即时通信 IM

更多实践

产品文档

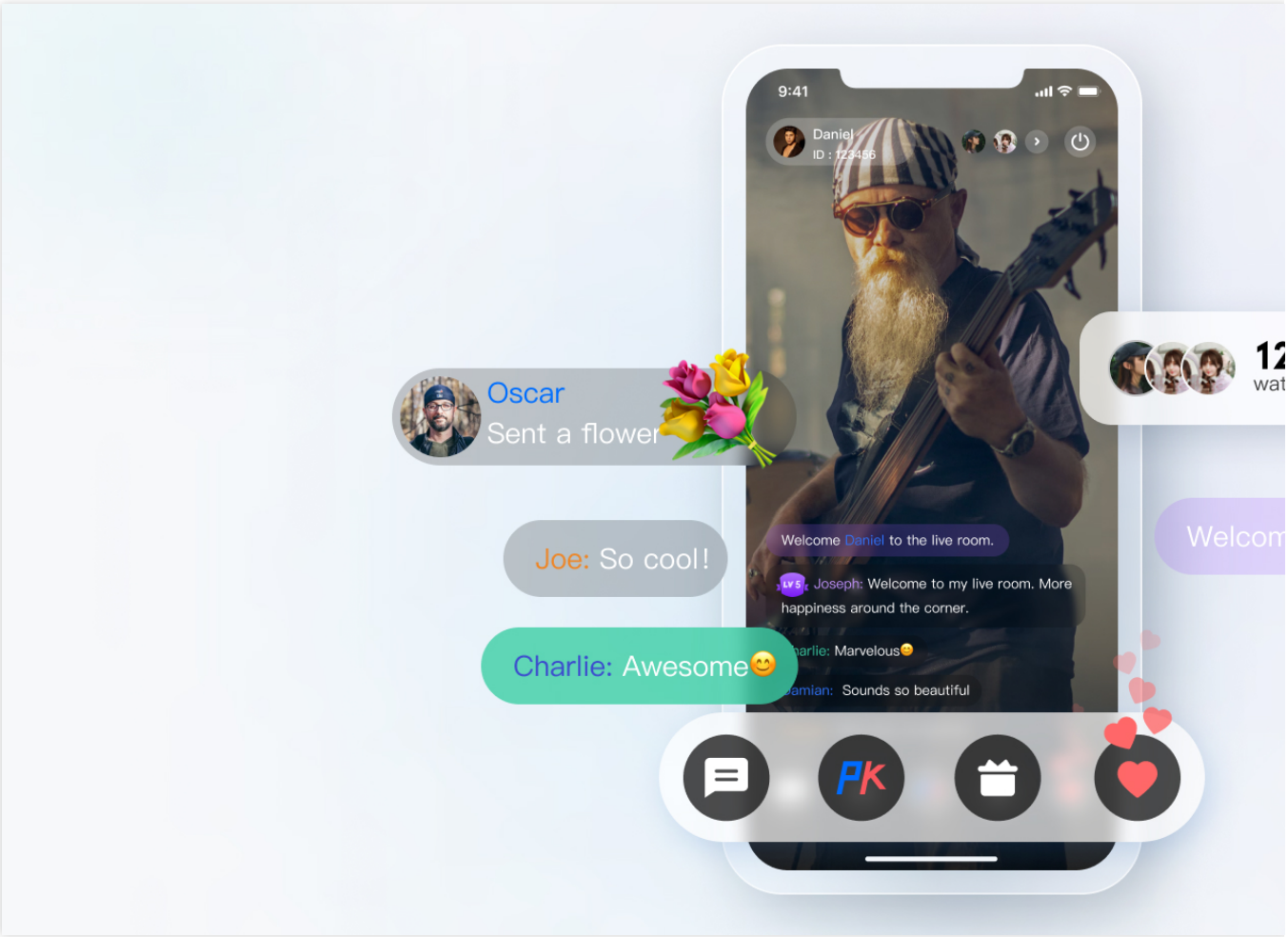
腾讯云

文档目录

更多实践
直播间相关
直播间搭建

最近更新时间：2024-05-13 17:39:45

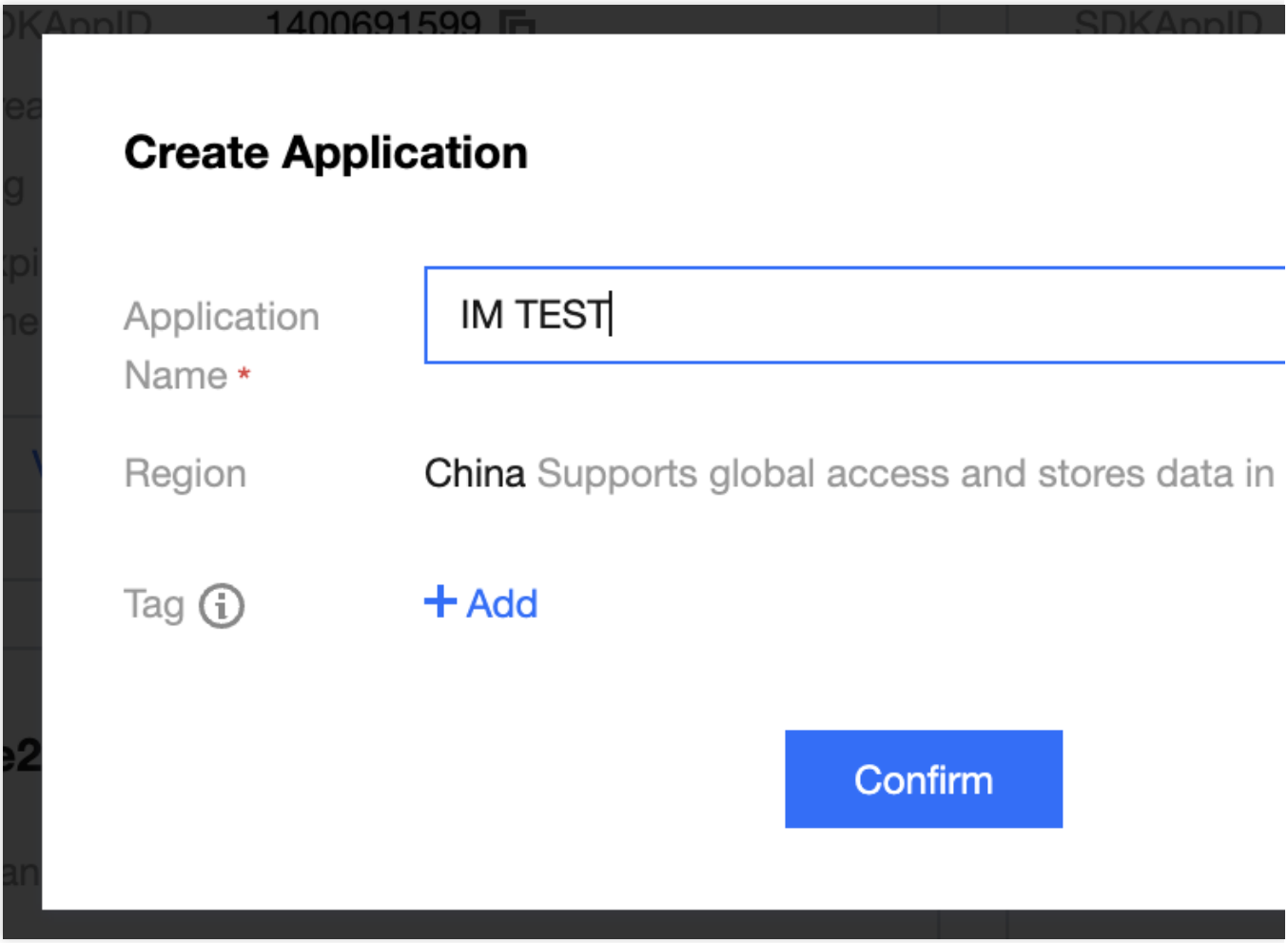
直播在生活中可谓是无处不在，越来越多的企业、开发者们正在搭建自己的直播平台。由于直播平台本身所涉及到的如推拉直播流、直播转码、直播截图、直播混流、直播间聊天室、直播间互动（点赞、送礼、连麦）、直播间状态管理等需求往往比较复杂，本文章以腾讯云相关产品（腾讯云即时通信 IM、腾讯云直播 CSS ）为基础，梳理了在搭建直播间过程中常见的需求的实现方案，以及可能遇到的问题、需要注意的细节点等，希望能帮助开发者们快速的理解业务、实现需求。
您也可以通过我们的 Demo 快速在线体验：



一、准备工作

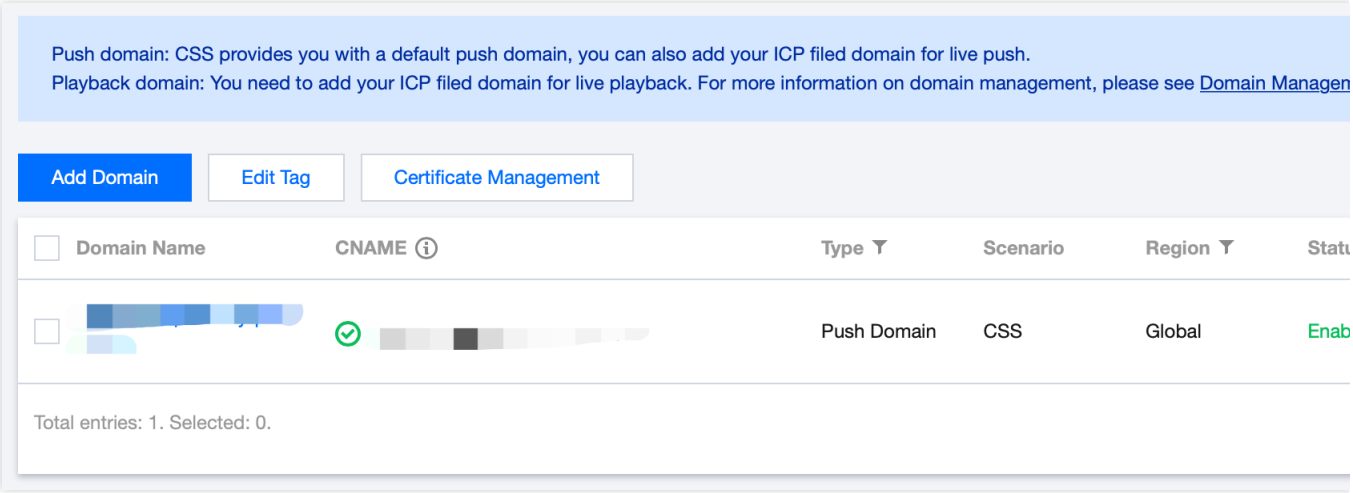
创建应用

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：



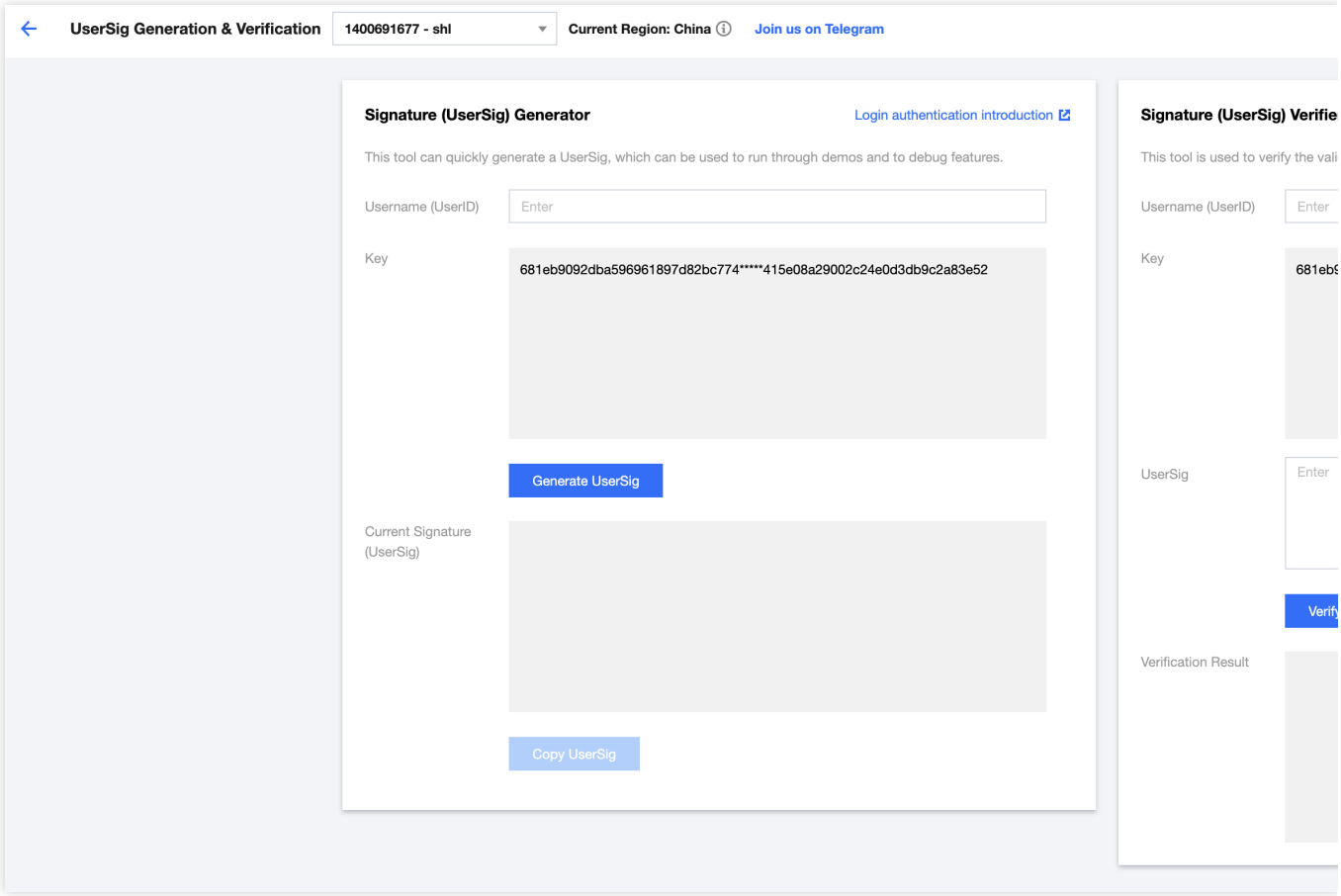
操作详情请参见 添加自有域名。

完成相关基本配置

准备工作步骤一中创建的应用为开发版，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，开通标准版或进阶版。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：

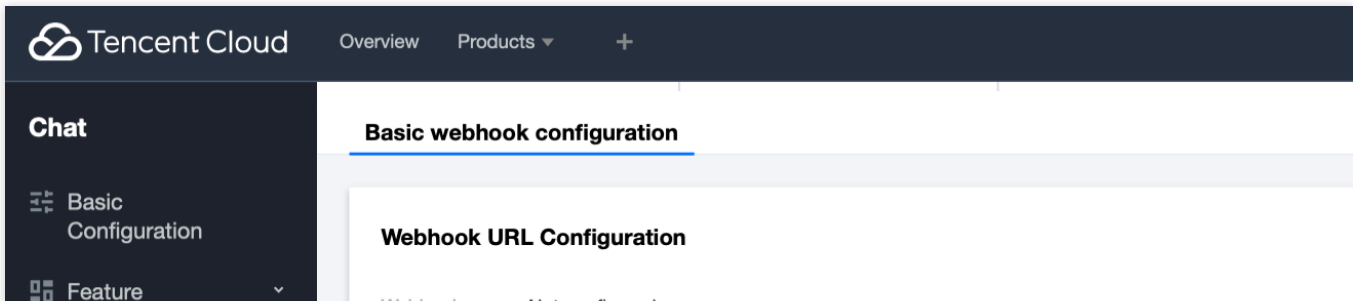


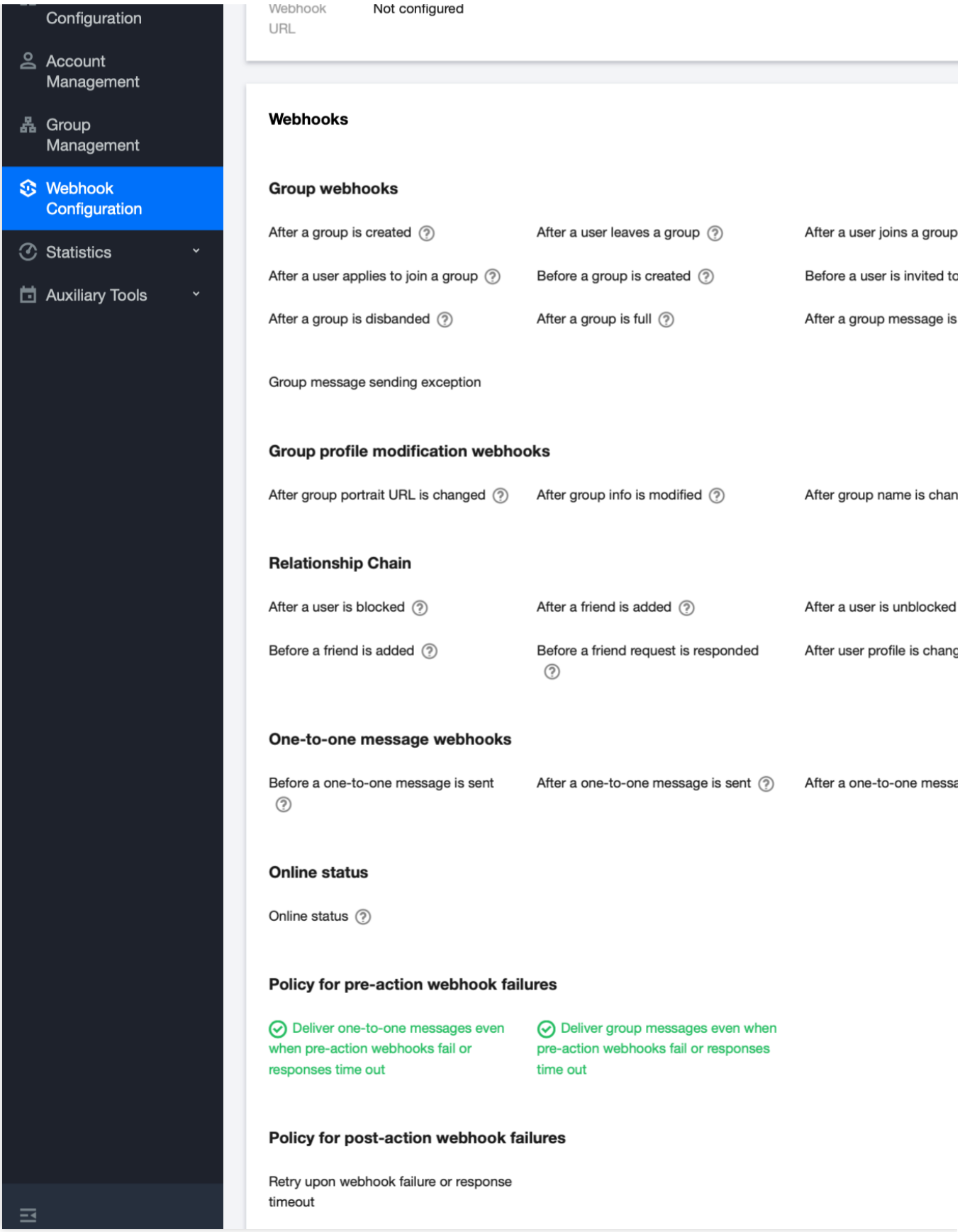
配置管理员账号

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、敏感内容检测等需求时，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：




Configuration
Account Management
Group Management
Webhook Configuration
Statistics
Auxiliary Tools
Webhook URL
Not configured
Webhooks
Group webhooks
After a group is created
After a user leaves a group
After a user joins a group
After a user applies to join a group
Before a group is created
Before a user is invited to
After a group is disbanded
After a group is full
After a group message is
Group message sending exception
Group profile modification webhooks
After group portrait URL is changed
After group info is modified
After group name is chan
Relationship Chain
After a user is blocked
After a friend is added
After a user is unblocked
Before a friend is added
Before a friend request is responded
After user profile is chang
One-to-one message webhooks
Before a one-to-one message is sent
After a one-to-one message is sent
After a one-to-one messa
Online status
Online status
Policy for pre-action webhook failures
Deliver one-to-one messages even when pre-action webhooks fail or responses time out
Deliver group messages even when pre-action webhooks fail or responses time out
Policy for post-action webhook failures
Retry upon webhook failure or response timeout

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，提供最佳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	开发者不需要提供额外的读写直	在高曝光模块需要频繁获取群资料，增大 IM 压力。

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。	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 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。

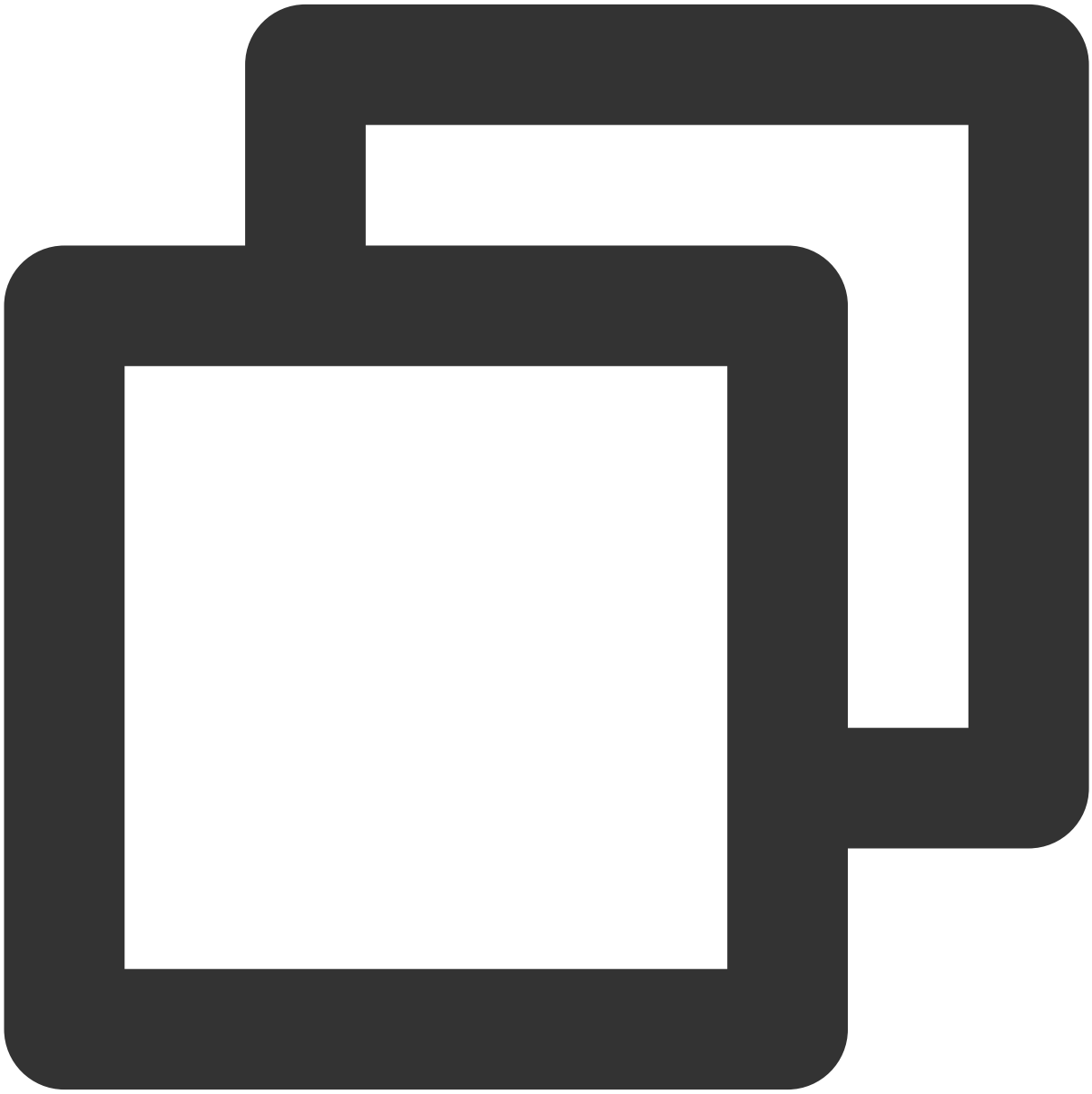
相关代码示例：

Android

iOS & Mac

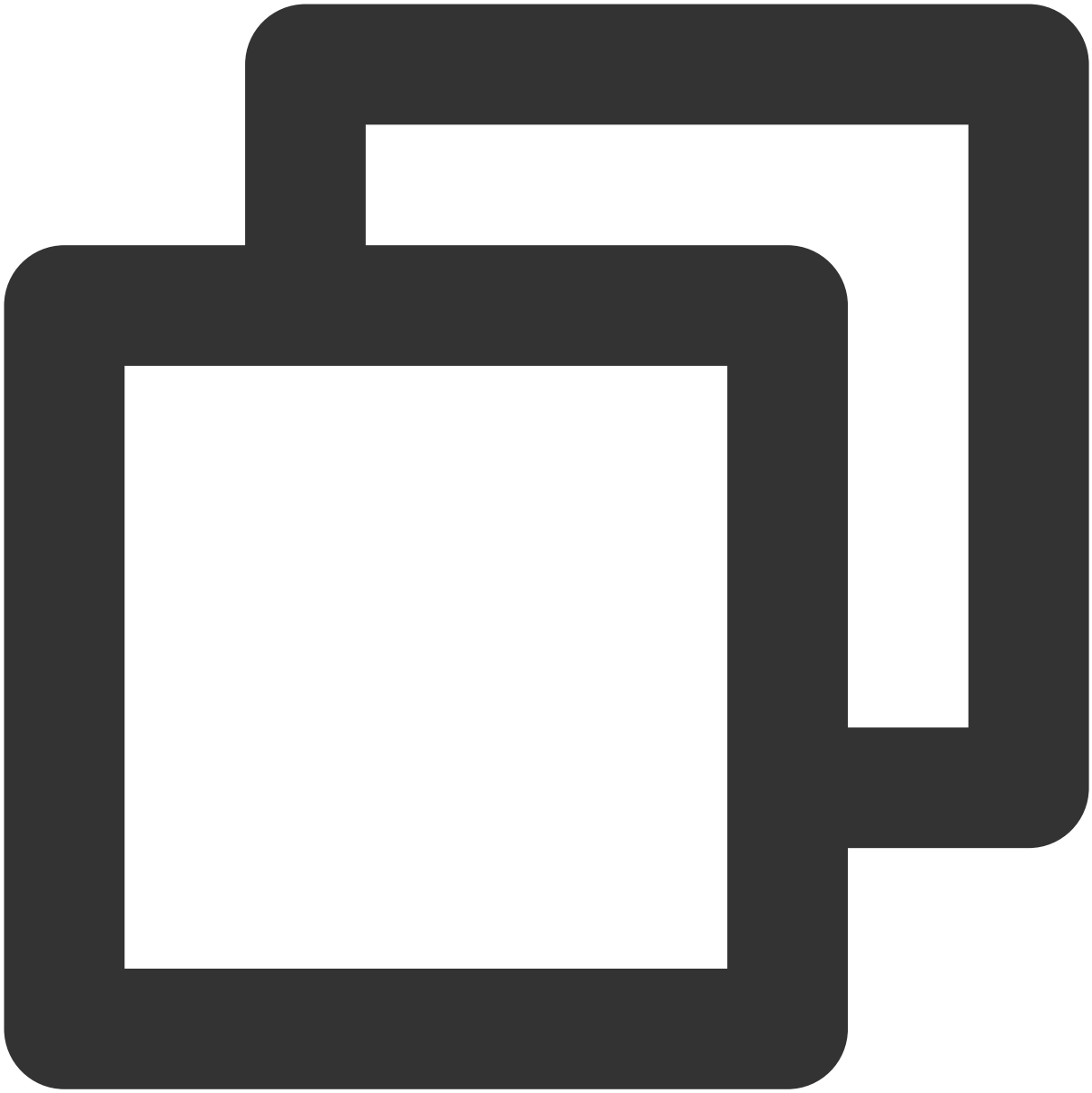
Flutter

Web



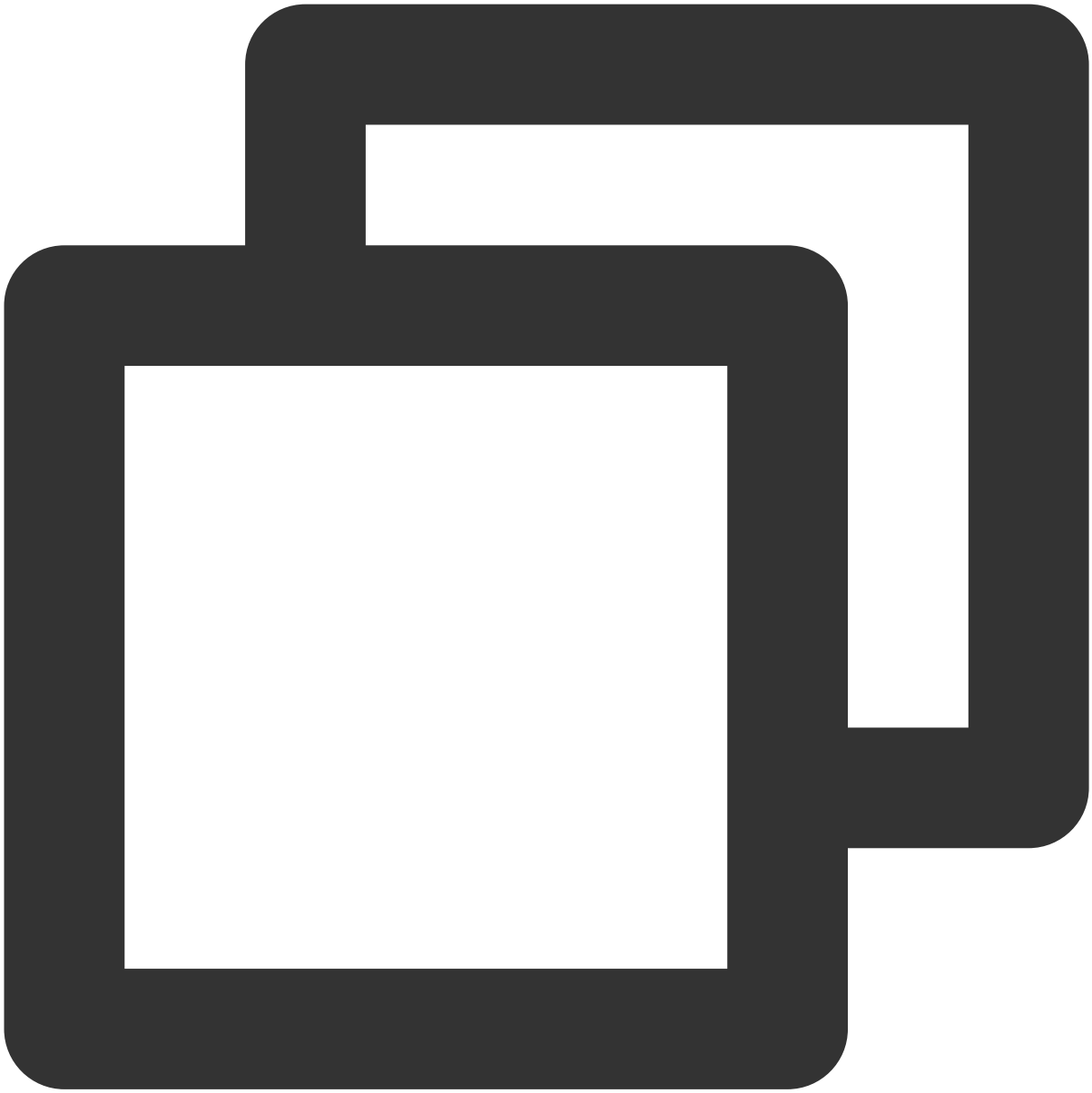
```
// 客户端获取群资料
V2TIMManager.getGroupManager().getGroupsInfo(groupIDList, new V2TIMValueCallback<Li
```

```
    @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
    }
});
```



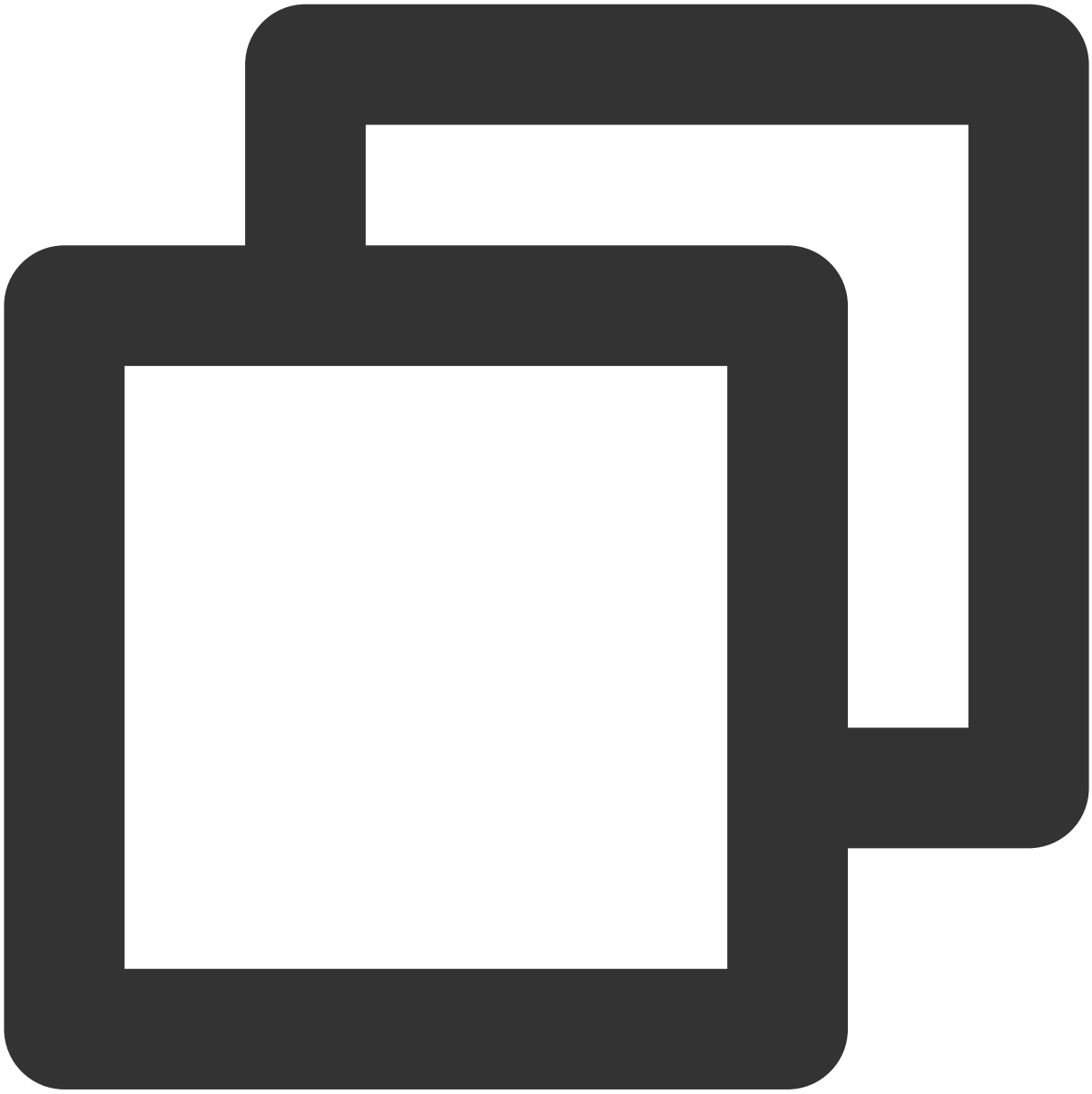
```
[[V2TIMManager sharedInstance] getGroupsInfo:@[@"groupA"] succ:^(NSArray<V2TIMGroup
    // 获取群资料成功
} fail:^(int code, NSString *desc) {
    // 获取群资料失败
}];
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"需要修改的群 ID";
info.faceURL = @"http://xxxx";
[[V2TIMManager sharedInstance] setGroupInfo:info succ:^{
    // 修改群资料成功
} fail:^(int code, NSString *desc) {
```

```
    // 修改群资料失败
}];
```



```
// 获取群资料
V2TimValueCallback<List<V2TimGroupInfoResult>> groupinfos = await groupManager.getG
// 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(onGro
    // 群信息更改回调
  }));
```



```
// 获取群资料
let promise = tim.getGroupProfile({ groupID: 'group1', groupCustomFieldFilter: ['ke
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.pa
  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。

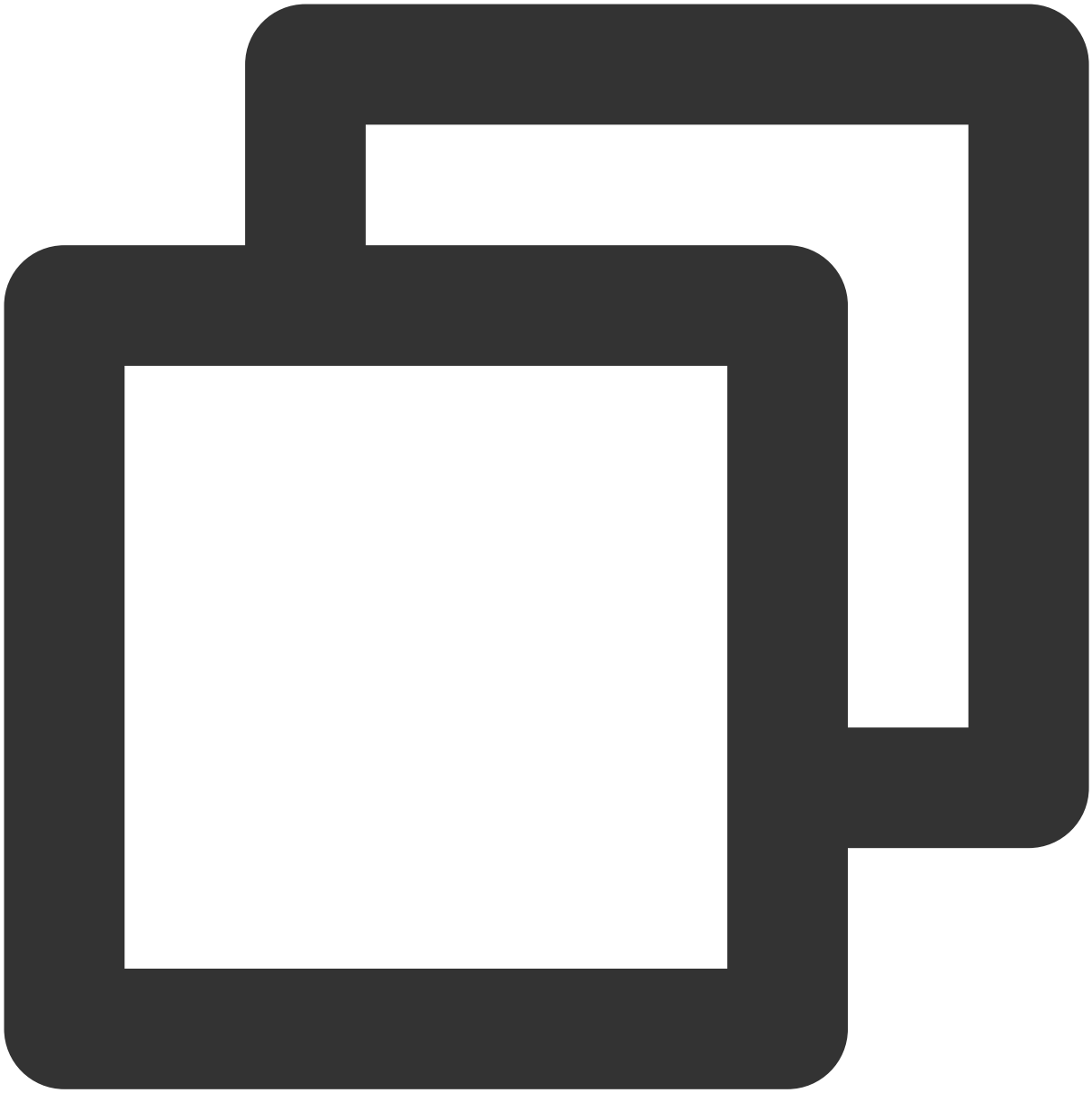
设置消息优先级代码示例：

Android

iOS&Mac

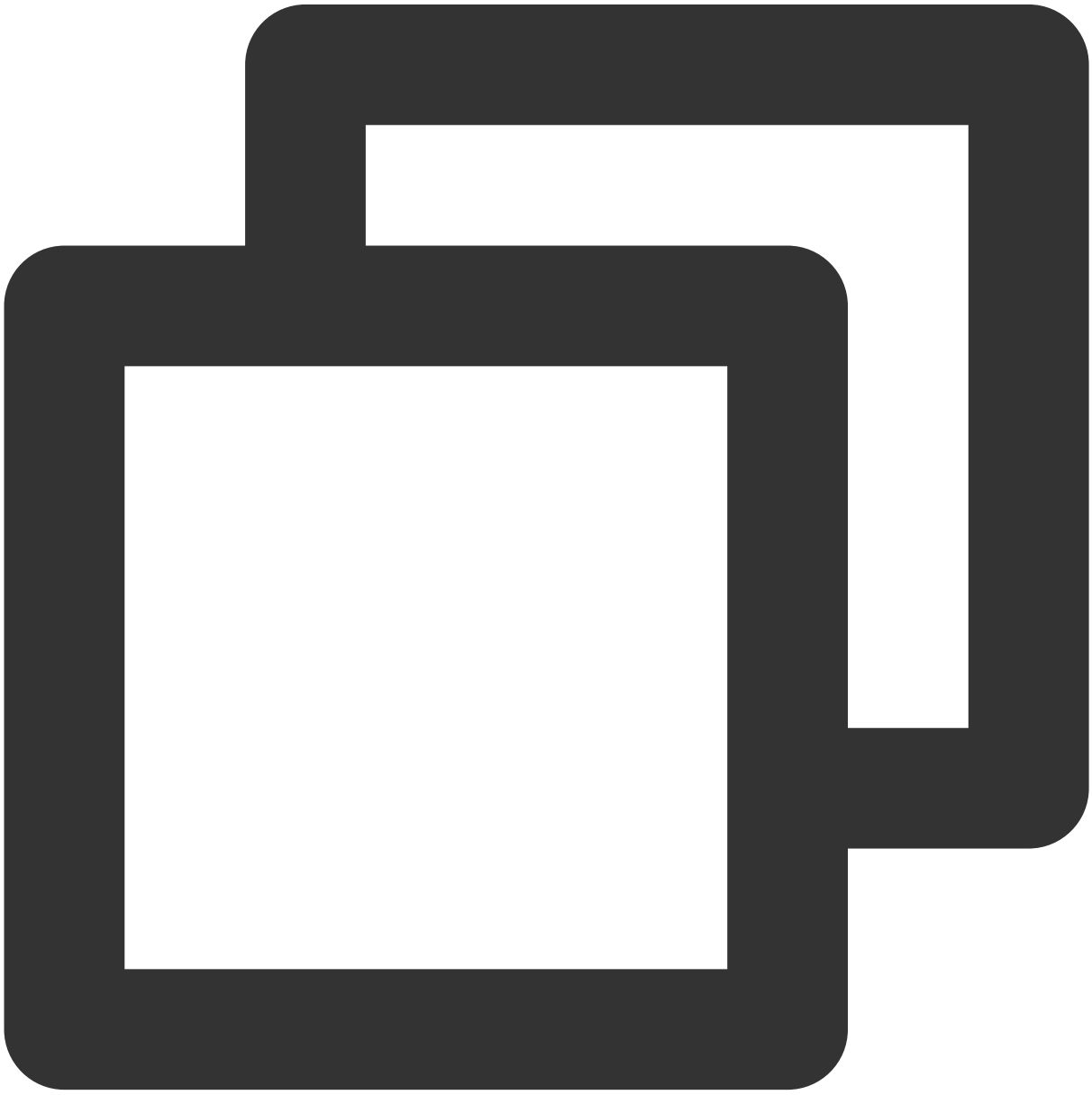
Flutter

Web

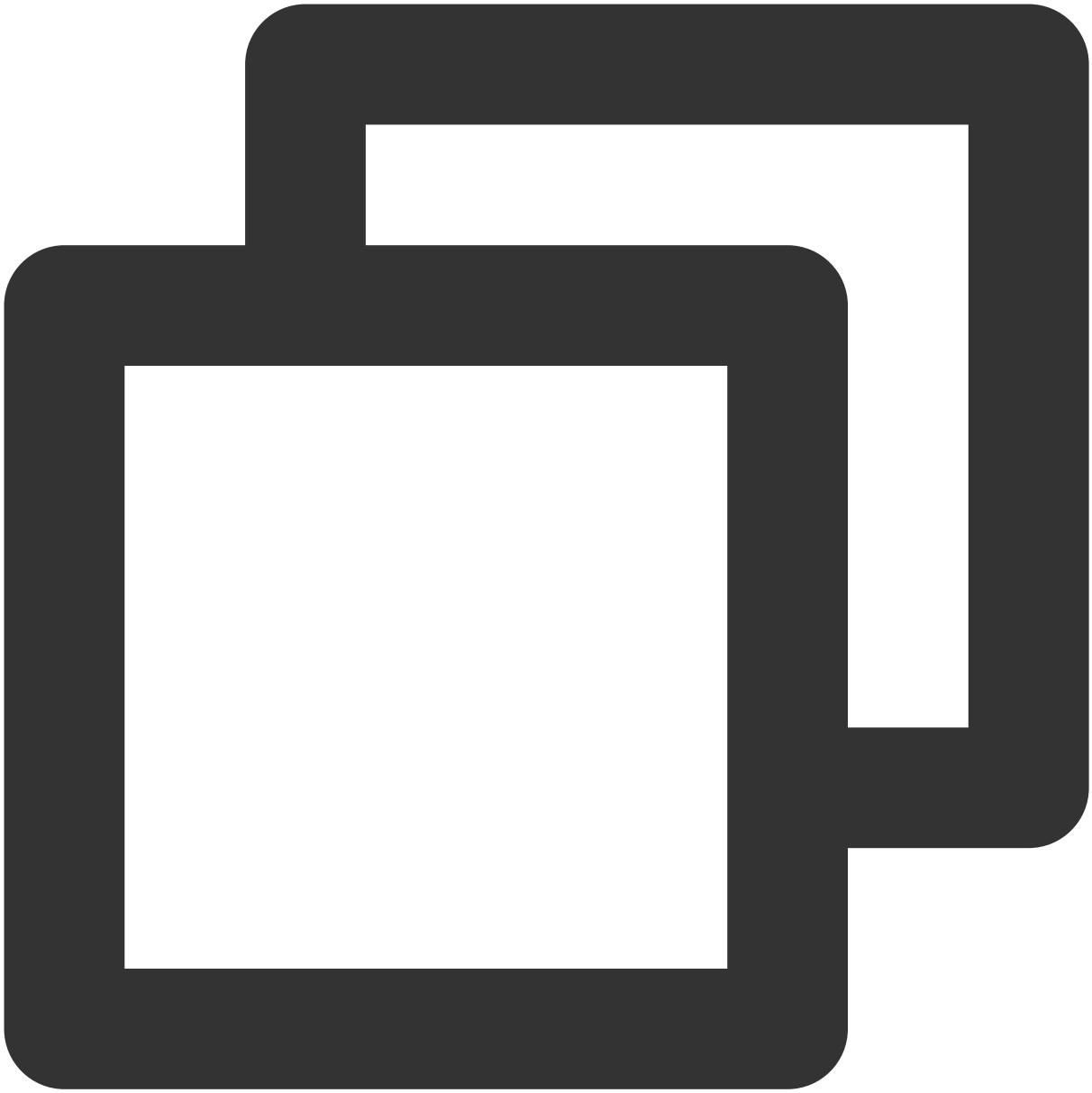


```
// 创建自定义消息
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createCustomMessage("自
// 设置消息优先级为高优先级消息
v2TIMMessage.setPriority(V2TIMMessage.V2TIM_PRIORITY_HIGH)
// 发送消息
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int 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
    }
});
```

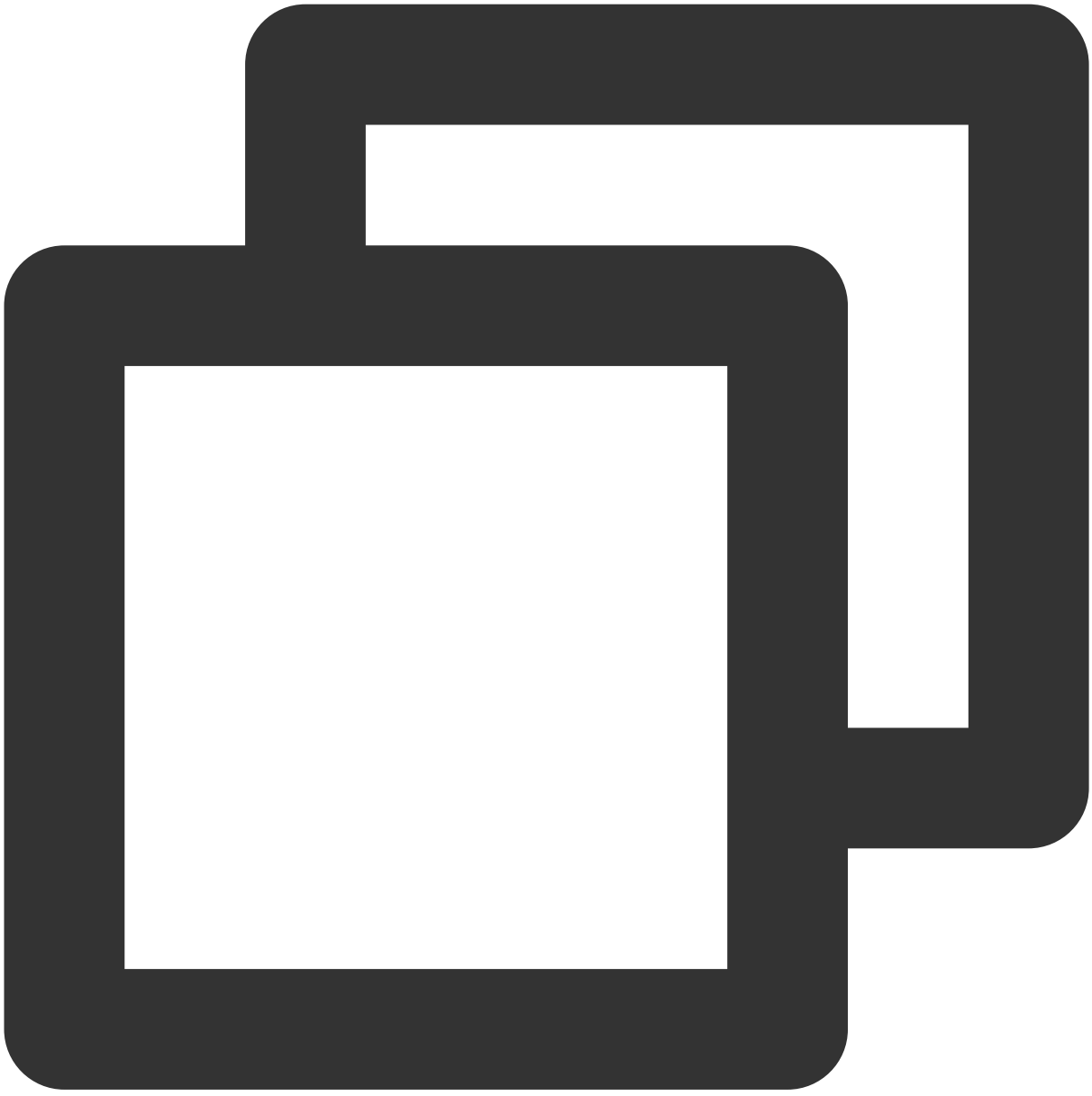


```
/ 创建文本消息
V2TIMMessage *message = [[V2TIMManager sharedInstance] createTextMessage:@"content"
// 发送消息
[V2TIMManager.sharedInstance sendMessage:message
                                receiver:@"userID"
                                 groupID:nil
                                priority:V2TIM_PRIORITY_NORMAL
                          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
}];
```



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

        // 发送文本消息
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin.v2TI
    if(sendMessageRes.code == 0){
```

```
      // 发送成功
    }
  }
```

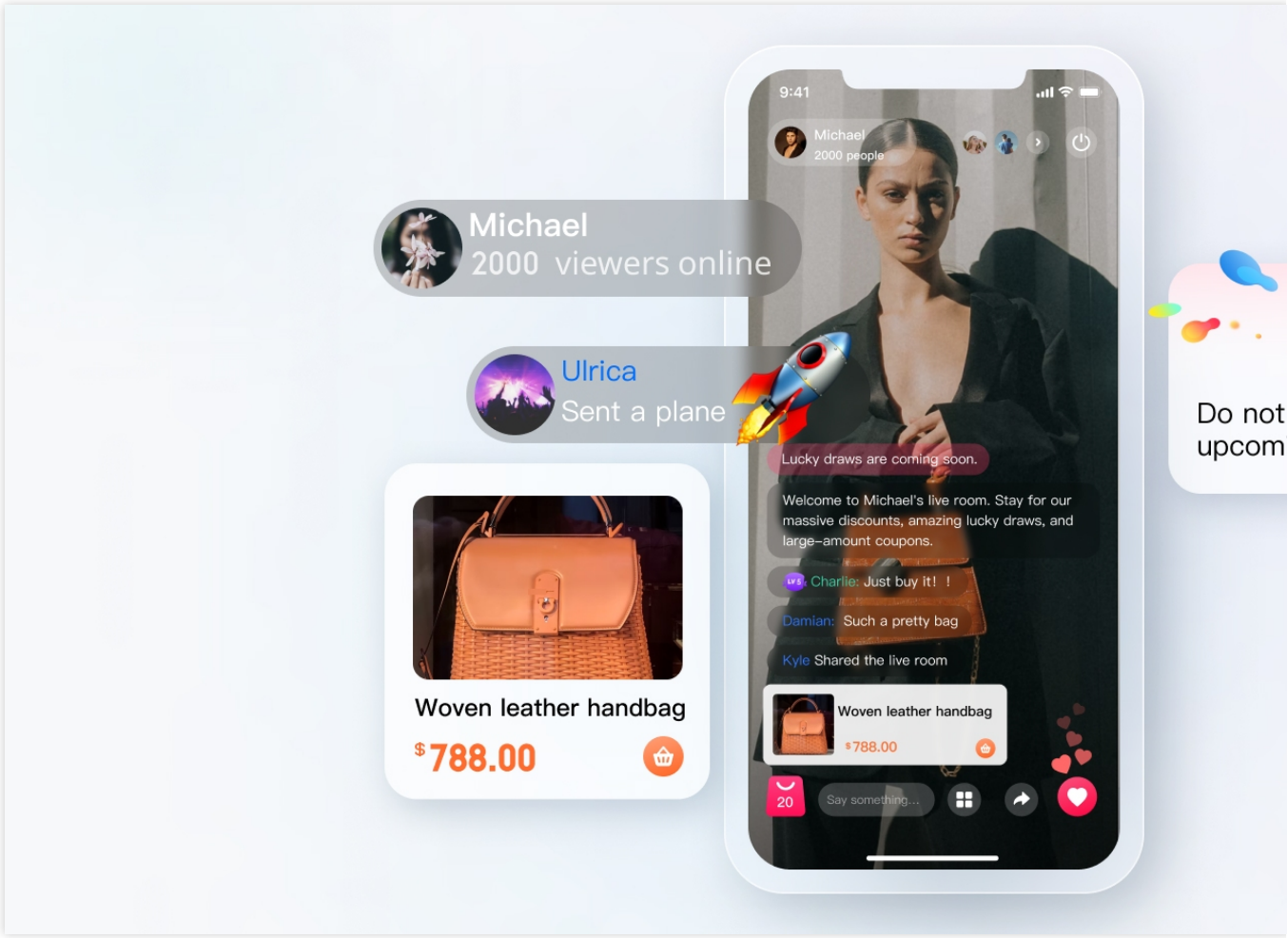


```
// 发送文本消息
// 1. 创建消息实例，接口返回的实例可以上屏
let message = tim.createTextMessage({
  to: 'user1',
  conversationType: TIM.TYPES.CONV_C2C,
  // 消息优先级，用于群聊（v2.4.2起支持）。如果某个群的消息超过了频率限制，后台会优先下发高优先级
  // 支持的枚举值：TIM.TYPES.MSG_PRIORITY_HIGH, TIM.TYPES.MSG_PRIORITY_NORMAL（默认），
```

```
  // 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，需购买进阶版套餐，并且创建消息时
  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

礼物与点赞消息最佳实践



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，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

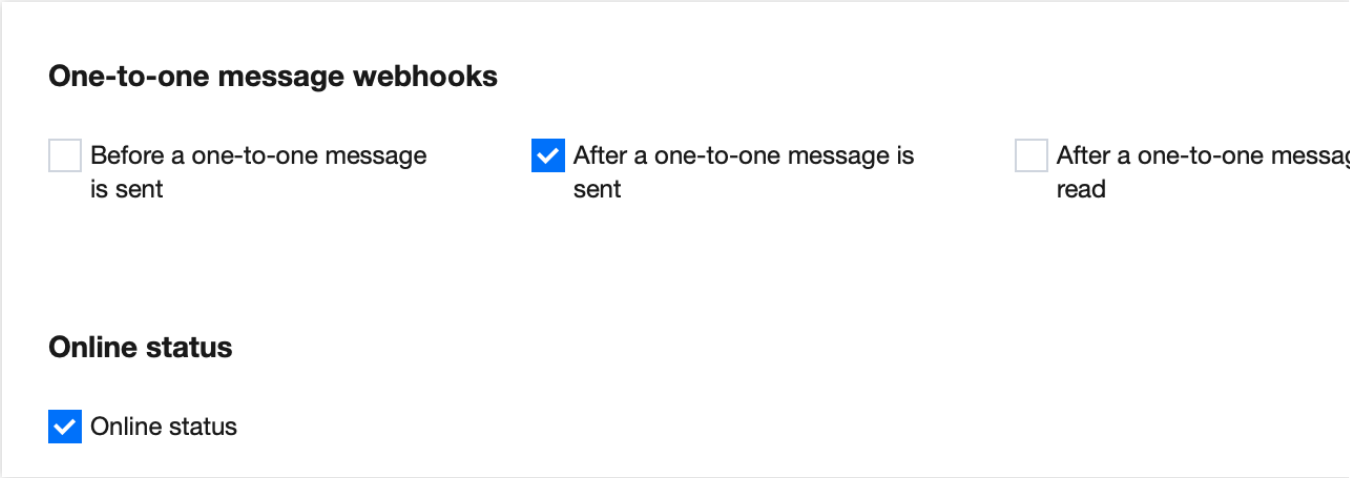
1. 点赞与礼物略不同，点赞往往不用计费，所以一般采用端上直接发送的方式
2. 针对与有需要服务端计数的点赞消息，在客户端节流之后，统计客户端点赞次数，将短时间内多次点赞消息合并之后，仅发送一次消息即可。业务方服务端在发消息前回调中拿到点赞次数进行统计
3. 针对与不需要计数的点赞消息，与步骤2中的逻辑一致，业务要在客户端对点赞消息节流后发送，不需要在发消息前回调中计数。

直播间用户身份、等级

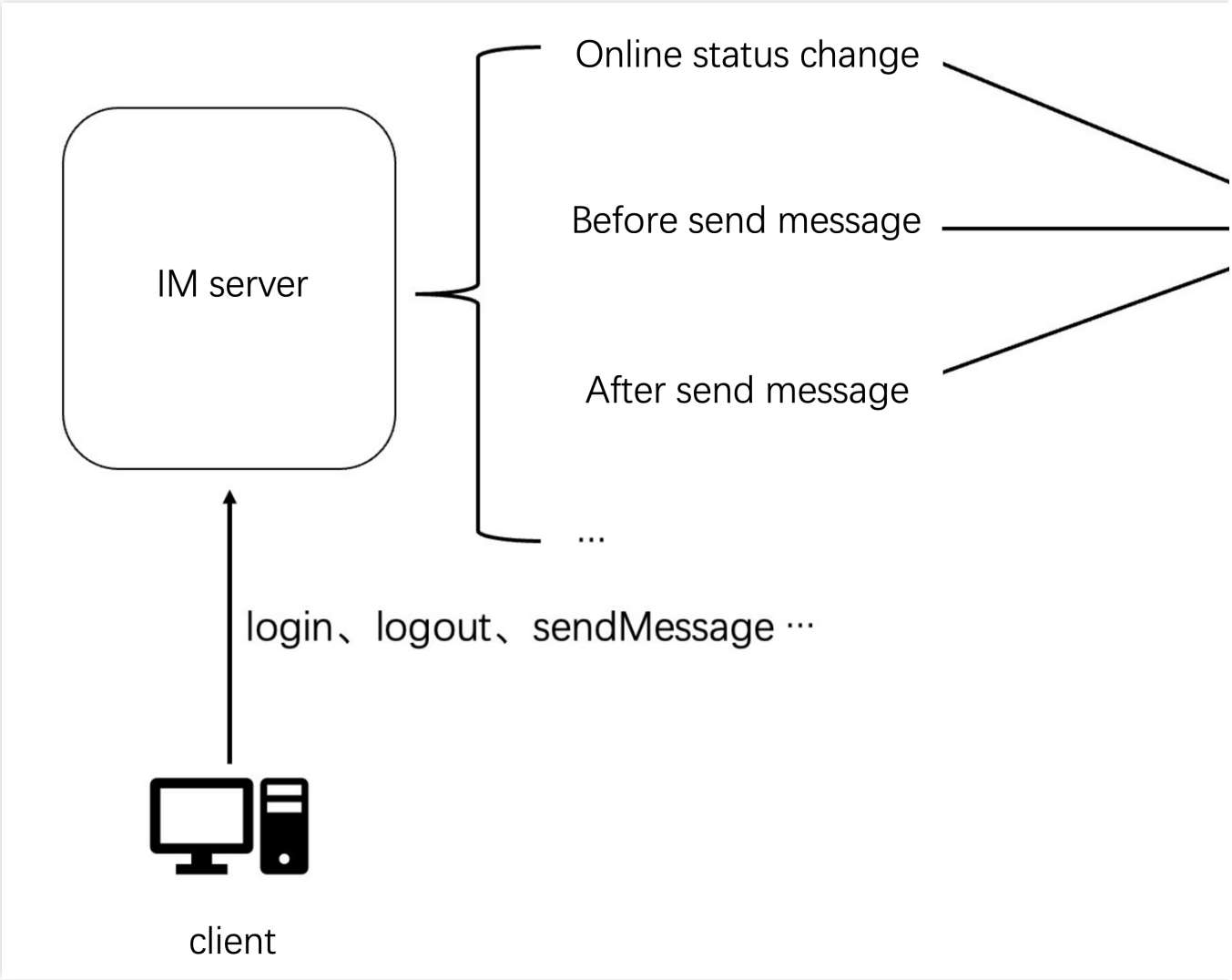
大多直播间都会有用户等级的概念，开发者可能会根据以下几点按照一定的权重来计算用户的等级：

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. ...

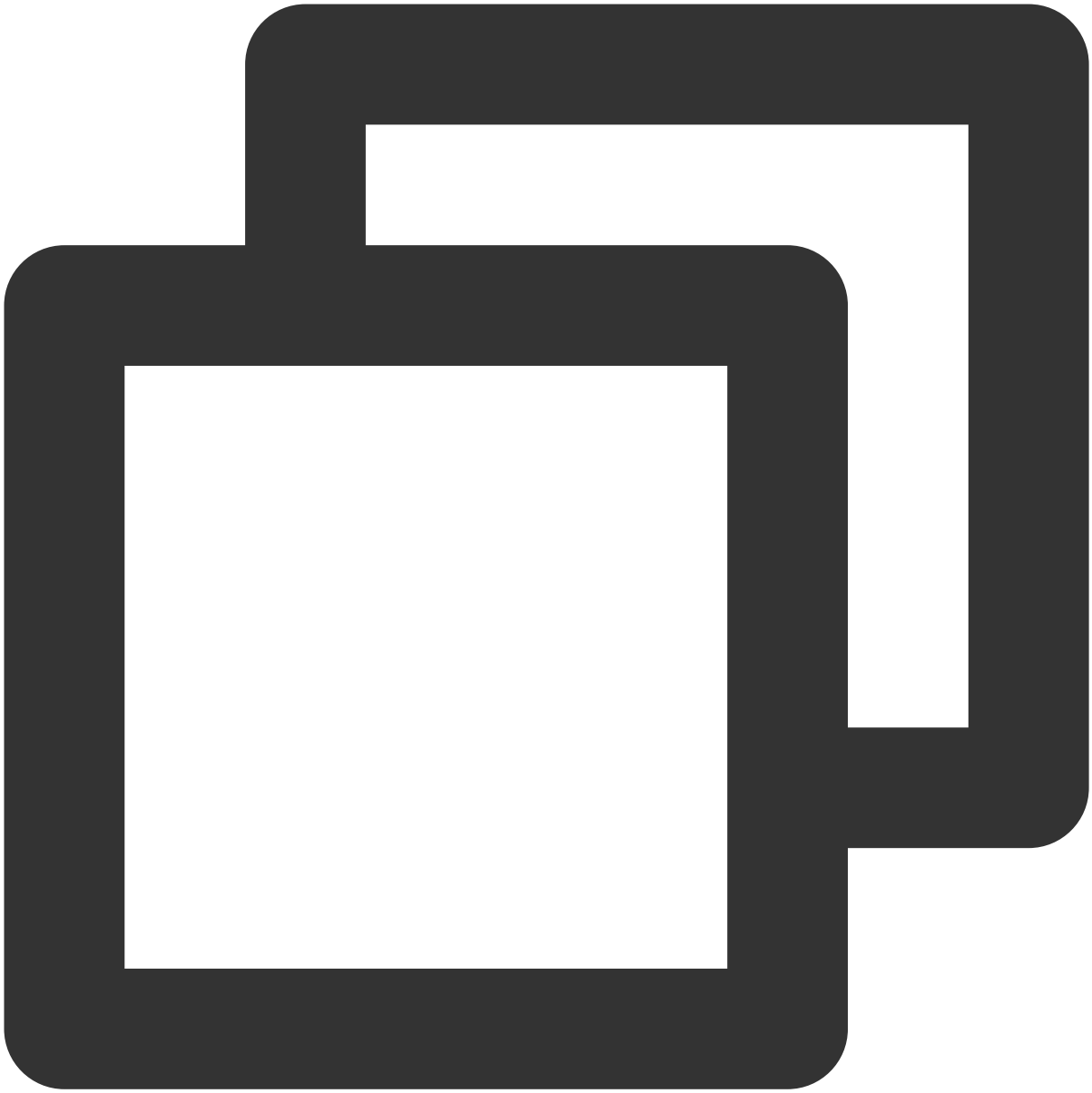
其中和 IM 相关的如统计在线时长、统计消息量等我们都需要使用 IM 的回调来实现。在第一个准备工作的模块中，有配置回调的相关指引，控制台 中具体的回调如图：



统计相关信息的流程图如图：



消息发送后回调数据示例：

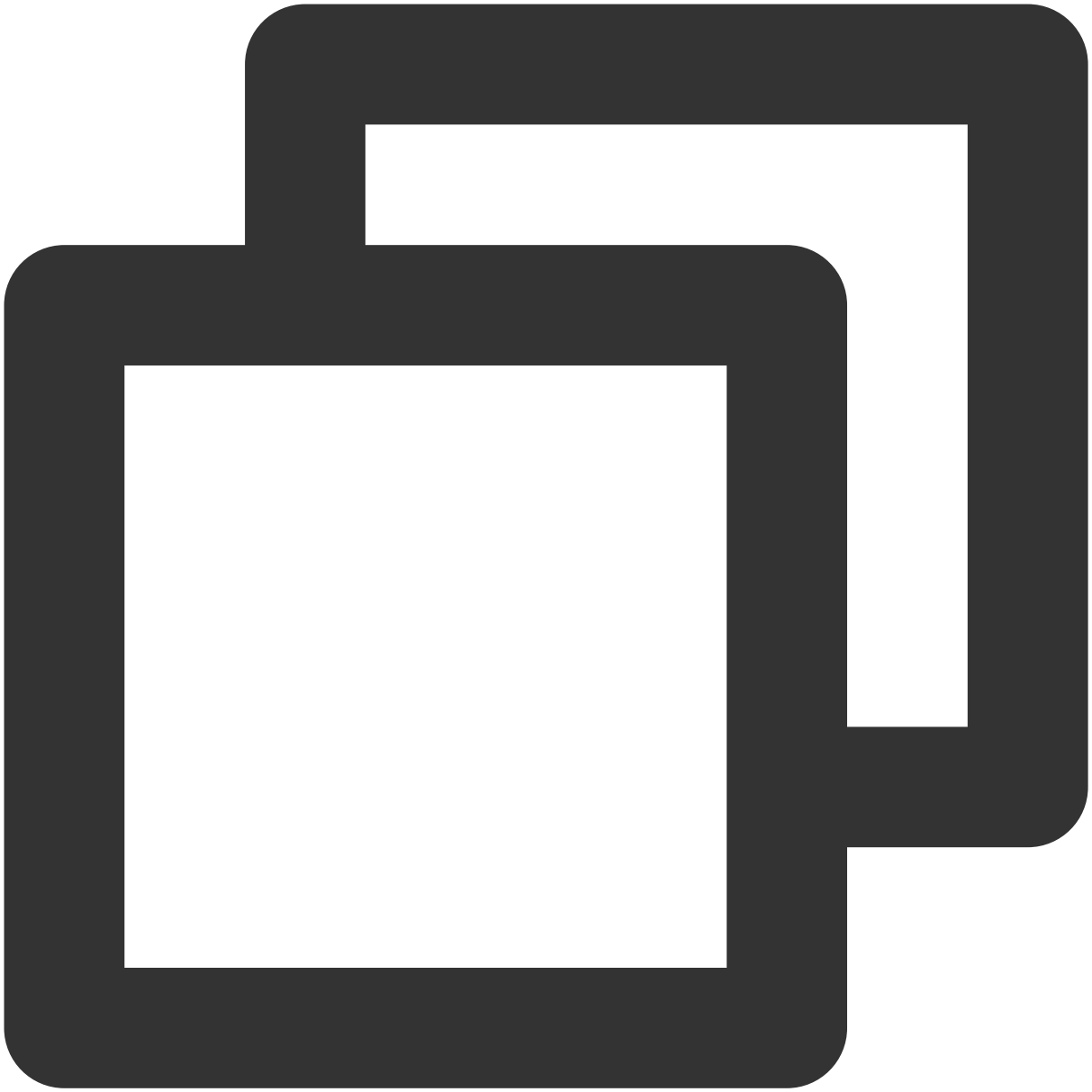


```
{
    "CallbackCommand": "Group.CallbackAfterSendMsg", // 回调命令
    "GroupId": "@TGS#2J4SZEAEL", // 群组 ID
    "Type": "Public", // 群组类型
    "From_Account": "jared", // 发送者
    "Operator_Account":"admin", // 请求的发起者
    "Random": 123456, // 随机数
    "MsgSeq": 123, // 消息的序列号
    "MsgTime": 1490686222, // 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"
            }
        ]
    }
```

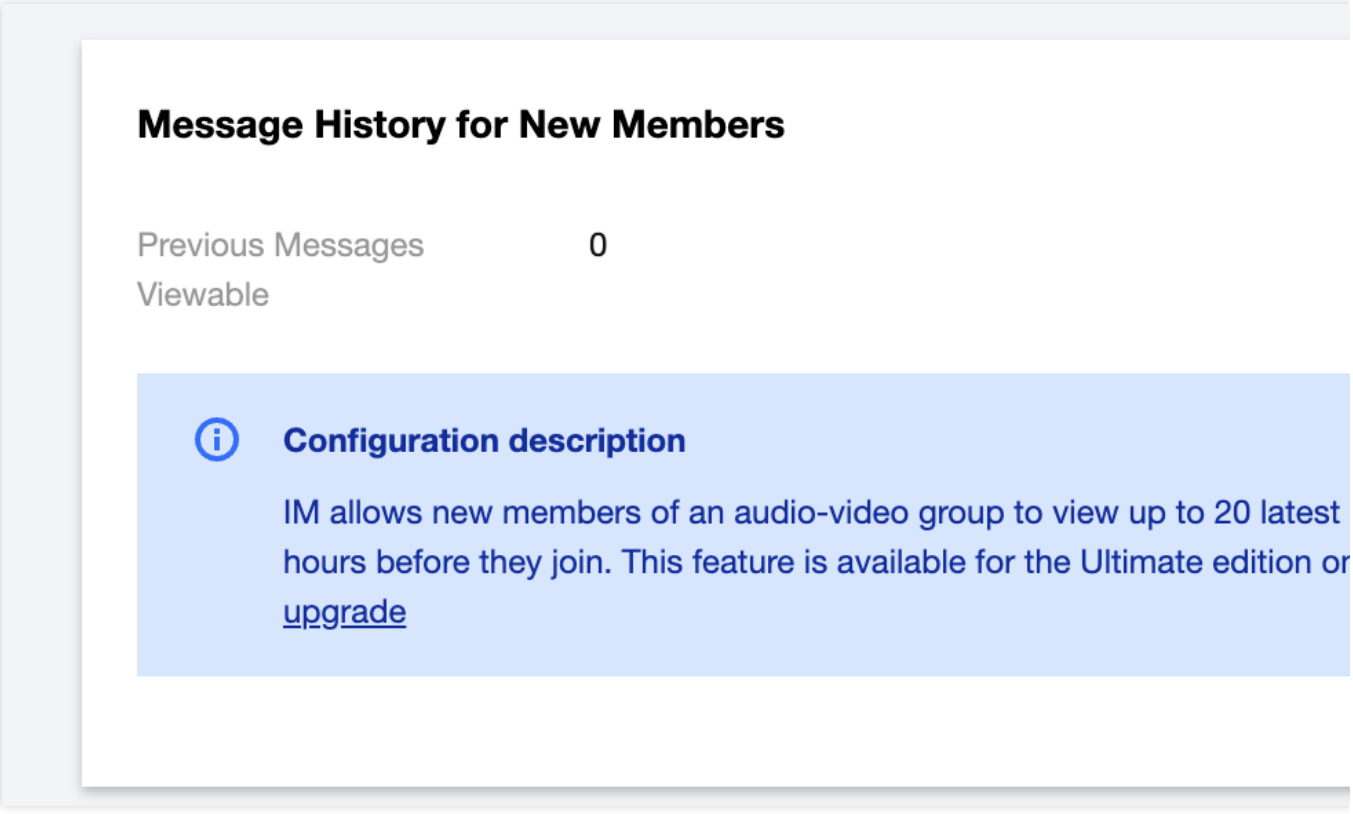
开发者可根据 Info 中的字段判断用户的在线状态，从而统计用户的在线时长。全部字段含义可参见 服务端 API。

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，讲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

此功能仅进阶版用户才可开通，且仅支持拉群24小时内最多20条历史消息。

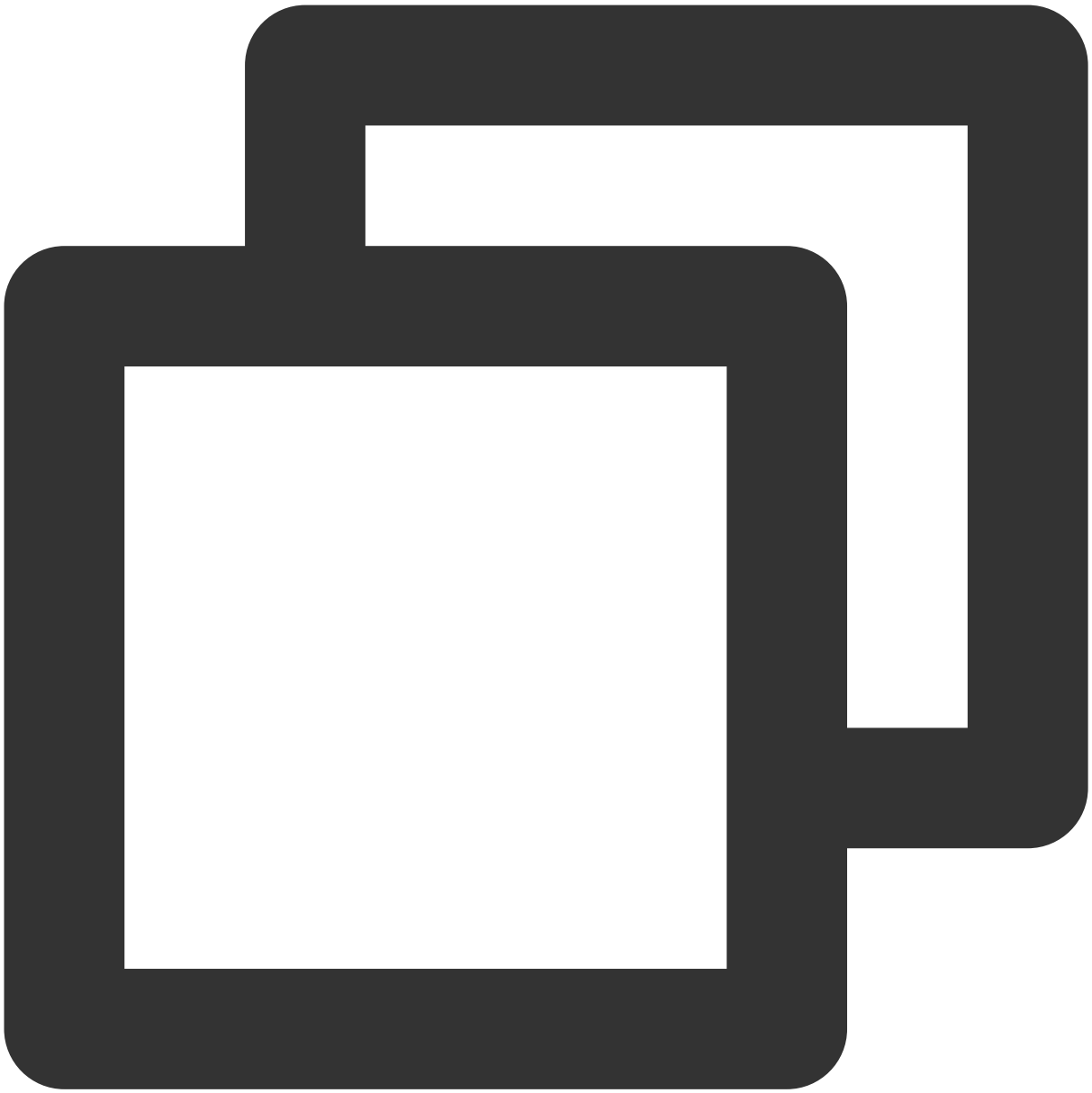
直播群用户拉取进群前历史消息与拉起其他群历史消息一样，代码示例：

Android

iOS&Mac

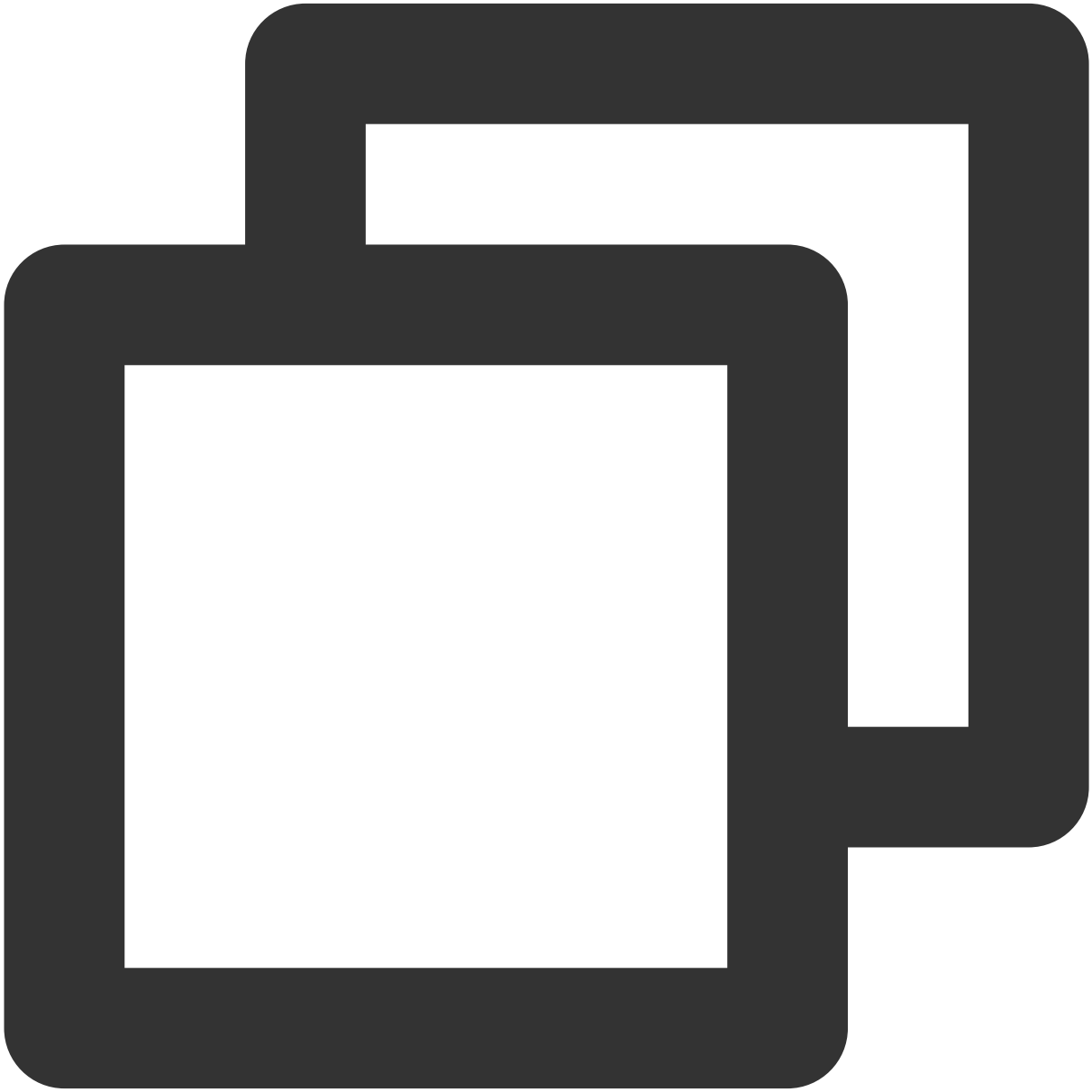
Flutter

Web



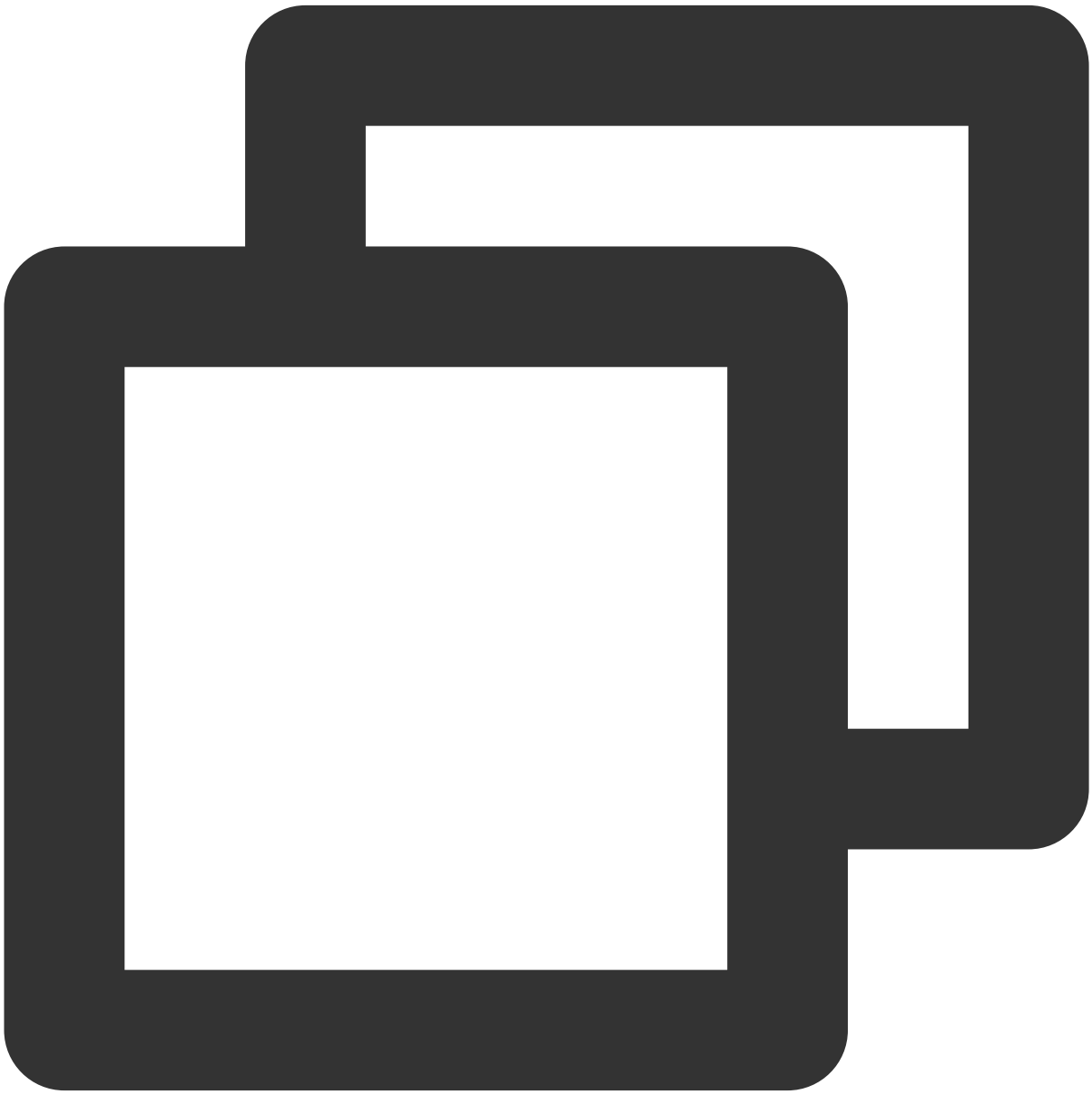
```
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // 拉取云端的
option.setGetTimeBegin(1640966400);          // 从 2022-01-01 00:00:00 开始
option.setGetTimePeriod(1 * 24 * 60 * 60);   // 拉取一整天的消息
option.setCount(Integer.MAX_VALUE);          // 返回时间范围内所有的消息
option.setGroupID(#you group id#);           // 拉取群聊消息
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
```

```
            Log.i("imsdk", "success");
        }

        @Override
        public void onError(int code, String desc) {
            Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
        }
    });
```



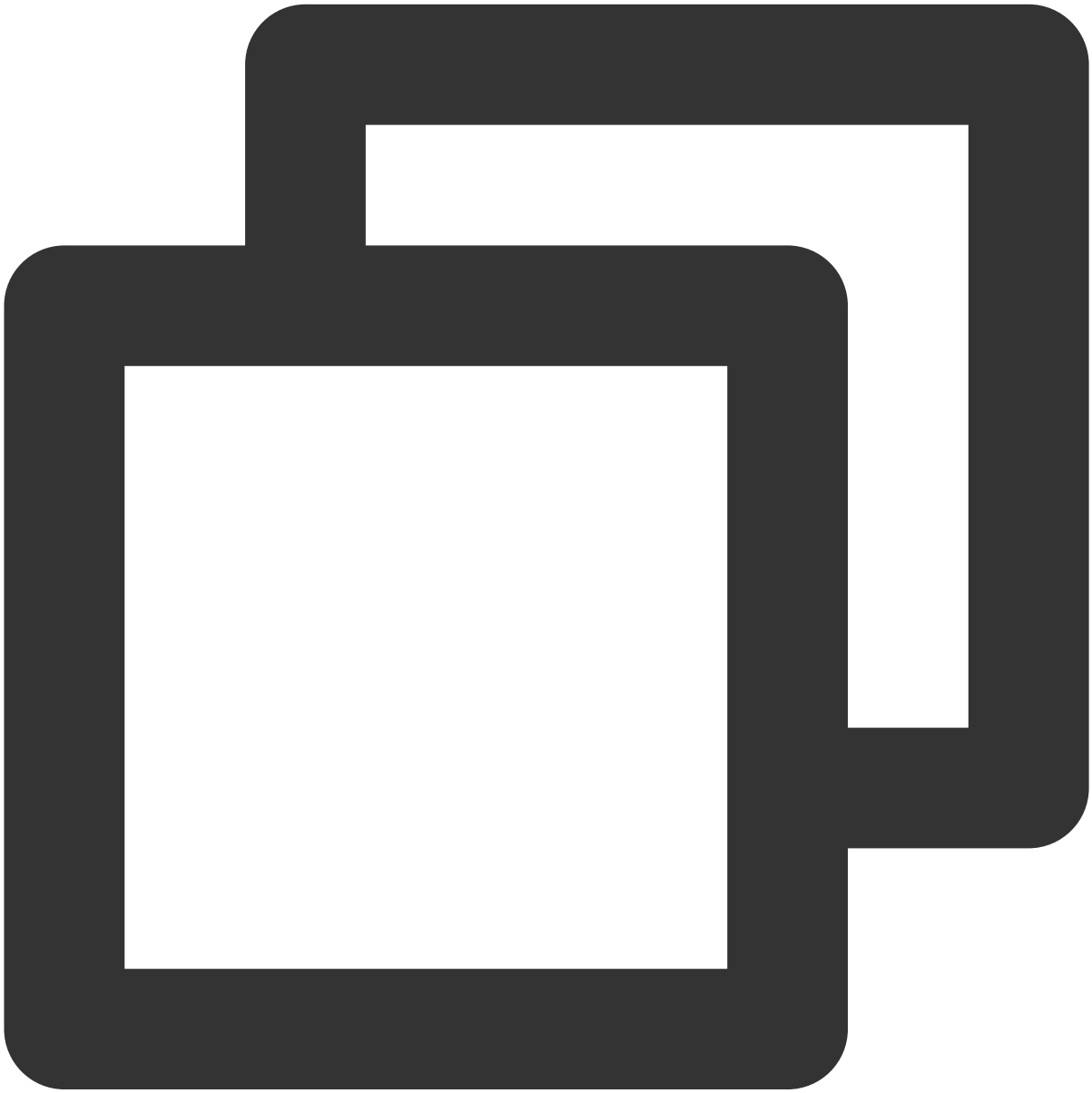
```
// 拉取单聊历史消息
// 首次拉取，lastMsg 设置为 nil
```

