基于ZigBee的智能家居综合样例

1. 实验环境

- 硬件：UP-CUP IOT-6410-II 型嵌入式物联网综合实验系统， PC 机。
- 软件：Vmware Workstation +Fedora Core 8 + MiniCom/Xshell + ARM-LINUX 交叉编译开发环境

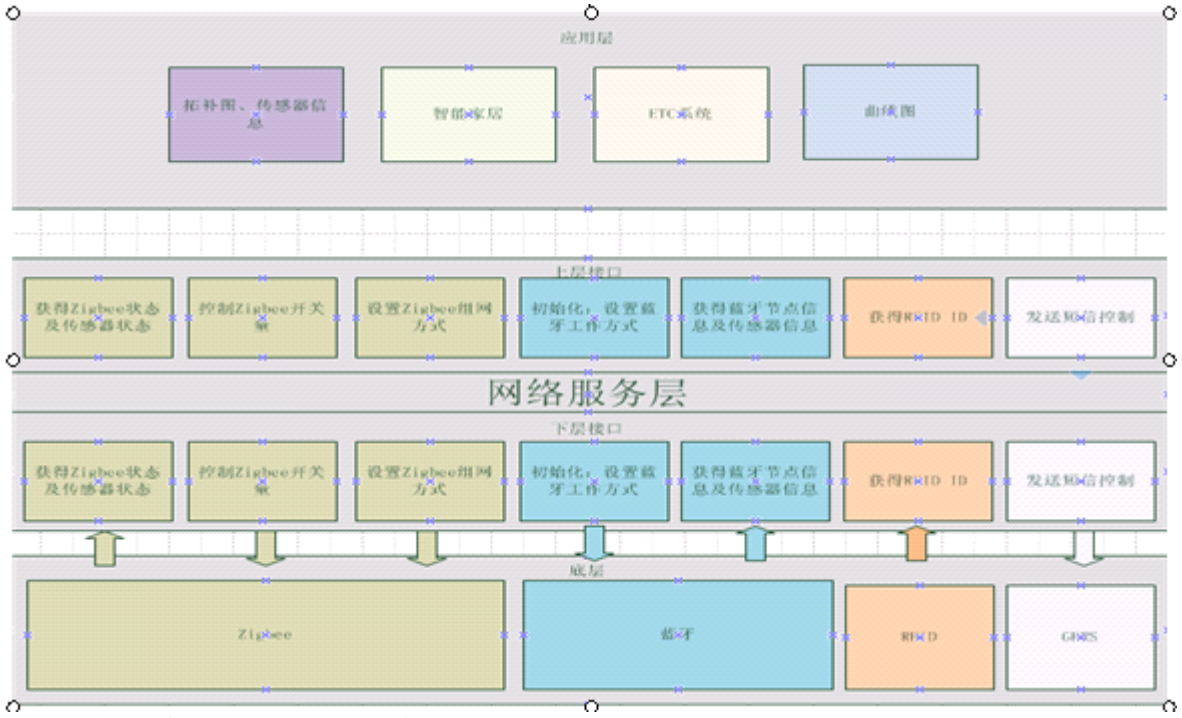
2. 实验内容

- 基于嵌入式网关系统，进行基于 Zigbee 无线传感器网络的智能家居的简单图形界面显示设计
- 基于嵌入式网关系统，了解掌握 Linux 系统下 socket 编程

3. 实验原理

针对 GUI 的综合实例，具体实现都是通络网络层对底层 Zigbee（传感器、控制设备）、蓝牙、RFID 等功能进行封装，提供给 GUI 上层界面统一的调用接口。

功能框图如下所示：



- 网络服务层实现（Socket 通信）

套接口（Socket）为目前 Linux 上最为广泛使用的一种的进程间通信机制，与其他的 Linux 通信机制不同之处在于除了它可用于单机内的进程间通信以外，还可用于不同机器之间的进程间通信。但是由于 Socket 本身不支持同时等待和超时处理，所以它不能直接用来多进程之间的相互实时通信。本实验采用事件驱动库 libev 的方式构建我们的服务器模型。

Socket 服务器端

Libev 是一种高性能事件循环 / 事件驱动库。作为 libevent 的替代作品，其第一个版本发布与 2007 年 11 月。Libev 的设计者声称 libev 拥有更快的速度，更小的体积，更多功能等优势，这些优势在很多测评中得到了证明。正因为其良好的性能，很多系统开始使用 libev 库。libev 同样需要循环探测事件是否产生，其循环体用 ev_loop 结构来表达，并用 ev_loop( ) 来启动。

void ev_loop(ev_loop* loop, int flags)

