

树莓派开发

23 在树莓派上运行PyQt4

回到树莓派

□ 树莓派的4个作用：

- 前端数据采集（上节课已实现）
- 数据预处理（略）
- 本地数据展示（本节课）
- 远程数据上传（下节课）

树莓派的展示作用与实现

- ❑ 树莓派上实用的用户界面GUI的工具：
- ❑ 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
- ❑ 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。
```

□使用如下命令，看看自动安装的软件放在哪里了:

```
dpkg -L python3-pyqt4
```

更多软件包管理可用aptitude

□获取PyQt的文档和范例程序（非必须，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。

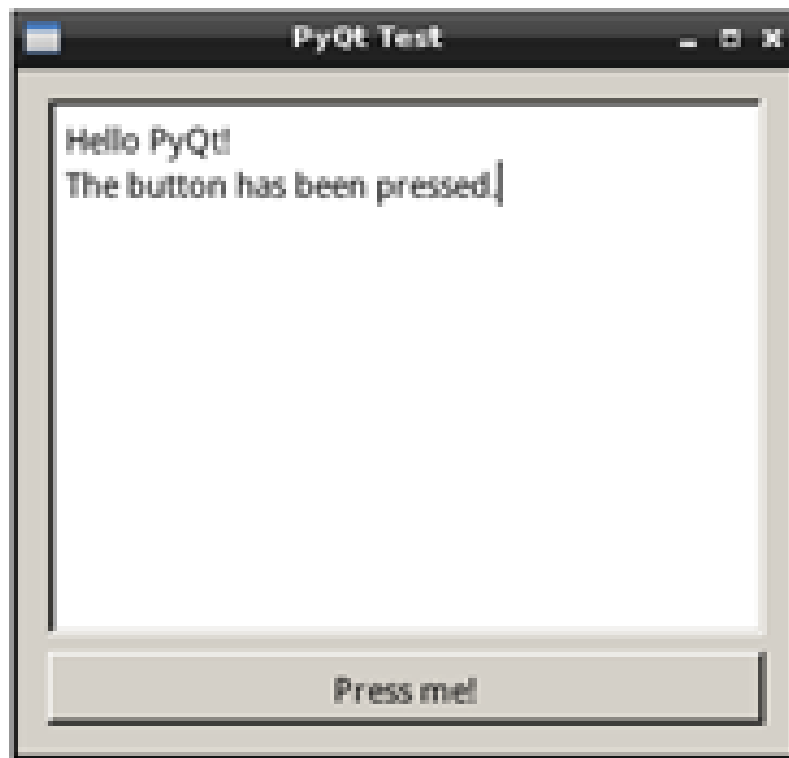


(2) Python3环境下的PyQt4测试代码

- 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：

(2) Python3环境下的PyQt4测试代码

- ❑ 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，就可以看见如下的运行结果了。



(2) Python3环境下的PyQt4测试代码

