

总体介绍

总体设计方案

本博客网站后端部分使用Python流行 Web框架Flask进行构建而成，前端由Bootstrap构建而成，通过后端传入数据给前端Bootstrap模板进行渲染，得到具有一定美观的博客网站，其中数据存储使用到SQLite及Redis，最后通过Gunicorn+Nignx部署该网站。

关键词： 博客网站； Flask； Bootstrap； 前后端交互

项目已完成需求

- ☑ 游客可以浏览会员公开发布的文章
- ☑ 游客可以注册为会员用户；
- ☑ 会员用户可以管理其个人基本资料；

会员用户在登录状态下，

- ☑ 可以创建个人博客空间、发布文章（可设定公开或不公开， 开放或禁止文字评论）；
- ☑ 管理文章（创建和管理文章分类、 标注索引关键字[?]、编辑、修改或 删除文章等）
- ☑ 文章评论（可对文章跟评、删除自己文章的评论等）；
- ☑ 系统管理员可以进行用户管理（浏览、更新用户资料）， 网站文章管理（删除博客文章和评论等）；
- ☑ 系统管理员可以查看在线用户

设计模式

MVC

MVC在本项目中运用

该博客网站使用MVC模式进行设计。

MVC 模式代表 Model-View-Controller（模型-视图-控制器） 模式。

- **Model（模型）** - 模型代表一个存取数据的对象(Model类)。它也可以带有逻辑，在数据变化时更新控制器。

这部分详情请看专题 [设计Model类](#)

- **View（视图）** - 视图代表模型包含的数据的可视化。

这部分详情请看专题 [前端部分](#)

- **Controller（控制器）** - 控制器作用于模型和视图上。它控制数据流向模型对象，并在数据变化时更新视图。它使视图与模型分离开。

这部分体现在后端设计的接口中，比如下面这个（搜索）

```
@main.route('/search_result', methods=['GET'])
def get_search_result():
    page = request.args.get('page', 1, type=int)
    keyword = request.args.get("keyword")
    # print(keyword)
    # https://www.jianshu.com/p/ca9bc9f8adab
    pagination = Post.query.filter_by(author_id =
current_user._get_current_object().id).filter(or_(Post.title.contains(keyword),
Post.body.contains(keyword), \
Post.summury.contains(keyword), \
)).order_by(Post.timestamp.desc()).paginate(
page, per_page=current_app.config['FLASKY_POSTS_PER_PAGE'],
error_out=False)
```

为什么使用MVC

个人感觉MVC设计模式最大的优势就是方法模块管理，各司其职，哪里出现问题就修改哪里，也就避免项目过于糅杂，各端代码耦合在一起，不方便修改，所以选择MVC设计模式。

ORM

ORM介绍

ORM 通过实例对象的语法，完成关系型数据库的操作的技术，是"对象-关系映射"（Object/Relational Mapping）的缩写。

ORM在本项目中运用

拿恢复评论代码来说

```
def moderate_enable(id):
    comment = Comment.query.get_or_404(id) # 找到具体实例对象
    comment.disabled = False # 直接操作
    db.session.add(comment) # 保存操作
```

从上面代码可以看到，整个过程基于面向对象，直接调用属性或者库的方法就可以操作数据库，而不用写冗余的SQL语句，这真的是既美观又大大提高了开发效率。

为什么使用ORM

原因在上面已经讲到，主要是为了美观及方便性，性能当然比不过直接调用SQL语句。但是对于这次作业，性能差距可以忽略，因为这是小网站，用户不多，数据库读出读入速度也不会至于差距过大。

开发环境

- 本地环境：win10专业工作站版，Version1909
- 服务器环境：阿里云学生服务器 Ubuntu16.04.6 LTS
- Python: 3.6.8
- Flask: 1.0.2
- SQLite: 3.2.80
- Redis: 3.0.6
- 硬件配置：学生服务器都能流畅运行，那近几年市面上的配置是绰绰有余

- Python需要安装的库目录放在邮件附录中 requirements.txt

```
[2020-06-09 07:33:00 +0800] [10491] [INFO] Listening at: http://0.0.0.0:8666 (10491)
[2020-06-09 07:33:00 +0800] [10491] [INFO] Using worker: sync
[2020-06-09 07:33:01 +0800] [10494] [INFO] Booting worker with pid: 10494
[2020-06-09 07:33:01 +0800] [10496] [INFO] Booting worker with pid: 10496
[2020-06-09 07:33:01 +0800] [10497] [INFO] Booting worker with pid: 10497
[2020-06-09 07:33:01 +0800] [10498] [INFO] Booting worker with pid: 10498
[2020-06-09 07:33:01 +0800] [10499] [INFO] Booting worker with pid: 10499
[2020-06-09 07:33:01 +0800] [10501] [INFO] Booting worker with pid: 10501
[2020-06-09 07:33:01 +0800] [10502] [INFO] Booting worker with pid: 10502
[2020-06-09 07:33:01 +0800] [10504] [INFO] Booting worker with pid: 10504
[2020-06-09 07:33:01 +0800] [10505] [INFO] Booting worker with pid: 10505
[2020-06-09 07:33:01 +0800] [10506] [INFO] Booting worker with pid: 10506
在线人数为:0
他们id分别为:
[2020-06-09 07:33:35 +0800] [10491] [CRITICAL] WORKER TIMEOUT (pid:10496)
[2020-06-09 07:33:35 +0800] [10496] [INFO] Worker exiting (pid: 10496)
[2020-06-09 07:33:35 +0800] [10515] [INFO] Booting worker with pid: 10515
在线人数为:0
他们id分别为:
在线人数为:0
他们id分别为:
在线人数为:1
他们id分别为:
```

开发工具

- XShell 6: 用于连接阿里云Ubuntu服务器;
- Filezilla: 用于进行本地计算机与阿里云Ubuntu服务器进行文件传输;
- Sublime Text3: 代码编辑器, 用于本地测试



项目部署站点

目前该博客网站已经部署成功, 访问<http://120.76.128.109:8666/>即可进行体验。

视频演示介绍

请观看邮件附件中的 JSP 综合实验项目运行视频 20182131141 唐高智.mp4

后端部分

Python Web框架Flask

Flask介绍



Flask是一个基于Python的Web开发微框架，微架构通常是很小的不依赖于外部库的框架。这既有优点也有缺点，优点是框架很轻量，更新时依赖少，并且专注安全方面的 bug，缺点是，你不得不自己做更多的工作，或通过添加插件增加自己的依赖列表。

Flask优点

- 入门简单，即便没有多少web开发经验，也能很快做出网站
- 非常适用于小型网站
- 非常适用于开发web服务的API

关于框架选择的思考

对于初学者来说，找到一个好的框架来学习或者项目开发都是非常有必要的。

面对一个项目需求，首先不要着急去开发，**先搞明白技术难点可能带来的额外的时间消耗**，这有利于我们控制开发成本。

Flask框架是一个微框架，没有太多的条框约束，比较自由灵活，性能肯定比不上Spring Boot主流框架，但是对于现阶段的我，我需要手动实现每一个功能，比如登陆认证权限、api接口等。而有些Web框架，这些基础功能可以快速生成，我们只需要需要相应位置即可，但这并不利于现阶段的学习，所以我选择了Flask。

• 为什么不选择JSP?