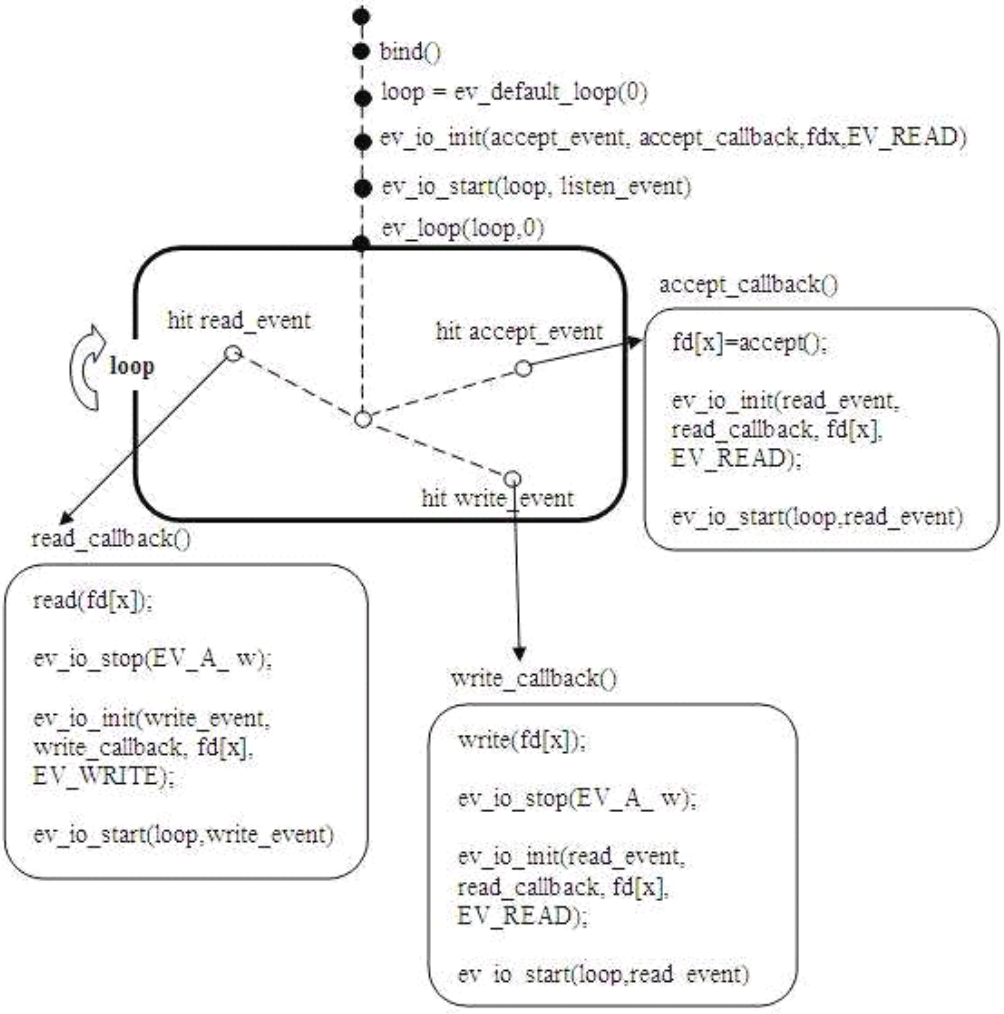
Libev 支持八种事件类型，其中包括 IO 事件。一个 IO 事件用 ev_io 来表征，并用 ev_io_init() 函数来初始化：void ev_io_init(ev_io *io, callback, int fd, int events) 初始化内容包括回调函数 callback，被探测的句柄 fd 和需要探测的事件，EV_READ 表“可读事件”，EV_WRITE 表“可写事件”。

用户需要做的仅仅是在合适的时候，将某些 ev_io 从 ev_loop 加入或剔除。一旦加入，下个循环即会检查 ev_io 所指定的事件有否发生；如果该事件被探测到，则 ev_loop 会自动执行 ev_io 的回调函数 callback()；如果 ev_io 被注销，则不再检测对应事件。无论某 ev_loop 启动与否，都可以对其添加或删除一个或多个 ev_io，添加删除的接口是 ev_io_start() 和 ev_io_stop()。

void ev_io_start(ev_loop *loop, ev_io* io)

void ev_io_stop(EV_A_*)

由此，可以容易得出如下的“一问一答”的服务器模型。由于没有考虑服务器端主动终止连接机制，所以各个连接可以维持任意时间，客户端可以自由选择退出时机。



上述模型可以接受任意多个连接，且为各个连接提供完全独立的问答服务。借助 libev 提供的事件循环 / 事件驱动接口，上述模型有机会具备其他模型不能提供的高效率、低资源占用、稳定性好和编写简单等特点。

服务器主要实现流程是：首先开启一个 Zigbee 后台线程（底层）监听服务器调用信息，接着利用 ev_io_start(loop,&ev_io_watcher);启动一个接收线程，专门用来接收客户端发送过来的命令信息，然后按照相应的协议进行解析，跳转到相应的接口，进一步调用底层 Zigbee 等信息并返回正确的信息给客户端。

封装接口：（libev_test.c）

```
bool Server_GetZigBeeNwkInfo(int fd);
bool Server_GetZigBeeNwkTopo(int fd);
bool Server_GetTempHum(int fd);
bool Server_SetSensorStatus(int fd,unsigned int addr,unsigned int state);
int Server_GetRfidId(int fd);
int Server_GetGPRSSignal(int fd);
bool Server_SendGprsMessage(int fd,unsigned int *phone,unsigned int sensor);
```

Main 函数：

```
int main(int argc ,char** argv)
{
    int listen;
```

```
    ev_io ev_io_watcher;
    signal(SIGPIPE,SIG_IGN);
    printf("start com monitor\n");
    listen=socket_init(argv);
    printf("flag_bind%d",flag_bind);
    if(flag_bind)
    {
        printf("---------flag_bind=1----------\n");
        ComPthreadMonitorStart();//开启 Zigbee 后台线程（底层）
    }
    //struct ev_loop *loop = ev_loop_new(EVBACKEND_EPOLL);
    struct ev_loop *loop = ev_default_loop(EVBACKEND_EPOLL);
    ev_io_init(&ev_io_watcher, accept_callback,listen, EV_READ);
    ev_io_start(loop,&ev_io_watcher); //开启一个接收线程，接收客户端信息
    ev_loop(loop,0);
    ev_loop_destroy(loop);
    if(flag_bind)
    {
        ComPthreadMonitorExit();
    }
    printf("exit com monitor\n");
    return 0;
}
```

命令协议定义：

接收协议：帧头 功能号 数据 结束符（接收帧头为 0x15）

1.获得拓补信息：

帧头（unsigned int）	功能号（unsigned int）	结束符（unsigned int）
0x15	0x01	0x0A

2.获得网络信息

帧头（unsigned int）	功能号（unsigned int）	结束符（unsigned int）
0x15	0x02	0x0A

3.设置传感器状态

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）		结束符（unsigned int）
0x15	0x03	addr	state	0x0A

4.获得 RFID 值

帧头（unsigned int）	功能号（unsigned int）	结束符（unsigned int）
0x15	0x04	0x0A

5.获得蓝牙温湿度值

帧头（unsigned int）	功能号（unsigned int）	结束符（unsigned int）
0x15	0x05	0x0A

6.发送短信

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）		结束符（unsigned int）
0x15	0x06	Phone_num（11 位）	sendor	0x0A

7.获得 GPRS 的初始化信息

帧头（unsigned int）	功能号（unsigned int）	结束符（unsigned int）
0x15	0x07	0x0A

8.清除中断传感器标志位

帧头（unsigned int）	功能号（unsigned int）	结束符（unsigned int）
0x15	0x08	0x0A

发送协议：帧头 功能号 数据 结束符（接收帧头为 0x26）

1.发送拓补信息：
包括三种：协调器掉电、协调器在线无子节点、协调器在线有节点
1）协调器掉电：

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）	结束符（unsigned int）
0x26	0x01	0x01	0x0A

2）协调器在线无子节点：

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）	结束符（unsigned int）
0x26	0x01	0x02	0x0A

