即时通信 IM

更多实践

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

更多实践
直播间搭建
Web 内网代理方案
熟人社交场景实践方案
类 Discord 社群方案
互动游戏场景方案
AI 聊天机器人方案
类微信公众号搭建方案

更多实践
直播间搭建

最近更新时间：2025-02-28 10:14:42

直播在生活中可谓是无处不在，越来越多的企业、开发者们正在搭建自己的直播平台。由于直播平台本身所涉及到的，例如推拉直播流、直播转码、直播截图、直播混流、直播间聊天室、直播间互动（点赞、送礼、连麦）、直播间状态管理等需求往往比较复杂，本文章以腾讯云相关产品（腾讯云即时通信 IM 、腾讯云直播 CSS ）为基础，梳理了在搭建直播间过程中常见的需求的实现方案，以及可能遇到的问题、需要注意的细节点等，希望能帮助开发者们快速的理解业务、实现需求。
您也可以通过我们的 Demo 快速在线体验：

Web

包含IM功能以及直播互动相关能力

在线 Demo　Github 地址

一、准备工作

创建应用

使用腾讯云搭建直播间首先要在 控制台 创建一个即时通信 IM 应用，如下图所示：

添加直播推流及播放域名

搭建直播间，少不了直播功能，直播功能可以通过 云直播 来实现。需要添加推流及播放域名，如下图所示：

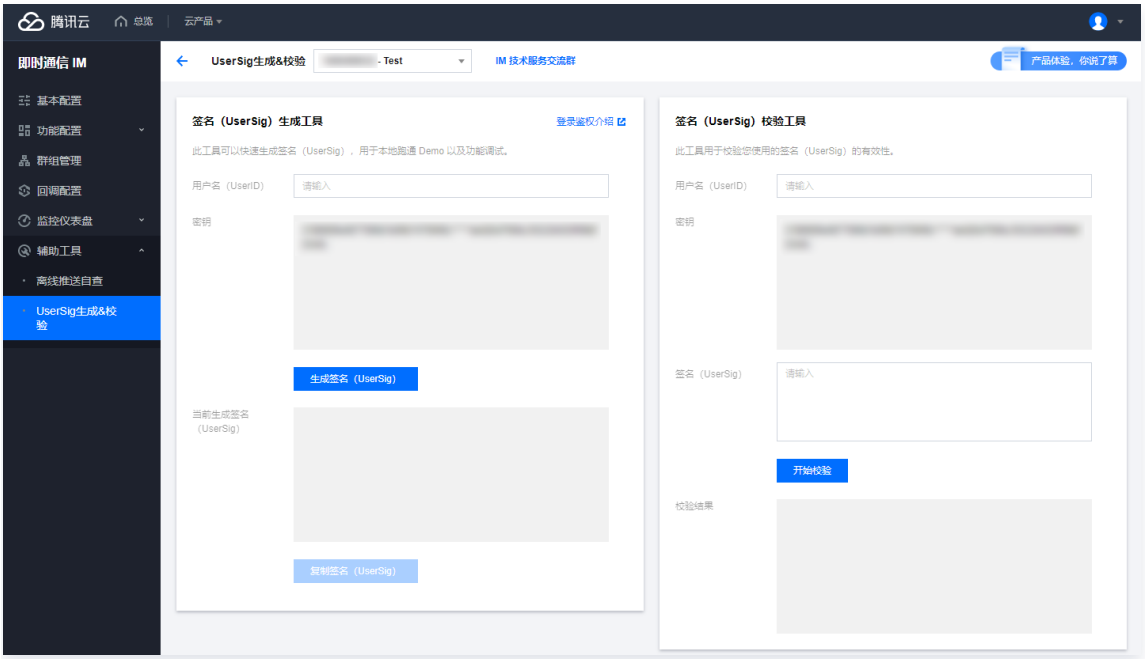
操作详情请参见 添加自有域名 。

完成相关基本配置

准备工作步骤一中创建的应用为体验版，只适合在开发阶段使用，正式环境用户可结合自己业务需求，开通专业版或旗舰版或企业版。不同版本之间的差异可参见 即时通信 IM 价格文档 。
在直播场景中，除了创建应用之外，还需要一些额外的配置：

使用密钥计算 UserSig

在 IM 的账号体系中，用户登录需要的密码由用户服务端使用 IM 提供的密钥计算，用户可参见 UserSig计算 文档，在开发阶段，为了不阻塞客户端开发，也可在 控制台计算 UserSig ，如下图所示：

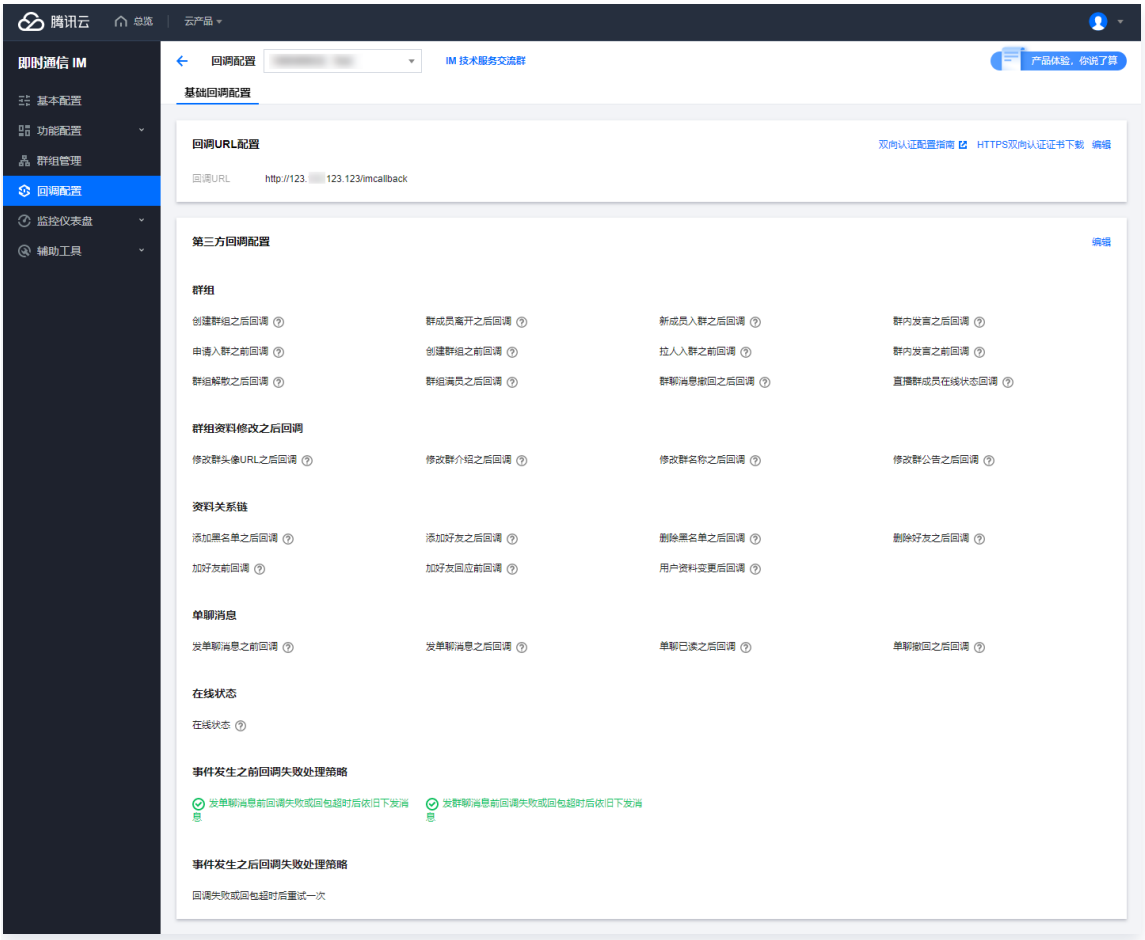


配置管理员账号

在直播过程中，可能需要管理员向直播间发送消息、禁言（踢出）违规用户等，这时就需要使用 即时通信 IM 服务端 API 来进行相应的处理，调用服务端api前需要 创建 IM 管理员账号，IM 默认提供一个 UserID 为 administrator 的账号供开发者使用，开发者也可以根据业务的场景，创建多个管理员账号。需要注意的是，IM 最多创建五个管理员账号。

配置回调地址以及开通回调

在实现直播间弹幕抽奖、消息统计等需求时，需要用到 IM 的回调模块，即 IM 后台在某些特定的场景回调开发者业务后台。开发者只需要提供一个 HTTP 的接口并且配置在 控制台 > 回调配置 模块即可，如下图所示：



集成客户端 SDK

在准备工作都完成好后，需要将即时通信 IM 以及云直播 CSS 的客户端 SDK 集成到用户项目中去。开发者可以根据自己业务需要，选择不同的集成方案。
即时通信 IM 可参见 IM 快速集成系列文档，
云直播 CSS 可参见 直播 SDK 快速集成系列文档 。
接下来文章梳理了直播间中常见的功能点，提供实践方案供开发者参见，并附上相关实现代码。

二、直播间各功能开发指引

直播间状态

直播间的状态一般分为如下几种：

序号	直播间的状态
1	直播待开始
2	直播中
3	直播暂停

4	直播结束
5	直播回放中

直播间状态有以下特点：

序号	特点
1	直播间状态的更改需要实时通知直播间用户
2	新进入直播间用户需要获取当前直播间状态

综上，这里提供两种实现方案。并分析两种方案的优缺点。

两种实现方案	优点	缺点
业务后台维护直播间状态，使用 IM 服务端 API 发送 群自定义消息 通知群内用户。	需要频繁多次获取直播间状态时，相比于将直播间状态存 IM 群资料，直播间状态存业务后台可以减少 IM SDK 的调用频率。提供在未集成 IM SDK 地方提供获取直播间状态的可能。	需要业务后台额外提供读写直播间状态模块。增加获取直播间数据异常概率，直播间数据来自业务后台和 IM 群资料两部分。发送自定义消息有丢失的可能，当消息量特别大时，低优先级消息会优先被丢弃从而影响直播间状态的显示。因此这里建议使用高优先自定义消息。
通过群自定义字段或者群属性存储直播间状态，通过客户端 SDK 群属性更改回调 通知群内用户。	开发者不需要提供额外的读写直播间状态模块。群属性变更回调理论上不存在丢失可能。直播间数据都从群资料中获取，统一数据源，减少异常。	在高曝光模块需要频繁获取群资料，增大 IM 压力。在未集成 IM SDK 模块，需要通过业务后台调用 IM 服务端 SDK 获取群资料。且调用频率有限制。

通过上面的分析，我们建议使用方案一与方案二相结合的方式，维护直播间状态：

1. 在直播间内，使用 方案一获取直播间状态。
2. 在直播间外、使用方案二获取直播间状态。
3. 业务后台直播间状态更改后及时通过 IM 服务端接口 同步数据到 IM 群资料（群属性、群自定义字段）。

群类型选择

在直播这个场景，用户聊天区域有以下特点：

1. 用户进出群频繁，且不需要管理该群会话信息（未读、lastMessage等）。
2. 用户自动进群不需要审核。
3. 用户临时发言，不关注群历史聊天记录。
4. 群人数通常比较多。

5. 可以不用存储群成员信息。

所以根据 IM 的 群特性，这里选择 AVChatRoom 作为直播间的群类型。
即时通信 IM 的直播群（AVChatRoom）有以下特点：

- **无人数限制，可实现千万级的互动直播场景。**
- 支持向全体在线用户推送消息（群系统通知）。
- 申请加群后，无需管理员审批，直接加入。

> **说明**
> IM Web &小程序 SDK 限制同一用户在同一时间内，只能进入一个 AVChatRoom，在 IM 的多端登录场景，如果用户登录终端一在直播间 A 观看直播，在 控制台配置 允许多端登录情况下，该用户登录终端二进入直播间 B 观看，这时终端一的直播间 A 会被退群。

直播间公告

直播间公告（主题）是每个直播间必备的内容，用户进入直播间可以看到该直播间的基本信息。同时，直播间公告也需要更改后实时通知给直播群中的群成员。与群直播状态类似，可以将直播状态存储在客户业务后台或者 IM 的群资料。区别于直播间状态，直播间公告有一些额外的事项，需要开发者关注。如下：

1. 群资料的 notification 以及 introduction 字段存储长度有限。客户下发不了比较长的通知（群简介最长**240**字节、群公告最长**300**字节）。
2. 群资料的 notification 以及 introduction 字段目前仅仅支持 string 类型。如果用户希望图文混排的公告，可通过 json string 形式设置，且图片为可访问的线上图片 url。
3. 如在 Web 端通过 editor工具设置的公告，如包含 html 标签在 native 端可能解析不了。

直播间公告可通过 服务端 API 或者客户端 SDK 设置群属性来设置。客户端通过获取群属性来获取当前群信息，可通过监听 GroupListener 中的 OnGroupInfoChange 来获取更改的群属性。
相关代码示例：

Android

```
// 客户端获取群资料
V2TIMManager.getGroupManager().getGroupsInfo(groupIDList, new
V2TIMValueCallback<List<V2TIMGroupInfoResult>>() {
  @Override
  public void onSuccess(List<V2TIMGroupInfoResult>
v2TIMGroupInfoResults) {
      // 获取群资料成功
  }

  @Override
```

```
  public void onError(int code, String desc) {
      // 获取群资料失败
  }
});
// 客户端修改群资料
V2TIMGroupInfo v2TIMGroupInfo = new V2TIMGroupInfo();
v2TIMGroupInfo.setGroupID("需要修改的群 ID");
v2TIMGroupInfo.setFaceUrl("http://xxxx");
V2TIMManager.getGroupManager().setGroupInfo(v2TIMGroupInfo, new
V2TIMCallback() {
  @Override
  public void onSuccess() {
      // 修改群资料成功
  }

  @Override
  public void onError(int code, String desc) {
      // 修改群资料失败
  }
});
```

iOS & Mac

```
[[V2TIMManager sharedInstance] getGroupsInfo:@[@"groupA"]
succ:^(NSArray<V2TIMGroupInfoResult *> *groupResultList) {
    // 获取群资料成功
} fail:^(int code, NSString *desc) {
    // 获取群资料失败
}];
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"需要修改的群 ID";
info.faceURL = @"http://xxxx";
[[V2TIMManager sharedInstance] setGroupInfo:info succ:^{
    // 修改群资料成功
} fail:^(int code, NSString *desc) {
    // 修改群资料失败
}];
```

Flutter

```
// 获取群资料
V2TimValueCallback<List<V2TimGroupInfoResult>> groupinfos = await
groupManager.getGroupsInfo(groupIDList: ['groupid1']);
// 修改群资料
groupManager.setGroupInfo(info: V2TimGroupInfo.fromJson({
    "groupAddOpt":GroupAddOptTypeEnum.V2TIM_GROUP_ADD_AUTH
    // ...其他资料
  }));
// 回调
TencentImSDKPlugin.v2TIMManager.addGroupListener(listener:
V2TimGroupListener(onGroupInfoChanged: ((groupID, changeInfos) {
    // 群信息更改回调
  })));
```

Web & 小程序

```
// 获取群资料
let promise = tim.getGroupProfile({ groupID: 'group1',
groupCustomFieldFilter: ['key1','key2'] });
promise.then(function(imResponse) {
  console.log(imResponse.data.group);
}).catch(function(imError) {
  console.warn('getGroupProfile error:', imError); // 获取群详细资料失
败的相关信息
});
// 修改群资料
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  name: 'new name', // 修改群名称
  introduction: 'this is introduction.', // 修改群简介
  // v2.6.0 起，群成员能收到群自定义字段变更的群提示消息，且能获取到相关的内容，
详见 Message.payload.newGroupProfile.groupCustomField
  groupCustomField: [{ key: 'group_level', value: 'high'}] // 修改群组
维度自定义字段
});
promise.then(function(imResponse) {
```

```
  console.log(imResponse.data.group) // 修改成功后的群组详细资料
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // 修改群组资料失
败的相关信息
});
// v2.6.2 起，提供了全体禁言和取消禁言的功能。目前群全体禁言后，不支持下发群提示
消息。
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  muteAllMembers: true, // true 表示全体禁言，false表示取消全体禁言
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 修改成功后的群组详细资料
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // 修改群组资料失
败的相关信息
});
```

群资料处理模块其他 SDK 代码示例

消息优先级

直播间的特色之一是用户的消息量非常大，每个用户可能都会频繁的刷消息。当上行消息达到 IM 的频控阈值时，IM 后台会丢弃部分消息，来保证系统的稳定运行。其实当消息达到客户端的频率太高，消息的可读性也降低了，所以对直播群消息进行频控是非常有必要的。
群消息分为 3 个优先级，后台会优先下发高优先级的消息。因此用户应根据消息的重要程度，来选择合适的优先级。
3个优先级从高到低，分别如下：

优先级	含义
High	高优先级
Normal	正常优先级
Low	低优先级

消息丢弃策略如下：
总消息数频控是指单个群每秒最多能发送的消息数限制，默认值为 40条/秒，采用每秒平均限频。消息数量超过限制后，后台优先下发优先级相对较高的消息，同等优先级的消息随机排序。
被频控限制的消息，不会下发，不会存入历史消息，但会给发送人返回成功；会触发 群内发言之前回调 ，但不会触发 群内发言之后回调 。

综上：制定直播间消息优先级规则是很有必要的。
对于直播间消息，我们按对于用户的重要性来进行优先级的划分从高到低依次是：

1. 用户的打赏、送礼消息，这类消息用户期望直播间的所有用户都能看到。
2. 正常的文字、语音、图片等消息。
3. 点赞消息。

> **说明：**
> 通常来说，C 端用户自己发送的消息，不论消息优先级的高低，都是一定要上屏的。在消息量特别大时，用户能够接受丢失少部分其他用户的消息。

设置消息优先级代码示例：

Android

```
// 创建自定义消息
V2TIMMessage v2TIMMessage =
V2TIMManager.getMessageManager().createCustomMessage("单聊自定义消
息".getBytes());
// 设置消息优先级为高优先级消息
v2TIMMessage.setPriority(V2TIMMessage.V2TIM_PRIORITY_HIGH)
// 发送消息
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage,
"receiver_userID", null, V2TIMMessage.V2TIM_PRIORITY_NORMAL, false,
null, new V2TIMSendCallback<V2TIMMessage>() {
    @Override
    public void onProgress(int progress) {
        // 自定义消息不会回调进度
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // 发送群聊自定义消息成功
    }

    @Override
    public void onError(int code, String desc) {
        // 发送群聊自定义消息失败
    }
```

```
});
```

iOS&Mac

```
/ 创建文本消息
V2TIMMessage *message = [[V2TIMManager sharedInstance]
createTextMessage:@"content"];
// 发送消息
[V2TIMManager.sharedInstance sendMessage:message
                                receiver:@"userID"
                                 groupID:nil
                                priority:V2TIM_PRIORITY_NORMAL
                           onlineUserOnly:NO
                          offlinePushInfo:nil
                                progress:nil
                                succ:^{
    // 文本消息发送成功
}
                                fail:^(int code, NSString *desc) {
    // 文本消息发送失败
}];
```

Flutter

```
/ 创建文本消息
V2TimValueCallback<V2TimMsgCreateInfoResult> createTextAtMessageRes
= await
TencentImSDKPlugin.v2TIMManager.getMessageManager().createTextAtMess
age(
    text: "test",
    atUserList: [],
  );
 if(createTextAtMessageRes.code == 0){
    String id =  createTextAtMessageRes.data.id;

       // 发送文本消息
```

```
    V2TimValueCallback<V2TimMessage> sendMessageRes = await
TencentImSDKPlugin.v2TIMManager.getMessageManager().sendMessage(id:
id, receiver: "", groupID: "groupID");
    if(sendMessageRes.code == 0){
      // 发送成功
    }
  }
```

Web&小程序

```
// 发送文本消息，Web 端与小程序端相同
// 1. 创建消息实例，接口返回的实例可以上屏
let message = tim.createTextMessage({
  to: 'user1',
  conversationType: TIM.TYPES.CONV_C2C,
  // 消息优先级，用于群聊（v2.4.2起支持）。如果某个群的消息超过了频率限制，后台
会优先下发高优先级的消息，详细请参见：
https://cloud.tencent.com/document/product/269/3663#.E6.B6.88.E6.81.
AF.E4.BC.98.E5.85.88.E7.BA.A7.E4.B8.8E.E9.A2.91.E7.8E.87.E6.8E.A7.E5
.88.B6)
  // 支持的枚举值：TIM.TYPES.MSG_PRIORITY_HIGH,
TIM.TYPES.MSG_PRIORITY_NORMAL（默认）, TIM.TYPES.MSG_PRIORITY_LOW,
TIM.TYPES.MSG_PRIORITY_LOWEST
  // priority: TIM.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    text: 'Hello world!'
  },
  // v2.20.0起支持C2C消息已读回执功能，如果您发消息需要已读回执，需购买旗舰版或
企业版套餐，并且创建消息时将 needReadReceipt 设置为 true
  needReadReceipt: true
  // 消息自定义数据（云端保存，会发送到对端，程序卸载重装后还能拉取到，v2.10.2起
支持）
  // cloudCustomData: 'your cloud custom data'
});
// 2. 发送消息
let promise = tim.sendMessage(message);
promise.then(function(imResponse) {
  // 发送成功
  console.log(imResponse);
}).catch(function(imError) {
```

```
    // 发送失败
    console.warn('sendMessage error:', imError);
  });
```

设置消息优先级其他 SDK 版本代码示例

礼物与点赞消息实践教程

礼物消息

1. 客户端短连接请求到自己的业务服务器，涉及到计费逻辑。
2. 计费后，发送人直接看到 XXX 送了 XXX 礼物。（以确保发送人自己看到自己发的礼物，消息量大的时候，可能会触发抛弃策略）。
3. 计费结算后，调用服务端接口发送发送自定义消息（礼物）。
4. 如果遇到连刷礼物的场景需要进行消息合并。
   4.1 如果直接选择礼物数量的刷礼物，例如：直接选择 99 个礼物，1 条消息发送，参数带入礼物数量 99。
   4.2 如果连击的礼物，不确定停留在多少个，可以合并每 20 个（数量自己调整）或者连击超过 1 秒，发送一个。按照上述逻辑，例如：连击 99 个礼物，优化后仅需要发送 5 条。

点赞消息

1. 点赞与礼物略不同，点赞往往不用计费，所以一般采用端上直接发送的方式。
2. 针对与有需要服务端计数的点赞消息，在客户端节流之后，统计客户端点赞次数，将短时间内多次点赞消息合并之后，仅发送一次消息即可。业务方服务端在发消息前回调中拿到点赞次数进行统计。
3. 针对与不需要计数的点赞消息，与步骤2中的逻辑一致，业务要在客户端对点赞消息节流后发送，不需要在发消息前回调中计数。

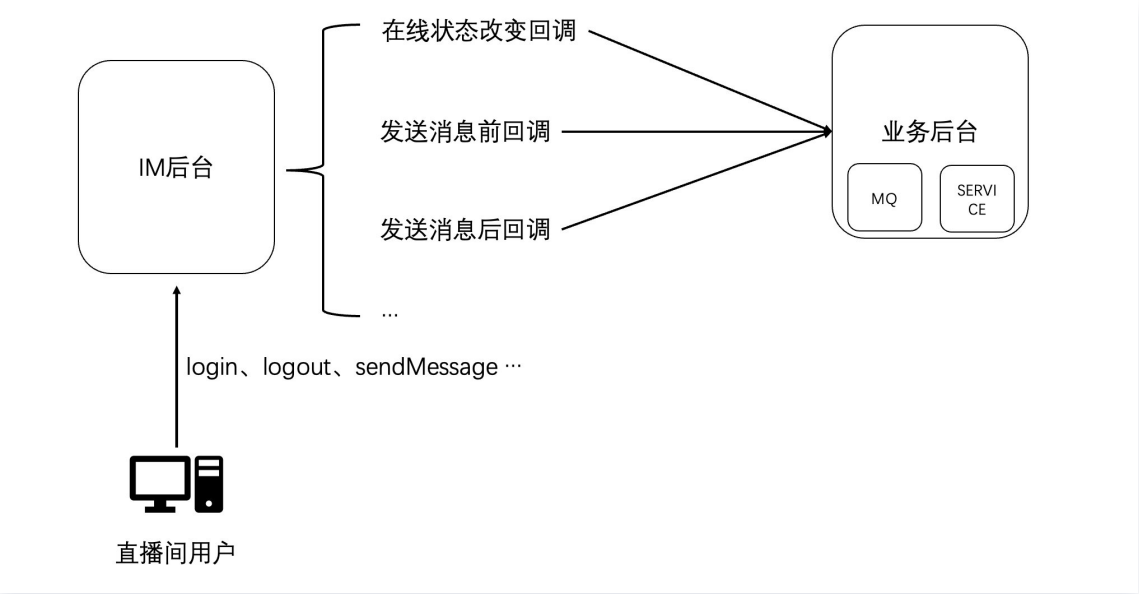
直播间用户身份、等级

大多直播间都会有用户等级的概念，开发者可能会根据以下几点按照一定的权重来计算用户的等级：

1. 直播间用户的在线时长。
2. 直播间用户成功发送普通直播间消息的数量。
3. 直播间用户点赞、送礼的数量。
4. 直播间用户是否开通直播间会员。
5. ...