```
import sys
from PyQt4 import QtCore, QtGui
class HelloPyQt(QtGui.QWidget)://封装了所有处理函数的类
    def __init__(self, parent = None):
        super(HelloPyQt, self).__init__(parent)
        self.setWindowTitle("PyQt Test")//设置窗体的标题
        self.textHello = QtGui.QTextEdit("This is a test program written in python with PyQt
lib!")//在文本对象中显示一行信息
        self.btnPress = QtGui.QPushButton("Press me!")//在按钮对象上显示按钮标题
        layout = QtGui.QVBoxLayout()
        layout.addWidget(self.textHello)
        layout.addWidget(self.btnPress)
        self.setLayout(layout)
        self.btnPress.clicked.connect(self.btnPress_Clicked)//按钮对象与处理函数的链接
    def btnPress_Clicked(self):
        self.textHello.setText("Hello PyQt!\n\nThe button has been pressed.")//按下后显示信息
if __name__=='__main__':
    app = QtGui.QApplication(sys.argv)
    mainWindow = HelloPyQt()//QT启动窗体，进入事件循环，典型的QT模式
    mainWindow.show()
    sys.exit(app.exec_())
```

(2) Python3环境下的PyQt4测试代码

- ❑ 在本例中，所有的PyQt控件处理都封装在HelloPyQt类中，而主程序则非常简单（QT应用初始化的标准动作）。
- ❑ 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）。
- ❑ 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。

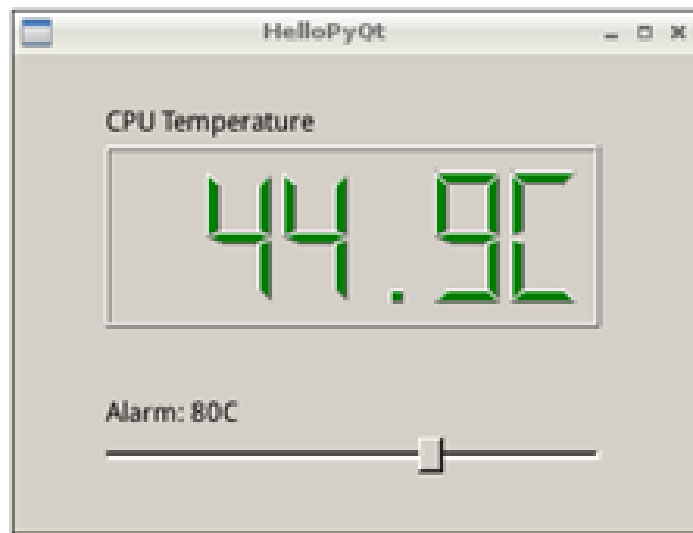
(3) QT的Qt Creator

- ❑ 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。

显示树莓派CPU温度的界面

□ 需求：

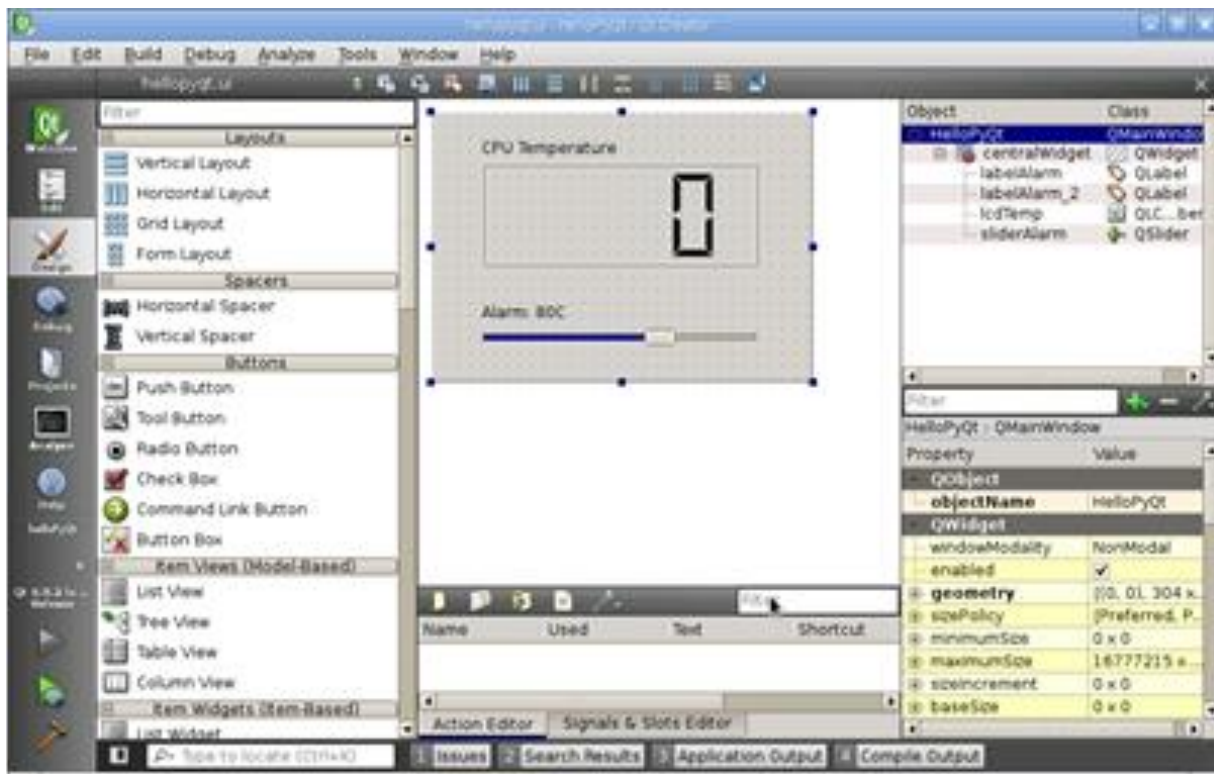
- 上一节课已经可以实现通过DHT11温度传感器，获得实时的温度信息。
- 现在的任务是，在树莓派的屏幕上，实现一个比较“美观”的温度控制器界面。
- 能够实时显示树莓派CPU的温度（树莓派暂时还没有与arduino通讯，先取自己的温度），设定预警温度，当温度接近预警温度时，可以改变显示的颜色，当超过预警温度时，可以报警。



使用Qt Creator创建用户界面

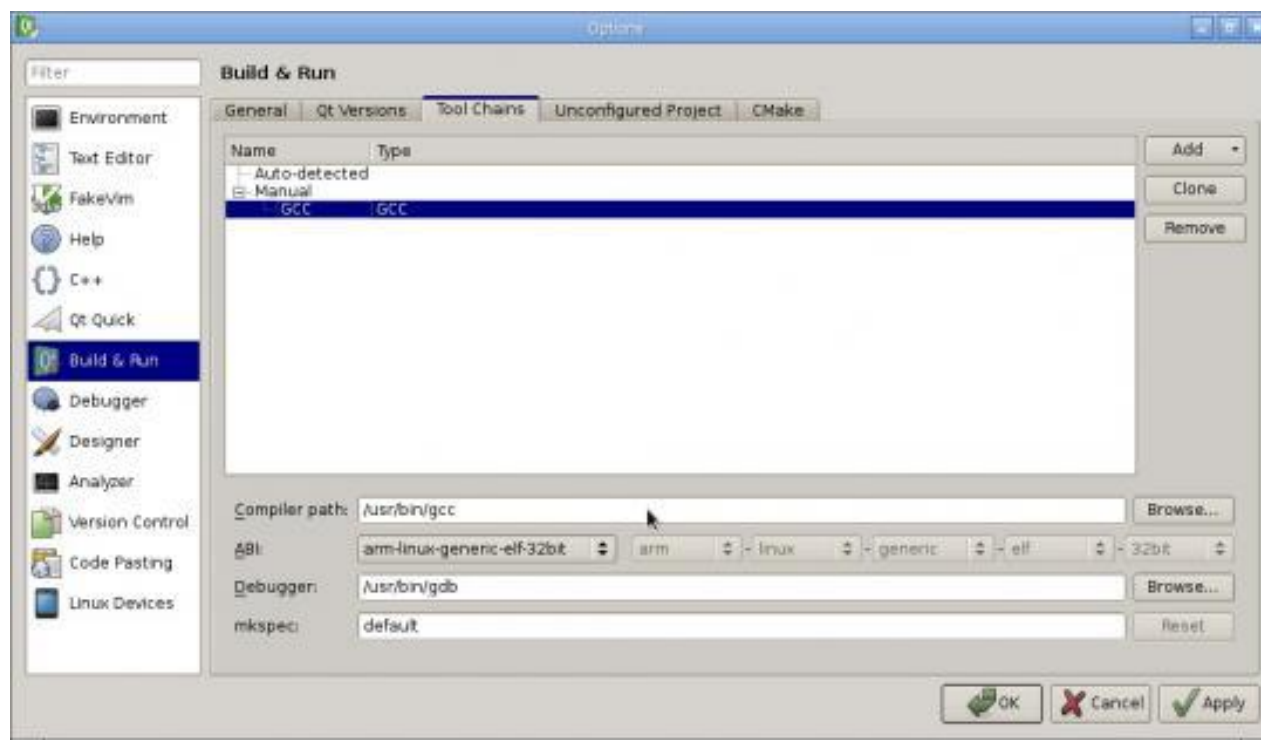
