

开源软件基础大作业

基于 Flask 实现的匿名论坛

Anonymous Forum Website Based on Flask

学 生 姓 名: 赵书彬

学 号: 201873030

学 生 姓 名:

学 号:

学 生 姓 名:

学 号:

学 生 姓 名:

学 号:

完 成 日 期: 2020.1.8

大连理工大学

目 录

引 言	1
1 项目背景以及意义	2
1.1 A 岛匿名版的基本功能	2
1.2 编写的意义	3
2 项目创新点	4
2.1 使用小饼干机制(kookies)实现发帖人的匿名	4
2.1.1 小饼干的展示	4
2.1.2 小饼干的创新点	4
2.2 对内容的编码解码，实现关键词的隐藏	5
2.2.1 编码器译码器展示	5
2.2.2 编码器译码器的创新点	5
3 项目的设计	6
3.1 体系结构	6
3.2 项目系统功能	7
3.2.1 API 模块	7
3.2.2 用户操作模块	7
3.2.3 查看页面模块	7
3.3 数据库设计	8
3.3.1 物理模型设计	8
3.3.2 表定义	8
4 如何实现	10
4.1 View 模块	10
4.1.1 前情提要：flask 中 view 层的实现方式	10
4.1.2 项目整体 View 层结构	11
4.2 Template 模块	14
4.2.1 Flask 中的模板渲染	14
4.2.2 Bootstrap	14
4.2.3 portal 模板（portal.html）	14
4.2.4 home 模板（home.html）	14
4.2.5 viewpost 模板（viewpost.html）	14
4.3 Model 模块	15

4.3.1	数据库连接对象的建立	15
4.3.2	user 类	15
4.3.3	posts 类	16
4.4	控件	17
4.4.1	发送验证邮件 (mail_sender)	17
4.4.2	返回错误信息 (errors.py)	17
4.4.3	生成表单(forms.py)	18
4.4.4	获得新发言饼干 (cookiemaker.py)	18
5	测试	19
5.1	测试环境	19
5.1.1	服务端	19
5.1.2	客户端	19
5.2	页面浏览测试	19
5.2.1	未登录状态浏览	19
5.2.2	已登录状态浏览	22
5.3	用户操作测试	23
5.3.1	注册	23
5.3.2	登陆/注销	27
5.3.3	获取新饼干	28
5.3.4	发表新串	29
5.3.5	删除串	30
5.3.6	评论	31
5.3.7	编码/解码	32
5.4	移动端适配测试	34
6	项目分析	35
6.1	优点	35
6.1.1	预定功能基本实现	35
6.1.2	新功能的开发	35
6.2	缺点以及改进设想	35
参 考 文 献	37
附录 A	开发环境所需运行库	38
成员贡献及感想	39

引 言

“人，是会思考的芦苇。”——帕斯卡，《思想录》

1 项目背景以及意义

受启发于 A 岛匿名版(<https://adnmb3.com/>), 我们产生了开发一个功能类似的匿名论坛的想法, 且在基本功能开发完成后改进匿名机制和管理方法。并且通过对发言内容的编码解码, 使发言内容有一定的匿名性(不执行操作则无法读取到明文)。同时利用 Bootstrap 的网格系统进行 PC 和移动设备的显示适配。

1.1 A 岛匿名版的基本功能



图 1.1 A 岛匿名版

- (1) 匿名论坛, 对发帖人的真实身份进行隐藏保护
- (2) 每一个串下有若干回复, 回复也作为子串
- (3) 对串和子串可以进行回复
- (4) 对用户进行真实性验证(邮箱)
- (5) 版块分类
- (6) 小饼干机制实现发帖人的辨别

1.2 编写的意义

通过编写该感兴趣的项目，学习 Flask+Bootstrap 的 web 开发栈。同时学习使用 Git，增强团队合作编码的能力。

2 项目创新点

2.1 使用小饼干机制(kookies)实现发帖人的匿名

2.1.1 小饼干的展示



图 2.1 小饼干的示例

2.1.2 小饼干的创新点

- (1) 匿名
- (2) 发帖人身份识别
- (3) 方便于用户管理

每一个用户可以同时拥有一个饼干作为身份识别的标识，也可以获取新的饼干，当然旧的饼干也就弃用了。这样可以实现在后台数据安全的前提下的用户匿名+身份识别。同时因为一个账户只能拥有一个非固定的小饼干，大大减少了用户新建身份而新建账户的需求，从而方便了用户管理。

2.2 对内容的编码解码，实现关键词的隐藏

2.2.1 编码器译码器展示



图 2.2 编码器译码器的示例

2.2.2 编码器译码器的创新点

(1) 对聊天内容可进行编码，使其原内容变成非明文，从而避开某些不必要的争端或者审查。

(2) 主页预览时，对内容进行非明文的展示，避免无意义的访问，减轻服务器的负担，增加访问的筛选性。

3 项目的设计

3.1 体系结构

因为使用了 Flask 这一轻量级的 Web 框架，本项目采用了 B/S 结构。同理，程序设计模式也为 Flask 所决定的 MVT 模式。

Flask 也是 MVC 框架。但是 Flask 框架内部作为控制器的角色，负责了接收用户请求和转发请求的工作，Flask 里更关注的是模型(Model)、模板(Template)和视图(Views)，故称之为 MVT 模式。

M 全拼为 Model，与 MVC 中的 M 功能相同，负责和数据库交互，进行数据处理。V 全拼为 View，与 MVC 中的 C 功能相同，接收请求，进行业务处理，返回应答。T 全拼为 Template，与 MVC 中的 V 功能相同，负责封装构造要返回的 html。

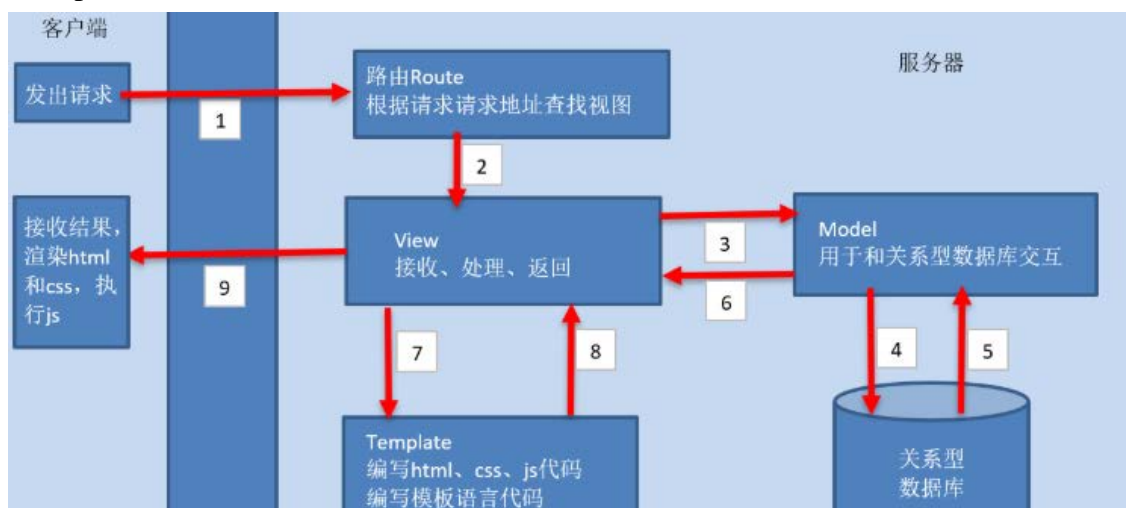


图 3.1 MVT 架构示意图

3.2 项目系统功能

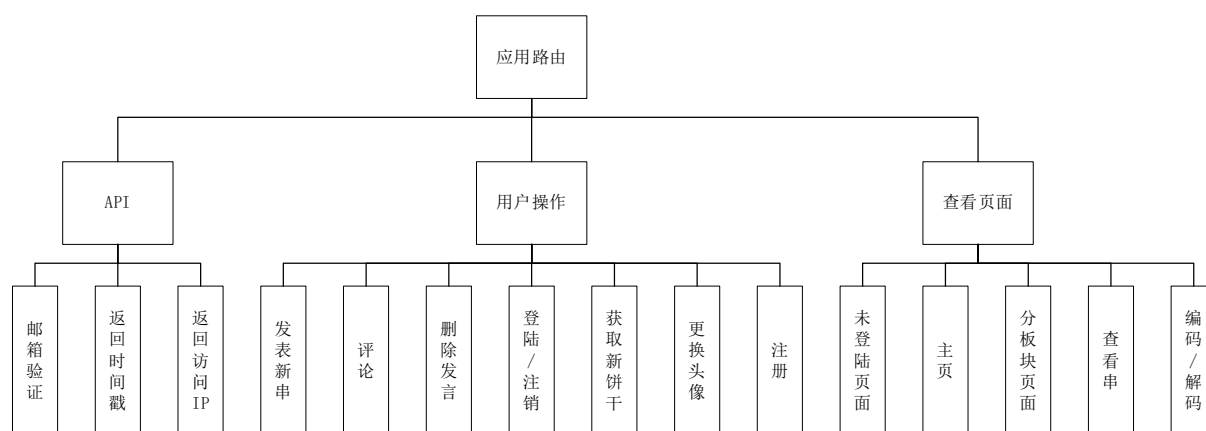


图 3.2 项目功能图

3.2.1 API 模块

(1) 邮箱验证功能。对注册页面发出的验证邮件中给出的验证链接给予响应，并将对应用户标记为已验证用户，使其能正常使用论坛功能。

(2) 返回时间戳功能。返回服务器的时间，用于核对客户机与服务器是否在同一时区/服务器时间是否正确。

(3) 返回浏览器访问 IP 功能。

3.2.2 用户操作模块

(1) 注册功能。用户输入用户名，密码，确认密码，注册邮箱进行注册，并通过验证邮件进行邮箱验证，通过后登陆即可进行其他操作。

(2) 登陆/注销。用户通过登录页面登陆，通过点击注销-确定注销进行登出。

(3) 更换头像。用户在登陆情况下点击头像，进行可选头像的更换。

(4) 获取新饼干。用户在登陆情况下点击饼干-获取新饼干，得到新的发言小饼干。

(5) 发表新串。用户在登陆情况下，点击 new，进行相关内容填写，即可发表新帖。

(6) 评论。用户在登陆情况下浏览串时，点击回复，填写相关内容。

(7) 删除发言。用户在查看自己发言时可以进行删除。

3.2.3 查看页面模块

(1) 未登录页面。限制用户只可查看内容而不可以进行其他操作。

(2) 主页。显示时间线上的串，展示每个串的预览内容。

(3) 分板块页面。按板块分类显示串和预览内容。

(4) 查看串。查看串的详细内容和评论内容。同时编码解码功能也集成在此页面中。

3.3 数据库设计

3.3.1 物理模型设计

本项目不需要过于复杂的模型关系，只需要一个 `user` 表存储用户信息，一个 `post` 表用于存储发言内容。其中 `post` 和 `user` 是一对一关系，通过应用层的 `kookie`（小饼干编号）来实现唯一联系。

