# 第 9 章 自建云平台及设备接入

在前面的学习过程中，我们项目上所用的都是第三方为我们提供的云平台，并且在云平台上我们也是使用 MQTT 协议进行数据传输，我想大家已经对 MQTT 协议有比较深入的了解，当然还有不了解的我们在此不再进行知识的讲解，请大家自行搜索相关资料进行学习了解。本章节主要是教大家实践动手的能力，在自己的云服务器或者是在本地个人 PC 机上搭建自己的 MQTT 服务器，并接入设备进行数据传输，让大家直观的了解这一传输过程！

## 9.1 MQTT 服务器搭建

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输协议），是一种基于发布/订阅（publish/subscribe）模式的"轻量级"通讯协议，该协议构建于 TCP/IP 协议上，由 IBM 在 1999 年发布。MQTT 最大优点在于，可以以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。作为一种低开销、低带宽占用的即时通讯协议，使其在物联网、小型设备、移动应用等方面有较广泛的应用。MQTT 协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。

MQTT 服务器也就是 MQTT Broker，目前有很多主流的热门版本比如：EMQ、HiveMQ、VerneMQ、ActiveMQ、Mosquitto，前三个版本包含开源和商业版，我们可以使用开源的版本进行学习，后两个版本是开源的，本书选择的是 Mosquitto 版本，开发团队是 Eclipse 开源社区，这个版本已经足够大家使用了。接下来将讲解在 windows 和 Ubuntu 下部署自己的 MQTT 服务器，不提供源码解读，只是供大家部署学习和使用。

