

阶段性总结报告：大数据采集与云边存储系统 (任务一至五详细复盘)

项目名称：大数据采集、存储与分析系统开发

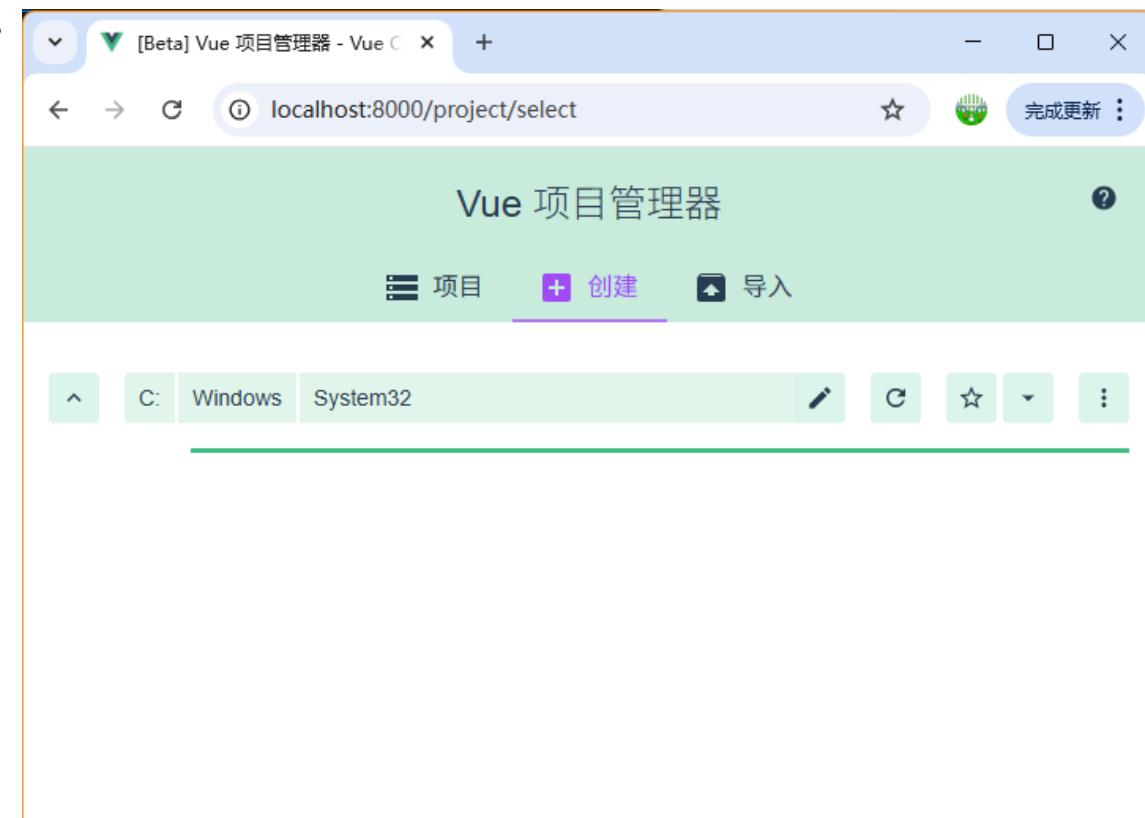
核心技术栈：VUE 2、Python (Flask)、MQTT、MySQL、InfluxDB

任务一：搭建系统框架 (VUE)

目标：按照指导书要求，从零开始搭建一个包含“欢迎界面”、“数据采集”、“数据显示”、“数据分析”四个页面的 VUE 前端应用框架。

1. 基础环境搭建

- **创建项目：**使用 `vue-cli` (通过 `vue ui` 命令) 创建了一个标准的 Vue 2 项目。



- **安装核心依赖：**为了实现后续功能，我们必须先安装几个核心库。

```
yarn add vue-router@3 element-ui axios echarts
```

- `vue-router@3`：用于页面跳转 (Vue 2 对应 v3)。
- `element-ui`：用于快速构建美观的前端 UI 界面。
- `axios`：用于前端 VUE 和后端 Python 之间的数据通信。
- `echarts`：(任务四预装) 用于数据可视化。

2. 编码与配置

- **创建组件：**在 `src/components/` 目录下，按要求新建了 `coreSource.vue`、`ShowData.vue` 和 `DataProcess.vue` 三个空白页面文件。

```

src > components > HelloWorld.vue > ...
1  <template>
2    <div class="hello">
3      <h1>{{ msg }}</h1>
4      <p>
5        For a guide and recipes on how to configure / customize
6        this project, check out the
7        <a href="https://cli.vuejs.org" target="_blank" rel="noreferrer">CLI Plugins</a>
8      </p>
9      <h3>Installed CLI Plugins</h3>
10     <ul>
11       <li><a href="https://github.com/vuejs/vue-cli/tree/dev/plugins">vue-cli-plugin-...
12       <li><a href="https://github.com/vuejs/vue-cli/tree/dev/packages">vue-cli-&...
13     </ul>
14     <h3>Essential Links</h3>
15     <ul>
16       <li><a href="https://vuejs.org" target="_blank" rel="noreferrer">Vue.js</a>
17       <li><a href="https://forum.vuejs.org" target="_blank" rel="noreferrer">Forum</a>
18       <li><a href="https://chat.vuejs.org" target="_blank" rel="noreferrer">Discord</a>
19       <li><a href="https://twitter.com/vuejs" target="_blank" rel="noreferrer">Twitter</a>
20       <li><a href="https://news.vuejs.org" target="_blank" rel="noreferrer">News</a>
21     </ul>
22     <h3>Ecosystem</h3>
23     <ul>
24       <li><a href="https://router.vuejs.org" target="_blank" rel="noreferrer">Vue Router</a>
25       <li><a href="https://vuex.vuejs.org" target="_blank" rel="noreferrer">Vuex</a>
26       <li><a href="https://github.com/vuejs/vue-devtools#vue-devtools" target="_blank" rel="noreferrer">Vue DevTools</a>
27       <li><a href="https://vue-loader.vuejs.org" target="_blank" rel="noreferrer">Vue Loader</a>
28       <li><a href="https://github.com/vuejs/awesome-vue" target="_blank" rel="noreferrer">Awesome Vue</a>
29     </ul>
30   </div>
31 </template>

```

- 配置路由 (`src/router.js`)**: 新建此文件，`import` 刚才创建的组件，并定义 `routes` 数组，将如 `/core` 这样的 URL 路径，精确地映射到 `coreSource` 组件。
- 修改入口 (`src/main.js`)**: 这是让插件生效的关键。我们添加了以下代码，来全局 `import` 并 `Vue.use()` 路由和 ElementUI。

```

import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';
import router from './router'; // 引入路由配置

Vue.use(ElementUI);
Vue.use(VueRouter);

new Vue({
  router, // 将 router 实例挂载到 VUE 根实例
  render: h => h(App),
}).$mount('#app');