```
// 再次拉取时，lastMsg 可以使用返回的消息列表中的最后一条消息
[V2TIMManager.sharedInstance getC2CHistoryMessageList:#your user id# count:20 lastM
    // 记录下次拉取的 lastMsg，用于下次拉取
    V2TIMMessage *lastMsg = msgs.lastObject;
    NSLog(@"success, %@", msgs);
} fail:^(int code, NSString *desc) {
    NSLog(@"fail, %d, %@", code, desc);
}];
```



```
// 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().getC2CHistoryMessageList(
        userID: "userId",
        count: 10,
        lastMsgID: null,
);
```



```
// 打开某个会话时，第一次拉取消息列表
let promise = tim.getMessageList({conversationID: 'C2Ctest', count: 15});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // 消息列表。
```

```
  const nextReqMessageID = imResponse.data.nextReqMessageID; // 用于续拉，分页续拉时需传
  const isCompleted = imResponse.data.isCompleted; // 表示是否已经拉完所有消息。
});
// 下拉查看更多消息
let promise = tim.getMessageList({conversationID: 'C2Ctest', nextReqMessageID, coun
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // 消息列表。
  const nextReqMessageID = imResponse.data.nextReqMessageID; // 用于续拉，分页续拉时需传
  const isCompleted = imResponse.data.isCompleted; // 表示是否已经拉完所有消息。
});
```

拉取历史消息其他 SDK 版本代码示例

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 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，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。

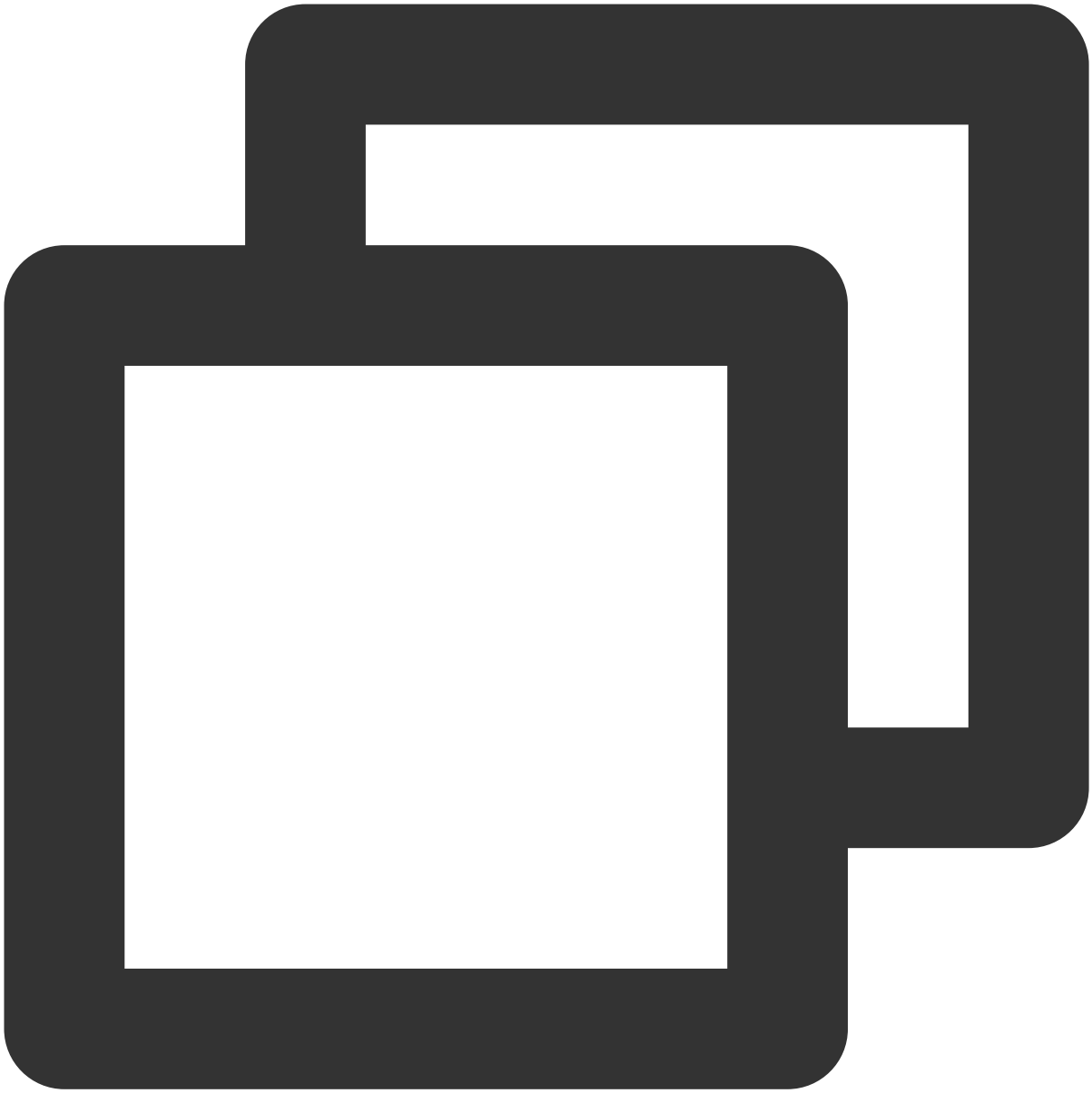
获取直播间在线人数代码如下：

Android

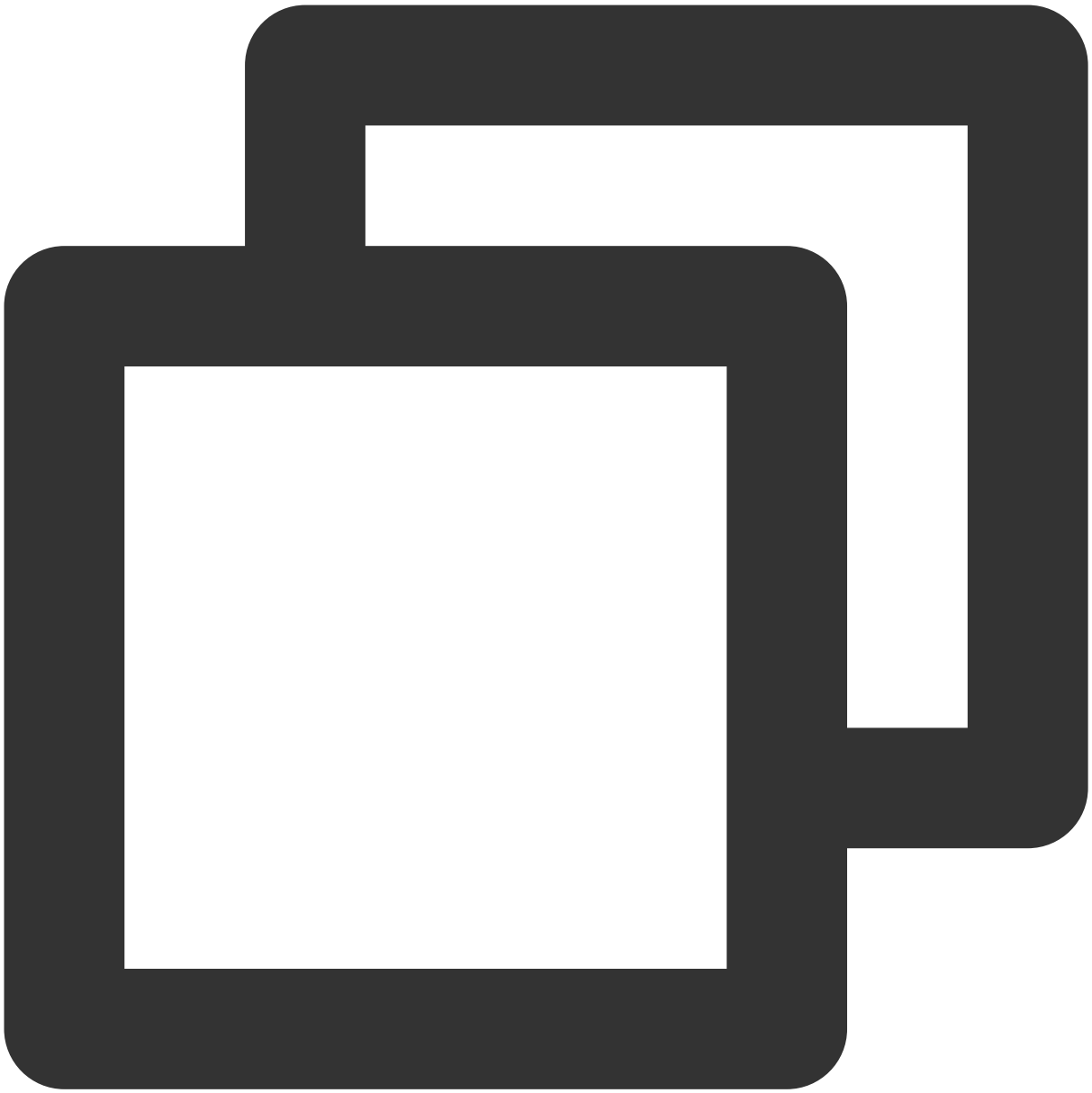
iOS&Mac

Flutter

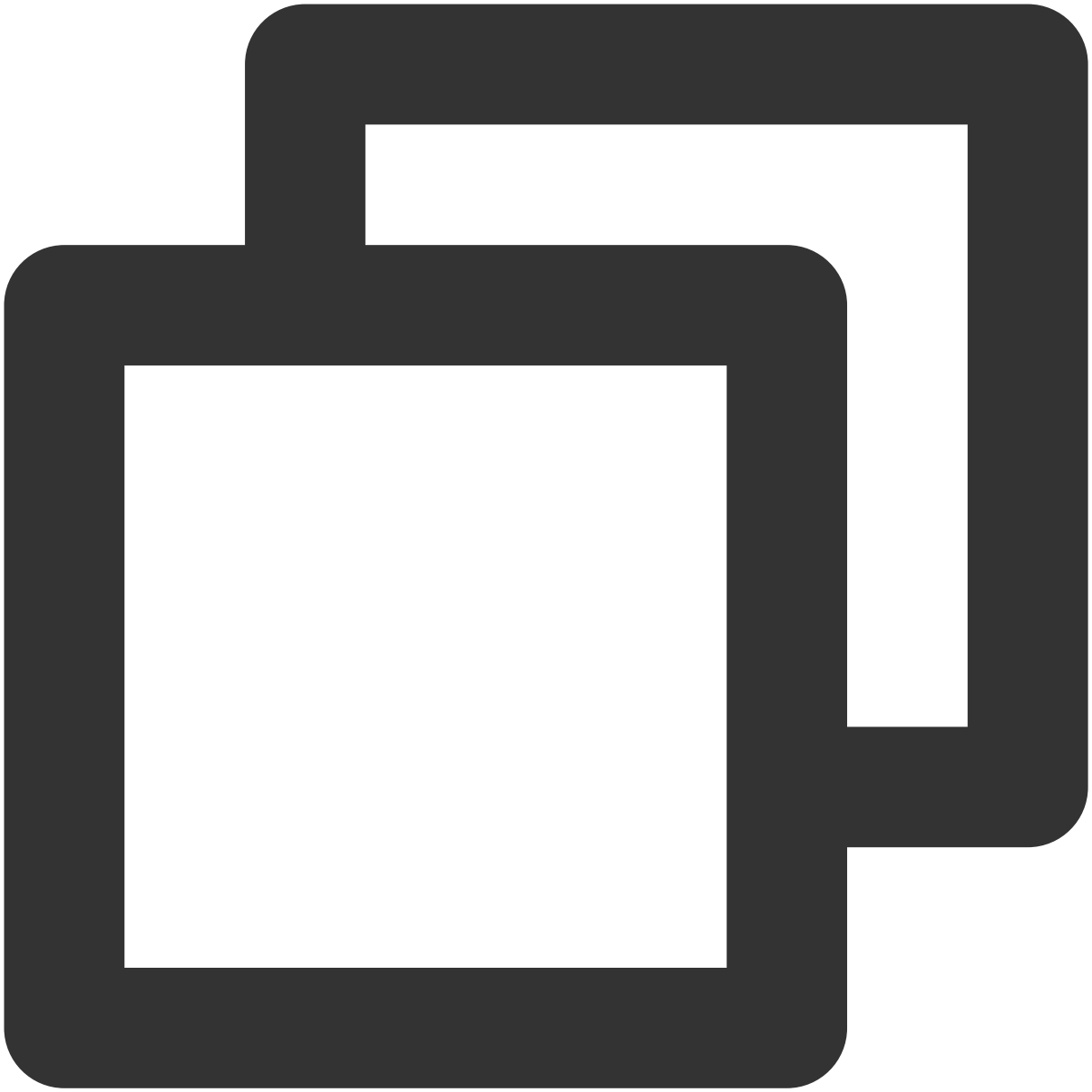
Web



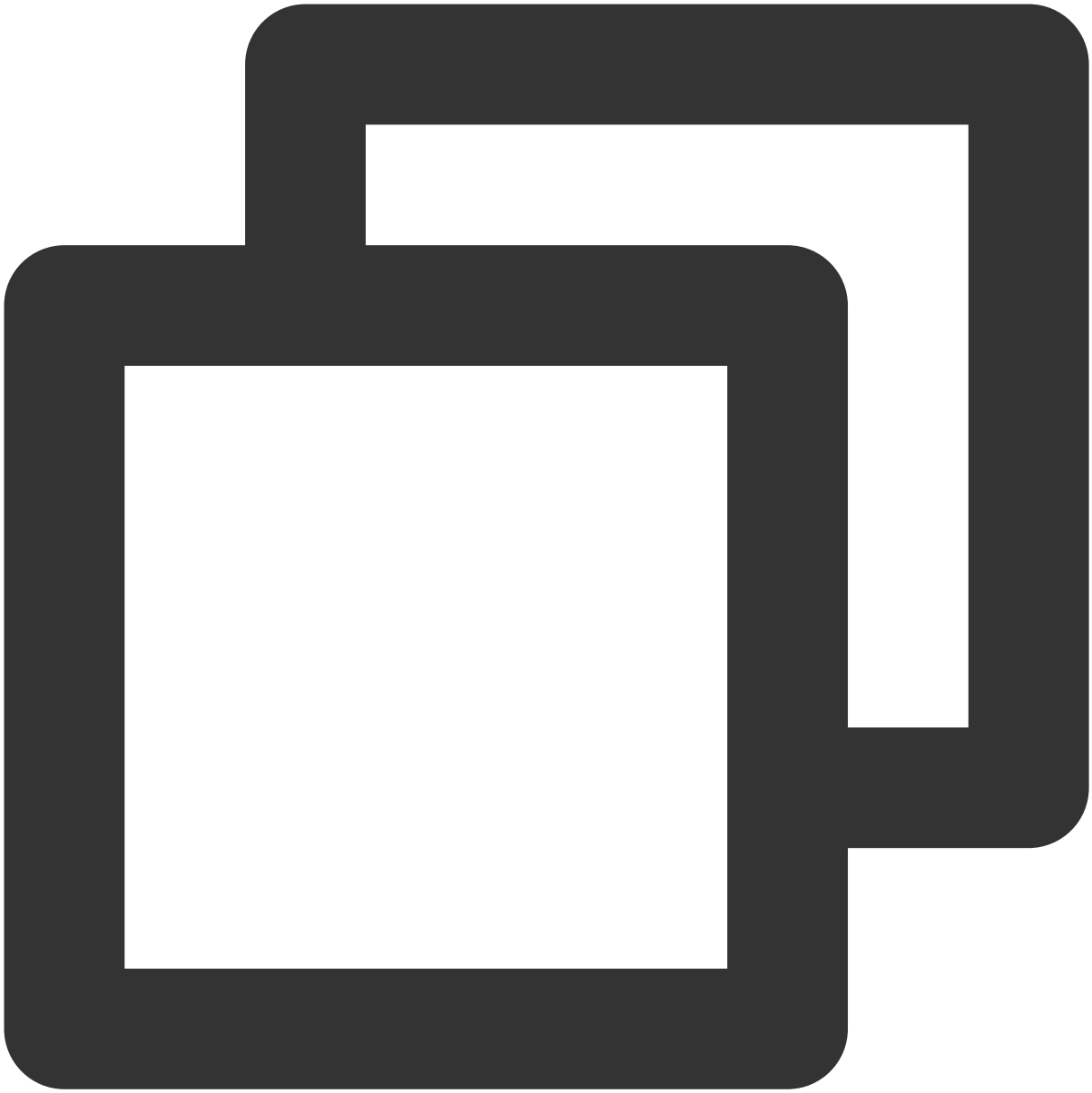
```
2TIMManager.getGroupManager().getGroupOnlineMemberCount("group_avchatroom", new V2T
  @Override
  public void onSuccess(Integer integer) {
      // 获取直播群在线人数成功
  }

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



```
[[V2TIMManager sharedInstance] getGroupOnlineMemberCount:@"group_avchatroom" succ:^
    // 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
}];
```



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



```
// 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
});
```

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。

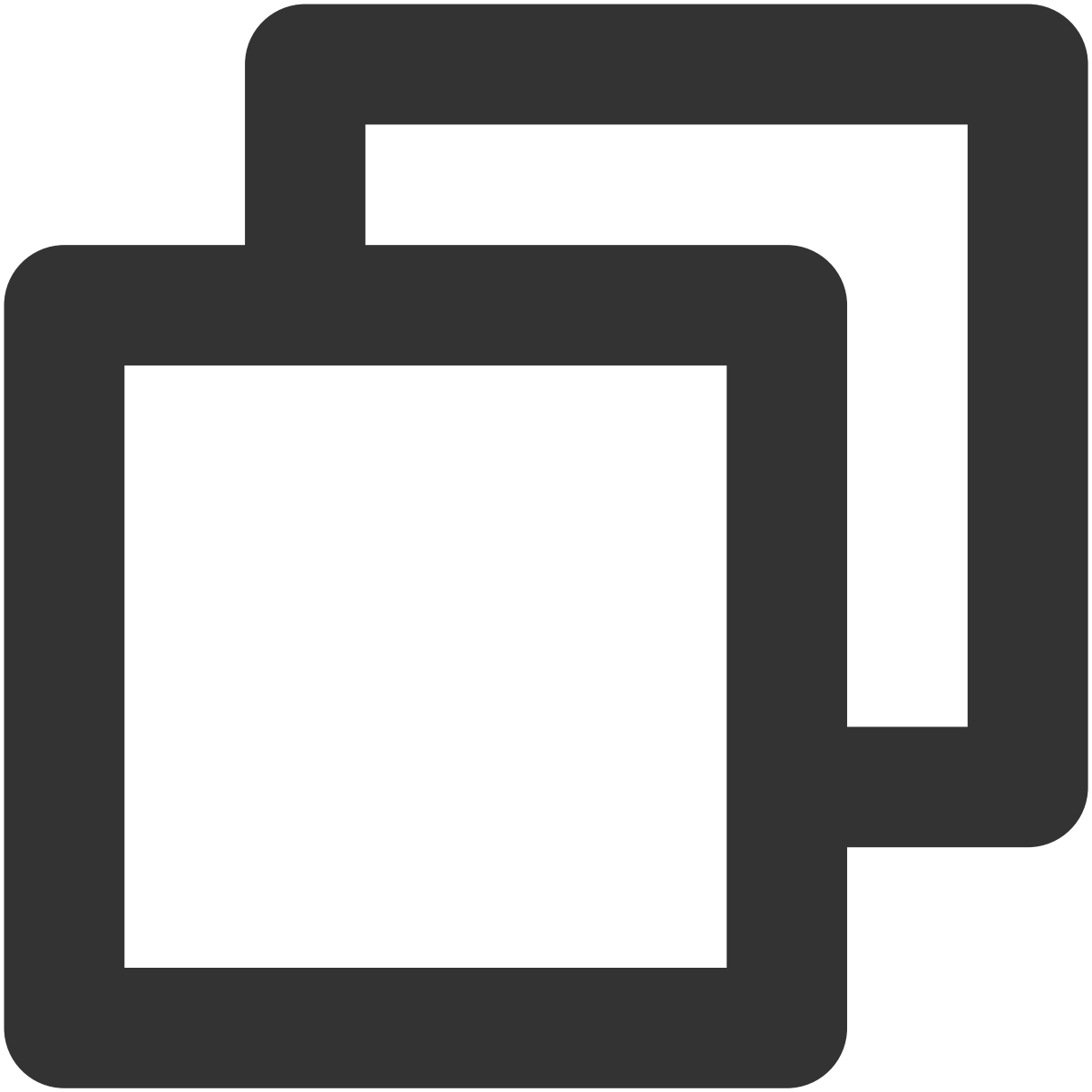
客户端对应的回调代码如下：

Android

iOS&Mac

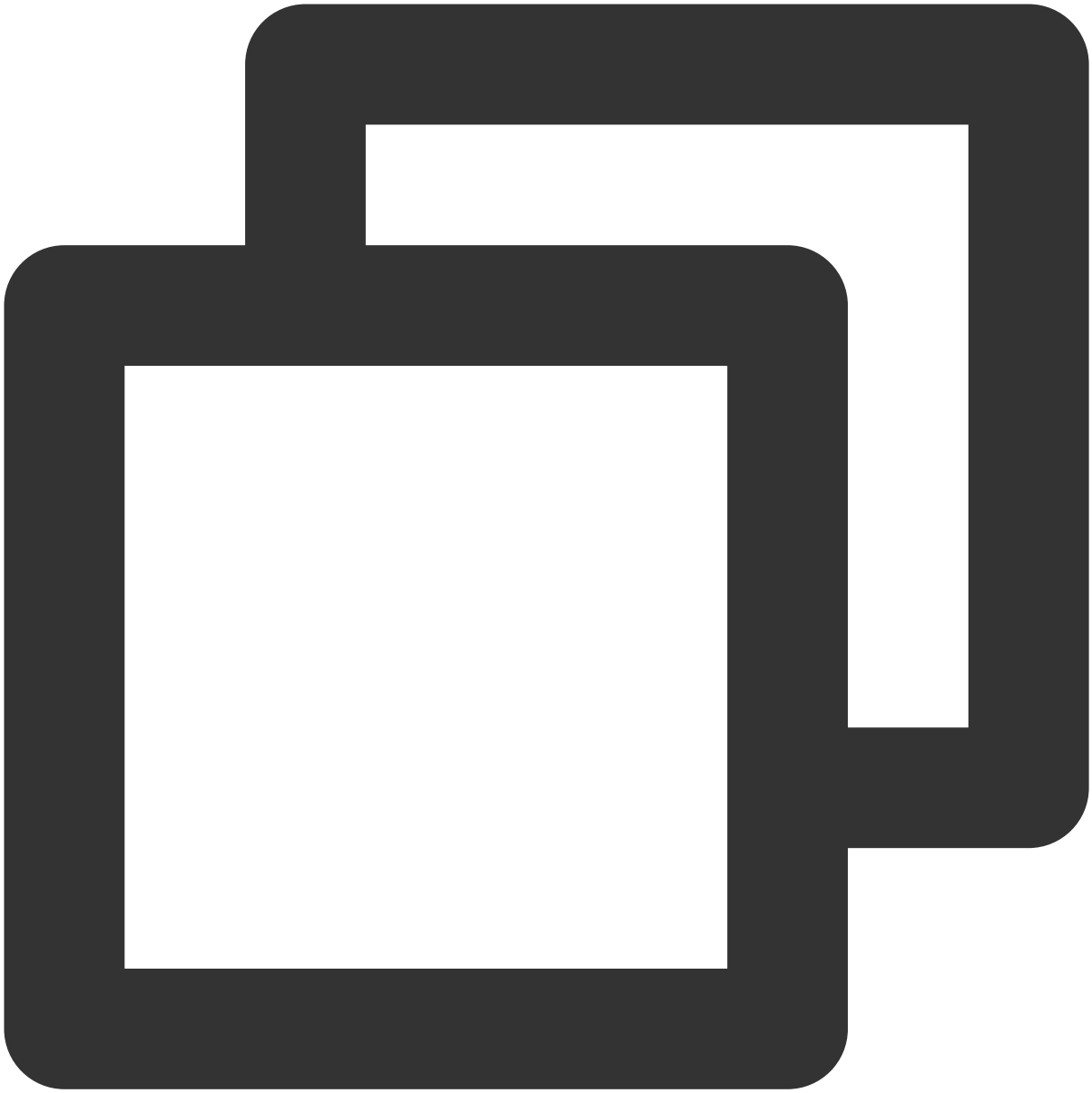
Flutter

Web



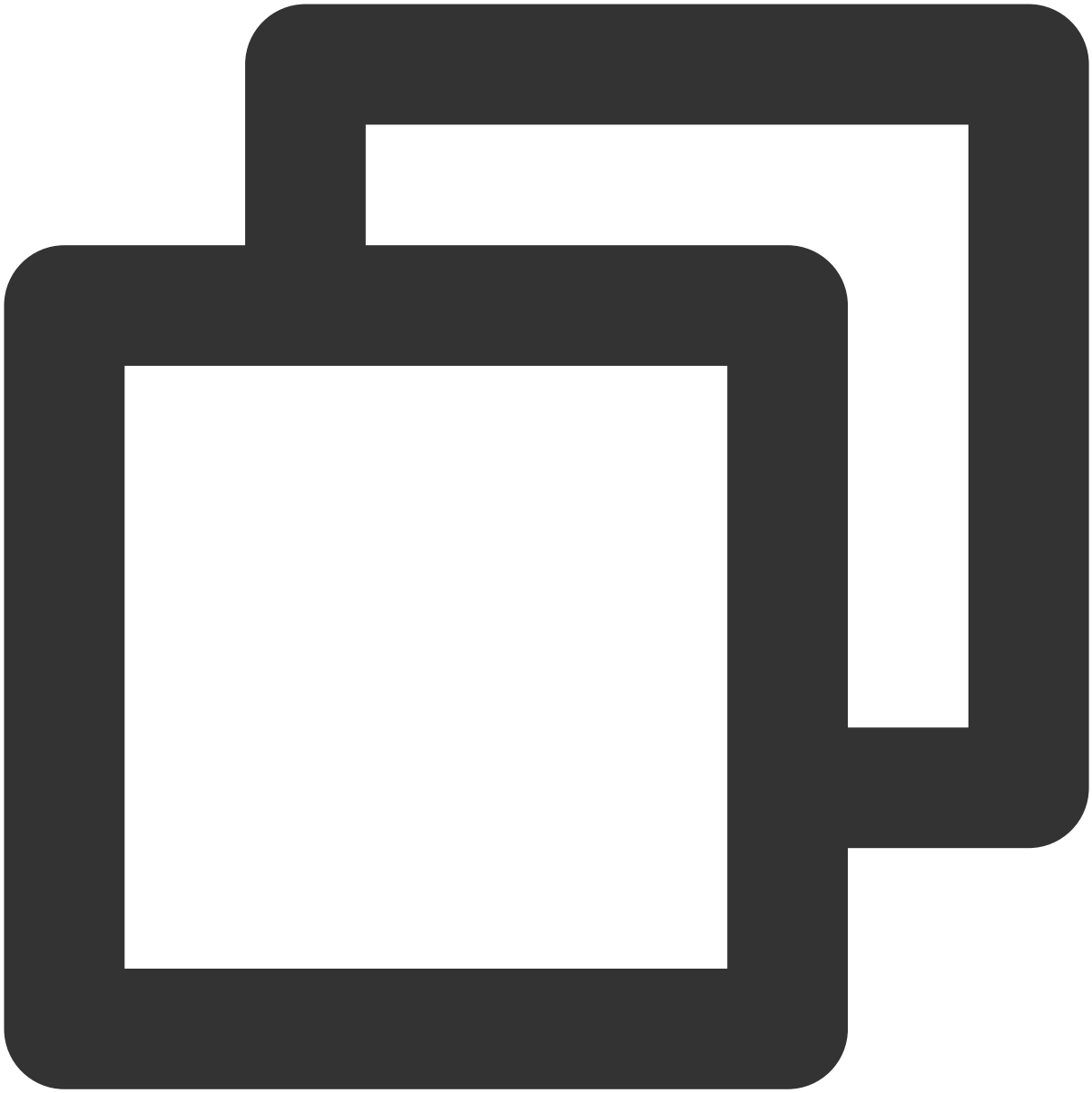
```
// 禁言群成员 userB 1分钟
V2TIMManager.getGroupManager().muteGroupMember("groupA", "userB", 60, new V2TIMCall
  @Override
  public void onSuccess() {
      // 禁言群成员成功
  }

  @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>
    // 禁言群成员监听
    for (V2TIMGroupMemberChangeInfo memberChangeInfo : v2TIMGroupMemberChangeInfoLi
      // 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> changeI
    // 全员禁言监听
    for (V2TIMGroupChangeInfo groupChangeInfo : changeInfos) {
      if (groupChangeInfo.getType() == V2TIMGroupChangeInfo.V2TIM_GROUP_INFO_CHANGE
        // 是否全员禁言
        boolean isMuteAll = groupChangeInfo.getBoolValue();
      }
    }
  }
});
```



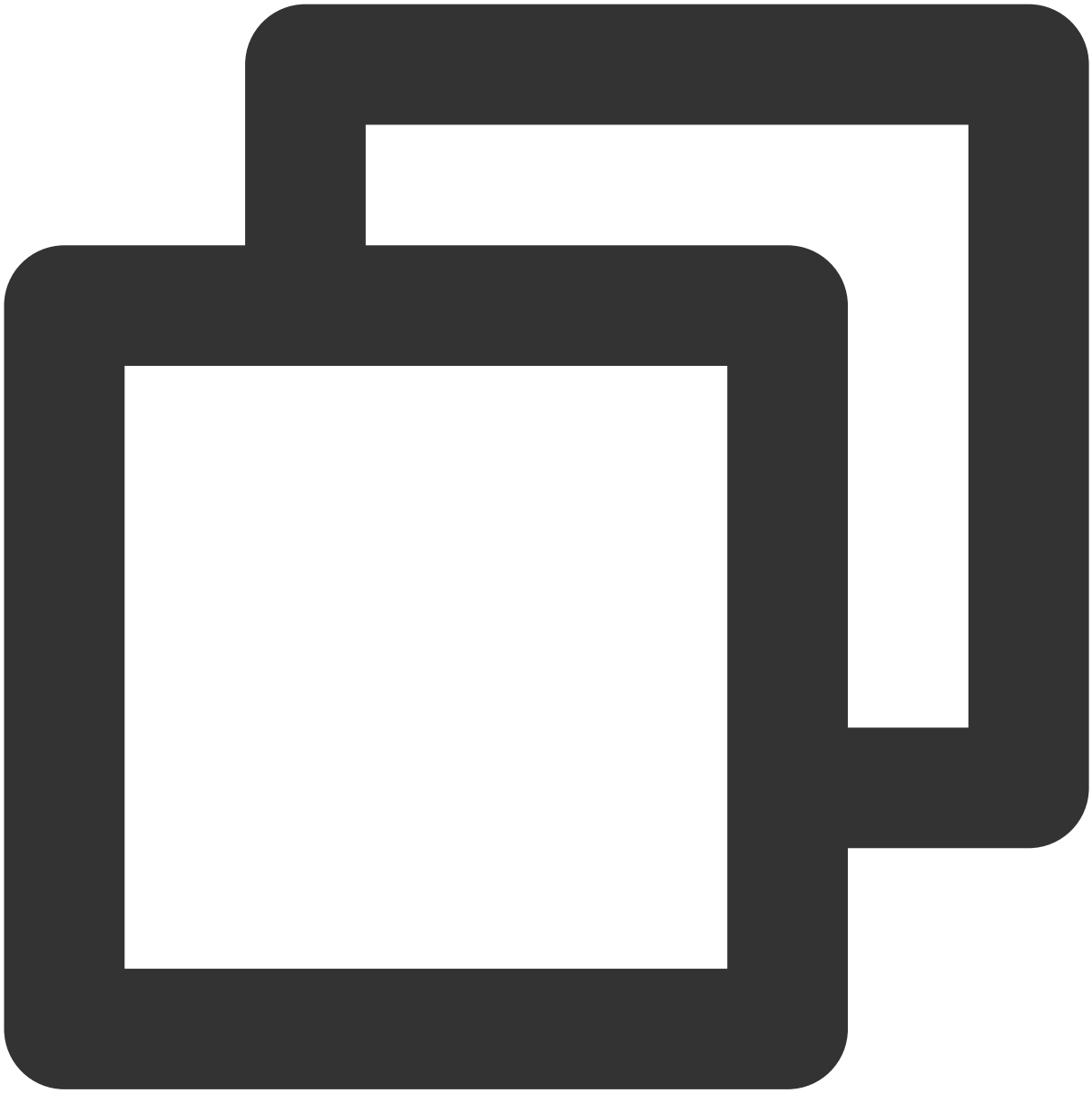
```
// 禁言群成员 user1 1分钟
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA" member:@"user1" muteTime:6
    // 禁言群成员成功
} fail:^(int code, NSString *desc) {
    // 禁言群成员失败
}];

// 全员禁言
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"groupA";
info.allMuted = YES;
```

```
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA" member:@"user1" muteTime:6
    // 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 <V2TIMGroup
    // 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 <V2TIMGroupC
    // 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;
      }
    }
}
```



```
// 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: '',groupTy

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

```
  }
  ));
```

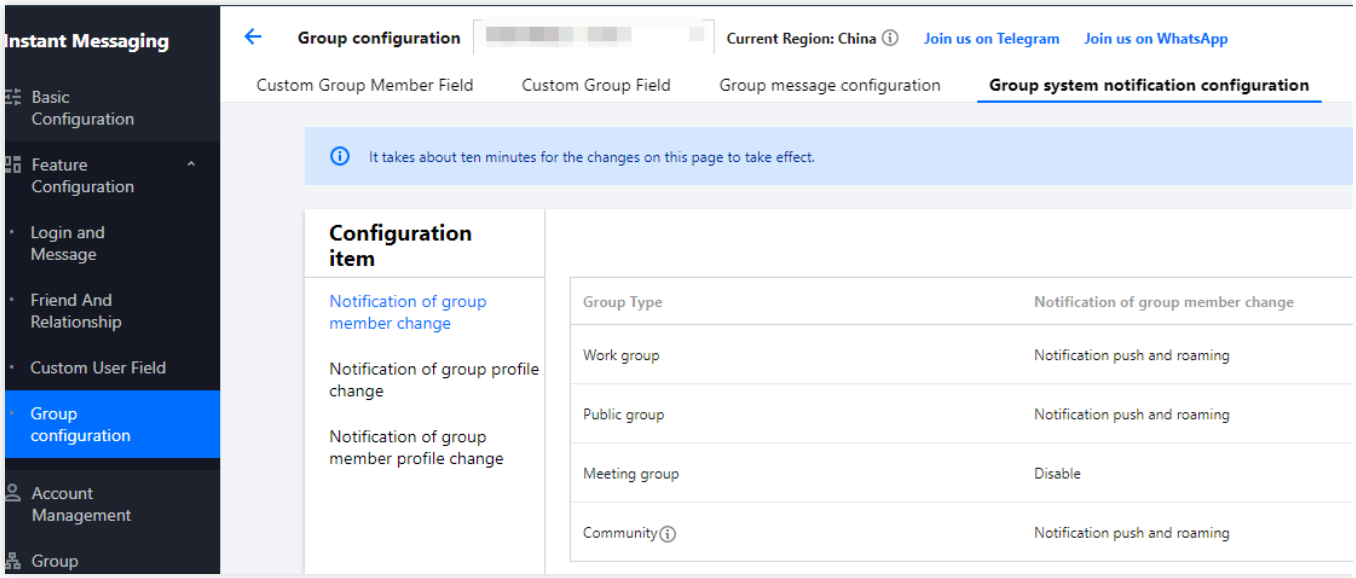


```
tim.setGroupMemberMuteTime(options);
let promise = tim.setGroupMemberMuteTime({
  groupID: 'group1',
  userID: 'user1',
  muteTime: 600 // 禁言10分钟；设为0，则表示取消禁言
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group); // 修改后的群资料
```

```
  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

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。

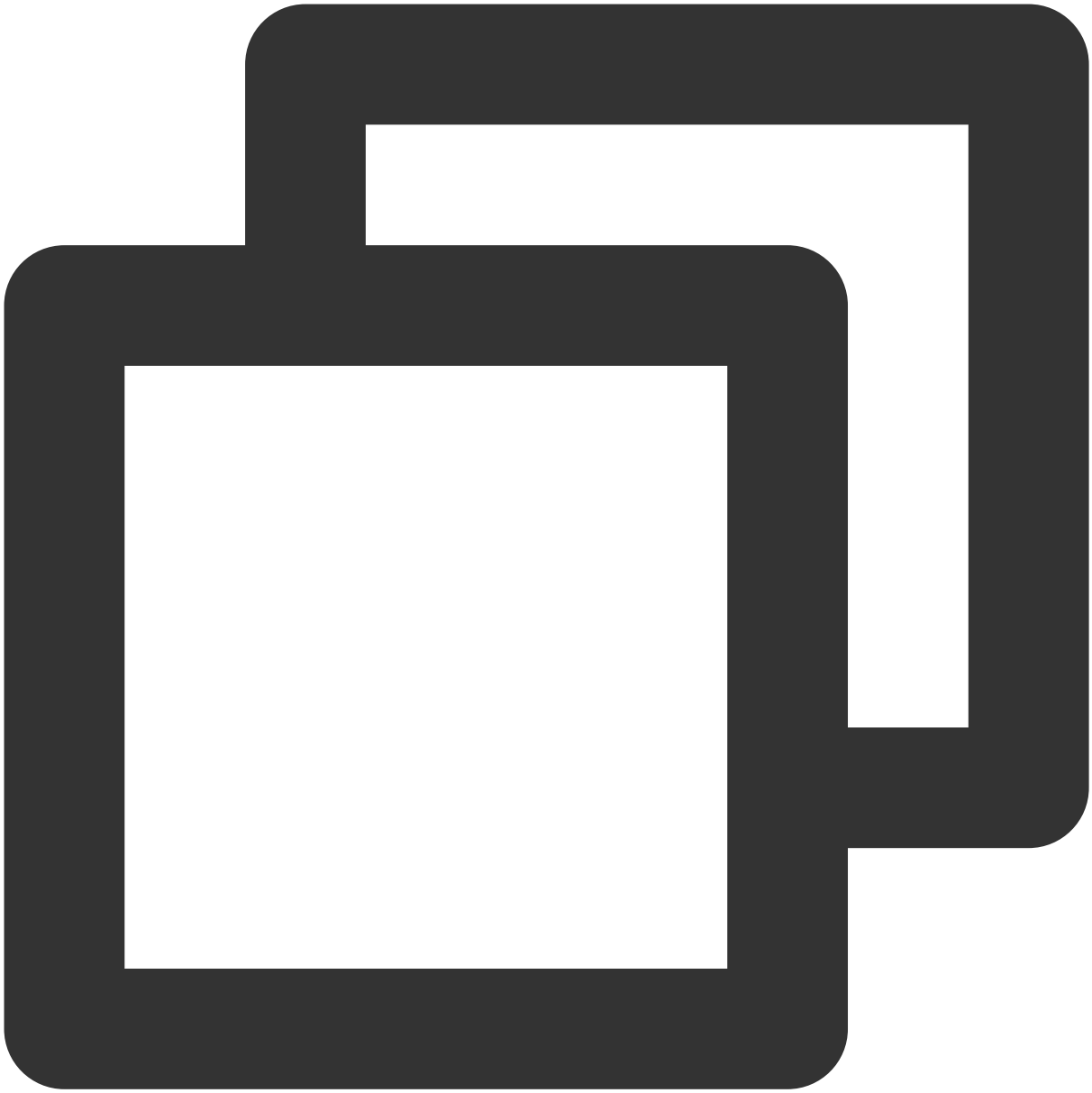
可参考代码如下：

Android

iOS&Mac

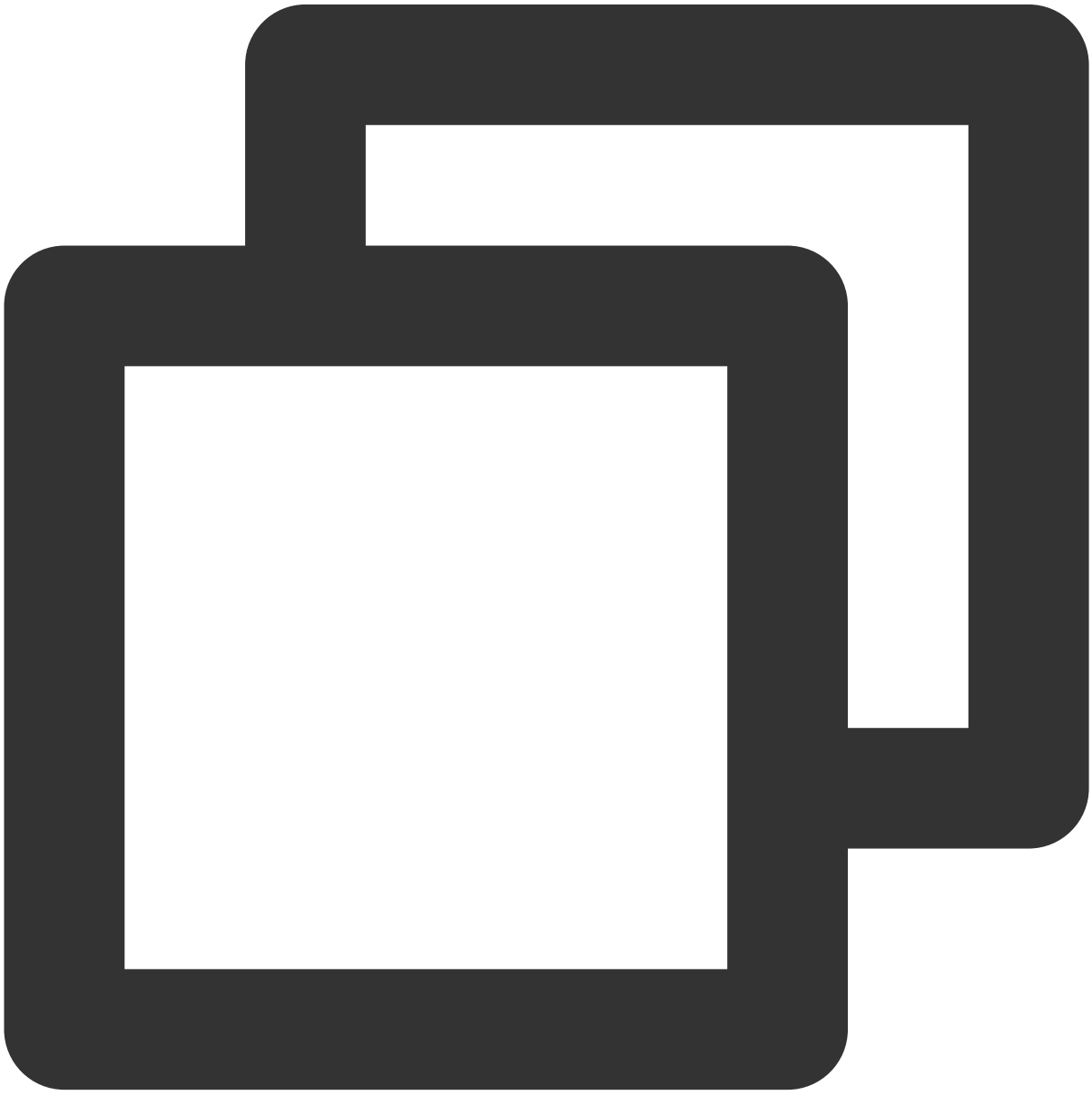
Flutter

Web

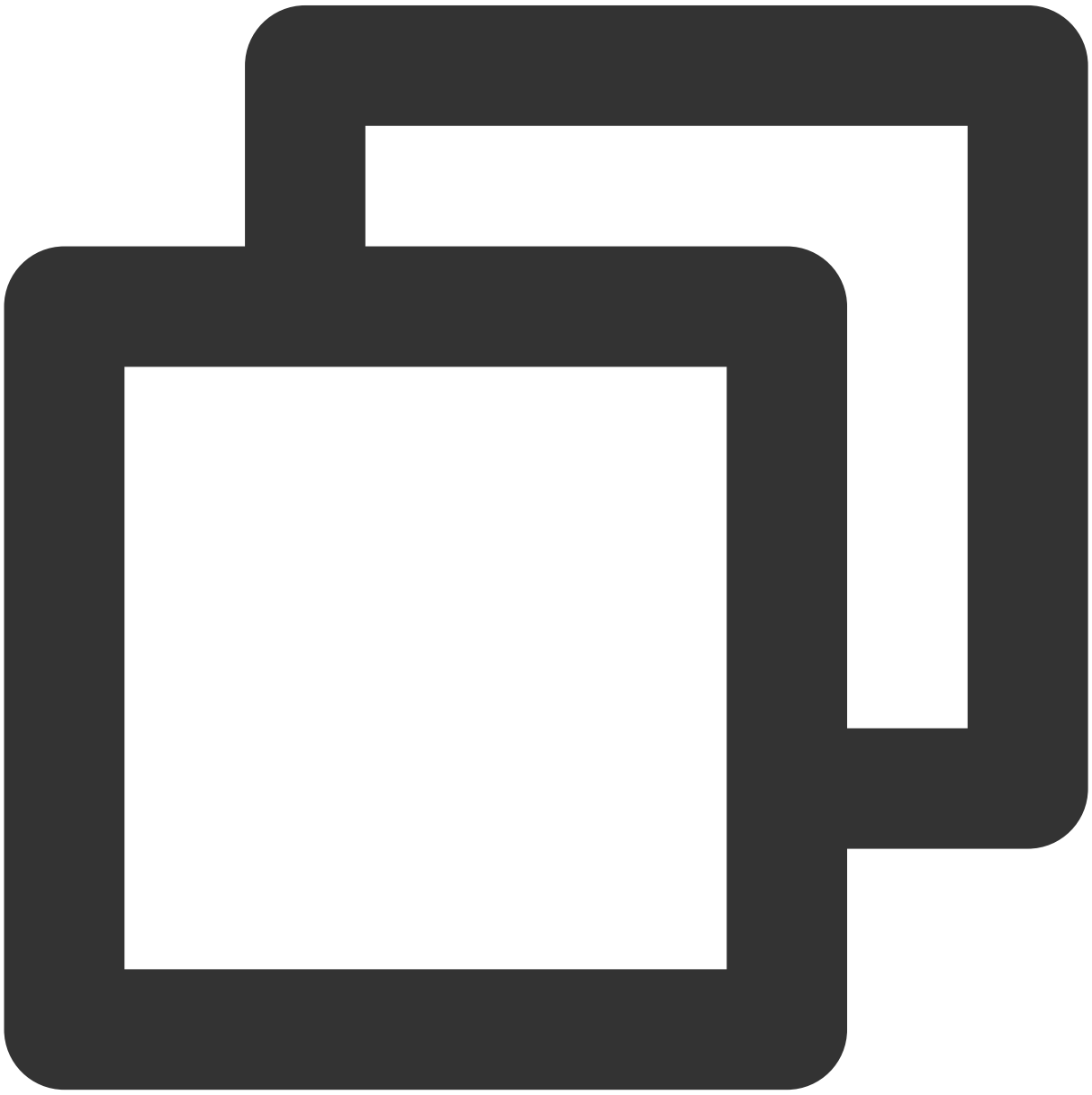


```
List<String> userIDList = new ArrayList<>();
userIDList.add("userB");
V2TIMManager.getGroupManager().kickGroupMember("groupA", userIDList, "", new V2TIMV
  @Override
  public void onSuccess(List<V2TIMGroupMemberOperationResult> v2TIMGroupMemberOpera
      // 踢人成功
  }

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

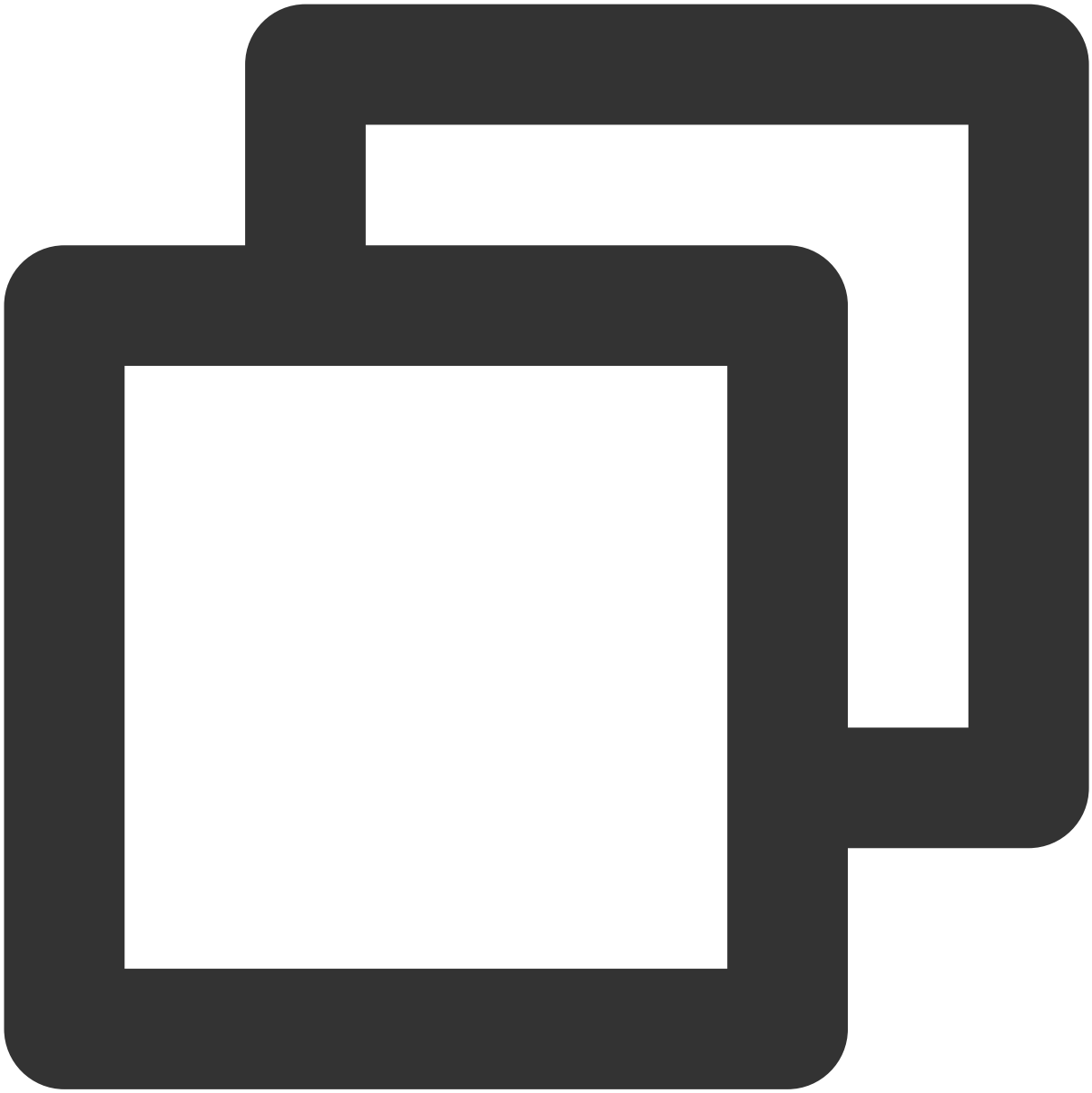
```
    }
});

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



```
[[V2TIMManager sharedInstance] kickGroupMember:@"groupA" memberList:@[@"user1"] rea
    // 踢人成功
} fail:^(int code, NSString *desc) {
    // 踢人失败
}];

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



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



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

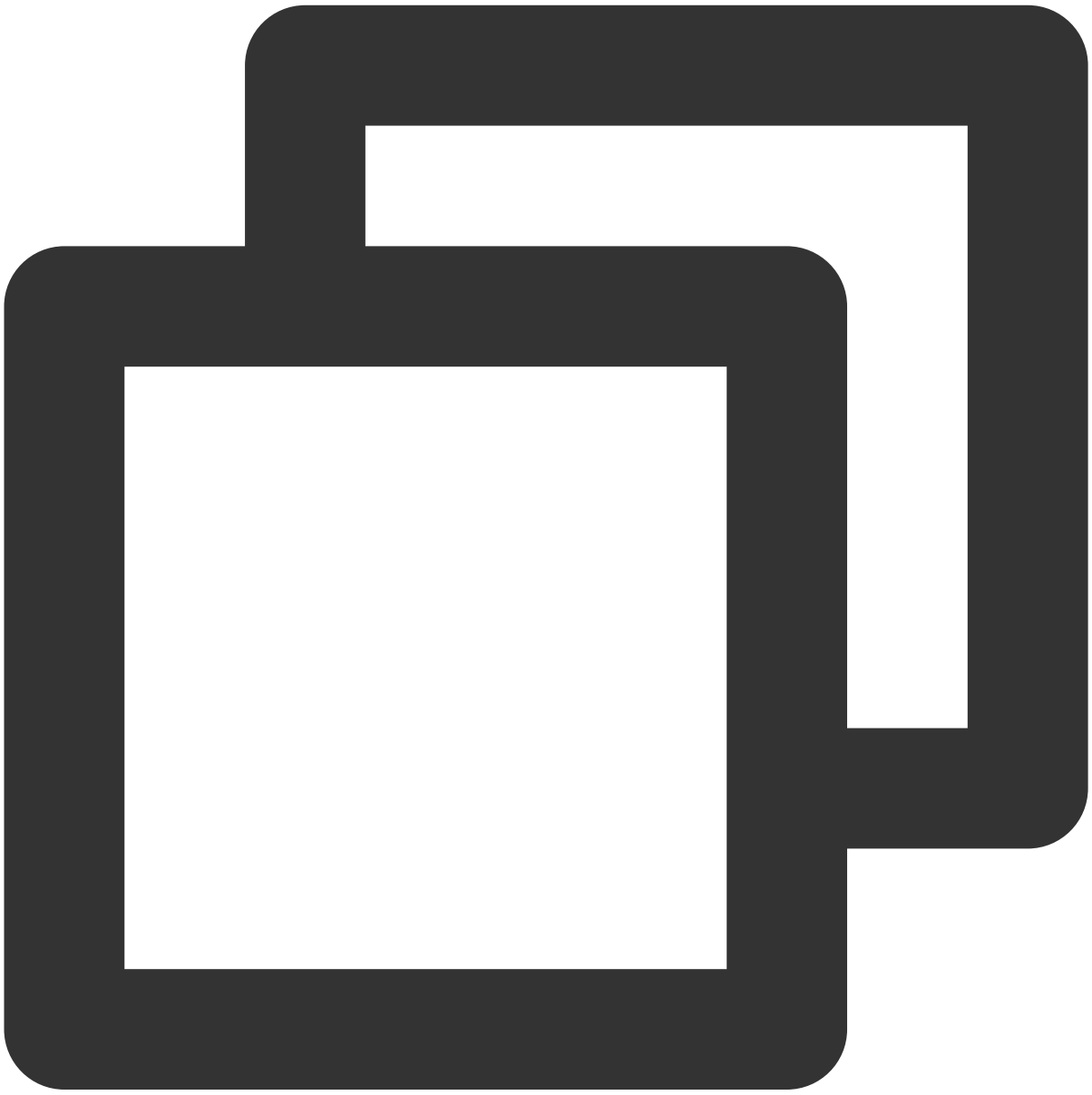
直播间敏感内容过滤

过滤直播间敏感内容也是直播业务非常重要的功能，实现方案如下：

1. 绑定用户群发消息前回调
2. 判断通过回调数据判断消息类型，将消息数据传递给天御或者其他第三方检测服务

3. 如消息为普通文本消息，可等天御检测同步返回，在将是否下发消息的数据包返回给 IM 后台
4. 如消息为多媒体消息，可以直接返回下发消息的数据包给 IM 后台。并且等待天御异步返回的结果返回后，如果发现消息不合规，可通过消息编辑接口或群自定义消息接口下发消息变更通知。客户端收到通知后，在屏蔽掉不合规的消息。

发送消息前回调数据示例：

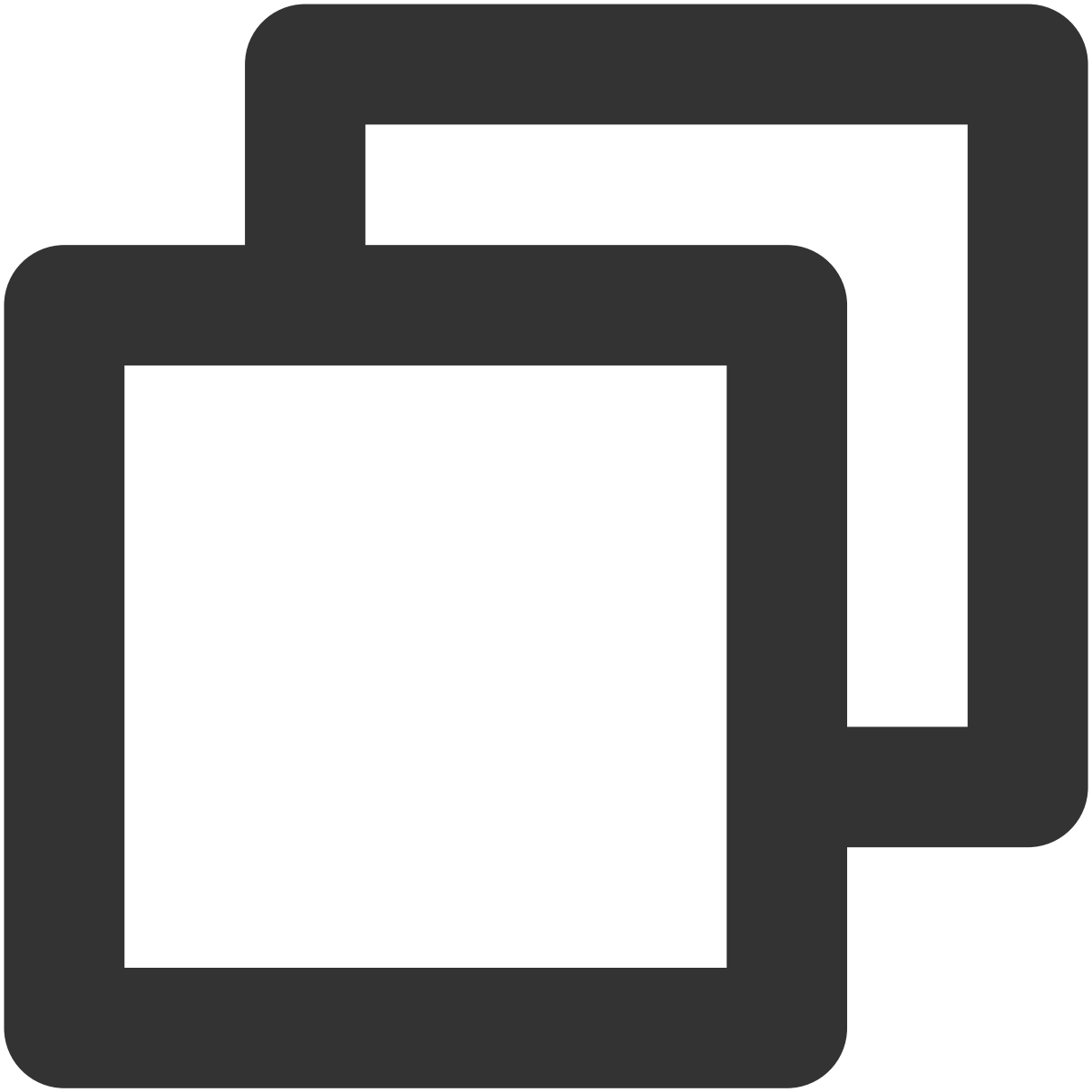


```
{
    "CallbackCommand": "Group.CallbackBeforeSendMsg", // 回调命令
    "GroupId": "@TGS#2J4SZEAEL", // 群组 ID
    "Type": "Public", // 群组类型
```

```
    "From_Account": "jared", // 发送者
    "Operator_Account":"admin", // 请求的发起者
    "Random": 123456, // 随机数
    "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"
}
```

开发者可通过 MsgBody 中 MsgType 字段判断消息类型。完整字段说明可参见 回调文档。

开发者可以选择对不合法的消息进行不一样的处理，可在回调中回包给 IM 后台来控制。

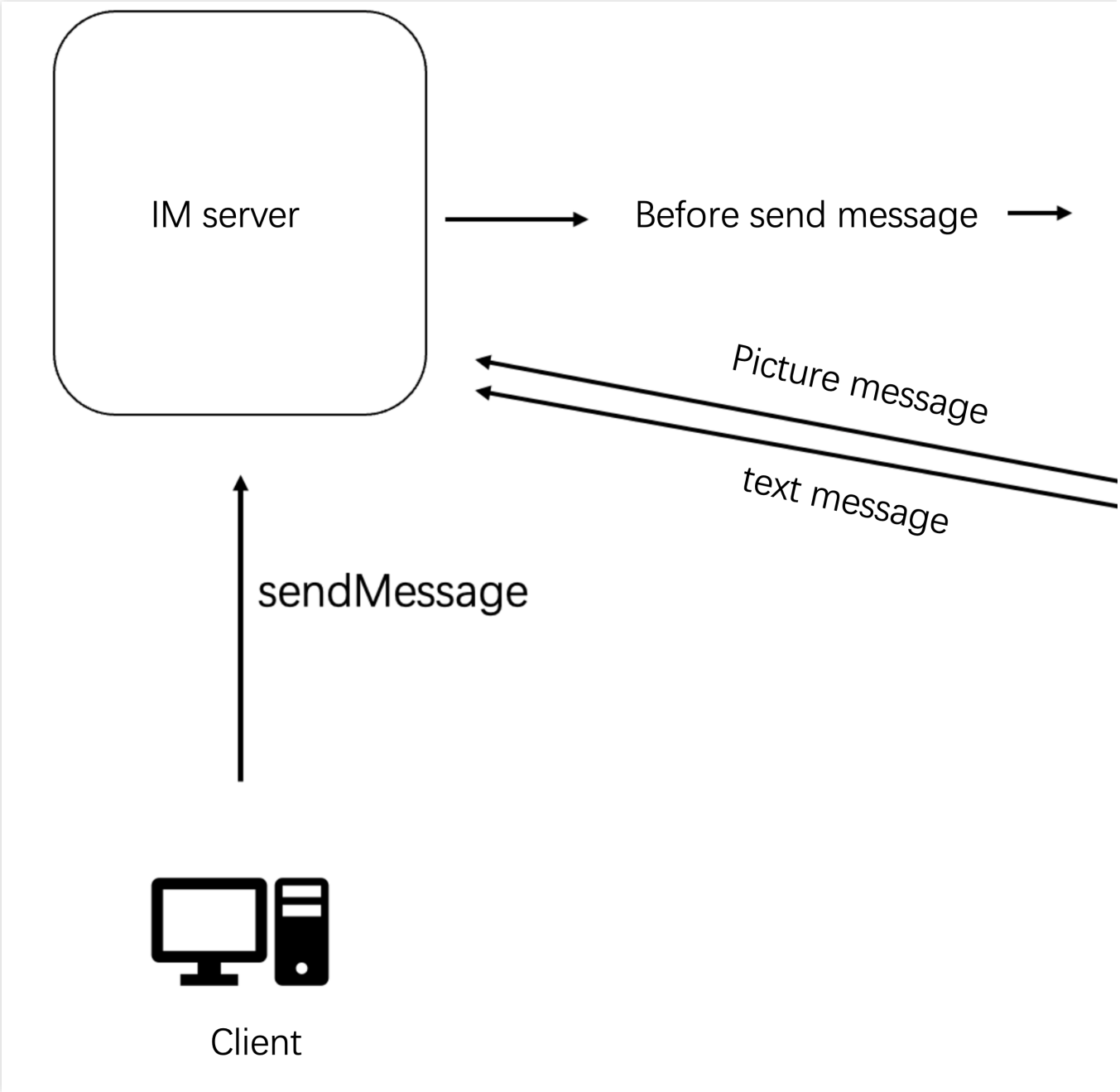


```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // 不同的ErrorCode有不同的含义
}
```

ErrorCode	含义
0	为允许发言，正常下发消息
1	为拒绝发言，客户端返回10016

2	为静默丢弃，客户端返回正常

开发者可根据自身业务进行选择使用
敏感内容检测流程图如图所示：

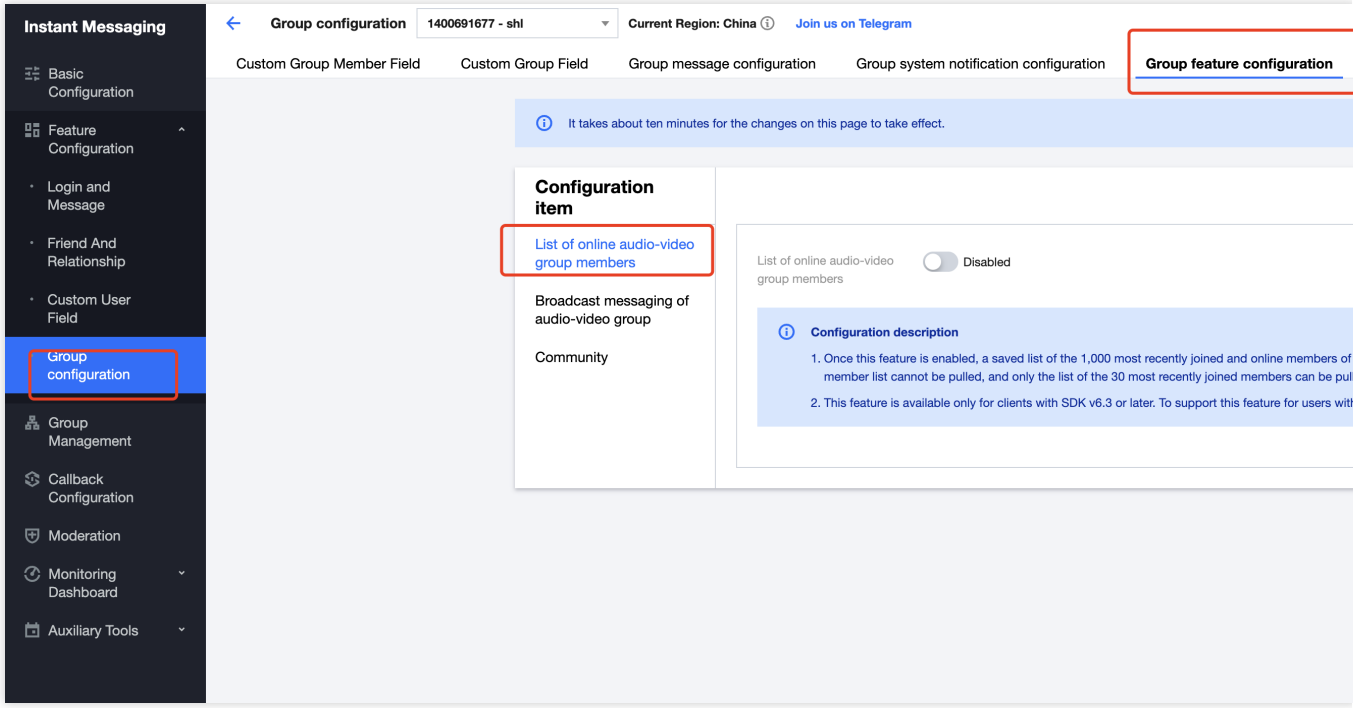


直播间群成员列表

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。根据进阶版和非进阶版区分不同的表现：

1. 非进阶版客户可调用 `getGroupMemberList` 拉取最近进群的 30 位群成员。
2. 进阶版客户可调用 `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，默认跟非进阶版一样，仅拉取最近进群 30 位群成员。

控制台配置如图2.6：



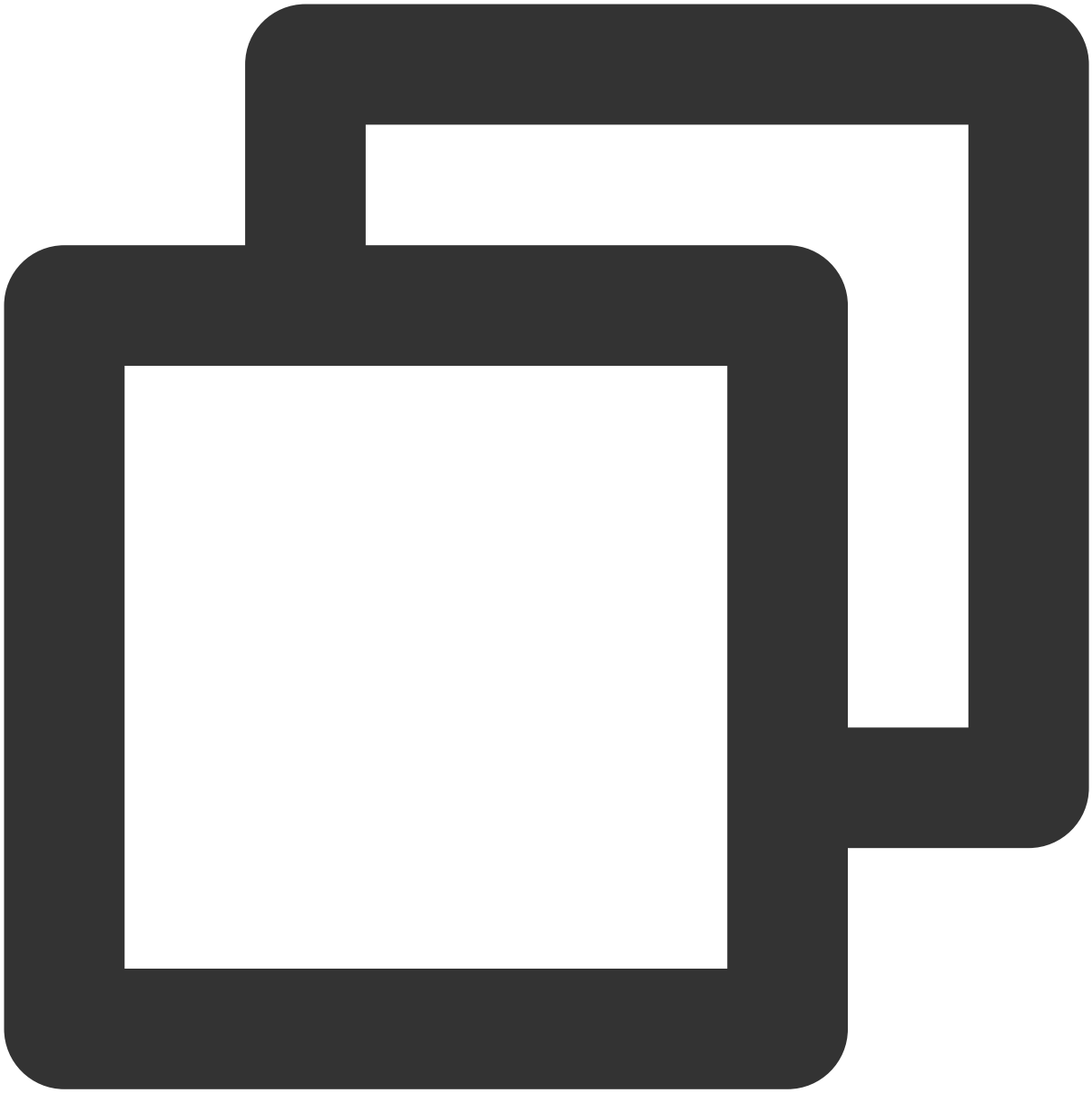
如果是非进阶版，开发者也可通过 `getGroupMemberList` 与群监听中的 onGroupMemberEnter 和 onGroupMemberQuit 回调，在客户端维护当前在线群成员列表。但此方案用户退出直播间重新进入后，也只能获取最新的30位群成员。

Android

iOS&Mac

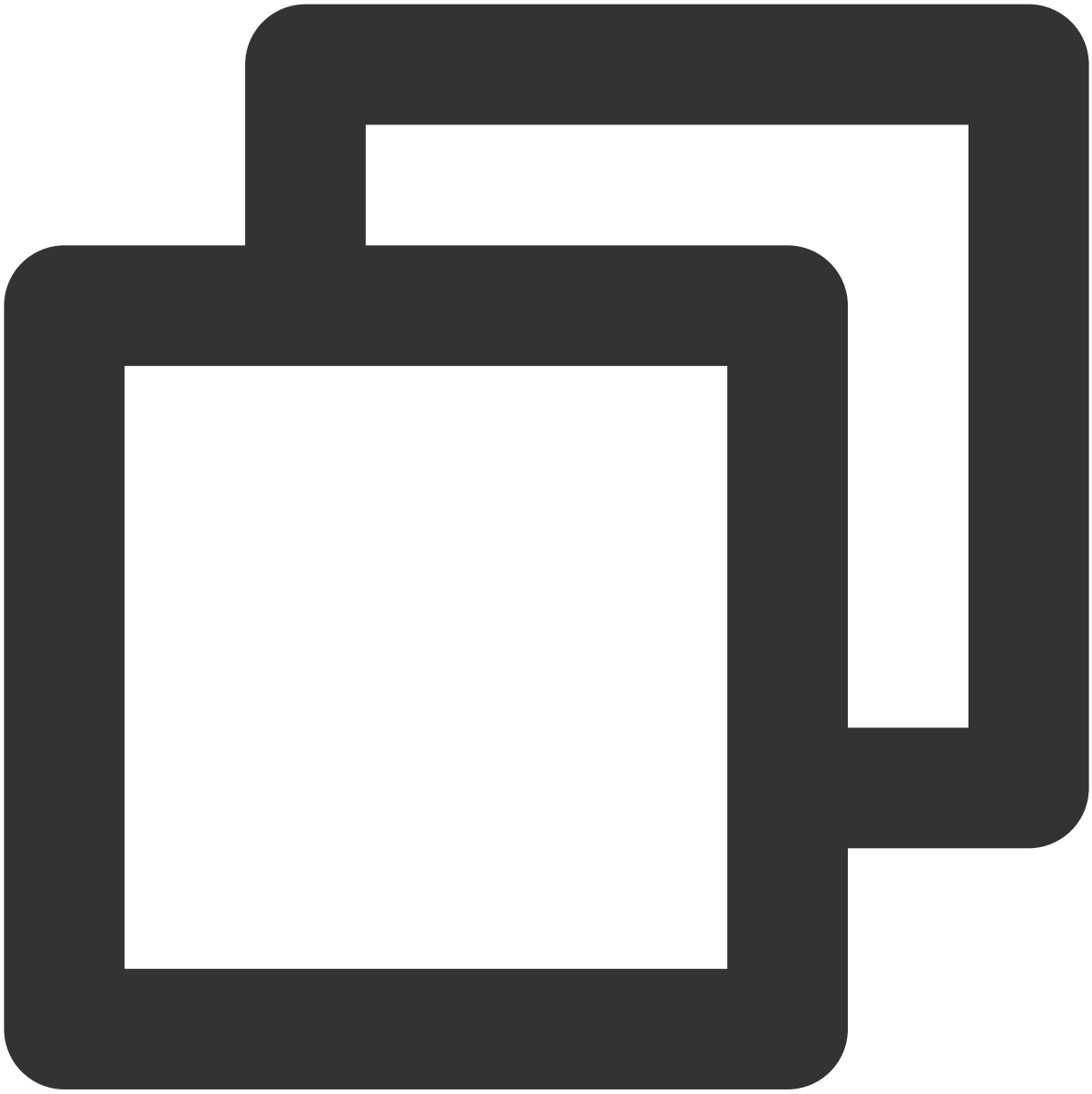
Flutter

Web

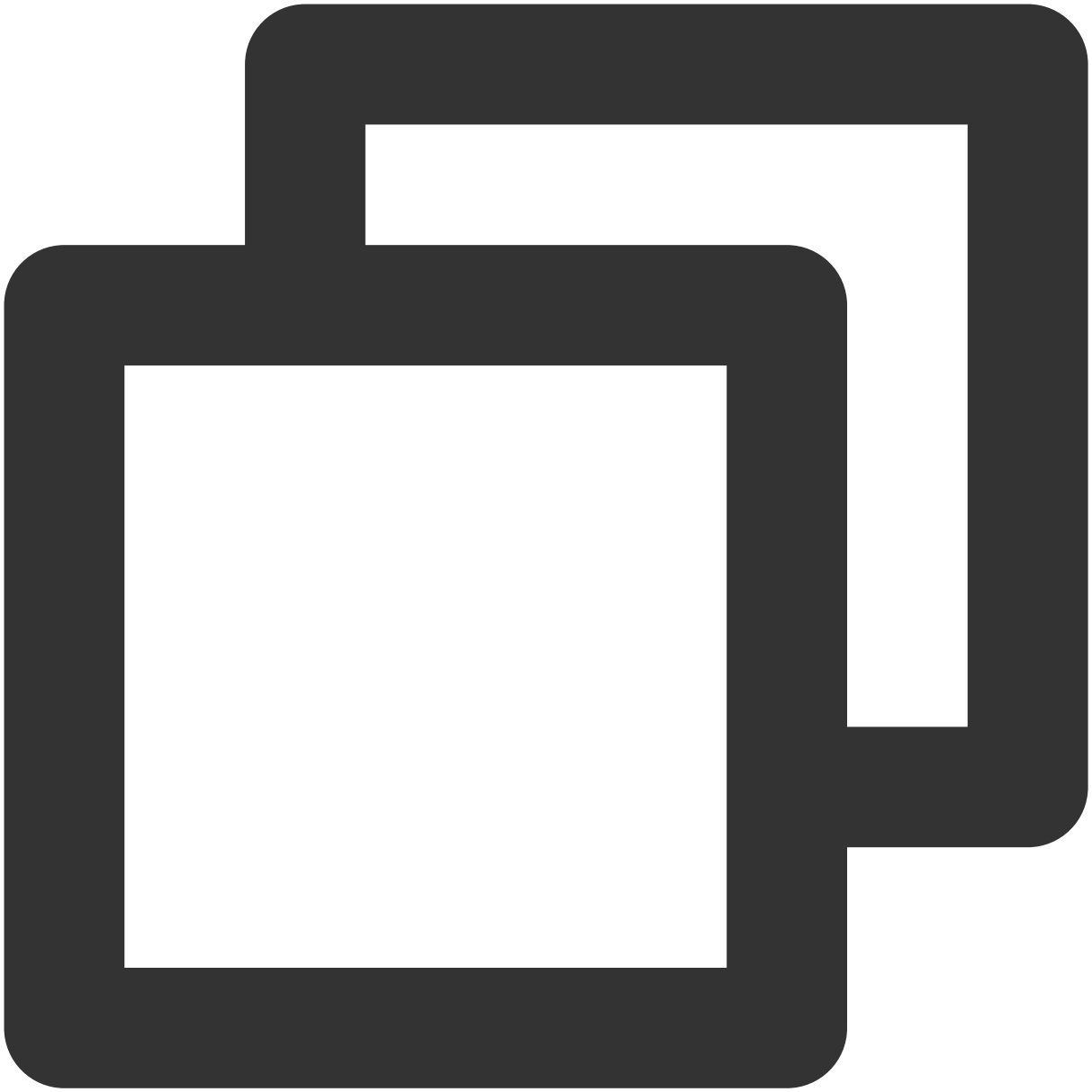


```
// 通过 filter 参数指定只拉取群主的资料
int role = V2TIMGroupMemberFullInfo.V2TIM_GROUP_MEMBER_FILTER_OWNER;
V2TIMManager.getGroupManager().getGroupMemberList("testGroup", role, 0,
    new V2TIMValueCallback<V2TIMGroupMemberInfoResult>() {
    @Override
    public void onError(int code, String desc) {
       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

    @Override
    public void onSuccess(V2TIMGroupMemberInfoResult v2TIMGroupMemberInfoResult) {
```

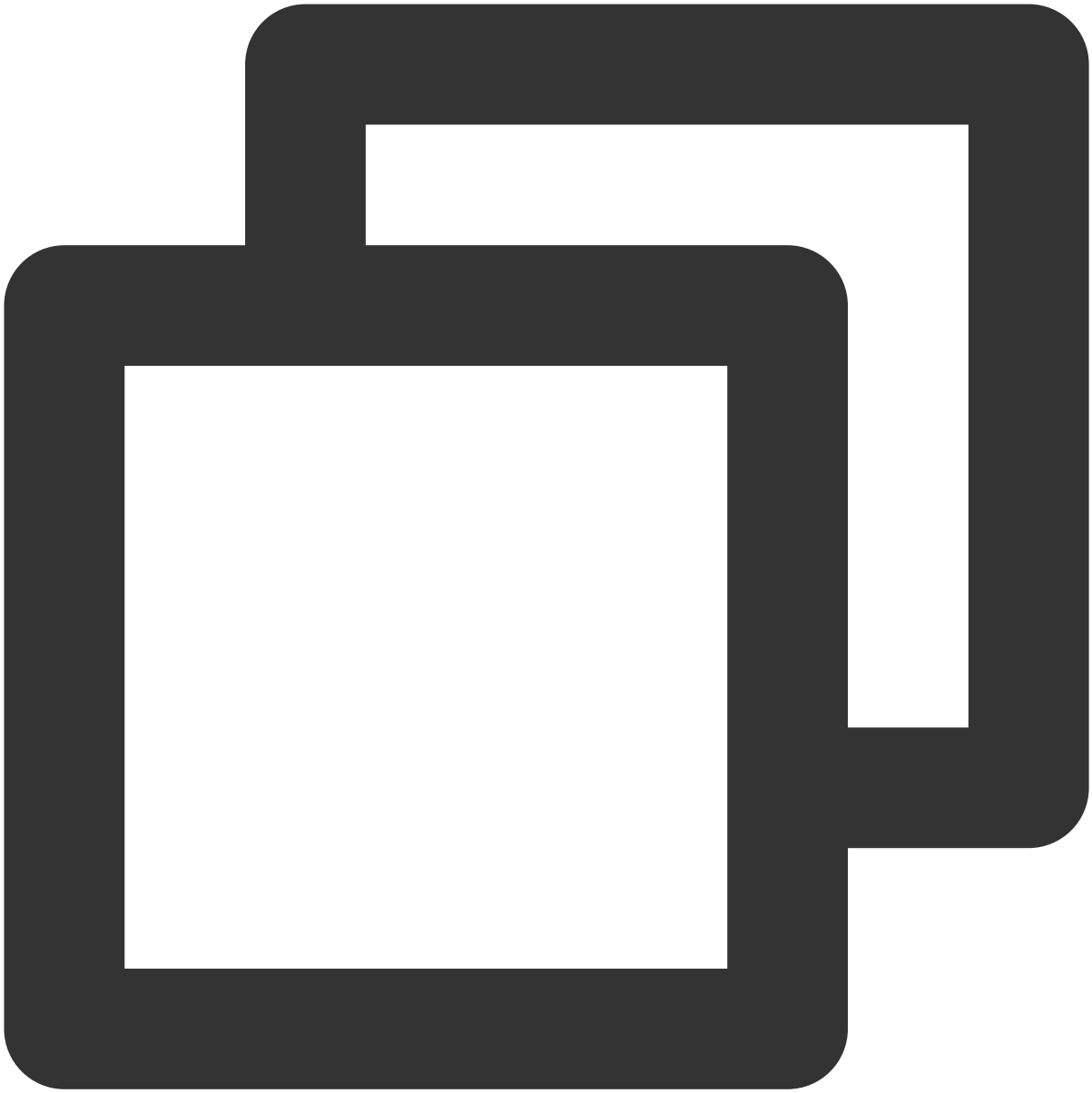
```
        // 拉取成功
    }
});
```



```
[[V2TIMManager sharedInstance] getGroupMemberList:@"groupA" filter:V2TIM_GROUP_MEMB
    // 拉取成功
} fail:^(int code, NSString *desc) {
    // 拉取失败
}];
```



