

## 摘要

当前世界，信息科技飞速发展，传统的单片机控制硬件电路已经不能满足社会发展的需要。在互联网的大潮下，传感器接入互联网已经成为现实，物联网的发展正在如火如荼的进行。在我的毕业设计中，将依赖于 MQTT 协议进行一个简单物联网系统的实践，接入传感器，监控传感器数据，并对相关后续设备进行控制。通过对 Android 平台、云服务器以及树莓派的整合，探索一个开源、可靠、简洁的物联网系统，为后续的研究和使用奠定基础。通过本文，尽量将搭建开发过程中的细节展示出来，希望后续的同学通过阅读本文，能够按照文章描述一步步构建自己的物联网系统，并发散思维，能够继续改进和创新，打造属于自己的物联网系统。

**关键词：**物联网；MQTT；传感器；Android；树莓派

## Abstract

The current world, with the rapid development of information technology, the traditional single-chip hardware control circuit can not meet the needs of social development. In the tide of the Internet, a sensor connected to the Internet has become a reality, the development of things is in full swing. I graduated from the design, the protocol will depend on MQTT to practice a simple system of things, the access sensor to monitor sensor data, and related follow-up control over the device. Through the integration of the Android platform, cloud server and Raspberry Pi explore an open source, reliable, simple networking system, lay the foundation for further research and use. In this article, try to set up the development process of the details of the show came out, hoping to follow the students through reading this article, in accordance with article describes a step by step to build their own system of things, and divergent thinking, to continue to improve and innovate, to create their own material networking system.

**Keywords:** IOT;MQTT;Sensor;Android;Raspi

# Contents

<b>摘要</b> .....	I
<b>Abstract</b> .....	II
<b>第一章 系统构想</b> .....	1
1.1 什么是物联网 .....	1
1.2 整体思路构想 .....	1
<b>第二章 场景、需求与方案</b> .....	2
2.1 具体场景 .....	2
2.2 需求分析 .....	2
2.3 方案制定 .....	2
2.4 整体设计图示 .....	3
<b>第三章 准备工作</b> .....	8
3.1 硬件准备 .....	8
3.2 软件准备 .....	8
3.3 开发前的语言和开发框架准备 .....	8
<b>第四章 系统设计</b> .....	10
4.1 MQTT 协议 v3.1.1 .....	10
4.2 通信接口 .....	12
4.2.1 温度信息传输接口 .....	12
4.2.2 继电器控制信息传输接口 .....	12
4.2.3 继电器开关状态接口 .....	12
4.2.4 温度低于 24 摄氏度时和温度高于 30 摄氏度时的通知信息 .....	13
<b>第五章 系统实现</b> .....	14
5.1 硬件电路搭建 .....	14
5.1.1 温度采集功能的电路搭建 .....	14
5.1.2 继电器模块的电路搭建 .....	15
5.1.3 整体接线图和整体原理图 .....	17
5.2 代码编写思路 .....	20

5.3 服务端代码编写 .....	20
5.3.1 环境搭建 .....	21
5.3.2 代码编写 .....	25
5.4 树莓派端的代码编写 .....	27
5.4.1 环境搭建 .....	27
5.4.2 树莓派端的代码编写 .....	35
5.4.3 与服务端交互的代码编写 .....	41
5.4.4 协议实现 .....	44
5.5 Android 端代码的编写 .....	45
5.5.1 引入 Paho 框架的 jar 包 .....	45
5.5.2 界面布局的编写 .....	47
5.5.3 业务逻辑编写 .....	49
<b>第六章 结论 .....</b>	<b>53</b>
<b>第七章 参考文献 .....</b>	<b>54</b>
<b>第八章 致谢及声明 .....</b>	<b>56</b>

# 第一章 系统构想

## 1.1 什么是物联网

从一个简单的使用场景引入，就是把家里的灯泡、空调、彩电、洗衣机等等接入互联网，来实现远程控制。而维基百科是这样定义的

物联网（英语：Internet of Things，缩写 IoT）是互联网、传统电信网等信息承载体，让所有能行使独立功能的普通物体实现互联互通的网络。

再大白话一点，物联网就是随时随地通过网络去监控和控制“远在天边”的传感器，就像你回家打开灯泡需要按开关是一样的，只不过这个开关转到你的“掌上终端”——手机上了，而且你也不是走过去按开关，而是远程“按”！严格来说，物联网将最底层的传感器和控制器接入到网络当中，使它们可以通过互联网被远程操控。物联网是一个中介，它构建了人和物体、物体和物体之间的通路。

## 1.2 整体思路构想

本文通过物联网的实践，将自己大学四年中所学习的内容进行梳理总结，结合 android 终端 app 开发、服务端开发和树莓派 +Arduino 的组合开发的内容，来构建自己的物联网系统。主要实现的内容是监测和控制，监测方面简单来说就是获取传感器数据，复杂点讲就是对获取的数据进行分析，制定相应的策略，来应对异常变化下的后续控制；控制方面简单来说就是类似于打开开关的操作，控制不仅仅是单独控制，还要添加到控制策略中，来应对环境变化。这两点是这次毕业论文中的核心，我将围绕这两点来设计整个系统，力求完整和简洁。希望你看了这篇论文后，可以根据论文上所阐述的内容，做出你自己的物联网系统来！

## 第二章 场景、需求与方案

### 2.1 具体场景

假设一个场景，夏天，室外 36 摄氏度高温，人走在回到家途中。但是你想看一下家中的温度如何，来确定是否开启空调降低温度。那么会有这样一个需求：得知家中温度太高，比如室内温度 32 摄氏度，想先远程打开空调，让它把室内温度降低，这样到家后不用受高温折磨；但是如果室温是 20 摄氏度，不需要远程打开空调，室内温度相对来说情况很好。如果室温超过 30 摄氏度，需要进行降温，降到 24 摄氏度时，停止空调运转，这样达到人体的舒适温度了，没有必要在继续开启。但是人的感觉是不断变化的，室温降到 24 摄氏度了，还是觉得不够，再开一会儿空调吧，过段时间再主动去关闭它。

那么这就是一个物联网中完整的使用场景，有监测（温度的数值变化）、有控制（打开空调）、有控制策略（温度变化到某个值时空调进行开启与关闭的动作）。这个就是当前所面对的使用场景，下面针对这个场景，进行需求分析。

### 2.2 需求分析

根据上面的场景，进行需求分析，在这里先列出来：

- 1) 监测室内温度，并发送到终端上
- 2) 控制空调的开关状态，发送控制动作完成后的空调状态到终端

需求很简单，就是这两条通路——监测和控制，但是需要制定控制策略，如下：

如果空调处于关闭状态，当室内温度高于 30 摄氏度时，询问用户是否开启空调；如果空调处于开启状态，当室内温度低于 24 摄氏度时，询问用户是否关闭空调，如果用户选择不关闭，会每 10 分钟提醒一次，直到关闭操作正确执行为止。

这里假定，空调接入物联网系统的时候，只对空调实现简单的开关功能。而且将开关操作的权利交给用户，完全由用户来执行。

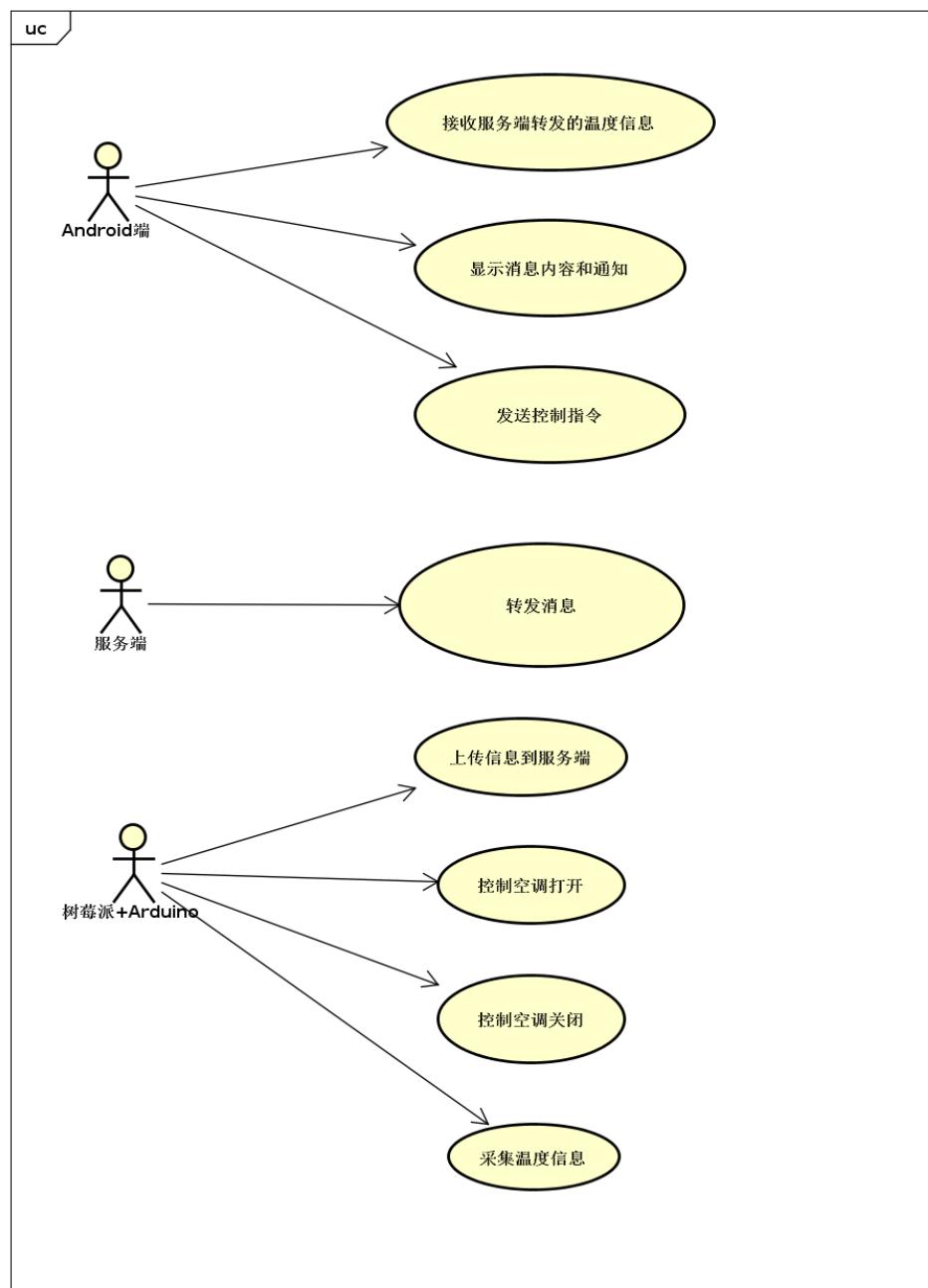
### 2.3 方案制定

这里需要规划三个部分的内容，分为 Android 端 app 开发，服务端开发以及树莓派 +Arduino 控制传感器的开发。树莓派 +Arduino 的开发又划分为程序编写和硬件电路搭建。app 实现的功能是接收温度数据并展示，以及控制指令的发送；服务端实现的功能

是消息转发；树莓派和 Arduino 实现的功能是数据上传，以及接收控制指令并执行（打开或关闭开关）。这里需要解释一下为什么使用树莓派和 Arduino 配合控制硬件电路。树莓派承担的角色相当于一个路由器的功能，对外连接互联网，进行数据的发送和接收，并将控制指令发送到 Arduino 上来执行；Arduino 承担的是一个面向传感器的角色，由于树莓派的 IO 口比较弱，虽然引脚较多但是性能不佳而且容易损坏，而 Arduino 本身的定位就是单片机，其 IO 口性能强，不易损坏，可以满足需求，所以接入 Arduino 来代替树莓派上的 IO 口，将 Arduino 作为树莓派的接口板使用。

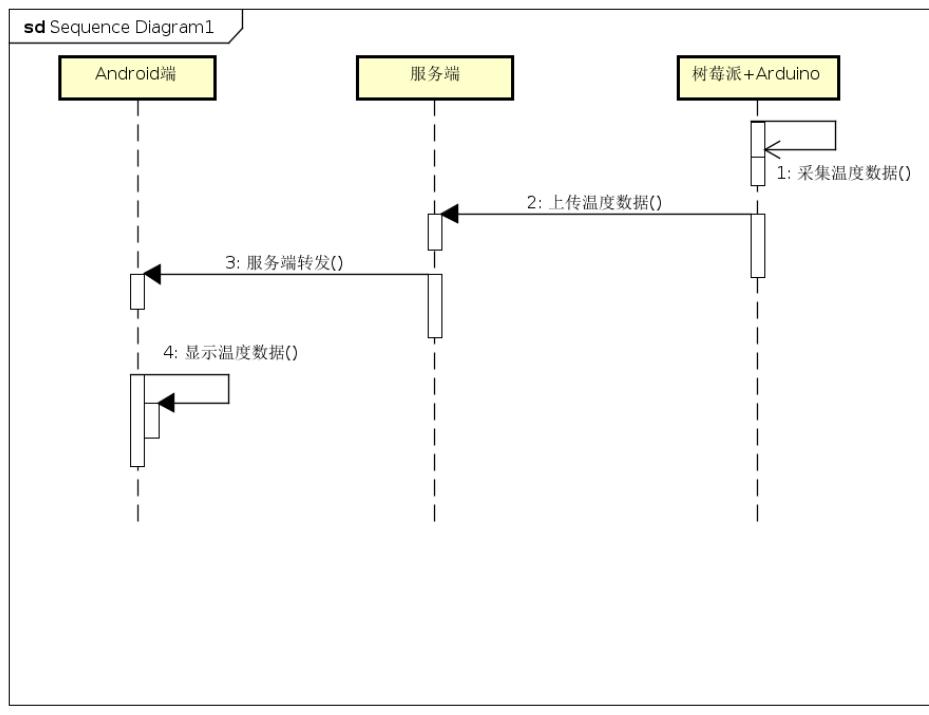
## 2.4 整体设计图示

下面通过 UML 图示来详细展示整个系统中的角色划分以及控制流程。图2-1展示了整个操作环境中，所有的角色以及角色职能；图2-2展示了温度数据采集的流程；图2-3展示了继电器控制流程；图2-4和图2-5分别展示了温度过高开启空调的操作流程，以及温度过低关闭空调的操作流程。



