## 管理APP程序设计说明书V4.1

易发胜 2017年5月27日

**一、功能背景**

能够登录平台和通过平台远程登录前端设备；也可以在局域网发现前端设备，直接登录前端设备。登录平台或者前端设备后，能够及时接收所关联的前端设备的报警信息。在没有界面时，收到信息点击可以启动界面查看消息详情。

后期目标：

可以聊天，收藏信息，加入联系人（好友），申请关联设备。对各种不同前端设备进行控制管理和查看视频。

系统连接示意图：

管理APP

Web服务器

前端监控设备

前端监控设备

**二、程序框架设计思路**

APP程序框架具有如下几个方面的功能架构：

1、完成数据存储访问机制（数据存储定义，访问接口）

2、完成网络服务和协议处理过程。

3、完成服务和界面通知机制。

4、完成后台消息通知和启动界面处理。

设计思想：

框架工作主要由BaseActivity和SmartService类完成，其中，BaseActivity是自定义的一个所有界面的基类，完成统一的数据访问、与SmartService通信的公共处理。从BaseActivity派生的各个具体功能界面Activity类只需要关注自己的具体业务处理即可，不需要大家都去关注公共访问接口机制，对各个开发人员的技术要求降低到最低。

具体而言，大部分与网络相关的信息由Service处理，并把结果保存在sqlite数据库中。然后通知前台从数据库中获取相关信息。部分不用写入数据库的信息由service提供的接口函数送给前台界面。前台访问数据库、文件、配置、网络等皆由BaseActivity提供相应接口实现，界面开发人员不用关心如何实现的。service有消息时，主动送到界面或者通知栏，由BaseActivity处理后，调用各界面的OnHandleMsg回调函数进行具体处理。

框架的实现原理图如下：

自带网络访问视频播放界面

前端设备插件

界面

有网络访问的界面

BaseActivity

全局数据信息

通知栏通知新消息

service

系统信息

网络访问

临时、永久数据访问对象

**三、搭建规范的源代码体系**

所有源代碼體系遵從如下目錄：

activity： contact, device, message, setting, base。所有的activity

fragment：所有的fragment

plugin：所有插件处理

service: protocol, net，jni；后臺服務處理

model：所有數據訪問處理

entity: 数据结构定义

global：全局性数据定义

utils：杂类

**四、为后续详细工作留下接口主要表现在如下方面**

1、先实现一个baseActivity类，实现所有界面都需要完成的公共功能及接口，比如通知机制、写入和读出数据等。

2、完成smartApplication，smartService，smartSqlite等服务类型，让前端界面可以不用关心后台实现逻辑。

其中：

smartApplication定义程序全局变量，主要是一些影响APP的配置，登录信息等

smartService 定义与网络有关的访问和数据

smartSqlite定义数据库访问的有关方法

**五、定义完整的数据对象**

**1、数据信息类别**

（1）保存在数据库的信息

设备的关联人——拥有管理员权限的设备，登录后从设备直接获取

我的设备表——局域网发现且登录成功的、或者从服务器获取的

我的设备附件表——记录设备有关的图片视频文件

设备的报警消息——登录设备时，从设备获取的，我的收藏

我的通告消息——账户相关, 多种类别的消息

我的聊天消息——帐号相关。

我的联系人及资料——从平台服务获取的，账户相关

我的帐号信息——账户相关

（2）简单键值对信息

APP配置信息——与账户无关

（3）文件信息

设备插件文件，与数据库的设备表关联。

设备视频照片，与数据库的设备附件表关联。

**2、数据库Sqlite表设计（用于存储离线信息）**

（1）用户帐号表Account（用于保存本手机上多个用户的帐号密码，注册信息不保存）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 用户名:username | varchar(20) | 否 | 登录成功后保存，唯一 |
| 2 | 密码:password | varchar(20) |  | 登录成功后保存 |
| 3 | 用户ID: id | interger |  | 一般固定 |
| 5 | 性别：sex | char(1) |  | 0女，1男 |
| 6 | 注册时间：time | timestamp |  |  |
| 7 | 用户类型：type | char(1) |  | 0一般用户1物管2警察 |
| 8 | 注册邮箱：email | varchar(20) |  |  |
| 4 | 最近登录平台状态：state | char(1) |  | 1登录，2认证，3在线，0离线，-1强制离线 |
| 9 | 最近登录设备id | interger |  |  |

