邮件发送

flask-mail

- 说明：专门用于邮件发送的扩展库，使用非常方便。
- 安装：`pip install flask-mail`
- 使用：

```
from flask_mail import Mail, Message
import os

# 邮件发送配置，一定要放在创建Mail对象之前
# 邮件服务器配置
app.config['MAIL_SERVER'] = "smtp.126.com"
# 用户帐号
app.config['MAIL_USERNAME'] = "landmark_csl@126.com"
# 授权码
app.config['MAIL_PASSWORD'] = "land123"

# 创建发送邮件的对象
mail = Mail(app)

@app.route('/send/')
def send():
    # 创建邮件消息对象
    msg = Message('账户激活',
                  recipients=['shuai_fmzj@163.com'],
                  sender=app.config['MAIL_USERNAME'])
    msg.html = '恭喜你，中奖了！！！'
    # 发送邮件
    mail.send(msg)
    return '邮件已发送'
```

- 封装函数发送邮件

```
def send_mail(subject, to, template, *args, **kwargs):
    if isinstance(to, (list,turple):
```

```
        recipients = to
    elif isinstance(to, str):
        recipients = to.split(',')
    else:
        raise Exception('邮件接收者参数类型有误')
    # 创建邮件消息对象
    msg = Message(subject,
                  recipients=recipients,
                  sender=app.config['MAIL_USERNAME'])
    # 将邮件模板渲染后作为邮件内容
    msg.html = render_template(template, *args, **kwargs)
    # 发送邮件
    mail.send(msg)
```

- 异步发送邮件

```
from flask import current_app

# 异步发送邮件任务
def async_send_mail(app, msg):
    # 邮件发送必须在程序上下文
    # 新的线程中没有上下文，因此需要手动创建
    with app.app_context():
        mail.send(msg)

# 封装函数发送邮件
def send_mail(subject, to, template, *args, **kwargs):
    if isinstance(to, list):
        recipients = to
    elif isinstance(to, str):
        recipients = to.split(',')
    else:
        raise Exception('邮件接收者参数类型有误')
    # 创建邮件消息对象
    msg = Message(subject,
                  recipients=recipients,
                  sender=app.config['MAIL_USERNAME'])
    # 将邮件模板渲染后作为邮件内容
    msg.html = render_template(template, *args, **kwargs)
    # 异步发送邮件
    # current_app是app的代理对象
    # 根据代理对象current_app找到原始的app
```

```
    app = current_app._get_current_object()
    # 创建线程
    thr = Thread(target=async_send_mail, args=(app, msg))
    # 启动线程
    thr.start()
    # 返回线程
    return thr
```

- QQ邮件发送额外配置：需要配置QQ邮箱开启smtp服务，然后设置授权码

```
# 邮箱端口
app.config['MAIL_PORT'] = 465
# 使用SSL(加密传输)
app.config['MAIL_USE_SSL'] = True
# 不是QQ邮箱的密码，而是授权码
app.config['MAIL_PASSWORD'] = '授权码'
```

分页显示

```
方法：paginate，分页查询
    参数：
        page：当前的页码
        per_page：每页的条数
        error_out：当查询出错时是否报错
    返回值：
        Pagination：分页对象，包含了所有的分页信息
Pagination：
    属性：
        page：当前页码
        per_page：每页的条数，默认为20条
        pages：总页数
        total：总条数
        prev_num：上一页的页码
        next_num：下一页的页码
        has_prev：是否有上一页
        has_next：是否有下一页
        items：当前页的数据
    方法：
        iter_pages：返回一个迭代器，在分页导航条上显示的页码列表，显示不完的时返回None
        prev：上一页的分页对象
        next：下一页的分页对象
```

- 封装分页显示的宏

```
{% macro show_pagination(pagination, endpoint) %}
    <nav aria-label="Page navigation">
        <ul class="pagination">
            {# 上一页 #}
            <li {% if not pagination.has_prev %}class="disabled"
{% endif %}>
                <a href="{% if pagination.has_prev %}{{
url_for(endpoint, page=pagination.prev_num, **kwargs) }}{% else
%}#{% endif %}" aria-label="Previous">
                    <span aria-hidden="true">&laquo;</span>
                </a>
            </li>

            {# 中间页码 #}
            {% for p in pagination.iter_pages() %}
                {% if p %}
                    <li {% if pagination.page == p
%}class="active"{% endif %}><a href="{{ url_for(endpoint, page=p,
**kwargs) }}">{{ p }}</a></li>
                {% else %}
                    <li><a href="#">&hellip;</a></li>
                {% endif %}
            {% endfor %}

            {# 下一页 #}
            <li {% if not pagination.has_next %}class="disabled"
{% endif %}>
                <a href="{% if pagination.has_next %}{{
url_for(endpoint, page=pagination.next_num, **kwargs) }}{% else
%}#{% endif %}" aria-label="Next">
                    <span aria-hidden="true">&raquo;</span>
                </a>
            </li>
        </ul>
    </nav>
{% endmacro %}
```

flask-login

- 说明：flask-login是一个专门用来管理用户登录退出的扩展库
- 安装：`pip install flask-login`
- 使用：

```
# 第一步：添加扩展
from flask_login import LoginManager

login_manager = LoginManager()

def config_extensions(app):
    ...
    login_manager.init_app(app)
    # 设置登录端点
    login_manager.login_view = 'user.login'
    # 设置登录信息
    login_manager.login_message = '请先登录，然后才能访问'


# 第二步：继承自UserMixin类(也可以自己实现相关方法)
from flask_login import UserMixin

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

# 第三步：实现回调
@login_manager.user_loader
def load_user(uid):
    return User.query.get(uid)
```

- 总结

```
状态切换：
    login_user      # 可以提供记住我的功能
    logout_user     # 退出登录
状态查询：
    is_authenticated    登录状态
    is_anonymous        匿名状态
路由保护：
    login_required      # 保护需要登录才能访问的路由
当前用户：
    current_user        # 哪里都可以使用，在模板中不需要分配
```

项目结构

目录结构

```
blog/                   # 项目根目录
    manage.py               # 启动控制代码
    requirements.txt        # 依赖包类表文件
    migrations/             # 数据库迁移目录
    tests/                  # 测试模块目录
    app/                    # 整个程序目录
        templates/              # 模板文件目录
            common/                 # 通用模板
            email/                  # 邮件模板
            。。。
        static/                 # 静态文件目录
            img/
            css/
            js/
            favicon.ico
        views/                  # 蓝本文件目录
        models.py               # 数据模型文件
        forms.py                # 表单类文件
        settings.py             # 配置文件
        extensions.py           # 扩展文件(存放所有扩展)
        email.py                # 邮件发送功能函数
        __init__.py             # 包文件
```

项目准备

- 根据目录结构，创建相关目录及文件
- 书写配置文件(就是书写各种环境的配置类)
- 使用工厂方法创建应用实例，并初始化配置
- 添加各种扩展(顺便粘贴邮件发送函数)
- 配置蓝本(添加各种蓝本文件，并注册)