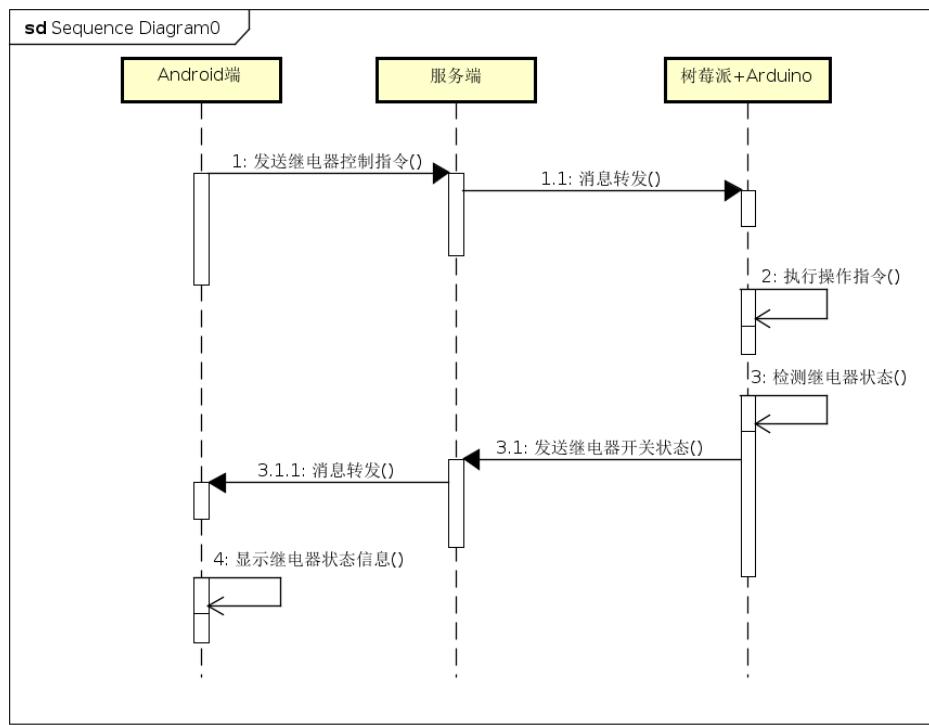
powered by Astah

Figure 2-1 用例图分析



powered by Astah

Figure 2-2 温度数据采集流程



powered by Astah

Figure 2-3 继电器控制流程

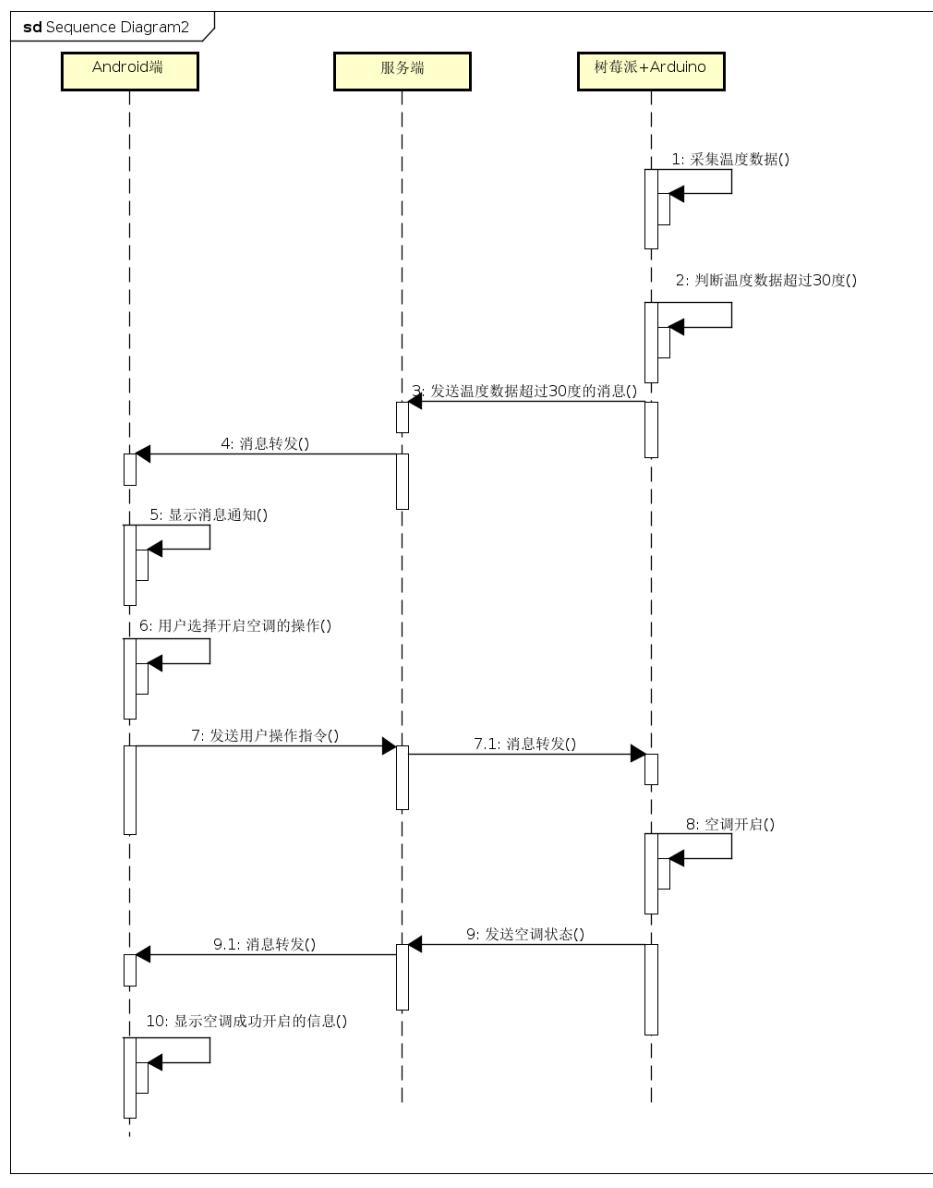


Figure 2-4 温度过高开启空调的操作

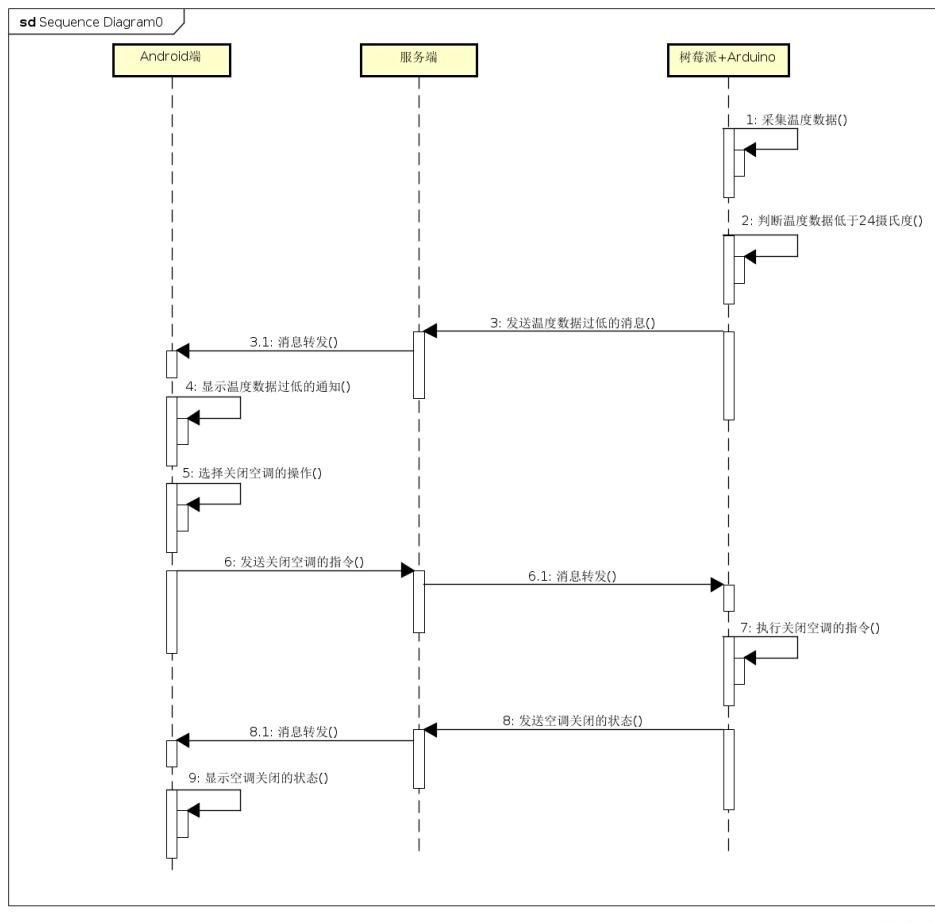


Figure 2-5 温度过低关闭空调的流程

## 第三章 准备工作

### 3.1 硬件准备

- 1) 树莓派 2 代 B 型开发板一块，SD 卡（做系统用，需要 class10 标准的，8GB 起）一张，读卡器一个，以及树莓派官方屏幕，免驱无线网卡和 5V 2A 电源一个
- 2) Arduino UNO R3 一块以及一根 Arduino leonardo 线
- 3) DHT11 温湿度采集模块，三线制或者四线制
- 4) 四路继电器模块
- 5) 发光二极管若干，使用红绿蓝等多种颜色的，便于区分不同功能
- 6) 杜邦线，公对公母对母的都需要，每一个类型至少准备 30 根
- 7) Android 终端一台，我这里用的我的平板来模拟，系统版本为 Android6.0，当然应用程序最低兼容到 Android4.0 版本的系统

### 3.2 软件准备

- 1) Android App 开发：使用 Android Studio 2.0 开发，App 最低版本兼容到 Android 4.0
- 2) 服务端开发，使用阿里云服务器，镜像为 Ubuntu14.04 的系统，通过 ssh 连接服务器进行操作，直接在服务端使用 vim 编写程序
- 3) 树莓派，使用 Raspbian jessie 镜像作为系统，该镜像基于 debian8，同样通过 ssh 连接树莓派进行操作，直接在树莓派端使用 vim 来编写程序
- 4) 电路图绘制，使用 fritzing-0.9.2b，跨平台的电路图绘制工具，对 Arduino 的相关操作友好，免安装

### 3.3 开发前的语言和开发框架准备

- 1) Android 端：使用 paho 框架中的 Android Service Client，版本为 1.0.2，使用 java 语言编写，解析 json 格式的数据时借助 Gson 框架来进行

- 2) 服务端：使用 hbmqtt 框架，python 语言编写
- 3) 树莓派 +Arduino 组合：UUGear 框架，包括接口板程序和树莓派上的编程，使用 python 语言编写

如果在这一小节了解到的不多，可以先放过，后续会一步一步导入所用到的东西，写在这里需要有个提醒，首先要学习 java 语言的基本语法以及 python 语言的基本语法。

## 第四章 系统设计

### 4.1 MQTT 协议 v3.1.1

MQTT 协议是传输过程中的核心，需要依赖这个协议实现通信接口的编写。在这里不打算讨论协议的详细实现，说一下需要知道的基本内容，而这些内容是后续制定通信协议的基础。

那么为什么采用这个协议？一句话，简单！

MQTT 是一个客户端服务端架构的发布/订阅模式的消息传输协议。它的设计思想是开源、可靠、轻巧、简单，MQTT 的传输格式非常精小，最小的数据包只有 2 个比特，且无应用消息头。MQTT 可以保证消息的可靠性，它包括三种不同的服务质量（最多只传一次、最少被传一次、一次且只传一次），如果客户端意外掉线，可以使用“遗愿”发布一条消息，同时支持持久订阅。MQTT 在物联网以及移动应用中的优势有：

- 1) 可靠传输。MQTT 可以保证消息可靠安全的传输，并可以与企业应用简易集成。
- 2) 消息推送。支持消息实时通知、丰富的推送内容、灵活的 Pub-Sub 以及消息存储和过滤。
- 3) 低带宽、低耗能、低成本。占用移动应用程序带宽小，并且带宽利用率高，耗电量较少。

MQTT 协议不依附于平台存在，无论使用 C 语言还是 python 语言编程，面向 Android 平台还是 Linux 平台，都可以。

上面是一个简单的介绍，下面是真正需要的东西——控制报文类型。MQTT 协议一共有 14 种控制报文类型，如图4-1所示。

这 14 种控制报文类型都用得着，在制定通信接口的时候标示该接口使用的消息类型。

除了消息类型，这里还有三种消息发布服务质量 Qos，分别是：

- 1) “至多一次” (Qos=0)，消息发布完全依赖底层 TCP/IP 网络。会发生消息丢失或重复。这一级别可用于如下情况，环境传感器数据（比如温度、湿度数据），丢失一次读记录无所谓，因为不久后还会有第二次发送。

表格 2.1 -控制报文的类型

名字	值	报文流动方向	描述
Reserved	0	禁止	保留
CONNECT	1	客户端到服务端	客户端请求连接服务端
CONNACK	2	服务端到客户端	连接报文确认
PUBLISH	3	两个方向都允许	发布消息
PUBACK	4	两个方向都允许	QoS 1消息发布收到确认
PUBREC	5	两个方向都允许	发布收到 (保证交付第一步)
PUBREL	6	两个方向都允许	发布释放 (保证交付第二步)
PUBCOMP	7	两个方向都允许	QoS 2消息发布完成 (保证交互第三步)
SUBSCRIBE	8	客户端到服务端	客户端订阅请求
SUBACK	9	服务端到客户端	订阅请求报文确认
UNSUBSCRIBE	10	客户端到服务端	客户端取消订阅请求
UNSUBACK	11	服务端到客户端	取消订阅报文确认
PINGREQ	12	客户端到服务端	心跳请求
PINGRESP	13	服务端到客户端	心跳响应
DISCONNECT	14	客户端到服务端	客户端断开连接
Reserved	15	禁止	保留

Figure 4-1 MQTT 协议控制报文类型

- 2) “至少一次” (Qos=1), 确保消息到达, 但消息重复可能会发生, 可以应用在确保消息到达而对精确度要求不高的场景。
- 3) “只有一次” (Qos=2), 确保消息到达一次。这一级别可用于网吧计费系统一类的精确系统中, 消息重复或丢失会导致不正确的结果。

结合这两项内容, 这样就可以灵活的制定通信接口。

那么 MQTT 协议是发布/订阅模式的消息传输协议, 这两种模式有自己特定的内容要求, 下面看一下具体的要求:

- 1) 发布 (Publish) 模式, 需要确定三项内容, 第一是标题 (Title), 通过标题来区分各项消息; 第二是消息内容 (Message), 这时真正需要获取的内容, 根据消息内容进行后续操作; 第三是服务质量 (Qos), 应用在不同场景下。这三项内容是明确需要提交到服务端的, 缺一不可。
- 2) 订阅 (Subscribe) 模式, 只需要确定两项内容, 第一是标题 (Title), 第二是服务质量 (Qos)。

了解这些内容后，下面开始制定通信接口。

## 4.2 通信接口

在通信接口中，需要指明发布者和订阅者的角色，然后制定标题和服务质量，最后是消息内容，消息内容使用 json 格式数据构建。然后给出实例，包含成功情况下的消息内容，以及失败情况下的消息内容。

### 4.2.1 温度信息传输接口

发布者 (Publisher): 树莓派 +Arduino

订阅者 (Subscriber): Android 端

标题 (Title): temperature\_get

服务质量 (Qos): 0

消息内容 (Message):

1) 成功: {"status": 0, "temp\_get": "25"}(单位: 摄氏度)

2) 失败: {"status": 1, "temp\_get": "null"}

说明: 温度信息的获取，通信质量要求不高，即使丢失，只要后续采集的温度信息能够顺利到达就可以。json 数据中，temp\_get 对应的后边的值是 DHT11 采集的温度数据。

### 4.2.2 继电器控制信息传输接口

发布者 (Publisher): Android 端

订阅者 (Subscriber): 树莓派 +Arduino

标题 (Title): switch

服务质量 (Qos): 2

消息内容 (Message):

{"status": 0, "port": 15, "isOn": "1"} (继电器打开) 或者 {"status": 0, "isOn": "0"}

(继电器关闭)

说明: 继电器的控制，有较高的要求，需要保证通信质量，确保信息能发送到树莓派上。参数 port 代表 Arduino 上的引脚编号，参数 isOn 代表继电器的开关状态，“1”表示继电器打开，”0”表示继电器关闭。

### 4.2.3 继电器开关状态接口

发布者 (Publisher): 树莓派 +Arduino

订阅者 (Subscriber): Android 端

标题 (Title): switch\_status

服务质量 (Qos): 2

消息内容 (Message):

- 1) 成功: {"status": 0, "port": 15, "status\_isOn": "1"} (继电器打开) 或者 {"status": 0, "status\_isOn": "0"} (继电器关闭)
- 2) 失败: {"status": 1, "port": 15, "status\_isOn": "null"}

说明: 继电器状态的传输, 也是需要较高的通信质量, 便于 Android 终端上继电器信息的更新。参数 port 代表 Arduino 上的引脚编号, 参数 status\_isOn 代表继电器的开关状态, “1” 表示继电器打开, ”0” 表示继电器关闭。

#### 4.2.4 温度低于 24 摄氏度时和温度高于 30 摄氏度时的通知信息

发布者 (Publisher): 树莓派 +Arduino

订阅者 (Subscriber): Android 端

标题 (Title): temp\_notification

服务质量 (Qos): 1

消息内容 (Message):

{"status": 100, "msg\_temp": "温度低于 24 摄氏度, 是否关闭空调?"} 或者 {"status": 101, "msg\_temp": "温度高于 30 摄氏度, 是否打开空调?"}

说明: 继电器状态的传输, 也是需要较高的通信质量, 便于 Android 终端上继电器信息的更新。参数 status 代表不同情况下的状态码, 参数 msg\_temp 代表需要发送到 Android 端的通知信息。

## 第五章 系统实现

### 5.1 硬件电路搭建

这里采用 fritzing-0.9.2b，可以完美展示通过面包板搭建的电路情况，fritzing 提供了面包板的模型以及各项常用的元器件，可以很方便的提供接线的直观展示。

在 Debian 或者 Ubuntu 下，可以直接下载并解压，通过命令行切换到所在目录后启动：

```
lsy@debian:~/IOT_Project/Arduino/ide/fritzing-0.9.2b$ ./Fritzing
```

启动后，根据下图所示来切换到面包板的操作页面，如图5-1所示。

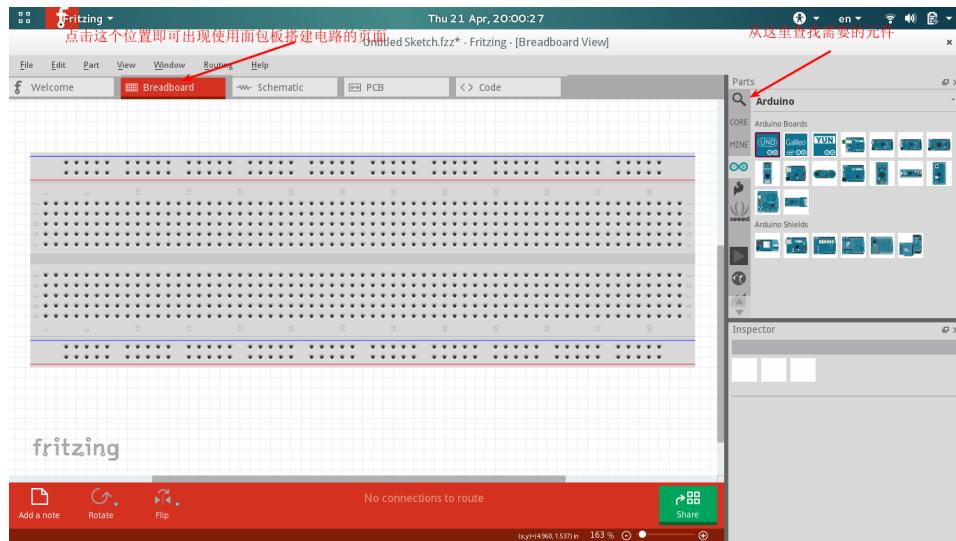


Figure 5-1 面包板电路搭建页面

工具准备完毕，开始电路搭建的内容。

#### 5.1.1 温度采集功能的电路搭建

首先选取所需的硬件设备以及传感器，上图中右侧 Parts 部分标注的是用来选择元器件以及单片机的地方，从这里选择当前需要的。下面开始搭建温度数据采集电路。

第一步需要添加所需的元件库，从 github 上这个地址：<https://github.com/adafruit/Fritzing-Library> 下载压缩包文件并解压。解压后，在 Fritzing 软件中，添加这个库，通过 File->Open... 弹出文件选择框，切换到解压后的文件目录，打开 AdaFruit.fzbz 文件，即可将这个元件库添加到软件中。操作如图5-2所示。

