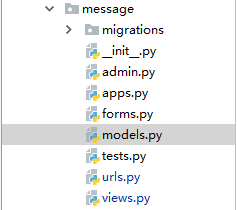
**消息中心开发文档**

# 意见反馈

文件结构：



文件详情：

（1）message/models.py

导入要用的包，定义每个字段，定义元数据和排序依据。

联系管理员相当于留言板，需要定义的字段主要是留言的标题、留言人、留言内容和留言时间。

# 在留言用户的字段是用户表的外键，使用前要导入

**from** django.db **import** models  
**from** users.models **import** UserProfile  
  
**class** Msg(models.Model):  
 # 设置一个自动增长的id  
 id = models.AutoField(primary\_key=**True**)  
 #存取留言标题  
 title = models.CharField(max\_length=**30**)  
 #存取留言内容  
 content = models.TextField()  
 #存取留言用户的id信息  
 user = models.ForeignKey(UserProfile, on\_delete=models.CASCADE)  
 # 用于存取留言时间。 其中， auto\_now\_add 参数  
 # 表示在建立一个新的 Msg 数据模型对象时，  
 # 自动设置该属性值为当前时间。  
 datetime = models.DateTimeField(auto\_now\_add=**True**)  
 # 内部类 class Meta 用于给 model 定义元数据  
 # 元数据：不是一个字段的任何数据  
 **class** Meta:  
 verbose\_name = **'留言板内容'** ordering = [**'-datetime'**] # 按datatime字段排序，默认为升序，`-`表示倒序，`?`表示随机  
 db\_table = **"msg"** # 该类对应数据库表名  
  
 # 函数 \_\_str\_\_ 定义当调用对象的 str() 方法时的返回值内容  
 # 它最常见的就是在Django管理后台中做为对象的显示值。因此应该总是为 \_\_str\_\_ 返回一个友好易读的字符串  
 **def** \_\_str\_\_(self):  
 # return self.title 将留言标题返回  
 **return** self.title

完成后在Tools>>Run manage.py Task输入makemigrations和migrate进行数据库的迁移。

（2）message/forms.py

定义表单传回的数据对应的是数据库的哪些字段

**from** django **import** forms  
**from** .models **import** Msg  
  
  
**class** MsgPostForm(forms.ModelForm):  
 **class** Meta:  
 model = Msg # 对应的表  
 fields = (**'title'**, **'content'**) #对应的标题和内容字段。

（3）message/views.py

显示索引页的函数

**def** index(*request*):  
 **return** render(*request*, **'message/msg\_index.html'**)

显示所有留言的函数，objects.all()表示取出所有留言的所有字段，”allInfo”表示这个记录集在**'message/msg\_showAll.html'**这个前端页面上的表示

#所有留言  
**def** showAll(*request*):  
 allInfo = Msg.objects.all()  
 **return** render(*request*, **'message/msg\_showAll.html'**, {**'allInfo'**: allInfo})

以留言的id作为参数，用get方法查找符合的记录

#留言详情  
**def** message\_detail(*request*, *id*):  
 message = get\_object\_or\_404(Msg, id=*id*)  
 **return** render(*request*, **'message/msg\_detail.html'**, {**'message'**:message})

写留言需要用户登录以后才能操作。先判断是get还是post，如果是post就存储表单传回来的数据到数据库，如果是get就显示留言界面。

#写留言  
@login\_required  
**def** msg\_post\_page(*request*):  
 **if** *request*.method==**'POST'**:  
 form =MsgPostForm(data=*request*.POST)  
 **if** form.is\_valid():  
 user = *request*.user  
 title = *request*.POST.get(**'title'**, **''**)  
 content = *request*.POST.get(**'content'**, **''**)  
 newmessage = Msg()  
 newmessage.user = user  
 newmessage.title = title  
 newmessage.content = content  
 newmessage.save()  
 **return** render(*request*, **'message/msg\_succeed.html'**)  
 **else**:  
 form = MsgPostForm()  
 context = {**'form'**:form}  
 **return** render(*request*, **'message/msg\_add.html'**, context)

#成功界面  
**def** succeed(*request*):  
 **return** render(*request*, **'message/msg\_succeed.html'**)

前端传回用户名为参数，获取该用户，再获取该用户的所有留言

#某个用户的所有留言

**def** user\_msg\_list\_page(*request*, *username*):  
 #需要用get\_object\_or\_404返回能够被Msg使用的参数。  
 user = get\_object\_or\_404(UserProfile, username=*username*)  
 #查找符合条件的记录集  
 user\_msgs = Msg.objects.filter(user=user)  
 context ={**'user\_msgs'**:user\_msgs}  
 **return** render(*request*, **'message/user\_msg\_list\_page.html'**, context)

