用户登录、登出实现

用户登录、登出实现
1. 用户登录登出扩展库
2. 扩展库初始化
3. 重构用户模型
4. 编写user_loader回调函数
5. 编写登录视图函数
6. 编写登出视图函数
7. 修改基础模板增加登出链接
8. 设置要求用户登录的视图
9. 登录重定向处理
10. 编写首页视图
11. 首页模板修改用户信息展示
12. 编写用户注册表单
13. 编写用户注册模板
14. 登录表单增加新用户注册链接
15. 编写用户注册视图函数
16. 注册首页、登出、注册视图
17. 启动服务

1. 用户登录登出扩展库

涉及用户登录、登出时，使用的Flask扩展库是`Flask-Login`，实现作用如下：

```
pip install Flask-Login
```

- 将用户的 id 储存在 session 中，方便用于 Login/Logout 等流程。
- 让你能够约束用户 Login/Logout 的视图
- 提供 **remember me** 功能
- 保护 cookies 不被篡改

2. 扩展库初始化

扩展库`Flask-Login`（登陆）和`Flask-OpenID`（三方登录）的初始化。

```
......
from flask_login import LoginManager

# 实例化flask_login
```

```
def create_app():
    """应用工厂函数"""
    application = Flask(__name__)
    # 加载config配置文件
    application.config.from_pyfile('config.py', silent=True)

    # 初始化数据库flask_sqlalchemy
    db.init_app(application)

    # 初始化数据库flask_migrate
    import app.models
    migrate.init_app(application, db)

    # 初始化登录扩展flask_login
    lm.init_app(application)
......
```

3. 重构用户模型

修改`app/models.py`文件，重构用户模型，`Flask-Login`扩展需要在用户类中实现一些特定的方法。通过继承Flask-Login提供了`UserMixin`类，可以继承其通用的`is_authenticated`、`is_active`、`is_anonymous`、`get_id`方法。

同时添加`set_password`、`check_password`方法用于密码加密与校验。

- `is_authenticated`：检验User的实例化对象是否登录，登录的话则为True。
- `is_active`：检验用户是否通过某些验证，如果为True，则表示此用户已激活，反之，未激活。
- `is_anonymous`：检验用户是否为匿名用户，如果为False，适用于普通用户，反之则为特殊匿名用户。
- `get_id()`：以字符串形式返回User实例化对象的唯一标识（如果使用Python 2，则返回unicode）。

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

from app import db


class User(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    # index用于添加索引
    # unique用于设置唯一索引
    username = db.Column(db.String(64), index=True, unique=True)
    email = db.Column(db.String(120), index=True, unique=True)
    password_hash = db.Column(db.String(128))
    # 初始化db.relationship
    # 并非实际的数据库字段，只是创建一个虚拟的列，该列会与 Post.user_id (db.ForeignKey) 建立联系
    # 第一个参数表示关联的模型类名；backref用于指定表之间的双向关系；lazy用于定义加载关联对象的方式
    posts = db.relationship('Post', backref='author', lazy='dynamic')

    def set_password(self, raw_password):
```

```
        """密码加密"""
        self.password_hash = generate_password_hash(raw_password)

    def check_password(self, raw_password):
        """校验密码是否正确"""
        return check_password_hash(self.password_hash, raw_password)

    def __repr__(self):
        """打印类对象时的展示方式"""
        return '<User %r>' % self.username
```

4. 编写user_loader回调函数

修改 `app/login.py` 脚本，编写 `user_loader` 回调，用于从数据库加载用户信息，此函数将会被 `Flask-Login` 扩展使用。

```
from app import lm
from app.models import User


@lm.user_loader
def load_user(id):
    """加载用户信息回调"""
    return User.query.get(int(id))
```

5. 编写登录视图函数

修改 `app/login.py` 脚本，编写登录视图函数。其中，`current_user` 是 `Flask-Login` 扩展提供的一个代理对象，视图和模板中均能访问。

```
from flask.views import View
from flask import redirect, render_template, url_for, flash, request
from werkzeug.urls import url_parse
from flask_login import current_user, login_user, logout_user

from app.forms import LoginForm, RegistrationForm
from app import lm, db
from app.models import User


class LoginView(View):
    methods = ['GET', 'POST']

    def dispatch_request(self):
        # 若全局变量中已存在解析出的用户信息，且用户已验证通过，则直接跳转至首页
        if current_user.is_authenticated:
            return redirect(url_for('index'))

        # 加载登录Form表单
        form = LoginForm()
```

```
        # 登录Form表单提交验证通过，跳转至首页
        if form.validate_on_submit():
            # 根据用户名查询本地用户信息
            user = User.query.filter_by(username=form.username.data).first()
            # 用户不存在或密码校验失败，增加闪现消息，并重定向到登录页
            if not user or not user.check_password(form.password.data):
                flash('用户名或密码有误')
                return redirect(url_for('login'))