第二步是，添加相应的元件。在温度采集这个功能模块中，需要表5-1中的元器件。

其中杜邦线在画图的时候不需要，实际连接的过程中需要使用。下面图5-3是添加

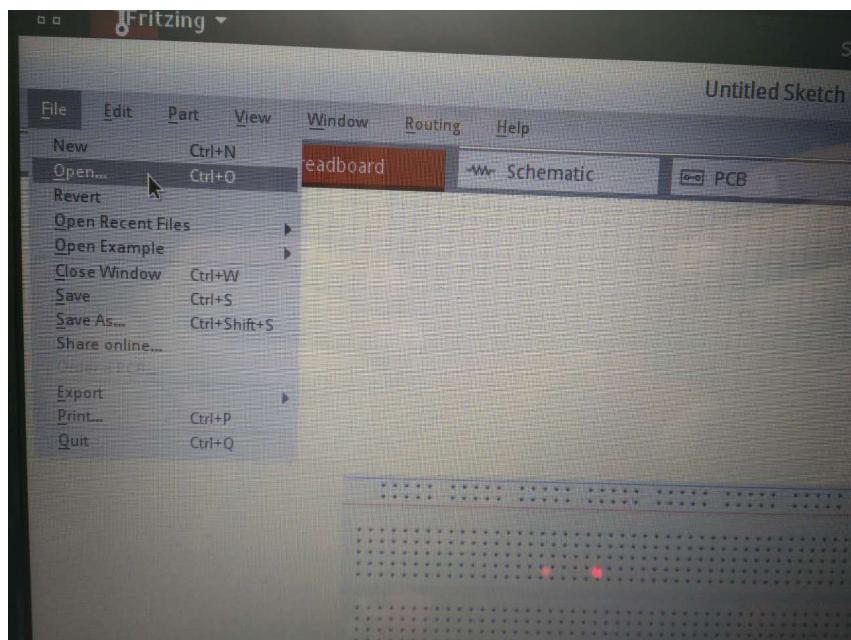


Figure 5-2 添加库操作

元器件	数量
Arduino UNO R3	1
DHT11 温湿度传感器	1
面包板	1
4.7kΩ 电阻	1
杜邦线（公对公）	5

Table 5-1 温度采集功能需要的元器件列表

过元件后的示意图。

第四步，在元件摆放完毕后，开始接线，如图5-4所示。

这样温度数据采集电路就设计完毕了。解释一下，这里图 5-4 中使用的是 4 线制的 DHT11 传感器，其中从左向右数第三个引脚需要悬空，而实际操作中，使用的是 3 线制的 DHT11，没有悬空的引脚。还有需要解释的是，接  $4.7\text{k}\Omega$  的上拉电阻，为的是在没有数据的时候保持高电位，减少读数的误差。下面图5-5是温度采集模块原理图的示意。

### 5.1.2 继电器模块的电路搭建

前面已经导入所需的元件库了，所以就不用再导入元件库了，还有就是元件库中只有两路的继电器，没有四路继电器模块，这里先用两路的继电器来替代。

第二步是，添加需要的元件。在继电器功能模块中，需要表5-2中的元器件。

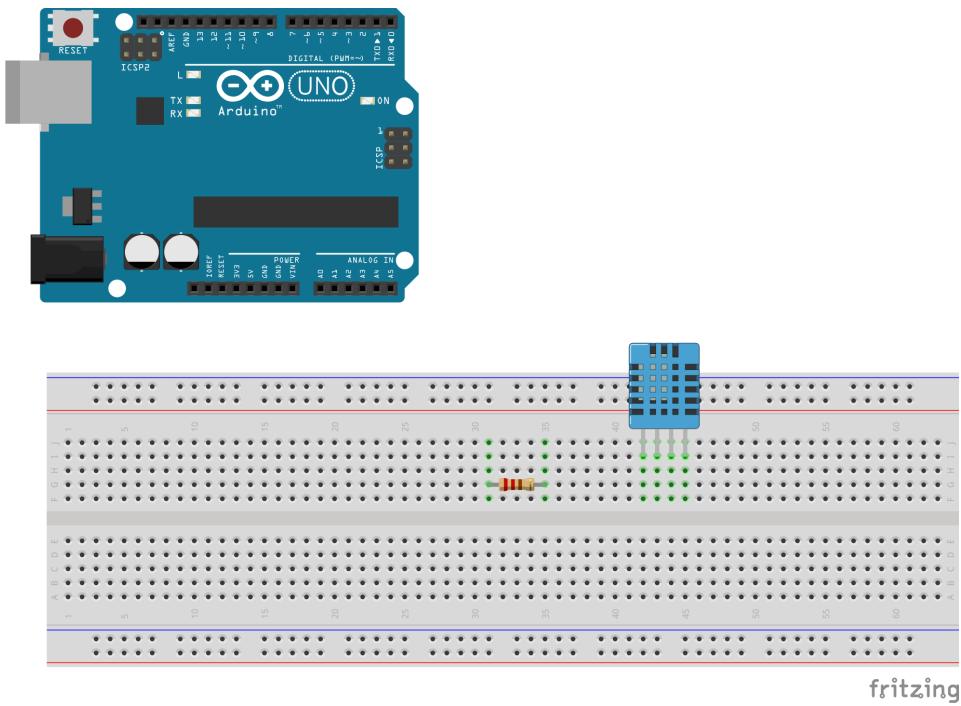


Figure 5-3 温度采集模块元件选择后拖动到工作区的示意图

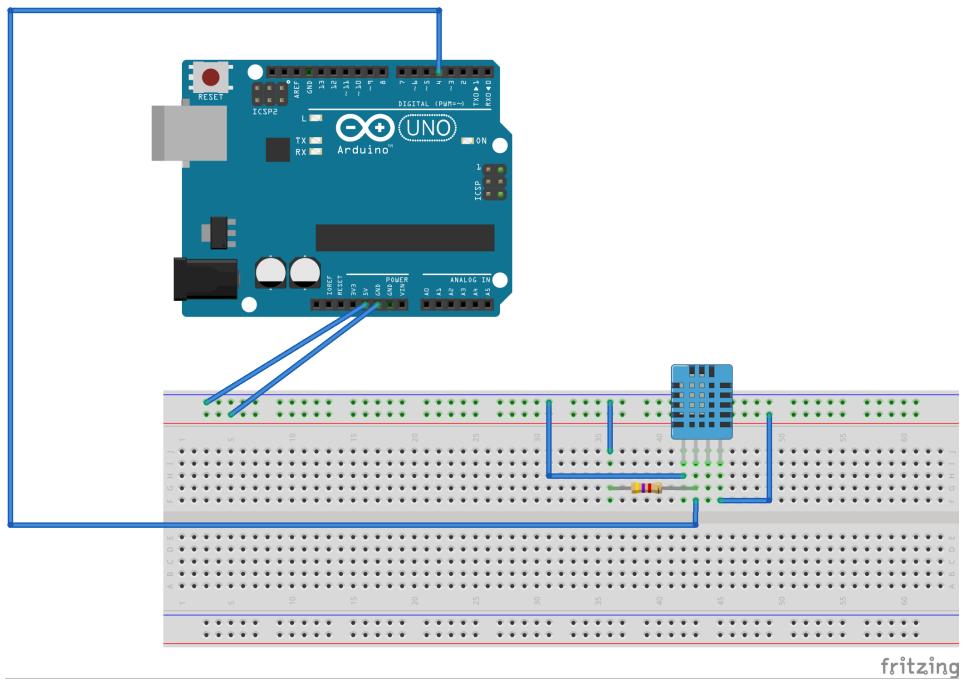
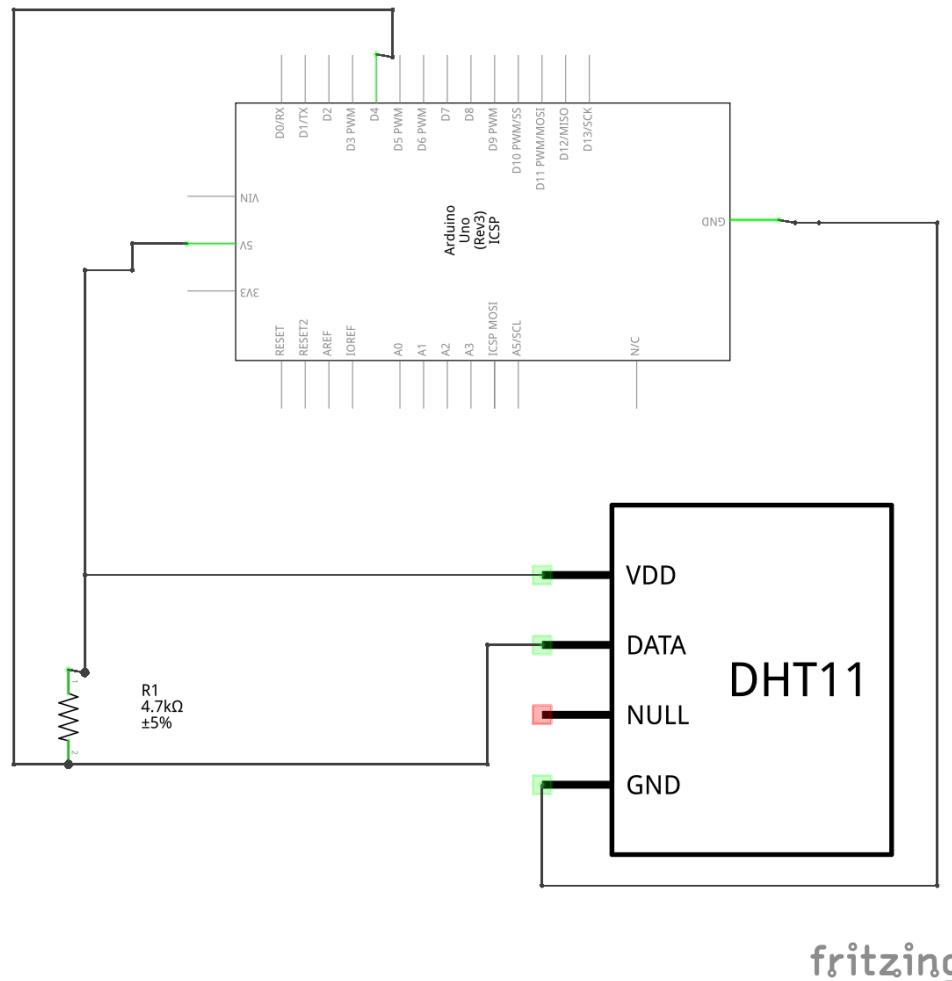


Figure 5-4 温度采集模块最终连线的示意图

然后就是将需要的器件放到工作区中，如图5-6所示。

然后下边进行接线，接线后如图5-7所示。

下面图5-8是通过接线图生成的原理图。



fritzing

Figure 5-5 温度采集模块原理图

元器件	数量
Arduino UNO R3	1
两路继电器模块	1
250Ω 电阻	1
发光二极管红色	1
杜邦线（公对公）	6

Table 5-2 继电器功能模块所需元器件

### 5.1.3 整体接线图和整体原理图

最终接线后的电路包含两个继电器控制的电路，接入另一个继电器的方法和上一个是一致的，所以整体接线图如图5-9：

整体电路原理图如图5-10：

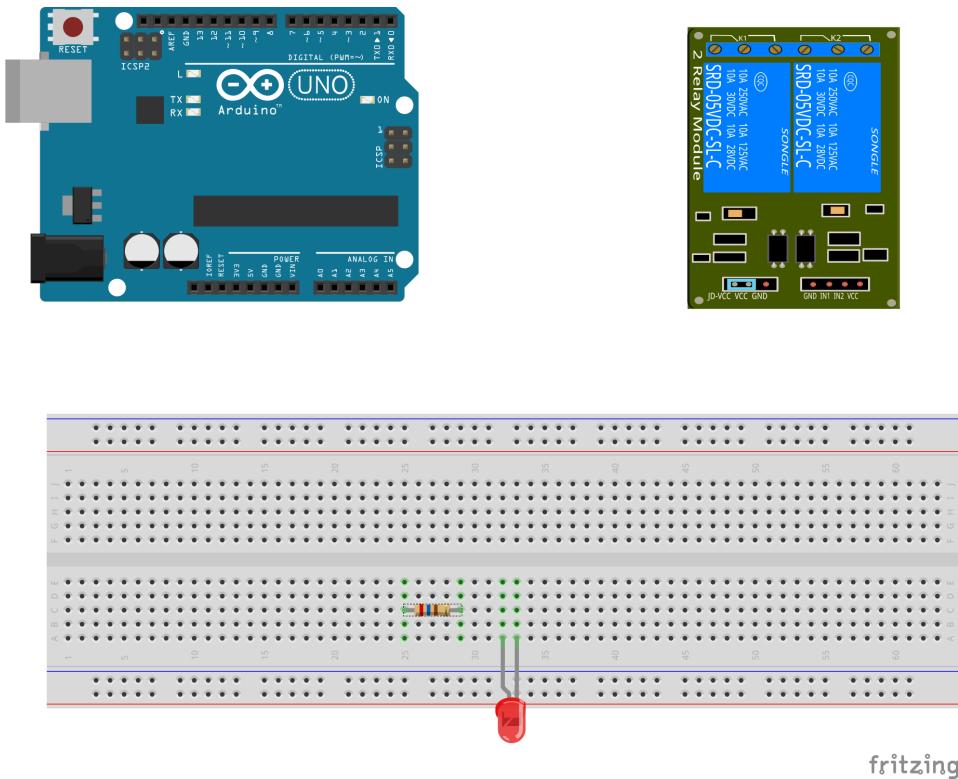


Figure 5-6 继电器功能模块元器件摆放图

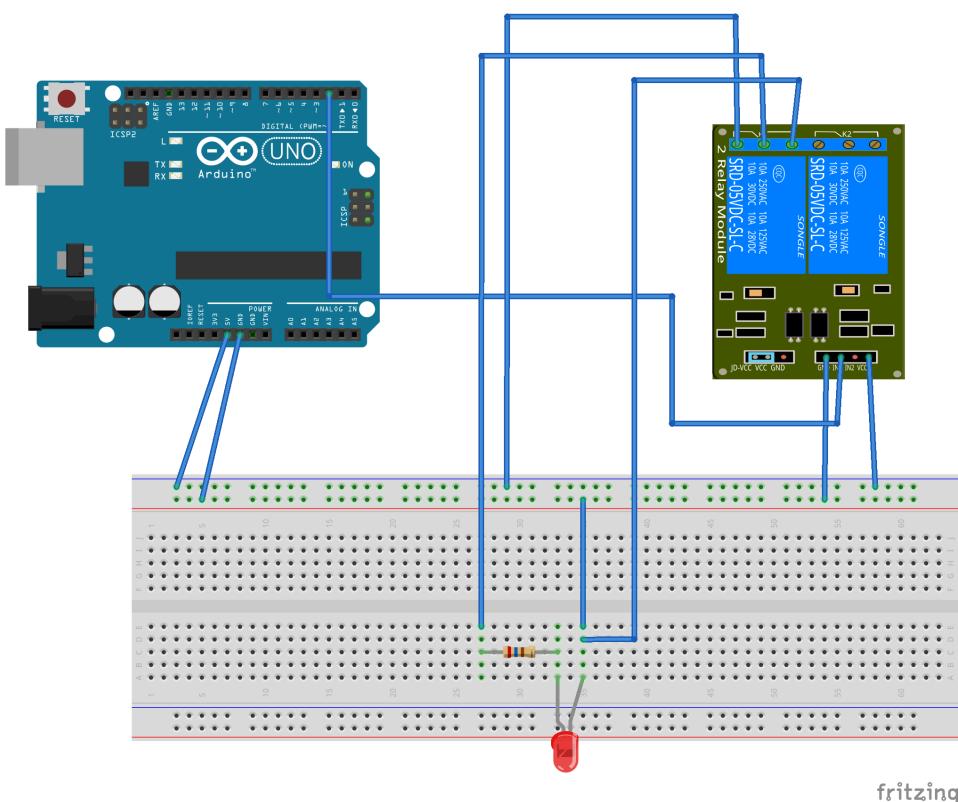
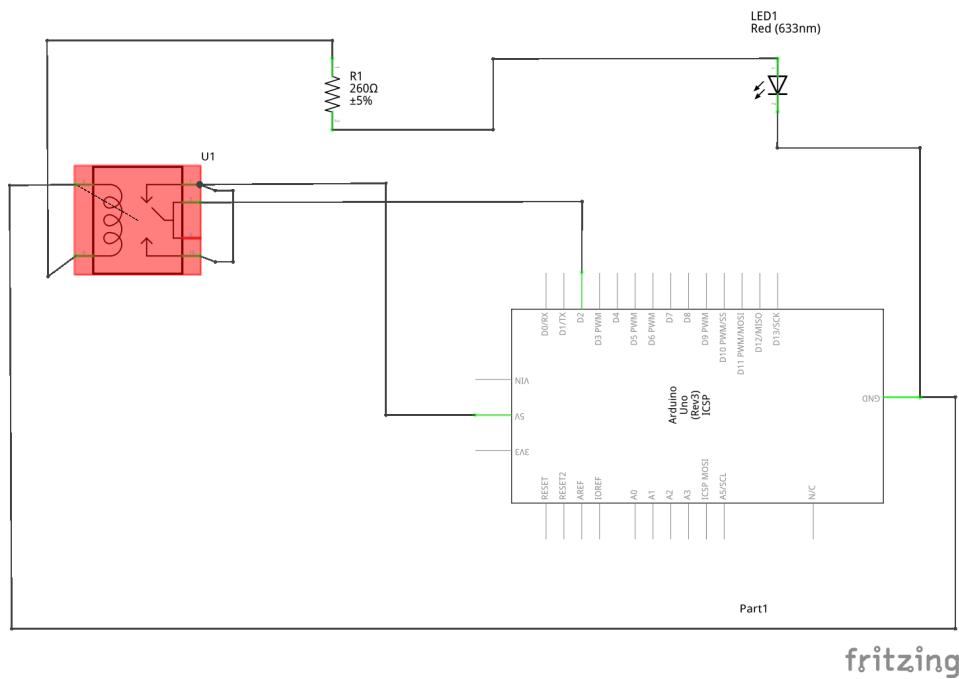