```

- 修改主布局 (`src/App.vue`)**: 替换了 `App.vue` 的全部内容，使用 `<el-menu>` (ElementUI的导航栏) 和 `<router-view>` (路由出口) 重构了应用主体。这让我们能通过点击导航栏来切换页面内容。
- 配置跨域代理 (`vue.config.js`)**: 这是为任务三做的关键准备。我们修改了 `vue.config.js`，添加 `devServer.proxy` 配置 [cite: 2180-2181]。

```

devServer: {
  proxy: {
    '/api': { // 监听所有 /api 开头的请求
      target: 'http://127.0.0.1:5000', // 转发到 Python 后端
      changOrigin: true,
      pathRewrite: { '^/api': '' } // 转发时去掉 /api
    }
  }
}

```

3. 遇到的问题与解决

- **Bug 1.1: 'vue-cli-service' 不是内部或外部命令**
 - **分析:** 这是因为我们 (1) 混用了 `npm` 和 `yarn` (项目有 `yarn.lock`) ; (2) 没有运行 `yarn install` 来下载依赖。
 - **解决:** 统一使用 `yarn`。先运行 `yarn install` 安装 `node_modules`, 再运行 `yarn serve` 启动。
- **Bug 1.2: Module not found: Can't resolve 'element-ui'**
 - **分析:** 这是因为我们在 `main.js` 里 `import` 了 `element-ui` 等库, 但忘记了用 `yarn add` 把它们安装到项目里。
 - **解决:** 运行 `yarn add vue-router@3 element-ui axios echarts` 补全所有依赖。

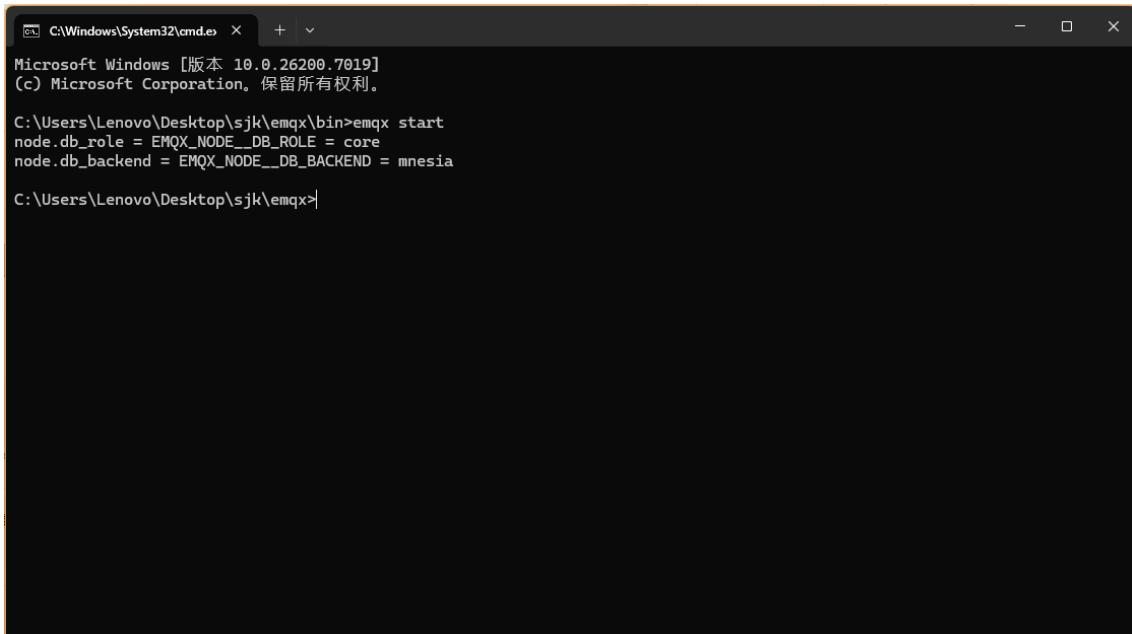
任务一成果: VUE 前端框架成功运行在 `http://localhost:8080`, 显示了四个导航标签, 并可以正确切换页面。

任务二：运行虚拟数据源 (MQTT)

