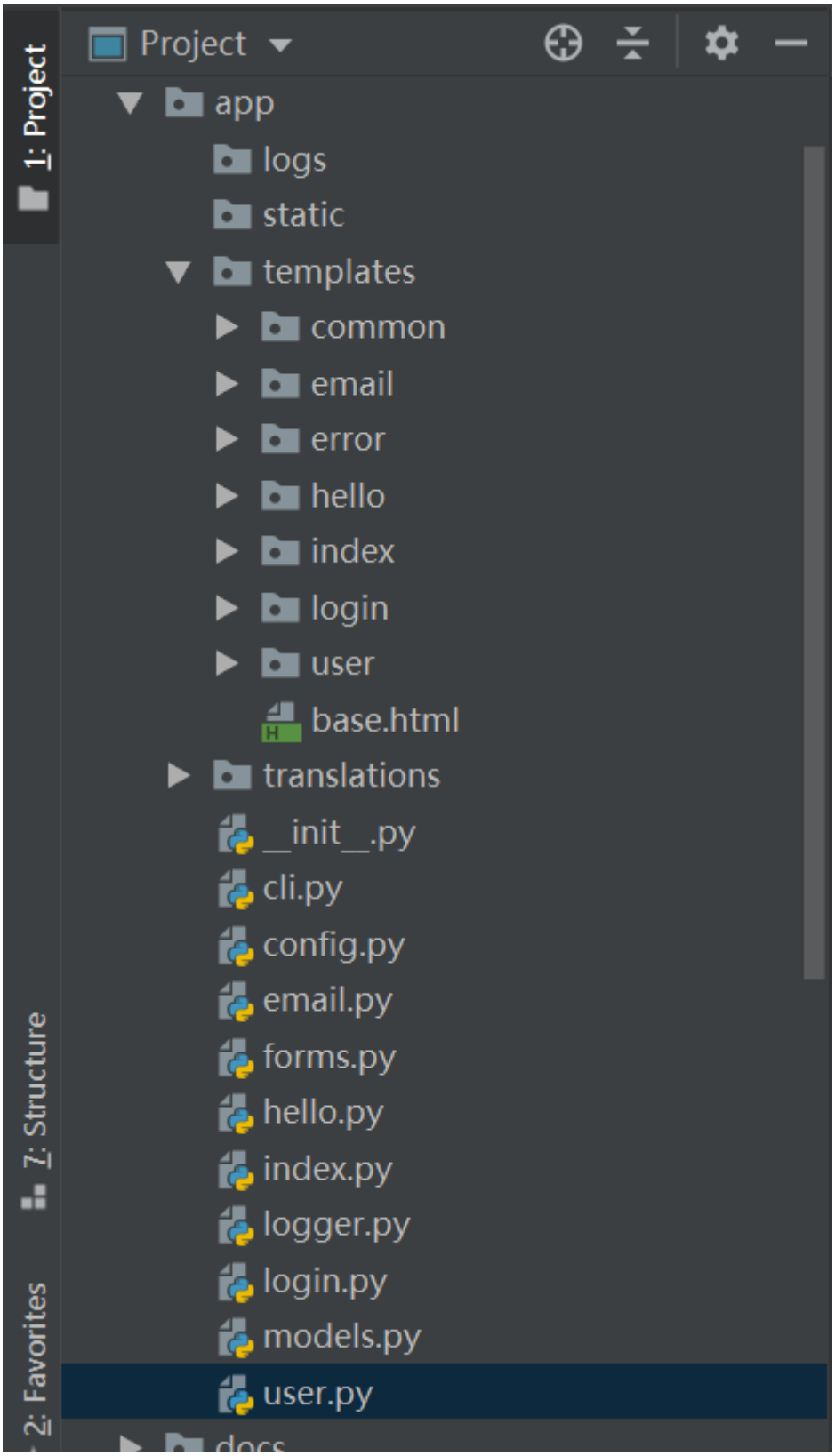
优化应用结构及全文搜索

1. 优化应用结构分析

目前应用结构在小型项目中还是比较清晰的，但是在大型项目中，如果能把项目才分成几个子应用的方式会更加便于管理和维护，也便于对于其他项目的代码复用。类似于`user`模块一样，每个子系统单独作为一个蓝图进行管理。所以经过分析，主要可以按照功能点的方式将应用分为以下几类子应用：

- 用户认证子应用
- 错误子应用
- 核心应用子应用

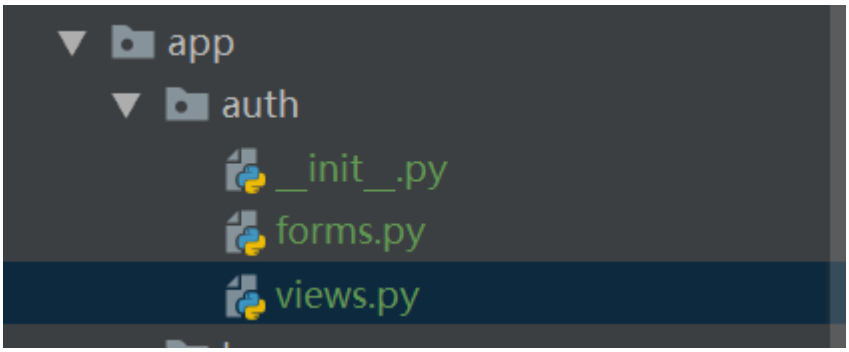


对于以上划分的三个子应用，每个子应用为单独的模块，互不影响，并且都有自己的`forms.py`、`views.py`文件，并且通过`__init__.py`脚本进行蓝图注册。

