

咕喃 Windows 客户端开发 SDK

1. 系统说明.....	2
2. 本系统引用的类库及 nuget 包出处及说明:.....	3
3. 注册、登录、登出.....	4
4. 好友列表.....	4
5. 加好友.....	5
6. 删好友.....	6
7. 发消息.....	6
8. 收消息.....	7
9. 群组列表.....	7
10. 新建群组.....	8
11. 加入群组.....	8
12. 删除群组.....	9
13. 收取历史消息.....	9
14. 邀请好友.....	9
15. 踢出成员.....	10
16. 禁言.....	10
17. 消息免打扰.....	11
18. 拉入黑名单.....	11
19. 类详细介绍.....	11
20. 聊天功能相关类详解.....	12
21. 语音和视频聊天.....	22
22. 安全机制.....	29
23. 多设备登录.....	30

1. 系统说明

本系统采用 ProtoBuf 协议实现用户与用户之间消息的发送和接收, 采用 http 实现了业务的定制如登陆, 注册等。

本地数据库 1: constant.db 数据库: 包含的表有

tb_areas 保存地区数据

emoji 保存每个语言版本的 emoji 表情 key

本地数据库 2: 用户 ID.db, 此数据库由 SqlSugar 框架生成, 数据库名称为用户登陆的 userid 数据库包含的表有:

Friend 好友|群组表, - 用于存储好友和群组

Msg_xxxx(friendid) 消息表- 用于存储和某个好友或群组的消息记录

RoomMember 群成员表- 用于存储某个群组中所有的群成员

VerifyingFriend 新朋友验证表-用于存储最近新的验证好友

其中:

消息表表名为 msg_用户 ID 或群组 Jid, 利用类库动态生成

各表对应以下实体类及数据库的增删改查:

0. 操作数据库配置及初始化: WinFrmTalk.DBSetting

- 1. 操作朋友数据的实体类: WinFrmTalk.Model.Friend
- 2. 操作新朋友数据的实体类: WinFrmTalk.Model.VerifyingFriend
- 3. 操作群组的实体类: WinFrmTalk.Model.Friend
- 4. 操作群成员的实体类: WinFrmTalk.Model.RoomMember

-- 5. 操作聊天记录实体类 : WinFrmTalk.Model.Messageobject

2. 本系统引用的类库及 nuget 包出处及说明:

引用库	下载地址	说明
AForge.dll	http://www.aforgenet.com/framework/downloads.html	录像拍照
AForge.Controls		
AForge.Video		
AForge.Video.DirectShow		
AForge.Video.FFMPEG		
agsXMPP	https://www.ag-software.net/agsxmpp-sdk/	xmpp核心库
AxInterop.WMPLib	https://download.csdn.net/download/wzt_1234/8778133	视频播放器库 (弃用)
CefSharp	https://github.com/cefsharp/cef-binary	浏览器
CefSharp.Core	https://github.com/cefsharp/cef-binary	浏览器
CefSharp.WinForms	https://github.com/cefsharp/cef-binary	浏览器
CommonServiceLocator	https://www.nuget.org/packages/CommonServiceLocator/	
CSkin	http://www.cskin.net/	第三方控件以及窗体
log4net	http://logging.apache.org/log4net/	日志框
Newtonsoft.Json	https://www.newtonsoft.com/json	json解析库
PresentationCore	https://github.com/pgrho/phash	
PresentationFramework	https://github.com/pgrho/phash	
RestSharp	https://github.com/restsharp/RestSharp	
SpeechLib	https://archive.codeplex.com/?p=speechlib	微软语音库
SqlSugar	http://www.codeisbug.com/	数据库框架
TalkBubble	https://www.probrewer.com/bubble-talk/	第三方控件库
ThoughtWorks.QRCode	https://www.nuget.org/packages/ThoughtWorks.QRCode/	生成二维码库
Vlc.DotNet.Core	https://github.com/ZeBobo5/Vlc.DotNet	视频播放器库
Vlc.DotNet.Core.Interop	https://github.com/ZeBobo6/Vlc.DotNet	视频播放器库
Vlc.DotNet.Forms	https://github.com/ZeBobo7/Vlc.DotNet	视频播放器库
WMPLib	https://download.csdn.net/download/wzt_1234/8778133	视频播放器库 (弃用)
lua文件夹	https://github.com/ZeBobo5/Vlc.DotNet	视频播放器文件夹
plugins文件夹	https://github.com/ZeBobo6/Vlc.DotNet	视频播放器文件夹
include文件夹	https://github.com/ZeBobo7/Vlc.DotNet	视频播放器文件夹

备注：以下红色字体为 HTTP 接口名称。

3. 注册、登录、登出

<1. 注册(url: `http://imapi.shiku.co + "user/register";`)

注册新用户调: `user/register` 接口,服务端会在接口数据库 `mongoDB` 和 `Tigase` 数据库中分别产生一条记录。

注册流程: `View.FrmLogin=>View.FrmRegister`

<2. 登录(url: `http://imapi.shiku.co + "user/login";`)

普通登录调: `View.FrmLogin.btnLogin_Click` 方法

<3. 登出(url: `Applicate.URLDATA.data.apiUrl + "user/logout";`)

登出调用: `base.OnClosing(e);` 方法;

<4. 记住密码

本地存储用户的 id 和密码, 见方法 `View.FrmLogin.RememberUser()`

<5. 某个用户最后一次退出时间可使用

`LocalDataUtils.GetStringData(Applicate.QUIT_TIME);` 方法取出;

4. 好友列表

服务器数据库好友列表: (url: `Applicate.URLDATA.data.apiUrl + "friends /list";`)

服务器数据库黑名单列表: (url: `Applicate.URLDATA.data.apiUrl + "friends /blacklist";`)

好友与黑名单保存在本地(表: `Friend`), 通过 `status` 字段区分 (-1: 黑名单, 0: 陌生人, 2: 好友)

登录后, 会根据账号是否更换过设备, 来同步好友列表, 群组列表, 好友标签

详情见：

`SyncDataDownlad.DownFriend()`;

`SyncDataDownlad.DownRoom()`;

`SyncDataDownlad.DownLable()`;

5. 加好友

该操作通过 XMPP 协议进行，根据好友隐私设置，是否需要验证，不需要的话调用：`ShiKuManager.SendBecomeFriendMsg()` 直接添加为好友，需要的话，会调用 `ShiKuManager .SendHelloFriendMsg` 进行对好友验证请求通知，流程如下：