其中和 IM 相关，例如：统计在线时长、统计消息量等我们都需要使用 IM 的回调来实现。在第一个准备工作的模块中，有配置回调的相关指引，控制台 中具体的回调如图：

统计相关信息的流程图如图：



消息发送后回调数据示例：

```
{
    "CallbackCommand": "Group.CallbackAfterSendMsg", // 回调命令
    "GroupId": "@TGS#2J4SZEAEL", // 群组 ID
    "Type": "Public", // 群组类型
    "From_Account": "jared", // 发送者
    "Operator_Account":"admin", // 请求的发起者
    "Random": 123456, // 随机数
```

```
    "MsgSeq": 123, // 消息的序列号
    "MsgTime": 1490686222, // 消息的时间
    "OnlineOnlyFlag": 1, //在线消息，为1，否则为0；直播群忽略此属性，为默认值0。
    "MsgBody": [ // 消息体，参见 TIMMessage 消息对象
        {
            "MsgType": "TIMTextElem", // 文本
            "MsgContent": {
                "Text": "red packet"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data"
}
```

开发者可根据数据中的 MsgBody 中的消息类型区分普通消息和点赞送礼消息。全部字段含义可参见 服务端 API 文档。

用户在线状态改变回调数据示例：

```
{
    "CallbackCommand": "State.StateChange",
    "EventTime": 1629883332497,
    "Info": {
        "Action": "Login",
        "To_Account": "testuser316",
        "Reason": "Register"
    },
    "KickedDevice": [
        {
            "Platform": "Windows"
        },
        {
            "Platform": "Android"
        }
    ]
}
```

开发者可根据 Info 中的字段判断用户的在线状态，从而统计用户的在线时长。全部字段含义可参见 服务端 API 。

> **说明：**
> 此外，不少直播间都会展示直播间热度，并且根据直播间热度，将直播间推荐给用户观看，直播间热度的统计方式和用户等级统计方式类似，也可以通过 IM 提供的回调系统来进行实现，这里就不在做过多的讲解。

直播间历史消息

使用 AVChatRoom 默认不存储直播间历史消息，当新用户进入直播间后，只能看到进入直播间后用户发送的消息。为了优化新进群用户的体验，可在控制台配置直播群用户拉取进群前消息条数，如图：

> **说明：**
> 此功能仅旗舰版或企业版用户才可开通，且仅支持拉群24小时内最多50条历史消息。

注意：直播群进群前历史消息通过信息消息回调返回。

Android

```
// 设置事件监听器
V2TIMManager.getMessageManager().addAdvancedMsgListener(v2TIMAdvance
dMsgListener);

// 接收消息
public void onRecvNewMessage(V2TIMMessage msg) {
    // 解析出 groupID 和 userID
    String groupID = msg.getGroupID();
    String userID = msg.getUserID();

    // 判断当前是单聊还是群聊：
    // 如果 groupID 不为空，表示此消息为群聊；如果 userID 不为空，表示此消息
为单聊

    // 解析出 msg 中的自定义消息
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_CUSTOM) {
        V2TIMCustomElem customElem = msg.getCustomElem();
        String data = new String(customElem.getData());
        Log.i("onRecvNewMessage", "customData:" + data);
```

```
    }
}
```

iOS&Mac

```
// 设置事件监听器
[[V2TIMManager sharedInstance] addAdvancedMsgListener:self];

/// 接收消息
/// @param msg 消息对象
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    // 解析出 groupID 和 userID
    NSString *groupID = msg.groupID;
    NSString *userID = msg.userID;

    // 判断当前是单聊还是群聊：
    // 如果 groupID 不为空，表示此消息为群聊；如果 userID 不为空，表示此消息
为单聊

    // 解析出 msg 中的自定义消息
    if (msg.elemType == V2TIM_ELEM_TYPE_CUSTOM) {
        V2TIMCustomElem *customElem = msg.customElem;
        NSData *customData = customElem.data;
        NSLog(@"onRecvNewMessage, customData: %@", customData);
    }
}
```

Flutter

```
// 拉取单聊历史消息
// 首次拉取，lastMsgID 设置为 null
// 再次拉取时，lastMsgID 可以使用返回的消息列表中的最后一条消息的id
TencentImSDKPlugin.v2TIMManager.getMessageManager().addAdvancedMessa
geListener(
        listener: V2TimAdvancedMessageListener(
          onRecvedNewMessage:(msg){
```

```
            }
        ),
);
```

Web & 小程序

```
let onMessageReceived = function(event) {
  // event.data - 存储 Message 对象的数组 - [Message]
  const messageList = event.data;
  messageList.forEach((message) => {
    if (message.type === TencentCloudChat.TYPES.MSG_TEXT) {
      // 文本消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.TextPayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_IMAGE) {
      // 图片消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.ImagePayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_SOUND) {
      // 音频消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.AudioPayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_VIDEO) {
      // 视频消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.VideoPayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_FILE) {
      // 文件消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.FilePayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_CUSTOM) {
      // 自定义消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.CustomPayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_MERGER) {
      // 合并消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.MergerPayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_LOCATION)
{
      // 地理位置消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.LocationPayload
    } else if (message.type === TencentCloudChat.TYPES.MSG_GRP_TIP)
{
      // 群提示消息 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.GroupTipPayload
```

```
    } else if (message.type ===
TencentCloudChat.TYPES.MSG_GRP_SYS_NOTICE) {
      // 群系统通知 - https://web.sdk.qcloud.com/im/doc/v3/zh-
cn/Message.html#.GroupSystemNoticePayload
      const { operationType, userDefinedField } = message.payload;
      // operationType - 操作类型
      // userDefinedField - 用户自定义字段（对应 operationType 为 255）
      // 使用 RestAPI 在群组中发送系统通知时，接入侧可从 userDefinedField
拿到自定义通知的内容。
    }
  });
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived)
```

直播间在线人数

直播间实时展示在线人数在直播场景也是一个十分常见的需求，实现方案分为两种，但两种也是各有优劣。

1. 通过客户端SDK提供的 getGroupOnlineMemberCount API 定时轮训的方式拉取群在线人数。
2. 通过用户进退群的后台回调统计，并通过服务端 API 例如， 群系统通知 或 群自定义消息 下发给群内所有成员。

通过客户端 SDK 提供的接口拉取在线人数的方式获取在线人数对于单直播间模式的用户来说基本可以满足用户需求。但对于 App 有多个直播间且需要在大量曝光位置展示直播间在线人数的需求，建议使用第二种方案来统计在线人数。

> **说明：**
> 开发者服务端在向客户端发送在线人数统计消息时，可采用定时发送的方式，例如每五秒发一次。但这种方式，在直播间人数变化不大时有额外的网络开销。建议开发者按照群人数变化率监测的方式来进行更新。直播间在线人数的准确性与实时性的优先级，开发者可以根据自身的业务来设置。

获取直播间在线人数代码如下：

Android

```
2TIMManager.getGroupManager().getGroupOnlineMemberCount("group_avcha
troom", new V2TIMValueCallback<Integer>() {
  @Override
  public void onSuccess(Integer integer) {
      // 获取直播群在线人数成功
```

```
    }

    @Override
    public void onError(int code, String desc) {
        // 获取直播群在线人数失败
    }
});
```

iOS&Mac

```
[[V2TIMManager sharedInstance]
getGroupOnlineMemberCount:@"group_avchatroom" succ:^(NSInteger
count) {
    // 获取直播群在线人数成功
} fail:^(int code, NSString *desc) {
    // 获取直播群在线人数失败
}];
```

Flutter

```
groupManager.getGroupOnlineMemberCount(groupID: '');
```

Web & 小程序

```
// v2.8.0 起，支持查询直播群在线人数
let promise = tim.getGroupOnlineMemberCount('group1');
promise.then(function(imResponse) {
  console.log(imResponse.data.memberCount);
}).catch(function(imError) {
  console.warn('getGroupOnlineMemberCount error:', imError); // 获取
直播群在线人数失败的相关信息
```

```
});
```

直播间禁言

直播间禁言分为两种，一种是整个直播间禁言，另一种是针对某个用户禁言。两种禁言都有其使用的业务场景。
一般来说我们使用服务端 SDK 来设置禁言。整个直播间禁言，通过 设置群属性 的全员禁言字段来设置。对单个群成员禁言，通过 设置群成员属性 来设置。
当直播间管理者在后台设置群禁言后，客户端在收到对应的回调事件后，应该把用户的输入框设置为 disable 状态。以免用户在发消息时提示发送消息失败。在解除禁言后也应该将输入框设置为 enable 状态。
客户端对应的回调代码如下：

Android

```
// 禁言群成员 userB 1分钟
V2TIMManager.getGroupManager().muteGroupMember("groupA", "userB",
60, new V2TIMCallback() {
  @Override
  public void onSuccess() {
      // 禁言群成员成功
  }

  @Override
  public void onError(int code, String desc) {
      // 禁言群成员失败
  }
});

// 全员禁言
V2TIMGroupInfo info = new V2TIMGroupInfo();
info.setGroupID("groupA");
info.setAllMuted(true);
V2TIMManager.getGroupManager().setGroupInfo(info, new
V2TIMCallback() {
  @Override
  public void onSuccess() {
      // 全员禁言成功
  }

  @Override
```

```
  public void onError(int code, String desc) {
      // 全员禁言失败
  }
});

V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
  @Override
  public void onMemberInfoChanged(String groupID,
List<V2TIMGroupMemberChangeInfo> v2TIMGroupMemberChangeInfoList) {
    // 禁言群成员监听
    for (V2TIMGroupMemberChangeInfo memberChangeInfo :
v2TIMGroupMemberChangeInfoList) {
      // 被禁言用户 ID
      String userID = memberChangeInfo.getUserID();
      // 禁言时间
      long muteTime = memberChangeInfo.getMuteTime();
    }
  }

  @Override
  public void onGroupInfoChanged(String groupID,
List<V2TIMGroupChangeInfo> changeInfos) {
    // 全员禁言监听
    for (V2TIMGroupChangeInfo groupChangeInfo : changeInfos) {
      if (groupChangeInfo.getType() ==
V2TIMGroupChangeInfo.V2TIM_GROUP_INFO_CHANGE_TYPE_SHUT_UP_ALL) {
        // 是否全员禁言
        boolean isMuteAll = groupChangeInfo.getBoolValue();
      }
    }
  }
});
```

iOS&Mac

```
// 禁言群成员 user1 1分钟
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA"
member:@"user1" muteTime:60 succ:^{
    // 禁言群成员成功
```

```
} fail:^(int code, NSString *desc) {
    // 禁言群成员失败
}];

// 全员禁言
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"groupA";
info.allMuted = YES;
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA"
member:@"user1" muteTime:60 succ:^{
    // 全员禁言成功
} fail:^(int code, NSString *desc) {
    // 全员禁言失败
}];

[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberInfoChanged:(NSString *)groupID changeInfoList:
(NSArray <V2TIMGroupMemberChangeInfo *> *)changeInfoList {
    // 禁言群成员监听
    for (V2TIMGroupMemberChangeInfo *memberChangeInfo in
changeInfoList) {
      // 被禁言用户 ID
      NSString *userID = memberChangeInfo.userID;
      // 禁言时间
        uint32_t muteTime = memberChangeInfo.muteTime;
    }
}

- (void)onGroupInfoChanged:(NSString *)groupID changeInfoList:
(NSArray <V2TIMGroupChangeInfo *> *)changeInfoList {
    // 全员禁言监听
    for (V2TIMGroupChangeInfo groupChangeInfo in changeInfoList) {
      if (groupChangeInfo.type ==
V2TIM_GROUP_INFO_CHANGE_TYPE_SHUT_UP_ALL) {
        // 是否全员禁言
        BOOL isMuteAll = groupChangeInfo.boolValue;
      }
    }
}
```

Flutter

```
// 禁言群成员 userB 10分钟
groupManager.muteGroupMember(groupID: '',userID: 'userB',seconds:
10);

// 全员禁言
groupManager.setGroupInfo(info: V2TimGroupInfo(isAllMuted:
true,groupID: '',groupType: 'Public'));

TencentImSDKPlugin.v2TIMManager.addGroupListener(listener:
V2TimGroupListener(onMemberInfoChanged: (groupID,
v2TIMGroupMemberChangeInfoList) {
    //群成员信息更改
  },
  onGroupInfoChanged: (groupID,info){
    // 群信息修改
  }
  ));
```

Web & 小程序

```
tim.setGroupMemberMuteTime(options);
let promise = tim.setGroupMemberMuteTime({
  groupID: 'group1',
  userID: 'user1',
  muteTime: 600 // 禁言10分钟；设为0，则表示取消禁言
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group); // 修改后的群资料
  console.log(imResponse.data.member); // 修改后的群成员资料
}).catch(function(imError) {
  console.warn('setGroupMemberMuteTime error:', imError); // 禁言失败
的相关信息
});
// 设置群成员在话题中的禁言时间
let promise = tim.setGroupMemberMuteTime({
  groupID: 'topicID',
  userID: 'user1',
  muteTime: 600 // 禁言10分钟；设为0，则表示取消禁言
```

```
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group); // 修改后的群资料
  console.log(imResponse.data.member); // 修改后的群成员资料
}).catch(function(imError) {
  console.warn('setGroupMemberMuteTime error:', imError); // 禁言失败的相关信息
});
// v2.6.2 起，提供了全体禁言和取消禁言的功能。目前群全体禁言后，不支持下发群提示消息。
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  muteAllMembers: true, // true 表示全体禁言，false表示取消全体禁言
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 修改成功后的群组详细资料
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // 修改群组资料失败的相关信息
});
```

群组监听其他 SDK 版本代码示例

需要注意的是，群成员禁言状态的变更默认不会下发通知给客户端，需要到 控制台进行配置：

> **说明：**
> 客户端 SDK 暂时不支持直播间禁言，可使用服务端 API 进行 封禁 以及 解封 。

直播间封禁

开发者可以通过服务端 封禁 的接口，来踢出直播间成员，并且封禁一段时间不允许被踢成员再次进群。
控制台相关配置 如图所示：

注意：

在客户端 SDK 6.6.X 及以上版本、Flutter SDK 4.1.1 及以上版本，可以使用直播间踢人接口实现封禁功能。

可参考代码如下：

Android

```
List<String> userIDList = new ArrayList<>();
userIDList.add("userB");
V2TIMManager.getGroupManager().kickGroupMember("groupA", userIDList,
"", new V2TIMValueCallback<List<V2TIMGroupMemberOperationResult>>()
{
  @Override
  public void onSuccess(List<V2TIMGroupMemberOperationResult>
v2TIMGroupMemberOperationResults) {
      // 踢人成功
  }

  @Override
  public void onError(int code, String desc) {
      // 踢人失败
  }
});
```

```
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
  @Override
  public void onMemberKicked(String groupID, V2TIMGroupMemberInfo
opUser,
  List<V2TIMGroupMemberInfo> memberList) {
      // 群成员被踢通知
  }
});
```

iOS&Mac

```
[[V2TIMManager sharedInstance] kickGroupMember:@"groupA"
memberList:@[@"user1"] reason:@""
succ:^(NSArray<V2TIMGroupMemberOperationResult *> *resultList) {
    // 踢人成功
} fail:^(int code, NSString *desc) {
    // 踢人失败
}];

[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberKicked:(NSString *)groupID opUser:
(V2TIMGroupMemberInfo *)opUser memberList:
(NSArray<V2TIMGroupMemberInfo *>*)memberList {
    // 群成员被踢通知
}
```

Flutter

```
groupManager.kickGroupMember(groupID: '',memberList: []);
```

Web & 小程序

```
tim.deleteGroupMember(options);
```

直播间敏感内容过滤

过滤直播间敏感内容也是直播业务非常重要的部分。您可以使用 IM 的内容审核（反垃圾信息）功能，针对不安全、不适宜的内容进行自动识别、处理，为您的产品体验和业务安全保驾护航。目前已提供云端审核服务：

- 如果您需要对文本、图片、音频、视频内容进行审核，且需要我们为您提供默认审核策略（也可支持自定义配置审核策略），推荐您接入 云端审核功能，在服务端检测由单聊、群聊、资料场景中产生的文本、图片、音频、视频内容，支持针对不同场景的不同内容分别配置审核策略，并对识别出的不安全内容进行拦截。

敏感内容过滤

在直播场景中，用户很可能会发送不合适的内容，特别是与敏感事件/人物相关、黄色不良内容等令人反感的内容，不仅严重损害了用户们的身心健康，更很有可能违法并导致业务被监管部门查封。
即时通信 IM 支持内容审核（反垃圾信息）功能，可针对不安全、不适宜的内容进行自动识别、处理，为您的产品体验和业务安全保驾护航。可以通过云端审核方式来实现：

- 云端审核功能：在服务端检测由单聊、群聊、资料场景中产生的文本、图片、音频、视频内容，支持针对不同场景的不同内容分别配置审核策略，并对识别出的不安全内容进行拦截。此功能已提供默认预设拦截词库和审核场景，只需在 IM 控制台打开功能开关，即可直接使用。

直播间群成员列表

在需要展示直播间在线群成员列表时，可以通过 `getGroupMemberList` 接口获取群成员列表，但 AVChatRoom 由于人数众多，不提供拉取全量成员列表的功能。根据旗舰版或企业版和非旗舰版或企业版区分不同的表现：

1. 非旗舰版或企业版客户可调用 `getGroupMemberList` 拉取最近进群的 30 位群成员。
2. 旗舰版或企业版客户可调用 `getGroupMemberList` 拉取最近进群的 1000 位群成员。此功能需要在 IM 控制台开启开关，如果不开启，默认跟非旗舰版或企业版一样，仅拉取最近进群 30 位群成员。

控制台配置如图2.6：

如果是非旗舰版或企业版，开发者也可通过 `getGroupMemberList` 与群监听中的 onGroupMemberEnter 和 onGroupMemberQuit 回调，在客户端维护当前在线群成员列表。但此方案用户退出直播间重新进入后，也只能获取最新的30位群成员。

Android

```
// 通过 filter 参数指定只拉取群主的资料
int role = V2TIMGroupMemberFullInfo.V2TIM_GROUP_MEMBER_FILTER_OWNER;
V2TIMManager.getGroupManager().getGroupMemberList("testGroup", role,
0,
    new V2TIMValueCallback<V2TIMGroupMemberInfoResult>() {
    @Override
    public void onError(int code, String desc) {
        // 拉取失败
    }

    @Override
    public void onSuccess(V2TIMGroupMemberInfoResult
v2TIMGroupMemberInfoResult) {
        // 拉取成功
    }
});
```

iOS&Mac

```
[[V2TIMManager sharedInstance] getGroupMemberList:@"groupA"
filter:V2TIM_GROUP_MEMBER_FILTER_OWNER nextSeq:0 succ:^(uint64_t
nextSeq, NSArray<V2TIMGroupMemberFullInfo *> *memberList) {
    // 拉取成功
} fail:^(int code, NSString *desc) {
    // 拉取失败
}];
```

Flutter

```
// 通过 filter 参数指定只拉取群主的资料 ，可指定ALL拉取全部群成员
groupManager.getGroupMemberList(count: 10,filter:
GroupMemberFilterTypeEnum.V2TIM_GROUP_MEMBER_FILTER_ADMIN,nextSeq:
'0',offset: 0,groupID: "",);
```

Web & 小程序

```
tim.getGroupMemberList(options);
```

直播间弹幕抽奖

直播抽奖与消息统计类似，都需要用到发消息后回调，通过检测消息内容，将命中抽奖关键词的用户加入抽奖池。在此不做过多的补充。

直播弹幕

AVChatRoom 支持弹幕、 送礼和点赞等多消息类型，轻松打造良好的直播聊天互动体验。

直播间大喇叭

大喇叭功能相当于一个直播间维度的系统公告功能，但有别于公告，大喇叭功能属于消息。当系统管理员下发大喇叭消息时，SDKAppID 下所有直播间均可收到该大喇叭消息。
大喇叭功能目前属于旗舰版或企业版功能，且需要在控制台开通。
业务后台发送大喇叭消息可参见文档：直播群广播消息

> **说明：**
> 若开发者版本非旗舰版或企业版，可在服务端发送群自定义消息进行实现。

三、直播音视频流部分

主播端（推流）

目前腾讯云直播推流有以下几种方式：
1.1 客户端推流可使用 OBS 工具推流
1.2 Web 端可使用 Web 推流
1.3 App 端可使用 直播 SDK 推流
1.4 小程序端可使用 小程序推流

> **说明：**
> 主播端推流前需要配置推流地址，请参见在 推流配置 或 地址生成器 生成地址，SDK接入过程详情可参见 直播推流 。

用户端（拉流）

用户端获取直播流的方式也有不同的方式例如：
1.1 PC 端可使用 VLC 工具播放
1.2 Web 端可使用 播放器 SDK 播放
1.3 App 端可使用 播放器 SDK 播放
1.4 小程序端可使用 小程序播放

> **说明：**
> 在拉流前也需要配置播放地址，请参见在 播放配置 或 地址生成器 生成地址，SDK接入过程详情可参见 直播播放 。

导播台

为了满足直播场景更多的需求，开发者可以使用云 导播台 对直播流进行处理，目前导播台支持如下功能：
1.1 直播水印、文本、转场。

1.2 点播转直播。

1.3 直播画质、码率、分辨率设置。

1.4 直播布局调整。

1.5 直播录制。

1.6 添加直播弹幕。

1.7 直播流监控。

更加详细的说明可参见 导播台文档 。

- **更多高级功能**

云直播除了推拉流业务，还有更多高级功能，例如：

1.1 极速高清转码 ，极速高清转码技术，智能识别场景动态编码，实现直播高清低码和画质提升。

1.2 直播时移 ，支持直播过程中暂停、回看精彩片段。

1.3 直播录制 ，可在直播过程中同步录制并存储，便于后续对录制视频的编辑和再传播。

1.4 直播水印 ，通过直播画面叠加明水印或数字水印实现视频防盗效果。

1.5 云端混流 ，根据您设定好的混流布局同步的将各路输入源混流成一个新的流，可实现直播互动效果。

1.6 拉流转推 ，可将其他平台的直播或点播视频通过直播形式分发，提供内容拉取并推送的功能，将已有的直播或视频推送到目标地址上。

四、常见问题

1. 云直播可以测试吗?

新用户开通云直播服务，会免费获得 20GB 的流量套餐包，有效期1年。此时测试产生的流量可以使用该套餐包抵扣，超出套餐部分按照后付费日结计费。同时也欢迎使用直播增值功能，例如：直播水印、转码、录制、截图、鉴黄等，增值功能默认关闭，按需开启使用并计费，详细功能了解可参见 产品概述 。

2. 直播的在线人数是否有上限?

腾讯云直播默认不限制观看直播的在线人数，只要网络等条件允许都可以观看直播。如果用户配置了带宽限制，当观看人数过多、超出了限制带宽时新的用户无法观看，此情况下在线人数是有限制的。

3. 自己发消息时，消息发送状态， `Message.nick` 与 `Message.avatar` 字段都为空，该怎么处理才能在界面上正常展示昵称和头像?

可通过 getUserInfo 接口，获取自己的用户信息中的 nick 和 avatar 字段作为消息发送发的 nick 和 avatar 字段。

4. 为什么会丢消息?