### 9.1.1 windows 下部署 MQTT 服务器

在 Windows 下我们直接到官方网站（https://mosquitto.org/download）下载安装包进行安装配置即可，下面是详细的安装和配置步骤：

步骤一：找到我们从官网下载的安装包，双击打开，点击下一步进入该界面，继续点击 next。

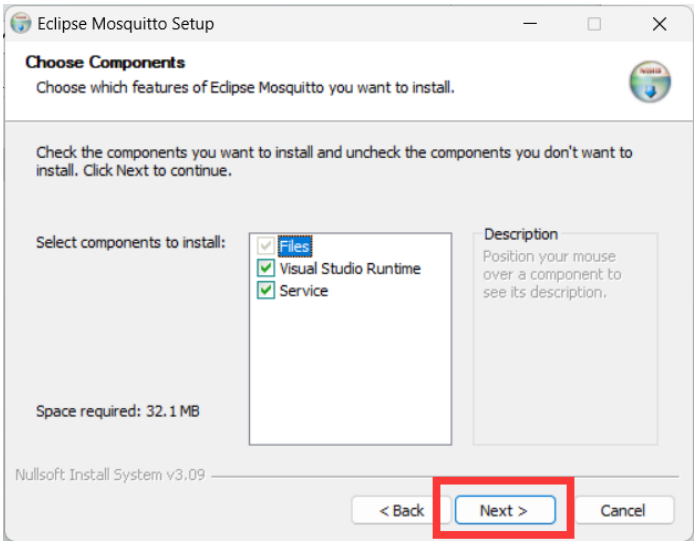


图 9-1 步骤一

步骤二：安装的时候我们尽量重新选择安装路径，保证等会我们能够找到安装位置，进行服务器的配置，选择后选择 install 后自动安装，安装完成点击 finish。

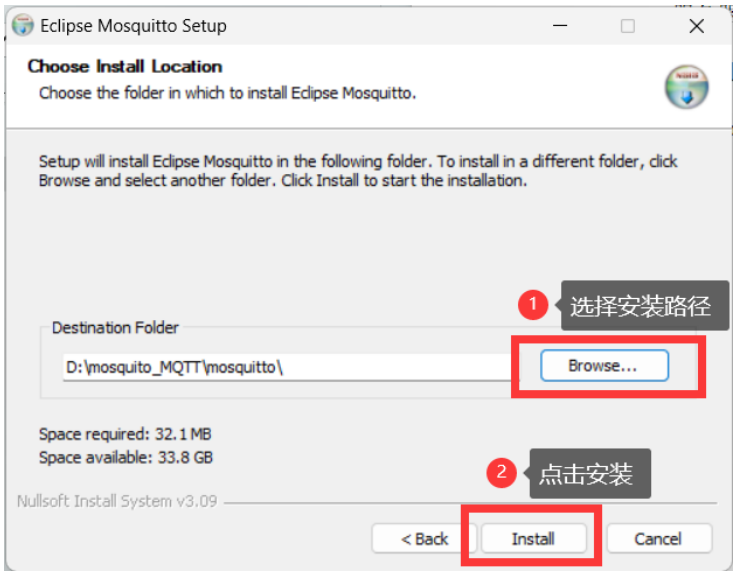


图 9-2 步骤二

步骤三：找到刚才安装的路径下，找到如图所示的 mosquitto.conf 文件，接下来用记事本打开该文件，我们在该文件下对端口、IP、用户密码的保存路径等进行配置。

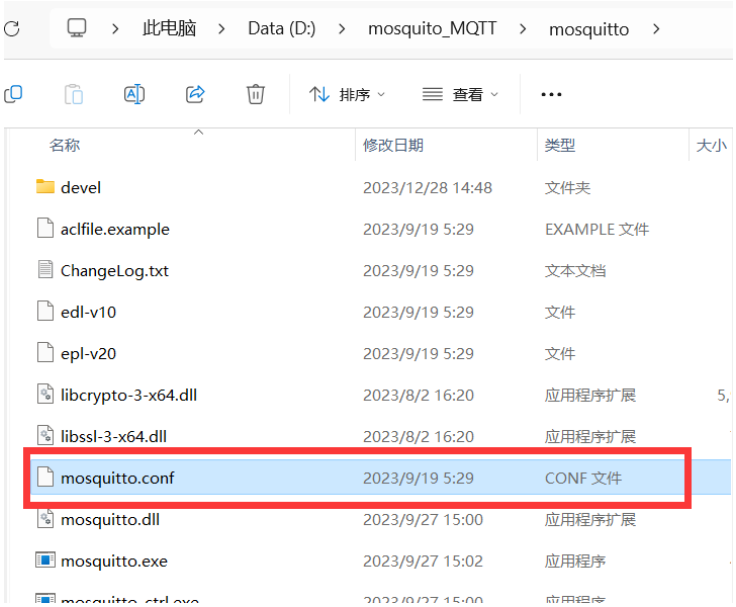


图 9-3 步骤三

步骤四：该步骤就是对服务器的一些配置，该步骤的有些配置是不一定要做的，我们可以根据自己的需要进行选择，我们打开配置文件的时候里面会有很多内容，这些内容是给我们的一些配置案例，我们可以直接删除或者保留内容，在文件的最底部添加我们的配置，listener 1883 127.0.0.1 是配置我们的端口号和 IP 地址，这里我们端口号使用 1883，这个是 MQTT 协议的固定端口，IP 地址使用本地回环地址，我们也可以查看当前局域网的 IP 地址进行配置，allow_anonymous true 配置的是是否允许匿名用户连接，这里我们配置为 true，不需要用户名和密码就可以进行连接，如果配置为 false 我们就需要配置密码的存储文件 password_file pwfile.example，这个 pwfile.example 文件默认创建在安装根路径下。这三个配置比较常用，如果感兴趣的可以深入阅读其他的配置。

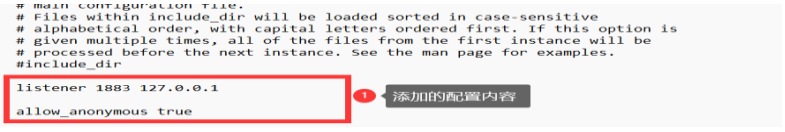


图 9-4 步骤四

如果我们配置 allow_anonymous 为 false 的时候就必须添加 password_file 配置，添加这些配置之后我们就需要使用 mosquitto_passwd.exe 添加用户、密码。