3.3.2 表定义

posts

Fields

Field	Type	Null	Key	Default	Extra	Comment
id	int(11)	NO	PRI	(NULL)	auto_increment	每个主串或子串的编号
poster	varchar(10)	YES		(NULL)		该串的作者的小饼干号
poster_ip	varchar(50)	YES		(NULL)		该串的作者的 IP
head	tinyint(1)	YES		(NULL)		该内容是否为主串
next	int(11)	YES		(NULL)		该串的下一个子串的编号
post_time	datetime	YES		(NULL)		该串的发送时间
title	text	YES		(NULL)		该串的标题
content	text	YES		(NULL)		该串的内容
section	varchar(40)	YES		(NULL)		该串的分类
avatar	int(11)	YES		(NULL)		该串的显示头像
withpic	tinyint(1)	YES		(NULL)		该串是否有图像
pic_route	varchar(100)	YES		(NULL)		图像的路由指针
ider	varchar(256)	YES		(NULL)		唯一识别符用于反查主键
replies	int(11)	YES		(NULL)		主串情况下 post 的回复数
topped	tinyint(1)	YES		(NULL)		是否置顶
update_time	datetime	YES		(NULL)		最后更新时间
no_show	tinyint(1)	YES		0		是否展示在时间线上

Indexes

Table	Non unique	Key name	Seq in index	Column name	Collation	Cardinality	Sub part	Packed	Null	Index type	Comment	Index comment
posts	0	PRIMARY	1	id	A	11	(NULL)	(NULL)		BTREE		
posts	0	id	1	id	A	11	(NULL)	(NULL)		BTREE		

图 3.3 post 表定义

user

Fields												
Field		Type		Null	Key	Default	Extra			Comment		
id		int(11)		NO	PRI	(NULL)	auto_increment			注册编号		
username		varchar(40)		YES		(NULL)				用户名		
password		varchar(40)		YES		(NULL)				密码		
email		varchar(80)		YES	UNI	(NULL)				注册邮箱/登录账号		
create_time		datetime		YES		(NULL)				注册时间		
active_time		datetime		YES		(NULL)				上次活动时间		
active_ip		varchar(50)		YES		(NULL)				上次活动 IP		
password_hash		varchar(256)		YES		(NULL)						
password_cmp		varchar(50)		YES		(NULL)				用于登陆时对比		
kookies		varchar(10)		YES		(NULL)				小饼干编号		
admin		tinyint(1)		YES		(NULL)						
confirmed		tinyint(1)		YES		(NULL)				注册邮箱是否验证		
avatar		int(11)		YES		(NULL)				头像编号		
oldkookies		text		YES		(NULL)				使用过的小饼干编号		
fav_color		varchar(10)		YES		(NULL)				使用 RGB		
cited		text		YES		(NULL)						
Indexes												
Table	Non unique	Key name	Seq in index	Column name	Collation	Cardinality	Sub part	Packed	Null	Index type	Comment	Index comment
user	0	PRIMARY	1	id	A	0	(NULL)	(NULL)		BTREE		
user	0	email	1	email	A	0	(NULL)	(NULL)	YES	BTREE		

图 3.4 user 表定义

4 如何实现

4.1 View 模块

4.1.1 前情提要：flask 中 view 层的实现方式

对于用户浏览器发出的 URL 请求，Flask 提供了处理函数，这个函数是处理某个请求的处理函数，Flask 官方把它叫做视图函数（view function），你可以理解为“请求处理函数”。所谓的“注册”，就是给这个函数戴上一个装饰器帽子。我们使用 `app.route()` 装饰器来为这个函数绑定对应的 URL，当用户在浏览器访问这个 URL 的时候，就会触发这个函数，获取返回值，并把返回值显示到浏览器窗口。为了便于理解，可以把整个 Web 程序看作是一堆这样的视图函数的集合：编写不同的函数处理对应 URL 的请求。

如图 4.1:

```
@app.route('/')
def hello():
    return 'Welcome to My Watchlist!'
```

图 4.1 user 表定义

填入 `app.route()` 装饰器的第一个参数是 URL 规则字符串，这里的 `/` 指的是根地址。我们只需要写出相对地址，主机地址、端口号等都不需要写出。所以说，这里的 `/` 对应的是主机名后面的路径部分，完整 URL 就是 `http://localhost:5000/`。如果我们这里定义的 URL 规则是 `/hello`，那么完整 URL 就为 `http://localhost:5000/hello`。整个请求的处理过程如下所示：

1. 当用户在浏览器地址栏访问这个地址，在这里即 `http://localhost:5000/`
2. 服务器解析请求，发现请求 URL 匹配的 URL 规则是 `/`，因此调用对应的处理函数 `hello()`
3. 获取 `hello()` 函数的返回值，处理后返回给客户端（浏览器）
4. 浏览器接受响应，将其显示在窗口上

4.1.2 项目整体 View 层结构

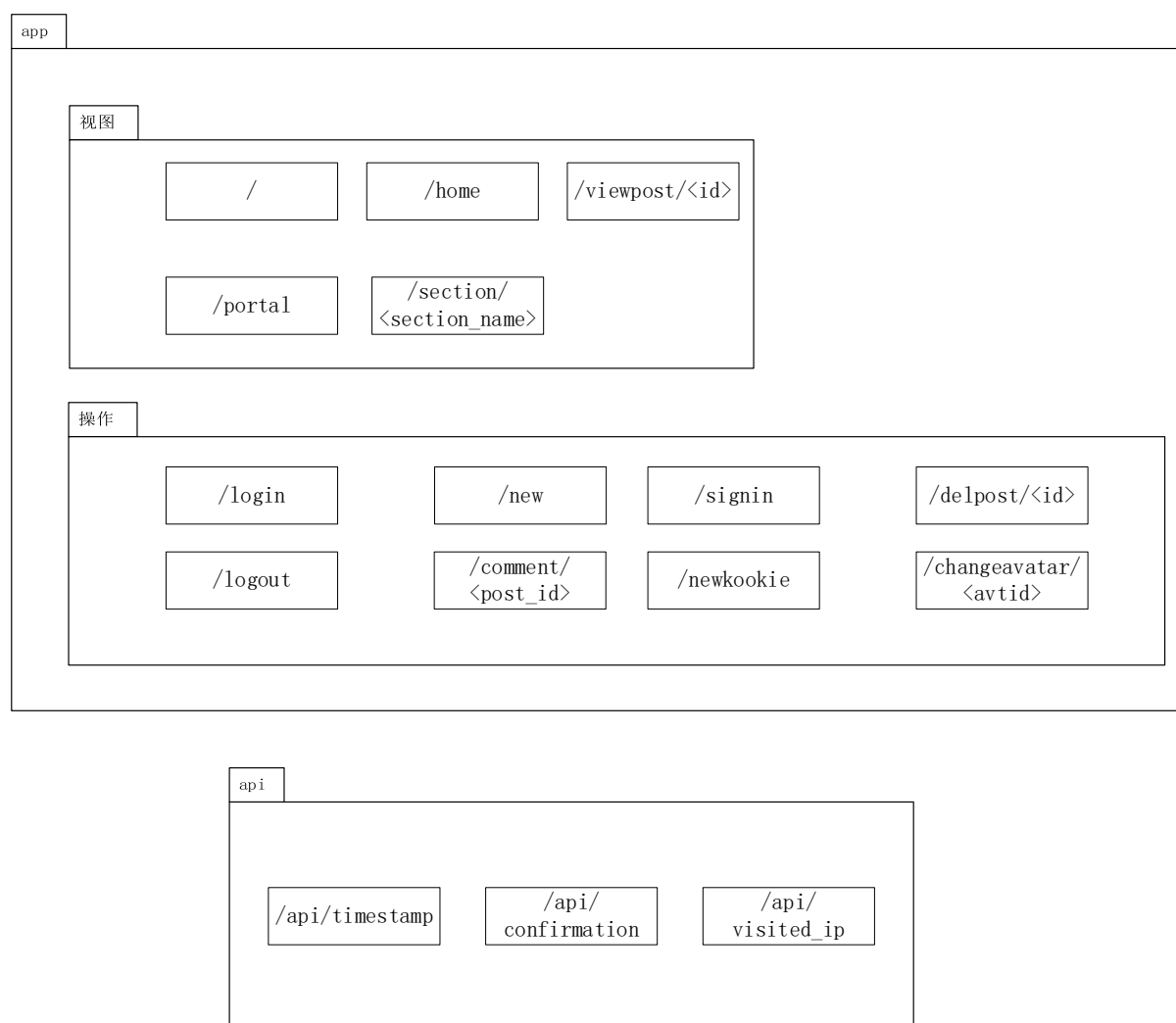


图 4.2 View 层结构

(1) 视图层的 **view function**。

视图函数主要用于将页面请求（GET 方法）路由到对应的 Template 上，以生成用户看到的页面。

如对默认页面(/)的请求返回页面的实现为：

```
@app.route('/', methods=['GET', 'POST'])
def home():
    # 已登录
    if not checklogin():
        return redirect(url_for('homepage'))
    nologin = request.values.get('nologin')
    # 未登录
    if request.remote_addr not in visited_ip:
        visited_ip.append(request.remote_addr)
    topped_posts = posts.query.filter(posts.topped == True).order_by(-posts.update_time).all()
    tenposts = posts.query.filter(posts.head == True).filter(posts.topped == False).filter(
        posts.no_show == False).order_by(-posts.update_time).all()
    return render_template('view/portal.html', topped_posts=topped_posts, tenposts=tenposts, userip=request.remote_addr,
        datetime=datetime.datetime.now(),
        loginform=loginform(), signinform=signinform(), nologin=str(nologin))
```

