**“蜂信(Beecom)智能开关系统”**

**之**

**“蜂信智能中心”软件详细设计**

拟制： 胡冰 日期：2017-02-18

审核： 日期：

## 修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 说明 | 版本 | 作者 |
| 2017-02-18 | “蜂信（Beecom）智能开关系统”之“蜂信智能中心”软件详细设计初稿 | V1.0 | 胡冰 |
|  |  |  |  |

## 术语和缩写

1、“服务器”：“蜂信智能服务器”简称；

2、“开关”：“蜂信智能开关”简称；

3、“中心”：“蜂信智能中心”简称；

4、LUI：Liquid User Interface——液晶用户界面；

5、SP：Beecom-S Protocol层；

## 范围

本文档是“中心”软件架构的详细设计，具体包括“中心”的软件模块划分、模块间通信与协作、模块进程设计，为开发者在实现“中心”软件功能时提供参考。

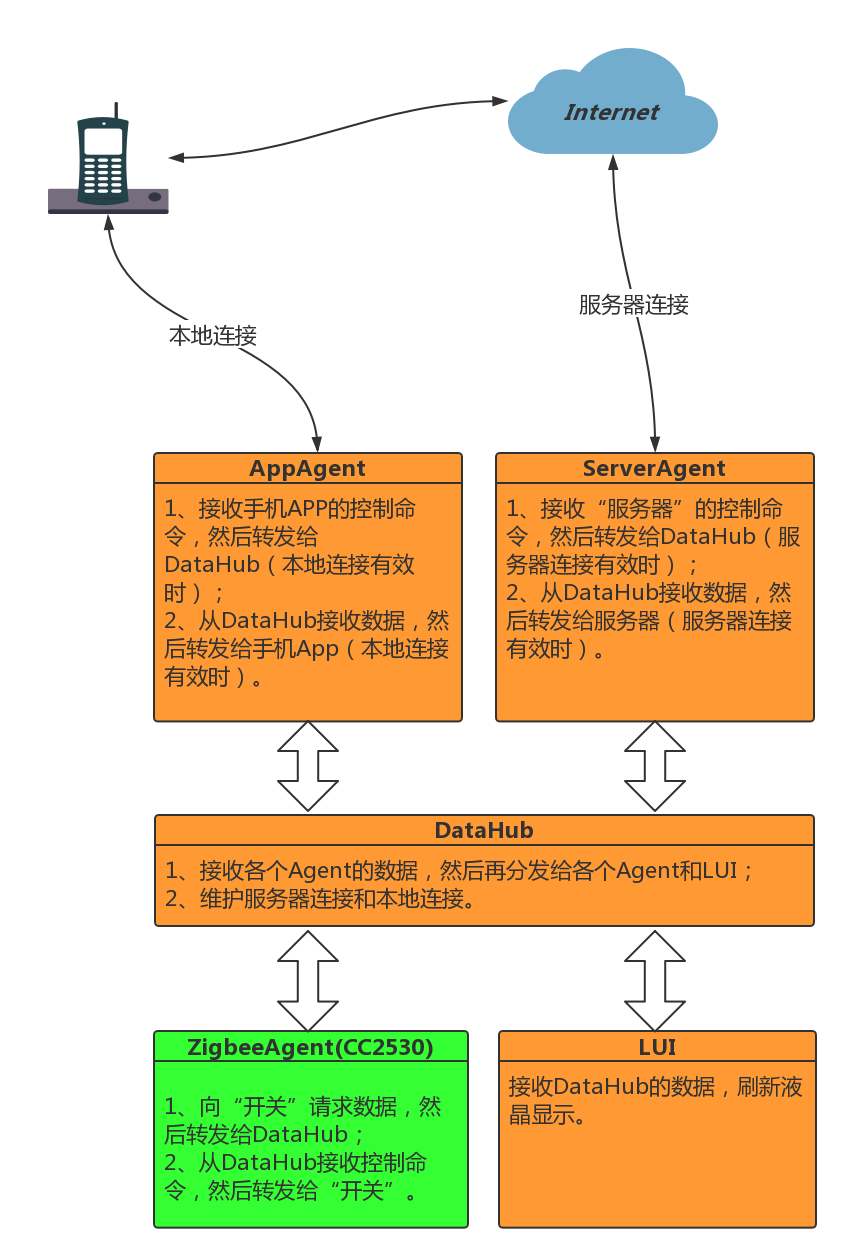
## 一、概述

在“蜂信智能开关系统”中，“中心”是WIFI和ZIGBEE之间的通信转换器，联系起手机APP、“服务器”和“开关”。在V1.0中（当前版本），以“最简设计”、“最小满足功能”指导原则来设计本软件，此中难免有对内存、CPU等资源的粗放使用，这些将在V1.0的升级版本中去优化。

## 二、模块划分

“中心”的软件设计将分成5个模块：AppAgent，ZigbeeAgent，ServerAgent，DataHub和Liquid User Interface(LUI)。如“图2-1”， AppAgent、ZigbeeAgent和ServerAgent分工“中心”和手机APP、“开关”和“服务器”的数据交互，并且将自身的链接状态（“断连”、“通信正常”等）反馈给DataHub。LUI负责界面的刷新，以及完成和用户的信息交互业务。DataHub是“中心”的数据分发总站，它收集其他模块发送过来的数据，然后进行判断将哪些数据推送至哪些模块的消息队列中去。同时也负责维护各个模块与外界的通讯状态，“中心”和手机APP的通信存在两条路，一个是通过AppAgent，在同一局域网下与手机APP连接（本地连接）；另一个是通过ServerAgent，通过Internet连接“服务器”，然后与手机APP连接（服务器连接），DataHub还要负责处理这两种连接的切换处理。其他模块的作用基本就是数据转发，“中心”的所有业务逻辑处理均在DataHub中完成。