加好友的 HTTP 接口

`APIHelper.addFriend`

返回值为 2 或 4，无需验证，发“508 新朋友消息”：`ShiKuManager.SendFriRequest(好友 ID, true)`,

收到回执，新增 Friend 表记录；

打招呼

返回值不为 2 或 4，需验证，发“500 打招呼消息”：`ShiKuManager.SendFriRequest(好友 ID, false)`,

收到好友的打招呼回话

(以此类推，直至对方同意)

说明：

`ShiKuManager.SendFriRequest(好友 ID, true)`会把关系保存到服务器，并且会将数据更新保存到本地数据库；当接收到打招呼/同意好友请求等信息，会更新数

据库并刷新列表见方法 `USEUserVerifyPage.ProcessVerifyMsg()`;

6. 删好友

删好友操作见: `FriendListLayout.DeleteFriendItem()`; 方法

说明:

1. 使用接口(url: `Applicate.URLDATA.data.apiUrl + "friends /delete";`) 更新服务器好友关系, 成功后使用 `ShiKuManager.SendDelFriendMsg(friend)`; 通知好友更新关系

7. 发消息

发送消息实现主要在 `ShiKuManger` 中实现

发送消息会通过消息类型调用不同的方法

`ShiKuManger.SendTextMessage` 文本消息

`ShiKuManger.SendGifMessage` gif 图片

`ShiKuManger.SendCardMessage` 语音

`ShiKuManger.SendImageMessage` 图片

`ShiKuManger.SendVideoMessage` 视频

`ShiKuManger.SendFileMessage` 文件

.....

详情可见 `ShiKuManger` 类

其中:

消息类型定义在枚举类 kWCMMessageType 中

说明:

发送的 message 最终都会走到 xmppManager.SendMessage (message), 发送图片或文件时(SendMessageFile), 会将文件上传至服务器, 得到服务器返回的文件下载地址后, 才去真正地去发送消息, 详情可见代码:

```
SendMsgPanel.IblSendFile_MouseDown()
```

8. 收消息

所有 xmpp 消息的接收均在 XmppManager 类中统一处理见方法

XmppManger.XmppCon_OnMessage()方法,

不同的消息类型会有不同的处理, 处理完成后会调用界面来刷新出消息

群控制消息通知 NotifactionRoomControlMsg();

新好友控制消息通知 NotifactionVerifyMessage

音视频消息通知 NotifactionMeetingMessage

普通消息通知 NotifactionNormalMessage

其中列表刷新分:

单条刷新: NotifactionListSingleMessage

多条刷新: NotifactionListAllMessage

9. 群组列表

服务器我的群组列表: (url: Applicate.URLDATA.data.apiUrl + "room/list/his";)

群组列表保存在本地 Friend 表中, 通过 isGroup 来区分是一个朋友还是一个群组

见代码 `GroupListLayout.LoadRoomList()`;

当群组发生改变时（退出群组，被踢出群组，创建群组）时，监听 xmpp 发送过来的消息来对群组列表进行刷新

见代码 `GroupListLayout.RegisterMessenger()`;

10. 新建群组

创建一个群组: `Applicate.URLDATA.data.apiUrl+ "room/add"`;

创建群组流程:

输入群组名、群组描述(窗口)、选择联系人、成功后进入群聊

补充:

步骤一: 编辑群组名、群组描述

步骤二: 通过 `ShiKuManager.xmpp.CreateGroup` 创建一个聊天室, 生成一个 `roomJid`
(`ShiKuManager.xmpp.CreateGroup`)

步骤三: 邀请成员加入, 并走一个 Http 接口, 将 `roomJid`、被邀请成员列表、群组名等一些参数传给服务器 (`HttpCreateRoom`)

通过该 Http 接口, 我们主要实现以下几点:

1. 在服务端创建同样的群组, 保持数据的持久化
2. 通过服务端给被邀请成员发送单聊消息, `type==907`, 当被邀请成员收到该条消息后主动加入该聊天室。

11. 加入群组

如果没有加入过群组, 调用接口 (`Applicate.URLDATA.data.apiUrl + "room/join"`);

调用 `ShiKuManager.xmpp.JoinRoom(群组 Jid)` 加入群组。

如果加入过，加载我的群组时（`JoinAllGroups()`），会调用 `ShiKuManager.xmpp.JoinRoom(群组 Jid)`；以 xmpp 的形式加入群组。

12. 删除群组

删除群组功能可见方法 `GroupListLayout.DeleteGroup()`；

说明：直接调用接口（`Applicate.URLDATA.data.apiUrl + "room/delete"`）成功后，服务器会产出一条 xmpp 消息并发送在群组中，xmpp 收到消息后判断是否是删除自己，如果不是，产生通知，存储在数据库，调用界面刷新，如果是自己，则调用 `ExitRoom` 方法退出群聊（后面就再也无法接受到此群消息），然后刷新群组列表，详情见代码 `XmppManager.ProcessGroupManageMessage`

13. 收取历史消息

获取历史消息的方式有两中：

1. 本地数据库 `MessageObject.GetPageListHistory()`；
2. 服务器接口获取：（`Applicate.URLDATA.data.apiUrl + "tigase/shiku_msgs"`）

传入开始时间和结束数据，即可从服务器中拉取到与某人产生的聊天记录

详情见方法 `SendMsgPanel.FirstDownloadRoaming`

14. 邀请好友

邀请好友主要有两个入口

1. 创建时邀请好友见方法 `FrmBuildGroups.InviteToGroup`

说明：使用接口创建群组成功后，

调用接口(Appliccate.URLDATA.data.apiUrl + "room/member/update")来邀请好友，调用成功后，服务器会产生 type 为 907 的消息发送到群组，来告知有新成员进入，详情见代码 XmppManager.ProcessGroupManageMessage

2. 已有群组邀请好友。

逻辑同上，见代码 USEGroupSet.InviteTogroup

15. 踢出成员

踢出成员方法主要在：FrmMoreMember.USEGrouops_Click ()中，该方法会调用接口 (url: Appliccate.URLDATA.data.apiUrl + "room/member/delete");同步服务器数据，成功后更新本地数据库并进行界面的刷新 (UpdateMemberList)。

