实验步骤

1. 运行数控系统 MQTT 虚拟服务，作为数据源用于系统测试，等同于实体产线中边缘服务器上运行的服务。MQTT 服务器是数据来源，在开发系统测试连接时，需要一直保持运行。

（1）在……/emqx-5.0.11-windows-amd64\bin 路径中输入 cmd，如图 1，启动运行，输入命令：emqx start，如图 2，运行 emqx。

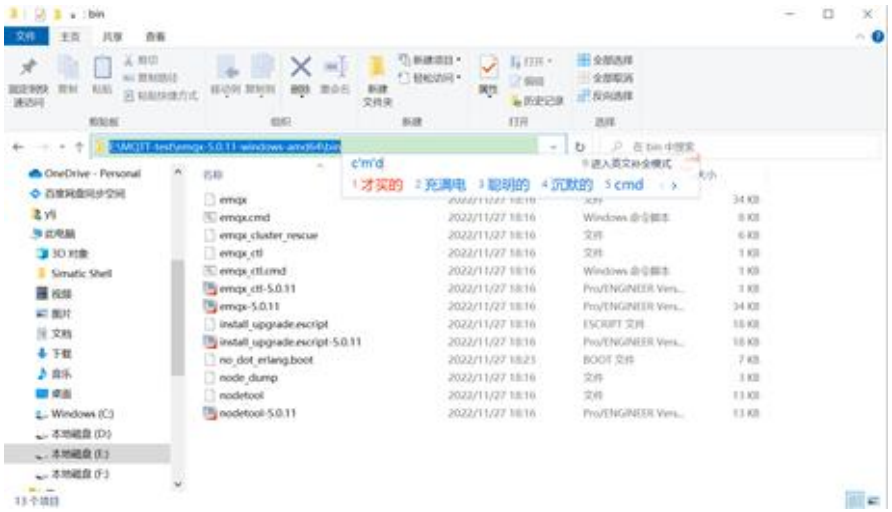


图 1 启动运行界面

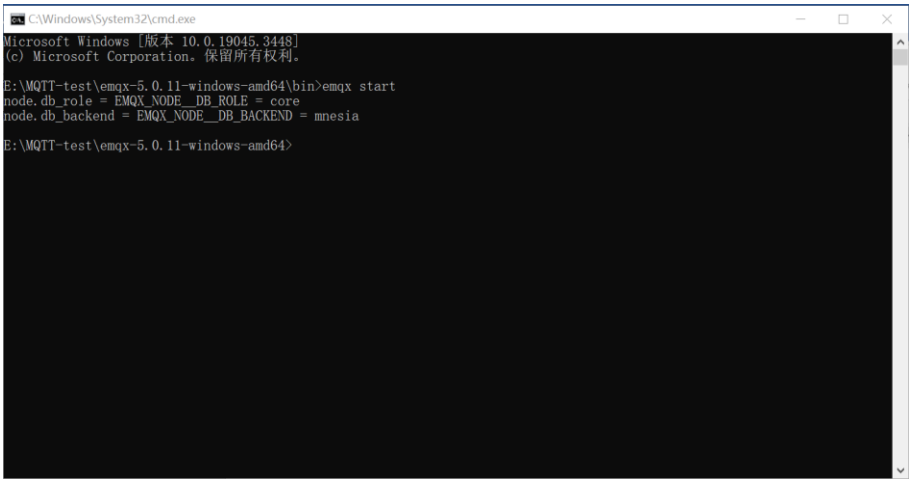


图 2 运行 emqx

（2）在……/StressTest-INC-Cloud 路径下输入 cmd，启动运行，运行命令：StressTest-INC-Cloud -n 1 -b 0，如图 3，生成一个数控系统虚拟的 MQTT 服务，n 后面的 1 表示 1 个数控系统，b 后面的 0 指序号从 0 开始。

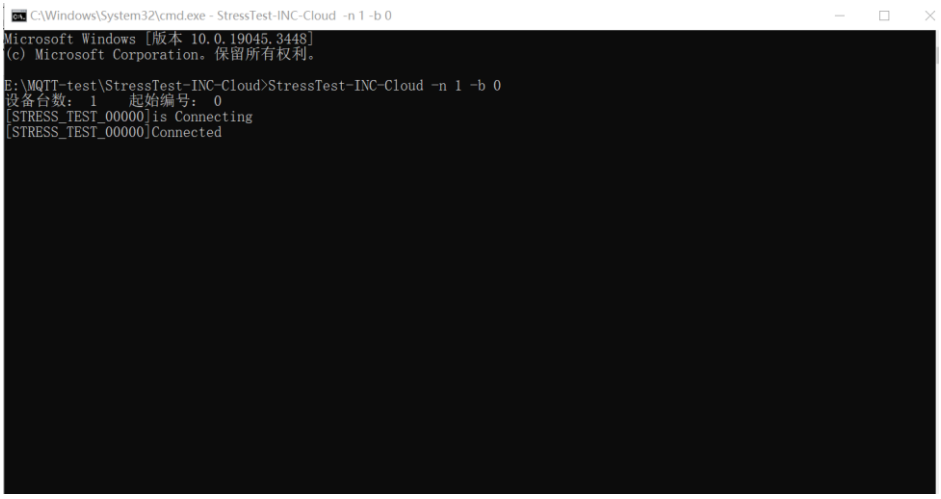


图 3 运行数控系统 MQTT 服务

2. MQTTX 数据订阅测试指令

安装 MQTTX 软件，用于测试 MQTT 服务是否有效。IP 地址为 127.0.0.1，端口号为 1883。指令含义可查询 Nc-link 协议的数据类型和查询方式。

（1）查看订阅查看整个服务的数据：

订阅响应：Probe/Query/Response/STRESS_TEST_00000

发布请求指令：Probe/Query/Request/STRESS_TEST_00000　指令字：“”

查看 json 文件的网址：**https://www.bejson.com/jsonviewernew/**

（2）订阅某一 id 对应的数据

订阅响应：Query/Response/STRESS_TEST_00000

发布请求指令：Query/Request/STRESS_TEST_00000　指令参数：{"ids":[{"id":"0103502101"}]}

MQTTX 仅用来测试 MQTT 服务是否正常，测试正常后即可关闭。

3．新建开发项目

以**管理员身份**运行命令提示符，运行命令：vue ui，

图 4

在弹出的网页中的 vue 项目管理器中，设置好项目存储路径 D:\，点击在此创建新项目，如图 5，设置项目名称，点击下一步，如图 6，然后下一步，选择默认 vue2，点击创建项目，如图 7，项目创建成功后的可视化面板如图 8。

