第四章 基于树莓派的开发与集成

目录

本章导读

上一章实现了物联网的“前端”开发与集成。在所谓“前端”，已经可以实现基于 arduino 的传感器信息采集、并在数码管上显示采集的数据。这一章将继续介绍基于树莓派的开发与集成，下一章讨论基于物联网数据服务器的、以及自己开发基于“云”服务器的应用系统开发与集成。传感器为物联网应用获取了“原始”的数据，“前端”是基础，而这些数据的加工、处理与应用，则是物联网的“后端”，是核心和灵魂。

数据的加工和处理，最简单和最基本的形式是“显示”。本章第一小节的实验项目是开发一个基于树莓派平台的模拟用户显示与控制界面，它将从传感器获得的温度，用比较“美观”的形式显示出来，并设有几个按钮，根据温度变化，可以自动或手工“反向”控制 arduino 上的 LED，模拟操作被控设备的“开关”。进一步的数据操作是把温度数据送到 YeeLink 服务器上，这样，在你自己的手机上，只要下载一个 APP，就能够看到实时的温度变化曲线。而最后一节课，则需要你自己在“云服务器”上，开发一个物联网应用，并通过 IE 浏览器（开发相应的 iOS 或 android APP 后，也可以在手机客户端上），获得基于温度的应用。

随着应用层次的提高，系统的复杂性越来越高，开发与集成的难度也在提高，在项目开发过程中，同学们会慢慢体验到这个变化。

4.1 树莓派平台集成的需求

一个“传感器采集+数码管显示”的应用，实在是太“小儿科”了。只是“入门级”的实验，练练手而已，除了用来在家里显示“时钟/温度”这样的应用，实在没有什么更大的实用价值。数码管能够显示的内容，是十分有限的。如果希望显示更多的内容、有更美观的、图形化的用户界面，只能靠树莓派来承担了。

本节在上一节课的基础上，继续努力，把树莓派也集成到已有的“前端系统”中。为了在树莓派上进一步体验系统应用开发与集成过程，本节在树莓派平台上，再添加一个“用户界面”项目，来体验系统的开发与集成过程。

在前一章 3.4 节，讨论平台选择的时候，已经谈到：作为物联网应用的总体架构，arduino 是前端传感器采集平台，树莓派是中间的数据处理和网络传输平台。所以，按照这样的架构设计和分工，本节课为树莓派添加一个简单而适用的用户显示和控制界面，其中需要将树莓派与 arduino 连接起来，实现两者之间的数据交互。

本节的需求非常简单：在树莓派的显示器上，显示 DHT11 传感器采集到的温度数据，并可实现自动（设定）或手动的反向控制。即当温度达到某个设定值时，可以自动或手动让 arduino 上的 LED 灯点亮（模拟控制开关动作）。既然（目前）应用系统的最上层是树莓派平台，是标准的显示屏，而不是数码管，则用户界面希望采用图形界面、包括：温度显示、按钮等，都希望能够做得美观一点，而不是简单的命令行字符显示。

由于本实训课程只是物联网传感器数据采集的初步体验，因此，所有采集的数据，包括数据加工，都是最“原始”的。一个真正的应用，绝对不是这么简单的。例如：一个生命监护系统，通过生物传感器，获得人体的各种生理信息（脉搏、血压、呼吸、心电、脑电等），再把这些信息加工（过滤、抽取、放大，直接显示是没有任何意义的），用符合生命监护仪的数据要求的形式显示、报告，甚至传输到监控中心等。显然，实训项目暂时还不可能用这样的数据处理标准来要求。

4.2 选择树莓派上的 PyQt 界面框架

在树莓派上实现比较实用的用户界面 GUI 的工具有不少，如：python、QT、PyQt 等。PyQt 是 Python 下的一套图形界面接口库，顾名思义就是在 Python 中调用 Qt 图形库和组件（在本教程第七章的股票软件系统分析与二次开发中，还将更多地介绍 QT5）。使用 PyQt 的优点在于，可以使用 Qt 成熟的 IDE（如 Qt Creator）进行图形界面设计，并自动生成可执行的 Python 代码。

1）PyQt 的安装：

PyQt 可以通过 apt-get 命令安装，其对应 Python 2.x 和 Python 3.x 的包名称不同。在目前的树莓派上，同时预装了 python2.7.3 和 python3.2.3。