16. 禁言

禁言在群管理界面的群成员列表的右键菜单中，详情可见

FrmMagent.MemberNoTalk 方法

禁言操作流程：

1. 打开群管理界面 FrmMagent 右键群成员菜单选择禁言，选择禁言时间，调用接口(Appliccate.URLDATA.data.apiUrl + "room/member/update")，服务器会产生相对应的 xmpp 消息，来告知相关的群成员
详情见代码 XmppManager.ProcessGroupManageMessage

17. 消息免打扰

消息免打扰：当开启消息免打扰时，在本地记录相对应的字段

```
LocalDataUtils.GetBoolData(friend.userId + "NOT_DISTURB" + myUserId);
```

到收到消息时，取出判断，如开启了消息免打扰，就不再任务栏进行消息闪烁通知

18. 拉入黑名单

在好友列表中右键选中，点击加入黑名单后，会弹出提示框“是否将该好友加入黑名单”

见代码 FriendListLayout.BlockFriend

确定后会调用接口（Applicate.URLDATA.data.apiUrl + "friends/blacklist/add"）

同步数据库数据，成功后发送拉黑好友推送（ShiKuManager.SendBlockfriend），

收到回执后将好友状态保存到本地然后刷新界面

见代码 XmppManager.ProcessFriendVerifyMessage

19. 类详细介绍

1. 目录结构

文件名或目录名	功能
AForge	录制视频，拍照
db	存放数据库 shiku.db
Res	表情包，Emijo 表情
Model	实体类和数据库相关
Resourec	资源文件
dll	第三方库
Controls	用户控件
Dictionarys	
View	窗体类

VlcLib	播放器相关
APIHelper.cs	HTTP 接口访问相关
App.config	项目配置文件
Applicate.cs	定义全局变量
Helper	
hrtfs	VLC 播放器配置文件
include	
locale	
lua	
plugins	
Libvlc、libvlccore	
APIHelper	
BaiDuMap.html	百度地图
DBSetting.cs	数据库配置
Ffmpeg.exe	视频解码
FrmLookImage	图片查看器
FrmMain	主窗体
FrmMainTest	主窗体（测试）
FrmPlayVideo	视频播放器（暂无用）
UserCardController	
Icon.ico	ICON
Meeting(大全版)	音视频通话类
Settings	Xmpp 监听及操作相关
ShikuManager.cs	

20. 聊天功能相关类详解

1. XMPP 相关

接受消息 XmppCon_OnMessage

1. 回执处理 XmppReceiptManager.OnReceiveReceipt

2. 消息处理 判断收到的 Message 的 Type 字段区分单聊消息

(MessageType.chat) 或群聊消息 (MessageType.groupchat)

连接状态 XmppCon_OnXmppConnectionStateChanged

1. 连接状态 枚举类 XmppConnectionState (Connecting 登录中, Authenticated 已经认证, StartSession 登录成功, Disconnected 断开)

2. 打开连接 XmppClientConnection.Open()

3. 关闭连接 XmppClientConnection.Close()

创建群组 xCreateGroup

退出群组 exitRoom

加入群组 joinGroup

发送消息

1. 发送普通消息 sendXml

2 发送送达回执 receiptXmlId

2. 数据库相关

1. 连接数据库 在 SQLiteDbContext.cs 类实现, 通过 `static string _connectionString = string.Format("Data Source={0};", Path.GetFullPath("./db/" + ConfigurationUtil.GetValue("UserDb")));`
`static SQLiteConnection conn = new SQLiteConnection(_connectionString);` 根据用户 ID 动态配置数据库连接字符串, EF 框架以 CodeFirst 模式在项目所在目录的 db 目录下生成数据库为用户 ID.db 的数据库文件和实体类的表映射, 连接对象 Applicate.dbContext, 执行数据库操作方法调用 SQLiteDbContext.DBAutoConnect() 检查并打开数据库连接。UserDbContext.cs 同理。

2. 动态建表 配置方法在 Model/DAL 目录下, 聊天记录表通过执行 SQL 动态创建, 创建表名为 msg_用户 ID 或群组 Jid, 在 Messageobject 类执行聊天记录表操作。

实体类 Model 目录下, 定义实体类字段和 CRUD 操作

3. 布局相关

登录界面 FrmLogin
注册界面 FrmRegister

主界面左侧 LeftLayout
主界面右边控件 MainPageLayout

最近消息列表 RecentListLayout
我的群组列表 GroupListLayout
我的好友列表 FriendListLayout

新的朋友控件 USEUserVerifyPage
黑名单列表 BlockListPage
群组信息面板 GroupInfo2
好友信息面板 USEFriendInfo

单聊设置页 USESingleSet
群聊设置页 USEGroupSet

查找好友界面 FrmFriendQuery

用户信息面板 FrmFriendsBasic
我的同事 UserMyColleague
我的标签 UserLabel

5. #分析单聊消息的收发以及群组消息的收发#

1. 单聊发送消息

1. 在 ShiKuManager 类 SendTextMessage 用于发送消息。

2. 初始化聊天界面，选中消息列表或双击好友列表中的元素，为单聊对象 Main.Sess 赋值，聊天界面为 SP_chatBottom、gd_medium 与 rtf_sendMessage 组成，SP_chatBottom 主要用于消息的展示，rtf_sendMessage 用于发送文本类型消息，gd_medium 用于发送其他类型的消息；

3. 初始化数据，之前已经提过，只要与某人产生聊天记录，都会在数据库内生成一张 msg_+用户 ID 的表，当我们进入与某人的聊天界面的时候，会通过 Ta

的 ID，去数据库内查询到这张表，得到对应聊天记录，

```
Messageobject tmpMsg = new Messageobject() { myUserId = Applicate.MyAccount.userId, fromUserId =  
Applicate.MyAccount.userId, toUserId = main.Sess.Jid };  
List<Messageobject> msgList = tmpMsg.GetPageList(ref main.msgPage, orderBy.byDesc);
```