            # 用户校验通过，进行用户登录，并重定向到首页
            login_user(user, remember=form.remember_me.data)
            next_page = request.args.get('next', '')
            if not next_page or not url_parse(next_page).decode_netloc():
                next_page = url_for('index')
            return redirect(next_page)

        # GET请求，直接展示登录页面
        return render_template('login/login.html', title='登录', form=form)
```

6. 编写登出视图函数

编辑`app/login.py`脚本，新增登出视图类。

```
from flask.views import View
from flask import redirect, url_for
from flask_login import logout_user


class LogoutView(View):
    methods = ['GET']

    def dispatch_request(self):
        logout_user()
        return redirect(url_for('index'))
```

7. 修改基础模板增加登出链接

修改`app/templates/base.html`，增加登出链接。并创建首页模板文件`app/templates/index/index.html`。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    {% if title %}
        <title>{{ title }} - 博客</title>
    {% else %}
        <title>博客</title>
    {% endif %}
</head>
```

```
<body>
    <div>
        博客:
        <a href="{{ url_for('index') }}">首页</a>
        {% if current_user.is_anonymous %}
            <a href="{{ url_for('login') }}">登录</a>
        {% else %}
            <a href="{{ url_for('logout') }}">退出</a>
        {% endif %}
    </div>
    <hr>
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            <ul>
                {% for message in messages %}
                    <li>{{ message }}</li>
                {% endfor %}
            </ul>
        {% endif %}
    {% endwith %}
    {% block content %}{% endblock %}
</body>
</html>
```

8. 设置要求用户登录的视图

修改`app/__init__.py`脚本，增加要求用户登录视图的配置`lm.login_view`，也可以通过`@login_required`装饰器实现此目的。若将此装饰器添加到视图函数Flask 的装饰器`@app.route`下面时，该函数将受到保护，并且不允许未经过身份验证的用户访问。

```
......
def create_app():
    """应用工厂函数"""
......
    # 初始化登录扩展flask_login
    lm.init_app(application)
    lm.login_view = 'login'
    # 重写需要登录提示信息
    lm.login_message = '请登录后访问此页面'
......
```

9. 登录重定向处理

当未登录的用户访问受`@login_required`保护的视图时，装饰器将重定向到登录页面，但它将在此重定向中包含一些额外信息，以便网站可以返回到之前的页面。

例如，如果用户跳转到`/index`，`@login_required`将拦截请求并使用重定向响应`/ login`，但它会向此URL添加一个查询字符串参数，从而形成完整的重定向URL`/login?next=/index`。`next`查询字符串参数设置为之前URL，因此网站可以使用该参数在登录后重定向回之前页面。

修改登录视图`app/login.py`，实现`next`的重定向处理。

```
class LoginView(View):
    methods = ['GET', 'POST']

    def dispatch_request(self):
        ......

            # 用户校验通过，进行用户登录，并重定向到首页
            lm.login_user(user, remember=form.remember_me.data)
            next_page = request.args.get('next', '')
            if not next_page or not url_parse(next_page).decode_netloc():
                next_page = url_for('index')
            return redirect(next_page)
        ......
```

10. 编写首页视图

新增 `app/index.py` 脚本，用于作为首页视图，同时通过 `decorators = [lm.login_required]` 代码限制该视图只能登陆用户访问。

```
from flask.views import View
from flask import render_template
from flask_login import login_required


class IndexView(View):
    methods = ['GET']
    decorators = [login_required]

    def dispatch_request(self):
        posts = [
            {
                'author': {'username': '李白'},
                'body': '举头望明月，低头思故乡'
            },
            {
                'author': {'username': '李清照'},
                'body': '知否，知否，应是绿肥红瘦'
            }
        ]

        return render_template('index/index.html', title='首页', posts=posts)
```

11. 首页模板修改用户信息展示

修改 `app/templates/index/index/html` 文件，修改用户信息展示方式。

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

{% block content %}
    <h1>Hi, {{ current_user.username }}!</h1>
    {% for post in posts %}
        <div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>
    {% endfor %}
{% endblock %}
```

12. 编写用户注册表单

编辑`app/form.py`脚本，新增用户注册表单函数。

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

from app.models import User


class RegistrationForm(FlaskForm):
    username = StringField('用户名', validators=[DataRequired()])
    email = StringField('邮箱', validators=[DataRequired(), Email])
    password = PasswordField('密码', validators=[DataRequired()])
    password2 = PasswordField('确认密码', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('注册')

    def validate_username(self, username):
        """自定义username验证器"""
        user = User.query.filter_by(username=username.data).first()
        if not user:
            raise ValidationError('该用户名已被使用')

    def validate_email(self, email):
        """自定义email验证器"""
        user = User.query.filter_by(email=email.data).first()
        if not user:
            raise ValidationError('该邮箱已被使用')
```