安装 Python 2.x 下的 PyQt：$ sudo apt-get install python-pyqt4 pyqt4-dev-tools；

安装 Python 3.x 下的 PyQt：$ sudo apt-get install python3-pyqt4 pyqt4-dev-tools。

获取 PyQt 的文档和范例程序（非必须，但是通过上一节课的体验，可以知道，有范例还是很有好处的）：

$ sudo apt-get install python-qt4-doc。

获取到的范例程序被存在/usr/share/doc/python-qt4-doc/examples 目录下。

2）Python3 环境下的 PyQt4 测试代码：

先跑一段简单的代码，测试一下 PyQt 的环境是否安装正确。选择 Python3 环境，新建如下的一个 Python 文件，命名为 hello_pyqt.py，代码如下：

```
import sys
from PyQt4 import QtCore, QtGui
class HelloPyQt(QtGui.QWidget):
    def __init__(self, parent = None):
        super(HelloPyQt, self).__init__(parent)
        self.setWindowTitle("PyQt Test")
        self.textHello = QtGui.QTextEdit("This is a test program written in python with PyQt lib!")
        self.btnPress = QtGui.QPushButton("Press me!")
            layout = QtGui.QVBoxLayout()
            layout.addWidget(self.textHello)
            layout.addWidget(self.btnPress)
            self.setLayout(layout)
```

```
            self.btnPress.clicked.connect(self.btnPress_Clicked)
        def btnPress_Clicked(self):
            self.textHello.setText("Hello PyQt!\nThe button has been pressed.")
if __name__=='__main__':
        app = QtGui.QApplication(sys.argv)
        mainWindow = HelloPyQt()
        mainWindow.show()
        sys.exit(app.exec_())
```

将此 hello_pyqt.py 文件，复制到树莓派的目录下，打开 IDLE3，装入 hello_pyqt.py，选择 Run|Run Module，就可以看见如图 4-1 的运行结果了。

图 4-1 hello_pyqt.py 运行结果

在本例中，所有的 PyQt 控件都封装在 HelloPyQt 类中。程序首先添加了一个 QTextEdit 控件 textHello 和 QPushButton 控件 btnPress，然后通过 self.btnPress.clicked.connect()语句将 btnPress 按钮的 clicked 信号连接至 btnPress_Clicked()函数（QT 信号与曹的关联机制是 QT 的基本运行方式，更多细节，将在第七章介绍）。

当按钮被按下时，会触发 clicked 事件，进而调用 btnPress_Clicked()函数。该函数的功能就是改变 textHello 中的文本。

3）QT 的 Qt Creator：

Qt 的最大好处，就是可以可视化地设计和编辑用户界面，并自动生成相应的代码，否则就没有必要使用 Qt。从上面的例子可以看出，如果要手动编写代码调用 PyQt，显然是十分不便的。QT 的好处是可以在 windows 下使用 Qt Creator 完成界面设计（类似交叉编译），然后，使用 PyQt 将 Qt Creator 生成的.ui 文件（用户界面）直接转换成 Python 代码的功能（在 windows 环境下没有这个必要）。

如果需要对 test.ui 进行转换，其命令如下：$ pyuic4 test.ui -x -o test.py。其中-x 参数相当于--execute，在代码中增加了一些测试语句，这样生成的 Python 文件就可以直接执行了。

4.3 显示树莓派 CPU 温度的界面

上一节课已经可以实现通过 DHT11 温度传感器，获得实时的温度信息。现在的任务是，在树莓派的屏幕上，实现一个比较“美观”的温度控制器界面。能够实时显示树莓派 CPU 的温度（树莓派暂时还没有与 arduino 通讯，先取自己的温度），设定预警温度，当温度接近预警温度时，可以改变显示的颜色，当超过预警温度时，可以报警。

首先，打开 Qt Creator，新建一个 Qt GUI 应用程序工程（可以在树莓派本地安装运行 Qt Creator，也可以用 windows 下的 Qt Creator，设计 Qt GUI 程序。然后，再如前所述，通

