

# WIN 开发文档

ChangeLog.....	4
1. 概述.....	5
1.1. ImSDK 集成.....	5
1.1.1. 下载 ImSDK.....	5
1.1.2. 集成 ImSDK.....	5
1.1.2.1. 头文件.....	5
1.1.2.2. 引用库.....	5
1.1.2.3. 关于回调等的注意事项.....	5
1.1.2.4. 关于句柄.....	6
1.1.3. 功能开发.....	6
1.1.4. 支持版本.....	6
1.2. ImSDK 基本概念.....	6
1.2.1. ImSDK 接口简介.....	7
1.2.2. 调用顺序介绍.....	8
2. 初始化.....	9
2.1. 设置测试环境.....	9
2.2. 设置通信模式.....	10
2.3. 设置日志等文件路径（可选）.....	10
2.4. 新消息通知.....	11
2.5. 网络事件通知.....	12
2.6. 用户状态变更.....	13
2.7. 初始化.....	13
3. 登录.....	14
3.1. 登录.....	14
3.2. 登出.....	15
3.3. 强制登录.....	16
4. 消息收发.....	16
4.1. 消息发送.....	16
4.1.1. 文本消息发送.....	17
4.1.2. 图片消息发送.....	18
4.1.3. 表情消息发送.....	20
4.1.4. 自定义消息发送.....	21
4.2. 从本地存储获取消息.....	23
4.3. 获取所有会话.....	24
4.4. 删除会话.....	25
4.5. 消息解析.....	25
4.5.1. 基本属性解析.....	26
4.5.2. 图片消息解析.....	27
4.6. 系统解析.....	28
5. 未读计数.....	28
5.1. 获取当前未读消息数量.....	28

5.2.	已读上报.....	28
6.	群组管理.....	29
6.1.	群组类型.....	29
6.2.	群组消息.....	30
6.3.	创建群组.....	31
6.4.	加用户入群.....	32
6.5.	退出群组.....	34
6.6.	删除群组成员.....	35
6.7.	获取群成员列表.....	37
6.8.	获取加入的群组列表.....	38
6.9.	获取群组资料.....	40
6.10.	修改群资料.....	42
6.11.	解散群组.....	45
6.12.	群事件消息.....	45
6.12.1.	用户被邀请加入群组.....	49
6.12.2.	用户退出群组.....	49
6.12.3.	用户被踢出群组.....	49
6.12.4.	被设置/取消管理员.....	50
6.12.5.	群资料变更.....	50
6.12.6.	群成员资料变更.....	51
6.13.	群系统消息.....	51
6.13.1.	申请加群消息.....	53
6.13.2.	申请加群同意/拒绝消息.....	53
6.13.3.	被管理员踢出群组.....	53
6.13.4.	群被解散.....	53
6.13.5.	创建群消息.....	53
6.13.6.	邀请加群.....	53
6.13.7.	主动退群.....	54
6.13.8.	设置/取消管理员.....	54
6.13.9.	群被回收.....	54
7.	用户资料.....	54
7.1.	设置自己昵称.....	54
7.2.	设置好友验证方式.....	55
7.3.	获取自己的资料.....	55
7.4.	获取好友的资料.....	56
7.4.1.	获取好友资料的固定字段.....	56
7.4.2.	获取好友资料的部分字段.....	57
7.5.	添加好友.....	60
7.6.	删除好友.....	61
7.7.	获取所有好友.....	62
7.8.	同意/拒绝好友申请.....	63
7.9.	关系链变更系统通知.....	64
7.9.1.	添加好友系统通知.....	65
7.9.2.	删除好友系统通知.....	65

7.9.3.	好友申请系统通知.....	66
7.9.4.	删除未决请求通知.....	66
7.10.	好友资料变更系统通知.....	66

## ChangeLog

### Version 1.1

- 1、设置好友备注
- 2、无网络情况下查看历史消息

### Version 1.1

- 1、公有群支持，增加群简介和群公告，增加禁言、消息屏蔽、群身份设置
- 2、聊天室支持
- 3、用户关系链操作接口
- 4、用户资料操作接口（昵称、加好友设置）
- 5、日志回调接口增加日志级别
- 6、增加记录日志接口，用户可以使用 SDK 日志功能输出日志
- 7、logout 增加回调，解决快速切换帐号出现登录失败的问题
- 8、图片三档：原图，缩略图，大图，上传下载接口变更，传递图片路径
- 9、增加互踢功能
- 10、修复重复登录报错
- 11、增加 crash 自动上报功能
- 12、图片独立上传接口

### Version 1.0

- 1、登录、登出
- 2、C2C 消息（文本、图片、表情、语音、地理位置、自定义消息）
- 3、私有群消息（文本、图片、表情、语音、地理位置、自定义消息）
- 4、私有群管理
- 5、APNs 离线推送（上报 Token，前台后切换事件上报）
- 6、消息本地存储

## 1. 概述

### 1.1. ImSDK 集成

本节主要介绍如何创建一个应用，并集成 ImSDK。

#### 1.1.1. 下载 ImSDK

从官网下载 ImSDK

#### 1.1.2. 集成 ImSDK

建立 VC 工程或其他 IDE 工程

##### 1.1.2.1. 头文件

引入头文件 sdk\include\tim\_\*.h

##### 1.1.2.2. 引用库

库文件 libtim.dll libtim.lib

**注意：**WinSDK 使用编译选项 /MD &/MDD，以动态库的形式引用运行时库。

调用时需保持一致

另外，运行时依赖 VS2010 SP1 相关 CRT。

##### 1.1.2.3. 关于回调等的注意事项

sdk 使用异步回调的方式，在大部分接口中返回调用结果。

例如：

//sdk 定义

```
typedef void (*CBOnSuccess) (void* data);
typedef void (*CBOnError) (intcode, constchar* desc, void*
data);
```

```
typedef struct_TIMCallback_c
{
    CBOnSuccessOnSuccess;
    CBOnError OnError;
    void* data;
}TIMCommCB;
TIMCommCBcallback;
callback.OnSuccess = CBCommOnSuccessImp;
```

```
callback.OnError = CCommOnErrorImp;  
TIMLogin(sdk_app_id, &user, user_sig, &callback);  
SLEEP(1);
```

在用户登陆成功以后，sdk 调用 callback 中的函数指针返回结果。

其中，如果登陆成功，sdk 调用 callback 中注册的 OnSuccess

并且将注册的用户数据 void\* data 在接口中传回用户。反之亦然

**注意：WinSDK 库使用 用户回调线程调用回调函数。如果阻塞回调线程将影响 SDK 库功能。例如无法收发包等等。所以建议用户以可靠的方式传递回调数据到主线程或用户自有线程处理。避免阻塞用户回调。**

#### 1.1.2.4. 关于句柄

SDK 库使用句柄封装资源。用户创建 SDK 库接口定义的资源前，调用相应的 Create\*Handle 接口创建句柄，在句柄使用完毕后，调用相应的 Destroy\*Handle 接口销毁接口，避免资源泄露。当句柄是在回调中传递给用户的时候，用户可选择使用 Clone\*Handle 接口复制句柄，在句柄使用完毕以后，仍然销毁资源。当然也可以直接操作 Handle，需要注意避免阻塞回调线程。

#### 1.1.3. 功能开发

根据后续章节的开发指引进行功能的开发。其中函数调用顺序可参见（调用顺序介绍）。

#### 1.1.4. 支持版本

ImSDK 支持 winxp 及 win7 系统。

### 1.2. ImSDK 基本概念

**会话：**ImSDK 中会话(Conversation)分为两种，一种是 C2C 会话，表示单聊情况自己与对方建立的对话，读取消息和发送消息都是通过会话完成；另一种是群会话，表示群聊情况下，群内成员组成的会话，群会话内发送消息群成员都可接收到。

如下图所示，一个会话表示与一个好友的对话：



**消息：**ImSDK 中消息(Message)表示要发送给对方的内容，消息包括若干属性，如是否自己已读，是否已经发送成功，发送人帐号，消息产生时间等；一条消息由若干 Elem 组合而成，每种 Elem 可以是文本、图片、表情等等，消息支持多种 Elem 组合发送。



**群组 Id：**群组 Id 唯一标识一个群，由后台生成，创建群组时返回。

### 1.2.1. ImSDK 接口简介

ImSDK 接口主要分为通讯管理器，会话、消息，群管理，具体参见下表：

接口	介绍	功能
tim_c.h	管理器，负责基本的 SDK 操作	初始化登录、注销、创建会话等
tim_conv_c.h	会话，负责会话相关操作	如发送消息，获取会话消息缓存，获取未读计数等
tim_msg_c.h	消息	包括文本、图片等不同类型消息

tim_group_c.h	群管理	负责创建群，增删成员，以及修改群资料等
tim_friend_c.h	资料关系链管理	负责获取、修改好友资料和关系链信息

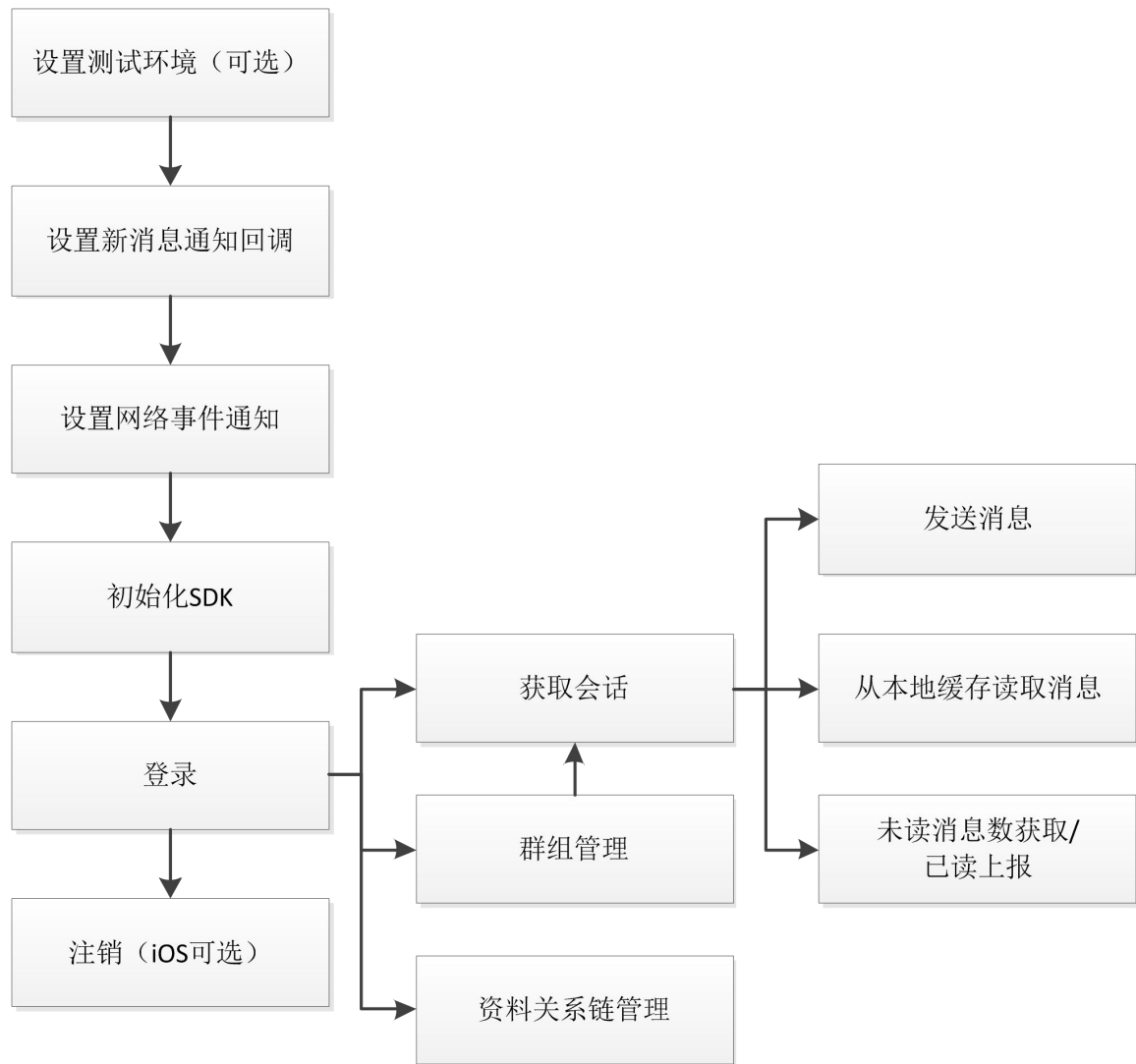
### 1.2.2. 调用顺序介绍

ImSDK 调用 API 需要遵循以下顺序，其余辅助方法需要在登录成功后调用。

步骤	对应函数	说明
初始化	<b>TIMEnv</b>	设置测试环境（可选）
	<b>TIMMode</b>	设置通信模式（根据需要）
	<b>TIMSetMessageCallBack</b>	设置消息回调
	<b>TIMSetConnCallBack</b>	设置链接通知回调
	<b>TIMInit</b>	初始化 SDK
登录	<b>TIMLogin</b>	登录
消息收发	<b>TIMGetConversation</b>	获取会话
	<b>SendMsg</b>	发送消息
群组管理	<b>TIMGroup*</b>	群组管理
资料关系链	<b>TIMFriend*</b>	资料关系链管理
注销	<b>TIMLogout</b>	注销（用户可选）

调用流程图：





## 2. 初始化

### 2.1. 设置测试环境

默认情况下，SDK 连接通讯后台的正式环境，此方法在大多数情况下可以忽略，不调用此方法，默认连接正式环境。如在特殊情况下需要连接测试环境，需要进行环境的设置，必须在通许管理器初始化 `initSdk` 之前调用：