```
// 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_G
```



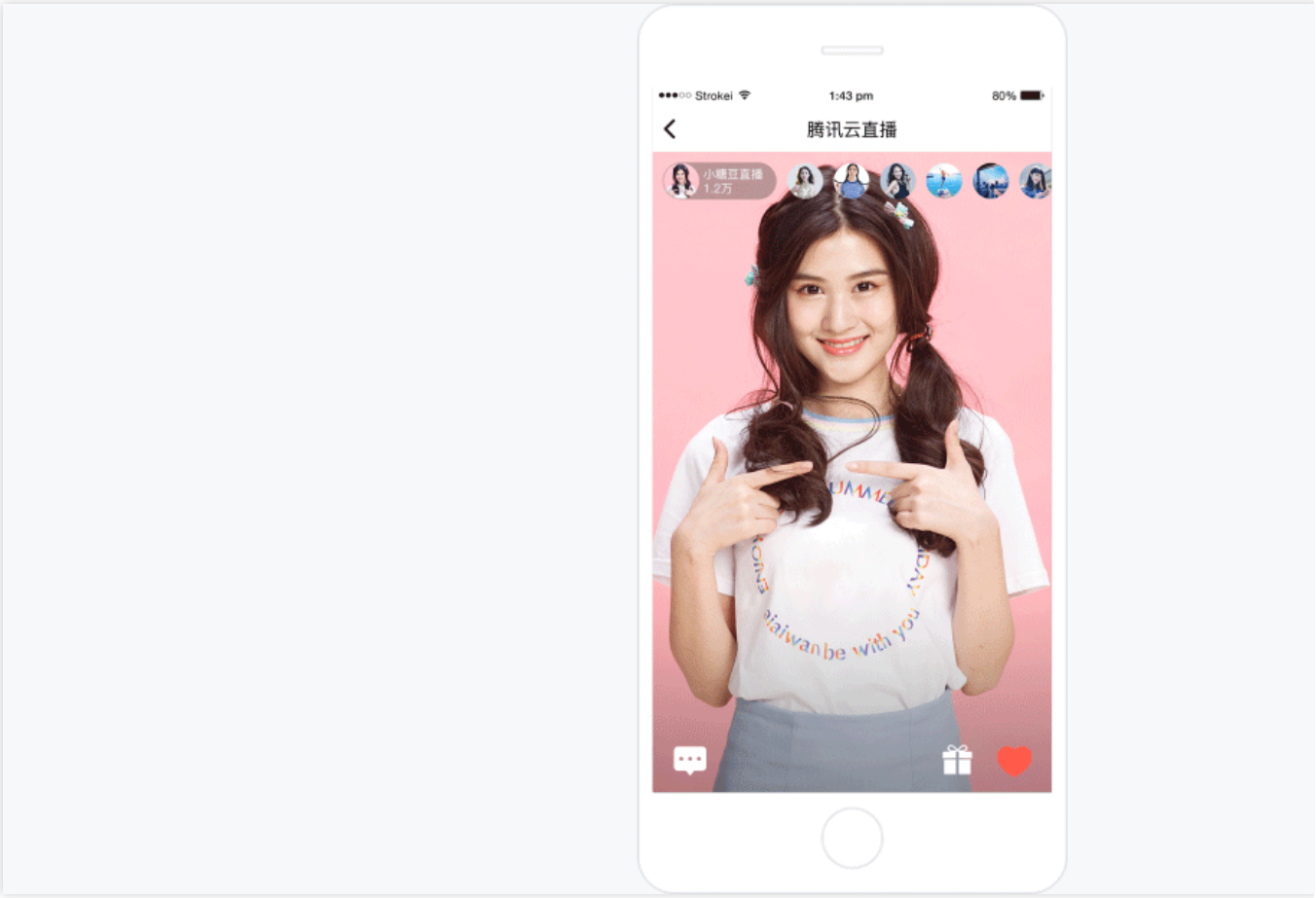
```
tim.getGroupMemberList(options);
```

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进阶版功能，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

若开发者版本非进阶版，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 移动直播 SDK 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

用户端（拉流）

用户端获取直播流的方式也有不同的方式如：

1.1 PC 端可使用 VLC 工具播放

2.App 端可使用 播放器 SDK 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

更多高级功能

云直播除了推拉流业务，还有更多高级功能如：

1.1 极速高清转码，独家极速高清转码技术，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，有效期1年。此时测试产生的流量可以使用其抵扣，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Web 端退出时，导致 Android/iOS/PC 同步退出。
如果是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。

直播聊天室

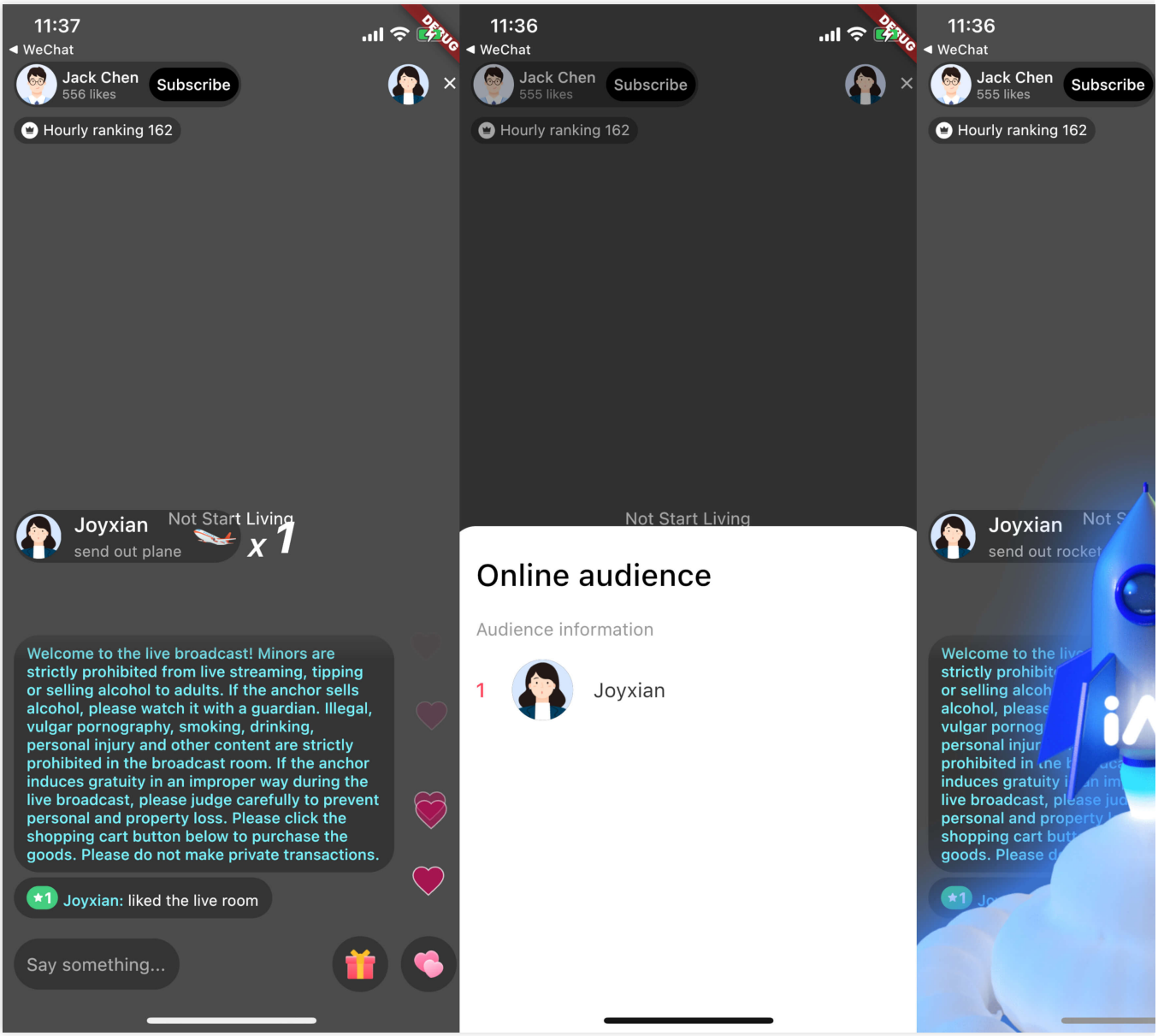
最近更新时间：2024-05-13 17:40:50

。前言

本文以 Tencent Cloud AV Chat Room 插件为基础，结合 直播 Flutter SDK 实现的直播业务场景。

Tencent Cloud AV Chat Room

`Tencent Cloud AV Chat Room` 是基于 Tencent Cloud Chat SDK 并围绕聊天互动场景业务实现的 UI 组件。您可通过该组件快速实现一个包含**千万级互动**的应用。



使用介绍

前置条件

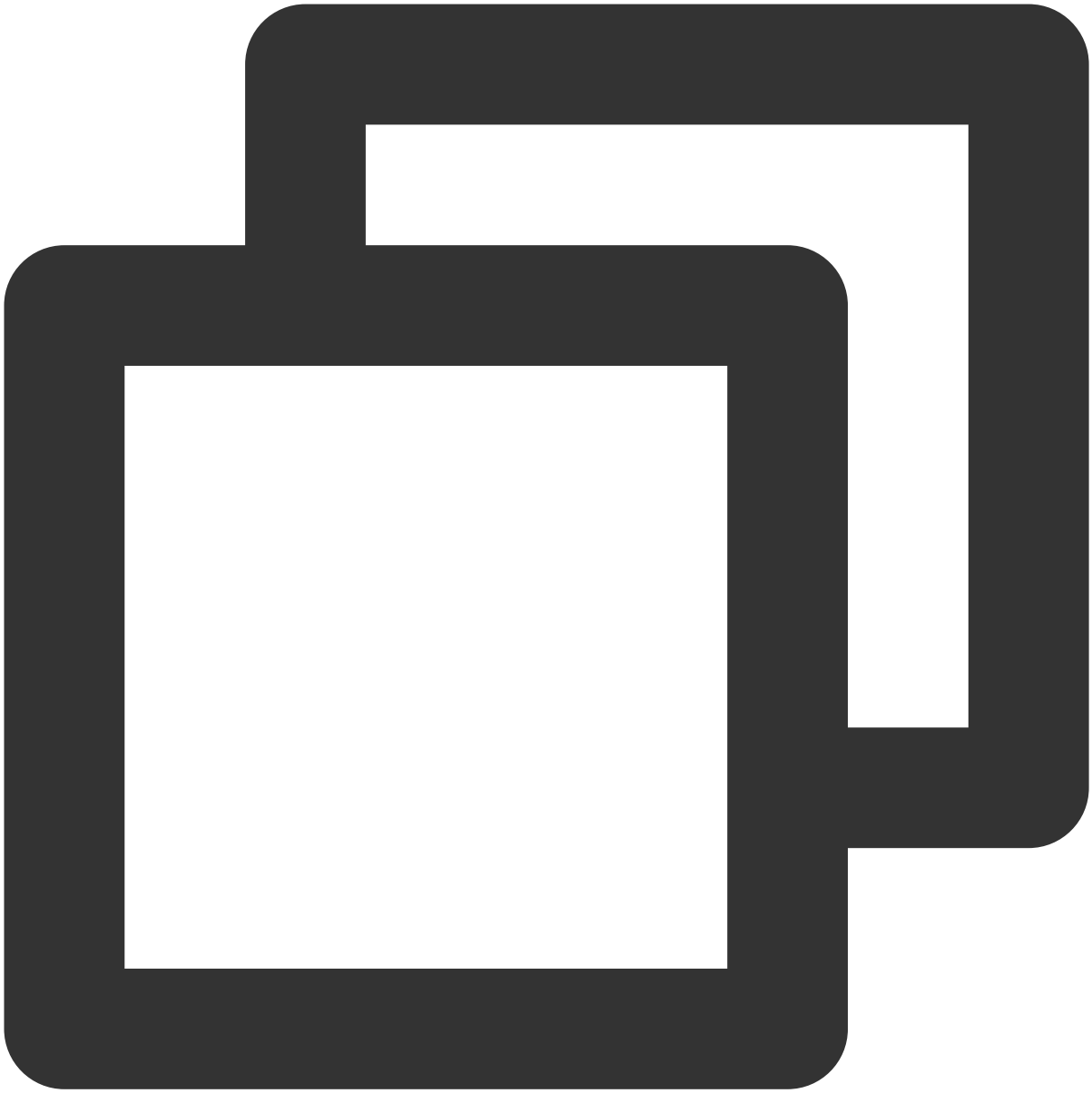
已 注册 腾讯云账号并完成 身份验证。

参照 创建并升级应用 创建应用，并记录好 `SDKAppID` 。

在 IM 控制台 选择您的应用，在左侧导航栏依次点击 辅助工具->UserSig 生成&校验 ，创建 UserID 及其对应的 `UserSig` ，复制 `UserID` 、 `签名（Key）` 、 `UserSig` 这三个，后续登录时会用到。

创建一个 Flutter 应用。

在 pubspec.yaml 文件中的 dependencies 下添加 tencent_cloud_av_chat_room。或者执行如下命令：



```
/// step 1:
flutter pub add tencent_cloud_av_chat_room

/// step 2:
flutter pub get
```

步骤1：初始化并登录 IM

初始化并登录 IM 有两种方式:

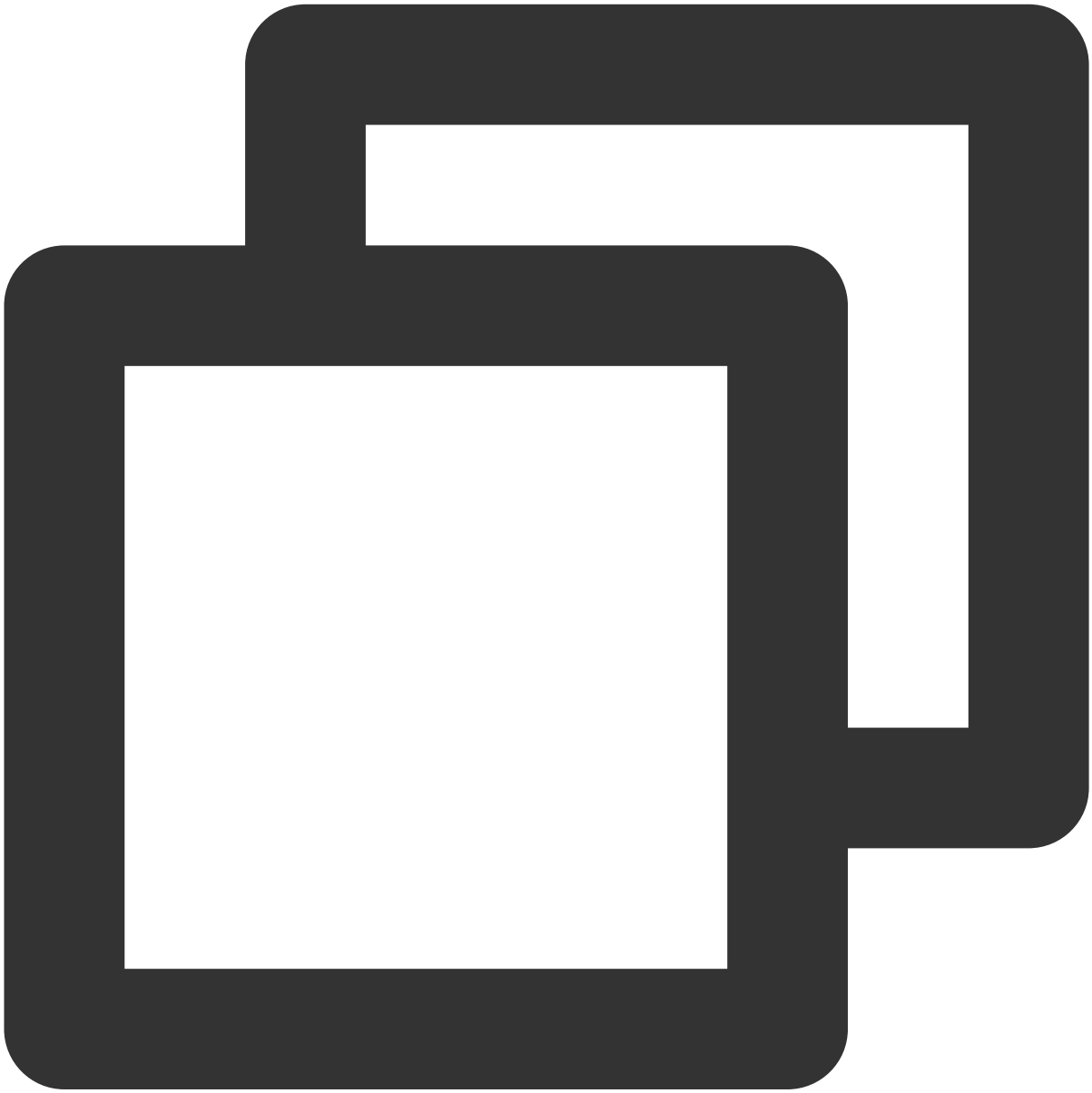
组件外部：整个应用初始化并登录一次即可。

组件内部：通过配置的方式将参数传入组件内部。

如若您在现有 IM flutter 项目中集成该插件，可跳过该步骤。

组件外部(推荐)

在您创建的 flutter 应用中初始化 IM, 注意 IM 应用只需初始化一次即可。如若在现有 IM 项目中集成可跳过该步骤。

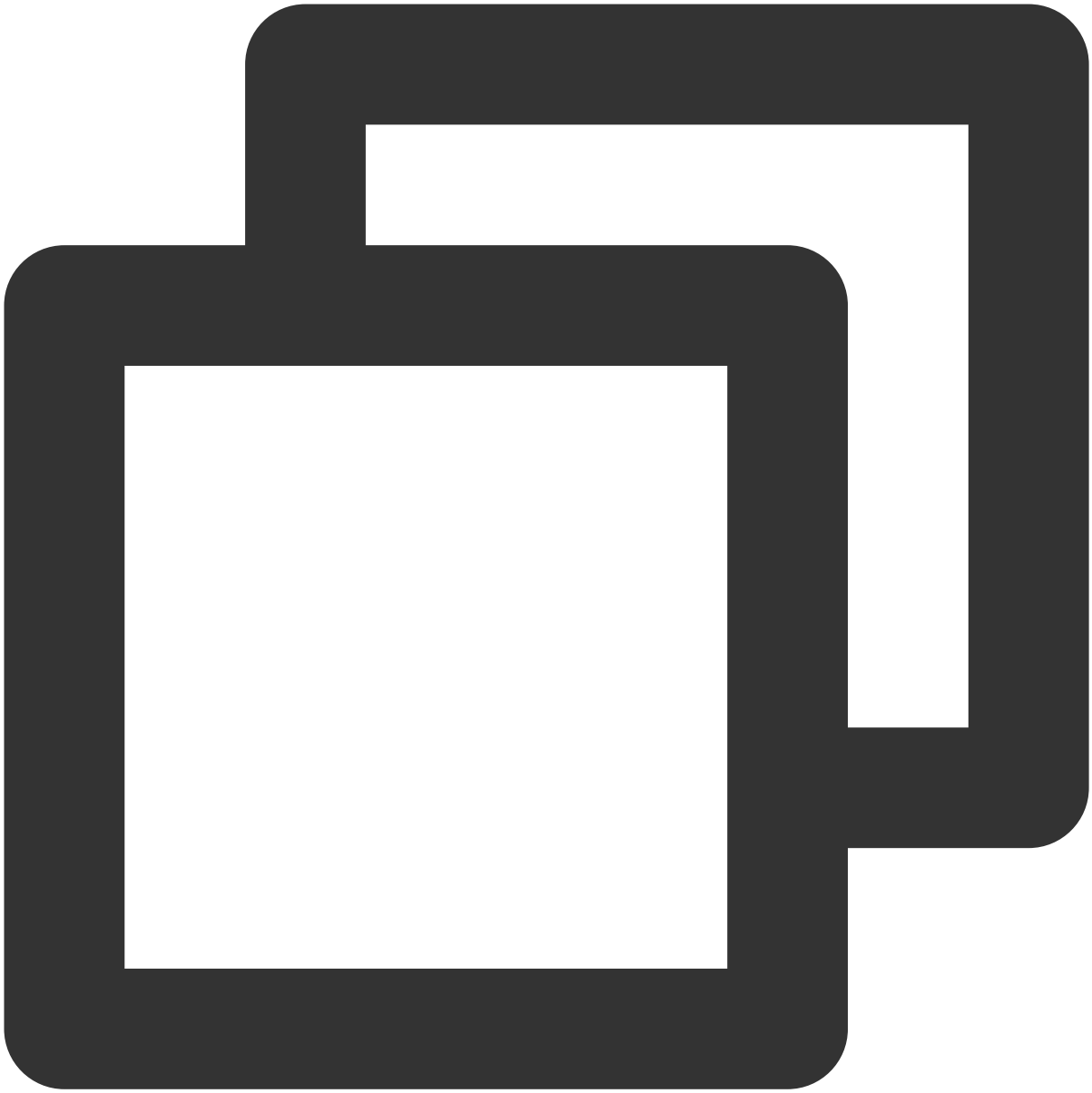


```
import 'package:tencent_av_chat_room_kit/tencent_cloud_chat_sdk_type.dart';

class _MyHomePageState extends State<MyHomePage> {
     final int _sdkAppID = 0; // 前置条件中创建的IM应用SDKAppID
     final String _loginUserID = ""; // 前置条件中的UserID
```

```
    final String _userSig = ""; // 前置条件中的UserSig
  @override
  void initState() {
      super.initState();
      _initAndLoginIm();
    }

  _initAndLoginIm() async {
      await TencentImSDKPlugin.v2TIMManager.initSDK(
          sdkAppID: _sdkAppID,
          loglevel: LogLevelEnum.V2TIM_LOG_ALL,
          listener: V2TimSDKListener());
      TencentImSDKPlugin.v2TIMManager
        .login(userID: _loginUserID, userSig: _userSig);
  }
}
```

组件内部

您也可将 `SDKAppID` 、 `UserSig` 、 `UserID` 通过配置的方式传入组件内部进行 IM 的初始化和登录。

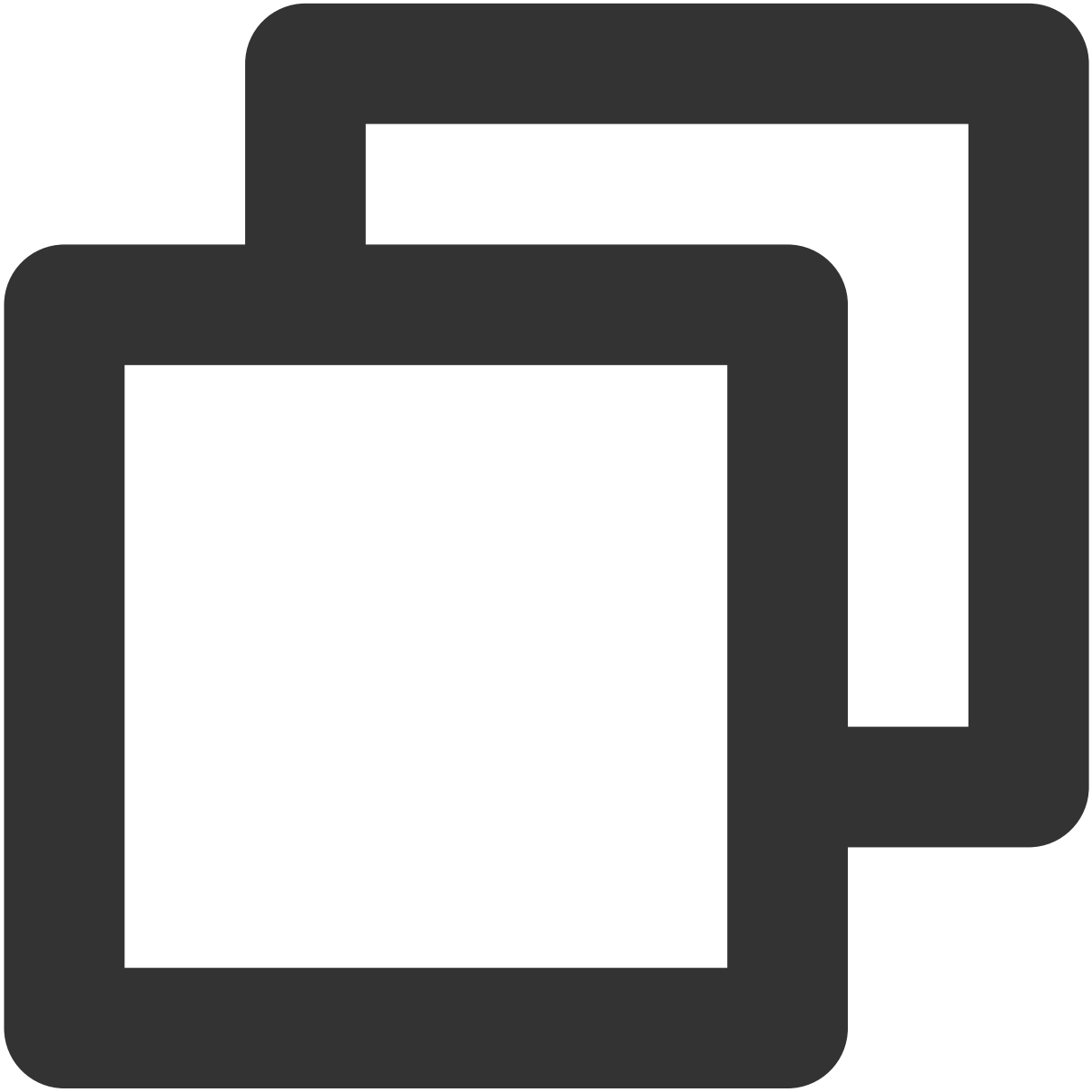


```
import 'package:tencent_cloud_av_chat_room/tencent_cloud_av_chat_room.dart';

class _TencentAVChatRoomKitState extends State<TencentCloudAVChatRoom> {
    final int _sdkAppID = 0; // 前置条件中创建的IM应用SDKAppID
    final String _loginUserID = ""; // 前置条件中的UserID
    final String _userSig = ""; // 前置条件中的UserSig
    @override
      Widget build(BuildContext context) {
        return TencentCloudAVChatRoom(
            config: TencentCloudAvChatRoomConfig(
                loginUserID: _loginUserID, sdkAppID: _sdkAppID, userSig: _userSig))
```

```
    }
}
```

步骤2：使用组件

在您合适的模块中使用该组件。



```
import 'package:tencent_cloud_av_chat_room/tencent_cloud_av_chat_room.dart';

class _TencentAVChatRoomKitState extends State<TencentCloudAVChatRoom> {
    final int _sdkAppID = 0; // 前置条件中创建的IM应用SDKAppID
```

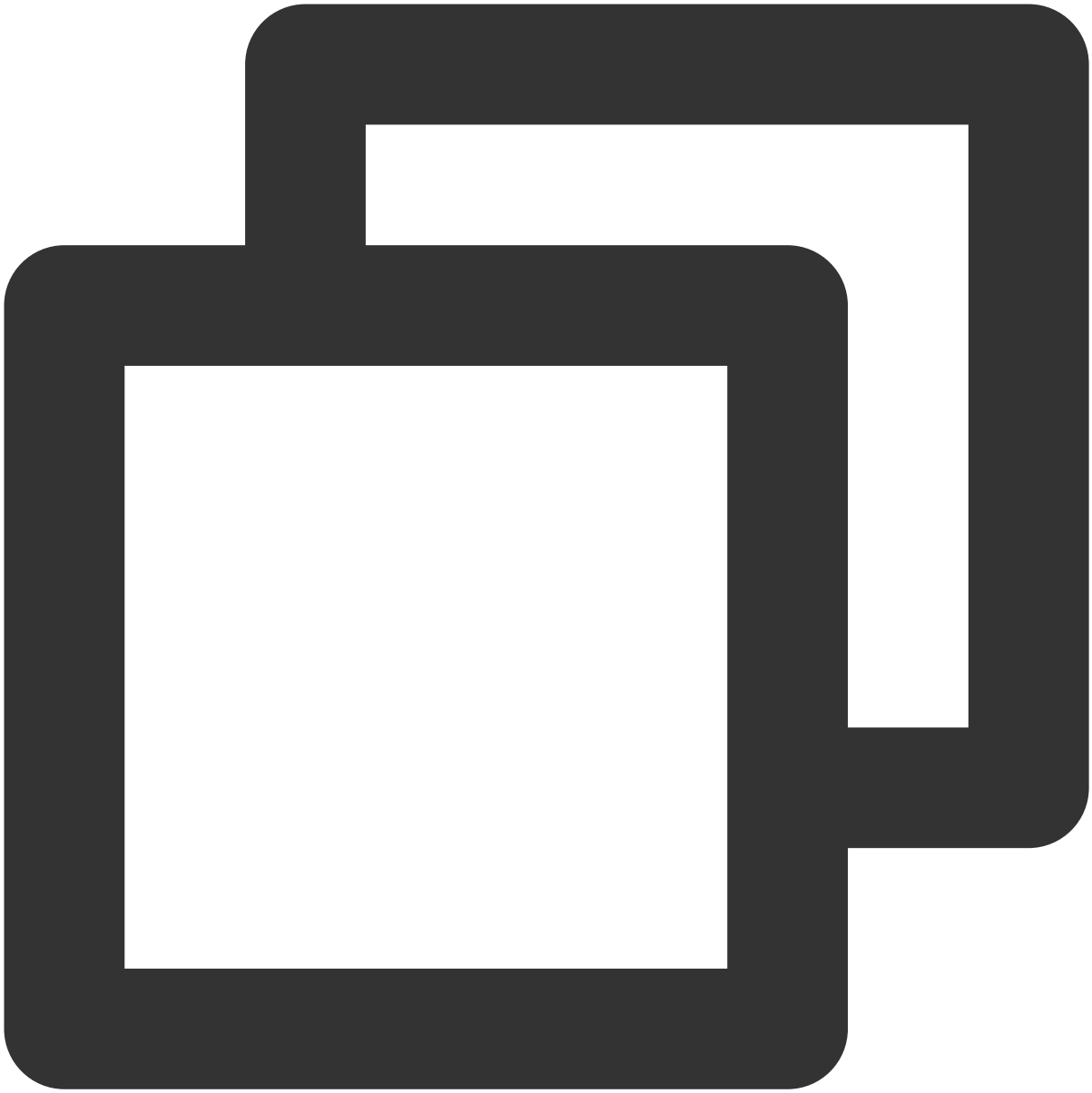
```
    final String _loginUserID = ""; // 前置条件中的UserID
    final String _userSig = ""; // 前置条件中的UserSig
    @override
      Widget build(BuildContext context) {
        return TencentCloudAVChatRoom(
            data: TencentCloudAvChatRoomData(anchorInfo: AnchorInfo()),
            config: TencentCloudAvChatRoomConfig(
                loginUserID: _loginUserID, sdkAppID: _sdkAppID, userSig: _userSig,
    }
}
```

配合腾讯云视立方搭建直播间

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。下面会介绍如何与腾讯云视立方搭建一个直播间。
直播间主要包含直播和互动两部分，直播部分我们会采用腾讯云视立方移动直播来实现直播拉流, 画面播放等。互动采用本插件来实现直播点赞、送礼、弹幕收发等。
本文相关代码均可在 Github 中下载查看。

直播拉流和画面播放

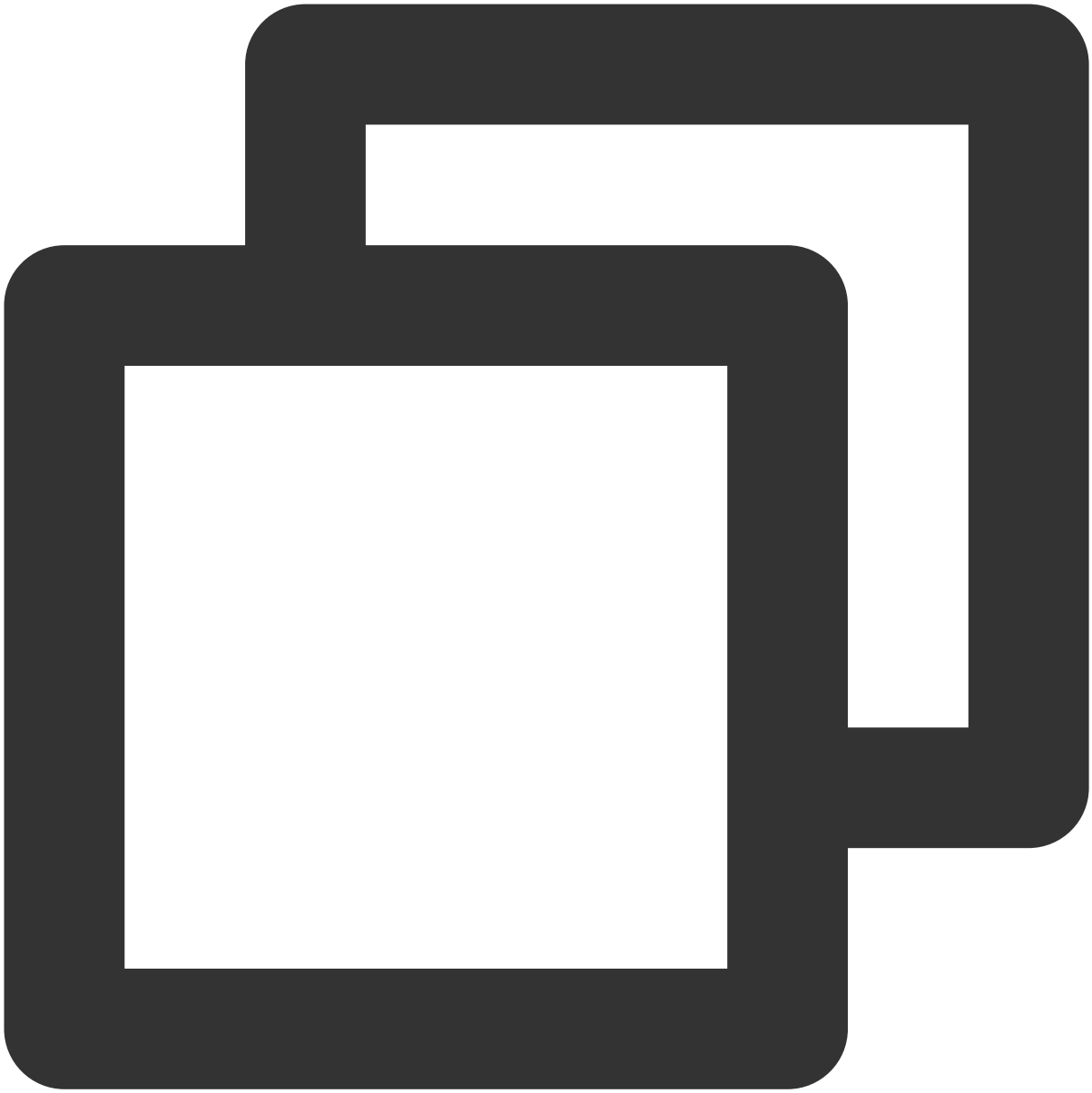
直播 SDK 是腾讯云视立方产品家族的子产品之一。直播 SDK 支持直播推流、拉流、主播观众互动连麦、主播跨房 PK 等能力，为用户提供稳定、极速的直播终端服务。
参见 直播 Flutter SDK 标准直播拉流 实现直播拉流和画面播放功能。
1. **设置 License：**



```
// live.dart
class Live extends StatefulWidget {
  const Live({Key? key}) : super(key: key);

  @override
  State<Live> createState() => _LiveState();
}

class _LiveState extends State<Live> {
    ...
    @override
```

```
    void initState() {
        super.initState();
        setupLicense();
    }

    /// 腾讯云License管理页面(https://console.tencentcloud.com/live/license)
    setupLicense() {
        // 当前应用的License LicenseUrl
        final licenseUrl = "";
        // 当前应用的License Key
        final licenseKey = "";
        V2TXLivePremier.setLicence(licenseUrl, licenseKey);
    }
    ...
}
```

2. **直播拉流、画面播放：**



```
// live_player.dart
...
class _LivePlayState extends State<LivePlayer> {
    ...
    @override
    void initState() {
        super.initState();
        initPlayer();
    }
    initPlayer() async {
        _livePlayer = await V2TXLivePlayer.create();
```

```
    }

    @override
    void dispose() {
        super.dispose();
        _livePlayer.destroy();
    }

    @override
    Widget build(BuildContext context) {
        return Container(
            color: Colors.black.withOpacity(0.7),
            child: V2TXLiveVideoWidget(
                onViewCreated: (viewId) async {
                _localViewId = viewId;
                startPlay();
                },
            ),
        );
    }

    void startPlay() async {
        if (_isPlaying) {
            return;
        }
        if (_localViewId != null) {
            debugPrint("_localViewId $_localViewId");
            var code = await _livePlayer.setRenderViewID(_localViewId!);
            if (code != V2TXLIVE_OK) {
                debugPrint("StartPlay error： please check remoteView load");
            }
        }
        var url = widget.playUrl;
        debugPrint("play url: $url");
        var playStatus = await _livePlayer.startLivePlay(url);
        debugPrint("play status: $playStatus");
        if (playStatus != V2TXLIVE_OK) {
        setState(() {
            _onError = true;
        });
        debugPrint("play error: $playStatus url: $url");
            return;
        }
        await _livePlayer.setPlayoutVolume(100);
        setState(() {
            _isPlaying = true;
        });
```

```
        }
    }
```

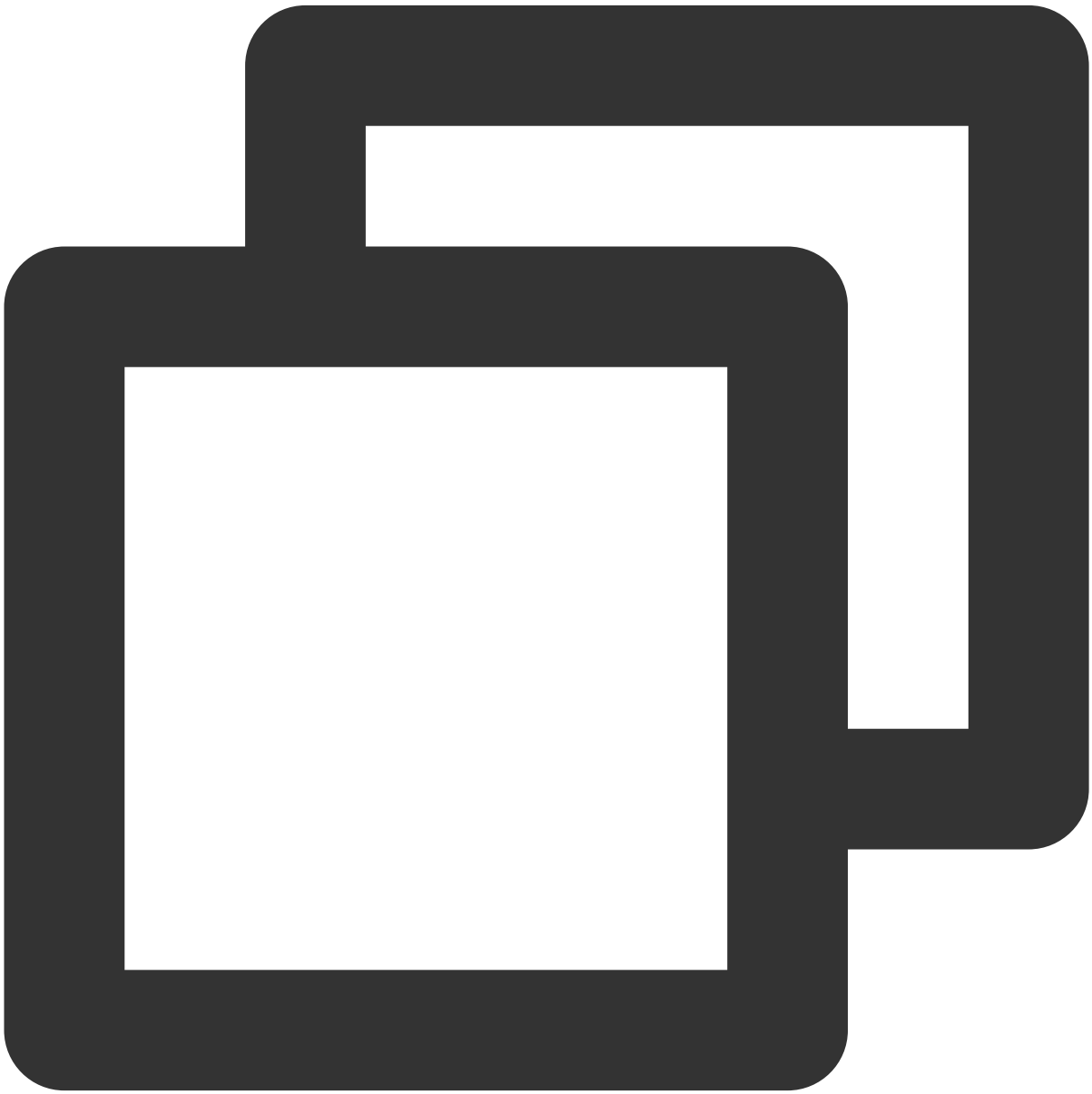
直播互动(点赞、送礼、弹幕收发)

直播互动在直播场景中尤为重要。用户通过弹幕、送礼等方式与主播进行互动。

接下来参见 使用介绍 来集成本插件。

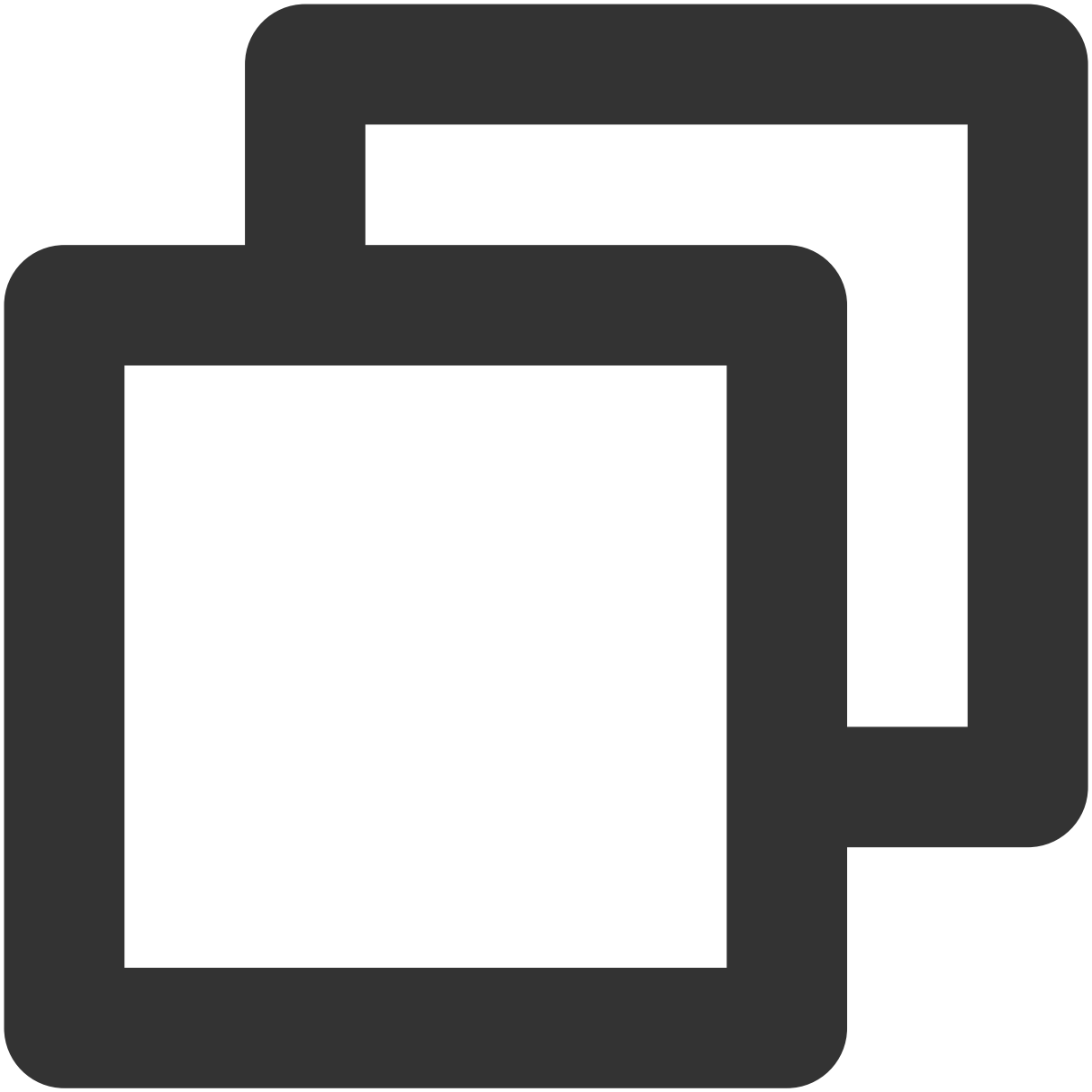
1. **初始化、登录 IM：**

我们在插件外部登录 IM,通常整个应用程序只需要初始化和登录 IM 一次即可。



```
import 'package:tencent_av_chat_room_kit/tencent_cloud_chat_sdk_type.dart';

class _MyHomePageState extends State<MyHomePage> {
     final int _sdkAppID = 0; // 前置条件中创建的IM应用SDKAppID
     final String _loginUserID = ""; // 前置条件中的UserID
     final String _userSig = ""; // 前置条件中的UserSig
    @override
    void initState() {
        super.initState();
        _initAndLoginIm();
      }

    _initAndLoginIm() async {
        await TencentImSDKPlugin.v2TIMManager.initSDK(
            sdkAppID: _sdkAppID,
            loglevel: LogLevelEnum.V2TIM_LOG_ALL,
            listener: V2TimSDKListener());
        TencentImSDKPlugin.v2TIMManager
          .login(userID: _loginUserID, userSig: _userSig);
    }
}
```

2. **使用插件：**



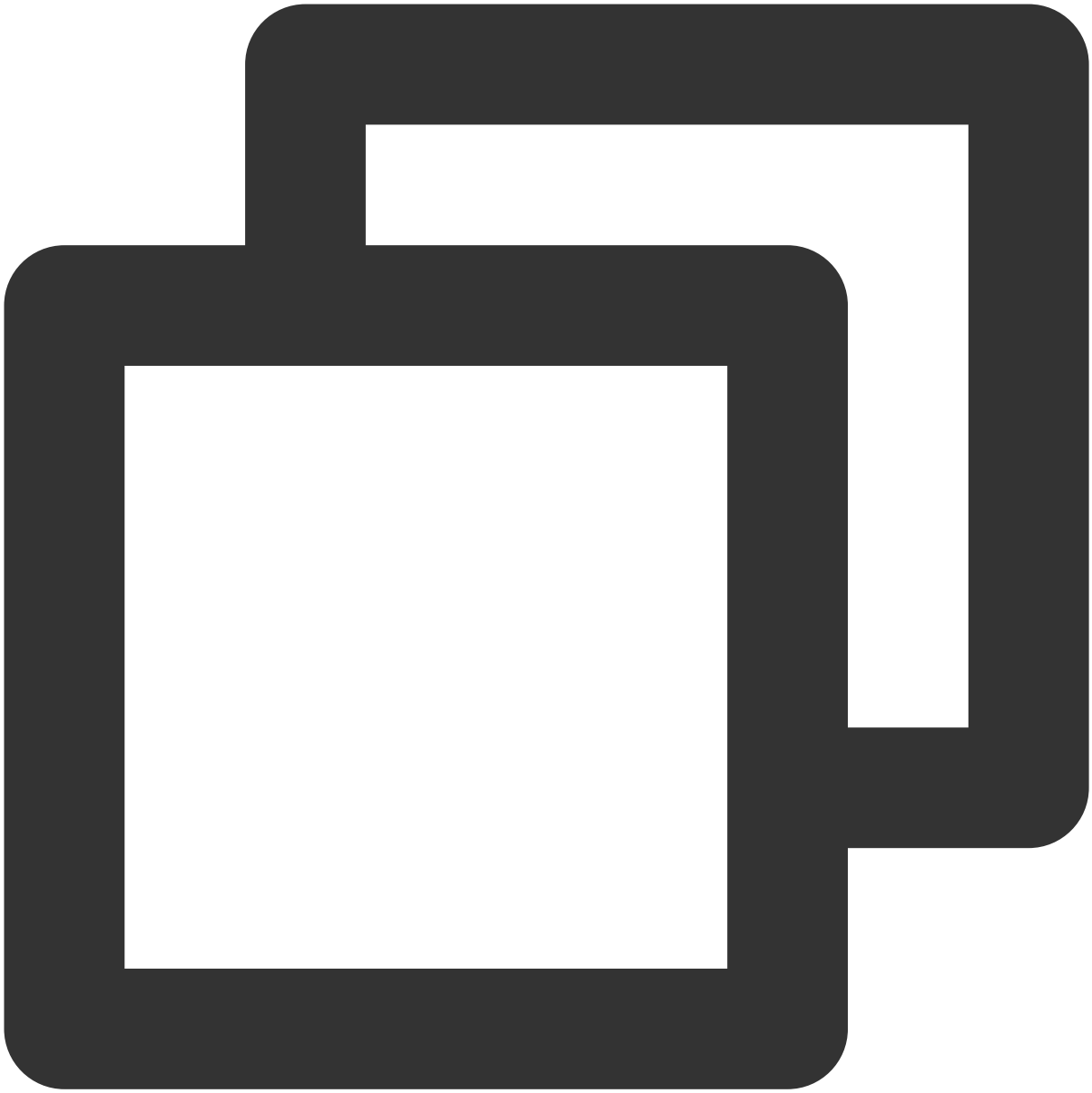
```
// live_room.dart

class LiveRoom extends StatelessWidget {
    final String loginUserID;
    final String playUrl;
    final String avChatRoomID = ''; // 一个直播间本质上是所有用户在一个av chat room 类型的

    LiveRoom({Key? key, required this.loginUserID, required this.playUrl})
      : super(key: key);

    @override
```

```
    Widget build(BuildContext context) {
        return Stack(
            children: [
                LivePlayer(playUrl: playUrl), // 直播拉流和画面播放
                TencentCloudAVChatRoom(
                    config: TencentCloudAvChatRoomConfig(
                        avChatRoomID: avChatRoomID,
                        loginUserID: loginUserID, //登录的用户ID
                    ),
                    data: TencentCloudAvChatRoomData(
                        isSubscribe: false,
                        notification: "直播间公告",
                        anchorInfo: AnchorInfo(
                            subscribeNum: 200,
                            fansNum: 5768,
                            nickName: "风雨人生",
                            avatarUrl:""
                        ),
                    )
                )
            ]
        )
    }
}
```

上面我们通过 `Stack Widget` 将 `LivePlayer(直播)` 和 `LiveRoom(互动)` 通过层叠的方式结合在一起。

3. 如何自定义：

插件本身提供默认的 UI，但是往往在实际的使用过程中，用户都有不同的 UI, 接下来会介绍如何自定义本插件。



```
...
@override
    Widget build(BuildContext context) {
        return Stack(
            children: [
                LivePlayer(playUrl: playUrl), // 直播拉流和画面播放
                TencentCloudAVChatRoom(
                    ...
                    TencentCloudAvChatRoomCustomWidgets(
                        roomHeaderAction: Container(), // 屏幕右上角显示区域自定义组件,
                        roomHeaderLeading: Container(), // 屏幕左上角显示区域自定义组件,
```

```
                        roomHeaderTag: Container(), // roomHeaderAction 和 roomHeade
                        onlineMemberListPanelBuilder: (context, id) { // 直播间在线人
                            return Container();
                        },
                        anchorInfoPanelBuilder: (context, id) { // 主播头像点击后展开的
                            return Container();
                        },
                        giftsPanelBuilder: (context) { // 屏幕右下方礼物按钮点击后展示的
                            return Container();
                        },
                        messageItemBuilder: (context, message, child) { // 弹幕消息自
                            return Container();
                        },
                        messageItemPrefixBuilder: (context, message) { // 弹幕消息前缀
                            return Container();
                        },
                        giftMessageBuilder: (context, message) { // 礼物消息自定义，从
                            return Container();
                        },
                        textFieldActionBuilder: ( // 屏幕右下方区域自定义
                            context,
                        ) {
                            return [Container()];
                        },
                        textFieldDecoratorBuilder: (context) { // 屏幕左下方输入框自定
                            return Container();
                        }
                    )
                )
            ]
        )
    }
```

4. **礼物：**

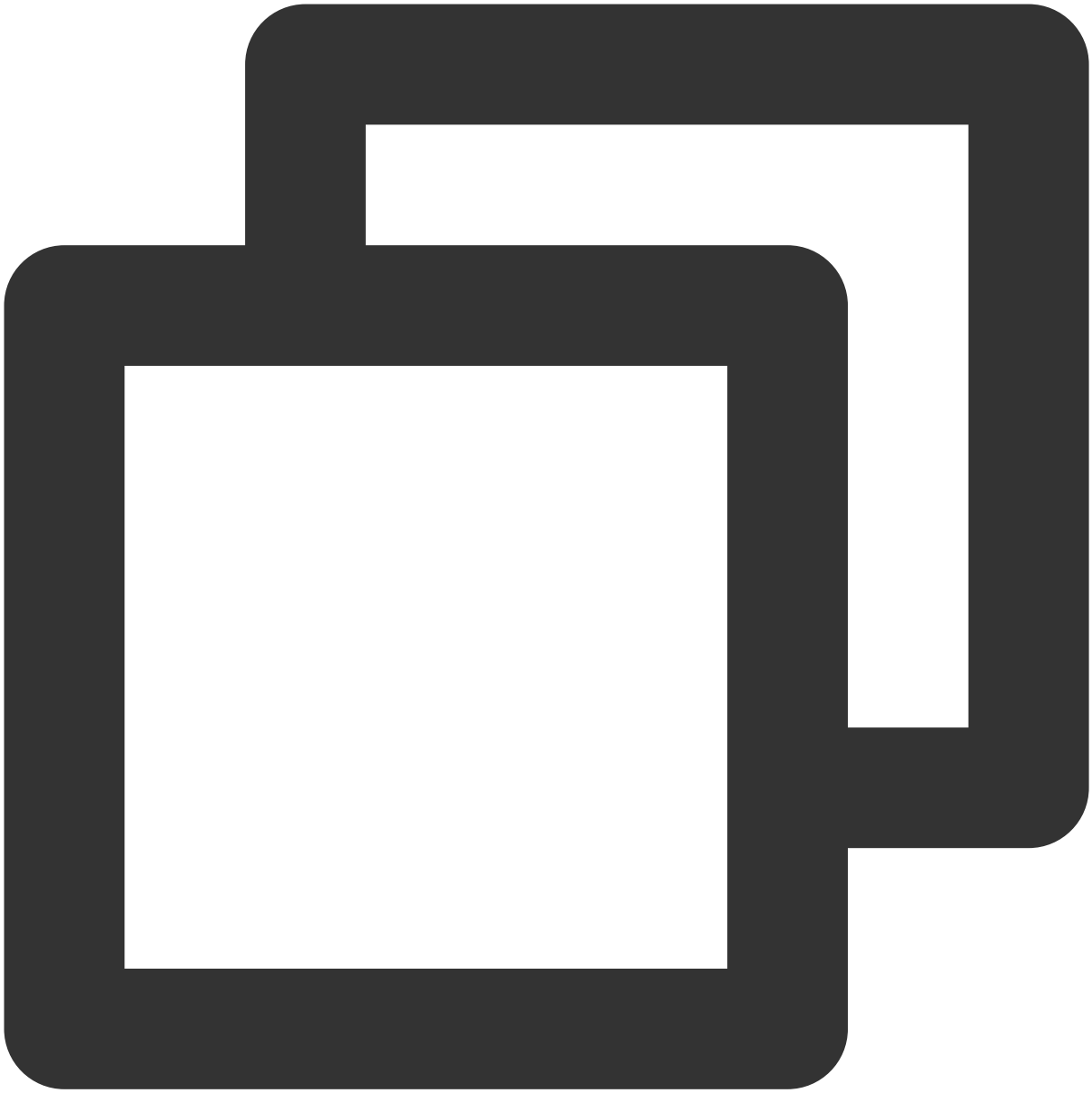
送礼在直播场景中是非常重要的场景。本插件默认提供了三种礼物类型的解析，用户只需要按照特定的格式来发送自定义消息即可在界面上展示礼物消息。本插件默认带的送礼面板只用于展示礼物消息解析的功能，通常礼物是会牵扯到用户计费计量的逻辑，所以需要用户根据自己的业务需求在服务端计费后调用服务端接口发送礼物消息。送礼流程如下:

客户端短连接请求到自己的业务服务器，涉及到计费逻辑。

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）

计费结算后，调用服务端接口发送发送自定义消息（礼物）

礼物消息我们通过发送自定义消息来实现。如下定义了三种礼物格式:



```
// 礼物带特效(会在屏幕左侧滑入送礼详情并会在屏幕上展示礼物特效(Lottie/SVAGA))
final customInfoRocket = {
    "version": 1.0, // 协议版本号
    "businessID": "flutter_live_kit", // 业务标识字段
    "data": {
        "cmd":
            "send_gift_message", // 必须为send_gift_message
        "cmdInfo": {
        "type": 3, // 礼物类型
        "giftUrl": "", // 礼物图片地址
        "giftCount": 1, // 礼物数量
```

```
          "giftSEUrl": "assets/live/rocket.json", // 礼物特效地址，如果是http 开头会加载网
          "giftName": "超级火箭", // 礼物名称
          },
      }
};

// 礼物不带特效(会在屏幕左测滑入)
final customInfoPlane = {
    "version": 1.0, // 协议版本号
    "businessID": "flutter_live_kit", // 业务标识字段
    "data": {
      "cmd":
          "send_gift_message", // 必须为send_gift_message
      "cmdInfo": {
        "type": 2, // 礼物类型
        "giftUrl": "", // 礼物图片地址
        "giftCount": 1, // 礼物数量
        "giftName": "飞机", // 礼物名称
      },
    }
};

// 普通礼物(礼物会展示到弹幕中，不会从屏幕左测滑入和展示特效)
final normalGift = {
    "version": 1.0, // 协议版本号
    "businessID": "flutter_live_kit", // 业务标识字段
    "data": {
      "cmd":
          "send_gift_message", // 必须为send_gift_message
      "cmdInfo": {
        "type": 1, //普通礼物
        "giftUrl": "", // 礼物图片地址
        "giftCount": 1, // 礼物数量
        "giftName": "花", // 礼物名称
        "giftUnits": "朵", // 礼物单位
      },
    }
};
```

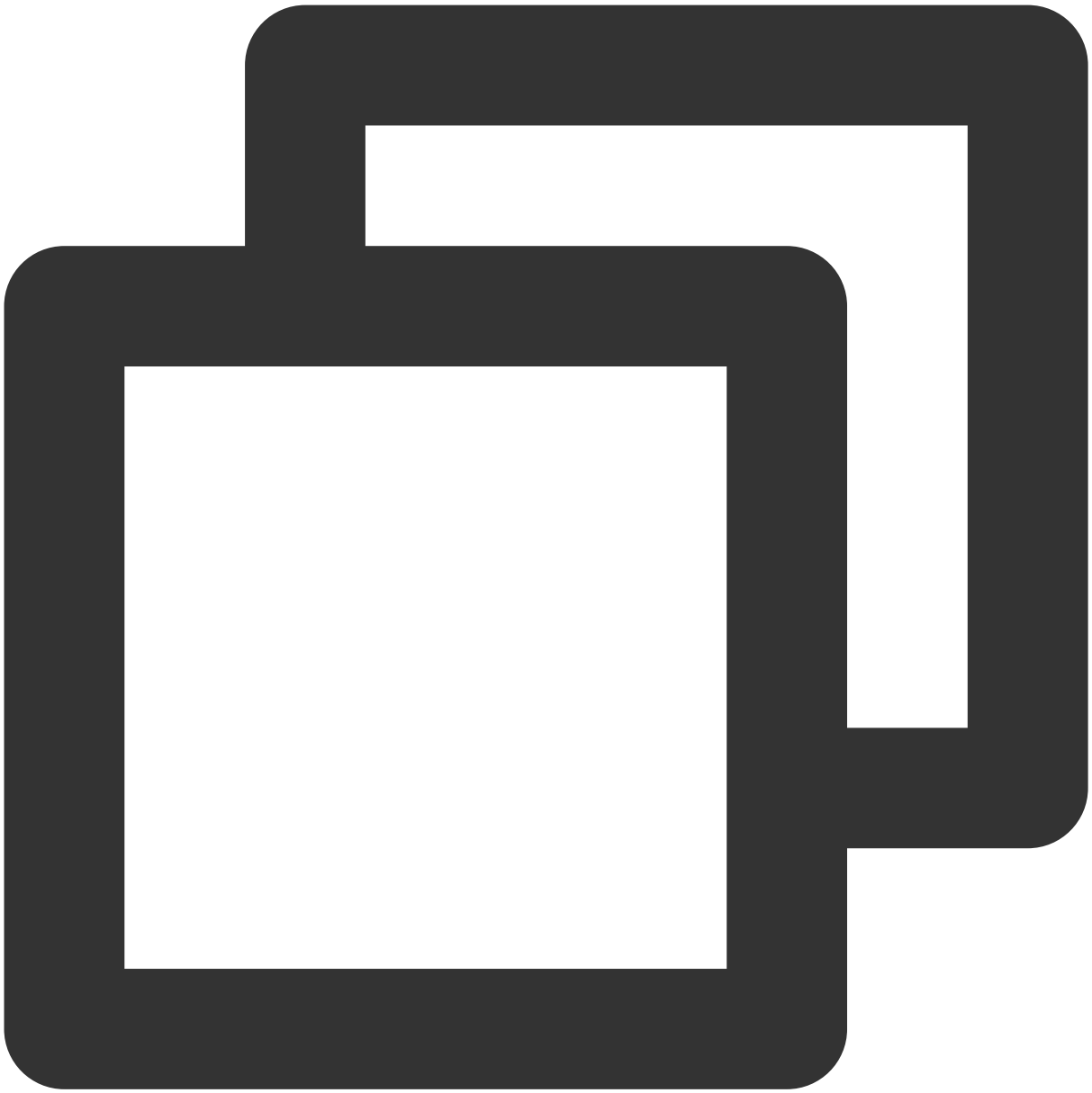
5. **主题：**

本插件除了自定义外同时提供了主题能力，主题分为颜色和字体两部分。可按照您的需求自定义 主题。

API Docs

TencentCloudAvChatRoomData

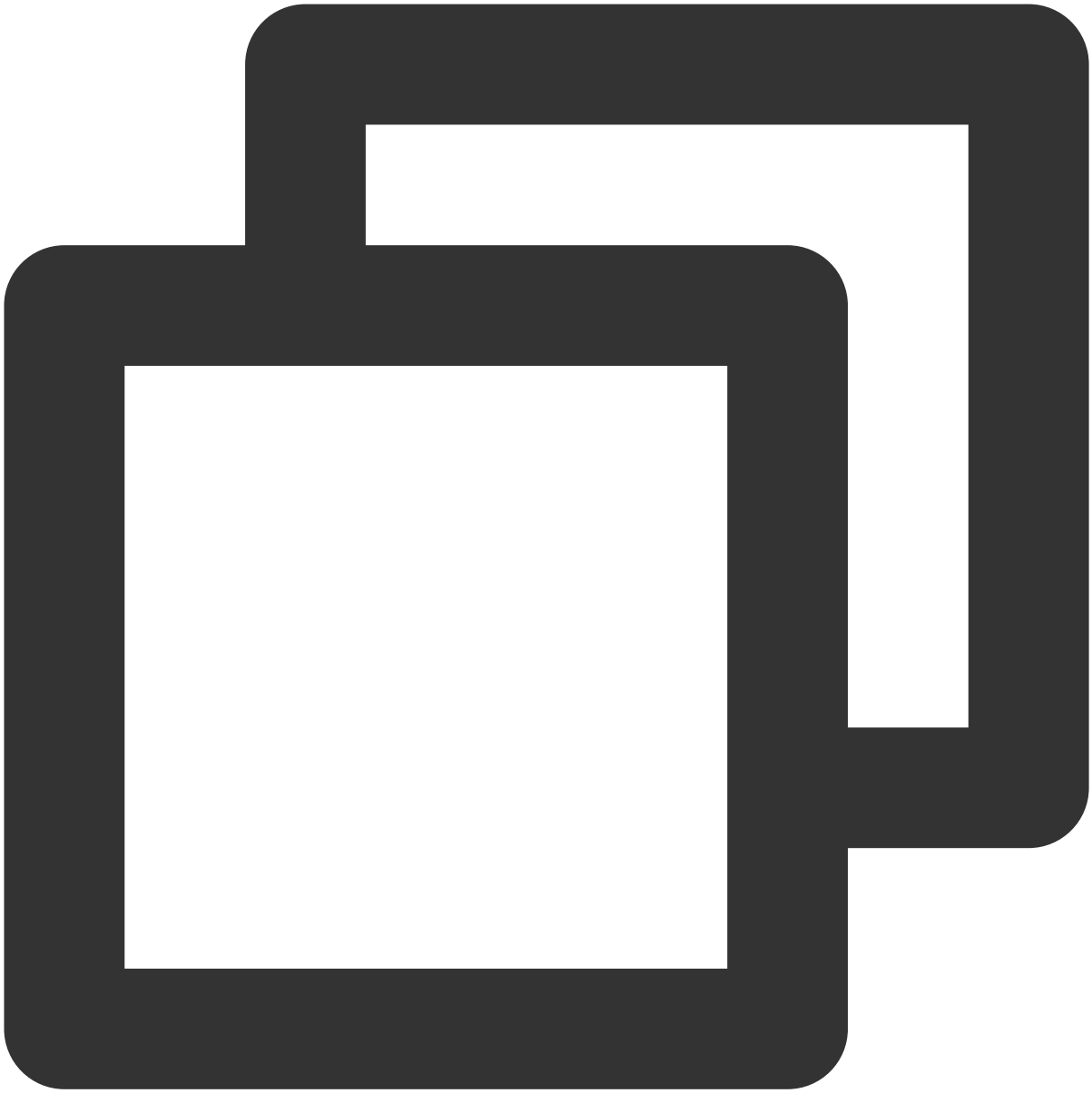
组件需要使用到的数据，例如主播信息、直播间公告等：



```
TencentCloudAvChatRoomData(
    anchorInfo: AnchorInfo(), // 主播信息
    isSubscribe: false, // 是否订阅
    notification: "直播间公告" // 直播间公告
)
```

TencentCloudAvChatRoomConfig

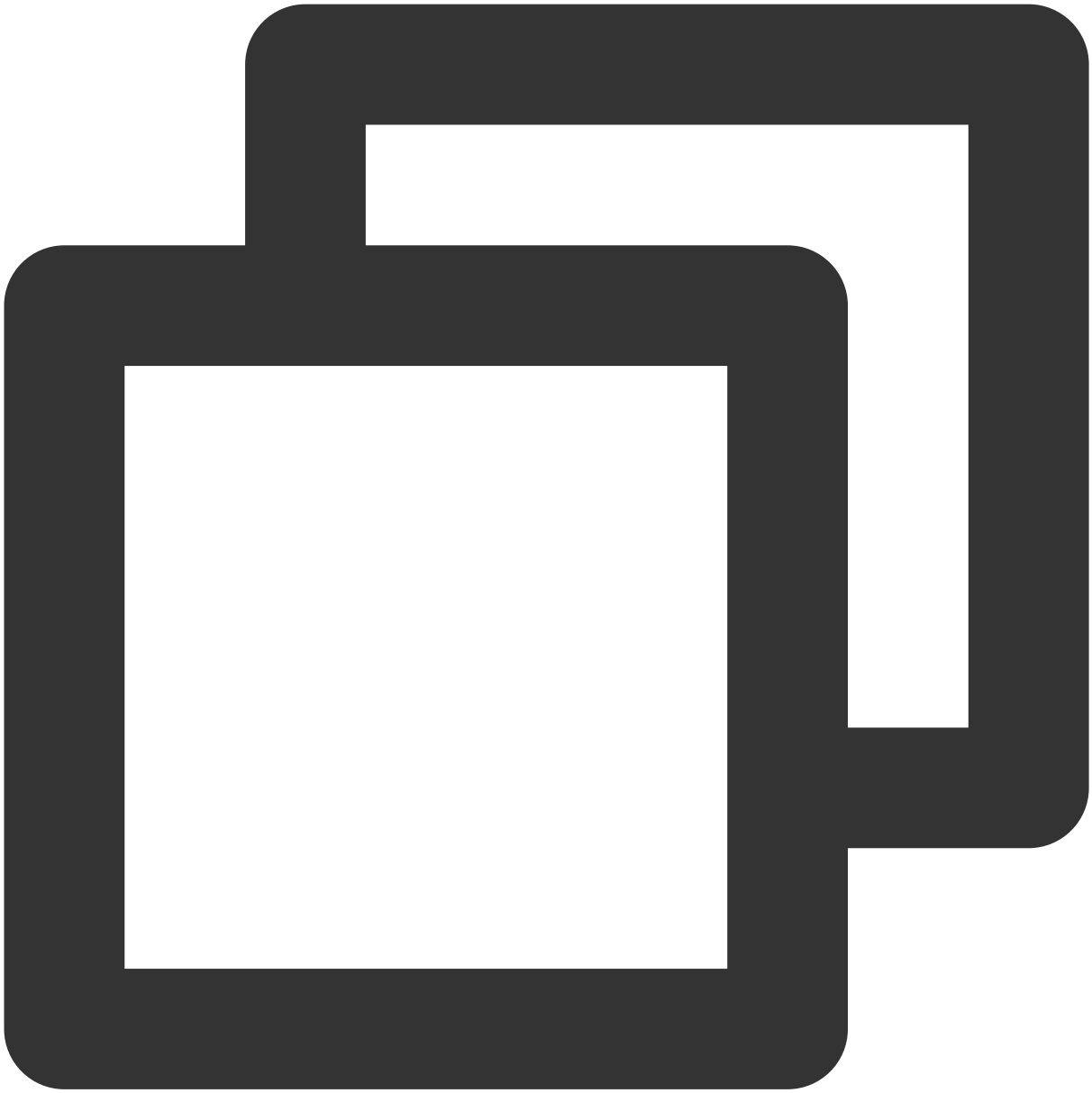
组件配置信息：



```
TencentCloudAvChatRoomConfig(
    avChatRoomID: '', // AV Chat Group. AV Chat Room类型的群ID.[https://cloud.tencent
    loginUserID: '', // 登录用户ID
    sdkAppID: 0, // IM 应用ID
    userSig: '', // 用户ID和secrectKey生成的sig
    barrageMaxCount: 200, // 弹幕最大条数。默认200条，当超出条数后，早些的弹幕会被清除。
    giftHttpBase: '', // 礼物消息http base。
    displayConfig: DisplayConfig() // 可控制界面部分组件的展示与隐藏
)
```

TencentCloudAvChatRoomController

组件控制器，可在组件外部调用，用于更新数据、发送消息等：



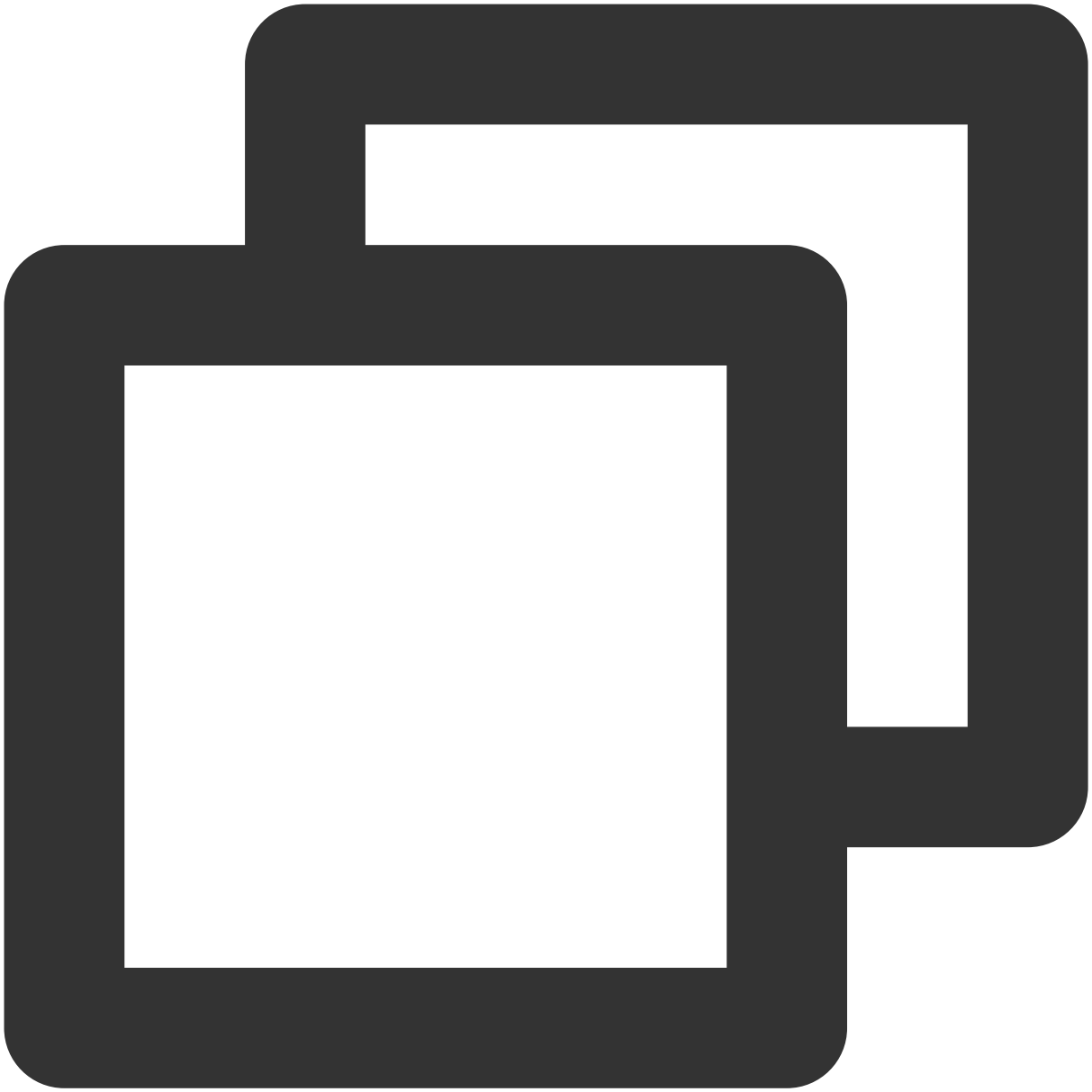
```
final controller = TencentCloudAvChatRoomController();
final _needUpdateData = TencentCloudAvChatRoomData(
        anchorInfo: AnchorInfo(), // 主播信息
        isSubscribe: false, // 是否订阅
        notification: "直播间公告" // 直播间公告
    );
final _textString = "我是一条文本消息";
```

```
final customInfoRocket = {
    "version": 1.0, // 协议版本号
    "businessID": "flutter_live_kit", // 业务标识字段
    "data": {
        "cmd":
            "send_gift_message", // 指令
        "cmdInfo": { // 指令携带的信息
        "type": 3, // 礼物类型
        "giftUrl": "1e8913f8c6d804972887fc179fa1fbd7.png", // 礼物图片地址
        "giftCount": 1, // 礼物数量
        "giftSEUrl": "assets/live/rocket.json", // 礼物特效地址
        "giftName": "超级火箭", // 礼物名称
        },
    }
};

final customInfoPlane = {
    "version": 1.0,
    "businessID": "flutter_live_kit",
    "data": {
      "cmd":
          "send_gift_message",
      "cmdInfo": {
        "type": 2,
        "giftUrl": "5e175b792cd652016aa87327b278402b.png",
        "giftCount": 1,
        "giftName": "飞机",
      },
    }
};

final customInfoFlower = {
    "version": 1.0,
    "businessID": "flutter_live_kit",
    "data": {
      "cmd":
          "send_gift_message",
      "cmdInfo": {
        "type": 1,
        "giftUrl": "8f25a2cdeae92538b1e0e8a04f86841a.png",
        "giftCount": 1,
        "giftName": "花",
        "giftUnits": "朵",
      },
    }
};
```

```
// 更新传入组件的data信息。
controller.updateData(_needUpdateData);

// 发送文本消息
controller.sendTextMessage(_textString);

// 发送礼物消息，礼物消息需要按照如上特定的格式。礼物分为三种类型：[1]：普通礼物 [2]：礼物不带特
controller.sendGiftMessage(jsonEncode(customInfoFlower));

// 发送任何类型的消息，[message] 需要自己创建
controller.sendMessage(message);

// 播放特效动画(Lottie, SVGA).
controller.playAnimation("assets/live/rocket.json");
```

TencentCloudAvChatRoomCallback

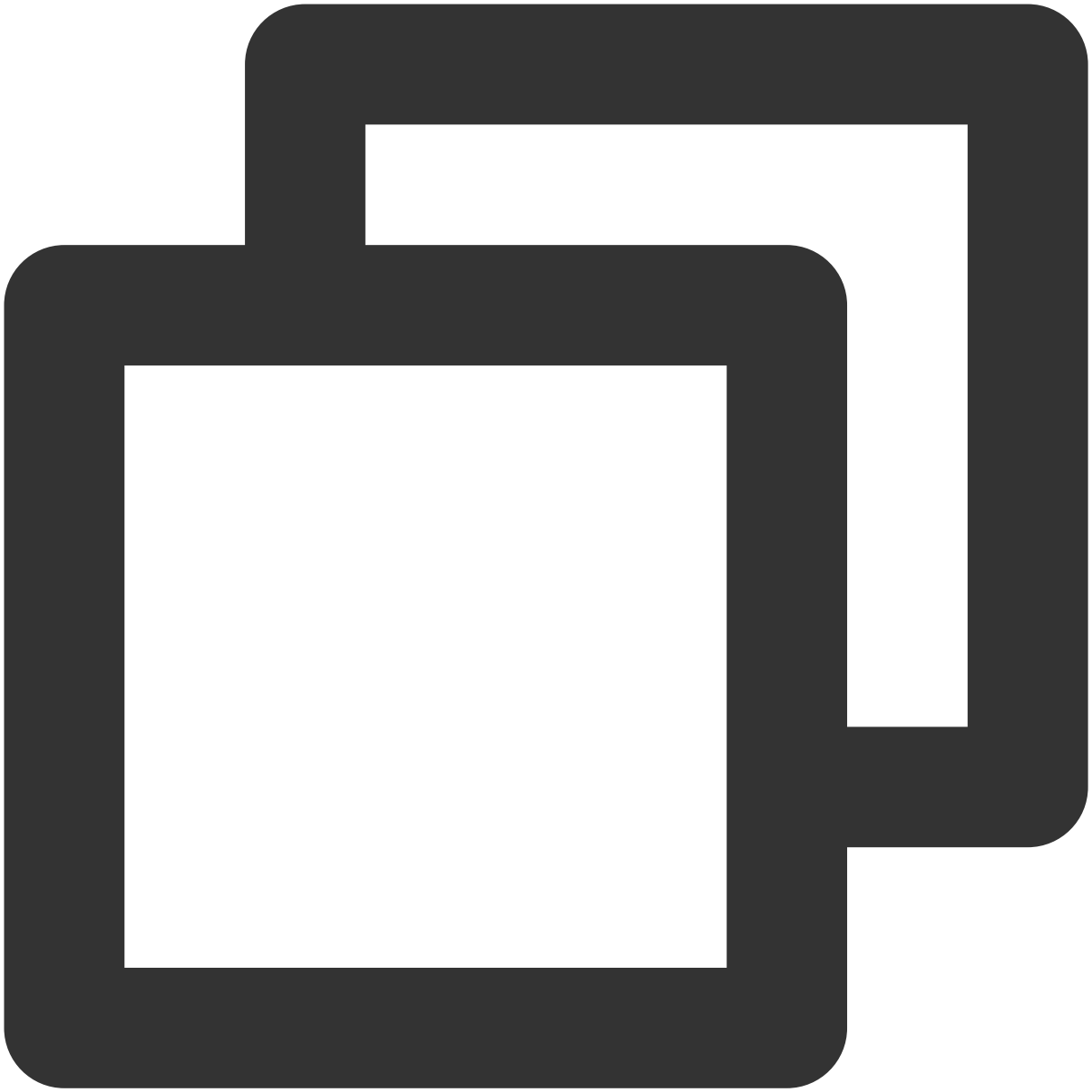
事件回调：



```
TencentCloudAvChatRoomCallback(
    onMemberEnter: (memberInfo) {}, // 有人进入直播间
    onRecvNewMessage: (message) {} // 收到弹幕消息
)
```

TencentCloudAvChatRoomCustomWidgets

自定义组件：

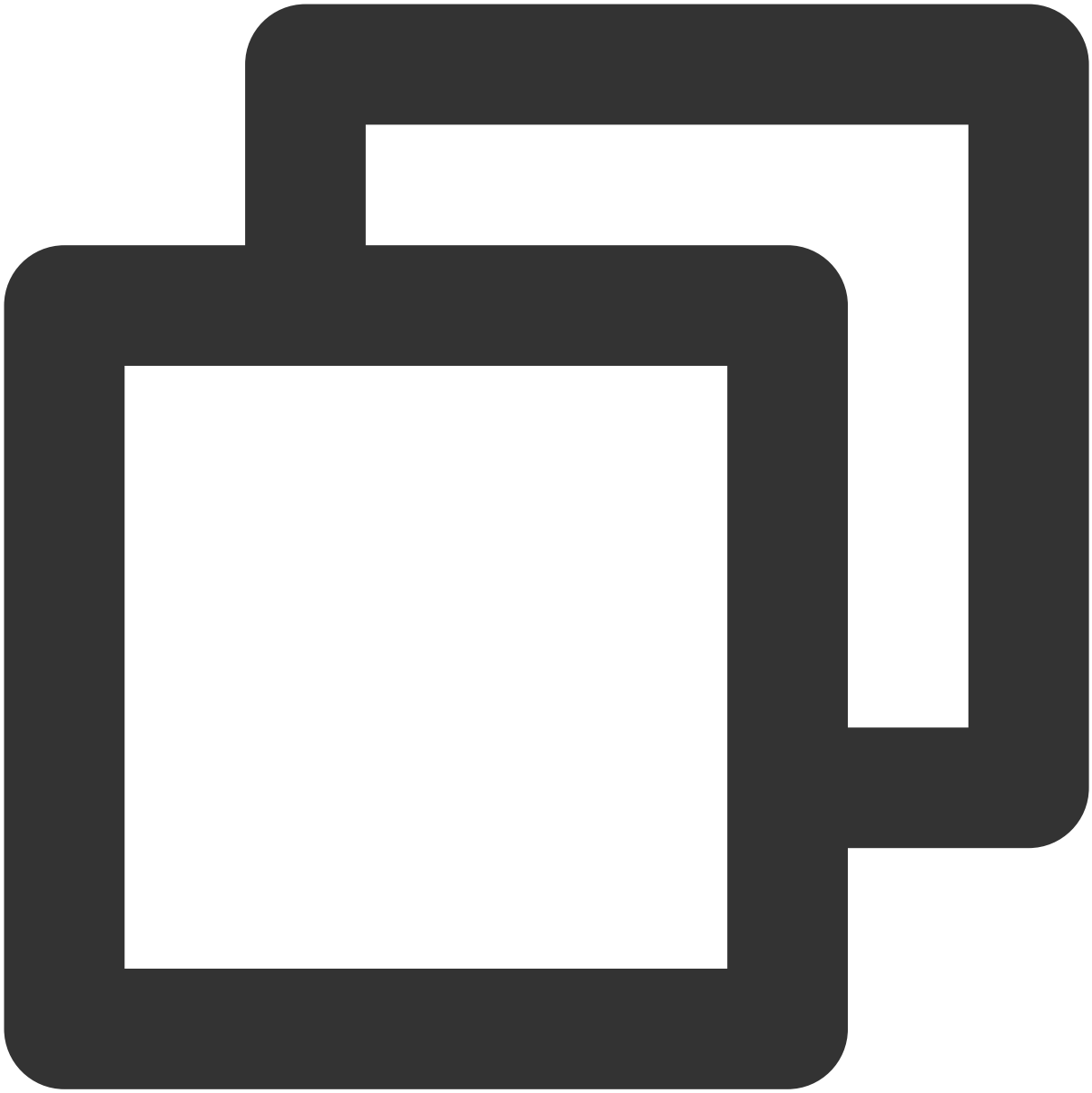


```
TencentCloudAvChatRoomCustomWidgets(
    roomHeaderAction: Container(), // 屏幕右上角显示区域自定义组件，默认显示的是直播间在线人
    roomHeaderLeading: Container(), // 屏幕左上角显示区域自定义组件，默认显示的是主播信息
    roomHeaderTag: Container(), // roomHeaderAction 和 roomHeaderLeading 下方，一般用
    onlineMemberListPanelBuilder: (context, id) { // 直播间在线人数点击后展开的面板自定义
        return Container();
    },
    anchorInfoPanelBuilder: (context, id) { // 主播头像点击后展开的面板自定义
        return Container();
    },
    giftsPanelBuilder: (context) { // 屏幕右下方礼物按钮点击后展示的面板自定义
```

```
            return Container();
        },
        messageItemBuilder: (context, message, child) { // 弹幕消息自定义
            return Container();
        },
        messageItemPrefixBuilder: (context, message) { // 弹幕消息前缀自定义，一般用于自定义粉
            return Container();
        },
        giftMessageBuilder: (context, message) { // 礼物消息自定义，从屏幕左侧滑入的礼物消息
            return Container();
        },
        textFieldActionBuilder: ( // 屏幕右下方区域自定义
            context,
        ) {
            return [Container()];
        },
        textFieldDecoratorBuilder: (context) { // 屏幕左下方输入框自定义
            return Container();
        }
    )
```

TencentCloudAvChatRoomTheme

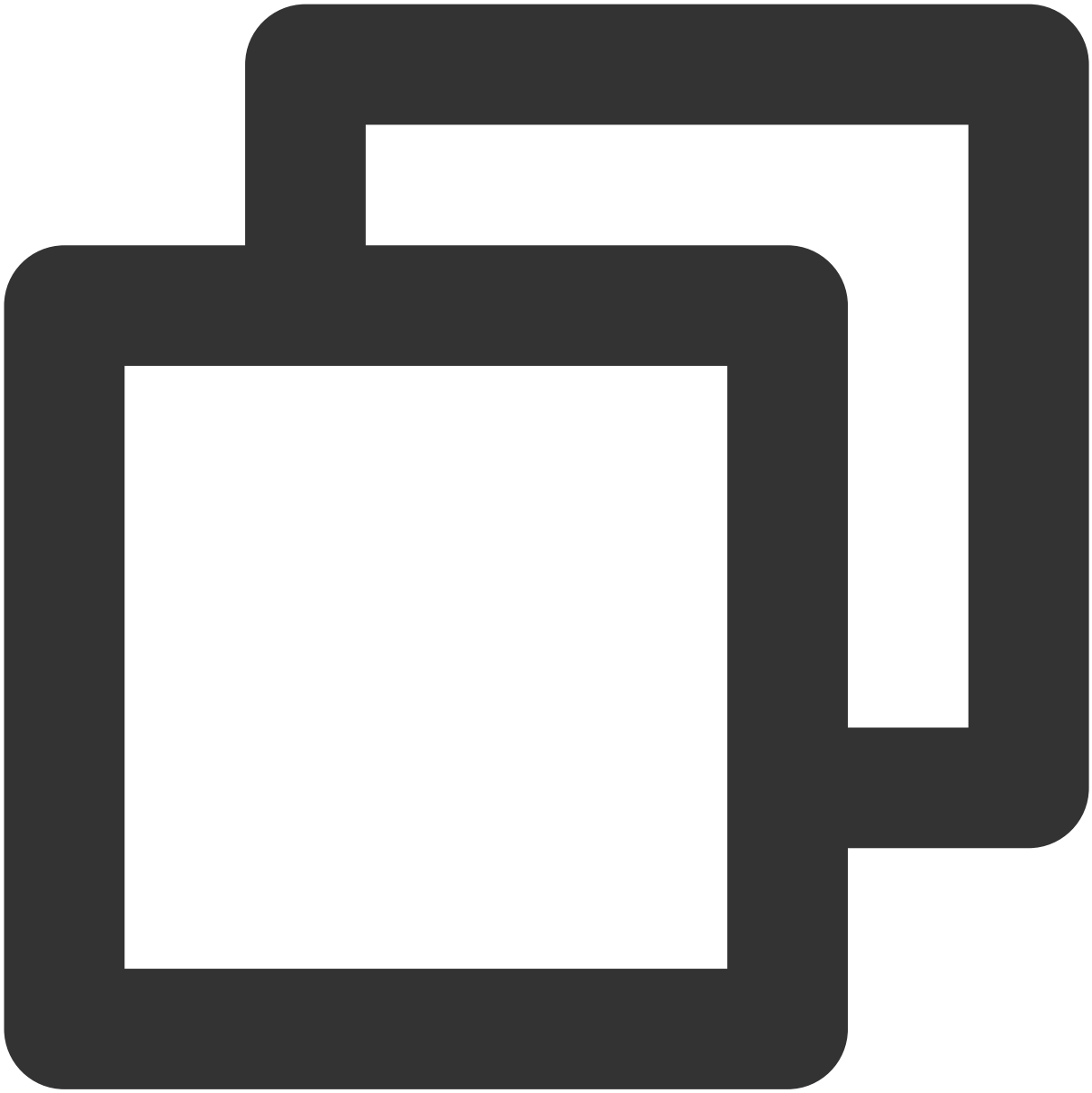
主题：



```
TencentCloudAvChatRoomTheme(
    backgroundColor: Colors.black, // 小部件的背景色,
    hintColor: Colors.red, // hint text 颜色
    highlightColor: Colors.orange, // 高亮颜色
    accentColor: Colors.white, // 前景色
    textTheme: TencentCloudAvChatRoomTextTheme(), // 字体主题
    secondaryColor: Colors.grey, // 次色
    inputDecorationTheme: InputDecorationTheme() // 输入框主题
)
```

TencentCloudAvChatRoomTextTheme

字体主题：



```
TencentCloudAvChatRoomTextTheme(
    giftBannerSubTitleStyle: TextStyle(), // 屏幕左侧划入的礼物消息 sub title 字体主题,
    giftBannerTitleStyle: TextStyle(), // 屏幕左侧划入的礼物消息 title 字体主题
    anchorTitleStyle: TextStyle(), // 主播名称字体主题
    anchorSubTitleStyle: TextStyle(), // 点赞字体主题
    barrageTitleStyle: TextStyle(), // 弹幕消息发送人名称主题
    barrageTextStyle: TextStyle() // 弹幕消息内容主题
)
```

AI聊天机器人方案

最近更新时间：2023-05-23 15:14:18

随着 ChatGPT 在全球范围的爆火，AI 已成为当下开发者最为关注的焦点，国内各大厂商也纷纷跟进，推出了各自的大模型应用与产品。很多应用都在尝试与AI结合，寻找新的发力点。而新一代大语言模型的强大对话交流能力与各类即时通信场景天然契合，这为 IM 与 AI 结合带来了广阔的想象空间。

在办公场景，用户可以直接通过对话让 AI 辅助自己高效完成工作纪要、文案编写、信息搜集等需求；在客服场景，与 AI 结合的智能客服可以提供真人客服般的会话体验，更有效地引导用户进行购买、使用；在社交场景，AI 聊天机器人可以为用户提供24小时在线的心理咨询和情感陪伴，提升用户粘性……腾讯云即时通信 IM 作为全球领先的通信云服务商，也看到了 AI 在即时通信场景的巨大潜力，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，高效实现场景创新。

那具体该如何接入 AI 服务呢？本文将拆解接入 AI 服务的各个步骤，为您详细介绍如何通过腾讯云即时通信 IM 第三方回调功能，将 AI 服务能力引入到 IM 应用中，创建一个可以智能聊天的 AI 机器人，为用户提供真人般对话体验，实现智能客服、创意辅助、工作助手等功能（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）。

准备工作

注册腾讯云 IM 账号

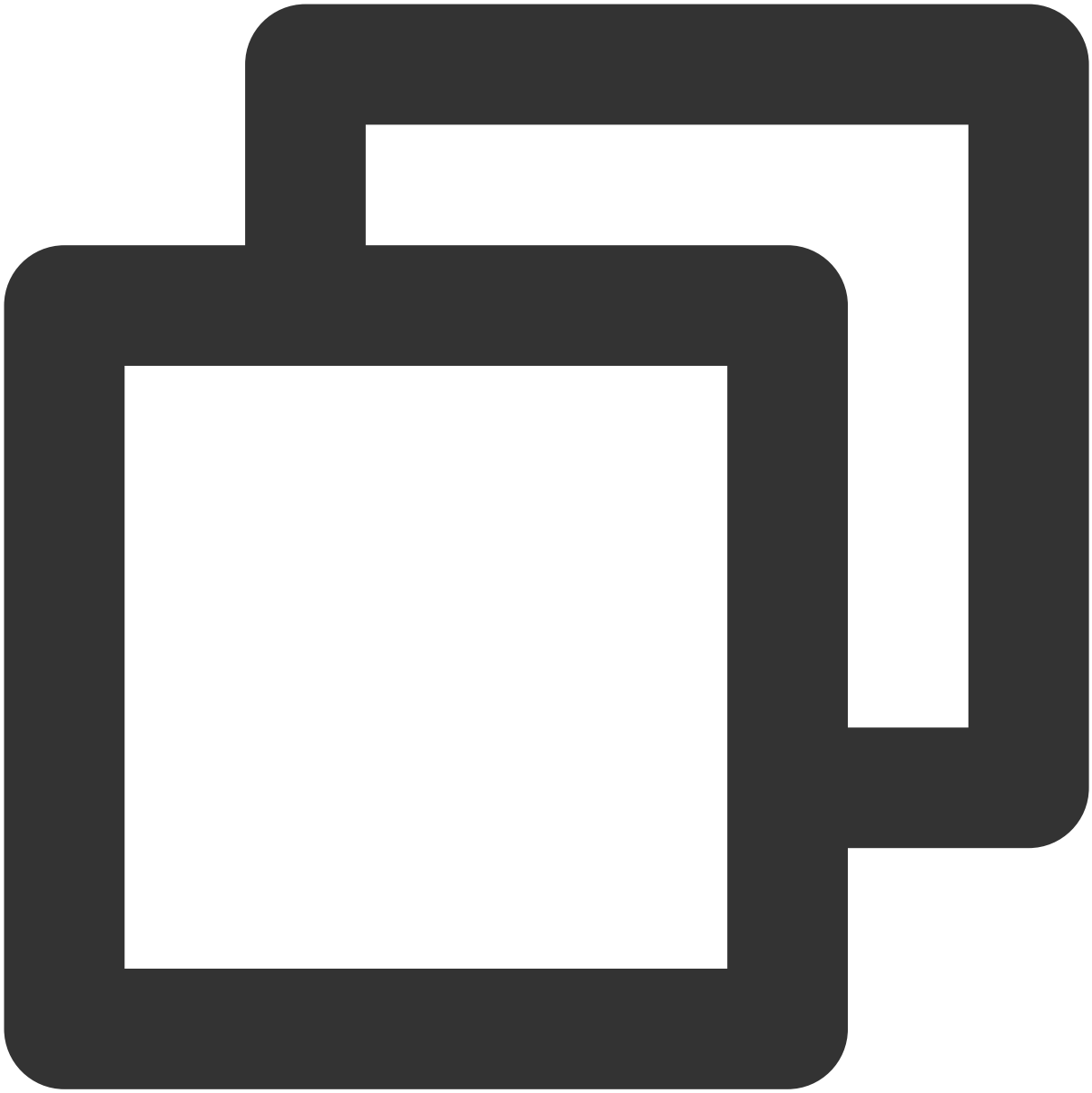
注册并登录腾讯云账号，进入即时通信 IM 控制台，创建应用，并获取应用的 SDKAppID 和密钥（以下称为 IM Key），并创建一个管理员账号 administrator。

注册对应 AI 服务商账号

注册并登录计划接入的 AI 服务商的相应账号并获取 API 密钥（以下称为AI_SECRET_KEY）。

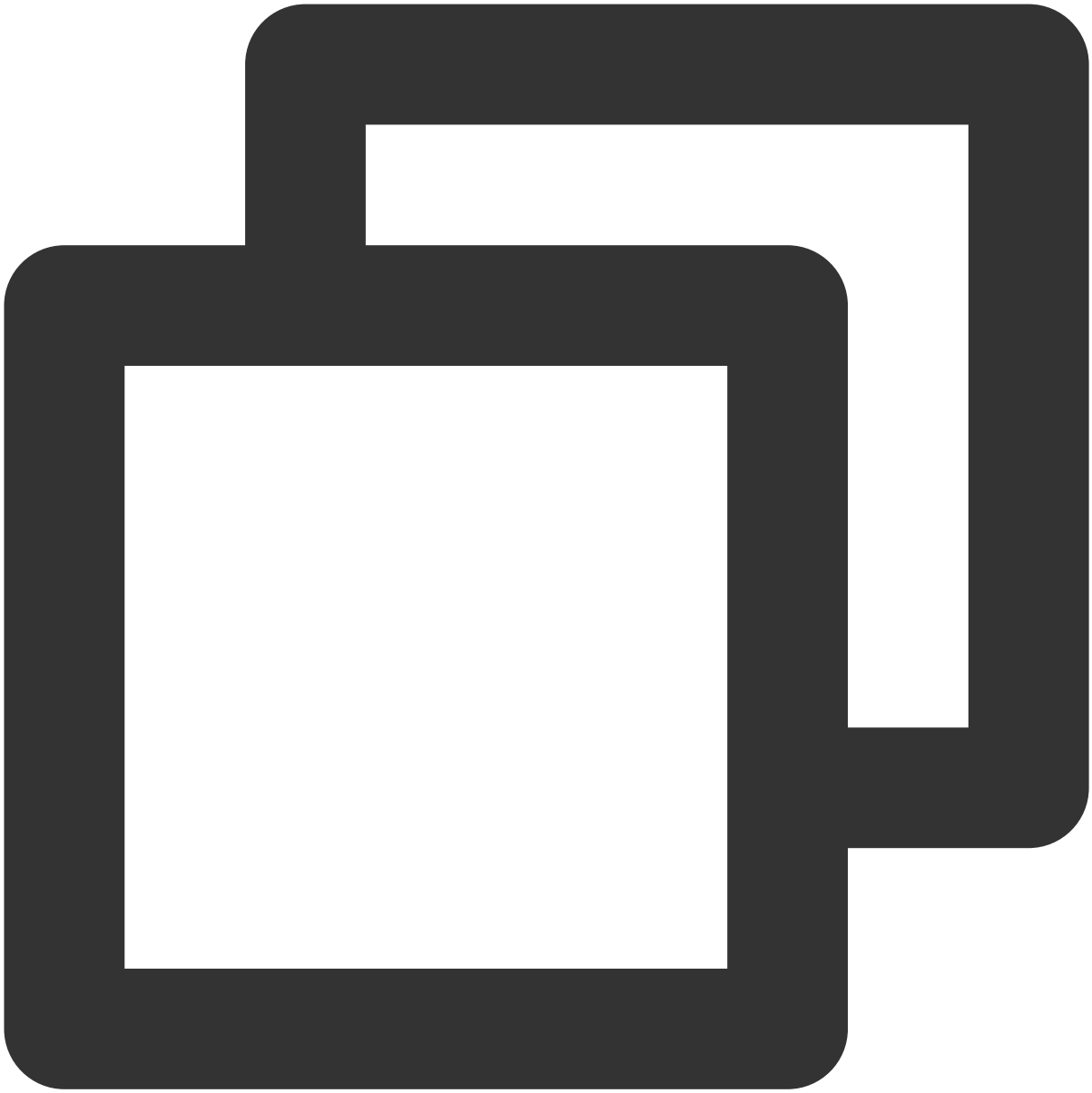
创建腾讯云 IM 机器人账号

通过 REST API 创建一个腾讯云 IM 机器人账号。腾讯云机器人是一种特殊的用户，UserID 以 @RBT# 开头。