另外还需要注意一点，ZigbeeAgent模块是属于CC2530，不占用主控芯片资源，其他4个模块则均属于主控芯片。



**图2-1 “中心”的各个模块功能划分**

## 三、模块间通信

AppAgent、ServerAgent和LUI模块通过提供数据交互的接口函数供DataHub使用，从而实现模块间通信。因为ZigbeeAgent模块并不运行在主控芯片上，所以ZigbeeAgent和DataHub的通信是通过串口实现的，通信协议请参见《Beecom-S Protocol》，DataHub通过使用协议API与ZigbeeAgent通信。

## 四、模块间协作

### 1. 初始化

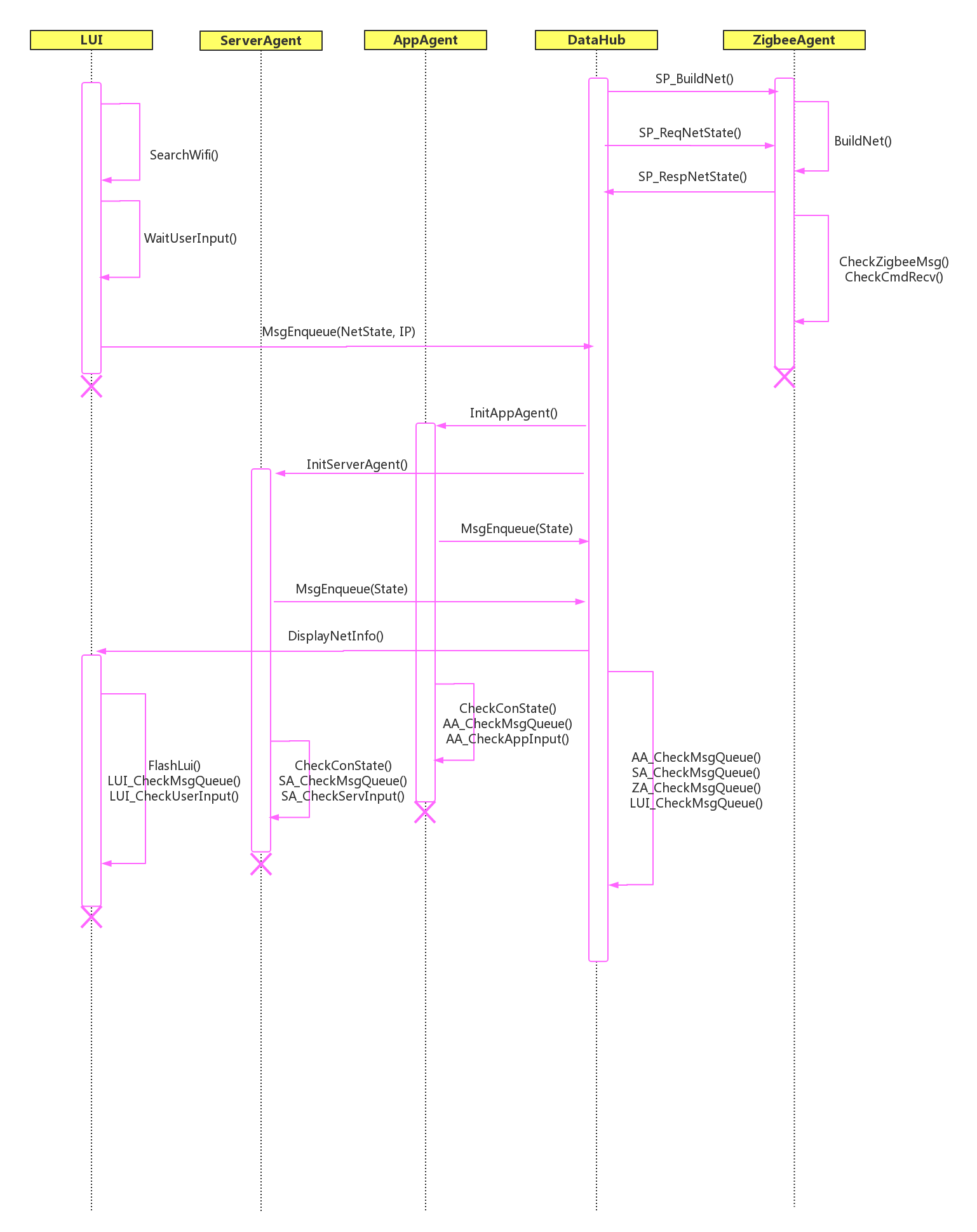
模块与模块之间的通信是通过消息队列完成，除DataHub外，每个模块均需要各自维护两个消息队列：InQueue和OutQueue。InQueue用于接收从外界环境输入的数据（wifi信号、zigbee信号等），OutQueue用于接收DataHub发送到本模块的数据，用于输出到外界环境。

DataHub的主要任务就是维护其他模块的InQueue和OutQueue（一共8个队列）。根据各个模块的InQueue的数据信息，然后根据业务逻辑，将相应的命令、数据等压入对应模块的OutQueue中去。

图4-1为“中心”的初始化流程，首先，DataHub向ZigbeeAgent发送建网命令，并一直询问其网络状态。另一方面，LUI搜索周边WIFI热点，并等待用户选择和输入密码。ZigbeeAgent完成建网后，向DataHub返回网络状态。然后循环检测Zigbee信号和DataHub发送过来的命令。由于ZigbeeAgent和DataHub不在同一芯片上运行，所以ZigbeeAgent的InQueue和OutQueue由DataHub代为管理。到此，Zigbee数据通路建立完成。

LUI获得用户输入的WIFI账号密码，然后加入路由器网络获得网络信息（IP等），并将网络信息压入InQueue中。如果网络信息有效，DataHub则调用接口，初始化AppAgent和ServerAgent。AppAgent和ServerAgent初始化成功后，向DataHub返回状态数据，然后循环检测Wifi信号和DataHub发送过来的数据。到此，Wifi数据通路建立完成。