3）协调器在线有节点：

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）	结束符（unsigned int）
0x26	0x01	节点链表数据（长度由个数确定）	0x0A

2.发送网络信息

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）					结束符（unsigned int）
0x26	0x02	panid	chnnal	maxchild	maxdepth	maxrouter	0x0A

3.发送设置传感器状态
无
4.发送 RFID 值

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）	结束符（unsigned int）
0x26	0x04	32 位 ID 数据	0x0A

5.发送蓝牙温湿度值

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）	结束符（unsigned int）
0x26	0x05	32 位温湿度数据	0x0A

6.发送短信成功标志

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）	结束符（unsigned int）
0x26	0x06	发送短信成功与否标志	0x0A

7.获得 GPRS 的初始化信息

帧头（unsigned int）	功能号（unsigned int）	数据（unsigned int）	结束符（unsigned int）

0x26	0x07	GPRS 的初始化标志	0x0A

8.清除中断传感器标志位
无

解析接收帧是由switch语句实现的，如下所示：（libev_test.c）

```
void recv_callback(struct ev_loop *loop, ev_io *w, int revents)
{
	printf("\nrecv_callback:\n");

	unsigned int buffer[15]={0};
	unsigned int temp;
	int i=0;
	tv.tv_sec=60;
	tv.tv_usec=0;
	FD_ZERO(&sds);
	FD_SET(w->fd, &sds);
	int ret = select((1+w->fd), &sds, NULL, NULL, &tv);
	if(ret >0)
	{
		//printf("server select wait...\n");
		if (FD_ISSET(w->fd, &sds))
		{
			//printf("server FD_ISSET...\n");
			int len=0;
			len=recv(w->fd,&buffer[i],sizeof(unsigned int),0);
			if(len>0)
			{
				//printf("server buffer[i]=%d\n",buffer[i]);
			while(buffer[i]!=0x0A)
			{
			    i++;
			    len=recv(w->fd,&buffer[i],sizeof(unsigned int),0);
			    //printf("server buffer[i]=%d\n",buffer[i]);
			}
			if(buffer[0]==0x15)
			{
				temp=buffer[1];
				switch(temp)
				{
				case 0x01:

						printf("COMMAND:-------TOPOINFO--------:\n");//获得拓补信息
						Server_GetZigBeeNwkTopo(w->fd);
						break;
				case 0x02:

						printf("COMMAND:-------GetZigBeeNwkInfo--------:\n");//获得网
络信息
						Server_GetZigBeeNwkInfo(w->fd);
						break;
				case 0x03:

						printf("CMOMAND:-------SET_SENSOR_STATUS:--------:\n");//
设置传感器状态
						Server_SetSensorStatus(w->fd,buffer[2],buffer[3]);
```

```
                        break;
                case 0x04:

                        printf("COMMAND:-------GetRfidId--------:\n");//获得 RFID ID
                        Server_GetRfidId(w->fd);
                        break;
                case 0x05:

                        printf("COMMAND:-------GetTempHum--------:\n");//获得蓝牙温
湿度
                        Server_GetTempHum(w->fd);
                        break;
                case 0x06:

                        printf("COMMAND:-------SendGprsMessage--------:\n");//发送
短信
                        Server_SendGprsMessage(w->fd,&buffer[2],buffer[13]);
                        break;
                case 0x07:

                        printf("COMMAND:-------get GPRSSignal--------:\n");//获得
GPRS 初始化信息
                        Server_GetGPRSSignal(w->fd);
                        break;
                case 0x08:

                        printf("COMMAND:-------clear intrupt--------:\n");//清楚中断传感
器的标志位
                        gIntLock=0;
                        break;
                default:
                        printf("error COMMAND\n");//其他
                        break;

                }
        }
        else
        {
                printf("other protrol.\n");
        }
    }
        else if(len ==0)
        {
                printf("remote socket closed!socket fd: %d\n",w->fd);//远程连接关
闭
                close(w->fd);
                ev_io_stop(loop, w);
                free(w);
                return;
        }
        else
        {
                if(errno == EAGAIN ||errno == EWOULDBLOCK)
                {
                        printf("errno == EAGAIN ||errno == EWOULDBLOCK\n");
                            //goto loop;
```

```
                        }
                        else
                        {
                                printf("ret :%d ,close socket fd : %d\n",ret,w->fd);
                                close(w->fd);
                                ev_io_stop(loop, w);
                                free(w);
                                return;
                        }
                }
        }
        else{
                                printf("not    server fd.\n");
                }
    }
    else if(ret == 0){
                printf("server read wait timeout!!!\n");//超时

    }
    else{// ret <0
                printf("server select error.\n");
                //perror(ret);
    }
}
```

实验源程序为 UP-CUP IOT-6410-II 型嵌入式物联网综合实验系统产品光盘\演示程序烧写目录\src\6410\GUI\目录下的 server.tar.bz2,服务器源程序的编译及 libev 库的编译与移植请参考相同目录下的 DOC 文件,本实验我们可以直接使用编译好的源程序 ev_server(开发板出厂烧写过演示程序时，已经存在于开发板上了，我们可以直接使用)。

Socket 客户器端

客户端程序主要是用于同服务器端进行交互,实现为上层 GUI 图形界面提供封装好的接口，上层 GUI 可以直接调用，实现的接口如下：

```
    发送帧接口：
    void GetConnect(char *ipaddr,int port); //连接服务器的 port 端口
    void Api_Cliect_GetZigBeeNwkInfo();//获得 Zigbee 网络的基本信息
    void Api_Cliect_GetZigBeeNwkTopo();//获得 Zigbee 网络拓补链表
    void Api_Cliect_GetTempHum();//获得蓝牙模块上的温湿度值
    void Api_Cliect_GetRfidId();//获得 RFID 的卡号
    void Api_Cliect_GetGPRSSignal();//获得 GPRS 初始化信息
    void Api_Cliect_SendGprsMessage(char *phone,int sensor); //进行 GPRS 短信报警
    void Api_Cliect_ClearIntlock();//清楚中断型传感器状态锁
    void Api_Cliect_SetSensorStatus(unsigned int nwkaddr,unsigned int Mode); //针对设置型传感器，控制其开关量

    接收后处理接口：
    void Cliect_ZigBeeNwkTopo_Process(unsigned int *node,unsigned int count);//拓扑链表处理
    void Cliect_ZigBeeNwkInfo_Process(unsigned int *nwkinfo);//设备信息处理
    void Cliect_RfidId_Process(unsigned int *id);//RFID  数据处理
    void Cliect_TempHum_Process(unsigned int *temp);//蓝牙温湿度处理
```

```
void Cliect_GPRSSignal_Process(unsigned int sig);//GPRS 初始化信息
void Cliect_GPRSSend_Process(unsigned int sig);//短信报警是否成功处理
```

