Flask Script扩展的作用(应用场景):

- 包括运行一个开发用的服务器,
- 一个定制的Python shell,
- 设置数据库的脚本

Flask-WTF :

为什么有 CSRF

Flask-WTF 表单保护你免受 CSRF 威胁，你不需要有任何担心。尽管如此，如果你有不包含表单的视图，那么它们仍需要额外的保护。

为了生成 CSRF 令牌，你必须指定一个密钥，这通常与你的 Flask 应用密钥一致。如果你想使用不同的密钥，可在配置中指定:

```
WTF_CSRF_SECRET_KEY = 'a random string'
```

控制代码块

if语句语法如下:

```
{%if user.is_logged_in() %}
    <a href='/logout'>Logout</a>
{% else %}
    <a href='/login'>Login</a>
{% endif %}
```

循环语句的语法如下:

```
{% for post in posts %}
    <div>
        <h1>{{ post.title }}</h1>
        <p>{{ post.text | safe }}</p>
    </div>
```

```
{% endfor %}
```

- **在一个 for 循环块中你可以访问这些特殊的变量:**

变量	描述
loop.index	当前循环迭代的次数（从 1 开始）
loop.index0	当前循环迭代的次数（从 0 开始）

-

控制语句后端代码：

```python
from flask import Flask,render_template


app = Flask(__name__)


@app.route("/")
def index():
    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"
        }
    ]
    # 第一个参数：模板的名字
    # 第二个参数：表示需要传入到模板里面的值
    return render_template("code_18_control.html",my_list = my_list)

if __name__ == '__main__':
```

```
    app.run()
```

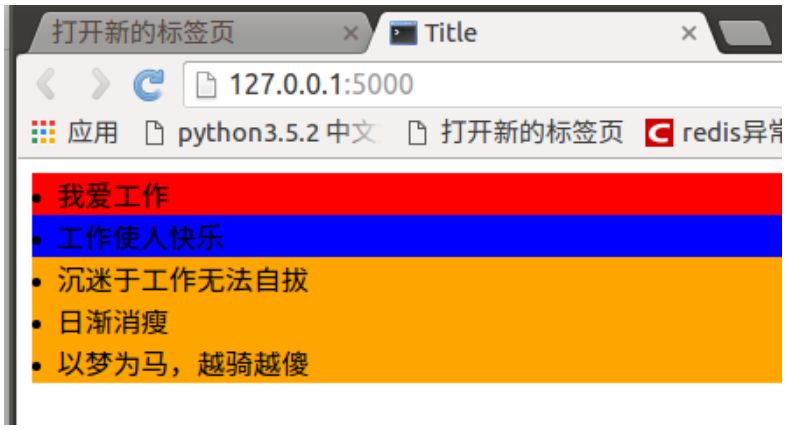
控制语句前端代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
{% for item in my_list %}
    {% if loop.index == 1 %}
        <li style="background-color: red">{{ item.value }}</li>
    {% elif loop.index == 2 %}
        <li style="background-color: blue">{{ item.value }}</li>
    {% else %}
         <li style="background-color: orange">{{ item.value }}</li>
    {% endif %}


{% endfor %}

</body>
</html>
```

运行效果：



模板中特有的变量和函数

内置变量的用法：

前端代码：

```
<h1>flask特有的变量和函数</h1>
<p>config变量的使用:{{ config.DEBUG }}</p>
<p>request变量的使用:{{ request.url }}</p>
<p>g变量的使用:{{ g.name }}</p>
```

后端代码：

```
# g变量我们可以理解是一个容器,可以往这个容器里面存值
    g.name = "itcast"
```

运行效果：

flash函数的使用：

flash的后端代码：

```
from flask import Flask,flash,render_template

app = Flask(__name__)

class Config(object):
    # 在flask里面只要是用到了session，那么都必须设置SECRET_KEY
    # SECRET_KEY:固定的语法
    SECRET_KEY = "jldkjlakfalfa"

app.config.from_object(Config)