2. 用户认证子应用优化

2.1. 创建用户认证子应用模块

新建`app/auth`模块，并在模块目录中创建`app/auth/forms.py`和`app/auth/views.py`文件分别用来放置表单类和视图类。



2.2. 表单类转移

将`app/forms.py`中与用户认证相关的表单类转移至`app/auth/forms.py`中。

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

from app.models import User


class LoginForm(FlaskForm):
    """登录表单"""
    # DataRequired: 数据不可为空的验证器
    username = StringField(_l('用户名'), validators=[DataRequired()])
    password = PasswordField(_l('密码'), validators=[DataRequired()])
    remember_me = BooleanField(_l('记住我'), default=False)
    submit = SubmitField(_l('登录'))


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

    def validate_username(self, username):
        """自定义username验证器"""
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError(_l('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 user:
```

```
            raise ValidationError(_l('该邮箱已被使用'))


class ResetPasswordRequestForm(FlaskForm):
    """重置密码请求表单"""
    email = StringField(_l('邮箱'), validators=[DataRequired(), Email()])
    submit = SubmitField(_l('请求密码重置'))


class ResetPasswordForm(FlaskForm):
    """重置密码表单"""
    password = PasswordField(_l('密码'), validators=[DataRequired()])
    password2 = PasswordField(_l('确认密码'), validators=[DataRequired(),
EqualTo('password')])
    submit = SubmitField(_l('请求密码重置'))
```

2.3. 视图类转移

将`app/user.py`中与用户认证相关的表单类转移至`app/auth/views.py`中。同时将原来存放用户认证相关功能模板文件的`app/templates/login`目录修改为`app/templates/auth`。因此需要修改`render_template()`函数中的模板文件目录层级。

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

from app import db
from app.auth.forms import LoginForm, RegistrationForm, ResetPasswordRequestForm,
ResetPasswordForm
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('auth/login.html', title=_('登录'), form=form)


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

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


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('auth/register.html', title=_('注册'), form=form)


class ResetPasswordRequestView(View):
    """重置密码申请视图"""
    methods = ['GET', 'POST']

    def dispatch_request(self):
        if current_user.is_authenticated:
            return redirect(url_for('index'))
        form = ResetPasswordRequestForm()
        if form.validate_on_submit():
            user = User.query.filter_by(email=form.email.data).first()
            if not user:
                flash(_('该电子邮箱未注册'))
                return redirect(url_for('reset_password_request'))
```

```
            from app.email import send_password_reset_email
            send_password_reset_email(user)
            flash(_('查看您的电子邮箱消息，以重置您的密码'))
            return redirect(url_for('login'))
        return render_template('auth/reset_password_request.html', title=_('重置密码'),
form=form)


class ResetPasswordView(View):
    """重置密码视图"""
    methods = ['GET', 'POST']

    def dispatch_request(self, token):
        if current_user.is_authenticated:
            return redirect(url_for('index'))
        user = User.verify_jwt_token(token)
        if not user:
            return redirect(url_for('index'))
        form = ResetPasswordForm()
        if form.validate_on_submit():
            user.set_password(form.password.data)
            db.session.commit()
            flash(_('您的密码已被重置'))
            return redirect(url_for('login'))
        return render_template('auth/reset_password.html', form=form)
```

2.4. 创建用户认证蓝图

修改 `app/auth/__init__.py` 文件，创建用户认证蓝图，并将所有视图类进行注册。

```
from flask import Blueprint


def register_views(bp):
    """注册视图类"""
    # 在函数中引入可以避免循环依赖问题
    from app.auth.views import LoginView, LogoutView, RegisterView,
ResetPasswordRequestView, ResetPasswordView
    # 登录视图
    bp.add_url_rule('/login', view_func=LoginView.as_view('login'))
    # 退出视图
    bp.add_url_rule('/logout', view_func=LogoutView.as_view('logout'))
    # 注册视图
    bp.add_url_rule('/register', view_func=RegisterView.as_view('register'))
    # 申请重置密码视图
    bp.add_url_rule('/reset_password_request',
view_func=ResetPasswordRequestView.as_view('reset_password_request'))
    # 重置密码视图
    bp.add_url_rule('/reset_password',
view_func=ResetPasswordView.as_view('reset_password'))
```

```
# 创建用户认证蓝图
bp = Blueprint('auth', __name__)
# 注册视图类
register_views(bp)
```

2.5. 注册用户认证蓝图

修改`app/__init__.py`文件，注册用户认证蓝图。

```
def create_app(test_config=None):
......
    # 用户认证蓝图注册
    from app import auth
    app.register_blueprint(auth.bp, url_prefix='/auth')
```

2.6. 修改视图类中url_for参数

修改`app/auth/views.py`文件，修改`url_for()`的第一个参数，原本是视图函数名称，但当在blueprint中定义路由时，该参数必须包含blueprint名称和视图函数名称，并以句点分隔。

同时包括`app/templates/auth`目录中的模板文件以及`app/templates/base.html`基础模板文件，其内部的`url_for()`也需要修改。

```
class LoginView(View):
......
            # 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('auth.login'))
......


class RegisterView(View):
......
            return redirect(url_for('auth.login'))

        return render_template('auth/register.html', title=_('注册'), form=form)


class ResetPasswordRequestView(View):
......
                return redirect(url_for('auth.reset_password_request'))

            from app.email import send_password_reset_email
            send_password_reset_email(user)
            flash(_('查看您的电子邮箱消息，以重置您的密码'))
            return redirect(url_for('auth.login'))
        return render_template('auth/reset_password_request.html', title=_('重置密码'),
form=form)


class ResetPasswordView(View):
```

```
.....
            return redirect(url_for('auth.login'))
        return render_template('auth/reset_password.html', form=form)
```

