“智能”家居系统原理及流程

之前发了一个介绍，拍了一个演示的视频，没看过的可以先看一下。主要就是实现手机对设备的控制以及获得设备的数据，由设备+服务器+客户端组成。流程以及使用到哪些技术等都未进行说明，这次来说一下吧，主要是说明主要的流程及重要的逻辑部分，那些细到哪个数组这样的细节当然省略，还有代码的结构也忽略啦，一下所有逻辑和代码均为应用层滴哦~~。

一：系统组成及涉及到的技术

1. 设备（硬件+嵌入式软件）

- 角色：作为设备存在，上电时登录服务器，可以随时与服务器进行通信、数据交换，并根据服务器的指令执行相应的动作
- 硬件：
    - MCU：任何MCU都可以滴，这里可以用cannon，也可以用其它MCU
    - 外设：要感知的数据的传感器和执行器，以及传输用的wifi（这里使用esp8266）
- IDE：因为是用STM32，所以当然是MDK,版本>=5，或者eclipse，或者文本编辑器+gcc，随你便
- 语言：c语言或者C++,cannon官方的库是c语言，我自己之前一直都是用的C++/C混合的方式
- 相关技术：C/C++基本语法，USART,ADC,GPIO,PWM等片上驱动，片外模块的驱动（如wifi模块、传感器、执行器驱动），网络通信应用，CRC校验，加密算法

2. 服务器

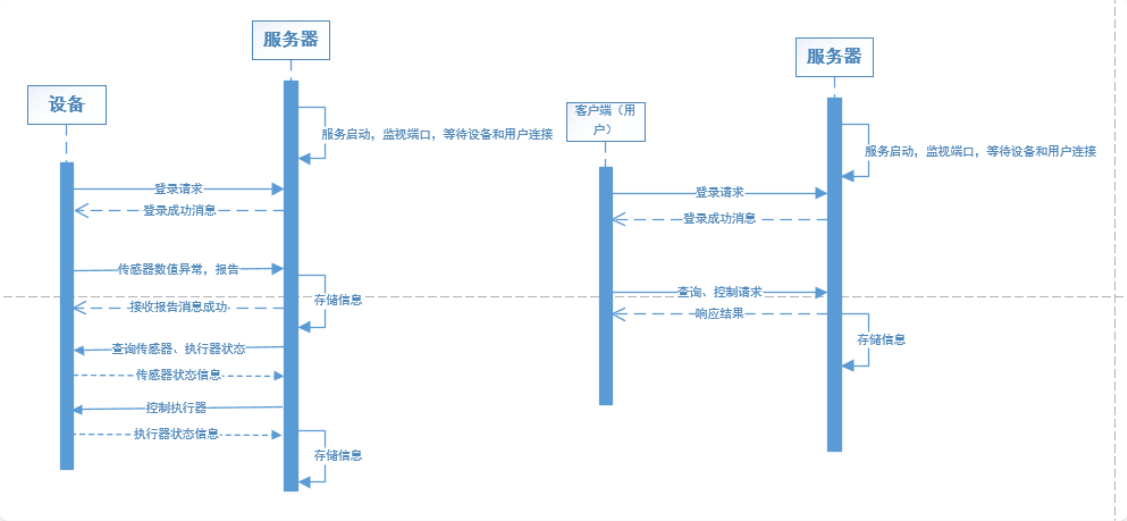
- 角色：作为服务器，作为用户与设备交互的中间人以及数据管理等。监听来自设备的登录请求、紧急消息以及向设备发送控制、查询命令；监听用户手机端的登录，用户发起的查询、控制命令等
- 语言：java
- IDE:eclipse/Myeclipse
- 相关技术：java基本语法，多线程，socket通信，数据库相关操作，http通信

3. 手机端（Android）。

- 角色：作为用户与设备交互的接口。用户使用手机端应用进行登录、控制设备、查询设备状态等等
- 语言：java
- IDE:Android Studio 2.1
- 相关技术：Activity相关界面设计，java基础语法，多线程，socket通信，sharepreference，Android相关

