[**flask插件系列之Flask-WTF表单**](https://www.cnblogs.com/cwp-bg/p/9714741.html)

flask\_wtf是flask框架的表单验证模块，可以很方便生成表单，也可以当做json数据交互的验证工具，支持热插拔。

**安装**

pip install Flask-WTF

Flask-WTF其实是对wtforms组件的封装，使其支持对flask框架的热插拔。

**简单使用**

# app.py

from flask import Flask, current\_app, request, render\_template

from forms import MyForm

app = Flask(\_\_name\_\_,template\_folder='static/html')

@app.route('/',methods=['GET','POST'])

def login():

form = MyForm()

if form.validate\_on\_submit():

return 'OK'

return render\_template('forms/index.html', form=form)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='127.0.0.1', port=80, debug=True)

# forms.py

from flask\_wtf import FlaskForm

from wtforms import StringField

from wtforms.validators import DataRequired

class MyForm(FlaskForm):

name = StringField('name', validators=[DataRequired()])

# forms/index.html

<form method="POST" action="/">

{{ form.csrf\_token }}

{{ form.name.label }} {{ form.name(size=20) }}

<input type="submit" value="Go">

</form>

**flask\_wtf定义字段**

flask\_wtf完全使用wtforms组件的字段模型，wtforms对字段的定义在fields模块；又分为core和simple，core模块定义了普通使用的字段，simple在core模块的基础上扩展了一些字段，这些字段会自动进行字段级别的校验。

* **字段类型**

# core.py

\_\_all\_\_ = (

'BooleanField', 'DecimalField', 'DateField', 'DateTimeField', 'FieldList',

'FloatField', 'FormField', 'IntegerField', 'RadioField', 'SelectField',

'SelectMultipleField', 'StringField',

)

常用字段说明：

BooleanField：布尔类型，如Flask,True

StringField：字符串类型

DecimalField：小数点文本字段，如：‘1.23’

DateField：日期字段，格式：'%Y-%m-%d'

DateTimeField:日期字段，格式：'%Y-%m-%d %H:%M:%S'

FieldList:统一字段类型组成列表，如：FieldList(StringField('Name', [validators.required()]))

FloatField:浮点数类型

IntegerField：整形

SelectMultipleField：多选框

RadioField：单选框

# simple.py

TextAreaField:文本域，可接受多行输入

PasswordField：密码字段，输入的不会直接在浏览器明文显示

FileField：上传文件，但不会处理验证文件，需要手动处理

HiddenField：隐藏字段

SubmitField：按钮

TextField:字符串类型的别名，弃用

* **表单定义**

# 参数：

class UserAdminForm(FlaskForm):

username = StringField(label='用户名', validators=[DataRequired(),Length(4,20)])

password\_hash = PasswordField(label='密码',validators=[DataRequired(),Length(4,20)])

limit = SelectField(label='用户权限',

choices=[('guest', '所有权限'),

('operation', '可读可写不可删除'),

('management', '可读不可写')],

default='guest') # 权限

# 字段一般的参数

# label:字段的名字

# default：默认

# validators：字段的验证序列

# description：字段的描述

# choices：SelectField和他的子类有的字段，选择框，多选一

* **字段的验证序列**

字段的参数validators可以指定提交表单的验证序列，按照从左到右的顺序，默认的可选验证在wtforms.validators模块，已经封装的验证方法有：

\_\_all\_\_ = (

'DataRequired', 'data\_required', 'Email', 'email', 'EqualTo', 'equal\_to',

'IPAddress', 'ip\_address', 'InputRequired', 'input\_required', 'Length',

'length', 'NumberRange', 'number\_range', 'Optional', 'optional',

'Required', 'required', 'Regexp', 'regexp', 'URL', 'url', 'AnyOf',

'any\_of', 'NoneOf', 'none\_of', 'MacAddress', 'mac\_address', 'UUID'

)

模块中大小写有对应的方式，如DataRequired对应data\_required。

DataRequired/data\_required:验证数据是否真实存在，即不能为空，必须是非空白字符串，否则触发StopValidation错误。

InputRequired/input\_required：和DataRequired的区别在于可以是空白字符串；

Required/required:data\_required的别名

Email/email：验证符合邮件的格式，只有最基本的验证；

EqualTo/equal\_to:比较两个字段的值，比如密码和确认密码，如果不相等就触发错误，equal\_to(field,message),需要输入另一个字段的名字。

IPAddress/ip\_address:验证是否是ip地址，默认验证IPV4地址。

MacAddress/mac\_address:验证是否符合mac格式；

UUID：是否是uuid格式；

URL/url:验证是否符合url格式；

Regexp/regexp:用提供的正则表达式验证字段；Regexp(r"")

Length/length:设置字段值的长度，Length(min,max);

NumberRange/number\_range:设置一个数字字段的取值范围，可以针对浮点数和小数；NumberRange(min,max)

Optional/optional:允许字段为空并停止验证；

NoneOf/none\_of:将传入的数据和无效的比较，是抛出异常；Noneof(values).

Anyof/any\_of:将传入的数据和预设的数据比较，不是异常。Anyof(values).

* **自定义字段验证**

如果默认的验证序列不满足我们的要求，我们可以通过继承的方式自定义字段。

from wtforms.validators import DataRequired,Length,StopValidation

class NewStringField(StringField):

"""

自定义一个新的字段

"""

def pre\_validate(self, form):

"""验证方法，在validators验证序列之前"""

x:str = form.name.data

if not x.startswith('g'):

raise StopValidation("your data must be startswith 'g'")

def post\_validate(self, form, validation\_stopped):

"""

验证方法，在validators验证序列之后

:param form:该字段所属的表单对象

:param validation\_stopped:前面验证序列的结果，True表示验证通过，False表示验证失败

:return:

"""

if not validation\_stopped:

raise ValueError("验证失败了！")

pass

* 触发StopValidation异常会停止验证链；
* **自定义表单验证**

一般来说，如果对表单有额外需要的验证，一般自定义表单的额外的验证方法而不是重新自定义新的字段,而form已经为我们提供了这种方法。  
看Form对象的源码：

def validate(self):

"""

Validates the form by calling `validate` on each field, passing any

extra `Form.validate\_<fieldname>` validators to the field validator.

