系统设计文档

1.系统名称：基于Flask 的博客系统

2.系统简介：博客是典型的CMS（Content Management System，内容管理系统），通常由两部分组成：一部分是博客前台，用来展示开放给所有用户的博客内容；另一部分是博客后台，这部分内容仅开放给博客作者用户，用来对博客资源进行添加、修改和删除等操作。

3.系统项目结构（主程序包的文件结构组成）：

MyBlog/ 主程序包

blueprints/ 蓝图包

- \_\_init\_\_.py

- blog.py内容模块

- auth.py账户管理模块

- admin.py管理模块

templates/ jinja2模板

- admin/

- auth/

- blog/

- base.html

- macros.html

static/ 静态资源文件

forms.py 表单类模块

models.py 数据库模型类模块

emails.py 电子邮件模块

utils.py 辅助函数模块

fakes.py 虚拟数据

extensions.py 扩

4.系统功能规划：

MyBlog

博客前台

用户认证

博客后台

用户注册

用户登陆

账号管理

博客展示

发表评论

回复评论

博客管理

评论管理

5.系统设计

5.1数据库设计

本系统数据库采用Flask扩展Flask-SQLAlchemy，此扩展集成了SQLAlchemy，它简化了连接数据库服务器、管理数据库操作会话等各类工作，让Flask中的数据处理体验变得更加轻松。

本系统一共需要使用四张表，分别存储博客用户（Admin）、分类（Category）、文章（Post）和评论（Comment）。

数据库模型如下所示：

1. MyBlog/models.py :用户模型

class Admin(db.Model):

id = db.Column(db.Integer, primary\_key=True) #主键

username = db.Column(db.String(20)) #用户名

email = db.Column(db.String(254), unique=True, index=True) #邮箱

password\_hash = db.Column(db.String(128)) #密码hash值

name = db.Column(db.String(30)) #真实姓名

confirmed = db.Column(db.Boolean, default=False) #邮箱确认

posts = db.relationship('Post', back\_populates='author', cascade='all') #发布的博文

comments = db.relationship('Comment', back\_populates='author', cascade='all') #发布的评论

def set\_password(self, password): #设置密码

self.password\_hash = generate\_password\_hash(password)

def validate\_password(self, password): #验证密码

return check\_password\_hash(self.password\_hash, password)

1. MyBlog/models.py :分类模型

class Category(db.Model):

id = db.Column(db.Integer, primary\_key=True) #主键

name = db.Column(db.String(30), unique=True) #类名

posts = db.relationship('Post', back\_populates='category') #分类下的文章

1. bluelog/models.py :文章模型

class Post(db.Model):

id = db.Column(db.Integer, primary\_key=True) #主键

title = db.Column(db.String(60)) #标题

body = db.Column(db.Text) #内容

timestamp = db.Column(db.DateTime, default=datetime.utcnow, index=True) #发布时间

can\_comment = db.Column(db.Boolean, default=True) #可以评论

author\_id = db.Column(db.Integer, db.ForeignKey('admin.id')) #作者id

category\_id = db.Column(db.Integer, db.ForeignKey('category.id')) #类别id

category = db.relationship('Category', back\_populates='posts') #类别

comments = db.relationship('Comment', back\_populates='post', cascade='all') #评论

author = db.relationship('Admin', back\_populates='posts') #作者

1. bluelog/models.py :评论模型

class Comment(db.Model):

id = db.Column(db.Integer, primary\_key=True) #主键

author\_name = db.Column(db.String(30)) #作者名

body = db.Column(db.Text) #内容

reviewed = db.Column(db.Boolean, default=False) #已审查

timestamp = db.Column(db.DateTime, default=datetime.utcnow, index=True) #发布时间

author\_id = db.Column(db.Integer, db.ForeignKey('admin.id')) #作者id

replied\_id = db.Column(db.Integer, db.ForeignKey('comment.id')) #类别id

post\_id = db.Column(db.Integer, db.ForeignKey('post.id')) #文章id

post\_id\_r = db.Column(db.Integer) #文章id

author = db.relationship('Admin', back\_populates='comments') #作者

post = db.relationship('Post', back\_populates='comments') #文章

replies = db.relationship('Comment', back\_populates='replied', cascade='all, delete-orphan') #回复的评论

replied = db.relationship('Comment', back\_populates='replies', remote\_side=[id]) #被回复的评论

