九章算法
www.jiuzhang.com

通知与图像上传

授课人　令狐东邪

版权声明

课程大纲

今天需要做什么

01

django-notifications-hq 的安装与使用

02

实现未读和标记已读 API

03

更新通知状态

04

定义用户信息模型并配置 Admin 界面

05

实现上传用户头像和更新昵称 API

课程大纲

06

定义推文图片模型

07

推文支持图片

00 前情回顾

前情回顾

在上周的课程中，我们实现了**两个核心模块**：

- Comment 模块
- Like 模块

前情回顾

接下来我们要学习**新的内容**：

1. 用 django-notification 增加评论点赞关注的消息提醒

- 如何引入第三方开源代码，在后序维护中需要注意什么问题
- 基于第三方开源项目自定制 API
- 使用 Service 类包装复杂逻辑

2. 图像上传

- 增加 UserProfile Model 实现头像，昵称等信息的存储
- django 的 listener 机制（什么时候用，什么时候不用）
- OneToOneField
- Tweet 图像上传

01

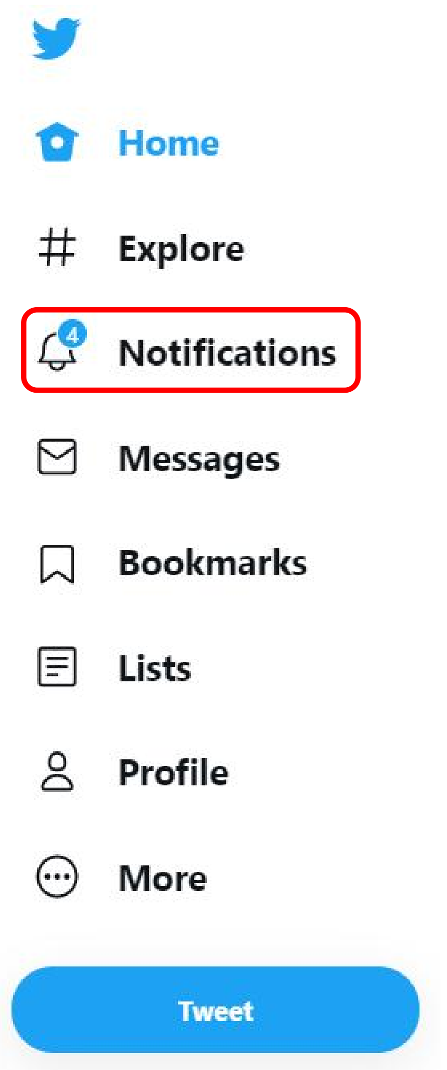
django-notifications-hq 的安装与使用

Notifications 的安装与使用

情景设计

让我们设想一下这样的场景，当我们成为推特的用户，每天你都会发推文来分享你的日常生活，也经常能收到你粉丝的评论。每次收到粉丝的评论，你都会和他们在评论区里互动。但是随着推文数量的增加，你很难知道在哪篇推文下有粉丝给你评论，因为把每篇推文下面的评论都浏览一遍是很麻烦的。这时候我们就需要一个通知功能。

通知功能是一个非常有用和常见的功能。在我们发了一篇推文后，当有人评论了你的推文，这时 Twitter 就会给你发一条通知，这样就可以知道有谁对你的某篇推文进行了评论。



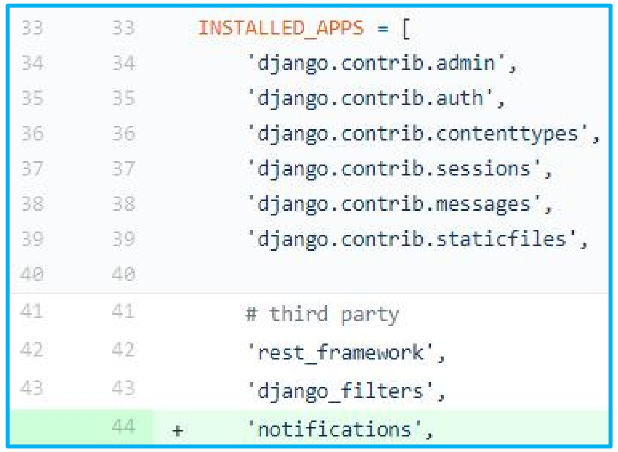
Notifications 的安装与使用

django-notifications-hq

官方文档：https://pypi.org/project/django-notifications-hq/

1 根据 requirements.txt 安装 django-notifications-hp

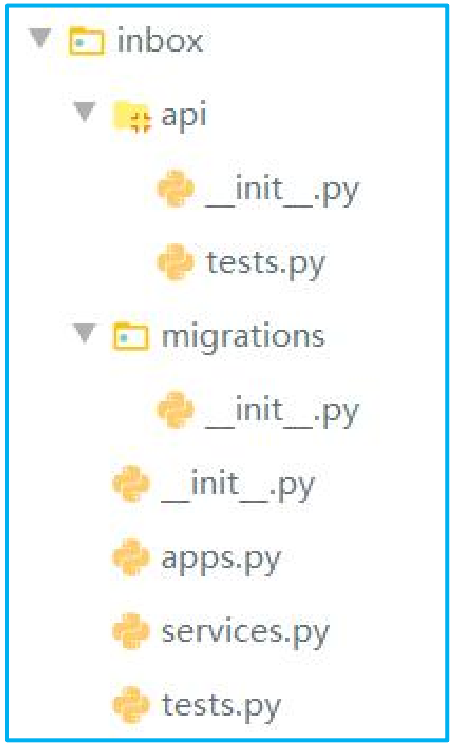
2 配置 django-notifications-hp



```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # third party
    'rest_framework',
    'django_filters',
+   'notifications',
```

Notifications 的安装与使用

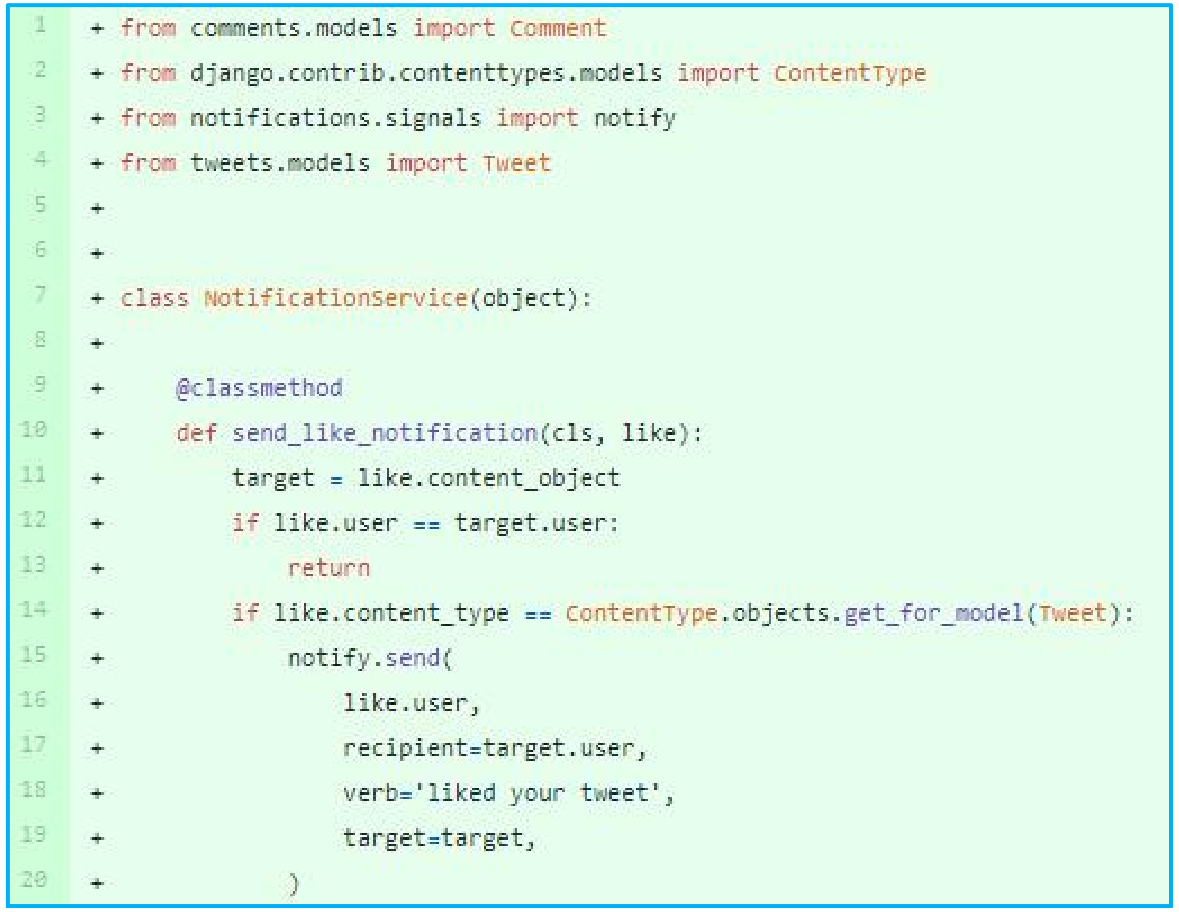
创建 inbox 目录


```
▼ inbox
  ▼ api
      __init__.py
      tests.py
  ▼ migrations
      __init__.py
    __init__.py
    apps.py
    services.py
    tests.py
```

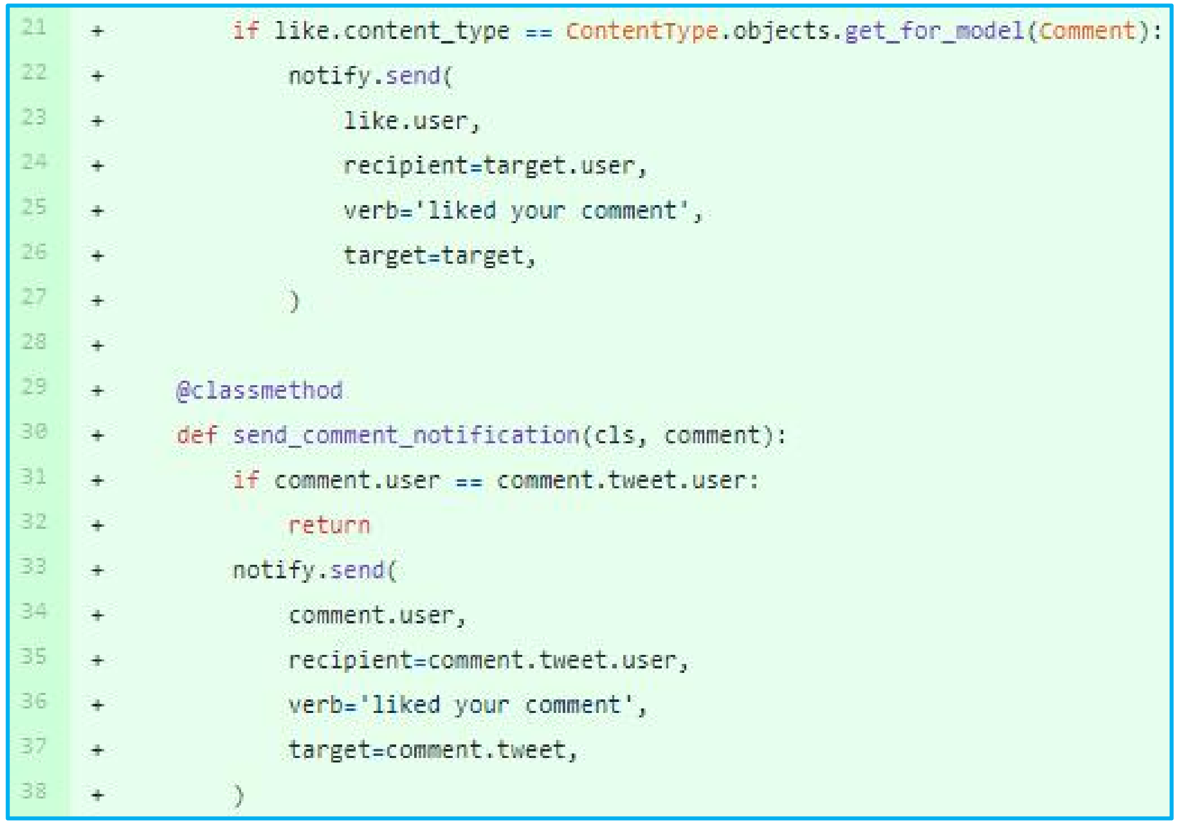
为了方便我们后面操作，我们可以现在项目的根目录下创建如右图所示的目录结构。

Notifications 的安装与使用

创建通知服务



```
+ from comments.models import Comment
+ from django.contrib.contenttypes.models import ContentType
+ from notifications.signals import notify
+ from tweets.models import Tweet
+
+
+ class NotificationService(object):
+
+     @classmethod
+     def send_like_notification(cls, like):
+         target = like.content_object
+         if like.user == target.user:
+             return
+         if like.content_type == ContentType.objects.get_for_model(Tweet):
+             notify.send(
+                 like.user,
+                 recipient=target.user,
+                 verb='liked your tweet',
+                 target=target,
+             )
```

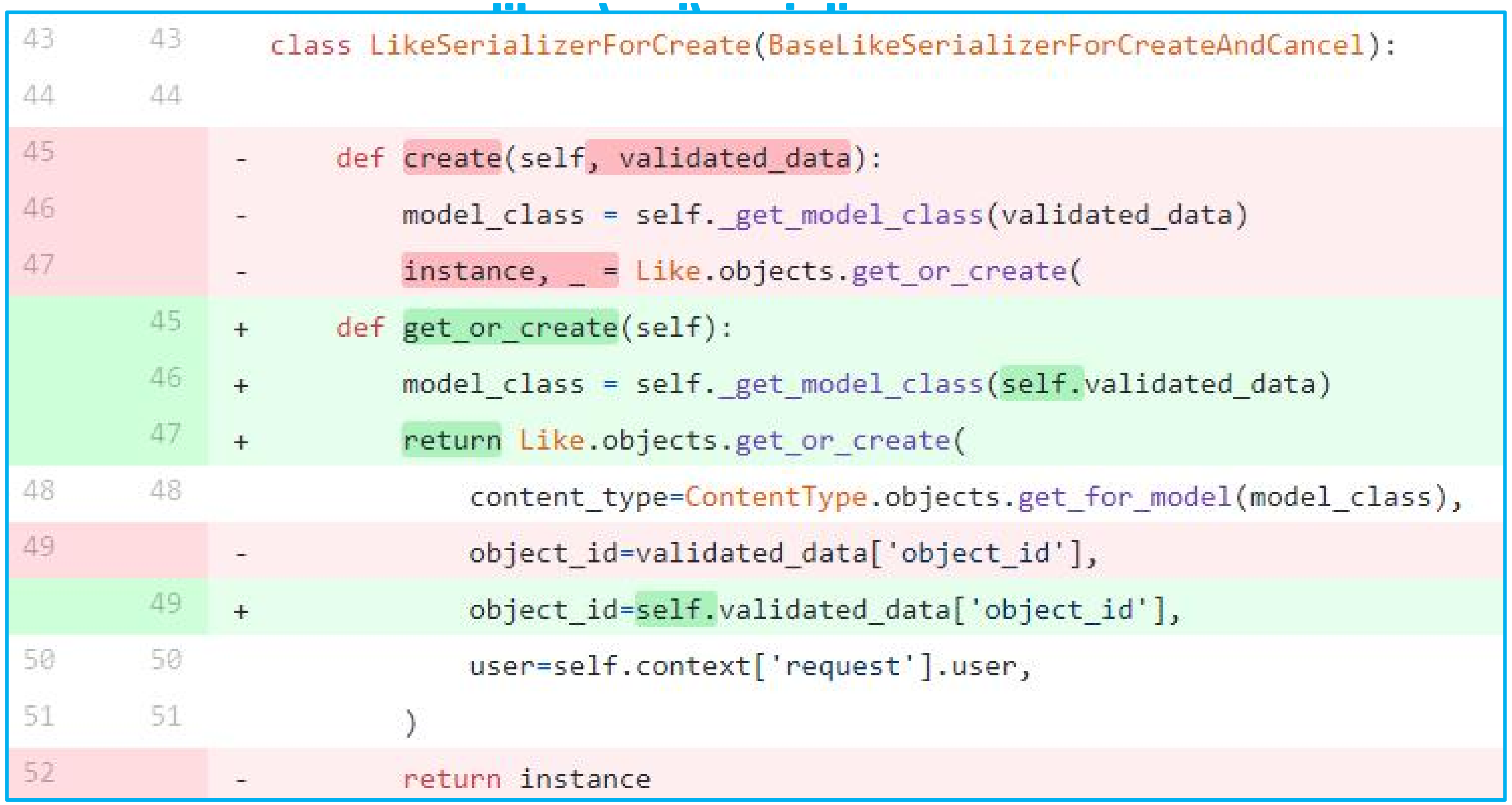


```
+         if like.content_type == ContentType.objects.get_for_model(Comment):
+             notify.send(
+                 like.user,
+                 recipient=target.user,
+                 verb='liked your comment',
+                 target=target,
+             )
+
+     @classmethod
+     def send_comment_notification(cls, comment):
+         if comment.user == comment.tweet.user:
+             return
+         notify.send(
+             comment.user,
+             recipient=comment.tweet.user,
+             verb='liked your comment',
+             target=comment.tweet,
+         )
```

