工业数据采集技术

目 录

1. 工业数据采集系统概述

1.1. 工业数据采集现状

随着信息技术的飞速发展，工业制造技术与信息技术的深度融合是发展趋势，“工业互联网”时代已经到来。工业互联网旨在实现智能化生产，通过集成大量的工业控制系统、工业嵌入式终端以及工业传感器于一体，运用通信技术将它们融入到智能网络中，进而可以实现不同企业之间信息的横纵相通。

现阶段的工业技术改革更加注重智能化生产。智能化生产的核心就是工业生产过程的智能监控，而工业现场设备层的数据采集、传输和处理是实现工业生产智能监控、工业互联网的重要基础环节。但是，工业现场的设备种类繁多，各种工业总线协议并存，这也就导致了数据采集这项工作是一件非常个性化的事情，很难总结出一套放之四海而皆准的方案来。需要根据项目经验，针对不同的设备设计出经济高效的数据采集技术路线，下面列出了不同场合的数据采集方法：

1. 有上位机系统的设备

这里的上位机系统一般是设备自带的监控系统，这样的设备往往自动化程度和信息化程度都比较高。其上位机系统往往已经对设备层的数据进行了采集、存储。因此，对于这种有上位机系统的设备，首选的数据采集方案就是直接从上位机系统中获取设备的数据，而不是去和设备打交道。这样就能够把一个 OT（Operation Technology）的问题（设备层的数据采集）转化为 IT（Information Technology）的问题（两个信息系统的信息集成）。

2. 基于 TCP/IP 通讯协议的设备

对于生产现场中没有上位机系统的众多设备，我们就需要从设备所支持的通讯协议角度入手，制定数据采集方案。

对于“基于 TCP/IP 通讯协议”，比较常用的有 OPC Classic、Siemens S7、Modbus-TCP 等通用协议，也有设备厂商自行定义的私有协议。这些协议的特点是都是基于 TCP/IP 协议簇，只是应用层协议不同。采用这些协议的设备一般都会有 RJ45 的接口，也就是以太网口，都可以直接通过以太网接入到车间网络中。

对这些设备的数据采集，需要做的就是基于 TCP/UDP 协议的解析。

3. 基于非 TCP/IP 通讯协议的设备

这一类协议不属于 TCP/IP 协议簇，往往在硬件接口方面就与以太网不兼容，比如 RS-485、CAN 等。因此，对于这类设备的数据采集首先需要增加网关，将其协议转换为 TCP/IP 协议，才能接入到车间的网络中并进行协议解析。否则，在物理层面就无法实现互通，就更不要提协议解析了。这类协议比较常用的包括 Modbus-RTU、DeviceNet、CANBus 等。

4. 不具备通讯接口的设备

这一类设备输出信号通常是电压型模拟量、电流型模拟量、脉冲、开关量，通常需要采集器将信号转换成工业现场总线如 RS485/CAN，或转换成工业以太网络 RJ45 接口。这样就可以通过上述方法接入到网络中进行协议解析。这种设备的数据采集主要利用微处理器、微控制器、或 PLC 采集器对信号进行采集和通信协议开发，开放出工业现场总线协议或 TCP/IP 协议用于联网。

1.2. 工业通信技术发展趋势

工业现场采集的数据，需要通过工业通信技术实现与物理世界的交流，工业通信技术随着信息技术的发展也在不断进步。目前现场总线、工业以太网、工业无线通信是工业通信领域的三大主流技术。

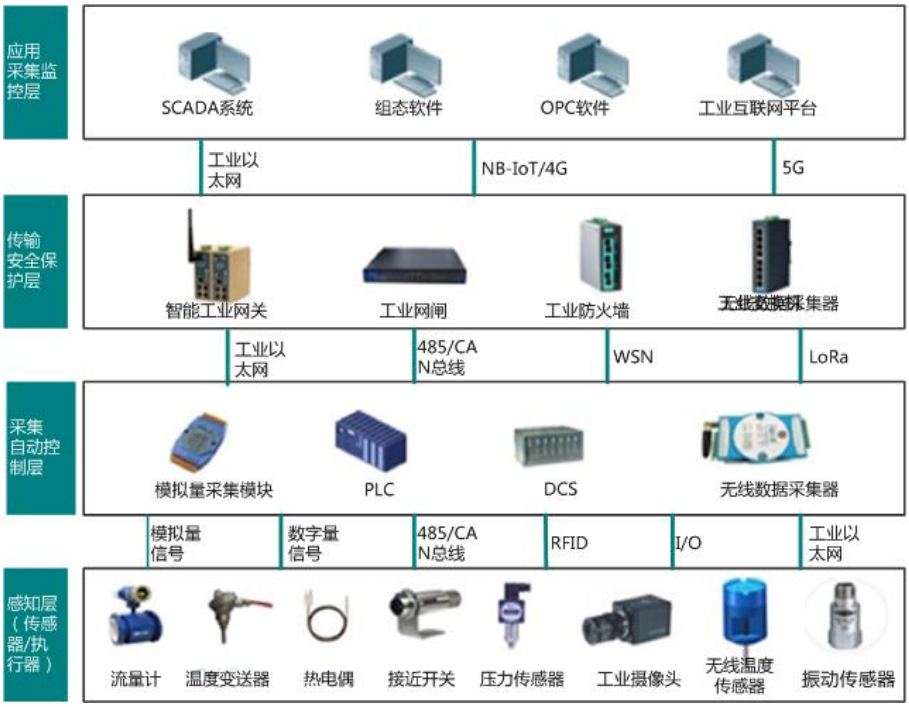
现场总线始于二十世纪 80 年代，简化了工业控制系统的安装、检修与维护，推动了工业自动化的进程。现场总线是指现场装置与控制室内的自动化装置之间起连接功能的工业数据总线，属于自动化领域中底层数据通信网络。但没有一个统一的标准。常用的现场总线有 CAN、HART、Modbus 等。对于工业现场，一般默认 Modbus 协议作为通用标准。缺点是成本高、速度低、缺少互联标准。

二十世纪 90 年代，以太网进入工业领域，出现了工业以太网技术。工业以太网比现场总线拥有更高、更稳定的传输速率因而迅速渗入到了工业领域。市场上高端的工业数据采集系统基本都涵盖了现场总线与工业以太网技术，方便了工业现场通信。但是随着智能制造的深入，对于一些恶劣，难以布线的工业现场，两种通信技术都有各自的局限性。

此时，工业无线传输技术应运而生。无线通过布线即可实现设备之间的通信，为工业现场提供了一种低成本、高可靠、高灵活的信息系统，是推动工业自动化提升和扩展的重要技术之一。德国在工业 4.0 研发白皮书中将无线技术作为工业 4.0 中通信技术研究的核心。我国提出的“中国制造 2025”计划中，也将无线通信技术列为信息技术发展的重点，因此无线通信技术对于工业现场数据传输的重要性不言而喻。比较常用的无线通信技术包括 WiFi、ZigBee、蓝牙等，适用于较小范围内无线通信，低于大型工业现场需要加入中继器，增加了系统的复杂性。以 LoRa、NB、LTE 4G/5G 为代表的远距离传输技术，具有更高的性能，更远的距离，更低的功耗，对于工业现场通信环境，会占据工业现场通信的一席之地。

1.3. 工业采集监控系统组成

工业采集监控系统架构如图所示，包含传统的工业数据采集监控系统（SCADA）、组态系统，也包括新型的工业互联网平台监控系统。



根据架构的不同，数据采集监控系统具有 C/S 架构和 B/S 架构两种模式。系统路线都是从工业现场传感器、执行装置的感知层信号采集、设备控制，通过网关或工业交换机联网，接入上位机监控系统或工业互联网平台，实现工业现场远程采集监控。但基于 B/S 架构的数据采集监控系统以海量连接性、扩展性强、随

时随地可用、可视化展示良好的特点被广泛使用。

随着工业互联网的发展，工业互联网平台发展强劲，我国已评选出了十大平台，分别是海尔 COSMOPlat、东方国信 Cloudiip、用友网络用友精智、树根互联根云、航天云网 INDICS、浪潮 IN-cloud、华为 fusionPlant、工业富联 BEACON、阿里云 supET、徐工信息汉云。其中 Cloudiip 具备近 200 个可复用的微服务，包括高铁云、工业锅炉云、冶金云、水电云、风电云、空压云、能源管理云、资产管理云、热网云等 10 个工业互联网子平台，形成工业 APP 超过 300 个，为产业生产过程感知层数据采集、数据存储及处理、数据分析及故障预测等数字化转型保驾护航。因此使用“工业互联网平台作为数据采集监控系统”也是我们本系统的首选。

1.4. 数据采集监控系统的应用

工业互联网数据采集系统在生活场景中有非常好的应用，比如电力方面，有电压电流数据实时监控报警、城市电网、路灯控制；能源方面，有煤矿、石油、天然气、油田等数据采集、供暖系统监控；交通方面，有机动车辆、车牌抓拍监控、交通灯控制；环保方面，有实时数据采集自来水，污水管道、井盖、泵站与水厂实时监控维护；流水线加工方面，有工件到位检测、自动分拣、设备运行状态监测、故障诊断等。

综上所述，掌握常用的数据采集、传输技术、平台运维技术，对于工业互联网应用系统建设非常必要。

1.5. 工业数据采集实训台简介

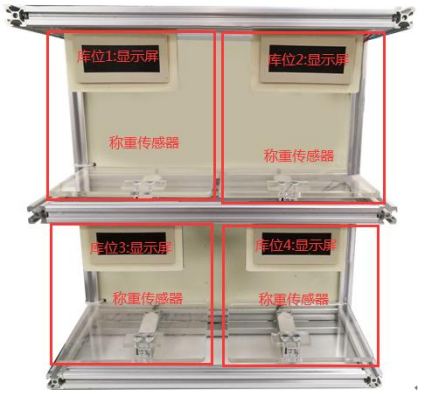
工业数据采集实训台，就是针对工业互联网操作技术（OT）和信息技术（IT），从工程布线、信号采集、数据传输、平台运维等方面，提供多种应用案例供学习训练的平台。通过训练，让学生具备工业数据采集监控系统的架构设计、设备选型、综合布线、网络通信以及应用开发等技能。

实训台感知层集成工业现场总线技术，CAN 总线与 RS485 总线，可连接

0-5V/0-10V/4-20mA/0-20mA 模拟量、开关量、数字量等工业级传感器及继电器。网关采用 Cortex-A9 四核处理器，与传感终端互联互通，支持多种感知层协议解析，可通过拓扑形式展示传感网结构和传感器采样，可通过 MQTT、HTTP 协议接入云服务平台；WEB 端云服务平台支持本地化部署或云端部署，可快速搭建行业应用系统，如智慧工厂、智慧城市、车载控制、仓储管理、安防监控、农田水利等；支持 API 接口 Web 应用开发和 Android 应用开发。

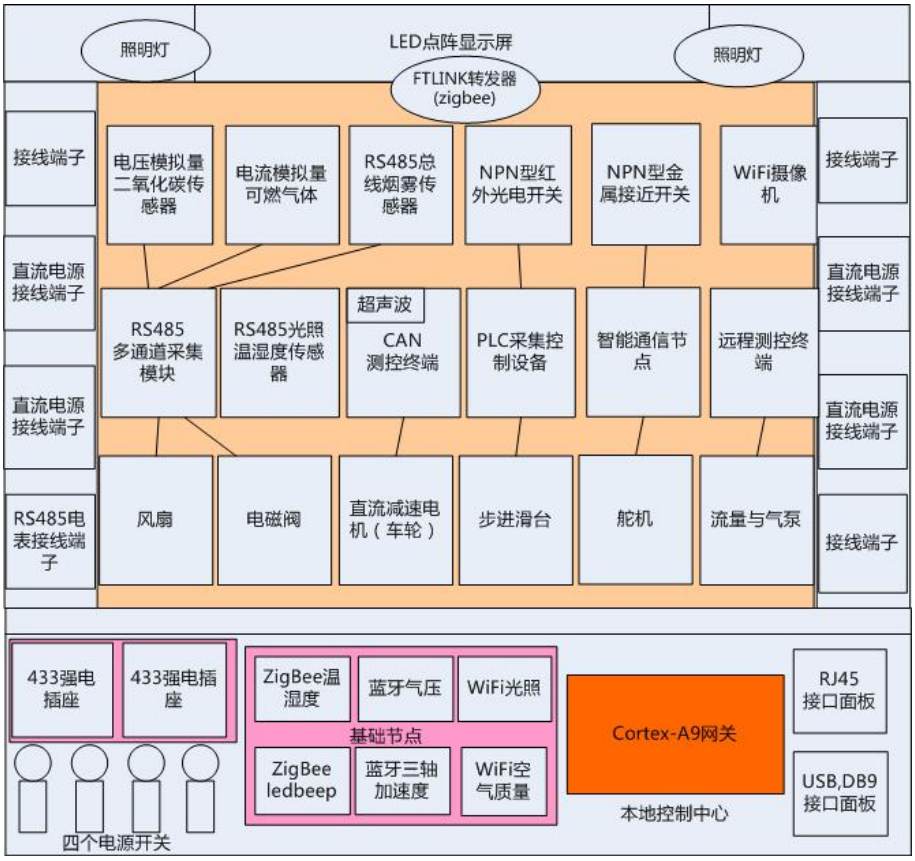
实训台由工业数据采集实训台和智能货架实训系统组成。

工业数据采集实训台实物图



智能货架系统实物图

实训台上硬件设备类型与布局，如图所示。



实训台网板的第一行安装的是工业常用传感器，信号接口类型覆盖电压型模拟量、电流型模拟量、开关量、485 总线型数字量、无线型信息数字量信号模拟量等。第二行安装的是市面常用工业采集器、PLC 设备和自主研发的测控终端，覆盖 RS485 总线、CAN 总线、LoRa 无线通信、NB 无线通信及各种 I/O 接口等。第三行安装的是执行器，控制类型覆盖继电器通断类、RS485 总线型、调速类等。

其中，电压型二氧化碳传感器、电流型可燃气体传感器、风扇、电磁阀等执行器，默认与多路采集器连接；直流减速电机车轮默认与 CAN 测控终端连接；NPN 型红外光电开关，步进滑台等默认与 PLC 设备连接；NPN 型金属接近开关，舵机等默认与智能通信节点连接；流量传感器、水泵执行器，默认与远程测控终端连接；RS485 烟雾传感器、多路采集器、RS485 电表、PLC 设备通过手拉手连接到 RS485 总线上。

下面章节将针对工业常用传感器、采集器、PLC 设备、嵌入式测控终端、执行器的配置、使用、开发等进行详细的说明。

课程安排：

第一章，工业数据采集系统概述。主要介绍工业互联网领域工业数据采集系

统的关键技术的发展现状、发展趋势、组成结构以及本文选用的实训设备简介等。

第二章，工业信号数据采集。通过对二氧化碳、可燃气体、接近开关、光电开关、风扇、电磁阀等工业元器件的模拟量、开关量等信号采集原理，继电器驱动控制原理的阐述，结合工控领域常用的多路采集控制模块，从工业元器件性能参数、接线规范、测试处理几个方面介绍工业信号的数据采集方法。

而且，围绕嵌入式微处理器的接口外设，从实验原理、实验内容、实验步骤、实验结果等几个方面详细介绍电位器、接近开关、步进电机等工业元器件的模拟量、开关量的采集编程方法、I/O 驱动控制编程方法。

第三章，数据通信与网络技术。主要讲解 RS485 总线、CAN 总线等有线传输特性和协议分析；接着讲解 ZigBee、WiFi、蓝牙、LoRa 等无线传输特性和协议分析；并通过测试工具或上位机应用软件对各种感知层协议进行测试。

第四章，网关应用与平台运维。介绍工业互联网云服务平台的功能、协议、接口；接着阐述工业智能网关的 MQTT 协议分析、Restful 接口调用、CoAP 协议分析，感知层传输协议的解析与数据格式的转换，以及网关接入平台参数配置方法；然后通过云平台的项目创建、设备添加、场景搭建，并通过网关配置参数，使感知层设备数据接入平台，达到可视化展示及远程采集控制的目的。

2. 工业信号数据采集

2.1. 工业测控系统的输入输出信号

工业测控系统中实现计算机控制的前提是，必须将工业生产过程的工艺参数、工况逻辑和设备运行状况等物理量经过传感器或变送器转变为计算机可以识别的电信号或逻辑量。

传感器和变送器输出的信号有多种规格，其中毫伏（mv）信号、0-5V 电压信号、0-10mA 电流信号、4-20mA 电流信号、电阻信号等都是计算机测控系统常用到的信号规格。在实际工程中，通常将这些信号分为模拟量信号和数字量信号两大类。针对一套计算机控制系统，必须了解输入输出信号的规格、接线方式、精度等级、量程范围、线性关系、工程量换算等诸多要素。

1. 模拟量信号

许多来自现场的检测信号都是模拟量，如液位、压力、温度、电压、电流等，通常都是将现场待检测的物理量通过传感器转换为电压或电流信号。许多执行装置所需的控制信号也是模拟量，如调节阀、电动机、电力电子的功率器件等的控制信号。

模拟量是指在时间上连续变化的量，这些信号在规定的一段连续的时间内，其幅值是连续的。

模拟量信号有两种类型，一种是各种传感器获得的低电平信号；如 k 分度热电偶在 1000℃时输出信号是 41.296mV。这些信号需要经过变送器转换成标准信号（4-20mA）再送给过程通道。再比如，热电阻输出的是电阻值，一般也要经

过变送器转换为标准信号，再送到过程通道。另一种是由仪器、变送器输出的1-5V 电压信号或 4-20mA 的电流信号。这些模拟信号不能直接进入计算机，必须经过 A/D 转换才能输入计算机，且要进行数据正确性判断、标度变换、线性化处理等。

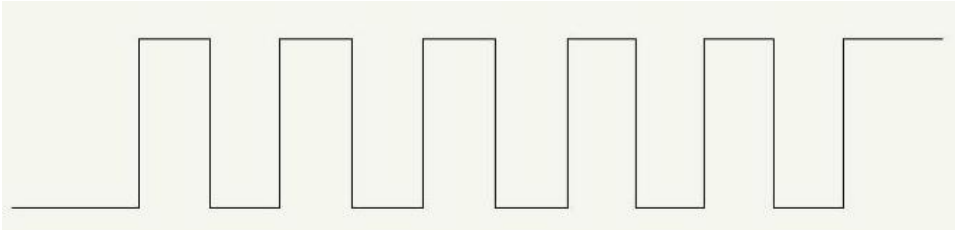
模拟信号非常便于传输，但它对干扰信号很敏感，容易使传送中信号的幅值或相位发送畸变。因此有时还需要对模拟信号做零漂修正、数字滤波等处理。

模拟量输出信号可以直接控制过程设备，而过程又可以对模拟量信号进行反馈。闭合 PID 控制系统就是这种形式。模拟量输出还可用来产生波形，D/A 变换器就成一个函数发生器。

2.数字量信号

有许多的现场设备往往只对应于两种状态，例如按钮、行程开关的闭合和断开，马达的起动和停止、指示灯的亮和灭、仪器仪表的 BCD 码、继电器或接触器的释放和吸合、晶闸管的通和断、阀门的开和关等，都可以用数字（开关）输出信号去控制，或对数字（开关）输入信号进行检测。

数字信号是指在有限的离散瞬时上取值间断的信号。在二进制系统中，数字（开关）信号是由有限字长的数字组成，每位数字是 0 或 1。数字（开关）信号的特点是，它代表某个瞬时的量值，是不连续的信号。数字（开关）信号的处理主要是监测开关器件的状态变化。



开关量信号，反映了生产过程、设备运行的现行状态、逻辑关系和动作顺序。例如，行程开关可以指示某个部件是否到达规定的位置，如果已经到位，则行程开关接通，并向工控系统输入“1”，否则行程开关断开，向工控系统输入“0”，如图所示。又如工控系统欲输出报警信号，则可以输出 1 个开关量信号，通过继电器、或接触器驱动报警设备发出声光报警。

数字量输入信号有触点输入和电平输入两种方式。触点输入又分常开和常闭，逻辑关系相反。工控系统实际上是按电平进行逻辑运算和处理的，因此工控系统必须为输入触点提供电源，将触点输入转换为电平输入。

数字量输出信号也有触点输出和电平输出两种方式。输出触点也有常开和常闭之分。

3.常用的数据采集方法

计算机测控系统中，数据采集是核心。在一些小规模的数据采集及监控系统中，上位机功能较简单，主要功能实现是下位机的数据采集。实际工控领域，可以使用基于嵌入式数据采集模块、PLC 数据采集模块、采集板卡、虚拟仪器等进行数据采集。

下面基于市面常用的多路采集控制模块介绍其数据采集方法，接着从编程角度阐述嵌入式远程测控终端信号采集控制系统的设计实现。

2.2. 多路采集控制模块的安装与配置

2.2.1. 模块概述

本系统使用多路采集模块对模拟量、数字量（开关量）信号进行采集，对继电器设备进行控制。

多路采集控制模块是将模拟量/数字量输入、继电器输出等信号转换到 RS485 总线上的一种模块，可以外接电压模拟量输出、电流模拟量输出、开关量输出的传感器或通断类控制的继电器设备，符合标准 modbusRTU 协议。

为了实训方便，将外围接口引入磁吸底板的插孔处，可根据插孔上定义连线，外观如图所示。

1.采集模块技术参数如表所示。

项目	参数
模拟量信号输入	1.输入通道：8 路通道隔离采集 2.信号类型：0~20mA、4~20mA、0~5V 和 0~10V 四种模拟量信号 3.采样速率：200HZ，8 个通道，每 25S 采集一遍 4.分辨率：12 位 5.采集精度：电压输入 0.2%　电流输入 0.3% 6.信号采集电路和电源输入隔离电压保护：1500V
开关量信号输入	1.输入通道：6 路干接点或湿节点开关量输入 2.信号类型：共正或共负输入，极性自动识别 3.信号电平：高电平(10V～30V)低电平(0V～1V) 4.采样速率：1000HZ 隔离电压保护：1500V
继电器信号输出	1.输出通道：4 路常开继电器输出 2.负载容量：阻性负载250V/3A　　感性负载250V/1A 3. 隔离电压保护：1500V
RS485 通讯输出	1.通讯协议：MODBUS-RTU 2.接口类型：隔离 RS485 通讯，输出接口采用过压过流双保护 3.波特率：1200bps、2400bps、4800bps、9600bps、19200bps 4.校验位：无校验、偶校验、奇校验 5.设置方式：模块地址、波特率、校验位均可通过软件设置 6.RS485 电路和电源输入隔离电压保护：1500V
工作电源	1.供电电压：10V~30V 宽范围供电，带电源极性保护 2.电源功耗：小于 3W

2.模块指示灯和开关功能说明

1) POW/SET：模块工作状态指示

A.绿灯亮：模块工作在运行状态。

B.红灯亮：模块有配置参数已写入，需重新上电。

2) TXD/RXD：通讯状态指示

A.绿灯闪亮：通讯接收到数据

B.红灯闪亮：模块正在发送数据

C.绿灯常亮：DATA+和 DATA-上接的通讯 RS485 线接反了或者接线有断线。

3.端子定义（重点）

端子	名称	说明	端子	名称	说明
1	AI0	模拟量输入通道 0	14	DIVCC	开关量输入供电正极
2	AI1	模拟量输入通道 1	15	DIGND	开关量输入供电负极
3	AI2	模拟量输入通道 2	16	DI0	开关量输入 0
4	AI3	模拟量输入通道 3	17	DI1	开关量输入 1
5	AIGND	模拟量输入公共端	18	DI2	开关量输入 2
6	AI4	模拟量输入通道 4	19	DI3	开关量输入 3
7	AI5	模拟量输入通道 5	20	DI4	开关量输入 4
8	AI6	模拟量输入通道 6	21	DI5	开关量输入 5
9	AI7	模拟量输入通道 7	22	RO0	继电器输出 0
10	DATA+	RS485 A 通讯+	23	RO1	继电器输出 1
11	DATA-	RS485 B 通讯-	24	RO2	继电器输出 2
12	+Vs	电源输入+	25	RO3	继电器输出 3
13	GND	电源输入-	26	ROCOM	继电器输出公共端

从表可知，模拟量传感器可以接到 AI0-AI7 通道上，开关量传感器可以接到 DI0~DI5 通道上，通断类执行器可以接到 RO0~RO3 通道上。通过对采集模块配置，将采样信号转换到 RS485 总线上变换为数字信号进行远传。

2.2.2. 参数配置

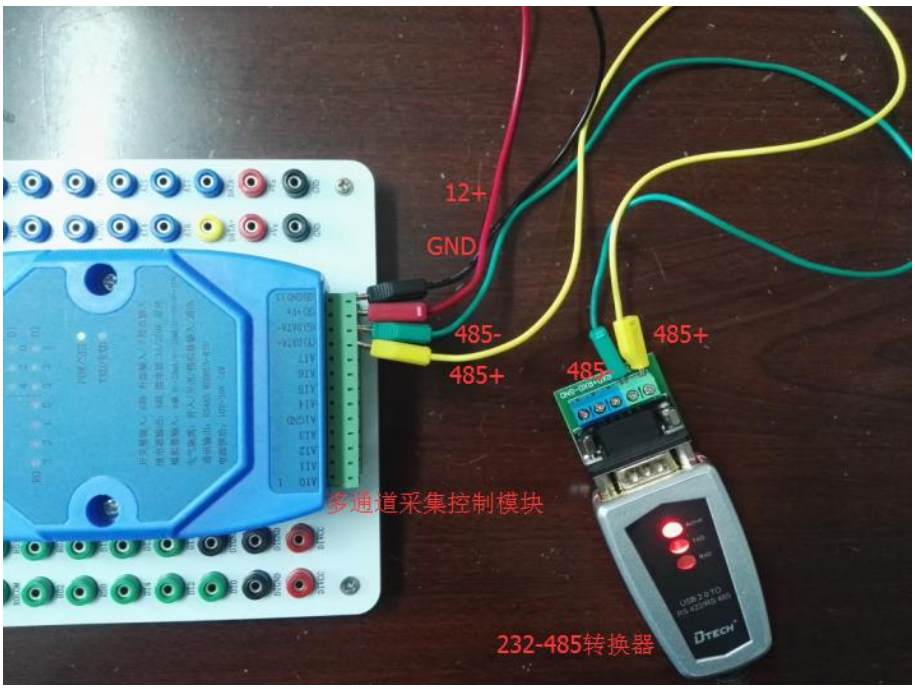
1.模块与 PC 硬件连接

多路采集模块的八个模拟量通道，支持 0-5V/0-10V 电压模拟量、4-20mA/0-20mA 电流模拟量等不同类型，可以根据外围传感器的信号接口，对模拟量通道类型进行配置。多路采集器符合 ModbusRTU 协议，地址码和波特率都可以根据实际使用情况进行修改。使用上位机应用软件进行参数配置。

配置之前，需要将采集器通过 USB-RS485 转换器连接到电脑。USB-RS485

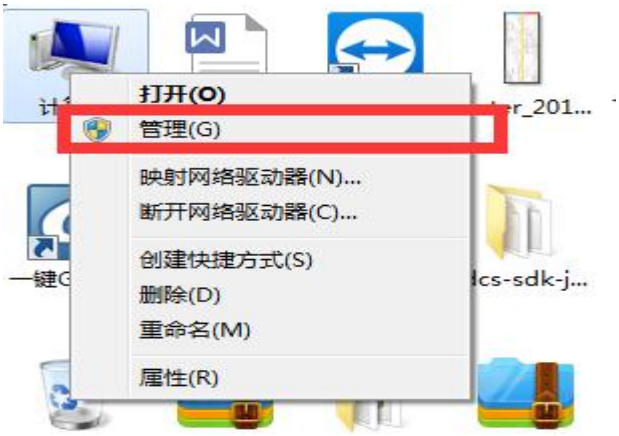
转换器外观如图所示。

首先，将 USB-RS485 转换器的 USB 一端连接到电脑的 USB 接口上，另一端的 RS485 总线连接到多通道模块上的 Data+，Data-接口处（注意：如果在系统布完线后设置软件时，可拔掉插在多通道模块磁吸底板 DATA+，DATA-香蕉头处的总线，使用转换器的 RS485 总线插入连接至电脑），如图所示。

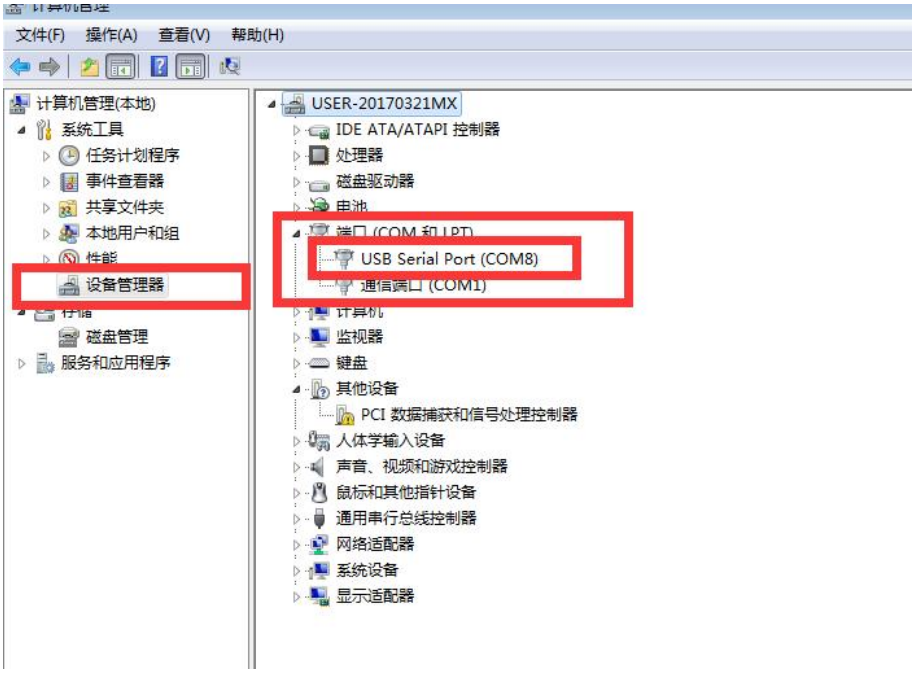


2.PC 端查看虚拟端号

若用户不知道 USB-RS485 转换器连接电脑的串口号为多少时，可以通过右击桌面“我的电脑”，选择“管理”选项，点击进入。

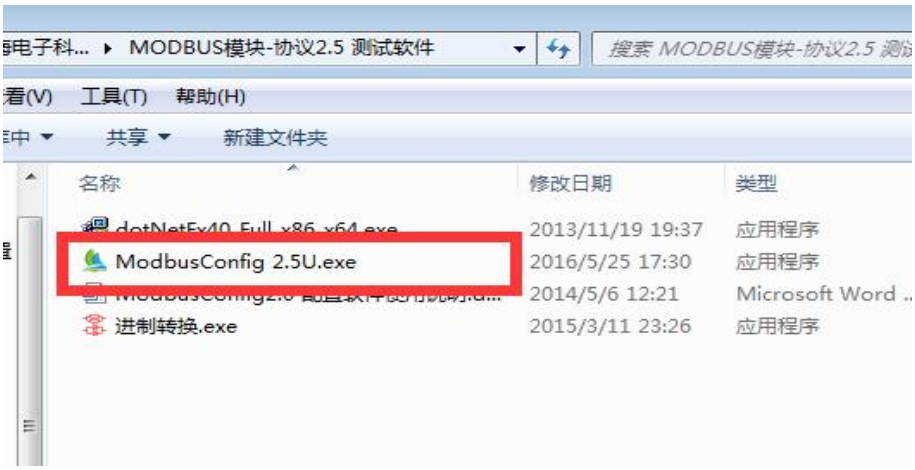


进入之后，选择“系统工具”下的“设备管理器”，选择“端口（COM 和 LPT）”，既可以看到一个 USB Serial Port 信息，注意后面的 COM，图中 COM8 即表示当前串口为 8.

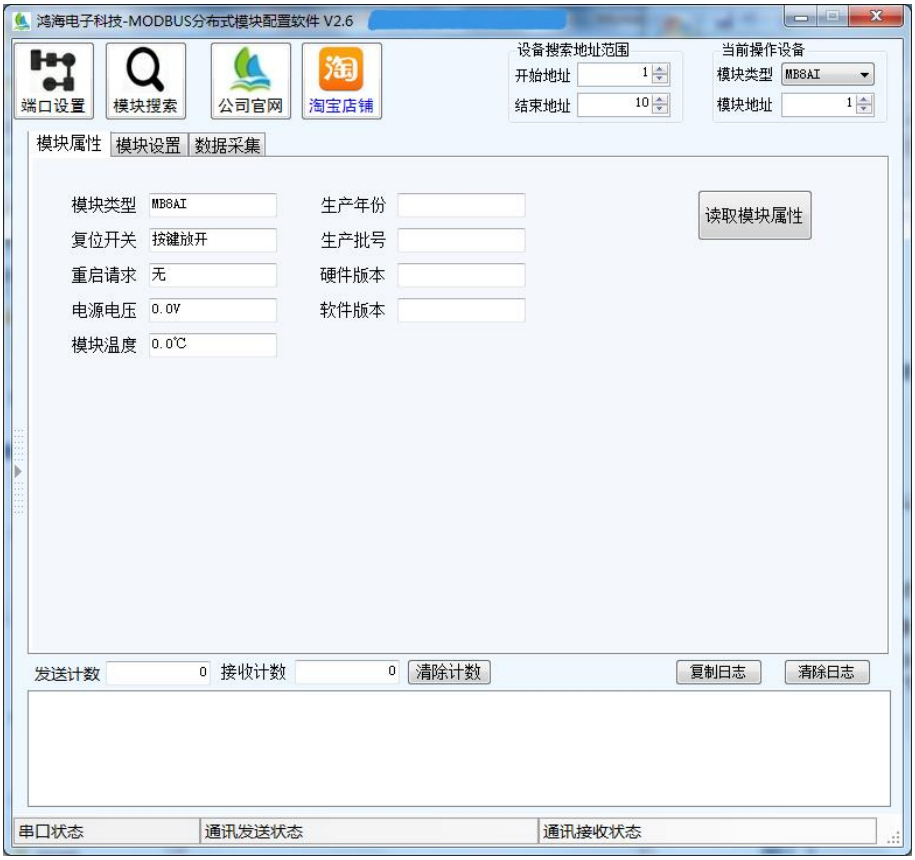


3.配置工具----地址搜索

配置工具位于“资料\软件工具\”目录，找到软件 ModbusConfig 2.6U.exe，打开即可。该软件为多通道模块专用 modbus 配置工具，可以通过这个软件配置模拟量通道类型。



打开软件，如图所示。

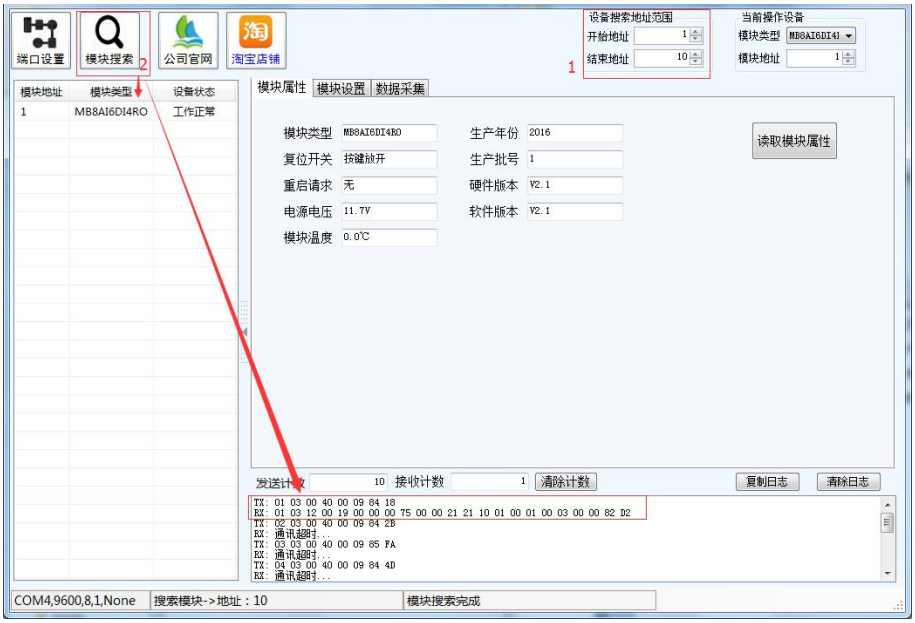


点击左上角“端口设置”按钮

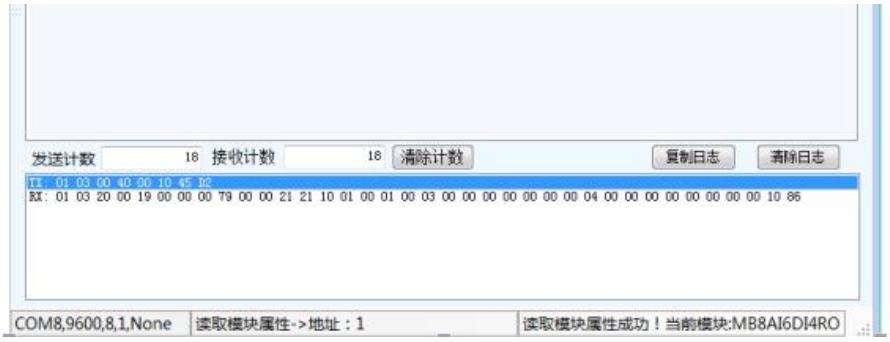
根据虚拟出的串口号，在该界面下选择对应的串口号，如 COM4，波特率默认为 9600，然后点击“打开串口”，提示“打开串口成功”。

串口打开成功后，返回 ModbusConfig 主页面。

在右上角的“设备搜索地址范围”，由用户设置可能的地址范围，如 1-10，然后点击左侧的“模块搜索”按钮，该软件就从地址 1，搜索到地址 10，搜索有效的多路采集模块。搜索到后，会在左侧的提示框出来，显示内容包括模块地址、模块类型型号、设备状态。同时下方也会出现具体的通信数据。如图所示。



通过搜索可知，当使用地址码“0x01”探测时，有实际的返回数据，说明当前模块地址为 1，其他地址探测时，返回“通讯超时”。工具操作区域的“模块属性”选项页面显示了当前模块的类型、软硬件版本、生产年限等信息。在页面底部还会显示出当前 Modbus 模块上反馈的连接设备的数据信息。

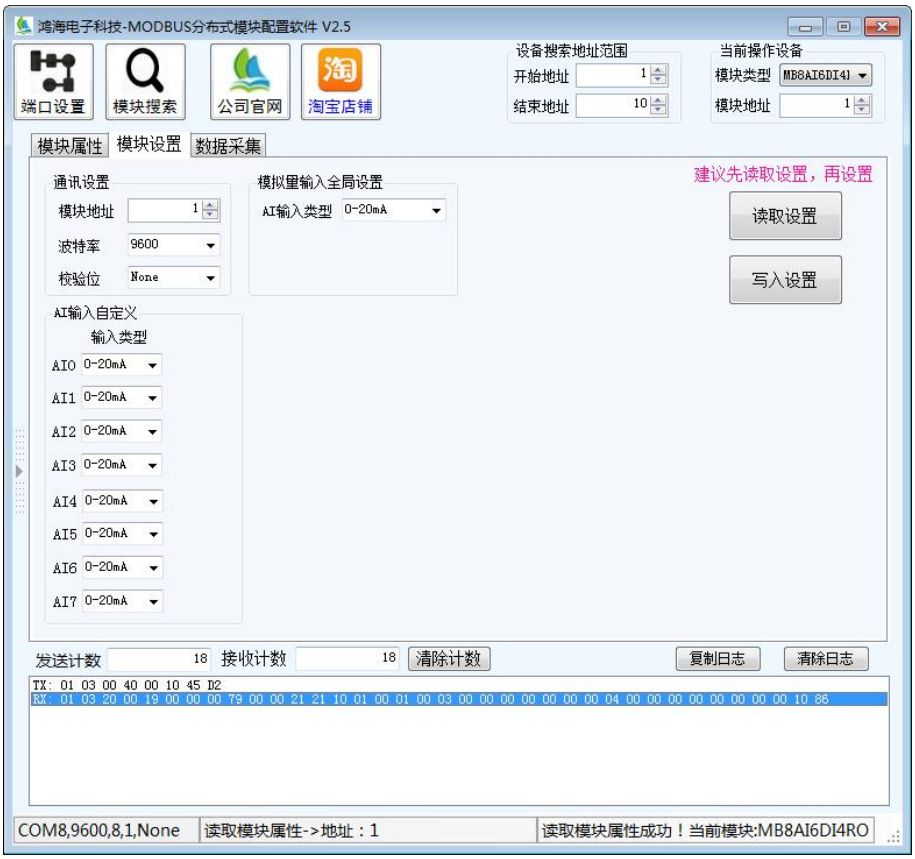


如果在页面底部，程序一直显示“读取数据超时”的字样，则可能的情况有：

1) Modbus 采集控制模块的 485 转 232 接线错误
2) 端口串口选择错误
3) Modbus 设备地址有误，可以在页面上扩大地址搜索的范围，从地址 1 开始，将结束地址修改为 15、或者更大的整数。

4.配置工具----通道设置

当软件能够读取到模块属性后，选择“模块设置”选项卡，可对模拟量通道的信号输入类型进行自定义。默认类型，如图所示。



采集器模块外接传感器的信号类型见下表。

序号	类别	通道号	位置	配置接口类型
1	模拟量输入	AI0	二氧化碳传感器	0-5V
2		AI1	风速传感器	4-20mA
3		AI2	大气压力传感器	4-20mA
4		AI3	风向传感器	4-20mA
5		AI4	土壤水分传感器	4-20mA
6		AI5	液位传感器	4-20mA
7		AI6	可燃气体传感器	4-20mA

8		AI7	/	任意，如 0-10V

因此，采集模块的 AI0（0-5V 二氧化碳）——》设置为 0-5V 电压型，AI1（4-20mA 风速）——》设置为 4-20mA 电流型，AI2（4-20mA 气压）——》设置为 4-20mA 电流型等，AI3(4-20mA 风向）——》设置为 4-20mA 等。按照上表传感器输出信号类型，设置多通道采集器模拟量通道的输入类型。

首先，确认模块地址与实际相符，此处地址码为 1.

然后，将 AI 输入类型通过右侧下拉框选择为“通道自定义”。

之后，将 AI0 通道，通过右侧下拉框选择为 0-5V；AI1 通道，通过右侧下拉框选择为 4-20mA；AI2 通道，通过右侧下拉框选择为 4-20mA。不使用的通道，可设置其他类型的接口，如 0-10V，0-20mA 等，用于外扩其他传感器。

最后，点击“写入设置”，就可以将用户自定义的通道类型写入模块硬件，固化。可通过“读取设置”，查看设置是否成功。

注意：4-20mA 模式，应在下拉框中选择“模式 1”，不同的模式，软件数据处理所使用计算公式不同，因此需要用户特别注意。

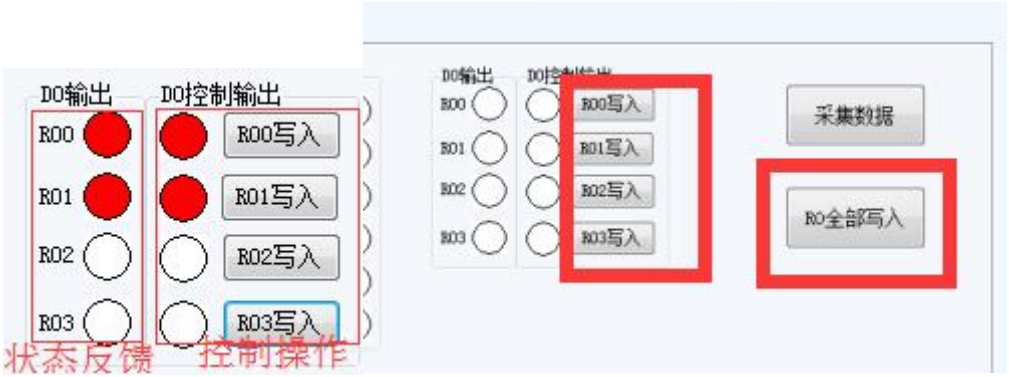
5.采集测试

点击“数据采集”选项卡，切换到数据采集界面，就可以对当前设置的通道进行轮询采集。采集结果如图所示。

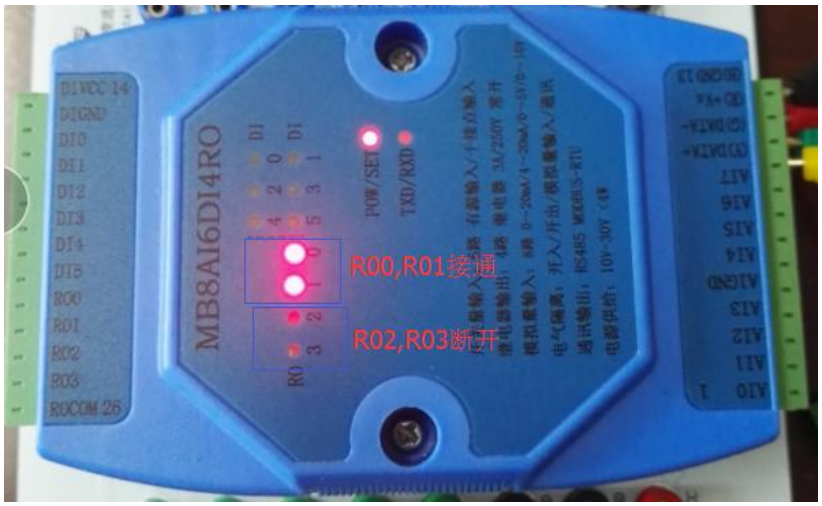
从图上可知，由于模拟量通道如 AI0-AI7 还没有接传感器，所以读出的数据分别是 0V，4mA，0mA。

6.控制测试

可以通过该配置工具对通道 RO0-RO3 进行控制测试。点击“RO0-RO3”前面的圆圈，选中圆圈，变成红色，表示要接通继电器。点击四个输出通道对应的“RO0-RO3 写入”按钮，就可以实际控制采集器 RO0-RO3 对应的继电器接通，采集器通道红色指示灯点亮，表示继电器接通，控制成功；未点亮，则代表断开。



控制测试，结果如图所示。



多通道采集控制模块配置完成后，就可以正常使用了。

默认出厂时，多通道采集控制模块已经配置好，RS485 地址配置为 0x01。

用户在没有深入了解的情况下，请不要重新设置，以免与总线上其他设备冲突。下面介绍各种信号输出的传感器如何连接到该采集器的对应通道上。

本系统中，多通道采集控制模块连接的传感器与执行器，如下表所示。

序号	产品名称	接口类型	本系统使用	个数
1	多通道采集控制模块	RS485	√	1
2	二氧化碳传感器	电压型模拟量	√	1
3	风速传感器	电流型模拟量		1
4	大气压力传感器	电流型模拟量		1
5	风向传感器	电流型模拟量		1
6	土壤水分传感器	电流型模拟量		1
7	液位传感器	电流型模拟量	√	1
8	可燃气体传感器	电流型模拟量	√	1
9	红外对射传感器	开关量数字量		1
11	红外光电开关	开关量	√	1
12	金属接近开关	开关量	√	1
13	风扇	继电器输出控制	√	1
14	声光报警器	继电器输出控制		1
15	电磁阀	继电器输出控制	√	1
16	水泵	继电器输出控制		1

多通道采集模块包含 8 个 AI 通道、6 个 DI 通道、4 个 RO 通道。每个通道所连接的设备、接口类型，如下表所示。

序号	类别	通道号	位置	配置接口类型	本系统使用
1	模拟量输入	AI0	二氧化碳传感器	0-5V	√
2		AI1	风速传感器	4-20mA	
3		AI2	大气压力传感器	4-20mA	
4		AI3	风向传感器	4-20mA	

5		AI4	土壤水分传感器	4-20mA	
6		AI5	液位传感器	4-20mA	√
7		AI6	可燃气体传感器	4-20mA	√
8		AI7	/	/	
9	开关量输入	DI0	红外对射传感器	无源常闭型	
10		DI1	红外光电开关	NPN 型	√
11		DI2	金属接近开关	NPN 型	√
12		DI3	/	/	
13		DI4	/	/	
14		DI5	/	/	
15	继电器输出	RO0	风扇	常开型输出	√
16		RO1	声光报警器	常开型输出	
17		RO2	电磁阀	常开型输出	√
18		RO3	水泵	常开型输出	
注：打“ √ ”的是本系统中可以接多路采集器的终端					

2.3. 基于多路采集模块的电压型模拟量采集

2.3.1. 电压型模拟量采集原理

模拟量信号是一种连续变化的物理量，如电压、电流、温度、压力、位移、速度等。一般而言，大多数物理量都是以模拟量的形式表示的。模拟量一般都有一个量程大小，也就是模拟量数值的上下限，比如液态水的温度量程是 0-100℃，电动阀门的开度是 0-100%等。通常采用电压电流信号表示模拟量的输入输出信号，有如下几个标准：0-5V、1-5V、0-10V、4-20mA、0-20mA 等。模拟量输入/输出英文简称为 AI/AO。

在工控领域，一般先用现场信号变送器把物理量变换成统一的标准信号（如 4-20mA 的直流电流信号、0-5V 的直流电压信号），然后再送入模拟量输入模块将模拟量信号转换成数字量信号，以便单片机或 PLC 处理器进行处理。

以电压模拟量为例，模拟量信号的采集通常由模数（A/D）转换电路完成模拟量到数字量的转换，通过 CPU 的数据处理读取现场物理量的值。工业常用的模拟量输入模块一般由滤波、模数转换电路、光电耦合器等部分组成。滤波电路主要滤除模拟输入信号中断噪声干扰信号，提高系统信噪比；模式转换电路种类

较多，常用有积分型、V/F 变换型、逐次逼近型及Σ-△型，目前常用的是Σ-△型。光电耦合器有效防止电磁干扰。

对多通道的模拟量输入模块，通常可以设置多路转换开关进行通道切换，或者多通道同时使用。

一、电压模拟量的采集和计算

电压模拟量信号的采集，通常可使用单片机的 ADC 接口将外部模拟量信号转化为数字信号。目前市场的很多单片机自带 ADC 接口，若无 ADC 转换接口，可以使用独立的 ADC 数模转换芯片外扩。

为选取性价比更高的模式转换芯片，将意法半导体公司和德州仪器公司分辨率 14 位以上的Σ-△型模数转换芯片，列出如下表，供参考使用。

芯片型号	分辨率	采集频率	价格
AD7792	16 位	>4Hz	25 元
AD7761	16 位	>100KHz	200 元
AD7781	20 位	>5Hz	100 元
AD7729	15 位	>10Hz	80 元
AMC1203	16 位	>100 Hz	60 元
ADS113	16 位	>5Hz	50 元
ADS1626	18 位	>1MHz	300 元

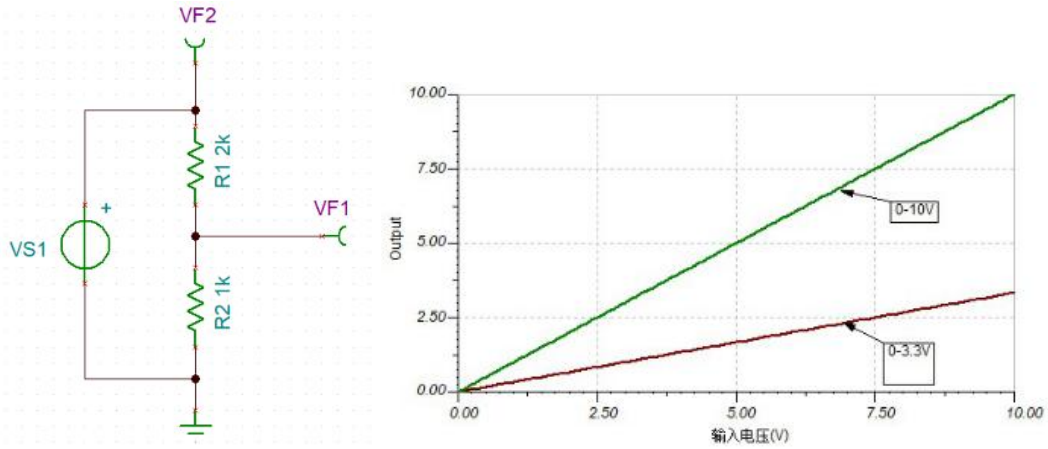
ADC 模块是将模拟信号转换为数字信号，数字信号用 0 和 1 表示，ADC 模块有参考电压。假设给的参考电压是 5V，ADC 是 12 位的，那么 12 位 ADC 可存储的数字量范围为（二进制）000000000000~111111111111，转换为十进制数字范围为 0~2^12 即 0~4095。也就是把参考电压分成 4096 份，最小分辨率为 VREF/4096。也就是二进制 000000000000 代表输入模拟量电压为 0V，而 111111111111 代表最大值 VREF。输入的电压值 V=VREF*（ADC 采集到的数字量÷4096）。

二、单片机采集高于参考电压的电压模拟量信号

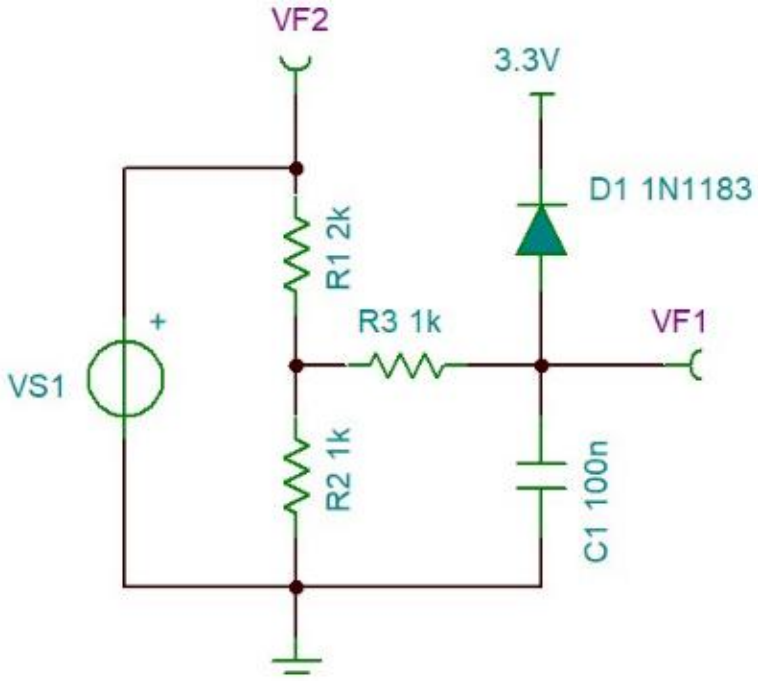
在工业物联网中，经常用到 0-5V，0-10V 电压输出的传感器，而单片机 ADC 参考电压、引脚输入电压比较低，如 STM32 参考电压为 3.3V，引脚的输入电压为 3.3V。当引脚输入电压高于 3.3V 会导致单片机损坏。那么如何用单片机采集高于参考电压的电压模拟量信号呢？以 0-10V 为例进行说明。

采用电阻分压法将 0-10V 等比例降到 0-3.3V，再接入单片机 ADC 接口。使

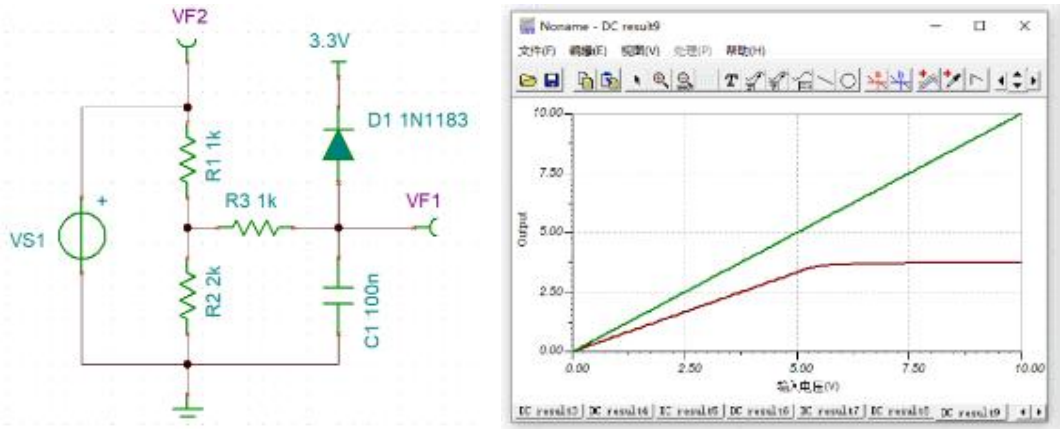
用两个电阻，阻值可取为 2:1，电阻分别是 2K 和 1K，分压后得到信号源的 1/3 的电压，这样就将 0-10V 之间变化的信号变成 0-3.33V 之间变化的信号，如图所示。



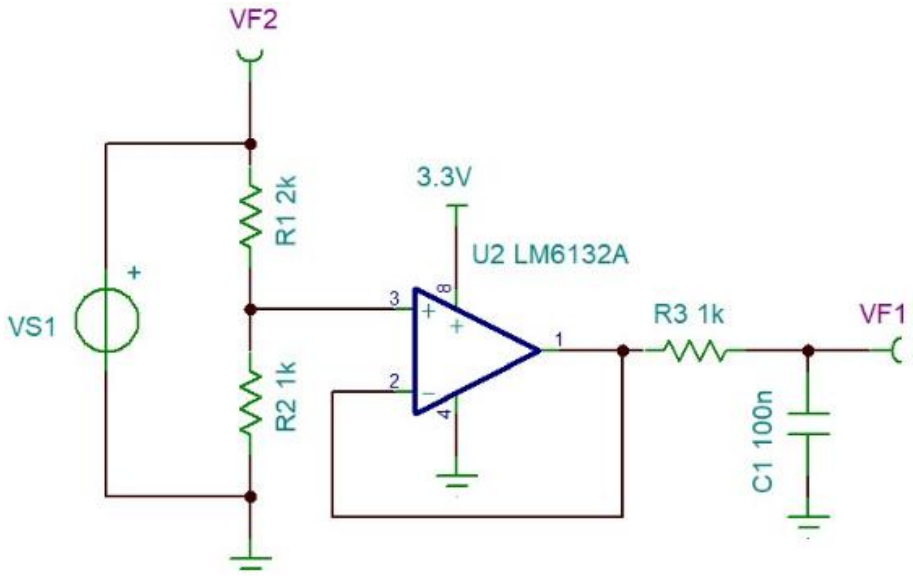
对上图进行完善，使用 R3 和 C1 构成低通滤波电路，用于滤除传输过程中的高频干扰信号，在 PCB 布局时电阻电容应靠近单片机 ADC 管脚。二极管 D1 为钳位二极管，用于保证在电路故障时（比如 R2、R1 电阻焊错位置等），或出现尖峰浪涌电压时，VF1 可以保持在一个安全电压，不至于损坏单片机。电路中的 D1 应选择导通压降低的肖特基二极管。



下图展示了故意将 R1 和 R2 焊错位置时，二极管 D1 开始作用，将 VF1 钳位在一个安全的电压，保护了单片机。



上述电路，工作时 R3 会有电流流过，影响到采样精度。对以上电路继续优化，使用一个输入输出 Rail-to-Rail 的运放构成一个电压跟随器，如图所示。



电路中使用理想情况下运放输入阻抗无穷大的特点。在信号采集中对 R1，R2 分压电路影响小，使电阻分压结果更加精准。由于该运放具有保护单片机的能力，因此去掉了钳位二极管。

根据实际情况，优化电压信号的电路设计。也可以直接选购模拟量采集模块进行采集。

2.3.2. 二氧化碳传感器数据采集

主要测量环境中二氧化碳的浓度。

1.传感器参数说明

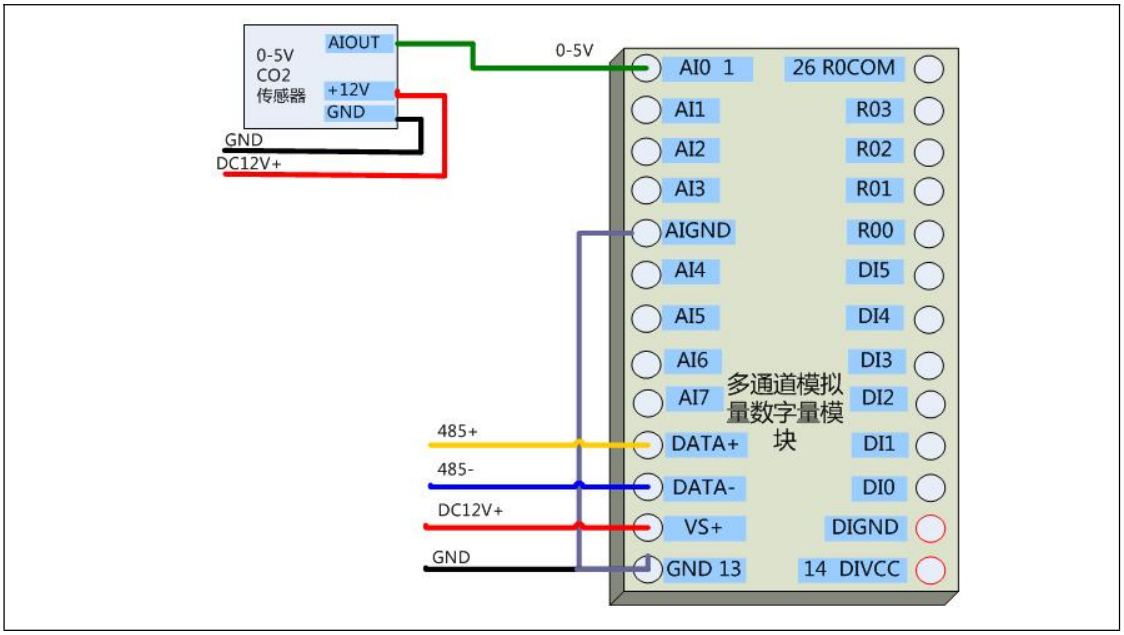
CO2测量范围：0~5000ppm（可定制）

CO2精度：±(40ppm+ 3%F·S) (25℃)
稳定性：<2%F·S 非线性：<1%F·S
数据更新时间：2s
系统预热时间：2min(可用)、10min(最大精度)
平均电流：<85mA
温度影响：自带温度补偿
输出信号：0~5V
工作环境：-40~+60℃、0-80%RH(无凝结)
供电电源：10~30V DC(0~10V 型只能 24V 供电)，典型电源12VDC。

2.硬件连接

该二氧化碳传感器是四线制电压型传感器，可与多路采集模块连接进行数据采集。

名称	接线说明
	①电源正：红色香蕉头线，一端接入实训台两侧电源接口板的 12V 处，另一端接入传感器磁吸底板的+12V 处。 ②电源负：黑色香蕉头线，一端接入台体两侧的电源接口的 GND 处，另一端接入传感器磁吸底板的 GND 处。 ③信号：绿色香蕉头线，一端接入多通道采集模块的 AI0 接口处，另一端接入传感器磁吸底板的 V1/I1/485A+接口处。 ④注意：多通道采集模块的 AGND 要与 GND 连接，模拟地与数字地共地。
连接示意图	



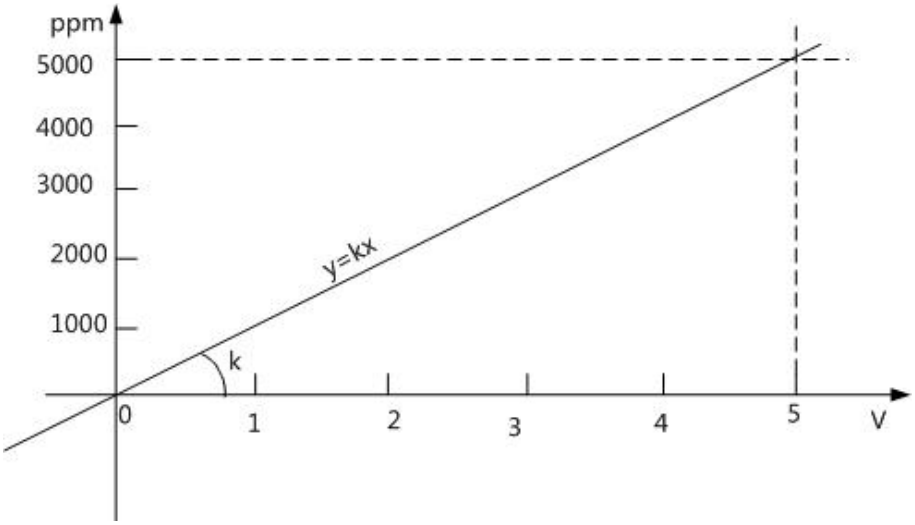
3.数据采集

上电后，二氧化碳磁吸底板电源指示灯会被点亮，多通道采集器的电源指示灯 POW/SET 点亮，说明电源接线正常。

打开多通道采集器的上位机配置软件 ModbusConfig，切换到“数据采集”选项页，可以看到第 0 通道，电压值发生了变化，采集到 0.56V，不再是 0V 了。说明传感器接线正确，数据采集正常，但数据还不是实际二氧化碳浓度值。可以对着二氧化碳传感器的气体导入膜吹气，二氧化碳的采集通道的数据会发生明显变化。

4.数据计算

实际应用时，该二氧化碳采集信号的电压值与实际量程近似成线性关系。根据数学知识，可做线性坐标系，如图所示。横坐标为电流值，范围为 0-5V，纵坐标为量程，范围是 0-5000ppm。



可以求得：k=1000。

因此计算公式：y=1000x。

当前，采集器测量的电压值为 0.56V，代入计算公式，可求得二氧化碳浓度为 560ppm。

2.4. 基于多路采集模块的电流型模拟量采集

2.4.1. 电流型模拟量采集原理

早期的传感器大多为电压输出型，即将测量信号转换为 0-5V 电压输出，通过模拟数字转换电路转换为数字信号供单片机读取、控制。但在信号需要远距离传输或使用环境电网干扰较大的场合时，电压输出型传感器的使用受到了限制，暴露了抗干扰能力差的缺点。而电流输出型传感器以其具有较高的抗干扰能力得到了广泛的应用。电流输出型传感器的输出范围常用的有 0-20mA 和 4-20mA 两种，传感器输出最小电流及最大电流时，分别代表传感器所标定的最小及最大额定输出值。

工业上普遍需要测量各类非电物理量，例如温度、压力、速度、角度等，这些都需要转换成模拟量电信号才能传输到几百米外的控制室或显示设备上。工业上最广泛采用的是用 4~20mA 电流来传输模拟量。

一、为什么使用电流传输信号

采用电流信号的原因是不容易受干扰，因为工业现场的噪声电压的幅度可能达到数 V，甚至几十 V，此时使用电压传输模拟量信号已完全失去意义。

虽然噪声电压幅度高，但功率微弱，所以噪声电流通常小于 nA 级别，因此给 4-20mA 电流传输带来的误差非常小；电流源内阻趋于无穷大，导线电阻串联在回路中不影响精度，因此在普通双绞线上可以传输数百米。

二、信号最大选择 20mA 的原因

30V 电压 30mA 电流所引起的火花是可以点燃危险气体的平均下限，为了保险起见，同时参照其它传统设定，所以将许多仪表定位 24V 供电，同时限定电流小于 30mA，为了留有余地，信号上限定为 20mA。

最大电流 20mA 的选择也是基于安全、使用、功耗、成本的考虑。安全火花仪表只能采用低电压、低电流，综合考虑生产现场仪表之间的连接距离，所带负载等因素，还有就是功耗及成本问题，对电子元器件的要求，供电功率的要求等因素，最后由国际电工委员会(IEC)确定 4-20mA 信号为过程控制系统用模拟信号标准。

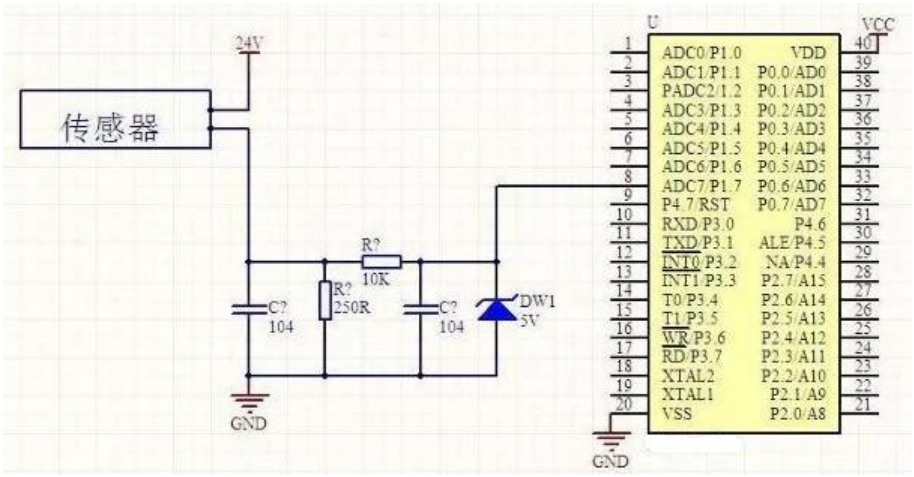
三、信号起点电流选择 4mA 的原因

下限没有取 0mA 的原因是为了能检测断线：正常工作时，不会低于 4mA，当传输线因故障断路，环路电流降为 0。常取 2mA 作为断线报警值。

4-20mA 电流作用在 250 欧姆电阻上，正好符合标准信号的电压标准 1-5V。因此我们的仪表、工业传感器也采用国际标准信号制，传输信号采用 4-20mA.DC，联络信号采用 1-5V.DC，即采用电流传输、电压接收的信号系统。

四、4-20mA 转电压信号的简单调理电路

使用一个 250 欧姆的电阻即可将 4-20mA 电流信号转换为 1-5V 电压信号，经过 RC 低通滤波电路滤除噪声，将有效模拟信号传入单片机进行模数转换。



2.4.2. 电流型传感器的接线方式

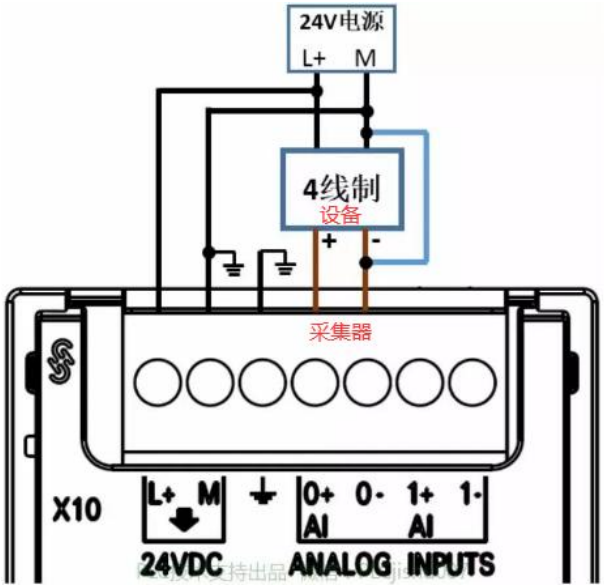
以设备线缆的条数为准，分为四线制、三线制、两线制三种类型，不同类型的信号其接线方式是不同的。

线制是学习模拟量的一个难点，大家记住主要看提供信号的设备的出线，有几根就是几线制。

1.四线制

四线制信号是提供信号的设备上，信号线和电源线加起来有 4 根线。提供信号的设备有单独的供电电源，除了两根电源线还有两根信号线。

四线制信号的接线方式，如图所示。

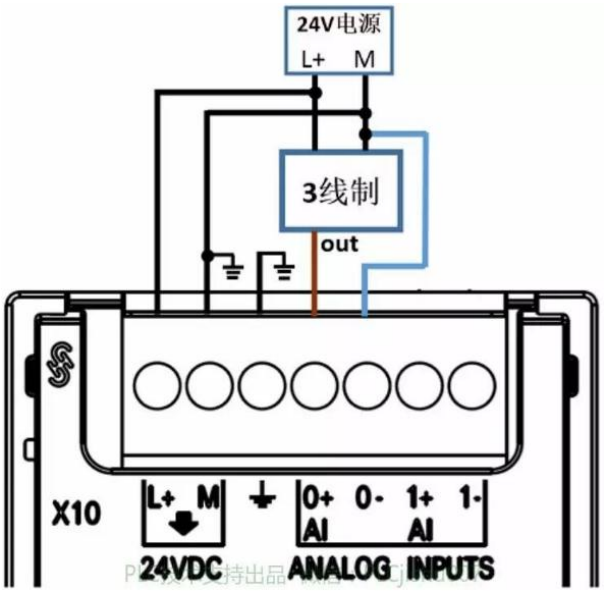


上图中，24V 电源的正负极分别接到传感器设备的电源正负极上，传感器的输出模拟量信号正（+）接到通道 AI0（0+），输出模拟量信号负（-）接到通道 AI0（0-）。模拟量输入通道的负极（0-）与采集模块的负极（M）相连，也可以不相连，但采集模块的负极一定要和电源的负极相连。

2.三线制

三线制信号是提供信号的设备上，信号线和电源线加起来有三根线，信号负和电源负线（M 线）为公共线。

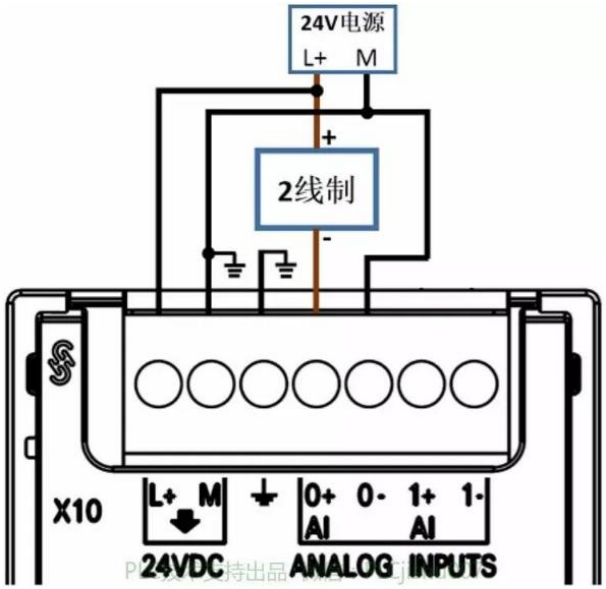
四线制信号的接线方式，如图所示。



常用的信号源设备很少有三线制的，大多是两线制与四线制。

3.两线制

两线制信号指提供信号的设备上，信号线和电源线加起来只有 2 根线。如果采集器的模拟量通道没有供电功能，则仪表或设备需要外接 24V 直流电源。两线制的接线方式如图所示。



上图中，24V 电源的正极除了给采集模块供电（L+）外，还接到了两线制传感器的正极（+）给传感器供电；传感器的模拟量信号从负极（-）输出，连接到采集模块的模拟量输入通道 AI0 的正极（0+），模拟量输入通道 AI0 的负极（0-）与采集模块的负极（M）及 24V 电源的负极相连。

注意：把模拟量输入通道的负极（0-）与采集模块的负极（M）相连接，是为了得到相同的参考电势，这样模块检测到的电流信号才准确。

2.4.3. 可燃气体传感器数据采集

1.传感器概述

可燃气体传感器采用专业测试可燃气体浓度传感器探头作为核心检测器件，测量环境中的甲烷、液化气等易燃气体的浓度；具有测量范围宽、精度高、线性度好的特点。

测量范围：0~100%LEL

分辨率：0.1ppm

响应时间：<20s

数据更新时间：2s

输出信号：4-20mA

电流输出负载：≤600欧姆

运行环境：-30~+50℃、0-100%RH

供电电源：12~24V DC宽电压输入，典型电源12VDC。

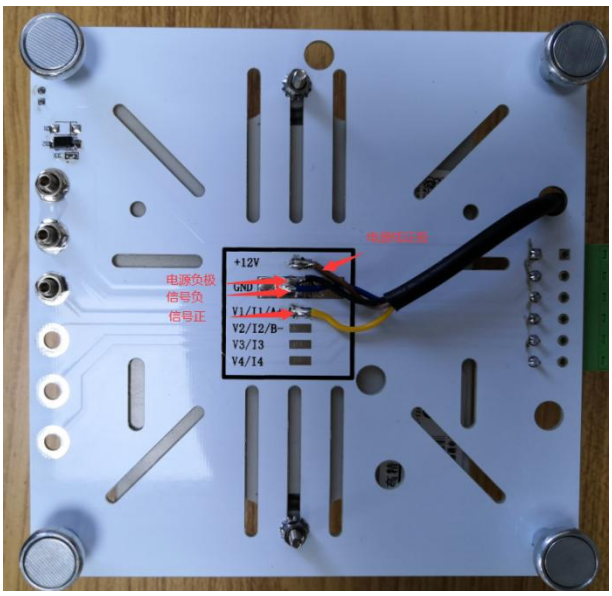
2.硬件连接

可燃气体传感器属于四线制电流型信号传感器，信号含义如下表所示。注意信号线的正负，不要将电流/电压信号线的正负接反。

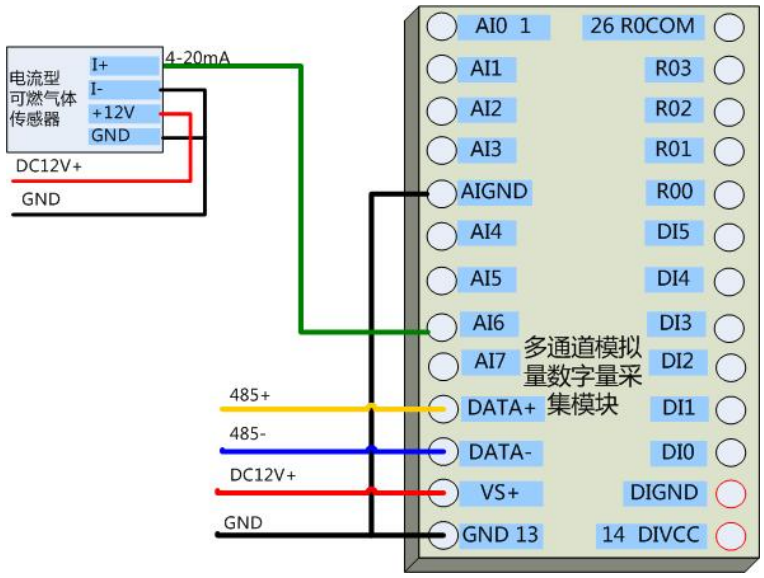
	线色	说明
电源	棕色	电源正(12-24VDC)
	黑色	电源负
通信	黄(灰)色	电压/电流输出正
	蓝色	电压/电流输出负

其中电源正，接 12V 直流电源的正极；电源负，接 12V 直流电压的负极；电流输出正，接采集器的模拟量输入通道；电流输出负，接采集器的模拟地。

为方便实训，将传感器固定在磁吸底板上，将电源线、信号线焊接在磁吸底板上，通过磁吸电路板将信号引到香蕉孔上，方便插拔和重复训练。另外，由于采集器的模拟地和数字地需要短接共地，因此，此处就将信号负和电源地短接在一起了，如图所示。



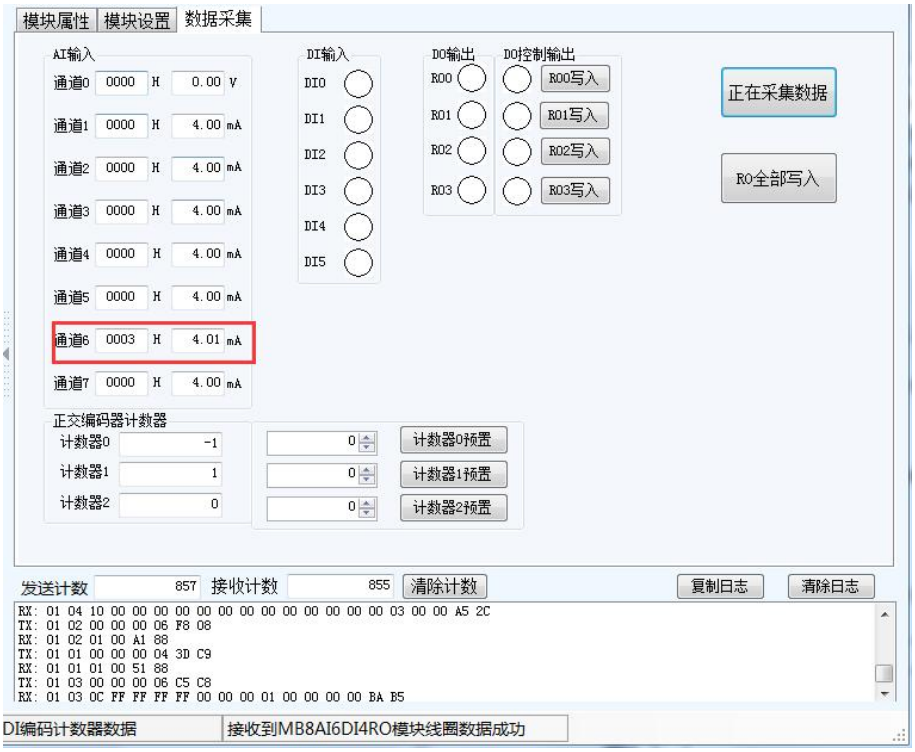
可燃气体传感器可以使用多路采集器进行采集，实时读取它的电流值。

名称	接线说明
	①电源正：红色香蕉头线，一端接入实训台两侧电源接口板的 12V 处，另一端接入传感器磁吸底板的+12V 处。 ②电源负：黑色香蕉头线，一端接入台体两侧的电源接口的 GND 处，另一端接入传感器磁吸底板的 GND 处。 ③信号：绿色香蕉头线，一端接入多通道采集模块的 AI6 接口处，另一端接入传感器磁吸底板的 V1/I1/485A+接口处。 ④注意：多通道采集模块的 AGND 要与 GND 连接，模拟地与数字地共地。
连接示意图	
	

3.信号采集测试

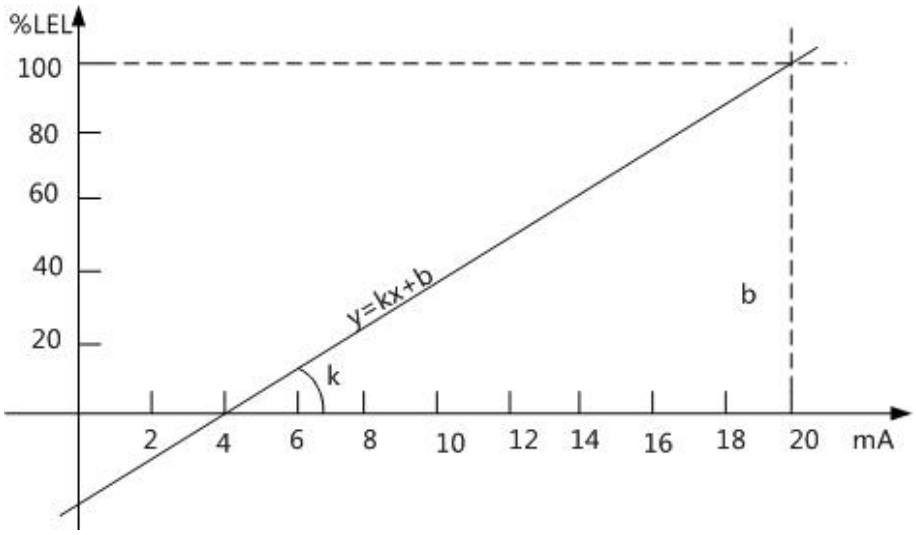
上电后，可燃气体传感器的磁吸底板电源指示灯会被点亮，多通道采集器的电源指示灯 POW/SET 点亮，说明电源接线正常。

打开多通道采集器的上位机配置软件 ModbusConfig，切换到“数据采集”选项页，观察第 6 通道，电流值发生了变化，空气中可燃气体浓度很低，采集到 4.01mA，不再是 4.00mA 了。采集器的电流模拟量通道如果未接任何传感器，则采集到的电流值都是 4.00mA。说明传感器接线正确，数据采集正常，但数据还不是实际可燃气体的浓度值。



4.数据计算

实际应用时，通常认为信号的电流值与实际量程成线性关系，根据数学知识，可做线性坐标系，如图所示。横坐标为电流值，范围为 4-20mA，纵坐标为量程，范围是 0-1 米。



可以求得：k=1/16，b=-1/4。

因此：y=(25/4)x-25。

当前，采集器测量的电流值 4.01mA，代入计算公式，可求得可燃气体浓度为 0.0625%LEL。说明空气中可燃气体的浓度很低。

2.4.4. 投入式液位变送器数据采集

液位变送器，主要用于测量液体的高度。投入式测量，适用于河水、水库、城市地下水、水处理池等液位监测场合。

1.传感器参数说明

量 程：0～1m

输出信号：4～20mA（二线制）

综合精度：±0.5%FS

非线性：≤0.5%FS

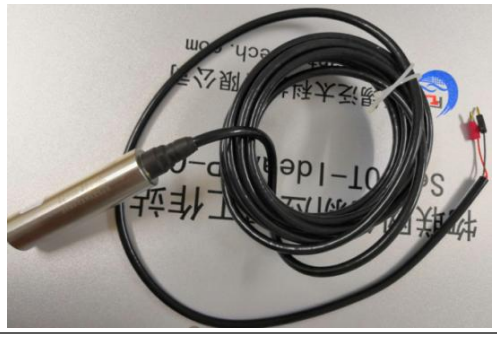
迟滞、重复性：≤0.2%FS

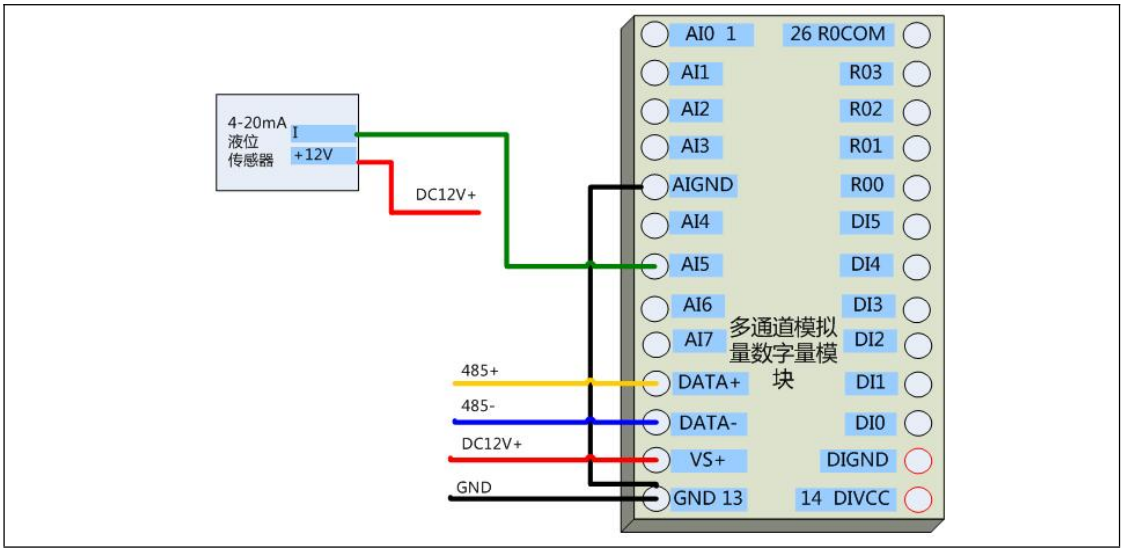
供 电： 12VDC、24VDC 均可。

注意：液位传感器上的量程写着“0-1m/5mH2O”，其中“0-1m”代表量程，可测量 1m 以内的液体高度，“5m”代表线长 5 米，可根据实际引线选择。

2.硬件连接

该液位传感器属于二线制电流型传感器。

名称	接口参数
液位传感器 	连接说明： ①红色线，电源正：红色香蕉头，直接接入台体电源+12V 接口处。 ②信号线：黑色香蕉头，直接接入多通道采集模块的 AI5。
连接示意图	



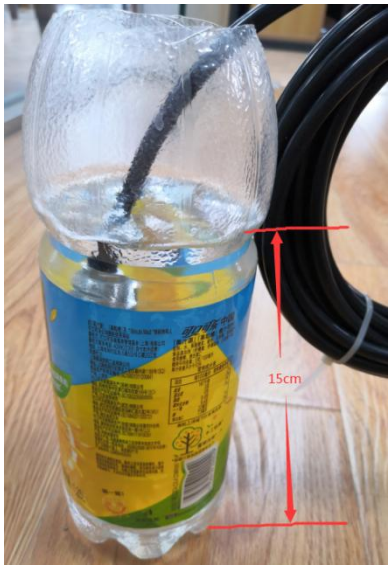
3.上电测试

上电后，多通道采集器的电源指示灯 POW/SET 点亮，说明供电正常。

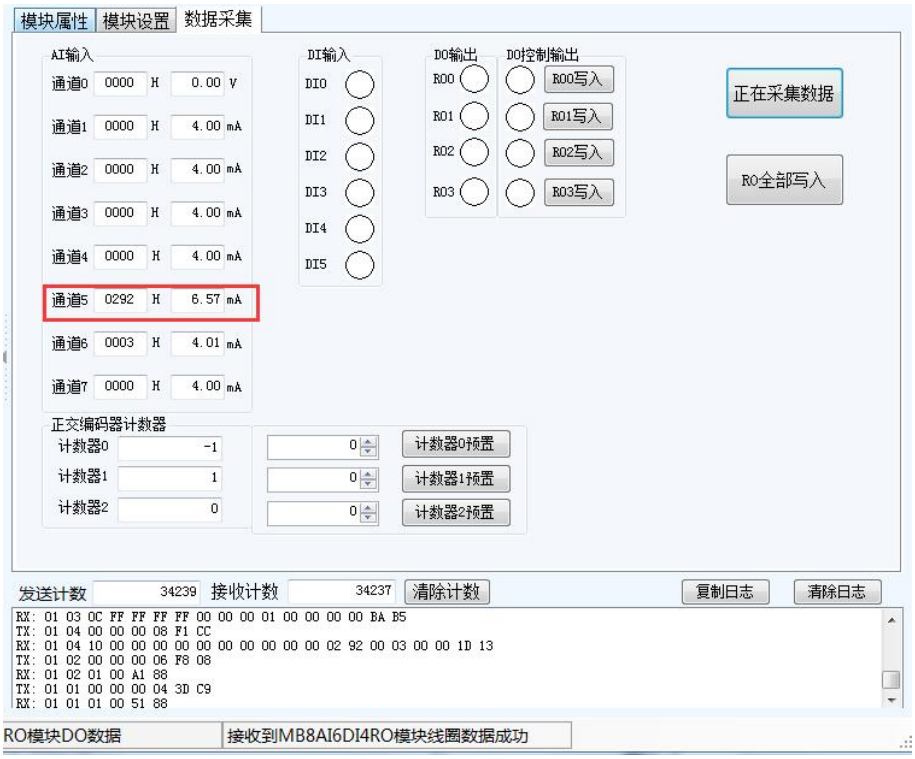
打开多通道采集器的上位机配置软件 ModbusConfig，切换到“数据采集”选项页，观察第 5 通道。

如果采集器的第 5 路电流模拟量通道未接液位传感器时，则采集到的电流值默认是量程的下限 4.00mA。

如果第 5 路模拟量通道已经连接了液位传感器，但液位传感器未投入液体中，则采集到的电流值约 4.00mA 或 4.01mA。将传感器投入液体容器中，如图所示，用尺子测量水位约 15cm。

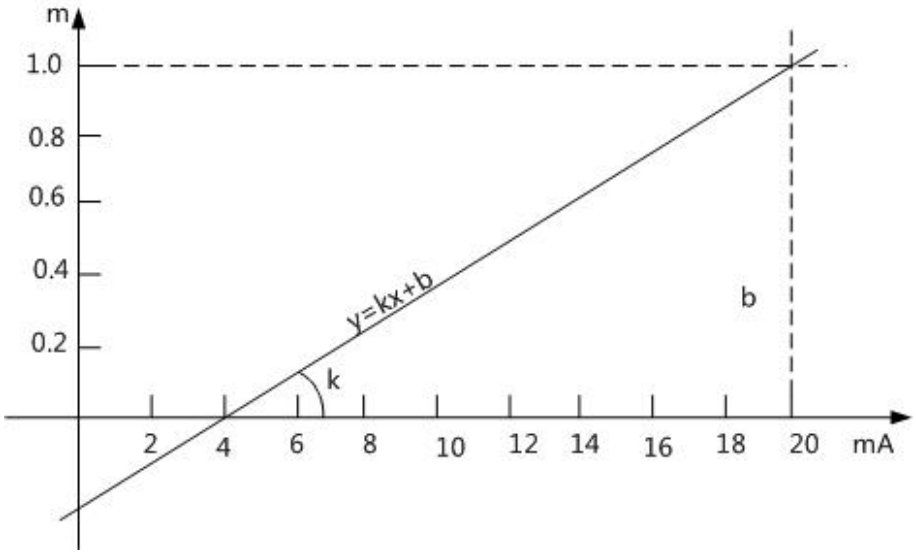


采集模块采集到的电流值发生了明显变化，电流值为 6.57mA，如图所示。



4.数据计算

实际应用时，通常认为信号的电流值与实际量程成线性关系，根据数学知识，可做线性坐标系，如图所示。横坐标为电流值，范围为 4-20mA，纵坐标为量程，范围是 0-1 米。



可以求得：k=1/16，b=-1/4。

因此：y=(1/16)x-1/4。

当前，采集器测量的电流值 6.57mA，带入计算公式，可求得液位为 0.16m，

与实际人工测量值基本一致。

2.5. 基于多路采集模块的开关量采集

2.5.1. 开关量信号采集原理

在工业测控领域，数字信号有编码数字（二进制数或十进制数）、开关量、脉冲序列等。各种按键、继电器和无触点开关（晶体管、晶闸管等）是典型的开关量，而控制步进电机的则是脉冲序列信号。这些信号有高电平和低电平两种状态，相当于二进制数的 1 和 0，计算机处理较为方便。由于数字量信号是计算机直接能接收和处理的信号，所以数字量输入输出通道比较简单，主要是解决信号的缓冲和锁存问题。

如果开关量信号按照一定的周期变化，这样的信号也称为脉冲量信号。频率、转速的测量一般都是通过对传感器输出的脉冲计量来实现的。在运动控制中，编码器输出的信号也是脉冲信号，根据脉冲的数目，可以知道电机角位移和转速。还可以输出脉冲信号控制步进电机的转角和转速。

一、常用术语

在开关量信号中，通常有常开、常闭的类型。其中，

常开，符号是 NO（Normal Open），未触发时，回路是断开的，触发后，回路闭合。

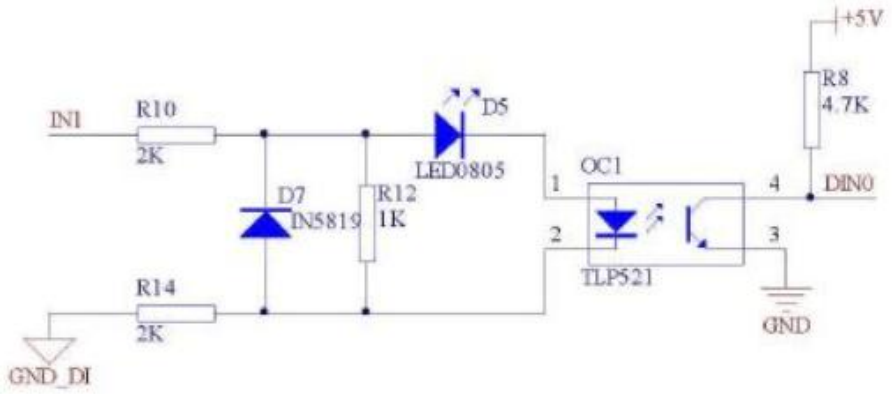
常闭，符号是 NC（Normal Close），未触发时，回路是闭合的，触发后，回路断开。

二、开关量信号的采集电路

开关量采集电路一般由光电耦合器件、保护二极管、限流电阻等组成。

举例说明开关量采集电路设计原理。设定当输入范围为 18~24VDC 时，认为是高电平，被检测设备处于工作状态；当输入范围低于 18VDC 时，认为是低电平，被检测设备处于停止状态。为了避免电气特性及恶劣工作环境带来的干扰，采集电路常采用光电耦合器 TLP521 对信号进行了一次电——光——电的转换，从而起到输入/输出隔离作用。

同时安装有 LED 工作指示灯，可以使用对每一通路的工作情况一目了然。典型的开关量采集电路如图所示。



光耦 TLP521 将红外发光二极管和发光三极管相互绝缘组合在一起，发光二极管为输入回路，将电能转换为光能；发光三极管为输出回路，再将光能转换为电能，实现了两部分电路的电气隔离。

当输入 IN1 为 18~24VDC 时，认为是高电平，希望此时光耦导通，电阻 R10、R14 和发光二极管共同构成输入回路。根据光耦导通时电流为 4-10mA，当输入最高电压 24V 时，

$$\frac{24V}{10mA} < R10 + R14 < \frac{24V}{4mA}\text{，即 } 2.4k\Omega < R10 + R14 < 6k\Omega$$

当输入 IN1 低于 18V 时，认为低电平，此处光耦工作电流肯定要低于 4mA 才能不导通，电阻 R10、R14 和 R12 共同构成输入回路，所以，

$$\frac{18V}{R10 + R14 + R12} < 4mA \text{，即 R10+R14+R12>4.5k}\Omega$$

因此，在设计中，选择 R10=R14=2KΩ，R12=1KΩ。

光耦导通的最小电流为 4mA，根据光耦的电流传输比 CTR（Current Transfer Ratio）为 50%，指当管压降 Uce 足够大时，集电极电流 IC 与发光二极管输入电流 IF 的百分比，所以集电极电流 Ic=IF*50%=4mA*50%=2mA，同时为了使光电三极管尽快进入饱和区，选取上拉电阻 R8 为 4.7KΩ。

最后，为了保护光耦，防止大的输入电压突变，在限流电阻 R12 的两端并联肖特基二极管 IN5819。

这样，一路开关量信号的采集电路就设计好了，同理可设计多路开关量采集电路。实际工业采集控制领域，可购买开关量采集模块，通过协议访问寄存器直

接读取开关量信号的状态值，更方便。

2.5.2. 红外光电开关数据采集

1.传感器概述

红外光电开关，是利用被检测物体对调制的红外光束的遮挡或反射来检测有无被检测物体的。当被检测物体经过检测区域时，红外光电开关的输出状态就会翻转，以达到自动检测的目的。红外光电开关的检测物体不同于金属，其他对红外光有反射、遮挡能力的物体均能检测。常用于流水线分拣。