个人觉得JSP后端部分手感我挺喜欢的（通过作业上机实验得出的感觉），但是我比较不喜欢前后端耦合在一起，感觉比较不灵活，其他还好。

• Bootstrap与JSP有共同点吗?

Bootstrap跟JSP的前端部分很像，Bootstrap中显示变量是 `{{ user.username }}`，也是后端传入参数给前端进行显示，只不过这次项目是前后端分离而已，但是思想类似。

设计Model类

SQLite数据库的使用

SQLite数据库介绍



SQLite与其他SQL数据库不同，SQLite没有单独的服务器进程。它直接读取和写入普通磁盘文件。具有多个表，索引，触发器和视图的完整SQL数据库包含在单个磁盘文件中。

为什么使用SQLite数据库

SQLite是一款轻型的数据库，在该博客网站项目中使用SQLite主要考虑到方便性。因为博客网站属于多读少写场合，文件数据库刚好这方面不输mysql，而且也可以绕过连接限制。还有以下优点：

- SQLite安装方便，不需要设置数据库账号密码；
- 网站搬家方便，搬家时只需要把整个web文件搬过去就行（包含sqlite后缀文件），不需要搬数据库。

当然性能和功能完整性比不过MySQL，但是该项目代码照样可以使用MySQL，只需要更换SQL_URI即可。

设计过程

所需Model类

结合博客网站特点，初步设计了一下Model类

- Permission（权限标识符类，该类下的不同属性为权限种类，并且有不同的标识符）
- Role（用户角色类，记录用户id、名字、角色）
- Follow（关注类，属性包括关注者用户id、被关注者用户id）
- User（用户类，属性包括用户所必需的，比如关注者、邮箱等）
- AnonymousUser（匿名用户类，供给未登录用户使用）
- Post（帖子类，属性包括作者id、标题、主题内容、评论等）
- Comment（评论类，属性包括评论者id、评论内容、评论时间等）
- Category（文章tag类，属性包括标签id、创建该标签用户id、标签名字等）

Model类后端定义

首先先导入

```
from flask_sqlalchemy import SQLAlchemy
```

拿Category类举例，代码如下

```

class Category(db.Model):
    __tablename__ = 'categories'
    id = db.Column(db.Integer, primary_key=True)
    author_id = db.Column(db.Integer, db.ForeignKey('users.id'))
    name = db.Column(db.String(64), unique=True)
    posts = db.relationship('Post', backref='category', lazy='dynamic') #设置关联

    def to_json(self):
        json_category = {
            'url': url_for('api.get_category', id=self.id, _external=True),
            'body': self.body,
            'author': url_for('api.get_user', id=self.author_id,
                              _external=True),
            'posts': url_for('api.get_category_posts', id=self.id,
                              _external=True),
            'post_count': self.posts.count()
        }
        return json_category

    @staticmethod
    def insert_categories():
        pass

    def __repr__(self):
        return '<Category %r>' % self.name

```

Model类SQL语言实现

由于篇幅原因，只展示部分代码

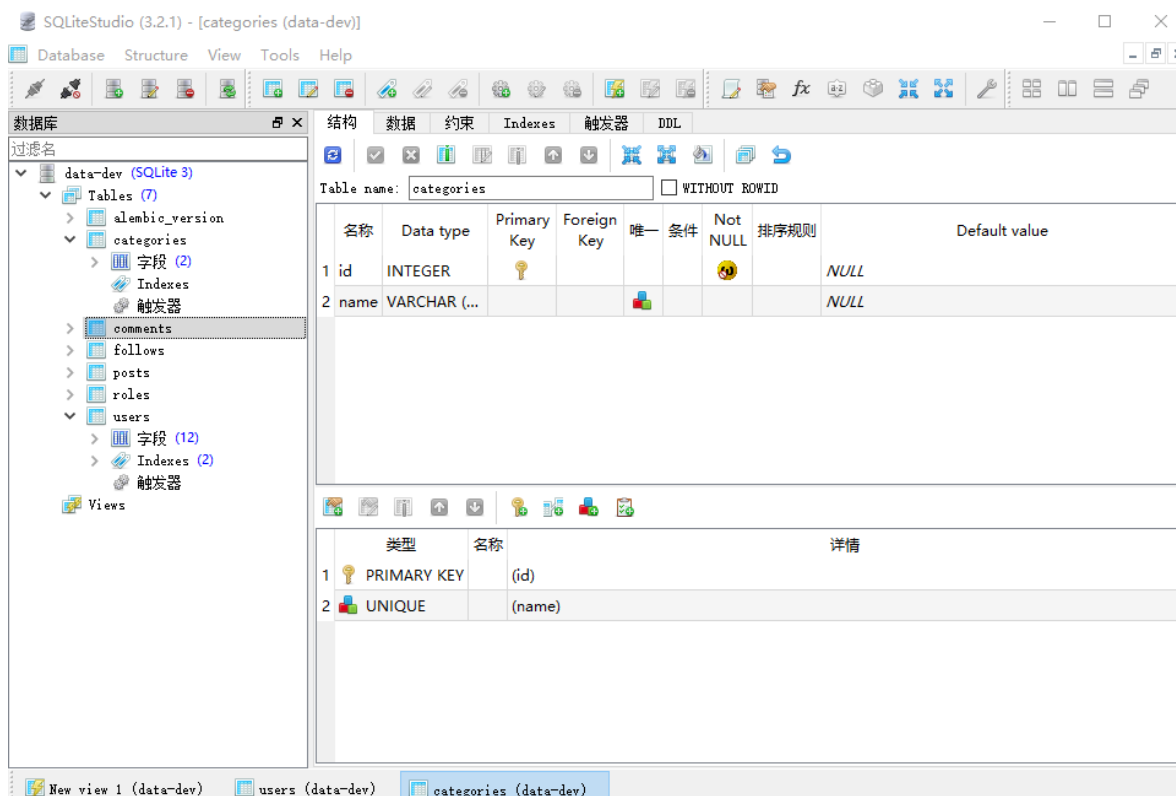
```

CREATE TABLE comments (
    id INTEGER NOT NULL,
    body TEXT,
    body_html TEXT,
    timestamp DATETIME,
    disabled BOOLEAN,
    author_id INTEGER,
    post_id INTEGER,
    PRIMARY KEY (id),
    FOREIGN KEY(author_id) REFERENCES users (id),
    FOREIGN KEY(post_id) REFERENCES posts (id),
    CHECK (disabled IN (0, 1))
);

```

Model类可视化

- 使用SQLiteStudio可视化数据库



- 使用SQLite原生语言

```
(flask) pmZ:~/flask, blog# sqlite3 data-dev.sqlite
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> .tables
alembic_version  comments  posts      users
categories       follows   roles
sqlite> .schema categories
CREATE TABLE categories (
  id INTEGER NOT NULL,
  author_id INTEGER,
  name VARCHAR(64),
  PRIMARY KEY (id),
  FOREIGN KEY(author_id) REFERENCES users (id)
);
sqlite> .quit
```