DataHub检测到获得APPAgent和ServerAgent的状态数据后，调用LUI接口，刷新液晶屏信息。然后循环检测4个模块的InQueue。到此，“中心”的初始化工作完成。



**图4-1 “中心”初始化**

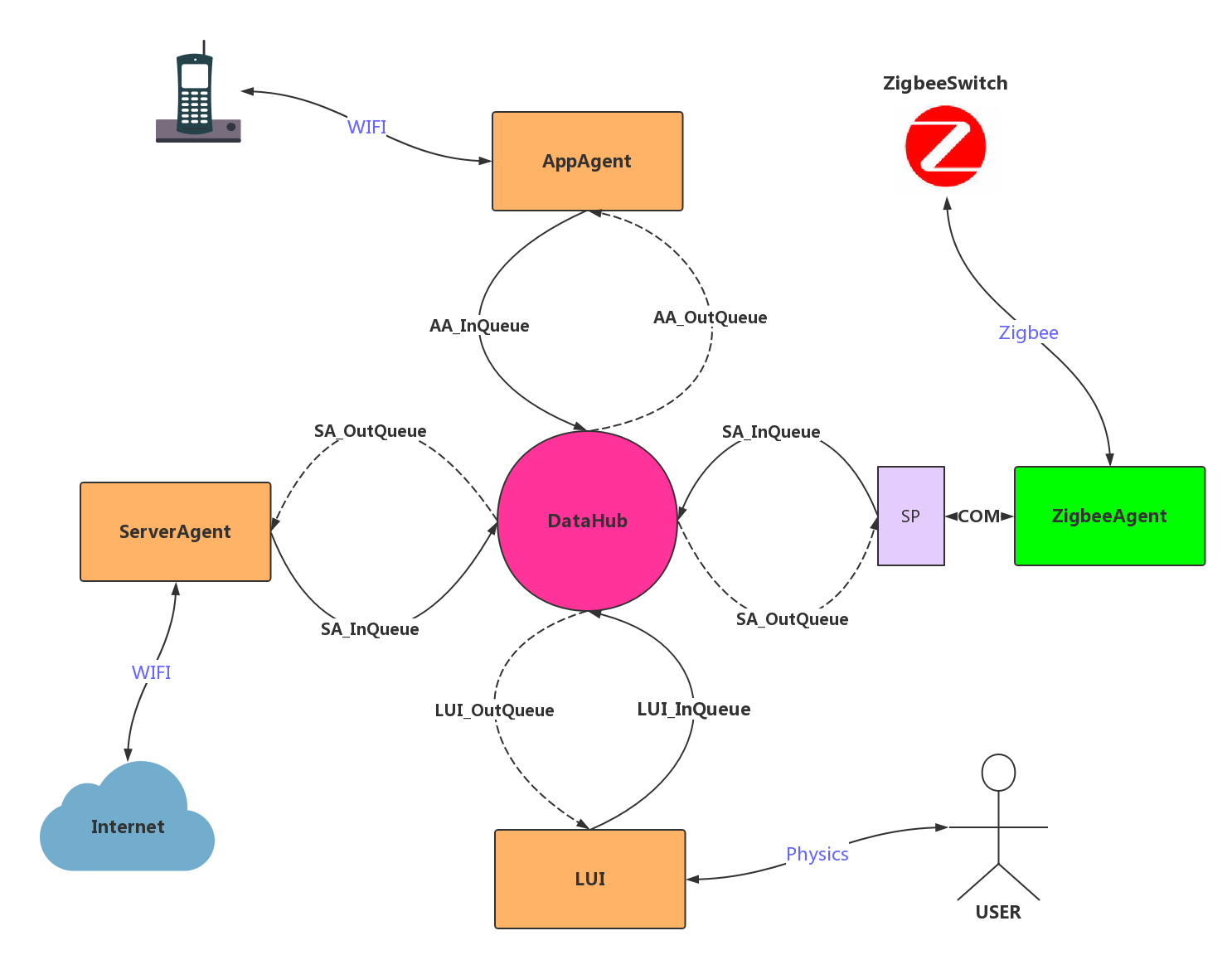
### 2. 运行

除DataHub外的其他模块在收到外界输入后，将数据Enqueue到自己的InQueue中。DataHub循环检测各个InQueue，当发现模块InQueue中有元素时，将其Dequeue，根据需要做相应处理，然后Enqueue到对应模块的OutQueue中。