获取当前用户，再从所有留言中获取当前用户的留言

#当前用户的所有留言  
**def** my\_msg\_list\_page(*request*):  
 user = *request*.user  
 my\_msgs = Msg.objects.filter(user=user)  
 context = {**'my\_msgs'**:my\_msgs}  
 **return** render(*request*, **'message/my\_msg\_list\_page.html'**, context)

（4）urls.py

除了显示某个用户的所有留言时需要用到正则表达式获取用户名用re\_path以外，其他的url表示都是用path，定义url,声明需要调用的参数和定义别名。需要声明一下该app的名称，否则在总的urls.py中无法用include调用该app的urls.py

**from** django.urls **import** path, re\_path  
**from** . **import** views  
  
app\_name=**'message'**urlpatterns = [  
 #消息首页  
 path(**''**, views.index, name=**'msg\_index'**),  
 # 所有留言  
 path(**'msg\_showAll/'**, views.showAll, name=**'msg\_showAll'**),  
 # 留言详情  
 path(**'msg\_detail/<int:id>'**, views.message\_detail, name=**'msg\_detail'**),  
 path(**'msg\_succeed/'**, views.succeed, name=**'succeed'**),  
 #添加留言  
 path(**'msg\_add/'**, views.msg\_post\_page, name=**'msg\_add'**),  
 #某个用户的留言,正则表达式\w+ 用于获取 url 请求地址中的用户名。  
 re\_path(**r'^user\_msg\_list/([\w\W]\*)/$'**, views.user\_msg\_list\_page, name=**'user\_msg\_list'**),  
 #我的留言  
 path(**'my\_msg\_list/'**, views.my\_msg\_list\_page, name=**'my\_msg\_list'**),  
  
]

设置总路由：

MxOnline/urls.py中增加如下的语句：

path(**'message/'**, include(**'message.urls'**, namespace=**'message'**)),

（5）前端页面

msg\_index.html

该是整个消息中心的索引页，有三个接口是分别指向三个功能的

<**ul**>  
 <**li**>  
 <**a** *href=***"**{% **url 'systermmsg:sys\_msg\_list'** %}**"** {% **block** *class1* %} {% **endblock** *class1* %}>  
 <**i** *class=***"icon iconfont icon-lingdang"**></**i**>  
 系统消息  
 </**a**>  
 </**li**>  
 <**li**>  
 <**a** *href=***"**{% **url "message:msg\_showAll"** %}**"** {% **block** *class4* %} {% **endblock** *class4* %}>  
 <**i** *class=***"icon iconfont icon-wenda"**></**i**>  
 意见反馈  
 </**a**>  
 </**li**>  
 <**li**>  
 <**a** *href=***"**{% **url 'chat:allchats'** %}**"** {% **block** *class5* %} {% **endblock** *class5* %}>  
 <**i** *class=***"icon iconfont icon-pinglun"**></**i**>  
 私聊  
 </**a**>  
 </**li**>  
</**ul**>

右边部分是每个部分的内容不同的，因此需要有被继承的空间

<**div** *class=***"right"**>  
 {% **block** *top* %}  
 <**div** *class=***"tap"**>  
 <**span**>查看消息</**span**>  
 </**div**>  
 {% **endblock** *top* %}  
 <**div** *class=***"container"**>  
 {% **block** *container* %}  
 <**div** *class=***"no-doc"**>  
 <**img** *src=***"../../static/message/images/left.jpg"** *width=***"200"**/>  
 <**p**>左侧查看消息哦~</**p**>  
 </**div**>  
 {% **endblock** *container* %}  
 </**div**>  
</**div**>

msg\_showAll.html

该页面是所有留言的列表，在该页面中还有添加留言和显示自己留言的入口。在刚刚的views.py 中定义了在此页面中用到的queryset的名称allInfo，用for循环取出该queryset中的每一条message，获取该message的每个字段并使用。点击留言详情的时候要把留言的id传回url作为参数以便views.py查询相应的内容

{% **block** *container* %}  
 <**header**>  
 <**button** *onclick=***"*window***.location.href=**'**{% **url 'message:msg\_add'** %}**'"**>添加留言</**button**>  
 <**button** *onclick=***"*window***.location.href=**'**{% **url 'message:my\_msg\_list'** %}**'"**>我的留言</**button**>  
 </**header**>  
<**div** *class=***"container"**>  
 <**div** *class=***"row mt-2"**>  
 {% **for** *message* **in** *allInfo* %}  
 <**div** *class=***"col-4 mb-4"**>  
 <**div** *class=***"card h-100"**>  
 <**h4** *class=***"card-header"**>{{ *message*.*title* }}</**h4**>  
 <**a** *href=***"**{% **url 'message:user\_msg\_list'** *message*.*user*%}**"**>留言人：{{ *message*.*user* }}</**a**>  
 <**div** *class=***"card-body"**>  
 <**p** *class=***"card-text"**>{{ *message*.*content*|*slice*:**'30'** }}...</**p**>  
 </**div**>  
 <**div** *class=***"card-footer"**>  
 <**a** *href=***"**{% **url 'message:msg\_detail'** *message*.*id* %}**"** *class=***"btn btn-primary"**>查看详情</**a**>  
 </**div**>  
 -----------------------  
 </**div**>  
 </**div**>  
 {% **endfor** %}  
 </**div**>  
</**div**>  
{% **endblock** *container* %}