- 打开Qt Creator，新建一个Qt GUI应用程序工程（可以在树莓派本地安装运行Qt Creator，也可以用windows下的Qt Creator，设计Qt GUI程序。然后，再如前所述，通过转换ui文件为Python代码的方式）。
- 在树莓派上安装的命令是：`$ sudo apt-get install qtcreator`

- 在树莓派上运行qtcreator出现的窗口界面如图：



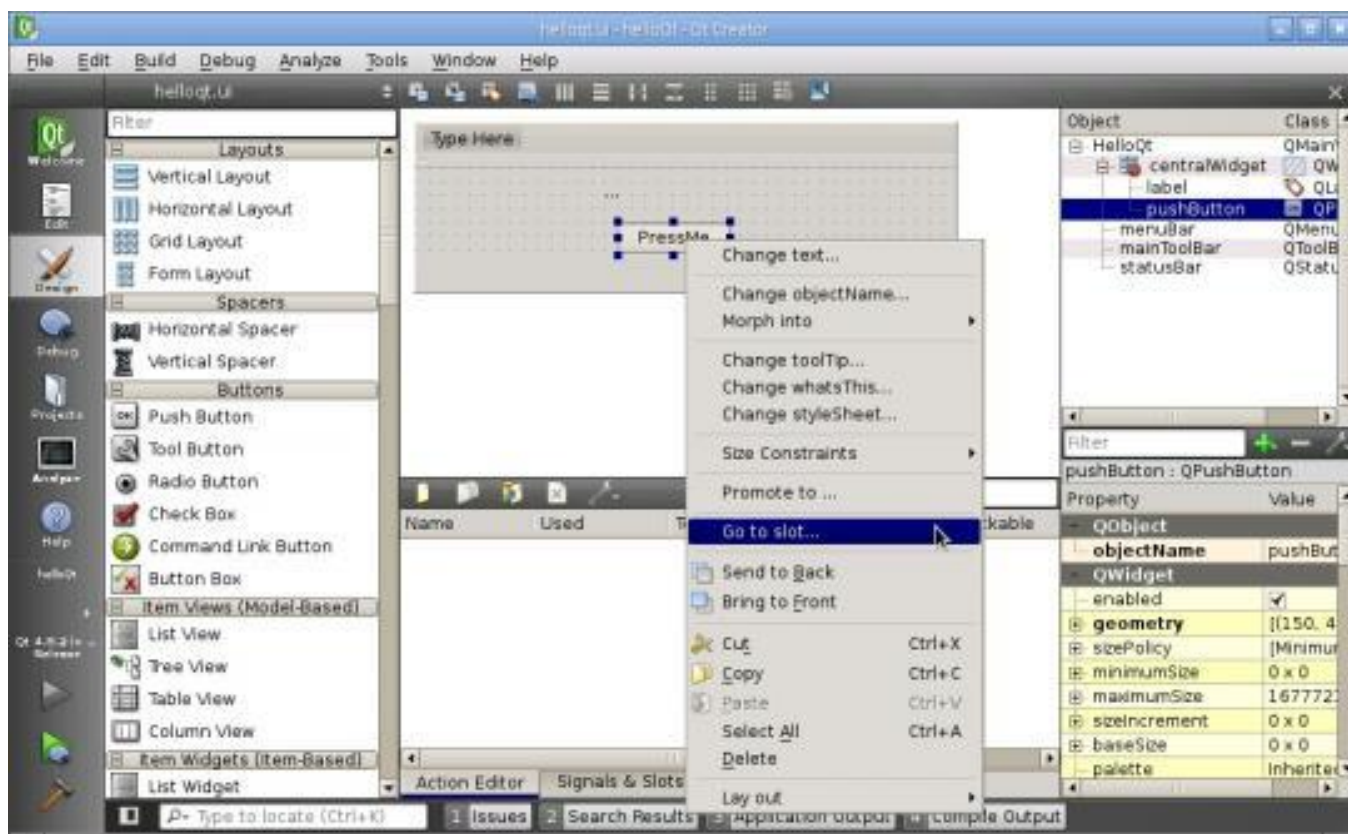
使用Qt Creator创建用户界面

- ❑ 由于Qt Creator不能自动识别树莓派上的工具链（编译命令等），因此需要手动添加。
- ❑ 点击Tools -> Options打开配置对话框，在Build & Run -> Tool Chains选项卡中点击Add添加GCC工具链，Compiler路径设置为/usr/bin/gcc，Debugger可设置为/usr/bin/gdb。



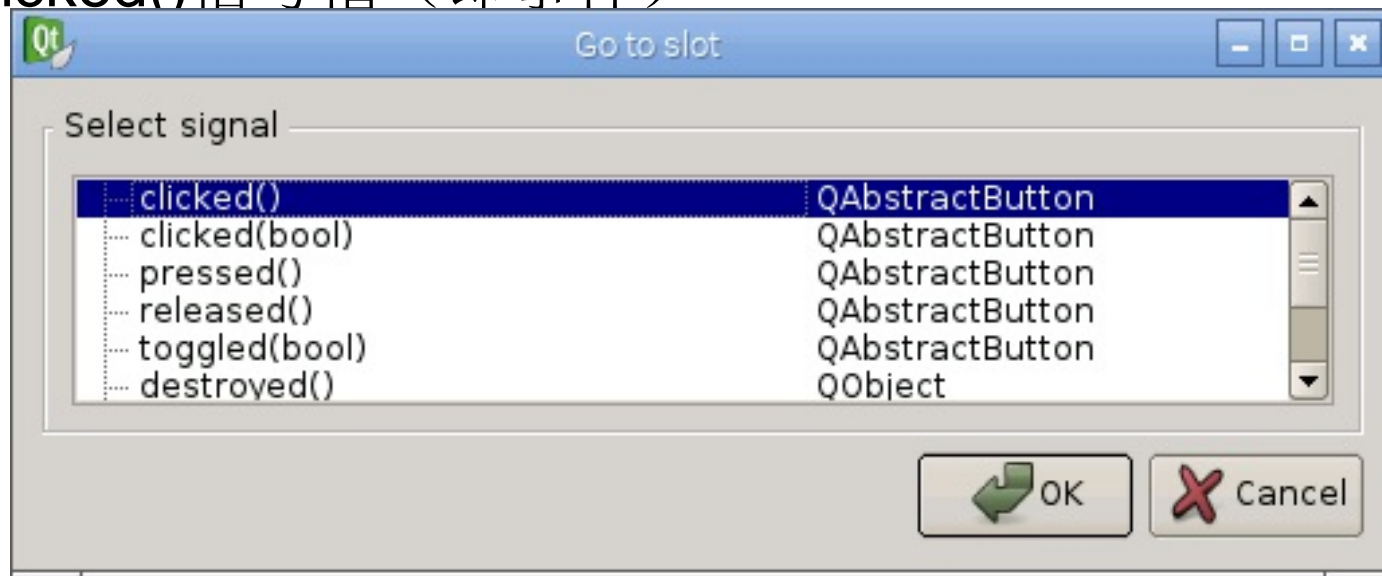
使用Qt Creator创建用户界面

- ❑ 完成设置后，可以通过一个简单的工程验证Qt是否可用
- ❑ 建立一个Qt Gui Application工程HelloQt。
- ❑ 添加一个Label和Push Button。
- ❑ 在Button上右键选择go to slot。



使用Qt Creator创建用户界面

- ❑ 在按钮上右键，在弹出菜单中选择goto slot
- ❑ 选择Clicked()信号槽（即事件）




使用Qt Creator创建用户界面

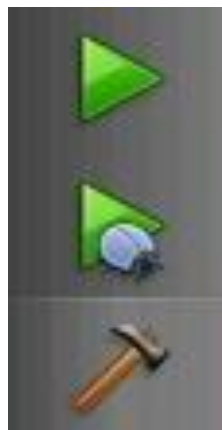
- ❑ 在on_pushButton_clicked()事件执行函数中添加改变label文字的语句。