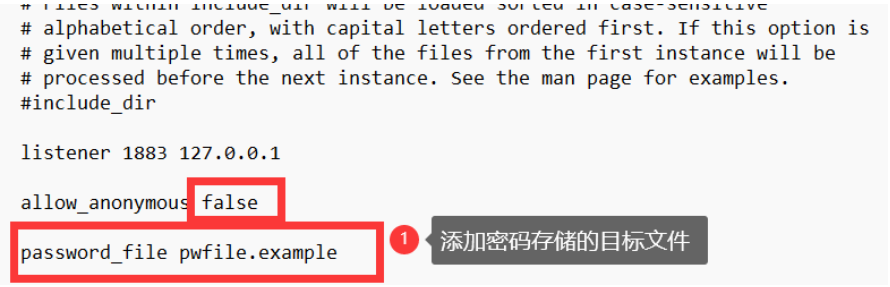


图 9-5 步骤四添加密码存储目标文件

配置匿名用户不可连接之后我们使用程序添加用户和密码，命令：.\mosquitto_passwd.exe -c pwfile.example myservice。


图 9-6 添加用户和密码

到此所有的配置已经完成，我们使用配置文件启动服务之后就可以使用客户端工具进行测试啦！

步骤五：使用命令行启动服务，命令：.\mosquitto -c mosquitto.conf -v。如图服务启动成功，我们使用客户端测试工具测试服务器是否工作正常。