msg\_add.html

该页面是添加留言的页面，要判断输入的内容不为空，还要把表单内容传回views.py以便存储到数据库中。

{% **block** *container* %}  
 <**script** *type=***"text/javascript"**>  
 **function** *verify*(){  
 //获取form标签元素  
 **var** form=***document***.getElementById(**'form'**);  
 //获取form下元素下所有input标签  
 **var** inputArray=form.getElementsByTagName(**"input"**);  
 **var** textareaArray=form.getElementsByTagName(**"textarea"**);  
 **var** inputArrayLength=inputArray.length;  
 //循环input元素数组  
 **for**(**var** int=**0**;int<inputArrayLength;int++){  
 //判断每个input元素的值是否为空  
 **if**(inputArray[int].value==**null** ||textareaArray[int].value==**null** ){  
 *alert*(**'输入的内容不能为空'**);  
 **return false**;  
 }  
 }  
 //如果所有Input标签的值都不为空的话  
 **return true**;  
 }  
 </**script**>  
 <**form** *action=***"."** *method=***"post"** *class=***"smart-green"** *onsubmit=***"return** *verify*();**"** *id=***"form"**>  
 <**h1**>留言信息</**h1**>  
 <**div** *class=***"error-msg"**>  
 <**label** *for=***"title"**>留言标题</**label**>  
 <**input** *id=***"title"** *type=***"text"** *name=***"title"** *class=***"error"** *placeholder=***"请输入您的标题"**/>  
 </**div**>  
 <**div** *class=***"error-msg"**>  
 <**label** *for=***"content"**>留言内容</**label**>  
 <**input** *id=***"content"** *type=***"text"** *name=***"content"** *class=***"error"** *placeholder=***"请输入您的留言"**/>  
 </**div**>  
 <**label**>  
 <**span**>**&nbsp;**</**span**>  
 <**input** *type=***"submit"** *class=***"button"** *value=***"提交"**/>  
 </**label**>  
 {#避免提交表单时出现CSRF验证失败. 请求被中断.的403错误#}  
 {% **csrf\_token** %}  
 </**form**>  
{% **endblock** *container* %}

msg\_detail.html

该页面显示留言详情，使用在views.py 中定义的queryset的名称message

{% **block** *container* %}  
 <**div** *class=***"row"**>  
 <**h1** *class=***"col-12 mt-4 mb-4"**>留言标题：{{ *message*.*title* }}</**h1**>  
 <**a** *href=***"**{% **url 'chat:chat\_to'** *message*.*user*.*username*%}**"**>留言人：{{ *message*.*user* }}</**a**>  
 <**br**>  
 留言时间：{{ *message*.*datetime* }}  
 </**div**>  
 <!-- 留言正文 -->  
 <**div** *class=***"col-12"**>  
 <**p**>{{ *message*.*content*|*safe* }}</**p**>  
 </**div**>  
 </**div**>  
{% **endblock** *container* %}

my\_msg\_list\_page.html

该页面是目前登录用户的所有留言，使用views.py中定义的以目前登录用户为参数的queryset.

{% **block** *container* %}  
 <**div** *class=***"row mt-2"**>  
 {% **for** *my\_msg* **in** *my\_msgs* %}  
 <**div** *class=***"col-4 mb-4"**>  
 <**div** *class=***"card h-100"**>  
 <**h4** *class=***"card-header"**>{{*my\_msg*.*title* }}</**h4**>  
{# <a href="{% url 'message:user\_msg\_list' message.user%}">留言人：{{ message.user }}</a>#}  
 <**div** *class=***"card-body"**>  
 <**p** *class=***"card-text"**>{{ *my\_msg*.*content*|*slice*:**'30'** }}...</**p**>  
 </**div**>  
 <**div** *class=***"card-footer"**>  
 <**a** *href=***"**{% **url 'message:msg\_detail'** *my\_msg*.*id* %}**"** *class=***"btn btn-primary"**>查看详情</**a**>  
 </**div**>  
 ----------------------------------------------------------------------------  
 </**div**>  
 </**div**>  
 {% **endfor** %}  
 </**div**>  
{% **endblock** *container* %}