之后将得到的数据放入 SP_chatBottom 内，便实现了初步的数据展示

4. 通过 gd_medium 的各种按钮，我们可以知道我们需要发送的消息类型是什么，之后我们对需要发送的消息执行相应的方法。

```
internal static void SendText(string id, string text)  
{  
    try  
    {  
        var mControl = ((MainUIController)Applicate.ViewModelList.FirstOrDefault(v => v is  
MainUIController));  
        //封装Message对象  
        Messageobject msg = new Messageobject  
        {  
            content = (ConfigurationUtil.GetValue("isEncrypt") == "1") ? (AES.EncryptDES(text,  
"1234567")) : (text), //设置信息内容  
            isEncrypt = Convert.ToInt16(ConfigurationUtil.GetValue("isEncrypt")),  
            FromId = Applicate.MyAccount.userId, //发送方UserId  
            fromUserId = Applicate.MyAccount.userId, //发送方UserId  
            fromUserName = Applicate.MyAccount.nickname, //接收方(好友)的好友名字  
            isMySend = 1, //是自己发送的  
            ToId = id,  
            toUserId = id, //接收方UserID  
            toUserName = mControl.main.Sess.NickName, //接收方昵称  
            myUserId = Applicate.MyAccount.userId  
        };  
        msg.isGroup = (mControl.main.Sess.isGroup) ? (1) : (0);  
        msg.jid = (msg.isMySend == 1) ? (msg.toUserId) : (msg.fromUserId);  
        msg.timeSend = Utils.DatetimeToStamp(DateTime.Now); //发送时间(使用时间戳)  
        msg.messageId = Guid.NewGuid().ToString("N"); //设置Message唯一ID  
        msg.type = kWCMessageType.Text; //发送信息1，文本类型  
        //更新消息列表中内容  
        mControl.main.MessageList[mControl.main.Lb_MessageList.SelectedIndex].Msg = msg;  
        mControl.CreateOrUpdateMsgItem(msg);  
        msg.myUserId = Applicate.MyAccount.userId;  
        msg.Insert(); //存库  
        xmpp.sendXml(msg);  
    }  
}
```

```

        catch (Exception ex)
        {
            LogHelper.log.Error(ex.Message, ex);
            Console.WriteLine("-----*****-----发送信息时出错: " + ex.Message);
        }
    }
}

```

2. 单聊接收消息

1. 在 xmppManager 内注册 XmppCon_OnMessage 监听，在之前已经详细介绍了这个监听的作用

```

private void XmppCon_OnMessage(object sender, Message xmppMsg)
{
    Console.WriteLine("-----XmppCon_OnMessage:" + xmppMsg);
    #region 如果Xmpp消息中body为null并且没有送达回执，跳出此方法
    if (xmppMsg.Body == null && !xmppMsg.HasTag("received"))
        return;
    #endregion
    Messageobject reciveJsonMsg = new Messageobject();
    receiptXmlId(xmppMsg.Id, xmppMsg.From, xmppMsg.To);//发送送达
    #region 消息回执(更新本地的发送中标志~)
    if (xmppMsg.HasTag("received"))//如果XML节点中含有recived节点则为送达回执
    {
        string id = xmppMsg.GetAttribute("id");
        xmppMsg.ChildNodes.ToString();
        string messageId = xmppMsg.FirstChild.GetAttribute("id");//获取首个子项的id属性值
        string from = xmppMsg.GetAttribute("from");
        int i = from.IndexOf('@');
        from = from.Substring(0, i);
        var tMsg = Applicate.SendingList.FirstOrDefault(m => m.MessageId == messageId);
        //撤回回执
        var rMsg = Applicate.RecallList.FirstOrDefault(m => m.messageId == messageId);
        if (tMsg != null)
        {
            DoMsgReceipt(tMsg.TmpMsg);//接收到消息回执时(通知所有带有MessageIdUI)
            new Messageobject() { myUserId = Applicate.MyAccount.userId, fromUserId = Applicate.MyAccount.userId, toUserId = from }.UpdateIsSent(messageId);
        }
        if (rMsg != null)
        {
            DoRecallMsgReceipt(rMsg);//接收到消息回执时(通知所有带有MessageIdUI)
            new Messageobject() { myUserId = Applicate.MyAccount.userId, fromUserId = Applicate.MyAccount.userId, toUserId = from, messageId = rMsg.content }.Delete();
        }
    }
}

```



```

        //你撤回一条消息
    }
    Console.WriteLine("//接收到消息回执 -- " + messageID);
    return;
}
#endregion
#region 不知道什么类型的消息
else if (xmppMsg.HasTag(typeof(agsXMPP.protocol.extensions.ibb.Data)))
{
    // ignore IBB messages

    MessageBox.Show("-----agsXMPP.protocol.extensions.ibb.Data-----XmppCon_OnMessage: " +
xmppMsg.Body);
    return;
}
#endregion
#region 单聊消息
else if (xmppMsg.Type == MessageType.chat)
{
    receiveJsonMsg = (Messageobject)JsonConvert.DeserializeObject(xmppMsg.Body,
typeof(Messageobject));//反序列化
    #region 拼接完善json消息
    if (receiveJsonMsg.messageId == null || receiveJsonMsg.messageId == "")
        receiveJsonMsg.messageId = xmppMsg.Id;
    if (receiveJsonMsg.fromUserId == "" || receiveJsonMsg.fromUserId == null)
        receiveJsonMsg.fromUserId = xmppMsg.From.User;
    if (receiveJsonMsg.toUserId == "" || receiveJsonMsg.toUserId == null)
        receiveJsonMsg.toUserId = xmppMsg.To.User;

    receiveJsonMsg.isGroup = 0;//非群聊消息
    receiveJsonMsg.jid = xmppMsg.From.User;//聊天对象
    receiveJsonMsg.isSend = 1;//已送达
    receiveJsonMsg.myUserId = Applicate.MyAccount.userId;
    #endregion
}
#endregion
#region 群聊消息
else if (xmppMsg.Type == MessageType.groupchat)
{
    Console.WriteLine("-----XmppCon_OnMess
age收到一条群组消息: " + xmppMsg);
    #region 判断是否包含json数据
    if (xmppMsg.Body != null)

```

```

{
    if (!xmppMsg.Body.Contains("{}"))
    {
        //如果消息Body中不包含 { 就跳出方法
        return;
    }
}
#endregion
//反序列化为MessageObject对象
recvJsonMsg = (MessageObject)JsonConvert.DeserializeObject(xmppMsg.Body,
typeof(MessageObject));

#region 解密
if (recvJsonMsg.isEncrypt == 1)
    recvJsonMsg.content = AES.DecryptDES(recvJsonMsg.content, null); //此处需要改参数
#endregion
#region 拼接json消息
if (recvJsonMsg.messageId == null || recvJsonMsg.messageId == "")
{
    recvJsonMsg.messageId = xmppMsg.Id;
}
//设置为群聊消息
recvJsonMsg.isGroup = 1;
//设置UserID
if (recvJsonMsg.toUserId == "")
{
    recvJsonMsg.toUserId = Applicate.MyAccount.userId; //接收者UserId
    recvJsonMsg.toUserName = Applicate.MyAccount.nickname; //接收者UserName
}
recvJsonMsg.jid = xmppMsg.From.User; //发送者
recvJsonMsg.myUserId = Applicate.MyAccount.userId;
#endregion
if (recvJsonMsg.fromUserId == Applicate.MyAccount.userId)
    return;
}
#endregion
recvJsonMsg.isMySend = 0;
recvJsonMsg.FromId = xmppMsg.From.User; //
recvJsonMsg.ToId = xmppMsg.To.User;

DoMessage(recvJsonMsg);
}