目标: 启动模拟的 MQTT 数据源 (数控机床), 为后续任务提供数据。

执行总结: 这是一个纯环境步骤, 没有修改代码。

1. **启动 MQTT Broker:** 在 `emqx-5.0.11-windows-amd64\bin` 目录运行 `emqx start`, 启动了 MQTT 消息服务器。



```

C:\Windows\System32\cmd.exe x + ^
Microsoft Windows [版本 10.0.26200.7019]
(c) Microsoft Corporation, 保留所有权利。

C:\Users\Lenovo\Desktop\sjk\emqx\bin>emqx start
node.db_role = EMQX_NODE_DB_ROLE = core
node.db_backend = EMQX_NODE_DB_BACKEND = mnesia

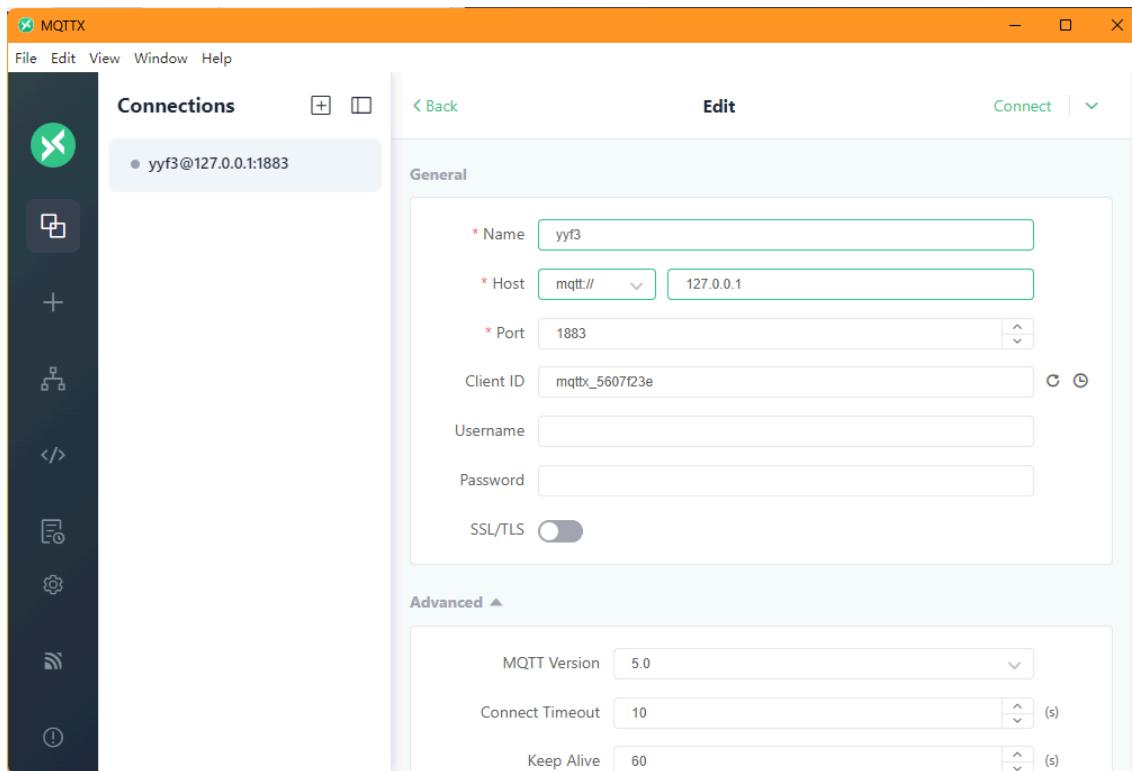
C:\Users\Lenovo\Desktop\sjk\emqx>

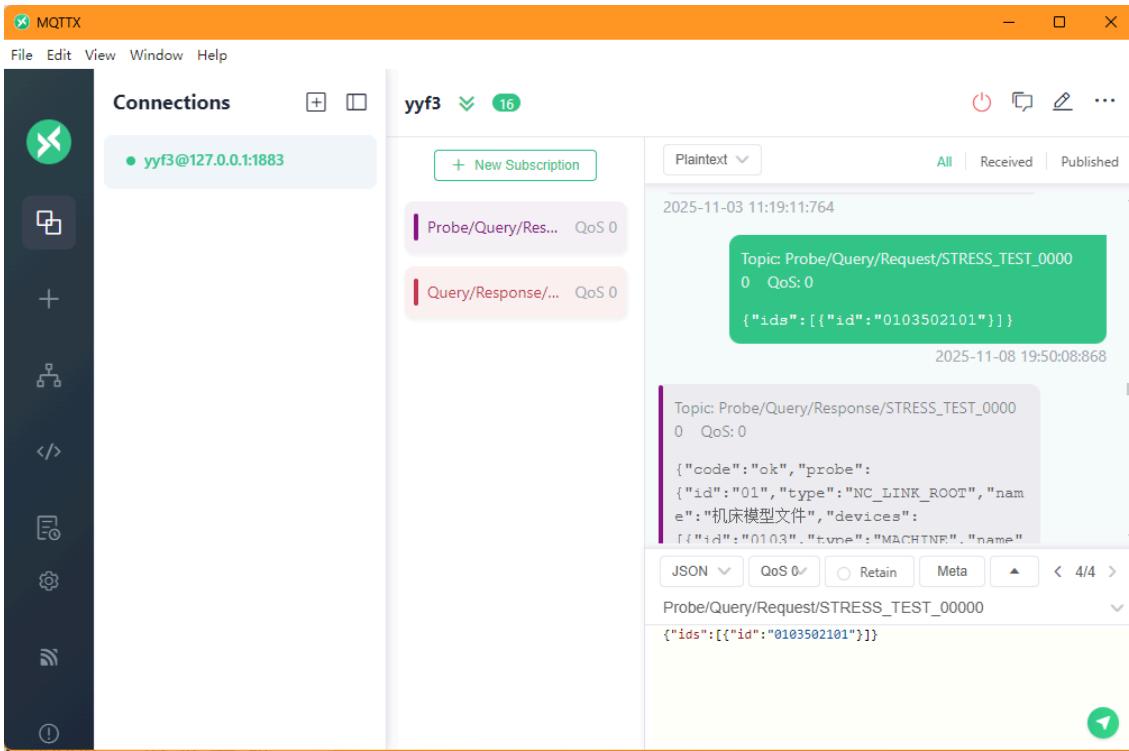
```

2. **启动虚拟数据源:** 在 `StressTest-INC-Cloud` 目录运行 `StressTest-INC-Cloud -n 1 -b 0`, 成功模拟了 `STRESS_TEST_00000` 设备。

```
C:\Users\Lenovo\Desktop\sjk\stressText>StressTest-INC-Cloud -n 1 -b 0  
设备台数: 1 起始编号: 0  
[STRESS_TEST_00000]is Connecting  
[STRESS_TEST_00000]Connected
```