本实验主要用到了：

本实验用到的发送帧接口：
void GetConnect(char *ipaddr,int port); //连接服务器的 port 端口
void Api_Cliect_GetZigBeeNwkTopo();//获得 Zigbee 网络拓补链表
void Api_Cliect_GetGPRSSignal();//获得 GPRS 初始化信息
void Api_Cliect_SendGprsMessage(char *phone,int sensor); //进行 GPRS 短信报警
void Api_Cliect_ClearIntlock();//清除中断型传感器状态锁
void Api_Cliect_SetSensorStatus(unsigned int nwkaddr,unsigned int Mode); //针对设置型传感器，控制其开关量
接收后处理接口：
void Cliect_ZigBeeNwkTopo_Process(unsigned int *node,unsigned int count);//拓扑链表处理
void Cliect_GPRSSignal_Process(unsigned int sig);//GPRS 初始化信息
void Cliect_GPRSSend_Process(unsigned int sig);//短信报警是否成功处理

主要是针对上层 GUI 的设计，网络服务客户端提供的接口可供上层应用直接使用，所以对于 socket 通信只是了解即可，有兴趣的可阅读源程序，其具体实现源程序位于 UP-CUP IOT-6410-II 型嵌入式物联网综合实验系统产品光盘\物联网无线传感网络部分\exp\linux\QT_exp\smarthome 目录下的 server 目录，此目录针对 4 个 GUI 的综合样例及演示程序都是一致的，可做了解即可。

- QT/E GUI 界面设计

本实验上位机界面软件采用 QT/E4.6作为界面的开发软件包，具体 QT/E 在嵌入式系统中的移植，请参考 ARM 网关系统 QT 实验相关章节，这里不再赘述。以下主要介绍和本次试验相关的 QT 软件设计方法。

本实验程序，大体流程是首先调用网络客户端的 GetConnect(char *ipaddr,int port)接口函数，连接到服务器的 port 端口，然后开启了一个 Zigbee 线程（zigbeetopo.cpp），用来调用网络客户端的 Api_Cliect_GetZigBeeNwkTopo(&state)接口函数，获得 Zigbee 的节点信息链表（含有节点信息、传感器信息等），然后遍历链表监测各个节点的传感器状态，并根据状态进行相应的信号发射，如果异常还会发射短信报警信号。通过 Api_Cliect_GetGPRSSignal()获得 GPRS 初始化的信息，如果初始化不成功将不会进行短信报警，初始化成功将会调用 Api_Cliect_SendGprsMessage(char *phone,int sensor)进行 GPRS 短信报警。

QT 上层接收到相关信号后，会转到相应的槽函数进行处理，例如设置传感器报警图标并短信报警等。

同时针对控制型设备如电灯，设计了一个按钮 ui->pushButton_light，通过 connect(ui->pushButton_light,SIGNAL(clicked()),this,SLOT(pushButton_light_clicked()));建立信号槽连接，当按下其按钮，会相应的触发 pushButton_light_clicked()函数，在此函数中会调用网络客户端的 Api_Cliect_SetSensorStatus(nwkaddr,state)接口函数设置开关灯。