过转换 ui 文件为 Python 代码的方式）。UI 设计的窗口界面如图 4-2：

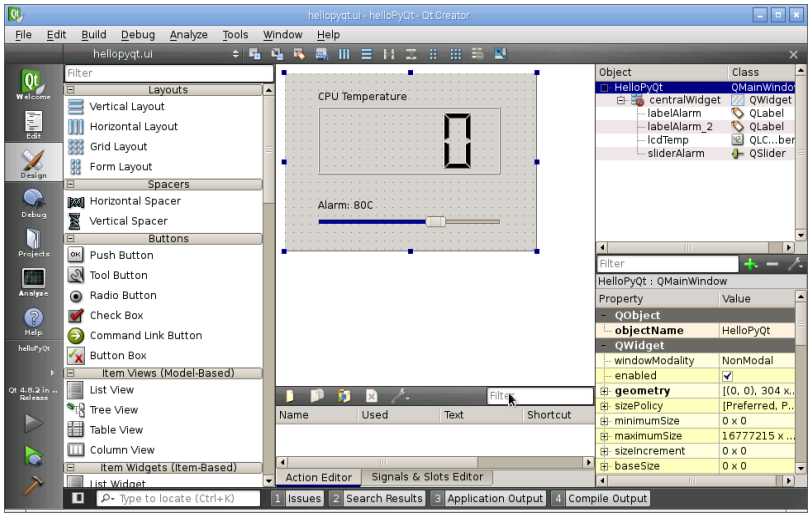

图 4-2 Qt Creator 的设计界面

通过 pyuic 命令将 GUI 文件转换成 Python 文件，然后在此基础上进行修改。

Qt 界面设计不单是一个图形界面的设计，而是创建了一个类似微软 MFC 的基于窗口的应用框架。其运行机制是基于控件的事件（消息）与代码（槽）之间的关联、互动机制。这个机制是在微软的 MFC 消息机制基础上、进一步扩展、发展起来的。而在控件的数量、种类、抽象度等方面，在“消息与槽”的对应与关联机制方面，Qt 都胜于 MFC。

