## IMSDK 快速集成文档文档

### 一、安卓端

一．安卓集成说明：

1. 本sdk为 原生java代码编程。
2. 本sdk因为分别有长链接及短链接，集成了2个不同的网络框架，okHttp 框架主要用来实现短链接接口请求，netty框架主要实现 tcp的长链接接口的收发信息。
3. 安卓接口说明：
4. 获取token api/application/token（已添加到 sdk）直接调用 RetrofitUtils.instance.api.login(userId)
5. 根据获取的token登陆

defaultLogicProcess?.login

1. 发信息

defaultLogicProcess?.sendMsg(chatMessage) 填写所需的参数

sendSuccessCallback 发送的回调

1. 收到信息

handlerMessage handlerMessage 收到的信息回调

e.退出登陆

defaultLogicProcess?.logout()

1. 安卓集成指南：
2. 将我们提供的sdk的jar 复制在项目的lisb 并右键点击本jar 后 点击add as library即可自动集成
3. 检查sdk是否完成集成
4. 先初始化sdk

private var defaultLogicProcess: DefaultLogicProcess? = null

defaultLogicProcess?.initLogicProcess(defaultLogicProcess)

1. 安卓sdk详细接口介绍：

1. 通过自定义的ID获取token接口（sdk已集成 接口请求）

RetrofitUtils.instance.api.login(userId)//userId为自行定义

1. 根据已获取的token进行登陆操作

//登陆im 接口地址及im服务器的长链接  
defaultLogicProcess?.initLogicProcess(defaultLogicProcess)  
defaultLogicProcess?.login(  
 ApiConfig.HOST\_JAVA\_IM, //im服务器的接口地址 例如："192.168.5.4"

ApiConfig.HOST\_PORT,//im服务器的聊天服务器地址端口 例如：1001

userId,//自行定义的userId  
 token//通过接口获取到的token  
)

1. 发送信息详细参数说明

private var chatMessage = ChatMessage()

chatMessage.*from* = userId //自定义的id 发送方的uesrId

chatMessage.*content* = binding.sideContext?.*text*.*toString*().*trim*()//文本内容  
chatMessage.*msg\_type* = 1//消息类型 可自定义  
chatMessage.*type* = 1//协议类型  
chatMessage.*to* = binding.sideId?.*text*.*toString*().*trim*() //发给对方的userId  
defaultLogicProcess?.sendMsg(chatMessage) 然后塞进 消息实体并发送

成功发送后 日志显示：

【IM】接收到服务器ACK包:{"ack":1,"data":"{\"c\":\"123\",\"mt\":1,\"t\":1645412058483}","from":"123","no":"01064e4c-1bb3-480a-bee8-61edae55053f","to":"10086","type":2}

1. 接收信息包括已发送消息，接收到的信息

【IM】接收到服务器ACK包:{"ack":1,"data":"{\"c\":\"123\",\"mt\":1,\"t\":1645412058483}","from":"123","no":"01064e4c-1bb3-480a-bee8-61edae55053f","to":"10086","type":2}

接收到的消息解析

*/\*\* 协议类型 \*/*private int type;  
*/\*\* 发送方的ID \*/*private String from;  
*/\*\* 发送目标ID \*/*private String to;  
*/\*\* 消息类型 \*/*private int msg\_type;  
*/\*\* 消息发送时间 无需设置,发送消息时SDK自动覆盖 \*/*private long time;  
*/\*\* 消息实际内容 \*/*private String content;  
*/\*\* 消息的唯一标志 \*/*private String msgId;  
*/\*\*  
 \* String类型的扩展字段  
 \*/*private String ext1;  
private String ext2;  
private String ext3;  
*/\*\*  
 \* Integer类型的扩展字段 不使用基本类型,是因为转json可以通过设置将为null值不输出,但是int则不行  
 \*/*private Integer ext4;  
private Integer ext5;

private var defaultLogicProcess: DefaultLogicProcess? = null

defaultLogicProcess = object : DefaultLogicProcess() {  
 override fun sendFailedCallback(chatMessage: ChatMessage?) {//发送失败回调  
 }  
  