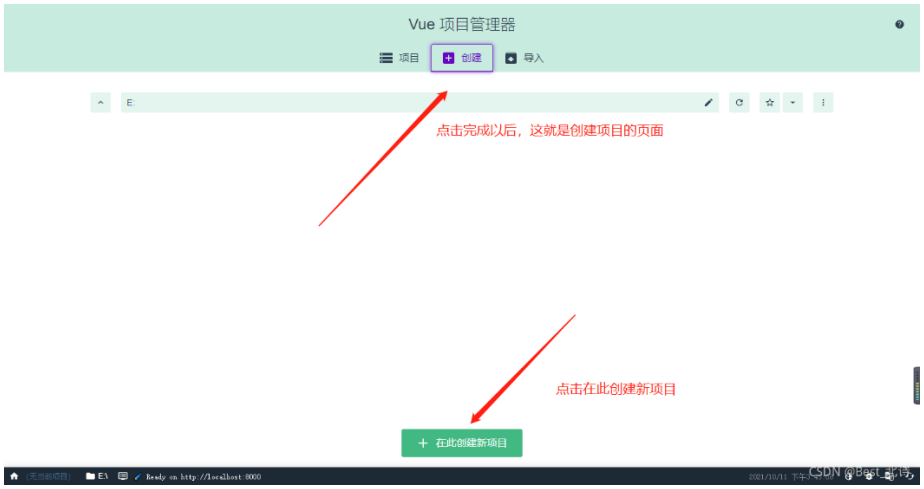

Vue 项目管理器
项目
创建
导入
点击完成以后，这就是创建项目的页面
点击在此创建新项目
在此创建新项目

图 5


创建新项目
详情
预设
功能
配置
项目文件夹
project1019
包管理器
默认
更多选项
Git
取消
下一步

图 6


http://localhost:8000/project/create
创建新项目
详情
预设
功能
配置
选择一套预设
Default (Vue 3)
Default (Vue 2)
手动
远程预设
上一步
创建项目

图 7

图 8 创建项目成功可视化面板

4. 运行项目，界面优化设计

在提供的项目文件的路径上输入 cmd 回车，然后运行命令 npm run serve 如图 9、图 10。在浏览器中打开相应网址 http://localhost:8081/，即可看到提供的项目对应的网页界面，如图 11，分析界面已有功能，进一步优化设计。