原型：

```
void TIMSetEnv(intenv);
```

参数说明：

env:

- 0 正式环境（默认）
- 1 测试环境

示例：

```
TIMSetEnv (1); //设置为测试环境
```

示例中指定 SDK 连接测试环境。

## 2.2. 设置通信模式

设置接入模式(可选) 调用 init 初始化前, 可使用此接口指定接入模式。SDK 支持两种连接模式:

1 IM 模式

2 只登陆模式

当用户需要使用 IM 功能时, 例如收发 C2C 消息, 群组消息等相关 IM 功能, 需要设置 SDK 使用 IM 模式。当用户不需要 IM 功能, 只需要登入登出功能时, 可设置连接模式为只登陆模式

原型:

```
void TIMSetMode(int mode);
```

参数说明:

**mode** - 1 - IM 连接模式 / 0 或不调用此接口 - 只登陆模式

示例:

```
TIMSetMode (1); //设置为IM连接模式
```

## 2.3. 设置日志等文件路径（可选）

设置 SDK 日志和其他 SDK 所写文件的生成路径。调用 init 初始化前, 可使用此接口指定 SDK 日志文件路径。当 SDK 运行时, 如写文件失败会导致部分功能异常, 所以用户在某些特定不可写环境运行 SDK 时, 需先设置此路径为可写目录。如果不设置, 则默认为当前路径 (workspace)

原型:

```
void TIMSetPath(constchar* path);
```

参数说明:

**path** - 日志路径（绝对路径和相对路径皆可）以 '/' 或者 "\\" 结尾。

示例：

```
TIMSetPath("./"); //设置SDK日志文件为当前路径
```

## 2.4. 新消息通知

在多数情况下，用户需要感知新消息的通知，这时只需注册新消息通知回调。在用户登录状态下，会拉取离线消息，为了不漏掉消息通知，需要在登录之前注册新消息通知。

原型：

```
voidTIMSetMessageCallBack(TIMMessageCB *callback);

typedefvoid* TIMMessageHandle;
typedefvoid (*CBOnNewMessage) (TIMMessageHandlehandle, void*
data);
typedefstruct_TIMMessageCB_C
{
    CBOnNewMessageOnNewMessage;
    void* data;
}TIMMessageCB;
```

回调消息内容通过参数 **TIMMessageHandle** 传递，通过 **TIMMessageHandle** 可以获取消息和相关会话的详细信息，如消息文本，语音数据，图片等等，可参阅（消息解析）部分。

示例：

```
voidCBOnNewMessageImp(TIMMessageHandlehandle, void* data)
{
    TIMConversationHandleconv = CreateConversation();
    GetConversationFromMsg(conv, handle);
    autounread = GetUnreadMessageNum(conv);
    printf("MsgCallBack unread cnt: %llu\n", unread);
    for (inti = 0; i<GetElemCount(handle); i++)
    {
        printf("Msg type = <%d>\n", GetElemType(GetElem(handle,
i)));
    }
}
```

//设置消息监听器，收到新消息时，通过此监听器回调

```
voidSetNewMsgCallBack()
{
    staticTIMMessageCBcallback;
    callback.OnNewMessage = CBOOnNewMessageImp;
    TIMSetMessageCallBack(&callback);
}
```

示例中设置消息回调通知,并且在有新消息时直接打印消息,更详细的消息解析,可参阅（消息解析）部分。

## 2.5. 网络事件通知

可选设置,如果要用户感知是否已经连接服务器,需要设置此回调,用于通知调用者跟通讯后台链接的连接和断开事件,另外,如果断开网络,等网络恢复后会自动重连,自动拉取消息通知用户,用户无需关心网络状态,仅作通知之用。**注意:**这里的网络事件不表示用户本地网络状态,仅指明 SDK 是否与 IM 云 Server 连接状态。

原型:

```
voidTIMSetConnCallBack(TIMConnCB *callback);

typedefvoid (*CBConnOnConnected) (void* data);
typedefvoid (*CBConnOnDisconnected) (void* data);

typedefstruct_TIMConnCallBack_c
{
    CBConnOnConnected    OnConnected;
    CBConnOnDisconnected OnDisconnected;
    void* data;
}TIMConnCB;
```

示例:

以下示例监听网络事件

```
voidCBConnOnConnectedImp(void* data)
{
    printf("OnConnect! callback:%x", data);
}

voidCBConnOnDisconnectedImp(void* data)
{
    printf("OnDisconnect! callback:%x", data);
}
```

```
voids SetConnCallBack()
{
    staticTIMConnCBcallback;
    callback.OnConnected = CBConnOnConnectedImp;
    callback.OnDisconnected = CBConnOnDisconnectedImp;
    callback.data = &callback;
    TIMSetConnCallBack(&callback);
}
```

## 2.6. 用户状态变更

用户如果在其他终端登录，会被踢下线，这时会收到用户被踢下线的通知，出现这种情况常规的做法是提示用户进行操作（退出，或者再次把对方踢下线）。

原型：

```
voidTIMSetKickOfflineCallBack(TIMForceOfflineCB* callback);
```

```
typedefvoid (*CBKickOffline) (void* data);
typedefstruct_TIMKickOfflineCallBack_c
{
    CBKickOfflineOnKickOffline;
    void* data;
}TIMForceOfflineCB;
```

示例中当用户被踢下线时，收到回调后打印日志。

注意：

用户如果被踢，将无法登录，需要调用 logout 后再调用 login 强制登录。

## 2.7. 初始化

在使用 SDK 进一步操作之前，需要初始 SDK：

原型：

```
int TIMInit();
```

参数：

**Return:**

0 //操作成功

非0 //操作失败

示例：

```
TIMInit();
```

### 3. 登录

#### 3.1. 登录

用户登录腾讯后台服务器后才能正常收发消息，登录需要用户提供 `account_type`、`identifier`、`user_sig` 等信息，具体含义可参阅帐号文档。

注意：如果此用户在其他终端被踢，登录将会失败，返回错误码 6208，详情可参见：强制登录。开发者必须进行 6208 错误码判断，否则用户一旦被踢，将无法登录。关于被踢的详细描述，参见：用户状态变更。

登录为异步过程，通过回调函数返回是否成功，成功后方能进行后续操作。  
原型：

```
int TIMLogin(int sdk_app_id, const TIMUserInfo *tim_user,
             const char* user_sig, TIMCommCB *callback);
```

`sdk_app_id` - 用于标识接入 sdk 的应用 id

`tim_user` - 用户账号, 必须填写 `account_type` `identifier` `app_id_at_3rd`

`user_sig` - 用户 key

`callback` - 回调接口

在自有帐号情况下：`app_id_at_3rd` 字段与 `sdk_app_id` 相同，其他字段填写相应内容。

示例

```
void CBCommOnSuccessImp(void* data)
{
    printf("Success\n");
}

void CBCommOnErrorImp(int code, const char* desc, void* data)
{
    printf("Error! code = <%d> desc = <%s>", code, desc);
}
```

```

}

voidLogin()
{
    intsdk_app_id = 1104620500;
    TIMUserInfouser;
    user.account_type = "107";
    user.app_id_at_3rd = "1104620500";
    user.identifier = "c9_2";
    constchar* user_sig = "pass_word";

    TIMCommCBcallback;
    callback.OnSuccess = CCommOnSuccessImp;
    callback.OnError = CCommOnErrorImp;
    TIMLogin(sdk_app_id, &user, user_sig, &callback);
    SLEEP(1);
}

```

### 3.2. 登出

如用户主动注销或需要进行用户的切换，则需要调用注销操作：

原型：

```
voidTIMLogout(TIMCommCB *callback);
```

示例：

//注销登录

```

voidLogout()
{
    TIMCommCBcallback;
    callback.OnSuccess = CCommOnSuccessImp;
    callback.OnError = CCommOnErrorImp;
    TIMLogout(&callback);
    SLEEP(1);
}

```

注意：在需要切换帐号时，需要 **logout** 回调成功或者失败后才能再次 **login**，否则 **login** 可能会失败。

### 3.3. 强制登录

如用户被其他用户踢掉，服务器端用户状态将会被设置，再次登录将会失败，返回错误码 **6208**，需要调用 `logout` 接口再调用 `login` 进行强制登录，此操作将会踢掉目前在线的用户：

## 4. 消息收发

### 4.1. 消息发送

#### a. 会话获取

会话是指面向一个人或者一个群组的对话，通过与单个人或群组之间会话收发消息，发消息时首先需要先获取会话，获取会话需要指定会话类型（群组&单聊），以及会话对方标志（对方帐号或者群号）。获取会话由 `TIMGetConversation` 实现：

原型：

```
int TIMGetConversation(TIMConversationHandlehandle,
TIMConversationType type, const char* peer);
```

示例：

通过指定类型，获取相应会话。

以下示例获取对方 identifier 为"WIN\_001"的单聊会话：

```
void CreateC2CConversation()
{
    const char* peer = "WIN_001"; //your conv peer
    TIMConversationHandle conv = CreateConversation();
    int rt = TIMGetConversation(conv, kCnvC2C, peer);

    //conv 使用完毕以后资源释放
    DestroyConversation(conv);
}
```

以下示例获取群组 Id 为"TGID1JYSZEAEQ"的群聊会话：

```
void CreateGRPConversation()
{
    const char* peer = "TGID1JYSZEAEQ"; //your conv peer
    TIMConversationHandle conv = CreateConversation();
    int rt = TIMGetConversation(conv, kCnvGroup, peer);
}
```



```

//conv 使用完毕以后资源释放
DestroyConversation(conv);
}

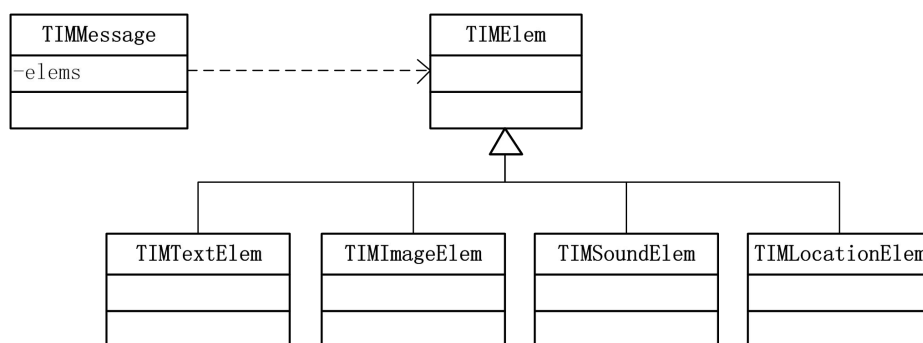
```

## b. 消息发送

通过 TIMManager 获取会话 TIMConversation 后，可发送消息和获取会话缓存消息；

ImSDK 中消息的解释可参阅（ImSDK 对象简介）。

ImSDK中的消息由TIMMessage表达(对应句柄 [TIMMessageHandle](#))， 一个 TIMMessage 由多个 TIMElem 组成（对应句柄 [TIMMsgTextElemHandle](#) ），每个 TIMElem 可以是文本和图片，也就是说每一条消息可包含多个文本和多张图片。



发消息通过 **SendMsg**实现。

原型：

```

voidSendMsg(TIMConversationHandleconv_handle,
TIMMessageHandlemsg_handle, TIMCommCB *callback);

```

参数说明：

**conv\_handle**- 会话句柄  
**msg\_handle** - 消息句柄  
**callback** - 消息回调

以下分别对发送文本和图片等类型的消息提供示例。

### 4.1.1. 文本消息发送

文本消息由 [TIMMsgTextElemHandle](#)

定义:

```
typedef void* TIMMsgTextElemHandle;
TIMMsgTextElemHandle CreateMsgTextElem();

void SetContent(TIMMsgTextElemHandlehandle, constchar*
content);
uint32_t GetContentLen(TIMMsgTextElemHandlehandle);
intGetContent(TIMMsgTextElemHandlehandle, char* content,
uint32_t* len);
```

content 传递需要发送的文本消息:

示例:

```
voidDemoSendMsg()
{

    constchar* peer = "c9_0"; //your conv peer
    TIMConversationHandleconv = CreateConversation();
    intrt = TIMGetConversation(conv, kCnvC2C, peer);

    TIMMessageHandlemsg = CreateTIMMessage();
    TIMMsgTextElemHandleelem = CreateMsgTextElem();
    charcontent[20] = {0};
    sprintf(content, "%s%s", "msg from ", peer);
    SetContent(elem, content);
    AddElem(msg, elem);

    TIMCommCBcallback;
    callback.OnSuccess = CBCommOnSuccessImp;
    callback.OnError = CBCommOnErrorImp;
    SendMsg(conv, msg, &callback);
    //....wait for callback
    SLEEP(1);

    DestroyConversation(conv);
    DestroyTIMMessage(msg);
    DestroyElem(elem);
}
```

#### 4.1.2. 图片消息发送

图片消息由 TIMMsgImageElemHandle定义, 其中需要设置发送图片的路径。消

息中的图片为TIMImageHandle。

```
int    GetImageElemUUID(TIMMsgImageElemHandlehandle, char* buf,
uint32_t* len);
void   SetImageElemPath(TIMMsgImageElemHandlehandle, constchar*
path, uint32_tlen);
int    GetImageElemPath(TIMMsgImageElemHandlehandle, char* path,
uint32_t* len);
void    SetImageCompressLeval(TIMMsgImageElemHandlehandle,
constTIM_IMAGE_COMPRESS_TYPEleval);
uint32_t GetImageNum(TIMMsgImageElemHandlehandle);
int     GetImages(TIMMsgImageElemHandlehandle, TIMImageHandle*
images, uint32_t* num);
```