2.7. 重置密码发送邮件函数转移

新建 `app/auth/email.py` 脚本，将 `app/email.py` 中 `send_password_reset_email()` 函数转移到该脚本，用于区分业务逻辑。

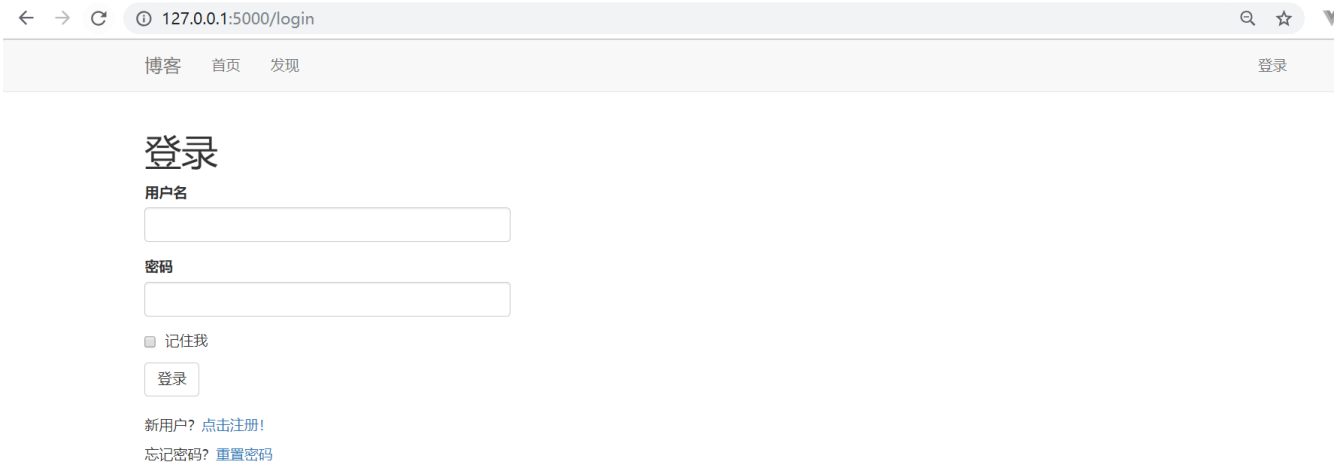
```
from flask import current_app, render_template
from flask_babel import _

from app.email import send_email


def send_password_reset_email(user):
    """发送密码重置电子邮件"""
    token = user.get_jwt_token()
    send_email(_('[博客] 重置您的密码'),
               sender=current_app.config['MAIL_USERNAME'],
               recipients=[user.email],
               text_body=render_template('email/reset_password.txt', user=user,
token=token),
               html_body=render_template('email/reset_password.html', user=user,
token=token))
```

修改 `app/auth/views.py` 脚本，修改 `send_password_reset_email()` 函数引入。

```
class ResetPasswordRequestView(View):
......
            from app.auth.email import send_password_reset_email
            send_password_reset_email(user)
            flash(_('查看您的电子邮箱消息，以重置您的密码'))
            return redirect(url_for('auth.login'))
        return render_template('auth/reset_password_request.html', title=_('重置密码'),
form=form)
```

2.8. 启动服务测试

直接访问登录页面可以正常运行。



3. 错误子应用优化

错误子应用优化与用户认证子应用优化类似，也需要新建 `app/errors` 模块，并在模块目录中创建 `app/errors/handlers.py` 文件用来放置错误页面定义。

```
from flask import render_template

from app.errors import bp


@bp.errorhandler(404)
def not_found_error(error):
    return render_template('errors/404.html'), 404


@bp.errorhandler(500)
def internal_error(error):
    return render_template('errors/500.html'), 500
```