Notifications 的安装与使用

修改点赞的序列化器

路径为



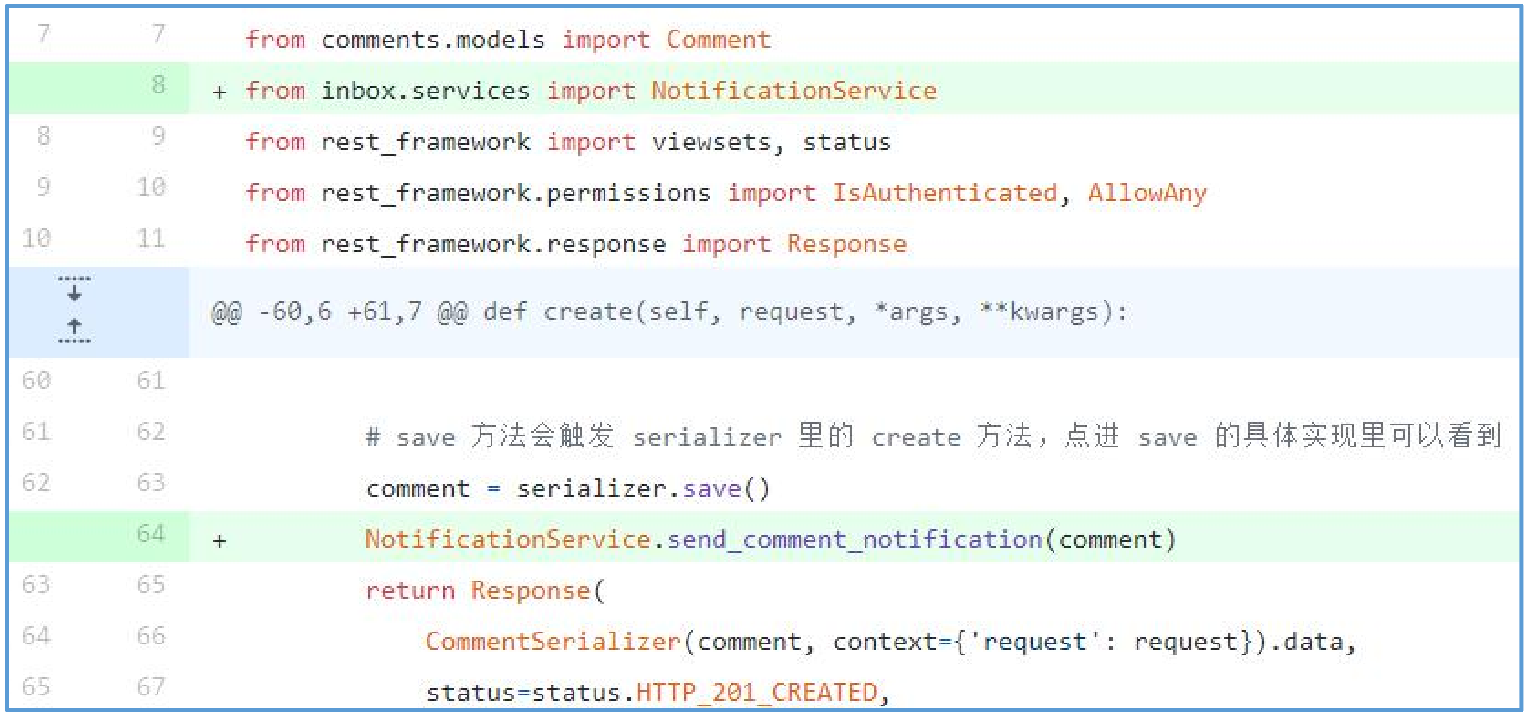
```diff
  class LikeSerializerForCreate(BaseLikeSerializerForCreateAndCancel):

-     def create(self, validated_data):
-         model_class = self._get_model_class(validated_data)
-         instance, _ = Like.objects.get_or_create(
+     def get_or_create(self):
+         model_class = self._get_model_class(self.validated_data)
+         return Like.objects.get_or_create(
              content_type=ContentType.objects.get_for_model(model_class),
-             object_id=validated_data['object_id'],
+             object_id=self.validated_data['object_id'],
              user=self.context['request'].user,
          )
-         return instance
```

Notifications 的安装与使用

九章算法
www.jiuzhang.com

修改评论视图

路径为 **comments\api\views.py**



```
   from comments.models import Comment
 + from inbox.services import NotificationService
   from rest_framework import viewsets, status
   from rest_framework.permissions import IsAuthenticated, AllowAny
   from rest_framework.response import Response
@@ -60,6 +61,7 @@ def create(self, request, *args, **kwargs):

           # save 方法会触发 serializer 里的 create 方法，点进 save 的具体实现里可以看到
           comment = serializer.save()
 +         NotificationService.send_comment_notification(comment)
           return Response(
               CommentSerializer(comment, context={'request': request}).data,
               status=status.HTTP_201_CREATED,
```

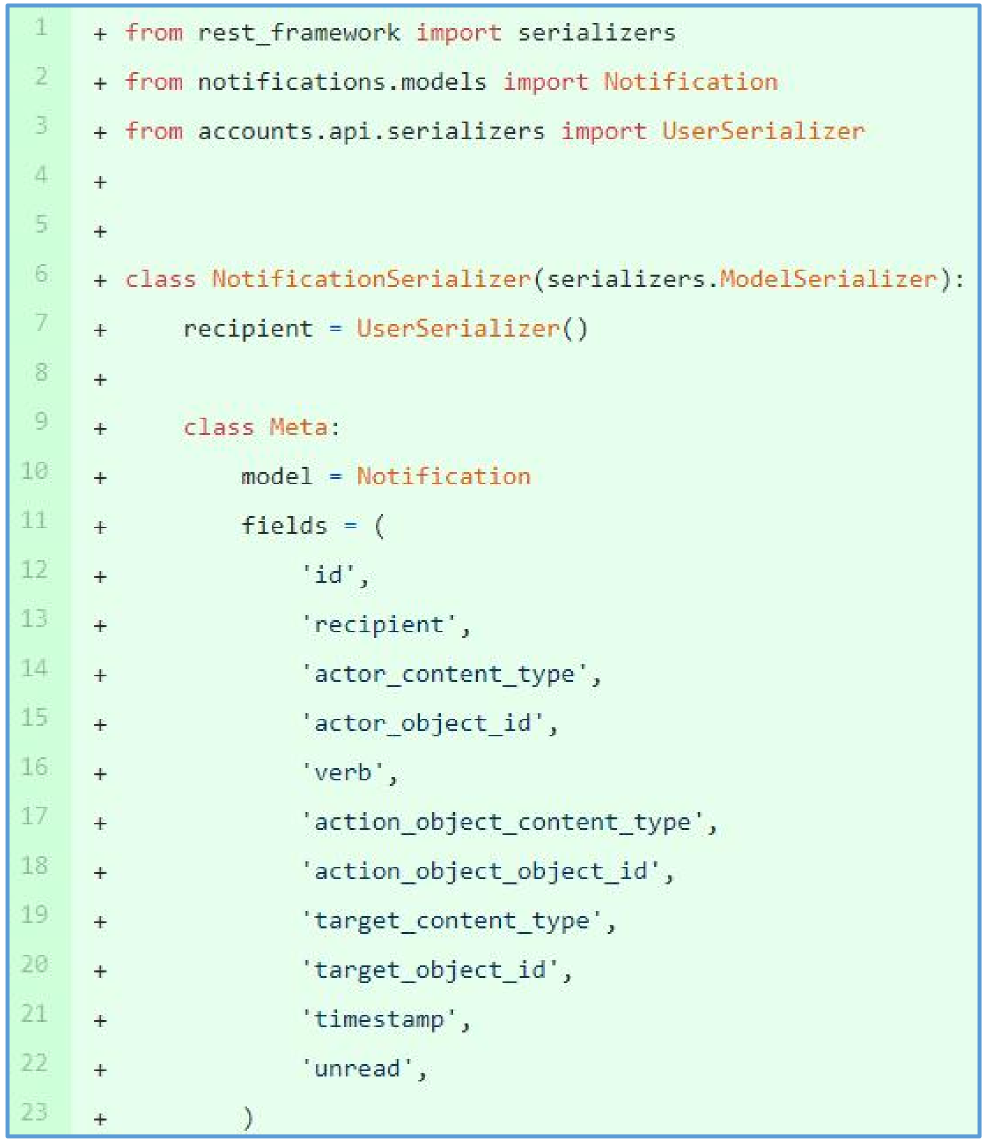
02 实现未读和标记已读 API

实现未读和标记已读 API

创建序列化器

路径为

inbox\api\serializers.py

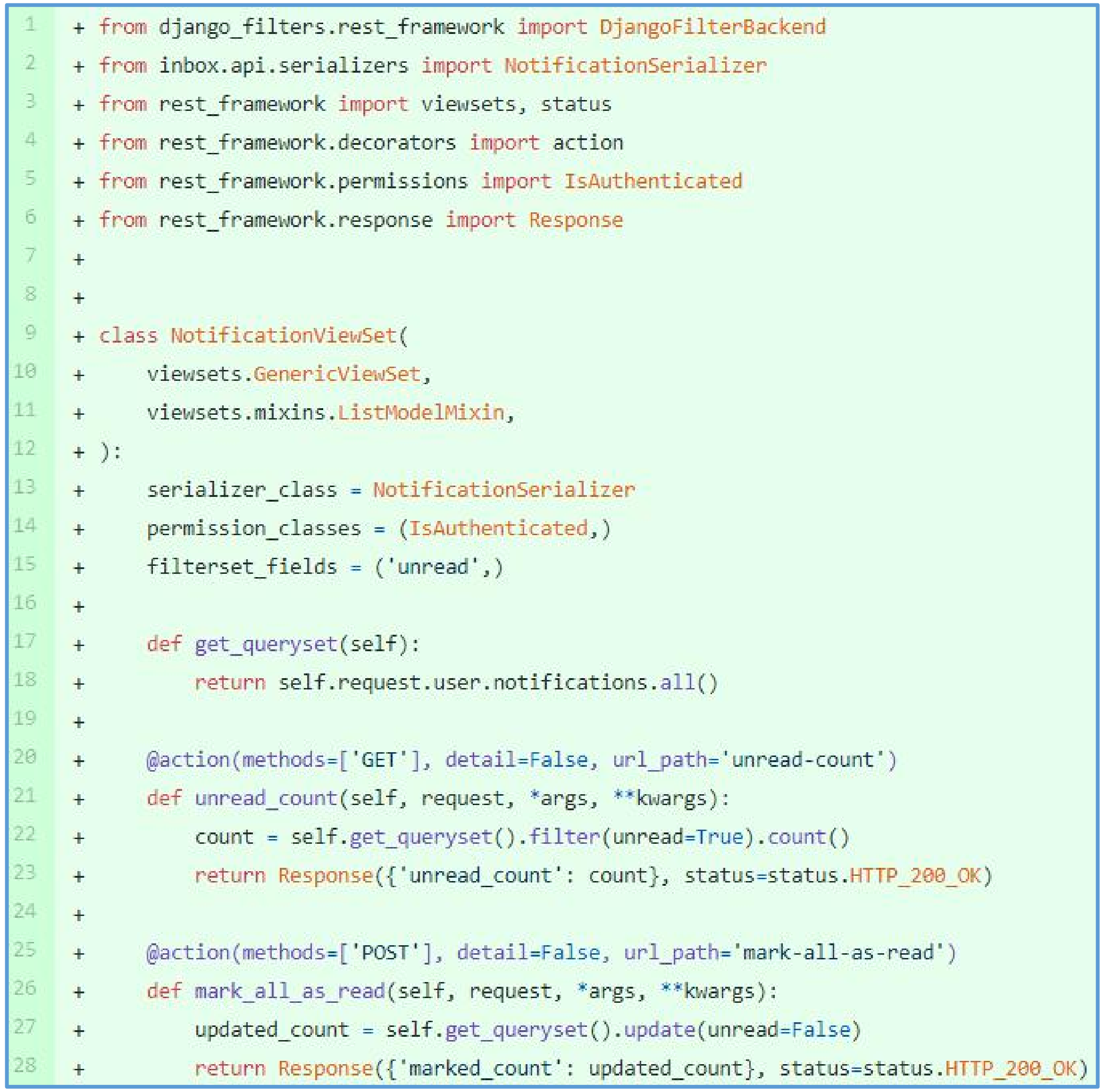


```
+ from rest_framework import serializers
+ from notifications.models import Notification
+ from accounts.api.serializers import UserSerializer
+
+
+ class NotificationSerializer(serializers.ModelSerializer):
+     recipient = UserSerializer()
+
+     class Meta:
+         model = Notification
+         fields = (
+             'id',
+             'recipient',
+             'actor_content_type',
+             'actor_object_id',
+             'verb',
+             'action_object_content_type',
+             'action_object_object_id',
+             'target_content_type',
+             'target_object_id',
+             'timestamp',
+             'unread',
+         )
```

实现未读和标记已读 API

创建视图

路径为 **inbox\api\views.py**

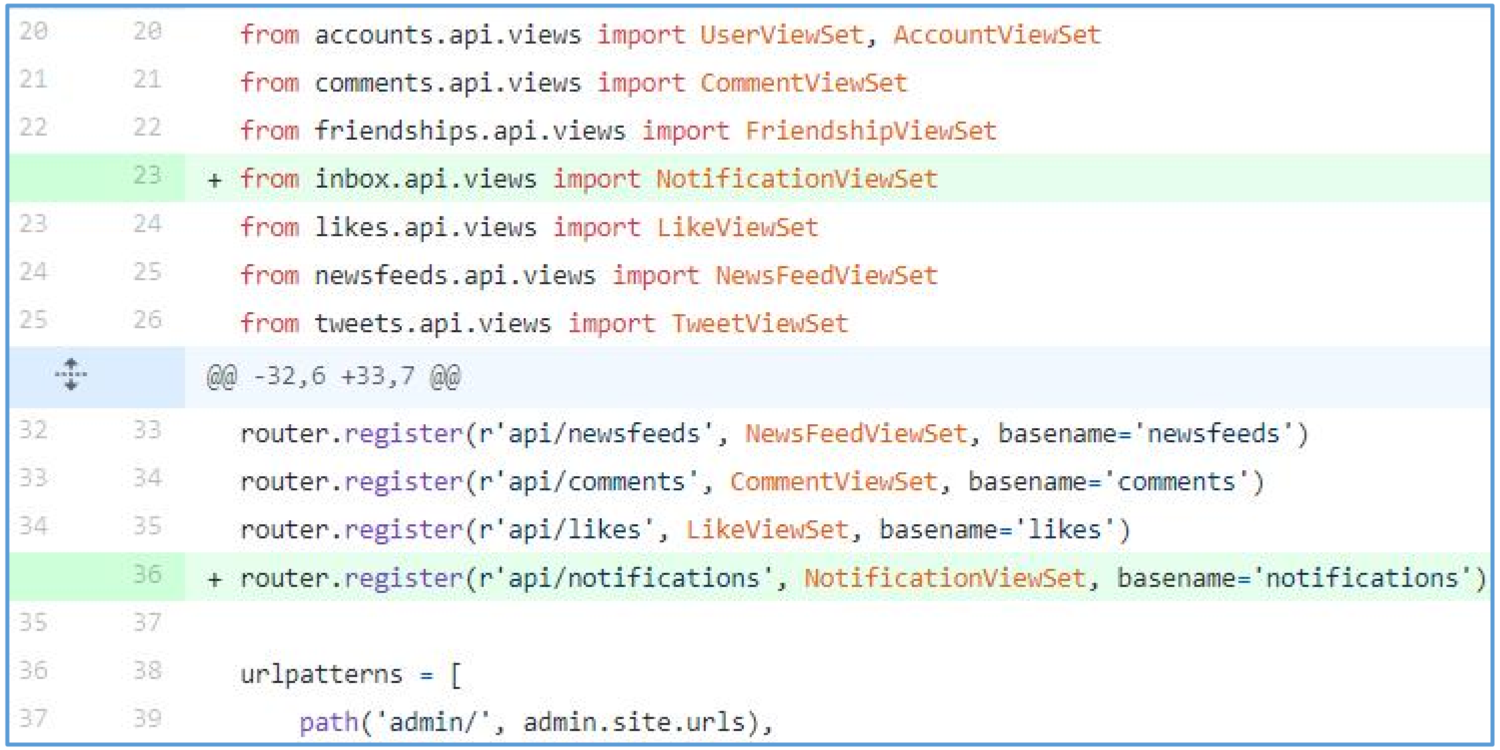


```
+ from django_filters.rest_framework import DjangoFilterBackend
+ from inbox.api.serializers import NotificationSerializer
+ from rest_framework import viewsets, status
+ from rest_framework.decorators import action
+ from rest_framework.permissions import IsAuthenticated
+ from rest_framework.response import Response
+
+
+ class NotificationViewSet(
+     viewsets.GenericViewSet,
+     viewsets.mixins.ListModelMixin,
+ ):
+     serializer_class = NotificationSerializer
+     permission_classes = (IsAuthenticated,)
+     filterset_fields = ('unread',)
+
+     def get_queryset(self):
+         return self.request.user.notifications.all()
+
+     @action(methods=['GET'], detail=False, url_path='unread-count')
+     def unread_count(self, request, *args, **kwargs):
+         count = self.get_queryset().filter(unread=True).count()
+         return Response({'unread_count': count}, status=status.HTTP_200_OK)
+
+     @action(methods=['POST'], detail=False, url_path='mark-all-as-read')
+     def mark_all_as_read(self, request, *args, **kwargs):
+         updated_count = self.get_queryset().update(unread=False)
+         return Response({'marked_count': updated_count}, status=status.HTTP_200_OK)
```

实现未读和标记已读 API

配饰路由

路径为 **twitter\urls.py**



```
  from accounts.api.views import UserViewSet, AccountViewSet
  from comments.api.views import CommentViewSet
  from friendships.api.views import FriendshipViewSet
+ from inbox.api.views import NotificationViewSet
  from likes.api.views import LikeViewSet
  from newsfeeds.api.views import NewsFeedViewSet
  from tweets.api.views import TweetViewSet
@@ -32,6 +33,7 @@
  router.register(r'api/newsfeeds', NewsFeedViewSet, basename='newsfeeds')
  router.register(r'api/comments', CommentViewSet, basename='comments')
  router.register(r'api/likes', LikeViewSet, basename='likes')
+ router.register(r'api/notifications', NotificationViewSet, basename='notifications')

  urlpatterns = [
      path('admin/', admin.site.urls),
```

