

用户个人资料页实现

用户个人资料页实现

1. 编写用户资料视图函数
2. 注册用户蓝图
 - 2.1. 创建用户蓝图并注册用户资料视图
 - 2.2. 注册用户蓝图
3. 编写用户资料页模板
4. 修改base页面增加用户资料链接
5. 添加用户头像
 - 5.1. 安装Flask-Avatars扩展库
 - 5.2. 注册Flask-Avatars扩展库
 - 5.3. 创建生成用户头像的模板的环境处理器
 - 5.4. 用户资料模板中使用注册方法获取用户头像
 - 5.5. 启动服务查看结果
6. 提取帖子模板
 - 6.1. 创建子模板
 - 6.2. 用户资料模板调用子模板
 - 6.3. 首页模板修改为调用子模板
 - 6.4. 启动服务查看结果
7. 扩展用户资料信息
 - 7.1. 扩展用户模型
 - 7.2. 数据库迁移
 - 7.3. 用户资料模板增加展示项
 - 7.4. 注册周期函数实现最后访问时间的更新
8. 新增资料编辑功能
 - 8.1. 新增用户资料编辑表单
 - 8.2. 新增用户资料编辑模板
 - 8.3. 新增用户资料编辑视图
 - 8.4. 用户模板增加用户资料编辑链接
 - 8.5. 启动服务查看结果

1. 编写用户资料视图函数

新建 `app/user.py` 脚本，编写用户资料视图函数。其中通过 `decorators = [login_required]` 设置只能登录用户访问，查询时通过 `first_or_404()` 可以实现查询不到数据则自动报404错误。

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

from app.models import User
```

```
class UserInfoView(View):
    """用户信息查询"""
    methods = ['GET']
    decorators = [login_required]

    def dispatch_request(self, username):
        user = User.query.filter_by(username=username).first_or_404()
        posts = [
            {'author': user, 'body': '测试1'},
            {'author': user, 'body': '测试2'},
        ]
        return render_template('user/user_info.html', user=user, posts=posts)
```

2. 注册用户蓝图

新增用户蓝图注册，并将用户资料视图类注册到蓝图中。

2.1. 创建用户蓝图并注册用户资料视图

修改 `app/user.py` 脚本，创建用户蓝图，并将用户资料视图注册到用户蓝图上。

```
from flask import Blueprint

# 创建用户蓝图
bp = Blueprint('user', __name__, url_prefix='/user')

# 将用户资料视图注册到用户蓝图上
bp.add_url_rule('/info/<username>', view_func=UserInfoView.as_view('user_info'))
```

2.2. 注册用户蓝图

修改 `app/__init__.py` 脚本，新增用户蓝图注册。

```
def create_app():
    .....
    # 用户蓝图注册
    from app import user
    application.register_blueprint(user.bp)
    .....
```

3. 编写用户资料页模板

新增 `app/templates/user/user_info.html` 文件，用户展示用户资料信息。

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

{% block content %}
    <h1>用户: {{ user.username }}</h1>
    <hr>
    {% for post in posts %}
    <p>
        {{ post.author.username }} says: <b>{{ post.body }}</b>
    </p>
    {% endfor %}
{% endblock %}
```

4. 修改base页面增加用户资料链接

修改 `app/templates/base.html` 文件，新增用户资料链接。通过 `current_user.username` 获取当前登录用户的用户名，并通过URL传递参数。

```
.....
<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('user.user_info', username=current_user.username) }}">个人资
料</a>
            <a href="{{ url_for('logout') }}">退出</a>
        {% endif %}
    </div>
.....
```

5. 添加用户头像

5.1. 安装Flask-Avatars扩展库

用户头像功能，可以通过安装 `Flask-Avatars` 扩展库。

```
pip install Flask-Avatars
```

5.2. 注册Flask-Avatars扩展库

修改 `app/__init__.py`，新增 `Flask-Avatars` 扩展库注册。

```

from flask_avatars import Avatars

# 实例化flask_avatars
avatars = Avatars()

def create_app():
    .....
    # 初始化flask_avatars
    avatars.init_app(application)
    .....

