

视频 : <https://www.youtube.com/watch?v=MwZwr5Tvyxo>

源代码: [https://github.com/CoreyMSchafer/code\\_snippets/tree/master/Python/Flask\\_Blog](https://github.com/CoreyMSchafer/code_snippets/tree/master/Python/Flask_Blog)

IDE: Pycharm professional

可以参考: <https://blog.miguelgrinberg.com/category/Flask/page/3>

目标: 一个博客网站, 具有:

1. User Management: Register/Login/ Forgot pwd/ User Profile edit
2. 编辑/ 更新/ 删除 Post

## Step 1: Getting Started

此步内容较少, 直接上全部源码贴图

```
flaskblog.py x
1  from flask import Flask
2  app = Flask(__name__)
3
4
5  @app.route("/")
6  @app.route("/home")
7  def home():
8      return "<h1>Home Page</h1>"
9
10
11  @app.route("/about")
12  def about():
13      return "<h1>About Page</h1>"
14
15
16  if __name__ == '__main__':
17      app.run(debug=True)
18
```

1.app 是个 Flask 类的对象, 当前 module 名 作为初始化参数

2. 此处两个装饰器, 不同地址指向同一 (网页) 资源

3. debug 模式启动

注释:

1. 这个 module `__name__` 提供给 Flask, 以便它去寻找 相应路径下的 `template` 和 `static file`
3. Debug mode 开启后, 修改文件内容后, 可不用重启 web server (IDE 中 stop - run) 保存修改后, 直接刷新网页, 即可看到修改后的效果

## Step 2: Template

如果不用 Template，可以用以下 `return` 直接返回（实际中当然不可能）

```
4
5 @app.route("/")
6 @app.route("/home")
7 def home():
8     return '''<!doctype html>
9         <html>
10
11
12     '''
13
```

实际使用中，通过 引用 `render_template` 解决

```
1 from flask import Flask, render_template, url_for
2 app = Flask(__name__)
3
4 posts = [
5     {
6         'author': 'Corey Schafer',
7         'title': 'Blog Post 1',
8         'content': 'First post content',
9         'date_posted': 'April 20, 2018'
10    },
11    {
12        'author': 'Jane Doe',
13        'title': 'Blog Post 2',
14        'content': 'Second post content',
15        'date_posted': 'April 21, 2018'
16    }
17 ]
18
19
20 @app.route("/")
21 @app.route("/home")
22 def home():
23     return render_template('home.html', posts=posts)
24
```

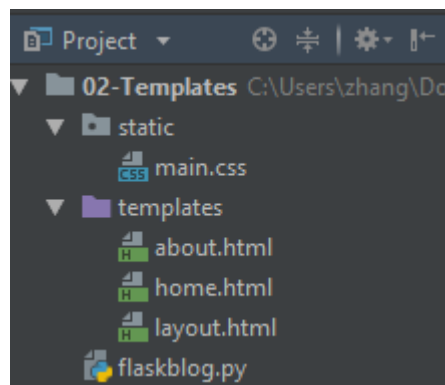
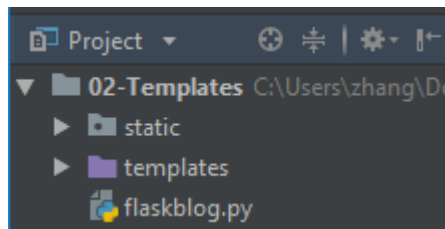
1. posts 参数