5.2 主要功能设计

5.2.1用户认证模块

本模块的功能包含用户注册、登陆、账号设置。

用户注册需要填写表单，如下所示为用于注册的RegisterForm表单类。

MyBlog/forms.py：注册表单

class RegisterForm(FlaskForm):

name = StringField('真实姓名', validators=[DataRequired(), Length(1, 30)])

email = StringField('电子邮件', validators=[DataRequired(), Length(1, 254), Email()])

username = StringField('用户名', validators=[DataRequired(), Length(1, 20),Regexp('^[a-zA-Z0-9]\*$',message='用户名仅能包含 a-z, A-Z and 0-9.')])

password = PasswordField('密码', validators=[

DataRequired(), Length(1, 128), EqualTo('password2')])

password2 = PasswordField('确认密码', validators=[DataRequired()])

submit = SubmitField('注册')

def validate\_email(self, field):

if Admin.query.filter\_by(email=field.data.lower()).first():

raise ValidationError('该电子邮件已注册！')

def validate\_username(self, field):

if Admin.query.filter\_by(username=field.data).first():

raise ValidationError('该用户名已被注册！')

用户的账户名（username）是用户在系统中的重要标识，用户的个人重要信息将基于这个值构建，因此这个值不能重复，而且应该由英文字母、数字和少数符号组成。为了达到这个目的，在username字段中使用了Regexp验证器，这会借助正则表达式来验证输入值。Regexp类的构造方法接收正则表达式作为第一个参数（regex），可选的flags参数用于指定正则表达式的旗标，默认为0。

为了防止用户输错密码，表单中有两个密码字段：password、password2。第一个密码字段通过EqualTo验证器来确保其输入值与password2字段的输入值相同，EqualTo类的构造方法的第一个参数为用于比较的目标字段名。

要确保User模型中的username和email字段的不会与已有的数据重复，所以系统添加了两个自定义验证方法validate\_username（）和validate\_email（），分别用于验证表单的username和email字段。这两个验证函数会验证数据库中是否存在包含相同字段值的记录。如果包含，就抛出ValidationError异常，错误信息作为参数传入。

如下所示是用于渲染表单和处理表单验证的register视图。

MyBlog/blueprints/auth.py：处理注册请求

@auth\_bp.route('/register', methods=['GET', 'POST'])

def register():

if current\_user.is\_authenticated:

return redirect(url\_for('blog.index'))

form = RegisterForm()

if form.validate\_on\_submit():

name = form.name.data

email = form.email.data.lower()

username = form.username.data

password = form.password.data

user = Admin(name=name, email=email, username=username)

user.set\_password(password)

db.session.add(user)

db.session.commit()

token = generate\_token(user=user, operation='confirm')

send\_confirm\_email(user=user, token=token)

login\_user(user)

flash('账户确认邮件已发送, 请注意检查邮箱！', 'info')

return redirect(url\_for('blog.index'))

return render\_template('auth/register.html', form=form)

用户的邮箱地址用于登录账户、和重置密码等。为了确保邮箱地址的真实性和有效性，系统需要对用户的邮箱地址进行验证，以便确认注册。

使用扩展Flask-Login实现相对成熟完整的登陆认证功能。扩展Flask-Login为Flask提供了用户会话管理功能，使用它可以轻松的处理用户登录、登出等操作。

使用Flask-Login登入/登出某个用户非常简单，只需要在视图函数中调用Flask-Login提供的login\_user（）或logout\_user（）函数，并传入要登入/登出的用户类对象。在这两个函数背后，Flask-Login使用Flask的session对象将用户的id值存储到用户浏览器的cookie中（名为user\_id），这时表示用户被登入。相对来说，登出则意味着在用户浏览器的cookie中删除这个值。默认情况下，关闭浏览器时，通过Flask的session对象存储在客户端的session cookie会被删除，所以用户会登出。另外，Flask-Login还支持记住登录状态，通过在login\_user（）中将remember参数设为True即可实现。这时Flask-Login会在用户浏览器中创建一个名为remember\_token的cookie，当通过session设置的user\_id cookie因为用户关闭浏览器而失效时，它会重新恢复user\_id cookie的值。