```

2. DoMessage 内：消息类型处理、“新的好友”处理、存入数据库、通知机制。

1. “新的好友”处理调用 ProcessVerifyingMsgBegin 方法，我们定义为用户 ID=10001 的特殊用户，用于部分消息类型，包括（ kWCMessageType.RequestFriendDirectly:508 打招呼， kWCMessageType.RequestAgree:对方通过验证， kWCMessageType.DeleteFriend:被彻底删除， kWCMessageType.BlackFriend:被拉黑， kWCMessageType.BlackFriendCancel:取消黑名单等）的处理，保存并显示在“新的好友”对话中。

2. 通知机制通过 Applicate.ViewModelList 里保存需要通知的对象，对象继承 IXmppInterface 接口，在收到回执或者不同的消息类型后遍历通知对象执行相应的接口方法，接口方法包括（OnMsgCome:收到一条消息，OnMsgReceipt:发送一条消息,收到回执，OnMsgRead:接收者已读消息，OnMsgReadSendReceipt:我已读了一条消息，并收到回执，OnDeleFriend:删除一个好友，OnQuitRoom:有人退出一个群组，OnDeleteRoom:群主解散了一个群组，OnSendTimeout:重发 3 次后，仍超时，OnMsgCancelReceipt:撤回消息，收到回执，OnReceiveFile:收到文件，下载完毕），方法主要实现通知 UI 更新和数据库操作，聊天消息处理在 MainUIController.OnMsgCome 执行。

```
private void DoMessage(MessageObject msg)
{
    if (msg == null)
        return;
    msg.myUserId = Applicate.MyAccount.userId;
    bool IsProcessed = false; //消息是否已处理
    if (msg.type >= kWCMessageType.kWCMessageTypeRoomNameChange
    && msg.type <= kWCMessageType.kWCMessageTypeRoomInvite) //群聊
    {
        ProcessGroupMessage(msg);
    }
    switch (msg.type)
```

```

{
    case kWCMessageType.kWCMessageTypeRoomExit://被踢
        msg.jid = msg.objectId;
        break;
    case kWCMessageType.kWCMessageTypeRoomInvite://收到群聊邀请
        msg.jid = msg.objectId;
        if (msg.toUserId == Applicate.MyAccount.userId)
        {
            this.joinGroup(msg.objectId, msg.content);
        }
        break;
    case kWCMessageType.kWCMessageTypeRoomNotice://公告更改
    case kWCMessageType.kWCMessageTypeRoomNameChange://名称更改
        msg.FromId = msg.objectId;
        msg.jid = msg.objectId;
        msg.fromUserId = msg.objectId;//赋值群聊Id
        break;
    case kWCMessageType.kWCMessageTypeRoomDismiss://解散群聊
        msg.jid = msg.objectId;
        msg.content = "该群已被群主解散";
        break;
    case kWCMessageType.kWCMessageTypeReqRefuse://拒绝的回复
    case kWCMessageType.kWCMessageTypeFriReq://打招呼
        msg.jid = "10001";
        break;
    case kWCMessageType.kWCMessageTypeGreetSb://508打招呼
    case kWCMessageType.kWCMessageTypeReqAgree://对方通过验证
    case kWCMessageType.kWCMessageTypeDelFri://被彻底删除
    case kWCMessageType.kWCMessageTypeDefriend://被拉黑
    case kWCMessageType.kWCMessageTypeDeblack://取消黑名单
        ProcessVerifyingMsgBegin(msg);//10001处理
        IsProcessed = true;
        break;
    case kWCMessageType.kWCMessageTypeIsRead:
        DoMsgRead(msg);//接收者已读消息
        IsProcessed = true;
        break;
    case kWCMessageType.kWCMessageTypeWithdraw:
        new Messageobject() { myUserId = Applicate.MyAccount.userId, fromUserId =
msg.fromUserId, toUserId = msg.toUserId, messageId = msg.content }.Delete();
        break;
    case kWCMessageType.kWCMessageTypeNone://空消息不处理
        return;
    case kWCMessageType.kWCMessageTypeRoomMemberDe:

```

```

        msg.jid = msg.objectId;
        break;
    default:
        break;
    }
    if (IsProcessed)
        return; //不处理
    else
        msg.Insert(); //存库

    for (int i = 0; i < Applicate.ViewModelList.Count; i++)
        App.Current.Dispatcher.Invoke(() =>
        { ((IXmppInterface)Applicate.ViewModelList[i]).OnMsgCome(msg); }); //通知机制
    }

```

3. 群组发送消息

1. 在 ShiKuManager 类 SendText 用于发送消息。在 SendText 方法中调用了 xmppManager 的 sendXml(Messageobject jsonMsg)的方法，群聊接收的 Jid 和 Type 与单聊不同

```

public void sendXml(Messageobject jsonMsg)
{
    //生成XML信息
    Message xmlMsg = new Message();
    if (jsonMsg.messageId == null)
        jsonMsg.messageId = Guid.NewGuid().ToString("N");
    if (jsonMsg.isGroup >= 1)
    {
        xmlMsg.To = new Jid(jsonMsg.toUserId + "@muc." + XmppCon.Server); //设置消息的接收人
        xmlMsg.Type = MessageType.groupchat; //设置为群消息类型
    }
    else
    {
        #region 送达回执
        //如果非群聊需要请求对方发送回执
        Element reqRec = new Element("request"); //请求接送方发送消息回执
        reqRec.SetAttribute("xmlns", "urn:xmpp:receipts"); //设置回执属性
        xmlMsg.AddChild(reqRec); //添加请求回执的节点e
        xmlMsg.SetAttribute("xmlns", "jabber:client");
        #endregion
        xmlMsg.To = new Jid(jsonMsg.toUserId + "@" + XmppCon.Server); //设置消息的接收人
        xmlMsg.Type = MessageType.chat; //设置为单聊类型
    }
}