user\_msg\_list\_page.html

该页面显示某个用户的所有留言，使用views.py中定义的queryset，用for循环取出该用户的所有留言

{% **block** *container* %}  
 <**div** *class=***"row mt-2"**>  
 {% **for** *user\_msg* **in** *user\_msgs* %}  
 <**div** *class=***"col-4 mb-4"**>  
 <**div** *class=***"card h-100"**>  
 <**h4** *class=***"card-header"**>{{*user\_msg*.*title* }}</**h4**>  
 <**a** *href=***"**{% **url 'message:user\_msg\_list'** *user\_msg*.*user*%}**"**>留言人：{{ *user\_msg*.*user* }}</**a**>  
 <**div** *class=***"card-body"**>  
 <**p** *class=***"card-text"**>{{ *user\_msg*.*content*|*slice*:**'30'** }}...</**p**>  
 </**div**>  
 <**div** *class=***"card-footer"**>  
 <**a** *href=***"**{% **url 'message:msg\_detail'** *user\_msg*.*id* %}**"** *class=***"btn btn-primary"**>查看详情</**a**>  
 </**div**>  
 --------------------------------------------------------------------------------------------  
 </**div**>  
 </**div**>  
 {% **endfor** %}  
 </**div**>  
{% **endblock** *container* %}

系统消息的实现与意见反馈类似，因此不再重复。

# 基于channels websocket 实现点对点聊天：

# 为什么要使用Channels

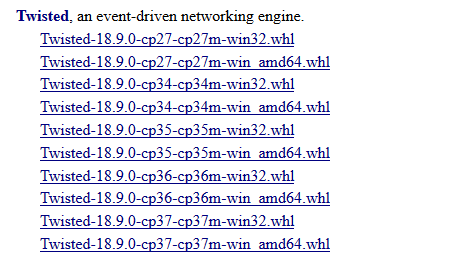
　　在Django中，默认使用的是HTTP通信，不过这种通信方式有个很大的缺陷，就是不能很好的支持实时通信。如果硬是要使用HTTP做实时通信的话只能在客户端进行轮询了，不过这样做的开销太大了。  
　　因此，在1.9版本之后，Django实现了对Channels的支持，他所使用的是WebSocket通信，解决了实时通信的问题，而且在使用WebSocket进行通信的同时依旧能够支持HTTP通信。

(windows)

服务器端安装redis(<https://blog.csdn.net/sinat_29699167/article/details/79699200>)

如果直接pip会失败。

安装channels前要安装Twisted: <https://www.lfd.uci.edu/~gohlke/pythonlibs/#lxml>



按版本选择。

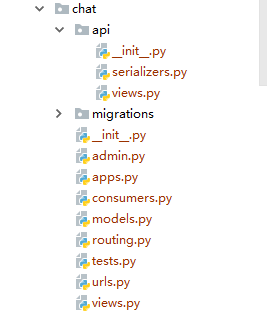
pip install channels,channels-redis,pypiwin32

pip install djangorestframework,markdown

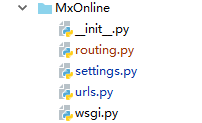
注册app：



文件结构：



在MxOnline中添加routing.py



在MxOnline/setting.py中添加如下代码, 使用redis作为消息传输通道层：

#设置channels  
ASGI\_APPLICATION = **"MxOnline.routing.application"**CHANNEL\_LAYERS = {  
 **'default'**: {  
 **'BACKEND'**: **'channels\_redis.core.RedisChannelLayer'**,  
   
 **'CONFIG'**: {  
 **"hosts"**: [(**'localhost'**, **6379**)],  
 },  
 },

chat/models.py:

定义用户聊天组、聊天记录的存储。用一张单独的表来存储每个聊天组中的两个用户会方便查询，因为一对多查询比多对多查询要简单些。

**class** chat\_group(models.Model):  
 #聊天组的名称，由两个用户名组成  
 name = models.CharField(max\_length=**500**)  
 #更新时间，如果消息不为空，以最后一条消息为准  
 #如果消息为空，则为创建时间  
 updated\_at = models.DateTimeField(auto\_now=**True**)  
 # members = models.ManyToManyField(UserProfile,related\_name='usergroup')  
 # owner = models.ForeignKey(UserProfile, on\_delete=models.CASCADE,default=None)  
  
 **def** \_\_str\_\_(self):  
 **return** self.name

**class** chat\_msgs(models.Model):  
 #显示是哪个组的消息  
 group = models.ForeignKey(chat\_group, on\_delete=models.CASCADE)  
 #显示是哪个用户发的消息  
 user = models.ForeignKey(UserProfile, on\_delete=models.CASCADE)  
 #文本信息  
 text\_msg = models.TextField()  
 # 图片信息，upload\_to='upload'表示用户上传数据存储的位置，  
 # 这里需要注意：在数据库中实际保存的并不是文件，而是文件存放的路径  
 image = models.ImageField(null=**True**, blank=**True**, upload\_to=**'image'**)  
 created\_at = models.DateTimeField(auto\_now\_add=**True**)  
  