用于登录用户的login视图如下所示。

MyBlog/blueprints/auth.py：处理登陆请求

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

def login():

if current\_user.is\_authenticated:

return redirect(url\_for('blog.index'))

form = LoginForm()

if form.validate\_on\_submit():

password = form.password.data

remember = form.remember.data

admin = Admin.query.filter\_by(email=form.email.data.lower()).first()

if admin:

if admin.validate\_password(password):

login\_user(admin, remember)

flash('欢迎回来.', 'info')

return redirect\_back()

flash('密码不正确.', 'warning')

else:

flash('无此用户.', 'warning')

return render\_template('auth/login.html', form=form)

用户忘记密码是很常见的事情，当需要重置密码时，用户需要输入注册的邮箱地址，系统通过表单数据中的邮箱地址查询对应的用户。如果用户存在就发送重置密码邮件，否则发送错误提示消息并重定向回登录页面。重置密码视图如下所示：

MyBlog/blueprints/auth.py：处理忘记密码请求

@auth\_bp.route('/forget-password', methods=['GET', 'POST'])

def forget\_password():

if current\_user.is\_authenticated:

return redirect(url\_for('blog.index'))

form = ForgetPasswordForm()

if form.validate\_on\_submit():

user = Admin.query.filter\_by(email=form.email.data.lower()).first()

if user:

token = generate\_token(user=user, operation=Operations.RESET\_PASSWORD)

send\_reset\_password\_email(user=user, token=token)

flash('密码重置邮件已发送，请在邮箱查看.', 'info')

return redirect(url\_for('.login'))

flash('邮箱不存在.', 'warning')

return redirect(url\_for('.forget\_password'))

return render\_template('auth/reset\_password.html', form=form)

5.2.2博客相关模块

本模块功能包含博客内容展示，博客评论管理，博客增、删、改管理。

博客最重要的功能就是撰写发布文章，new\_post视图负责渲染创建文章的模板，并处理页面中表单提交的POST请求，new\_post视图如下所示。

MyBlog/blueprints/admin.py：处理新建博文请求

@admin\_bp.route('/post/new', methods=['GET', 'POST'])

@login\_required

def new\_post():

form = PostForm()

if form.validate\_on\_submit():

title = form.title.data

body = form.body.data

category = Category.query.get(form.category.data)

post = Post(title=title, body=body, category=category, author=current\_user)

db.session.add(post)

db.session.commit()

flash('博文已创建.', 'success')

return redirect(url\_for('blog.show\_post', post\_id=post.id))

return render\_template('admin/new\_post.html', form=form)

编辑文章的具体实现和撰写新文章类似，这两个功能使用同一个表单类PostForm，而且视图函数和模板文件都基本相同，主要的区别是需要在用户访问编辑页面时把文章数据预先放置到表单中，edit\_post视图如下所示：

MyBlog/blueprints/admin.py：处理编辑博文请求

@admin\_bp.route('/post/<int:post\_id>/edit', methods=['GET', 'POST'])

@login\_required

def edit\_post(post\_id):

form = PostForm()

post = Post.query.get\_or\_404(post\_id)

if form.validate\_on\_submit():

post.title = form.title.data

post.body = form.body.data

post.category = Category.query.get(form.category.data)

db.session.commit()

flash('博文已更新.', 'success')

return redirect(url\_for('blog.show\_post', post\_id=post.id))

form.title.data = post.title

form.body.data = post.body

form.category.data = post.category\_id

return render\_template('admin/edit\_post.html', form=form)

通过delete\_post视图可以删除文章，首先从数据库中获取指定id的文章记录，然后使用db.session.delete（）方法删除记录并提交数据库。delete\_post视图如下所示：

MyBlog/blueprints/admin.py：处理删除博文请求

@admin\_bp.route('/post/<int:post\_id>/delete', methods=['POST'])