InQueue：DataHub只做Dequeue操作，其他模块只做Enqueue操作。

OutQueue：DataHub只做Enqueue操作，其他模块只做Dequeue操作。

因为ZigbeeAgent需要通过COM通信，所以中间插入SP层进行数据格式转换，SP层不作为单独模块，嵌入在DataHub中作为DataHub的一部分。



**图4-2 “中心”正常运作**

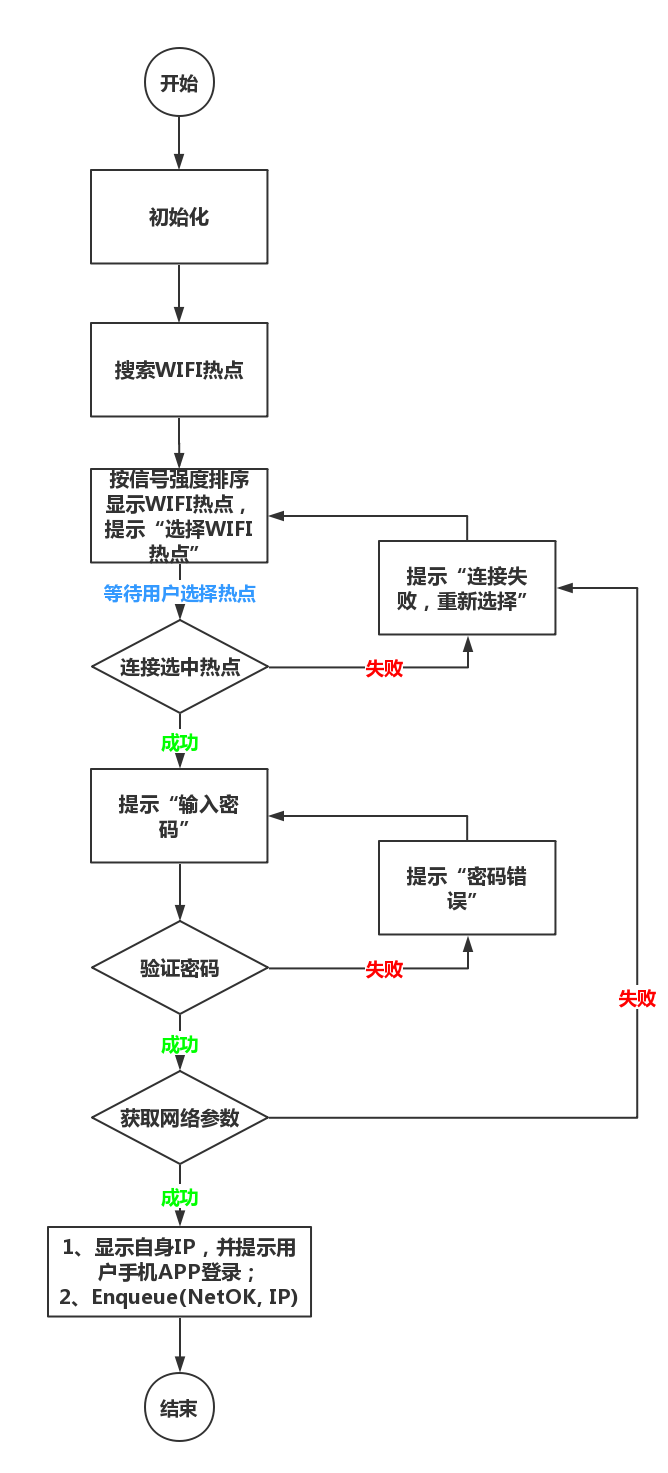
## 五、模块进程

每个模块的功能都由每个模块的进程（task）去执行。每个模块各有一个进程，分别为TaskLUI、TaskAppAgent、TaskServerAgent、TaskZigbeeAgent、TaskDataHub。除TaskZigbeeAgent之外，其他其他4个进程共同分享主控芯片的资源，彼此之间优先级相等。TaskZigbeeAgent独立运行在CC2530上，不占用主控芯片资源。

### 1. void TaskLUI(void \* para)

#### 初始化

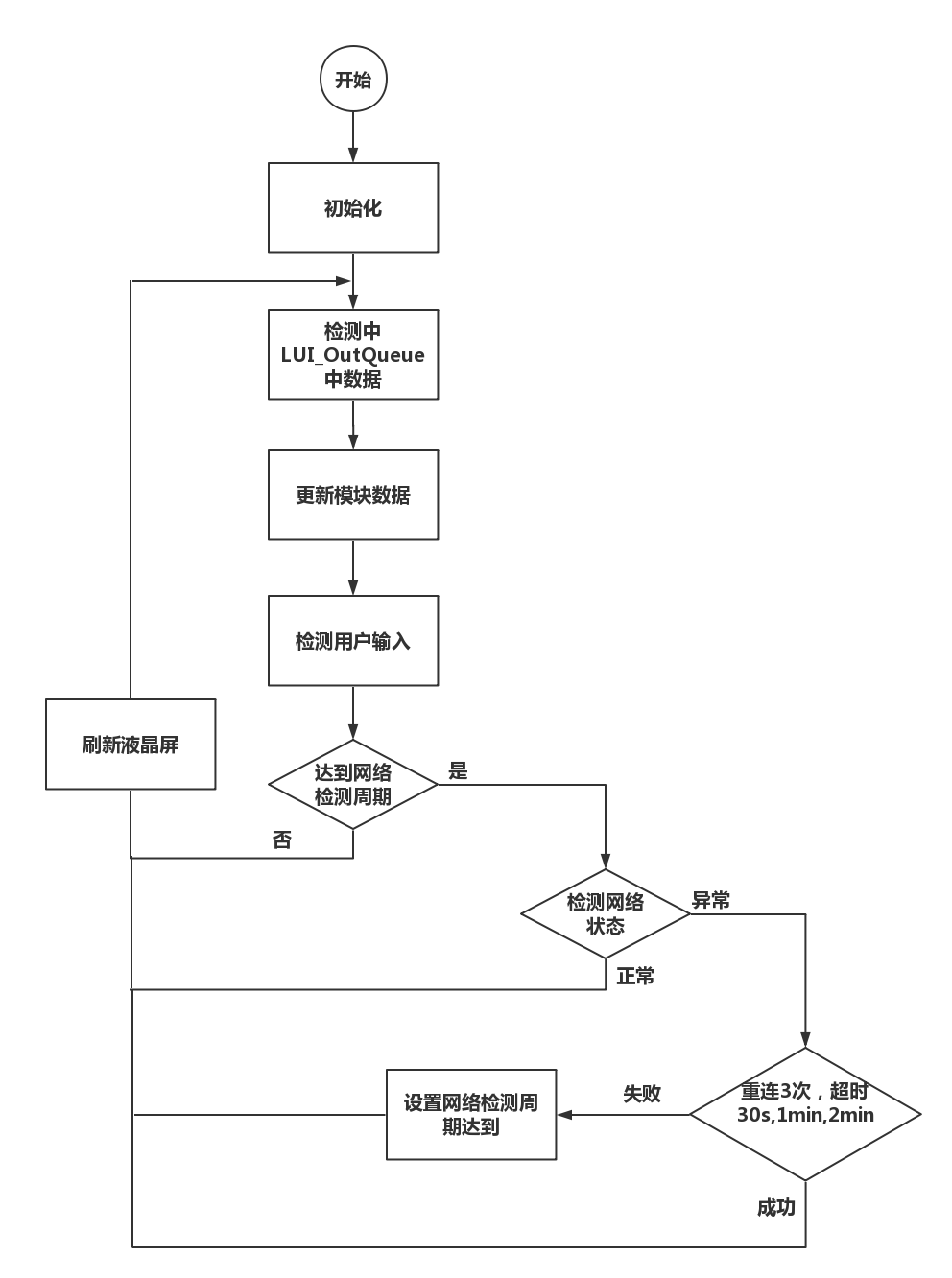
如图4-3，TaskLUI初始化的主要工作就是搜索WIFI热点，接着接收用户输入的WIFI账号密码，连接无线路由器，获取有效网络参数，并Enqueue给DataHub，期间还要做一些相应的提示语。