@app.route("/")
```

```python
def index():
    # 参数是给用户一个友好提示
    flash("请输入用户名")
    flash("请输入用户名")
    flash("请输入用户名")
    return render_template("code_19_flash.html")

if __name__ == '__main__':
    app.run()
```

flash前端代码：

```html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

{% for msg in get_flashed_messages() %}
<p>{{ msg }}</p>
{% endfor %}




</body>
</html>
```

运行效果：

创建数据库

优点：

- 只需要面向对象编程, 不需要面向数据库编写代码.

    - 对数据库的操作都转化成对类属性和方法的操作.
    - 不用编写各种数据库的`sql语句`.
- 实现了数据模型与数据库的解耦, 屏蔽了不同数据库操作上的差异.
    - 不在关注用的是`mysql`、`oracle`...等.
    - 通过简单的配置就可以轻松更换数据库, 而不需要修改代码.

缺点：

- 相比较直接使用SQL语句操作数据库,有性能损失.
- 根据对象的操作转换成SQL语句,根据查询的结果转化成对象, 在映射过程中有性能损失.

需要牢记的事情:

- 您的所有模型的基类叫做 *db.Model*。它存储在您必须创建的 SQLAlchemy 实例上。

创建数据库的后端代码：

Python | L1 (default)

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate,MigrateCommand
from flask_script import Manager
app = Flask(__name__)

class Config(object):
    # 设置数据库的uri地址:SQLALCHEMY_DATABASE_URI:这个是固定的写法
    # mysql:表示数据库的类型
    # root:表示用户名
    # mysql:表示密码
    # 127.0.0.1:3306:数据库的ip地址和端口号
    # db:表示数据库的名字
    SQLALCHEMY_DATABASE_URI = "mysql://root:mysql@127.0.0.1:3306/db"
    # 信号追踪,当我们的数据库进行更改的时候，系统会发送信号,如果不需要那么可以设置为flase
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    # 打印出来原始的数据
    # app.config['SQLALCHEMY_ECHO'] = True

app.config.from_object(Config)
# 初始化数据库的对象
db = SQLAlchemy(app)




# 角色表(admin,user)
# 管理员 :张三设置成管理
```

```python
# 普通用户:李四和王五是普通的用户
# 一的一方
class Role(db.Model):
    __tablename__ = "roles"
    id = db.Column(db.Integer,primary_key=True)
    name = db.Column(db.String(128))
    # relationship:固定的写法，表示关系
    # 第一个参数表示想跟哪张产生关联
    # 相当于给role这张表定义了一个属性,跟u s e r进行关联
    # backref：固定的写法，这个属性表示可以通过一的一方推出来多的一方
    # role：可以随便取名字
    us = db.relationship("User",backref= "role")
    # __repr__:表示友好提示
    def __repr__(self):
        return "Role = %s"%self.name


# 多的一方
# 用户表(张三，李四，王五)
# db.Model是固定的写法：告诉数据库当前创建的数据库表
class User(db.Model):
    # 创建数据库表的名字:__tablename__：表示固定的写法
    __tablename__ = "users"
    # 设置表的列名
    # Column:Column是固定的写法,表示设置数据库的列
    # primary_key:primary_key是固定的写法,表示当前i d为主键
    id = db.Column(db.Integer,primary_key=True)
    name = db.Column(db.String(128))
    email = db.Column(db.String(128))
    password = db.Column(db.String(128))
    # 定义外键是在多的一方进行定义
    # ForeignKey:固定的写法，表示外键
    role_id = db.Column(db.Integer,db.ForeignKey("roles.id"))
    # 表示友好提示
    def __repr__(self):
        return "User = %s"%self.name

@app.route("/")
def index():
    return "index page"

if __name__ == '__main__':
    db.drop_all()
    # 创建数据库的表
    # db.create_all()
    #
    # ro1 = Role(name='admin')
    # db.session.add(ro1)
    # db.session.commit()
    # # 再次插入一条数据
    # ro2 = Role(name='user')
    # db.session.add(ro2)
```

```
    # db.session.commit()
    #
    # us1 = User(name='wang', email='wang@163.com', password='123456', role_id=ro1.id)
    # us2 = User(name='zhang', email='zhang@189.com', password='201512', role_id=ro2.id)
    # us3 = User(name='chen', email='chen@126.com', password='987654', role_id=ro2.id)
    # us4 = User(name='zhou', email='zhou@163.com', password='456789', role_id=ro1.id)
    # us5 = User(name='tang', email='tang@itheima.com', password='158104',
role_id=ro2.id)
    # us6 = User(name='wu', email='wu@gmail.com', password='5623514', role_id=ro2.id)
    # us7 = User(name='qian', email='qian@gmail.com', password='1543567', role_id=ro1.id)
    # us8 = User(name='liu', email='liu@itheima.com', password='867322', role_id=ro1.id)
    # us9 = User(name='li', email='li@163.com', password='4526342', role_id=ro2.id)
    # us10 = User(name='sun', email='sun@163.com', password='235523', role_id=ro2.id)
    # db.session.add_all([us1, us2, us3, us4, us5, us6, us7, us8, us9, us10])
    # db.session.commit()


    # user1 = User()
    # user1.name = "itcast"
    # user1.email = "123@itcast.cn"
    # user1.password = "123"
    # #  把数据存到数据库
    # # 下面是固定写法，只有第一次进行添加的时候，才需要add,
    # # 如果是第二次更新数据就不需要add，直接commit
    # db.session.add(user1)
    # db.session.commit()




    app.run()
```