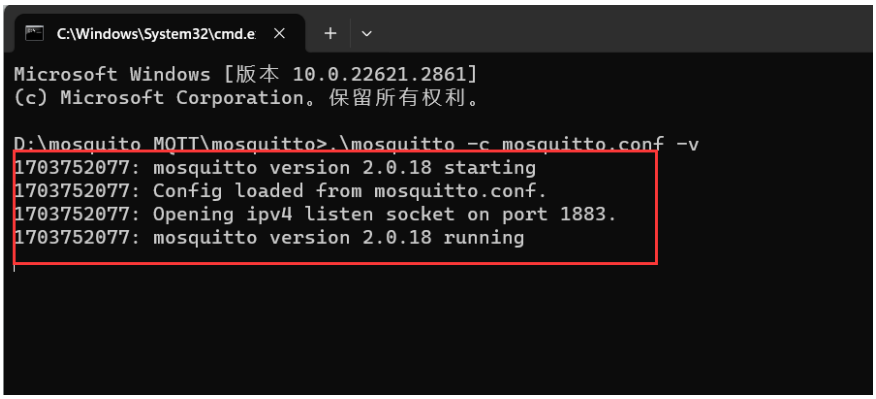


图 9-7 服务启动

步骤六：打开 mqtt 客户端测试工具 MQTT.fx 进行测试。

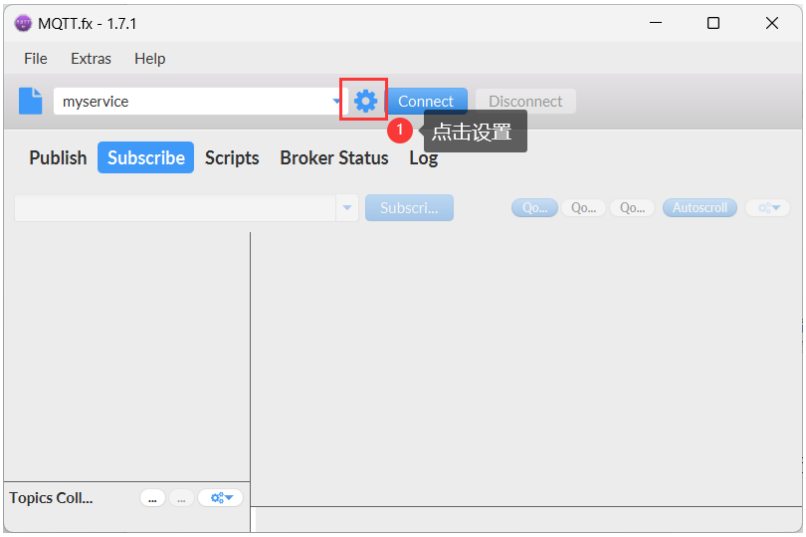


图 9-8 测试步骤一

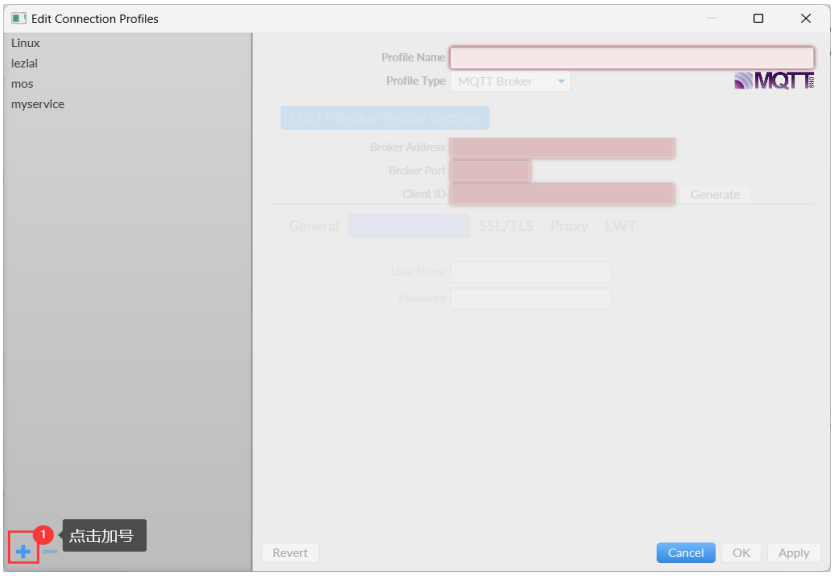


图 9-9 测试步骤二

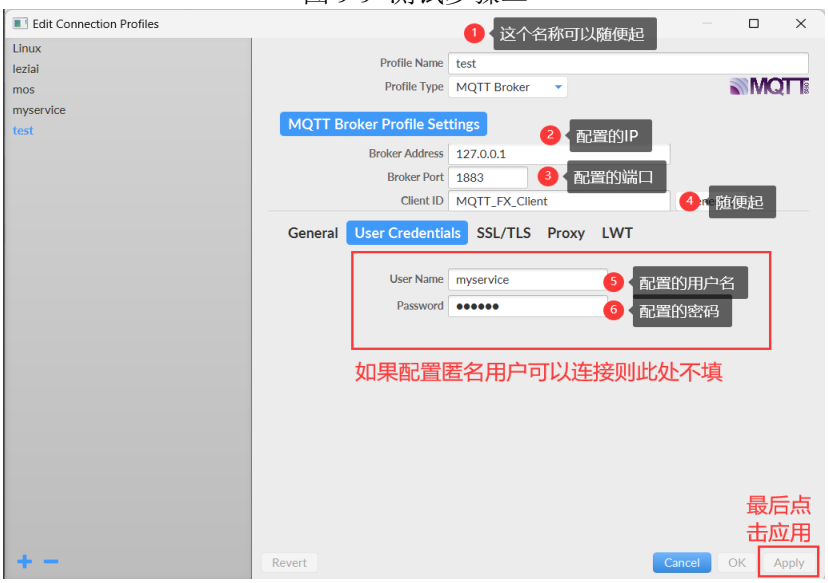


图 9-10 测试步骤三

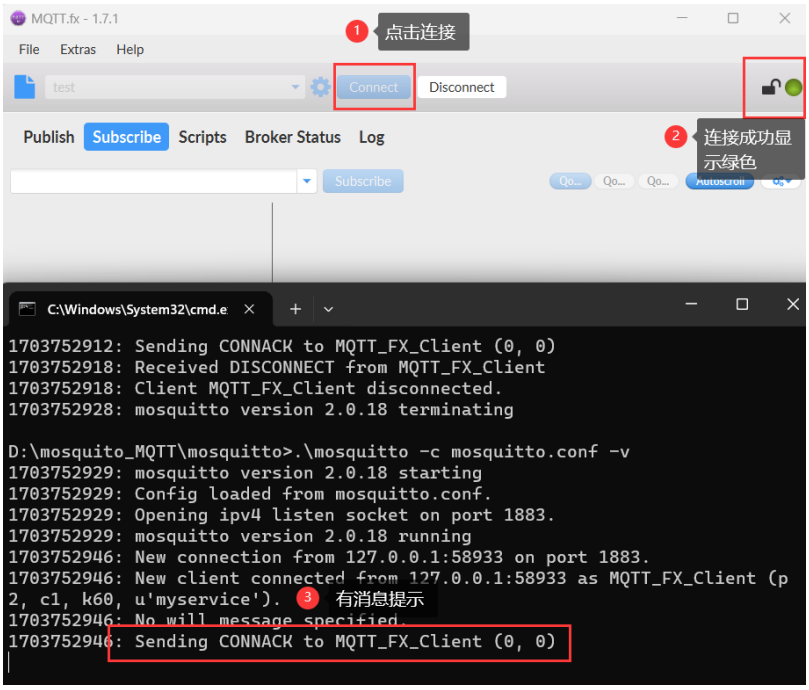


图 9-11 测试步骤四

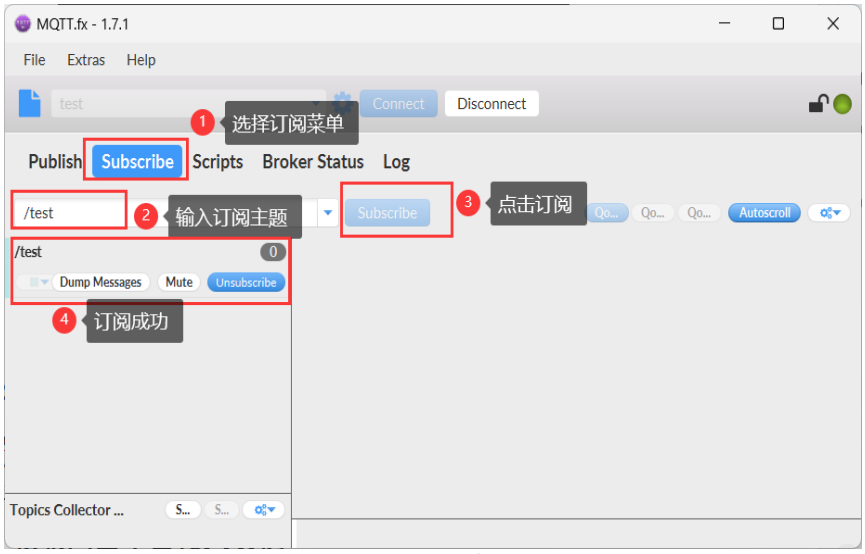


图 9-11 测试步骤五

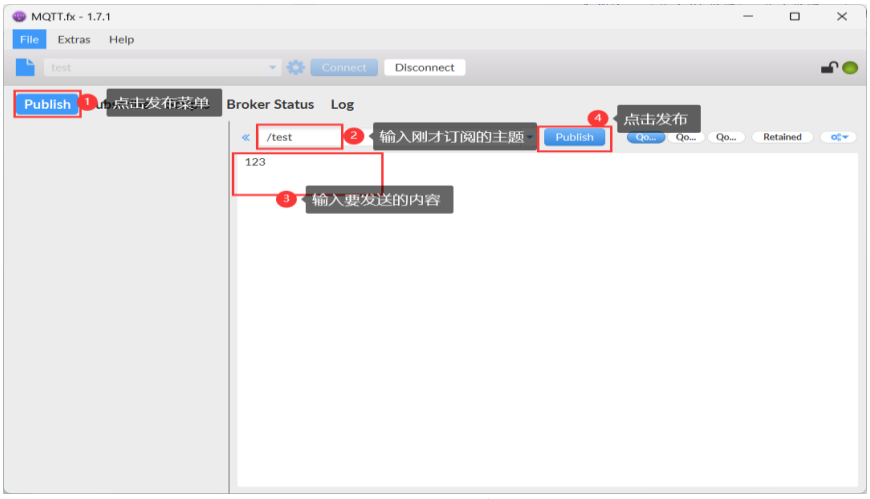


图 9-12 测试步骤六