针对中断型传感器会通过 Api_Cliect_ClearIntlock()清除中断型传感器状态锁

界面构造函数：

```
MainWidget::MainWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::MainWidget)
{
    ui->setupUi(this);
    //设置主界面背景
    QPalette palette1=this->palette();
    palette1.setBrush(QPalette::Window,QBrush(QPixmap(":/images/up_wsn_bg2.jpg")));
    this->setPalette(palette1);
    this->setAutoFillBackground(true);
//设置智能家居界面背景
    palette1.setBrush(QPalette::Window,QBrush(QPixmap(":/images/05.jpg")));
    ui->frame_10->setPalette(palette1);
    ui->frame_10->setAutoFillBackground(true);
//设置报警电话号码及内容的长度
    phone_data=(char*)malloc(39);
    phone_num=(char *)malloc(ui->number_value->text().length());
    {//for htu sensor 设置门磁、红外对射、烟雾传感器及电灯图标
        ui->pushButton_light->setStyleSheet("QPushButton{background-image:
url(:/images/node_light.png);}"
        "QPushButton{background-color: rgba(255, 255, 255, 0);}"
        "QPushButton:hover{background-image: url(:/images/node_light.png);}"
        "QPushButton:pressed{background-image: url(:/images/node_light.png);}");
        ui->pushButton_light->setFlat(true);
        QPalette palette2=this->palette();

palette2.setBrush(QPalette::Window,QBrush(QPixmap(":/images/node_warning.png")));
        palette2.setColor(QPalette::WindowText,Qt::green);
        ui->label_warn->setPalette(palette2);
        ui->label_warn->setAutoFillBackground(true);
        int_label=new MyLabelHtu(ui->frame_11);
        int_label->setGeometry(QRect(260,350, 51, 51));
        int_label->setText(QString::fromUtf8("  门磁"));

palette1.setBrush(QPalette::Window,QBrush(QPixmap(":/images/node_dir.png")));
        palette1.setColor(QPalette::WindowText,Qt::red);
        int_label->setPalette(palette1);
        int_label->setAutoFillBackground(true);
        smog_label=new MyLabelHtu(ui->frame_11);
        smog_label->setGeometry(QRect(500,280, 51, 51));
        smog_label->setText(QString::fromUtf8("  烟雾"));

palette1.setBrush(QPalette::Window,QBrush(QPixmap(":/images/node_smog.png")));
        palette1.setColor(QPalette::WindowText,Qt::red);
        smog_label->setPalette(palette1);
        smog_label->setAutoFillBackground(true);
        irda_label=new MyLabelHtu(ui->frame_11);
        irda_label->setGeometry(QRect(20,250, 51, 51));
        irda_label->setText(QString::fromUtf8("  对射"));

palette1.setBrush(QPalette::Window,QBrush(QPixmap(":/images/node_magt.png")));
        palette1.setColor(QPalette::WindowText,Qt::red);
        irda_label->setPalette(palette1);
        irda_label->setAutoFillBackground(true);
        int_flag=0;
        smog_flag=0;
```

```
        irda_flag=0;
        connect(int_label,SIGNAL(clicked()),this,SLOT(int_show()));
        connect(smog_label,SIGNAL(clicked()),this,SLOT(smog_show()));
        connect(irda_label,SIGNAL(clicked()),this,SLOT(irda_show()));
    }
    char *ipaddr="192.168.12.248";//连接本地服务器，端口号是 7838
    int port=7838;
    cliect_thread = new Cliect();
    cliect_thread->GetConnect(ipaddr,port);
//建立相关的信号槽

connect(ui->pushButton_light,SIGNAL(clicked()),this,SLOT(pushButton_light_clicked()));//开关灯
    connect(cliect_thread,SIGNAL(sendMsgsignal(int )),this,SLOT(gprs_sendMsg(int)));//发送短信报警
    connect(cliect_thread,SIGNAL(Int_StateChanged_1()),this,SLOT(Int_change_1()));//门磁传感器异常

connect(cliect_thread,SIGNAL(Irda_StateChanged_1()),this,SLOT(Irda_change_1()));//红外对射传感器异常

connect(cliect_thread,SIGNAL(Smog_StateChanged_1()),this,SLOT(Smog_change_1()));//烟雾传感器异常
    connect(cliect_thread,SIGNAL(Int_StateChanged_0()),this,SLOT(Int_change_0()));//门磁传感器正常

connect(cliect_thread,SIGNAL(Irda_StateChanged_0()),this,SLOT(Irda_change_0()));//红外对射传感器正常

connect(cliect_thread,SIGNAL(Smog_StateChanged_0()),this,SLOT(Smog_change_0()));//烟雾传感器正常
    connect(cliect_thread,SIGNAL(sendTempHum(unsigned long )),this,SLOT(show_TempHum(unsigned long )));//显示温湿度
    connect(cliect_thread,SIGNAL(Sim(int )),this,SLOT(Set_Sim(int)));//
    connect(cliect_thread,SIGNAL(Send(int)),this,SLOT(Set_Send(int)));
    connect(this,SIGNAL(Int_StateChanged_1()),this,SLOT(Int_change_1()));
    connect(this,SIGNAL(Irda_StateChanged_1()),this,SLOT(Irda_change_1()));
    connect(this,SIGNAL(Smog_StateChanged_1()),this,SLOT(Smog_change_1()));
    connect(this,SIGNAL(Int_StateChanged_0()),this,SLOT(Int_change_0()));
    connect(this,SIGNAL(Irda_StateChanged_0()),this,SLOT(Irda_change_0()));
    connect(this,SIGNAL(Smog_StateChanged_0()),this,SLOT(Smog_change_0()));
    cliect_thread->start();
    zb_thread = new ZigbeeTopo();
    light_state=0;
    Sim_state=0;
    ui->state_sim->setText(QString::fromUtf8("未初始化"));
    cliect_thread->Api_Cliect_GetGPRSSignal();
    initPad();///初始化界面中的软键盘
}
```

以上函数即为 QT 主界面构造函数，具体代码参见工程源码。

发送线程及接收处理函数：

发送线程实现发送获取 Zigbee 节点链表命令（Api_Cliect_GetZigBeeNwkTopo()），然

后接收处理函数循环检测个节点传感器类型、状态，并根据状态发射相应的信号，进行报警。然后接收处理函数参照实验四部分说明。

```
ZigbeeTopo::ZigbeeTopo()
{
    cliect_thread1 = new Cliect();
    this->start();
}
ZigbeeTopo::~ZigbeeTopo()
{
}
void ZigbeeTopo::run(){
    while(1)
    {
        mutex.lock();
        cliect_thread1->Api_Cliect_GetZigBeeNwkTopo();
        mutex.unlock();
        usleep(800000);
    }
}
```

发送获取拓补信息、短信报警、获得 GPRS 初始化信息、清除中断锁命令帧的函数如下：

```
void Cliect::Api_Cliect_GetZigBeeNwkTopo()
{
    qDebug("Api_Cliect_GetZigBeeNwkTopo send \n");
    unsigned int buffer[3]={0};
    buffer[0]=0x15;
    buffer[1]=0x01;
    buffer[2]='\n';
    //qDebug("Api_Cliect_GetZigBeeNwkInfo send sockfd:%d\n",sockfd);
    int len=::write(sockfd, (unsigned int*)(&buffer), sizeof(buffer));
    if(len<=0)
    {
        qDebug("Api_Cliect_GetZigBeeNwkTopo send error\n");
    }
}
void Cliect::Api_Cliect_ClearIntlock()
{
    unsigned int buffer[3]={0};
    buffer[0]=0x15;
    buffer[1]=0x08;
    buffer[2]='\n';
    int len=::write(sockfd, (unsigned int*)(&buffer), sizeof(buffer));
    if(len<=0)
    {
        qDebug("Api_Cliect_ClearIntlock send error\n");
    }
}
void Cliect::Api_Cliect_GetGPRSSignal()
{
    qDebug("Api_Cliect_GetGPRSSignal send \n");
    unsigned int buffer[3]={0};
    buffer[0]=0x15;
    buffer[1]=0x07;
    buffer[2]='\n';
//qDebug("Api_Cliect_GetZigBeeNwkInfo send sockfd:%d\n",sockfd);
```

```
    int len=::write(sockfd, (unsigned int*)(&buffer), sizeof(buffer));
    if(len<=0)
    {
        qDebug("Api_Cliect_GetGPRSSignal send error\n");
    }
}
void Cliect::Api_Cliect_SendGprsMessage(char *phone,int sensor)
{
    qDebug("Api_Cliect_SendGprsMessage send \n");
    unsigned int buffer[15]={0};
    buffer[0]=0x15;
    buffer[1]=0x06;
    int i=0;
    qDebug("Api_Cliect_SendGprsMessage send :\n");
    for(i=0;i<11;i++)
    {
        buffer[2+i]=*(phone+i);
        qDebug("%d",*(phone+i));
    }
    qDebug("\n");
    buffer[13]=sensor;
    buffer[14]='\n';
//qDebug("Api_Cliect_GetZigBeeNwkInfo send sockfd:%d\n",sockfd);
    int len=::write(sockfd, (unsigned int*)(&buffer), sizeof(buffer));
    if(len<=0)
    {
        qDebug("Api_Cliect_SendGprsMessage send error\n");
    }
}
void Cliect::Api_Cliect_SetSensorStatus(unsigned int nwkaddr,unsigned int Mode)
{
    qDebug("Api_Cliect_SetSensorStatus send \n");
    unsigned int buffer[5]={0};
    buffer[0]=0x15;
    buffer[1]=0x03;
    buffer[2]=nwkaddr;
    buffer[3]=Mode;
    buffer[4]='\n';
//qDebug("Api_Cliect_GetZigBeeNwkInfo send sockfd:%d\n",sockfd);
    int len=::write(sockfd, (unsigned int*)(&buffer), sizeof(buffer));
    if(len<=0)
    {
        qDebug("Api_Cliect_SetSensorStatus send error\n");
    }
}
```

