实验四：物联网监控系统

1. 实验内容

使用websocket建立网络聊天室，聊天室成员既包括web用户，也包括物联网节点，web用户可对物联网节点进行查询和控制。物联网节点可以是虚拟节点，如：mqtt.fx。

1. 实验准备

- 腾讯云虚拟机

- Docker 环境

- EMQX （服务器端）

- Python编译环境 （WEB端）

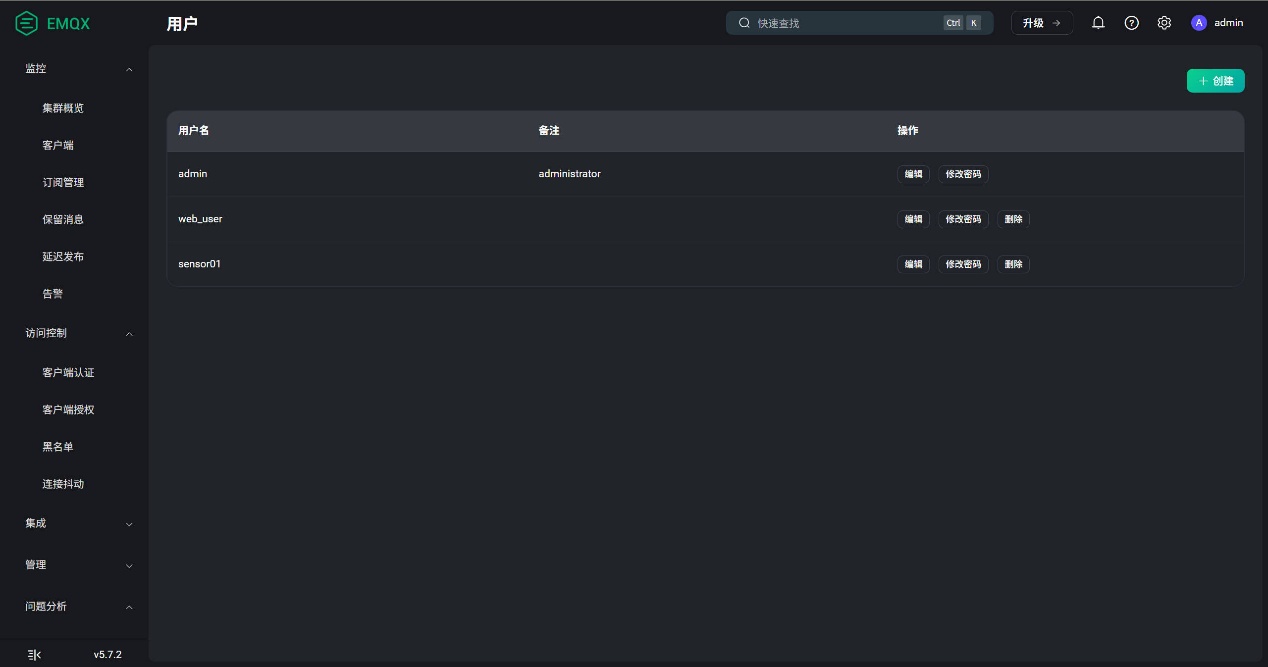
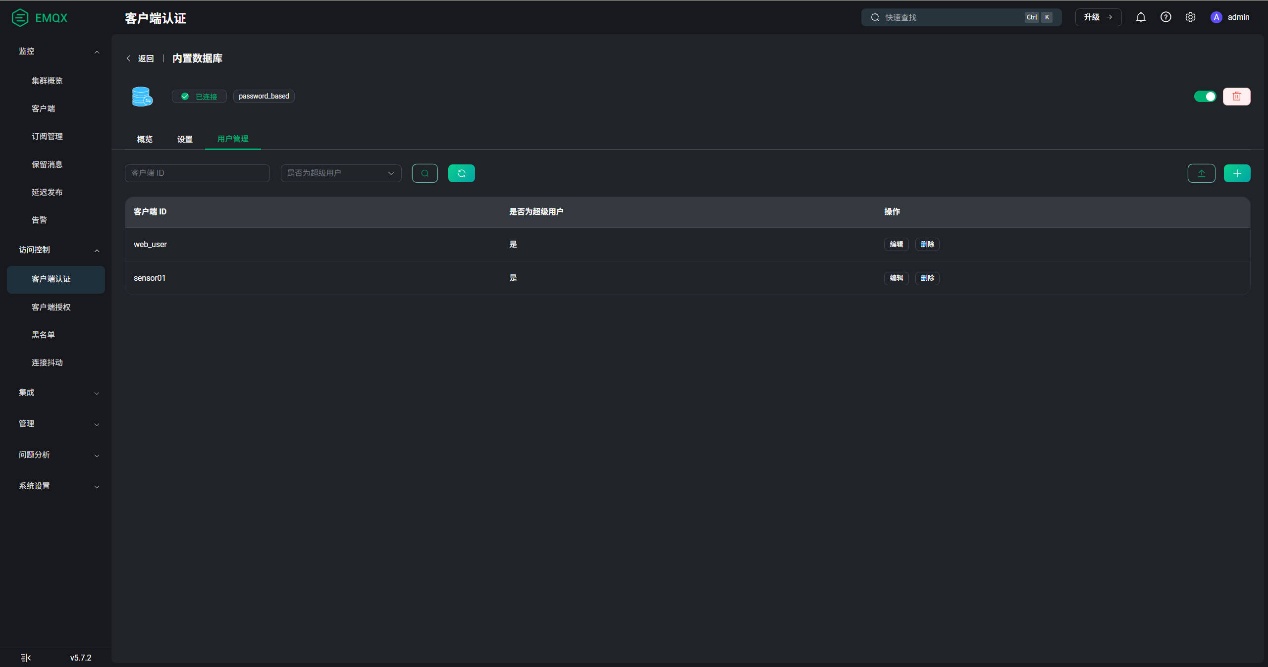
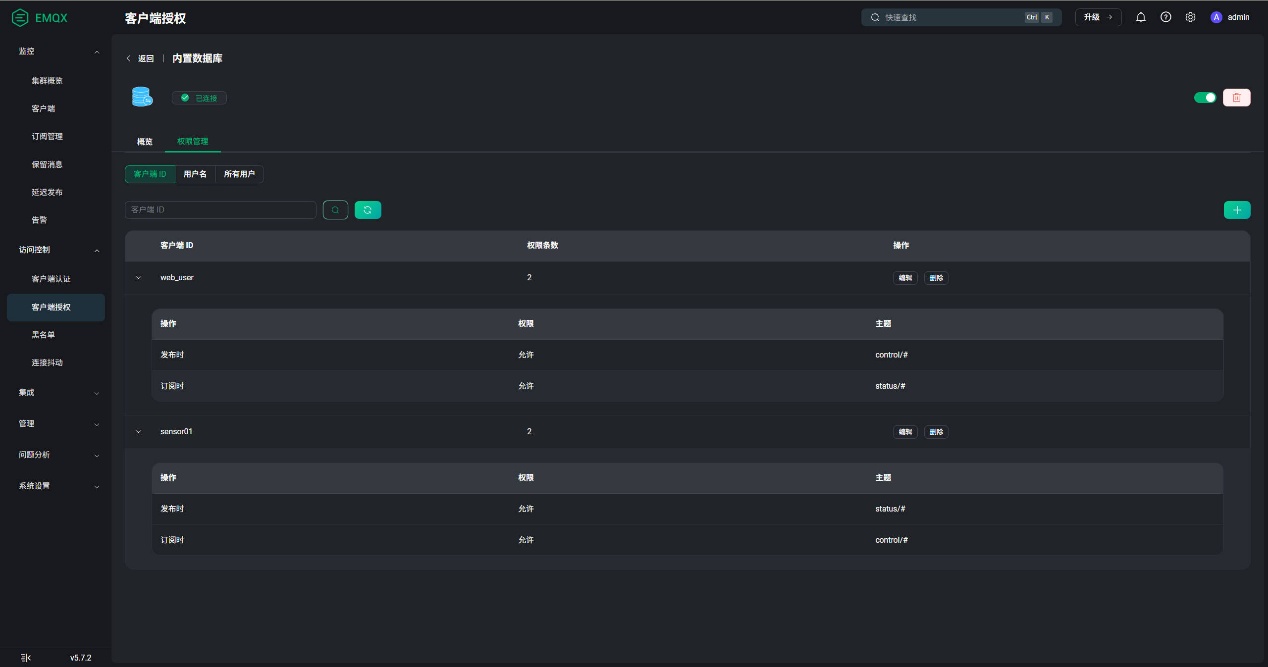
- 本机