3. 直接检测是否运行成功：在MQTTX软件中进行相关设置与连接，并进行订阅，以查看是否成功启动并运行。





JSON在线预览查看器(Online) 重新启动即可更新

JSON数据

```

{
    "data": [
        {
            "id": "0103502101",
            "type": "VARIABLE",
            "name": "VAR_A1",
            "value": "100"
        },
        {
            "id": "0103502102",
            "type": "VARIABLE",
            "name": "VAR_A2",
            "value": "200"
        },
        ...
    ],
    "code": "ok",
    "probe": {
        "id": "01",
        "type": "NC_LINK_ROOT",
        "name": "机床模型文件",
        "devices": [
            {
                "id": "0103",
                "type": "MACHTNFR",
                "name": "MACH-TNFR"
            }
        ]
    }
}

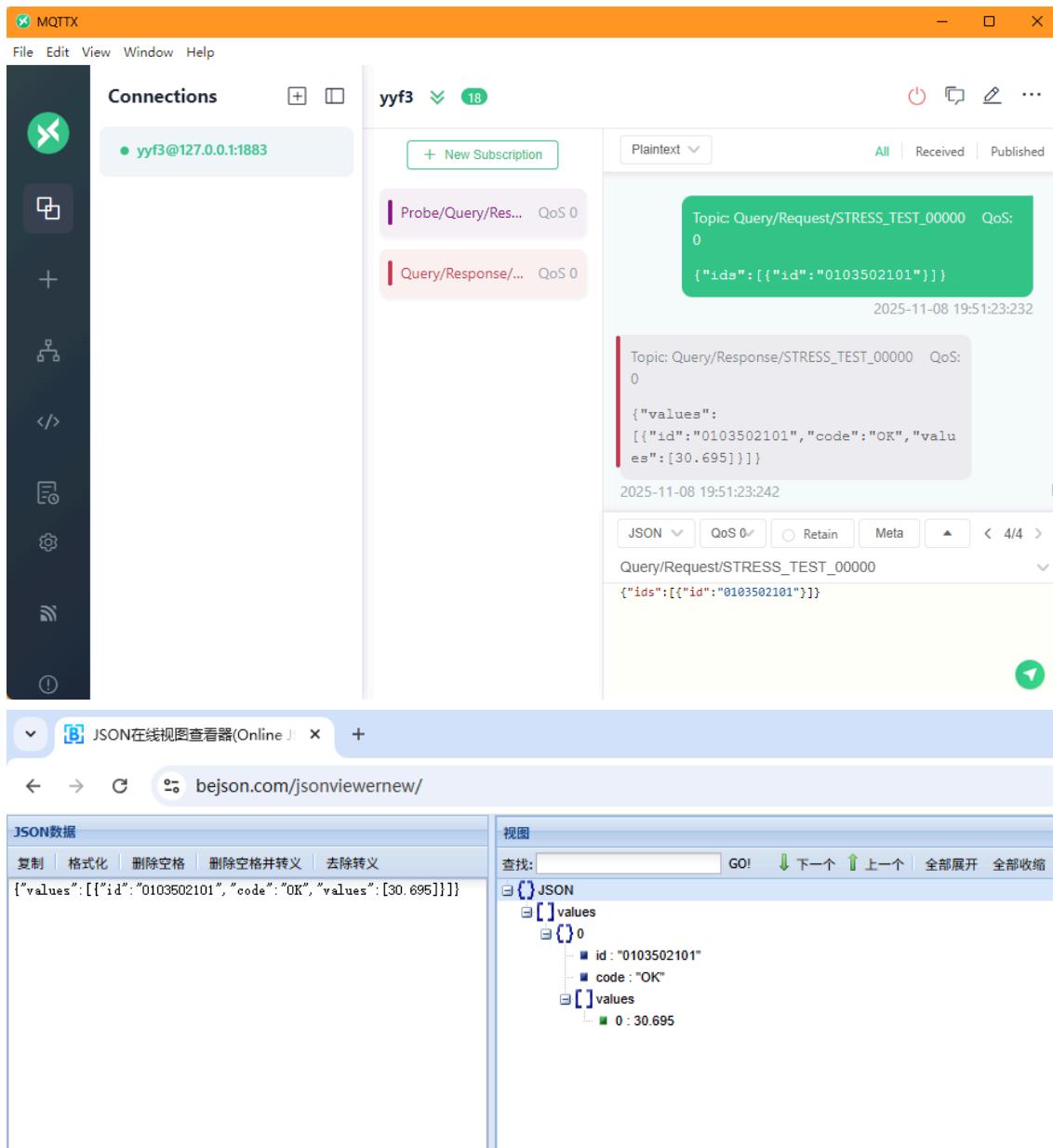
```

树状图

顶部工具栏：格式化、折叠所有、展开所有、选择所有、高级模式、重置视图、帮助和支持、退出。

右侧过滤器：实时、Name、Value。

右侧状态栏：重新启动即可更新。

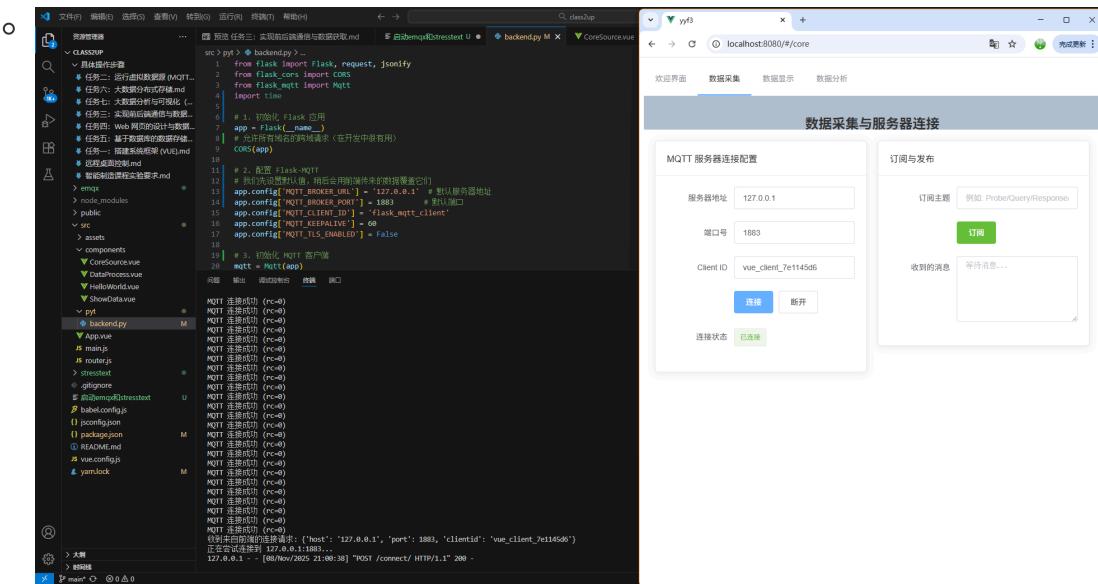


任务三：实现前后端通信

目标：打通 VUE -> Python -> MQTT 的链路。

1. 前端 CoreSource.vue (数据采集页) 修改:

- **<template>**: 添加了 **<el-form>** 表单，包含服务器地址、端口等输入框，以及一个“连接”按钮，它绑定了 `@click="onConnect"`。
- **<script>**:
 - `import axios from "axios";` 并设置 `axios.defaults.baseURL = "/api";`。
 - 在 `methods` 中添加 `onConnect` 函数，其核心是 `axios.post('/connect/', { data: this.connection })`。
 - 在 `.then()` 回调中，检查后端返回的 `res.data.rc_status`，如果为 "success"，则设置 `this.connectStatus = true`，并在页面上显示“连接成功”。



2. 后端 `pyt/backend.py` 新建与修改:

- 安装依赖:** `pip install Flask flask-cors flask-mqtt`
- 代码框架:** 使用 Flask 搭建了 Web 服务，并初始化了 MQTT 客户端。
- 添加 API 接口 (`/connect/`):**

```
@app.route('/connect/', methods=['POST', 'GET'])
def make_connect():
    # 1. 从 VUE 获取配置
    data_connect = request.get_json()['data']
    # 2. 更新 MQTT 配置
    app.config['MQTT_BROKER_URL'] = data_connect['host']
    app.config['MQTT_BROKER_PORT'] = data_connect['port']
    # 3. 连接到 MQTT Broker
    mqtt.client.connect(data_connect['host'], data_connect['port'])
    # 4. 向 VUE 返回成功消息
    return jsonify({'rc_status': 'success'})
```