对数据库进行查询操作

```
mysql> select * from roles;
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  2 | user  |
+----+-------+
2 rows in set (0.01 sec)

mysql> select * from users;
+----+-------+------------------+----------+---------+
| id | name  | email            | password | role_id |
+----+-------+------------------+----------+---------+
|  1 | wang  | wang@163.com     | 123456   |       1 |
|  2 | zhang | zhang@189.com    | 201512   |       2 |
|  3 | chen  | chen@126.com     | 987654   |       2 |
|  4 | zhou  | zhou@163.com     | 456789   |       1 |
|  5 | tang  | tang@itheima.com | 158104   |       2 |
|  6 | wu    | wu@gmail.com     | 5623514  |       2 |
|  7 | qian  | qian@gmail.com   | 1543567  |       1 |
|  8 | liu   | liu@itheima.com  | 867322   |       1 |
|  9 | li    | li@163.com       | 4526342  |       2 |
| 10 | sun   | sun@163.com      | 235523   |       2 |
+----+-------+------------------+----------+---------+
```

```
Terminal
+ (flask3_11) python@ubuntu:~/PycharmProjects/flask_11$ ipython3
× /usr/local/lib/python3.5/dist-packages/IPython/core/interactiveshell.py:724
roblems, please install IPython inside the virtualenv.
  warn("Attempting to work in a virtualenv. If you encounter problems, plea
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from code_20_db import *
```