Figure 5-7 继电器功能模块接线图



fritzing

Figure 5-8 继电器功能模块原理图

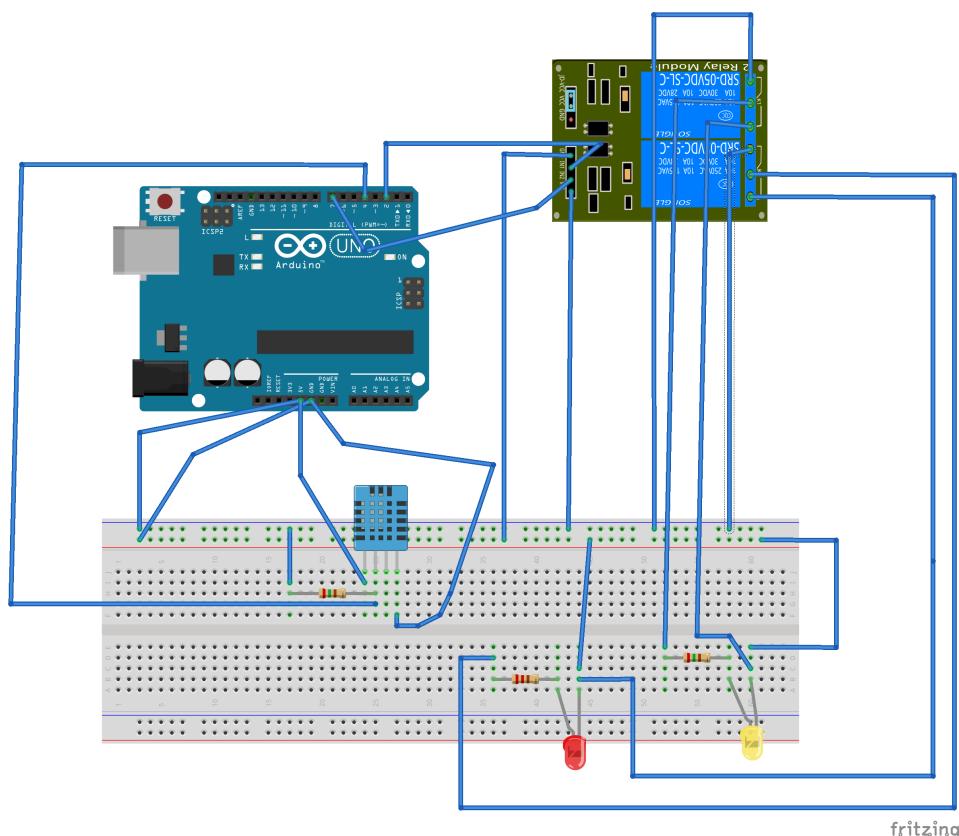


Figure 5-9 整体接线图

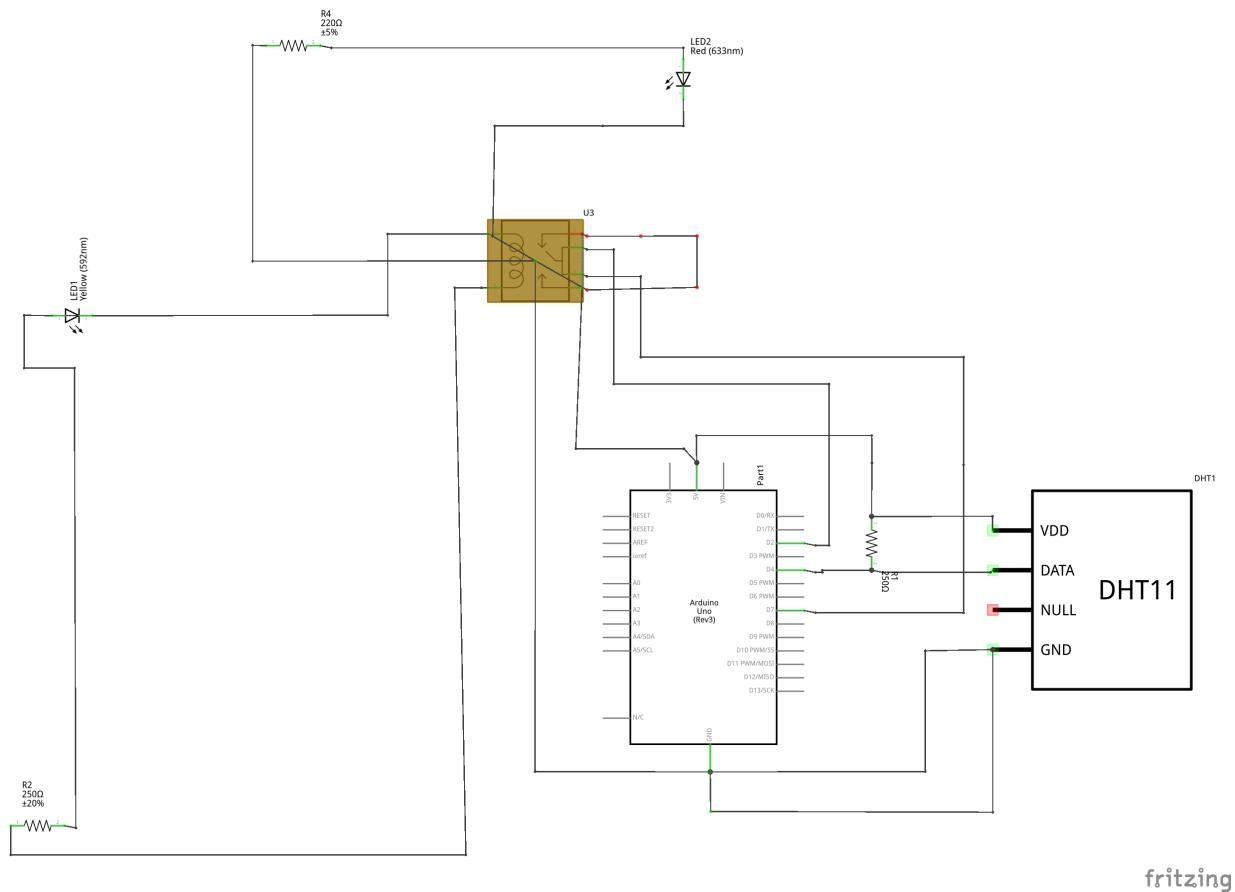


Figure 5-10 整体电路原理图

## 5.2 代码编写思路

硬件方面的电路搭建在上一节已经完成，剩下的重中之重就是编写代码了。出于需求难易的考虑，先来编写服务端的代码，因为服务端代码功能是实现消息转发，难度比较低，而且可以使用成熟的第三方开源框架，这样就不用去为实现整个 MQTT 协议而考虑了，更多的将精力放到业务逻辑的开发上；紧接着来实现树莓派端的代码编写，需要实现的是面向硬件的操作以及消息的发送和接收；最后来实现 Android 端的代码，主要实现 UI 界面的编写以及消息的发送和接收这两个功能。所有的代码编写都需要遵循环境搭建和代码编写这样两个环节，在下面每一节的编写过程中，都遵循这两个环节来进行。

## 5.3 服务端代码编写

服务端的主要功能就是进行消息转发。如果实现整个的 MQTT 协议，工作量很大，而且业务逻辑复杂，超出不少工作量。这里借助第三方的开源框架——`hbmqtt` 来进行开发，将更多精力放到接口协议的开发上。

### 5.3.1 环境搭建

当你购买了阿里云服务器时，只有一个 root 用户。直接对 root 用户操作不是不可以，但是这样做危险性很大，万一操作不当出现了问题，很难补救。这样先来创建一个用户，并赋予他 root 权限。然后在这个用户下，搭建 python 开发环境。这个就是搭建开发环境的简要步骤。说明一下，所有终端命令的操作步骤中，# 后面空格后跟随的中文内容均为注释。

#### 5.3.1.1 创建新用户并赋予 root 权限

前面已经说明，这里所有操作均在 Debian8 环境下进行。

首先打开终端，通过 ssh 命令连接到服务器，命令如下：

```
abcd@1234:~$ ssh root@114.225.92.215 # root是用户名, @符号后边是服务器外网ip,
      需要替换成你自己的外网ip
The authenticity of host '192.168.0.106 (192.168.0.106)' can't be established.
ECDSA key fingerprint is b4:0c:a9:dd:b2:bd:e9:57:fd:9d:96:10:90:59:48:b2.
Are you sure you want to continue connecting (yes/no)? yes # 第一次登陆需要创建ssh-key
root@114.225.92.215's password: # 从这里输入密码

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Apr 24 14:01:16 2016
root@iZ280pdfadf:~#                                     # 由于是root用户登陆，所以登陆后显示#,
      普通用户显示$
```

经过上面的代码演示，当前已经以 root 用户的身份登陆远程服务器了。

下面来创建用户，使用 useradd 命令，演示如下：

```
root@iZ280pdfadf:~# adduser iot_test # iot_test是所创建用户的用户名
Adding user `iot_test' ...
Adding new group `iot_test' (1001) ...
Adding new user `iot_test' (1001) with group `iot_test' ...
Creating home directory `/home/iot_test' ...
Copying files from `/etc/skel' ...
Enter new UNIX password: # 在这里输入密码并记住该用户的密码
Retype new UNIX password: # 密码重复输入
passwd: password updated successfully
Changing the user information for iot_test
Enter the new value, or press ENTER for the default      # 确认个人信息,
      这里均使用默认值(default)
      Full Name []:
      Room Number []:
      Work Phone []:
      Home Phone []:
      Other []
Is the information correct? [Y/n] Y # 信息确认
```

```
root@iZ280pdfadf: # 到此用户创建完毕
```

依据以上步骤，用户就创建完毕了，下面需要为创建的用户添加 root 权限，有两种方式：

方法一：修改 /etc/sudoers 文件，找到下面一行，把前面的注释 (#) 去掉

```
## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)  ALL
```

然后修改用户，使其属于 root 组 (wheel)，命令如下：

```
#usermod -g root iot_test
```

修改完毕，现在可以用 iot\_test 帐号登录，然后用命令 su –，即可获得 root 权限进行操作。

方法二：修改 /etc/sudoers 文件，找到下面一行，在 root 下面添加一行，如下所示：

```
## Allow root to run any commands anywhere
root  ALL=(ALL)  ALL
iot_test  ALL=(ALL)  ALL
```

修改完毕，现在可以用 iot\_test 帐号登录，然后用命令 sudo –，即可获得 root 权限进行操作。

方法三：修改 /etc/passwd 文件，找到如下行，把用户 ID 修改为 0，如下所示：

```
iot_test:x:0:33:iot_test:/data/webroot:/bin/bash
```

这里我采用的是方法二进行操作，演示如下：

```
root@iZ280pdfadf:~# vim /etc/sudoers
# 下面是sudoers文件中的全部内容
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults      env_reset
Defaults      mail_badpass
Defaults      secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin"
#
# Host alias specification
#
# User alias specification
#
# Cmnd alias specification
#
# User privilege specification
```

```

root    ALL=(ALL:ALL)  ALL
iot_test      ALL=(ALL:ALL)  ALL          # 要从这个位置添加
# Members of the admin group may gain root privileges
%admin  ALL=(ALL)  ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL)  ALL

# See sudoers(5) for more information on "#include" directives:

#includeincludedir /etc/sudoers.d

:w !sudo tee %      # 先敲Esc, 然后按照前面所示键入这条命令。这条命令是保存文件操作的命令
W12: Warning: File "/etc/sudoers" has changed and the buffer was changed in Vim as
     well
See ":help W12" for more info.
[0]K, (L)oad File: 0      # 输入0选择保存

Press ENTER or type command to continue  # 再次回车确认

:q!      # 输入这条命令退出编辑
root@iZ280pdfadf:~#

```

这样就为刚刚创建的用户添加了 root 权限，这时候需要退出 root 用户的登陆，使用 iot\_test 用户来登陆服务器，操作如下：

```

root@iZ280pdfadf:~# exit          # 通过exit命令退出当前登陆
logout
Connection to 114.225.92.215 closed.
abcd@1234:~$ ssh iot_test@114.225.92.215    # 以iot_test用户的身份登陆
iot_test@114.225.92.215's password:
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-32-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Welcome to aliyun Elastic Compute Service!

iot_test@iZ280pdfadf:~$

```

到此为止，添加用户以及为所添加的用户添加 root 权限已经顺利完成。

### 5.3.1.2 搭建 python 开发环境

首先来检测一下 python 环境，直接在命令行中输入 python 或者 python3，看看控制台输出如何，代码如下：

```
iot_test@iZ280pdfadf:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>                                         # 退出的时候快捷键为ctrl+d
iot_test@iZ280pdfadf:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
iot_test@iZ280pdfadf:~$
```

这样就验证了 python 环境的完整性，python2.7 和 python 3.4 版本均存在，不需要安装。

那么在开发的时候会有两种选择，一种是直接在当前环境下操作，这样比较直接，但是可能对整个环境中其他程序的运行造成影响，因为你是对全局的 python 环境进行操作；还有一种方式是，通过建立虚拟环境，让 python 程序运行在虚拟环境中，这样就会将虚拟环境与全局环境进行隔离，方便管理。所以采用虚拟环境的方式来运行 python 程序。那么首先要安装搭建虚拟环境需要的程序，如下面代码所示：

```
iot_test@iZ280pdfadf:~$ sudo apt-get update      # 安装之前习惯性的先update一下
iot_test@iZ280pdfadf:~$ sudo apt-get python-pip python-dev # 安装python环境中需要的组件
iot_test@iZ280pdfadf:~$ sudo pip install virtualenv # 安装虚拟环境创建的工具
```

所有程序安装完毕后，开始创建 python 虚拟环境，创建过程如下：

```
iot_test@iZ280pdfadf:~$ mkdir IOT    # 创建文件夹
iot_test@iZ280pdfadf:~$ ls          # 列举文件夹位置
IOT
iot_test@iZ280pdfadf:~$ cd IOT/    # 切换到IOT目录下
iot_test@iZ280pdfadf:~/IOT$ virtualenv -p python3 venv  # 创建虚拟环境，-p指定python
的版本，使用python3
Running virtualenv with interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in venv/bin/python3
Also creating executable in venv/bin/python
Installing setuptools, pip, wheel...done.
iot_test@iZ280pdfadf:~$
```

这样虚拟环境就创建好了，下一步就是让虚拟环境生效。过程如下：

```
iot_test@iZ280pdfadf:~/IOT$ source venv/bin/activate  # 使虚拟环境生效的命令
(venv)iot_test@iZ280pdfadf:~/IOT$ deactivate # 退出虚拟环境的命令
iot_test@iZ280pdfadf:~/IOT$
```

很明显，当虚拟环境生效时，iot\_test 前面多出了“(venv)”这些字符，表明当前处于虚拟环境中；需要退出虚拟环境时，可以在任何时刻执行 deactivate 命令，即可退出。当虚拟环境生效后，在这个虚拟环境中进行开发，同外部的 python 环境进行隔离，方

便管理。

那么在最终开发之前，还需要安装 hbmqtt 组件，一条命令即可：

```
(venv)iot_test@iZ280pdfadf:~/IOT$ pip install hbmqtt
```

在虚拟环境下安装组件，不需要 root 权限，直接安装即可。这样环境搭建就完成了。

### 5.3.2 代码编写

代码编写的时候分为三个部分来处理，第一个是配置文件，配置文件需要使用 YAML 语言来进行配置，简单易懂；第二个就是真正的源代码了，包含日志处理、配置文件导入、服务器启动三个功能函数；最后是启动文件，需要配置开机启动，而且在后台自动运行。设置日志处理的原因就是防止程序突然停止，通过日志找到程序崩溃的根源。所有代码使用 vim 编辑器进行编写。源代码如下：

```
# YAML配置文件——config.yaml
listeners:
  default:
    max-connections: 5000      # 最大连接数
    type: tcp                # TCP方式
    bind: 0.0.0.0:1883       # 绑定端口，设置允许外网访问
```