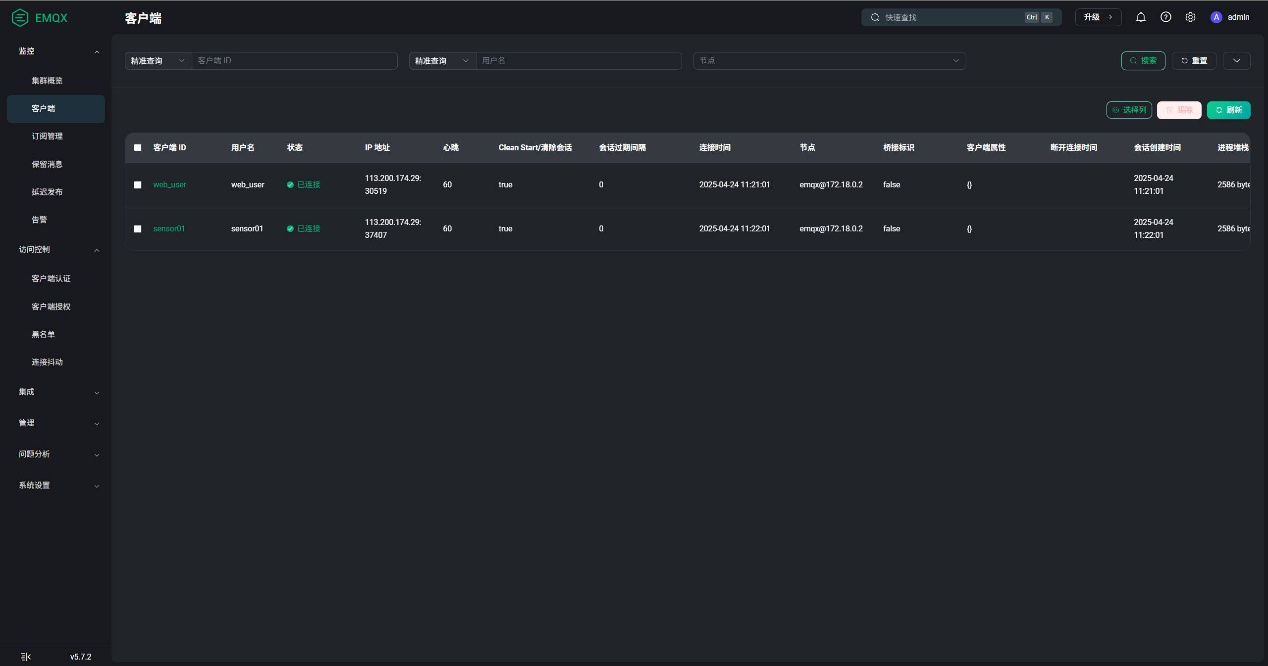
- 下载MQTT.fx （设备端）

三、核心配置

装好EMQX打开`http://139.155.27.201:18083/`，开始配置测试用户

1. 服务器用户认证配置/权限规则配置