二：协议

3. 通信顺序图



QQ截图20160525233718.png

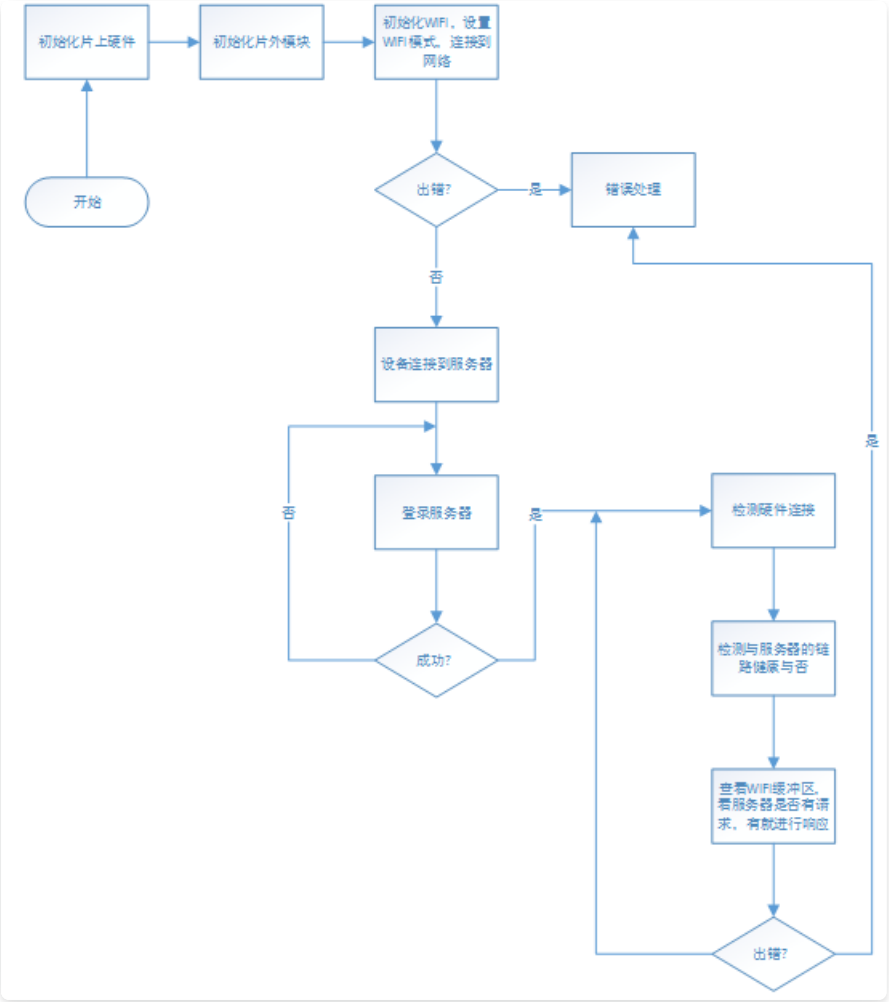
2. 服务器<---->设备

帧头	数据类型	操作类型	设备号	UTC时间戳	信息长度	信息	CRC校验码
2字节	2字节	1字节	6字节	6字节	2字节	N字节	2字节
0xabac		0x01:控制请求 0x02:信息响应 0x03:信息询问					前面所有的CRC16值

3. 手机端<---->服务器

帧头	数据类型	操作类型	session码	UTC时间戳	信息长度	信息	CRC校验码
2字节	2字节	1字节	32字节	6字节	2字节	N字节	2字节
0xabac		0x01:控制请求 0x02:信息响应 0x03:信息询问					前面所有的CRC16值