```
void HelloQt::on_pushButton_clicked()
{
    ui->label->setText("Hello Qt!");
}
```



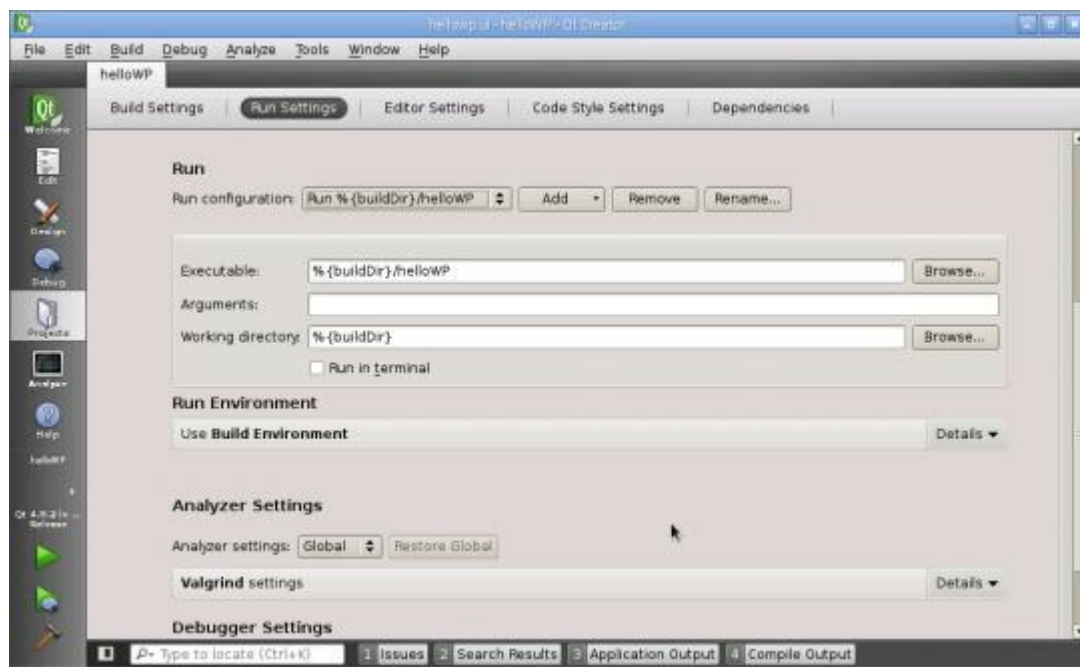
- ❑ 保存工程，通过  按钮编译工程。这时候还无法直接在Qt Creator中运行编译好的程序，这是因为缺少某些设置。
- ❑ 如果不做设置的话，编译好的目标将被QT放置在工程的缺省编译目录（HelloQT-build-embedded-Qt_4_8_2_in_PATH__System__Release）下，可以发现已经有编译好的可执行文件HelloQT，双击执行即可。

使用Qt Creator创建用户界面



- ❑ 为了能够在QtCreator上直接运行Build好的程序，设置如下两步：（此设置只对本项目有效）
- ❑ （1）设置编译路径（也可以在项目生成的时候指定）
 - ❑ 在Projects -> BuildSettings选项下修改Build directory为你自己制定的目录，也就是下一步要用的buildDir的值
- ❑ （2）设置可执行代码的路径

- ❑ 在Projects -> Run Settings选项卡的Run configuration栏中添加Custom Executable，将Executable属性设置为`%{buildDir}/*****`（*****为你的工程编译生成的可执行文件），可以通过浏览build目录，直接指定可执行代码。



使用Qt Creator创建用户界面

□ 运行效果：



点击PressMe，显示的内容就变了。

使用Qt Creator创建用户界面

- ❑ 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。

使用Qt Creator创建用户界面

□ 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")
```

```
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_())
```

