







售货机安全卫士 APP 端

User Guide

内部文件：B07-1710024SF01

颁布时间：2017/10/20

目 录

	文件版本说明.....	2
	参考资料.....	2
	手册目的.....	2
	声明.....	2
	名词定义和缩略语说明.....	2
1	项目名称.....	3
2	设计要求及性能指标.....	3
3.	业务模式.....	3
3.1	总体软件架构.....	3
3.2	技术难点.....	3
4	开发过程.....	4
4.1	开发软件及环境介绍.....	4
4.2	Android 开发环境搭建.....	4
4.2.1	工具下载.....	4
4.4.2	Qt 环境配置.....	5
5	设计细节.....	9
5.1	总体软件流程.....	10
5.2	蓝牙部分开发.....	11
5.2.1	平台准备.....	11
5.2.2	Qt 蓝牙组件.....	11
5.3	接收机制.....	17
5.3.1	短数据连接.....	18
5.3.2	长数据连接.....	19
5.4	发送机制.....	20
5.5	UI 设计	21
6	GNU 协议声明 	22

文件版本说明

表 1 版本说明

版本	发布时间	修订章节	作者
1.0	2017/10/15	创建文档	Carlos Wei
1.0	2017/10/20	编辑文档	Carlos Wei

参考资料

手册目的

该手册用于用户参考、传达设计思路、兼具培训讲义功能。

声明

手册内部培训使用，不要上传到任何公共区域。

名词定义和缩略语说明

表 2 名词定义及缩略语说明

序号	缩写	说明
1	报警行为	闪光灯闪烁 拨打电话 播放警报

1 项目名称

《售货机安全卫士》之 APP 技术文档

2 设计要求及性能指标

根据需求分析书，配套文档 B07-1710024EM01，进行软件部分的延拓和扩展。根据设计要求，要求在 Android 智能手机上设计一款 APP，能够对自动售货机的以树莓派为核心的主控中心（以下称为下位机）进行监测和控制，能够获取自动售货机的运行状态和并能够通过 P2P（peer-to-peer）方式进行图像数据的获取和显示。

3. 业务模式

3.1 总体软件架构

我们将 APP 部分划分为四大部分，分别是设备驱动部分、状态指示、图像显示和控制区域，每个部分都是 APP 中不可或缺的机制。

第一个机制为设备驱动部分，主要调用了 Android 系统底层提供的蓝牙接口，依托 Android 系统的 API 参数，要使用 Bluetooth 进行数据通信就需要启动是那个服务，在该 APP 中，启用了这三个服务，为整个系统接收和发送数据提供了基础。

第二个机制为状态只是部分，能够显示整个系统的状态，在运行的时候 APP 处于被状态。

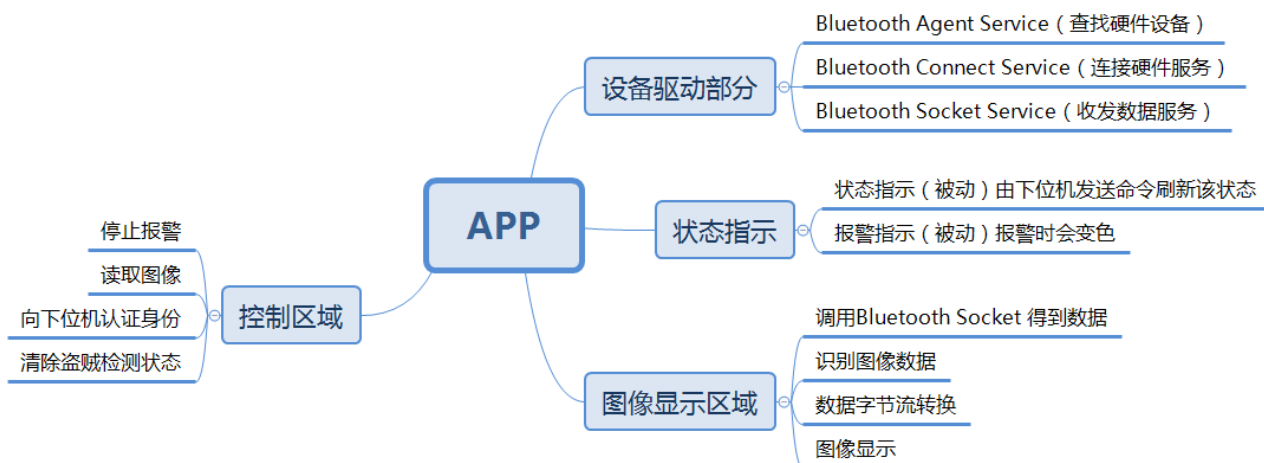


图 1 业务思路

3.2 技术难点

在本设计中，有几项技术难点需要。

- 1) C++需要联合 Java 进行编程，C++如何做好与 Java 的通信。
- 2) 在图像显示的功能上，需要解析下位机的字节流，如何保证字节流不丢失的完全显示。

4 开发过程

4.1 开发软件及环境介绍

通常的 Android 系统软件主流使用的是 Java 语言及谷歌公司提供的 Android Studio 软件，而本文选择 Qt 公司提供的 Qt on Android 版本，使用的是 C++ 语言，（选择理由：1. Qt 主要是面向工业级的 APP 软件，界面丑陋但性能能够满足工业数据需求，对于工业通信协议支持很好。2. 作为一名学生没有时间精通那么多语言，在会 C 语言的基础上，所以选择了 C++ 开发）。（以上可能是老师要问的，为什么选择一个非主流，你就这么答就好了）

我们选择的是 Qt 5.8.0 on Android 版本，当然虽然使用 C++ 语言进行开发，但是还是需要提供 Java 语言环境、Android SDK、Android NDK，开发的套件。（实际上，当我们使用 C++ 语言进行编程时，系统会自动和你配置好的 Java 环境进行转换，这个过程不需要考虑）

我们所有的开发环境都是在 Windows 系统进行的，所以在开发时候需要准备：

1. Qt 5.8.0 on Android for Windows x64 的 Qt 环境。
2. JDK 8u25
3. Android SDK 套件
4. Android NDK 套件

将以上文件准备好后，就可以开始在你的 Windows 系统上配置 Qt 的开发环境了。

4.2 Android 开发环境搭建

4.2.1 工具下载

1) 安装 Qt 5.7.0 for Android

Qt 5.8.0 for Android 只有 Qt 5.8.0 for Android (Windows 32-bit, 1.2 GB)，

下载地址：<http://mirrors.ustc.edu.cn/qtproject/archive/qt/5.8/5.8.0/qt-opensource-windows-x86-android-5.8.0.exe>，