实现树莓派 CPU 温度显示和控制界面的代码如下：

```
from PyQt4 import QtCore, QtGui
import os
try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    _fromUtf8 = lambda s: s
class Ui_HelloPyQt(object):
    def setupUi(self, HelloPyQt):
        HelloPyQt.setObjectName(_fromUtf8("HelloPyQt"))
        HelloPyQt.resize(304, 212)
        self.centralWidget = QtGui.QWidget(HelloPyQt)
        self.centralWidget.setObjectName(_fromUtf8("centralWidget"))
        self.lcdTemp = QtGui.QLCDNumber(self.centralWidget)
        self.lcdTemp.setGeometry(QtCore.QRect(40, 40, 221, 81))
        self.lcdTemp.setSmallDecimalPoint(False)
        self.lcdTemp.setDigitCount(6)
        self.lcdTemp.setObjectName(_fromUtf8("lcdTemp"))
        self.sliderAlarm = QtGui.QSlider(self.centralWidget)
        self.sliderAlarm.setGeometry(QtCore.QRect(40, 170, 221, 16))
        self.sliderAlarm.setMaximum(120)
        self.sliderAlarm.setProperty("value", 80)
        self.sliderAlarm.setOrientation(QtCore.Qt.Horizontal)
        self.sliderAlarm.setTickPosition(QtGui.QSlider.NoTicks)
```

```
        self.sliderAlarm.setObjectName(_fromUtf8("sliderAlarm"))
        self.labelAlarm = QtGui.QLabel(self.centralWidget)
        self.labelAlarm.setGeometry(QtCore.QRect(40, 150, 221, 16))
        self.labelAlarm.setObjectName(_fromUtf8("labelAlarm"))
        self.labelTemp = QtGui.QLabel(self.centralWidget)
        self.labelTemp.setGeometry(QtCore.QRect(40, 20, 221, 16))
        self.labelTemp.setObjectName(_fromUtf8("labelTemp"))
        #Add timer
        self.timerTemp = QtCore.QTimer(self.centralWidget)
        HelloPyQt.setCentralWidget(self.centralWidget)
        # Add slots
        self.sliderAlarm.valueChanged.connect(self.sliderAlarm_ValueChanged)
        self.timerTemp.timeout.connect(self.timerTemp_TimeOut)
        # Use the timeout event to initialize the LCD
        self.timerTemp_TimeOut()
        # Start timer, time out per 2 seconds
        self.timerTemp.start(2000)
        self.retranslateUi(HelloPyQt)
        QtCore.QMetaObject.connectSlotsByName(HelloPyQt)
    def retranslateUi(self, HelloPyQt):
        HelloPyQt.setWindowTitle(QtGui.QApplication.translate("HelloPyQt", "HelloPyQt",
None, QtGui.QApplication.UnicodeUTF8))
        self.labelAlarm.setText(QtGui.QApplication.translate("HelloPyQt", "Alarm: 80C", None,
QtGui.QApplication.UnicodeUTF8))
        self.labelTemp.setText(QtGui.QApplication.translate("HelloPyQt", "CPU Temperature",
None, QtGui.QApplication.UnicodeUTF8))
    # Event triggered when the value of labelAlarm changed
    def sliderAlarm_ValueChanged(self):
        self.labelAlarm.setText("Alarm: " + str(self.sliderAlarm.value()) + "C")
    # Event triggered when timerTemp time out
    def timerTemp_TimeOut(self):
        # Get temperature from sensor file
        sensor = os.popen("cat /sys/class/thermal/thermal_zone0/temp")
        temp = float(sensor.readline()) / 1000
        alarm = float(self.sliderAlarm.value())
        # Display temperature
        self.lcdTemp.display("%.1fC" % temp)
        # Check whether the temperature is too high
        if temp <= alarm * 0.6:
            self.lcdTemp.setStyleSheet("color: green")
        elif temp <= alarm * 0.8:
            self.lcdTemp.setStyleSheet("color: orange")
        elif temp <= alarm:
            self.lcdTemp.setStyleSheet("color: red")
```

```
            else:
                self.lcdTemp.setStyleSheet("color: red")
                msg = QtGui.QMessageBox()
                msg.setWindowTitle("Alarm")
                msg.setText("Temperature is too high!")
                msg.setIcon(QtGui.QMessageBox.Warning)
                msg.exec_()
                # You can do something else here, like shut down the system
    if __name__ == "__main__":
        import sys
        app = QtGui.QApplication(sys.argv)
        HelloPyQt = QtGui.QMainWindow()
        ui = Ui_HelloPyQt()
        ui.setupUi(HelloPyQt)
        HelloPyQt.show()
sys.exit(app.exec_())
```

在上面的代码中，有很大一段是有关控件本身的代码（从 Qt 生成的 ui 文件转换过来的代码），包括位置、色彩、字体等基本属性。这是用 Python 实现的控件功能。读程序的时候，只要知道某段代码与那个控件有关就可以了，不要关心其细节。要设计或改变这些细节，可以用 Qt Creator、可视化地进行设计和修改，不要直接改代码，这就是借助 IDE 进行可视化开发的好处。

例如：这段代码中新建了一个 QTimer 定时器控件，用于定时查询树莓派 CPU 当前的温度并更新显示。self.timerTemp.timeout.connect()语句将定时器的超时信号链接至 timerTemp_TimeOut()函数。self.timerTemp.start(2000)设置定时器超时时间为 200ms，即每 2 秒执行一次 timerTemp_TimeOut()函数。

LCD 控件用于显示 CPU 当前的温度。在显示界面和运行定时器之前，可以通过手动调用 self.timerTemp_TimeOut()函数，读取 CPU 温度来初始化 LCD 控件内容。

树莓派 CPU 的温度通过读取 /sys/class/thermal/thermal_zone0/temp 文件内容获得。具体的方法是：采用 os.popen()方法，在 Linux Shell 中通过 cat 命令读取文件，并将返回的字符串信息转换为数字显示在 LCD 控件中。