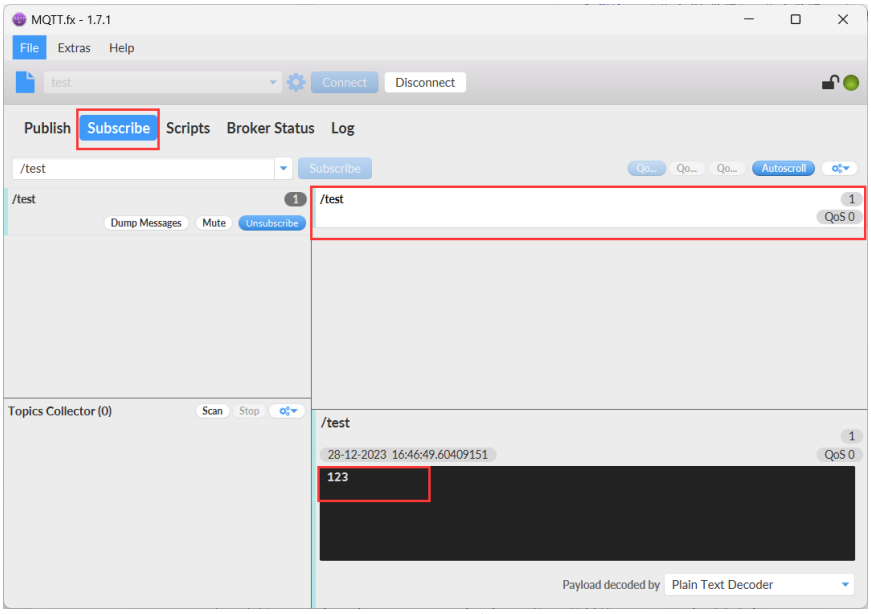


图 9-12 测试步骤七

至此我们已经能够成功连接服务器，并且利用服务器传输消息，也就说明我们的服务器搭建成功了，该服务器可以在局域网下面使用。

## 9.1.2 Ubuntu 下部署 MQTT 服务器

前面我们已经在 Windows 下面成功部署了我们的 MQTT 服务器，现在我们在虚拟机 Ubuntu 上面部署一下我们的本地服务器。在 Ubuntu 下的部署有两种方法，一种是直接使用 apt-get install 直接下载安装工具包，另一种是到 mosquitto 官网下载源码使用 make 进行编译安装，为了避免一些不必要的麻烦，我们直接使用 Ubuntu 的软件管理工具下载安装包进行安装。

步骤一：先使用 sudo apt-get update 命令更新一下。

步骤二：使用 sudo apt-get install mosquitto 安装 mosquitto。

