

# 项目架构

# 概述

## 项目架构

尽管在单一脚本中编写小型 Web 程序很方便,但这种方法并不能广泛使用。

程序变复杂后,使用单个大型源码文件会导致很多问题。

不同于大多数其他的 Web 框架,Flask 并不强制要求大型项目使用特定的组织方式。

程序结构的组织方式完全由开发者决定。

## 创建项目架构

```
lzy@embsky:/home/zyli/test/python/flask$ mkdir bkProject
lzy@embsky:/home/zyli/test/python/flask$ mkdir bkProject/app
lzy@embsky:/home/zyli/test/python/flask$ mkdir bkProject/tests
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/config.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/manage.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/app/__init__.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/app/email.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/app/models.py
lzy@embsky:/home/zyli/test/python/flask$ mkdir bkProject/app/templates
lzy@embsky:/home/zyli/test/python/flask$ mkdir bkProject/app/static
lzy@embsky:/home/zyli/test/python/flask$ mkdir bkProject/app/main
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/app/main/__init__.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/app/main/errors.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/app/main/forms.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/app/main/views.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/tests/__init__.py
lzy@embsky:/home/zyli/test/python/flask$ touch bkProject/tests/test.py
```

## 项目架构总览

bkProject/

```
|—— app
|   |—— email.py
|   |—— __init__.py
|   |—— main
|   |   |—— errors.py
|   |   |—— forms.py
|   |   |—— __init__.py
```

```
| | | └── views.py
| | └── models.py
| | └── static
| | └── templates
| └── config.py
| └── manage.py
| └── tests
| | └── __init__.py
| | └── test.py
```

# 具体步骤

## 构建虚拟环境

```
lzy@embsky:/home/zyli/test/python/flask$ cp myFlask/ bkProject/venv -rf
lzy@embsky:/home/zyli/test/python/flask$ cd bkProject/
```

## 构建需求文件

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/pip freeze
>requirements.txt
```

**注意：**如果现在有一个新的虚拟环境需要安装需求的模块，则可以执行`pip install -r requirements.txt`

## 添加项目配置

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim config.py
```

```
class Config:
```

```
    # 表单需要
```

```
    CSRF_ENABLED = True
```

```
    SECRET_KEY = 'www.embsky.com'
```

```
    # 邮件需要
```

```
    MAIL_SERVER = 'smtp.163.com'
```

```
    MAIL_PORT = '25'
```

```
    MAIL_USE_TLS = True
```

```
    MAIL_USERNAME = 'lizhiyong_beyond@163.com'
```

```
    MAIL_PASSWORD = 'xxxxxx'
```

```
    # 数据库需要
```

```
    # 可以设置为True来启用在每个请求中自动提交数据库更改
```

```
    SQLALCHEMY_COMMIT_ON_TEARDOWN = True
```

```
    SQLALCHEMY_TRACK_MODIFICATIONS = True
```

```
    @staticmethod
```

```
    def init_app(app):
```

```
        pass
```

```
class DevelopConfig(Config):
```

```
    DEBUG = True
```

```
    # 数据库需要
```

```
SQLALCHEMY_DATABASE_URI='mysql://root:xxx@localhost/devdb'
```

```
class TestConfig(Config) :
```

```
    TEST = True
```

```
    #数据库需要
```

```
    SQLALCHEMY_DATABASE_URI='mysql://root:xxx@localhost/testdb'
```

```
class ProductionConfig(Config) :
```

```
    # 数据库需要
```

```
    SQLALCHEMY_DATABASE_URI = 'mysql://root:xxx@localhost/embsky'
```

```
config = {
```

```
    'develop':DevelopConfig,
```

```
    'test':TestConfig,
```

```
    'product':ProductionConfig,
```

```
    'default':DevelopConfig,
```

```
}
```

## 编写创建app函数

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/__init__.py
```

```
from flask import render_template
```

```
from flask_bootstrap import Bootstrap
```

```
from flask_mail import Mail, Message
```

```
from flask_moment import Moment
```

```
from flask import Flask
```

```
from flask_sqlalchemy import SQLAlchemy
```

```
from config import config
```

```
bootstrap = Bootstrap()
```

```
mail = Mail()
```

```
moment = Moment()
```

```
db = SQLAlchemy()
```

```
def create_app(config_name) :
```

```
    app = Flask(__name__)
```

```
    app.config.from_object(config[config_name])
```

```
    bootstrap.init_app(app)
```

```
    mail.init_app(app)
```

```
moment.init_app(app)
db.init_app(app)
```

```
//以后在此可以注册蓝本
```

```
return app
```

## 创建蓝本

只有调用create\_app的时候才会创建app，然后在去创建路由，这样的代码不漂亮。  
在这里提同一个更好的办法，创建蓝本。

蓝本可以注册为app的一部分，专门负责处理路由和错误页面问题。

蓝本可以写成一个文件，也可是一个包。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/__init__.py
```

```
from flask import Blueprint
```

```
main = Blueprint('main', __name__)
```

```
from . import views, errors
```

## 在创建app的函数中注册蓝本

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/__init__.py
```

#在create\_app函数中添加

```
from .main import main as main_blueprint
```

```
app.register_blueprint(main_blueprint)
```

## 为蓝本添加视图

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

```
from datetime import datetime
```

```
from flask import render_template, session, redirect, url_for
```

```
from . import main
```

```
from .. import db
```

```
from ..models import Student, Teacher
```

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

```
def index():
```

```
    return render_template('index.html')
```

# 为视图创建模板

基于bootstrap创建一个基模板

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/base.html

```
{% extends "bootstrap/base.html" %}
```

```
{% block title %}
```

EMBSKY博客站

```
{% endblock %}
```

```
{% block navbar %}
```

```
<div class="navbar navbar-inverse" role="navigation">
```

```
<div class="container">
```

```
<div class="navbar-header">
```

```
<button type="button" class="navbar-toggle"
```

```
data-toggle="collapse" data-target=".navbar-collapse">
```

```
<span class="sr-only">Toggle navigation</span>
```

```
<span class="icon-bar"></span>
```

```
<span class="icon-bar"></span>
```

```
<span class="icon-bar"></span>
```

```
</button>
```

```
<a class="navbar-brand" href="/">EMBSKY博客站</a>
```

```
</div>
```

```
<div class="navbar-collapse collapse">
```

```
<ul class="nav navbar-nav">
```

```
<li><a href="/">主页</a></li>
```

```
<li><a href="/">主页</a></li>
```

```
</ul>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
{% endblock %}
```

```
{% block content %}
```

```
<div class="container">
```

```
<div class="page-header">
```

```
{% block page_content %}
```

```
{% endblock %}
```

```
</div>
```

```
</div>
```

```
{% endblock %}
```

## 继承基模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/index.html
{% extends 'base.html' %}
{% block page_content %}
欢迎{{name}}访问
{% endblock %}
```

## 编写出错处理方法

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/errors.py
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/templates/404.html
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/templates/500.html
```

```
from flask import render_template
from . import main
```

```
@main.app_errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404
```

```
@main.app_errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500
```

## 编写启动脚本

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim manager.py
import pymysql
pymysql.install_as_MySQLdb()
from app import create_app, db
```

```
app = create_app('develop')
```

#在此可以添加测试代码命令



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

# 测试单元

## 添加数据模型

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

```
from app import db
```

```
class Student(db.Model):
```

```
    __tablename__ = 'students'
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    name = db.Column(db.String(32))
```

```
    teacher_id = db.Column(db.Integer, db.ForeignKey('teachers.id'))
```

```
class Teacher(db.Model):
```

```
    __tablename__ = 'teachers'
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    name = db.Column(db.String(32))
```

```
    teachers = db.relationship('Student', backref='teacher')
```

## 编写单元测试代码

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim tests/test.py

```
import unittest
```

```
from flask import current_app
```

```
from app import create_app, db
```

```
class TestClass(unittest.TestCase):
```

```
    #在测试每一个测试函数运行之前都会运行setUp
```

```
    def setUp(self):
```

```
        self.app = create_app('test')
```

```
        self.app_content = self.app.app_context()
```

```
        self.app_content.push()
```

```
        db.create_all()
```

```
    #在测试每一个测试函数结束后都会运行tearDown
```

```
    def tearDown(self):
```

```
        db.session.remove()
```

```
        db.drop_all()
```

```
        self.app_content.pop()
```

#下面定义了两个测试函数

```
def test_app_exists(self):  
    self.assertFalse(current_app is None)  
def test_app_is_test(self):  
    self.assertTrue(current_app.config['TEST'])
```

## 在启动脚本中添加测试命令

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim manager.py

添加如下:

```
from app.models import Student, Teacher  
from flask_script import Manager, Shell
```

#用于执行命令

```
manager = Manager(app)  
def make_shell_context():  
    return dict(app=app, db=db, Student=Student, Teacher=Teacher)  
manager.add_command('shell', Shell(make_context=make_shell_context))
```

```
@manager.command
```

```
def test():  
    import unittest  
    t = unittest.TestLoader().discover('tests')  
    unittest.TextTestRunner().run(t)
```

## 开始测试

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3  
manage.py test

# 构建数据库

## 创建数据库

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim manager.py

添加如下：

```
from flask_migrate import Migrate, MigrateCommand
```

#用于数据迁移

```
migrate = Migrate(app, db)
```

```
manager.add_command('db', MigrateCommand)
```

注意：确保devdb数据存在

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python manage.py  
db init

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python manage.py  
db migrate -m 'init'

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python manage.py  
db upgrade

# 用户认证

# 密码散列

## 使用 Werkzeug 实现密码散列

### 添加User模型

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

```
from werkzeug.security import generate_password_hash, check_password_hash
```

```
class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64))
    password_hash = db.Column(db.String(128))
    def __str__(self):
        return self.name + " " + str(self.id)

    @property
    def password(self):
        raise AttributeError('password can not read')
    @password.setter
    def password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```

### 更新数据库

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3  
manage.py db migrate -m "add user model"

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3  
manage.py db upgrade

### 测试

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3  
manage.py shell

```
>>> from app.models import User
```

```

>>> u1 = User(name='zhangsan')
>>> u1.password = '123456'
>>> u1.check_password('123456')
True
>>> u1.check_password('1234567')
False
>>> u1.password
AttributeError('password can not read,')
>>> u1.password_hash
'pbkdf2:sha256:50000$DM8zdClr$5cf7d09a86fc1f86e4a05bcb410fc98aee65216e76922dc7

```

## 添加测试单元

```

from app.models import User
class TestUserModel(unittest.TestCase):
    def test_password_set(self):
        u1 = User(name='zhangsan')
        u1.password = '123456'
        self.assertTrue(u1.password_hash is not None)
    def test_password_get(self):
        u1 = User(name='zhangsan')
        with self.assertRaises(AttributeError):
            u1.password
    def test_check_password(self):
        u1 = User(name='zhangsan')
        u1.password = '123456'
        self.assertTrue(u1.check_password('123456'))
    def test_password_random(self):
        u1 = User(name='zhangsan')
        u2 = User(name='lisi')
        u1.password = '123456'
        u2.password = '123456'
        self.assertTrue(u1.password_hash != u2.password_hash)

```

## 利用测试单元测试

```

lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3
manage.py test

```





# 认证蓝本

## 使用蓝本提供登录功能

### 创建auth蓝本

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ mkdir app/auth
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/auth/__init__.py
from flask import Blueprint
auth = Blueprint('auth', __name__)
from . import views
```

### 为auth蓝本创建视图

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ mkdir app/auth/views.py
from flask import render_template
from . import auth
```

```
@auth.route('/login')
def login():
    return render_template('auth/login.html');
```

### 为视图创建模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ mkdir app/templates/auth
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/templates/auth/login.html
{% extends 'base.html' %}
{% block page_content %}
{% for m in get_flashed_messages() %}
    {{m}}
{% endfor %}
{% endblock %}
```

### 注册蓝本到app

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/__init__.py
from .auth import auth as auth_blueprint
#在create_app函数中添加:
app.register_blueprint(auth_blueprint, url_prefix='/auth')
```

# Flask-Login认证

## 什么是Flask-Login?

Flask-Login 是个非常有用的小型扩展,专门用来管理用户认证系统中的认证状态. 不依赖特定的认证机制。

Flask-Login提供Flask的用户会话管理。

它处理登陆、登出任务，并且记住你的会话一段时间。

它能做到的：

- 存储在会话中的活跃用户，让你很轻松的登入登出
- 可以让你限制未登陆的用户访问某些页面
- 处理“记住我”的功能
- 帮你保护你的会话cookie不被小偷偷走
- 很轻松的集成到Flask-Principal或其他授权扩展

它不会做的：

- 强制你使用某个数据库或者存储，你必须完全负责用户的加载
- 限制你使用用户名和密码，OpenID，或任何其它的认证方法
- 处理你登入或登出之外的权限
- 处理用户注册或账户恢复

## Flask-Login原理

### `login_user(user)`

调用user中的`get_id`方法获取用户ID，写到session的`user_id`里面，同时设置`_request_ctx_stack.top.user = user`。

### `@login_required`

判断当前用户是否已经登录。需要调用user中的`is_authenticated()` 方法来判断用户是否登录。

### `logout_user(user)`

清掉session的相关字段，然后调用`reload_user`，然后把这个用户设置为匿名用户。需要调用由[`@login\_manager.user\_loader`](#)修饰的函数来重新加载user。

## 安装Flask-Login

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/pip3 install flask-login
```

## 修该User模型

修改模型，使User模型支持Flask\_Login

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

注意：红色部分为新添加行，其他不变

```
from flask_login import UserMixin
class User(db.Model, UserMixin):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64))
    password_hash = db.Column(db.String(128))
    email = db.Column(db.String(64))
```