图 4.2 默认路由

通过此函数将默认地址请求路由至 view/portal.html 编写的模板中，并将本函数处理传入的参数渲染进入页面中，最终生成用户页面。在登陆的情况下返回主页，未登录的情况下返回未登录的主页。其中传递参数的获取依赖于 model 层的对象实例获取的数据库记录。

(2) 视图函数中的操作函数。

操作函数主要处理传递请求（GET/POST 方法）来对数据进行处理。在执行处理之前先检验用户权限（是否登陆，是否获得小饼干 kookie，是否验证邮箱），若验证通过则响应请求执行对应操作。

如切换头像的操作函数的实现为：

```
@app.route('/changeavatar/<avtid>', methods=['GET', 'POST'])
def changeavt(avtid):
    if checklogin():
        return redirect(url_for('home', nologin=1))
    if check_confirmation():
        return redirect(url_for('homepage', no_confirmation=1))
    account = session.get('account')
    kookie = User.query.filter(User.email == account).first().kookies
    User.query.filter(User.email == account).update({'avatar': avtid})
    try:
        db.session.commit()
        try:
            posts.query.filter(posts.poster == kookie).update({'avatar': avtid})
            db.session.commit()
        except Exception as e:
            db.session.rollback()
            print(e)
            return return500()
    except Exception as e:
        db.session.rollback()
        print(e)
        return return500()
    return redirect(url_for('homepage'))
```


图 4.3 更换头像的实现

在验证用户权限通过的前提下（是否登陆，是否获得小饼干 kookie，是否验证邮箱），通过此函数调用的 sql 连接对象，更新用户的头像 id，实现头像的更换。若验证不通过则提醒用户权限不足。

（3）API 函数。

将常用或可复用或独立性较强的数据处理操作接口独立出来方便调用。

API 接口实现为：

```
@app.route('/api/<kw>')
def api(kw):
    # print(kw)
    if kw == 'timestamp':
        return str(int((time.time())))
    if kw == 'datetime':
        return str(datetime.datetime.now())
    if kw == 'confirmation':
        psw_pash = request.values.get('code') # 现在确认码是（密码+时间戳）的HASH
        User.query.filter(User.password_hash == psw_pash).update({'confirmed': True})
        try:
            db.session.commit()
            return redirect(url_for('home', confirmation_ok=1))
        except Exception as e:
            db.session.rollback()
            print(e)
            return return500('Confirmation failed!')
    if kw == 'visited_ip':
        result = []
        for ip in visited_ip:
            result.append(str('ip:' + ip + checkip(ip)))
        return str(result)
    if kw == 'power_fee':
        room_id = request.values.get('id')
        print(room_id)
        return getfee(room_id)
    return return404()
```

图 4.2 API 接口的实现

如图 4.2 中，验证邮箱的/api/confirmation 接口实现中，在用户页面进行注册后，mail_sender.py 中的 sender()方法发送注册邮件，内含确认链接。而 api 实现中的 confirmation 模块将 URL 中的 code 部分取出并与数据库内存的用户校验码进行对比，若对比成功则将用户的邮箱验证状态改为已验证，否则返回验证失败页面。

4.2 Template 模块

4.2.1 Flask 中的模板渲染

在一般的 Web 程序里，访问一个地址通常会返回一个包含各类信息的 HTML 页面。因为我们的程序是动态的，页面中的某些信息需要根据不同的情况来进行调整，比如对登录和未登录用户显示不同的信息，所以页面需要在用户访问时根据程序逻辑动态生成。我们把包含变量和运算逻辑的 HTML 或其他格式的文本叫做模板，执行这些变量替换和逻辑计算工作的过程被称为渲染，这个工作由我们这一章要学习使用的模板渲染引擎——Jinja2 来完成。按照默认的设置，Flask 会从程序实例所在模块同级目录的 `templates` 文件夹中寻找模板。

Jinja2 的语法和 Python 大致相同，你在后面会陆续接触到一些常见的用法。在模板里，需要添加特定的定界符将 Jinja2 语句和变量标记出来，下面是三种常用的定界符：

`{{ ... }}` 用来标记变量。

`{% ... %}` 用来标记语句，比如 `if` 语句，`for` 语句等。

`{# ... #}` 用来写注释。

模板中使用的变量需要在渲染的时候传递进去。

4.2.2 Bootstrap

Bootstrap 是一个用于快速开发 Web 应用程序和网站的前端框架。Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的。基于该框架，我们可以免除大量的前端编写工作，直接使用框架提供的类来进行样式的设计。

4.2.3 portal 模板 (`portal.html`)

模板 `portal` 为未登录页面的渲染模板。需要传递给模板的参数为浏览器传入的 `ip`、数据库目前存在的串、当前日期、预设登录信息、预设注册信息、登录状态。

4.2.4 home 模板 (`home.html`)

模板 `home` 为已经登陆页面的渲染模板。需要传递给模板的参数为数据库内存在的串，置顶的串，用户在页面内选择的模块，是否登陆，用户是否有小饼干。

4.2.5 viewpost 模板 (`viewpost.html`)

模板 `viewpost` 为查看串具体内容的渲染模板。需要传递给模板的参数为该串的内容，用户的数据对象，预设评论信息。

4.3 Model 模块

Model 层用于与数据库等数据来源进行数据交互，并且传递给 View 层进行下一步的模板渲染。

4.3.1 数据库连接对象的建立

基于 SQLAlchemy，MySQL 服务器与 flask 应用程序建立连接。

```
import datetime
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from cookiemaker import *

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = "mysql+pymysql://StuG:x74rtw05@localhost:3306/nmb0"
app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'wtftwtft'
visited_ip = []
db = SQLAlchemy(app)
```

图 4.3 数据库连接

4.3.2 user 类

```
class User(db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(40), nullable=True)
    password = db.Column(db.String(40), nullable=True)
    email = db.Column(db.String(80), nullable=True, unique=True)
    create_time = db.Column(db.DateTime, nullable=True) # 注册时间
    active_time = db.Column(db.DateTime, nullable=True) # 上次活动时间
    active_ip = db.Column(db.String(50), nullable=True)
    # role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))
    password_hash = db.Column(db.String(256), nullable=True)
    password_cmp = db.Column(db.String(50), nullable=True) # 用于登陆时比对
    kookies = db.Column(db.String(10), nullable=True, default='00000000')
    admin = db.Column(db.Boolean, default=False)
    confirmed = db.Column(db.Boolean, default=False)
    avatar = db.Column(db.Integer, default=0)
    oldkookies = db.Column(db.Text, nullable=True)
    fav_color = db.Column(db.String(10), nullable=True, default='000000') # 用的是RGB标识
    cited = db.Column(db.Text, nullable=True)

    def __init__(self, username, password, email, active_ip, password_hash, confirmed=False, kookies='00000000',
                  admin=False, avatar=0, fav_color='000000', cited=''):
        self.username = username
        self.password = password
        self.email = email
        self.create_time = datetime.datetime.now()
        self.active_time = datetime.datetime.now()
        self.active_ip = active_ip
        self.password_hash = password_hash
        self.password_cmp = md5(password)
        self.kookies = kookies
        self.admin = admin
        self.confirmed = confirmed
        self.avatar = avatar
        self.oldkookies = '#'
        self.fav_color = fav_color
        self.cited = cited
```

图 4.4 user 类定义

User 类用于处理数据库中 user 表的数据，建立对应对象，便于应用处理。

4.3.3 posts 类

```
class posts(db.Model):
    __tablename__ = 'posts'
    id = db.Column(db.Integer, primary_key=True, unique=True) # 主键，标识
    poster = db.Column(db.String(10), nullable=True) # 发布者的kookie
    poster_ip = db.Column(db.String(50), nullable=True)
    head = db.Column(db.Boolean, default=False) # 是否是第一条
    next = db.Column(db.Integer, nullable=True, default=0) # 下一条
    post_time = db.Column(db.DateTime, nullable=True) # 生成时间
    title = db.Column(db.Text, nullable=True) # 标题
    content = db.Column(db.Text, nullable=True) # 内容
    section = db.Column(db.String(40), nullable=True, default='main') # 板块
    avatar = db.Column(db.Integer, default=0)
    withpic = db.Column(db.Boolean, default=False)
    pic_route = db.Column(db.String(100), nullable=True)
    ider = db.Column(db.String(256), nullable=True) # 唯一识别符，用来反查主键
    replies = db.Column(db.Integer, nullable=True) # head=True的post的回复数，回复+=1
    topped = db.Column(db.Boolean, default=False)
    update_time = db.Column(db.DateTime, nullable=True) # 最后更新时间，由reply控制更新
    no_show = db.Column(db.Boolean, nullable=True) # 是否展示在时间线上

    def __init__(self, poster, poster_ip, head, next, title, content, section='main', withpic=False, pic_route='null',
                  replies=0, topped=False, update_time=datetime.datetime.now(), no_show=False):
        self.poster = poster
        self.poster_ip = poster_ip
        self.head = head
        self.next = next
        self.avatar = User.query.filter(User.kookies == poster).first().avatar
        self.post_time = datetime.datetime.now()
        self.title = title
        self.content = content
        self.section = section
        self.withpic = withpic
        self.pic_route = pic_route
        self.ider = md5(content + str(int(time.time()))).hexdigest()
        self.replies = replies
        self.topped = topped
        self.update_time = datetime.datetime.now()
        self.no_show = no_show
```

