

《基于MQTT的车载环境传感器测试系统》3天快速启动计划

核心目标：3天内完成软件模拟环境搭建+核心测试用例编写+测试执行+成果整理，产出可直接写入实习简历的项目素材（测试用例文档+1页项目总结+缺陷记录），完全复用MQTT/Postman/Python/测试用例设计技能，无硬件需求，每日耗时3-4小时。

一、前置准备（启动前1小时完成）

1. 工具确认

- 已安装：Postman（需提前安装MQTT插件）、Python 3.x（含pip）、Excel/WPS表格（用于编写用例和记录结果）
- 可选安装：VS Code/PyCharm（Python代码编辑，用自带IDLE也可）

2. 资源准备（直接使用，无需额外配置）

- 公共MQTT服务器：<mqtt://broker.emqx.io:1883>（免费，无需注册，直接连接）
- 备用公共MQTT服务器：<mqtt://test.mosquitto.org:1883>（若第一个连接失败时使用）
- Python模拟代码模板（详见第1天步骤，可直接复制运行）
- 测试用例模板、缺陷记录模板（均为Excel格式，详见第2、3天步骤）

二、第1天：搭建MQTT测试环境 + 模拟传感器数据（3.5小时）

核心任务：打通“模拟车载传感器→MQTT服务器→云端订阅”完整通信链路，验证基础连通性。

上午（2小时）：Postman连接MQTT服务器，完成基础发布/订阅

步骤1：Postman MQTT插件配置（30分钟）

1. 打开Postman，点击左侧导航栏「Extensions」（扩展程序）；
2. 在搜索框输入「MQTT Client」，找到对应插件后点击「Install」安装；
3. 安装完成后，左侧导航栏会出现「MQTT Client」，点击进入；

4. 点击「New Connection」新建连接，填写以下信息：

- Connection Name（连接名称）：车载传感器测试_MQTT
- Broker（服务器地址）：broker.emqx.io
- Port（端口）：1883
- Protocol（协议）：MQTT/TCP

5. 点击「Connect」，若显示「Connected」则连接成功；若失败，检查网络后切换至备用服务器重试。

步骤2：Postman实现订阅+发布测试（1.5小时）

1. 创建订阅（模拟云端接收数据）：

- 在MQTT Client页面点击「Subscribe」（订阅）标签；
- Topic（主题）：填写 `car/sensor/001`（符合车联网“车/传感器/设备ID”的命名规范）；
- QoS Level（服务质量等级）：选择「1」（至少一次，车载数据常用等级）；
- 点击「Subscribe」，下方会出现“Subscription Messages”监听窗口，等待接收数据。

2. 创建发布（模拟传感器上报数据）：

- 点击「Publish」（发布）标签；
- Topic（主题）：填写与订阅一致的 `car/sensor/001`；
- QoS Level：选择「1」；
- Payload（消息体）：粘贴以下JSON格式数据（模拟车载温湿度、PM2.5数据）：

```
{  
  "device_id": "car_001",  
  "temp": 25.5,  
  "humidity": 45.2,  
  "pm25": 32,  
  "timestamp": "2025-12-26 10:30:00"  
}
```