出现丢消息的可能原因如下：

- 直播群有40条/秒的频率限制，可通过消息发送前回调与消息发送后回调进行判断，若丢失的消息有收到消息发送前回调，未收到消息发送后回调，则该消息被限频。
- 可参见 消息相关问题 ，判断是否因为小程序/Web 端退出时，导致 Android/iOS/PC 同步退出。
- 如果是小程序/Web 出现问题，请确认您使用的 SDK 版本是否早于V2.7.6，如果是，请升级最新版。

如果您已排除以上可能性，您可以 提交工单 联系我们。

Web 内网代理方案

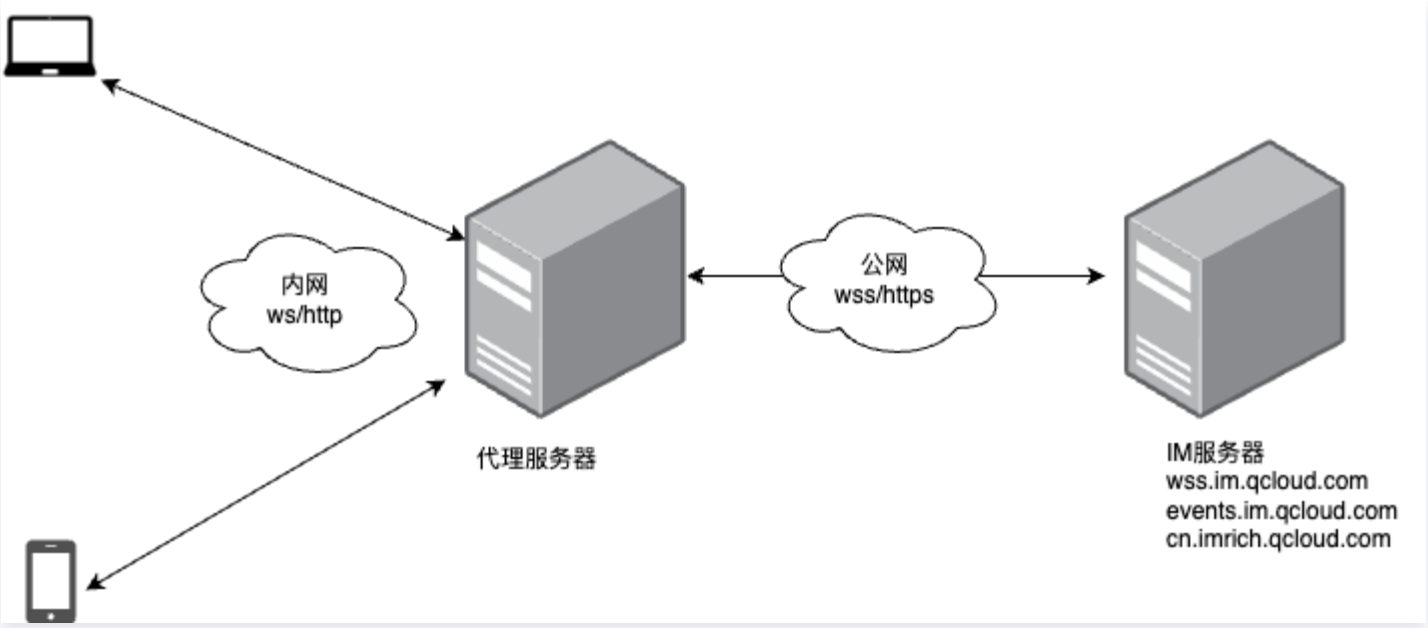
最近更新时间：2024-07-26 12:01:11

适用场景

用户终端在局域网内，需要跟外网的 IM 服务器完成信令和富媒体文件收发。

工作原理

在局域网内的用户终端与 IM 服务器之间搭建代理服务器，用户终端向代理服务器发送一个请求并指定目标，代理服务器向 IM 服务器转发请求并将响应返回给用户终端。



内网终端通过 ws/http 访问代理服务器，代理服务器通过 wss/https 访问 IM 富媒体服务器完成公网访问。 这种方式，满足内网以及外网用户同时访问。

Nginx 正向代理配置指引

```
#daemon off; # Don't run nginx in the background, good for monitoring
apps
worker_processes 7;
worker_rlimit_nofile 200000;
user root root;
events {
    worker_connections 80000;
}

http {
```

```
    log_format access '$remote_addr:$remote_port -
$server_addr:$server_port - $upstream_addr - $host[$time_local] '
'"$request" $content_length $status $body_bytes_sent '
'$request_time $upstream_response_time $realip_remote_addr';
    access_log logs/access.log access;
    error_log logs/error.log;
    resolver 119.29.29.29 ipv6=off;
    proxy_ssl_server_name on;

    server {
        listen 8805;

        location /binfo {
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "Upgrade";
            proxy_set_header Host "wss.im.qcloud.com";
            proxy_http_version 1.1;
            proxy_pass https://wss.im.qcloud.com/binfo$is_args$args;
        }

        location /v4/imopenstat {
            proxy_set_header Connection "close";
            proxy_set_header Host "events.im.qcloud.com";
            proxy_http_version 1.1;
            proxy_pass https://events.im.qcloud.com$uri$is_args$args;
        }

        location /upload {
            if ($request_method = 'OPTIONS') {
                add_header Access-Control-Allow-Origin *;
                add_header Access-Control-Allow-Headers X-Requested-
With,Content-Type,X-Real-Host;
                add_header Access-Control-Allow-Methods
'GET,PUT,HEAD,OPTIONS';
                return 204;
            }
            proxy_set_header Connection "close";
            proxy_set_header Host "cn.imrich.qcloud.com";
            proxy_http_version 1.1;
            proxy_pass https://cn.imrich.qcloud.com$uri$is_args$args;
        }

        location /imageinfo {
```

```
            if ($request_method = 'OPTIONS') {
                add_header Access-Control-Allow-Origin *;
                add_header Access-Control-Allow-Headers X-Requested-
With,Content-Type,X-Real-Host;
                add_header Access-Control-Allow-Methods
'GET,PUT,HEAD,OPTIONS';
                return 204;
            }
            proxy_set_header Connection "close";
            proxy_set_header Host "cn.imrich.qcloud.com";
            proxy_http_version 1.1;
            proxy_pass https://cn.imrich.qcloud.com$uri$is_args$args;
        }

        location /download {
            if ($request_method = 'OPTIONS') {
                add_header Access-Control-Allow-Origin *;
                add_header Access-Control-Allow-Headers X-Requested-
With,Content-Type,X-Real-Host;
                add_header Access-Control-Allow-Methods
'GET,PUT,HEAD,OPTIONS';
                return 204;
            }
            proxy_set_header Connection "close";
            proxy_set_header Host "cn.imrich.qcloud.com";
            proxy_http_version 1.1;
            proxy_pass https://cn.imrich.qcloud.com$uri$is_args$args;
        }
    }
}
```

WebSDK 配置

1. 搭建 nginx 的正向代理，如果分配的代理内网IP为10.0.0.10，则按上述指引 nginx 代理配置，IM 服务代理为 10.0.0.10:8805。
2. SDK 需要升级到 v3，可参见 V3集成指引 进行升级。开发者在初始化 SDK 时，填入以下参数，即可在内网通过代理正常使用腾讯云 IM 服务。

Methods

(static) create(options) → {Object}

即时通信 IM Web SDK 的工厂函数，用于生成 SDK 实例。
接入前，您需要在 云通信控制台 中创建一个云通信应用，并取得 `SDKAppID`。

Examples

```
let options = {
  SDKAppID: 0 // 接入时需要将0替换为您的云通信应用的 SDKAppID，类型为 Number
};
let tim = TIM.create(options); // SDK 实例通常用 tim 表示
```

```
let options = {
  SDKAppID: 0, // 接入时需要将0替换为您的即时通信应用的 SDKAppID
  proxyServer: `ws://${server_ip}:${server_port1}`, // WebSocket 服务器代理地址
  fileUploadProxy: `http://${server_ip}:${server_port2}`, // 图片、视频、文件上传代理地址
  fileDownloadProxy: `http://${server_ip}:${server_port3}` // 图片、视频、文件下载代理地址
};
let tim = TIM.create(options); // SDK 实例通常用 tim 表示
```

Parameters:

Name	Type	Description
options	Object	配置

Properties

Name	Type	Attributes	Description
SDKAppID	Number		云通信应用的 `SDKAppID`。如果您的业务有出海需求，请参考 出海指南
proxyServer	String | undefined	<optional>	WebSocket 服务器代理地址（小程序平台不支持）
fileUploadProxy	String | undefined	<optional>	图片、视频、文件上传代理地址。v2.27.0起支持（小程序平台不支持）
fileDownloadProxy	String | undefined	<optional>	图片、视频、文件下载代理地址。v2.27.0起支持（小程序平台不支持）

IM 服务域名及端口

1. 中国站

```
1、IM 长连接服务，端口 TCP：443
wss.im.qcloud.com
wss.my-imcloud.com
events.im.qcloud.com
events.my-imcloud.com

对应 IP 列表拉取接口：
https://cloud.tencent.com/document/product/269/45438，接口参数nettype=3

2、IM富媒体服务，端口 TCP：443
cn.imrich.qcloud.com
cn.rich.my-imcloud.com
file.im.qcloud.com
down.im.qcloud.com
cos.ap-shanghai.myqcloud.com
cos.ap-guangzhou.myqcloud.com
```

2. 新加坡站

```
1、IM 长连接服务，端口TCP：443
wsssgp.im.qcloud.com
```

```
wsssgp.my-imcloud.com
apisgp.im.qcloud.com
apisgp.my-imcloud.com
webimsgp.im.qcloud.com
webimsgp.my-imcloud.com

对应 IP 列表拉取接口：
https://www.tencentcloud.com/zh/document/product/1047/36742

2、IM富媒体服务，端口TCP：443
sgp.imrich.qcloud.com
sgp.rich.my-imcloud.com
cos.ap-singapore.myqcloud.com
cos.ap-singapore.tencentcos.cn
filesgp.im.qcloud.com
downsgp.im.qcloud.com
```

3. 韩国站

```
1、IM 长连接服务，端口TCP：443
wsskr.im.qcloud.com
wsskr.my-imcloud.com
apikr.im.qcloud.com
apikr.my-imcloud.com
webimkr.im.qcloud.com
webimkr.my-imcloud.com

对应 IP 列表拉取接口：
https://www.tencentcloud.com/zh/document/product/1047/36742

2、IM 富媒体服务，端口 TCP：443
kr.imrich.qcloud.com
kr.rich.my-imcloud.com
cos.ap-seoul.myqcloud.com
cos.ap-seoul.tencentcos.cn
filekr.im.qcloud.com
downkr.im.qcloud.com
```

4. 德国站

```
1、IM 长连接服务，端口 TCP：443
wssger.im.qcloud.com
wssger.my-imcloud.com
apiger.im.qcloud.com
apiger.my-imcloud.com
webimger.im.qcloud.com
webimger.my-imcloud.com

对应 IP 列表拉取接口：
https://www.tencentcloud.com/zh/document/product/1047/36742

2、IM 富媒体服务，端口 TCP：443
ger.imrich.qcloud.com
ger.rich.my-imcloud.com
cos.eu-frankfurt.myqcloud.com
cos.eu-frankfurt.tencentcos.cn
fileger.im.qcloud.com
downger.im.qcloud.com
```

5. 美国站

```
1、IM 长连接服务，端口 TCP：443
wssusa.im.qcloud.com
wssusa.my-imcloud.com
apiusa.im.qcloud.com
apiusa.my-imcloud.com
webimusa.im.qcloud.com
webimusa.my-imcloud.com

对应 IP 列表拉取接口：
https://www.tencentcloud.com/zh/document/product/1047/36742

2、IM富媒体服务，端口TCP：443
usa.imrich.qcloud.com
usa.rich.my-imcloud.com
cos.na-siliconvalley.myqcloud.com
cos.na-siliconvalley.tencentcos.cn
fileusa.im.qcloud.com
downusa.im.qcloud.com
```

熟人社交场景实践方案

最近更新时间：2025-07-08 15:29:12

社交模块是目前主流应用程序最常见的功能之一。有了社交模块，用户在您的应用内，可以自由的交流互动，并添加好友，关注其他用户等等。
这可在很大程度上，促进您应用程序的活跃度，吸引用户留存，获取更多新用户，并可拓展您应用的业务范围。让更多用户花更多时间在您的应用程序上。
例如：

- 交友软件，其核心便是社交聊天模块，用于匹配对话及用户关系链维护，让更多的用户相聚与相识。
- 音乐软件，可用社交模块让乐迷及粉丝群体实时沟通，打造音乐社区文化。
- 教育软件，可用社交模块打通 "学校-教师-家长" 循环，促进家校互动，形成家校社三合力，更大程度发挥教育影响作用，保证教育的一致性与连贯性。
- 电子游戏，特别是RPG类型，内置的聊天模块让玩家能在线组局，一起作战，并组建工会，创造游戏内社区，提升游戏社交属性，丰富体验，提升活跃度。
- 医疗软件，聊天及社群模块让患者间得以互助交流，一起战胜病魔，走出病情，让大家看到希望。
- 导航软件，堵车交流群，让归心似箭的旅行者们，交流前方最新情报，不再无聊干等，发发段子调侃一下，为拥堵的出行，带来一些希望与欢乐。

因此，社交聊天模块可谓是您应用程序不可或缺的能力。
本文将以腾讯云IM为基础，梳理现有应用在接入社交场景过程中常见需求，给出解决实现方案。以及可能遇到的问题、需要注意的细节点等，希望能帮助客户朋友们快速的理解业务、实现需求。

> **说明**
> 本文档主要介绍社交场景的通用 SDK 实现方案，文中示例截图来自于我们的 Flutter TUIKit，您可根据需要，选用我们提供的全部平台 SDK 或 TUIKit。

用户

腾讯云 IM 支持托管维护用户信息与用户资料。您可直接将您应用的用户资料存储于我们的服务内，并通过相关 API 进行读取/更新/维护操作。
对于社交场景，常见用户资料可分为基本信息资料和其他信息资料。

基本信息	其他信息
用户名，性别，生日，所在地，个性签名，昵称等	其他社交模块内需要的资料

导入现有用户数据

如果您需要给您现有应用，添加社交能力。可直接使用我们的服务端 API，快速将您现有用户数据，完整导入至腾讯云 IM 中。

导入完成后，现有用户可直接使用其原有身份数据，和其他用户发起会话，一起聊天，开启社交之旅。详情请参见 导入多个账号 – 服务端 API 。

用户在线状态

腾讯云IM支持自动上报并让其他用户获取 在线状态信息 。
状态包括：前台运行状态 / 后台运行状态 / 未登录状态。
利用这一能力，您可让用户看到其他用户的在线状态，增强互动性。
此外，您还可使用这一能力，针对您的业务场景，做许多功能拓展。例如，交友软件，能够优先推荐匹配在线的用户，让聊天可进行的更顺畅，缘分更快相聚。

会话列表用户在线状态	通讯录用户在线状态

获取用户在线状态

在客户端上， 您可调用 `getUserStatus` 方法， 批量查询其他用户的在线状态 。

此外，在服务端上，也可通过REST API，获取用户状态 。

订阅用户在线状态变更

其他用户的在线状态总是实时在变化的，您可在客户端上，调用 `subscribeUserStatus` 方法，批量订阅其他用户的在线状态 。当发生变化时，将通过回调函数通知您，您可根据其，修改界面UI并完成其他您的业务操作。

好友

好友管理，又称关系链管理，是社交场景的基础。众多会话/聊天特性，都要依赖于好友关系状态。有了好友关系链能力，众多的用户才能得以串联起来，互动形成整体。
例如微信/QQ只允许好友间发起一对一单聊；交友软件则常常可在非好友的情况下，进行有限度的聊天；在线娱乐社区软件，则常常不需要好友关系即可会话。
因此，您需要根据您的应用使用场景，确定好友及关系链管理的用法。

通讯录

好友关系

腾讯云IM支持单个用户添加最多3000个好友。

好友关系类型

好友关系类别包含单向好友和双向好友。

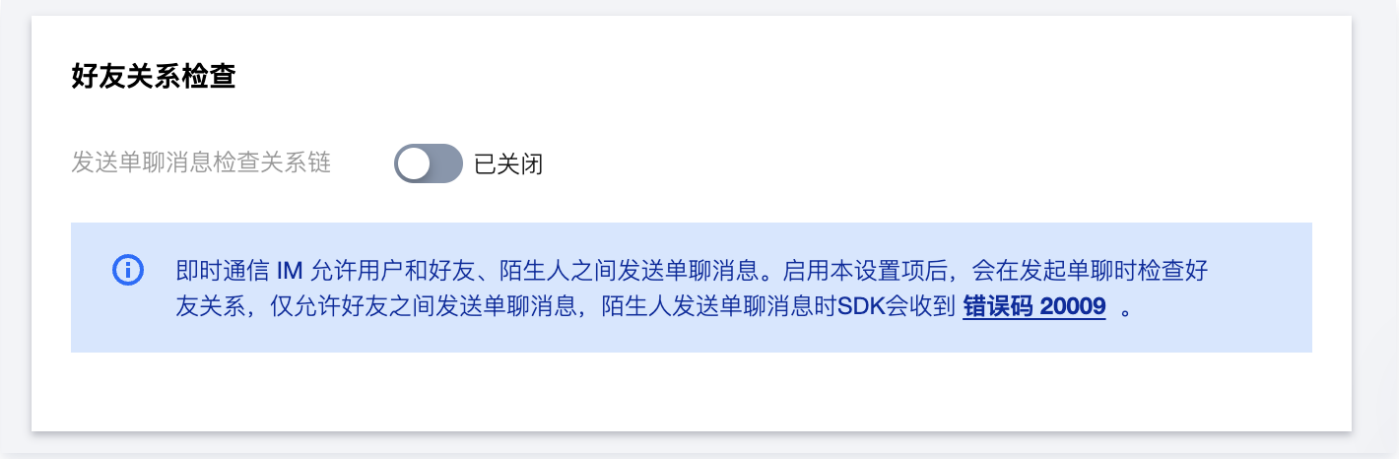
- 双向好友：用户 A 的好友表中有用户 B，B 的好友表中也有 A。
- 单向好友：用户 A 的好友表中有用户 B，但 B 的好友表中却没有 A。

添加好友验证方式

一回合加好友：如果账号 A 设置的加好友验证方式是 AllowType_Type_AllowAny，那么任何人添加 A 为好友都可直接添加成功，这种一个请求就添加好友成功的场景称作一回合加好友。
两回合加好友：如果账号 A 设置的加好友验证方式是 AllowType_Type_NeedConfirm，那么任何人添加 A，A 都会收到一个请求加好友验证消息，这是第一个回合，然后 A 对这个请求加好友验证消息进行同意操作时，这是第二个回合，这种需要验证的加好友场景就被称为两回合加好友。

非好友发消息

对于某些场景，需要非好友关系也能发送消息。例如对于交友软件，常常允许匹配到的陌生人，发送若干条消息打招呼。
这需要您在 腾讯云 IM 的控制台，关闭 “好友关系检查” 功能。



如果您需要针对陌生人发消息的数量加以限制，可在您的业务层实现。发送若干条消息后，不再允许发送即可。

群组

有的时候，仅一对一单聊无法满足您的社交场景要求。
例如对于音乐类型 App，优质的乐迷群及粉丝群，有助于打造音乐社区文化，吸引用户留存。

群组类型

腾讯云 IM 支持多种类型的群，为便于理解，在这里以常见的群聊举例。

微信群 – 好友工作群 Work

Work 群，类似普通微信群。创建后仅支持已在群内的好友邀请加群，且无需被邀请方同意或群主审批。
该群适合用于打造好友间互动拉入，产生的群。

QQ 群 – 陌生人社交群 Public

Public 群，类似 QQ 群。创建后群主可指定群管理员，用户搜索群 ID 发起加群申请后，需要群主或管理员审批通过（可选）才能入群。
该群适合用于打造兴趣社区，用户在您的 App 中，发现好玩的兴趣群组，可按需主动加入。

会议群 – Meeting

创建后可随意进出，且支持查看入群前消息。
适合用于音视频会议场景、在线教育场景等与实时音视频产品结合的场景。

直播群 – AVChatRoom

创建后可随意进出，没有群成员数量上限，但不支持历史消息存储。
适合与直播产品结合，用于弹幕聊天场景。

Discord – 社群 Community

创建后可随意进出，最多支持10w人，支持历史消息存储，用户搜索群 ID 发起加群申请后，无需管理员审批即可进群。详情可查看此文档 。

群资料

群资料主要包括 群组本身的资料 和 群成员资料。

群组本身的资料

群组资料是指单个群组维度的属性，包括群名称、简介、公告、群主等，以及群组维度自定义字段。

获取群资料

在客户端上，可调用 `getGroupsInfo` 方法，获取特定群组资料详情。

修改群资料

可修改群组名称、群组简介、群组公告、群组头像、群名片，修改加群选项、群纬度自定义字段、用户群内身份、群成员维度自定义字段和接收群消息选项等信息。
在客户端上，可调用 `setGroupInfo` 方法，修改特定群组资料详情。

群成员资料

群成员资料主要包括，特定成员，在群内的备注名/角色/禁言状态/自定义字段信息。在强社交场景的群内，会非常实用。

获取群成员资料

可通过客户端 `getGroupMembersInfo` 方法或 `getGroupMemberList` 方法获取此信息。

修改群成员资料

不同的群成员资料，调用不同的API方法修改。详情可查看此处 。

会话

一个会话，您可理解为同某个特定用户的单聊，或一个群聊的消息集合。当用户创建了一个单聊或群聊，当其中有消息的收发时，对应的会话就随之创建。
在腾讯云 IM 层面，每个会话都是一个 `V2TIMConversation` 类的实例，包括了 `会话类型 / 会话ID / 用户ID / 群ID / 显示名称 / 头像 / 最后一条消息 / 草稿 / 群聊类型 / 消息接收方式 / 群 @ 信息列表 / 是否置顶 / 标记列表 / 所属分组信息 / 自定义数据` 信息。

会话列表

会话列表，您可以理解成微信软件的首页。即，所有会话的集合。方便用户找到目标会话。
会话列表功能主要分为获取会话列表、处理会话列表更新。

获取会话列表

您可在客户端上调用 `getConversationList` 。该接口获取的是本地缓存的会话，如果服务器会话有更新，SDK 内部会自动同步，然后在 `V2TIMConversationListener` 通知回调。
如果您的应用场景会产生较多的会话数，考虑到加载效率、网络省流，我们建议您采用分页拉取的方式。每次分页拉取的数量建议不超过 100 个。具体方案可参考此处。

会话列表实时更新

当会话信息发生变化，例如收到一条新消息/设置消息草稿/出现一个新的会话，都会导致会话列表发生更新。
如果需要实时获取更新信息，需要通过客户端的 `addConversationListener` 方法，添加会话监听器。收到更新触发后，更改UI。

会话草稿

在发送消息时，可能会遇到消息尚未编辑完，就要切换至其它聊天窗口的情况。
这些未编辑完的消息可通过 `setConversationDraft` 接口保存，以便于下次回到这个聊天界面时，通过 `V2TIMConversation` 对象的 `draftText` 字段，获取到尚未编辑完的内容，继续编辑。
此类草稿信息，仅保存在本地。

置顶会话

会话置顶指的是把单聊或者群聊会话固定在会话列表的最顶部，不会被其他会话更新挤到底部，方便用户查找。
在社交场景中，用户常常需要将一些重要的人或群置顶。这在我们使用微信的过程中，很普遍。
置顶状态会存储在服务器，切换终端设备后，置顶状态会同步到新设备上。

置顶会话，通过客户端 `pinConversation` 即可。
调用 `getConversationList` 获取会话列表时，该接口返回的会话列表中，置顶的会话在前，未置顶的会话在后。您可以通过 `V2TIMConversation` 对象的 `isPinned` 字段，检查会话有没有置顶。

会话标记

在某些社交场景下，您可能需要对会话进行标记，例如 "会话标星"、"会话折叠"、"会话隐藏"、“会话标记未读”。
直接在客户端调用 `markConversation` 方法，即可标记，或取消标记一条会话。
标记后，在后续通过 `getConversationListByFilter` 方法请求会话列表时，可按照标记，过滤获取指定会话。

消息

消息是社交模块的灵魂。众多各种类型的消息，组成了一个个会话，使得用户与用户之间，紧密的串联在一起。腾讯云IM中，一对一单聊消息与群聊消息，用法在大部分场景中都类似，下面着重介绍几点。

消息类型

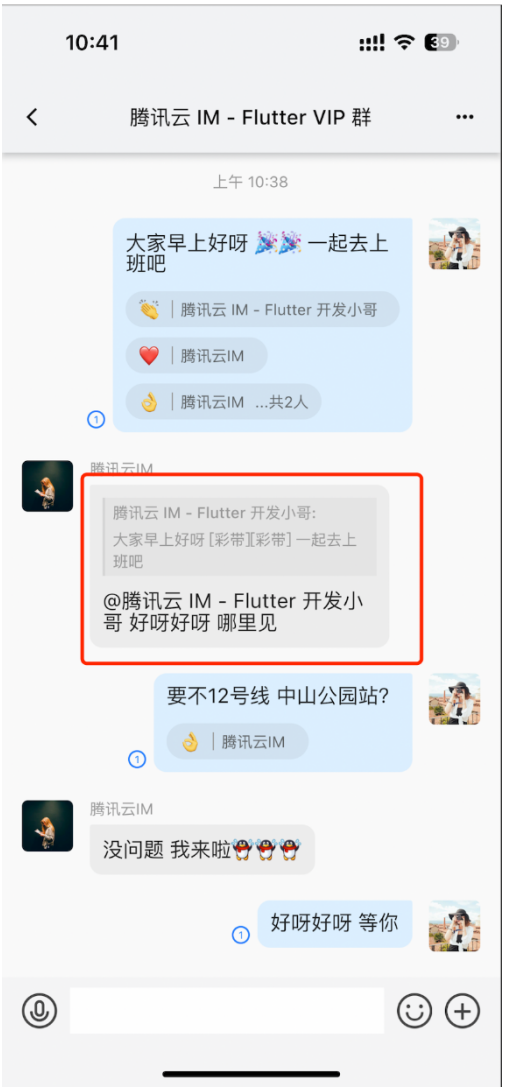
腾讯云IM支持多种类型的消息，如下：

功能类型	功能描述
文本消息	消息内容是普通文本
表情消息	表情消息为开发者自定义，可传入表情资源路径
地理位置消息	消息内容为地理位置标题、经度、纬度信息
图片消息	消息内容为图片的 URL 地址、尺寸、图片大小等信息，最大支持大小为28M的图片
语音消息	消息内容为语音文件的 URL 地址、大小、时长等信息，最大支持大小为28M的语音文件
文件消息	消息内容为文件的 URL 地址、大小、格式等信息，格式不限，最大支持大小为100M的文件
短视频消息	消息内容为短视频文件的 URL 地址、时长、大小、格式等信息，最大支持大小为100M的短视频文件
自定义消息	开发者自定义的消息类型，例如红包消息、石头剪刀布等形式的消息

消息回复

回复一条消息，既支持使用文字内容，发一条新消息，引用原消息；也支持使用 Emoji表情回应 。

引用回复文本	表情回应



引用回复文本

此方案效果和微信中，长按一条消息，选择 “引用”，效果一致。

引用消息，实际上，在腾讯云 IM SDK 层面，也是一条普通文本消息。文本消息的主体，则是回复的文字内容。

为了展示原消息的引用，需要在发送文本消息的时候，将原消息的信息，传入新消息的 `cloudCustomData` 字段中。例如，我们的 TUIKit，为了实现这个功能，传入了如下 JSON。