可能占用空间：80x4=320B

（2）用户设备表信息记录表Device

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 用户id | interger | 否 | 关联帐号表 |
| 2 | 设备名称:devname | varchar(20) | 否 |  |
| 3 | 设备序列号：serial | varchar(30) |  | 唯一 |
| 4 | 设备条形码：code | varchar(16) |  |  |
| 5 | 设备IP地址：ipaddr | interger |  |  |
| 6 | 设备别名：alias | varchar(20) |  |  |
| 7 | 登录设备用户名：user | varchar(20) |  | 可与用户名一样 |
| 8 | 登录设备密码：pass | varchar(20) |  |  |
| 9 | 设备权限：power | char(1) |  | 0管理权限，1操控权限，2查看权限，3接收权限 |
| 10 | 设备类型：type | char(1) |  | 0前端设备1监控APP |
| 11 | 插件：plugin | varchar(20) |  |  |
| 12 | 设备id：id | interger |  |  |
| 13 | 软件版本：version | varchar(10) |  |  |
| 14 | 最近登录时间：time | timestamp |  | 最近登录设备时间 |
| 15 | 经度：longitude | float |  |  |
| 16 | 纬度：latitude | float |  |  |
| 17 | 设备状态：state | char(1) |  | 0离线，1登录平台 |

可能大小：200x10=2KB

（3）设备附属文件表document（保存某个设备的视频、图片文件）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 设备id | interger | 否 |  |
| 2 | 设备序列号:serial | varchar(30) | 否 | 关联设备 |
| 3 | 附件类型:sort | char(1) | 否 | 0视频、1图片 |
| 4 | 产生时间：time | timestamp | 否 |  |
| 5 | 附属文件：url | varchar（20） |  | 保存在本地的文件名 |

可能大小80x100=8KB

（4）我的联系人信息表contact（保存我的联系人）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 用户名称:username | varchar(20) | 否 |  |
| 2 | 用户id | interger | 否 |  |
| 3 | 联系人用名户:contact | varchar(20) | 否 |  |
| 4 | 用户类型: type | char(1) | 否 | 0一般1物管2警察 |
| 5 | 性别：sex | char(1) | 否 | 0女1男 |
| 6 | 别名：alias | varchar(20) |  |  |
| 7 | 是否在线：online | char(1) |  | 0离线，1在线 |
| 8 | 添加时间：settime | date |  |  |
| 9 | 所属组别：group | varchar(10) |  |  |

80X100=8KB

（5）用户设置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 用户名称:username | varchar(20) | 否 | 仅记录用户修改的记录 |
| 2 | 用户id | interger | 否 |  |
| 3 | 新消息提醒声音:sound | char(1) |  |  |
| 4 | 新消息提示震动 | char(1) |  |  |
| 5 | 通知时指示灯闪烁 | char(1) |  |  |
| 6 | 不提示聊天信息 | char(1) |  |  |
| 7 | 不提示公告信息 | char(1) |  |  |
| 8 | 添加联系人需要验证 | char(1) |  |  |
| 9 | 制定设备关联需要验证 | char(1) |  |  |
| 10 | 允许别人通过帐号找到我 | char(1) |  |  |
| 11 | 允许接收陌生人消息 | char(1) |  |  |

30B

==============================================================================

（6）报警信息表alarm（保存某个设备的报警消息）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 消息编号 | int | 否 | 从1开始，一直递增 |
| 2 | 设备id | interger | 否 |  |
| 3 | 消息类型: type | char(1) | 否 |  |
| 4 | 消息级别：level | char(1) | 否 |  |
| 5 | 产生时间：time | timestamp | 否 |  |
| 6 | 消息附件：url | varchar（20） |  |  |
| 7 | 消息附件本地lurl | varchar（20） |  | 下载后存在 |
| 8 | 消息状态：state | char(1) |  | 0未查看，1已查看，2拒绝4收藏（按位可加） |