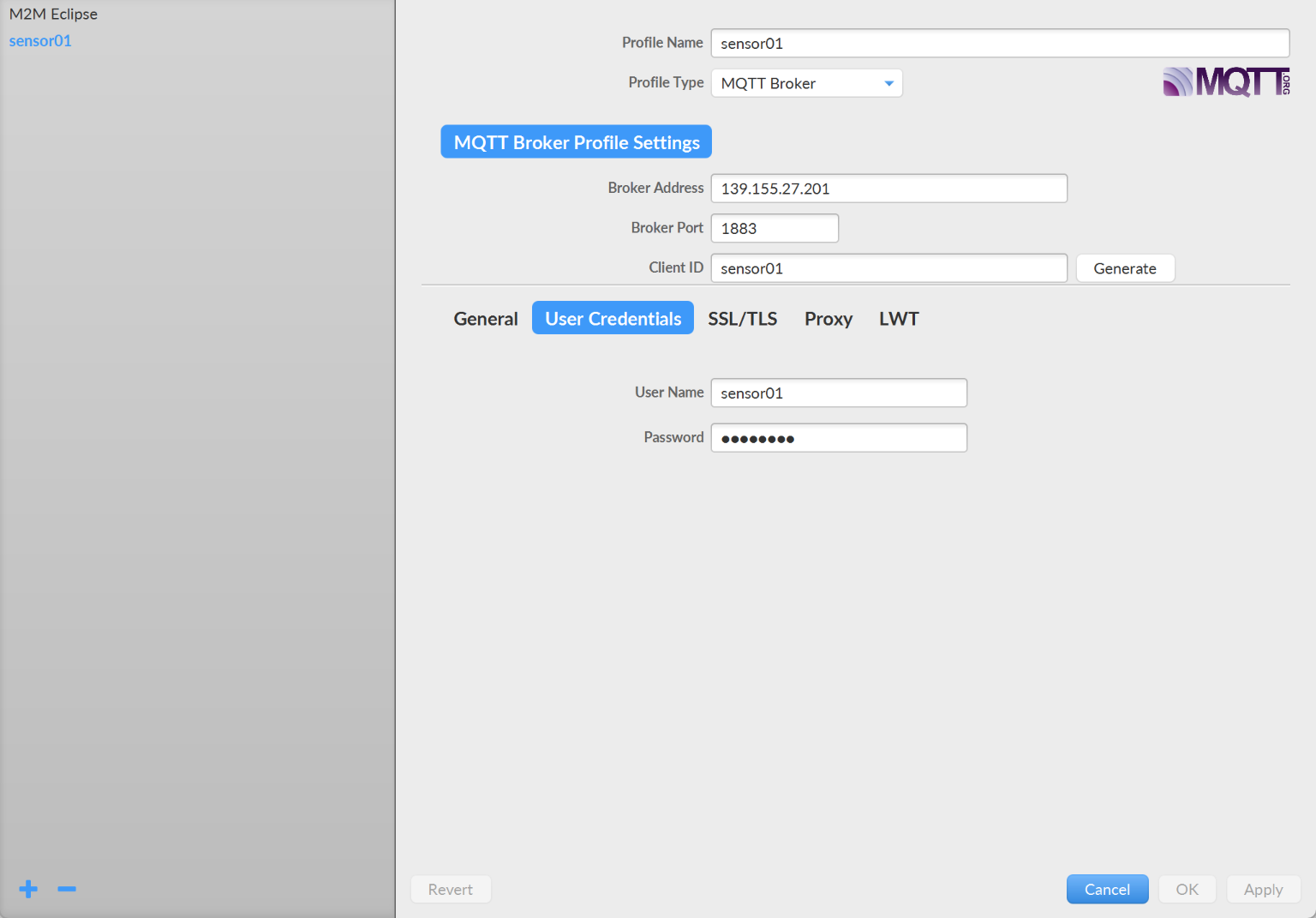
* 1. 用户认证配置表

|  |  |  |
| --- | --- | --- |
| 用户名 | 密码 | 用户类型 |
| sensor01 | sensor01 | 物联网设备 |
| web\_user | web\_user | Web控制端 |