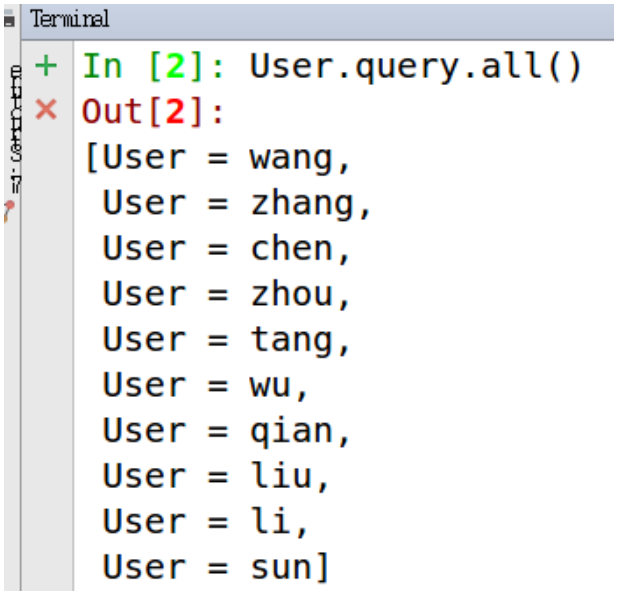
需要增加友好提示：

```
In [2]: User.query.all()
Out[2]:
[<User 1>,
 <User 2>,
 <User 3>,
 <User 4>,
 <User 5>,
 <User 6>,
 <User 7>,
 <User 8>,
 <User 9>,
 <User 10>]
```

```python
class Role(db.Model):
    __tablename__ = "roles"
    id = db.Column(db.Integer,primary_key=True)
    name = db.Column(db.String(128))
    # relationship:固定的写法，表示关系
    # 第一个参数表示想跟哪张产生关联
    # 相当于给role这张表定义了一个属性,跟 u s e r 进行关联
    # backref：固定的写法，这个属性表示可以通过一的一方推出来多的一方
    # role：可以随便取名字
    us = db.relationship("User",backref= "role")
    # __repr__:表示友好提示
    def __repr__(self):
        return "Role = %s"%self.name


# 多的一方
# 用户表(张三，李四，王五)
# db.Model是固定的写法：告诉数据库当前创建的数据库表
class User(db.Model):
    # 创建数据库表的名字:__tablename__：表示固定的写法
    __tablename__ = "users"
    # 设置表的列名
    # Column:Column是固定的写法,表示设置数据库的列
    # primary_key:primary_key是固定的写法,表示当前 i d 为主键
    id = db.Column(db.Integer,primary_key=True)
    name = db.Column(db.String(128))
    email = db.Column(db.String(128))
    password = db.Column(db.String(128))
    # 定义外键是在多的一方进行定义
    # ForeignKey:固定的写法，表示外键
    role_id = db.Column(db.Integer,db.ForeignKey("roles.id"))
    # 表示友好提示
    def __repr__(self):
        return "User = %s"%self.name
```

查询所有的用户



查询单个用户

```
In [3]: User.query.get(3)
Out[3]: User = chen
```

查询数据库里面姓唐的用户

```
In [5]: User.query.filter(User.name == 'tang').first()
Out[5]: User = tang

In [6]: User.query.filter(User.name == 'tang').all()
Out[6]: [User = tang]
```

查询所有邮箱是163.com结尾的语句

```
In [7]: User.query.filter(User.email.endswith('163.com')).all()
Out[7]: [User = wang, User = zhou, User = li, User = sun]

In [8]: User.query.filter(User.email.endswith('163.com')).first()
Out[8]: User = wang
```

对id进行排序

```
In [10]: User.query.order_by(User.id.desc()).all()
Out[10]:
[User = sun,
 User = li,
 User = liu,
 User = qian,
 User = wu,
 User = tang,
 User = zhou,
 User = chen,
 User = zhang,
 User = wang]
```

获取到当前数据库的数量

```
In [11]: User.query.count()
Out[11]: 10
```

查询当前姓名为wang的哥们是管理员还是普通用户

```
In [12]: user = User.query.get(1)

In [13]: user
Out[13]: User = wang

In [14]: user.role
Out[14]: Role = admin
```

查询当前角色表里面有多少个是管理员

```
In [15]: role = Role.query.get(1)

In [16]: role
Out[16]: Role = admin

In [17]: role.us
Out[17]: [User = wang, User = zhou, User = qian, User = liu]
```

更新数据库操作

```
In [2]: user = User.query.get(4)

In [3]: user
Out[3]: User = zhou

In [4]: user.name = 'itcast'

In [5]: db.session.commit()
```

删除itcast这条数据

```
In [7]: user = User.query.get(4)

In [8]: user
Out[8]: User = itcast

In [9]: db.session.delete(user)

In [10]: db.seesion.commit()
```