注释：

1. 纯静态网页可以 没这个参数

作为值的 `posts`（蓝色）本例子中使用的是一个文件中定义的列表，稍后，实现数据库后，应为数据库中取出的值。

使用 Template 的目录结构图，



Flask 中的 CSS 是要放在 static 目录下的

Flask 使用的 Template engine 是 Jinja

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title></title>
5 </head>
6 <body>
7     {% for post in posts %}
8         <h1>{{ post.title }}</h1>
9         <p>By {{ post.author }} on {{ post.date_posted }}</p>
10        <p>{{ post.content }}</p>
11    {% endfor %}
12 </body>
13 </html>
14
```

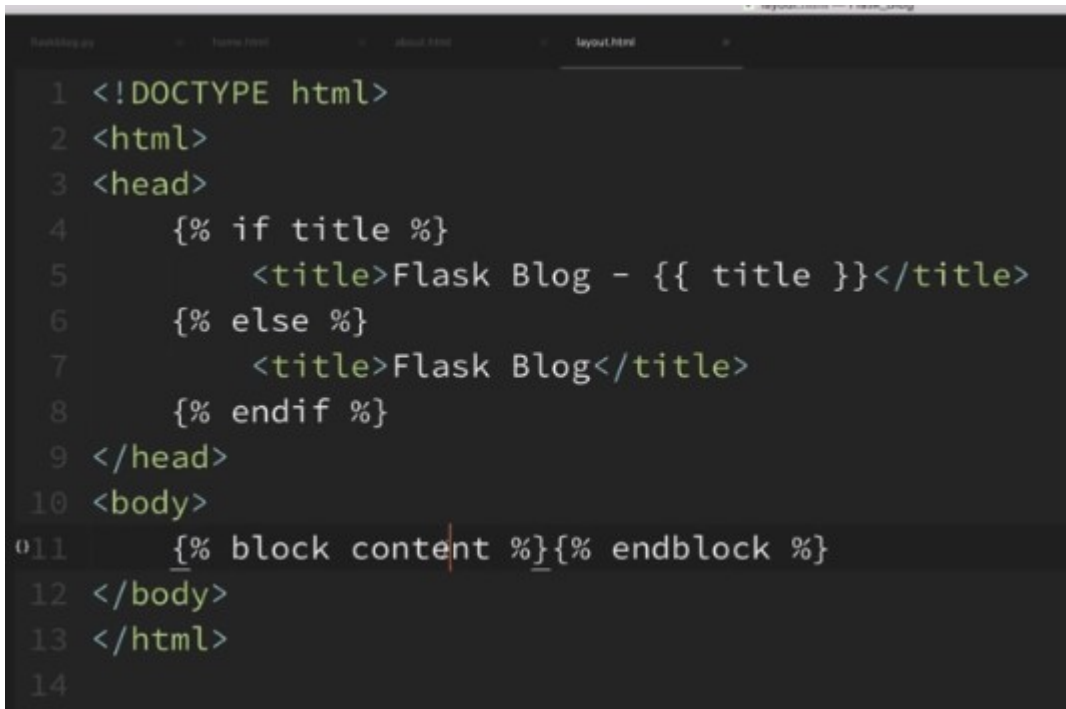
注意上图中，Jinja 的语法习惯

1. {% %} 中为语句块，有 endfor / endif 等 结束标志
2. {{ }} 为引用变量
3. #7 行 中的 posts 是 在 flaskblog.py 中的如下语句，传递的参数

```
def home():
    return render_template('home.html', posts=posts)
```

应提取 所有 HTML 中共同的部分，单独形成形成一个文件，如 layout.html

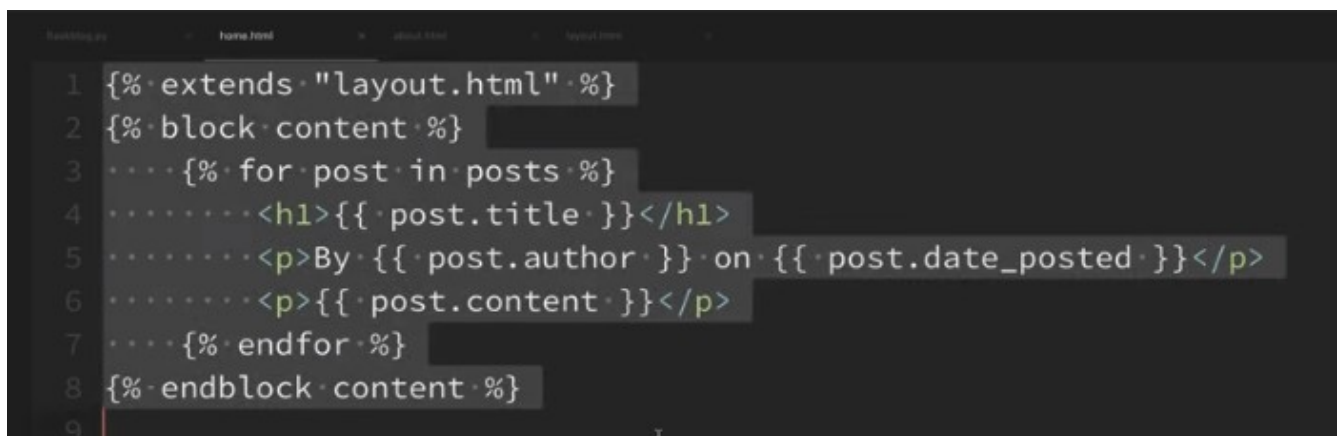
示意性的 layout.html 如下



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     {% if title %}
5         <title>Flask Blog - {{ title }}</title>
6     {% else %}
7         <title>Flask Blog</title>
8     {% endif %}
9 </head>
10 <body>
11     {% block content %}{% endblock %}
12 </body>
13 </html>
14
```

