
```

```
  ps2 = PasswordField(validators=[DataRequired()])
  submit = SubmitField()
#定义视图函数，实例化自定义的表单类，
@app.route('/',methods=['GET','POST'])
def forms():
  #实例化表单对象
  form = Login()
  if form.validate_on_submit():
  #获取表单数据
    user = form.us.data
    pswd = form.ps.data
    pswd2 = form.ps2.data
    print user,pswd,pswd2
    session['name'] = form.us.data
    flash(u'登陆成功')
    return redirect(url_for('forms'))
  else:
    print form.validate_on_submit()
  return render_template('forms.html',form=form)
if __name__ == "__main__":
  app.run(debug=True)
```

控制语句

常用的几种控制语句：

模板中的if控制语句

```
@app.route('/user')
def user():
  user = 'dongGe'
  return render_template('user.html',user=user)
```

```
<html>
 <head>
   {% if user %}
    <title> hello {{user}} </title>
  {% else %}
     <title> welcome to flask </title>
  {% endif %}
 </head>
 <body>
   <h1>hello world</h1>
 </body>
 </html>
```

模板中的for循环语句**

```
@app.route('/loop')
def loop():
 fruit = ['apple','orange','pear','grape']
 return render_template('loop.html',fruit=fruit)
```

```
<html>
<head>
```

```
  {% if user %}
   <title> hello {{user}} </title>
 {% else %}
    <title> welcome to flask </title>
 {% endif %}
</head>
<body>
  <h1>hello world</h1>
 <ul>
   {% for index in fruit %}
     <li>{{ index }}</li>
   {% endfor %}
 </ul>
</body>
</html>
```

宏：

类似于python中的函数，宏的作用就是在模板中重复利用代码，避免代码冗余。

Jinja2支持宏，还可以导入宏，需要在多处重复使用的模板代码片段可以写入单独的文件，再包含在所有模板中，以避免重复。

定义宏

```
{% macro input() %}
 <input type="text"
     name="username"
     value=""
     size="30"/>
{% endmacro %}
```

调用宏

```
{{ input()}}
#定义带参数的宏
#调用宏，并传递参数
  <input type="password"
      name=""
      value="name"
      size="40"/>
```

模板继承：

模板继承是为了重用模板中的公共内容。{% block head %}标签定义的元素可以在衍生模板中修改，extends指令声明这个模板继承自哪？父模板中定义的块在子模板中被重新定义，在子模板中调用父模板的内容可以使用super()。

```
{% extends 'base.html' %}
{% block content %}
 <h1> hi,{{ name }} </h1>
{% for index in fruit %}
 <p> {{ index }} </p>
{% endfor %}
{% endblock %}
```

Flask中的特殊变量和方法：

在Flask中，有一些特殊的变量和方法是可以在模板文件中直接访问的。

config 对象:

config 对象就是Flask的config对象，也就是 app.config 对象。

```
{{ config.SQLALCHEMY_DATABASE_URI }}
```

request 对象:

就是 Flask 中表示当前请求的 request 对象。

```
{{ request.url }}
```

url_for 方法:

url_for() 会返回传入的路由函数对应的URL，所谓路由函数就是被 app.route() 路由装饰器装饰的函数。如果我们定义的路由函数是带有参数的，则可以将这些参数作为命名参数传入。

```
{{ url_for('index') }}
{{ url_for('post', post_id=1024) }}
```

get_flashed_messages方法：

返回之前在Flask中通过 flash() 传入的信息列表。把字符串对象表示的消息加入到一个消息队列中，然后通过调用 get_flashed_messages() 方法取出。

```
{% for message in get_flashed_messages() %}
  {{ message }}
{% endfor %}
```

希望本文所述对大家基于flask框架的Python程序设计有所帮助。