注意：UserMixin中实现了4个我们需要的方法

点击这里	点击这里
is_authenticated()	如果用户已经登录,必须返回 True ,否则返回 False。在@login_required中
is_active()	如果这是一个活动用户且通过验证，账户也已激活，未被停用，也不符合
is_anonymous()	如果是一个匿名用户，返回 True ,真实用户应返回 False 。
get_id()	必须返回用户的唯一标识符,使用 Unicode 编码字符串。在login_user中

注意：由于修改了模型，所以要更新数据库

## 注册Flask\_Login到APP

构建login\_manager来管理用户

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/\_\_init\_\_.py

构建login\_manager对象：

```
from flask_login import LoginManager
login_manager = LoginManager()
login_manager.session_protection = 'strong'
#指定登录路由，在蓝本auth中的login，访问登录后才能访问的页面时候重定向的
auth.login
login_manager.login_view = 'auth.login'
```

在create\_app函数中添加：

```
login_manager.init_app(app)
```

注意：

session\_protection 属性可以设为 None 、 'basic' 或 'strong' ,以提供不同的安全等级防止用户会话遭篡改。

设为 'strong' 时,Flask-Login 会记录客户端 IP地址和浏览器的用户代理信息,如果发现异动

就登出用户。

## 在模型中再添加一个回调函数

加载用户的回调函数接收以 Unicode 字符串形式表示的用户标识符。

如果能找到用户,这个函数必须返回用户对象,否则应该返回 None。

这个方法在logout\_user中调用,获取用户后会该用户设置为匿名用户。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

```
from . import login_manager
@login_manager.user_loader
def load_user(id):
    return User.query.get(int(id))
```

## 使用Flask-Login保护路由

路由经过保护后只能由登录的用户访问。

```
from flask_login import login_required
```

*#在路由函数上加修饰符*

```
@login_required
```

# 用户登录

## 构建表单类

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ touch app/auth/forms.py

```
from flask_wtf import FlaskForm
```

```
from wtforms import StringField, PasswordField, BooleanField, SubmitField
```

```
from wtforms.validators import DataRequired, Length, Email
```

```
class LoginForm(FlaskForm):
```

```
    email = StringField('Email', validators = [DataRequired(), Length(1,64), Email()])
```

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

```
    remember_me = BooleanField('Remember_Me')
```

```
    submit = SubmitField('Login')
```

## 修改视图中路由函数

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ mkdir app/auth/views.py

添加如下:

```
from .forms import LoginForm
```

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

```
def login():
```

```
    form = LoginForm()
```

```
    return render_template('auth/login.html', form = form);
```

## 为视图编写模板

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/auth/

login.html

**#在这里使用bootstrap渲染表单**

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

```
{% block page_content %}
```

```
{% for m in get_flashed_messages() %}
```

```
    {{m}}
```

```
{% endfor %}
```

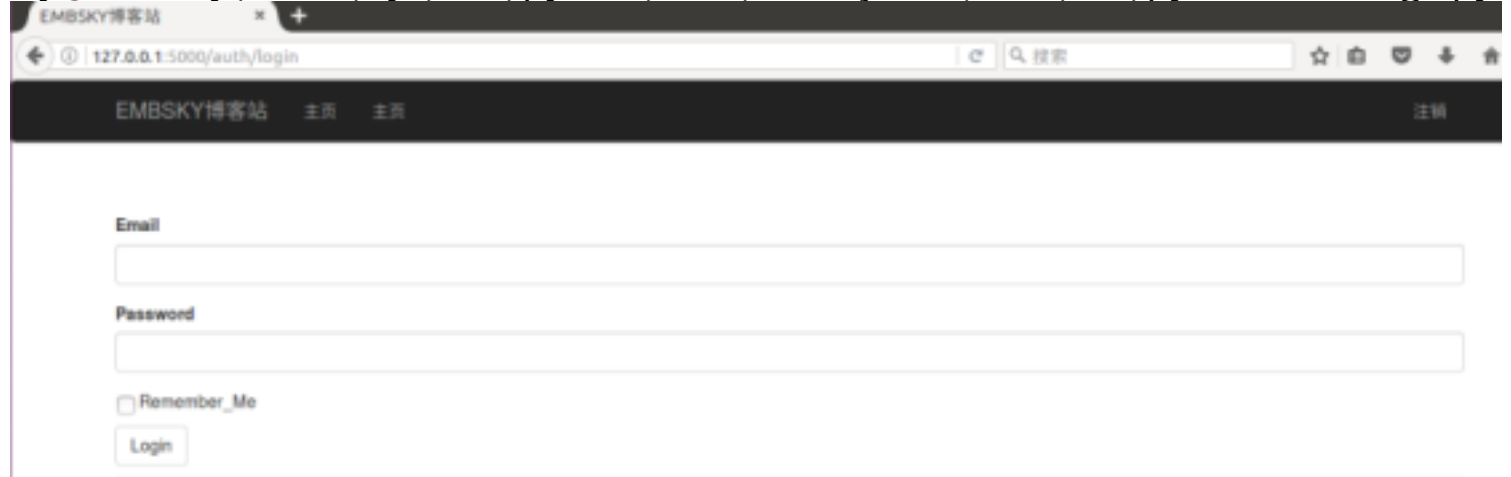
```
{% import "bootstrap/wtf.html" as wtf %}
```

```
{{ wtf.quick_form(form) }}
```

```
{% endblock %}
```

## 测试效果

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3 manage.py



## 用户登录

修改视图支持用户登录

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ mkdir app/auth/views.py  
添加如下：

```
from flask import redirect, flash, url_for, request
```

```
from ..models import User
```

```
from flask_login import login_user
```

```
from flask import session
```

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

```
def login() :
```

```
    form = LoginForm()
```

```
    if form.validate_on_submit() :
```

```
        #获取用户模型对象
```

```
        user = User.query.filter_by(email=form.email.data).first();
```

```
        #判断用户是否存在，若存在则判断密码是否正确
```

```
        if (user is not None and user.check_password(form.password.data)) :
```

```
            #使用login_user记住用户的登录状态
```

```
            login_user(user, form.remember_me.data)
```

```
            session['name'] = form.email.data
```

```
            return redirect(url_for('main.index'))
```

```
        flash('name or password error')
```

```
    return render_template('auth/login.html', form = form);
```

## login\_user函数

login\_user() 函数的参数是要登录的用户,以及可选的“记住我”布尔值。

如果第二个参数值为 False ,那么关闭浏览器后用户会话就过期了,所以下次用户访问时要重新登录。

如果第二个参数值为 True ,那么会在用户浏览器中写入一个长期有效的 cookie,使用这个 cookie 可以复现用户会话。

# 用户注销

## 修改基模板

让基模板能够显示用户登录的状态

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/base.html  
添加如下:

```
<ul class="nav navbar-nav navbar-right">
    {% if current_user.is_authenticated %}
    <li><a href="{{ url_for('auth.logout') }}">注销</a></li>
    {% else %}
    <li><a href="{{ url_for('auth.logout') }}">登录</a></li>
    {% endif %}
</ul>
```

判断条件中的变量 `current_user` 由 Flask-Login 定义,且在视图函数和模板中自动可用。  
这个变量的值是当前登录的用户,如果用户尚未登录,则是一个匿名用户代理对象。  
如果是匿名用户, `is_authenticated` 返回 `False`。  
所以这个方法可用来判断当前用户是否已经登录

## 修改视图函数

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ mkdir app/auth/views.py  
添加如下:

```
from flask_login import logout_user
```

```
@auth.route('/logout')
```

```
@login_required
```

```
def logout():
```

```
    logout_user()
```

```
    #flash('Logout Ok')
```

```
    session['name'] = '游客'
```

```
    return redirect(url_for('.login'))
```

## 测试

手动注册一个用户

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3  
manage.py shell

```
>>> from app.models import User
```

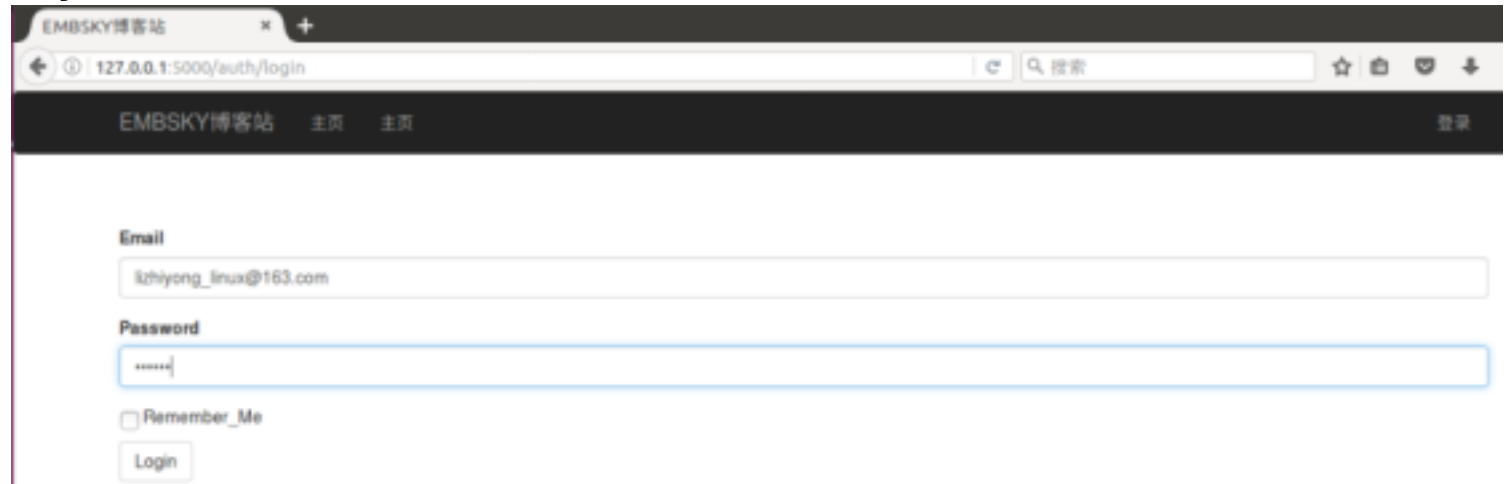
```
>>> u1 = User(email='lizhiyong_linux@163.com', name='lizhiyong')
```



```
>>> u1.password = '123456'
>>> db.session.add(u1)
>>> db.session.commit()
```

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3
manage.py
```

登录



EMBSKY博客站

127.0.0.1:5000/auth/login

EMBSKY博客站 主页 主页 登录

Email

lizhiyong\_linux@163.com

Password

\*\*\*\*\*

☐ Remember\_Me

Login

登录成功



EMBSKY博客站

127.0.0.1:5000

EMBSKY博客站 主页 主页 注册

欢迎lizhiyong\_linux@163.com访问

# 用户注册

在基模板base.html中添加一个我要注册的链接

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/base.html

```
{% if current_user.is_authenticated %}
<li><a href="{{ url_for('auth.logout') }}">注销</a></li>
{% else %}
<li><a href="{{ url_for('auth.login') }}">登录</a></li>
{% endif %}
<li><a href="{{ url_for('auth.register') }}">我要注册</a></li>
```

在auth蓝本中添加注册表单

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/forms.py

```
from wtforms.validators import Regexp,EqualTo
from wtforms import ValidationError
from ..models import User

class RegisterForm(FlaskForm):
    email = StringField('Email', validators = [Required(), Length(1,64), Email()])
    name = StringField('Name', validators = [Required(), Length(6, 64), Regexp('^[a-zA-Z]\w*$', 0, 'Name Must Be Char And Num')])
    password = PasswordField('Password', validators = [Required()])
    password1 = PasswordField('Password Again', validators =
[Required(),EqualTo('password', message='Password Not Match')])
    submit = SubmitField('Register')

    def validate_email(self, field):
        if (User.query.filter_by(email=field.data).first() is not None):
            raise ValidationError('Email Is Already Registered')
    def validate_name(self, field):
        if (User.query.filter_by(name=field.data).first() is not None):
            raise ValidationError('User Is name Already Used')
```

在蓝本auth的视图中添加register路由

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/views.py

```
from .forms import RegisterForm
from .. import db
```

```

@auth.route('/register', methods = ['GET', 'POST'])
def register():
    form = RegisterForm()
    if (form.validate_on_submit()):
        user = User();
        user.password = form.password.data
        user.name = form.name.data
        user.email = form.email.data
        db.session.add(user)
        db.session.commit()
        flash('Now Login')
        return redirect(url_for('.login'))
    return render_template('auth/register.html', form = form)

```

在蓝本auth的视图中添加异常处理方法

因为如果用户注册时邮箱冲突的话会抛出AttributeError异常

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/views.py

```

@auth.errorhandler(AttributeError)
def error_handler(error):
    flash('Email Is Alreay In Use!')
    return redirect(url_for('auth.register'))

```

为视图添加模板

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ touch app/templates/auth/register.html

```

{% import 'bootstrap/wtf.html' as wtf %}
{% extends 'base.html' %}
{% block page_content %}
{% for m in get_flashed_messages() %}
    {{m}}
{% endfor %}
{{ wtf.quick_form(form) }}
{% endblock %}

```

测试

EMBSKY博客站

127.0.0.1:5000/auth/register

搜索

☆

📁

🔖

⬇️

🔍

EMBSKY博客站

主页

文章

登录

我要注册

Email

Name

Password

Password Again

Register

# 用户邮箱验证

# 基本验证

## 通过邮箱确认账户

需要实现用户注册后发送一个确认邮件。

发给不同用户的确定链接应该没有规律的，防止反推。

因此，账户确认过程中，需要用户点击一个包含确认令牌的特殊 URL 链接。

### 使用 itsdangerous 生成确认令牌

itsdangerous 提供了多种生成令牌的方法。

其中, TimedJSONWebSignatureSerializer 类生成具有过期时间的 JSON Web 签名(JSON Web Signatures,JWS)。

**#导入JWS**