在线用户

Redis数据库的使用

Redis数据库介绍



redis

- 是一个完全开源免费的**key-value**内存数据库；
- 通常被认为是一个数据结构服务器，主要是因为其有着丰富的数据结构 strings、map、list、sets、sorted sets

为什么使用Redis数据库

Redis优点涉及多种方面，既有安全性的、也有性能方面的，这次博客网站项目需要统计在线人员，多种页面当中需要用到数据的写入读出，也就是需要**频繁地写入读出**，如果只单纯地对普通数据库(如MySQL)进行操作，这将是一笔不可小觑的性能开销。

Redis性能极高：能支持超过 100K+ 每秒的读写频率，这也是我使用Redis数据库缓存用户在线信息的重要原因。

设计过程

定义在线指标

5 分钟内**请求过连接**的Client端认为处于**在线状态**。

使用Redis存储数据

(1) Flask 中使用 redis 有方便的第三方库 flask_redis。可以直接通过 pip 命令进行安装。

```
pip install flask-redis
```

(2) 确保 redis 已经在本机正常启动，并在配置文件中配置 URL：

```
REDIS_URL="redis://localhost:6379"
```

```
from flask import Flask
from flask_redis import FlaskRedis

app = Flask(__name__)
app.config['REDIS_URL'] = 'redis://:@127.0.0.1:6379/0'
redis_client = FlaskRedis(app)
```

记录数据

在每次请求中记录下Client端的**用户ID** 及**当前的时间**。

```
def mark_online(user):
    user_id = str(user.id).encode('utf-8')
    now = int(time.time())
    expires = now + (5 * 60) + 10
    all_users_key = "online-users/%d" % (now // 60)
    user_key = "user-activity/%s" % user_id
    p = redis_client.pipeline()
    p.sadd(all_users_key, user_id)
    p.set(user_key, now)
    p.expireat(all_users_key, expires)
    p.expireat(user_key, expires)
    p.execute()
```

读取在线数据

读取用户存储在Redis数据库中

```
@main.route('/online', methods=['GET'])
def get_online():
    current = int(time.time()) // 60
    minutes = range(5)
```



```

users = redis_client.sunion(
    ["online-users/%d" % (current - x) for x in minutes]
)

#print("在线人数为:"+str(len([int(u.decode('utf-8')) for u in users])))
#print("他们id分别为:")

USER = []

for u in users:#获取每个在线用户
    USERS.append(User.query.filter_by(id=int(u.decode('utf-8'))).first())

#给Bootstrap传入每个在线用户的值
online_users = [{'username':user.username
, "gravatar":user.gravatar(size=32), "last_seen":user.last_seen}
    for user in USERS]

return render_template('online_user.html', title="在线用
户",user=current_user._get_current_object(),users=online_users)

```

效果展示

线下测试

```

14 from flask import Flask
15 from flask_redis import FlaskRedis
16 import time
17 import random
18
19 app = Flask(__name__)
20 app.config['REDIS_URL'] = 'redis://:@127.0.0.1:6379/0'
21 redis_client = FlaskRedis(app)
22
52
86
79
84
33
27
127.0.0.1 - - [06/Jun/2020 23:00:58] "GET / HTTP/1.1" 200 -
在线人数为:1
他们id分别为:
18
127.0.0.1 - - [06/Jun/2020 23:05:38] "GET / HTTP/1.1" 200 -


```

线上测试

[BLOG](#)
[主页](#)
[写文章](#)
[新分类](#)

[搜索](#)
[在线用户](#)
[管理评论](#)
[账户](#)

在线用户

用户昵称	活跃时间
 tgz2022	a few seconds ago

© Copyright By [linggaozi](#) 2020

文章管理

设计Post类

主要分以下几步进行设计：

- (1) Post类需要哪些属性？
- (2) Post类主键是哪个？需要哪些约束？
- (3) Post类需要跟哪个Model类相关联？相关联的话又跟这个Model类的哪个属性建立建立
- (4) Post类需要哪些常用方法？
- (5) 修改文章时Post类对应的方法如何设计？

- Post类属性定义

```
class Post(db.Model):
    __tablename__ = 'posts'
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    author_id = db.Column(db.Integer, db.ForeignKey('users.id'))
    title = db.Column(db.String(64))
    body = db.Column(db.Text)
    body_html = db.Column(db.Text)
    summary = db.Column(db.Text)
    summary_html = db.Column(db.Text)
    category_id = db.Column(db.Integer, db.ForeignKey('categories.id'))
    comments = db.relationship('Comment', backref='post', lazy='dynamic') #关联
    # Comment类

    is_public_disabled = db.Column(db.Boolean) #是否公开
    is_commented_disabled = db.Column(db.Boolean) #是否允许评论
```

- Post类请求json接口方法

```
def to_json(self):
    json_post = {
        'url': url_for('api.get_post', id=self.id, _external=True),
        'body': self.body,
        'body_html': self.body_html,
        'timestamp': self.timestamp,
        'author': url_for('api.get_user', id=self.author_id,
                           _external=True),
        'comments': url_for('api.get_post_comments', id=self.id,
                             _external=True),
        'comment_count': self.comments.count()
    }
    return json_post
```

功能实现及展示

新建文章

- 表格Form设计

```
class PostForm(FlaskForm):
    title = StringField(u'标题', validators=[Required()])
    body = PageDownField(u'内容', validators=[Required()])
    summury = PageDownField(u'摘要', validators=[Required()])
    category = SelectField(u'分类', coerce=int)
    is_public = SelectField(u'是否公开', coerce=int)
    is_comment = SelectField(u'是否允许评论', coerce=int)
    submit = SubmitField(u'提交')

    def __init__(self, *args, **kwargs):
        super(PostForm, self).__init__(*args, **kwargs)
        self.category.choices = [(category.id, category.name) for category
                                in Category.query.filter_by(author_id =
current_user._get_current_object().id).order_by(Category.name).all()]
        self.is_public.choices = [(1, "是"), (0, "否")]
        self.is_comment.choices = [(1, "是"), (0, "否")]
```

- 新建文章表格效果展示

The screenshot shows a web application interface for creating a new article. The header includes a navigation bar with links for 'BLOG', '主页' (Home), '写文章' (Write Article), '新分类' (New Category), a search bar, and user links for '在线用户' (Online Users), '管理评论' (Manage Comments), and '账户' (Account). The main form area contains several input fields: '标题' (Title) with a text input, '内容' (Content) with a large text area, '摘要' (Summary) with a text area, '分类' (Category) with a dropdown menu showing '测试2', '是否公开' (Is Public) with a dropdown menu showing '是' (Yes), and '是否允许评论' (Is Comment Allowed) with a dropdown menu showing '是' (Yes). A '提交' (Submit) button is located at the bottom of the form. The footer of the page contains the copyright notice: '© Copyright By 18ggg.com 2020'.