参数说明:

**buf** : 图片 ID, 内部标识, 可用于外部缓存 key  
**path** : 要发送的图片路径  
**len** : 要发送的图片路径字符串长度  
**Leval** : 图片压缩等级, 详见 TIM\_IMAGE\_COMPRESS\_TYPE  
**images** : 发送成功后各种类型的图片

发送图片时, 只需要设置图片路径 path。发送成功后可通过 **GetImages** 获取所有图片类型, 目前只支持缩略图, 大图, 原图。

```
typedefenum_TIMImageType
```

```
{
    kOriginalImage = 0,
    kThumbImage,
    kLargeImage,
}TIMImageType;
```

```
typedefvoid* TIMImageHandle;
TIMImageType GetImageType(TIMImageHandlehandle);
uint32_t GetImageSize(TIMImageHandlehandle);
uint32_t GetImageHeight(TIMImageHandlehandle);
uint32_t GetImageWidth(TIMImageHandlehandle);
uint32_t GetImageURLLen(TIMImageHandlehandle);
int GetImageURL(TIMImageHandlehandle, char* url, uint32_t*
len);
int GetImageFile(TIMImageHandlehandle, char* filename,
TIMCommCB* cb);
```

参数说明:

type : 图片类型，原图、缩略图、大图，参见 TIM\_IMAGE\_TYPE  
size : 图片大小  
width : 图片宽  
height : 图片高  
url : 图片 URL，建议使用 **GetImageFile** 接口下载

**TIMImageHandle** 存储了图片列表的类型，大小，宽高信息，如需要图片二进制数据，需通过 **GetImageFile** 接口下载。

示例：

```
void DemoSendImage()
{
    constchar* peer = "c9_0"; //your conv peer
    TIMConversationHandle conv = CreateConversation();
    intrt = TIMGetConversation(conv, kCnvC2C, peer);

    TIMMessageHandle msg = CreateTIMMessage();
    TIMMsgImageElemHandle elem = CreateMsgImageElem();

    constchar* PATH = "./xxx/imgPath.jpg";
    SetImageElemPath(elem, PATH, strlen(PATH));
    AddElem(msg, elem);

    TIMCommCB callback;
    callback.OnSuccess = CBCommOnSuccessImp;
    callback.OnError = CBCommOnErrorImp;
    SendMsg(conv, msg, &callback);
    //....wait for callback
    SLEEP(1);

    DestroyConversation(conv);
    DestroyTIMMessage(msg);
    DestroyElem(elem);
}
```

示例中发送一张相对路径是 **"./xxx/imgPath.jpg"** 的图片。

#### 4.1.3. 表情消息发送

表情消息由 **TIMMsgFaceElemHandle** 定义，SDK 并不提供表情包，如果开发者有表情包，可使用 **index** 存储表情在表情包中的索引，由用户自定义，或者直接使用 **data** 存储表情二进制信息以及字符串 **key**，都由用户自定义，SDK 内部只做

透传：

```
typedef void* TIMMsgFaceElemHandle;
TIMMsgFaceElemHandle CreateFaceElemHandle();
int GetFaceElemIndex(TIMMsgFaceElemHandle handle);
uint32_t GetFaceElemDataLen(TIMMsgFaceElemHandle handle);
int GetFaceElemData(TIMMsgCustomElemHandle handle, char* date,
uint32_t* len);
void SetFaceElemIndex(TIMMsgFaceElemHandle handle, int index);
void SetFaceElemIndexData(TIMMsgCustomElemHandle handle,
const char* date, uint32_t len);
```

示例：

```
void DemoSendFace()
{
    const char* peer = "c9_0"; //your conv peer
    TIMConversationHandle conv = CreateConversation();
    int rt = TIMGetConversation(conv, kCnvC2C, peer);

    TIMMessageHandle msg = CreateTIMMessage();
    TIMMsgFaceElemHandle elem = CreateFaceElemHandle();

    SetFaceElemIndex(elem, 10);
    AddElem(msg, elem);

    TIMCommCB callback;
    callback.OnSuccess = CCommOnSuccessImp;
    callback.OnError = CCommOnErrorImp;
    SendMsg(conv, msg, &callback);
    //....wait for callback
    SLEEP(1);

    DestroyConversation(conv);
    DestroyTIMMessage(msg);
    DestroyElem(elem);
}
```

示例中发送了索引为 10 的表情。

#### 4.1.4. 语音消息发送

语音消息由 TIMSoundElem 定义，其中存储语音数据，语音数据需要提供时长信息，以秒为单位，注意，一条消息只能有一个语音 Elem，添加多条语音 Elem

时，AddElem 函数返回错误 1，添加不生效。

示例：

#### 4.1.5. 小文件消息发送

文件消息由 TIMFileElem 定义，其中存储文件数据，另外还可以提供额外的显示文件名信息，注意：一条消息只能添加一个文件 Elem，添加多个文件时，AddElem 函数返回错误 1。

属性说明：

data : 要发送的文件二进制数据  
filename : 文件名，SDK 不校验是否正确，只透传

示例：

```
void DemoSendFile()
{
    const char* peer = "c9_0"; //your conv peer
    TIMConversationHandle conv = CreateConversation();
    int rt = TIMGetConversation(conv, kCnvC2C, peer);

    TIMMessageHandle msg = CreateTIMMessage();
    TIMMsgFileElemHandle elem = CreateFileElemHandle();

    std::string file_name = "1.dat";
    std::fstream send_file(file_name.c_str(), std::fstream::in |
std::fstream::binary);
    std::string
file_data((std::istream_iterator<char>(send_file)),
std::istream_iterator<char>());

    SetFileElemFileName(elem, file_data.c_str(),
file_name.Length());
    SetFileElemData(elem, file_data.data(), file_data.Length());
    AddElem(msg, elem);

    TIMCommCB callback;
    callback.OnSuccess = CCommOnSuccessImp;
    callback.OnError = CCommOnErrorImp;
    SendMsg(conv, msg, &callback);
    //....wait for callback
    SLEEP(1);

    DestroyConversation(conv);
    DestroyTIMMessage(msg);
}
```

```

        DestroyElem(elem);
    }

```

#### 4.1.6. 自定义消息发送

自定义消息是指当内置的消息类型无法满足特殊需求，开发者可以自定义消息格式，内容全部由开发者定义，ImSDK 只负责透传。另外如果需要 iOS APNs 推送，还需要提供一段推送文本描述，方便展示。

自定义消息由 TIMCustomElem 定义，其中 data 存储消息的二进制数据，其数据格式由开发者定义，desc 存储描述文本。一条消息内可以有多个自定义 Elem，并且可以跟其他 Elem 混合排列，离线 Push 时叠加每个 Elem 的 desc 描述信息进行下发。

```

typedef void* TIMMsgCustomElemHandle;
TIMMsgCustomElemHandle CreateCustomElemHandle();
uint32_t GetCustomElemDataLen(TIMMsgCustomElemHandle handle);
int GetCustomElemData(TIMMsgCustomElemHandle handle, char*
data, uint32_t* len);
void SetCustomElemData(TIMMsgCustomElemHandle handle,
constchar* data, uint32_t len);
int GetCustomElemDesc(TIMMsgCustomElemHandle handle, char*
desc, uint32_t* len);
void SetCustomElemDesc(TIMMsgCustomElemHandle handle,
constchar* desc, uint32_t len);

```

参数说明：

data: 自定义消息二进制数据

desc: 自定义消息描述信息

示例：

```

void DemoSendCustomData()
{
    constchar* peer = "c9_0"; //your conv peer
    TIMConversationHandle conv = CreateConversation();
    int rt = TIMGetConversation(conv, kCnvC2C, peer);

    TIMMsgElemHandle msg = CreateTIMMessage();
    TIMMsgCustomElemHandle elem = CreateCustomElemHandle();

    FILE* file = fopen("custom.data", "rb");
    fseek(file, 0, SEEK_END);
}

```

```

uint32_t file_size = ftell(file);
fseek(file, 0, SEEK_SET);

void* buf = malloc(file_size);
fread(buf, 1, file_size, file);
fclose(file);

SetCustomElemData(elem, (constchar*)buf, file_size);
AddElem(msg, elem);

TIMCommCBcallback;
callback.OnSuccess = CCommOnSuccessImp;
callback.OnError = CCommOnErrorImp;
SendMsg(conv, msg, &callback);
//....wait for callback
SLEEP(1);

free(buf);
DestroyConversation(conv);
DestroyTIMMessage(msg);
DestroyElem(elem);
}

```

示例中读取文件内容当做消息内容，具体展示由开发者决定。

## 4.2. 从本地存储获取消息

ImSDK 会在本地进行消息存储，可通过 **GetMsgs** 获取，此方法为异步方法，需要通过设置回调得到消息数据。

原型：

```

voidGetMsgs(TIMConversationHandleconv_handle, intcount,
TIMMessageHandlelast_msg, TIMGetMsgCB * callback);

typedefvoid (*CBGetMsgOnSuccess) (TIMMessageHandle* handle_array,
intsize, void* data);
typedefvoid (*CBGetMsgOnError) (intcode, constchar* desc, void*
data);
typedefstruct_TIMGetMsgCallBack_c
{
    CBGetMsgOnSuccessOnSuccess;
    CBGetMsgOnErrorOnError;
    void* data;
}

```



```
}TIMGetMsgCB;
```

参数说明:

count - 从最后一条消息往前的消息数  
last\_msg - 已取得的最后一条消息  
callback - 回调, 参数中返回获取的消息列表

示例:

```
void DemoGetMessages()
{
    TIMConversationHandle conv = CreateConversation();
    constchar* peer = "c9_0"; //your conv peer
    TIMGetConversation(conv, kCnvC2C, peer);
    TIMGetMsgCBcallback;
    callback.OnSuccess = CBGetMsgOnSuccessImp;
    callback.OnError = CBGetMsgOnErrorImp;
    GetMsgs(conv, 3, NULL, &callback); // 获取最后三条消息
    //wait for callback.....

    DestroyConversation(conv);
}

void CBGetMsgOnSuccessImp(TIMMessageHandle* handle_array, int size,
void* data)
{
    for (inti = 0; i<size; i++)
    {
        printf("Msg time = <%u>", GetMsgTime(handle_array[i]));
    }
}

void CBGetMsgOnErrorImp(intcode, constchar* desc, void* data)
{
    printf("Error! code = <%d> desc = <%s>", code, desc);
}
```

对于图片等资源类消息, 消息体只会包含描述信息, 需要通过额外的接口下载数据, 可参与消息解析部分, 下载后的真实数据不会缓存, 需要调用方进行缓存。

#### 4.3. 获取所有会话

会话相关内容可参阅：（消息发送）的会话获取部分。另外，SDK 会保存用户的会话和消息，可以通过 `ConversationCount` 获取当前会话数量，从而得到所有本地会话：

原型：

```
uint64_t TIMGetConversationCount();
int TIMGetConversationByIndex(TIMConversationHandlehandle,
uint64_tindex);
```

示例：

```
voidDemoGetConversations()
{
    uint64_tcount = TIMGetConversationCount();
    for (uint64_t i = 0; i < count; i++)
    {
        //获得会话
        TIMConversationHandleconv = CreateConversation();
        TIMGetConversationByIndex(conv, i);
        /*处理会话
        ....
        */
        //释放会话
        DestroyConversation(conv);
    }
}
```

#### 4.4. 删除会话

会话相关内容可参阅：（4.1 消息发送的）会话获取部分。存在本地的最近联系会话，可以删除，需要注意的是，此处仅删除会话记录，会话中的消息并未删除。

原型：

```
bool TIMDeleteConversation(TIMConversationType type,
constchar* peer);
```

参数解释：

**type:**

会话类型，如果是单聊，填写 `TIM_C2C`，如果是群聊，填写 `TIM_GROUP`

**peer:**

会话标识, 单聊情况下, receiver 为对方用户 identifier, 群聊情况下, receiver 为群组 Id

示例:  
无

#### 4.5. 消息解析

收到消息后, 可用过 `GetElem` 从 `TIMMessageHandle` 中获取所有的 Elem 节点:

原型:

```
int GetElemCount(TIMMessageHandle handle);  
TIMMsgElemHandle GetElem(TIMMessageHandle handle, int index);
```

示例:

```
void CBOOnNewMessageImp(TIMMessageHandle handle, void* data)  
{  
    TIMConversationHandle conv = CreateConversation();  
    GetConversationFromMsg(conv, handle);  
    for (int i = 0; i < GetElemCount(handle); i++)  
    {  
        printf("type = <%d>\n", GetElemType(GetElem(handle, i)));  
    }  
}
```

##### 4.5.1. 基本属性解析

另外可通过 `TIMMessageHandle` 相关方法获取消息属性:

原型:

```
uint32_t GetMsgTime(TIMMessageHandle handle);  
int GetSender(TIMMessageHandle handle, char* buf,  
uint32_t* len);  
bool IsMsgRead(TIMMessageHandle handle);  
bool IsMsgFromSelf(TIMMessageHandle handle);  
int GetMsgStatus(TIMMessageHandle handle);
```

解释:

isSelf: 表示是否是自己发送的消息

timestamp: 标识消息的时间戳

sender: 消息发送方用户 identifier

status: 消息状态

#### 消息状态的解释:

消息发送后, 有三种状态, 发送中, 发送失败, 发送成功, 可通过成员 status 获取当前的消息状态:

```
// 消息发送中
#define TIM_MSG_STATUS_SENDING 1
// 消息发送成功
#define TIM_MSG_STATUS_SEND_SUCC 2
// 消息发送失败
#define TIM_MSG_STATUS_SEND_FAIL 3
// 消息被删除
#define TIM_MSG_STATUS_HAS_DELETED 4
```