本系统采用的红外光电开关型号为E3F-DS30C4，其性能特点为：

检测方式：漫反射红外光电检测

检测距离：7-30cm，可调；

工作电压：DC6V-36V

输出电流：300mA

输出信号：三根线，电源正极、电源负极、输出信号

输出模式：NPN NO（常开），感应到反射光线被阻挡时，灯亮常开，灯灭常闭。

响应时间：2ms以下。

2.硬件连接

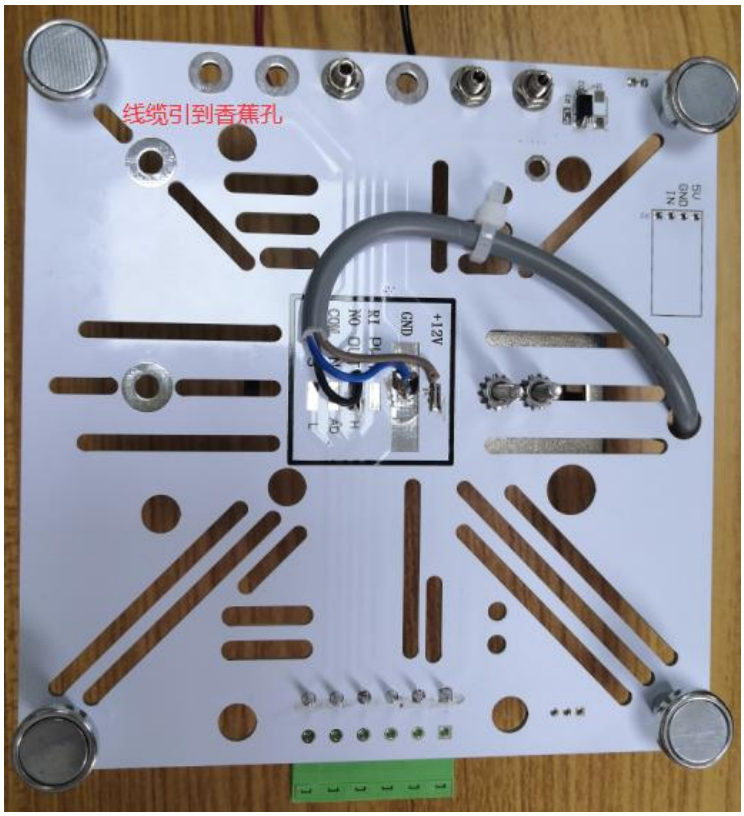
红外光电开关属于三线制开关量信号传感器，信号含义如下表所示。

线色	说明

电源	棕色	电源正
	蓝色	电源负
信号	黑色	输出信号

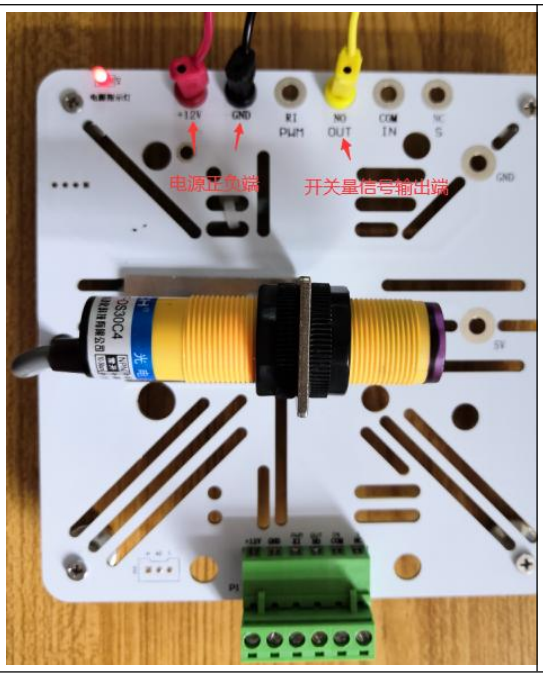
其中电源正，接 12V 直流电源的正极；电源负，接 12V 直流电压的负极；信号，接采集器的开关量输入通道。本系统，我们可以使用多路采集控制模块实现开关量信号的采集。

为方便布线实训，将光电开关固定在磁吸底板上，将电源线、信号线焊接在磁吸底板上，通过磁吸电路板将信号引到香蕉孔上，方便插拔和重复训练，如图所示。



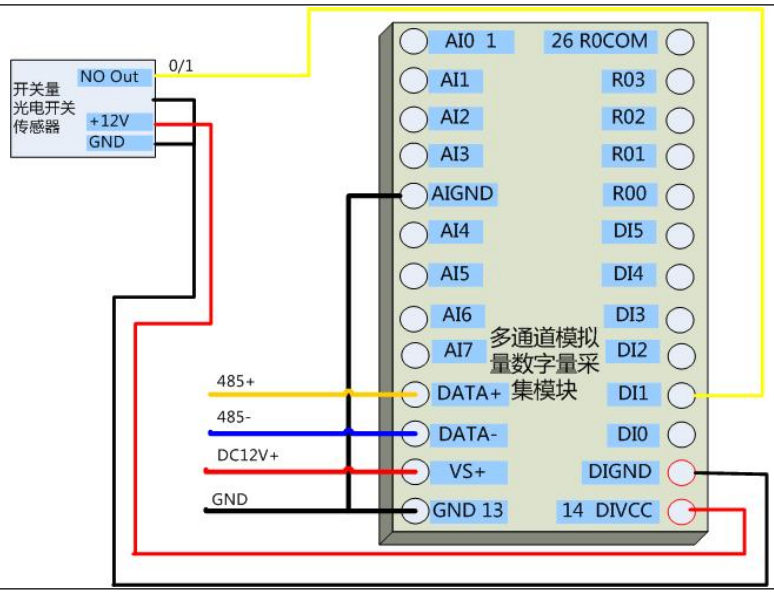
本系统使用多路采集器实时采集红外光电开关的开关量信号。

名称	接线说明



①电源正：12V 供电，红色香蕉头线，一端接入多通道采集模块的 DIVCC 处，另一端接入传感器磁吸底板的+12V 处。
②电源负：黑色香蕉头线，一端接入接入多通道采集模块的 DIGND 处，另一端接入传感器磁吸底板的 GND 处。
多路采集模块给光电开关提供电源
③信号：黄色香蕉头线，一端接入多通道采集模块的数字量输入端 DI1 通道处，另一端接入传感器磁吸底板的 NO(OUT)接口处。

连接示意图



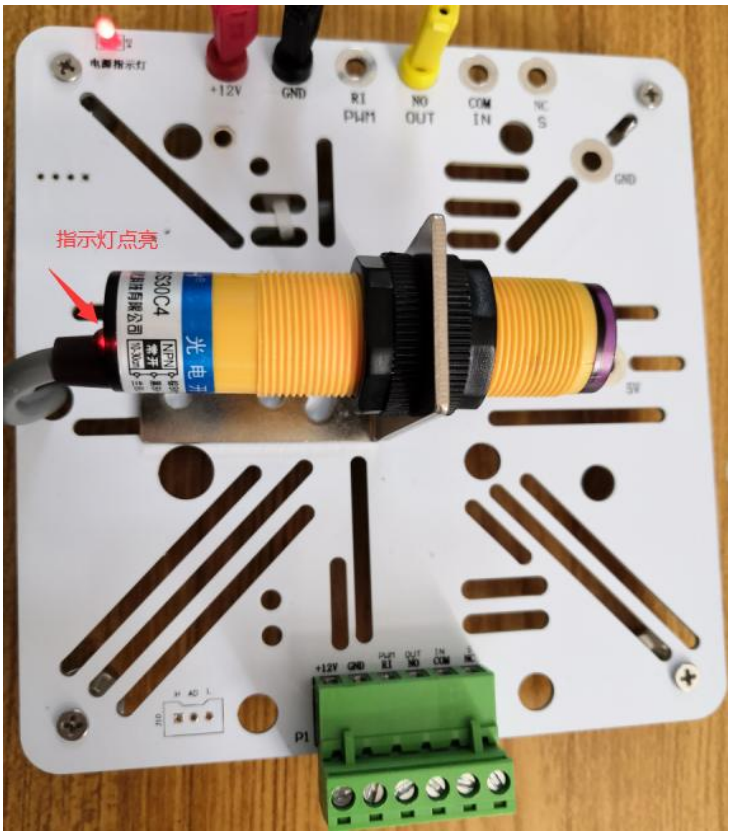
3.信号采集测试

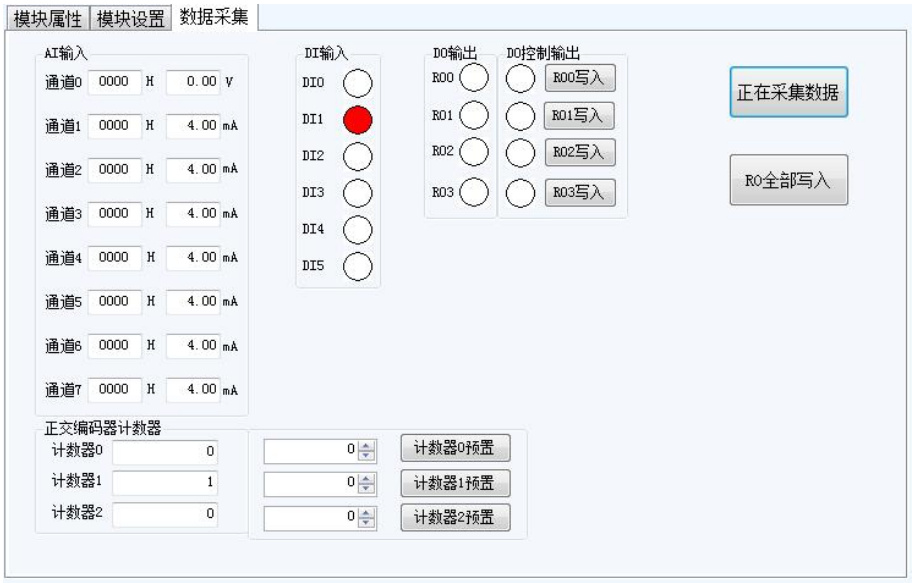
接线完成，上电，光电开关传感器磁吸底板电源指示灯会被点亮，多通道采集器的电源指示灯 POW/SET 点亮，说明电源接线正常。

打开多通道采集器的上位机配置软件 ModbusConfig，切换到“数据采集”选项页，观察 DI 输入通道的第 1 通道 DI1。

当光电开关前方没有障碍物时，光电开关本身的指示灯熄灭，，多通道采集器的 DI1 指示灯熄灭，该界面 DI1 为空。

当光电开关前方放置障碍物时，光电开关本身的指示灯点亮，多通道采集器的 DI1 指示灯点亮，该界面 DI1 着色，显示检测到物体。





4.数据计算

实际应用时，当未检测到障碍物时，数据为 0，检测到障碍物时，数据为 1。

2.5.3. 金属接近开关数据采集

1.传感器概述

金属接近开关，是一种无需与运动部件进行机械直接接触就可以操作的位置开关，当物体接近开关的感应面达到动作距离时，就可以感应到。接近开关是开关型传感器（即无触点开关），它既有行程开关、微动开关的特性，同时具有传感性能，且动作可靠，稳定，频率响应快。适用于机床设备、纺织业、车间流水线、快递业、化工业等各种行业物体位置检测。

本系统采用的电感式接近开关，型号为LJ18A3-8-Z/BX，其性能特点为：

检测方式：电感感应

检测距离：8mm；

工作电压：DC6V-36V

输出电流：最大300mA

输出信号：三根线，电源正极、电源负极、输出信号

输出模式：NPN NO（常开），感应到金属物体时，灯亮常开，灯灭常闭。

响应时间：2ms以下。

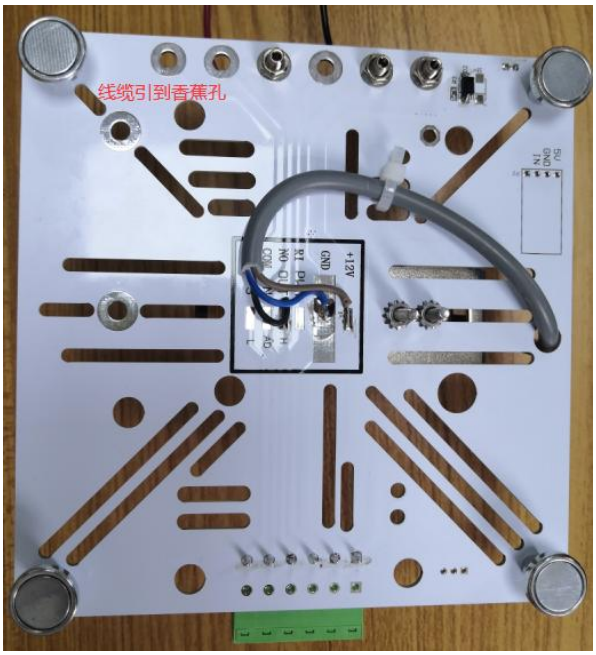
2.硬件连接

金属接近开关属于三线制开关量信号传感器，信号含义如下表所示。

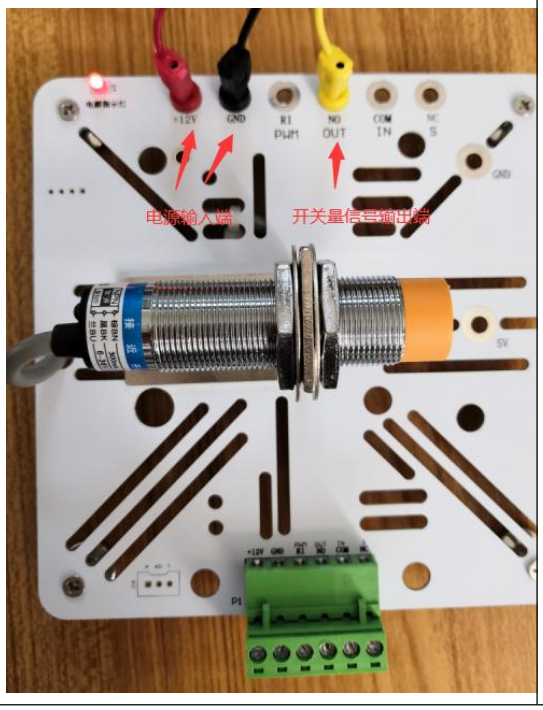
	线色	说明
电源	棕色	电源正
	蓝色	电源负
信号	黑色	输出信号

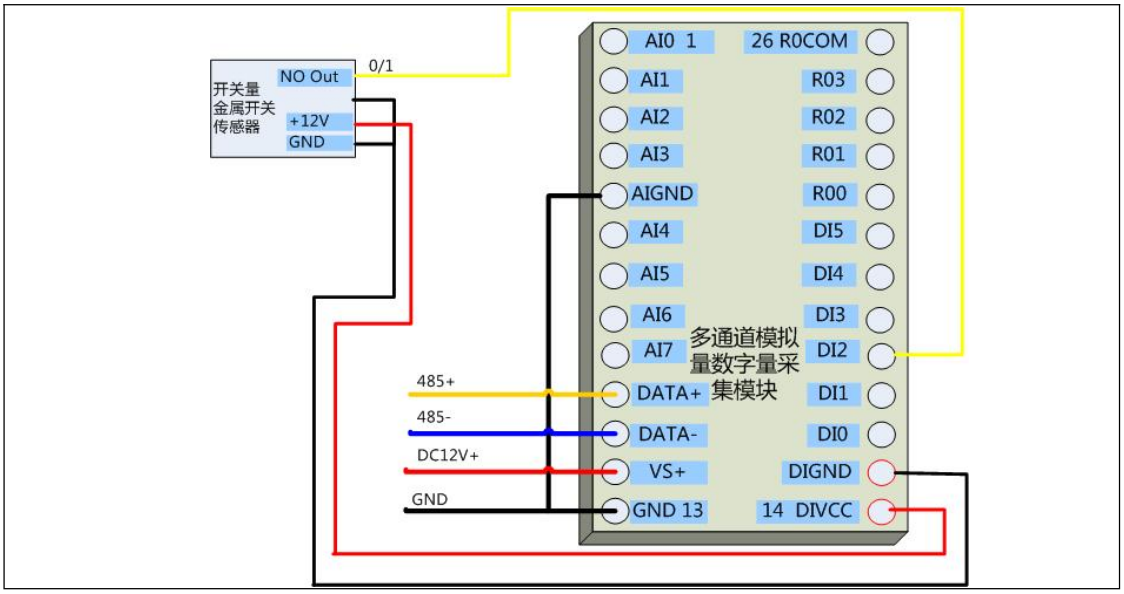
其中电源正，接 12V 直流电源的正极；电源负，接 12V 直流电压的负极；黑色线，输出信号。本系统，我们使用多路采集控制模块实现开关量信号的采集。

为方便布线实训，将金属接近开关固定在磁吸底板上，将电源线、信号线焊接在磁吸底板上，通过磁吸电路板将信号引到香蕉孔上，方便插拔和重复训练，如图所示。



本系统使用多路采集器实时采集金属接近开关的开关量信号。

名称	接线说明
	①电源正：12V 供电，红色香蕉头线，一端接入多通道采集模块的 DIVCC 处，另一端接入传感器磁吸底板的+12V 处。 ②电源负：黑色香蕉头线，一端接入接入多通道采集模块的 DIGND 处，另一端接入传感器磁吸底板的 GND 处。 多路采集模块给光电开关提供电源 ③信号：黄色香蕉头线，一端接入多通道采集模块的数字量输入端 DI2 通道处，另一端接入传感器磁吸底板的 NO(OUT)接口处。
连接示意图	



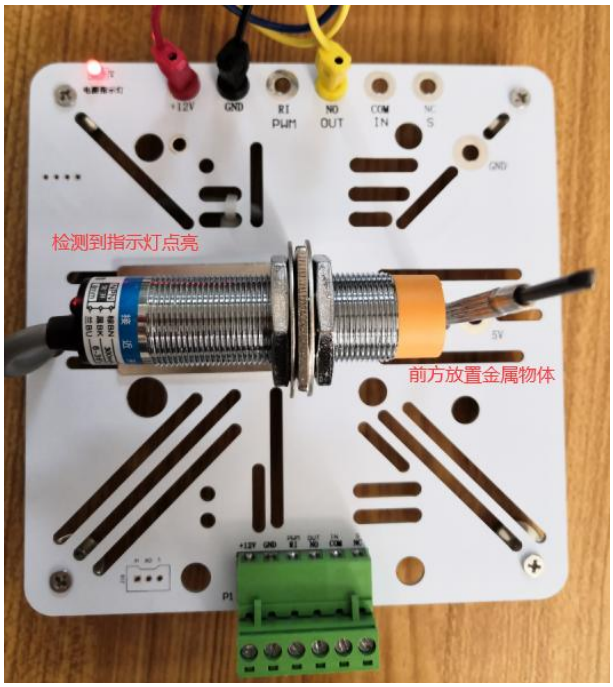
3.信号采集测试

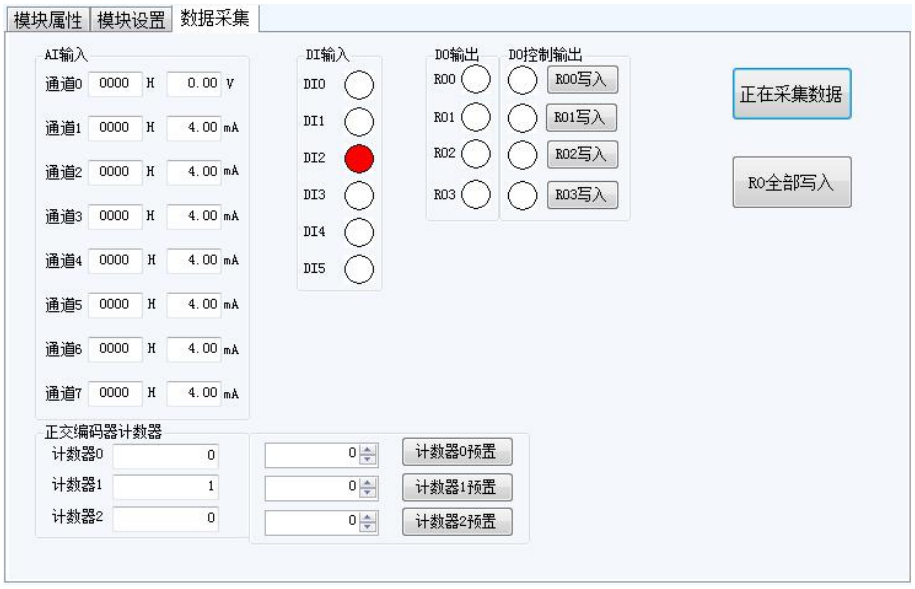
接线完成，上电，金属接近开关传感器磁吸底板电源指示灯会被点亮，多通道采集器的电源指示灯 POW/SET 点亮，说明电源接线正常。

打开多通道采集器的上位机配置软件 ModbusConfig，切换到“数据采集”选项页，观察 DI 输入通道的第 2 通道 DI2。

当金属接近开关前方没有障碍物时，接近开关本身的指示灯熄灭，多通道采集器的 DI2 指示灯熄灭，该界面 DI2 为空。

当金属接近开关前方放置障碍物时，接近开关本身的指示灯点亮，多通道采集器的 DI2 指示灯点亮，该界面 DI2 着色，代表检测到物体。如图所示。





4.数据计算

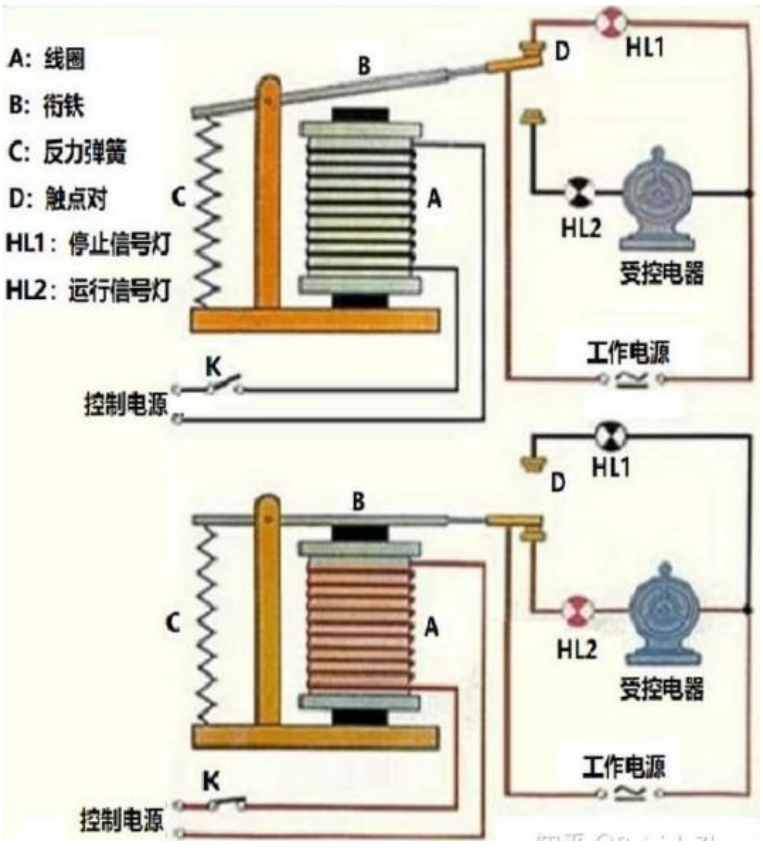
未检测到金属物体时，数据为 0，检测到金属物体时，数据为 1。

2.6. 基于多路采集模块的继电器控制

2.6.1. 继电器工作原理

继电器一般由铁芯、线圈、衔铁、反力弹簧、触点对等组成的。只要在线圈两端加上一定的电压，线圈中就会流过一定的电流，从而产生电磁效应，衔铁就会在电磁力吸引的作用下克服返回弹簧的拉力吸向铁芯，从而带动衔铁的动触点与静触点（常开触点）吸合。当线圈断电后，电磁的吸力也随之消失，衔铁就会在弹簧的反作用力返回原来的位置，使动触点与原来的静触点（常闭触点）吸合。这样吸合、释放，从而达到了在电路中的导通、切断的目的。对于继电器的“常开、常闭”触点，可以这样来区分：继电器线圈未通电时处于断开状态的静触点，称为“常开触点”；处于接通状态的静触点称为“常闭触点”。

10



继电器主要技术参数定义：

额定工作电压：是指继电器正常工作时线圈所需要的电压。根据继电器的型号不同，一般使用直流电压，但交流继电器可以是交流电压。

继电器直流电阻：是指继电器中线圈的直流电阻，可以通过三用电表测量。

继电器的接触电阻：指继电器中接点接触后的电阻值。

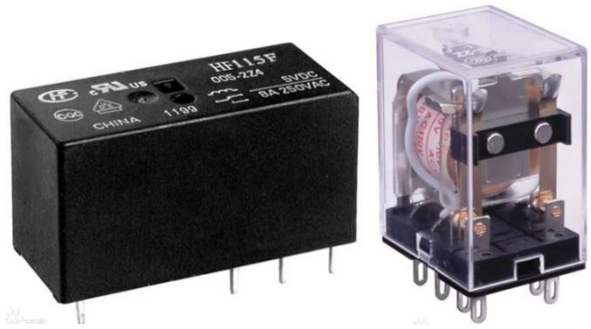
继电器的吸合电流或电压：是指继电器能够产生吸合动作的最小电流或最小电压。在正常使用时，给定的电流必须略大于吸合电流，这样继电器才能稳定地工作。而对于线圈所加的工作电压，一般也不要超过额定工作电压的 1.5 倍，否则会产生较大的电流而把线圈烧毁。

继电器的释放电流或电压：是指继电器产生释放动作的最大电流或最大电压。当继电器吸合状态的电流减小到一定程度时，继电器就会恢复到未通电的释放状态。这时的电流远远小于吸合电流。

继电器的触点切换电压和电流：是指继电器接点允许承载的电压和电流。它决定了继电器能控制的电压和电流大小，使用时不能超过此值，否则很容易损坏继电器的触点。

继电器通常用于自动控制电路中，实际上用较小的电流去控制较大电流的一种“自动开关”。

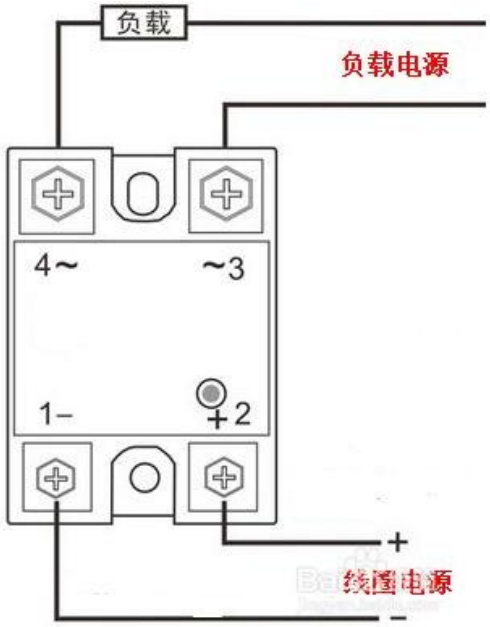
常用的继电器有很多，比如中间继电器、热继电器、固态继电器等。



2.6.2. 继电器的接线与驱动电路

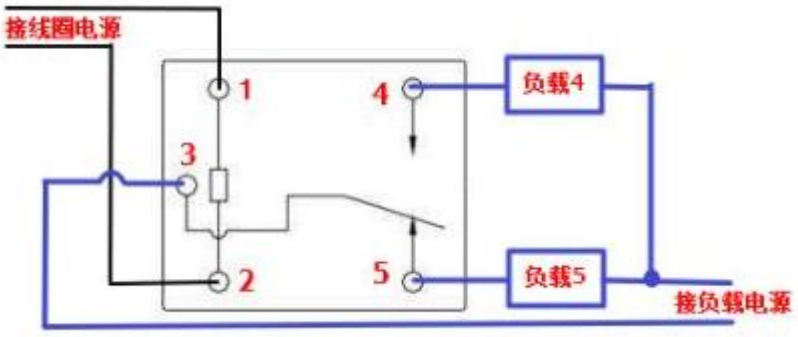
1.继电器的接线

（1）4 脚继电器接线方法，如图所示。4 脚继电器为常见的单刀单掷继电器。1、2 脚为线圈端子，3、4 脚为触点端子。1、2 导通，线圈通电，内部机械传动，3、4 触点闭合（常闭型则断开），负载回路导通工作。

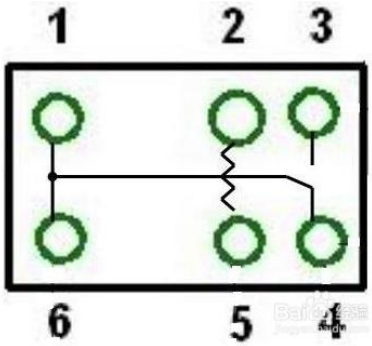


（2）5 脚继电器接线方法，如图所示。5 脚继电器为常见的单刀双掷继电器。1、2 为线圈端子，3 为公共触点，4、5 分别为常开、常闭触点。

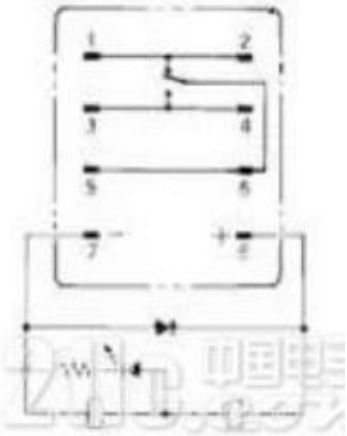
初始状态，3、5 触点导通，负载 5 工作；线圈电源给电后，1、2 导通，线圈通电，3 公共触点切换到与 4 常开触点闭合，负载 4 开始工作。

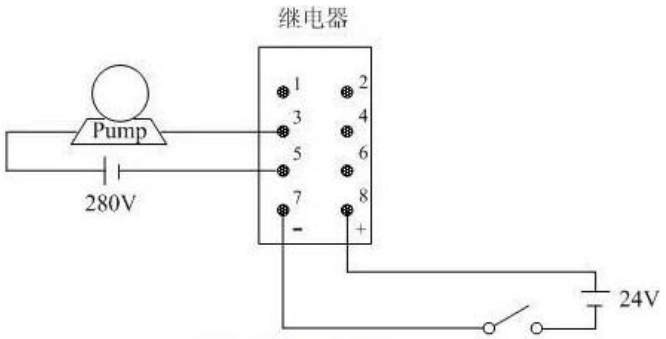


（3）6 脚继电器。与 5 脚继电器原理差别不大，就是多了一个公共端子，1/6 两个端子连接在一起，同时作为公共端，如图所示。



（4）8 脚继电器接线。常规的 8 脚继电器接线如图所示。5 和 6 是公共端，1 和 2 是常闭触点，3 和 4 是常开触点。7 和 8 不通电时，5-6 和 1-2 接通；通电后，5-6 和 3-4 接通。其中 1 和 2 是常闭触点，3 和 4 是常开触点。

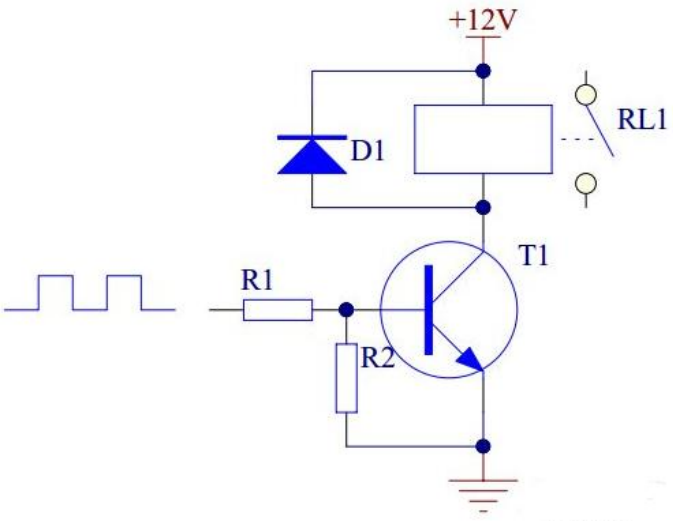




实际使用时，端子 7 和 8 要并联一个续流二极管，二极管接入时的极性要和继电器端子标注的极性相反（8 脚接二极管的负极，7 脚接二极管的正极），目的是让继电器驱动电流断开瞬间产生的较高的感应电动势激发的电流能流过二极管，而不经过其他电路损坏、或击穿其他电路中的元件。

2.继电器的驱动电路

继电器驱动电流一般需要 20-40mA 或更大，线圈电阻 100-200 欧姆，因此需要驱动电路。通常使用晶体管开关电路驱动继电器。



当输入高电平时，晶体管 T1 饱和导通，继电器线圈通电，触点吸合。

当输入低电平时，晶体管 T1 截止，继电器线圈断电，触点断开。

其中，晶体管 T1 为控制开关，电阻 R1 主要起限流作用，降低晶体管 T1 功耗。电阻 R2 使晶体管 T1 可靠截止。二极管 D1 反向续流，为三极管由导通转向关断时为继电器线圈中的电流提供泄放通路，并将其电压钳位在 12V 上。

在实际应用中，可使用多个晶体管集成的芯片，使用集成电路能简化驱动多个继电器的电路设计过程，如常用的驱动芯片 TD62003 等。

工控领域，通常使用市面上继电器控制模块。本系统中，使用多路采集控制模块驱动执行器，如风扇、电磁阀。

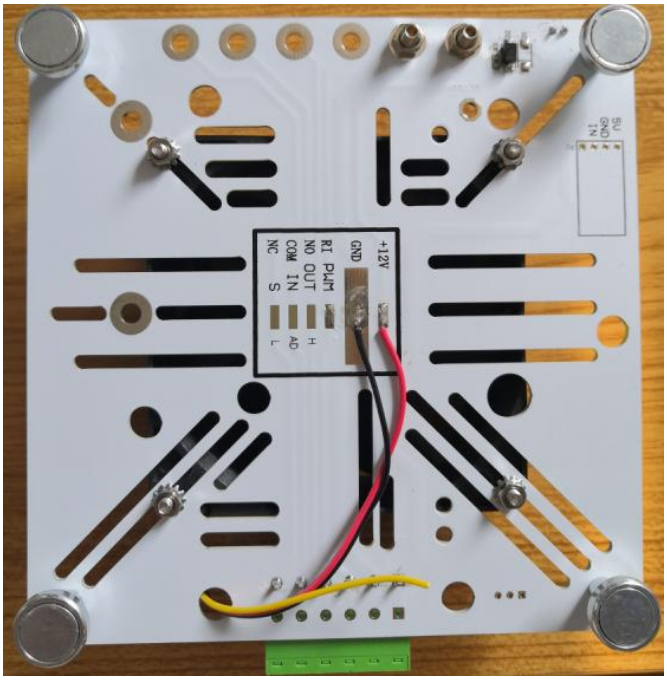
2.6.3. 风扇控制

1.风扇概述

风扇为执行器， 12V直流电源即可转动。

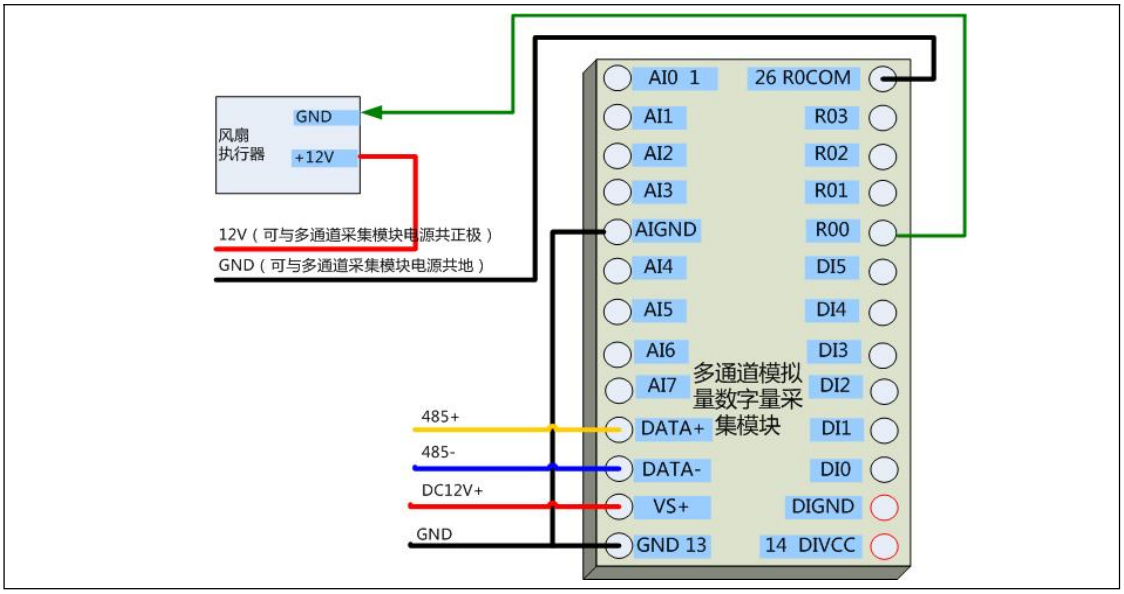
2.硬件连接

风扇执行器有两根线，分别是电源正极、电源负极。只要电源接通，风扇即可转动。为方便实训，将风扇固定在磁吸底板上，将电源线焊接在磁吸底板上，通过磁吸电路板将信号引到香蕉孔上，方便插拔和重复训练。如图所示。



本系统使用采用多路采集器的继电器输出口控制风扇转动。采集器具有 4 个继电器输出通道，内置四个继电器驱动电路，常开型，即未导通时，输出回路是断开的。

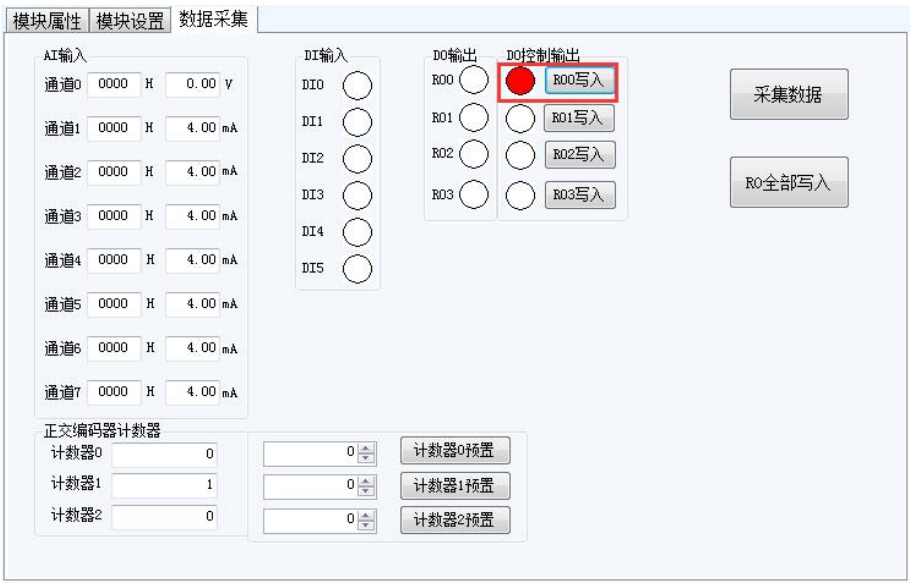
名称	接线说明
	①电源正：12V 供电，红色香蕉头线，一端接入实训台两侧电源接口板的 12V 处（如果是 5V/24 供电，就接对应的电源孔），另一端接入传感器磁吸底板的 +12V 处。 ②电源负：黑色香蕉头线，一端接入多通道采集模块的 RO0 通道，另一端接入传感器磁吸底板的 GND 处。 此时，多路采集模块继电器输出通道 RO0 回路是断开的，所以风扇无法转动。 ③公共端 ROCOM：黄色或绿色香蕉头线，一端接入多通道采集模块的继电器输出端 ROCOM 通道处，另一端接入实训台两侧电源接口板的 GND 处（一定是否负载电源 GND）。
连接示意图	



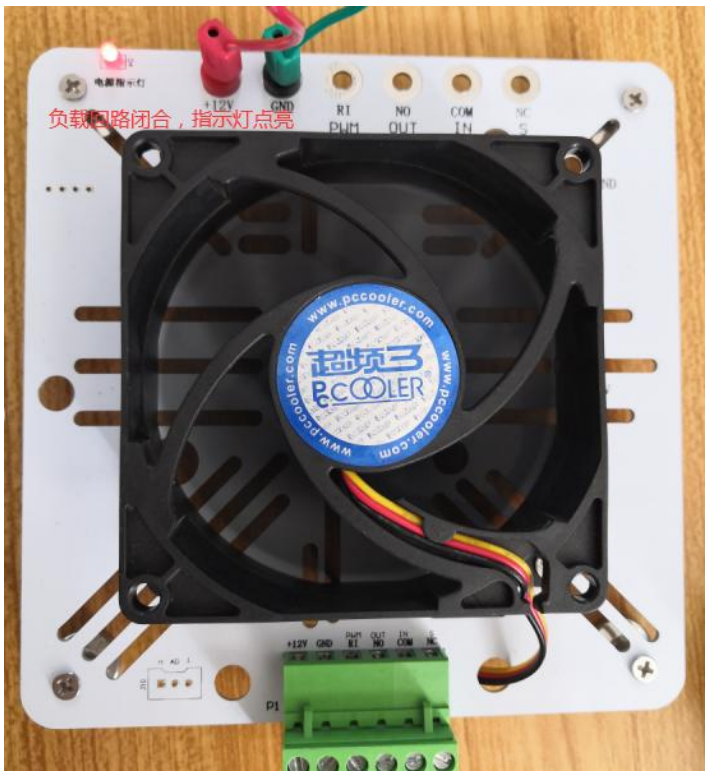
3.风扇驱动控制

接线完成，上电，多通道采集器的电源指示灯 POW/SET 点亮，说明电源接线正常，风扇控制板上的电源指示灯熄灭，因为供电回路是断开的。

打开多通道采集器的上位机配置软件 ModbusConfig，切换到“数据采集”选项页。选中“R00 写入”按钮使其变红，代表控制继电器驱动电路导通，点击“R00 写入”，即可向多路采集器发送驱动命令，如图所示。



驱动电路导通，继电器常开端立即闭合，风扇负载电源回路闭合，风扇转动，电源指示灯点亮，如图所示。



选中“R00 写入”按钮使其变白，代表控制继电器驱动电路截止，点击“R00 写入”，即可向多路采集器发送驱动命令，负载回路断开，风扇停止转动。

2.6.4. 电磁阀控制

1.电磁阀概述

电磁阀为执行器，用于工业上中性气体回路、水油回路通断控制。

本系统使用的电磁阀型号为2W-025-08，为直动式膜片式电磁阀，特性为：

阀体材料：黄铜

供电电源：DC12V

零压力启动，通电立即打开，断电立即关闭。

类型：常闭式

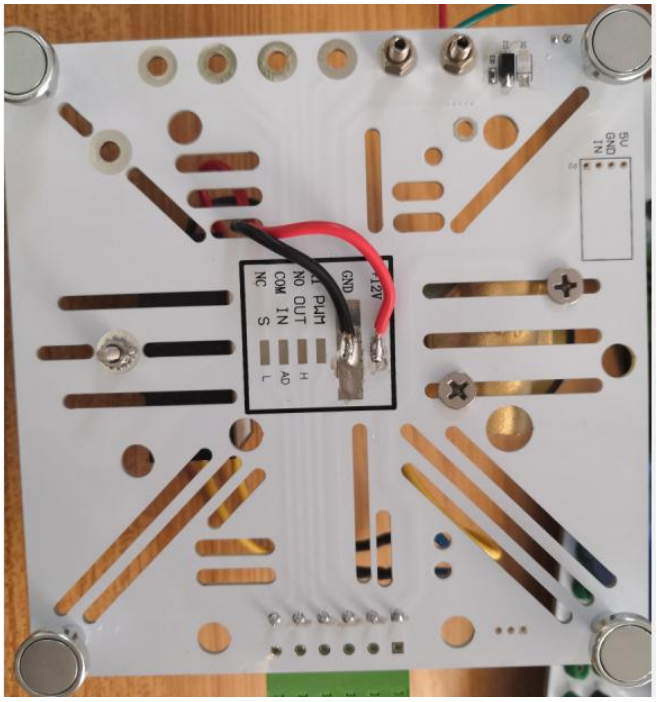
介质温度范围：-5~80℃

压力范围：0-1Mpa

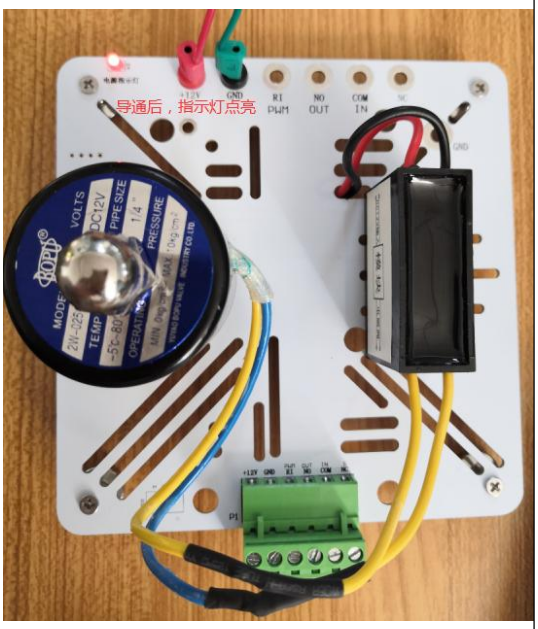
2.硬件连接

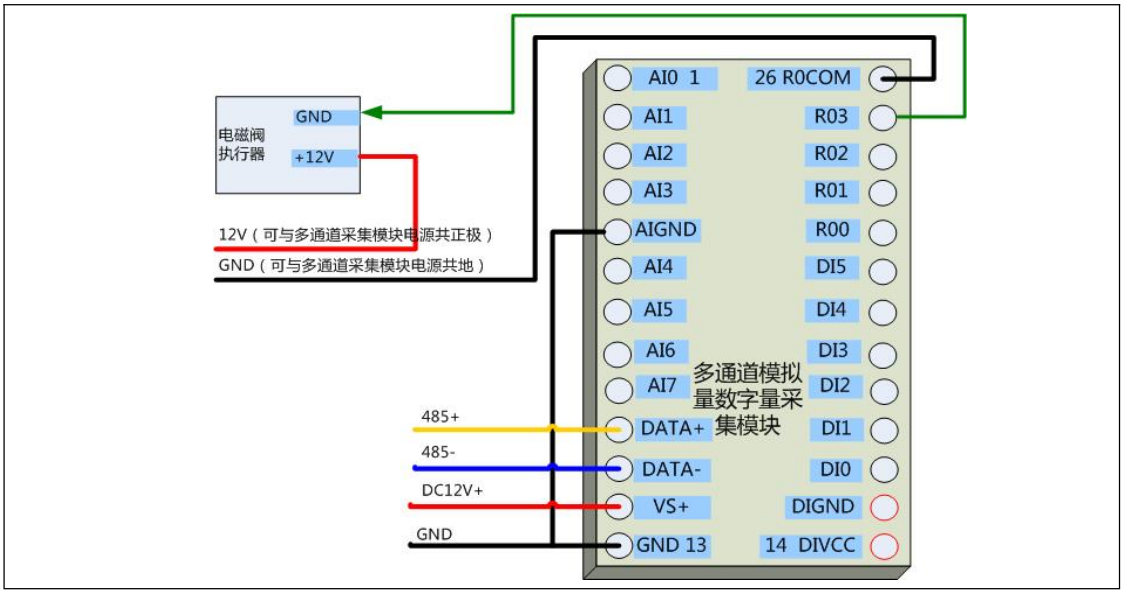
电磁阀执行器有两根线，分别是电源正极、电源负极。只要电源接通，电磁阀的线圈即可通电。为了防止通电过程中线圈发热，电源经过一个四线节能模块

给电磁阀供电。为方便实训，将电磁阀、节能模块均固定在磁吸底板上，将电源线焊接在磁吸底板上，通过磁吸电路板将信号引到香蕉孔上，方便插拔和重复训练。如图所示。



本系统使用采用多路采集器的继电器输出口控制电磁阀转动。采集器具有 4 个继电器输出通道，内置四个继电器驱动电路，常开型，即未导通时，输出回路是断开的。

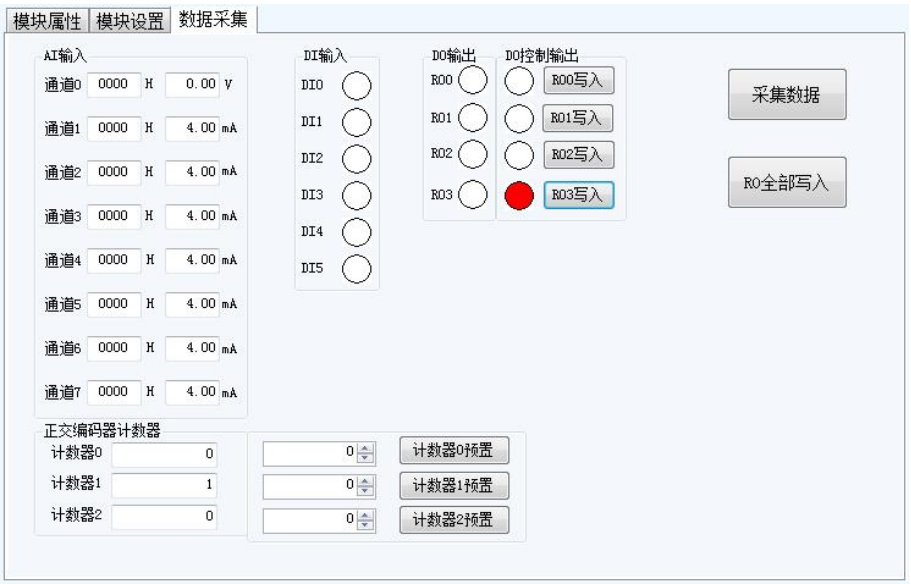
名称	接线说明
	①电源正：12V 供电，红色香蕉头线，一端接入实训台两侧电源接口板的 12V 处（如果是 5V/24 供电，就接对应的电源孔），另一端接入传感器磁吸底板的 +12V 处。 ②电源负：黑色香蕉头线，一端接入多通道采集模块的 RO3 通道，另一端接入传感器磁吸底板的 GND 处。 此时，多路采集模块继电器输出通道 RO0 回路是断开的，所以电磁阀不工作，管道不通。 ③公共端 ROCOM：黄色或绿色香蕉头线，一端接入多通道采集模块的继电器输出端 ROCOM 通道处，另一端接入实训台两侧电源接口板的 GND 处（一定是否负载电源 GND）。
连接示意图	



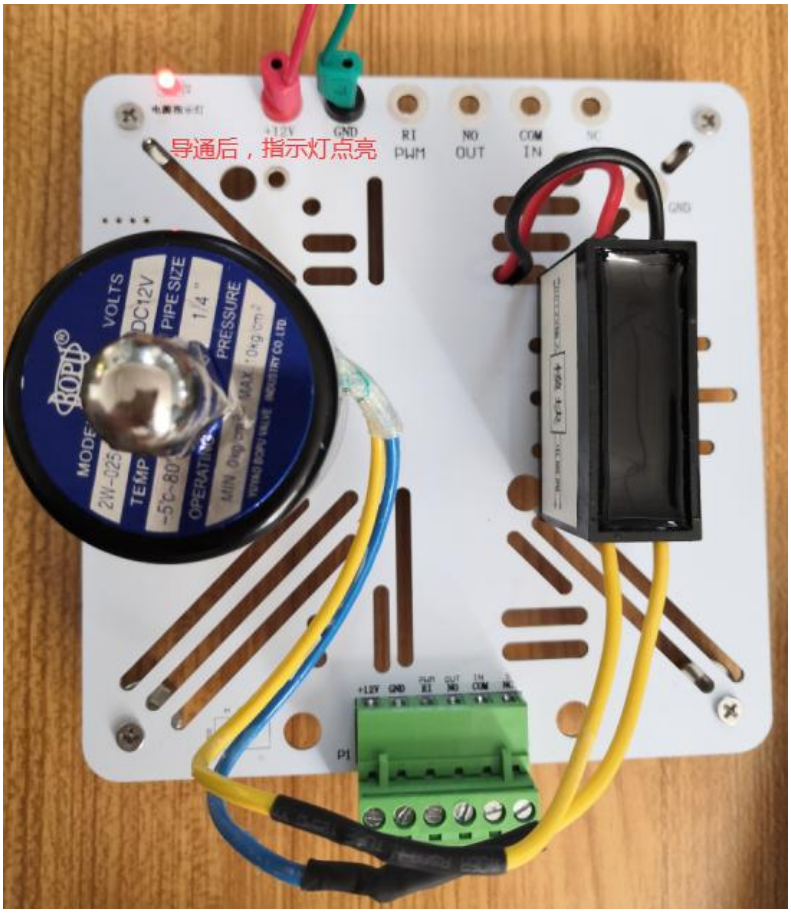
3.电磁阀驱动控制

接线完成，上电，多通道采集器的电源指示灯 POW/SET 点亮，说明电源接线正常，电磁阀控制板上的电源指示灯熄灭，因为供电回路是断开的。

打开多通道采集器的上位机配置软件 ModbusConfig，切换到“数据采集”选项页。选中“R03 写入”按钮使其变红，代表控制继电器驱动电路导通，点击“R03 写入”，即可向多路采集器发送驱动命令，如图所示。



驱动电路导通，继电器常开端立即闭合，电磁阀负载电源回路闭合，电磁阀上电导通，电源指示灯点亮，如图所示。



选中“R03 写入”按钮使其变白，代表控制继电器驱动电路截止，点击“R03 写入”，即可向多路采集器发送驱动命令，负载回路断开，电磁阀供电断开。

2.7. 基于嵌入式的模拟量信号采集

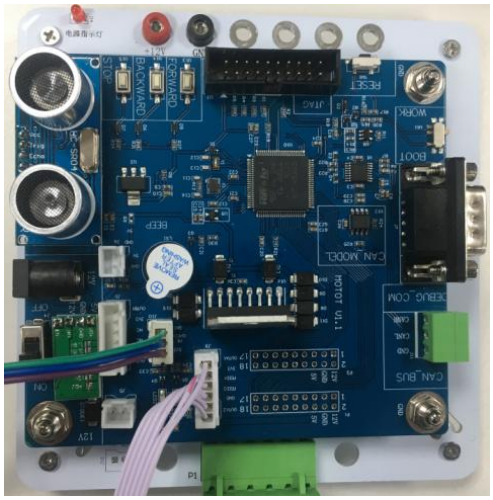
2.7.1. 实验目的

理解模拟量信号的概念；

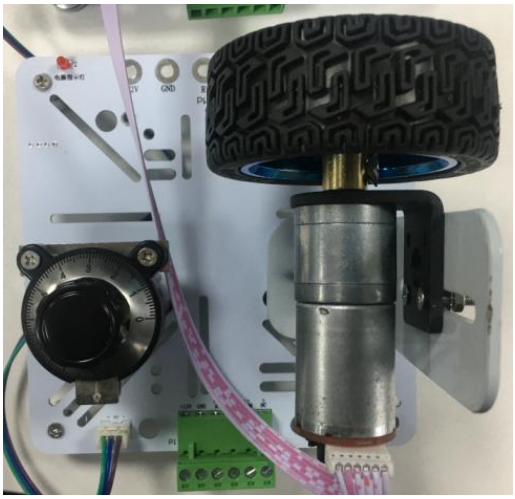
掌握模拟量信号的嵌入式单片机采集方法。

2.7.2. 实验环境

硬件环境：工业数据采集实训台的 CAN 测控终端和直流电机转动轮。



CAN 测控终端



车轮控制板

软件环境：Windows 7 及以上操作系统，Keil4 for ARM 4.7 开发环境。

2.7.3. 实验原理

ADC 简介

ADC（analog to digital converter）也称为模数转换器，是指一个将模拟量转变为数字量。（A:模拟信号，D:数字信号），ADC 就是起到把连续的信号用离散的数字表达出来的作用。

模拟量：就是指变量在一定范围内连续变化的量，也就是在一定范围内可以取任意值。比如日常使用的卷尺，它总长是 1 米，我既可以抽出来 0.5 米，也可以抽出来 0.22555....米，在 1 米范围内任意取。

ADC 主要技术指标

1.ADC 的位数

一个 n 位的 ADC 表示这个 ADC 共有 2 的 n 次方个刻度。8 位的 ADC，输出的是从 0 到 255 一共 256 个数字量，也就是 2 的 8 次方个数据刻度。

2.基准源

基准源，也叫基准电压，是 ADC 的一个重要指标，要想把输入 ADC 的信号测量准确，那么基准源首先要准，基准源的偏差会直接导致转换结果的偏差。就像那根卷尺，被火烤了热涨冷缩变长了，自然误差变大了。

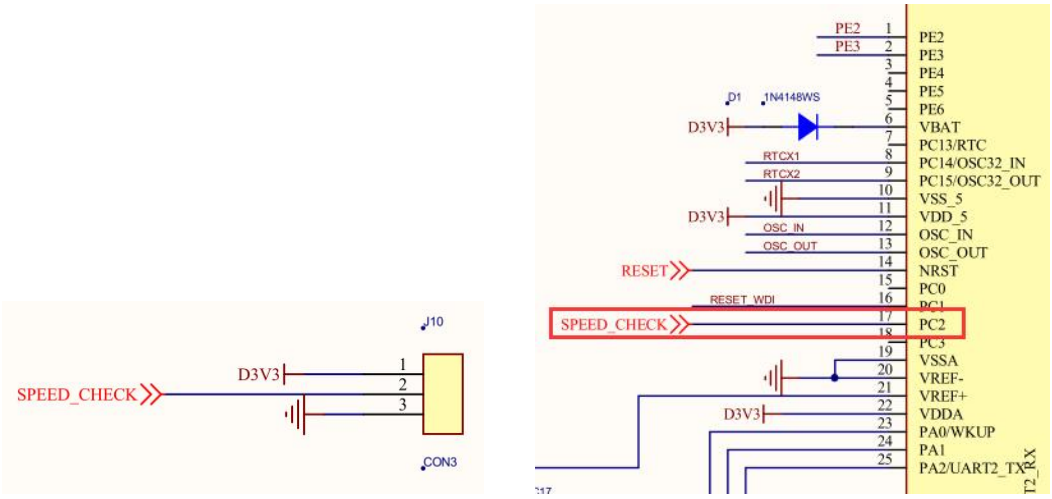
3.分辨率

分辨率是数字量变化一个最小刻度时，模拟信号的变化量，定义为满刻度量程与 2n-1 的比值。5.10V 的电压系统，使用 8 位的 ADC 进行测量，那么相当于 0 到 255 一共 256 个刻度，把 5.10V 平均分成了 255 份，那么分辨率就是 5.10/255 = 0.02V。

2.7.4. 实验内容

电位器是具有三个引出端、阻值可按某种变化规律调节的电阻元件。电位器通常由电阻体和可移动的电刷组成。当电刷沿电阻体移动时，在输出端即获得与位移量成一定关系的电阻值或电压。

电位器既可作三端元件使用也可作二端元件使用。后者可视作一可变电阻器，由于它在电路中的作用是获得与输入电压（外加电压）成一定关系得输出电压，因此称之为电位器。硬件原理图：



如图所示：电位器连接 CON3，输入电压为 3.3V。按钮电位器模拟量输出端输出电压 0~3.3V，连接到了 STM32 的 PC2 端口，通过芯片手册可以查到 PC2 是 ADC123_IN12,代表是 ADC1 或者 2，3 的第 12 个通道。

STM32 拥有 1~3 个 ADC（STM32F101/102 系列只有 1 个 ADC），这些 ADC 可以独立使用，也可以使用双重模式（提高采样率）。STM32 的 ADC 是 12 位逐次逼近型的模拟数字转换器。它有 18 个通道，可测量 16 个外部和 2 个内部信号源。各通道的 A/D 转换可以单次、连续、扫描或间断模式执行。ADC 的结果可以左对齐或右对齐方式存储在 16 位数据寄存器中。模拟看门狗特性允许应用程序检测输入电压是否超出用户定义的高/低阀值。

STM32F103 系列最少都拥有 2 个 ADC，我们选择的 STM32F103CBT6 芯片正是包含 2 个 ADC。STM32 的 ADC 最大的转换速率为 1Mhz，也就是转换时间为 1us（在 ADCCLK=14M,采样周期为 1.5 个 ADC 时钟下得到），不要让 ADC 的时钟超过 14M，否则将导致结果准确度下降。

STM32 将 ADC 的转换分为 2 个通道组：规则通道组和注入通道组。规则通道相当于你正常运行的程序，而注入通道呢，就相当于中断。在你程序正常执行的时候，中断是可以打断你的执行的。同这个类似，注入通道的转换可以打断规则通道的转换，在注入通道被转换完成之后，规则通道才得以继续转换。

这里，我们介绍一下我们执行规则通道的单次转换，需要用到的 ADC 寄存器。主要用到了：ADC 控制寄存器（ADC_CR1 和 ADC_CR2）、ADC 采样事件寄存器(ADC_SMPR1 和 ADC_SMPR2）、ADC 规则序列寄存器(ADC_SQR1~3)、ADC 规则数据寄存器(ADC_DR)和 ADC 状态寄存器(ADC_SR）。由于寄存器详解太多，我们这里也不再做详细介绍，这些寄存器我们可以在《STM32 中文参考手册》的 282 页的寄存器描述章节上看到详细的介绍。下面我们介绍使用库函数来设定使用 ADC1 的通道 12 进行 AD 转换。这里需要说明一下，使用到的库函数分布在 stm32f10x_adc.c 文件和 stm32f10x_adc.h 文件中。下面我们来看一下 ADC 的详细设置的步骤：

1）开启 PC 口时钟和 ADC1 时钟，设置 PC2 为模拟输入。

STM32F103VCT6 的 ADC1 通道 21 对应的 IO 口为 PC12，所以，我们先要使能 PORTC 的时钟和 ADC1 时钟，然后设置 PC2 为模拟输入。使能 GPIOC 和 ADC 时钟用 RCC_APB2PeriphClockCmd 函数，设置 PC2 的输入方式，使用 GPIO_Init 函数即可。这里我们列出 STM32 的 ADC 通道与 GPIO 对应表：

表 2-1 ADC 通道与 GPIO 对应表

	ADC1	ADC2	ADC3
通道0	PA0	PA0	PA0
通道1	PA1	PA1	PA1
通道2	PA2	PA2	PA2
通道3	PA3	PA3	PA3
通道4	PA4	PA4	PF6
通道5	PA5	PA5	PF7
通道6	PA6	PA6	PF8
通道7	PA7	PA7	PF9
通道8	PB0	PB0	PF10
通道9	PB1	PB1	
通道10	PC0	PC0	PC0
通道11	PC1	PC1	PC1
通道12	PC2	PC2	PC2
通道13	PC3	PC3	PC3
通道14	PC4	PC4	
通道15	PC5	PC5	
通道16	温度传感器		
通道17	内部参照电压		

2）复位 ADC1，同时设置 ADC1 分频因子。

开启 ADC1 时钟之后，我们要复位 ADC1，将 ADC1 的全部寄存器重设为缺省值之后我们就可以通过 RCC_CFGR 设置 ADC1 的分频因子。分频因子要确保 ADC1 的时钟（ADCCLK）不要超过 14Mhz。这个我们设置分频因子位 6，时钟为 72/6=12MHz,库函数的实现方法是：

```
RCC_ADCCLKConfig(RCC_PCLK2_Div6);
```

ADC 时钟复位的方法是：

```
ADC_DeInit(ADC1);
```

这个函数非常容易理解，就是复位指定的 ADC。

3）初始化 ADC1 参数， 设置 ADC1 的工作模式以及规则序列的相关信息。

在设置完分频因子之后，就可以开始 ADC1 的模式配置了，设置单次转换模式、触发方式选择、数据对齐方式等都在这一步实现。同时，我们还要设置 ADC1 规则序列的相关信息，我们这里只有一个通道，并且是单次转换的，所以设置规则序列中通道数为 1。这些在库函数中是通过函数 ADC_Init 实现的，下面我们看看其定义：

```
void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct);
```

从函数定义可以看出，第一个参数是指定 ADC 号。这里我们来看看第二个参数，跟其他外设初始化一样，同样是通过设置结构体成员变量的值来设定参数。

```
typedef struct
{
uint32_t ADC_Mode;
FunctionalState ADC_ScanConvMode;
FunctionalState ADC_ContinuousConvMode;
uint32_t ADC_ExternalTrigConv;
uint32_t ADC_DataAlign;
uint8_t ADC_NbrOfChannel;
}ADC_InitTypeDef;
```

参数 ADC_Mode 顾名思义是用来设置 ADC 的模式。前面讲解过，ADC 的模式非常多，包括独立模式，注入同步模式等等，这里我们选择独立模式，所以参数为 ADC_Mode_Independent。参数 ADC_ScanConvMode 用来设置是否开启扫描模式，因为是单次转换，这里我们选择不开启，DISABLE 即可。

参数 ADC_ContinuousConvMode 用来设置是否开启连续转换模式，因为是单次转换模式，所以我们选择不开启连续转换模式，DISABLE 即可。

参数 ADC_ExternalTrigConv 是用来设置启动规则转换组转换的外部事件，这里我们选择软件触发，选择值为 ADC_ExternalTrigConv_None 即可。

参数 DataAlign 用来设置 ADC 数据对齐方式是左对齐还是右对齐，这里我们选择右对齐方式 ADC_DataAlign_Right。

参数 ADC_NbrOfChannel 用来设置规则序列的长度，这里我们是单次转换，所以值为 1 即可。通过上面对每个参数的讲解，下面来看看我们的初始化范例：

```
ADC_InitTypeDef ADC_InitStructure;
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; //ADC 工作模式:独立模式
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //AD 单通道模式
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //AD 单次转换模式
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
//转换由软件而不是外部触发启动
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right; //ADC 数据右对齐
ADC_InitStructure.ADC_NbrOfChannel = 1; //顺序进行规则转换的 ADC 通道的数目
1
ADC_Init(ADC1, &ADC_InitStructure); //根据指定的参数初始化外设 ADCx
```

4）使能 ADC 并校准。

在设置完了以上信息后， 我们就使能 AD 转换器，执行复位校准和 AD 校准，注意这两步是必须的！不校准将导致结果很不准确。

使能指定的 ADC 的方法是：

```
ADC_Cmd(ADC1, ENABLE); //使能指定的 ADC1
```

执行复位校准的方法是：

```
ADC_ResetCalibration(ADC1);
```

执行 ADC 校准的方法是：

```
ADC_StartCalibration(ADC1); //开始指定 ADC1 的校准状态
```

记住，每次进行校准之后要等待校准结束。 这里是通过获取校准状态来判断是否校准是否结束。

下面我们一一列出复位校准和 AD 校准的等待结束方法：

```
while(ADC_GetResetCalibrationStatus(ADC1)); //等待复位校准结束
while(ADC_GetCalibrationStatus(ADC1)); //等待校 AD 准结束
```

5）读取 ADC 值。

在上面的校准完成之后，ADC 就算准备好了。接下来我们要做的就是设置规则序列 1 里面的通道，采样顺序，以及通道的采样周期，然后启动 ADC 转换。在转换结束后，读取 ADC 转换结果值就是了。这里设置规则序列通道以及采样周期的函数是：

```
void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel,
uint8_t Rank, uint8_t ADC_SampleTime);
```

这里是规则序列中的第 1 个转换，同时采样周期为 239.5，所以设置为：

```
ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_239Cycles5 );
```

软件开启 ADC 转换的方法是：

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);//使能指定的 ADC1 的软件转换启动功能
```

开启转换之后，就可以获取 ADC 转换结果数据， 方法是：

```
ADC_GetConversionValue(ADC1);
```

同时在 AD 转换中，我们还要根据状态寄存器的标志位来获取 AD 转换的各个状态信息。 库函数获取 AD 转换的状态信息的函数是：

```
FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint8_t ADC_FLAG)
```

比如我们要判断 ADC1 的转换是否结束，方法是：

```
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束
```

6）最后将采集到的 ADC 值通过公式转换，以 0~3.3V 形式通过串口打印。

总结使用 ADC 步骤：

第一步：ADC 初始化规则通道，默认 0~3 通道；

```
void  Adc_Init(void)
{
   ADC_InitTypeDef ADC_InitStructure;
   GPIO_InitTypeDef GPIO_InitStructure;

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC |RCC_APB2Periph_ADC1  , ENABLE );   //使能ADC1通道时钟


   RCC_ADCCLKConfig(RCC_PCLK2_Div6);   //设置ADC分频因子6 72M/6=12,ADC最大时间不能超过14M

   //PA1 作为模拟通道输入引脚
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;      //模拟输入引脚
   GPIO_Init(GPIOC, &GPIO_InitStructure);

   ADC_DeInit(ADC1);  //复位ADC1

   ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; //ADC工作模式:ADC1和ADC2工作在独立模式
   ADC_InitStructure.ADC_ScanConvMode = DISABLE;   //模数转换工作在单通道模式
   ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;   //模数转换工作在单次转换模式
   ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;  //转换由软件而不是外部触发启动
   ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;   //ADC数据右对齐
   ADC_InitStructure.ADC_NbrOfChannel = 1;   //顺序进行规则转换的ADC通道的数目
   ADC_Init(ADC1, &ADC_InitStructure); //根据ADC_InitStruct中指定的参数初始化外设ADCx的寄存器


   ADC_Cmd(ADC1, ENABLE);  //使能指定的ADC1

   ADC_ResetCalibration(ADC1);   //使能复位校准

   while(ADC_GetResetCalibrationStatus(ADC1));  //等待复位校准结束

   ADC_StartCalibration(ADC1);    //开启AD校准

   while(ADC_GetCalibrationStatus(ADC1));  //等待校准结束

   // ADC_SoftwareStartConvCmd(ADC1, ENABLE);      //使能指定的ADC1的软件转换启动功能

}
```

第二步：多次采集通道值，提高准确率；

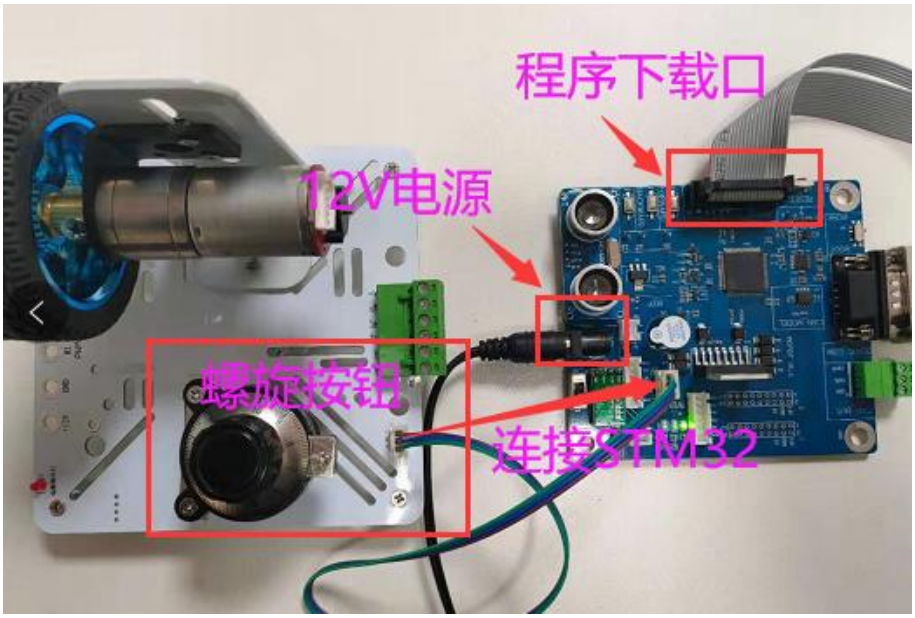
```
//获得ADC值
//ch:通道值 0~3
u16 Get_Adc(u8 ch)
{
   //设置指定ADC的规则组通道，一个序列，采样时间
   ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_239Cycles5 );   //ADC1,ADC通道,采样时间为239.5周期

   ADC_SoftwareStartConvCmd(ADC1, ENABLE);      //使能指定的ADC1的软件转换启动功能

   while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束

   return ADC_GetConversionValue(ADC1);   //返回最近一次ADC1规则组的转换结果
}

u16 Get_Adc_Average(u8 ch,u8 times)
{
   u32 temp_val=0;
   u8 t;
   for(t=0;t<times;t++)
   {
      temp_val+=Get_Adc(ch);
      delay_ms(5);
   }
   return temp_val/times;
}
```

第三步：将得到的值，通过公式转换并且通过串口输出；

```
  while(1)
  {
       adc=Get_Adc_Average(ADC_Channel_12,10);
       temp=(float)adc*(3.3/4096);     //转化成0~3.3V形式
       snprintf((char *)strl, sizeof(strl), "ADC=: %.2lf\r\n",temp); //拼接函数
       UART_Put(USART1,strl,18); //串口输出打印
       delay_ms(1000);
  }
```

2.7.5. 实验步骤

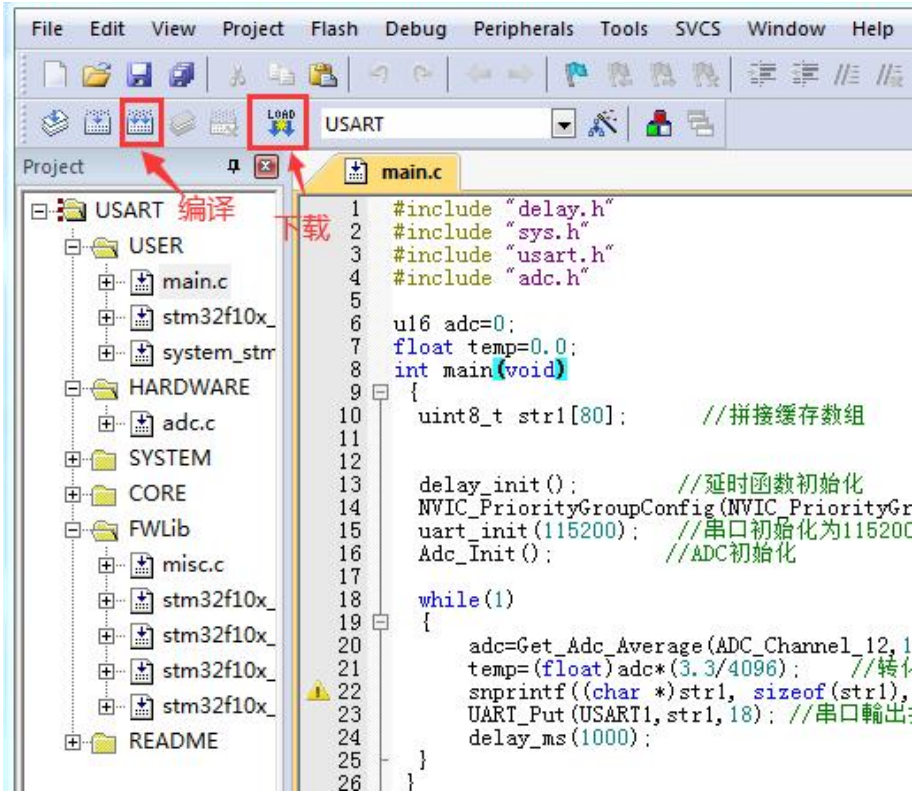
1.硬件连接



连接螺旋按钮和单片机的端口，插入 12V 电源，连接 JINK 仿真器，交叉串口线缆一端连接 CAN 测控终端的 DB9，另一端连接 PC 机。

2.程序下载

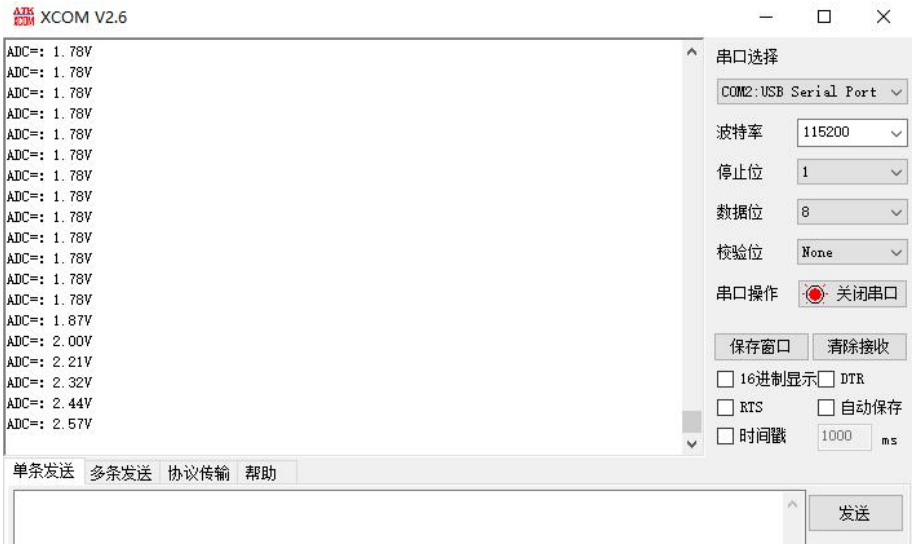
工程文件在 xxx\2.8 嵌入式的模拟信号数据采集实验\目录下。打开工程 USART.uvproj→编译程序→下载程序。如下图所示:



3.旋动螺旋按钮并观察串口调试助手打印信息。

2.7.6. 实验结果

在 PC 上打开串口调试工具，根据虚拟串口选择串口号，波特率为 115200，8 位数据位，1 位停止位，无校验位。配置好后，就可以正常接收电位器的电压值了，如图所示。



旋动螺旋按钮，可以观察电压在变化。嵌入式单片机模拟量数据采集完成。工控领域的多路采集控制模块内部就是通过类似这样的编程实现了模拟量信号的数据采集。

2.8. 基于嵌入式的开关量信号采集

2.8.1. 实验目的

理解模拟量信号的概念；

掌握开关量信号嵌入式单片机采集方法。

2.8.2. 实验环境

1. 硬件：微型滑台模块，如图 2-1 所示，程序下载调试板、方口 USB 线、交叉串口线、10 针排线、20 针排线、ARM JLINK 仿真器、USB 方口线及 PC 机。

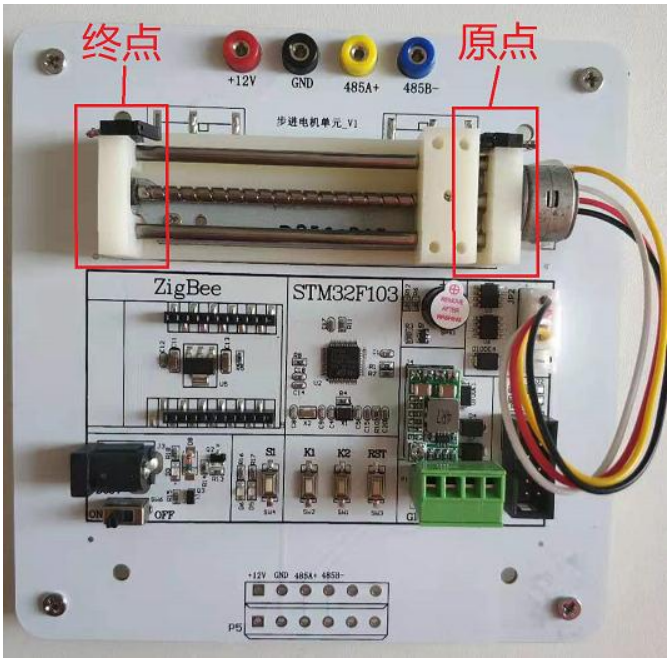


图 2-1 滑台模块示意图

2. 软件：Windows 7 及以上操作系统，Keil4 for ARM 4.7 开发环境。

2.8.3. 实验原理

1. 开关量信号

开关量是指非连续性信号的采集和输出，开关量信号的变化不是连续的，即跳跃变化，故又有脉冲信号的说法。它有 1 和 0 两种状态，这是数字电路中的开关性质，而电力上是指电路的开和关，或者说是触点的接通和断开。“开”和“关”是电器最基本、最典型的功能。相对于模拟信号它具有抗干扰能力强的特点，广泛应用于现代电子技术信号处理中。

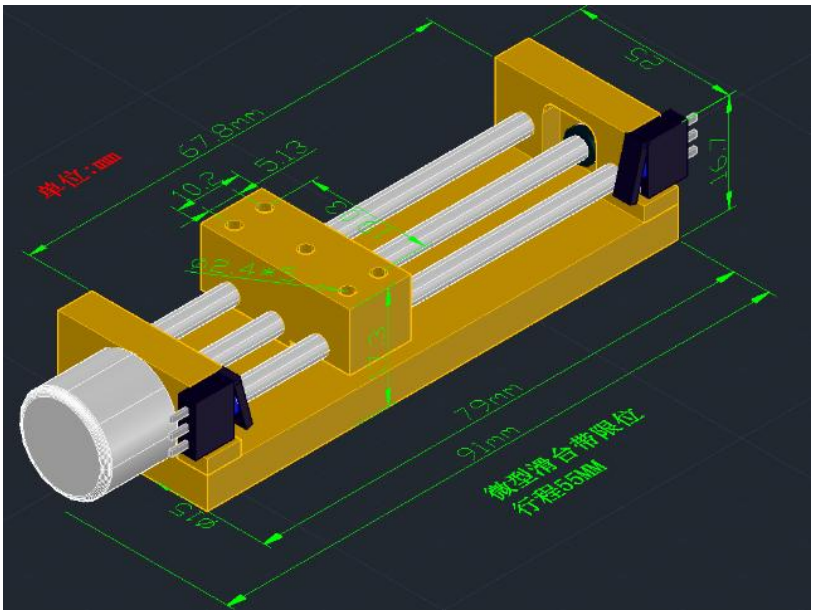
2. 步进电机

滑台的滑动是通过步进电机转动实现的，步进电机是一种将电磁脉冲转化为角位移的执行机构。当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角度（称为“步距角”），它的旋转是以固定的角度一步一步运行的。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

3. 限位开关检测电路

限位开关安装在滑台的两端，如图所示。利用滑块的机械运动部件的碰撞使

其触头动作来实现接通或断开控制电路，达到一定的控制目的。通常这类开关被用来限制机械运动的位置或行程，使运动机械按一定位置或行程自动停止、反向运动、或自动往返运动等。



限位开关检测电路原理如图 2-1 所示，两个限位开关分别连接在 STM32 的引脚 PA4 和 PA5 上，这里 P2 是终点限位开关，P4 是原点限位开关。所以当滑台的目标位置值大于当前位置值，也就是滑台的移动方向是原点到终点，就需要检测终点的限位开关的信号，即 PA4 引脚的电平；反之就需要检测原点限位开关的信号，即 PA5 引脚的电平。

通过原理图可得：正常情况下 PA4 和 PA5 引脚都是高电平，当滑台到达终点时接触到终点限位开关，PA4 会变为低电平；当滑台到达原点时接触到原点限位开关，PA5 会变为低电平。程序中就可以通过监测引脚 PA4 和 PA5 的高低电平信号来判断滑台是否到达终点位置或原点位置。

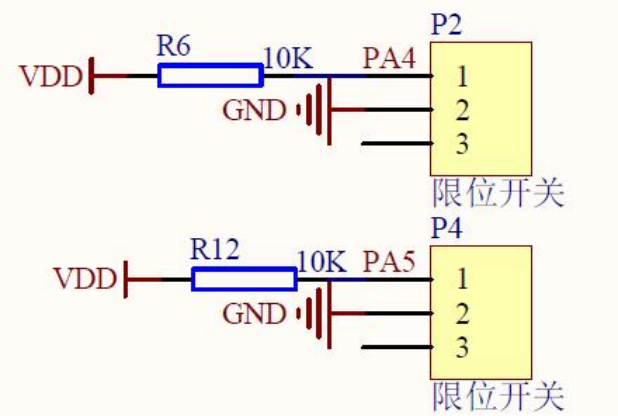


图 2-2 电路原理图

2.8.4. 实验内容

通过对限位开关的数据采集实现对滑台滑动范围的限制，分别通过按键 SW1 和 SW2 来控制滑台从终点到原点和从原点到终点的滑动，具体代码如下。

1. 主函数

在主函数中首先完成初始化，在步进电机的初始化中包含了对限位开关的初始化，初始化完成后将当前位置设置为 100，目标位置设置为 0,滑台就会滑到位置 0，也就是原点位置。在主循环中监测按键，当按键 SW1 按下，设置目标位置为 0，滑台会从当前位置滑到原点位置；当按键 SW2 按下，设置目标位置为 100，滑台会从当前位置滑到终点位置。

```
int main(void)
{
    KEYVALUE keyValue = KEY_UNKNOW;

    delay_init(72);                         //初始化延时函数
    StepperMotorInit();                     //步进电机配置
    General_TIM2_Init();                     //初始化定时器
    KeyInit();

    locationData.currentLocation = 100;      //假设当前不在原点     100 60mm
    locationData.targetLocation = 0;         //移动到原点

    while(1)
    {
        keyValue = KeyScan();

        if(keyValue == KEY_SW1)
        {
            locationData.targetLocation = 0;                //移动到原点
        }
        else if(keyValue == KEY_SW2)
```

```
        {
            locationData.targetLocation = 100;              //移动到终点
        }
        else
        {
            //do nothing
        }
    }
}
```

2. 定时器 2 中断函数

在定时器 2 的中断函数中定时去查看滑台的目标位置和当前位置是否相等，如果不相等就启动步进电机转动将滑台从当前位置滑到目标位置，同时使用限位开关限制滑台的滑动范围。如下程序中 ENDLIMIT 和 STARTLIMIT 就是终点限位开关和起点限位开关的状态。所以当 ENDLIMIT 或 STARTLIMIT 为低电平时，说明滑台已经滑到最大可滑动的位置，将停止步进电机转动。

```
  void TIM2_IRQHandler(void)
  {
    if(TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
    {
        if(locationData.targetLocation == locationData.currentLocation)       //已经在目标位置
        {
            A_OFF;B_OFF;C_OFF;D_OFF;
        }
        if((locationData.targetLocation) > (locationData.currentLocation)) //向终点移动
        {
            if((!ENDLIMIT))      //限位器触发,已经到达终点
            {
                locationData.targetLocation = 100;
                locationData.currentLocation = 100;   //当前位置赋值到要移到的位置
            }
            else
            {
                switch(steperVale)
                {
                    case 0:
                        A_OFF;B_OFF; C_ON;D_ON;
                        steperVale=1;
                    break;
                    case 1:
                        A_OFF;B_ON; C_ON;D_OFF;
                        steperVale=2;
                    break;
                    case 2:
                        A_ON;B_ON; C_OFF;D_OFF;
                        steperVale=3;
                    break;
                    case 3:
                        A_ON;B_OFF; C_OFF;D_ON;
```

```
                            steperVale=0;
                            locationData.currentLocation++;
                            if(locationData.currentLocation > 100) //最大目标位置不
大于 100
                            {
                                locationData.targetLocation                    =
locationData.currentLocation;       //已经到达最大位置
                            }
                            break;
                        default:
                            steperVale=0;
                        break;
                    }
                }
            }
            if(locationData.targetLocation < locationData.currentLocation)       //向原点
移动
            {
                if((!STARTLIMIT))     //限位器触发,已经到达起点
                {
                  locationData.targetLocation= 0;
                  locationData.currentLocation= 0;//当前位置赋值到要移到的位置
                }
                else
                {
                    switch(steperVale)
                    {
                        case 0:
                            A_OFF;B_OFF; C_ON;D_ON;
                            steperVale=1;
                        break;
                        case 1:
                            A_ON;B_OFF; C_OFF;D_ON;
                            steperVale=2;
                        break;
                        case 2:
                            A_ON;B_ON; C_OFF;D_OFF;
                            steperVale=3;
                        break;
                        case 3:
                            A_OFF;B_ON; C_ON;D_OFF;
                            steperVale=0;
                            locationData.currentLocation--;
                            if(locationData.currentLocation == 0)
                            {
                                locationData.targetLocation                    =
locationData.currentLocation;
                            }
                            break;
                        default:
                            steperVale=0;
                        break;
                    }
                }
            }
            TIM_ClearITPendingBit(TIM2, TIM_FLAG_Update);       //清中断
        }
```

```
}
```

2.8.5. 实验步骤

1. 打开“基于嵌入式的开关量信号数据采集”实验的代码，重新编译代码并将代码下载到滑台模块中。硬件连接如图 2-3 所示，方口 USB 线的另外一端连接电脑。

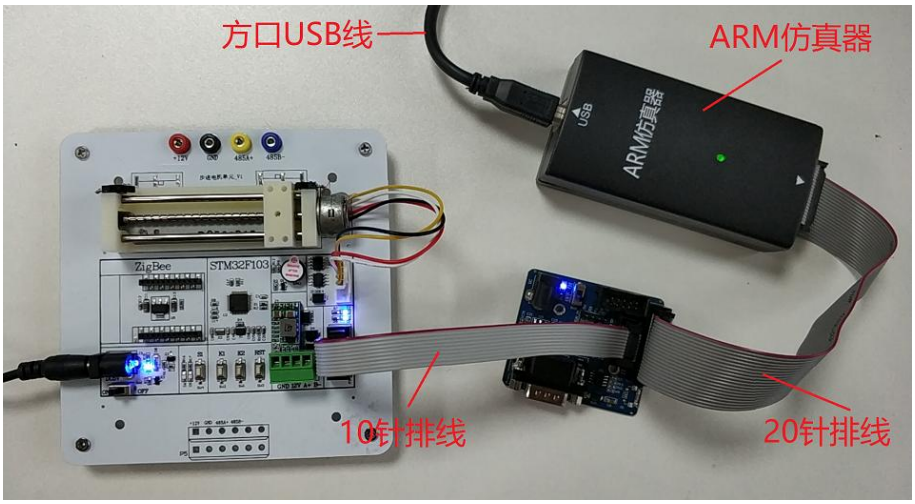


图 2-3 硬件连接图

2. 先后按下按键 SW2，SW1 来观察实验现象。

2.8.6. 实验结果

1. 代码下载完成后滑台会回到原点位置，如图 2-4 所示。

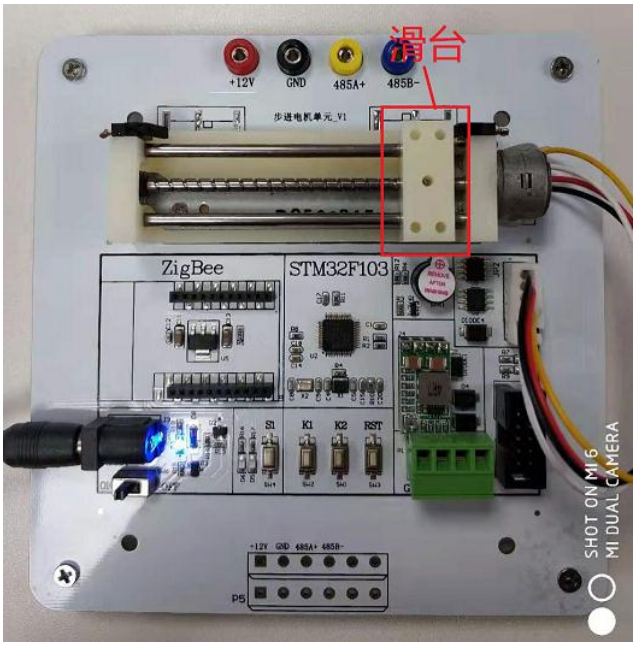


图 2-4 滑台在原点位置示意图

2. 按下按键 SW2，滑台滑到终点位置后停止滑动，如图 2-5 所示。

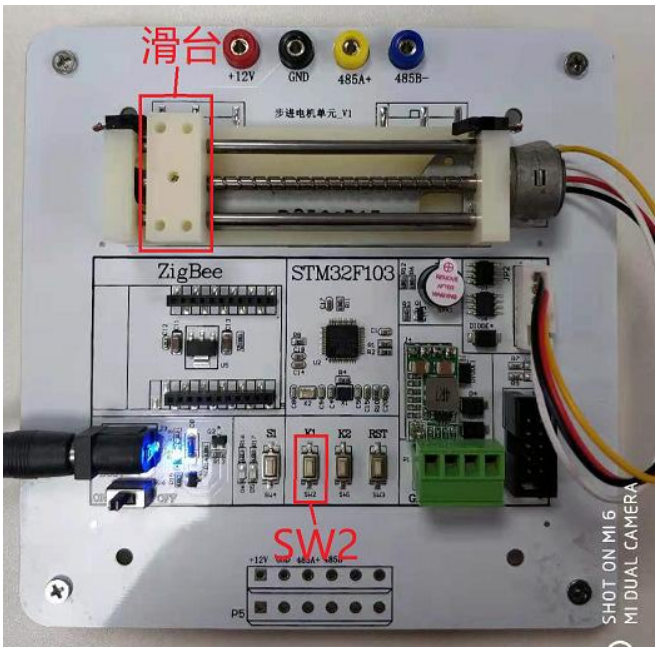


图 2-5 滑台到达终点示意图

3. 按下按键 SW1，滑台滑到原点位置后停止滑动，如图 2-6 所示。

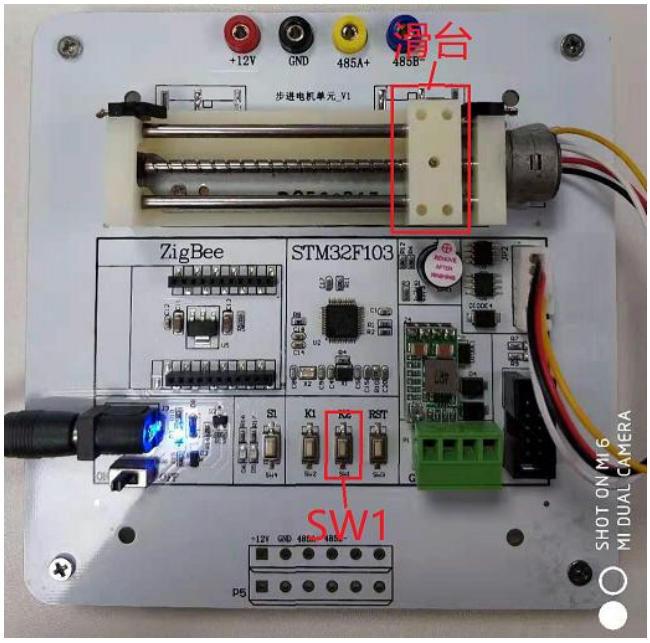


图 2-6 滑台原点示意图

嵌入式单片机开关量信号数据采集完成。工控领域的多路采集控制模块内部就是通过类似这样的编程实现了开关量信号的数据采集。

2.9. 基于嵌入式的继电器控制

2.9.1. 实验目的

理解继电器工作原理和驱动电路设计方法；

掌握继电器控制的嵌入式单片机编程方法。

2.9.2. 实验环境

1. 硬件：远程测控终端模块、程序下载调试板、10 针排线、20 针排线、ARM JLINK 仿真器、USB 方口线、DC 12V 电源以及 PC 机。

2. 软件：Windows 7 及以上操作系统，Keil4 for ARM 4.7 开发环境。

图 2-7 远程测控终端模块示意图

2.9.3. 实验原理

1. 继电器

在本实验中涉及到的继电器模块是电磁继电器，电磁继电器一般由铁芯、线圈、衔铁、触点簧片等组成。只要在线圈两端加上一定的电压，线圈中就会流过一定的电流，从而产生电磁效应，衔铁就会在电磁力吸引的作用下克服返回弹簧的拉力吸向铁芯，从而带动衔铁的动触点与静触点（常开触点）吸合。当线圈断电后，电磁的吸力也随之消失，衔铁就会在弹簧的反作用力返回原来的位置，使动触点与原来的静触点（常闭触点）释放。这样吸合、释放，从而达到了在电路中的导通、切断的目的。对于继电器的“常开”、“常闭”触点，可以这样来区分：继电器线圈未通电时处于断开状态的静触点，称为“常开触点”；处于接通状态的静触点称为“常闭触点”。

2. 电路原理

继电器原理图如图 2-8 所示，由原理图可以看到，Relay 连接到 STM32 的 PB0 引脚。继电器 2 脚有供电，当 PB0 为高电平时，三极管导通，继电器 5 脚为低电平，于是电流流过线圈，继电器吸合，1、4 常闭端断开，1、3 常开端闭合；反之，当 PB0 为低电平时，三极管截止，继电器线圈没有电流流过，会保持 1、4 端常闭，1、3 端常开。

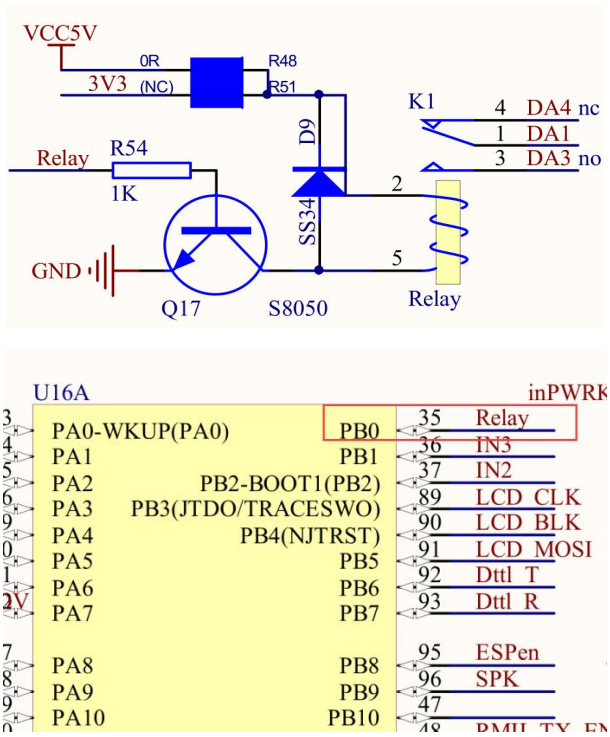


图 2-8 继电器原理图

2.9.4. 实验内容

本实验的实验内容是按下远程测控终端的上按键（UP），继电器打开（吸合），按下远程测控终端的下按键（DN），继电器关闭（断电）。

主函数的内容如下所示：

```
int main(void)
{
    u8 key=0;
    delay_init(168);     //时钟初始化
  RELAY_Init();         //继电器以及按键初始化

    while(1)
    {

     key= KEY_Scan(0);     //按键扫描，判断哪个按键按下

        switch(key)
        {
            case 1: RELAY=1;    break;            //开

            case 2: RELAY=0;    break;            //关

            default:    break;
        }
    }
}
```

主函数中首先初始化继电器和按键，然后在主循环判断哪个按键按下，如果按下上按键，打开继电器，反之，按下下按键，关闭继电器。RELAY 的定义如下所示：