配置文件主要就是设定了三项内容，下面是源文件的代码：

```
# 源文件——serins.py
#!/usr/bin/env python
# encoding: utf-8

import logging
import asyncio
import os
import yaml
from hbmqtt.broker import Broker

# 日志文件配置
def init_log():
    console_file = logging.FileHandler('application.log')
    # 设置日志级别
    console_file.setLevel(logging.DEBUG)
    # 设置日志格式
    file_formatter = logging.Formatter('[(asctime)s] :: %(levelname)s :: %(name)s :: %(message)s')
    console_file.setFormatter(file_formatter)
    logging.getLogger('').addHandler(console_file)

# 导入配置文件
stream = open('/home/iot_test/IOT/config.yaml','r')
yaml_conf = yaml.load(stream)
broker = Broker(yaml_conf)

# 启动hbmqtt服务器
@asyncio.coroutine
```

```

def test_coro():
    yield from broker.start()

if __name__ == '__main__':
    # 这里是控制台显示日志信息的配置
    formatter = "[%(asctime)s] :: %(levelname)s :: %(name)s :: %(message)s"
    logging.basicConfig(level=logging.INFO, format=formatter)
    init_log()
    # 确保服务器启动完成
    asyncio.get_event_loop().run_until_complete(test_coro())

```

其实到这里，这个服务器程序可以正常运行了，可以在命令行当中这样执行：

```
(venv)iot_test@iZ280pdfadf:~/IOT$ python serins.py
```

但是这样执行后，整个终端只是在运行这一个程序，每一次服务器启动都得去像上面那样手动执行这条命令，如果需要在这个终端中再去做其他工作，必须终止程序运行才行。所以下面需要配置开机启动。

首先编写开机启动的脚本文件：

```

#!/bin/bash

source /home/iot_test/IOT/venv/bin/activate # 启动虚拟环境

python /home/iot_test/IOT/serins.py & # 脚本执行

```

需要先启动虚拟环境再去执行脚本启动的命令，这样保证确实是在自己创建的虚拟环境中运行。

最后需要把这个文件添加开机启动，过程如下：

```

(venv)iot_test@iZ280pdfadf:~/IOT$ sudo chmod 755 start.sh # 修改文件权限
(venv)iot_test@iZ280pdfadf:~/IOT$ sudo cp start.sh /etc/init.d/ # 将文件粘贴到etc/
init.d/目录下
(venv)iot_test@iZ280pdfadf:~$ cd /etc/init.d
(venv)iot_test@iZ280pdfadf:/etc/init.d$ sudo update-rc.d start.sh defaults 95 #
设置启动顺序
update-rc.d: warning: /etc/init.d/start.sh missing LSB information
update-rc.d: see <http://wiki.debian.org/LSBInitScripts>
Adding system startup for /etc/init.d/start.sh ...
/etc/rc0.d/K95start.sh -> ../init.d/start.sh
/etc/rc1.d/K95start.sh -> ../init.d/start.sh
/etc/rc6.d/K95start.sh -> ../init.d/start.sh
/etc/rc2.d/S95start.sh -> ../init.d/start.sh
/etc/rc3.d/S95start.sh -> ../init.d/start.sh
/etc/rc4.d/S95start.sh -> ../init.d/start.sh
/etc/rc5.d/S95start.sh -> ../init.d/start.sh
iot_test@iZ280p9hiuZ:/etc/init.d$

```

添加完成后，可以通过阿里云的控制台或者直接在终端中使用 sudo reboot 命令来重启服务器。重启完毕后，重新登陆 iot\_test 用户，再使用下面的命令来查看：

```
lsyAndroid@iZ280p9hiuZ:/etc/init.d$ ps -ef | grep python
# 下面这一行显示的就是自己编写的脚本已经在后台运行了
root      995      1  0 Apr25 ?          00:00:00 python /home/iot_test/IOT/serins.py
lsyAndr+  3998  3747  0 07:59 pts/0    00:00:00 grep --color=auto python
lsyAndroid@iZ280p9hiuZ:/etc/init.d$
```

这样整个服务端的内容就全部完成了，此时服务端已经正常工作了。

## 5.4 树莓派端的代码编写

在硬件电路搭建完毕的情况下，需要通过树莓派 +Arduino 的组合来对硬件电路实现控制。Arduino 充当树莓派与硬件电路之间的接口板，树莓派作为与服务端对接的端口，也就是通过树莓派来实现消息的发送和接收。

在这个部分中，同样分为环境搭建和代码编写两个部分。环境搭建包含树莓派的系统镜像制作、使用前的初步设置（主要是解决屏幕显示以及内存卡空间不足的问题）、相关软件的安装以及接口板的制作这四个方面的内容；代码编写分为硬件控制内容的编写和消息接口的实现这两个方面的内容。

### 5.4.1 环境搭建

#### 5.4.1.1 树莓派系统镜像制作

先从 <https://www.raspberrypi.org/downloads/raspbian/> 上下载树莓派的镜像，选择图5-11所示的镜像文件，点击 Download Zip 下载。

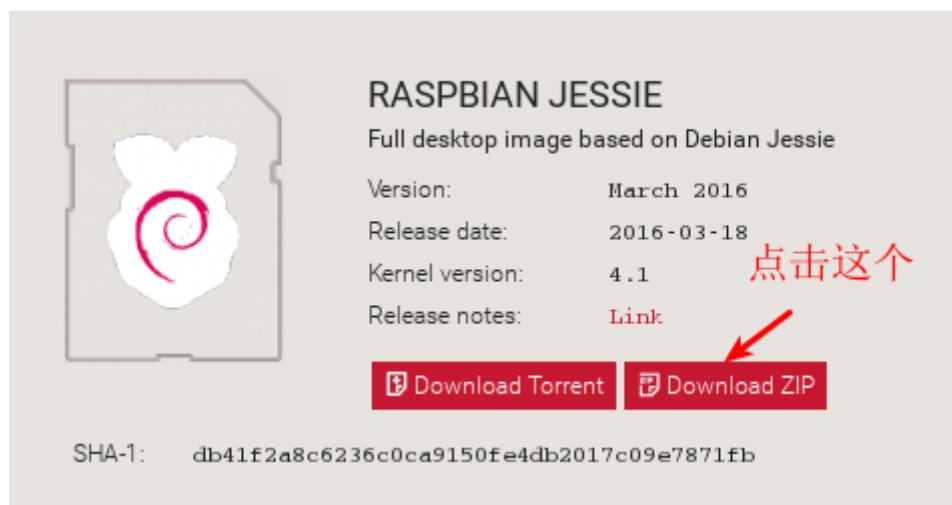


Figure 5-11 下载镜像示意图

下载完毕后将镜像写入到 SD 卡中，这里使用命令行来操作。

首先来验证下压缩包没有被串改，命令如下：

```
lsyAndroid@iZ280p9hiiuZ:~$ cd Downloads/
lsyAndroid@iZ280p9hiiuZ:~/Downloads$ sha1sum 2016-03-18-raspbian-jessie.zip
db41f2a8c6236c0ca9150fe4db2017c09e7871fb  2016-03-18-raspbian-jessie.zip
lsyAndroid@iZ280p9hiiuZ:~/Downloads$
```

将计算出的 sha1 码，同官网给出的 sha1 码进行比对，也就是如图 5-12 所示，如果比对结果不正确，需要重新下载。

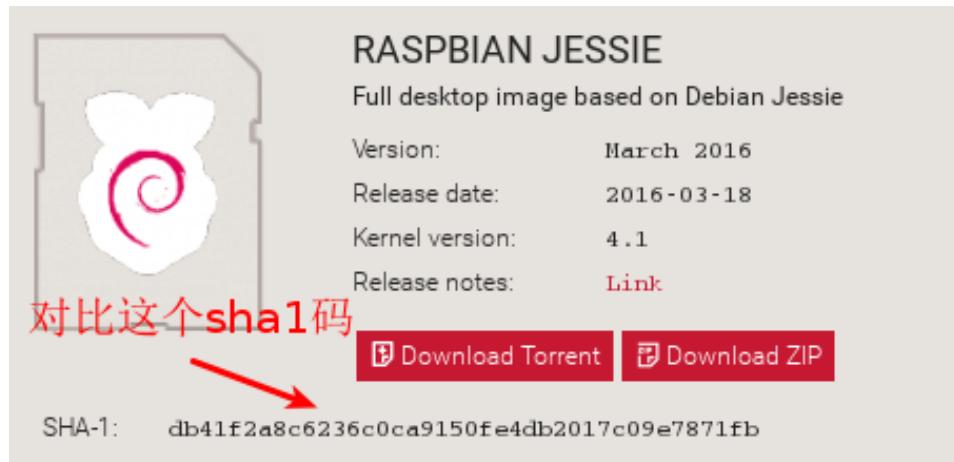


Figure 5-12 签名文件对比图

然后解压文件，得到 2016-03-18-raspbian-jessie.img 镜像文件。下一步将准备的内存卡格式化，只有格式化之后，内存卡才能被系统所识别，这一步需要你自行操作。

随后用两次 df 命令来确定内存卡的“代号”——所属的 FileSystem，过程如下：

```
lsyAndroid@iZ280p9hiiuZ:~/Downloads$ df -h # 第一次使用df -h命令
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb5        54G   43G   7.4G  86% /
udev            10M    0    10M   0% /dev
tmpfs           1.5G  9.2M  1.5G   1% /run
tmpfs           3.7G  36M   3.6G   1% /dev/shm
tmpfs           5.0M  4.0K   5.0M   1% /run/lock
tmpfs           3.7G    0   3.7G   0% /sys/fs/cgroup
tmpfs          742M  8.0K  742M   1% /run/user/120
tmpfs          742M   28K  742M   1% /run/user/1000

# 这时通过读卡器将内存卡插到电脑上

lsyAndroid@iZ280p9hiiuZ:~/Downloads$ df -h # 第二次使用df -h命令
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb5        54G   43G   7.4G  86% /
udev            10M    0    10M   0% /dev
tmpfs           1.5G  9.2M  1.5G   1% /run
tmpfs           3.7G  36M   3.6G   1% /dev/shm
tmpfs           5.0M  4.0K   5.0M   1% /run/lock
tmpfs           3.7G    0   3.7G   0% /sys/fs/cgroup
tmpfs          742M  8.0K  742M   1% /run/user/120
tmpfs          742M   28K  742M   1% /run/user/1000
/dev/sdc1       15G   8.0K   15G   1% /media/iZ280p9hiiuZ/my # 制作系统的内存卡
```

下一步就是需要卸载设备，操作如下：

```
lsyAndroid@iZ280p9hiuZ:~/Downloads$ umount /dev/sdc1
lsyAndroid@iZ280p9hiuZ:~/Downloads$
```

卸载完成后，使用 dd 命令将镜像文件写入到内存卡中，操作如下：

```
lsyAndroid@iZ280p9hiuZ:~/Downloads$ sudo dd bs=4M if=2016-03-18-raspbian-jessie.img
of=/dev/sdc1
```

这里需要解释一下命令中的参数，bs 代表一次写入多大的块，是 blocksize 的缩写，4M 一般都没问题，如果不行，试试改成 1M，if 参数为下载的镜像的路径（应该是 input file 缩写），of 后参数为设备地址（应该是 output file 的缩写，linux 上一切都是文件）。

这里必须强调的是，千万不要写错这里的参数，否这你可能丢失硬盘所有数据。dd 命令的破坏力很强，操作不当可能会删除硬盘上的所有数据，所以操作起来一定要慎重！

这样执行命令即可，会出现什么都不显示的现象，这是因为 dd 命令没有进度的提示，这样等待几分钟即可，当镜像完全写入后，会出现下面的提示：

```
961+1 records in
961+1 records out
4033871872 bytes (4.0 GB) copied, 512.258 s, 7.9 MB/s
lsyAndroid@iZ280p9hiuZ:~/Downloads$
```

这样镜像就制作完成了。

#### 5.4.1.2 树莓派系统的基本设置

把内存卡插到树莓派上，开机启动，屏幕上显示的内容与正常的显示完全是相反的，如图5-13所示：

如果需要翻转 180 度，需要修改 config.txt 文件，操作过程如下：

```
pi@raspberrypi:~$ sudo nano /boot/config.txt # 树莓派刚刚安装没有vim编辑器，所以使用nano
编辑器

# 跳转到文档最后
# 添加下面的一句代码
lcd_rotate=2    # 记住“=”两边不要带空格，这一句的意思是将树莓派屏幕旋转180度
# 保存退出
```

上边这种方式实在树莓派上直接修改，也可以关机断电后将内存卡取出，在你的机器上按照同样的路径找到文件修改，修改完毕后再将内存卡插到树莓派上启动即可，这样开机后就正常了！

然后是需要让树莓派使用整个内存卡空间。当你完成镜像制作的时候，内存卡还是

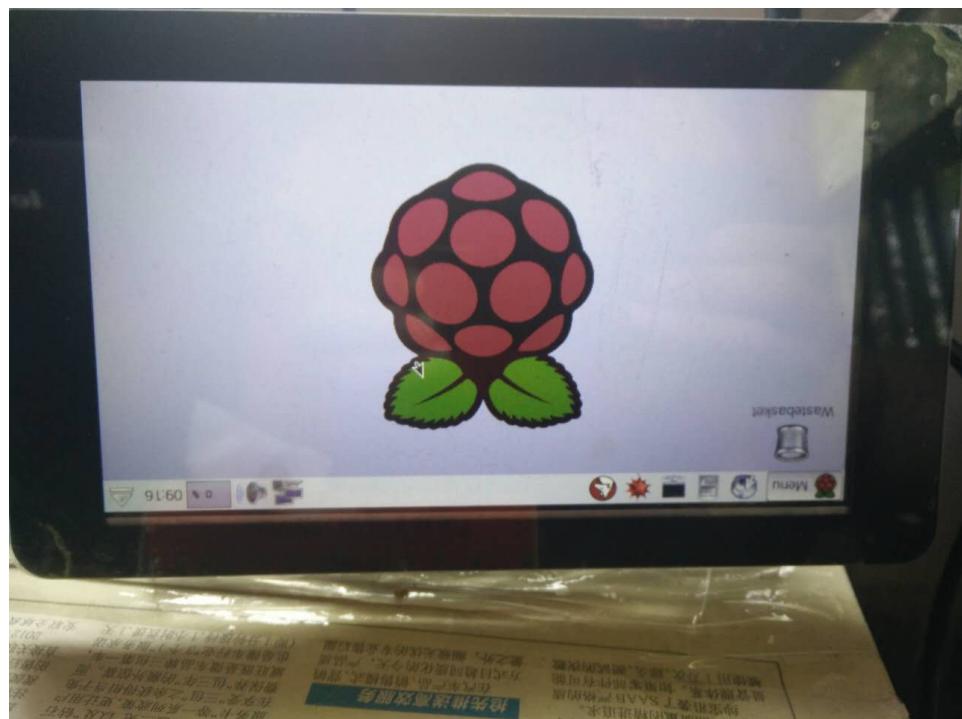


Figure 5-13 树莓派官方屏幕显示内容倒置

有很大一部分空间没有使用，如果不去利用这部分空间，在后边安装完 git 和 vim 剩余空间将会很小，一个直观的感受就是在输入命令的时候，敲 tab 进行提示，就会出现下面的情况：

```
pi@raspberrypi:/etc/vim $ cd /usr-bash: cannot create temp file for here-document: No
space left on device
```

首先来看看目前的空间使用情况，使用下面的命令来展示：

```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       3.6G  3.4G     0 100% /
devtmpfs        459M    0  459M   0% /dev
tmpfs          463M    0  463M   0% /dev/shm
tmpfs          463M  6.4M  457M   2% /run
tmpfs          5.0M  4.0K  5.0M   1% /run/lock
tmpfs          463M    0  463M   0% /sys/fs/cgroup
/dev/mmcblk0p1    60M   20M   41M  34% /boot
tmpfs          93M    0   93M   0% /run/user/1000
```

下面是解决这个问题的步骤，首先输入以下命令：

```
pi@raspberrypi:~$ sudo raspi-config
```

输入命令后会出现如图5-14所示的情况。

选择第一项并敲回车键会出现如图5-15所示的情况。

继续敲回车，会回到最开始输入命令后展示的页面，这时候通过方向左键或者方向

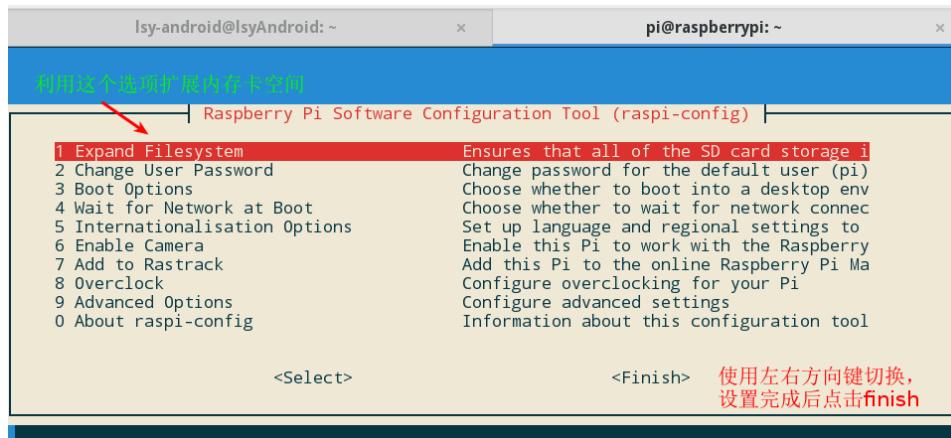


Figure 5-14 输入 sudo raspi-config 后的界面



Figure 5-15 扩展存储空间后的页面

右键选择 finish 并回车后，会出现重启的界面，如图5-16所示。



Figure 5-16 设置完成后的重启界面

这时候选择 YES 来重新启动，重启完毕后，再来执行 df -h 命令，如下所示：

```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       15G   3.4G   11G  25% /
# 容量大大扩展
```

devtmpfs	459M	0	459M	0%	/dev
tmpfs	463M	0	463M	0%	/dev/shm
tmpfs	463M	6.4M	457M	2%	/run
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	463M	0	463M	0%	/sys/fs/cgroup
/dev/mmcblk0p1	60M	20M	41M	34%	/boot
tmpfs	93M	0	93M	0%	/run/user/1000