校验信息：

Filename: qt-opensource-windows-x86-android-5.8.0.exe

Size: 1.2G (1247483552 bytes)

Last modified: Wed, 15 Jun 2016 06:44:56 GMT (Unix time: 1465973096)

SHA-256 Hash: 4ea371091a74d50c94d1a7463d7b802dac4c76a09d731ceb35d76a85ef6f830a

SHA-1 Hash: 68851d6555bc07207d22e7a8cc4f4b612529a7c9

MD5 Hash: 518be2a341d70df61b3749018c441c60

安装方式：默认安装。

安装完后，启动 qtcreator.exe 程序，然后“工具”->“选项”->“Android”，可以看到，Qt 需要 JDK、Android SDK、Android NDK、Ant 这 4 个软件。并且 Qt 给出了每个软件的下载地址。下面有截图。

Qt 给出的下载地址如下：

JDK: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>，

Android SDK: <http://developer.android.com/sdk>，

Android NDK: <http://developer.android.com/tools/sdk/ndk/index.html#Downloads>，

Ant: <http://ant.apache.org/bindownload.cgi>,

2) 安装 JDK,

因为 Android Studio 依赖 1.8 的 JDK, 所以 JDK 的版本至少为 1.8.0, 安装最新版就行, 的 JDK 版本为 jdk-8u101-windows-x64.exe,

下载地址: http://download.oracle.com/otn-pub/java/jdk/8u101-b13/jdk-8u101-windows-x64.exe?AuthParam=1469878902_445005a2cb14c433236175a9e8ac226c,

校验信息:

sha256: cbd29f09cc3c3320e300d7e587d2ffa5a3bd5fe0b9b7f70926909ab806ef0b1d

md5: a1eb90c4152787567ab2af80d3beb34c

安装方式: 默认安装。

3) 安装 Android SDK,

下载地址: <https://dl.google.com/dl/android/studio/install/2.1.2.0/android-studio-bundle-143.2915827-windows.exe>,

用户向导: <https://developer.android.com/studio/install.html>,

校验信息:

Size: 1187 MB (1245610376 bytes)

SHA-1 checksum: 9d677be09ccbb0195f52a429020b5bf0939e95d3

安装方式: 参考"用户向导", 参考下面的截图。

4) 安装 Android NDK,

下载地址: https://dl.google.com/android/repository/android-ndk-r12b-windows-x86_64.zip,

校验信息:

Size (Bytes): 749567353

SHA1 Checksum: 337746d8579a1c65e8a69bf9cbdc9849bcacf7f5

安装方式: 解压即可。压缩包内的文件夹为 android-ndk-r12b, 建议将其匹配到目录"C:\android-ndk\android-ndk-r12b"。

5) 安装 Ant,

下载地址: <http://www.trieuvan.com/apache/ant/binaries/apache-ant-1.9.7-bin.zip>,

校验信息:

sha1: f6d3f9aa55661a5cb2dff3f1933ca9a59910206c

md5: 48662e4d37b28dab1937f3745ca0da8b

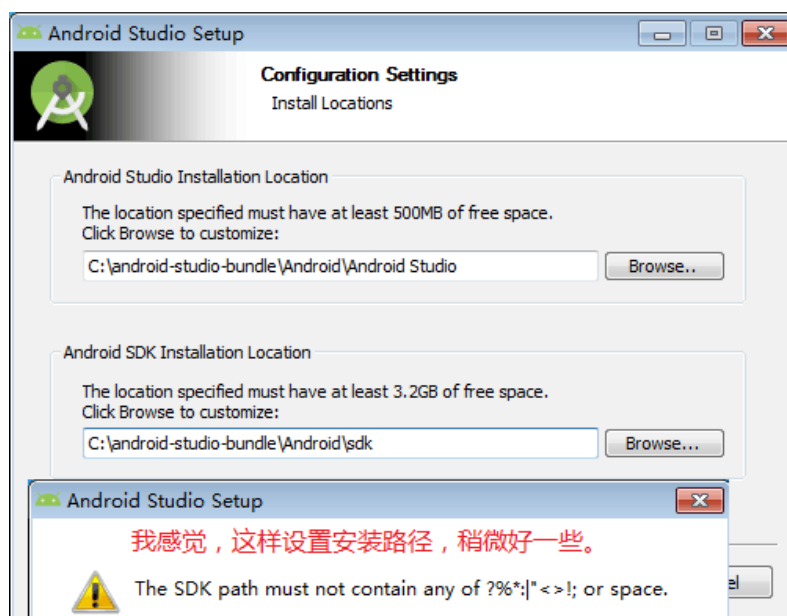
安装方式: 解压即可。压缩包内的文件夹为 apache-ant-1.9.7, 建议将其匹配到目录"C:\apache-ant\apache-ant-1.9.7"。