#### 消息删除:

使用消息的 remove 方法可以在本地删除消息, 注意: 这里仅仅改变本地消息状态, 重新安装应用程序或删除本地数据后如果从漫游拉取, 消息状态恢复。remove 方法仅修改本地消息状态为 TIM\_MSG\_STATUS\_HAS\_DELETED, 并不真正删除消息, 需要界面进行消息的过滤。

#### 4.5.2. 图片消息解析

收到图片消息元素后, 可通过相应接口解析, 原型如下:

```
typedef void* TIMImageHandle;
TIMImageType GetImageType(TIMImageHandle handle);
uint32_t GetImageSize(TIMImageHandle handle);
uint32_t GetImageHeight(TIMImageHandle handle);
uint32_t GetImageWidth(TIMImageHandle handle);
uint32_t GetImageURLLen(TIMImageHandle handle);
int GetImageURL(TIMImageHandle handle, char* url, uint32_t* len);
int GetImageFile(TIMImageHandle handle, char* filename, TIMCommCB* cb);
```

示例:

```
void DemoReceiveImage(TIMMsgImageElemHandle image_elem)
{
    const uint32_t FILE_NAME_LEN = 100;
    uint32_t image_num = GetImageNum(image_elem);
```

```

TIMImageHandle* handles = newTIMImageHandle[image_num];
GetImages(image_elem, handles, &image_num);
for (uint32_t i = 0; i<image_num; i++)
{
    TIMImageHandlehandle = handles[i];
    printf("type: %d width :%u height : %u size :%u",
        GetImageType(handle),
        GetImageHeight(handle),
        GetImageWidth(handle),
        GetImageSize(handle));
    charfilename[FILE_NAME_LEN] = {0};
    sprintf(filename, "%d_pic.image", i);

    TIMCommCBcallback;
    callback.OnSuccess = CBCommOnSuccessImp;
    callback.OnError = CBCommOnErrorImp;
    GetImageFile(handle, filename, &callback);
    //....wait for callback
    SLEEP(1);
}
}

```

## 4.6. 系统解析

会话类型（TIMConversationType）除了 C2C 单聊和 Group 群聊以外，还有一种系统消息，系统消息不能由用户主动发送，是系统后台在相应的事件发生时产生的通知消息。系统消息目前分为两种，一种是关系链系统消息，一种是群系统消息。

关系链变更系统消息，当有用户加自己为好友，或者有用户删除自己好友的情况下，系统会发出变更通知，开发者可更新好友列表。相关细节可参阅[关系链变更系统通知](#) 部分。

当群资料变更，如群名变更或者群内成员变更，在群里会有系统发出一条群事件消息，开发者可在收到消息时可选择是否展示给用户，同时可刷新群资料或者群成员。详细内容可参阅[群事件消息](#)。

当被管理员踢出群组，被邀请加入群组等事件发生时，系统会给用户发出群系统消息，相关细节可参阅[群系统消息](#)。

## 5. 未读计数

### 5.1. 获取当前未读消息数量

可通过[GetUnreadMessageNum](#)方法获取当前会话中未读消息的数量：

原型:

```
uint64_tGetUnreadMessageNum(TIMConversationHandleconv_handle);
```

示例:

无

## 5.2. 已读上报

当用户阅读某个会话的数据后，需要进行会话消息的已读上报，SDK 根据会话中最后一条阅读的消息，设置会话中之前所有消息为已读。

原型:

```
voidSetReadMsg(TIMConversationHandleconv_handle,  
TIMMessageHandlelast_read_msg);
```

参数说明:

**last\_read\_msg:**

为当前会话中最后一条读过的消息，ImSDK 会把比 **last\_read\_msg** 时间更早的消息标记为已读消息。

示例 1:

无

C2C 和群组设置已读用法相同，区别在于会话类型。

## 6. 群组管理

### 6.1. 群组类型

开放群组分为三种内置类型：Private、Public、ChatRoom，三种类型的特点如下，用户可以根据不同的场景需要选择不同的群组类型：

**私有群 Private:** 无法被搜索到，可通过群成员邀请直接加入。类似微信群。

**公开群 Public:** 可通过群组 Id 搜索到，可申请加入，需要管理员审批。类似 QQ 群。

**聊天室 ChatRoom:** 可通过群组 Id 搜索到，可直接加入群聊，可翻看所有历史消息。

	控制维度	讨论组	群	聊天室
群基础资料	是否允许普通成员修改资料	○	×	×
	是否允许被搜索到	×	○	○
	是否允许申请入群	×	○	○
群关系链	是否允许直接拉人入群	○	×	×
	是否允许群主退群	○	×	×
	是否允许解散群	×	○	○
	是否允许设置管理员	×	○	○
	是否允许普通成员踢人	×	×	×
群消息	无限制历史消息	×	×	○
	成员变更通知	○	○	×
	是否在首条消息之后激活群	○	×	×

更详细的群资料控制如下：

控制字段	备注	讨论组	公开群	聊天室
是否允许普通成员修改资料	0：普通成员（非群主、管理员）不能修改群基础资料； 1：普通成员可以修改群名称、简介、公告、头像，但不能修改其他字段。	1	0	0
是否公开	0：非群成员无法获取该群的公开资料； 1：非群成员可以通过群 ID 获取该群的公开资料，例如群名称等。	0	1	1
是否允许需申请入群	0：任何申请加群的请求都不被允许； 1：依照申请加群选项（ApplyJoinOption）来决定后续操作。	0	1	1
是否允许直接拉人入群	0：群内任意成员可以不经其他用户同意便将其拉入群中；	1	0	0

	1: 任何邀请他人入群的操作都是不允许的（注 1）。			
是否允许群主退群	0: 群主不得退群，如果要退群，只能通过解散群的方式； 1: 群主可以退群，退群之后，群组将没有群主。	1	0	0
是否允许群主解散群	0: 群主不能解散群（但是 APP 管理员可以强制将其解散）； 1: 群主可以解散群。	0	1	1
是否允许设置管理员	0: 群主不能设置管理员； 1: 群主可以设置管理员。	0	1	1
是否允许普通成员踢人	0: 群内任意成员可以将任意其他成员踢出该群； 1: 普通群成员不具有踢人权限，仅群主、管理员才可以。	1	1	1

## 6.2. 群组消息

群组消息与 C2C 消息相同，仅在获取 Conversation 时的会话类型不同，可参照（消息发送）部分。

## 6.3. 创建群组

群组相关操作，需在用户登录成功后操作。

原型：

```
#define TIM_PRIVATE_GROUP "Private"
#define TIM_PUBLIC_GROUP "Public"
#define TIM_CHATROOM_GROUP "ChatRoom"
```

```
void TIMCreateGroup(constchar* group_type, constchar** members,
uint32_t members_count, constchar* group_name, TIMCreateGroupCB*
callback);
```

```
typedef void (*CBCreateGroupOnSuccess) (constchar* group_id, void*
data);
```

```
typedef void (*CBCreateGroupOnError) (intcode, constchar* desc,
void* data);
```

```
typedef struct TIMCallBack_CreateGroup
```

```
{
    CBCreateGroupOnSuccessOnSuccess;
    CBCreateGroupOnErrorOnError;
    void* data;
}TIMCreateGroupCB;
```



**group\_type**- 群组类型。**Private PublicChatRoom**这三种是内置的  
**members**- 群组成员 指定加入群组的成员，创建者默认加入，无需指定（群内最多 2000 人）  
**group\_name**- 群组名（最长 30 字节）

示例：

```
#define DEMO_MEM_COUNT 10
void DemoCreateGroup()
{
    //创建群组
    constchar** members = newconstchar*[DEMO_MEM_COUNT];
    members[0] = "c9_1";
    members[1] = "c9_2";
    members[2] = "c9_3";
    TIMCreateGroupCBcallback;
    callback.OnSuccess = CBCreateGroupOnSuccessImp;
    callback.OnError = CBCreateGroupOnErrorImp;
    TIMCreateGroup(TIM_PRIVATE_GROUP, members, 3, "c9_group_0",
&callback);
    //wait for callback
    SLEEP(1);

    deletemembers;
}

void CBCreateGroupOnSuccessImp(constchar* group_id, void* data)
{
    printf("Create Group Success! group_id = <%s>", group_id);
}

void CBCreateGroupOnErrorImp(intcode, constchar* desc, void* data)
{
    printf("CBCreateGroupOnErrorImp Error! code = <%d> desc = <%s>",
code, desc);
}
```

示例创建一个私有群组，并且把用户"**c9\_1**"等三人拉入群组，创建者默认加入群组，无需显式指定。

公有群和聊天室调用方式和参数相同，仅参数不同。

## 6.4. 加用户入群

**TIMInviteGroupMember**可以拉人进入群组，此操作只对私有群有效。

权限说明：

只有私有群可以拉用户入群；  
公有群和聊天室不能拉人入群；

此操作无需对方用户或者管理员同意，操作成功后被加入的用户就已经加入群组。所以此处需要开发者进行权限控制，或者使用 Server 回调，可由开发者 Server 进行加群事件审核，加强安全管理。

原型：

```
void TIMInviteGroupMember(constchar* groupid, constchar** members,
uint32_t members_count, TIMInviteGroupMemberCB* callback);
```

```
typedef void (*CBDeleteGroupMemberOnSuccess)
(TIMGroupMemberResultHandle* handle_array, uint32_t array_size,
void* data);
typedef void (*CBDeleteGroupMemberOnError) (int, constchar*, void*
data);
```

```
typedef struct _TIMCallBack_DeleteGroupMember
{
    CBDeleteGroupMemberOnSuccessOnSuccess;
    CBDeleteGroupMemberOnErrorOnError;
    void* data;
}TIMDeleteGroupMemberCB;
```

```
typedef TIMDeleteGroupMemberCB TIMInviteGroupMemberCB;
```

参数说明：

**groupid**: 群组 Id

**members**: 列表，加入群组用户列表

**callback**: 回调，

**OnSuccess** : **TIMGroupMemberResultHandle** 数组，返回成功加入群组的用户列表以及成功状态

**OnError**: 失败回调

其中 result 表示操作某个用户是否成功，具体定义如下：

```
int GetGroupMemberResultID(TIMGroupMemberResultHandle handle,
char* id, uint32_t* len);
uint32_t GetGroupMemberResult(TIMGroupMemberResultHandle handle)
;
```

参数说明:

//操作失败

```
#defineTIM_GROUP_MEMBER_STATUS_FAIL 0
```

//操作成功

```
#defineTIM_GROUP_MEMBER_STATUS_SUCC 1
```

Id: identifier

示例:

```
voidDemoInviteGroupMember()  
{  
    constchar** members = newconstchar*[DEMO_MEM_COUNT];  
    members[0] = "c9_1";  
    members[1] = "c9_2";  
    members[2] = "c9_3";  
  
    TIMInviteGroupMemberCBcallback;  
    callback.OnSuccess = CBIInviteGroupMemberOnSuccessImp;  
    callback.OnError = CBIInviteGroupMemberOnErrorImp;  
  
    TIMInviteGroupMember("test_groupid", members, 3, &callback);  
    //wait for callback  
    SLEEP(1);  
  
    deletemembers;  
}  
  
voidCBIInviteGroupMemberOnSuccessImp(TIMGroupMemberResultHandle  
* handle_array, uint32_tarray_size, void* data)  
{  
    charbuf[BUF_LEN] = {0};  
    for (uint32_ti = 0; i <array_size; i++)  
    {  
        uint32_tlen = BUF_LEN;  
        GetGroupMemberResultID(handle_array[i], buf, &len);  
        printf("Member OpenId = <%s>", buf);  
        printf("Member Result = <%u>",  
            GetGroupMemberResult(handle_array[i]));  
    }  
}  
  
voidCBIInviteGroupMemberOnErrorImp(intcode, constchar* desc,
```

```

void* data)
{
    printf("CBInviteGroupMemberOnErrorImp Error! code = <%d> desc
= <%s>", code, desc);
}

```

示例中邀请"c9\_1"等三人加入群组 Id"test\_groupid"，成功后返回操作列表以及成功状态。

## 6.5. 退出群组

群组成员可以主动退出群组。

权限说明：

- 对于私有群，全员可退出群组；
- 对于公有群和聊天室，群主不能退出；

原型：

```

voidTIMQuitGroup(constchar* groupid, TIMCommCB* callback);

```

参数说明：

**groupid**: 群组 Id

示例：

```

voidDemoQuiteGroup()
{
    TIMCommCBcallback;
    callback.OnSuccess = CBCommOnSuccessImp;
    callback.OnError = CBCommOnErrorImp;

    TIMQuitGroup("TGID1JYSZEAEQ", &callback);
    //wait for callback
    SLEEP(1);
}

```

示例中主动退出群组 "TGID1JYSZEAEQ"。

## 6.6. 删除群组成员

群组成员也可以删除其他成员，函数参数信息与加入群组相同。

权限说明:

对于私有群: 只有创建者可删除群组成员

对于公有群和聊天室: 只有管理员和群主可以踢人

原型:

```
voidTIMDeleteGroupMember(constchar* groupid, constchar** members,
uint32_t members_count, TIMDeleteGroupMemberCB* callback);
```