3. 遇到的问题与解决

- Bug 3.1: `Proxy error: ... ECONNREFUSED`**
 - 分析:** VUE 报这个错 (在 F12 控制台) 是因为它找不到 `localhost:5000` 上的服务。
 - 解决:** 启动 `python pyt/backend.py` 服务。
- Bug 3.2: `ModuleNotFoundError: No module named 'flask_mqtt'`**
 - 分析:** Python 后端启动失败，因为 `backend.py` 中 `import` 了 `flask_mqtt`，但 Python 环境里没装。
 - 解决:** 运行 `pip install flask-mqtt` (以及 `Flask`、`flask-cors`、`pymysql`) 安装所有 Python 依赖。

任务三成果: VUE 页面点击“连接”后，成功将配置发给 Python，Python 后端成功连接到 EMQX，并返回“连接成功”的状态给 VUE。链路打通。

任务四：实现数据可视化 (Echarts)

目标：从 MQTT 获取实时数据，并在 `showData.vue` 页面使用 Echarts 动态展示。

1. 后端 `pyt/backend.py` 功能追加

- 添加全局缓存：`latest_data_store = {}`。
- 修改 `@mqtt.on_connect`：添加
`mqtt.subscribe("Query/Response/STRESS_TEST_00000")`，使后端自动订阅数据主题。
- 修改 `@mqtt.on_message (handle_message)`：添加数据处理逻辑。收到消息时，解析 JSON，并把数据存入缓存：`latest_data_store[item['id']] = item['values'][0]`。
- 新增 API 接口 (`/publish/`)：VUE 调用此接口时，后端会通过 `mqtt.publish(...)` 向 MQTT 发布数据请求。
- 新增 API 接口 (`/get_data/`)：VUE 调用此接口时，后端从 `latest_data_store` 中查询数据并返回。

2. 前端 `ShowData.vue` (数据显示页) 修改

- `<template>`：添加 `<div id="data-chart" ...>` 作为 Echarts 渲染容器。
- `<script>`：
 - `import * as echarts from "echarts";`。
 - `mounted()`：页面加载时，调用 `this.initChart()` 和 `this.startFetching()`。
 - `initChart() (新增)`：调用 `echarts.init(document.getElementById('data-chart'))` 初始化图表，并设置好 `series: [{ type: 'line', smooth: true }]` 等基础样式。
 - `startFetching() (新增)`：使用 `setInterval(this.requestAndFetch, 3000)` 创建了一个 3 秒的定时器。
 - `requestAndFetch() (新增)`：实现“轮询”。它先 `axios.post('/publish/')`（命令后端请求新数据），再 `setTimeout` 延迟 500ms 后调用 `this.fetchData()`（去后端拿数据）。
 - `fetchData() (新增)`：图表更新的核心。它 `axios.post('/get_data/')` 取数，在 `.then()` 回调中，将新数据 `push` 进 `this.chartData` 数组（并用 `shift()` 保持数组长度不超过 50），最后调用 `this.chart.setOption({...})` 更新图表。

3. 遇到的问题与解决

- Bug 4.1：“图表没线”(来自 `image_1e4f16.png`)
 - 现象：**VUE 页面图表是空的，F12 控制台显示 `数据ID ... 尚未收到`。但后端终端明确显示 `已更新数据存储`。
 - 分析：**这是一个 Python 作用域 Bug。`handle_message` 存入的是全局 `latest_data_store`，但 `get_data` 函数没有声明 `global`，它读取的是一个同名的局部空字典。
 - 解决：**在 `backend.py` 的 `get_data` 函数开头，添加 `global latest_data_store` 这一行，让它去读取正确的全局字典。
- Bug 4.2：“还是没线”(来自 `image_2b90d9.png`)
 - 现象：**修复 Bug 4.1 后，F12 控制台确认 VUE **拿到数据了**（日志显示 `前端获取数据成功...value=-10.449`），但图表上**只有一个点**，不滚动。

- 分析: Echarts 的一个特性。我们在 `initchart` 时设置了 `type: 'line'`, 但在 `fetchData` 里用 `setOption` 更新数据时, 只传了 `data`, Echarts “忘记了”它应该画线。
- 解决: 修改 `ShowData.vue` 的 `fetchData` 函数, 在 `this.chart.setOption` 中, 必须完整地传入 `series: [{ data: this.chartData, type: 'line', smooth: true }]`。

任务四成果: VUE 前端的“数据显示”页面成功绘制出一条实时滚动的 Echarts 折线图。

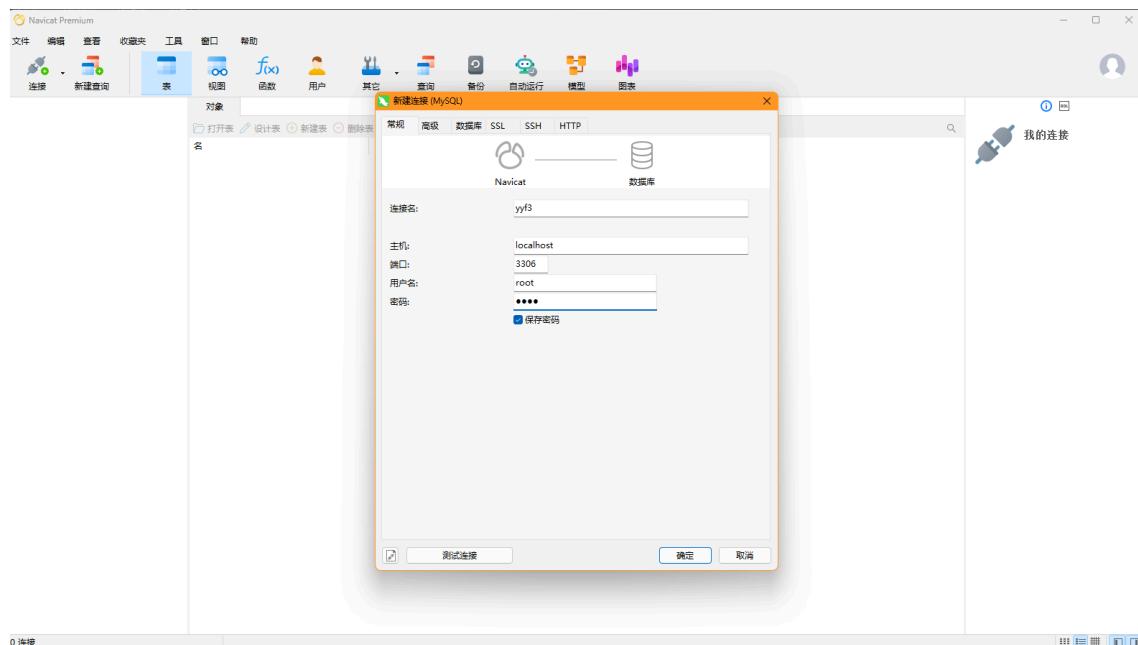
The screenshot shows a dual-pane development interface. On the left, a code editor displays a Python file named `backend.py` with code related to MQTT message handling and a Flask API endpoint for connecting. On the right, a web browser window titled "yyf3" shows a real-time chart titled "实时数据显示 (X轴实际速度)" (Real-time Data Display (X-axis Actual Speed)). The chart has a Y-axis ranging from -21 to -6 and an X-axis from 0 to 7. It displays a single horizontal blue line at a value of approximately -14. A status bar at the bottom indicates "实时数据 (010350202)".