4.4.2 Qt 环境配置

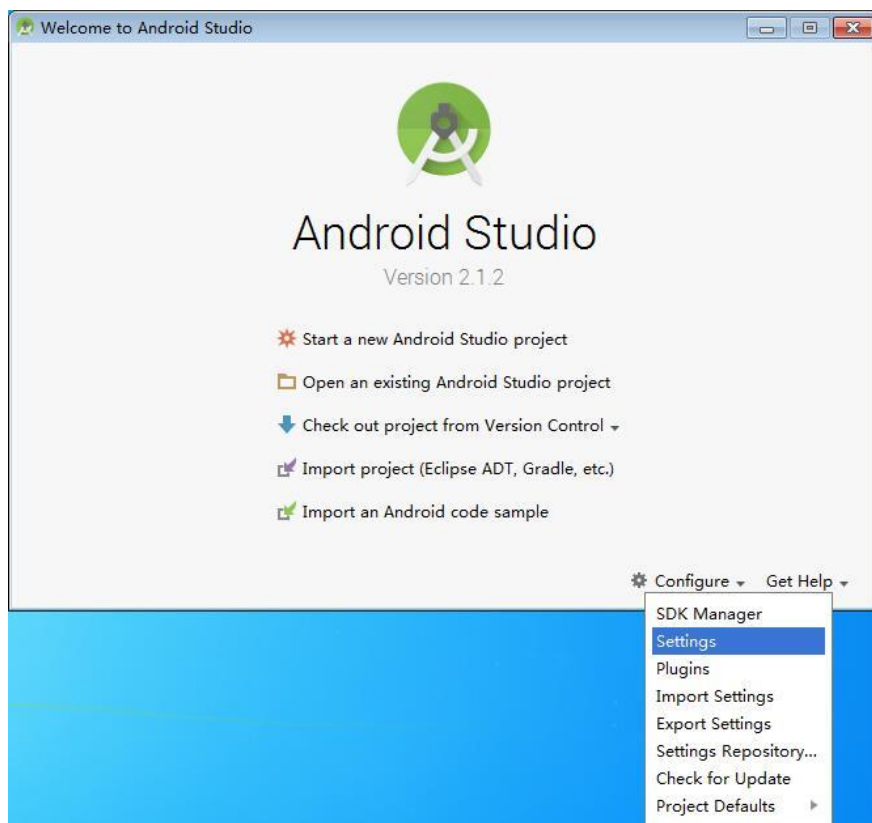
Qt 尚未配置软件时的样子:



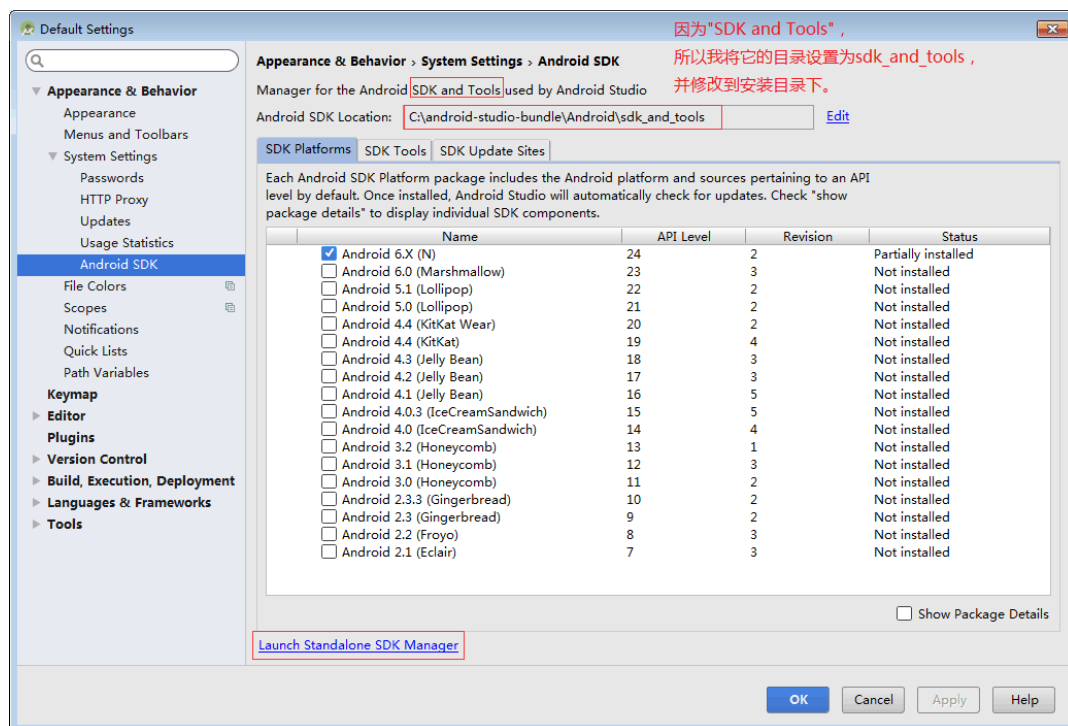
安装 Android SDK (android-studio-bundle)时, 我感觉, 修改一下安装路径稍微好一些:



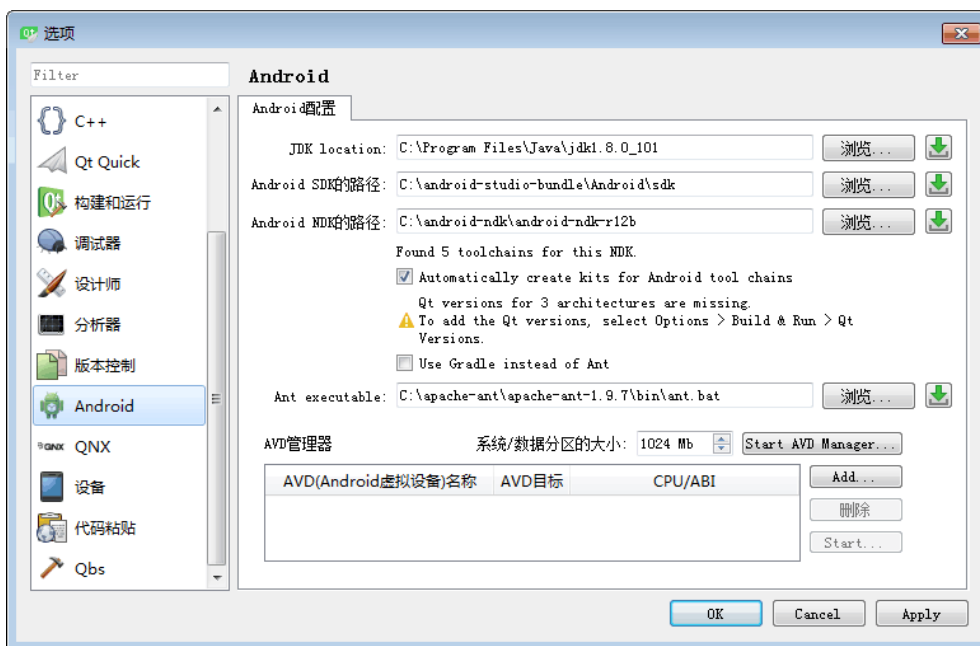
初次启动时, 一切默认就行。直到出现 Android Studio 的首页, 然后点击 Configure, 转到 Settings,