```
#define RELAY PBout(0)
```

这里利用的是 STM32 的位带操作，简单理解就是 RELAY 代表的就是 GPIO PB0，这是继电器对应的 IO 口，RELAY = 1 就是打开继电器，RELAY = 0 就是关闭继电器。

2.9.5. 实验步骤

1. 首先进行硬件连接，使用 USB 方口线连接 ARM JLINK 仿真器和 PC，使用 20 针排线连接仿真器和下载调试板，使用 10 针排线连接下载调试板和远程测控终端，最后 使用 DC 12V 电源给远程测控终端供电。硬件连接如图 2- 所示。

图 2- 9 硬件连接图

2. 打开本实验的代码：“按键控制继电器开关\USER”路径下的工程文件，重新编译代码并将代码下载到远程测控终端模块中，编译和下载按钮如图 2-10 所示。

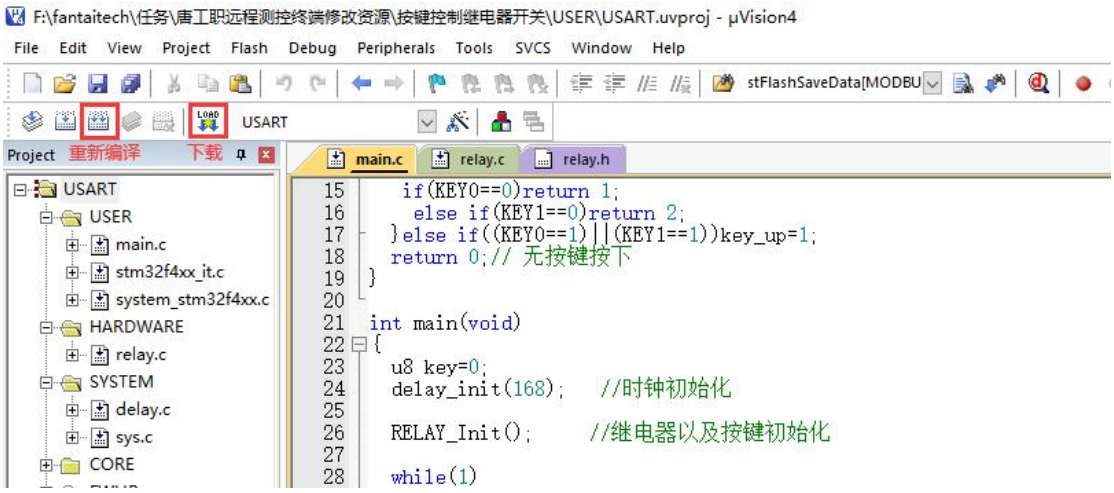


图 2-10 编译下载程序

3. 从远程测控终端上拔下仿真器，然后给远程测控终端重新上电，让程序重新运行。

2.9.6. 实验结果

按下远程测控终端的上按键，可以听到“咔嗒”一声，此时继电器被打开，然后按下下按键，也能听到继电器动作的声音，此时继电器关闭。远程测控终端的上按键和下按键如下图所示。

图 2-11 远程测控终端上按键和下按键的位置

3. 数据通信与工业网络

3.1. 工业测控系统的数据通信

数据是指对数字、字母及组合意义的一种表达。在工业测控系统中，通信数据与监控系统的各种信息紧密相关，如用数字 1 表示电机处于工作状态，用数字 0 表示电机处于停止状态；而对于温度、压力、液位、电流、电压等变量可以用一定数值范围的数字来描述。

数据通信是指建立在特定通讯协议的基础上，利用数据编码、传输、转换、存储、处理等技术，实现两地通信终端之间传递数据信息。它是计算机技术与通信技术相结合的产物。根据传输媒体不同，有有线数据通信和无线数据通信之分。

数据通信系统就是以计算机为中心，通过数据传输信道将分布在各处的数据终端设备连接起来，以实现数据通信目的的系统。实际的数据通信系统千差万别，如可以是两台计算机点对点近距离数据传输，可以是工业现场智能设备与控制器之间的数据通信，也可以是分布在各地的数百台计算机互相传输的数据。

3.1.1. 典型的数据通信场景

在工业数据监控系统中，通常包含以下几种数据通信过程。

1. 现场测控站点仪表、执行机构与下位机的通信

现场仪表、执行机构、及各种类型的测控设备与下位机的通信多采用平行线，即采用硬接线方式把每个测控点连接到控制系统的 I/O 设备上。这种点对点的布线方式，在现场总线技术出现后显得落后。目前，多数系统现场测控站多采用现场总线和平行接线混合的方式。下位机系统配置现场总线接口，在测控点相对集中的设备附近设置现场 I/O 站，现场 I/O 站与下位机系统采用现场总线通信。在不方便布线的地方，也会采用短距离无线通信技术。

2.下位机系统与监控服务器（上位机）的远程通信

下位机系统数量多，结构型号多样，与上位机的物理距离较大，因此与上位机的通信形式多样，从通信介质看，既有有线通信、也有无线通信，其中以无线

通信为主，有线通信为辅。无线通信种，包含数传电台、微波、GPRS、卫星，还有现在发展起来的窄带广域网技术，例如 LoRa、NB-IoT。

3.2. 通用串行通信

3.2.1. 几种主要串行通信技术

串行通信是指使用一条数据线，将数据一位一位地依次传输，每一位数据占据一个固定的时间长度。其只需要少数几条线就可以在系统间交换信息，特别适用于计算机与计算机、计算机与外设之间的远距离通信。

RS-232-C 是美国电子工业协会 EIA（Electronic Industry Association）制定的一种串行物理接口标准。RS 是英文“推荐标准”的缩写，232 为标识号，C 表示修改次数。RS-232-C 总线标准设有 25 条信号线，包括一个主通道和一个辅助通道。在多数情况下主要使用主通道，对于一般双工通信，仅需几条信号线就可实现，如一条发送线、一条接收线及一条地线。

EIA-RS-232C 对电气特性、逻辑电平和各种信号线功能都作了规定。

在 TxD 和 RxD 上：

逻辑 1(MARK)=-3V～-15V

逻辑 0(SPACE)=+3～+15V

在 RTS、CTS、DSR、DTR 和 DCD 等控制线上：

信号有效（接通，ON 状态，正电压）=+3V～+15V

信号无效（断开，OFF 状态，负电压）=-3V～-15V

以上规定说明了 RS-232C 标准对逻辑电平的定义。对于数据（信息码）：逻辑“1”（传号）的电平低于-3V，逻辑“0”（空号）的电平高于+3V；对于控制信号；接通状态（ON）即信号有效的电平高于+3V，断开状态（OFF）即信号无效的电平低于-3V，也就是当传输电平的绝对值大于 3V 时，电路可以有效地检查出来，介于-3～+3V 之间的电压无意义，低于-15V 或高于+15V 的电压也认为无意义，因此，实际工作时，应保证电平在-3V～-15V 或+3V～+15V 之间。

RS-422 标准全称是“平衡电压数字接口电路的电气特性”，它定义了接口电

路的特性。实际上还有一根信号地线，共 5 根线。由于接收器采用高输入阻抗和发送驱动器比 RS232 更强的驱动能力，故允许在相同传输线上连接多个接收节点，最多可接 10 个节点。即一个主设备（Master），其余为从设备（Slave），从设备之间不能通信，所以 RS-422 支持点对多的双向通信。接收器输入阻抗为 4k，故发端最大负载能力是 10×4k+100Ω（终接电阻）。

RS-485 总线采用平衡发送和差分接收，因此具有抑制共模干扰的能力。加上总线收发器具有高灵敏度，能检测低至 200mV 的电压，故传输信号能在千米以外得到恢复。市场上一般 RS-485 采用半双工工作方式，任何时候只能有一点处于发送状态，因此，发送电路须由使能信号加以控制。RS-485 用于多点互连时非常方便，可以省掉许多信号线。应用 RS-485 可以联网构成分布式系统，其允许最多并联 32 台驱动器和 32 台接收器。

RS232,RS422,RS485 是电气标准，主要区别就是逻辑如何表示。

RS232 使用 12V,0,-12V 电压来表示逻辑，（-12V 表示逻辑 1，12V 表示逻辑 0），全双工，最少 3 条通信线（RX,TX,GND），因为使用绝对电压表示逻辑，由于干扰，导线电阻等原因，通讯距离不远，低速时几十米也是可以的。

RS422，在 RS232 后推出，使用 TTL 差动电平表示逻辑，就是两根的电压差表示逻辑，RS422 定义为全双工的，所以最少要 4 根通信线（一般额外地多一根地线），一个驱动器可以驱动最多 10 个接收器（即接收器为 1/10 单位负载），通讯距离与通讯速率有关系，一般距离短时可以使用高速率进行通信，速率低时可以进行较远距离通信，一般可达数百上千米。

RS485，在 RS422 后推出，绝大部分继承了 422，主要的差别是 RS485 可以是半双工的，而且一个驱动器的驱动能力至少可以驱动 32 个接收器（即接收器为 1/32 单位负载），当使用阻抗更高的接收器时可以驱动更多的接收器。所以现在大多数全双工 485 驱动/接收器对都是标：RS422/485 的，因为全双工 RS485 的驱动/接收器对一定可以用在 RS422 网络。

3.2.2. RS485 的网络

RS485 接口组成的半双工网络，一般是两线制（以前有四线制接法，只能实现点对点的通信方式，现很少采用），多采用屏蔽双绞线传输。这种接线方式为

总线式拓扑结构在同一总线上最多可以挂接 32 个结点。在 RS485 通信网络中一般采用的是主从通信方式，即一个主机带多个从机。很多情况下，连接 RS-485 通信链路时只是简单地用一对双绞线将各个接口的“A”、“B”端连接起来。

网络拓扑一般采用终端匹配的总线型结构，不支持环形或星形网络。在构建网络时，应注意如下几点：

（1）采用一条双绞线电缆作总线，将各个节点串接起来，从总线到每个节点的引出线长度应尽量短，以便使引出线中的反射信号对总线信号的影响最低。有些网络连接尽管不正确，在短距离、低速率仍可能正常工作，但随着通信距离的延长或通信速率的提高，其不良影响会越来越严重，主要原因是信号在各支路末端反射后与原信号叠加，会造成信号质量下降。

（2）应注意总线特性阻抗的连续性，在阻抗不连续点就会发生信号的反射。

在低速、短距离、无干扰的场合可以采用普通的双绞线，反之，在高速、长线传输时，则必须采用阻抗匹配（一般为 120Ω）的 RS485 专用电缆（STP-120Ω（for RS485 & CAN） one pair 18 AWG），而在干扰恶劣的环境下还应采用铠装型双绞屏蔽电缆（ASTP-120Ω（for RS485 & CAN） one pair 18 AWG）。在使用 RS485 接口时，对于特定的传输线路，从 RS485 接口到负载其数据信号传输所允许的最大电缆长度与信号传输的波特率成反比，这个长度数据主要是受信号失真及噪声等影响所影响。理论上，通信速率在 100Kbps 及以下时，RS485 的最长传输距离可达 1200 米，但在实际应用中传输的距离也因芯片及电缆的传输特性而所差异。在传输过程中可以采用增加中继的方法对信号进行放大，最多可以加八个中继，也就是说理论上 RS485 的最大传输距离可以达到 10.8 公里。如果真需要长距离传输，可以采用光纤为传播介质，收发两端各加一个光电转换器，多模光纤的传输距离是 5~10 公里，而采用单模光纤可达 50 公里的传播距离。

3.2.3. RS485 传输电路设计

485 是一个平衡（差分）数字传输线接口，是为了改善 TIA/EIA-232（以下简称 232）的局限性而开发出来的。485 具有以下特性：

通讯速率高 – 可达到 50M bits/s

通讯距离远 – 可达到 1200 米（注：100Kbps 情况下）

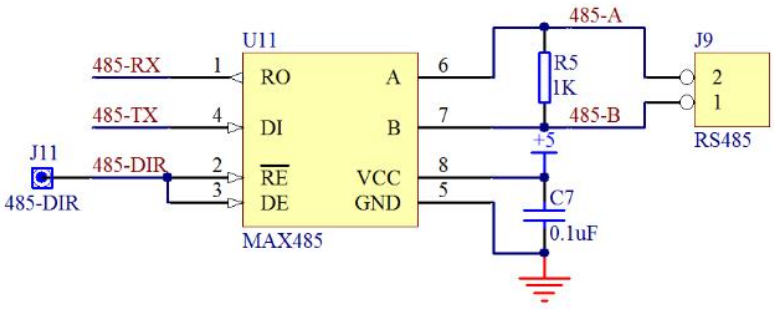
差分传输 – 较小的噪声辐射

多驱动器和接收器

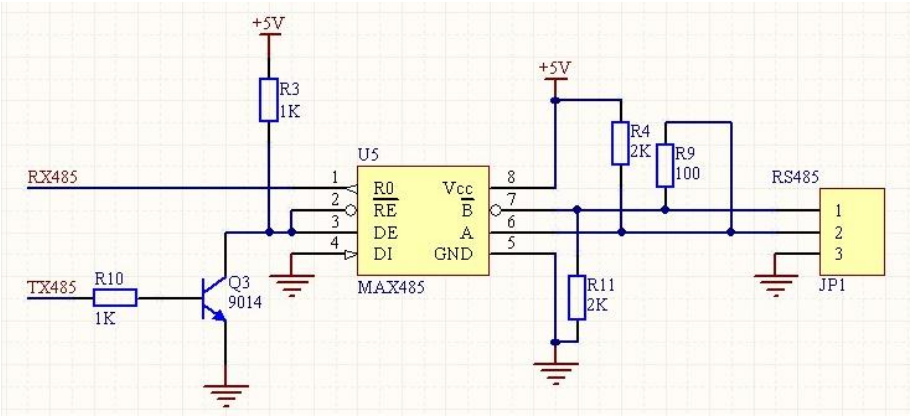
RS485 的接口非常简单，与 RS232 所使用的 MAX232 是类似的，485 控制线只需要一个 RS485 转换器，就可以直接与单片机的 UART 串口连接起来，并且使用完全相同的异步串行通信协议。但是由于 RS485 是差分通信，因此接收数据和发送数据是不能同时进行的，也就是说它是一种半双工通信。那我们如何判断什么时候发送，什么时候接收呢？

RS485 转换芯片很多，电路设计也有所区别。

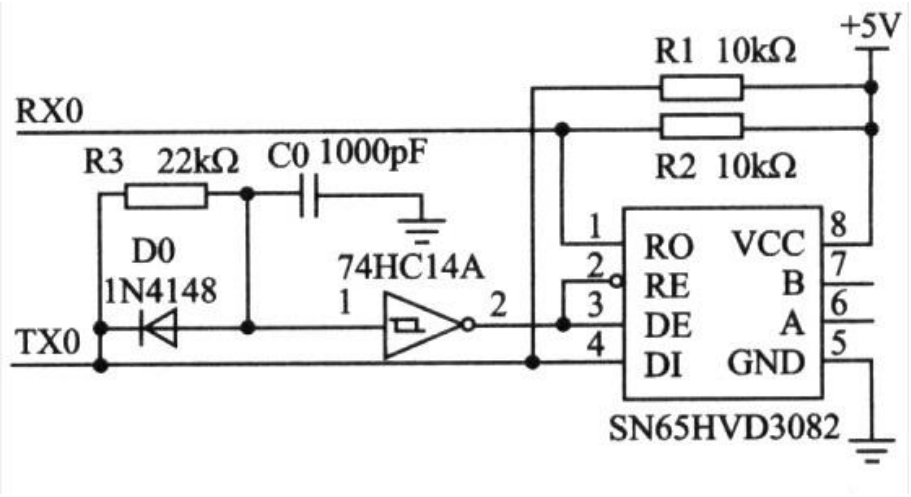
常规三线的 RS485 电路:



两线的 RS485 电路:



两线的 RS485 电路:



两线的 RS485 电路，如果发 1,485 芯片被设置为接受状态，485 总线空闲，主机收到的状态就是 1。

3.3. Modbus 通信协议

3.3.1. Modbus 协议概述

1.Modbus 协议

Modbus 协议是 Modicon 公司（现已经为施耐德公司并购，成为其旗下的子品牌）于 1979 年开发的通信协议，最初的目的是实现可编程控制器之间的通信。通过此协议，控制器之间、控制器通过网络（如以太网）和其他设备之间可以实现串行通信。该协议已经成为通用工业标准。采用 Modbus 协议，不同厂商生产的控制设备可以互连成工业网络，实现集中控制和管理。

Modbus 协议，定义了一个控制器能够识别使用的消息结构，而不管他们是经过何种网络进行通信的。它描述了控制器请求访问其他设备的过程，如何响应来自其他设备的请求，以及怎样侦测错误并记录。它指定了信息帧的格式。

通过 Modbus 协议在网络上通信时，必须清楚每个控制器的设备地址，识别按地址发来的消息，决定产生何种动作。如果需要响应，控制器将生成反馈信息并用 Modbus 协议发出。另外在其他网络上使用 Modbus 协议，包含协议转换为此网络上使用的帧或包结构。

2.Modbus 网络上传输

标准的 Modbus 接口使用 RS-232 串口接口，它定义了连接口的针脚、电缆、信号位、传输波特率、奇偶校验。控制器能直接或间接（通过 Modem 拨号）组网。控制器通信使用主—从技术，即只有一个设备（主设备）能初始化传输（查询）。其他设备（从设备）根据主设备查询提供的数据做出相应反应。典型的主设备有计算机主机和可编程仪表。典型的从设备有可编程控制器、传感器、各种仪表等。

Modbus 协议建立了主设备查询的格式：设备地址、功能码、要发送的数据和错误检测域。从设备回应的消息由 Modbus 协议构成，包括确认要行动的域、要返回的数据和错误检测域。如果在消息接收过程中发生错误，或从设备不能执行其命令，从设备将建立错误消息并把它作为回应发送出去。

3.3.2. Modbus 的 RTU 消息帧

Modbus 通信协议具有多个变种，其具有支持串口（主要是 RS-485 总线），以太网多个版本，其中最著名的是 Modbus RTU，Modbus ASCII 和 Modbus TCP 三种。其中 Modbus RTU 与 Modbus ASCII 均为支持 RS-485 总线的通信协议，其中 Modbus RTU 由于其采用二进制表现形式以及紧凑数据结构，通信效率较高，应用比较广泛。而 Modbus ASCII 由于采用 ASCII 码传输，并且利用特殊字符作为其字节的开始与结束标识，其传输效率要远远低于 Modbus RTU 协议，一般只有在通信数据量较小的情况下才考虑使用 Modbus ASCII 通信协议，在工业现场一般都是采用 Modbus RTU 协议。一般而言，大家说的基于串口通信的 Modbus 通信协议都是指 Modbus RTU 通信协议。而 Modbus TCP 协议则是在 RTU 协议上加一个 MBAP 报文头，由于 TCP 是基于可靠连接的服务，RTU 协议中的 CRC 校验码就不再需要，所以在 Modbus TCP 协议中是没有 CRC 校验码。

本小节主要介绍 Modbus 协议的 RTU 消息帧，有关 ASCII 消息帧请查阅其他相关专业书籍。

1. Modbus RTU 报文基本格式

ADU： 应用数据单元

PDU： 协议数据单元

详细的协议格式如下表所示。

起始符	地址码	功能码	寄存器起始地址	寄存器长度	CRC 低 8 位校验	CRC 高 8 位校验	结束符
			数据				
起始应用不小于 3.5 个字符的报文间隔	1 字节	1 字节	2 字节	2 字节	1 字节	1 字节	结束时不小于 4 字节的报文间隔（时间）
			N 字节				

1）起始符与结束符

数据报前后都有至少 3.5 个字节的时间间隔，起始位和结束符实际上没有任何数据，时间间隔是为了防止总线上多组不同数据“黏连”而规定的，真正有意义的第一个字节是设备地址。

那么，3.5 个字节时间到底多久呢？

这跟设置的总线传输波特率有关。例如，波特率是 9600bps，9600bit per second，即 1s 传输 9600 位，9600/8 个字节。则换算成，传输 1 个字节多少秒？即 8/9600s。那么，modbus 传输前后间隔，按照 4 个字节来算，就是要预留 4*8/9600=3ms 的间隔。

2）地址码

每个设备都有一个自己的地址，当设备接收到一帧数据后，程序首先对设备地址字节进行判断比较，如果与自己的地址不同，则对这帧数据直接不予理会，如果与自己的地址相同，就要对这帧数据进行解析，按照之后的功能码执行相应的功能。如果地址是 0x00，则认为是一个广播命令，就是所有的从机设备都要执行的指令。

3）功能码

在第二个字节功能代码字节中，Modbus 规定了部分功能代码，此外也保留了一部分功能代码作为备用或者用户自定义，这些功能码大家不需要去记忆，甚

至都不用去看，直到你用到的那天再过来查表格即可。举个例子，如果功能码是0x03，也就是读保持寄存器。如果功能码是 0x06，就是写寄存器。

4）数据字段

跟在功能代码后边的是 n 个 8bit 的数据字段，数据字段由寄存器起始地址和寄存器数量组成。

这个 n 值的到底是多少，是功能代码来确定的。不同的功能代码后边跟的数据数量不同。举个例子，如果功能码是 0x03，也就是读保持寄存器，那么主机发送数据 n 的组成部分就是：2 个字节的寄存器起始地址，加 2 个字节的寄存器数量 N。从机数据 n 的组成部分是：1 个字节的字节数，因为我们回复的寄存器的值是 2 个字节，所以这个字节数也就是 2N 个，再加上 2N 个寄存器的值。

5）CRC 校验

CRC 校验函数把一帧数据除最后两个字节外，前边所有的字节进行特定的算法计算，计算完后生成了一个 16bit 的数据，作为 CRC 校验码，添加在一帧数据的最后。接收方接收到数据后，同样会把前边的字节进行 CRC 计算，计算完了再和发过来的 16bit 的 CRC 数据进行比较，如果相同则认为数据正常，没有出错，如果比较不相同，则说明数据在传输中发生了错误，这帧数据将被丢弃，就像没收到一样，而发送方会在得不到回应后做相应的处理错误处理。

3.3.3. Modbus-RTU 通信格式

主机要想获得传感器的采样数值，首先要知道该传感器的设备地址，然后组成 modbus 帧，发送到 485 总线上。传感器收到后，经过匹配，地址相同，则通过总线向主机返回应答帧。通俗的，modbus rtu 的报文格式转变为：

（1）主机问询帧结构：

地址码	功能码	寄存器起始地址	寄存器长度	校验码低位	校验码高位
1 字节	1 字节	2 字节	2 字节	1 字节	1 字节

（2）从机应答帧结构：

地址码	功能码	有效字节数	数据一区	第 2 数据区	第 N 数据区	校验码
1 字节	1 字节	2 字节	2 字节	2 字节	2	1 字节

举例：

读取功能码 03：读保持寄存器（模拟量输出，在一个或多个保持寄存器中取得当前的二进制）

请求格式：

地址码	功能码	起始地址高字节	起始地址低字节	读取个数高字节	读取个数低字节	CRC 校验码
1 字节	03	1 字节	1 字节	1 字节	1 字节	2 字节

正确应答格式：

地址码	功能码	返回数据字节数	寄存器 1 高字节	寄存器 1 低字节	...	CRC 校验码
1 字节	03	1 字节	1 字节	1 字节	...	2 字节

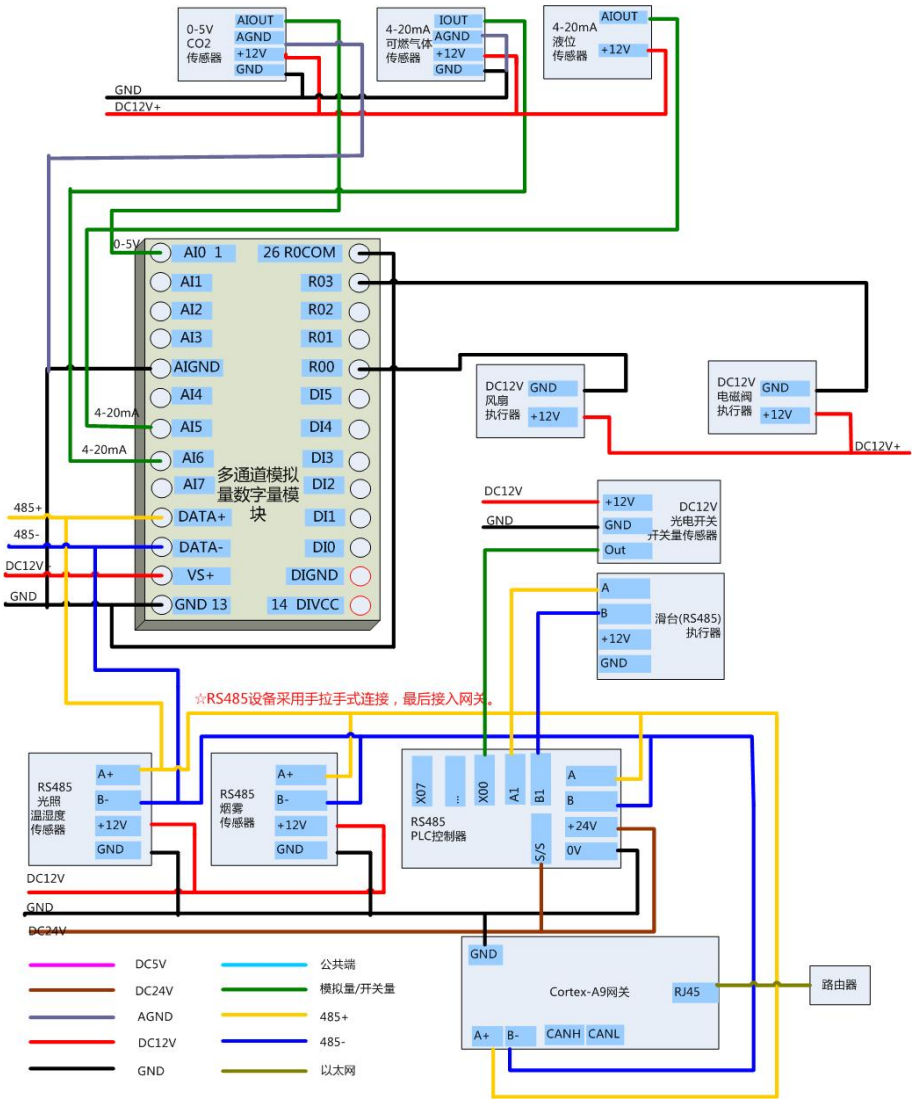
注意：在返回的信息中一个模拟量需要返回 2 个字节。

3.4. 工业现场 RS485 总线网络

3.4.1. RS485 总线设备接线图

该实训台所有 RS485 工业级传感器、采集器，都通过手拉手方式连接，接到网关的 485 总线接口。此时网关作为主机，与 RS485 传感器进行通信，采用的是 Modbus-RTU 协议。也可以将 485 总线接到 PC 电脑上，使用上位机工具测试。

出厂时，默认所有传感器与网关连接。RS485 网络接线图，如图所示。



3.4.2. RS485 总线设备地址表

该实训台使用的 RS485 设备，出厂时，Modbus 的地址码已经配置好，参考下表。

名称	连接口	ModbusRTU 报文					量程
		地址码	功能码	起始地址	寄存器长度	CRC 校验低 8b+高 8b	
		1B	1B	2B	2B	2B	
多路采集控制模块	485	01				/	AIGND 与 GND 共地
二氧化碳	AI0 0-5V	01	04	00 00	00 01	/	400-5000 ppm

液位传感器	AI5 4-20ma	01	04	00 05	00 01	/	0-1m/3m
甲烷可燃气体	AI6 4-20mA	01	04	00 06	00 01	/	0-100%
风扇 12V	R00	01	05	00 00	FF 00(闭合) 0000(断开)	/	开/关
电磁阀 12v	R03	01	05	00 03	FF 00(闭合) 0000(断开)	/	开/关
步进滑台	485	02	06	00 03	00 0A	/	行程
光照温湿度传感器	485	04	03	00 06	00 01	/	0-65535Lux
烟雾探测器	485	08	03	00 03	00 01	/	0-100%
电能表电量	485	0C	03	00 00	00 02	/	
电能表电压		0C	03	00 64	00 02	/	
电能表电流		0C	03	00 6A	00 02	/	
PLC 控制器	485	0D	06	00 03	00 05	控制滑台加工次数	示例加工 5 次

3.4.3. 光照温湿度通信测试

1.传感器介绍

可以采集光照、温度、湿度三种环境参数。采用高灵敏度的感光探头，信号稳定，精度高。具有测量范围宽、线形度好、防水性能好、使用方便、便于安装、传输距离远等特点。广泛适用于农业大棚、花卉培养等需要光照度及温湿度监测的场合。 主要技术指标，如表所示。

直流供电（默认）	10-30VDC，典型值12VDC	
最大功耗	RS485输出	0.4W
精度	光照强度	±7%(25℃)
	温度	±0.5℃（25℃）
	湿度	±3%RH（5%RH`95%RH，25℃）

光照强度量程	0~65535lux；0~20 万Lux
温湿度量程	-20℃~+60℃，0%RH~80%RH
工作温度	-20℃~+60℃，0%RH~80%RH
长期稳定性	光照≤5%/y,温度≤0.1℃/y，湿度≤1%y.
响应时间	光照0.1s，温度≤18s(1m/s风速)，湿度≤6s(1m/s风速)
输出信号	RS485输出，信号：四根线，电源正、负、485正极、485负极，接入磁吸底板。ModbusRTU协议

2.硬件连接

名称

光照温湿度传感器

连接说明

连接说明：

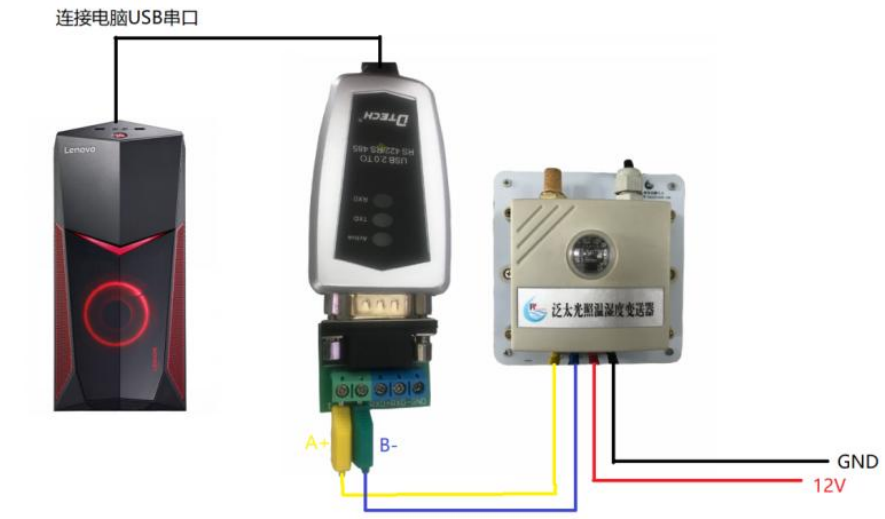
①电源正：红色香蕉头线，一端接入台体两侧的电源接口的 12V 处，另一端接入传感器磁吸底板的+12V 处。

②电源负：黑色香蕉头线，一端接入台体两侧的电源接口的 GND 处，另一端接入传感器磁吸底板的 GND 处。

③485 正极：黄色香蕉头线，一端接入多通道采集模块的 DATA+接口处，另一端接入传感器磁吸底板的 V1/I1/485A+接口处。

④485 负极：蓝色香蕉头线，一端接入多通道采集模块的 DATA-接口处，另一端接入传感器磁吸底板的 V2/I2/485B-接口处。

连接 PC 电脑示意图



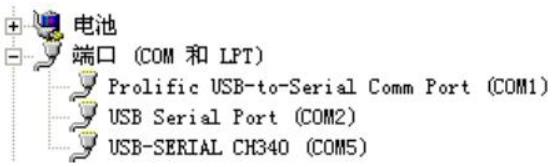
连接后，光照温湿度传感器磁吸底板电源指示灯被点亮，说明 12VDC 供电

正常。

3.参数配置

该配置软件仅针对“仁科测控”设备。在资料\软件工具\中找到并打开“485参数配置软件”。

1）选择正确的 COM 口，点击我的电脑—设备管理器找到对应的端口。



2）点击测试波特率按钮，软件弹出当前传感器出厂时的地址、波特率。空气温湿度传感器默认的设备地址为 1，波特率为 4800，如图所示

为防止 RS485 总线冲突，将光照温湿度传感器地址设置为 0x04，波特率设为 9600bps。 出厂时，这一步已经设置好，可忽略此步。

4.传感器协议命令

（1）只读光照

发送	485	04	03	00 06	00 01	/	0-65535Lux
应答		04	03	02	Xx xx	/	

（2）只读湿度：

发送	485	04	03	00 00	00 01	/	0-65535Lux
应答		04	03	02	Xx xx	/	

（3）只读温度：

发送	485	04	03	00 02	00 01	/	0-65535Lux
应答		04	03	02	Xx xx	/	

（3）读取温湿度：

发送	485	04	03	00 00	00 02	/	0-65535Lux
应答		04	03	04	Xx xx 湿度 Xx xx 温度	/	

（4）读取光照、湿度、温度：

发送	485	04	03	00 00	00 07	/	0-65535Lux
应答		04	03	0E	Xx xx 湿度 Xx xx 温度 00...00 填充 Xx xx 光照	/	

5.PC 电脑上 ModbusRTU 工具测试

可采用电脑的 ModbusRTU 工具发送命令进行测试，也可以直接使用网关应用软件测试。使用电脑测试时，485 总线通过 USB-RS485 转换器连接到电脑上。使用网关时，485 总线的 A、B 线缆对应连接到网关的 485 总线接口上。

通过电脑管理器得知 USB-485 转换器连接的串口号为 COM6，设置串口波特率为 9600，无校验，1 个停止位，勾选 16 进制，在“设备地址”处输入传感器的地址码（0x04），“功能码”输入 0x03，寄存器（起始）地址输入 0x00 0x00，寄存器数量输入 0x00 06，勾选自动组合，则软件会自动给报文尾部增加校验，可在“发送原始内容”框中看到组合后的命令帧。点击“发送”按钮，上位机软件就会向总线发送命令，总线上地址码为 0x04 的传感器接收到后，就会进行采样，并返回应答帧。由于光照温湿度传感器是三合一传感器，可以一次性读取三种环境参数。

返回的应答帧：

04 03 08 02 2A 00 AF 00 00 00 B4 5A 6F

从左往右依次为，04 为地址码，03 为功能码，08 为数据长度，022A 为湿度传感器值，00 AF 为温度采样值，00 00 为填充字段，00 B4 为光照度采样值。

02 2A = 554（十进制）= 55.4%RH

00 AF =175（十进制）= 17.5℃

00 B4 =180（十进制）=180 Lux（针对量程 0-65535Lux 的光照传感器）

6. Android 网关通信测试

默认光照温湿度传感器接入实训台的 RS485 总线上，连接到网关。用网关的“工业数据采集应用程序 APP”进行通信测试，可以看到网关采样的光照、温度、湿度，如图所示。

3.4.4. 烟雾探测器通信测试

1.传感器介绍

实时探测烟雾，自身带有蜂鸣器。报警后可发出强烈声响。

主要技术指标如表所示。

直流供电（默认）	10-30VDC，典型值12VDC	
功耗	静态功耗	0.12W
	报警功耗	0.7W
灵敏度	烟雾灵敏度	1.06±0.26%F T
工作温度	-10℃~+50℃，≤95%，无凝露	
输出信号	RS485输出，信号：四根线，电源正、负、485正极、485负极，接入磁吸底板。ModbusRTU协议	

2.硬件连接

名称	连接说明
烟雾传感器	连接说明： ①电源正：红色香蕉头线，一端接入台体两侧的电源接口的 12V 处，另一端接入传感器磁吸底板的+12V 处。 ②电源负：黑色香蕉头线，一端接入台体两侧的电源接口的 GND 处，另一端接入传感器磁吸底板的 GND 处。 ③485 正极：黄色香蕉头线，一端接入 RS485 总线黄色上，另一端接入传感器磁吸底板的 485A+接口处。 ④485 负极：蓝色香蕉头线，一端接入 RS485

总线蓝色线上，另一端接入传感器磁吸底板的 485B-接口处。

连接后，烟雾传感器磁吸底板如果有电源指示灯的话，会被点亮，说明 12VDC 供电正常。

3.参数配置

在资料\软件工具\中找到并打开“485 参数配置软件”（该配置软件仅针对“仁科测控”设备）。

1）选择正确的 COM 口，点击我的电脑—设备管理器找到对应的端口。

2）点击测试波特率按钮，软件弹出当前传感器出厂时的地址、波特率。烟雾传感器默认的设备地址为 1，波特率为 4800。为防止 RS485 总线冲突，将烟雾传感器地址设置为 0x08，波特率设为 9600bps。 出厂时，这一步已经设置好，可忽略此步。

4.传感器协议命令

（1）读取烟雾状态：

发送	485	08	03	00 03	00 01	/	
应答		08	03	02	Xx xx	/	00 00 正常 00 01 报警

5. Android 网关通信测试

打开网关的“工业数据采集应用程序 APP”，可以看到网关采样的烟雾状态，如图所示。

该烟雾传感器带有模拟触发按钮，可以按下按钮，模拟烟雾报警，这样网关就能采样到烟雾报警信息了，如图所示。

3.4.5. 多路采集器传感器通信测试

以二氧化碳传感器为例，其接线图参考第 2 章对应章节，或者参考“RS485 传感器接线图”小节。此处忽略。

1.协议命令

（1）读取多通道采集器第 0 个模拟量输入通道二氧化碳的采样值，命令如下。

发送	485	01	04	00 00	00 01	/
应答		01	04	02	xx xx	/

2.使用 Android 网关通信测试

打开网关的“工业数据采集应用程序 APP”，可以通过网关采样 RS485 传感器的数据，如图所示。

多路采集器外围的可燃气体传感器、液位传感器，均可以通过网关数据采集应用程序进行采集测试。

3.4.6. 多路采集器执行器通信测试

以风扇执行器的接线图参考第 2 章对应章节的接线图，或者参考“RS485 传感器接线图”。此处忽略。

1.协议命令

（1）控制多通道采集器第 0 个继电器输出通道闭合，则风扇供电回路上电，风扇旋转，命令如下。

发送	485	01	05	00 00	FF 00	/
应答		01	05	00 00	FF 00	/

（2）控制多通道采集器第 0 个继电器输出通道断开，则风扇供电回路断电，风扇停止旋转，命令如下。

发送	485	01	05	00 00	00 00	/
应答		01	05	00 00	00 00	/

2.使用 Android 网关通信测试

打开网关的“工业数据采集应用程序 APP”，可以通过网关控制风扇开关，如图所示。

长按风扇图标，弹出操作对话框。选择“采集/控制”按钮，如图所示。

进入“风扇控制界面”，点击“打开”、“关闭”按钮，可以控制风扇开关，如图所示。

多路采集器外围的电磁阀，测试方法与上同。

3.5. RS485 设备的 PLC 传输控制

3.5.1. 实验目的

- 熟悉 PLC 的外围电路的接线原理，理解开关量、模拟量的概念；
- 掌握开关量信号、RS485 设备的 PLC 模块采集控制编程方法。

3.5.2. 实验环境

1. 硬件：光电开关，PLC 模块，滑台模块，Mini USB Type-B 下载线/RS232 下载线（8 孔鼠标头母座），若干连接导线，24V 电源。光电开关，PLC 模块，滑台模块的实物如下图所示。

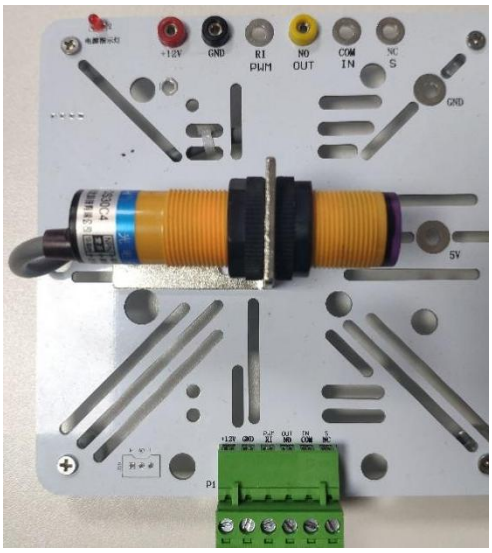


图 3-1 光电开关

图 3-2 PLC 模块

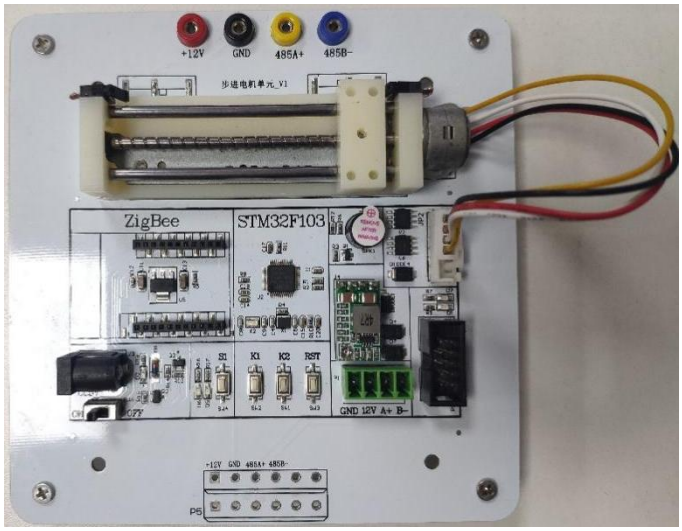


图 3-3 滑台模块

2. 软件：Windows 7 及以上操作系统，GX Works2 开发环境。

3.5.3. 实验原理

1. CX3G 系列 PLC 硬件结构示意图如图 3-4 所示：

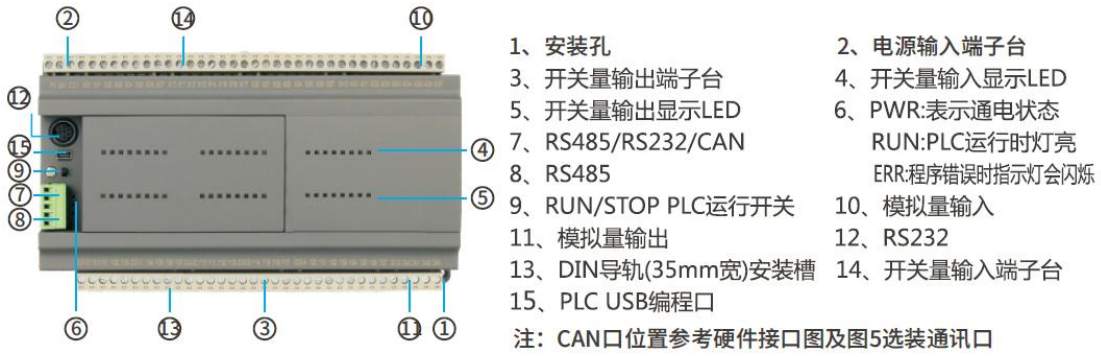


图 3-4 PLC 硬件结构示意图

本实验中使用的 PLC 型号为 CX3G-16MT，它的实物图如图 3-5 所示，其硬件接口的端子较少，如图 3-6 所示：

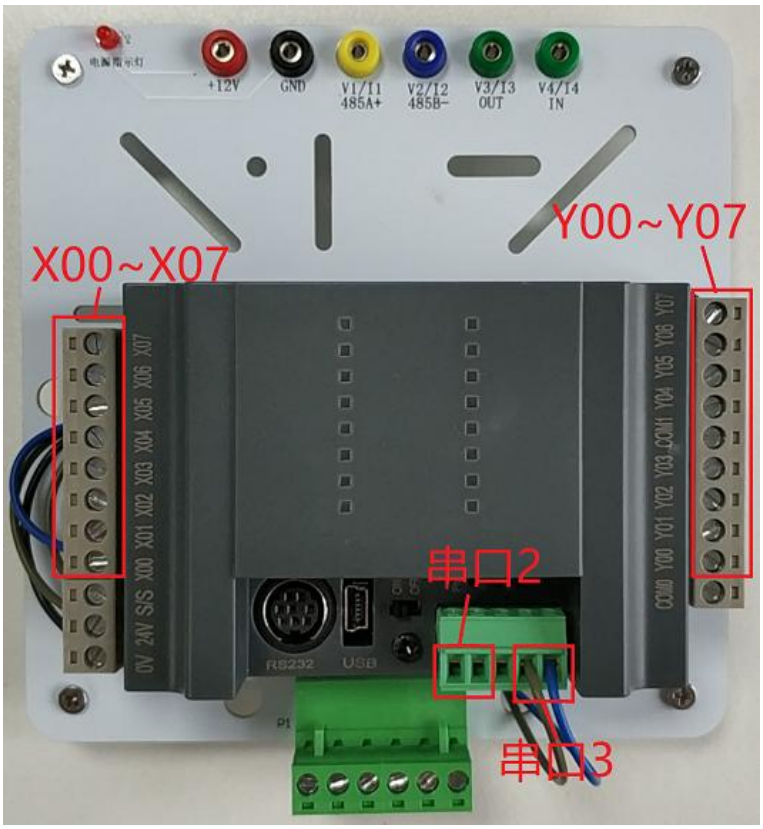


图 3-5 PLC 模块实物图

图 3-6 CX3G-16MT PLC 硬件接口示意图

由实物图或者 PLC 硬件接口图可以看到，CX3G-16MT 一侧的端子分别是 0V、24V、S/S、X00~07；另一侧的端子分别是 COM0、Y00~Y03、COM1、Y04~Y07。

0V 端子接地，24V 端子和 S/S 端子接+24V。光电开关的输出连接在 PLC 的开关量输入端子台的第一个端子即 X00 上，滑台模块通过 RS485 和 PLC 的串口 3 连接。PLC 的供电是 24V DC，滑台模块和光电开关的供电可以是 24V/12V DC。

由于 PLC 模块中的 X00 对应底板上的接口 V4/I4 IN，串口 3 的 A+，B+分别对应底板上的 V1/I1 485A+，V2/I2 485B-，它们被导线连接起来了，所以光电开关的输出可以直接连到 PLC 模块底板上的 V4/I4 IN 口，PLC 模块底板上的 V1/I1 485A+，V2/I2 485B-口分别连接到滑台模块底板上的 485A+和 485A-口。

2. 传输协议

1) RS485 通信协议实现(Modbus RTU)

（1） 协议框架

名称	从机地址	功能码	起始地址	数据	校验码
长度（字节）	1	1	2	2*数据个数	2
编码方式	HEX	HEX	HEX	HEX	HEX

注：

a) 从机地址：被写从机站号，本实验中，PLC 模块的从机站号为 0x0D(13)，滑台模块的从机站号为 0x02；

b) 功能码：0X00-0XFF。

0X03：读取单个或多个寄存器

0X06：写单个寄存器

0X10：写多个寄存器

c) 起始地址：被写从机寄存器首地址，高字节在前，低字节在后

d) 数据：高字节在前

e) 校验码：低字节在前

（2） 简短应答

当从站收到一条指令数据并且校验成功，就要给主站一个简短的应答信号，以示收到发来的命令信息，应答机制为明文。

2) 指令详解

（1） A9 网关发送指令给 PLC，PLC 再发送指令控制滑台。

a) A9 网关向 PLC 发送数据

数据	滑台加工次数
数据长度	2 字节

滑台加工次数

字段	标志意义
0x00 0x01	加工一次
0x00 0x10	加工 16 次

- 举例：

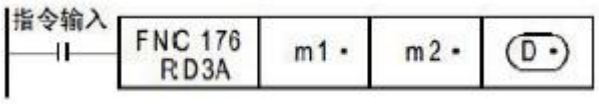
01 06 00 0D 00 05 B9 C9 ： A9 网关向 PLC 发送指令控制滑台加工 5 次，此为 A9 网关向 PLC 发送的数据。

响应指令 ：01 06 00 0D 00 05 B9 C9

b) PLC 向滑台发送数据

串口采用 MODBUS RTU 功能，MODBUS 指令解释：PLC 作为主机时,支持 ADPRW 指令、RD3A 指令，WR3A 指令。本小节仅对 RD3A 指令，WR3A 指令进行解释说明。

- 读取/写入数据指令功能和动作说明

读取数据 RD3A： 

在 CoolMayPLC 中，RD3A 指令对应 Modbus 的 03 号功能。

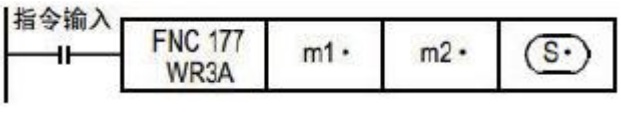
m1 表示被读从机设备的站号，范围 1-247；

m2 表示被读数据在从机设备中的首地址编号；

D.表示读取的寄存器个数，范围 1-125(Modbus ASCII 时范围为 1-45，CAN 通讯时范围为 1-90)，

被读取的数据依次保存在主机 D.+1、D.+2. . 中。

D.-1 地址数值必须设置(=0：串口 2； =1：串口 3； =2：CAN； =3：网络 MODBUS)

写数据 WR3A： 

在 CoolMayPLC 中，WR3A 指令对应 Modbus 的 06 号功能和 10 号功能。

m1 表示被写从机设备的站号，范围 1-247。

m2 表示被写寄存器在从机设备中的首地址编号；

S.表示被写的寄存器个数，范围 1-123(Modbus ASCII 时范围为 1-45，CAN

通讯时范围为 1-90)。

即将被写的数据依次保存在主机 S.+1、S.+2. . 中。

S=1 时，WR3A 指令对应 Modbus 的 06 号功能；

S=2-123 时，WR3A 指令对应 Modbus 的 10 号功能；

S.-1 地址数值必须设置(=0：串口 2； =1：串口 3； =2：CAN； =3：网络 MODBUS)

- RD3A 和 WR3A 仅支持 MODBUS RTU 的以下功能:

03 号功能：读取保持寄存器，在一个或多个保持寄存器中取得当前的二进制值范围 1-125 个。

06 号功能：把具体二进值装入一个保持寄存器(写寄存器)，范围 1 个。

10 号功能：预置多寄存器，把具体的二进制值装入一串连续的保持寄存器(写多个寄存器)，范围 1-123 个。

- 举例：

以下两条指令表示滑台加工一次

WR3A K2 K3 D400 ：PLC 一体机让滑台滑到终点的位置

WR3A K2 K3 D500 ：PLC 一体机让滑台滑到起点的位置

K2 表示写入的从机站号，滑台默认站号为 2；

K3 表示写入从机寄存器首地址；

D400 表示写入数据的个数，写入的数据存储在 D401 中，这里 D401 被赋值为 0x54，D399 为 1，因为 PLC 和滑台通过串口 3 相连；

D500 表示写入数据的个数，写入的数据存储在 D501 中，这里 D501 被赋值为 0；D499 也是 1。

- PLC 指令对应的 485 数据为：

WR3A K2 K3 D400 ：02 06 00 03 00 54 78 06

WR3A K2 K3 D500 ：02 06 00 03 00 00 79 F9

滑台位移位置算法：位移到 0X54*0.6MM=84*0.6MM=50.4MM 位置点

- 此外滑台的加工次数保存在寄存器 D10 中，读取 D10 数据的指令为：

0D 03 00 0A 00 01 A4 C4

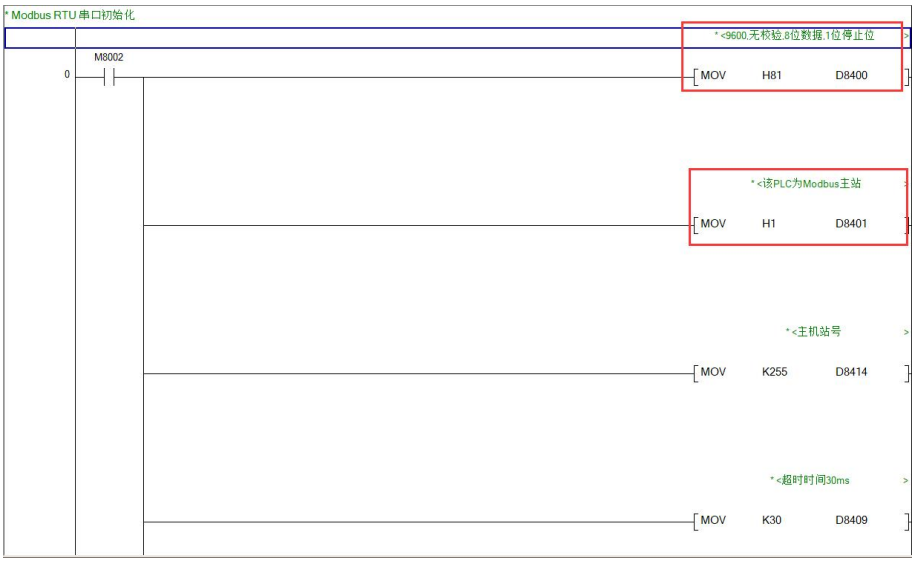
3.5.4. 实验内容

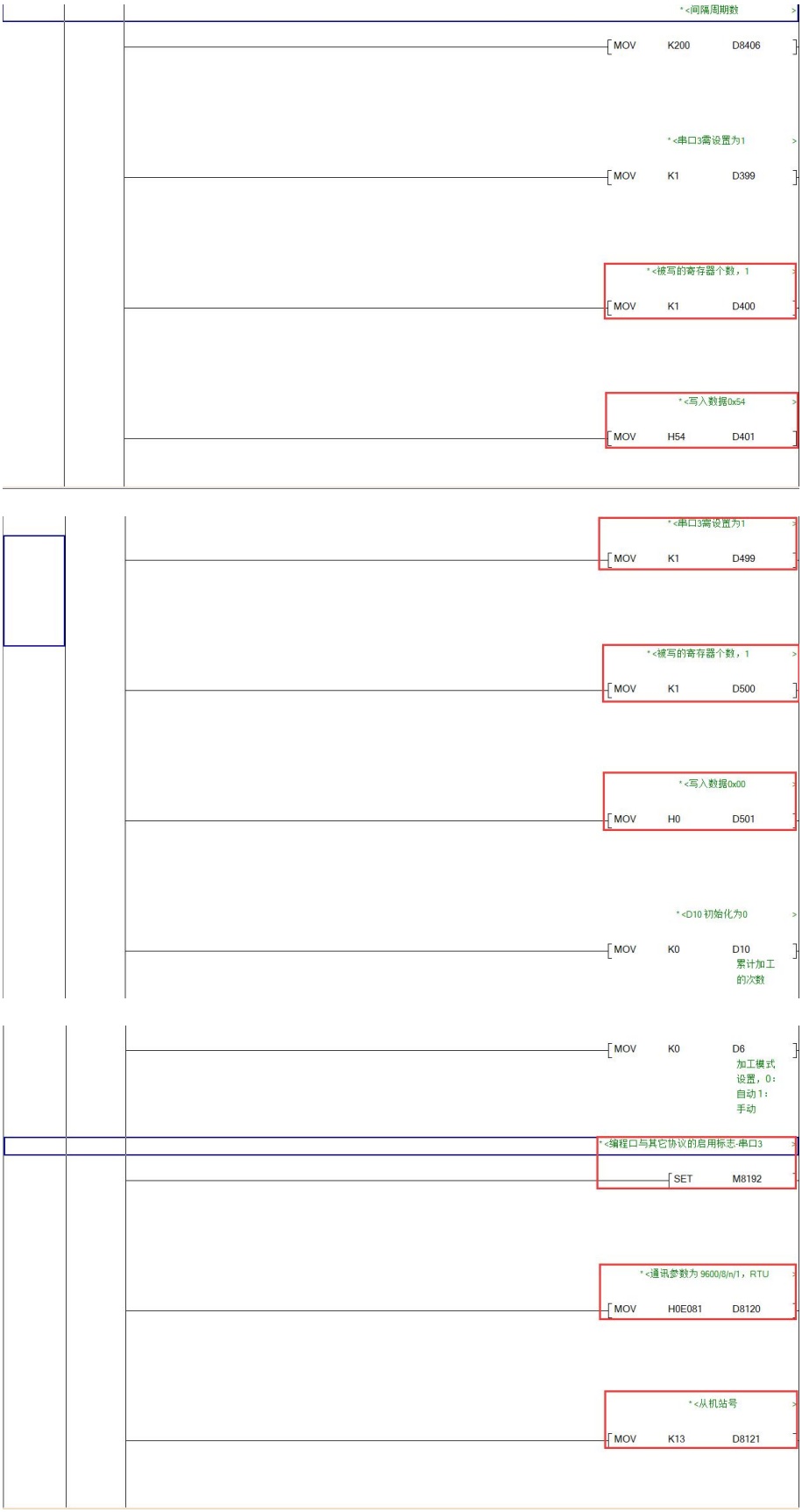
本实验的实验内容是完成基于 PLC 的数据采集与控制，当光电开关有输入时（有障碍物遮挡光电开关），PLC 发送指令控制滑台来回滑动一次。

具体代码如下。

1. 初始化

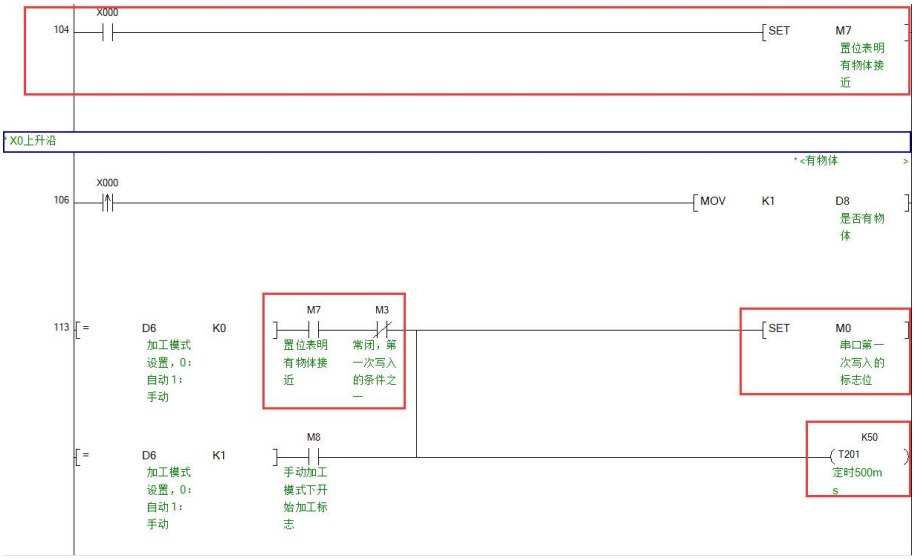
初始化程序如下所示，首先设置串口 3 波特率为 9600，无校验，8 位数据位，1 位停止位，串口通讯模式为 Modbus RTU 并且 PLC 为主站。控制滑台滑动到 50mm 和 0mm，被写寄存器的个数分别为 D400 和 D500，所以被写的数据分别是 D401 和 D501。最后配置串口 2 波特率为 9600，无校验，8 位数据位，1 位停止位，PLC 设为从机，串口 2 是与网关 A9 的通讯接口，这里就不做过多介绍。

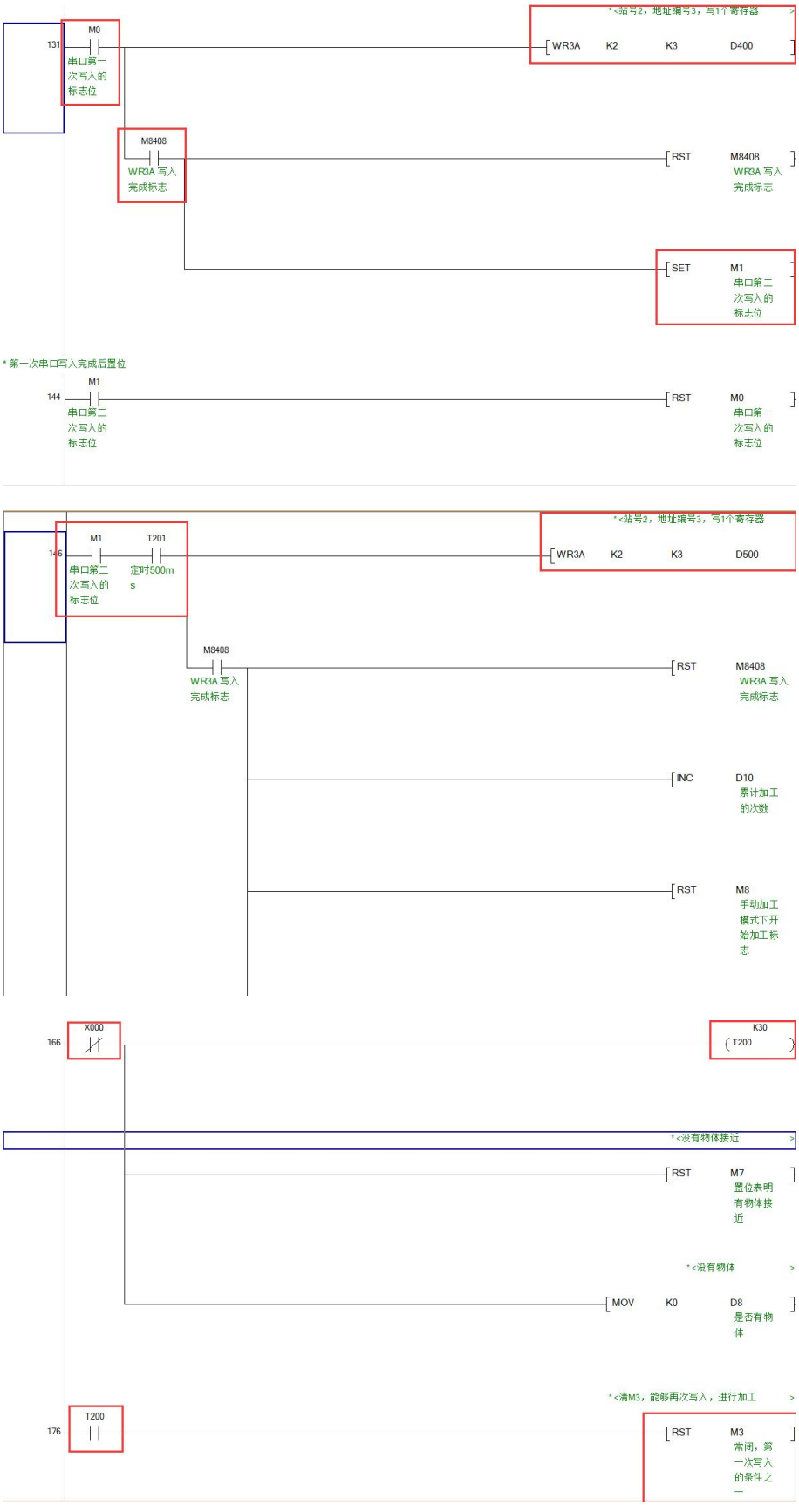



*<间隔周期数
MOV K200 D8406
*<串口3需设置为1
MOV K1 D399
*<被写的寄存器个数，1
MOV K1 D400
*<写入数据0x54
MOV H54 D401
*<串口3需设置为1
MOV K1 D499
*<被写的寄存器个数，1
MOV K1 D500
*<写入数据0x00
MOV H0 D501
*<D10 初始化为0
MOV K0 D10
累计加工的次数
MOV K0 D6
加工模式设置，0：自动 1：手动
*<编程口与其它协议的启用标志-串口3
SET M8192
*<通讯参数为 9600/8/n/1，RTU
MOV H0E081 D8120
*<从机站号
MOV K13 D8121

2. 采集与控制

在光电开关没有输入的情况下，X00 是常开，反之 X00 为常闭状态。所以当光电开关有输入时，X000 被导通，相关寄存器被置位，PLC 通过串口 3 发送数据寄存器 D401 的数据给滑台模块，当数据发送完成后，再发送 D501 的数据给滑台模块，实现了滑台来回滑动的控制效果。




* <站号2，地址编号3，写1个寄存器
131
M0
串口第一次写入的标志位
WR3A K2 K3 D400
M8408
WR3A 写入完成标志
RST M8408 WR3A 写入完成标志
SET M1 串口第二次写入的标志位
* 第一次串口写入完成后置位
144
M1
串口第二次写入的标志位
RST M0 串口第一次写入的标志位
* <站号2，地址编号3，写1个寄存器
1-6
M1
串口第二次写入的标志位
T201
定时500ms
WR3A K2 K3 D500
M8408
WR3A 写入完成标志
RST M8408 WR3A 写入完成标志
INC D10 累计加工的次数
RST M8 手动加工模式下开始加工标志
166
X000
K30
T200
* <没有物体接近
RST M7 置位表明有物体接近
* <没有物体
MOV K0 D8 是否有物体
* <清M3，能够再次写入，进行加工
176
T200
RST M3 常闭，第一次写入的条件之一

3.5.5. 实验步骤

1. 首先将硬件连接好，如图 3-7 所示。

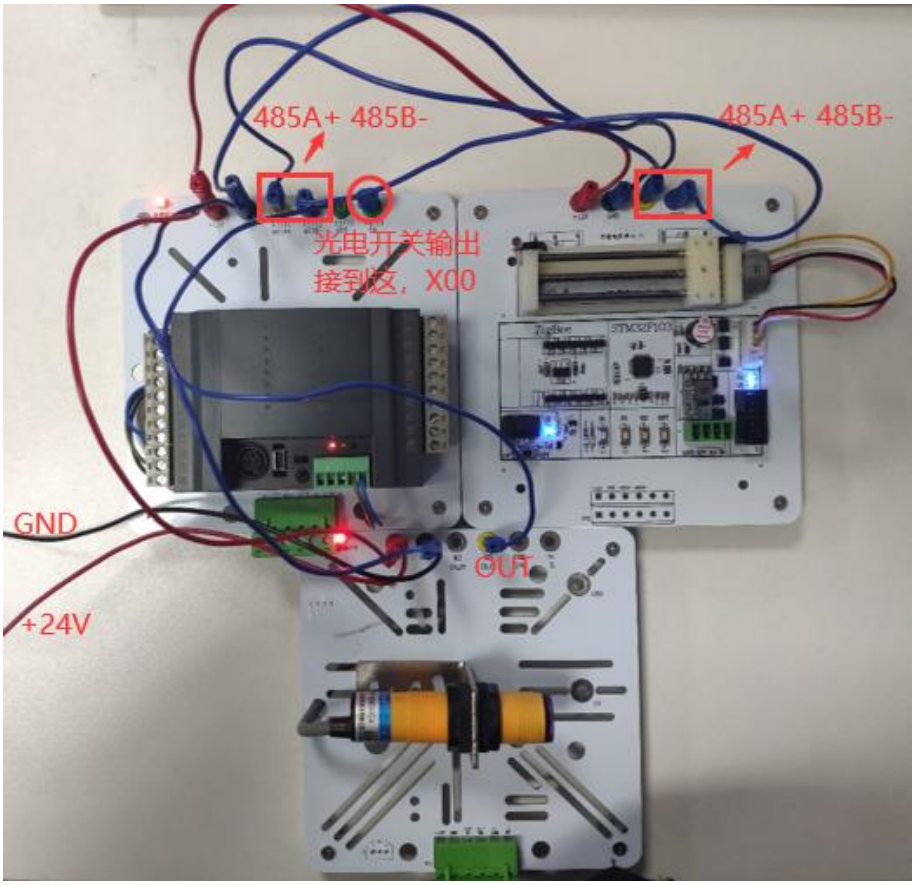


图 3-7 硬件连接图

2. 使用 Mini USB Type-B 下载线连接 PC 和 PLC。

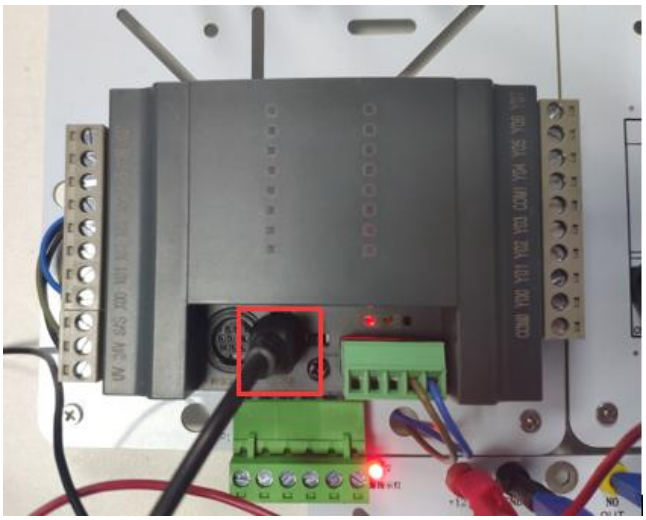


图 3-8 使用下载线连接 PC 和 PLC

3. 打开本实验的代码“motorControl”，如图 3-9 所示。

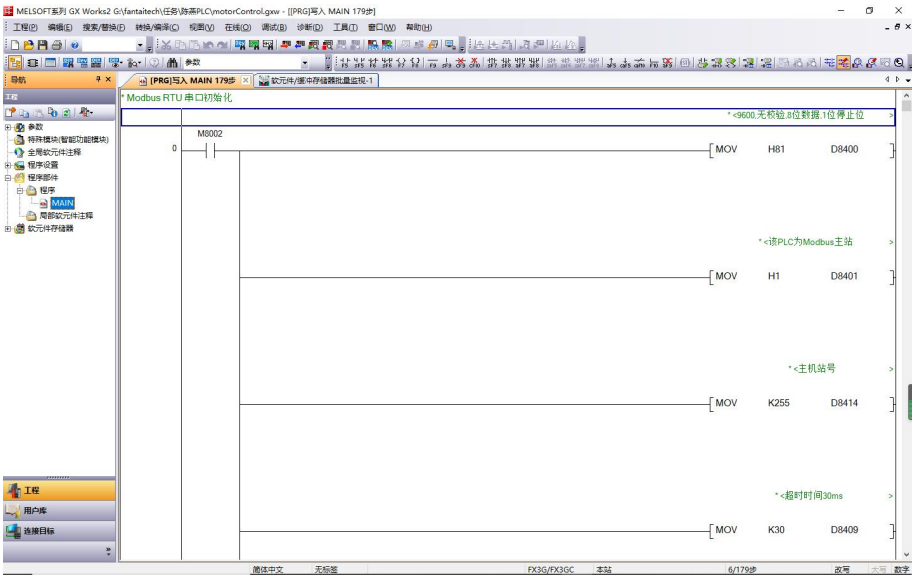


图 3-9 打开 PLC 程序

4. 点选界面左下侧连接目标，然后双击出现的“Connection1”（连接多个 PLC 的话要注意可能不是 Connection1），弹出界面如右侧的窗口所示。

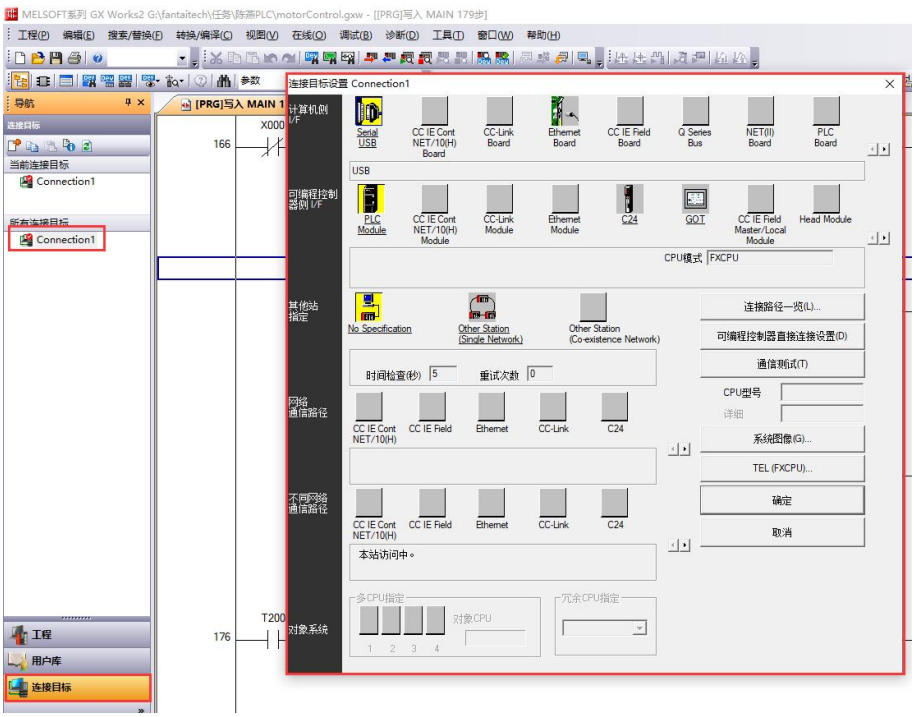


图 3-10 点击“连接目标”

5. 在弹出窗口点击“Serial USB”,然后在次级弹窗中选中“USB”，这是因为此处使用的下载线是 Mini USB Type-B，如果使用的是 RS232 下载线，就需要选择“RS-232C”。点击确定。

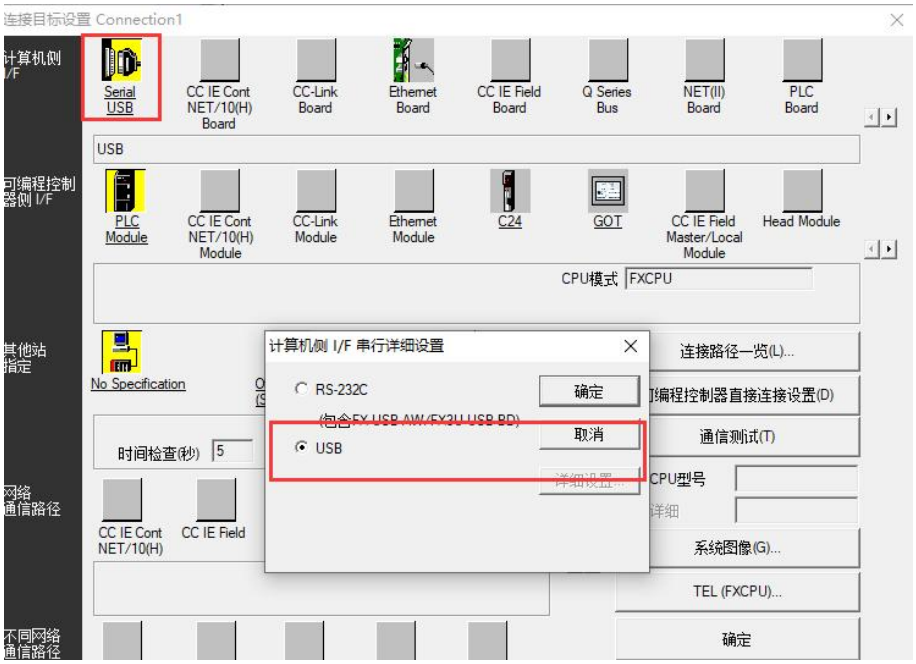


图 3-11 选择“USB”串行方式

6. 点击通信测试，如果设置正确，硬件连接无误，并且驱动已经安装好，会弹出成功连接提示。Win10 在有网络的情况下会在一段时间后自动安装好驱动，非此情况，请从资源中找到驱动并手动安装。

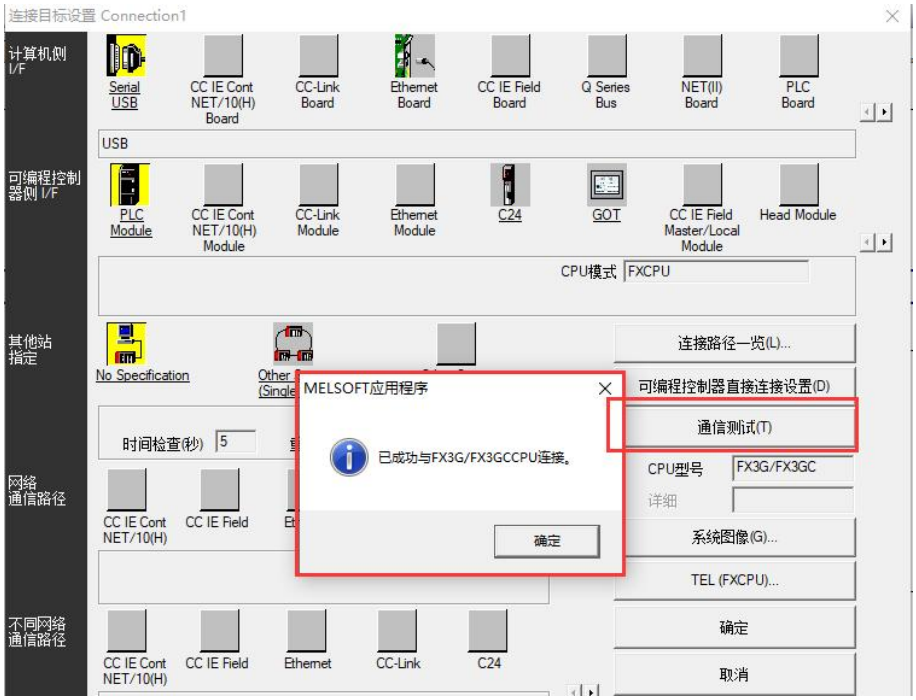


图 3-12 通信测试

7. 通信测试成功后，编译程序，点击转换/编译→转换（所有程序）。编译按钮位置如图 3-13 所示。

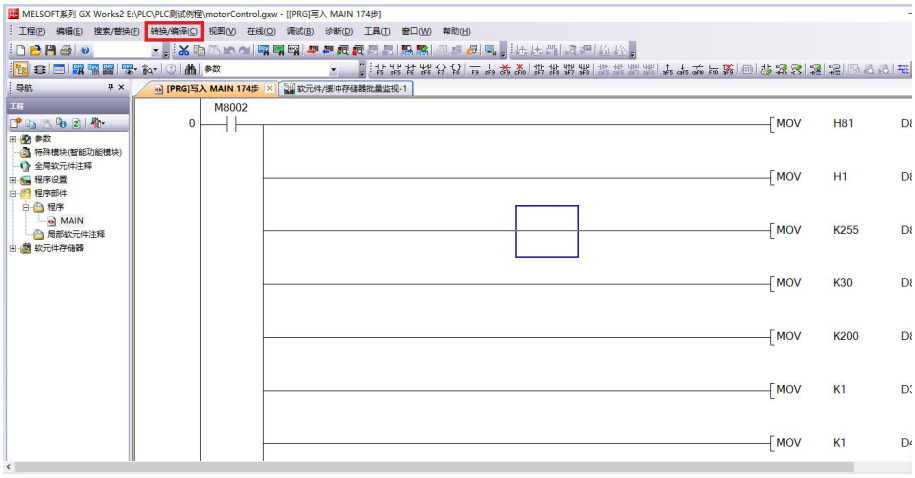


图 3-13 编译按钮示意图

8. 编译好后下载程序，点击 PLC 写入，按钮位置如图 3-14 所示。在如图 3-15 所示界面弹窗中点击“参数+程序”，然后再点击执行；如图 3-16 所示界面为 PLC 写入成功，关闭弹窗即可。

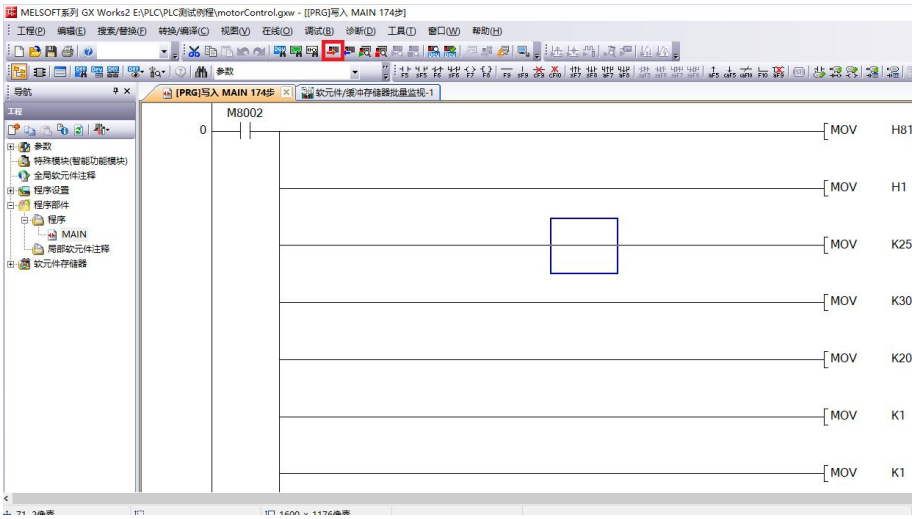


图 3-14 下载按钮示意图

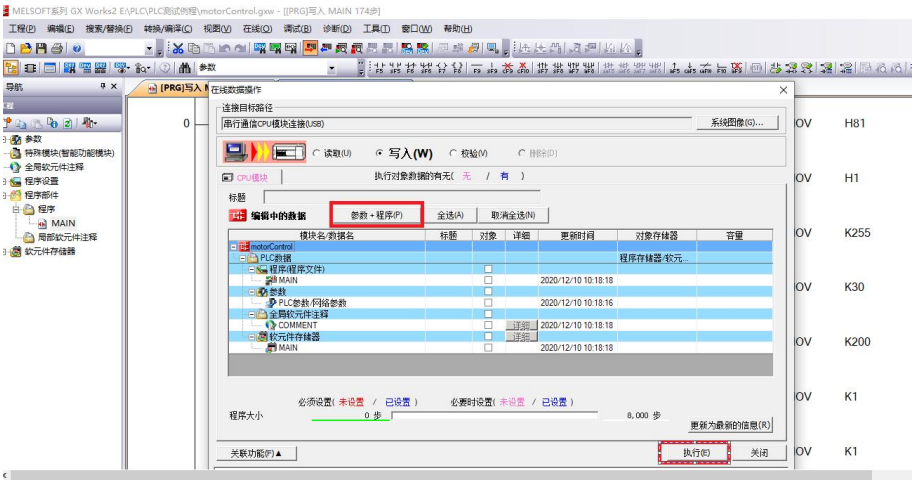


图 3-15 下载程序示意图

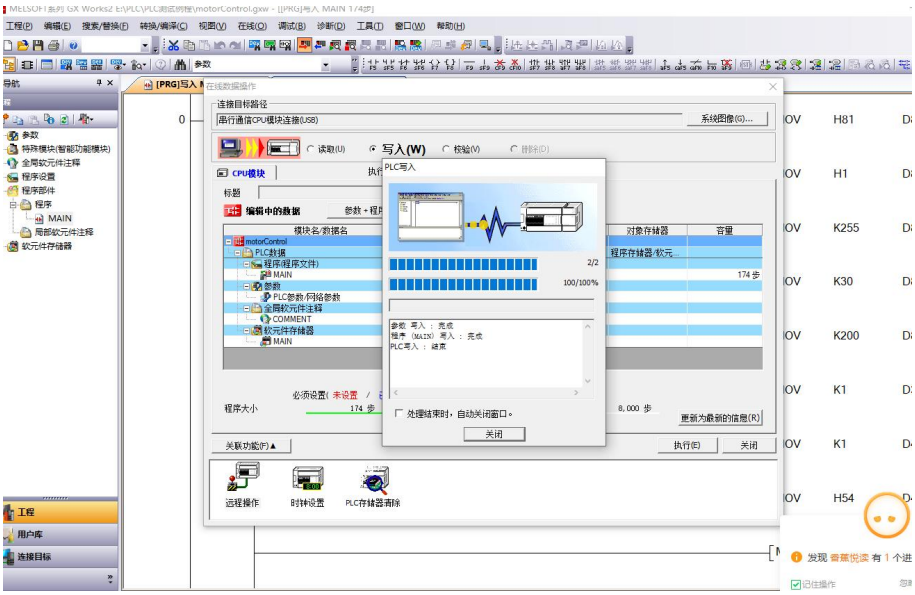


图 3-16 下载成功示意图

9. 用手挡住光电开关，观察滑台。

3.5.6. 实验结果

用手挡住光电开关时，可以看到光电开关的提示灯亮起，此外，PLC 的 X00 的指示灯也亮起。此时，滑台如果在起点位置（靠近步进电机的一侧），会来回滑动一次（起点->终点->起点）；如果滑台一开始不在起点，而是在终点，那么滑台会回到起点，不进行来回滑动。

在滑台滑动期间，需要保持光电开关导通的状态，否则可能出现滑台滑到终点直接停止，不能够往回滑动的状态。

3.6. 工业现场 CAN 总线网络

3.6.1. CAN 总线网络简介

CAN,全称为“Controller Area Network”,即控制器局域网,是国际上应用最广泛的现场总线之一。最初,CAN 被设计作为汽车环境中的微控制器通讯,在车载各电子控制装置 ECU 之间交换信息,形成汽车电子控制网络。比如：发动机管理系统、变速箱控制器、仪表装备、电子主干系统中,均嵌入 CAN 控制装置。

一个由 CAN 总线构成的单一网络中,理论上可以挂接无数个节点。实际应用中,节点数目受网络硬件的电气特性所限制。例如,当使用 Philips P82C250 作为 CAN 收发器时,同一网络中允许挂接 110 个节点。CAN 可提供高达 1Mbit/s 的数据传输速率,这使实时控制变得非常容易。另外,硬件的错误检定特性也增强了 CAN 的抗电磁干扰能力。

CAN 最初出现在 80 年代末的汽车工业中,由德国 Bosch 公司最先提出。当时,由于消费者对于汽车功能的要求越来越多,而这些功能的实现大多是基于电子操作的,这就使得电子装置之间的通讯越来越复杂,同时意味着需要更多的连接信号线。提出 CAN 总线的最初动机就是为了解决现代汽车中庞大的电子控制装置之间的通讯,减少不断增加的信号线。于是,他们设计了一个单一的网络总线,所有的外围器件可以被挂接在该总线上。1993 年,CAN 已成为国际标准 ISO11898(高速应用)和 ISO11519（低速应用）。

CAN 是一种多主方式的串行通讯总线,基本设计规范要求有高的位速率,高抗电磁干扰性,而且能够检测出产生的任何错误。当信号传输距离达到 10Km 时,CAN 仍可提供高达 50Kbit/s 的数据传输速率。

由于 CAN 总线具有很高的实时性能,因此,CAN 已经在汽车工业、航空工业、工业控制、安全防护等领域中得到了广泛应用。

3.6.2. CAN 总线传输电路设计

为了增强 CAN 总线的抗干扰能力，CAN 收发器的 TX0 和 RX0 引脚通过高速光耦 6N137 与 CAN 收发器的引脚 TXD 和 RXD 相连，这样能够实现总线上

各 CAN 节点间的电气隔离。需要特别注意一点：光耦部分电路所采用的两个电源 VCC 和 CAN_V 必须完全隔离，否则采用光耦也就失去了意义。电源的完全隔离可采用小功率电源隔离模块，列如 B0505D-1W 电源隔离模块。这些电路虽然增加了 CAN 节点的复杂程度，但是提高了 CAN 节点的稳定性和安全性。

CAN 收发器与 CAN 总线的接口部分采用了一定的安全和抗干扰措施：CAN 收发器的 CANH 和 CANL 引脚各自通过一个 5 欧姆的电阻与 CAN 总线相连，电阻可起到一定的限流作用，保护 CAN 收发器 免受过流的冲击。CANH 和 CANL 与地之间分别并联了一个 30P 的电容，可以起到滤除总线上的高频干扰的作用，也具有一定的防电磁辐射的能力。另外，在两根 CAN 总线接入端与地之间分别反接了一个保护二极管 IN4148，当 CAN 总线有较高的负电压时，通过二极管的短路可起到一定的过压保护作用。

3.7. CAN 总线通信协议

在 CAN2.0B 的版本协议中有两种不同的帧格式，不同之处为标识符域的长度不同，含有 11 位标识符的帧称之为标准帧，而含有 29 位标识符的帧称为扩展帧。如 CAN1.2 版本协议所描述，两个版本的标准数据帧格式和远程帧格式分别是等效的，而扩展格式是 CAN2.0B 协议新增加的特性。为使控制器设计相对简单，并不要求执行完全的扩展格式，对于新型控制器而言，必须不加任何限制的支持标准格式。但无论是哪种帧格式，在报文传输时都有以下四种不同类型的帧:

折叠帧类型

在报文传输时，不同的帧具有不同的传输结构，下面将分别介绍四种传输帧的结构，只有严格按照该结构进行帧的传输，才能被节点正确接收和发送。

(1)数据帧由七种不同的位域(Bit Field)组成:帧起始(Start of)、仲裁域(Arbitration Field)、控制域(Control Field)、数据域(DataField)、CRC 域(CRC Field)、应答域(ACK Field)和帧结尾(End of)。数据域的长度可以为 0~8 个字节。

1)帧起始(SOF):帧起始(SOF)标志着数据帧和远程帧的起始，仅由一个"显性"位组成。在 CAN 的同步规则中，当总线空闲时(处于隐性状态)，才允许站点开始发送(信号)。所有的站点必须同步于首先开始发送报文的站点的帧起始前沿(该

方式称为"硬同步")。

2)仲裁域:仲裁域由标识符和 RTR 位组成，标准帧格式与扩展帧格式的仲裁域格式不同。标准格式里，仲裁域由 11 位标识符和 RTR 位组成。标识符位有 ID28~IDl8。扩展帧格式里，仲裁域包括 29 位标识符、SRR 位、IDE(Identifier Extension，标志符扩展)位、RTR 位。其标识符有 ID28~IDO。为了区别标准帧格式和扩展帧格式，CANl.0~1.2 版本协议的保留位 r1 现表示为 IDE 位。IDE 位为显性，表示数据帧为标准格式;IDE 位为隐性，表示数据帧为扩展帧格式。在扩展帧中，替代远程请求(Substitute Remote Request，SRR)位为隐性。仲裁域传输顺序为从最高位到最低位，其中最高 7 位不能全为零。RTR 的全称为"远程发送请求(Remote TransmissionRequest)"。RTR 位在数据帧里必须为"显性"，而在远程帧里必须为"隐性"。它是区别数据帧和远程帧的标志。

3)控制域:控制域由 6 位组成，包括 2 个保留位(r0、r1 同于 CAN 总线协议扩展)及 4 位数据长度码，允许的数据长度值为 0~8 字节。

4)数据域:发送缓冲区中的数据按照长度代码指示长度发送。对于接收的数据，同样如此。它可为 0~8 字节，每个字节包含 8 位，首先发送的是 MSB(最高位)。

5)CRC 校验码域:它由 CRC 域(15 位)及 CRC 边界符(一个隐性位)组成。CRC 计算中，被除的多项式包括帧的起始域、仲裁域、控制域、数据域及 15 位为 0 的解除填充的位流给定。此多项式被下列多项式 X15+X14+X10+X8+X7+X4+X3+1 除(系数按模 2 计算)，相除的余数即为发至总线的 CRC 序列。发送时，CRC 序列的最高有效位被首先发送/接收。之所以选用这种帧校验方式，是由于这种 CRC 校验码对于少于 127 位的帧是最佳的。

6)应答域:应答域由发送方发出的两个(应答间隙及应答界定)隐性位组成，所有接收到正确的 CRC 序列的节点将在发送节点的应答间隙上将发送的这一隐性位改写为显性位。因此，发送节点将一直监视总线信号已确认网络中至少一个节点正确地接收到所发信息。应答界定符是应答域中第二个隐性位，由此可见，应答间隙两边有两个隐性位:CRC 域和应答界定位。

7)帧结束域:每一个数据帧或远程帧均由一串七个隐性位的帧结束域结尾。这样，接收节点可以正确检测到一个帧的传输结束。

(2)错误帧错误帧由两个不同的域组成:第一个域是来自控制器的错误标志;第二个域为错误分界符。

1)错误标志:有两种形式的错误标志。

①激活(Active)错误标志。它由 6 个连续显性位组成。

②认可(Passive)错误标志。它由 6 个连续隐性位组成。

它可由其他 CAN 总线协议控制器的显性位改写。

2)错误界定:错误界定符由 8 个隐性位组成。传送了错误标志以后，每一站就发送一个隐性位，并一直监视总线直到检测出 1 个隐性位为止，然后就开始发送其余 7 个隐性位。

(3)远程帧: 远程帧也有标准格式和扩展格式,而且都由 6 个不同的位域组成:帧起始、仲裁域、控制域、CRC 域、应答域、帧结尾。与数据帧相比，远程帧的 RTR 位为隐性，没有数据域，数据长度编码域可以是 0~8 个字节的任何值，这个值是远程帧请求发送的数据帧的数据域长度。当具有相同仲裁域的数据帧和远程帧同时发送时，由于数据帧的 RTR 位为显性，所以数据帧获得优先。发送远程帧的节点可以直接接收数据。

(4)过载帧 过载帧由两个区域组成:过载标识域及过载界定符域。下述三种状态将导致过载帧发送:

1)接收方在接收一帧之前需要过多的时间处理当前的数据(接收尚未准备好);

2)在帧空隙域检测到显性位信号;

3)如果 CAN 节点在错误界定符或过载界定符的第 8 位采样到一个显性位节点会发送一个过载帧。

3.8. CAN 测控终端的传输控制实验

3.8.1. 实验目的

- 熟悉 CAN 测控终端的硬件电路设计、信号接口定义，CAN 总线协议；
- 掌握 CAN 通信的嵌入式编程方法。

3.8.2. 实验环境

1. 硬件：CAN 测控终端，转动车轮模块，网关，实物如下图所示。

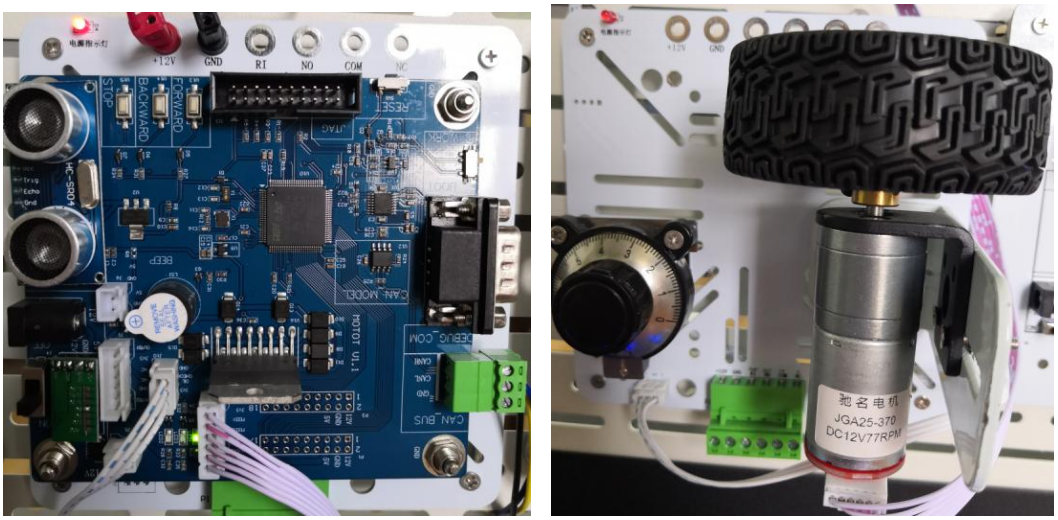


图 3-17 CAN 测控终端、转动轮模块

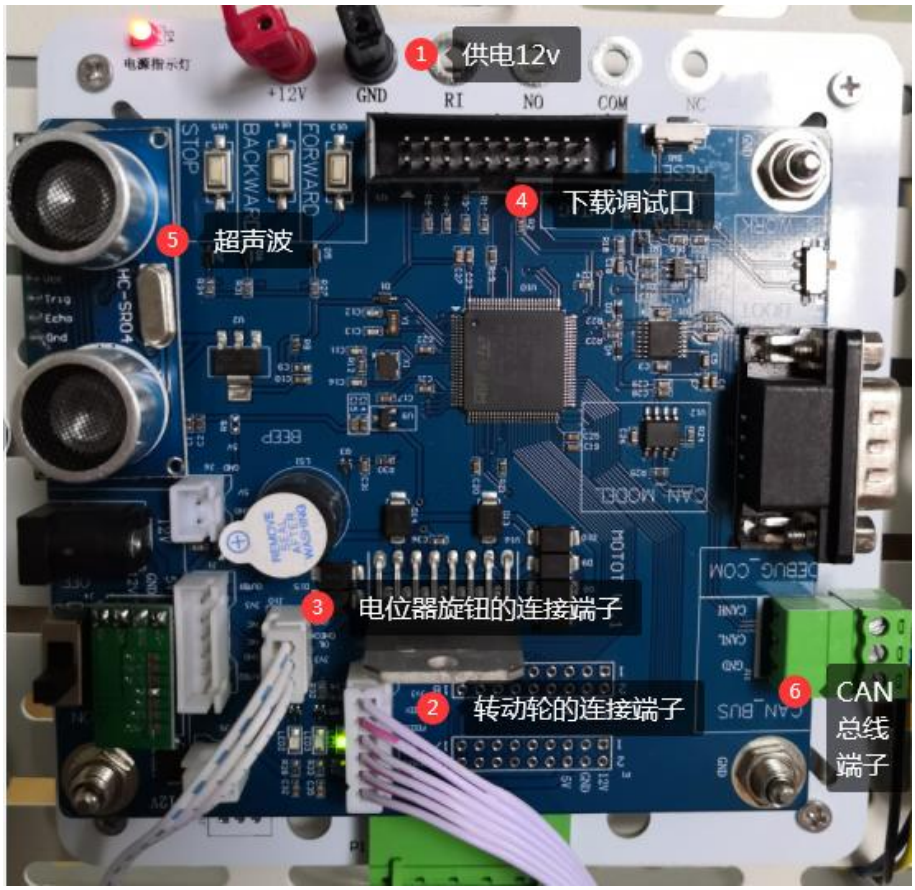
2. 软件：Windows 7 及以上操作系统，keil5 开发环境、CAN 测控终端车速油耗嵌入式程序。

3.8.3. 实验原理

1. 硬件组成

CAN 测控终端由处理器单元、电源管理单元、指示单元、RS232 通信单元、CAN 通信单元、马达控制单元、马达测速单元、油门采集单元、功能按键单元等组成，完成一套车载的行驶方向、速度控制、速度采集、油耗采集等组成的系统；如下图所示。

CAN 测控终端，实物图如图所示。



直流电机车轮执行器板，实物如图所示。

其中，直流电机车轮主要由直流编码器电机、减速器、论坛、霍尔编码器等组成，组成结构如图所示。

编码器接线定义：

红色线	电机电源+	调换可以控制电机正反转
黑色线	编码电源-负	正负极不可接错，3.3-5V
黄色线	信号反馈	电机转一圈 11 个信号
绿色线	信号反馈	电机转一圈 11 个信号
蓝色线	编码电源+正	正负极不可接错，3.3-5V
白色线	电机电源-	调换可以控制电机正反转

电机驱动原理：

采用 L298N 芯片驱动直流电机正反、快慢旋转。L298N 芯片引脚编号与功能，如表所示。

引脚编号	名称	功能
1	电流传感器 A	在该引脚和地之间接小阻值电阻可用来检测电流
2	输出引脚 1	内置驱动器 A 的输出端 1，接至电机 A
3	输出引脚 2	内置驱动器 A 的输出端 2，接至电机 A
4	电机电源端	电机供电输入端，电压可达 46V
5	输入引脚 1	内置驱动器 A 的逻辑控制输入端 1
6	使能端 A	内置驱动器 A 的使能端
7	输入引脚 2	内置驱动器 A 的逻辑控制输入端 2
8	逻辑地	逻辑地
9	逻辑电源端	逻辑控制电路的电源输入端为 5V
10	输入引脚 3	内置驱动器 B 的逻辑控制输入端 1
11	使能端 B	内置驱动器 B 的使能端

12	输入引脚 4	内置驱动器 B 的逻辑控制输入端 2
13	输出引脚 3	内置驱动器 B 的输出端 1，接至电机 B
14	输出引脚 4	内置驱动器 B 的输出端 2，接至电机 B
15	电流传感器 B	在该引脚和地之间接小阻值电阻可用来检测电流

L298N 驱动 A/B 电机的控制逻辑真值表，如表所示。

输入信号			电机运动方式
使能端 A/B	输入引脚 1/3	输入引脚 2/4	
1	1	0	前进
1	0	1	后退
1	1	1	紧急停车
1	0	0	紧急停车
0	x	x	自由转动

因此可使用 MCU 管脚控制输入引脚 1、输入引脚 2 的电平，经过驱动芯片增大驱动电流后，将驱动后的输出引脚 1、输出引脚 2，分别接到电机的“电机电源+”、“电机电源负-”，就可以把电机驱动起来。

2. 电路原理图

1) CAN 接口电路图和描述

如图 3-18、图 3-19 所示，CAN 总线读写引脚分别连接在单片机的引脚 PA12、PA13 上。J11 为 CAN 接口。

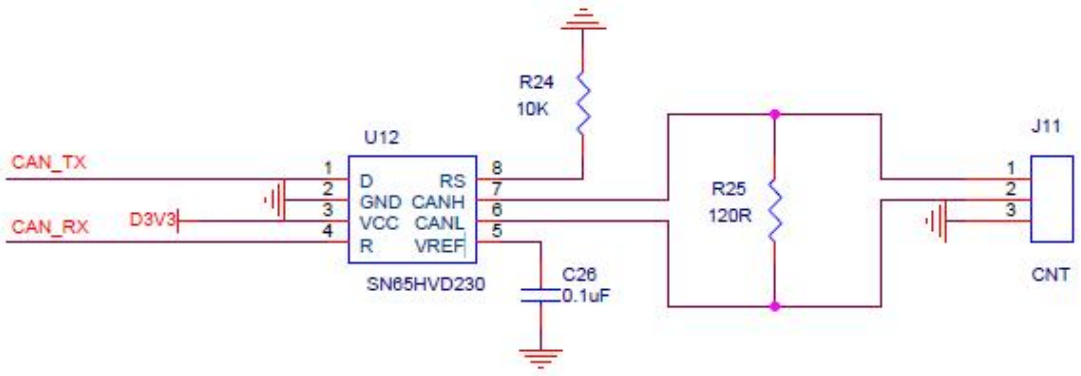


图 3-18CAN 接口原理图

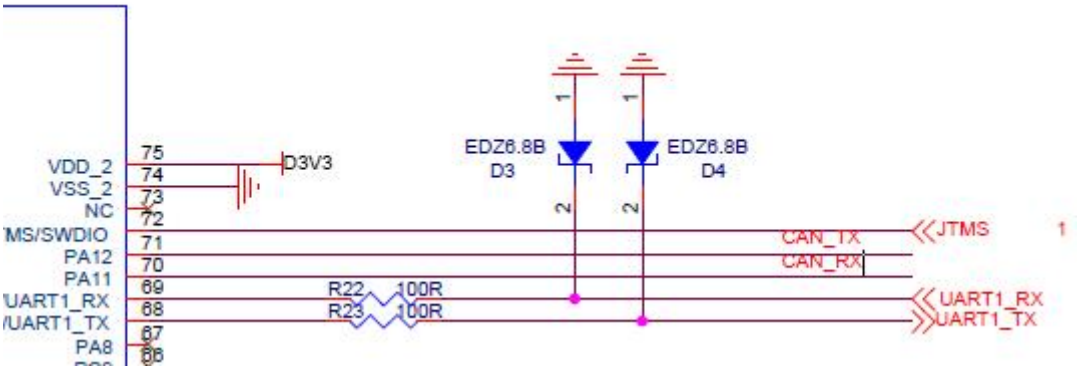


图 3-19CAN 接口与 MCU 连接原理图

2) 直流电机控制电路原理图和描述

如图 3-20、图 3-21 所示，直流电机通过 U16 的引脚 IN1，IN2、ENA 与 MCU 相连，IN1、IN2、ENA 分别连接在引脚 PD15、PD14 和 PA6 上。IN1 和 IN2 控制电机的转动方向，ENA 产生 PWM 波驱动电机，可以通过控制 PWM 波的占空比来控制电机的转动速度。

如图 3-22 所示，转速检测引脚 MOTOR_FEED1 和 MOTOR_FEED1 分别连接在引脚 PB1 和 PD3 上，用来检测小车的速度。

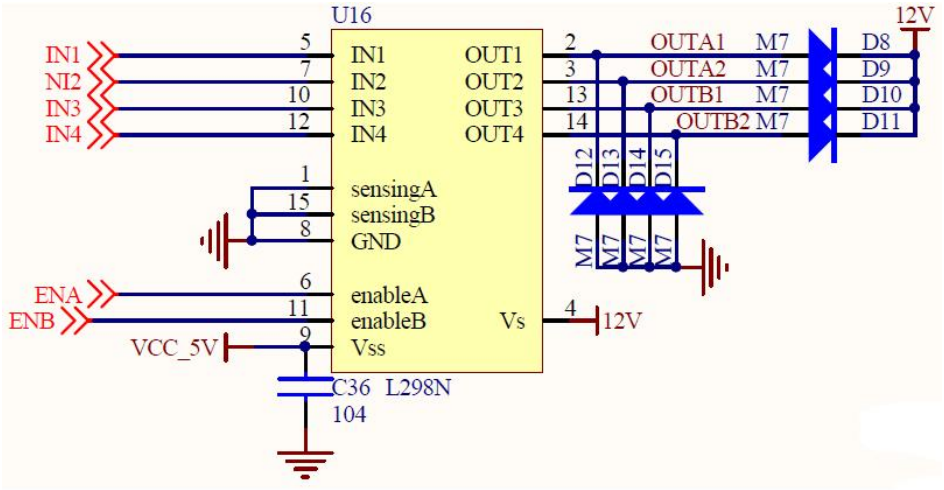


图 3-20 直流电机原理图

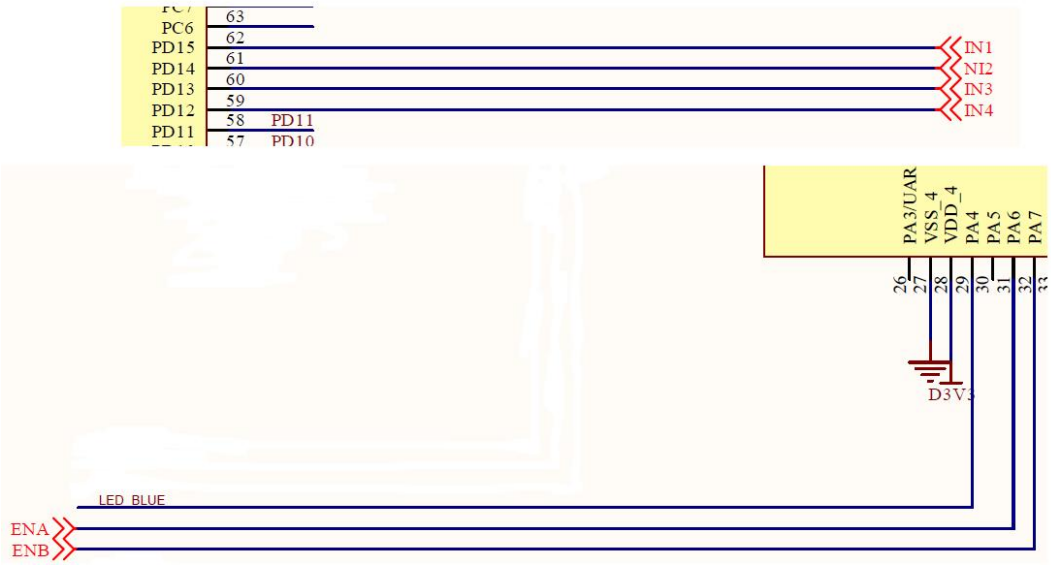


图 3-21 直流电机与 MCU 连接原理图

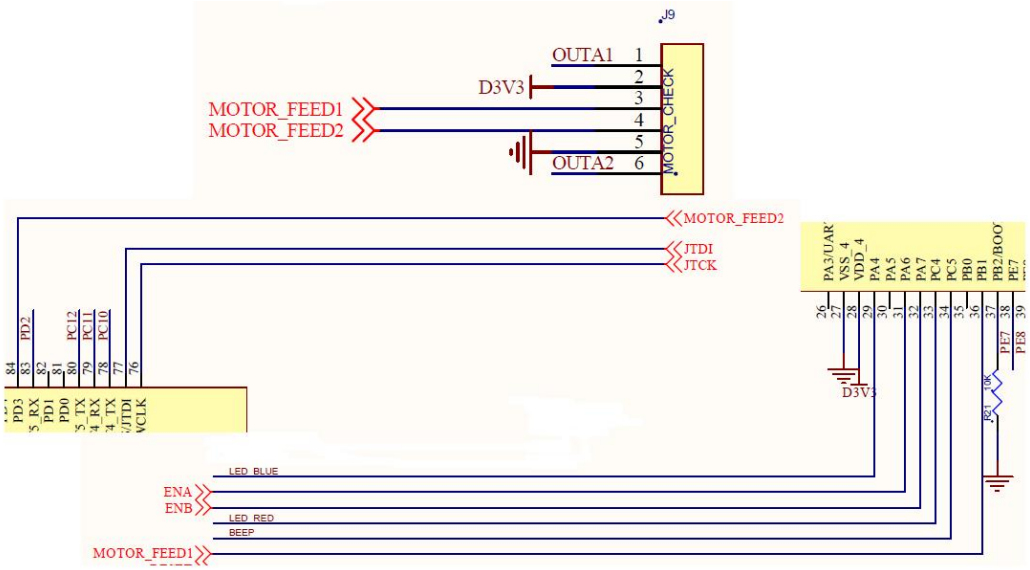


图 3-22 转速检测原理图

3) 超声波电路原理图和描述

如图 3-23 所示，超声波模块通过 P4 上的引脚 2，3 与 MCU 连接，分别连接在 MCU 的引脚 PE2 和 PE3 上。MCU 可以通过引脚 PE2 和 PE3 来读取超声波传感器采集到的数据，根据该数据可以计算出传感器与障碍物的距离。

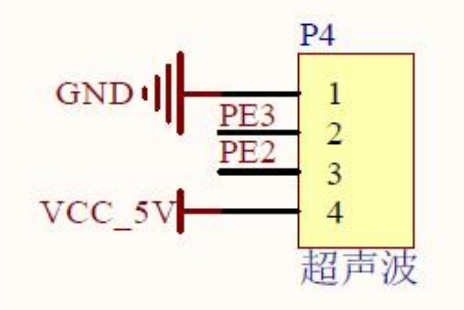


图 3-23 超声波模块原理图

4) 电位器采集电路

如图 3-24 所示，调速模块通过 J10 上的引脚 2 与 MCU 连接，连接在 MCU 的引脚 PC2 上。MCU 通过采集引脚 PC2 上的电压值来调节小车转速。

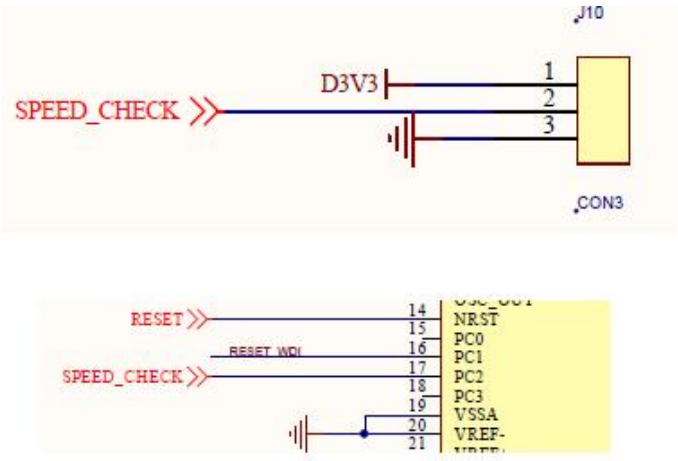


图 3-24 调速原理图

3. CAN 总线通信协议

1）协议框架

名称	协议 ID	指令类型	指令代码	参数长度	参数内容	校验码
长度（字节）	1	1	1	1	n	1
编码方式	HEX	HEX	HEX	HEX	HEX	HEX

注：

1、协议 ID：现暂取 0x7E；

2、指令类型：用于标识“上下行”“加密类型”“回复方式”和“是否广播”，具体格式如下：

比特 7	比特 6	比特 5	比特 4	比特 3	比特 2	比特 1	比特 0
0：下行 1：上行	保留	00：无加密 01：加密类型 1		00：无回复 01：回复 ACK 10：回操作结果		0：广播 1：点对点	保留

注：当加密类型为 1 时，对下行报文中的指令代码、操作口令、参数长度、参数内容进行加密，对上行报文中的指令代码、参数长度、参数内容进行加密；具体加密算法待商榷。

3、参数长度：参数内容的字节长度；

4、CRC 校验码：报文结构中的数据格式都采用大端模式（即：高字节在前，低字节在后）。

5、指令代码

6、参数内容：见指令详解

校验码：对除校验码外的数据根据规定把相关字段加密后而进行 CRC（Cyclic Redundancy Check）校验所得的 CRC 校验码，是一个附加的错误侦查机制,采用 CRC8 校验，见附件。

简短应答

当接收方收到一条指令数据并且校验成功，就要给发送方一个简短的应答信号，以示收到发来的命令信息，应答机制为明文。

名称	协议 ID	指令类型	指令代码	参数长度	参数内容	校验码
长度(字节)	1	1	1	1	0	1
编码方式	7E	HEX	01	HEX	0	HEX

指令详解：

3.1 车辆行驶方向上传(0X11/ACK)

1.上行指令结构(指令代码 0x11)：

参数内容	方向	保留
参数长度	1 字节	n 字节

行驶方向

字段	标志意义
0x00	**STOP**
0x01	**前进**
0x02	**后退**

举例：

7E　　协议 ID

86　　上行，不加密，需回复 ACK，点对点通信

11　　指令代码

01　　参数内容长度

01　　参数内容：行车 D 档

CRC(1B)　　CRC8 校验

2.响应指令结构（指令代码 ACK）：

7E 42 01 00　CRC(1B)

3.2 油耗，车速上传(0X13/ACK)

1.上行指令结构(指令代码 0x13)：

参数内容	方向	油耗	车速
参数长度	1 字节	1 字节	1 字节

行驶方向

字段	标志意义
0x00	停止
0x01	前进
0x02	后退

油耗

字段	标志意义
0x00-0xFF	0-255L/100KM

车速

字段	标志意义
0x00-0xFF	0-255KM/H

举例：

7E　　协议 ID

86　　上行，不加密，需回复 ACK，点对点通信

13　　指令代码

03　　参数内容长度

01　　参数内容：行车 D 档

05　　油耗 5L/KM

1F　　车速 31KM/H

CRC(1B)　　CRC8 校验

2.响应指令结构（指令代码 ACK）：

7E 42 01 00　CRC(1B)

3.3 车辆行驶方向控制 (0X15/ACK)

1.下行指令结构(指令代码 0x15)：

参数内容	方向	保留
参数长度	1 字节	N 字节

方向

字段	标志意义
0x03	停止
0x01	前进
0x02	后退

举例：

7E 协议 ID

06 上行，不加密，需回复 ACK，点对点通信

15 指令代码

01 参数内容长度

01 参数内容：03 停止 01 前进 02 后退

CRC(1B) CRC8 校验

2.响应指令结构（指令代码 ACK）：

7E 82 01 00 54

3.8.4. 实验内容

使用网关数据采集应用程序，与 CAN 测控终端连接，实时接收车轮的转动速度（模拟车速）、模拟油耗。旋转转动轮上的旋钮，可以调节车轮转动速度。同时可以点击网关按钮控制车轮前进、后退、停止转动。

1. 程序流程图

硬件上电以后完成软件系统的初始化、外设及通信初始化、定时任务的车辆信息采集（油门、行驶方向、数据上传等）、接收到的 CAN 通信命令处理、按键任务处理、超声波检测、喂狗等；如图 3-25 所示。

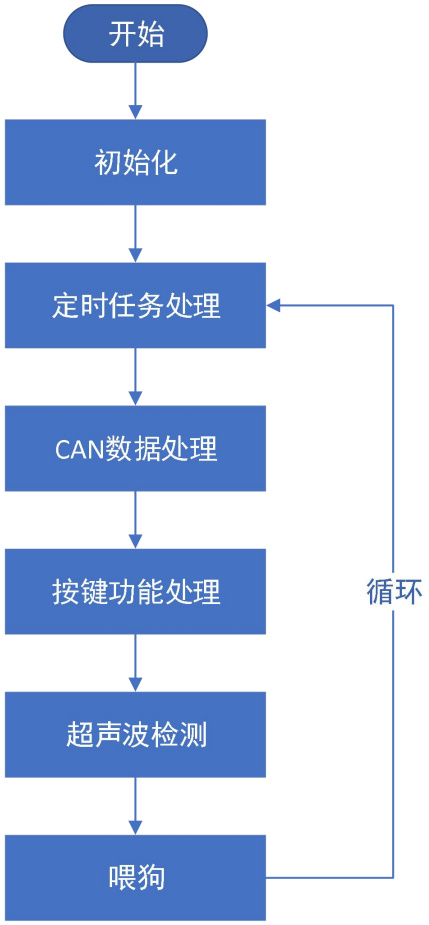


图 3-25 程序流程图

2. 关键代码解析

(1) 定时处理任务函数

在定时处理函数中，实现定时采集调速模块的电压值对车辆进行调速；另外定时上传车辆速度以及油耗等数据。