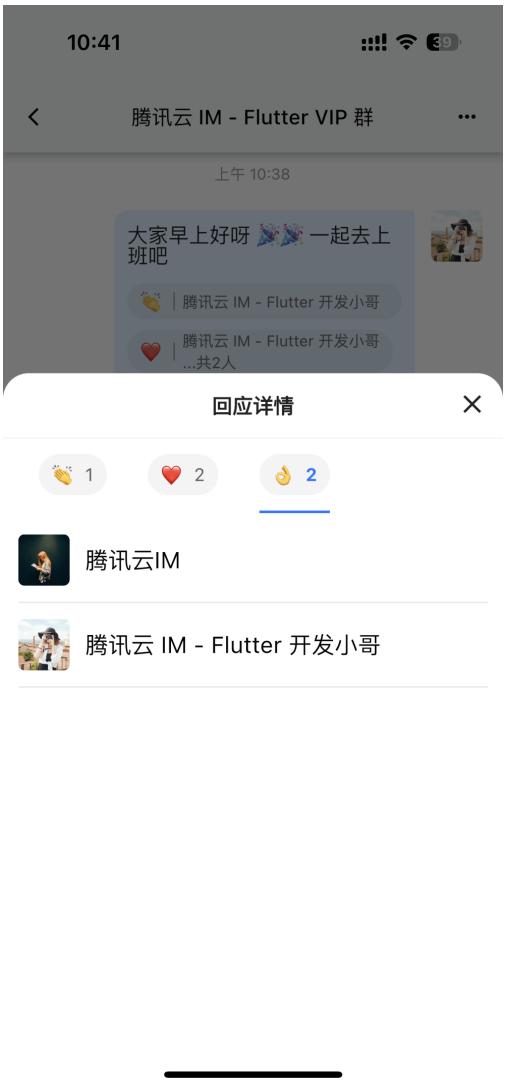
```
"messageReply": {
  "messageID": 原消息的ID,
  "messageAbstract": 原消息的描述，用于显示在消息列表气泡中,
  "messageSender": 原消息的发送者，建议使用备注名或昵称,
  "messageType": 原消息类型,
  "version": 协议版本
}
```

在消息列表中展示时，从 `cloudCustomData` 字段中，提取出上述JSON信息，直接用于拼接展示 `"${messageSender}: ${messageAbstract}"` 即可。
如需支持点击展示引用消息的区域，跳转至被引用原始消息。可根据上述JSON中的 `messageID` 字段，在消息列表中，找到这条消息，跳转即可。如果消息当前不存在于数组中，可直接调用 `getHistoryMessageList` ，参数传入 `messageID` ，获取本条消息及其上下文消息。

表情回应

在回复特定的单条消息时，您不仅可以直接引用原消息并回复，还可使用Emoji表情回应，高效表达“好的”、“赞同”、“很棒”、“哭笑不得”、“加油”等多种信息，大大降低沟通成本，解决多人聊天中消息冗杂的问题。
通常，若干个用户，可对同一条消息，或多条消息，点击一个或若干个回应表情。在显示上，这些回应信息，常常承载在不同的气泡中，以表情为首，后面跟着若干个名字。如本章节图片所示。
这些名称，需要支持点击，并跳转至用户Profile详情页中。若名字过多，还需要加以折叠，通过新窗口详情页展示。
发送表情回应，则可放置于消息的长按菜单中。

发送表情回应 TUIKit	表情回应详情 TUIKit



下面介绍实施细节：

表情回应的数据，存储于消息的 `cloudCustomData` 字段中，可以如下JSON格式示例。其中的 `key` 字段，如采用Emoji Unicode表情字符，可直接传入 Unicode；若采用图片小表情，可传入路径或文件名。

```
messageReactions: [
    {
        key: "表情名称1",
        users: ["用户1", "用户2", ...]
    },
    {
        key: "表情名称2",
        users: ["用户1", "用户2", ...]
    },
    ...
]
```

本部分代码可根据您的需求，加以修改。
展示时，渲染遍历上述结构体即可。
发送表情回应，则直接在客户端调用 `modifyMessage` 方法，修改消息本身的 `cloudCustomData` 字段即可。将当前用户头像，添加或从上述结构体中移除，完成回应或取消回应。
调用方法修改后，所有用户的 `V2TimAdvancedMsgListener` => `onRecvMessageModified` 监听器会触发，您可依此修改消息UI，展示最新表情回应内容。

删除消息

删除消息分为两种：删除本地消息和删除云端消息。
删除云端消息会在删除本地消息的基础上，同步删除云端存储的消息，且无法恢复。
删除本地消息，在客户端调用 `deleteMessageFromLocalStorage` 方法。需要注意的是，如果程序卸载重装，依旧能获取到被删除的消息。
删除云端存储的消息，在客户端调用 `deleteMessages` 方法。此方法不支持多端同步，即无法自动删除，其他设备上，已经存在的消息。

搜索消息

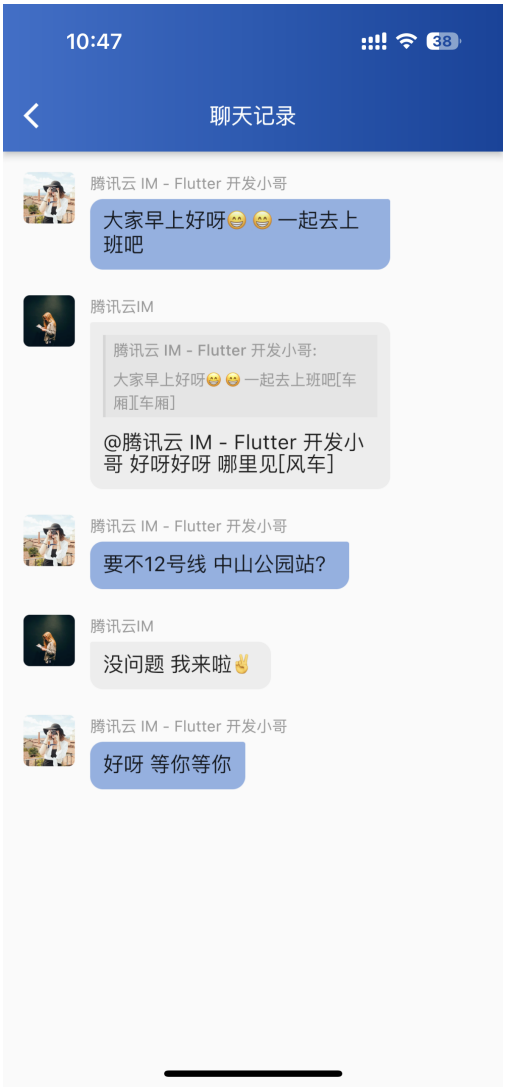
搜索能力是社交场景中的重要一环。
用户们常常需要，在特定会话中，抑或是全局中，快速准确搜索到某条消息。此外，也可作为社交活动运营工具，增加相关内容的引导，简洁高效。
您可在客户端，调用 `searchLocalMessages` 方法，并传入以 `V2TIMMessageSearchParam` 对象封装的关键词信息，即可完成搜索。
如果您希望在全部会话范围内搜索，只需要将 `V2TIMMessageSearchParam` 中的 `conversationID` 设置为空或者不设置即可。

全局搜索	会话内搜索

转发消息

在日常生活聊天或工作场景中，将一个会话中的消息，合并或逐条转发至另一个会话，是个非常高频且基础的操作。

合并转发消息	合并消息详细内容



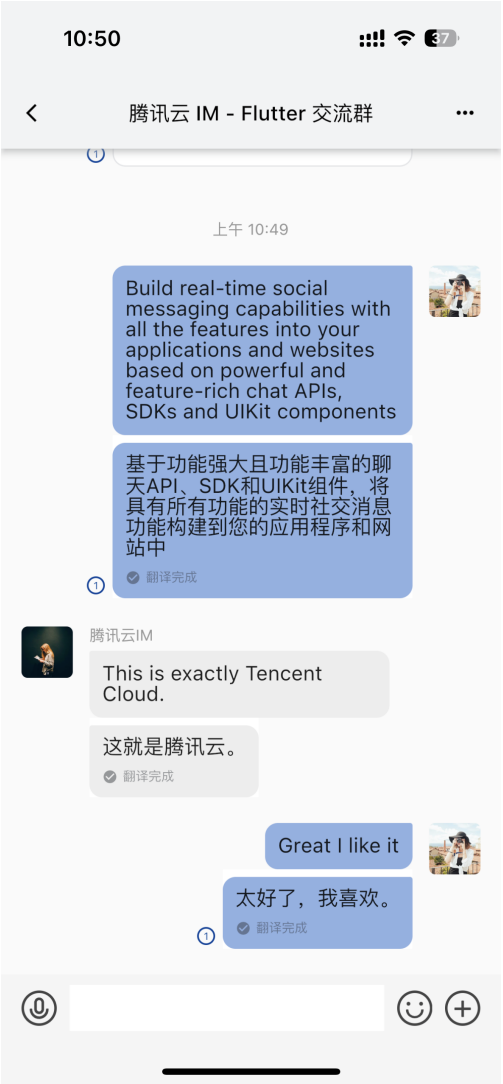
逐条转发消息，需要先在客户端调用 `createForwardMessage` 方法创建一条和原消息内容完全一样的转发消息，再调用 `sendMessage` 方法把转发消息发送出去。
合并转发消息，同样需要先创建后转发。需要额外注意的是，在客户端调用 `createMergerMessage` 方法创建一条合并消息时，需要设置原始消息列表，合并消息标题、合并消息摘要等信息。
若想转发至多个接收者，遍历调用 `sendMessage` 方法即可。
对于接收者端，若想展示上方右侧图片的合并消息详情，需要当用户点击合并消息的时候再调用 `downloadMergerMessage` 方法下载合并消息列表 UI 展示。

撤回消息

消息撤回是目前社交软件中必备的功能。
发送方可撤回一条已经发送成功的消息。默认情况下，发送者只能撤回 2 分钟以内的消息，此配置可按需修改。
撤回方在客户端，调用 `revokeMessage` 方法，接收方会收到消息撤回通知 `onRecvMessageRevoked` 。通知中包含了撤回消息的 msgID，您可根据这个 msgID 判断 UI 层是哪一条消息撤回了，然后把对应的消息气泡切换成 "消息已被撤回" 状态。

消息翻译

对于国际化的聊天场景，消息翻译功能必不可少，可大大提升跨语言交流效率。社交场景中，大型群聊内，有不同语言的交流存在，是非常之常见的。

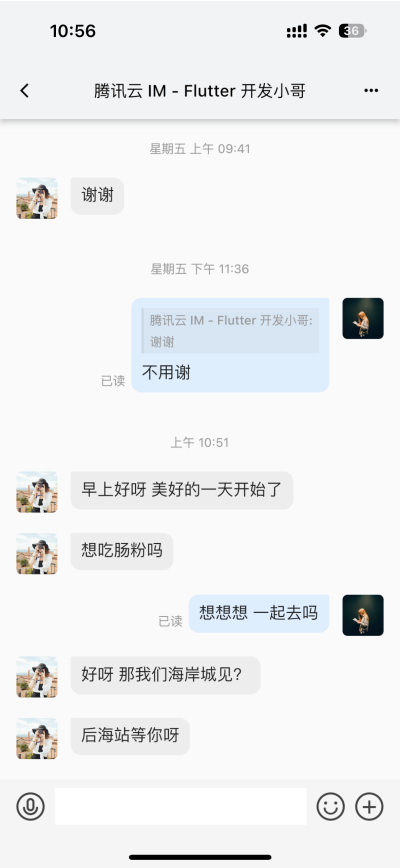


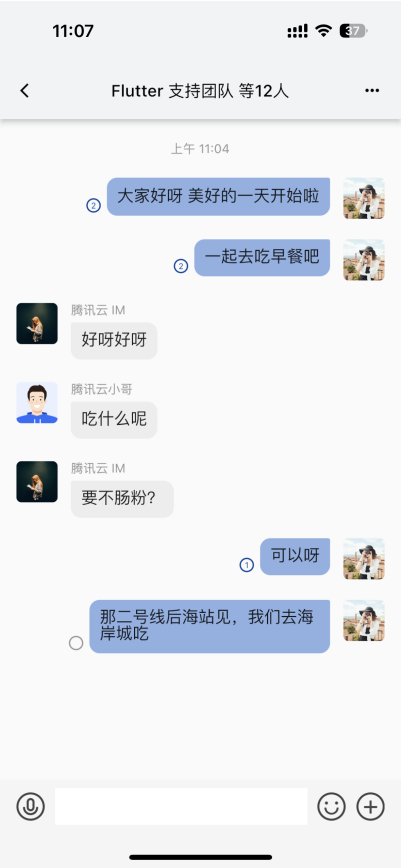
对于文本类型的消息，您可在客户端上调用 `translateText` 方法，将待翻译文本列表和目标语言传至我们的服务端。原语言可由您自行判断，也可由我们判断。
翻译结果返回后，我们建议您将其通过客户端的 `setLocalCustomData` 方法，存放于消息的 `localCustomData` 字段中，以便于后续直接展示，避免用户重复翻译，多次发送翻译请求。
我们支持众多语言的互相翻译，所有支持的语言可查看此处。

消息已读回执

单聊和群聊均支持消息已读回执功能，操作步骤一致。

单聊，TUIKit 中以文字承载	群聊，TUIKit 中以圆圈承载





是否启用此功能，可根据您的社交业务需求决定。

例如对于类似微信的熟人社交，已读回执的用处可能不是非常大；但是对于陌生人交友场景，已读回执则十分重要，帮助用户来确认，对方是否愿意跟自己聊下去，是否已读不回；对于工作聊天场景，群已读回执还能发挥更大的作用，可便捷看到群内哪些人已读哪些人未读，帮助发送者确认信息传递效率。

具体用法如下：

发送端创建消息后，先通过消息对象 `V2TIMMessage` 的 `needReadReceipt` 字段设置这条消息需要已读回执，再发送消息到会话中。

接收端收到消息后，根据消息对象 `V2TIMMessage` 的 `needReadReceipt` 字段判断消息是否需要已读回执。如果需要，当用户查看消息后，调用 `sendMessageReadReceipts` 方法发送消息已读回执。

接收端发送消息已读回执后，发送端可在 `V2TIMAdvancedMsgListener` 的 `onRecvMessageReadReceipts` 中收到已读回执通知，在通知中更新 UI，例如更新某条消息为 “2 人已读”。

此外，发送端也可主动请求消息已读回执信息。发送端从其他界面进入消息列表后，先请求获取历史消息，再调用 `getMessageReadReceipts` 方法请求获取消息已读回执信息。

群聊场景的消息已读回执，通常需要能够查看详情,显示群内哪些人已读，哪些人未读。当用户点击已读回执角标时，可调用 `getGroupMessageReadMemberList` 方法分页拉取消息已读或未读群成员列表。

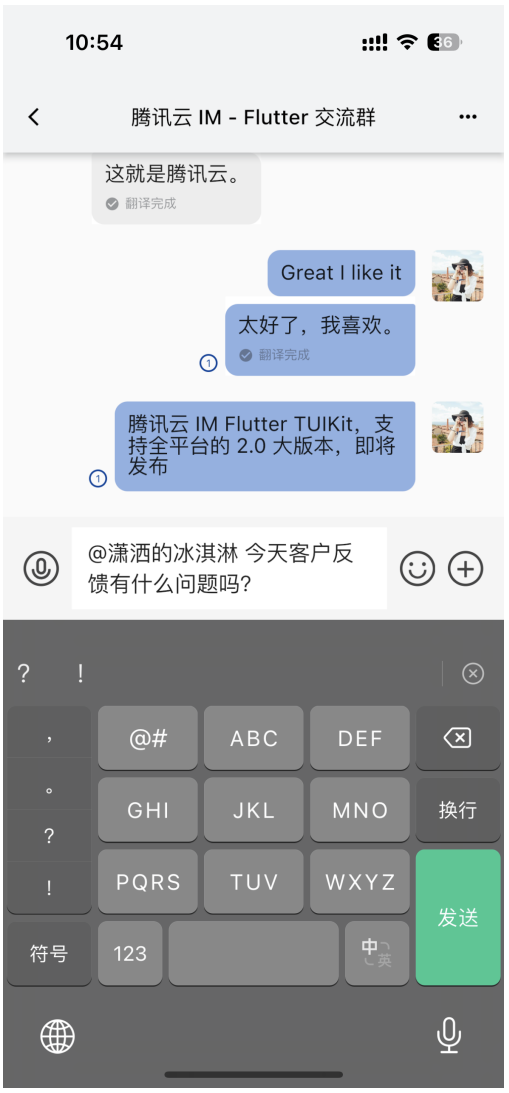
已读 群成员	未读 群成员

群内@消息

相信大家已经很熟悉，在群聊交流过程中，如果需要提及或提醒某些群成员，我们可直接 @ 他们。所有的社交聊天软件，都有这个基础功能。

当用户输入 @ 字符后，弹出群成员选择界面。选择完需要 @ 的成员后以 “@A @B @C……” 形式显示在输入框，并可继续编辑消息内容，完成消息发送。

监听 @ 字符选择群成员	编辑群 @ 消息发送	收到群 @



说明

图一：在聊天界面监听到输入框输入 "@" 字符后，可跳转到群成员选择界面，选择需要 @ 的群成员。

图二：在群成员选择完成后，重新返回聊天界面，继续编辑群 @ 消息发送。

图三：如果有消息 @ 我，自己会收到会话更新，可在会话 Cell 展示 “有人@我” 信息。

由于实现方案内容较多，您可查看此文档，获取详情。

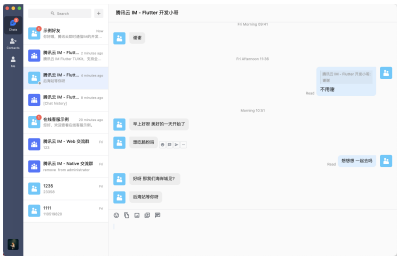
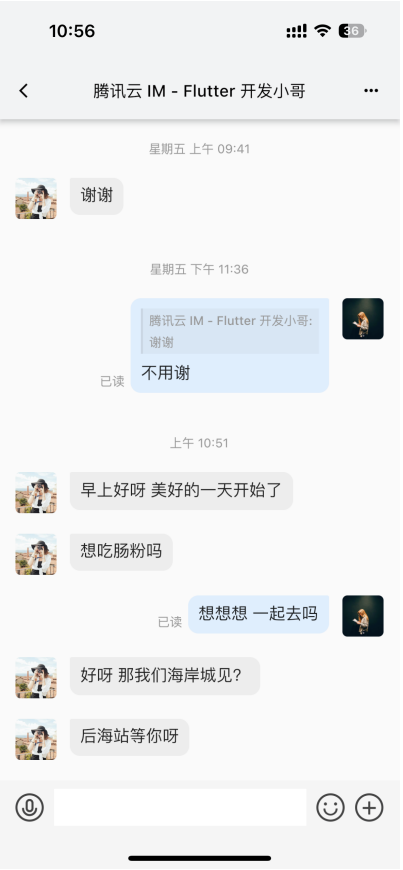
消息漫游

如果用户有多台设备，或者同时使用电脑和手机登录您的应用程序，用户们希望看到，无论在哪一端，历史消息都能尽可能完整。能从提供的历史消息上下文中，快速无障碍的加入聊天，满足社交场景高频率聊天的要求。

为了保证交流的连续性与流畅性，我们提供一套消息漫游存储能力，即用户更换终端的情况下，也可以获取到跟其他用户或者某个群的聊天记录，可以达到类似 QQ 软件的效果。

默认情况下，单聊消息和群聊消息有7天漫游，超过漫游时长的消息会被删除。此外，还支持在控制台修改消息漫游时长，延长消息漫游时长是增值服务。

以下截图演示了消息在手机和电脑之间漫游。**图片来自 Flutter TUIKit，一套代码完成电脑桌面端/Web 端/移动端应用的开发。**

电脑端	手机移动端
	

更多丰富的消息形态

我们默认提供的消息类型，可满足您大部分的聊天场景需求。但是对于社交软件来说，仅有这些还远远不够。
红包/送礼物/投票/发送匹配度/闪照等等一系列创新玩法，让您 App 的社交场景模块变得更加丰富多彩。
因此，您可使用我们提供的**自定义消息**能力，来发挥您的想象力，尽情创造激动人心的玩法及贴合您业务需求的功能。
发送自定义消息分两步：

- 调用 `createCustomMessage` 创建自定义消息，传入消息体。
- 调用 `sendMessage` 发送消息。

消息体中，您可以 JSON 格式，传入任何符合您需求的数据。例如，包含一个字段控制消息形态类型，还包含一个字段控制消息当前数据。
如果是投票/红包等类型的消息，如果您想实时更新投票数据/红包领取信息，可将此类信息放在消息体中，在客户端上，通过 `modifyMessage` 方法实时修改。

更多能力

云端审核

在消息发送、资料修改等场景中，有可能会扩散不合适的内容，特别是与敏感事件/人物相关、黄色不良内容等令人反感的内容，不仅严重损害了用户们的身心健康，还有可能违法并导致业务被监管部门查封。
即时通信 IM 支持 云端审核（反垃圾信息）功能，可针对不安全、不适宜的内容进行自动识别、处理，为您的产品体验和业务安全保驾护航。您可直接前往 云端审核控制台 开通免费试用，每个应用可免费试用7天（含10000次免费抵扣量）。

离线推送

社交场景下，用户需要随时都能够得知最新的消息，以加快聊天效率，促进社交关系的形成。
由于移动端设备的性能与电量有限，当 App 处于后台时，为了避免维持长连接而导致的过多资源消耗，我们推荐您使用各厂商提供的系统级推送通道来进行消息通知。系统级的推送通道相比第三方推送拥有更稳定的系统级长连接，可以做到随时接受推送消息，且资源消耗大幅降低。
我们目前原生支持的厂商系统有：苹果 iOS/Google FCM/OPPO/VIVO/华为/小米/魅族/荣耀。
理论上，集成系统原生的离线推送，需要手动对接各个厂商的 SDK，手动注册服务/获取 Token/承载点击回调页面，非常之复杂。
因此，我们提供了如下几款离线推送插件，封装了上述厂商的原生 SDK，大大降低了使用上手成本。您可直接按照文档配置。开箱即用。

- Native
- Flutter
- uni-app

音视频通话

许多时候，仅靠文字和图片还是不足以抒发我们内心的情感，可能打字聊天一小时，也比不是直接打一通视频电话来的爽快。

特别是对于社交场景下的用户们，他们一定有很多想法想要交流与诉说。

因此，我们也强烈推荐您，再集成我们的音视频通话能力，一步到位，完成一对一或多人群组的音频/视频通话，并且支持离线唤起能力。

详情可查看此文档 。

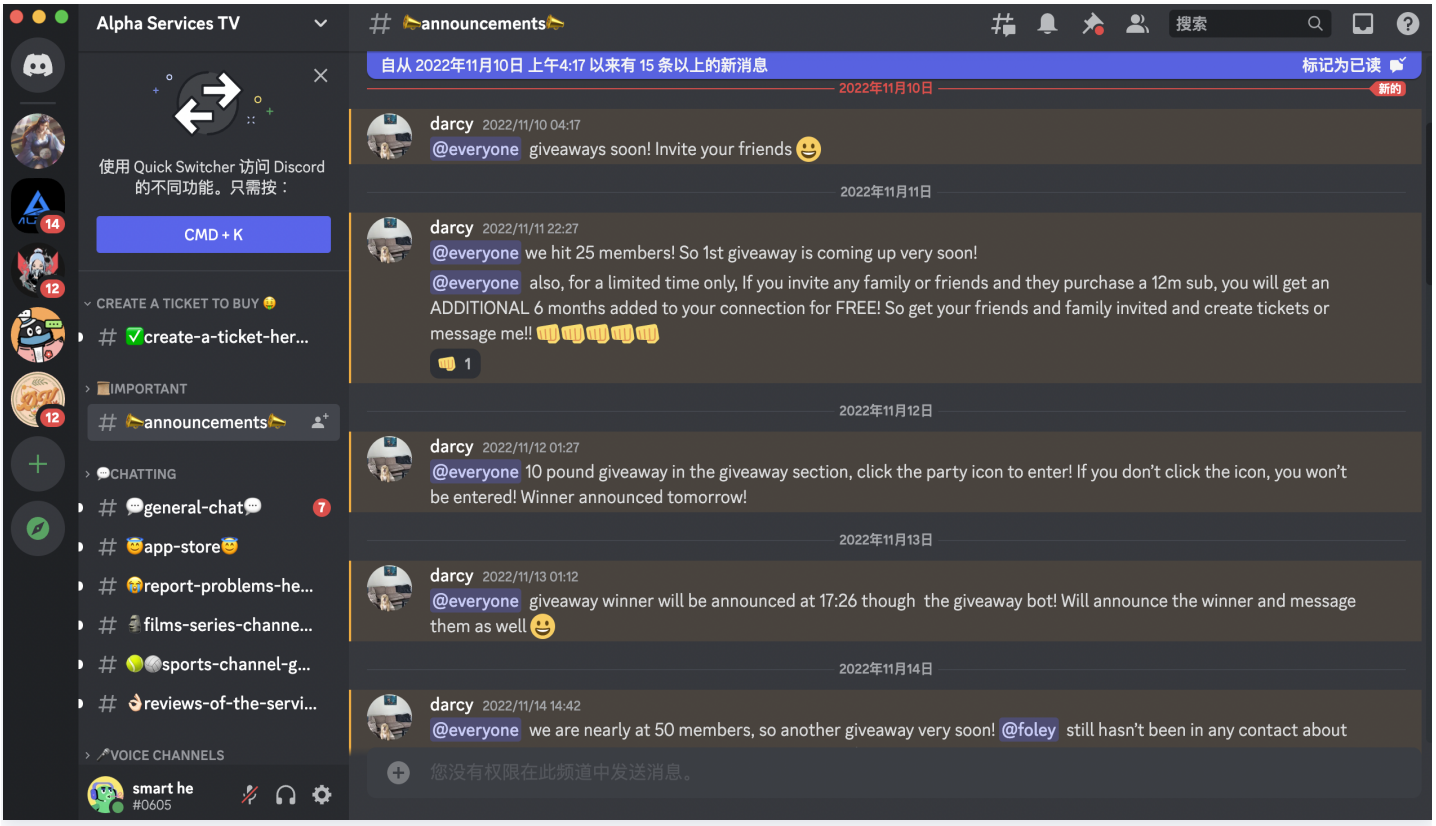
以上，就是使用腾讯云IM实现社交场景常见需求的解决方案。快来使用腾讯云 IM 打造属于您的社交产品吧！