修改一下“Android SDK Location”的值，

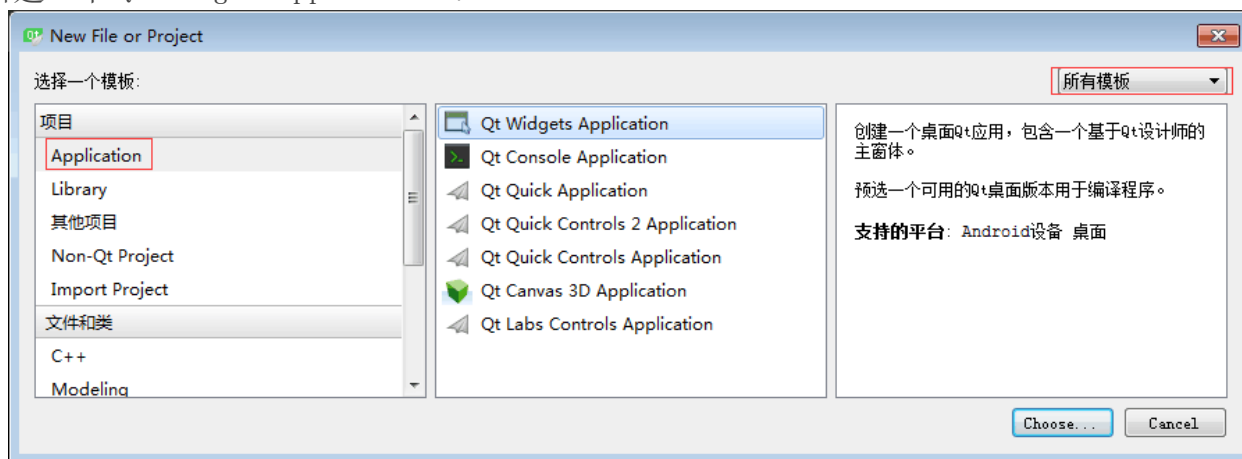


将软件的路径配置到 Qt 里面：

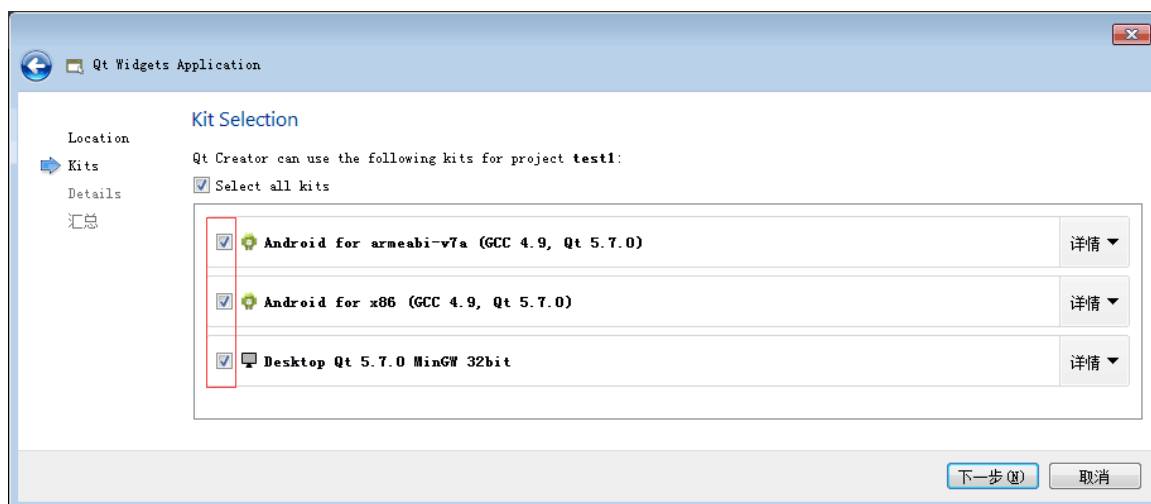


下面就是新建一个 Project，写一个 demo 程序，然后编译出来一个 .apk 文件，并将其安装到手机里，运行一下，看看情况。

新建一个 Qt Widget Application:



给这个 Project 取名为 test1，其目录为“C:\Qt\Qt_projects\test1”，它使用了下面的 kit，



创建完一个名字为 test1 的 Project 之后，就是稍微添加几行代码，让程序能执行一个反馈

1. //拖动一个"Push Button"到 UI，然后右键这个 Button，转到槽，选择"clicked()"信号，
2. //先 include 头文件 QMessageBox:
3. #include "QMessageBox"
4. //再在函数里添加如下代码:
5. QMessageBox::information(this, "I am title", "I am text", QMessageBox::Yes|QMessageBox::No|QMessageBox::Close|QMessageBox::Ok|QMessageBox::Cancel);