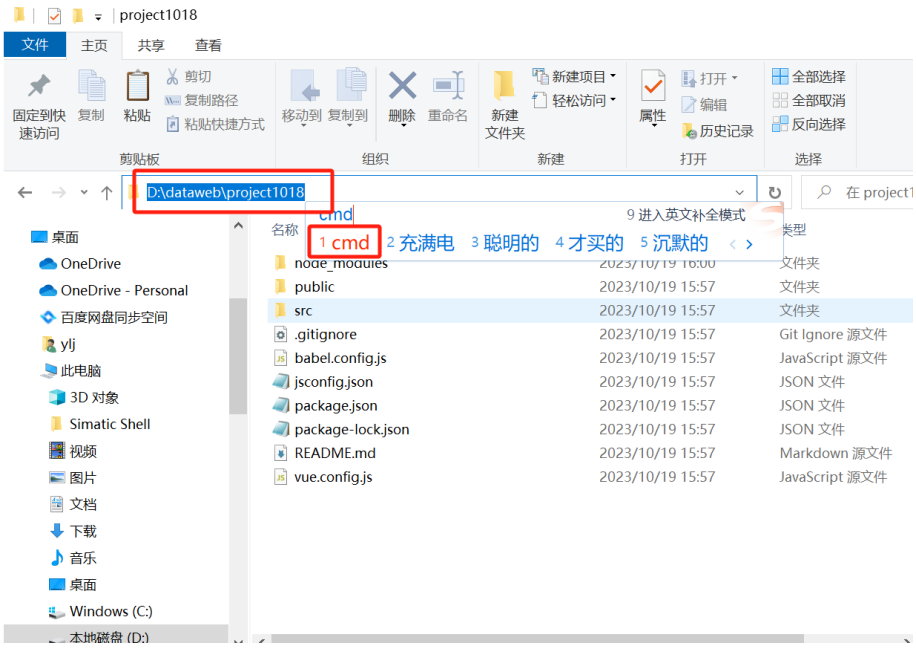


图 9 启动运行

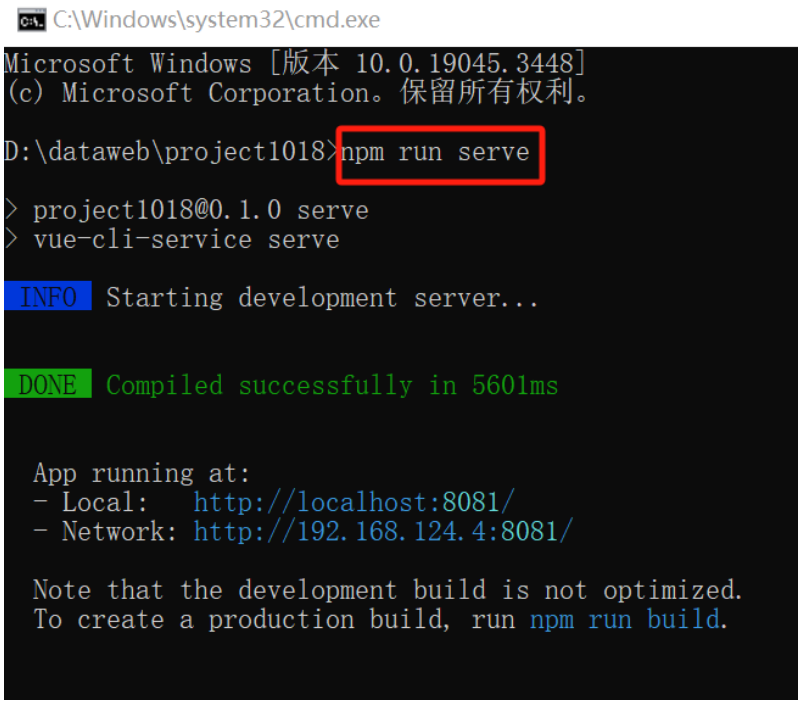


图 10 运行项目服务

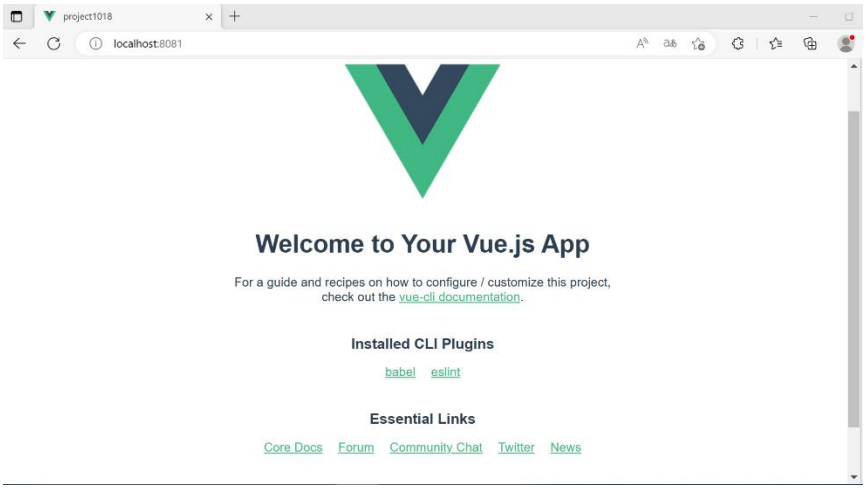


图 11 基本界面

4. 系统开发

启动 Visual Studio Code，点击 file-open folder，选择项目文件夹，如图 12。

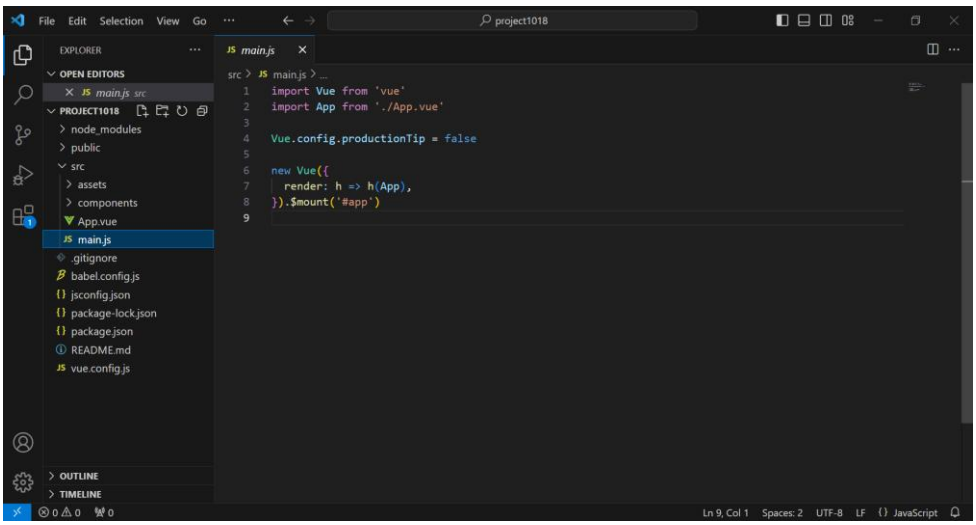


图 12 程序打开界面

（1）大数据采集与分析系统框架设计与开发

在项目文件的 compoments 文件夹中，新建 CoreSource.vue、ShowData.vue、DataProcee.vue 文件，新建 pyt 文件夹和 backend.py 文件。基于 VUE 的整个项目文件如图 13 所示，src 文件夹是项目主要部分，components 中.vue 文件是前端界面程序，pyt 文件夹是根据需要添加的后端 python 程序文件，APP.vue 做页面管理，.js 文件是项目的主要信息配置文件，重点是前端界面.vue 程序和后端 python 程序。

.vue 文件主要包括三部分，如图 14 所示，<template>……</template>主要是界面元素的设置，<script>……</script>主要导入相关库，定义变量和函数等，<style>……</style>用来设置界面样式。

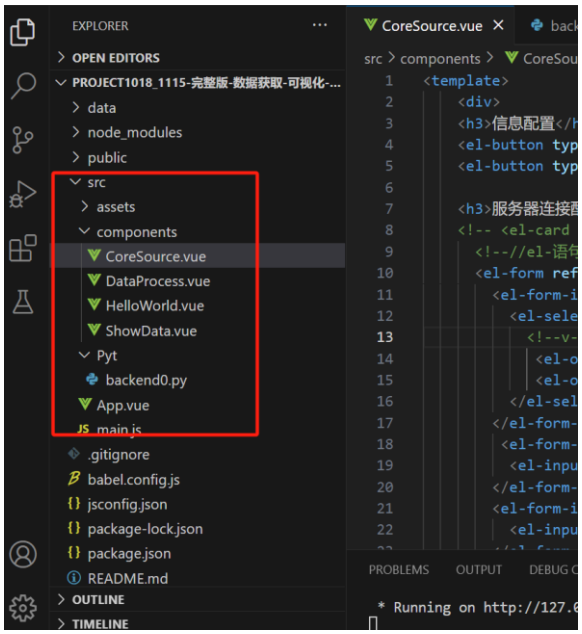


图 13 项目文件

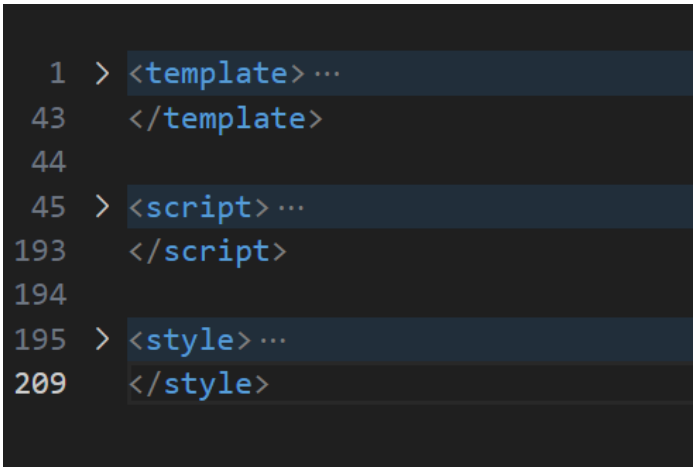


图 14 vue 文件的构成

1）建立页面，添加页面元素

界面元素设计采用 Element ui 库，调用方法：在 main.js 中导入库，即可在其他 Vue 界面中使用 Element ui 库设计界面，详细可查看参考链接网站中的组件。

参考链接：https://element.eleme.cn/#/zh-CN/component/quickstart

```
import ElementUI from 'element-ui';//引入 element UI
import "element-ui/lib/theme-chalk/index.css";//引入 element ui  样式
Vue.use(ElementUI);//全局使用 element ui
```