4. 实验步骤

进行本次试验之前，需要按照 UP-CUP IOT-6410-II 型嵌入式物联网综合实验系统产品光盘\演示程序烧写目录\img 的 UP-CUP IOT-6410-II 型演示程序烧写说明.doc，将 Zigbee 模块烧写上演示程序，以便进行接下来的 QT 界面上层实验。

- **编译源程序**

将本次实验源码目录 smarthome（光盘：UP-CUP IOT-6410-II 型嵌入式物联网综合实验系统产品光盘\物联网无线传感网络部分\exp\linux\QT_exp\smarthome），拷贝至宿主机 FC8 的/UP-CUP6410/目录下(当然任意目录皆可以)

本实验上位机界面软件采用 QT/E4.6 作为 ZIGBEE 网络拓扑界面的开发软件包，请参考 ARM 网关系统 QT 实验相关章节，进行 QT/E 在嵌入式系统中的移植，这里不再赘述。我们假设移植好的 QT/E 目录为/usr/local/Trolltech/Qt-embedded-4.6.2/,则我们进行实验按如下操作：

1、进入实验目录：

```
[root@localhost /]# cd /UP-CUP6410/smarthome/
[root@localhost smarthome]# ls
Makefile    main.o              moc_mainwidget.cpp    moc_zigbeetopo.o    qrc_images.o
zigbeetopo.cpp
api.o       mainwidget.cpp      moc_mainwidget.o      mylabelhtu.cpp      server
zigbeetopo.h
cliect.o    mainwidget.h        moc_mylabelhtu.cpp    mylabelhtu.h        smarthome
zigbeetopo.o
images      mainwidget.o        moc_mylabelhtu.o      mylabelhtu.o        smarthome.pro
main.cpp    mainwidget.ui       moc_zigbeetopo.cpp    qrc_images.cpp      ui_mainwidget.h
[root@localhost smarthome]#
```

2、清除中间代码，重新编译

```
[root@localhost smarthome]# make clean
rm -f moc_mainwidget.cpp moc_mylabelhtu.cpp moc_zigbeetopo.cpp
rm -f qrc_images.cpp
rm -f ui_mainwidget.h
rm -f main.o mainwidget.o mylabelhtu.o zigbeetopo.o api.o cliect.o moc_mainwidget.o
moc_mylabelhtu.o moc_zigbeetopo.o qrc_images.o
rm -f *~ core *.core
[root@localhost smarthome]# /usr/local/Trolltech/Qt-embedded-4.6.2/bin/qmake
-project
QFileInfo::absolutePath: Constructed with empty filename
[root@localhost smarthome]# /usr/local/Trolltech/Qt-embedded-4.6.2/bin/qmake
[root@localhost smarthome]# make
```

当前目录下生成可执行程序 smarthome.

3、将 smarthome 拷贝到 tftp 下载目录

```
[root@localhost smarthome]# cp smarthome /tftpboot
```

◆ 下载程序测试

1、启动 UP-CUP IOT-6410-II 型实验系统。连好网线、串口线。

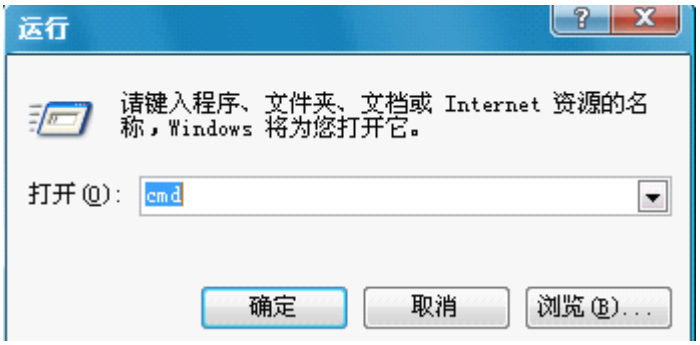
2、下载演示程序（如若下载了演示程序，此步骤可省略）

需要按照UP-CUP IOT-6410-II型嵌入式物联网综合实验系统产品光盘\演示程序烧写目录\img的UP-CUP IOT-6410-II型演示程序烧写说明.doc，将Zigbee模块烧写上演示程序，以便进行接下来的QT界面上层实验。