选择 Android for armeabi-v7a 的 Release,



构建它



到此 Qt on Android 的开发环境已经建立完毕。

5 设计细节

上一章，通过建立一个简单的 Test 工程完成了对于 Qt 的初步建立，理解是如何建立一个 Qt 工程的，Qt 是如何和手机连接上的。本章将对此次设计进行展开。

5.1 总体软件流程

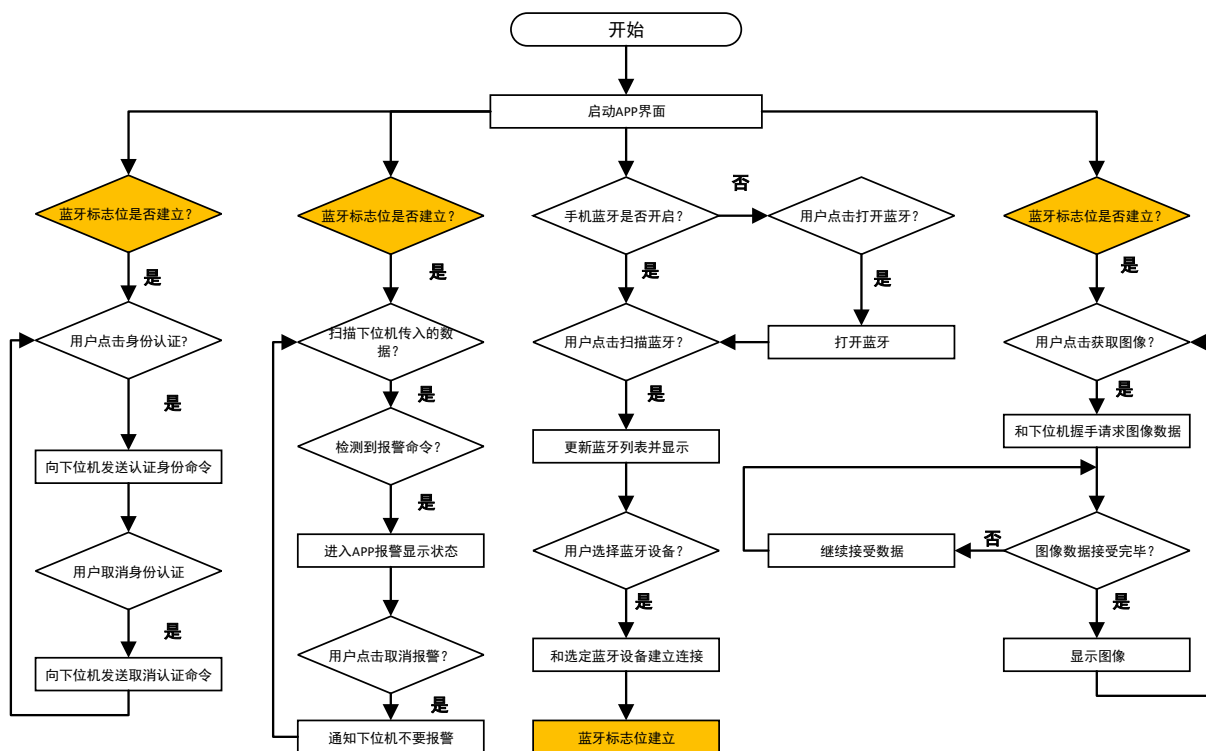


图 APP 运行总体流程

如上图所示为 APP 运行总体流程，橘黄色的标记应该为用户最优先运行，其他线程是建立在“蓝牙标志位建立”的基础之上的，（只有建立了蓝牙的链接，才能进行数据的通信）。

信号连接：

设备列表：

打开蓝牙

扫描蓝牙

☐ 蓝牙可见

关闭蓝牙

断开连接

图 蓝牙设备链接部分

在图中这个位置 APP 启动后会检测蓝牙是否开启，如果蓝牙已经开启了，则“打开蓝牙”按钮会为灰化状态无法点击，如果蓝牙没有开启，则为如图状态，用户打开蓝牙之后，就会开启 Android 系统的蓝牙，之后就可以进行蓝牙扫描了，扫描成功后的列表会显示在“设备列表”里面，列表里面的设备，双击之后就可以进行连接，（注意点击后会提示正在链接要点击弹窗的 OK），建立蓝牙标志之后，我们就可以对 APP 进行操作了。

APP 主要分为三个部分，一个为巡检下位机的报警状态、获取图像和身份认证。这三个状态如流程图中的三个分支，在流程图中已经表示很清晰了，就不再赘述。

5.2 蓝牙部分开发

5.2.1 平台准备

1) 硬件平台

1. 蓝牙: HC-05, 它的接口就是跟串口一样的，我们用到了 TX, RX, GND, VCC 四个引脚。跟下位机或者用 CH340G TTL 转 USB 模块接到 PC 机上。蓝牙工作在串口模式可以通过 AT 指令调节。具体参考蓝牙配套的说明文档，最主要的就是请将蓝牙设定为从机模式，否则安卓手机搜寻链接不上。

2. 安卓手机: 我这里测试用了 2 台安卓手机，一台是小米 4 移动版，安卓版本 6.0.1；一台是 MOTO MT887，安卓版本 4.1.2。

2) 软件平台

本项目 Qt 版本是 5.8，系统是 windows 8.1 x64

5.2.2 Qt 蓝牙组件

1) 蓝牙基础组件

蓝牙组件需要实现蓝牙状态检测、蓝牙的开关、蓝牙的扫描和蓝牙配对链接，并且能像串口助手一样完成数据收发。如图，就是本一开始做的最简单的软件界面，本软件基于 QWidget 控件制作，当然你可以选择 mainwinodw，更可以自己定义类。

蓝牙打开后通过扫描，会将蓝牙的 MAC 地址还有名字显示在 List 中，我们双击 List 列表中的蓝牙，就会进入 **actived** 信号连接的槽函数，执行蓝牙的配对连接。建立连接之后，就类似串口一样可以进行数据通信了。另外，点击 **send** 按钮之后会发送一堆字符串。

需要用到蓝牙就需要在 .pro 文件中引入库，你需要在 .pro 文件中加入这句话：

```
QT += bluetooth
```

如果没有这句话的话，包含蓝牙目录下的头文件，会提示找不到该文件。

之后就是要包含一些蓝牙用到的头文件：

```
#include <QtBluetooth/qbluetoothglobal.h>
#include <QtBluetooth/qbluetoothlocaldevice.h>
#include <qbluetoothaddress.h>
#include <qbluetoothdevicediscoveryagent.h>
#include <qbluetoothlocaldevice.h>
#include <qbluetoothsocket.h>
```

一会儿介绍每个都是做什么的。

请在类中声明定义蓝牙相关句柄：

```
QBluetoothDeviceDiscoveryAgent *discoveryAgent;  
QBluetoothLocalDevice *localDevice;  
QBluetoothSocket *socket;
```

2) 蓝牙的可见性

第一个 **discoveryAgent** 是用来对周围蓝牙进行搜寻，**localDevice** 顾名思义，就是对本地设备进行操作，比如进行设备的打开，设备的关闭等等。**socket** 就是用来进行蓝牙配对链接和数据传输的。这里要用到这三个。

在构造函数中，请为 **localDevice** 使用 **new** 运算符分配内存。

```
localDevice = new QBluetoothLocalDevice();
```

2.1) 蓝牙开关

本设计在运行 APP 的时候，会检测一下我们本地设备的蓝牙是否打开，如果判断是开启状态，我们可以将打开蓝牙的按钮 **disable** 掉，将关闭蓝牙的按钮 **enable**，所以在 APP 运行的时候需要进行蓝牙状态检测。检测方法如下：