```

5.3. 创建生成用户头像的模板的环境处理器

生成用户头像，当前项目使用 `gravatar` 头像，生成该头像就需要传入邮箱的hash值。修改 `app/__init__.py` 脚本，增加 `email_hash` 的计算，并通过 `email_hash` 获取头像的方法，并将其注册到环境处理其中。模板的环境处理器可以将变量、函数加入到模板环境中，传入后可以在模板中直接调用。

```

def create_app():
    .....
    @application.context_processor
    def utility_processor():
        """模板环境处理器注册"""
        def get_avatars(email, *args, **kwargs):
            """根据用户邮箱获取用户头像"""
            import hashlib
            email_hash = hashlib.md5(email.lower().encode('utf-8')).hexdigest()
            return avatars.gravatar(email_hash, *args, **kwargs)
        return dict(get_avatars=get_avatars)

    return application

```

5.4. 用户资料模板中使用注册方法获取用户头像

修改 `app/templates/user/user_info.html` 文件，增加用户头像展示。通过 `get_avatars(user.email, 64)` 获取用户头像地址。第一个参数传入用户的email地址，第二个参数用于设置头像图片大小。

```

{% extends 'base.html' %}

{% block content %}
    <table>
        <tr valign="top">
            <td></td>
            <td><h1>用户: { { user.username }}</h1></td>
        </tr>
    </table>
    <hr>
    {% for post in posts %}
    <table>
        <tr valign="top">
            <td></td>

```

```
<td>{{ post.author.username }} says:<br>{{ post.body }}</td>
</tr>
</table>
{% endfor %}
{% endblock %}
```

5.5. 启动服务查看结果

← → ↻ ⓘ 127.0.0.1:5000

博客: [首页](#) [个人资料](#) [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

← → ↻ ⓘ 127.0.0.1:5000/user/info/admin

博客: [首页](#) [个人资料](#) [退出](#)



用户: admin



admin says:
测试1



admin says:
测试2

6. 提取帖子模板

6.1. 创建子模板

由于首页、用户资料页均需要展示帖子信息，而且格式一致，所以考虑将帖子展示部分提取出子模板进行共用。新建 `app/templates/common/post.html` 文件，用于编写共用的帖子展示模板。

```
<table>
  <tr valign="top">
    <td></td>
    <td>{{ post.author.username }} says:<br>{{ post.body }}</td>
  </tr>
</table>
```

6.2. 用户资料模板调用子模板

修改 `app/templates/user/user_info.html` 模板文件，修改为通过 `include` 调用子模板方式。

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

{% block content %}
  <table>
    <tr valign="top">
      <td></td>
      <td><h1>用户: {{ user.username }}</h1></td>
    </tr>
  </table>
  <hr>
  {% for post in posts %}
    {% include 'common/post.html' %}
  {% endfor %}
{% endblock %}
```

6.3. 首页模板修改为调用子模板

修改 `app/templates/index/index.html` 模板文件，修改为调用子模板方式。

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

{% block content %}
  <h1>Hi, {{ current_user.username }}!</h1>
  {% for post in posts %}
    {% include 'common/post.html' %}
  {% endfor %}
{% endblock %}
```

修改 `app/index.py` 脚本，修改测试数据。

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

    def dispatch_request(self):
        posts = [
```

```
{
  'author': {'username': '李白', 'email': 'lib@126.com'},
  'body': '举头望明月，低头思故乡'
},
{
  'author': {'username': '李清照', 'email': 'liqz@126.com'},
  'body': '知否，知否，应是绿肥红瘦'
}
]

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

6.4. 启动服务查看结果

← → ↻ ⓘ 127.0.0.1:5000

博客: [首页](#) [个人资料](#) [退出](#)

Hi, admin!



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

← → ↻ ⓘ 127.0.0.1:5000/user/info/admin

博客: [首页](#) [个人资料](#) [退出](#)



用户: admin



admin says:

测试1



admin says:

测试2

7. 扩展用户资料信息

7.1. 扩展用户模型

在 `app/models.py` 用户模型中增加关于自己、最后访问时间等字段，用于记录用户的个人介绍和登录情况。

```
class User(UserMixin, db.Model):
    .....
    # 个人介绍
    about_me = db.Column(db.String(140))
    # 最后访问时间
    last_seen = db.Column(db.DateTime, default=datetime.timezone)
    .....
```