```

```

    }
    xmlMsg.From = jsonMsg.fromUserId + "@" + XmppCon.Server; //自己的
    xmlMsg.Body = jsonMsg.ToString(); //JsonConvert.SerializeObject(jsonMsg); //
    xmlMsg.Id = jsonMsg.messageId; //设置协议中uid为body中的ui
    AddSendingList(jsonMsg); //添加到临时重发消息列表中
    XmppCon.Send(xmlMsg);
}

```

2. 初始化聊天界面，选中消息列表或双击群组列表中的元素，为群聊对象 Main.Sess 赋值，聊天界面为 SP_chatBottom、gd_medium 与 rtf_sendMessage 组成，SP_chatBottom 主要用于消息的展示，rtf_sendMessage 用于发送文本类型消息，gd_medium 用于发送其他类型的消息；

3. 初始化数据，之前已经提过，只要与某个群产生聊天记录，都会在数据库内生成一张 msg_+群组 Jid 的表，当我们进入与某个群的聊天界面的时候，会通过群组 Jid，去数据库内查询到这张表，得到对应聊天记录，与单聊一样

4. 通过 gd_medium 的各种按钮，我们可以知道我们需要发送的消息类型是什么，与单聊一样。

4. 群组接收消息

步骤与单聊一样。

21. 语音和视频聊天

1. 音视频下的 XMPP 协议说明

下面为音视频通话过程中每个操作所对应的 XMPP 消息协议，每条协议都代表一种操作，ex：A 想与 B 进行语音通话，此时，A 需要发送一条 Type 为 100 的 XMPP 消息给 B，当 B 收到消息时，判断 Type，当 Type==100 时，知道 A 正在呼叫我，于是弹起来电显示界面，当 A 或 B 想结束此次语音通话时，也需要发送一条 Type 为 104 的 XMPP 消息给对方。

```

#region 语音前
kWCMessageTypeAudioChatAsk = 100, //询问是否准备好语音通话
kWCMessageTypeAudioChatReady = 101, //已准备好语音通话
kWCMessageTypeAudioChatAccept = 102, //接受语音通话
kWCMessageTypeAudioChatCancel = 103, //拒绝语音通话 或 取消拨号
kWCMessageTypeAudioChatEnd = 104, //结束语音通话
#endregion

#region 视频
kWCMessageTypeVideoChatAsk = 110, //询问是否准备好视频通话
kWCMessageTypeVideoChatReady = 111, //已准备好视频通话
kWCMessageTypeVideoChatAccept = 112, //接受视频通话
kWCMessageTypeVideoChatCancel = 113, //拒绝视频通话 或 取消拨号
kWCMessageTypeVideoChatEnd = 114, //结束视频通话
#endregion

#region 视频会议
kWCMessageTypeVideoMeetingInvite = 115, //邀请加入视频会议
kWCMessageTypeVideoMeetingJoin = 116, //加入视频会议
kWCMessageTypeVideoMeetingQuit = 117, //退出视频会议
kWCMessageTypeVideoMeetingKick = 118, //踢出视频会议
#endregion

#region 语音会议
kWCMessageTypeAudioMeetingInvite = 120, //邀请加入语音会议
kWCMessageTypeAudioMeetingJoin = 121, //加入语音会议
kWCMessageTypeAudioMeetingQuit = 122, //退出语音会议
kWCMessageTypeAudioMeetingKick = 123, //踢出语音会议
kWCMessageTypeAudioMeetingSetSpeaker = 124, //踢出语音会议
kWCMessageTypeAudioMeetingAllSpeaker = 125, //踢出语音会议
#endregion

```

2. 音视频通话

MeetForManger.cs 是音视频通话中我们把所有的拨打，接听，通话界面包括单聊与群聊的方法都在这个管理类中，然后分别处理这些方法分别传递给不同界面。

```

/// <summary>
/// 显示拨号界面
/// </summary>

```

```

public void ShowDialForm(MessageObject message, string url, bool isGroup=false)
{
    Friend friend = message.GetFriend();
    //好友
    if (!isGroup)
    {
        if (mCurrState == STATE_IDLE)
        {
            //拨打界面
            FrmDial frmDial = new FrmDial();
            frmDial.CallPhone(friend, url, message.type == kWCMessageType.VideoChatAsk);
            mCurrState = STATE_DIALING;
            return;
        }
    }
    //群组
    else
    {
        if (mCurrState == STATE_IDLE)
        {
            FrmRecivePhone frmRecive = new FrmRecivePhone();
            // 进到视频通话界面 url
            //通话界面
            frmRecive.ShowGroup(url, true);
            mCurrState = STATE_CALLING;
            return;
        }
    }
    //不等于 0 发忙线
    if (mCurrState != STATE_IDLE)
    {
        ShiKuManager.SendBusyMeetMessage(friend);
    }
}

/// <summary>
/// 显示接听界面
/// </summary>
public void ShowAnswerForm(MessageObject message, string url, bool isGroup=false) {
    Friend friend = message.GetFriend();
    //好友
    if (!isGroup)
    {

```



```

        if (mCurrState == STATE_IDLE)
        {
            FrmAnswer frm = new FrmAnswer();
            frm.fillDataFrined(friend, message, message.type == kWCMessageType.VideoChatAsk,
url);

            mCurrState = STATE_ANSWERING;
            return;
        }
    }
    //群组
    else
    {
        if (mCurrState == STATE_IDLE)
        {
            FrmAnswer frm = new FrmAnswer();
            frm.fillDataGroup(friend, message, message.type == kWCMessageType.VideoChatAsk,
true);

            frm.Url = url;
            mCurrState = STATE_ANSWERING;
            return;
        }
    }
    //不等于 0 发忙线
    if (mCurrState != STATE_IDLE)
    {
        ShiKuManager.SendBusyMeetMessage(friend);
    }
}

/// <summary>
/// 显示通话中界面
/// </summary>
public void ShowRecivePhoneForm(MessageObject message,string url,bool isGroup=false)
{
    Friend friend = message.GetFriend();
    //群组
    if (!isGroup)
    {
        if (!isRecovePhone)
        {
            FrmRecivePhone meetHtml = new FrmRecivePhone();
            meetHtml.fillDataFrined(friend, message.type == kWCMessageType.VideoChatAccept,
url);

            mCurrState = STATE_CALLING;