 override fun sendSuccessCallback(chatMessage: ChatMessage?) {//发送成功的回调 }  
  
 override fun handlerMessage(chatMessage: ChatMessage?) {//收到信息的回调

}  
}

\*特别提示：

请先获取token，再登陆im服务器，并按照所需的参数填写。

### 二、IOS端

sdk代码为OC原生，flutter使用需要创oc原生项目参考

将在Build Settings->ENABLE\_BITCODE设置为NO

Build Settings->OTHER\_LINKER\_FLAGS中设置编译条件：-ObjC

设置info中网络允许<key>NSAllowsArbitraryLoads</key> <true/>

版本iOS10 以上

flutter混编请参考https://www.jianshu.com/p/abd937d0d9aa?utm\_campaign=hugo。

首先要完成SDK的初始化，配置IM服务和端口，日志和自动重连可选。

[IMSDKServer startup:@"192.168.5.3" port:1001];

    [IMSDKServer setAutoLogin:YES];

    [IMSDKServer opendebug:YES];

搭建消息回调对象。

    [IMMessageManager shareInstance].reviceEvent = self;

本地用户绑定IM前需要获取授权token，根据用户ID请求。示例如下【具体请求接口以后台SDK约定为主】：

- (void)loadLoginData {

    if (self.sendFiled.text) {

        self.userID = self.sendFiled.text;

    }

    NSMutableDictionary \*param = [NSMutableDictionary dictionary];

    [param setObject:self.userID forKey:@"userId"];

    [self.manager POST:KBaseUrl parameters:param headers:nil progress:nil success:^(NSURLSessionDataTask \* \_Nonnull task, id  \_Nullable responseObject) {

        NSDictionary \*dict = (NSDictionary \*)responseObject;

        NSLog(@"返回==%@",responseObject);

        if ([dict isKindOfClass:[NSDictionary class]] && [[dict objectForKey:@"code"] intValue] == 200) {

            self.token = [dict objectForKey:@"data"];

            if (self.token && self.userID) {

                [[IMMessageManager shareInstance] doLoginWithUserID:self.userID andToken:self.token];

            }

        }

    } failure:^(NSURLSessionDataTask \* \_Nullable task, NSError \* \_Nonnull error) {

        NSLog(@"=====+++++++++++++++++++++error:%@",error.description);

    }];

}

用户ID和token不能为空！IM登录如下

if (self.token && self.userID) {

[[IMMessageManager shareInstance] doLoginWithUserID:self.userID andToken:self.token];

}

IM退出：

[[IMMessageManager shareInstance]logout];

IM消息发送:

IMMessage \*message = [[IMMessage alloc]init];

    message.type = IMMSGType\_C2C;

    message.msg\_type = 0;

    message.to = self.toUserID;

    message.from = self.userID;

    [[IMMessageManager shareInstance]sendMesage:message];

IM消息接收

//收到消息

- (void)onRecieveMessage:(nonnull IMMessage \*)message {

    NSString \*text = self.contentView.text;;

    NSString \*msg = [NSString stringWithFormat:@"%@\n%@:%@",text,message.from,message.content];

    self.contentView.text = msg;

}

警告⚠️消息发送说明以下消息协议不建议修改，

扩展字段说明 ：扩展字段用来处理需要底层协议直接读取或处理的内容

@property(nonatomic, assign) int type;

@property(nonatomic, assign) int msg\_type;

@property(nonatomic, copy) NSString \*to;

@property(nonatomic, copy) NSString \*from;

@property(nonatomic, copy) NSString \*content;

@property(nonatomic, copy) NSString \*msgId;

@property(nonatomic, assign) long time;

@property(nonatomic, readwrite,) NSString \*ext1;

@property(nonatomic, readwrite,) NSString \*ext2;

@property(nonatomic, readwrite,) NSString \*ext3;

@property(nonatomic, readwrite) int32\_t ext4;

@property(nonatomic, readwrite) int32\_t ext5;

消息类型说明：content 可以直接穿入json字符串。

示例：

message.content = @"{\"msg\_Type\":\"image\",\"imageurl\":\"http://url.image.png\",\"nickname\":\"小明\",\"arvart\":\"http://xiaomingheadicon.png\"}"