窗口下面的滚动条用于设置温度报警门限，如果 CPU 温度接近门限值，则会改变温度显示的颜色。如果超过门限值则弹出对话框报警。代码运行的结果如图 4-3：

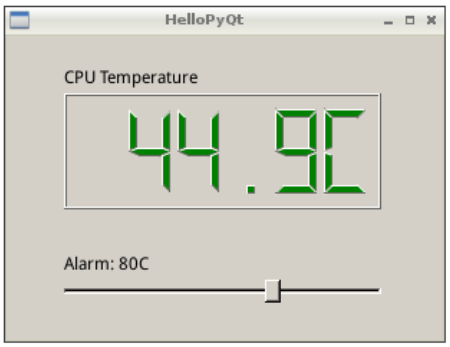

图 4-3 树莓派 CPU 温度显示界面

4.4 前端 arduino 与树莓派通信集成

上一小节实现了树莓派 CPU 温度的采集、显示与控制（报警），这既是熟悉和了解 Qt 工具的过程，也为接下来的项目，建立了一个很好的实验项目的基础。还使本实训项目开发过程，逐渐地由简单到复杂、需求的复杂度和代码量逐渐增加，用软件工程的观点看，这就是所谓：“迭代开发”或“二（N）次开发”，来“累加”式地添加、实现新需求。

1）理解二次开发的新需求：

根据新需求，接下来项目的任务是：把已经实现的“显示树莓派 CPU 温度”的功能，修改成显示 DHT11 采集的温度，同时，报警方式增加：如果超出了预警温度，自动或手动的点亮连接在 arduino 上的某个 LED 报警灯（模拟电器控制开关）。

新需求与已有系统在功能上的区别（新添加）是比较容易理解的：CPU 温度改为 DHT11 的温度、新加一个反向操作、点亮 LED 灯的功能。如果你是一名需求分析师，做出如上的回答，就可以了。接下来开始写测试用例，检查显示的温度与室内温度是否一致、测试当要求自动或手动点亮 LED 灯时，灯有没有被点亮。

而作为架构师，仅仅这样的理解显然是不够的。例如：通过什么方式，在树莓派上获得 DHT11 的温度数据？通过什么方式，把树莓派的控制命令送到 arduino 上去点亮某个 LED 灯？

这两个问题首先都涉及树莓派与 arduino 的信息交互通道以及交互内容（数据格式）和交互方式（通讯协议）问题。这是系统集成工程中，不同系统之间进行交互、协同的基本问题。

1） arduino 与树莓派通信的数据通道：

第四章 3.6.4 节介绍了 arduino 与树莓派的 GPIO、USB 以及 I2C 三种“有线”的通信方式。当然 arduino 也可以经扩展（添加 WiFi 卡）后，直接使用无线 WiFi，进行通信。Arduino 还可以使用 Zigbee 协议，这是一种低功耗、近距离的无线组网通讯技术，并且是 arduino 自带的。

由于上一节课已经做过利用 USB 串口，进行 arduino 与树莓派之间通信实验的例子，本节还是继续将 USB 串口，作为两者交互的“通道”。由于 arduino 上只有一个串口（串口通讯也是点对点的），所以，在 arduino 上看，这样的“通道”只有一条。也就是说，假如树莓派希望显示和控制多个 arduino，（不考虑其他因素的话），B+版本的树莓派最多可以控制 4 个 arduino。而每个 arduino 只能与一台树莓派相连接。

这就是硬件平台的局限。当然可以采用一些方法，绕开这个限制，例如：采用无线“通道”，就可以扩展通讯的通道限制。

2） arduino 与树莓派通信的数据格式：

因为是串口通信，所以，所谓 arduino 与树莓派之间的通信数据格式，在数据格式组织和理解以及具体代码实现上，就是 arduino 上的编程语言以及树莓派上的 Python 对于串口的读写格式。

例如：最简单的方法是：（1）在形式上，发送和接收方都把数据转换成 ASCII 字符串的形式，进行传输，而不是 2 进制、整数或浮点数等。（2）每次用一个“回车/换行”符，表示一组（一个温度值或一个命令）的结束、下一组数据的开始。至于对一组数据内容的理解，则由下面讨论的通信协议所规定。