可能大小80X1000=80KB

（7）我的聊天消息表chat（保存我收到的聊天消息）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 消息编号 | int | 否 | 从1开始的编号 |
| 3 | 用户id | interger | 否 |  |
| 4 | 对方id | interger | 否 |  |
| 5 | 聊天类型: type | char(1) | 否 | 0发出、1接收 |
| 6 | 产生时间：time | timestamp | 否 |  |
| 7 | 消息类容：content | varchar(100) | 否 |  |
| 8 | 字型：type | int |  |  |
| 9 | 颜色：color | int |  |  |
| 10 | 状态：state | char(1) |  | 0未确认，1已查看，2拒绝4收藏（按位可加） |

150X1000=150KB

（8）我收到的通知信息表notice（保存我收到的通知消息）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 1 | 消息编号 | int | 否 | 从1开始编号 |
| 2 | 用户id | interger | 否 |  |
| 3 | 通知类型：sort |  |  | 0联系通知、1关联通知 |
| 4 | 通知类别: kind | char(1) | 否 | 0请求确认；1确认；2、否认；3 解除通知；4 请求通知（不需要用户确认） |
| 5 | 产生时间：time | timestamp | 否 |  |
| 6 | 留言内容：message | varchar(100) |  |  |
| 7 | 状态：state | char(1) |  | 0未确认，1已查看，2拒绝查看 4收藏（按位可加） |
| 8 | 请求者帐号：request | varchar(20) |  |  |
| 9 | **请求权限：power** | char(1) |  |  |
| 10 | **设备序列号:serial** | varchar(20) | 否 | 关联设备 |
| 11 | **关联密码：pass** | varchar(20) |  |  |

230X100=23KB

（9）我收到的系统通告info（保存我收到的通知消息）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 字段名 | 值类型 | 是否为空 | 备注 |
| 0 | 消息编号 | int | 否 | 从1开始编号 |
| 1 | 用户id | interger | 否 |  |
| 2 | 消息提供者id | int | 否 |  |
| 3 | 通知附件：url | varchar(20) |  | 可能的图片和视频文件 |
|  | 通知附件：lurl | varchar(20) |  | 本地文件 |
| 4 | 产生时间：time | timestamp | 否 |  |
| 5 | 消息内容：message | varchar(100) | 否 |  |
| 6 | 消息标题：title | varchar(50) |  |  |
| 7 | 字符集：set | varchar(10) |  |  |
| 8 | 状态：state | char(1) |  | 0未确认，1已查看，2拒绝查看 4收藏（按位可加） |

前面共：200KB

**3、键值对设计（一般配置，与用户无关）**

帮助文档 help string（文件名）

功能介绍 intro string（文件名）

服务条款 item string（文件名）

自动更新插件plugin bool

自动更新程序 program bool

自动登录 login bool

允许保存登录密码pass bool

非wifi下载报警附件dattach bool

非wifi下自动下载插件dplugin bool

非wifi下自动下载升级 dprogram bool

非wifi下实时视频 dlive bool

wifi下自动发现新设备 discover bool

最大收藏记录数 collect int

占用存储空间限制 limit int(KB)

最近登录用户 last string

更新的程序 url

最新版本 version

系统信息

最近休眠时间sleeptime timestamp

最近启动时间 starttime timestamp

估计1KB

**4、文件定义**

插件，设备图片视频，报警消息图片

（按照时间命名将文件名写入数据库，便于查询）

估计100MB

注意：

* 软件logo、软件的版本号和软件的id定义在程序中。

**六、页面框架设计**

|  |  |  |
| --- | --- | --- |
| activity | 管理界面 | 说明 |
| 登录注册管理 | 登录界面，忘记密码，注册界面 | 与service进行通信 |
| 具体设备管理 | 设备详情及处理界面 | 部分不需要和service通信 |
| 一般设备管理 | 其他设备发现及登录界面 |  |
| 联系人管理 | 有关联系人管理界面 |  |
| 消息管理 | 消息详情相关界面 | 不需要和service通信 |
| 聊天界面 | 聊天界面 |  |
| 插件界面 | 插件控制界面 |  |
| 视频播放 | 视频播放界面 | 不需要和service通信 |
| 设置界面 | 设置有关界面 | 大部分没有网络通信 |
| 主界面 | 主界面 | 有多个fragment构成标签 |

**activity框架：与service通信，管理数据交互，广播及activity之间传数据。**

**登录与注册：**

1. 登录，注册界面 -->一个 activity
2. 忘记密码处理界面

**设备（device）**

1. 设备主界面 -->一个 activity
2. 设备登录登出界面
3. 设备搜索界面
4. 设备详情界面，设备设置，设备分享 -->一个 activity
5. 设备控制管理
6. 设备插件更新 -->一个 activity
7. 给关联人发送消息 -->一个 activity
8. 设备视频 -->一个 activity