类 Discord 社群方案

最近更新时间：2025-01-15 21:46:22

Discord 介绍

Discord 是一款专为社群设计的免费网络实时通话软件与数字发行平台，主要针对游戏玩家、教育人士、朋友及商业人士，用户之间可以在软体的聊天频道通过讯息、图片、影片和音讯进行交流。



Discord 概念

服务器

在 Discord 中有一种别于一般通讯软体之群组的群体聊天，称作伺服器（类似社团），伺服器拥有者可以在伺服器中创造属于自己的社群。

频道

在服务器中可以建立名为频道的聊天管道，分为语音、文字，其中的语音频道可以用来直播游戏与聊天等，频道可以设定与身份组整合各种权限，让 Discord 社群系统更加多样化。

子区

用户可以对特定的话题在子区内进行讨论。

类 Discord 功能适用场景

兴趣交友：新颖的用户增长模式

- 支持聚集超大规模爱好者，并通过**社群–分组–话题**让兴趣圈子保持细分垂直。
- 在开放的大社群中，为用户提供了封闭的小话题，这样舒服的中间地带让用户可以随意选择话题交流，提升了成员参与的积极性。

游戏社交：提高用户粘性和活跃度

一个社群内多种话题，可解决玩家的获取资讯、招募队友、探讨剧情、分享攻略等诸多信息需求。玩游戏前可以了解咨询，玩游戏中可以随时语音（聊天室永远在线），在游戏结束后还能在话题中继续深入交流。

粉丝运营：拥有一个高效的运营工具

告别一个又一个独立的分群（取代“深圳用户1群”“深圳用户2群”“广州用户1群”“上海用户1群”等多个群）运营，一个社群不同话题就搞定，不必再担心分身乏术，让粉丝运营更轻松精准！

组织管理：实现一种清晰的分层级沟通方式

组织内所有成员可拉入同一个社群中，再依托社群的层级功能和权限设置，实现分层沟通。

腾讯云 IM 类 Discord 技术优势

超大规模群成员

腾讯 IM 社群相比普通 QQ、微信群，容量拓展了近万倍，满足您兴趣交友、粉丝运营、游戏社交、组织管理等场景下的海量成员需求。

消息可靠性

腾讯 IM 社群大幅拓展成员容量的同时继承了腾讯强劲的消息能力，依托20余年技术积累构建的完善且可靠的消息系统。为客户提供超过99.99%的消息收发成功率及服务可靠性，帮助客户轻松应对亿级海量并发。

消息推送性能

腾讯 IM 社群采用“快慢通道”+“两级合并推送”的全新消息推送架构，有效平衡时间和空间，提升系统推送性能，降低终端性能消耗，在超大群中也能为用户提供与常规群组一致的消息互动体验。

消息状态与用户权限维护

腾讯 IM 社群提供消息编辑、撤回、转发等丰富拓展能力。禁言、消息免打扰、未读消息计数、资料编辑等均支持用户进行全局、社群、话题级别的分别自定义。

版权所有：腾讯云计算（北京）有限责任公司

基于腾讯云 IM 搭建的准备工作

创建腾讯云 IM 应用

本篇教程以腾讯云 IM 为基础展开，因此，需要在 腾讯云 IM 控制台 进行应用创建。如图所示：

创建应用后，在 腾讯云 IM 控制台应用基本信息页面，查看应用基础信息。

了解相关配置和能力

使用腾讯云 IM 实现 Discord 相关的功能，您需要提前了解腾讯云 IM 相关的基础概念以及本教程后续会提到的一些专有名词，包括但不限于如下内容：

- SDKAppID：腾讯云 IM 会给每个应用分配一个 SDKAppID，在控制台创建应用后在应用详情页查看，开发者可以在初始化腾讯云IM客户端 SDK 和计算用户登录票据时使用。详情可参见无 UISDK 初始化 以及 登录 文档。
- 密钥：在腾讯云 IM 控制台应用详情页可查看当前应用密钥，在计算用户登录 SDK 票据时会用到。
- 用户账号：登录腾讯云 IM 用户必须在腾讯云 IM 的账号体系中，当用户使用客户端 SDK 登录成功时，腾讯云 IM 后台会自动创建 IM 用户。同时，可以使用腾讯云 IM 提供的服务端 API 将用户导入到 IM 的用户体系中。
- 群组：到目前为止，IM根据不同场景的需要，提供了 5种类型的群，用户在群里发言，群成员均可收到消息。
- 回调配置：开发者除了可以主动集成 IM 提供的客户端 SDK 与服务端 API，IM 也会主动在特定的业务逻辑出发时将必要的信息回到给开发者服务端。只需要开发者在腾讯云 IM 控制台 回调配置 模块完成相应的配置即可。腾讯云 IM 提供了丰富的回调配置，并且保证回调的高可靠，开发者可通过回调实现很多自定义需求。
- 自定义字段：默认情况下，腾讯云 IM 提供给开发者使用的字段能满足绝大部分需求，如用户需要扩展字段，腾讯云 IM 也为各个模块提供了自定义字段，如：
  - 用户自定义字段
  - 好友自定义字段
  - 群自定义字段
  - 群成员自定义字段

> **说明**
> 用户使用自定义字段，可先在控制台进行配置，再使用 SDK\API 进行读写即可。

集成客户端&服务端 SDK

在实现 Discord 相关的功能时，需要集成 IMSDK，腾讯云 IM 提供了丰富且易用的 SDK 以及服务端 API，使用同一 SDKAppID 登录的应用，在各个端上消息互通。开发者可根据自己的业务需求场景以及技术栈进行评估，选择合适的 SDK。

基于腾讯云 IM 搭建功能解析

如上图所示，Discord 的功能主要分为服务器、频道、以及子区，服务器与服务器之间是内容上的区别，如王者荣耀服务器，和平精英服务器等，在服务器内可以创建不同类型的频道，如文字频道，语音频道、公示频道等。用户的交流实际是在各个频道中进行的。当用户对交流的某一个内容有更多的想法时，可以选择对该内容创建子区来进行更多的交流。Discord 的核心玩法就如分析的这样，接下来本教程会逐一分析如何通过腾讯云 IM 来实现相关的功能。

> **说明**
> **如涉及到代码演示，本教程以 Android 端（Java SDK）为例进行展示，其他版本的SDK接口调用可参见文档 无 UI 集成方案 部分。**

服务器

创建服务器

经过分析可以知道，Discord 的服务器列表有这样一些特性：

1. 服务器中人数可能特别多。
2. 用户实际不会在服务器中交流，只会在服务器内的频道或者子区中聊天。
3. 服务器中的聊天历史需要存漫游。
4. 可自由进入。
5. 可以在服务器中创建频道。

虽然 IM 提供了五种不同类型的群，但从 Discord 的服务器特性来看，只有 社群（Community） 符合服务器的特性，腾讯云 IM 社群创建后可以随意进出，最多支持10w人，支持历史消息存储，用户搜索群 ID 发起加群申请后，无需管理员审批即可进群。

可通过腾讯云 IM 提供的 createGroup 接口进行服务器（群）创建，需要注意的是，创建的群类型为 Community、setSupportTopic 需要设置为 true，这样才可在服务器中创建频道。演示代码如下：

```
V2TIMGroupInfo  groupinfo = new V2TIMGroupInfo();
```

```
groupinfo.setGroupName("测试服务器");
groupinfo.setSupportTopic(true);
// 初始群成员
List<V2TIMCreateGroupMemberInfo> memberList = new
LinkedList<V2TIMCreateGroupMemberInfo>();
// 其他配置，如服务器头像等
V2TIMManager.getGroupManager().createGroup(groupinfo, memberList, new
V2TIMValueCallback<String>() {
            @Override
            public void onError(int i, String s) {
                // 创建失败
            }

            @Override
            public void onSuccess(String s) {
                // 创建成功，返回服务器ID
            }
});
```

接口调用成功后，会在 onSuccess 回调中返回服务器 ID，在后续的在服务器中创建频道的功能中会用到。

注意

开发者也可以使用 IM 提供的服务端 API，在服务端创建服务器 。关键参数如下：

```
{
    "Type": "Community",      // 群组类型 (必填)
    "Name": "TestCommunityGroup", // 群组名称 (必填)
    "SupportTopic": 1             // 是否支持话题选项, 1代表支持,0代表不
支持
}
```

```
###### 服务器列表

在 Discord 最左侧有个服务器列表的功能，展示的是用户已加入的服务器列表。针对与社群场
景，腾讯云 IM 提供了专门的 API 进行查询

```java
V2TIMManager.getGroupManager().getJoinedCommunityList(new
V2TIMValueCallback<List<V2TIMGroupInfo>>() {
```
```

```
            @Override
            public void onSuccess(List<V2TIMGroupInfo> v2TIMGroupInfos)
{
                // 获取服务器列表成功，返回的List<V2TIMGroupInfo>代表服务器列表
的基本信息
            }

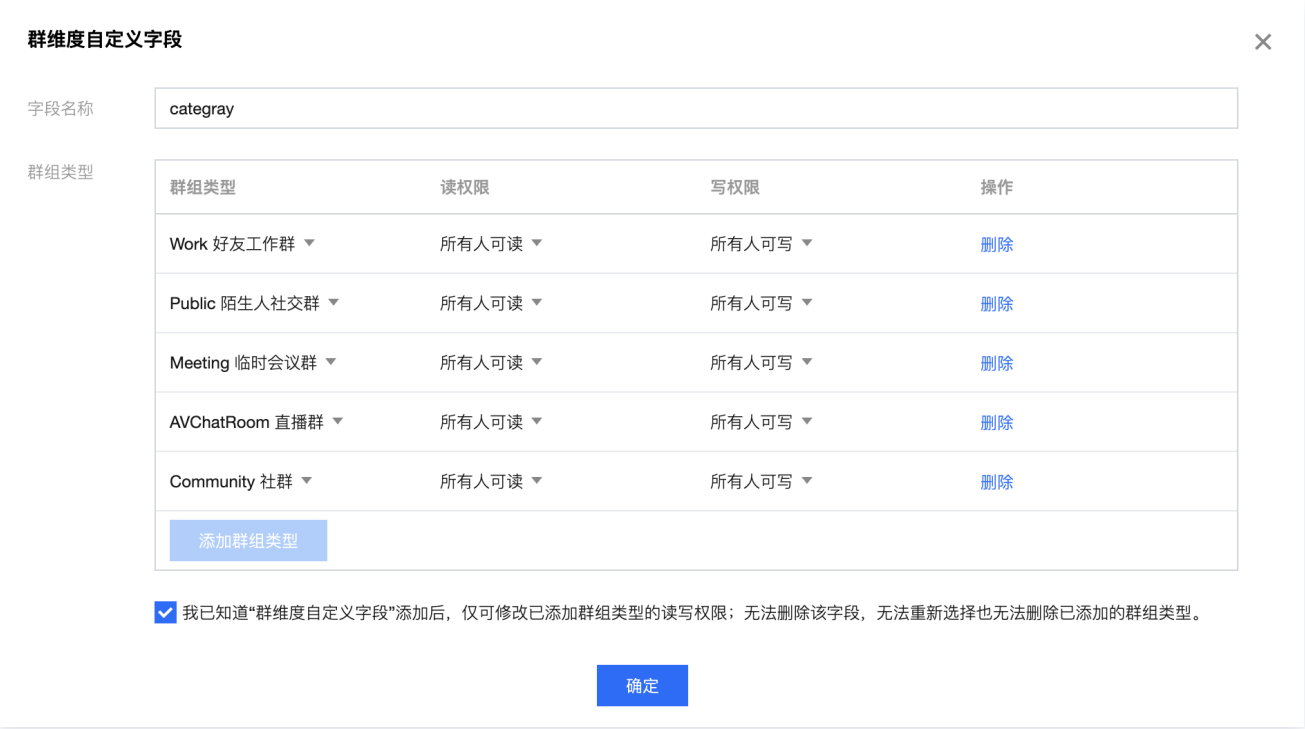
            @Override
            public void onError(int i, String s) {
              // 获取服务器列表失败
            }
});
```

在返回的 V2TIMGroupInfo 列表，代表的是服务器的基本信息。但是在基本信息中，并没有服务器的未读数以及自定义状态等相关信息。因此想要实现 Discord 一样的效果需要借助 IM 提供的其他 API 来实现，服务器列表的未读数，在后面的小节中讲到。

服务器分类

创建服务器时，每个服务器会创建默认的分类，创建后也可在创建新的分类，服务器在腾讯云 IM 中本质上是一个社群，因此可以通过设置社群的自定义字段来进行实现。使用群自定义字段可分为两步：

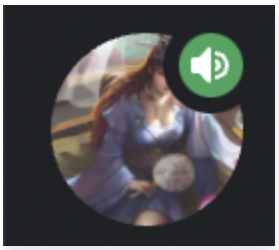
1. 在控制台开启自定义字段 key。
2. 使用客户端 SDK\服务端 API 进行读写。

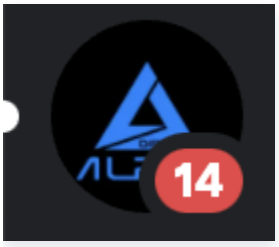


设置群自定义字段的 服务端 API 以及 客户端 SDK 。

> **注意**
> 社群为旗舰版或企业版能力，设置社群的自定义字段需要先购买旗舰版或企业版。

服务器消息未读数&自定义状态展示





上一小节提到在获取已加入的服务器列表 API 中，没有返回未读数以及服务器状态等信息。需要注意的是，我们不仅仅要获取到这个数据，还需要监听这个数据的变化从而及时的更新客户端 UI，由于服务器使用IM社群实现，且社群在 IM 中不会产生会话，因此需要统计所有公有的频道的会话以及私有的频道会话之和。通过 V2TIMTopicInfo 的 getUnreadCount 获取公有频道的未读数，由于私有频道由 work 群实现所以可通过 getConversation 来获取私有频道的未读数。

```
// 公有频道
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList,
new V2TIMValueCallback<List<V2TIMTopicInfoResult>>() {
                @Override
                public void onSuccess(List<V2TIMTopicInfoResult>
v2TIMTopicInfoResults) {

                }

                @Override
                public void onError(int i, String s) {
                }
});
// 私有频道
List<String> conversationIDList = new LinkedList();
conversationIDList.add("GROUP_$GROUPID");
V2TIMManager.getConversationManager().getConversationList(conversationID
List, new V2TIMValueCallback<List<V2TIMConversation>>() {
            @Override
```

```
            public void onError(int i, String s) {
                // 获取服务器对应的会话信息失败
            }

            @Override
            public void onSuccess(V2TIMConversationList
List<V2TIMConversation>) {
                // 获取服务器对应的会话信息成功
            }
});
```

对于服务器的自定义状态功能，可以通过设置服务器会话的自定义数据来实现，对应 API 为 setConversationCustomData 。

```
List<String> conversationIDList = new LinkedList();
String customData = "通话中"
V2TIMManager.getConversationManager().setConversationCustomData(conversa
tionIDList, customData, new
V2TIMValueCallback<List<V2TIMConversationOperationResult>>() {
            @Override
            public void onSuccess(List<V2TIMConversationOperationResult>
v2TIMConversationOperationResults) {
                // 设置群会话自定义数据成功
            }

            @Override
            public void onError(int i, String s) {
                // 设置群会话自定义数据失败
            }
});
```

当服务器会话的相关数据发生改变时，客户端需要试试更新 UI 的展示，对于监听服务器会话的改变，IM 提供了对应的事件监听函数 addConversationListener ，当以下信息修改时，会触发该回调函数。

1. 服务器消息增删改
2. 服务器消息未读数改变
3. 服务器自定义信息改变
4. 服务器置顶
5. 服务器接受消息配置改变
6. 服务器标记改变
7. 服务器分组改变

8. ...

```
V2TIMConversationListener conversationLister = new
V2TIMConversationListener() {
            @Override
            public void onSyncServerStart() {
            }

            @Override
            public void onSyncServerFinish() {
            }

            @Override
            public void onSyncServerFailed() {
            }

            @Override
            public void onNewConversation(List<V2TIMConversation>
conversationList) {
            }

            @Override
            public void onConversationChanged(List<V2TIMConversation>
conversationList) {
            }
            @Override
            public void onTotalUnreadMessageCountChanged(long
totalUnreadCount) {
            }

            @Override
            public void onConversationGroupCreated(String groupName,
List<V2TIMConversation> conversationList) {
            }

            @Override
            public void onConversationGroupDeleted(String groupName) {
            }

            @Override
            public void onConversationGroupNameChanged(String oldName,
String newName) {
            }
```

```
            @Override
            public void onConversationsAddedToGroup(String groupName,
List<V2TIMConversation> conversationList) {
            }

            @Override
            public void onConversationsDeletedFromGroup(String
groupName, List<V2TIMConversation> conversationList) {
            }
}
V2TIMManager.getConversationManager().addConversationListener(conversati
onLister);
```

综上通过 getJoinedCommunityList、getConversationList 接口以及 addConversationListener 回调，可以实现 Discord 的展示服务器列表的功能。

频道

在服务器内，可创建多个频道。如图所示，在该服务器下创建了四个频道，并且将四个频道放到了两个分类中。

对于频道，用户可以邀请用户加入以及对频道进行基本的设置。用户大多数的聊天是在频道内进行的，因此频道的能力在 Discord 中是最重要的。对应于腾讯云 IM，即是话题的能力。在 IM 社群支持在社群内创建话题的能力。

默认频道

Discord 会在创建服务器时，默认创建四个频道，使用腾讯云 IM 也可以实现这样的功能，具体流程如下：

1. 使用 创建群后回调 通知业务服务端创服务器成功。
2. 业务侧判断是否创建的社群，以免影响其他群相关的业务。
3. 根据服务器属性，在服务端创建话题 。

服务端创建话题主要参数如下：

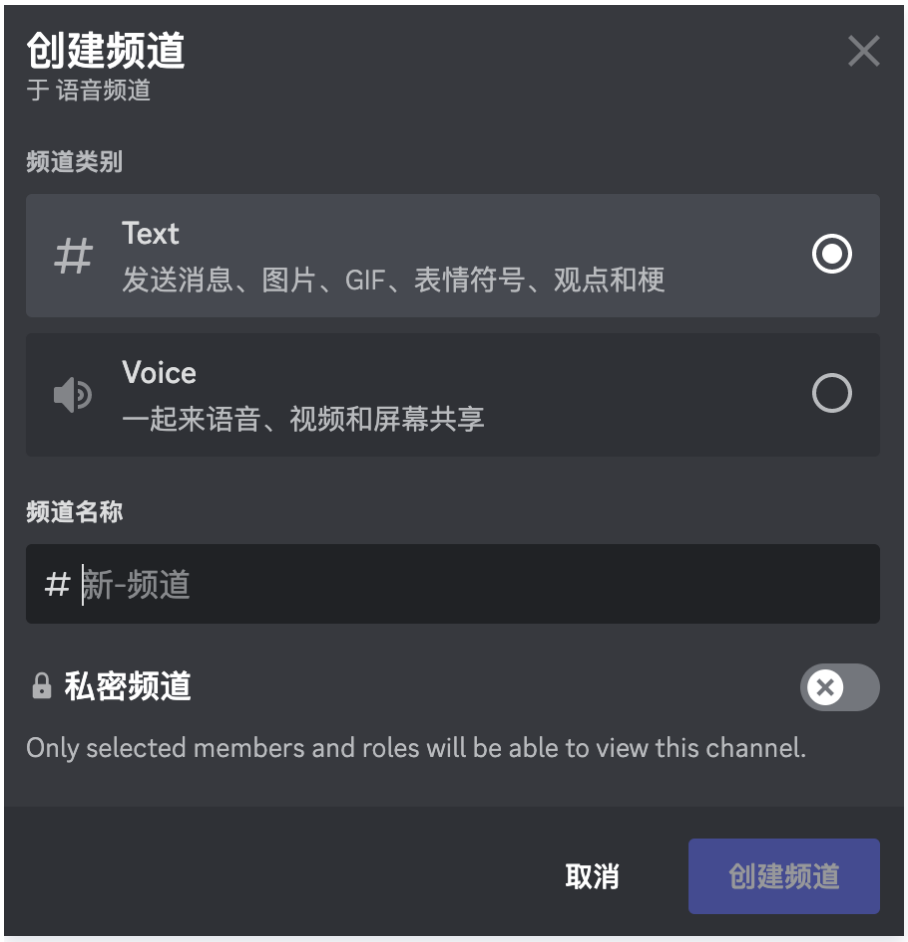
```
{
    "GroupId": "@TGS#_@TGS#cQVLVHIM62CJ", // 话题所属的群ID（必填）
    "TopicId": "@TGS#_@TGS#cQVLVHIM62CJ@TOPIC#_TestTopic",     // 用户自定义话题 ID（选填）
    "TopicName": "TestTopic",          // 话题的名称（必填）
    "From_Account": "test_user", // 创建话题的成员
    "CustomString": "This is a custom string",    // 自定义字符串
```

```
    "FaceUrl": "http://this.is.face.url", // 话题头像 URL（选填）
    "Notification": "This is topic Notification", // 话题公告（选填）
    "Introduction": "This is topic Introduction" // 话题简介（选填）
}
```

创建频道

在 Discord 客户端，用户可以在频道分类下创建频道，如图：

如上所分析，创建频道在 IM 中即为在社群中创建话题。并且在创建时，可以对话题进行分类，设置话题基本信息。

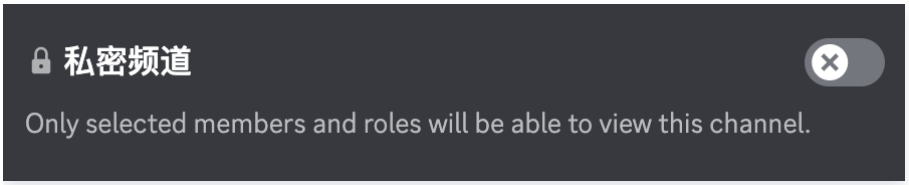


可以通过 createTopicInCommunity 来实现相关的功能。

```
String groupID = "服务器ID"
V2TIMTopicInfo info = new V2TIMTopicInfo();
info.setCustomString("{'categray':'游戏','type':'text'}") // 设置频道分类以及类型
// 这里可以设置V2TIMTopicInfo的具体信息
V2TIMManager.getGroupManager().createTopicInCommunity(groupID, info, new V2TIMValueCallback<String>() {
            @Override
            public void onSuccess(String s) {
                // 创建频道成功
            }

            @Override
            public void onError(int i, String s) {
                // 创建频道失败
            }
});
```

创建频道时，可以通过调用 V2TIMTopicInfo 的成员方法来设置频道的信息，频道分类以及频道类型，可以通过 setCustomString，进行设置。



在创建频道时，可设置频道是否是私密频道，私密频道和普通频道不同。

1. 用户加入服务器后，不会加入私密频道。
2. 私密频道需要服务器管理员邀请进入。

因此，我们可以使用一个 work 群来实现私密频道的功能，但是一个服务器绑定了哪些私密频道的信息需要业务侧来存储。

频道类型

Discord 在创建频道时，可选择频道类型为语音频道或者文字频道，文字频道即为普通的文字、表情、图片聊天，语音频道则为音视频聊天。需要注意的是，同一用户同一时间内只能在一个语音频道内。用户加入新的语音频道需要退出当前已加入的语音频道。

因此这里有4个注意点：

1. 创建频道时需设置频道类型，在创建频道小节中有讲到如何设置频道类型。
2. 用户加入语音频道时需判断是否已经在其他语音频道。
3. 语音频道是全局的。
4. 腾讯云 IM 暂时未提供用户是否在语音频道以及在哪个语音频道的 API，因此此部分的数据需要业务侧来维护。

针对第4点，开发者可以使用腾讯云 IM 提供的进群后回调以及退群回调，维护用户是否在语音频道的状态，并且将该状态存在业务侧存储内。需要注意的是，腾讯云 IM 的回调可能存在延迟，即用户退出语音频道后，短时间内仍可能不能加入其他语音频道。因此也可以用过客户端上报的方式来处理，即将用户的进退语音频道的状态实时上报给业务服务端。

邀请用户进入频道

用户进入频道有三种方式：

1. 邀请用户进入服务器，进入服务器的用户会进入所有的公开的频道。
2. 用户通过搜索进入服务器。
3. 私有频道通过服务器管理员邀请进入。

通过社群的特性可知，用户进入共有频道，只需加入服务器即可。

1. 支持精准搜索群 ID 加入。
2. 支持邀请人进入。
3. 支持主动申请进入，且无需审批。

频道设置

可以对频道进行静音设置以及通知相关设置，对应的 API 为 setAllMute 以及 setGroupReceiveMessageOpt 。

```
// 设置频道基本信息
V2TIMManager.getGroupManager().setTopicInfo(topicInfo, new
V2TIMCallback() {
    @Override
    public void onSuccess() {
        // 设置成功
    }

    @Override
    public void onError(int i, String s) {
        // 设置失败
    }
});
```

```
// 设置频道如何接受消息
String groupID = "topicid"
int opt = 0;
V2TIMManager.getMessageManager().setGroupReceiveMessageOpt(groupID, opt,
new V2TIMCallback() {
    @Override
    public void onSuccess() {

    }

    @Override
    public void onError(int i, String s) {

    }
```

```
});
```