```

```

        return;
    }
}
//好友
else
{
    if (!isRecovePhone)
    {
        FrmRecivePhone meetHtml = new FrmRecivePhone();
        meetHtml.ShowGroup(url, true);
        mCurrState = STATE_CALLING;
        return;
    }
}
if (mCurrState != STATE_IDLE)
{
    ShiKuManager.SendBusyMeetMessage(friend);
}
}

```

发起方：在 meeting 文件夹下的 MeetFormManager.cs 类中调用 ShowDialForm()方法给拨打界面，进行传递参数，在拨打界面有个方法来接受类传递过来的参数
拨号界面的方法，接受到 MeetForManger.cs 类传递过来的参数，弹起拨打界面，并且开启计时器

//拨打电话的方法

```

public void CallPhone(Friend friend, string url, bool video)
{
    Url = url;
    Isvideo = video;
    //实例化
    myFriend = friend;
    //数据填充
    FillDataList(friend);
    //拨打
    isChoose = true;
    ///开启线程池
    Task.Factory.StartNew(() =>
    {
        //等待 30 秒
        Thread.Sleep(30 * 1000);
        //如果接听
    }
    );
}

```

```

        if (!isChoose)
        {
            //对方不愿意
            ShiKuManager.SendCancelMeetMessage(friend, video);
        }
    });
    this.Focus();
    this.Show();
}

```

如果对方响应，我们会收到对方发过来的 102 消息，先进入到 MeetForManger.cs 类中调取通话界面的传递参数的方法，在将次参数传递给通话界面的方法，，进入到通话界面，我们传递了一个自己的 url，并且开启计时器，内置浏览器控件访问 url(<https://meet.youjob.co/>+自己的 userId)，此时我们在进入通话界面。

```

#region 参数传递
//好友的接听
internal void fillDataFrined(Friend friend, bool video, string url)
{
    myVdieo = video;
    myFriend = friend;
    Url = url;
    this.Show();
    //开始计时
    StartTime = TimeUtils.CurrentIntTime();
}
#endregion

```

接收方：对方发起了与我的语音通话请求，我们能够收到 type 为 100 的消息，马上弹起来电显示页面，与发起方一样，在跳转到来电显示界面时，需要将一些参数传过去。在来电显示界面，我们也创建一个定时器

当我们按下接听按钮时，我们需要发送 type==102 的消息告诉对方我们已接听，并且和发起方一样带参进入通话界面，在通话界面，发起方与接收方所做的操作一致。

视频通话同理，只是我们收到、发出的消息 Type 不一致，具体查看 kWCMesageType 类。

3. 音频会议:

发起方: 在聊天窗口点击了语音会议的按钮, 此时会弹出会议邀请界面, 我们在跳转至邀请界面时, 需要也需要传过去一些参数, bool isType: 通话的类型, 用于拼接会议地址的群组 Jid., bool isGroup 类型的参数该参数用来判断是否为群是群就为 true。在邀请玩成员后, 我们需要给所有被邀请的人发送一条 type==120 的消息。

当邀请信息发送后, 我们就直接进入通话页面, 当我们退出会议时, 我们不需要在发送退出协议给其他人, 直接关闭窗口即可。

接收方: 当我们收到 Type==120 的消息时, 我们知道某人邀请我参加语音会议, 直接弹出来电显示界面

在进入来电显示界面后, 如果我们挂断和接听, 都不需要发送对应的协议, 直接进入通话界面或关闭窗口即可。

视频会议同理, 只是我们收到、发出的消息 Type 不一致, 具体查看 kWCMMessageTypes 类。

8. 配置说明

音视频通话依靠访问 url, 需要依赖 cefSharp 插件, debug 需要选择 32 位或 64 位平台, 配置属性在 WinFrmTalk.Model.OpenVideo 类中。

```
#region 浏览器配置
//默认浏览器
public readonly string Default = "http://www.google.com/";
private static readonly bool DebuggingSub = Debugger.IsAttached;
//验证窗体被动打开事件
public static bool isInitSet;

//加载视频的方法
public static RequestContext RequestContext { get; }
public static void Init()
```

```

{
    //默认为 true
    isInitSet = true;
    var setting = new CefSettings();
    //端口号
    setting.RemoteDebuggingPort = 8080;
    //语言
    setting.Locale = "zh-CN";
    setting.AcceptLanguageList = "zh-CN";
    //开启音视频
    setting.CefCommandLineArgs.Add("enable-media-stream", "enable-media-stream");
    //验证证书
    setting.IgnoreCertificateErrors = true;
    //日志
    setting.LogSeverity = CefSharp.LogSeverity.Verbose;
    //开启验证
    if (!Cef.Initialize(setting))
    {
        if (Environment.GetCommandLineArgs().Contains("--type=renderer"))
        {
            Environment.Exit(0);
        }
        else
        {
            return;
        }
    }
}

#endregion

```

22. 安全机制

1. 接口安全机制: 所有接口, 都需要额外传递两个参数给服务端, time 与 secrte 参数, 且不同的接口, secret 生成的规则不同, 如果为需要支付密码, 收付款调用的接口, 又引入了一套需要支付密码参与运算的生成规则, 同时, 服务端会对 secrte 参数进行验证, 如 secret 参数不一致, 接口将无法访问成功, secret 生成规程可查看 GetBuilder 类 HttpEncrypt 方法。

2. 消息安全机制:

1. 消息加密传输, 用户在隐私设置内可选消息加密传输, 开启该选项后, 所有消息在传输过程中 content 字段均为密文, 加密方式采用的 3DES 的加密方式, 对称密钥取自消息内, 且经过复杂运算, 能保证每条消息的对称密钥均不一致。见
`SkSSLUtils.EncryptMessage ()`;
2. 阅后即焚, 用户在好友设置内开启阅后即焚后, 所产生的阅后即焚消息均不存服务器, 且对方阅读之后本地立即销毁, 保证私密聊天安全性