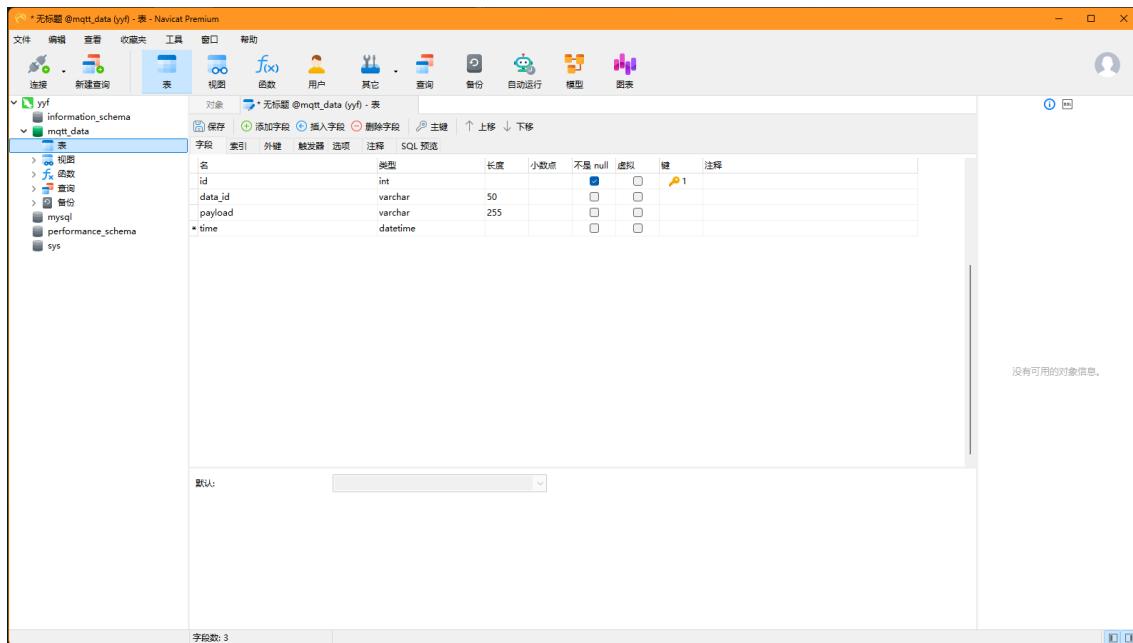
任务五: 实现多数据库存储 (MySQL & InfluxDB)

目标: 将采集到的数据, 同时写入边缘端 (本地 MySQL + 本地 InfluxDB) 和云端 (InfluxDB Cloud)。

1. 环境准备与配置

- MySQL: 在 Navicat 中按指导书创建了 `mqtt_data` 数据库和 `mac_data` 表。





- **InfluxDB (本地):** 通过 `influxd` 启动服务，在 `localhost:8086` 上配置了 Org (`xJTU`)、Bucket (`class2down`) 和 Token (`INFLUX_LOCAL_TOKEN`)。

Welcome — Initial User Setup — Complete

You are ready to go!

Make sure to copy your operator API token now.

This token enables superuser privileges like creating users, orgs, etc. You won't be able to see it again!

7ucc4S8rrzwu85NA5nUYb_CNG7C-03Rbuyf2A85A51eATuxcPH_U1FvrCNXSGQtxvZQTuY_C607BUWNg4oIH-g==

COPY TO CLIPBOARD

Your InfluxDB has 1 organization, 1 user, and 1 bucket.

Let's start collecting data!

QUICK START

ADVANCED

CONFIGURE LATER

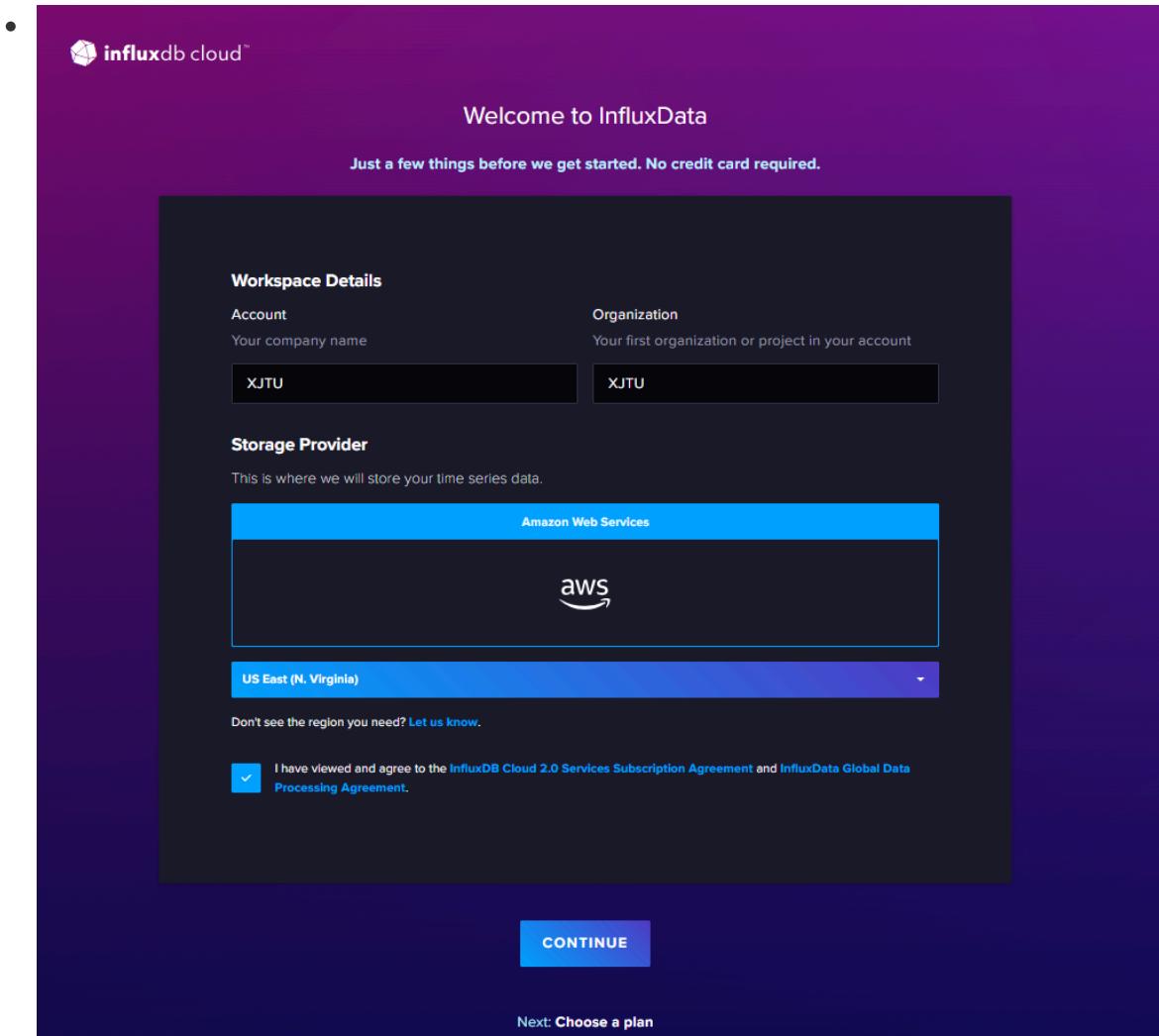
Timing is everything!
This will set up local metric collection and allow you to explore the features of InfluxDB quickly.