频道消息列表

可通过 getHistoryMessageList 来获取频道的历史消息记录。

```
final V2TIMMessageListGetOption option = new
V2TIMMessageListGetOption();
option.setGroupID("频道ID");
option.setCount(20);
// 其他配置
V2TIMManager.getMessageManager().getHistoryMessageList(option, new
V2TIMValueCallback<List<V2TIMMessage>>() {
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // 获取频道历史消息成功
    }

    @Override
    public void onError(int code, String desc) {
        // 获取频道历史消息失败
    }
});
```

频道消息未读数

频道内消息未读数和服务器消息未读数不同，服务器未读数在会话信息中，而频道的未读数在频道的基本信息中，切通过腾讯云 IM 的群组回调 onTopicInfoChanged 来实时获取未读数。

```
String groupID = "服务器ID";
List< String > topicIDList= new LinkedList(); // 频道消息列表
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList,
new V2TIMValueCallback<List<V2TIMTopicInfoResult>>() {
    @Override
    public void onSuccess(List<V2TIMTopicInfoResult>
v2TIMTopicInfoResults) {
        // 获取频道信息，如频道id、频道名字、未读数等
    }

    @Override
    public void onError(int i, String s) {

    }
```

```
});
```

频道成员名单

加入服务器的用户默认会加入到所有的公有频道中，因此查看服务器的群成员列表即可。功能频道获取成员列表：

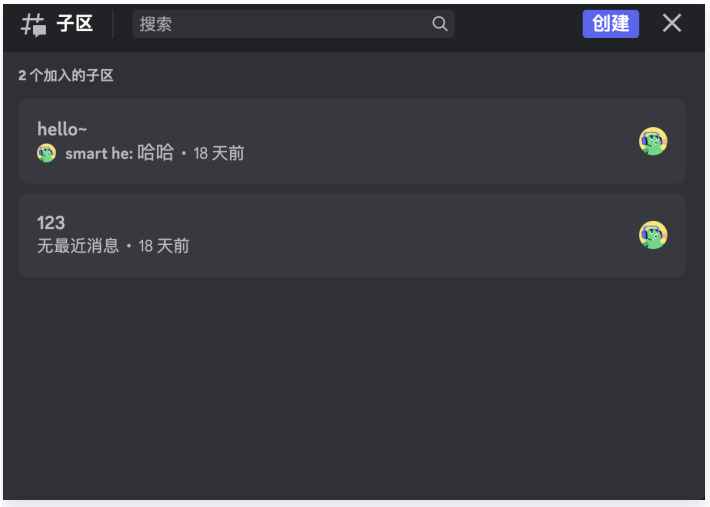
```
String groupID = "服务器ID";
int filter = 0; // 群成员角色 管理员 普通成员 ...
long nextSeq = 0;// 分页参数
V2TIMManager.getGroupManager().getGroupMemberList(groupID, filter,
nextSeq , new V2TIMValueCallback<V2TIMGroupMemberInfoResult>() {
    @Override
    public void onError(int i, String s) {
        CommonUtil.returnError(result,i,s);
    }

    @Override
    public void onSuccess(V2TIMGroupMemberInfoResult
v2TIMGroupMemberInfoResult) {

    }
});
```

私有频道获取群成员列表也是调用 getGroupMemberList，传入私有频道的群 ID 即可。

子区



子区是对于频道内消息进行进一步讨论的群组，用户可以集中浏览一个频道内用户讨论的重点，在频道内消息列表中，也可以看到子区的概要。子区在腾讯云IM里也是群的概念，一个子区即为一个群。

创建子区

子区的创建可以绑定频道内的一条消息，也可以单独创建，可以通过腾讯云 IM 提供的创建群 createGroup 接口创建子区,但需要注意以下几点：

1. 新创建的子区，服务器内的所有成员，默认都在子区内。
2. 子区创建后加入服务器的成员也都会加入到子区。
3. 后加入子区的用户能看到子区创建后的历史聊天记录。

综上，创建一个子区，需要创建群时，将服务器中群成员加入子区，用户加入服务器时，需要在用户进群后回调中，将用户加入到服务器所有子区中。这两个步骤建议是在业务服务端进行，需要用到的 API 如下：

1. 查询 服务器中成员 。
2. 创建群并设置群成员 。
3. 邀请用户进群 。

服务端回调为 进群后回调 。

子区数量

在频道列表中，可以获取到频道下的子区列表，因此需要绑定子区与服务器的多对一的关系。如果子区对应有历史消息，也要绑定子区与消息的一对一关系，腾讯云 IM 暂未提供绑定上述关系的能力，因此需要业务侧额外维护。通过业务提供的 HTTP 接口来获取子区的数量以及子区列表。通过发送在线消息，来实时刷新子区列表。

子区概要

当对一条消息进行创建子区时，会在消息列表中看到子区的消息概要：

1. 该子区有多少条消息。
2. 该子区中 lastMessage 的相关信息。
3. 信息需要实时显示在消息列表中。

要实现子区的消息概要能力，需要结合群内发消息后回调、以及消息编辑的能力。用户在子区发消息后，IM 服务触发发消息后回调，将相关信息同步给业务侧，业务进行消息计数，并且维护 lastMessage 相关信息，同时将信息通过消息编辑 API 保存到消息上。

消息相关

消息反馈

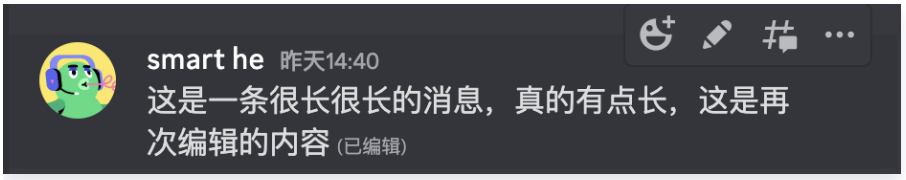
消息反馈即为用户对消息进行扩展，如图所示：

因为所有的消息类型均可以进行编辑和反馈，并且需要支持社群，因此我们建议将数据保存在 cloudCustomData 字段上，详细的的数据存储格式可以参见：

```
{
  "reaction": {
    "simle":["user1","user2"]
  }
}
```

```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new
V2TIMCompleteCallback<V2TIMMessage>() {
    @Override
    public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
        // 消息修改完成
    }
});
```

消息编辑



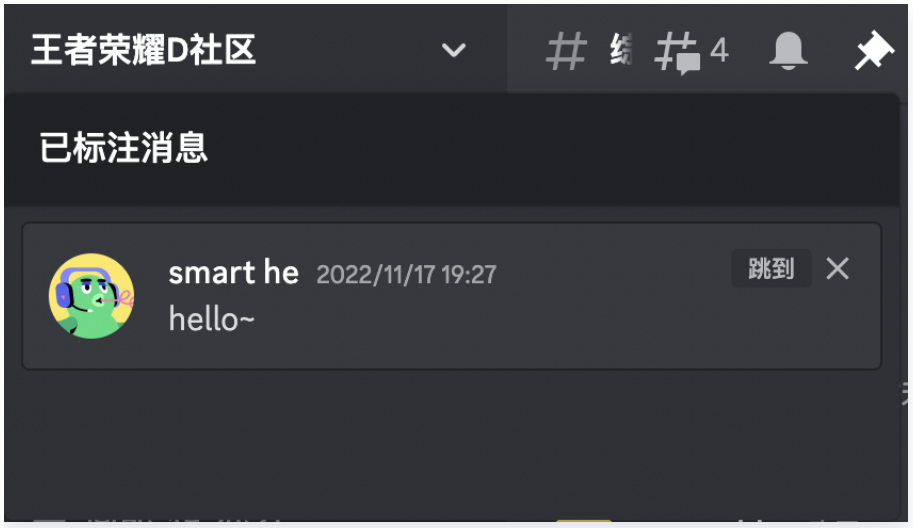
消息编辑与消息反馈原理一样，仅仅设置的自定义数据上有些区别，同样为了所有消息都能够适用消息编辑，建议编辑 cloudCustomData 字段，数据格式如下：

```
{
```

```
  "isEdited": true
}
```

```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new
V2TIMCompleteCallback<V2TIMMessage>() {
    @Override
    public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
        // 消息编辑完成
    }
});
```

消息标注



消息标注即用户在群聊中将消息进行标注，其他用户可以看到被标注的消息。由于社群暂不支持群属性，因此这里使用自定义消息实现消息标注能力。

使用群自定义消息需要现在 腾讯云 IM 控制台 进行配置如下：

其次进行设置：

```
V2TIMGroupInfo info =  new V2TIMGroupInfo();
info.setCustomInfo("pin data");
V2TIMManager.getGroupManager().setGroupInfo(info, new V2TIMCallback() {
    @Override
    public void onError(int i, String s) {
        // 设置失败
    }

    @Override
    public void onSuccess() {
        // 设置成功
    }
});
```

设置成功后群成员可收到 onMemberInfoChanged 事件，同时，未在线的用户也可以在通过群资料获取到标注的消息内容：

```
List< String > groupIDList = new LinkedList();
V2TIMManager.getGroupManager().getGroupsInfo(groupIDList, new
V2TIMValueCallback<List<V2TIMGroupInfoResult>>() {
    @Override
    public void onError(int i, String s) {

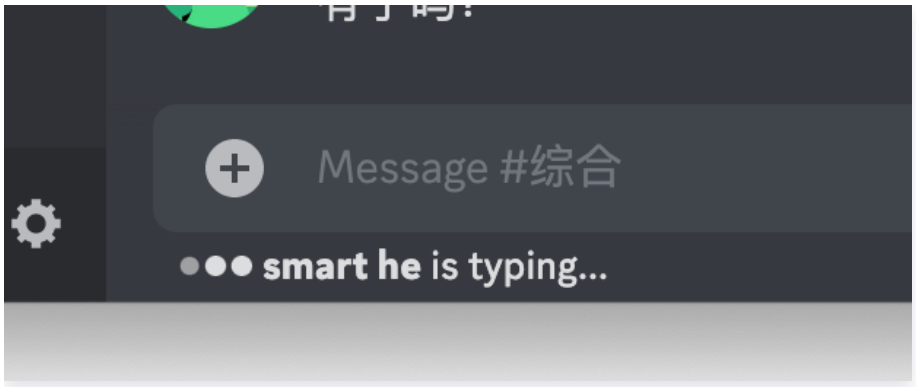
    }

    @Override
    public void onSuccess(List<V2TIMGroupInfoResult>
v2TIMGroupInfoResults) {

    }
});
```

需要注意的是，目前自定义字段只能配置在服务器上，不能配置在频道上。

正在输入中状态



当好友正在输入时，其他用户端可以看见用户正在输入中的状态，如上图所示，可以使用腾讯云IM提供的在线消息提供，此消息：

1. 仅在线用户才会收到。
2. 不会存入消息漫游。

同时，为了优化性能，我们也可以有一些优化如：

- 20s之内发送给消息的用户双方才会触发正在输入中状态消息的发送。
- 同一文本消息不触发多次在线消息发送。

私信

私信即 Discord 用户与用户之间可以发送消息，不管用户之间是否有好友关系。

- 如果用户没有好友关系，发送消息除了需要知道用户 ID 之外，还需要在腾讯云 IM 控制台关闭 发送消息检测 关系链的选项，否则会发送消息失败。
- 用户也可以先通过 addFriend 接口添加好友，通过 getFriendList 来获取自己的好友列表。

相关代码如下：

```
// 添加好友
V2TIMManager.getFriendshipManager().addFriend(info, new
V2TIMValueCallback<V2TIMFriendOperationResult>() {
            @Override
            public void onError(int i, String s) {
                // 添加好友失败
            }

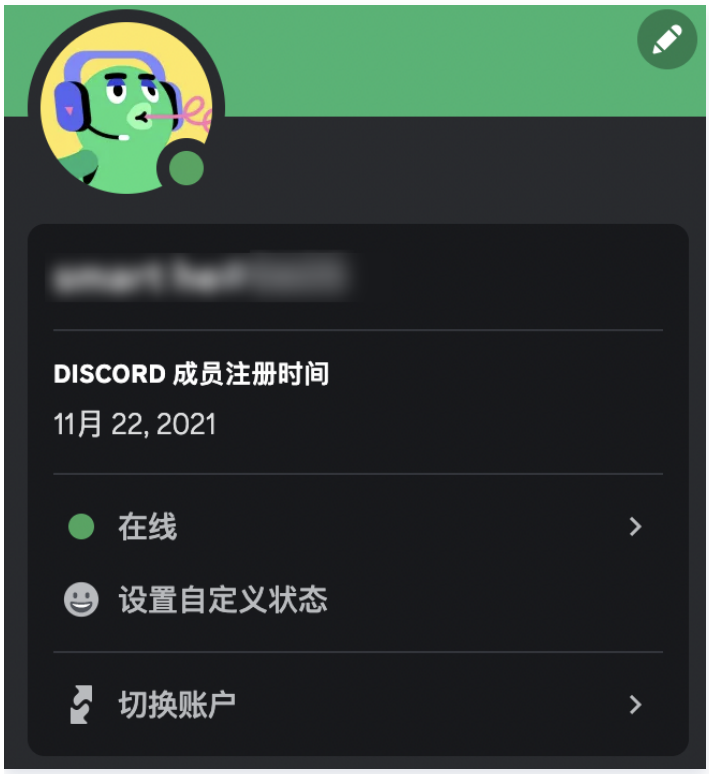
            @Override
            public void onSuccess(V2TIMFriendOperationResult
v2TIMFriendOperationResult) {
                // 添加好友成功
            }
});
```

```
// 获取好友列表
V2TIMManager.getFriendshipManager().getFriendList(new
V2TIMValueCallback<List<V2TIMFriendInfo>>() {
            @Override
            public void onError(int i, String s) {
                // 获取好友列表失败
            }

            @Override
            public void onSuccess(List<V2TIMFriendInfo>
v2TIMFriendInfos) {
                // 获取好友列表成功
            }
});
```

个人中心

在线状态



Discord 用户面板的在线状态能力，可以通过腾讯云 IM 提供的在线状态 API 来实现：

- 通过 setSelfStatus 来设置自己的自定义在线状态，用户的自己的在线\离线状态是 IM 设置的，开发者不能更改。
- 通过 getUserStatus 来获取好友的在线状态。
- 通过 onUserStatusChanged 来监听好友的在线状态改变。

相关代码如下：

```
// 设置个人在线状态
V2TIMUserStatus customStatus = new V2TIMUserStatus();
V2TIMManager.getInstance().setSelfStatus(customStatus, new
V2TIMCallback() {
    @Override
    public void onSuccess() {

    }

    @Override
    public void onError(int i, String s) {

    }
});
```

```
// 获取好友在线状态
List<String> userIDList = new LinkerList();
V2TIMManager.getInstance().getUserStatus(userIDList, new
V2TIMValueCallback<List<V2TIMUserStatus>>() {
    @Override
    public void onSuccess(List<V2TIMUserStatus> v2TIMUserStatuses) {

    }

    @Override
    public void onError(int i, String s) {

    }
});
```

个人信息相关

个人信息相关的功能可通过腾讯云 IM 提供的资料关系链相关的 API 实现：

- 获取个人资料 getUsersInfo 。
- 设置个人资料 setSelfInfo 。

```
// 设置个人资料
final V2TIMUserFullInfo userFullInfo = new V2TIMUserFullInfo();
```

```
V2TIMManager.getInstance().setSelfInfo(userFullInfo, new V2TIMCallback()
{
    @Override
    public void onError(int i, String s) {

    }

    @Override
    public void onSuccess() {

    }
});
```

```
// 获取用户资料
List<String> userIDList = new LinkedList();
V2TIMManager.getInstance().getUsersInfo(userIDList, new
V2TIMValueCallback<List<V2TIMUserFullInfo>>() {
    @Override
    public void onError(int i, String s) {

    }

    @Override
    public void onSuccess(List<V2TIMUserFullInfo> v2TIMUserFullInfos) {

    }
});
```

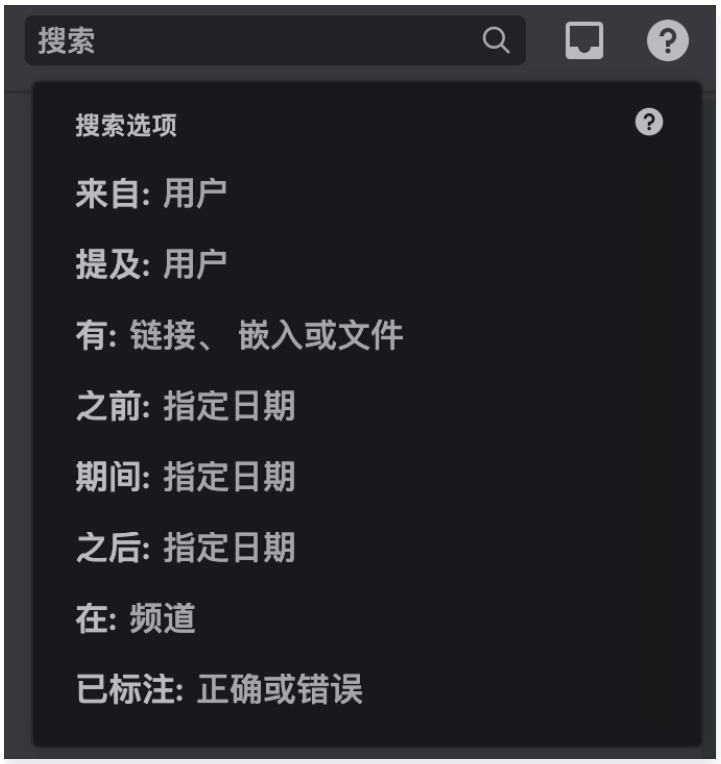
开通云端审核

在消息发送、资料修改等场景中，有可能会扩散不合适的内容，特别是与敏感事件/人物相关、黄色不良内容等令人反感的内容，不仅严重损害了用户们的身心健康，还有可能违法并导致业务被监管部门查封。
即时通信 IM 支持 云端审核（反垃圾信息）功能，可针对不安全、不适宜的内容进行自动识别、处理，为您的产品体验和业务安全保驾护航。您可直接前往 云端审核控制台 开通免费试用，每个应用可免费试用7天（含10000次免费抵扣量）。

其他

搜索功能

Discord 的搜索能力如下：



腾讯云 IM 提供了丰富的搜索能力，包括：

- 搜索消息 searchLocalMessages 。
- 搜索群组 searchGroups 。
- 搜索群成员 searchGroupMembers 。
- 搜索好友 searchFriends 。

详细的使用 API 可参见官文档 搜索部分 。

离线推送功能

在用户离线时，腾讯云 IM 的在线消息触达不到用户，这时便需要终端设备厂商提供的离线推送能力，腾讯云 IM 接入离线推送也十分方便，详情可参见 离线推送 。

联系我们

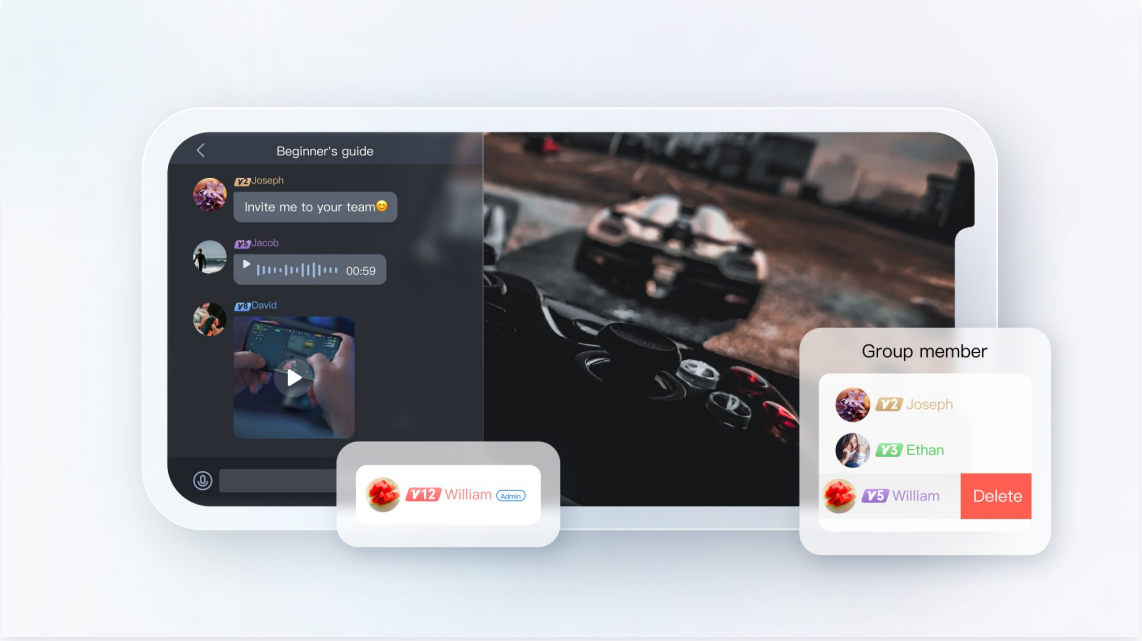
如果您有任何使用问题，请 联系我们 。您也可以使用 Android 手机扫描下图二维码，下载 Demo 直接体验部分功能。

9:41
腾讯IM官方社群
满意度调查
公告
新人须知
新人指引
产品通知
文字交流区
话题分组
语音交流区
一起语音吧
直播活动区
一起看直播
多种社群
Android
扫码体验

互动游戏场景方案

最近更新时间：2025-01-15 21:46:22

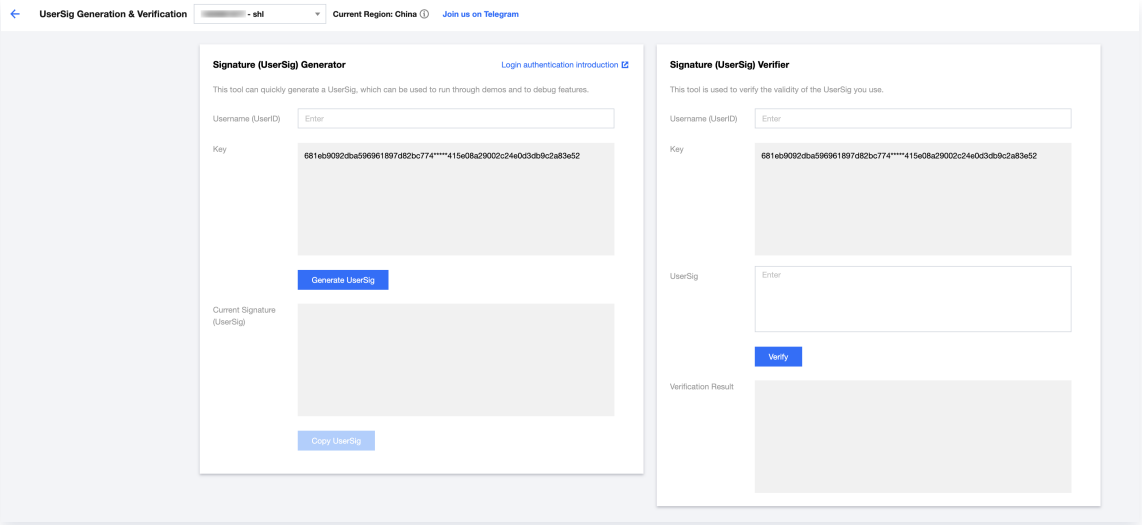
在游戏中，即时通信需求很常见，在多人游戏中即时聊天也成为了必不可少的功能。由于游戏平台本身所涉及到的多种群组类型、自定义消息类型（如游戏内道具赠送、交易等业务）、全球接入等需求往往比较复杂，本文梳理了在搭建游戏聊天时过程中常见的需求的实现方法，以及可能遇到的问题、需要注意的细节等，希望能帮助开发者们快速的理解业务，实现需求。



准备工作

使用密钥计算 UserSig

在 IM 的账号体系中，用户登录需要的密码由用户服务端使用 IM 提供的密钥计算，用户可参见 UserSig计算 文档，在开发阶段，为了不阻塞客户端开发，也可在 控制台计算 UserSig ，如下图所示：



配置管理员账号

在管理游戏内即时通信过程中，可能需要管理员向游戏发送邮件公告、管理临时组队消息等，这时就需要使用 即时通信 IM 服务端 API 来进行相应的处理，调用服务端 API 前需要 创建 IM 管理员账号，IM 默认提供一个 UserID 为 administrator 的账号供开发者使用，开发者也可以根据业务的场景，创建多个管理员账号。需要注意的是，IM 最多创建五个管理员账号。

配置回调地址以及开通回调

在实现游戏内组队等需求时，需要用到 IM 的回调模块，即 IM 后台在某些特定的场景回调开发者业务后台。开发者只需要提供一个 HTTP 的接口并且配置在 控制台 > 回调配置 模块即可，如下图所示：

集成客户端 SDK

在准备工作都完成好后，需要将即时通信 IM 客户端 SDK 集成到用户项目中去。开发者可以根据自己业务需要，选择不同的集成方案。可参见 快速集成系列文档 。
接下来文章梳理了游戏中集成IM时常见的功能点，提供实践方案供开发者参见，并附上相关实现代码。

游戏聊天室各功能开发指引

用户资料

常见用户资料

游戏业务中保存的常见用户资料可分为基本信息资料和其他信息资料。

基本信息	其他信息
用户名，性别，生日，等级，角色，手机号等	其他游戏内需要的资料

资料存储

游戏业务内有众多的用户，而存储庞大的用户资料也是较大的难点。腾讯云 IM 开放了用户资料托管能力，提供资料相关的一套完整解决方案。下面对比保存到 腾讯云 IM 与保存到业务后台的区别。