其中 {% block content %} {% endblock content %} 之间 是每个引用它的 html 独特的内容；  
以外 则为共同内容。 content 为该 block 的标识符

提取共同部分后，home.html 内容如下， 注意 {% extends “layout.html” %}



```
1 {% extends "layout.html" %}
2 {% block content %}
3     {% for post in posts %}
4         <h1>{{ post.title }}</h1>
5         <p>By {{ post.author }} on {{ post.date_posted }}</p>
6         <p>{{ post.content }}</p>
7     {% endfor %}
8 {% endblock content %}
9
```

## Bootstrap

在以上示意性，极简的 `layout.html` 中，无脑拷贝 引用 `bootstrap` 的几条语句，如下图

国内国外，要选择不同的 CDN

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <!-- Required meta tags -->
5     <meta charset="utf-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1, sh
7
8     <!-- Bootstrap CSS -->
9     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap
10
```

在结尾处，还要引用 JavaScript

```
19
20     <!-- Optional JavaScript -->
21     <!-- jQuery first, then Popper.js, then Bootstrap JS -->
22     <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" in
23     <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.1
24     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/b
25 </body>
26 </html>
```

源码中的 `python/Flask_Blog/snippets/` 目录中，有些可以直接 `ctrl-C` 的素材，如导航栏等，该段操作在视频的第 25 分钟

`url_for` 关键字 （下段代码来自 `layout.html`, 目的是引用 `/static/main.css`）

```
10
11     <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='main.
12
```

```
href="{{ url_for('static', filename='main.css') }}">
```

## Step 3: Forms and User Input

这部分用到的模块比较多，需要在 Pycharm IDE 环境中安装（File – settings – project interpreter – ‘+’）  
注意视频中的部分模块大小写，于 IDE 环境中的不一致，小心装错。

Package	Version	Latest
Flask	1.0.2	1.0.2
Flask-Bcrypt	0.7.1	0.7.1
Flask-SQLAlchemy	2.3.2	2.3.2
Flask-WTF	0.14.2	0.14.2
Jinja2	2.10	2.10

Flask-WTF : form 相关的 module

实现一个 form， 需要 flaskblog.py/ forms.py/ layout.html/register.html 共同作用，以下分别描述  
**forms.py**

```
forms.py
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, BooleanField
from wtforms.validators import DataRequired, Length, Email, EqualTo

class RegistrationForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email',
                       validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')

class LoginForm(FlaskForm):
    email = StringField('Email',
                       validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me')
    submit = SubmitField('Login')
```

1. 用户定义的表带，都是继承自 FlaskForm
2. StringField 中的 ‘Username’ 和 HTML 中的相应 label 相对应
3. validator 用来检查表单数据有效性，它是一个 list
4. 注意 EqualTo 的参数 是 小写 的 ‘password’

## layout.html

```
<main role="main" class="container">
  <div class="row">
    <div class="col-md-8">
      {% with messages = get_flashed_messages(with_categories=true) %}
      {% endwith %}
      {% block content %}
```

1. {% with message %} 响应 flaskblog.py 中的 **flash**, with\_categories 响应 'success' 等消息类型
2. {% endwith %}, 所有的在 HTML 中的语句块, 都有结束标志

```
{% with messages = get_flashed_messages(with_categories=true) %}
{% if messages %}
  {% for category, message in messages %}
    <div class="alert alert-{{ category }}">
      {{ message }}
    </div>
  {% endfor %}
{% endif %}
{% endwith %}
```

1. python 风格的 for 循环, 显示所有 message
2. register.html 中有结构完全类似的 error 显示 (见本节末页)

flaskblog.py 中 用到 python 自带的工具，生成一个 secret key

```
$ python
Python 3.6.2 |Continuum Analytics, Inc.| (default, Jul 20 2016)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more
>>> import secrets
>>> secrets.token_hex(16)
'5791628bb0b13ce0c676dfde280ba245'
```