**图4-3 TaskLUI初始化**

#### 常规运行

如图4-4，TaskLUI的日常工作就是检测LUI\_OutQueue中的数据，监测用户输入，检测网络状态，并更新模块数据然后通过液晶屏显示出来。

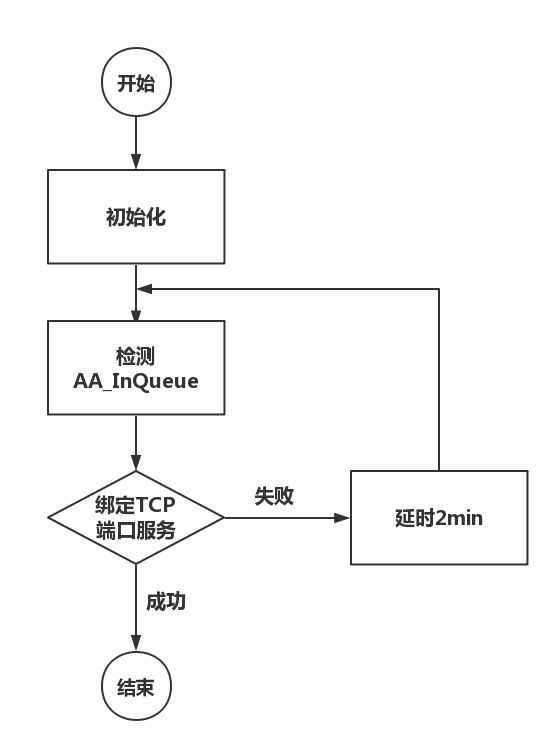


**图4-4 TaskLUI常规运行逻辑**

### 2. void TaskAppAgent(void \* para)

#### 初始化

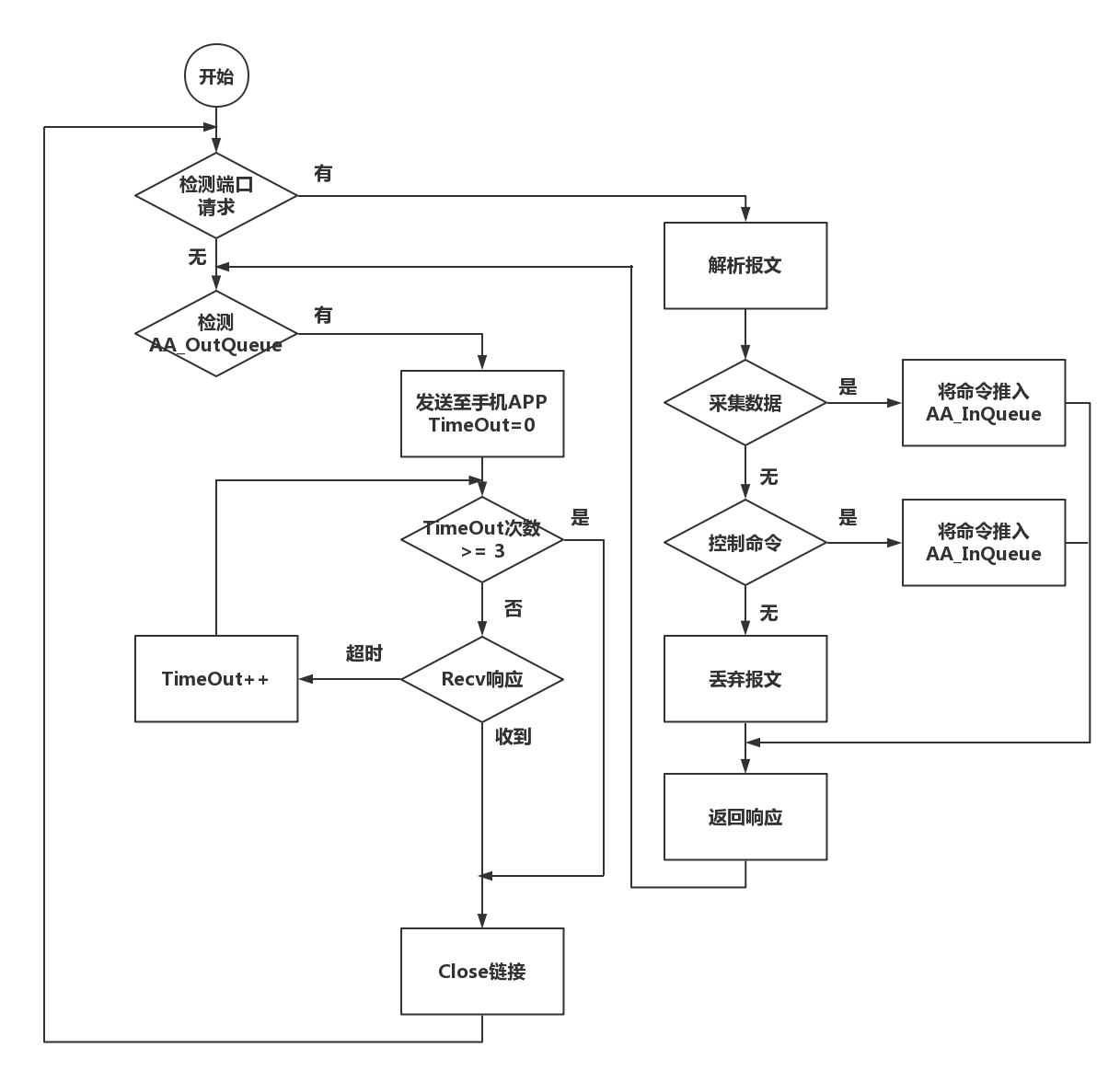
如图4-5，TaskAppAgent初始化的主要工作就是绑定1个TCP端口，用于后续接收手机App的发问请求。