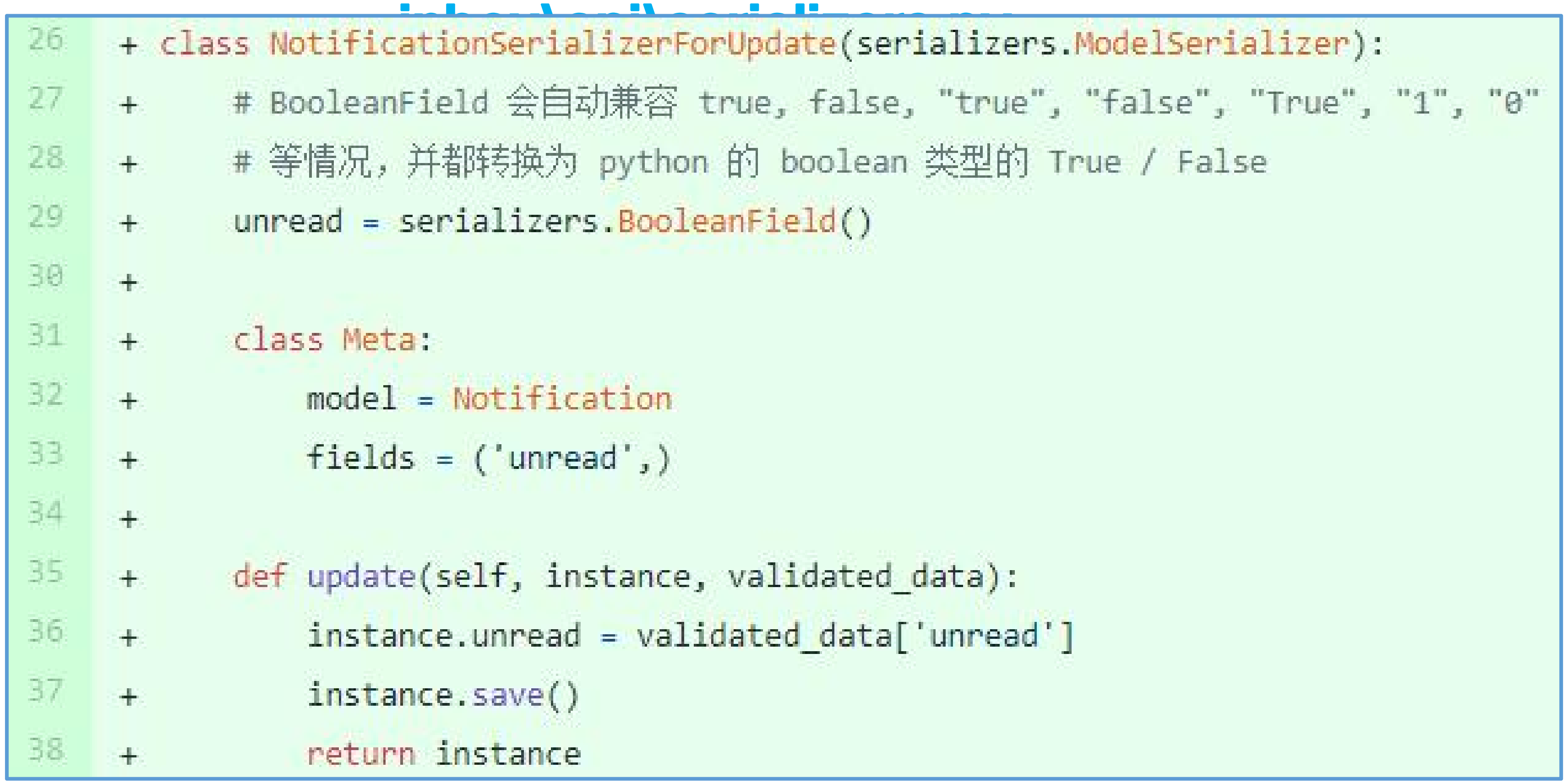
03 更新通知状态

更新通知状态

修改序列化器

路径为

inbox\api\serializers.py

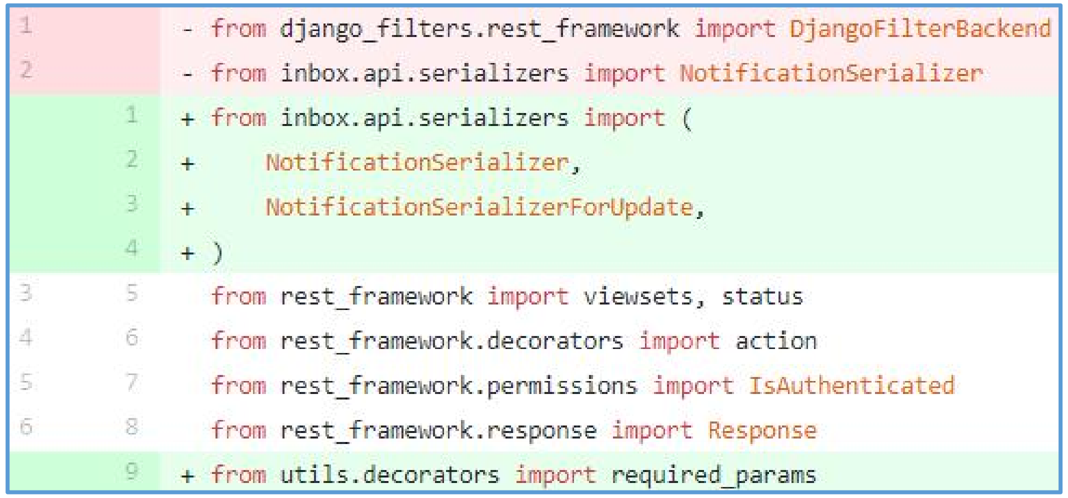


```
+ class NotificationSerializerForUpdate(serializers.ModelSerializer):
+     # BooleanField 会自动兼容 true, false, "true", "false", "True", "1", "0"
+     # 等情况，并都转换为 python 的 boolean 类型的 True / False
+     unread = serializers.BooleanField()
+
+     class Meta:
+         model = Notification
+         fields = ('unread',)
+
+     def update(self, instance, validated_data):
+         instance.unread = validated_data['unread']
+         instance.save()
+         return instance
```

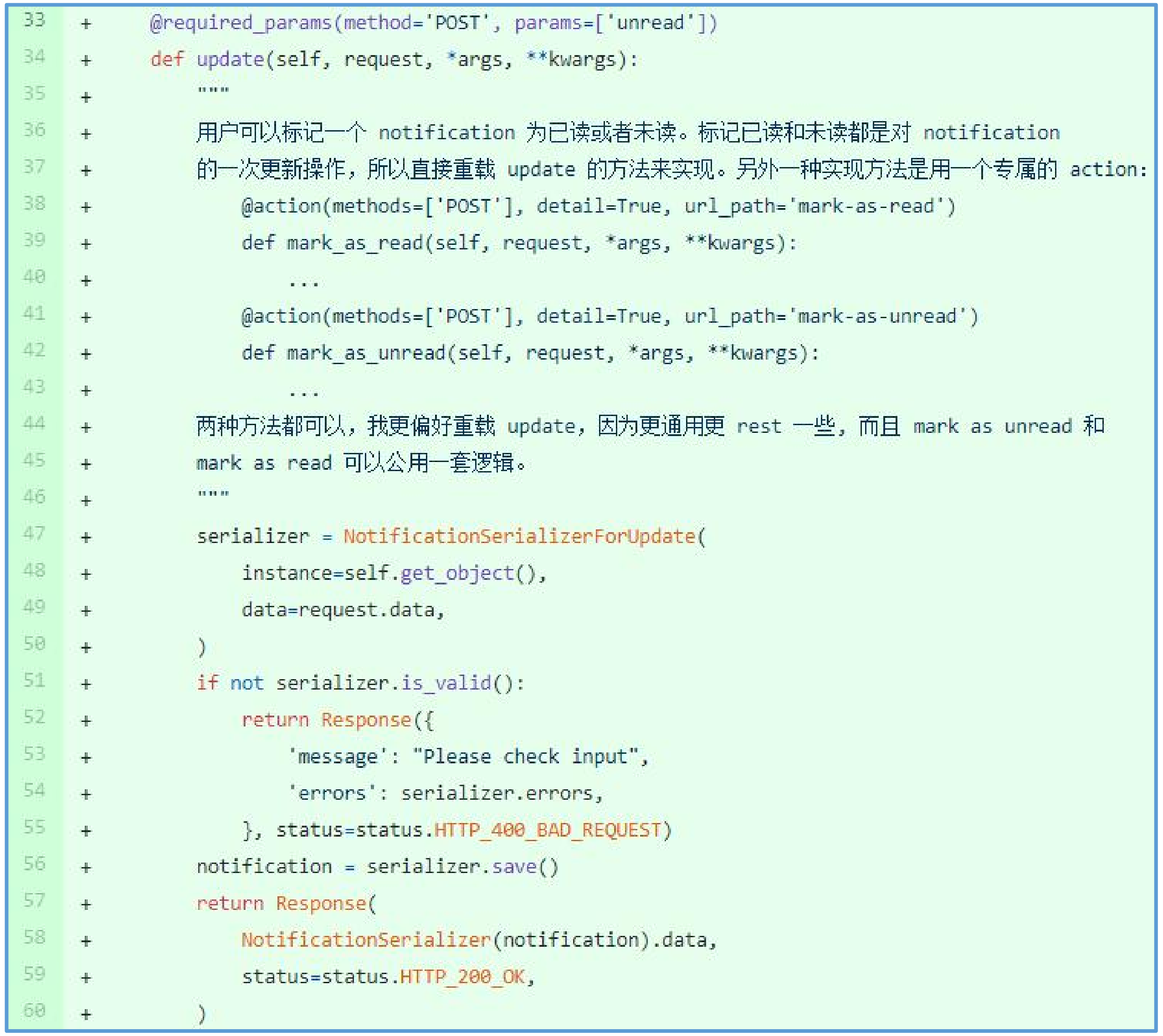
更新通知状态

修改视图

路径为 **inbox\api\views.py**



```
- from django_filters.rest_framework import DjangoFilterBackend
- from inbox.api.serializers import NotificationSerializer
+ from inbox.api.serializers import (
+     NotificationSerializer,
+     NotificationSerializerForUpdate,
+ )
  from rest_framework import viewsets, status
  from rest_framework.decorators import action
  from rest_framework.permissions import IsAuthenticated
  from rest_framework.response import Response
+ from utils.decorators import required_params
```



```
+    @required_params(method='POST', params=['unread'])
+    def update(self, request, *args, **kwargs):
+        """
+        用户可以标记一个 notification 为已读或者未读。标记已读和未读都是对 notification
+        的一次更新操作，所以直接重载 update 的方法来实现。另外一种实现方法是用一个专属的 action:
+            @action(methods=['POST'], detail=True, url_path='mark-as-read')
+            def mark_as_read(self, request, *args, **kwargs):
+                ...
+            @action(methods=['POST'], detail=True, url_path='mark-as-unread')
+            def mark_as_unread(self, request, *args, **kwargs):
+                ...
+        两种方法都可以，我更偏好重载 update，因为更通用更 rest 一些，而且 mark as unread 和
+        mark as read 可以公用一套逻辑。
+        """
+        serializer = NotificationSerializerForUpdate(
+            instance=self.get_object(),
+            data=request.data,
+        )
+        if not serializer.is_valid():
+            return Response({
+                'message': "Please check input",
+                'errors': serializer.errors,
+            }, status=status.HTTP_400_BAD_REQUEST)
+        notification = serializer.save()
+        return Response(
+            NotificationSerializer(notification).data,
+            status=status.HTTP_200_OK,
+        )
```