对比项	IM	业务后台
存储容量	可自动扩容/缩容	容量有限，增减容量较困难
用户资料	提供标配字段和自定义字段，字段的长度和命名有限制	可自行定义，更加灵活
资料读写	提供简单易用的服务接口和帮助指引	需自行开发
接口	有调用接口频率限制：最高200次/秒	接口调用等能力可按照自己需求自行开发
安全性	可异地容灾、多地部署	需自行维护

综上，使用 IM 用户资料托管服务除可获得资料的存储、读写能力外还有以下优点：

1. IM 提供异地容灾、多地部署和自动扩容/缩容的能力，帮助您从服务器宕机、多拷贝主从复制和扩容缩容等复杂处理流程中得到完全地解放。
2. IM 提供业界通用的业务处理流程，帮助您在用户资料的业务逻辑上彻底地解放。
3. IM 提供专业的运营流程和运营团队，全年99.99%的稳定服务质量，帮助您为用户提供具有稳定口碑的服务。
4. IM 提供简单易用的服务接口和快捷接入的帮助指引，全程为您提供星级服务。

IM 存储用户资料的方式

IM 存储方案包含资料的存储和其读写能力。下面介绍 IM 存储用户资料、好友资料和拓展资料的方式。所有资料均以 `Key-Value` 形式表示。其中 `Key` 为 `String` 类型，仅支持英文大小写字母、数字、下划线。 `Value` 有以下几种类型：

类型	说明
uint64_t	整数（自定义字段不支持）
string	字符串，长度不得超过500字节
bytes	一段 buffer，长度不得超过500字节
string 数组	字符串数组，每个字符串长度不得超过500字节，仅供好友表的 Tag_SNS_IM_Group 字段使用

- **用户资料**：用户资料包含标配字段和自定义字段。自定义字段可见下文的拓展资料。目前即时通信 IM 支持的用户资料标配字段可参见 用户资料标配资料字段 。
- **好友资料**：好友资料支持标配好友字段和自定义好友字段。即时通信 IM 好友列表最多允许添加3000个好友。其中，标配好友字段如下：

字段名称	类型	描述
Tag_SNS_IM_Group	Array	好友分组，为字符串数组
Tag_SNS_IM_Remark	String	好友备注
Tag_SNS_IM_AddSource	String	加好友来源
Tag_SNS_IM_AddWording	String	加好友附言
Tag_SNS_IM_AddTime	Integer	加好友时间戳

更详细内容请参见 好友标配字段 。

- **自定义资料**：自定义资料字段可以通过即时通信 IM 控制台 > 应用配置 > 功能配置 申请，提交申请后，自定义资料字段将在5分钟内生效。
- 更多关于用户资料管理，请参见 用户资料管理 。
- 更多关于好友关系链自定义资料管理，请参见 好友自定义字段 。

IM 用户资料存储限制

- **业务特性限制**
- 存储数据：string 和 buffer 类型的存储长度不得超过500字节。
- 自定义字段：自定义字段的关键字必须是英文字母且长度不得超过8字节，自定义字段的值最长不能超过500字节。
- 好友关系链：单个用户支持3000个好友。
- **接口相关限制**
- 账号管理：单次最多导入100个用户名，一次请求最多可以查询500个用户状态。
- 其他调用频率：最高200次/秒。

更多关于使用限制请参见 使用限制 。

邮件系统

现在游戏中，邮件系统几乎是必备的功能。邮件包含文字消息，也可以包含游戏道具、奖励等邮件附件。邮件可以发送给一个人、也可以像发放活动奖励群发邮件 。下面从玩家接收邮件、邮件列表、邮件未读计数、全员邮件、邮件有效期等几个方面详细介绍邮件系统的功能实现。

收发邮件

- **玩家接收邮件：**当系统邮件发送成功并且玩家正处联网状态时，玩家能够正确接收到系统邮件。接收到的邮件可通过获取邮件会话的消息列表取得历史/最新邮件，同时可以增加/删除接收新消息的回调监听接收所有类型的消息（包含文本、自定义、富媒体消息等）。以Unity为例的示例代码如下所示：

```
// 设置接收消息事件监听器
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string
user_data)=>{
foreach(Message message in messages)
{
  foreach (Elem elem in message.message_elem_array)
  {
    // 有下一个消息
    if (elem.elem_type == TIMElemType.kTIMElem_Text)
    {
       string text = elem.text_elem_content;
    }
  }
}
})
// 监听 `RecvNewMsgCallback`回调，在其中接受消息
// 希望停止接收消息，调用 `RemoveRecvNewMsgCallback`移除监听。该步骤非必需，可
按照业务需求调用。
```

关于更多有关内容请参见 Unity-接收消息 。

- **系统发送邮件：**系统向用户发系统邮件可以通过 服务端 API 的几种方式，下面列出不同方法的特点：

方法	特点	应用场景
单发单聊消息	向指定账号发消息，接收方看到的发送者不是管理员，而是管理员指定的账号	向某个特定用户发送消息，如段位赛奖励等
批量发单聊消息	支持一次最多对500个用户单发消息，最高调用频率为200次/秒	向某些特定用户发送消息，因无需创建组，所以使用比较灵活，但若发送用户较多则需要分批发送
在群组中发送普通消息	向群组发送普通消息，需要将用户添加到同一个组内	向较多用户发送消息可以系统创建群组发送普通消息，群组最大人数限制为 社交群(community) 10万人
全员推送服务	向 App 内全员推送消息，可指定用户标签和属性发送消息	向 App 内全员推送消息或人数非常多且有特征属性时可使用，如活动推送邮件等

说明

> “全员推送” 为 消息推送插件的其中一项功能，需进入 IM 控制台 > 插件市场 > 消息推送，单击 **立即购买** 或 **免费试用**，成功购买或领取试用后，全员推送功能将自动开启。
> **注意：**
> 消息推送插件**试用或购买到期后，将自动停止提供推送服务（包括普通消息离线推送、全员推送等服务）**。为避免影响您业务正常使用，请提前购买/续费。

其中，在群组中发送普通消息的请求包基础形式示例如下所示：

```
{
    "GroupId": "@TGS#2C5SZEAEF",
    "Random": 8912345, // 随机数字，五分钟数字相同认为是重复消息
    "MsgBody": [ // 消息体，由一个 element 数组组成，详见字段说明
        {
            "MsgType": "TIMTextElem", // 文本
            "MsgContent": {
            "Text": "red packet"
            }
        },
        {
            "MsgType": "TIMCustomElem", // 自定义
            "MsgContent": {
            "Data": "message",
            "Desc": "notification",
            "Ext": "url",
            "Sound":"dingdong.aiff"
            }
        }
    ],
}
```

`MsgBody`（消息体）为一个消息数组，可以将文本消息和自定义消息放入 MsgBody 中发送消息。

邮件列表

历史邮件列表存储同消息存储，可分为 `C2C 单聊历史消息存储` 和 `群聊历史消息存储`。由于群聊最小人数为2人，可以将管理者指定账号与收到邮件的用户创建新的组进行存储。

> **说明**
> 体验版和专业版存储时长为7天，旗舰版或企业版为 30 天。专业版和旗舰版或企业版支持延长时长。您可登录 控制台 修改相关配置。
> 延长历史消息存储时长是付费增值服务，具体计费说明请参见 增值服务计费说明。

在网络正常下会拉取最新的云端数据，若网络异常则返回本地存储的历史消息。本接口支持分页拉取。拉取历史邮件列表的 Unity 示例代码如下：

```
// 拉取单聊历史消息
// 首次拉取，msg_getmsglist_param_last_msg 设置为 null
// 再次拉取时，msg_getmsglist_param_last_msg 可以使用返回的消息列表中的最后一条
消息
var get_message_list_param = new MsgGetMsgListParam
{
    msg_getmsglist_param_last_msg = LastMessage
};
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id,
TIMConvType.kTIMConv_C2C, get_message_list_param, (int code, string
desc, string user_data) => {
// 处理回调逻辑
});
```

更多拉取历史消息内容，请参见 历史消息-Unity 。

邮件未读计数

一个用户-系统邮件的记录相当于一个聊天中的会话。腾讯云 IM 提供会话未读计数功能，提醒用户尚未阅读消息。用户点进该会话后退回会话列表时，未读消息数会被清空。Unity示例代码如下：

```
// 获取全部未读数
TIMResult res = TencentIMSDK.ConvGetTotalUnreadMessageCount((int code,
string desc, GetTotalUnreadNumberResult unread, string user_data)=>{
 // 处理异步逻辑
});

// 未读计数变更通知
TencentIMSDK.SetConvTotalUnreadMessageCountChangedCallback((int
total_unread_count, string user_data)=>{
 // 处理回调逻辑
});

// 清空所有会话的未读消息数
TIMResult res = TencentIMSDK.MsgMarkAllMessageAsRead((int code, string
desc, string user_data)=>{
 // 处理异步逻辑
});
```

更多内容请参见 Unity-会话未读消息数 。

全员邮件

全员邮件相当于对游戏内所有玩家发送邮件消息。腾讯云 IM 提供服务端 `全员推送` 功能。示例代码如下：

```
https://console.tim.qq.com/v4/all_member_push/im_push?usersig=xxx&identifier=admin&sdkappid=88888888&random=99999999&contentype=json
```

请求包示例如下：

```
{
    "From_Account": "admin",
    "MsgRandom": 56512,
    "MsgLifeTime": 120, // 离线保存120s（2分钟）
    "MsgBody": [
        {
        "MsgType": "TIMTextElem",
        "MsgContent": {
            "Text": "hi, beauty"
            }
        }
    ]
}
```

全员推送可以设置消息离线存储时间，这样即便某些用户不在线，但在离线存储的时间范围内，不在线用户也能收到消息。若需要设置离线保存时间，可设置 `MsgLifeTime` ，单位秒，最多保存7天（604800s）。默认为0，表示不离线存储。
更多相关全员推送的内容请参见 全员推送 。

邮件有效期

对于历史邮件，邮件存储有效期为：体验版/专业版为7天，旗舰版或企业版为30天，专业版和旗舰版或企业版支持延长时长。对于历史存储的更多内容请参见 历史消息存储时长配置 。
此外，在服务端发送消息时，可通过在请求包内设置 `MsgLifeTime` 设置消息离线保存时长（最长为7天）。若该字段为0，则表示消息只发在线用户，不离线保存。 更多内容请参见 服务端 API 。

临时组队

多人联网游戏中临时组队必不可少。下面从组队场景和后台、队伍内成员分别所需获取的队内信息讲解临时组队的内容。

组队场景

组队场景一般分为如下几点：创建组队、退出临时组队、成为队长、邀请加入组队、解散组队等。下面通过一些代码示例分别解释不同场景的实现方法。

- **游戏开始前创建组队**：当第一个人进入游戏时，在服务端自动创建群组并可以设置最大群成员数量。请求时如果指定了群组或群成员，那么在创建时群主或群成员回自动加入到该群中。请求 URL 示例如下：

```
https://console.tim.qq.com/v4/group_open_http_svc/create_group?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&content
type=json
```

请求包基础形式示例如下：

```
{
  "Owner_Account": "leckie", // 群主的 UserId（选填）
  "Type": "Public", // 群组类型:
Private/Public/ChatRoom/AVChatRoom/Community
  "Name": "TestGroup", // 群名称（必填）
  "MaxMemberCount":5 // 最大群成员数量（选填）
}
```

> **说明**
> App 中 同时存在的所有群组数量最多为10万，若超过10万则需要支付一定费用。详情请参见 价格说明 。更多关于服务端创建群组请参见 创建群组 。

- **增加群成员**：创建群聊后后续有新的玩家进入游戏则需要在原来群组的基础上添加新的群成员。请求 URL 示例如下：

```
https://console.tim.qq.com/v4/group_open_http_svc/add_group_member?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&content
type=json
```

请求包如下：

```
{
  "GroupId": "@TGS#2J4SZEAEL", // 要操作的群组（必填）
  "MemberList": [ // 一次最多添加300个成员
      {
          "Member_Account": "tommy" // 要添加的群成员ID（必填）
      },
      {
          "Member_Account": "jared"
```

```
  }]
}
```

更多内容请参见 增加群成员 。

- **组队成功回调**：当在创建群组时设置了最大群成员数量，则当组队需要人数全部进群后可以开始游戏。可以通过 `群组满员之后回调` 感知并开始游戏。请求 URL 示例如下：

```
https://www.example.com?
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&C
lientIP=$ClientIP&OptPlatform=$OptPlatform
```

请求包示例如下：

```
{
  "CallbackCommand": "Group.CallbackAfterGroupFull", // 回调命令
  "GroupId": "@TGS#2J4SZEAEL" // 群组 ID
}
```

更多相关内容请参见 群组满员之后回调 。

- **新成员入群通知**：当有新的玩家进入游戏（群聊）后，通过 `新成员入群之后回调` 可通知其他群成员入群成功。请求 URL 示例如下：

```
https://www.example.com?
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&C
lientIP=$ClientIP&OptPlatform=$OptPlatform
```

请求包示例如下：

```
{
  "CallbackCommand": "Group.CallbackAfterNewMemberJoin", // 回调命令
  "GroupId" : "@TGS#2J4SZEAEL",
  "Type": "Public", // 群组类型
  "JoinType": "Apply", // 入群方式：Apply（申请入群）；Invited（邀请入群）
  "Operator_Account": "leckie", // 操作者成员
  "NewMemberList": [ // 新入群成员列表
      {
          "Member_Account": "jared"
      },
      {
          "Member_Account": "tommy
      }
```

```
  ]
}
```

> **说明**
> 要启用回调，必须配置回调URL，并打开本条回调协议对应的开关。配置方法参见 第三方回调配置 文件。更多相关内容请参见 新成员入群之后回调 。

- **游戏途中退出组队**：当玩家自主退出游戏或由于网络问题退出游戏时，服务端可以通过 `群成员离开之后回调` 通知组内其他玩家有人退出群聊，也可以对因网络问题退出游戏的玩家发送消息。群成员离开之后回调 的请求 URL 示例如下：

```
https://www.example.com?
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&C
lientIP=$ClientIP&OptPlatform=$OptPlatform
```

请求包示例如下：

```
{
  "CallbackCommand": "Group.CallbackAfterMemberExit", // 回调命
  "GroupId": "@TGS#2J4SZEAEL", // 群组 ID
  "Type": "Public", // 群组类型
  "ExitType": "Kicked", // 成员离开方式：Kicked-被踢；Quit-主动退群
  "Operator_Account": "leckie", // 操作者
  "ExitMemberList": [ // 离开群的成员列表
      {
          "Member_Account": "jared"
      },
      {
          "Member_Account": "tommy"
      }
  ]
}
```

- **游戏结束后解散群聊**：游戏结束后，服务端可直接解散群聊。解散群聊 的请求 URL 示例如下：

```
https://console.tim.qq.com/v4/group_open_http_svc/destroy_group?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&content
type=json
```

请求包示例如下：

```
{
  "GroupId": "@TGS#2J4SZEAEL"
}
```

对于临时组队内语音或视频聊天，内容比较复杂，下文有详细说明。

后台获取队内情况

- **获取群内详细信息**：对于群内详细信息（包含当前群成员数量，群内成员基本信息等）可以通过服务端 API `获取群详细资料` 获取。可以在请求包中设定Filter字段拉取置顶的信息。示例代码如下所示：

```
https://console.tim.qq.com/v4/group_open_http_svc/get_group_info?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&content
type=json
```

请求包示例如下：

```
{
  "GroupIdList": [ // 群组列表（必填）
      "@TGS#1NVTZEAE4",
      "@TGS#1CXTZEAET"
  ]
}
```

更多详细信息请参见 获取群详细资料 。

- **获取群成员详细信息**：若只需要群成员信息（包含自定义字段），则可使用 `获取群成员详细资料` 获取到群成员的信息。示例代码如下所示：

```
https://console.tim.qq.com/v4/group_open_http_svc/get_group_member_inf
o?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&content
type=json
```

请求包示例如下：

```
{
  "GroupId":"@TGS#1NVTZEAE4"  // 群组 ID（必填）
}
```

返回字段中 `MemberNum` 可获取本群组的群成员总数， `AppMemberDefinedData` 可获取群成员自定义字段信息。

> **说明**
> 该接口不支持直播群（AVChatRoom）。更多详细信息请参见 获取群成员详细资料 。

队内成员获取队内情况

对于队内成员，可通过 `获取群成员资料` 获取到队内其他成员的资料，如成员的角色、准备状态等。示例代码如下：

```
GroupGetMemberInfoListParam param = new GroupGetMemberInfoListParam
{
  group_get_members_info_list_param_group_id = "group_id",
  group_get_members_info_list_param_identifier_array = new List<string>
  {
    "user_id"
  }
};
TIMResult res = TencentIMSDK.GroupGetMemberInfoList(param, (int code,
string desc, GroupGetMemberInfoListResult result, string user_data)=>{
 // 处理异步逻辑
});
```

> **说明**
> - 详细内容请参见 Unity-获取群成员资料 。
> - 其他关于群信息如 `群满员开始游戏` 、 `有成员加入/退出组队` 等可参见 组队场景 并使用回调通知到群内。
> - 更多有关群组选择和关于群组的其他事项，请参见 群组系统 。更多有关**群组管理的控制台指南**，请参见 群组管理控制台指南 。

敏感信息过滤

在游戏场景中，用户很可能会发送不合适的内容，特别是与敏感事件/人物相关、黄色不良内容等令人反感的内容，不仅严重损害了用户们的身心健康，更很有可能违法并导致应用被监管部门查封。
即时通信 IM 支持内容审核（反垃圾信息）功能，可针对不安全、不适宜的内容进行自动识别、处理，为您的产品体验和业务安全保驾护航。可以通过云端审核方式来实现：

- 云端审核功能： 在服务端检测由单聊、群聊、资料场景中产生的文本、图片、音频、视频内容，支持针对不同场景的不同内容分别配置审核策略，并对识别出的不安全内容进行拦截。此功能已提供默认预设拦截词库和审核场景，只需在 IM 控制台打开功能开关，即可直接使用。

自定义消息类型（如道具赠送、交易）

在游戏聊天环境中，不仅只使用文本消息、表情消息、语音消息等简单的消息类型。自定义消息类型可提供开发者自定义客制化消息内容格式的功能，可通过自定义消息类型完成游戏聊天室内游戏道具的赠送、交易等更多的聊天室功能。
腾讯云 IM 提供 9 种基本消息类型，包含文本消息、表情消息、地理位置消息、图片消息、语音消息、文件消息、短视频消息、系统通知和自定义消息。其中除了自定义消息之外的其他消息格式已经确定，只需要填入相应的信息即可。更多关于消息类型的具体描述请参见 消息类型，关于消息类型的格式请参见 消息格式描述 。
服务端可通过 单发单聊消息 、 批量发单聊消息 、 在群组中发送普通消息 向用户发送自定义消息。在群组中发送普通消息的基础请求包示例如下：

```
{
    "GroupId": "@TGS#2C5SZEAEF",
    "Random": 8912345, // 随机数字，五分钟数字相同认为是重复消息
    "MsgBody": [ // 消息体，由一个 element 数组组成，详见字段说明
        {
            "MsgType": "TIMTextElem", // 文本
            "MsgContent": {
                "Text": "red packet"
            }
        },
        {
            "MsgType": "TIMFaceElem", // 表情
            "MsgContent": {
            "Index": 6,
            "Data": "abc\u0000\u0001"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data",
    "SupportMessageExtension": 0,
}
```

其中，可以修改 `CloudCustomData` 定义消息自定义数据（云端保存），并将自定义消息填入 `MsgBody` （消息体）发送。

组内语音/视频聊天

游戏内的语音/视频聊天也是一项很重要的功能。腾讯云 IM 提供即时通信 IM 和 实时音视频 TRTC 音视频通话功能（TUICallKit），为 IM 聊天提供实时的音频通话或视频通话。

> **说明**

- 为了更好的体验音视频通话功能，我们为每个 SDKAppID 提供了音视频通话能力7天体验版。您可以在 IM 控制台 中为您的应用领取体验版进行体验使用，每个 SDKAppID 有且仅有一次领取体验版的机会。
- 更多有关集成音视频通话能力和价格，详情请见 音视频通话 。更多关于 TUICallKit 的内容，请参见 含 UI 集成方法–音视频通话 。

游戏聊天室类型

游戏聊天室类型一般分为如下几种：

类型	特点
频道	聊天人数非常多，无固定成员列表，无需申请加入群聊，可随时加入、退出。无需发送离线消息推送。
大厅	聊天人数较多，可随时加入、退出。支持历史消息查询。
组队	聊天人数较少，聊天对象可能为陌生人，游戏结束时聊天室销毁。无需发送离线消息推送。
好友	为 C2C 聊天，聊天记录保存，聊天对象存在于好友列表。
私信	为 C2C 聊天，聊天对象可能为陌生人。
直播群	聊天人数无上限，可随时加入、退出。

IM 提供的聊天室类型分为以下几种：

类型	特点
好友工作群（work）	类似普通微信群，创建后仅支持已在群内的好友申请加群，无需被邀请方同意或群组审批。
陌生人社交群（Public）	类似 QQ 群，创建后群主可以指定群主管理员，用户搜索群ID发起群申请后，需要群主或管理员审批通过才能入群。
临时会议群（Meeting）	创建后可以随意进出，且支持查看入群前消息；适合用于音视频会议场景、在线教育场景等与实时音视频产品结合的场景，同旧版本中的 ChatRoom。
直播群（AVChatRoom）	创建后可以随意进出，没有群成员数量上限，但不支持历史消息存储；适合与直播产品结合，用于弹幕聊天场景。
社群（Community）	创建后可以随意进出，最多支持10w人，支持历史消息存储，用户搜索群 ID 发起加群申请后，无需管理员审批即可进群。

根据 IM 的群特性，这里提供示例解决方案，请按照需求应用到您的游戏中：

类型	解决方案	特点
频道、大厅	直播群（AVChatRoom）	聊天人数较多，创建后可随意进出，无需审批
好友	单聊+权限控制（只允许给好友发送消息）	支持仅好友发送消息
私信	单聊+权限控制（App 内任意两个用户之间发送单聊消息）	支持任意两个陌生人发送消息
组队	陌生人社交群（Public）、临时会议群（Meeting）	只有游戏内组队人员能够进入聊天群，支持音视频聊天
直播群	直播群（AVChatRoom）	没有群成员上线，可随时进出群

说明

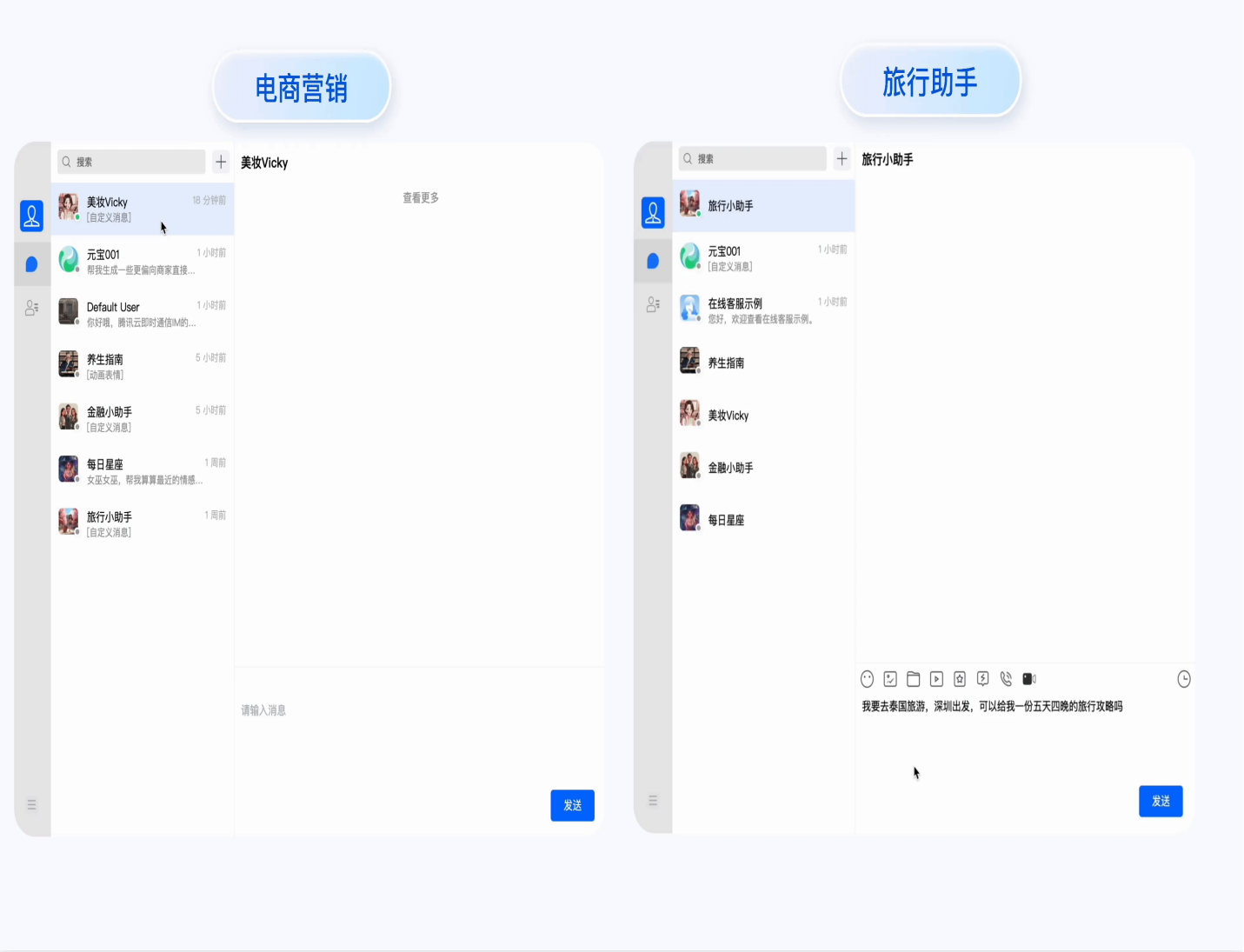
- 关于好友聊天和私信聊天的单聊权限控制请参见 单聊消息权限控制 。
- 更多群组系统相关内容请参见 群组系统 。