@login\_required

def delete\_post(post\_id):

post = Post.query.get\_or\_404(post\_id)

db.session.delete(post)

db.session.commit()

flash('博文已删除.', 'success')

return redirect\_back()

博客前台需要开放给所有用户，这里包括显示文章内容信息和评论。

文章页面通过show\_post视图渲染，路由的URL规则中包含一个post\_id变量，我们将post\_id作为主键值来查询对应的文章对象，并传入模板，show\_post视图如下所示：

MyBlog/blueprints/blog.py：处理删除博文请求

@blog\_bp.route('/post/<int:post\_id>', methods=['GET', 'POST'])

@login\_required

def show\_post(post\_id):

post = Post.query.get\_or\_404(post\_id)

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

per\_page = current\_app.config['BLUELOG\_COMMENT\_PER\_PAGE']

pagination = Comment.query.with\_parent(post).filter\_by(reviewed=True).order\_by(Comment.timestamp.asc()).paginate(page, per\_page)

comments = pagination.items

if current\_user.is\_authenticated:

form = AdminCommentForm()

reviewed = False

if current\_user == post.author:

reviewed = True

if form.validate\_on\_submit():

author = current\_user.\_get\_current\_object()

author\_name = author.username

body = form.body.data

if post.author == current\_user:

comment = Comment(

author=author, author\_name=author\_name, body=body,

post=post, reviewed=reviewed)

else:

comment = Comment(

author=author, author\_name=author\_name, body=body, 接下

post\_id\_r=post.id, reviewed=reviewed)

replied\_id = request.args.get('reply')

if replied\_id:

replied\_comment = Comment.query.get\_or\_404(replied\_id)

comment.replied = replied\_comment

db.session.add(comment)

db.session.commit()

if post.author==current\_user:

flash('评论发布成功.', 'success')

else:

flash('你的评论将在作者审阅后发布.', 'info')

return redirect(url\_for('.show\_post', post\_id=post\_id))

return render\_template('blog/post.html', post=post, pagination=pagination, form=form, comments=comments)

接上

post\_id\_r=post.id, reviewed=reviewed)

replied\_id = request.args.get('reply')

if replied\_id:

replied\_comment = Comment.query.get\_or\_404(replied\_id)

comment.replied = replied\_comment

db.session.add(comment)

db.session.commit()

if post.author==current\_user:

flash('评论发布成功.', 'success')

else:

flash('你的评论将在作者审阅后发布.', 'info')

return redirect(url\_for('.show\_post', post\_id=post\_id))

return render\_template('blog/post.html', post=post, pagination=pagination, form=form, comments=comments) post\_id\_r=post.id, reviewed=reviewed)

replied\_id = request.args.get('reply')

if replied\_id:

replied\_comment = Comment.query.get\_or\_404(replied\_id)

comment.replied = replied\_comment

db.session.add(comment)

db.session.commit()

if post.author==current\_user:

flash('评论发布成功.', 'success')

else:

flash('你的评论将在作者审阅后发布.', 'info')

return redirect(url\_for('.show\_post', post\_id=post\_id))

return render\_template('blog/post.html', post=post, pagination=pagination, form=form, comments=comments)

评论列表在显示文章的页面显示，我们首先在显示文章的show\_post视图中获取对应的文章，然后使用Comment.query.with\_parent（post）方法获取文章所属的评论，并对其进行排序和分页处理（per\_page的值通过配置变量BLUELOG\_COMMENT\_PER\_PAGE获取），获取对应页数的评论记录，最后传入模板中。评论列表里仅需要列出通过审核的评论，所以在视图函数里的数据库查询使用filter\_by（reviewed=True）来筛选出通过审核的评论记录。因为评论表单要显示在文章页面的评论列表下方，所以用户评论数据的验证和保存在show\_post视图中处理。当评论用户为文章作者时，发布的评论不需要审核，而其他用户的评论需要经文章作者审核后才能发布。

作为用户，可以在博客系统后台管理自己发布的博客文章以及别人对自己发布的文章的评论，如审核通过评论，删除评论。