3） arduino 与树莓派通信的通信协议：

通道和格式，可以看成通信“外部形式”的“硬性规定”，而下面要讨论的通信协议，则涉及信息通信的“内部约定”。所谓“内部”就是信息的发送和接收者之间的“约定”。本

例中的发送和接收者，就是树莓派与 arduino。

树莓派通过 USB 串口，已经能够读取温度信息了，还有什么“协议”要研究的吗？当然！

你可能会发现，在 3.6.4 的例子中，arduino 代码向串口写了一串字符——“testing”，在 3.7.6 小节，arduino 获取 DHT11 的温度之后，用一个 loop()死循环，向串口不断发送温度信息。在上述两个例子中，虽然串口的接收方不是树莓派，而是 arduino 的监视窗口，但本质是一样的。Arduino 是一个“傻瓜型”的信号发送器，它只会不断地发送信息，而不管信息的接收方如何接收、是否收到，以及还有什么其他的要求。例如：采样速度要再快一点/慢一点。或者接收方没有收到刚刚发出的信息，请重新发送等等与信息无关，而与信息的发送和接收本身有关的事情。再如本例新需求的、反向显示 LED 的指令（因为只有一个通道）等。在上面的两个例子中，即使接收方就是树莓派，也不能实现这样的通信要求。

要在一个“通道”上，实现上述功能，就需要对双方发送和接收信息的内容含义（不是格式）、处理要求、处理的顺序、对应关系等，进行“约定”。而这就是所谓“通信协议”。实际上，在本例中虽然没有讨论 arduino 和树莓派之间的“通信协议”，但是，双方还是假定：（1）arduino 是信息发送方，树莓派是接收方，而且始终是单向的，即只有 arduino 向树莓派的信息发送，没有反向的发送；（2）发送的内容和节奏，完全由 arduino 决定，树莓派只是被动地按“回车”作为一次接收单位，进行接收和信息理解；（3）由此，发送和接收双方，不考虑“同步”的问题。即不考虑双方计算机、传感器运行的速度、采集的速度、显示的速度等等的差异。因此，也不对一次采集、发送的数据，是否完整地被接收到，不做任何检查和要求。例如：如果对每次 DHT11 得到的温度值进行编号，然后在树莓派的显示上，除了显示温度外，还要显示此编号，则会发现，DHT11 与树莓派同时显示的编号号码，肯定会出现不一致、不能一一对应的情况。在实际应用中，这种信息编号（实质是传送的信息包号）的对应，被称为“同步”。在这里还只是“单向同步”，即树莓派与 arduino 同步、以 arduino 的编号为基准。更复杂的情况可能还需要“双方同步”，即当发生不同步时，需要双方协调，而不是单纯的以一方为基准。这就更复杂了，因为当发现不同步时，要考虑协调的机制（例如：主从）、协调实现的方法（建立缓冲池、进行缓冲控制），以及可靠性（保证达到信息不丢失）、效率（达到同步的速度）、开销（实现可靠性的代价）等。《软件架构设计实践教程》中 ATM 机故障恢复的架构设计方案讨论，就可以看成是一种同步恢复（故障恢复）的机制的平衡与取舍案例，可以参阅。