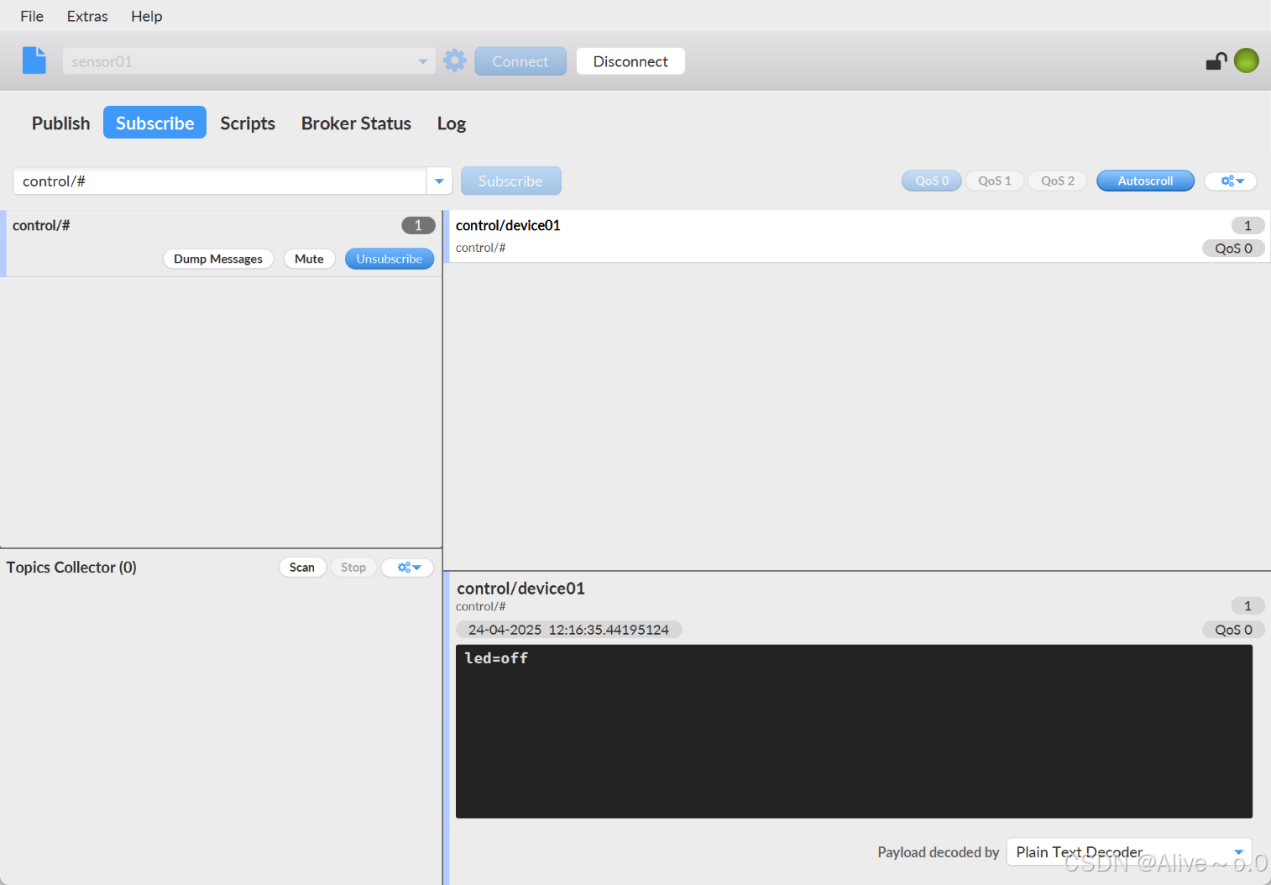
1.2 权限规则配置

|  |  |  |  |
| --- | --- | --- | --- |
| 用户名 | 操作类型 | 主题过滤器 | 权限状态 |
| web\_user | 发布 | control/# | 允许 |
| web\_user | 订阅 | status/# | 允许 |
| sensor01 | 发布 | status/# | 允许 |
| sensor01 | 订阅 | control/# | 允许 |