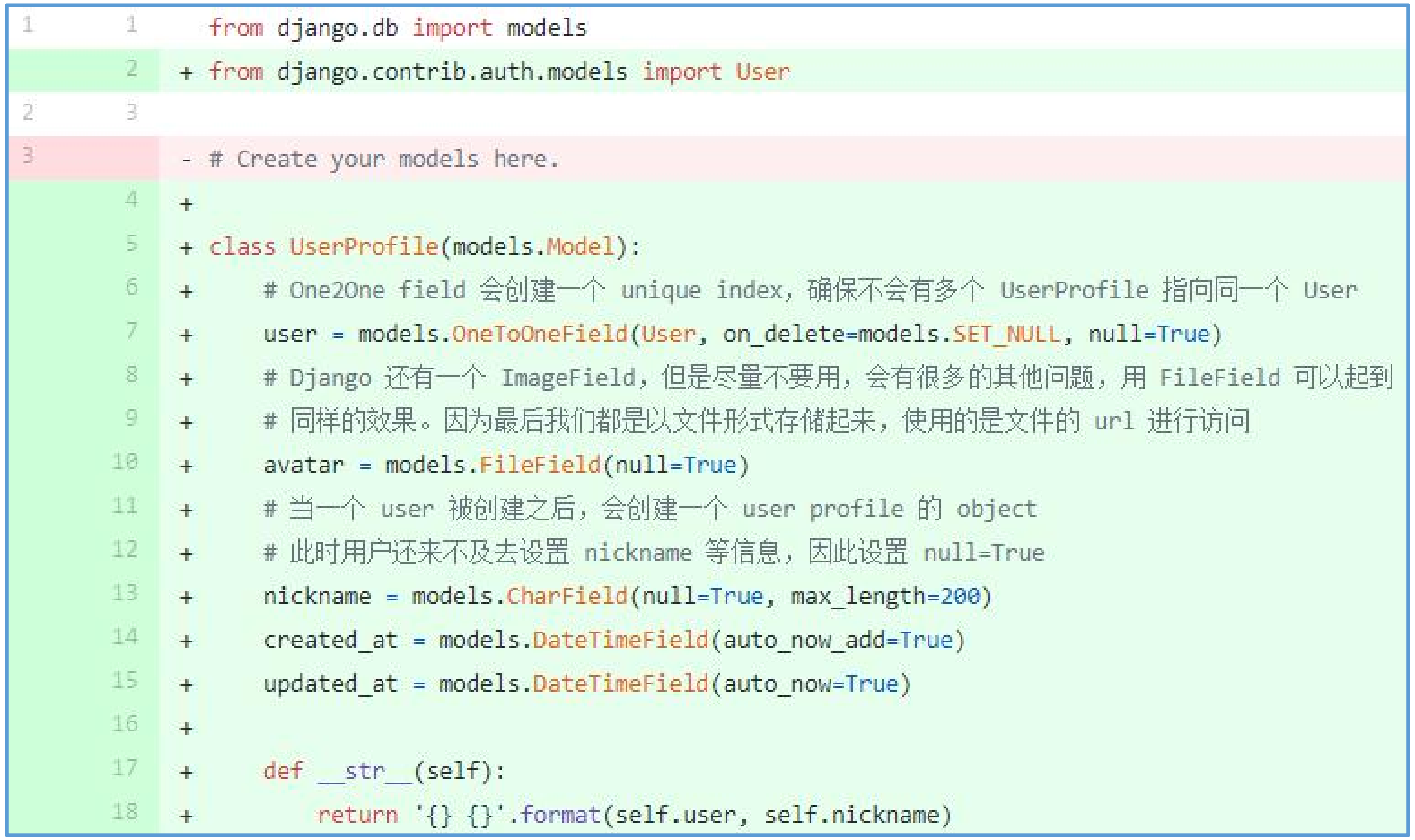
04

定义用户信息模型并配置 Admin 界面

定义用户信息模型并配置 Admin 界面

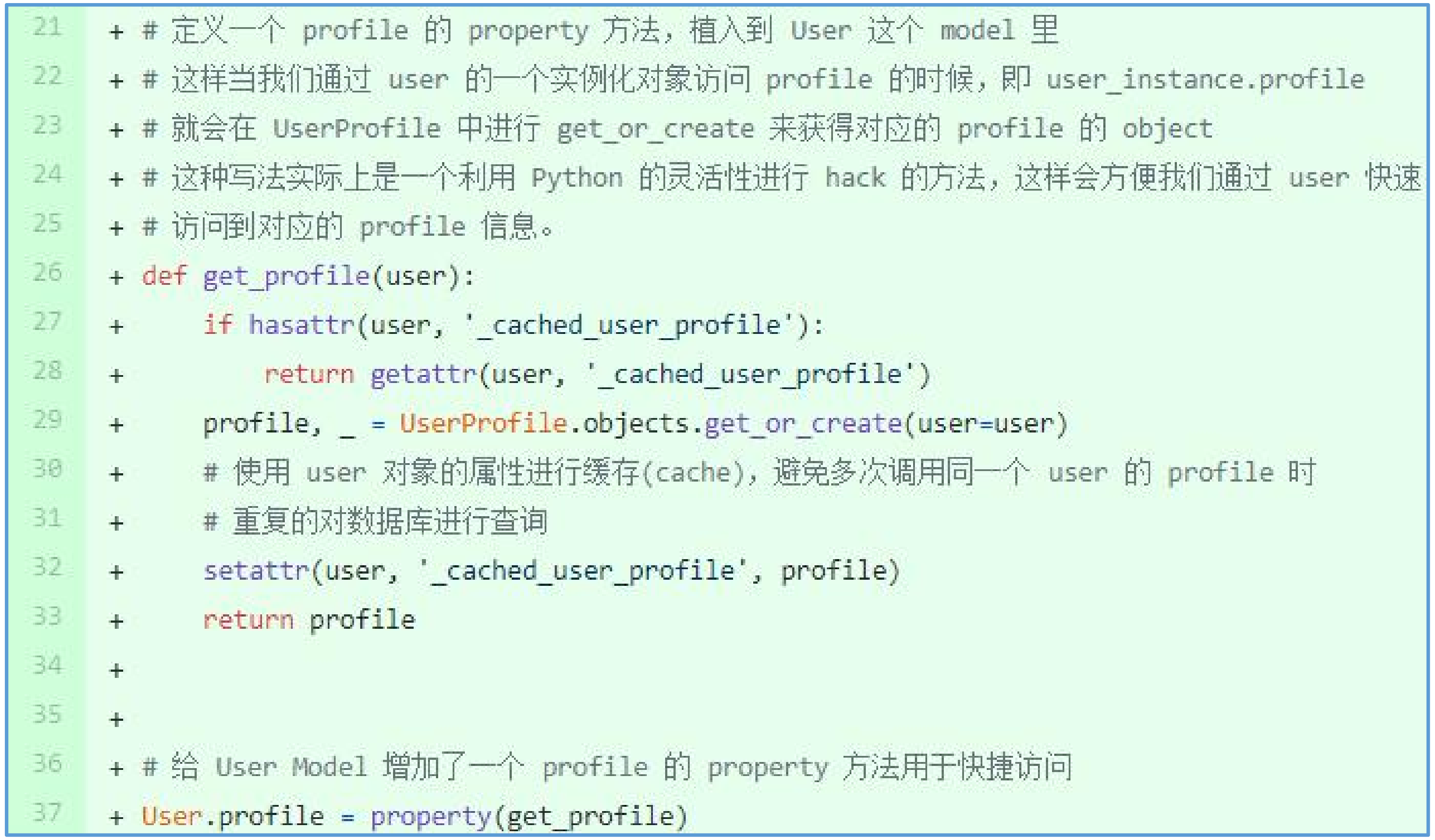
定义模型

路径为



```
 from django.db import models
+from django.contrib.auth.models import User

-# Create your models here.
+
+class UserProfile(models.Model):
+    # One2One field 会创建一个 unique index, 确保不会有多个 UserProfile 指向同一个 User
+    user = models.OneToOneField(User, on_delete=models.SET_NULL, null=True)
+    # Django 还有一个 ImageField, 但是尽量不要用, 会有很多的其他问题, 用 FileField 可以起到
+    # 同样的效果。因为最后我们都是以文件形式存储起来, 使用的是文件的 url 进行访问
+    avatar = models.FileField(null=True)
+    # 当一个 user 被创建之后, 会创建一个 user profile 的 object
+    # 此时用户还来不及去设置 nickname 等信息, 因此设置 null=True
+    nickname = models.CharField(null=True, max_length=200)
+    created_at = models.DateTimeField(auto_now_add=True)
+    updated_at = models.DateTimeField(auto_now=True)
+
+    def __str__(self):
+        return '{} {}'.format(self.user, self.nickname)
```

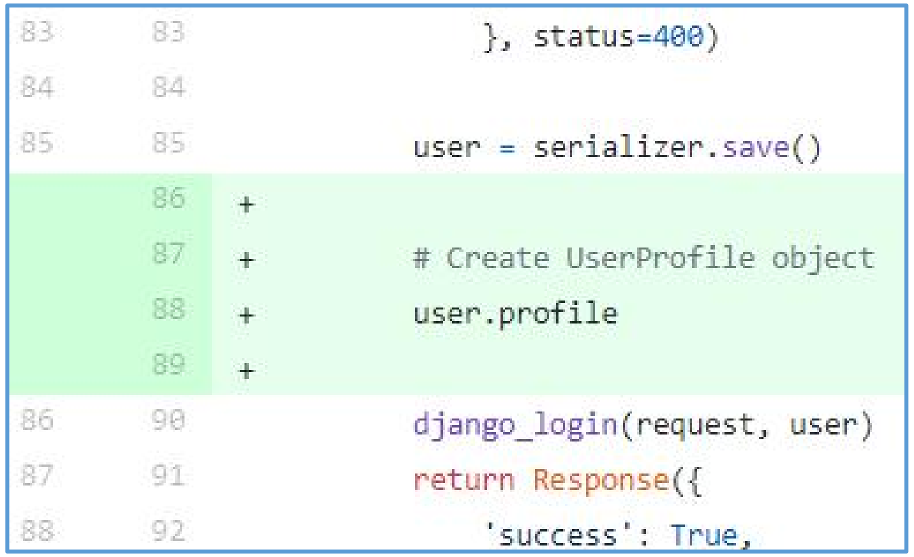
定义用户信息模型并配置 Admin 界面



```
+ # 定义一个 profile 的 property 方法，植入到 User 这个 model 里
+ # 这样当我们通过 user 的一个实例化对象访问 profile 的时候，即 user_instance.profile
+ # 就会在 UserProfile 中进行 get_or_create 来获得对应的 profile 的 object
+ # 这种写法实际上是一个利用 Python 的灵活性进行 hack 的方法，这样会方便我们通过 user 快速
+ # 访问到对应的 profile 信息。
+ def get_profile(user):
+     if hasattr(user, '_cached_user_profile'):
+         return getattr(user, '_cached_user_profile')
+     profile, _ = UserProfile.objects.get_or_create(user=user)
+     # 使用 user 对象的属性进行缓存(cache)，避免多次调用同一个 user 的 profile 时
+     # 重复的对数据库进行查询
+     setattr(user, '_cached_user_profile', profile)
+     return profile
+
+
+ # 给 User Model 增加了一个 profile 的 property 方法用于快捷访问
+ User.profile = property(get_profile)
```

定义用户信息模型并配置 Admin 界面

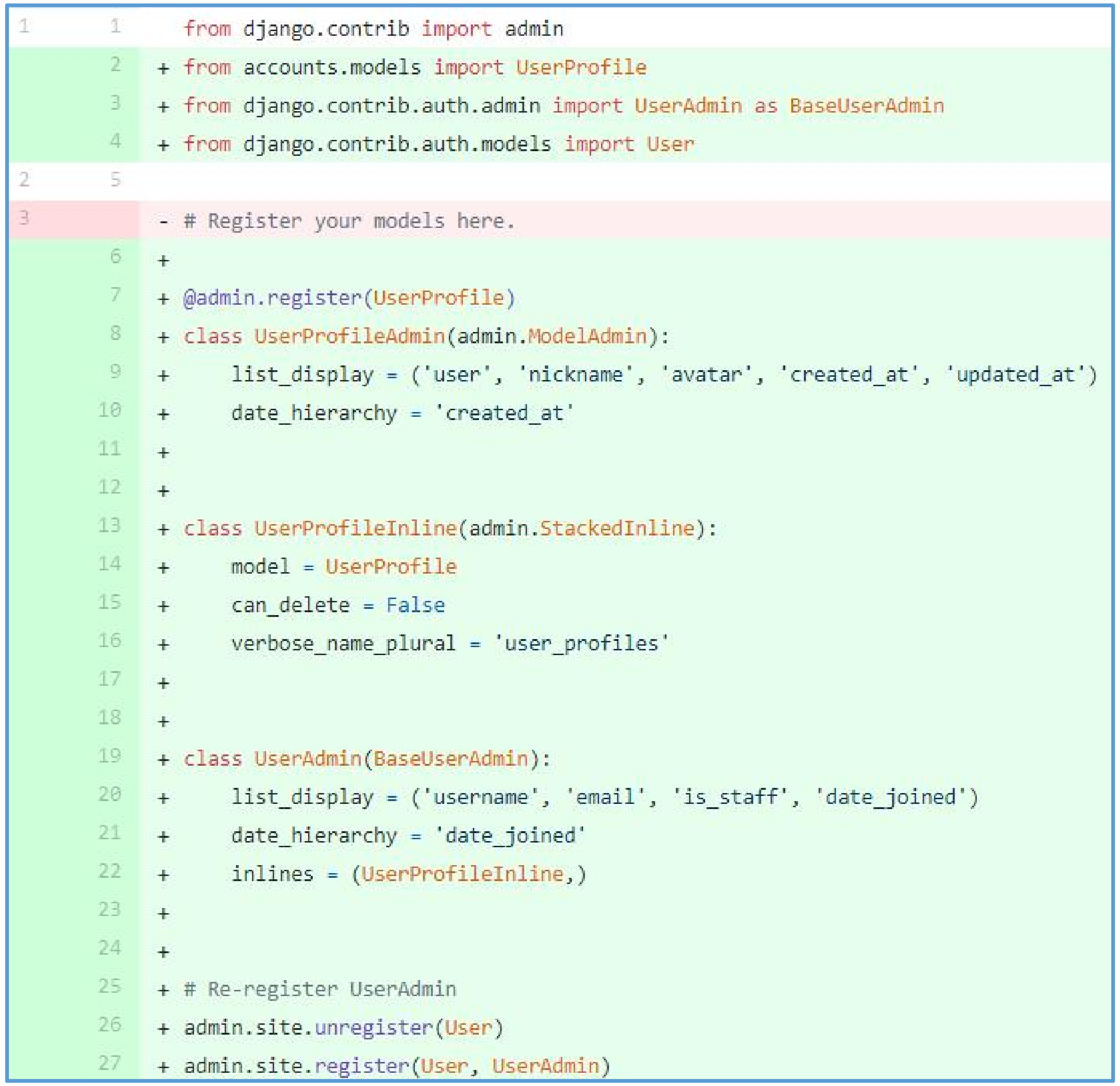
修改视图



```diff
            }, status=400)

        user = serializer.save()
+
+        # Create UserProfile object
+        user.profile
+
        django_login(request, user)
        return Response({
            'success': True,
```

定义用户信息模型并配置 Admin 界面

添加到 Admin



```
  from django.contrib import admin
+ from accounts.models import UserProfile
+ from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
+ from django.contrib.auth.models import User

- # Register your models here.
+
+ @admin.register(UserProfile)
+ class UserProfileAdmin(admin.ModelAdmin):
+     list_display = ('user', 'nickname', 'avatar', 'created_at', 'updated_at')
+     date_hierarchy = 'created_at'
+
+
+ class UserProfileInline(admin.StackedInline):
+     model = UserProfile
+     can_delete = False
+     verbose_name_plural = 'user_profiles'
+
+
+ class UserAdmin(BaseUserAdmin):
+     list_display = ('username', 'email', 'is_staff', 'date_joined')
+     date_hierarchy = 'date_joined'
+     inlines = (UserProfileInline,)
+
+
+ # Re-register UserAdmin
+ admin.site.unregister(User)
+ admin.site.register(User, UserAdmin)
```

05 实现上传用户头像和更新昵称 API

实现上传用户头像和更新昵称 API

amazon
S3

亚马逊简单的存储服务是互联网的存储。 它旨在使开发人员更容易制作网络级计算。

Amazon S3有一个简单的Web服务界面，您可以用来从Web上的任何位置存储和检索任何数量的数据。 它使任何开发人员访问相同的高度可扩展，可靠，快速，廉价的廉价的数据存储基础架构，即亚马逊用于运行自己的网站全球网络。 该服务旨在最大限度地提高规模的好处，并将这些利益传递给开发人员。

实现上传用户头像和更新昵称 API

在官网附件准备了AWS S3注册使用和配置教程，移步官网下载

AWS S3配置.pdf

https://www.jiuzhang.com/course/89?activeTab=5

教程包含以下内容

- 注册AWS账号，并登陆
- 创建S3 Bucket
- 创建IAM账户并授权访问S3 Bucket
- 获取Access key ID 和 Secret access key并配置到项目

实现上传用户头像和更新昵称 API

django-storages 和 boto3

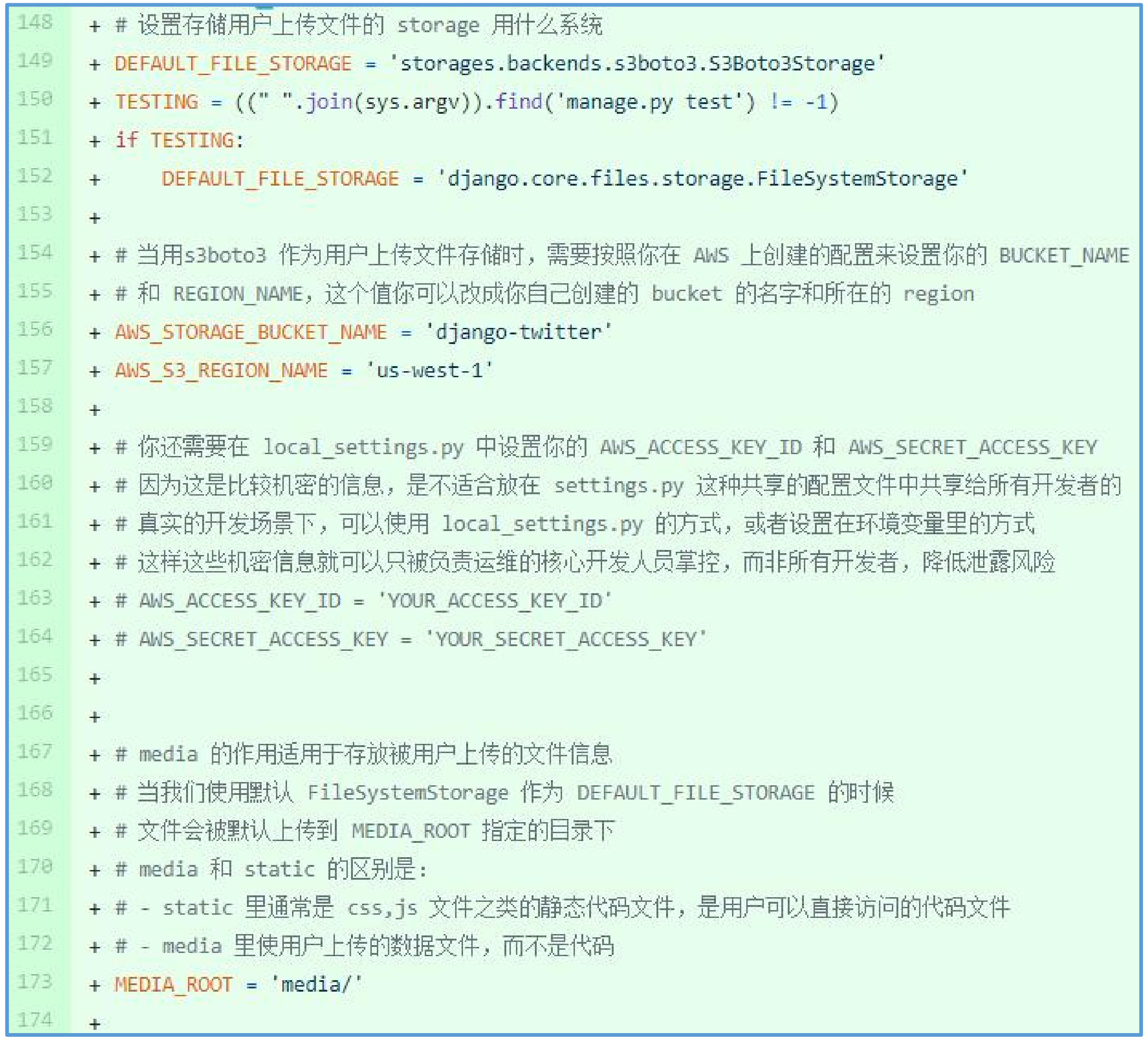
django-storages 官方文档:
https://django-storages.readthedocs.io/en/latest/backends/amazon-S3.html

boto3 官方文档:
https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

1 根据 requirements.txt 安装 django-storages 和 boto3

实现上传用户头像和更新昵称 API

2 配置 django-storages



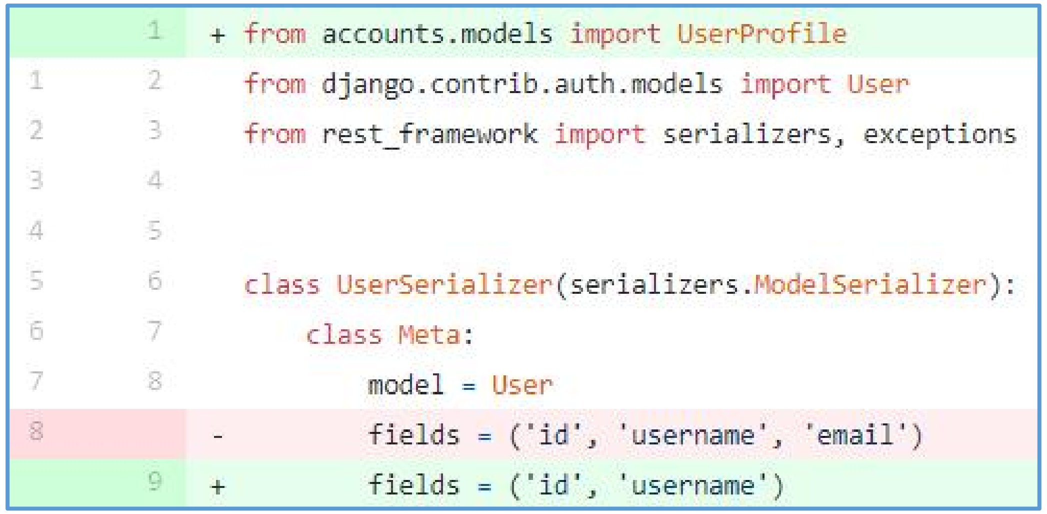
```
+ # 设置存储用户上传文件的 storage 用什么系统
+ DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
+ TESTING = ((" ".join(sys.argv)).find('manage.py test') != -1)
+ if TESTING:
+     DEFAULT_FILE_STORAGE = 'django.core.files.storage.FileSystemStorage'
+
+ # 当用s3boto3 作为用户上传文件存储时，需要按照你在 AWS 上创建的配置来设置你的 BUCKET_NAME
+ # 和 REGION_NAME，这个值你可以改成你自己创建的 bucket 的名字和所在的 region
+ AWS_STORAGE_BUCKET_NAME = 'django-twitter'
+ AWS_S3_REGION_NAME = 'us-west-1'
+
+ # 你还需要在 local_settings.py 中设置你的 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY
+ # 因为这是比较机密的信息，是不适合放在 settings.py 这种共享的配置文件中共享给所有开发者的
+ # 真实的开发场景下，可以使用 local_settings.py 的方式，或者设置在环境变量里的方式
+ # 这样这些机密信息就可以只被负责运维的核心开发人员掌控，而非所有开发者，降低泄露风险
+ # AWS_ACCESS_KEY_ID = 'YOUR_ACCESS_KEY_ID'
+ # AWS_SECRET_ACCESS_KEY = 'YOUR_SECRET_ACCESS_KEY'
+
+
+ # media 的作用适用于存放被用户上传的文件信息
+ # 当我们使用默认 FileSystemStorage 作为 DEFAULT_FILE_STORAGE 的时候
+ # 文件会被默认上传到 MEDIA_ROOT 指定的目录下
+ # media 和 static 的区别是：
+ # - static 里通常是 css,js 文件之类的静态代码文件，是用户可以直接访问的代码文件
+ # - media 里使用户上传的数据文件，而不是代码
+ MEDIA_ROOT = 'media/'
+
```