```
curl -d '{"UserID":"@RBT#001","Nick":"MyRobot"}' "https://console.tim.qq.com/v4/ope
```

将上述命令的 sdkappid={} 和 usersig={} 替换成您的 SDKAppID 和使用 IM Key 生成的 Usersig，详情可参见 生成 UserSig。在 Linux 环境运行上述命令后，腾讯云服务器返回：



```
{"ActionStatus": "OK", "ErrorCode": 0, "ErrorInfo": ""}
```

表示成功创建了一个昵称为 MyRobot 的机器人@RBT#001。

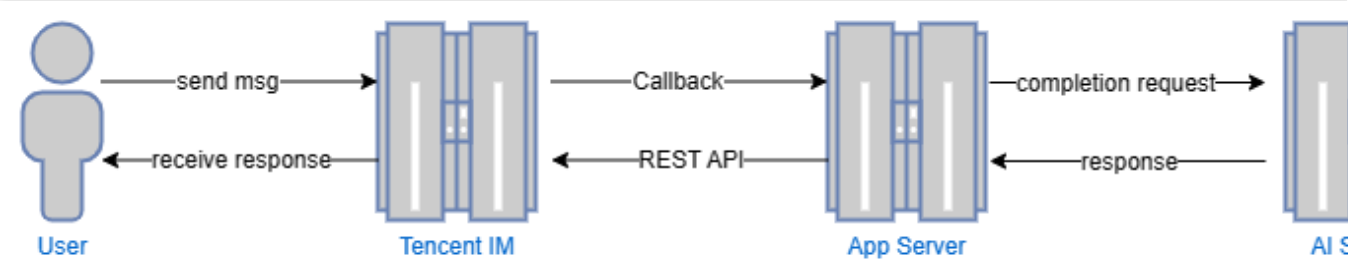
配置腾讯云 IM 第三方回调

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。

编写 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 后台发送第三方回调将事件通知 App 后台；
3. App 后台收到事件通知，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；
4. App 后台调用 AI 服务接口（即MiniMax API），并得到响应回复内容，如 nice to meet you；
5. App 后台调用云 IM 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。



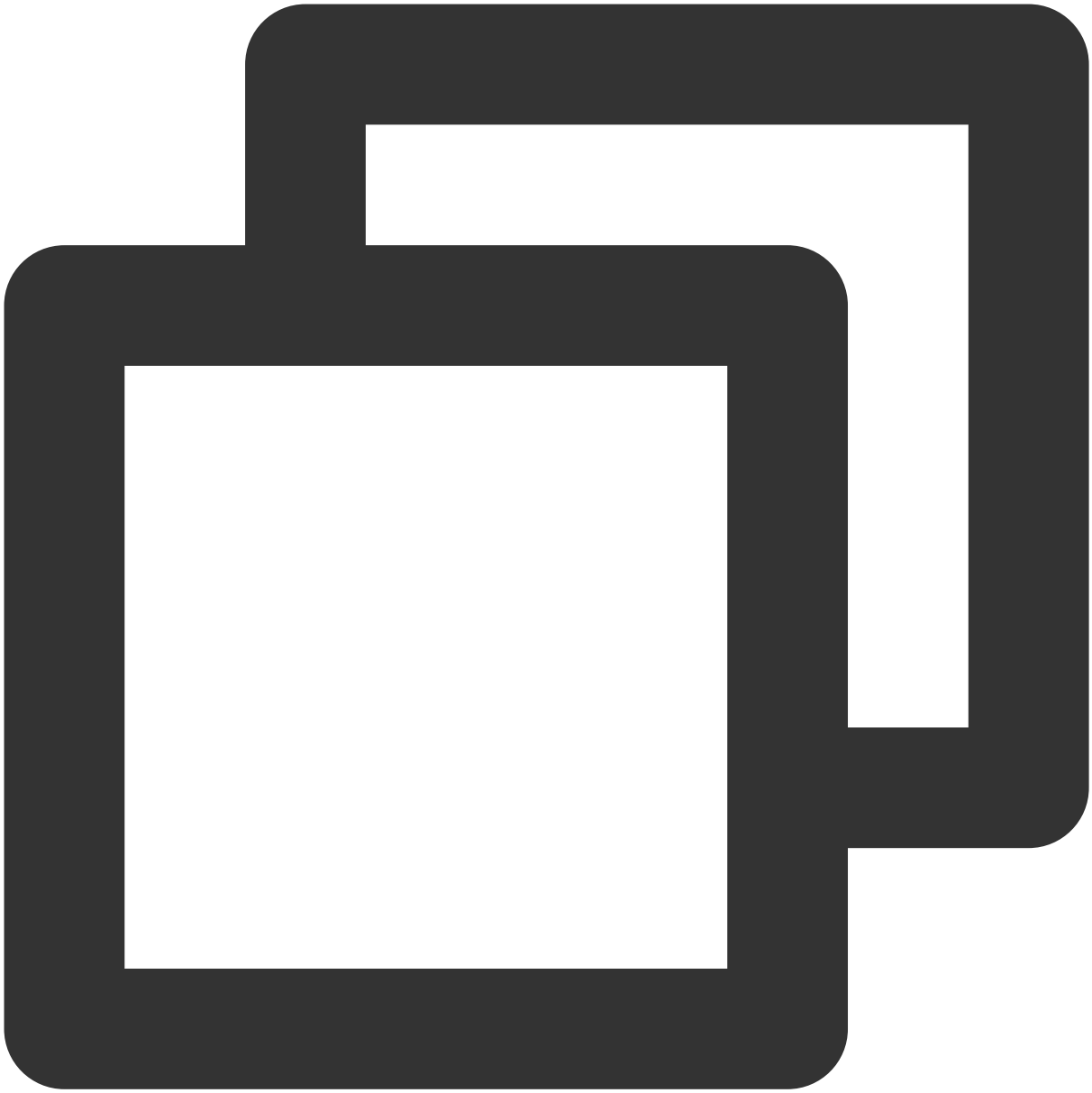
以 Golang 为例，App 后台的关键代码可参见下文。

注意：

下文代码仅作为展示用途，省略了大量异常处理代码，不可直接用于生产环境。

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。

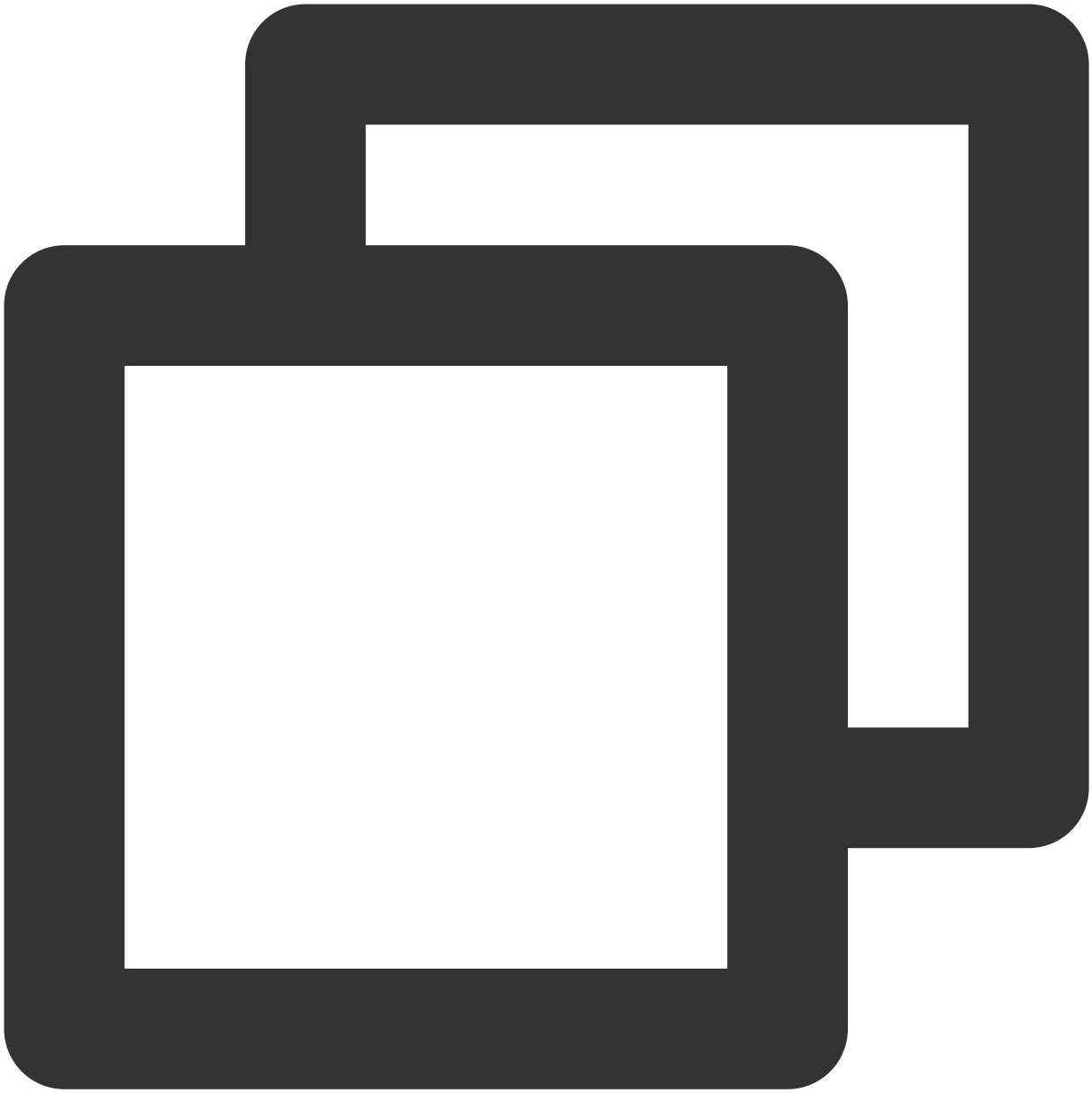


```
func handler(w http.ResponseWriter, r *http.Request) {
    command := r.URL.Query().Get("CallbackCommand")
    reqbody, _ := io.ReadAll(r.Body)
    var rspbody []byte
    switch command {
    case "Bot.OnC2CMessage": // 机器人C2C回调命令字
        dealC2c(context.Background(), reqbody)
        rspbody =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

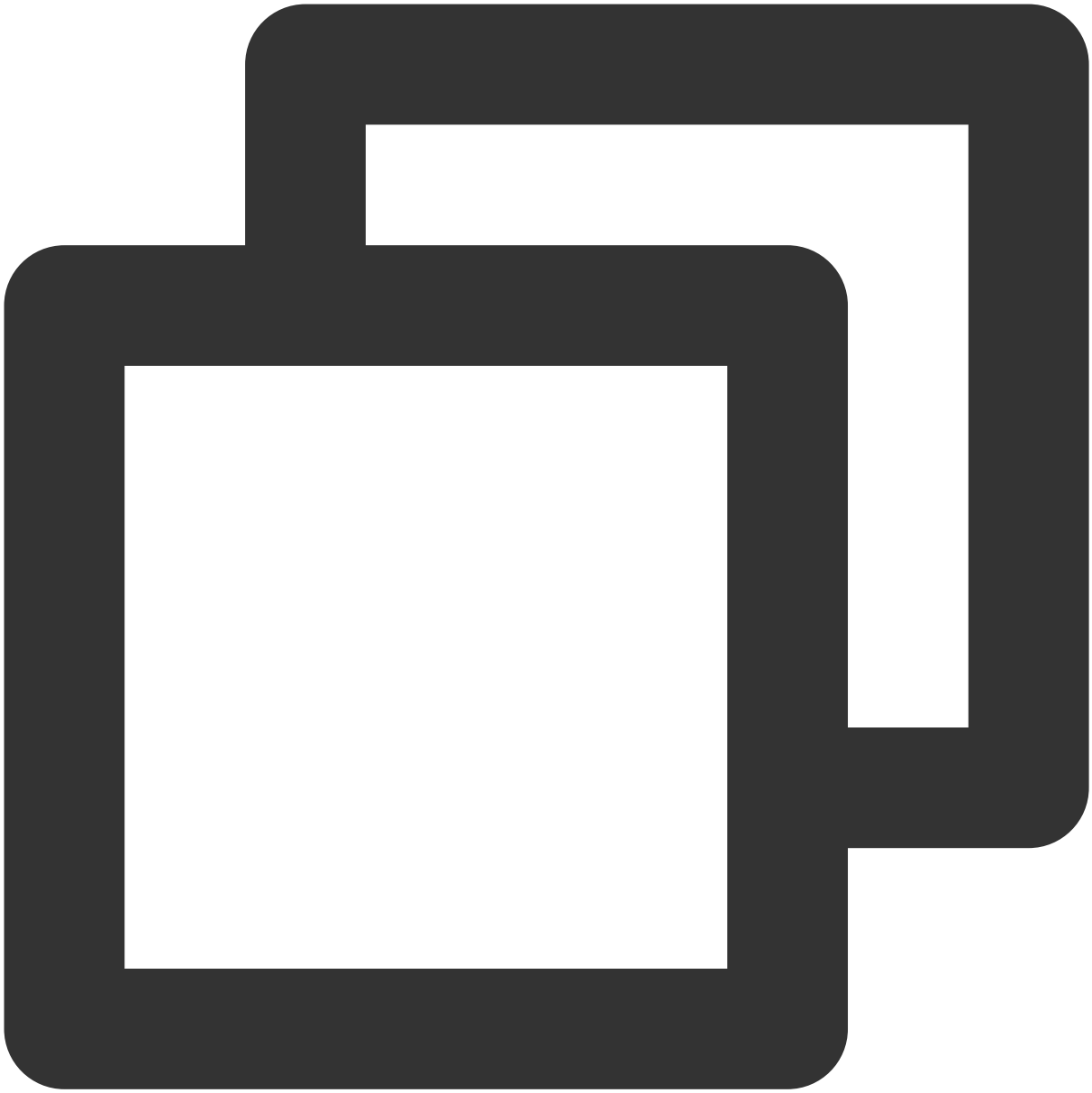


```
func dealC2c(ctx context.Context, reqbody []byte) error {
  root, _ := simplejson.NewJson(reqbody)
  jFromAccount := root.Get("From_Account")
  fromAccount, _ = jFromAccount.String()
  // 检查发送方ID，不处理机器人发送给机器人的请求，防止无限循环
  if strings.HasPrefix(fromAccount, "@RBT#") {
    return nil
  }
  jToAccount := 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 MiniMax 实现智能聊天的功能，您可以将 MiniMax 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。

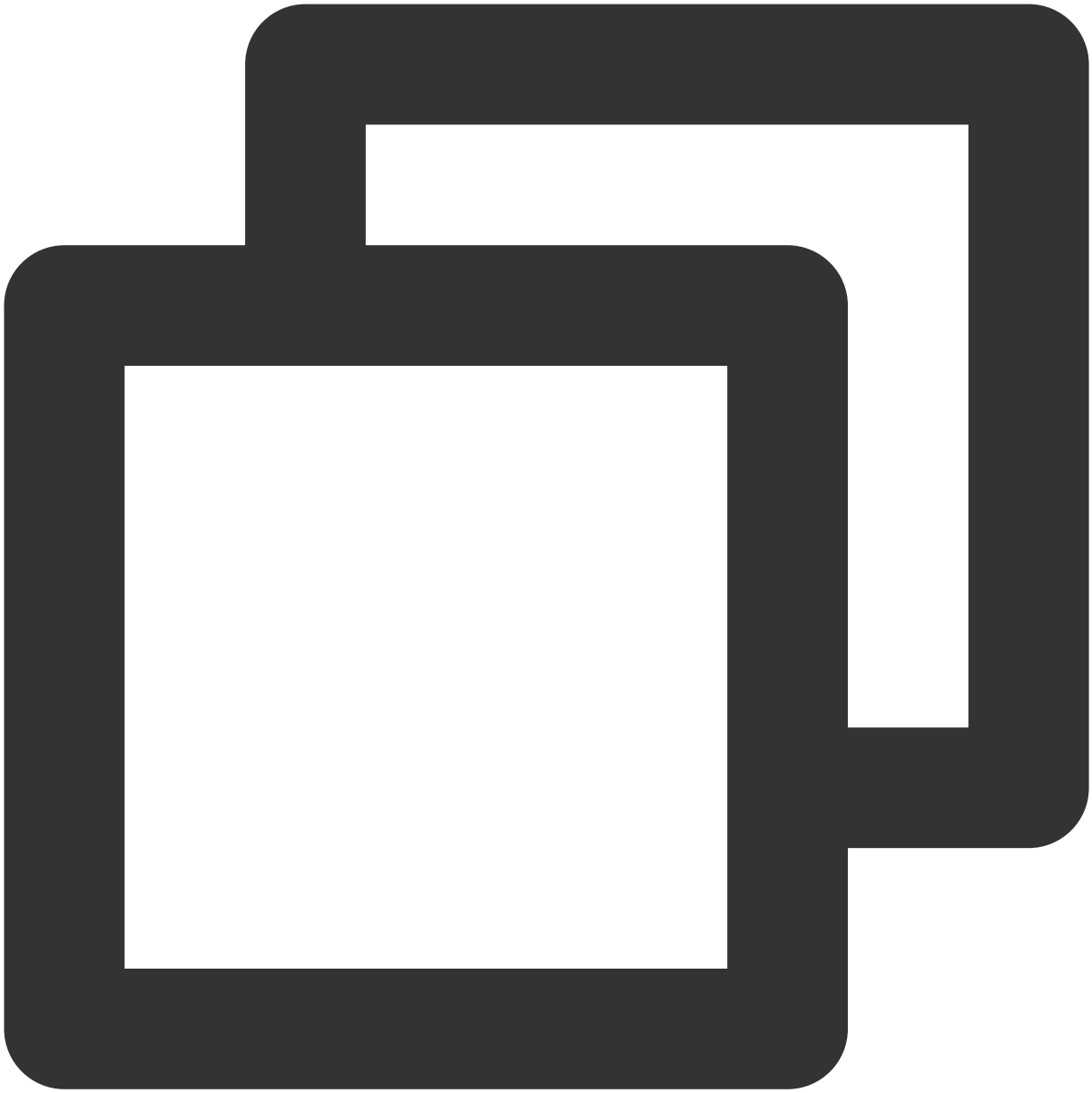


```
type MiniMaxRsp struct {
  Reply string `json:"reply"`
}

// 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，制定消息发送方为@RBT#001，接收方为 user1，模拟机器人回复用户。

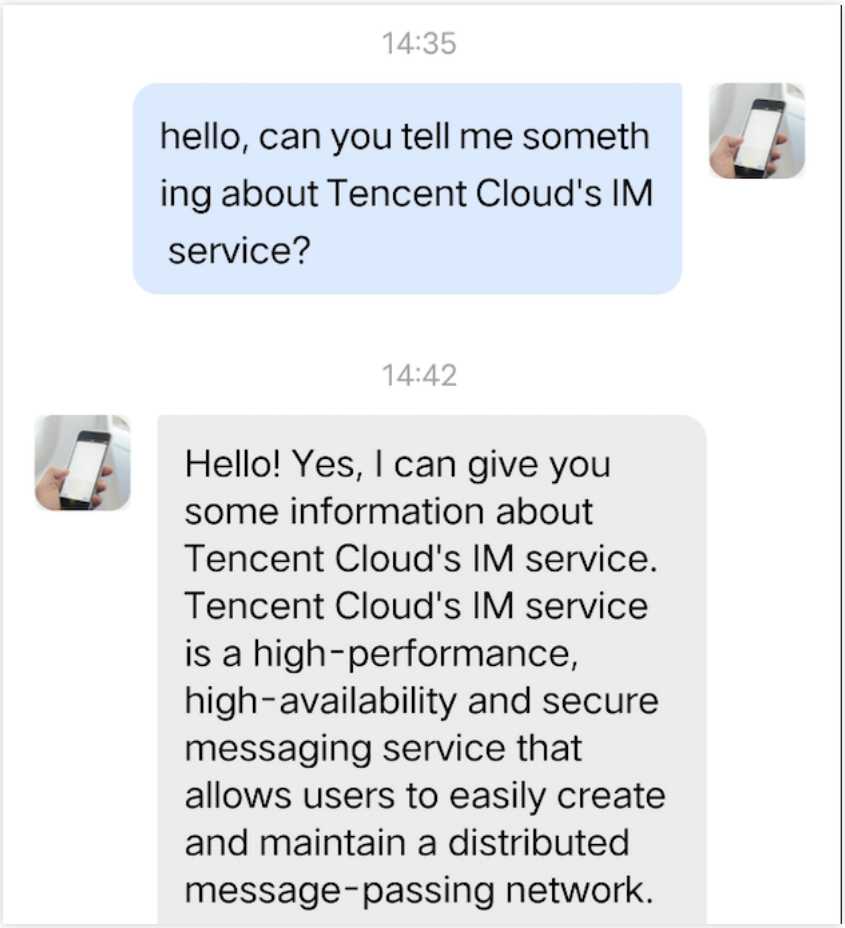


```
// 发送一个REST API请求
func doRestAPI(host string, sdkappid int, admin, usersig, command, body string) {
  url := fmt.Sprintf("https://%s/v4/%s?sdkappid=%d&identifier=%s&usersig=%s&random=
    host, command, sdkappid, admin, usersig, rand.Uint32())
  req, _ := http.NewRequest("POST", url, bytes.NewBufferString(body))
  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 的实现可以参考腾讯云文档
    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/se
}
```

效果展示

使用腾讯云 IM 的 demo 实现，最终的效果展示如下：



通过以上步骤，我们便实现了腾讯云 IM 在服务端和 MiniMaxAI 开放平台的单聊对接，接入其他 AI 服务商也可参照上述步骤，仅需将 askAI 函数替换成其他AI服务商的对应 API 调用即可。对于群聊机器人，开发者仅需补充实现 Bot.OnGroupMessage 回调命令处理即可。

端到端加密聊天

最近更新时间：2023-07-27 16:34:39

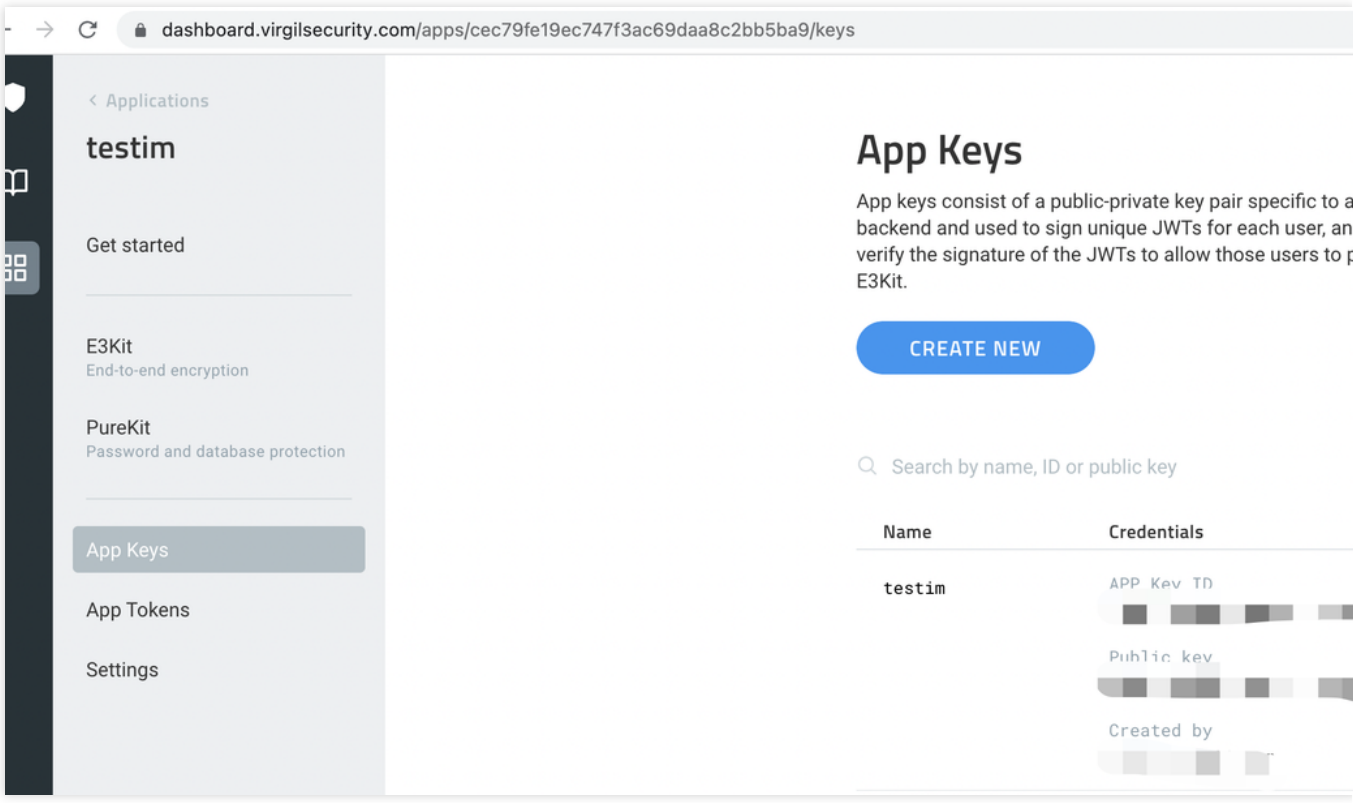
端到端加密聊天

方案概述

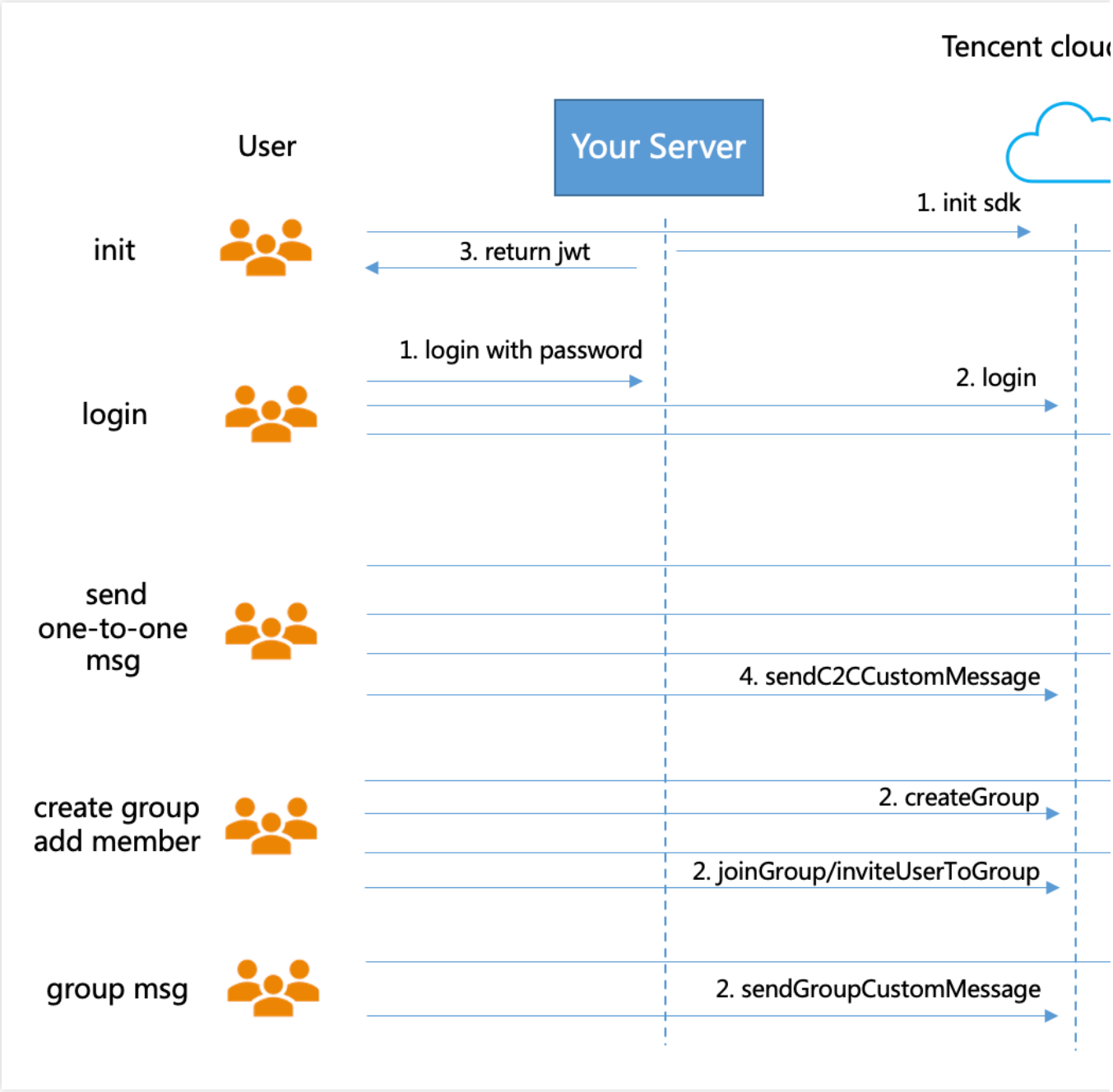
该方案要由客户应用侧来基于 virgilsecurity的E3Kit + 腾讯云im sdk进行实现，E3Kit对应的传送门：Virgil E3Kit | Virgil Security

阅读下列方案前先对 Group Encryption - End-to-End Encryption - E3Kit | Virgil Security的文档进行了解，特别是jwt token（JSON Web Token）, create channel/create group，Encrypt and decrypt messages部分。

使用前先在virgilsecurity控制台开通应用，该产品是收费产品，具体定价看Pricing | Virgil Security



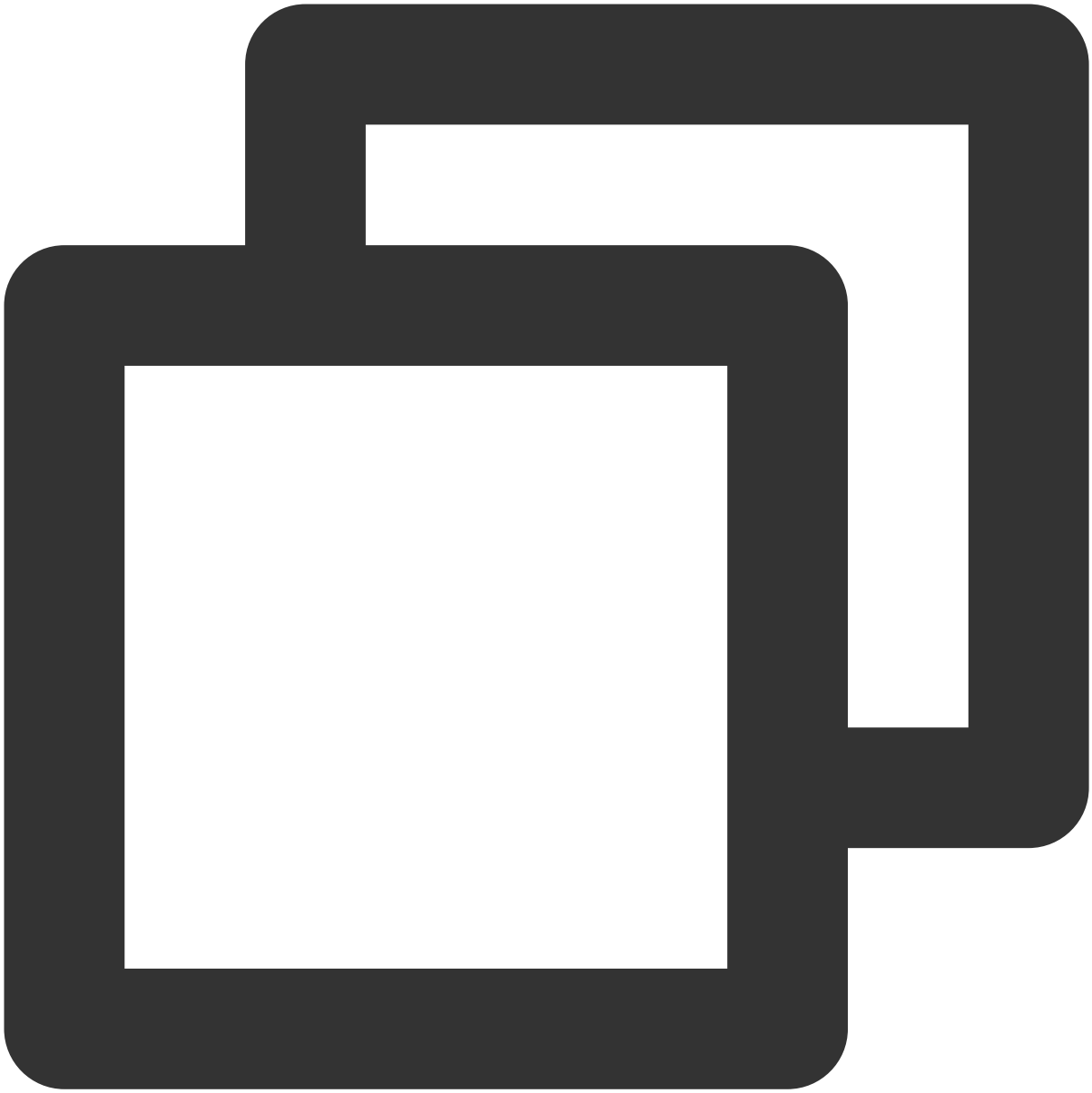
整体系统交互流程如下：



方案详解(以安卓为例)

初始化

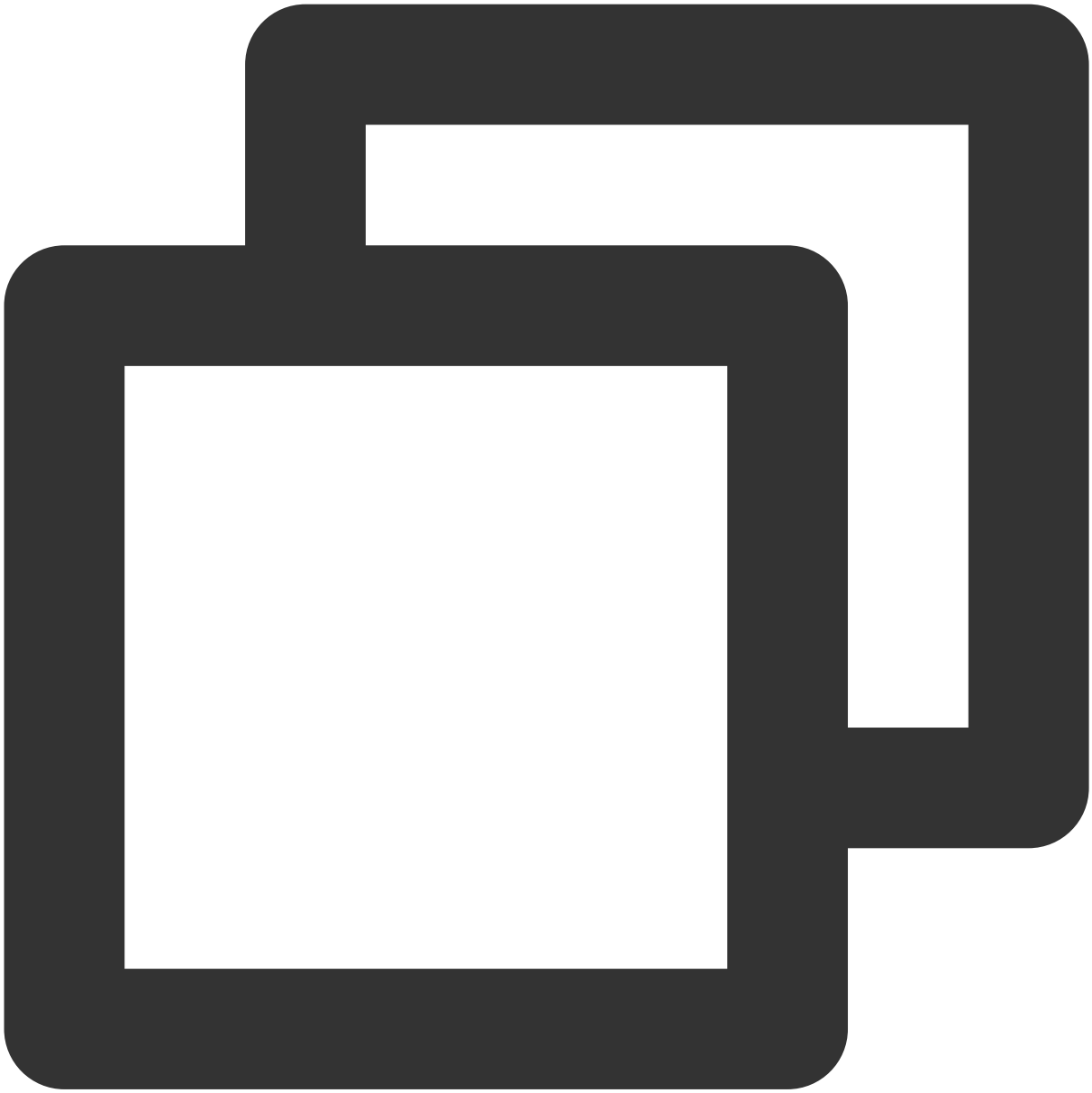
1. 腾讯云 im sdk初始化



```
// 1. 从即时通信 IM 控制台获取应用 SDKAppID。
// 2. 初始化 config 对象。
V2TIMSDKConfig config = new V2TIMSDKConfig();
// 3. 指定 log 输出级别。
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_INFO);
// 4. 添加 V2TIMSDKListener 的事件监听器，sdkListener 是 V2TIMSDKListener 的实现类，如果您
V2TIMManager.getInstance().addIMSDKListener(sdkListener);
// 5. 初始化 IM SDK，调用这个接口后，可以立即调用登录接口。
V2TIMManager.getInstance().initSDK(context, sdkAppID, config);
```

2.E3Kit初始化，传入控制台的配置信息，生成jwt。对应操作文档链接：Generate Client Tokens - Get Started - E3Kit | Virgil Security

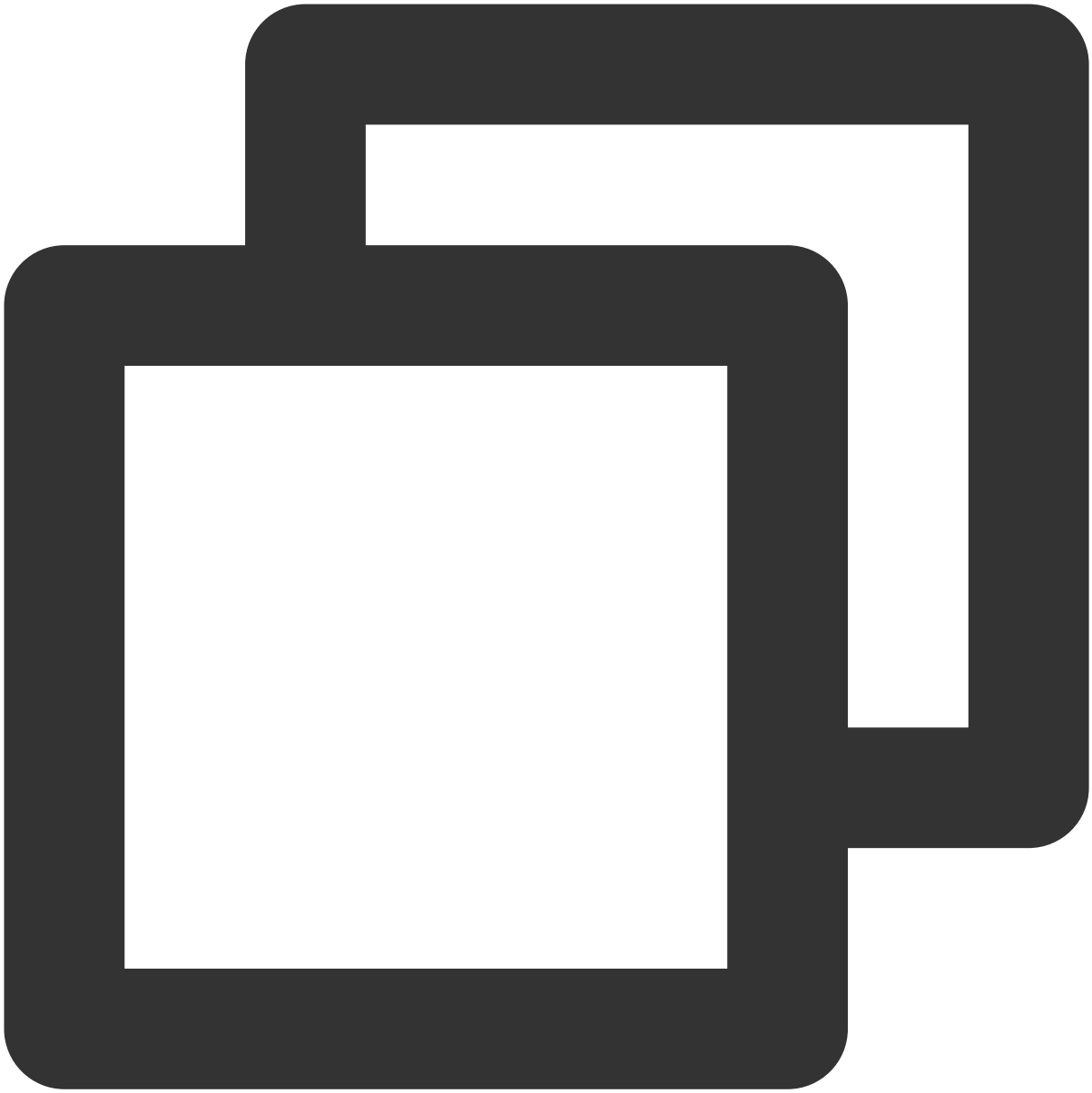
服务器生成jwt token，下发给客户端



```
// 生成jwt
// App Key (you got this Key at the Virgil Dashboard)
String appKeyBase64 = "MC4CAQAwBQYDK2VwBCIEINlK4BhgsijAbNmUqU6us0ZU9MGi+HxdYCA6TdZe
byte[] appKeyData = ConvertionUtils.base64ToBytes(appKeyBase64);

// Crypto library imports a key pair
VirgilCrypto crypto = new VirgilCrypto();
```

```
VirgilKeyPair keyPair = crypto.importPrivateKey(appKeyData);

// Initialize an access token signer that signs users JWTs
VirgilAccessTokenSigner accessTokenSigner = new VirgilAccessTokenSigner();

// Use your App Credentials you got at the Virgil Dashboard:
String appId = "be00e10e4e1f4bf58f9b4dc85d79c77a";
String appKeyId = "70b447e321f3a0fd";
TimeSpan ttl = TimeSpan.fromTime(1, TimeUnit.HOURS); // 1 hour - JWT's lifetime

// Setup a JWT generator with the required parameters:
JwtGenerator jwtGenerator =
    new JwtGenerator(appId, keyPair.getPrivateKey(), appKeyId, ttl, accessTokenSign

// Generate a JWT for a user
// Remember that you must provide each user with a unique JWT.
// Each JWT contains unique user's identity (in this case - Alice).
// Identity can be any value: name, email, some id etc.
String identity = "Alice";
Jwt aliceJwt = jwtGenerator.generateToken(identity);

// As a result you get user's JWT, it looks like this: "eyJraWQiOiI3MGI0NDdlMzIxZjN
// You can provide users with JWT at registration or authorization steps.
// Send a JWT to client-side.
String jwtString = aliceJwt.stringRepresentation();
```

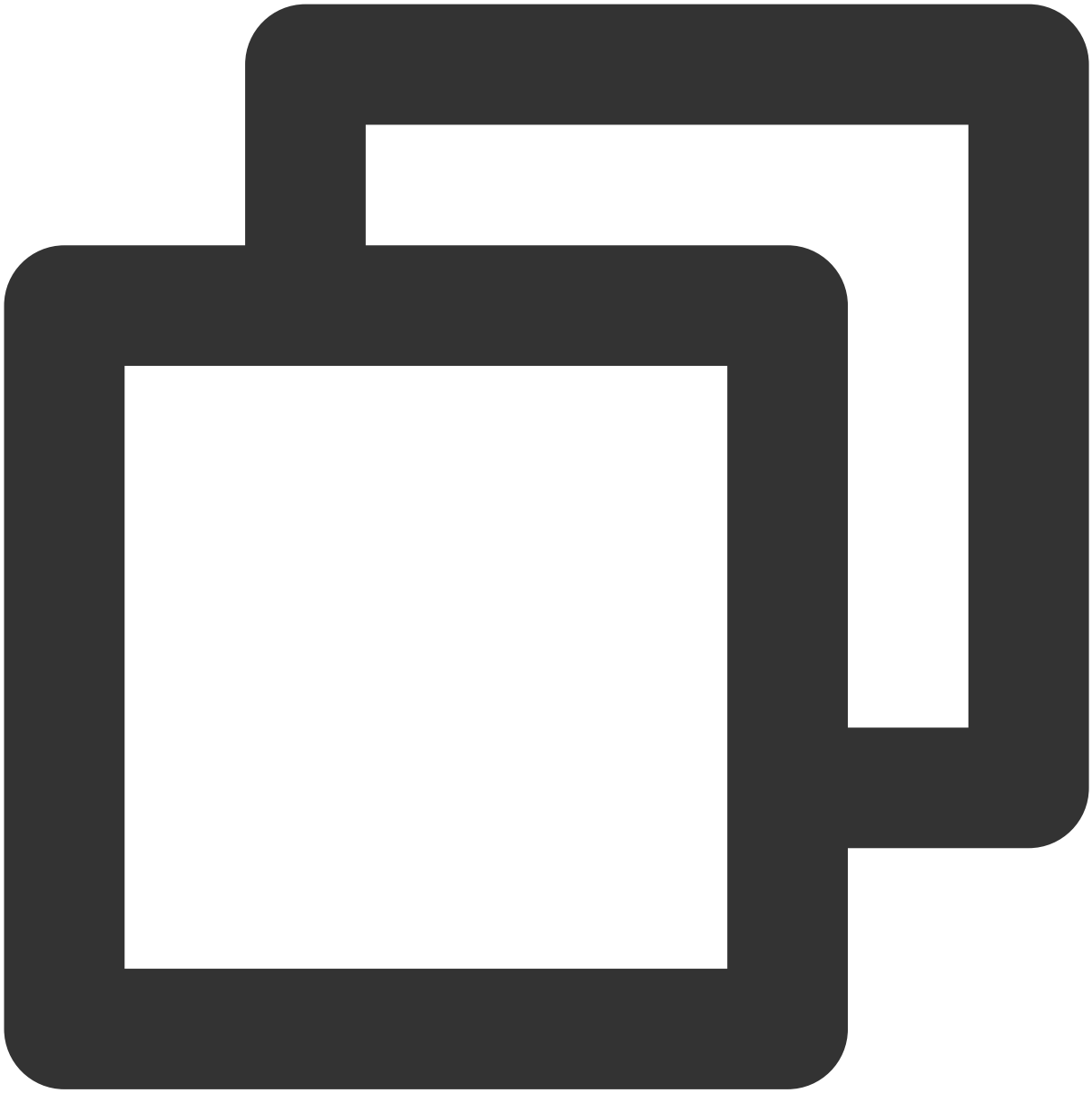
客户端安卓端初始化E3Kit, tokenCallback是上面服务器返回的jwt token, User1是用户id



```
// 初始化E3Kit
// create EThreeParams with mandatory parameters
// such as identity, tokenCallback and context
EThreeParams params = new EThreeParams("User1",
                                       tokenCallback,
                                       context);

// initialize E3Kit with the EThreeParams
EThree ethree = new EThree(params);
```

用户登陆

1. 调用腾讯云 im sdk login 接口进行账号登陆



```
String userID = "your user id";
//usersig生成看下面链接
//即时通信 IM 生成 UserSig-服务端 API-文档中心-腾讯云
String userSig = "userSig from your server";
V2TIMManager.getInstance().login(userID, userSig, new V2TIMCallback() {
    @Override
    public void onSuccess() {
        Log.i("imsdk", "success");
```

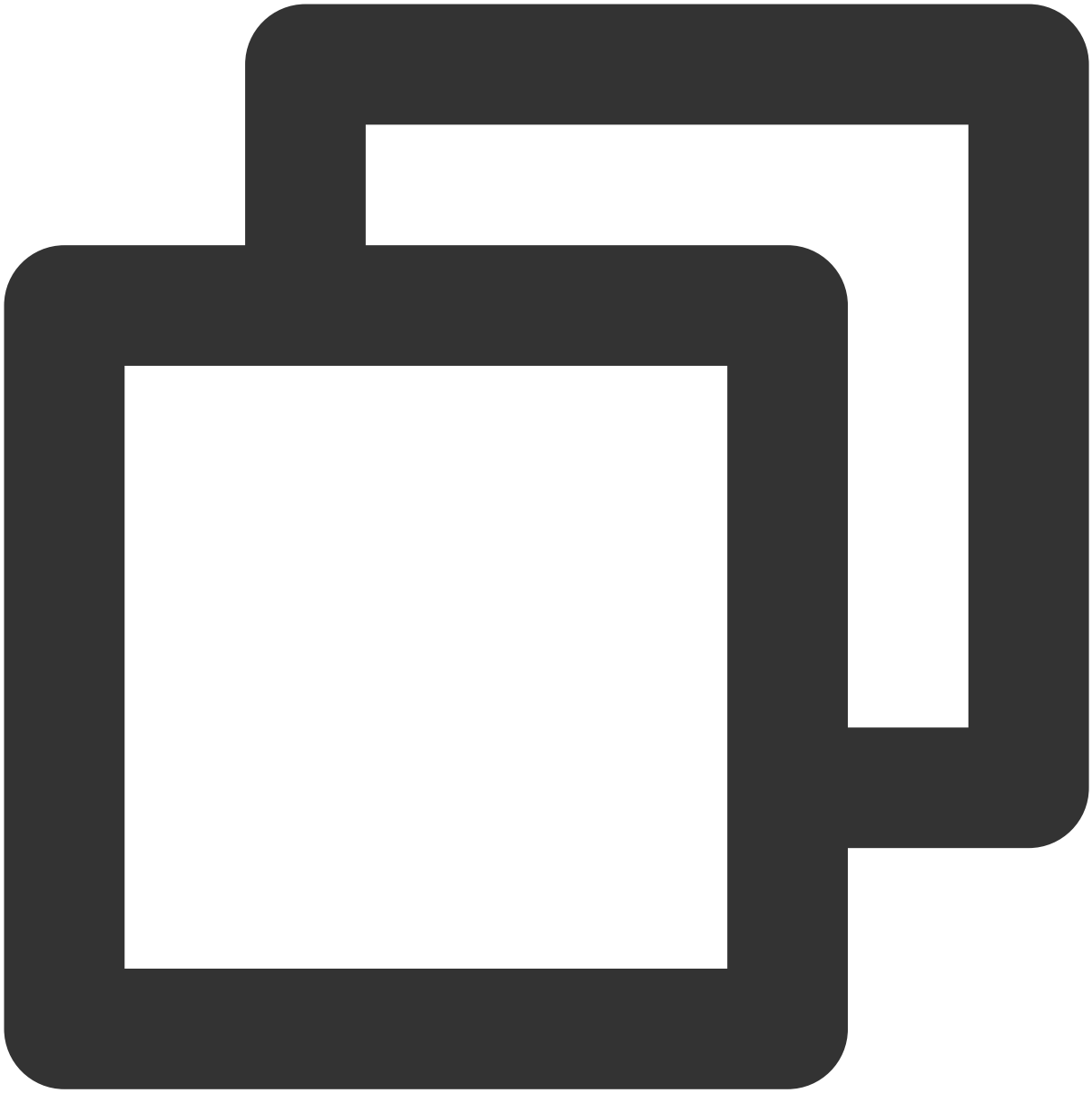
```
    }

    @Override
    public void onError(int code, String desc) {
        // 如果返回以下错误码，表示使用 UserSig 已过期，请您使用新签发的 UserSig 进行再次登录。
        // 1. ERR_USER_SIG_EXPIRED (6206)
        // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)
        // 注意：其他的错误码，请不要在这里调用登录接口，避免 IM SDK 登录进入死循环。
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

2.调用**eThree.****register**接口把用户注册到virgilsecurity上，对应操作文档链接：User Authentication - E3Kit | Virgil Security

> 注意：im的user_id和注册到virgilsecurity上的user_id要保持一致

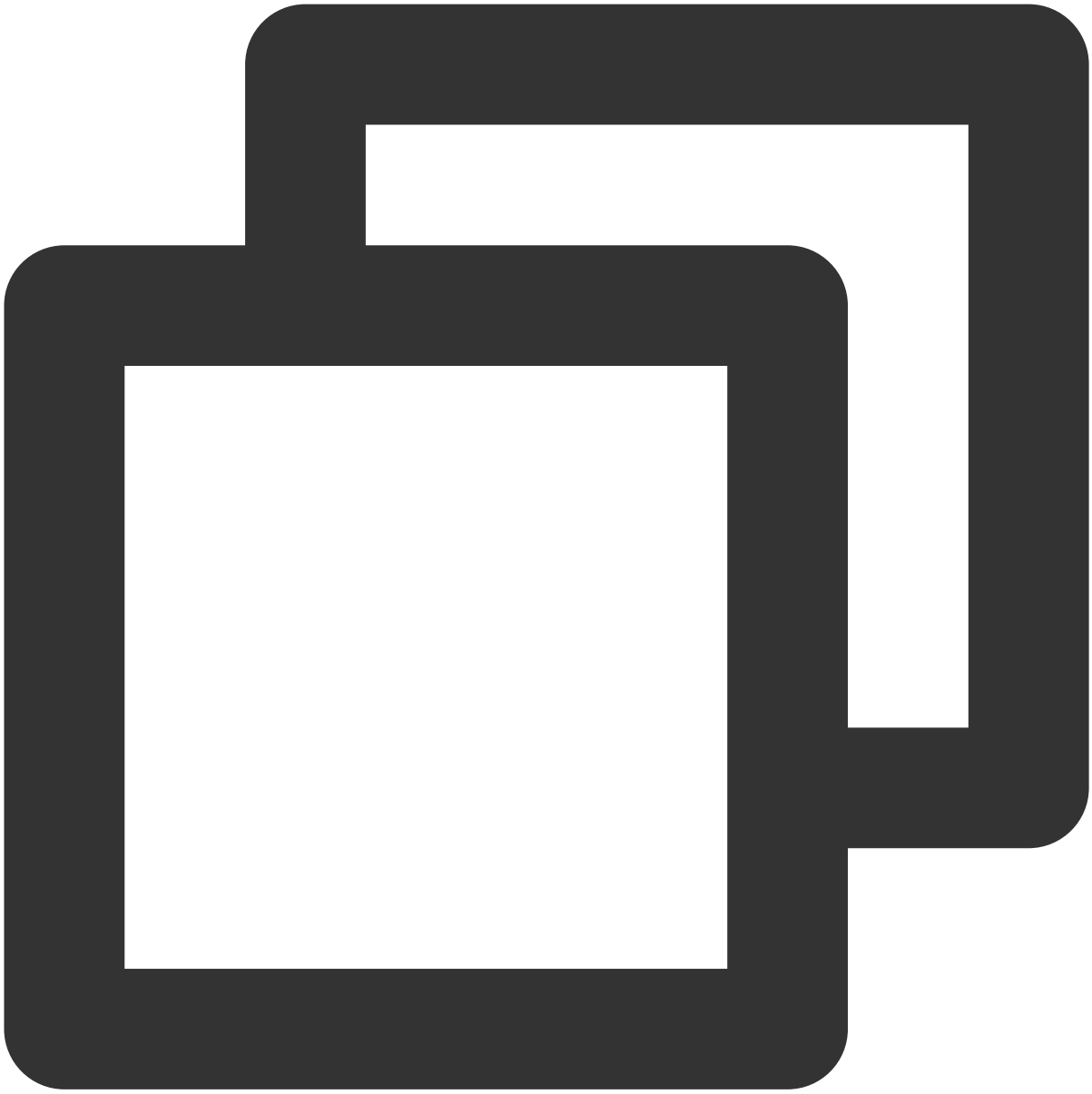
发起单聊会话

1.在E3Kit中调用 ethree.createRatchetChannel 接口创建one-to-one会话(**User1和User2**), 对应操作文档链接：
https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/double-ratchet/?#create-channel
User1创建了跟User2之间的channel



```
// 创建one-to-one channel
ethree.createRatchetChannel(users.get("User2"))
       .addCallback(new OnResultListener<RatchetChannel>() {
           @Override public void onSuccess(RatchetChannel ratchetChannel) {
               // Channel created and saved locally!
           }

           @Override public void onError(@NotNull Throwable throwable) {
               // Error handling
           }
       });
```

然后User2就可以加入channel

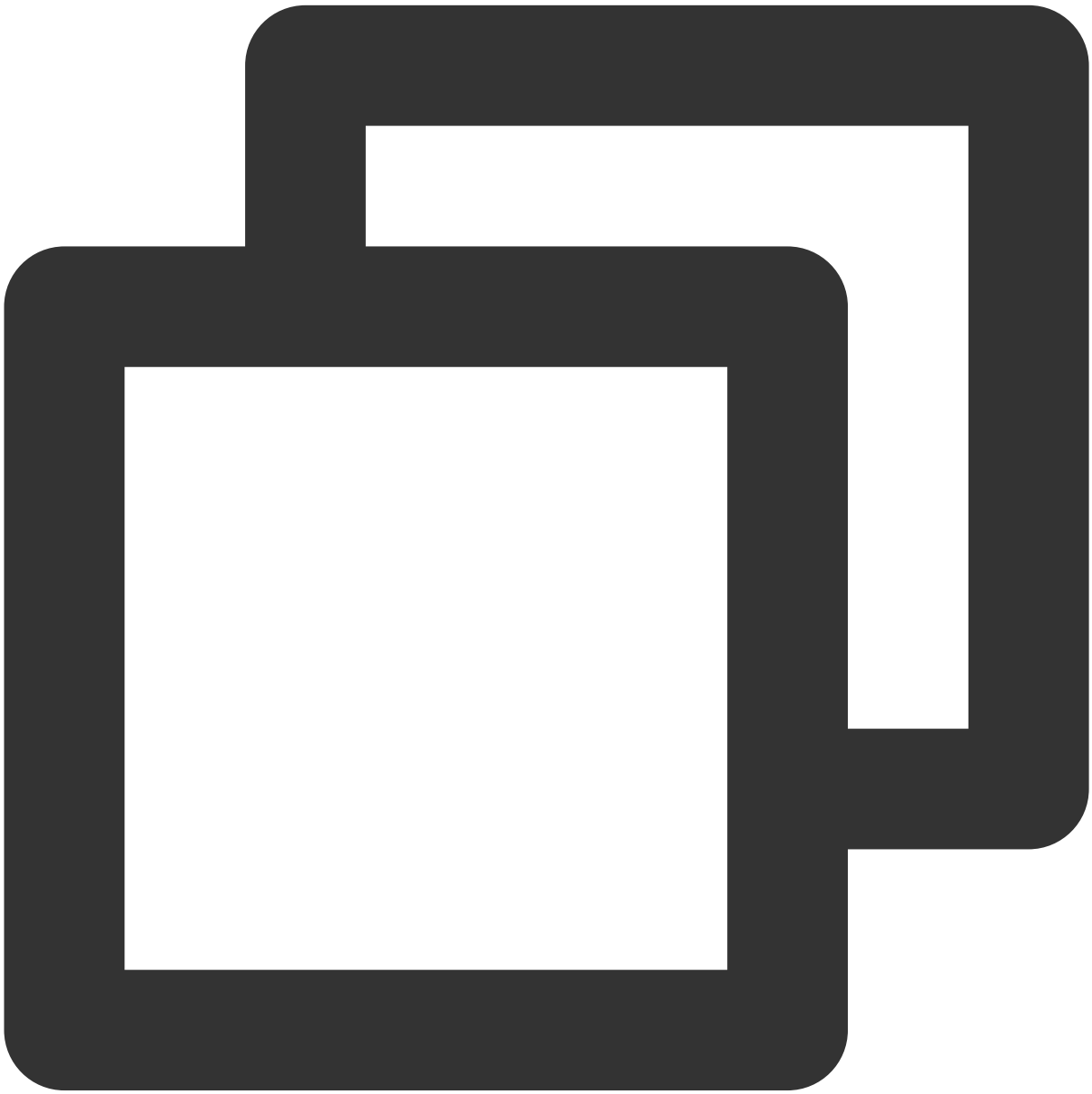


```
// 单聊的另一方加入channel
ethree.joinRatchetChannel(users.get("User1"))
       .addCallback(new OnResultListener<RatchetChannel>() {
    @Override public void onSuccess(RatchetChannel ratchetChannel) {
        // Channel joined and saved locally!
    }

    @Override public void onError(@NotNull Throwable throwable) {
        // Error handling
```

```
    }
});
```

单聊收发消息加解密

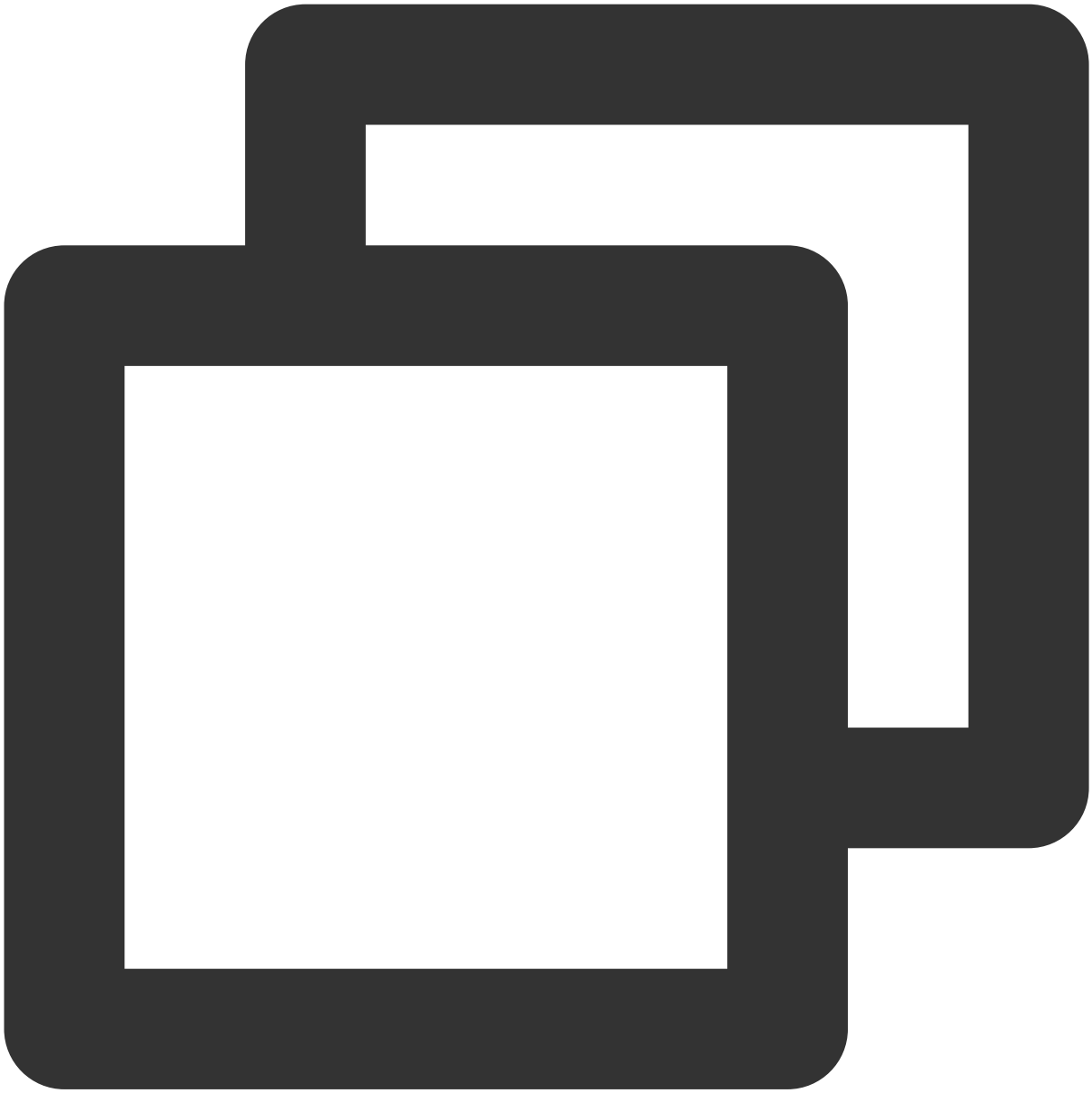
1.使用上面创建的channel进行消息加密，文档链接：https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/double-ratchet/?#encrypt-and-decrypt-messages



```
// 单聊消息加密
// prepare a message
String messageToEncrypt = "Hello, User2!";
```

```
String encrypted = channel.encrypt(messageToEncrypt);
```

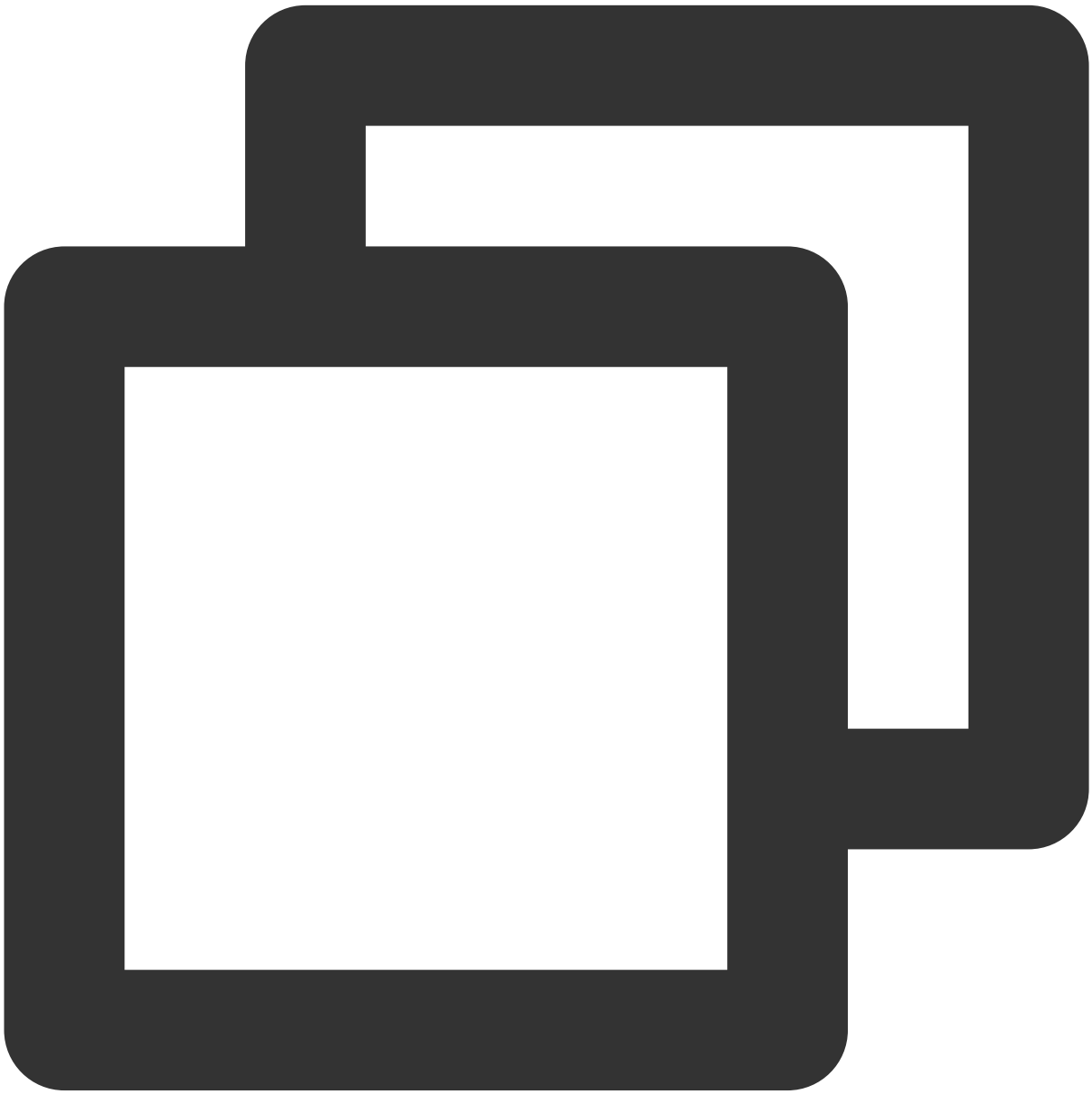
1. 加密后的消息内容传给腾讯云 im sdk，使用自定义消息发送



```
// API 返回 msgID，按需使用
String msgID = V2TIMManager.getInstance().sendC2CCustomMessage("virgil加密消息".getBy
    @Override
    public void onSuccess(V2TIMMessage message) {
        // 发送单聊文本消息成功
    }
```

```
    @Override
    public void onError(int code, String desc) {
        // 发送单聊文本消息失败
    }
});
```

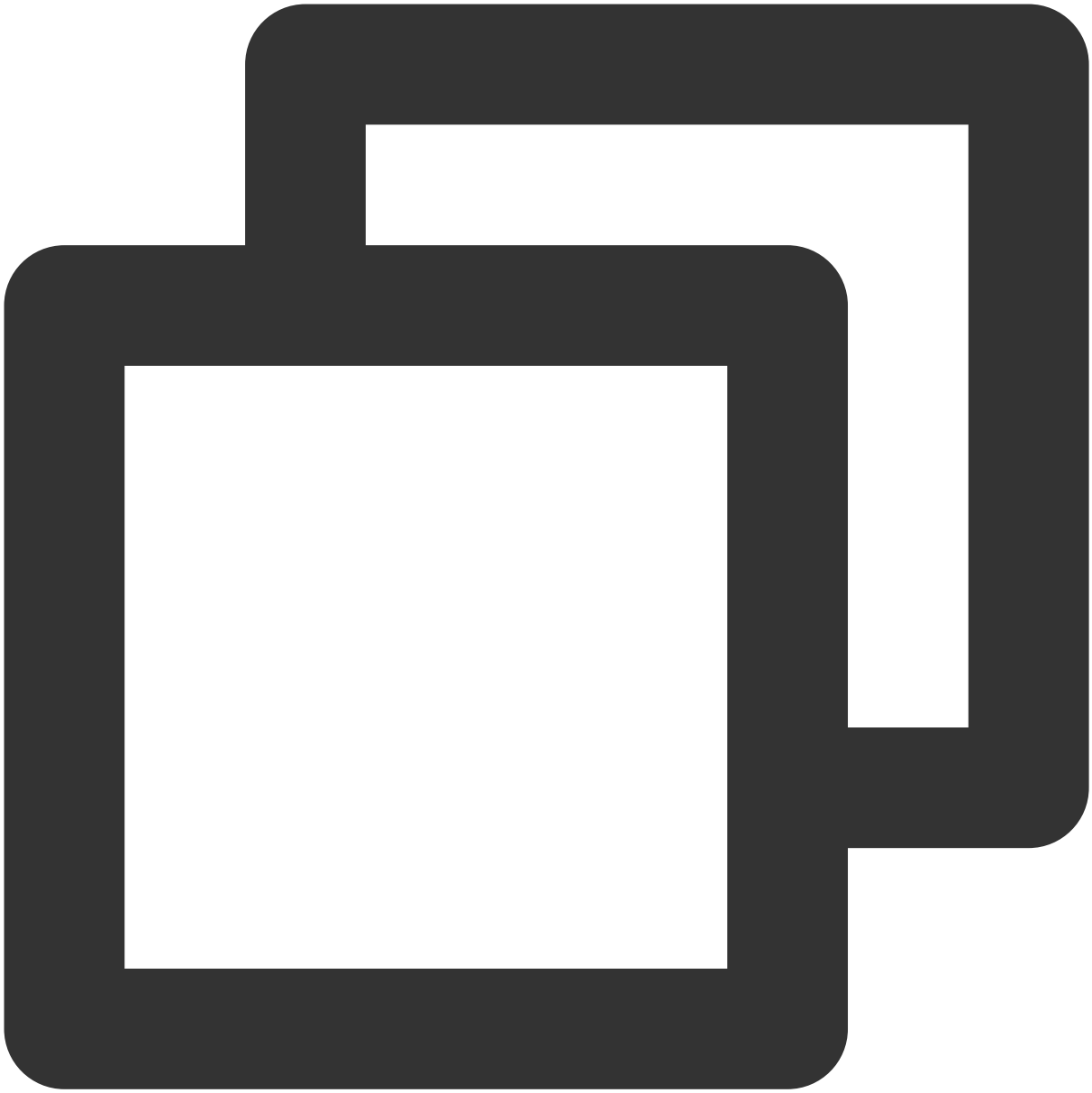
2. 对端腾讯云im sdk收到自定义消息后



```
// 设置事件监听器
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);

// 接收单聊自定义消息
```

```
/**
 * 收到 C2C 自定义消息
 *
 * @param msgID 消息唯一标识
 * @param sender 发送方信息
 * @param text 发送内容
 */
public void onRecvC2CCustomMessage(String msgID, V2TIMUserInfo sender, byte[] bEncr
    // 可解析消息并调用vilgil的channel.decrypt方法去解密
}
```

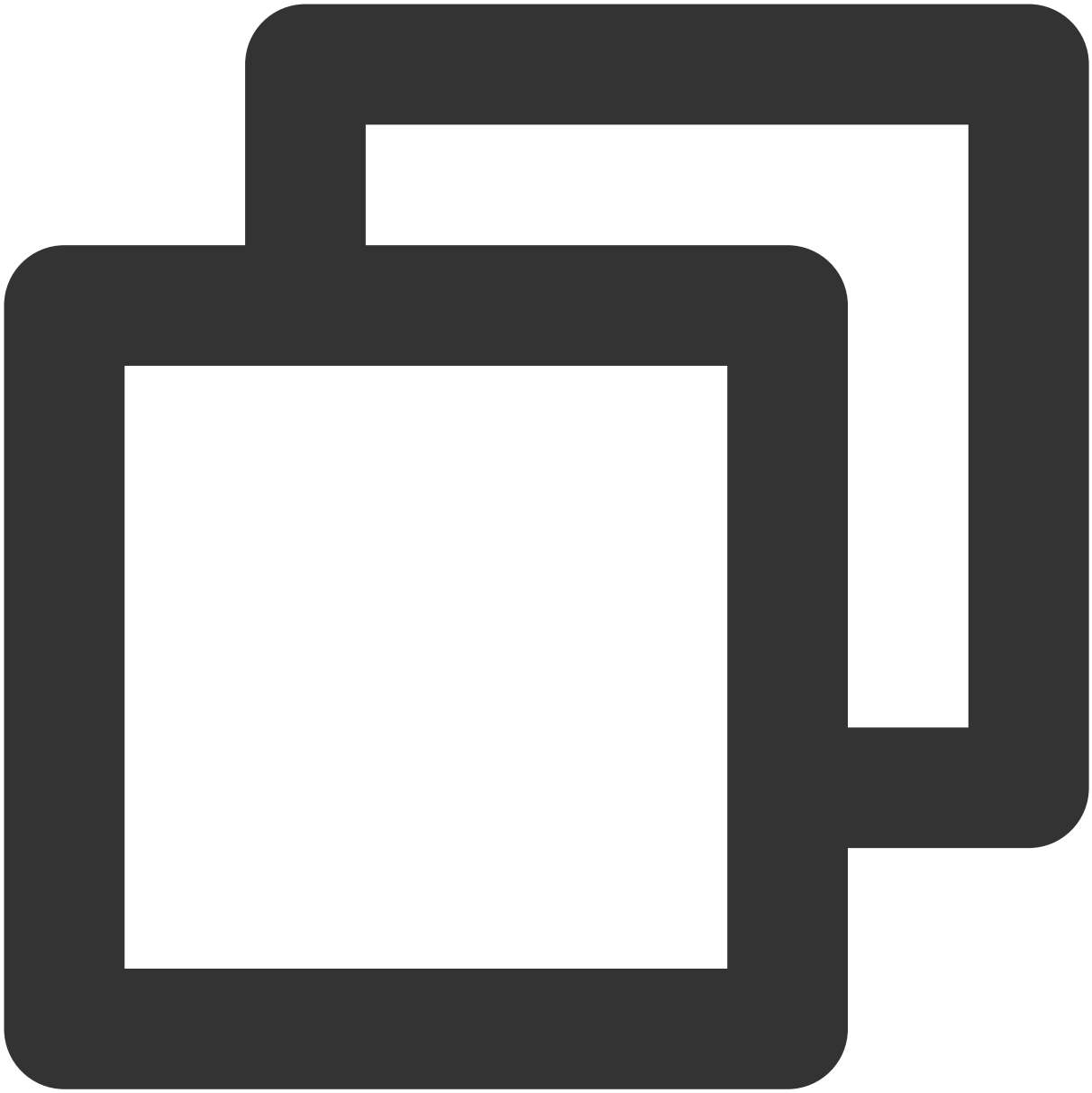
4.调用E3Kit解密后渲染出来，如下



```
// 解密消息
String decrypted = channel.decrypt(encrypted);
```

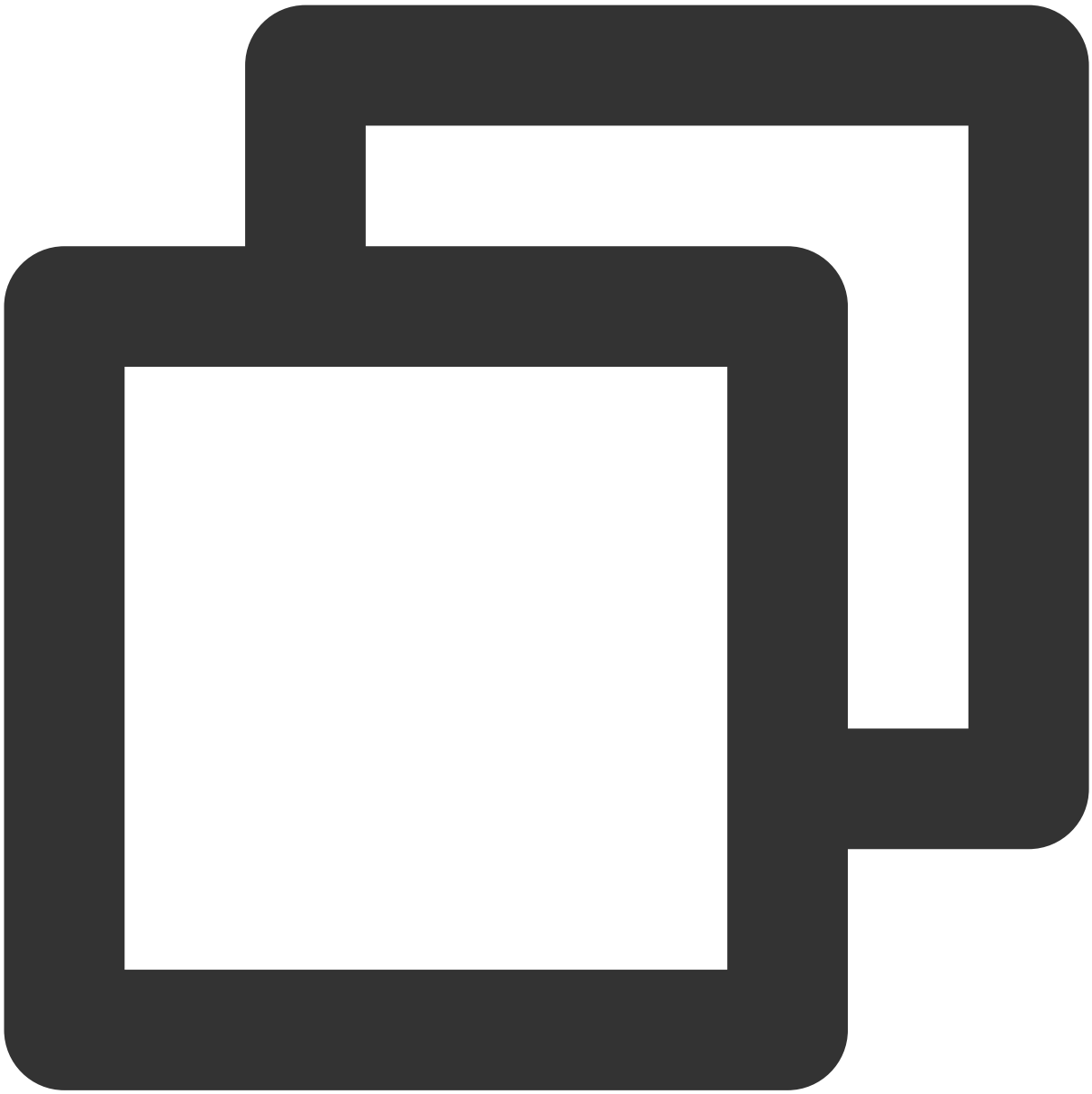
发起群聊会话

1.调用 ethree.createGroup 创建群，文档链接：https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#create-group-chat



```
// 建群
ethree.createGroup(groupId, users).addCallback(new OnResultListener<Group>() {
    @Override public void onSuccess(Group group) {
        // Group created and saved locally!
    }

    @Override public void onError(@NotNull Throwable throwable) {
        // Error handling
    }
});
```

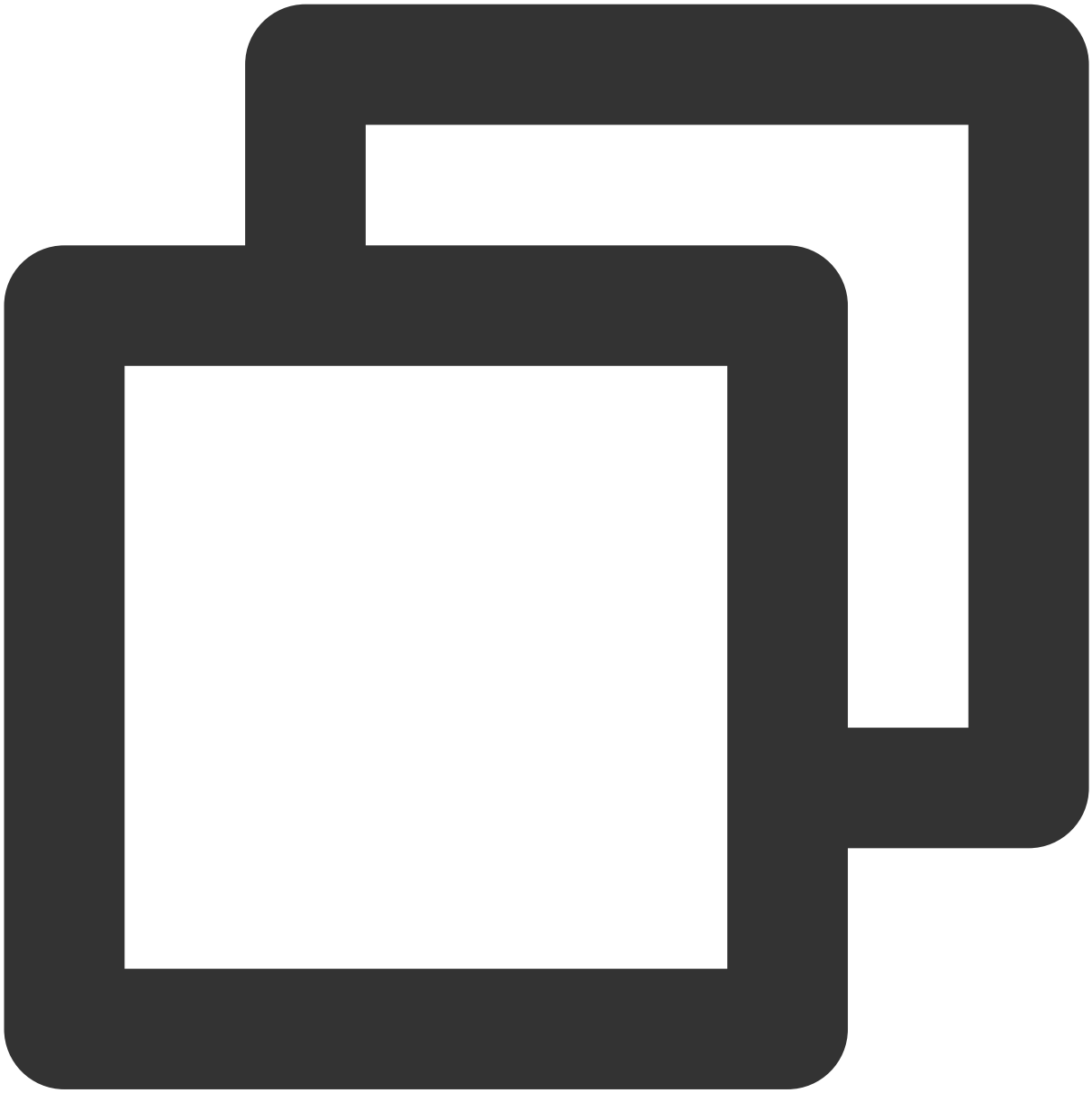
2.后续还需要添加群成员的话，如下代码。删除群成员可以调用remove接口。文档链接：
https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#add-new-participant



```
// 添加群成员
group.add(users.get("Den")).addCallback(new OnCompleteListener() {
    @Override public void onSuccess() {
        // Den was added!
    }

