



智能家居学习手册

广州粤嵌通信科技股份有限公司

作者	粤嵌教仪
日期	2013-11
版本	beta

第一章 智能家居-----	3
1.1 智能家居的需求-----	3
1.2 系统功能-----	3
1.1.1 中央控制系统-----	4
1.1.2 智能家居的功能-----	4
1.1.3 功能界面-----	5
1.3 开发流程-----	5
第二章 图形界面编程-----	5
2.1 QT 界面设计-----	5
2.1.1 建立工程-----	5
2.1.2 界面编程-----	7
2.1.2 编译运行-----	12
2.2 设计多界面-----	13
2.2.1 创建多界面类-----	13
2.2.2 界面编程-----	14
2.4 加载驱动-----	18
2.4.1 上位机操作-----	18
2.4.2 开发板操作-----	22
2.5 界面调用驱动-----	22
第三章 空调模块-----	24
3.1 图形界面设计-----	24
3.1.1 建立工作目录-----	24
3.1.2 建立 QT 工程-----	24
3.2 界面调用驱动-----	25
3.3 界面调用驱动-----	27
3.3.1 上位机操作-----	27
3.3.2 开发板操作-----	30
第四章 温湿度模块-----	31
4.1 图形界面设计-----	31
4.1.1 建立工作目录-----	31
4.1.2 界面设计-----	31
4.2 界面调用驱动-----	34
4.3 加载驱动-----	35
4.3.1 上位机操作-----	35
4.3.2 开发板操作-----	41
第五章 RFID 模块-----	42
5.1 图形界面设计-----	42
5.1.1 创建工程目录-----	42
5.1.2 创建工程-----	42
5.1.3 界面设计-----	43
5.2 源码实现-----	45
第六章 报警模块-----	50
6.1 创建工程-----	51
6.1.1 创建工程目录-----	51

6.1.2 创建工程-----	51
6.2 界面设计-----	52
6.3 源码实现-----	54
6.4 驱动实现-----	59
第七章 GPRS 模块-----	65
7.1.1 创建工程目录-----	65
7.1.2 创建工程-----	65
7.1.3 源码实现-----	66

第一章 智能家居

伴随着数字化和网络化的进程，智能化的浪潮席卷了世界的每一个角落，成为一种势不可挡的历史化大趋势。这一切的最终目的为人们提供一个以人为本的舒适、便捷、高效安全的生活环境。

如何建立一个高效率、高性能、低成本的智能家居系统是我们项目的主题。

1.1 智能家居的需求

智能家居系统的使用者，他们追求舒适的生活环境，所有的系统一定要稳定、所有的软件界面操作起来一定要友好。

智能家居系统有一个稳定可靠的中央控制系统，稳定的无线网络，通过这个无线网络可以控制多个智能模块。像一键触控的智能灯光模块、智能空调模块、智能窗帘模块、智能通讯模块；自动监控的智能温湿度模块、报警模块、RFID 模块等。

1.2 系统功能

智能家居根据用户的需求，如下图所示：中央控制系统通过无线网络，控制了智能灯光模块、智能温湿度模块、智能空调模块、智能窗帘模块、智能通讯模

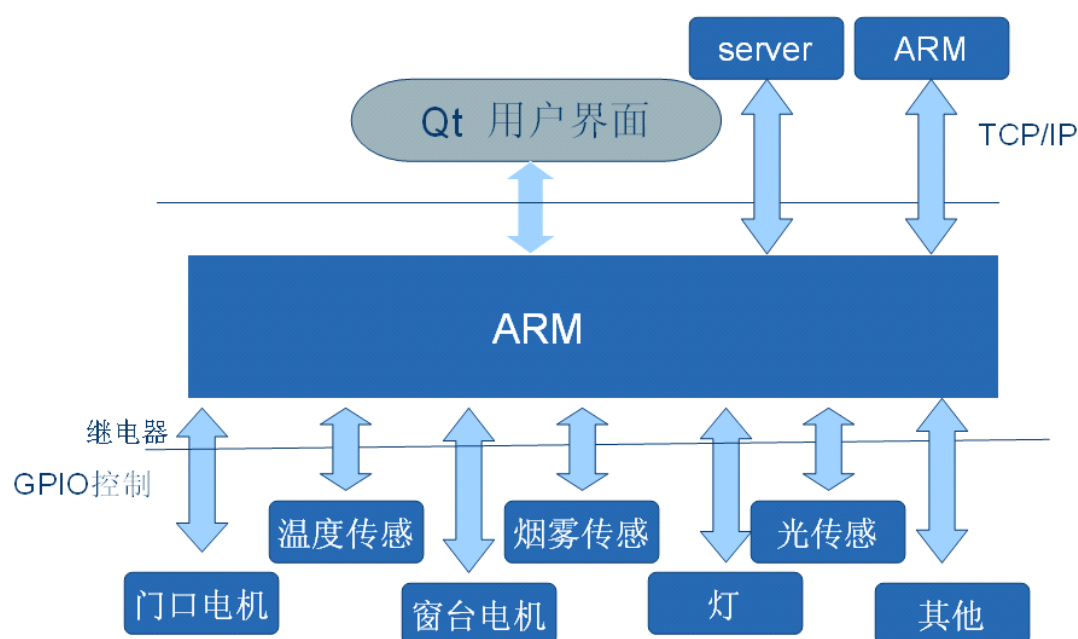


块、智能报警模块等多个功能模块。

1.1.1 中央控制系统

智能家居，需要一个使用 GEC2440 或者 GEC210 的开发板来设计一个中央控制系统，通过它来实现对室内电器的监控与控制。

这个中央控制系统，使用 Linux 作为系统内核、QT 作为用户界面、TCP/IP 协议来通信、GPIO 口来控制门口电机、温度传感器、窗台电机、烟雾传感器、灯、光传感器等功能模块。示意图如下所示：



1.1.2 智能家居的功能

智能家居要实现的功能：智能门禁、智能温湿度监控、智能灯光调节、智能空调、智能窗帘、RFID 模块、智能 GPRS 通讯、智能报警。

智能门禁：轻轻一点触摸屏上的按键，就可以驱动继电器开门。

智能温湿度监控：从 LCD 屏幕上，我们可以看到读取的温度和湿度。

智能灯光调节：进去灯光模块，轻轻一点触摸屏上的按键，就可以控制房间或者大厅的灯光。

智能空调：除了可以通过轻轻一点触摸屏上的按键实现空调的开关控制，也能和温湿度监控系统结合起来，在室内温度高于预设的温度时，自动开启。

智能窗帘：除了可以通过轻轻一点触摸屏上的按键实现窗帘的开关控制，也能根据光感电阻的变化，在室内光线高于预设值时，自动关闭。

RFID 模块：刷卡识别是不是房主。

智能 GPRS 通讯：可以实现通话和短信通知的功能，当房间的红外监控模块发现异常的事情发生时，会通过 GPRS 模块，将异常发送到房主的手机上。

智能报警：当发现火警或者异常的事情发生，报警器会发出警告，同时通过

GPRS 模块，通知房主，房间发生异常。

1.1.3 功能界面

所有的功能模块，都需要通过友好的界面来控制，那么，我们就要为客户设计一个容易操作的友好界面。

功能界面共分为：主界面、功能主界面、门禁界面、窗帘界面、空调界面、GPRS 设置界面、报警界面。

1.3 开发流程

简易的智能家居开发分为三个环节：构建家居的 Linux 系统、设计功能模块的驱动、设计图形界面。

构建智能家居的 Linux 系统的开发流程是：移植 uboot ---> 移植 Linux 内核 ---> 移植 Linux 根文件系统 ---> 移植 Linux 驱动，具体的操作步骤请查看：Linux 系统移植手册。

设计功能模块驱动：了解硬件 ---> 设计 Linux 驱动 ---> 加载驱动。具体的设计步骤请查看光盘目录中参考资料里所对应功能模块的源码和手册。

设计图形界面：编写 Qt 界面 ---> 移植 qte 到开发板 ---> 界面调用驱动 ---> 再次移植 qte 到开发板。具体的步骤请查看本手册以下内容。

第二章 图形界面编程

智能家居都是由多个界面切换而成的，所以首先我们要学会在 pc 机上面编写 qt 的界面，从单个界面到多个界面之间的切换，再到驱动调用和文件编译，接下来请跟着我们的文档提示，一步步地操作。

我们先从最简单的灯光模块开始。

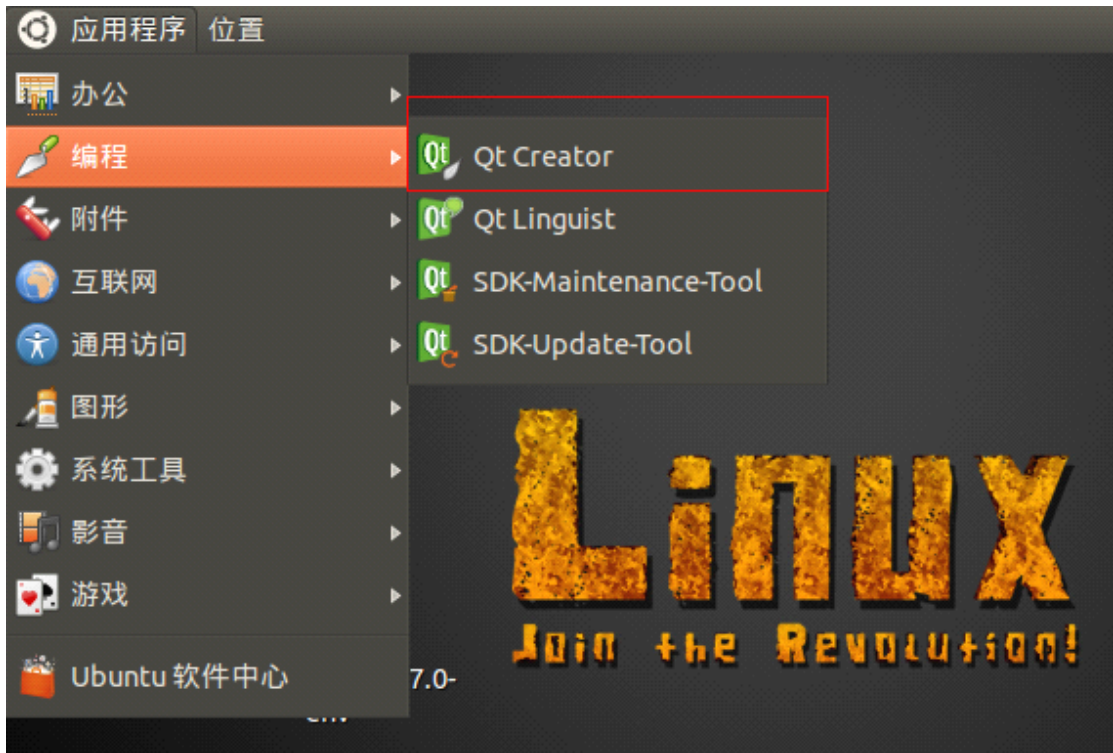
2.1 QT 界面设计

2.1.1 建立工程

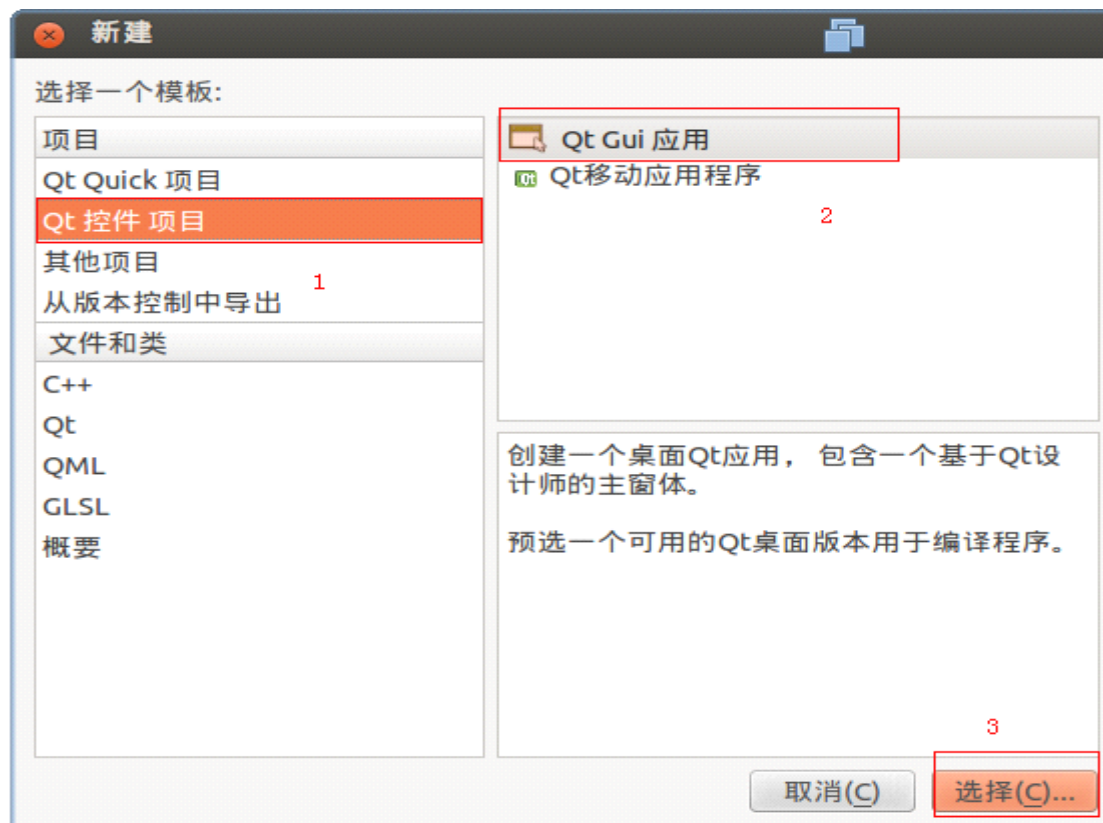
1. 建立工作目录

`mkdir /home/example/SmartHome`

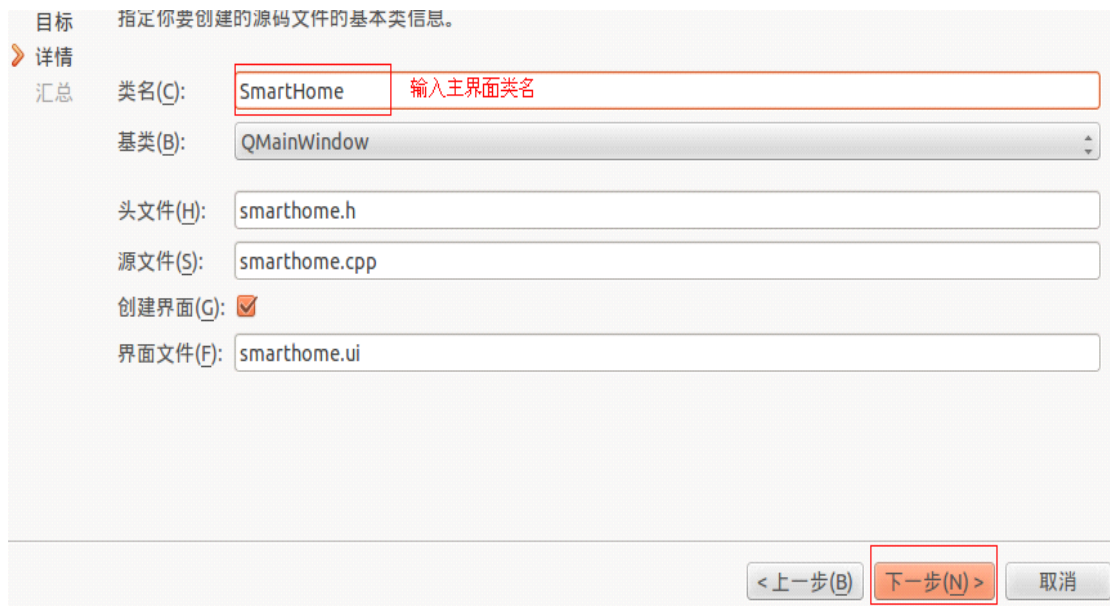
2. 打开 ubuntu 上已经装好的 Qt Creator 软件，如下图所示。如果还没装 Qt Creator 并且搭建好开发环境，请参考文档《GEC210 开发环境搭建.doc》。