这样内存卡的扩展操作也就完成了。

最后是连接外网，由于使用了免驱 wifi 网卡，可以像正常的电脑那样连接无线网络，也可以通过命令行来进行设置，方法如下：

```
pi@raspberrypi:~$ lsusb # 列举出usb设备
# 下面这个就是无线网卡的设备
Bus 001 Device 004: ID 0bda:8176 Realtek Semiconductor Corp. RTL8188CUS 802.11n WLAN
    Adapter
.... 下面还有很多设备

pi@raspberrypi:~$ ifconfig
wlan0      Link encap:Ethernet  HWaddr 00:16:3e:00:4f:15
            inet addr:fdasffadffdsaf  # 关键在这一行, wlan0不显示ip地址,
但是这个设备是存在的

pi@raspberrypi:~$ sudo nano /etc/network/interfaces
# 修改文件内容如下
auto lo

iface lo inet loopback
iface eth0 inet dhcp

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
    wpa-ssid "MERCURY_4E68"      # 这个是wifi名称, 这个需要填写你自己的wifi名称
    wpa-psk "password" # 这个是wifi密码, 这个需要填写wifi的密码
address 192.168.1.102 # 设定的静态IP地址, ssh连接的时候就不需要再通过路由器查看树莓派的
ip地址
netmask 255.255.255.0 # 网络掩码
gateway 192.168.1.1 # 网关
network 192.168.1.1 # 网络地址
iface default inet dhcp # 设置默认连接
# 保存退出
```

详细来解释一下修改的内容：

auto lo //表示使用 localhost

iface eth0 inet dhcp //表示如果有网卡 eth0, 则用 dhcp 获得 IP 地址 (这个网卡是本机的网卡, 而不是 WIFI 网卡)

auto wlan0 //表示如果有 wlan 设备, 使用 wlan0 设备名

allow-hotplug wlan0 //表示 wlan 设备可以热插拔

iface wlan0 inet dhcp //表示如果有 WLAN 网卡 wlan0 (就是 WIFI 网卡), 则用 dhcp 获

## 得 IP 地址

其他数据在上面命令行代码展示的过程中已经解释过了。

最后使用下面的命令来立即连接所配置的 wifi:

```
pi@raspberrypi:~$ sudo /etc/init.d/networking restart
. ok 说明启动成功，也可以直接重启树莓派来连接
pi@raspberrypi:~$ ifconfig # 用 ifconfig 命令可以看到 wlan0 设备
wlan0      Link encap:Ethernet HWaddr 00:16:3e:00:4f:15
           inet addr:192.168.1.102 # 有了IP地址(已连接)，说明无线网络配置成功
```

这样 wifi 配置完成后可以通过路由器接入互联网了，这时候可以通过 ssh 来连接树莓派，就不再需要屏幕了。这里需要强调的是，通过 HDMI 接口也可以连接到显示器上，如果显示器没有 HDMI 接口的，需要通过 HDMI 转 VGA 转换器来进行信号转换，需要强调的是转换器需要通过外部电源供电才能保证稳定工作，也就是使用有源的转换器才可以。

下面通过 ssh 命令连接树莓派，操作如下：

```
lsy-android@lsyAndroid:~$ ssh pi@192.168.0.102
ssh: connect to host 192.168.0.102 port 22: No route to host
# 可能会出现失败的情况，所以需要查看设置没有错误的情况下，再重试几次
lsy-android@lsyAndroid:~$ ssh pi@192.168.0.102The authenticity of host '192.168.0.102
(192.168.0.102)' can't be established.
ECDSA key fingerprint is 3f:7d:8f:8c:4b:6d:9e:70:0f:6e:28:28:bb:51:01:86.
Are you sure you want to continue connecting (yes/no)? yes # 保存ssh key
Warning: Permanently added '192.168.0.102' (ECDSA) to the list of known hosts.
pi@192.168.0.102's password: # 树莓派的初始密码是raspberry

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Mar 18 09:16:05 2016
pi@raspberrypi:~ $
```

这样网络配置就完成了，还需要更换一下树莓派的源，因为默认使用的是国外的源，下载和更新的速度都比较慢，所以需要更换为国内的源，过程如下：

```
pi@raspberrypi:~ $ sudo nano /etc/apt/source.list
# 删除原文件所有内容，编辑文档内容如下：
deb http://mirror.sysu.edu.cn/raspbian/raspbian/ jessie main contrib non-free
deb-src http://mirror.sysu.edu.cn/raspbian/raspbian/ jessie main contrib non-free
# 保存退出

pi@raspberrypi:/etc/apt $ sudo apt-get update # 手动更新源
```

需要说明的是最新的系统中，即使你更新了国内源，它还是会去连接国外的源，但是国内源的优先级高，除了 update 的时候略有影响，其他安装软件的时候影响不大！

下面需要安装一下基本的开发工具和配置环境，首先需要安装 git 和 vim，过程如下：

```
pi@raspberrypi:~ $ sudo apt-get install git vim # 主要是安装git和vim这两个工具
```

vim 这里需要简单来配置一下，以方便使用，编写代码的过程中可以更加方便。由于树莓派性能不高，这里的配置比较简单，过程如下：

```
pi@raspberrypi:~ $ cd Downloads/ # 切换到Downloads目录下
pi@raspberrypi:~/Downloads $ git clone https://github.com/wklken/vim-for-server.git
# 下载

# 下载完毕后需要将vim-for-server下的vimrc文件链接到用户pi的目录下
pi@raspberrypi:~/Downloads $ sudo ln -s vim-for-server/vimrc ~/.vimrc
```

这样 vim 就配置完成，效果图如图5-17所示。

```
168     normal o
169 endfunc
170
171 autocmd FileType c,cpp,java,go,php,javascript,puppet,python,rust,twig,xml,yml,perl autocmd B
172 fun! <SID>StripTrailingWhitespaces()
173     let l = line(".")
174     let c = col(".")
175     %s/\s\+$//e
176     call cursor(l, c)
177 endfun
178
179 " ===== key map =====
180
181 nnoremap k gk
182 nnoremap gk k
183 nnoremap j gj
184 nnoremap gj j
185
186 map <C-j> <C-W>j
187 map <C-k> <C-W>k
188 map <C-h> <C-W>h
189 map <C-l> <C-W>l
.vimrc
"-./.vimrc" 252L, 6560C
```

Figure 5-17 vim 配置后的效果图

最后，需要搭建 python 环境以及下载 UUGear 框架。操作如下：

```
pi@raspberrypi:~ $ mkdir IOT # 创建文件夹
pi@raspberrypi:~ $ cd IOT/ # 切换文件夹
pi@raspberrypi:~/IOT $ sudo apt-get install python-pip python-gevent python-dev #
需要安装的python组件
pi@raspberrypi:~/IOT $ sudo pip install virtualenv # 安装虚拟环境工具
pi@raspberrypi:~/IOT $ virtualenv venv # 创建虚拟环境
pi@raspberrypi:~/IOT $ source venv/bin/activate # 启用虚拟环境
(venv) pi@raspberrypi:~/IOT $ git clone https://github.com/uugear/UUGear.git # 下载
UUGear框架
```

这一部分就完成了。

#### 5.4.1.3 接口板制作软件的安装

由于使用到了 Arduino，而且 Arduino 是作为一个接口板的角色，所以需要在自己的 PC 上安装 Arduino IDE，可以从这个地址 <https://www.arduino.cc/en/Main/Software> 下

载 Arduino IDE。在 Linux 下，这个 IDE 是开箱即用的，解压即可，然后切换到解压目录，运行下面的命令即可启动：

```
lsyAndroid@iZ280p9hiuZ:~/IOT_Project/Arduino/ide/arduino-1.6.7$ ./arduino
```

启动后会出现图形界面，如图5-18所示。

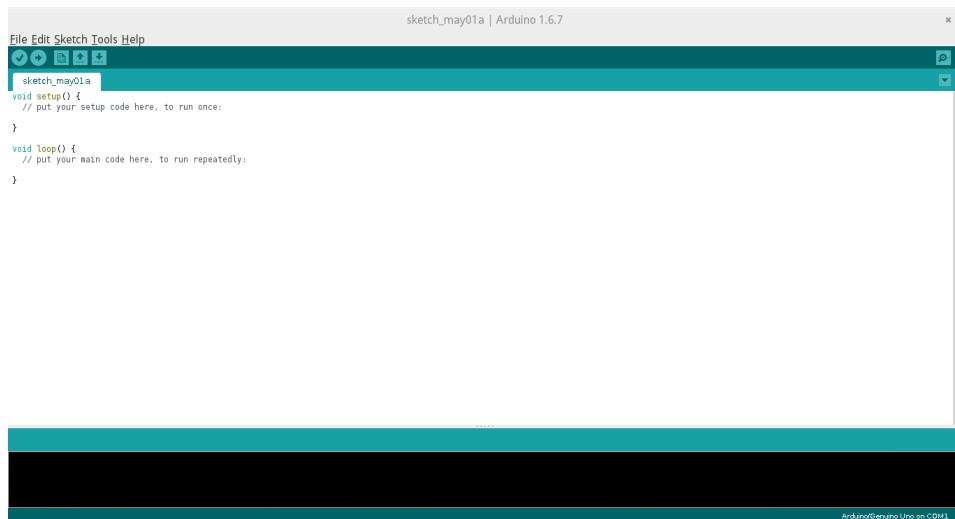


Figure 5-18 Arduino IDE 启动后的界面

## 5.4.2 树莓派端的代码编写

### 5.4.2.1 接口板的制作

在当前使用的电脑上也需要下载一份 UUGear 框架的代码，操作过程和在树莓派上的过程是一致的，然后切换到 UUGear 下的 Arduino 目录，用 Aduino IDE 打开这个名称为 UUGear.ino 的文件，如图5-19所示。

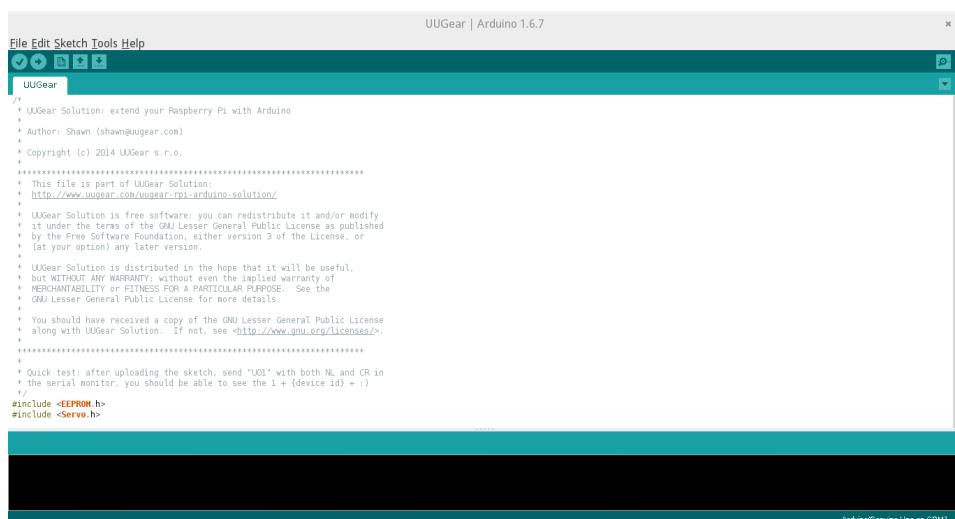


Figure 5-19 UUGear 框架的 Arduino 接口板代码

这时候需要把 Arudino UNO R3 通过 USB 数据线（也就是 Arduino leonardo 线）接入 PC，先进行两步设置，如图5-20，5-21所示。