3. 验证通信：点击「Publish」，返回订阅监听窗口，若能看到刚才发布的JSON数据，则说明MQTT链路打通；

4. 记录：在记事本中简单记录“通信链路验证成功”，若遇到连接失败、数据接收不到等问题，需记录问题现象及解决方法（如切换服务器、检查主题一致性）。

下午/晚上（1.5小时）：Python编写模拟传感器数据脚本（自动上报）

步骤1：编写模拟脚本（1小时）

1. 打开Python编辑工具（VS Code/PyCharm/自带IDLE），新建文件，命名为

```
car_sensor_mqtt.py;
```

2. 复制以下代码（无需修改，直接适配公共MQTT服务器）：

```
import random
import time
from paho.mqtt import client as mqtt_client

# MQTT服务器配置（与Postman连接信息一致）
BROKER = 'broker.emqx.io'
PORT = 1883
TOPIC = 'car/sensor/001' # 与Postman订阅主题完全一致
CLIENT_ID = f'python_car_sensor_{random.randint(0, 1000)}' # 随机生成客户端ID，避免冲突

# 连接MQTT服务器的回调函数
def connect_mqtt():
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("✅ 成功连接到MQTT服务器（模拟车载传感器）")
        else:
            print(f"❌ 连接失败，错误码：{rc}，请检查服务器地址或网络")
    # 创建MQTT客户端实例
    client = mqtt_client.Client(CLIENT_ID)
    client.on_connect = on_connect
    client.connect(BROKER, PORT)
    return client

# 生成模拟车载传感器数据（符合车载环境参数范围）
def generate_sensor_data():
    return {
        "device_id": "car_001",
        "temp": round(random.uniform(-40, 85), 1), # 车载传感器温度范围：-40°C ~ 85°C
```

```

        "humidity": round(random.uniform(10, 90), 1), # 湿度范围:
10% ~ 90%
        "pm25": random.randint(0, 500), # PM2.5浓度范围: 0 ~ 500
        μg/m³
        "timestamp": time.strftime("%Y-%m-%d %H:%M:%S") # 实时时间戳
    }

# 向MQTT服务器发布数据（模拟传感器上报）
def publish(client):
    while True:
        sensor_data = generate_sensor_data()
        # 将字典格式数据转为字符串发布
        result = client.publish(TOPIC, str(sensor_data))
        # 检查发布结果
        status = result[0]
        if status == 0:
            print(f"📶 成功上报数据: {sensor_data}")
        else:
            print(f"📶 上报失败, 主题: {TOPIC}")
        time.sleep(5) # 每5秒上报一次, 模拟传感器实时采集频率


if __name__ == '__main__':
    # 启动MQTT连接并发布数据
    mqtt_client = connect_mqtt()
    mqtt_client.loop_start() # 启动客户端循环, 持续监听连接状态

```

步骤2: 运行脚本并验证¹(0.5小时)

1. 安装Python MQTT依赖库: 打开命令行 (Windows: Win+R输入cmd; Mac: 终端), 输入以下命令并回车:


```
pip install paho-mqtt
```

 (若提示“pip不是内部命令”, 需先配置Python环境变量, 可搜索“Python pip环境变量配置”快速解决)
2. 运行Python脚本: 在命令行进入脚本所在文件夹, 输入 `python car_sensor_mqtt.py` 并回车; 若显示“ 成功连接到MQTT服务器”, 则脚本启动成功;
3. 验证数据传输: 保持Postman的订阅窗口打开, 观察是否每隔5秒自动接收Python脚本上报的模拟数据; 若能稳定接收, 则“模拟传感器→MQTT服务器→云端订阅”的完整链路打通。

第1天预期成果

- Postman成功连接MQTT服务器，可手动发布/订阅数据；
- Python脚本运行正常，实现车载传感器数据自动模拟上报；
- 记录文档：通信链路验证结果、遇到的问题及解决方法（100字左右）。

三、第2天：编写核心测试用例（3小时）

核心任务：按“功能测试+异常测试+性能测试”分类，编写20条测试用例，覆盖车载传感器MQTT传输核心场景，复用黑盒测试方法（等价类划分、边界值分析、场景法）。

步骤1：创建测试用例表格（0.5小时）

1. 打开Excel/WPS，新建工作簿，命名为「车载传感器MQTT测试用例.xlsx」；
2. 在工作表1中输入以下表头，设置列宽适配内容：
用例编号测试模块测试场景前置条件输入数据/操作步骤预期结果实际结果测试状态优先级
TC_MQTT_001MQTT功能测试正常数据上报（QoS1）1. MQTT服务器连接正常；2. Postman已订阅主题car/sensor/0011. 运行Python脚本，发送正常数据（temp=25.5，humidity=45.2，pm25=32）；2. QoS等级设为1Postman成功接收数据，数据与上报内容完全一致待测试待测试高