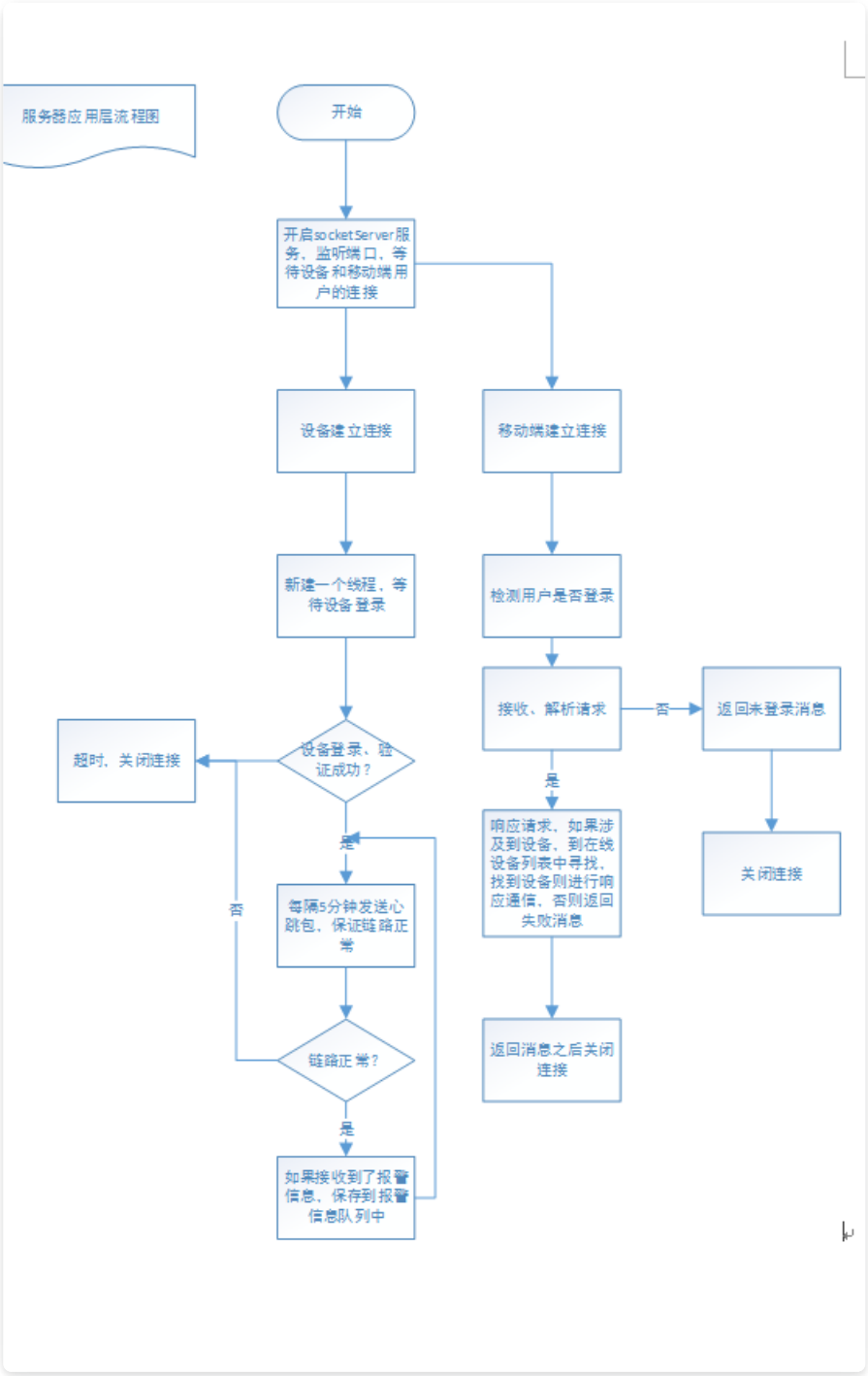
二：设备端应用层程序流程及代码



QQ截图20160525233601.png

```
/**
  *初始化
  */
void App::Init()
{
    //关闭LED
    light.Off();
    //步进电机失能
    stepMotor.Disable();
    //设置步进电机速度，值越小速度越大
    stepMotor.Run(true,2);
    //初始化wifi
    WifiInit();

    TaskManager::DelayS(5);
    if(!SignIn())
        com1<<"sign in fail!!!!!!!!!!!!\n\n\n";

    com1<<"initialize complete!\n";
}

/**
  *循环体
  *@pre 传感器值已经在loop前获取到成员变量中
  */
void App::loop()
{
    static float time;
    if(TaskManager::Time()-time>=20)
    {
        time = TaskManager::Time();
        com1<<".";
    }
    //硬件健康状态检查
    if(!CheckHardware())
        com1<<"haredware error!\n";


    //连接状态检查
    if(!CheckConnectionToServer())
        com1<<"connection to server error!\n";


    //接收来自服务器的数据
    ReceiveAndDeal();
}

void App::WifiInit()
{
    //WIFI initialize
    if(!wifi.Kick())//检查连接
    {
        com1<<"no wifi!\n\n\n";
        light.Blink(3,300);
        return;
    }
    else
        com1<<"wifi is healthy\n";
    wifi.SetEcho(false);//关闭回响
    wifi.SetMode(esp8266::esp8266_MODE_STATION_AP,esp8266::esp8266_PATTERN_DEF);//设置为station+ap模式
    wifi.SetMUX(false);//单连接模式
    wifi.SetApParam(mApSetName,mApSetPasswd,esp8266::esp8266_PATTERN_DEF);//设置热点信息
    wifi.JoinAP(mApJoinName,mApJoinPasswd,esp8266::esp8266_PATTERN_DEF);//加入AP

    //连接服务器
    if(!wifi.Connect((char*)"192.168.191.1",8090,Socket_Type_Stream,Socket_Protocol_IPV4))
    {
        com1<<"connect server fail!\n\n\n";
        light.Blink(4,300);
        return;
    }
    com1<<"WIFI initialize complete!\n";
    light.Blink(2,200);
    light.Off();
}
```