3. 开始创建工程，并且将工程文件放在第一步创建的路径下。如下如所示：

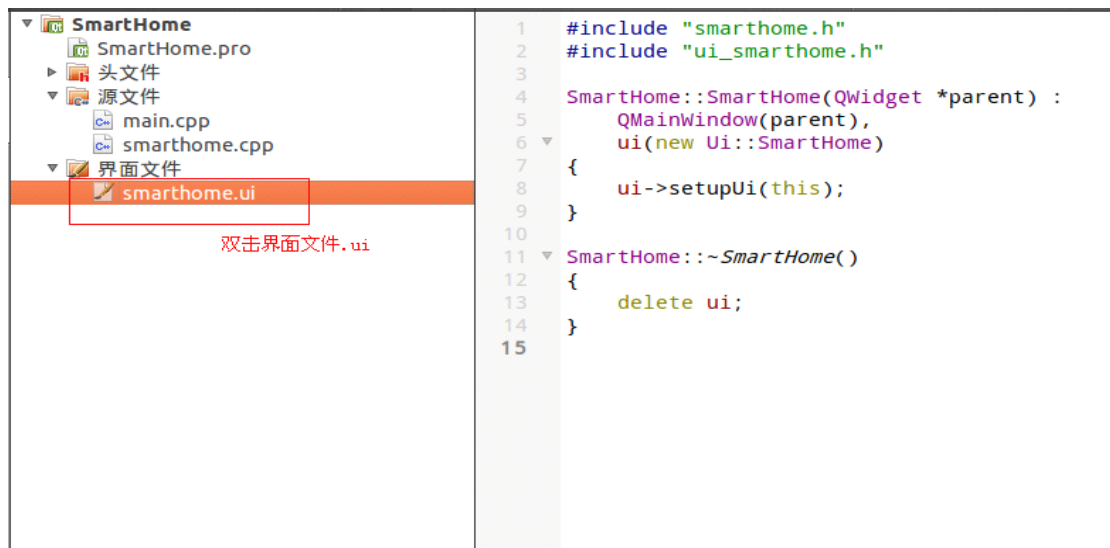


并且按默认，选择下一步。

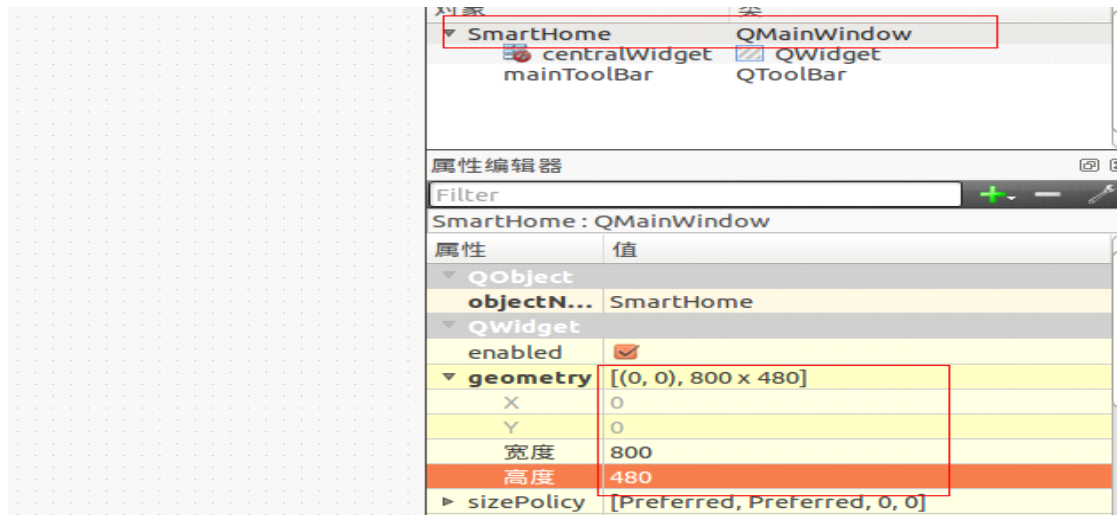


2.1.2 界面编程

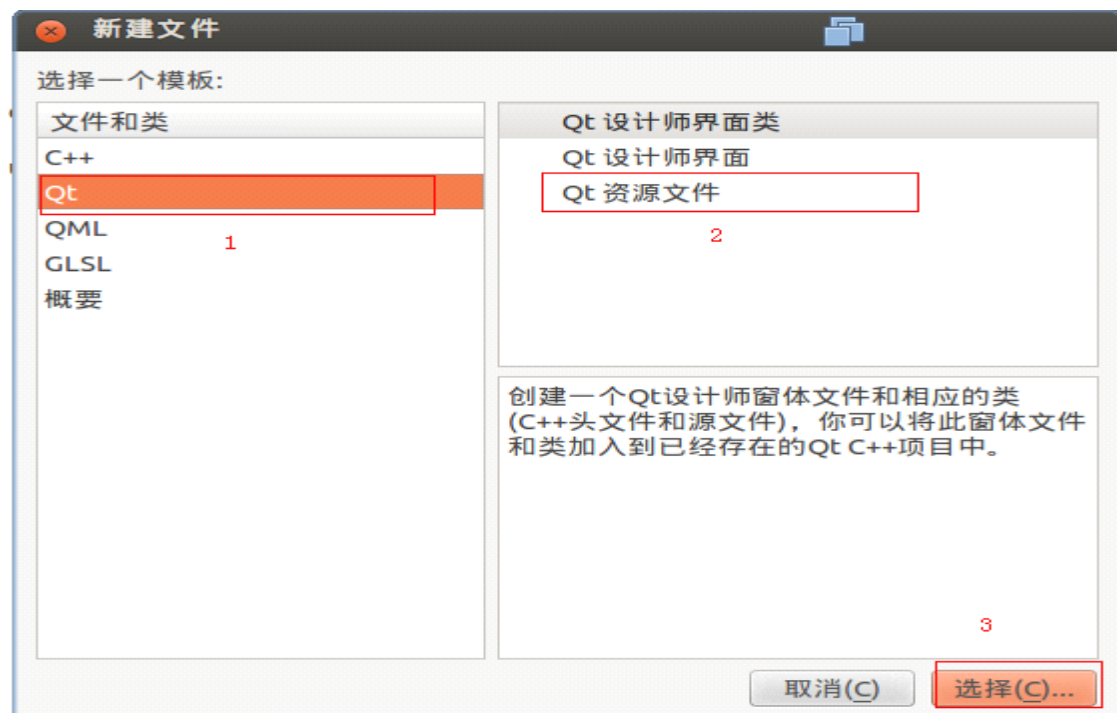
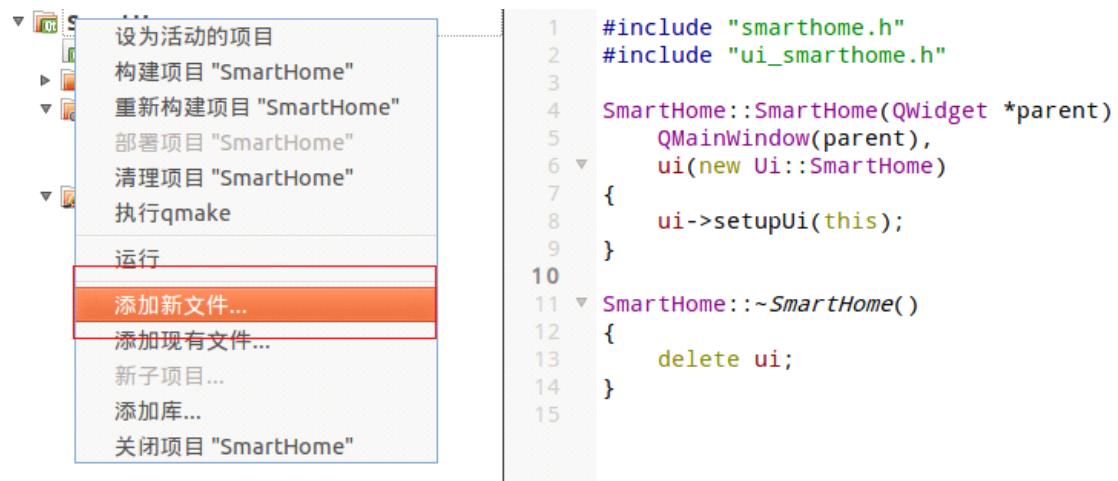
1、正式开始界面编程。首先我们要知道的是 210 开发板配套的屏幕大小是 7 寸，其分辨率是 800*480，因此我们要使主界面和屏幕往前匹配。于是，点击界面文件，如下如：



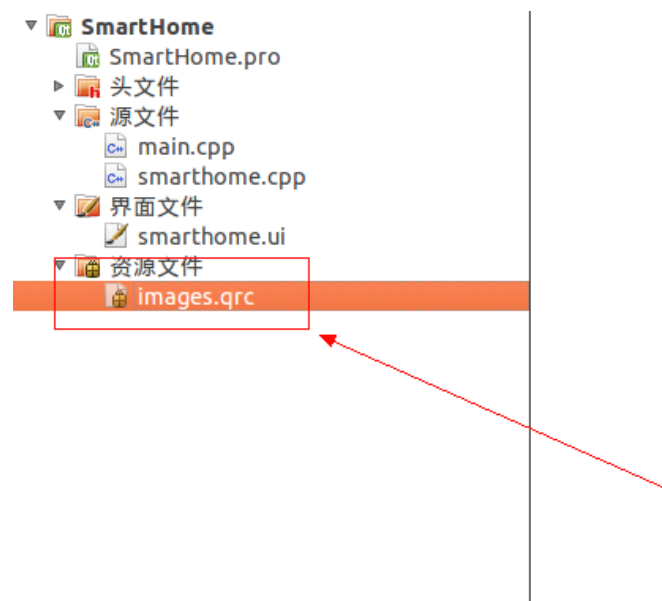
来到界面控制窗口，看到丰富的控件种类。但是我们要先调整界面大小吧，如下图：



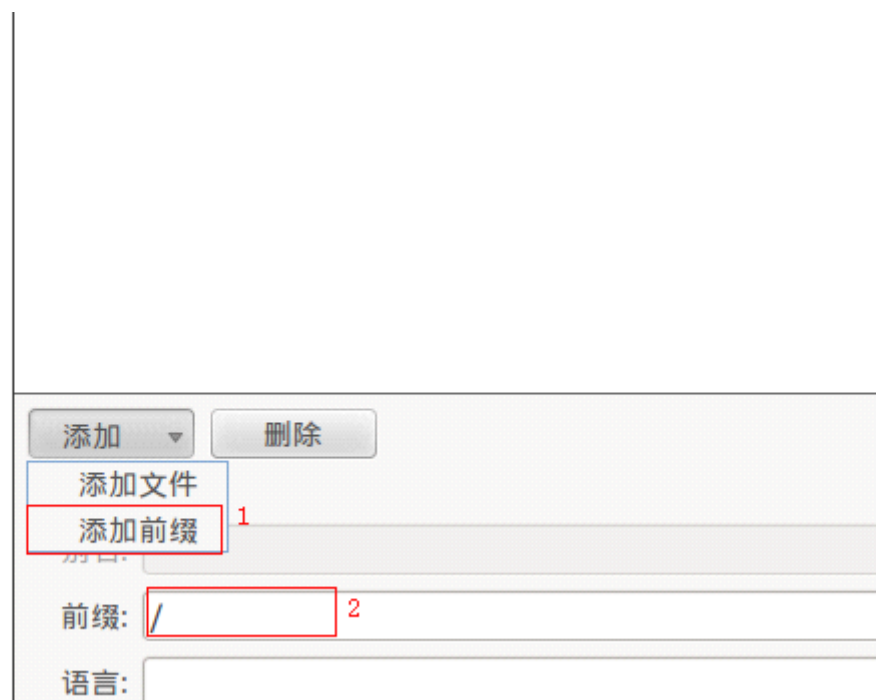
2、添加 Qt 资源文件。为了使界面更加的绚丽和增强交互性，我们必要性加入点资源文件，比如说图片。在工程右键，请看下图：



在此之后，要对资源文件命名。添加完成后会出现下图文件。



在双击此文件开始添加图片资源。首先要添加前缀，再是添加文件。



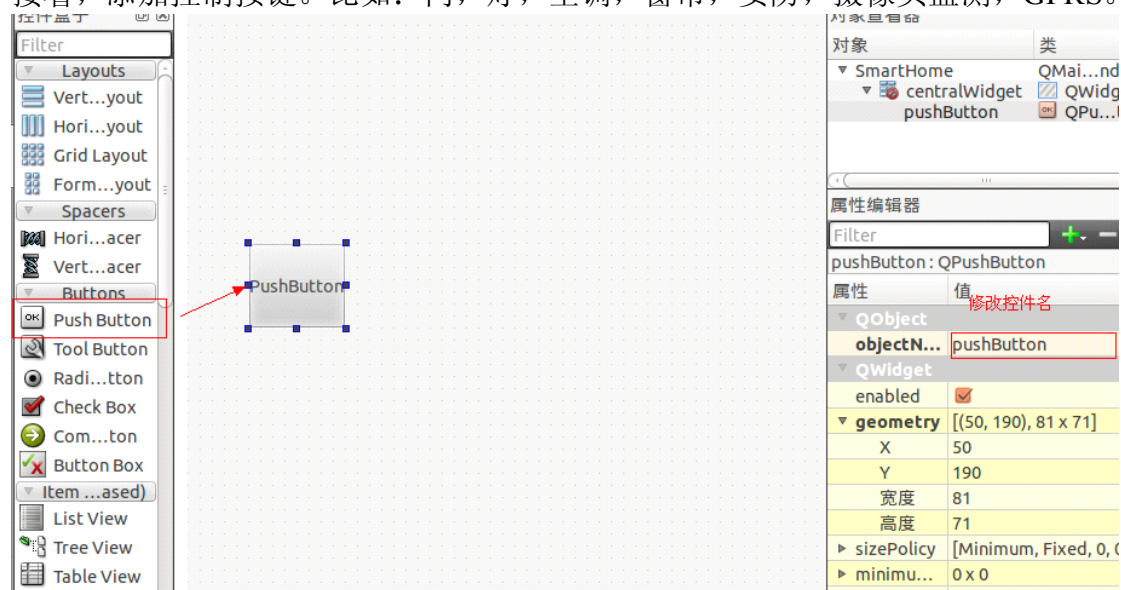
再次是添加文件，并且选择自己想选择的图片，推荐把图片统一放在当前工程目录下。

3、添加控件和资源。首先添加背景图片，双击进去界面文件，右键改变样式表。

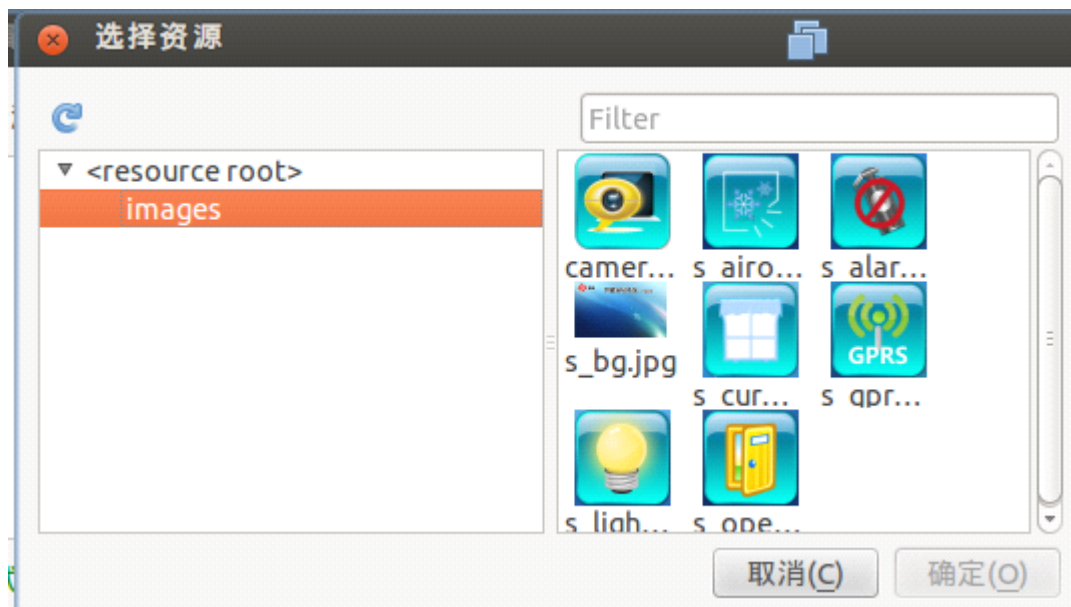


Ps:格式: QWidget#类名{background-image:url(:/图片.jpg);}

接着，添加控制按钮。比如：门，灯，空调，窗帘，安防，摄像头监测，GPRS。



拖动完空间后，并按照自己的喜好修改控件大小和控件名以及位置。同样可以拖进其他的 6 个控件。在控件上右键，改变样式表，如下图：



选择控件的图片，一般情况下，当控件显示图片情况下，为了界面更加绚丽，应当取消控件的静态文字显示，于是，双击控件，删除文字。其他控件改变图片方法一样。完成上述工作之后，请看下图：