步骤2：按分类编写测试用例（2.5小时）

要求：共20条，其中功能测试10条、异常测试8条、性能测试2条，直接按以下场景填写，确保步骤可复现、预期结果明确。

分类1：功能测试（10条，高优先级，验证正常传输场景）

1. TC_MQTT_001：正常数据上报（QoS0）
 - 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001，QoS0
 - 步骤：运行Python脚本，修改QoS为0，发送正常数据
 - 预期结果：Postman成功接收数据（允许丢失，多次运行无丢包即可）
2. TC_MQTT_002：正常数据上报（QoS1）（已填模板）
3. TC_MQTT_003：正常数据上报（QoS2）
 - 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001，QoS2
 - 步骤：运行Python脚本，修改QoS为2，发送正常数据
 - 预期结果：Postman成功接收数据，无重复、无丢失
4. TC_MQTT_004：设备ID变更后上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/002
- 步骤：修改Python脚本device_id为car_002、主题为car/sensor/002，运行脚本发送数据
- 预期结果：Postman成功接收car_002的上报数据

5. TC_MQTT_005：传感器数据正常边界值上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：修改Python脚本，发送边界值数据（temp=-40°C、85°C；pm25=0、500）
- 预期结果：Postman成功接收数据，数据值与上报一致

6. TC_MQTT_006：订阅端断连重连后接收补发数据（QoS1）

- 前置条件：MQTT服务器连接正常；Python脚本已启动（QoS1）
- 步骤：1. 断开Postman订阅连接；2. 等待10秒（期间脚本持续上报）；3. 重新连接订阅
- 预期结果：重新连接后，Postman能接收断连期间的上报数据

7. TC_MQTT_007：多主题同时上报

- 前置条件：MQTT服务器连接正常；Postman同时订阅car/sensor/001和car/sensor/002
- 步骤：启动两个Python脚本，分别上报car/sensor/001和car/sensor/002
- 预期结果：Postman两个主题均能正常接收数据

8. TC_MQTT_008：单主题多设备上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：启动两个Python脚本，均上报car/sensor/001（device_id分别为car_001、car_003）
- 预期结果：Postman能接收两个设备的上报数据，可区分device_id

9. TC_MQTT_009：数据字段完整上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：运行Python脚本，发送包含device_id、temp、humidity、pm25、timestamp的完整数据
- 预期结果：Postman接收的数据包含所有字段，无缺失

10. TC_MQTT_010：数据字段部分缺失上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001

- 步骤：修改Python脚本，删除timestamp字段，发送数据
- 预期结果：Postman能接收数据，缺失字段无默认值

分类2：异常测试（8条，中优先级，验证异常处理能力）

1. TC_MQTT_011：数据格式错误上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：修改Python脚本消息体为错误JSON（如缺少引号：
{device_id:car_001,temp:25.5}），发送数据
- 预期结果：Postman接收数据为乱码或无法解析，系统无崩溃

2. TC_MQTT_012：主题不匹配上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：修改Python脚本主题为car/sensor/000，发送数据
- 预期结果：Postman无法接收数据

3. TC_MQTT_013：QoS等级不匹配

- 前置条件：MQTT服务器连接正常；Postman订阅QoS0，主题car/sensor/001
- 步骤：修改Python脚本发布QoS2，发送数据
- 预期结果：Postman接收数据，按QoS0等级（最多一次）传输

4. TC_MQTT_014：服务器断连后重连上报

- 前置条件：Python脚本已启动（QoS1），Postman订阅正常
- 步骤：1. 手动断开Postman与MQTT服务器的连接；2. 等待5秒；3. 重新连接订阅
- 预期结果：重新连接后，Python脚本仍能正常上报，Postman恢复接收

5. TC_MQTT_015：传感器数据超出范围上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：修改Python脚本，发送超出范围数据（temp=90°C、pm25=600）
- 预期结果：Postman能接收数据（传输层不校验业务逻辑）

6. TC_MQTT_016：空数据上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：修改Python脚本消息体为{}，发送数据

- 预期结果：Postman接收空数据，无报错

7. TC_MQTT_017: QoS2重复上报去重验证

- 前置条件：MQTT服务器连接正常；Postman订阅QoS2，主题car/sensor/001
- 步骤：修改Python脚本，连续2次发送完全相同的数据（QoS2）
- 预期结果：Postman仅接收1条数据，无重复

8. TC_MQTT_018: 高频上报验证

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：修改Python脚本上报间隔为1秒，运行1分钟
- 预期结果：Postman能稳定接收数据，无大量丢包