实现上传用户头像和更新昵称 API

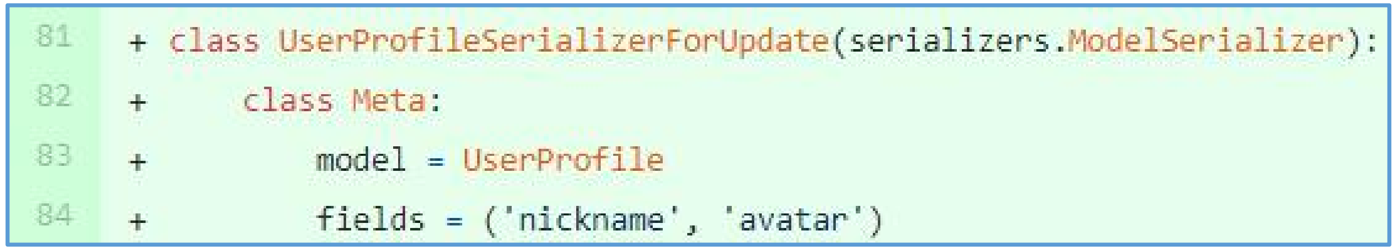
九章算法 www.jiuzhang.com

修改账户的序列化器

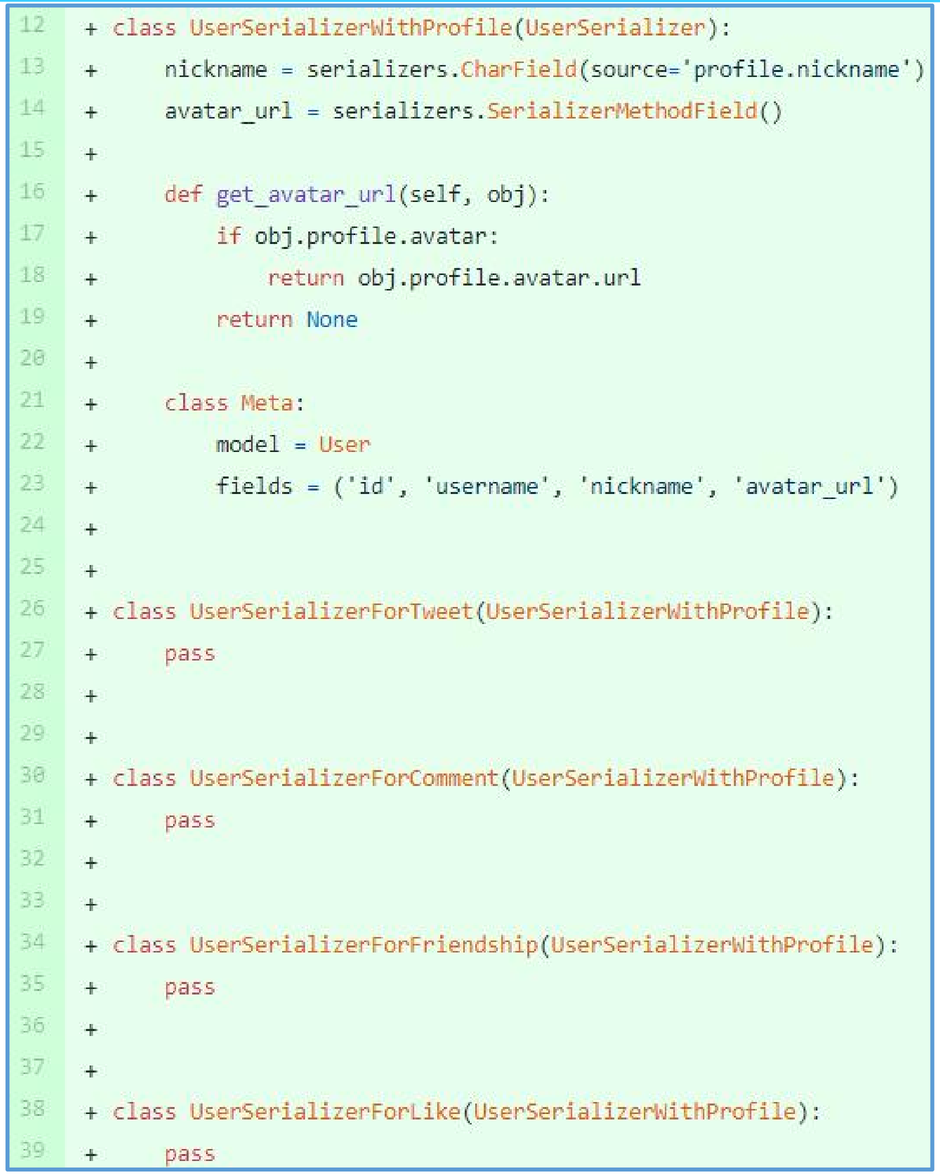
路径为 **accounts/api/serializers.py**



```diff
+ from accounts.models import UserProfile
  from django.contrib.auth.models import User
  from rest_framework import serializers, exceptions


  class UserSerializer(serializers.ModelSerializer):
      class Meta:
          model = User
-         fields = ('id', 'username', 'email')
+         fields = ('id', 'username')
```



```diff
+ class UserProfileSerializerForUpdate(serializers.ModelSerializer):
+     class Meta:
+         model = UserProfile
+         fields = ('nickname', 'avatar')
```

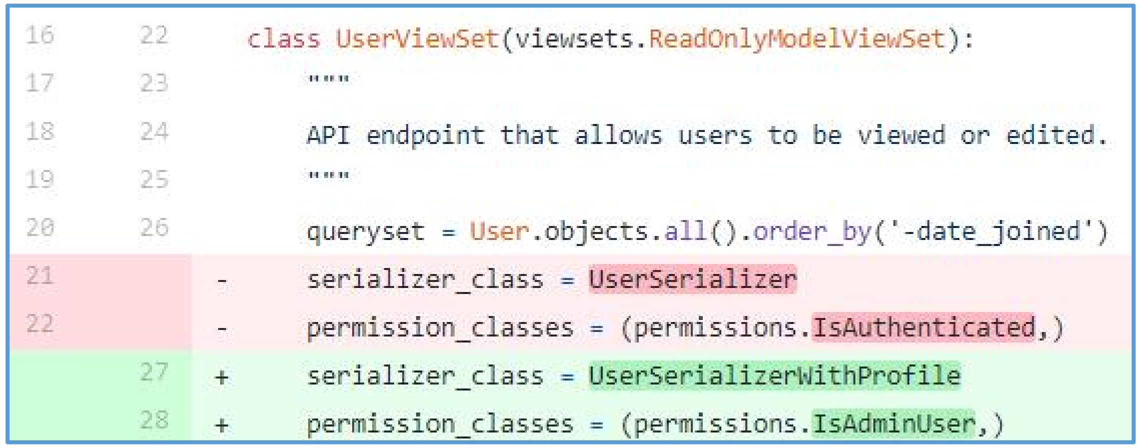


```diff
+ class UserSerializerWithProfile(UserSerializer):
+     nickname = serializers.CharField(source='profile.nickname')
+     avatar_url = serializers.SerializerMethodField()
+
+     def get_avatar_url(self, obj):
+         if obj.profile.avatar:
+             return obj.profile.avatar.url
+         return None
+
+     class Meta:
+         model = User
+         fields = ('id', 'username', 'nickname', 'avatar_url')
+
+
+ class UserSerializerForTweet(UserSerializerWithProfile):
+     pass
+
+
+ class UserSerializerForComment(UserSerializerWithProfile):
+     pass
+
+
+ class UserSerializerForFriendship(UserSerializerWithProfile):
+     pass
+
+
+ class UserSerializerForLike(UserSerializerWithProfile):
+     pass
```

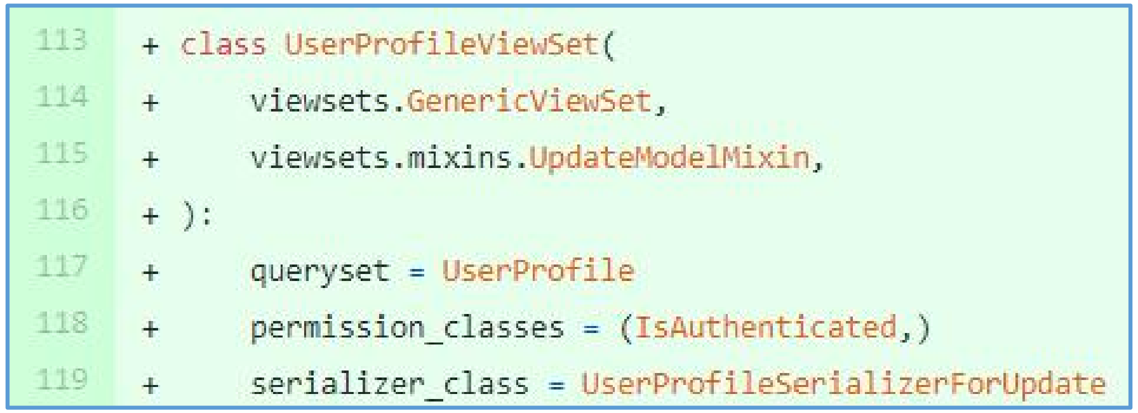
实现上传用户头像和更新昵称 API

修改账户的视图

路径为 **accounts/api/views.py**



```diff
  class UserViewSet(viewsets.ReadOnlyModelViewSet):
      """
      API endpoint that allows users to be viewed or edited.
      """
      queryset = User.objects.all().order_by('-date_joined')
-     serializer_class = UserSerializer
-     permission_classes = (permissions.IsAuthenticated,)
+     serializer_class = UserSerializerWithProfile
+     permission_classes = (permissions.IsAdminUser,)
```

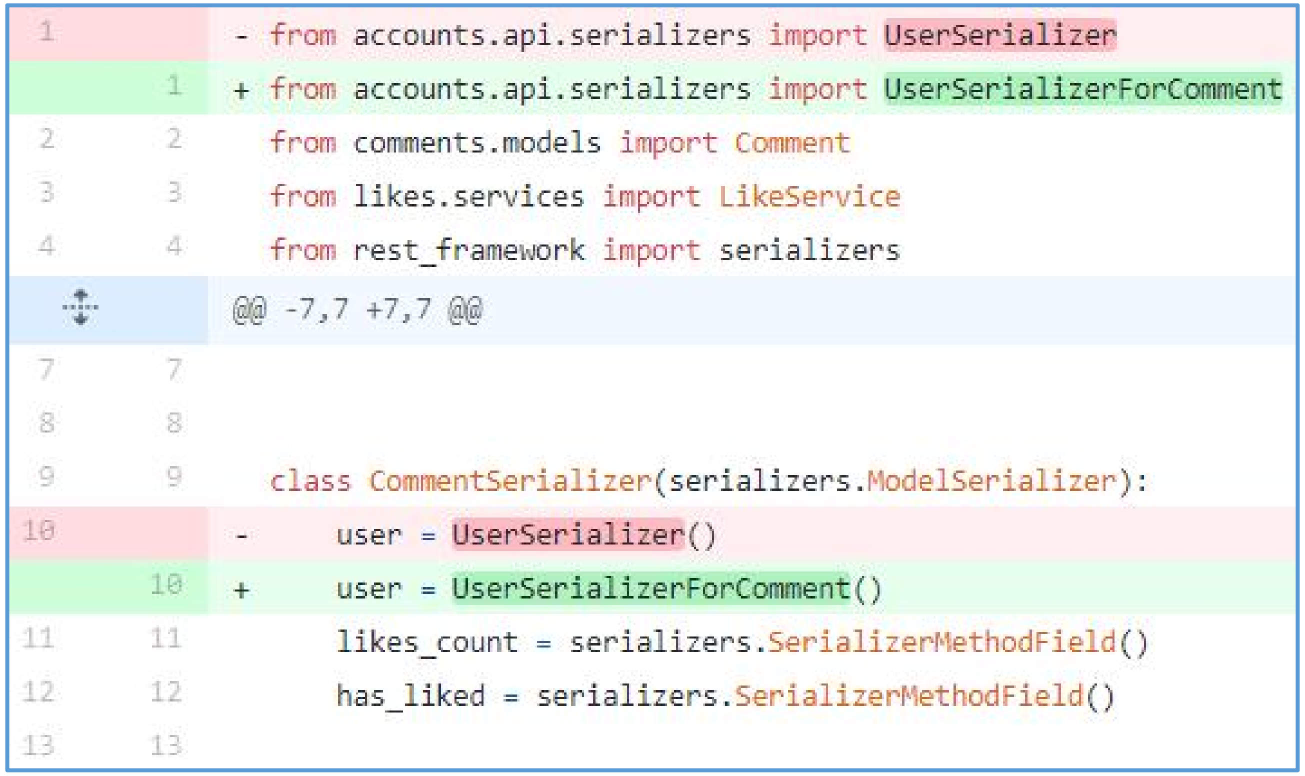


```diff
+ class UserProfileViewSet(
+     viewsets.GenericViewSet,
+     viewsets.mixins.UpdateModelMixin,
+ ):
+     queryset = UserProfile
+     permission_classes = (IsAuthenticated,)
+     serializer_class = UserProfileSerializerForUpdate
```

实现上传用户头像和更新昵称 API

修改评论的序列化器

路径为 **comments/api/serializers.py**



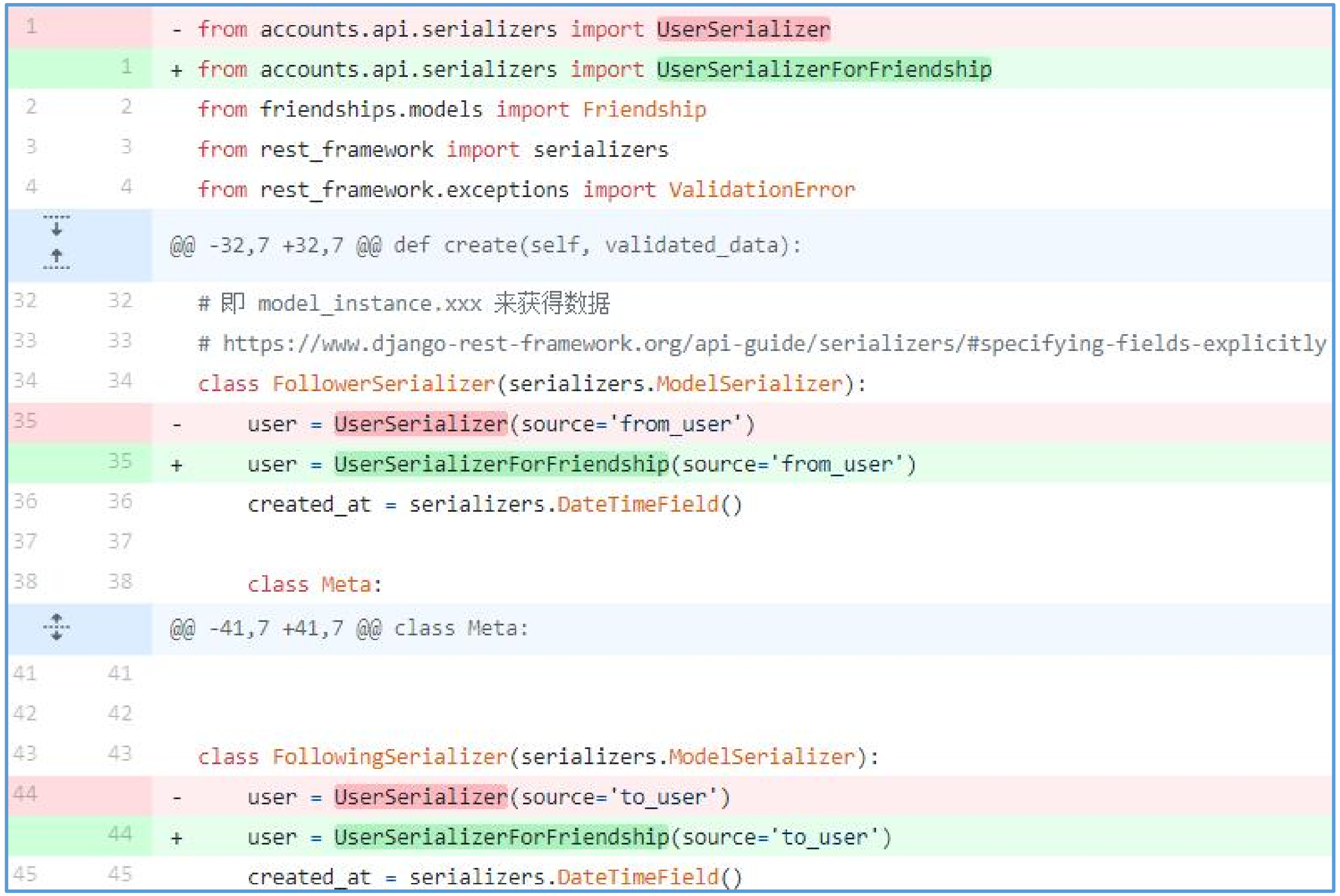
```
- from accounts.api.serializers import UserSerializer
+ from accounts.api.serializers import UserSerializerForComment
  from comments.models import Comment
  from likes.services import LikeService
  from rest_framework import serializers
@@ -7,7 +7,7 @@


  class CommentSerializer(serializers.ModelSerializer):
-     user = UserSerializer()
+     user = UserSerializerForComment()
      likes_count = serializers.SerializerMethodField()
      has_liked = serializers.SerializerMethodField()