数据库迁移

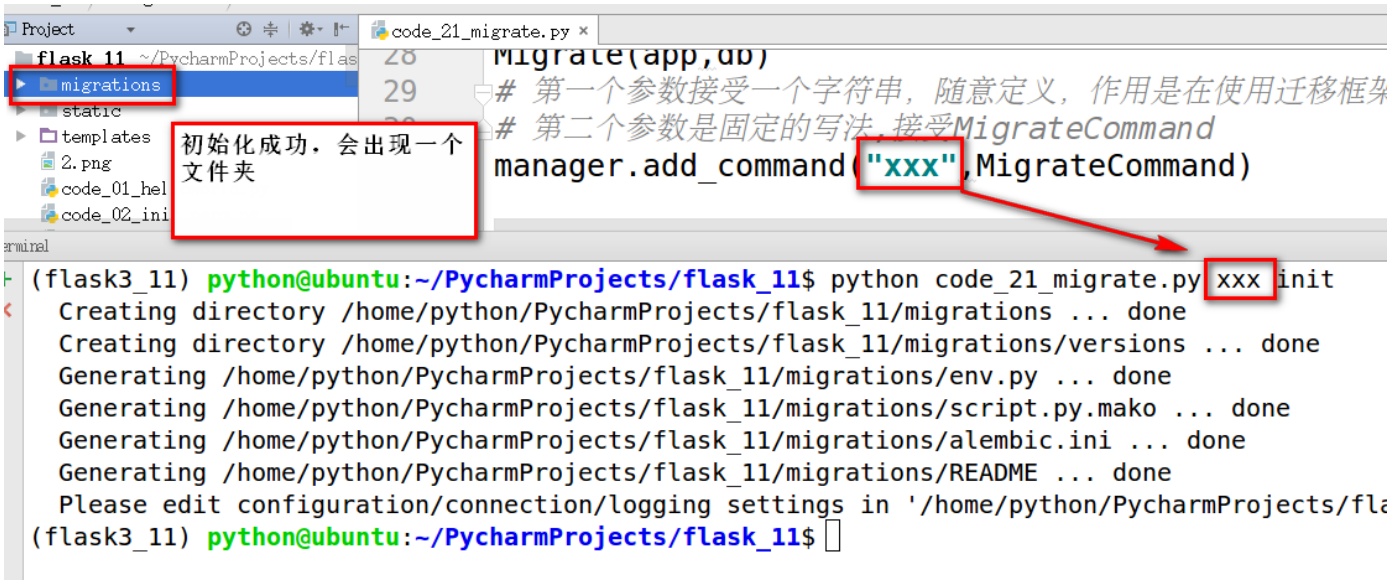
- 在开发过程中，需要修改数据库模型，而且还要在修改之后更新数据库。最直接的方式就是删除旧表，但这样会丢失数据。
- 更好的解决办法是使用数据库迁移框架，它可以追踪数据库模式的变化，然后把变动应用到数据库中。
- 在Flask中可以使用Flask-Migrate扩展，来实现数据迁移。并且集成到Flask-Script中，所有操作通过命令就能完成。
- 为了导出数据库迁移命令，Flask-Migrate提供了一个MigrateCommand类，可以附加到flask-script的manager对象上。