在编译运行之前，应该加上一行代码，使界面和屏幕更加匹配。目的是去掉窗口的标题栏：

```
SmartHome::SmartHome(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::SmartHome)
{
    ui->setupUi(this);
    setWindowFlags(Qt::FramelessWindowHint); //设置全屏，去标题栏
}
```

2.1.2 编译运行

提示：此步骤的前提是要安装交叉编译链和交叉环境，如果还没安装，请参考文档《GEC210 开发环境搭建.doc》。务必也要搭建好 NFS 环境。

进到工程所在路径。

qmake -v 如果出现下图所示，证明环境安装成功

```
QMake version 2.01a
Using Qt version 4.7.0 in /usr/local/Trolltech/QtEmbedded-4.7.0-arm/lib
```

接着，

qmake //生成 Makefile

make

编译完成后，用 file 命令查看可执行文件

File SmartHome

```
SmartHome: ELF 32-bit LSB executable ARM, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.16, not stripped
```

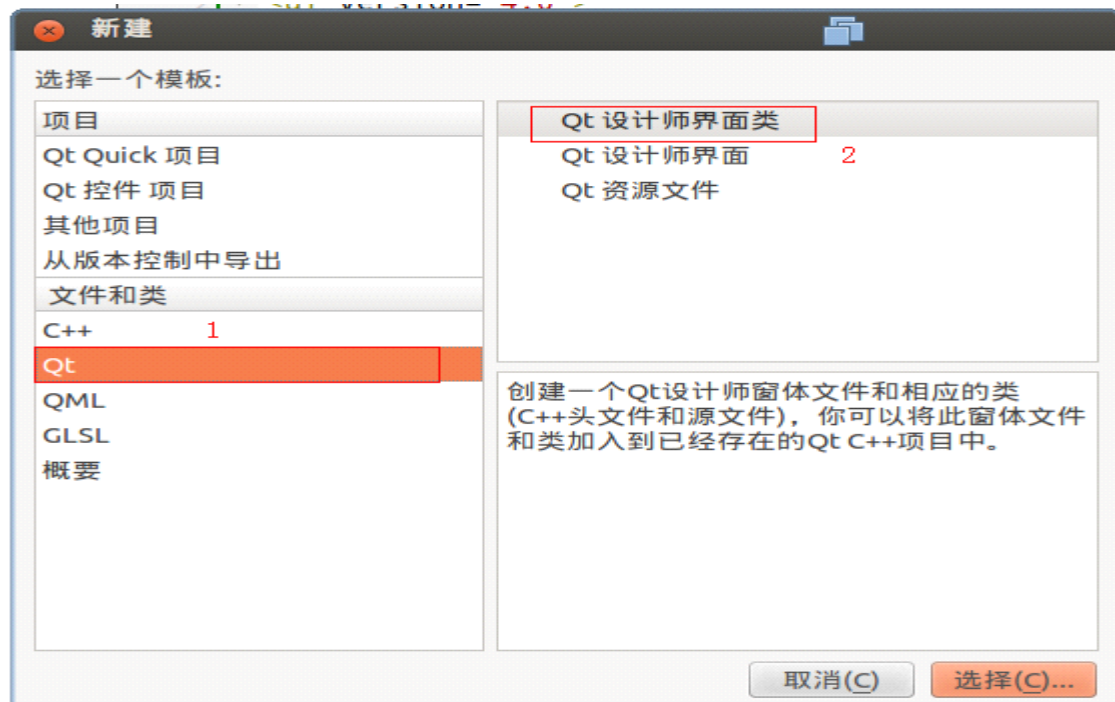
看到上图出现的“ARM”，证明已经成功了，那么试着到开发板上运行吧，可以通过 tftp，nfs（推荐 nfs）或者其他方法移植到 ARM 板上。



2.2 设计多界面

2.2.1 创建多界面类

Qt 编程里，一般来说一个界面肯定是独立的一个类，如果要创建多界面工程，必须创建多个类。在菜单栏上点击新建文件，如下图：

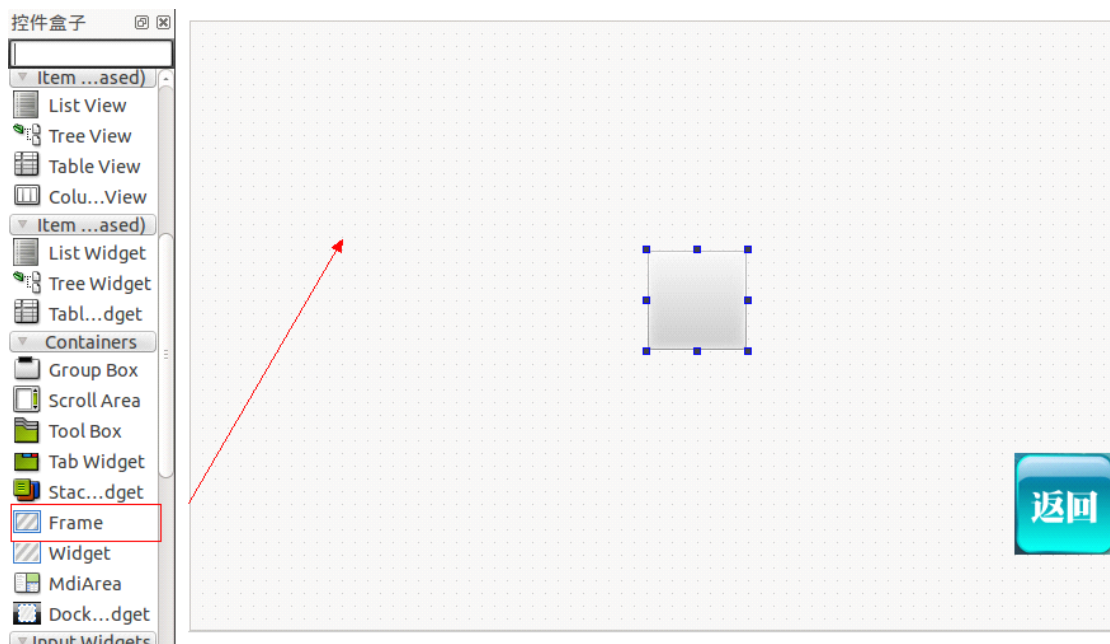




接着输入创建的类型名，这里以 door 为类名。

2.2.2 界面编程

如 2.1.2 一样，对界面文件 door.ui 进行编辑的话，会出现控件继承父窗口的图片，因为界面是一个 QWidget，因此，先拖入一个 QFrame 并调整和 QWidget 一样大小：



接着在 Frame 上改变样式表 `QFrame#frame{border-image: url(:/images/door/d_bg.jpg);}`，拖放控件，更改命名，调整大小，添加图片资源，

以及改变样式表对此界面装饰。可根据自己的喜好，完成此步骤有下图：



到这里我们已经创建好和做好界面设计，但是我们并没有在源码中实现调用此界面。在这步骤我们要实现的是：当我们点击最左边的门控制按钮的时候，会跳转到我们刚刚实现的 UI 界面那里。打开 2.1 中创建的界面文件 SmartHome.ui，在门控制按钮 `pushButton_door` 右键，选择转到槽，如下图：



并选择 `click()` 信号，也就是当控件发送 `click()` 信号的时候，会自动去执行此槽函数，槽函数的命名机制是根据控件的命令和信号的类型。如下图：

```
17 void SmartHome::on_pushButton_door_clicked()
18 {
19     |
20 }
21
```

现在就可以在这个函数实现想实现的操作。而我们就是为了实现界面跳转。思路是：当主界面的 `pushButton_door` 触发 `click()` 的时候，会显示 `door` 控制界面，同时主界面隐藏。而当在 `door` 控制界面的返回按钮 `pushButton_back` 触发 `click()` 的时候，发送一个信号给主界面，主界面收到这个信号就删除 `door` 对象，显示主界面。

首先在 `door.h` 上定义发给 `SmartHome` 的信号：

```
1  #ifndef DOOR_H
2  #define DOOR_H
3
4  #include <QWidget>
5
6  namespace Ui {
7      class door;
8  }
9
10 class door : public QWidget
11 {
12     Q_OBJECT
13
14 public:
15     explicit door(QWidget *parent = 0);
16     ~door();
17
18     signals:
19         void sig_GoBack_SmartHome();
20         //当点击返回按钮的时候，需要向SmartHome界面发送的信号，以便于实现界面返回
21     private slots:
22         void on_pushButton_back_clicked();
23
24 private:
25     Ui::door *ui;
```

然后去到 `door.ui` 的返回按钮槽并选择 `click()` 信号：

```
15     setWindowFlags(Qt::FramelessWindowHint);    //设置全屏，去标题栏
16 }
17
18 door::~door()
19 {
20     delete ui;
21 }
22
23 void door::on_pushButton_back_clicked()
24 {
25     emit sig_GoBack_SmartHome();
26 }
27
```

接着，在 `SmartHome.h` 添加头文件 `door.h`，并且声明 `door` 的对象和显示主界面的槽函数：


```

1 | #ifndef SMARTHOME_H
2 | #define SMARTHOME_H
3 |
4 | #include <QMainWindow>
5 | #include "door.h" //
6 | namespace Ui {
7 |     class SmartHome;
8 | }
9 |
10 | class SmartHome : public QMainWindow
11 | {
12 |     Q_OBJECT
13 |
14 | public:
15 |     explicit SmartHome(QWidget *parent = 0);
16 |     ~SmartHome();
17 |
18 | private slots:
19 |     void on_pushButton_door_clicked();
20 |     void ShowSmartHome(); //显示主界面的槽函数
21 |
22 | private:
23 |     Ui::SmartHome *ui;
24 |     door *DoorControl; //door的对象
25 | };
26 |
27 | #endif // SMARTHOME_H

```

并且实现 `pushButton_door` 的 `click()` 的函数体以及 `ShowSmartHome` 的函数体：

```

6 |     QMainWindow(parent),
7 |     ui(new Ui::SmartHome)
8 | {
9 |     ui->setupUi(this);
10 |    setWindowFlags(Qt::FramelessWindowHint); //设置全屏，去标题栏
11 | }
12 |
13 | SmartHome::~SmartHome()
14 | {
15 |     delete ui;
16 | }
17 |
18 | void SmartHome::ShowSmartHome()
19 | {
20 |     DoorControl = NULL;
21 |     delete DoorControl;
22 |     this->show();
23 | }
24 |
25 | void SmartHome::on_pushButton_door_clicked()
26 | {
27 |     DoorControl = new door;
28 |     connect(DoorControl,SIGNAL(sig_GoBack_SmartHome()),this,SLOT(ShowSmartHome()));
29 |     this->hide();
30 |     DoorControl->show();
31 | }
32 |
33 |

```

记住，和主界面一样，这个界面也要设置全屏模式，那么就可以保存编译了，大家看看效果吧。

2.4 加载驱动

2.4.1 上位机操作

建立工作目录

```
#mkdir /home/driver
```

```
#cd /home/driver
```

```
vim relay.c
```

```
#include <linux/kernel.h>
```

```
#include <linux/module.h>
```

```
#include <linux/miscdevice.h>
```

```
#include <linux/fs.h>
```

```
#include <linux/types.h>
```

```
#include <linux/moduleparam.h>
```

```
#include <linux/slab.h>
```

```
#include <linux/ioctl.h>
```

```
#include <linux/cdev.h>
```

```
#include <linux/delay.h>
```

```
#include <mach/gpio.h>
```

```
#include <mach/regs-gpio.h>
```

```
#include <plat/gpio-cfg.h>
```

```
#define DEVICE_NAME "relay" //驱动模块节点名字
```

```
static int relay_gpios[] = {  
    S5PV210_GPH3(3),  
};
```

```
#define RELAY_NUM ARRAY_SIZE(relay_gpios)
```

```
static long gec210_relay_ioctl(struct file *filp, unsigned int cmd,  
    unsigned long arg)  
{  
    switch(cmd) {  
        case 0:  
            if (arg > RELAY_NUM) {  
                return -EINVAL;  
            }  
            gpio_set_value(relay_gpios[arg], cmd);  
            break;
```

```

        case 1:
            if (arg > RELAY_NUM) {
                return -EINVAL;
            }

            gpio_set_value(relay_gpios[arg], cmd);
            //printk("DEVICE_NAME": %d %d\n", arg, cmd);
            break;

        default:
            return -EINVAL;
    }

    return 0;
}

static struct file_operations gec210_relay_dev_fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl = gec210_relay_ioctl,
};

static struct miscdevice gec210_relay_dev = {
    .minor          = MISC_DYNAMIC_MINOR,
    .name           = DEVICE_NAME,
    .fops           = &gec210_relay_dev_fops,
};

static int __init gec210_relay_dev_init(void) {
    int ret;
    int i;

    for (i = 0; i < RELAY_NUM; i++) {
        ret = gpio_request(relay_gpios[i], "RELAY");
        if (ret) {
            printk("%s: request GPIO %d for RELAY failed, ret = %d\n", DEVICE_NAME,
                relay_gpios[i], ret);
            return ret;
        }

        s3c_gpio_cfgpin(relay_gpios[i], S3C_GPIO_OUTPUT);
        gpio_set_value(relay_gpios[i], 1);
    }
}

```

```

    ret = misc_register(&gec210_relay_dev);

    printk(DEVICE_NAME"\tinitialized\n");

    return ret;
}

static void __exit gec210_relay_dev_exit(void) {
    int i;

    for (i = 0; i < RELAY_NUM; i++) {
        gpio_free(relay_gpios[i]);
    }

    misc_deregister(&gec210_relay_dev);
}

module_init(gec210_relay_dev_init);
module_exit(gec210_relay_dev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gec Lab.");