图 4.4 posts 类定义

posts 类用于处理数据库中 post 表的数据，建立对应对象，便于应用处理。

4.4 控件

4.4.1 发送验证邮件 (mail_sender)

```
import smtplib
from email.mime.text import MIMEText

def sender(msg_to, send_info):
    msg_from = 'stug_iii@foxmail.com' # 发送方邮箱
    passwd = 'hklvpugmdyaoiebb' # 填入发送方邮箱的授权码
    msg_to = msg_to # 收件人邮

    subject = "账号注册确认邮件"
    content = '以下是您的确认链接: \n' + send_info + '\n请点击此链接以确认激活账号\n如果不是本人操作, 请忽略本邮件\n'
    # content = render_template('mail/confirmation.html', send_info=send_info)
    msg = MIMEText(content)
    msg['Subject'] = subject
    msg['From'] = msg_from
    msg['To'] = msg_to
    try:
        s = smtplib.SMTP_SSL("smtp.qq.com", 465) # 邮件服务器及端口号
        s.login(msg_from, passwd)
        s.sendmail(msg_from, msg_to, msg.as_string())
        print("发送成功")
    except Exception as e:
        print(e)
    finally:
        s.quit()
```

图 4.5 发送激活邮件模块

该模块实现了激活邮件的发送。通过腾讯邮箱提供的 POP3 服务端口与 smtplib 提供的发送邮件功能，将验证注册链接发送至注册者的邮箱。

4.4.2 返回错误信息 (errors.py)

```
from flask import render_template

def return404(errormsg='404 error'):
    return render_template('error/error.html', errormsg=errormsg)

def return500(errormsg='500 error'):
    return render_template('error/error.html', errormsg=errormsg)
```

图 4.6 用户操作错误提示模块

该模块实现返回特定的错误页面的功能。

4.4.3 生成表单(forms.py)

```
from wtforms import SubmitField, SelectField, StringField, TextAreaField
from wtforms.validators import DataRequired, Email, length
from flask_wtf import FlaskForm

class loginform(FlaskForm):
    name = StringField('账号', validators=[DataRequired()])
    password = StringField('密码', validators=[DataRequired()])

class signinform(FlaskForm):
    name = StringField('用户名', validators=[DataRequired()])
    password1 = StringField('密码', validators=[DataRequired(), length(min=6, max=32)])
    password2 = StringField('确认密码', validators=[DataRequired()])
    email = StringField('确认邮箱', validators=[DataRequired(), Email()])

class new_post_form(FlaskForm):
    title = StringField('标题', validators=[DataRequired()])
    content = TextAreaField('内容', validators=[DataRequired()])
    section = SelectField('分区', validators=[DataRequired()], choices=[('综合', '综合'), ('校内', '校内'), ('形势与政策', '形势与政策'), ('问答', '问答'), ('深夜', '深夜'), ('其他', '其他')])
    no_show = SelectField('展示在时间线上', validators=[DataRequired()], choices=[('1', '是'), ('0', '否')], default='1')
    submit = SubmitField('提交')

class comment_form(FlaskForm):
    content = TextAreaField('内容', validators=[DataRequired()])
    submit = SubmitField('提交')
```

图 4.7 生成表单对象模块

该模块实现了各种提交给 View 层函数的表单对象，包括了：登陆表单、注册表单、发新串表单、评论表单。

4.4.4 获得新发言饼干 (cookiemaker.py)

```
from enc import *
import hashlib

def cookie(username):
    username = hashlib.md5(bytes(username, encoding='utf-8'))
    username.update(bytes(str(time.time()), encoding='utf-8'))
    username = username.hexdigest()
    username = md5(username)[0:8].upper()
    return username
```