```
void Timing_Process(void)
{
    if(1 == Flag_Adc)                    //定时器 100ms   置 1
    {
        Flag_Adc = 0;
        Adc_Check();                     //定时器每 100ms 进行 ad 采样  并调速
    }
    if(1 == Flag_Oil_Speed)              //定时器 200ms   置 1
    {
```

```
            Flag_Oil_Speed = 0;
            Sensor_Data_Update();            //定时器每 200ms 进行数据上传
        }
        Watchdog();
}
```

(2) CAN 数据处理函数

CAN 数据处理函数中主要对接收到的 CAN 数据进行处理。

```
    void Data_Dispose(void)
    {
        int i =0;
        unsigned char *p;
        u8 crcValue;
        if(1 == Flag_RxMessage)
        {
            /*  数据转存  */
            Flag_RxMessage = 0;
            if(0x7e == RxMessage.Data[0])
            {
            if(0x86 == RxMessage.Data[1])       //接收到数据进行解析
                {
                    CAN_RxMessage1.StdId = RxMessage.StdId;
                    CAN_RxMessage1.ExtId = RxMessage.ExtId;
                    CAN_RxMessage1.IDE          = RxMessage.IDE;
                    CAN_RxMessage1.RTR  = RxMessage.RTR;
                    CAN_RxMessage1.DLC          = RxMessage.DLC;
                    CAN_RxMessage1.FMI          = RxMessage.FMI;

                    CAN_RxMessage1.StdId = RxMessage.StdId;
```

```
                    memcpy(&CAN_RxMessage1.Data[0],&RxMessage.Data[0],
CAN_RxMessage1.DLC);
                    switch(CAN_RxMessage1.Data[2])
                    {
                        case 0x11:
                        //车辆方向
                            printf("车辆行驶方向：%0x
                            ",CAN_RxMessage1.Data[4]);
                            break;
                        case 0x13:
                        //油耗和速度
                            printf("车辆方向、油耗和速度：%0d      %0d
    %0d
    ",CAN_RxMessage1.Data[4],CAN_RxMessage1.Data[5],CAN_RxMessage1.Data[6]);
                            break;
                    }
                    CAN_Command_DATA.flag                                    = 0x7E;
                    CAN_Command_DATA.Protocol_ID                 = 0x42;
                    CAN_Command_DATA.Instruction                 = 0x01;
                    CAN_Command_DATA.Instruction_Length = 0x00;
                    CAN_Command_DATA.CRC8                                    =
MakeCRC8(&CAN_Command_DATA.flag,((CAN_Command_DATA.Instruction_Length)+4));
//计算 CRC8
                    p = &CAN_Command_DATA.flag;
                    for(i = 0; i < (CAN_Command_DATA.Instruction_Length + 4); i++){
                        Package_Send[i] = *p;
                        p++;
                    }
                *p = CAN_Command_DATA.CRC8;
```

```
    Can_Send_Msg(Package_Send,((CAN_Command_DATA.Instruction_Length)+5));
                }
                  else if(0x42 == RxMessage.Data[1]) //收到 ACK 回复
                  {
                      if(0x01 == RxMessage.Data[2])
                      {
                          Flag_Direction_Time = 0;
                          Direction_Stick = 0;
                          Cnt_Direction_Time = 0;
                          Cnt_Oil_Time = 0;
                          printf("数据已上传成功");
                      }
                  }
                  else if(0x06 == RxMessage.Data[1] && 0x15 == RxMessage.Data[2])
    //CAN 控制小车启停
                  {
                      crcValue = MakeCRC8(&RxMessage.Data[0],5);
                      //printf("crc=%02x",crcValue);
                      if(crcValue == RxMessage.Data[5])
                      {
                          if(0x03 == RxMessage.Data[4])   //停止
                          {
                              Car_State = 0;

    GPIO_ResetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15);          //
停止
                              GPIO_ResetBits(GPIOC,GPIO_Pin_4);
                  //红灯灭
```

```
                    GPIO_ResetBits(GPIOA,GPIO_Pin_4);
            //蓝灯灭
                    CanControlCarAck();
                }
                else if(0x01 == RxMessage.Data[4]) //前进
                {
                    Car_State = 1;
                    GPIO_SetBits(GPIOA,GPIO_Pin_4);
                  //蓝灯亮
                    GPIO_ResetBits(GPIOC,GPIO_Pin_4);
                //红灯灭
                    GPIO_SetBits(GPIOD,GPIO_Pin_13|GPIO_Pin_15);
//前进
                    GPIO_ResetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_14);
                    CanControlCarAck();
                }
                else if(0x02 == RxMessage.Data[4]) //后退
                {
                    Car_State = 2;
                    GPIO_ResetBits(GPIOA,GPIO_Pin_4);
            //蓝灯灭
                    GPIO_SetBits(GPIOC,GPIO_Pin_4);
                //红灯亮
                    GPIO_ResetBits(GPIOD,GPIO_Pin_13|GPIO_Pin_15);
    //后退
                    GPIO_SetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_14);
                    CanControlCarAck();
                }
              }
```

```
                }
                CAN_Command_DATA.flag=0;
                CAN_Command_DATA.Protocol_ID=0;
                CAN_Command_DATA.Instruction=0;
                for(i = 0; i < 8; i++)
                {
                    RxMessage.Data[i] = 0;
                }
        }
}
Watchdog();
  }
```

(3) 按键功能处理函数

按键功能处理函数中根据不同功能的按键按下后对车辆的运行方向进行切换。

```
  void Check_Key(void)
  {
    if(1 == Key_Flag)
      {
          Key_Flag=0;
          Key_Val=KEY_Scan(1);
          if(Key_Val!=0)
          {
              if(Key_Val == KEY0_PRES)
              {
                  Car_State = 1;
                  GPIO_SetBits(GPIOA,GPIO_Pin_4);          //蓝灯亮
                  GPIO_ResetBits(GPIOC,GPIO_Pin_4);     //红灯灭
                  GPIO_SetBits(GPIOD,GPIO_Pin_13|GPIO_Pin_15);   //前进
```

```
                    GPIO_ResetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_14);
                    Direction_Data_Update();        //进行数据上传
                }
                    else if(Key_Val == KEY1_PRES)
                    {
                        Car_State = 2;
                        GPIO_ResetBits(GPIOA,GPIO_Pin_4);   //蓝灯灭
                        GPIO_SetBits(GPIOC,GPIO_Pin_4);     //红灯亮
                        GPIO_ResetBits(GPIOD,GPIO_Pin_13|GPIO_Pin_15);          //后退
                        GPIO_SetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_14);
                        Direction_Data_Update();        //进行数据上传
                    }
                    else if(Key_Val == KEY2_PRES)
                    {
                        Car_State = 0;
    GPIO_ResetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15);        //
停止
                        GPIO_ResetBits(GPIOC,GPIO_Pin_4);           //红灯灭
                        GPIO_ResetBits(GPIOA,GPIO_Pin_4);           //蓝灯灭
                Direction_Data_Update();                    //进行数据上传
    }
                    Key_Val = 0;
                }
                Watchdog();
            }
    }
```

(4) 超声波检测函数

在超声波检测函数中根据超声波传感器测到的距离，决定车辆是否继续前行

或后退。

```
void Ultrasonic_Check(void)
{
    static uint8_t fristCarStatus = 0;
    delay_ms(500);                          //隔一段时间开始超声波检测
    /*给一个 20us 左右的高电平，启动超声波测距*/
    GPIO_ResetBits(GPIOE, GPIO_Pin_2);
    delay_us(5);
    GPIO_SetBits(GPIOE, GPIO_Pin_2);
    delay_us(20);
    STOP_TIME4();
    GPIO_ResetBits(GPIOE, GPIO_Pin_2);                          //启动测距
    while (!GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_3));      //等待测距芯片给出的高电平
    START_TIME4();                                              //定时器开始计算时间
    while (GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_3));       //当测距芯片输出的高电平结束时，检测高电平持续的时间
    distance = TIM4->CNT / 1000000.0f * 34000 / 2;          //测试距离=高电平时间*声速(340m/s)/2
    STOP_TIME4();
    if((distance < 5) && ((Car_State == 2) || (Car_State == 1)))
    {
        if(Car_State == 1)
        {
            fristCarStatus = 1;
        }
        Car_State = 0;
        GPIO_ResetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15);//停止
        GPIO_ResetBits(GPIOC,GPIO_Pin_4);                       //红灯灭
        GPIO_ResetBits(GPIOA,GPIO_Pin_4);                       //蓝灯灭
        Direction_Data_Update();        //进行数据上传
    }
    if((distance >= 5) && (fristCarStatus == 1))
    {
        fristCarStatus = 0;
        Car_State = 1;
        GPIO_SetBits(GPIOA,GPIO_Pin_4);             //蓝灯亮
        GPIO_ResetBits(GPIOC,GPIO_Pin_4);           //红灯灭
        GPIO_SetBits(GPIOD,GPIO_Pin_13|GPIO_Pin_15);        //前进
        GPIO_ResetBits(GPIOD,GPIO_Pin_12|GPIO_Pin_14);
        Direction_Data_Update();        //进行数据上传
    }
```

3.8.5. 实验步骤

第一步：硬件连接

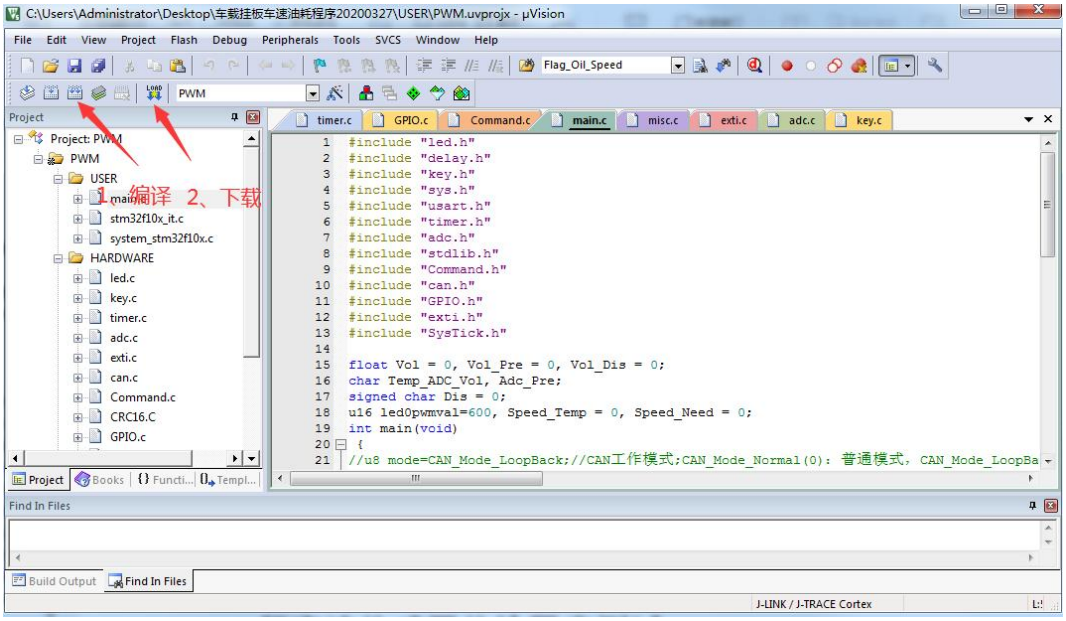
1. 将转动轮板子上的 6 芯排线连接到 CAN 测控终端的 6 芯防插反排座上；3 芯排线连接到 CAN 测控终端的 3 芯防插反排座上。

2. 使用 CAN 总线一端连接 CAN 测控终端的 CAN 端子排，另一端连接网关的 CAN 端子排。

第二步：编译烧写程序

1) CAN 测控终端车载单元工程采用 KEIL5 编译环境，使用方口 USB 数据线及 JLINK 连接线，插入 JTAG 烧录口。

2) 用 KEIL 5.14 打开车载挂板车速油耗程序例程（路径：…->车载挂板车速油耗程序->USER->PWM.uvprojx），点击重编译按钮完成编译，然后点击下载按钮完成程序的烧录，如下图所示。



程序烧录

第三步：网关添加节点

1.找到并打开数据采集应用程序  ，点击右上角的图标“  ”，展开下拉菜单，找到“添加 CAN”选项并点击，如图所示。

2.根据 CAN 协议，输入“CAN 行驶方向”的指令信息，如图所示。点击添加，将其添加到系统中。

3. 返回主界面，传输类型选择 CAN，就可以看到添加的 CAN 终端。

3.8.6. 实验结果

CAN 测控终端每隔 200ms 定时上传一次，网关数据采集应用程序接收到进行解析显示，可以解析转动速度、方向、超声波障碍物距离等，如图所示。

长按“CAN 方向”图标，弹出操作框，可以点击“采集/控制”选项，如图所示。

进入二级操作界面，可以点击“前进、后退、停止”按钮，控制车轮转动方向、转动和停止，如图所示。

3.9. 工业以太网网络

3.9.1. 工业以太网网络简介

工业控制网络的发展，基本趋势是逐渐向开放性、透明的通信协议。在工业控制领域，多台智能仪表信息交互传输的网络代替了传统的模拟信号用导线传输的方式，从用户到厂商都强烈要求形成统一的标准，这促进了现场总线技术的形成。但现场总线的开放性是有条件的、不彻底的。即使像工业 PC、OPC 等技术，

被镶嵌在传统的系统结构中，也只能是对系统的功能做些边缘性的提高。随着信息技术的不断飞跃，工业控制领域急需一种能够弥补工业现场总线、实现全系统统一、高效、实时的新的总线。工业以太网就是适应这一需求而迅速发展起来的。

所谓工业以太网，一般是指技术上与商用以太网（IEEE802.3 标准）兼容，但在产品设计时，在材质选用、产品强度、适用性及实时性、可互操作性、可靠性、抗干扰性和安全性等方面能满足工业现场的需求。

由于工业自动化网络控制系统不仅是一个完成数据传输的通信系统，而且是一个借助网络完成控制功能的自控系统。它除了完成数据传输之外，往往还需要依靠所传输的数据和指令，执行某些控制计算与操作功能。目前工业以太网主要还是用在控制级及其以上各级，测控现场仍然采用现场总线。

几种典型的工业以太网：

1）Ethernet/IP 实时以太网

2）Profinet 以太网。它是基于以太网的自动化标准。

3）EtherCAT 实时以太网。它使用标准的以太网物理层和常规的以太网卡，介质可为双绞线或光纤。

4）Modbus-IDA 实时以太网。

3.9.2. ModbusTCP 通信协议

实训台所有工业级传感器都通过手拉手方式连接，可以选择接到串口服务器的 485 总线接口上。串口服务器经过以太网与网关通信，网关作为主机，串口服务器设置为从机，此时网关与串口服务器之间采用的就是 Modbus-TCP 协议，获取传感器数值，控制执行器开关。

ModbusTCP 协议是在 ModbusRTU 协议基础上发展而来的。它是将 Modbus 协议嵌入到底层 TCP/IP 协议中构成的，这样就在 TCP/IP 的以太网上实现了客户机-服务器架构的 Modbus 报文通信。

- **Modbus-TCP 报文格式**

Modbus-TCP 与 Modbus-RTU 协议的数据帧结构如下表所示。

Modbus-RTU数据帧	地址码	功能码	数据	校验码
Modbus-TCP数据帧	MBPA报头	功能码	数据	

可以清楚地看到两者的区别。与 ModbusRTU 相比，ModbusTCP 数据帧不再有 CRC 校验，而校验的任务是由 TCP/IP 协议和以太网的链路层来完成的。另外，ModbusTCP 还加入了了 MBPA 报文头，由它来解释说明 modbus 的参数和功能。其他两部分可以通用。

ModbusTCP 详细报文格式如下表所示。

MBPA 报文头（起始字符组）				功能码	寄存器起始地址	寄存器长度
					数据	
传输标志	协议标志	长度	单元标志（地址码）			
2 字节	2 字节	2 字节	1 字节	1 字节	2 字节	字节

如果 TCP 协议转换为 RTU 协议，只需把 TCP 协议中 MBAP 头中的“单元标识域（即设备地址码）”和后续字节组成一帧，再后面加上此帧的 CRC 校验，就可以组成 RTU 协议，在串行链路上进行发送。

如果 RTU 协议转换为 TCP 协议，那么需要根据实际情况组建一个 MBAP 报头，就可以在以太网链路上进行发送。

关于 MBAP（Modbus Application Protocol, Modbus 应用协议）报文头详细说明如表所示。

域	长度	描述	客户端	服务器端
传输标志	2 字节	标志某个 modbus 请求/应答的传输	由客户端生成	应答时复制该值
协议标志	2 字节	0=modbus 协议 1=UNI-TE 协议	由客户端生成	应答时复制该值
长度	2 字节	后续字节计数	由客户端生成	应答时有服务器端重新生成
单元标志	1 字节	定义连接目的节点的设备地址	由客户端生成	应答时复制该值

- **Modbus-TCP 通信方式**

Modbus-TCP 请求报文举例

描述		大小(字节)	示例	备注
MBAP	传输标志 Hi	1	0x15	传输标志用于和应答配合使用
	传输标志 Lo	1	0x01	每对传输使用唯一的标志

	协议标志	2	0x0000	该域用作寻址 Modbus，包含目的设备地址
	长度	2	0x0006	
	单元标志	1	0x01	
Modbus 请求	功能码	1	0x03	读寄存器
	起始寄存器地址	2	0x0005	
	寄存器数量	2	0x0001	

1. 串口服务器设置为服务器模式时

数据报文结构

客户端请求：00 00 00 00 00 06 0F 04 00 01 00 01

客户端响应：00 00 00 00 00 05 0F 04 02 03 53

从左往右分析报文，

请求：

00 00：为此次通信事物处理标识符，一般每次通信之后将被要求加 1 以区别不同的通信数据报文;

00 00：标识协议标识符，00 00 位 modubs 协议；

00 06：数据长度，用来指示接下来数据的长度，单位字节；

0F：设备 modbus 地址，用来标识连接在串口线或网络上的远程服务器的的地址。以上七个字节也被称为 modbus 报文头；

04：功能码，04 代表去读多通道采集控制模块的模拟量通道；

00 01：为寄存器的起始地址；

00 01：为要读取的寄存器的数量；

响应：

00 00 为此次通信事务处理标识符，应答报文要求与先前对应的请求保持一致;

00 00 为协议标识符，与先前对应的请求保持一致;

00 05 为数据长度，用来指示接下来数据的长度，单位字节;

0F 为设备地址，应答报文要求与先前对应的请求保持一致；

04 为功能码，正常情况下应答报文要求与先前对应的请求保持一致，如果出错则返回 80h+先前的功能码；

02 指示接下来数据的字节长度;

03 53 为被读取的保持寄存器中的数据值，即要求被读取的地址为 00 00 的

保持寄存器中的数值为 1234h

2. 串口服务器设置为客户端模式时

数据报文结构

服务器请求：00 00 00 00 00 06 0F 04 00 01 00 01

服务器响应：00 01 00 00 00 05 0F 04 02 03 53

服务器请求：FF FF 00 00 00 06 0F 04 00 01 00 01

服务器响应：00 00 00 00 00 05 0F 04 02 03 53

仅对高两个字节做说明：

在客户端模式下，应答报文，总对请求报文的“通信事务处理标识符”自动加 1。

3.10. 无线数据通信

3.10.1. 无线数据通信概述

无线通信(Wireless Communication)是利用电磁波信号可以在自由空间中传播的特性进行信息交换的一种通信方式，近些年信息通信领域中，发展最快、应用最广的就是无线通信技术。在移动中实现的无线通信又通称为移动通信，人们把二者合称为无线移动通信。

无线技术给人们带来的影响是无可争议的。如今每一天大约有 15 万人成为新的无线用户，全球范围内的无线用户数量现今已经超过 2 亿。这些人包括大学教授、仓库管理员、护士、商店负责人、办公室经理和卡车司机。他们使用无线技术的方式和他们自身的工作一样都在不断地更新。这一应用已深入到人们生活和工作的各个方面，包括日常使用的手机、无线电话等，其中 4G、LORA、WLAN、ZIGBEE、蓝牙、宽带卫星系统、数字电视都是 21 世纪最热门的无线通信技术的应用。

通俗的说无线数据通信就是无线终端与中心主机或无线终端之间采用无线连接，通过无线方式传输数据。

无线数据通信是指无线终端与中心主机或无线终端之间采用无线连接，通过无线方式传输数据。

3.10.1.1. ZigBee 通信概述

ZigBee，也称紫蜂，是一种低速短距离传输的无线网上协议，底层是采用IEEE 802.15.4 标准规范的媒体访问层与物理层。主要特色有低速、低功耗、低成本、支持大量网上节点、支持多种网上拓扑、低复杂度、快速、可靠、安全。

ZigBee 是一项新型的无线通信技术，适用于传输范围短数据传输速率低的一系列电子元器件设备之间。 ZigBee 无线通信技术可于数以千计的微小传感器相互间，依托专门的无线电标准达成相互协调通信，因而该项技术常被称为Home RF Lite 无线技术、FireFly 无线技术。ZigBee 无线通信技术还可应用于小范围的基于无线通信的控制及自动化等领域，可省去计算机设备、一系列数字设备相互间的有线电缆，更能够实现多种不同数字设备相互间的无线组网，使它们实现相互通信，或者接入因特网。

① 低功耗。

在低耗电待机模式下，2 节 5 号干电池可支持 1 个节点工作 6～24 个月，甚至更长。这是 ZigBee 的突出优势。相比较，蓝牙能工作数周、WiFi 可工作数小时。

TI 公司和德国的 Micropelt 公司共同推出新能源的 ZigBee 节点。该节点采用 Micropelt 公司的热电发电机给 TI 公司的 ZigBee 提供电源。

②低成本。

通过大幅简化协议(不到蓝牙的 1/10)，降低了对通信控制器的要求，按预测分析，以 8051 的 8 位微控制器测算，全功能的主节点需要 32KB 代码，子功能节点少至 4KB 代码，而且 ZigBee 免协议专利费。每块芯片的价格大约为 2 美元。

③低速率。

ZigBee 工作在 20～250kbps 的速率，分别提供 250 kbps(2.4GHz)、40kbps(915 MHz)和 20kbps(868 MHz)的原始数据吞吐率，满足低速率传输数据的应用需求。

④近距离。

传输范围一般介于 10～100m 之间，在增加发射功率后，亦可增加到 1～3km。

这指的是相邻节点间的距离。如果通过路由和节点间通信的接力，传输距离将可以更远。

⑤短时延。

ZigBee 的响应速度较快，一般从睡眠转入工作状态只需 15ms，节点连接进入网络只需 30ms，进一步节省了电能。相比较，蓝牙需要 3～10s、WiFi 需要 3 s。

⑥高容量。

ZigBee 可采用星状、片状和网状网络结构，由一个主节点管理若干子节点，最多一个主节点可管理 254 个子节点；同时主节点还可由上一层网络节点管理，最多可组成 65000 个节点的大网。

⑦高安全。

ZigBee 提供了三级安全模式，包括安全设定、使用访问控制清单(Access Control List, ACL) 防止非法获取数据以及采用高级加密标准(AES 128)的对称密码，以灵活确定其安全属性。

⑧免执照频段。

使用工业科学医疗(ISM)频段，915MHz(美国), 868MHz(欧洲), 2.4GHz(全球) 。

ZigBee 此前被称作“HomeRF Lite”或“FireFly”无线技术，主要用于近距离无线连接。它有自己的无线电标准，在数千个微小的传感器之间相互协调实现通信。这些传感器只需要很低的功耗，以接力的方式通过无线电波将数据从一个传感器传到另一个传感器，因此它们的通信效率非常高。最后，这些数据就可以进入计算机用 于分析或者被另外一种无线技术如 wiMax 收集。 ZigBee 的目标市场主要有 PC 外设(鼠标、键盘、游戏操控杆)、消费类电子设备(TV、VCR、CD、VCD、DVD 等设备上的遥控装置)、家庭内智能控制(照明、煤气计量控制及报警等)、玩具(电子宠物)、医护(监视器和传感器)、工控(监视器、传感器和自动控制设备)等非常广阔的领域。

ZigBee 技术的先天性优势，使得它在物联网行业逐渐成为一个主流技术，在工业、农业、智能 家居等领域得到大规模的应用。例如，它可用于厂房内进行设备控制、采集粉尘和有毒气体等数据；在农业，可以实现温湿度、PH 值等

数据的采集并根据数据分析的结果进行灌溉、通风等联动动作；在矿井，可实现环境检测、语音通讯和人员位置定位等功能。

3.10.1.2. 蓝牙通信概述

蓝牙技术是一种无线数据与语音通信的开放性全球规范，它以低成本的近距离无线连接为基础，为固定与移动设备通信环境建立一个特别连接。其程序写在一个 9 x 9 mm 的微芯片中。

例如，如果把蓝牙技术引入到移动电话和膝上型电脑中，就可以去掉移动电话与膝上型电脑之间的令人讨厌的连接电缆而通过无线使其建立通信。打印机、PDA、桌上型电脑、传真机、键盘、游戏操纵杆以及所有其它的数字设备都可以成为蓝牙系统的一部分。除此之外，蓝牙无线技术还为已存在的数字网络和外设提供通用接口以组建一个远离固定网络的个人特别连接设备群。

蓝牙工作在全球通用的 2.4GHz ISM(即工业、科学、医学）频段。蓝牙数据速率为 1Mb/s，时分双工传输方案，使用 IEEE802.15 协议。

蓝牙通信的主从关系

蓝牙技术规定每一对设备之间进行蓝牙通讯时，必须一个为主角色，另一为从角色，才能进行通信，通信时必须由主端进行查找，发起配对，连接成功后，双方即可收发数据。理论上一个蓝牙主端设备，可同时与 7 个蓝牙从端设备进行通讯。一个具备蓝牙通讯功能的设备，可以在两个角色间切换，平时工作在从模式，等待其它主设备来连接，需要时转换为主模式，向其它设备发起呼叫。一个蓝牙设备以主模式发起呼叫时，需要知道对方的蓝牙地址，配对密码等信息，配对完成后，可直接发起呼叫。

蓝牙的呼叫过程

蓝牙主端设备发起呼叫，首先是查找，找出周围处于可被查找的蓝牙设备。主端设备找到从端蓝牙设备后，与从端蓝牙设备进行配对，此时需要输入从端设备的 PIN 码，也有设备不需要输入 PIN 码。配对完成后，从端蓝牙设备会记录主端设备的信任信息，此时主端即可向从端设备发起呼叫，已配对的设备在下次呼叫时，不再需要重新配对。已配对的设备，做为从端的蓝牙耳机也可以发起建链请求，但做数据通讯的蓝牙模块一般不发起呼叫。链路建立成功后，主从两端

之间即可进行双向的数据或语音通讯。在通信状态下，主端和从端设备都可以发起断链，断开蓝牙链路。

蓝牙一对一的串口数据传输应用

蓝牙数据传输应用中，一对一串口数据通讯是最常见的应用之一，蓝牙设备在出厂前即提前设好两个蓝牙设备之间的配对信息，主端预存有从端设备的PIN 码、地址等，两端设备加电即自动建链，透明串口传输，无需外围电路干预。一对一应用中从端设备可以设为两种类型，一是静默状态，即只能与指定的主端通信，不被别的蓝牙设备查找；二是开发状态，既可被指定主端查找，也可以被别的蓝牙设备查找建链。

3.10.1.3. WiFi 通信概述

WiFi（Wireless Fidelity），俗称无线宽带，是 IEEE 定义的一个无线网络通信的工业标准。随着技术的发展，以及 IEEE802.11a 及 IEEE 802.11g 等标准的出现，现在 IEEE802.11 这个标准已被统称作 WiFi。WiFi 工作在 2.4Ghz 频段，最高传输速率达 11Mbps。WiFi 技术与蓝牙技术一样，同属于在办公室或家庭中使用的无线局域网通信技术。其本质的特点是不再使用电缆将计算机与网络连接起来，而是通过无线的方式连接，从而使网络的构建和终端的移动更加灵活，不受约束。

WiFi 局域网由一个 AP（Access Point）和若干个无线网卡（无线客户端）组成。AP 一般称为网络接入点，既有普通站点的身份，又有接入到分配系统的功能。它是当做传统的有线局域网与无线局域网之间的桥梁。AP 每隔 100ms 将 SSID（Service Set Identifier）经信号台分组广播一次，信号台分组的传输速率是 1 Mbit/s，并且长度相当得短，所以这个广播动作对网络性能的影响不大。因为 WIFI 规定的最低传输速率是 1 Mbit/s，所以可以确保所有 WIFI 客户端都能收到这个 SSID 广播分组，可以借此决定是否要和这一个 SSID 的 AP 连接。

3.10.1.4. LoRa 通信概述

LoRa，英文 Long Range 的缩写，为低功耗广域网（Low PowerWide Area Network,LPWAN）通信技术的一种。在 LPWAN 产生之前，似乎只能在远距离

和低功耗两者之间取舍一种。而 LoRa 无线技术的出现，不仅可以实现远距离传输，并且间距低功耗、低成本的优点。

在 LoRa 网络中，每个节点并不会彼此连接，需先连接至网关后，才能连到中央主机，或是透过中央主机，将数据传到另一个节点。终端节点的信息，可以同时传给多个网关，信息也可以透过网关之间的桥接，进一步延伸传输距离。

LoRa 网络有以下特点和目标：

- 大范围覆盖（5--10 公里）
- 抗干扰
- 数据率从 300bps 到 50kbps 不等
- 低能耗（电池驱动 10 年）
- 双向通信
- 高网络容量 -- 单个网关可以支持数万个终端节点。

3.10.2. ZigBee 无线通信实验

3.10.2.1. 实验目的

熟悉实训台 ZigBee 节点的结构，供电电源、按键、指示灯的作用。

理解 ZigBee 无线传感网的组网原理，掌握 ZigBee 数据传输帧格式，以及“Zigbee 传感器透明传输综合应用程序”的使用方法。

理解协调器与上位机通信的方法，会使用上位机应用程序进行 ZigBee 通信测试、传感器数据采集及控制。

3.10.2.2. 实验环境

硬件：智能网关（含 ZigBee 协调器模块）、ZigBee 温湿度节点，ZigBee LED 蜂鸣器节点，CC2530 Debugger 仿真器，方口 USB 数据线等；

		
网关	ZigBee 温湿度节点	ZigBee LED 蜂鸣器

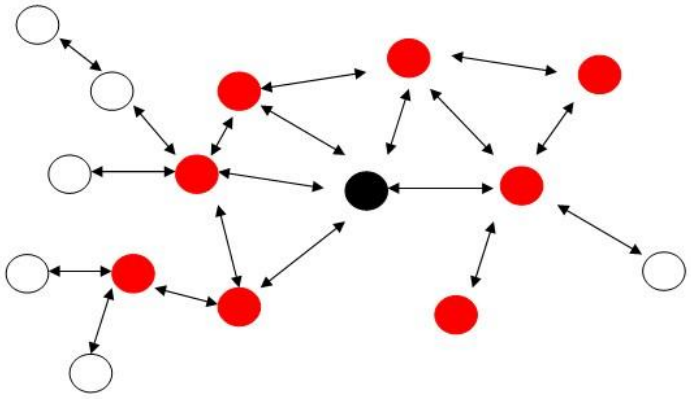
软件：“Zigbee 传感器透明传输综合应用程序”、SmartRF Flash Programmer 软件，网关数据采集应用程序。

3.10.2.3. 实验原理

1. ZigBee 组网的基本原理

1）ZigBee 逻辑设备类型

ZigBee 网络中存在三种逻辑设备类型：协调器（Coordinator）、路由器（Router）、终端设备（EndDevice）。



黑色节点为协调器，灰色（红色）节点为路由器，白色节点为终端设备 。.

（1）协调器：是整个网络的核心，它最主要的作用是启动网络，其方法是选择一个相对空闲的信道以及一个 PANID，然后启动网络。它会协助建立网络中的安全层以及处理应用层的绑定。当整个网络启动和配置完成后，它的功能退化为一个普通路由器。

（2）路由器：一般情况下，路由器应该一直处于活动状态，不应该休眠，它主要提供接力作用，能扩展信号的传输范围。 并且允许其他设备加入网络；具有多条路由；能协助它的电池供电子终端设备通信 。

（3）设备终端：终端设备没有维护网络基础结构的职责，它可以选择睡眠或唤醒。因此，它可以作为一个电池供电节点。一般来说，一个终端设备的存储

需求（特别是 RAM）是比较少的。

2） ZIGBEE 信道

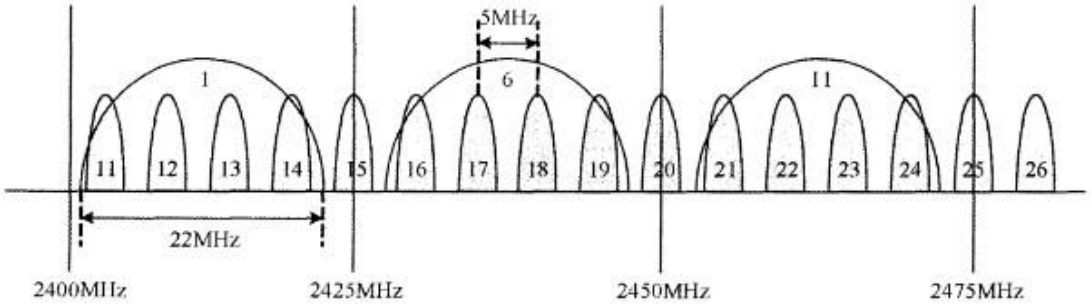
ZigBee 网络所使用的 2.4GHz 射频段被划分为 16 个独立的信道。这 16 个独立的信道编号从 11 到 26，对应的中心频率从 2405MHZ,按照每 5MHZ 递增至 2480MHz。Zigbee 的带宽比较宽。在 Z-Stack 中，每一种设备类型都有一个默认的信道集 DEFAULT_CHANLIST 在“f8wConfig.cfg”文件中定义，该文件主要包含 ZigBee 网络的一些配置参数。

信道的分类：

（1）专用信道：15、20、25、26，zigbee 使用这四个信道，用于信标帧的广播，避免被较大功率的 IEEE 802.11b 系统干扰，这 4 个信道与 IEEE802.11b 工作信道不重合；

（2）一般信道：专用信道外的信道；

Zigbee 信道与 wifi 信道的比较：



3）个域网络标识符 PANID

个域网标识符 PANID 用于区分不同的 ZigBee 网络。该值在“f8wConfig.cfg”文件中定义。使用时，有以下几种情况需要注意：

（1）如果协调器的 ZDAPP_CONFIG_PAN_ID 值设置为 0xFFFF，则协调器将产生一个随机的 PANID。如果路由器或终端节点的 ZDAPP_CONFIG_PAN_ID 值也设置为 0xFFFF，那么路由器或终端节点将会在自己的默认信道上随机选择一个干扰较少、较纯净的网络加入，网络协调器的 PANID 即为自己的 PANID。

这种情况会导致，当前使用的传感器节点可能会添加到其他协调器组建的网络中。

（2）如果协调器的 ZDAPP_CONFIG_PAN_ID 值设置为非 0xFFFF 值，则协调器建立的网络标识符就是 ZDAPP_CONFIG_PAN_ID 的值。如果路由器或终端

节点的 ZDAPP_CONFIG_PAN_ID 值也设置为与协调器的 PANID 值相同，那么就会自动加入 PANID 相同的协调器网络中，而不论当前信道是否存在干扰。

（3）如果在默认信道上已经有该 PANID 值的网络存在，则协调器不会在邻近的空间内建立一个有相同 PANID 的无线网络，它会在设置的 PANID 值的基础上加 1，继续搜索新 PANID，直到找到网络不冲突为止。

这样，就有可能产生一些问题：如果协调器因为在默认信道上发生 PANID 冲突而更换 PANID（例如路由器上电状态，而协调器掉电后重新上电，那么协调器新建网络的 PANID 就会自动加 1），而终端节点并不知道协调器已经更换了 PANID，还是继续加入到 PANID 为 ZDAPP_CONFIG_PAN_ID 值的网络中，那么就会造成终端节点没有添加到我们需要的网络中。若想恢复节点自动添加到本地的协调器中，只需保证协调器、路由器、终端节点都掉电，确保协调器先上电，其他节点再上电，或者三种设备同时重启。

4） 地址

每个 ZigBee 设备都有一个 64 位，8 个字节的 IEEE 长地址，即 MAC 地址，跟网卡 MAC 一样，它是全球唯一的，但在实际网络中，为了方便，通常用 16 位，2 字节的短地址来标识自身和识别对方，也称为网络地址。对于协调器来说，短地址为 0x0000，对于路由器和终端来说，短地址是由它们所在网络中的协调器分配的。 关于地址的数据结构：

(1) shortaddr:短地址

(2) extAddr:长地址

(3) addrMode:单播，组播或广播

AddrNotPresent=0,//按照绑定方式传输

AddrGroup=1,//组播；Addr16Bit=2,//指定目标网络地址进行单播传输;Addr64Bit=3,//指定 IEEE 地址单播传输；

AddrBroadcast=15,//广播传输.

(4) endPoint：通信端口

(5) panId: 网络 ID 号

5）组网原理

（1）若协调器、路由器、终端设备具有相同的通信信道和网络标识符 PANID

（如信道均为 18 号信道，网络 ID 均为 0x1212），保持协调器先上电，确保 zigbee 网络建立完成。此时路由器、终端设备再上电，那么这些设备会自动添加到已经建立成功的 ZigBee 网络中。

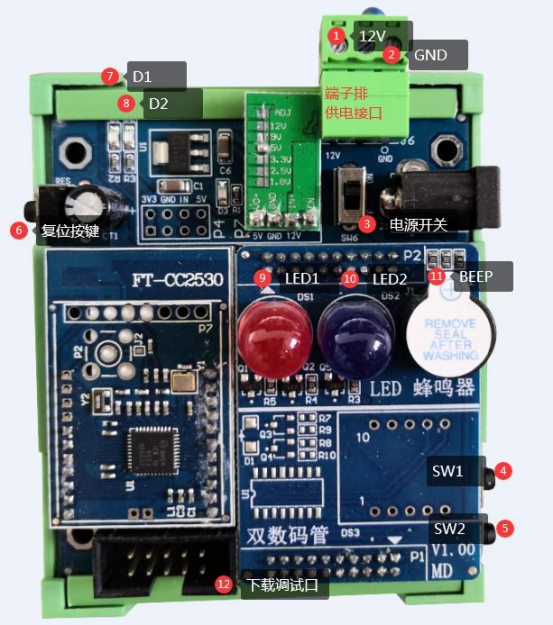
也可以确保协调器、路由器、终端节点同时上电，那么可以自动按照上次组网信息重新建网、组网。

（2）若协调器、路由器、终端具有相同的通信信道，但网络标识符 PANID 均设置的是 0XFFFF，那么协调器上电后建立一个随机生成的 PANID 的网络。路由器或终端会自动扫描寻找信号优质的网络随机加入。

本实验中，为了确保当前实训台的节点不会添加到别的实训台的 ZigBee 网络中，我们使用第（1）种网络参数设置方法。

2. ZigBee 节点的结构

该实训台 ZigBee 节点由 ZigBee 通信模块、传感器模块、及接口底板组成，采用 DC12V 供电。



按键 SW1：按下一次，触发一次数据上报；

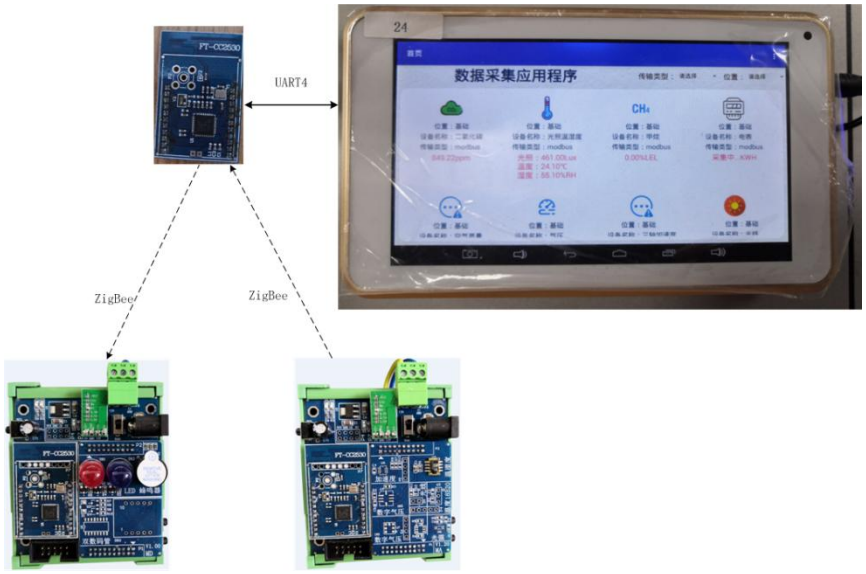
按键 SW2：按下一次后，启动定时 10s 触发一次数据上报；

D1 和 D2 指示灯： D1 灯闪烁 3 次后，D2 灯紧接着闪烁 1 次，同时协调器模块所在网关的 LED8 灯也闪烁 1 次，说明路由器或终端成功添加到网络中。

3. 本系统 Zigbee 网络拓扑

网关背面的 ZigBee 通信模块为 ZigBee 协调器，ZigBee 温湿度节点、ZigBe

e LED 蜂鸣器节点为 ZigBee 路由器。当设置为相同信道、相同 PANID 后，这些节点就可以自组网。因此，每个实训台的 ZigBee 网络设置相同的信道和 PANID，PANID 可以根据实训台组号定义，可以防止网络冲突。网络拓扑如图所示。



ZigBee 节点与协调器自组网，无线通信；协调器与网关通过 UART 有线通信。

4. ZigBee 数据格式

（1）查询信道和 PANID 的命令

上位机可通过串口向协调器发送命令，查询当前协调器组建的网络信道、PANID。

特殊设置命令							
发送信道和 PAN_ID 命令	CCH	BBH	BBH	DDH			
获得信道和 PAN_ID 命令应答	CCH	1AH	BBH	01H	02H	DDH	1AH: 信道 26 01H02H : PAN_ID

（2）传感器节点上传到协调器

通用格式，如下表所示。

传感器上传至协调器的数据格式							
标志	长度	父节点地址	本节点地址	类型	数据	校验和	IEEE 地址
FDH	03H	00H 00H	05H 09H	45H	...H	98H	01H 02H 03H 04H 05H 06H 07H 08H
数据标志	类型+数据+校验和+IEEE	父节点短地址	本身的短地址	传感器类型	传感器数据	(类型+数据)/256	Zigbee 的 IEEE 地址（在最后 8 个字节）

	地址						
1 字节	1 字节	2 字节	2 字节	1 字节	N 字节	1 字节	8 字节

本系统中使用的传感器、执行器的类型为：

0x45：ZigBee 温湿度节点；

0x40：LEDBEEP 节点

0x25：FTlink 遥控器节点

本小节使用的温湿度传感器节点的数据格式，示例如下：

温湿度节点上报给协调器的数据格式							
FDH	0EH	00H 00H	C7H FBH	45H	43 3B 11 30	14H	33 D2 6F 02 00 4B 12 00

温湿度数据字段占 4 个字节，前两个是湿度，后两个是温度。湿度的两个字节，高位字节是湿度的整数部分，低位字节是湿度的小数部分，计算方法：

湿度值：43H=67D，3BH =59D，所以湿度=67.59%RH；

温度值：11H=17D，30H=48D，所以温度=17.48℃。

（3）协调器控制执行器节点的数据格式

不同的执行器，控制命令有所不同，本小节实验以 LEDBEEP 节点为例。

按下 LEDBEEP 节点的 SW1 或 SW2，协调器收到 LEDBEEP 上传的数据格式，就可以获取 IEEE 地址，组成控制帧，对其实现控制。控制命令格式如下：

向控制节点发送命令格式							
标志	长度	节点地址	本机	类型	节点地址	组合命令	长地址
FDH	0EH	X1H x2H	00H 00H	XXH	X1H x2H	N1H n2H N3H	01H 02H 03H 04H 05H 06H 07H 08H
默认	类型+节点地址+组合命令+长地址	控制节点的短地址	默认	传感器类型	控制节点的短地址	组合控制命令	节点 IEEE 地址

控制节点接收到控制命令后返回的应答格式							
标志	长度	父节点	本节点	类型	数据	校验	长地址
FDH	0DH	00H 00H	08H 79H	XXH	BBH BBH BBH	00H	01H 02H 03H 04H 05H 06H 07H 08H
FDH 终端 FAH	类型+数据+校验+长地址	父节点短地址	本节点短地址	传感器类型	默认		节点 IEEE 地址

路由							

本系统中使用的执行器的类型为：

0x40：LEDBEEP 节点

这个被控设备的控制命令，参考下表。

受控节点，受控采集，组合命令说明				
传感器类型	组合命令			控制说明
0x40（LEDBEEP 节点）	DDH	A1H	AAH	LED1（红灯）开
	DDH	A1H	BBH	LED1（红灯）关
	DDH	A2H	AAH	LED2（蓝灯）开
	DDH	A2H	AAH	LED2（蓝灯）关
	DDH	B1H	AAH	蜂鸣器开
	DDH	B1H	BBH	蜂鸣器关

以 LEDBEEP 执行器节点为例，协调器控制格式，示例如下：

协调器向 LEDBEEP 发送“LED1 开”命令							
FD	0E	C7 FB	00 00	4B	C7 FB	DD A1 AA	FD C0 6F 02 00 4B 12 00
协调器向 LEDBEEP 发送“LED1 关”命令							
FD	0E	C7 FB	00 00	4B	C7 FB	DD A1 BB	FD C0 6F 02 00 4B 12 00
协调器向 LEDBEEP 发送“LED2 开”命令							
FD	0E	C7 FB	00 00	4B	C7 FB	DD A2 AA	FD C0 6F 02 00 4B 12 00
协调器向 LEDBEEP 发送“LED2 关”命令							
FD	0E	C7 FB	00 00	4B	C7 FB	DD A2 BB	FD C0 6F 02 00 4B 12 00
协调器向 LEDBEEP 发送“BEEP 开”命令							
FD	0E	C7 FB	00 00	4B	C7 FB	DD B1 AA	FD C0 6F 02 00 4B 12 00
协调器向 LEDBEEP 发送“BEEP 关”命令							
FD	0E	C7 FB	00 00	4B	C7 FB	DD B1 BB	FD C0 6F 02 00 4B 12 00

这样组网成功后，网关数据采集应用程序就可以进行温湿度数据的无线采集测试、LED 和蜂鸣器的无线控制测试。

3.10.2.4. 实验内容

先通过网关查询 ZigBee 协调器组建的网络信道和 PANID，再编译“ZigBee 传感器透明传输应用程序”，设置 ZigBee 温湿度节点、LED 蜂鸣器节点与协调器具有相同的信道和 PANID，使其组网成功。然后使用网关数据采集应用程序

进行温湿度的无线数据采集显示，及 LED 蜂鸣器的无线控制。

1）关键代码分析：

（1）以 ZigBee LED 蜂鸣器节点为例，下面这段代码的作用是对数据帧的类型字段、帧头字段进行判断，如果是 0x40（ZigBee LED 蜂鸣器），则提取携带的数据字段，进行解析，控制两个 LED 点亮或熄灭，蜂鸣器蜂鸣或停止。

```
#if(SENSOR_TYPE  ==0X40)     //判断类型  LED 蜂鸣器
if((pkt->cmd.Data[0]==0xFD)&&(pkt->cmd.Data[6]==SENSOR_TYPE))
        {
#if  defined(  ZigBee_C_R_E_IEEE  )
   char  GetExtAddr=0;
   byte  *  Send_d;
   Send_d=NLME_GetExtAddr();
   for(int  i=0;i<8;i++)
   {
   if(pkt->cmd.Data[i+pkt->cmd.Data[1]-2]==(*Send_d++))
      GetExtAddr++;
   }
       if(GetExtAddr==8)
#else
       if((pkt->cmd.Data[7]==Send_data[4])&&(pkt->cmd.Data[8]==Send_data[5]))
#endif
        {T_MGSbit|=0X20;   //xx1x  xxx0  应答  正常  状态位
           if(pkt->cmd.Data[9]==0xDD)//
             {GenericApp_applicationbuf=1;
             if(pkt->cmd.Data[10]!=0XB1)T_MG=0X00;
        switch  (pkt->cmd.Data[10])
     {  case  0XA1:if(pkt->cmd.Data[11]==0xAA){  P1_3=0;  P1_4=1;}   //LED1  红色  点亮
                   if(pkt->cmd.Data[11]==0xBB){P1_3=1;P1_4=0;}   //LED1  红色  熄灭
                   break;
```

```
        case 0XA2:if(pkt->cmd.Data[11]==0xAA){P1_2=0; P1_7=1;}    //LED2 蓝色 点亮
                  if(pkt->cmd.Data[11]==0xBB){P1_2=1;P1_7=0;}    //LED2 蓝色 熄灭
                  break;
       case 0XA3: if(pkt->cmd.Data[11]==0xAA){T_MG=0X01;P1_2=1;P1_7=0;}    //LED2
熄灭 LED1 闪烁功能
                  if(pkt->cmd.Data[11]==0xBB){T_MG=0X00;P1_3=1;P1_4=0;}    //LED1
停止闪烁熄灭
                  break;
       case 0XA4:if(pkt->cmd.Data[11]==0xAA){P1_2=1; P1_7=0; P1_3=0; P1_4=1; } //LE
D1 打开，LED2 关闭功能
                  if(pkt->cmd.Data[11]==0xBB){P1_2=1; P1_7=0; P1_3=1; P1_4=0;}//LED
1，LED2 关闭
                  break;
       case 0XA5:if(pkt->cmd.Data[11]==0xAA){P1_2=0; P1_7=1; P1_3=1; P1_4=0; } //LE
D2 打开，LED1 关闭功能
                  if(pkt->cmd.Data[11]==0xBB){P1_2=1; P1_7=0; P1_3=1; P1_4=0;}//LED
1，LED2 关闭
                  break;
       case 0XB1:if(pkt->cmd.Data[11]==0xAA){ P1_1=0; P1_0=1;  }    //开蜂鸣器
                    if(pkt->cmd.Data[11]==0xBB){ P1_1=1;P1_0=0;}    //关蜂鸣器
                  break;
        default: break;}GenericApp_SendTheMessage();
          }
       }
}
```

3.10.2.5. 实验步骤

第一步：查询 ZigBee 网络信息

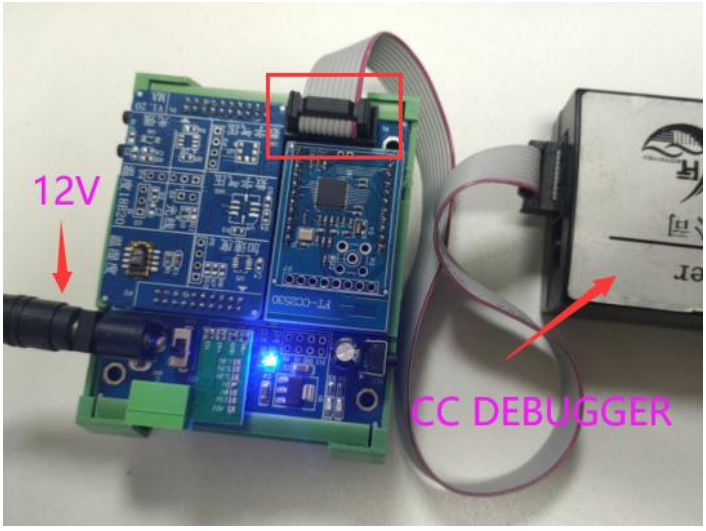
找到并打开数据采集应用程序  ，点击右上角的图标“  ”，展开下拉

菜单，找到“系统设置”选项并点击，如图所示。

在弹出的“系统设置”界面中，点击“查询信道”按钮，可以看到 ZigBee 协调器组建的网络信息（信道 26（十进制），PANID 为 0x6016（16 进制）），如图所示。

第二步：硬件连接

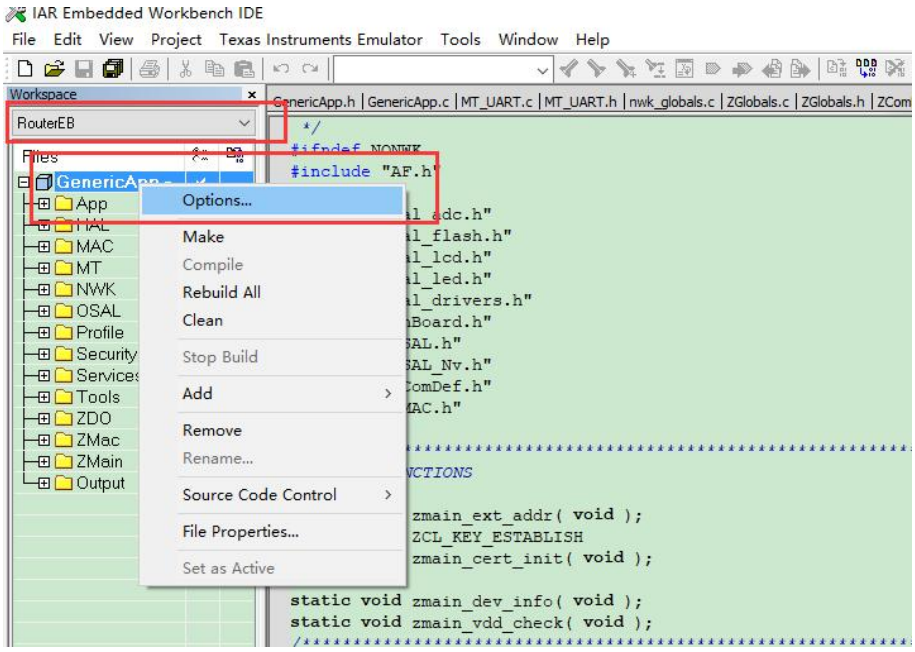
使用 USB 方口数据线连接 CC2530 Debugger 仿真器和 PC 机，用 10 针排线连接 ZigBee 温湿度传感器节点板，然后给 ZIGBEE 传感器节点板接通 12V 电源，如图所示。ZigBee LED 蜂鸣器的下载连接方式一样。

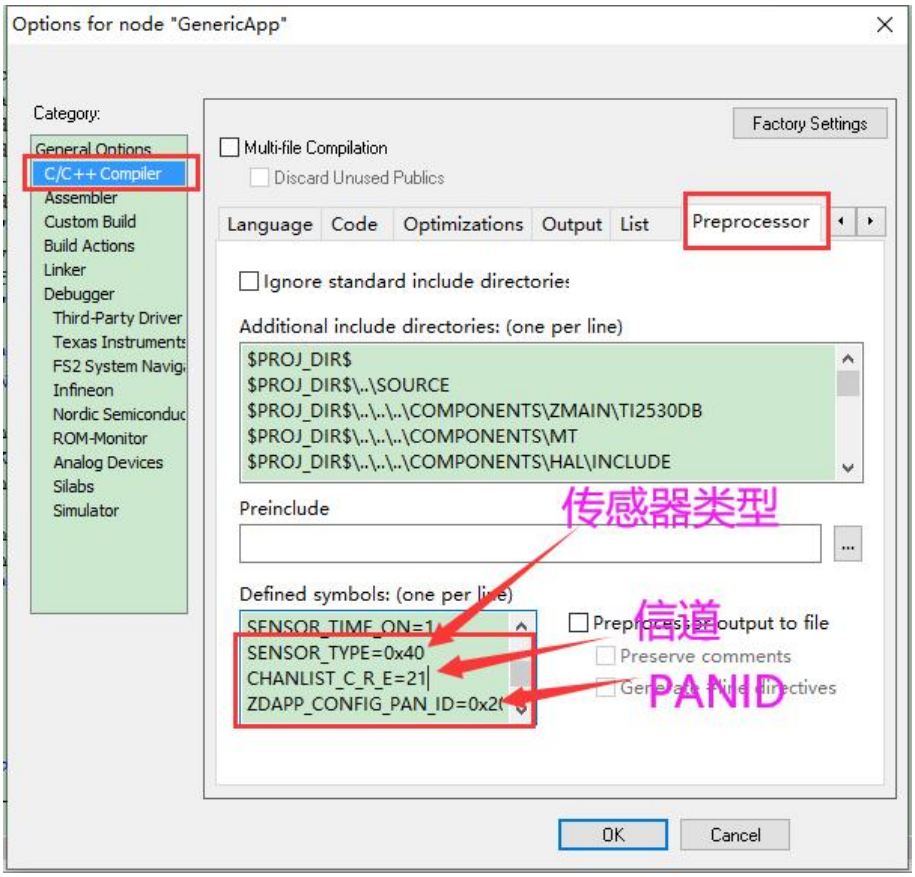


第三步：程序编译烧写

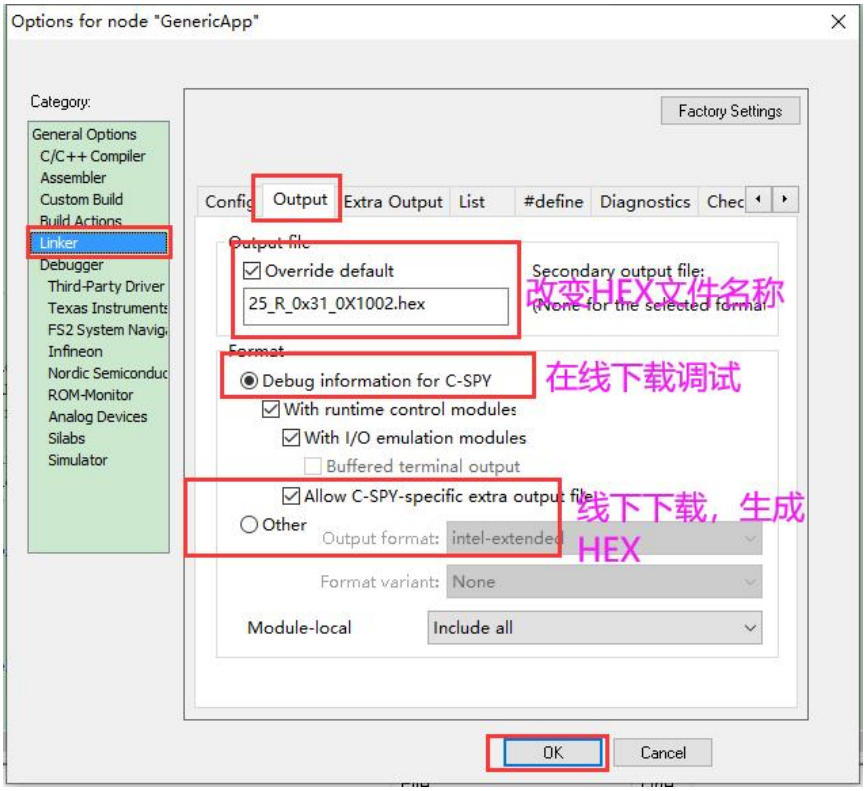
1.用 IAR Workbenc for 8051 开发环境打开实验例程“\ZStack 传感器透明传输源程序 V2.45145\Projects\GenericApp\CC2530DB”，点击 GenericApp.eww 打开工程。

2.节点设备类型选择 RouterEB(路由器)，右键单击 GenericApp，选择 options。选择 C/C++ Compiler----→Preprocessor，找到 SENSOR_TYPE（传感器类型编码，LED 蜂鸣器 0X40，温湿度'E'），CHANLIST_C_R_E（信道）和 ZDAPP_CONFIG_PAN_ID（PANID）根据查询协调器组建的网络信息，设定对应的值。



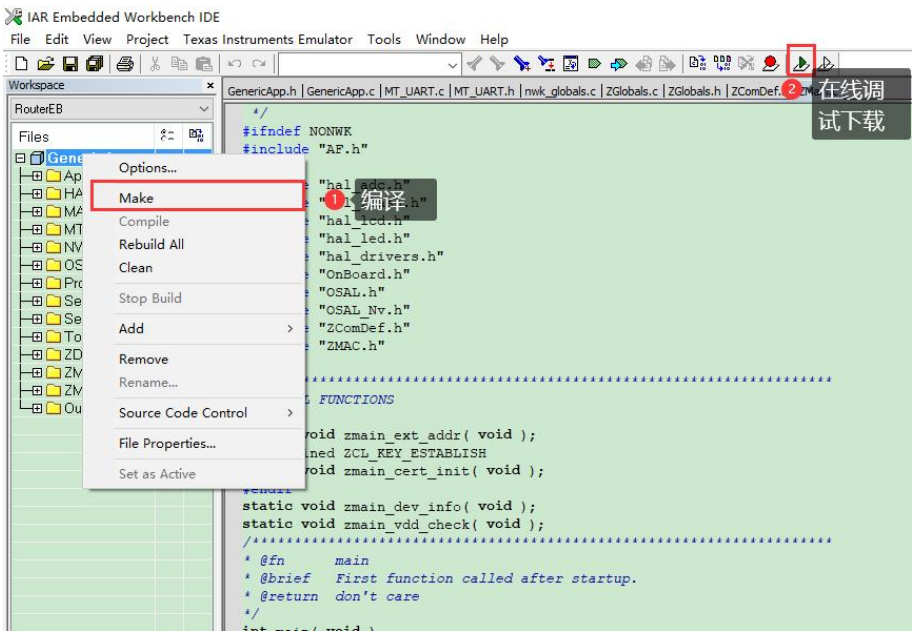


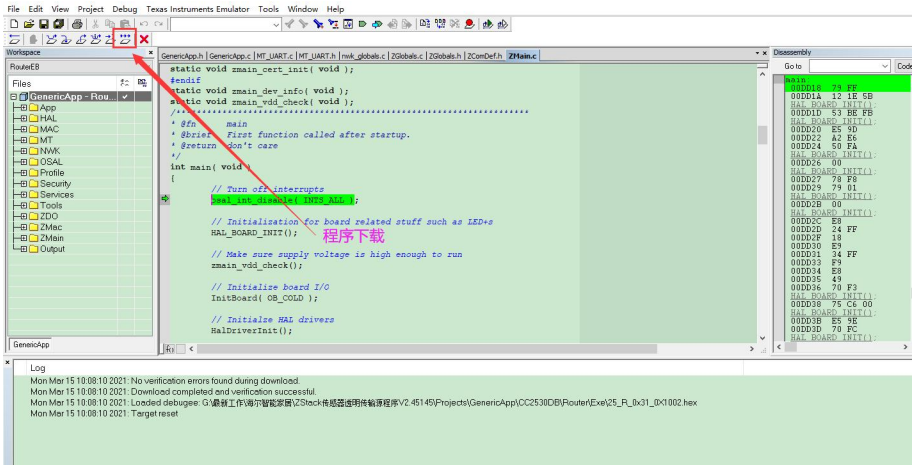
3.配置下载相关参数，选择 Linker。第一种选择在线调试，勾选 Debug information for C-SPY 点击 Ok 即可。第二种选择离线烧写，需要输入 HEX 文件名称，然后勾选 Other 点击 OK 即可。如下图所示：



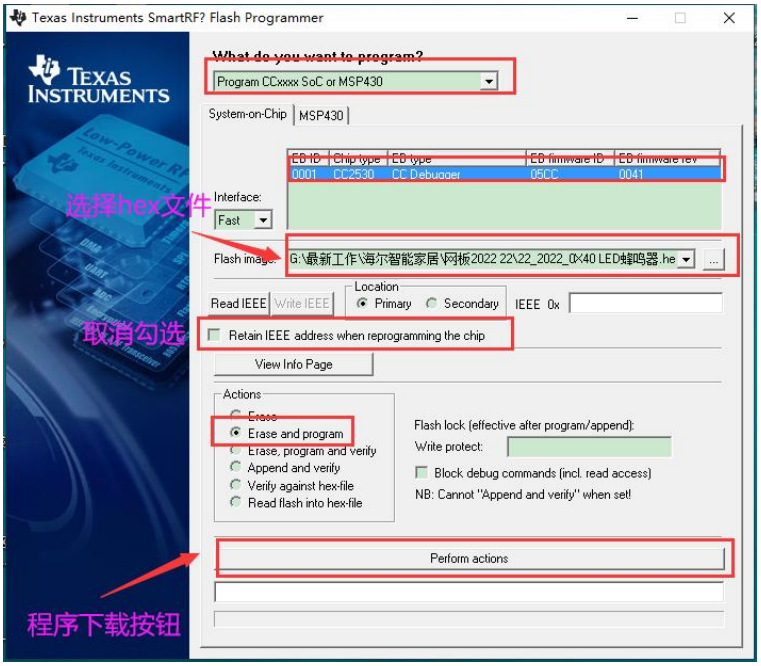
4.程序编译下载

在线调试下载：提前配置在线下载调试模式（上一步），右键单击 GenericApp，点击 MAKE。点击下载调试按钮，点击程序下载即可；如下图所示：





离线烧写：提前配置线下下载模式（上一步），右键单击 GenericApp，点击 MAKE。打开 SmartRF Flash Programmer，点击 CC DEBUGGER 按钮使指示灯变为绿色，配置使用如下图：



等待进程条结束，说明在线下载或离线烧写完成。

第四步：网关添加 ZigBee 节点

1. 网关数据采集应用程序返回主界面，点击右上角的图标“  ”，点击“添加 ZIGBEE”选项，弹出“添加 ZIGBEE 界面”。

2. 按下“ZigBee 温湿度节点”的 SW1 按钮（按键触发），每按下一次，就向协调器发送一次数据帧，“网关数据采集应用程序”接收解析并显示到对应的栏目，如图所示。

3. 然后点击添加按钮，将 ZigBee 温湿度节点温湿度节点添加到系统中。同样的方式添加 LED 蜂鸣器节点。

4. 添加完成后，返回主界面。菜单栏的传输类型选择“zigbee”，这样主界面只显示 zigbee 类型传输的传感器节点。

3.10.2.6. 实验结果

1. 针对 ZigBee 温湿度节点等采集类节点，可以按下 SW1 按键触发数据一次数据上传，也可以按下 SW2 按键使节点每隔 10s 定时上传，网关接收到数据帧后自动解析显示。控制类 ZigBee LED 蜂鸣器节点按下 SW1 按钮后，网关接收到数据帧自动解析显示。如图所示。

2. 选中任一个传感器节点图标，长按弹出下拉菜单，可以修改、删除节点，或者点击“采集与控制”选项，进入传感器节点的二级界面。

3. 针对温湿度节点，可以绘制温度、湿度的实时曲线，如图所示。

4. 针对“LED 蜂鸣器”节点，可以按钮控制 LED1 或 LED2 灯亮灭，或蜂鸣器蜂鸣或停止，如图所示。

ZigBee 无线通信实验完毕。

3.10.3. 蓝牙无线数据通信

3.10.3.1. 实验目的

熟悉实训台蓝牙节点的结构，供电电源、按键、指示灯的作用。

理解蓝牙无线传感网的组网原理，掌握蓝牙数据传输帧格式，以及“嵌入式 STM32 新协议节点程序”的使用方法。

理解蓝牙上位机通信的方法，会使用上位机应用程序进行蓝牙通信测试、传感器数据采集。

3.10.3.2. 实验环境

硬件：智能网关、蓝牙数字气压节点、蓝牙三轴加速度节点、JLINk 仿真器、USB 方口数据线等；

网关

蓝牙气压节点

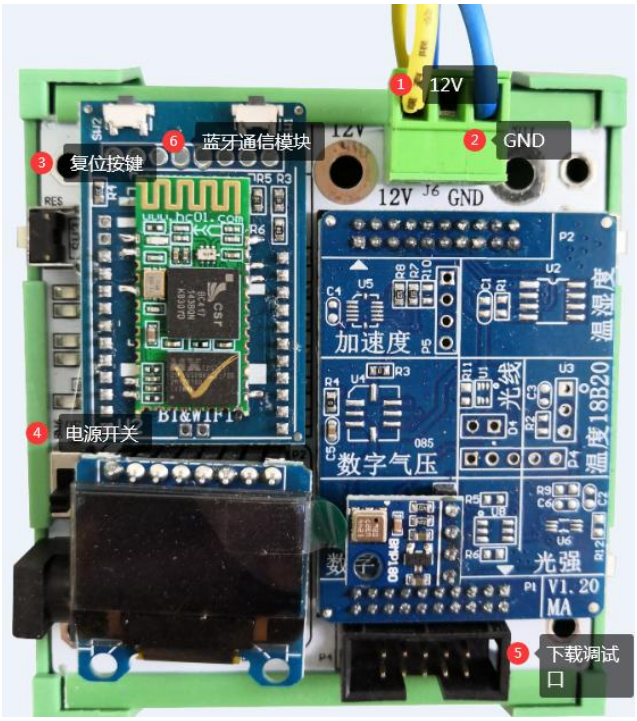
蓝牙三轴加速度

软件：keil4 开发环境、嵌入式新协议节点应用程序、数据采集应用程序。

3.10.3.3. 实验原理

1. 蓝牙传感器节点的结构

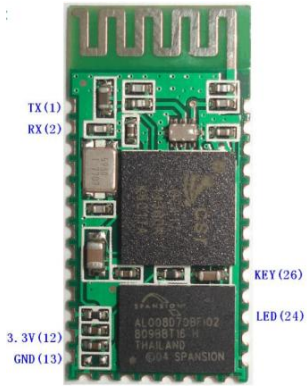
蓝牙传感器节点由蓝牙通信模块、传感器模块、嵌入式接口底板组成。



D5、D6 组网状态指示灯：当蓝牙节点没有配对连接成功时，节点上 D5 指示灯快速闪烁，D6 指示灯熄灭；当蓝牙节点配对连接成功后，节点上 D5 指示灯间隔闪烁，D6 指示灯常亮。

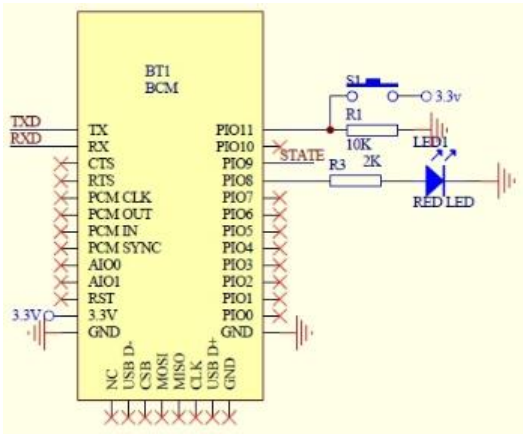
D1、D2 指示灯：不起作用。

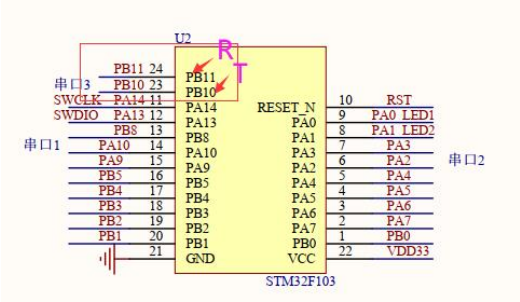
1）蓝牙通信模块，采用 HC-05 嵌入式蓝牙串口通讯模块（简称 HC-05）。



HC-05 模块的特点：

- 采用 CSR 主流蓝牙芯片，蓝牙 V2.0 协议标准；
- 供电电源：3.6-6V，不能超过 7V；
- 可连接状态指示灯，LED 快闪表示没有蓝牙连接；LED 慢闪表示进入 AT 命令模式；
- 未配对时，电流约 30mA；配对成功时，电流约 10mA;
- 具有两种工作模式：命令响应工作模式和自动连接工作模式。命令响应工作模式下，能执行所有 AT 命令，用户可向模块发送各种 AT 指令，为模块设定控制参数或发布控制命令。自动连接工作模式下，可分为主（Master）、从（Slave）和回环（Loopback）三种工作角色。当模块处于自动连接模式下时，可自动根据事先设定的连接方式进行数据传输。
- 模块管脚定义如下图所示。





串口模块用到的引脚定义：

- TXD：模块发送端，TTL 电平，必须与 MCU 的 RXD PB11 连接。
- RXD：模块接收端，TTL 电平，必须与 MCU 的 TXD PB10 相连。

- PIO8：可连接 LED，指示模块工作状态，模块上电后闪烁，不同的工作状态闪烁间隔不同。
- PIO9 ：STATE 管脚，可连接 LED，指示模块是否连接成功，蓝牙串口匹配连接成功后，LED 长亮。

2）蓝牙配置 AT 命令

AT+POLAR=0,0\r\n	设置 LED 指示驱动及连接状态输出极性 0 PIO8 低电平 LED 亮，PIO9 低电平连接成功
AT+UART=38400,0,0\r\n	设置波特率，停止位，校验位
AT+ROLE=0\r\n	设置模块角色 0 从角色
AT+NAME=\""DEV_NAME"\"\r\n	设置名称 DEV_NAME
AT+PSWD=\""DEV_PSWD"\"\r\n	设置配对码 DEV_PSWD 默认 1234（第一次）
AT+RESET\r\n	模块复位

2. 本系统蓝牙网络拓扑

网关板载蓝牙模块，蓝牙传感器节点直接与网关进行蓝牙通信，如图所示。

3. 蓝牙数据格式

蓝牙传感器节点通过该协议将采样信息上报给网关应用程序；网关应用程序可以向蓝牙传感器节点下发命令。

1）协议框架

3-1 协议框架

名称	协议开始	通信类型	指令类型	随机数	本地ID	目标ID	指令代码	参数长度	参数内容	校验码	协议结束
长度(Byte)	1	1	1	1	4	4	2	1	n	1	1
编码方式	FE	HEX	HEX	HEX	HEX	HEX	HEX	HEX	HEX	HEX	FF

上行：含义是指传感器节点向上位机发送数据。

下行：含义是指上位机向传感器节点下发命令。

协议框架各字段说明：

（1）协议开始：固定字符 0xFE，占 1 个字节，代表当前协议数据的开始；

（2）通信类型：占 1 个字节，16 进制表示，具体说明见下表。

比特 7	比特 6	比特 5	比特 4	比特 3	比特 2	比特 1	比特 0
00：有线通信 01：RF 通信 10：网络通信 11：保留		0：RF(433M); 1:RF(868M); 2:RF(2.4G); 3:ZIGBEE; 4:LORA; 5:... 20:WIFI; 21:BLT; 22:GPRS 23:3G; 24:4G; 25:NB-IOT; 30:RS232; 31:RS485; 32:CAN; 33:...					
BIT7,BIT6 代表通信种类；BIT5-BIT0 代表每种通信种类下的具体通信类型。							

（3）指令类型：用于标识“上下行”“加密类型”“回复方式”和“是否广播”，具体格式如下表所示。

比特 7	比特 6	比特 5	比特 4	比特 3	比特 2	比特 1	比特 0
0：下行 1：上行	0：长度 1Byte 1：长度 2Byte	00：无加密 01：加密类型 1		00：无回复 01：回复 ACK 10：回操作结果		0：广播 1：点对点	串口通信-转发
注：当加密类型为 1 时，对下行报文中的指令代码、操作口令、参数长度、参数内容进行加密，对上行报文中的指令代码、参数长度、参数内容进行加密；具体加密算法待商榷。BIT6 为 1 时，数据长度大于 255，数据长度 2byte，BIT6 为零时数据长度小于 255；数据长度 1byte；当 BIT0=1 时串口通信-转发，BIT0=0 时串口直接控制；串口通信-转发时对应的参数格式组包；串口直接控制时按对应的参数格式组包。							

（4）随机数：用于识别回码或者 ACK 与之前发送的哪条指令匹配。

（5）本地 ID：表示本机设备的 ID 编码；用于设备身份识别；

备注：在有线通信时，两个 ID 填写电路背面对应 RS485 通信地址（高字节补零），在无线通信时，两个 ID 填写对应的无线通信之间的产品 ID 编号；具体使用时，参见节点屏幕显示信息。具体 ID 分类详见本协议附录部分。

（6）目标 ID：表示目标设备的 ID 编码；用于设备身份识别；

备注：在有线通信时，两个 ID 填写电路背面对应 RS485 通信地址（高字节补零），在无线通信时，两个 ID 填写对应的无线通信之间的产品 ID 编号；具体 ID 分类详见本协议附录部分。

（7）指令代码：代表本条指令的具体功能（2 个字节，前一个字节代表指令类，后一个字节代表本类下的具体指令）。

（8）参数长度：参数内容的字节长度，如果是密文则包含补偿部分的长度；

（9）参数内容：具体指令对应的参数数据，详见具体下一小节各个传感器的指令解析。

（10）CRC 校验码：校验是从 FE 之后（第二字节开始）到校验位之前的内容校验，CRC 格式采用 CRC8 格式校验。

（11）协议结束符：0XFF，代表当前协议数据结束的标志。

注意：指令代码、参数长度、参数内容三个字段的内容随着传感器节点的类型不同而不同，下面重点说明。

2）传感器的指令代码部分说明

（1）蓝牙三轴加速度数据格式

根据协议框架，蓝牙三轴加速度完整的数据格式如下表所示。

名称	协议开始	通信类型	指令类型	随机数	本地 ID	目标 ID	指令代码	参数长度	参数内容	校验码	协议结束
长度（B）	1	1	1	1	4	4	2	1	n	1	1
编码方式	FE	83	80	00	11000003	00000000	0444	05	00000201C8	5A	FF

a. 三轴加速度传感器信息请求

下行指令代码：04 43

b. 三轴加速度传感器信息回复

上行指令代码：04 44

参数长度：05，5 个字节，分别是 1 个字节的设备电压、1 个字节的报警种类、3 个字节的传感器值，如表所示。

参数内容	设备电压	报警种类	传感值	保留

参数长度	1 字节	1 字节	3 字节	n 字节

三轴加速度的传感器值占用 3 个字节，从高字节到低字节，依次为 x、y、z 的加速度值，有符号浮点数。

（2）蓝牙数字气压数据格式

根据协议框架，蓝牙数字气压完整的数据格式如下表所示。

名称	协议开始	通信类型	指令类型	随机数	本地 ID	目标 ID	指令代码	参数长度	参数内容	校验码	协议结束
长度（B）	**1**	**1**	**1**	**1**	**4**	**4**	**2**	**1**	**n**	**1**	**1**
编码方式	**FE**	**A1**	**80**	**00**	**11000004**	**00000000**	**0404**	**05**	**0000f101C8**	**B9**	**FF**

a. 气压传感器信息请求

下行指令代码：04 03

b. 气压传感器信息回复

上行指令代码：04 04

参数长度：04，4 个字节，分别是 1 个字节的设备电压、1 个字节的报警种类、2 个字节的传感器值，如表所示。

参数内容	设备电压	报警种类	传感值	保留
参数长度	1 字节	1 字节	2 字节	n 字节

将传感器值的 2 个字节转换成十进制数。

气压传感器完整的协议包为：

fe a1 8000 11000003 00000000 0444 050000f1000eb9ff

3.10.3.4. 实验内容

程序流程：

第一步：初始化时钟，串口，定时器等内容；

第二步：通过发送 AT 指令配置蓝牙；

第三步：定时采集数据(修改程序参数设置定时上报时间)，整合数据帧通过蓝牙发送；

第四步：接收数据，判断行为指令进行相关动作；

关键代码：

（1）通过串口，发送 AT 命令配置蓝牙相关应用；

```
void hc05_init(void)
{
            UART3_Put_String("AT+POLAR=0,0\r\n"); delay_ms(500);IWDG_ReloadCounter();
UART_3_Data_Flage();                    //喂狗
          if((RxProData_buf1[0]=='O')&&(RxProData_buf1[1]=='K'))
       { RxProData_buf1[0]=0;
       UART3_Put_String("AT+UART=38400,0,0\r\n");delay_ms(500);IWDG_ReloadCounter();
UART_3_Data_Flage();                    //喂狗
            if((RxProData_buf1[0]=='O')&&(RxProData_buf1[1]=='K'))
       { RxProData_buf1[0]=0;
       UART3_Put_String("AT+ROLE=0\r\n");delay_ms(500);IWDG_ReloadCounter();
UART_3_Data_Flage();                    //喂狗
            if((RxProData_buf1[0]=='O')&&(RxProData_buf1[1]=='K'))
       { RxProData_buf1[0]=0;

UART3_Put_String("AT+NAME=\""DEV_NAME"\"\r\n");delay_ms(500);IWDG_ReloadCounte
r(); UART_3_Data_Flage();                    //喂狗
            if((RxProData_buf1[0]=='O')&&(RxProData_buf1[1]=='K'))
       { RxProData_buf1[0]=0;
UART3_Put_String("AT+PSWD=\""DEV_PSWD"\"\r\n");delay_ms(500);IWDG_ReloadCounter
(); UART_3_Data_Flage();                    //喂狗
            if((RxProData_buf1[0]=='O')&&(RxProData_buf1[1]=='K'))
       { RxProData_buf1[0]=0;
       UART3_Put_String("AT+RESET\r\n");delay_ms(50);IWDG_ReloadCounter();
UART_3_Data_Flage();                    //喂狗
            if((RxProData_buf1[0]=='O')&&(RxProData_buf1[1]=='K'))
       { RxProData_buf1[0]=0;
            LED_1_L();//透传模式
            BT_AT_L();//透传模式
            OLED_P8x16Str(110,0,"OK");
       }
       }
}}}}
}
```

（2）数字气压传感器代码解析：获取气压值，并上报数据！

```
static void Pressure_Poll(int tick)
{
if(time4_1s_on)
{time4_1s_on=0;
   //每 1s 读取一次气压值
   if ((tick % 1 == 0) &&(tick    != 32423)){
      Multiple_Read_BMP085(&p, &t);
            p/=10;
   }
   //定时上报数据
   if (tick % Sensor_UpData_time_Interval == 0) {
      if (report_enable == 0) {report_enable=1;}
      else {
            }
   }
}
if(time4_100ms_on)
```