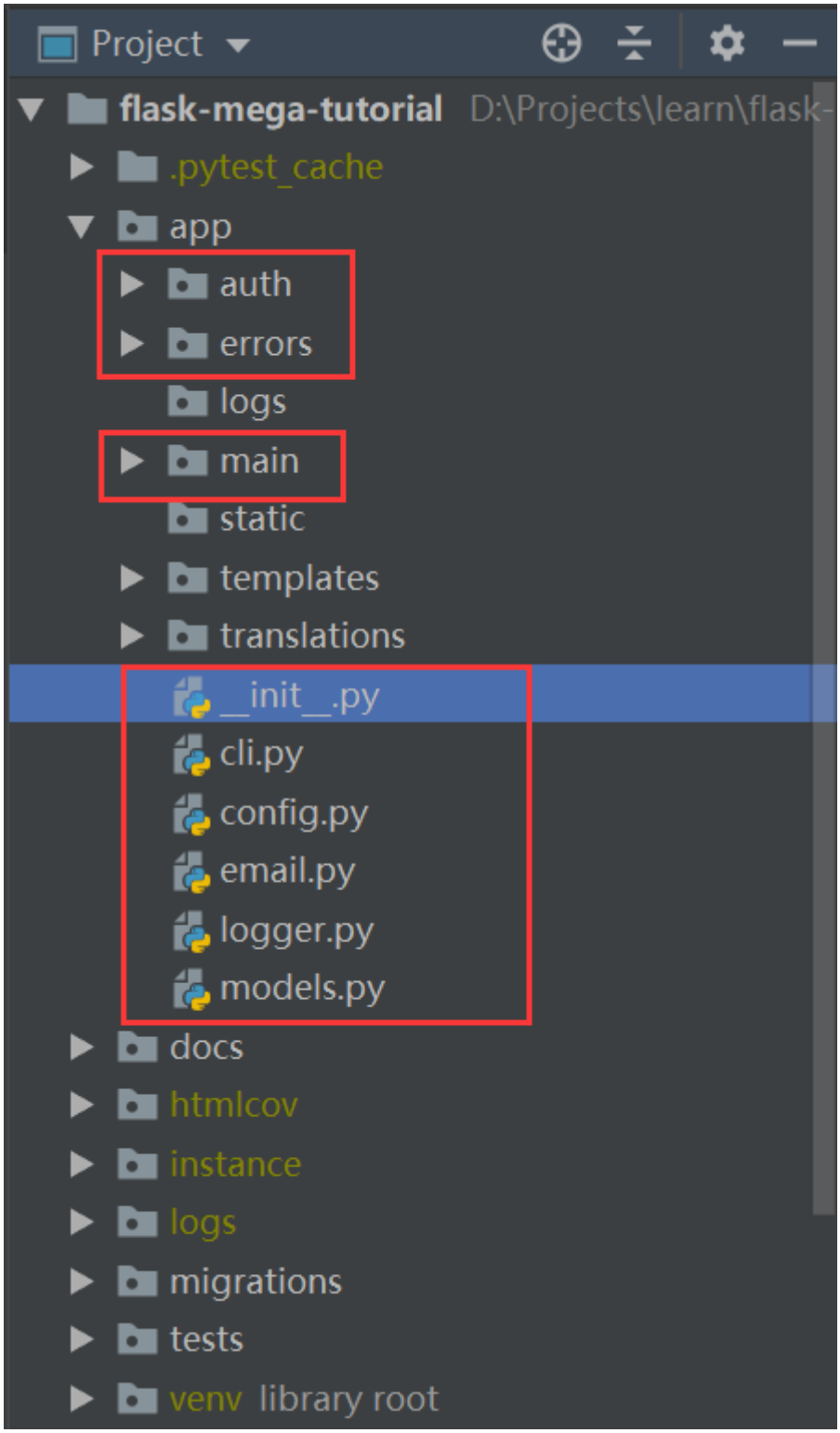
修改 `app/errors/__init__.py` 脚本，创建错误子应用蓝图。

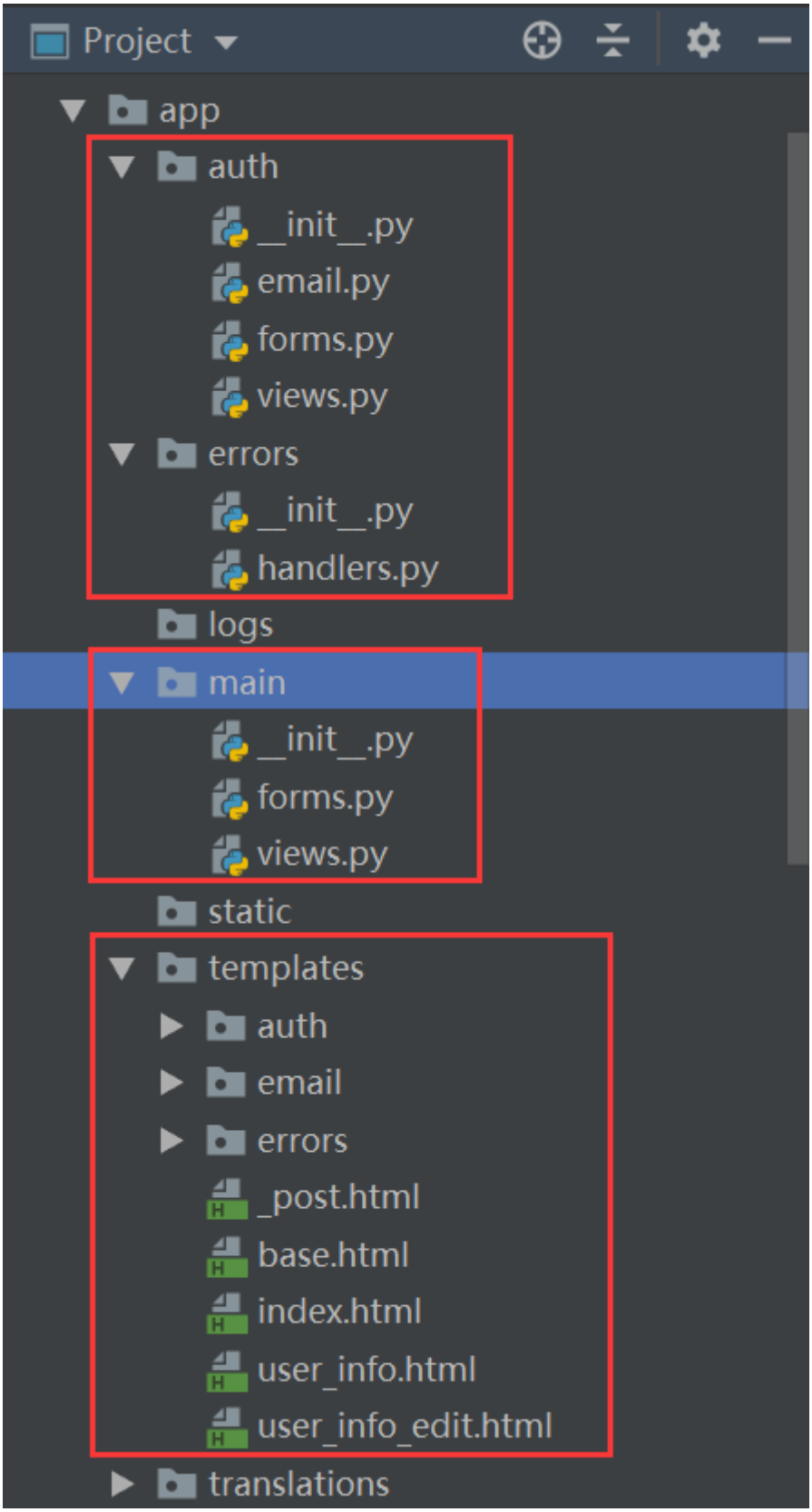
```
from flask import Blueprint

bp = Blueprint('errors', __name__)

from app.errors import handlers
```

