开发文档-后端

开发文档-后端

- Django 框架
  - 前后端交互服务
    - 模型
    - **路由**
    - 视图
  - mqtt服务
- CORS处理

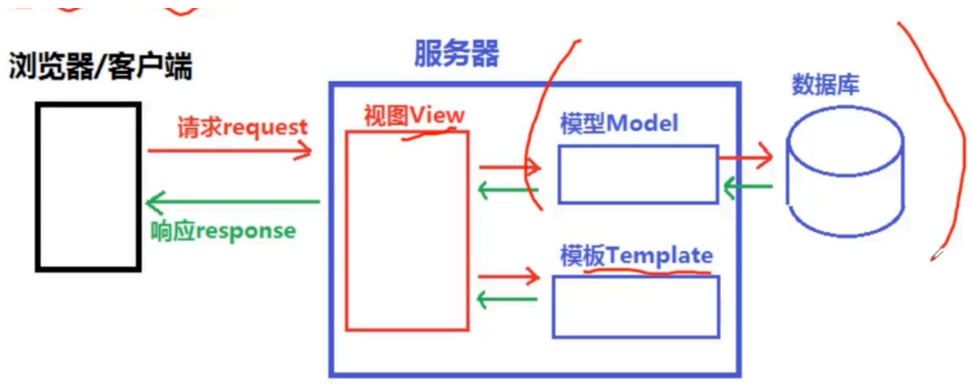
Django 框架

前后端交互服务

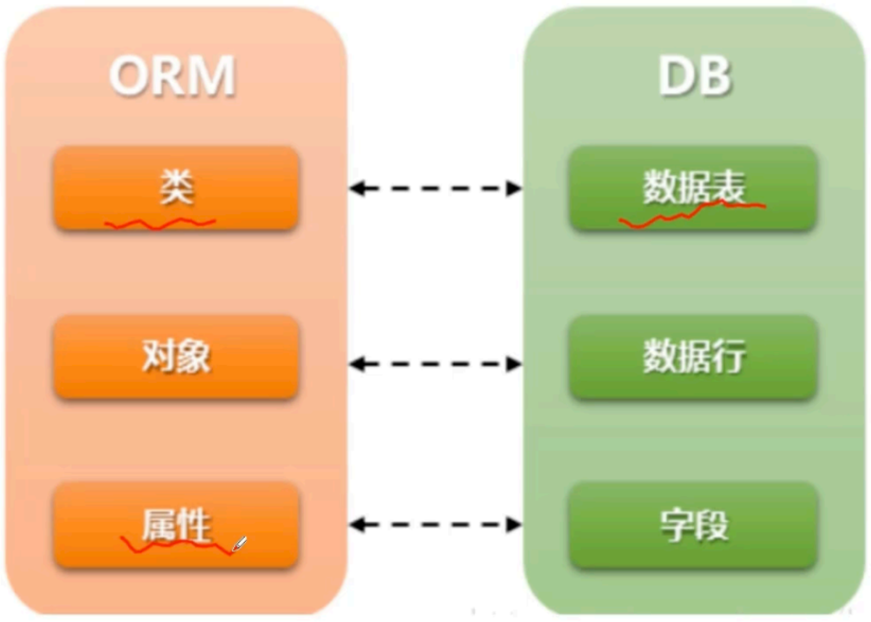
针对给前端的响应给django开启三个app，分别是

```
uesr
device
message
```

分别用来处理登陆信息，设备信息，和设备消息。



- **安装 mysqlclient [版本 mysqlclient 1.3.13以上 ， 官网目前为1.4.x]**
- **安装前确认ubuntu是否已安装 python3-dev 和 default-libmysqlclient-dev**
  1. sudo apt list --installed|grep -E 'libmysqlclient-dev|python3-dev'
  2. 若命令无输出则需要安装 - sudo apt-get install python3-dev default-libmysqlclient-dev
- **sudo pip3 install mysqlclient**



模型

django中的 model对应着sql数据库中的表，每一个类都会被映射到一张甚至多张表，类的属性就定义了表的属性和表之间的关联。

如针对设备制定的Device类：

```
from django.db import models

from user.models import CustomUser


# Create your models here.
class Device(models.Model):
    id = models.AutoField(primary_key=True)
    user = models.ForeignKey(CustomUser, on_delete=models.CASCADE,
related_name='devices')
    device_id = models.CharField(max_length=100)
    device_name = models.CharField(max_length=255)
    device_location = models.CharField(max_length=255)
    device_type = models.CharField(max_length=100)
    device_status = models.CharField(max_length=10, choices=[('on', 'On'), ('off',
'Off')])
    is_online = models.BooleanField(default=False)  # 新添加的字段

    class Meta:
        unique_together = ('user', 'device_id')

    def __str__(self):
        return (f"Device(user_id={self.user.user_id}, device_id={self.device_id},
device_name={self.device_name}, "
```

```
                f"device_location={self.device_location}, device_type=
{self.device_type}, "
                f"device_status={self.device_status}, is_online={self.is_online})")
```

路由

django使用路由分发的方式来确定服务的响应模式，简单来说，我们在主路由中可以匹配第一级的url，然后根据第一级的url的情况再把路由分配到不同的app中的子路由处理。

主路由中：

```
from django.urls import path
from . import views

urlpatterns = [
    path('add', views.add),
    path('get', views.get),
    path('modify', views.modify),
    path('get_message', views.get_message)
]
```

子路由中（举例）：

```
from django.urls import path
from . import views

urlpatterns = [
    path('post', views.post),
    path('location_list', views.location_list),
    path('location_history', views.location_history)
]
```