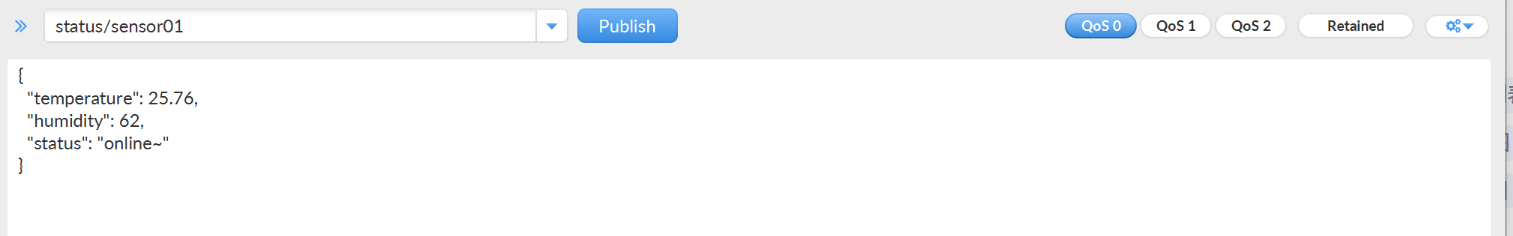
1. MQTT.fx配置



订阅



发布



1. WEB端配置
   1. 核心逻辑​
2. *// MQTT连接配置（对应实验参数）*
3. const client = mqtt.connect('ws://你的服务器IP:8083/mqtt', {
4. username: 'web\_user',
5. password: 'web\_123456',
6. clientId: `web\_${Date.now()}`
7. });
8. *// 主题订阅管理*
9. const subscribeDevice = (deviceId) => {
10. client.subscribe(`status/${deviceId}`, { qos: 1 }, (err) => {
11. if (!err) addLog(`已订阅设备${deviceId}状态`);
12. });
13. };
14. *// 指令下发验证*
15. const validateCommand = (input) => {
16. const pattern = /^[a-z0-9]{8}:(start|stop|reset|led=(on|off))$/;
17. return pattern.test(input);
18. };
    1. 完整代码
19. <!DOCTYPE html>
20. <html lang="zh-CN">
21. <head>
22. <meta charset="UTF-8">
23. <meta name="viewport" content="width=device-width, initial-scale=1.0">
24. <title>物联网设备控制台</title>
25. <script src="https://unpkg.com/mqtt@5.3.0/dist/mqtt.min.js"></script>
26. <style>
27. body {
28. margin: 0;
29. padding: 20px;
30. font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
31. background-color: #fff;
32. }
33. .status-bar {
34. color: #2ecc71; */\* 连接中状态颜色 \*/*
35. font-size: 14px;
36. margin-bottom: 15px;
37. }
38. .message-container {
39. height: 300px;
40. border: 1px solid #e0e0e0;
41. border-radius: 4px;
42. margin-bottom: 15px;
43. overflow-y: auto;
44. padding: 10px;
45. }
46. .input-group {
47. display: flex;
48. gap: 10px;
49. }
50. #commandInput {
51. flex: 1;
52. padding: 8px 12px;
53. border: 1px solid #e0e0e0;
54. border-radius: 4px;
55. font-size: 14px;
56. }
57. #sendButton {
58. background: #007bff;
59. color: white;
60. border: none;
61. padding: 8px 16px;
62. border-radius: 4px;
63. cursor: pointer;
64. }
65. </style>
66. </head>
67. <body>
68. *<!-- 状态显示 -->*
69. <div class="status-bar" id="status">状态：连接中...</div>
71. *<!-- 消息展示区域 -->*
72. <div class="message-container" id="messages"></div>
74. *<!-- 控制输入区 -->*
75. <div class="input-group">
76. <input
77. type="text"
78. id="commandInput"
79. placeholder="输入控制指令（格式：设备ID:指令）"
80. value="device01:led=on"
81. >
82. <button id="sendButton" onclick="sendCommand()">发送</button>
83. </div>
84. <script>
85. *// ================= MQTT连接配置 =================*
86. const client = mqtt.connect('ws://139.155.27.201:8083/mqtt', {
87. username: 'web\_user',
88. password: 'web\_user',
89. clientId: 'web\_user'
90. });
91. *// ================= 连接状态管理 =================*
92. client.on('connect', () => {