```
1 from flask import Flask, render_template, url_for
2 app = Flask(__name__)
3
4 app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba245'
```

flaskblog.py 中 最基本的 页面调用 register （实际代码会增加一些条件判断）

```
34 @app.route("/register")
35 def register():
36     form = RegistrationForm()
37     return render_template('register.html', title='Register', form=form)
```

对比本阶段的完整源码：

1. @app.route("/register") 无 method = ['GET', 'POST'], 则 无法响应 submit form, 产生 "Method not allowed"
2. flash 产生一次性 alert 信息, 'success' 是符合 bootstrap 的信息类型
3. redirect

```
34 @app.route("/register", methods=['GET', 'POST'])
35 def register():
36     form = RegistrationForm()
37     if form.validate_on_submit():
38         flash(f'Account created for {form.username.data}!', 'success')
39         return redirect(url_for('home'))
40     return render_template('register.html', title='Register', form=form)
```



## register.html

```
<div class="content-section">
  <form method="POST" action="">
    </form>
</div>
```

图中，action = "" 意味着 form post 后，并不会跳转到其他页面，而是保留在当前页面

```
<form method="POST" action="">
  {{ form.hidden_tag() }}
</form>
```

cross-site request forgery token  
与 secret key 相关联

```
</div>
<div class="form-group">
  {{ form.email.label(class="form-control-label") }}
  {{ form.email(class="form-control form-control-lg") }}
</div>
<div class="form-group">
  {{ form.password.label(class="form-control-label") }}
  {{ form.password(class="form-control form-control-lg") }}
</div>
<div class="form-group">
  {{ form.confirm_password.label(class="form-control-label") }}
  {{ form.confirm_password(class="form-control form-control-lg") }}
```

form 是一个对象，  
其类定义在 forms.py 中。  
username, email, password  
都是 form 的属性，label 应该是继承而来。

```
<fieldset class="form-group">
  <legend class="border-bottom mb-4">Join Today</legend>
</fieldset>
```

mb4 = margin bottom with value of 4

```
<div class="border-top pt-3">
```

pt-3 = padding top of a value of 3

```
<small class="text-muted ml-2"></small>
```

ml-2 = margin left with value of 2

```

<div class="form-group">
  {{ form.username.label(class="form-control-label") }}

  {% if form.username.errors %}
    {{ form.username(class="form-control form-control-lg is-invalid") }}
    <div class="invalid-feedback">
      {% for error in form.username.errors %}
        <span>{{ error }}</span>
      {% endfor %}
    </div>
  {% else %}
    {{ form.username(class="form-control form-control-lg") }}
  {% endif %}
</div>

```

1. python 风格的 for 循环，显示所有 message
2. layout.html 中有结构完全类似的 message 显示（见本节前页）
3. 对比 if / else 语句块 的异同，对比显示效果

## login.html

该文件中大部分内容，与 register.html 类似，下图中为新内容：

```

<div class="form-check">
  {{ form.remember(class="form-check-input") }}
  {{ form.remember.label(class="form-check-label") }}
</div>

```

remember me

```

<small class="text-muted ml-2">
  <a href="#">Forgot Password?</a>
</small>

```

forget password（当前只是用“#”示意）后面部分会学习到如何完整处理

## Step 4: Database (SQLite)

Flask-SQLAlchemy

2.3.2

2.3.2

Pycharm 中安装以上 package，以支撑 sqlite

```
2 from flask_sqlalchemy import SQLAlchemy
```

(in flaskblog.py)

```
7 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
8 db = SQLAlchemy(app)
```

建一个数据库对象

/// 代表相对路径，即与 .py 同目录

```
12 class User(db.Model):
13     id = db.Column(db.Integer, primary_key=True)
14     username = db.Column(db.String(20), unique=True, nullable=False)
15     email = db.Column(db.String(120), unique=True, nullable=False)
16     image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
17     password = db.Column(db.String(60), nullable=False)
18     posts = db.relationship('Post', backref='author', lazy=True)
19
20     def __repr__(self):
21         return f"User('{self.username}', '{self.email}', '{self.image_file}')
```

用定义类的方法，定义数据库中的表 (in .py)

注意 几个重要参数

unique/nullable/default, 参数传递的是所用函数的标识符，如下图