**class** relationship(models.Model):  
 #用户拥有的聊天组  
 chat = models.ForeignKey(chat\_group, on\_delete=models.CASCADE)  
 #用户的id  
 user1 = models.ForeignKey(UserProfile, on\_delete=models.CASCADE, related\_name=**'user1'**)  
 #聊天组的id  
 user2 = models.ForeignKey(UserProfile, on\_delete=models.CASCADE, related\_name=**'user2'**)  
 #创建的时间  
 created\_at = models.DateTimeField(auto\_now\_add=**True**)

接下来是用rest-framework将消息记录序列化方便查询。

chat/api/serializers.py

该文件是用来序列化聊天记录的

**from** rest\_framework **import** serializers  
  
**from** ..models **import** chat\_msgs  
#序列化聊天消息  
  
**class** ChatMsgSerializer(serializers.ModelSerializer):  
 text\_msg = serializers.CharField(default=**None**)  
 created\_at = serializers.DateTimeField(format=**"%H:%M:%S %Y/%m/%d"**)  
 #获取发布者姓名  
 publisher = serializers.SerializerMethodField()  
  
 **def** get\_publisher(self, *obj*):  
 **return "{} {}"**.format(*obj*.user.username, *obj*.user.nick\_name)  
  
 **class** Meta:  
 model = chat\_msgs  
 fields = (**'id'**, **'user\_id'**, **'group\_id'**, **'text\_msg'**, **'created\_at'**,**'publisher'**)

chat/api/views.py

该文件让序列化的聊天记录得以被调用

**from** rest\_framework.views **import** APIView  
**from** django.http **import** JsonResponse  
  
**from** ..models **import** chat\_msgs  
**from** .serializers **import** ChatMsgSerializer  
  
  
**class** ListChats(APIView):  
 """  
 View to list all users in the system.  
 """  
  
 **def** get(self, request, format=**None**, *group\_id*=**None**):  
 """  
 Return a list of all chats by a chat group  
 """  
 queryset = chat\_msgs.objects.filter(group\_id=*group\_id*)  
 queryset = queryset.order\_by(**'-created\_at'**)  
 queryset = queryset.select\_related(**'user'**, **'group'**)  
 queryset = queryset[:**50**]  
  
 serializer = ChatMsgSerializer(queryset, many=**True**)  
  
 **return** JsonResponse(serializer.data[::-**1**], safe=**False**)

chat/consumers.py

consumers其实相当于views,也是用来处理函数的，我们需要在这里处理需要channels发送的聊天组和在聊天组中获取和显示的聊天记录。

**import** json  
**from** channels.generic.websocket **import** AsyncWebsocketConsumer  
**from** chat.models **import** chat\_group,chat\_msgs  
  
  
**class** ChatConsumer(AsyncWebsocketConsumer):  
 # 连接到channels  
 **async def** connect(self):  
 # if self.user and not self.user.is\_authenticated:  
 # return  
 #获取当前用户  
 self.user = self.scope[**'user'**]  
 #获取当前聊天组名  
 self.chat\_group\_name = self.scope[**'url\_route'**][**'kwargs'**][**'chat\_group\_name'**]  
 #获取能被channels处理的组名  
 self.group\_name = **'chat\_{}'**.format(self.chat\_group\_name)  
  
 #指定加入到聊天的通道层  
 **await** self.channel\_layer.group\_add(  
 self.group\_name,  
 self.channel\_name  
 )  
  
 **await** self.accept()  
  
 **async def** disconnect(self, *code*):  
 **if** self.user **and not** self.user.is\_authenticated:  
 **return** #断开连接  
 **await** self.channel\_layer.group\_discard(  
 self.group\_name,  
 self.channel\_name  
 )  
  
 **async def** receive(self, *text\_data*):  
 #接收文字聊天消息  
 **if** self.user **and not** self.user.is\_authenticated:  
 **return** text\_data\_json = json.loads(*text\_data*)  
 text\_msg = text\_data\_json[**'text\_msg'**]  
  
 #处理用户名  
 nickname = **"{} {}"**.format(self.user.username, self.user.nick\_name)  
 **if** nickname ==**" "**:  
 nickname = **"--"** #异常处理  
 **try**:  
 #获取当前所在的聊天组  
 chat = chat\_group.objects.get(name=self.chat\_group\_name)  
  
 **except** chat\_group.DoesNotExist:  
 **return** #文字消息存到数据库  
  
 chat\_msg\_object = chat\_msgs.objects.create(user\_id=self.user.id, text\_msg=text\_msg, group\_id=chat.id)  
 #发送时间  
 created\_at = chat\_msg\_object.created\_at.strftime(**'%H:%M:%S %Y/%m/%d'**)  
  
 #channels向该组成员发送以下信息  
 **await** self.channel\_layer.group\_send(  
 self.group\_name,  
 {  
 **'type'**: **"chat\_msg"**,  
 **'text\_msg'**: text\_msg,  
 **'user\_id'**: self.user.id,  
 **'publisher'**: nickname,  
 **'created\_at'**: created\_at,  
 }  
 )  
  