举例：页面切换按钮，代码如下，<el-button>设置按钮元素，按钮类型为"primary"，按钮名称为“欢迎界面”，当点击按钮时，执行函数"onSubmit"，函数功能是调用 router，将页面转至地址为“……/CoreSource”的网页。

```
<el-button type="primary" @click="onSubmit">欢迎界面</el-button>


  methods: {
      onSubmit() {
        this.$router.push({
          path: "/CoreSource"
        })
      },
```

2）页面切换：vue router

参考链接 https://router.vuejs.org/zh/guide/essentials/named-routes.html

调用方法：Main.js 中引入 Vue router

```
import VueRouter from "vue-router";
Vue.use(VueRouter)
```

引入组件，将相关的.vue 界面引入；

```
import CoreSource from './components/CoreSource.vue'
import ShowData from './components/ShowData.vue'
import HelloWorld from './components/HelloWorld.vue'
```

定义页面的路由形式，定义各个页面的网址路径等。

```
const router = new VueRouter({
  routes:[
    {
      path: "/",  //定义页面网址的路径，比如此句意思为 localhost:8080/对应网页为
HelloWorld
      component: HelloWorld
    },
    {path: "/ShowData",
    component:ShowData
    },
    {
      path: "/CoreSource",
      component: CoreSource
    }
  ]
});
export default  router;
```

注册路由：添加命令：router,

```
Vue.config.productionTip = false
new Vue({
  router,
  // 注册路由对象
  render: h => h(App),
}).$mount('#app')
```

在 App.vue 中，实现路由显示，添加命令：<router-view></router-view>

```
<template>
  <div id="app">
  <router-view></router-view>
  <!--// 路由显示 -->
  </div>
</template>
```

在各个网页.Vue 文件中通过调用 router 来实现页面切换，参考按钮案例，添加按钮的页面跳转功能。

```
<el-button type="primary" @click="onSubmit">欢迎界面</el-button>

  methods: {
    onSubmit() {
      this.$router.push({
        path: "/CoreSource"
      })
    },
```

（2）前后端通信

1）在 **vue.config.js** 中定义跨域连接信息，采用 devServer proxy 实现。

可参考：https://blog.csdn.net/qq_47443027/article/details/125985081

```
module.exports = {
  devServer: {
  proxy: {    //配置跨域
    "/api": {
      target: "http://127.0.0.1:5000",  // 此地址为后端 Python 运行的服务地址和端口
      changOrigin: true, //允许跨域
      pathRewrite: {
        "^/api": '', //请求的时候使用这个 api
      },
    },
  },
},
}
```

2）前端（vue 网页）向后端（Python）发送请求接收信息，使用组件 axios，采用 post、get 等方法实现参数传递。

可参考 https://www.runoob.com/vue2/vuejs-ajax-axios.html

在.vue 中引入 axios 组件，配置 baseURL

```
import axios from "axios";
axios.defaults.baseURL = "/api";
```

3）下面以服务器连接这部分前后端传输作为案例介绍前后端通信的实现过程。案例前端界面如图 23，功能是输入框中可输入数据相关配置信息，点击“连接”按钮时执行 onConnect 函数，onConnect 函数功能：前端将输入的信息 this.connection 发送至后端 Python，后端 Python 程序获取数据后再发送到前端。

前端获取后端的响应信号，并打印在控制台。

注意：Python 往前端传输的数据类型包括 string, dict, tuple, Response instance, or WSGI callable，数组不能直接传输，可以转换为 JSON。前端获得的响应信息在使用时要注意数据类型。

图 15 案例示例图

主要实现过程：

a. 前端配置：前端定义输入变量：connection，绑定给输入框，并作为变量传输给后端。

定义 onConnect 函数，数据传输核心代码如下，注意红字批注说明。

可参考：https://www.runoob.com/vue2/vuejs-ajax-axios.html

```
    onConnect (){
      axios.post('/connect/',{data:this.connection}).then(res=>{
        console.log(res.data)
      })
    },
```

////connect//相当于端口信号，与后端相对应，可以根据需要修改，data 后面的信息 this. connection 为向后端发送的信息，res.data 为接收的后端返回的信息，res.data.rc_status 指返回数据中名称为 rc_status 的数据。