```

#vim Makefile

修改红色部分

CONFIG_HOUSELED ?=m

ifneq (\$(KERNELRELEASE),)

 #relay-objs:=relay.o //编译的模块的命名

 obj-\$(CONFIG_HOUSELED)+=relay.o

else

PWD :=\$(shell pwd)

KERN_VER = \$(shell uname -r)

#KERN_DIR = /lib/modules/\$(KERN_VER)/build

YOUR CROSS COMPILE KERNEL DIR

KERN_DIR = /home/kernel/linux-2.6.35.7-gec-v3.0 //注意这里，我的 linux 源代码是在/home/kernel 下，这里的源码必须是已经编译过的源码

modules:

 \$(MAKE) -C \$(KERN_DIR) M=\$(PWD) modules

endif

clean:

 rm -rf *.o *~core .depend *.cmd *.ko *.mod.c *.tmp_versions

进行编译

`#make`

得到.ko 文件，这个文件就是内核模块。

测试程序也是在/home/driver/创建

`vim relay_test.c`

```
#define RELAY_DEVICE_FILENAME "/dev/relay"
#define RELAY_ON 0
#define RELAY_OFF 1

int main(void)
{
    int val = 0;
    int fd;
    fd = open(RELAY_DEVICE_FILENAME, O_RDWR);

    while(val != 3)
    {
        printf("please select number to run program\n");
        printf("1:RELAY on\n2:RELAY off\n");
        scanf("%d",&val);
        if(val==1)
        {
            ioctl(fd, RELAY_ON, 0);
            printf("RELAY_ON\n");
        }
        else if(val==2)
        {
            ioctl(fd, RELAY_OFF, 0);
            printf("RELAY_OFF\n");
        }
    }
    close(fd);
    return 0;
}
```

目录下，进行交叉编译

`#arm-linux-gcc -o relay_test relay_test.c`

2.4.2 开发板操作

把内核模块 relay.ko 和编译好的测试程序 relay_test 通过 nfs 挂载到开发板(这部分在文档《GEC210 开发环境搭建.doc》里面有介绍)

把开发板和智能家居底板连好线

在开发板终端上测试

```
#chmod 777 relay.ko
```

```
#chmod 777 relay_test
```

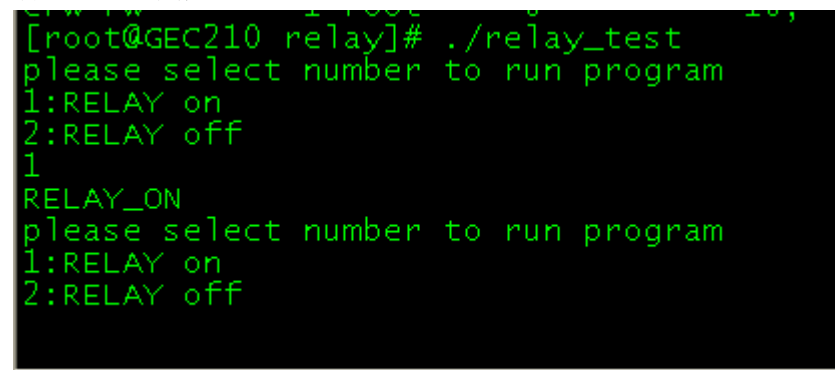
加载内核模块

```
#insmod relay.ko
```

改驱动模块是已经自动创建该设备节点了，这时候我们可以测试驱动了

```
# ./relay_test
```

有以下打印信息



```
[root@GEC210 relay]# ./relay_test
please select number to run program
1:RELAY on
2:RELAY off
1
RELAY_ON
please select number to run program
1:RELAY on
2:RELAY off
```

可以看到，输入 1 为打开继电器，2 为关闭继电器。在此过程可以听到继电器的声音。

2.5 界面调用驱动

在开始这章的实验前，要先认真照着上面两章：图形界面界面开发和 linux 驱动开发的步骤，完成图形界面和 Linux 驱动代码的编写。那么接下来看源码实现：修改 door.cpp,提示：红色代码为修改代码

```
#include "door.h"
#include "ui_door.h"
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/types.h>
```

```
#include <sys/ioctl.h>
```

```
const int RELAY_ON = 0;
```

```
const int RELAY_OFF = 1;
```

```
const char *RELAY_DEV = "/dev/relay";
```

```
door::door(QWidget *parent) :
```

```
    QWidget(parent),
```

```
    ui(new Ui::door)
```

```
{
```

```
    ui->setupUi(this);
```

```
    setWindowFlags(Qt::FramelessWindowHint);    //设置全屏，去标题栏
```

```
    fd = open(RELAY_DEV, O_RDWR);
```

```
    if(fd<0)
```

```
        qDebug()<<"Relay dev open fail!";
```

```
}
```

上面为构造函数，打开设备，那么下面就是控制了，来到控制按钮 pushButton_door 的槽函数：

```
void door::on_pushButton_door_clicked()
```

```
{
```

```
    static bool on_off_flag = false;
```

```
    if(!on_off_flag)
```

```
    {
```

```
        on_off_flag = true;
```

```
        ioctl(fd, RELAY_ON, 0);
```

```
        ui->pushButton_door->setStyleSheet(QString::fromUtf8("border-image: url(./images/door/d_close.png);"));
```

```
    }
```

```
    else
```

```
    {
```

```
        on_off_flag = false;
```

```
        ioctl(fd, RELAY_OFF, 0);
```

```
        ui->pushButton_door->setStyleSheet(QString::fromUtf8("border-image: url(./images/door/d_open.png);"));
```

```
    }
```

```
}
```

同时

```
void door::on_pushButton_back_clicked()
```

```
{
```

```
    ::close(fd);
```

```
    emit sig_GoBack_SmartHome();
```

```
}
```

到这里，就可以编译运行了，看看效果吧！

第三章 空调模块

有了前一章的基础，那么我们可以开始智能家居空调模块的设计工作，设计空调模块的方法与灯光模块相似，也是从编写单个界面开始，加载驱动，调用驱动，编译文件，完成空调模块的设计。

3.1 图形界面设计

3.1.1 建立工作目录

```
mkdir /home/example/AirCondition
cd /home/example/AirCondition
```

3.1.2 建立 QT 工程

和之前建立工程一样，并将工程路径设在/home/example/AirCondition



然后双击进去.ui 界面文件进行界面设计。这个步骤包括：大小调整，添加资源文件，添加图片，拖放控件，设置背景图片等等，可参考前面第二章。先是资源文件：



接着是对.ui 文件右键改变样式表设置背景等的设计。完成设计之后：



3.2 界面调用驱动

首先增加头文件

在 `airconditon.cpp` 中增加相对应的头文件

```
#include <sys/ioctl.h>
```

```
#include <fcntl.h>
```

```
const int MOTOR_ON=1;
const int MOTOR_OFF=0;
```

```
AirCondition::AirCondition(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::AirCondition)
{
    ui->setupUi(this);
    Getmotorflag = false;
    setWindowFlags(Qt::FramelessWindowHint);    //去标题栏
    motorfd = open("/dev/dc_motor",O_RDWR);      //直流电击
    if(motorfd < 0)
    {
        qDebug("open motorfd err...");
    }
}
```

控制空调的按键 pushButton_onoff 的槽函数：如下

```
void AirCondition::on_pushButton_onoff_clicked()
{
    if(Getmotorflag)
    {
        ioctl(motorfd,MOTOR_OFF,0);
        ui->pushButton_onoff->setIcon(QIcon(":/images/air_x.png"));
        ui->pushButton_air->setIcon(QIcon(":/images/air_close.png"));
        Getmotorflag = false;
    }
    else
    {
        ioctl(motorfd,MOTOR_ON,0);
        ui->pushButton_onoff->setIcon(QIcon(":/images/air_o.png"));
        ui->pushButton_air->setIcon(QIcon(":/images/air_open.png"));
        Getmotorflag = true;
    }
}
```

```
}
AirCondition::~AirCondition()
{
    ::close(motorfd);
    delete ui;
}
```

那么到这里就基本实现了对空调的控制，但是自动控制模块我们还没有实现。那么就先来测试一下吧！编译传送到 210 开发板上。

3.3 界面调用驱动

3.3.1 上位机操作

vim moto.c

开始编程驱动模块

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/moduleparam.h>
#include <linux/slab.h>
#include <linux/ioctl.h>
#include <linux/cdev.h>
#include <linux/delay.h>
```

```
#include <mach/gpio.h>
#include <mach/regs-gpio.h>
#include <plat/gpio-cfg.h>
```

```
#define DEVICE_NAME "dc_motor"
```

```
static int motor_gpios[] = {
    S5PV210_GPH3(1),
    S5PV210_GPH2(3),
};
```

```
#define MOTOR_NUM    ARRAY_SIZE(motor_gpios)
```

```
static void motor_init(void)
{
    gpio_set_value(motor_gpios[0], 0);
    gpio_set_value(motor_gpios[1], 0);
    //s3c_gpio_setpull(S5PV210_GPH3(1), 0);
    //s3c_gpio_setpull(S5PV210_GPH2(2), 0);
}
static void motor_foreward(void)
{
    gpio_set_value(motor_gpios[0], 1);
```

```

        gpio_set_value(motor_gpios[1], 0);
    }

static void motor_rollback(void)
{
    gpio_set_value(motor_gpios[1], 1);
    gpio_set_value(motor_gpios[0], 0);
}

static int gec210_motor_open(struct inode *inode, struct file *filp)
{
    printk(DEVICE_NAME ":open\n");
    motor_init();
    return 0;
}

static long gec210_motor_ioctl(struct file *filp, unsigned int cmd,
                               unsigned long arg)
{
    switch(cmd) {
        case 0:
            if (arg > MOTOR_NUM) {
                return -EINVAL;
            }
            motor_init();
            printk("Motor Stop.\n");
            break;

        case 1:
            if (arg > MOTOR_NUM) {
                return -EINVAL;
            }
            motor_rollback();
            printk("Motor Rollback.\n");
            break;

        case 4:
            if (arg > MOTOR_NUM) {
                return -EINVAL;
            }
            motor_foreward();
            printk("Motor Foreward.\n");
            break;

        default:

```

```

        return -EINVAL;
    }

    return 0;
}

static struct file_operations gec210_motor_dev_fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl = gec210_motor_ioctl,
};

static struct miscdevice gec210_motor_dev = {
    .minor          = MISC_DYNAMIC_MINOR,
    .name           = DEVICE_NAME,
    .fops           = &gec210_motor_dev_fops,
};

static int __init gec210_motor_dev_init(void) {
    int ret;
    int i;

    for (i = 0; i < MOTOR_NUM; i++) {
        ret = gpio_request(motor_gpios[i], "MOTOR");
        if (ret) {
            printk("%s: request GPIO %d for MOTOR failed, ret = %d\n",
DEVICE_NAME,
                motor_gpios[i], ret);
            return ret;
        }

        s3c_gpio_cfgpin(motor_gpios[i], S3C_GPIO_OUTPUT);
        gpio_set_value(motor_gpios[i], 0);
    }
    gpio_set_value(motor_gpios[0], 0);
    gpio_set_value(motor_gpios[1], 0);

    ret = misc_register(&gec210_motor_dev);

    printk(DEVICE_NAME"\tinitialized\n");

    return ret;
}

static void __exit gec210_motor_dev_exit(void) {

```

```

    int i;

    for (i = 0; i < MOTOR_NUM; i++) {
        gpio_free(motor_gpios[i]);
    }

    misc_deregister(&gec210_motor_dev);
}

module_init(gec210_motor_dev_init);
module_exit(gec210_motor_dev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gec Lab.");

```

3.3.2 开发板操作

```

#vim Makefile
修改红色部分
CONFIG_HOUSELED ?=m
ifneq ($(KERNELRELEASE),)
    # dc_motor -objs:= dc_motor.o //编译的模块的命名
    obj-$(CONFIG_HOUSELED)+= dc_motor.o
else
    PWD :=$(shell pwd)
    KERN_VER = $(shell uname -r)
    #KERN_DIR = /lib/modules/$(KERN_VER)/build
    # YOUR CROSS COMPILE KERNEL DIR
    KERN_DIR = /home/kernel/linux-2.6.35.7-gec-v3.0 //注意这里，我的 linux 源代码是在/home/kernel 下，这里的源码必须是已经编译过的源码
modules:
    $(MAKE) -C $(KERN_DIR) M=$(PWD) modules
endif
clean:
    rm -rf *.o *~core .depend *.cmd *.ko *.mod.c *.tmp_versions

进行编译
#make

```

如前面内容 2.4.1 操作一样，编译出.ko 文件。并传送到 210 板上，执行命令：
insmod motor.ko

```
[root@GEC210 driver]# ls /dev/dc_motor  
/dev/dc_motor  
[root@GEC210 driver]#
```

就可以成功生成节点了！那么现在就执行刚刚完成的可执行程序吧：

```
[root@GEC210 AirCondition]# ./AirCondition -qws
```

这个时候触摸屏幕控制直流点击吧。

第四章 温湿度模块

智能家居的开始界面的右下方有一个显示温湿度的表，大家一定很好奇这个温湿度的表是如何做出来，这一章就来满足大家的好奇心，认真完成这一章的学习，你将编写出属于自己的温湿度模块。

4.1 图形界面设计

4.1.1 建立工作目录

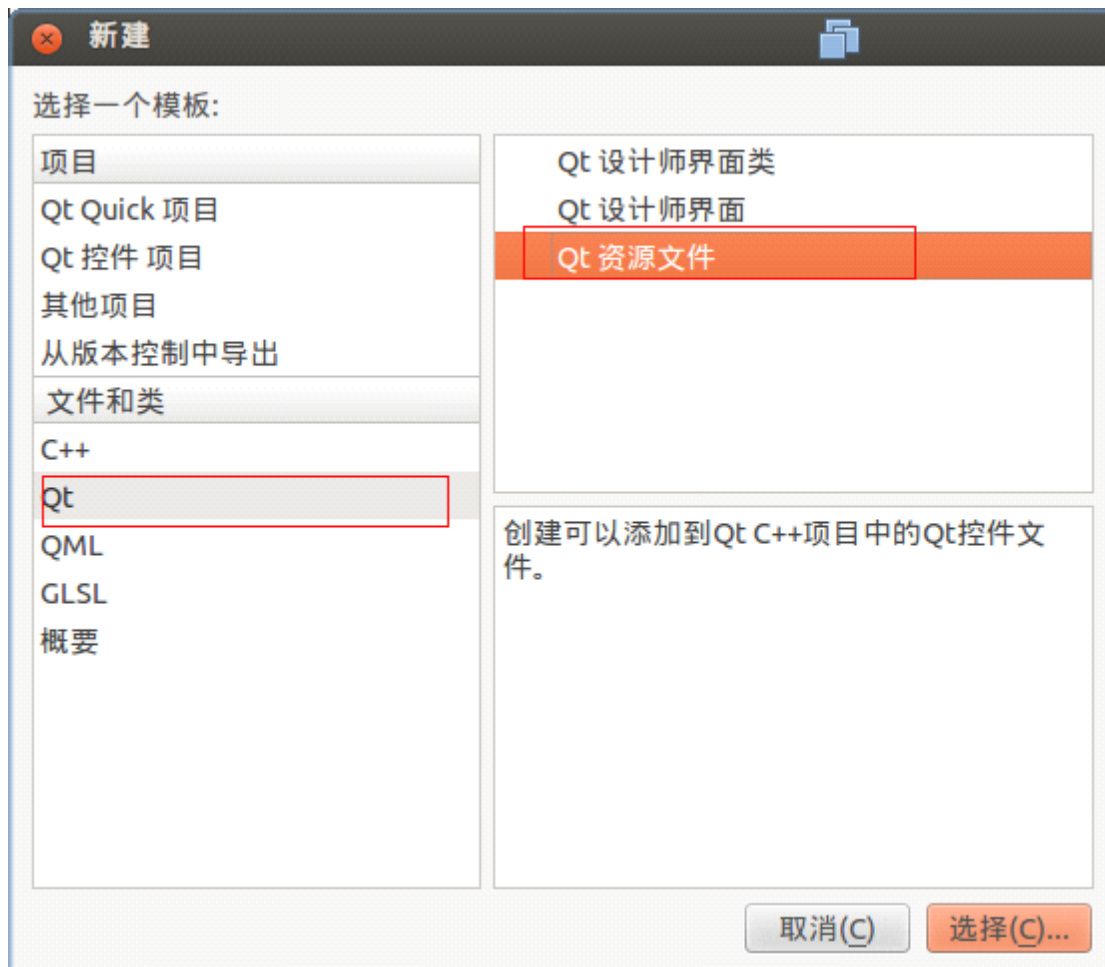
```
mkdir /home/example/humidity  
cd /home/example/humidity
```

4.1.2 界面设计

首先创建工程



添加资源文件



添加前缀，添加文件：



然后针对界面文件设计，完成效果图：



4.2 界面调用驱动

增加头文件

在 humidity.cpp 中增加相对应的头文件

