该文档是基于django 4.0以上版本，来编写的

1. **安装需要的库**

pip install channels daphne

1. **修改settings.py**

在settings.py的INSTALLED\_APPS中添加:

'daphne',

在settings.py中添加:

ASGI\_APPLICATION = 'WebSocket.asgi.application'，

红色部分为项目名称，需要修改

INSTALLED\_APPS = [

    'daphne',  #注册daphne组件，在channels4.0开始，注册组件使用daphne,一定放在开头

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]

#添加该选项，表示django以asgi的模式启动

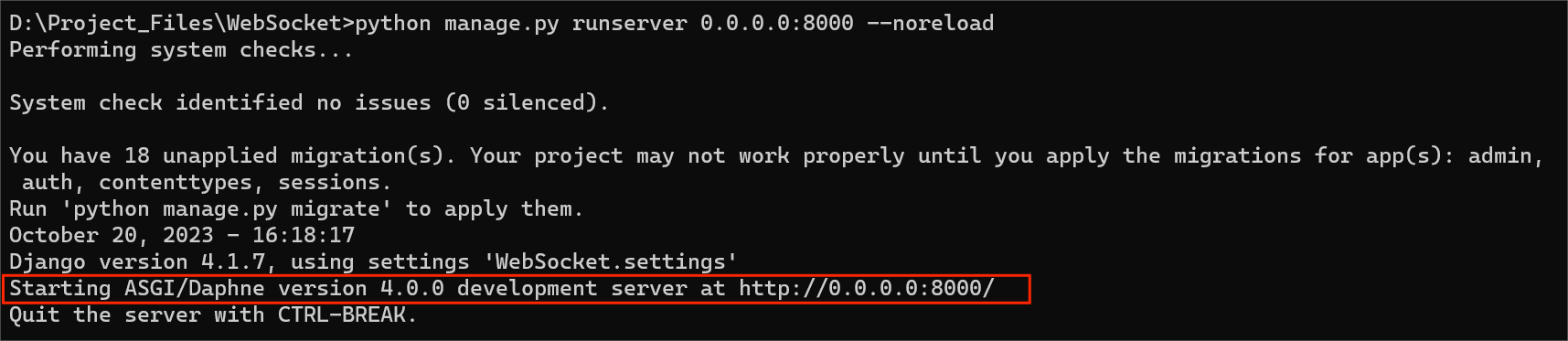
ASGI\_APPLICATION = 'WebSocket.asgi.application'

[wsgi](https://so.csdn.net/so/search?q=wsgi&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/system_post/article/details/_blank)：django默认启动模式，wsgi只支持同步处理请求，即http请求

asgi：该模式下支持wsgi，并且新增了[异步请求](https://so.csdn.net/so/search?q=%E5%BC%82%E6%AD%A5%E8%AF%B7%E6%B1%82&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/system_post/article/details/_blank)和webSocket

以上两步设置完成，启动服务可以看到

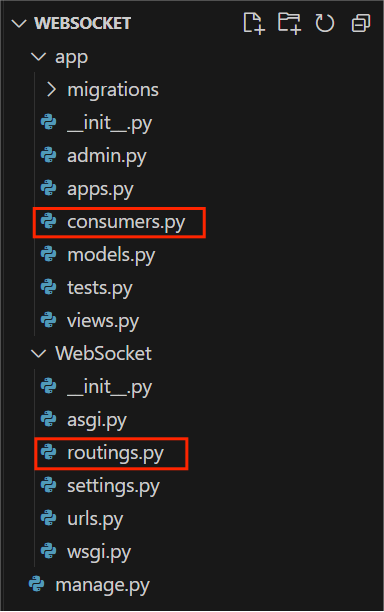
Starting ASGI/Daphne version 4.0.0 development server at <http://0.0.0.0:8000/>