图 4.8 生成表单对象模块

该模块实现了用户新饼干的生成，并且返回。此处新饼干是用户原本饼干+二进制的服务器时间的哈希值转换为 16 进制取前八位再全大写。安全性基本得到保障。

5 测试

本项目是面向多端适配的，移动端测试与主机端测试方式相同，将移动端的测试放到 5.4 中。

5.1 测试环境

5.1.1 服务端

测试环境为 Pycharm - Flask 开发环境，所需环境包 `venv` 在文件目录/`venv` 中，所用包在附录 A 中。

5.1.2 客户端

测试环境为 Google Chrome 版本 76.0.3809.132（正式版本）（64 位）。移动端使用浏览器自带模拟移动设备功能。

5.2 页面浏览测试

5.2.1 未登录状态浏览

(1) 主页



图 5.1 未登录-主页

可以看到此时登入主页时，显示串预览内容正常，导航栏提示登录/注册。

(2) 查看串详情

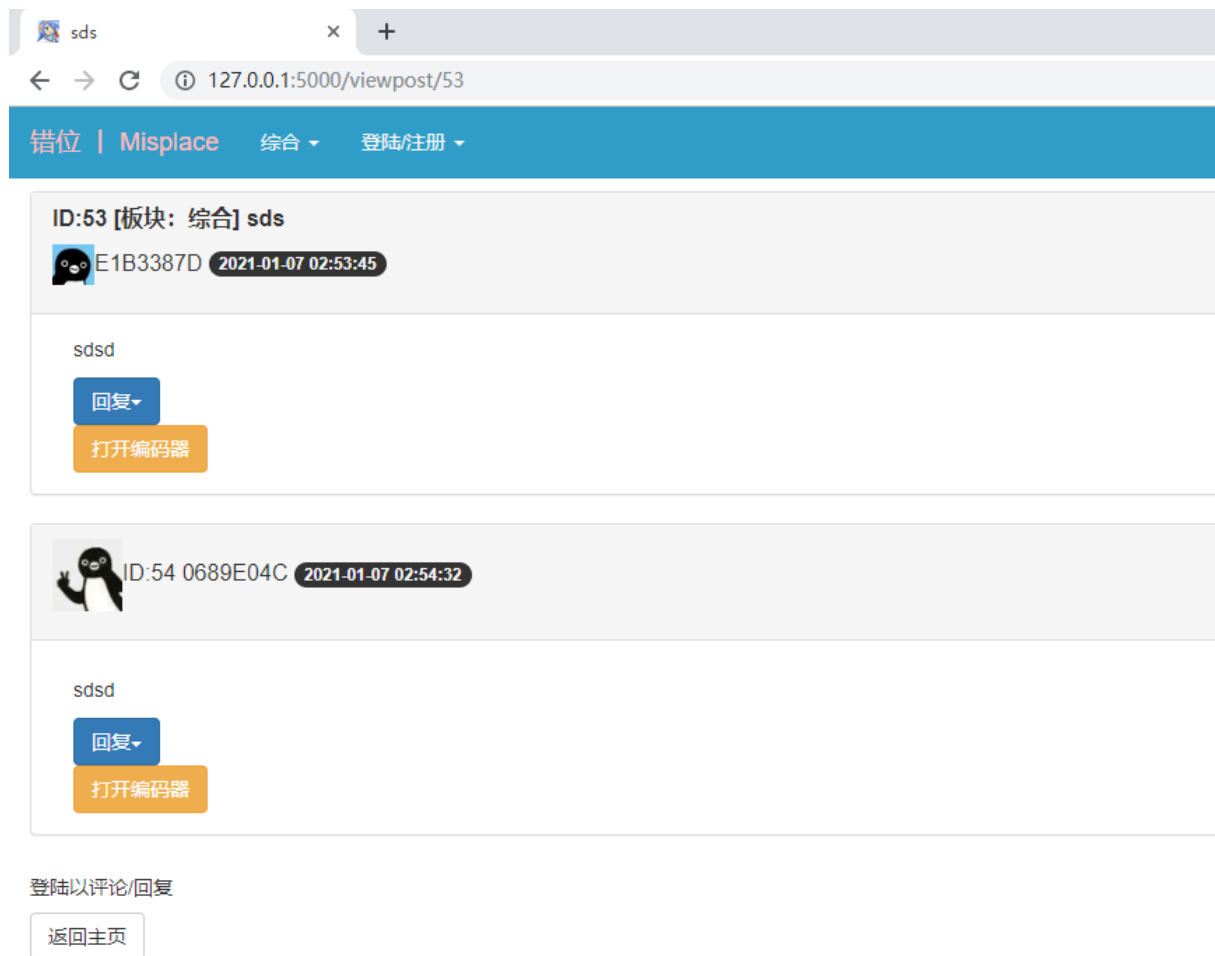


图 5.2 未登录-串详情

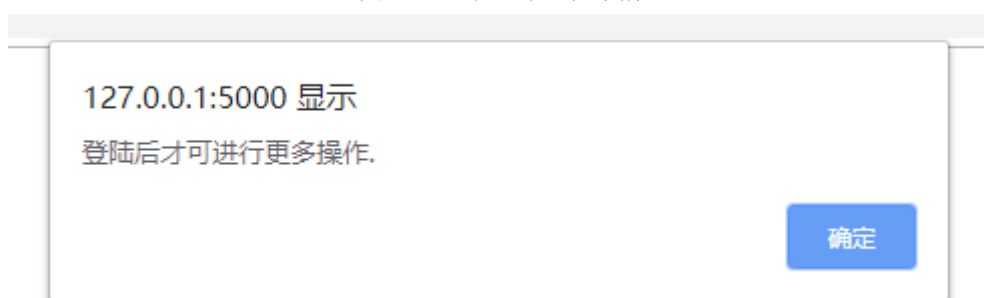


图 5.3 未登录-串详情-提交评论

此时看到串内容可以正常浏览，但是不可以进行回复。

5.2.2 已登录状态浏览

(1) 主页