```
#include <fcntl.h>
```

```
#include <sys/ioctl.h>
```

在构造函数中，

```
Humidity::Humidity(QWidget *parent) :
```

```
    QMainWindow(parent),
```

```
    ui(new Ui::Humidity)
```

```
{
```

```
    ui->setupUi(this);
```

```
    setWindowFlags(Qt::FramelessWindowHint);    //去标题栏
```

```
    hdfd = ::open("/dev/humidity",0);            //温湿度
```

```
    if(hdfd < 0)
```

```
        qDebug("Open humidity err...");
```

```
    showtimer = new QTimer;
```

```
    connect(showtimer,SIGNAL(timeout()),this,SLOT(showhumidity()));
```

```
    showtimer->start(2000);
```

```
}
```

接着是显示的函数，这里是一个定时器槽函数，读取温湿度并且显示出来：

```
void Humidity::showhumidity()
```

```
{
```

```
    QString humidity;
```

```
    QString temp;
```

```
    if (read(hdfd,&temperature,sizeof( temperature ))==0)
```

```
    {
```

```

        humidiyz = (temperature & 0xff000000)>>24; //湿度整数
        humidiyx = (temperature & 0x00ff0000)>>16; //湿度小数
        tempz = (temperature & 0x0000ff00)>>8;    //温度整数
        tempx = (temperature & 0x000000ff);        //温度小数

        qDebug("humidiyz = %d tempz = %d",humidiyz,tempz);
        humidity = QString("%1.%2").arg(humidiyz).arg(humidiyx);
        temp = QString("%1.%2").arg(tempz).arg(tempx);
        ui->lineEdit_humidity->setText(humidity+" %");
        ui->lineEdit_temp->setText(temp+ "C");
    }
}

Humidity::~Humidity()
{
    ::close(hdfd);
    delete ui;
}

```

4.3 加载驱动

4.3.1 上位机操作

```

vim humidity.c
开始编程温湿度驱动模块
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/clock.h>
#include <linux/wait.h>
#include <linux/sched.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

#include <asm/io.h>
#include <asm/irq.h>

```

```
#include <asm/uaccess.h>
#include <mach/regs-adc.h>
#include <mach/regs-gpio.h>
#include <plat/gpio-cfg.h>
#include <mach/gpio.h>
```

```
#define DEVICE_NAME "humidity"
```

```
//XEINT24/KP_ROW0/GPH3_0
```

```
unsigned long receive_value;
```

```
unsigned long receive_jy;
```

```
int data_in(void)
```

```
{
    s3c_gpio_cfgpin(S5PV210_GPH3(0), S3C_GPIO_INPUT);
    return gpio_get_value(S5PV210_GPH3(0));
}
```

```
void data_out(int data)
```

```
{
    s3c_gpio_cfgpin(S5PV210_GPH3(0), S3C_GPIO_OUTPUT);
    gpio_set_value(S5PV210_GPH3(0), data);
}
```

```
void read_data(void)
```

```
{
    unsigned int flag = 0;
    unsigned int u32i = 0;

    data_out(0);
    mdelay(20);
    data_out(1);
    udelay(40);
    if (data_in() == 0)
    {
        while( (data_in() == 0) && ((flag++)<50000) );
        flag = 0;
        receive_value = 0;
        receive_jy = 0;
        while( data_in() != 0 );
        for (u32i=0x80000000; u32i>0; u32i>>=1)
        {
            flag = 0;
```

```

        while( (data_in() == 0) && ((flag++)<50000) );
        flag = 0;
        while( (data_in() != 0) && ((flag)<500))
        {
            udelay(10);
            flag++;
        }

        if(flag > 5)
        {
//            printk("flag 1= %d\n",flag);
            receive_value |= u32i;
        }
    }
//    printk("flag 0= %d\n",flag);
    for (u32i=0x80; u32i>0; u32i>>=1)
    {
        flag = 0;
        while( (data_in() == 0) && ((flag++)<50000) );
        flag = 0;
        while( (data_in() != 0) && ((flag)<500))
        {
            udelay(10);
            flag++;
        }

        if(flag > 5)
        {
            receive_jy |= u32i;
        }
    }
}
}

```

```

void humidity_read_data(void)
{
    unsigned int flag = 0;
    unsigned int u32i = 0;
    receive_value = 0;
    receive_jy = 0;
    data_out(0);
    mdelay(20);
    data_out(1);
}

```

```

    udelay(40);
    if (data_in() == 0)
    {
        flag = 0;
        while(data_in() == 0) //响应 80us 的低电平
        {
            udelay(10);
            flag++;
            if(flag > 10)
                return;//超过 100us，器件还是没有响应，认为其出现问题
        }
        printk("80us low flag=%d\n",flag);
        flag = 0;
        while(data_in() == 1) //响应 80us 的高电平
        {
            udelay(10);
            flag++;
            if(flag > 10)
                return;//超过 100us，器件还是没有响应，认为其出现问题
        }
        printk("80us high flag=%d\n",flag);
        flag = 0;

        for (u32i=0x80000000; u32i>0; u32i>>=1)
        {
            flag = 0;
            while(data_in() == 0) //响应 50us 的低电平 ,开始接收
            {
                udelay(10);
                flag++;
                if(flag > 10)
                    break;
            }

            flag = 0;
            while( data_in() == 1) //响应高电平， 持续高电平时间最多不超过
            {
                udelay(10);
                flag++;
                if(flag > 10)
                    break;
            }
        }
    }
}

```

数据

70us

```

    }

    if(flag > 5) //低电平持续时间为 26us-28us 为数据 0 高电平持续时间为 70us 为数据 1
    {
        receive_value |= u32i;
    }
}

#ifdef 1
//      printf("flag 0= %d\n",flag);
for (u32i=0x80; u32i>0; u32i>>=1)
{
    flag = 0;
    while(data_in() == 0) //响应 50us 的低电平 ,开始接收
    {
        udelay(10);
        flag++;
        if(flag > 10)
            break;
    }

    flag = 0;
    while( data_in() == 1) //响应高电平, 持续高电平时间最多不超过
    {
        udelay(10);
        flag++;
        if(flag > 10)
            break;
    }

    if(flag > 5) //低电平持续时间为 26us-28us 为数据 0 高电平持续时间为 70us 为数据 1
    {
        receive_jy |= u32i;
    }
}
#endif
}
}

```

```

static ssize_t gec210_humidiy_read(struct file *file, char __user *buf, size_t size,

```

```

loff_t *off)
{
    unsigned char tempz = 0;
    unsigned char tempx = 0;
    unsigned char humidityz = 0;
    unsigned char humidityx = 0;
    unsigned char ecc,jy;

    humidity_read_data();

    humidityz = (receive_value & 0xff000000)>>24;
    humidityx = (receive_value & 0x00ff0000)>>16;
    tempz = (receive_value & 0x0000ff00)>>8;
    tempx = (receive_value & 0x000000ff);
    jy = receive_jy & 0xff;

    ecc = humidityz + humidityx + tempz + tempx;
//    printk("=====ecc=%0x jy=%0x \n",ecc,jy);
    if(ecc != jy)
        return -EAGAIN;
    copy_to_user(buf,&receive_value,sizeof(receive_value));
    return 0;
}

static int gec210_humidiy_open(struct inode *inode, struct file *file)
{
    printk("open in kernel\n");
    return 0;
}

static void gec210_humidiy_release(struct inode *inode, struct file *file)
{
    printk("");
}

static struct file_operations gec210_humidity_dev_fops = {
    .owner          = THIS_MODULE,
    .open = gec210_humidiy_open,
    .read = gec210_humidiy_read,
    .release = gec210_humidiy_release
};

static struct miscdevice gec210_humidity_dev = {
    .minor          = MISC_DYNAMIC_MINOR,

```



```

        .name          = DEVICE_NAME,
        .fops          = &gec210_humidity_dev_fops,
    };

static int __init gec210_humidity_dev_init(void) {
    int ret;
    ret = gpio_request(S5PV210_GPH3(0), "humidity");
    if (ret) {
        printk("%s: request GPIO %d for humidity failed, ret = %d\n",
            DEVICE_NAME,
                S5PV210_GPH3(0), ret);
        return ret;
    }

    s3c_gpio_cfgpin(S5PV210_GPH3(0), 1);
    gpio_set_value(S5PV210_GPH3(0), 1);

    ret = misc_register(&gec210_humidity_dev);

    printk(DEVICE_NAME"\tinitialized\n");
    return ret;
}

static void __exit gec210_humidity_dev_exit(void)
{
    gpio_free(S5PV210_GPH3(0));
    misc_deregister(&gec210_humidity_dev);
}

module_init(gec210_humidity_dev_init);
module_exit(gec210_humidity_dev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gec Lab.");

```

4.3.2 开发板操作

编译生成 humidity.ko,并且在开发板执行命令

```
insmod humidity.ko
```

查看/dev/目录是不是有 humidity 节点，如果有则是加载成功了。然后运行已经完成的 Qt 可执行程序，程序启动后会自动采集温湿度并且显示的。

第五章 RFID 模块

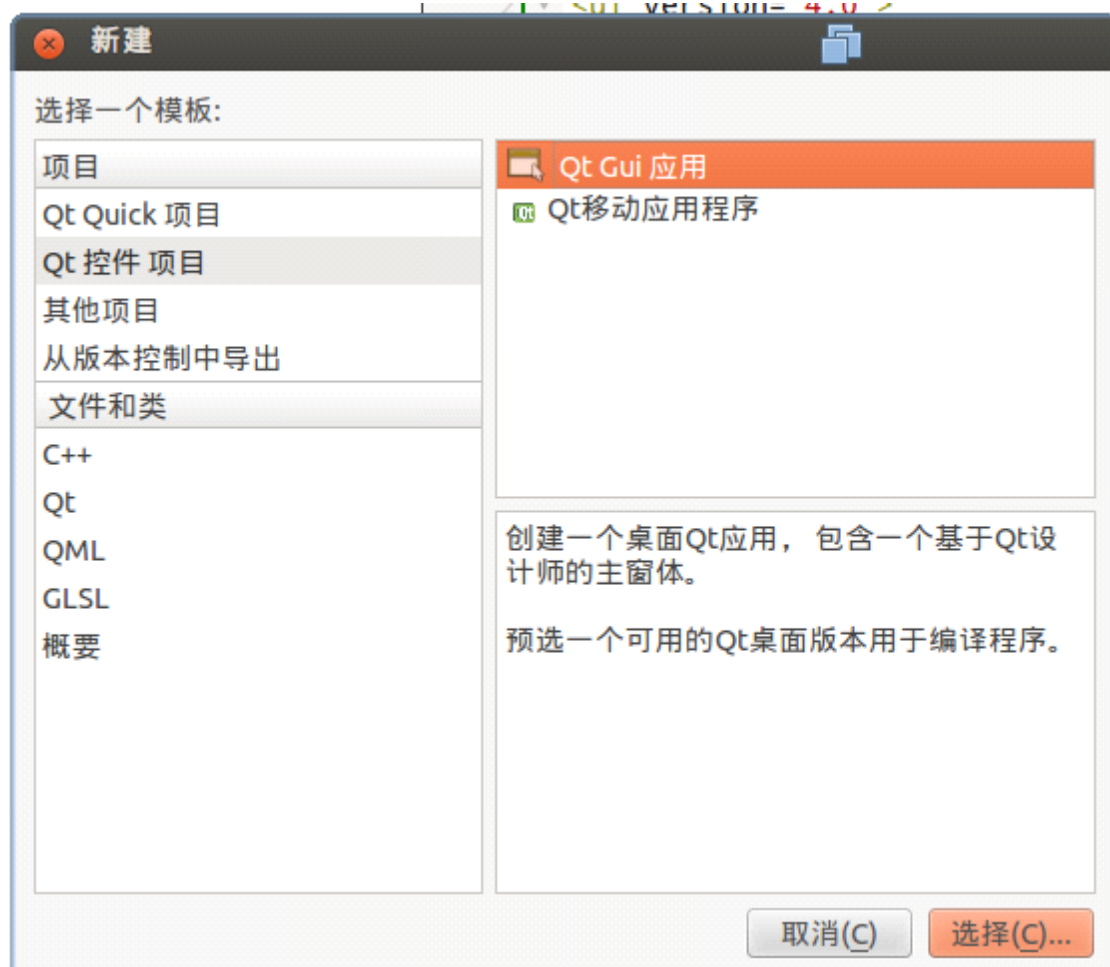
智能家居在操作所有界面之前，必须要进行刷卡，对刷卡进行检验工作的是 RFID 模块，这一章让我们来学习如何设计 RFID 模块的界面。

5.1 图形界面设计

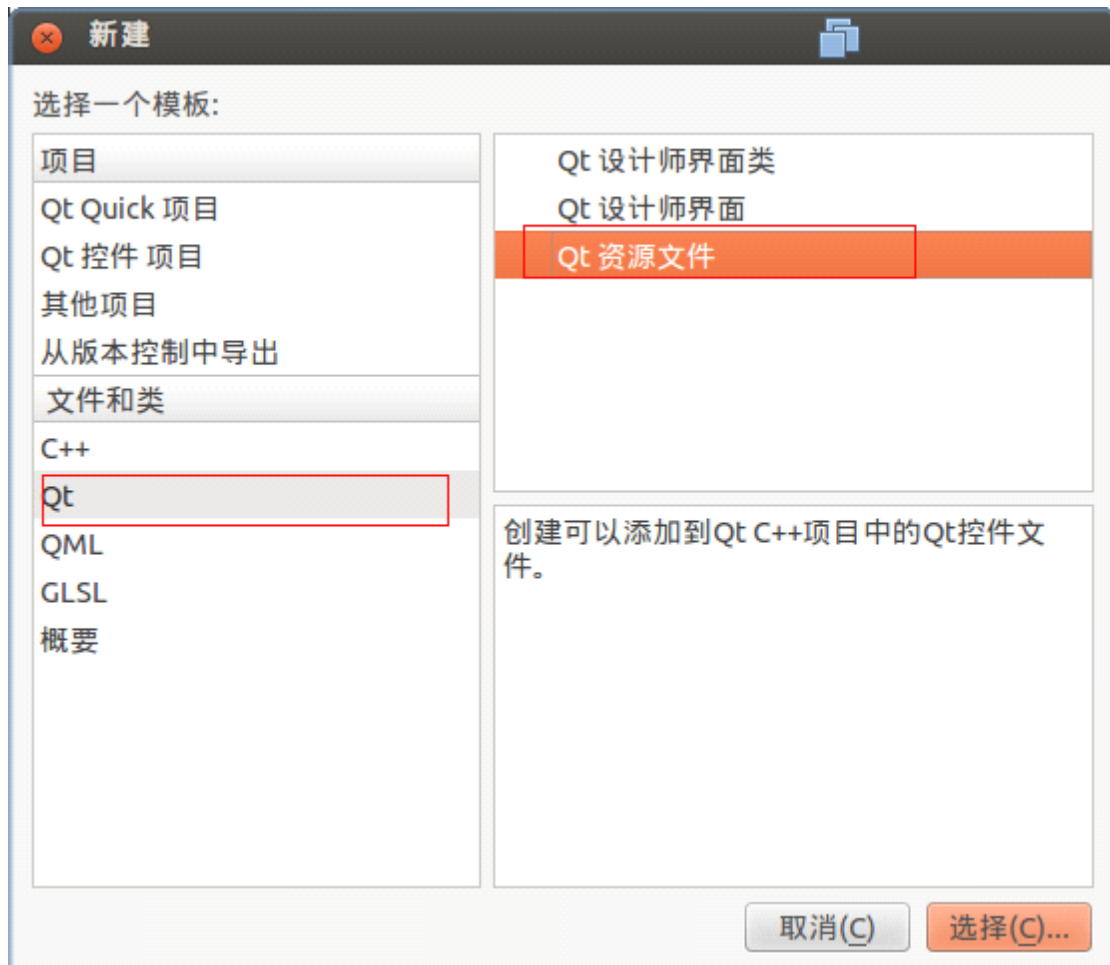
5.1.1 创建工程目录