4. 核心子应用优化

错误子应用优化与用户认证子应用优化类似，也需要创建自己的 `forms.py` 、 `views.py` 文件，并且通过 `__init__.py` 脚本进行蓝图注册。

最终调整后目录结构如下：


Project
flask-mega-tutorial D:\Projects\learn\flask-
.pytest_cache
app
auth
errors
logs
main
static
templates
translations
__init__.py
cli.py
config.py
email.py
logger.py
models.py
docs
htmlcov
instance
logs
migrations
tests
venv library root



5. 环境依赖包信息导出

通过`pip freeze`命令导出依赖包信息。

```
pip freeze > requirements.txt
```

通过`pip install -r`命令可以根据导出的依赖包信息统一安装。

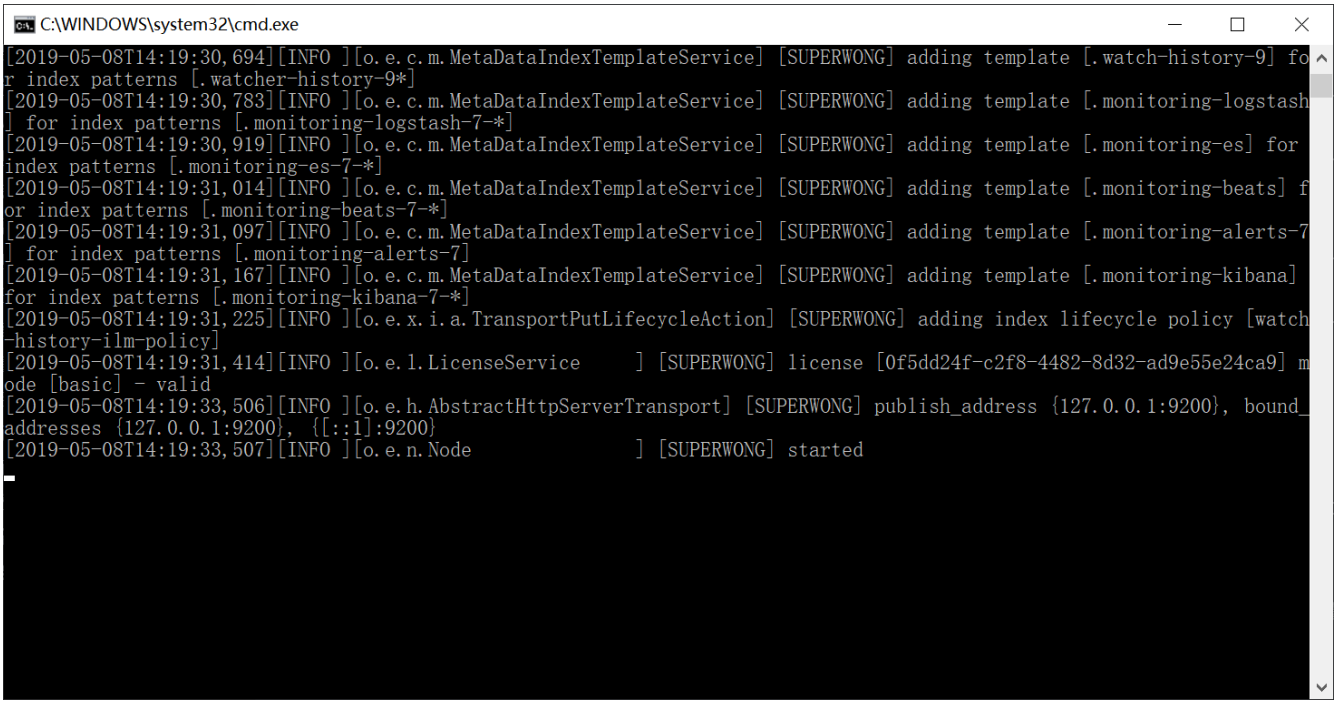
```
pip install -r requirements.txt
```

6. 实现全文搜索

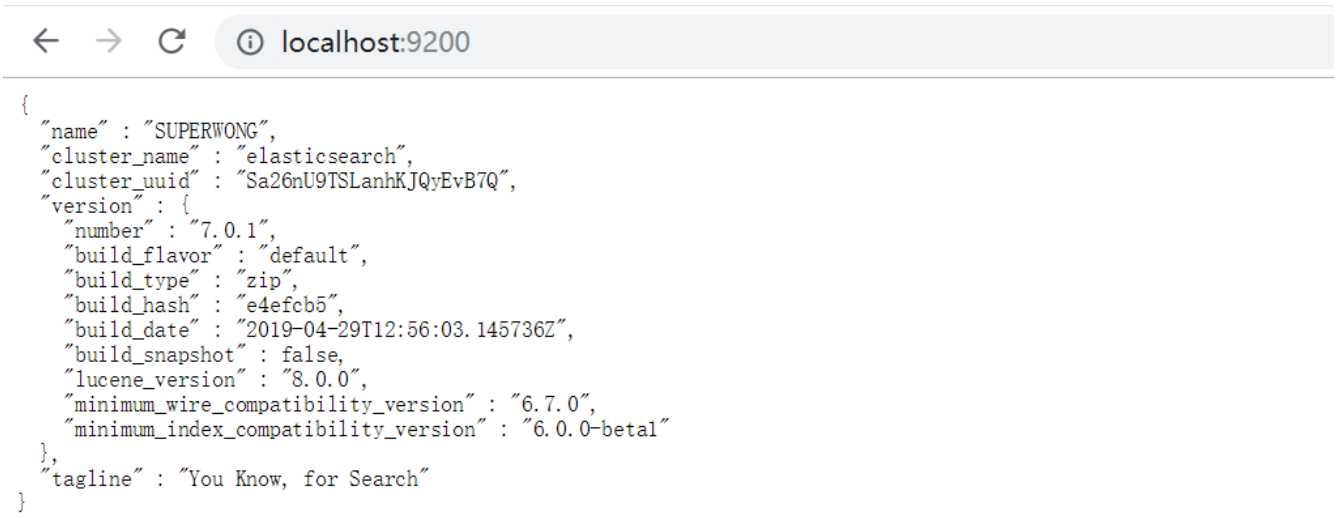
6.1. 全文搜索引擎安装

对于全文搜索的实现，需要借助于搜索引擎来完成。依据流行程度选择使用`Elasticsearch`。安装`Elasticsearch`需要下载二进制文件进行自行安装。[下载地址]

windows系统下载完毕后，将二进制打包文件解压，并执行`bin/elasticsearch.bat`即可启动搜索引擎服务。



通过浏览器访问`http://localhost:9200`地址验证是否启动成功。



若要通过Python来管理Elasticsearch，就需要安装客户端三方库。

```
pip install elasticsearch
```

更新*requirements.txt*文件

```
pip freeze > requirements.txt
```

6.2. Elasticsearch的基本使用

创建一个`Elasticsearch`类的实例，并将连接URL作为参数传递，从而建立与Elasticsearch的连接。

```
>>> from elasticsearch import Elasticsearch
>>> es = Elasticsearch('http://localhost:9200')
```

Elasticsearch中的数据需要被写入*索引*中。 与关系数据库不同，数据只是一个JSON对象。 以下示例将一个包含`text`字段的对象写入名为`my_index`的索引：

```
>>> es.index(index='my_index', doc_type='my_index', id=1, body={'text': 'this is a test'})
{'_index': 'my_index', '_type': 'my_index', '_id': '1', '_version': 1, 'result': 'created',
'_shards': {'total': 2, 'successful': 1, 'failed': 0}, '_seq_no': 0, '_primary_term': 1}

>>> es.index(index='my_index', doc_type='my_index', id=2, body={'text': 'a second test'})
{'_index': 'my_index', '_type': 'my_index', '_id': '2', '_version': 1, 'result': 'created',
'_shards': {'total': 2, 'successful': 1, 'failed': 0}, '_seq_no': 1, '_primary_term': 1}
```

搜索`this test`，其会根据录入的查询条件进行匹配，匹配度越高，返回的`max_score`值越高：

```
>>> es.search(index='my_index', body={'query': {'match': {'text': 'this test'}}})
{'took': 6, 'timed_out': False, '_shards': {'total': 1, 'successful': 1, 'skipped': 0,
'failed': 0}, 'hits': {'total': {'value': 2, 'relation': 'eq'}, 'max_score': 0.82712996,
'hits': [{'_index': 'my_index', '_type': 'my_index', '_id': '1', '_score': 0.82712996,
'_source': {'text': 'this is a test'}}, {'_index': 'my_index', '_type': 'my_index', '_id':