```
{ time4_100ms_on=0;
      if(UART3_Time_Flage==0)//数据线,无数据
            {     if (Sensor_UpData_time_Interval) {
           sensor_misc.getValue();          //传感器数据采集
                  if((report_enable<2)&&(report_enable>0))report_enable++;
                  else report_enable=0;
                  if(report_enable)
                    { sensor_s_report(CMD_TYPE_NOT_ACK,0);        //发送数据
           }
      }
       }
}
}
```

（3）三轴加速度传感器代码解析：获取三轴加速度值，并上报数据！

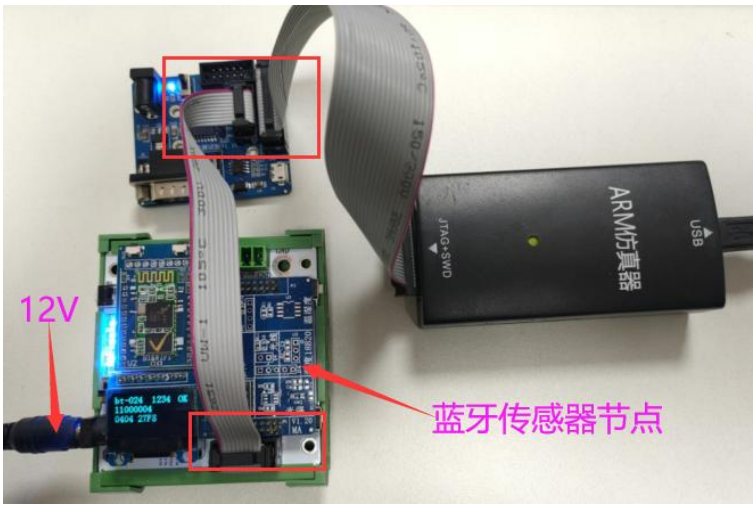
```
static void Hall_Poll(int tick)
{
 if(time4_1s_on)
{time4_1s_on=0;
      //每 1s 读取一次温湿度值
      if ((tick % 1 == 0) &&(tick    != 32423))
      {
    MMA7660_GetXYZ(&x, &y, &z);
 //定时上报数据
   if (tick % Sensor_UpData_time_Interval == 0) {
      if (report_enable == 0) {report_enable=1;}
      else {
             }
   }
}
}

if(time4_100ms_on)
{ time4_100ms_on=0;
      if(UART3_Time_Flage==0)//数据线,无数据
            {     if (Sensor_UpData_time_Interval) {
           sensor_misc.getValue();          //传感器数据采集
                  if((report_enable<2)&&(report_enable>0))report_enable++;
                  else report_enable=0;
                  if(report_enable)
                    { sensor_s_report(CMD_TYPE_NOT_ACK,0);
           }
      }
       }
}
}
```

3.10.3.5. 实验步骤

第一步：硬件连接（气压和三轴加速度一致）

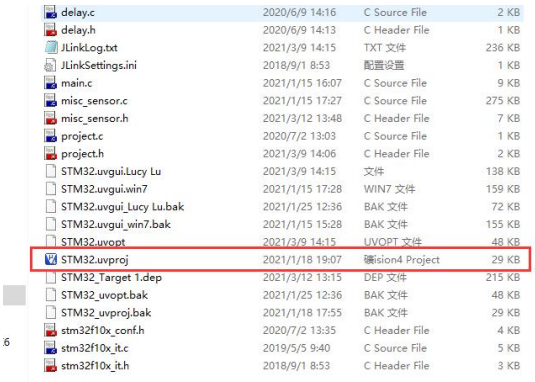
使用 USB 方口数据线连接 J-LINK 仿真器和 PC 机，用 20 针排线连接 J-LINK

仿真器和下载调试板，10 针排线连接下载调试板和蓝牙传感器节点板，然后给蓝牙传感器节点板接通 12V 电源，如图所示：



第二步：程序烧写

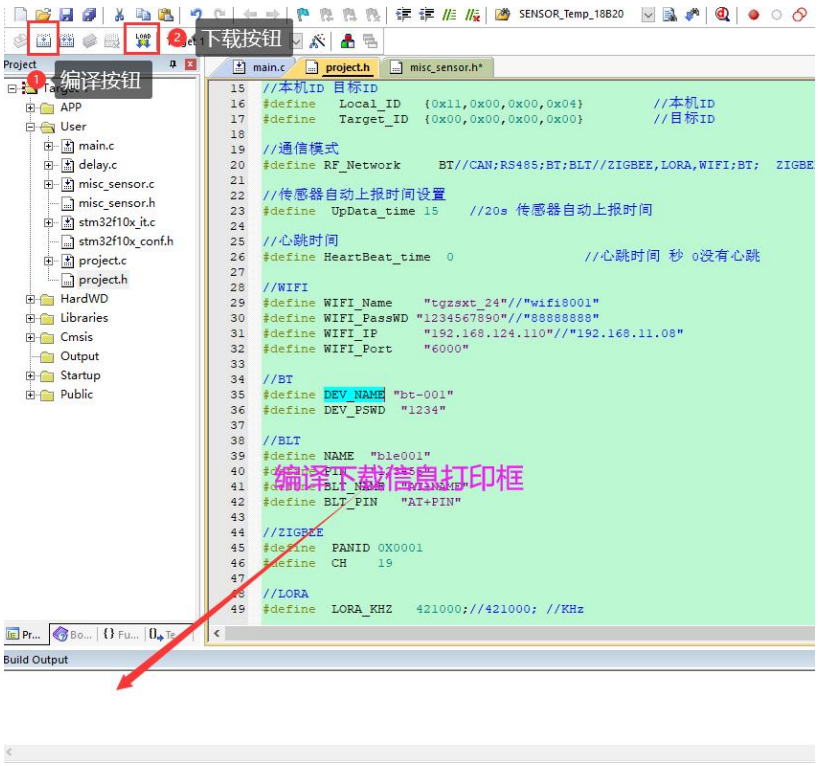
1.用 Keil4 开发环境打开实验例程……\ G:\最新工作\行业柜\stm32 新协议节点程序 2021-01-01\User ，点击 STM32.uvproj 打开工程。



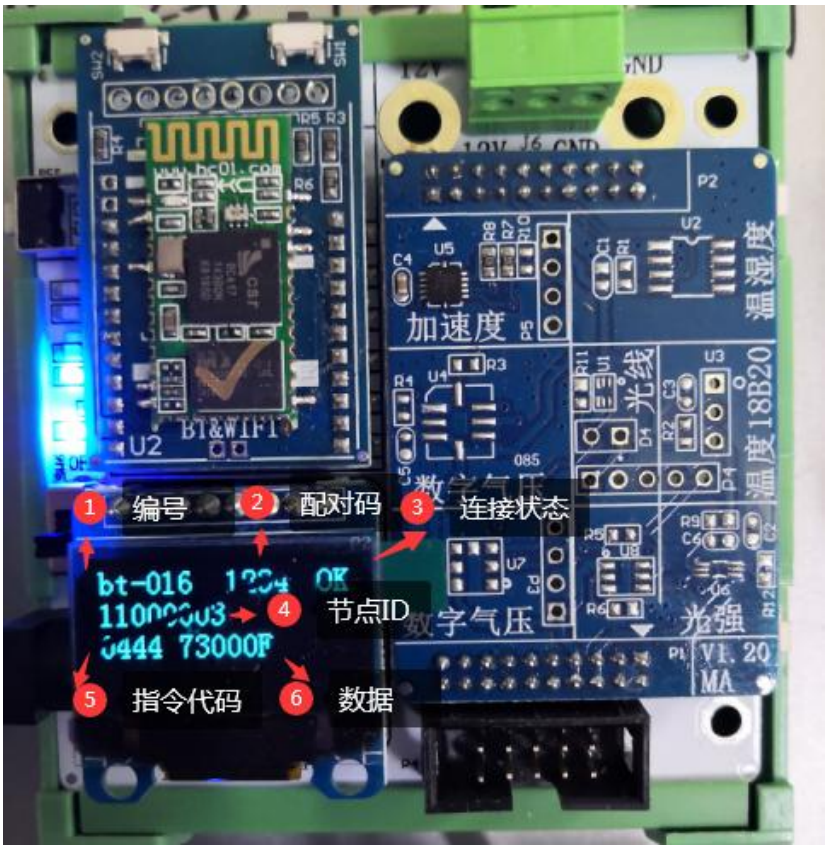
2.打开工程中 User--- >misc_sensor.h 并且找到宏定义 CONFIG_SENSOR 改成 SENSOR_Pressure（数字气压传感器）或者 SENSOR_Acceleration（三轴加速度传感器）

3. 打开工程中 User--- >project.h 修改 Local_ID（本地 ID）（气压 1100 0004）（三轴加速度 1100 0003），通信方式选择 BT，设备名称 DEV_NAME bt-001(根据实训台编号决定，例如实训台为 2 号，则可以设置 bt-002)其余不变，数据上报时间参数 upData_time（单位 s）用来设定自动上传时间间隔；

4.编译下载，点击编译图标，编译完成后下载即可；

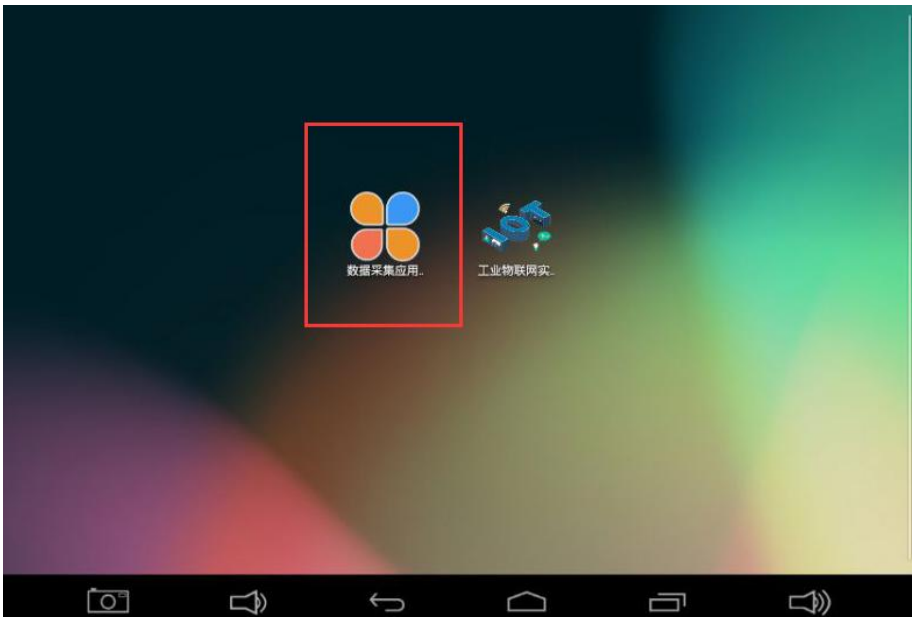


5. 程序下载后，查看并熟悉蓝牙节点屏幕上信息的含义。



第四步：网关添加蓝牙节点

1.打开网关 APP 数据采集应用。



2.点击右上角 ⋮ 图标，展开下拉菜单，找到“添加 BLUETOOTH”，如图所示。

3.根据提示框选择传感器名称、输入传感器地址码、传感器类型、及任选传感器位置，点击添加按钮，将传感器添加到软件。如图所示，添加三轴加速度。

如图所示，添加数字气压传感器节点。

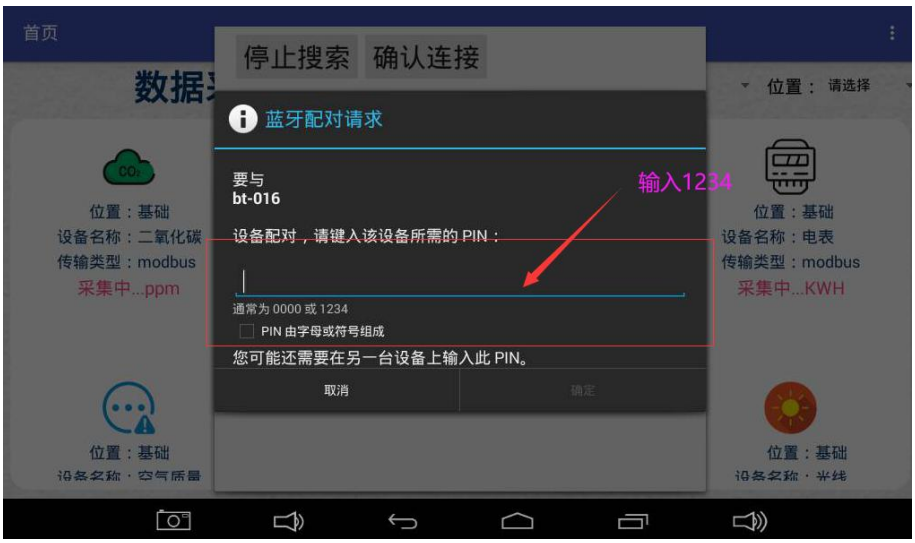
4.添加完节点后，返回主界面，“传输类型”选择 bluetooth 后，可以显示已经添加的传感器节点，但此时和硬件节点还没有连接，不会接收到数据。

第五步：网关添加蓝牙节点

1. 点击右上角 ⋮ 图标，展开下拉菜单，找到“搜索 BLUETOOTH”选项并点击，网关软件就会启动蓝牙搜索，在范围内的蓝牙节点都会被搜到，如图所示。

点击与该实训台一致的节点进行连接，以免连接其他节点。然后确认连接。

2. 如果第一次连接，会需要配对码输入 1234，如下图所示：



3.10.3.6. 实验结果

蓝牙节点每隔 15s 定时上传一次，网关数据采集应用程序实时接收、解析并显示上传的采集数据，如下图所示。

3.10.4. WiFi 无线数据通信

3.10.4.1. 实验目的

熟悉实训台 WIFI 节点的结构，供电电源、按键、指示灯的作用。

理解 WiFi 无线传感网的组网原理，掌握 WiFi 数据传输帧格式，以及“嵌入

式新协议节点程序”的使用方法。

理解 WiFi 上位机通信的方法，会使用上位机应用程序进行 WiFi 通信测试、传感器数据采集。

3.10.4.2. 实验环境

硬件：网关，WIFI 空气质量节点，WIFI 光照度节点， JLINk 仿真器，方口 USB 数据线等；

网关

蓝牙气压节点

蓝牙三轴加速度

软件：keil4 开发环境、嵌入式新协议节点程序、网关数据采集应用程序。

3.10.4.3. 实验原理

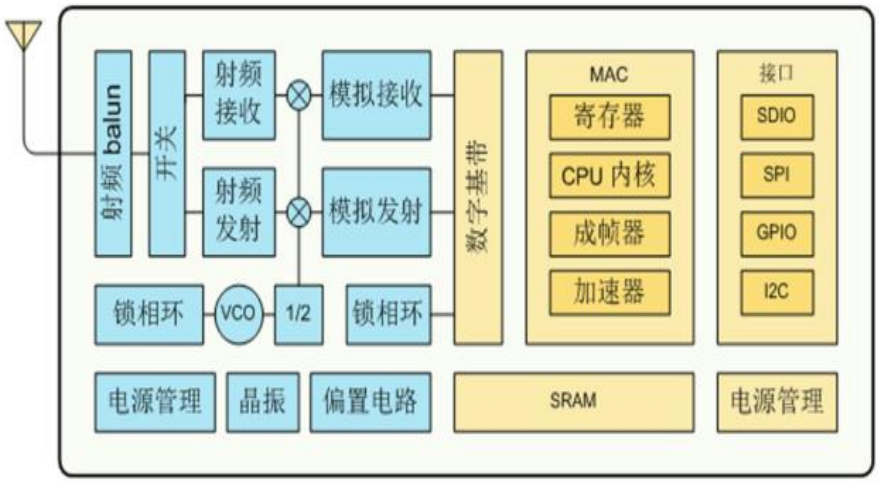
1. WiFi 传感器节点的结构

WiFi 传感器节点由 WiFi 通信模块、传感器模块、嵌入式接口底板组成。

D1、D2 连接状态指示灯：直接与 WiFi 通信模块连接的，当 WiFi 节点与服务器连接失败时，节点上 D1 指示灯闪烁，D2 指示灯熄灭；当 WiFi 节点与服务器连接成功后，节点上 D1 指示灯熄灭，D2 指示灯熄灭。

D5、D6 指示灯：不起作用。

1）ESP WiFi 通信模块



ESP8266 是一个完整且自成体系的 WIFI 网络解决方案，能够搭载软件应用，或通过另一个应用处理器卸载所有 WIFI 网络功能。

ESP8266 在搭载应用并作为设备中唯一的应用处理器时，能够直接从外接闪

存中启动。内置的高速缓冲存储器有利于提高系统性能，并减少内存需求。

另外一种情况是，无线上网接入承担 WIFI 适配器的任务时，可以将其添加到任何基于微控制器的设计中，连接简单易行，只需通过 SPI/SDIO 接口或中央处理器 AHB 桥接口即可。

ESP8266 强大的片上处理和存储能力，使其可通过 GPIO 口集成传感器及其他应用的特定设备，实现了最低前期的开发和运行中最少地占用系统资源。ESP8266 高度片内集成，包括天线开关 balun、电源管理转换器，因此仅需极少的外部电路，且包括前端模块在内的整个解决方案在设计时将所占 PCB 空间降到最低。

装有 ESP8266 的系统表现出来的领先特征有：节能 VoIP 在睡眠/唤醒模式之间的快速切换、配合低功率操作的自适应无线电偏置、前端信号的处理功能、故障排除和无线电系统共存特性为消除蜂窝 /蓝牙/DDR/LVDS/LCD 干扰。

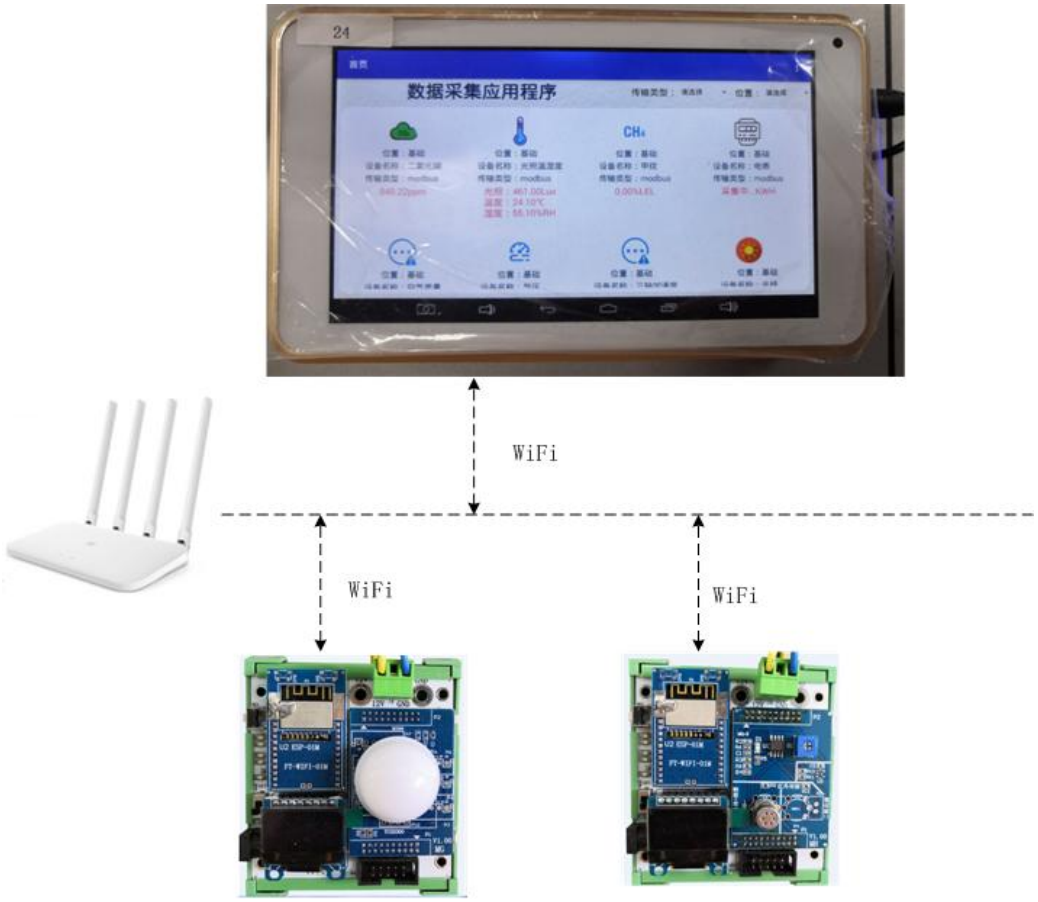
2）WIFI 配置 AT 命令

```
static unsigned char Wifi_CMD_Reply[Wifi_CMD_Reply_Num][60]=
{
    "AT\r\r\n\r\nOK\r\n",                    //0    AT 测试通讯
    "\r\nOK\r\n",                            //1    OK
    "\r\nERROR\r\n",                         //2    错误
    "\r\n+CWJAP:1\r\n\r\nFAIL\r\n",          //3
    "\r\n+CWJAP:2\r\n\r\nFAIL\r\n",          //4
    "\r\n+CWJAP:3\r\n\r\nFAIL\r\n",          //5
    "+CWJAP:1\r\n\r\nFAIL\r\n",              //6
    "\r\nbusy p...\r\n",                     //7    //忙绿
   "WIFI CONNECTED\r\n",                     //8
    "WIFI CONNECTED\r\nOK\r\n",              //9
    "CONNECT\r\n",                           //10
    "CONNECT\r\n\r\nOK\r\n",                 //11
    "ALREADY CONNECTED\r\n",                 //12
    "\r\nALREADY CONNECTED\r\n\r\nERROR\r\n", //13
    "CLOSED\r\n",                            //14
    "STATUS:2\r\n\r\nOK\r\n",                //15  已连接 AP
    "STATUS:3\r\n+CIPSTATUS:",               //16  已建立 TCP 或者 UDP 链接
    "STATUS:4\r\n\r\nOK\r\n",                //17  断开网络链接
    "STATUS:5\r\n\r\nOK",                    //18  未连接 AP
    "WIFI DISCONNECT\r\n",                   //19wifi 未连接
    "WIFI GOT IP\r\n\r\n",                   //20 获取 IP 成功
    "IP ERROR\r\n\r\n",                      //21 服务器地址错误
    "link is builded\r\n\r\nERROR\r\n",  //22  连接时建立的
    "\r\nOK\r\n> ",                          //23 准备发送数据
    "\r\nlink is not valid\r\n\r\nERROR\r\n",    //24 数据发送失败
    "Receivefrom\r\n\r\nERROR\r\n",          //25 数据发送失败
```

```
    "\r\nSEND OK\r\n",                              //26 发送成功
      "\r\nRecv ",                                   //27 发送成功
      "\r\n+IPD,",                                   //28 接收到数据
    "link is not valid\r\n\r\nERROR\r\n",     //29 数据发送失败
      "STATUS:3\r\n",              //30 已建立 TCP 或者 UDP 链接        1E
      "\r\nSTATUS:3\r\n",            //31  已建立 TCP 或者 UDP 链接    1F
      "busy",                      //32    //忙绿
      "ATE0\r\r\n\r\nOK\r\n",                    //33      0x21
      "CLOSED"                      //34    0x22
};
```

2. 本系统 WiFi 网络拓扑

网关板载 WiFi 模块，与 WiFi 传感器节点都连接到路由器，节点与网关进行 TCP 通信，如图所示。



其中，网关应用程序是 TCP 服务器、WiFi 传感器节点是 TCP 客户端。TCP 服务器信息为 192.168.XXX.110，端口号为 6000。其中 XXX 由实训台编号决定，如果是 16 号台，则 XXX 为 116，如果是 21 号台，则 XXX 为 121。

当关闭网关，节点重新上电，可以从 WiFi 节点屏幕的初始信息看到所要连接服务器的 IP 信息，如图所示。

3. WIFI 数据格式

WiFi 传感器节点通过该协议将采样信息上报给网关应用程序；网关应用程序可以向 WiFi 传感器节点下发命令。

1）协议框架

3-2 协议框架

名称	协议开始	通信类型	指令类型	随机数	本地 ID	目标 ID	指令代码	参数长度	参数内容	校验码	协议结束
长度(Byte)	1	1	1	1	4	4	2	1	n	1	1
编码方式	FE	HEX	HEX	HEX	HEX	HEX	HEX	HEX	HEX	HEX	FF

上行：含义是指传感器节点向上位机发送数据。

下行：含义是指上位机向传感器节点下发命令。

协议框架各字段说明：

（1）协议开始：固定字符 0xFE，占 1 个字节，代表当前协议数据的开始；

（2）通信类型：占 1 个字节，16 进制表示，具体说明见下表。

比特 7	比特 6	比特 5	比特 4	比特 3	比特 2	比特 1	比特 0
00：有线通信 01：RF 通信		0：RF(433M); 1:RF(868M); 2:RF(2.4G); 3:ZIGBEE; 4:LORA; 5:...					

10：网络通信 11：保留	20:WIFI;　21:BLT;　22:GPRS　23:3G;　24:4G; 25:NB-IOT; 30:RS232;　31:RS485;　32:CAN;　33:...
BIT7,BIT6 代表通信种类；BIT5-BIT0 代表每种通信种类下的具体通信类型。	

（3）指令类型：用于标识“上下行”“加密类型”“回复方式”和“是否广播”，具体格式如下表所示。

比特 7	比特 6	比特 5	比特 4	比特 3	比特 2	比特 1	比特 0
0：下行 1：上行	0：长度 1Byte 1：长度 2Byte	00：无加密 01：加密类型 1		00：无回复 01：回复 ACK 10：回操作结果		0：广播 1：点对点	串口通信-转发
注：当加密类型为 1 时，对下行报文中的指令代码、操作口令、参数长度、参数内容进行加密，对上行报文中的指令代码、参数长度、参数内容进行加密；具体加密算法待商榷。BIT6 为 1 时，数据长度大于 255，数据长度 2byte，BIT6 为零时数据长度小于 255；数据长度 1byte；当 BIT0=1 时串口通信-转发，BIT0=0 时串口直接控制；串口通信-转发时对应的参数格式组包；串口直接控制时按对应的参数格式组包。							

（4）随机数：用于识别回码或者 ACK 与之前发送的哪条指令匹配。

（5）本地 ID：表示本机设备的 ID 编码；用于设备身份识别；

备注：在有线通信时，两个 ID 填写电路背面对应 RS485 通信地址（高字节补零），在无线通信时，两个 ID 填写对应的无线通信之间的产品 ID 编号；具体使用时，参见节点屏幕显示信息。具体 ID 分类详见本协议附录部分。

（6）目标 ID：表示目标设备的 ID 编码；用于设备身份识别；

备注：在有线通信时，两个 ID 填写电路背面对应 RS485 通信地址（高字节补零），在无线通信时，两个 ID 填写对应的无线通信之间的产品 ID 编号；具体 ID 分类详见本协议附录部分。

（7）指令代码：代表本条指令的具体功能（2 个字节，前一个字节代表指令类，后一个字节代表本类下的具体指令）。

（8）参数长度：参数内容的字节长度，如果是密文则包含补偿部分的长度；

（9）参数内容：具体指令对应的参数数据，详见具体下一小节各个传感器的指令解析。

（10）CRC 校验码：校验是从 FE 之后（第二字节开始）到校验位之前的内容校验，CRC 格式采用 CRC8 格式校验。

（11）协议结束符：0XFF，代表当前协议数据结束的标志。

注意：指令代码、参数长度、参数内容三个字段的内容随着传感器节点的类

型不同而不同，下面重点说明。

2）传感器的指令代码部分说明

（1）光照传感器

a. 光照传感器下行信息请求 (0X04　0X47)

参数内容	保留
参数长度	n 字节

b. 光照传感器信息响应 (0X04　0X48)

信息响应内容如下表所示。

参数内容	设备电压	报警种类	光照	保留
参数长度	1 字节	1 字节	2 字节 A：高字节 B：低字节	n 字节

其中，报警种类含义：

比特位	事件
0	电池欠压，1 为电池欠压
1	超过阈值，1 为超过阈值
2	保留
3	保留
4	保留
5	保留
6	保留
7	保留

光照采样值内容：

参数内容
大端模式； 计算方法：(A<<8+B)*10；计算结果表示真实传感器值； 单位：LUX。

（2）空气质量传感器

a. 空气质量传感器下行信息请求（0x04 x6B）

参数内容	保留
参数长度	n 字节

b. 空气质量（模拟量）传感器响应信息 (0X04　0X6C)

参数内容	设备电压	报警种类	传感值	保留
参数长度	1 字节	1 字节	2 字节 A：高字节 B：低字节	n 字节

其中，报警种类含义：

比特位	事件
0	电池欠压，1 为电池欠压
1	气压超过阈值，1 为超过阈值
2	保留
3	保留
4	保留
5	保留
6	保留
7	保留

空气质量采样值：

参数内容：
计算方法：A<<8+B；

空气质量完整的协议包为：

fe9480001100000200000000046c040000009d1bff

3.10.4.4. 实验内容

程序流程：

第一步配置 WIFI 入网；

第二步定时采集数据(程序内可设置)，组合数据包；

第三步定时上传数据；

关键代码

（1）空气质量传感器采集发送的部分例程

```
static void Hall_Poll(int tick)
{
      if(time4_1s_on)
{time4_1s_on=0;
       //每 1s 读取一次检测值
   if ((tick % 1 == 0) &&(tick    != 32423)) {
            static char last = 0xff;
   if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_8) == 0)
      hall_status = 1;
   else
      hall_status = 0;
      adc_v = ADC_ReadValue();
   //传感器值改变则上报数据
   if (hall_status != last) {
      sensor_s_report(CMD_TYPE_NOT_ACK,0);
      last = hall_status;
   }
```

```
   //定时上报数据
 //定时上报数据
   if (tick % Sensor_UpData_time_Interval == 0) {
      if (report_enable == 0) {report_enable=1;}
      else {
             }
   }
      }
}
if(time4_100ms_on)
{ time4_100ms_on=0;
      if(UART3_Time_Flage==0)//数据线,无数据
            {     if (Sensor_UpData_time_Interval) {
           sensor_misc.getValue();          //传感器数据采集
                  if((report_enable<2)&&(report_enable>0))report_enable++;
                  else report_enable=0;
                  if(report_enable)
                    { sensor_s_report(CMD_TYPE_NOT_ACK,0);
           }
      }
       }
}
}
```

（2）光线采集上传数据部分例程

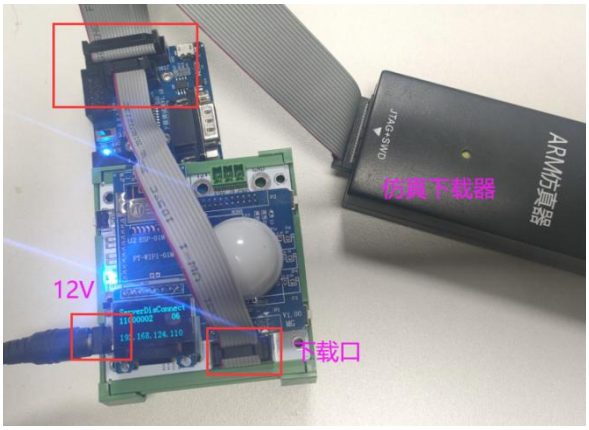
```
static void Hall_Poll(int tick)
{
      if(time4_1s_on)
{time4_1s_on=0;
       //每 1s 读取一次检测值
   if ((tick % 1 == 0) &&(tick    != 32423)) {
   static char last = 0xff;
   if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_8) == 0)
      hall_status = 1;
   else
      hall_status = 0;
      adc_v = ADC_ReadValue();
      //传感器值改变则上报数据
   if (hall_status != last) {
      sensor_s_report(CMD_TYPE_NOT_ACK,0);
      last = hall_status;
   }
   //定时上报数据
 //定时上报数据
   if (tick % Sensor_UpData_time_Interval == 0) {
      if (report_enable == 0) {report_enable=1;}
      else {
             }
   }
      }
}
if(time4_100ms_on)
{ time4_100ms_on=0;
      if(UART3_Time_Flage==0)//数据线,无数据
            {     if (Sensor_UpData_time_Interval) {
```

```
            sensor_misc.getValue();          //传感器数据采集
                    if((report_enable<2)&&(report_enable>0))report_enable++;
                    else report_enable=0;
                    if(report_enable)
                      { sensor_s_report(CMD_TYPE_NOT_ACK,0);
            }
        }
          }
}
}
```

3.10.4.5. 实验步骤

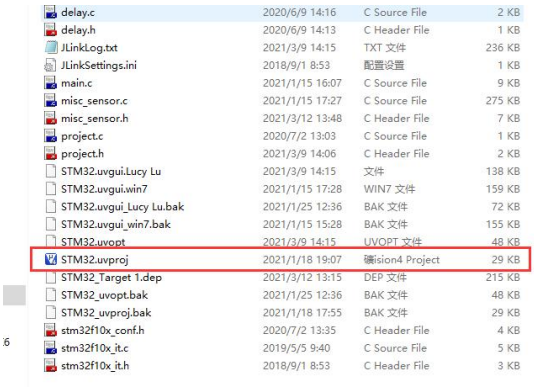
第一步：硬件连接（空气质量和光照一致）

使用 USB 方口数据线连接 J-LINK 仿真器和 PC 机，用 20 针排线连接 J-LINK 仿真器和下载调试板，10 针排线连接下载调试板和 WIFI 传感器节点板，然后给 WIFI 传感器节点板接通 12V 电源，如图 xx 所示：

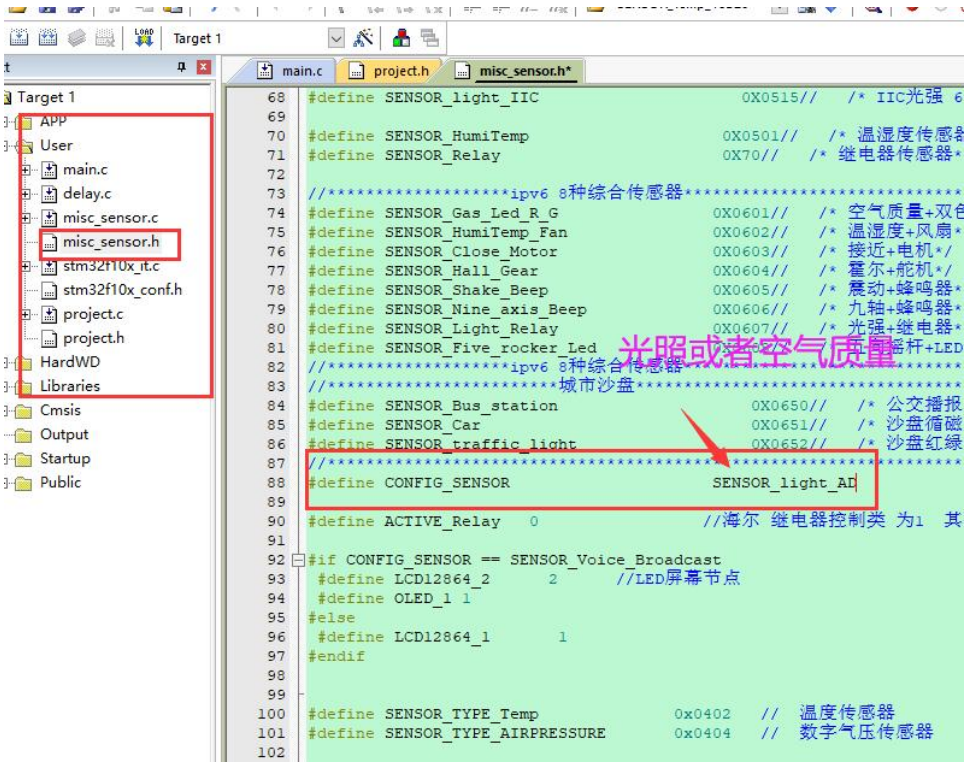


第二步：程序烧写

1. 用 Keil4 开发环境打开实验例程...\stm32 新协议节点程序 2021-01-01\User ，点击 STM32.uvproj 打开工程。

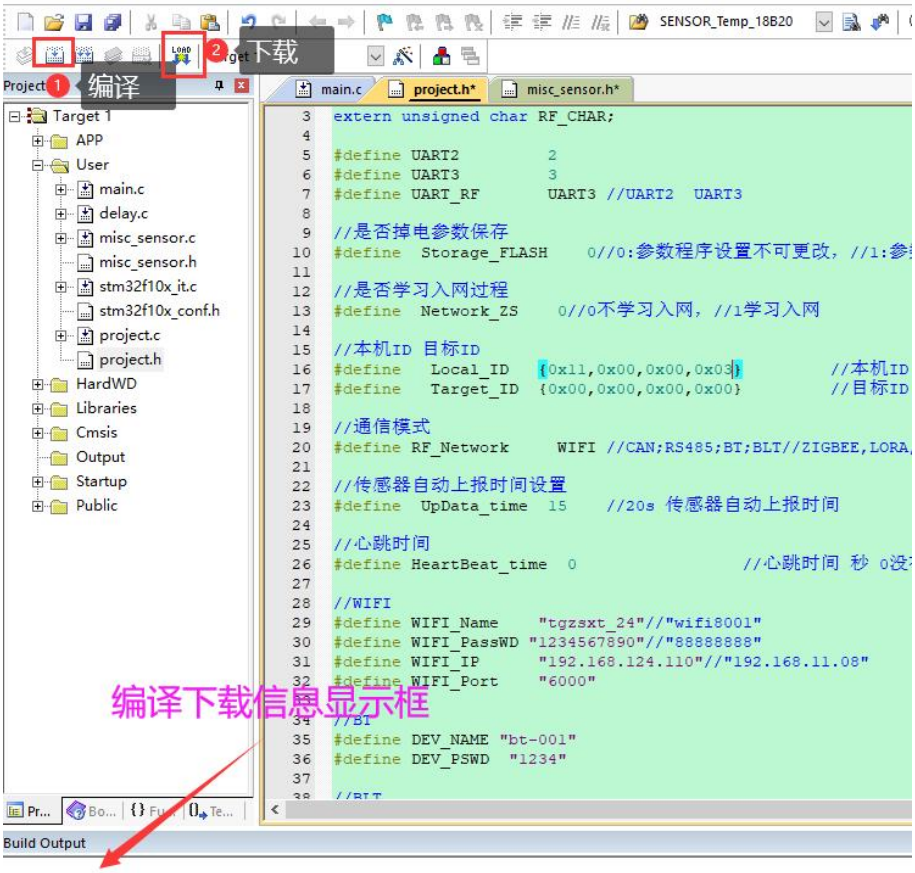


2.打开工程左侧目录中 User--- >misc_sensor.h 并且找到宏定义 CONFIG_SENSOR 改成 SENSOR_AirGas（空气质量传感器）或者 SENSOR_light_AD（光照传感器）



3. 打开工程中 User--- >project.h 修改 Local_ID（本地 ID）（空气质量 1100 0004）（光照 1100 0003），通信方式选择 WIFI，根据实训台编号设置相关 WIFI 参数，数据上报时间参数 upData_time（默认 15s），用来设定自动上传时间间隔；

4.编译下载，点击编译图标，编译完成后下载即可；



5. 程序下载后，查看并熟悉 WiFi 节点屏幕上信息的含义。

①：代表连接 TCP Server 的状态，成功连接，显示“Server Connect”；

②：节点 ID 号，光照节点为 11000002，空气质量节点为 11000001.

③：连接状态编号，0A 代表连接成功，07 代表连接失败。

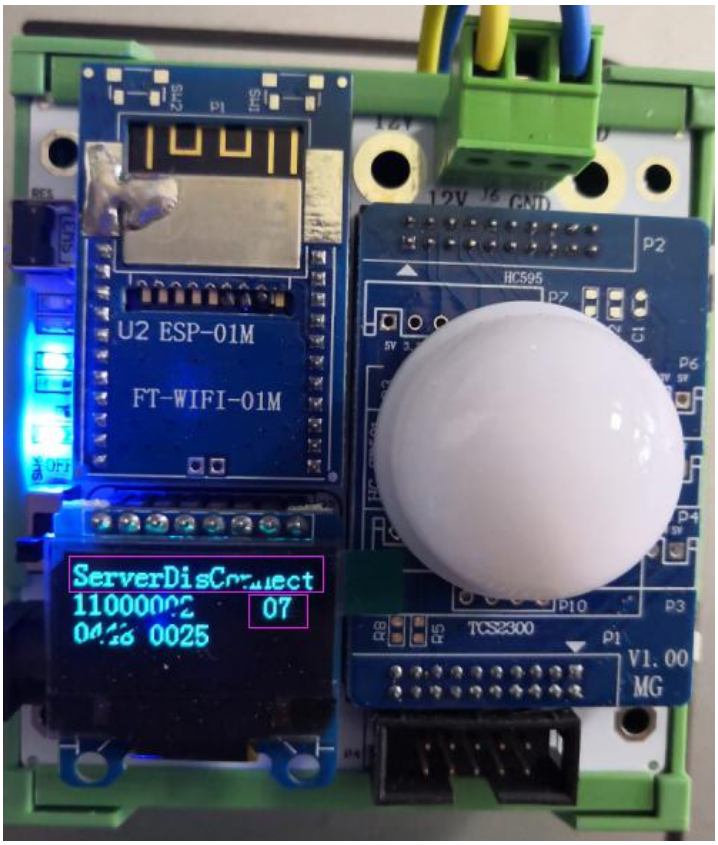
④：节点传感器指令代码

⑤：采样数据

⑥：D1/D2 指示灯：通信模块的状态灯，连接成功时，熄灭；连接失败时，闪烁。

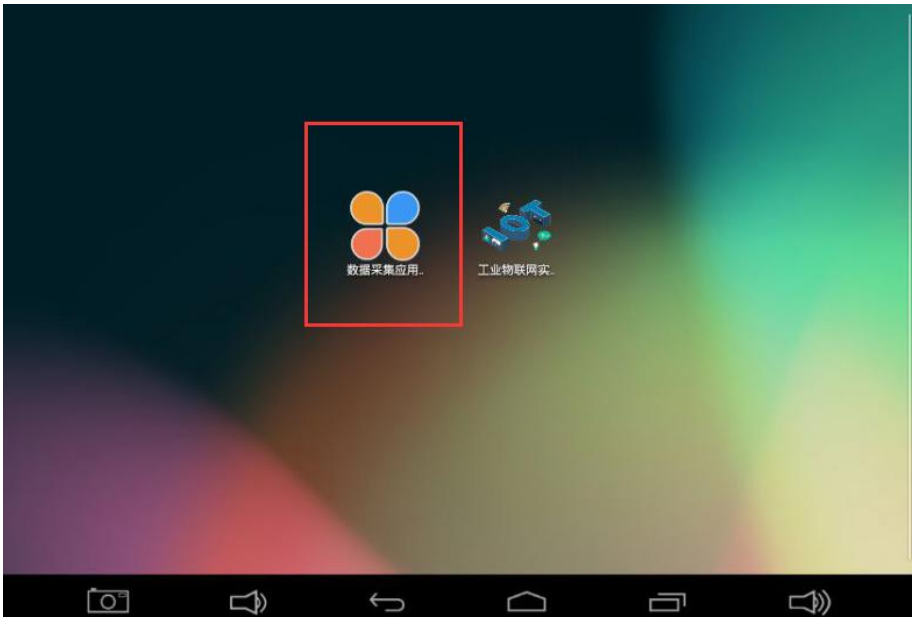
⑦：D5/D6 指示灯：嵌入式接口板控制的指示灯。

网关未添加节点时，节点连接服务器失败，屏幕显示信息如图所示。



第四步：网关添加 WIFI 节点

1.打开网关 APP 数据采集应用。



2.点击右上角 图标，展开下拉菜单，找到“添加 WiFi”，如图所示。

3. 根据提示框选择传感器名称、输入传感器地址码、传感器类型、及任选传感器位置，点击添加按钮，将传感器添加到软件。如图所示，添加光照度。

如图所示，添加空气质量。

4. 返回主界面，传输类型选择 wifi，就可以看到添加的 WiFi 传感器，等待数据上传即可。

3.10.4.6. 实验结果

WiFi 传感器节点每隔 15s 定时上传一次，数据采集应用程序实时接收传感器采样数据解析，并显示，如下图所示。

3.10.5. LoRa 无线数据通信

3.10.5.1. 实验目的

1. 理解 LoRa 无线数据通信工作原理和驱动电路设计方法；
2. 掌握 LoRa 无线数据发送和接收的嵌入式单片机编程方法。

3.10.5.2. 实验环境

1. 硬件：远程测控终端模块*2、程序下载调试板、10 针排线、20 针排线、ARM JLINK 仿真器、USB 方口线、DC 12V 电源以及 PC 机。
2. 软件：Windows 7 及以上操作系统，Keil4 for ARM 4.7 开发环境。

图 3-26 远程测控终端模块示意图

3.10.5.3. 实验原理

1. LoRa 简介及原理

LoRa 作为一种无线技术，基于 Sub-GHz 的频段使其更易以较低功耗远距离通信。本实验使用的 lora 模块基于 SX1278 芯片和一个 8 位单片机。其中 SX1278 芯片配合外围电路就形成了 SX1278 模块，这个模块和单片机之间通过 SPI 进行通行。经过单片机转换后，本实验使用的 lora 模块最终通过串口和远程测控终端的主控 STM32F407 进行通信。

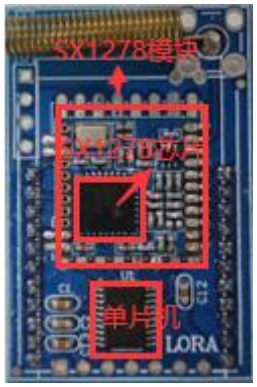


图 3-27 lora 模块示意图

图 3-27 所示的 lora 模块和远程测控终端主控的串口 4 进行通信。

SX1276/77/78 系列产品采用了 LoRaTM 扩频调制解调技术，使器件传输距离远远超出现有的基于 FSK 或 OOK 调制方式的系统。在最大数据速率下，LoRaTM 的灵敏度要比 FSK 高出 8dB；但若使用低成本材料和 20ppm 晶体的 LoRaTM，收发器灵敏度可以比 FSK 高出 20dB 以上。此外，LoRaTM 在选择性和阻塞性能方面也具有显著优势，可以进一步提高通信可靠度。同时，它还提供了很大的灵活性，用户可自行决定扩频调制带宽（BW）、扩频因子（SF）和纠错率（CR）。扩频调制的另一优点就是，每个扩频因子均呈正交分布，因而多个传输信号可以占用同一信道而不互相干扰，并且能够与现有基于 FSK 的系统简单共存。

此外，SX1276/77/78 还支持标准的 GFSK、 FSK、 OOK 及 GMSK 调制模式，因而能够与现有的 M-BUS 和 IEEE 802.15.4g 等系统或标准兼容。

SX1276 的带宽范围为 7.8~500kHz，扩频因子为 6~12，并覆盖所有可用频段。 SX1277 的带宽和频段范围与 SX1276 相同，但扩频因子为 6~9。 SX1278

的带宽和扩频因子选择与 SX1276 相同，但仅覆盖较低的 UHF 频段。

SX1278 的应用：

- 自动抄表
- 家庭和楼宇自动化
- 无线告警和安防系统
- 工业监视与控制
- 远程灌溉系统

Lora 模块的配置参数如下所示：

LORA 配置流程		
FT+CZQSEND\r\n	\r\nOK\r\n	结束发送，退出透传模式
配置命令		
{0xF7,0x4C,0x4F,0x52,0x41,	0xF7:发命令 0xF1:回复 LORA	
0x90,	//0x90:写 0xF0:读 （操作寄存器）	
0x02,	//起始地址	
0x07,	//长度 后面的总长度	
0x06,0x9F,0x00,	//频段 410MHz~510MHz 不同频段需要间隔 100KHz~1MHz	
0x65,	//速率 0x01~0x70	速率等级 1~112
0x0F,	//发送功率 0x00~0x0F	0~15DB
0x05,	//串口速率 0x05:38400	2400 4800 9600 19200 38400 57600 115200
0x00};	//串口效验 无校验	
LORA 进入透传工作状态		
FT+CLSEND\r\n	\r\n>\r\n	发送数据，进入 LORA 透传模式

2. 电路原理

Lora 模块原理图如图 3-28 所示。STM32 的 PA8 口是 Lora 模块的供电使能口，高电平时 Lora 模块供电使能；STM32 的 PC10 和 PC11 也就是 STM32 的串口 4，和 Lora 模块相连，STM32 和 Lora 模块通过串口 4 进行通信。

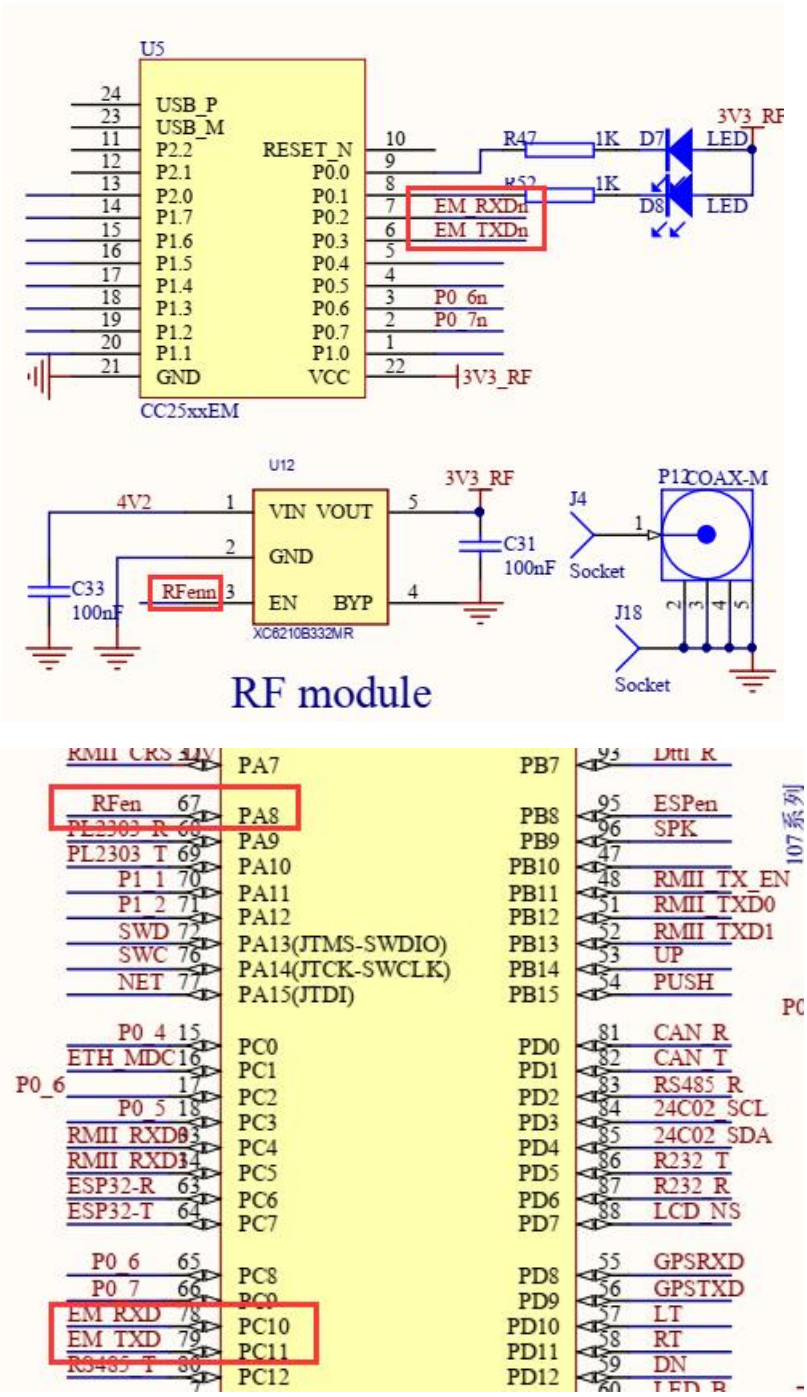


图 3-28 LoRa 硬件原理图

3.10.5.4. 实验内容

本实验主要是一个远程测控终端作为发送端通过 Lora 发送数据，另一个远程测控终端作为接收端接收 Lora 数据，并将接收到的数据在 LCD 屏上显示出来。

值得一提的是，发送方的的远程测控终端需要长按右键 2 秒以上进入 Lora 通信模式才会发送 Lora 数据，接收方的远程测控终端也要长按右键 2 秒以上进

入 Lora 通信模式才能接收数据并显示。

1. 发送方的远程测控终端

发送方远程测控终端的主函数内容如下所示：

```
int main(void)
{
    SysMainInit();    //系统初始化，包括看门狗，系统延时，LED，按键，定时器，LCD
等初始化

    while(1)
    {
        Iwdg_Feed();    //喂狗

        KeyDetection_Process();  //按键检测并执行相应操作

        if(0 == viewModeFlg)       //正常模式，非查看模式
        {
            Periodic_Process();      //周期性任务

            if(
#ifdef MODBUS_RTU_RS485
                MODBUS_RTU_RS485 ||
#endif
            (WirelessType == LORA && GET_LORA_INIT_FINISH_FLAG() != _FALSE)
                || (WirelessType == ZIGBEE && GET_ZIGBEE_INIT_FINISH_FLAG() !=
_FALSE)
                || (WirelessType == WIFI && GET_WIFI_INIT_FINISH_FLAG() !=
_FALSE)
                || (WirelessType == NB_IOT && GET_NBIOT_INIT_FINISH_FLAG() !=
_FALSE)
                || (WirelessType == ZIGBEE_LORA &&
GET_ZIGBEE_LORA_INIT_FINISH_FLAG() != _FALSE)
                && (_TRUE == _SUCCESS))
            {
                if(ModbusRecv_Process_EN)       //允许处理 Modbus 格式命令
                {
                    ModbusRecv_Process_EN = 0;  //禁止处理 Modbus 格式命令
                    Modbus_Cmd_Process();          //接收到 Modbus 格式的命令后，执
行相应的采集或者控制操作
                }
            }
        }
        else if(2 == viewModeFlg)
        {
            if(periodicLoraSendFlg)             //Lora 周期性发送数据标志位置位
            {
                periodicLoraSendFlg = 0;

                LoraSendCmd(LoraSendData);
                Display_Clear_Rect(82, 64, LCDW, 8, BLUE);
                Display_ASCII5X8(82,64, BRRED, (char *)LoraSendData);
            }
        }
    }
```

```
}
```

首先进行系统初始化，包含了 Lora 初始化。主循环中，喂狗，按键检测并执行相应操作，比如长按右键会进入 Lora 通信模式，然后判断是否是正常模式，是的话执行周期性任务，否则如果是 Lora 通信模式，周期性发送 Lora 数据，发送的内容是“Hello world!”字符串。这里的发送函数是 LoraSendCmd()，其实就是串口 4 的字符串发送函数 USART4_SendStr()。

发送 Lora 数据的周期是通过定时器设定的，这里设定的时间是 3 秒。

2. 接收方的远程测控终端

接收方远程测控终端的主函数内容如下所示：

```
int main(void)
{
    SysMainInit();    //系统初始化，包括看门狗，系统延时，LED，按键，定时器，LCD 等初始化

    while(1)
    {
        Iwdg_Feed();    //喂狗

        KeyDetection_Process();  //按键检测并执行相应操作

        Uart4RxBufUpdate();        //检测是否接收到 Lora 数据

        if(0 == viewModeFlg)       //正常模式，非查看模式
        {
            Periodic_Process();     //周期性任务

            if(
#ifdef MODBUS_RTU_RS485
                MODBUS_RTU_RS485 ||
#endif
            (WirelessType == LORA && GET_LORA_INIT_FINISH_FLAG() != _FALSE)
                || (WirelessType == ZIGBEE && GET_ZIGBEE_INIT_FINISH_FLAG() != _FALSE)
                || (WirelessType == WIFI && GET_WIFI_INIT_FINISH_FLAG() != _FALSE)
                || (WirelessType == NB_IOT && GET_NBIOT_INIT_FINISH_FLAG() != _FALSE)
                || (WirelessType == ZIGBEE_LORA && GET_ZIGBEE_LORA_INIT_FINISH_FLAG() != _FALSE)
                && (_TRUE == _SUCCESS))
            {
                if(ModbusRecv_Process_EN)       //允许处理 Modbus 格式命令
                {
                    ModbusRecv_Process_EN = 0;  //禁止处理 Modbus 格式命令
                    Modbus_Cmd_Process();         //接收到 Modbus 格式的命令后，执行相应的采集或者控制操作
                }
            }
```

```
            }
            else if(2 == viewModeFlg)
            {
                if(UART4_PendingDataBuffer[0] != 0)
                {
                    LoraNotRecvCount = 0;
                    Display_Clear_Rect(82, 64, LCDW, 8, BLUE);     //部分清屏
                    Display_ASCII5X8(82,64, BRRED, (char *)UART4_PendingDataBuffer);
//显示接收的 Lora 数据
                }
                else   //未接收到 Lora 数据
                {
                    LoraNotRecvFlg = 1;

                    if(LoraNotRecvCount > 2000)     //超过 4 秒未接收到数据
                    {
                        LoraNotRecvCount = 0;
                        Display_Clear_Rect(82, 64, LCDW, 8, BLUE);     //部分清屏
                    }
                }
            }

            if(UART4_PendingDataBuffer[0] != 0)
            {
                memset(UART4_PendingDataBuffer, 0, UART4_REV_BUFFER_SIZE);
            }
        }
}
```

首先是进行系统初始化，当然也包含了 Lora 初始化。然后主循环中喂狗，按键检测并执行相应操作，如果长按右键会进入 Lora 通信模式。判断是否为正常模式，正常模式下执行周期性任务。如果是 Lora 通信模式，将接收到的 Lora 数据在 LCD 屏上显示出来，不过如果超过一定时间没有收到数据，也会对数据显示的位置进行清屏，这样有个好处，就是发送端如果停止发送了，很快接收端就不再显示接收到数据。

3.10.5.5. 实验步骤

1. 首先进行硬件连接，使用 USB 方口线连接 ARM JLINK 仿真器和 PC，使用 20 针排线连接仿真器和下载调试板，使用 10 针排线连接下载调试板和发送方远程测控终端，最后 使用 DC 12V 电源给远程测控终端供电。硬件连接如图 2- 所示。

图 3-29 硬件连接图

2. 打开发送方远程测控终端的代码，路径：03-无线网络通信\3.9.3LoRa 无线通信 -Lora 发送\Project\RTU.uvproj。如果多组设备共同进行实验，不同组之间需要保证Lora频段不同，修改位置如图3-30所示，修改范围为410Mhz-510MHz，间隔最好保证有 2MHz，以免发生冲突。重新编译代码并将代码下载到发送方远程测控终端模块中，编译和下载按钮如图 3-30 所示。

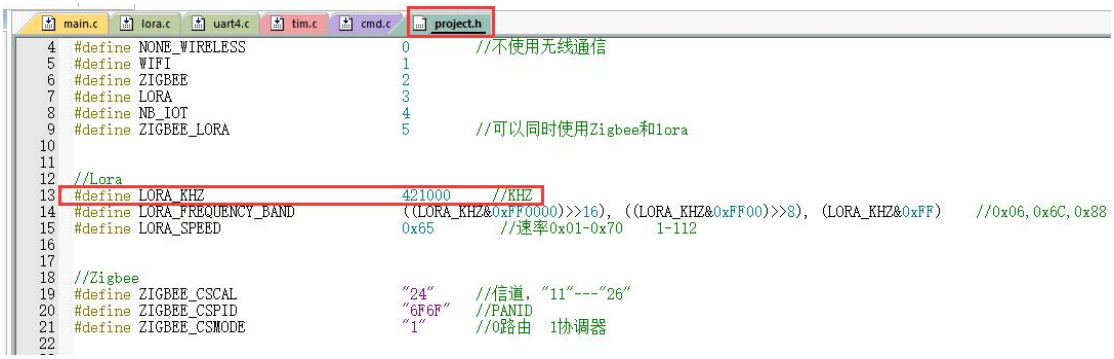


图 3-30 修改 Lora 频段

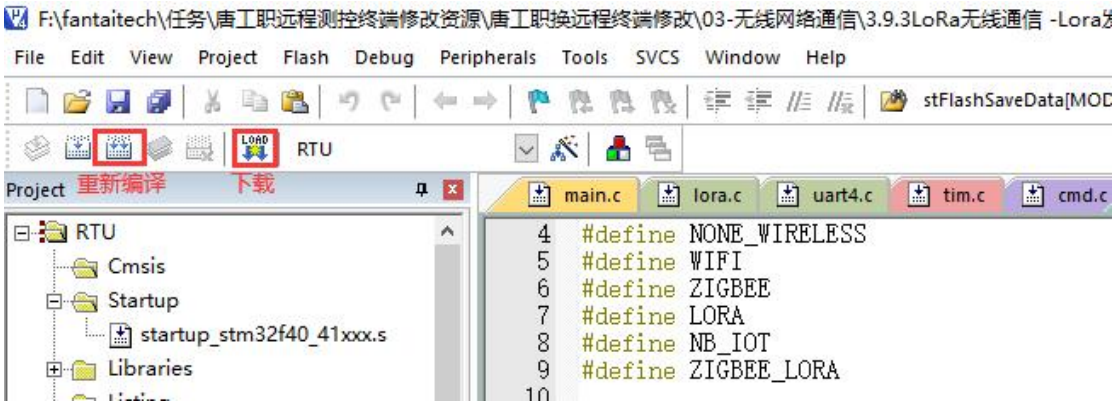


图 3-31 编译下载程序

3. 发送方的远程测控终端烧录好程序后，将 10 针排线从发送方的远程测控终端烧录口上拔下，插到接收方的远程测控终端烧录口，使用 DC 12V 电源给接收方的远程测控终端供电。

4. 打开接收方远程测控终端的代码，路径：03-无线网络通信\3.9.3LoRa 无线通信 -Lora 接收\Project\RTU.uvproj。如果多组设备共同进行实验，不同组之间需要保证 Lora 频段不同，修改范围为 410Mhz-510MHz，间隔最好保证有 2MHz，以免发生冲突。尤其需要注意，同一组设备的接收方远程测控终端 Lora 频段需要和发送方保持一致。重新编译代码并将代码下载到接收方远程测控终端模块中。

5. 从接收方远程测控终端上拔下仿真器，然后给发送方和接收方的远程测控终端重新上电，让它们的程序重新运行。

3.10.5.6. 实验结果

长按发送方和接收方远程测控终端的右键 2 秒以上，直到看到 LCD 屏显示切换到 Lora 通信界面再松开按键。

发送方和接收方远程测控终端的界面显示如下所示：

图 3-32 发送方远程测控终端 LCD 显示

图 3-33 接收方远程测控终端 LCD 显示

可以看到，发送方的远程测控终端显示“Send Data : Hello world!”，接收方的远程测控终端显示“Recv Data : Hello world!”，发送方和接收方远程测控终端的 Lora 指示灯（壳上显示 D1）每 3 秒闪烁一下，这是因为本实验 Lora 数据发送的周期是 3 秒。

如果将发送方远程测控终端断电或者短按中间的 PUSH 按键（OK 按键）退出 Lora 通信模式，那么接收端就不再收到数据，LCD 显示如下：

图 3-34 停止 Lora 数据发送后接收方远程测控终端的 LCD 显示

4. 云平台网关接入应用

4.1. 云平台功能简介

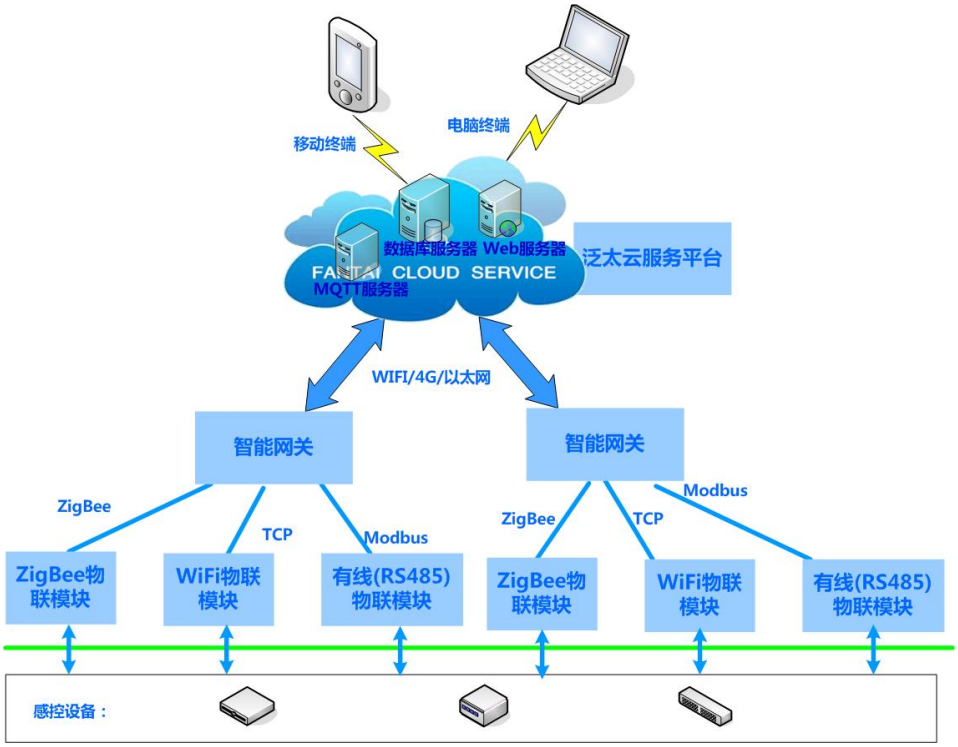
泛太物联网云服务平台旨在为物联网前端设备提供云端接入、云端存储、消息分发、数据处理、在线采集、远程控制、逻辑策略、Web API 接口二次开发等服务的物联网教学系统。 通过物联网云服务平台相关的 API、应用场景（项目）创建管理等功能，为实验、实训、项目设计、毕业设计等提供一套完整的软硬件环境，可以快速了解物联网行业应用及相关技术。

目标：

（1）物联网应用场景搭建服务：以创建项目（物联网应用）的形式，应用云平台提供的各种功能，快速搭建物联网网关设备、传感器节点、控制节点以及控制策略，使得物联网场景传感器数据存储、数据展示变得轻松简单，让开发者经过简单的操作就能开发出专业的物联网应用系统；

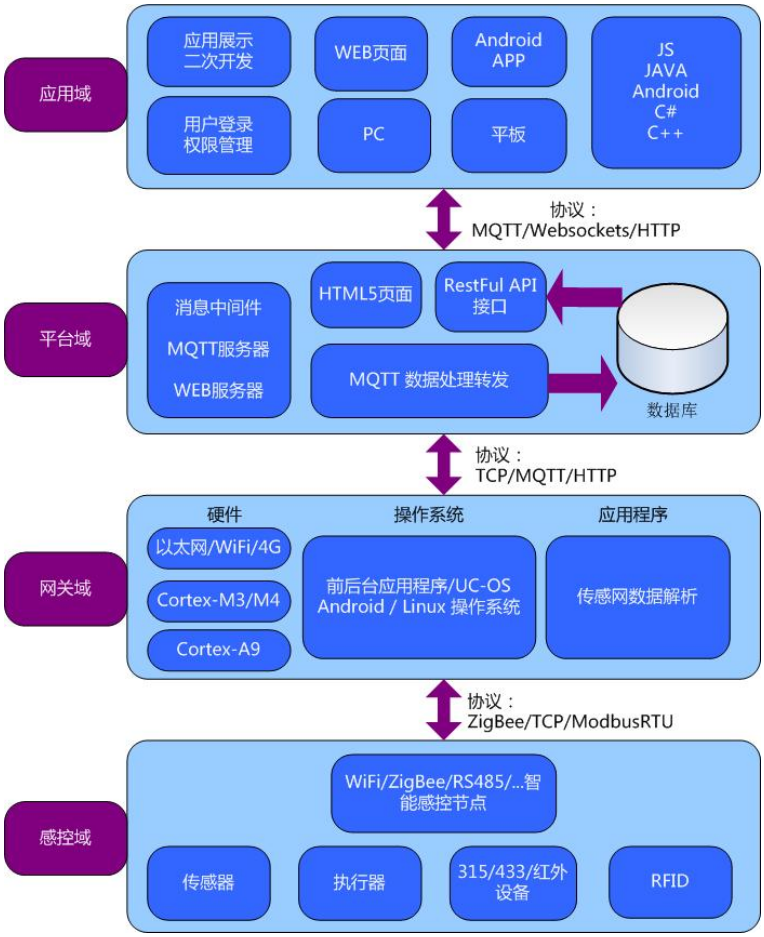
（2）物联网前端设备接入服务：为用户的传感器设备提供接入服务，按照系统提供的平台接入协议，就可将传感器设备连入平台。

（3）Web API 接口开发服务：根据提供的二次开发接口，让开发者经过编程就能开发出属于自己的物联网应用系统。



整个平台的运行可简单的理解为“通过物联网网关将现场传感器的数据传输到云平台，用户访问云平台获取传感器数据、定制行业应用的过程”。

平台采用 Browser/Server 处理各模块之间的数据传输，主要包括感控域、网关域、平台域、应用域。



1) 感控域：由传感器、执行器、RFID、射频设备与传感网通信节点组成。底层感控设备支持如 485 探测器、zigbee 温湿度、wifi 光线、315 探测器、315 强电、433 窗帘、红外遥控等；传感网通信模块支持 ZigBee、WIFI、RS485 等；通讯协议支持 ZigBee、TCP、ModbusRTU 等。

2) **设备域：**针对智能网关，集成并解析多种物联网传感网协议，如 zigbee、modbusRTU、TCP、蓝牙等，支持采集、控制、传输等功能。

3) 平台域：

- 支持 MQTT、UDP、TCP、HTTP 多种协议的测试与网关接入；

4) 应用域：

- 浏览已发布的应用
- 搭建自己的应用界面

- 通过 API 二次开发接口自定义应用开发

4.2. 云平台协议分析

网关接入平台的协议支持 MQTT、CoAP、TCP/UDP 协议，及 HTTP Restful 接口调用。

一、MQTT 协议

Topic 主题：

上行：/upstream/{deviceId}

下行：/downstream/{deviceId}

Payload 负载：采用 json 数据格式，内容如表所示。

字段	名称	备注
deviceId	设备 ID、IMEI 号	
sensorId	传感器 ID、IMEI+指令代码+传感器 Mac	
sensortype	传感器类型	
key	传输密钥	
data	json 数据(扩展)	{"func":"","funcCode":""} {"value":""} 当控制设备为面板设备时,funcCode 填入面板 Mac
code	传感器码(FTLink 地址,wifi 地址,zigbee 地址,面板 ID)	
connecttype	连接类型(MQTT,TCP,UDP,CoAP)	
trantype	传输类型 (Zigbee,Modbus,Wifi,315-zigbee)	

具体协议参考资料中提供的“MQTT 协议 JSON 数据格式.doc”文档。

二、CoAP 协议

COAP 主题：

上行：/api/{imei}/upstream

下行：/api/{imei}/downstream

消息与 MQTT 的 Payload 负载一致。

三、TCP、UDP 协议

数据格式与 MQTT 的 Payload 负载一致。

四、平台 RESTFUL 接口

平台提供 Restful 接口供终端设备调用，可以获取传感器属性信息、发送传感器控制命令，获取历史数据，下面介绍常用的几个 Restful 接口函数。

（1）根据项目 id 查询传感器信息

请求路径：

http://{ip}:{port}/RabbitMqServer/{pid}/device/sensorinfo

请求方式: GET

请求参数: int pid

响应参数:

[{"di_id":"","scti_id":"","scti_name":"","sei_id":,"sei_mac":"","sei_status":,"sei_value":"","sti_control":,"sti_id":"","sti_name":"","sti_unit":,"tti_id":"","tti_name":""},{"di_id":"","scti_id":"","scti_name":"","sei_id":,"sei_mac":"","sei_status":,"sei_value":"","sti_control":,"sti_id":"","sti_name":"","sti_unit":,"tti_id":"","tti_name":""},...]

参数说明:

[请求参数]: pid 项目 ID

[响应参数]: di_id: 设备 ID,scti_id: 场景类型 ID,scti_name: 场景类型,sei_id: 传感器 ID,sei_mac: 传感器 Mac,sei_status: 传感器状态,sei_value: 传感器当前值,sti_control: 控制类型,sti_id: 传感器类型 ID, sti_name: 传感器类型,sti_unit: 单位,tti_id: 传输类型 ID,tti_name: 传输类型

（2）根据传感器 id 查询传感器信息

请求路径：

http://{ip}:{port}/RabbitMqServer /sensorinfo/{seid}

请求方式: GET

请求参数: string seid

响应参数:

{"di_id":"","scti_id":"","scti_name":"","sei_id":,"sei_mac":"","sei_status":,"sei_

value":"","sti_control":,"sti_id":"","sti_name":"","sti_unit":,"tti_id":"","tti_name":""}

参数说明:

[请求参数]: seid 传感器 ID

[响应参数]: di_id: 设备 ID,scti_id: 场景类型 ID,scti_name: 场景类型,sei_id: 传感器 ID,sei_mac: 传感器 Mac,sei_status: 传感器状态,sei_value: 传感器当前值,sti_control: 控制类型,sti_id: 传感器类型 ID, sti_name: 传感器类型,sti_unit: 单位,tti_id: 传输类型 ID,tti_name: 传输类型

（3）根据设备 id 查询设备信息

请求路径：

http://{ip}:{port}/RabbitMqServer /deviceinfo/{did}

请求方式: GET

请求参数: string did

响应参数:

{"di_conttype":,"di_id":"","di_key":"","di_mac":,"di_name":"","di_status":,"di_type":,"pi_seq":}

参数说明:

[请求参数]: did 设备 ID

[响应参数]: di_conttype: 连接类型,di_id: 设备 ID,di_key: 加密 Key,di_mac: 设备 Mac,di_name: 设备,di_status: 设备状态,di_type: 设备类型,pi_seq: 项目 ID

（4）根据传感器 id、传感器类型及动作 id 查询动作码值

请求路径：

http://{ip}:{port}/RabbitMqServer/sensorfuncinfo/code

请求方式: POST

请求参数: {"sei_id":"","sti_id":"","fui_id":""}

响应参数: {"sfi_code":""}

参数说明:

[请求参数]: sei_id: 传感器 ID,sti_id: 传感器类型 ID,fui_id: 动作 ID

[响应参数]: sfi_code: 动作码值

（5）页面控制传感器

请求路径：

http://{ip}:{port}/RabbitMqServer/cmds

请求方式: POST

请求参数:

{"deviceId":"","sensorId":"","key":"","data":{"func":""},"code":"","trantype":"","connecttype":"MQTT"}

响应参数:

boolean

参数说明:

[请求参数]: deviceId: 设备 ID, sensorId: 传感器 ID, key: 加密 Key, code: 传感器 Mac,func: 动作 ID, trantype: 传输类型 ID, connecttype: 连接类型

[响应参数]: true/false 执行成功/失败

（6）根据时间查询传感器记录

请求路径：

http://{ip}:{port}/RabbitMqServer/historyrecord/time

请求方式: POST

请求参数:

{"pi_seq":"","sei_id":"","startTime":"","endTime":""}

响应参数:

[{"hr_seq":"","di_id":"","sei_id":"","hr_time":"","hr_value":"","pi_seq":""},{"hr_seq":"","di_id":"","sei_id":"","hr_time":"","hr_value":"","pi_seq":""},…]

参数说明:

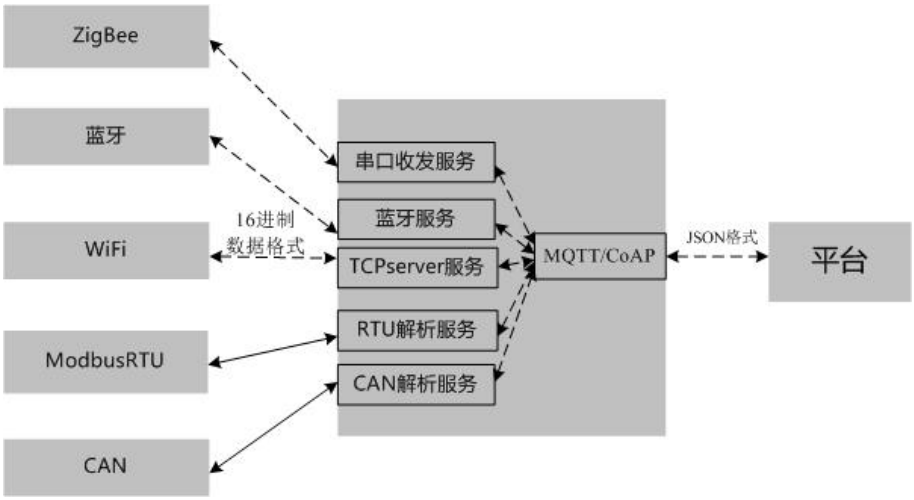
[请求参数]: pi_seq: 项目 ID, sei_id: 传感器 ID, startTime: 查询起始时间, endTime: 查询结束时间

[响应参数]: hr_seq: 历史数据自增 ID, di_id: 设备 ID, sei_id: 传感器 ID,

hr_time: 历史采集时间, hr_value: 数值，pi_seq:项目 ID

4.3. 网关多网协议解析与转换

协议转换如图所示。网关就是将一种协议转换成另一种协议的装置。本系统，网关将 ZigBee 协议、蓝牙协议、wifi、Rtu、CAN 等多种 16 进制数据格式的协议，经过解析与重组，转换成可读性更强的 JSON 数据格式的 MQTT、CoAP 协议，再接入平台。



4.4. 工业以太网网关 MQTT 协议接入平台实验

4.4.1. 实验目的

- 了解 Android MQTT 网络通信访问协议
- 掌握 Android MQTT 网络通信编程方法

4.4.2. 实验环境

1. 软件：用户 PC（Microsoft Windows XP 以上系统平台）1 台，确保已正确安装及配置 jdk 等 Android 开发环境，浏览器用于登陆云平台 MQTT 调试工具。

2. 硬件：物联网网关 1 个。

4.4.3. 实验原理

MQTT 的全称为 Message Queue Telemetry Transport（Message Queuing Telemetry Transport，消息队列遥测传输协议），是在 1999 年，由 IBM 的 Andy Stanford-Clark 和 Arcom 的 Arlen Nipper 为了一个通过卫星网络连接输油管道的项目开发的。简单地来说 MQTT 协议有以下特性：

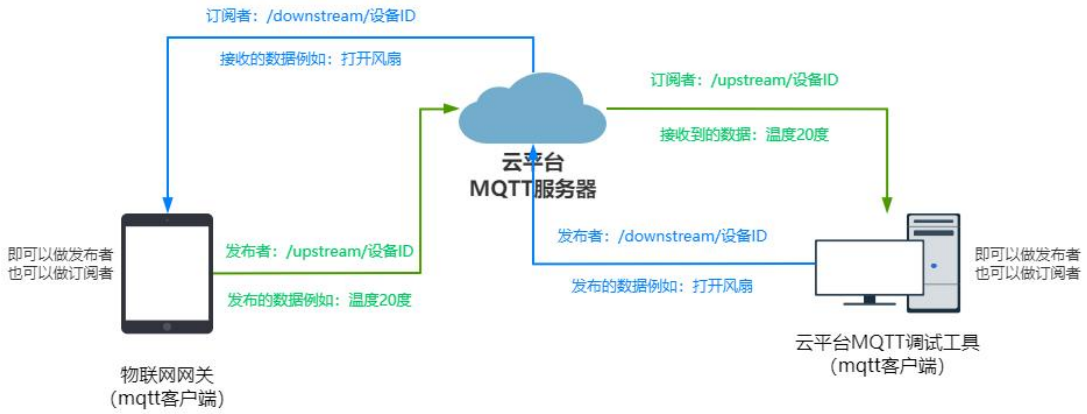
——基于 TCP 协议的应用层协议；

——采用 C/S 架构；

——使用订阅/发布模式，将消息的发送方和接受方解耦；

——提供 3 种消息的 QoS（Quality of Service）：至多一次，最少一次，只有一次；

——收发消息都是异步的，发送方不需要等待接收方应答。



MQTT 协议需要三个角色：发布者（publisher）、订阅者（subscriber）、代理服务器（broker）。以上图为例，两个客户端（物联网网关和云平台 mqtt 调试工具）两个客户端即可以做发布者也可以做订阅者。

1. 物联网网关**订阅**了主题为“/downstream/设备 ID”的消息，一旦云平台 mqtt 调试工具向 MQTT 服务器**发布**主题为“/downstream/设备 ID”的消息，那么 MQTT 服务器就会向订阅了这个主题的物联网网关**推送**该消息（比如打开风扇）。

2. 云平台 mqtt 调试工具**订阅**了主题为“/upstream/设备 ID”的消息，一旦物联网网关向 MQTT 服务器**发布**主题为“/upstream/设备 ID”的消息，那么 MQTT 服务器就会向订阅了这个主题的云平台 mqtt 调试工具**推送**该消息（比如温度 20 度）。

3. Mqtt 的消息格式可以是多种数据格式， Json 格式也可以作为 mqtt 的收发数据格式，JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于 ECMAScript (欧洲计算机协会制定的 js 规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

Json 格式通过键值对的方式组成 json 字符串如下：

{"data":{"value":"11"},"code":"11111","connecttype":"MQTT","key":"121212","deviceId":"TGZ001","trantype":"1","sensorId":"wendu"}

各字段含义如下，data:代表传感器的采集数据；code:代表传感器的地址；connecttype:代表连接类型；key:代表 web 端生成的唯一识别码；deviceId:代表设备 ID；trantype:代表该传感器属于什么类型；sensorId:代表传感器的标识。

网关端组成 json 数据格式使用 JSONObject 类的方法如下;

```
String str = "";
JSONObject json = new JSONObject();
JSONObject jsondata = new JSONObject();
try {
	json.put("deviceId", "TGZ001");
	json.put("sensorId", "wendu");
	json.put("key", "121212");
	jsondata.put("value", "11");
	json.put("data", jsondata);
	json.put("connecttype","MQTT");
	json.put("trantype", "1");
	json.put("code", "11111");
	str = json.toString();
} catch (JSONException e) {
	e.printStackTrace();
}
```

网关端解析 json 数据格式使用 JSONObject 类的方法如下;

```
JSONObject json;
JSONObject jsonObject;
try {
json = new JSONObject(msg);
jsonObject = new JSONObject(json.getJSONObject("data").toString());
String deviceId=json.getString("deviceId");
```

```
String sensorId=json.getString("sensorId");
String key=json.getString("key");
String connecttype=json.getString("connecttype");
String trantype=json.getString("trantype");
String code=json.getString("code");
String value=jsonObject.getString("value");
} catch (JSONException e) {
      e.printStackTrace();
}
```

4.4.4. 实验内容

1. Android 网关做 mqtt 客户端与云平台的 MQTT 调试工具客户端进行数据通讯。

   实现的效果如下：

   1) 连接 MQTT 服务器 ，断开后自动重连

```
//启动一个线程连接 mqtt 服务器
sub = new MqttThread
(server_ip,DataType.getMac(app), subscribe_topic, handler, app);
app.setSub(sub);
sub.start();
      @Override
      public void run() {
            try {
                  startReconnect();
                  if (clientId == null) {
                        clientId = Random();
                  }
                  sampleClient = new MqttClient(broker, clientId, persistence);
                  Log.v("mqtt", "Connecting to broker: " + broker+"       clientId==="+clientId);
                  //MQTT 的连接设置
                  connOpts = new MqttConnectOptions();
                  //设置是否清空 session,这里如果设置为 false 表示服务器会保留客户端的连接
记录，这里设置为 true 表示每次连接到服务器都以新的身份连接
                  connOpts.setCleanSession(true);
                  // 设置超时时间 单位为秒
```

```
                connOpts.setConnectionTimeout(10);
                // 设置会话心跳时间 单位为秒 服务器会每隔 1.5*20 秒的时间向客户端发送
个消息判断客户端是否在线，但这个方法并没有重连的机制
                connOpts.setKeepAliveInterval(50);
                sampleClient.setCallback(new MqttCallback() {
                    @Override
                    public void connectionLost(Throwable throwable) {
                        try {
                            sampleClient.connect(connOpts);
                            sampleClient.subscribe(topic);
                        } catch (MqttException e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void messageArrived(String topic, MqttMessage mqttMessage) throws
Exception {
                        Log.v("mqtt", "接受到 Mqtt 数据 1111：" + mqttMessage.toString());
                        try {
                            if (app.getiMqttInterface() != null) {
                                app.getiMqttInterface().RecMqttMsg(mqttMessage.toString());
                            }
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
                        System.out.println("deliveryComplete");
                    }
                });
                connect();
            } catch (MqttException me) {
                System.out.println("reason " + me.getReasonCode());
                System.out.println("msg " + me.getMessage());
                System.out.println("loc " + me.getLocalizedMessage());
                System.out.println("cause " + me.getCause());
                System.out.println("excep " + me);
            }
        }

        /**
         * 连接服务器
         */
        private void connect() {
            try {
                sampleClient.connect(connOpts);
                sampleClient.subscribe(topic, 2);
                handler.sendEmptyMessage(0x01);
            } catch (Exception e) {
                e.printStackTrace();
                handler.sendEmptyMessage(0x02);
            }
        }
```

2) 断开服务器

```
/**
 * 断开连接
 */
public void disconnect(){
    if (sampleClient!=null&&sampleClient.isConnected()) {
        try {
            sampleClient.disconnect();
            handler.sendEmptyMessage(0x03);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
}
```

3) 发送数据

```
/**
 * 通过 mqtt 发送数据
 *
 * @param topic     发送主题
 * @param message  发送信息内容
 */
public void sendMessage(String topic, String message) {
    MqttMessage mqMessage = new MqttMessage(message.getBytes());
    mqMessage.setQos(2);
    if (sampleClient == null || !sampleClient.isConnected()) {
        run();
    }

    // 发布自己的消息
    try {
        sampleClient.publish(topic, mqMessage);
        System.out.println("Mqtt 数据发送到：" + topic + ",内容是：" + mqMessage);
    } catch (MqttPersistenceException e) {
        e.printStackTrace();
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

4) 接收数据

```
/**
 * 接收 mqtt 推送
 * @param msg
 */
@Override
public void RecMqttMsg(String msg) {
    Message message=new Message();
    message.what=4;
    message.obj=msg;
    mHandler.sendMessage(message);
}
/**
```

```
        * 更新 ui
        */
       Handler mHandler =new Handler(){
            public void handleMessage(Message msg){
                switch (msg.what){
                    case 1:
                        break;
                    case 2:
                        break;
                    case 3:
                        break;
                    case 4://mqtt 消息推送测试
                        String topic=et_subscribe_topic.getText().toString();
                        String mssg= (String)msg.obj;
                        tv_receive.append("接收到订阅主题为   "+topic+"   的数据："+mssg +
"\r\n");
                        int offset=tv_receive.getLineCount()*tv_receive.getLineHeight();
                        if(offset>tv_receive.getHeight()){
                            tv_receive.scrollTo(0,offset-tv_receive.getHeight());
                        }
                        break;
                }
            }
        };
```

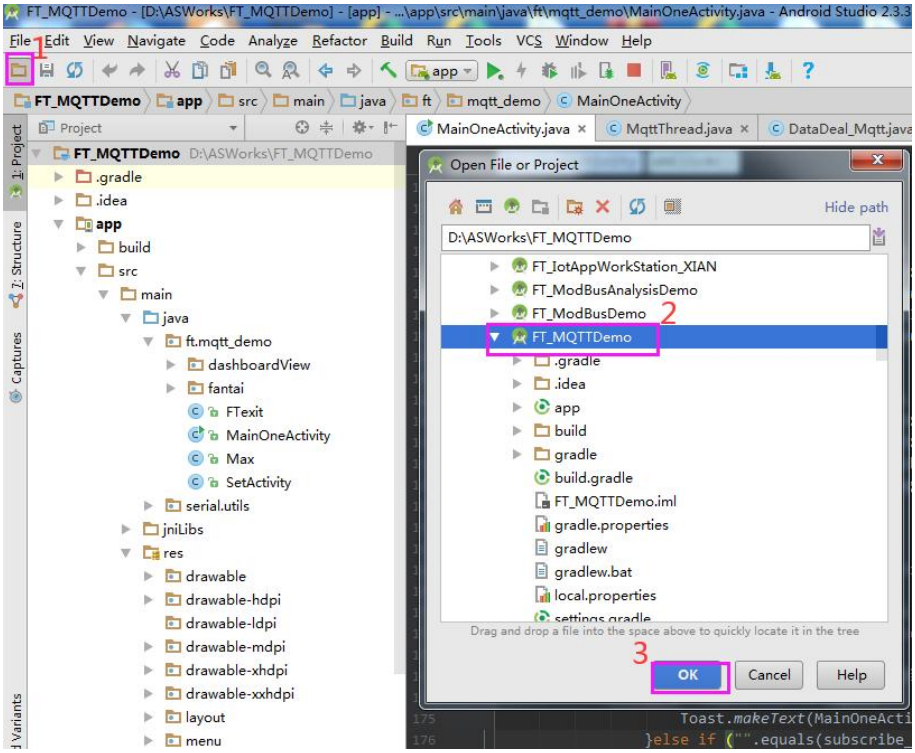
4.4.5. 实验步骤

1. 接入网络

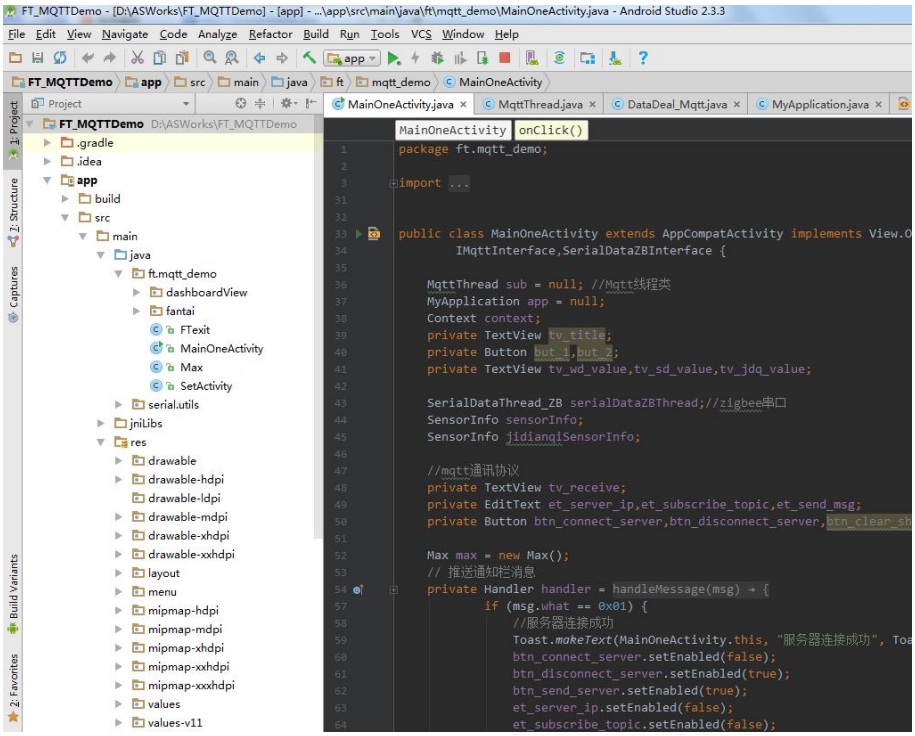
网关通过 WiFi 连接路由器，保证该网关可以连接互联网。如下图所示：

2. 打开项目

使用 Android studio 工具打开光盘资料（路径名）路径下的工程名为 FT_MQTTDemo 如下图所示：

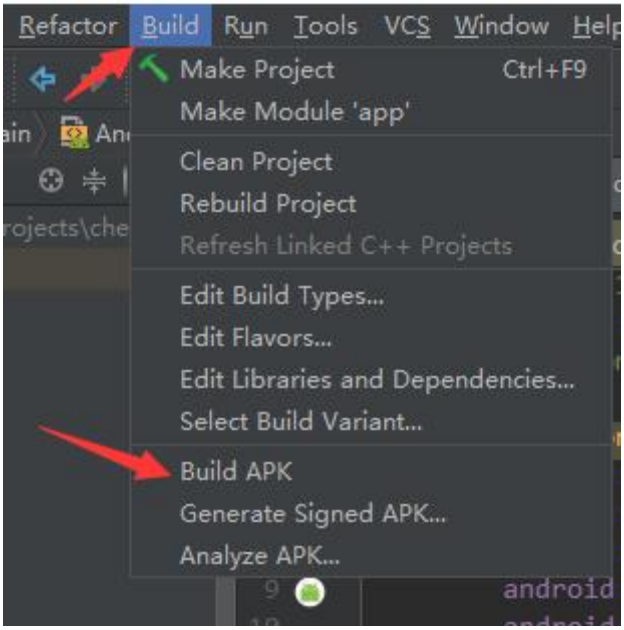


按照上图步骤引入工程项目至 Android studio 开发工具中。如下图所示：

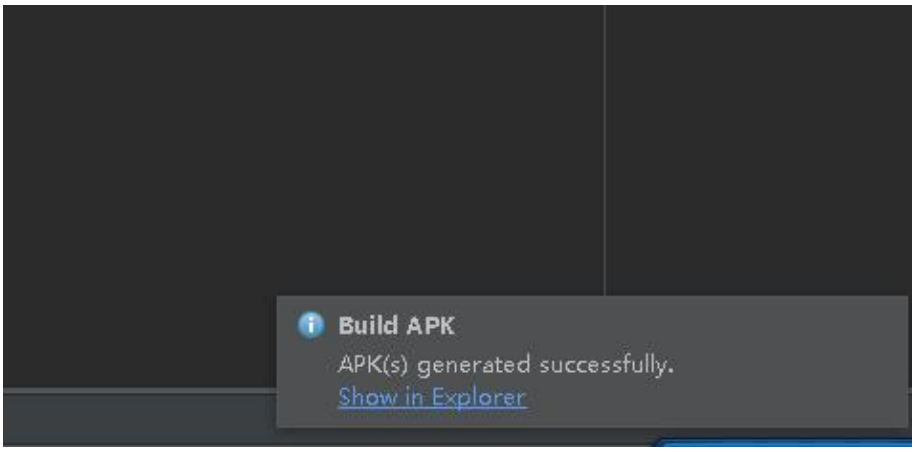


3. 编译生成 APK

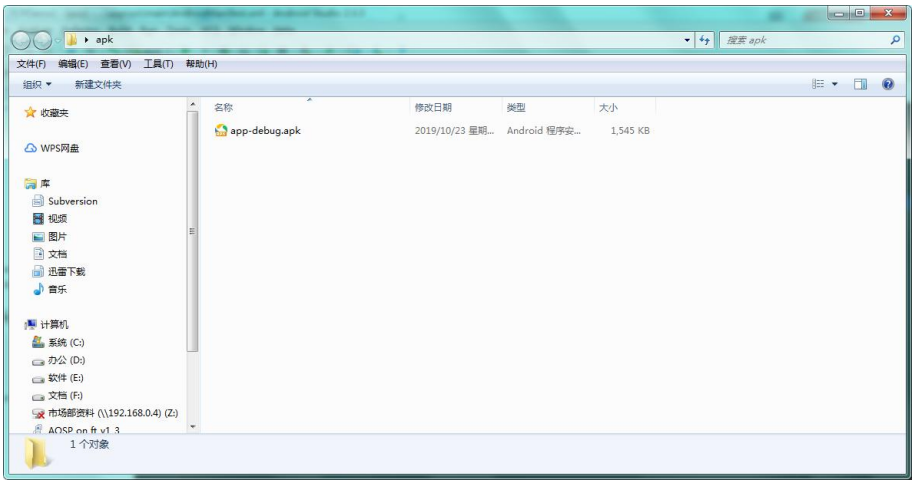
依次点击 Build—》Build APK，如下图所示：



等待编译结束右下角会弹出提示框如下图所示：



点击蓝色字体便会打开生成 apk 的文件夹，如下图所示：

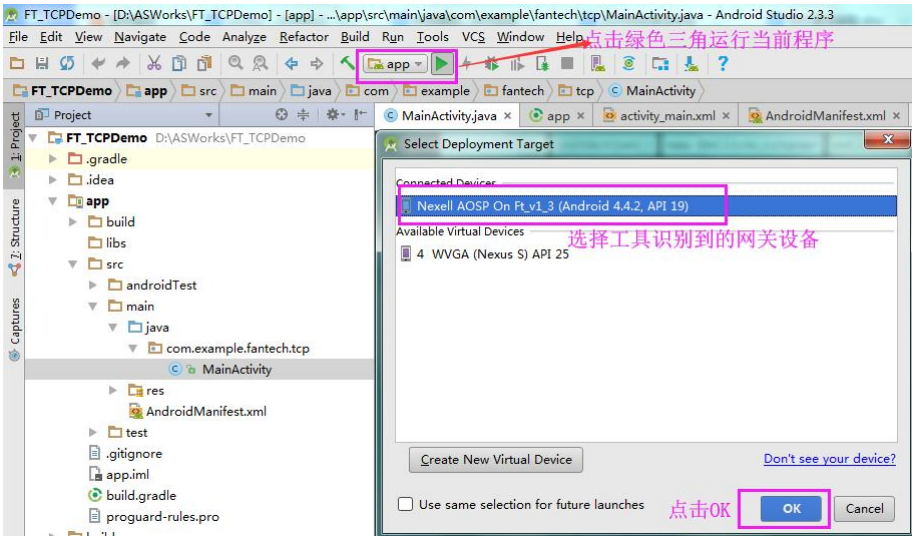


将生成的 APK，拷贝到 TF 内存卡中，将 TF 卡插到 A9 网关上，将 A9 网关

上电，将这个 APK 文件安装到 A9 网关上，点击运行即可。

4. 直接运行程序至网关（另一种生成 apk 的方式）

首先拿一根 Android 手机数据线，一头插入网关的 OTG 接口，一头插入 PC 端的 USB 接口。接着如下图操作：



4.4.6. 实验结果

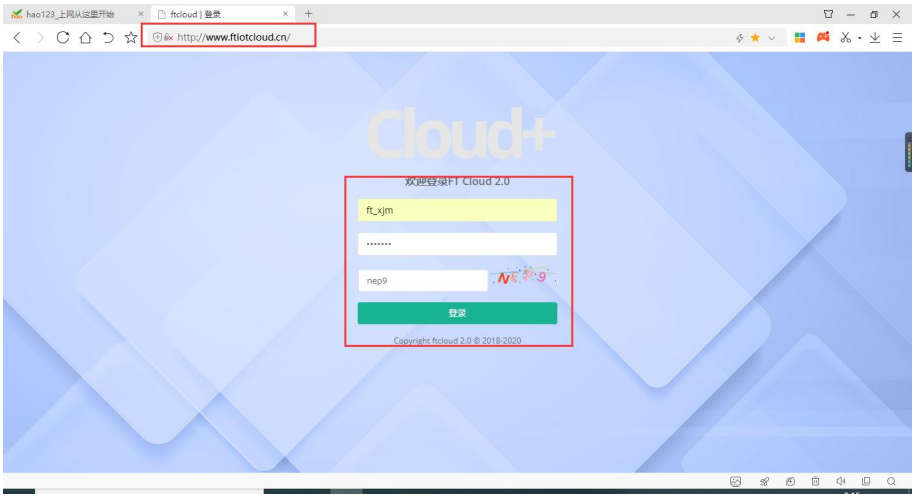
软件图标如下图所示：

点击打开如下图所示：

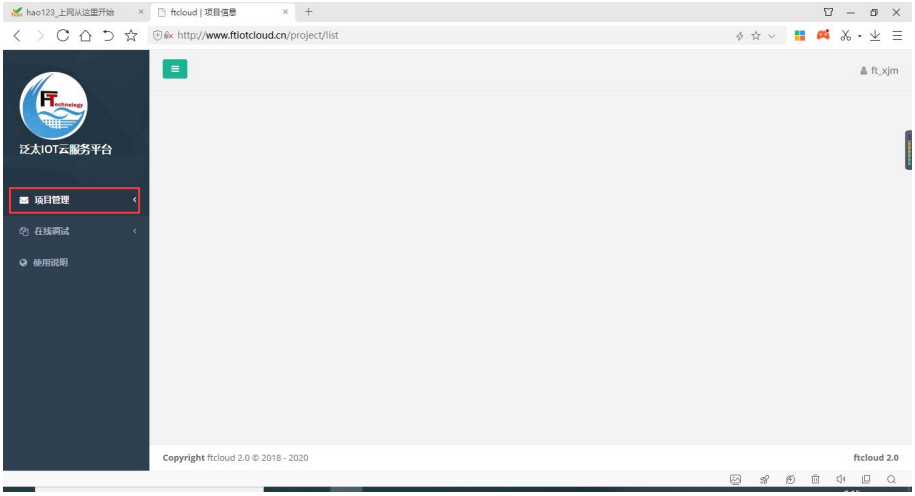
该 MQTT 网络通信网关做客户端，与云平台的 mqtt 调试工具的客户端

进行数据的收发。

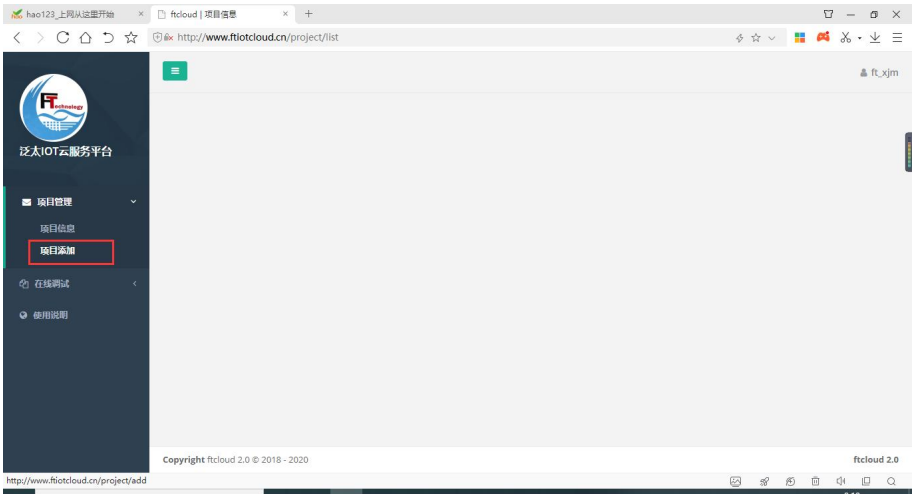
1) MQTT 服务器部署在阿里云服务器上，云平台端 mqtt 调试工具当做 mqtt 的一个客户端，连接该 mqtt 服务器，首先登陆云平台，打开浏览器输入云平台网址“http://www.ftiotcloud.cn/”输入项目提供的用户名密码，进行登陆。如下图