```
>>> from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
```

**#生成令牌**

```
>>> s = Serializer(app.config['SECRET_KEY'], expires_in = 168)
```

**#生成加密串**

```
>>> token = s.dumps({'confirm':100})
```

**#解密（如果超时则会发生异常）**

```
>>> data = s.loads(token)
```

```
>>> data
```

```
{'confirm': 10}
```

### 添加邮件支持

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/email.py
```

```
from flask_mail import Mail, Message
```

```
from . import mail
```

```
from flask import current_app
```

**#编写一个同步发送邮件的函数**

```
def send_email(subject, recvers, html) :
```

```
    msg = Message(subject, sender = current_app.config['MAIL_USERNAME'],  
recipients = recvers)
```

```
    msg.html = html
```

```
    with current_app.app_context() :
```

```
        mail.send(msg)
```

#编写一个异步发送邮件的函数

```
from threading import Thread
```

```
def thread_fun(app, message):
```

```
    with app.app_context():
```

```
        mail.send(message)
```

```
def send_async_email(subject, recvers, body, html):
```

```
    message = Message(subject=subject, recipients=recvers, body=body, html=html)
```

```
    message.sender = current_app.config['MAIL_USERNAME']
```

```
    app = current_app._get_current_object()
```

```
    thread = Thread(target=thread_fun, args=[app, message])
```

```
    thread.start()
```

**注意：**current\_app只是一个代理app，current\_app.context()在不同的线程中获取到的上下文是不一样的，因此必须把真正的app对象传递给子线程，使用current\_app.\_get\_current\_object()获取真正的app的对象。

## 修改User用户模型

在用户模型中添加生成token和验证token的功能

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

导入包：

```
from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
```

```
from flask import current_app
```

在User中添加：

```
confirmed = db.Column(db.Boolean, default = False)
```

```
def generate_confirm_token(self, expiration = 7200):
```

```
    s = Serializer(current_app.config['SECRET_KEY'], expiration)
```

```
    return s.dumps({'confirm':self.id})
```

```
def confirm(self, token):
```

```
    s = Serializer(current_app.config['SECRET_KEY'])
```

```
    try:
```

```
        data = s.loads(token)
```

```
    except:
```

```
        return False
```

```
    if (data.get('confirm') != self.id):
```

```
        return False
```

```
    self.confirmed = True
```

```
    db.session.add(self)
```

```
    db.session.commit()
```

**return True**

**注意：**修改数据模型后要更新数据库

## 修改auth.views视图中路由register

在用户注册后发送给邮件给客户邮件

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/views.py

导入包：

**from ..email import send\_email**

在register函数中添加（在commit之后）：

```
token = user.generate_confirm_token()
```

```
html = render_template('email/register.html', token=token, name=user.name)
```

```
send_email('EMBSKY注册确认邮件', [user.email], html)
```

```
flash('请登录邮箱验证注册') #在flash中不能使用user（除非重定向的页面是当前页面）
```

添加验证链接的路由：

```
@auth.route('/confirm', methods=['GET', 'POST'])
```

```
def confirm():
```

```
    return render_template('index.html')
```

## 为邮件创建模板

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ mkdir app/templates/email

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ touch app/templates/email/

register.html

Dear {{name}}: <br>

你好！感谢您在EMBSKY注册，点击以下链接完成用户验证： <br>

<a href={{url\_for('auth.confirm', token=token, \_external=True)}}>点击这里完成验证<br></a>

注意：此链接2小时之内有效！ <br>

EMBSKY开发团队。

**注意：**token的内容会以GET请求的方式追加在url\_for生成的url后边。

**注意：**因为url\_for生成的是相对url，\_external=True代表生成完整的url。

测试：邮件发送成功，点击链接能够发送get请求。

## 处理用户验证

当用户注册完帐号后，先登录，然后提示用户需要邮箱验证，点击链接验证成功即可访问。

修该auth的视图，增加before\_request等函数



lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/views.py

```
from flask_login import current_user
#调用这个路由之前用户应该是要先登录。
@auth.route('/confirm', methods=['GET', 'POST'])
@login_required
def confirm():
    #如果用户已经验证，则重定向
    if (current_user.confirmed):
        return redirect(url_for('main.index'))
    #开始验证用户
    if current_user.confirm(request.args.get('token')):
        flash('验证成功')
    else:
        flash('验证失败') #这里应该重定向到发送邮件页面
    return render_template('index.html', name=current_user.name)
```

截获没有验证的用户

在这里定一个钩子函数。

使用`auth.before_app_request`修饰能够截获全局的请求。

也就是说在访问任何本站的url之前都会先执行`before_request`函数。

```
@auth.before_app_request
def before_request():
    #这里只处理已登录但没有验证的用户
    if (current_user.is_authenticated \
        and not current_user.confirmed \
        and request.endpoint[:5] != 'auth.' \
        and request.endpoint != 'static'):
        #endpoint就是你要访问的路由
        #如果用户登录了，并且用户没有验证，并且访问的不是auth中的路由，也不是静态页面
        #截获，重定向到没有验证路由
        return redirect(url_for('auth.unconfirmed'))
    #其他的请求都正常处理，不截获
```

注意：如果只想截获auth蓝本的请求，则把`before_app_request`改为`before_request`

```
@auth.route('/unconfirmed')
def unconfirmed():
    #如果是匿名用户，或者用户已经验证，则重定向到主页
```

```
if (current_user.is_anonymous or current_user.confirmed) :  
    return redirect(url_for('main.index'))  
#否则渲染一个页面告知用户需要验证  
return render_template('auth/unconfirmed.html', name=current_user.name)
```

## 创建未验证模板

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/auth/unconfirmed.html

```
{% extends 'base.html' %}  
{% block page_content %}
```

您好{{name}}!<br>

您的帐号还没有验证，您的邮箱有一封由本站发送的邮件。<br>  
邮件中包含一个链接，点击链接激活您的帐号！<br><br>

EMBSKY开发团队！<br>

感谢您的支持

```
{% endblock %}
```

测试：



# 超时重发邮件

修改auth/unconfirmed.html模板

支持渲染超时重发邮件和提示用户验证两种界面。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/auth/unconfirmed.html
```

```
{% extends 'base.html' %}
{% block page_content %}
Dear {{ current_user.name }}!<br>
    您的帐号还没有经过认证! <br>
    {% if timeout %}
    您的验证已经超时! 点击下面链接重新发送验证邮件。 <br>
    <a href={{ url_for('auth.resend_email') }}>点击这里重新发送验证邮件</a>
    {% else %}
    您的邮箱中有一封认证邮件! 请查收! <br>
    邮箱内认证链接的有效时间是两小时。 <br>
    如果在10分钟内收不到邮件, 请点击
    <a href={{ url_for('auth.resend_email') }}>这里</a>
    重新发送验证邮件<br>
    {% endif %}
    <h2>EMBSKY开发团队</h2><br>
{% endblock %}
```

修改unconfirmed路由函数

#渲染提示用户需要验证

```
return render_template('auth/unconfirmed.html', name=current_user.name,
timeout=False)
```

添加重新发送邮件路由

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/auth/views.py
```

```
@auth.route('/resend_email')
```

```
@login_required
```

```
def resend_email():
```

```
    token = current_user.generate_confirm_token()
```

```
    html = render_template('email/register.html', token=token,
```

```
name=current_user.name)
```

```
    send_email('EMBSKY注册确认邮件', [current_user.email], html)
```

```
return redirect(url_for('.login'))
```

## 修改auth.views.confirm

如果用户验证不成功提示重新发送邮件。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/views.py

```
def confirm():  
    if (current_user.confirmed):  
        return redirect(url_for('main.index'))  
    #验证成功则渲染主页  
    if current_user.confirm(request.args.get('token')):  
        return redirect(url_for('main.index'))  
    #否则渲染没有验证成功页面  
    else:  
        return render_template('auth/unconfirmed.html', name=current_user.name,  
timeout=True)
```

# 显示登录用户

在导航栏显示登录的用户名

修改基模板，添加红色行

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/base.html

```
{% if current_user.is_authenticated %}
```

```
<li><a href="">{{current_user.name}}</a></li>
```

```
<li><a href="{{ url_for('auth.logout') }}">注销</a></li>
```

```
{% else %}
```

```
<li><a href="{{ url_for('auth.login') }}">登录</a></li>
```

```
{% endif %}
```

```
<li><a href="{{ url_for('auth.register') }}">我要注册</a></li>
```

# 用户角色

# 角色分类

每一个用户都属于某一种角色，每种角色都有各自的权限

用户权限分为六种：

权限	权限编码
浏览博客	0x00
关注其他用户	0x01
发表评论	0x02
发表博客	0x04
管理评论	0x08
管理员权限	0x80

定一个权限类

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/models.py
```

```
class Permission():
```

```
    FOLLOW = 0X01
```

```
    COMMENT = 0X02
```

```
    WRITE = 0X04
```

```
    MODE_COMMENT = 0X08
```

```
    ADMIN = 0X80
```

用户角色分为四类：

角色	权限	描述
未登录的用户	0x00	只能浏览别人的博客
普通用户	0x01   0x02   0x04	能发评论、发博客、关注其他博主
协管员	0x01   0x02   0x04   0x08	能发评论、发博客、关注其他博主、管理博客
管理员	0xff	所有权限

创建角色模型

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/models.py
```

```
class Role(db.Model):
```

```
__tablename__ = 'roles'  
id = db.Column(db.Integer, primary_key = True)  
name = db.Column(db.String(64), unique = True)  
default = db.Column(db.Boolean, default = False)  
permissions = db.Column(db.Integer)  
users = db.relationship('User', backref = 'role')
```

## 在User模型中添加外键

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/models.py  
role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))
```

一个用户属于一个角色，一个角色可以对应各用户

**注意：**修改了数据模型要更新数据库

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3  
manage.py db upgrade  
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3  
manage.py db migrate -m 'add Role model'  
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3  
manage.py db upgrade
```



# 创建角色

在Role模型中添加一个静态方法

这个方法用来创建角色。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/models.py
```

```
@staticmethod
```

```
def create_roles():
```

```
    roles = {
```

```
        'user':[Permission.FOLLOW | Permission.COMMENT | Permission.WRITE, True],
```

```
        'moderator':[Permission.FOLLOW | Permission.COMMENT | Permission.WRITE |
```

```
Permission.MODE_COMMENT, False],
```

```
        'admin':[0xff, False]
```

```
    }
```

```
    for r in roles:
```

```
        role = Role.query.filter_by(name=r).first()
```

```
        if (role is None):
```

```
            role = Role()
```

```
            role.name = r
```

```
            role.default = roles[r][1]
```

```
            role.permissions = roles[r][0]
```

```
            db.session.add(role)
```

```
            db.session.commit()
```

## 创建角色

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3
```

```
manage.py shell
```

```
>>> from app.models import Role
```

```
>>> Role.create_roles()
```

```
>>> Role.query.all()
```

```
[<app.models.Role object at 0x7f2bee2c0240>, <app.models.Role object at  
0x7f2bee2c01d0>, <app.models.Role object at 0x7f2bee2c0320>]
```

```
>>>
```

# 分配角色

在创建用户的时候需要制定用户的角色

修改User模型

添加一个构造函数，在构建User模型对象的时候可以传入email，然后构造函数可以根据email来判断角色

比如:email等于MAIL\_USERNAME的话就认为admin角色，否则就是user角色

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

添加代码如下：

```
def __init__(self, **kwargs):
    #调用父类的构造函数
    super(User, self).__init__(**kwargs) #这个地方不太明白 调用父类的构造函数怎么就把子类对象的成员给初始化了
    #如果初始化的时候没有制定role
    if (self.role is None):
        if (self.email == current_app.config['MAIL_USERNAME']):
            self.role = Role.query.filter_by(permissions=0xff).first()
        else:
            self.role = Role.query.filter_by(default=True).first()
```

把注册路由函数稍作修改

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/views.py

红色行为修改行：

```
if (form.validate_on_submit()):
    user = User(email = form.email.data);
    user.password = form.password.data
    user.name = form.name.data
    #user.email = form.email.data
    db.session.add(user)
    db.session.commit()
```

测试

多注册几个用户，观看数据库效果

```
mysql> select * from roles;
+----+-----+-----+-----+
| id | name   | default | permissions |
+----+-----+-----+-----+
| 8  | admin  | 0       | 255         |
| 9  | user   | 1       | 7           |
| 10 | moderator | 0       | 15          |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select * from users;
+----+-----+-----+-----+-----+-----+
| id | name       | password_hash | email | confirmed | role_id |
+----+-----+-----+-----+-----+-----+
| 48 | lizhiyong | pbkdf2:sha256:50000$HwGqclU$99a687746495f168f18fa29986d20712f88668ce93e888b66ac47c31d0fe2ab7 | lizhiyong_beyond@163.com | 0 | 8 |
| 49 | lizhiyong1 | pbkdf2:sha256:50000$B09v6YCKSe82f3f6e68c3e178dd20923c0cbf86eb9eff4951977c50bf222bc88d659835f5 | lizhiyong_linux@163.com | 0 | 9 |
| 50 | lizhiyong2 | pbkdf2:sha256:50000$KvYvQ7HKSb05ba38a46bcfba5ddac5a23828c0507ca036247551d2f59d339b42161fd18da | lizhiyong@uplooking.com | 0 | 9 |
| 51 | lizhiyong3 | pbkdf2:sha256:50000$kiGubYnB$9w6eafSafc921b02f9dfbe2070c271f02ade94d4b4db05a0f2e22d4ec35f0edc | lizhiyong@123.com | 0 | 9 |
+----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

## 终端测试

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3

manage.py shell