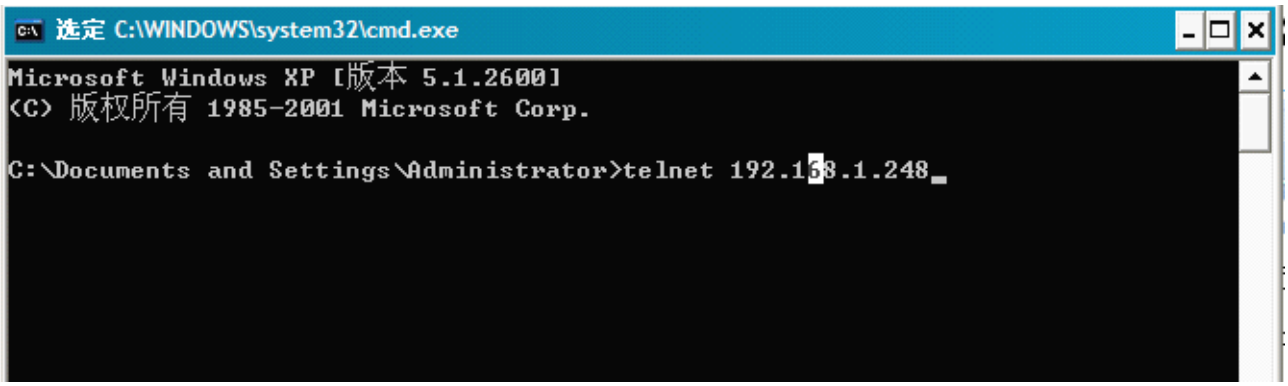
3、进入开发板实验目录

启动开发板，默认会自启动演示程序，我们必须先关闭它，按照如下方法，若已关闭自启动，可省略。

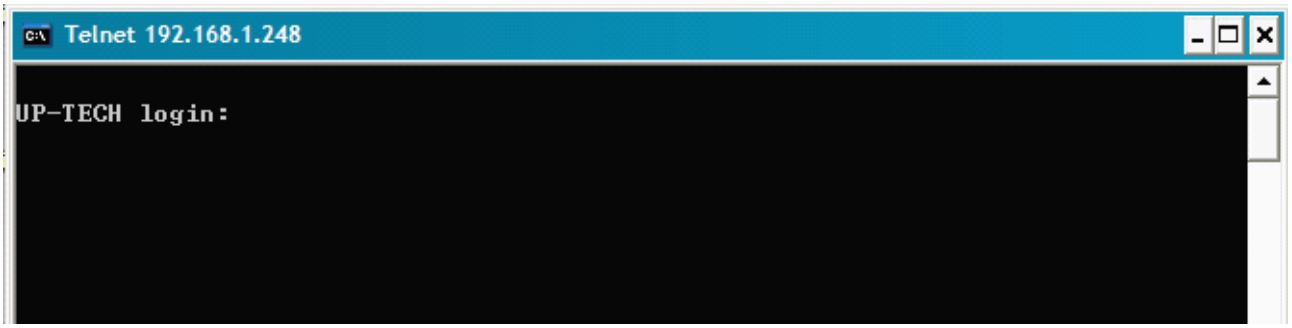
点击PC机“开始”->“运行”->输入“cmd”回车，如下图所示：



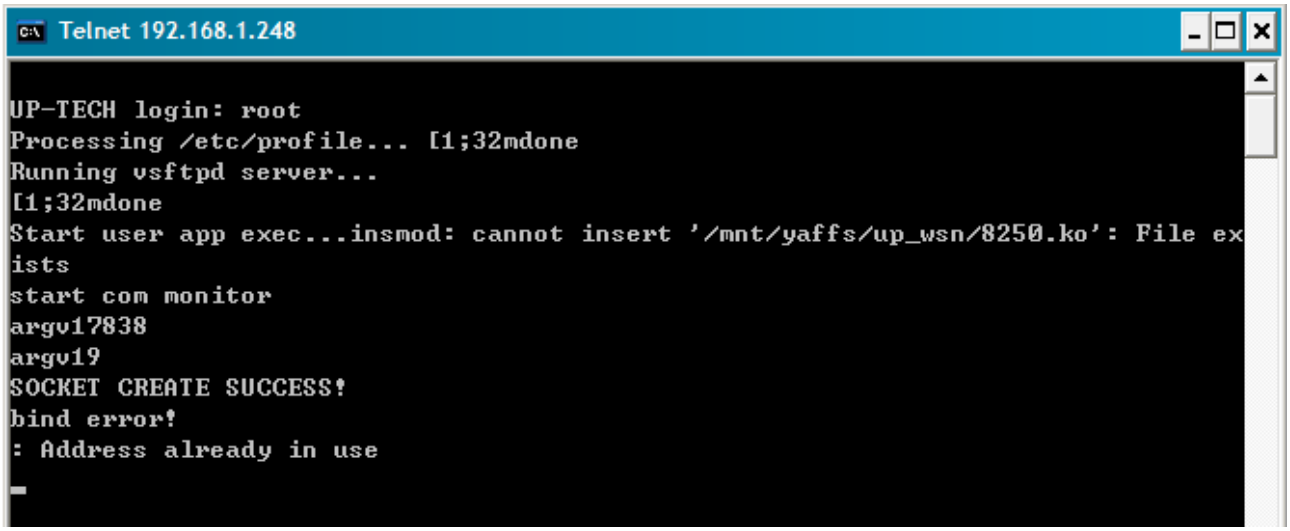
然后通过telnet登陆开发板，命令为telnet 192.168.1.248,如下图所示：