- 文章主页效果展示

百度推出针对疫情dwddqw防控的智能外呼平台，免费用

tgz2022 2020/06/07

测试

版权声明：本文版权归作者所有，转载请注明出处。

wewev

1 评论 编辑 返回首页

编辑文章

- 编辑文字Form表格效果图

编辑文章

标题

百度推出针对疫情dwddqw防控的智能外呼平台，免费用

内容

wewev

wewev

摘要

ewvewv

ewvewv

删除文章

- 说明：由于删除文章功能是最最后加入的，写报告时还没加入删除文章功能，所以本报告有些截图可能看不到删除栏，是正常的，但实际上是都有的。
- 设计过程

这个删除相当简单（我评论使用的屏蔽，屏蔽比直接删除的业务逻辑多），删除文章代码如下：

```
@main.route('/delete/<int:id>', methods=['GET', 'POST'])
@login_required
@push_online(user)
def delete(id):
    post = Post.query.get_or_404(id) #找到要删除的文章
    if current_user != post.author and \
        not current_user.can(Permission.ADMINISTER):
        abort(403)

    db.session.delete(post) #删除
    db.session.commit() #确定

    user = User.query.filter_by(id =
current_user._get_current_object().id).first()
    return redirect(url_for('.user', username=user.username))
```

删除后跳转到到改用户主页

- 效果展示



用户管理

设计User类

主要分以下几步进行设计：

- (1) User类需要哪些属性?
- (2) User类主键是哪个? 需要哪些约束?
- (3) User类需要跟哪个Model类相关联? 相关联的话又跟这个Model类的哪个属性建立建立
- (4) User类需要哪些常用方法?
- (5) 修改资料时User类对应的方法如何设计?

- User类属性定义

[illegible]

```
        backref=db.backref('followed', lazy='joined'),
        lazy='dynamic', cascade='all, delete-orphan')
    comments = db.relationship('Comment', backref='author', lazy='dynamic')
```

- User类请求json接口方法

```
def to_json(self):
    json_user = {
        'url': url_for('api.get_user', id=self.id, _external=True),
        'username': self.username,
        'member_since': self.member_since,
        'last_seen': self.last_seen,
        'posts': url_for('api.get_user_posts', id=self.id, _external=True),
        'followed_posts': url_for('api.get_user_followed_posts',
                                   id=self.id, _external=True),
        'post_count': self.posts.count()
    }
    return json_user
```

功能实现及展示

用户注册

在这里使用到了邮箱验证功能，只有用户在规定时间内点击激活链接才能注册成功。

- 注册界面

注册

邮箱

20182131141@m.scnu.edu.cn

用户名

tgz2023

密码

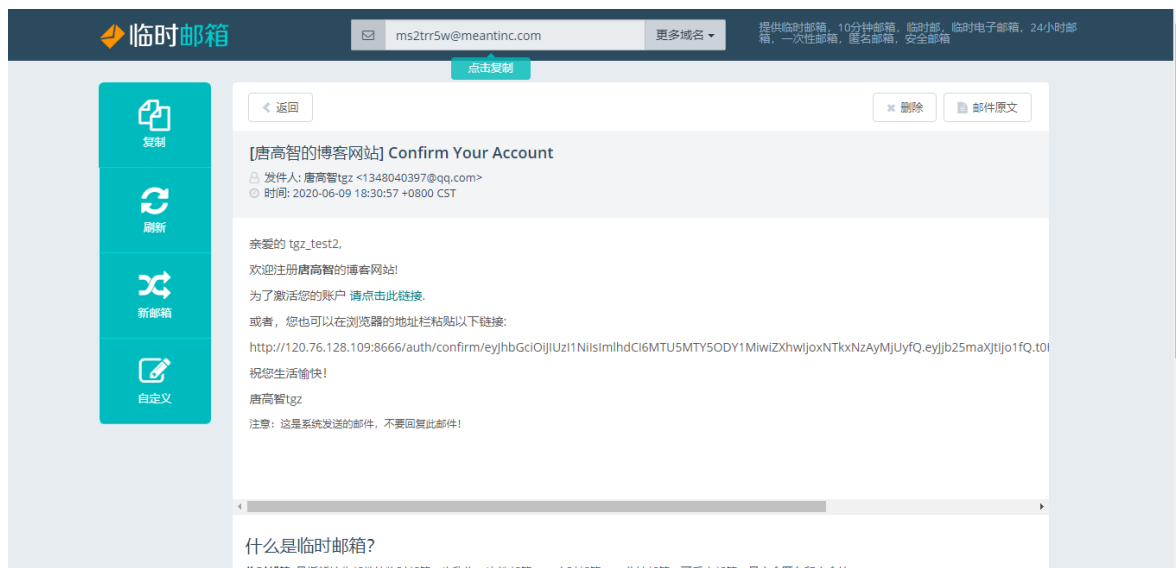
•••••

确认密码

•••••

注册

- 使用STMP发送激活邮件



用户登录

在这里使用到了模块 `flask_login`

用户登录时，先检查用户填写的邮箱是否在SQLite数据库中，还有填写的信息是否完整，前奏工作做好后然后再交由 `login_user` 处理

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user is not None and user.verify_password(form.password.data):
            login_user(user, form.remember_me.data)
            return redirect(request.args.get('next') or url_for('main.index'))
        flash(u'无效的用户名或密码! ')
    return render_template('auth/login.html', form=form)
```

- 登陆界面效果图

A screenshot of a login interface on a light green background. At the top, the title '登录' (Login) is displayed in a large, bold, dark blue font. Below the title is a horizontal line. Underneath the line, the label '邮箱' (Email) is positioned to the left of a white rectangular input field. Below the email field, the label '密码' (Password) is positioned to the left of another white rectangular input field. Below the password field, there is a small square checkbox followed by the text '自动登录' (Auto login). At the bottom of the form, there is a grey rectangular button with the text '登录' (Login) in white.

- 用户重置密码(发送邮箱，然后跳转到修改密码界面)

A screenshot of a reset password interface on a light green background. At the top, the title '重置密码' (Reset Password) is displayed in a large, bold, dark blue font. Below the title is a horizontal line. Underneath the line, the label '邮箱' (Email) is positioned to the left of a white rectangular input field. Below the email field, there is a grey rectangular button with the text '重置密码' (Reset Password) in white.

用户主页

- 用户资料界面

[BLOG](#)[主页](#)[写文章](#)[新分类](#)[搜索](#)[在线用户](#)[管理评论](#)[账户](#)



tgz2020

唐高智 来自 广州

码农

注册时间: 06/06/2020. 最后上线: a few seconds ago.

6 篇文章. 3 条评论.

关注者: 0 关注了: 0

编辑资料

- 用户文章

tgz2020 的文章

百度推出针对疫情防控的智能外呼平台，免费用

tgz2020 2020/06/07 flask

版权声明: 本文版权归原作者所有, 转载请注明出处。

二问

[0 评论](#)[编辑](#)[详情](#)

分类管理

设计Category类

- Category类属性定义

```
class Category(db.Model):
    __tablename__ = 'categories'
    id = db.Column(db.Integer, primary_key=True)
    author_id = db.Column(db.Integer, db.ForeignKey('users.id'))
    name = db.Column(db.String(64), unique=True)
    posts = db.relationship('Post', backref='category', lazy='dynamic')
```

- Category类与User类相关联，即将该tag与具体用户挂钩起来，更直白点就是用户之间的tag不冲突，即使重名。

功能实现及展示

新建tag

- 设计分类Form