三：服务器端应用层程序流程及代码



QQ截图20160525233842.png

监视端口，等待连接

```
//开启一个线程监视用户的请求
new Thread(){

@Override
public void run() {
    try{
        // 1.创建一个服务器端Socket,即ServerSocket,指定绑定的端口，并监听此端口
        ServerSocket serverSocket = new ServerSocket(8099);
        Socket socket = null;

        // 2.调用accept()方法开始监听，等待客户端的连接
        System.out.println("Server Started，waiting for User request at port 8099: ");
        while (true) {// 循环监听等待客户端的连接
            socket = serverSocket.accept();// 调用accept()方法开始监听，等待客户端的连接
            ServerToUserThread userThread = new ServerToUserThread(mSocketList,socket);// 创建一个新的线程响应客
户端的连接
            userThread.start();// 启动线程
            System.out.println("用户发起请求，地址："+socket.getInetAddress());
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

}.start();


//监视设备的连接
try {
    // 1.创建一个服务器端Socket,即ServerSocket,指定绑定的端口，并监听此端口
    ServerSocket serverSocket = new ServerSocket(8090);
    Socket socket = null;

    // 2.调用accept()方法开始监听，等待客户端的连接
    System.out.println("Server Started，waiting for devices connect at port 8090:");
    while (true) {// 循环监听等待客户端的连接
        socket = serverSocket.accept();// 调用accept()方法开始监听，等待客户端的连接
        ServerToDeviceThead serverThread = new ServerToDeviceThead(socket);// 创建一个新的线程响应客户端的连接
        serverThread.start();// 启动线程
        mSocketList.add(serverThread);//储存连接信息
        showInfo(socket);

    }

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

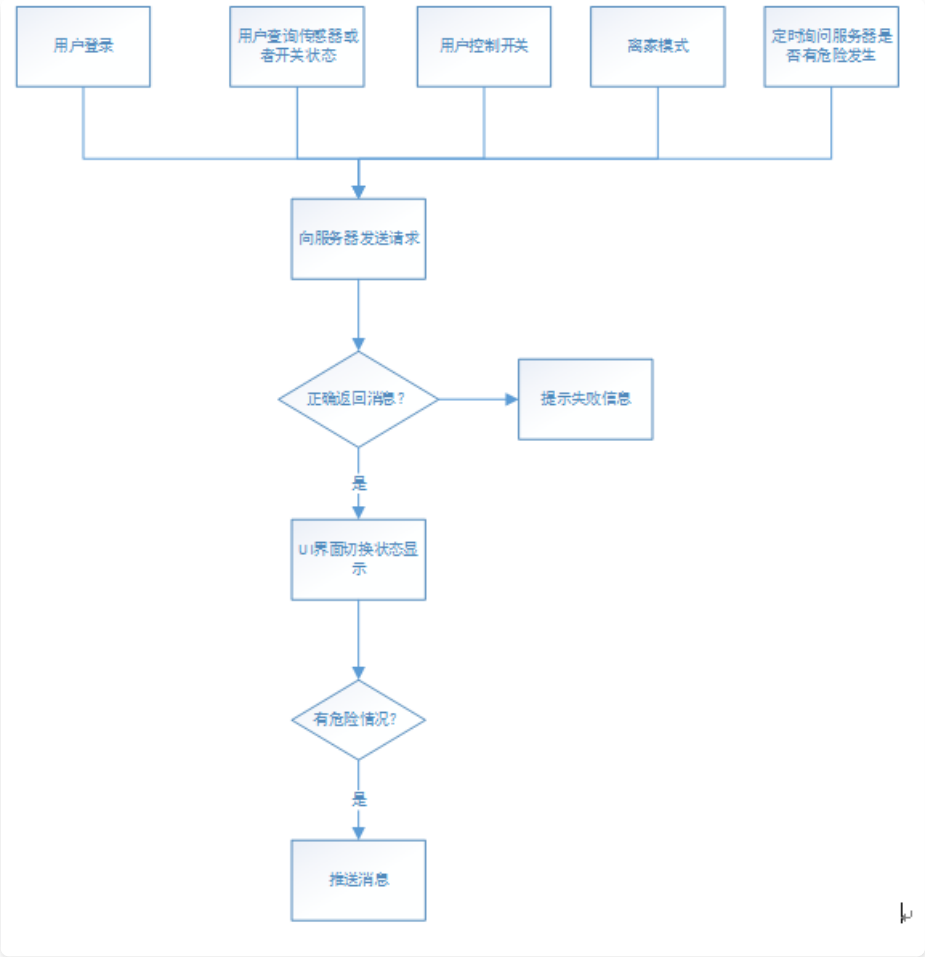
ServerToDeviceThread类主要执行的内容：