'2', '_score': 0.19363809, '_source': {'text': 'a second test'}}]}}
```

返回结果格式化：

```
{
    'took': 6,
    'timed_out': False,
    '_shards': {
        'total': 1,
        'successful': 1,
        'skipped': 0,
        'failed': 0
    },
    'hits': {
        'total': {
            'value': 2,
```

```
            'relation': 'eq'
        },
        'max_score': 0.82712996,
        'hits': [{
            '_index': 'my_index',
            '_type': 'my_index',
            '_id': '1',
            '_score': 0.82712996,
            '_source': {
                'text': 'this is a test'
            }
        }, {
            '_index': 'my_index',
            '_type': 'my_index',
            '_id': '2',
            '_score': 0.19363809,
            '_source': {
                'text': 'a second test'
            }
        }]
    }
}
```

删除索引，对于不存在的索引在查询是会报异常：

```
>>> es.indices.delete('my_index')
{'acknowledged': True}
>>> es.search(index='my_index', body={'query': {'match': {'text': 'this test'}}})
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "D:\Projects\learn\flask-mega-tutorial\venv\lib\site-
packages\elasticsearch\client\utils.py", line 84, in _wrapped
    return func(*args, params=params, **kwargs)
  File "D:\Projects\learn\flask-mega-tutorial\venv\lib\site-
packages\elasticsearch\client\__init__.py", line 811, in search
    "GET", _make_path(index, "_search"), params=params, body=body
  File "D:\Projects\learn\flask-mega-tutorial\venv\lib\site-
packages\elasticsearch\transport.py", line 318, in perform_request
    status, headers_response, data = connection.perform_request(method, url, params, body,
headers=headers, ignore=ignore, timeout=timeout)
  File "D:\Projects\learn\flask-mega-tutorial\venv\lib\site-
packages\elasticsearch\connection\http_urllib3.py", line 239, in perform_request
    self._raise_error(response.status, raw_data)
  File "D:\Projects\learn\flask-mega-tutorial\venv\lib\site-
packages\elasticsearch\connection\base.py", line 131, in _raise_error
    raise HTTP_EXCEPTIONS.get(status_code, TransportError)(status_code, error_message,
additional_info)
elasticsearch.exceptions.NotFoundError: NotFoundError(404, 'index_not_found_exception', 'no
such index [my_index]', my_index, index_or_alias)
```