```
date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
```

db.relationship, 用于定义数据库中表的 (一对多) 对应关系，

“posts = db.relationship...” 相当于 在 post 中新建一列 (backref)，列名为 author，与 User 中内容相对应。 Lazy=true 参数说明 是从数据库中读取 user 信息。

实际上 posts 是从数据库中 (Post) 查询提取 (lazy = True) 而生成的，并非实际的一列

```

24 class Post(db.Model):
25     id = db.Column(db.Integer, primary_key=True)
26     title = db.Column(db.String(100), nullable=False)
27     date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
28     content = db.Column(db.Text, nullable=False)
29     user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
30
31     def __repr__(self):
32         return f"Post('{self.title}', '{self.date_posted}')"

```

`db.ForeignKey('user.id')`

`db.relationship('Post')`

关于上图中， 存在使用不同大小写的问题，此处视频 16 分 30 秒前后解释

Post 指的是一个类，user.id 指的是 指的是是一个数据表以及表的一列，因此分别用大小写。

case so the user model automatically has  
this table name set the lower case

<<<视频中的解释，见左

用户也可以自行设定 table column 的名字，在这里，用系统缺省的小写方式。

（个人理解， db.relationship 指向的是类间的关系，而 db.Column 指向的是数据表中的关系）

有了以上定义之后，可以在 pycharm 的 console 中 生成数据库，如下所示：

```

>>> from flaskblog import db
/Users/coreyschafer/anaconda/envs/flaskblog/lib/python3.6/site-packages/flask_sqlalchemy/extension.py:125: FSADeprecationWarning: SQLAlchemy_TRACK_MODIFICATIONS adds significant overhead
default in the future. Set it to True or False to suppress this warning.
'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
>>> db.create_all()
>>> from flaskblog import User, Post
>>> user_1 = User(username='Corey', email='C@demo.com', password='password')
>>> db.session.add(user_1)
>>> user_2 = User(username='JohnDoe', email='jd@demo.com', password='password')
>>> db.session.add(user_2)
>>> db.session.commit()
>>>

```

1. db.create\_all() 会根据之前设定（本节开头部分） 生成数据库: site.db
2. add 两个 user
3. 用 commit 写入数据库

数据库常用的查询命令如下图所示：

```
>>> User.query.all()
[User('Corey', 'C@demo.com', 'default.jpg'), User('JohnDoe', 'jd@demo.com', 'default.jpg')]
>>> User.query.first()
User('Corey', 'C@demo.com', 'default.jpg')
>>> User.query.filter_by(username='Corey').all()
[User('Corey', 'C@demo.com', 'default.jpg')]
>>> User.query.filter_by(username='Corey').first()
User('Corey', 'C@demo.com', 'default.jpg')
>>> user = User.query.filter_by(username='Corey').first()
>>> user
User('Corey', 'C@demo.com', 'default.jpg')
>>> user.id
1
>>> user = User.query.get(1)
>>> user
User('Corey', 'C@demo.com', 'default.jpg')
>>> user.posts
[]
>>>
```

向数据库中添加两条 post，作者都是 user.id =1

```
>>> user.id
1
>>> post_1 = Post(title='Blog 1', content='First Post Content!', user_id=user.id)
>>> post_2 = Post(title='Blog 2', content='Second Post Content!', user_id=user.id)
>>> db.session.add(post_1)
>>> db.session.add(post_2)
>>> db.session.commit()
>>> user.posts
[Post('Blog 1', '2018-04-29 00:52:12.533764'), Post('Blog 2', '2018-04-29 00:52:12.534451')]
>>>
```

```
>>> user.posts
[Post('Blog 1', '2018-04-29 00:52:12.533764'), Post('Blog 2', '2018-04-29 00:52:12.534451')]
>>> for post in user.posts:
...     print(post.title)
...
Blog 1
Blog 2
>>> post = Post.query.first()
>>> post
Post('Blog 1', '2018-04-29 00:52:12.533764')
>>> post.user_id
1
>>> post.author
User('Corey', 'C@demo.com', 'default.jpg')
>>>
```

```
>>> db.drop_all()
```

清空当前数据库结构， db.create\_all() 重建



## Step 5: Package Structure