 **async def** chat\_msg(self,*event*):  
 #向channels发送消息  
 **if** self.user **and not** self.user.is\_authenticated:  
 **return** user\_id = *event*[**'user\_id'**]  
 text\_msg = *event*[**'text\_msg'**]  
 created\_at = *event*[**'created\_at'**]  
 publisher = *event*[**'publisher'**]  
  
 **await** self.send(text\_data=json.dumps({  
 **'user\_id'**: user\_id,  
 **'created\_at'**: created\_at,  
 **'text\_msg'**: text\_msg,  
 **'publisher'**:publisher,  
 }))

chat/routing.py

这个文件相当于urls.py,引导channels找到对应的consumers

**from** django.urls **import** path  
**from** . **import** consumers  
  
websocket\_urlpatterns = [  
 path(**'ws/chat/<str:chat\_group\_name>/'**, consumers.ChatConsumer),  
]

chat/views.py

当用户在帖子详情点击会话时，先判断当前登录用户与发帖用户是否有过聊天，如果有，则直接显示聊天，如果没有，则创建聊天。

**def** add\_relationship(*request*,*username*):  
 **if not** *request*.user.is\_authenticated:  
 redirect(**'login'**)  
 #获取当前用户  
 user = UserProfile.objects.get(id=*request*.user.id)  
 # 获取另一个用户  
 other\_user =get\_object\_or\_404(UserProfile,username=*username*)  
 **if** user!= other\_user:  
 question = relationship.objects.filter(Q(user1\_id=user.id, user2\_id=other\_user.id) | Q(user2\_id=user.id, user1\_id=user.id))  
 **if** question.exists():  
 **return** redirect(**'chat:allchats'**)  
 **else**:  
 str = random.choice(**'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'**)  
 new\_group = chat\_group.objects.create(name=str)  
 new\_group.save()  
 new\_relationship = relationship()  
 new\_relationship.user1 = user  
 new\_relationship.user2 = other\_user  
 new\_relationship.chat = new\_group  
 new\_relationship.save()  
 **return** render(*request*,**'chat/chat\_base.html'**)  
 **else**:  
 **return** HttpResponse(**"请选择其他用户进行聊天"**)

获取当前登录用户拥有的所有聊天并显示在前端。通过关系表获取与当前登录用户处于同一个聊天组的用户，使用其用户名作为聊天组的名称

**def** chats(*request*):  
 user= *request*.user  
 allchats = relationship.objects.filter(Q(user1\_id=user.id)|Q(user2\_id=user.id))  
 left = allchats.filter(user1=user)  
 right = allchats.filter(user2=user)  
 context = {  
 **'left'**: left,  
 **'right'**: right  
 }  
 **return** render(*request*, **'chat/chat\_base.html'**, context)

chat/urls.py

urlpatterns = [  
 re\_path(**r'^chat\_to/([\w\W]\*)/$'**, views.add\_relationship, name=**'chat\_to'**),  
 path(**'allfriends/<int:group\_id>'**, api\_views.ListChats.as\_view(), name=**'get-chat\_groups-api'**),  
 path(**'allchats/'**, views.chats, name=**'allchats'**),

]

MxOnline/urls.py增加如下语句：

path(**'chat/'**, include(**'chat.urls'**, namespace=**'chat'**)),

MxOnline/routing.py增加如下语句：

**from** channels.auth **import** AuthMiddlewareStack  
**from** channels.routing **import** ProtocolTypeRouter, URLRouter  
**import** chat.routing  
  
# application = ProtocolTypeRouter({  
# Empty for now (http->django views is added by default)  
# })  
  
application = ProtocolTypeRouter({  
 # (http->django views is added by default)  
 **'websocket'**: AuthMiddlewareStack(  
 URLRouter(  
 chat.routing.websocket\_urlpatterns  
 )  
 ),  
})

chat\_base.html:

该页面左侧显示当前登录用户的所有聊天，右侧显示聊天详情

{% **block** *container* %}  
  
 <**div** *class=***"messaging"**>  
 <**div** *class=***"inbox\_msg"**>  
 <**div** *class=***"inbox\_people"**>  
  
 <**div** *class=***"headind\_srch"**>  
 <**div** *class=***"recent\_heading"**>  
 <**h4**>最近聊天</**h4**>  
 </**div**>  
 </**div**>  
  