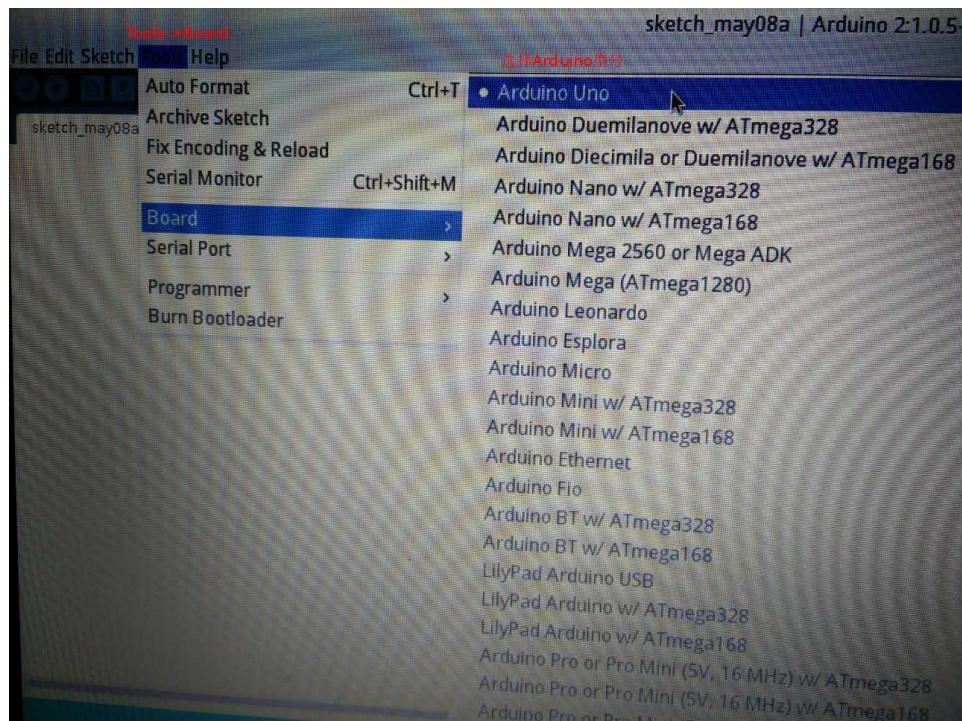


Figure 5-20 Arduino IDE 选择设备类型

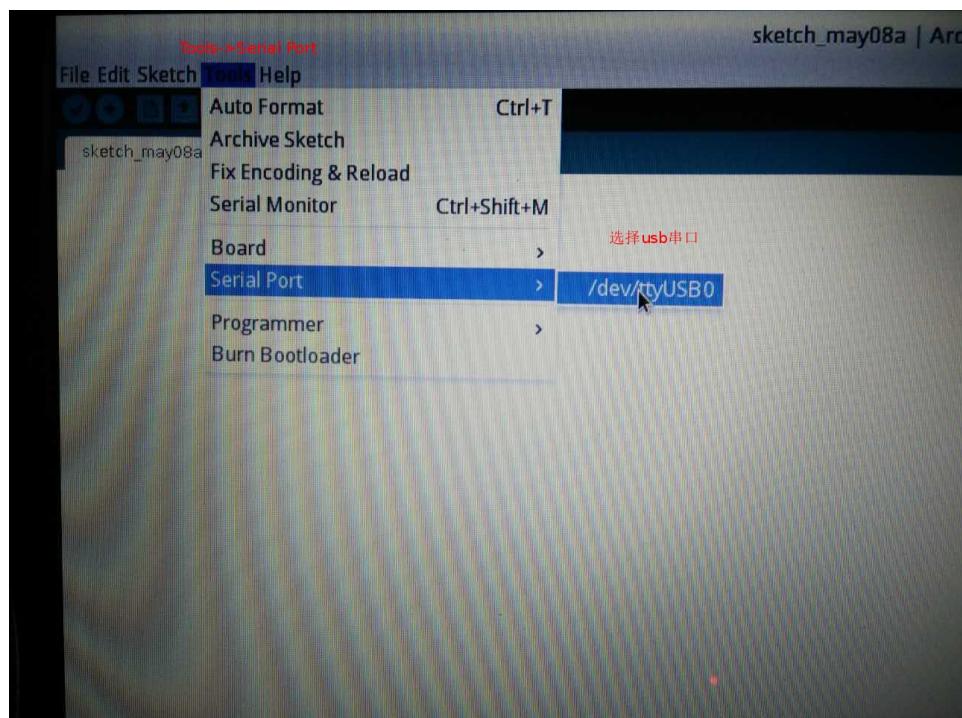


Figure 5-21 Arduino IDE 选择串行端口

选择完成后，进行代码的编译和烧录，编译的时候系统会监测代码是否符合规范，

编译没有问题后，进行烧录操作，操作过程如图5-22，5-23所示。

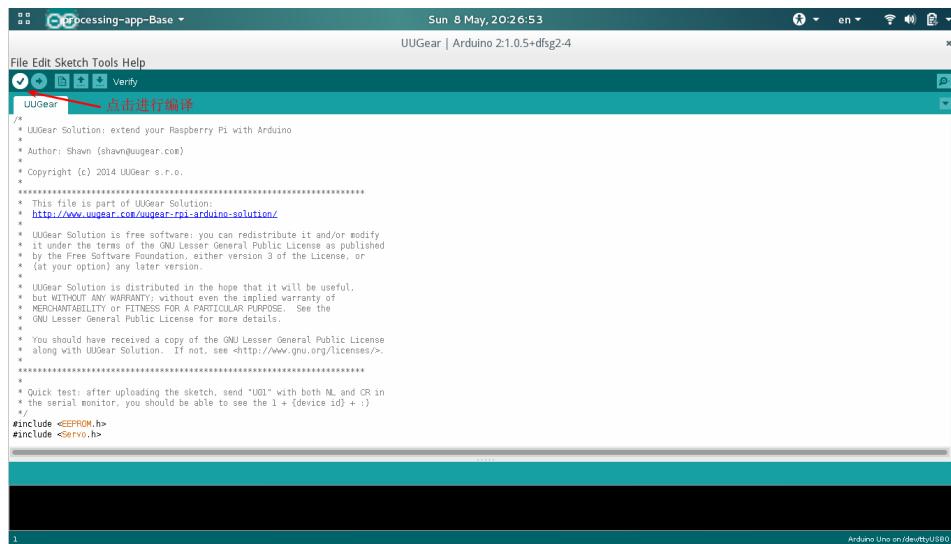


Figure 5-22 Arduino IDE 编译操作

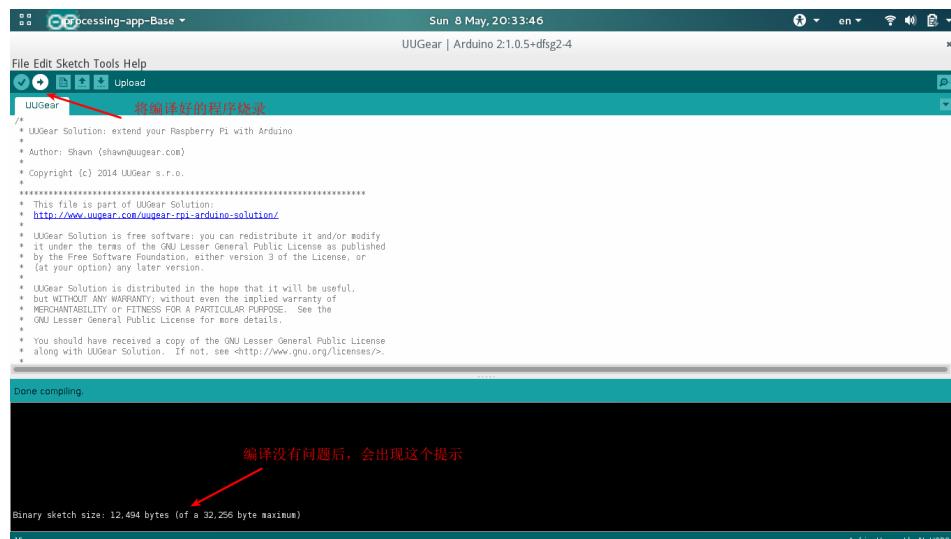


Figure 5-23 Arduino IDE 烧录操作

当烧录成功后也会从下面的控制台弹出如图5-23所示的提示信息，这样接口板就制作完毕了。

#### 5.4.2.2 UUGear 框架的使用之前

接口板做好后，需要将接口板通过 USB 数据线（也就是 Arduino leonardo 线）接入树莓派。

在编写正式的程序之前，需要在自己的 PC 上通过 ssh 连接到树莓派，并且需要将 UUGear 的源代码下载到树莓派上。操作如下：

```
pi@raspberry:~/IOT_Project/examples$ git clone https://github.com/uugear/UUGear.git
# 如果无法执行, 请使用root权限执行, 即在命令开始的时候添加sudo
```

当源代码准备好以后, 需要验证一下接口板是否能够正常工作, 操作过程如下:

```
pi@raspberry:~/IOT_Project/examples$ cd ./UUGear/Raspberry/bin
pi@raspberry:~/IOT_Project/examples/UUGear/Raspberry/bin$ ./lsuu  # 执行这个命令,
来查找接口板

UUGear-Arduino-5162-9034  (/dev/ttyUSB0) # 这样就说明设备已经找到了, 而且正常可用
-----
1 device(s) found.
pi@raspberrypi:~/IOT-Project/examples/UUGear/RaspberryPi/bin$
```

如果接口板找不到而且发生错误, 一般会有两种情况:

1、程序的不正常退出。当程序执行出现问题时, 例如命令行打印出了异常信息, 或者执行 **ctrl+c** 结束程序时, 无法正常结束, 这样的情况下再去执行 **./lsuu** 命令, 会发生找不到接口板的问题, 这时先试试 Arduino 上的 RESET 键操作, 然后进行连接, 如果不行的话, 就需要断开 usb 连接线, 再重新插上, 这时再执行命令, 就可以找到接口板了。

2、接口板制作过程中发生异常, 这时需要重新制作接口板。

然后需要做的就是记住这个接口板的编号, 由于开头的名称是一致的, 都是“UUGear-Arduino-”, 所以只需要记住后边的 8 位数字即可, 后边需要根据这个编号来找到 Arduino 设备。

最后需要强调的是, 在编写代码的过程中, 要遵循“高内聚低耦合”的原则, 单个函数的功能尽量单一, 组合的时候尽量灵活可控制。

#### 5.4.2.3 UUGear 框架使用 (1) —— 读取温湿度信息

这里要实现这样一个功能, 通过函数读取温湿度信息, 并将温湿度信息以字典的形式返回。这里先在控制台以日志的形式将温湿度信息输出。这段代码对应的接线图如图5-4, 原理图如图5-5。代码编写如下:

```
#!/usr/bin/python3
# encoding: utf-8

# 文件名称: GetTempDamp.py

# 导入相关包
# 注意: 代码需要在UUGear/RaspberryPi/bin目录下才能正常运行, 因为需要从这个目录下导入
#       UUGear框架
from time import sleep
from UUGear import *
from time import gmtime, strftime
```

```

UUGearDevice.setShowLogs(1) # 显示日志级别

device = UUGearDevice('UUGear-Arduino-5162-9034') # 初始化设备

# 定义读取信息的函数，传入获取的设备和引脚参数
def getTempDamp(device_get, port):
    data = device_get.readDHT(port) # 框架中自带的方法，读取DHT11传感器的数据
    humidity = (data >> 16) / 10
    if humidity > 100:
        humidity = (data >> 24)
    temperature = (data & 32767) / 10
    if temperature > 125:
        temperature = ((data & 32512) >> 8)
    if (data & 32768):
        temperature = -temperature
    if temperature != -127 and humidity != -1:
        return {"Temp":temperature,"Damp":humidity} # 正常数据
    else:
        return {"Temp":1111,"Damp":1111} # 异常数据

# 编写测试程序
if device.isValid(): # 当设备真正可用时执行
    try:
        while(True):
            temp_damp = getTempDamp(device,4)
            print("Temp = %s",temp_damp['Temp'],"Damp = ",temp_damp['Damp'])
            sleep(1)
    except KeyboardInterrupt: # 捕获在键盘上使用ctrl+c终止程序时带来的异常
        pass
    device.detach()
    device.stopDaemon()

else:
    print('device is not ready!')

```

这时在命令行中执行下面命令即可运行该程序并查看运行结果：

```

pi@raspberrypi:~/IOT-Project/examples/UUGear/RaspberryPi/bin$ python GetTempDamp.py
Client queue name=/uugear_response_queue_1
Client queue name=/uugear_response_queue_1
UUGear device found. fd=8
(Temp =', 20, 'Damp = ', 21)
(Temp =', 20, 'Damp = ', 21)
(Temp =', 21, 'Damp = ', 20)
(Temp =', 21, 'Damp = ', 20)
(Temp =', 20, 'Damp = ', 21)
(Temp =', 1111, 'Damp = ', 1111) # 异常数据也会显示出来
(Temp =', 21, 'Damp = ', 20)
(Temp =', 1111, 'Damp = ', 1111)
^Cpi@raspberrypi:~/IOT-Project/examples/UUGear/RaspberryPi/bin$

```

#### 5.4.2.4 UUGear 框架使用 (2) —— 控制继电器开闭

这里需要实现的功能是，设置继电器打开或者关闭的状态，设置完成后要获取这个状态，保证继电器设置的状态与获取的状态一致，确保是自己真正设置的状态。这段代码对应的接线图如图5-7，原理图如图5-8。代码如下：

```
#!/usr/bin/env python
# encoding: utf-8

# 文件名称: SetPin.py

from UUGear import *
from time import sleep

UUGearDevice.setShowLogs(1) # 设置日志级别

device = UUGearDevice('UUGear-Arduino-5162-9034') # 获取设备

device.setPinModeAsOutput(2) # 设置引脚为输出电平端

device.setPinHigh(2) # 初始化引脚电平为高电平

# 定义读取信息的函数，传入引脚编号，高低电平的boolean值
def setPinHighOrLow(device,pinNum,highOrLow):
    if highOrLow:
        device.setPinHigh(pinNum)
    else:
        device.setPinLow(pinNum)
    status = device.getPinStatus(pinNum) # 获取当前引脚的状态，判断引脚电平设置是否生效
    return status

# 测试程序
if device.isValid():
    try:
        while(True):
            setPinHighOrLow(device,2,True)
            sleep(1)
            print("status=====",status)
            setPinHighOrLow(device,2,False)
            sleep(1)
            print("status=====",status)
    except KeyboardInterrupt:
        pass
    device.detach()
    device.stopDaemon()

else:
    print('device is not ready!')
```

这时执行如下命令并查看运行结果如下：

```
pi@raspberrypi:~/IOT-Project/examples/UUGear/RaspberryPi/bin$ python SetPin.py
Client queue name=/uugear_response_queue_1
Client queue name=/uugear_response_queue_1
UUGear device found. fd=8
('status1=====', 1) # 引脚处于高电平状态
```

```
('status2====', 0) # 引脚处于低电平状态
('status1=====', 1)
('status2====', 0)
('status1=====', 1)
('status2====', 0)
('status1=====', 1)
('status2====', 0)
^Cpi@raspberrypi:~/IOT-Project/examples/UUGear/RaspberryPi/bin$
```

在程序执行过程中，会看到 LED 有规律的亮灭，时间间隔为 1s。

实际上到这里，针对硬件部分的编程就基本完成了，下面需要针对服务端进行收发数据的编程。

### 5.4.3 与服务端交互的代码编写

无论是在树莓派上，还是在 android 终端，均使用 paho 框架同服务端进行通信，在 android app 上，通过导入 paho 框架的 jar 包来实现引用，而在树莓派上，需要通过 pip 来安装，过程如下：

```
pi@raspberrypi:~/IOT-Project/examples/UUGear/RaspberryPi/bin$ pip install paho-mqtt
```

安装完成之后就可以在 python 代码的编写过程中引入。

#### 5.4.3.1 初始化客户端

初始化客户端只需要一句代码即可：

```
import paho.mqtt.client as mqtt # 导入相应包
mqttc = mqtt.Client()
```

但是这样是不够的，原因是客户端初始化的时候，默认使用的是 MQTTv3.1 协议，现行服务端的协议为 MQTTv3.1.1，这样造成了无法对接的现象，存在较大偏差，在调试的过程中也是吃了不少苦头，所以这里必须指定协议版本，写法如下：

```
mqttc = mqtt.Client(protocol=mqtt.MQTTv311) # 指定MQTT协议版本
```

#### 5.4.3.2 发布消息（publish）

发布消息也很简单，只需要一句代码就可以实现：

```
import paho.mqtt.publish as publish # 导入相应包
publish.single("paho/test/single", "payload", hostname="iot.eclipse.org") #
发送一条消息

msgs = [ {'topic': "paho/test/multiple", 'payload': "multiple 1"}, ("paho/test/multiple", "multiple 2", 0, False)]
```

```
publish.multiple(msgs, hostname="iot.eclipse.org") # 发送多条消息
```

解释一下，publish.single() 方法中第一个参数是发送主题（topic），第二个参数是消息类型，第三个参数是服务端地址；publish.multiple() 方法中，第一个参数是消息集合，第二个参数是服务端地址。

这里使用的是 publish.single() 方法，将信息一条一条按照顺序发出去。

#### 5.4.3.3 订阅消息（subscribe）

订阅消息的编写相对来讲比较麻烦，需要实现相应的回调函数，才能使订阅起作用，代码示例如下：

```
import sys
try:
    import paho.mqtt.client as mqtt
except ImportError:
    # 导入包异常时的处理
    import os
    import inspect
    cmd_subfolder = os.path.realpath(os.path.abspath(os.path.join(os.path.split(
        inspect.getfile( inspect.currentframe() ))[0], "../src")))
    if cmd_subfolder not in sys.path:
        sys.path.insert(0, cmd_subfolder)
    import paho.mqtt.client as mqtt

# 实现四个回调方法
def on_connect(mqttc, obj, flags, rc):
    print "Connected to %s:%s" % (mqttc._host, mqttc._port)

def on_message(mqttc, obj, msg):
    # 当收到订阅消息时，在这个回调方法中进行进一步处理
    print(msg.topic+" "+str(msg.qos)+" "+str(msg.payload))

def on_publish(mqttc, obj, mid):
    print("mid: "+str(mid))

def on_subscribe(mqttc, obj, mid, granted_qos):
    print("Subscribed: "+str(mid)+" "+str(granted_qos))

# 输出日志的回调方法
def on_log(mqttc, obj, level, string):
    print(string)

# 指定回调方法以及初始化订阅信息
mqttc = mqtt.Client(protocol = mqtt.MQTTv311)
mqttc.on_message = on_message
mqttc.on_connect = on_connect
mqttc.on_publish = on_publish
mqttc.on_subscribe = on_subscribe
# 显示日志信息
mqttc.on_log = on_log
# 连接服务端并执行订阅
mqttc.connect_srv("mosquitto.org", 60)
```

```

mqttc.subscribe("$SYS/broker/version", 0)

# 当收到消息时，自动打印消息；未收到消息时，会阻塞线程
rc = 0
while rc == 0:
    rc = mqttc.loop()

print("rc: "+str(rc))

```

这样去编写是不太好的，如果单纯就是测试一下是可以的，但是在正式环境中，需要处理的内容不止有订阅消息以及收到订阅消息后根据接口文档进行后续操作，还需要发送消息，因此下面会先对这个类进行下封装，在正式环境中会引入多线程来进行处理。

#### 5.4.3.4 完整封装

这里对整个与服务端交互的部分进行封装，代码如下：

```

#!/usr/bin/env python
# encoding: utf-8

import paho.mqtt.client as mqtt
import paho.mqtt.publish as publish

class MainMQTTClass:
    def __init__(self, clientid=None, protocol= mqtt.MQTTv311):
        self._mqttc = mqtt.Client(clientid, protocol)
        self._mqttc.on_message = self.mqtt_on_message
        self._mqttc.on_connect = self.mqtt_on_connect
        self._mqttc.on_publish = self.mqtt_on_publish
        self._mqttc.on_subscribe = self.mqtt_on_subscribe

    def mqtt_on_connect(self, mqttc, obj, flags, rc):
        print("rc: "+str(rc))

    def mqtt_on_message(self, mqttc, obj, msg):
        # 从这里处理接收的消息，并操控电平高低
        print(msg.topic+" "+str(msg.qos)+" "+str(msg.payload))
    def mqtt_on_publish(self, mqttc, obj, mid):
        print("mid: "+str(mid))

    def mqtt_on_subscribe(self, mqttc, obj, mid, granted_qos):
        print("Subscribed: "+str(mid)+" "+str(granted_qos))

    def mqtt_on_log(self, mqttc, obj, level, string):
        print(string)
    # 订阅消息的方法
    def subscribe_all(self, address, port = 1883, keepalive = 60, topic, qos = 0):
        self._mqttc.connect(address, port, keepalive)
        self._mqttc.subscribe(topic, qos)
        # 阻塞线程，持续运行
        mqttc.loop_forever()
    # 发送消息的方法
    def publish_msg(self, topic, msg, address):
        publish(topic, msg, address)

```

#### 5.4.4 协议实现

这一部分需要把前面所编写的内容进行整合，引入线程的概念，让订阅消息的操作单独一个线程运行。得到消息后进行 Json 解析，并调用相应的硬件电路控制函数来执行解析后的指令。根据一开始制定的流程图和通信协议，来分功能进行编写，在最后给出完整的可运行的程序。

首先是温度数据采集的流程在树莓派上实现的部分，功能函数实现如下：

```
# 定义读取温度信息的函数，传入获取的设备和引脚参数
def getTemp(device,port):
    data = device.readDHT(port) # 框架中自带的方法，读取DHT11传感器的数据
    humidity = (data >> 16) / 10
    if humidity > 100:
        humidity = (data >> 24)
    temperature = (data & 32767) / 10
    if temperature > 125:
        temperature = ((data & 32512) >> 8)
    if (data & 32768):
        temperature = -temperature
    if temperature != -127:
        return {"temp":temperature} # 正常数据

def publishTempData():
    temp = readTemp(device,port)
    # 字典合并
    jsonData = {"status":0}
    jsonData.update(temp)
    print(jsonData)
    # 字典转Json
    clientData = json.dumps(jsonData)
    publish.single("temperature",clientData,hostname,protocol)
```

然后是继电器控制流程在树莓派上实现的部分，功能函数实现如下：

```
# 定义读取信息的函数，传入引脚编号，高低电平的boolean值
def setPinHighOrLow(device,pinNum,highOrLow):
    if highOrLow:
        device.setPinHigh(pinNum)
    else:
        device.setPinLow(pinNum)
    sleep(1) # 延时 1 秒
    status = device.getPinStatus(pinNum) # 获取当前引脚的状态，判断引脚电平设置是否生效
    return status

# 订阅消息的操作执行后，获取消息的回调函数
def onMessage(mqtte,obj,msg):
    print("订阅的消息为: "+msg.topic+" "+str(msg.qos)+" "+str(msg.payload))
    if msg.topic == "switch" and str(msg.qos) == "2":
        # 解析获取的信息
        switchData = str(msg.payload)
        jsonData = json.loads(switchData)
        # 执行对引脚的操作
        status = setPinHighOrLow(device,jsonData['port'],jsonData['isOn'])
```

```

# 拼接内容
returnData = {"status":0,"port":jsonData['port'],"status_isOn":status}
# 返回执行的信息
publish.single("switch_status",json.dumps(returnData),hostname,protocol,qos =
2)
# 后续执行订阅操作