```

实现上传用户头像和更新昵称 API

修改好友关系的序列化器

路径为

comments/api/serializers.py



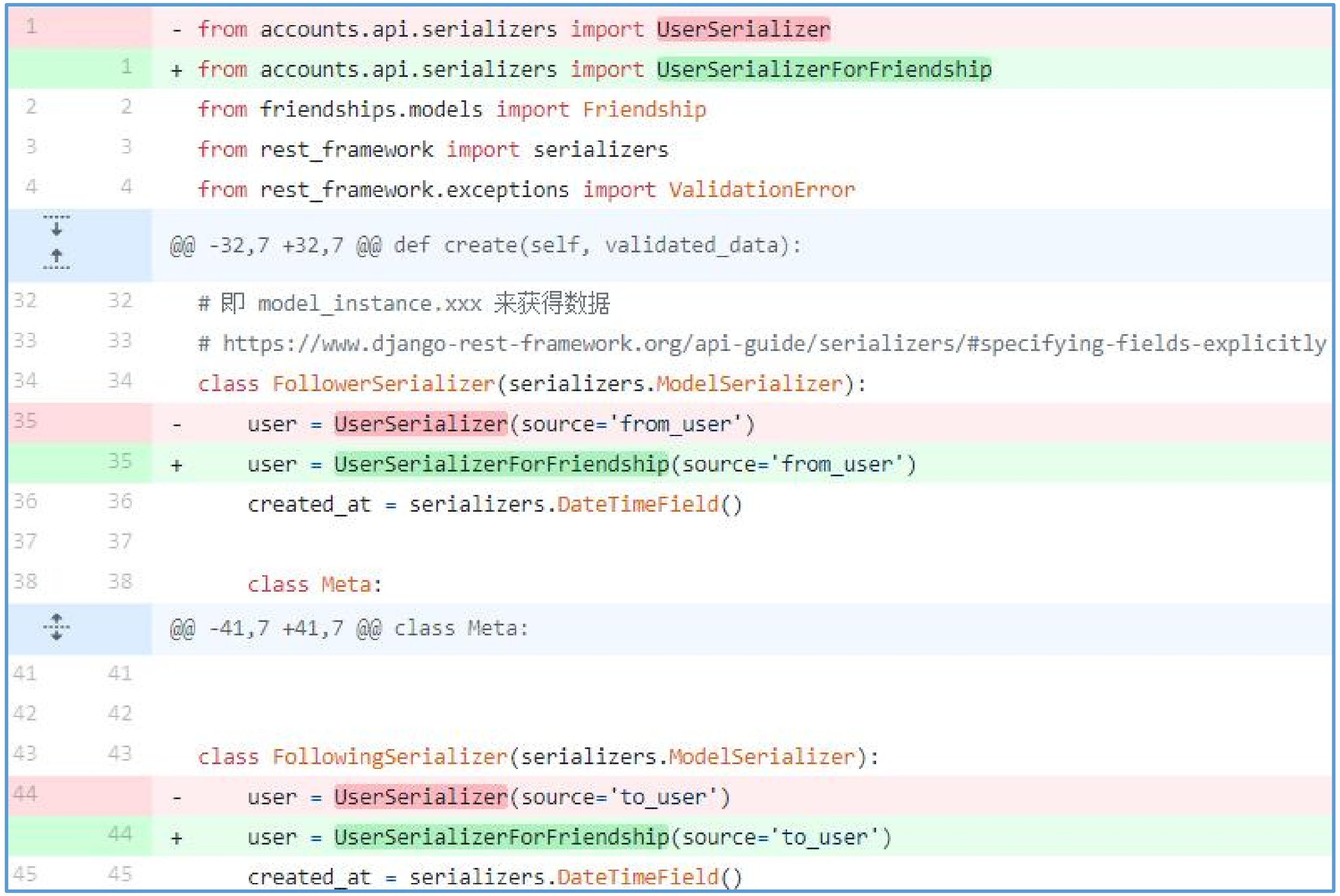
```
- from accounts.api.serializers import UserSerializer
+ from accounts.api.serializers import UserSerializerForFriendship
  from friendships.models import Friendship
  from rest_framework import serializers
  from rest_framework.exceptions import ValidationError
@@ -32,7 +32,7 @@ def create(self, validated_data):
  # 即 model_instance.xxx 来获得数据
  # https://www.django-rest-framework.org/api-guide/serializers/#specifying-fields-explicitly
  class FollowerSerializer(serializers.ModelSerializer):
-     user = UserSerializer(source='from_user')
+     user = UserSerializerForFriendship(source='from_user')
      created_at = serializers.DateTimeField()

      class Meta:
@@ -41,7 +41,7 @@ class Meta:


  class FollowingSerializer(serializers.ModelSerializer):
-     user = UserSerializer(source='to_user')
+     user = UserSerializerForFriendship(source='to_user')
      created_at = serializers.DateTimeField()
```

实现上传用户头像和更新昵称 API

修改好友关系的序列化器

路径为

friendships/api/serializers.py



```
- from accounts.api.serializers import UserSerializer
+ from accounts.api.serializers import UserSerializerForFriendship
  from friendships.models import Friendship
  from rest_framework import serializers
  from rest_framework.exceptions import ValidationError
@@ -32,7 +32,7 @@ def create(self, validated_data):
  # 即 model_instance.xxx 来获得数据
  # https://www.django-rest-framework.org/api-guide/serializers/#specifying-fields-explicitly
  class FollowerSerializer(serializers.ModelSerializer):
-     user = UserSerializer(source='from_user')
+     user = UserSerializerForFriendship(source='from_user')
      created_at = serializers.DateTimeField()

      class Meta:
@@ -41,7 +41,7 @@ class Meta:


  class FollowingSerializer(serializers.ModelSerializer):
-     user = UserSerializer(source='to_user')
+     user = UserSerializerForFriendship(source='to_user')
      created_at = serializers.DateTimeField()
```

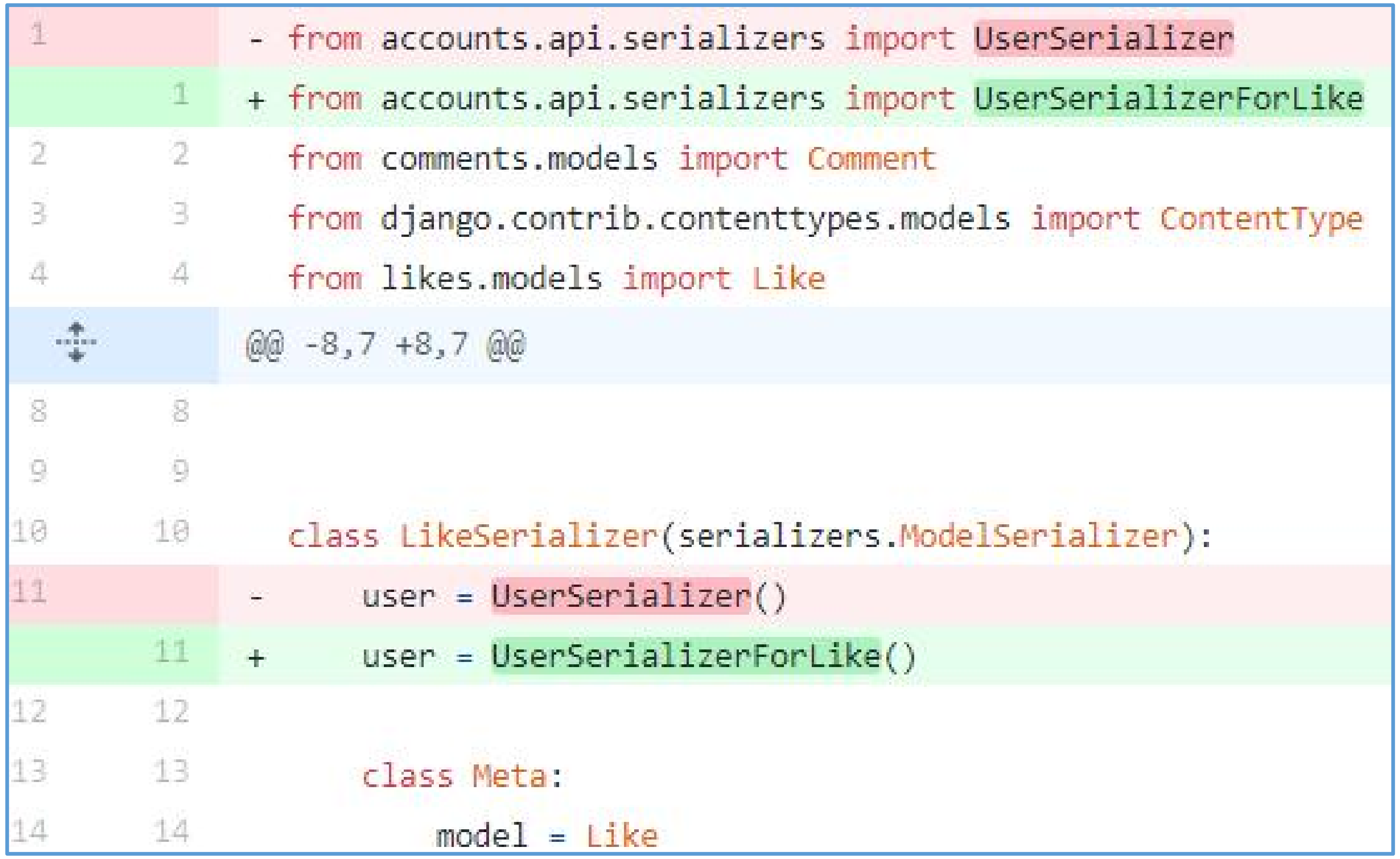
实现上传用户头像和更新昵称 API

九章算法
www.jiuzhang.com

修改点赞的序列化器

路径为

likes/api/serializers.py



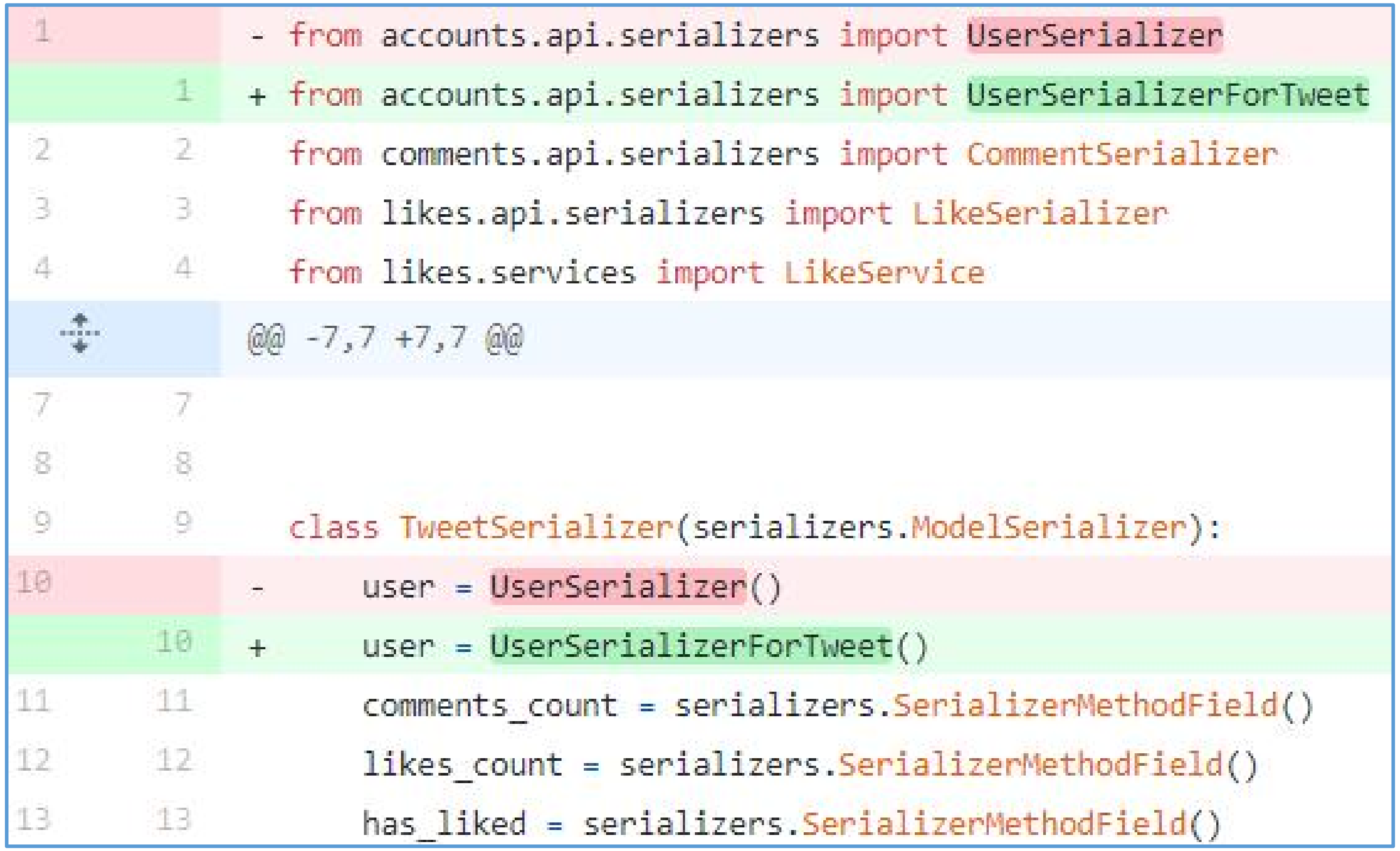
```
- from accounts.api.serializers import UserSerializer
+ from accounts.api.serializers import UserSerializerForLike
  from comments.models import Comment
  from django.contrib.contenttypes.models import ContentType
  from likes.models import Like
@@ -8,7 +8,7 @@


  class LikeSerializer(serializers.ModelSerializer):
-     user = UserSerializer()
+     user = UserSerializerForLike()

      class Meta:
          model = Like
```

实现上传用户头像和更新昵称 API

九章算法 www.jiuzhang.com

修改推文的序列化器

路径为
tweets/api/serializers.py

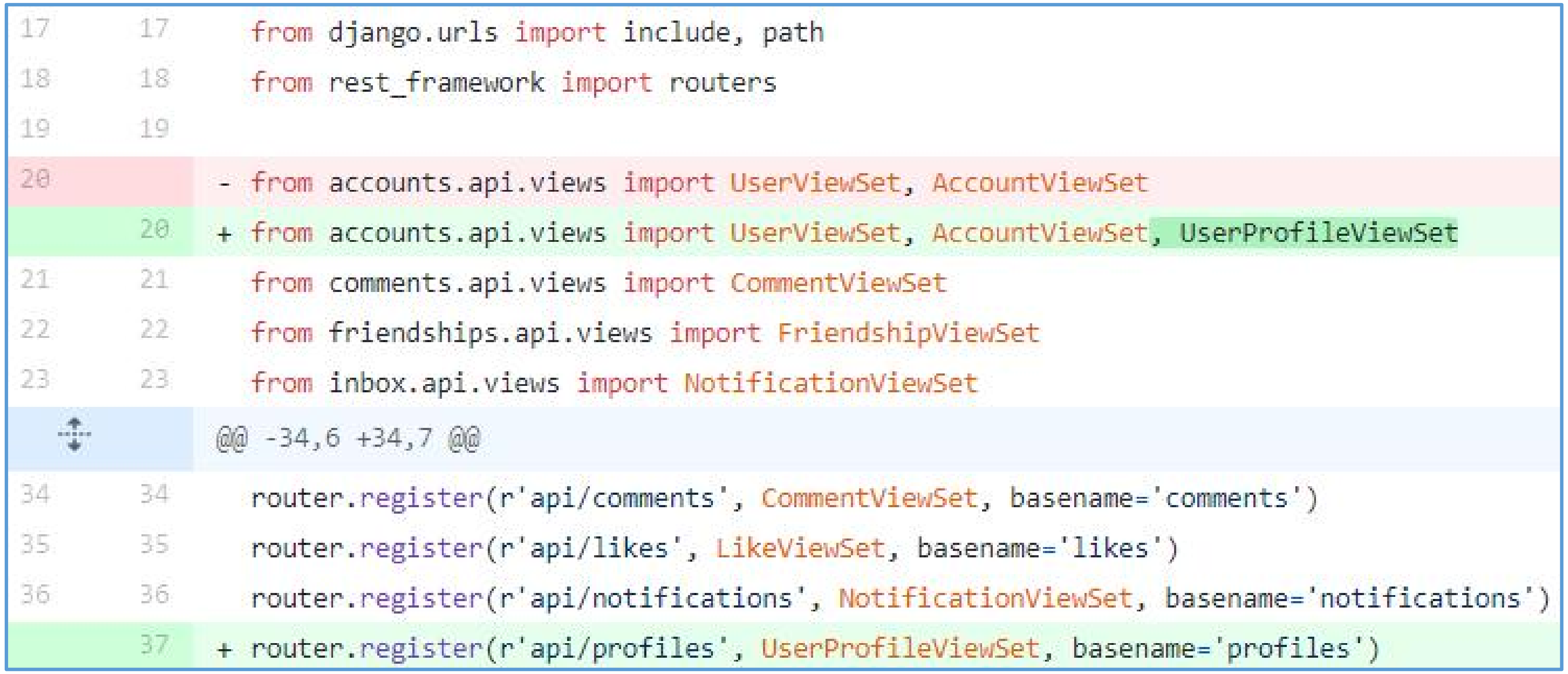


```diff
- from accounts.api.serializers import UserSerializer
+ from accounts.api.serializers import UserSerializerForTweet
  from comments.api.serializers import CommentSerializer
  from likes.api.serializers import LikeSerializer
  from likes.services import LikeService
@@ -7,7 +7,7 @@


  class TweetSerializer(serializers.ModelSerializer):
-     user = UserSerializer()
+     user = UserSerializerForTweet()
      comments_count = serializers.SerializerMethodField()
      likes_count = serializers.SerializerMethodField()
      has_liked = serializers.SerializerMethodField()
```