6.3. Elasticsearch配置

修改`app/config.py`脚本，增加Elasticsearch的连接URL参数。

```
# Elasticsearch连接URL，从环境变量获取
ELASTICSEARCH_URL = os.environ.get('ELASTICSEARCH_URL')
```

修改`app/__init__.py`文件，在应用实例中添加`elasticsearch`属性。

```
from elasticsearch import Elasticsearch

def create_app(test_config=None):
    ......
    # ------添加Elasticsearch属性------- #
    app.elasticsearch = Elasticsearch([app.config['ELASTICSEARCH_URL']]) if
app.config['ELASTICSEARCH_URL'] else None
    return app
```

6.4. 全局搜索抽象化

为了可以使用通用的方式来指定哪个模型以及其中的某个或某些字段将被索引。 设定任何需要索引的模型都需要定义一个`__searchable__`属性，它列出了需要包含在索引中的字段。修改`app/models.py`：为Post模型添加一个`__searchable__`属性。

```
class Post(db.Model):
    __tablename__ = 'posts'
    __searchable__ = ['body']
......
```

新建`app/search.py`脚本，用于实现`Elasticsearch`的增删查。

```
from flask import current_app

def add_to_index(index, model):
    """新增、修改索引"""
    # 未初始化elasticsearch实例，不进行处理
    if not current_app.elasticsearch:
        return

    # 获取需要检索的字段
    payload = {}
    for field in model.__searchable__:
        payload[field] = getattr(model, field)
    # 将字段信息直接存入elasticsearch
    current_app.elasticsearch.index(index=index, doc_type=index, id=model.id, body=payload)


def remove_from_index(index, model):
    """删除索引"""
    if not current_app.elasticsearch:
        return
    current_app.elasticsearch.delete(index=index, id=model.id, doc_type=index)
```

```
def query_index(index, query, page, per_page):
    """搜索"""
    if not current_app.elasticsearch:
        return [], 0
    search = current_app.elasticsearch.search(
        index=index,
        body={'query': {'multi_match': {'query': query, 'fields': ['*']}},
              'from': (page-1)*per_page,
              'size': per_page
              }
    )
    ids = [int(hit['_id']) for hit in search['hits']['hits']]
    return ids, search['hits']['total']['value']
```

6.5. 集成SQLAlchemy到搜索

修改`app/models.py`脚本，新增查询mixin类，用于全局搜索集成到数据库模型中，实现自动管理与SQLAlchemy模型关联的全文索引。

```
class SearchableMixin(object):
    @classmethod
    def search(cls, expression, page, per_page):
        """搜索"""
        ids, total = query_index(cls.__tablename__, expression, page, per_page)
        if total == 0:
            return cls.query.filter_by(id=0), 0
        when = []
        for i in range(len(ids)):
            when.append((ids[i], i))
        # 根据查询出的对象ID匹配对象信息
        # CASE语句，用于确保查询出的数据库中的结果与给定ID的顺序相同
        return cls.query.filter(cls.id.in_(ids)).order_by(
            db.case(when, value=cls.id)
        ), total

    @classmethod
    def before_commit(cls, session):
        """会话提交前，记录对象变更"""
        session._changes = {
            'add': [obj for obj in session.new if isinstance(obj, cls)],
            'update': [obj for obj in session.dirty if isinstance(obj, cls)],
            'delete': [obj for obj in session.deleted if isinstance(obj, cls)]
        }

    @classmethod
    def after_commit(cls, session):
        """会话提交后，根据记录的变更同步elasticsearch"""
        for obj in session._changes['add']:
            add_to_index(cls.__tablename__, obj)
        for obj in session._changes['update']:
```

```
            add_to_index(cls.__tablename__, obj)
        for obj in session._changes['delete']:
            remove_from_index(cls.__tablename__, obj)
        session._changes = None

    @classmethod
    def reindex(cls):
        """用于初始化数据库已有数据"""
        for obj in cls.query:
            add_to_index(cls.__tablename__, obj)
```

修改`app/models.py`脚本，Post模型继承`SearchableMixin`类，并且还需要监听提交之前和之后的事件。

```
class Post(SearchableMixin, db.Model):
    __tablename__ = 'posts'
    __searchable__ = ['body']
......

# 增加监听Post模型的提交事件
db.event.listen(db.session, 'before_commit', Post.before_commit)
db.event.listen(db.session, 'after_commit', Post.after_commit)
```

执行`Post.reindex()`初始化`Post`数据库中的数据，同步到elasticsearch。

```
(venv) D:\Projects\learn\flask-mega-tutorial>set ELASTICSEARCH_URL=http://localhost:9200

(venv) D:\Projects\learn\flask-mega-tutorial>python
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from app.models import Post
>>> from app import create_app
>>> app = create_app()
[2019-05-08 17:23:03,623] INFO in logger: 博客已启动
>>> app.app_context().push
<bound method AppContext.push of <flask.ctx.AppContext object at 0x000002567796FFD0>>
>>> with app.app_context():
...     Post.reindex()
...     query, total = Post.search('测试', 1, 5)
...     print('>>>>>>query:{}'.format(query.all()))
...     print('>>>>>>total:{}'.format(total))
...
>>>>>>query:[<Post '测试1'>, <Post '测试2'>, <Post '测试3'>, <Post '测试4'>, <Post '测试5'>]
>>>>>>total:16
>>>
```