**联系人（contact）**

1. 联系人主界面，设备关联人，聊天界面 -->一个 activity
2. 组管理
3. 新联系人 -->一个 activity
4. 联系人详情，聊天设置 -->一个 activity
5. 发现新联系人，添加联系人 -->一个 activity

**消息（message）**

1. 消息主界面，消息详情 -->一个 activity

**设置（setting）**

1. 设置主界面，个人资料，我的收藏，关于 -->一个 activity
2. 账号管理，消息管理，流量管理，记录管理，隐私管理，插件管理 -->一个 activity

**按照页面结构分配，程序一共可能使用13个activity，根据实际情况可能调整。**

**各个页面数据使用注意事项：**

* 登录平台时，从平台得到最新信息，然后插入数据库；
* 离线时，从本地数据库获得信息。一旦登录成功，从平台获得信息，并对比，如变化用数据库信息刷新并更新。
* 各个页面都要认真分析数据来源、是否在线离线等。

**七、有关数据的各种处理接口设计**

1、数据库创建处理

2、添加各表信息

3、删除各表记录信息

4、查询各表记录信息

5、修改键值信息

6、查询键值信息

7、查询文件路径

**八、优先完成如下页面的全部**

登录、注册页面

设备、消息、联系人、设置主页面

局域网发现和关联设备页面

设备详情页面

控制设备页面

播放设备页面

设置页面

登录设备、登出设备页面

九、全局信息

全局信息保存在APPlication中，主要为各种状态信息。具体包括：

平台用户名

平台登录密码

平台登录状态

平台登录时间

当前联网方式

登录设备ID

登录设备状态

登录设备方式

登录设备时间

**九、Service内部数据结构定义**

1、和平台通信信息（登录平台的时候使用该对象）。

interger update; //最近检查更新的时间

interger recvseq; //最近收到的平台报文序号

interger recvtime; //最近收到的平台报文时间

interger sendtime; //最近发送给平台报文时间

interger sendseq; //最近发送给平台报文序号

interger waitnum; //等待重发报文数量

class sendpkt\_node \*waitlist; //等待重发报文列表

2、和设备通信信息：（登录设备的时候使用该对象）

interger ipaddr; //登录IP地址，如果等于服务器ip，表示转发过来的。

smallint port; //登录端口

int recvseq; //最近收到设备报文序号

int recvtime; //最近收到设备报文时间

int sendseq; //最近发送设备报文序号

int sendtime; //最近发送设备报文时间

int waitnum; //等待重发报文数量

struct sendpkt\_node \*waitlist; //等带重发报文列表

3、后处理任务信息（协议处理完后需要善后处理的任务。如有界面等待协议处理结果的通知）

char face //界面代号activity

char cmd //等待应答的命令

char type //等待应答的报文类型

char sort //等待应答的报文来源

int time //挂起任务时间

int id //等到应答的对方id

4、延迟处理任务信息（一个协议处理完后，需要自动进行的下一个处理任务。一般是下载任务或者登陆后查询未读消息等处理）

char code //任务代码：1、下载文件；2、查询未读消息；

char file[20]

peer id

char md5[16] //下载文件的md5

int num; //文件大小或者查询数据的代码字

5、正在下载的文件信息

文件打开对象（指针）

文件名url

文件总大小

文件md5

最近访问时间

文件打开时间

文件已读大小（为0表示还未读、被下载-1、已下载>0）

**十、service模块结构设计**

1、主要功能说明

* 程序启动后，常驻后台运行（需要进一步明确service的运行机制）的程序
* 收到前端设备或者平台的消息，主动推送给相应的activity或者向通知栏发消息。
* 维持网络通信，与平台服务器和前端设备的信息交互。
* 接收activity访问要求，完成网络访问任务或者状态变量查询。

2、模块结构

定时器管理模块

服务接口

通知接口

专有数据

网络通信模块

activity访问接口

说明：

（1）activity访问接口：提供让activity访问service的信息的处理。在baseactivity获得service对象的实例，然后调用service的接口来获取service的信息或者给service发送信息。一般采用异步模式，即处理请求后产生一个后处理任务，马上返回。service得到结果后再执行后处理任务，将结果通告请求的activity。