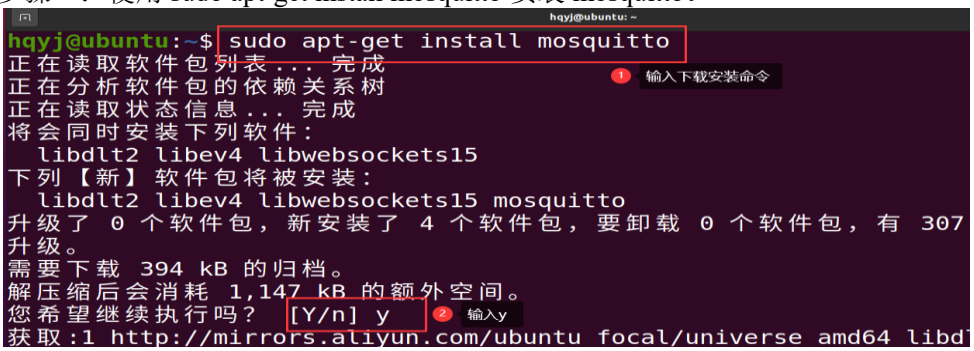


图 9-13 步骤二

步骤三：使用 cd /etc/mosquitto/命令进入目录下，再使用 ls 命令，我们就可以看到和 Windows 下一样的配置文件 mosquitto.conf，但是在 Linux 下这个文件是主配置文件，我们不进行修改，我们的配置文件都放在 conf.d 目录下，名字以.conf 结尾都可以，我们使用 cd conf.d 命令进入目录。

步骤四：在步骤三进入的目录下使用 sudo vim mytest.conf 命令创建我们自己的配置文件，输入的配置内容和 Windows 下配置步骤四相同，这里不在赘述，最后使用:wq 保存。要保证在/etc/mosquitto 目录下存在 pwfile 文件，如果不存在的话，请使用 sudo touch /etc/mosquitto/pwfile 命令进行创建。