```
>>> from app.models import Role
>>> from app.models import User
>>> r = Role.query.first()
>>> r.users;
[<app.models.User object at 0x7f33c8b32f60>]
>>> r = Role.query.filter_by(default=True).first()
>>> r.users;
[<app.models.User object at 0x7f33c8b32c18>, <app.models.User object at 0x7f33c8b32ef0>, <app.models.User object at 0x7f33c8b32fd0>]
>>> u = User.query.filter_by(role=r).all()
>>> u
[<app.models.User object at 0x7f33c8b32c18>, <app.models.User object at 0x7f33c8b32ef0>, <app.models.User object at 0x7f33c8b32fd0>]
>>>
```

# 角色权限

## 不同的角色应该有不同的权限

比如可以限制，普通用户不能访问某些页面，只有admin才能访问。

### 修改User模型

在User模型中加入能够验证用户权限的方法。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

在User模型中添加如下：

```
def has_permission(self, permissions):  
    return self.role is not None and (self.role.permissions & permissions) ==  
permissions  
def is_admin(self):  
    return self.has_permission(Permission.ADMIN)
```

### 定义匿名用户模型

匿名用户是没有角色的用户，因为没有登录。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

添加如下：

```
from flask_login import AnonymousUserMixin  
class AnonymousUser(AnonymousUserMixin):  
    def has_permission(self, permissions):  
        return False  
    def is_admin(self):  
        return False
```

### 设置匿名用户

以前的代码没有登录的用户我们是不能使用current\_user来访问的。

有了如下代码，匿名用户也能用current\_user来访问。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/\_\_init\_\_.py

添加如下：

```
from .models import AnonymousUser  
login_manager.anonymous_user = AnonymousUser
```

## 自定义装饰器

使用自己定义的装饰器来装饰视图函数。

这样可以限制有某些权限的用户才能访问特定的视图函数。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/decorators.py
```

```
from flask import abort
```

```
from flask_login import current_user
```

```
from .models import Permission
```

```
from functools import wraps
```

```
def permission_decorator(permissions):
```

```
    def decorator(fun):
```

```
        @wraps(fun)
```

```
        def dec_fun(*args, **kwargs):
```

```
            if not current_user.has_permission(permissions):
```

```
                abort(403)
```

```
            #这里要注意：写成下面的return不写是不行的，因为fun函数返回的相应必须返回给
            flask，如果没有return则反不回去
```

```
            return fun(*args, **kwargs)
```

```
        return dec_fun
```

```
    return decorator
```

```
def admin_decorator(f):
```

```
    return permission_decorator(Permission.ADMIN)(f)
```

**理解：**最终能返回红色部分的函数就是一个装饰器。

permission\_decorator(Permission.ADMIN)得到的是一个decorator函数，其实就是一个装饰器。

permission\_decorator(Permission.ADMIN)(f)得到的是红色部分。

在函数中return permission\_decorator(Permission.ADMIN)(f)，那说明这个函数就是装饰器。

## 定义两个特殊的视图函数

其中一个只能admin访问，另外一个只能admin和moderator访问

创建manager蓝本

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ mkdir app/manager
```

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/manager/
```

```
__init__.py
```

```
from flask import Blueprint
```

```
manager = Blueprint('manager', __name__)
```

```
from . import views
```

## 注册蓝本

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/__init__.py
```

```
from .manager import manager as manager_blueprint
app.register_blueprint(manager_blueprint, url_prefix='/manager')
```

## 为蓝本添加视图函数

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/manager/views.py
```

```
from flask import render_template
from ..decorators import permission_decordator
from ..decorators import admin_decortator
from ..models import Permission
from . import manager
```

```
@manager.route('/admin')
@admin_decortator
def for_admin() :
    return render_template('manager/admin.html');
```

```
@manager.route('/moderator')
@permission_decordator(Permission.MODE_COMMENT)
def for_moderator() :
    return render_template('manager/moderator.html');
```

## 为视图创建模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ mkdir app/templates/manager
```

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/templates/
manager/admin.html
```

```
{% extends 'base.html' %}
{% block page_content %}
    Hello,Admin!
{% endblock %}
```

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/templates/
manager/moderator.html
```

```
{% extends 'base.html' %}
{% block page_content %}
    Hello,Admin Or Morderator!
```

```
{% endblock %}
```

在基模板的导航栏中添加两个按钮用于访问管理中心和评论管理

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ touch app/templates/  
base.html

```
<li><a href="/">主页</a></li>
```

```
<li><a href="/">文章</a></li>
```

```
<li><a href="{{ url_for('manager.for_admin') }}">管理中心</a></li>
```

```
<li><a href="{{ url_for('manager.for_moderator') }}">评论管理</a></li>
```

# 用户资料



# 基本用户资料

## 扩展User模型

为了丰富用户信息，我们接下来扩展User模型

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

```
from datetime import datetime
```

```
location = db.Column(db.String(64))
```

```
about_me = db.Column(db.Text())
```

```
#很奇怪下面两行的字段中不能有以下划线，比如register_time就会报错
```

```
registertime = db.Column(db.DateTime(), default=datetime.utcnow())
```

```
accesstime = db.Column(db.DateTime(), default=datetime.utcnow())
```

```
#下面两行和上面两行的区别是每次创建对象后datetime.utcnow都会自动运行来获取时间
```

```
#而上面的两行代码只是获取创建表的时间
```

```
registertime = db.Column(db.DateTime(), default=datetime.utcnow)
```

```
accesstime = db.Column(db.DateTime(), default=datetime.utcnow)
```

```
#添加一个更新用户访问的时间
```

```
def flush_access_time(self):
```

```
    self.access_time = datetime.utcnow()
```

```
    db.session.add(self)
```

```
    db.session.commit()
```

## 更新用户访问时间

因为before\_app\_request可以截获请求，因此在此调用函数来更新用户访问时间

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/auth/views.py

```
@auth.before_app_request
```

```
def before_request():
```

```
    if (current_user.is_authenticated):
```

```
        current_user.flush_access_time()
```

```
    if (current_user.is_authenticated \
```

```
        and not current_user.confirmed \
```

```
        and request.endpoint[:5] != 'auth.' \
```

```
        and request.endpoint != 'static'):
```

```
        return redirect(url_for('auth.unconfirmed'))
```

```
from ..models import User
```

## 添加访问用户资料视图

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

#在这里使用动态请求

```
@main.route('/user/<username>')
def user(username):
    user = User.query.filter_by(name=username).first()
    if (user is None):
        abort(404)
    return render_template('user.html', user=user)
```

## 编写模板

服务器需要统一时间单位，这与用户所在的地理位置（时区）无关。

所以服务器一般使用协调世界时（UTC，Coordinated Universal Time）。

对用户而言，会对 UTC 格式的时间感到困惑，更希望看到本地时间，而且采用当地所惯用的格式。

一个优雅的解决方案是，把时间单位发送给 web 浏览器，转换称当地时间，然后渲染。

moment.js 是一个使用 JavaScript 开发的优秀客户端开源代码库，可在浏览器中渲染日期和时间。

flask-moment 是一个 flask 程序扩展，能把 moment.js 集成到 Jinja2 模板中。

在基模板中添加，引入moment库，用来渲染时间。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/base.html
{%block scripts %}
{{super() }}
{{moment.include_moment() }}
{{ moment.lang('zh-cn') }}
{%endblock %}
```

编写user模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ touch app/templates/main/
user.html
{% extends 'base.html' %}
{% block page_content %}

{{user.name}}<br>
```

```
{{user.about_me}}<br>
{% if user.location %}
所在地: <a href="https://www.baidu.com/s?
wd={{user.location}}">{{user.location}}</a>
{% else %}
没有设定位置
{% endif %}
<br>
{% if user.role %}
{{user.role.name}}
{% endif %}
<br>
{% if current_user.is_admin() %}
<a href="mailto:{{user.email}}">{{user.email}}</a>
{% else %}
{{user.email}}
{% endif %}
<br>
{% if user.confirmed %}
邮箱验证通过
{% else %}
未经过邮箱验证
{% endif %}
<br>
注册时间: {{ moment(user.registertime).format('LLLL') }}
<br>
最后访问时间: {{ moment(user.accesstime).fromNow() }}
{% endblock %}
```

# 用户修改自己资料

## 修改用户资料

普通用户只能修改自己的资料

## 创建编辑用户资料链接

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/user.html

添加如下：

```
{% if user==current_user and not user.is_admin()%}
```

```
<a class="btn btn-default" href="{{url_for('main.edit_user_info')}}">编辑资料</a>
```

```
{% endif %}
```

## 创建表单

创建一个表单，用于让用户修改资料时候输入。

对于普通用户我们只允许修改location和about\_me。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/forms.py

添加如下：

```
from wtforms.validators import Length
```

```
from wtforms import TextAreaField
```

```
class EditInfoUserForm(FlaskForm):
```

```
    location = StringField('Location', validators = [Length(0, 64)])
```

```
    about_me = TextAreaField('AboutMe')
```

```
    submit = SubmitField('提交')
```

## 为表单创建视图函数

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/views.py

```
from .forms import EditInfoUserForm
```

```
from flask_login import current_user
```

**注意：**路由要放到最上面，因为如果路由在下面的话会直接跳过login\_required检测

```
@main.route('/edit-user-info', methods=['GET', 'POST'])
```

```
@login_required
```

```
def edit_user_info():是
```

```
    form = EditInfoUserForm()
```

```
    if (form.validate_on_submit()):
```

```
current_user.location = form.location.data
current_user.about_me = form.about_me.data
db.session.add(current_user)
db.session.commit()
return redirect(url_for('.user', username=current_user.name))
form.location.data = current_user.location
form.about_me.data = current_user.about_me
return render_template('edit_user.html', form=form)
```

## 为视图创建模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/
edit_user.html
```

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

```
{% block page_content %}
```

```
{% import 'bootstrap/wtf.html' as wtf %}
```

```
{{ wtf.quick_form(form)}}
```

```
{% endblock %}
```

# 管理员修改资料

## 修改用户资料

管理员能修改所有用户的所有资料

## 创建编辑用户资料链接

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/user.html

添加如下：

```
{% if current_user.is_admin() %}
```

```
<a class="btn btn-danger" href="{{url_for('main.edit_admin', id=user.id)}}"
```

```
{% endif %}
```

## 创建表单

创建一个表单，用于让用户修改资料时候输入。

对于管理员用户能修改除了id的所有字段。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/forms.py

添加如下：

```
from wtforms.validators import Required, Email, Regexp
```

```
from wtforms import SelectField, ValidationError
```

```
from ..models import Role, User
```

```
class EditInfoAdminForm(FlaskForm):
```

```
    name = StringField('Name', validators = [DataRequired(), Length(6, 64), Regexp('^[a-zA-Z]\w*$', 0, 'Name style error')])
```

```
    password = StringField('Password')
```

```
# 下拉菜单选择类型，coerce=int代表选中后产生的一个整数，默认是字符串
```

```
    role_id = SelectField('Role', coerce=int)
```

```
    confirmed = BooleanField('Confirmed')
```

```
    location = TextAreaField('Location')
```

```
    about_me = StringField('AboutMe')
```

```
    submit = SubmitField('提交')
```

```
def __init__(self, user, *args, **kwargs):
```

```
    super(EditInfoAdminForm, self).__init__(*args, **kwargs)
```

```
# 这里生成了一个[(整数, 字符串), (整数, 字符串)]类型的列表
```

```
# 将来在下拉列表中看到的是字符串，选择后产生的对应的整数
```

```
    self.role_id.choices = [(role.id, role.name) for role in
```

```
Role.query.order_by(Role.id).all()]
```

*#这里是被修改信息的用户*

```
self.user = user
```

**注意：**如果在表单中重写\_\_init\_\_的话需要调用父类的\_\_init\_\_方法

## 为表单创建视图函数

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/  
views.py
```

```
from ..decorators import admin_decoratator
```

```
from .forms import EditInfoAdminForm
```

```
@main.route('/edit-admin/<int:id>', methods=['GET', 'POST'])
```

```
@admin_decoratator
```

*#这里访问的时候需要传入用户id*

```
def edit_admin(id):
```

```
    user = User.query.get_or_404(id)
```

```
    form = EditInfoAdminForm(user=user)
```

```
    if form.validate_on_submit():
```

```
        user.name = form.name.data
```

```
        user.email = form.email.data
```

```
        if form.password.data:
```

```
            user.password = form.password.data
```

```
        user.confirmed = form.confirmed.data
```

```
        user.role_id = form.role_id.data
```

```
        user.about_me = form.about_me.data
```

```
        user.location = form.location.data
```

```
        db.session.add(user)
```

```
        db.session.commit()
```

```
        return redirect(url_for('main.user', username=user.name))
```

*#在表单中填写原来的旧信息*

```
    form.email.data = user.email
```

```
    form.name.data = user.name
```

```
    form.role_id.data = user.role_id
```

```
    form.about_me.data = user.about_me
```

```
    form.location.data = user.location
```

```
    form.confirmed.data = user.confirmed
```

```
    return render_template('edit_admin.html', form=form)
```

## 为视图创建模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/  
edit_user.html  
{% extends 'base.html' %}  
{% block page_content %}  
  
{% import 'bootstrap/wtf.html' as wtf %}  
{{ wtf.quick_form(form)}}  
  
{% endblock %}
```