实现上传用户头像和更新昵称 API

配置路由

路径为 **twitter/urls.py**



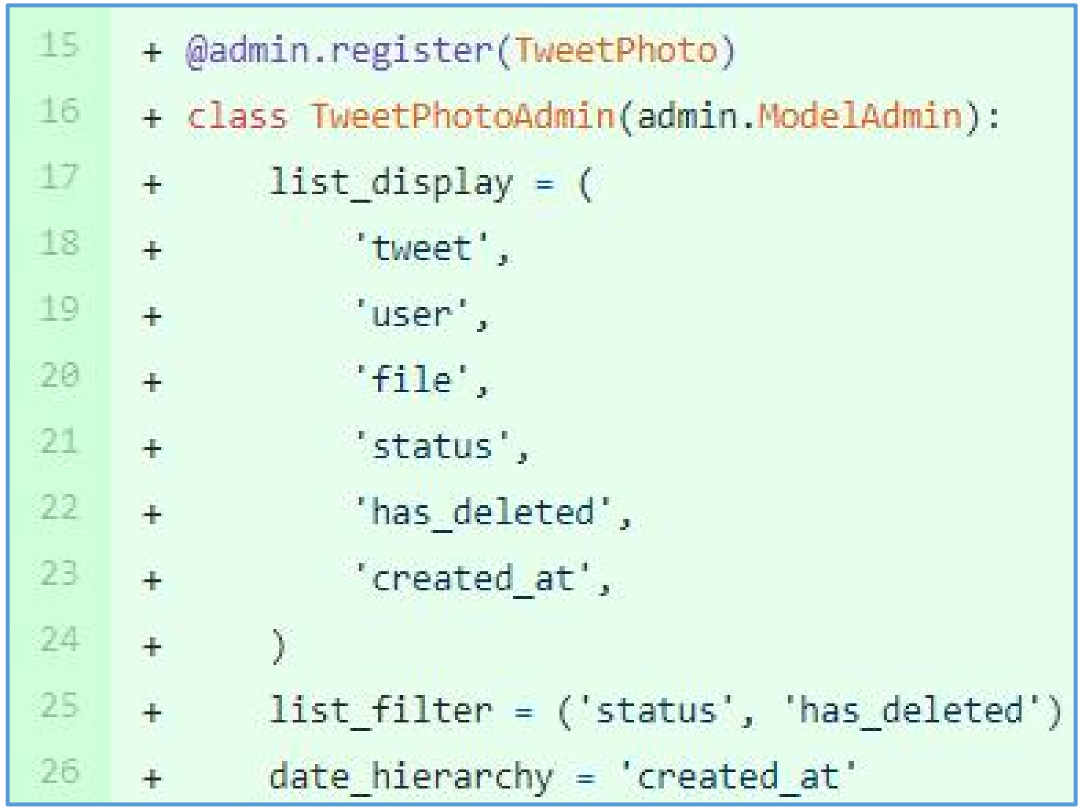
```
   from django.urls import include, path
   from rest_framework import routers

 - from accounts.api.views import UserViewSet, AccountViewSet
 + from accounts.api.views import UserViewSet, AccountViewSet, UserProfileViewSet
   from comments.api.views import CommentViewSet
   from friendships.api.views import FriendshipViewSet
   from inbox.api.views import NotificationViewSet
@@ -34,6 +34,7 @@
   router.register(r'api/comments', CommentViewSet, basename='comments')
   router.register(r'api/likes', LikeViewSet, basename='likes')
   router.register(r'api/notifications', NotificationViewSet, basename='notifications')
 + router.register(r'api/profiles', UserProfileViewSet, basename='profiles')
```

06 定义推文图片模型

定义推文图片模型

配置 Admin 界面

路径为 **tweets/admin.py**

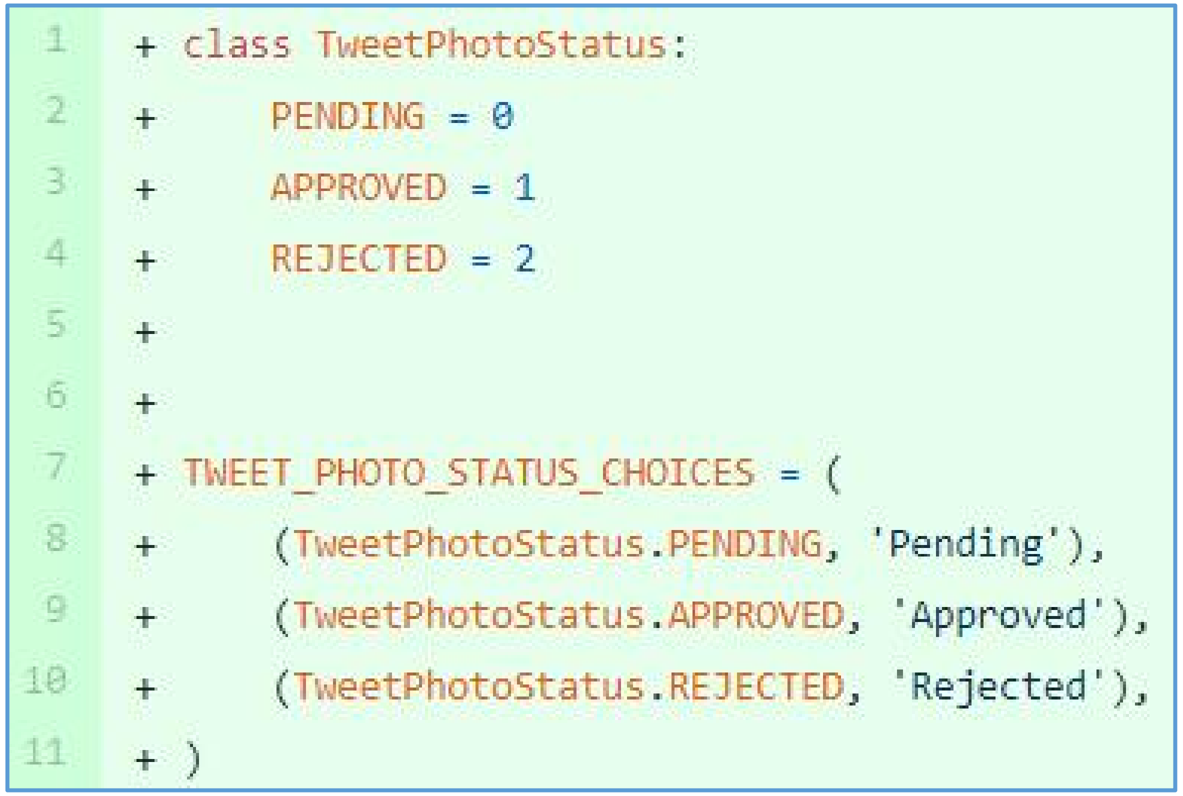


```
+ @admin.register(TweetPhoto)
+ class TweetPhotoAdmin(admin.ModelAdmin):
+     list_display = (
+         'tweet',
+         'user',
+         'file',
+         'status',
+         'has_deleted',
+         'created_at',
+     )
+     list_filter = ('status', 'has_deleted')
+     date_hierarchy = 'created_at'
```

定义推文图片模型

创建常量文件

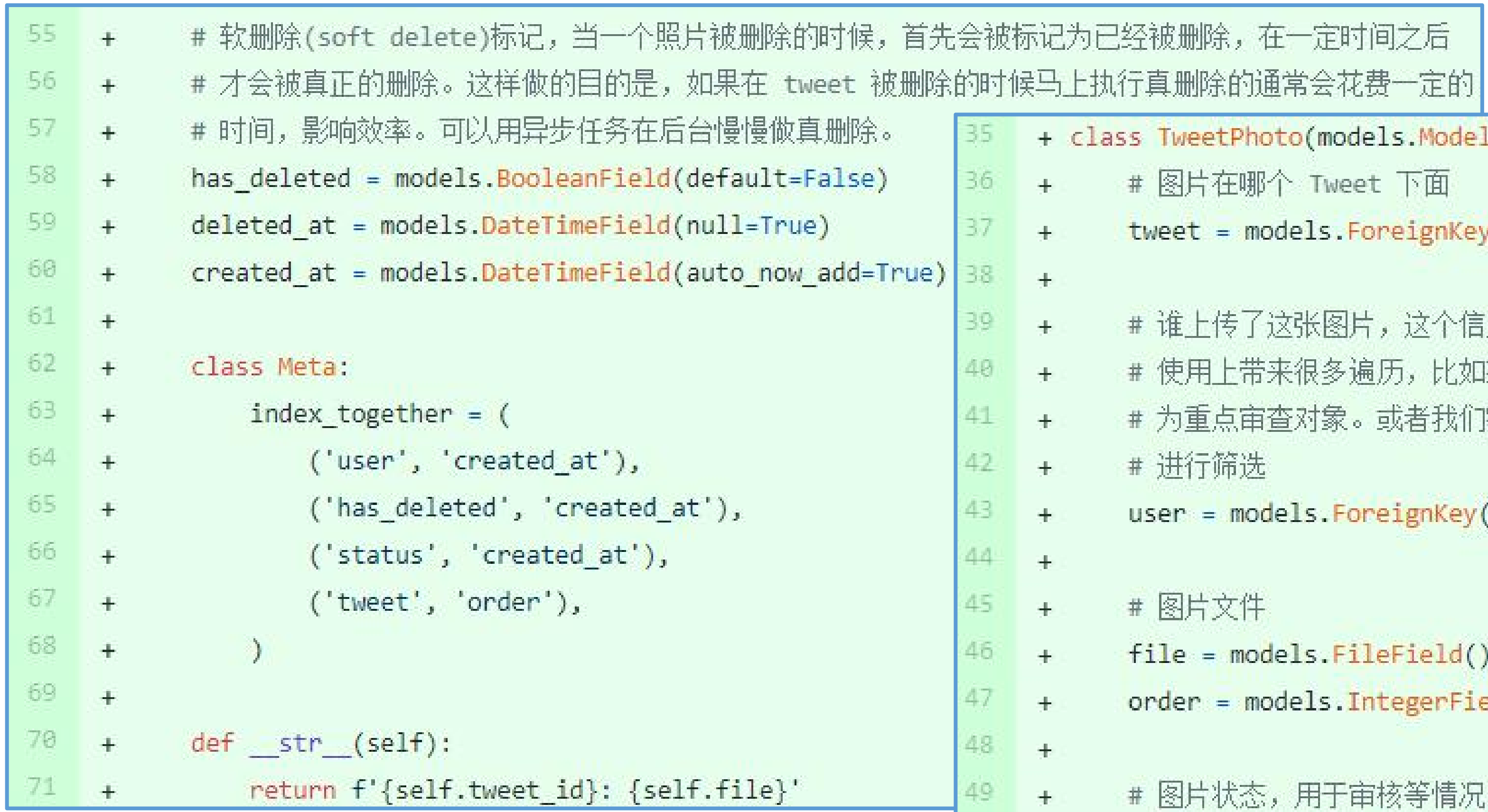
路径为 **tweets/constants.py**



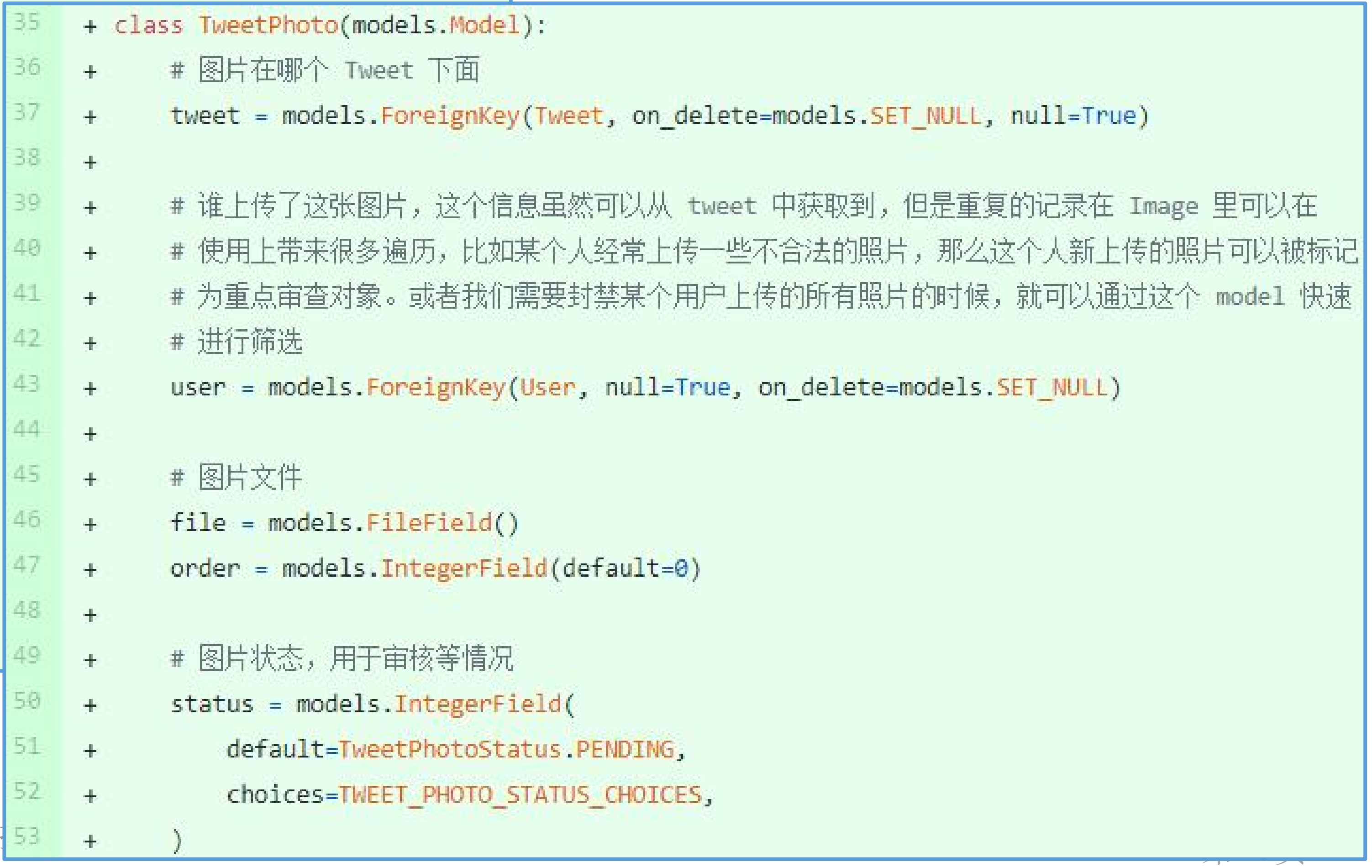
```
+ class TweetPhotoStatus:
+     PENDING = 0
+     APPROVED = 1
+     REJECTED = 2
+
+
+ TWEET_PHOTO_STATUS_CHOICES = (
+     (TweetPhotoStatus.PENDING, 'Pending'),
+     (TweetPhotoStatus.APPROVED, 'Approved'),
+     (TweetPhotoStatus.REJECTED, 'Rejected'),
+ )
```

定义推文图片模型

创建模型



```
+     # 软删除(soft delete)标记，当一个照片被删除的时候，首先会被标记为已经被删除，在一定时间之后
+     # 才会被真正的删除。这样做的目的是，如果在 tweet 被删除的时候马上执行真删除的通常会花费一定的
+     # 时间，影响效率。可以用异步任务在后台慢慢做真删除。
+     has_deleted = models.BooleanField(default=False)
+     deleted_at = models.DateTimeField(null=True)
+     created_at = models.DateTimeField(auto_now_add=True)
+
+     class Meta:
+         index_together = (
+             ('user', 'created_at'),
+             ('has_deleted', 'created_at'),
+             ('status', 'created_at'),
+             ('tweet', 'order'),
+         )
+
+     def __str__(self):
+         return f'{self.tweet_id}: {self.file}'
```



```
+ class TweetPhoto(models.Model):
+     # 图片在哪个 Tweet 下面
+     tweet = models.ForeignKey(Tweet, on_delete=models.SET_NULL, null=True)
+
+     # 谁上传了这张图片，这个信息虽然可以从 tweet 中获取到，但是重复的记录在 Image 里可以在
+     # 使用上带来很多遍历，比如某个人经常上传一些不合法的照片，那么这个人新上传的照片可以被标记
+     # 为重点审查对象。或者我们需要封禁某个用户上传的所有照片的时候，就可以通过这个 model 快速
+     # 进行筛选
+     user = models.ForeignKey(User, null=True, on_delete=models.SET_NULL)
+
+     # 图片文件
+     file = models.FileField()
+     order = models.IntegerField(default=0)
+
+     # 图片状态，用于审核等情况
+     status = models.IntegerField(
+         default=TweetPhotoStatus.PENDING,
+         choices=TWEET_PHOTO_STATUS_CHOICES,
+     )
```