**图4-5 TaskAppAgent初始化**

#### 常规运行

如图4-6，TaskAppAgent的日常工作就是检测是否有用户请求数据或者下发控制命令，并将数据或命令Enqueue到AA\_InQueue中。然后检测AA\_OutQueue，如果发现有数据，就发送到最近5个登录的手机APP。



**图4-6 TaskAppAgent常规运行**

### 3. void TaskServerAgent(void \* para)

#### 初始化

#### 常规运行

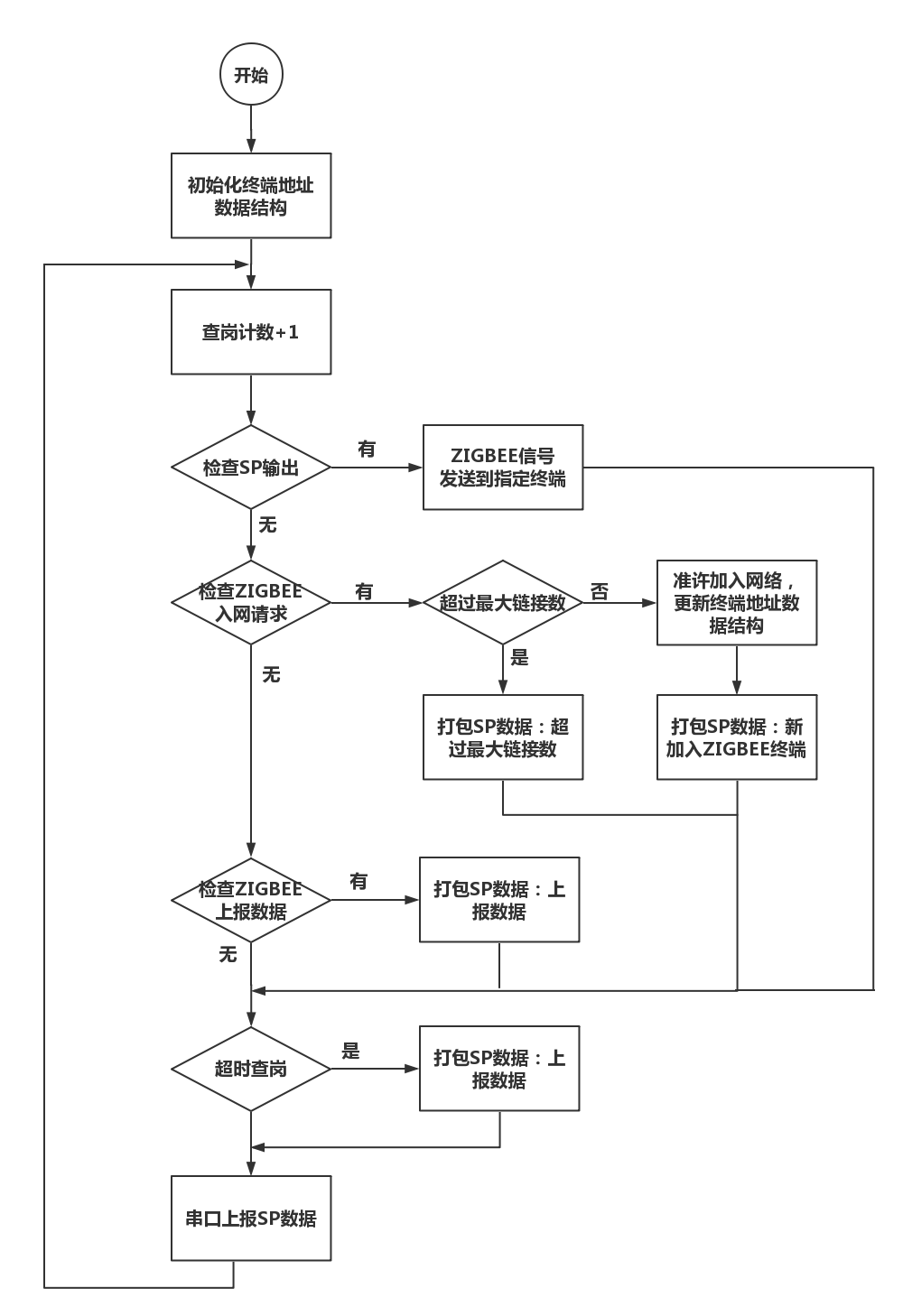
### 4. void TaskZigbeeAgent(void \* para)

#### 初始化

TaskZigbeeAgent运行在CC2530上，作为Zigbee协调者，独立于主控板运行。其初始化的工作很单纯，可以依照CC2530的示例程序设置协调者参数来初始化。