Qt Creator生成代码分析

- 在生成的代码中，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。

Qt Creator生成代码分析

□ 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。

前端arduino与树莓派通信集成

萬和® IT教育
因专业而精彩

我们 · 始于1993年

- 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。



理解二次开发的新需求

□ 新需求：

- （1）实时显示经arduino-DHT11采集的温度（其他）
- （2）如果超出了预警温度，在树莓派上自动报警
- （3）当树莓派报警后，可在树莓派上自动或手动方式点亮连接在arduino上的某个LED报警灯（模拟反向电器控制开关）
- （4）树莓派报警解除，LED灯熄灭。

前端arduino与树莓派通信集成

- 新需求与已有系统在功能上的区别（修改或添加）：
 - （1）取CPU温度改为取DHT11的温度
 - （2）设置报警临界值（范围）
 - （3）新加一个反向操作、点亮LED灯的功能
 - （4）报警解除
- 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。

前端arduino与树莓派通信集成

- 通过什么方式，在树莓派与arduino之间传递数据？
- 要传递的数据：
 - 上传：DHT11采集的温度
 - 下传：控制信号（报警控制/解除）
- 实现方案：
 - 通道（USB串口）
 - 双向（单一通道、双向通讯）
 - 内容（上传：温度值（时间？））、下传：控制命令）
 - 实时、并发、同步/异步、连接（暂时无要求）
 - 容错（暂时无要求）
- 这些问题涉及树莓派与arduino的信息交互通道以及交互内容（数据格式）、交互方式（通讯协议）、交互控制（实时性）、错误处理等问题。
- 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。