b. 后端配置

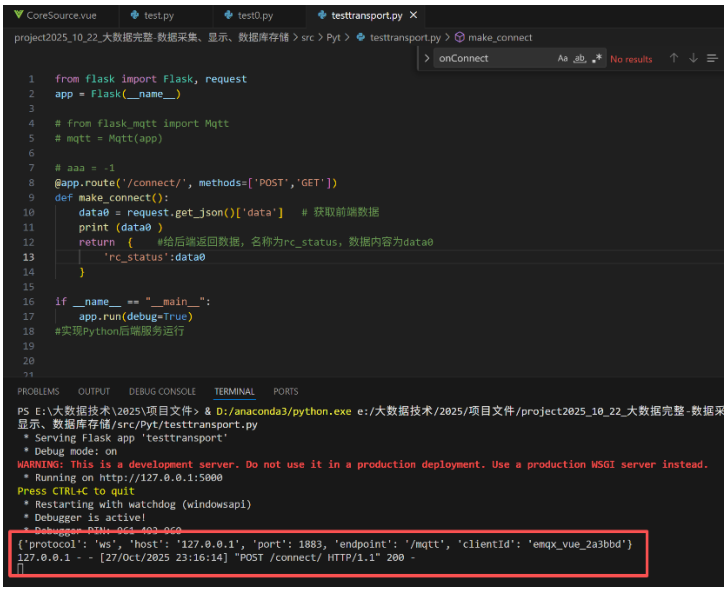
引入所需要的库，采用 flask 框架与前端通信。

可参考：https://www.cnblogs.com/huchong/p/8227606.html

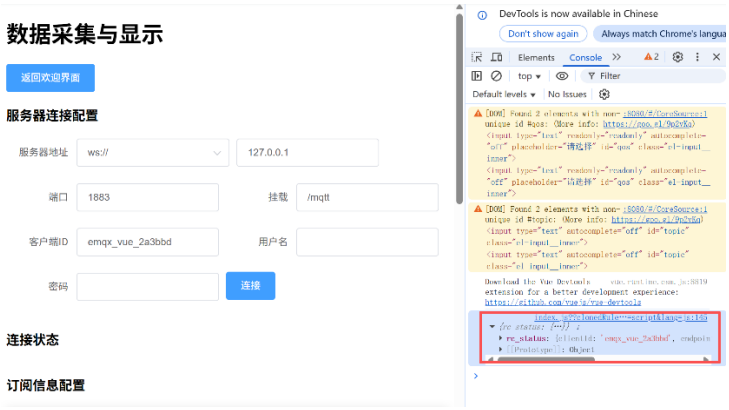
```
from flask import Flask, request
app = Flask(__name__)


@app.route('/connect/', methods=['POST','GET'])
def make_connect():
    data_connect = request.get_json()['data']
    print(data_connect)
    # 接收数据’/connect/’与前端对应接口一致，名称可以修改,’data’为接收的前端发送的信息。
    return {
     'rc_status': data_connect
     }
#return 向前端返回数据，这里是后端向前端传输 1 个数据 data_connect,名称为'rc_status'，在前端通过名称可以查询到相应数据。
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

将后端 python 程序运行起来，在网页点击“连接”按钮，python 程序控制台和网页控制台分别显示如图 66，将传输的数据打印出来。



（a）后端程序控制台



（b）前端网页控制台

图 66 数据前后端传输结果

（3）数据采集

理清 MQTT 传输数据的原理，利用 Python 编写程序，采用 flask 库与前端 MQTT 连接界面通信，读取前端界面填写的 MQTT 服务器信息、MQTT 主题信息等。调用 flask_mqtt 库连接 MQTT 服务器、订阅消息、发布消息，获取数据。

参考链接：

1）Flask_MQTT 应用资料

https://flask-mqtt.readthedocs.io/en/latest/index.html

https://flask-mqtt.readthedocs.io/en/latest/configuration.html

https://flask-mqtt.readthedocs.io/en/latest/usage.html

2） MQTT 官网：https://mqtt.org/

（4）数据可视化

要求自主设计显示，实现多个数据、多种图表样式的数据显示。下面以一个数据显示在一个图表中为例介绍。

1）设计可视化界面

建议使用数据可视化库：echarts

参考链接：https://echarts.apache.org/examples/zh/index.html

调用方法：在 main.js 中导入库

```
import * as echarts from "echarts";
Vue.prototype.$echarts = echarts;
```

在显示数据的.vue 中，<template>里定义一个图框用于显示图表

```
<el-card>
  <h3>X 轴实际速度</h3>
  <div id='x.actualspeed' class="graphic">
  </div>
</el-card>
```

在<script>下面定义 echarts

```
const echarts = require('echarts');
```

定义变量，即定义图表的样式，即两个坐标轴、曲线样式，data 为要显示的数据。

```
xactualspeed: {      //定义 echart 的样式
  xAxis: {
   type: 'category',
        },
  yAxis: {
   type: 'value'
  },
   series: [
   {
  data: this.data,
  type: 'line',
  smooth: true
    }
   ],
  title: {
  left: 'center',
```

```
      // text: 'x 轴实际速度'
       },
     },
```

定义初始化函数，图表 Id 为'x.actualspeed'，与前面定义的图框对用，Option 设置为 xactualspeed，与定义的图标样式的变量一致。

```
initCharts() {
        this.xactualspeed0 =
echarts.init(document.getElementById('x.actualspeed'))
        this.xactualspeed0.setOption(this.xactualspeed);
      },
```

在 mounted 下面初始化 echarts

```
    mounted(){
      this.initCharts();
    },
```

2）解析后端数据，并在前端界面可视化

在后端 python 程序中，解析通过 MQTT 协议获取到的 json 数据，添加与前端数据显示界面通信的接口，将数据发送至前端。

在前端.vue 中，根据需要通过定义函数，与后端通信，获取采集的数据，写入 actualspeeddata 中。然后，将数据显示在 echarts 中，实现图表的数据更新，参考代码如下，将获得数据 actualspeeddata 显示到图表上。

```
        this.xactualspeed0.setOption({
          series: [{
          data: this.actualspeeddata,
          }]
          });
});
```

（5）数据的保存与下载

利用 FileSaver 库保存数据，引入

```
import FileSaver from "file-saver";
```

（1）下载获取到的全部数据

定义一个按钮关联 download 函数，定义 download 函数

```
download() {
      const blob0 = new Blob([this.loadcurrentdata,], {
        type: "application/csv",
      });
      FileSaver.saveAs(blob0, "全部下载.csv");
    },
```

以上实现单组数据的下载，可以尝试多个数据同时下载，提示：定义数据，利用 push 将多组数据组合到一起，再下载。

（2）截取部分数据下载

定义一个变量和一个数据组

```
downloadtemp:",
savedata: [],
```

method 中定义开始保存函数和停止保存函数

```
startsave() {
        this.downloadtemp = true;
        // 开始保存将 downloadtemp 置为 1 通过该变量控制挂载的要下载的数据更新
      },
```

```
stopsave() {
        this.downloadtemp = false;
        console.log("停止截取")
        // 控制台显示停止截取
         const blob = new Blob([this.savedata], {
           type: "application/csv",
         });
        FileSaver.saveAs(blob, "截取下载.csv");
        this.savedata = [];
      },
```

在接收后端数据的函数中，添加条件数据更新，将要下载的数据保存至 savedata 中