```
class CategoryForm(FlaskForm):
    title = StringField(u'tag标题', validators=[Required()])
    submit = SubmitField(u'提交')
```

- 新建tag界面

BLOG

主页

写文章

新分类

请输入您要查找的内容

搜索

在线用户

管理评论

账户

tag标题

提交

tag显示界面

- 主页tag显示



- 文章tag显示

测试3

tgz2020 2020/06/07

flask

版权声明：本文版权归作者所有，转载请注明出处。

师德师风

0 评论

编辑

详情

- 文章tag选择（下拉选择框）

分类

flask

flask

是否公开

是

是否允许评论

是

提交

评论管理

设计Comment类

主要分以下几步进行设计：

- (1) Comment类需要哪些属性？
- (2) Comment类主键是哪个？需要哪些约束？
- (3) Comment类需要跟哪个Model类相关联？相关联的话又跟这个Model类的哪个属性建立建立
- (4) Comment类需要哪些常用方法？

- Comment类属性定义

```
class Comment(db.Model):
    __tablename__ = 'comments'
    id = db.Column(db.Integer, primary_key=True)
    author_id = db.Column(db.Integer, db.ForeignKey('users.id'))
    body = db.Column(db.Text)
    body_html = db.Column(db.Text)
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    disabled = db.Column(db.Boolean)
    post_id = db.Column(db.Integer, db.ForeignKey('posts.id'))
```

- Comment类请求json接口方法

```
def to_json(self):
    json_comment = {
        'url': url_for('api.get_comment', id=self.id, _external=True),
        'post': url_for('api.get_post', id=self.post.id, _external=True),
        'body': self.body,
        'body_html': self.body_html,
        'timestamp': self.timestamp,
        'author': url_for('api.get_user', id=self.author_id, _external=True),
    }
    return json_comment
```

功能实现及展示

新建评论

- 设计评论Form

```
class CommentForm(FlaskForm):
    body = PageDownField(u'留言', validators=[Required()])
    submit = SubmitField(u'提交')
```

- 新建评论请求接口

```
@main.route('/post/<int:id>', methods=['GET', 'POST'])
def post(id):
    post = Post.query.get_or_404(id)
```

```

form = CommentForm()
if form.validate_on_submit():
    comment = Comment(body=form.body.data,
                      post=post,
                      author=current_user._get_current_object())
    db.session.add(comment)
    flash(u'留言成功')
    return redirect(url_for('.post', id=post.id, page=-1))
page = request.args.get('page', 1, type=int)
if page == -1:
    page = (post.comments.count() - 1) / \
           current_app.config['FLASKY_COMMENTS_PER_PAGE'] + 1
pagination = post.comments.order_by(Comment.timestamp.asc()).paginate(
    page, per_page=current_app.config['FLASKY_COMMENTS_PER_PAGE'],
    error_out=False)
comments = pagination.items
return render_template('post.html', post=post, form=form,
                      comments=comments, pagination=pagination)

```

- 新建评论效果展示



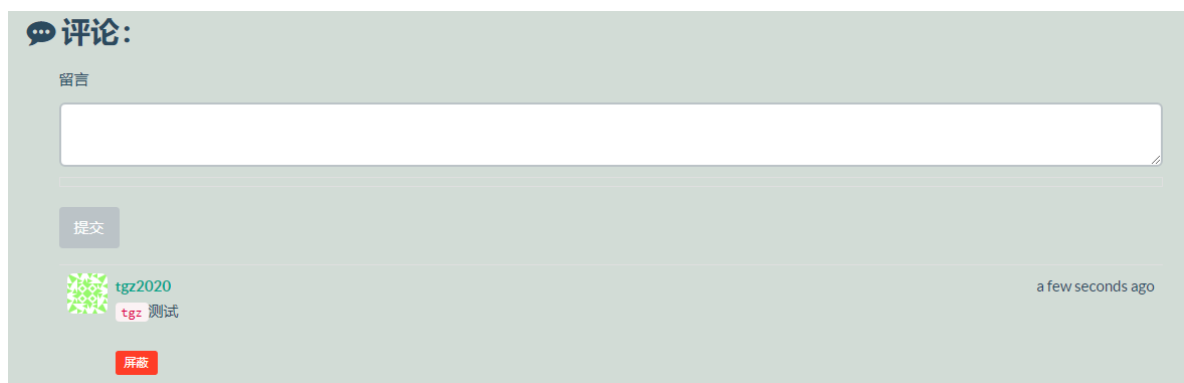
评论:

留言

tgz 测试

提交


- 评论展示



评论:

留言

提交


 tgz2020
 tgz 测试
 屏蔽

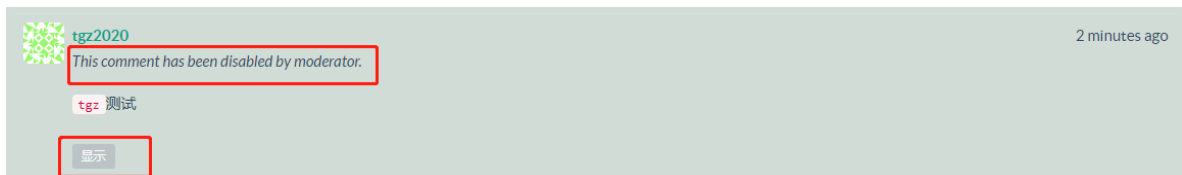
a few seconds ago

删除评论（屏蔽评论）

这里并不是实际上删除评论，而是为Comment类定义一个 `disable` 属性，要想"删除"掉这个评论，将 `disable` 这个属性的值设置为 `True` 即可（其实删除也就是使用orm直接delete而以，不是难事，**反而屏蔽业务逻辑要多点**）