    @Override public void onError(@NotNull Throwable throwable) {
        // Error handling
```

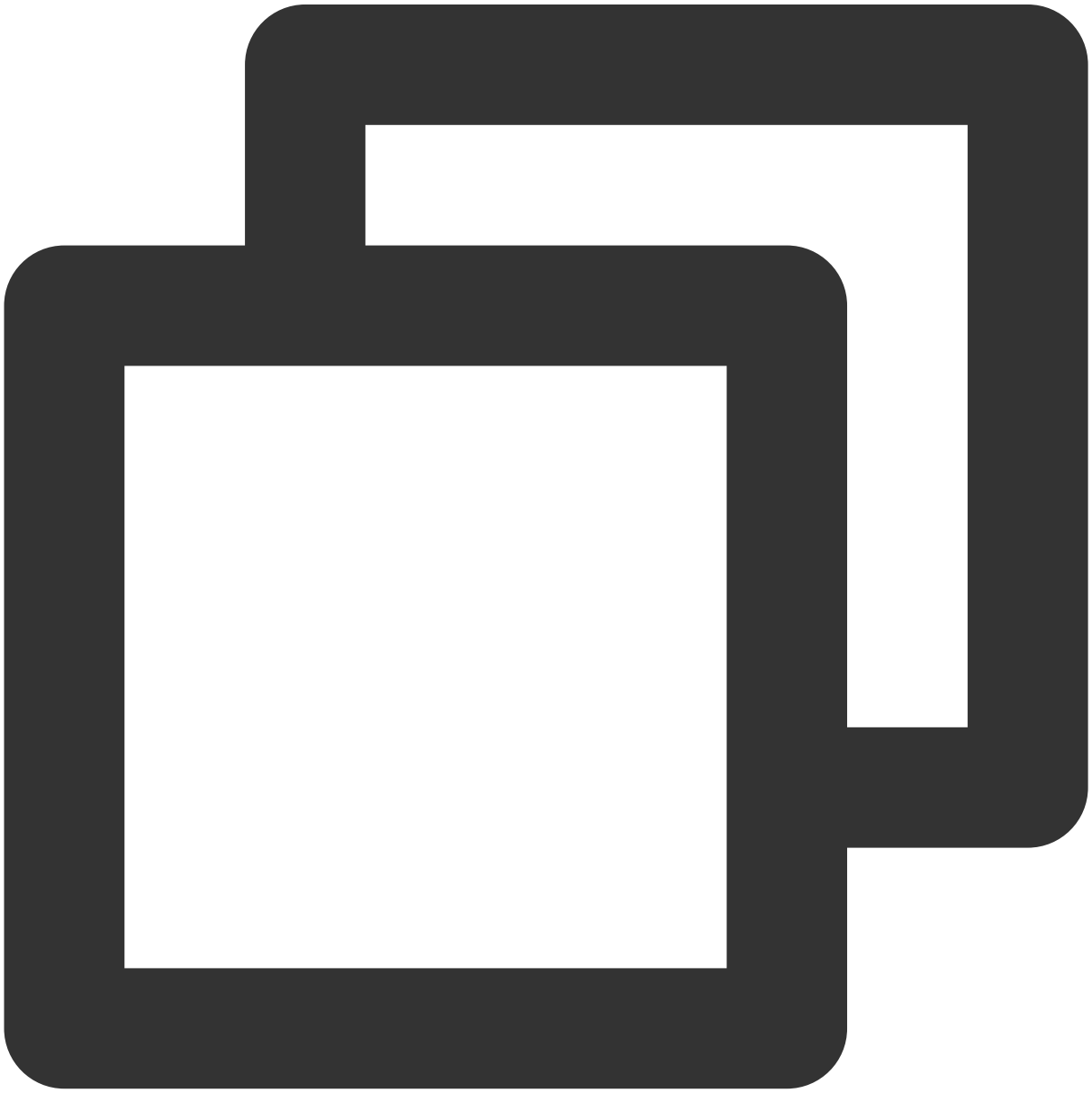
```
    }
});
```

1. 调用腾讯云im sdk的createGroup来创建群组，**joinGroup**或**inviteUserToGroup**进行群成员的新增



```
V2TIMManager.getInstance().createGroup(V2TIMManager.GROUP_TYPE_WORK, null, "groupA"
  @Override
  public void onSuccess(String s) {
      // 创建群组成功
  }

  @Override
```

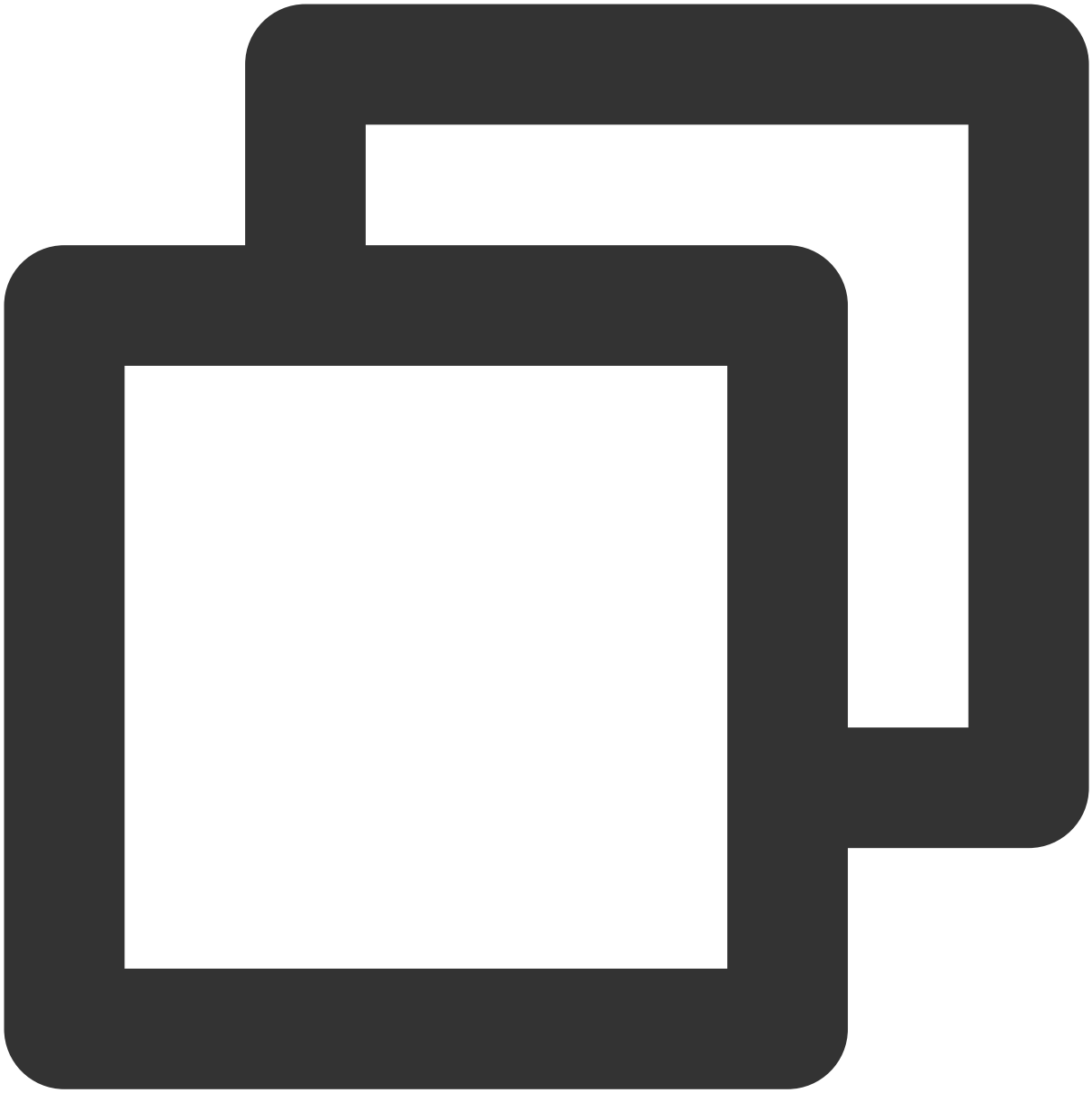
```
    public void onError(int code, String desc) {
        // 创建群组失败
    }
});
// 监听群组创建通知
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
    @Override
    public void onGroupCreated(String groupID) {
        // 群创建回调，groupID 为新创建群组的 ID
    }
});
```



```
// 邀请 userA 用户进入群组 groupA 中
List<String> userIDList = new ArrayList<>();
userIDList.add("userA");
V2TIMManager.getGroupManager().inviteUserToGroup("groupA", userIDList, new V2TIMVal
  @Override
  public void onSuccess(List<V2TIMGroupMemberOperationResult> v2TIMGroupMemberOpera
      // 邀请进群成功
  }

  @Override
  public void onError(int code, String desc) {
      // 邀请进群失败
  }
});

// 监听群组邀请事件
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
  @Override
  public void onMemberInvited(String groupID, V2TIMGroupMemberInfo opUser, List<V2T
      // 邀请进群事件。可以在这个回调中做一些 UI 上的提示
  }
});
```

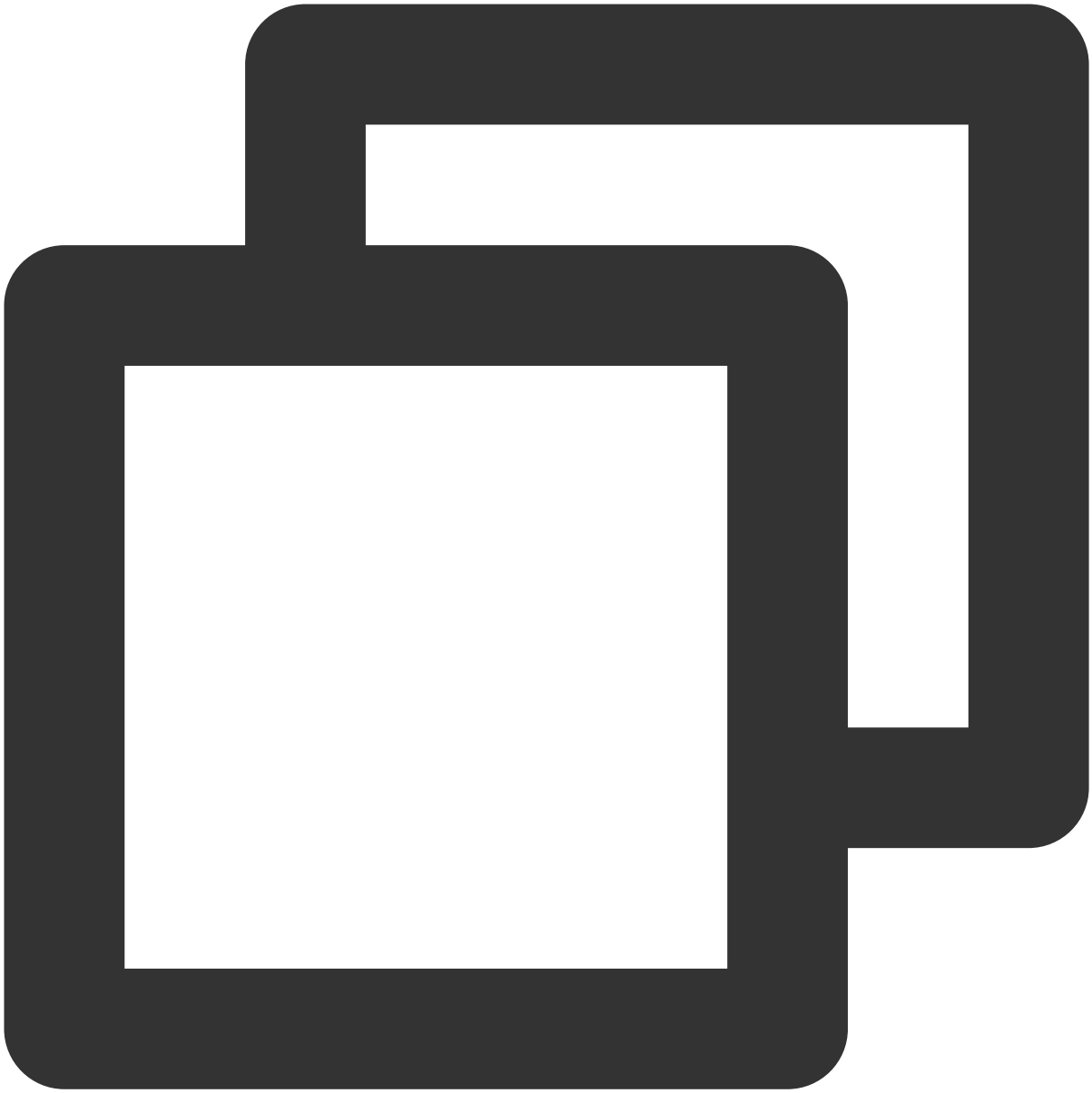
群聊收发消息加解密

1.使用上面创建的group进行消息加密，文档链接：https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#encrypt-and-decrypt-messages



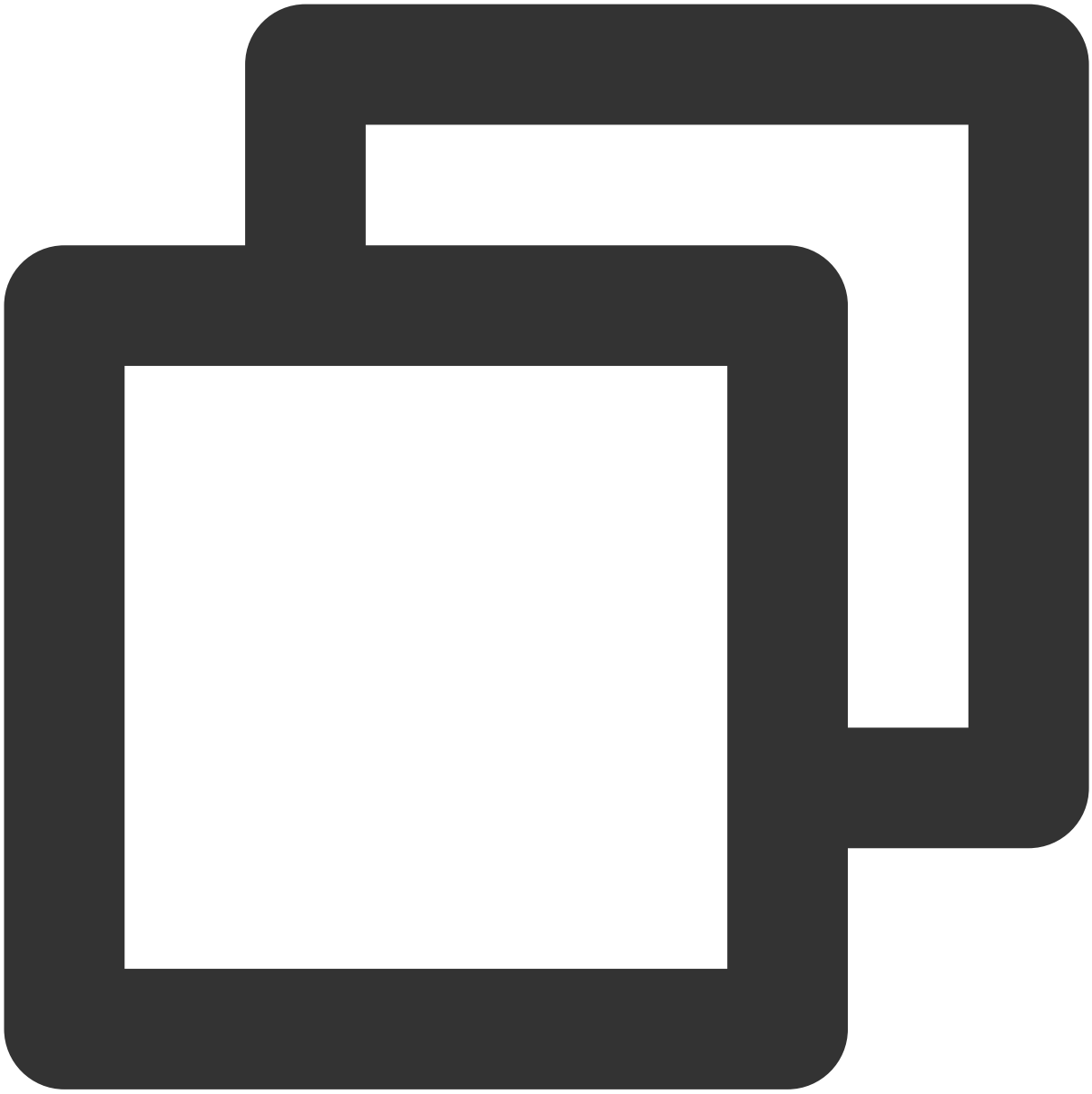
```
//群消息加密
// prepare a message
String messageToEncrypt = "Hello, Bob and Carol!";

String encrypted = group.encrypt(messageToEncrypt);
```

1. 加密后的消息内容传给腾讯云im sdk，使用自定义消息发送



```
String msgID = V2TIMManager.getInstance().sendGroupCustomMessage("virgil加密消息".get
    @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
    }
});
```

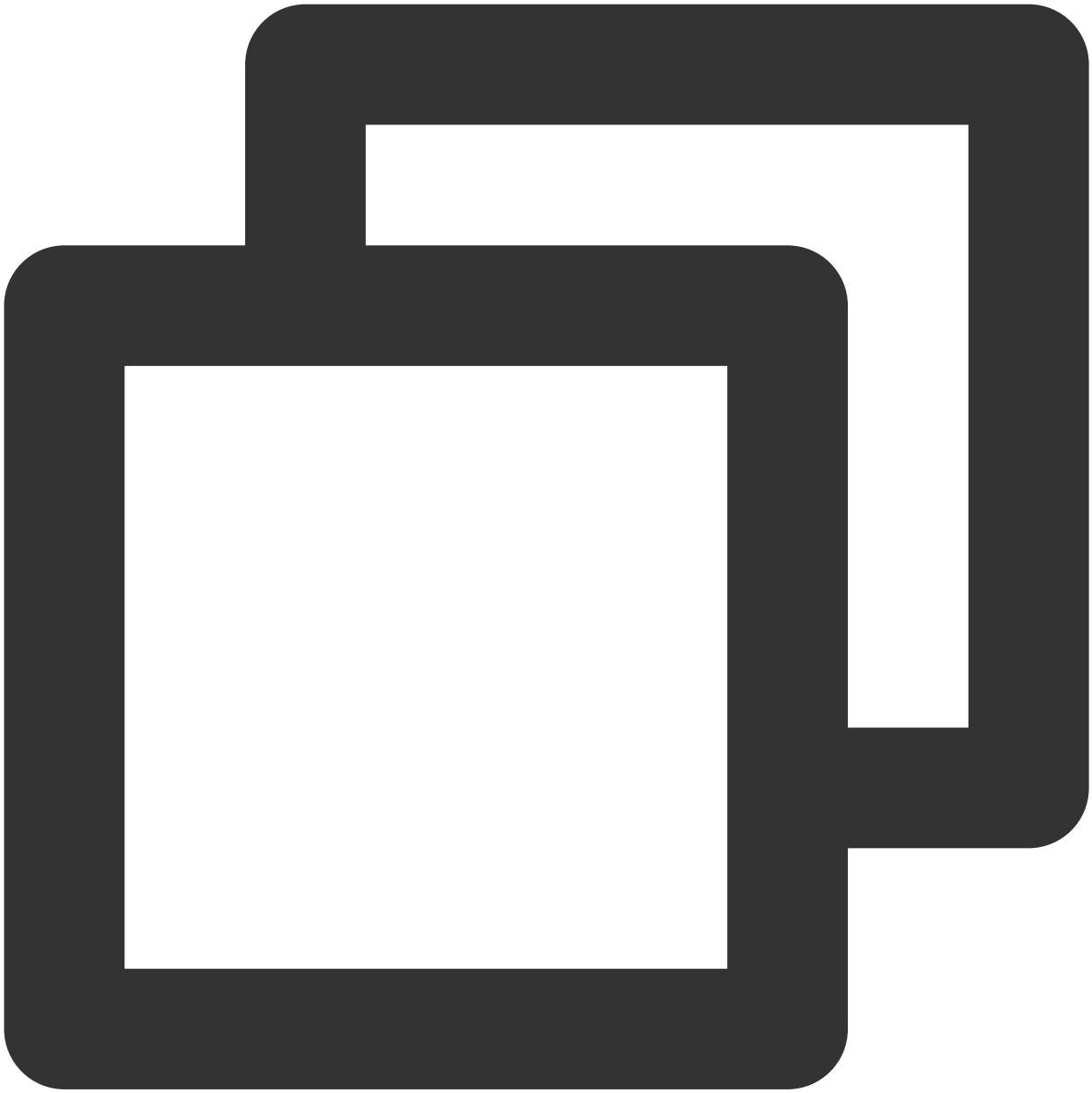
3.对端腾讯云im sdk收到群自定义消息



```
/**
 * 接收群聊自定义消息
 * @param msgID 消息 ID
 * @param groupID 群 ID
 * @param sender 发送方群成员信息
 * @param customData 发送内容
 */
public void onRecvGroupCustomMessage(String msgID, String groupID, V2TIMGroupMember
    Log.i("onRecvGroupCustomMessage", "msgID:" + msgID + ", groupID:" + groupID + "
```

```
    //可解析消息并调用vilgil的group.decrypt方法去解密
}
```

4.调用E3Kit解密后渲染出来，如下

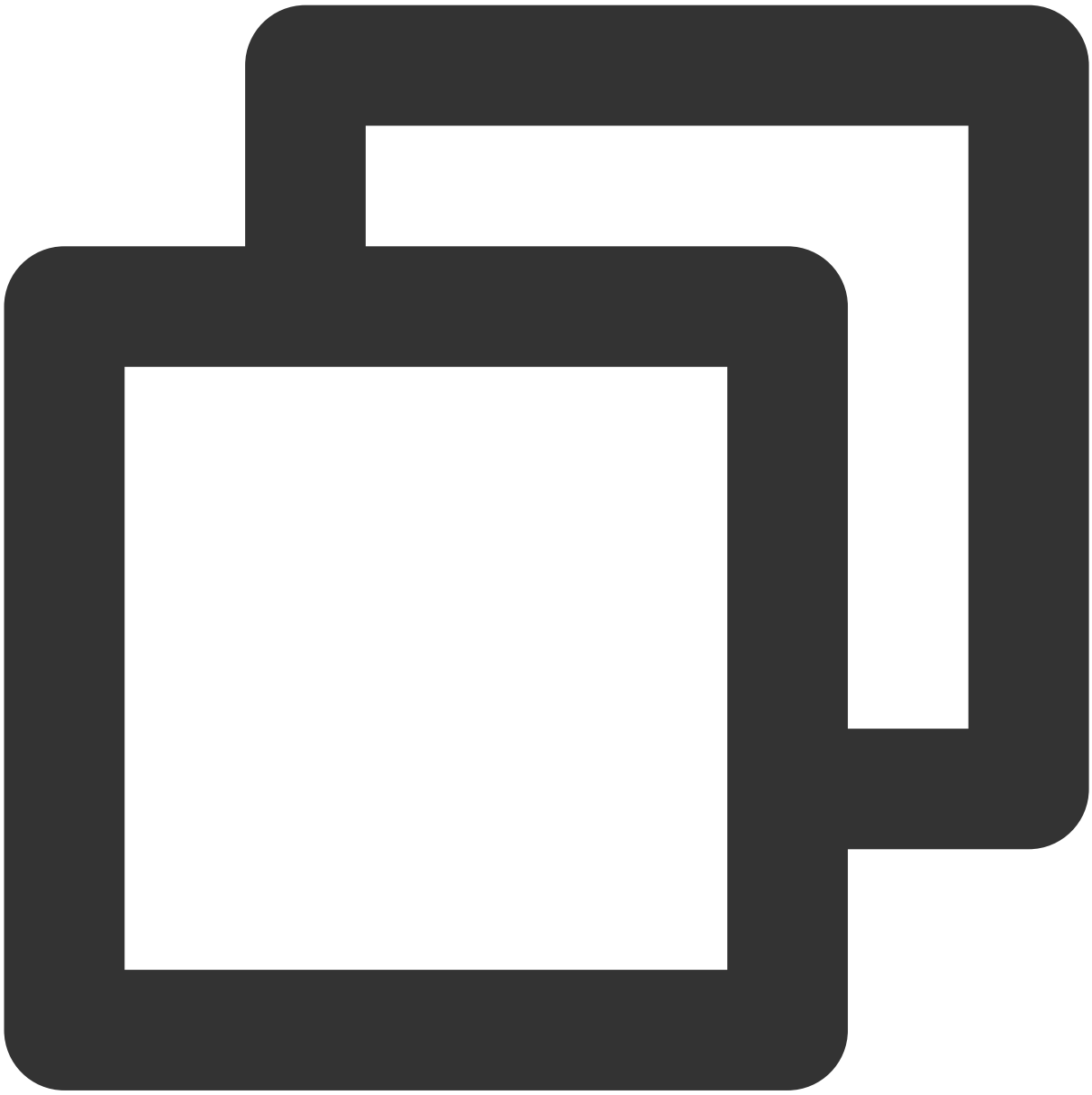


```
//群消息解密
String decrypted = group.decrypt(encrypted, users.get("Alice"));
```

注意：使用该端到端加密方案后，im sdk的本地聊天记录搜索功能将不可用

即时通信 IM - 交流群

加入腾讯云即时通信 IM 技术交流群，您将获得：



- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

Telegram交流群：点击加入

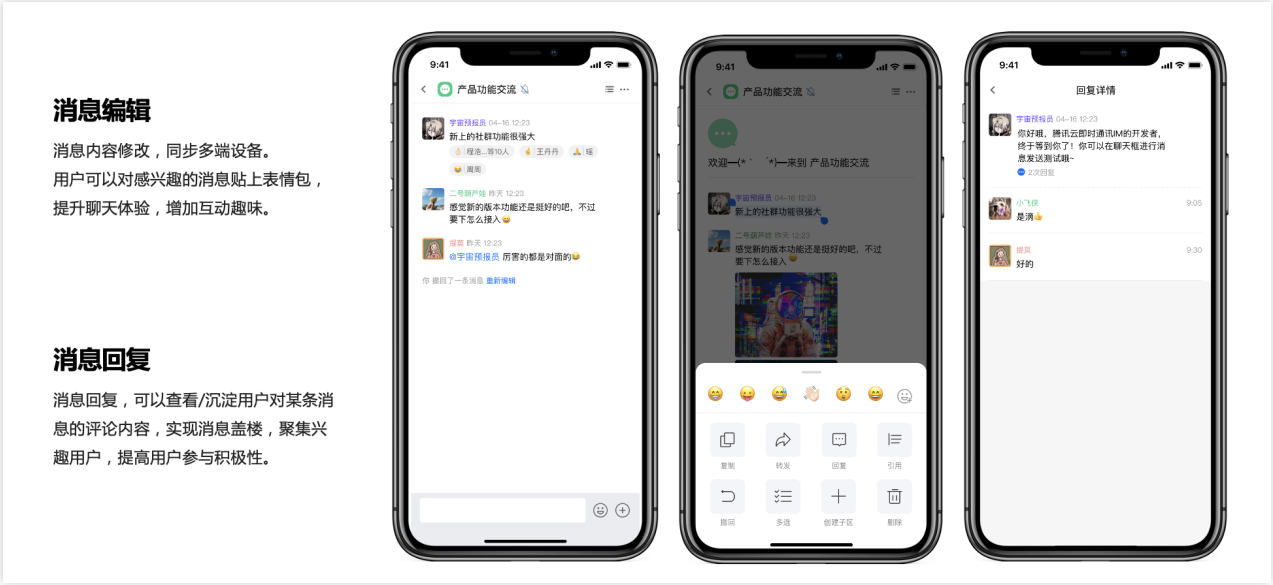
WhatsApp交流群：点击加入

超大娱乐协作社群

最近更新时间：2024-02-19 11:21:10

功能介绍

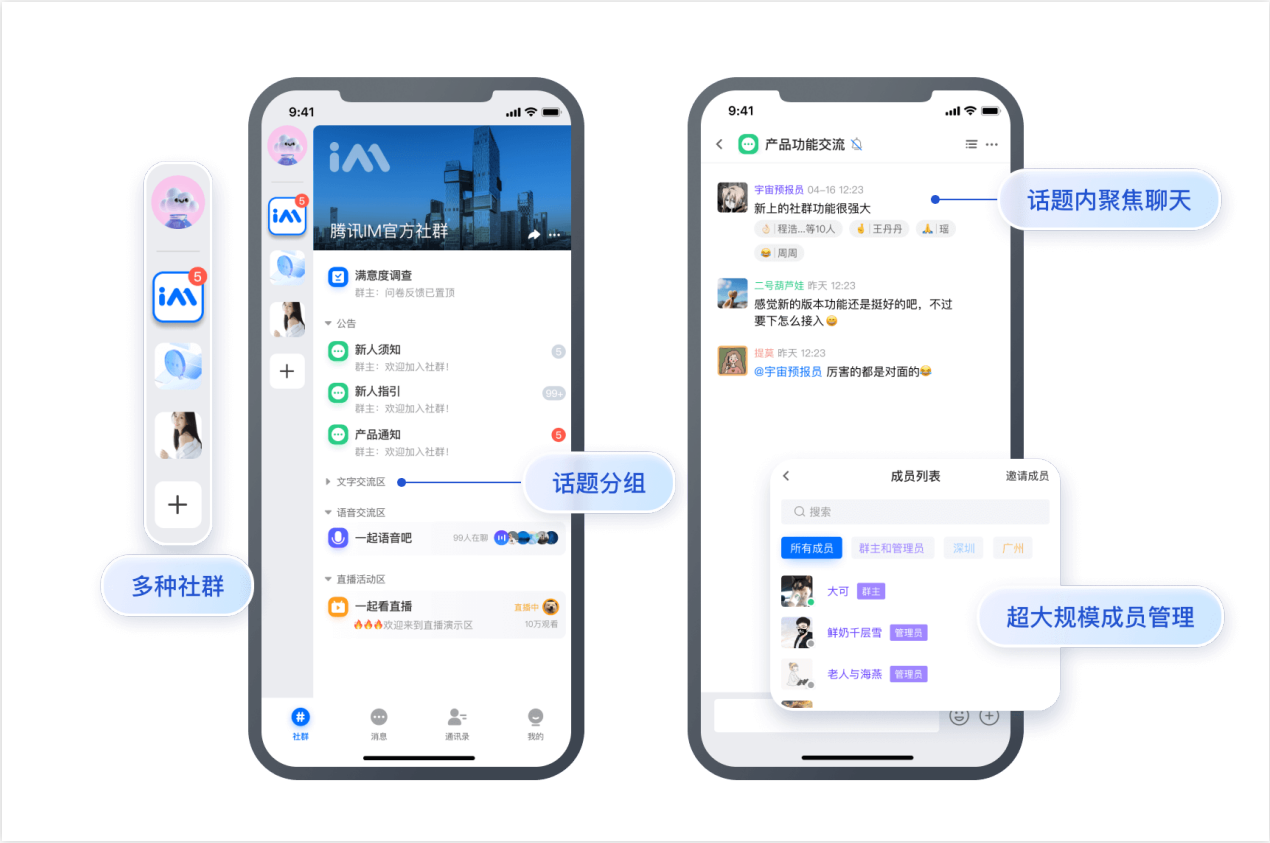
社群模式（娱乐协作新利器），既支持**社群-分组-话题**三级结构，将消息相互区隔，又可运营超大规模成员，共用一套好友关系，还可将成员分组，设置成员组的查看、发言、管理等权限。



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，可完美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。

技术优势

超大规模群成员

腾讯 IM 社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。

终端 SDK 集成指南

注意：

社群（Community）话题（Topic）功能仅 IM 终端 SDK 6.2.2363 增强版及以上版本支持，需 购买旗舰版 并 申请开通 后方可使用。

下文以 Android 为例，介绍社群话题的接口功能。

下载 Demo 快速体验社群功能

请使用 Android 手机扫描下图二维码，下载 Demo 后可直接体验。

下面从 API 调用的角度来介绍下社群话题的使用：

社群话题的 API 使用

1. 首先调用接口 createGroup 创建支持话题的社群，可以参见社群管理 “创建社群”步骤来实现。
2. 然后通过调用 getJoinedCommunityList 接口，可以获取到创建和加入的该类社群列表。

注意：

社群用来管理群成员，但不可以在社群中收发消息。社群的其他功能可以参考“其他管理接口”列表。

3. 成功创建社群后，可以在社群下调用 createTopicInfoCommunity 创建多个话题，参见话题管理 中“创建话题”步骤实现。
4. 话题的管理还包括“删除话题”、“修改话题信息”、“获取话题列表”、“监听话题回调”功能。同时，话题也是可以供用户交流的地方，因此可以进行收发消息，相关接口可以参见话题消息介绍。
5. 社群成员分组，可通过把分组信息设置到群成员自定义字段里，实现按照分组展示的效果。这需要拉取所有群成员到本地按照分组排序，若群人数较多，建议您在服务端自行实现。

Web SDK 集成指南

注意：
社群（Community）话题（Topic）功能仅 IM Web SDK v2.19.1 及以上版本支持，需 购买旗舰版 并在 **控制台>群功能配置>社群** 打开开关后方可使用。
社群话题的接口功能如下：

下载并配置 Demo 源码快速体验

您可参见 快速入门，快速体验 IM 的功能。
下面从 API 调用的角度来介绍下社群话题的使用：

社群话题的 API 使用

1. 首先调用接口 createGroup 创建支持话题的社群，可以参见 社群管理 “创建社群”步骤来实现。
2. 然后通过调用 getJoinedCommunityList 接口，可以获取到创建和加入的该类社群列表。社群用来管理群成员，但不可以在支持话题的社群中收发消息。社群的其他功能可以参考普通群组API。
3. 成功创建社群后，可以在社群下调用 createTopicInfoCommunity 创建多个话题，参见 话题管理 “创建话题”步骤实现。
4. 话题的管理还包括“删除话题”、“修改话题信息”、“获取话题列表”、“监听话题回调”功能。同时，话题也是可以供用户交流的地方，因此可以进行收发消息，相关接口可以参见 创建消息 介绍。
5. 社群成员分组，可通过把分组信息设置到群成员自定义字段里，实现按照分组展示的效果。这需要拉取所有群成员到本地按照分组排序，若群人数较多，建议您在服务端自行实现。

相关文档

群组管理
群组系统
SDK 下载
SDK 手册
集成 SDK（Android）

集成 SDK（iOS）
集成 SDK（Web & 小程序 & uni-app）

联系我们

如果您有任何使用问题，请 联系我们。

How to integrate Tencent IM with Salesforce

最近更新时间：2024-02-07 17:30:51

Introduction

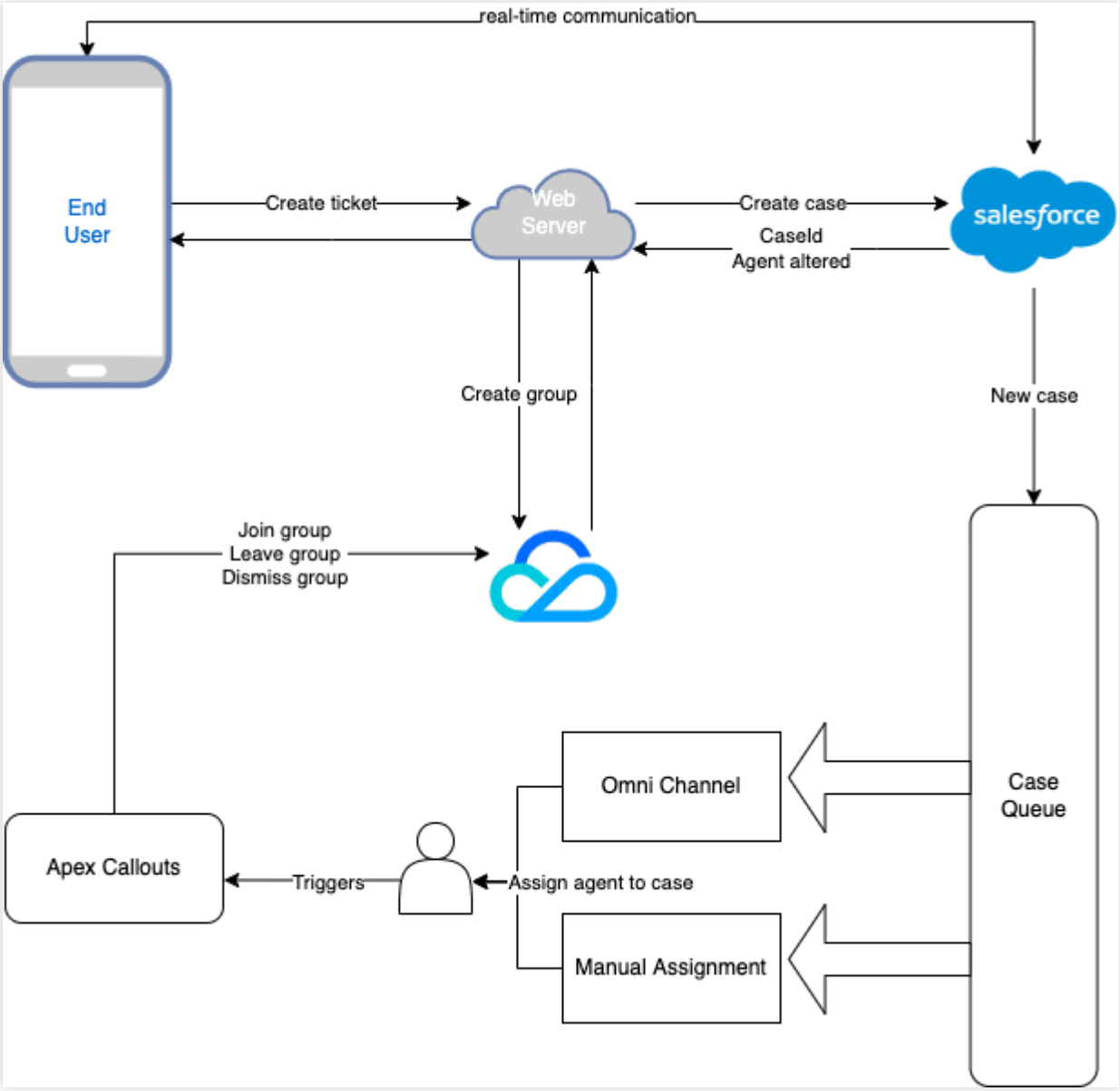
This tutorial aims to demonstrate an approach to integrate the Tencent Cloud IM SDK into Salesforce's workflow to leverage the communication between a Salesforce agent and an end user.

Preliminary

1. Sign up for Tencent Cloud, register IM service and create an app, see guidence.
2. Sign up for a Salesforce developer account in case you don't have, click here.

Road Map

Three parts are essential to achieve the goal.
1. An end user application for end user to start an conversation.
2. An online server for creating a Salesforce case and creating an Tencent Cloud IM chat group. Also provide APIs for invite/delete Salesforce Agent to the group and dismiss the group when case is closed.
3. An custom Salesforce utilities component for in-Salesforce communication.
Here's the integration map:



End User Application

Tencent Cloud IM provides a variety of SDKs for the most popular platforms, and you can simply choose the one for your platform, check out our SDKs here: Android, iOS, Web, Flutter, Windows, Unity, Unreal Engine. The recommended way to build your application from scratch is to utilize our TUIKit to layer up the interface, see here: Android, iOS, Web, Flutter.

Online Server Relay

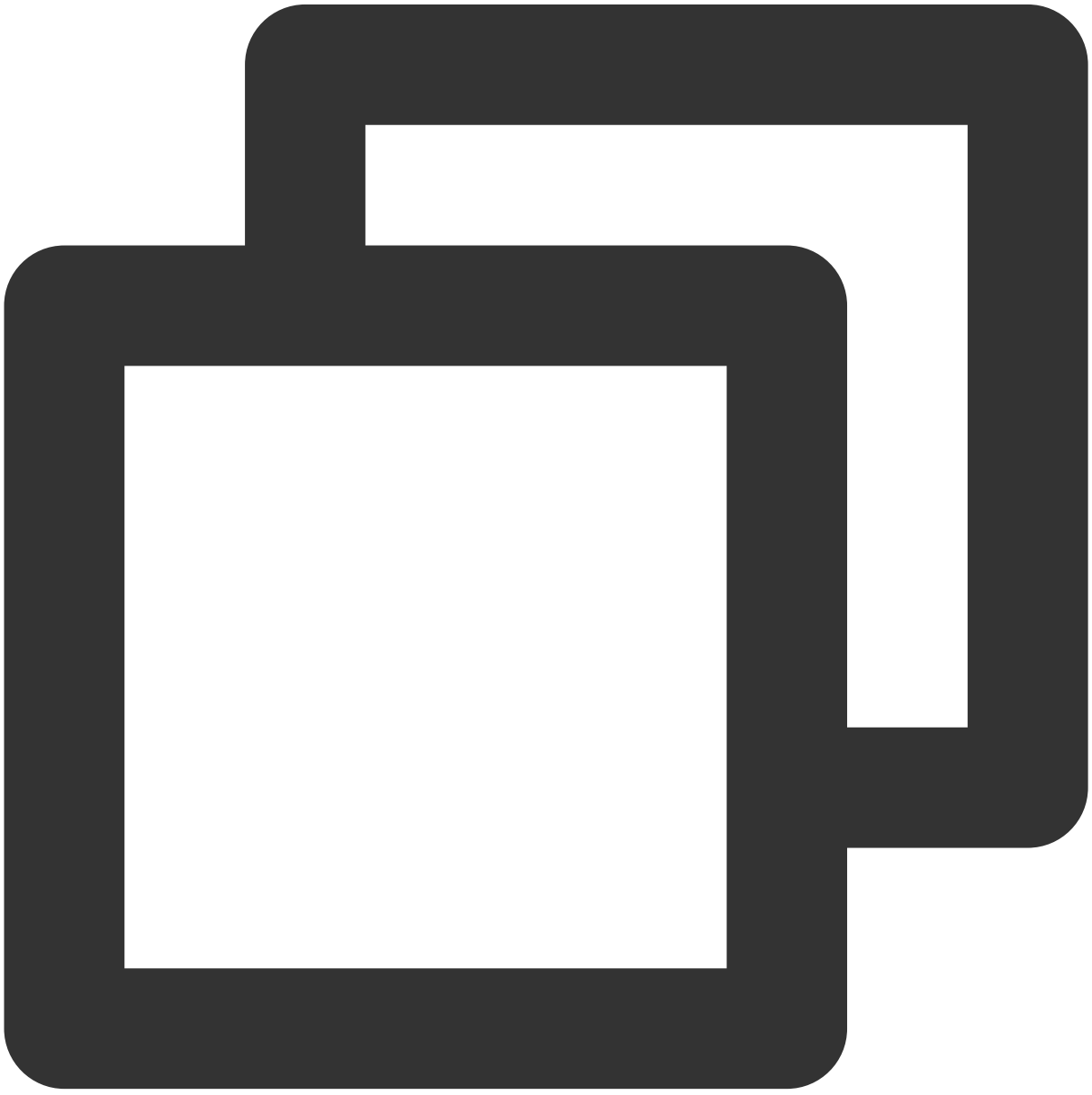
In here, we will guide you step by step to create a web server to allow an end user to submit a Salesforce case and create an Tencent Cloud IM chat group, also allow Salesforce agents to be invited/removed from Tencent Cloud IM chat group and also delete the group when case is closed.

Agent Chat Interface

This tutorial will give you the instructions to create an chat interface in Salesforce and invite agent into the Tencent Cloud IM chat group. Also, you can use our Web UIKit to build this plug-in widget.

Step 1. Create an online server

The online server is for the purpose to connect Salesforce and Tencent Cloud IM, and enables the end user to create a Salesforce case and create an corresponding Tencent Cloud IM chat group.
The example Node server is shown below:



```
const express = require("express")
const axios = require("axios")
var TLSSigAPIv2 = require("tls-sig-api-v2") // Generate UserSig for Tencent Cloud I
const sf = require("node-salesforce") // Salesforce API Connection Library for Node

const YOUR_SDKAPPID = 1400000000
const YOUR_SECRET = ""
const ADMIN_USERID = ""

const app = express()
app.use(express.json())

const port = process.env.PORT || 3000

app.use(express.json())

app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "https://YOUR_DOMAIN")
  res.header("Access-Control-Allow-Headers", "*")
  next()
})

// End user calls /createticket to create a Salesforce case and a Tencent Cloud IM
app.post("/createticket", async (req, res) => {
  const { userId, caseInfo } = req.body
  if (!userId) return res.status(500).send("Missing userId")

  const auth = await getSalesforceAccessToken()
  if (auth.error) return res.status(500).send("Salesforce auth error")

  const salesforceCase = await createCase(auth.token, caseInfo)
  if (!salesforceCase.success)
    return res.status(500).send("Case creation failed")

  const groupName = salesforceCase.id
  const result = await createGroup(groupName, userId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Group creation failed")

  res.status(200).send(result)
})

// When detect a new agent assigned to the group, Salesforce sends a request to joi
app.post("/joingroup", async (req, res) => {
  const { groupId, userId } = req.body
  if (!userId) return res.status(500).send("Missing userId")
```

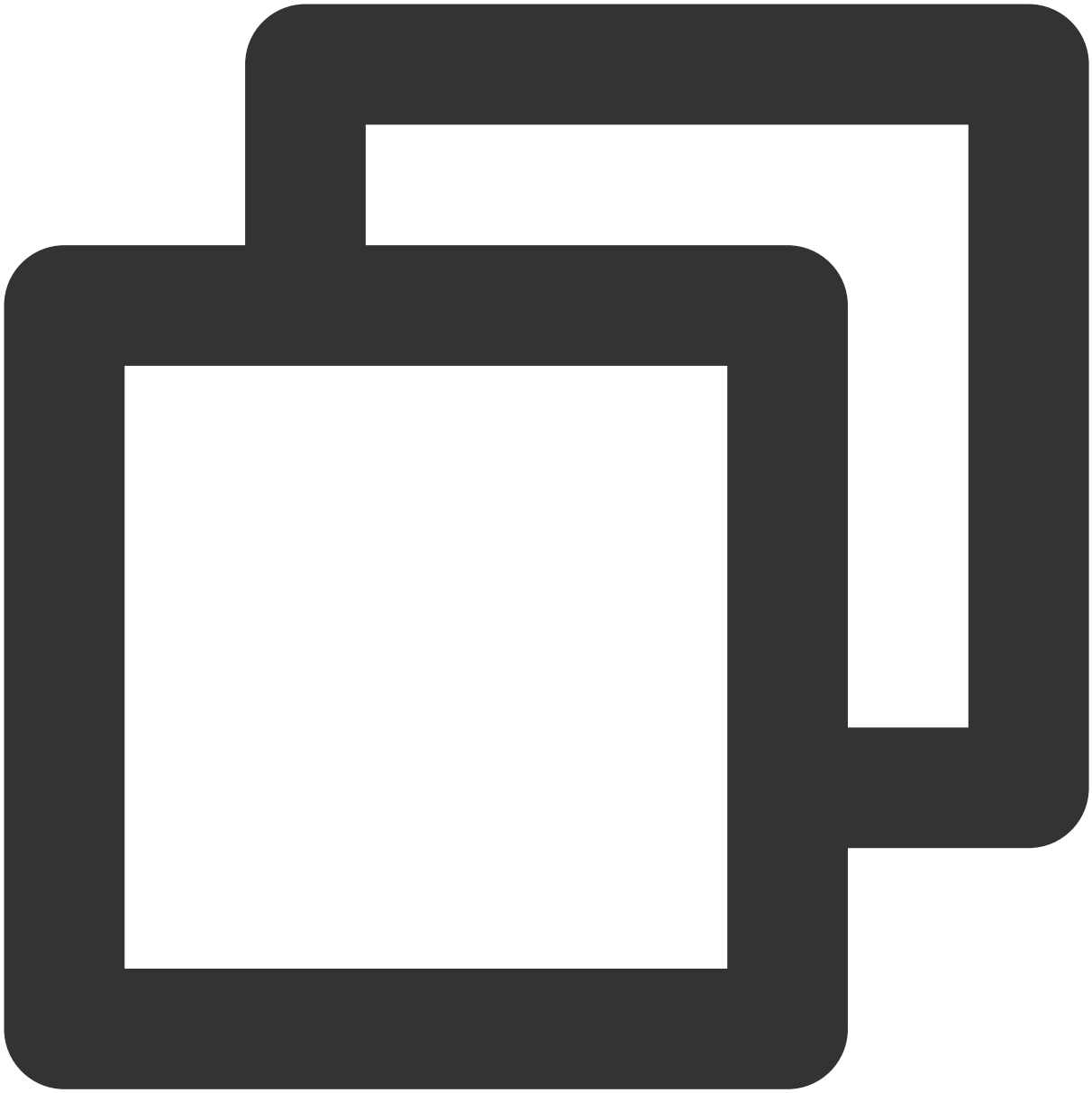
```
  if (!groupId) return res.status(500).send("Missing groupId")

  const result = await joinGroup(groupId, userId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Join group failed")

  res.status(200).send(result)
})

// When detect an agent removed from the group, Salesforce sends a request to remov
app.post("/leavegroup", async (req, res) => {
  const { groupId, userId } = req.body
  if (!userId) return res.status(500).send("Missing userId")
  if (!groupId) return res.status(500).send("Missing groupId")

  const result = await leaveGroup(groupId, userId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Leave group failed")

  res.status(200).send(result)
})

app.post("/deletegroup", async (req, res) => {
  const { groupId } = req.body
  if (!groupId) return res.status(500).send("Missing groupId")

  const result = await deleteGroup(groupId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Delete group failed")

  res.status(200).send(result)
})

const getSalesforceAccessToken = async function () {
  const url = "https://{your_instance}.salesforce.com"
  const conn = new sf.Connection({ loginUrl: url })
  try {
    await conn.login("SF_EMAIL", "SF_PASSWORDSF_TOKEN")
    return { error: undefined, token: conn.accessToken }
  } catch (e) {
    return { error: e, token: undefined }
  }
}

const createCase = async function (token, caseInfo) {
  const { subject, desc, name, email } = caseInfo
  const body = {
```

```
    Subject: subject,
    Description: desc,
    SuppliedName: name,
    SuppliedEmail: email,
  }
  const headers = {
    headers: {
      "Content-Type": "application/json",
      Authorization: "Bearer " + token,
    },
  }
  const url =
    "https://{your_instance}.salesforce.com/services/data/v{api_version}/sobjects/C
  try {
    const result = await axios.post(url, body, headers)
    return result.data
  } catch (e) {
    return { id: undefined, success: false, error: e }
  }
}

const generateUserSig = function () {
  const expires = 600
  const api = new TLSSigAPIv2.Api(YOUR_SDKAPPID, YOUR_SECRET)
  return api.genSig(ADMIN_USERID, expires)
}

const generateRandom = function () {
  return Math.floor(Math.random() * 4294967295)
}

const createGroup = async function (groupName, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  // Use salesforceCase.id as group ID
  const data = { Owner_Account: userId, Type: "Public", Name: groupName, GroupId: g
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

const joinGroup = async function (groupId, userId) {
  const sig = generateUserSig()
```

```
  const random = generateRandom()
  const data = { GroupId: groupId, MemberList: [{ Member_Account: userId }] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/add_group_member?s
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

const leaveGroup = async function (groupId, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId, MemberToDel_Account: [userId] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/delete_group_membe
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

const deleteGroup = async function (groupId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/destroy_group?sdka
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

app.listen(process.env.PORT || port, () =>
  console.log(`Example app listening on port ${port}!`)
)
```

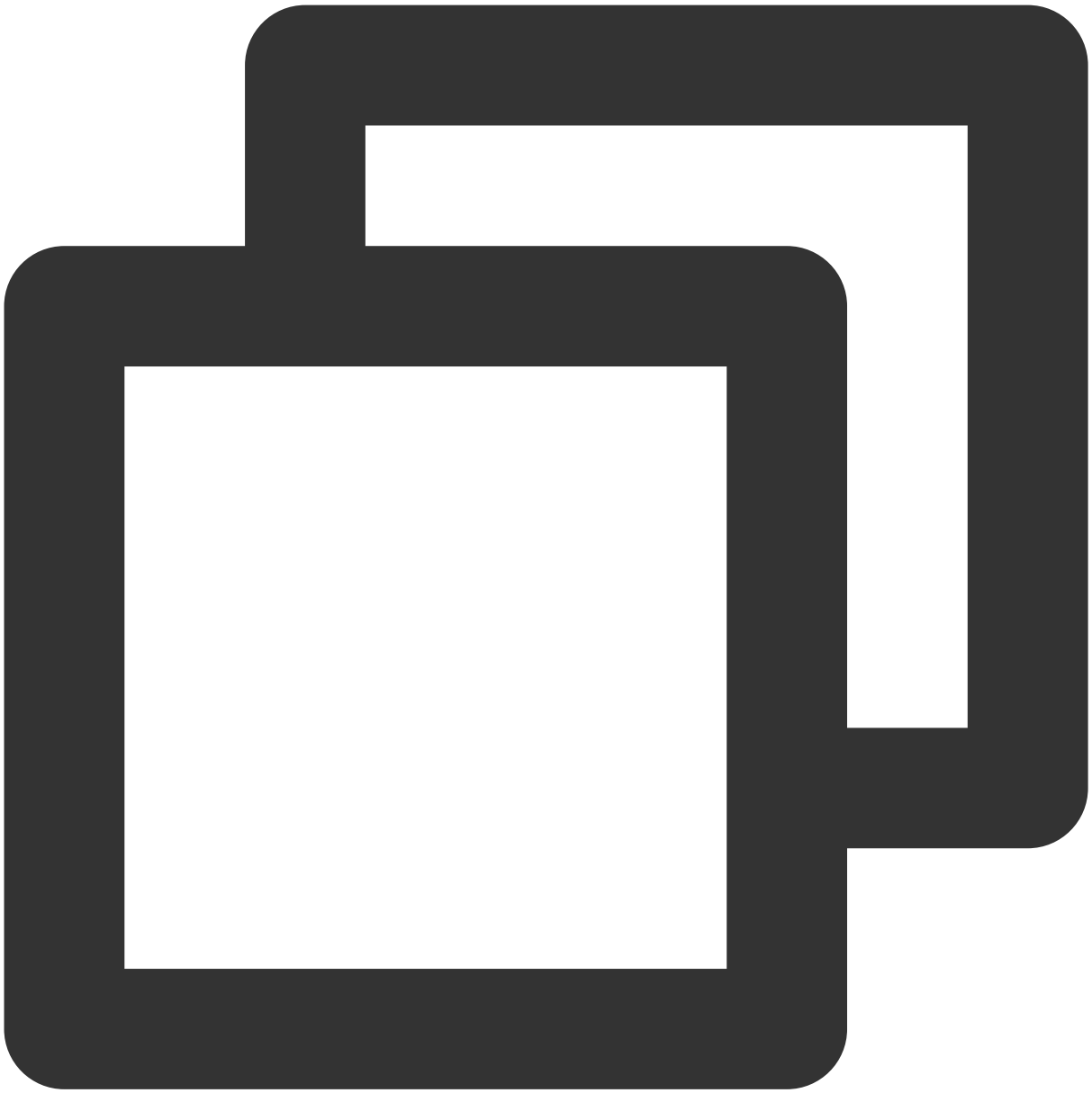
The server supports a route /createticket for the end user to create a case in Salesforce and a chat group in Tencent Cloud IM. Here's what we do here:

1. Fisrt we fetch an accessToken from Salesforce, more about SF_TOKEN.



```
const getSalesforceAccessToken = async function () {
  const url = "https://{your_instance}.salesforce.com"
  const conn = new sf.Connection({ loginUrl: url })
  try {
 await conn.login("SF_EMAIL", "SF_PASSWORDSF_TOKEN")
 return { error: undefined, token: conn.accessToken }
  } catch (e) {
 return { error: e, token: undefined }
  }
}
```

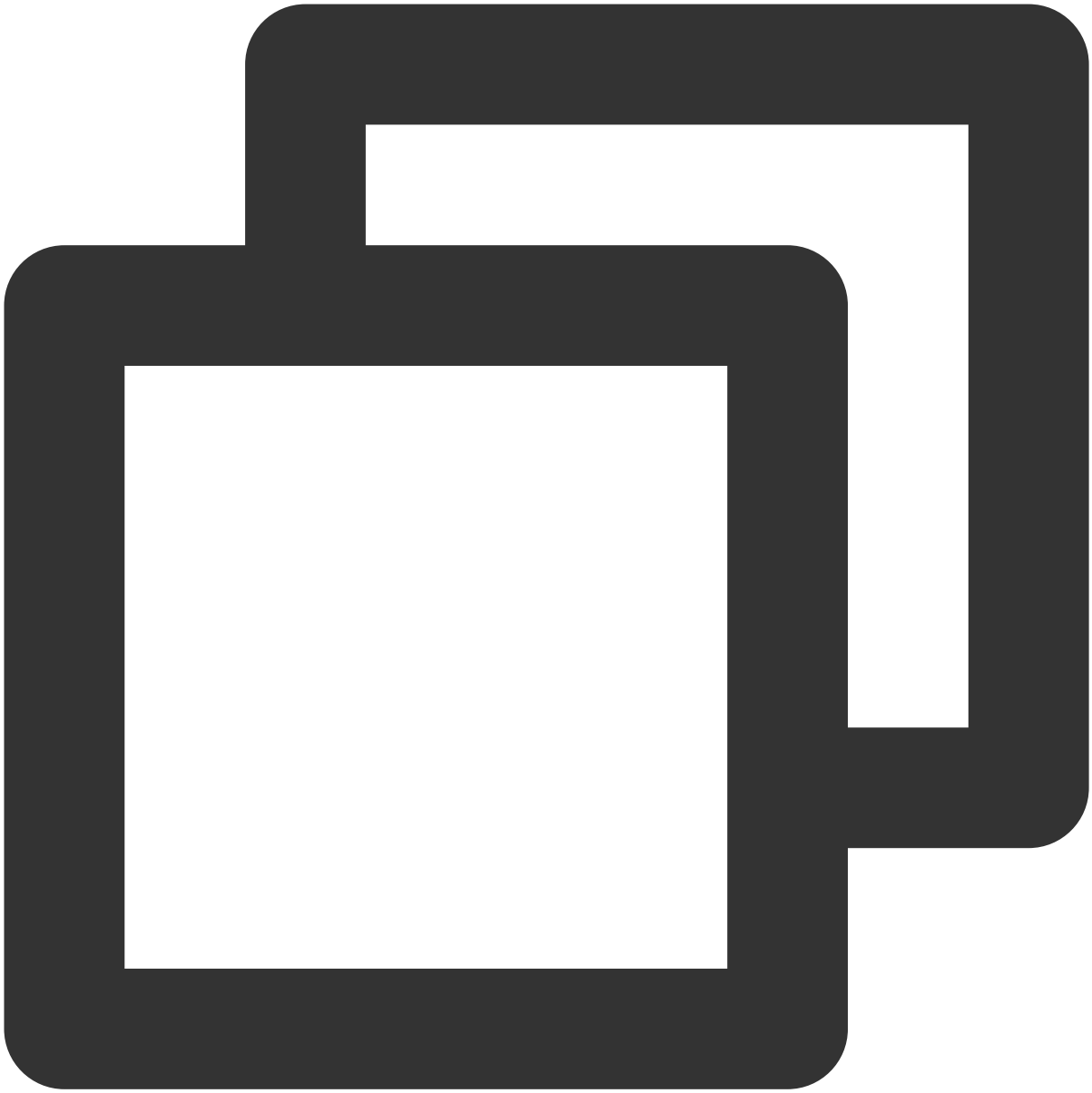
2. Create a Salesforce case.



```
const createCase = async function (token, caseInfo) {
  const { subject, desc, name, email } = caseInfo
  const body = {
 Subject: subject,
 Description: desc,
 SuppliedName: name,
 SuppliedEmail: email,
  }
  const headers = {
```

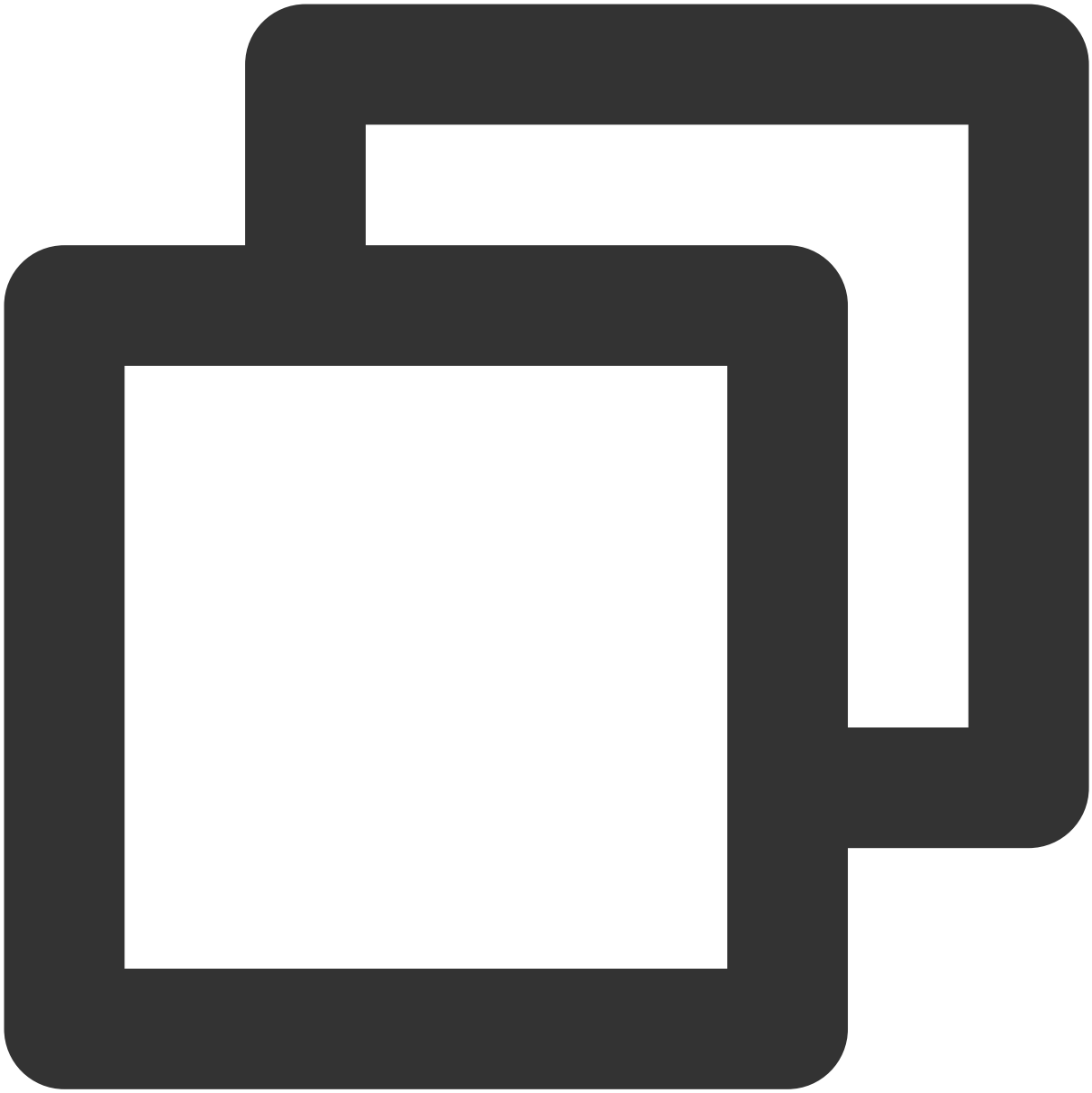
```
  headers: {
    "Content-Type": "application/json",
    Authorization: "Bearer " + token,
  },
   }
   const url =
  "https://{your_instance}.salesforce.com/services/data/v{api_version}/sobjects/Case
   try {
  const result = await axios.post(url, body, headers)
  return result.data
   } catch (e) {
  return { id: undefined, success: false, error: e }
   }
}
```

3. Generate UserSig for Tencent Cloud IM



```
 const generateUserSig = function () {
  const expires = 600
  const api = new TLSSigAPIv2.Api(YOUR_SDKAPPID, YOUR_SECRET)
  return api.genSig(ADMIN_USERID, expires)
}
```

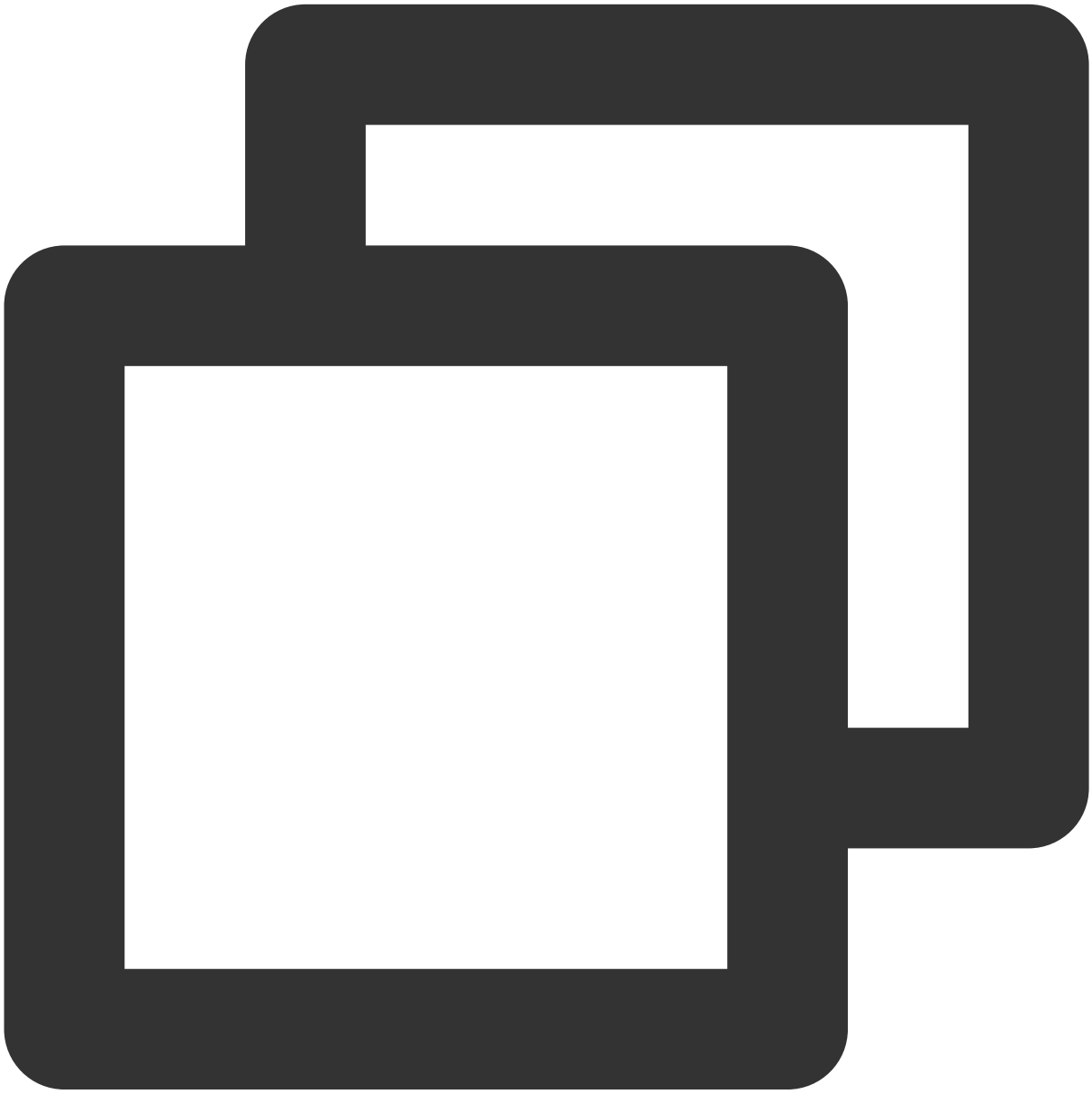
4. Create a Tencent Cloud IM chat group by case ID and user ID



```
const createGroup = async function (groupName, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { Owner_Account: userId, Type: "Public", Name: groupName }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
```

```
}
```

In addition, the web server provides routes to join/leave/delete a Tencent Cloud IM chat group by case ID and agent ID. These are used when Salesforce trigger detects the changing of case agent. More details will be discussed in Step 3.



```
const joinGroup = async function (groupId, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId, MemberList: [{ Member_Account: userId }] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/add_group_member?s
```

```
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
}

const leaveGroup = async function (groupId, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId, MemberToDel_Account: [userId] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/delete_group_membe
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
}

const deleteGroup = async function (groupId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/destroy_group?sdka
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
}
```

That's all for the web server side. Once you set up the server, call the endpoint /createticket and

Check in Salesforce that a case is created.

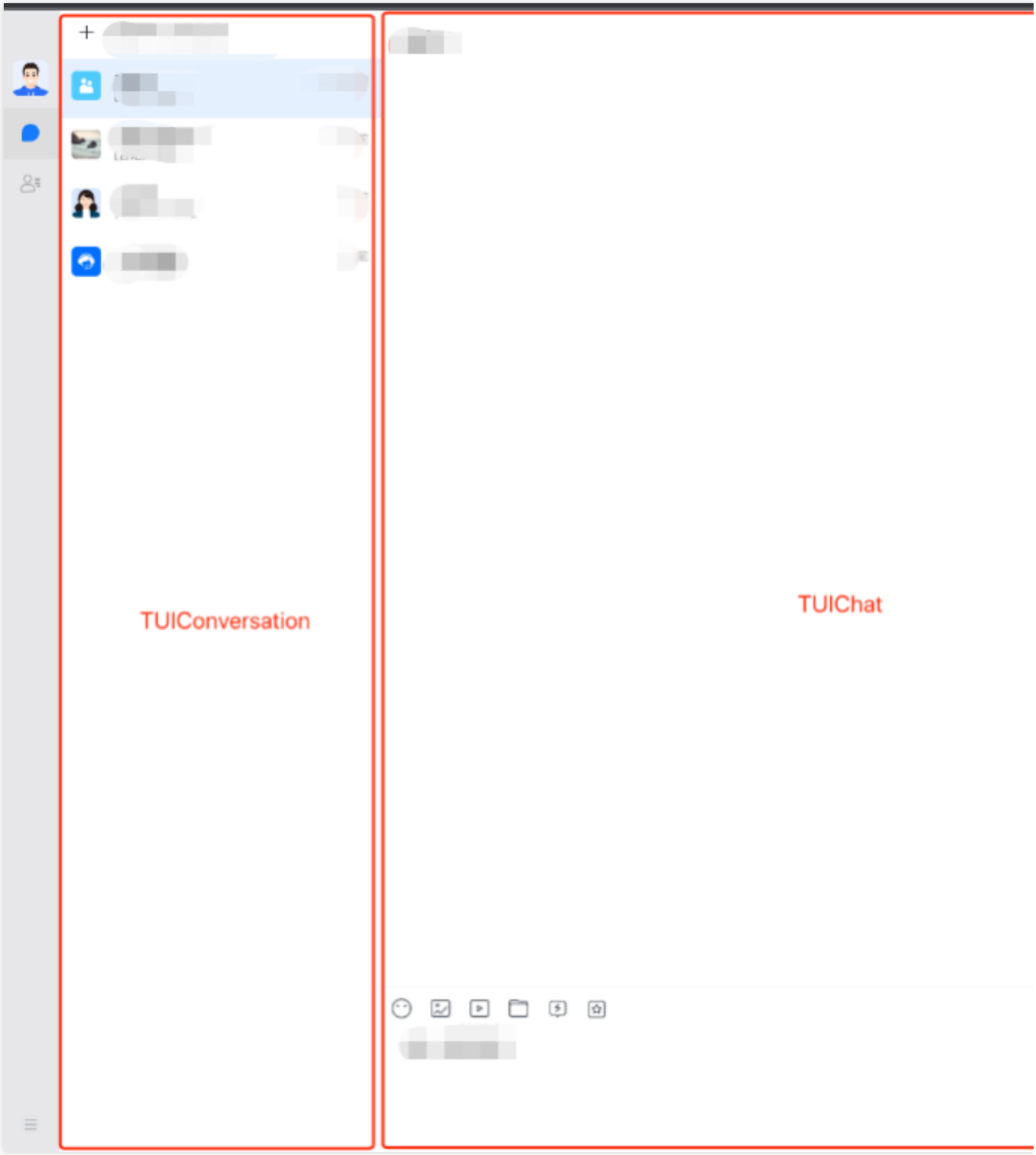
Check in the Tencent Cloud IM console for the group with the case ID as the group ID.

Step 2. Use the Tencent Cloud IM Web UI Kit to build a Salesforce utilities component

Here we show the steps to create a Salesforce utilities component with Tencent Cloud IM UI Kit. In Salesforce you may use Lightning Container to upload a third-party i-frame as a static resource, and host the content in an Aura

component using `lightning:container` . And you can use Tencent Cloud IM Web UIKit to build an agent chat component, and deploy it in the Lightning Container as a Salesforce utilities bar widget at the bottom.

1. First develop a chat component by using Tencent Cloud IM Web UIKit. Build it as a static resource with a root index.html and compress it as an zip file. Case agent's ID will be transmitted to the chat component, use that ID to init and login Tencent Cloud IM in the chat component.



2. Create a Lightning Container for your component, details arehere.

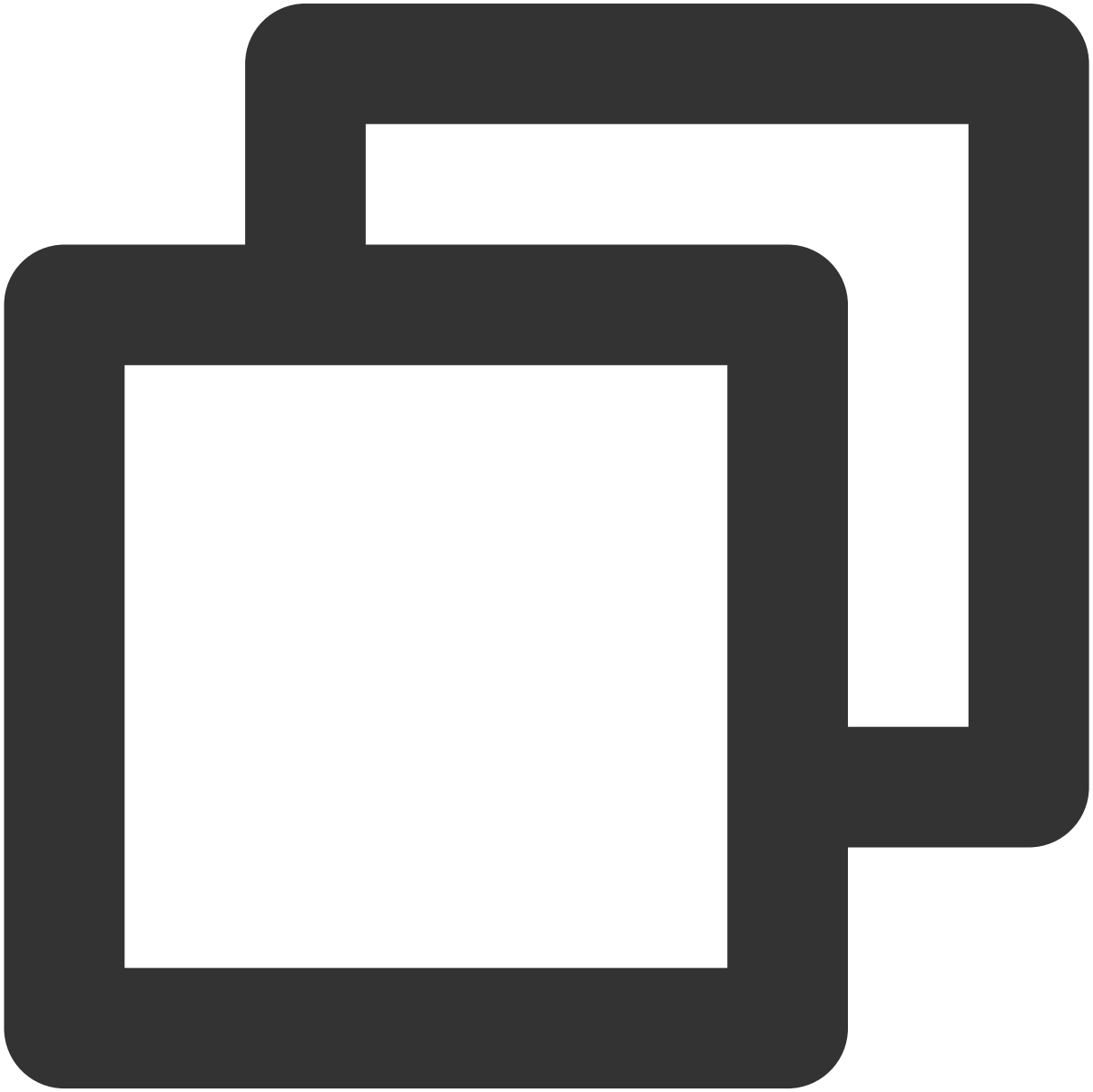
2.1 Go to Salesforce Developer Console

2.2 Click File -> New -> Lightning Component

2.3 Name = "tim_utilities_bar"

2.4 Click submit

3. Render the custom component to your bar widget. 3.1 Set aura:component as a utility bar and provide an aura:id



```
<!-- tim_utilities_bar.cmp -->
```

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
  <lightning:utilityBarAPI aura:id="utilitybar" />
</aura:component>
```

3.2 Upload the static resource to the Lightning Container

a. Go to Salesforce static resources and create a new resource called "tim_bar"

b. Upload the zip file of the resource and set "Cache Control" to "Public"

c. Click Save

Notes:

Index.html should always be at the root level of the .zip file

Be sure to click “save” after uploading your file.

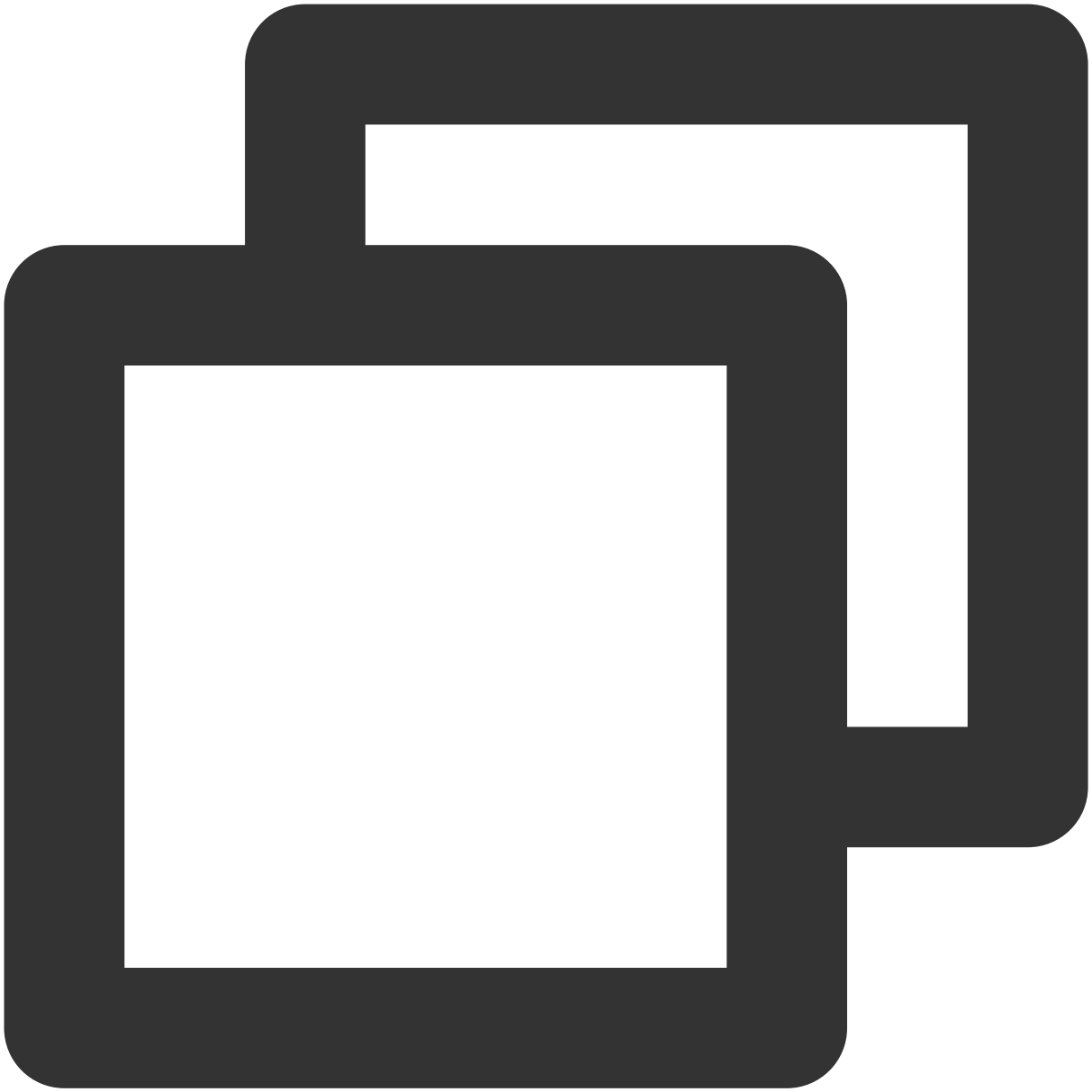
Salesforce saves the resource name, not the name of the .zip file.

100% of the code and assets you use in a Lightning Component will need to be included in the .zip file. Any external code dependencies will not work even if they are whitelisted in the CSP trusted sites list.

3.3 Reference the static resource "tim_bar" in the Utilities Bar widget

a. Add a lightning:container tag to the Utilities Bar widget. The aura:id should be "TIM_Bar".

b. Reference the static resource. "!$Resource.tim_bar + '/index.html'}" . Note that tim_bar is the saved "static resource", not the name of the uploaded .zip file.



```
<!-- tim_utilities_bar.cmp -->
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
  <lightning:utilityBarAPI aura:id="utilitybar" />
  <aura:attribute name="recordId" type="String" />
  <aura:attribute name="data" type="String" />
  <lightning:navigation aura:id="navService" />
  <lightning:container
    aura:id="TIM_Bar"
    src="{!$Resource.tim_bar + '/index.html'}"
  />
</aura:component>
```

3.4 Add the Utilities Bar Widget to display in Salesforce

a. Click "Setup" and search for "App Manager" to set where the Utilities Bar widget will appear

b. Click "▾" and "Edit" in the App called Service Console

c. In App Setting, click "Utility Items (Desktop Only)"

d. Click "Add Utility Item"

e. Select the "tim_utilities_bar"

f. Set the width and height

g. Check "Start automatically"

h. Click -> "Save"

3.5 Update the Salesforce CSP file to grant permissions to Access Tencent Cloud IM in Salesforce

a. Go to Salesforce -> Setup -> Search -> "CSP Trusted sites"

b. Add "New Trusted Sites" (allow all CSP Directives):

wss://wss.im.qcloud.com

c. Go to Salesforce -> Setup -> Search -> "CORS"

d. Add "New" Allowed Origins List:

https://*.qq.com

https://*.qcloud.com

e. Go to Salesforce -> Setup -> Search -> ""

f. Add "New Remote Site" List:

The Url of the web server!