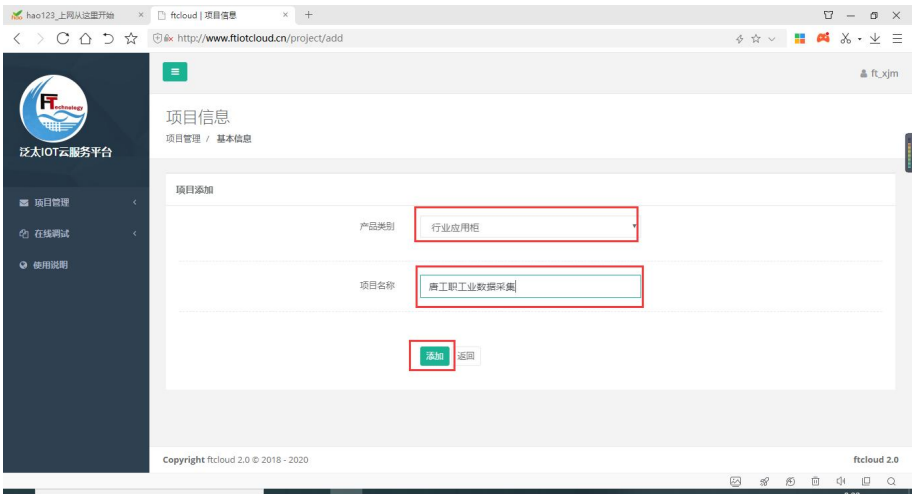
登陆成功后显示如下页面，点击左侧列表中的项目管理选项。



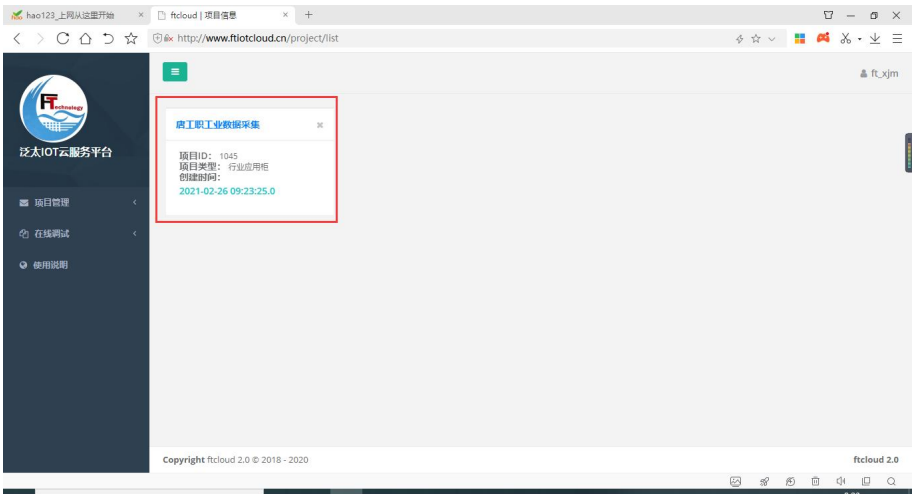
点击项目添加，如下图



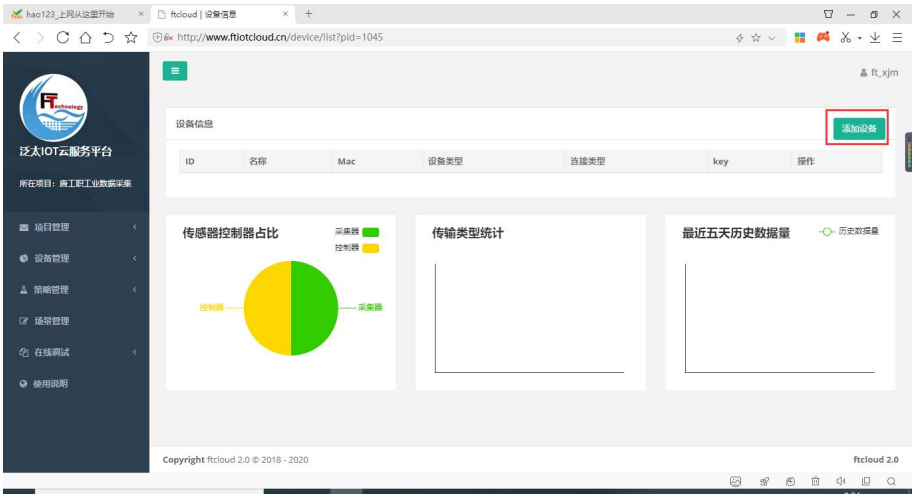
产品类别选择行业应用柜，项目名称：自定义输入，点击添加按钮。如下图



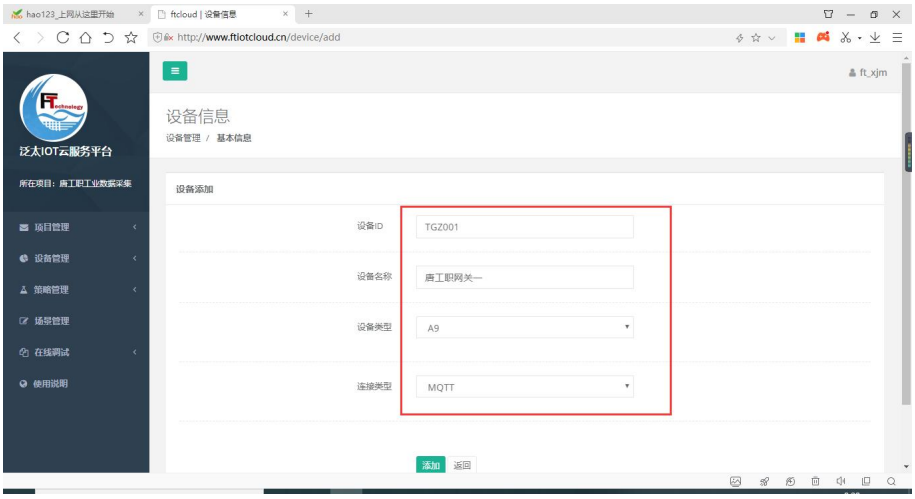
项目添加成功后如下图，点击蓝色项目名称，进入项目。



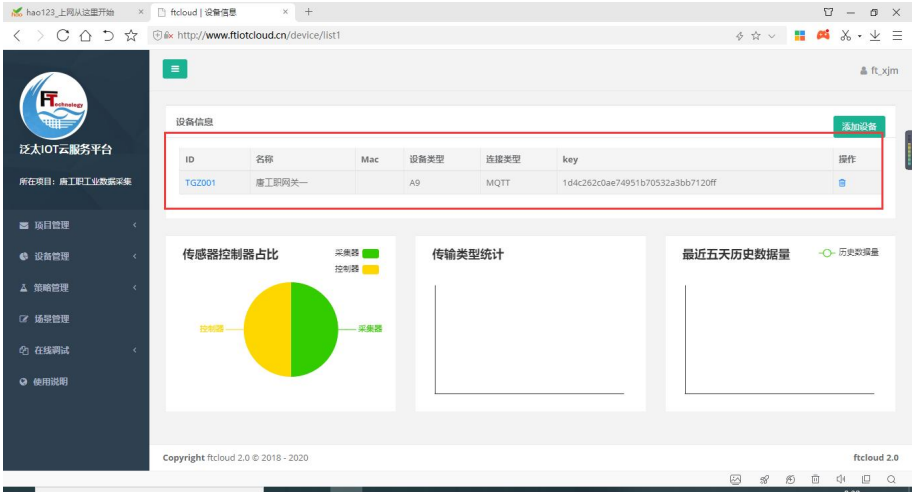
点击右侧的添加设备按钮。



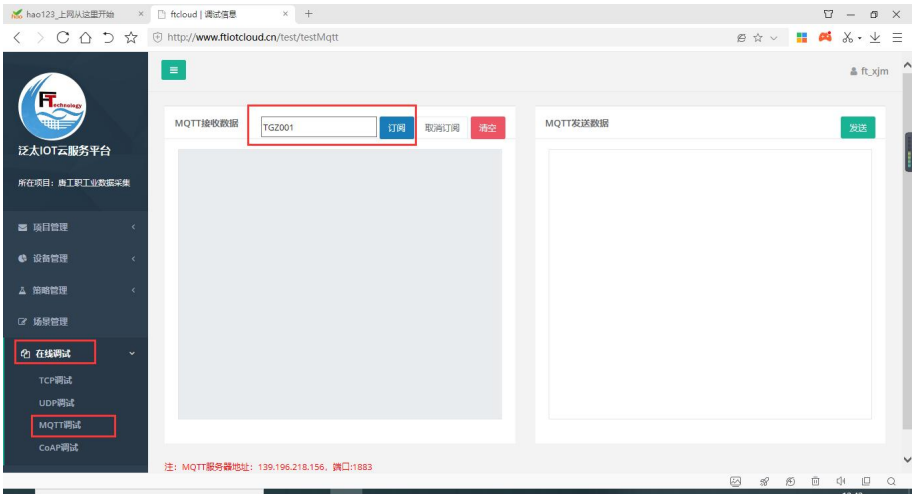
输入设备 ID，设备名称，选择设备类型，连接类型，点击添加按钮。如下图。



添加成功后如下图，点击蓝色设备 ID，进入



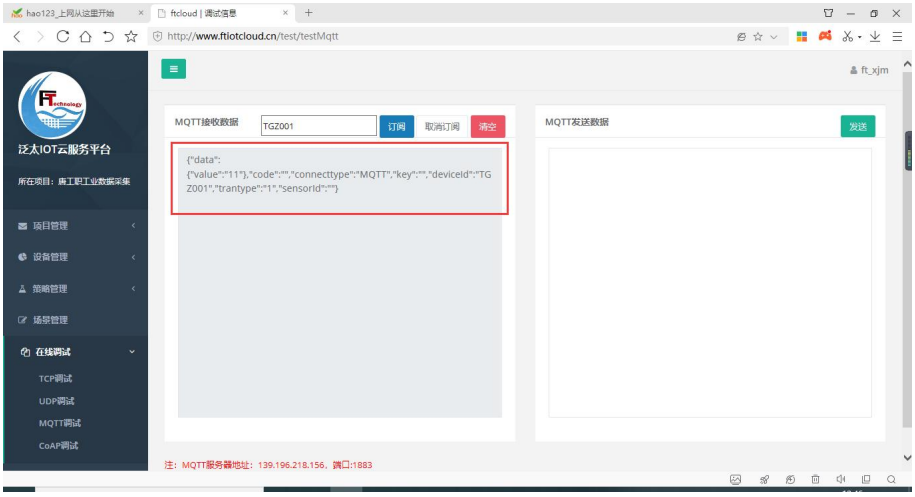
点击左侧菜单栏中在线调试项，展开子菜单,点击 MQTT 调试，进入 MQTT 调试页面，输入刚才添加的设备 ID，点击订阅按钮，如图所示



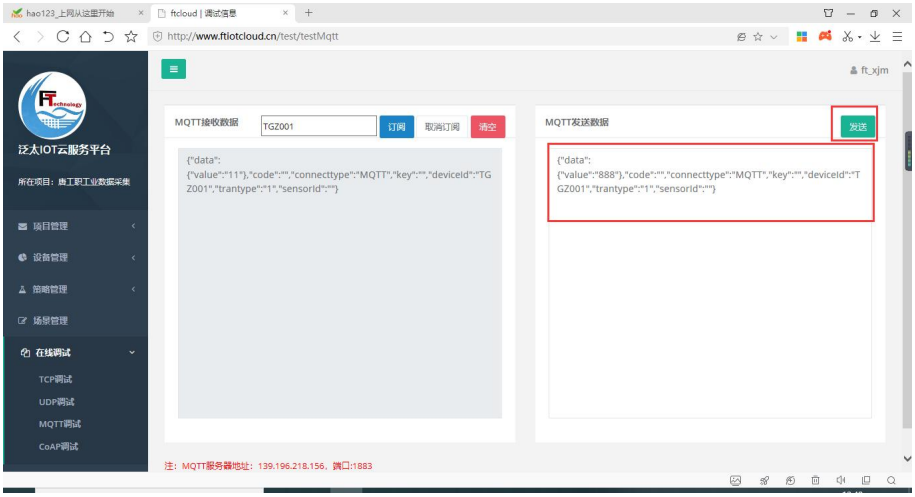
2) MQTT 服务器部署在阿里云服务器上，所以客户端要连接的 mqtt 服务器地址为 tcp://www.ftiotcloud.cn:1883。设备 ID 是在云平台添加的设备 ID，订阅主题为/downstream/设备 ID，上行发送主题为/upstream/设备 ID。然后点击连接按钮。如下图

连接成功后如下图所示，发布编辑框会自动填充 json 格式的数据，用于和云平台进行 MQTT 通信的测试数据。

点击发送按钮，云平台客户端 mqtt 调试工具会接收到网关发送的数据。如下图



将云平台客户端 mqtt 调试工具接收到的数据复制到右侧的发送测试框内修改一下 value 的值，点击发送按钮。如下图



网关端会收到该数据。如下图

到此，平台 MQTT 协议测试实验功能完成。

4.5. 工业以太网网关 CoAP 协议接入平台实验

4.5.1. 实验目的

- 了解 Android CoAP 网络通信访问协议
- 掌握 Android CoAP 网络通信编程方法

4.5.2. 实验环境

1. 软件：用户 PC（Microsoft Windows XP 以上系统平台）1 台，确保已正确安装及配置 jdk、Android Studio 开发环境、浏览器（用于登陆云平台，使用 CoAP 调试工具）。
2. 硬件：物联网网关 1 个。

4.5.3. 实验原理

1. CoAP 约束应用协议（Constrained Application Protocol）是一种专用于受限设备的 Internet 应用协议，如 RFC 7252 所定义，它使那些被称为“节点”的受约束设备能够使用类似的协议与更广泛的 Internet 进行通信。CoAP 被设计用

于同一受限网络（例如，低功耗、有损网络）上的设备之间、设备和因特网上的一般节点之间以及由因特网连接的不同受限网络上的设备之间使用。CoAP 也被用于其他机制，如移动通信网络上的 SMS。

2. CoAP 的消息格式可以是多种数据格式， Json 格式也可以作为 CoAP 的收发数据格式，JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于 ECMAScript (欧洲计算机协会制定的 js 规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

Json 格式通过键值对的方式组成 json 字符串如下：

{"data":{"value":"11"},"code":"11111","connecttype":"CoAP","key":"121212","deviceId":"TGZ001","trantype":"1","sensorId":"wendu"}

各字段含义如下，data:代表传感器的采集数据；code:代表传感器的地址；connecttype:代表连接类型；key:代表 web 端生成的唯一识别码；deviceId:代表设备 ID；trantype:代表该传感器属于什么类型；sensorId:代表传感器的标识。

网关端组成 json 数据格式使用 JSONObject 类的方法如下;

```
String str = "";
JSONObject json = new JSONObject();
JSONObject jsondata = new JSONObject();
try {
	json.put("deviceId", "TGZ001");
	json.put("sensorId", "wendu");
	json.put("key", "121212");
	jsondata.put("value", "11");
	json.put("data", jsondata);
	json.put("connecttype","CoAP");
	json.put("trantype", "1");
	json.put("code", "11111");
	str = json.toString();
} catch (JSONException e) {
	e.printStackTrace();
}
```

网关端解析 json 数据格式使用 JSONObject 类的方法如下。

```
JSONObject json;
JSONObject jsonObject;
try {
json = new JSONObject(msg);
jsonObject = new JSONObject(json.getJSONObject("data").toString());
String deviceId=json.getString("deviceId");
String sensorId=json.getString("sensorId");
String key=json.getString("key");
```

```
String connecttype=json.getString("connecttype");
String trantype=json.getString("trantype");
String code=json.getString("code");
String value=jsonObject.getString("value");
} catch (JSONException e) {
      e.printStackTrace();
}
```

4.5.4. 实验内容

1. 通过 Android 网关当做客户端与云平台的 CoAP 调试工具进行数据通讯

   实现的效果如下：

5) 连接 CoAP 服务器

```
/**
 * coap 服务器连接
 */
private void connectCoap(){
      URI uri = null;
      try {
            uri = new URI(url+subscribeTopic);
      } catch (URISyntaxException e) {
            e.printStackTrace();
      }
      client = new CoapClient(uri);
      client.observe(new CoapHandler() {
            @Override
            public void onLoad(CoapResponse response) {
                  boolean isSuccess=response.isSuccess();
                  if (!isSuccess) {//连接失败
                        Message message=new Message();
                        message.what=2;
                        mHandler.sendMessage(message);
                  }else if (isSuccess) {//连接成功
                        Message message=new Message();
                        message.what=1;
```

```
                    mHandler.sendMessage(message);
                }

                if (response.getResponseText() != null) {
                    Message message=new Message();
                    message.what=4;
                    message.obj=response.getResponseText();
                    mHandler.sendMessage(message);
                }
            }
            @Override
            public void onError() {

            }
        });
    }
```

6) 发送数据

```
    /**
     * 发送数据到 coap 服务器
     */
    private void sendMsgCoap(String send_msg){
        URI uri = null;
        try {
            uri = new URI(url+releaseTopic);
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
        CoapClient client = new CoapClient(uri);
        client.post(send_msg,MediaTypeRegistry.TEXT_PLAIN);
    }
```

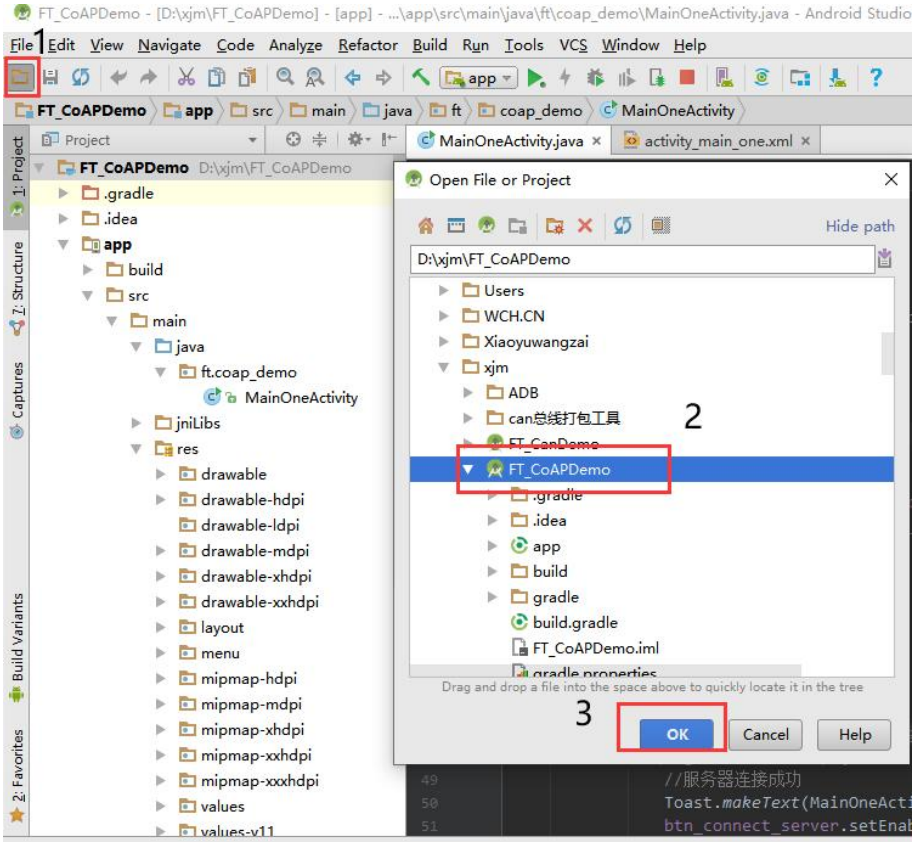
4.5.5. 实验步骤

1. 接入网络

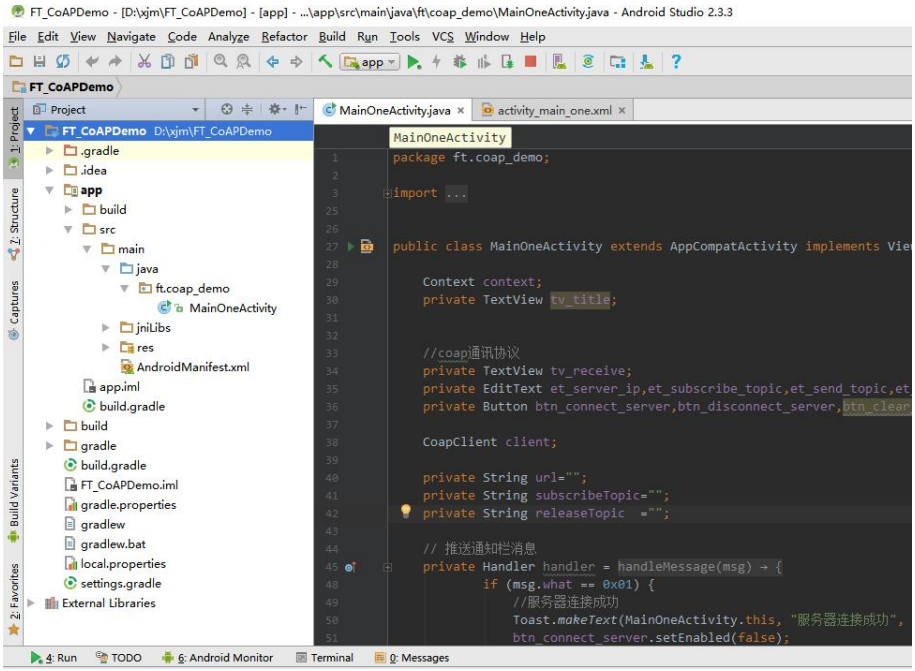
网关通过 WiFi 连接接入路由器下，保证该网关可以连接互联网。如下图所示：

2. 打开项目

使用 Android studio 工具打开光盘资料（路径名）路径下的工程名为 FT_CoAPDemo 如下图所示：

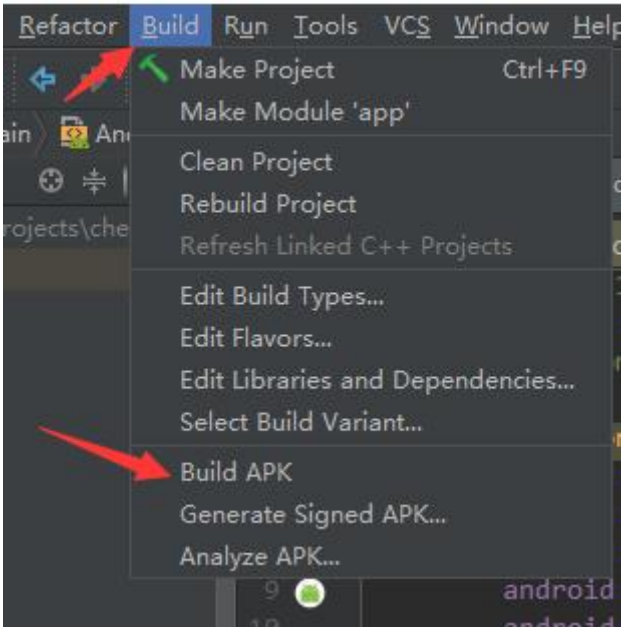


按照上图步骤引入工程项目至 Android studio 开发工具中。如下图所示：

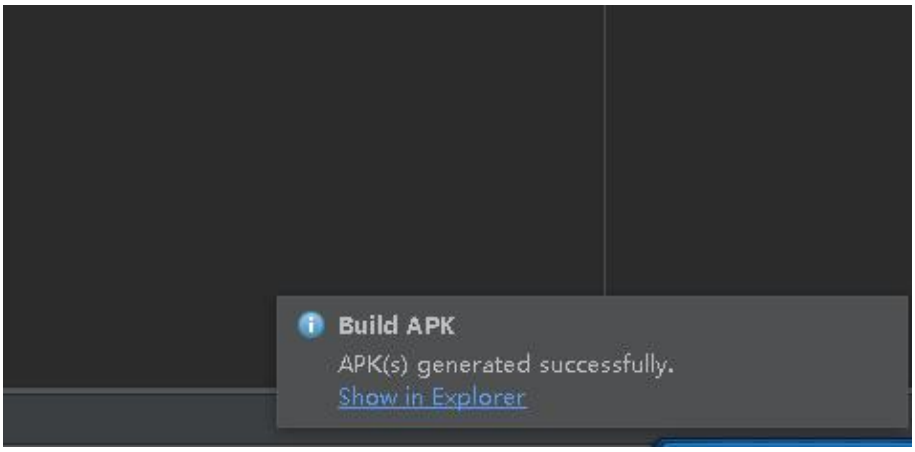


3. 编译生成 APK

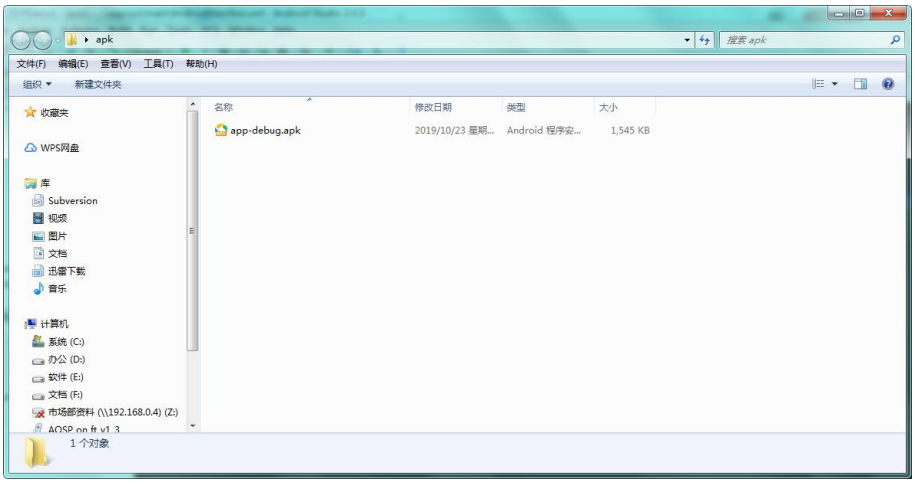
依次点击 Build—》Build APK，如下图所示：



等待编译结束右下角会弹出提示框如下图所示：



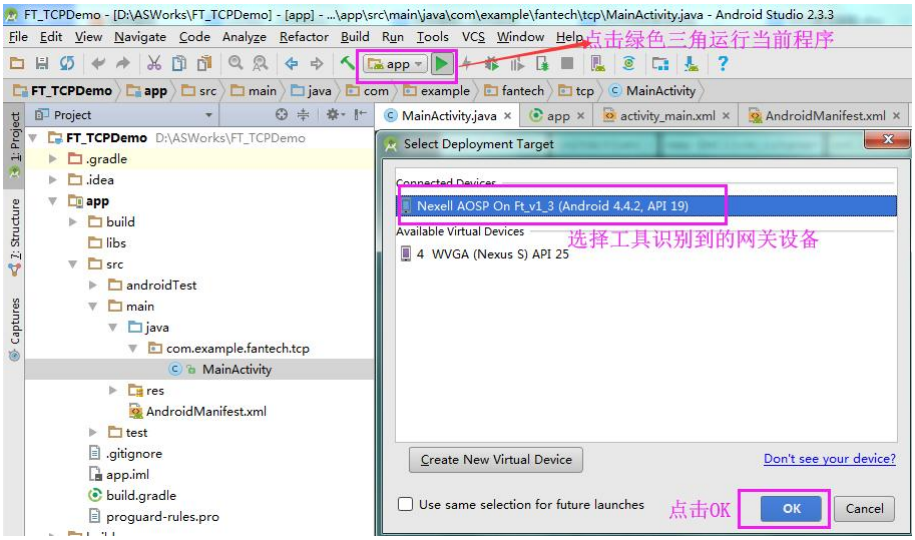
点击蓝色字体便会打开生成 apk 的文件夹，如下图所示：



将生成的 APK，拷贝到 TF 内存卡中，将 TF 卡插到 A9 网关上，将 A9 网关上电，将这个 APK 文件安装到 A9 网关上，点击运行即可。

4. 直接运行程序至网关（另一种生成 apk 的方式）

首先拿一根 Android 手机数据线，一头插入网关的 OTG 接口，一头插入 PC 端的 USB 接口。接着如下图操作：



4.5.6. 实验结果

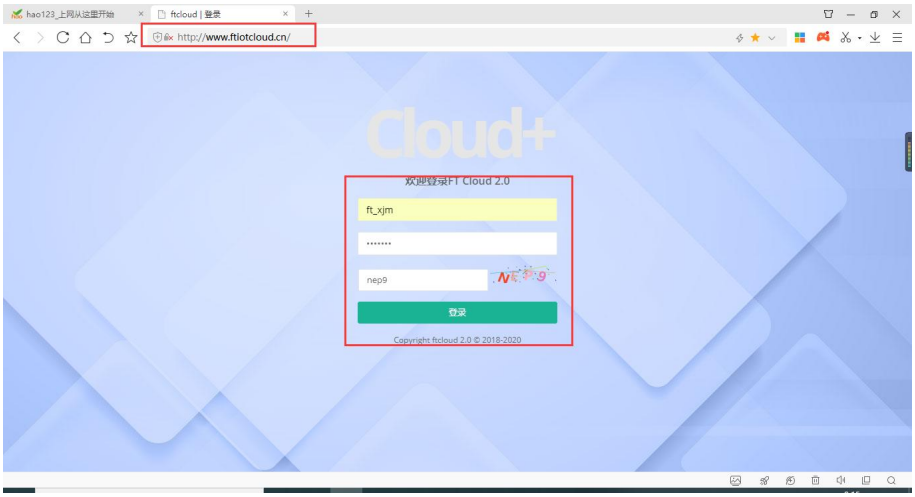
软件图标如下图所示：

点击打开如下图所示：

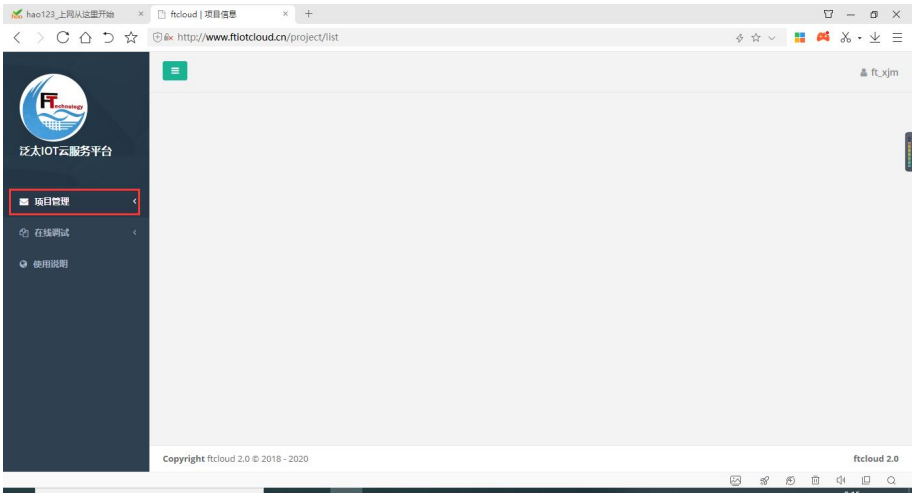
该 CoAP 网络通信网关做客户端，与云平台的 CoAP 调试工具的客户端进行数据的收发。

1) CoAP 服务器部署在阿里云服务器上，云平台端 CoAP 调试工具当做 CoAP 的一个客户端，连接该 CoAP 服务器，首先登陆云平台，打开浏览器输

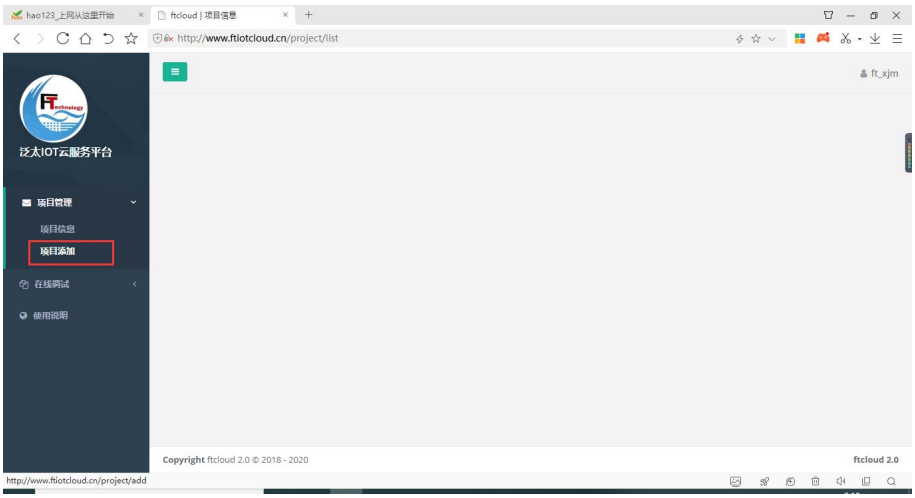
入云平台网址“http://www.ftiotcloud.cn/”输入项目提供的用户名密码，进行登陆。如下图



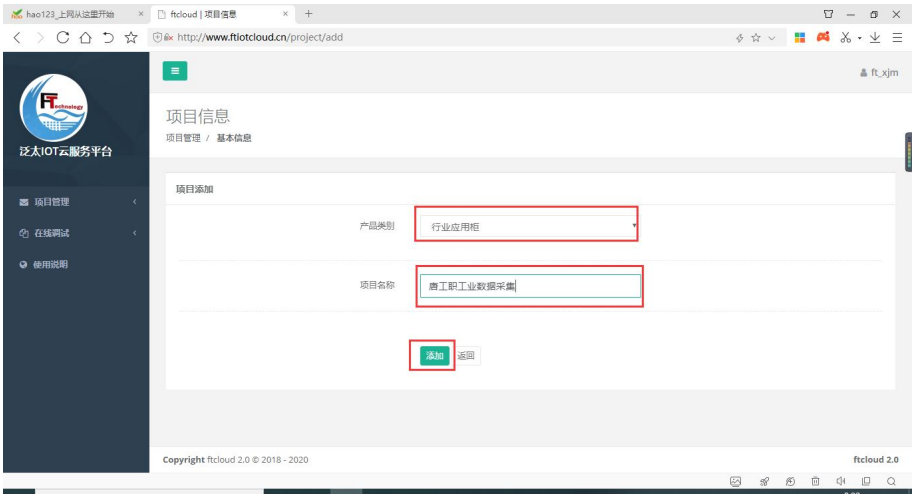
登陆成功后显示如下页面，点击左侧列表中的项目管理选项。



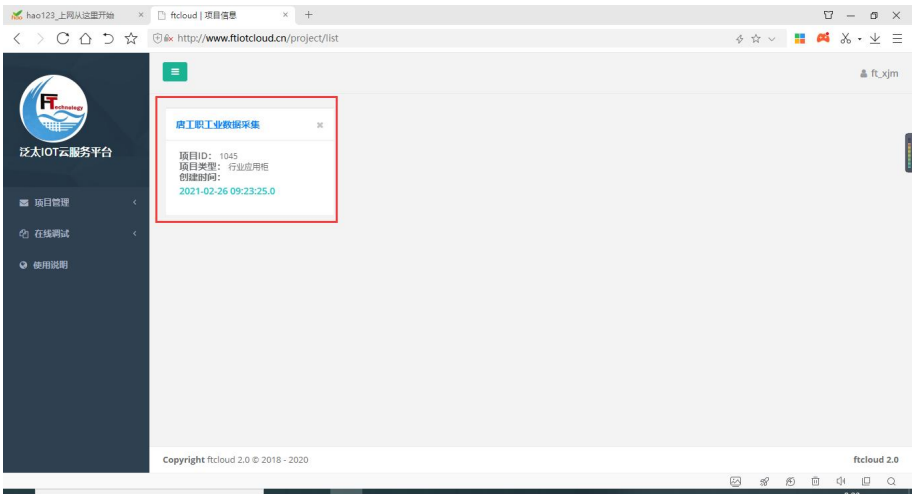
点击项目添加，如下图



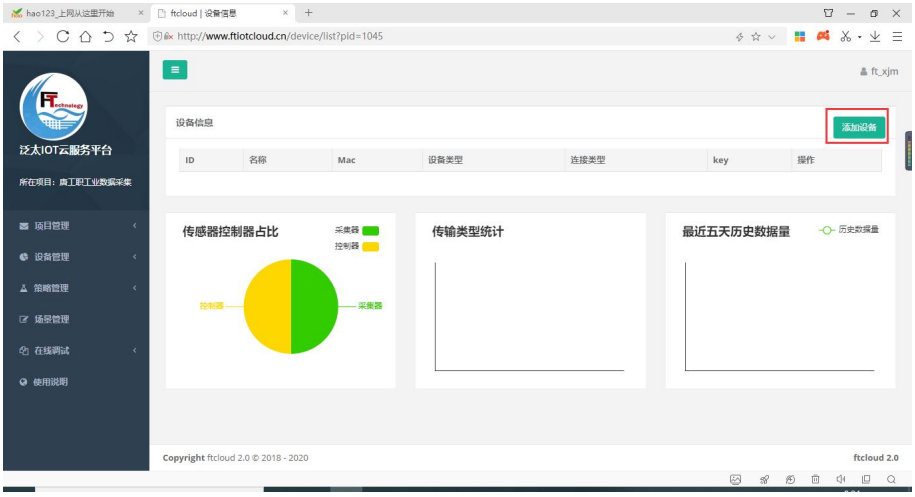
产品类别选择行业应用柜，项目名称：自定义输入，点击添加按钮。如下图



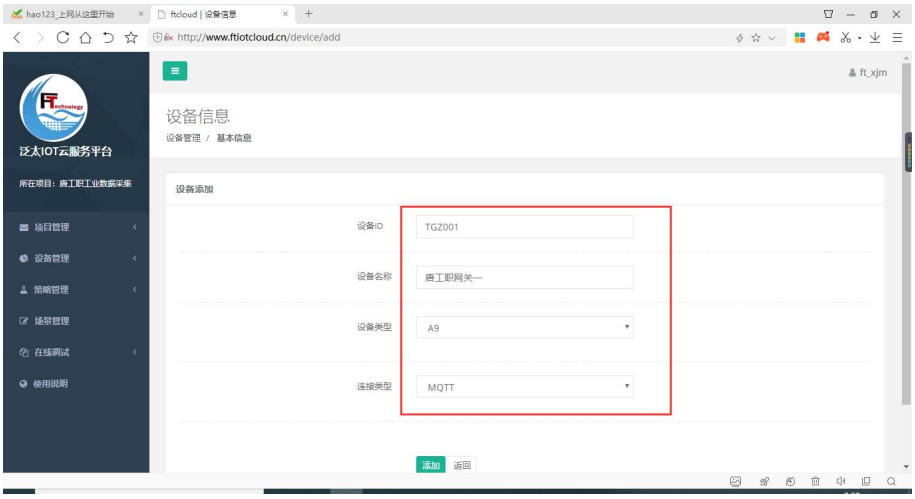
项目添加成功后如下图，点击蓝色项目名称，进入项目。



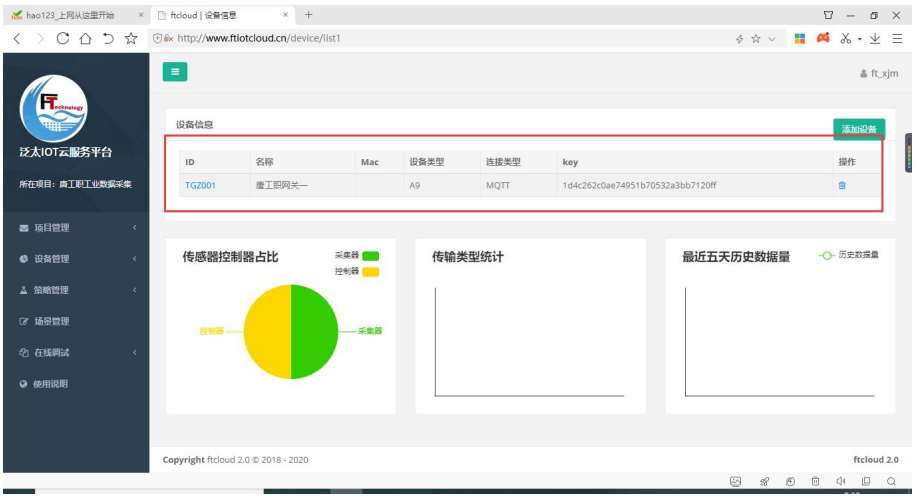
点击右侧的添加设备按钮。



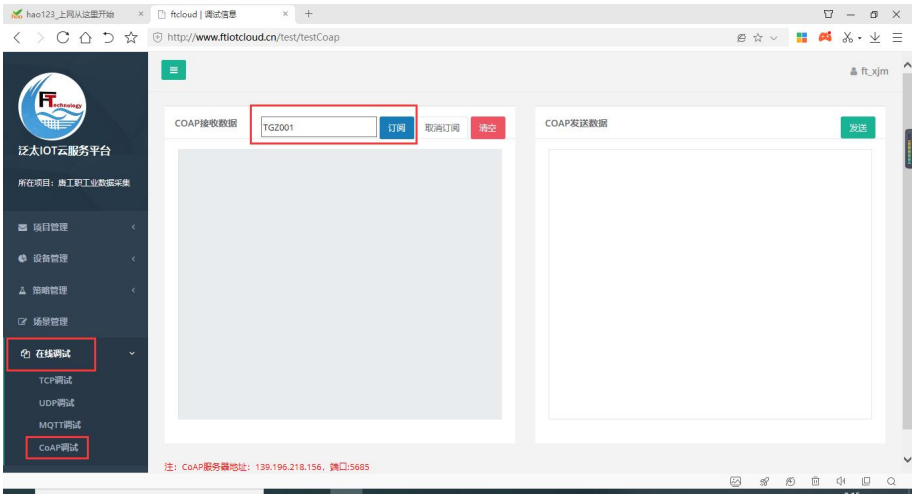
输入设备 ID，设备名称，选择设备类型，连接类型，点击添加按钮。如下图。



添加成功后如下图，点击蓝色设备 ID，进入



点击左侧菜单栏中在线调试项，展开子菜单,点击 CoAP 调试，进入 CoAP 调试页面，输入刚才添加的设备 ID，点击订阅按钮，如图所示

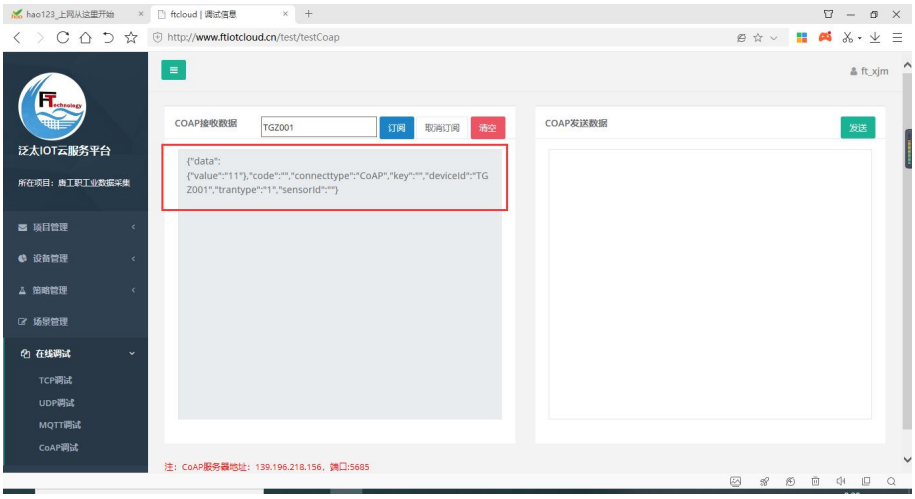


2) CoAP 服务器部署在阿里云服务器上，端口配置为 5685，所以客户端要连接的 CoAP 服务器地址为 coap://139.196.218.156:5685。设备 ID 是在云平台

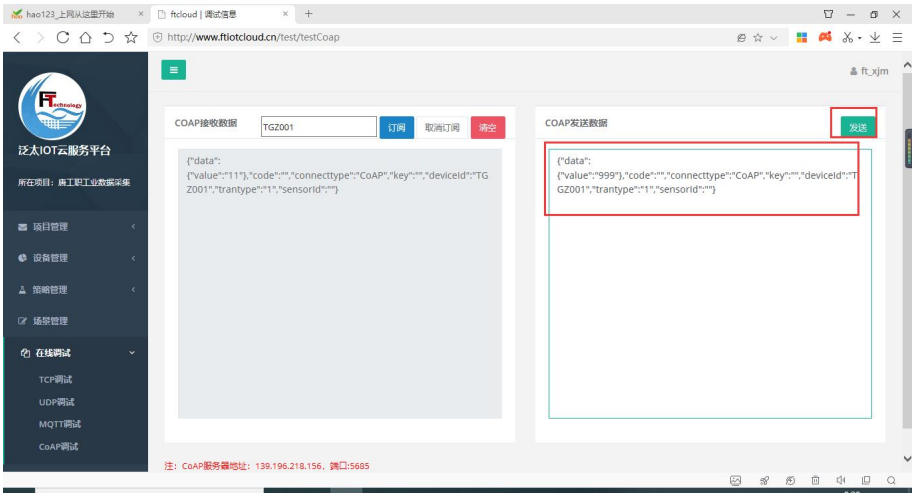
添加的设备 ID，订阅主题为“/api/设备 ID/downstream”，上行发送主题为“/api/设备 ID/upstream”。然后点击连接按钮。如下图

连接成功后如下图所示，发布编辑框会自动填充 json 格式的数据，用于和云平台进行 CoAP 通信的测试数据。

点击发送按钮，云平台客户端 CoAP 调试工具会接收到网关发送的数据。如下图



将云平台客户端 CoAP 调试工具接收到的数据复制到右侧的发送测试框内修改一下 value 的值，点击发送按钮。如下图



网关端会收到该数据。如下图

到此，CoAP 协议测试实验功能完成。

4.6. 工业以太网网关 Restful 接口平台调用

4.6.1. 实验目的

- 了解 Android Restful 网络通信访问协议
- 掌握 Android Restful 接口使用编程方法

4.6.2. 实验环境

1. 软件：用户 PC（Microsoft Windows XP 以上系统平台）1 台，确保已正确安装及配置 jdk、Android 开发环境，浏览器用于登陆云平台。
2. 硬件：物联网网关 1 个。

4.6.3. 实验原理

REST 全称是 Representational State Transfer，中文意思是表述（编者注：通常译为表征）性状态转移。

REST 本身并没有创造新的技术、组件或服务，而隐藏在 RESTful 背后的理念就是使用 Web 的现有特征和能力， 更好地使用现有 Web 标准中的一些准则和约束。虽然 REST 本身受 Web 技术的影响很深， 但是理论上 REST 架构风格并不是绑定在 HTTP 上，只不过目前 HTTP 是唯一与 REST 相关的实例。 所以我们这里描述的 REST 也是通过 HTTP 实现的 REST。

Restful 接口遵循统一接口原则，

4.6.4. 实验内容

1. 通过 Android 网关 Restful 获取云平台的传感器数据

实现的效果如下：

7) 请求服务器方法

```
public static String executeGetMethod(String path, String authorization) {
        String response = "";
        try {
                URL url = new URL(path);
                HttpURLConnection connection = (HttpURLConnection)
        url.openConnection();
                connection.setRequestMethod("GET");
                connection.setRequestProperty("Charset", "UTF-8");
                connection.setRequestProperty("Content-Type", "application/json");
                connection.setConnectTimeout(5000);
                connection.setReadTimeout(5000);
                connection.connect();

                // 获得返回值
                InputStream in = connection.getInputStream();
                response = getResponse(in);
                Log.i("response", response);
                connection.disconnect();
        } catch (MalformedURLException e) {
                e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
        return response;
}
```

8) 创建线程调用 Restful 接口获取数据

```
/**
 * 通过 HTTP Restful 获取传感器数据
 */
public class GetHttpData extends Thread {
        private MyApplication application;
        private String url;
        private MainOneActivity callBack; //主函数回调
        public GetHttpData(MainOneActivity callBack,MyApplication application,String url) {
                this.callBack = callBack;
```

```
        this.application = application;
        this.url = url;
    }

    @Override
    public void run() {
        super.run();
        String jsonArraymsg = HttpUtil.executeGetMethod(url, "");
        if (jsonArraymsg == null||"".equals(jsonArraymsg)||jsonArraymsg.length()<3) {
            callBack.receiveFileStr(false);
        }else{
            try {
                JSONArray jsonArray = new JSONArray(jsonArraymsg);
                JSONObject jsonObject = null;
                List<SensorInfo> infos = new ArrayList<SensorInfo>();
                SensorInfo sensorInfo = null;
                byte[] sensorkey_all = null;
                byte[] sensorkey = null;
                for (int i = 0; i < jsonArray.length(); i++) {
                    try {
                        jsonObject = jsonArray.getJSONObject(i);
                        sensorInfo = new SensorInfo();
                        sensorInfo.setValue(jsonObject.getString("sei_value"));
                        sensorInfo.setIsControl(jsonObject.getInt("sti_control"));
                        sensorInfo.setCode(jsonObject.getString("sei_mac"));
                        sensorInfo.setTypeId(jsonObject.getString("sti_id"));
                        sensorInfo.setType(jsonObject.getString("sti_name"));
                        sensorInfo.setTrantype(jsonObject.getString("tti_name"));
                        sensorInfo.setTrantypeId(jsonObject.getString("tti_id"));
                        sensorInfo.setSensorId(jsonObject.getString("sei_id"));
                        sensorInfo.setDeviceId(jsonObject.getString("di_id"));
                        infos.add(sensorInfo);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                application.setSensorInfoList(infos);
                application.sensorList.addAll(infos);
                //通知线程调用者 run 方法执行结果
                callBack.receiveFileStr(true);
                for (int i = 0; i < infos.size(); i++) {
                    System.out.println("循环查询数据第"+i+"条："
                            +"sensorId:"+infos.get(i).getSensorId()
                            +",传感器码:"+infos.get(i).getCode()
                            +",trantype:"+infos.get(i).getTrantype()
                            +",type:"+infos.get(i).getType());

                }
                infos = null;
            } catch (Exception e) {
                e.printStackTrace();
                callBack.receiveFileStr(false);
            }
        }
    }
}
```

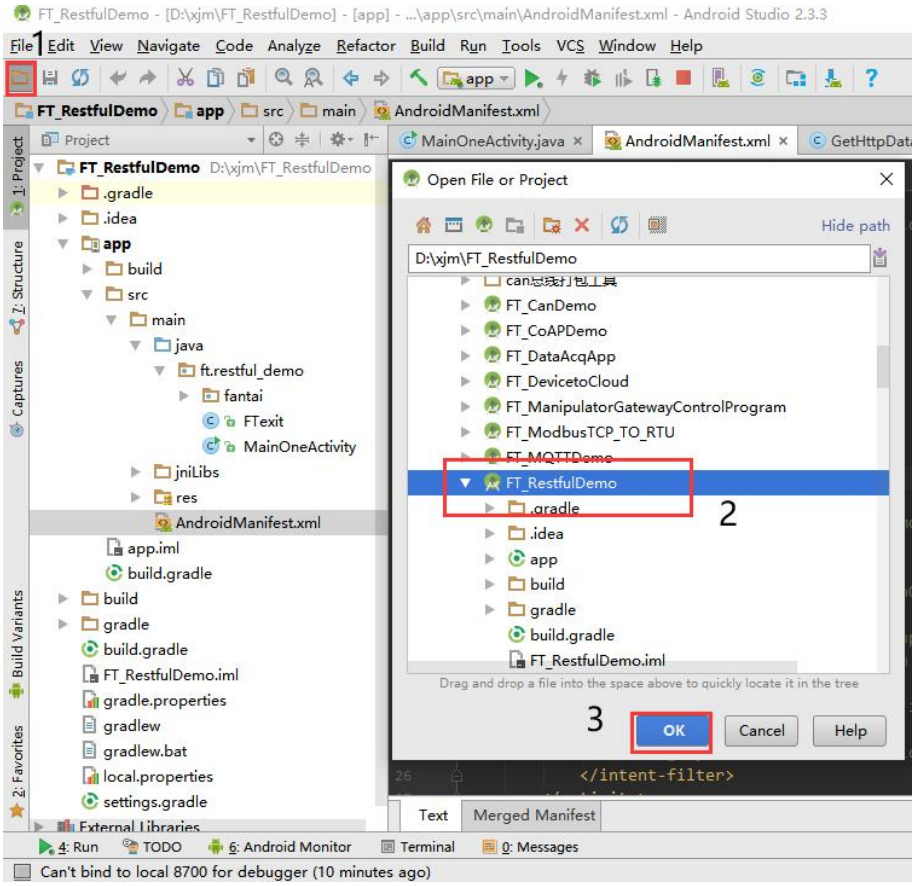
4.6.5. 实验步骤

1. 接入网络

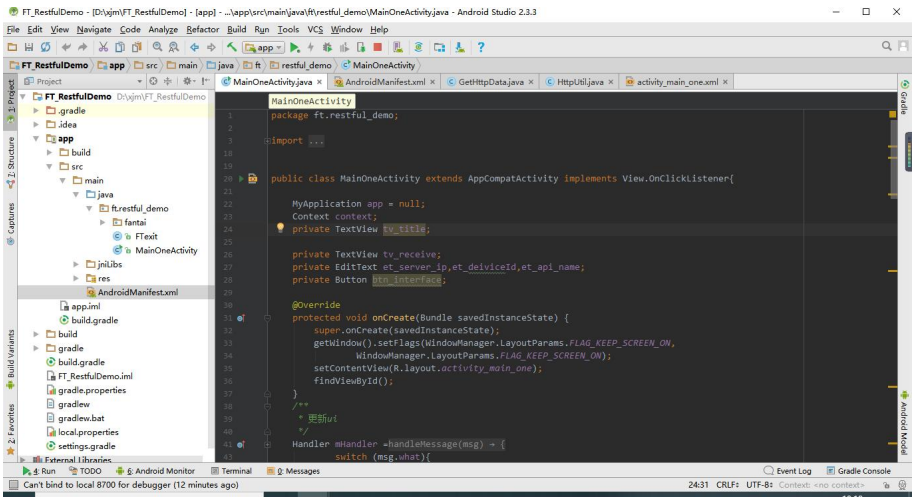
网关通过 WiFi 连接接入路由器下，保证该网关可以连接互联网。如下图所示：

2. 打开项目

使用 Android studio 工具打开光盘资料（路径名）路径下的工程名为 FT_RestfulDemo 如下图所示：

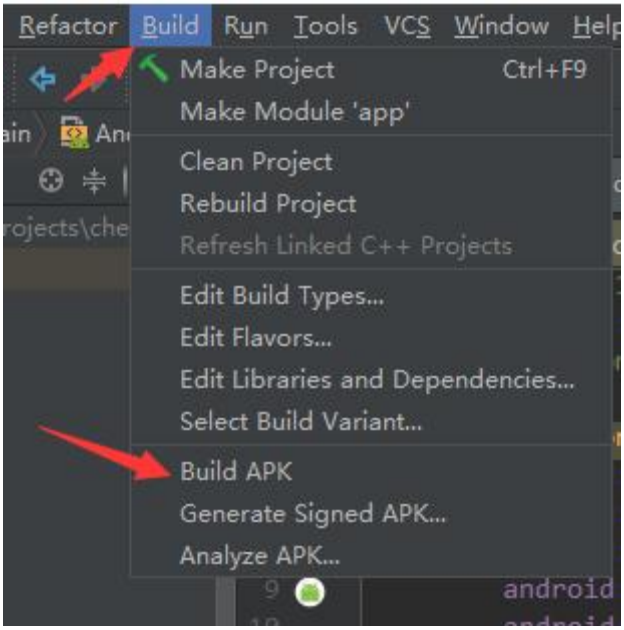


按照上图步骤引入工程项目至 Android studio 开发工具中。如下图所示：

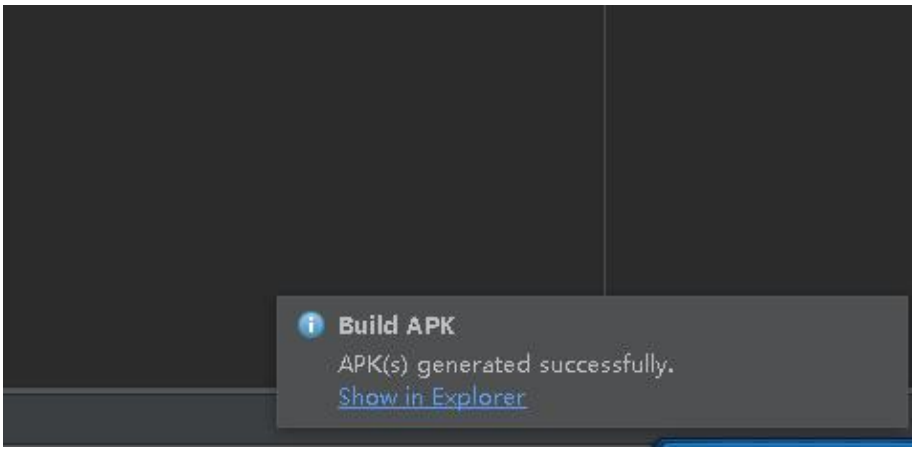


3. 编译生成 APK

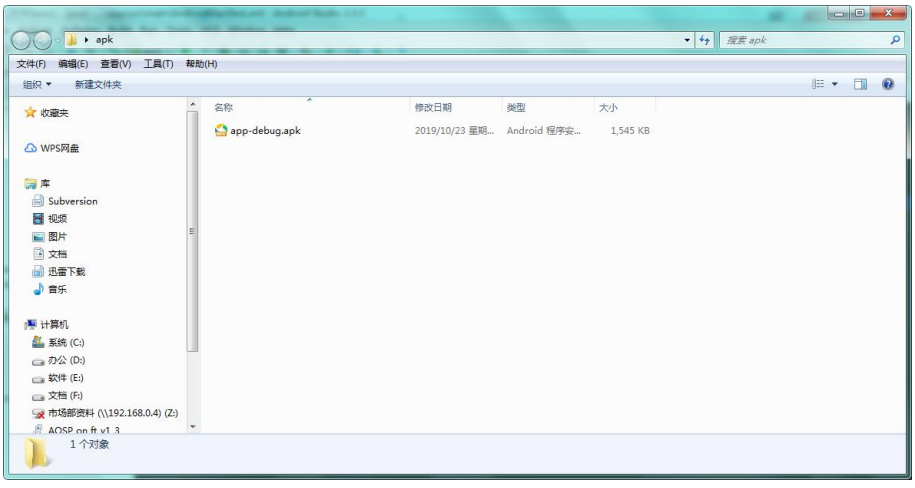
依次点击 Build—》Build APK，如下图所示：



等待编译结束右下角会弹出提示框如下图所示：



点击蓝色字体便会打开生成 apk 的文件夹，如下图所示：

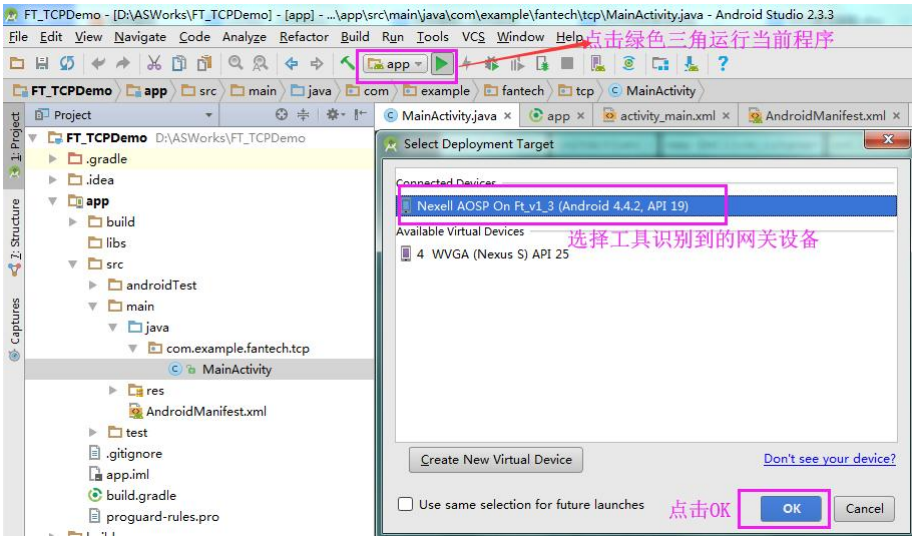


将生成的 APK，拷贝到 TF 内存卡中，将 TF 卡插到 A9 网关上，将 A9 网关

上电，将这个 APK 文件安装到 A9 网关上，点击运行即可。

4. 直接运行程序至网关（另一种生成 apk 的方式）

首先拿一根 Android 手机数据线，一头插入网关的 OTG 接口，一头插入 PC 端的 USB 接口。接着如下图操作：

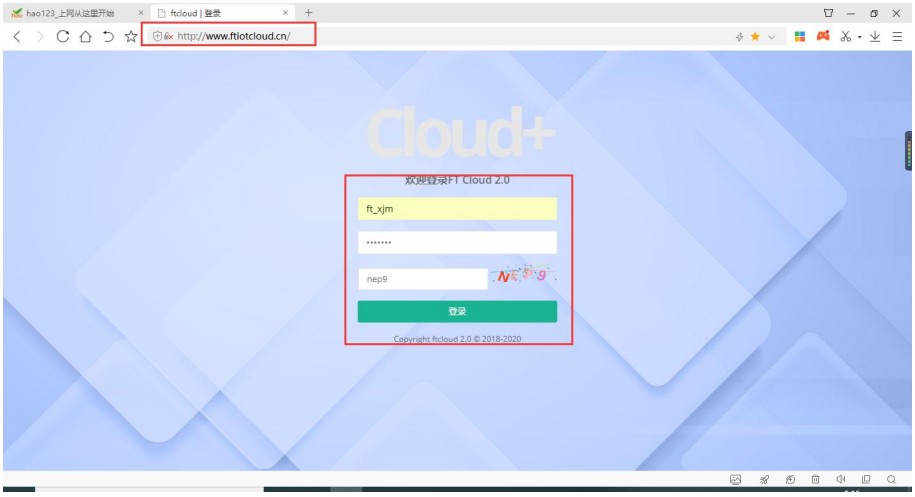


4.6.6. 实验结果

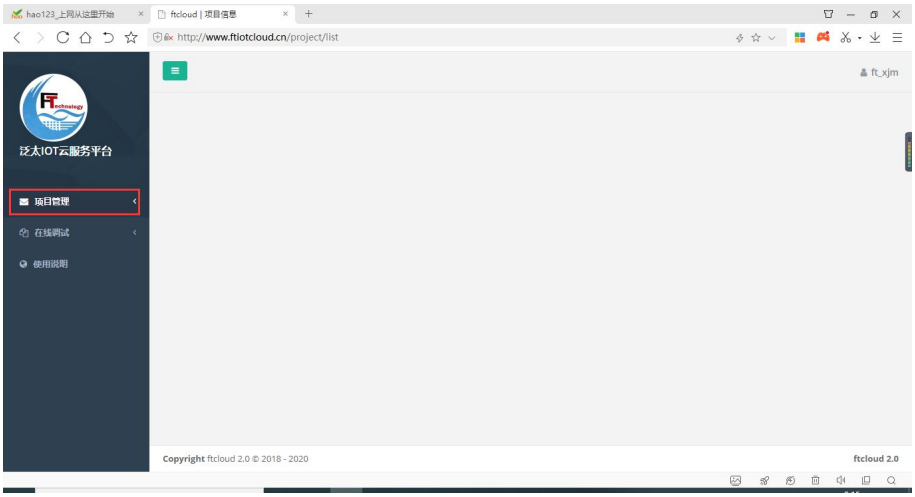
软件图标如下图所示：

点击打开如下图所示：

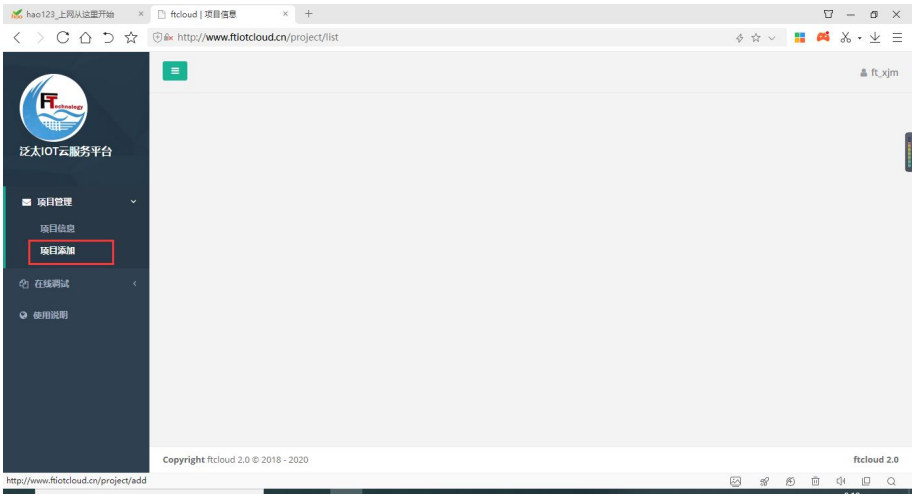
1) 首先登陆云平台，打开浏览器输入云平台网址“http://www.ftiotcloud.cn/”输入项目提供的用户名密码，进行登陆。如下图



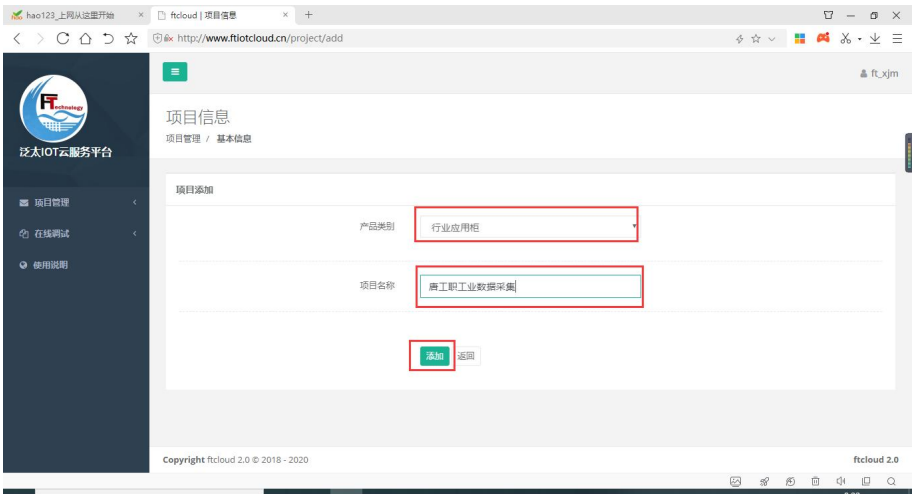
登陆成功后显示如下页面，点击左侧列表中的项目管理选项。



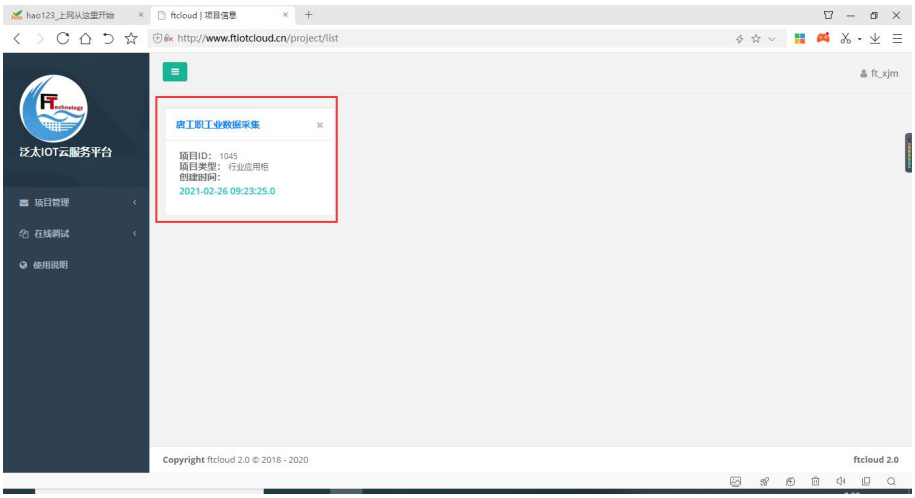
点击项目添加，如下图



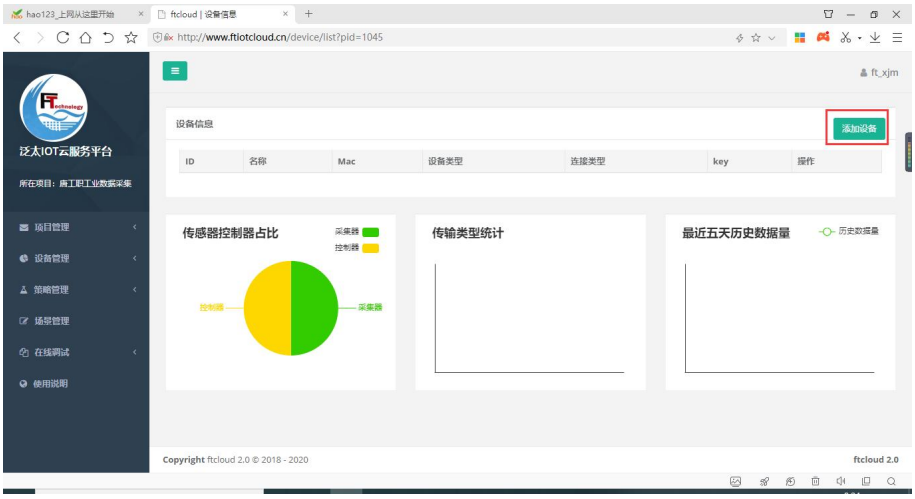
产品类别选择行业应用柜，项目名称：自定义输入，点击添加按钮。如下图



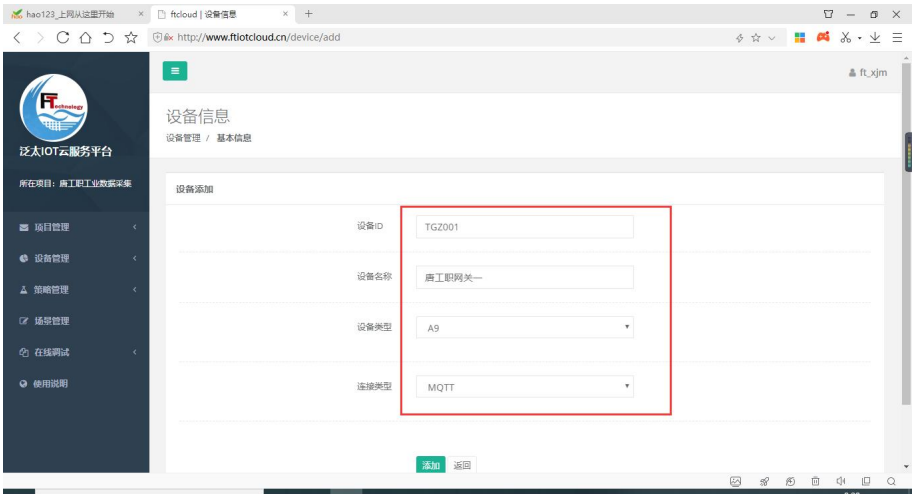
项目添加成功后如下图，点击蓝色项目名称，进入项目。



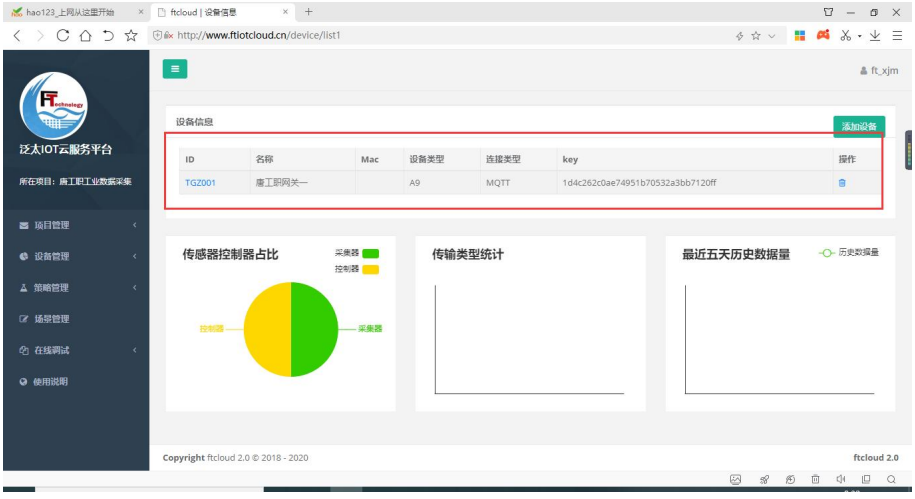
点击右侧的添加设备按钮。



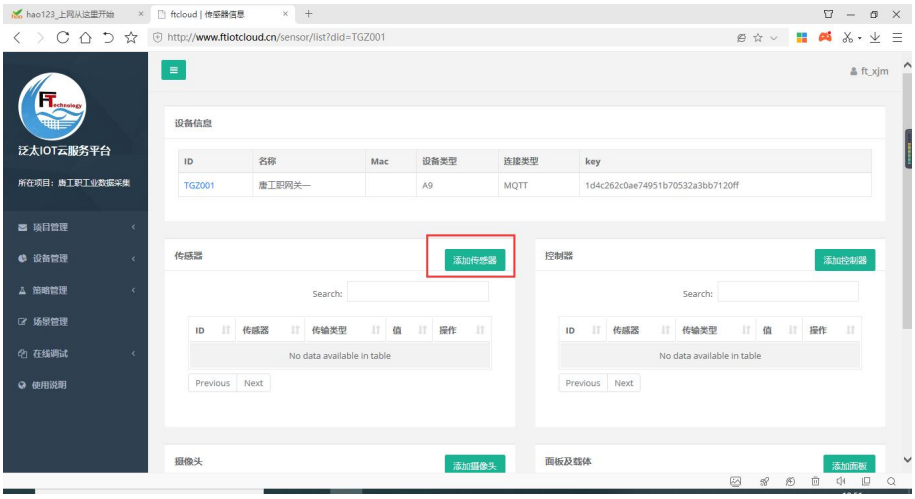
输入设备 ID，设备名称，选择设备类型，连接类型，点击添加按钮。如下图。



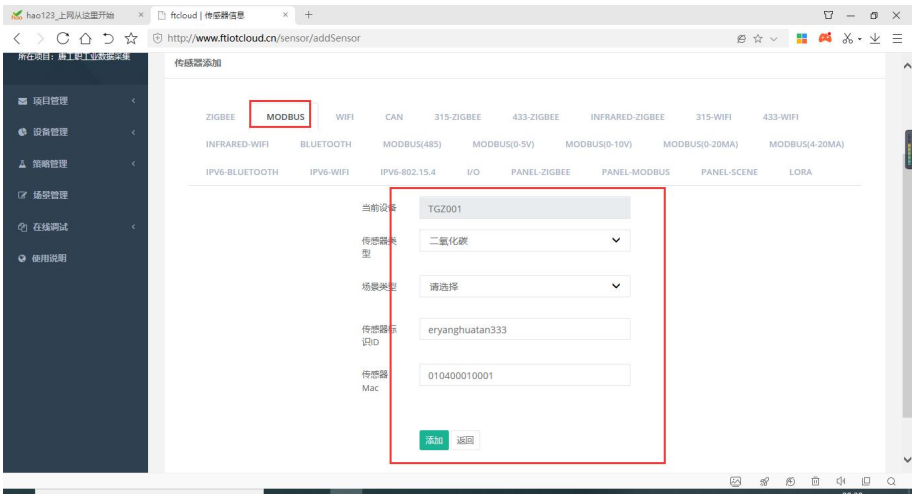
添加成功后如下图。



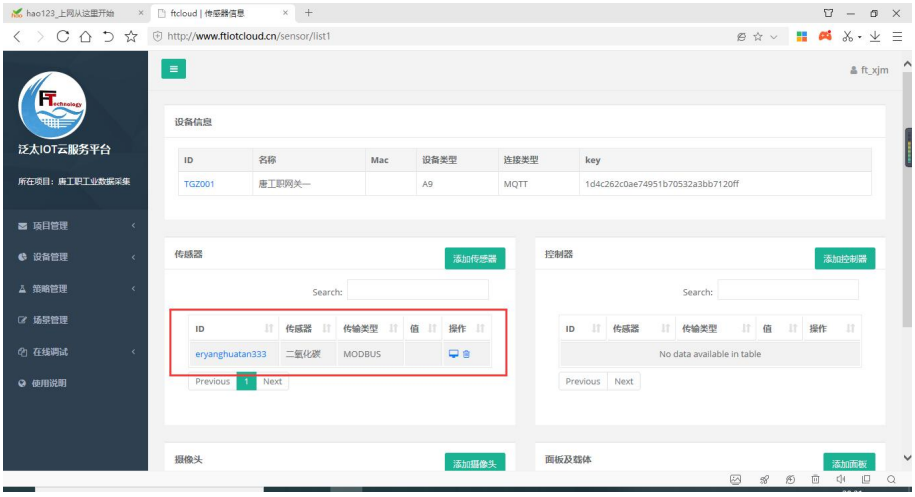
点击蓝色设备 ID，进入设备信息页面，如下图。



点击添加传感器按钮，进入添加传感器页面。如下图



添加成功后如下图



2) 网关客户端要获取云平台服务器地址为 http://www.ftiotcloud.cn:80/RabbitMqServer/。的 Restful 接口数据，设备 ID 是在云平台添加的设备 ID，然后点击调用接口按钮。如下图

调用接口成功后如下图所示，获取到数据会显示在左侧框内，当前数据即之前在云平台添加的传感器。

到此，Restful 接口实验功能完成。

4.7. 远程测控终端 MQTT 协议接入平台实验

4.7.1. 实验目的

1. 了解 MQTT 协议的基础知识；
2. 了解 NB 模块入网的相关 AT 命令；
3. 掌握远程测控终端通过 MQTT 协议接入云平台的方法。

4.7.2. 实验环境

1. 硬件：远程测控终端模块、程序下载调试板、10 针排线、20 针排线、ARM JLINK 仿真器、USB 方口线、DC 12V 电源以及 PC 机。
2. 软件：Windows 7 及以上操作系统，Keil4 for ARM 4.7 开发环境。

图 4-1 远程测控终端模块示意图

4.7.3. 实验原理

1. NB-IoT 简介及原理

NB-IoT 就是窄带物联网（Narrow Band Internet of Things, NB-IoT）， 它已经成为万物互联网络的一个重要分支。NB-IoT 构建于蜂窝网络，只消耗大约 180KHz 的带宽，可直接部署于 GSM 网络、UMTS 网络或 LTE 网络，以降低部署成本、实现平滑升级。NB-IoT 是 IoT 领域一个新兴的技术，支持低功耗设备在广域网的蜂窝数据连接，也被叫作低功耗广域网(LPWAN)。本实验中使用的 NB 模块型号为：BC20。

NB-IOT 的技术特点如下:

(1) 频谱窄：200kHz，终端发射窄带信号，提升了信号的功率谱密度，覆盖增益，频谱利用效率。

(2) 低速率：上行速率峰值 5.6-204.8kbit/s，下行速率峰值 176-234.7kbit/s，适合于小数据量，小速率应用场景。

(3) 低功耗：NB-IoT 设备功耗可以做到非常小，设备续航时间可以从过去

的几个月大幅提升到几年。

(4) 低成本：NB-IoT 不需重新构建网，射频和天线基本上都是复用的。

(5) 广覆盖：NB-IoT 室内覆盖能力强，不仅可以满足农村这样的广覆盖需求，对于厂区、地下车库、井盖这类对深度覆盖有要求的应用同样适用。

(6) 多连接：在同一基站的情况下，NB-IoT 可以比现有无线技术提供 50-100 倍的接入数。一个扇区能够支持 10 万个连接，支持低延时敏感度、超低的设备成本、低设备功耗和优化的网络架构。

2. MQTT 协议

（1）MQTT 是什么？

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 最大的优点在于可以以极少的代码和有限的带宽，为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，MQTT 在物联网、小型设备、移动应用等方面有广泛的应用。

当然，在物联网开发中，MQTT 不是唯一的选择，与 MQTT 互相竞争的协议有 XMPP 和 CoAP 协议等。

（2）MQTT 是哪一层的协议？

众所周知，TCP/IP 参考模型可以分为四层：应用层、传输层、网络层、链路层。TCP 和 UDP 位于传输层，应用层常见的协议有 HTTP、FTP、SSH 等。MQTT 协议运行于 TCP 之上，属于应用层协议，因此只要是支持 TCP/IP 协议栈的地方，都可以使用 MQTT。

（3）MQTT 消息格式

每条 MQTT 命令消息的消息头都包含一个固定的报头，有些消息会携带一个可变报文头和一个负荷。消息格式如下：

```
固定报文头 | 可变报文头 | 负荷
```

固定报文头（Fixed Header）

MQTT 固定报文头最少有两个字节，第一字节包含消息类型（Message Type）和 QoS 级别等标志位。第二字节开始是剩余长度字段，该长度是后面的可变报文头加消息负载的总长度，该字段最多允许四个字节。

剩余长度字段单个字节最大值为二进制 0b0111 1111，16 进制 0x7F。也就是说，单个字节可以描述的最大长度是 127 字节。为什么不是 256 字节呢？因为 MQTT 协议规定，单个字节第八位（最高位）若为 1，则表示后续还有字节存在，第八位起“延续位”的作用。

例如，数字 64，编码为一个字节，十进制表示为 64，十六进制表示为 0×40。数字 321（65+2*128）编码为两个字节，重要性最低的放在前面，第一个字节为 65+128=193（0xC1），第二个字节是 2（0x02），表示 2×128。

由于 MQTT 协议最多只允许使用四个字节表示剩余长度，如下表所示，并且最后一字节最大值只能是 0x7F 不能是 0xFF，所以能发送的最大消息长度是 256MB，而不是 512MB。

表 4-1

Digits	From	To
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

可变报文头（Variable Header）

可变报文头主要包含协议名、协议版本、连接标志（Connect Flags）、心跳间隔时间（Keep Alive timer）、连接返回码（Connect Return Code）、主题名（Topic Name）等，后面会针对主要部分进行讲解。

有效负荷（Payload）

Payload 直译为负荷，可能让人摸不着头脑，实际上可以理解为消息主体。

当 MQTT 发送的消息类型是 CONNECT（连接）、PUBLISH（发布）、SUBSCRIBE（订阅）、SUBACK（订阅确认）、UNSUBSCRIBE（取消订阅）时，则会带有负荷。

3. AT 指令：

BC20 NB 模块采用 AT 指令集进行操作。

前缀 AT 或 at 必须加在每个命令行的开头。输入 <CR> 将终止命令行。通常，命令后面跟随形式为<CR><LF><response><CR><LF> 的响应。

<CR> 回车符。

<LF> 换行符。

<…> 参数名称。 实际命令行中不包含尖括号。

[…] 可选参数或 TA 信息响应的可选部分。 实际命令行中不包含方括号。若无特别说明，配置命令中的可选参数被省略时，将使用其保存至 NVRAM 的值或其默认值。

下划线 参数的默认设置。

AT 指令集主要分为：基础类 AT 指令及扩展类 AT 指令。

基础类 AT 命令的格式为 AT<x><n> 或 AT&<x><n>，其中 <x> 是命令，<n> 是该命令的参数。以 ATE<value> 为例，DCE 会根据< value> 的取值确定是否将接收到的字符回显给 DTE。 若 <n> 为可选参数， 则其被省略时将使用其默认值或已保存至 NVRAM 的设置值。

扩展类 AT 命令可以在多种模式下运行，如下表所示：

测试命令	AT+<cmd>=?	返回相应设置命令或内部程序可支持的参数取值列表或范围。
查询命令	AT+<cmd>?	返回相应设置命令的当前参数设置值。
设置命令	AT+<cmd>= <p1>[,<p2>[,<p3>[...]]]	设置用户可自定义的参数值。
执行命令	AT+<cmd>	主动执行内部程序实现的功能集。

注意：

每次仅支持执行一个 AT 命令。仅当上一个命令执行完成后，方可执行下一个命令。

AT 指令均以回车、换行字符（“\r\n”）结尾。

此处仅示例列出几条 AT 命令。

通讯测试命令

发送：AT

返回：OK

模块复位（重启）命令

发送：AT+QRST=<mode>

返回：(立即自动重启)

查询制造商版本号命令（默认为固件版本号）

发送：AT+CGMR

返回：Revision: <revision>

OK

详细的 AT 命令请参照 BC20 的 AT 命令说明文档。

4. 电路原理

远程测控终端原理图如图 4-、4-3 所示，STM32 的引脚 PE15 控制模块电源，当 PE15 为高电平时，NB-IoT 模块通电；STM32 通过引脚 PD8 和 PD9（串口 3）与 NB-IoT 模块通信。STM32 通过串口 3 发送 AT 指令对 NB-IoT 模块进行配置以及进行数据的读取和发送。

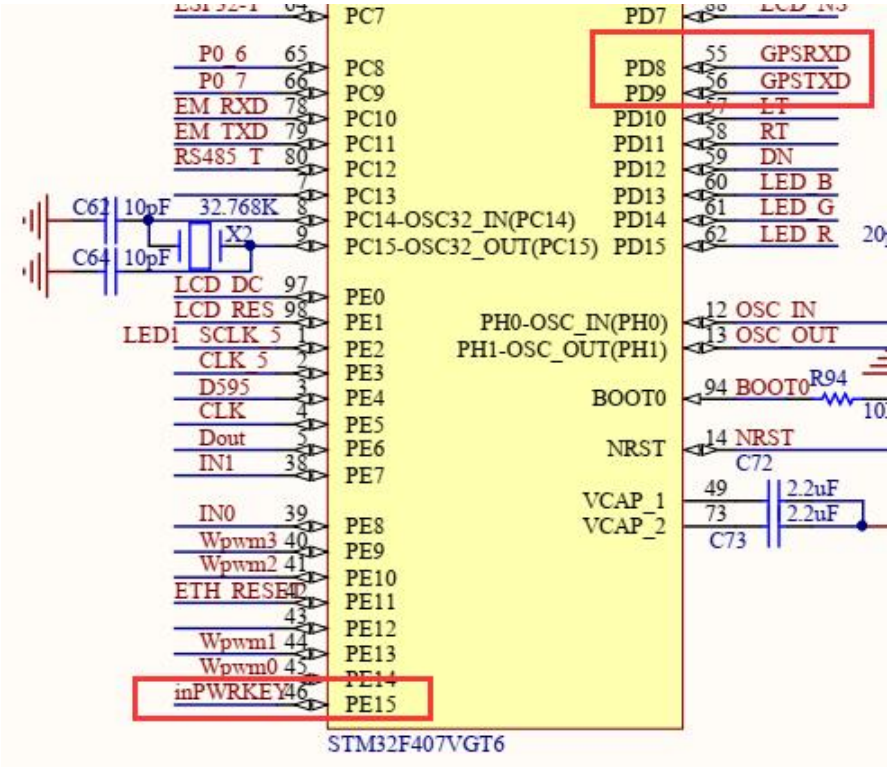


图 4-2 STM32 和 NB 模块相关的 IO 口

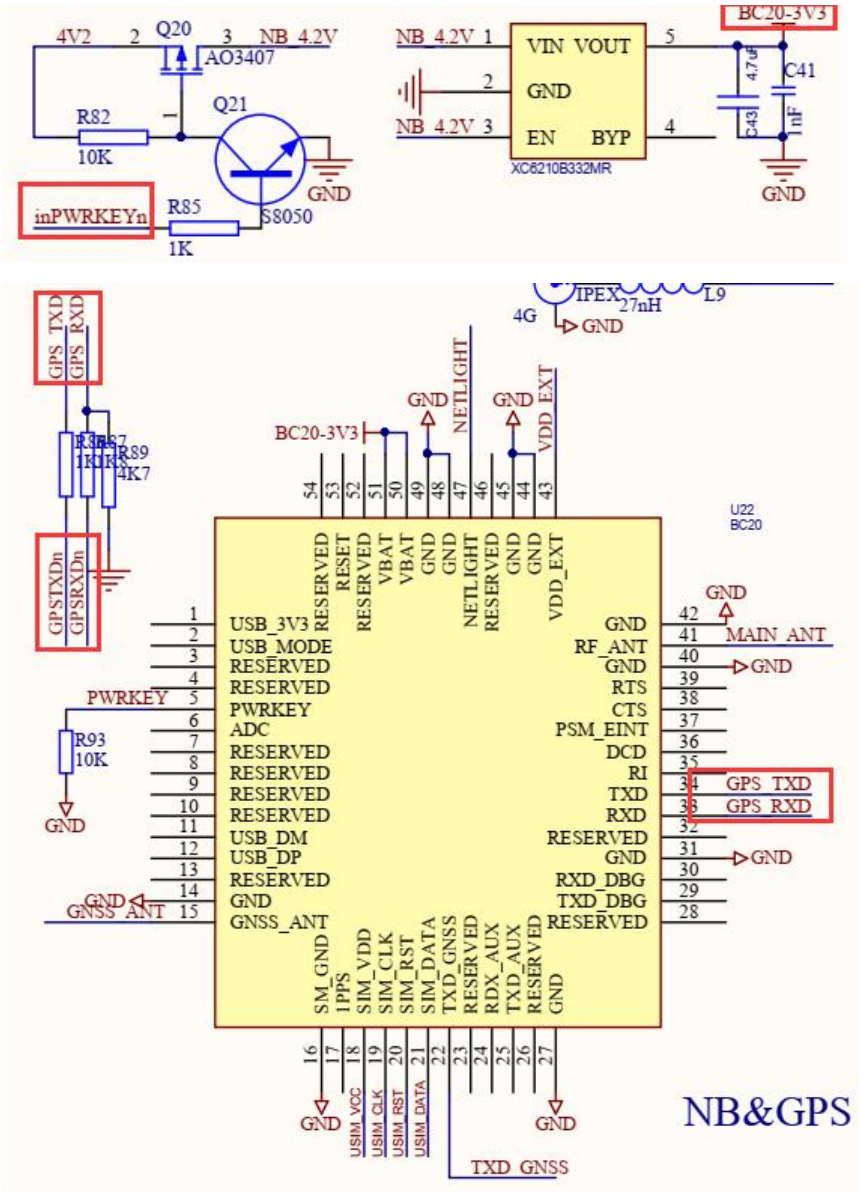


图 4-3 NB-IoT 模块硬件原理图

4.7.4. 实验内容

本实验主要是通过 STM32 控制 NB-IoT 模块基于 MQTT 协议接入平台，然后通过 NB 向平台发送数据，并且接收和解析平台下发的数据，执行相应的操作，具体代码如下。

主函数的内容如下所示：

```
int main(void)
{
    SysMainInit();     //系统初始化，包括看门狗，系统延时，LED，按键，定时器，LCD等初始化
```

```
        while(1)
        {
            Iwdg_Feed();    //喂狗

            KeyDetection_Process();  //按键检测并执行相应操作

            if(0 == viewModeFlg)       //正常模式，非查看模式
            {
                Periodic_Process();     //周期性任务

                if(
    #ifdef MODBUS_RTU_RS485
                    MODBUS_RTU_RS485 ||
    #endif
                    (WirelessType == LORA && GET_LORA_INIT_FINISH_FLAG() !=
_FALSE)
                    || (WirelessType == ZIGBEE &&
GET_ZIGBEE_INIT_FINISH_FLAG() != _FALSE)
                    || (WirelessType == WIFI && GET_WIFI_INIT_FINISH_FLAG() !=
_FALSE)
                    || (WirelessType == NB_IOT &&
GET_NBIOT_INIT_FINISH_FLAG() != _FALSE)
                    || (WirelessType == ZIGBEE_LORA &&
GET_ZIGBEE_LORA_INIT_FINISH_FLAG() != _FALSE)
                    && (_TRUE == _SUCCESS))
                {
                    if(ModbusRecv_Process_EN)       //允许处理 Modbus 格式命令
                    {
                        ModbusRecv_Process_EN = 0;  //禁止处理 Modbus 格式命令
                        Modbus_Cmd_Process();          //接收到 Modbus 格式的命令后，
执行相应的采集或者控制操作
                    }
                }
            }
        }
    }
```

主函数中首先进行了系统初始化，系统初始化中包含了串口 3 的初始化、NB 的初始化、入网和 MQTT 订阅等等，然后进入主循环。主循环中喂狗，进行按键检测并执行相应操作，如果是正常模式下，执行周期性任务，其中就包含了定时上报 MQTT 消息以及接收到平台下发消息的解析和处理。

NB 初始化、入网和 MQTT 订阅的函数如下所示：

```
    void NBIOT_Init(void)
    {
        CLR_NBIOT_CONNECT_NET_FINISH_FLAG();
        CLR_NBIOT_CREATE_SOCKET_FINISH_FLAG();
        CLR_NBIOT_INIT_FINISH_FLAG();

        delay_ms(3000);

        while(!GET_NBIOT_INIT_FINISH_FLAG())      //如果 NB-IOT 初始化没有完成
        {
            NBIoTCmdId.ucCurrNbIoTCmdId = eNBIOT_AT_AT_CMD;
```

```
            NBIoTCmdId.ucInfoNbIoTCmdId = eNBIOT_AT_EMPTY;
            while(!GET_NBIOT_CONNECT_NET_FINISH_FLAG())      //如果 NB-IOT 入网没有完成
            {
                NbIoTConnectNetwork();          //NB-IOT 入网过程
            }

            if(CommProtocol == PROTOCOL_TCP || CommProtocol == PROTOCOL_UDP)
            {
                NBIoTCmdId.ucCurrNbIoTCmdId = eNBIOT_AT_QICFG_DATAFORMAT_CMD;
                NBIoTCmdId.ucInfoNbIoTCmdId = eNBIOT_AT_EMPTY;
                while(!GET_NBIOT_CREATE_SOCKET_FINISH_FLAG())   //如果 NB-IOT 创建 socket 通道没有完成
                {
                    if( CommProtocol == PROTOCOL_TCP)
                    {
                        NbIoTCreatingTCPSocket();                   //建立 TCP socket 通道
                    }
                    else        //UDP
                    {
                        NbIoTCreatingUDPSocket();                   //建立 UDP socket 通道
                    }
                }
            }
            else        //MQTT
            {
                NBIoTCmdId.ucCurrNbIoTCmdId = eNBIOT_AT_QMTCFG_DATAFORMAT_CMD;
                NBIoTCmdId.ucInfoNbIoTCmdId = eNBIOT_AT_EMPTY;
                while(!GET_NBIOT_MQTT_SUBSCRIBE_FINISH_FLAG())      //如果 MQTT 订阅没有完成
                {
                    NbIoTMQTTConnectServer();               //MQTT 订阅
                }
            }

            Iwdg_Feed();        //喂狗
        }

        USART1_PrintString("\r\nNB-IoT initialized successfully!\r\n");     //串口打印 NB-IOT 初始化完成
    }
```

入网及 MQTT 订阅的函数内容可以在实验代码中进一步追进去查看，了解涉及到的 AT 命令，这里就不多描述了。

MQTT 上报消息的代码如下：

```
                if(GET_NBIOT_SEND_DATA_MQTT_FLAG() != _FALSE && GET_NBIOT_DATA_SEND_MQTT_ENABLE() != _FALSE)        //发送数据定时时间到
                {
                    uint8_t dataStrBuf[255] = {'\0'};

                    //定时上传的数据
                    snprintf((char *)dataStrBuf, sizeof(dataStrBuf),
```

```
    "{\"deviceId\":\"%s\",\"sensorId\":\"0000\",\"key\":\"%s\",\"data\":{\"value\":\"NB_IoT
MQTT access platform test\"},\"code\":\"%s\",\"connecttype\":\"MQTT\",\"trantype\":\"I/O\"}",

                                    NBIOT_MQTT_CLIENTID,FT_KEY,"11111111");

                        NBIoTUploadByMQTT((uint8_t *)dataStrBuf);   //向云平台发送数据
                        delay_ms(20);
                        CLR_NBIOT_SEND_DATA_MQTT_FLAG();
                    }
```

这是一个周期性任务，通过定时器设置 4 秒向云平台上报一次消息，消息上报函数 NBIoTUploadByMQTT()的内容如下：

```
    void NBIoTUploadByMQTT(uint8_t * stUploadData)
    {
        uint8_t tempbuf[512] = {'\0'};

        snprintf((char *)tempbuf, sizeof(tempbuf),
"AT+QMTPUB=0,0,0,0,\"%s%s\",\"%s\"\r\n",

                        NBIOT_MQTT_TOPIC_PUBLISH,

                        NBIOT_MQTT_CLIENTID,

                        stUploadData);

        SET_NBIOT_RETURN_MSG_MQTT_FLAG();

        NbIoTSendCmd(tempbuf);          //通过 NB-IOT 向云平台发送数据
    }
```

其实最终是通过串口 3 发送发布的 AT 命令来进行消息上报的。

接收到云平台下发消息的处理如下：

```
                        if(strstr((char *)&NBIOT_USART_INFO[0], "+QMTRECV:") != NULL)
//接收到订阅的消息
                        {
                            MQTTServerCtrlSensor(&NBIOT_USART_INFO[0]);//MQTT 消
息处理
                        }
```

这个函数中包含的 MQTTServerCtrlSensor()函数执行具体的处理，它如下所示：

```
    void MQTTServerCtrlSensor(uint8_t *data)
    {
        char *dataBuff = NULL;
        uint8_t *pCtrCmdParams = NULL;
        char    TOPIC_PUB[50] = {'\0'};

        snprintf((char *)TOPIC_PUB, sizeof(TOPIC_PUB), "%s%s",
NBIOT_MQTT_TOPIC_SUBSCRIBE,NBIOT_MQTT_CLIENTID);

        if(strstr((char *)data, TOPIC_PUB) != NULL)
        {
```

```
            pCtrCmdParams = (uint8_t *)strstr(strstr((char *)data, "data"), "\"value\"");       //
将指针定位到“data”里的“value”处

            if(pCtrCmdParams != NULL)
            {
                //分割字符串，取出 value 的内容
                strtok((char *)pCtrCmdParams, ":");
                dataBuff = strtok(NULL, "\"");


                if(strncmp(dataBuff, "LED_ON", 6) == 0)       //如果 value 的内容是控制命
令“开灯”
                {
                    LED_R_ON;
                }

                if(strncmp(dataBuff, "LED_OFF", 7) == 0)     //如果 value 的内容是控制命
令“关灯”
                {
                    LED_R_OFF;
                }
            }
        }
```

这个函数解析平台下发的消息，找到“value”中的内容，判断是什么消息，如果是开灯命令，就点亮 R 灯，如果是关灯命令，就熄灭 R 灯。

4.7.5. 实验步骤

1. 首先进行硬件连接，使用 USB 方口线连接 ARM JLINK 仿真器和 PC，使用 20 针排线连接仿真器和下载调试板，使用 10 针排线连接下载调试板和远程测控终端，最后 使用 DC 12V 电源给远程测控终端供电。硬件连接如图 2- 所示。