```
@main.route('/moderate/disable/<int:id>')
@login_required
@permission_required(Permission.MODERATE_COMMENTS)
def moderate_disable(id):
    comment = Comment.query.get_or_404(id)
    comment.disabled = True
    db.session.add(comment)
    return redirect(url_for('.moderate',
                             page=request.args.get('page', 1, type=int)))
```

- 屏蔽评论效果展示



个人评论管理

有一个专门页面负责该用户评论管理，显示该用户所有评论

- 设计方案

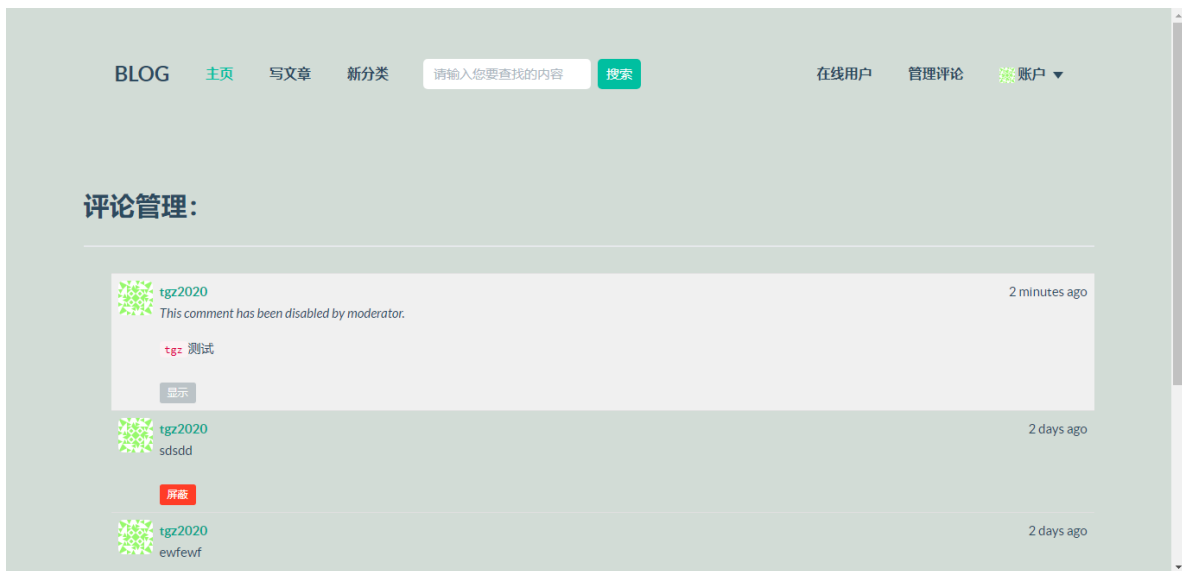
使用 `filter_by()` 过滤器即可只筛选该用户所有评论

```
@main.route('/moderate')
@login_required
@permission_required(Permission.MODERATE_COMMENTS)
def moderate():
    page = request.args.get('page', 1, type=int)

    #使用filter_by()过滤器即可只筛选该用户所有评论
    pagination = Comment.query.filter_by(author_id =
current_user._get_current_object().id).order_by(Comment.timestamp.desc()).paginate(
        page, per_page=current_app.config['FLASKY_COMMENTS_PER_PAGE'],
        error_out=False)

    comments = pagination.items
    return render_template('moderate.html', comments=comments,
                           pagination=pagination, page=page)
```

- 效果展示



搜索功能

设计过程

其实这个搜索功能也不难，前面基础做好了，这步只要分解成两步来执行

- 设计搜索框，并把输入的内容作为请求内容的一部分
- 设计search_result()接口，传入搜索keyword，然后使用orm的filter_by()和or_()函数进行搜索，搜索目标为文章标题、摘要、主题包括keyword即可，用到的函数是xx.contains()

搜索框代码及展示

- 搜索框使用Bootstrap实现，代码如下：

```
<form class="navbar-form navbar-left" role="search" method="get"
action="search_result" target="_blank">
  <div class="form-group">
    <input type="text" class="form-control" name="keyword" placeholder="请输入您要查找的内容">
  </div>
  <button type="submit" class="btn btn-primary">搜索</button>
</form>
```

- 搜索框效果展示



搜索结果

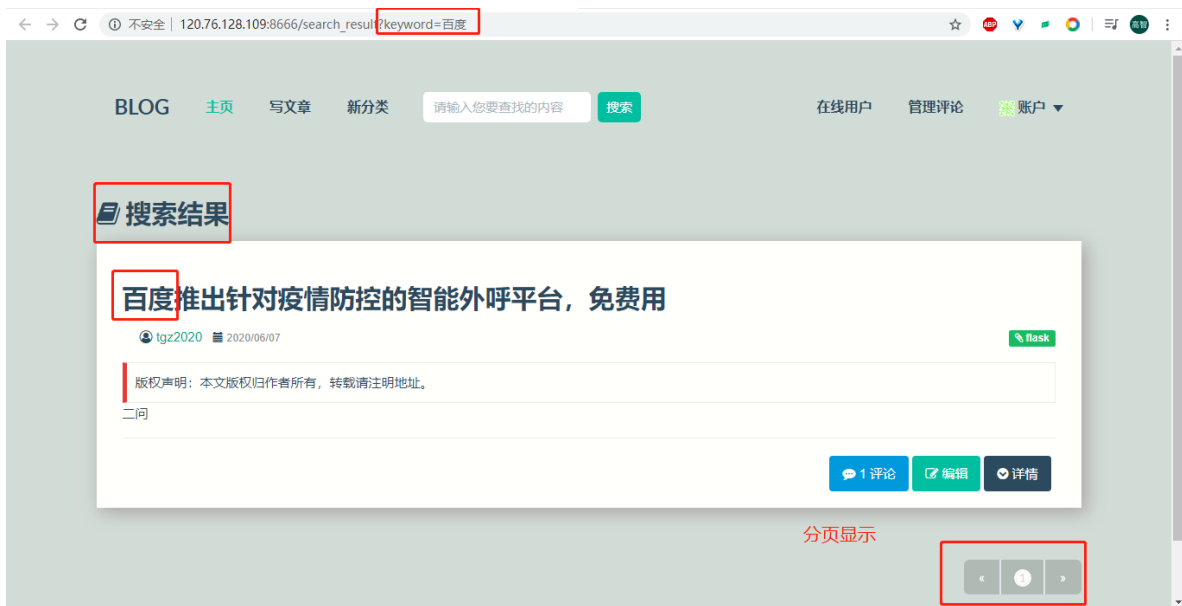
- 后端接口代码如下

```
@main.route('/search_result', methods=['GET'])
def get_search_result():
    page = request.args.get('page', 1, type=int)
    keyword = request.args.get("keyword")
    # print(keyword)
```

```
#https://www.jianshu.com/p/ca9bc9f8adab
pagination = Post.query.filter_by(author_id =
current_user._get_current_object().id).filter(or_(Post.title.contains(keyword),
Post.body.contains(keyword),\
Post.summury.contains(keyword),\
)).order_by(Post.timestamp.desc()).paginate(
page, per_page=current_app.config['FLASKY_POSTS_PER_PAGE'],
error_out=False)

posts = pagination.items
return render_template('search_result.html', user=user, posts=posts,
pagination=pagination)
```

• 效果展示



分页显示

设计过程

这个设计分两步执行（分页请求、获得数据）：

- 得到前端发送过来的页数请求page属性，若发送过来为1，表明请求第一页数据，默认为第一页；

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

- 后端获得该page目录下的数据，然后将该参数传送给模板html文件