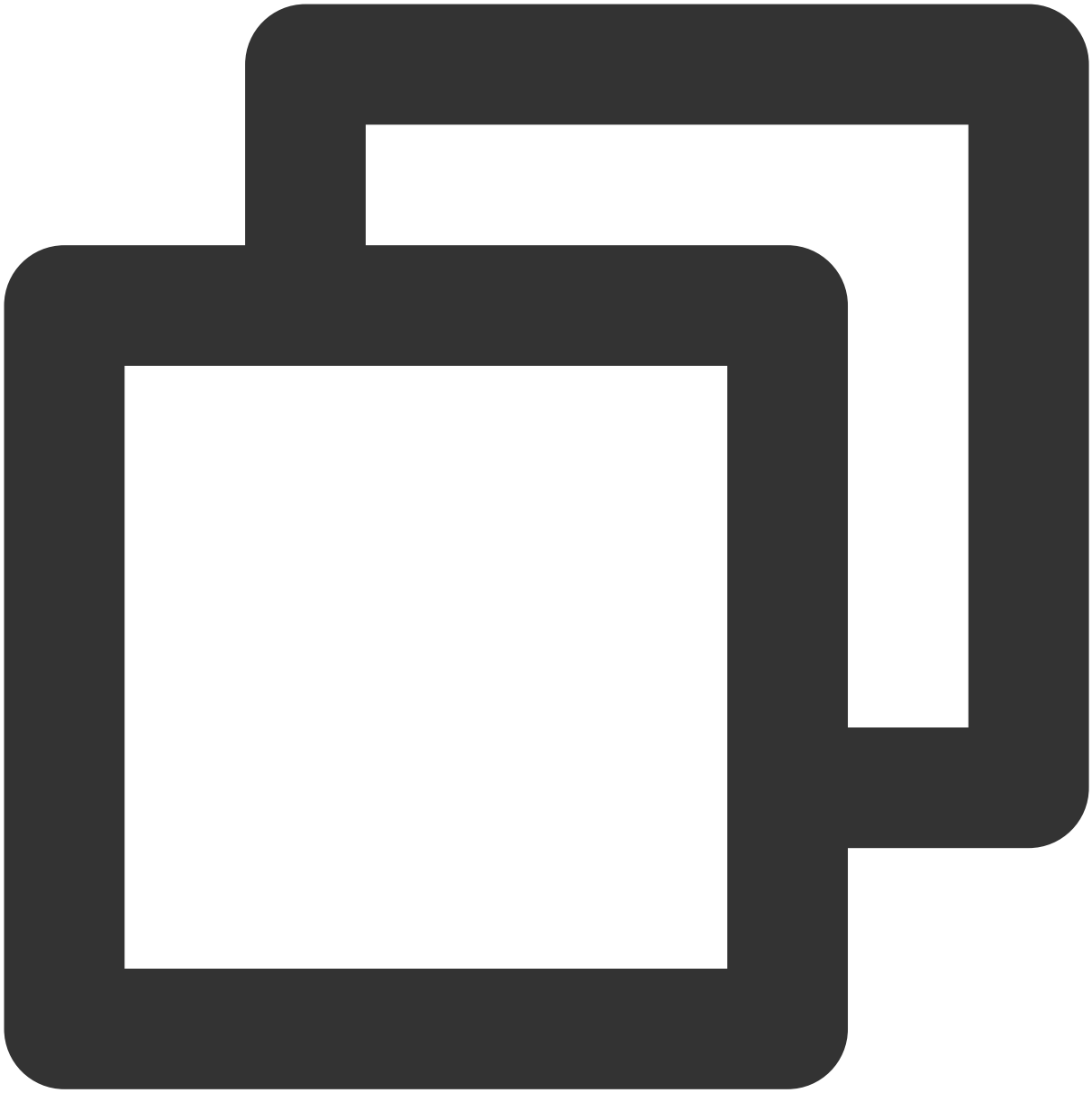
4. Initialize the Lightning Container

Once Lightning Container is ready, send an LLC message to inform Utilities Bar Widget

Utilities Bar Widget needs to send Agent's id to our component

Once received messages from Utilities Bar Widget, we render the UIKit

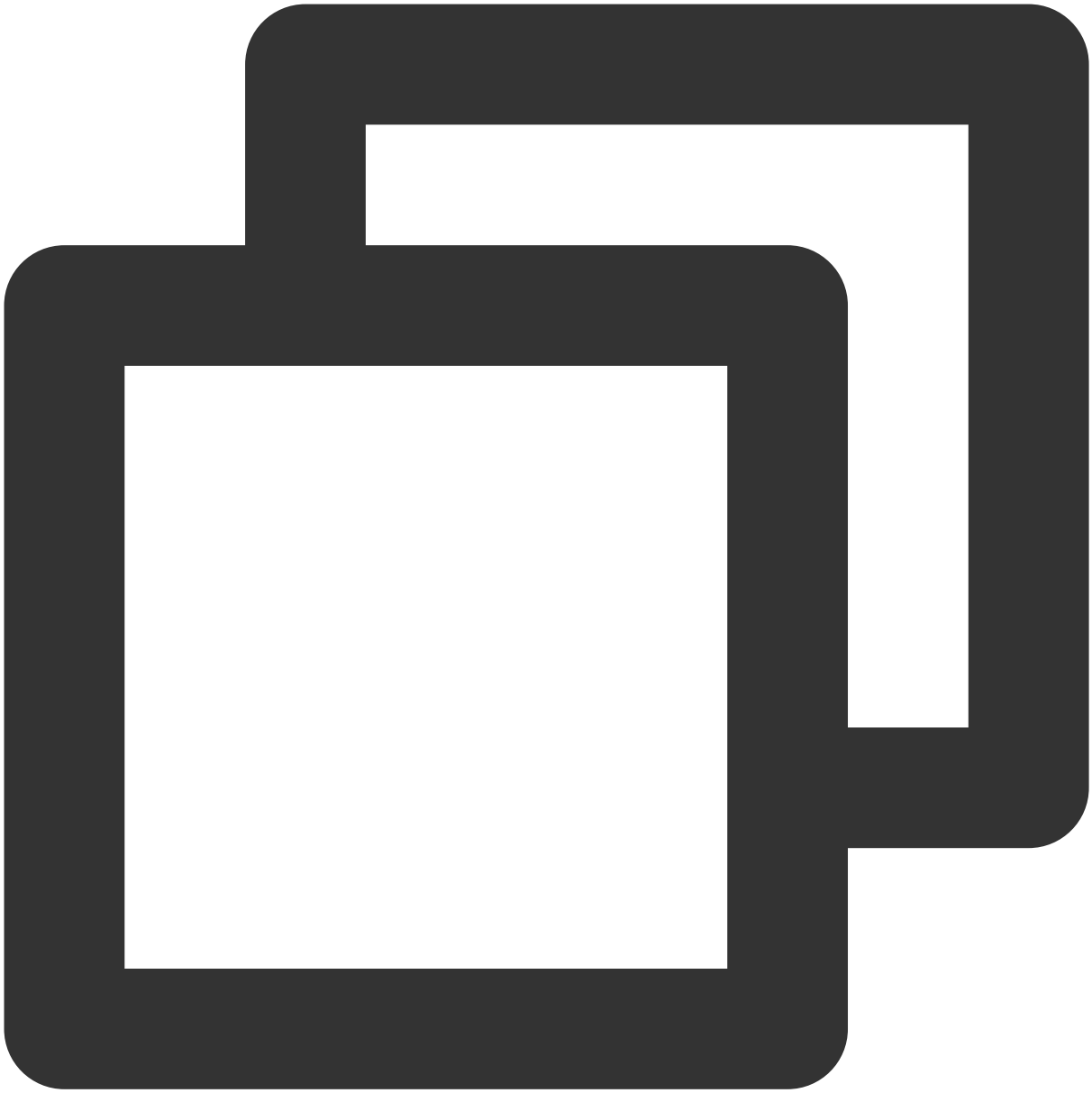
Follow the code shown below:



```
// tim_utilities_barController.js
({
handleMessage: function(component, message, helper) {
var payload = message.getParams().payload
// Once container is ready, initUIKit
if (payload === "READY") helper.initUIKit(component, message, helper)
}
});
({
initUIKit: function (component, message, helper) {
// Get Agent's ID
```

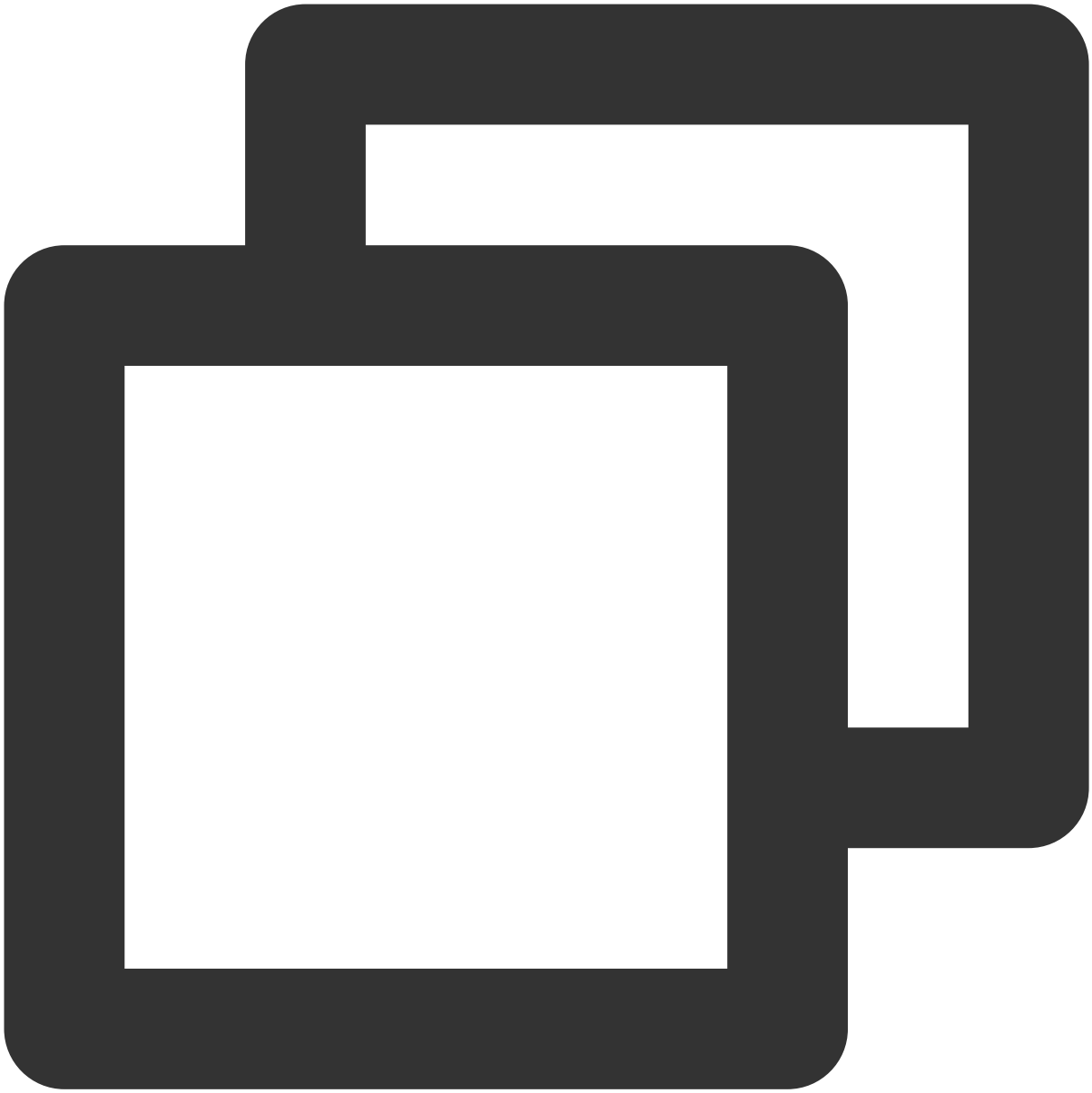
```
var userId = $A.get("$SObjectType.CurrentUser.Id")
var message = { userId: userId }
try {
// Send the ID to the component
component.find("TIM_Bar").message(message)
} catch (err) {
console.error("Error from Utilities Bar:", err)
}
},
})
```

Add onMessage handler to the Utilities Bar Widget：



```
<!-- tim_utilities_bar.cmp -->
<lightning:container
aura:id="TIM_Bar"
src="{!$Resource.tim_bar + '/index.html'}"
onmessage="{!c.handleMessage}"
/>
```

In your script, use LLC package to render your app when Lightning Container has loaded.



```
// index.js
try {
const clientState = "READY";
LLC.sendMessage(clientState);
console.warn("Lightning Container --> TO SALESFORCE --> Sent:", clientState);
} catch (e) {
console.error("LLC NOT WORKING", e);
}
try {
LLC.addErrorHandler((error) => console.log("LLC ERROR:", error));
LLC.addMessageHandler((salesforceMessage) => {
```

```
console.warn("SALESFORCE --> Lightning Container --> Arrived:", salesforceMessage);
const app = createApp(App, {
user: salesforceMessage
});
app
.use(store)
.use(router)
.use(TUIKit)
.use(Aegis)
.use(ElementPlus)
.mount('#app');
});
} catch (e) {
console.error("Error from LLC!!", e);
}
```

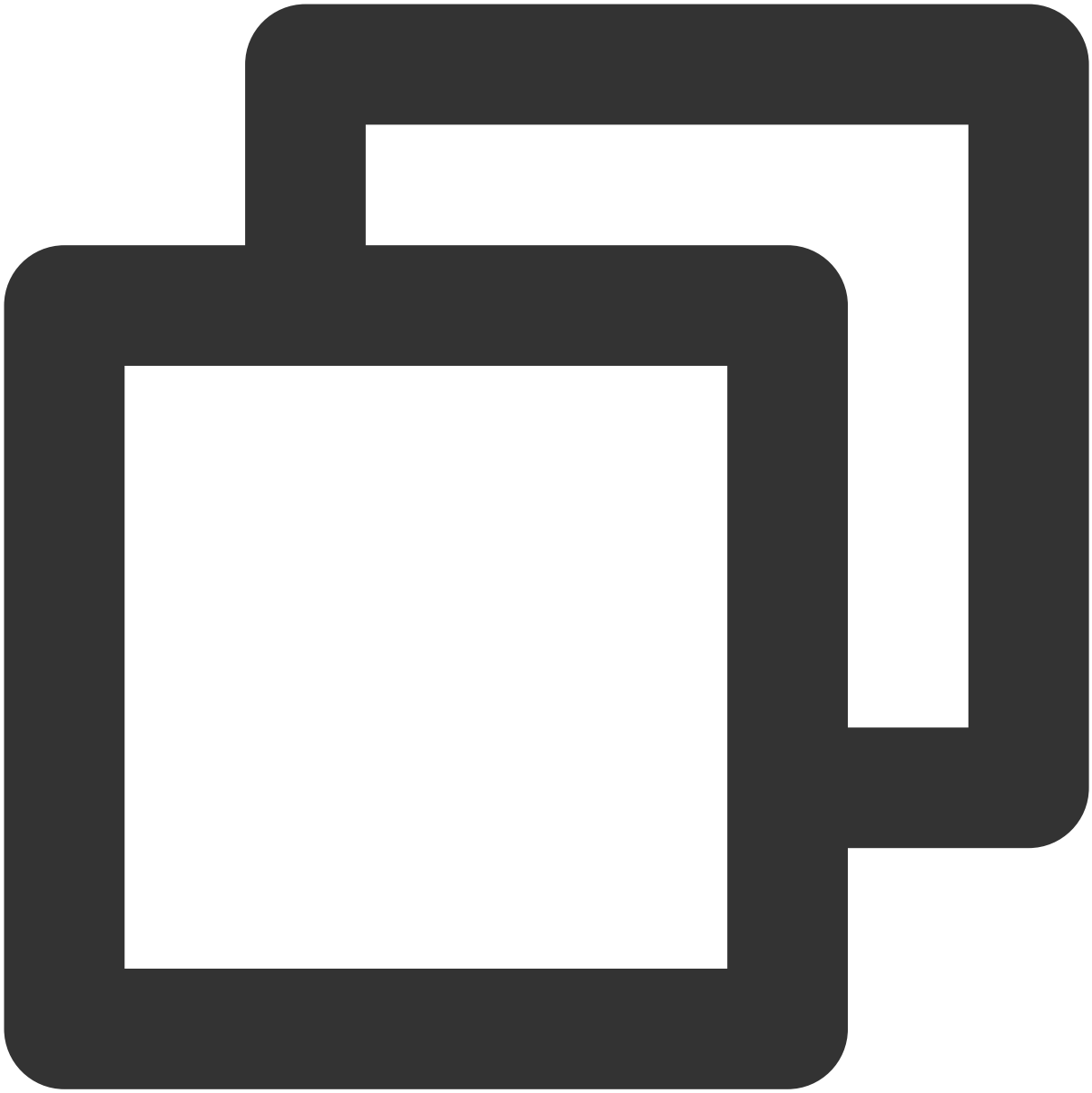
Step 3. Listen for Salesforce Case assignment and set IM Chat Group members

In Salesforce, a case is assigned to an agent manually or automatically. Hence we need to invite new agent to the group and make the previous agent leaves the group. We use Salesforce Apex Callous to listen for the changing of the case agent assignment. Here's what to do by calling the Salesforce Apex Callout.

1. Listen to manual case assignment

When the designated agent to one case is changed, the Apex Case Change Trigger calls the Apex Callout. In the callout, we a. delete the previous agent from the Tencent Cloud IM chat group and b. invite the new agent to the group. And when case is deleted, dismiss the Tencent Cloud IM group accordingly.

Go to Salesforce Developer console -> New -> Apex Trigger -> Name = "AssignAgent" & sObject = "Case"



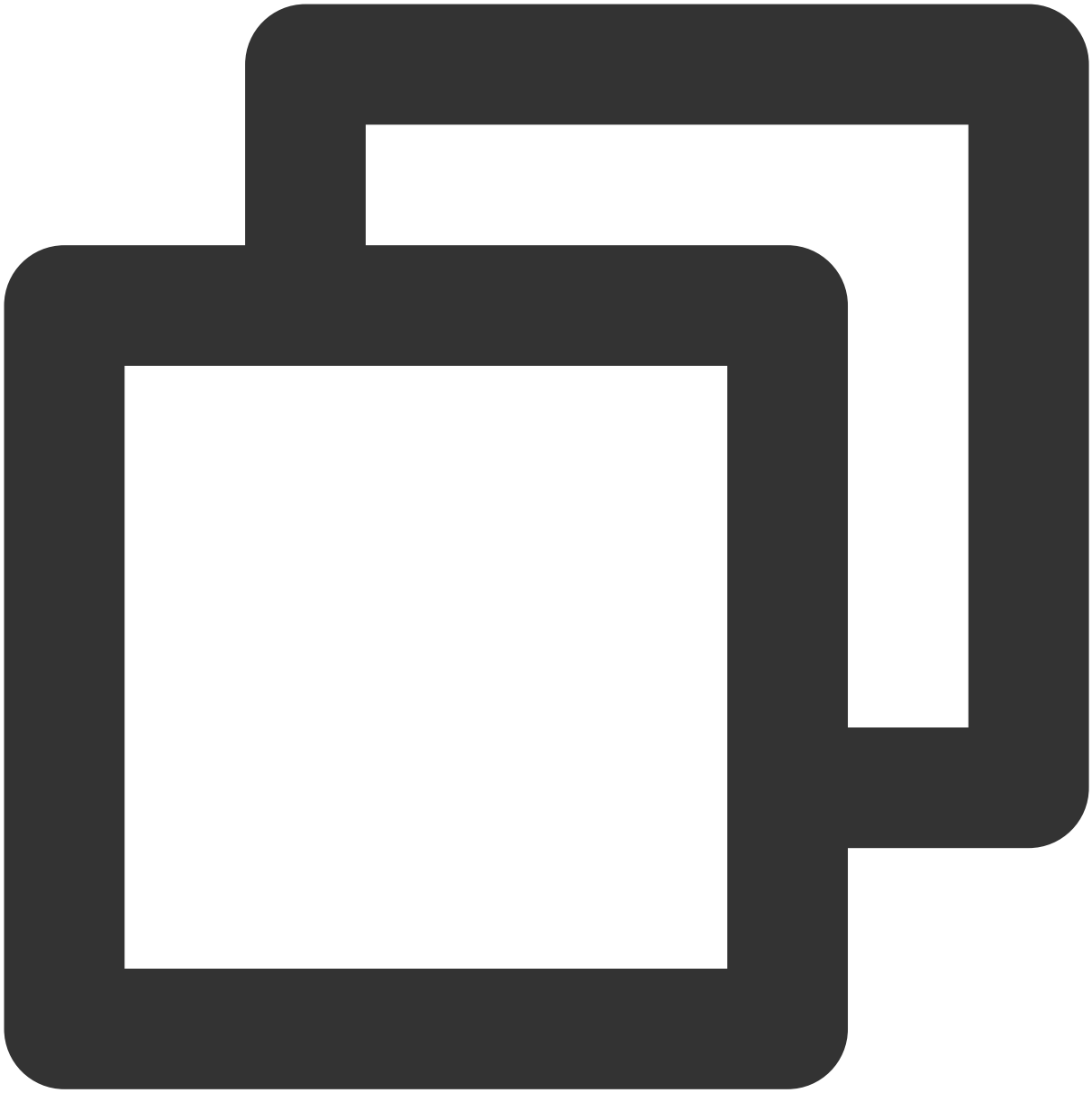
```
// AssignAgent.apxt
trigger AssignAgent on Case (after update, after delete) {
     if(trigger.isUpdate){
       // Case is updating
       System.debug('Case Update Fired:');
       for(Case a : trigger.new){
         Case oldCase = trigger.oldMap.get(a.ID);
         if(String.valueOf(a.OwnerId).substring(0, 3) == '005'){
           // Owner Agent ID is changed, 005 prefix means agent ID
           System.debug('Agent invited :' + a.OwnerId);
           // Assign new owner to the Tencent Cloud IM group
```

```
            String[] data = new String[2];
            data[0] = a.Id; // Case ID is group ID
            data[1] = a.OwnerId;
            TimCallouts.joinGroup(data); //Custom callout class
          }
          // New case agent is different fromthe current agent
          if(String.valueOf(oldCase.OwnerId).substring(0, 3) == '005' && oldCase.Own
            // Delete the old agent from the group.
            // Note: A Case's very first owner will be the system owner.
            System.debug('Old Agent will be removed from group' + a.Id);
            System.debug('leaveGroup: ' + oldCase.OwnerId);
            String[] removeData = new String[2];
            removeData[0] = a.Id; // Case ID is group ID
            removeData[1] = oldCase.OwnerId;
            TIMCallouts.leaveGroup(removeData);
          }
        }
    }
    if(trigger.isDelete ){
      // Case is deleting
      System.debug('Case Delete Fired:');
      for(Case a : trigger.old){
        if(String.valueOf(a.OwnerId).substring(0, 3) == '005'){
          System.debug('Delete Group :' + a.Id);
          String[] data = new String[1];
          data[0] = a.Id;
          TimCallouts.deleteGroup(data); //Custom callout class
        }
      }
    }
  }
```

2. Listen to automatic case assignment by Salesforce Omni Channel

Omni Channel detects a case assignment and Salesforce automatically creates an AgentWork object. If an agent accepts the assignment, the AgentWork Trigger may use Salesforce Callout to join the group.

Go to Salesforce Developer console -> New -> Apex Trigger -> Name = "AgentOmniChannel" & sObject = "AgentWork"

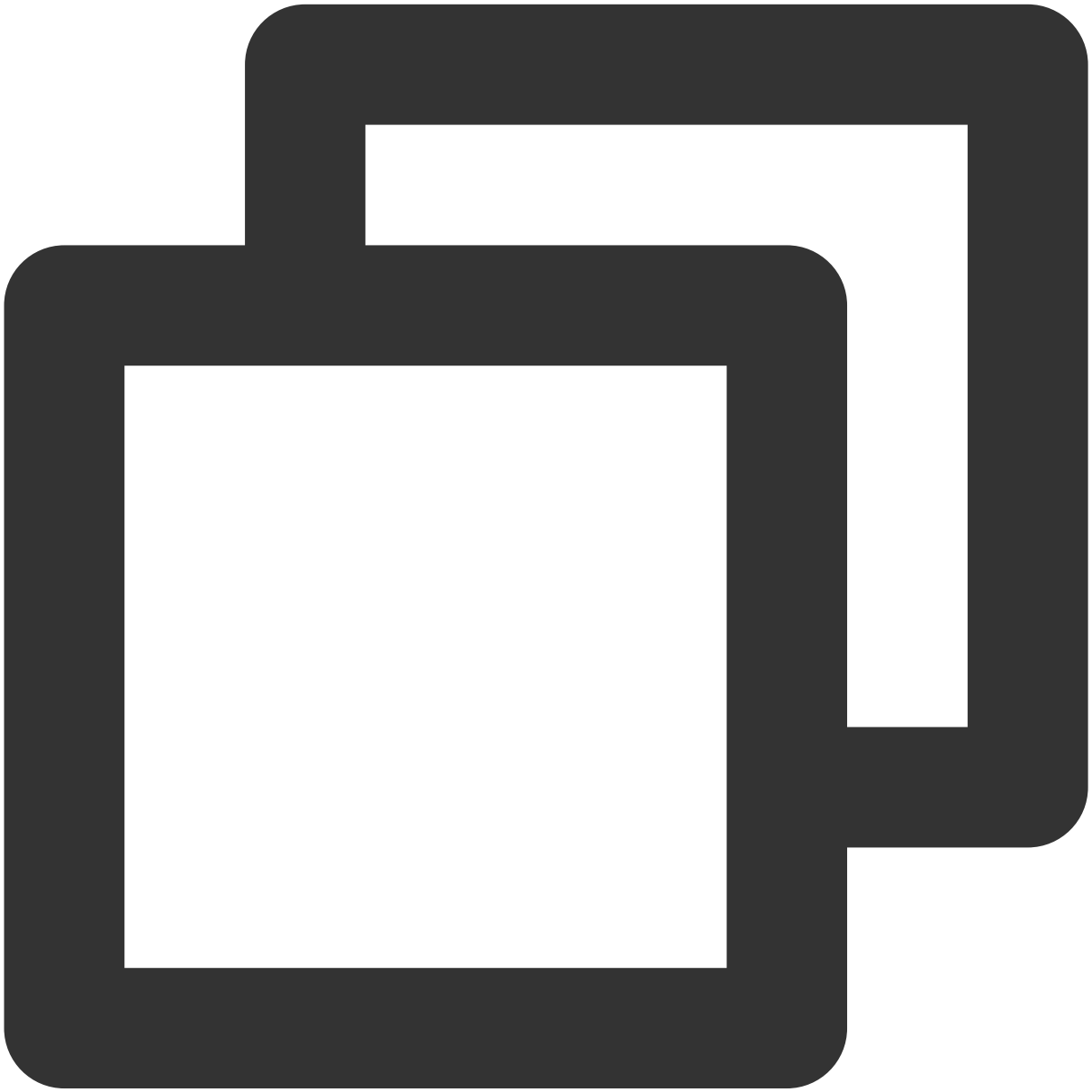


```
// AgentOmniChanne.apxt
trigger AgentOmniChannel on AgentWork (after update, after insert) {
    if(Trigger.isUpdate){
      for(AgentWork a : Trigger.new){
            AgentWork oldCase = Trigger.oldMap.get(a.ID);
            if(a.Status == 'Opened' && String.valueOf(a.OwnerId).substring(0, 3) ==
              String[] data = new String[2];
              data[0] = a.WorkItemId;
              data[1] = a.OwnerId;
              TIMCallouts.joinGroup(data);
            }
```

```
        }
    }
}
```

3. Set Salesforce Callout to invite/remove agents.

Go to Salesforce Developer console -> New -> Apex Class -> Name = "TIMCallOuts"



```
// TIMCallOuts.apxc
public class TIMCallouts {
  @future(callout=true)
  public static void joinGroup(String[] data) {
```

```
    String groupId = data[0];
    String userId = data[1];
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://{your_web_server}/joingroup');
    request.setMethod('POST');
    request.setHeader('Content-Type', 'application/json;charset=UTF-8');
    request.setBody('{"userId":"'+ userId +'", "groupId":"'+ groupId +'"}');
    HttpResponse response = http.send(request);
    // Parse the JSON response
      if (response.getStatusCode() != 200) {
        System.debug('Join group failed: '+response.getStatusCode()+' '+response.ge
      } else {
        System.debug('Tencent Cloud IM Response: ' + response.getBody());
      }
  }

  @future(callout=true)
  public static void leaveGroup(String[] data) {
      String groupId = data[0];
      String userId = data[1];
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://{your_web_server}/leavegroup');
      request.setMethod('POST');
      request.setHeader('Content-Type', 'application/json;charset=UTF-8');
      // Set the body as a JSON object
      request.setBody('{"userId":"'+ userId +'", "groupId":"'+ groupId +'"}');
      HttpResponse response = http.send(request);
      // Parse the JSON response
      if (response.getStatusCode() != 200) {
        System.debug('Leave group failed: ' + response.getStatusCode() + ' ' + resp
      } else {
        System.debug('Tencent Cloud IM Response: ' + response.getBody());
      }
    }

    @future(callout=true)
    public static void deleteGroup(String data) {
      String groupId = data;
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('http://{your_web_server}/deletegroup');
      request.setMethod('POST');
      request.setHeader('Content-Type', 'application/json;charset=UTF-8');
      // Set the body as a JSON object
      request.setBody('{"groupId":"'+ groupId + '}');
```

```
        HttpResponse response = http.send(request);
        // Parse the JSON response
        if (response.getStatusCode() != 200) {
          System.debug('Delete group failed: ' + response.getStatusCode() + ' ' + res
        } else {
          System.debug('Tencent Cloud IM Response: ' + response.getBody());
        }
    }
}
```

Conclusion

That's all you need to know to allow end users from any appliction to start chatting with a Salesforce agent. If you have any further questions please send an e-mail at tencentcloud_im@tencent.com, we'd be delighted to give you more details about the solution or any other solutions to build a modern real-time communication system via Tencent Cloud IM.

How to integrate Tencent IM with Zendesk

最近更新时间：2024-02-07 17:30:51

Introduction

Zendesk, one of the most prevalent SaaS products in customer support, sales, and other customer communications. Meanwhile, according to Gartner, Tencent Cloud Instant Messaging made the champion in Chinese market and one of the most compotent providers in Communication Platform as a Service (CPaaS) in the global market. Naturally, it comes to us to bring you the solution of integrating Tencent Cloud IM with Zendesk.
In this essay, we will discuss how to build an client-agent-real-time communication system that extends the boundary of support team in Zendesk and enables support team to chat with clients in any platforms. We have accomplished Tencent Cloud IM App for Zendesk ticket bar and all you need to do is to install it as well as publish your own client side by the following instructions.
If that's not the integrate solution on your interest list, don't rush to the close button. Go through the essay and build your own private App for Zendesk by TUIKit, which is not a painstaking errand on your ticket list but instead a few man hours or days depending on the complexity of the App you want to build.

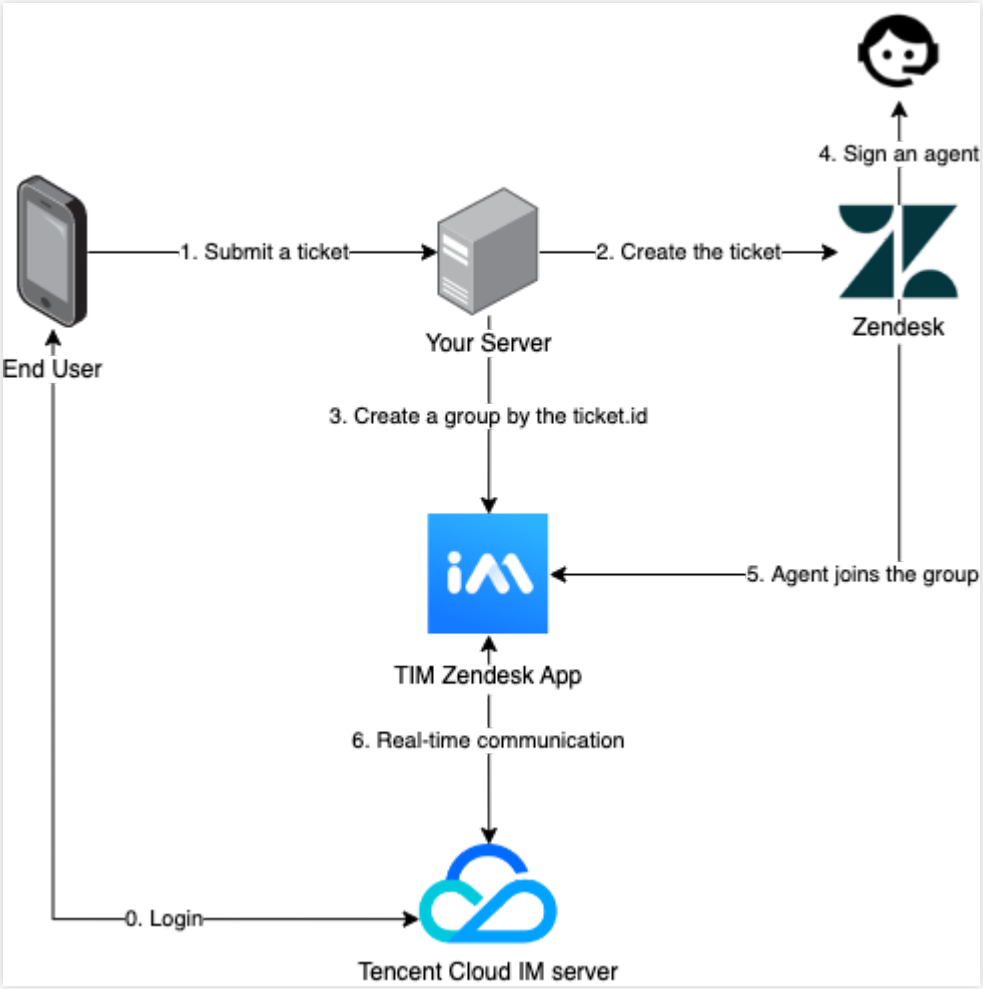
Prerequisites

If you haven't registered for Zendesk, try free trail.
To use any service of Tencent Cloud IM, you must register for Tencent Cloud, which contains so many cloud-based services other than IM. After that, click `Create Application` on the IM Console to get your `SDKAppID` , that's what you need to initiate an IM service.
For client side construction, you'll need a web server as well, which will not be included in this article.

Integration Map

The goal of this integrate soluion is straight and clear: listen to the assignment of agent and invite the agent to the conversation with the client outside Zendesk to discuss the issue or consultation in the ticket.

Here is the general integration map:



The general workflow is:

1. End user logs in to Tencent Cloud IM.

1. End user submits a ticket to your backend server.

2. According to the submitted information, create the ticket for Zendesk.

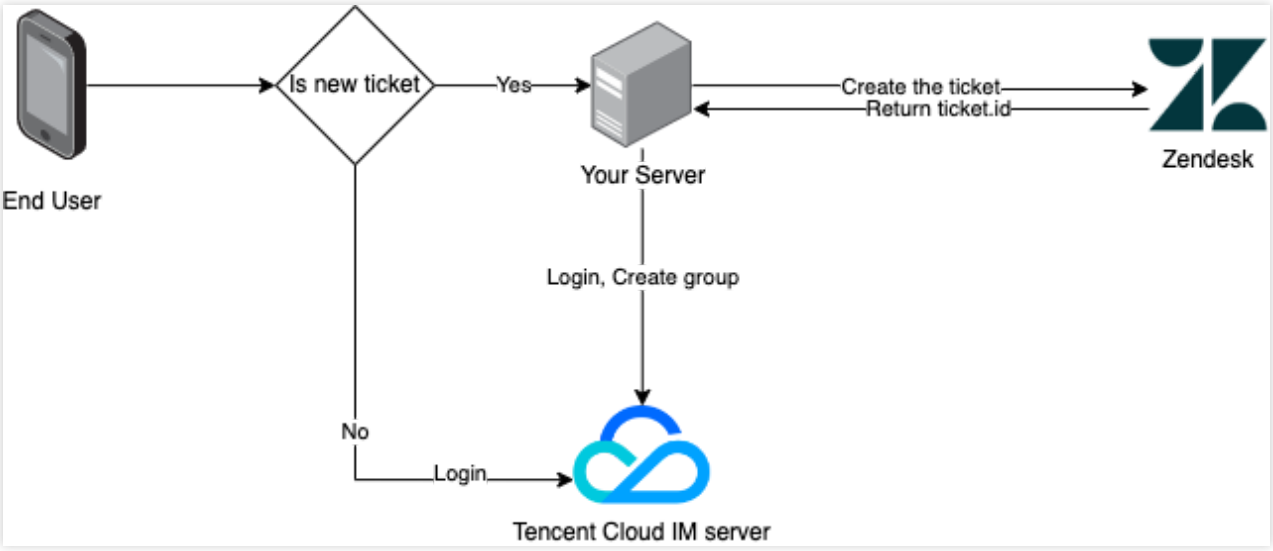
3. Create a group in Tencent Cloud IM, waiting for agent to join the group.

4. An agent takes the ticket or it is assigned to an agent.

5. The assignee joins the Tencent Cloud IM group.

6. Now client and agent can chat freely.

The workflow for the client side is:

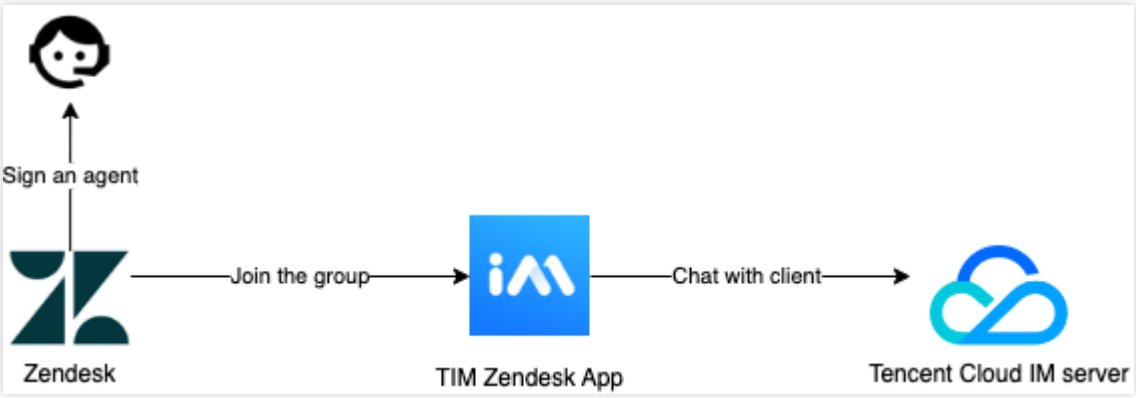


Before logged in, client may choose to reuse the ticket created last time. And for that, just login to Tencent Cloud IM server.

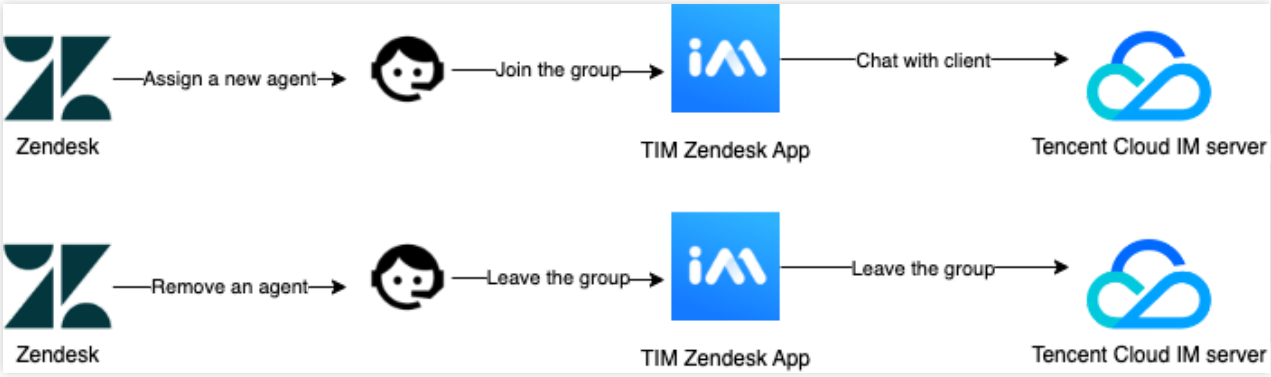
Otherwise submit a new ticket to Zendesk.

Login to the server and use the returned ticket.id to create a group.

The process of Tencent Cloud IM for Zendesk is:



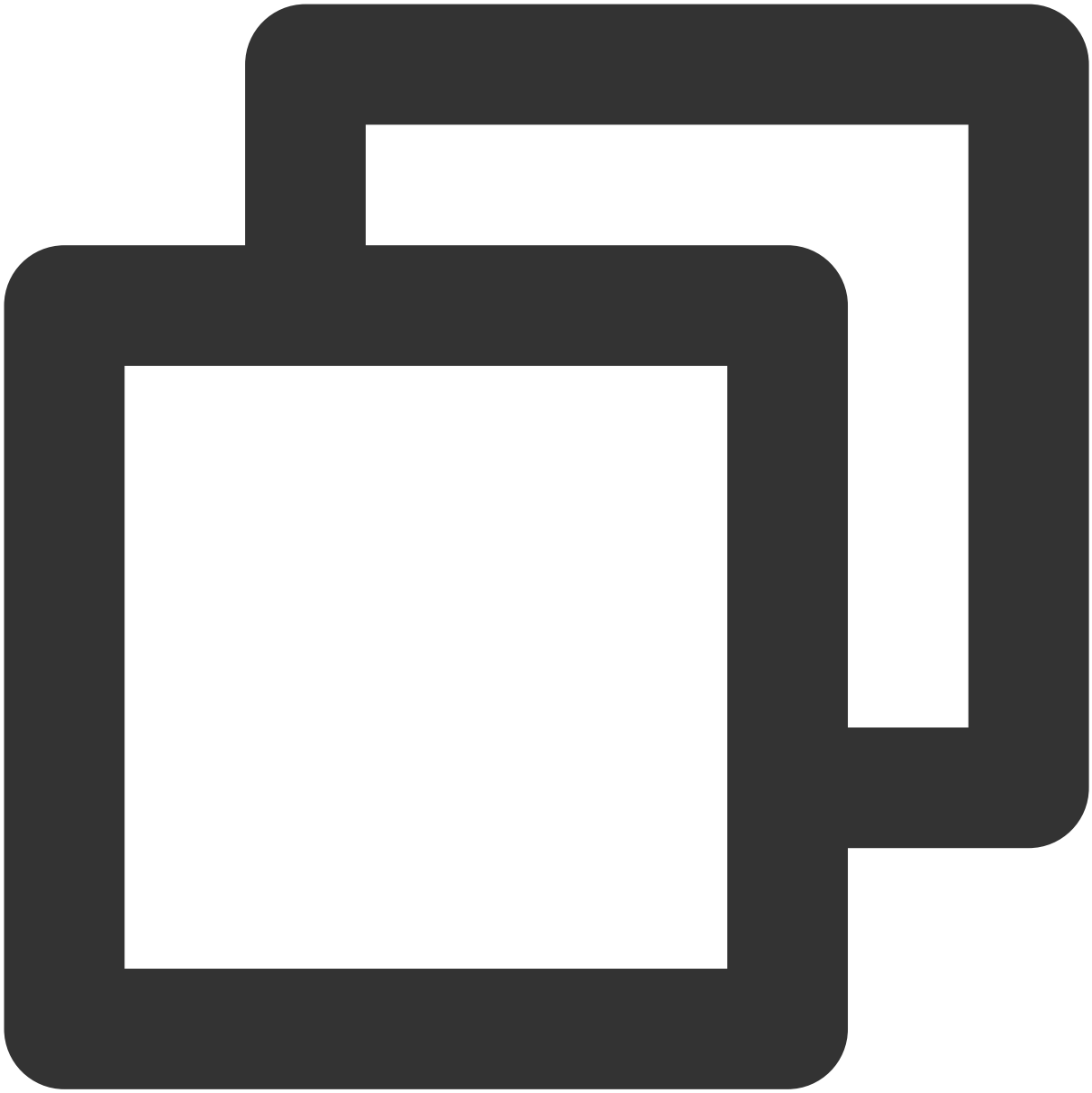
The process of updating the ticket:



Client Side Construction

You'll need to develop a frontend project to accomplish the chatting features, and a backend project to invoke Zendesk requests. Don't be panic about the works need to do, the frontend can be done smoothly with TUIKit like piling up blocks. TUIKit is a set of TUI components based on IM SDKs, which provides independent components including conversation, chat, searching, relationship chain, group, and audio/video call. TUIKit currently covers platforms including iOS, Android, Web and Flutter, and we're working on React Native as well. Stay tuned on the channel to get the lastest status. By applying TUIKit, you just have to define your strategy to invoke the `createTicket` API.

Here's all the code to build the backend project:



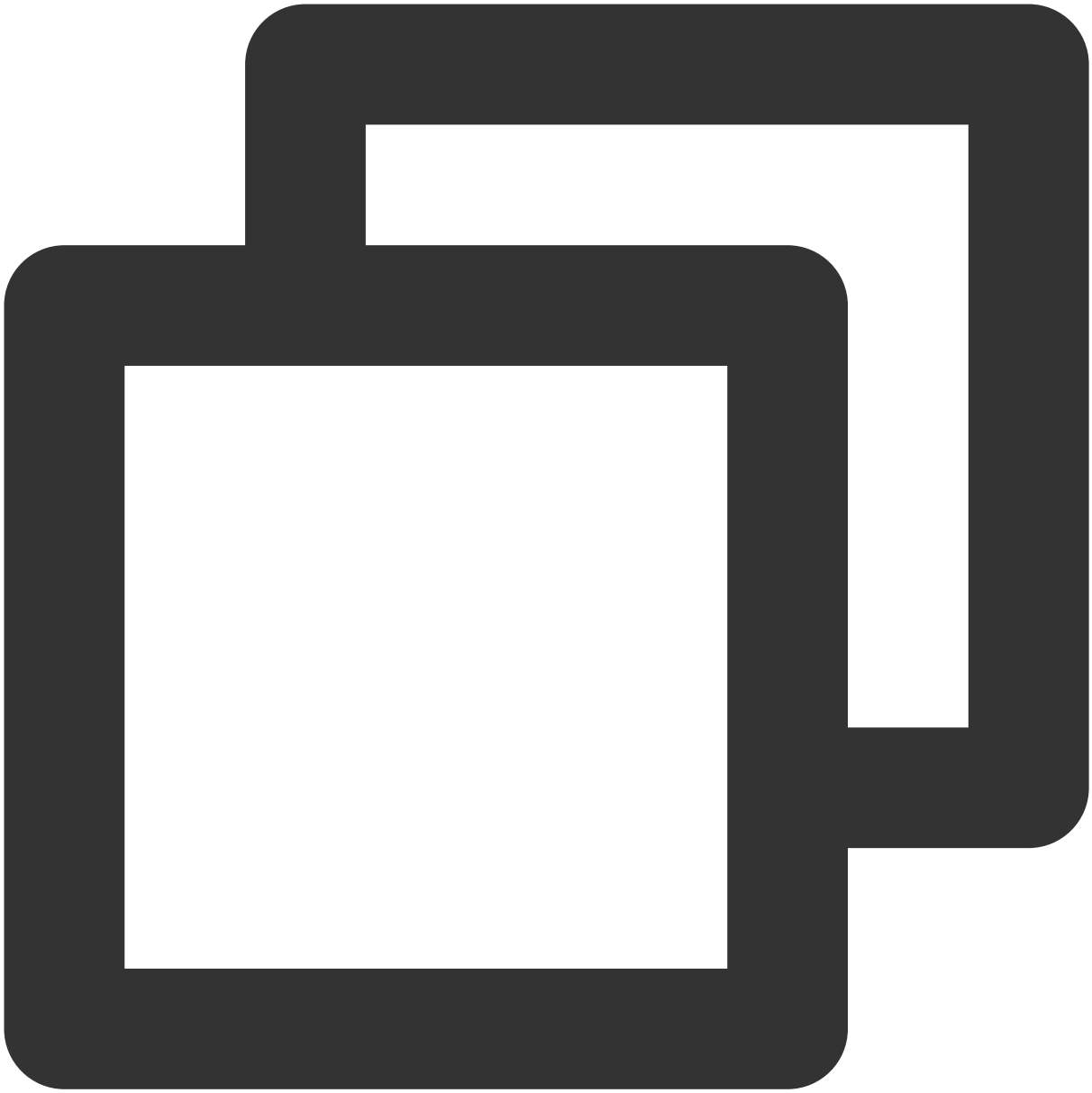
```
const express = require('express');
const axios = require('axios').default;
var TLSSigAPIv2 = require('tls-sig-api-v2'); // Generate UserSig for TIM
require('dotenv').config();

const app = express();
app.use(express.json());

const port = process.env.PORT || 15000;
const YOUR_SDKAPPID = process.env.YOUR_SDKAPPID || 0;
const YOUR_SECRET = process.env.YOUR_SECRET || '';
```

```
const ADMIN_USERID = process.env.ADMIN_USERID || '';

const zendeskAxioInstance = axios.create({
  baseURL: `https://${process.env.SUB_DOMAIN}.zendesk.com/api/v2/`,
  auth: {
    username: process.env.EMAIL || '',
    password: process.env.TOKEN || ''
  },
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  timeout: 35000
});

app.use(express.json());

app.use(function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', '*');
  next();
});

app.post('/createticket', async (req, res) => {
  const { userId, caseInfo } = req.body;
  if (!userId) return res.status(500).send('Missing userId');

  const zendeskCase = await createTicket(caseInfo);
  if (!zendeskCase.success) return res.status(500).send('Case creation failed');

  const groupId = zendeskCase.id;
  const result = await createGroup(
    groupId,
    userId,
    caseInfo.subject || groupId
  );
  if (result.ErrorCode !== 0)
    return res.status(500).send('Group creation failed');
  sendGreeting(result.GroupId);

  res.status(200).send(result);
});

const createTicket = async function (caseInfo) {
  const { subject, desc, name, email } = caseInfo;
  const data = {
    request: {
      requester: {
```

```
        name: name || 'Anonymous customer',
        email: email
      },
      subject: subject,
      comment: {
        body: desc
      }
    }
  };
  try {
    const result = await zendeskAxioInstance({
      method: 'POST',
      url: '/requests.json',
      data: data
    });
    return {
      id: result.data.request.id,
      success: true
    };
  } catch (e) {
    return { id: undefined, success: false, error: e };
  }
};

const generateUserSig = function () {
  const expires = 600;
  const api = new TLSSigAPIv2.Api(YOUR_SDKAPPID, YOUR_SECRET);
  return api.genSig(ADMIN_USERID, expires);
};

const generateRandom = function () {
  return Math.floor(Math.random() * 4294967295);
};

const createGroup = async function (groupId, userId, groupName) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    Type: 'Public',
    Name: groupName,
    GroupId: `ZENDESK#${groupId}`,
    ApplyJoinOption: 'FreeAccess',
    MemberList: [{ Member_Account: userId }]
  };
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
    const groupRes = await axios.post(url, data);
```

```
    return groupRes.data;
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e };
  }
};

const sendGreeting = async function (groupId) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    GroupId: groupId,
    Content: "We're assigning an agent to your subject, please wait..."
  };
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/send_group_system_
  try {
    const groupRes = await axios.post(url, data);
    return groupRes.data;
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e };
  }
};

app.listen(process.env.PORT || port, () =>
  console.log(`Example app listening on port ${port}!`)
);
```

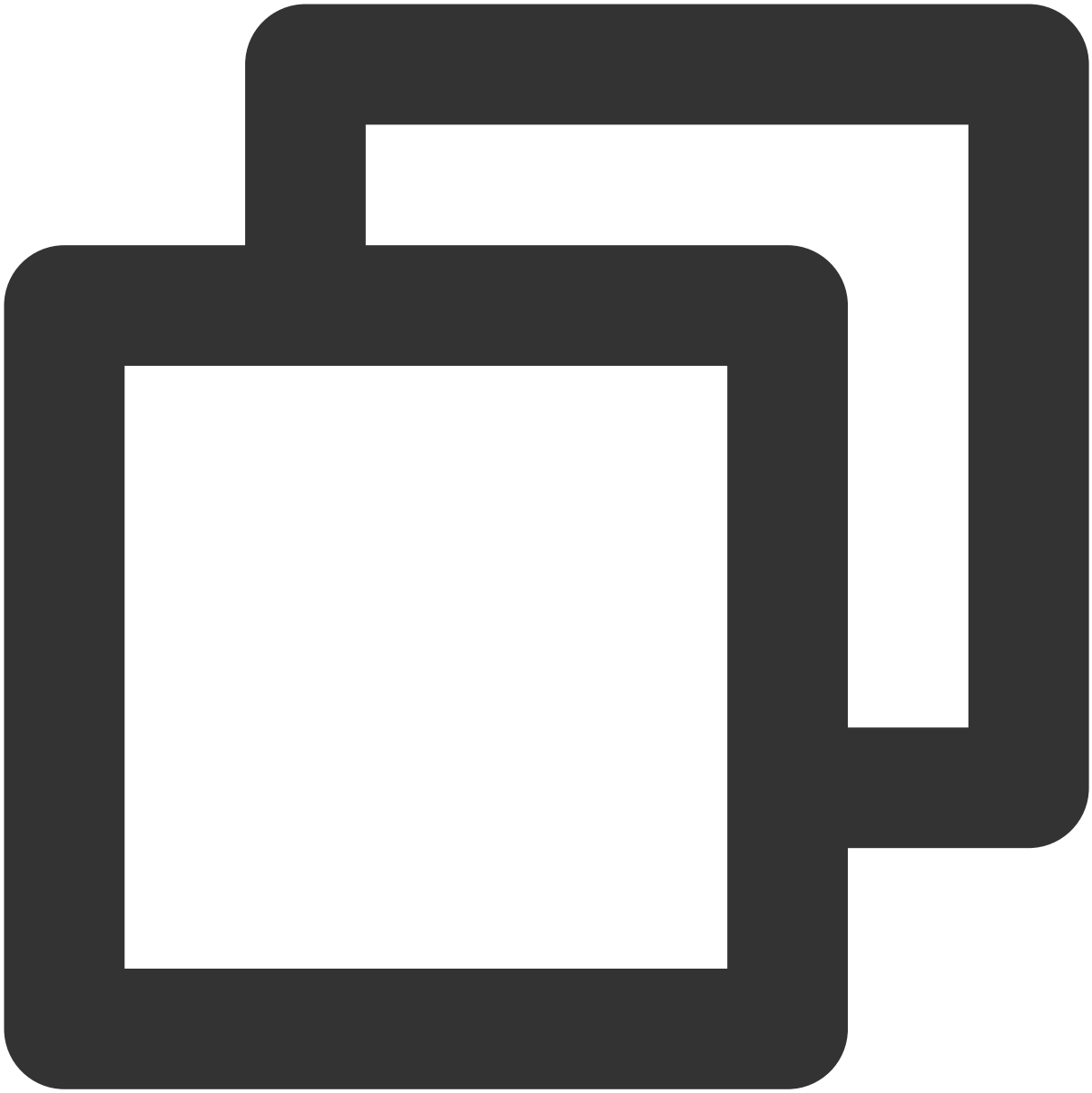
Let's break down the code.



```
const port = process.env.PORT || 15000;
const YOUR_SDKAPPID = process.env.YOUR_SDKAPPID || 0;
const YOUR_SECRET = process.env.YOUR_SECRET || '';
const ADMIN_USERID = process.env.ADMIN_USERID || '';
```

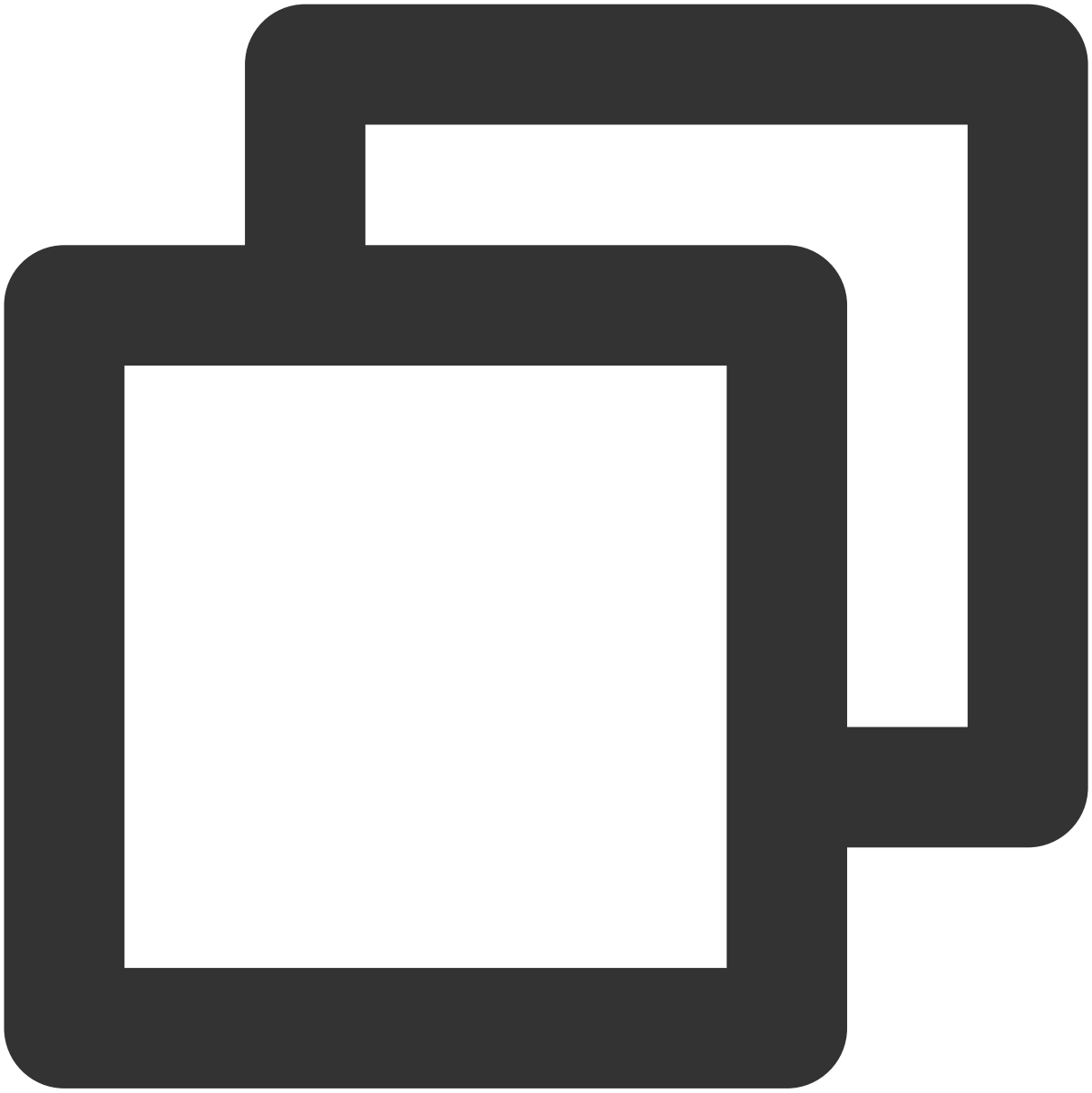
The first part requires you to set your Tencent Cloud IM identification information as the environment variables. You can find all the information here.

`Caveat: Do not note down your Secret in your code for it might be leaked to the Internet.`



```
const zendeskAxioInstance = axios.create({
  baseURL: `https://${process.env.SUB_DOMAIN}.zendesk.com/api/v2/`,
  auth: {
    username: process.env.EMAIL || '',
    password: process.env.TOKEN || ''
  },
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  timeout: 35000
});
```

Secondly, config Zendesk request.



```
// Create zendesk ticket
const createTicket = async function (caseInfo) {
  const { subject, desc, name, email } = caseInfo;
  const data = {
    request: {
      requester: {
        name: name || 'Anonymous customer',
        email: email
      },
```

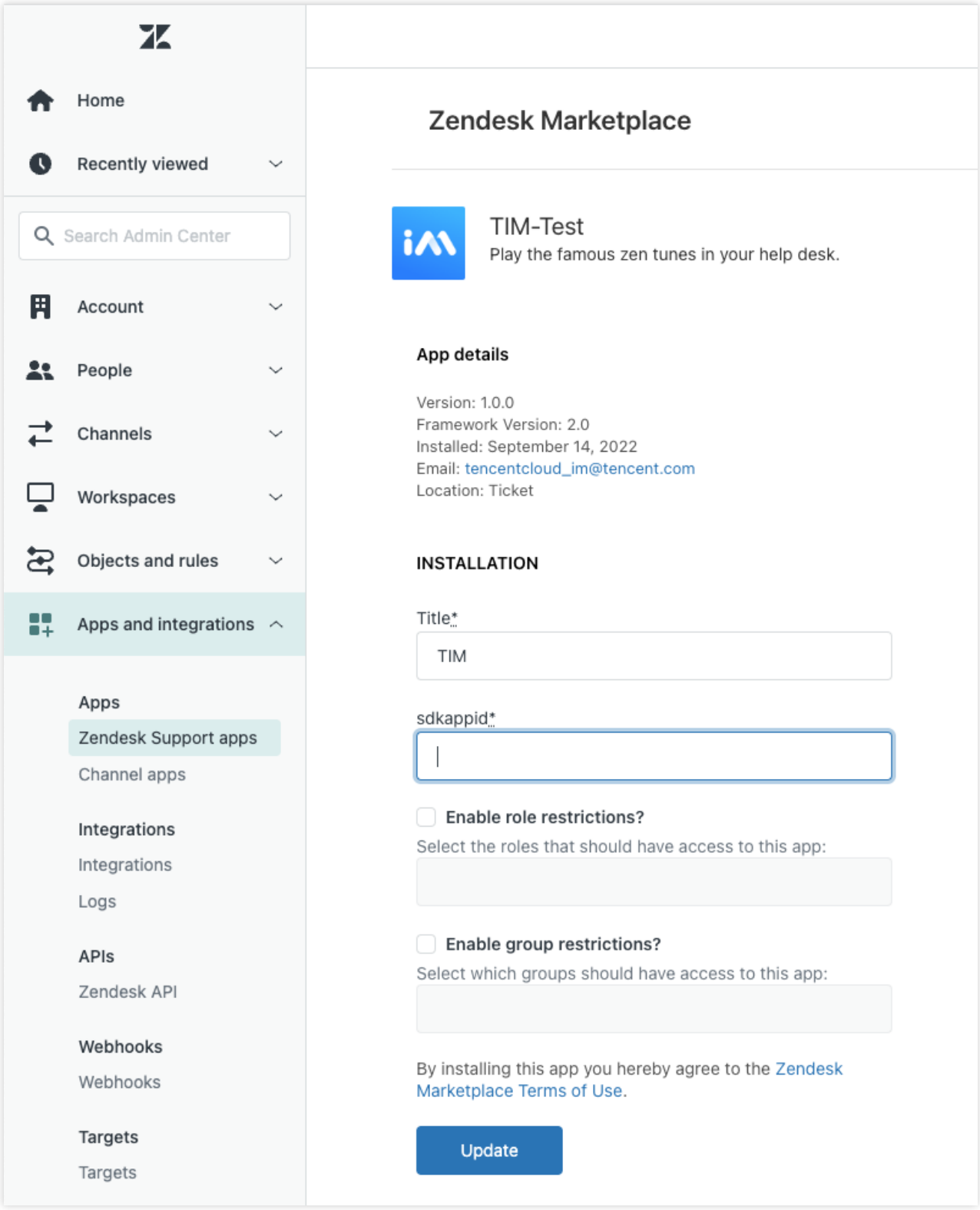
```
      subject: subject,
      comment: {
        body: desc
      }
    }
  };
  try {
    const result = await zendeskAxioInstance({
      method: 'POST',
      url: '/requests.json',
      data: data
    });
    return {
      id: result.data.request.id,
      success: true
    };
  } catch (e) {
    return { id: undefined, success: false, error: e };
  }
};

// Create IM group
const createGroup = async function (groupId, userId, groupName) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    Type: 'Public',
    Name: groupName,
    GroupId: `ZENDESK#${groupId}`,
    ApplyJoinOption: 'FreeAccess',
    MemberList: [{ Member_Account: userId }]
  };
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
    const groupRes = await axios.post(url, data);
    return groupRes.data;
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e };
  }
};
```

Last, create the Zendesk ticket by the input parameters and create an IM group with returned `zendeskCase.id` as the groupID, which uses `ZENDESK#` as the groupID prefix.

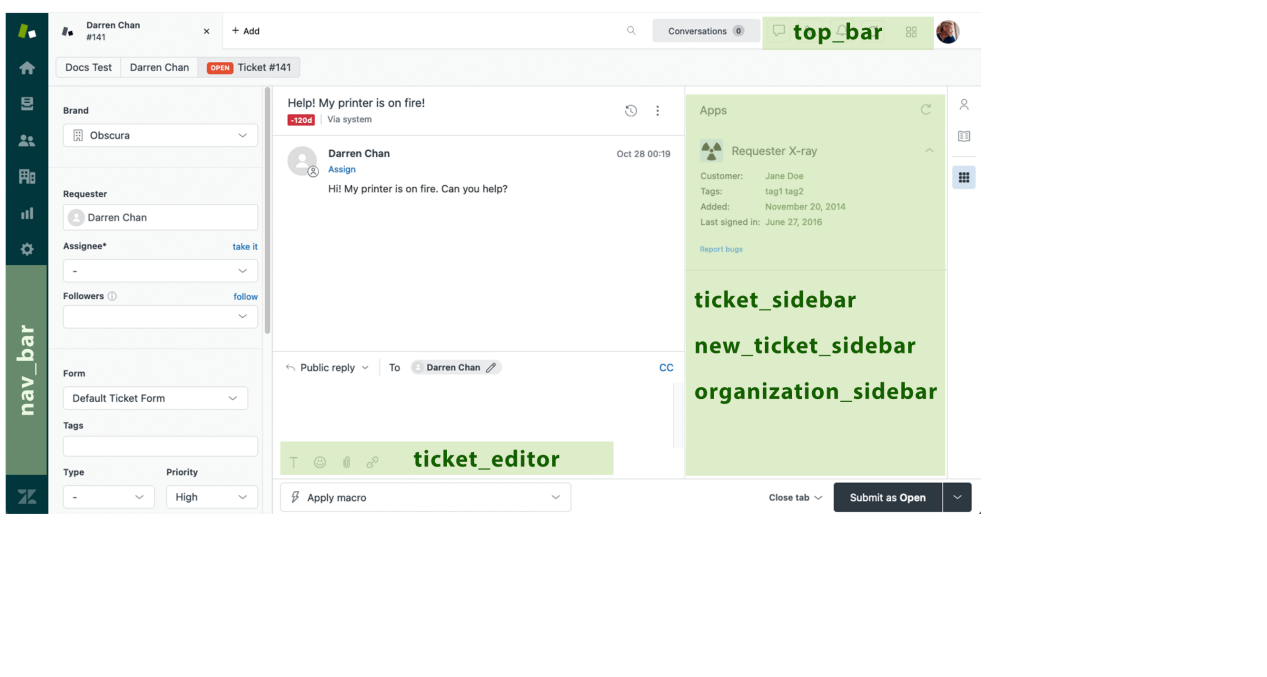
Test and publish the server code online for the client side.

TIM for Zendesk Installment

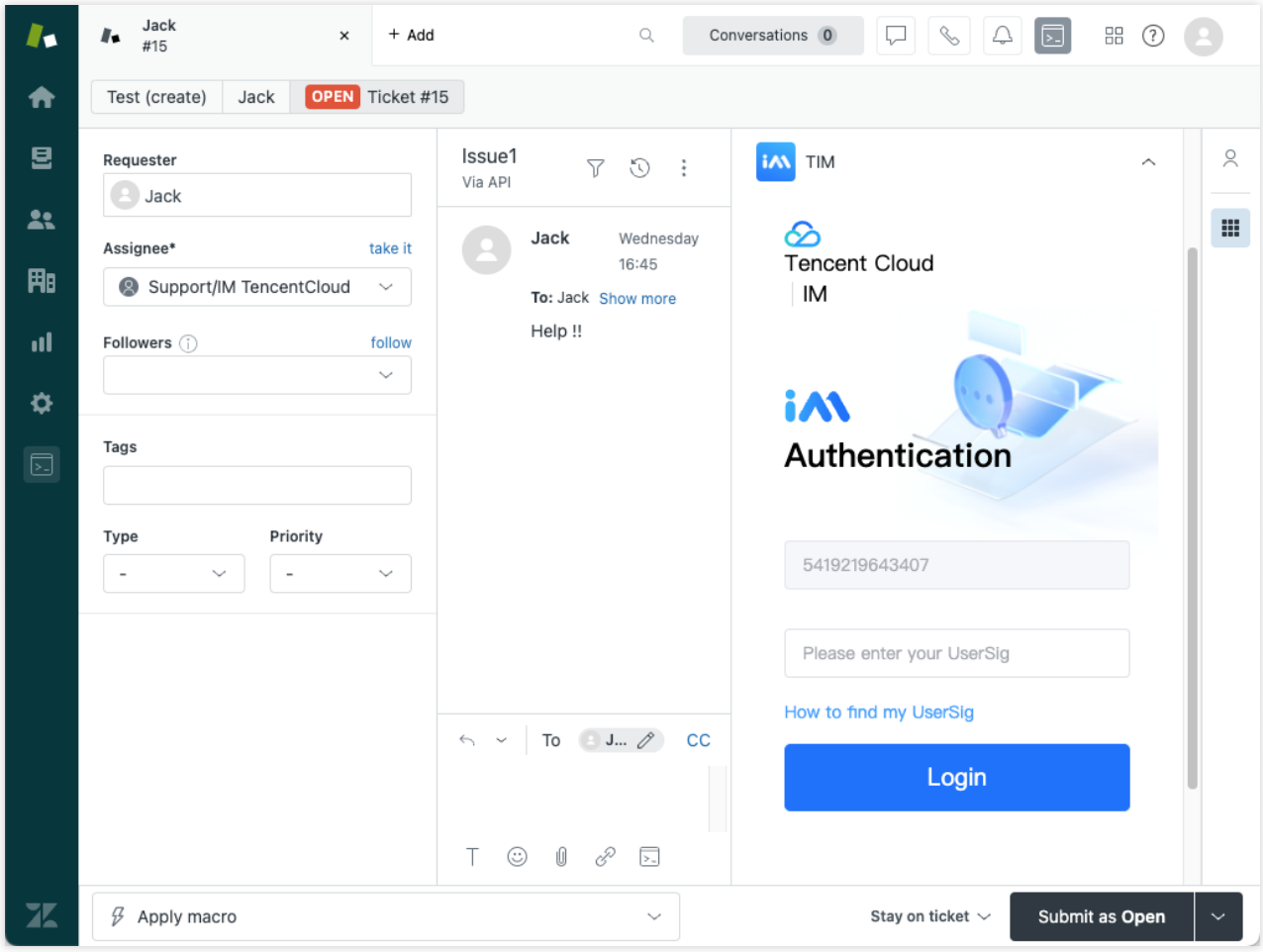
Install Tencent Cloud IM for Zendesk by searching Tencent Cloud IM for Zendesk on Zendesk marketplace.
Next, type your SDKAppId for the App to complete the installation.



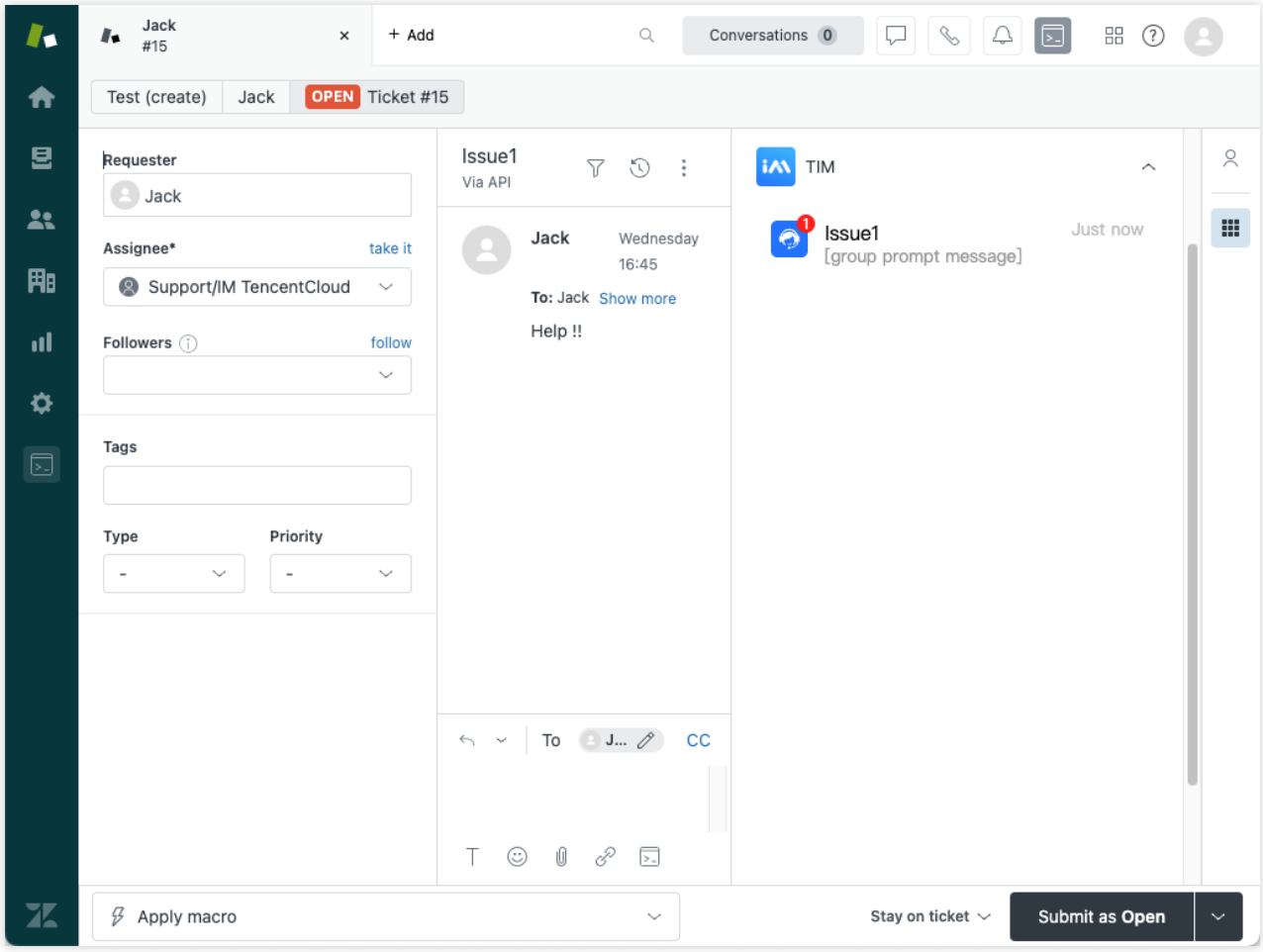
The App will show on the ticket bar area.

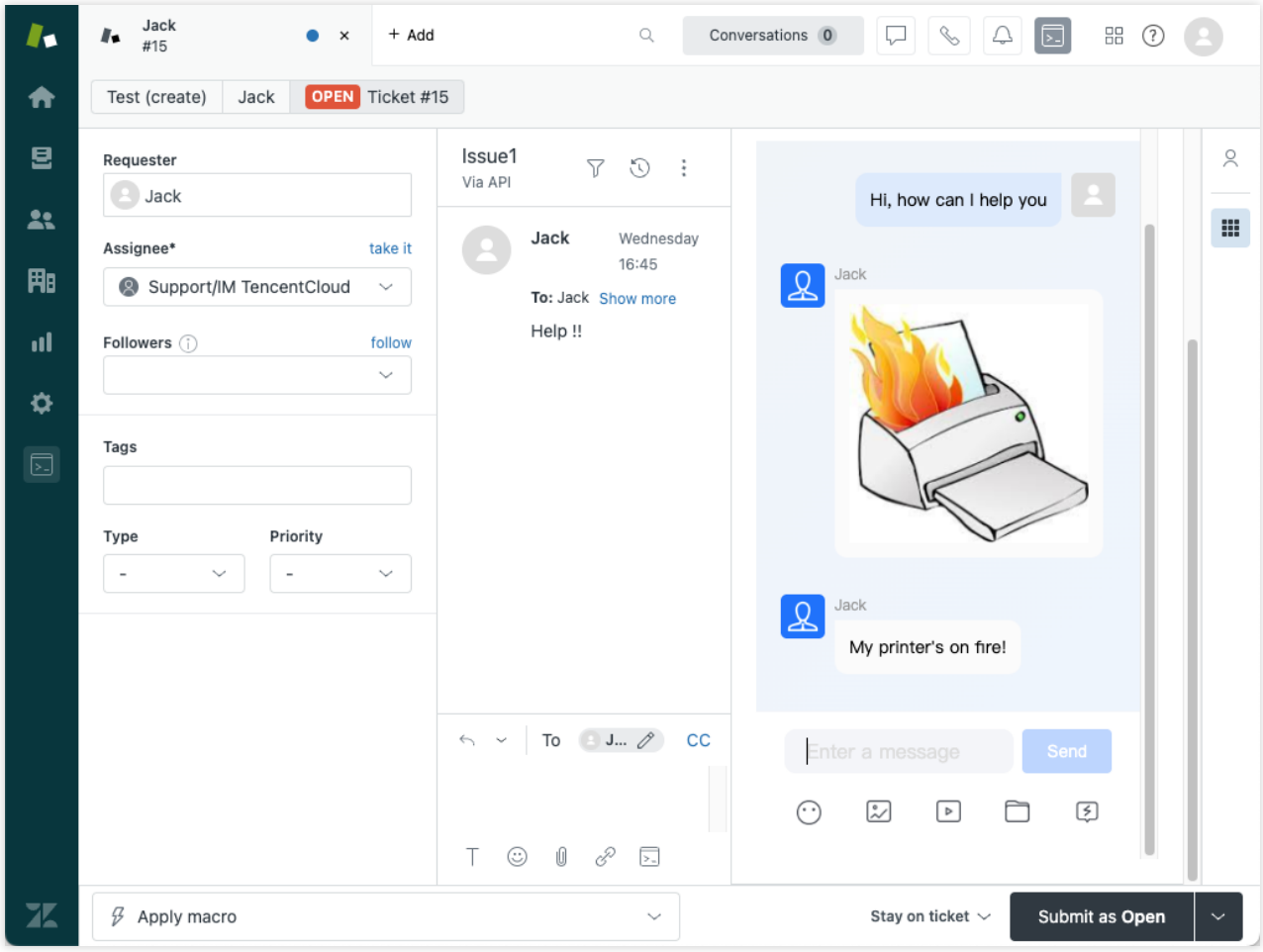


It uses agent's ID as UserID for Tencent Cloud IM, and you need to acquire UserSig for this UserID. See how to retrieve UserSig. Once logged in, the username in IM is automatically updated by the agent's name, which can be updated with other profile information on the profile page. Once logged in, you'll stay logged in until UserSig is expired or log out triggered.



And when an agent is assigned to the ticket, he/she will be invited to the IM group and chat with clients with various types of messages.


Jack
#15
+ Add
Conversations 0
Test (create)
Jack
OPEN
Ticket #15
Requester
Jack
Assignee*
take it
Support/IM TencentCloud
Followers
follow
Tags
Type
Priority
-
-
Issue1
Via API
Jack
Wednesday
16:45
To: Jack
Show more
Help !!
To
J...
CC
TIM
Issue1
[group prompt message]
Just now
Apply macro
Stay on ticket
Submit as Open



Also you can install the App privately, contact tencentcloud_im@tencent.com to get the lastest TIM package or the source code. Follow the instructions to upload private App here.

If the provided App is not what you expect, you may also develop your own private App. You may apply TUIKit to boost UI construction, and follow the instructions to create your own App. More Zendesk API References are listed here.

Conclusion

That's all you need to integrate Tencent Cloud IM with Zendesk. By now, you may have apprehended the process of the integration and the workflow of Zendesk and established your own client-agent-real-time communication App for Zendesk. If there's anything unclear or you have more thinkings about the integration with Zendesk, feel free to contact us!

How to integrate chat widget to your Shopify online store

最近更新时间：2024-02-07 17:30:52

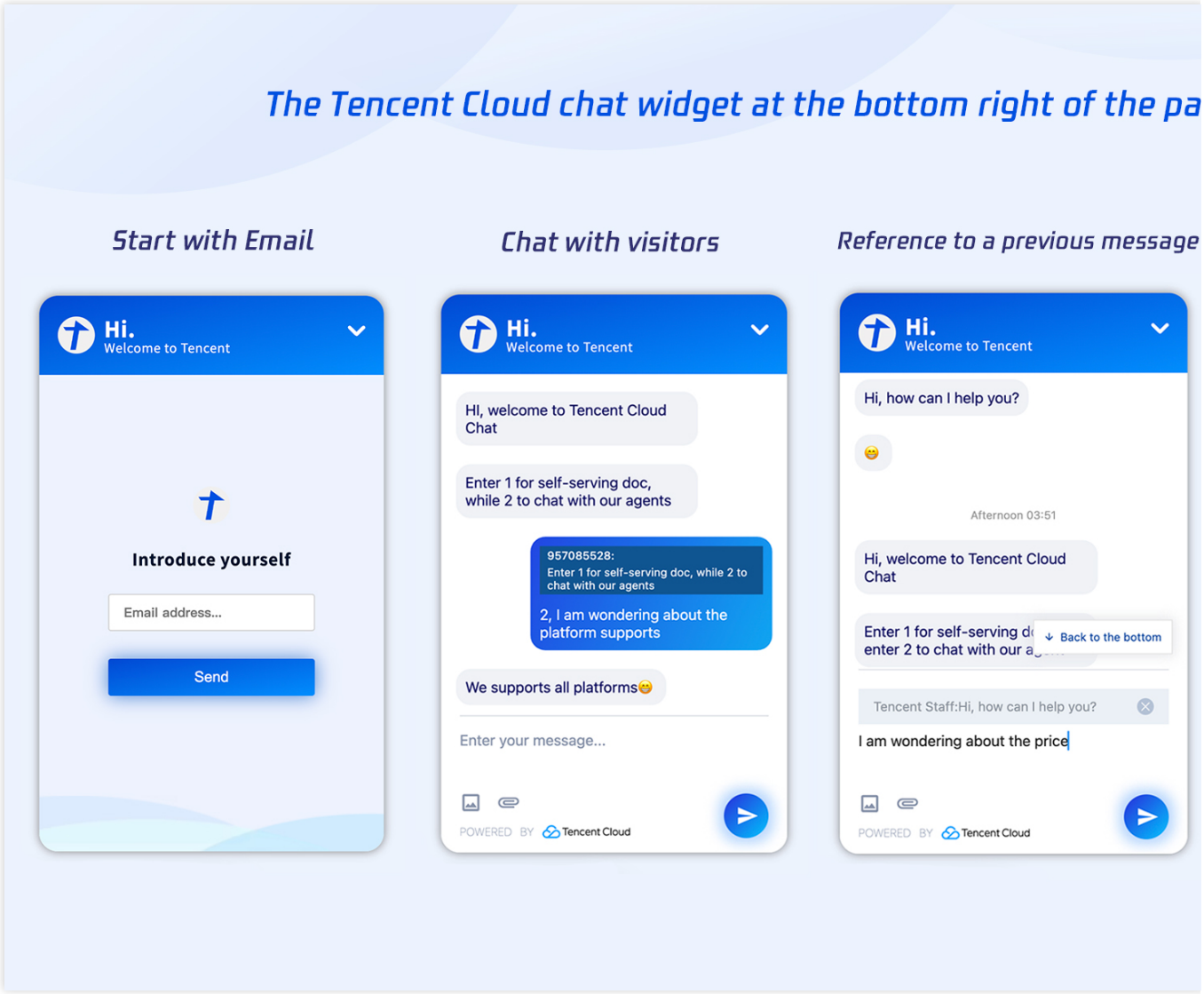
Try it now with our Demo, password is `tencentim` .

Introduction

Shopify, the world's leading e-commercial SaaS platform, with numbers of merchants building their online shops based on it globally. Meanwhile, according to Gartner, Tencent Cloud Chat became the champion in the Chinese market and one of the most competent providers in Communication Platform as a Service (CPaaS) in the global market.

Naturally, it comes to us to bring you the solution of building the chat widget on your Shopify online store based on Tencent Cloud Chat, in order to convert more online shop visitors to your customers, and deliver personal customer support at any scale no matter where they are from.

After the integration, the effect is shown below. Also, you can experience it with our Demo, password is `tencentim` .



What you'll learn

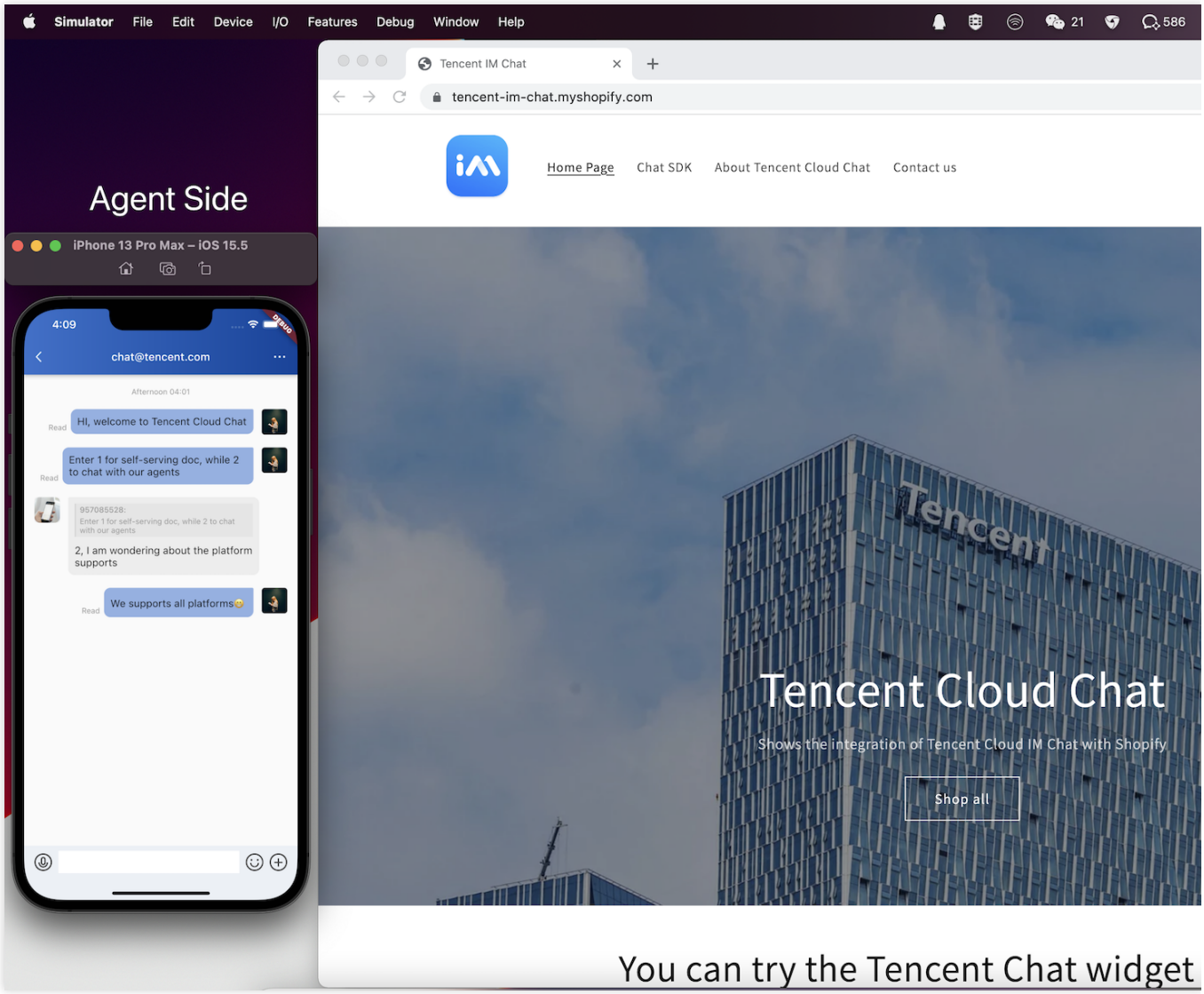
In this tutorial, we will discuss how to build this chat box on your Shopify online store with the codes we provided.

Initialize a Shopify app.

Building this app with the codes, related to the chat module, we provided.

Install this app to your Shopify store.

Enable it to communicate with your existing chat APP, for agent serving purposes.



Note:

For the agent side shows on the left of the picture above, you can implement it with our TUIKit or DEMO easily, or you can build your own APP with our Chat SDK.

Requirements

You've created a Tencent Cloud Chat APP, with the specific SdkAppID.

You've built a chat APP with the SDKs we provided, binding with this SdkAppID. It's recommended to build it based on our DEMO, if you do not have one.

You've created a Shopify Partner account and a development store.

You've installed Node.js 14.13.1 or higher.

You've installed a Node.js package manager: either npm, Yarn 1.x, or pnpm.

You've installed Git.

You're using the latest version of Chrome or Firefox.

You've installed an Online Store 2.0 theme, such as Dawn, that uses JSON templates.

You have at least Ruby 2.7.5 installed on your system.

Note:

The Shopify Partner account used in this tutorial is only for developing your own app purposes, you don't need to publicize this app to the Shopify App store.

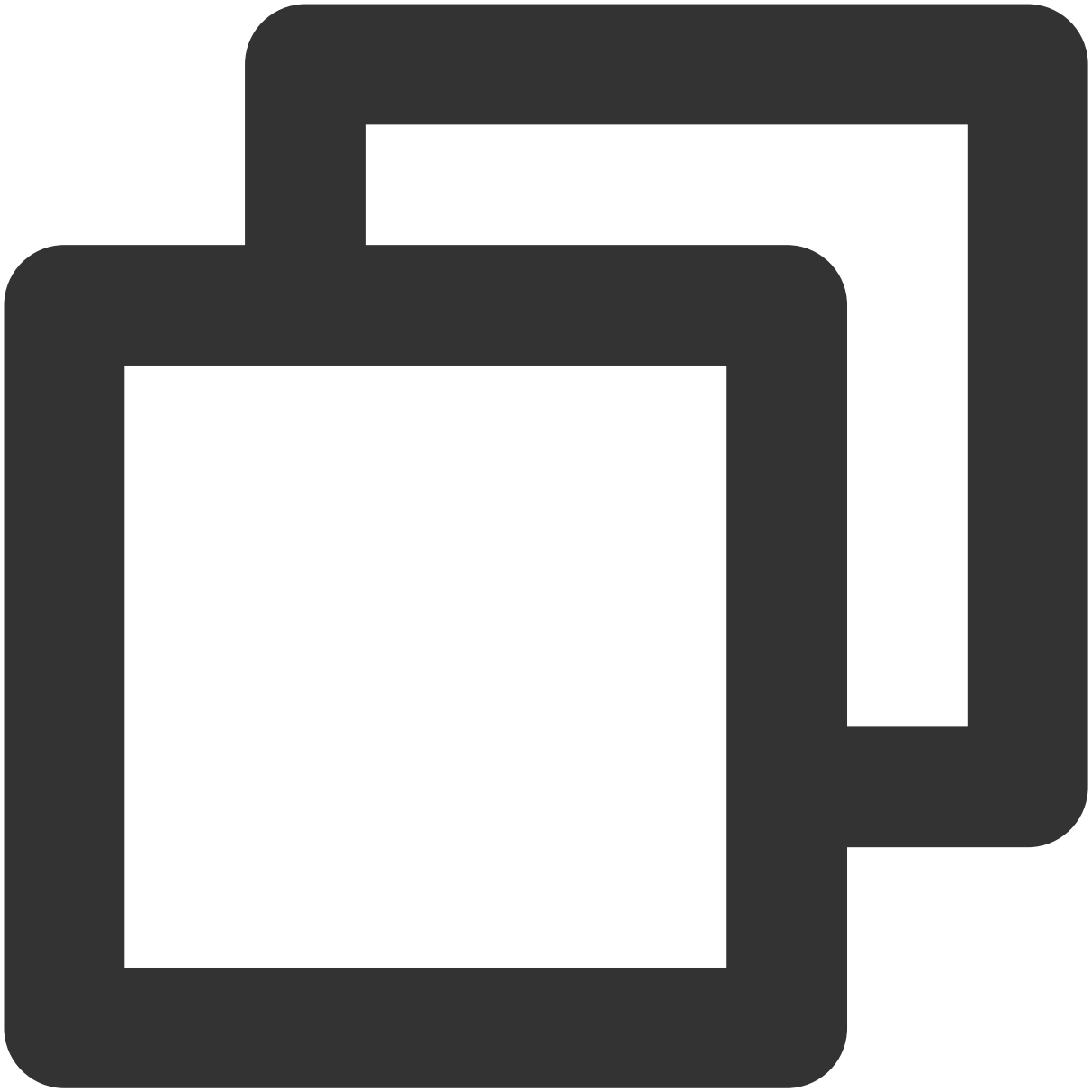
The development store is only for testing and previewing your app while developing, we will guide you to install your app to your online store by the end of this tutorial.

Let's get started

Step 1: Create a new Shopify app

You can create a new Shopify app using an npm, yarn, or pnpm command. Using npm as an example in this tutorial.

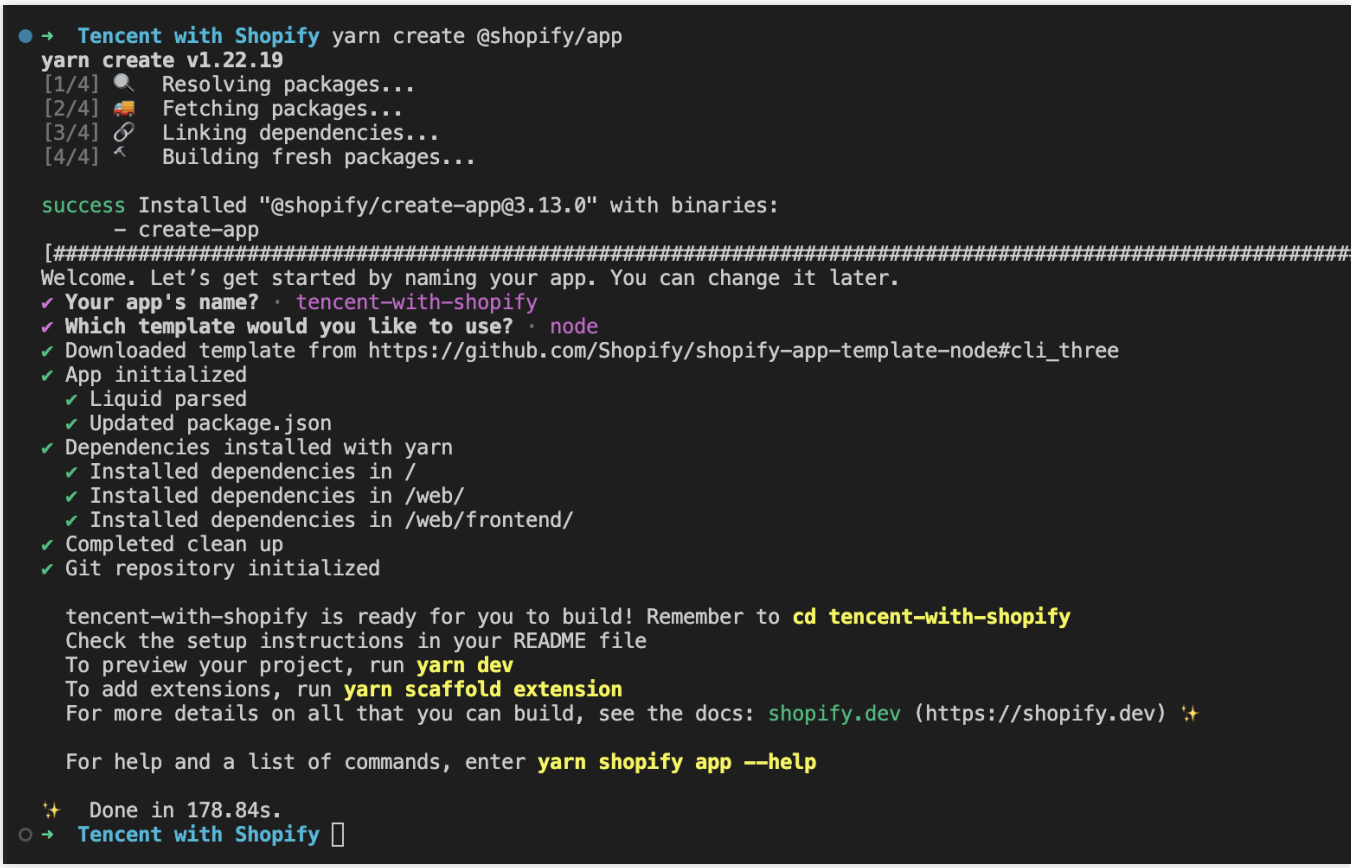
1. Navigate to the directory where you want to create your app. Your app will be created in a new subdirectory.
2. Run one of the following commands to create a new app, and select `node` as the template.



```
npm init @shopify/app@latest
```

A new app is created, and Shopify CLI is installed along with all of the dependencies that you need to build Shopify apps. Shopify CLI is also added as a dependency in your app's package.json.