```
mkdir /home/example/RFID
cd /home/example/RFID
```

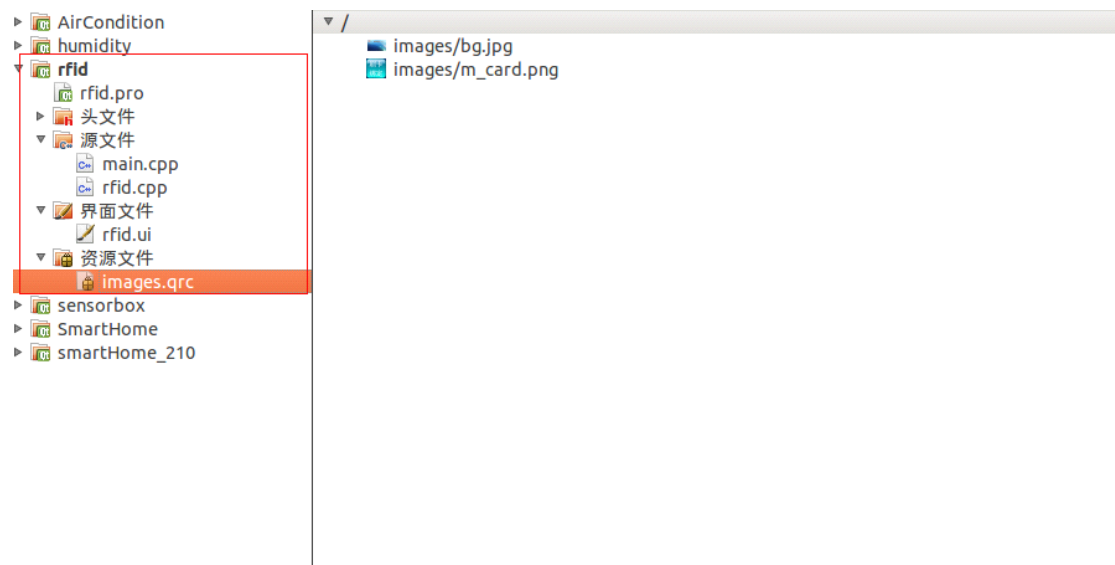
5.1.2 创建工程



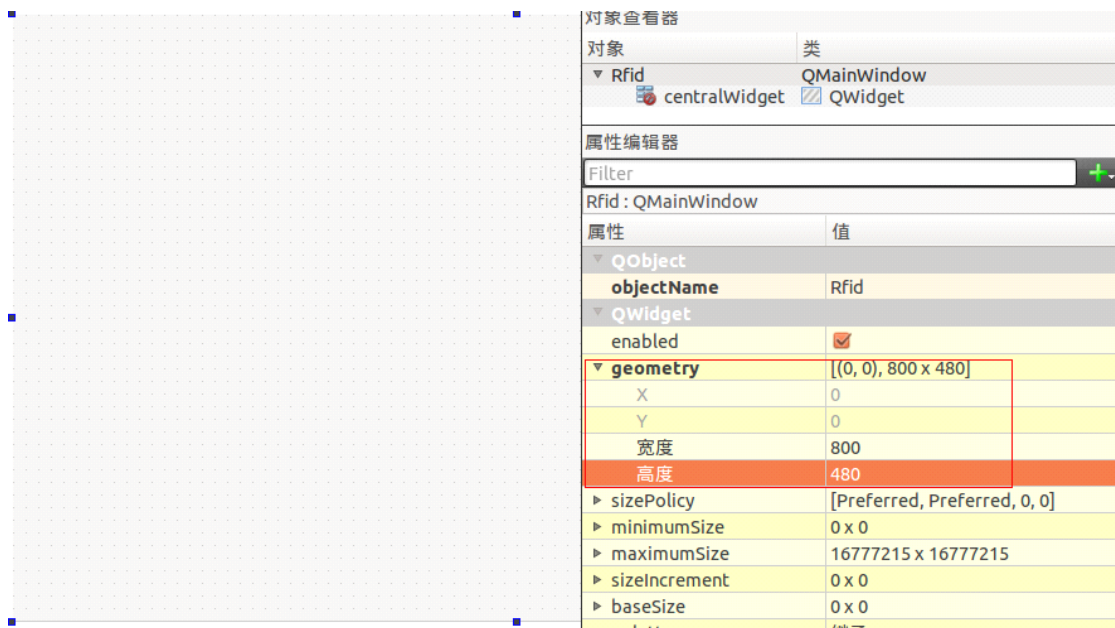
5.1.3 界面设计



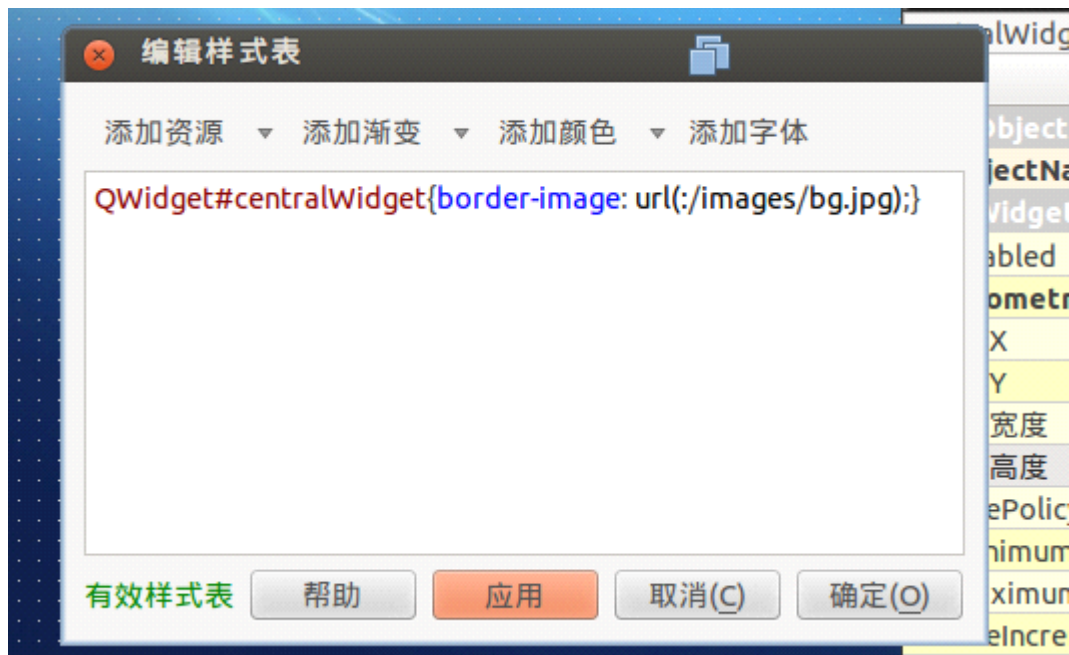
添加前缀和添加文件:



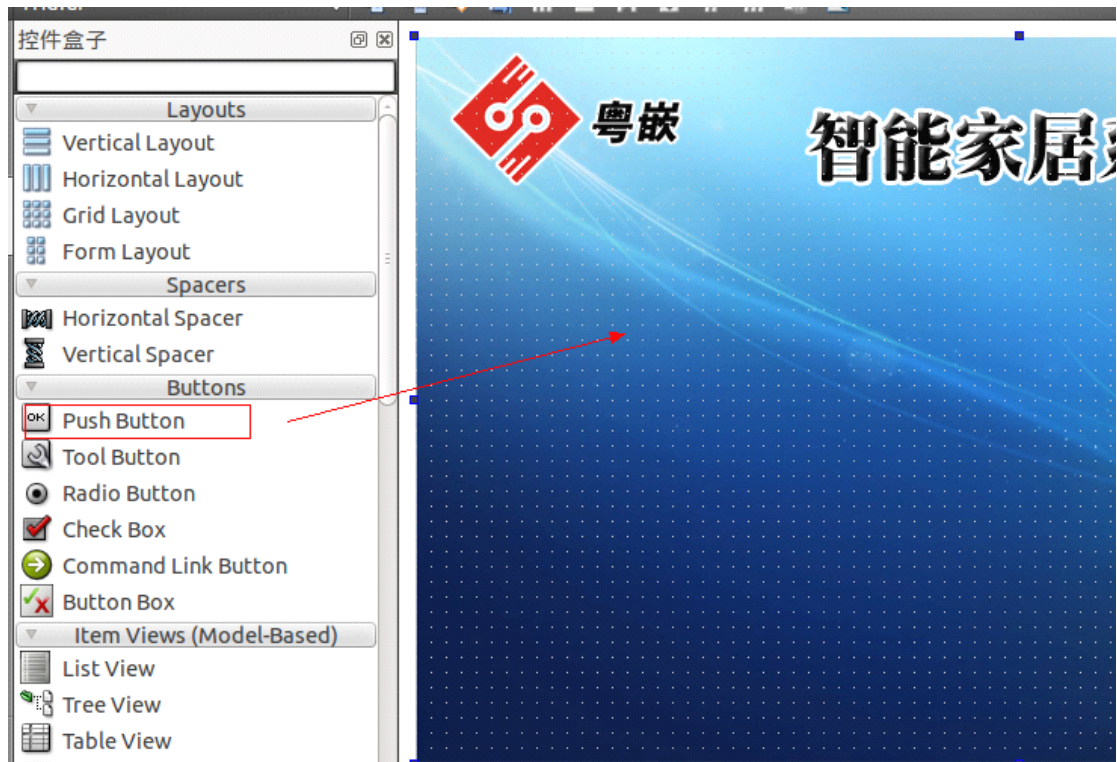
调整主界面大小为 800*480:



修改主界面样式表:



拖拉控件并调整大小:



同时我们要加入一个 lineEdit 来显示 RFID 模块监测到的卡 ID.



5.2 源码实现

修改 rfid.cpp

添加头文件:

```
#include <stdio.h>
```

```

#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/select.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <errno.h>
#include <time.h>

```

下面涉及到几个读卡 ID 的 API

```

/* 设置窗口参数:9600 速率 */
void Rfid::init_ttyS(int fd)
{
    //声明设置串口的结构体
    struct termios termios_new;
    //先清空该结构体
    bzero( &termios_new, sizeof(termios_new));
    // cfmakeraw()设置终端属性，就是设置 termios 结构中的各个参数。
    cfmakeraw(&termios_new);
    //设置波特率
    //termios_new.c_cflag=(B9600);
    cfsetispeed(&termios_new, B9600);
    cfsetospeed(&termios_new, B9600);
    //CLOCAL 和 CREAD 分别用于本地连接和接受使能，因此，首先要通过位
    掩码的方式激活这两个选项。
    termios_new.c_cflag |= CLOCAL | CREAD;
    //通过掩码设置数据位为 8 位
    termios_new.c_cflag &= ~CSIZE;
    termios_new.c_cflag |= CS8;
    //设置无奇偶校验
    termios_new.c_cflag &= ~PARENB;
    //一位停止位
    termios_new.c_cflag &= ~CSTOPB;
    tcflush(fd,TCIFLUSH);
    // 可设置接收字符和等待时间，无特殊要求可以将其设置为 0
    termios_new.c_cc[VTIME] = 10;
    termios_new.c_cc[VMIN] = 1;
    // 用于清空输入/输出缓冲区

```

```

    tcflush (fd, TCIFLUSH);
    //完成配置后，可以使用以下函数激活串口设置
    if(tcsetattr(fd,TCSANOW,&termios_new) )
        printf("Setting the serial1 failed!\n");

}
/*计算校验和*/
unsigned char Rfid::CalBCCS(unsigned char *buf, int n)
{
    int i;
    unsigned char bcc=0;
    for(i = 0; i < n; i++)
    {
        bcc ^= *(buf+i);
    }
    return (~bcc);
}
/*请求天线范围内的卡*/
int Rfid::PiccRequestS(int fd)
{
    unsigned char WBuf[128], RBuf[128];
    int ret;
    fd_set rdfd;

    memset(WBuf, 0, 128);
    memset(RBuf, 1, 128);
    WBuf[0] = 0x07;    //帧长= 7 Byte
    WBuf[1] = 0x02;    //包号= 0 , 命令类型= 0x01
    WBuf[2] = 0x41;    //命令= 'C'
    WBuf[3] = 0x01;    //信息长度= 0
    WBuf[4] = 0x52;    //请求模式: ALL=0x52
    WBuf[5] = CalBCCS(WBuf, WBuf[0]-2);    //校验和
    WBuf[6] = 0x03;    //结束标志

    FD_ZERO(&rdfd);
    FD_SET(fd,&rdfd);

    write(fd, WBuf, 7);
    ret = select(fd + 1,&rdfd, NULL,NULL,&timeout);
    switch(ret)
    {
        case -1:
            perror("select error\n");
            break;
    }
}

```

```

case 0:
    printf("Request timed out.\n");
    break;
default:
    ret = read(fd, RBuf, 8);
    if (ret < 0)
    {
        // printf("ret = %d, %m\n", ret, errno);q
        break;
    }
    if (RBuf[2] == 0x00)          //应答帧状态部分为 0 则请求成功
    {
        return 0;
    }
    break;
}
return -1;
}

```

/*防碰撞，获取范围内最大 ID*/

```

int Rfid::PiccAnticollS(int fd)
{
    unsigned char WBuf[128], RBuf[128];
    int ret;
    fd_set rfd;
    memset(WBuf, 0, 128);
    memset(RBuf, 0, 128);
    WBuf[0] = 0x08;    //帧长= 8 Byte
    WBuf[1] = 0x02;    //包号= 0 , 命令类型= 0x01
    WBuf[2] = 0x42;    //命令= 'B'
    WBuf[3] = 0x02;    //信息长度= 2
    WBuf[4] = 0x93;    //防碰撞 0x93 --一级防碰撞
    WBuf[5] = 0x00;    //位计数 0
    WBuf[6] = CalBCCS(WBuf, WBuf[0]-2);    //校验和
    WBuf[7] = 0x03;    //结束标志