 <**div** *class=***"inbox\_chat"**>  
 {% **for** *chat* **in** *left* %}  
 <**div** *class=***"chat\_list"** *id=***"chat\_list"** *data-chat-id=***"**{{ *chat*.*chat\_id* }}**"** *data-chat-name=***"**{{ *chat*.*chat*.*name* }}**"**>  
 <**div** *class=***"chat\_people"**>  
 <**div** *class=***"chat\_img"**>  
 <**img** *src=***"/media/**{{ *chat*.*user2*.*image* }}**"** *alt=***"sunil"**/>  
 </**div**>  
 <**div** *class=***"chat\_ib"**>  
 <**h5**>{{ *chat*.*user2*.*username* }}</**h5**>  
 </**div**>  
 </**div**>  
 </**div**>  
 {% **endfor** %}  
 {% **for** *chat* **in** *right* %}  
 <**div** *class=***"chat\_list"** *id=***"chat\_list"** *data-chat-id=***"**{{ *chat*.*chat\_id* }}**"** *data-chat-name=***"**{{ *chat*.*chat*.*name* }}**"**>  
 <**div** *class=***"chat\_people"**>  
 <**div** *class=***"chat\_img"**>  
 <**img** *src=***"/media/**{{ *chat*.*user1*.*image* }}**"** *alt=***"sunil"**/>  
 </**div**>  
 <**div** *class=***"chat\_ib"**>  
 <**h5**>{{ *chat*.*user1*.*username* }}</**h5**>  
 </**div**>  
 </**div**>  
 </**div**>  
 {% **endfor** %}  
 </**div**>  
 </**div**>  
 <**div** *class=***"mesgs"**>  
  
 <**div** *id=***"messages-list"** *class=***"msg\_history"**>  
 {% verbatim %}  
 <**div** *v-for=***"item in items"**>  
 <**div** *class=***"outgoing\_msg"** *v-if=***"item.own"**>  
 <**div** *class=***"sent\_msg"**>  
 <**p**>{{ item.text\_msg }}</**p**>  
 <**span** *class=***"time\_date"**>{{ item.publisher}} || {{ item.created\_at }}</**span**>  
 </**div**>  
 </**div**>  
  
 <**div** *class=***"incoming\_msg"** *v-if=***"!item.own"**>  
 <**div** *class=***"received\_msg"**>  
 <**div** *class=***"received\_withd\_msg"**>  
 <**p**>{{ item.text\_msg }}</**p**>  
 <**span** *class=***"time\_date"**>{{ item.publisher}} || {{ item.created\_at }}</**span**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 {% endverbatim %}  
 </**div**>  
 <**div** *class=***"type\_msg"**>  
  
 <**div** *class=***"input\_msg\_write"**>  
 <**input** *disabled=***""** *type=***"text"** *id=***"message-text"** *class=***"write\_msg"** *placeholder=***"选择聊天"**/>  
 <**button** *id=***"chat-message-submit"** *class=***"msg\_send\_btn"** *type=***"button"**>  
 <**i** *class=***"fab fa-telegram-plane"**></**i**>  
 </**button**>  
 </**div**>  
 </**div**>  
  
 </**div**>  
 </**div**>  
  
 </**div**>  
  
 <**input** *type=***"hidden"** *id=***"user-id"** *value=***"**{{ *request*.*user*.*id* }}**"**>  
 <**input** *type=***"hidden"** *id=***"MxOnline-bot"** *value=***"**{{ *MxOnline\_bot* }}**"**>  
  
{% **endblock** %}  
{% **block** *js* %}  
 <**script** *src=***"https://cdn.jsdelivr.net/npm/vue/dist/vue.js"**></**script**>  
 <**script** *src=***"https://unpkg.com/axios/dist/axios.min.js"**></**script**>  
  
 <**script** *src=***"**{% **static 'chat/js/messages-send.js'** %}**?v=1.0.0"**></**script**>  
 <**script** *src=***"**{% **static 'chat/js/web-socket.js'** %}**?v=1.0.0"**></**script**>  
 <**script** *src=***"**{% **static 'chat/js/room.js'** %}**?v=1.0.0"**></**script**>  
  
{% **endblock** %}

messages-send.js

**function** *initMessageInputTextActions*() {  
 ***document***.querySelector(**'#message-text'**).focus();  
 ***document***.querySelector(**'#message-text'**).onkeyup = **function** (e) {  
 **if** (e.keyCode === **13**) {  
 ***document***.querySelector(**'#chat-message-submit'**).click();  
 }  
 };  
}

room.js

该文件显示聊天的具体内容

*initActions*();  
**let *currentRealtimeChatSocket*** = **null**;  
**let *currentRealtimeChatBotSocket*** = **null**;  
**let *messagesList*** = **new** Vue({  
 el: **'#messages-list'**,  
 data: {  
 items: []  
 },  
});  
  
**function** *initActions*() {  
 *initMessageInputTextActions*();  
 *initChatsActions*();  
}  
  
**function** *initChatsActions*() {  
 $(**'.chat\_list'**).click(**function** () {  
 **let** chatId = $(**this**).data(**'chat-id'**);  
 **let** chatName = $(**this**).data(**'chat-name'**);  
  