**

证明asgi和daphne配置完成，接下来编写业务逻辑

1. **业务逻辑开发前准备**

*在指定目录下创建routings.py、consumers.py*

**

1. **业务逻辑**

创建consumers.py

类似于视图views.py，创建用于socket处理的类

from channels.exceptions import StopConsumer

from channels.generic.websocket import WebsocketConsumer

class ChatConsumer(WebsocketConsumer):

    def websocket\_connect(self, message):

        '''

        当有客户端向后端发送websocket连接请求时，自动触发该函数

        :param message:

        :return:

        '''

        # 服务器允许客户端创建连接

        self.accept()

    def websocket\_receive(self, message):

        '''

        浏览器基于websocket向后端发送数据，自动触发接受消息，并且处理信息

        :param message:

        :return:

        '''

        # 输出消息

        print(message)

        #服务端向前端回消息

        self.send('服务器收到了你的消息：%s' % (message['text']))

    def websocket\_disconnect(self, message):

        '''

        客户端与服务端断开连接时，自动触发该函数

        :param message:

        :return:

        '''

        print('断开连接')

        raise StopConsumer()

创建routings.py

与urls.py功能类似，当接受到客户发来的socket请求时，使用哪个类socket类进行处理

from django.urls import  re\_path, path

from app.consumers import ChatConsumer

websocket\_urlpatterns = [

path('room/',ChatConsumer.as\_asgi()),

    #也可以使用正则路径,这种方式用在群组通信当中

    re\_path('ws/chat/(?P<group>\w+)/$',ChatConsumer.as\_asgi())

]

修改asgi.py

添加 "websocket": URLRouter(routings.websocket\_urlpatterns) 以支持webSocket

import os

from django.core.asgi import get\_asgi\_application

from channels.routing import ProtocolTypeRouter, URLRouter

from . import routings

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'WebSocket.settings')

# 支持http请求和websocket请求

application = ProtocolTypeRouter({

    "http": get\_asgi\_application(),

    "websocket": URLRouter(routings.websocket\_urlpatterns)

})

1. **测试**

使用浏览器的console进行简单测试，浏览器按F12，在console逐条输入以下语句，注意路径需要修改

#创建websocket

var socket = new WebSocket('ws://127.0.0.1:8000/room/')

#定义收到服务器消息之后如何处理

socket.onmessage = function(e){console.log(e.data)}

#向服务器发送消息

socket.send('你好')

测试的结果展示



1. **异步服务**

以上方法为同步webSocket的收发，想要建立连接后循环推送数据给web端，则要采用异步的方式

首先在consumers.py中创建类的时候要继承AsyncWebsocketConsumer基类，其次就是在函数前加async，调用函数时加await，asyncio是Python异步协程库，不太懂，还需要再研究研究

from channels.exceptions import StopConsumer

from channels.generic.websocket import AsyncWebsocketConsumer

import asyncio

class VehicleState(AsyncWebsocketConsumer):

    async def websocket\_connect(self, message):

        '''

        当有客户端向后端发送websocket连接请求时，自动触发该函数

        :param message:

        :return:

        '''

        # 服务器允许客户端创建连接

        await self.accept()

        asyncio.ensure\_future(self.send\_data\_loop())

    async def websocket\_receive(self, message):

        '''

        浏览器基于websocket向后端发送数据，自动触发接受消息，并且处理信息

        :param message:

        :return:

        '''

        # 输出消息

        print(message)

        #服务端向前端回消息

        await self.send('服务器收到了你的消息：%s' % (message))

    async def websocket\_disconnect(self, message):

        '''

        客户端与服务端断开连接时，自动触发该函数

        :param message:

        :return:

        '''

        print('断开连接')

        self.isSend = False

        raise StopConsumer()

    async def send\_data\_loop(self):

        self.isSend = True

        while self.isSend: # 使循环在断开连接后跳出

            print("推送数据")

            await self.send('推送数据')

            await asyncio.sleep(5)

具体为啥还需学习，总之这样写就可以实现功能