回车：



输入root登陆：



“ctrl+c”终止掉后台程序，然后进入/mnt/yaffs/up_wsn 目录：

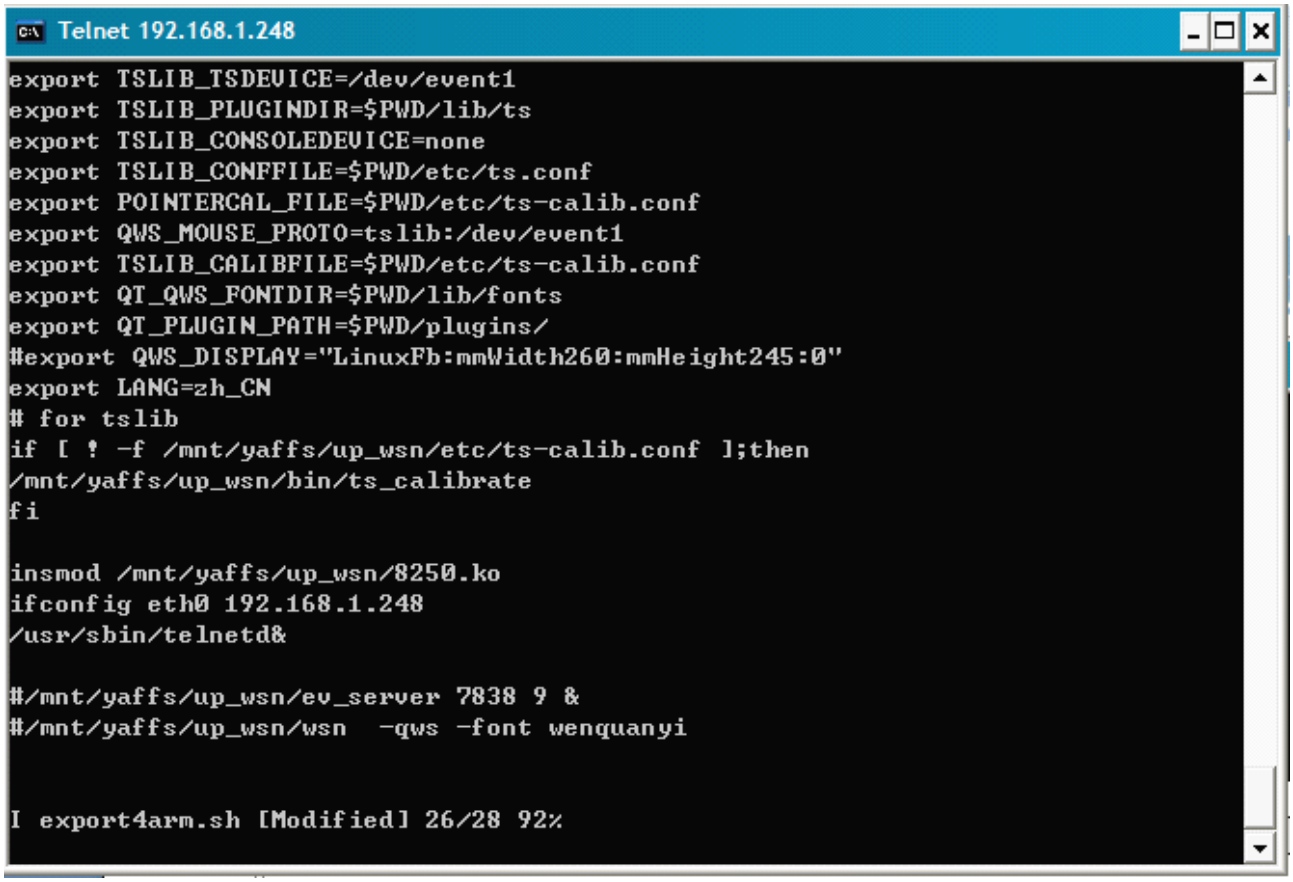
[root@ up_wsn]# vi export4arm.sh

修改自启动脚本 export4arm.sh，注释掉（“#”为注释符）以下两行

/mnt/yaffs/up_wsn/ev_server 7838 9 &

/mnt/yaffs/up_wsn/wsn -qws -font wenquanyi

更改如下图所示：



4、下载实验程序

```
[root@up_wsn]# tftp -gr smarthome 192.168.1.145
[root@localhost up_wsn]# ls
8250.ko bin etc ev_server export4arm.sh lib plugins smarthome translations web wsn
[root@localhost up_wsn]#
```

5、修改环境变量脚本

```
[root@localhost up_wsn]# cp export4arm.sh export4arm-smarthome.sh
[root@localhost up_wsn]#vi export4arm-smarthome.sh
```

脚本内容如下：

```
#!/bin/bash
cd /mnt/yaffs/up_wsn/
export QTDIR=$PWD
export LD_LIBRARY_PATH=$PWD/lib
export TSLIB_TSDEVICE=/dev/event1
export TSLIB_PLUGINDIR=$PWD/lib/ts
export TSLIB_CONSOLEDEVICE=none
export TSLIB_CONFFILE=$PWD/etc/ts.conf
export POINTERCAL_FILE=$PWD/etc/ts-calib.conf
export QWS_MOUSE_PROTO=tslib:/dev/event1
export TSLIB_CALIBFILE=$PWD/etc/ts-calib.conf
export QT_QWS_FONTDIR=$PWD/lib/fonts
```

```
export QT_PLUGIN_PATH=$PWD/plugins/
#export QWS_DISPLAY="LinuxFb:mmWidth260:mmHeight245:0"
export LANG=zh_CN
# for tslib
if [ ! -f /mnt/yaffs/up_wsn/etc/ts-calib.conf ];then
/mnt/yaffs/up_wsn/bin/ts_calibrate
fi

insmod /mnt/yaffs/up_wsn/8250.ko
ifconfig eth0 192.168.1.248
/usr/sbin/telnetd&
/mnt/yaffs/up_wsn/ev_server 7838 9&
/mnt/yaffs/up_wsn/wsn    -qws -font wenquanyi
```

将脚本 export4arm-smarthome.sh 中/mnt/yaffs/up_wsn/wsn -qws -font wenqunayi 一行修改为：

/mnt/yaffs/up_wsn/smarthome -qws -font wenquanyi

```
#!/bin/bash
cd /mnt/yaffs/up_wsn/
export QTDIR=$PWD
export LD_LIBRARY_PATH=$PWD/lib
export TSLIB_TSDEVICE=/dev/event1
export TSLIB_PLUGINDIR=$PWD/lib/ts
export TSLIB_CONSOLEDEVICE=none
export TSLIB_CONFFILE=$PWD/etc/ts.conf
export POINTERCAL_FILE=$PWD/etc/ts-calib.conf
export QWS_MOUSE_PROTO=tslib:/dev/event1
export TSLIB_CALIBFILE=$PWD/etc/ts-calib.conf
export QT_QWS_FONTDIR=$PWD/lib/fonts
export QT_PLUGIN_PATH=$PWD/plugins/
#export QWS_DISPLAY="LinuxFb:mmWidth260:mmHeight245:0"
export LANG=zh_CN
# for tslib
if [ ! -f /mnt/yaffs/up_wsn/etc/ts-calib.conf ];then
/mnt/yaffs/up_wsn/bin/ts_calibrate
fi

insmod /mnt/yaffs/up_wsn/8250.ko
ifconfig eth0 192.168.1.248
/usr/sbin/telnetd&
/mnt/yaffs/up_wsn/ev_server 7838 9&
/mnt/yaffs/up_wsn/smarthome    -qws -font wenquanyi
```

/mnt/yaffs/up_wsn/ev_server 7838 9& 表示后台执行 ev_server 程序（为网络通信服务器程序）

/mnt/yaffs/up_wsn/smarthome -qws -font wenquanyi 中 smarthome 为要执行的程序

6、执行脚本进行测试应用程序

```
[root@localhost up_wsn]#./export4arm-smarthome.sh
```

7、测试结果

执行脚本之后，在开发板的 LCD 显示屏上显示如下图形界面：

当传感器异常时会触发相应的报警，进行发送短信报警，如果没有设置电话号码的话，会弹出如上“电话号码长度错误”的对话框，可关闭对话框，在界面的右下角输入电话号码并点击“保存”按钮。

点击“电灯”按钮，可开关传感器上的 LED 灯。在右上角会显示相关信息，报警时也会自动更新。

界面的左上角会显示温湿度值，此时没有连接温湿度传感器，所以显示为空。

备注：当 SIM 初始化不成功是不发送报警短信的。