进行一个这样的检测，对本地设备模式进行判断。

```
if( localDevice->hostMode() == QBluetoothLocalDevice::HostPoweredOff ) {  
    ui->pushButton_openBluetooth->setEnabled(true);  
    ui->pushButton_closeDevice->setEnabled(false);  
} else {  
    ui->pushButton_openBluetooth->setEnabled(false);  
    ui->pushButton_closeDevice->setEnabled(true);  
}
```

在构造函数中

那么，我们如何来对蓝牙进行打开和关闭呢？我在 **open** 按钮和 **close** 按钮的槽函数中对蓝牙进行开关操作。

open 按钮的槽函数：

```
void Widget::on_pushButton_openBluetooth_clicked()  
{  
    localDevice->powerOn();  
    ui->pushButton_closeDevice->setEnabled(true);  
    ui->pushButton_openBluetooth->setEnabled(false);  
    ui->pushButton_scan->setEnabled(true);  
}
```

localDevice->powerOn();方法调用打开本地的蓝牙设备，然后你可以根据自己的喜好完成对按钮的使能和禁止操作。

close 按钮的槽函数：

```
void Widget::on_pushButton_closeDevice_clicked()
{
    localDevice->setHostMode(QBluetoothLocalDevice::HostPoweredOff);
    ui->pushButton_closeDevice->setEnabled(false);
    ui->pushButton_openBluetooth->setEnabled(true);
    ui->pushButton_scan->setEnabled(false);
}
```



close 设备和我们的 **open** 设备的方法在形式上不一样，我还以为他们两个是对称的，但是事实上不是，只能用这样的方法对蓝牙进行关闭。

2.2) 可见性

同样地，在蓝牙使用过程中，安卓手机提供了蓝牙是否可以被其他蓝牙搜索到这样的功能，也就是蓝牙可见，我们也可以用 **localDevice** 下的 **HostMode()** 方法，对这个状态进行检测。如下：

```
if( localDevice->hostMode() == QBluetoothLocalDevice::HostDiscoverable ) {
    ui->checkBox_discoverable->setChecked(true);
}else {
    ui->checkBox_discoverable->setChecked(false);
}
```

我的设计中，蓝牙可见如界面图用的是 **checkBox** 空间完成的，通过 **setChecked()** 方法，一开机对是否可见进行。

在翻转 **checkBox** 的时候，会激发进入 **checkBox** 的槽函数，我们在 **checkBox** 的槽函数中，完成对蓝牙可见性的设定。代码如下：

```
localDevice->setHostMode( QBluetoothLocalDevice::HostDiscoverable);
```

同理，不可见你也能想到对吧。

2.3) 蓝牙查找

使用蓝牙设备的查找，就要用到 **discoveryAgent** 这个类的实例化。我们需要在构造函数中对 **discoveryAgent = new QBluetoothDeviceDiscoveryAgent();** 分配内存。然后就可以使用这个类的方法对蓝牙进行查找了。除此之外，还要进行一个信号和槽的链接。

```
connect(discoveryAgent,
        SIGNAL(deviceDiscovered(QBluetoothDeviceInfo)),
        this,
        SLOT(addBlueToothDevicesToList(QBluetoothDeviceInfo))
);
```

在我们发现设备的时候，这个 **deviceDiscovered** 信号被触发，进入到 **addBlueToothDevicesToList** 的函数中。在上面的软件界面，我们的最上面蓝牙列表下的控件是 **ListIt** 控件，这里做一个槽函数，将发现的设备打印到这个列表中列出来。



```
void Widget::addBlueToothDevicesToList( const QBluetoothDeviceInfo &info )
{
    QString label =
    QString("%1 %2").arg(info.address().toString()).arg(info.name());

    QList<QListWidgetItem*> items = ui->list->findItems(label,
    Qt::MatchExactly);
```

```

    if (items.empty()) {
        QListWidgetItem *item = new QListWidgetItem(label);
        QBluetoothLocalDevice::Pairing pairingStatus =
localDevice->pairingStatus(info.address());
        if (pairingStatus == QBluetoothLocalDevice::Paired || pairingStatus ==
QBluetoothLocalDevice::AuthorizedPaired )
            item->setTextColor(QColor(Qt::green));
        else
            item->setTextColor(QColor(Qt::black));
        ui->list->addItem(item);
    }
}

```

这里给出这个函数，每一句话十分的好理解，这里增加点选操作，当点击 `listItem` 中的项目的时候，背景颜色会翻转，双击这个项目就会和这个蓝牙设备建立连接，这里有个 `activated` 槽函数，在这个槽函数里面就会进行蓝牙的连接。下一章节写这个如何连接。

2.4) 蓝牙建立连接

在说蓝牙设备连接之前，不得不提一个非常重要的概念，就是蓝牙的 `Uuid`，引用一下百度的：

在蓝牙中，每个服务和属性都唯一地由“全球唯一标识符”（UUID）来校验。正如它的名字所暗示的，每一个这样的标识符都要在时空上保证唯一。UUID 类可表现为短整形（16 或 32 位）和长整形（128 位）UUID。他提供了分别利用 String 和 16 位或 32 位数值来创建类的构造函数，提供了一个可以比较两个 UUID（如果两个都是 128 位）的方法，还有一个可以转换一个 UUID 为一个字符串的方法。UUID 实例是不可改变的（immutable），只有被 UUID 标示的服务可以被发现。

在 Linux 下你用一个命令 `uuidgen -t` 可以生成一个 UUID 值；在 Windows 下则执行命令 `uuidgen`。UUID 看起来就像如下的这个形式：2d266186-01fb-47c2-8d9f-10b8ec891363。当使用生成的 UUID 去创建一个 UUID 对象，你可以去掉连字符。

在我们的项目中，用到的模式是串口模式，我们需要建立一个存储 `Uuid` 的机制，如下：

```
static const QLatin1String serviceUuid("00001101-0000-1000-8000-00805F9B34FB");
```

这个字符串里面的内容就是串口模式的 `Uuid`，如果你开发的蓝牙也是要使用串口，你直接 Copy 过去就可以了，如果你使用其他模式，自己去找这个 `Uuid` 码是多少。