视图

前面的urls中有两个参数，第二个参数是views中的函数。这是因为views虽然叫做视图函数，但它其实是用来处理和对应url相关的所有服务的。下面以请求设备列表举例。

```
@csrf_exempt
@require_GET
def get(request):
    # 从URL参数中获取user_id
    user_id = request.GET.get('user_id')
    try:
        # 检查用户是否存在
        try:
            user = CustomUser.objects.get(user_id=user_id)
        except CustomUser.DoesNotExist:
            return JsonResponse({'status': 'error', 'message': '当前用户不存在'})

        # 获取用户的所有设备
        devices = Device.objects.filter(user=user)

        # 构造设备列表
        device_list = [
            {
                'id': device.device_id,
                'name': device.device_name,
                'location': device.device_location,
                'type': device.device_type,
                'status': device.device_status,
                'is_online': device.is_online
            }
            for device in devices
        ]

        return JsonResponse({'status': 'success', 'devices': device_list})

    except Exception as e:
        return JsonResponse({'status': 'error', 'message': str(e)})
```

mqtt服务

针对跟mqtt服务器的交互再开开启一个app，`mqtt` 用来处理mqtt相关的业务。这里主要用到的库是`paho.mqtt`。后端每次启动的时候我们开启一个客户端，并且连接我们的服务器，订阅需要的主题，并且设置接收到业务相关信息的时候的相关处理逻辑。这样在mqtt服务器有相关的信息发布时，后端就可以及时作出响应。

```
import json

import paho.mqtt.client as mqtt
from django.apps import apps

from IotManager import settings
from device.models import Device
from message.models import Message


def on_connect(mqtt_client, userdata, flags, rc):
    if rc == 0:
        print('Connected successfully')
        mqtt_client.subscribe('$SYS/brokers/+/metrics/bytes/received')
        mqtt_client.subscribe('$SYS/brokers/+/metrics/bytes/sent')
        mqtt_client.subscribe('iot/#')  # 订阅主题
    else:
        print('Bad connection. Code:', rc)
```

```python
def receive_message(mqtt_client, topic, device_id, payload):
    try:
        data = json.loads(payload)
        device_id = data.get('device_id')
        message_type = data.get('type')
        if not device_id or not message_type:
            mqtt_client.publish('response/'+topic, '请提供设备id和类型')

        valid_types = dict(Message.MESSAGE_TYPES).keys()
        if message_type not in valid_types:
            mqtt_client.publish('response/'+topic, "无效的消息类型")
            return

        if message_type == 'location':
            latitude = data.get('latitude')
            longitude = data.get('longitude')
            if not latitude or not longitude:
                mqtt_client.publish('response/' + topic, '请提供经纬度内容')
                return
            text = json.dumps({
                'latitude': latitude,
                'longitude': longitude
            })
        else:
            text = data.get('text')
        if not text:
            mqtt_client.publish('response/' + topic, '请提供消息内容')

        device = Device.objects.get(device_id=device_id)
        message = Message.objects.create(device=device, type=message_type, text=text)

        mqtt_client.publish('response/'+topic,f'消息成功接收')
    except json.JSONDecodeError as e:
        mqtt_client.publish('response/'+topic,f'JSON 格式错误 {e}')
    except Device.DoesNotExist:
        mqtt_client.publish('response/'+topic, f'设备 {device_id} 不存在')


def on_message(mqtt_client, userdata, msg):
    # print(f'Received message on topic: {msg.topic} with payload: {msg.payload}')
    if msg.topic.startswith('iot/'):
        try:
            device_id = msg.topic.split('/')[1]
            receive_message(mqtt_client, msg.topic, device_id, msg.payload)
        except IndexError:
            mqtt_client.publish('response/'+msg.topic, f'Error: 未提供设备id -
{msg.topic}')
```

```
    elif msg.topic.endswith('/bytes/received'):
        mqtt_app_config = apps.get_app_config('mqtt')
        mqtt_app_config.received_bytes = int(msg.payload.decode('utf-8'))
        # print('received', mqtt_app_config.received_bytes)
    elif msg.topic.endswith('/bytes/sent'):
        mqtt_app_config = apps.get_app_config('mqtt')
        mqtt_app_config.sent_bytes = int(msg.payload.decode('utf-8'))
        # print('sent', mqtt_app_config.sent_bytes)


def connect_to_mqtt():
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.username_pw_set(settings.MQTT_USER, settings.MQTT_PASSWORD)
    client.connect(
        host=settings.MQTT_SERVER,
        port=settings.MQTT_PORT,
        keepalive=settings.MQTT_KEEPALIVE
    )
    return client
```

CORS处理

前后端刚开始联调的时候必然会遇到的一个问题就是CORS。CORS（Cross-Origin Resource Sharing）是一种用于在Web浏览器中处理跨域资源访问的机制。在同源策略（Same-Origin Policy）的限制下，Web页面只能从相同的源（协议、域名和端口）加载资源，而CORS允许服务器声明哪些源被允许访问其资源。简单来说，如果前后端在不同的ip或者不同的端口运行，他们之间不允许进行直接访问，这就会导致我们的前后端无法联通。

在本项目中从后端解决此问题（指的是在开发中，上线到正式服务器的时候不应该这样做），主要是使用了 `corsheaders` 包，在setting.py中设置了允许cors的ip和端口。

```
# setting.py
CORS_ALLOWED_ORIGINS = [
    "http://localhost:5173",
    "http://10.162.58.80"
]

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',
    'user',
    'device',
    'message',
    'mqtt',
]
```