    FD_ZERO(&rfd);
    FD_SET(fd, &rfd);
    write(fd, WBuf, 8);

    ret = select(fd + 1, &rfd, NULL, NULL, &timeout);
    switch(ret)

```



```

{
case -1:
    perror("select error\n");
    break;
case 0:
    perror("Timeout:");
    break;
default:
    ret = read(fd, RBuf, 10);
    if (ret < 0)
    {
        // printf("ret = %d, %m\n", ret, errno);
        break;
    }
    if (RBuf[2] == 0x00) //应答帧状态部分为 0 则获取 ID 成功
    {
        cardidS = (RBuf[4]<<24) | (RBuf[5]<<16) | (RBuf[6]<<8) | RBuf[7];
        return 0;
    }
}
return -1;
}

```

QString Rfid::readCardID()

```

{
    /* int ret, i; */
    int fd;

    fd = open(DEV_RFID, O_RDWR | O_NOCTTY | O_NONBLOCK);
    if (fd < 0)
    {
        fprintf(stderr, "open /dev/ttySAC1 fail!\n");
        return "error";
    }
    /*初始化串口*/
    init_ttyS(fd);
    timeout.tv_sec = 1;
    timeout.tv_usec = 0;

    /*请求天线范围的卡*/
    if ( PiccRequestS(fd) )
    {

```

```

        printf("The request failed!\n");
        ::close(fd);
        return "error";
    }
    /*进行防碰撞，获取天线范围内最大的 ID*/
    if( PiccAnticollS(fd) )
    {
        printf("Couldn't get card-id!\n");
        ::close(fd);
        return "error";
    }
    printf("card ID = %x\n", cardidS);
    ::close(fd);
    return QString::number(cardidS,16);
}

```

因此我们直接在 pushbutton_card 的槽函数直接去获得 ID 号，再显示于 lineedit 上；

```

void Rfid::on_pushButton_card_clicked()
{
    QString cardid;
    sleep(2);
    cardid = readCardID();
    ui->lineEdit->setText(cardid);
}

```

到这里可以编译运行

提示：这个 RFID 模块用到的是串口 1。开发板必须和智能家居板通过杜邦线连接串口 1。

第六章 报警模块

智能家居报警模块有两种分类，1 是当烟雾浓度超过警戒值的火警报警，2 是当有红外监测到有人情况的防盗报警。这个模块涉及到的驱动有三个，烟雾传感器，红外传感器和蜂鸣器，蜂鸣器是一个执行器，也就是需要提醒用户的时候鸣叫。那么下面就开始：

6.1 创建工程

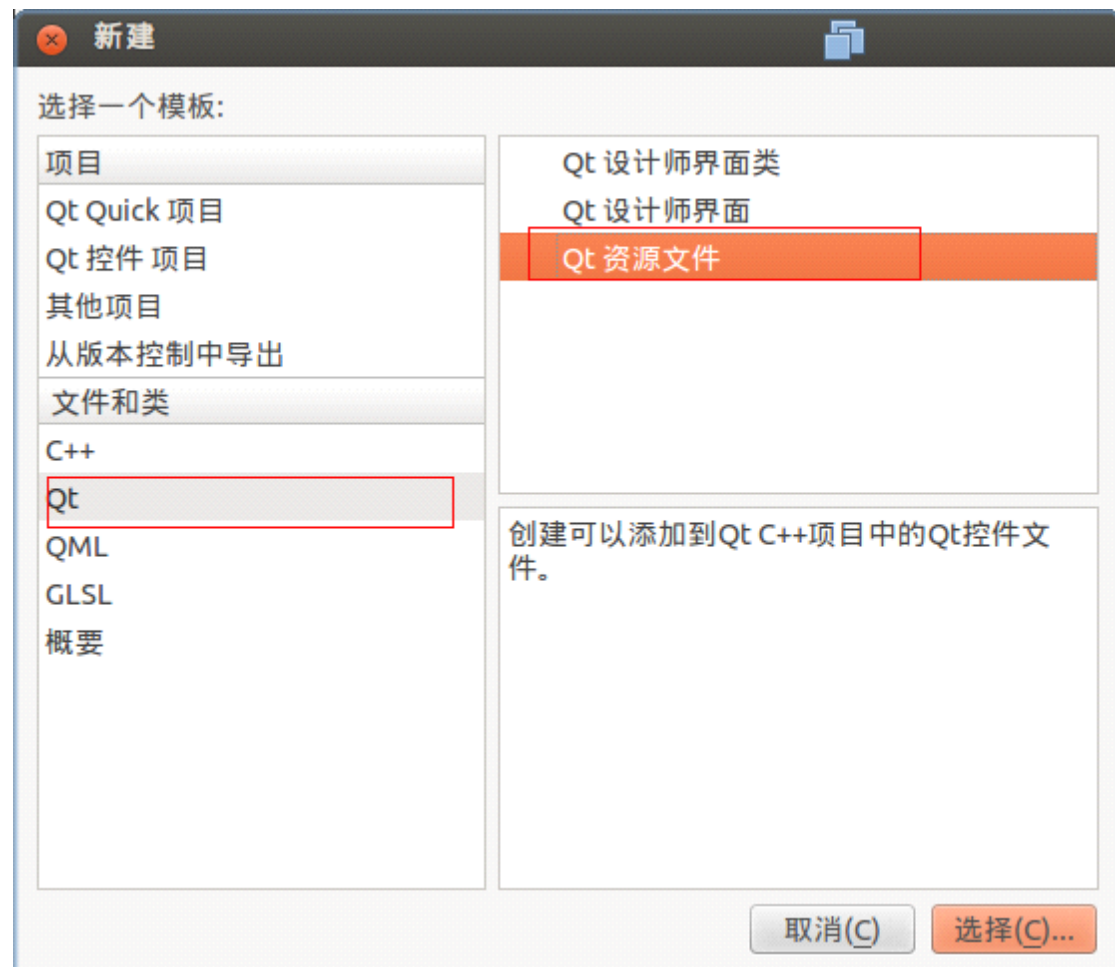
6.1.1 创建工程目录

```
mkdir /home/example/alarm  
cd /home/example/alarm
```

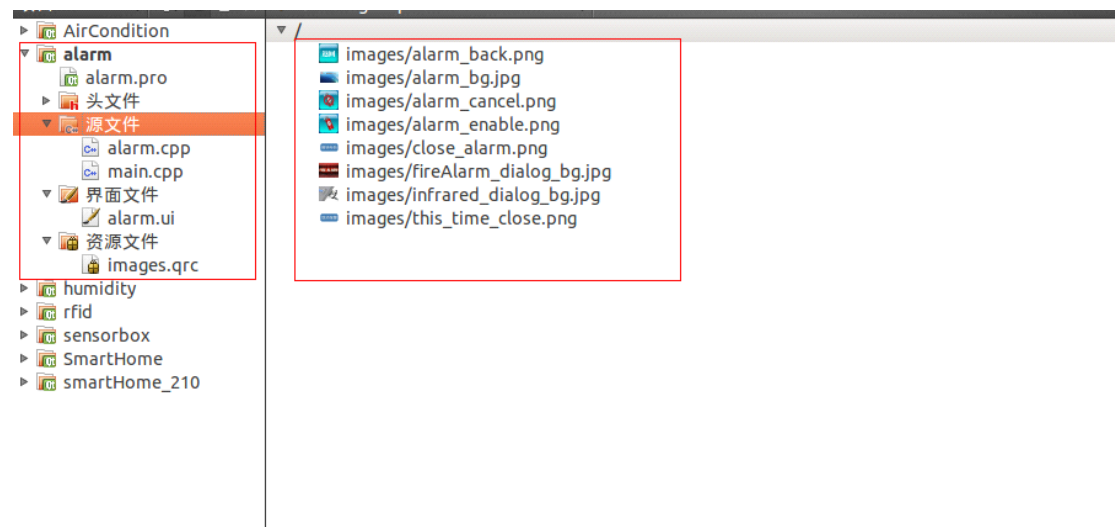
6.1.2 创建工程



6.2 界面设计



添加前缀和添加文件:



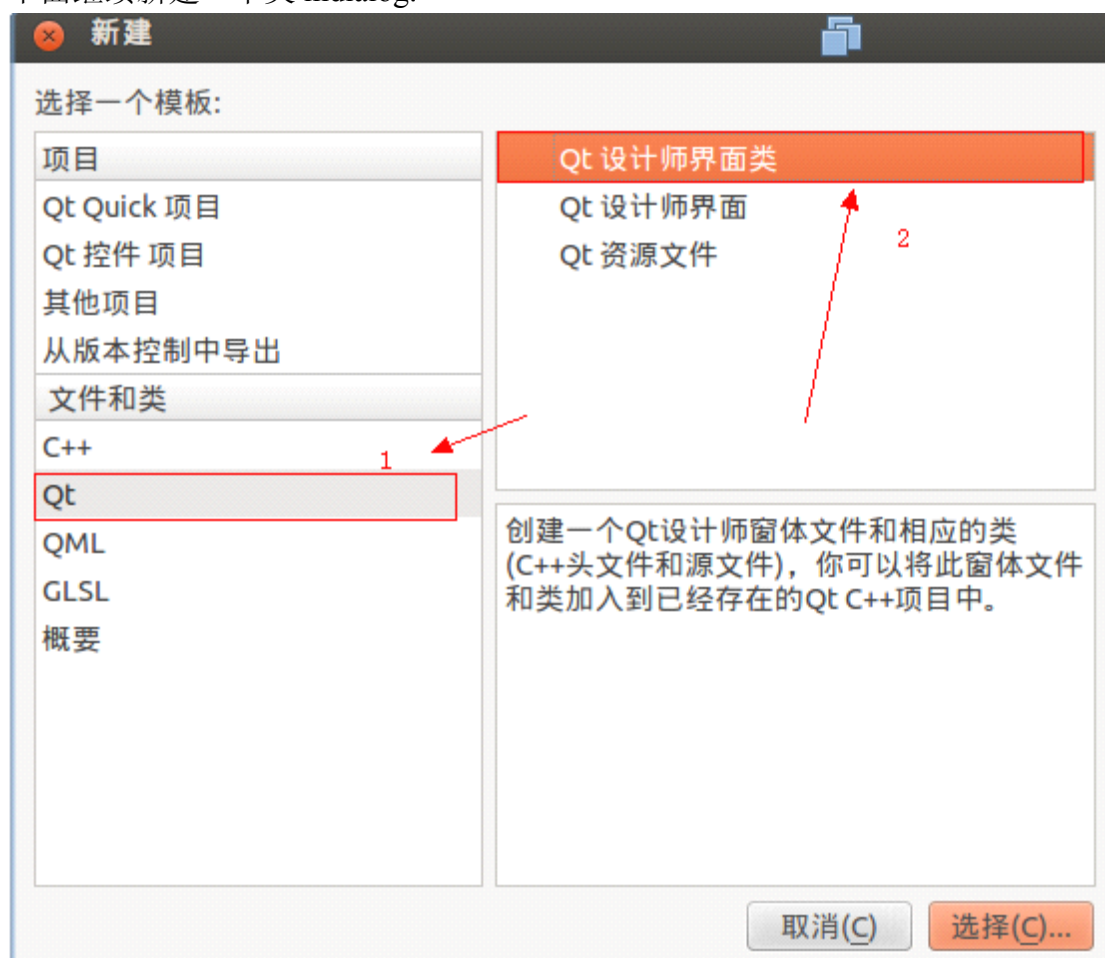
和之前的几个模块来比较，这里不同的地方在于这里需要用到两个界面类，因为在控制报警这里是一个类，另一个是报警进行时弹出的一个窗口。

首先我们先对着控制界面进行设计，这里就不详细讲，不明之处请参考前面界面

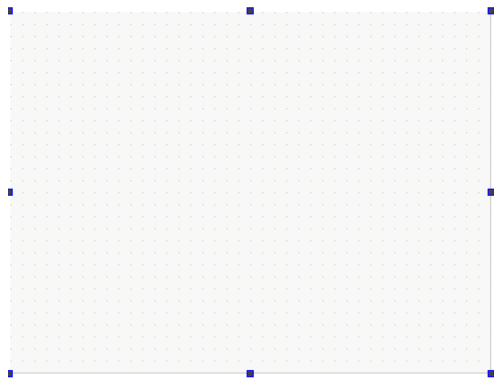
设计：



下面继续新建一个类 mdialog:

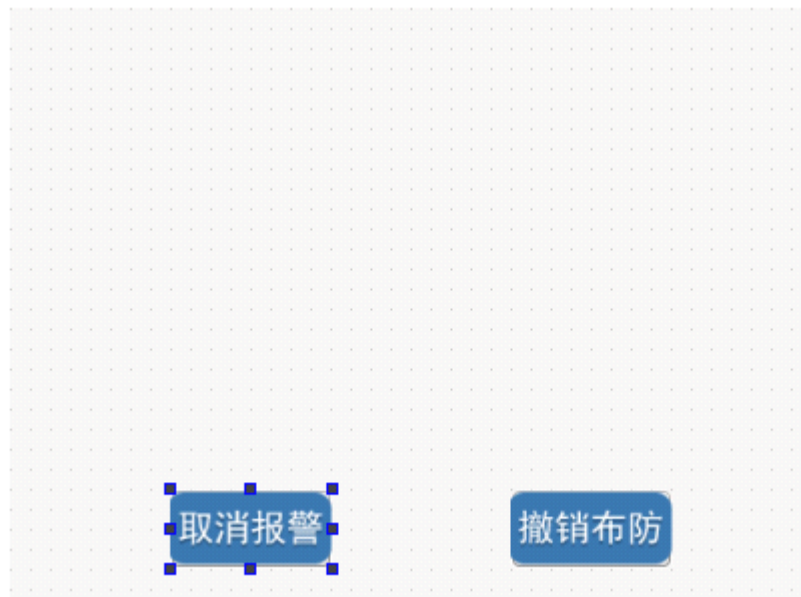


进去 mdialog.ui 文件后进行一系列的设计，可自行设计，这里用到的界面大小是，



对象查看器	
对象	类
mdialog	QWidget
属性编辑器	
Filter	
mdialog: QWidget	
属性	值
▼ QObject	
objectName	mdialog
▼ QWidget	
enabled	<input checked="" type="checkbox"/>
▼ geometry	[(0, 0), 400 x 300]
X	0
Y	0
宽度	400
高度	300
▶ sizePolicy	[Preferred, Preferred, 0, 0]
▶ minimumSize	0 x 0

设计完之后是这样的：



6.3 源码实现

```
#ifndef ALARM_H
#define ALARM_H
```

```
#include <QMainWindow>
#include <QTimer>
#include <QDebug>
#include "mdialog.h"
namespace Ui {
    class Alarm;
}
```

```
class Alarm : public QMainWindow
```

```

{
    Q_OBJECT

public:
    explicit Alarm(QWidget *parent = 0);
    ~Alarm();

private slots:
    void on_pushButton_alarm_clicked(); //开关控制按钮
    void timeroutslot(); //定时器槽函数，轮巡红外和烟雾到
状态
    void setonhere(int); //弹出窗口发过来到信号
public:
    void show_alarm_dialog(QString mode); //判断是烟雾还是红外
    void alarmMonitor(); //实时读取状态到函数
    void InitAllDev(); //打开所有用到的设备
private:
    Ui::Alarm *ui;
    bool onoff;
    bool AlarmFlag; //报警功能开关标志
    QTimer *ONtimer;
    mdialog *show_dialog; //弹出窗口
};

#endif // ALARM_H

```

alarm.cpp 源码

```

#include "alarm.h"
#include "ui_alarm.h"

#include <fcntl.h>
#include <sys/ioctl.h>
int gasfd ;
int hwfd;
int beepfd;
Alarm::Alarm(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Alarm)
{
    ui->setupUi(this);
    setWindowFlags(Qt::FramelessWindowHint);

```