参数说明:

groupid: 群组 Id

members: 被操作的用户列表

callback: 回调

```
typedefvoid (*CBDeleteGroupMemberOnSuccess)
(TIMGroupMemberResultHandle* handle_array, uint32_t array_size,
void* data);
typedefvoid (*CBDeleteGroupMemberOnError) (int, constchar*, void*
data);
```

```
typedefstruct_TIMCallBack_DeleteGroupMember
{
    CBDeleteGroupMemberOnSuccessOnSuccess;
    CBDeleteGroupMemberOnErrorOnError;
    void* data;
}TIMDeleteGroupMemberCB;
```

[TIMGroupMemberResultHandle](#) 含义参照加用户入群。

示例:

```
voidDemoDeleteGroupMember()
{
    constchar** members = newconstchar*[DEMO_MEM_COUNT];
    members[0] = "WIN_001";
    members[1] = "WIN_002";
    members[2] = "WIN_003";

    TIMDeleteGroupMemberCBcallback;
    callback.OnSuccess = CBDeleteGroupMemberOnSuccessImp;
    callback.OnError = CBDeleteGroupMemberOnErrorImp;

    TIMDeleteGroupMember("TGID1JYSZAEQ", members, 3, &callback);
```

```

        //wait for callback
        SLEEP(1);
        deletemembers;
    }

#define BUF_LEN 100
void CBDeleteGroupMemberOnSuccessImp(TIMGroupMemberResultHandle
* handle_array, uint32_tarray_size, void* data)
{
    charbuf[BUF_LEN] = {0};
    for (uint32_ti = 0; i < array_size; i++)
    {
        uint32_tlen = BUF_LEN;
        GetGroupMemberResultID(handle_array[i], buf, &len);
        printf("Member OpenId = <%s>", buf);
        printf("Member Result = <%u>",
            GetGroupMemberResult(handle_array[i]));
    }
}

void CBDeleteGroupMemberOnErrorImp(intcode, constchar* desc,
void* data)
{
    printf("CBDeleteGroupMemberOnErrorImp Error! code = <%d> desc
= <%s>", code, desc);
}

```

示例中把好友 "WIN\_001"等 3 人从群组 @ "TGID1JYSZEAEQ" 中删除, 执行成功后返回操作列表以及操作状态。

## 6.7. 获取群成员列表

**GetGroupMembers** 方法可获取群内成员列表。

权限说明:

任何群组类型都可以获取成员列表;

原型:

```

void TIMGetGroupMembers(constchar* groupid,
TIMGetGroupMemberInfoCB *cb);

typedef void (*CBGetGroupMemberInfoOnSuccess)
(TIMGroupMemberInfoHandle* handle_array, uint32_tarray_size,

```

```

void* data);
typedef void (*CBGetGroupMemberInfoOnError) (intcode, constchar*
desc, void* data);
typedef struct _TIMCallBack_GetGroupMemberInfo
{
    CBGetGroupMemberInfoOnSuccessOnSuccess;
    CBGetGroupMemberInfoOnErrorOnError;
    void* data;
}TIMGetGroupMemberInfoCB;

//群成员ID
intGetGroupMemberID(TIMGroupMemberInfoHandlehandle, char* id,
uint32_t* len);
//加入群组时间
uint32_tGetGroupMemberInfoJoinTime(TIMGroupMemberInfoHandlehan
dle);
//成员类型
uint32_tGetGroupMemberInfoRole(TIMGroupMemberInfoHandlehandle)
;

```

参数说明:

groupid: 群组 Id

示例:

```

voidDemoGetGroupMemberList()
{
    TIMGetGroupMemberInfoCBcallback;
    callback.OnSuccess = CBGetGroupMemberInfoOnSuccessImp;
    callback.OnError = CBGetGroupMemberInfoOnErrorImp;
    TIMGetGroupMembers("TGID1JYSZAEQ", &callback);
    //wait for callback
    SLEEP(1);
}

voidCBGetGroupMemberInfoOnSuccessImp(TIMGroupMemberInfoHandle*
handle_array, uint32_tarray_size, void* data)
{
    charbuf[BUF_LEN] = {0};
    for (uint32_t i = 0; i<array_size; i++)
    {
        uint32_t len = BUF_LEN;
        GetGroupMemberID(handle_array[i], buf, &len);
    }
}

```

```

        printf("ID = <%s>\n", buf);
    }
}

void CBGetGroupMemberInfoOnErrorImp(intcode, constchar* desc,
void* data)
{
    printf("CBGetGroupMemberInfoOnErrorImp Error! code = <%d> desc
= <%s>", code, desc);
}

```

示例中获取群 @TGID1JYSZEAQ 的成员列表，handle\_array 存储成员的相关信息。

## 6.8. 获取加入的群组列表

通过 GetGroupList 可以获取当前用户加入的所有群组：

权限说明：

此接口可以获取自己所加入的群列表，返回 group\gropuName\groupType 信息，想要获取详细信息，可调用 GetGroupInfo 获取详细资料。

原型：

```

void TIMGetGroupList(TIMGetGroupListCB *callback);

typedef void (*CBGetGroupListOnSuccess) (TIMGroupBaseInfoHandle*
handle_array, uint32_t array_size, void* data);
typedef void (*CBGetGroupListOnError) (intcode, constchar* desc,
void* data);

typedef struct _TIMCallBack_GetGroupList
{
    CBGetGroupListOnSuccessOnSuccess;
    CBGetGroupListOnErrorOnError;
    void* data;
}TIMGetGroupListCB;

int GetGroupBaseInfoID(TIMGroupBaseInfoHandle handle, char* id,
uint32_t* len);
int GetGroupBaseInfoName(TIMGroupBaseInfoHandle handle, char* name,
uint32_t* len);
int GetGroupBaseInfoType(TIMGroupBaseInfoHandle handle, char* type,
uint32_t* len);

```



示例：

```
void DemoGetGroupList()
{
    TIMGetGroupListCBcallback;
    callback.OnSuccess = CBGetGroupListOnSuccessImp;
    callback.OnError = CBGetGroupListOnErrorImp;
    TIMGetGroupList(&callback);
    //wait for callback
    SLEEP(1);
}

void CBGetGroupListOnSuccessImp(TIMGroupBaseInfoHandle*
handle_array, uint32_t array_size, void* data)
{
    char buf[BUF_LEN] = {0};
    for (uint32_t i = 0; i < array_size; i++)
    {
        TIMGroupBaseInfoHandle handle = handle_array[i];
        uint32_t len = BUF_LEN; GetGroupBaseInfoName(handle, buf,
        &len);
        printf("GroupName = <%s>\n", buf);
        len = BUF_LEN; GetGroupBaseInfoID(handle, buf, &len);
        printf("GroupId = <%s>\n", buf);
        len = BUF_LEN; GetGroupBaseInfoType(handle, buf, &len);
        printf("GroupType = <%s>\n", buf);
    }
}

void CBGetGroupListOnErrorImp(intcode, constchar* desc, void*
data)
{
    printf("CBGetGroupListOnErrorImp Error! code = <%d> desc =
    <%s>", code, desc);
}
```

示例中获取群组列表，并打印群组 Id，群类型（Private/Public/ChatRoom）以及群名。

## 6.9. 获取群组资料

`TIMGetGroupDetailInfo`方法可以获取群组资料。

权限说明：

注意：获取群组资料接口只能由群成员调用，非群成员无法通过此方法获取资料。

群资料信息由TIMGroupDetailInfoHandle  
定义：

```
// 群组 Id
intGetGroupDetailInfoID(TIMGroupDetailInfoHandlehandle, char* id,
uint32_t* len);
// 群名
intGetGroupDetailInfoName(TIMGroupDetailInfoHandlehandle, char*
name, uint32_t* len);
// 创建人
intGetGroupDetailInfoOwner(TIMGroupDetailInfoHandlehandle,
char* owner, uint32_t* len);
// 群公告
intGetGroupDetailInfoNotification(TIMGroupDetailInfoHandlehand
le, char* buf, uint32_t* len);
// 群简介
intGetGroupDetailInfoIntroduction(TIMGroupDetailInfoHandlehand
le, char* buf, uint32_t* len);
intGetGroupDetailInfoType(TIMGroupDetailInfoHandlehandle, char*
type, uint32_t* len);
// 群创建时间
uint32_t
GetGroupDetailInfoCreateTime(TIMGroupDetailInfoHandlehandle);
// 最近一次修改资料时间
uint32_tGetGroupDetailInfoLastInfoTime(TIMGroupDetailInfoHandl
ehandle);
// 最近一次发消息时间
uint32_tGetGroupDetailInfoLastMsgTime(TIMGroupDetailInfoHandle
handle);
// 群成员数量
uint32_tGetGroupDetailInfoMemberNum(TIMGroupDetailInfoHandleha
ndle);
uint32_tGetGroupDetailInfoMaxMemberNum(TIMGroupDetailInfoHandl
ehandle);
```

通过 TIMGetGroupDetailInfo可获取群组资料：

定义：

```
voidTIMGetGroupDetailInfo(constchar** groupids,
uint32_tgroupid_count, TIMGetGroupDetailInfoCB * cb);
```

参数说明:

```
typedef void (*CBGetGroupDetailInfoOnSuccess)
(TIMGroupDetailInfoHandle* handle_array, uint32_t array_size,
void* data);
typedef void (*CBGetGroupDetailInfoOnError) (intcode, const char*
desc, void* data);
```

```
typedef struct TIMCallback_GetGroupDetailInfo
{
    CBGetGroupDetailInfoOnSuccessOnSuccess;
    CBGetGroupDetailInfoOnErrorOnError;
    void* data;
}TIMGetGroupDetailInfoCB;
```

groupids: 需要获取资料的群组 ID 列表

groupid\_count: 群个数

示例:

```
void DemoGetGroupDetailInfo()
{
    const char** groups = new const char*[DEMO_MEM_COUNT];
    groups[0] = "TGID1JYSZAEQ";
    TIMGetGroupDetailInfoCB callback;
    callback.OnSuccess = CBGetGroupDetailInfoOnSuccessImp;
    callback.OnError = CBGetGroupDetailInfoOnErrorImp;
    TIMGetGroupDetailInfo(groups, 1, &callback);
    //wait for callback
    SLEEP(1);

    delete []groups;
}

void CBGetGroupDetailInfoOnSuccessImp(TIMGroupDetailInfoHandle*
handle_array, uint32_t array_size, void* data)
{
    for (uint32_t i = 0; i < array_size; i++)
    {
        TIMGroupDetailInfoHandle handle = handle_array[i];
        printf("group :%d createtime : %u", i,
GetGroupDetailInfoCreateTime(handle));
    }
}
```

```
void CBGetGroupDetailInfoOnErrorImp(int code, const char* desc,
void* data)
{
    printf("CBGetGroupDetailInfoOnErrorImp Error! code = <%d> desc
= <%s>", code, desc);
}
```

示例中获取群组 "TGID1JYSZEAEQ" 的详细信息

## 6.10. 修改群资料

通过 TIMModifyGroupDatilInfoV2 可以修改群组信息:

权限说明:

对于公有群和聊天室, 只有群主或者管理员可以修改群名、群简介、群公告、群头像、加群选项;

对于私有群, 任何人可修改群资料;

```
void TIMModifyGroupDatilInfoV2(TIMModifyGroupBaseInfoOptionHand
leopt, TIMCommCB * callback);
```

参数说明:

opt: 修改设置

```
typedef enum _T_TIMModifyGroupFlag
{
    kModifyNone = 0x00,
    kModifyGroupName = 0x01,
    kModifyNotification = 0x01 << 1,
    kModifyIntroduction = 0x01 << 2,
    kModifyFaceUrl = 0x01 << 3,
    kModifyAddOption = 0x01 << 4, // 申请加群选项: 0 禁止加
入; 1 需要审批; 2 随意加入;
    kModifyMaxMmeberNum = 0x01 << 5, // 最大群成员数量 (仅 ROOT
可以操作)
}TIMModifyGroupFlag;
```

```
int SetGroupId4ModifyGroupBaseInfoOptionHandle(TIMModifyGroupBa
seInfoOptionHandlehandle, char* group_id);
int SetFlag4ModifyGroupBaseInfoOptionHandle(TIMModifyGroupBaseI
nfoOptionHandlehandle, TIMModifyGroupFlagflag);
int SetGroupName4ModifyGroupBaseInfoOptionHandle(TIMModifyGroup
```

```

BaseInfoOptionHandlehandle, char* group_name);
intSetNotification4ModifyGroupBaseInfoOptionHandle(TIMModifyGroupBaseInfoOptionHandlehandle, char* notification);
intSetIntroduction4ModifyGroupBaseInfoOptionHandle(TIMModifyGroupBaseInfoOptionHandlehandle, char* introduction);
intSetFaceUrl4ModifyGroupBaseInfoOptionHandle(TIMModifyGroupBaseInfoOptionHandlehandle, char* face_url);
intSetAddOption4ModifyGroupBaseInfoOptionHandle(TIMModifyGroupBaseInfoOptionHandlehandle, uint32_tadd_option);

```

**TIMModifyGroupBaseInfoOptionHandle** 属性说明:

**ID** : 设置欲修改群组 ID （必选）

**FLAG**: 设置修改标志（必选）

接口以位操作方式定义 **FLAG**。调用者拼接需要修改的标志。

所修改属性的值是否有效与相应 **FLAG** 是否设置一一对应。

**GroupName**: 修改后的群组名

**Notification**: 修改后的群公告

**Introduction**: 修改后的群简介

**FaceUrl**: 修改后的群头像 URL

**AddOption**: 修改后的申请加群选项

0禁止加入;

1需要审批;

2随意加入;