 ***messagesList***.items = [];  
  
 **let** chatSocket = **new *WebSocket***(**'ws://'** + ***window***.location.host + **'/ws/chat/'** + chatName + **'/'**);  
 **let** chatBotSocket = **new *WebSocket***(**'ws://'** + $(**'#MxOnline-bot'**).val() + **'/ws/chat/bot/'** + chatName + **'/'**);  
  
 **if** (***currentRealtimeChatSocket***) {  
 ***currentRealtimeChatSocket***.close();  
 }  
  
 **if** (***currentRealtimeChatBotSocket***) {  
 ***currentRealtimeChatBotSocket***.close();  
 }  
  
 ***currentRealtimeChatSocket*** = chatSocket;  
 ***currentRealtimeChatBotSocket*** = chatBotSocket;  
  
 *initChatSocketActions*(chatSocket,chatBotSocket);  
  
 $(**".chat\_list"**).removeClass(**'active\_chat'**);  
 **this**.classList.add(**'active\_chat'**);  
  
 **let** messageTextDOM = $(**'#message-text'**);  
  
 messageTextDOM.removeAttr(**'disabled'**);  
 messageTextDOM.attr(**'placeholder'**, **"Write a message"**);  
  
 *updateChatMsg*(chatId);  
 });  
}  
  
**function** *updateChatMsg*(chatId) {  
 **let** userId = ***document***.getElementById(**'user-id'**).value.toString();  
 **let** newData = [];  
  
 axios  
 .get(**`/chat/allfriends/**${chatId}**`**)  
 .then(response => {  
 $.each(response.data, **function** (index, value) {  
 newData.push({  
 text\_msg: value.text\_msg,  
 user\_id: value.user\_id,  
 created\_at: value.created\_at,  
 own: *parseInt*(userId) === *parseInt*(value.user\_id),  
 })  
 });  
 });  
  
 ***messagesList***.items = newData;  
  
}  
  
$(**'#messages-list'**).bind(**"DOMSubtreeModified"**, **function** () {  
 **try** {  
 ***document***.getElementById(**"messages-list"**).lastChild.scrollIntoView();  
 }  
 **catch** (e) {  
 }  
});  
  
***document***.addEventListener(**'DOMContentLoaded'**, **function** () {  
 **if** (!***Notification***) {  
 *alert*(**'请打开he!p通知.'**);  
 **return**;  
 }  
  
 **if** (***Notification***.permission !== **"granted"**) {  
 ***Notification***.*requestPermission*();  
 }  
});

web-socket.js

该文件创建websocket，当表单提交时，我们就通过WebSocket发送数据（而不是 POST数据），然后需要将回调函数与WebSocket上的新数据接收事件对接起来，并且给用户发送通知。

**function** *initChatSocketActions*(chatSocket, chatBotSocket) {  
 **let** currentUserId = ***document***.getElementById(**'user-id'**).value.toString();  
  
 chatSocket.onmessage = **function** (e) {  
 *addNewMessage*(e)  
 };  
  
 chatBotSocket.onmessage = **function**(e){  
 *addNewMessage*()  
 };  
  
 // 点击发送时的一系列动作  
 ***document***.querySelector(**'#chat-message-submit'**).onclick = **function** (e) {  
 **let** messageInputDom = $(**'#message-text'**);  
 **let** text\_msg = messageInputDom.val();  
 **if** (text\_msg) {  
 **if** (text\_msg === **'/stock=APPL'**) {  
 chatBotSocket.send(***JSON***.stringify({  
 **'user\_id'**: currentUserId,  
 }))  
 }  
 **else** {  
 chatSocket.send(***JSON***.stringify({  
 **'text\_msg'**: text\_msg,  
 }));  
 }  
 messageInputDom.val(**''**);  
 }  
 };  
}  
  
**function** *addNewMessage*(e) {  
 **let** data = ***JSON***.parse(e.data);  
 **let** currentUserId = ***document***.getElementById(**'user-id'**).value.toString();  
  
 **let** isOwn = *parseInt*(currentUserId) === *parseInt*(data[**'user\_id'**]);  
  
 ***messagesList***.items.push({  
 text\_msg: data[**'text\_msg'**],  
 user\_id: data[**'user\_id'**],  
 created\_at: data[**'created\_at'**],  
 publisher: data[**'username'**],  
 image:data[**'image'**],  
 own: isOwn,  
 });  
 **if** (!isOwn) {  
 *sendNotification*()  
 }  
}  
  
  
**function** *sendNotification*() {  
 ***console***.log(**"Sending alert"**)  
  
 **if** (***Notification***.permission !== **"granted"**)  
 ***Notification***.*requestPermission*();  
 **else** {  
 ***console***.log(**"sent alert"**)  
 **new *Notification***(**'Realtime Chat Notification'**, {  
 icon: **'http://cdn.sstatic.net/stackexchange/img/logos/so/so-icon.png'**,  
 body: **"你在he！p有新消息哦"**,  
 });  
 }  
}