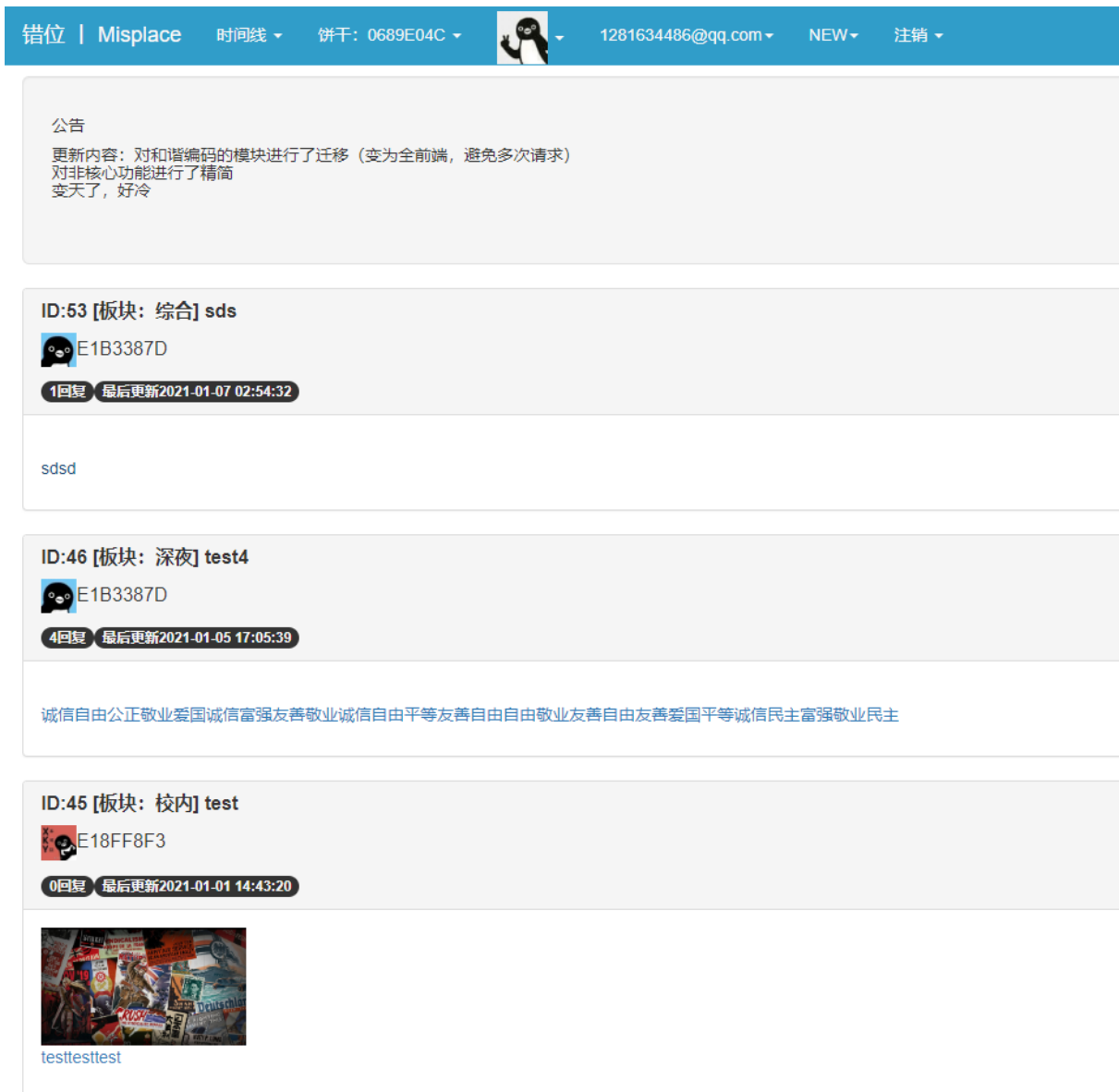


图 5.4 已登录-主页

可以看到内容显示正常, 且导航栏可以进行各种用户操作。

(2) 查看串详情

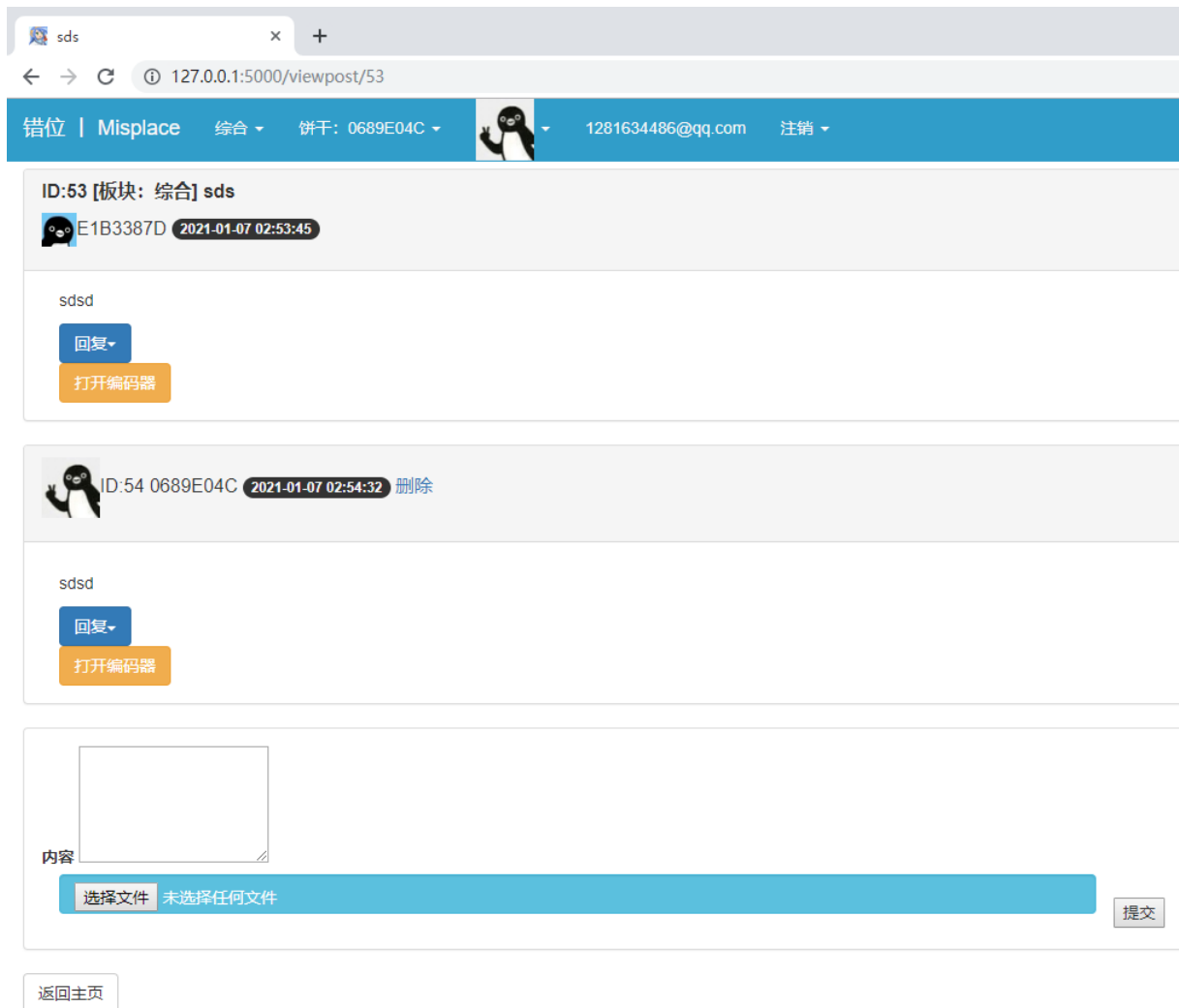


图 5.5 已登录-串详情

可以看到，登陆情况下，可以正常对主串或者评论的子串进行回复。

5.3 用户操作测试

5.3.1 注册

点击主页导航栏登陆/注册，在注册一栏输入用户名，密码，验证密码，注册邮箱进行注册。

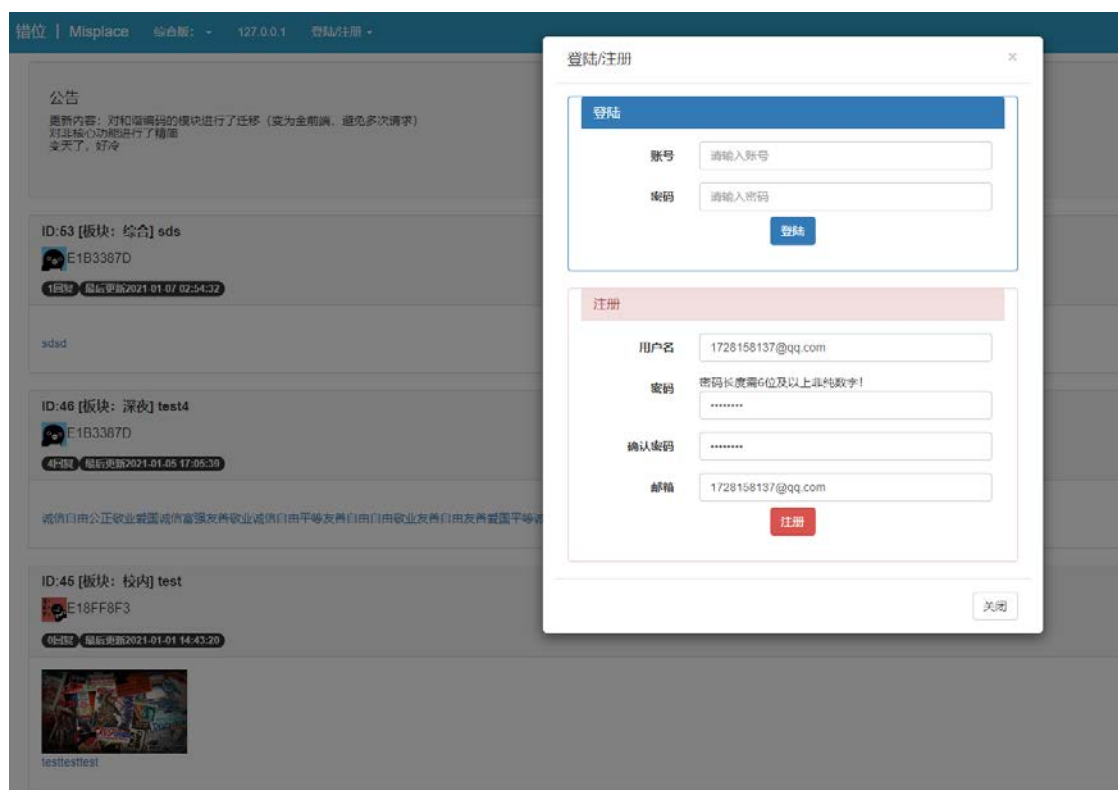


图 5.6 注册界面

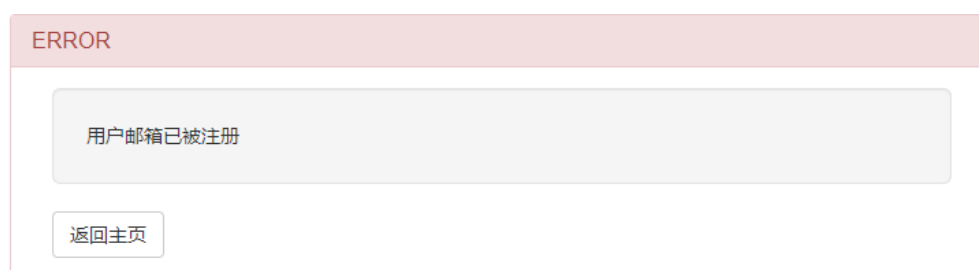


图 5.6a 注册失败-邮箱占用

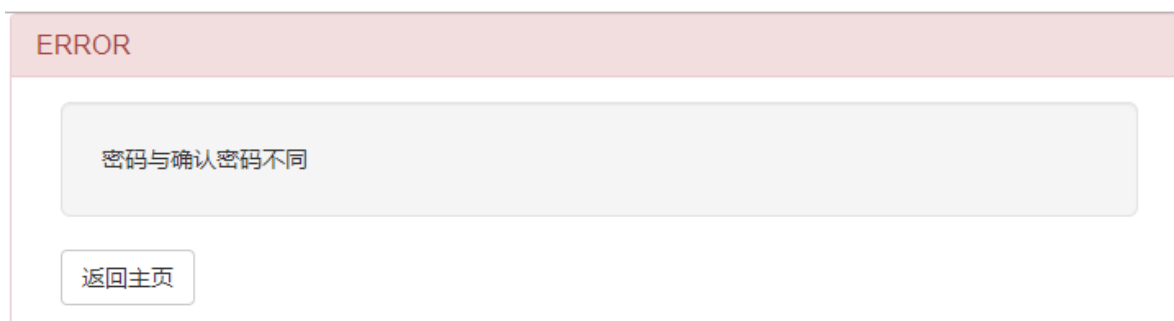


图 5.6a 注册失败-两次密码不同



图 5.6c 注册成功返回界面

可以看到，两次密码输入不同，以及邮箱已经被注册都将导致注册失败。而用户将收到激活邮件如图 5.6d:

以下是您的确认链接:
localhost:5000/api/confirmation?code=783d4387ea6dfd1a17a37773ec993d91
请点击此链接以确认激活账号
如果不是本人操作, 请忽略本邮件