```
$ tree
flaskblog
├── __init__.py
├── forms.py
├── models.py
├── routes.py
├── static
│   └── main.css
├── templates
│   ├── about.html
│   ├── home.html
│   ├── layout.html
│   ├── login.html
│   └── register.html
└── run.py
```

<<< 目录结构

此节要点是，避免 **circular import**

为避免此问题，在 flaskblog 目录引入 `__init__.py`

models.py：包含数据库定义

routes.py：定义网页路径指向

run.py

```
run.py
1 from flaskblog import app
2
3 if __name__ == '__main__':
4     app.run(debug=True)
5
```

\_\_init\_\_.py

```
__init__.py
1 from flask import Flask
2 from flask_sqlalchemy import SQLAlchemy
3 from flask_bcrypt import Bcrypt
4
5 app = Flask(__name__)
6 app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba245'
7 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
8 db = SQLAlchemy(app)
9 bcrypt = Bcrypt(app)
10
11 from flaskblog import routes
```

在“底部”

import routes 也是为  
了避免

**circular import**

其他 .py 文件的 import 请参照源码

## Step 6: User Authentication

安装插件

Package	Version	Latest
Flask	1.0.2	1.0.2
Flask-Bcrypt	0.7.1	0.7.1

```
>>> from flask_bcrypt import Bcrypt
>>> bcrypt = Bcrypt()
>>> bcrypt.generate_password_hash('testing')
b'$2b$12$QzKkLJpGs5F7bN7.Jz9Kd.WiFFQ0YrP3u.Ix0cvGPo8G8Cjgtc3ia'
>>> bcrypt.generate_password_hash('testing').decode('utf-8')
'$2b$12$MlH4RmeFVJDo.7ueV3JzhuUDLakFtHdgIOmtdhhB08j2jWWeegbu'
>>> bcrypt.generate_password_hash('testing').decode('utf-8')
'$2b$12$KKcQJFmIpZ6gFSVl9riUZeeixzsAHNr8EtwM4neQprAaidDop685a'
>>> bcrypt.generate_password_hash('testing').decode('utf-8')
'$2b$12$umPSlEMzMWqybzwg52yW.e4ECR4ac89dDV0p8DVcwAeewfgKxXcZC'
>>>
```

每次哈希所得结果，并不相同，但可以用于比较是否都由一个密码（上图中为 “testing”）产生。

```
>>> bcrypt.check_password_hash(hashed_pw, 'password')
False
>>> bcrypt.check_password_hash(hashed_pw, 'testing')
True
>>>
```

实际项目中，在 `__init__.py` 中，使用

```
1 from flask import Flask
2 from flask_sqlalchemy import SQLAlchemy
3 from flask_bcrypt import Bcrypt
4
5 app = Flask(__name__)
6 app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba245'
7 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
8 db = SQLAlchemy(app)
9 bcrypt = Bcrypt(app)
10
11 from flaskblog import routes
```

## Register 模块

在 routes.py 中

```
2 from flaskblog import app, db, bcrypt
```

基础的数据库写入（user name/email 的 unique 校验，放在 forms.py 中实现）

```
35 def register():
36     form = RegistrationForm()
37     if form.validate_on_submit():
38         hashed_password = bcrypt.generate_password_hash(form.password.data).decode('u
39         user = User(username=form.username.data, email=form.email.data, password=hash
40         db.session.add(user)
41         db.session.commit()
42         flash('Your account has been created! You are now able to log in', 'success')
43         return redirect(url_for('login'))
44     return render_template('register.html', title='Register', form=form)
```

校验函数的原型

```
def validate_field(self, field):
    if True:
        raise ValidationError('Validation Message')
```

实际的校验函数（需要以下两个 import）

```
1 from flaskblog.models import User
```

```
3 from wtforms.validators import ValidationError
```

```
def validate_username(self, username):
    user = User.query.filter_by(username=username.data).first()
    if user:
        raise ValidationError('That username is taken. Please choose a
```



## Login 模块

\_\_init\_\_.py

```
4 from flask_login import LoginManager
...
11 login_manager = LoginManager(app)
```

forms.py

```
2 from flaskblog import db, login_manager
3 from flask_login import UserMixin
```

```
11 class User(db.Model, UserMixin):
12     id = db.Column(db.Integer, primary_key=True)
13     username = db.Column(db.String(20), unique=True)
14     email = db.Column(db.String(120), unique=True,
```

(还记得 python 教程里的 mixin 么?)

UserMixin 会实现一些关于用户登录的函数功能，在 routes.py 中我们看这些函数如何起作用的

routes.py 中