Whoa looks like you're an expert!
This allows you to set up Telegraf, scrapers, and much more.

I've got this...
Jump into InfluxDB and set up data collection when you're ready.

Powered by [InfluxData](#)

- **InfluxDB (云端):** 在 `influxdata.com` 网站注册并登录。创建了 Bucket (`class2down`)，获取了云端的 URL、Org (`xJTU`) 和 Token (`INFLUX_CLOUD_TOKEN`) [cite: 1066, 2224-2236, 3013]。



- **backend.py 配置:** 将上述所有 5 个关键凭证 (MySQL 密码 `cps@CPS123`、本地 Token、云端 Token 等) 添加为 Python 的全局变量。

```
# --- 数据库配置 ---
# 1. MySQL 配置
MYSQL_HOST = 'localhost'
MYSQL_USER = 'root'
MYSQL_PASSWORD = 'cps@CPS123' # 你的 MySQL 密码
MYSQL_DB = 'mqtt_data'
MYSQL_PORT = 3306

# 2. InfluxDB (本地) 配置
INFLUX_LOCAL_URL = "http://localhost:8086"
INFLUX_LOCAL_TOKEN = "7ucc4S8rrzwu85NA5nUYb_CNG7C-03Rbuyf2A85A5leATuxcPH_U1FvrCNXSGQtxvZQTuY_C607BUWNg4oIH-g==" # 你的本地 Token
INFLUX_LOCAL_ORG = "XJTU"
INFLUX_LOCAL_BUCKET = "class2down"

# 3. InfluxDB (云端) 配置
INFLUX_CLOUD_URL = "https://us-east-1-1.aws.cloud2.influxdata.com"
INFLUX_CLOUD_TOKEN = "zs0j7fEHwI3DUOhFvhZ0ir39tRyWxko0oGAS5grC8PVyr-RhLe9X3WwHWPjdMjm1ZRHwxqt2pE5XWi3TQ340g==" # 你的云端 Token
INFLUX_CLOUD_ORG = "XJTU"
INFLUX_CLOUD_BUCKET = "class2down"
```

2. 后端 `backend.py` 逻辑修改

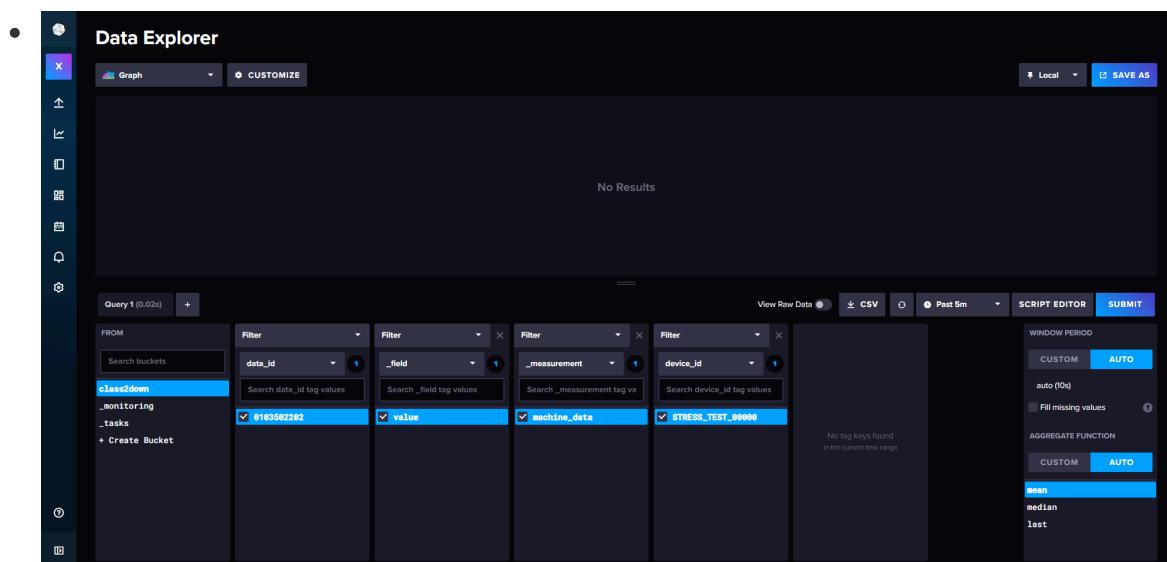
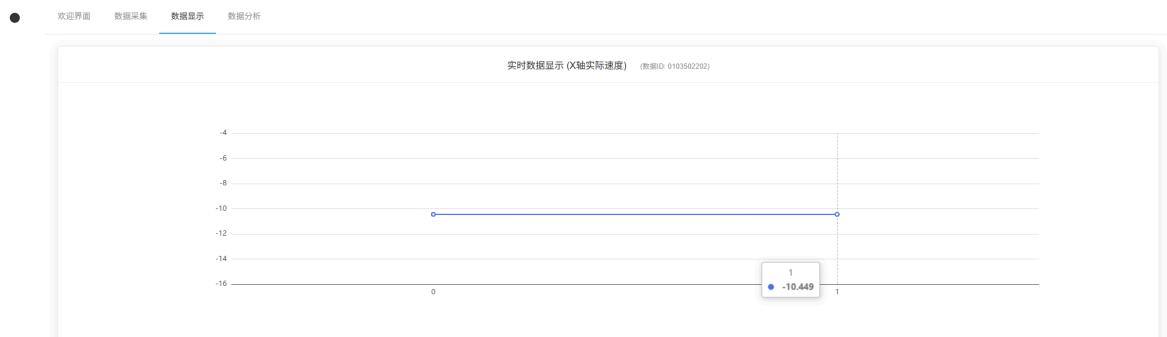
- **安装依赖:** `pip install pymysql influxdb-client`。
- **初始化客户端:** 在启动时，分别创建了 `influx_client_local` 和 `influx_client_cloud` 两个 InfluxDB 客户端实例。
- **添加三个写入函数:**
 - `save_to_mysql(...)`: 使用 `pymysql.connect(...)` 和 `cur.execute("INSERT ...")` 将数据写入 MySQL。
 - `save_to_influxdb_local(...)`: 创建 `Point` 并使用 `write_api_local.write(...)` 写入本地 InfluxDB。