示例:

```

voidDemoModifyGroupInfo()
{
    TIMModifyGroupBaseInfoOptionHandlehd =
        CreateModifyGroupBaseInfoOptionHandle();
    SetGroupId4ModifyGroupBaseInfoOptionHandle(hd,
"TGID1JYSZEAQ");
    TIMModifyGroupFlagflag = TIMModifyGroupFlag(kModifyGroupName
| kModifyIntroduction);
    SetFlag4ModifyGroupBaseInfoOptionHandle(hd, flag);
    SetGroupName4ModifyGroupBaseInfoOptionHandle(hd,
"ModifyGroupName");
    SetIntroduction4ModifyGroupBaseInfoOptionHandle(hd, "this is
a group");

    TIMCommCBcallback;
    callback.OnSuccess = CBCommOnSuccessImp;
    callback.OnError = CBCommOnErrorImp;

    ModifyGroupDatilInfoV2(hd, &callback);
}

```

```

    //wait for callback
    SLEEP(1);

    DestroyModifyGroupMemberInfoOptionHandle(hd);
}

```

示例修改群 "TGID1JYSZEAEQ" 的名字为 "ModifyGroupName", 修改群简介为 "this is a group"

## 6.11. 获取群组公开资料

当用户不在群组时, 需要通过接口 **TIMGetGroupPublicInfo** 获取群的公开资料。获取结果为 **TIMGroupDetailInfoHandle** 结构, 该结构只会有公开资料。

群资料信息由 **TIMGroupDetailInfoHandle** 定义:

属性如下

```

group;           // 群组 Id
groupName;       // 群名
groupType;       // 群组类型
owner;           // 创建人
createTime;      // 群创建时间
memberNum;       // 群成员数量
introduction;    // 群简介

```

**TIMGroupDetailInfoHandle** 的以上字段为公开字段, 可以获取, 其他字段为空。通过 **TIMGetGroupPublicInfo** 可获取群组资料:

定义:

```

void TIMGetGroupPublicInfo(constchar** groupids,
constuint32_tgroup_num,      constTIMGetGroupBaseInfoFlagflag,
TIMGroupCustomInfoHandlecustom, TIMGetGroupDetailInfoCB
*callback);

```

参数说明:

**groupids**: 群 id 数组, 需要获取资料的群组列表

**flag** : 获取公开资料标志位, 详见定义。

**custom**: 接口预留, 暂不支持。

**callback**: 回调

示例:

无

## 6.12. 解散群组

通过 `TIMDeleteGroup` 可以解散群组。

权限说明：

对于私有群，任何人都无法解散群组

对于公有群和聊天室，群主可以解散群组：

原型：

```
void TIMDeleteGroup(constchar* groupid, TIMCommCB * callback);
```

参数说明：

`groupid` : 群组 Id

示例：

无

## 6.13. 群事件消息

当有用户被邀请加入群组，或者有用户被移出群组时，群内会产生有提示消息，调用方可以根据需要展示给群组用户，或者忽略。提示消息使用一个特殊的 Elem 标识，通过新消息回调返回消息（参见新消息通知）：

如下图中，展示一条修改群名的事件消息：



消息原型:

```
typedef enum _E_TIM_GROUPTIPS_TYPE
{
    TIM_GROUP_TIPS_TYPE_INVITE           = 0x01, //邀请加入群
    TIM_GROUP_TIPS_TYPE_QUIT_GRP         = 0x02, //退出群
    TIM_GROUP_TIPS_TYPE_KICKED           = 0x03, //踢出群
    TIM_GROUP_TIPS_TYPE_SET_ADMIN         = 0x04, //设置管理员
    TIM_GROUP_TIPS_TYPE_CANCEL_ADMIN      = 0x05, //取消管理员
    TIM_GROUP_TIPS_TYPE_INFO_CHANGE      = 0x06, //群资料变更
    TIM_GROUP_TIPS_TYPE_MEMBER_INFO_CHANGE = 0x07, //群成员资料
    变更
}E_TIM_GROUPTIPS_TYPE;

//群Tips类型
typedef void* TIMMsgGroupTipsElemHandle;
E_TIM_GROUPTIPS_TYPE GetGroupTipsInfoType(TIMMsgGroupTipsElemHandle handle);
//群名
int GetGroupTipsInfoGroupName(TIMMsgGroupTipsElemHandle handle,
char* group_name, uint32_t * len);
//操作人ID
int GetGroupTipsInfoOperatorID(TIMMsgGroupTipsElemHandle handle,
char* id, uint32_t * len);
//被操作人列表
uint32_t
GetGroupTipsInfoUsersNum(TIMMsgGroupTipsElemHandle handle);
int GetGroupTipsInfoUsers(TIMMsgGroupTipsElemHandle handle,
TIMGroupTipsUserInfoHandle* handles, uint32_t* num);
//被操作人ID
typedef void* TIMGroupTipsUserInfoHandle;
int GetGroupTipsUserInfoID(TIMGroupTipsUserInfoHandle handle,
char* id, uint32_t * len);
//群信息变更列表
uint32_t GetGroupTipsInfoGroupChangeInfoNum(TIMMsgGroupTipsElemHandle handle);
int GetGroupTipsInfoGroupChangeInfo(TIMMsgGroupTipsElemHandle handle,
TIMGroupChangeInfoHandle* handles, uint32_t* num);
//群成员变更列表
uint32_t GetGroupTipsInfoMemberChangeInfoNum(TIMMsgGroupTipsElemHandle handle);
int GetGroupTipsInfoMemberChangeInfo(TIMMsgGroupTipsElemHandle handle,
TIMGroupMemberInfoChangeHandle* handles, uint32_t* num);
```



调用方可选择是否予以展示，以及如何展示。

消息元素属性说明：

群 Tips 类型：见枚举定义

操作人：引起发该 Tips 的用户

被操作人列表：该操作影响到的用户

该属性目前只包含被操作人的 ID

群信息变更列表：群相关的变更信息

群信息变更元素属性说明：

群信息变更类型：定义如下

```
#defineTIM_GROUP_INFO_CHAGE_TYPE_GROUP_NAME 0x1 //群名更改
#defineTIM_GROUP_INFO_CHAGE_TYPE_INTRODUCTION 0x2 //群简介更改
#defineTIM_GROUP_INFO_CHAGE_TYPE_NOTIFACTION 0x3 //群通知更改
#defineTIM_GROUP_INFO_CHAGE_TYPE_FACE_URL 0x4 //群头像更改
#defineTIM_GROUP_INFO_CHAGE_TYPE_OWNER 0x5 //群主更改
```

变更的群信息：在各种群信息变更类型下对应的信息

群员变更列表：TIM\_GROUP\_TIPS\_TYPE\_MEMBER\_INFO\_CHANGE 类型时对应的信息 包括的属性有：

群成员禁言时间

示例：

```
voidDemoGetGroupTips(TIMMsgElemHandlehandle)
{
    constuint32_tMAX_CHANGE_NUM = 100; //实际根据情况动态创建
    TIMElemType type = GetElemType(handle);
    if (kElemGroupTips != type)
    {
        printf("not group tips!");
        return;
    }
    E_TIM_GROUP_TIPS_TYPEtips_type=GetGroupTipsInfoType(handle);
    switch (tips_type)
    {
        caseTIM_GROUP_TIPS_TYPE_INVITE:
        caseTIM_GROUP_TIPS_TYPE_KICKED:
        caseTIM_GROUP_TIPS_TYPE_QUIT_GRP:
        {
            printf("invited kicked or quit");
        }
        break;
        caseTIM_GROUP_TIPS_TYPE_SET_ADMIN:
```

```

    caseTIM_GROUP_TIPS_TYPE_CANCEL_ADMIN:
    {
        printf("set or cancel ADMIN");
    }
    break;
    caseTIM_GROUP_TIPS_TYPE_INFO_CHANGE:
    {
        uint32_tchange_num=GetGroupTipsInfoGroupChangeInfoNum(handle);
        TIMGroupChangeInfoHandle* change_infos =
        newTIMGroupChangeInfoHandle[change_num];
        GetGroupTipsInfoGroupChangeInfo(handle, change_infos,
        &change_num);
        for (uint32_ti = 0; i<change_num; i++)
        {
            printf("type :%d",GetGroupChangeInfoType(change_infos[i]));
        }
        delete []change_infos;
        break;
    }
    caseTIM_GROUP_TIPS_TYPE_MEMBER_INFO_CHANGE:
    {
        uint32_tchange_num =
        GetGroupTipsInfoMemberChangeInfoNum(handle);
        TIMGroupMemberInfoChangeHanlde* change_infos =
        newTIMGroupMemberInfoChangeHanlde[change_num];
        GetGroupTipsInfoMemberChangeInfo(handle, change_infos,
        &change_num);
        for (uint32_ti = 0; i<change_num; i++)
        {
            printf("shutup time :%d",
            TIMGetGroupMemberChangeInfoShutTime(change_infos[i]));
        }
        delete []change_infos;
        break;
    }
}
}
}

```

### 6.13.1. 用户被邀请加入群组

当有用户被邀请加入群组时，群组内会由系统发出通知，开发者可选择展示样式。收到的消息 type 为 TIM\_GROUP\_TIPS\_TYPE\_INVITE。

该类型下可获取参数说明：

Type : TIM\_GROUP\_TIPS\_TYPE\_INVITE

操作人 : 邀请人

群名 : 群名

被操作人列表: 被邀请入群的用户列表

### 6.13.2. 用户退出群组

当有用户主动退群时，群组内会由系统发出通知。收到的消息 type 为 TIM\_GROUP\_TIPS\_TYPE\_QUIT\_GRP。

该类型下可获取参数说明：

Type : TIM\_GROUP\_TIPS\_TYPE\_QUIT\_GRP

操作人 : 退出用户 identifier

群名 : 群名

### 6.13.3. 用户被踢出群组

当有用户被踢出群组时，群组内会由系统发出通知。收到的消息 type 为 TIM\_GROUP\_TIPS\_TYPE\_KICKED。

TIMGroupTipsElem 参数说明：

Type : TIM\_GROUP\_TIPS\_TYPE\_KICKED

操作人 : 踢人的用户 identifier

群名 : 群名

被操作人列表: 被踢用户列表

### 6.13.4. 被设置/取消管理员

当有用户被设置为管理员或者被取消管理员身份时，群组内会由系统发出通知。收到的消息 type 为 TIM\_GROUP\_TIPS\_TYPE\_SET\_ADMIN 和 TIM\_GROUP\_TIPS\_TYPE\_CANCEL\_ADMIN 。

TIMGroupTipsElem 参数说明：

type: 设置: TIM\_GROUP\_TIPS\_TYPE\_SET\_ADMIN

取消: TIM\_GROUP\_TIPS\_TYPE\_CANCEL\_ADMIN

操作人 : 操作用户 identifier

群名 : 群名

被操作人列表: 被设置/取消管理员身份的用户列表

### 6.13.5. 群资料变更

当群资料变更，如群名、群简介等，会有系统消息发出，可更新相关字段展示，或者选择性把消息展示给用户。

TIMGroupTipsElem 参数说明：

type: TIM\_GROUP\_TIPS\_TYPE\_INFO\_CHANGE  
操作人 : 操作用户 identifier  
群名 : 群名  
群信息变更列表: 群变更的具体资料信息，为TIMGroupChangeInfoHandle数组

TIMGroupChangeInfoHandle 原型：

```
#defineTIM_GROUP_INFO_CHAGE_TYPE_GROUP_NAME 0x1
#defineTIM_GROUP_INFO_CHAGE_TYPE_INTRODUCTION 0x2
#defineTIM_GROUP_INFO_CHAGE_TYPE_NOTIFACTION 0x3
#defineTIM_GROUP_INFO_CHAGE_TYPE_FACE_URL 0x4
#defineTIM_GROUP_INFO_CHAGE_TYPE_OWNER 0x5

intGetGroupChangeInfoType(TIMGroupChangeInfoHandlehandle);
uint32_tGetGroupChangeInfoLen(TIMGroupChangeInfoHandlehandle);
uint32_tGetGroupChangeInfo(TIMGroupChangeInfoHandlehandle,
char* info, uint32_t *len);
```

说明：

Type: 变更类型  
groupName: 变更后的群名，如果没有变更则为 NULL  
introduction: 变更后的群简介，如果没有变更则为 NULL  
notification: 变更后的群公告，如果没有变更则为 NULL  
faceUrl: 变更后的群头像 URL，如果没有变更则为 NULL  
owner: 变更后的群主，如果没有变更则为 NULL

### 6.13.6. 群成员资料变更

当群成员的资料变更时，会有系统消息发出，可更新相关字段展示，或者选择性把消息展示给用户。

TIMGroupTipsElem 参数说明：

type: TIM\_GROUP\_TIPS\_TYPE\_MEMBER\_INFO\_CHANGE  
操作人 : 操作用户 identifier  
群名: 群名

群成员资料变更列表：变更的群成员的具体资料信息，为  
TIMGroupMemberInfoChangeHandle 列表

TIMGroupMemberInfoChangeHandle 原型：

```
typedef void* TIMGroupMemberInfoChangeHandle;  
int GetGroupMemberChangeInfoID(TIMGroupMemberInfoChangeHandle handle, char* id, uint32_t * len);  
uint32_t TIMGetGroupMemberChangeInfoShutTime(TIMGroupMemberInfoChangeHandle handle);
```