AI 聊天机器人方案

最近更新时间：2025-06-18 18:09:32

随着 AI 已成为当下开发者最为关注的焦点，AI和即时通信的结合也得到了进一步的发展和应用。腾讯云即时通信 IM 作为全球领先的通信云服务商，快速发布了相关 AI 能力调用接口，开发者基于腾讯云 IM 提供的通信底座，可以自由调用业内领先的大模型能力，用丰富的 AI 能力赋能自己，实现智能客服、创意辅助、工作助手等业务功能。同时，腾讯云即时通信 IM 还配备有完善的 AI 聊天场景 UI 组件，即开即用，基于 UI 组件您可以像搭积木一样快速搭建起自己的业务逻辑。

应用场景



两种方案

腾讯云即时通信 IM 提供两种接入 AI 能力方案供您参见。

- 方案一通过机器人第三方消息回调将 AI 服务能力引入到 IM 应用中。

- 方案二通过腾讯云提供的智能客服接入 AI 能力，通过智能机器人可以轻松实现 FAQ 智能问答、闲聊、数据运营看板等AI能力，结合腾讯云即时通信 IM 产品，您可以快速实现多平台跨终端互通的聊天机器人。

下文中重点阐述方案一。方案二请您跳转至 智能客服 进行接入。

后台接入流程

方案一整体接入流程如下：

下文中的实践步骤以接入 MiniMax 中文大语言模型为例，类 ChatGPT 服务均可通过文中介绍的方法实现接入。

1. 创建即时通信 IM 应用

登录 即时通信 IM 控制台，创建应用，并获取应用的 `SDKAppID` 和密钥（以下称为 IM Key）。

2. 注册 AI 服务商账号

注册并登录计划接入的 AI 服务商，并获取 API 密钥（以下称为 `AI_SECRET_KEY` ）。

3. 创建机器人账号

通过 REST API 创建一个机器人账号。机器人账号是一种特殊的用户账号，它也通过 UserID 来唯一标识（但是其 UserID 必须以 `@RBT#` 开头），具有昵称、头像、签名等基本属性。假设您成功创建了一个昵称为 MyRobot 的机器人@RBT#001。

4. 配置第三方回调

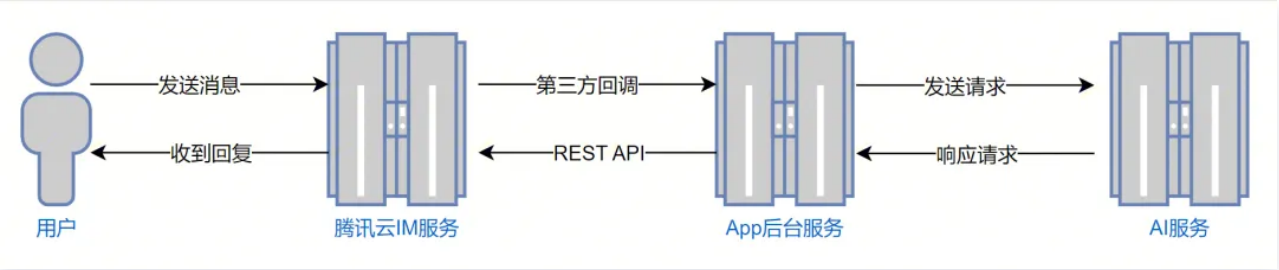
即时通信 IM 第三方回调即云 IM 后台会在某一事件发生之前或者之后，向 App 的后台服务器发送请求，App 后台可以据此进行必要的数据同步，或者干预事件的后续处理流程。我们将使用“机器人事件回调”监听用户发消息给机器人，或者在群聊中@机器人的事件，并对其做出反应。在腾讯云 IM 控制台中找到“机器人事件回调”，点击开启并保存。

5. 开发 APP 后台服务

以单聊为例，总体上的工作流程如下：

1. 用户 user1 发消息 "hello" 给机器人 @RBT#001。
2. 云 IM 后台通过第三方回调将事件通知到 App 后台，通知内容包含发送方 user1，接收方 @RBT#001，消息内容 "hello" 以及其他信息。

3. App 后台调用 AI 服务接口（本文示例 MiniMax API），并得到响应回复内容，如 "nice to meet you"；
4. App 后台调用 REST API 接口（单聊为 sendmsg 接口，群聊为 send_group_msg 接口），将回复内容以 @RBT#001 的身份发送给 user1。



下面我们使用 Golang ，向您展示 App 后台的关键代码。

> **说明：**
> 下文代码仅作示例，省略了大量异常处理代码，请勿直接用于生产环境。

分发处理回调命令

我们创建一个监听在80端口的 HTTP 服务，注册一个 url 为 /im 的处理函数 handler，所有发送给 `http:///im` 的请求都会被 handler 处理。所有云 IM 发送的回调请求都带有 `CallbackCommand` 参数，不同的值代表不同的回调命令。在 handler 中，根据云 IM 设置的参数 `CallbackCommand` 进行对应的处理。

```
func handler(w http.ResponseWriter, r *http.Request) {
    command := r.URL.Query().Get("CallbackCommand")
    reqbody, _ := io.ReadAll(r.Body)
    var rspbody []byte
    switch command {
    case "Bot.OnC2CMessage": // 机器人C2C回调命令字
        dealC2c(context.Background(), reqbody)
        rspbody = []byte("{\"ActionStatus\": \"OK\", \"ErrorCode\": 0,
\"ErrorInfo\": \"\"}")
    default:
        rspbody = []byte("invalid CallbackCommand.")
    }
    w.Write(rspbody)
}

func main() {  // 注册一个handler，处理发送给App后台的回调命令
     http.HandleFunc("/im", handler)
     http.ListenAndServe(":80", nil)
}
```

处理机器人接收到单聊消息事件

处理单聊消息时，我们先检查发送方是不是机器人（一般不会出现这种机器人发送消息给机器人的情况），以防止无限的回调循环。接着，我们解析消息体，拿到用户发送给机器人的消息内容 text，将发送方 UserID 保存到 context 中以方便后续调用 REST API 回复，最后调用 askAI 请求 AI 服务。

```
func dealC2c(ctx context.Context, reqbody []byte) error {
  root, _ := simplejson.NewJson(reqbody)
  jFromAccount := root.Get("From_Account")
  fromAccount, _ = jFromAccount.String()
  // 检查发送方ID，不处理机器人发送给机器人的请求，防止无限循环
  if strings.HasPrefix(fromAccount, "@RBT#") {
    return nil
  }
  jToAccount := root.Get("To_Account")
  toAccount, _ := jToAccount.String()
  msgBodyList, _ := root.Get("MsgBody").Array()
  for _, m := range msgBodyList {
    msgBody, _ := m.(map[string]interface{})
    msgType, _ := msgBody["MsgType"].(string)
    if msgType != "TIMTextElem" {
      continue
    }
    msgContent, _ := msgBody["MsgContent"].(map[string]interface{})
    text, _ := msgContent["Text"].(string)
    ctx = context.WithValue(ctx, "from", fromAccount)
    ctx = context.WithValue(ctx, "to", toAccount)
    go askAI(ctx, text)
  }
  return nil
}
```

调用 AI 服务接口

在这一步我们使用第三方 AI 服务公司 DeepSeek 实现智能聊天的功能，您可以将 DeepSeek 服务替换成任意的其它 AI 服务。需要注意的是这里演示的是简单的 completion 接口，没有保存对话的上下文，其他接口可按需查阅 MiniMax 文档。

```
type MiniMaxRsp struct {
  Reply string `json:"reply"`
}

// 请求MiniMax并得到回复
```

```
func askAI(ctx context.Context, prompt string) {
  url := "https://api.minimax.chat/v1/text/completion"
  var reqData = []byte(`{
    "model": "abab5-completion",
    "prompt": prompt
  }`)
  request, _ := http.NewRequest("POST", url, bytes.NewBuffer(reqData))
  request.Header.Set("Content-Type", "application/json; charset=UTF-8)
  request.Header.Set("Authorization", API_SECRET_KEY)
  client := &http.Client{}
  response, _ := client.Do(request)
  defer response.Body.Close()
  body, _ := ioutil.ReadAll(response.Body)
  rsp := &MiniMaxRsp{}
  json.Unmarshal(body, rsp)
  reply(ctx, rsp.Reply) // 将AI回复的内容发送给用户
}
```

将 AI 返回的结果返回给用户

从 AI 服务得到回复之后，我们只需调用云 IM 的 REST API 接口 sendmsg，指定消息发送方为 @RBT#001，接收方为 user1，模拟机器人回复用户。

```
// 发送一个REST API请求
func doRestAPI(host string, sdkappid int, admin, usersig, command, body
string) {
  url := fmt.Sprintf("https://%s/v4/%s?
sdkappid=%d&identifier=%s&usersig=%s&random=%d&contenttype=json",
    host, command, sdkappid, admin, usersig, rand.Uint32())
  req, _ := http.NewRequest("POST", url, bytes.NewBufferString(body))
  req.Header.Set("Content-Type", "application/json")
  cli := &http.Client{}
  rsp, err := cli.Do(req)
  if err != nil {
    log.Printf("REST API failed. %s", err.Error())
    return
  }
  defer rsp.Body.Close()
  rsptext, _ := io.ReadAll(rsp.Body)
  log.Printf("rsp:%s", rsptext)
}

// 调用腾讯云IM的REST API，回复用户
```

```
func reply(ctx context.Context, text string) {
  rsp := make(map[string]interface{})
  msgbody := []map[string]interface{}{{
    "MsgType":    "TIMTextElem",
    "MsgContent": map[string]interface{}{"Text": text},
  }}
  // GenUserSig 的实现可以参见腾讯云文档
  usersig, _ := GenUserSig(IM_SDKAPPID, IM_KEY, "administrator", 60)
  rsp["From_Account"] = ctx.Value("to").(string) //"@RBT#001"
  rsp["To_Account"] = ctx.Value("from").(string)
  rsp["SyncOtherMachine"] = 2
  rsp["MsgLifeTime"] = 60 * 60 * 24 * 7
  rsp["MsgSeq"] = rand.Uint32()
  rsp["MsgRandom"] = rand.Uint32()
  rsp["MsgBody"] = msgbody
  rspbody, _ := json.Marshal(rsp)
  doRestAPI("console.tim.qq.com", IM_SDKAPPID, "administrator", usersig,
"openim/sendmsg", string(rspbody))
}
```

说明：
以上教程仅实现接入三方 API 发送文本消息，如需发送流式响应消息，请参见 **发送流式响应消息 REST API**。

客户端接入流程

1. 请您参见 Web & H5 集成指引 集成 TUIKit 到您的项目。

说明：
v2.3.6 起支持机器人回复消息的流式效果展示，markdown 解析。

2. 流式响应消息规则

- 流式响应消息均为自定义消息。
- 当客户端接收到的机器人自定义消息的 `payload` 中的 `data` 字段符合以下规则时，自动展示 markdown 格式的流式输出效果。
- 本文介绍的自定义消息的 `payload` 中的 `data` 字段为 JSON 结构体经过序列化后的值，详情请参见 CustomPayload 。
- TUIKit 内部会通过监听 MESSAGE_MODIFIED 事件监听消息内容的变更，此处您无需额外处理。

- TUIKit 机器人消息解析在客服插件之后，请务必保证该机器人账号没有被配置为客服账号，否则将无法进入机器人消息解析逻辑。

```
// 以下为自定义消息 payload 字段中的 data 字段格式规则
{
    // 机器人自定义消息标识字段，此字段为 1 时，表示此自定义消息为机器人自定义消息
    "chatbotPlugin": 1,
    // chunks 代表流式消息的内容
    "chunks": [
        "",
        "您",
        "可以在",
        "管理",
        "端",
        "左侧",
        "菜单",
     ],
     // isFinished 为 0 时表示流式消息未输出结束，1 代表流式输出结束
     "isFinished": 1,
}
```

常见问题

如何实现机器人欢迎语？（打开机器人聊天对话，机器人自动发送欢迎消息）

客户端可以在用户打开机器人聊天对话框时，客户端发送一条静默的自定义消息，设置 `onlineUserOnly` 为 `true`（设置该 flag 表示消息不存漫游），App 后台通过回调接收到此自定义消息后，调用 REST API 发送欢迎消息。

如何发送流式响应消息？

请参见 发送流式响应消息 REST API 。如您需和 Web TUIKit 搭配使用，请务必保证您的 WebTUIKit 版本是 v2.3.6 或更高，且机器人发送的自定义消息符合 流式响应规则。

类微信公众号搭建方案

最近更新时间：2025-06-11 17:12:41

概述

公众号功能可以提供类似微信的公众号的能力，为客户提供订阅和发布等功能。通过公众号相关接口，用户可以创建和管理公众号，发布文章、图文消息等内容，与粉丝进行互动，粉丝也可对公众号发单聊消息。同时 App 后台可以基于 RestAPI + 第三方回调方式对后台运营相关数据进行统计和管理。

场景用途：

公众号是商家申请的账号，可以通过文字、图片、视频等多种形式向用户推送信息。

- 信息传播，能快速将各类资讯如新闻、知识等传递给大量用户。
- 品牌推广，商家可借此提升品牌知名度与影响力。
- 提供便捷服务，例如一些银行公众号可办理转账等业务。

09:36
公众号
搜索
T
腾讯云 TVP
腾讯云开发者
腾讯混元
腾讯云
4个公众号
09:36
腾讯云开发者
09:36
欢迎关注腾讯云开发者社区
这里是腾讯官方技术分享阵地
星标我，不再错过鹅厂新技术～
近期热门文章：十年前的微信消息收发架构长啥样?
点击腾讯技术人原创集、腾讯架构启示录、鹅厂人真实 AIGC 体验集、前端、后端 看技术文集
回复“代金券”，领 5 元腾讯云产品代金券。
加入最有料有趣的程序员交流圈，享专属福利～
技术精华
技术活动
官方福利

通过类微信公众号的功能实践方案，帮助企业通过标准化接入，快速实现“社交增长→私域运营→商业变现”闭环。

集成接入

功能描述

公众号可以为订阅的用户发送广播消息，也可以与订阅的用户进行单聊。
当其中有消息的收发时，会产生 单聊会话，会话 conversationID 的组成为 c2c_officialAccountID。
公众号的创建等管理功能参见 服务端的接口，IMSDK 主要提供订阅公众号、取消订阅公众号、获取公众号列表等功能。

> **说明：**
> 从 7.6.5011 版本开始支持。

公众号资料类介绍

属性	含义	说明

officialAccountID	公众号 ID	以 @TOA#_ 作为前缀，最长48个字节，可自定义公众号 ID 。
officialAccountName	公众号名称	最长150字节，使用 UTF-8 编码，1个汉字占3个字节。
faceUrl	公众号展示头像	最长500字节。
organization	组织名称	最长500字节，使用 UTF-8 编码，1个汉字占3个字节。
introduction	简介	最长400字节，使用 UTF-8 编码，1个汉字占3个字节。
customData	自定义数据	最长3000字节。
createTime	创建时间	单位：秒。
subscriberCount	订阅的人数	公众号目前的订阅者人数。
subscribeTime	登录用户订阅的时间	单位：秒。

订阅公众号

调用 subscribeOfficialAccount（Java / Swift / Objective-C / C++）订阅公众号，直接传入公众号的 officialAccountID 即可。

1. 订阅公众号成功后，订阅者会收到 onOfficialAccountSubscribed（Java / Swift / Objective-C / C++）通知。
2. 当调用 服务端接口 修改**订阅的**公众号资料时，订阅者会收到 onOfficialAccountInfoChanged（Java / Swift / Objective-C / C++）通知。
3. 当调用 服务端接口 销毁**订阅的**公众号资料时，订阅者会收到 onOfficialAccountDeleted（Java / Swift / Objective-C / C++）通知。

示例代码如下：

Java

```
V2TIMManager.getFriendshipManager().subscribeOfficialAccount("official_t
est", new V2TIMCallback() {
    @Override
    public void onSuccess() {
        // 订阅公众号成功
    }
```

```
    @Override
    public void onError(int code, String desc) {
        // 订阅公众号失败
    }
});

V2TIMManager.getFriendshipManager().addFriendListener(new
V2TIMFriendshipListener() {
    @Override
    public void onOfficialAccountSubscribed(V2TIMOfficialAccountInfo
officialAccountInfo) {
        // 订阅公众号通知
    }

    @Override
    public void onOfficialAccountDeleted(String officialAccountID) {
        // 订阅的公众号被销毁通知
    }

    @Override
    public void onOfficialAccountInfoChanged(V2TIMOfficialAccountInfo
officialAccountInfo) {
        // 订阅的公众号资料更新通知
    }
});
```

Swift

```
V2TIMManager.shared.subscribeOfficialAccount(officialAccountID:
"officialAccountID") {
    print("subscribeOfficialAccount succ")
} fail: { code, desc in
    print("subscribeOfficialAccount fail, \(code), \(desc)")
}
V2TIMManager.shared.addFriendListener(listener: self)

func onOfficialAccountSubscribed(officialAccountInfo:
V2TIMOfficialAccountInfo) {
    print("officialAccountInfo:\(officialAccountInfo.description)");
}

func onOfficialAccountDeleted(officialAccountID: String) {
```

```
    print("officialAccountID:\(officialAccountID)");
}

func onOfficialAccountInfoChanged(officialAccountInfo:
V2TIMOfficialAccountInfo) {
    print("officialAccountInfo:\(officialAccountInfo.description)");
}
```

Objective-C

```
[[V2TIMManager sharedInstance] subscribeOfficialAccount:@"official_test"
succ:^ {
    NSLog(@"订阅公众号成功");
} fail:^(int code, NSString *desc) {
    NSLog(@"订阅公众号失败, code: %d, msg: %@", code, msg);
}];

[[V2TIMManager sharedInstance] addFriendListener:self];

- (void)onOfficialAccountSubscribed:(V2TIMOfficialAccountInfo
*)officialAccountInfo {
    // 公众号订阅通知
}

- (void)onOfficialAccountDeleted:(NSString *)officialAccountID {
    // 订阅的公众号被销毁通知
}

- (void)onOfficialAccountInfoChanged:(V2TIMOfficialAccountInfo
*)officialAccountInfo {
    // 订阅的公众号资料更新通知
}
```

C++

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
```

```
    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback
error_callback) {
      success_callback_ = std::move(success_callback);
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message)
override {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

auto callback = new Callback;
callback->SetCallback(
    [=]() {
      // 订阅公众号成功
      delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
      // 订阅公众号失败
      delete callback;
    });
V2TIMManager::GetInstance()->GetFriendshipManager()-
>SubscribeOfficialAccount(
    "official_test", callback);

// 公众号通知
class FriendshipListener final : public V2TIMFriendshipListener {
public:
```

```
    FriendshipListener() = default;
    ~FriendshipListener() override = default;

    void OnOfficialAccountSubscribed(const V2TIMOfficialAccountInfo
&info) override {
        // 公众号订阅通知
    }

    void OnOfficialAccountDeleted(const V2TIMString &officialAccountID)
override {
        // 订阅的公众号被销毁通知
    }

    void OnOfficialAccountInfoChanged(const V2TIMOfficialAccountInfo
&info) override {
        // 订阅的公众号资料更新通知
    }
};

FriendshipListener friendshipListener;
V2TIMManager::GetInstance()->AddFriendshipListener(&friendshipListener);
```

取消订阅公众号

调用 unsubscribeOfficialAccount（Java / Swift / Objective-C / C++）订阅公众号，直接传入公众号的 officialAccountID 即可。
取消订阅公众号成功后，取消订阅的用户会收到 onOfficialAccountUnsubscribed（Java / Swift / Objective-C / C++）通知。
示例代码如下：

Java

```
V2TIMManager.getFriendshipManager().unsubscribeOfficialAccount("official
_test", new V2TIMCallback() {
    @Override
    public void onSuccess() {
        // 取消订阅公众号成功
    }

    @Override
    public void onError(int code, String desc) {
        // 取消订阅公众号失败
```

```
    }
});

V2TIMManager.getFriendshipManager().addFriendListener(new
V2TIMFriendshipListener() {
    @Override
    public void onOfficialAccountUnsubscribed(String officialAccountID)
{
        // 取消订阅公众号通知
    }
});
```

Swift

```
V2TIMManager.shared.unsubscribeOfficialAccount(officialAccountID:
"officialAccountID") {
    print("unsubscribeOfficialAccount succ")
} fail: { code, desc in
    print("unsubscribeOfficialAccount fail, \(code), \(desc)")
}

V2TIMManager.shared.addFriendListener(listener: self)

func onOfficialAccountUnsubscribed(officialAccountID: String) {
    print("officialAccountID:\(officialAccountID)");
}
```

Objective-C

```
[[V2TIMManager sharedInstance]
unsubscribeOfficialAccount:@"official_test" succ:^ {
    NSLog(@"取消订阅公众号成功");
} fail:^(int code, NSString *desc) {
    NSLog(@"取消订阅公众号失败, code: %d, msg: %@", code, msg);
}];

[[V2TIMManager sharedInstance] addFriendListener:self];

- (void)onOfficialAccountUnsubscribed:(V2TIMOfficialAccountInfo
*)officialAccountInfo {
    // 取消订阅公众号通知
```

```
}
```

C++

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback
error_callback) {
      success_callback_ = std::move(success_callback);
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message)
override {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

auto callback = new Callback;
callback->SetCallback(
    [=]() {
      // 取消订阅公众号成功
      delete callback;
    },
```

```
    [=](int error_code, const V2TIMString& error_message) {
      // 取消订阅公众号失败
      delete callback;
    });

V2TIMManager::GetInstance()->GetFriendshipManager()-
>UnsubscribeOfficialAccount(
    "official_test", callback);

// 公众号通知
class FriendshipListener final : public V2TIMFriendshipListener {
public:
    FriendshipListener() = default;
    ~FriendshipListener() override = default;

    void OnOfficialAccountUnsubscribed(const V2TIMString
&officialAccountID) override {
        // 取消订阅公众号通知
    }
};

FriendshipListener friendshipListener;
V2TIMManager::GetInstance()->AddFriendshipListener(&friendshipListener);
```

获取公众号列表

调用 getOfficialAccountsInfo（Java / Swift / Objective-C / C++）接口获取公众号列表。

- 当 officialAccountIDList 列表为空时，为获取订阅的公众号列表；
- 当 officialAccountIDList 填入指定的公众号 ID 时，为获取填入的公众号列表。

示例代码如下：

Java

```
List<String> officialAccountIDList = new ArrayList<>();
V2TIMManager.getFriendshipManager().getOfficialAccountsInfo(officialAcco
untIDList, new V2TIMValueCallback<List<V2TIMOfficialAccountInfoResult>>
() {
    @Override
    public void onSuccess(List<V2TIMOfficialAccountInfoResult>
v2TIMOfficialAccountInfoResults) {
        // 获取订阅的公众号列表成功
    }
```

```
    @Override
    public void onError(int code, String desc) {
        // 获取订阅的公众号列表失败
    }
});
```

Swift

```
V2TIMManager.shared.getOfficialAccountsInfo(officialAccountIDList:
["officialAccountID"]) { officialAccountResultList in
    officialAccountResultList.forEach { item in
        print(item.description)
    }

} fail: { code, desc in
    print("getOfficialAccountsInfo fail, \(code), \(desc)")
}
```

Objective-C

```
[[V2TIMManager sharedInstance] getOfficialAccountsInfo:nil
succ:^(NSArray<V2TIMOfficialAccountInfoResult *> *resultList) {
    [self appendString:[NSString stringWithFormat:@"获取订阅的公众号列表完
成: %@", resultList]];
} fail:^(int code, NSString *desc) {
    [self appendString:[NSString stringWithFormat:@"获取订阅的公众号列表失
败, code: %d msg: %@",code,desc]];
}];
```

C++

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;
```

```
    void SetCallback(SuccessCallback success_callback, ErrorCallback
error_callback) {
      success_callback_ = std::move(success_callback);
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message)
override {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMStringVector officialAccountIDList;

auto callback = new ValueCallback<V2TIMTopicInfoResultVector>{};
callback->SetCallback(
    [=](const V2TIMOfficialAccountInfoResultVector&
officialAccountInfoResultList) {
      // 获取订阅的公众号列表成功
      delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
      // 获取订阅的公众号列表失败
      delete callback;
    });

V2TIMManager::GetInstance()->GetFriendshipManager()-
>GetOfficialAccountsInfo(
    officialAccountIDList, callback);
```

公众号收发消息

公众号的消息分为：公众号给所有订阅者发送的广播消息和公众号与某一个订阅者之间的单聊消息。

公众号发送广播消息

调用 服务端公众号发送广播消息接口 可以向关注公众号的所有订阅者发送广播消息。

公众号与订阅者互发单聊消息

调用 IMSDK 的 sendMessage（Java / Swift / Objective-C / C++）接口发送给公众号消息，其中 receiver 填公众号 officialAccountID。
调用 服务端发送单聊消息接口 向该订阅者发送单聊消息，其中 From_Account 填公众号 officialAccountID，To_Account 填订阅者 userID。

接收消息

两种消息都可以通过 IMSDK 的 消息监听器 来接收。