# 【译】WTForms 2 中文入门教程(速成课程)

## 主要概念

- `Forms` 类是 WTForms 的核心容器. 表单(Forms)表示域(Fields)的集合, 域能通过表单的字典形式或者属性形式访问.
- `Fields` (域)做最繁重的工作. 每个域(field)代表一个数据类型, 并且域操作强制表单输入为那个数据类型. 例如, `InputRequired` 和 `StringField` 表示两种不同的数据类型. 域除了包含的数据（data）之外, 还包含大量有用的属性, 例如标签、描述、验证错误的列表.
- 每个域(field)拥有一个 `Widget` (部件)实例. Widget 的工作是渲染域(field)的HTML表示. 每个域可以指定Widget实例, 但每个域默认拥有一个合理的widget. 有些域是简单方便的, 比如 `TextAreaField` 就仅仅是默认部件(widget) 为 `TextArea` 的 `StringField`.
- 为了指定验证规则, 域包含验证器(Validators)列表.

## 开始

让我们直接进入正题并定义我们的第一个表单::

```
from wtforms import Form, BooleanField, StringField, validators

class RegistrationForm(Form):
    username     = StringField('Username', [validators.Length(min=4, max=25)])
    email        = StringField('Email Address', [validators.Length(min=6, max=35)])
    accept_rules = BooleanField('I accept the site rules', [validators.InputRequired()])
```

当你创建一个表单(form), 你定义域(field)的方法类似于很多ORM定义它们的列(columns)：通过定义类变量, 即域的实例.

因为表单是常规的 Python 类, 你可以很容易地把它们扩展成为你期望的::

```
class ProfileForm(Form):
    birthday  = DateTimeField('Your Birthday', format='%m/%d/%y')
    signature = TextAreaField('Forum Signature')

class AdminProfileForm(ProfileForm):
    username = StringField('Username', [validators.Length(max=40)])
    level    = IntegerField('User Level', [validators.NumberRange(min=0, max=10)])
```

通过子类, `AdminProfileForm` 类获得了已经定义的 `ProfileForm` 类的所有域. 这允许你轻易地在不同表单之间共享域的共同子集, 例如上面的例子, 我们增加 admin-only 的域到 `ProfileForm` .

## 使用表单

使用表单和实例化它一样简单. 想想下面这个django风格的视图函数, 它使用之前定义的 `RegistrationForm` 类::

```
def register(request):
    form = RegistrationForm(request.POST)
    if request.method == 'POST' and form.validate():
        user = User()
        user.username = form.username.data
        user.email = form.email.data
        user.save()
        redirect('register')
    return render_response('register.html', form=form)
```

首先, 我们实例化表单, 给它提供一些 `request.POST` 中可用的数据. 然后我们检查请求(request)是不是使用 POST 方式, 如果它是, 我们就验证表单, 并检查用户遵守这些规则. 如果成功了, 我们创建新的 `User` 模型, 并从已验证的表单分派数据给它, 最后保存它.

### 编辑现存对象

我们之前的注册例子展示了如何为新条目接收输入并验证, 只是如果我们想要编辑现有对象怎么办？很简单::

```
def edit_profile(request):
    user = request.current_user
    form = ProfileForm(request.POST, user)
    if request.method == 'POST' and form.validate():
        form.populate_obj(user)
        user.save()
        redirect('edit_profile')
    return render_response('edit_profile.html', form=form)
```

这里, 我们通过给表单同时提供 `request.POST` 和用户(user)对象来实例化表单. 通过这样做, 表单会从 `user` 对象得到在未在提交数据中出现的任何数据.

我们也使用表单的 `populate_obj` 方法来重新填充用户对象, 用已验证表单的内容. 这个方法提供便利, 用于当域(field)名称和你提供数据的对象的名称匹配时. 通常的, 你会想要手动分配值, 但对于这个简单例子, 它是最好的. 它也可以用于CURD和管理(admin)表单.

## 在控制台中探索

WTForms 表单是非常简单的容器对象, 也许找出表单中什么对你有用的最简单的方法就是在控制台中玩弄表单:

```
>>> from wtforms import Form, StringField, validators
>>> class UsernameForm(Form):
...     username = StringField('Username', [validators.Length(min=5)], default=u'test')
...
>>> form = UsernameForm()
>>> form['username']
<wtforms.fields.StringField object at 0x827eccc>
>>> form.username.data
u'test'
>>> form.validate()
False
>>> form.errors
{'username': [u'Field must be at least 5 characters long.']}
```

我们看到的是当你实例化一个表单的时候, 表单包含所有域的实例, 访问域可以通过字典形式或者属性形式. 这些域拥有它们自己的属性, 就和封闭的表单一样.

当我们验证表单, 它返回逻辑假, 意味着至少一个验证规则不满足. `form.errors` 会给你一个所有错误的概要.

```
>>> form2 = UsernameForm(username=u'Robert')
>>> form2.data
{'username': u'Robert'}
>>> form2.validate()
True
```

这次, 我们实例化 `UserForm` 时给 `username` 传送一个新值, 验证表单是足够了.

## 表单如何获取数据

除了使用前两个参数( `formdata` 和 `obj` )提供数据之外, 你可以传送关键词参数来填充表单. 请注意一些参数名是被保留的: `formdata` , `obj` , `prefix` .