实现温度显示与控制的集成

- 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arduino之间串口通信

Qt框架下的监控系统逻辑层次架构

(1) 用户界面层，这一层实现的内容，就是采用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；

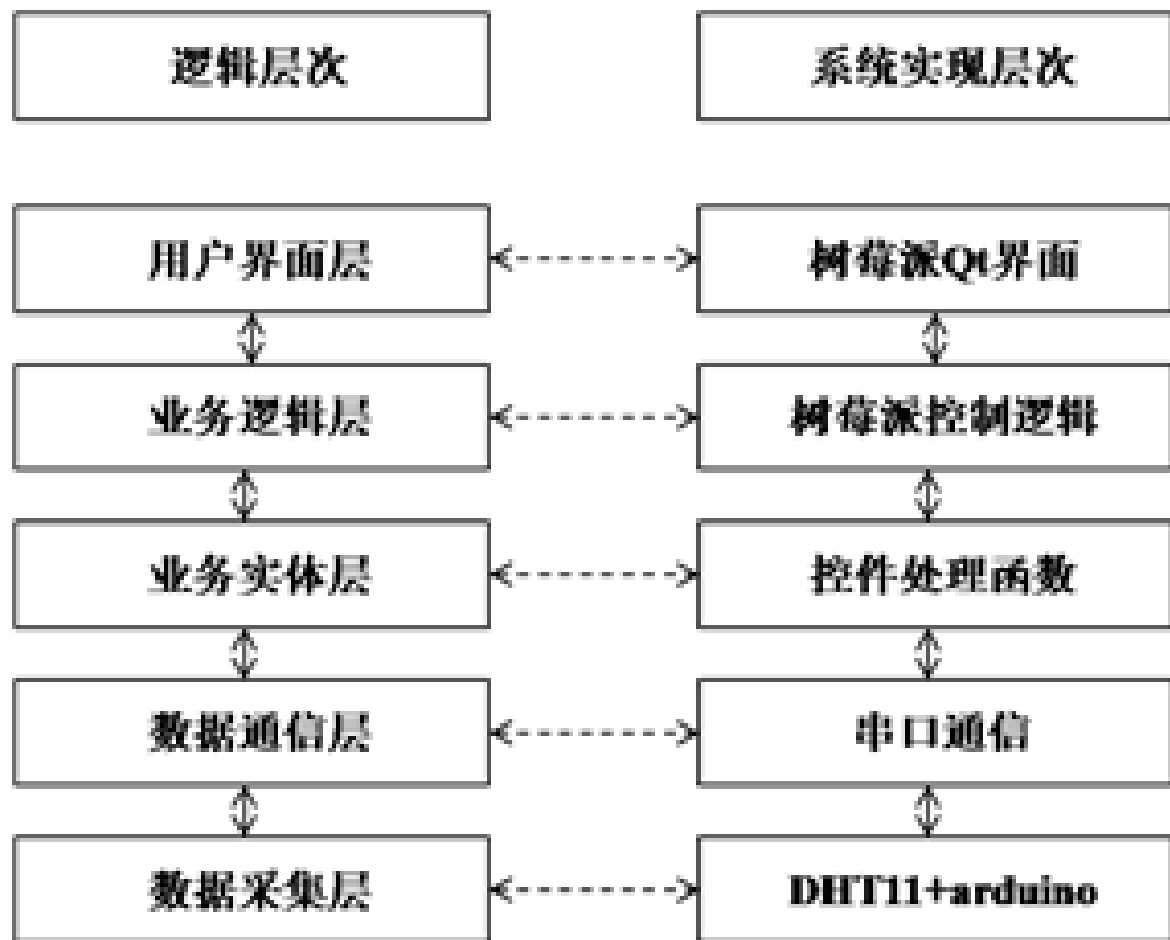


图 4-4 系统各逻辑层与各层的具体实现

Qt框架下的监控系统逻辑层次架构

(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。

Qt框架下的监控系统开发层次架构

- ❑ 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，很容易地找到自己所关注的部分。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。

在Qt框架下，实现“监控系统”开发

- ❑ 本周的任务，就是请同学们自己实现上述的“系统设计”构想。具体包括：
- ❑ （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处理函数，显示来自串口的DHT11传感器温度，而不是树莓派CPU的温度（过渡任务）；
- ❑ （3）补充控制按钮被按下以后的处理函数，添加反向处理；
- ❑ （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
- ❑ （6）反向控制有两种：报警（电灯）、解除报警（灭灯）