```
if (this.downloadtemp) {
          this.savedata.push('this.x.loadcurrent');
          // 单个数据截取下载
}
```

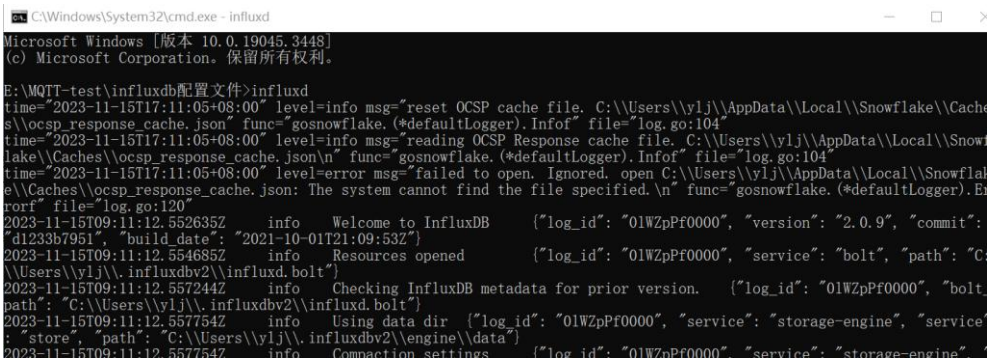
以上实现截取保存单组数据功能，可以尝试同时下载多组数据，提示：定义多个数组在后端数据的函数中存储多组数据，在 stopsave 函数中，将多个数组的数据 push 到 savedata 中。

（6）数据边缘存储

采用 Influxdb 数据库存储数据，主要包括数据库信息配置和数据存储。

1）数据配置

首先运行数据库服务，在 Influxdb 数据库配置文件夹下，运行 Influxdb 数据库本地服务



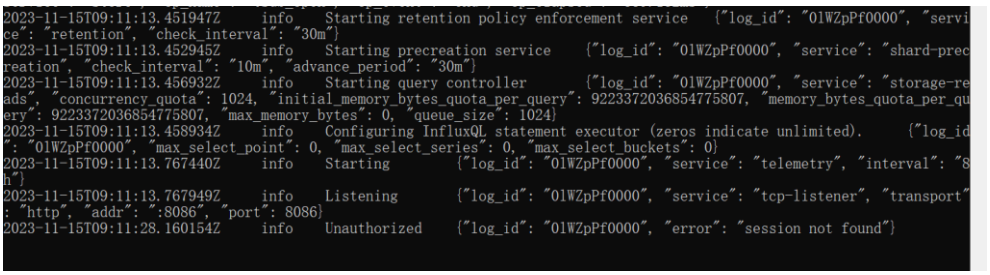


图 16 Influxdb 数据库运行服务

启动后，可以看到服务地址、端口 8086，此时可以在浏览器中，打开 http://localhost:8086，进入按指示注册账号，然后登录。

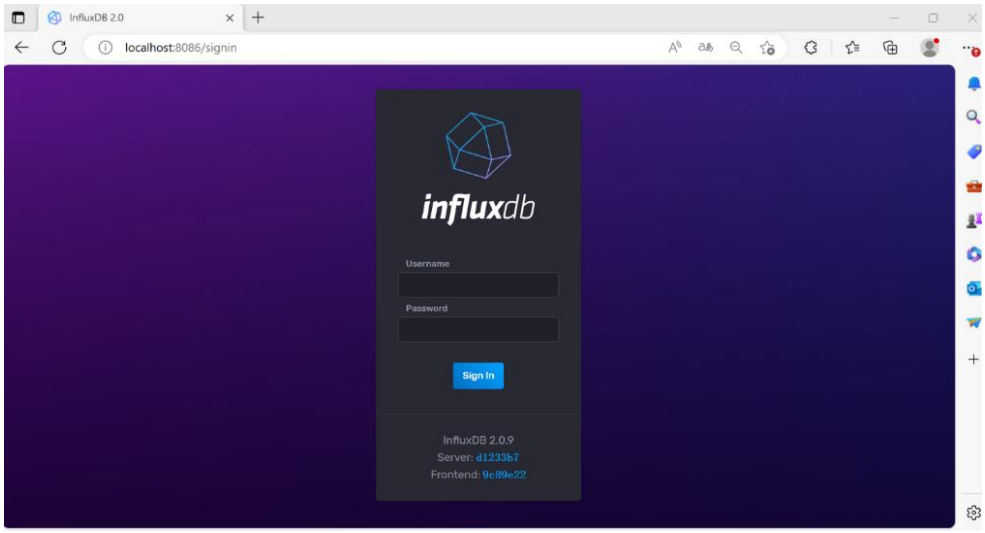


图 17 Influxdb 数据库登录

点击 buckets，在 buckets 里可以创建 bucket，用于存储数据。

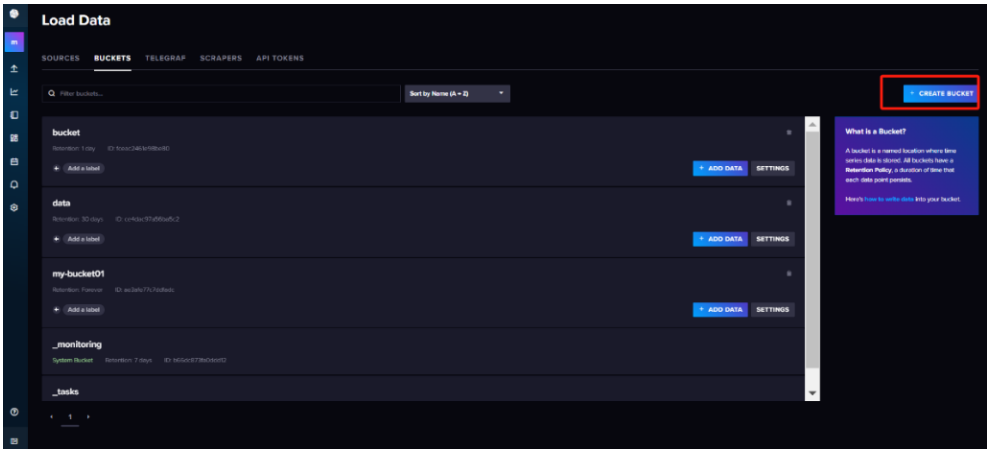


图 18 Buckets 配置

2） 数据存储接口开发

数据存储可以选择 node 或者 Python 开发，node 在.vue 中开发，Python 在后端程序中开发。以 node 为例，如图 19 中，选择 source 页面的 node 后进入图 20 页面，按照指引下方生成调用的程序。

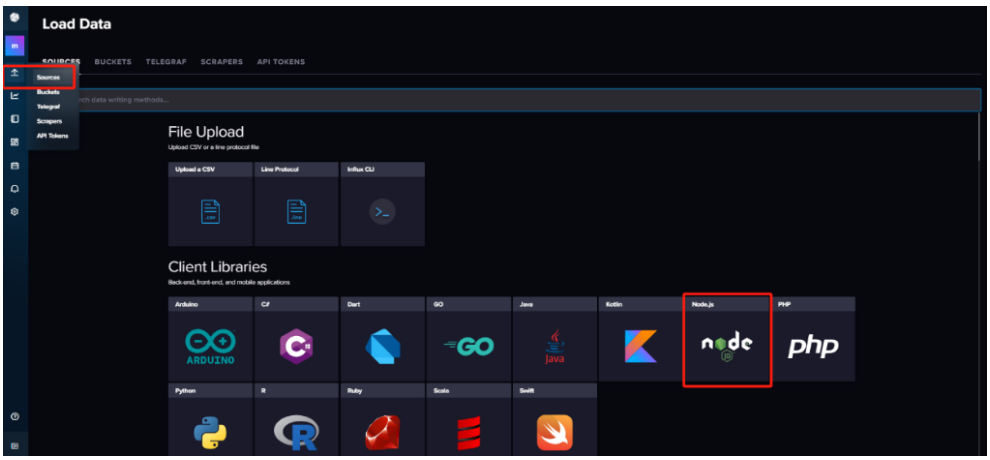


图 19 配置界面

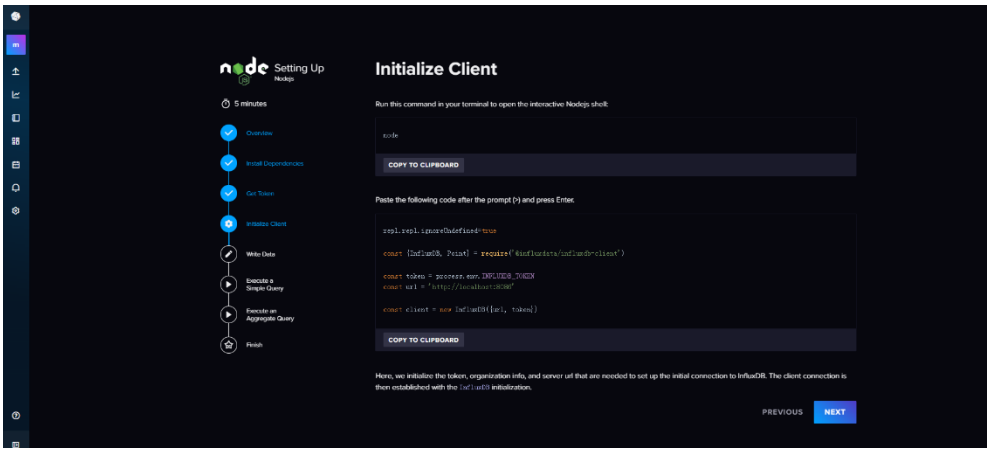