分类3：性能测试（2条，中优先级，验证并发/持续传输能力）

1. TC_MQTT_019: 5个设备并发上报

- 前置条件：MQTT服务器连接正常；Postman同时订阅car/sensor/001~005
- 步骤：复制5份Python脚本，分别修改device_id为car_001~005、主题为对应car/sensor/001~005，同时运行
- 预期结果：Postman 5个主题均能正常接收数据，无明显延迟

2. TC_MQTT_020: 10分钟持续上报

- 前置条件：MQTT服务器连接正常；Postman订阅主题car/sensor/001
- 步骤：运行Python脚本，持续上报10分钟，期间观察接收状态
- 预期结果：全程无断连，数据接收成功率 $\geq 98\%$ （允许少量丢包）

第2天预期成果

- 完成「车载传感器MQTT测试用例.xlsx」，包含20条规范用例；
- 用例要求：前置条件清晰、操作步骤可复现、预期结果明确，覆盖核心测试场景。

四、第3天：执行测试 + 整理项目成果（3.5小时）

核心任务：执行20条测试用例，记录结果，模拟发现缺陷，整理1页项目总结，所有成果可直接用于实习简历。

上午（2小时）：执行测试用例，记录结果与缺陷

步骤1：执行测试用例（1.5小时）

1. 打开Postman（保持MQTT订阅配置）、Python脚本（按用例需求修改参数）、测试用例Excel；
2. 逐条执行20条用例，按以下要求记录：
 - 实际结果：填写与预期结果一致/不一致的具体情况（如“接收数据重复”“无法接收数据”）；
 - 测试状态：通过/失败/阻塞（阻塞需注明原因，如“服务器异常”）；
 - 重点关注：QoS2重复上报、数据格式错误、5设备并发这3个易出现“失败”的用例，确保能模拟发现缺陷。
3. 参考执行结果：建议设置17条“通过”、3条“失败”（模拟真实测试场景），失败用例推荐：
 - TC_MQTT_017（QoS2重复上报，实际结果：接收2条重复数据）；
 - TC_MQTT_011（数据格式错误，实际结果：接收乱码但无错误提示）；
 - TC_MQTT_019（5设备并发，实际结果：偶尔丢包，成功率92%）。

步骤2：记录缺陷（0.5小时）

1. 在Excel工作簿中新建工作表2，命名为「缺陷记录」；
2. 输入以下表头，填写3个失败用例对应的缺陷信息（符合禅道缺陷报告规范）：

缺陷编号 缺陷标题 所属模块 严重程度 优先级 复现步骤 预期结果 实际结果 测试环境

BUG_001 QoS2等级重复上报导致数据重复接收 MQTT协议测试 严重（影响数据准确性） 高（需优先修复） 1. Postman订阅QoS2，主题car/sensor/001；2. Python脚本发布QoS2，连续2次发送相同数据；3. 查看Postman接收记录仅接收1条数据，无重复接收2条相同数据，导致数据冗余公共MQTT服务器（broker.emqx.io:1883）、Postman 10.24.0、Python 3.9

BUG_002 数据格式错误 时订阅端无错误提示 MQTT协议测试 一般（不影响核心功能） 中（下一版本修复） 1. Postman订阅主题car/sensor/001；2. Python脚本发送错误JSON格式数据（缺少引号）；3. 查看接收结果订阅端提示“数据格式错误”接收乱码数据，无任何错误提示，不利于问题定位公共MQTT服务器（broker.emqx.io:1883）、Postman 10.24.0、Python 3.9

BUG_003 5设备并发上报时存在丢包现象 MQTT性能测试 严重（影响高并发场景可用性） 高（需优先修复） 1. Postman同时订阅car/sensor/001~005；2. 5个Python脚本同时上报对应主题；3. 运行1分钟，统计接收数据量接收成功率100%，无丢包接收成功率92%，丢失4条数据，高并发场景下可用性不足公共MQTT服务器（broker.emqx.io:1883）、Postman 10.24.0、Python 3.9