7.2. 数据库迁移

```
flask db migrate -m '新增about_me、last_seen字段'
flask db upgrade
```

7.3. 用户资料模板增加展示项

修改 `app/templates/user/user_info.html` 文件，增加 `about_me`、`last_seen` 展示。

```
.....
<td></td>
<td>
    <h1>用户: { { user.username } }</h1>
    {% if user.about_me %}
        <p>{ { user.about_me } }</p>
    {% endif %}
    {% if user.last_seen %}
        最后访问于: <p>{ { user.last_seen } }</p>
    {% endif %}
</td>
.....
```

7.4. 注册周期函数实现最后访问时间的更新

修改 `app/__init__.py` 脚本，增加 `@before_request` 周期函数的处理，用于在每次请求时，登记其最后访问时间字段。


```
def create_app():
    .....
    @application.before_request
    def before_request():
        """请求前周期函数"""
        # 用户已登录则登记用户请求时间
        if current_user.is_authenticated:
            current_user.last_seen = datetime.timezone
            db.session.commit()

    return application
```

8. 新增资料编辑功能

8.1. 新增用户资料编辑表单

修改 `app/forms.py` 脚本，增加用户资料编辑表单。

```
class UserInfoEditForm(FlaskForm):
    """用户信息编辑表单"""
    username = StringField('用户名', validators=[DataRequired()])
    about_me = TextAreaField('个人简介', validators=[Length(min=0, max=140)])
    submit = SubmitField('提交')
```

8.2. 新增用户资料编辑模板

新增 `app/templates/user/user_info_edit.html` 文件，新建用户资料修改模板。

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

{% block content %}
    <h1>个人资料编辑</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.about_me.label }}<br>
            {{ form.about_me(cols=50, rows=4) }}<br>
            {% for error in form.about_me.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>{{ form.submit() }}</p>
    </form>
```

```
{% endblock %}
```

8.3. 新增用户资料编辑视图

修改 `app/user.py` 文件，新建用户资料修改视图，并注册到用户蓝图中。

```
class UserInfoEditView(View):
    """用户资料编辑"""
    methods = ['GET', 'POST']
    decorators = [login_required]

    def dispatch_request(self):
        form = UserInfoEditForm()
        # 验证通过更新当前登录用户信息
        if form.validate_on_submit():
            current_user.username = form.username.data
            current_user.about_me = form.about_me.data
            db.session.commit()
            flash('您的修改已保存')
            return redirect(url_for('user_info_edit'))
        elif request.method == 'GET':
            # 查询是获取当前用户的信息
            form.username.data = current_user.username
            form.about_me.data = current_user.about_me
            return render_template('user/user_info_edit.html', title='个人信息编辑', form=form)

# 将用户资料修改视图注册到用户蓝图上
bp.add_url_rule('/edit', view_func=UserInfoEditView.as_view('user_info_edit'))
```

8.4. 用户模板增加用户资料编辑链接

修改 `app/templates/user/user_info.html` 文件，新增用户资料编辑链接。

```
.....
        {% if user.last_seen %}
            最后访问于: <p>{{ user.last_seen }}</p>
        {% endif %}
        {% if user == current_user %}
            <p><a href="{{ url_for('user.user_info_edit') }}">编辑您的资料</a></p>
        {% endif %}
    </td>
.....
```

8.5. 启动服务查看结果

博客: [首页](#) [个人资料](#) [退出](#)





用户: admin

最后访问于:

2019-04-26 07:58:39.343264

[编辑您的信息](#)

 admin says:
测试1

 admin says:
测试2

博客: [首页](#) [个人资料](#) [退出](#)

个人资料编辑

用户名

admin

个人简介

这是测试信息

提交

博客: [首页](#) [个人资料](#) [退出](#)

- 您的修改已保存

个人资料编辑

用户名

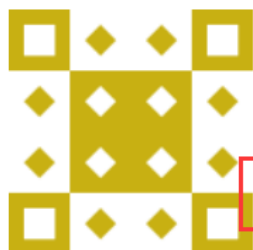
admin

个人简介

这是测试信息

提交

博客: [首页](#) [个人资料](#) [退出](#)




用户: admin


这是测试信息

最后访问于:

2019-04-26 08:01:25.496628

[编辑您的资料](#)

 admin says:
测试1

 admin says:
测试2