`formdata` 比 `obj` 优先级高, `obj` 比关键词参数优先级高. 例如:

```
def change_username(request):
    user = request.current_user
    form = ChangeUsernameForm(request.POST, user, username='silly')
    if request.method == 'POST' and form.validate():
        user.username = form.username.data
        user.save()
        return redirect('change_username')
    return render_response('change_username.html', form=form)
```

虽然你在实践中几乎从未一起使用所有3种方式, 举例说明WTForms是如何查找 `username` 域:

1. 如果表单被提交( `request.POST` 非空), 则处理表单输入. 实践中, 即使这个域没有 表单输入, 而如果存在任何种类的表单输入, 那么我们会处理表单输入.
2. 如果没有表单输入, 则按下面的顺序尝试:
    1. 检查 `user` 是否有一个名为 `username` 的属性.
    2. 检查是否提供一个名为 `username` 的关键词参数.
    3. 最后, 如果都失败了, 使用域的默认值, 如果有的话.

## 验证器

WTForms中的验证器(Validators)为域(field)提供一套验证器, 当包含域的表单进行验证时运行. 你提供验证器可通过域构造函数的第二个参数 `validators`:

```
class ChangeEmailForm(Form):
    email = StringField('Email', [validators.Length(min=6, max=120), validators.Email()])
```

你可以为一个域提供任意数量的验证器. 通常, 你会想要提供一个定制的错误消息:

```
class ChangeEmailForm(Form):
    email = StringField('Email', [
        validators.Length(min=6, message=_(u'Little short for an email address?')),
        validators.Email(message=_(u'That\'s not a valid email address.'))
    ])
```

这通常更好地提供你自己的消息, 作为必要的默认消息是通用的. 这也是提供本地化错误消息的方法.

对于内置的验证器的列表, 查阅 Validators.

## 渲染域

渲染域和强制它为字符串一样简单:

```
>>> from wtforms import Form, StringField
>>> class SimpleForm(Form):
...   content = StringField('content')
...
>>> form = SimpleForm(content='foobar')
>>> str(form.content)
'<input id="content" name="content" type="text" value="foobar" />'
>>> unicode(form.content)
u'<input id="content" name="content" type="text" value="foobar" />'
```

然而, 渲染域的真正力量来自于它的 `__call__()` 方法. 调用(calling)域, 你可以提供关键词参数, 它们会在输出中作为HTML属性注入.

```
>>> form.content(style="width: 200px;", class_="bar")
u'<input class="bar" id="content" name="content" style="width: 200px;" type="text" value="f
```

现在, 让我们应用这个力量在 Jinja 模板中渲染表单. 首先, 我们的表单:

```
class LoginForm(Form):
    username = StringField('Username')
    password = PasswordField('Password')

form = LoginForm()
```

然后是模板文件:

```
<form method="POST" action="/login">
    <div>{{ form.username.label }}: {{ form.username(class="css_class") }}</div>
    <div>{{ form.password.label }}: {{ form.password() }}</div>
</form>
```

相同的, 如果你使用 Django 模板, 当你想要传送关键词参数时, 你可以使用我们在Django扩展中提供的模板标签 `form_field`:

```
{% load wtforms %}
<form method="POST" action="/login">
    <div>
        {{ form.username.label }}:
        {% form_field form.username class="css_class" %}
    </div>
    <div>
        {{ form.password.label }}:
        {{ form.password }}
```

```
    </div>
</form>
```

这两个将会输出:

```
<form method="POST" action="/login">
    <div>
        <label for="username">Username</label>:
        <input class="css_class" id="username" name="username" type="text" value="" />
    </div>
    <div>
        <label for="password">Password</label>:
        <input id="password" name="password" type="password" value="" />
    </div>
</form>
```

WTForms是模板引擎不可知的, 同时会和任何允许属性存取、字符串强制(string coercion)、函数调用的引擎共事. 在 Django 模板中, 当你不能传送参数时, 模板标签 `form_field` 提供便利.

## 显示错误消息

现在我们的表单拥有一个模板, 让我们增加错误消息::

```
<form method="POST" action="/login">
    <div>{{ form.username.label }}: {{ form.username(class="css_class") }}</div>
    {% if form.username.errors %}
        <ul class="errors">{% for error in form.username.errors %}<li>{{ error }}</li>{% er
    {% endif %}

    <div>{{ form.password.label }}: {{ form.password() }}</div>
    {% if form.password.errors %}
        <ul class="errors">{% for error in form.password.errors %}<li>{{ error }}</li>{% er
    {% endif %}
</form>
```

如果你喜欢在顶部显示大串的错误消息, 也很简单:

```
{% if form.errors %}
    <ul class="errors">
        {% for field_name, field_errors in form.errors|dictsort if field_errors %}
            {% for error in field_errors %}
                <li>{{ form[field_name].label }}: {{ error }}</li>
            {% endfor %}
        {% endfor %}
    </ul>
{% endif %}
```

由于错误处理会变成相当冗长的事情, 在你的模板中使用 Jinja 宏(macros, 或者相同意义的) 来减少引用是更好的. (例子)

## 定制验证器

这有两种方式定制的验证器. 通过定义一个定制的验证器并在域中使用它:

```
from wtforms.validators import ValidationError

def is_42(form, field):
    if field.data != 42:
        raise ValidationError('Must be 42')

class FourtyTwoForm(Form):
    num = IntegerField('Number', [is_42])
```

或者通过提供一个在表单内的特定域(in-form field-specific)的验证器:

```
class FourtyTwoForm(Form):
    num = IntegerField('Number')

    def validate_num(form, field):
        if field.data != 42:
            raise ValidationError(u'Must be 42')
```

对于更多带参数的复杂验证器, 查阅 Custom validators部分.