在使用蓝牙建立连接，需要建立蓝牙 `socket` 服务。请在构造函数中增加对 `socket` 的分配内存，要注意的是构造函数中的参数需要给定模式。

```
socket = new QBluetoothSocket(QBluetoothServiceInfo::RfcommProtocol);
```

在 Qt 文档中，给了 3 中模式，具体如何这里不做引申，读者需要请自己查询文档。但 `RfcommProtocol`，属于模拟 `RS232` 模式，我就叫串口模式了。

在上一节中说了，当双击 `ItemList` 控件中的项目时候，会进入到 `activated` 槽函数和蓝牙进行链接，那么如何连接呢？在 `itemList` 中会打印一个蓝牙的 `MAC` 地址信息，我们会将这个 `Mac` 地址保存在 `QBluetoothAddress` 这个类的实例化中，并将这个 `address` 传递给 `socket`，作为链接依据。

```

void Widget::itemActivated(QListWidgetItem *item)
{
    QString text = item->text();

    int index = text.indexOf(' ');

    if (index == -1)
        return;

    QBluetoothAddress address(text.left(index));
    QString name(text.mid(index + 1));
    qDebug() << "You has choice the bluetooth address is " << address;
    qDebug() << "The device is connneting.... ";
    QMessageBox::information(this, tr("Info"), tr("The device is connecting..."));
    socket->connectToService(address,
    QBluetoothUuid(serviceUuid) , QIODevice::ReadWrite);
}

```

我们通过对字符串的处理，将得到 **address** 信息。通过 **socket->connectToService(...)**，把地址，Uuid，和蓝牙模式传递进去，当执行完这句话的时候， 安卓手机开始和你

选择的蓝牙设备进行链接。

同样在 **socket** 中也提供了丰富的槽函数，比如成功建立连接信号，成功断开信号，这里在槽函数中可以做一些例子，这里给出例子：

connect (socket,

```

    SIGNAL (connected()),
    this,
    SLOT (bluetoothConnectedEvent ())
);
connect (socket,
    SIGNAL (disconnected()),
    this,
    SLOT (bluetoothDisconnectedEvent ())
);

```



```

void Widget::bluetoothConnectedEvent()
{
    // 2017/10/8 更新一下，请在这里插入关闭蓝牙查找服务，否则数据会断。
    // 具体语句是什么我忘记了，反正使用 discoveryAgent 的一个什么 close，或者 stop
    的方法

    qDebug() << "The android device has been connected successfully!";
}

```



```
    QMessageBox::information(this, tr("Info"), tr("successful connection!"));
}

void Widget::bluetoothDisconnectedEvent()
{
    qDebug() << "The android device has been disconnected successfully!";
    QMessageBox::information(this, tr("Info"), tr("successful disconnection!"));
}
```

最后，还有一个断开连接函数。通过断开连接按钮的槽函数实现。

```
void Widget::on_pushButton_disconnect_clicked()
{
    socket->disconnectFromService();
}
```

2.5) 发送和接受数据

蓝牙发送和接收数据，也是通过 **socket** 进行。发送数据十分简单：

```
void Widget::on_pushButton_send_clicked()
{
    QByteArray arrayData;
    QString s("Hello Windows!!!\nThis message is send via bluetooth of android device!\n");
    arrayData = s.toUtf8();
    socket->write(arrayData);
}
```

这里通过 **socket->write** 函数，完成发送。发送之后，上位机就可以接收到数据。
串口助手接收到信息

那么接收数据呢？

我们在构造函数中，需要建立这样的一个信号和槽的链接：

```
connect(socket,
        SIGNAL(readyRead()),
        this,
        SLOT(readBluetoothDataEvent())
);
```

readyRead()信号触发，跳进 **readBluetoothDataEvent** 中。

```
void Widget::readBluetoothDataEvent()
```

```
{  
  
    QByteArray line = socket->readAll();  
    QString strData = line.toHex();  
    comStr.append(strData);  
    qDebug() <<"rec data is: " << comStr;  
    qDebug() <<"The comStr length is: " << comStr.length();  
    if(comStr.length() >= 30) {  
  
        ui->textBrowser_info->append(comStr + "\n");  
        comStr.clear();  
    }  
  
}
```

5.3 接收机制

我们当前所有的执行，全部在接收机制里面，接收机制负责接收下位机的数据，下位机（树莓派）发送数据之后，APP 立刻响应判定命令是否为本机的命令，如果是本机的命令则快速执行命令。