# 模板

## 在主页中显示所有的博客

### 在base.html模板中添加一块用于显示博客

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/base.html

```
{% block content %}
<div class="container">
  <div class="page-header">
    {% block page_content %}
    {% endblock %}
  </div>
  #此处为添加部分
  <div class="page-header">
    {% block page_content1 %}
    {% endblock %}
  </div>
  <div class="page-header">
    {% block page_content2 %}
    {% endblock %}
  </div>
</div>
{% endblock %}
```

### 注册Permission为全局变量

方便模板中使用

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim manager.py

添加如下：

```
from app.models import Permission
app.add_template_global(Permission, 'Permission')
```

### 在index.html模板中添加用于显示博客

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/index.html

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

```
{% block page_content %}
<font color="#7fffd4", size="5">欢迎, {{name}}!</font>
{% endblock %}
#此处为添加部分
{% block page_content1 %}
#此处用于显示博客表单
{% import 'bootstrap/wtf.html' as wtf %}
{% if current_user.has_permission(Permission.WRITE) %}
{{ wtf.quick_form(form) }}
{% endif %}
{% endblock %}
{% block page_content2%}
此处用于显示博客
{% endblock%}
```

## 创建博客表单

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/forms.py

```
class PostForm(FlaskForm):
    title = StringField("博客标题", validators = [DataRequired()])
    body = TextAreaField("博客正文", validators = [DataRequired()])
    submit = SubmitField("提交")
```

## 修改视图渲染表单

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/views.py

```
from .forms import PostForm
@main.route('/', methods=['GET', 'POST'])
@login_required
def index():
    form = PostForm()
    if (form.validate_on_submit()):
        return redirect(url_for('.index'))
    return render_template('index.html', form = form,
                           name = session.get('name'))
```

## 测试

EMBSKY博客站

127.0.0.1:5000

搜索

EMBSKY博客站

主页

文章

lizhiyong

注册

管理中心

评论管理

我要注册

欢迎, lizhiyong!

博客标题

博客正文

提交

# 模型

## 添加博客数据模型类

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

添加如下：

```
class Post(db.Model) :  
    __tablename__ = "posts"  
    id = db.Column(db.Integer, primary_key = True)  
    title = db.Column(db.String(1024))  
    body = db.Column(db.Text)  
    time = db.Column(db.DateTime, index = True, default = datetime.utcnow)  
    author_id = db.Column(db.Integer, db.ForeignKey('users.id'))
```

## 在User数据模型中添加对博客关系

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

添加如下：

```
posts = db.relationship('Post', backref='author', lazy='dynamic')
```

## 修改数据模型后要更新数据库

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3  
manage.py db migrate -m "add Post"

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3  
manage.py db upgrade

## 修改视图函数

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/views.py

修改如下：

```
from ..models import Post  
@main.route('/', methods=['GET', 'POST'])  
#@login_required  
def index() :  
    form = PostForm()  
    if (current_user.has_permission(Permission.WRITE) and  
form.validate_on_submit()) :  
        post = Post()  
        post.title = form.title.data
```

```

#把\n都换成<br>, html中的换行是<br>
post.body = form.body.data.replace("\n","<br>")
#因为current_user是对user模型的封装, 所以使用_get_current_object来获取原始的
user模型数据
post.author = current_user._get_current_object()
db.session.add(post)
db.session.commit()
return redirect(url_for('.index'))
posts = Post.query.order_by(Post.timestamp.desc()).all()
return render_template('index.html', form = form,
                        name = session.get('name'), posts=posts)

```

## 为视图添加模板

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/index.html

添加如下:

```

{% block page_content2 %}
    {% for post in posts %}
        <div class="page-header">
            <font size="4" color="orange">{{post.title}}</font><br>
            <font size="3" color="red">{{post.author.name}}</font><br>
            <font color="aqua">{{post.timestamp}}</font><br>
            #这里只显示概要
            {{post.body[0:100] | safe}}<br>
        </div>
    {% endfor %}
{% endblock %}

```

## 测试

欢迎, lzhzyong!

博客标题

博客正文

提交

### Ubuntu16.04下编译Android5.1.1

lzhzyong

2017-08-27 16:16:00

```
树源的仓库请到~java_jdk  
mkdir ~java_jdk  
cp ~/jdk-7u80-linux-x64.tar.gz ~java_jdk  
cd ~java_jdk  
tar -xzf jdk-7u80-linux-x64.tar.gz  
在~/bashrc中添加环境变量  
export PATH=~java_jdk/jdk1.7.0_80/bin:$PATH
```

### QTS.4解析http服务器的Json数据

lzhzyong

2017-08-27 16:15:26

```
//服务器返回返回的字符串是 json  
void MainPage::onReplyHandler(QNetworkReply *reply)  
{  
    if (reply->error() == QNetworkReply::NoError) {  
        QByteArray b = reply->readAll();  
        qDebug() << b;  
        QJsonParseError parse;  
        QJsonDocument json = QJsonDocument::fromJson(b, &parse);  
        //创建Json对象  
        QJsonObject obj = json.object();  
    }  
}
```

# 在用户页显示博客

## 在用户首页显示自己发过的博客

### 创建一个html专门显示博客

以后每个需要显示博客的地方都include这个html即可

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/_posts.html
```

```
{% for post in posts %}
    <div class="page-header">
        <font size="4" color="orange">{{post.title}}</font><br>
        <font size="3" color="red">{{post.author.name}}</font><br>
        <font color="aqua">{{post.timestamp}}</font><br>
        {{post.body[0:100] | safe}}<br>
    </div>
{% endfor %}
```

### 修改index.html模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/index.html
```

添加如下：

```
{% block page_content2 %}
{% include '_posts.html' %}
{% endblock %}
```

### 修改user.html

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/user.html
```

添加如下：

```
{% block page_content2 %}
{% include '_posts.html' %}
{% endblock %}
```

### 修改user路由

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

```
def user(username):
    user = User.query.filter_by(name=username).first()
```



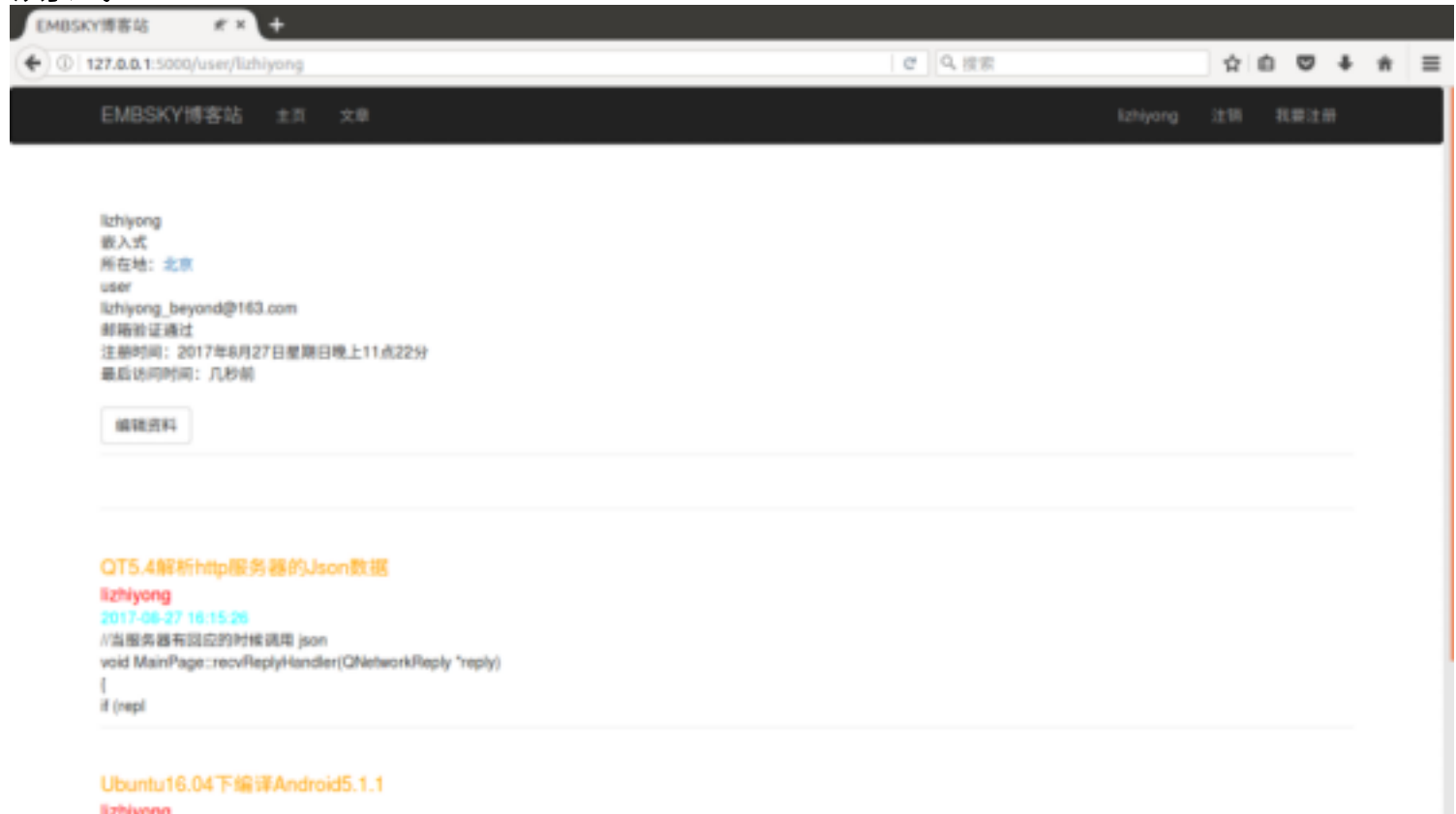
if (user is None) :

abort(404)

posts = Post.query.filter\_by(author=user).order\_by(Post.timestamp.desc()).all()

return render\_template('user.html', user=user, posts=posts)

## 测试



# 虚拟账户和文章

## 虚拟账户和文章

在开发过程中需要添加大量的User和Post模型数据来进行测试。

这里我们使用ForgeryPy来产生大量的用户和文章。

### 安装Forgery

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/pip3 install  
forgerypy
```

### 在User模型类中添加产生user的方法

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/models.py
```

添加如下：

```
@staticmethod
```

```
def generate_fake_user(count):
```

```
    import forgery_py
```

```
    for i in range(count):
```

```
        user = User()
```

```
        user.email=forgery_py.internet.email_address()
```

```
        user.location = forgery_py.address.city()
```

```
        user.about_me = forgery_py.lorem_ipsum.title()
```

```
        user.name = forgery_py.internet.user_name()
```

```
        user.password = forgery_py.lorem_ipsum.word()
```

```
        user.confirmed = True
```

```
        db.session.add(user)
```

```
    try:
```

```
        db.session.commit()
```

```
    except:
```

```
        db.session.rollback()
```

### 生成随机user

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3
```

```
manage.py shell
```

```
>>> from app.models import User
```

```
>>> User.generate_fake_user()
```

## 在Post模型中添加随机生成post的方法

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

@staticmethod

def generate\_fake\_post(count):

import forgery\_py

from random import seed, randint

seed()

num = User.query.count() - 1

for i in range(count):

post = Post()

post.title = forgery\_py.lorem\_ipsum.title().replace(' ', '\_')

post.body = forgery\_py.lorem\_ipsum.sentence()

post.author = User.query.offset(randint(1, num)).first()

db.session.add(post)

try:

db.session.commit()

except:

db.session.rollback()

## 生成随机post

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3

manage.py shell

>>> from app.models import Post

>>> Post.generate\_fake\_post(1000)

# 博客分页显示

## 支持分页

为了显示美观和节省内存资源，文章太多的时候需要分页显示。

## 修改视图路由

修改index路由，使得支持分页显示。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/views.py
```

修改如下：

#默认值是整数1

```
page_index = request.args.get('page', 1, type=int)
```

#第三个参数代表出错后不提示

```
pagination = Post.query.order_by(Post.timestamp.desc()).paginate(page_index, 10, False)
```

```
posts = pagination.items
```

```
#posts = Post.query.order_by(Post.timestamp.desc()).all()
```

```
return render_template('index.html', form = form,
                        name = session.get('name'), posts=posts, a = pagination)
```

## 创建一个用来展示分页样式的宏

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/_macro.html
```

利用bootstrap中的css样式表实现：

```
{% macro pagination_widget(pagination, endpoint) %}
```

```
<ul class="pagination">
```

```
<li {% if not pagination.has_prev %} class="disabled" {% endif %}>
```

```
<a href="{% if pagination.has_prev %}{{ url_for(endpoint, page = pagination.page - 1, **kwargs) }}{% else %}# {% endif %}">
```

上一页

```
</a>
```

```
</li>
```

```
{% for p in pagination.iter_pages() %}
```

```
{% if p %}
```

```
{% if p == pagination.page %}
```

```
<li class="active">
```

```
<a href="{{ url_for(endpoint, page = p, **kwargs) }}">{{ p }}</a>
```

```

    </li>
{% else %}
    <li>
        <a href="{{ url_for(endpoint, page = p, **kwargs) }}">{{ p }}</a>
    </li>
{% endif %}
{% else %}
    <li class="disabled"><a href="#">&hellip;</a></li>
{% endif %}
{% endfor %}
<li {% if not pagination.has_next %} class="disabled" {% endif %}>
    <a href="{% if pagination.has_next %}{{ url_for(endpoint,page =
pagination.page + 1, **kwargs) }}{% else %} {% endif %}">
        下一页
    </a>
</li>
</ul>
{% endmacro %}