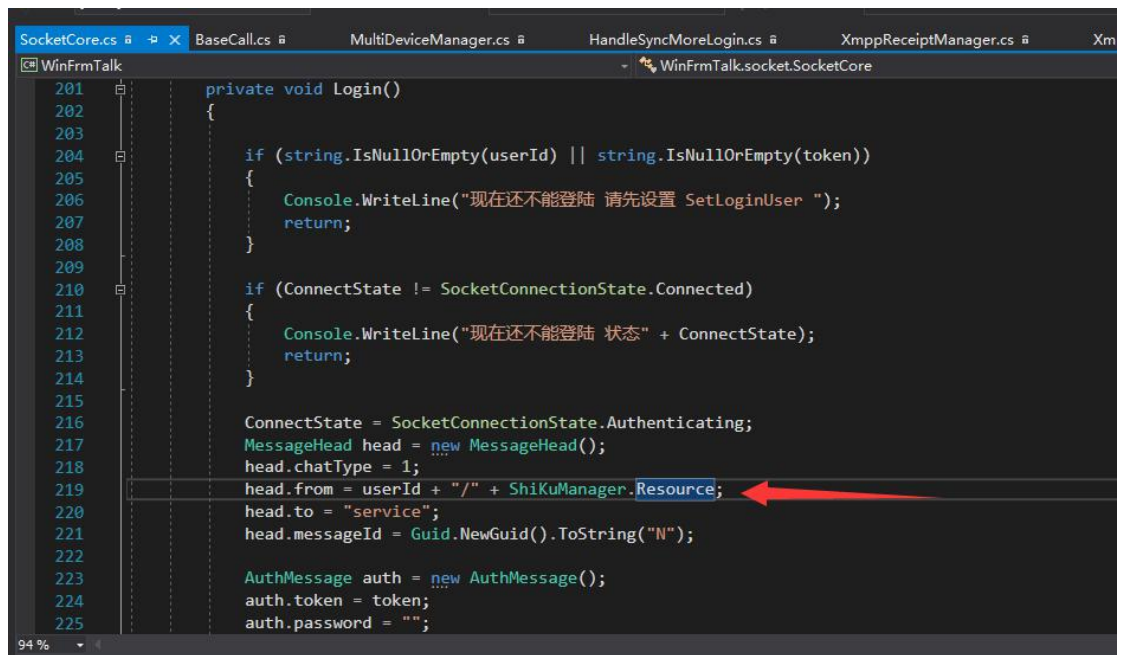
23. 多设备登录

允许一个账号同时登录多种不同平台的设备, 达到在多地同时登录的功能

1、 如何实现多点登录?

```
#region 初始化XMPP连接
/// <summary>
/// 初始化XMPP连接对象
/// </summary>
/// <param name="curruser">当前的用户对象</param>
/// <param name="Pwd">用户密码</param>
1 个引用
public static void InitialXmpp()
{
    Resource = MultiDeviceManager.Instance.IsEnabled ? "pc" : "youjob";
    //实例化XmppManager类
    mSocketCore = new XmppManager();
}
#endregion
```

当开启多点登录时使用PC为resource登录



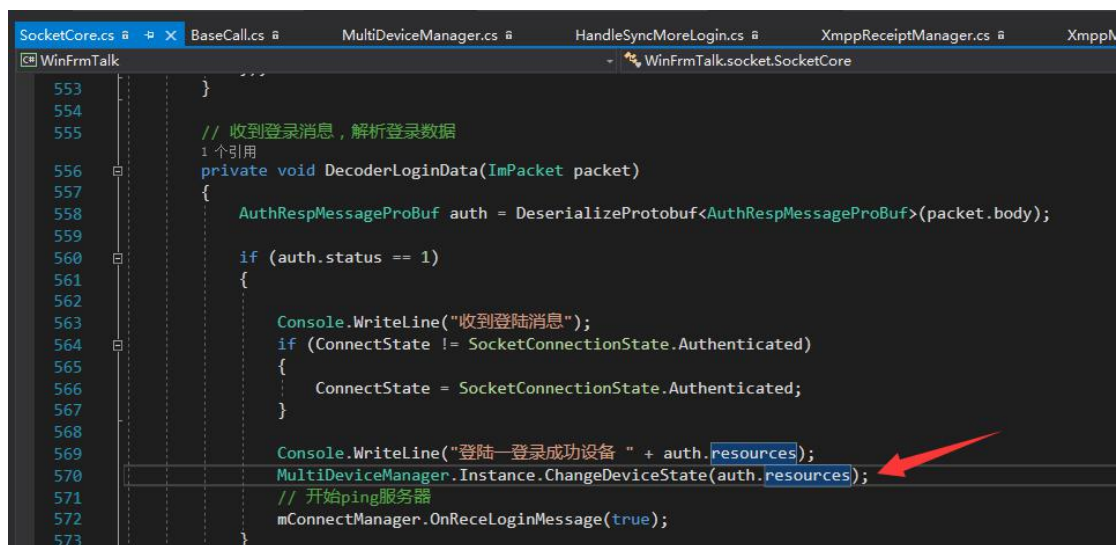
```
201 private void Login()
202 {
203     if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(token))
204     {
205         Console.WriteLine("现在还不能登陆 请先设置 SetLoginUser ");
206         return;
207     }
208
209     if (ConnectState != SocketConnectionState.Connected)
210     {
211         Console.WriteLine("现在还不能登陆 状态" + ConnectState);
212         return;
213     }
214
215     ConnectState = SocketConnectionState.Authenticating;
216     MessageHead head = new MessageHead();
217     head.chatType = 1;
218     head.from = userId + "/" + ShiKuManager.Resource;
219     head.to = "service";
220     head.messageId = Guid.NewGuid().ToString("N");
221
222     AuthMessage auth = new AuthMessage();
223     auth.token = token;
224     auth.password = "";
225 }
```

在发送登录请求给服务端时，每端在拼接 From 参数时，使用不一样的后缀即可实现多设备登录

2、如何只让手机与电脑端多点登录?如果关闭多点登录?

手机端使用同一个后缀，电脑端使用同一个后缀，所有端使用同一个后缀。

3、如何检测到其他端是否在线?



```
553 }
554
555 // 收到登录消息，解析登录数据
556 private void DecoderLoginData(ImPacket packet)
557 {
558     AuthRespMessageProBuf auth = DeserializeProtobuf<AuthRespMessageProBuf>(packet.body);
559
560     if (auth.status == 1)
561     {
562         Console.WriteLine("收到登陆消息");
563         if (ConnectState != SocketConnectionState.Authenticated)
564         {
565             ConnectState = SocketConnectionState.Authenticated;
566         }
567
568         Console.WriteLine("登陆—登录成功设备 " + auth.resources);
569         MultiDeviceManager.Instance.ChangeDeviceState(auth.resources);
570         // 开始ping服务器
571         mConnectManager.OnReceLoginMessage(true);
572     }
573 }
```

登录成功后，服务器会在登录信息中返回所有在线设备的 resources 后缀，根据这个即可判断出当前有多少设备同时在线了