（2）服务接口：service需要的各种标准接口，比如创建、销毁等。主要执行服务的初始化、关闭等操作。

（3）通知接口：service在定时器管理模块或者网络通信模块发生定义的事件后，需要通知用户或者某activity时提供的接口。如果有activity存在，则直接送消息给activity，否则发通知给消息栏，用户点击通知时再启动相关的activity。发送信息给activity时，采用广播机制，将消息分类为消息通知、执行结果、状态变化、更新通知等类别。采用ACTION机制与界面绑定，结合规范定义的消息类别进行处理。

（4）专有数据：存储上述的各种过程数据及数据处理方法。

（5）定时器管理模块：执行各种定时执行的工作，比如定时检查通信超时并保活，定时执行延迟处理任务，定时自动登录、更新检查，定时检查网络状态变化等。用线程实现。

（6）网络通信模块：与平台服务器和前端设备进行通信的模块，最复杂，需要按照协议规定要求进行处理。包括一个网络接收线程及各种协议处理对象。

远程控制协议

联系人协议

局域网发现协议

平台管理协议

网络接收服务线程netcom

加密协议

网络发送接口netcom

升级查询协议

文件传输协议

关联协议

网络访问接口（nethandler, timerproc等）

连接管理协议

消息传递协议

数据查询协议

注：黑底部分优先完成。

3、有关注意事项

（1）发起一个网络访问的来源，有界面、定时器或者接收报文。如果是界面发起的处理，需要填写后处理任务，待协议处理结束后，执行后处理任务去通知界面协议执行结果。

（2）一些协议执行后，可能还要执行后续其他协议，如升级查询协议执行了，如果有新升级，则需要启动文件传输协议下载新版本程序，下载之后还需要自动安装等，就需要登记延迟处理任务。延迟处理任务由定时器根据用户设置和当前状况判断是否执行。

（3）一些需要自动不断执行的工作，比如没有登录平台时的APP自动登录。都定义为定时器任务，由定时器驱动。

（4）本部分的主要工作在网络通信模块和定时器处理模块，其他都是辅助性的，与系统或者其他部分进行通信交互的模块，属于全局框架性内容。

（5）本模块需要访问配置和数据库等获得有关配置和用户信息，使用定义的数据访问对象获得，框架图上并没有表示。

4、模块对象或者接口设计

（1）smartservice对象：派生于service，自定义的服务对象。

（2）srvinfo对象：定义全局数据对象，具体信息见《**Service内部数据结构定义》**

（3）timer对象：包含一个定时器及其需要定时处理的方法。

（4）netcom对象：网络通信对象，具有基本网络数据收发和线程。包含构造函数初始化线程和网络套接字处理；设置函数，设置回调接口函数；发送接口函数，负责发送信息等。析构函数负责释放资源。

（5）nethandler接口：网络信息协议处理，包括接收处理接口函数和发送处理函数，接收时，netcom回调使用；发送时，由sendmsg调用。

（6）timerproc接口：让timer进行回调使用，也是协议处理的一部分。主要操作srvinfo对象，判断是否有超时事件interface

（7）notify接口：给activity或者通知栏发送通知的接口。由timerproc和nethandler使用。

（8）sendmsg接口：activity发送信息给service接口。如果需要网络访问，则调用nethandler的发送接口，进行协议处理后，进行信息发送。

（9）getmsg接口：activity从service这里获取信息的接口。

要处理。

（10）protocol对象：协议处理基本对象。包括发送和接收接口等构成。由nethandler内部使用本对象。

（11）protoxxx对象：由protocol派生具体协议处理对象，由nethandler使用。

（12）packet对象：报文对象，由protocol对象内部使用。

定义myservice类大体如下：

public class MyService extends Service {

private srvinfo gdata;

private netcom net;

private timer srvtimer;

public void onCreate()

public void onDestroy()

private int notify( );

public sendmsg( );

public getmsg( );

public interface nethandler;

public interface timerproc;}

**十一、插件化控制实现方法**

1、插件与设备的关联方法

每类设备有一个条形码，一类设备都有相同的插件。所以插件里面有一个元素就是其条形码。通过设备条形码和插件中内置的条形码比对，可以防止使用错误的条形码。前端设备也需要检查插件内置的条形码是否与设备条形码一致。在APP使用插件时，也需要比较设备的条形码和插件内置的条形码是否一致。