下午/晚上（1.5小时）：整理项目总结（简历直接复用）

步骤1：编写1页项目总结（1小时）

1. 打开Word/WPS，新建文档，命名为「车载传感器MQTT测试项目总结.docx」；

2. 按以下模板填写（1页篇幅，简洁突出核心成果）：

车载环境传感器MQTT协议测试项目总结一、项目背景针对车联网车载环境传感器（温湿度、PM2.5）数据传输场景，基于MQTT轻量级协议设计测试系统，验证数据传输的功能正确性、异常处理能力和并发性能，为车联网传感器测试提供可落地的方案参考。二、测试范围三、核心成果四、技术栈MQTT协议、Postman测试工具、Python编程（paho-mqtt库）、黑盒测试方法、Excel用例与缺陷管理。

- 协议：MQTT（覆盖QoS0/1/2三个服务质量等级）；
- 测试类型：功能测试（正常上报、多设备/多主题交互）、异常测试（格式错误、主题不匹配、服务器断连）、性能测试（5设备并发、10分钟持续上报）；
- 工具：Postman（MQTT协议测试）、Python（传感器数据模拟）、Excel（用例管理与缺陷记录）。
- 用例设计：采用黑盒测试方法（等价类、边界值、场景法），编写20条测试用例，覆盖车载传感器MQTT传输核心场景；
- 测试执行：完成所有用例执行，测试通过率85%，发现3个核心缺陷（QoS2数据重复、格式错误无提示、高并发丢包），均提交规范缺陷报告；
- 自动化能力：实现Python自动化模拟车载传感器数据上报，支持多设备并发、高频上报等测试场景，提升测试效率。

步骤2：整理简历项目经验描述（0.5小时）

1. 从项目总结中提炼核心信息，整理为简历适配的项目经验描述（3-4行，突出技能与成果）：

车载环境传感器MQTT协议测试项目 | 个人项目 | 2025.12

- 基于MQTT协议搭建车载传感器测试环境，用Python开发自动化脚本模拟温湿度、PM2.5数据实时上报，Postman实现云端订阅验证；
- 采用黑盒测试方法设计20条测试用例，覆盖功能、异常、性能三大类场景，完成全量用例执行，测试通过率85%；
- 发现3个核心缺陷（QoS2数据重复、高并发丢包等），编写规范缺陷报告，输出1页项目总结，形成完整的测试闭环；
- 技术栈：MQTT、Postman、Python、黑盒测试、Excel用例管理。

2. 将项目总结文档、测试用例Excel、缺陷记录Excel整理为“车载传感器MQTT测试项目成果包”，用于面试展示。

第3天预期成果

- 完成测试的「车载传感器MQTT测试用例.xlsx」（含实际结果、测试状态）；
- 规范的「缺陷记录」Excel（3个核心缺陷）；
- 1页「项目总结」Word文档；
- 简历适配的项目经验描述（可直接粘贴使用）。

五、3天总成果（实习简历直接复用）

1. 项目经验描述（简历核心素材）；
2. 可展示材料：测试用例文档、缺陷记录、项目总结（面试时可主动提及“有完整的测试产出物可展示”）；
3. 技能验证：已熟练掌握MQTT协议测试、Postman实操、Python模拟开发、黑盒测试用例设计，完全匹配车联网测试实习岗位需求。

六、后续升级方向（毕设/考研复试用）

在3天成果基础上，通过以下升级可满足毕设工作量与技术深度要求，适配考研复试“项目递进性”考察点：

1. 环境升级：搭建本地EMQ X MQTT服务器（替代公共服务器），增加服务器性能监控模块（统计并发连接数、消息延迟、丢包率）；
2. 自动化升级：用Python+pytest框架编写自动化测试脚本，实现用例自动执行、结果自动断言、测试报告自动生成；
3. 功能升级：增加CAN总线数据交互测试（用Python-CAN库解析车载CAN报文，实现“CAN总线数据→MQTT服务器”的跨协议融合测试）；
4. 可视化升级：用Flask+ECharts开发测试结果可视化 dashboard，实时展示用例通过率、缺陷分布、性能指标趋势；
5. 硬件升级（可选，100元内）：接入ESP32开发板+DHT11温湿度传感器，采集真实环境数据，替换Python模拟数据，实现“软硬结合”的测试系统。

说明：本计划完全基于软件模拟实现，无硬件需求，所有步骤可直接落地；3天成果可直接用于实习简历投递与面试，后续升级方向可根据毕设要求和考研复试时间灵活推进。

（注：文档部分内容可能由 AI 生成）