说明：

identifier: 变更的用户 identifier  
shutupTime: 被禁言的时间

## 6.14. 群系统消息

当有用户申请加群等事件发生时，管理员会收到邀请加群系统消息，用户可根据情况接受请求或者拒绝，相应的消息通过群系统消息展示给用户。

群系统消息类型定义：

```
//群系统消息类型  
typedef enum _E_TIM_GROUP_SYSTEM_TYPE  
{  
    TIM_GROUP_SYSTEM_ADD_GROUP_REQUEST_TYPE = 0x01, //申请加群请求（只有管理员会收到）  
    TIM_GROUP_SYSTEM_ADD_GROUP_ACCEPT_TYPE = 0x02, //申请加群被同意（只有申请人能够收到）  
    TIM_GROUP_SYSTEM_ADD_GROUP_REFUSE_TYPE = 0x03, //申请加群被拒绝（只有申请人能够收到）  
    TIM_GROUP_SYSTEM_KICK_OFF_FROM_GROUP_TYPE = 0x04, //被管理员踢出群（只有被踢的人能够收到）  
    TIM_GROUP_SYSTEM_DELETE_GROUP_TYPE = 0x05, //群被解散（全员能够收到）  
    TIM_GROUP_SYSTEM_CREATE_GROUP_TYPE = 0x06, //创建群消息（创建者能够收到）  
    TIM_GROUP_SYSTEM_INVITED_TO_GROUP_TYPE = 0x07, //邀请加群（被邀请者能够收到）  
    TIM_GROUP_SYSTEM_QUIT_GROUP_TYPE = 0x08, //主动退群（主动退群者能够收到）  
    TIM_GROUP_SYSTEM_GRANT_ADMIN_TYPE = 0x09, //设置管
```

```

    理员(被设置者接收)
        TIM_GROUP_SYSTEM_CANCEL_ADMIN_TYPE = 0x0a, //取消管
    理员(被取消者接收)
        TIM_GROUP_SYSTEM_REVOKE_GROUP_TYPE = 0x0b, //群已被
    回收(全员接收)
}E_TIM_GROUP_SYSTEM_TYPE;

//操作类型
E_TIM_GROUP_SYSTEM_TYPE GetGroupReportType(TIMMsgGroupReportElemHandlehandle);
//群组Id
int GetGroupReportID(TIMMsgGroupReportElemHandlehandle, char* id,
uint32_t* len);
//操作人
int GetGroupReportOperatorID(TIMMsgGroupReportElemHandlehandle,
char* id, uint32_t* len);
//操作理由
int
GetGroupReportRemarkInfoLen(TIMMsgGroupReportElemHandlehandle)
;
int GetGroupReportRemarkInfo(TIMMsgGroupReportElemHandlehandle,
char* remark_info, uint32_t* len);

//审批入群申请，目前只对申请加群消息生效
int HandleJoinRequest(TIMMsgGroupReportElemHandlehandle, int flag,
TIMCommCB* cb);

```

参数说明:

Flag 0x00:拒绝入群 0x01:同意入群

示例:

示例中收到群系统消息，如果是入群申请，默认同意，如果是群解散通知，打印信息。其他类型消息解析方式相同。

#### 6.14.1. 申请加群消息

当有用户申请加群时，群管理员会收到  
TIM\_GROUP\_SYSTEM\_ADD\_GROUP\_REQUEST\_TYPE 类型的消息，带有群  
组 Id，申请人和申请理由，如果管理员同意，可调用 **HandleJoinRequest** 方法

#### 6.14.2. 申请加群同意/拒绝消息

当有管理员同意加群请求时，申请人会收到

TIM\_GROUP\_SYSTEM\_ADD\_GROUP\_ACCEPT\_TYPE 类型的消息，当管理员拒绝时，收到TIM\_GROUP\_SYSTEM\_ADD\_GROUP\_REFUSE\_TYPE 类型消息，带有群组 Id，申请人和同意/拒绝理由。

#### 6.14.3. 被管理员踢出群组

当用户被管理员踢出群组时，申请人会收到TIM\_GROUP\_SYSTEM\_KICK\_OFF\_FROM\_GROUP\_TYPE 类型的消息，带有群组 Id，操作人信息。

#### 6.14.4. 群被解散

当群被解散时，全员会收到TIM\_GROUP\_SYSTEM\_DELETE\_GROUP\_TYPE 类型的消息，带有群组 Id，操作人信息。

#### 6.14.5. 创建群消息

当群创建时，初始成员会收到群的创建消息，类型为TIM\_GROUP\_SYSTEM\_CREATE\_GROUP\_TYPE 的消息，带有群组 Id，操作人信息。

#### 6.14.6. 邀请加群

当用户被邀请加入群组时，该用户会收到邀请消息，类型为TIM\_GROUP\_SYSTEM\_INVITED\_TO\_GROUP\_TYPE，带有群组 Id，操作人信息。（注意：创建群组时初始成员无需邀请即可入群）。

审批同意入群，可调用 **HandleJoinRequest**。

#### 6.14.7. 主动退群

当用户主动退出群组时，该用户会收到退群消息，只有退群的用户可以收到，类型为TIM\_GROUP\_SYSTEM\_QUIT\_GROUP\_TYPE，带有群组 Id，操作人信息。

#### 6.14.8. 设置/取消管理员

当用户被设置为管理员或者被取消管理员身份时，会分别收到类型为TIM\_GROUP\_SYSTEM\_GRANT\_ADMIN\_TYPE 和TIM\_GROUP\_SYSTEM\_CANCEL\_ADMIN\_TYPE 的消息，带有群组 Id，操作人信息。

#### 6.14.9. 群被回收

当群组被系统回收时，全员可收到群组被回收消息，消息类型为：TIM\_GROUP\_SYSTEM\_REVOKE\_GROUP\_TYPE，带有群组 Id，操作人信息。

## 7. 用户资料

IM 云提供的帐号分为两种，一种是独立帐号体系，由用户自己保存并更新用户资料和关系链信息，另一种是托管模式，此模式下用户可完全不用搭建自己的后台服务，把用户的资料和关系链托管在 IM 云服务。

如果开发者选择资料和关系链托管，需要调用本章的接口进行资料和关系链的操作。

### 7.1. 设置自己昵称

可通过 **TIMSetNickName** 方法设置用户自己的昵称，昵称最大为 64 字节：

原型：

```
voidTIMSetNickName(char* nick, uint32_t len, TIMCommCB * cb);
```

参数说明：

**nick** 要设置的昵称

**len** 昵称长

示例：

无

### 7.2. 设置好友验证方式

可通过 **TIMSetAllowType** 方法设置好友验证方式，有同意任何人加好友、拒绝任何人加好友、需要验证三种方式，用户可根据需要设置其中一种：

原型：

```
voidTIMSetAllowType(E_TIMFriendAllowType type, TIMCommCB * cb);
```

参数说明：

**type** 好友验证方式，详见

```
typedefenum_E_TIMFriendAllowType
```

```
{
    TIM_FRIEND_ALLOW_ANY      = 0, //同意任何用户加好友
    TIM_FRIEND_DENY_ANY       = 1, //拒绝任何人加好友
    TIM_FRIEND_NEED_CONFIRM   = 2, //需要验证
}E_TIMFriendAllowType;
```



示例：

无

此示例中设置了自己的好友验证方式为需要验证，此时如果有用户申请加好友，会收到加好友的系统通知（详见关系链变更系统通知）。

### 7.3. 获取自己的资料

可通过 **TIMGetSelfProfile**方法获取用户自己的资料：

原型：

```
void TIMGetSelfProfile(TIMGetSelfProfileCallback* cb);

typedef void (*CBGetSelfProfileCallbackOnSuccess)
(TIMSelfProfileHandle* handles, uint32_t num, void* data);
typedef void (*CBGetSelfProfileCallbackOnError) (intcode,
constchar* desc, void* data);
    typedef struct _T_TIMGetSelfProfileCallback
    {
        CBGetSelfProfileCallbackOnSuccessOnSuccess;
        CBGetSelfProfileCallbackOnErrorOnError;
        void* data;
    }TIMGetSelfProfileCallback;

typedef void* TIMSelfProfileHandle;
TIMSelfProfileHandleCloneSelfProfileHandle(TIMSelfProfileHandle
ehandle);
voidDestroySelfProfileHandle(TIMSelfProfileHandle handle);
intGetNickName4SlefProfileHandle(TIMSelfProfileHandle handle,
char* buf, uint32_t* len);
E_TIMFriendAllowTypeGetAllowType4SlefProfileHandle(TIMSelfProf
ileHandle handle);
```

参数说明：

成功回调，返回 **TIMSelfProfileHandle** 数组

失败回调，会返回错误码和错误信息，详见错误码表

属性说明：

identifier : 自己的用户标识  
nickname : 自己的昵称  
allowType : 好友验证方式

示例：

无

## 7.4. 设置好友的备注

可通过 `TIMSetFriendRemark` 方法设置好友备注：

原型：

```
void TIMSetFriendRemark(constchar* id, constchar* remark,
constuint32_t remark_len, TIMCommCB * cb);
```

参数说明：

<code>id</code>	要设置备注的好友标识
<code>remark</code>	要设置的备注
<code>cb</code>	回调

示例：

## 7.5. 获取好友的资料

### 7.5.1. 获取好友资料的固定字段

可通过 `TIMGetFriendProfile` 方法获取好友的资料：

原型：

```
void TIMGetFriendProfile(char** id, uint32_t num,
TIMGetFriendProfileCallback* cb);
```

```
typedef void (*CBGetFriendProfileCallbackOnSuccess)
(TIMFriendProfileHandle* handles, uint32_t num, void* data);
typedef void (*CBGetFriendProfileCallbackOnError) (intcode,
constchar* desc, void* data);
```

```
typedef struct _TIMGetFriendProfileCallback
{
    CBGetFriendProfileCallbackOnSuccessOnSuccess;
    CBGetFriendProfileCallbackOnErrorOnError;
    void* data;
}TIMGetFriendProfileCallback;
```

```
typedef void* TIMFriendProfileHandle;
TIMFriendProfileHandleCloneFriendProfileHandle(TIMFriendProfil
eHandle handle);
voidDestroyFriendProfileHandle(TIMFriendProfileHandle handle);
intGetID4FriendProfileHandle(TIMFriendProfileHandle handle,
char* id, uint32_t* len);
```

```

intGetNickName4FriendProfileHandle(TIMFriendProfileHandle
handle, char* buf, uint32_t* len);
intGetRemark4FriendProfileHandle(TIMFriendProfileHandle handle,
char* remark, uint32_t* len);
E_TIMFriendAllowTypeGetAllowType4FriendProfileHandle(TIMFriend
ProfileHandle handle);

```

参数说明:

**id** 用户列表

**num** 用户个数

成功回调, 返回 **TIMFriendProfileHandle** 数组, 包含用户资料

失败回调, 会返回错误码和错误信息, 详见错误码表

属性说明:

**identifier** : 好友的 identifier

**nickname** : 好友昵称

**remark** : 好友备注

**allowType** : 好友验证方式

示例:

无

### 7.5.2. 获取好友资料的部分字段

可通过**TIMGetPartialProfile**方法获取部分好友的资料:

原型:

```

voidTIMGetPartialProfile(char** id, uint32_tnum,
TIMGetProfileOptionHandlehandle, TIMGetFriendProfileCallback*
cb);

```

```

typedefenum_E_TIMGetProfileType
{
    TIM_GET_PROFILE_None            = 0,
    TIM_GET_PROFILE_Nick            = 0x01,          //获取昵称
    TIM_GET_PROFILE_AllowType       = 0x01 << 1,    //获取ALLOWTYPE
    TIM_GET_PROFILE_FaceUrl         = 0x01 << 2,    //获取FaceURL
}E_TIMGetProfileType;

```

```

typedefvoid* TIMGetProfileOptionHandle;
TIMGetProfileOptionHandleCreateGetProfileOptionHandle();
voidDestroyGetProfileOptionHandle(TIMGetProfileOptionHandlehan
dle);

```

```
voidSetFlag4GetProfileOptionHandle(TIMGetProfileOptionHandlehandle, E_TIMGetProfileType);
voidSetCustomInfo4GetProfileOptionHandle(TIMGetProfileOptionHandlehandle, TIMProfileCustomInfoHandlecustom_info);
```

参数说明：

**id** 用户列表

**num** 用户个数

**handle** 获取资料选项

成功回调，返回 **TIMFriendProfileHandle** 数组，包含用户资料

失败回调，会返回错误码和错误信息，详见错误码表

示例：

```
voidDemoGetPartialProfile()
{
    char** members = newchar*[DEMO_MEM_COUNT];
    members[0] = "user1";
    members[1] = "user2";

    TIMGetProfileOptionHandleopt =
CreateGetProfileOptionHandle();
    SetFlag4GetProfileOptionHandle(opt,
E_TIMGetProfileType(TIM_GET_PROFILE_Nick |
TIM_GET_PROFILE_FaceUrl));
    TIMProfileCustomInfoElemHandlecustom_elem =
CreateProfileCustomInfoElemHandle();
    char* key = "my_key";
    SetKey4ProfileCustomInfoElemHandle(custom_elem, key,
strlen(key)); // 设置查询key为“my_key”的profile自定义字段
    TIMProfileCustomInfoHandlecustom_info =
CreateProfileCustomInfoHandle();
    SetProfileCustomInfo(custom_info, &custom_elem, 1);
    SetCustomInfo4SetProfileOptionHandle(opt, custom_info);

    TIMGetFriendProfileCallbackcallback;
    callback.OnSuccess = CBGetFriendProfileCallbackOnSuccessImp;
    callback.OnError = CBGetFriendProfileCallbackOnErrorImp;
    TIMGetPartialProfile(members, 2, opt, &callback);
    //wait for callback
    SLEEP(1);

    DestroyProfileCustomInfoHandle(custom_info);
    DestroyProfileCustomInfoElemHandle(custom_elem);
```

```

        DestroyGetProfileOptionHandle(opt);
    }

void CBGetFriendProfileCallbackOnSuccessImp(TIMFriendProfileHandle* handles, uint32_t num, void* data)
{
    char buf[BUF_LEN] = {0};
    for(uint32_t i = 0; i < num; i++)
    {
        auto handle = handles[i];
        uint32_t id_len = BUF_LEN; GetID4FriendProfileHandle(handle, buf, &id_len); printf("friend id: %s", buf);
        uint32_t nick_len = BUF_LEN; GetNickName4FriendProfileHandle(handle, buf, &nick_len); printf("nick name: %s", buf);

    }
}

void CBGetFriendProfileCallbackOnErrorImp(int code, const char* desc, void* data)
{
    printf("CBGetFriendProfileCallbackOnErrorImp Error! code = %d desc = %s", code, desc);
}