图 4-4 硬件连接图

2. 打开本实验的代码，路径：04-云平台接入测试\04-远程测控终端 MQTT 协议接入平台\Project\RTU.uvproj，重新编译代码并将代码下载到远程测控终端模块中，编译和下载按钮如图 4-5 所示。

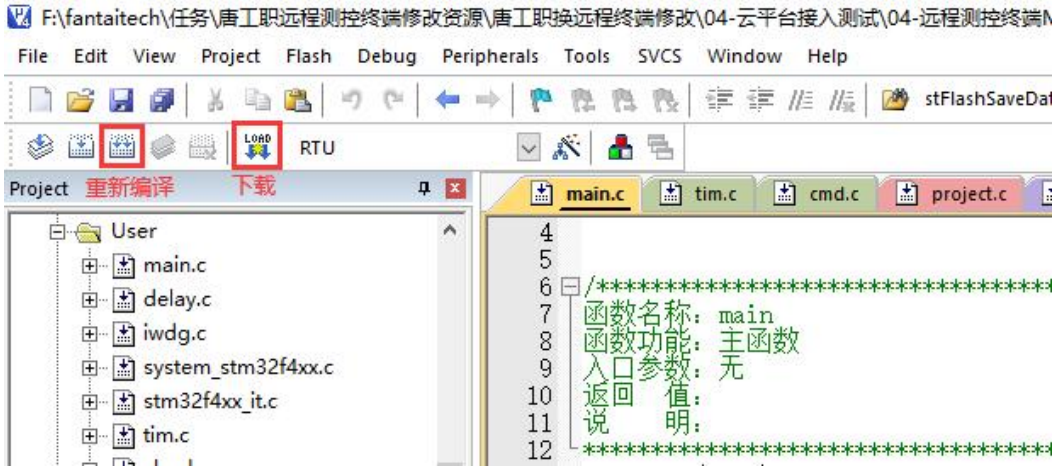


图 4-5 编译下载程序

3. 从远程测控终端上拔下仿真器，然后给远程测控终端重新上电，让程序重新运行。

4. 打开浏览器，进入网址为 www.ftiotcloud.cn:3000 的网页，如图 4-所示。

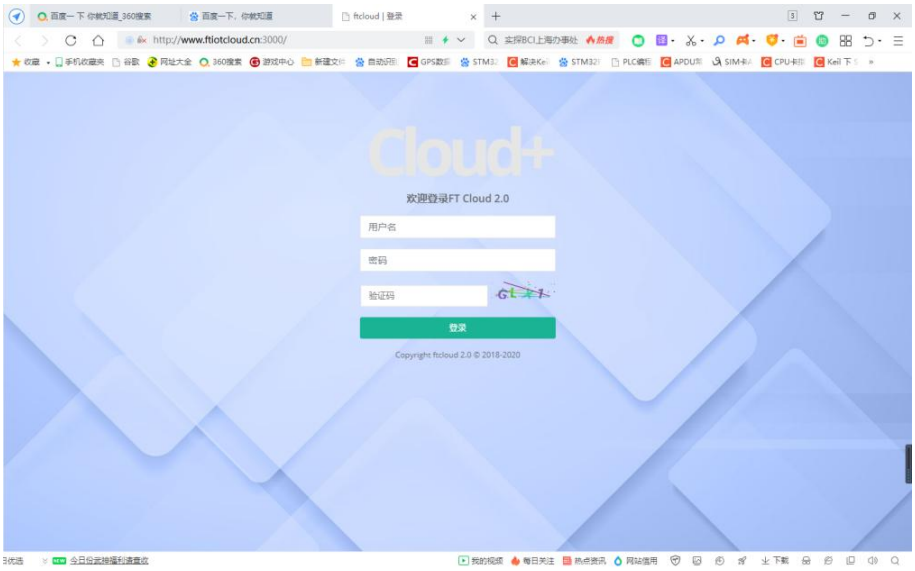


图 4-6 登录平台

5. 输入给定的用户名和密码，然后点击登录，可以进入如图 4-所示页面。

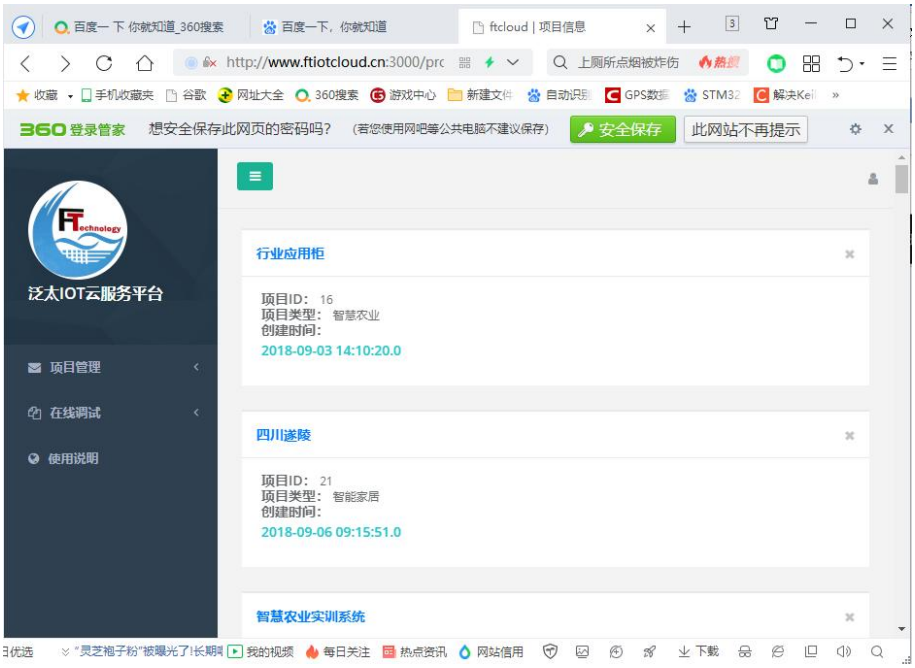


图 4-7 登录成功页面

6. 点击左侧的“项目管理”—>”项目添加”，进入如图 4-所示页面。

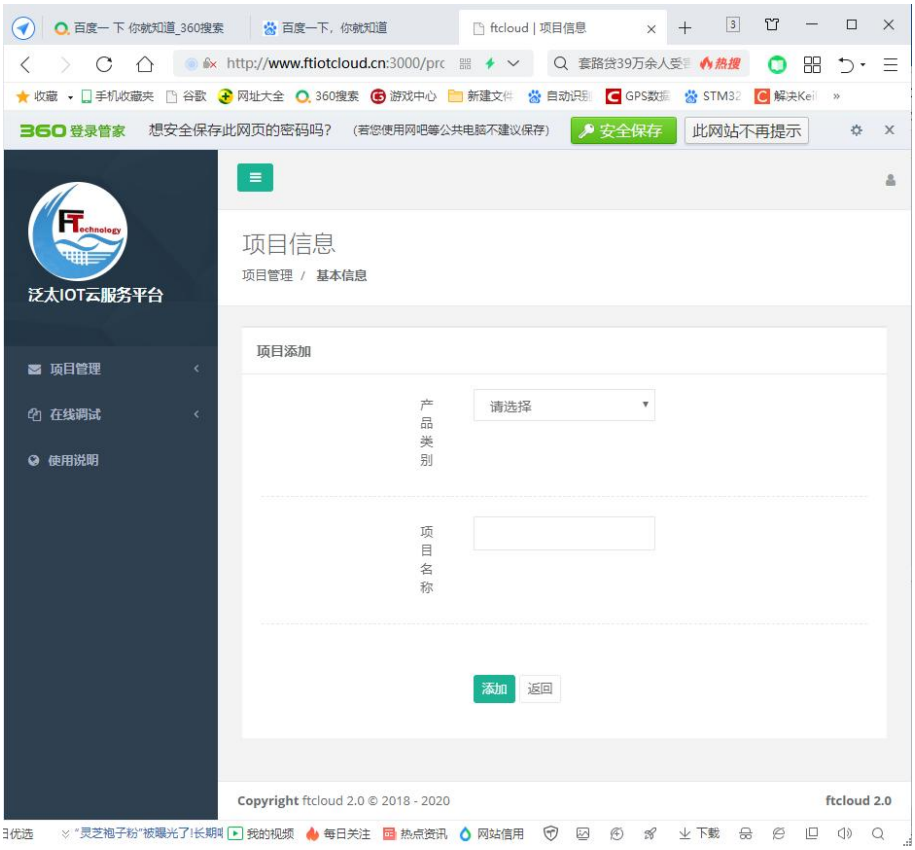


图 4-8 项目添加页面

7. 项目类型按照需要选择，比如“实验箱”、“工业物联网”等等，项目名称可以自定义。如图 4-所示。最后点击“添加”。

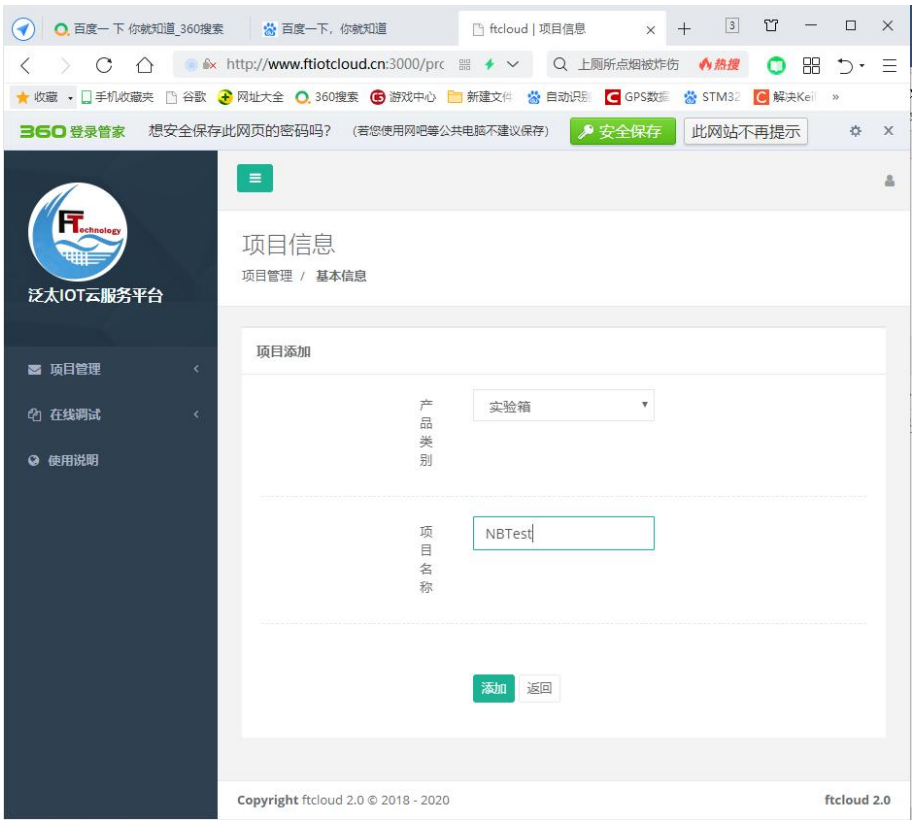


图 4-9 项目添加

8. 然后可以在项目信息中找到刚刚添加的项目“NBTest”，如图 4-所示。

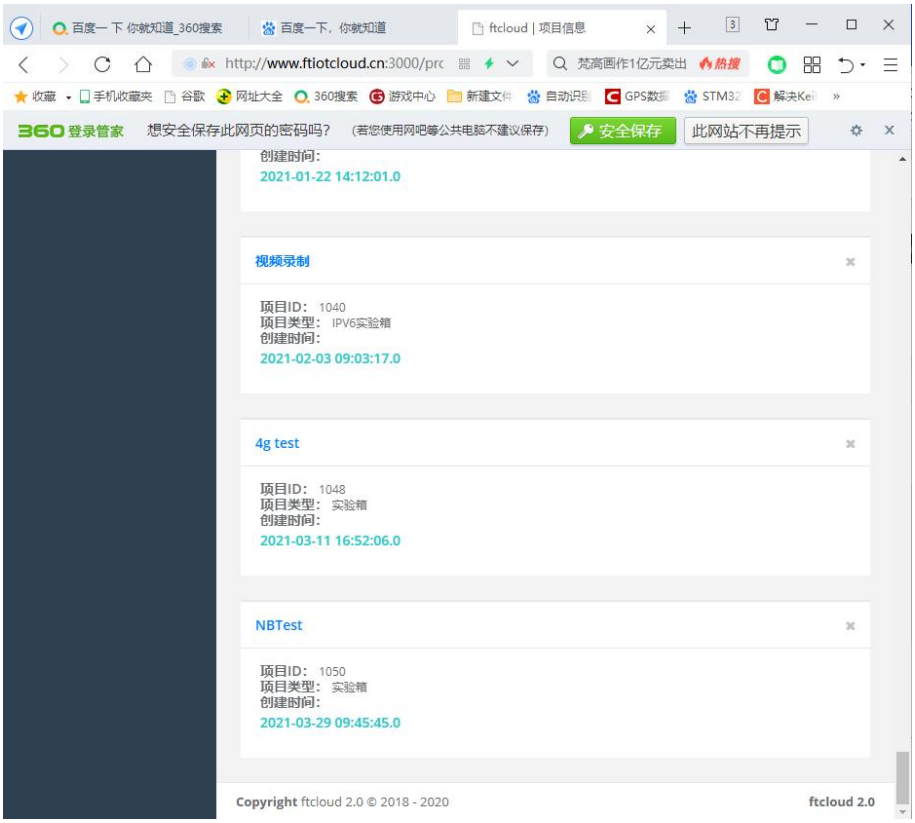


图 4-10 项目信息

9. 点击项目名称进到如图 4-所示页面，然后再点击“添加设备”，进入到如图 4-3 所示页面，信息填写如图 4-4 所示，这里的设备 ID 是远程测控终端的 IMEI 号（远程测控终端的 IMIE 号显示在 LCD 屏上）；设备名称是自定义的，如果设备 ID 号重复了平台将无法添加，设备类型选择 STM32，连接类型是 MQTT，最后点击添加。添加成功后就可以在设备信息栏看到刚刚添加的设备。如图 4-5 所示。

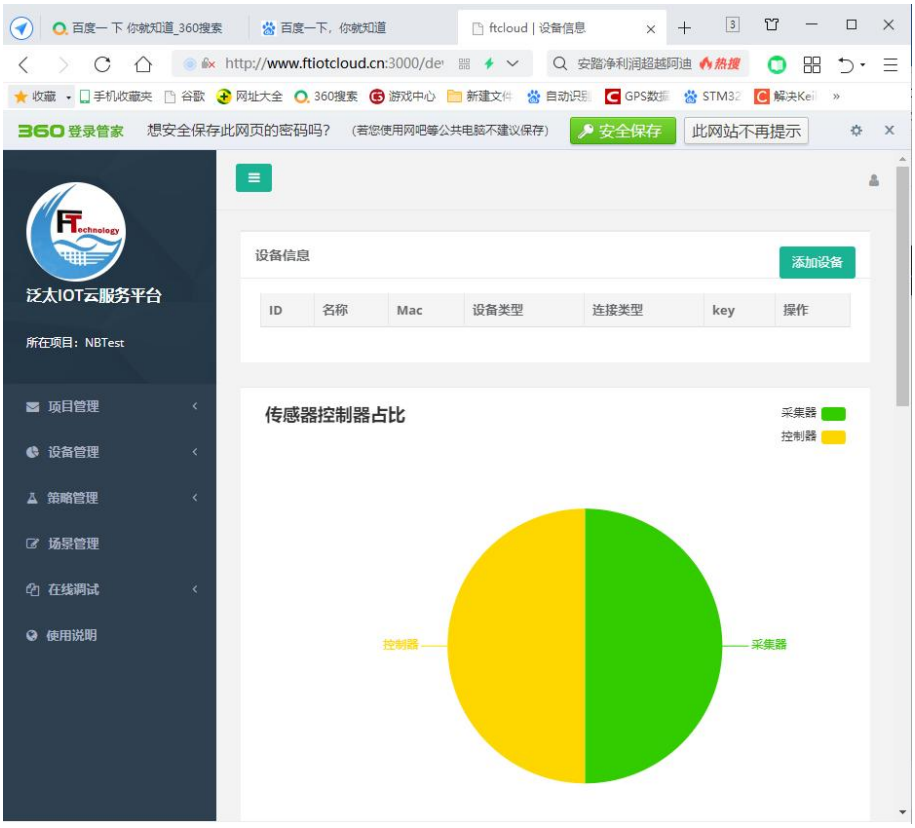


图 4-11 设备信息

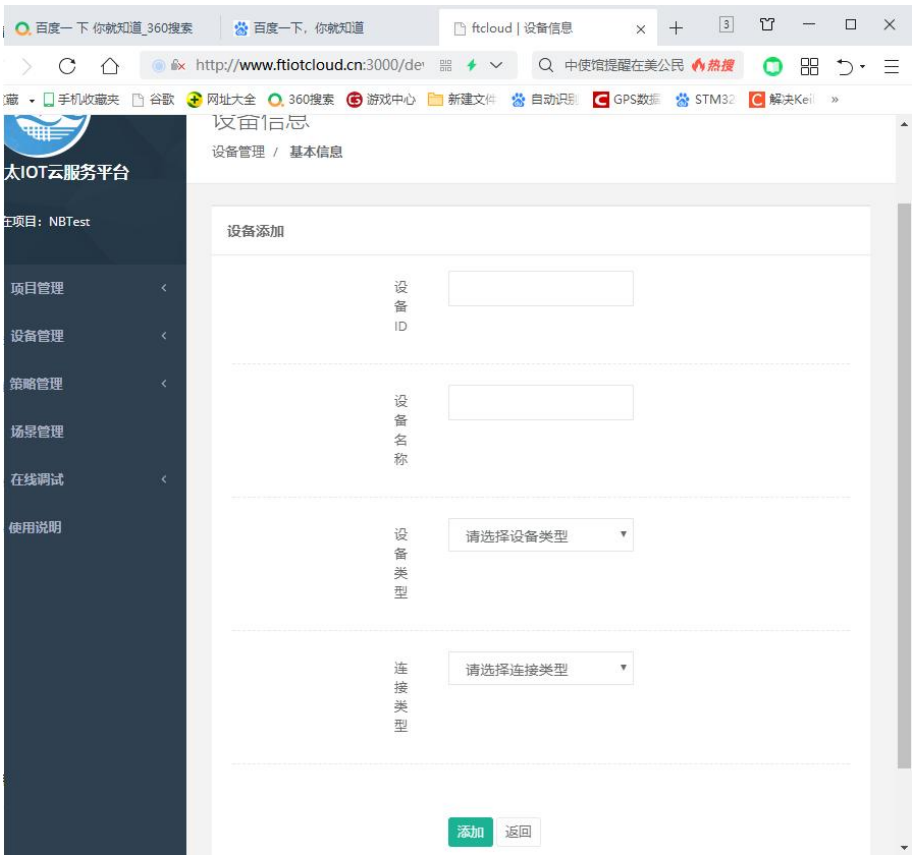


图 4-3 设备信息添加

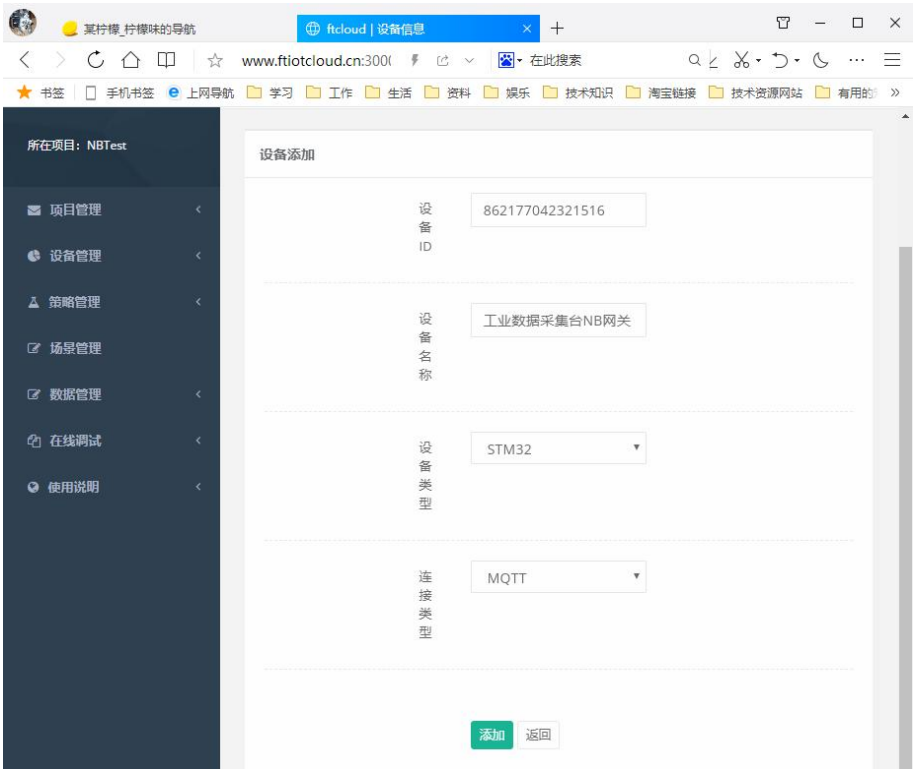


图 4-4 设备信息填写

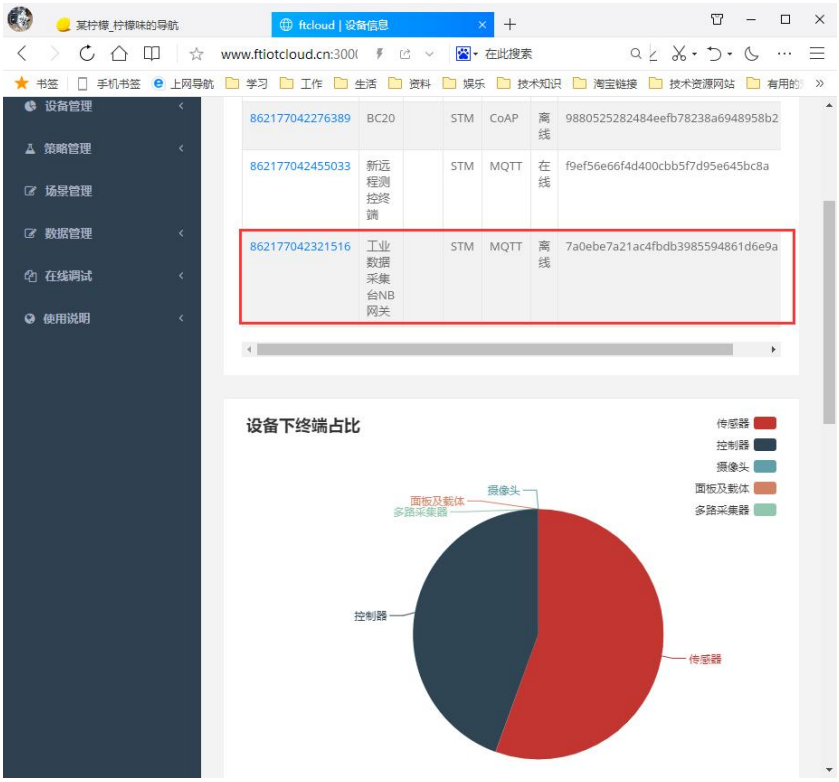


图 4-5 设备添加完成

10. 接下来点开左侧的“在线调试”，再打开里面的“MQTT 调试”，如图 4-6 所示，就可以看到如图 4-7 所示页面。

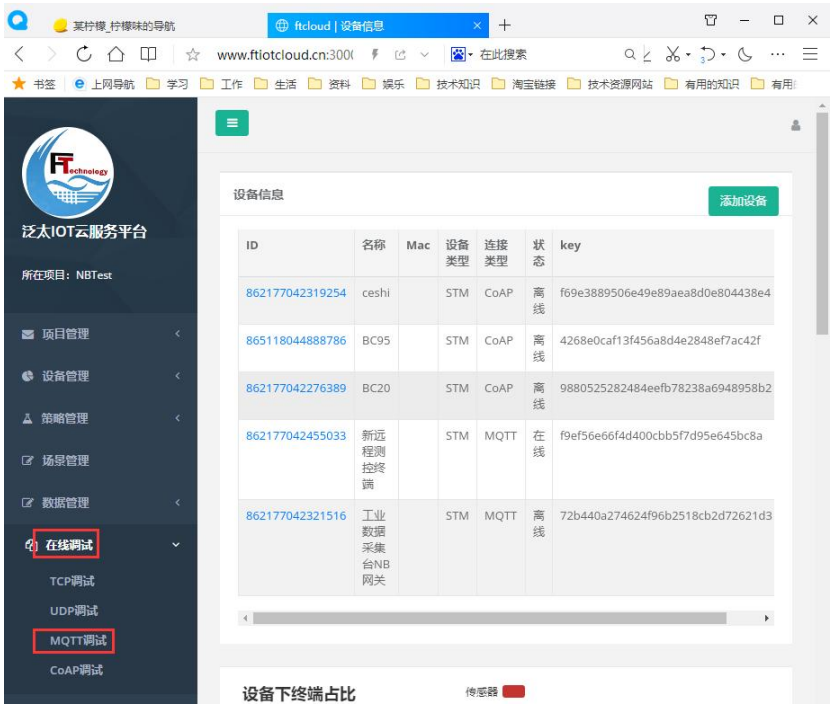


图 4-6 打开在线调试页面

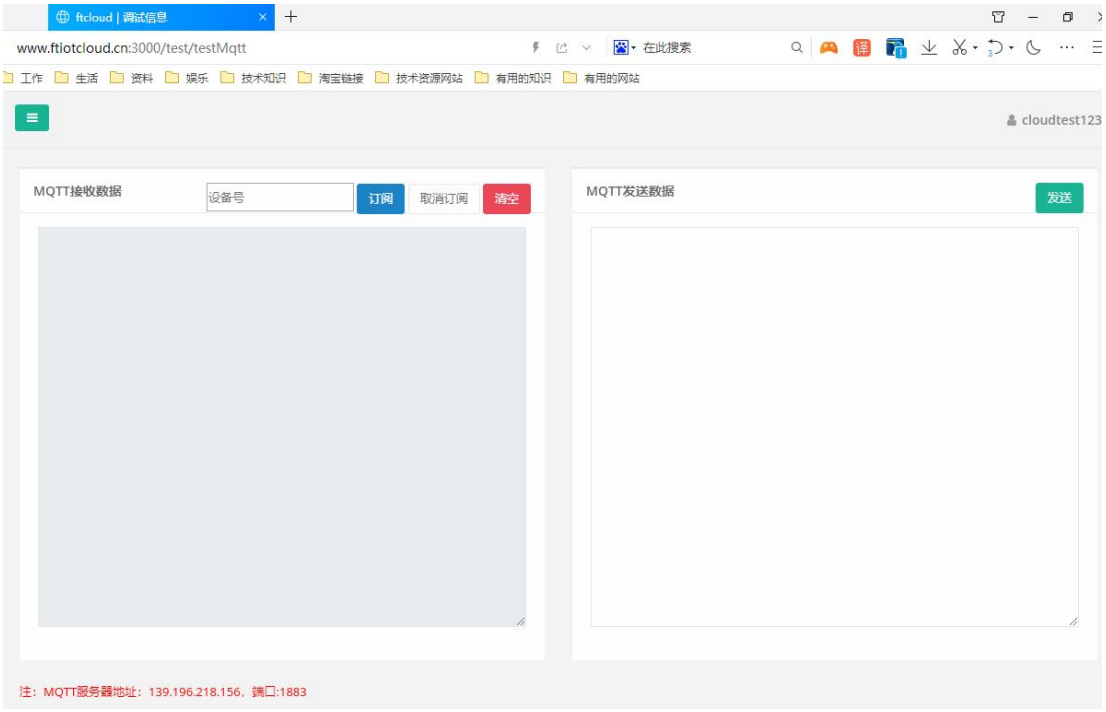


图 4-7 MQTT 调试页面

11. 因为多组人员同时实验时平台接收的数据都会显示出来，显然不利于实验，所以需要在平台上订阅一下设备号，也就是远程测控终端 NB 模块的 IMEI 号，这样只会显示该设备号对应的数据，如下所示：

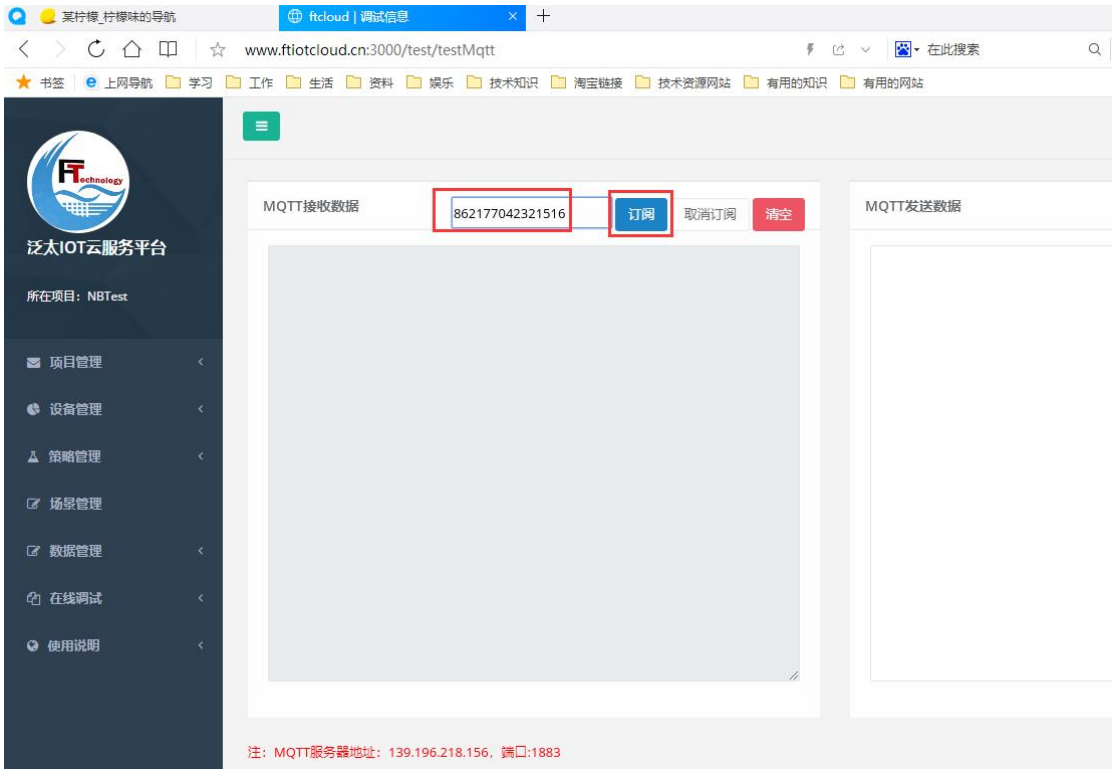


图 4-8 订阅设备号

4.7.6. 实验结果

在 MQTT 接收区可以看到模块向平台发送的数据，如图 4-9 所示。

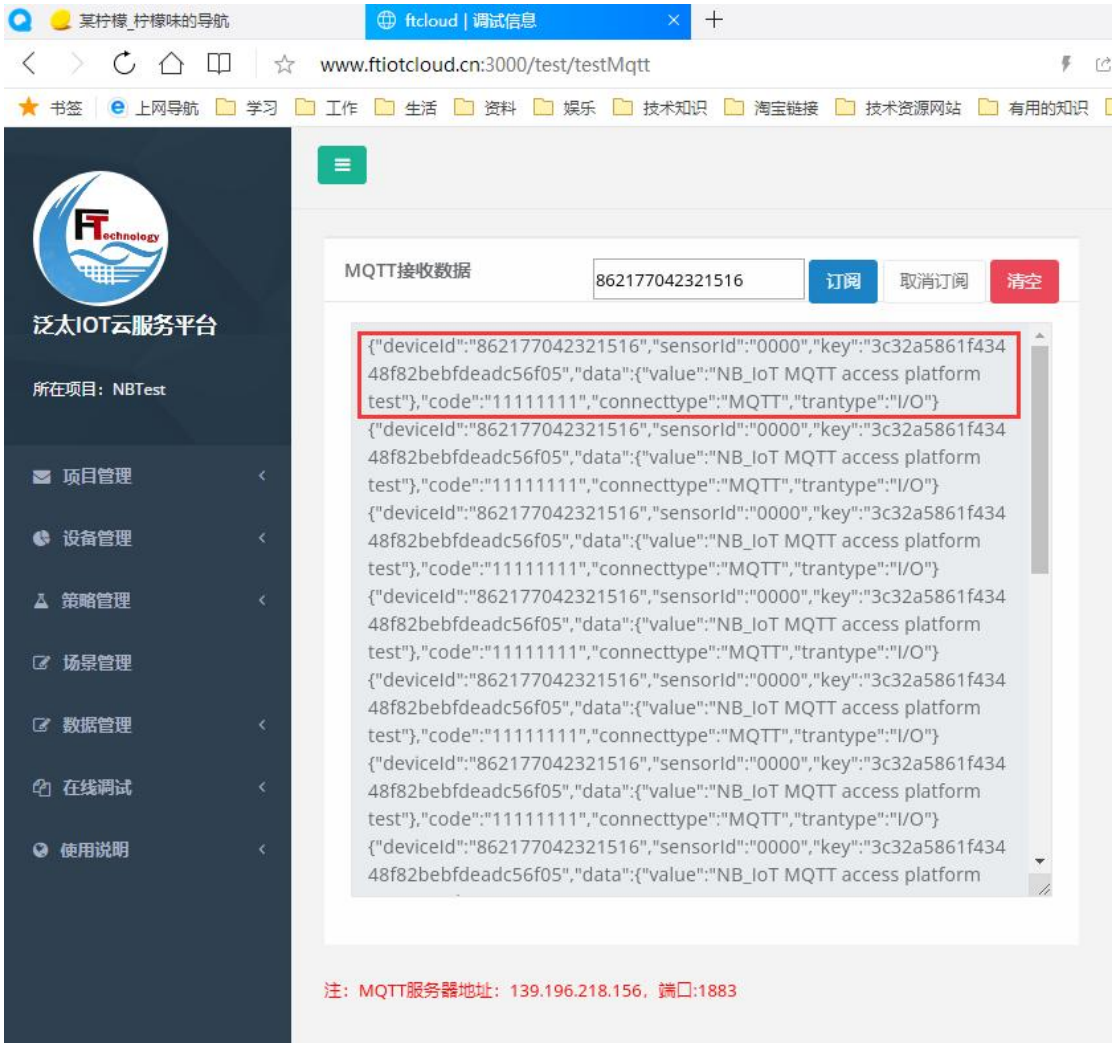


图 4-9 平台接收数据

平台接收到的数据内容如下：

{"deviceId":"862177042321516","sensorId":"0000","key":"3c32a5861f43448f82bebfdeadc56f05","data":{"value":"NB_IoT MQTT access platform test"},"code":"11111111","connecttype":"MQTT","trantype":"I/O"}

复制此数据内容，粘贴到发送区，修改“value”后面的数据，发送的内容如下：

{"deviceId":"862177042321516","sensorId":"0000","key":"3c32a5861f43448f82bebfdeadc56f05","data":{"value":"LED_OFF"},"code":"11111111","connecttype":"MQTT","trantype":"I/O"}

点击“发送”，发送成功的话，本来初始化完成时点亮的 R 灯会熄灭。

图 4-10 平台发送数据熄灭 R 灯

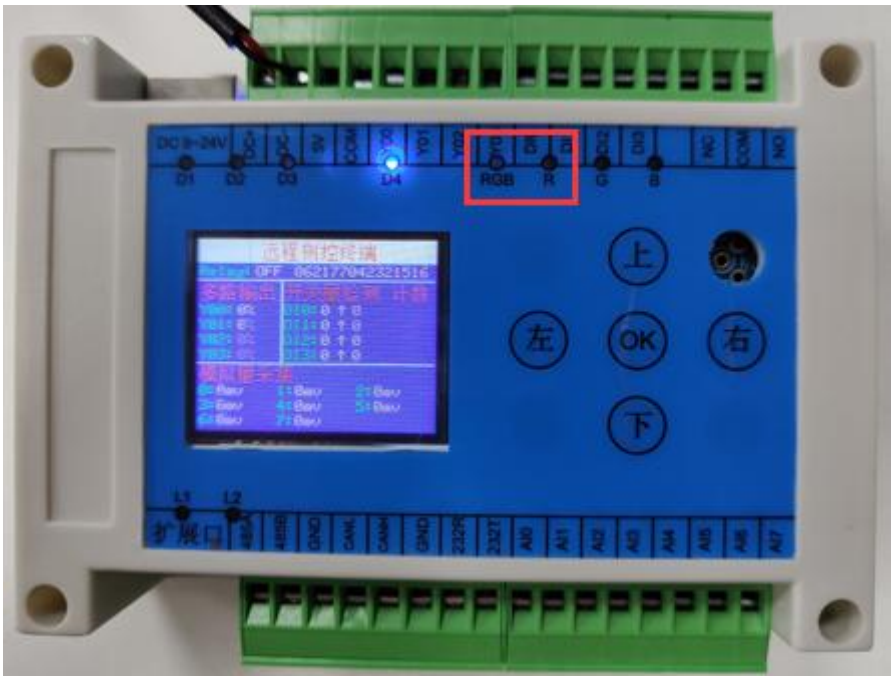


图 4-20 远程测控终端的 R 灯熄灭

将“LED_OFF”修改成“LED_ON”，如下所示：

{"deviceId":"862177042321516","sensorId":"0000","key":"3c32a5861f43448f82bebfdeadc56f05","data":{"value":"LED_ON"},"code":"11111111","connecttype":"MQTT","trantype":"I/O"}

点击“发送”，R 灯会点亮。

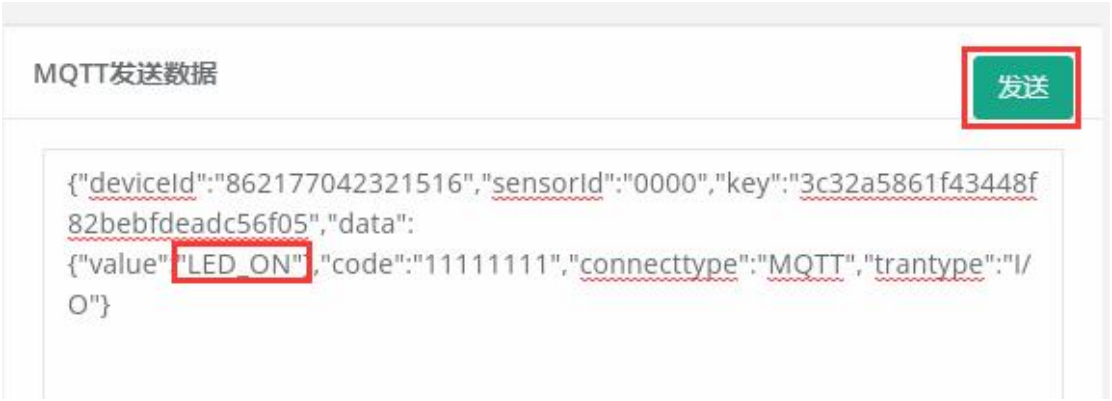


图 4-21 平台发送数据点亮 R 灯

图 4-22 远程测控终端的 R 灯点亮

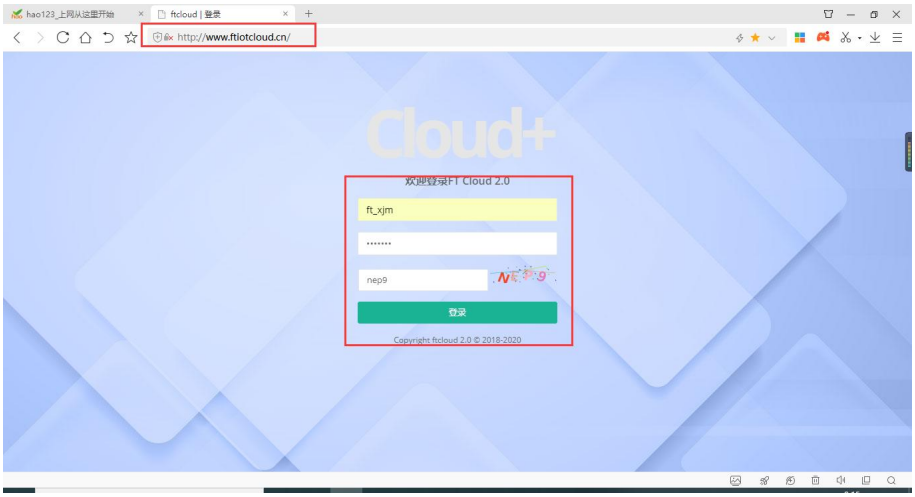
4.8. 工业现场感知终端数据接入平台测试

4.8.1. 项目创建

1) 登录云平台

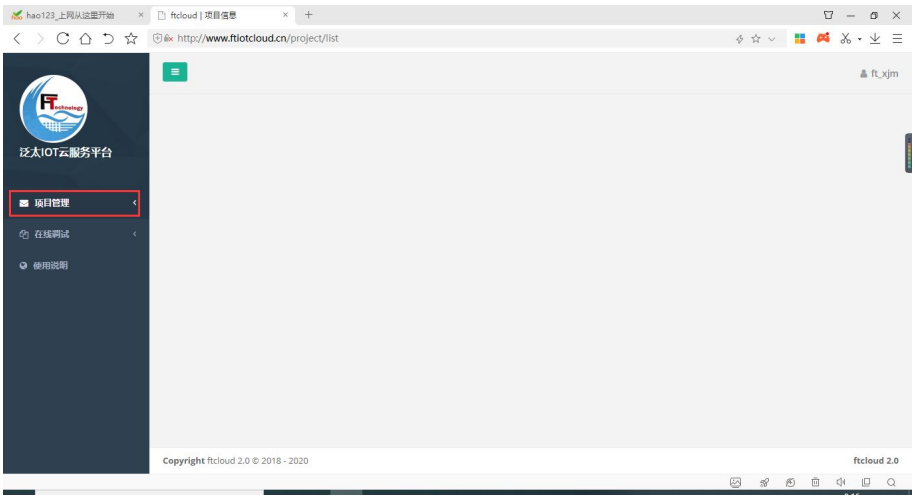
打开浏览器输入云平台网址“http://www.ftiotcloud.cn/”（如果本地化部署，

则输入局域网服务器云平台的网址），使用项目提供的用户名、密码，进行登录，如下图所示。

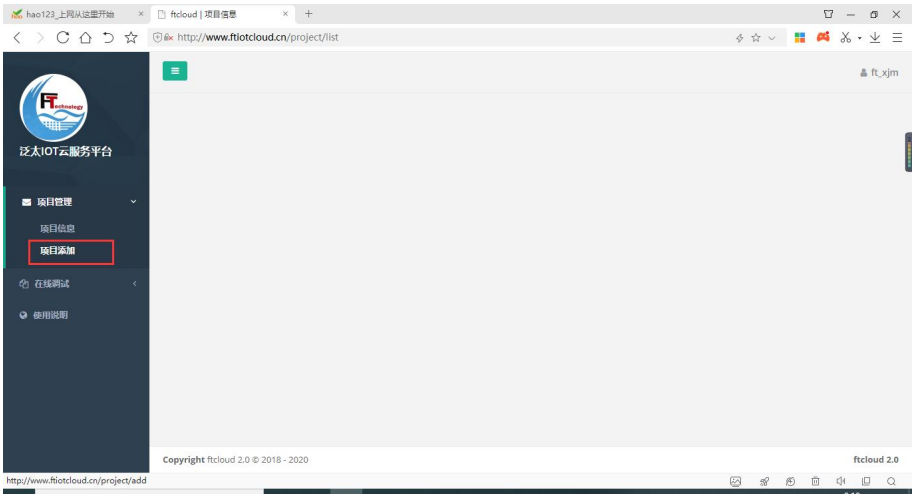


2) 新建项目

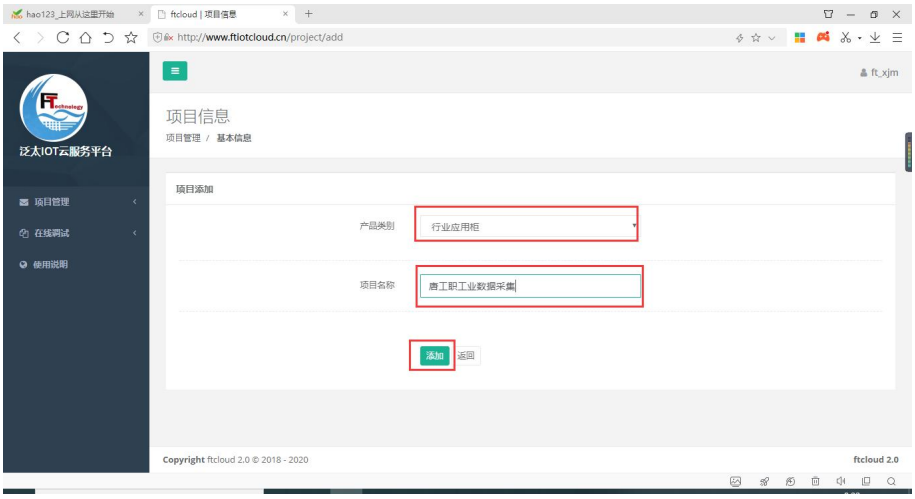
登录成功后，显示如下图所示页面。点击左侧列表中的“项目管理”箭头，自动弹出下拉选项。



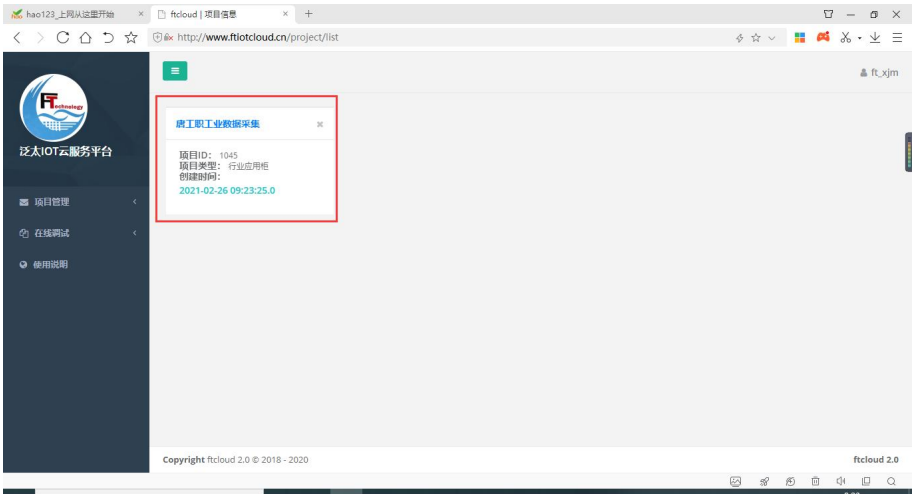
点击“项目添加”，如下图所示。



其中，产品类别：选择接入平台的产品名称，如行业应用柜；项目名称：自定义输入，如“工业数据采集实训台”，点击添加按钮，如下图所示。



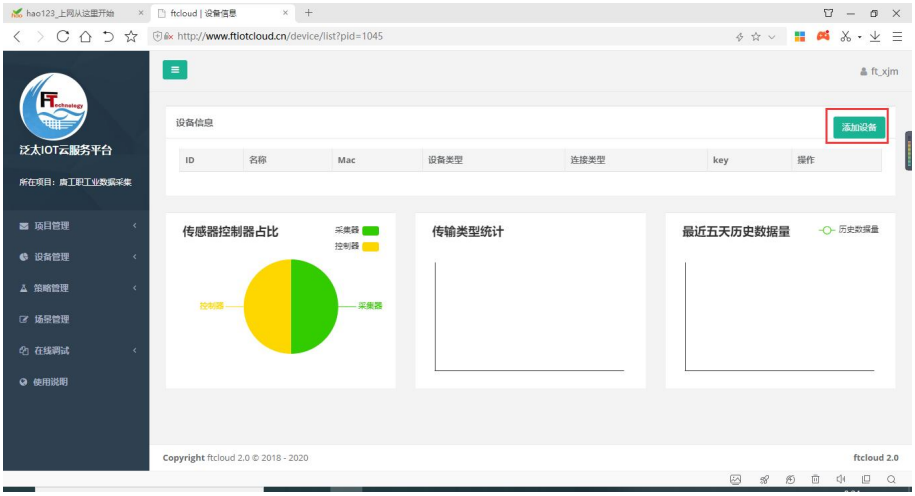
项目添加成功后如下图所示，显示平台为当前项目分配的“项目 ID”、“创建时间”、“项目类型”、蓝色字体的“项目名称”等信息。



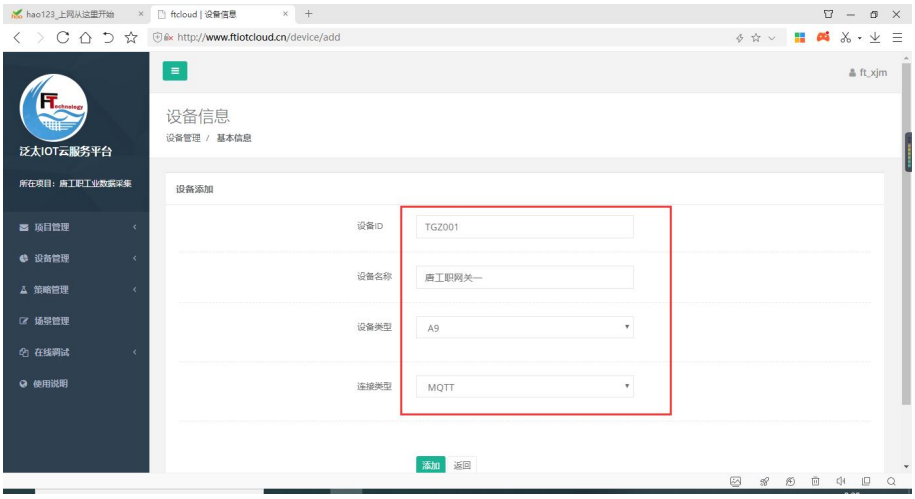
4.8.2. 网关设备添加

点击左侧“项目信息”按钮，右侧主窗体罗列出当前用户下建立的所有项目。点击蓝色字体的“项目名称”，即可进入项目，进行设备（网关）添加。例如，点击项目名称“xxx 工业数据采集”，进入设备添加界面，如图所示。

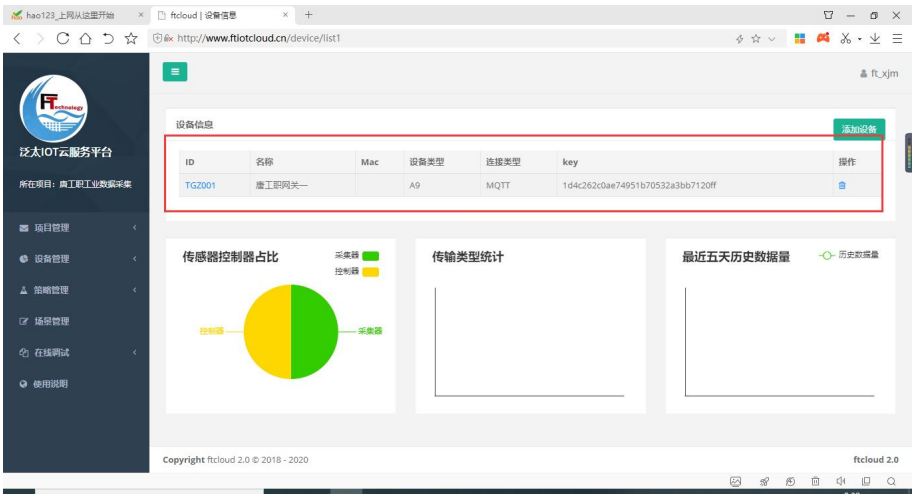
点击“添加设备”按钮。



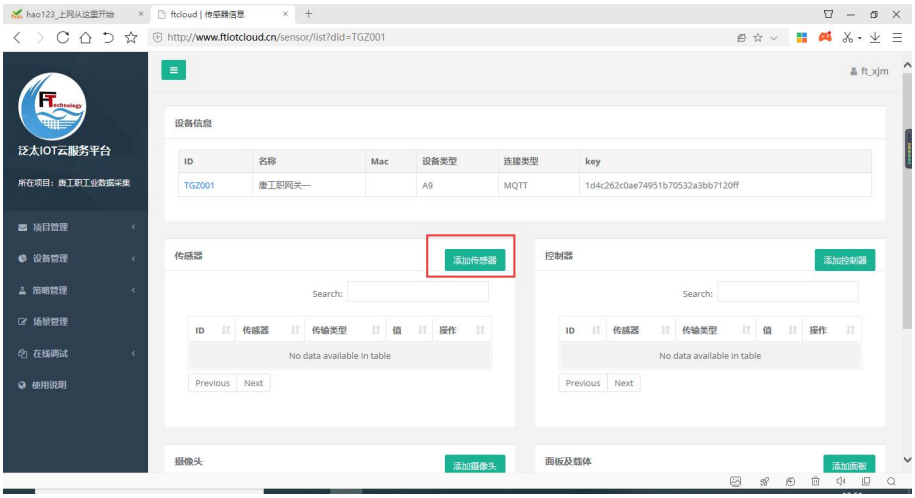
依次输入设备 ID（不能有重复命名），设备名称，选择设备类型，连接类型，点击添加按钮，如下图。