图 20 采用 node 开发数据存储

点击 data explore 可以查看上传成功的数据，如图 21。

图 21 数据库中数据查询

（7）云端 Influxdb 时序数据库存储

打开云端 Influxdb 时序数据库服务网页 https://us-east-1-1.aws.cloud2.influxdata.com，注册账号登录。点击 buckets 可以设置数据存储桶，与边缘端相同。点击 source-node 进入配置界面。根据指引可以进行编程，也可参考本地数据库配置代码，功能相同。在 Dataexplorer 中可以查看上传的数据。

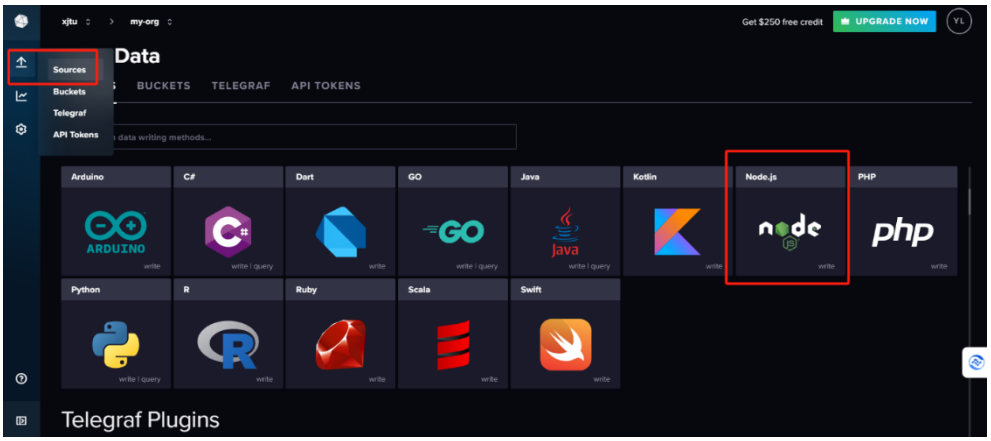


图 22

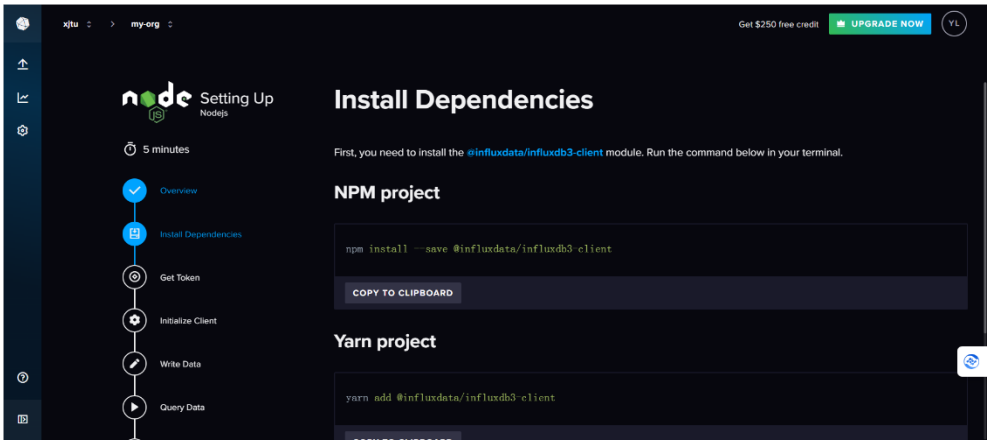
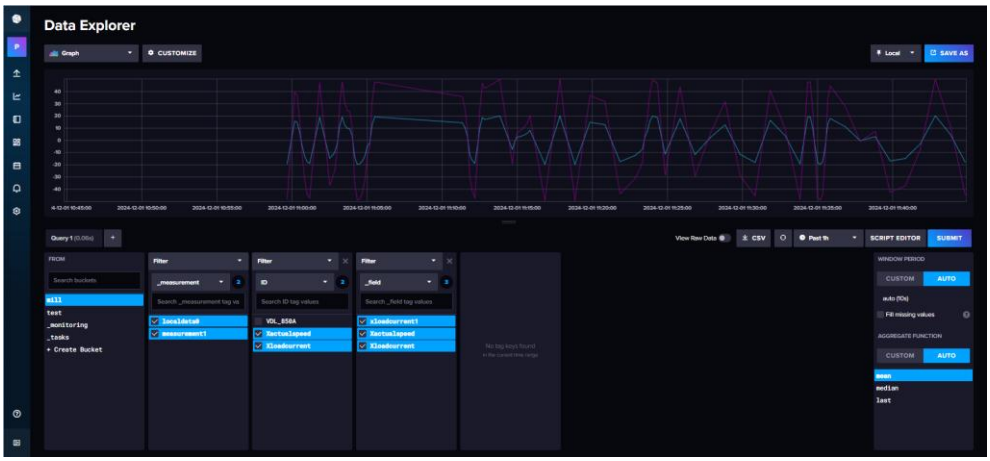


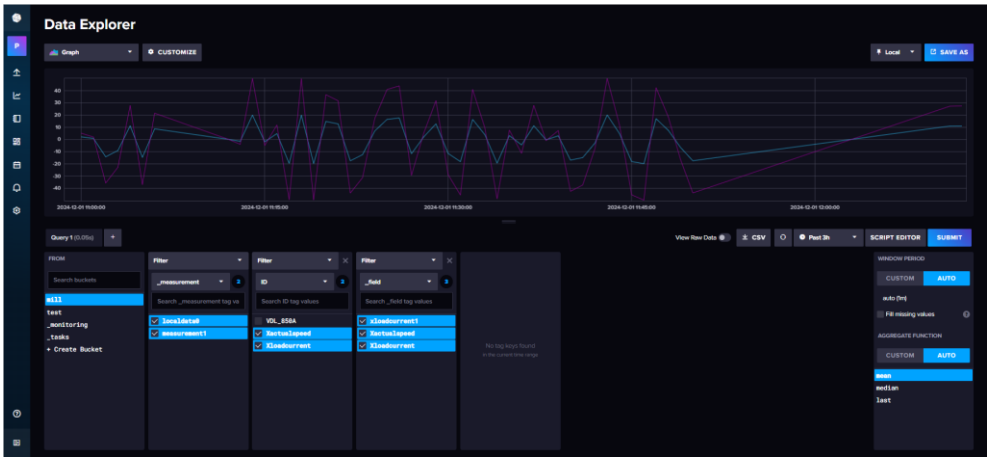
图 23 云端数据存储开发

图 24 云端数据查询

前端和后端同时将数据写入的对比：蓝色为前端写入，玫红色为后端写入，可以看出前端写入有一定延迟和丢数据。

通过后端程序同时写入 2 个数据





（8）MySQL 数据库进行数据存储

1）创建 MySQL 数据库

首先确保已安装 MySQL 服务，已知安装过程中设置服务密码。启动 navicat premium，点击连接，选择 MySQL，如图 27，输入连接名，可以自定义，输入服务密码，点击测试连接显示连接成功，如图 28。确定后，双击连接，图标变绿色如图 29，即可右击新建数据库，如图 30。然后，打开数据库，右击表，可以添加列表，再列表中输入信息，点击保存，弹窗中输入列表名称，如图 31。在查询中可以测试数据的读写。

图 27 创建 MySQL 连接

图 28 配置 MySQL 服务信息

图 29 连接 MySQL

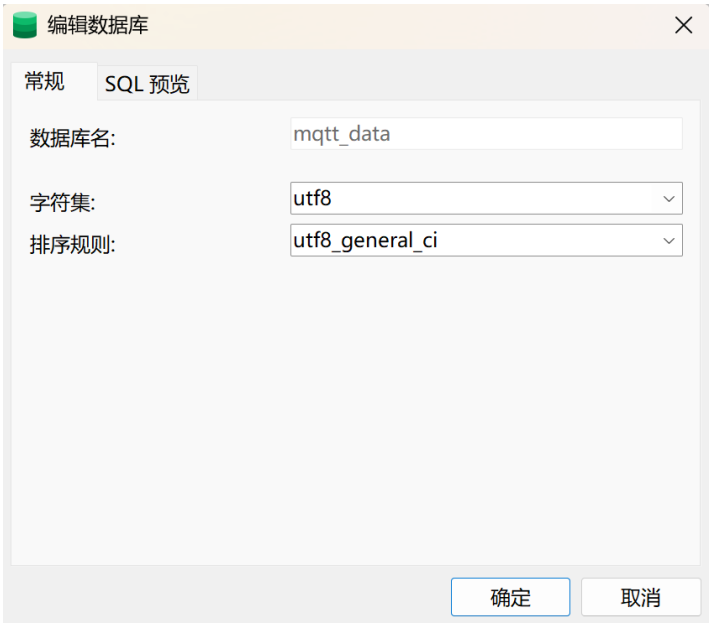