#### 常规运行

如图所示，TaskZigbeeAgent监测从DataHub传送过来的SP数据，检查新的



**图 TaskZigbeeAgent常规运行**

入网请求，ZIGBEE上报数据，并且设置查岗超时，定时查岗ZIGBEE终端。CC2530和主控芯片的通信通过SP（Beecom-S Protocol）完成。

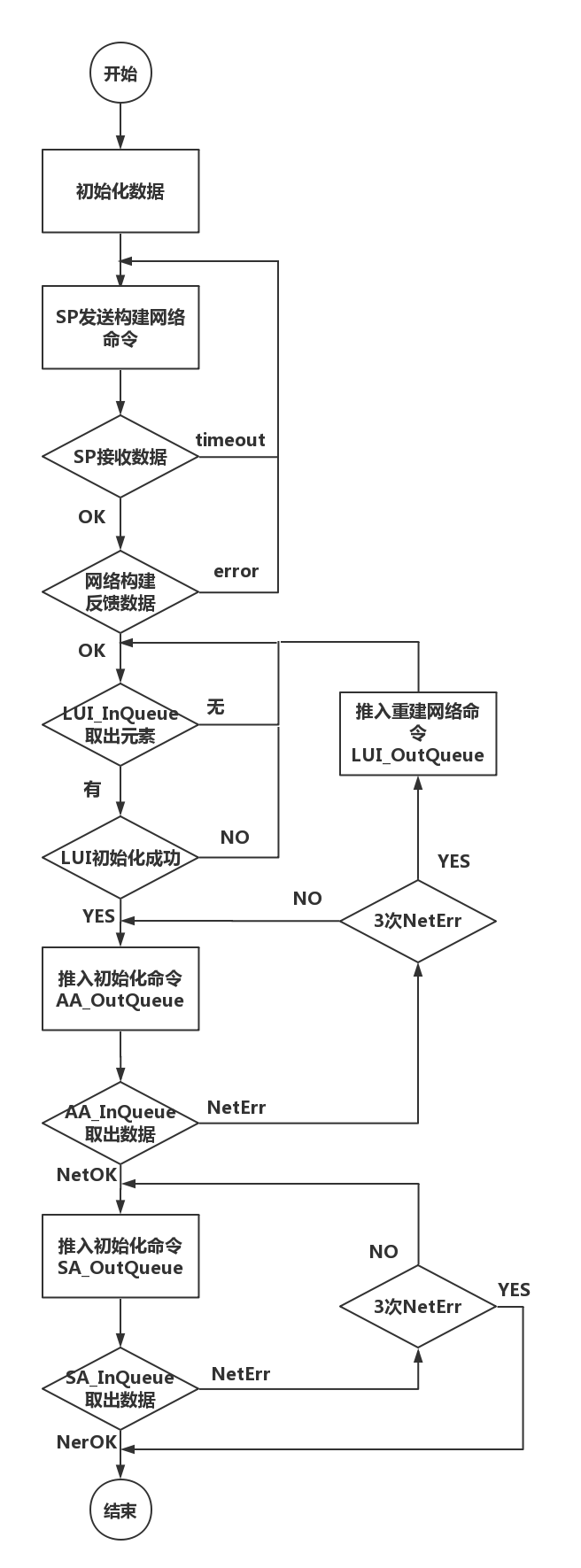
### 5. void TaskDataHub(void \* para)

#### SP

CC2530与主控芯片通信是通过串口完成的，特此定义Beecom-S Protocol来规范数据传输方式，具体内容可参见《Beecom-S Protocol》。SP提供函数库和头文件，DataHub通过调用SP的API和CC2530交互数据，然后分别对ZA\_InQueue和ZA\_OutQueue进行操作。

#### 初始化

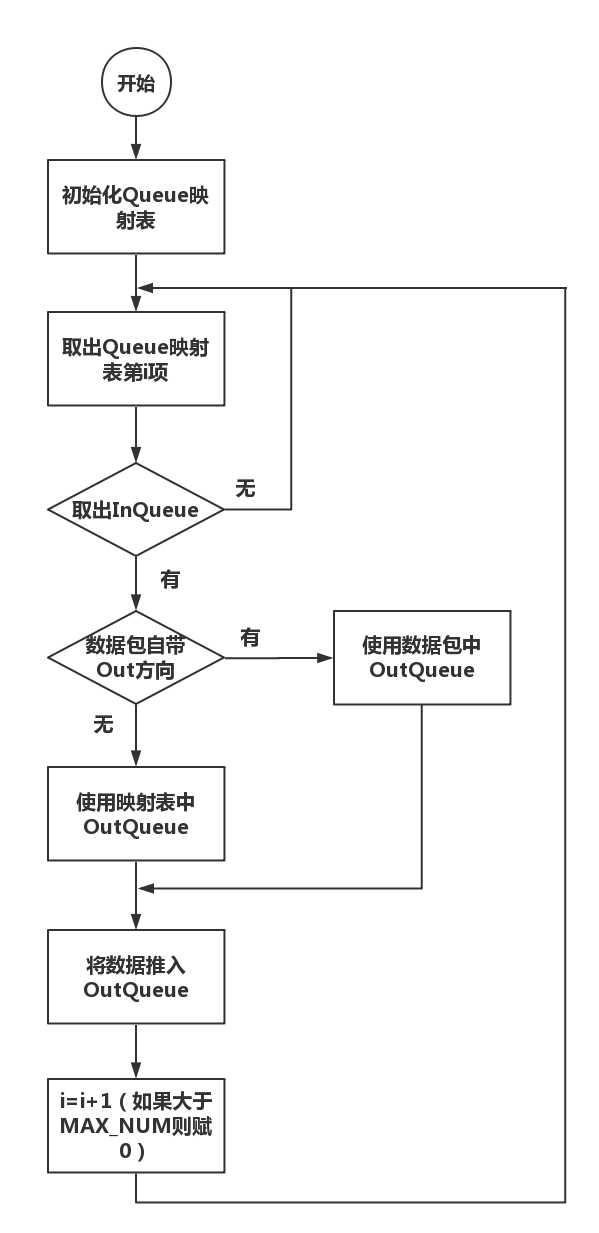
如图，TaskDataHub初始化的主要工作就是完成三个Agent的网络建立和链接，其中ServerAgent的链接不是必须的，其他两个网络必须链接成功，DataHub的初始化方能完成，否则将一直尝试去创建两个网络。