路径为

tweets/models.py

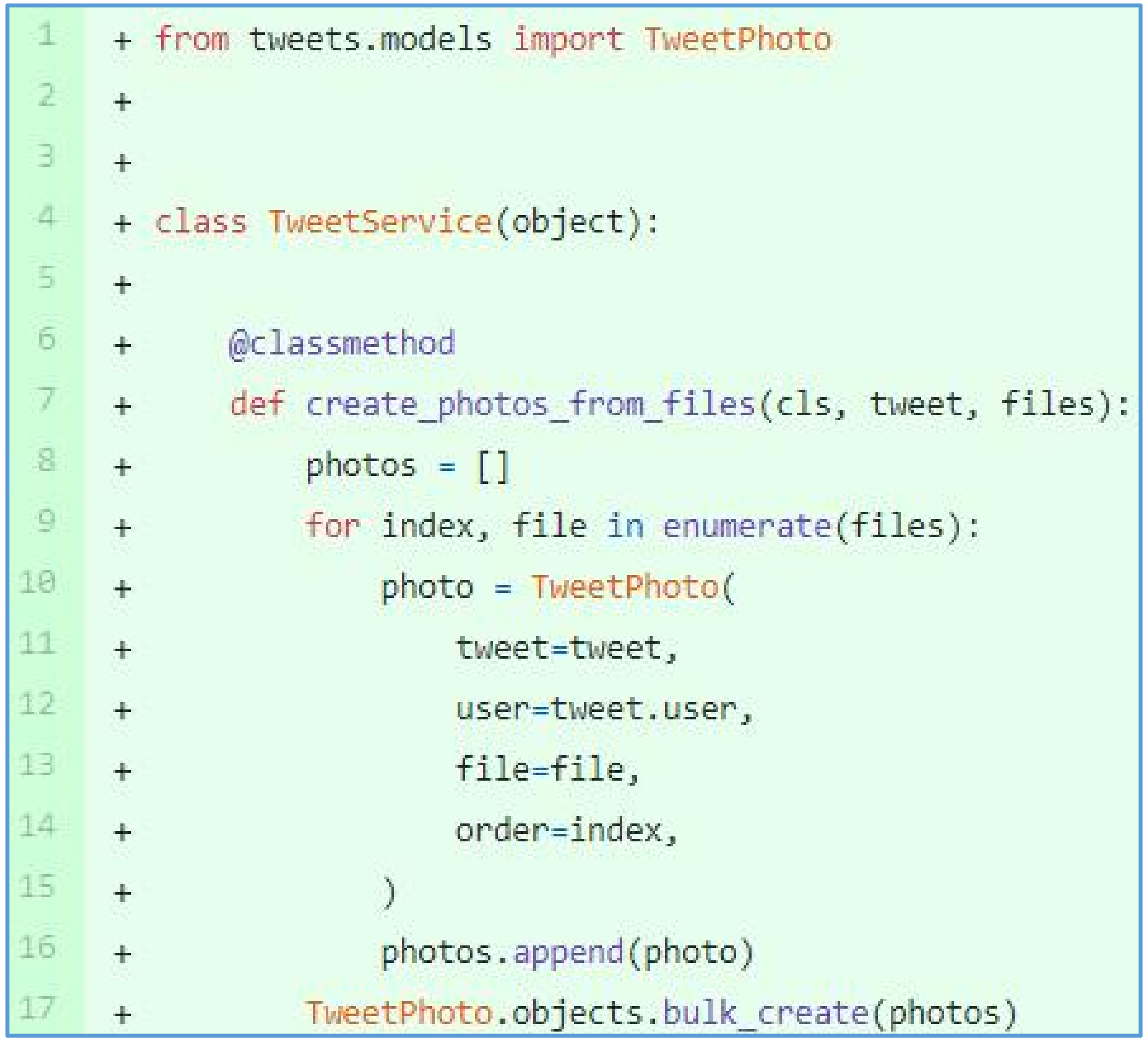
07 推文支持图片

推文支持图片

九章算法
www.jiuzhang.com

创建推文服务

路径为 **tweets/services.py**

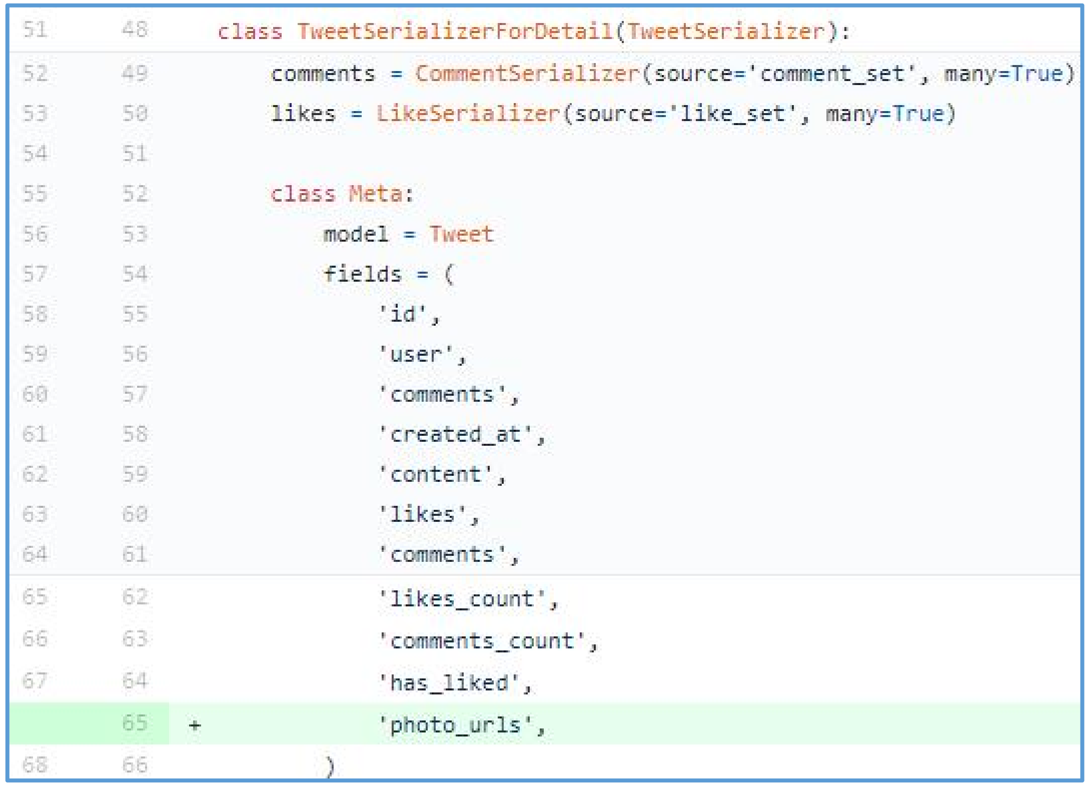


```
+ from tweets.models import TweetPhoto
+
+
+ class TweetService(object):
+
+     @classmethod
+     def create_photos_from_files(cls, tweet, files):
+         photos = []
+         for index, file in enumerate(files):
+             photo = TweetPhoto(
+                 tweet=tweet,
+                 user=tweet.user,
+                 file=file,
+                 order=index,
+             )
+             photos.append(photo)
+         TweetPhoto.objects.bulk_create(photos)
```

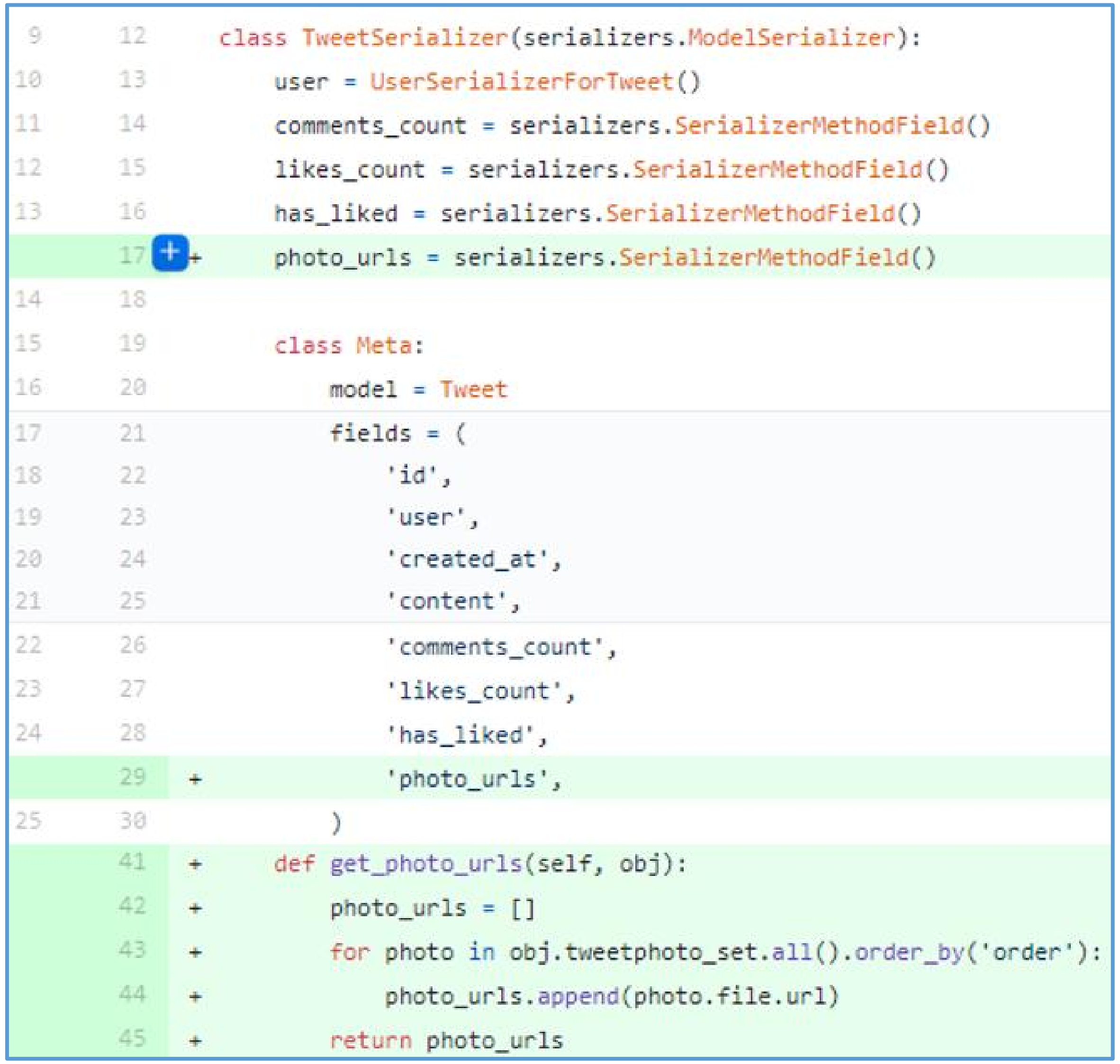
推文支持图片

创建推文服务

路径为 **tweets/services.py**

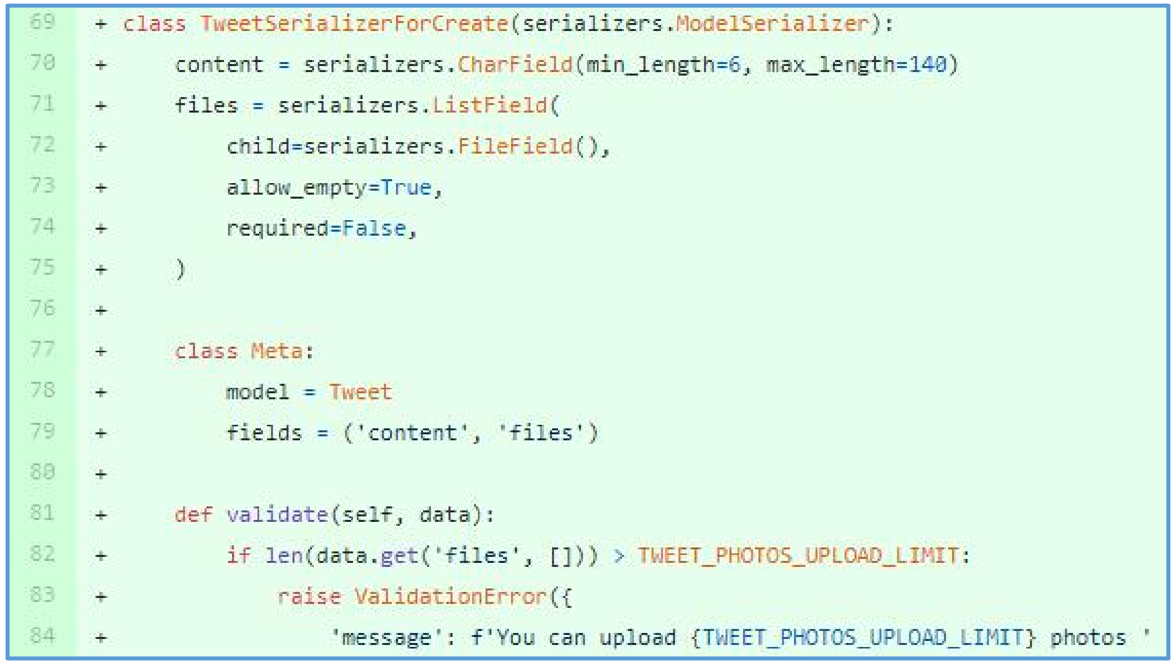


```
+   class TweetSerializerForDetail(TweetSerializer):
        comments = CommentSerializer(source='comment_set', many=True)
        likes = LikeSerializer(source='like_set', many=True)

        class Meta:
            model = Tweet
            fields = (
                'id',
                'user',
                'comments',
                'created_at',
                'content',
                'likes',
                'comments',
                'likes_count',
                'comments_count',
                'has_liked',
+               'photo_urls',
            )
```

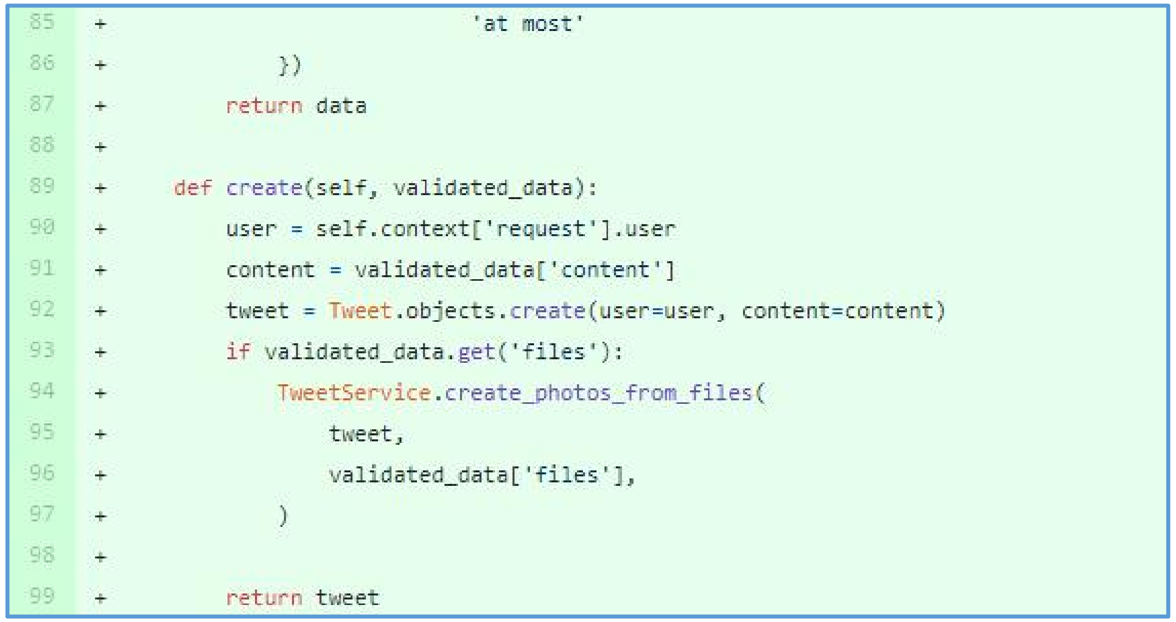


```
class TweetSerializer(serializers.ModelSerializer):
    user = UserSerializerForTweet()
    comments_count = serializers.SerializerMethodField()
    likes_count = serializers.SerializerMethodField()
    has_liked = serializers.SerializerMethodField()
+   photo_urls = serializers.SerializerMethodField()

    class Meta:
        model = Tweet
        fields = (
            'id',
            'user',
            'created_at',
            'content',
            'comments_count',
            'likes_count',
            'has_liked',
+           'photo_urls',
        )
+   def get_photo_urls(self, obj):
+       photo_urls = []
+       for photo in obj.tweetphoto_set.all().order_by('order'):
+           photo_urls.append(photo.file.url)
+       return photo_urls
```

推文支持图片



```
+ class TweetSerializerForCreate(serializers.ModelSerializer):
+     content = serializers.CharField(min_length=6, max_length=140)
+     files = serializers.ListField(
+         child=serializers.FileField(),
+         allow_empty=True,
+         required=False,
+     )
+
+     class Meta:
+         model = Tweet
+         fields = ('content', 'files')
+
+     def validate(self, data):
+         if len(data.get('files', [])) > TWEET_PHOTOS_UPLOAD_LIMIT:
+             raise ValidationError({
+                 'message': f'You can upload {TWEET_PHOTOS_UPLOAD_LIMIT} photos '
```



```
+                            'at most'
+             })
+         return data
+
+     def create(self, validated_data):
+         user = self.context['request'].user
+         content = validated_data['content']
+         tweet = Tweet.objects.create(user=user, content=content)
+         if validated_data.get('files'):
+             TweetService.create_photos_from_files(
+                 tweet,
+                 validated_data['files'],
+             )
+
+         return tweet
```