6.6. 编写搜索表单

修改`app/main/forms.py`脚本，增加搜索表单。

```
class SearchForm(FlaskForm):
    """搜索表单"""
    q = StringField(_l('搜索'), validators=[DataRequired()])

    def __init__(self, *args, **kwargs):
        # 由于选择GET的方式提交表单，所以通过request.args中获取参数
        if 'formdata' not in kwargs:
            kwargs['formdata'] = request.args
        # 将csrf_enabled设置为False，设置Flask-WTF忽略此表单的CSRF验证。
        if 'csrf_enabled' not in kwargs:
            kwargs['csrf_enabled'] = False
        super(SearchForm, self).__init__(*args, **kwargs)
```

由于需要在所有页面中都显示此表单，因此无论用户在查看哪个页面，我都需要创建一个`SearchForm`类的实例。所以直接修改`app/main/__init__.py`脚本，在请求处理前的处理器中初始化搜索表单。

```
from app.main.forms import SearchForm

def register_views(bp):
......
    @bp.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.datetime.utcnow()
            db.session.commit()
            g.search_form = SearchForm()
......
```

修改`app/templates/base.html`文件，在导航栏中增加搜索表单的渲染。

```
......
              <ul class="nav navbar-nav">
                    <li><a href="{{ url_for('main.index') }}">{{ _('首页') }}</a></li>
                    <li><a href="{{ url_for('main.explore') }}">{{ _('发现') }}</a></li>
                </ul>
                {% if g.search_form %}
                    <form class="navbar-form navbar-left" method="get"
                          action="{{ url_for('main.search') }}">
                        <div class="form-group">
                            {{ g.search_form.q(size=20, class='form-control',
placeholder=g.search_form.q.label.text) }}
                        </div>
                    </form>
                {% endif %}
......
```

6.7. 编写搜索视图类

修改`app/main/views.py`文件，增加搜索视图类。

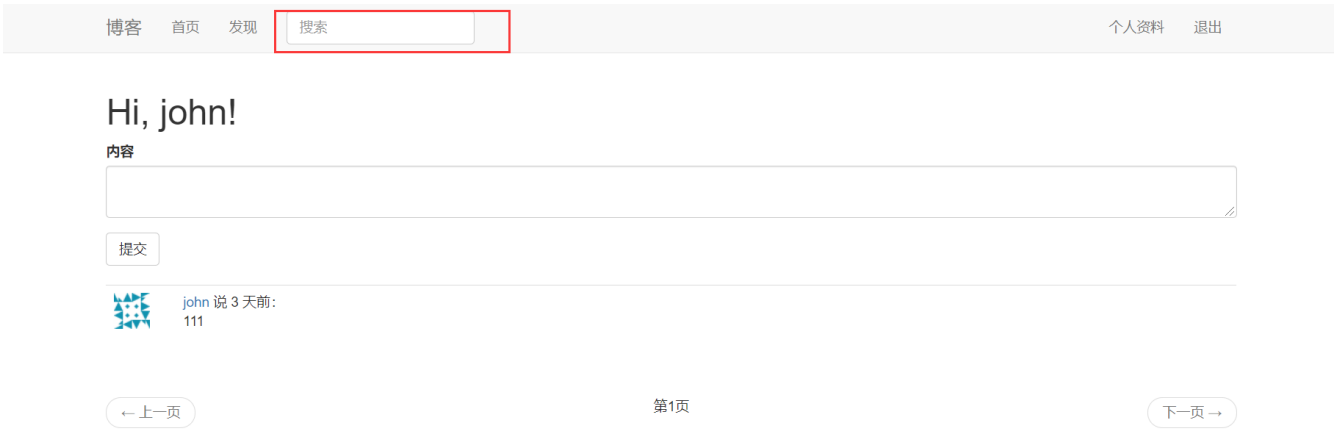
```
class SearchView(View):
    """搜索视图"""
    methods = ['GET']
    decorators = [login_required]

    def dispatch_request(self):
        # 表单校验失败，跳转至发现页
        if not g.search_form.validate():
            return redirect(url_for('main.explore'))
        # 分页查询
        page = request.args.get('page', 1, type=int)
        posts, total = Post.search(g.search_form.q.data, page,
current_app.config['POSTS_PER_PAGE'])
        next_url = url_for('main.search', q=g.search_form.q.data, page=page+1) \
            if total > page*current_app.config['POSTS_PER_PAGE'] else None
        prev_url = url_for('main.search', q=g.search_form.q.data, page=page-1) \
            if page > 1 else None
        return render_template('search.html', title=_('搜索'), posts=posts,
                               page=page, next_url=next_url, prev_url=prev_url)
```

新增`app/templates/search.html`文件，编写搜索结果展示模板。

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

{% block app_content %}
    <h1>{{ _('搜索结果') }}</h1>
    {% for post in posts %}
        {% include '_post.html' %}
    {% endfor %}
    <nav aria-label="...">
        <ul class="pager">
            <li class="previous{% if not prev_url %} disabled{% endif %}">
                <a href="{{ prev_url or '#' }}">
                    <span aria-hidden="true">&larr;</span>
                    {{ _('上一页') }}
                </a>
            </li>
            <li class="next{% if not next_url %} disabled{% endif %}">
                <a href="{{ next_url or '#' }}">
                    {{ _('下一页') }}
                    <span aria-hidden="true">&rarr;</span>
                </a>
            </li>
        </ul>
    </nav>
{% endblock %}
```

6.8. 启动服务测试

首页导航栏搜索框



点击搜索输入查询条件，进行查询