13. 编写用户注册模板

新增`app/templates/login/register.html`文件，编写用户注册模板

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

{% block content %}
    <h1>注册</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
```

```
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
                <span style="color: red;">{{ error }}</span>
            {% endfor %}
        </p>
        <p>
            {{ form.email.label }}<br>
            {{ form.email(size=64) }}<br>
            {% for error in form.email.errors %}
                <span style="color: red;">{{ error }}</span>
            {% endfor %}
        </p>
        <p>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
                <span style="color: red;">{{ error }}</span>
            {% endfor %}
        </p>
        <p>
            {{ form.password2.label }}<br>
            {{ form.password2(size=64) }}<br>
            {% for error in form.password2.errors %}
                <span style="color: red;">{{ error }}</span>
            {% endfor %}
        </p>
        <p>
            {{ form.submit() }}
        </p>
    </form>
{% endblock %}
```

14. 登录表单增加新用户注册链接

修改`app/templates/login/login.html`，增加新用户注册链接。

```
......
        <p>{{ form.submit() }}</p>
        <p>新用户? <a href="{{ url_for('register') }}">点击注册! </a></p>
    </form>
......
```

15. 编写用户注册视图函数

修改`app/login.py`脚本，新增用户注册视图函数。

```
from flask.views import View
from flask import redirect, render_template, url_for, flash, request
from werkzeug.urls import url_parse
from flask_login import current_user, login_user, logout_user
```

```
from app.forms import LoginForm, RegistrationForm
from app import lm, db
from app.models import User


class RegisterView(View):
    methods = ['GET', 'POST']

    def dispatch_request(self):
        if current_user.is_authenticated:
            return url_for('index')

        form = RegistrationForm()
        # 校验成功，创建用户信息，跳转至登录页面
        if form.validate_on_submit():
            user = User(username=form.username.data, email=form.email.data)
            user.set_password(form.password.data)
            db.session.add(user)
            db.session.commit()
            flash('祝贺，你现在已成为一个注册用户！')
            return redirect(url_for('login'))

        return render_template('login/register.html', title='注册', form=form)
```

16. 注册首页、登出、注册视图

修改`app/__init__.py`注册首页、登出、注册视图。

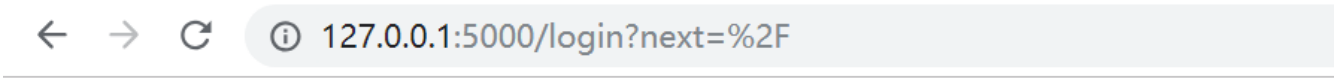
```
......
def create_app():
    """应用工厂函数"""
......
    # 注册Index首页视图URL
    from app.index import IndexView
    application.add_url_rule('/', view_func=IndexView.as_view('index'))

    # 注册Logout登出视图URL
    from app.login import LogoutView
    application.add_url_rule('/logout', view_func=LogoutView.as_view('logout'))

    # 注册Register注册视图URL
    from app.login import RegisterView
    application.add_url_rule('/register', view_func=RegisterView.as_view('register'))
......
```

17. 启动服务

首页跳转登录页面



博客： 首页 登录

- 请登录后访问此页面

登录

用户名

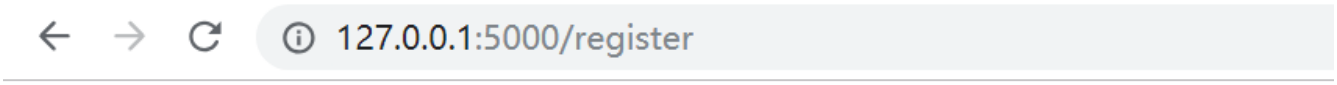
密码

记住我

登录

新用户？ 点击注册！

注册页面



博客： 首页 登录

注册

用户名

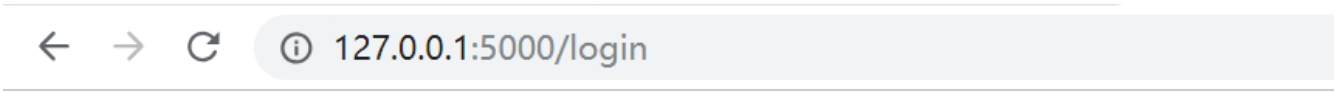
邮箱

密码

确认密码

注册

注册成功，跳转至登录页面



博客： 首页 登录

- 祝贺，你现在已成为一个注册用户！

登录

用户名

密码

记住我

登录

新用户？ 点击注册！

登录成功，跳转至首页

博客： 首页 退出

Hi, admin1!

李白 says: **举头望明月，低头思故乡**

李清照 says: **知否，知否，应是绿肥红瘦**