2、插件版本与支持的元素

每个插件都有版本号，就用一个自然数表示，从1开始不断增加。

插件文件目前支持如下元素：

文件标识、按钮、文本框、静态文字

3、插件中各个元素的属性

文件标识：设备条形码，版本号

按钮：id、位置、大小、文字、功能码、关联类型(数据，id)、关联数、关联运算类型

文本框：id、位置、大小、文字、功能码

静态文字：id，位置、大小、提示文字

定时器：指令码，时间间隔

4、插件文件的命名与升级查询

插件文件用设备条形码命名，利于APP端名字不冲突。

在APP端和前端设备端，对现有的插件文件都需要写一条记录在数据库（或者配置文件中），记录其文件名和版本号。（在上传下载的时候，要从插件中的文件标识元素得到其版本属性进行核对）如果不对则提示出错。但是升级查询时，仅仅用内存或者数据库中保存的信息。

5、插件的工作过程

APP每次登陆设备时；如果还没有此设备的控制插件，则查询插件文件信息，下载之；如果已经存在此设备的插件文件，则查询是否有更新，有更新则重新下载之。否则打开插件文件，根据插件文件的要求及时生成控制界面。生成界面前，获取插件的序列号属性与设备序列号比较，如果不一致则报错。用户操控的结果通过service发到前端设备。前端设备返回的信息由界面解析后在相应的文本框中显示。

用户切换界面时，关闭插件界面。

插件文件的文件标识元素属性随时可能需要读出，便于比较、查询和核对。

6、插件界面的生成

打开插件文件，解析xml，得到各个元素的值。其中文件标识用于核对信息。依次读出其他每个元素，根据其位置大小构造界面。根据其他属性构造响应处理。

7、插件与APP的接口及处理机制

插件界面独立使用一个activity，整个插件界面在一个view或者fragment中。所有需要与前端设备交互的信息，由view通过回调告知activity，然后再通知到service中。对于收到的要显示的信息，由activity调用view的成员函数即可。与service的通信由activity保证。

**十二、activity设计**

1、完善界面并分析哪些界面属于一个activity

理解管理app功能，弄清各个界面的操作功能目标；

完善所有的界面，补充还没有画出的界面，弄清每个界面功能和要展示的信息，要做的操作；

画图表达各个界面的跳转关系；

（完善以前的界面设计说明书）

将叶节点界面及其根节点界面设计为一个activity。

（根据上面的图填写如表1：）

表1 xx界面类别设计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | activity符号名称 | 作用 | 包含界面名称 | 界面作用 | 界面设计特点 |
| 1 | loginActivity | 登录处理 | loginView  registerView | 显示登录界面  注册界面 | 显示进度 |
|  |  |  |  |  |  |

2、分析各个activity及其要显示的界面信息

分析各个界面需要的信息，哪些从存储空间获取，哪些需要送到service，哪些需要从service获取。（填写表2）

在分析信息的基础上，定义信息基类和各个信息类，确定所有信息的表示方法；

分析每个页面在各种情况下的信息来源和处理方法。

3、编写派生类baseactivity，在其中实现：

启动服务，绑定服务，调用服务方法给服务传信息或者从服务获得信息；

注册一个广播接收器，接收来自服务的信息并解析；

构造数据访问对象，实现对所有数据访问的方法，初始化、读、写；针对不同信息访问要求给出访问数据信息方法。

实现一个方法，启动另外一个activity并传输信息给另一个activity；

4、利用baseactivity派生各个具体的activity，实现界面逻辑。

表2 xx界面信息设计

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| activity名称 | 界面名称 | 获得信息 | 信息来源 | 产生信息 | 信息去向 | 启动下一个activity |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**十三、Service中进行网络通信处理过程**

1、分析网络需要回调的事件有哪些，便于定义MSGObject的具体类型有哪些。

2、定义回调接口

public interface NetCallback

{

public void NetNotify(MSGObject); //系统通知处理

public void NetSend(DatagramPacket); //网络发送处理

}

3、在Service implement NetCallback 定义netnotify的实现，对收到的网络信息进行具体处理，比如发通知到通知栏等

｛

DatagramSocket sock;