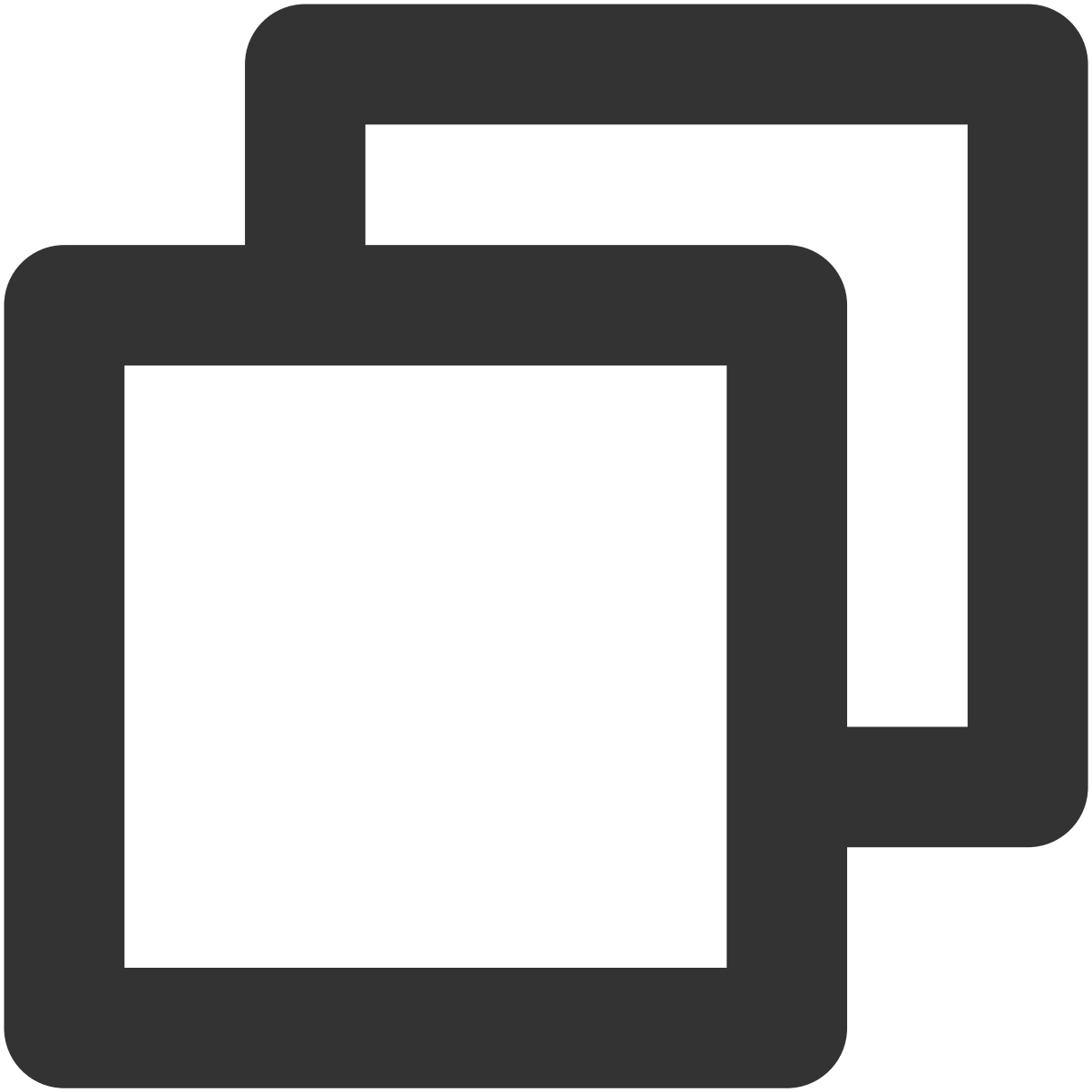
The following image shows an app being successfully created in the terminal:



Step 2: Start a local development server

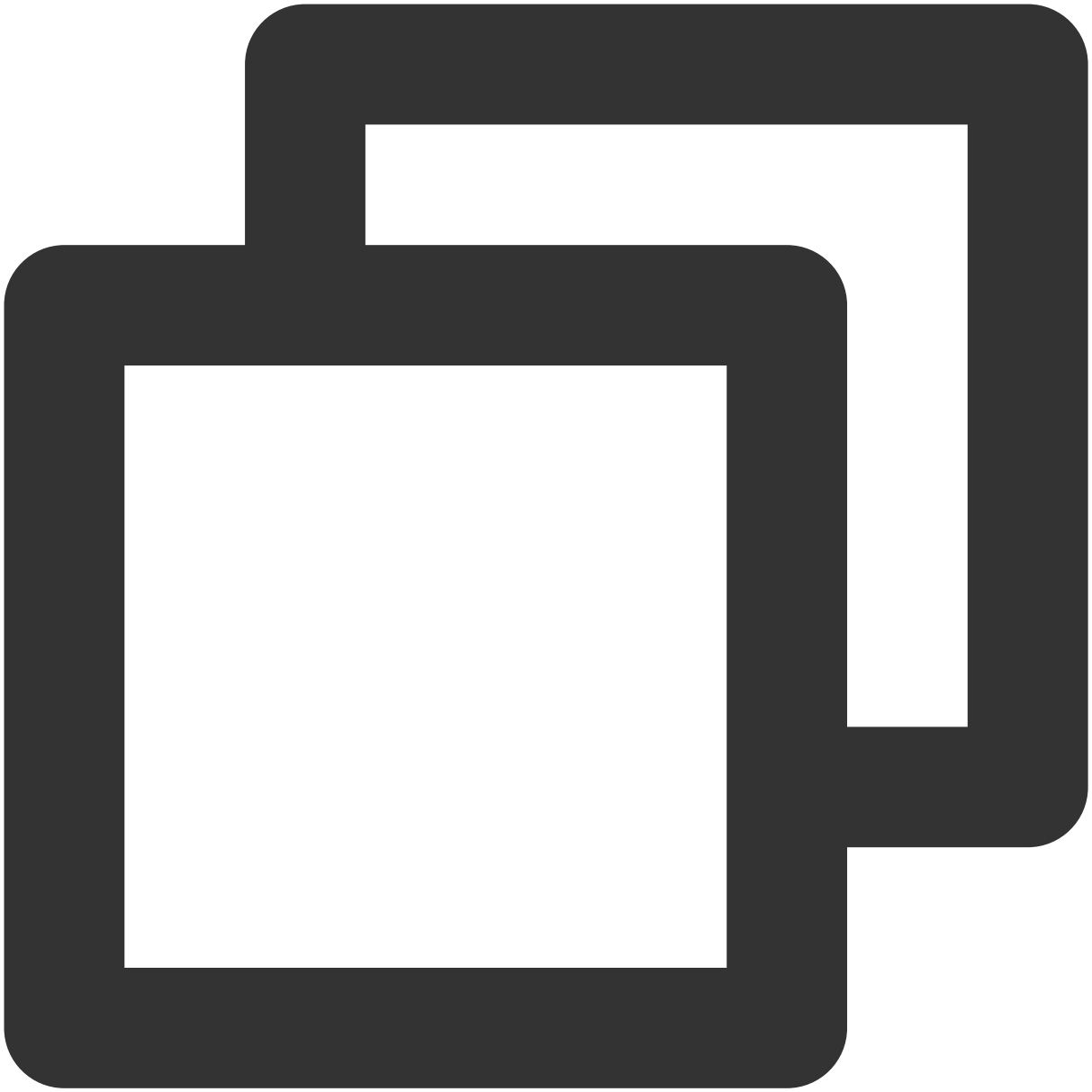
After your app is created, you can work with it by building the app and starting a local development server. Shopify CLI uses ngrok to create a tunnel that allows your app to be accessed using a unique HTTPS URL. You need to create an ngrok account and auth token to preview your app using Shopify CLI.

1. Navigate to your newly-created app directory.



```
cd your-app
```

2. To start a local server for your app, run the following command:



```
npm run dev
```

Shopify CLI walks you through the following:

Logging into your Shopify Partner account and, if needed, selecting a Partner organization

Creating an app in the Partner Dashboard, and connecting your local app files to the app

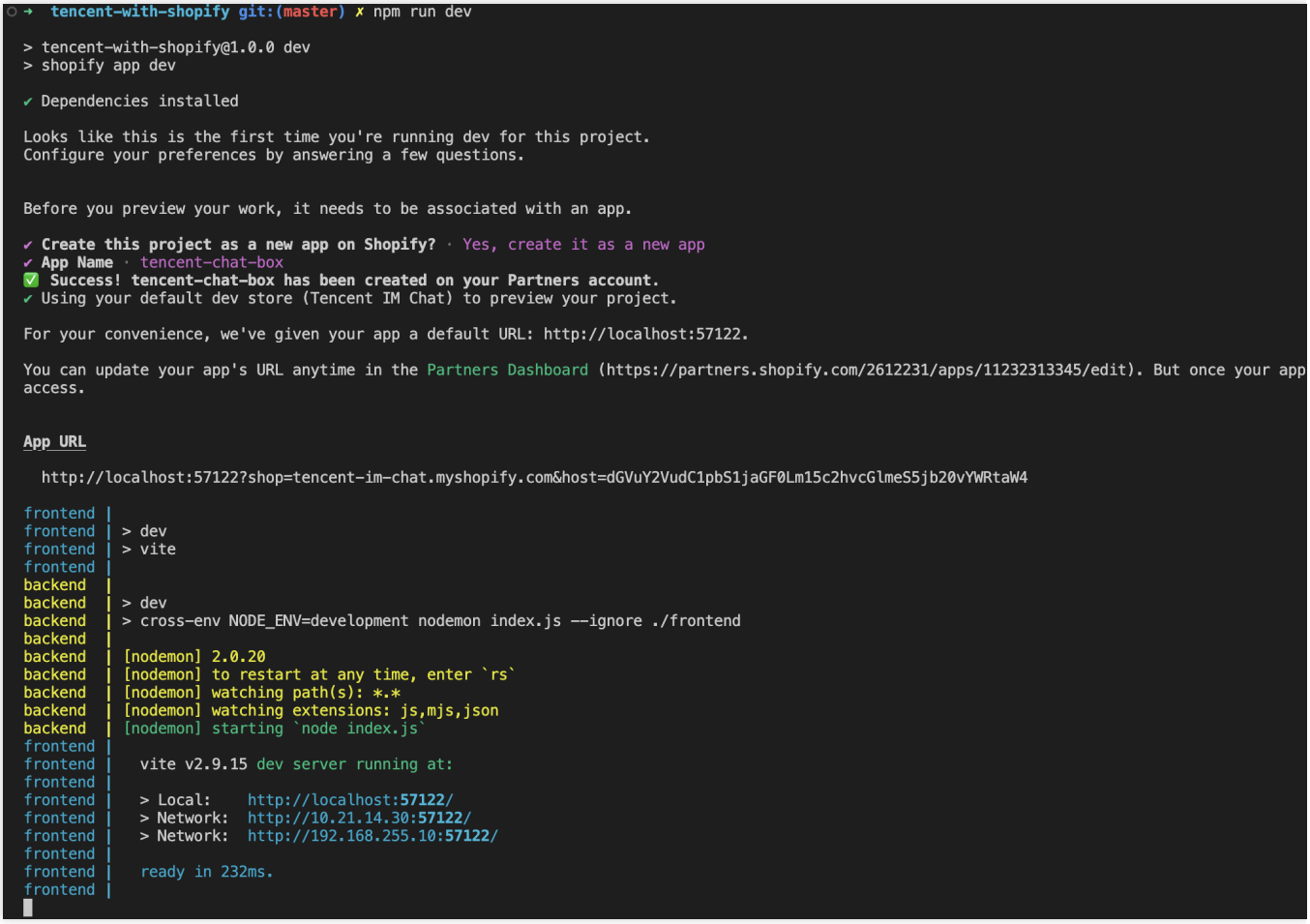
Storing your ngrok token, and then creating a tunnel between your local environment and the development store using ngrok

Note:

If you encounter ngrok errors in your browser when you try to preview your app or app extension, then consider using a tunnel created with your own tunneling software instead. Shopify recommends using Cloudflare Tunnel.

To use your own tunnel, pass your tunnel URL and port to the `dev` command with the --tunnel-url flag.

The following image shows a development server being started using `dev` :



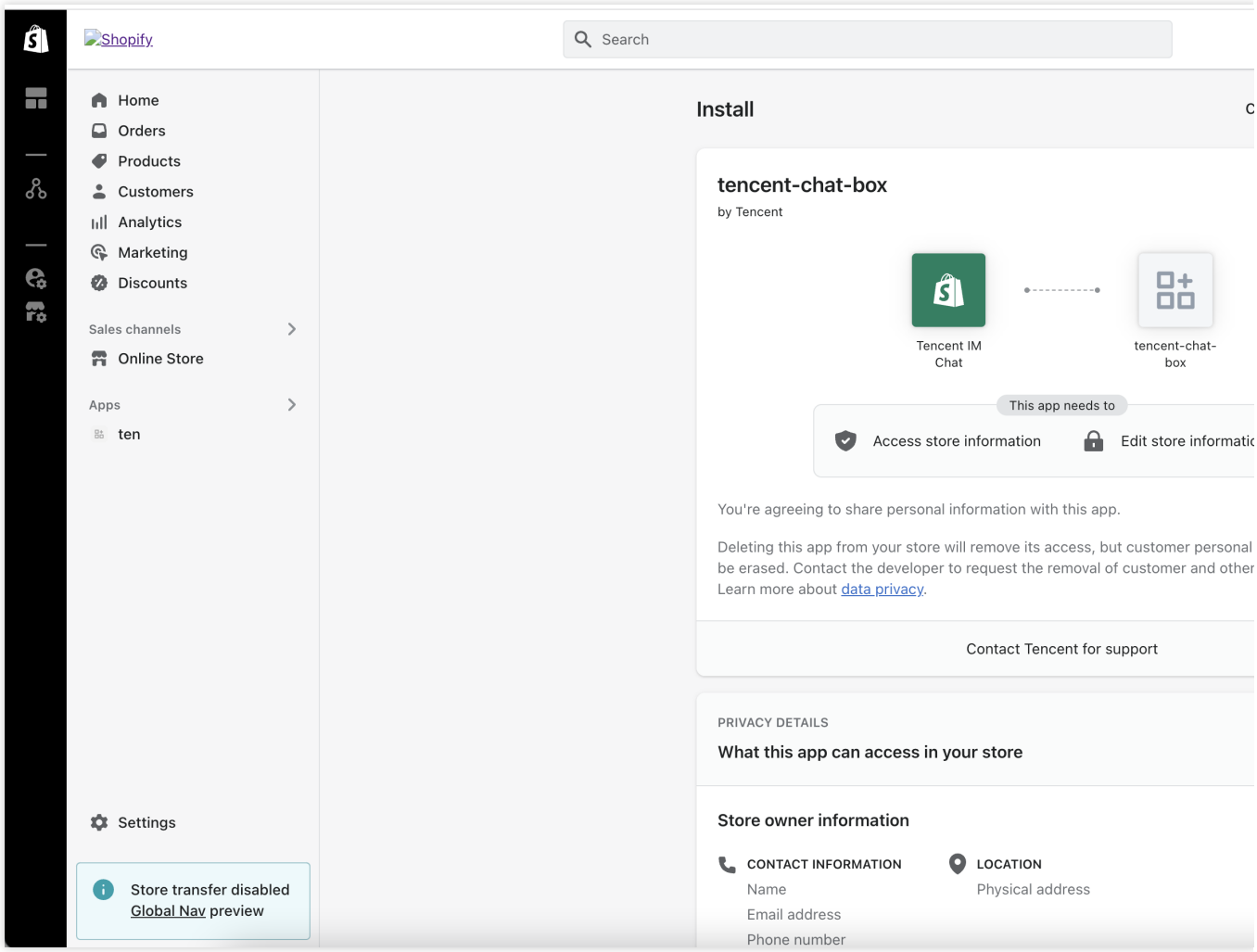
Step 3: Install your app on your development store

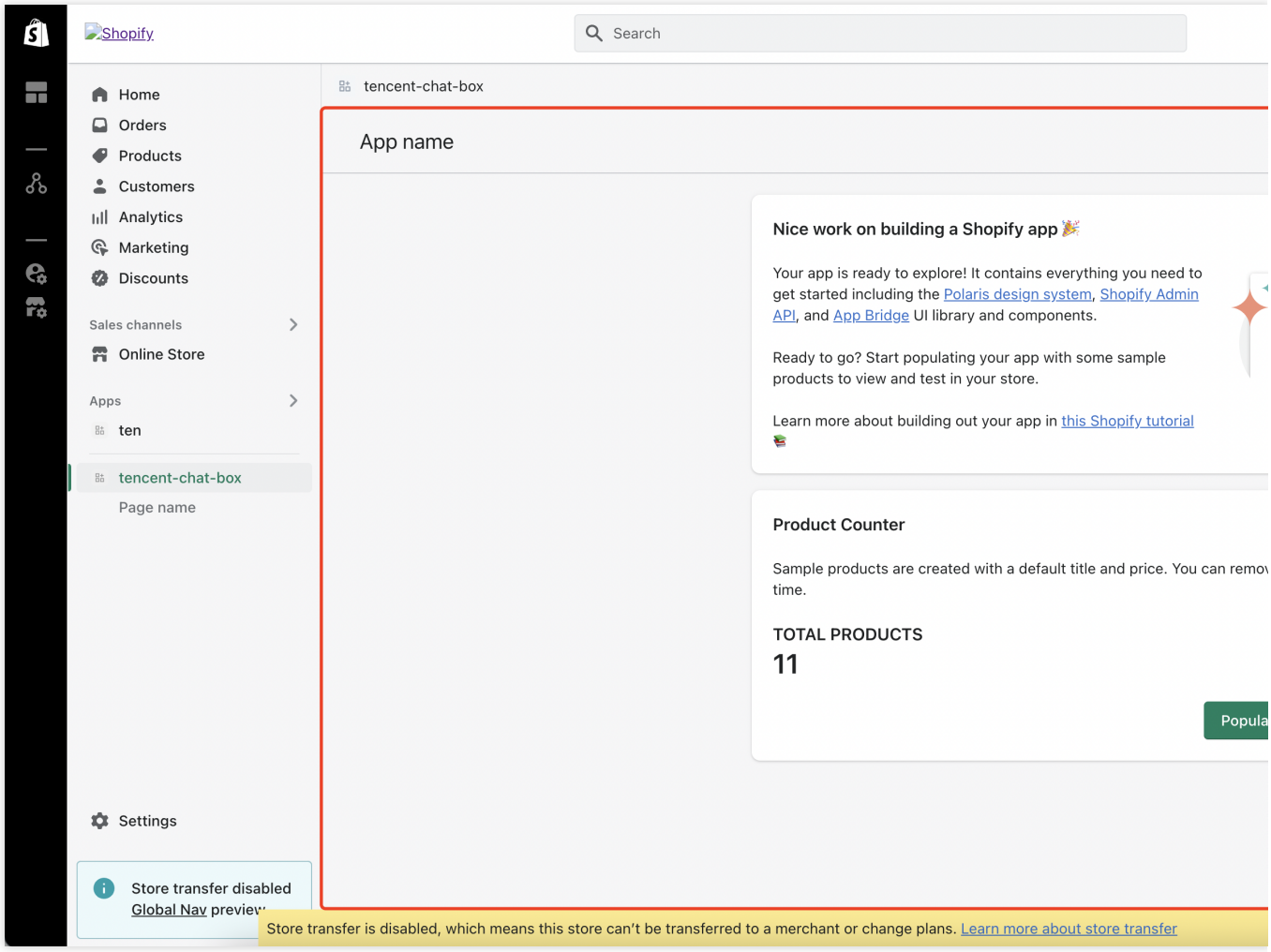
With the server running, open the URL in the App URL section of the terminal output in the previous step.

The URL follows the format `https://[tunnel_url]?shop=[dev_store].myshopify.com&host=[host]` , where `[host]` is the base64-encoded host parameter used by App Bridge, and represents the container the app is running in.

When you open the URL, you're prompted to install the app on your development store.

Click **Install app** to install the app in the store.





Step 4: Modify the frontend code in this admin dashboard

Note:

This step is optional, mainly to replace the default app page on admin with the configuration guidance page.

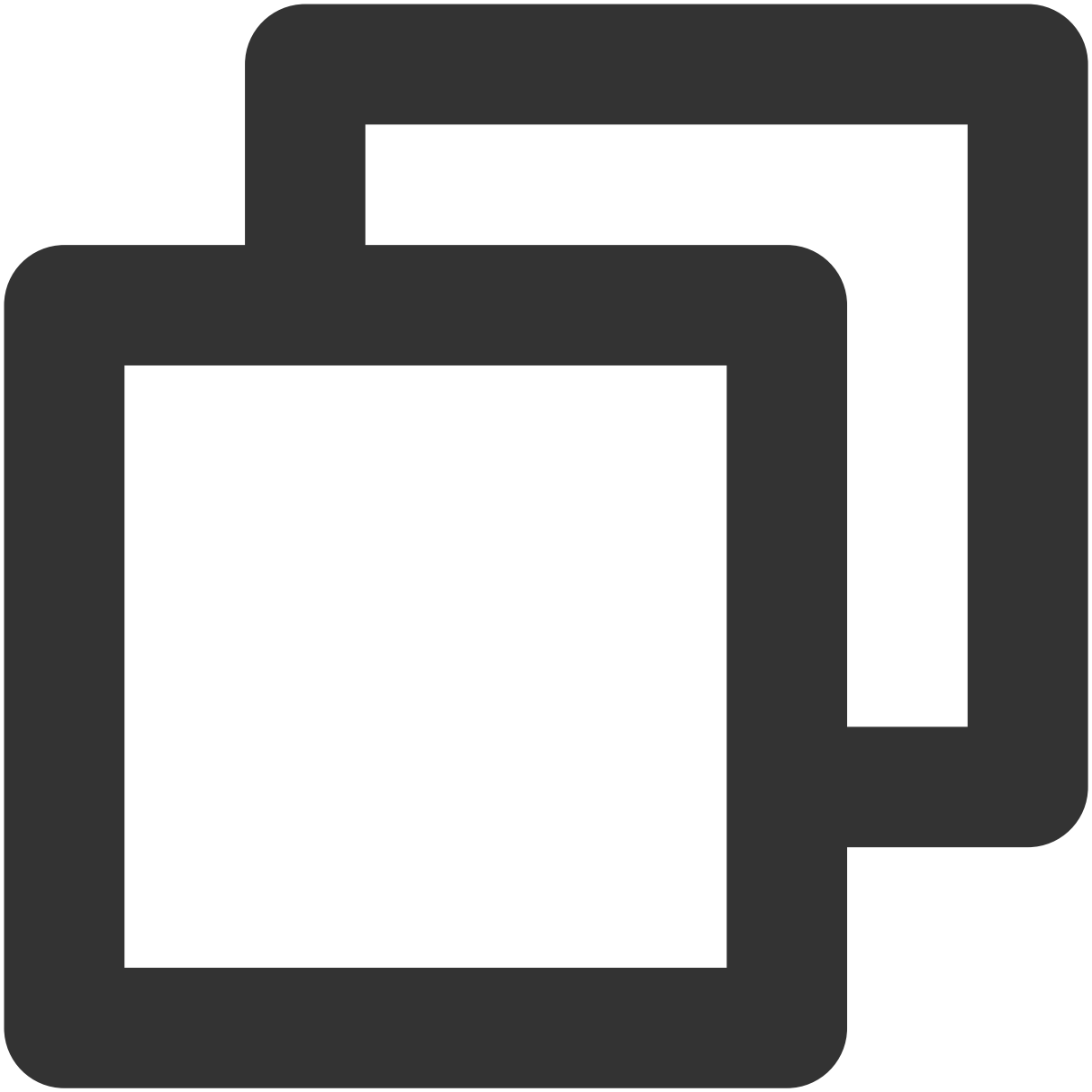
1. Replace the `web/frontend/pages/index.jsx` file.

Download the new file, named it as `index.jsx` and replace the file of `web/frontend/pages/index.jsx`.

2. Add the guide page.

Download the file, named it as `Guide.jsx`, and move it into `web/frontend/components/Guide.jsx`.

3. Add a line of export class to `web/frontend/components/index.js`.



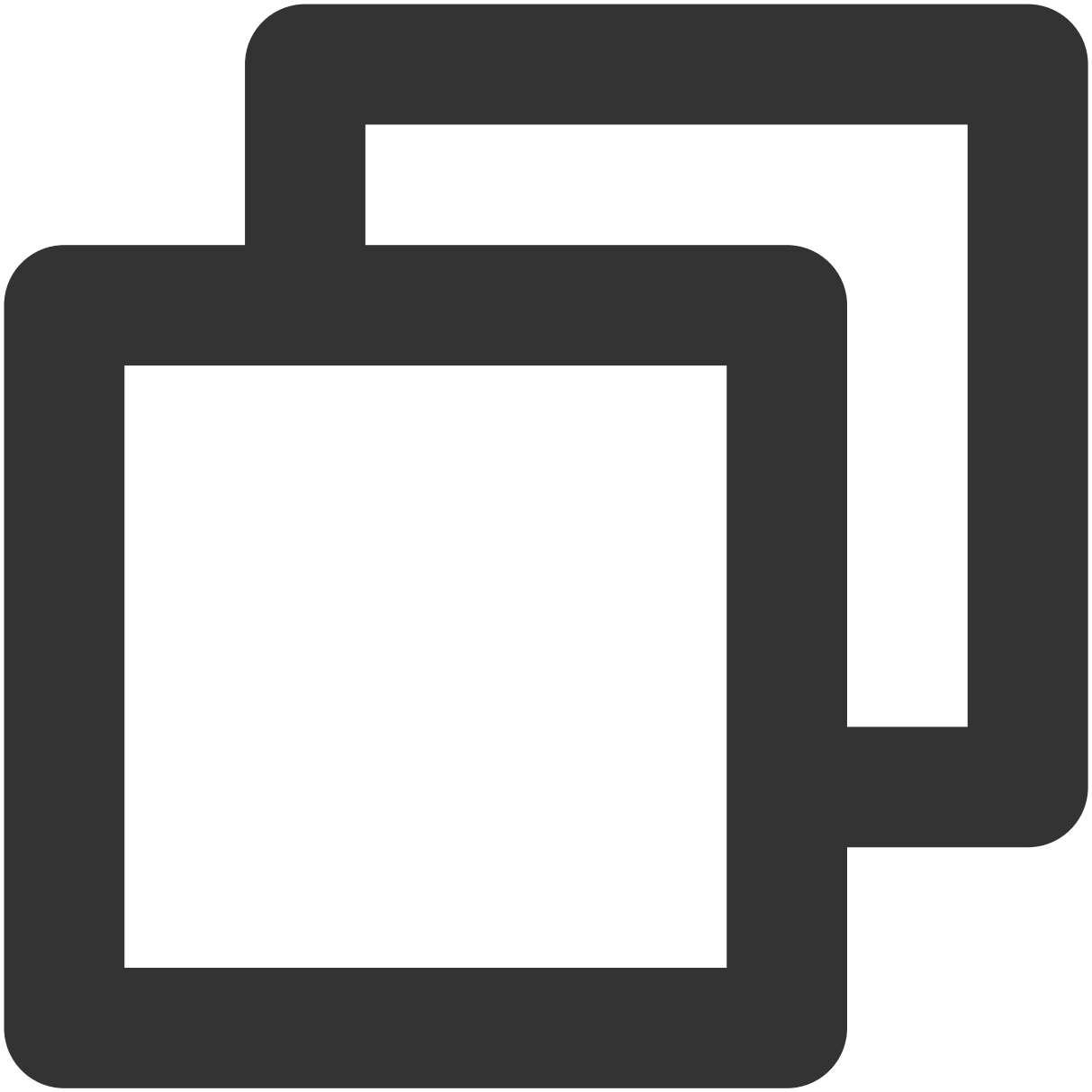
```
export { Guide } from "./Guide";
```

4. Add the file resources to `web/frontend/assets` .

Logo, named as `logo.png` .

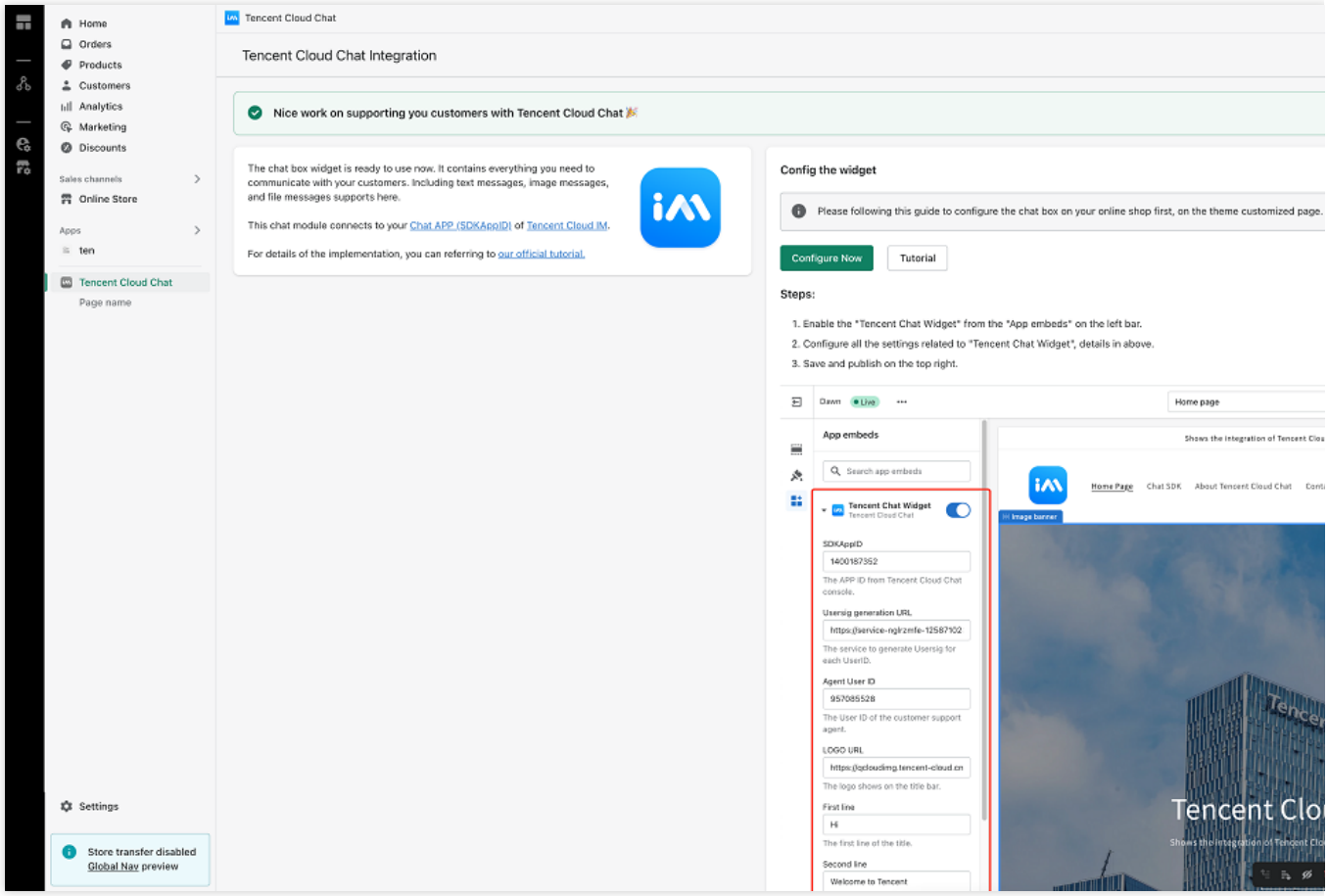
Demo instructions, named as `demo.png` .

Add the export to `web/frontend/assets/index.js` .



```
export { default as logo } from "./logo.png";
export { default as demoImage } from "./demo.png";
```

Now, the updating of the admin page is finished, you can refresh the page and see the following page.



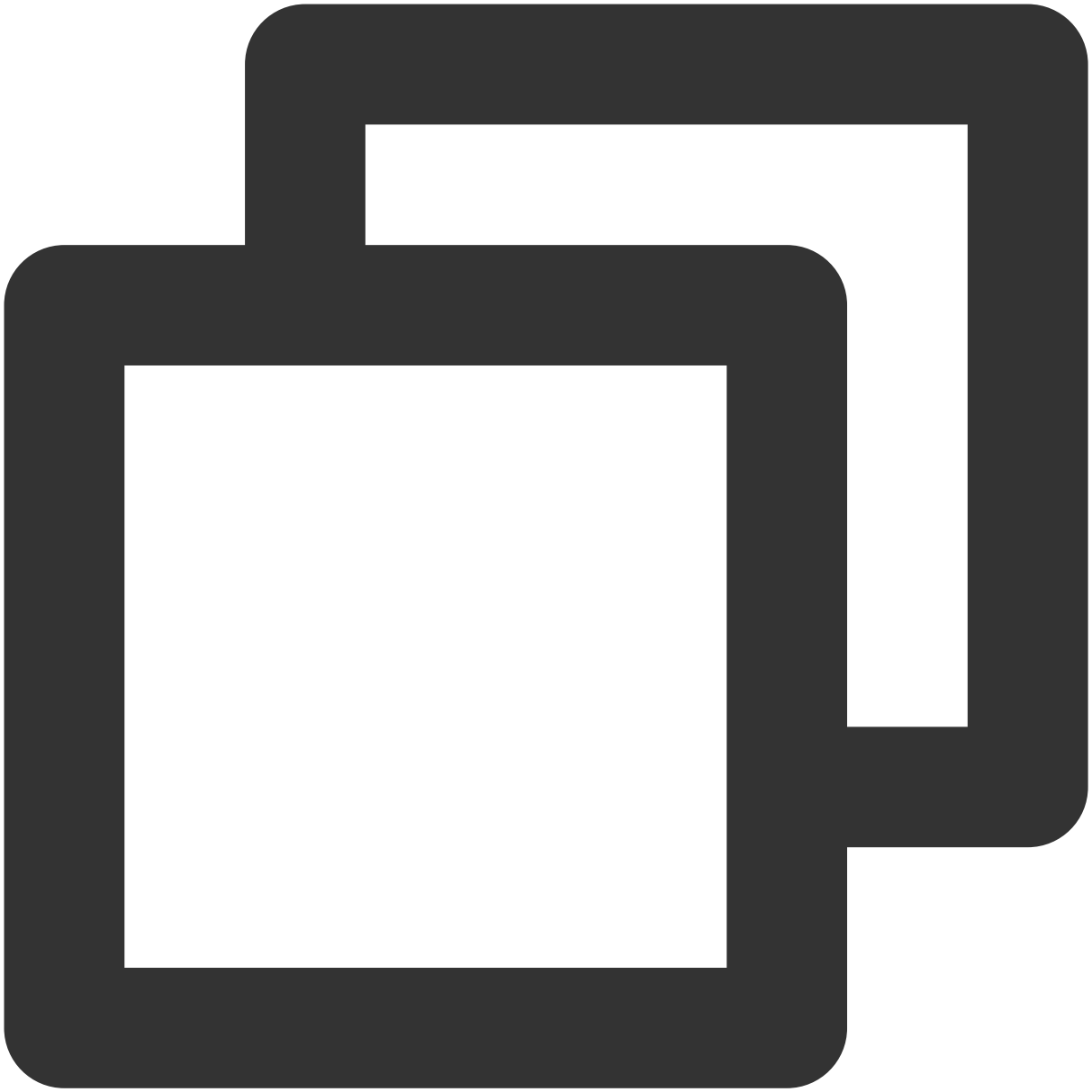
Step 5: Create a new extension

This extension will be the chat box widget shown at the bottom right of the online store.

You'll use Shopify CLI to generate a new extension first.

1. Navigate to the directory of the app that you want to add your extension to.

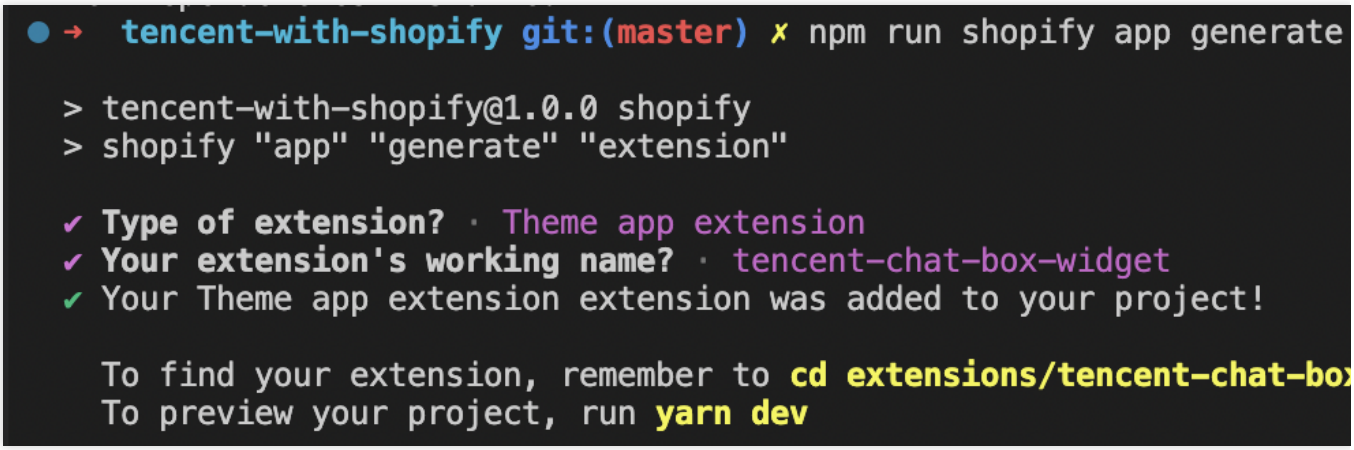
2. Run the following command to start creating the extension, choose `Theme app extension` as the `Type of extension` .



```
npm run shopify app generate extension
```

Note:

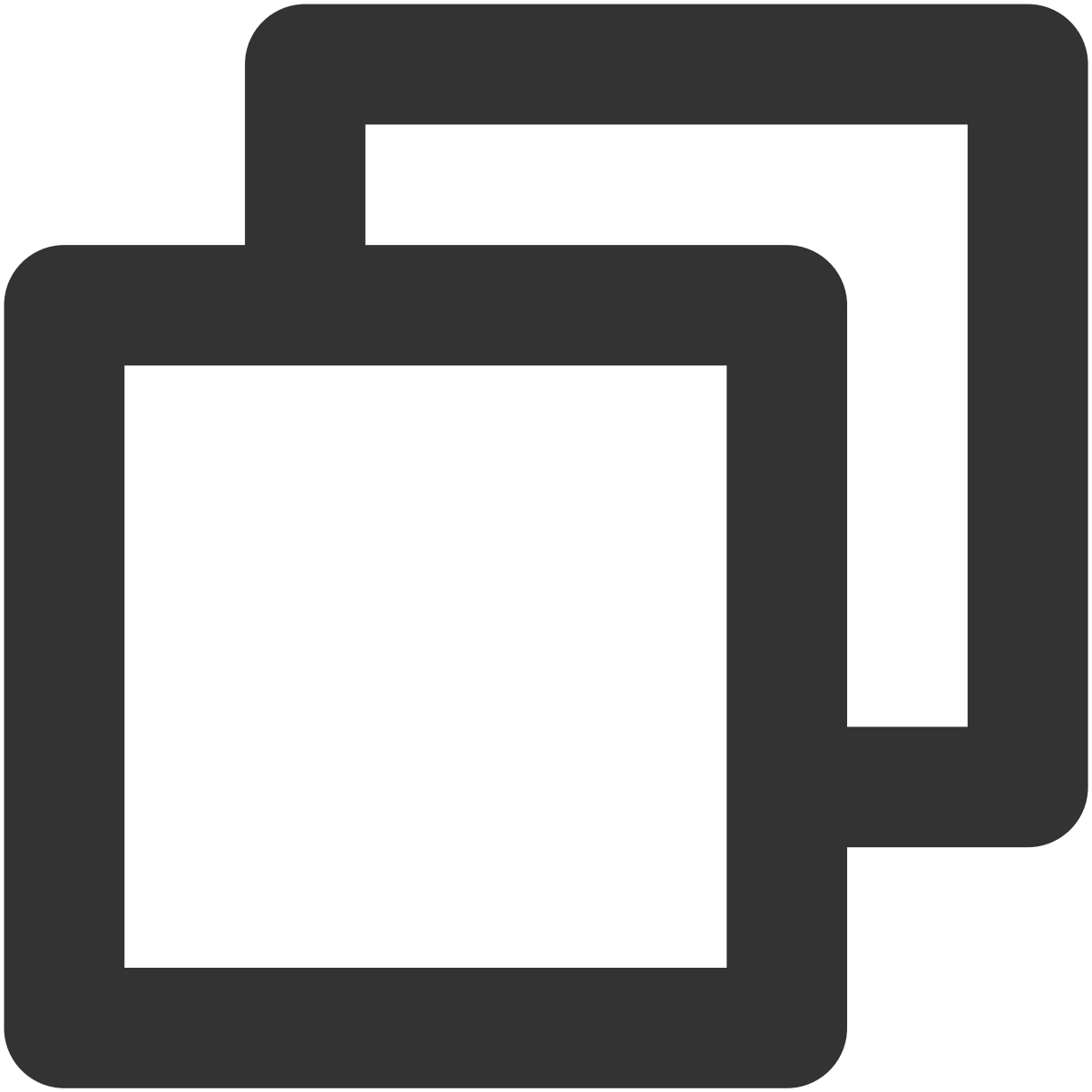
Please make sure you use Shopify CLI 3.0 or higher, if this command does not work.



After your theme app extension is created, you can preview your changes in real time by starting a local development server.

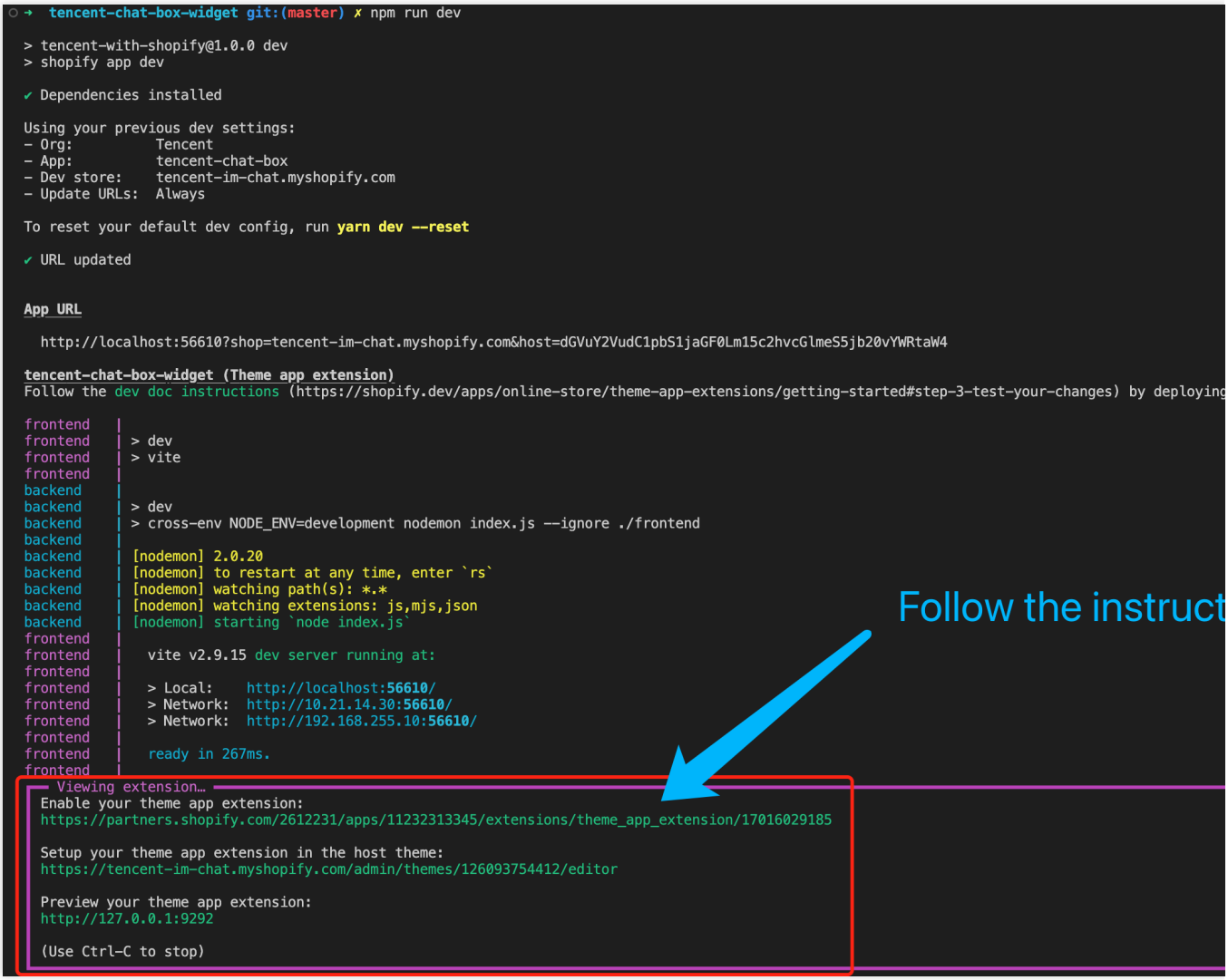
The `dev` command returns a preview URL that reloads local changes, allowing you to preview changes in real time using the store's data. This preview is only available in Google Chrome.

1. Navigate to your app directory.
2. Run one of the following commands:



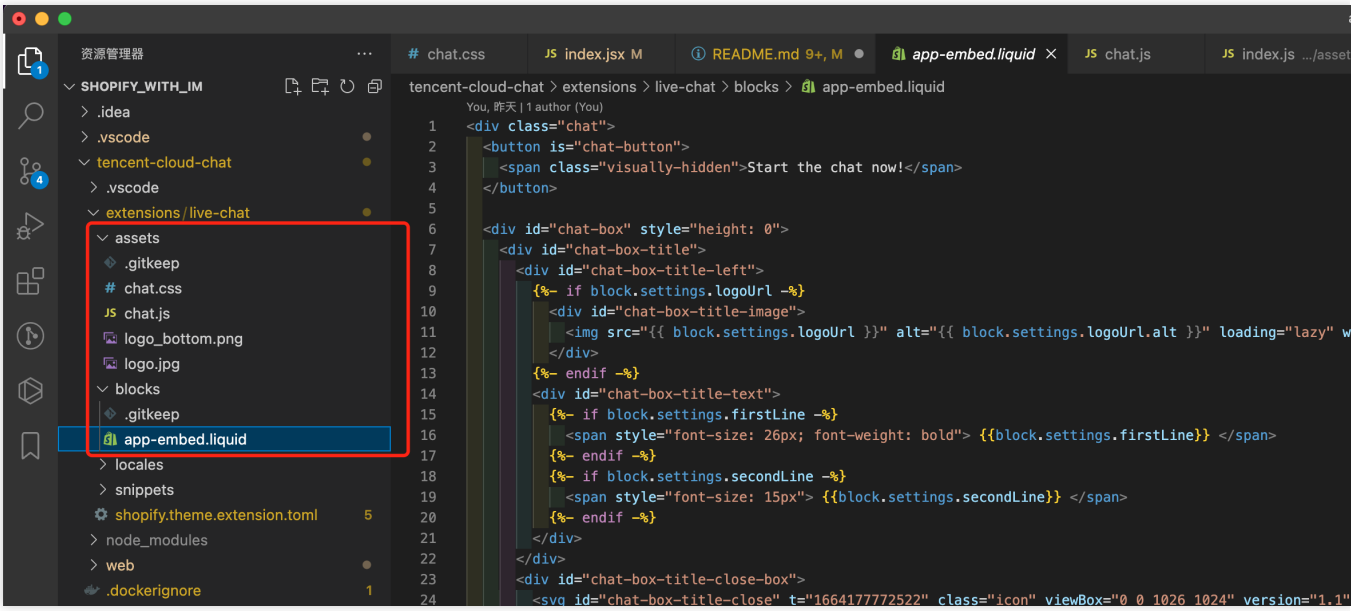
```
npm run dev
```

3. Follow the instructions in the CLI output to enable your theme app extension and set it up in the host theme.
4. Click the URL that's printed at the bottom of the CLI output to preview your extension.



Step 6: Modify the code of extension to build a chat box widget

1. Download our sample code package.
2. Unzip it.
3. Move and replace the `assets` and `blocks` directory to `extensions/(your extension directory)` , and it will be like:



Step7: Start up a server to generate UserSig for each UserID

As you can see in our DEMO (password is `tencentim`), visitors are supposed to start a chat with their email address, and this field will be the user ID for you app of Tencent Cloud Chat.

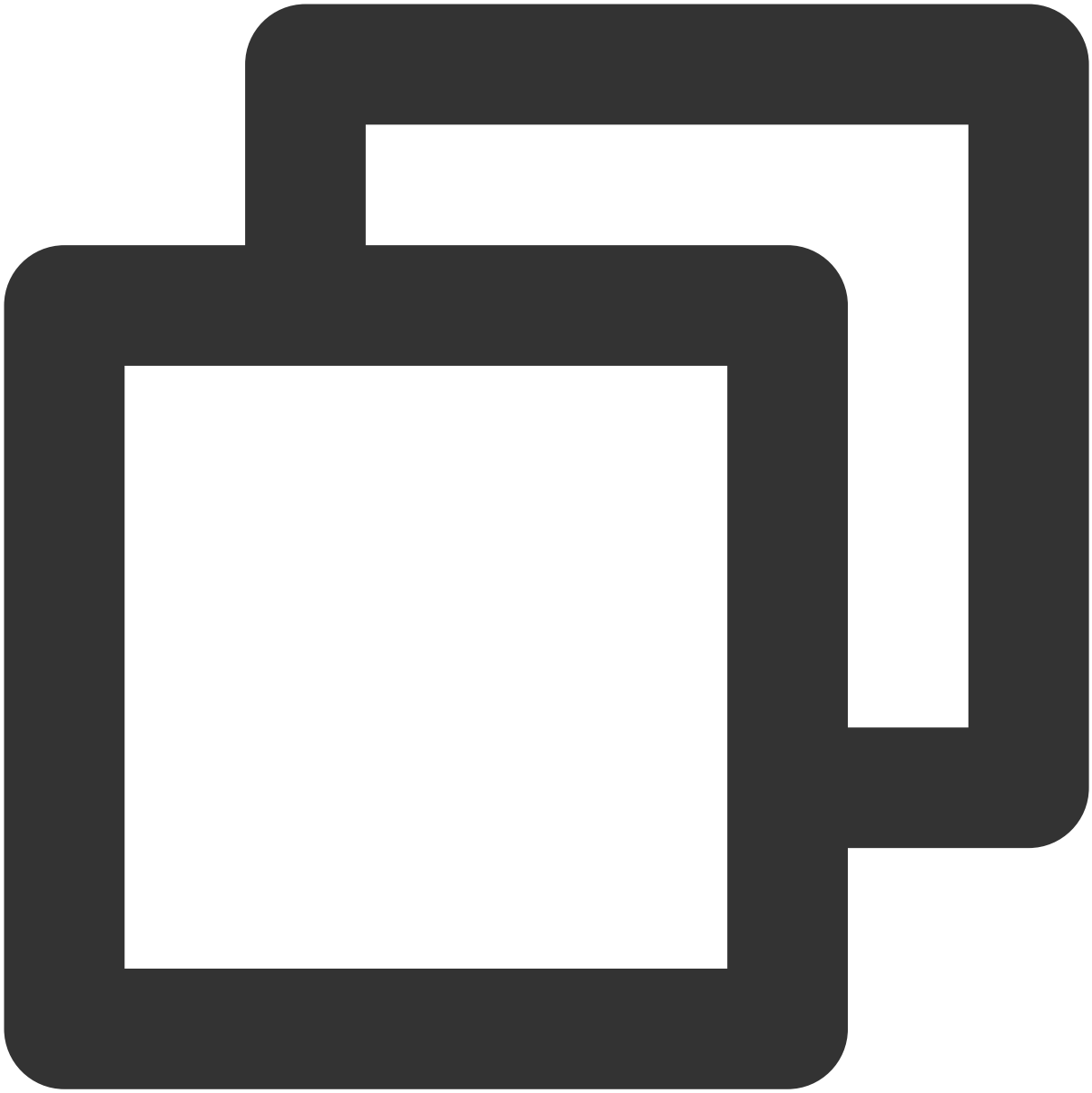
Both `UserID` and `UserSig` are needed for login before chatting, while the `UserSig` is generated with `UserID`, `SECRETKEY` and `SDKAppID` dynamically.

As a result, you should start up a server to generate `UserSig` from `UserID`, while the `SECRETKEY` and `SDKAppID` have been stored on your server for safety purposes.

You could build this service by yourself or using the serverless template we provided.

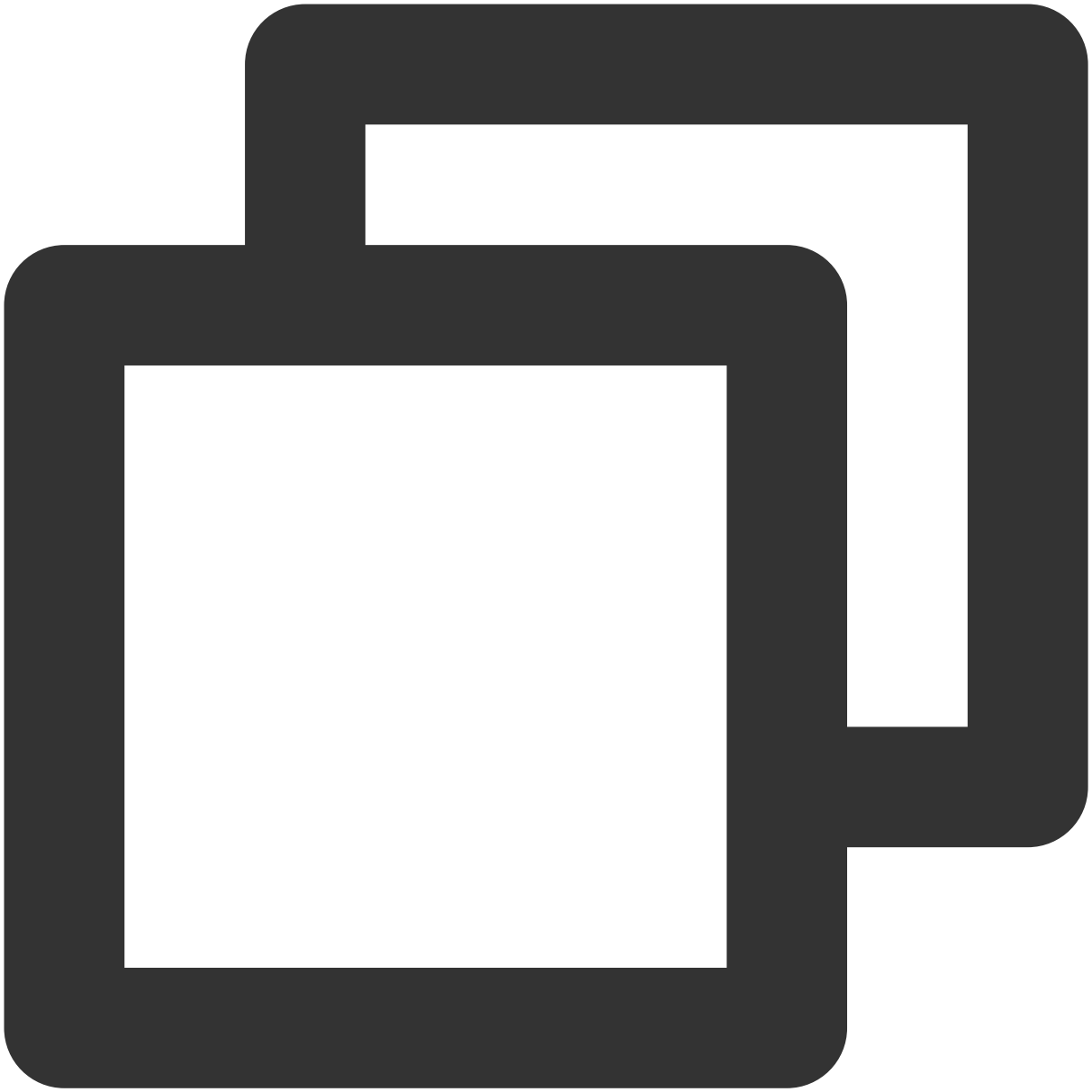
The communication between the chat box and your server should use RESTful API, with POST request.

The following shows request parameters from the chat widget. The `Content-Type` is `application/json`.



```
{
   "userID": "The User ID"
}
```

The response body from your server should be like following:

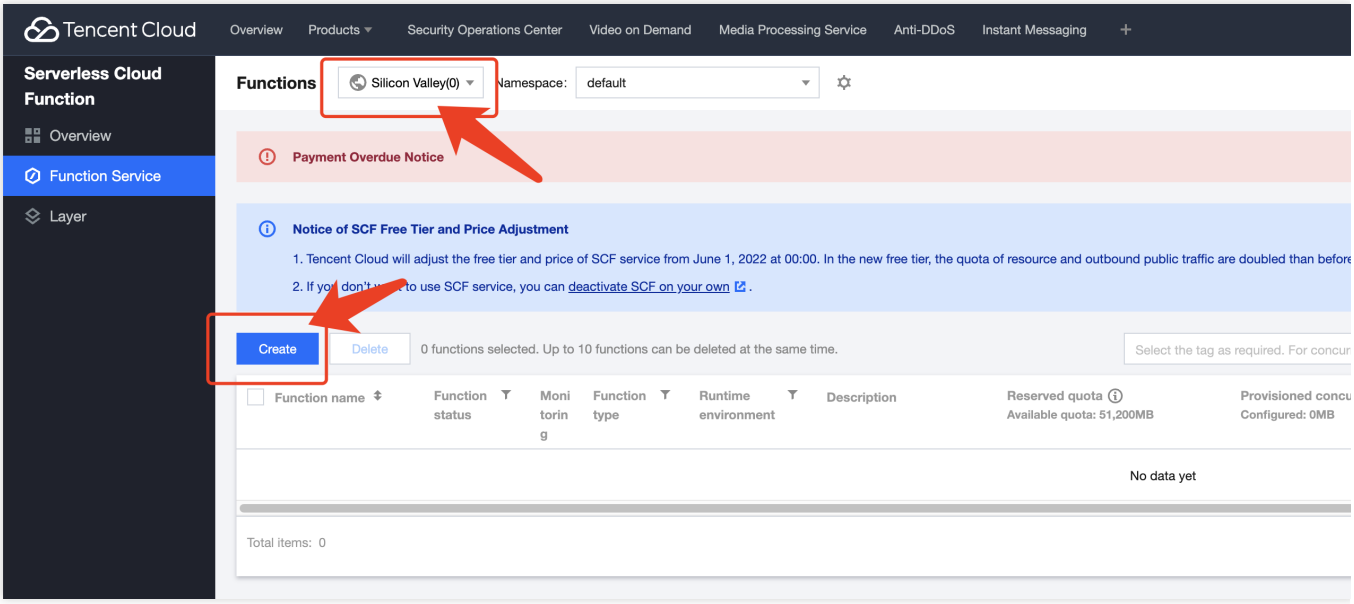


```
{
   "userID": "",
   "userSig": ""
}
```

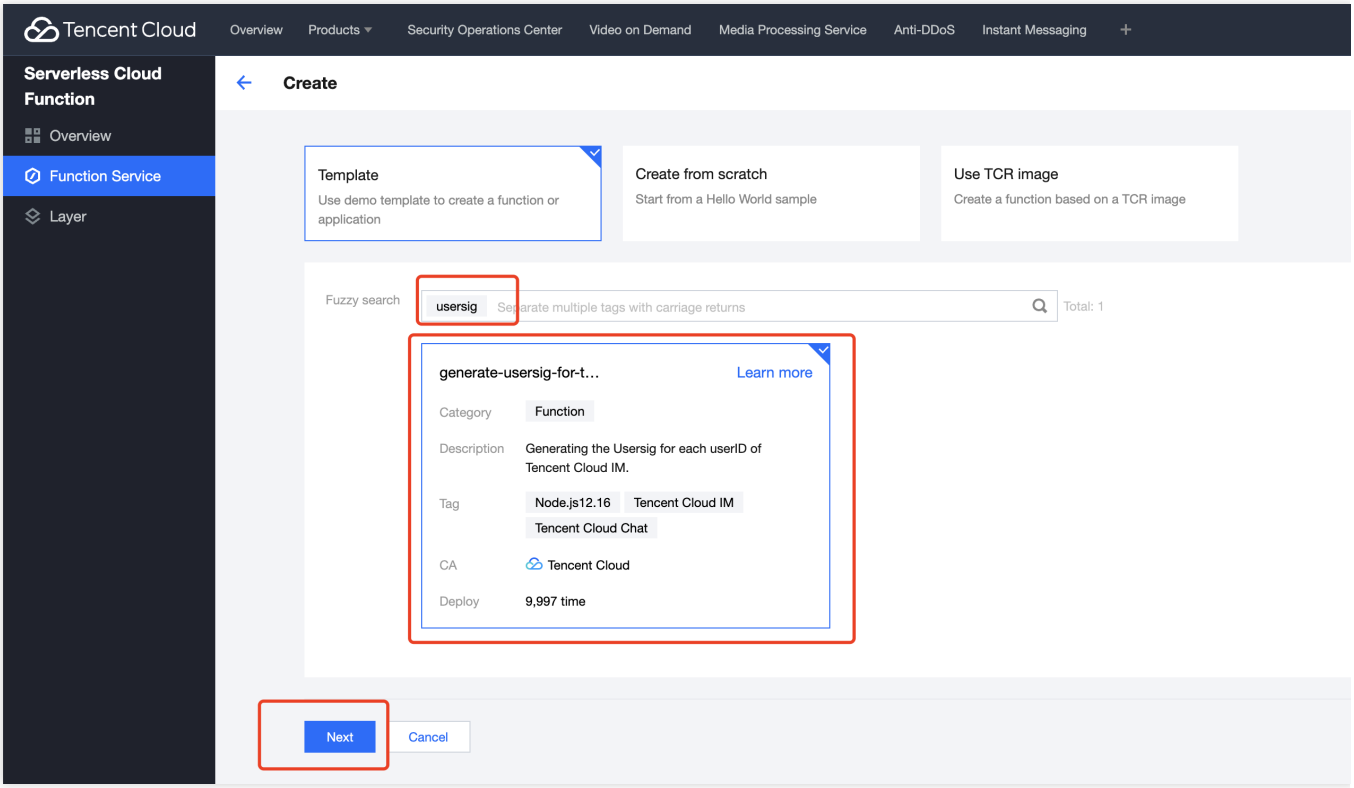
Set up the `UserSig` generation service with serverless function

You are also encouraged to use our template to create a serverless function on Tencent Cloud, if you find complicating to build such a `UserSig` generation service.

1. Navigate to the console of Tencent Cloud Serverless Cloud Function, choose a region and click `Create` .



2. Search by keyword, `usersig` , choose the following one, then click `Next` .

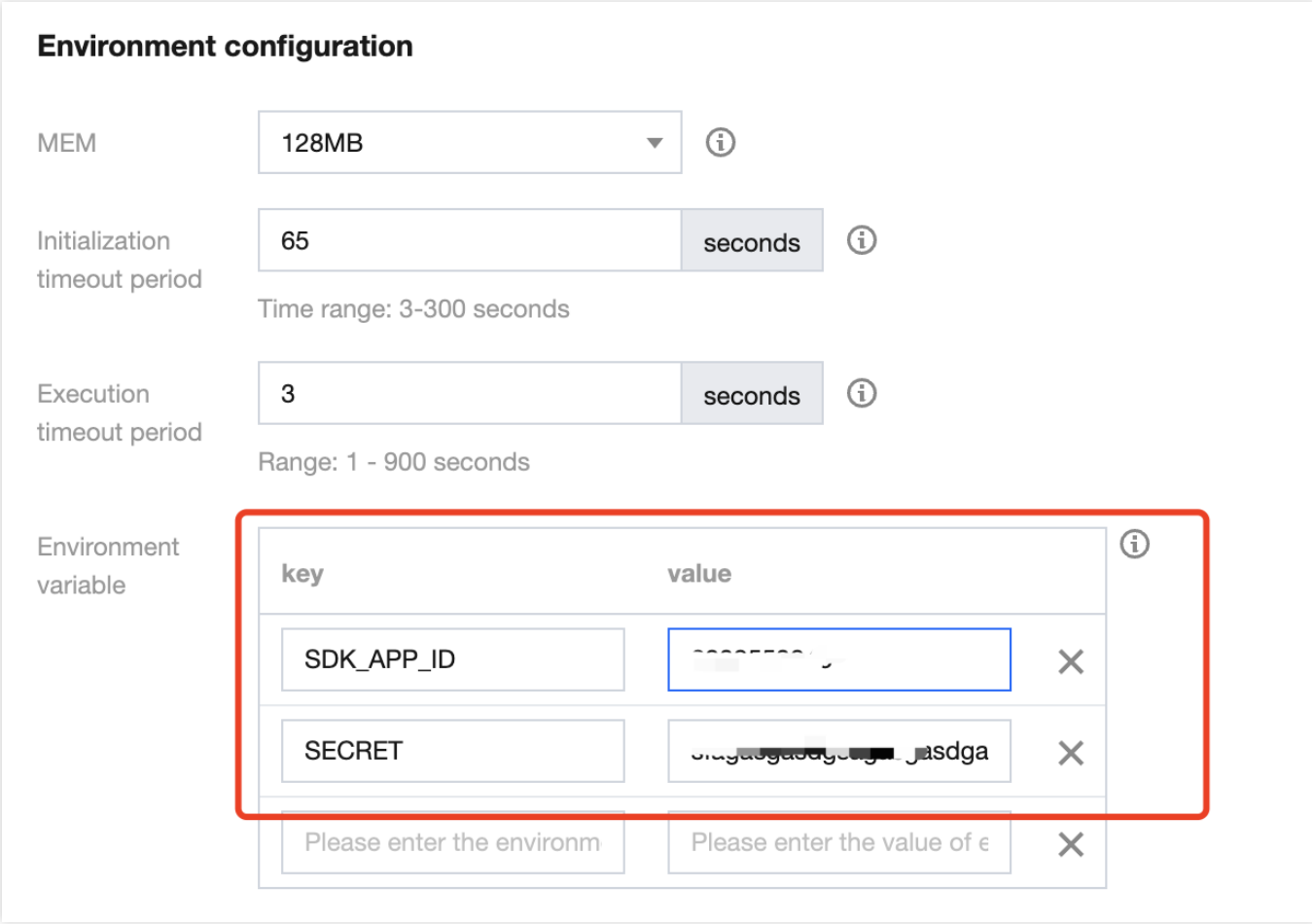


3. Specify the `Function name` based on your needs.

4. Click the `Advanced configuration` module and add the following `Environment configuration` .

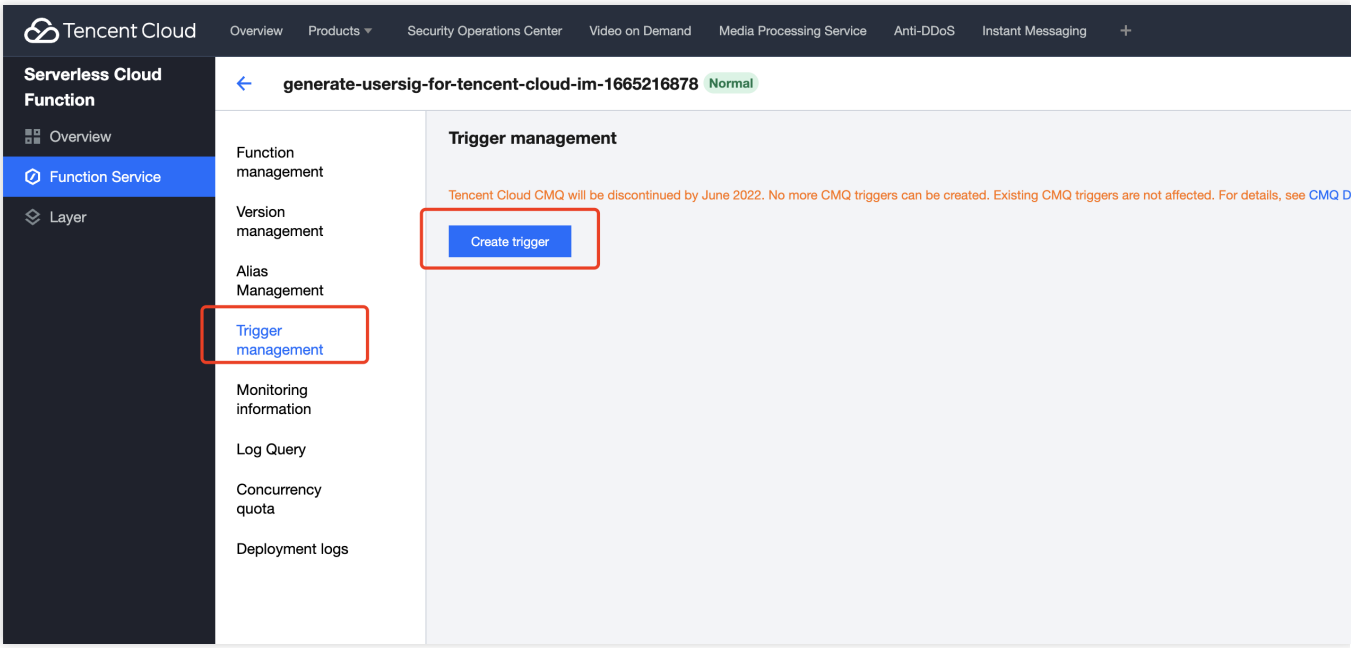
key	value

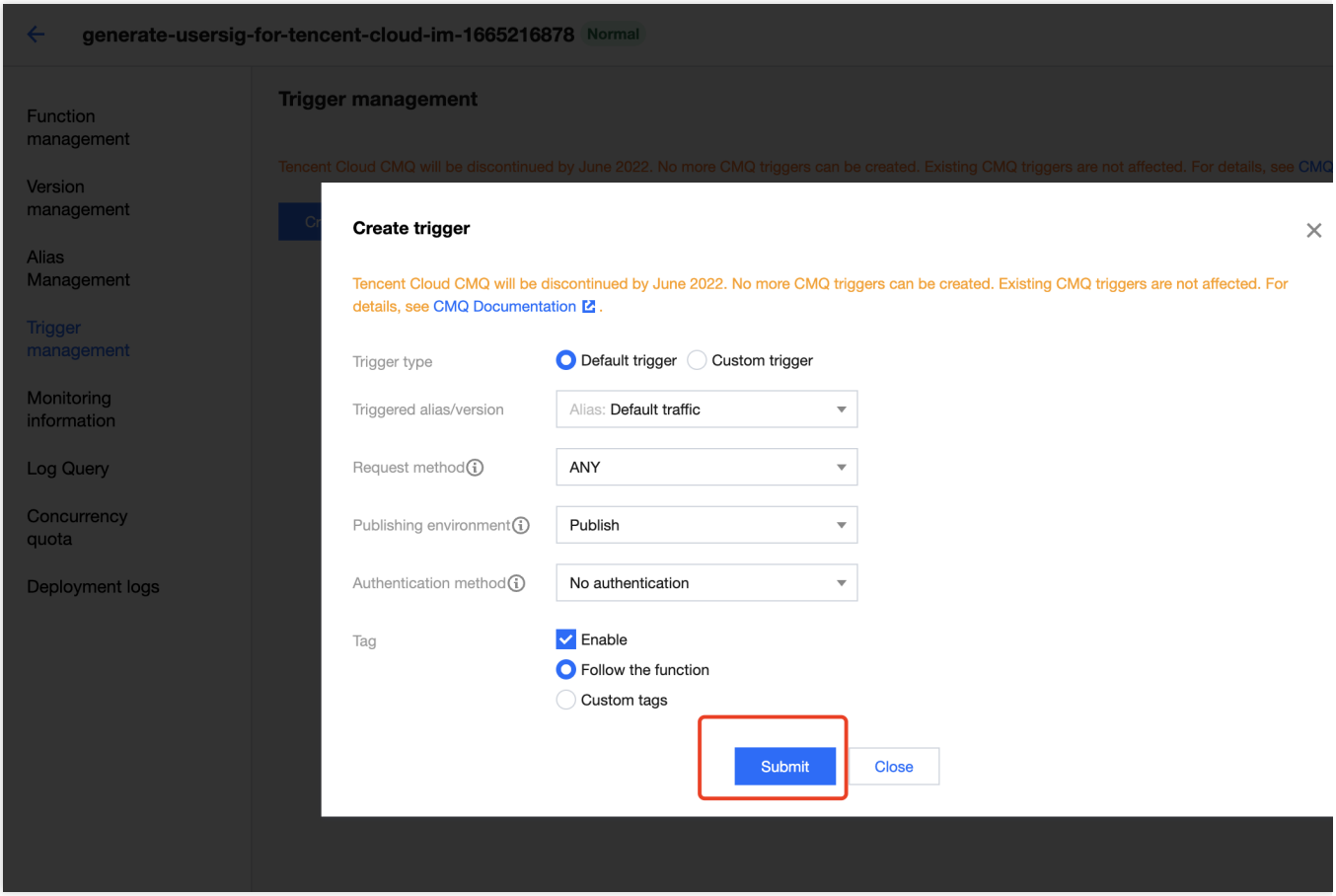
SDK_APP_ID	(Your SDKAppID)
SECRET	(The `Key` shows on the main page of the IM console)



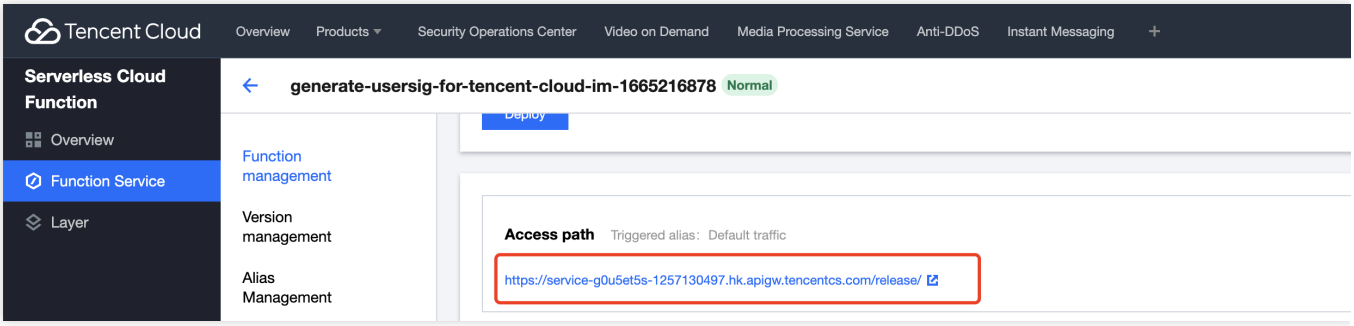
5. After deployment, create a trigger for it by clicking `Trigger management` on the left bar, then click `Create trigger`. Use the default settings on the modal opened, then click `Submit`.

> It might require granting permissions to `API Gateway` first, please follow the instructions to grant.

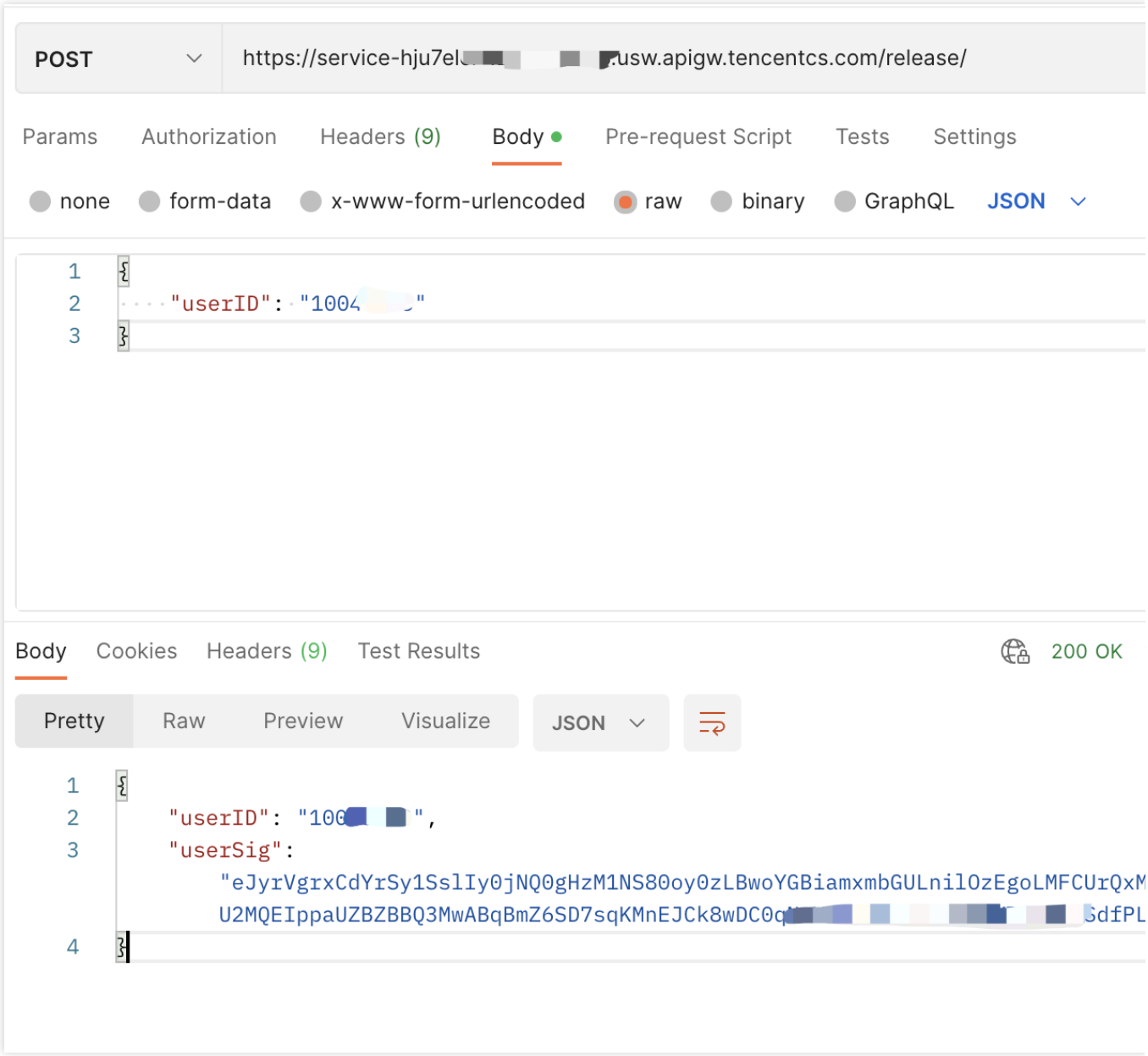




6. Find the URL of the API created from `Function management` => `Function codes` => `Access path` . This URL will be used in the following steps.

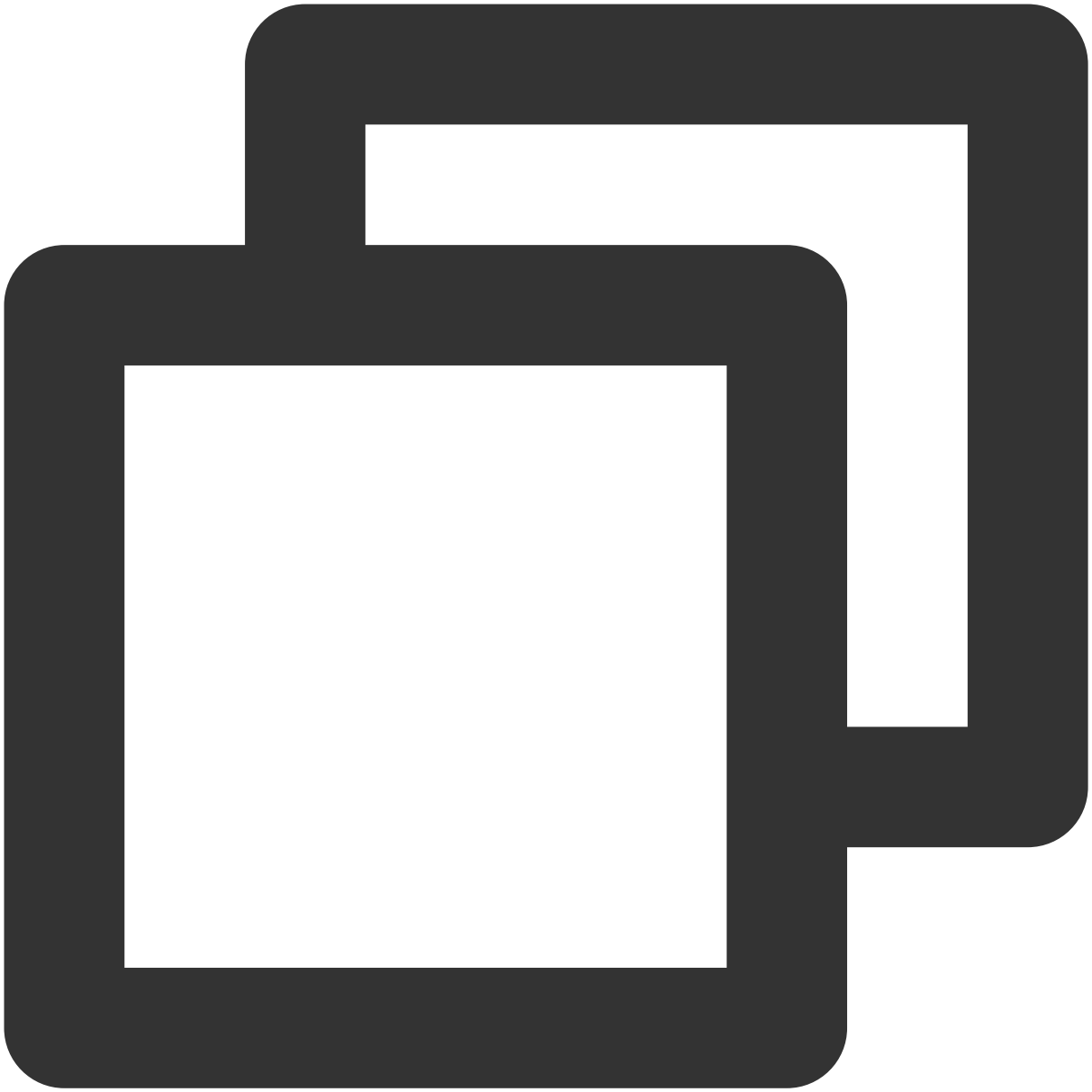


7. [Optional] You can test it by `Postman` .



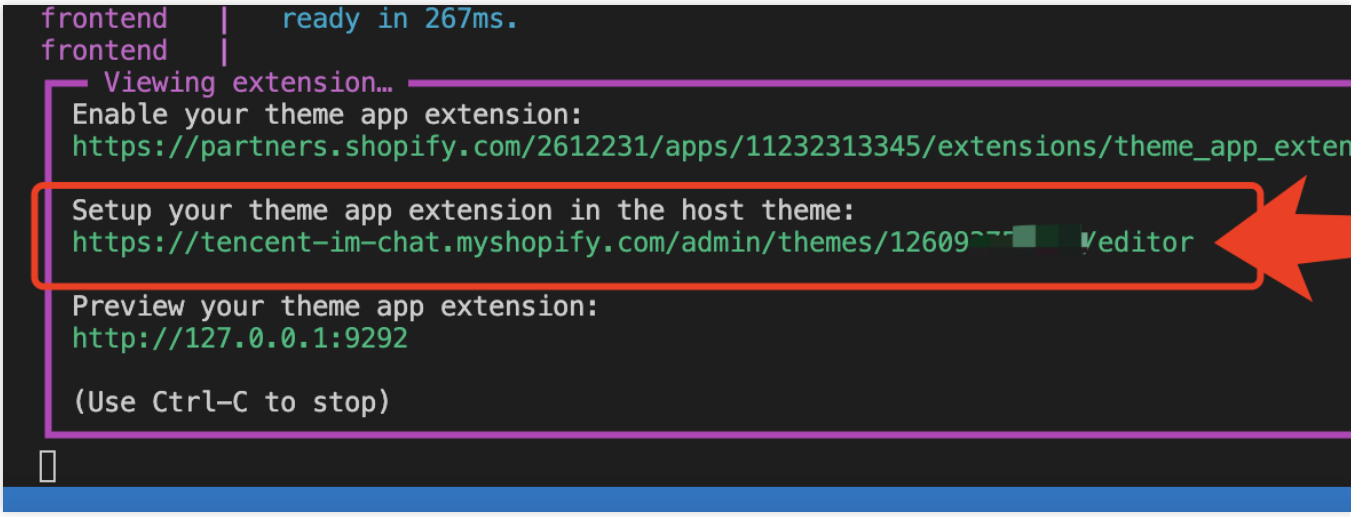
Step 8: Configure this extension

Run the following commands to start the extension:



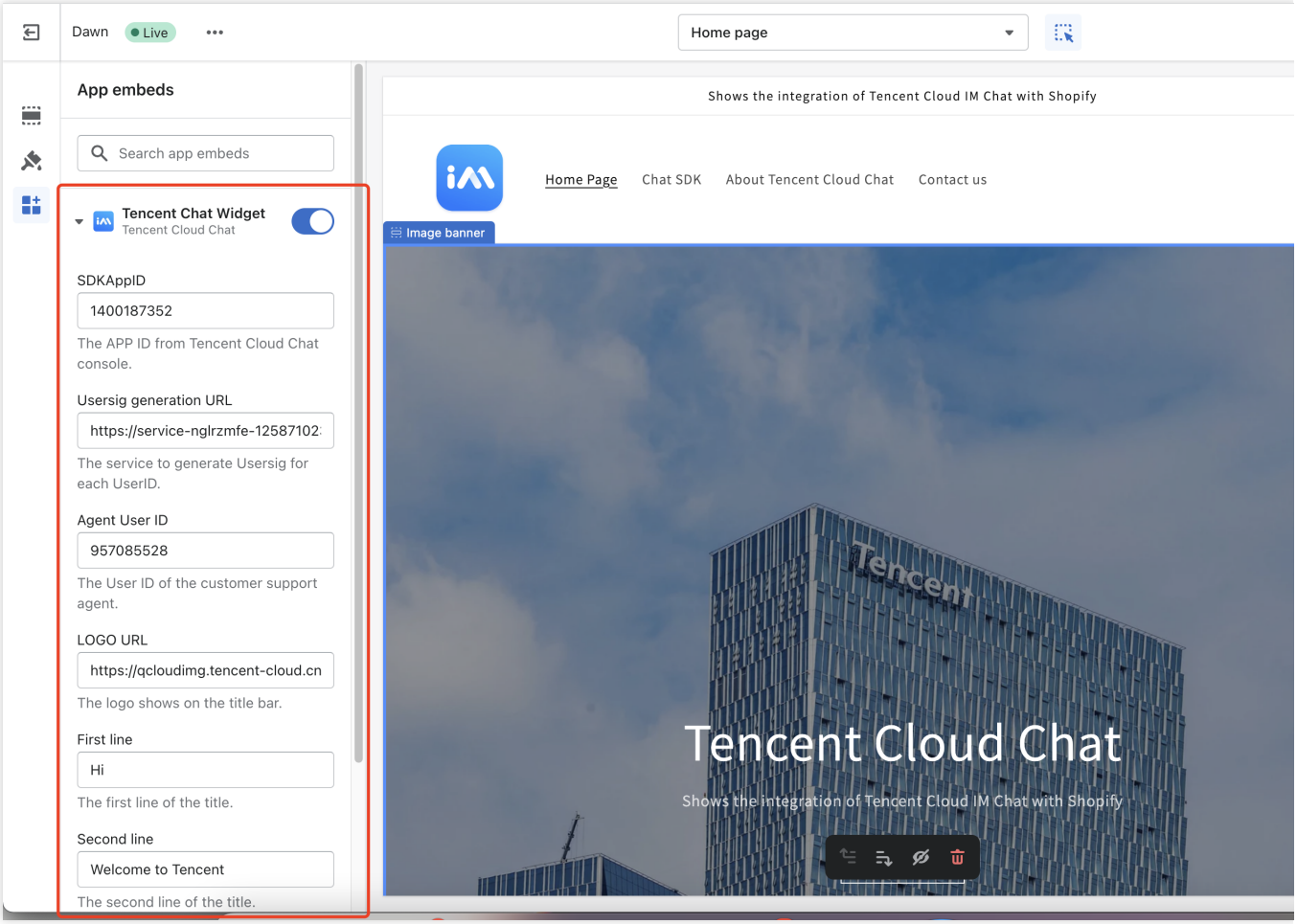
```
npm run dev
```

Click the second URL that's printed at the bottom of the CLI output to set up the extension.



Switch to 'App embeds' on the left menu, and enable 'Tencent Chat Widget'.

Fill in the settings related to it. The detailed introduction for each field shows below the image.