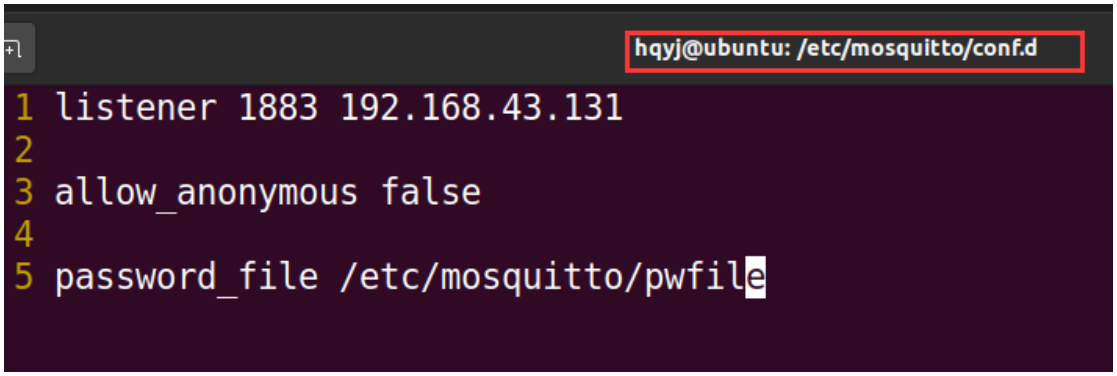


图 9-14 步骤四

如果配置非匿名用户不可接入，我们任然需要使用命令创建用户名和密码，和 Windows 下一样，命令如下 sudo mosquitto_passwd -c /etc/mosquitto/pwfile mytestuser 。

```
hqyj@ubuntu:/etc/mosquitto$ sudo mosquitto_passwd -c /etc/mosquitto/pwfile mytestuser
Password:
Reenter password:
hqyj@ubuntu:/etc/mosquitto$
```

图 9-15 创建用户

步骤五：启动 mosquitto 服务器，命令 sudo service mosquitto start。

```
hqyj@ubuntu:/etc/mosquitto$ sudo service mosquitto start
hqyj@ubuntu:/etc/mosquitto$
```

图 9-16 启动服务

至此启动成功，没有报错，整个服务器就已经搭建成功了，我们测试服务器的方法和 Windows 一样，我们只要虚拟机能上网，就说明主机和虚拟机处于同一网段，我们在 Windows 下使用同一个工具进行测试就可以了，修改的方法不在赘述，还没掌握 MQTT.fx 客户端使用方法的同学，请再次阅读 Windows 下 MQTT 服务器部署的章节。

## 11.2 自建云平台接入实例

我们在嵌入式设备 STM32 和 QT 应用程序利用别人编写的开源的 MQTT 客户端代码框架，写我们自己的代码，连接我们搭建的服务器，即可完成在局域网下的 MQTT 通信。下面就以 STM32F103ZET6 单片机模拟被控的设备，iMUX6ULL 微控制器运行 QT 程序下发控制指令，教会大家连接 MQTT 服务器，实现无线数据传输和控制。

### 11.2.1 代码预备

使用 STM32 连接服务器之前，我们需要保证单片机能够连接网络，本书中我们使用单片机联网的模块就是 ESP8266，对于该模块不熟悉的在此不再进行讲解，自行查阅资料，在本章节中因为我们的服务器是在本地，只需要连接上本地局域网就行，一般使用手机开移动热点的方式，供单片机连接无线网，接下来我们直接开始干货内容。

**第一部分：**

设备准备和连接，单片机选择 STM32F103ZET6，联网模块 ESP8266，这里的单片机大家可以根据手上的资源进行选择，不必选择相同的型号，我们需要学会编程方法和思路。本章节单片机和 ESP8266 的通信我使用的是 UART3，因为我的单片机已经预留出了 ESP8266 的接口。