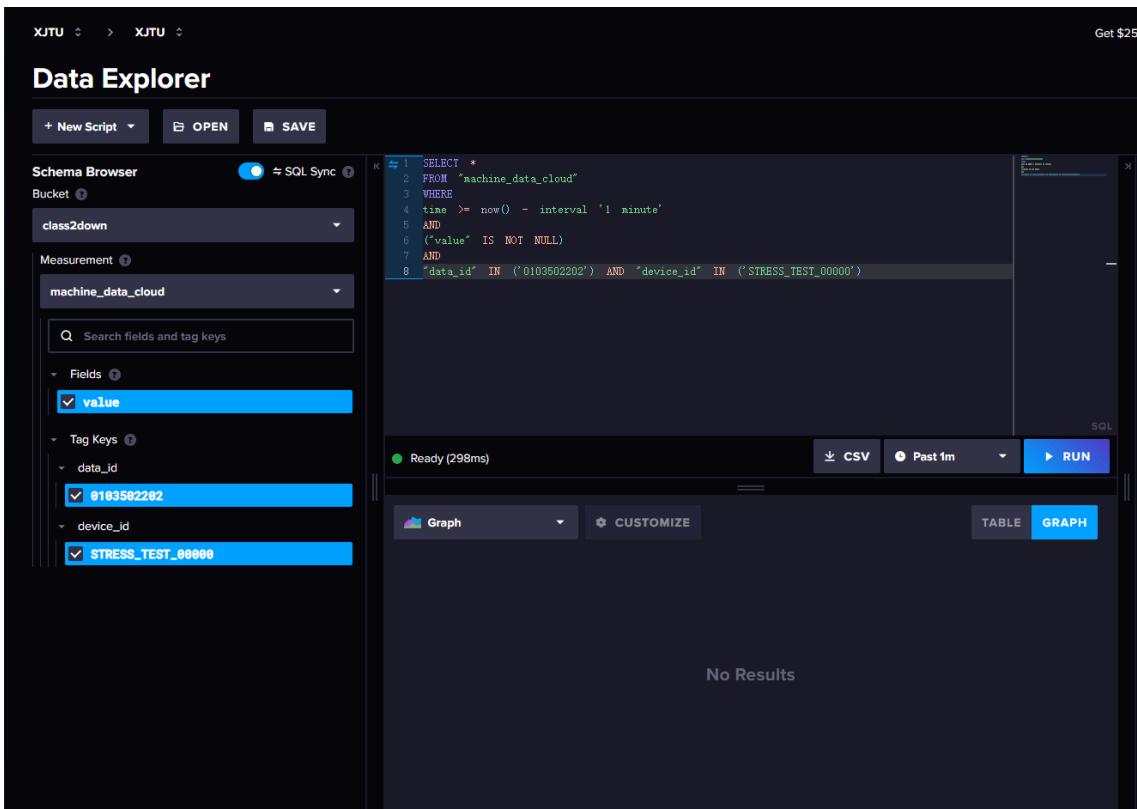
- `save_to_influxdb_cloud(...)` : 创建 Point 并使用 `write_api_cloud.write(...)` 写入云端 InfluxDB。
- 修改 `@mqtt.on_message(handle_message)`:
 - 这是数据分发的核心。当一条新数据被收到时，它会依次调用：
 - `latest_data_store[data_id] = value` (存入内存，给 VUE 用)
 - `save_to_mysql(...)` (存入 MySQL)
 - `save_to_influxdb_local(...)` (存入本地 InfluxDB)
 - `save_to_influxdb_cloud(...)` (存入云端 InfluxDB)
 - `save_to_distributed_platform(...)` (为任务六准备缓存文件)

3. 遇到的问题与解决

- Bug 5.1: “无限重启” Bug 导致无法接收到数据

● MQTT 连接成功 (`rc=0`)
 已自动订阅主题: `Query/Response/STRESS_TEST_00000`
 MQTT 连接成功 (`rc=0`)
 已自动订阅主题: `Query/Response/STRESS_TEST_00000`





- **现象:** 添加完所有数据库代码后，整个系统崩溃。Python 后端无限重启（不停打印 MQTT 连接成功）。导致 VUE 图表没线、InfluxDB 也没数据（`No Results`）。
- **分析:** `app.run(debug=True)` 模式会监视项目文件。而我们为任务六准备的 `save_to_distributed_platform` 函数会写入项目内的 `pyt/hdfs_temp_data.csv` 文件。这个“写入”动作被监视器捕捉到，误以为代码被修改，从而触发服务自动重启。服务在数据写入（异步）完成前就重启，导致内存（`latest_data_store`）丢失，数据库写入失败。
- **解决:** 修改 `backend.py` 的最后一行，添加 `use_reloader=False`，禁止 Flask 监视文件变动导致重启。

```
if __name__ == '__main__':
    app.run(debug=True, use_reloader=False, host='127.0.0.1', port=5000)
```

任务五成果：

在解决了“无限重启”Bug 后，所有 5 个服务（EMQX、StressTest、Influxd、VUE、Python）终于稳定协同工作。

1. VUE 页面成功显示实时图表。
2. Python 后端日志稳定打印 MySQL 写入成功、InfluxDB 本地 写入成功、InfluxDB 云端 写入成功。
3. 在 Navicat、本地 InfluxDB (`localhost:8086`) 和云端 InfluxDB (`influxdata.com`) 中均能查询到实时写入的数据。

问题 输出 调试控制台 终端 端口

InfluxDB 客户端初始化成功。
 MQTT 连接成功 (rc=0)
 已自动订阅主题: Query/Response/STRESS_TEST_00000
 * Serving Flask app 'backend'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
 Press CTRL+C to quit
 * Restarting with stat
 InfluxDB 客户端初始化成功。
 MQTT 连接成功 (rc=0)
 已自动订阅主题: Query/Response/STRESS_TEST_00000
 * Debugger is active!