图 30 创建数据库

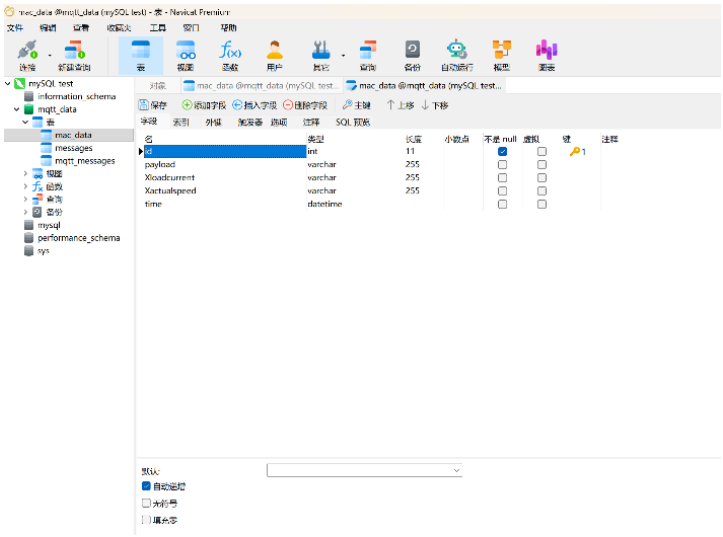


图 31 创建列表

2）编程实现 MySQL 数据库读写

利用 python 编写程序，实现数据写入数据库和从数据库中删除查询。

参考案例：以下案例是连接到 MySQL 数据库，将固定的数值 dat 和 time（系统时间）写入数据库 mac_data 表格的 payload 和 time 中。

```
# 增加、修改、删除表中元素
import pymysql
import datetime
try:
    # 建立连接
    conn = pymysql.connect(
        host = 'localhost',#默认 localhost
        user = 'root',#默认 root
        password = 'xjtu2025',#无默认
        database = 'mqtt_data',
        port = 3306,#默认 3306
        charset = 'utf8'
    )
    # 创建游标对象
    cur = conn.cursor()
    # 定义写入数据库的变量，time 为当前系统时间
    time= datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    dat = 0

    # sql 语句，数据库读写
    sql = "insert into mac_data(payload, time)
values('%s','%s')" %(dat,time)
    cur.execute(sql)
    # 提交操作（对数据库有修改时需要）
    conn.commit()
except Exception as e:
    print(e)
finally:
    # 关闭游标对象和连接
    cur.close()
    conn.close()
```