```
from flask_login import login_user, current_user, logout_user, login_required
```

在用户已登录的情况下，current\_user 将起到作用，register 函数中，

```
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
```

login 函数

```
50 @app.route("/login", methods=['GET', 'POST'])
51 def login():
52     if current_user.is_authenticated:
53         return redirect(url_for('home'))
54     form = LoginForm()
55     if form.validate_on_submit():
56         user = User.query.filter_by(email=form.email.data).first()
57         if user and bcrypt.check_password_hash(user.password, form.password.data):
58             login_user(user, remember=form.remember.data)
59             next_page = request.args.get('next')
60             return redirect(next_page) if next_page else redirect(url_for('home'))
61         else:
62             flash('Login Unsuccessful. Please check email and password', 'danger')
63     return render_template('login.html', title='Login', form=form)
```

增加 logout 函数

```
56 @app.route("/logout")
57 def logout():
58     logout_user()
59     return redirect(url_for('home'))
```

在 layout.html 中的导航栏中，显示 logout

```
<!-- Navbar Right Side -->
<div class="navbar-nav">
    {% if current_user.is_authenticated %}
    <a class="nav-item nav-link" href="{ url_for('account') }">Account</a>
    <a class="nav-item nav-link" href="{ url_for('logout') }">Logout</a>
    {% else %}
    <a class="nav-item nav-link" href="{ url_for('login') }">Login</a>
    <a class="nav-item nav-link" href="{ url_for('register') }">Register</a>
    {% endif %}
</div>
```

以上导航栏中，还增加了 account

相应的还需要增加 account.html，以及在 routes.py 中增加相应函数

\_\_init\_\_.py 中增加以下

```
11 login_manager = LoginManager(app)
12 login_manager.login_view = 'login'
```

44 分钟起，解释了 next 相关的问题（对照上页底部的源码）可以参照以下链接

<https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-v-user-logins>

Right after the user is logged in by calling Flask-Login's `login_user()` function, the value of the `next` query string argument is obtained. Flask provides a `request` variable that contains all the information that the client sent with the request. In particular, the `request.args` attribute exposes the contents of the query string in a friendly dictionary format. There are actually three possible cases that need to be considered to determine where to redirect after a successful login:

- If the login URL does not have a `next` argument, then the user is redirected to the index page.
- If the login URL includes a `next` argument that is set to a relative path (or in other words, a URL without the domain portion), then the user is redirected to that URL.
- If the login URL includes a `next` argument that is set to a full URL that includes a domain name, then the user is redirected to the index page.

The first and second cases are self-explanatory. The third case is in place to make the application more secure. An attacker could insert a URL to a malicious site in the `next` argument, so the application only redirects when the URL is relative, which ensures that the redirect stays within the same site as the application. To determine if the URL is relative or absolute, I parse it with Werkzeug's `url_parse()` function and then check if the `netloc` component is set or not.

## Step 7: User Account and Profile Picture

routes.py 中

```
89 @app.route("/account", methods=['GET', 'POST'])
90 @login_required
91 def account():
92     form = UpdateAccountForm()
93     if form.validate_on_submit():
94         if form.picture.data:
95             picture_file = save_picture(form.picture.data)
96             current_user.image_file = picture_file
97             current_user.username = form.username.data
98             current_user.email = form.email.data
99             db.session.commit()
100             flash('Your account has been updated!', 'success')
101             return redirect(url_for('account'))
102     elif request.method == 'GET':
103         form.username.data = current_user.username
104         form.email.data = current_user.email
105         image_file = url_for('static', filename='profile_pics/' + current_user.image_file)
106         return render_template('account.html', title='Account',
107                                image_file=image_file, form=form)
```

#106 `render_template` 的参数, `account.html` 外, `title`, `image_file`, `form` 都是可以直接在 `account.html` 中调用的参数。如下

```
5 
```

#101 / #106 的区别

because of something called the post get  
redirect pattern and you

`redirect` / `render_template` 是有区别的, 但区别视频中没讲清楚, 似乎是 `redirect` 会避免“重新提交数据”的警告 (只引发 `get request`)

上传用户头像

forms.py

```
2 from flask_wtf.file import FileField, FileAllowed
```

FileAllowed 实现文件（类型）检查

```
<form method="POST" action="" enctype="multipart/form-data">
```

以上这句