```
pagination = Post.query.filter_by(author_id =
current_user._get_current_object().id).order_by(Post.timestamp.desc()).paginate(
#page属性利用
page, per_page=current_app.config['FLASKY_POSTS_PER_PAGE'],
error_out=False)

posts = pagination.items
#将该参数传送给模板html文件
return render_template('search_result.html', user=user, posts=posts,
pagination=pagination)
```

效果展示



装饰器Decorators在本项目中的应用

装饰器介绍

装饰器(Decorators)是 Python 的一个重要部分。简单地说：他们是修改其他函数的功能的函数。他们有助于让我们的代码更简短，也更Pythonic（Python范儿）。

为什么需要装饰器？

在博客网站中，很多功能需要检查是否已经登录才可以使用，我们总不可能在每个函数中都重写这段代码，一来不容易统一唯一（过于分散，改一处地方差不多要改很多地方），二来代码不简洁，一百行代码中有十几行是登录检测函数。

所以装饰器就派上用场了，在一个功能函数前应用装饰器的话，我们只需要在函数定义前@这个装饰器，然后系统执行这个函数时会先执行装饰器函数，这样不仅达到检查目的，还大大简洁代码。

- 思考：好像封装成普通函数也可以的，为什么要专门用装饰器呢？

其实这里用普通函数和用装饰器感觉没区别。只是装饰器放在函数前头更醒目点，要是调用函数的话，可能不一定能那么直观了。另外很多内置模块方法也用到装饰器，这是不可能要求用户自己在某个函数手动调用的..装饰器作用还大着。

装饰器在本项目中的应用

`push_online()` 函数，这是装饰器之一，用户在请求任意功能之前都会之前这个，记录活动时间，从而方便我们统计在线用户。


```
def push_online(user):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            user = current_user._get_current_object()
            isAnonymousUser = isinstance(user, AnonymousUser) #没登陆的话值为True
            if not isAnonymousUser:
                mark_online(user) #记录在线信息
            return f(*args, **kwargs)
        return decorated_function
    return decorator
```

应用例子如下：

```
@main.route('/unfollow/<username>')
@login_required
@permission_required(Permission.FOLLOW)
@push_online(user)
def unfollow(username):
    pass
```

前端部分

前端框架Bootstrap

前端框架Bootstrap介绍



Bootstrap是一个免费的前端框架，并且是基于html和JavaScript、css三者开发的框架，主要用于响应式网站上的结构和布局，Bootstrap的出现主要是简化Web工作者的工作，其中还包括对JavaScript中的动态调整。

Bootstrap优点

- 提供一套美观大方地界面组件；
- 提供一套优雅的 HTML+CSS 编码规范；
- 让我们的 Web 开发更简单，更快捷；

为什么选择Bootstrap

- twitter 出品

首先，Bootstrap 出自 twitter，大厂出品，并且开源，自然久经考验，减少了测试的工作量。站在巨人的肩膀上，不重复造轮子。

- 丰富的组件

Bootstrap 的[HTML组件](#)和[JS组件](#)非常丰富，并且代码简洁，非常易于修改，你完全可以在其基础之上修改成自己想要的任何样子。这是工作效率的极大提升。

对于一个博客网站来说，丰富的组件有利于提高我们的开发效率，**实际开发中只需要传入数据，就可以显示出我们想要的效果**，若不满意还可以修改参数和组件。

设计过程

各自分工

每个html文件负责每一个模块的显示，如果需要用到，直接调用即可，而不必有每次都重写，例如跟根模板 `base.html` (只显示部分代码)

```
{% extends "bootstrap/base.html" %}
{% block html_attribs %} lang="zh-CN"{% endblock html_attribs %}
{% block title %}唐高智博客网站{% endblock %}
....
{% block scripts %}
    {{ super() }}
    {{ moment.include_moment() }}
    <script src="https://cdn.bootcss.com/jquery/3.2.0/jquery.min.js"></script>
    <script type="text/javascript" src="{{ url_for('static',
filename='mystyle.js') }}"></script>
{% endblock %}
```

渲染流程

- 后端使用Flask内置模块 `render_template()` 方法将所需要展示参数传递给Bootstrap，例如新建文章表格部分

```
form = PostForm()
return render_template('write_article.html', form=form, posts=posts,
                      pagination=pagination)
```

- 前端Bootstrap接收参数，自动渲染（这个例子中传入的是form）

```
{% block page_content %}
    <div class="container">
        <div class="row" style="padding:30px 20px;">
            <div class="col-md-12">
                {{ wtf.quick_form(form) }}
            </div>
        </div>
    </div>
{% endblock %}
```

从上面前前后后分工合作来看，项目流程清晰明了，甚至不需要等待后端是否已经完成项目，前端人员就可以把任务完成，只需要实现规定传进的参数即可。

界面一览

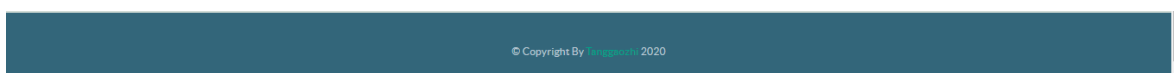
- 根模板

这里的根模板跟C++中的父类差不多，而其他模板都要继承这个根模板base.html。这个base.html主要负责博客网站的头部和尾部。其余模板文件都继承这个，自然而然一直显示头部和尾部。

- 头部



- 尾部



- 主页



- 写文章

← → ↺ ① 不安全 | 120.76.128.109:8666/new_article ☆ 404 100% 100% 100% 100%

BLOG 主页 写文章 新分类

请输入您要查找的内容 搜索

在线用户 管理评论 账户 ▾

标题

内容

摘要

分类

flask ▾

- 新建分类

BLOG 主页 写文章 新分类

请输入您要查找的内容 搜索

在线用户 管理评论 账户 ▾

tag标题

提交

© Copyright By tangqinchi 2020

- 在线用户

BLOG 主页 写文章 新分类

请输入您要查找的内容 搜索

在线用户 管理评论 账户 ▾

在线用户

用户昵称

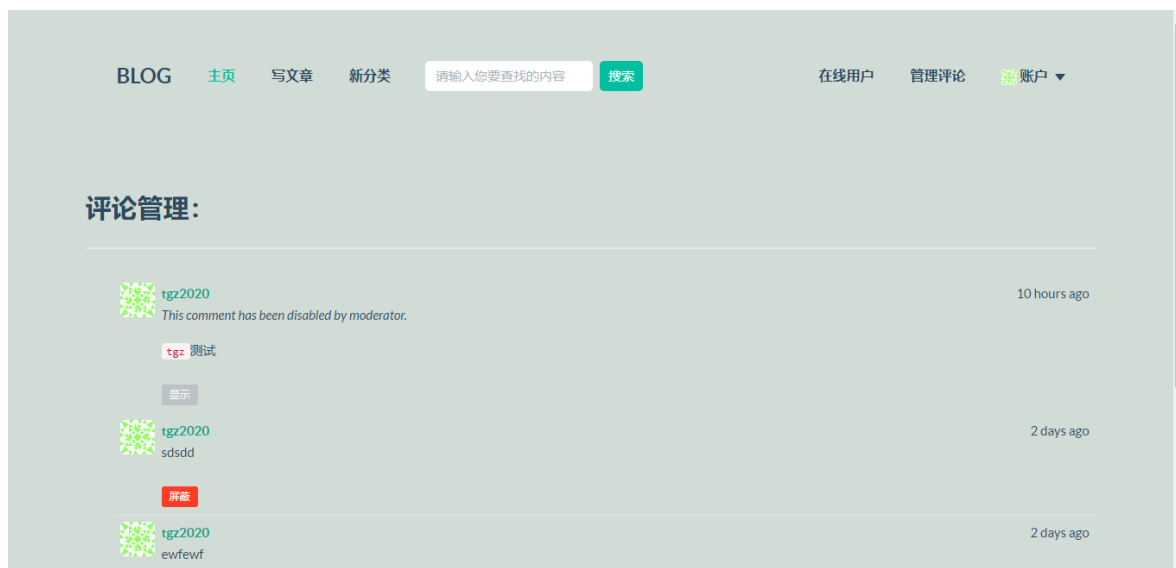
活跃时间

tgz2020

a few seconds ago

© Copyright By tangqinchi 2020

- 管理评论



- 用户主页

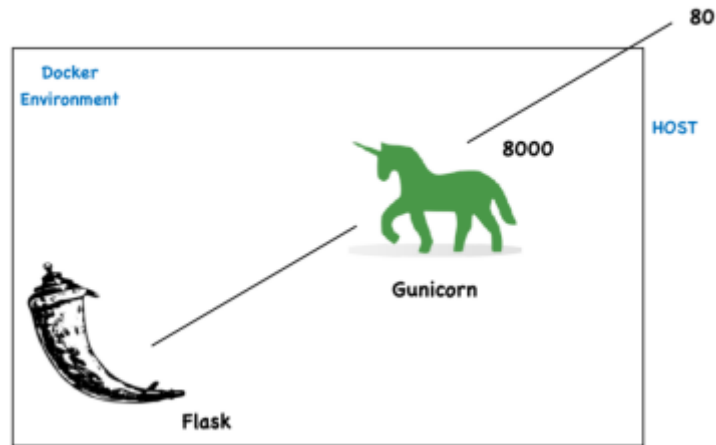


- 搜索结果



项目部署

Gunicorn+Nignx部署Flask



Gunicorn 绿色独角兽 是一个Python WSGI UNIX的HTTP服务器。这是一个pre-fork worker的模型，从Ruby的独角兽（Unicorn）项目移植。该Gunicorn服务器大致与各种Web框架兼容，只需非常简单的执行，轻量级的资源消耗，以及相当迅速。

启动时，我们只需要执行以下语句便可以实现局域网内访问，若需要外网访问，可以省略127.0.0.1

- 使用Gunicorn部署Flask命令（本案例实际使用命令）