ProtoProc mProtoProc; //进行具体的协议处理

public void NetNotify(MSGObject); //实现回调函数的处理

public void NetSend(DatagramPacket); //基本的网络发送信息处理

public void SendMessage(MSGObject msgObject) //调用mprotoproc进行协议发送处理

在OnCreate中打开套接字sock定义线程，在线程中调用mProtoProc进行接收信息处理

在OnCreate中打开定时器，定时检查SmartData中的数据，根据协议调用mprotoproc进行协议发送处理或者通知操作。

}

4、定义网络协议处理类

public class ProtoProc

{

MyProtocol mProtocol[n] 根据协议个数，定义具体的协议对象

public SendNetMsg(MSGObject）要求发送网络信息的处理

public RecvNetMsg(DatagramPacket) 接收网络信息的处理

｝

5、定义具体协议处理基类

public class MyProtocol

{

NetCallback mCallback; //回调函数

SmartData mData; //协议处理相关数据，在service中定义，这里引用

PacketData mPacket；//报文分析的中间结果

DatagramPacket mpkt; //正处理的报文

private int ParseHead()

private int EncapHead()

MyProtocol(SmartData) //构造函数的时候设置数据引用

public SetCallback() //设置回调函数

//需要重载的函数

public int sendproc(MSGObject)

public int recvproc(DatagramPacket)

｝

6、定义具体协议处理类

public class LoginProtocol extend MyProtocol

{

//定义协议特有数据信息

//需要重载的函数

public int sendproc(MSGObject)

public int recvproc(DatagramPacket)

}

典型处理过程：

1、发送处理过程

activity调用service的SendMessage(MSGObject msgObject)接口

在SendMessage(MSGObject msgObject)中发现要进行网络发送处理，该函数调用mprotoproc的SendNetMsg(MSGObject）接口函数分析应该采用什么协议进行发送处理。

mprotoproc调用m协议处理对象mprotocol[m]的sendproc(MSGObject)进行处理。

在mprotocol[m]的sendproc(MSGObject)根据协议进行封装，期间可能访问SmartData。

最后调用mcallback的NetSend(DatagramPacket)接口发送到网络上。

2、接收处理过程

线程中收到一个DatagramPacket

然后调用mProtoProc的RecvNetMsg(DatagramPacket)函数接口进行判断是哪个协议的。

然后调用mprotoproc的m协议处理对象mprotocol[m]的recvproc(DatagramPacket)进行处理。

在mprotocol[m]的 recvproc(DatagramPacket)根据协议进行解析处理，期间可能访问SmartData

最后调用mcallback的NetNotify(MSGObject)接口在Service中进行通知处理，也可能发送通知到通知栏或者activity中。

**十四、登录注册界面设计**

1、Activity：

（1）登录界面：包含用户名、密码输入，状态信息显示。有一个登录按钮，不登录进入等两个按钮。有忘记密码和新用户注册两个链接。登录的用户名可以列表选择。密码根据配置可以显示点或者为空。

（2）忘记密码界面：由二个子页面构成

1）输入手机号并验证

2）显示登录用户名并重置密码页面

（3）注册页面：由四个子页面构成，上面有一个进度显示。

1）输入手机号，验证序列码

2）输入注册信息并检查

3）设置登录用户名和密码

4）显示成功信息，有登录按钮

2、Fragment

1）输入手机号并验证

2）设置登录用户名和设置密码

各个界面之间的关系

主页

忘记密码

用户名密码

手机号验证

用户使用条款

登录

（不登录，新用户注册，忘记密码，用户使用条款）

注册

注册结果

用户名密码

填写资料

手机号验证

附录一：数据库操作信息表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 对象 | 操作 | 接口函数 | 数据库表 | 说明 |
| 1 | int type，MsgObject msg | 添加 | addData | alarm  chat | 添加消息；type消息类型，msg消息体 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 1 | int type，int no，byte state | 修改 | modData |  | 修改消息状态；type消息类别；no 消息编号；state状态 |
| 2 | int sort，int id，String lurl， | 修改 | modData |  | type文件类别，id 设备id或者消息编号；lurl本地文件路径 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 1 | int type，int no | 删除 | delData |  | type:删除某类表信息0表示所有表；  no:表示消息编号，0表示全部消息； |