93. document.getElementById('status').textContent = '状态：已连接';
94. document.getElementById('status').style.color = '#27ae60';
96. *// 自动订阅设备状态主题*
97. client.subscribe('status/#', (err) => {
98. if (!err) addSystemMessage('成功订阅设备状态主题');
99. });
100. });
101. client.on('error', (err) => {
102. document.getElementById('status').textContent = `错误：${err.message}`;
103. document.getElementById('status').style.color = '#e74c3c';
104. });
105. *// ================= 消息处理逻辑 =================*
106. client.on('message', (topic, payload) => {
107. const msg = document.createElement('div');
108. msg.textContent = `[${new Date().toLocaleTimeString()}] ${topic}: ${payload.toString()}`;
109. msg.style.padding = '5px 0';
110. msg.style.borderBottom = '1px solid #eee';
111. document.getElementById('messages').appendChild(msg);
112. messages.scrollTop = messages.scrollHeight; *// 自动滚动到底部*
113. });
114. *// ================= 设备控制逻辑 =================*
115. const sendCommand = () => {
116. const input = document.getElementById('commandInput').value.trim();
117. if (!input) return;
118. try {
119. const [deviceId, command] = input.split(':');
120. if (!deviceId || !command) throw new Error('指令格式错误');
122. client.publish(`control/${deviceId}`, command);
123. addSystemMessage(`已发送指令到 ${deviceId}`);
124. } catch (err) {
125. addSystemMessage(`错误：${err.message}`, true);
126. }
127. }
128. *// ================= 辅助函数 =================*
129. const addSystemMessage = (text, isError = false) => {
130. const msg = document.createElement('div');
131. msg.textContent = `[系统] ${new Date().toLocaleTimeString()} ${text}`;
132. msg.style.color = isError ? '#e74c3c' : '#7f8c8d';
133. msg.style.fontSize = '12px';
134. msg.style.margin = '4px 0';
135. document.getElementById('messages').appendChild(msg);
136. messages.scrollTop = messages.scrollHeight;
137. }
138. </script>
139. </body>
140. </html>

启动WEB：

1. *# 1. 部署到服务器指定目录*
2. cd /home/lighthouse/chat-server/public
3. vim index.html *# 粘贴上述代码*
4. *# 2. 重启HTTP服务*
5. pkill python3
6. nohup python3 -m http.server 8080 --bind 0.0.0.0 >/dev/null 2>&1 &
7. *# 3. 访问地址*
8. echo "控制台地址：http://139.155.27.201:8080"

四、实验结果

如图,主要功能如下：

|  |  |  |
| --- | --- | --- |
| 测试场景 | 操作步骤 | 预期结果 |
| Web用户订阅状态 | 浏览器控制台输入client.subscribe('status/#') | 成功接收所有设备状态消息 |
| 非法主题发布拦截 | 使用web\_user尝试发布到status/device01 | 返回PUBLISH not authorized错误 |
| 设备离线消息保留 | 设备离线时发送控制指令，设备上线后 | 立即执行离线期间最后一条指令 |
| 跨用户权限隔离 | 使用sensor01尝试订阅system/alerts | 连接被立即断开 |