```
! [img]
(file:///C:/Users/20182/AppData/Local/Temp/ksohtml213588/wps1.jpg)
```

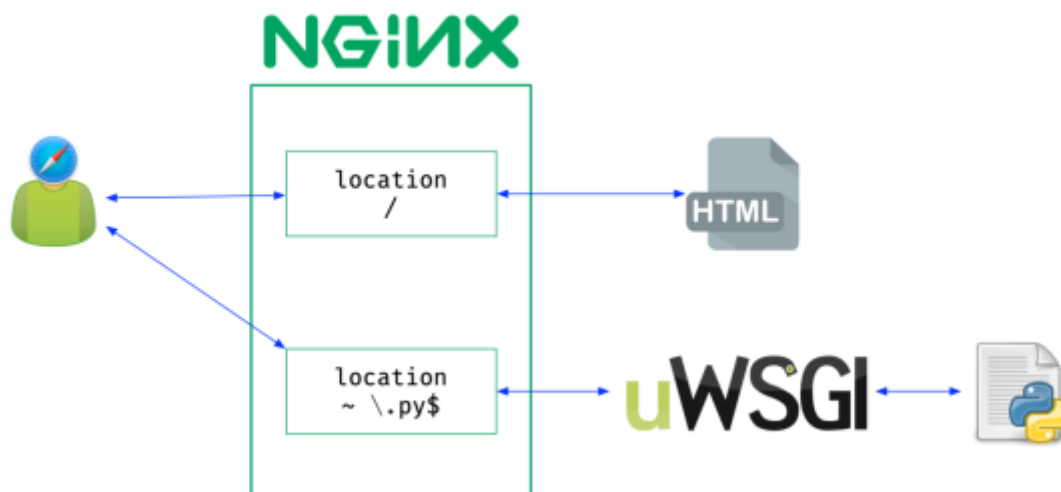
```
nohup gunicorn -w 10 -b 0.0.0.0:8666 manage:app &disown
```

使用Nignx反向代理，可以进行域名访问并处理静态文件。

- 配置Nginx文件

```
server {
    listen 80;
    server_name example.org; # 这是HOST机器的外部域名，用地址也行

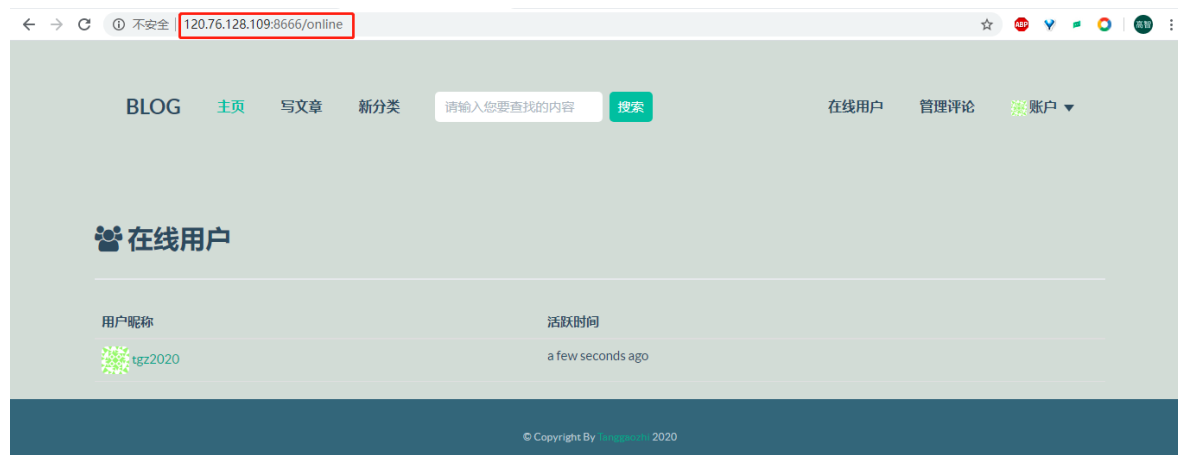
    location / {
        proxy_pass http://127.0.0.1:8000; # 这里是指向 gunicorn host 的服务地址
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```



实际情况

由于此次博客网站使用到了flask的内置模块 `render_template`，可以自主指定模块文件，所以Nignx配置这一步骤可以省略，直接使用Gunicorn部署即可。

- 效果展示



其中，120.76.128.109是我的阿里云学生服务器公网IP，8666是我为这个博客网站开放的端口号。

项目总结

关于本报告

本人一开始写这报告没发现什么困难，但是写着写着发现需要写的实在太多了，如果都详细讲，甚至可以出一本书，所以到后面写的比较简洁（甚至只贴出效果图），而且期末考降至，需要专心复习，望老师能理解。

关于本项目进程

基本按照项目计划书中的流程进行，但中途中遇到一些困难，有些功能实现拖了好久，比如在线用户功能的实现（一开始想的太复杂）

学习感悟

此次博客项目将我所学到的后端知识运用于实际当中，真正感觉到动手能力的重要性，特别是在功能实现的过程中，能学到的东西或者对某一知识的理解往往比书本上来的更多、深刻！