**第二部分：**

项目编程实现，在第一部分我们单片机需要与 ESP8266 通信使用串口，我们就需要编写串口通信相关的代码。

**（1）串口初始化。**

```
void Usart3_Init(unsigned int baud)
{
```

```
    GPIO_InitTypeDef gpio_initstruct;
    USART_InitTypeDef usart_initstruct;
    NVIC_InitTypeDef nvic_initstruct;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);

    //PB10    TXD
    gpio_initstruct.GPIO_Mode = GPIO_Mode_AF_PP;
    gpio_initstruct.GPIO_Pin = GPIO_Pin_10;
    gpio_initstruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &gpio_initstruct);

    //PB11    RXD
    gpio_initstruct.GPIO_Mode = GPIO_Mode_IPU;
    gpio_initstruct.GPIO_Pin = GPIO_Pin_11;
    gpio_initstruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &gpio_initstruct);

    //片选
    gpio_initstruct.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_15;
    gpio_initstruct.GPIO_Mode = GPIO_Mode_Out_PP;
    gpio_initstruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init( GPIOA, &gpio_initstruct );  /* 拉低 WiFi 模块的片选引脚  */
    GPIO_SetBits( GPIOA, GPIO_Pin_4 );      /* 拉高 WiFi 模块的复位重启引脚    */
    GPIO_SetBits( GPIOA, GPIO_Pin_15 );

    usart_initstruct.USART_BaudRate = baud;
    //无硬件流控
    usart_initstruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    //接收和发送
    usart_initstruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    //无校验
    usart_initstruct.USART_Parity = USART_Parity_No;
    //1 位停止位
    usart_initstruct.USART_StopBits = USART_StopBits_1;
    //8 位数据位
    usart_initstruct.USART_WordLength = USART_WordLength_8b;
    //初始化
    USART_Init(USART3, &usart_initstruct);
    //使能串口
    USART_Cmd(USART3, ENABLE);
    //使能接收中断
    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);

    nvic_initstruct.NVIC_IRQChannel = USART3_IRQn;
    nvic_initstruct.NVIC_IRQChannelCmd = ENABLE;
    nvic_initstruct.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_initstruct.NVIC_IRQChannelSubPriority = 1;
    NVIC_Init(&nvic_initstruct);
}
```

**（2）为了后续方便发送与 ESP8266 交互的 AT 指令和 MQTT 数据包，单独编写了一个使用串口发送字符串的函数。**

```
//串口数据发送
void Usart_SendString(USART_TypeDef *USARTx, unsigned char *str, unsigned short len)
{
```

```
    unsigned short count = 0;

    for(; count < len; count++)
    {
        USART_SendData(USARTx, *str++);  //发送数据
        while(USART_GetFlagStatus(USARTx, USART_FLAG_TC) == RESET);//等待发送
    }
}
```

**（3）利用（2）封装的函数我们再次封装一个函数用于 AT 指令交互。**

```
//发送 AT 指令
_Bool ESP8266_SendCmd(char *cmd, char *res)
{
    unsigned char timeOut = 200;

    Usart_SendString(USART3, (unsigned char *)cmd, strlen((const char *)cmd));

    while(timeOut--)
    {
        if(ESP8266_WaitRecive() == REV_OK)//如果收到数据
        {
            if(strstr((const char *)esp8266_buf, res) != NULL)//如果检索到关键词
            {
                ESP8266_Clear();//清空缓存
                return 0;
            }
        }
        delay_ms(10);//延时 10ms
    }
    return 1；
}
```

**（4）AT 指令交互我们也要接收 ESP8266 回传的数据，我们使用串口中断接收，前面我们已经初始化过串口中断了，这部分是串口中断接收数据的代码。**

```
void USART3_IRQHandler(void)
{
    if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET) //接收中断
    {
        if(esp8266_cnt >= sizeof(esp8266_buf))       esp8266_cnt = 0;
        esp8266_buf[esp8266_cnt++] = USART3->DR;
        USART_ClearFlag(USART3, USART_FLAG_RXNE);
    }
}
```

**（5）前面对串口收发数据的一些重要代码编写完成之后，我们就可以编写发送 MQTT 数据包的代码。**

```
    void ESP8266_SendData(unsigned char *data, unsigned short len)
    {

        char cmdBuf[32];

        ESP8266_Clear();                                    //清空接收缓存
        sprintf(cmdBuf, "AT+CIPSEND=%d\r\n", len);          //发送命令
        if(!ESP8266_SendCmd(cmdBuf, ">"))                   //收到‘>’时可以发送数据
        {
            Usart_SendString(USART3, data, len);        //发送设备连接请求数据
```

```
        }
    }
```

至此我们在 STM32 上使用 MQTT 协议通信的一些主要代码就已经编写完成了，上面只是描述了一些关键代码，要使整个项目完全运转，仅仅凭借这些是不够的，还需要阅读相关源码进行学习，了解 MQTT 协议，下面就开始讲解关于 MQTT 连接服务器的代码。

### 11.2.2 STM32 连接服务器

### 11.2.3 QT 程序连接服务器

### 11.2.4 QT 程序下发控制指令

### 11.2.5 STM32 上报数据