```

    AlarmFlag = false;
    onoff=true;
    ONtimer = new QTimer;
    connect(ONTimer,SIGNAL(timeout()),this,SLOT(timeroutslot()));
    InitAllDev();
    ONtimer->start(1000);
    show_dialog = NULL;
}

Alarm::~Alarm()
{
    ::close(gasfd);
    ::close(hwfd);
    ::close(beepfd);
    delete ui;
}

void Alarm::on_pushButton_alarm_clicked()
{
    if(onoff)
    {
        AlarmFlag = true;
        ui->pushButton_alarm->setIcon(QIcon(":/images/alarm_cancel.png"));
    }
    else
    {
        AlarmFlag = false;
        ui->pushButton_alarm->setIcon(QIcon(":/images/alarm_enable.png"));
    }
    onoff = !onoff;
}

void Alarm::InitAllDev()
{
    if((gasfd = open("/dev/gas", O_RDWR)) < 0)           //烟雾
    {
        qDebug("open gas device fail...");
        return;
    }

    if((hwfd = open("/dev/infrared",O_RDWR)) < 0)       //红外
    {
        qDebug("open infrared err...");
        return;
    }
}

```



```

    }

    if((beepfd = open("/dev/pwm", O_RDWR)) < 0)
//蜂鸣器
    {
        qDebug("open beepfd err...");
        return;
    }
}

void Alarm::timeroutslot()
{
    if(!AlarmFlag)
        alarmMonitor();
}

void Alarm::alarmMonitor()
{
    int hwbye, hwbuf;
    int gasbye, gasbuf;
    gasbye = read(gasfd, &gasbuf, sizeof(gasbuf));
    qDebug() << "gas =" << gasbuf;
    if(gasbuf == 16)
    {
        qDebug("Fire...");
        ioctl(beepfd, 1, 1000);    //控制蜂鸣器
        /*显示是否暂时取消警报对话框*/
        if(show_dialog == NULL)
        {
            QString tmp = "fire";
            show_alarm_dialog(tmp); //显示火警警告窗体
        }
    }
}

hwbye = read(hwfd, &hwbuf, sizeof(hwbuf));
qDebug() << "hwbye =" << hwbuf;
if(hwbuf == 0)    //报警处理(控制蜂鸣器，发送短信)
{
    ioctl(beepfd, 1, 1000);
    /*显示是否暂时取消警报对话框*/
    if(show_dialog == NULL)
    {
        QString tmp = "hw";
        show_alarm_dialog(tmp); //显示红外警告窗体
    }
}

```

```

    }
}
/*****
*****显示和关闭警报窗口*****
*****/

void Alarm::show_alarm_dialog(QString mode)
{
    show_dialog = new mdialog(this);
    connect(show_dialog,SIGNAL(sig_toalarm(int)),this,SLOT(setonhere(int)));
    if (mode == "fire")
    {
        show_dialog->setStyleSheet("#frame{border-image:
url(/images/fireAlarm_dialog_bg.jpg)}");
    }
    else
    {
        show_dialog->setStyleSheet("#frame{border-image:
url(/images/infrared_dialog_bg.jpg)}");
    }
    show_dialog->setGeometry(QRect(200,100,400,300) );
    show_dialog->show();
}

void Alarm::setonhere(int index )
{
    switch(index)
    {
        case 2:
            onoff = false;
            AlarmFlag = true;
            ui->pushButton_alarm->setIcon(QIcon(":/images/alarm_cancel.png"));
            break;
    }
    show_dialog = NULL;
}

```

```

mdialog.cpp 源码
#include "mdialog.h"
#include "ui_mdialog.h"
#include <fcntl.h>
#include <sys/ioctl.h>

extern int beepfd;
mdialog::mdialog(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::mdialog)
{
    ui->setupUi(this);
    setWindowFlags(Qt::FramelessWindowHint);
}

mdialog::~mdialog()
{
    delete ui;
}

void mdialog::on_pushButton_stop_clicked()
{
    ioctl(beepfd,0);
    emit sig_toalarm(1);
    parentWidget()->show();
    delete this;
}

void mdialog::on_pushButton_off_clicked()
{
    ioctl(beepfd,0);
    emit sig_toalarm(2);
    parentWidget()->show();
    delete this;
}

```

6.4 驱动实现

烟雾传感器源码

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/clock.h>
#include <linux/wait.h>
#include <linux/sched.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>
```

```
#include <asm/io.h>
#include <asm/irq.h>
#include <asm/uaccess.h>
```

```
#include <mach/map.h>
#include <mach/regs-clock.h>
#include <mach/regs-adc.h>
#include <mach/regs-gpio.h>
#include <plat/regs-timer.h>
#include <mach/gpio.h>
#include <linux/irq.h>
```

```
#define DEVICE_NAME "gas"
```

```
// XEINT16/KP_COL0/GPH2_0
```

```
static int interrupt = 1;
```

```
static int irq;
```

```
static irqreturn_t irq_handler(int irq, void *dev_id)
```

```
{
    printk("Eint16...\n");
    interrupt = 16;
    return IRQ_HANDLED;
}
```

```
static int gas_open(struct inode *inode, struct file *file)
```

```
{
    return 0;
}
```

```

static ssize_t gas_read(struct file *file, char __user *buf,
                        size_t count, loff_t *offset)
{
    unsigned long err;
    if(copy_to_user((void *)buf, &interrup, min(sizeof(interrup), count)))
    {
        printk("copy_to_user fail.\n");
    }
    // printk("err = %ld\n", err);
    interrump = 1;
    return err ? -EFAULT : min(sizeof(interrup), count);
}

static struct file_operations gec210_gas_dev_fops = {
    .owner          = THIS_MODULE,
    .read           = gas_read,
    .open           = gas_open,
};

static struct miscdevice gec210_gas_dev = {
    .minor          = MISC_DYNAMIC_MINOR,
    .name           = DEVICE_NAME,
    .fops           = &gec210_gas_dev_fops,
};

static int __init gec210_gas_dev_init(void) {
    int ret;
    int err = 0;
    s3c_gpio_cfgpin(S5PV210_GPH2(0), S3C_GPIO_SFN(0XF));
    gpio_set_value(S5PV210_GPH2(0), 1);
    s3c_gpio_setpull(S5PV210_GPH2(0), S3C_GPIO_PULL_NONE);

    irq = gpio_to_irq(S5PV210_GPH2(0));
    err = request_irq(irq, irq_handler, IRQ_TYPE_EDGE_BOTH, DEVICE_NAME,
NULL);
    if (err) {
        printk("request irq %d failed for infrared, %d\n", irq, err);
        return -1;
    }
    ret = misc_register(&gec210_gas_dev);
    printk(DEVICE_NAME"\ninitialized\n");
    return ret;
}

```

```
static void __exit gec210_gas_dev_exit(void) {  
    free_irq(irq, NULL);  
    misc_deregister(&gec210_gas_dev);  
}
```

```
module_init(gec210_gas_dev_init);  
module_exit(gec210_gas_dev_exit);
```

```
MODULE_LICENSE("GPL");  
MODULE_AUTHOR("Gec Lab.");
```

红外传感器:

```
#include <linux/kernel.h>  
#include <linux/module.h>  
#include <linux/slab.h>  
#include <linux/input.h>  
#include <linux/init.h>  
#include <linux/errno.h>  
#include <linux/serio.h>  
#include <linux/delay.h>  
#include <linux/clock.h>  
#include <linux/wait.h>  
#include <linux/sched.h>  
#include <linux/cdev.h>  
#include <linux/miscdevice.h>
```

```
#include <asm/io.h>  
#include <asm/irq.h>  
#include <asm/uaccess.h>
```

```
#include <mach/map.h>  
#include <mach/regs-clock.h>  
#include <mach/regs-adc.h>  
#include <mach/regs-gpio.h>  
#include <plat/regs-timer.h>  
#include <mach/gpio.h>  
#include <linux/irq.h>
```

```
#define DEVICE_NAME "infrared"
```

```

// XEINT0/GPH0_0
volatile int interrup = 1;
static int irq;
static irqreturn_t irq_handler(int irq, void *dev_id)
{
    printk("Eint0...\n");
    interrup = 0;
    printk("iqr = %d\n",irq);
    return IRQ_HANDLED;
}

static int infrared_open(struct inode *inode, struct file *file)
{
    return 0;
}

static ssize_t infrared_read(struct file *file, char __user *buf,
                             size_t count, loff_t *offset)
{
    unsigned long err;
    err = copy_to_user((void *)buf, &interrup, sizeof(interrup));
    // printk("err = %ld\n",err);
    if(err)
    {
        printk("copy from user fail \n");
        return -EFAULT;
    }
    interrup = 1;
    return err ? -EFAULT : min(sizeof(interrup), count);
}

static struct file_operations gec210_infrared_dev_fops = {
    .owner          = THIS_MODULE,
    .read           = infrared_read,
    .open           = infrared_open,
};

static struct miscdevice gec210_infrared_dev = {
    .minor          = MISC_DYNAMIC_MINOR,
    .name           = DEVICE_NAME,
    .fops           = &gec210_infrared_dev_fops,
};

static int __init gec210_infrared_dev_init(void) {

```

```

int ret;
int err = 0;
s3c_gpio_cfgpin(S5PV210_GPH0(0), S3C_GPIO_SFN(0XF));
gpio_set_value(S5PV210_GPH0(0), 0);
s3c_gpio_setpull(S5PV210_GPH0(0), S3C_GPIO_PULL_NONE);

set_irq_type(IRQ_EINT0, IRQ_TYPE_EDGE_FALLING);
err = request_irq(IRQ_EINT0, irq_handler, IRQ_TYPE_EDGE_FALLING,
DEVICE_NAME, NULL); //EINT0 注册
if(err)
{
    printk("request irq %d failed for infrared, %d\n", irq, err);
    return -EBUSY;
}
// irq = gpio_to_irq(S5PV210_GPH0(0));
/* err = request_irq(irq, irq_handler, IRQ_TYPE_EDGE_BOTH, DEVICE_NAME,
NULL);
if (err) {
    printk("request irq %d failed for infrared, %d\n", irq, err);
    return -1;
} */
ret = misc_register(&gec210_infrared_dev);
printk(DEVICE_NAME"\tinitialized\n");
return ret;
}

static void __exit gec210_infrared_dev_exit(void) {
    free_irq(IRQ_EINT0, NULL);
    misc_deregister(&gec210_infrared_dev);
}

module_init(gec210_infrared_dev_init);
module_exit(gec210_infrared_dev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gec Lab.");

```

至于蜂鸣器驱动已经编译进去内核，不用我们加载了。按照前面编译驱动的方法，编译出来的.ko 文件，在开发板上执行命令 `insmod 驱动模块.ko` 则可以自动生成设备节点，就可以运行我们的 Qt 程序了。

第七章 GPRS 模块

智能家居有一个 GPRS 模块，我们这一章开始学习如何设计 GPRS 的界面，那么现在我们就开始这段旅程吧。

7.1.1 创建工程目录

```
mkdir /home/example/gprs  
cd /home/example/gprs
```

7.1.2 创建工程



至于界面设计方面，这里就做讲解，不明之处请参考前面部分文档，可以参考下面设计做：



这个模块我们设计的方案，一开始进来之后要设置发送报警信息对象，也就是在软键盘上输入号码之后并且确定。在此之后，我们先测试一下 GPRS 模块，也就是点击一下 GPRS 那个控件，这操作会像你设置的号码手机发送一条短信息，内容是：This is a test message!

7.1.3 源码实现

软件盘设计可以用到一个 `QButtonGroup` 来管理控件，这样可以多个控件信号链接到一个槽函数，避免代码过于冗余；

```
QButtonGroup *numbergroup = new QButtonGroup;
numbergroup->addButton(ui->pushButton_1,1);
numbergroup->addButton(ui->pushButton_2,2);
numbergroup->addButton(ui->pushButton_3,3);

numbergroup->addButton(ui->pushButton_4,4);
numbergroup->addButton(ui->pushButton_5,5);
numbergroup->addButton(ui->pushButton_6,6);
numbergroup->addButton(ui->pushButton_7,7);
numbergroup->addButton(ui->pushButton_8,8);
numbergroup->addButton(ui->pushButton_9,9);
numbergroup->addButton(ui->pushButton_0,0);
numbergroup->addButton(ui->pushButton_c,10);
```

```
connect(numbergroup,SIGNAL(buttonClicked(int)),this,SLOT(AllButtonSlot(int)));
```

```
void gprs::AllButtonSlot(int id) //槽函数
{
```

```

str = ui->lineEdit->text();
switch(id)
{
case 0:
    str += "0";
    break;
case 1:
    str += "1";
    break;
case 2:
    str += "2";
    break;
case 3:
    str += "3";
    break;
case 4:
    str += "4";
    break;
case 5:
    str += "5";
    break;
case 6:
    str += "6";
    break;
case 7:
    str += "7";
    break;
case 8:
    str += "8";
    break;
case 9:
    str += "9";
    break;
case 10:
    str = str.left(str.length()-1);
    break;
default:
    break;
}
ui->lineEdit->setText(str);
}

```

当然这里最重要的部分代码是对 GPRS 模块的操作，智能家居实验板里用到的 GPRS 模块是通过串口 2 通信，因此注意要用杜邦线连接 J10 和开发板的串口 2。

因此首先要初始化串口：

```
void init_ttys2(int fd)
{
    struct termios termios_new;

    bzero( &termios_new, sizeof(termios_new));
    cfmakeraw(&termios_new);
    termios_new.c_cflag=(B9600);
    termios_new.c_cflag |= CLOCAL | CREAD;

    termios_new.c_cflag &= ~CSIZE;
    termios_new.c_cflag |= CS8;

    termios_new.c_cflag &= ~PARENB;

    termios_new.c_cflag &= ~CSTOPB;
    tcflush(fd,TCIFLUSH);
    termios_new.c_cc[VTIME] = 0;
    termios_new.c_cc[VMIN] = 0;
    tcflush (fd, TCIFLUSH);
    tcsetattr(fd,TCSANOW,&termios_new);
}
```

因为我们这里用到的仅仅是发送短信息的接口，所以我们封装了一个发短信的接口：

```
int sendmessage(unsigned char *num,unsigned char *messge)
{
    int fd;
    unsigned char WBuf[128],RBuf[128];
    unsigned char message[140];
    char *mes_mode="AT+CMGF=1\xd0";
    char *mes_sdms="AT+CMGS=\"";
    char *mes_cend="\\"xd0";
    char *mes_end="\x1a\xd0a";

    strcpy((char *)message, (char *)messge);

    fd = open( DEV_PATH, O_RDWR|O_NOCTTY|O_NDELAY);
    if (-1 == fd){
        //      perror("Can't Open Serial Port");
    }
}
```

```

init_ttys2(fd);

memset(WBuf, 0, 128);
memset(RBuf,0,128);

write(fd,mes_mode,strlen(mes_mode));
usleep(200000);
write(fd,mes_sdms,strlen(mes_sdms));
write(fd,num,strlen( (char *)num));
write(fd,mes_cend,strlen(mes_cend));
usleep(200000);
write(fd,messge,strlen( (char *)messge));
write(fd,mes_end,strlen(mes_end));
printf("phone test end");
close(fd);
return 1;

}

```

此处涉及到的 AT 指令，也就是红色字体。需要参考一下 AT 指令资料。当设置好信息发送对象的号码之后，我们就要点击发送信息的控件 `pushButton_test`:

```

void gprs::on_pushButton_test_clicked()
{
    QString number = ui->label->text();
    QByteArray ba = number.toLatin1();

    char *numberstr=ba.data();
    sendmessage((unsigned char *)numberstr,(unsigned char *)"This is a test
message!");
    qDebug()<<"send ok";
}

```

那么到现在我们就实现了发送短信的一个小模块。