迁移框架的作用：

- 1： 创建数据库里面的表
- 2： 更新字段，不需要删除原来的旧表

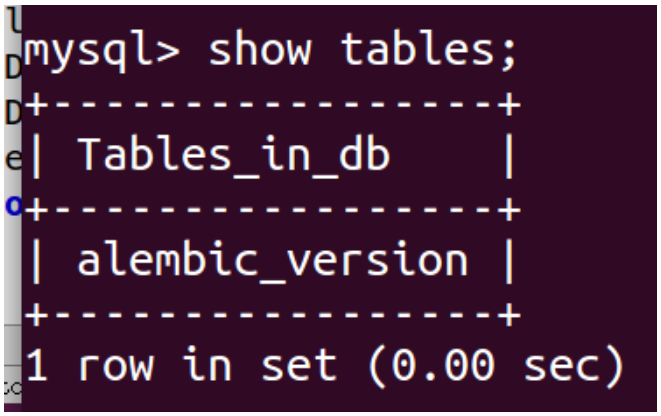
迁移步骤：

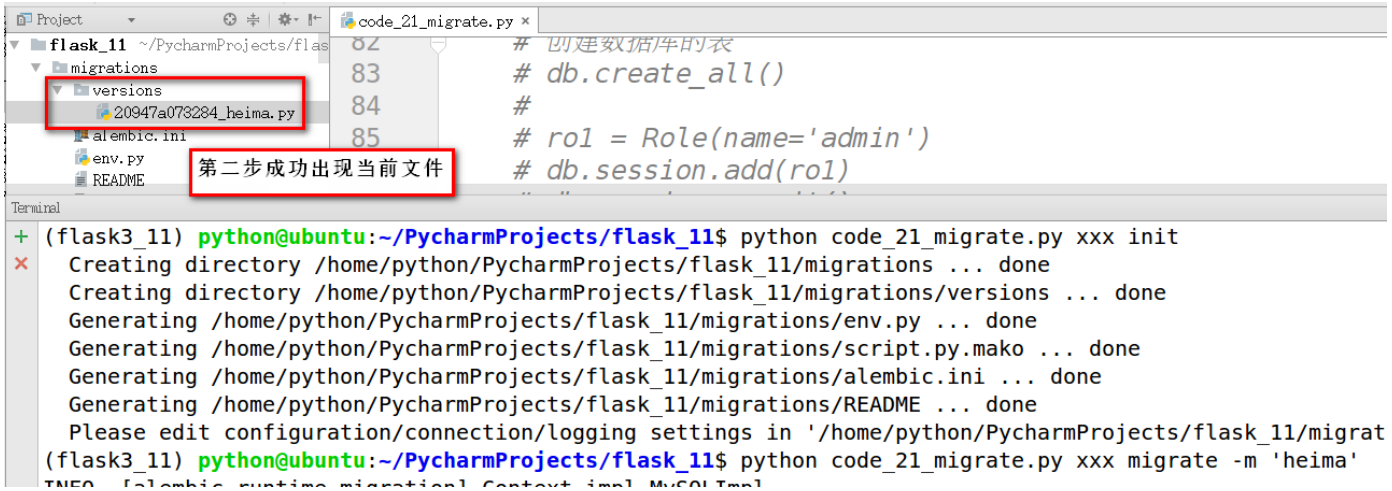
1 第一步：使用如下的命令



第二步:使用如下命令

第二步完成之后，会在数据里面添加一张版本号的表:





第三步：创建数据库的表：

```
(flask3_11) python@ubuntu:~/PycharmProjects/flask_11$ python code_21_migrate.py xxx upgrade
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 20947a073284, heima
```

更新数据库里面的字段：直接重复上面的第二步和第三步

```
(flask3_11) python@ubuntu:~/PycharmProjects/flask_11$ python code_21_migrate.py xxx migrate -m 'add_role_title'
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.autogenerate.compare] Detected added table 'roles'
INFO  [alembic.autogenerate.compare] Detected added table 'users'
  Generating /home/python/PycharmProjects/flask_11/migrations/versions/61bd7d1375c0_add_role_title.py ... done
(flask3_11) python@ubuntu:~/PycharmProjects/flask_11$ python code_21_migrate.py upgrade
usage: code_21_migrate.py [-?] {xxx,shell,runserver} ...
code_21_migrate.py: error: invalid choice: 'upgrade' (choose from 'xxx', 'shell', 'runserver')
(flask3_11) python@ubuntu:~/PycharmProjects/flask_11$ python code_21_migrate.py xxx upgrade
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade 20947a073284 -> 61bd7d1375c0, add_role_title
```

作者和书的后端代码：

```python

from flask import Flask,render_template
from flask_wtf import FlaskForm
from wtforms import StringField,SubmitField
from wtforms.validators import DataRequired
app = Flask(__name__)
app.config["SECRET_KEY"] = "fADFASFAF"
class AuthorBookForm(FlaskForm):
    author_name = StringField(label="作者",validators=[DataRequired("请输入作者的名字")])
    book_name = StringField(label="书名",validators=[DataRequired("请输入书的名字")])
    submit = SubmitField("添加")


@app.route("/")
def index():
    form = AuthorBookForm()
    return render_template("code_22_author_book.html",form = form)

if __name__ == '__main__':
    app.run(debug=True)


```

作者和书前端代码：

```html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="post">
    {{ form.csrf_token }}
```

```
10
11      {{ form.author_name.label }}
12      <p>{{ form.author_name }}</p>
13
14       {{ form.book_name.label }}
15      <p>{{ form.book_name }}</p>
16
17
18       {{ form.submit }}
19  </form>
20
21
22
23  </body>
24  </html>
```

运行效果：