```

最后是策略实现的部分，代码如下：

```

def publishTempLegency(device, portTemp, portSwitch):
    temp = readTemp(device, port)
    jsonDataBasic = {"status":100}
    jsonDataLowTemp = {"msg_temp":"温度低于24摄氏度，是否关闭空调？"}
    jsonDataHighTemp = {"msg_temp":"温度高于30摄氏度，是否打开空调？"}
    if temp['temp'] < 24:
        if device.getPinStatus(portSwitch) == 1: # 空调已经打开
            jsonLow = jsonDataBasic.update(jsonDataLowTemp)
            publish.single("temp_notification",json.dumps(jsonLow),hostname,protocol,
qos = 1)
    else if temp['temp'] > 30:
        if device.getPinStatus(portSwitch) == 0: # 空调已经关闭
            jsonHigh = jsonDataBasic.update(jsonDataHighTemp)
            publish.single("temp_notification",json.dumps(jsonHigh),hostname,protocol,
qos = 1)
    else:
        print("正常状态。。。")

```

整个树莓派端的代码实现，放在附录一上。

## 5.5 Android 端代码的编写

### 5.5.1 引入 Paho 框架的 jar 包

新建工程，命名为 IOT，创建完成之后目录结构如下：

项目创建完成后，需要做的第一件事情就是引入 paho 框架，可以直接导入 jar 包，还有更简单的办法就是在 build.gradle 文件中引入。注意这里的 build.gradle 文件是属于 Module 的，也就是你在图5-24看到的后边括号中注明了 Module:app 的文件，需要添加的内容如下：

```

# build.gradle(Module:app)

apply plugin: 'com.android.application'

android {
    .....
}

# 在dependencies节点下添加下面所示的代码
dependencies {
    .....
    // 这里是引入paho框架的相关jar包的代码
    compile (group: 'org.eclipse.paho', name: 'org.eclipse.paho.android.service',

```

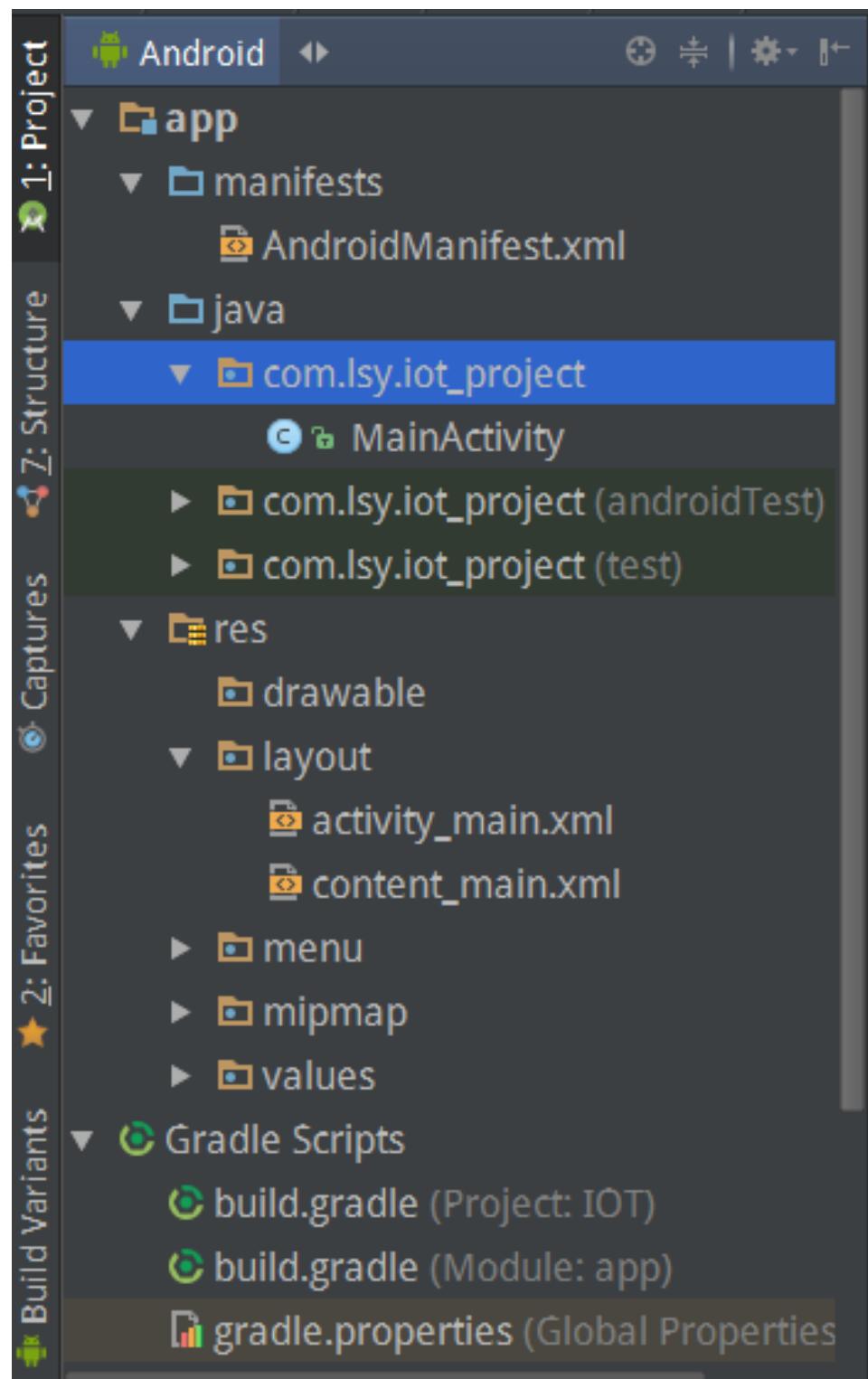


Figure 5-24 创建项目后的工程目录

```

version: '1.0.2') {
    exclude module: 'support-v4'
}

```

引入 jar 包完成之后，需要编译一下程序，在 AndroidStudio 界面的右上方会出现

Sync Now 的提示，点击即可。这样编译器 Gradle 就是自动去仓库寻找需要的 jar 文件，并下载到项目中，在编写代码的时候才能真正引用 jar 包中的代码。

### 5.5.2 界面布局的编写

这里主要分三部分，一个是温度数据监测部分，第二个是开关控制部分，第三个是策略通知部分。在这三个部分中，策略通知部分可以不显示在界面上，而是通过通知栏的方式来显示，在通知栏提供相应的操作继电器打开和关闭的按钮，所以主界面代码拆分为两部分，下面是数据监测部分的代码：

```
<!-- 温度数据展示 -->
<android.support.v7.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    app:cardCornerRadius="8dp"
    app:cardElevation="8dp"
    app:contentPadding="15dp"
    >
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="5dp"
        >

        <ImageView
            android:id="@+id/t_img_tag"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:scaleType="centerCrop"
            android:src="@mipmap/ic_launcher"
            />

        <TextView
            android:id="@+id/t_temp_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignBottom="@+id/t_img_tag"
            android:layout_alignParentRight="true"
            android:layout_alignTop="@+id/t_img_tag"
            android:layout_toRightOf="@+id/t_img_tag"
            android:gravity="center"
            android:text="温度监控"
            android:textSize="30dp"
            />

        <TextView
            android:id="@+id/t_temp_int_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_below="@+id/t_img_tag"
            />
    
```

```

        android:layout_toLeftOf="@+id/t_temp_tag_du"
        android:layout_toStartOf="@+id/t_temp_tag_du"
        android:gravity="center"
        android:textSize="30dp"/>

<TextView
    android:id="@+id/t_temp_tag_du"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignEnd="@+id/t_temp_title"
    android:layout_alignRight="@+id/t_temp_title"
    android:layout_alignTop="@+id/t_temp_int_view"
    android:gravity="center"
    android:text="C"
    android:textSize="30dp"/>

</RelativeLayout>

</android.support.v7.widget.CardView>

```

继电器控制部分代码如下：

```

<!-- 继电器控制模块 -->
<android.support.v7.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    app:cardCornerRadius="8dp"
    app:cardElevation="8dp"
    app:contentPadding="15dp"
    >
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="5dp"
        >

        <ImageView
            android:id="@+id/t_img_tag_s"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:scaleType="centerCrop"
            android:src="@mipmap/ic_launcher"
            />

        <TextView
            android:id="@+id/t_temp_title_s"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignBottom="@+id/t_img_tag_s"
            android:layout_alignParentRight="true"
            android:layout_alignTop="@+id/t_img_tag_s"
            android:layout_toRightOf="@+id/t_img_tag_s"
            android:gravity="center"
            android:text="开关控制"
            />
    
```

```

        android:textSize="30dp"
    />

<android.support.v7.widget.SwitchCompat
    android:id="@+id/relay_switch_4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="继电器"
    android:checked="false"
    android:paddingTop="16dp"
    android:layout_centerVertical="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_alignLeft="@+id/t_img_tag_s"
    android:layout_alignStart="@+id/t_img_tag_s" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="继电器状态"
    android:id="@+id/tv_title"
    android:paddingBottom="4dp"
    android:layout_alignParentBottom="true"
    android:layout_alignLeft="@+id/relay_switch_4"
    android:layout_alignStart="@+id/relay_switch_4" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="关闭"
    android:paddingBottom="4dp"
    android:id="@+id/tv_status"
    android:layout_alignTop="@+id/tv_title"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

</RelativeLayout>

</android.support.v7.widget.CardView>

```

通知部分需要从代码中编写，在下一部分中介绍。

### 5.5.3 业务逻辑编写

在引入框架之后，需要进一步封装相应代码，这里使用单例模式封装了 PahoConnection，代码如下：

```

public class PahoConnection implements ConnectionInterface {
    .....
    private PahoConnection(Context context) {
        // 从数据库中获取，如果获取不到就创建
        //RealmHelper helper = new RealmHelper(context);
        PahoClientCallback callback = new PahoClientCallback(context);
        mqttConnectOpts = new MqttConnectOptions();
        mqttConnectOpts.setConnectionTimeout(60);
        //mqttConnectOpts.setKeepAliveInterval(60);
        clientCallback = new PahoClientConnectionCallback(context);
    }
}

```

```

        mqttAndroidClient = addNewClient(context);
        // 设置回调方法,处理连接断开和消息接收的问题
        mqttAndroidClient.setCallback(callback);
    }

    public static PahoConnection getInstance(Context context) {
        if (pahoConnection == null) {
            pahoConnection = new PahoConnection(context);
        }
        return pahoConnection;
    }
}

```

然后是下面两个核心函数，发布(publish)消息的函数以及订阅(subscribe)消息的函数，代码如下：

```

/**
 * 发布函数
 */
public void publish(@NonNull String topic, @NonNull String message, @NonNull int qos,
@NonNull boolean retained) {
    try {
        Log.d("TAG", "topic == " + topic + "====message==== " + message + " ===qos
===== " + qos);
        mqttAndroidClient.publish(topic, message.getBytes(), qos, retained);
        Log.d("TAG", "published");
    } catch (MqttSecurityException e) {
        Log.e(this.getClass().getCanonicalName(), "Failed to publish a messged
from the client with the handle ", e);
    } catch (MqttException e) {
        e.printStackTrace();
        Log.e(this.getClass().getCanonicalName(), "Failed to publish a messged
from the client with the handle ", e);
    } finally {
        Log.d("TAG", "publish finished!");
    }
}

/**
 * 订阅函数
 */
public void subscribe(@NonNull String topic, @NonNull int qos) {
    if (mqttAndroidClient == null) {
        Log.d("TAG", "Client未初始化!");
        return;
    }
    try {
        mqttAndroidClient.subscribe(topic, qos);
        Log.d("TAG", "subscribed!");
    } catch (MqttException e) {
        e.printStackTrace();
    } finally {
        Log.d("TAG", "subscribe finished!");
    }
}

```

当订阅函数执行后，需要接收所有服务端转发到客户端的消息，获取到消息以后，通过广播的方式来将消息发送到 Activity 中来更新界面内容。代码如下：

```
public class PahoClientCallback implements MqttCallback {
    .....

    @Override
    public void messageArrived(String topic, MqttMessage message) throws Exception {
        String messageContent = new String(message.getPayload());
        // 接收消息并进行数据处理与解析
        Log.d("TTTT", "topic == " + topic + "message == " + messageContent);

        Intent intent = new Intent("com.iot.lsy.iot_android.data_received");
        intent.putExtra("topic", topic);
        intent.putExtra("message", messageContent);
        context.sendBroadcast(intent);
    }

    .....
}

/**
 * 广播接收者，更新界面
 */
public class DataBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String topic = intent.getStringExtra("topic");
        String message = intent.getStringExtra("message");
        Log.d("TAG", " 获取的消息内容为::::"+topic+::::"+message);
        // 更新界面信息
        dataMonitorListener.onDataReceived(topic, message);
    }
}
```

最后要在 Activity 中初始化相关内容使其进行工作，代码如下：

```
public class MainActivity extends AppCompatActivity {

    private IntentFilter filter;
    private DataBroadcastReceiver receiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        .....

        // 创建线程，连接服务器并订阅消息
        PahoConnection connection = PahoConnection.getInstance(getApplicationContext());
        MqttConnectOptions options = connection.getMqttConnectOpts();
        connection.connect(options);

        // 注册广播接收
        filter = new IntentFilter();
        filter.addAction("com.iot.lsy.iot_android.data_received");
        receiver = new DataBroadcastReceiver();
        registerReceiver(receiver, filter);
    }
}
```

```
        Log.d("TAG","注册广播接收者！");  
    }  
}
```

---

这样所有的代码就编写完成了！

## 第六章 结论

这篇论文详细阐述了利用树莓派和 Arduino 来搭建物联网系统的过程，说明了在搭建过程中的详细操作，力求将各个工作环节描述清楚，让后续对物联网方面有兴趣的同学能在这篇论文的引导下，搭建自己的物联网系统，并发散思维，培养结合多种编程语言的开发能力，能够为深入学习做一个入门引导，提升自身整合资源、利用资源的能力。

## 第七章 参考文献

- [1] 关庆余, 李鸿彬, 于波 MQTT 协议在 Android 平台上的研究与应用 《计算机系统应用》 2014 年 04 期
- [2] 顾亚文 基于 MQTT 协议的通用智能家居系统设计与实现 《西安电子科技大学》 2014 年
- [3] 温彬民 一种基于自适应心跳机制的 MQTT 通信协议的研究与应用 《华南理工大学》 2015 年
- [4] Urs Hunkeler, Hong Linh Truong, Andy Stanford-Clark MQTT-S —A publish/subscribe protocol for Wireless Sensor Networks 《Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on》 6-10 Jan. 2008
- [5] Ian Warren Andrew Meads ; Satish Srirama ; Thiranjith Weerasinghe ; Carlos Paniagua Push Notification Mechanisms for Pervasive Smartphone Applications 《IEEE Pervasive Computing (Volume:13 , Issue: 2 )》 Apr.-June. 2014
- [6] 盖荣丽, 钱玉磊, 李鸿彬, 贾军营 基于 MQTT 的企业消息推送系统 《计算机系统应用》 2015 年 11 期
- [7] 何宏辉 Android 开发进阶 -从小工到专家人民邮电出版社
- [8] Bill Lubanovic Introducing Python 人民邮电出版社
- [9] Matteo Collina Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST 《ARCES (Advanced Research Centre for Electronic Systems)》 9 - 12 Sept. 2012
- [10] Seong-Min Kim, Hoan-Suk Choi, Woo-Seop Rhee IoT home gateway for auto-configuration and management of MQTT devices 《Wireless Sensors (ICWiSe), 2015 IEEE Conference on》 12-17 24-26 Aug. 2015
- [11] 姚丹, 谢雪松, 杨建军 基于 MQTT 协议的物联网通信系统的研究与实现 《信息通信》 2016 年第 3 期 33-35 页
- [12] 张亚慧 物联网环境下轻量级发布/订阅系统的设计与实现 《北京邮电大学》 2015 年
- [13] 孔祥龙, 王燕 适用于低成本物联网终端的消息通讯协议比较研究 《无线互联科技》 2015 年第 16 期 49-52 页

[14] 张波, 杨国华 MQTT 发布 / 订阅消息机制在 Arduino 传感节点的实现 《电子世界》2013 年第 22 期 93-94 页

[15] 关庆余, 李鸿彬, 于波 MQTT 协议在 Android 平台上的研究与应用 《计算机系统应用》2014 年第 23 卷第 4 期 196-200 页

## 第八章 致谢及声明

首先感谢指导老师宿宝臣教授，从初期搭建系统到后期编写论文，宿老师给予了我最大限度的帮助，鼓励我发散思维的同时，也要用科学严谨的态度去思考和操作；然后感谢我的同学张树新、林振男，在搭建过程中给予我很多建议，提供了元器件支持和编写代码上的规范指导，令我获益良多；最后需要感谢整个一号实验楼 314 实验室，感谢每一个同学的支持，希望能将开源和探索的思想继续传递下去，影响更多的同学和老师！