图 5.6d 确认注册邮件

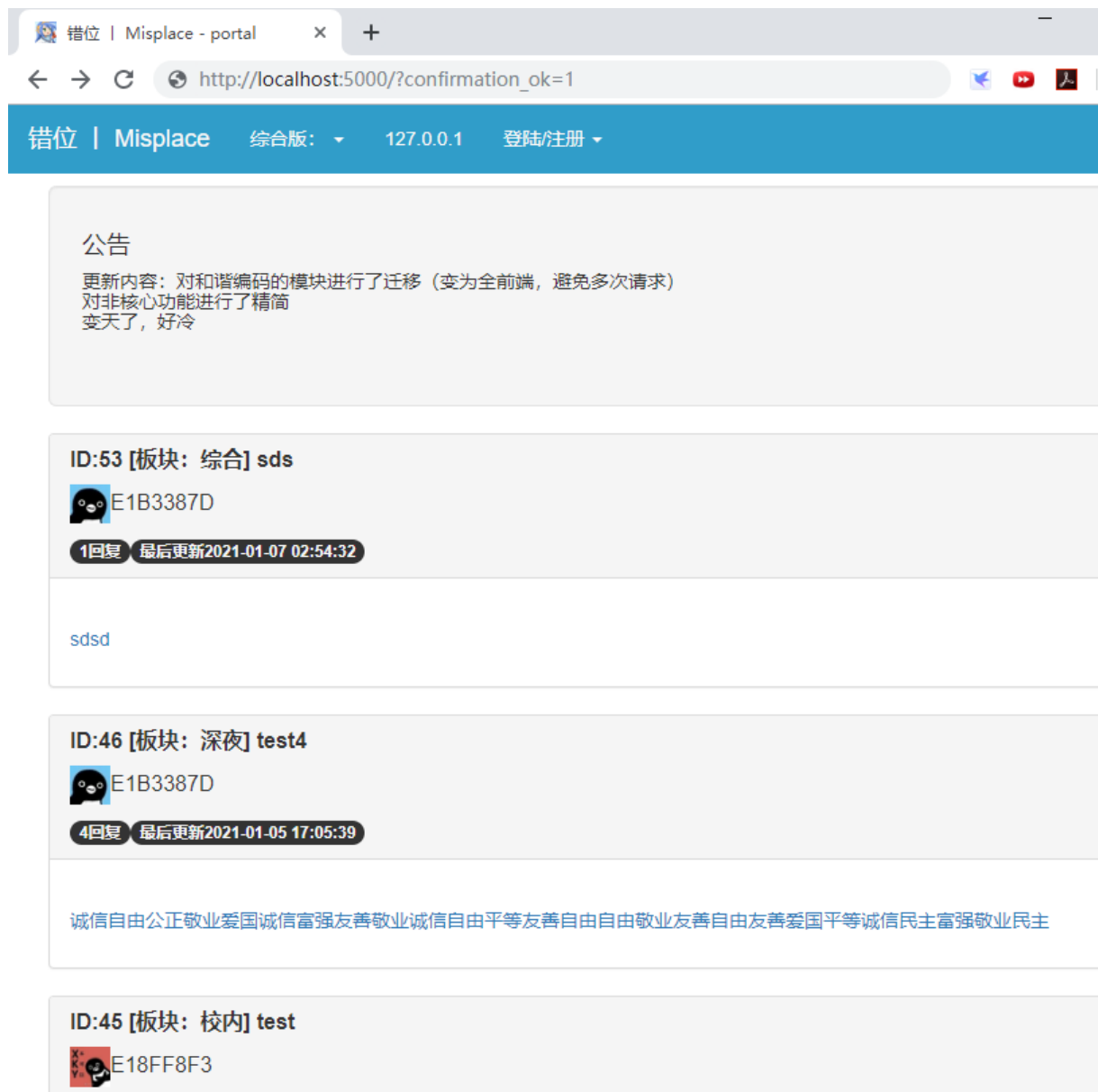


图 5.6e 确认注册成功

验证注册成功后, 用户登陆后即可进行正常操作。

5.3.2 登陆/注销



图 5.7 登陆

点击主页导航栏登陆/注册，在登陆一栏输入用户名，密码进行登陆。成功则为图 5.4 所示界面，失败则为图 5.1 所示界面。

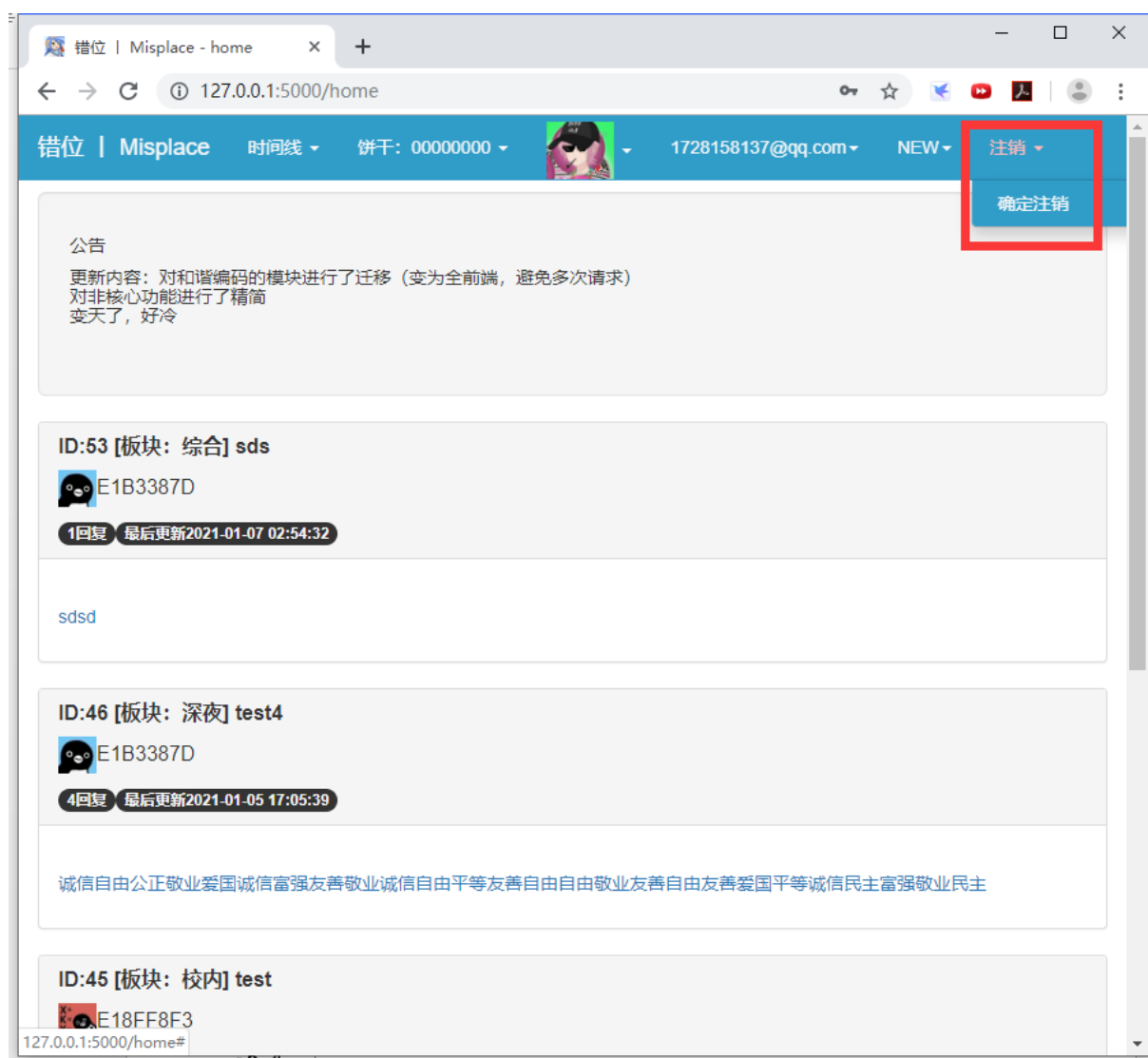


图 5.8 注销

点击注销-确定注销，注销成功后回到图 5.1 所示界面。

5.3.3 获取新饼干



图 5.8a 获取新饼干



图 5.8b 新饼干

5.3.4 发表新串



图 5.9a 点击 NEW

图 5.9b 发表新串-填写内容

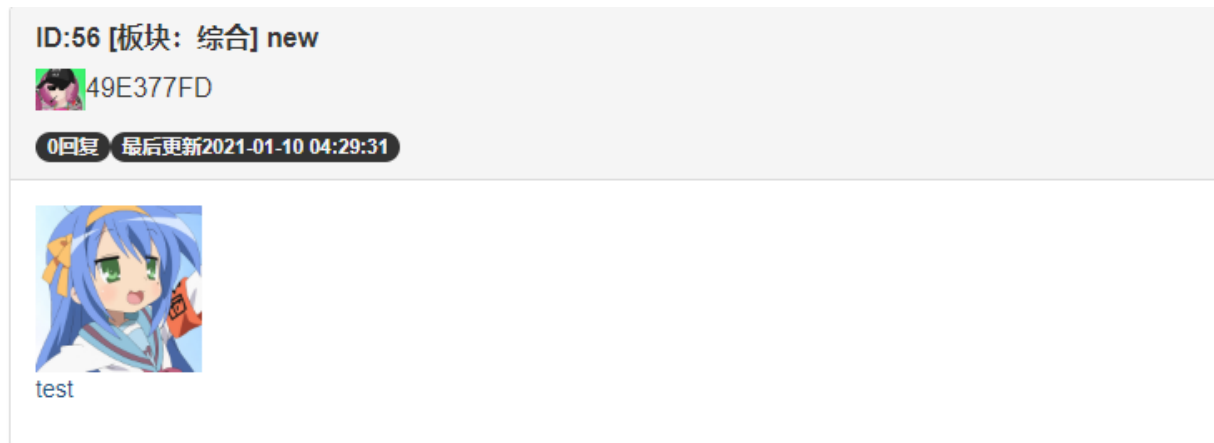


图 5.9c 发表成功

5.3.5 删除串

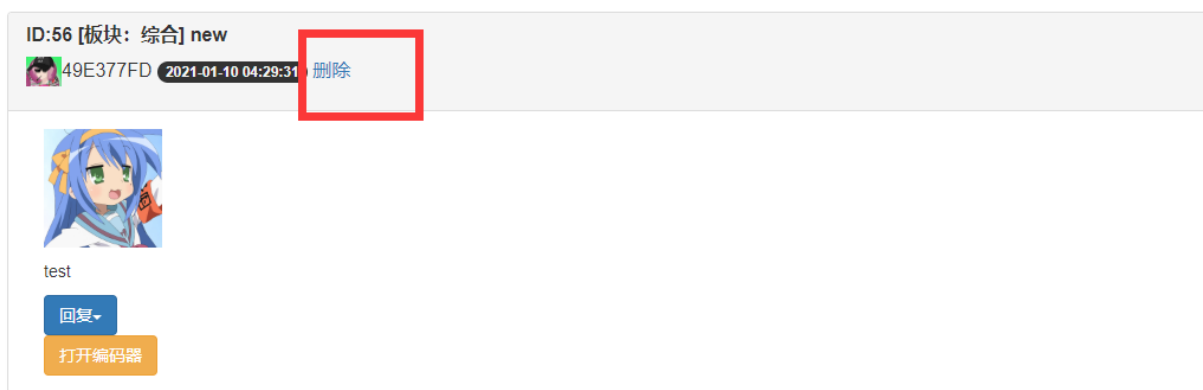




图 5.10 删除串

删除成功后即返回 5.4 所示的已登录界面。

5.3.6 评论

[错位](#) | [Misplace](#) [综合](#) ▾ 饼干: 49E377FD ▾  ▾ [1728158137@qq.com](#) [注销](#) ▾


ID:53 [板块: 综合] sds

 E1B3387D 2021-01-07 02:53:45

sdsd

[回复](#) ▾

[打开编码器](#)

 ID:54 0689E04C 2021-01-07 02:54:32

sdsd

[回复](#) ▾

[打开编码器](#)

comment!

内容

[选择文件](#) 808fb7bfc205ba0018d81f02.jpg

[提交](#)

[返回主页](#)