```

## 修改index模板

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/index.html  
增加如下：

```

{% block page_content2 %}
{% include '_posts.html' %}
{% import "_macros.html" as macros %}
<div align="center">
    {{ macros.pagination_widget(a, 'index') }}
</div>
{% endblock %}

```

### 注意:

paginate() 方法的返回值是一个 Pagination 类对象,这个类在 Flask-SQLAlchemy 中定义。这个对象包含很多属性,用于在模板中生成分页链接,因此将其作为参数传入了模板。

属性	描述
items	当前页面中的记录
query	分页的源查询
page	当前的页数
prev_num	上一页的页数
next_num	下一页的页数
has_next	如果有下一页返回True
has_prev	如果有上一页返回True
pages	查询得到的总页数
total	查询返回的总记录数

这个对象包含了几个方法。

方法	描述
iter_pages(left_edge=2, left_current=2, right_current=5, right_edge=2)	一个迭代器,返回一个在分页导航中显示的页数列表。这个列表的最左边显示 left_edge 页,当前页的左边显示 left_current 页,当前页的右边显示 right_current 页,最右边显示 right_edge 页。例如,在一个 100 页的列表中,当前页为第 50 页。默认配置,这个方法会返回以下页数:1、 2、 None 、 48、 49、 50、 51。None 之间的间隔
prev()	上一页的分页对象
next()	下一页的分页对象

# 富文本

## 支持富文本和Markdown语法

如果用户想发布长文章,就会觉得在格式上受到了限制。

现在我们需要让多行文本输入框升级, 使其支持 Markdown语法,还要添加富文本文章的预览功能。

## 安装需要的包

PageDown:使用 JavaScript 实现的客户端 Markdown 到 HTML 的转换程序。

Flask-PageDown:为 Flask 包装的 PageDown,把 PageDown 集成到 Flask-WTF 表单中。

Markdown:使用 Python 实现的服务器端 Markdown 到 HTML 的转换程序。

Bleach:使用 Python 实现的 HTML 清理器。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/pip3 install flask-pagedown markdown bleach
```

## 为app添加PageDown

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/__init__.py
```

添加如下:

```
from flask_pagedown import PageDown
```

```
pagedown = PageDown()
```

**#在create\_app中添加如下:**

```
pagedown.init_app(app)
```

## 修改表单

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

**#导入包**

```
from flask_pagedown.fields import PageDownField
```

**#修改body字段**

```
#body = TextAreaField("博客正文", validators = [Required()])
```

```
body = PageDownField("博客正文", validators = [Required()]);
```

## 修改基模板

让模板支持pagedown, 支持后表单中的PageDownField会有预览效果

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/base.html
```

添加如下（红色部分）：

```
{%block scripts %}
```

```
{{ super() }}
```

```
{{ moment.include_moment() }}
```

```
{{ pagedown.include_pagedown() }}
```

```
{{ moment.lang('zh-cn') }}
```

```
{%endblock %}
```



# 后端处理Markdown

## 在服务器上处理富文本

提交表单后, POST 请求只会发送纯 Markdown 文本,页面中显示的 HTML 预览会被丢掉。  
可以在服务器上使用 Markdown 将其转换成 HTML。  
得到 HTML 后,再使用 Bleach 进行清理,确保其中只包含几个允许使用的HTML 标签。

## 在Post模型中添加字段

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

在Post中添加如下:

```
body_html = db.Column(db.Text)
```

```
@staticmethod
```

```
def on_change_body(target, value, oldvalue, initiator):
```

```
    allowed_tags = ['a', 'abbr', 'acronym', 'b', 'blockquote', 'code',  
                    'em', 'i', 'li', 'ol', 'pre', 'strong', 'ul',  
                    'h1', 'h2', 'h3', 'p', 'br']
```

```
    #把markdown转换为html
```

```
    m = markdown(value, output_format='html')
```

```
    #删除不支持的标签
```

```
    bm = bleach.clean(m, tags = allowed_tags, strip = True)
```

```
    #处理链接
```

```
    target.body_html = bleach.linkify(bm)
```

注意: 此句需要执行, 因此要写到类的外边

```
#添加监听器, 监听Post.body, 如果有变化则执行on_change_body方法
```

```
db.event.listen(Post.body, 'set', Post.on_change_body)
```

## 修改数据模型后要更新数据库

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3  
manage.py db migrate -m "add body_html Post"
```

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3  
manage.py db upgrade
```

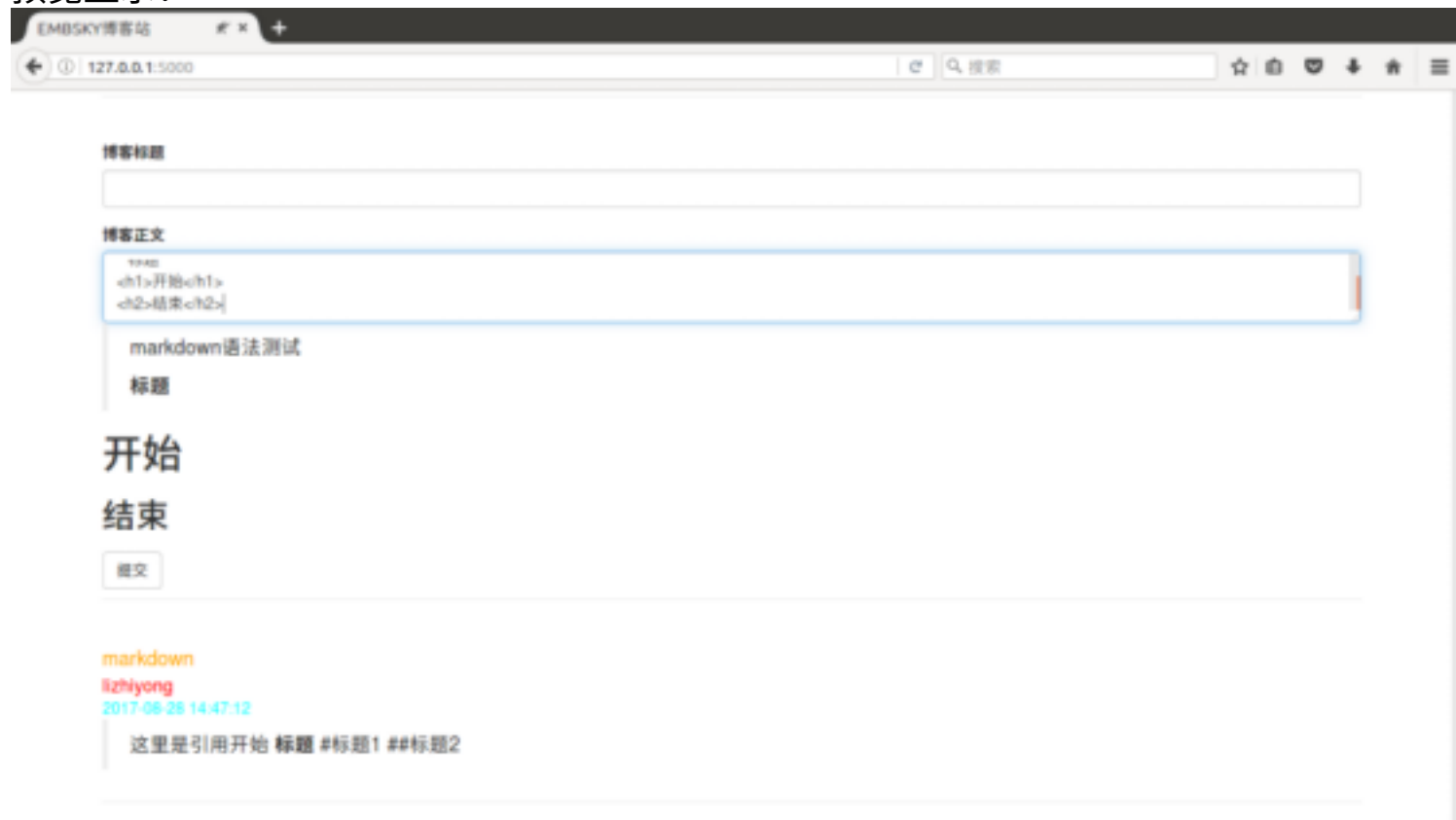
## 修改模板\_post.html

#如果body\_html存在则不展示body

```
{% if post.body_html %}  
    {{ post.body_html | safe }}  
{% else %}  
    {{post.body[0:100] | safe}}<br>  
{% endif %}
```

## 测试

预览显示：



实际显示：

欢迎, lizhiyong!

博客标题

博客正文

提交

markdown

lizhiyong

2017-06-26 14:49:38

markdown语法测试 > 标题

开始

结束

# 其他扩展

## 文章唯一ID

每一篇文章都应该有一个唯一的url，这样方便把链接发送给别人查看。

## 给博客标题和博主名字加超链接

博客在主页只显示一部分，点击标题后切换到另一个界面显示全部。

修改模板\_posts.html，给文章标题添加超链接。

并加入变量all，如果all为True则显示全部内容，否则只显示一部分。

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/\_posts.html

修改如下：

```
{% for post in posts %}
    <a href="{{ url_for('main.post', id=post.id) }}"> {{ post.title }}</a><br>
    <a href="{{ url_for('main.user', username=post.author.name) }}">
{{ post.author.name }}</a> <br>
    {{ post.timestamp }} {{ moment(post.timestamp).fromNow() }}<br>
    {% if not all %}
        {% if post.body_html %}
            {{ post.body_html[:200] | safe }}<br><br>
        {% else %}
            {{ post.body[:200] | safe }}<br><br>
        {% endif %}
    {% else %}
        {% if post.body_html %}
            {{ post.body_html | safe }}<br><br>
        {% else %}
            {{ post.body | safe }}<br><br>
        {% endif %}
    {% endif %}
{% endfor %}
```

## 创建路由

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/views.py

```
@main.route('/post/<int:id>')
```

```
def post(id) :
```

```
post = Post.query.get_or_404(id)
return render_template('post.html', posts = [post], all=True)
```

## 创建模板

用来单独显示一个post。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/posts.html
```

```
{% extends 'base.html' %}
{% block page_content %}
{% include '_posts.html' %}
{% endblock %}
```

# 数据库关系

# 关系

## 数据库关系

- 一对一
- 一对多
- 多对一
- 多对多

### 一对多的关系

一个角色能对应多个用户， 但一个用户只能是一个角色。  
用户表中CLASS列是指向角色表的外键。

ID	NAME	CLASS
1	zhangsan	1
2	lisi	1
3	wangwu	1
4	zhangliu	2
5	wangqi	2
6	zhaoba	2
7	haojiu	3

ID	NAME
1	USER
2	MODERATOR
3	ADMIN

### 多对多的关系

关系表中SID是指向学生表的外键。  
关系表中CID是指向课程 表的外键。

学生表                      关系表                      课程表

ID	NAME
1	zhangsan
2	lisi
3	wangwu
4	zhangliu
5	wangqi
6	zhaoba
7	haojiu

ID	SID	CID
1	2	1
2	2	2
3	3	2
4	3	1
5	4	2
6	5	1
7	5	3

ID	NAME
1	ENGLISH
2	CHINESE
3	MATH



# 数据模型

## 添加测试模型

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

添加如下：

*#这是一个内部表，不需要我们操作*

```
rel_table = db.Table('rel_table', \
    db.Column('sid', db.Integer, db.ForeignKey('stus.id')), \
    db.Column('cid', db.Integer, db.ForeignKey('clas.id')))
```

```
class Stu(db.Model):
```

```
    __tablename__ = 'stus'
```

```
    id = db.Column(db.Integer, primary_key = True)
```

```
    name = db.Column(db.String(32))
```

```
    classes = db.relationship('Cla', \
        secondary=rel_table, \
        backref=db.backref('students', lazy='dynamic'), \
        lazy='dynamic')
```

*#backref="xxxxx" xxxx仅仅是一个列表*

*#backref=db.backref('xxxx', lazy='dynamic') xxxx是一个可以append的查询*

```
class Cla(db.Model):
```

```
    __tablename__ = 'clas'
```

```
    id = db.Column(db.Integer, primary_key = True)
```

```
    name = db.Column(db.String(32))
```

## 修改数据模型后要更新数据库

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3

manage.py db migrate -m "add test"

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3

manage.py db upgrade

## 测试

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ ./venv/bin/python3

manage.py shell

```
>>> from app.models import Stu,Cla
```

```
>>> s1 = Stu(name='zhangsan')
```

```

>>> s2 = Stu(name='lisi')
>>> s3 = Stu(name='wangwu')
>>> c1 = Cla(name='english')
>>> c2 = Cla(name='chinese')
>>> c3 = Cla(name='math')
>>> db.session.add(s1)
>>> db.session.add_all([s2,s3,c1,c2,c3])
>>> db.session.commit()
>>> s1.classes.append(c1)
>>> s1.classes.append(c2)
>>> db.session.add(s1)
>>> db.session.commit()
>>> s2.classes.append(c3)
>>> s2.classes.append(c2)
>>> s2.classes.append(c1)
>>> s2.classes.remove(c1)
>>> db.session.add(s2)
>>> db.session.commit()
>>> for c in c1.students :
...     print(c.id, c.name)
...
1 zhangsan
>>> for c in c2.students :
...     print(c.id, c.name)
...
1 zhangsan
2