**图 TaskDataHub初始化**

#### 常规运行

如图，TaskDataHub首先建立Queue映射表，对应将InQueue的数据发送到



**图 TaskDataHub常规运行**

相应的OutQueue中。OutQueue也可以是EnQueue中的元素指定的。

## 附录：

### 1. 函数索引

#### void TaskLUIInit(lui\_init\_t \* st\_init\_para)

**所属模块：**LUI；

**调用模块：**LUI；

**参数：**

st\_Init\_para：预留。

**说明：**加载液晶屏驱动（根据操作系统的不同，也可以交由操作系统加载），加载中文字库，初始化模块变量；

#### void TaskLUIRun(net\_para\_t \* st\_net\_para)

**所属模块：**LUI；

**调用模块：**main（主函数创建任务，由系统调用）；

**参数：**

st\_net\_para：TaskLUI初始化时获得WIFI网络参数。

**说明：**检测用户输入、LUI\_OutQueue和检测维护网络状态，并刷新液晶屏提示信息。该函数一旦进入就不会退出，直至系统关闭。

#### void TaskAppAgentInit(net\_para\_t \* st\_net\_para)

**所属模块：**AppAgent；

**调用模块：**AppAgent；

**参数：**

st\_net\_para：从AA\_Inqueue中获得的WIFI网络参数。

**说明：**绑定TCP服务端口。

#### void TaskAppAgentRun(net\_para\_t \* st\_net\_para)

**所属模块：**AppAgent；

**调用模块：**main（主函数创建任务，由系统调用）；

**参数：**

st\_net\_para：从DataHub中取得的WIFI网络参数。

**说明：**检测是否有手机APP数据请求或者命令输入，检测是否有上报手机的数据。

#### void TaskZigbeeInit()

**所属模块：**ZigbeeAgent；

**调用模块：**ZigbeeAgent；

**说明：**建立Zigbee协调者网络，更新终端数据结构。

**参数：**

#### void TaskZigbeeRun()

**所属模块：**ZigbeeAgent；

**调用模块：**main（主函数创建任务，由系统调用）；

**说明：**监测下行SP数据，并通过Zigbee传输到终端；监测Zigbee信号，然后通过SP打包上行发送到DataHub；定时查岗Zigbee终端。

**参数：**

#### sint32\_t TaskDataHubInit(DataHubInit\_Type \* init\_type)

**所属模块：**DataHub；

**调用模块：**DataHub；

**参数：**

**init\_type：**初始化数据结构

**说明：**暂不做任何处理，代码预留。

#### void TaskDataHub()

**所属模块：**DataHub；

**调用模块：**main（主函数创建任务，由系统调用）；

**参数：**

**说明：**

1. 遍历Queue映射表（ModMsgMap），取出来源模块的消息（如果有的话），然后转发给目的模块；
2. 处理发送给Datahub的消息（ProcQueueElm()）；
3. 任务挂起（vTaskSuspend），当其他模块发送消息时，会再次唤醒Datahub。

#### sint32\_t BC\_Transmit2Mod(BC\_QueueElement \* que\_elm, BC\_MsgDirMap \* msg\_dir\_map)

**所属模块：**DataHub；

**调用模块：**DataHub；

**参数：**

**que\_elm：**待转发的队列元素；

**msg\_dir\_map**：消息转发映射关系表；当que\_elm中的目的模块为默认模块（BC\_MOD\_DEFAULT）时，启用该值中的目的模块作为消息转发目的模块。

**说明：**

将消息源模块的数据（que\_elm）放入目的模块的OutQueue当中。如果que\_elm中的目的模块为默认模块（BC\_MOD\_DEFAULT），则使用msg\_dir\_map中的目的模块作为目的模块，否则使用que\_elm中指定的目的模块。

**返回值：**

如果成功则返回0，否则返回相应的错误码。

### 数据结构索引

#### struct BC\_MsgDirMap

*typedef struct BC\_MsgDirMap {*

*uint8\_t u8SrcID;*

*uint8\_t u8DstID1;*

*uint8\_t u8DstID2;*

*uint8\_t u8DstID3;*

*struct BC\_MsgDirMap \* pExtDst;*

*}BC\_MsgDirMap;*

消息转发映射表。Datahub转发消息时使用，u8SrcID指明源ModuleID，目的ModuleID可以为多个。3个以内的目的ModuleID可以用u8DstID1、u8DstID2、u8DstID3指定，多于3个的目的ModuleID放在pExtDst中。

#### struct BC\_QueueElement

*typedef struct BC\_QueueElement {*

*uint8\_t u8SrcID;*

*uint8\_t u8DstID;*

*uint16\_t u16MsgLen;*

*uint8\_t pText[BC\_CONFIG\_QUEUE\_ELEMENT\_BUF\_SIZE];*

*}BC\_QueueElement;*

消息队列元素。含有该消息的源ModuleID（u8SrcID）、目的Module（u8DstID）、消息大小（u16MsgLen）和消息内容（pText）。消息最大容量为BC\_CONFIG\_QUEUE\_ELEMENT\_BUF\_SIZE。

### 枚举索引

#### enum BC\_ModID

*enum BC\_ModID {*

*BC\_MOD\_DEFAULT=0,*

*BC\_MOD\_DATAHUB,*

*BC\_MOD\_PHONE\_APP,*

*BC\_MOD\_ZIGBEE,*

*BC\_MOD\_BLUETOOTH,*

*BC\_MOD\_NET\_SERVER,*

*BC\_MOD\_TERMINAL,*

*BC\_MOD\_INVALID,*

*};*

ModuleID枚举值。用于标识模块。当一个模块要引用其他模块的数据或向其他模块发送数据，必须先引用对方模块的ModuleID。

BC\_MOD\_DEFAULT：空置模块，仅Datahub转发消息时判断转发目的模块时使用；

BC\_MOD\_INVALID：无效模块；

其他：实际模块的ModuleID。