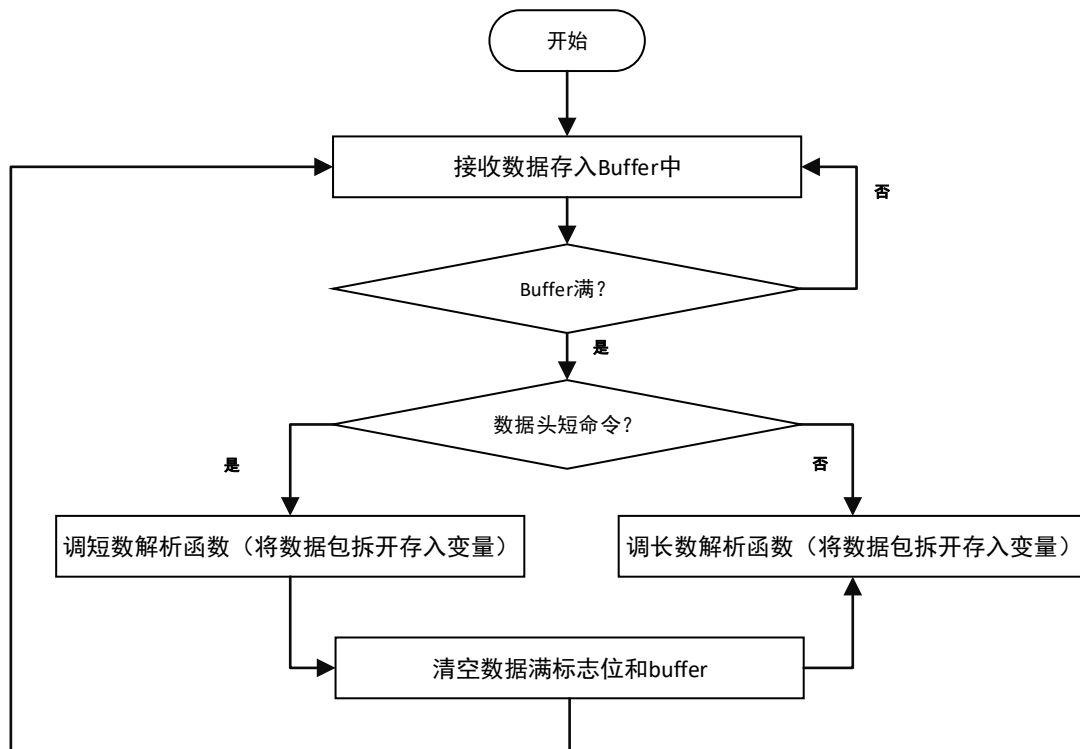


图 接收数据

接收机制中分为两种，一种是接收命令数据，命令数据通常只有几个字节，则称为短数据，另一种是图像数据，图像数据通常很长且字节大小不确定，这里称为长数据。数据种类的不同 APP 处理的方式就不同，为了甄别数据类型，我们在接收机制这一部分做了一些处理，在下面进行解说。

5.3.1 短数据连接

代码如下：

```
// 读取命令,则接受为命令数据 需要处理命令
if( rxDataBuffer->contains("@@@") ) {

    ui->textBrowser->append("a cmd signal.\n");
    array = rxDataBuffer->left( rxDataBuffer->indexOf("@@@") );
    rxString = array.toHex();
    qDebug() << " Rec cmd:" << rxString;
    // array 为整个命令栈
    // 判断 cmd 的数据头信息
    if( rxString.contains("aabb") ) {
        uint t_cmd;
        t_cmd = rxString.mid(4,2).toInt();
        qDebug() << " Rec cmd:" << t_cmd;
        switch( t_cmd ) {

            case CMD_ID_CONFIRM:

                QMessageBox::StandardButton reply;
                // 弹出身份确认按钮, 并获得用户的选择
                reply = QMessageBox::question(NULL, "认证身份确认", "设备请求确认身份, 确认吗?", QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
                // 如果选择 Yes 身份确认通过之后, 将允许信息发送给树莓派, 交给树莓派处理
                if( reply == QMessageBox::Yes ) {
                    // 如果选择 No 身份没有认定, 将信息发送给树莓派, 让树莓派来处理
                    SendCmdToRaspi( CMD_ID_CONFIRM_YES );
                }else {
                    SendCmdToRaspi( CMD_ID_CONFIRM_NO );
                }
                rxString.clear();
                rxDataBuffer->clear();
                array.clear();
                rxArray.clear();
                break;

            case CMD_THEFT_CHECKED:

                QMessageBox::warning(this,"Warring","设备检测到盗窃行为!");
                QPalette pal = ui->pushButton_alartLed->palette();
                pal.setColor( QPalette::Window,QColor(255,0,0) );
                ui->pushButton_alartLed->setPalette(pal);
                ui->label_stateText->setText("有人盗窃....");
                rxString.clear();
```

```

        rxDataBuffer->clear();
        array.clear();
        rxArray.clear();
        break;
    }
}
}

```

我们在让树莓派发送信息的时候使用了连续的@，“@@@”字符进行判定，如果收到的字符是“@@@”则代表为短数据，此时程序就会进入到接收状态，把后面的数据接收完毕，接收完毕之后对命令进行识别，识别之后就能拿到数据了，分别是 `switch case` 语句里面的，每一个 `case` 都是一种命令的代码。

5.3.2 长数据连接

```

// 如果发送命令包含 ###，则是图像文件 读图像数据
if( rxDataBuffer->contains("###") ) {

    array = rxDataBuffer->left(rxDataBuffer->indexOf("###"));

    ui->textBrowser->append("A pic signal... \n");
    //ui->textBrowser->append(array.toHex());
    bool flag = image.loadFromData((const uchar *)array.data(),array.size());
    if ( flag == true ) {

        ui->textBrowser->append("drawing....\n");
        QPixmap pixmap = QPixmap::fromImage( image );
        ui->imageLabel->setPixmap( pixmap );
        QMessageBox::information(this,"提示", "图像获取完毕。。");
    }else {
        ui->imageLabel->setText(" 获取图像失败，请重新获取！ ");
        QMessageBox::information(this,"提示", "图像获取失败。。");
    }
    // 清空缓冲区
    rxString.clear();
    rxDataBuffer->clear();
    array.clear();
    rxArray.clear();
    ui->textBrowser->append("Draw the pic!\n");
}

```

表 -1 表位记录

与短数据对应的，当接受的数据是“###”的时候，则识别为图像数据，我们就开始不断的接受数据，当一段时间没有数据的时候，判定为接受数据完毕，通过 Qt 提供的 QPixmap 的方法，将数据重组，并调用 showImage 的方法，把数据显示到图像显示区域去。

5.4 发送机制

通过 5.2 节蓝牙讲述，socket->write()函数，为写字节函数，我们通过 socket-write()函数完成命令的传送。我们建立了这个函数：

```
void SafeGuard::SendCmdToRaspi( uint cmd )
{
    QByteArray cmd_array;

    cmd_array.clear();

    cmd_array.append(0xAA);
    cmd_array.append(0xBB);
    cmd_array.append(cmd);
    cmd_array.append('@');
    socket->write(cmd_array);
}
```

在该函数中，写入 cmd 命令，然后这个函数会对命令进行编写，首先定义一个 cmd_array，然后不断的增加字符，0xAA 0xBB cmd 和结束符@，然后发送命令即可，当树莓派接收到这个信息，识别出 0xaa 0xbb 的时候，认定为这是 APP 发送过来的命令，当识别到@的时候，树莓派认定这个命令接收完毕，然后就开始进行命令识别执行。

在该项目中，主要用来发送清除报警状态、获取图像、手动报警和语音喊话的，在树莓派的 python 程序中，也可以看到对应的命令解析。


5.5 UI 设计



图 UI 设计

图为本项目的 UI 设计，使用的是 Qt 自带的 UI 组件，可以在 Qt 工程文件的 safeGuard.ui 文件中看到设计，在 Qt 中可以通过拖拽组件的方式完成 UI 设计。

6 GNU 协议声明

Copyright (c) 2017 MULTIBEANS. 

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".