```

## 数据库

```
mysql> select * from clas;
```

id	name
1	english
2	chinese
3	math

```
3 rows in set (0.00 sec)
```

```
mysql> select * from stus;
```

id	name
1	zhangsan
2	lisi
3	wangwu

```
3 rows in set (0.00 sec)
```

```
mysql> select * from rel_table;
```

sid	cid
1	2
1	1
2	3
2	2

```
4 rows in set (0.00 sec)
```

总结：

这种方法的缺点在于关系表是内部表，我们的代码无法直接访问。

关系表无法加入自己的属性。

## 关注博主

# 多对多关系

## 多对多关系

这里使用模型来表示关系，这样的好处在于能够自定义关系表的机构。

由于关注者和被关注者都是user，都在一个users表中，所以这里采用自引用关系。

## 创建关系数据模型类

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

**#构建关系数据模型**

**class Follow(db.Model) :**

**\_\_tablename\_\_ = "follows"**

*#关注者id*

**follower\_id = db.Column(db.Integer, db.ForeignKey('users.id'), primary\_key=True)**

*#被关注者id*

**followed\_id = db.Column(db.Integer, db.ForeignKey('users.id'), primary\_key=True)**

*#事件发生的时间*

**timestamp = db.Column(db.DateTime, default=datetime.utcnow)**

**#在User模型类中添加关系**

**class User(db.Model, UserMixin) :**

*#其他内容*

*#添加如下：*

*#我们的关注者*

**followers = db.relationship('Follow', \**  
                                **foreign\_keys=[Follow.followed\_id], \**  
                                **backref=db.backref('followed', lazy='joined'), \**  
                                **lazy='dynamic', \**  
                                **cascade='all, delete-orphan')**

*#注意：*

*#foreign\_keys指定关联的属性，*

*#即Follow.followed\_id等于User.id的所有Follow实例都在followers中，*

*#即我们的关注者，也就是关注我们的用户*

*#在Follow实例中添加一个属性叫做followed，*

*#通过Follow实例的followed属性能够找到followed\_id等User.id的User实例，*

*#即被关注者*

*#lazy='joined'代表当找到Follow实例时，其中的followed已经放好了User实例*

#lazy='dynamic'代表找到User实例时，其中的followers找到的只是一个查询  
#cascade='all,delete-orphan'代表删除父对象后把与其关联的字对象也都删掉，all代表其他选项按默认

#被我们关注的

```
followeds = db.relationship('Follow', \
                             foreign_keys=[Follow.follower_id], \
                             backref=db.backref('follower', lazy='joined'), \
                             lazy='dynamic', \
                             cascade='all, delete-orphan')
```

#注意：

#foreign\_keys指定关联的Follow中的属性，  
#即Follow实例中follower\_id等于User.id的都放在followeds中，  
#即我们关注的用户  
#在Follow实例中添加一个属性follower，  
#Follow实例通过follower属性能够找到follower\_id等于User.id的User实例，  
#即关注别人的用户，也就是关注者  
#lazy='joined'代表当找到Follow实例时，其中的follower已经放好了User实例  
#lazy='dynamic'代表找到User实例时，其中的followeds找到的只是一个查询  
#cascade='all,delete-orphan'代表删除父对象后把与其关联的字对象也都删掉，all代表其他选项按默认

```
def follow(self, user):
    if not self.is_follower_of(user):
        f = Follow()
        f.follower = self
        f.followed = user
        db.session.add(f)
        db.session.commit()
```

```
def unfollow(self, user):
    if self.is_follower_of(user):
        f = self.followeds.filter_by(followed_id=user.id).first()
        db.session.delete(f)
        db.session.commit()
```

```
def is_followed_by(self, user):
    return self.followers.filter_by(follower_id=user.id).first() is not None
```

```
def is_follower_of(self, user):  
    return self.followeds.filter_by(followed_id=user.id).first() is not None  
}
```

## 修改数据模型后要更新数据库

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3  
manage.py db migrate -m "add test"  
lzy@embsky:/home/zyli/test/python/flask/bkProject$ ./venv/bin/python3  
manage.py db upgrade
```

## 测试：

```
mysql> show tables;  
+-----+  
| Tables_in_devdb |  
+-----+  
| alembic_version |  
| clas             |  
| follows          |  
| posts           |  
| rel_table        |  
| roles            |  
| students         |  
| stus             |  
| teachers         |  
| users            |  
+-----+  
10 rows in set (0.00 sec)
```

## 关注信息

## 用户页面调整

在用户页面添加：

- 1.关注/取消关注按钮
- 2.关注数量
- 3.被关注数量
- 4.自己是否被该用户关注

## 修改用户页面模板user.html

在用户资料页面显示关注信息

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/user.html
```

<!--关注信息-->

```
{% block page_content1 %}
```

`<a href="{ url_for('main.followers', id=user.id) }">我的粉丝</a>`

```
<span class="badge">{{ user.followers.count() }}</span>
```

`<a href="{ url_for('main.followeds', id=user.id) }">我关注的</a>`

```
<span class="badge">{{ user.followeds.count() }}</span>
```

```
{% if current_user.has_permission(Permission.FOLLOW) %}
```

```
{% if current_user.is_follower_of(user) %}
```

`<a href="{ { url_for('main.unfollow', id=user.id) } }">取消关注</a>`

```
{% else %}
```

[关注]({url_for('main.follow', id=user.id)})

```
{% endif %}
```

```
{% endif %}
```

```
{% if current_user.is_authenticated %}
```

```
{% if current_user.is_followed_by(user) %}
```

`<span class="badge">他(她)关注了我</span>`

```
{% else %}
```

`<span class="badge">他(她)没有关注我</span>`

```
{% endif %}
```

```
{% endif %}
```



```
{% endblock %}
```

```
<!--关注信息-->
```

## 添加路由

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

```
from ..decorators import permisson_decordator
```

```
from ..models import Permission
```

```
#关注用户
```

```
@main.route('/follow/<int:id>')
```

```
@login_required
```

```
@permisson_decordator(Permission.FOLLOW)
```

```
def follow(id) :
```

```
    user = User.query.filter_by(id=id).first()
```

```
    current_user.follow(user)
```

```
    return redirect(url_for('.user', username=user.name))
```

```
#取消关注用户
```

```
@main.route('/unfollow/<int:id>')
```

```
@login_required
```

```
@permisson_decordator(Permission.FOLLOW)
```

```
def unfollow(id) :
```

```
    user = User.query.filter_by(id=id).first()
```

```
    current_user.unfollow(user)
```

```
    return redirect(url_for('.user', username=user.name))
```

```
#查看关注的人
```

```
@main.route('/followeds/<int:id>')
```

```
def followeds(id) :
```

```
    user = User.query.filter_by(id=id).first()
```

```
    fs = user.followeds.order_by(Follow.timestamp.desc()).all()
```

```
    return render_template('main/followeds.html', title='大V们', fs=fs)
```

```
#查看粉丝
```

```
from app.models import Follow
```

```
@main.route('/followers/<int:id>')
```

```
def followers(id) :
```

```
    user = User.query.filter_by(id=id).first()
```

```
    fs = user.followers.order_by(Follow.timestamp.desc()).all()
```

```
return render_template('main/followers.html', titile='粉丝们', fs=fs)
```

## 添加模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/main/
followeds.html
```

```
{% extends 'base.html' %}
{% block page_content %}
    {{ title }}
{% endblock %}

{% block page_content1 %}
    {% for f in fs %}
        昵称: {{ f.followed.name }}<br>
        {% if f.followed.location %}
            位置: <a href="https://www.baidu.com/s?
wd={{ f.followed.location }}">{{ f.followed.location }}</a><br>
        {% endif %}
        关于我: {{ f.followed.about_me }}<br>
        {% if current_user.is_admin() %}
            邮箱: <a href="mailto:{{ f.followed.email }}">{{ f.followed.email }}</a><br>
        {% else %}
            邮箱: {{ f.followed.email }}<br>
        {% endif %}
        {% if f.followed.confirm %}
            邮箱验证: 是<br>
        {% else %}
            邮箱验证: 否<br>
        {% endif %}
        角色: {{ f.followed.role.name }}<br>
        注册时间: {{ moment(f.followed.r_time).format('LLL') }}<br>
        最近访问时间: {{ moment(f.followed.a_time).fromNow() }}<br>
        关注时间: {{ moment(f.timestamp).fromNow() }}
        <br>
        <br>
    {% endfor %}
{% endblock %}
```

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/main/
```

followers.html

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

```
{% block page_content %}
```

```
    {{ title }}
```

```
{% endblock %}
```

```
{% block page_content1 %}
```

```
    {% for f in fs %}
```

```
        昵称: {{ f.follower.name }}<br>
```

```
        {% if f.follower.location %}
```

```
            位置: <a href="https://www.baidu.com/s?
```

```
wd={{ f.follower.location }}">{{ f.follower.location }}</a><br>
```

```
        {% endif %}
```

```
        关于我: {{ f.follower.about_me }}<br>
```

```
        {% if current_user.is_admin() %}
```

```
            邮箱: <a href="mailto:{{ f.follower.email }}">{{ f.follower.email }}</a><br>
```

```
        {% else %}
```

```
            邮箱: {{ f.follower.email }}<br>
```

```
        {% endif %}
```

```
        {% if f.follower.confirm %}
```

```
            邮箱验证: 是<br>
```

```
        {% else %}
```

```
            邮箱验证: 否<br>
```

```
        {% endif %}
```

```
        角色: {{ f.follower.role.name }}<br>
```

```
        注册时间: {{ moment(f.follower.r_time).format('LLL') }}<br>
```

```
        最近访问时间: {{ moment(f.follower.a_time).fromNow() }}<br>
```

```
        被关注时间: {{ moment(f.timestamp).fromNow() }}
```

```
    {% endfor %}
```

```
{% endblock %}
```

# 联结查询

## 什么是联结查询？

多张表有关系，通过一条sql语句同时查询多张表得到一个虚拟表的过程叫做联结查询。

比如：要查询lizhiyong关注的所有user的post。

此时要涉及到三张表：users表，follows表，posts表。

首先通过users表找到lizhiyong对应的id。

然后再通过id从follows表中找到lizhiyong关注的uesr的id。

然后再通过lizhiyong关注的user的id从posts中找到post。

## 利用传统的方法查询

```
mysql> select id,name from users where name='lizhiyong';
```

```
+---+-----+
```

```
| id | name |
```

```
+---+-----+
```

```
| 2 | lizhiyong |
```

```
+---+-----+
```

```
1 row in set (0.01 sec)
```

```
mysql> select * from follows where follower_id=2;
```

```
+-----+-----+-----+
```

```
| follower_id | followed_id | timestamp |
```

```
+-----+-----+-----+
```

```
| 2 | 34 | 2017-08-30 05:47:08 |
```

```
| 2 | 50 | 2017-08-30 05:47:02 |
```

```
| 2 | 52 | 2017-08-30 05:47:12 |
```

```
| 2 | 56 | 2017-08-30 05:41:13 |
```

```
| 2 | 104 | 2017-08-30 08:17:25 |
```

```
+-----+-----+-----+
```

```
5 rows in set (0.00 sec)
```

```
mysql> select id,title from posts where author_id=34 or author_id=50 or  
author_id=52 or author_id=56 or author_id=104;
```

```
+---+-----+
```

```
| id | title |
```

```
+---+-----+
```

```

| 104 | Imperdiet_lectus_cubilia_orci.          |
| 217 | Quis_adipiscing_orci_consectetuer.      |
| 251 | Quisque_vivamus_leo_a?                  |
| 170 | Elit_ante_vestibulum_consectetuer!      |
| 283 | Mus_vestibulum_nibh_praesent.           |
| 299 | Parturient_aenean_urna_pellentesque?    |
| 23  | Dis_quis_praesent_ridiculus?            |
| 52  | Sit_placerat_duis_ultrices?             |
| 292 | Et_et_semper_faucibus!                  |
| 3   | Nam_luctus_tempor_auctor!               |
| 122 | Cras_at_in_leo!                         |
| 171 | Eros_justo_ut_amet?                     |
| 301 | ubuntu用户注意了                       |
| 302 | 6818开发板emmc无法使用解决办法         |
| 303 | ubuntu下sudo apt-get update出错解决方法 |
+----+-----+
15 rows in set (0.00 sec)

```

## 利用联结方法查询

`mysql> select posts.id, title from users, follows, posts where users.name='lizhiyong' and users.id=follows.follower_id and follows.followed_id=posts.author_id;`

```

+----+-----+
| id | title          |
+----+-----+
| 104 | Imperdiet_lectus_cubilia_orci.          |
| 217 | Quis_adipiscing_orci_consectetuer.      |
| 251 | Quisque_vivamus_leo_a?                  |
| 170 | Elit_ante_vestibulum_consectetuer!      |
| 283 | Mus_vestibulum_nibh_praesent.           |
| 299 | Parturient_aenean_urna_pellentesque?    |
| 23  | Dis_quis_praesent_ridiculus?            |
| 52  | Sit_placerat_duis_ultrices?             |
| 292 | Et_et_semper_faucibus!                  |
| 3   | Nam_luctus_tempor_auctor!               |
| 122 | Cras_at_in_leo!                         |
| 171 | Eros_justo_ut_amet?                     |
| 301 | ubuntu用户注意了                       |
| 302 | 6818开发板emmc无法使用解决办法         |