有关计算机通信的知识，是一门课程要讲授的内容，不在此过多展开。

4.5 实现温度显示与控制的集成

至此，讨论了在树莓派上，实现 DHT11 温度信息显示和控制的两个基本问题：用户界面设计与串口通信。

1）Qt 框架下的监控系统逻辑层次架构：

到目前为止，前一节及本节课程希望开发和集成的“系统”，可以概括为以下五个逻辑层次（这是第二章介绍的系统逻辑的概念），如图 4-4：

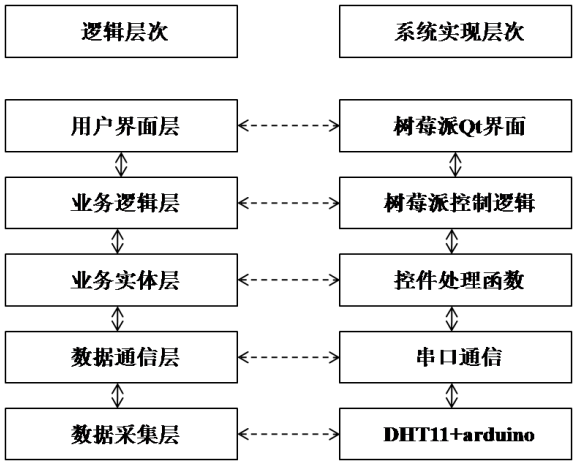

图 4-4 系统各逻辑层与各层的具体实现

（1）用户界面层，这一层实现的内容，就是图 4-3 的、采用 Qt Creator 设计的用户界面，重点是界面上窗口的大小、按钮的位置、显示的字符样式等等提供给用户使用的、用户直观能感受得到的东西；（2）最下面的数据采集层，是上一章实现的 DHT11 和 arduino 温度集成的传感器采集；（3）数据通信层是前一下小节讨论的串口通信；（4）业务实体是 Qt 控件（如：模拟数码管显示控件、按钮控件、横拉条控件等）的对应处理函数；（5）业务逻辑就是系统的总体组织。包括：用户界面设计——这不是指用户界面的外观设计，而是窗口、显示信息、按钮，以及按钮事件、鼠标事件等这些元素，应该有哪些，对应这些要素的处理功能应该是什么？这是面向对象分析中的 OMT 模型：对象是按钮、连接关系是消息和槽、功能是槽（函数）的处理代码。这三者之间联系的实现，主要是靠 Qt 的信号与槽的关联机制，即：Qt 自动实现了“事件消息”与“消息处理”链接，“系统”运行就在按钮被按下、消息发送、事件处理、返回窗口之间来回循环。这样的系统总体运行逻辑的组织，是 windows 桌面应用的基本形式。

2）Qt 框架下的系统开发架构：

虽然有关系统开发架构的内容，打算安排在第六章、介绍更大的应用系统的时候才会详细介绍，但这里还是需要简单提示一下。所谓软件系统的开发架构，是软件五个架构视角之一，主要反映构成软件系统的“部件”（子系统、组件、代码集）等，在解决方案下，是如何组织、存放的。例如：SSH 架构或 MVC 架构的系统，通常将 SSH 或 MVC 的三个主要部件，分别放在三个不同的包/目录下。

本系统由于比较小，可能就是一个“.py”的 Python 程序文件。但是，就是这一个程序文件中，也可以根据上述的架构层次，找出不同的“构件”的程序部分。

例如：用户界面显示部分的代码、控件的代码、处理函数的代码、串口通信的代码等。除此之外，还一定会有不属于上述部分的、例如：程序初始化、特定数据处理、转换、出错处理、退出程序的处理等，这都是一个系统所不可缺少的部分。

理解系统的开发架构的好处，是可以在二次开发的时候，很容易地找打自己所关注的部分。在维护、扩展、再测试的时候，也可以比较容易地圈定“相关”范围，不需要考虑与所修改无关的部分，节省了开发的时间和成本，也保证了开发的质量。

3）在 Qt 框架下，实现“监控系统”开发：

下面的任务，就是请同学们自己实现上述的“系统设计”构想。包括：（1）扩展图 4.3 的用户界面（添加控制按钮）；（2）修改模拟数码管显示函数，显示来自串口的 DHT11 传感器温度，而不是树莓派 CPU 的温度；（3）补充控制按钮被按下以后的处理函数；（4）修改串口通信协议，至少支持反向回送控制命令；（5）修改 arduino 程序，支持接到控制命令后，

点亮 LED 灯。

4.6 课程小结

这节课在上节的基础上，把树莓派的温度控制，“大大地”向前推进了一步，有点像“真”的控制系统的样子了。包括有了一个简单的用户界面，有了简单的参数设置和反向控制。虽然就这么简单，但它初步具备了一个“监控系统”的采集、显示、数据处理、阀值设置、状态判断、反向命令控制、控制动作执行等一系列的要素。同时，它也初步具备了一个“软件系统”用户界面、业务逻辑、业务实体、数据存取、数据存储（采集）的五个逻辑层级的内容。这是一个在“（监控）业务”功能和软件系统架构两个方面，初步具有雏形的“系统”。

总结一下：作为一名架构师，在讨论一个应用系统的时候，业务的“系统性”说的是功能是否齐全、更多考虑的是作为一个“产品”满足用户当前和未来需求的能力。而讨论软件系统的时候，则关注的是软件实现架构，是否完整，特别是后者。在本节课中，作为一个软件的“系统”、软件系统的“架构”，比起第二章五子棋案例中看到的软件“系统”与“架”构，还大得多（不只是代码量）、涉及的面广得多、内容丰富的多、深入的多了吧。这些方面，都需要架构师在系统开发前，进行完整的调研、分析、设计，并最终通过代码，把它们开发出来，集成在一起。

本节只是有关“系统”与“架构”的初步体验，这还只是开始。