添加设备成功，如下图所示。

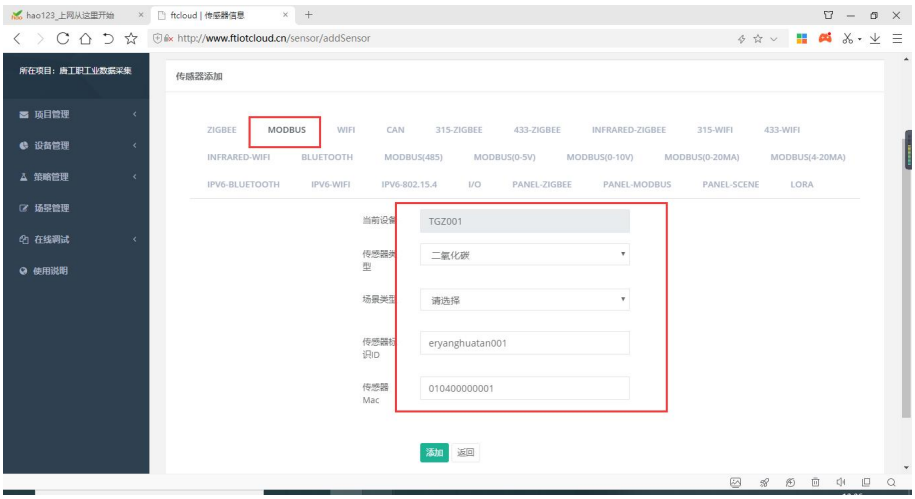


点击蓝色设备 ID，进入设备信息页面，如下图。

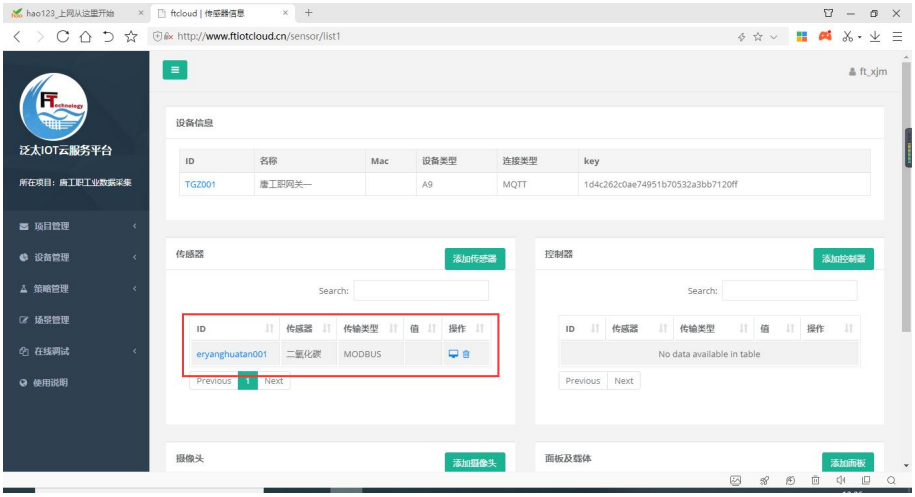


4.8.3. 485 节点云平台接入测试

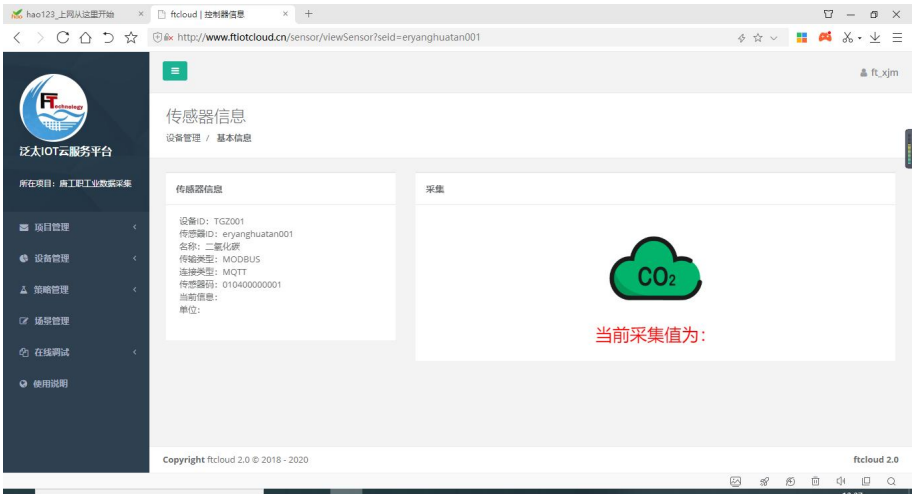
点击添加传感器按钮，进入添加传感器页面。如下图



输入相关信息后，点击添加按钮，添加成功后如下图

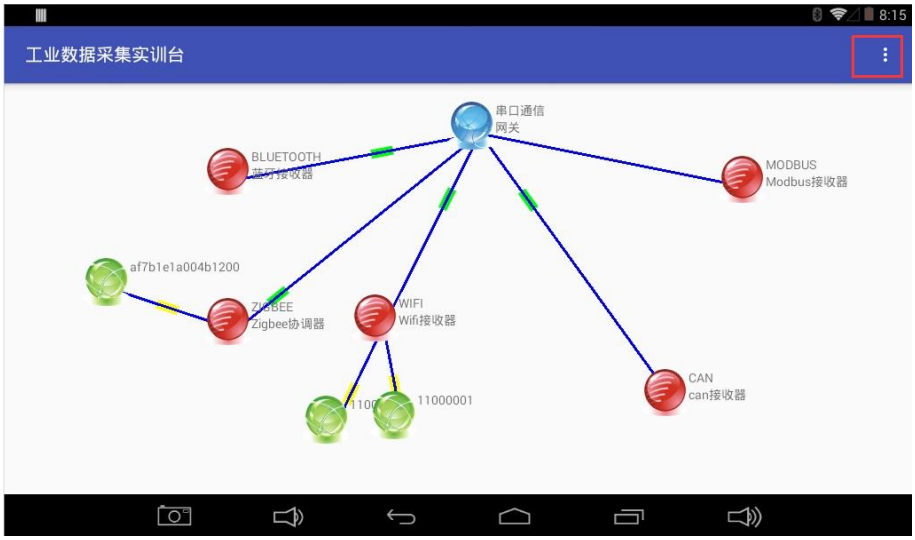


点击传感器 ID 进入传感器详情页面，如下图

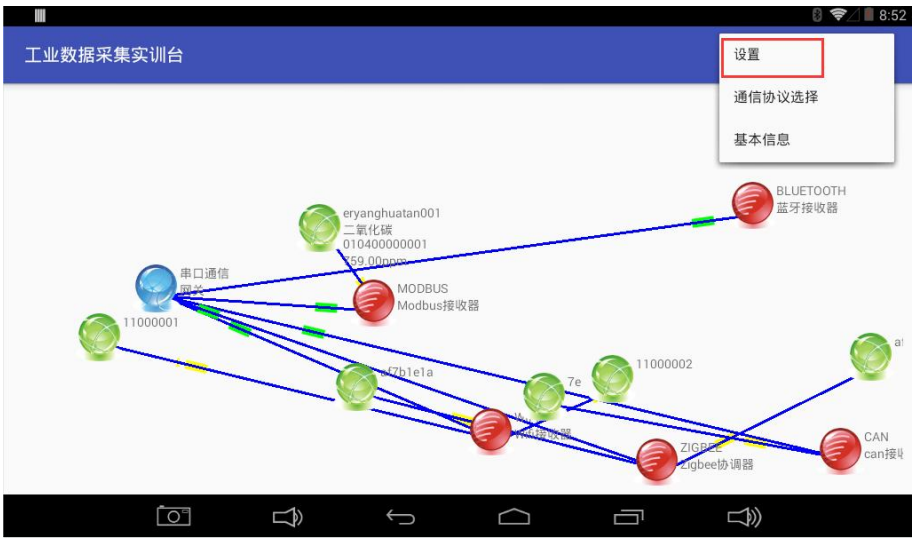


重新打开网关的“工业数据采集实训台”app，如下图标所示

进入应用程序，如下图所示

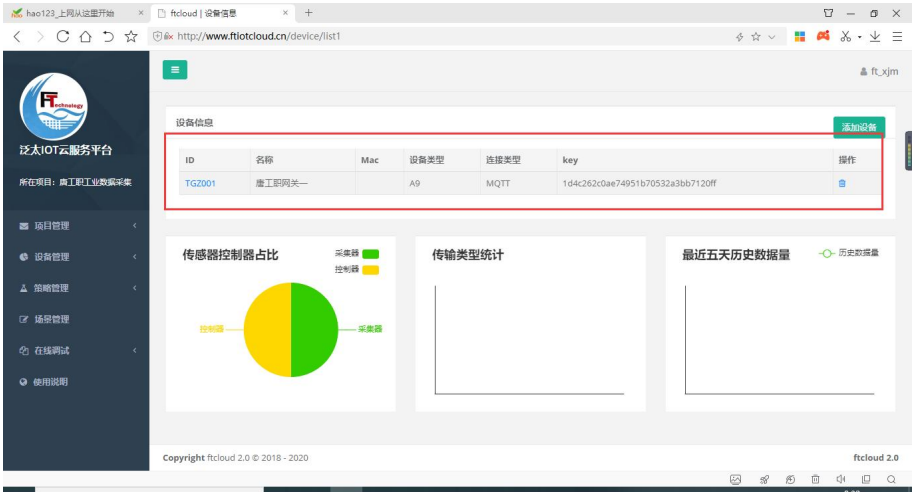


点击右上角按钮，如下图所示。

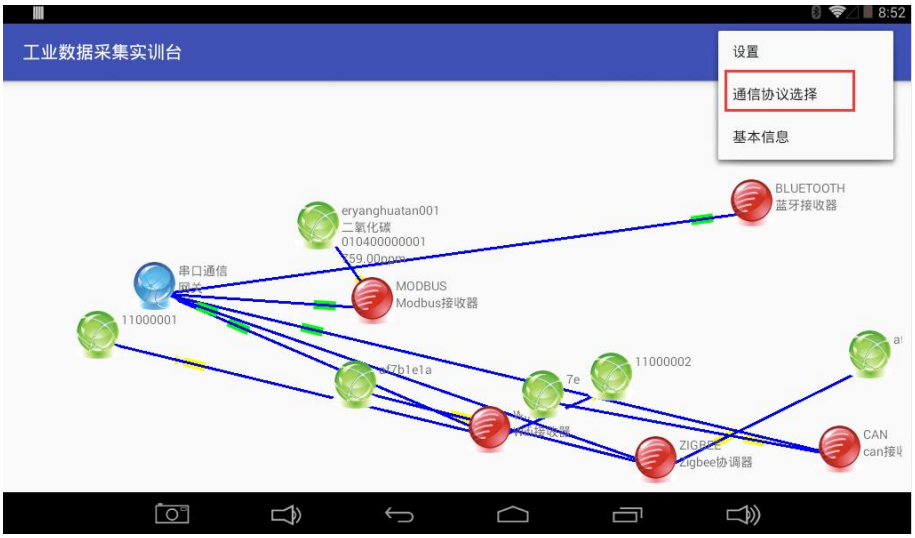


点击“设置”项，打开子页面，如下图所示

修改网关设备编号，即在云平台添加的设备 ID，如下图蓝色字体

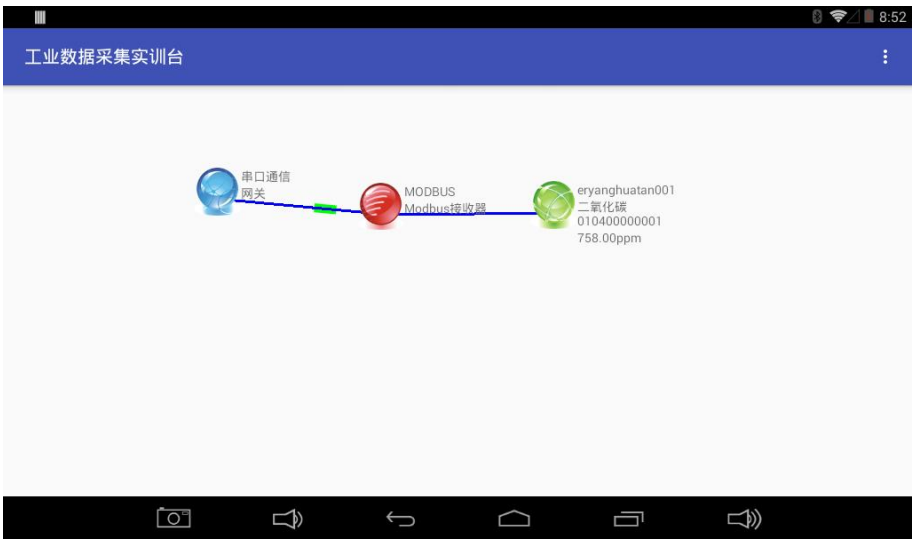


点击“确定修改”按钮，修改成功后，重启网关程序。重启程序后点击右上角按钮，如下图所示。

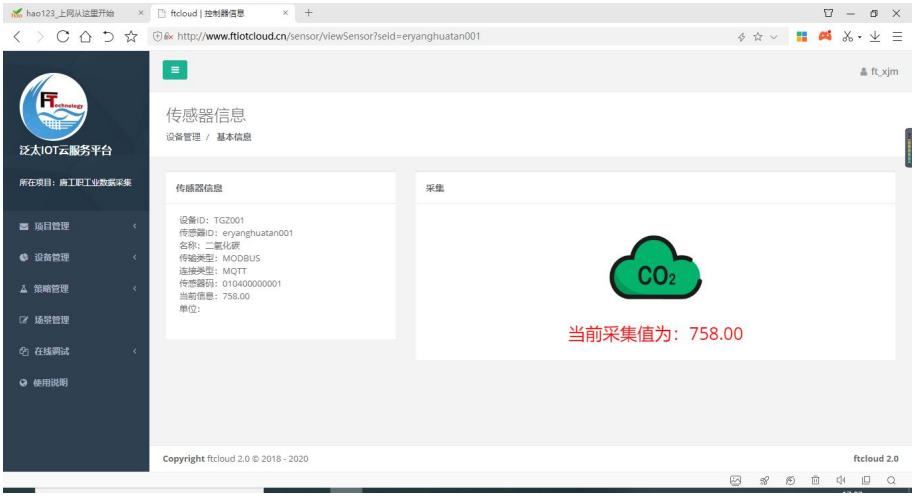


点击“通信协议选择”项，打开子页面，如下图所示

勾选 Modbus 主机选项，点击确定选择按钮，此时只显示 Modbus 协议下的传感器。如下图所示

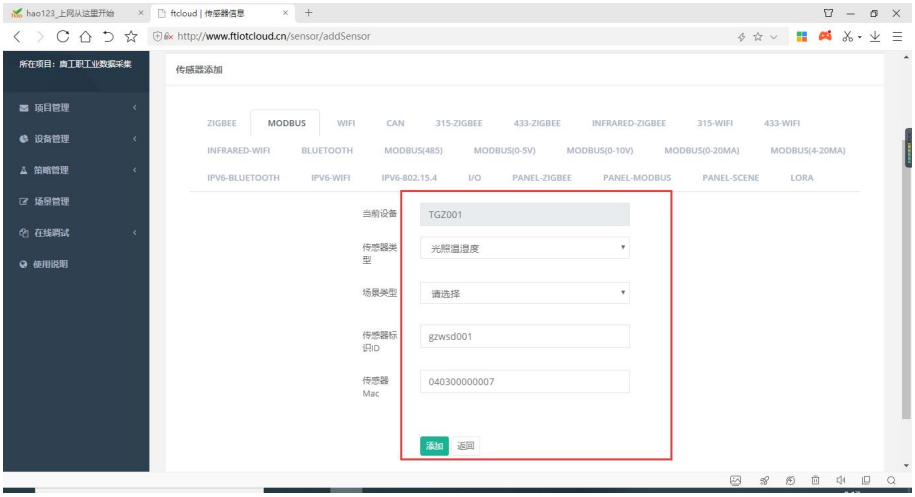


同时传感器的数据会采集到该页面并且同时上传至云平台展示，如下图。

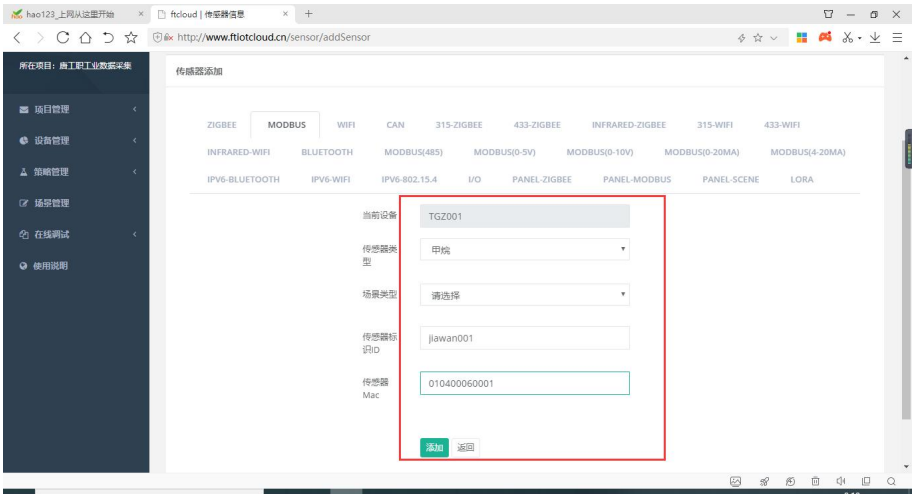


同理添加 Modbus 协议下的其他传感器和执行器至云平台，

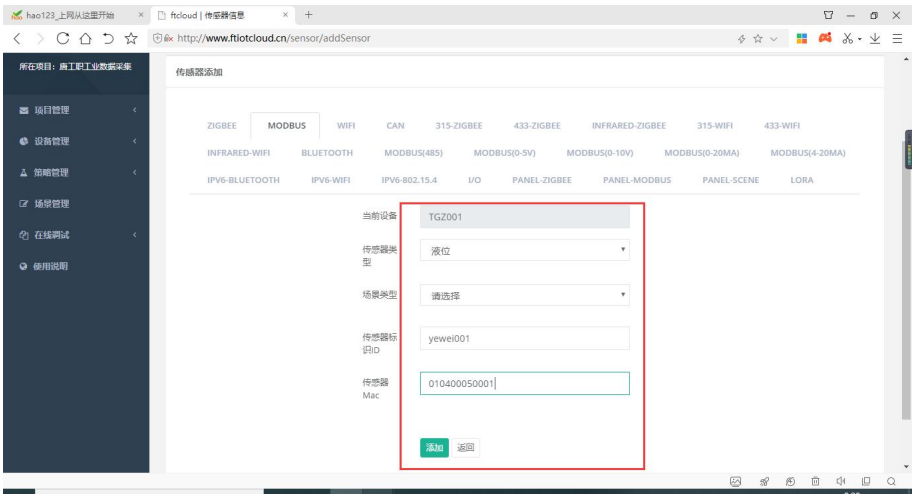
光照温湿度传感器的平台添加界面，如图所示。



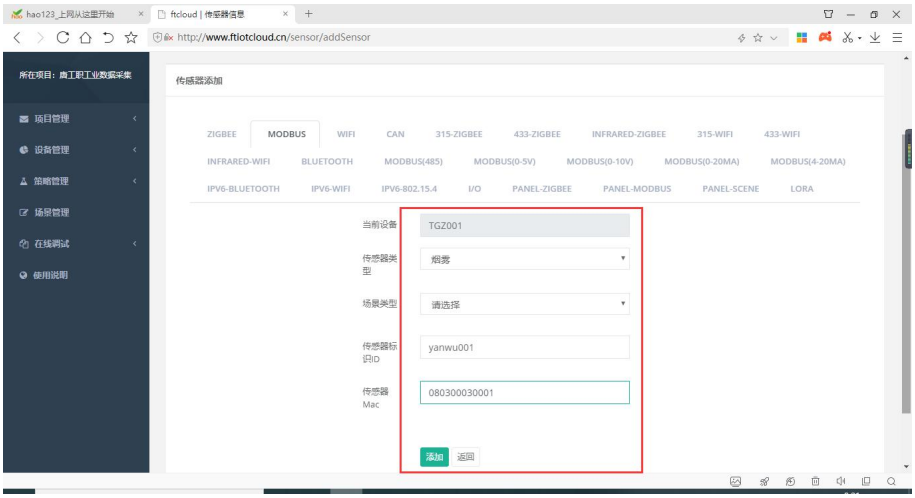
甲烷传感器的平台添加界面，如图所示。



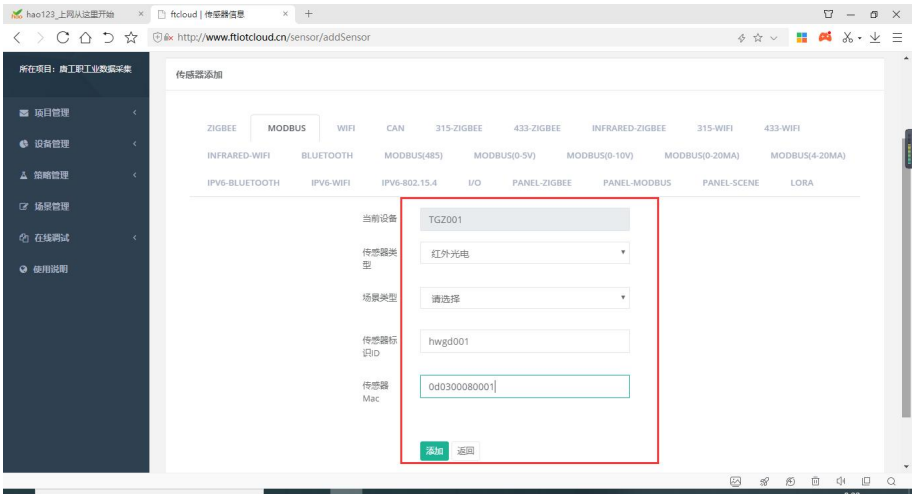
液位传感器的平台添加界面，如图所示。



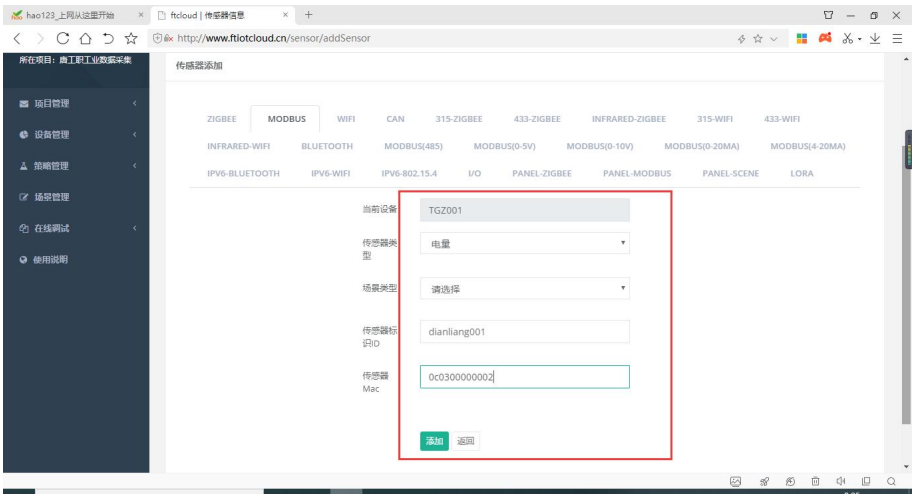
烟雾传感器的平台添加界面，如图所示。



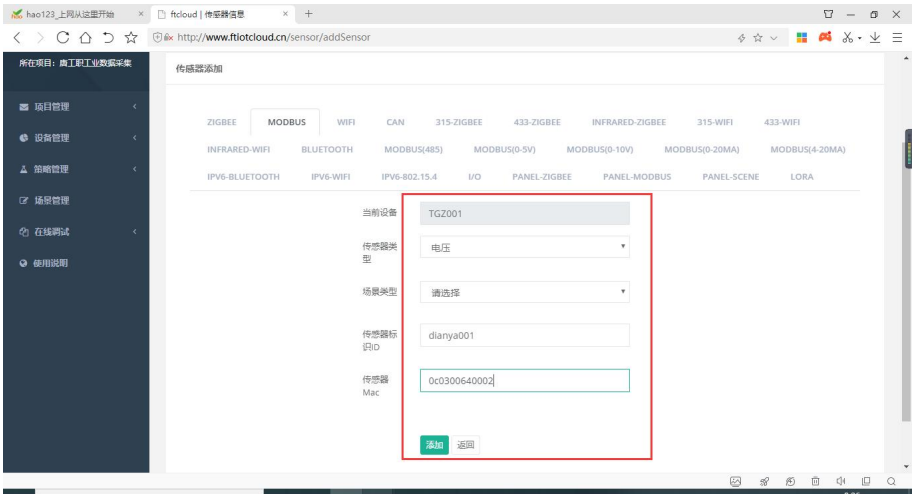
红外光电传感器的平台添加界面，如图所示。



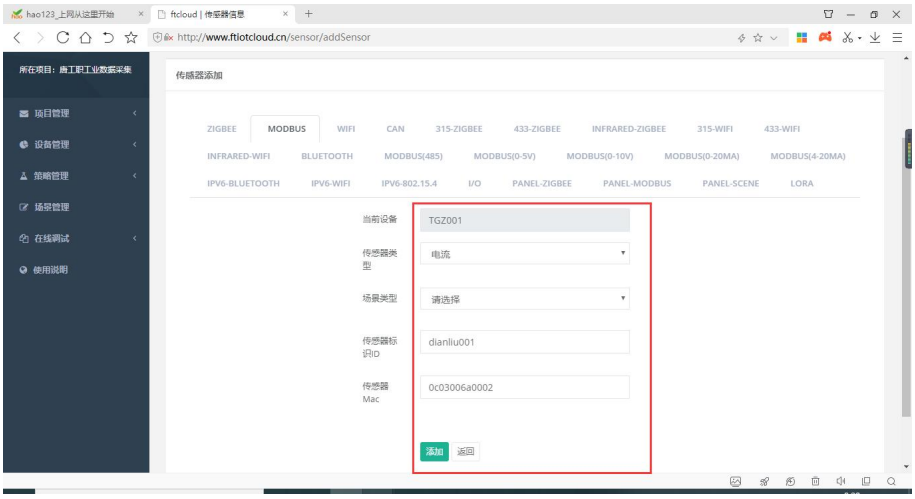
电表-电量传感器的平台添加界面，如图所示。



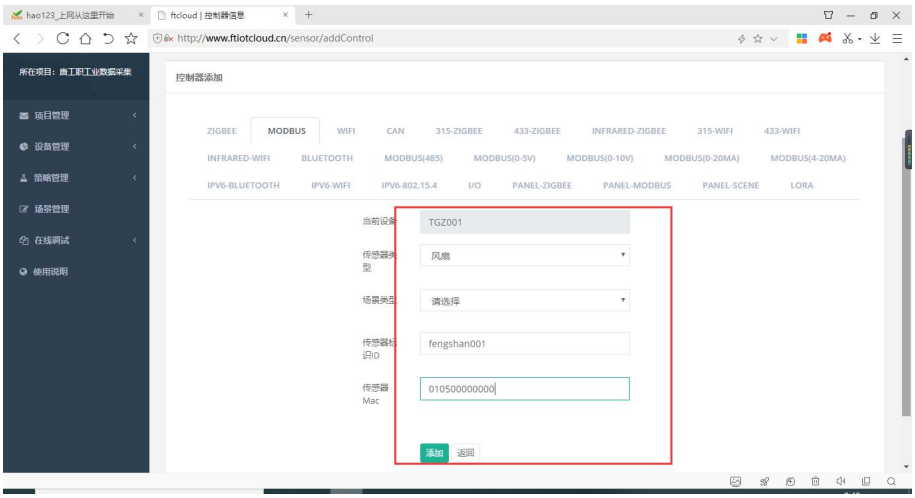
电表-电压传感器的平台添加界面，如图所示。



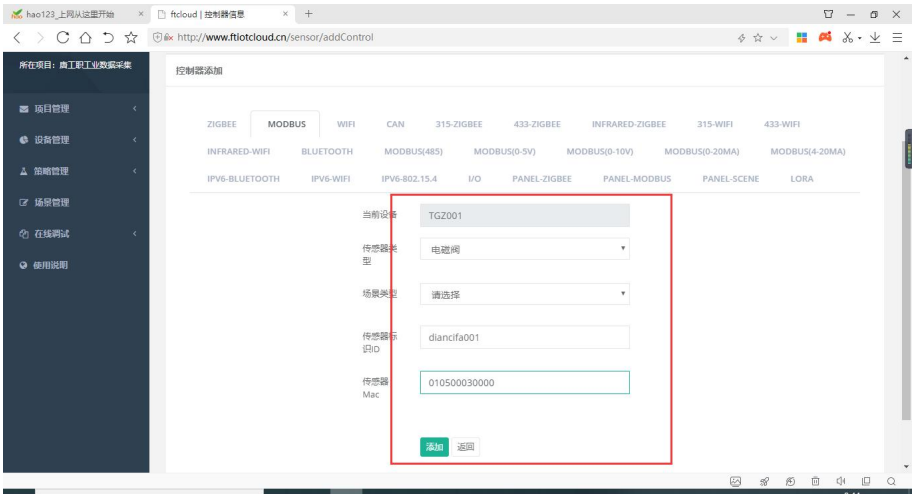
电表-电流传感器的平台添加界面，如图所示。



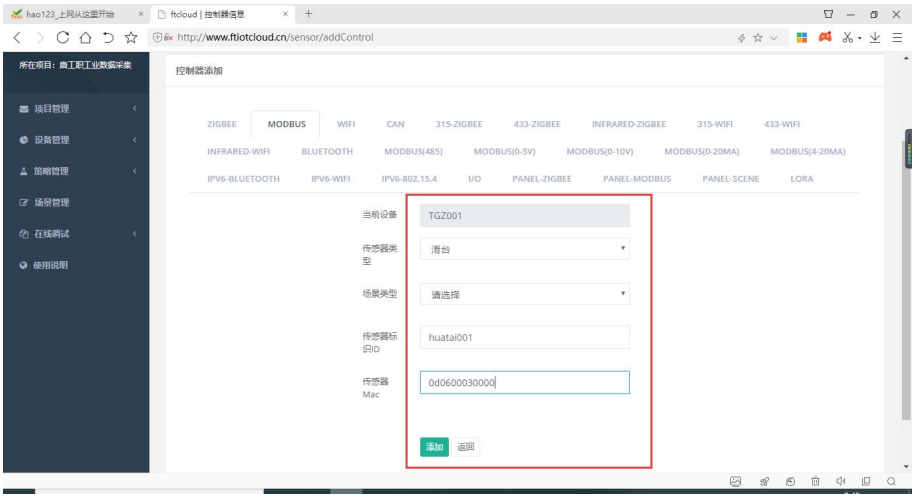
风扇执行器的平台添加界面，如图所示。



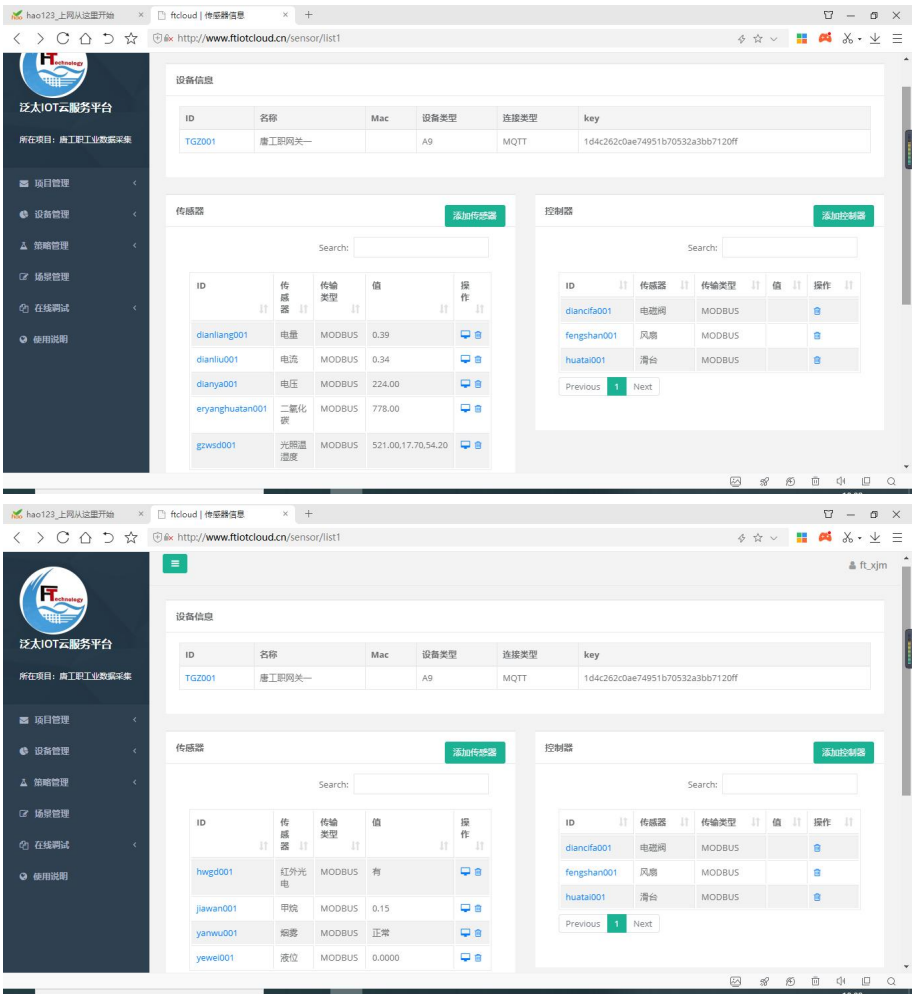
电磁阀执行器的平台添加界面，如图所示。



滑台执行器的平台添加界面，如图所示。



添加完成后，会显示该实训台所有的 Modbus 协议下的设备，如下图所示。

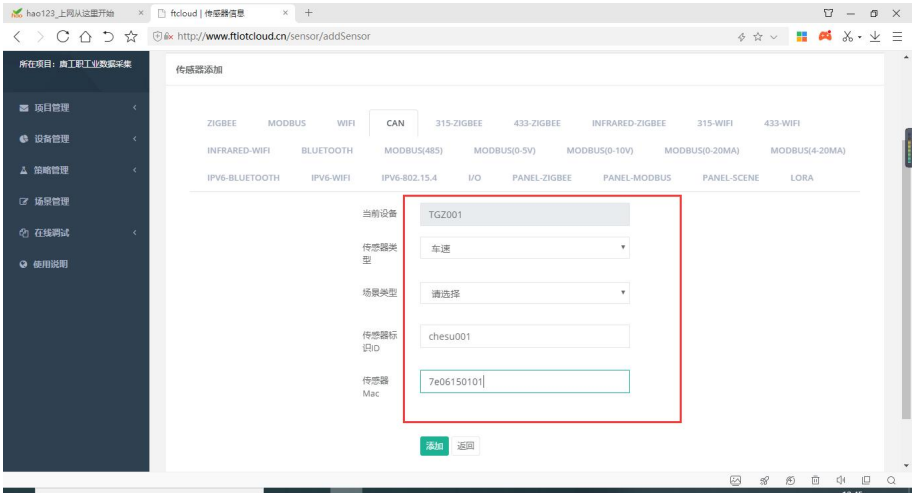


网关端显示如下图所示。

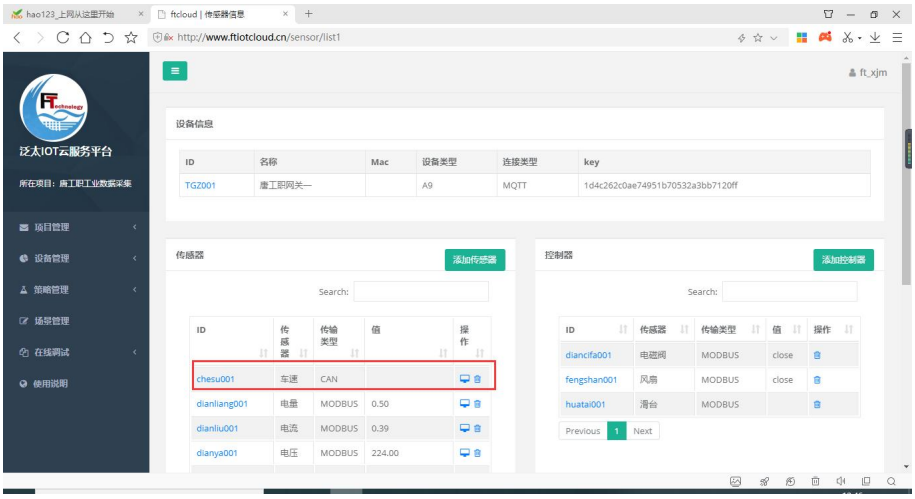
网关端与平台端数据保持同步。

4.8.4. CAN 节点云平台接入测试

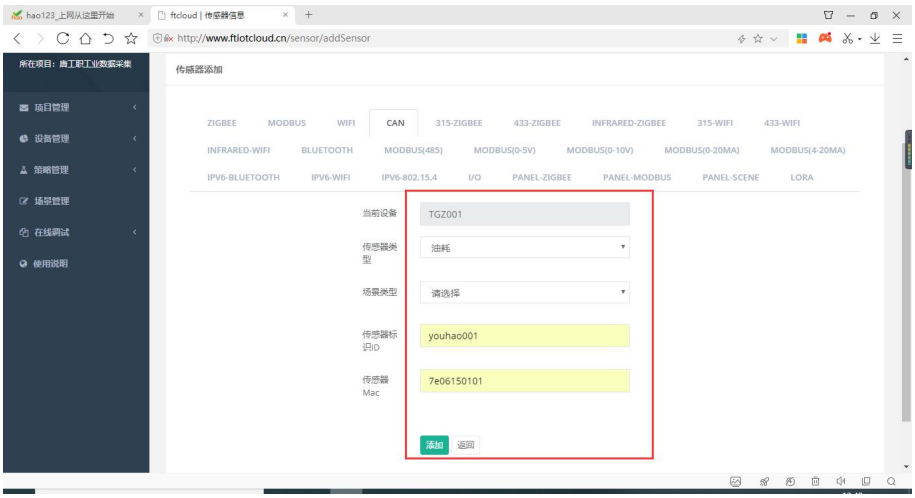
点击添加传感器按钮，进入添加传感器页面。如下图



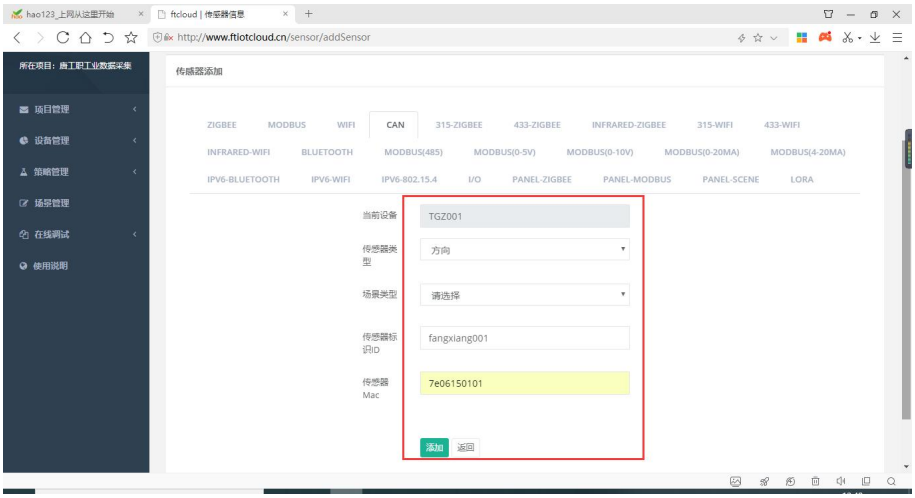
输入车速相关信息后，点击添加按钮，添加成功后如下图



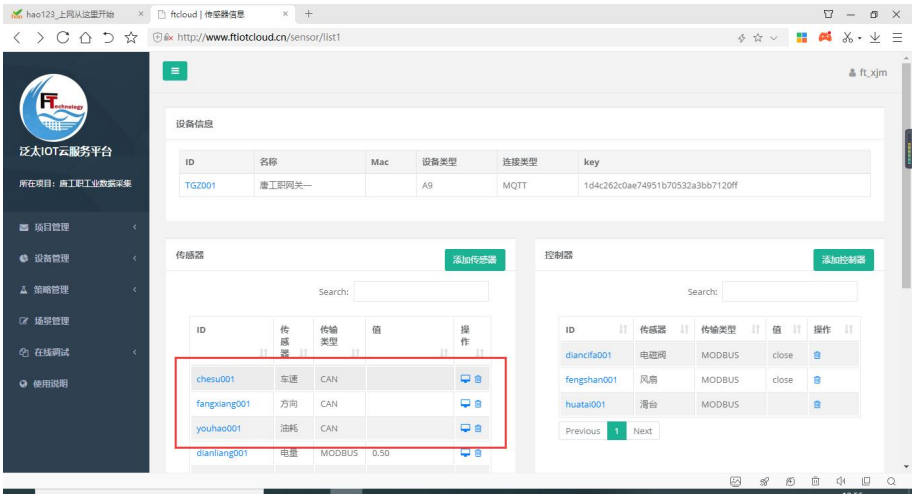
继续添加油耗，如下图



继续添加方向，如下图

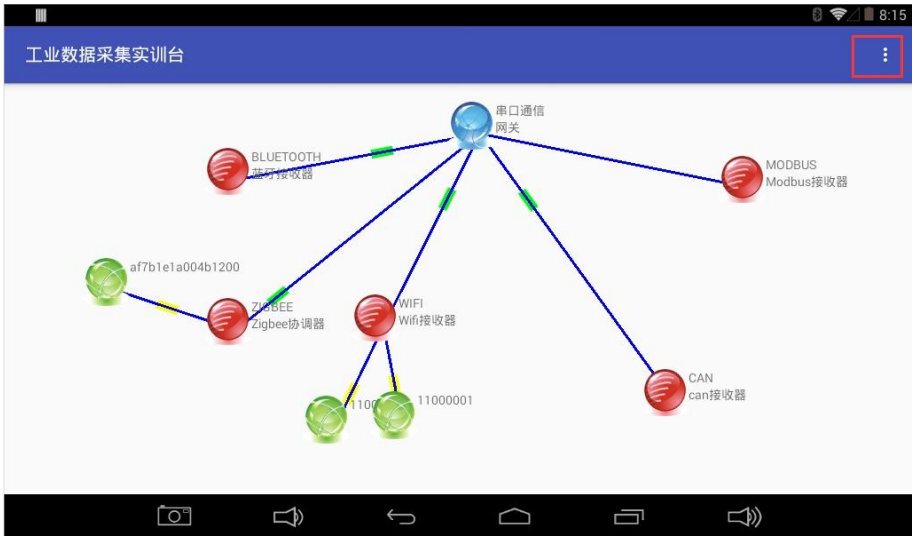


添加完成后如下图所示。

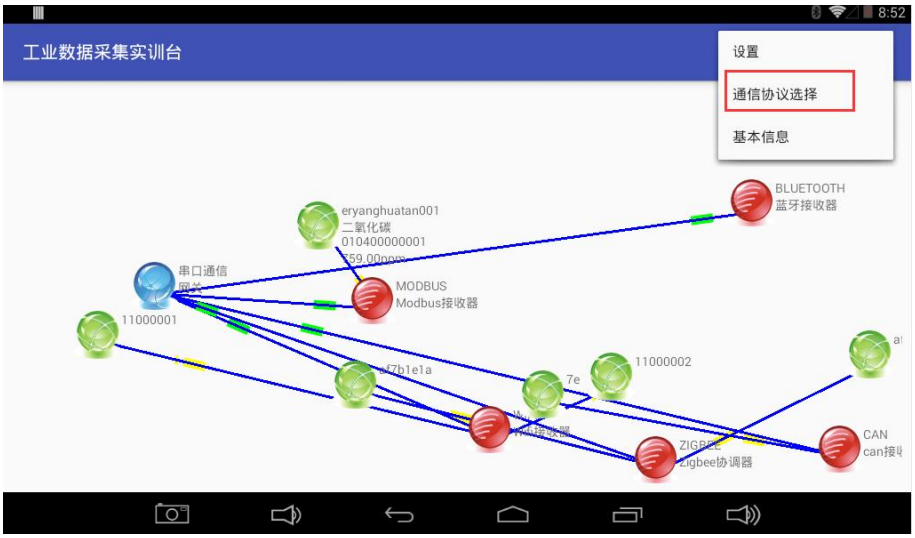


重新打开网关的“工业数据采集实训台”app，如下图标所示

进入应用程序，如下图所示

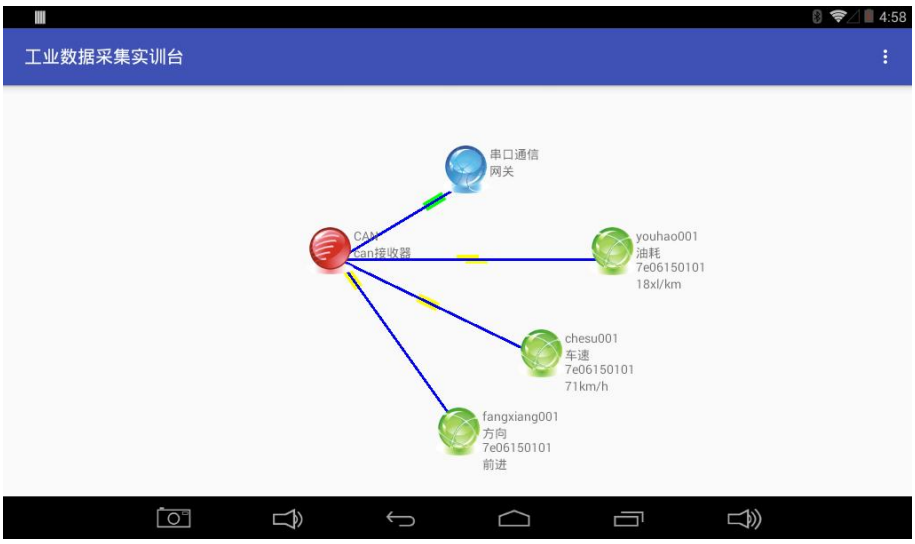


点击右上角按钮，如下图所示。

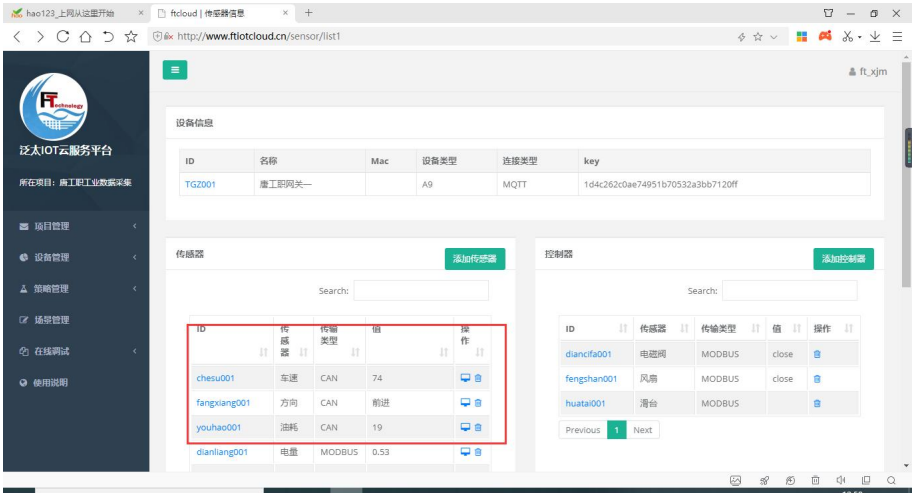


点击“通信协议选择”项，打开子页面，如下图所示

勾选 Can 主机选项，点击确定选择按钮，此时只显示 Can 协议下的传感器。如下图所示



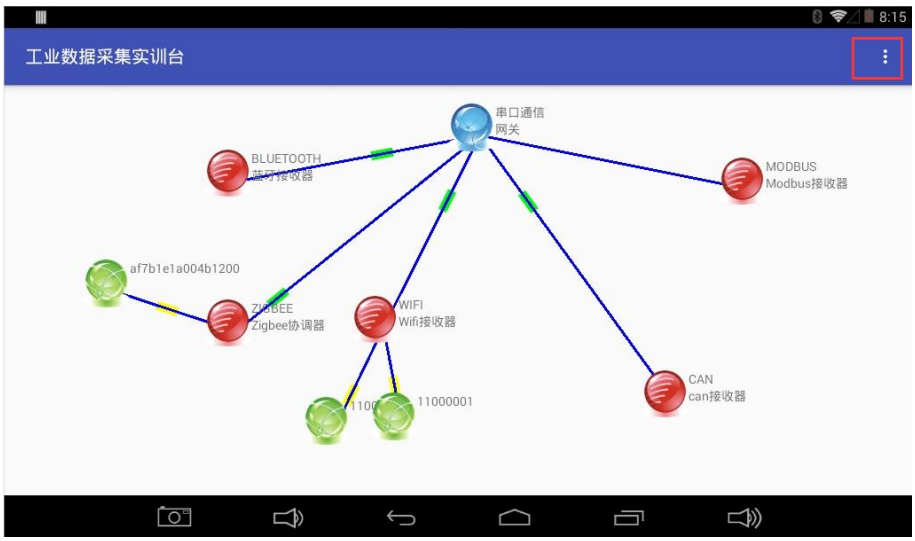
同时传感器的数据会采集到该页面并且同时上传至云平台展示，如下图



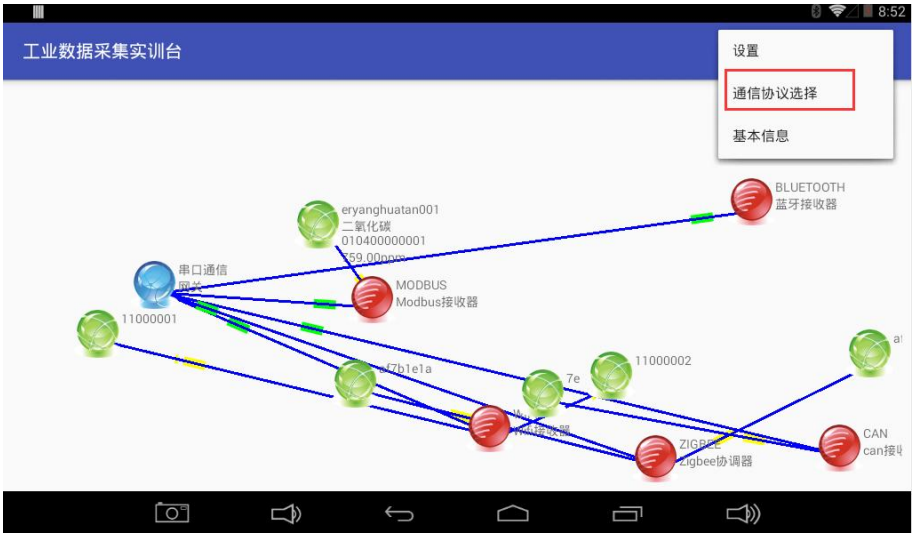
4.8.5. ZigBee 节点云平台接入测试

打开网关的“工业数据采集实训台”app，如下图标所示

进入应用程序，如下图所示



点击右上角按钮，如下图所示。

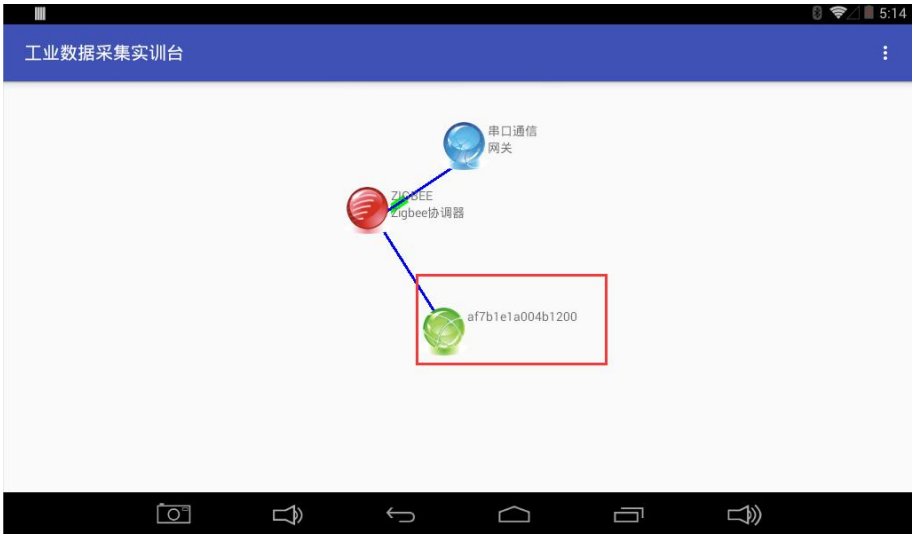


点击“通信协议选择”项，打开子页面，如下图所示

勾选 Zigbee 协调器选项，点击确定选择按钮，此时只显示 Zigbee 协议下的

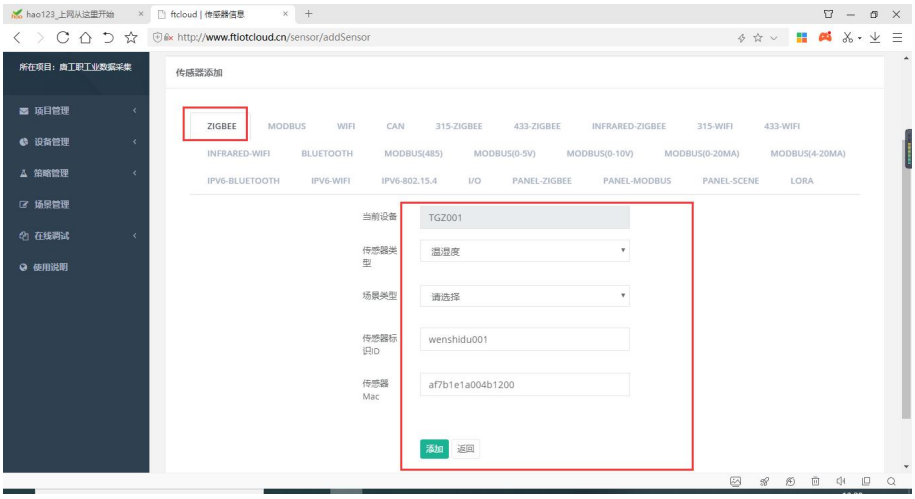
传感器。如下图所示

此时找到无线传感网区域的最左侧两个 zigbee 通信协议下的传感器节点，按下温湿度节点的 SW1 按钮，显示如下图。

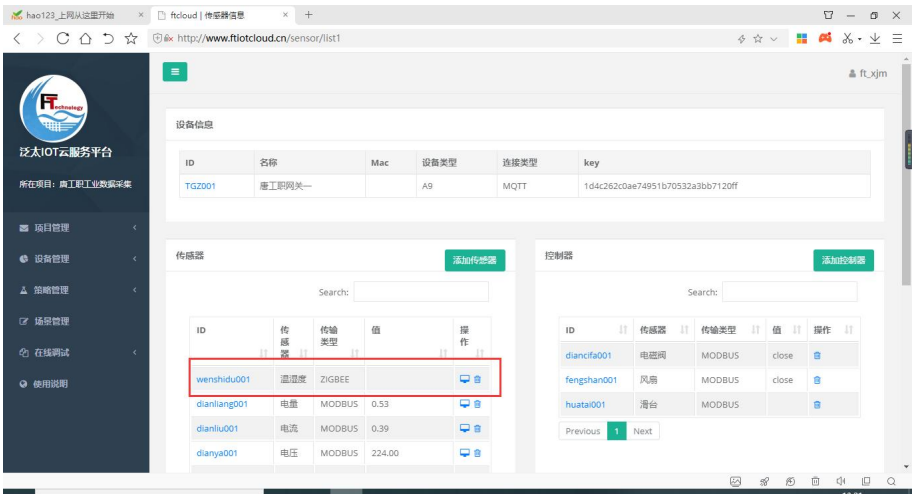


应用中页面会添加一个圆球，和该传感器节点的 MAC 地址

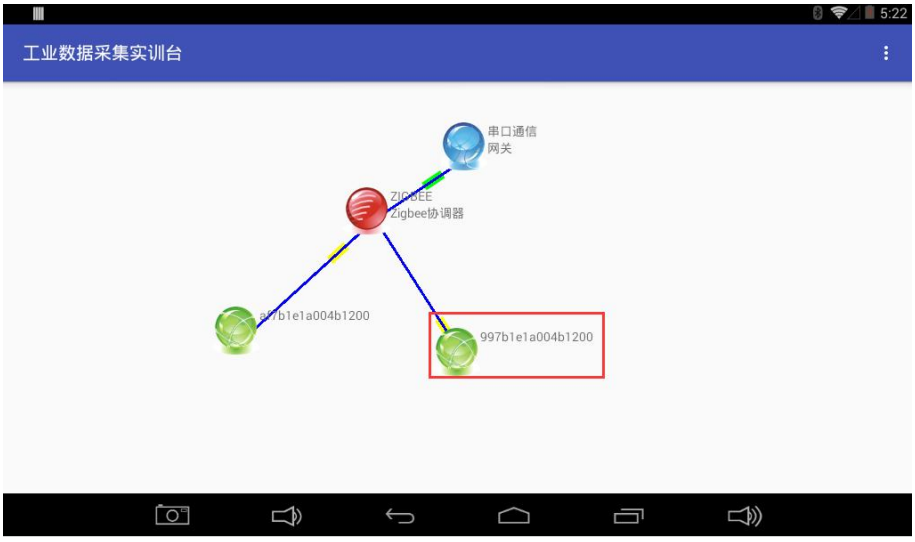
此时打开云平台，点击添加传感器按钮，进入添加传感器页面。如下图



输入 zigbee 温湿度传感器相关信息后，点击添加按钮，添加成功后如下图

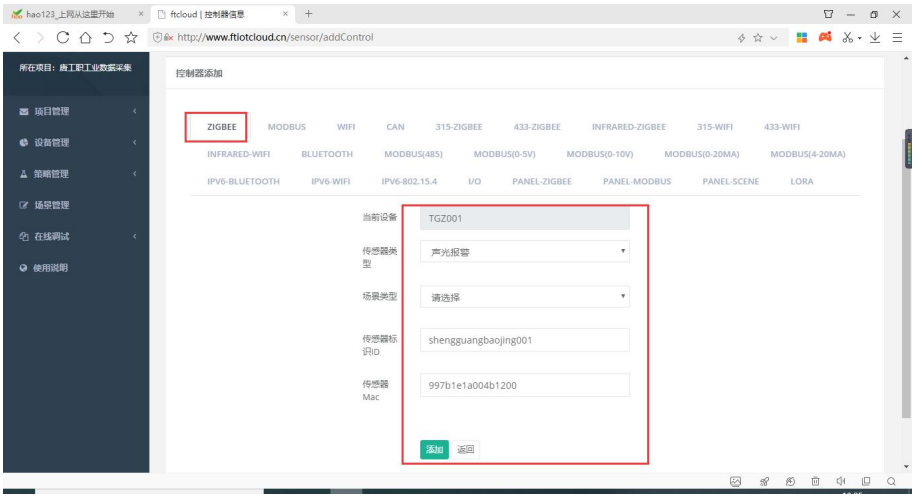


同理继续添加声光报警节点，此时找到无线传感网区域的最左侧两个 zigbee 节点，按下声光报警节点的 SW1 按钮，显示如下图。

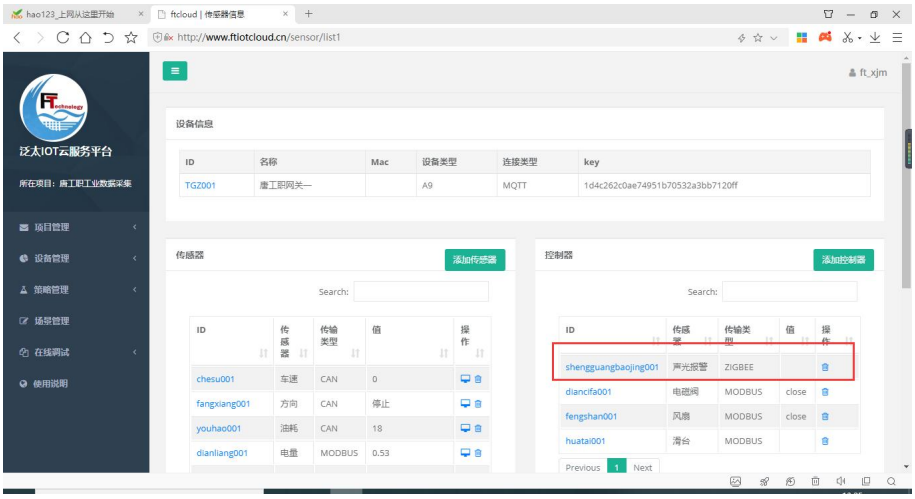


应用中页面会添加一个圆球，和该传感器节点的 MAC 地址

此时打开云平台，点击添加控制器按钮，进入添加控制器页面。如下图

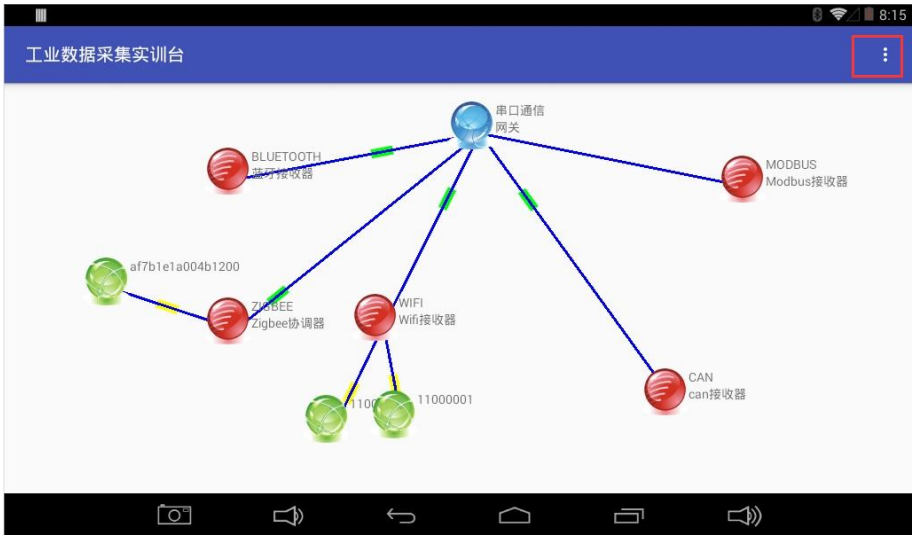


输入 zigbee 声光报警控制器相关信息后，点击添加按钮，添加成功后如下图

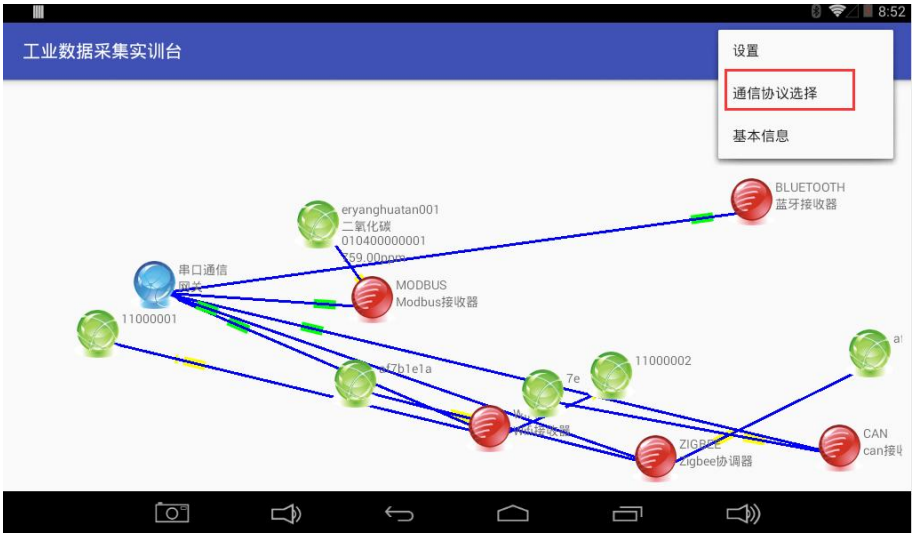


重新打开网关的“工业数据采集实训台”app，如下图标所示

进入应用程序，如下图所示



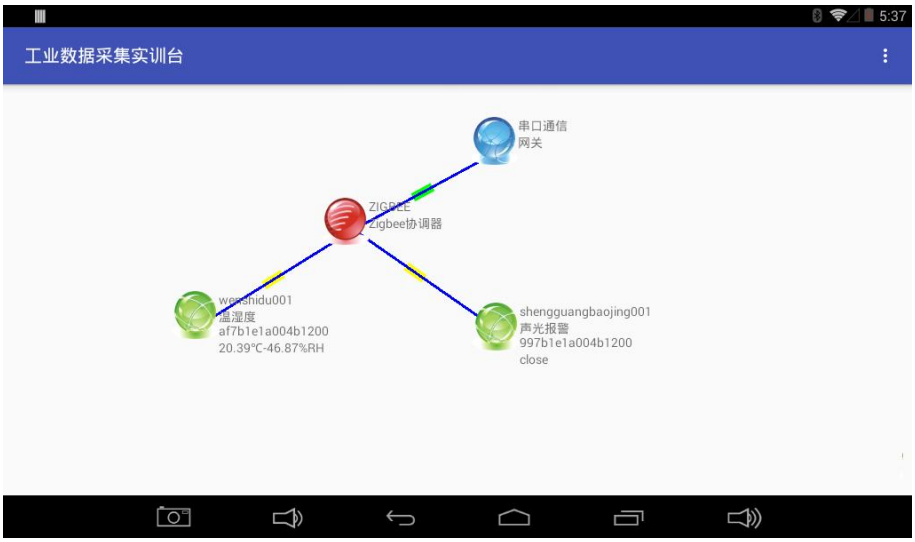
点击右上角按钮，如下图所示。



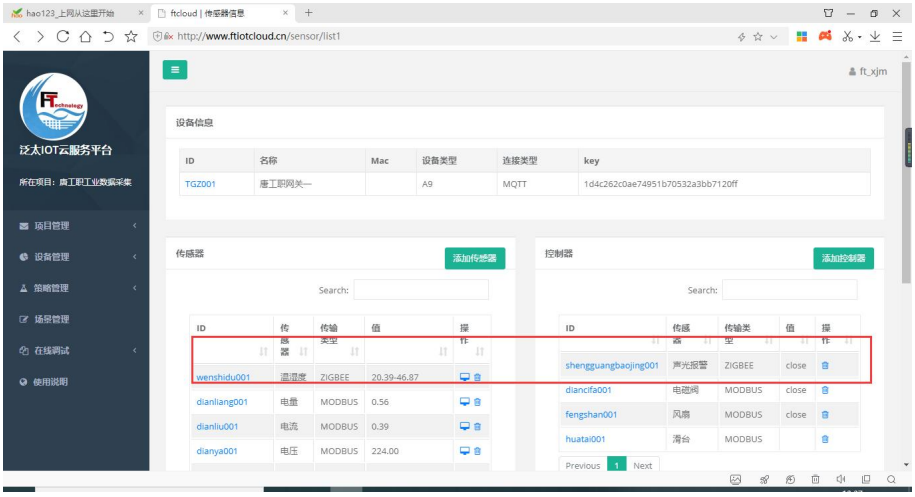
点击“通信协议选择”项，打开子页面，如下图所示

勾选 Zigbee 协调器选项，点击确定选择按钮，此时只显示 Zigbee 协议下的

传感器。如下图所示。

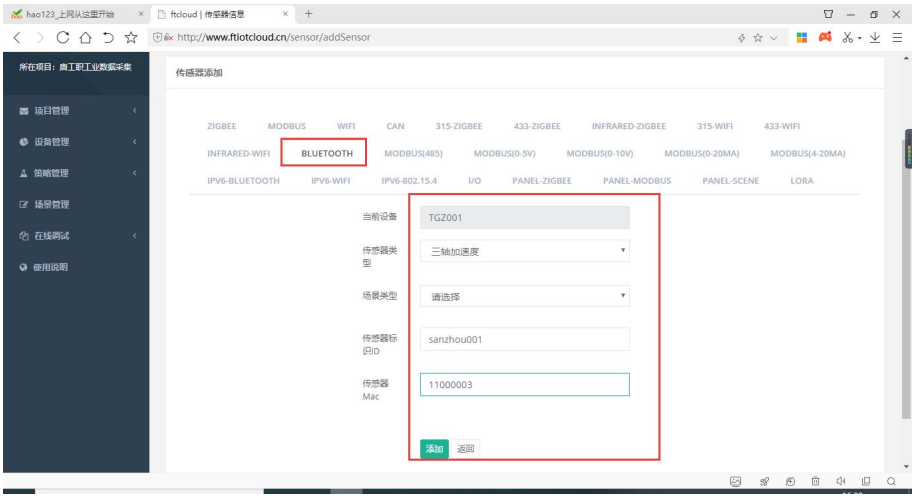


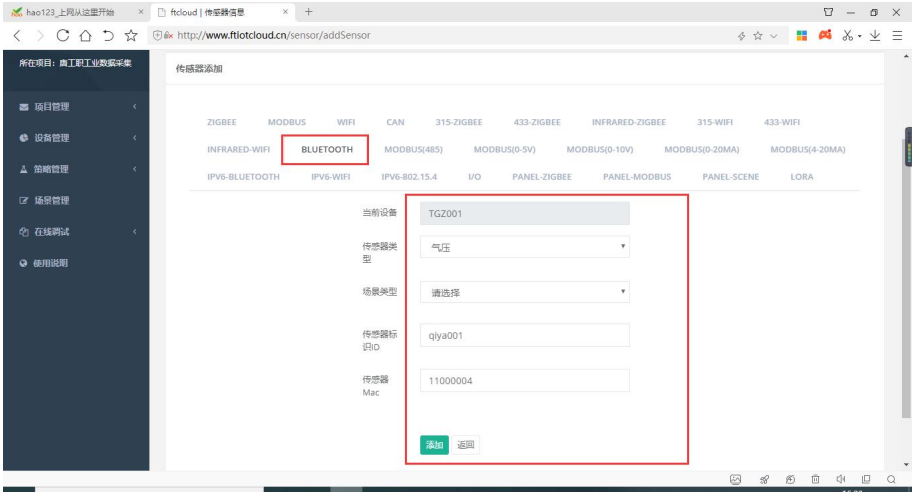
同时传感器的数据会采集到该页面并且同时上传至云平台展示，如下图



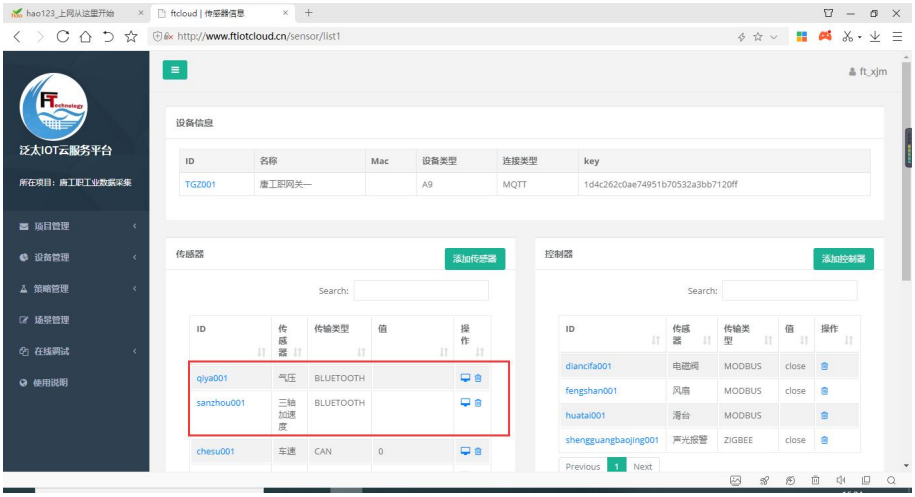
4.8.6. 蓝牙节点云平台接入测试

点击添加传感器按钮，进入添加传感器页面。如下图，此时找到蓝牙通信协议下的三轴加速度节点和气压节点，根据节点屏幕上第二行显示的节点地址添加对应的传感器节点。



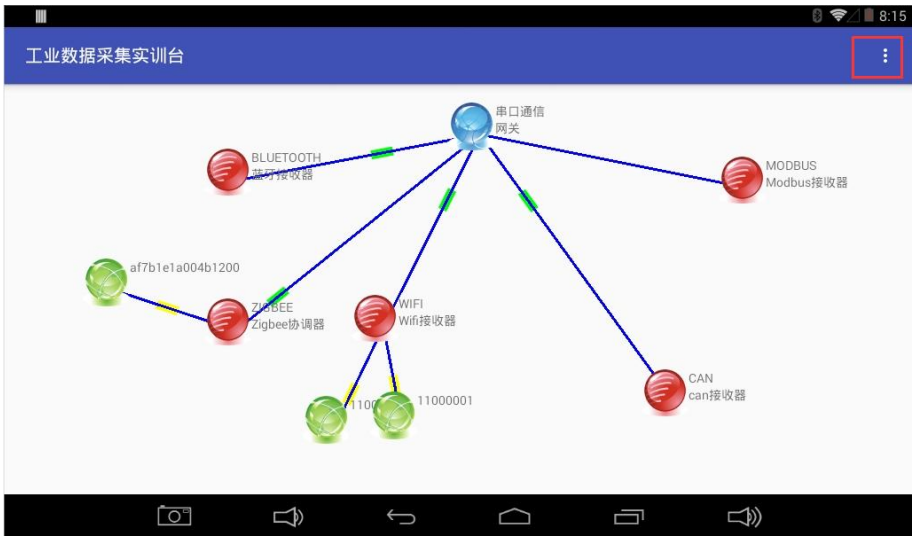


输入相关信息后，点击添加按钮，添加成功后如下图

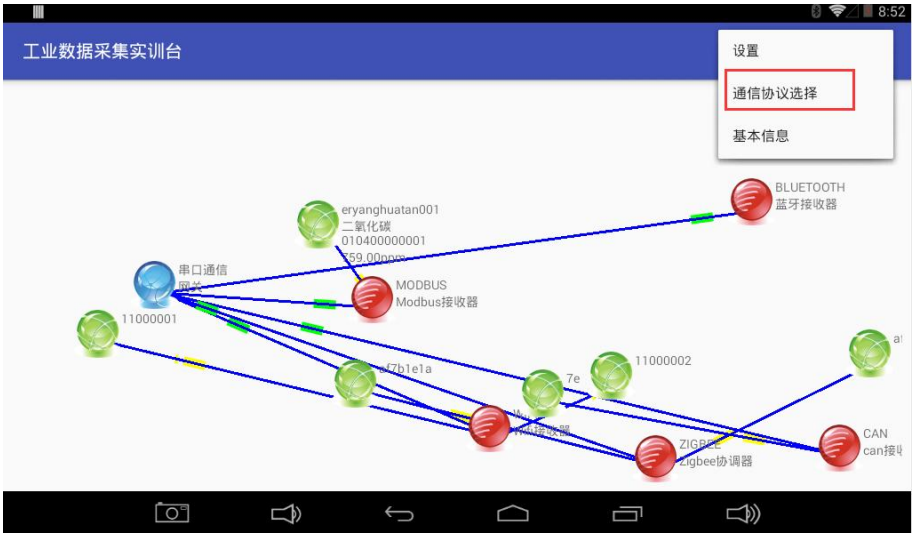


重新打开网关的“工业数据采集实训台”app，如下图标所示

进入应用程序，如下图所示

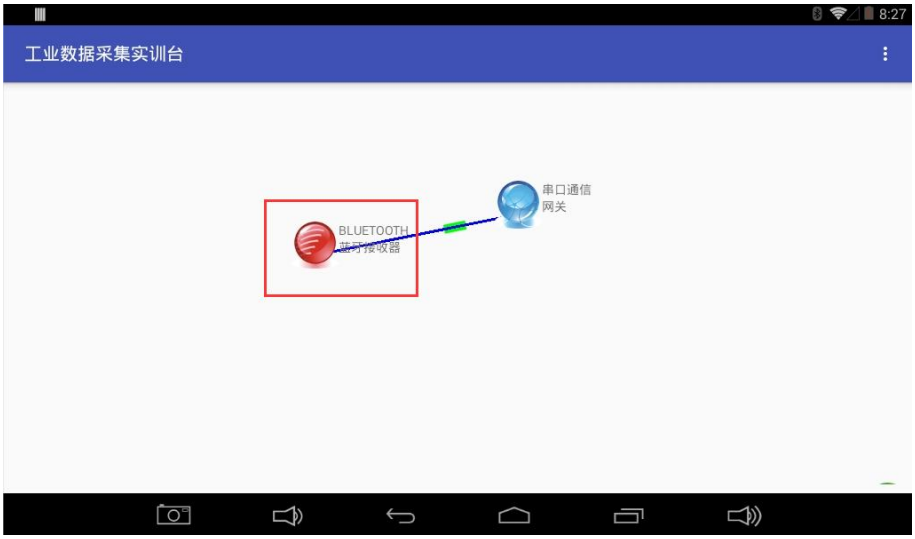


点击右上角按钮，如下图所示。



点击“通信协议选择”项，打开子页面，如下图所示

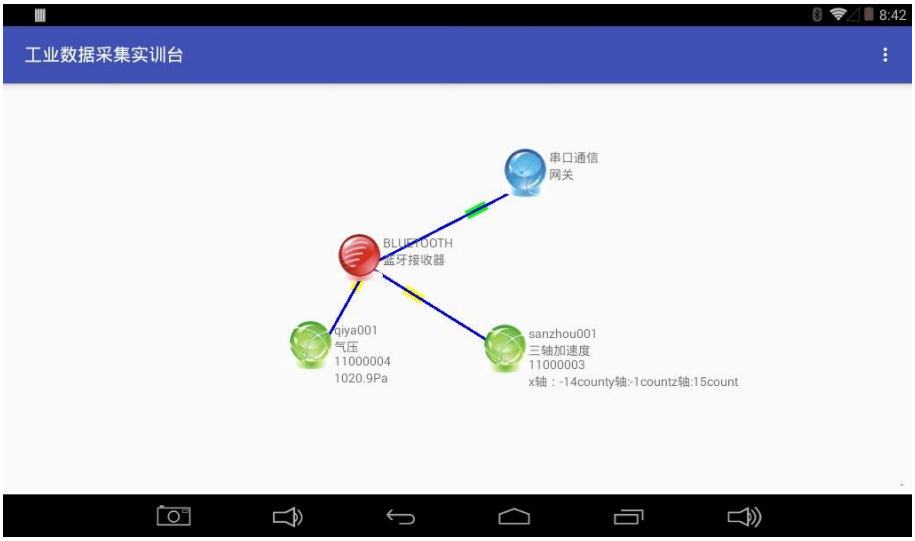
勾选蓝牙主机选项，点击确定选择按钮，此时只显示蓝牙协议下的传感器。如下图所示



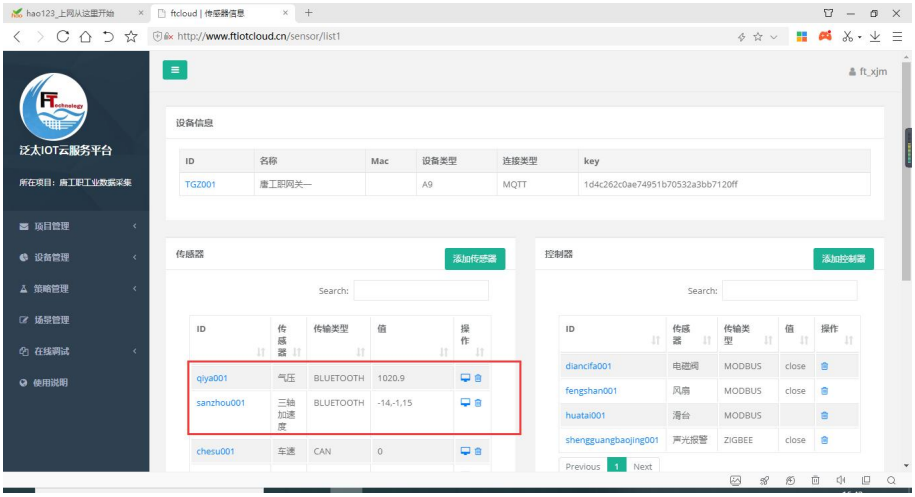
当前因没有蓝牙设备连接所以只显示蓝牙协议小球，此时点击红色小球，弹出搜索蓝牙页面，如下图

找到与当前实训台序号相符的蓝牙设备比如上图所示，点击列表 item 后，如下图

当列表显示已连接，此时点击上方的确认连接按钮，蓝牙通信协议下的传感器会显示到页面上。

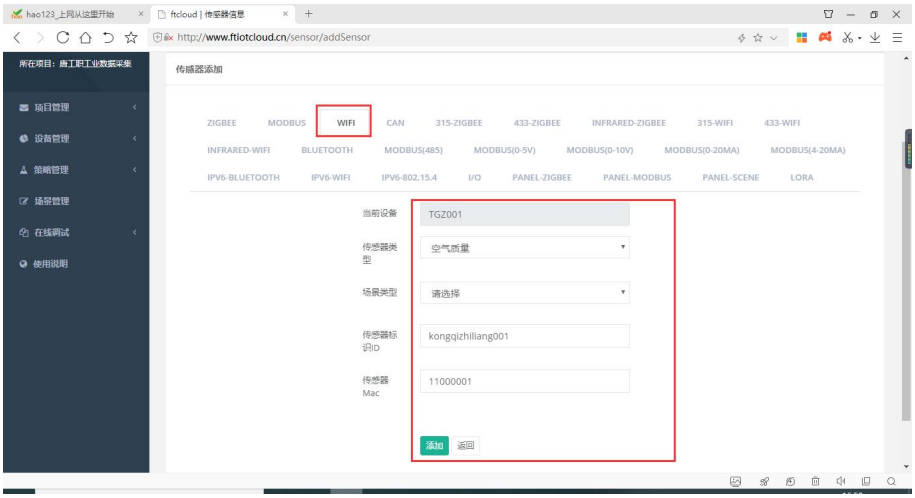


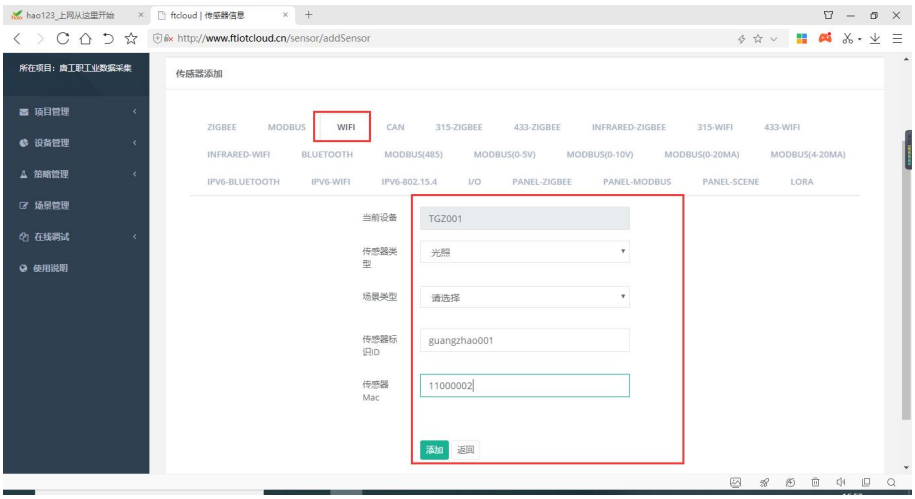
同时传感器的数据实时上传至云平台，如下图



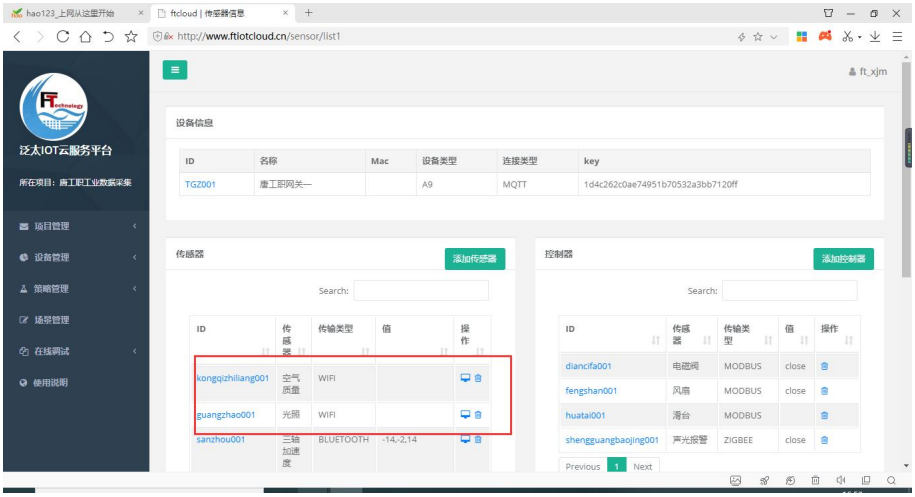
4.8.7. WiFi 节点云平台接入测试

点击添加传感器按钮，进入添加传感器页面。如下图，此时找到无线传感网区域的最右侧两个 wifi 通信下的光照节点和空气质量节点，根据节点屏幕上第二行显示的节点地址添加对应的传感器节点。



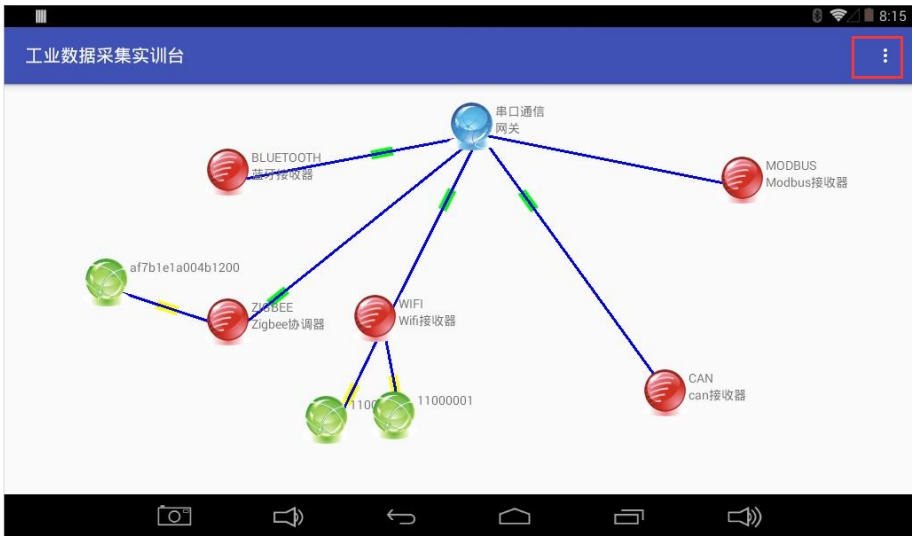


输入相关信息后，点击添加按钮，添加成功后如下图

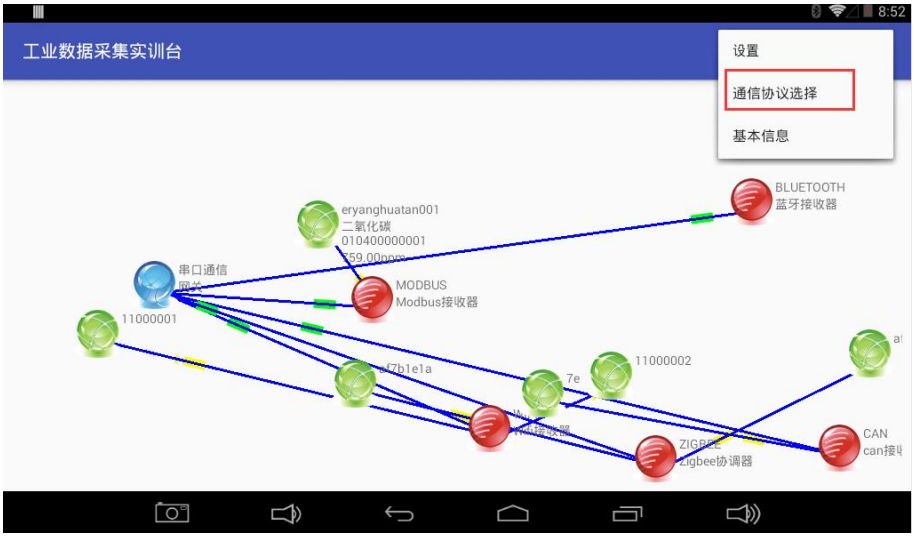


重新打开网关的“工业数据采集实训台”app，如下图标所示

进入应用程序，如下图所示

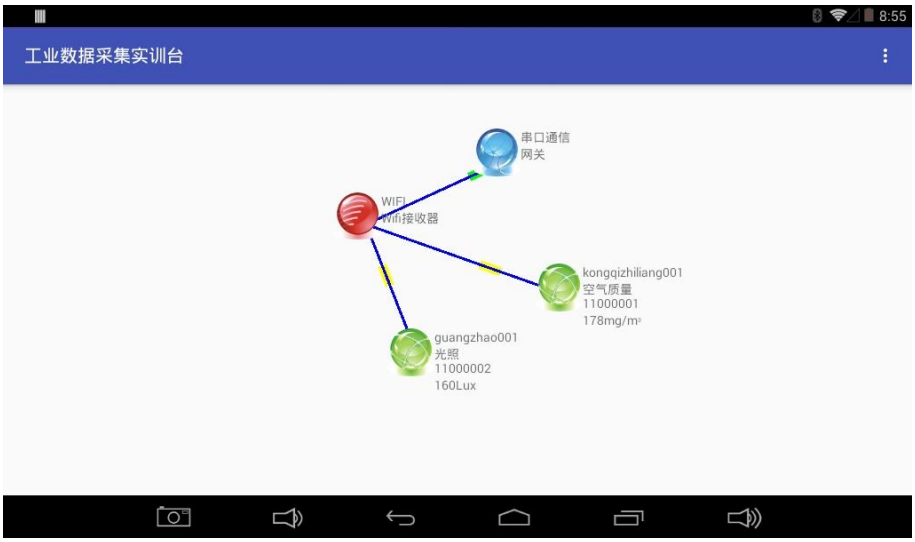


点击右上角按钮，如下图所示。

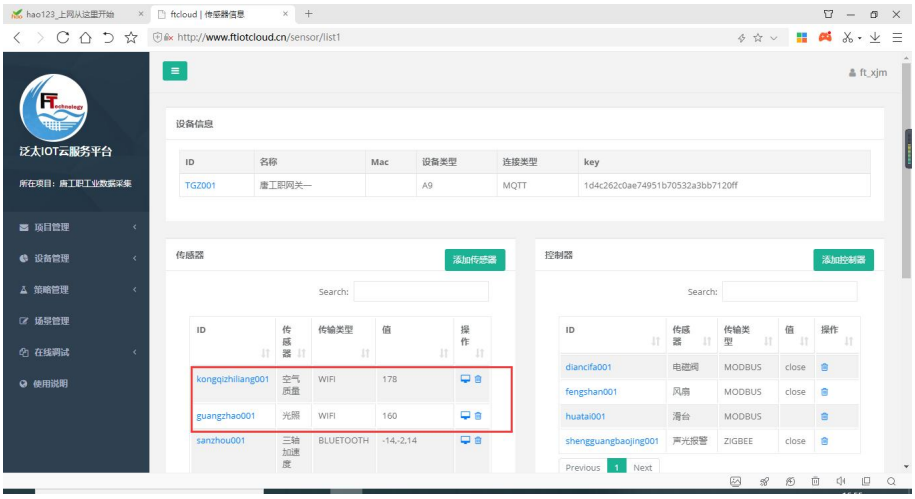


点击“通信协议选择”项，打开子页面，如下图所示

勾选 WiFi Server 选项，点击确定选择按钮，此时只显示 wifi 协议下的传感器。如下图所示 wifi 通信协议下的传感器会显示到页面上。



同时传感器的数据实时上传至云平台，如下图



4.8.8. LoRa 节点云平台接入测试

将传感器采集控制节点（远程测控终端，无需 NB 模块）管理的金属接近开关状态数据通过 LoRa 无线通信传输给 NB 网关（远程测控终端，需 NB 模块和 NB 卡），进而上传到云平台，同时将 NB 网关管理的水流量、水泵电机接入到云平台。可以在平台查看金属接近开关的状态、水流量值及远程控制电机调速。

(一) 硬件连接

传感器采集控制节点需要连接金属接近传感器，NB 网关需要连接水流量电机模块。此外，烧录程序之前需要连接仿真器，仿真器一端通过 USB 方口线连接到 PC，一端通过 20 针排线连接到下载调试板，下载调试板通过 10 针排线连接到传感器采集控制节点的烧录口。烧录完传感器采集控制节点后，将 10 针排线从传感器采集控制节点上拔下，再插到 NB 网关的烧录口。硬件连接图如下所示：

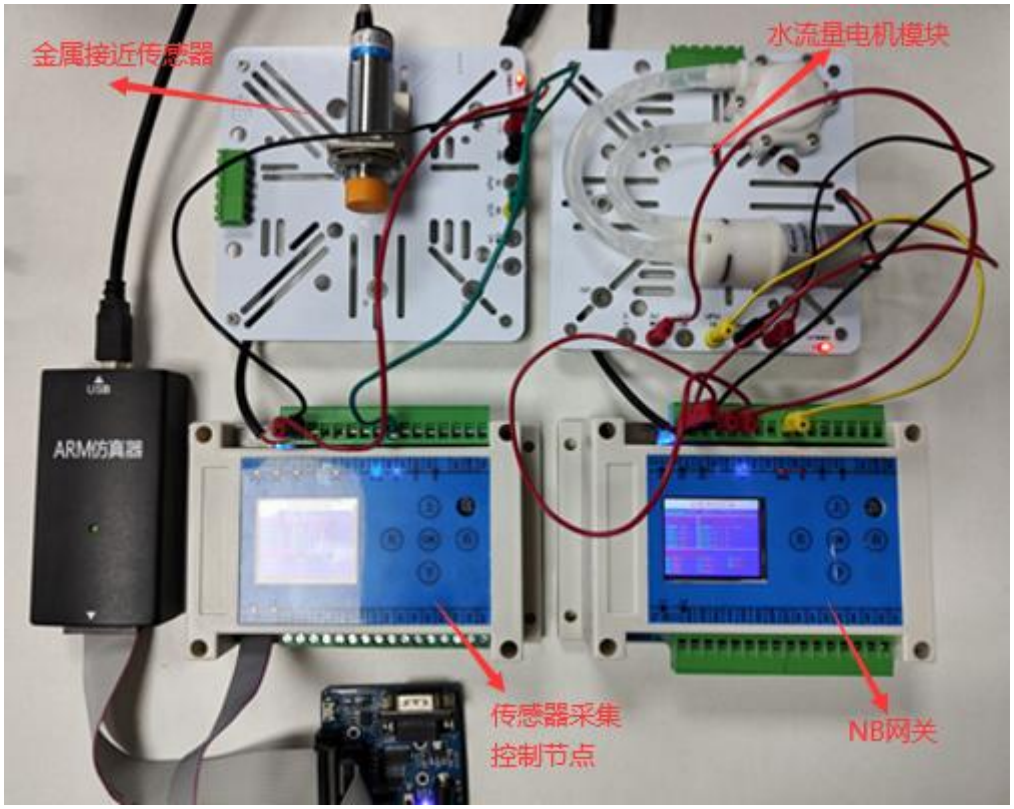


图 4-23 硬件连接图

不考虑 DC 12 V 供电和 GND 的连线（图示的 4 个设备都需要 DC 12V 供电），介绍下金属接近传感器和传感器采集控制节点的连接以及水流量电机模块和 NB

网关的连接。

金属接近传感器模块上的 OUT(NO)口是它的输出信号，连接到传感器采集控制节点的 DI0，这个口可以进行开关量检测。

水流量电机模块的 PWM(RI)口连接到 NB 网关的 DI0，这个口可以进行脉冲计数，进而计算水流量；水流量电机模块的 IN(COM)口连接到 NB 网关的 Y00，Y00 口可以进行输出调节，进而调节电机转速。不过 Y00 口的最大电压是由 Y00 左侧的 COM 口输入电压决定的，所以需要将 DC 12V 输入连接到 COM 口。

(二)　程序烧录

1. 打开传感器采集控制节点的实验例程，路径为：04-云平台接入测试\05-LoRa 感知节点平台接入测试\1.远程测控终端 -Lora 发送+金属接近\Project\RTU.uvproj，重新编译代码并将代码下载到传感器采集控制节点中，如图 4-所示。

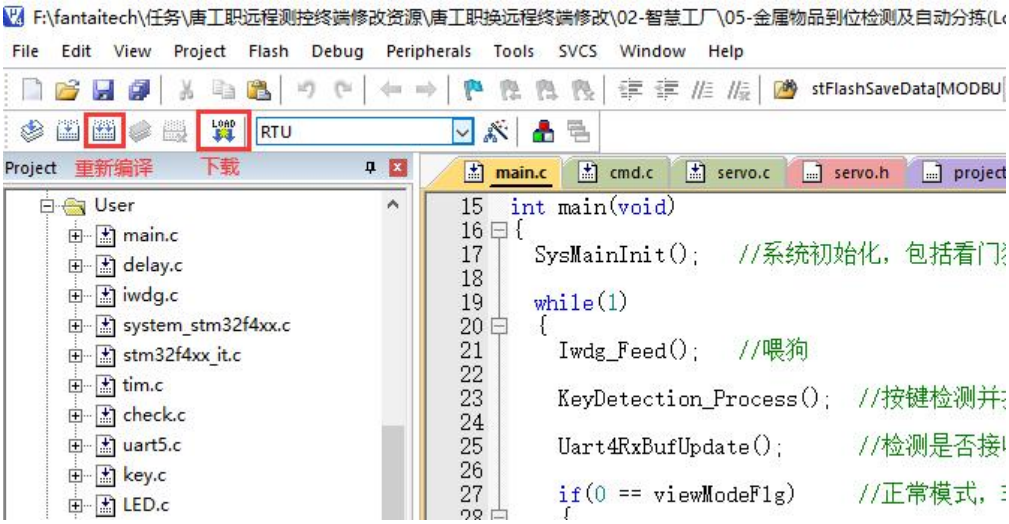


图 4-24 编译下载按钮示意图

2. 将仿真器从传感器采集控制节点上拔下，插到 NB 网关的烧录口上。

3. 打开 NB 网关的实验例程，路径为：04-云平台接入测试\05-LoRa 感知节点平台接入测试\2.远程测控终端 -Lora 接收+NB MQTT 上报+水流量电机\Project\RTU.uvproj，重新编译代码并将代码下载到 NB 网关中。

这里需要注意的是，NB 网关的 LoRa 频段必须和传感器采集控制节点的 LoRa 频段一致，可以在代码中修改和查看，此外，多组设备共同实验时，需要保证各组的 Lora 频段不同，否则可能互相干扰，使用频段间隔最好为2MHz及以上。NB网关的LoRa频段定义位置如图 4-11

所示；传感器采集控制节点的LoRa频段定义位置如图 4-12所示。（LoRa频段范围可以为410~510MHz。）

main.c | cmd.c | tim.c | project.h

```
#include "stm32f4xx.h"

//宏定义
#define NONE_WIRELESS                   0       //不使用无线通信
#define WIFI                            1
#define ZIGBEE                          2
#define LORA                            3
#define NB_IOT                          4
#define ZIGBEE_LORA                     5       //可以同时使用Zigbee和lora


//Lora
#define LORA_KHZ                        421000      //KHZ
#define LORA_FREQUENCY_BAND             ((LORA_KHZ&0xFF0000)>>16), ((LORA_KHZ&0xFF00)>>8), (LORA_KHZ&0xFF)
#define LORA_SPEED                      0x65        //速率0x01-0x70    1-112


//Zigbee
#define ZIGBEE_CSCAL                    "24"    //信道，"11"---"26"
#define ZIGBEE_CSPID                    "6F6F"  //PANID
#define ZIGBEE_CSMODE                   "1"     //0路由  1协调器
```

图 4-11 NB网关 Lora频段修改位置

main.c | cmd.c | servo.c | servo.h | project.h

```
#define LORA                            3
#define NB_IOT                          4
#define ZIGBEE_LORA                     5       //可以同时使用Zigbee和lora


//Lora
#define LORA_KHZ                        421000      //KHZ
#define LORA_FREQUENCY_BAND             ((LORA_KHZ&0xFF0000)>>16), ((LORA_KHZ&0xFF00)>>8), (LORA_KHZ&0xFF)
#define LORA_SPEED                      0x65        //速率0x01-0x70    1-112


//Zigbee
#define ZIGBEE_CSCAL                    "24"    //信道，"11"---"26"
#define ZIGBEE_CSPID                    "6F6F"  //PANID
#define ZIGBEE_CSMODE                   "1"     //0路由  1协调器
```

图 4-12 传感器采集控制节点Lora频段修改位置

(三)登录网页

1. 打开浏览器，进入网址为www.ftiotcloud.cn:3000的网页，如图 4-13所示。

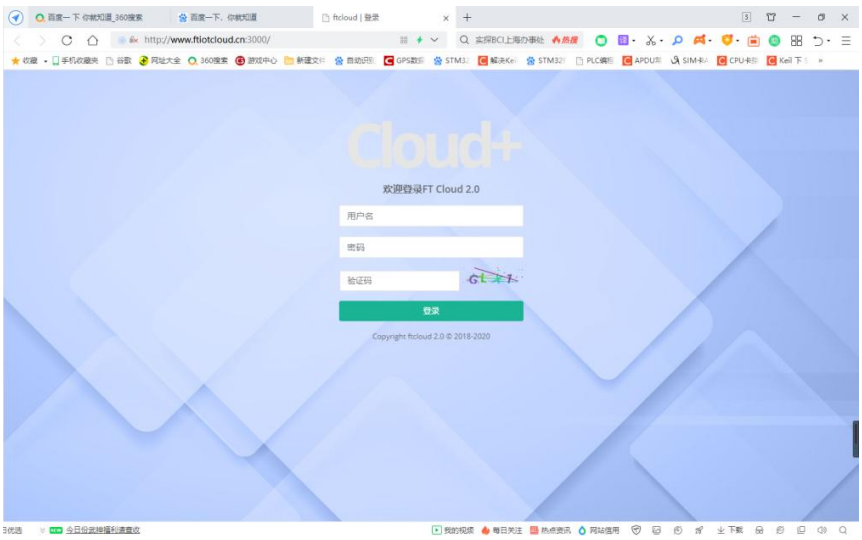


图 4-13 登录平台

2. 输入给定的用户名和密码，然后点击登录，可以进入如图 4-14 所示页面。

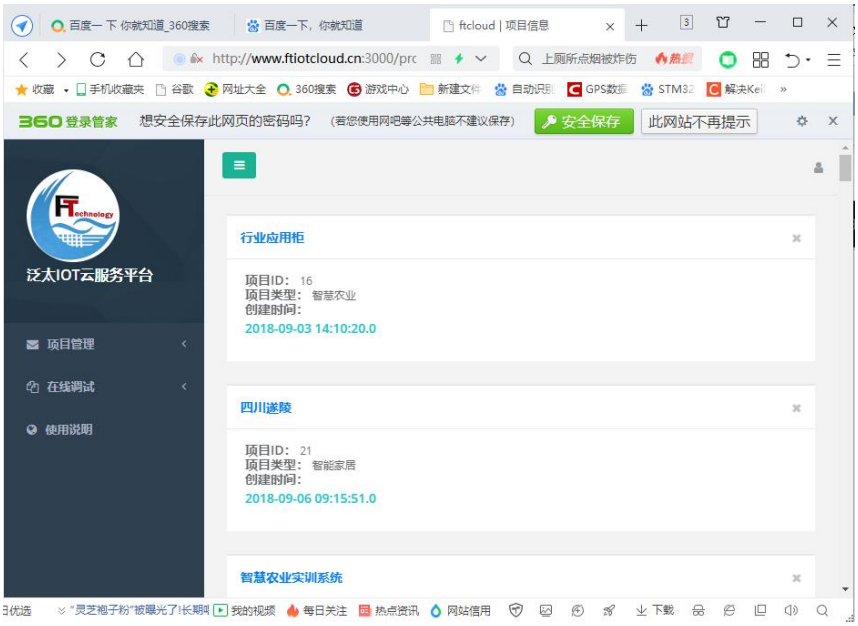


图 4-14 登录成功页面

3. 点击左侧的“项目管理”—>”项目添加”，进入如图 4-15 所示页面。

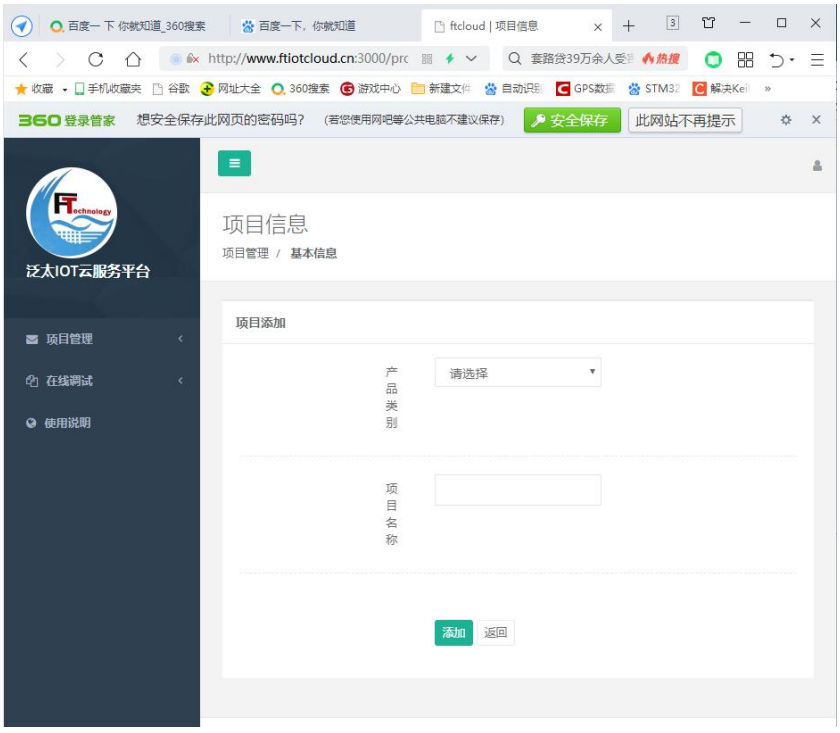


图 4-15 项目添加页面

4. 项目类型按照需要选择，比如“实验箱”、“工业物联网”等等，项目名称可以自定义，如图 4-所示。最后点击“添加”。

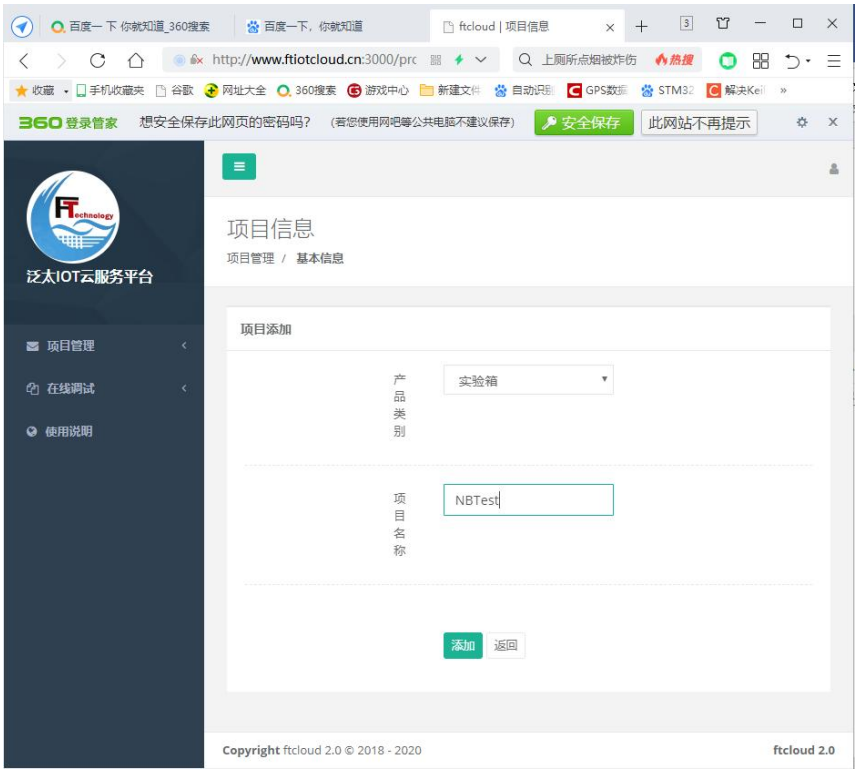


图 4-30 项目添加

5. 然后可以在项目信息中找到刚刚添加的项目“NBTest”，如图 4-所示。

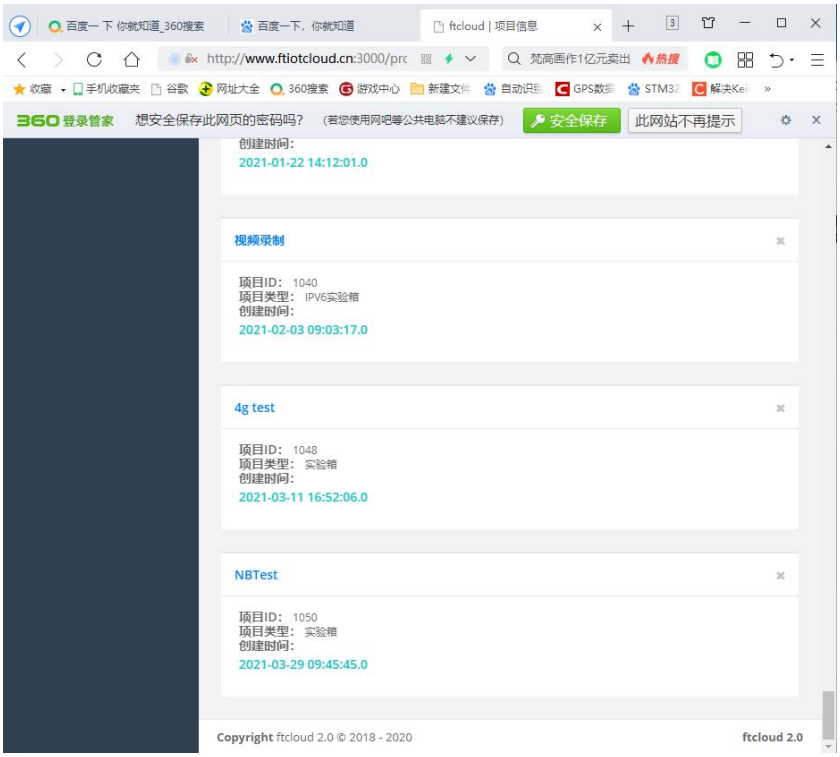


图 4-31 项目信息

6. 点击项目名称进入到如图 4-所示页面，然后再点击“添加设备”，进入到如图 4-16 所示页面，信息填写如图 4-17 所示，这里的设备 ID

是 NB 网关的 IMEI 号（NB 网关的 IMIE 号显示在 LCD 屏上）；设备名称是自定义的，如果设备 ID 号重复了平台将无法添加，设备类型选择 STM32，连接类型是 MQTT，最后点击添加。添加成功后就可以在设备信息栏看到刚刚添加的设备。如图 4-18 所示。

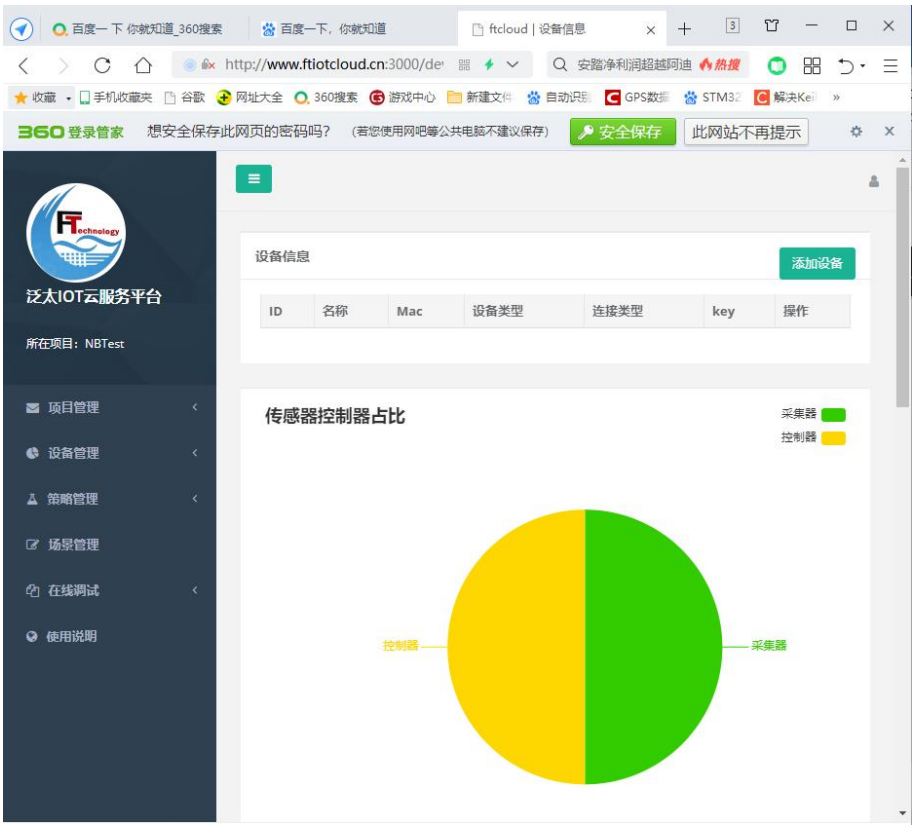


图 4-32 设备信息

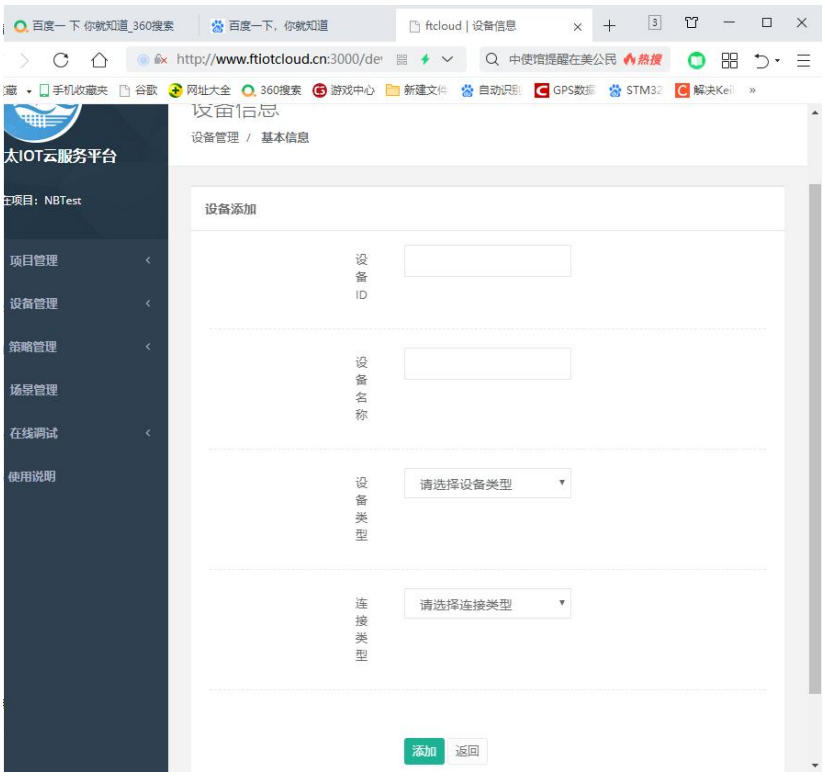


图 4-16 设备信息添加

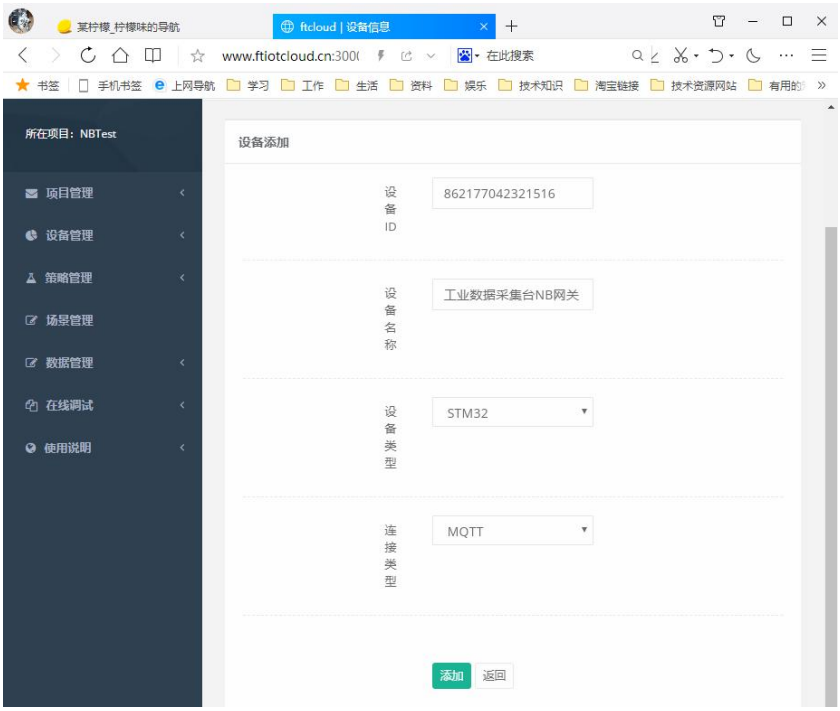


图 4-17 设备信息填写

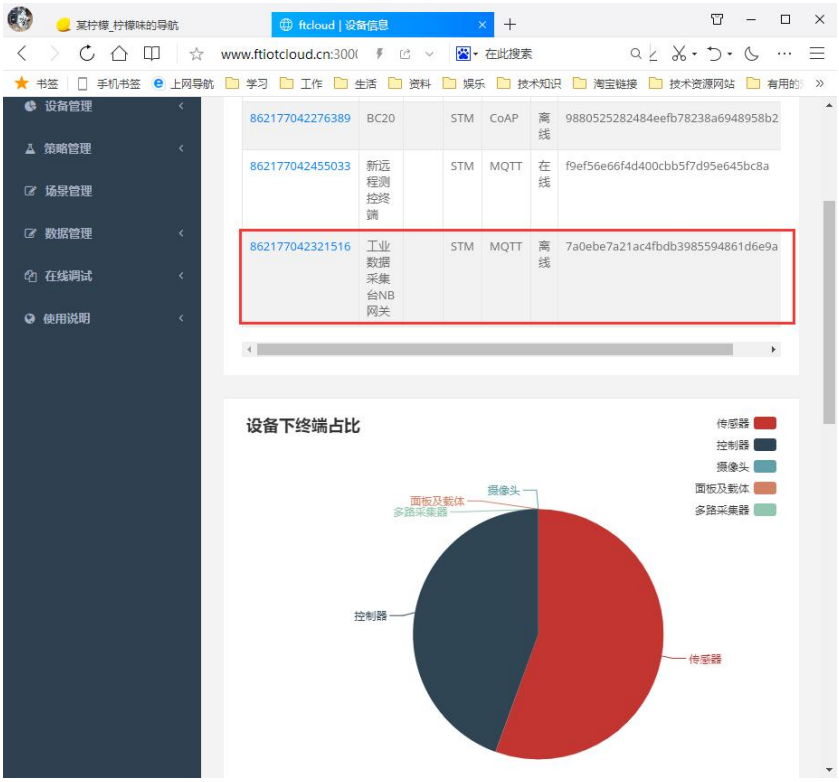


图 4-18 设备添加完成

7. 点击页面上的设备 ID 号，如图 4-19 所示。进入到设备信息页面，如图 4-20 所示。

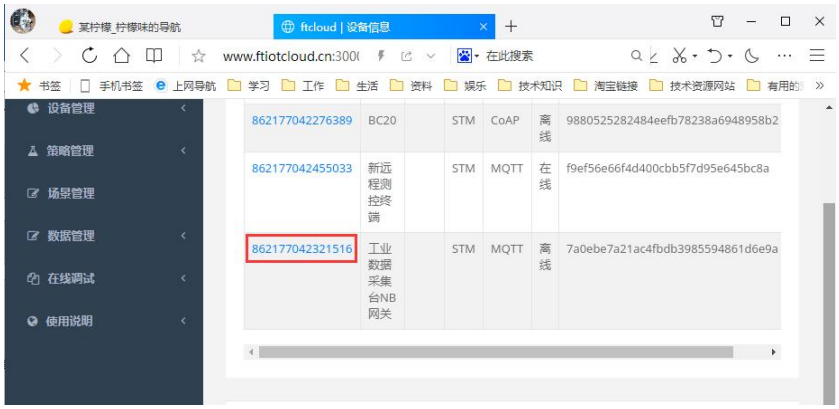


图 4-19 ID 号位置示意图

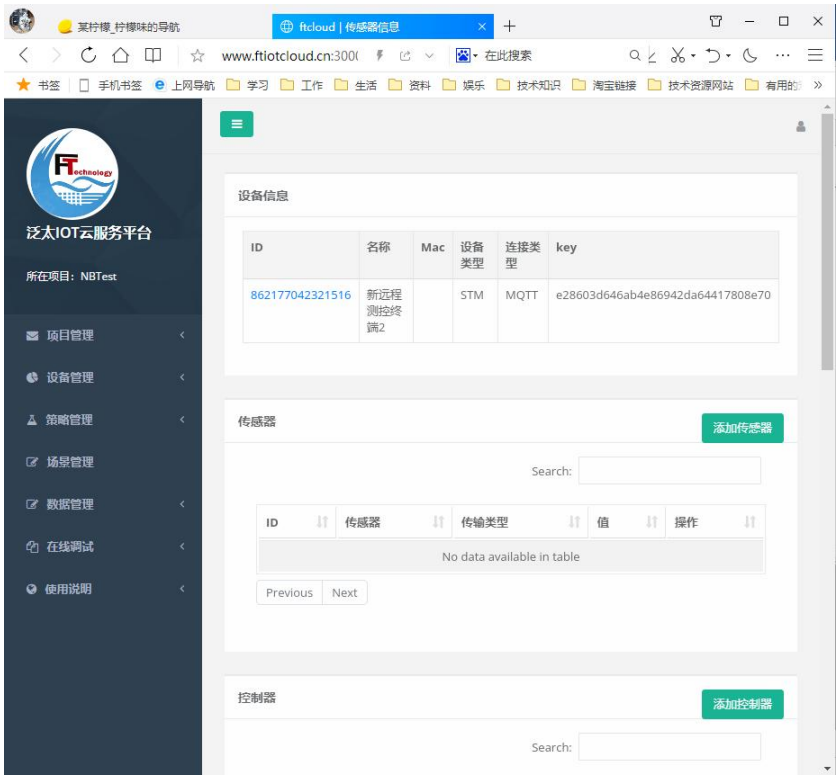


图 4-20 设备信息页面

8. 点击“添加传感器”，进入如图 4-21 所示页面。

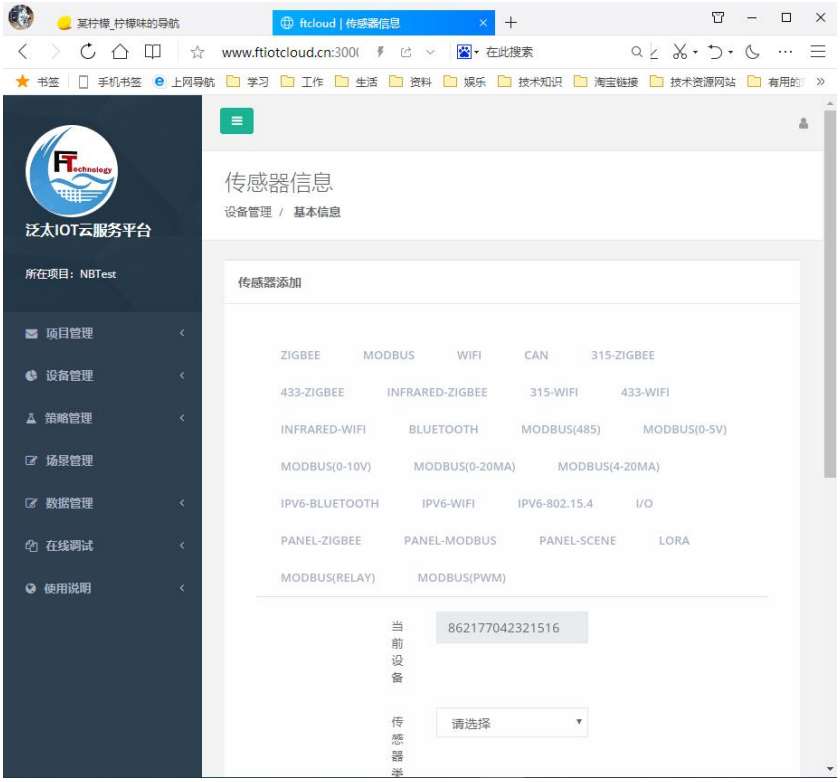


图 4-21 添加传感器页面

9. 如图 4-22 所示为添加金属接近传感器的设置，传输方式选择“LORA”

传感器类型选择“金属接近”，场景类型可以随意选择，传感器标识 ID 为：IMEI 号+传感器名称字符串（+无线通信方式），传感器 Mac 这里默认为 11111111。最后点击添加。

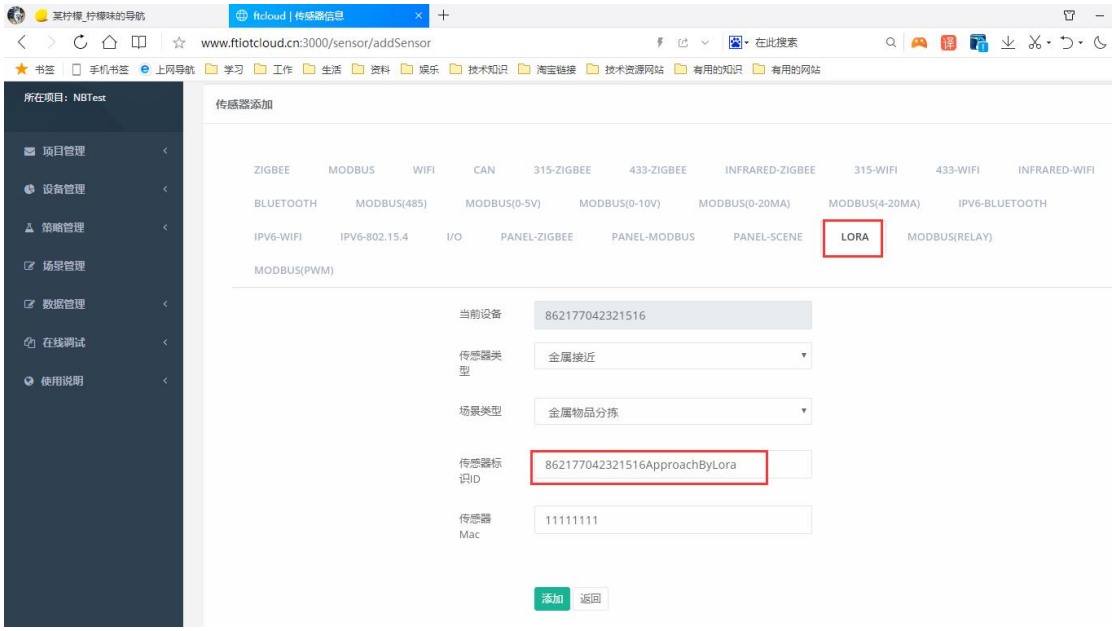


图 4-22 添加金属接近传感器

10. 用相同的方式添加水流量传感器，传输方式选择“I/O”，如图 4-所示。

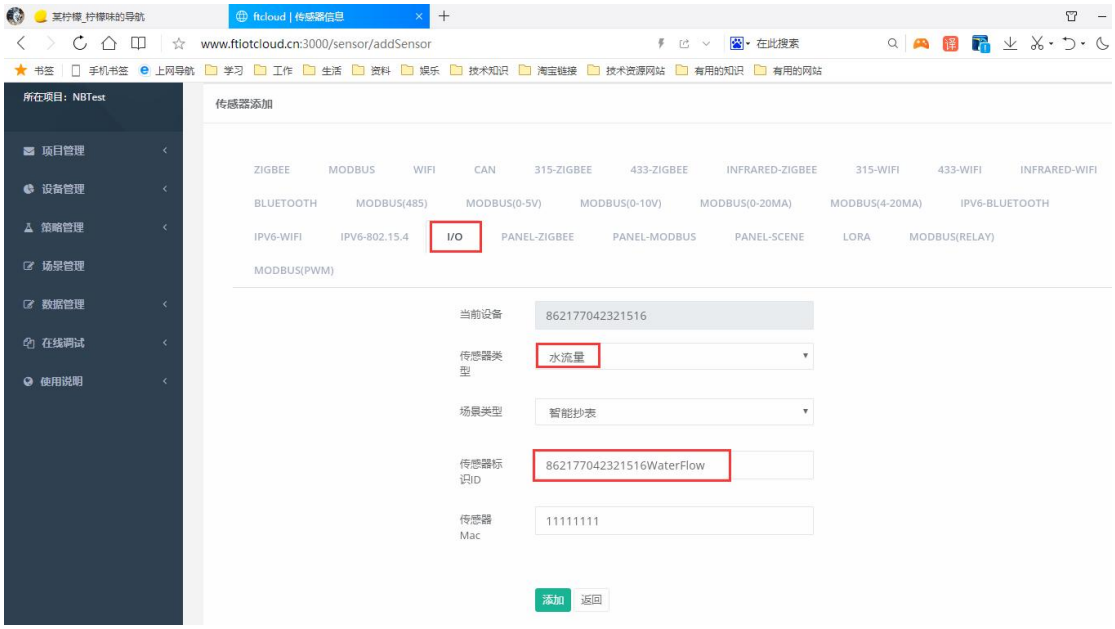


图 4-40 添加水流量传感器

11. 在如图 4-20 所示设备信息页面点击“添加控制器”添加调速电机，如图 4-所示。

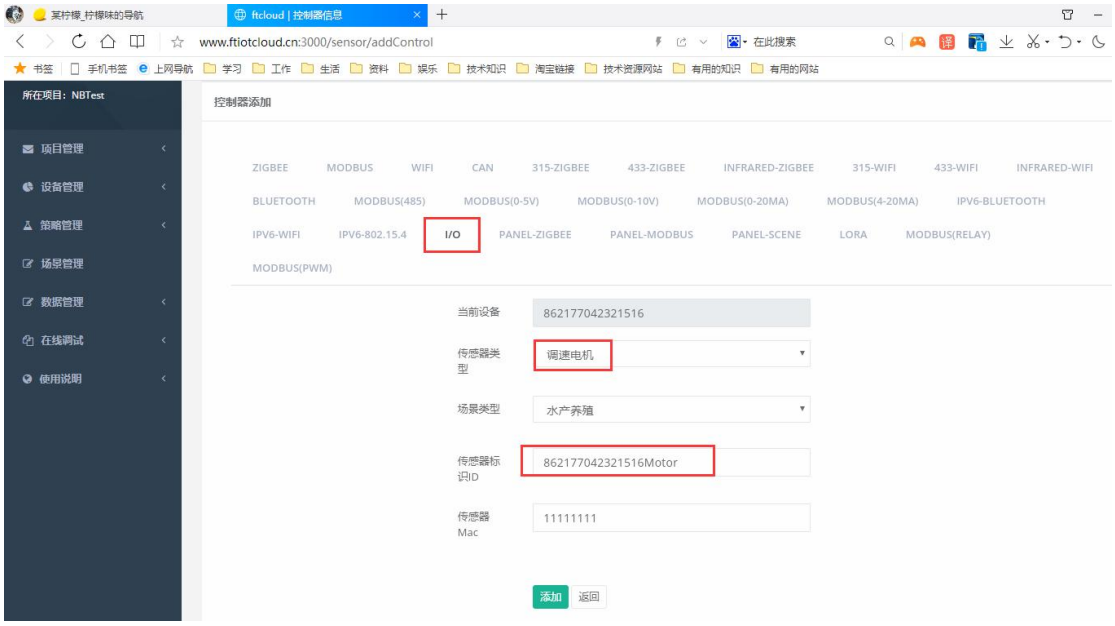


图 4-41 添加调速电机

12. 在设备信息页面就可以看到添加的传感器，如图 4-所示。

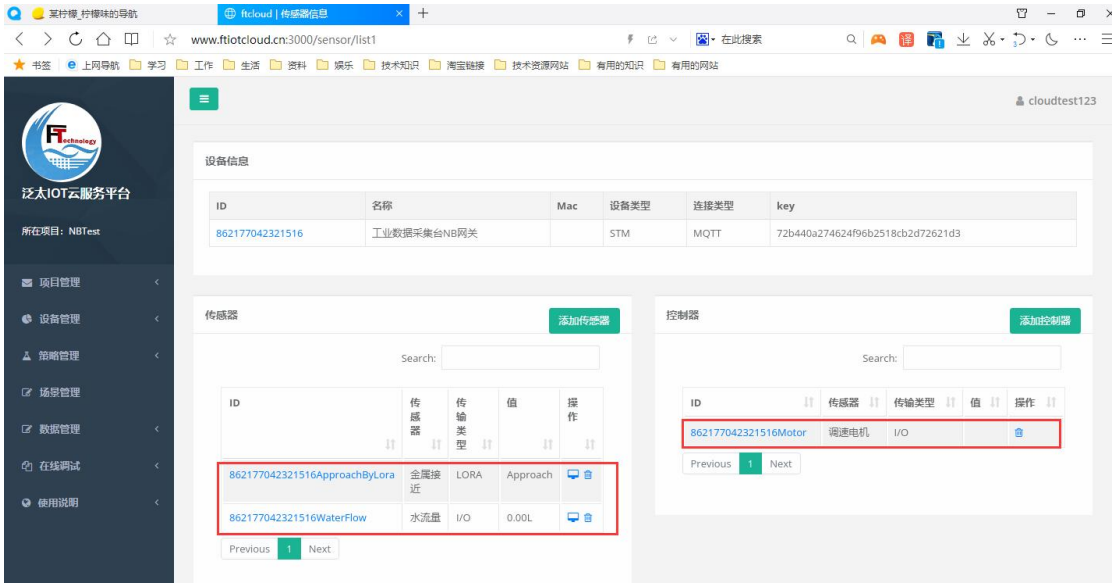


图 4-42 添加传感器列表

13. 点击传感器的 ID 号就可以进入对应的传感器页面，如图 4-23 所示为金属接近传感器页面。使用金属靠近金属接近传感器，平台数据显示如图 4-24 所示。

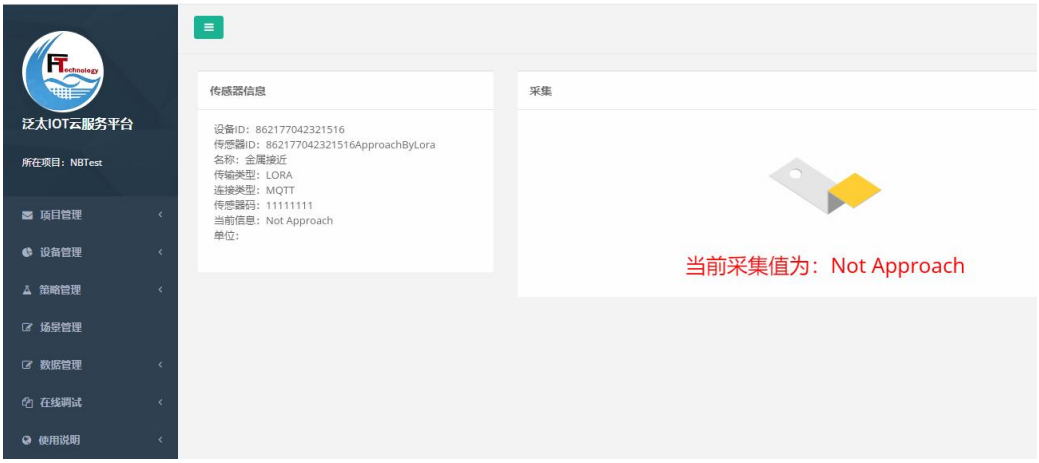


图 4-23 金属接近传感器页面

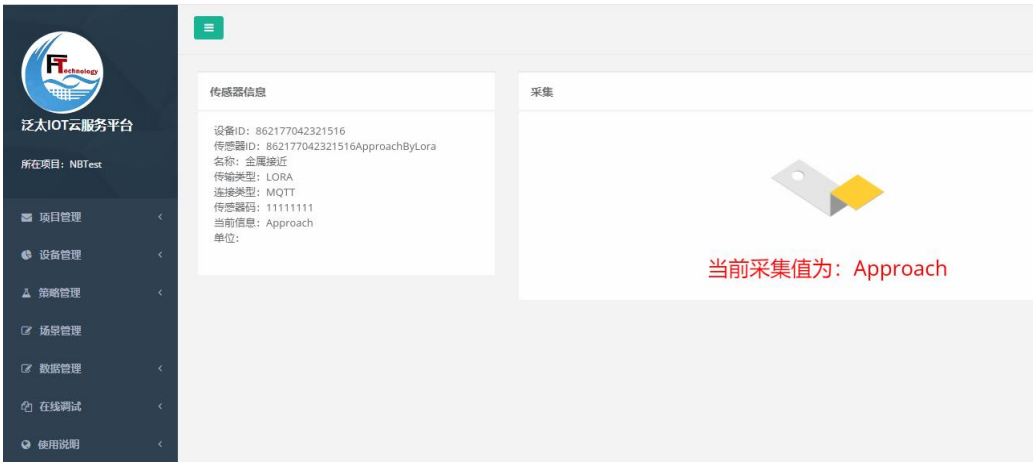


图 4-24 有金属靠近接近开关

14. 如图 4-23 所示为水流量传感器页面，短按 NB 网关的右键可以增大电机转速，短按左键可以减小电机转速，长按左键电机速度为 0，长按右键电机为最大转速。长按右键让电机以最大速度转动，观察平台水流量的数据，可以看到平台数据如图 4-26 所示，一段时间后，长按左键让电机停止转动。

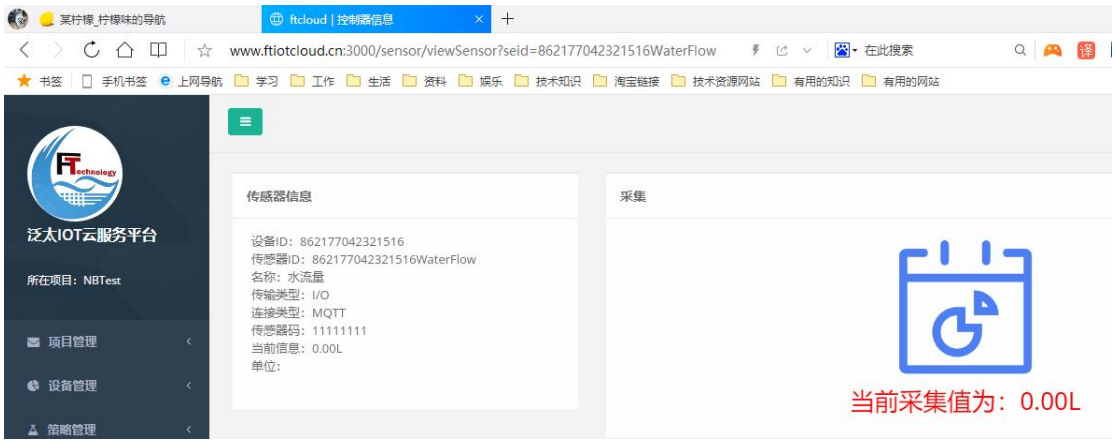


图 4-25 水流量传感器显示页面

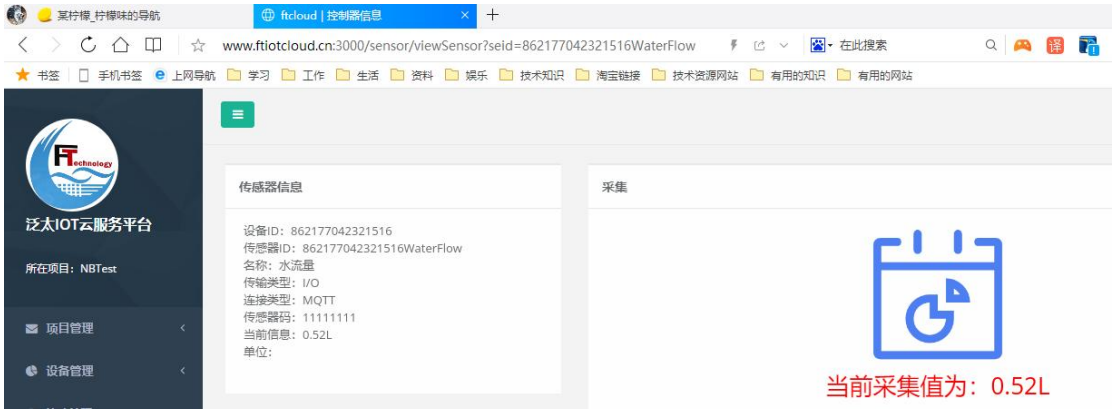


图 4-26 水流量上传平台显示

15. 如图 4-27 所示为调速电机页面，在输入框中输入 Y00 输出的值（0-100，0 停止，100，最大转速），如图 4-28 所示，再点击“F”，调速电机的速度被改变。

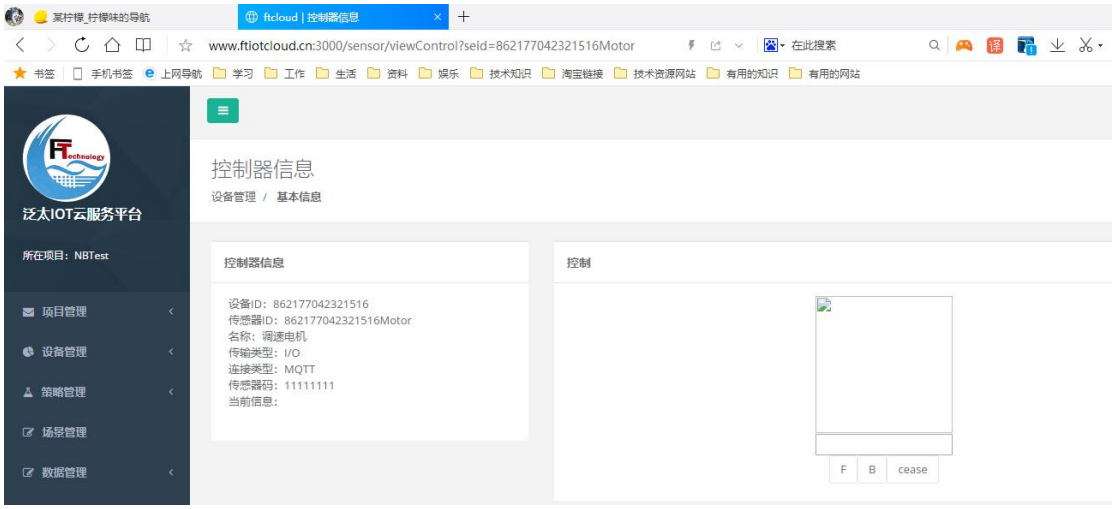


图 4-27 调速电机显示页面

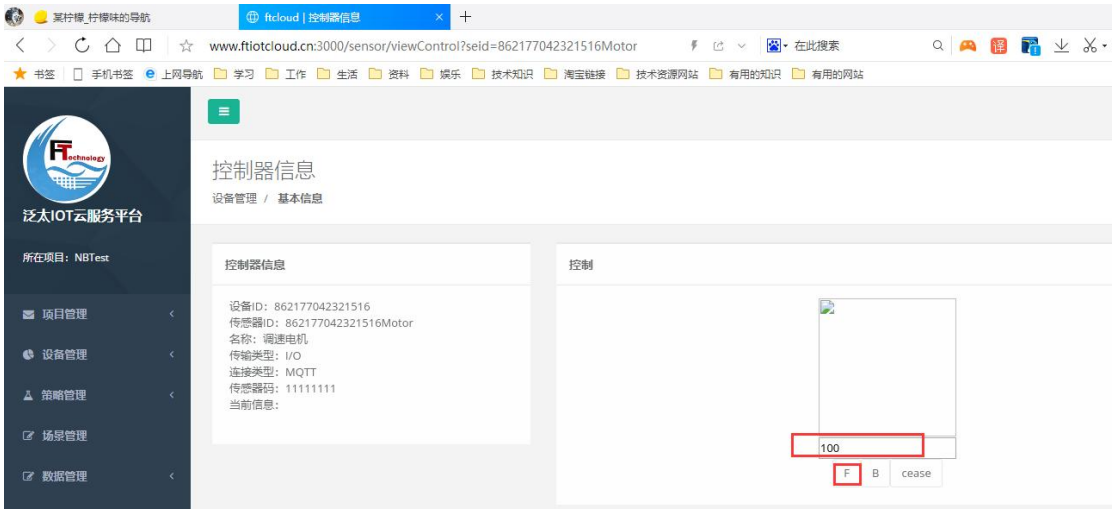


图 4-28 调节调速电机的速度

5. 附录一、有线终端接线图