```

```
| 303 | ubuntu下sudo apt-get update出错解决方法 |
```

```
+-----+-----+
```

```
15 rows in set (0.00 sec)
```

```
mysql> select posts.id, title from users inner join follows inner join posts on  
users.name='lizhiyong' and users.id=follows.follower_id and  
follows.followed_id=posts.author_id;
```

```
+-----+-----+
```

```
| id | title |
```

```
+-----+-----+
```

```
| 104 | Imperdiet_lectus_cubilia_orci. |
```

```
| 217 | Quis_adipiscing_orci_consectetuer. |
```

```
| 251 | Quisque_vivamus_leo_a? |
```

```
| 170 | Elit_ante_vestibulum_consectetuer! |
```

```
| 283 | Mus_vestibulum_nibh_praesent. |
```

```
| 299 | Parturient_aenean_urna_pellentesque? |
```

```
| 23 | Dis_quis_praesent_ridiculus? |
```

```
| 52 | Sit_placerat_duis_ultrices? |
```

```
| 292 | Et_et_semper_faucibus! |
```

```
| 3 | Nam_luctus_tempor_auctor! |
```

```
| 122 | Cras_at_in_leo! |
```

```
| 171 | Eros_justo_ut_amet? |
```

```
| 301 | ubuntu用户注意了 |
```

```
| 302 | 6818开发板emmc无法使用解决办法 |
```

```
| 303 | ubuntu下sudo apt-get update出错解决方法 |
```

```
+-----+-----+
```

```
15 rows in set (0.00 sec)
```

```
mysql>
```

# 关注者博客

## 在主页只显示关注的user的博客

通过ALL和FOLLOW按钮切换究竟是显示全部博客还是所关注用户的博客。

## 在主页中添加ALL和FOLLOW按钮

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/index.html

```
<a href="{{url_for('main.all_posts')}}"><span class="btn btn-danger">ALL</span></a>
<a href="{{url_for('main.follow_posts')}}"><span class="btn btn-danger">FOLLOW</span></a>
```

## 添加路由

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/views.py

```
from flask import make_response
```

#点击ALL按钮后的路由

```
@main.route('/show-all-posts')
```

```
def show_all_posts():
```

#手工构建响应对象，设置相应的路由

```
resp = make_response(redirect(url_for('.index')))
```

#设置响应的cookie，show\_follow\_posts的值为''

```
resp.set_cookie('show_follow_posts', '', 60 * 60 * 24 * 30)
```

#返回响应后浏览器能记住show\_follow\_posts的值

```
return resp
```

#点击FOLLOW按钮后的路由

```
@main.route('/show-follow-posts')
```

```
@login_required
```

```
def show_follow_posts():
```

#手工构建响应对象，设置相应的路由

```
resp = make_response(redirect(url_for('.index')))
```

#设置响应的cookie，show\_follow\_posts的值为'1'

```
resp.set_cookie('show_follow_posts', '1', 60 * 60 * 24 * 30)
```

#返回响应后浏览器能记住show\_follow\_posts的值

```
return resp
```

#修改index路由

```

def index():
    form = PostForm()
    if (current_user.has_permission(Permission.WRITE) and
form.validate_on_submit()):
        post = Post()
        post.title = form.title.data
        post.body = form.body.data.replace("\n", "<br>")
        post.author = current_user._get_current_object()
        db.session.add(post)
        db.session.commit()
        return redirect(url_for('.index'))
    page_index = request.args.get('page', 1, type=int)
    follow_post = False
    if current_user.is_authenticated:
        #从浏览器的cookie中获取show_follow_posts的值
        follow_post = bool(request.cookies.get('show_follow_posts', ''))
    if follow_post:
        #从当前用户的属性followed中获取关注的用户的博客BaseQuery
        query = current_user.followed_posts
    else:
        query = Post.query
    pagination = query.order_by(Post.timestamp.desc()).paginate(page_index, 10,
False)

    posts = pagination.items
    #posts = Post.query.order_by(Post.timestamp.desc()).all()
    return render_template('index.html', form = form,
        name = session.get('name'), posts=posts, a = pagination)

```

## 修改User模型

@property

```
def followed_posts(self):
```

#采用联结方式查询

#过滤器join把当前查询的结果和Follow联结查询（条件是Follow.followed\_id==Post.author\_id），返回一个查询

#然后再用filter过滤器过滤（条件是Follow.follower\_id==self.id）

```

    return Post.query.join(Follow,
Follow.followed_id==Post.author_id).filter(Follow.follower_id==self.id)

```





# 用户评论

# 发表评论

## 用户评论

登录的用户能够对post发表评论

## 定义评论表单

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/main/forms.py

```
class CommentForm(FlaskForm):  
    body = PageDownField('评论', validators=[DataRequired(), Length(1, 128)])  
    submit = SubmitField('提交')
```

## 创建评论模型

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

```
class Comment(db.Model):  
    __tablename__ = 'comments'  
    id = db.Column(db.Integer, primary_key = True)  
    body = db.Column(db.Text)  
    body_html = db.Column(db.Text)  
    timestmap = db.Column(db.DateTime, default=datetime.utcnow)  
    disabled = db.Column(db.Boolean)  
    author_id = db.Column(db.Integer, db.ForeignKey('users.id'))  
    post_id = db.Column(db.Integer, db.ForeignKey('posts.id'))  
    @staticmethod  
    def on_change_body(target, value, oldvalue, initiator):  
        allowed_tags = ['a', 'abbr', 'acronym', 'b', 'blockquote', 'code',  
                        'em', 'i', 'li', 'ol', 'pre', 'strong', 'ul',  
                        'h1', 'h2', 'h3', 'p', 'br']  
        m = markdown(value, output_format='html')  
        bm = bleach.clean(m, tags = allowed_tags, strip = True)  
        target.body_html = bleach.linkify(bm)
```

```
db.event.listen(Comment.body, 'set', Comment.on_change_body)
```

## 修改User和Post模型

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/models.py

#分别加入以下关系：

```
comments = db.relationship('Comment', backref='post', lazy='dynamic')
```

```
comments = db.relationship('Comment', backref='author', lazy='dynamic')
```

## 修改视图路由

点击某一篇博客的标题时跳转到该路由。

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

```
from .forms import CommentForm
```

```
from ..models import Comment
```

```
@main.route('/post/<int:id>', methods=['POST', 'GET'])
```

```
def post(id) :
```

```
    form = CommentForm()
```

```
    post = Post.query.get_or_404(id)
```

```
    if form.validate_on_submit() :
```

```
        if current_user.has_permission(Permission.COMMENT) and post is not None:
```

```
            comment = Comment()
```

```
            comment.post = post
```

```
            comment.body = form.body.data.replace('\n', '<br>')
```

```
            comment.author = current_user
```

```
            comment.disabled = False
```

```
            db.session.add(comment)
```

```
            db.session.commit()
```

```
            return redirect(url_for('main.post', id=post.id))
```

```
    else :
```

```
        return redirect(url_for('auth.login'))
```

```
    comments = None
```

```
    if post :
```

```
        comments = post.comments.order_by(Comment.timestamp.desc()).all();
```

```
    return render_template('post.html', posts = [post], form=form,
```

```
    comments=comments, all=True)
```

## 添加模板post.html

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/post.html
```

添加如下：

```
{% block page_content1 %}
```

```
    {% if current_user.has_permission(Permission.COMMENT) %}
```

```
        {% import 'bootstrap/wtf.html' as wtf %}
```

[illegible]

# 评论管理

## 管理员和协管理员有权限对评论进行管理

可以把评论屏蔽

在两个地方对评论进行管理：

- 1.评论管理页面
- 2.博客页面（博客下方会有这个博客的评论）

## 修改评论管理路由

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/manager/views.py

在base.html中有一个按钮叫做'评论管理'，点击后跳转到这个路由。

所有的评论分页显示

```
from ..models import Comment
@manager.route('/moderator')
@permission_decordator(Permission.MODE_COMMENT)
def for_moderator():
    #comments = Comment.query.order_by(Comment.timestmap.desc()).all()
    page_index = request.args.get('page', 1, type=int)
    pagniation =
    Comment.query.order_by(Comment.timestmap.desc()).paginate(page_index, 10,
False)
    comments = pagniation.items
    return render_template('manager/moderator.html', comments = comments, a =
pagniation);
```

## 在评论管理页面管理评论

### 修改评论管理模板moderator.html

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/manager/moderator.html

```
{% extends 'base.html' %}
{% block page_content %}
    {% include 'manager/comments.html' %}
{% endblock %}
```

## 创建显示评论管理页面模板

zy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/templates/  
comment.html

```
<h1>欢迎来到评论管理</h1>
{% for c in comments %}
    主题: {{ c.post.title }}<br>
    作者: {{ c.post.author.name }}<br>
    评论: {{ c.body | safe }}<br>
    评论者: {{ c.author.name }}<br>
    评论时间: {{ moment(c.timestamp).fromNow() }}<br>
    {% if c.disable %}
        <font color="#dc143c">评论已经被禁止</font><br>
        <a class="badge" href="{{ url_for('manager.enable_comment', id=c.id,
page=paginate.page) }}">解除屏蔽</a><br>
    {% else %}
        <a class="badge" href="{{ url_for('manager.disable_comment', id=c.id,
page=paginate.page) }}">屏蔽评论</a><br>
    {% endif %}
<br>
{% endfor %}
<br>
{% import 'page/page_macro.html' as p %}
<div align="center">
    {{ p.pagination_widget(paginate, 'manager.moderator_fun') }}
</div>
```

## 添加屏蔽和取消屏蔽路由

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/manager/views.py

```
from .. import db
from flask import redirect,url_for,request
@manager.route('/disable-comment/<int:id>/<int:page>', methods=['POST',
'GET'])
@permission_decordator(Permission.MODE_COMMENT)
def disable_comment(id,page) :
    comment = Comment.query.filter_by(id=id).first()
    comment.disabled = True
    db.session.add(comment)
    db.session.commit()
    return redirect(url_for('.for_moderator', page=page))
```

```
@manager.route('/enable-comment/<int:id>/<int:page>', methods=['POST',
'GET'])
@permission_decorator(Permission.MODE_COMMENT)
def enable_comment(id, page) :
    comment = Comment.query.filter_by(id=id).first()
    comment.disabled = False
    db.session.add(comment)
    db.session.commit()
    return redirect(url_for('.for_moderator', page=page))
```

## 测试



## 在博客页面管理评论

修改屏蔽和取消屏蔽的路由

**注意：**在一个路由函数不是动态路由的时候，参数都会放到request中，通过request.args可以获取

lzy@embsky:/home/zyli/test/python/flask/bkProject\$ vim app/manager/views.py

```
@manager.route('/disable-comment')
@permission_decorator(Permission.MODE_COMMENT)
def disable_comment():
    id = request.args.get('id')
    page = request.args.get('page')
    #加入url用来表示博客页面的url，方便重定向到原来的页面
```







# 其他功能扩展

# 博客编辑

## 博客编辑

添加一个编辑博客的功能。

一个博客只能由博主和管理员修改。

在完全显示博客的页面增加一个编辑按钮。

## 修改posts.html模板

增加红色部分

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/posts.html
```

```
{% extends 'base.html' %}
{% block page_content %}
    {% include 'main/_posts.html' %}
    {% if current_user == posts[0].author or current_user.is_admin() %}
        <a href="{{ url_for('main.edit_post', id=posts[0].id, url=request.url) }}"
class="badge">
        编辑
    </a>
    {% endif %}
{% endblock %}
```

## 构建表单

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/forms.py
```

```
class EditPostForm(PostForm):
    pass
```

## 创建视图路由

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

```
from .forms import EditPostForm
@login_required
@main.route('/edit-post', methods=['GET', 'POST'])
def edit_post():
    id = request.args.get('id')
    url = request.args.get('url')
    post = Post.query.get_or_404(id)
```

#如果当前用户不是文章的作者并且当前用户也不是管理员的话不能编辑文章

```
if current_user != post.author and not current_user.is_admin():  
    return redirect(url)
```

```
form = EditPostForm()
```

```
if form.validate_on_submit():
```

```
    post.body = form.body.data.replace("\n", "<br>")
```

```
    post.title = form.title.data
```

```
    db.session.add(post)
```

```
    db.session.commit()
```

```
    return redirect(url)
```

```
form.title.data = post.title
```

```
form.body.data = post.body_html
```

```
return render_template('edit_post.html', form=form)
```

## 创建模板

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/  
edit_post.html
```

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

```
{% block page_content %}
```

```
{% import 'bootstrap/wtf.html' as wtf %}
```

```
{{ wtf.quick_form(form) }}
```

```
{% endblock %}
```

# 博客删除

## 博客删除

添加一个删除博客的功能。

一个博客只能由博主和管理员删除。

在完全显示博客的页面增加一个删除按钮。

## 修改posts.html模板

增加红色部分

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/templates/posts.html
```

```
{% extends 'base.html' %}
{% block page_content %}
    {% include 'main/_posts.html' %}
    {% if current_user == posts[0].author or current_user.is_admin() %}
        <a href="{{ url_for('main.edit_post', id=posts[0].id, url=request.url) }}"
class="badge">
            编辑
        </a>
        <a href="{{ url_for('main.delete_post', id=posts[0].id, url=request.url) }}"
class="badge">
            删除
        </a>
    {% endif %}
{% endblock %}
```

## 创建视图路由

```
lzy@embsky:/home/zyli/test/python/flask/bkProject$ vim app/main/views.py
```

添加如下：

```
@main.route('/delete_post', methods=['GET', 'POST'])
@login_required
def delete_post():
    id = request.args.get('id')
    url = request.args.get('url')
    post = Post.query.filter_by(id=id).first()

    if current_user != post.author and not current_user.is_admin():
```

```
return redirect(url)
```

```
db.session.delete(post)
```

```
db.session.commit()
```

```
return redirect(url_for('.user_info', username=post.author.name))
```