```
// 获取输入流，并读取客户端信息
if(!getIn_Out_stream())
{
    System.out.println("获取输入输出流错误");
    Close();
    return;
}

long keepAliveTime = Date_TimeStamp.timeStamp();
while(true){
    //登录检测
    if(!mIsSginIn){
        try {
            socket.setSoTimeout(20000);
        } catch (SocketException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //等待登录请求
        if(mToDevices.WaitSignIn(mSignInfo)){
            System.out.println("设备登录请求到达，设备号："+mSignInfo.device+"用户名："+mSignInfo.userName);
            //用户和设备验证
            if(!SignInVerify()){//验证失败
                System.out.println("登录验证失败！！！！");
                break;
            }
            System.out.println("设备登录成功!!设备号："+mSignInfo.device+"用户名："+mSignInfo.userName);
            mIsSginIn = true;
            try {
                socket.setSoTimeout(5000);
            } catch (SocketException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            continue;
        }
        if(Date_TimeStamp.timeStamp()-startTime>20){//20s还没有登录则关闭连接
            System.out.println("等待登录请求超时");
            Close();
        }
        continue;
    }

    //心跳保持包,5分钟一次
    if(Date_TimeStamp.timeStamp()-keepAliveTime>=300){
        keepAliveTime = Date_TimeStamp.timeStamp();
        int i = 0;
        for(;i<5;++i){
            if(!mToDevices.KeepAlive(mSignInfo.device)){
                System.out.println("心跳保持失败,正在重试,重试次数："+i);
                continue;
            }
            System.out.println("链路保持成功！");
        }
        if(i==5)
            System.out.println("心跳保持失败!!!");
    }

}
```

手机端与服务器端跟服务器端与设备端原理一样，只是在多了一个查询设备的过程

```
//从在线设备列表中获得该设备号的设备通信的对象
ToDevices toDevice = null;
for (int i = 0;i<mSocketList.size();++i) {
    if(!mSocketList.get(i).IsAlive())//已经关闭连接了，可以移除
        mSocketList.remove(i);
    else{//连接还正常（设备在线）//判断是否与要通信的设备名相同
        if(mSocketList.get(i).mSignInfo.device.equals(device)){
            toDevice = mSocketList.get(i).mToDevices;
            break;
        }
    }
}
if(toDevice==null){
    System.out.println("该设备没有登录,无法连接到设备！！！");
    Close();
    return;
}
//这里利用获得的ToDevices对象跟设备进行通信即可
```

四：手机端应用层程序流程及代码



QQ截图20160525234016.png

移动端较为简单，只是很费时间。

- 需要注意的是，与服务器和设备之间的通信不同，虽然都是使用socket进行通信，前者建立的连接在设备正常在线过程中会一直保持，并使用心跳包进行链路保持，但手机和服务器之间的通信是使用的无连接通信，如同http一样，即建立连接-->发起请求-->请求响应-->关闭连接。无需长期保持，因此，服务器需要使用session来保存用户登录信息。用户登录时，服务器会发送一个session码给用户，用户收到后在后面的每一次请求中都会在消息帧中携带这个session码来作为身份验证的依据。

- 另一个需要注意的是，Android由于有UI界面，即主线程为UI线程，而网络操作往往很耗费时间，为了避免在等待网络响应的时候UI线程阻塞而导致UI界面假死（程序失去响应）的情况，所有的网络操作一定要记得在新的线程中操作。如果操作后还要修改UI线程中的东西或者向UI线程中返回值，可以使用Runnable来处理网络操作，Handler来进行处理网络操作结束后的UI操作，比如登录：

```
Handler SignInHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        Bundle data = msg.getData();
        boolean result = data.getBoolean("result",false);
        if ( result) {
            Toast.makeText(getApplicationContext(), "登录成功", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(getApplicationContext(), "登录失败", Toast.LENGTH_SHORT).show();
        }
    }
};

/**
 * 网络操作相关的子线程
 */
Runnable SignIn = new Runnable() {

    @Override
    public void run() {
        // TODO
        // 在这里进行 http request.网络请求相关操作
        boolean result = mToServer.SignIn(mUser);
        Message msg = new Message();
        Bundle data = new Bundle();
        data.putBoolean("result", result);
        msg.setData(data);
        SignInHandler.sendMessage(msg);
    }
};
```

然后调用

```
new Thread(SignIn).start();
```