"""

extra = {}

for name in self.\_fields:

inline = getattr(self.\_\_class\_\_, 'validate\_%s' % name, None)

if inline is not None:

extra[name] = [inline]

return super(Form, self).validate(extra)

Form对象调用validate函数时会自动寻找validate\_%s的方法添加到验证序列,并在原先字段的验证序列验证完毕后执行。

class MyForm(FlaskForm):

name = StringField('name', validators=[DataRequired(), Length(4,20)])

def validate\_name(self, field):

print(field.data)

if hasattr(self, 'name') and len(self.name.data) > 5:

print(self.name.data)

return True

raise ValidationError('超过5个字符')

# 在自定义的验证方法中，抛出异常使用ValidationError，validate会自动捕捉。

**表单对象**

flask\_wtf推荐使用Form对象的子类FlaskForm代替，该对象提供了所有表单需要的属性和方法。那么Form对象是如何自动实现表单功能的呢？  
分析FlaskForm对象源码:

class FlaskForm(Form):

class Meta(DefaultMeta):

def wrap\_formdata(self, form, formdata):

pass

def \_\_init\_\_(self, formdata=\_Auto, \*\*kwargs):

csrf\_enabled = kwargs.pop('csrf\_enabled', None)

pass

def is\_submitted(self):

pass

def validate\_on\_submit(self):

pass

def hidden\_tag(self, \*fields):

pass

def validate(self):

pass

* FlaskForm内部定义了一个Meta类，该类添加csrf保护的一些方法，所以创建表单的时候一定要导入FlaskForm而不是Form.
* is\_submitted:检查是否有一个活跃的request请求;
* validate\_on\_submit:调用is\_submitted和validate方法，返回一个布尔值，用来判断表单是否被提交；
* validate：字段级别的验证，每个字段都有一个validate方法，FlaskForm调用validate会对所有的字段调用validate方法验证，如果所有的验证都通过返回Ture，否则抛出异常。
* hidden\_tag：获取表单隐藏的字段；
* wrap\_formdata:获取request中的form，每次form对象初始化时会执行该函数从request获取form。
* **重要属性**

form.data:字段名字和值组成的字典；

form.errors：验证失败的信息字典，在调用validate\_on\_submit方法后才有效；

form.name.data：字段name的值；

form.name.type：字段name的类型

**常用场景**

* **登录验证**

# froms.py

class UserPasswordForm(FlaskForm):

"""

登录提交的表单

"""

username = StringField('User', validators=[DataRequired()])

password = PasswordField('Password', validators=[DataRequired()])

# form.html

<form method="POST" action="/">

{{ form.csrf\_token }}

{{ form.username.label }} {{ form.username(size=20) }}

{{ form.password.label }} {{ form.password }}

<input type="submit" value="Go">

</form>

# views.py

@app.route('/login',methods=['GET','POST'])

def login():

form = UserPasswordForm()

if form.validate\_on\_submit():

# 验证表单

if form.username.data == "xiaoming" and form.password.data == '123':

return 'OK'

return render\_template('forms/index.html', form=form)

* **ajax请求转化表单**

有时候我们没有html页面的表单，只有ajax请求的数据交互，但是想借用Form来定义接口和验证接收的数据，如果ajax的请求方法是('POST', 'PUT', 'PATCH', 'DELETE')中的一种，FlaskForm会自动从request对象中调用request.form和request.get\_json()方法来接收数据，因此这种方式十分方便。注意：get方法不再其中。

# form.py

class MyForm(FlaskForm):

name = StringField('name', validators=[DataRequired(), Length(4,20)])

# view.py

@app.route('/form',methods=['GET','POST'])

def form():

if request.method != "GET":

form = MyForm() # form会获取请求数据

print(form.data)

return 'ok'

return ''

# test.py

import requests as req

import json

class ProTest():

baseurl = 'http://127.0.0.1:80'

def test\_form(self):

url = self.baseurl + '/form'

rsp = req.post(url,json={'name':'hhhh'})

# rsp = req.get(url,json={'name':'hhhh'})

if \_\_name\_\_ == '\_\_main\_\_':

test = ProTest()

test.test\_form()

* **form启用csrf保护**

默认csrf保护是开启的，只要在html文件中添加{{ form.csrf\_token }}，app必须设置SECRET\_KEY参数。

# 禁用保护

form = Form(csrf\_enabled=False)

# 或配置app时

WTF\_CSRF\_ENABLED = False

* **一般数据csrf保护**

同理必须设置SECRET\_KEY参数。

from flask\_wtf.csrf import CsrfProtect

csrf = CsrfProtect()

def create\_app():

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

csrf.init\_app(app)

# 模板中添加一个隐藏域

<input type="hidden" name="csrf\_token" value="{{ csrf\_token() }}" />

<meta name="csrf-token" content="{{ csrf\_token() }}">

# 如果是ajax请求，可以在脚本中

var csrftoken = "{{ csrf\_token() }}"

# 然后每个请求添加 X-CSRFToken 头部

# 全局禁用csrf

WTF\_CSRF\_CHECK\_DEFAULT = False

# 对一些视图跳过csrf检查

@csrf.exempt

@app.route('/foo', methods=('GET', 'POST'))

def my\_handler():

return 'ok'