```

## 7.6. 添加好友

通过 **TIMAddFriend** 方法可以批量添加好友，目前所能支持的最大好友列表为 3000 个：

原型：

```
void TIMAddFriend(TIMAddFriendHandle* handles, uint32_t num,
TIMAddFriendCallback * cb);

typedef void* TIMAddFriendHandle;
TIMAddFriendHandleCreateAddFriendHandle();
void DestroyAddFriendHandle(TIMAddFriendHandle handle);
void SetID4AddFriendHandle(TIMAddFriendHandle handle, char* id);
    //用户昵称 最大 96 字节
void SetNickName4AddFriendHandle(TIMAddFriendHandle handle, char*
nick_name, uint32_t len);
    //好友备注 最大 120 字节
void SetRemark4AddFriendHandle(TIMAddFriendHandle handle, char*
remark, uint32_t len);
    //好友来源
void SetAddSource4AddFriendHandle(TIMAddFriendHandle handle,
char* source);

/**
```

参数说明：

用户列表，**TIMAddFriendHandle** 数组  
成功回调，返回 **TIMFriendResult\***数组，包含用户添加结果  
失败回调，会返回错误码和错误信息，详见错误码表

属性说明：

**identifier** : 用户标识  
**remark** : 添加成功后给用户的备注信息，最大 96 字节  
**addWording** : 添加请求说明，最大 120 字节，如果用户设置为添加好友需要审核，对方会收到此信息并决定是否通过。  
**addSource** : 添加来源，固定字符串，在页面上申请，留空表示未知来源

返回码说明：

成功回调会返回操作用户的**TIMFriendResultHandle**结果数据，添加好友的错误码如下：

```
//操作成功
#define TIM_FRIEND_STATUS_SUCC 0
//被加好友在自己的黑名单中
```

```

#define TIM_ADD_FRIEND_STATUS_IN_SELF_BLACK_LIST      30515
//被加好友设置为禁止加好友
#define TIM_ADD_FRIEND_STATUS_FRIEND_SIDE_FORBID_ADD  30516
//好友数量已满
#define TIM_ADD_FRIEND_STATUS_SELF_FRIEND_FULL       30519
//已经是好友
#define TIM_ADD_FRIEND_STATUS_ALREADY_FRIEND         30520
//已被被添加好友设置为黑名单
#define TIM_ADD_FRIEND_STATUS_IN_OTHER_SIDE_BLACK_LIST 30525
//等待好友审核同意
#define TIM_ADD_FRIEND_STATUS_PENDING                30539

```

开发者可根据对应情况提示用户。

示例：

无

### 7.7. 删除好友

可通过**TIMDelFriend**方法可以批量删除好友：

原型：

```

voidTIMDelFriend(inttype, char** id, uint32_t num,
TIMDelFriendCallback * cb);

/**
 * 删除好友
 *
 * @param delType 删除类型（单向好友、双向好友）
 * @param users 要删除的用户列表 NSString* 列表
 * @param succ 成功回调，返回 TIMFriendResult* 列表
 * @param fail 失败回调
 *
 * @return 0 发送请求成功
 */
-(int) DelFriend:(TIMDelFriendType)delType users:(NSArray*) users
succ:(TIMFriendSucc)succ fail:(TIMFail)fail;

@end

```

参数说明：

**type**删除类型，可选择删除双向好友或者单向好友 **1-单向好友 2-双向好友**

**id** 要删除的用户 id

成功回调，返回 **TIMFriendResultHandle**数组，包含用户添加结果

失败回调，会返回错误码和错误信息，详见错误码表

返回码说明：

成功回调会返回操作用户的 **TIMFriendResultHandle** 结果数据，删除好友的错误码如下：

```
//操作成功
#define TIM_FRIEND_STATUS_SUCC          0
//删除好友时对方不是好友
#define TIM_DEL_FRIEND_STATUS_NO_FRIEND 30515
```

开发者可根据情况提示用户。

示例：

无

## 7.8. 获取所有好友

可通过 **TIMGetFriendList**方法可以获取所有好友：

原型：

```
void TIMGetFriendList(TIMGetFriendListCallback * cb);

typedef void (*CBGetFriendListCallbackOnSuccess)
(TIMFriendListElemHandle* handles, uint32_t num, void* data);
typedef void (*CBGetFriendListCallbackOnError) (intcode,
constchar* desc, void* data);
typedef struct _TIMGetFriendListCallback
{
    CBGetFriendListCallbackOnSuccessOnSuccess;
    CBGetFriendListCallbackOnErrorOnError;
    void* data;
}TIMGetFriendListCallback;
```

参数说明：

成功回调，返回好友列表**TIMFriendListElemHandle**数组，只包含identifier, nickname, remark 三个属性

失败回调，会返回错误码和错误信息，详见错误码表

**TIMFriendListElemHandle** 属性说明：

identifier	: 自己的用户标识
nickname	: 自己的昵称
remark	: 用户备注



其他属性需要通过拉取好友的详细资料获得

示例:

无

## 7.9. 同意/拒绝 好友申请

可通过 `TIMFriendResponse` 方法可以获取所有好友:

原型:

```
void TIMFriendResponse(TIMFriendResponseHandle* handles, uint32_t num, TIMFriendResponseCallback * cb);
```

```
typedef void* TIMFriendResponseHandle;
```

```
    //用户ID
```

```
void SetID4FriendResponseHandle(TIMFriendResponseHandle handle, char* id);
```

```
    //响应类型
```

```
void SetResponseType4FriendResponseHandle(TIMFriendResponseHandle handle, E_TIMFriendResponseType);
```

```
typedef enum _E_TIMFriendResponseType
```

```
{
```

```
    TIM_FRIEND_RESPONSE_AGREE                = 0, //同意加好友 (建立单向好友)
```

```
    TIM_FRIEND_RESPONSE_AGREE_AND_ADD        = 1, //同意加好友并加对方为好友 (建立双向好友)
```

```
    TIM_FRIEND_RESPONSE_REJECT               = 2, //拒绝对方好友请求
```

```
}E_TIMFriendResponseType;
```

```
typedef void (*CBFriendResponseCallbackOnSuccess)
```

```
(TIMFriendResultHandle* handles, uint32_t num, void* data);
```

```
typedef void (*CBFriendResponseCallbackOnError) (intcode,
```

```
constchar* desc, void* data);
```

```
    typedef struct _T_TIMFriendResponseCallback
```

```
    {
```

```
        CBFriendResponseCallbackOnSuccessOnSuccess;
```

```
        CBFriendResponseCallbackOnErrorOnError;
```

```
        void* data;
```

```
    }TIMFriendResponseCallback;
```

参数说明:

`handles` 响应的用户列表, `TIMFriendResponseHandle` 数组

成功回调，返回 `TIMFriendResultHandle` 数组  
 失败回调，会返回错误码和错误信息，详见错误码表  
 返回码说明：  
 成功回调会返回操作用户的 `TIMFriendResultHandle` 结果数据，处理用户请求的错误码如下：

```
//操作成功
#define TIM_FRIEND_STATUS_SUCC                                0
//响应好友申请时有效：对方没有申请过好友
#define TIM_RESPONSE_FRIEND_STATUS_NO_REQ                    30614
//响应好友申请时有效：自己的好友满
#define TIM_RESPONSE_FRIEND_STATUS_SELF_FRIEND_FULL          30615
//响应好友申请时有效：好友已经存在
#define TIM_RESPONSE_FRIEND_STATUS_FRIEND_EXIST              30617
//响应好友申请时有效：对方好友满
#define TIM_RESPONSE_FRIEND_STATUS_OTHER_SIDE_FRIEND_FULL    30630
```

开发者可根据情况提示用户。

## 7.10. 关系链变更系统通知

`TIMMsgElemHandle`中Elem类型 `kElemFriendChange`为关系链变更系统消息：

原型和属性：

```
//关系链变更消息类型
typedef enum_E_TIM_SNS_SYSTEM_TYPE
{
    TIM_SNS_SYSTEM_ADD_FRIEND = 0x01, //添加好友系统通知
    TIM_SNS_SYSTEM_DEL_FRIEND = 0x02, //删除好友系统通知
    TIM_SNS_SYSTEM_ADD_FRIEND_REQ = 0x03, //好友申请系统通知
    TIM_SNS_SYSTEM_DEL_FRIEND_REQ = 0x04, //删除未决请求通知
}E_TIM_SNS_SYSTEM_TYPE;

typedef void* TIMMsgSNSChangeInfoHandle;
intGetID4SNSChangeInfoHandle(TIMMsgSNSChangeInfoHandle handle,
char* id, uint32_t * len);
intGetNick4SNSChangeInfoHandle(TIMMsgSNSChangeInfoHandle handle,
char* nick, uint32_t * len);
intGetAddWord4SNSChangeInfoHandle(TIMMsgSNSChangeInfoHandle
handle, char* word, uint32_t * len);
intGetAddSource4SNSChangeInfoHandle(TIMMsgSNSChangeInfoHandle
handle, char* source, uint32_t * len);

E_TIM_SNS_SYSTEM_TYPEGetType4SNSSystemElem(TIMMsgSNSSystemElem
Handlehandle);
```

```
uint32_t
GetChangeInfoNum4SNSSystemElem(TIMMsgSNSSystemElemHandle
handle);
intGetChangeInfo4SNSSystemElem(TIMMsgSNSSystemElemHandleelem_h
andle, TIMMsgSNSChangeInfoHandle* info_handle, uint32_t* num);
```

元素属性说明:

类型 : 变更消息子类型 `E_TIM_SNS_SYSTEM_TYPE`  
 用户信息列表 : 信息列表 `TIMMsgSNSChangeInfoHandle` 数组

用户信息属性:

ID : 用户 identifier  
 AddWord : 申请理由  
 AddSource : 申请来源, 申请时填写, 由系统页面分配的固定字符串  
 Nick : 用户昵称

### 7.10.1. 添加好友系统通知

当两个用户成为好友时, 两个用户均可收到添加好友系统消息:

触发时机:

当自己的关系链变更, 增加好友时, 收到消息 (如果已经是单向好友, 关系链没有变更的一方不会收到)

参数说明:

类型 : `TIM_SNS_SYSTEM_ADD_FRIEND`  
 用户信息列表 : 成为好友的用户列表

用户信息属性: 说明:

ID : 用户 identifier

### 7.10.2. 删除好友系统通知

当两个用户解除好友关系时, 会收到删除好友系统消息:

触发时机:

当自己的关系链变更, 删除好友时, 收到消息 (如果删除的是单向好友, 关系链没有变更的一方不会收到)

参数说明:

类型 : `TIM_SNS_SYSTEM_DEL_FRIEND`  
 用户信息列表 : 删除好友的用户列表

用户信息属性说明:

ID : 用户 identifier

### 7.10.3. 好友申请系统通知

当申请好友时对方需要验证, 自己和对方会收到好友申请系统通知:

触发时机：

当申请好友时对方需要验证，自己和对方会收到好友申请系统通知，对方可选择同意或者拒绝，自己不能操作，只做信息同步之用。

参数说明：

类型 : TIM\_SNS\_SYSTEM\_ADD\_FRIEND\_REQ

用户信息列表 : 申请的好友列表

用户信息属性说明：

ID : 用户 identifier

AddWord : 申请理由

AddSource : 申请来源，申请时填写，由系统页面分配的固定字符串

#### 7.10.4. 删除未决请求通知

触发时机：

当申请对方为好友，申请审核通过后，自己会收到删除未决请求消息，表示之前的申请已经通过。

参数说明：

类型 : TIM\_SNS\_SYSTEM\_DEL\_FRIEND\_REQ

用户信息列表 : 删除未决请求的好友列表

用户信息属性说明：

ID : 用户 identifier

#### 7.11. 好友资料变更系统通知

TIMMsgElemHandle中Elem类型 kElemProfileChange为关系链变更系统消息：

原型：

```
typedef enum_E_TIM_PROFILE_SYSTEM_TYPE
{
    TIM_PROFILE_SYSTEM_FRIEND_PROFILE_CHANGE = 0x01, //好友资料
    变更
}E_TIM_PROFILE_SYSTEM_TYPE;
```

```
E_TIM_PROFILE_SYSTEM_TYPEGetType4ProfileProfileSystemElemHandle(
TIMMsgProfileSystemElemHandlehandle);
intGetID4ProfileProfileSystemElemHandle(TIMMsgProfileSystemElem
Handlehandle, char* id, uint32_t* len);
intGetNick4ProfileProfileSystemElemHandle(TIMMsgProfileSystemE
lemHandlehandle, char* nick, uint32_t* len);
```

属性说明:

[E\\_TIM\\_PROFILE\\_SYSTEM\\_TYPE](#) 资料变更类型

ID 资料变更的用户

Nick 昵称变更, 注意, 如果昵称没有变更, 为 **NULL**