图 5.11a 发表评论-填写内容

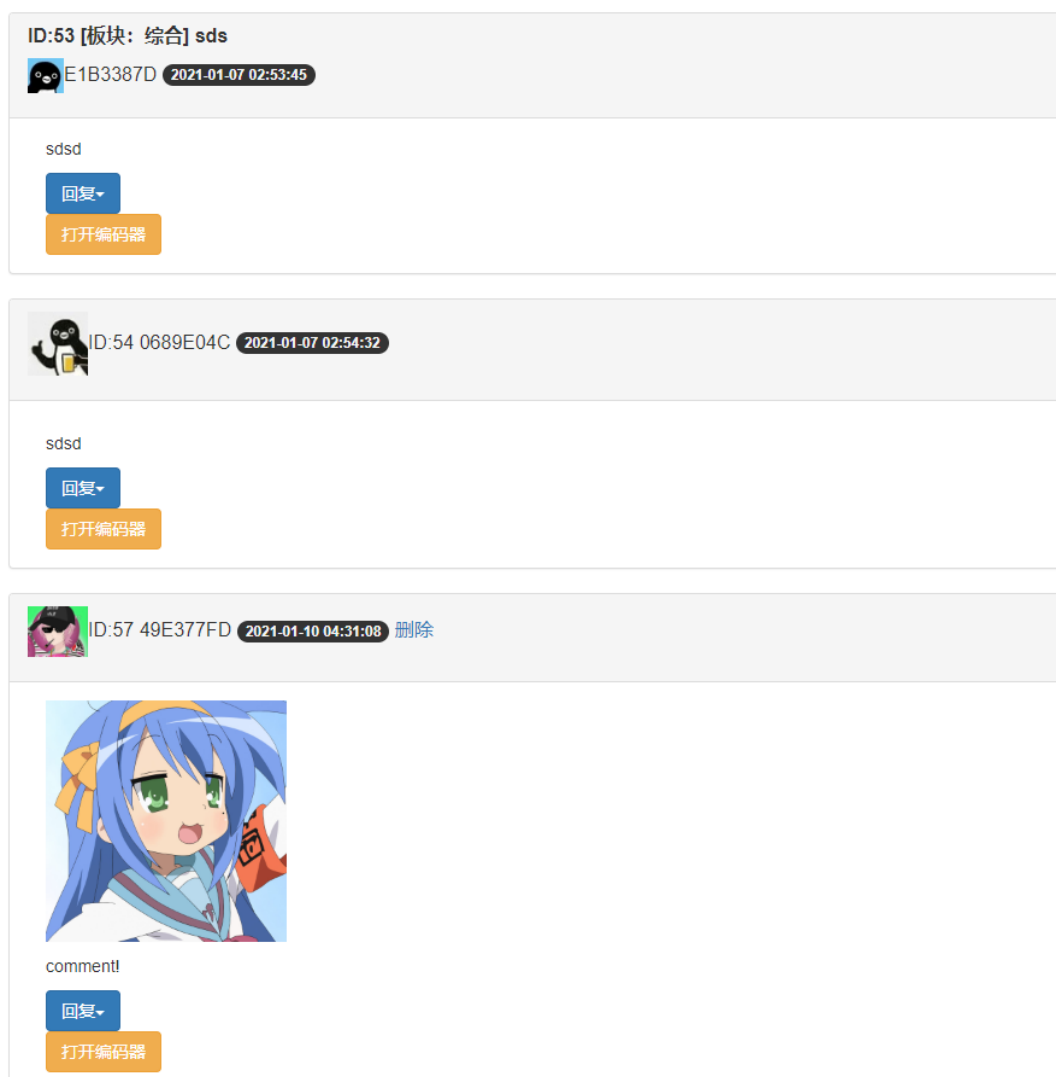


图 5.11b 发表评论-成功

5.3.7 编码/解码

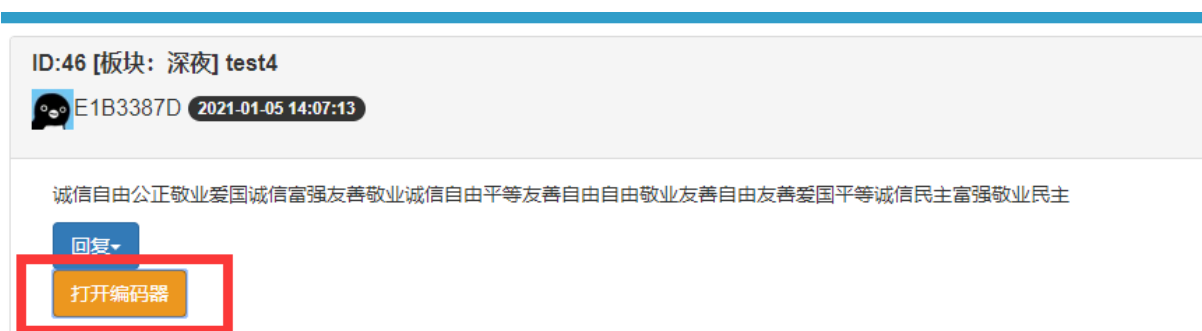


图 5.12a 编码器（关闭状态）



图 5.11 编码器（打开使用状态）

在打开状态下，用户将已经编码的信息输入“编码”框，选择 **Decode** 即可进行信息的解码。将欲编码的信息输入“解码”框，选择 **Encode** 即可进行明文的编码。

5.4 移动端适配测试

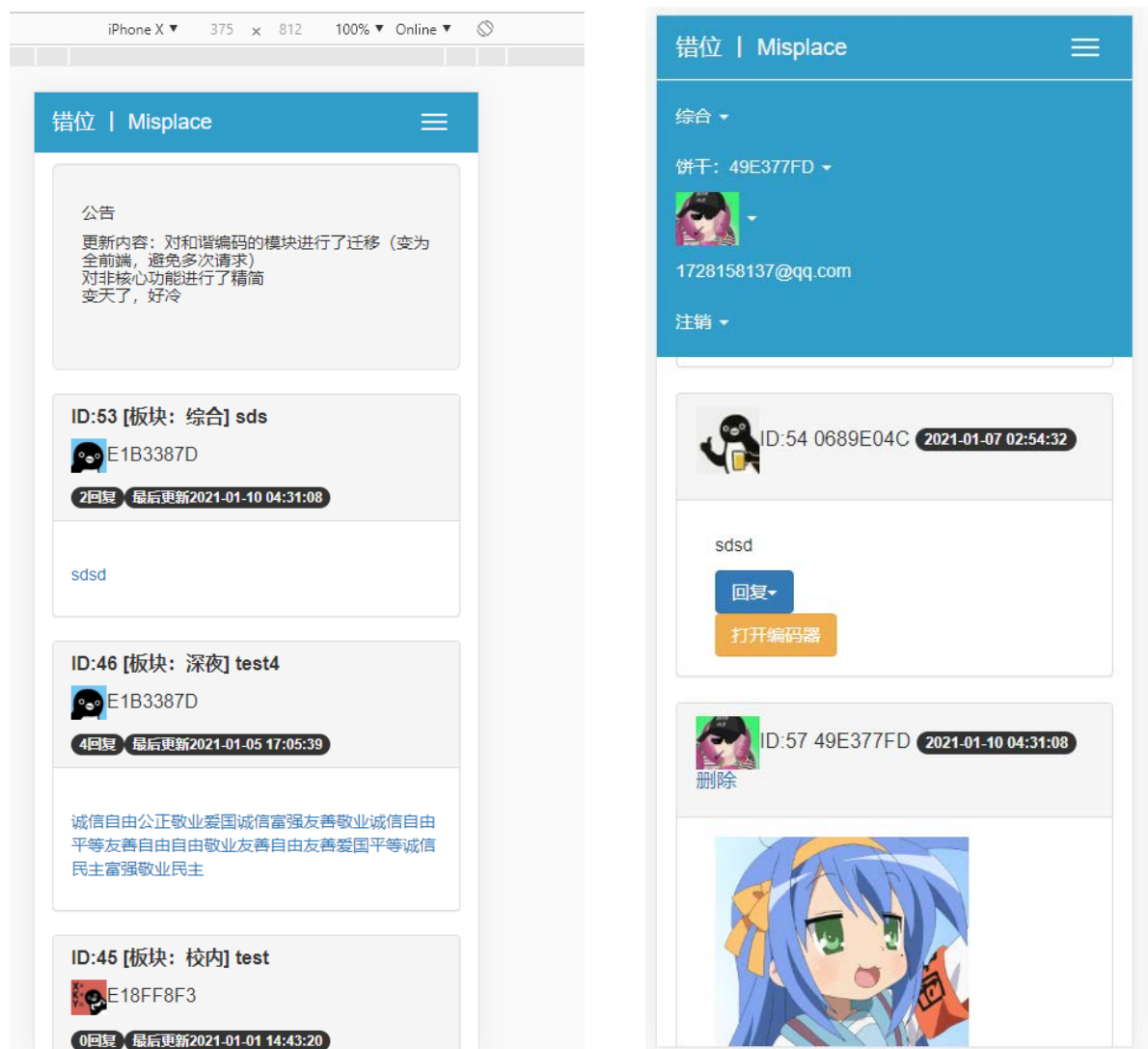


图 5.12 移动端浏览器浏览页面

显然 Bootstrap 的网格系统对移动端的适配效果是令人舒适的。

6 项目分析

6.1 优点

6.1.1 预定功能基本实现

开发一个功能类似 A 岛匿名版的匿名论坛，且在基本功能开发完成后改进匿名机制和管理方法。并且通过对发言内容的编码解码，使发言内容有一定的匿名性（不执行操作则无法读取到明文）。同时利用 Bootstrap 的网格系统进行 PC 和移动设备的显示适配。

6.1.2 新功能的开发

- （1）使用小饼干机制(kookies)实现发帖人的匿名
- （2）对内容的编码解码，实现关键词的隐藏

6.2 缺点以及改进设想

- （1）前后端登陆的表单未进行验证，待增加验证登陆时专门的密码验证加密方式，前端向后端发起 ajax 请求，后端返回 salt，前端加密提交数据与后端比对。
- （2）对可能的攻击未进行防范，比如发出请求的参数未进行过滤等。
- （3）前后端的分离程度不够大，在功能复杂后应该采用新的技术进行分离，如 Vue 作为前端服务，实现前后端的进一步分离。

设计总结

本项目目的在于使用 **Flask+Bootstrap** 等技术模仿实现 A 岛匿名版的基本功能，并实现在其基础上的改进。从目前来看，要求基本完成。用户的注册登陆、查看串、评论串等基本功能已经实现。提升目标也基本达成，收获颇丰。

通过本次项目的设计与实现，帮助我们提升了使用 **Git** 工具的能力与团队合作编程的能力。同时也对 **Web** 开发有了初步的认识（毕竟 **Flask** 还只是个轻量级框架）。

参 考 文 献

- [1] Flask 官方文档中文版翻译.
<http://www.pythondoc.com/flask/index.html>.
- [2] Flask-WTF 官方文档中文版翻译.
<http://www.pythondoc.com/flask-wtf/index.html>.
- [3] SQLAlchemy 1.4 Documentation.
<https://www.osgeo.cn/sqlalchemy/>.
- [4] sym233/core-values-encoder.
<https://github.com/sym233/core-values-encoder>

附录 A 开发环境所需运行库

certifi==2020.12.5
chardet==4.0.0
click==7.1.2
docopt==0.6.2
Flask==1.1.2
Flask-SQLAlchemy==2.4.4
Flask-WTF==0.14.2
idna==2.10
itsdangerous==1.1.0
Jinja2==2.11.2
MarkupSafe==1.1.1
numpy==1.19.4
pandas==1.2.0
protobuf==3.14.0
PyMySQL==0.9.3
python-dateutil==2.8.1
pytz==2020.5
requests==2.25.1
six==1.15.0
SQLAlchemy==1.3.3
urllib3==1.26.2
Werkzeug==0.15.4
WTForms==2.2.1
yarg==0.1.9

成员贡献及感想

(1) 201873030 赵书彬 (组长)

工作占比: 33.34%

主要工作内容: Flask 主要程序编写、渲染页面模板编写、项目后期完善

感想: 通过本次项目, 我体会到了合作开发的不易, 以及 Git 等版本控制工具对合作开发项目的重要性, 以后我将重视这方面的自我提高。同时, 能在课程中回归 Web 开发这一个人兴趣也算是十分幸运, 极大提升了我对 web 开发的理解, 同时使我的兴趣更加浓厚了。

(2) 201892075 XXX (组员)

工作占比: 33.33%

主要工作内容: 静态页面编写、Model 层编写

感想: xxxxxxxx

(3) 201892075 XXX (组员)

工作占比: 33.33%

主要工作内容: 文档组织、样式文件编写

感想: xxxxxxxx