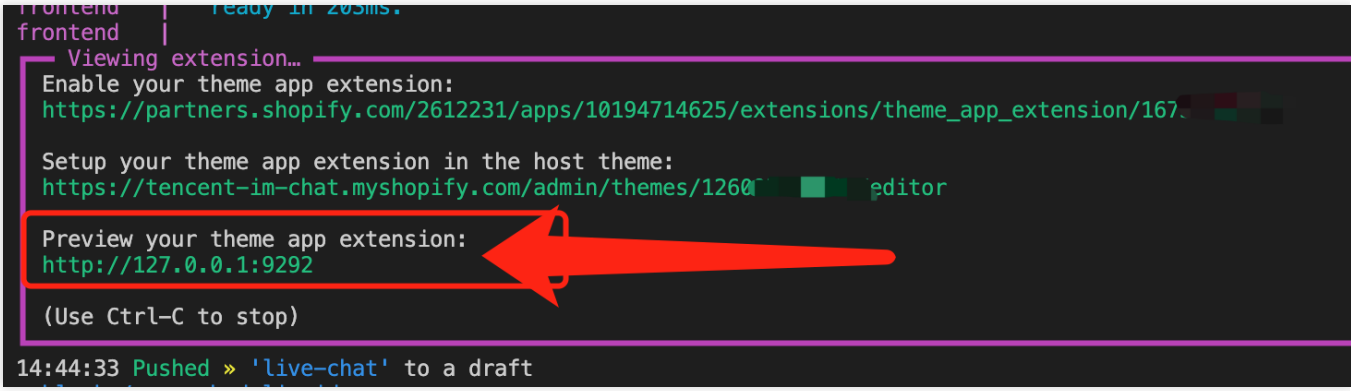


Field	Description	Required

SDKAppID	The `SDKAppID` from Tencent Cloud Chat console.	true
Usersig generation URL	The service to generate Usersig for each UserID. The URL of the API was created from step 7.	true
Agent User ID	The User ID of the customer support agent. The user who visitors chat with.	true
LOGO URL	The logo shows on the title bar.	false
First line	The first line of the title.	false
Second line	The second line of the title.	false

After the configuration, click the `Save` button on the top right.

Now, you can click the third URL that's printed at the bottom of the CLI output to preview the extension.

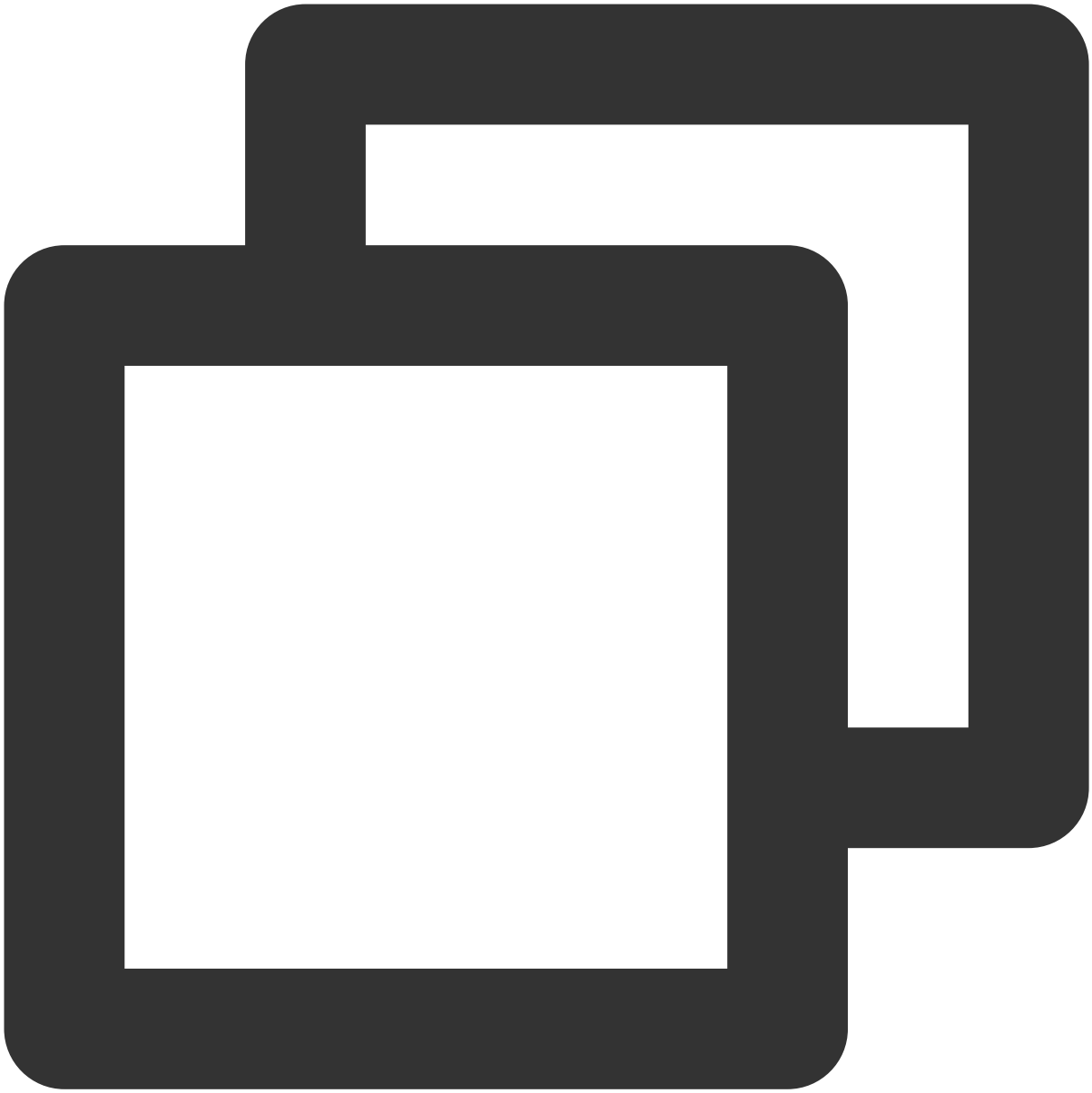


Make sure the chat widget works well. You can refer to our sample codes, if errors are encountered.

Step 9: Deploy the chat widget extension

After your chat widget extension is ready for online stores, you can deploy the latest code and make the app extension public to the development store.

1. Navigate to your app directory.
2. Run the following commands:



```
npm run deploy
```

3. After you deploy the extension to Shopify, navigate to your app in the Partner Dashboard, click `Extensions` , and click the draft version that you previously created.

4. Click `Create version` to create a version of your theme app extension.

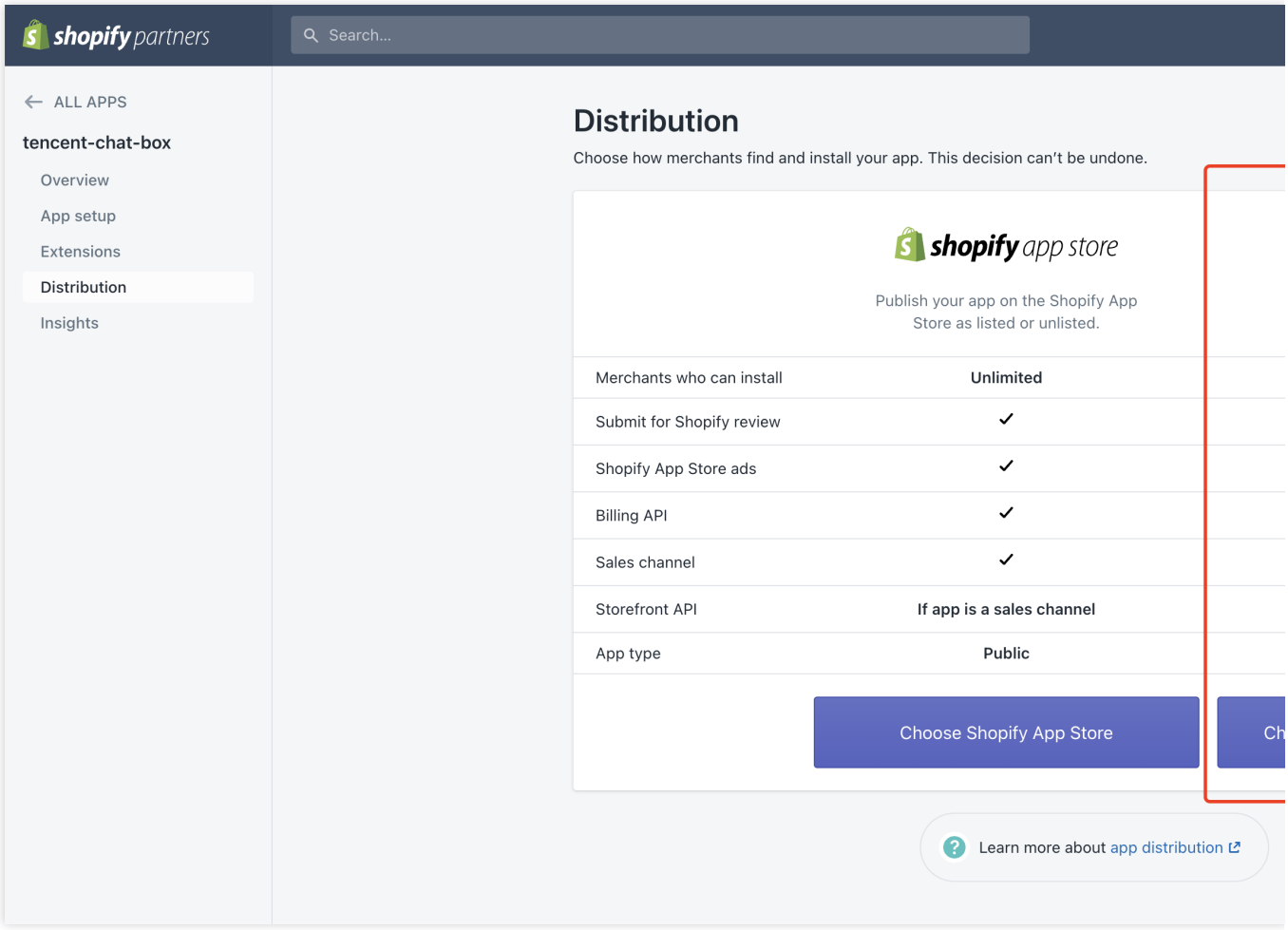
5. Click `Publish` beside the version that you want to publish, and click `Publish` in the popup modal to confirm.

Step 10: Install this app to your online store

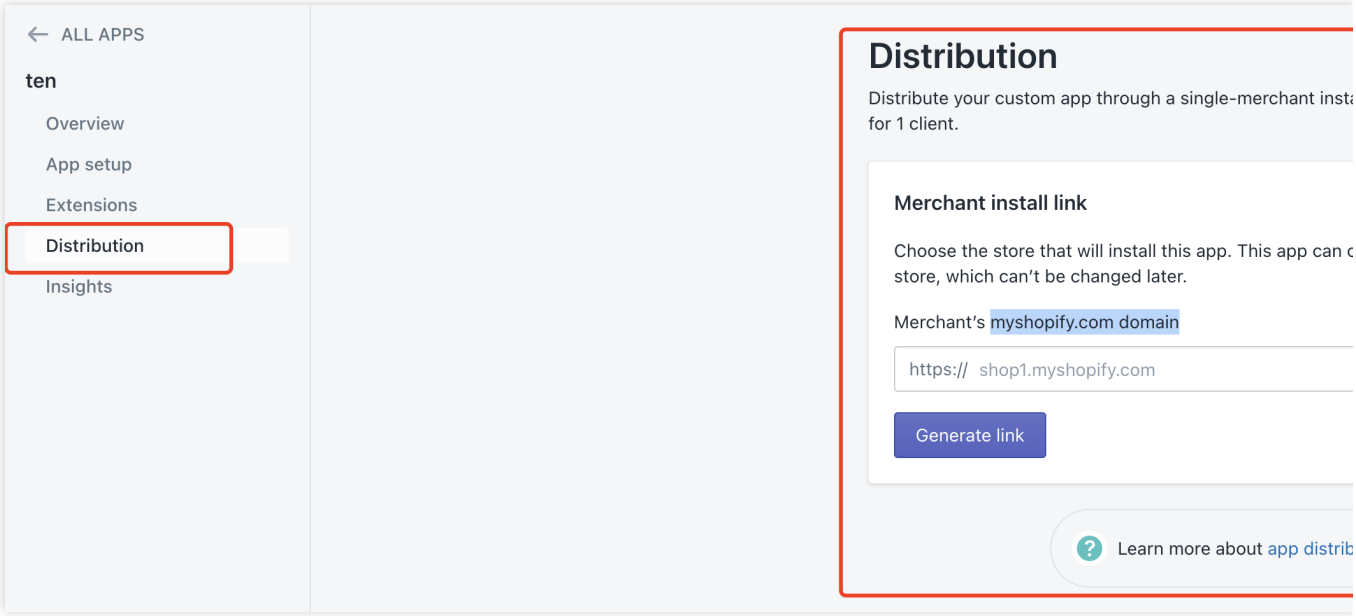
1. From the Shopify Partner Dashboard, go to Apps and then select your newly created app from the list.

2. In the sidebar, click **Distribution**.

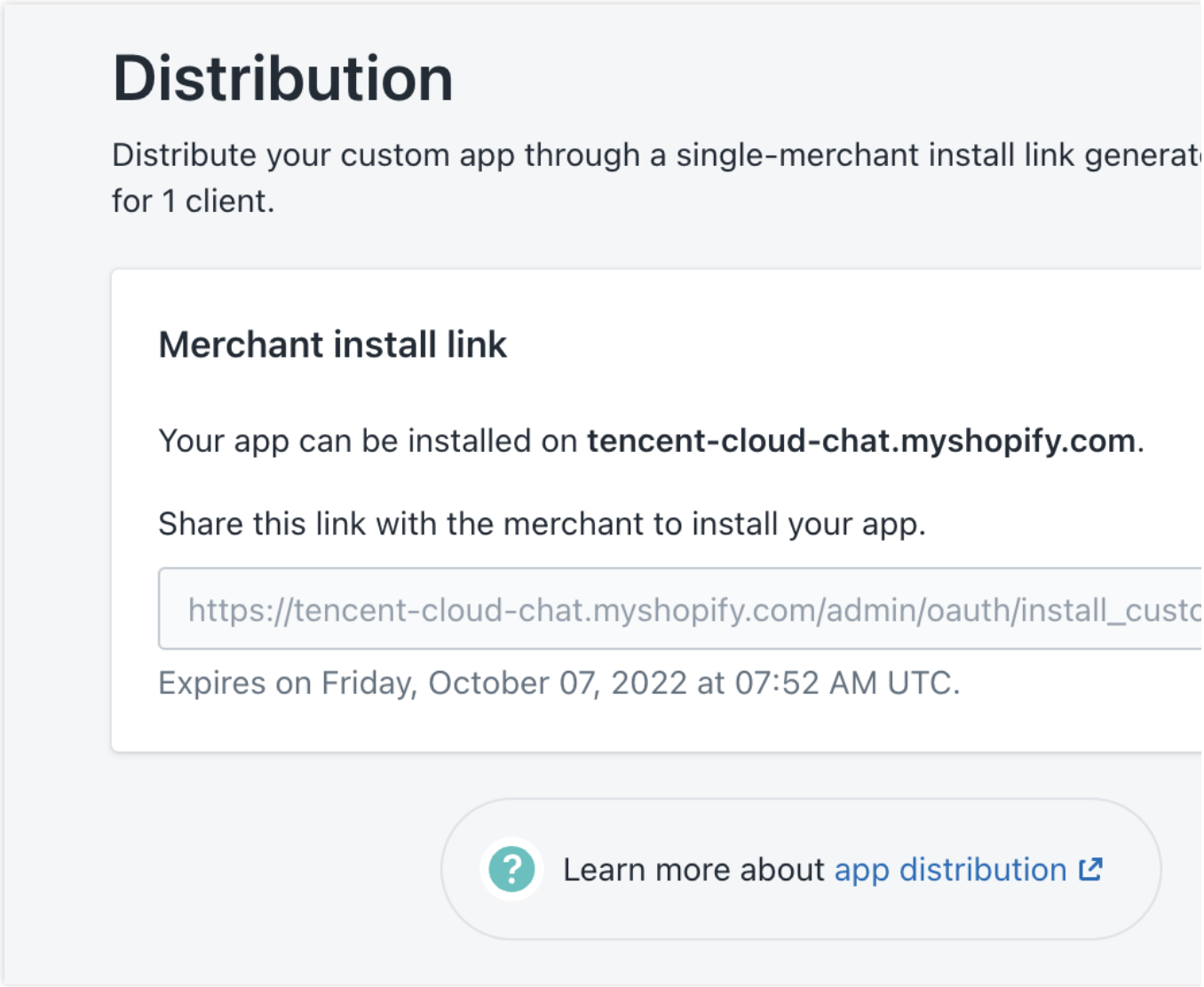
3. Select "Single-merchant install link" as the distribution method.



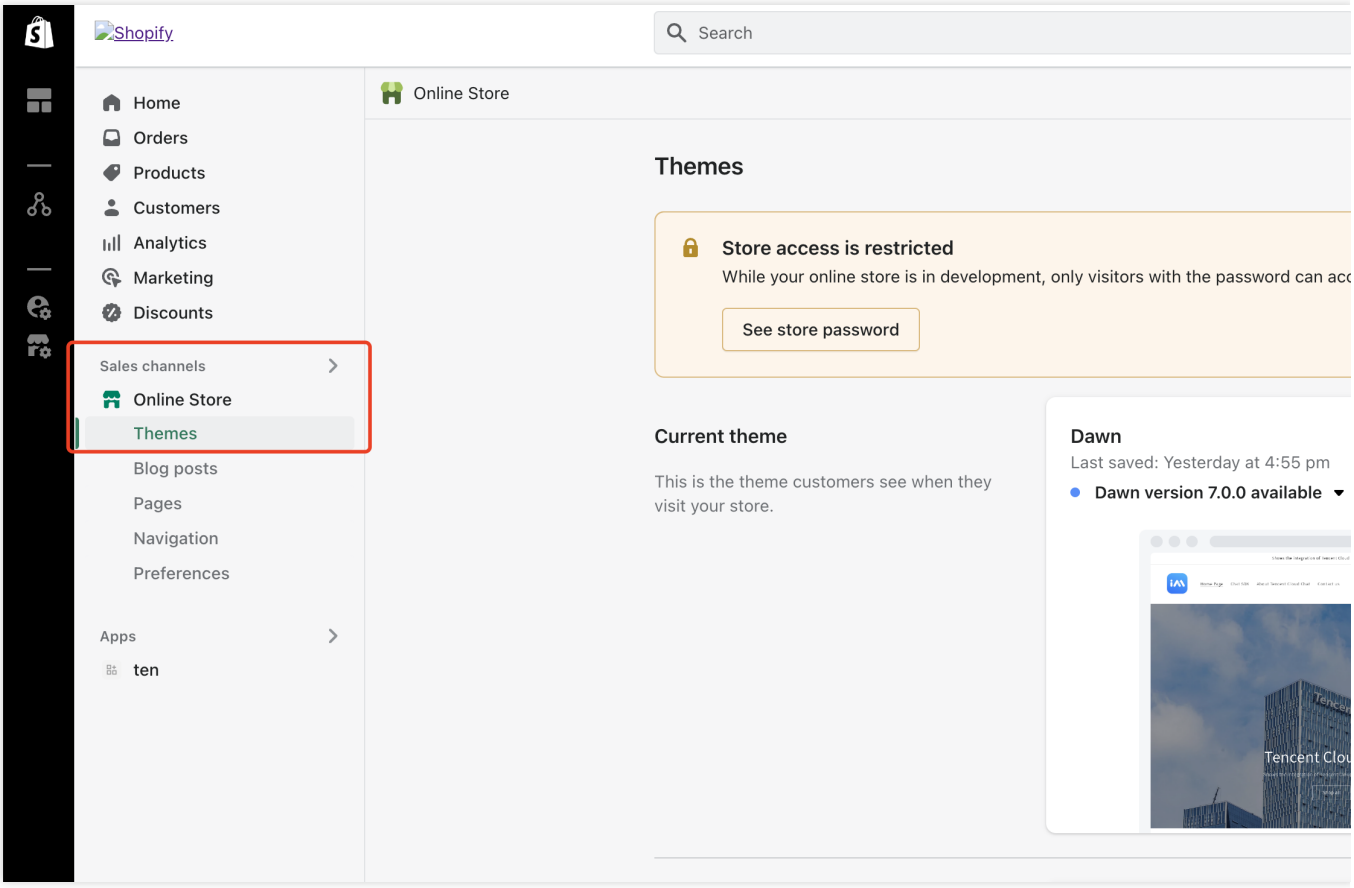
4. Generate the install link for your Shopify store, by entering your `myshopify.com` domain here.



5. Use the generated link to install the app in your current store.



6. Enable and configure the widget on the `Themes` section of your admin dashboard. The steps of enabling and configuring the widget can refer to step 8.



Conclusion

That's all you need to add a Tencent Cloud Chat widget to your Shopify store.

If there's anything unclear or you have more thinkings about the integration with Shopify, feel free to contact us!

Discord 实现指南

最近更新时间：2024-02-19 11:22:21

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。

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。
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再应用详情页查看，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。详情可参考无 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一字段，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，可现在控制台进行配置，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。

Discord 功能分析

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接口调用可参考文档 无 UI 集成方案 部分。

服务器

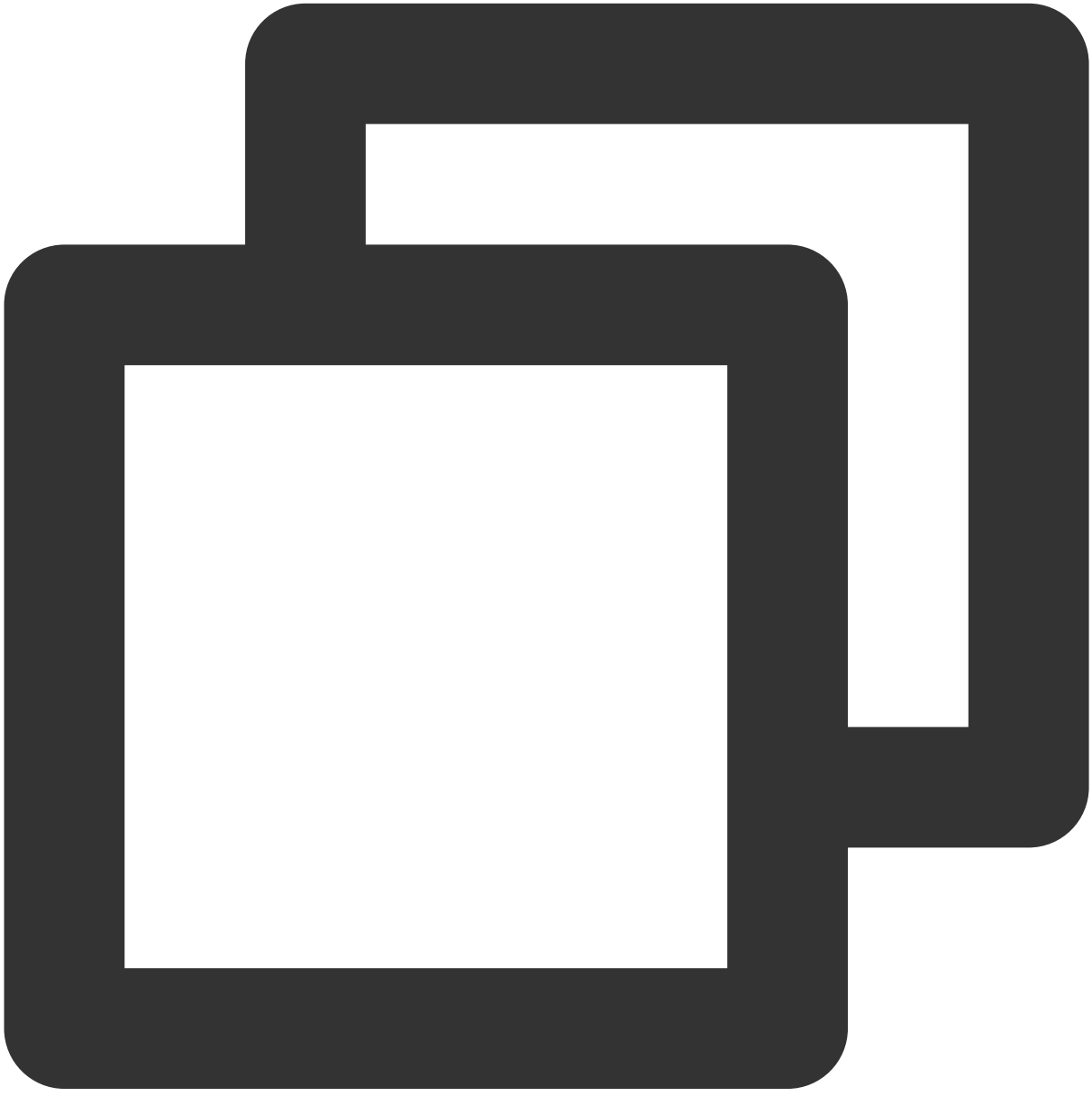
创建服务器

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：



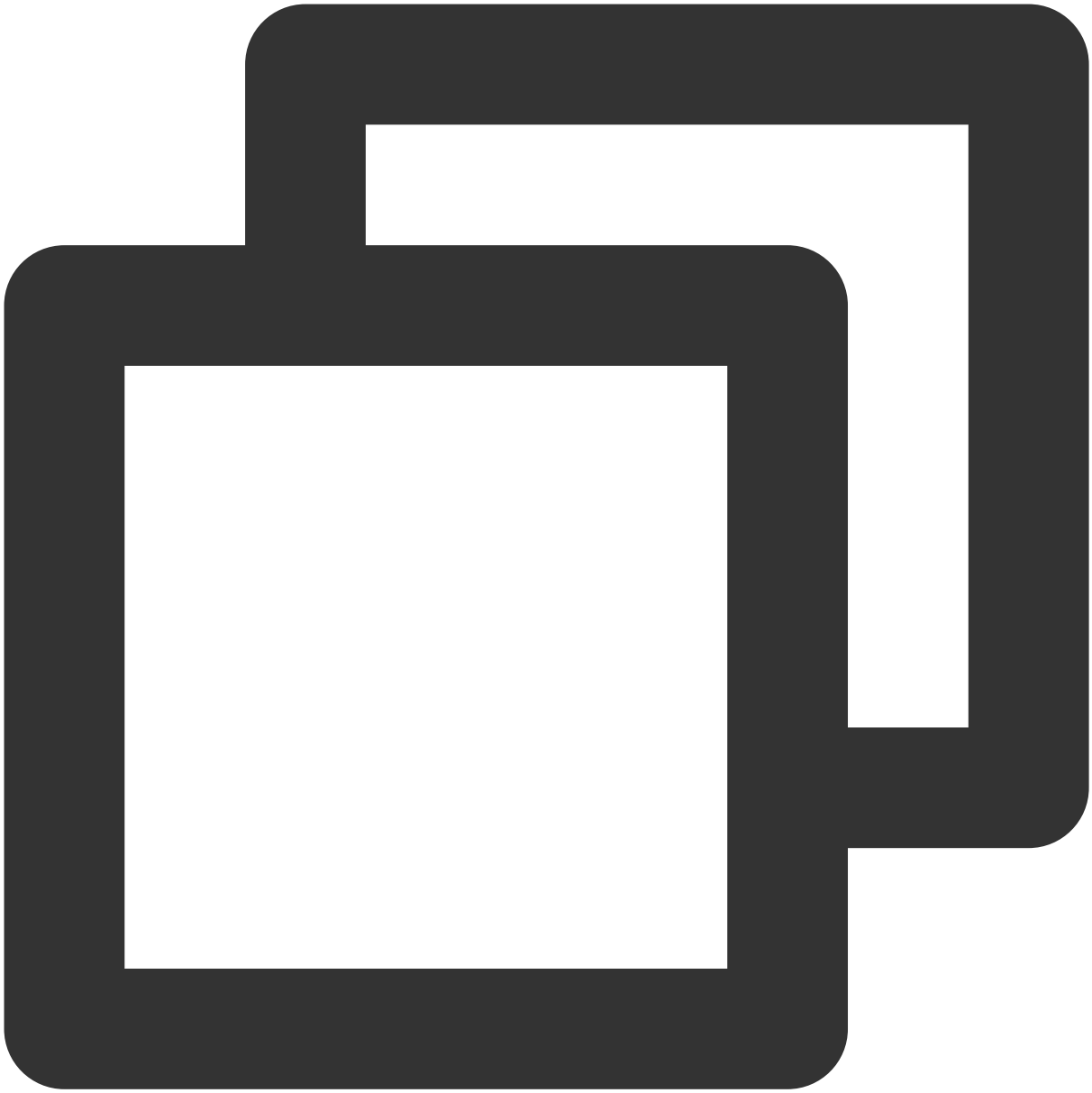
```
V2TIMGroupInfo  groupinfo = new V2TIMGroupInfo();
groupinfo.setGroupName("测试服务器");
groupinfo.setSupportTopic(true);
```

```
// 初始群成员
List<V2TIMCreateGroupMemberInfo> memberList = new LinkedList<V2TIMCreateGroupMember
// 其他配置，如服务器头像等
V2TIMManager.getGroupManager().createGroup(groupinfo, memberList, new V2TIMValueCal
            @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
            }
});
```

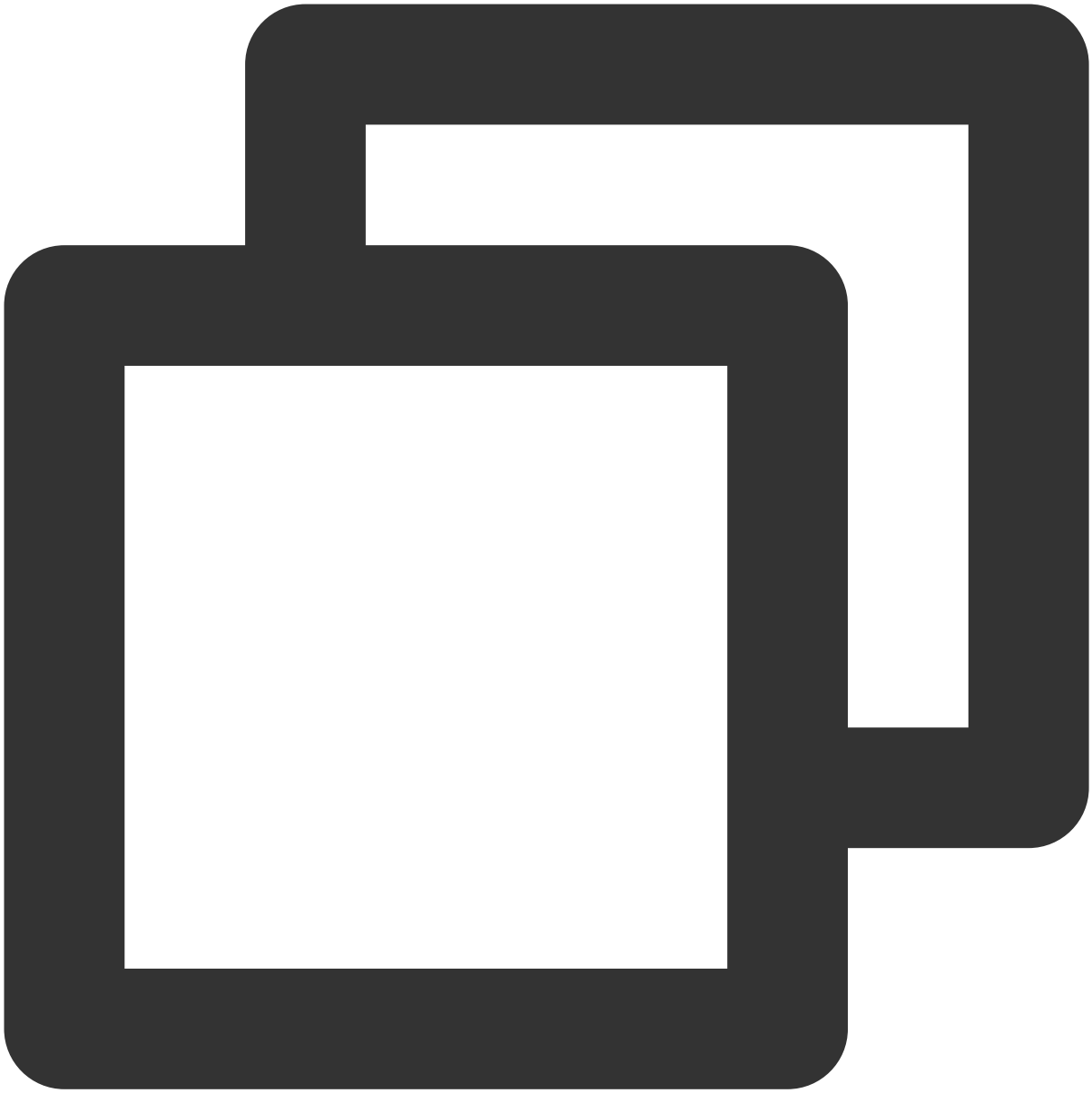
接口调用成功后，会在 onSuccess 回调中返回服务器 ID，在后续的在服务器中创建频道的功能中会用到。

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

```java
V2TIMManager.getGroupManager().getJoinedCommunityList(new V2TIMValueCallback<List<V
            @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，会个服务器创建默认的分类，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：

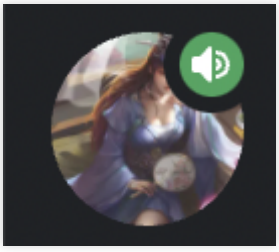
1. 在控制台开启自定义字段 key。
2. 使用客户端 SDK\\服务端 API 进行读写。

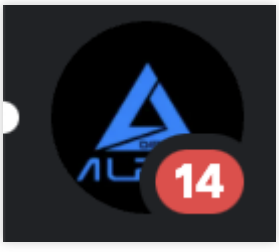
设置群自定义字段的 服务端 API 以及 客户端 SDK。

注意：

社群为旗舰版能力，设置社群的自定义字段需要先购买旗舰版。

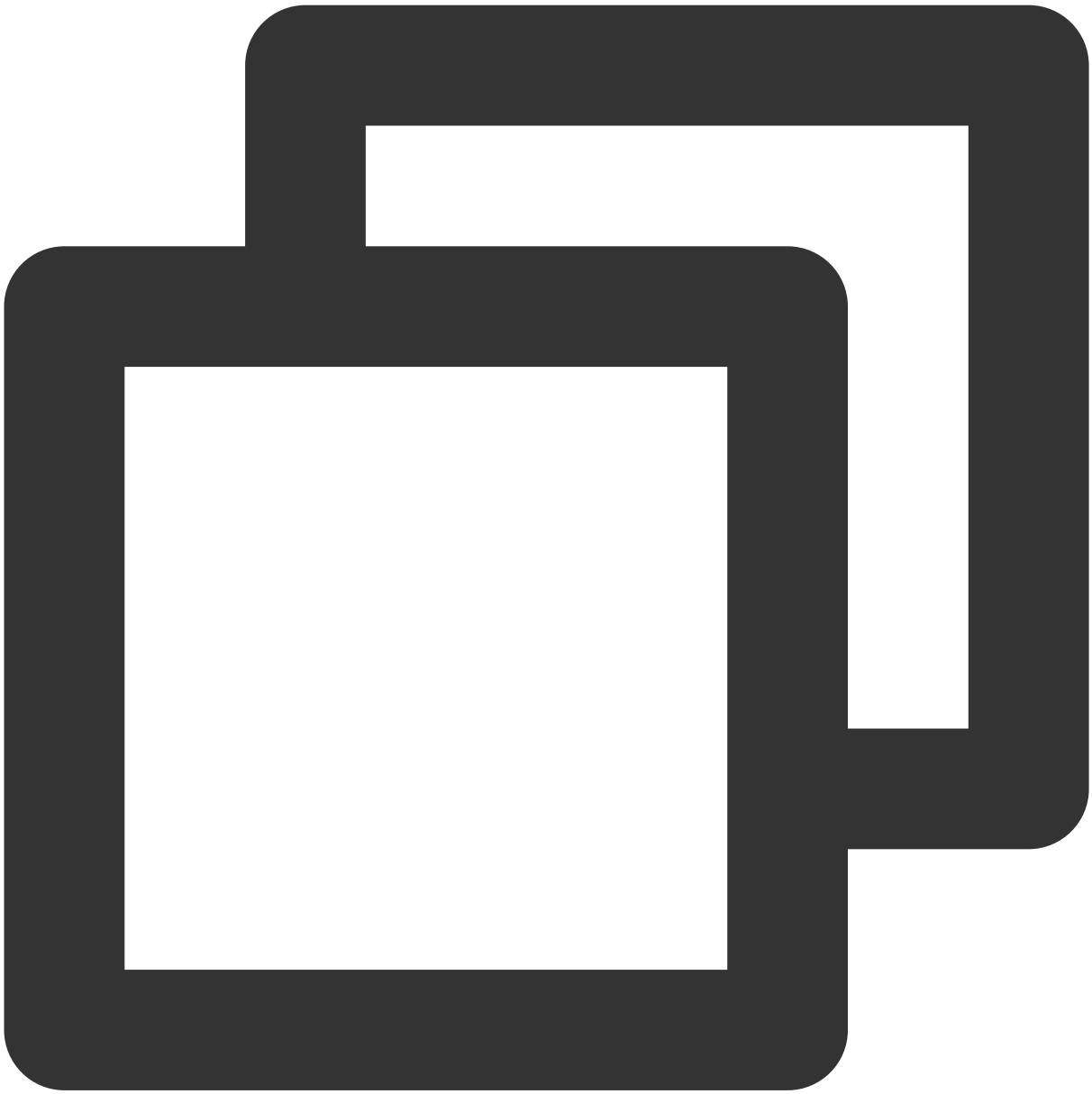
服务器消息未读数&自定义状态展示





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

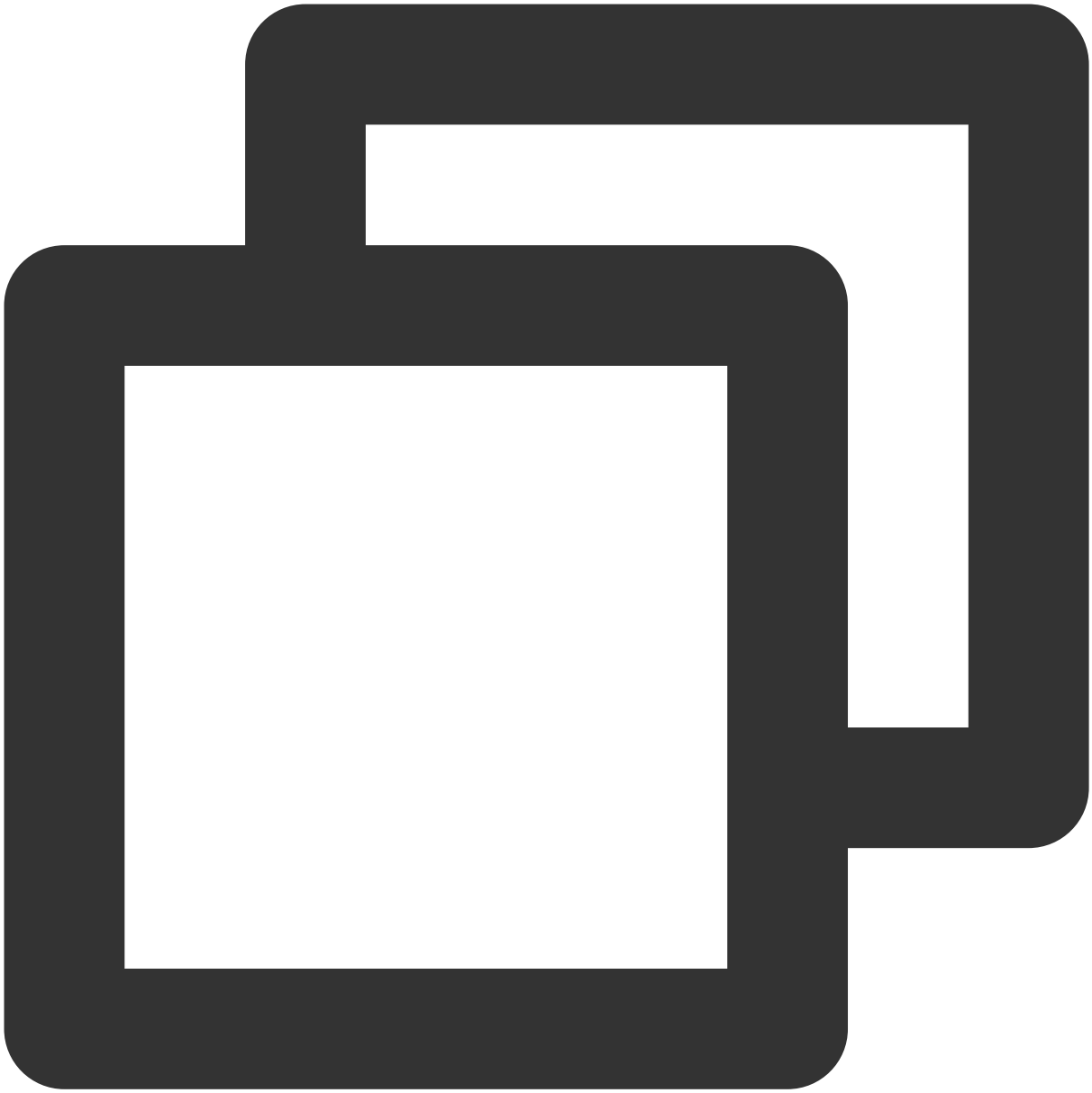
getUnreadCount 获取公有频道的未读数，由于私有频道由 work 群实现所以可通过 getConversation 来获取私有频道的未读数。



```
// 公有频道
List<String> conversationIDList = new LinkedList();
conversationIDList.add("GROUP_$GROUPID");
V2TIMManager.getConversationManager().getConversationList(conversationIDList, new V
            @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
            }
});
// 私有频道
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList, new V2TIMValu
                @Override
                public void onSuccess(List<V2TIMTopicInfoResult> v2TIMTopicInfoResu

                }

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

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。

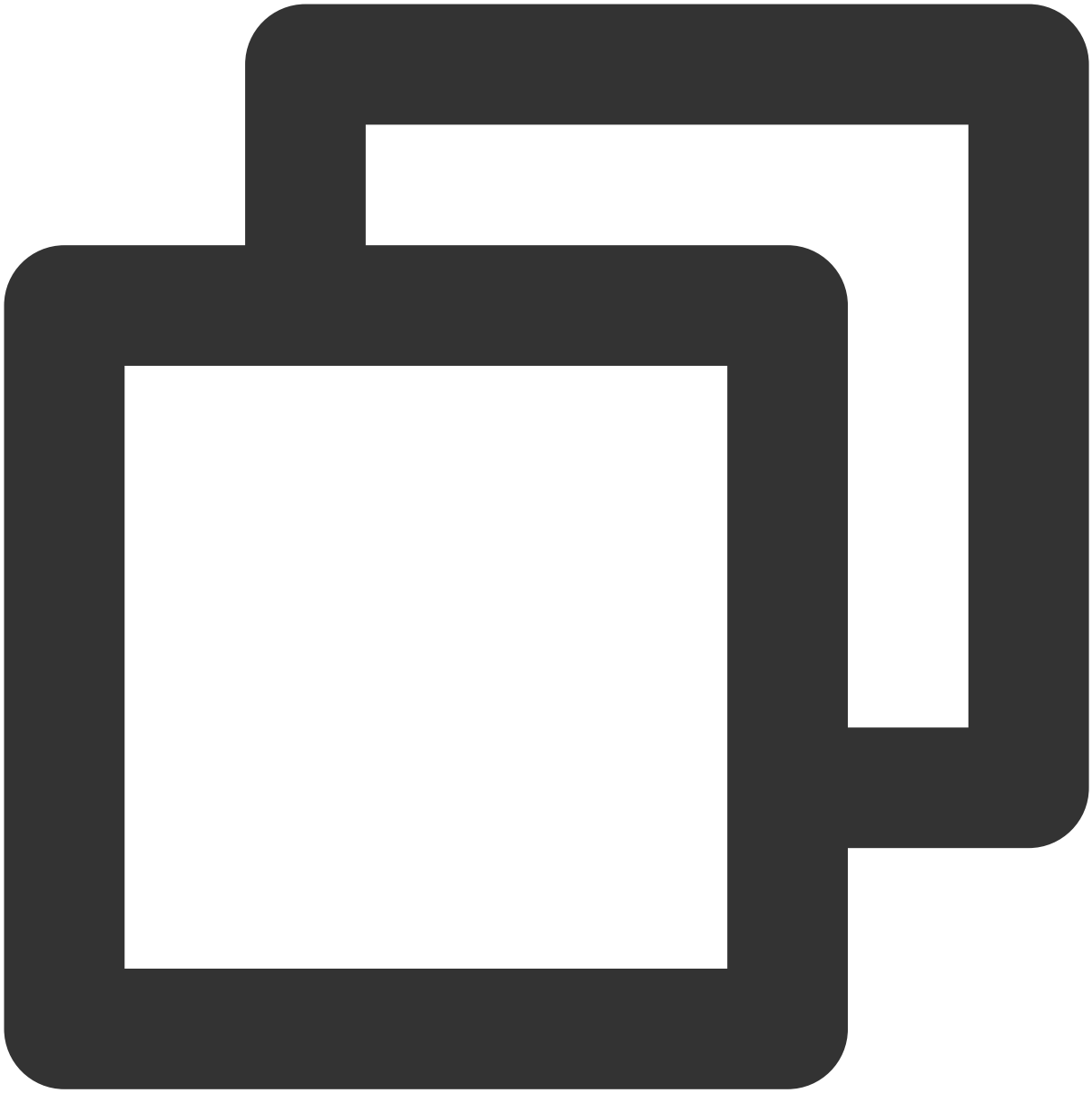


```
List<String> conversationIDList = new LinkedList();
String customData = "通话中"
V2TIMManager.getConversationManager().setConversationCustomData(conversationIDList,
            @Override
            public void onSuccess(List<V2TIMConversationOperationResult> v2TIMConve
                // 设置群会话自定义数据成功
            }

            @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> conversationL
            }
            @Override
            public void onTotalUnreadMessageCountChanged(long totalUnreadCount) {
            }

            @Override
            public void onConversationGroupCreated(String groupName, List<V2TIMConv
            }

            @Override
            public void onConversationGroupDeleted(String groupName) {
            }

            @Override
            public void onConversationGroupNameChanged(String oldName, String newNa
            }

            @Override
            public void onConversationsAddedToGroup(String groupName, List<V2TIMCon
            }

            @Override
            public void onConversationsDeletedFromGroup(String groupName, List<V2TI
            }
    }
    V2TIMManager.getConversationManager().addConversationListener(conversationLister);
```

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

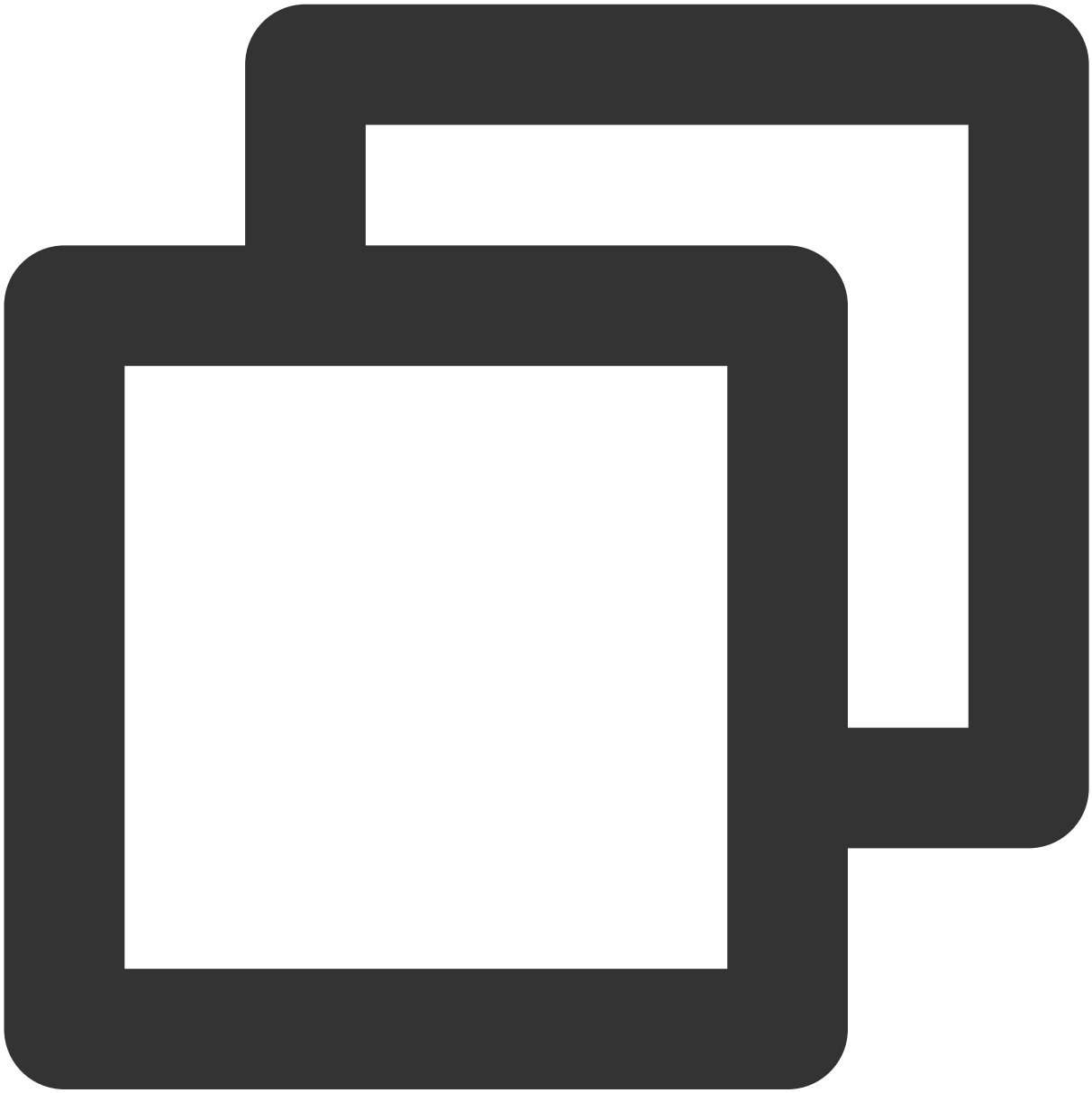
在服务器内，可创建多个频道。在该服务器下创建了四个频道，并且将四个频道放到了两个分类中。
对于频道，用户可以邀请用户加入以及对频道进行基本的设置。用户大多数的聊天是在频道内进行的，因此频道的能力在 Discord 中是最重要的。对应于腾讯云 IM，即是话题的能力。在 IM 社群支持在在社群内创建话题的能力。

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：

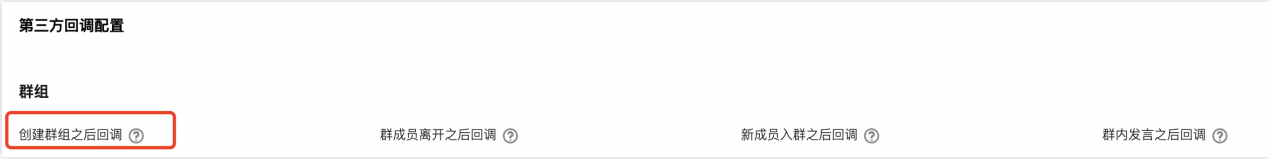
1. 使用 创建群后回调 通知业务服务端创服务器成功。
2. 业务侧判断是是否创建的社群，以免影响其他群相关的业务。
3. 根据服务器属性，在服务端创建话题。

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（
    "TopicName": "TestTopic",          // 话题的名称（必填）
    "From_Account": "1400187352", // 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）
}
```

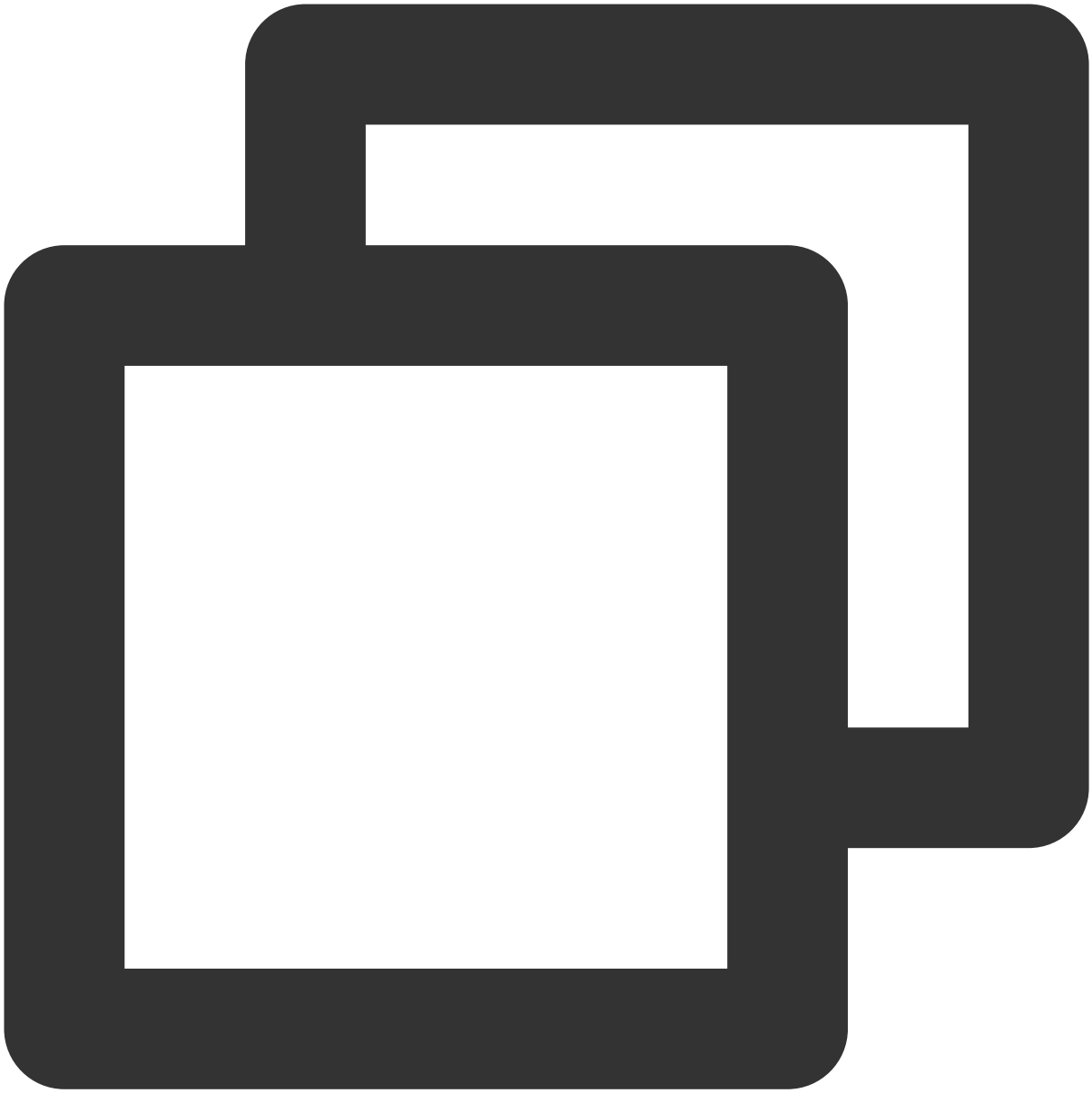


创建频道

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 V2TIMValue
            @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：

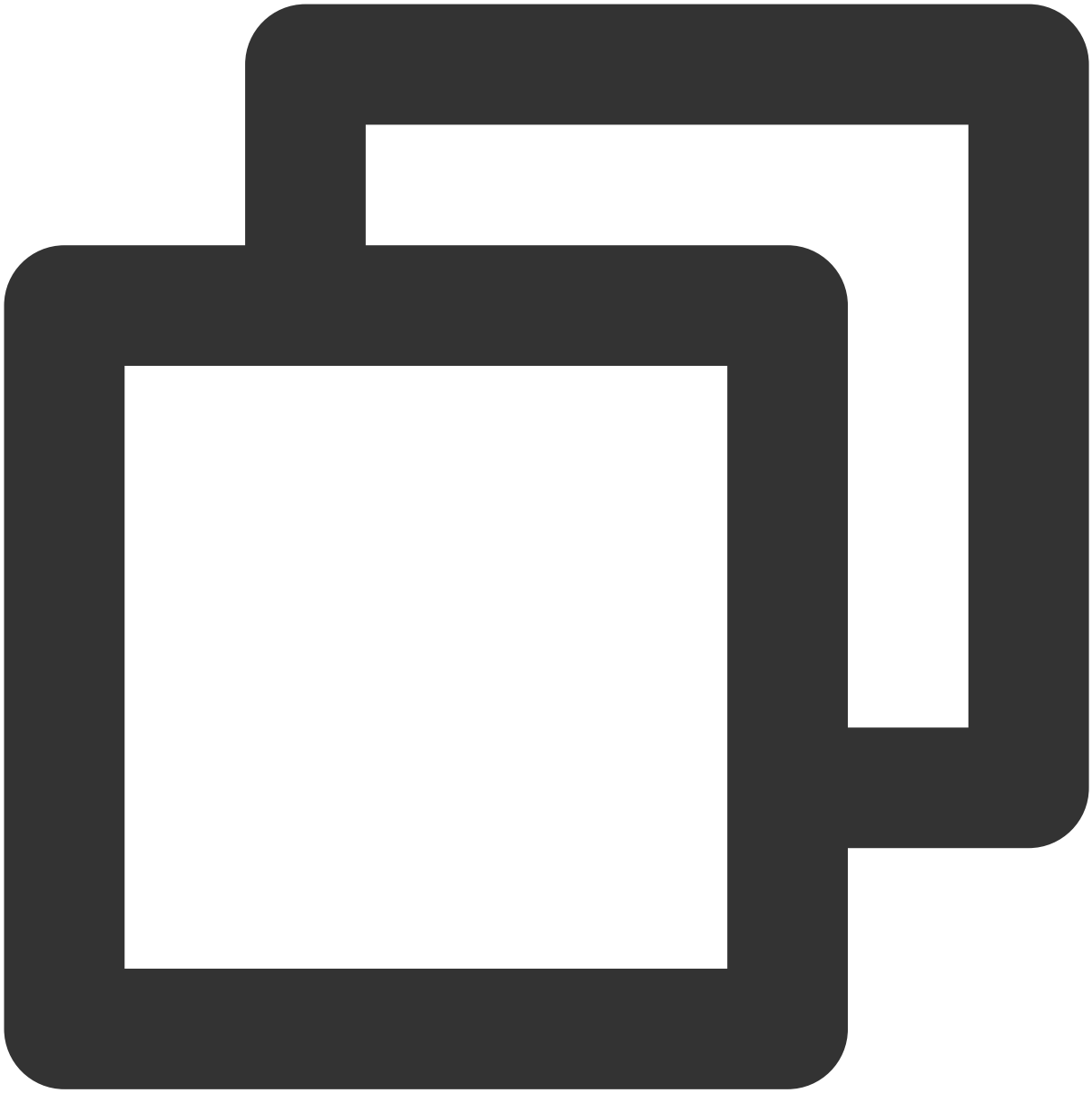
1. 邀请用户进入服务器，进入服务器的用户会进入所有的公开的频道。
2. 用户通过搜索进入服务器。
3. 私有频道通过服务器管理员邀请进入。

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。

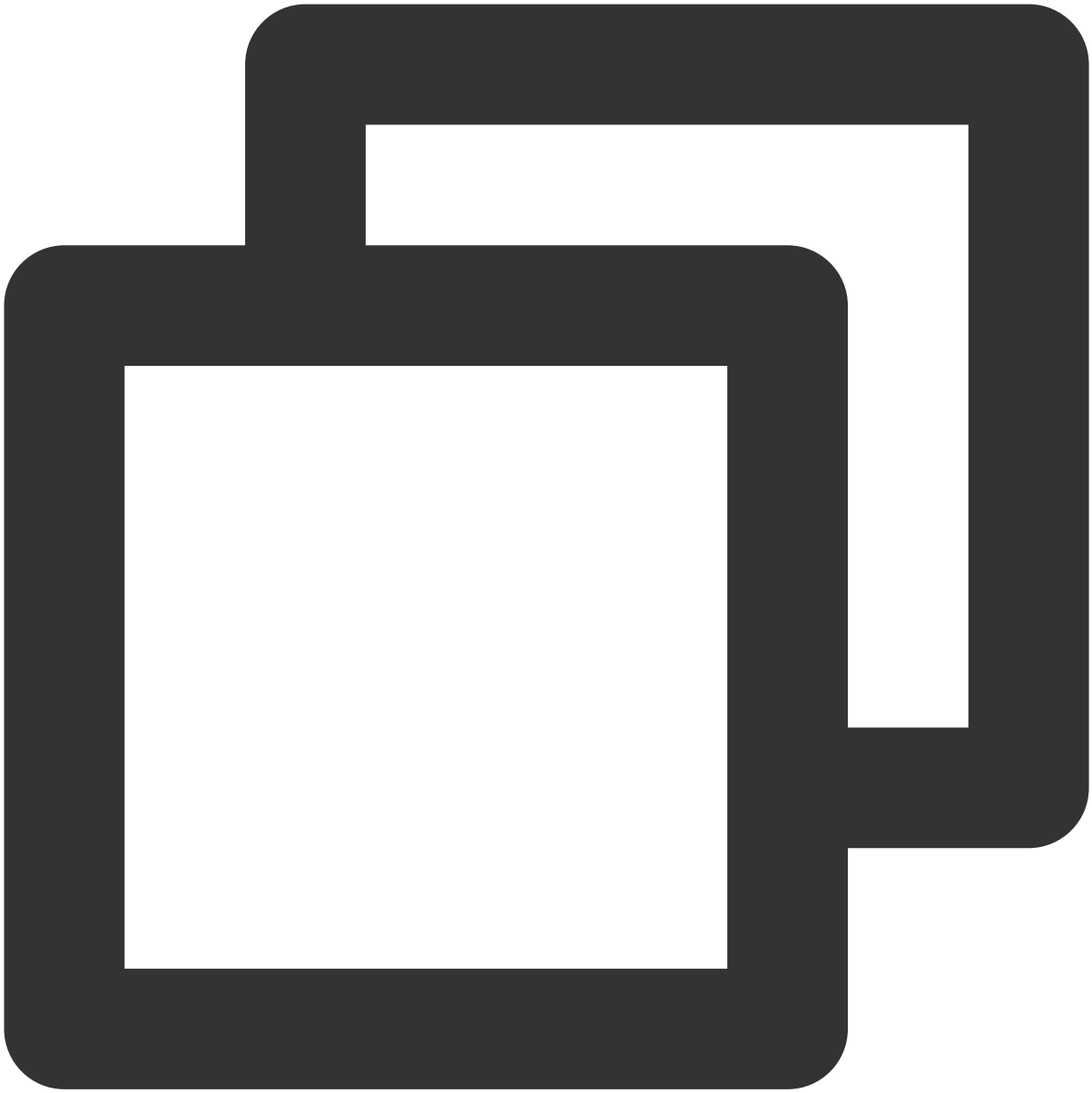
频道设置

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
    }
```

```
});
```

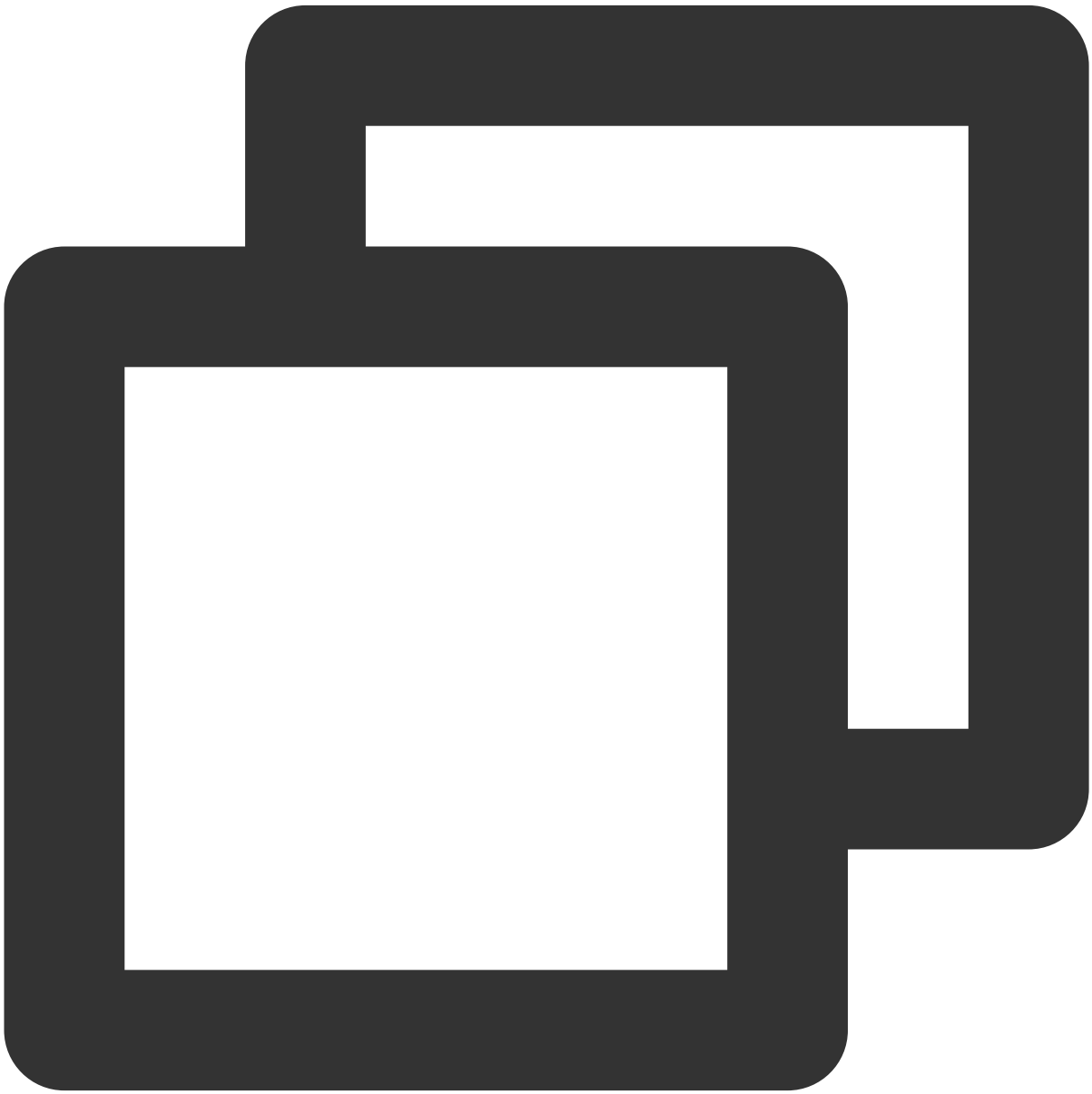


```
// 设置频道如何接受消息
String groupID = "topicid"
int opt = 0;
V2TIMManager.getMessageManager().setGroupReceiveMessageOpt(groupID, opt, new V2TIMC
    @Override
    public void onSuccess() {

    }
```

```
    @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。

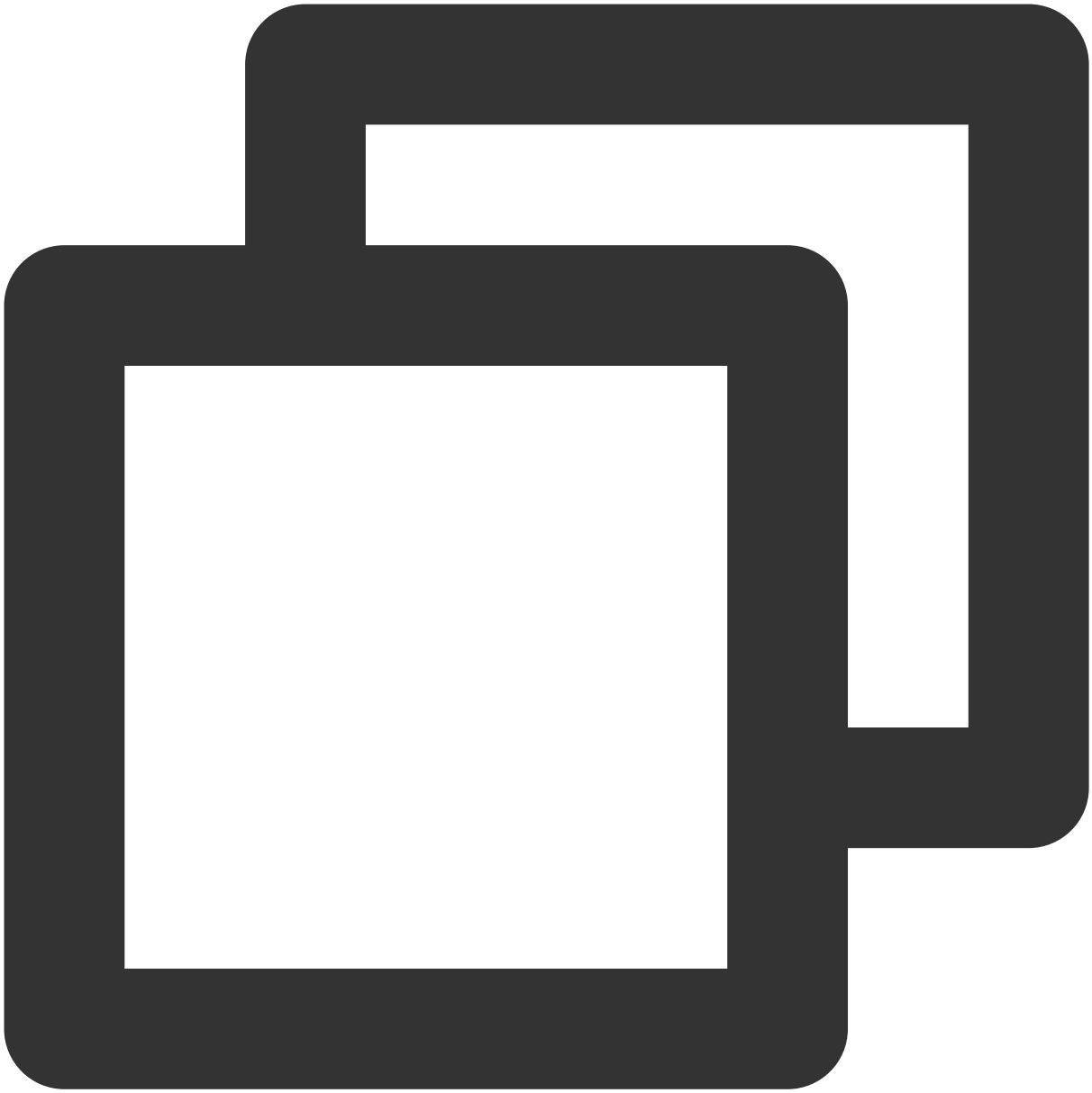


```
final V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
```

```
option.setGroupID("频道ID");
option.setCount(20);
// 其他配置
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @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。

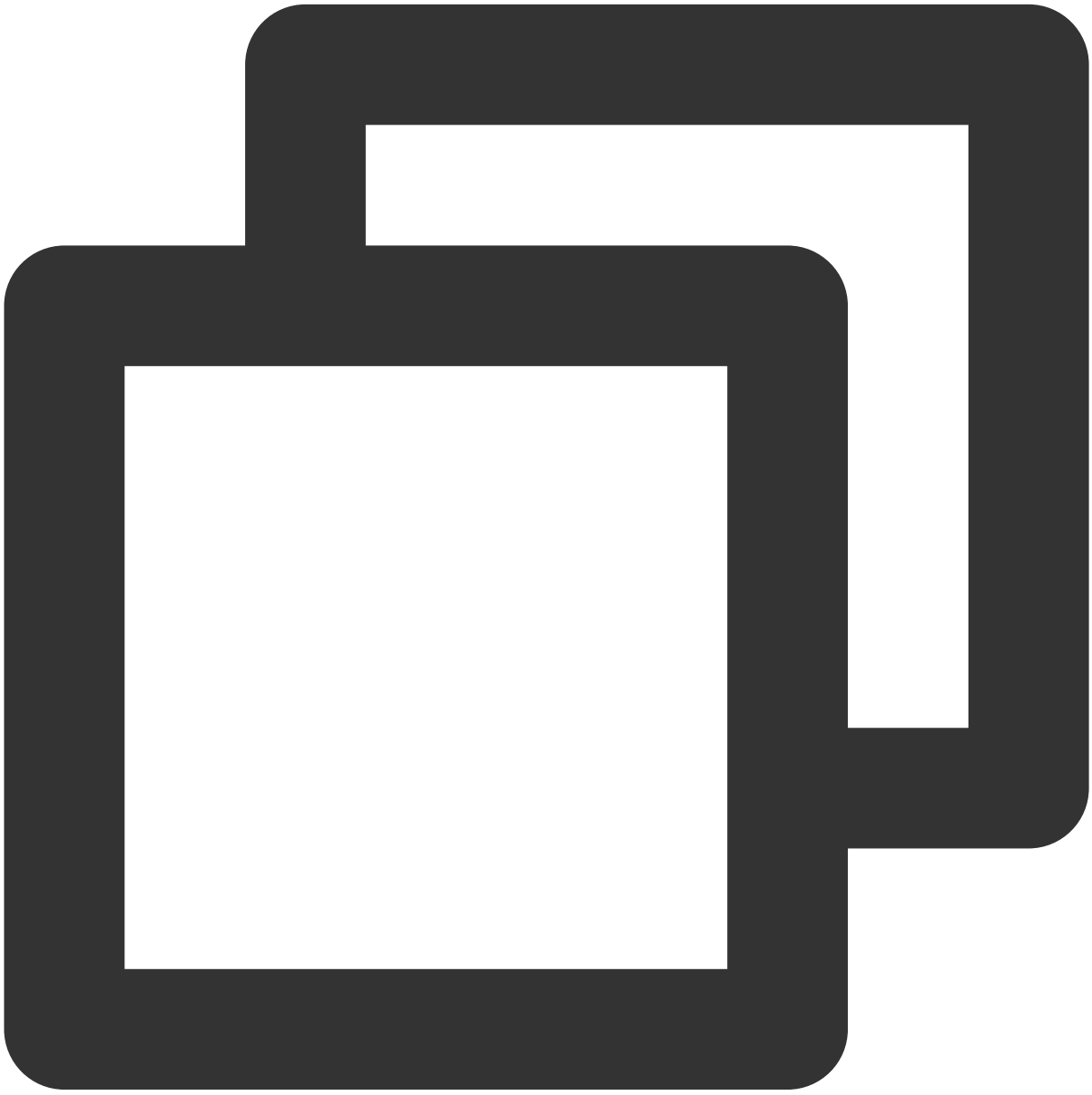


```
String groupID = "服务器ID";
List< String > topicIDList= new LinkedList(); // 频道消息列表
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList, new V2TIMValu
    @Override
    public void onSuccess(List<V2TIMTopicInfoResult> v2TIMTopicInfoResults) {
        // 获取频道信息，如频道id、频道名字、未读数等
    }

    @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 V2
```

```
    @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

子区的创建可以邦迪频道内的一条消息，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。这两个步骤最好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：

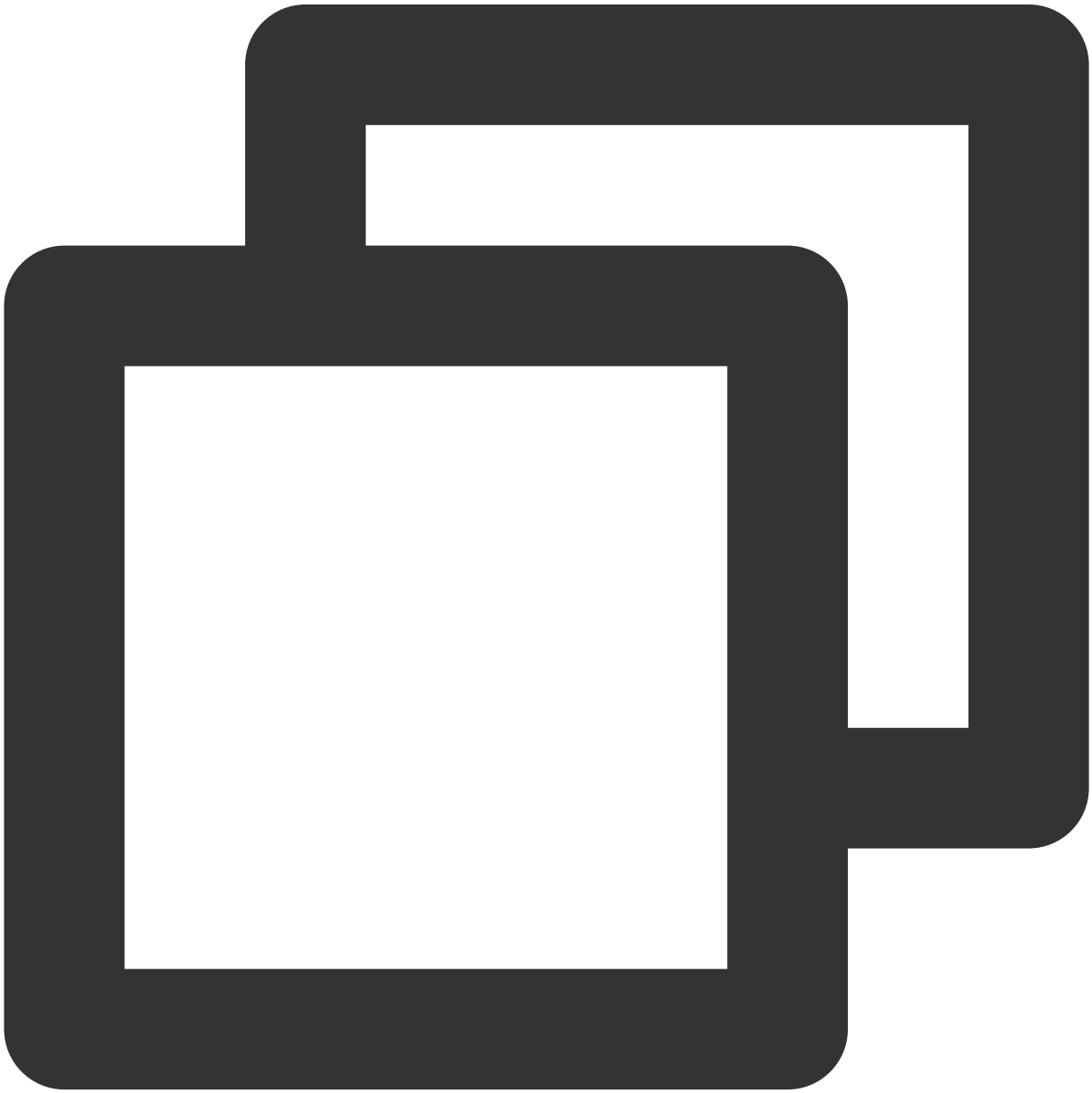
1. 该子区有多少条消息。
2. 该子区中 lastMessage 的相关信息。
3. 信息需要实时显示在消息列表中。

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。

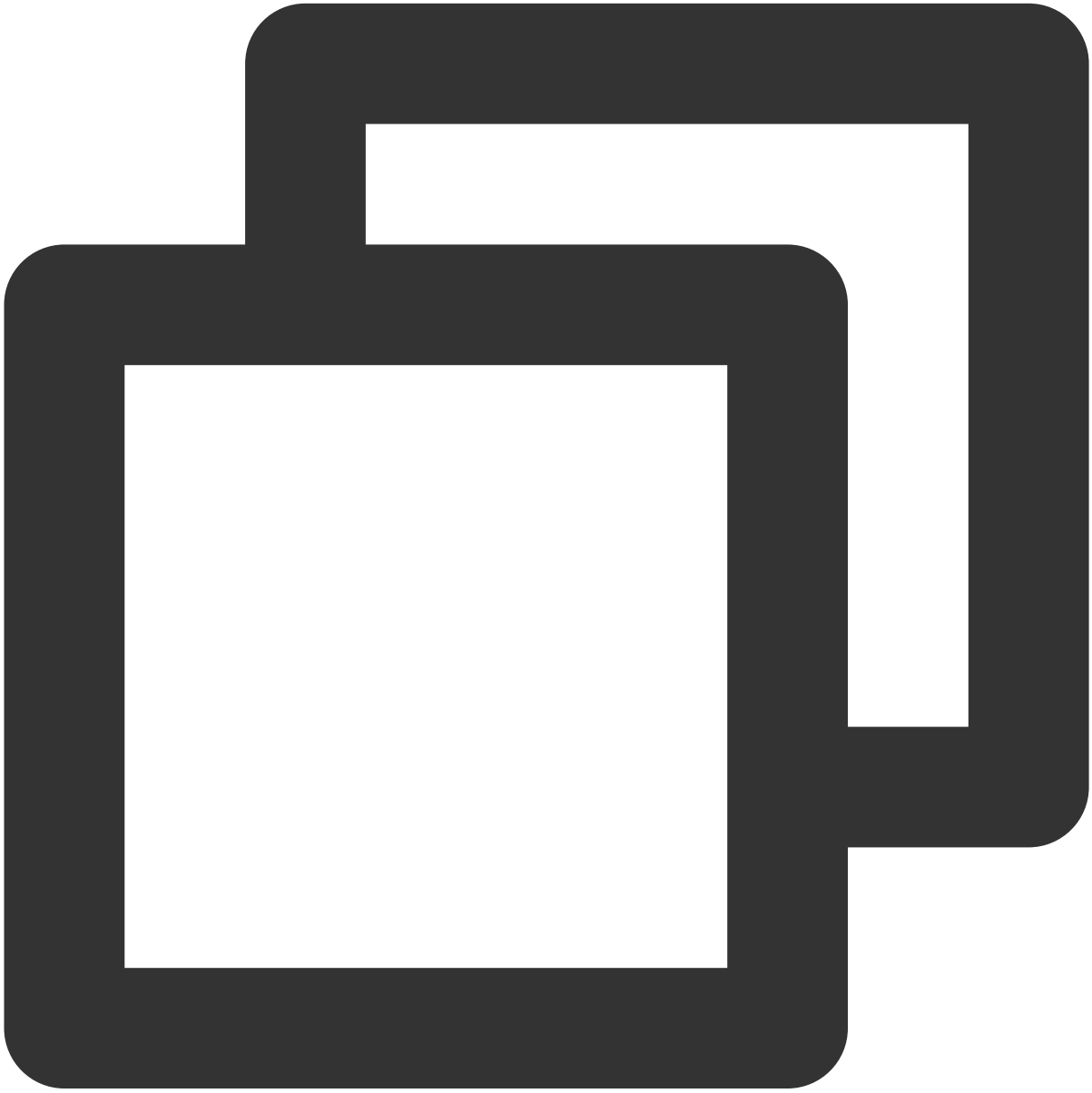
消息相关

消息反馈

消息反馈即为用户对消息进行扩展。
因为所有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考：



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

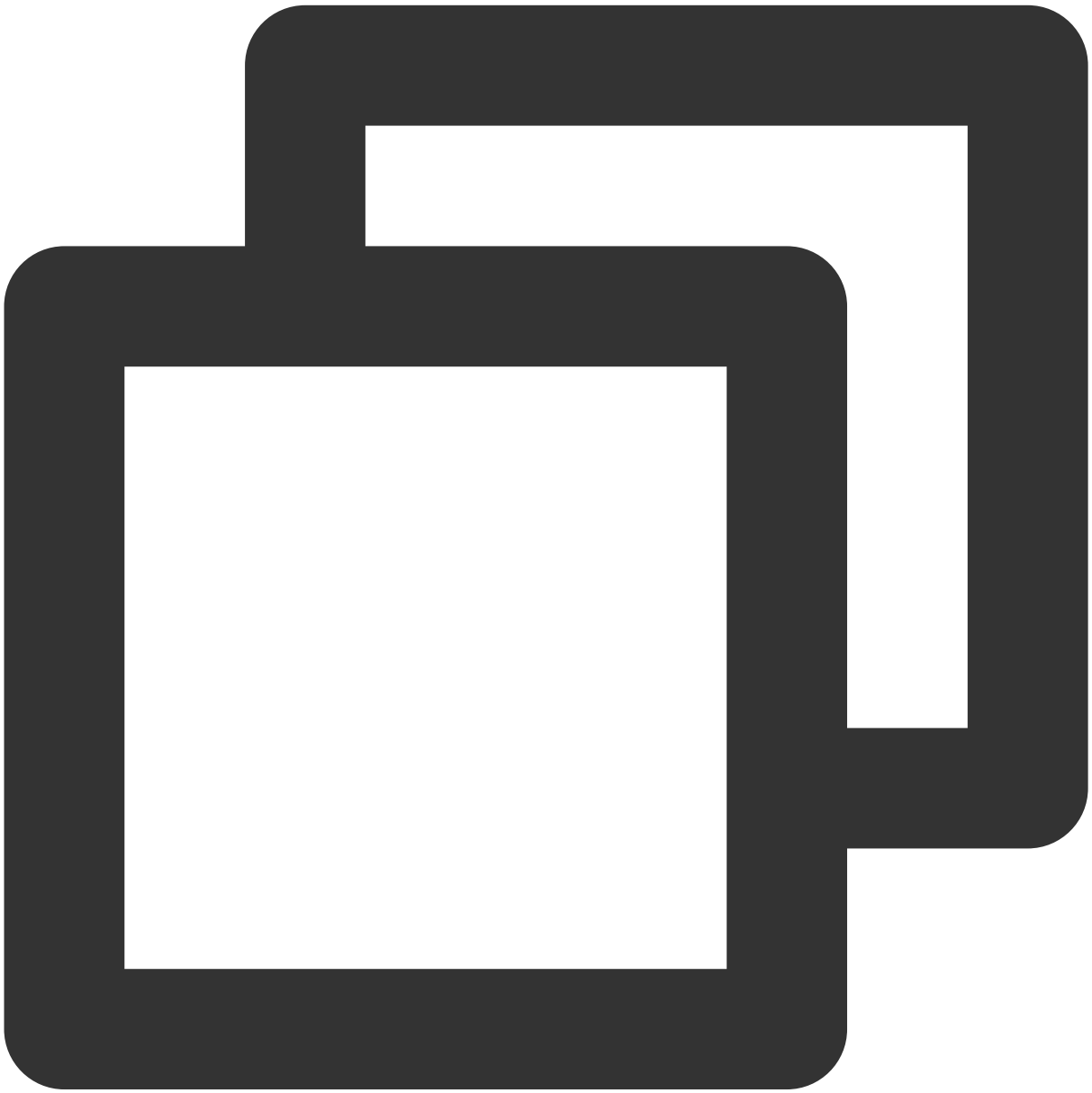


```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new V2TIMCompleteCa
    @Override
    public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
        // 消息修改完成
```

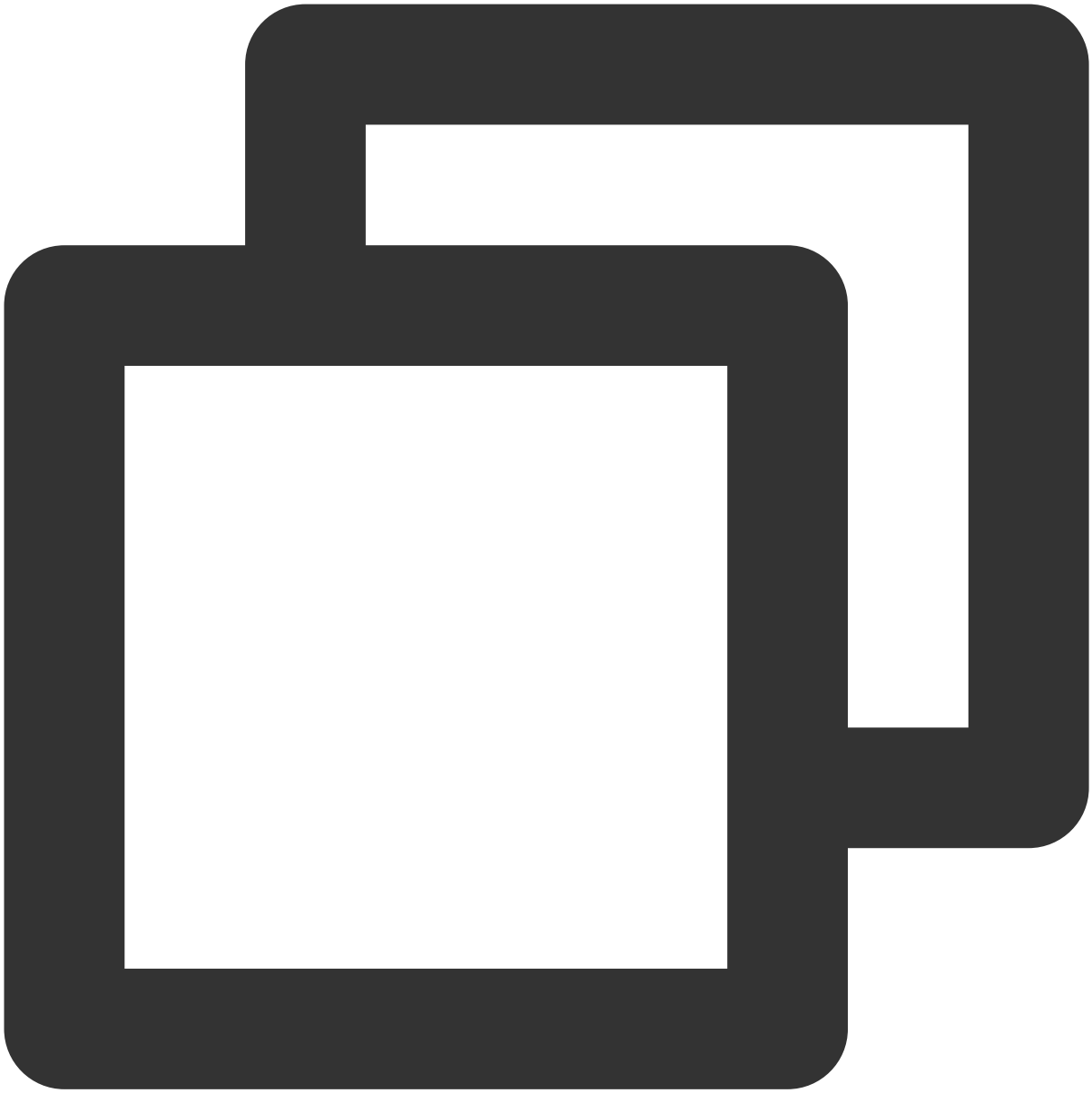
```
    }
});
```

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
}
```



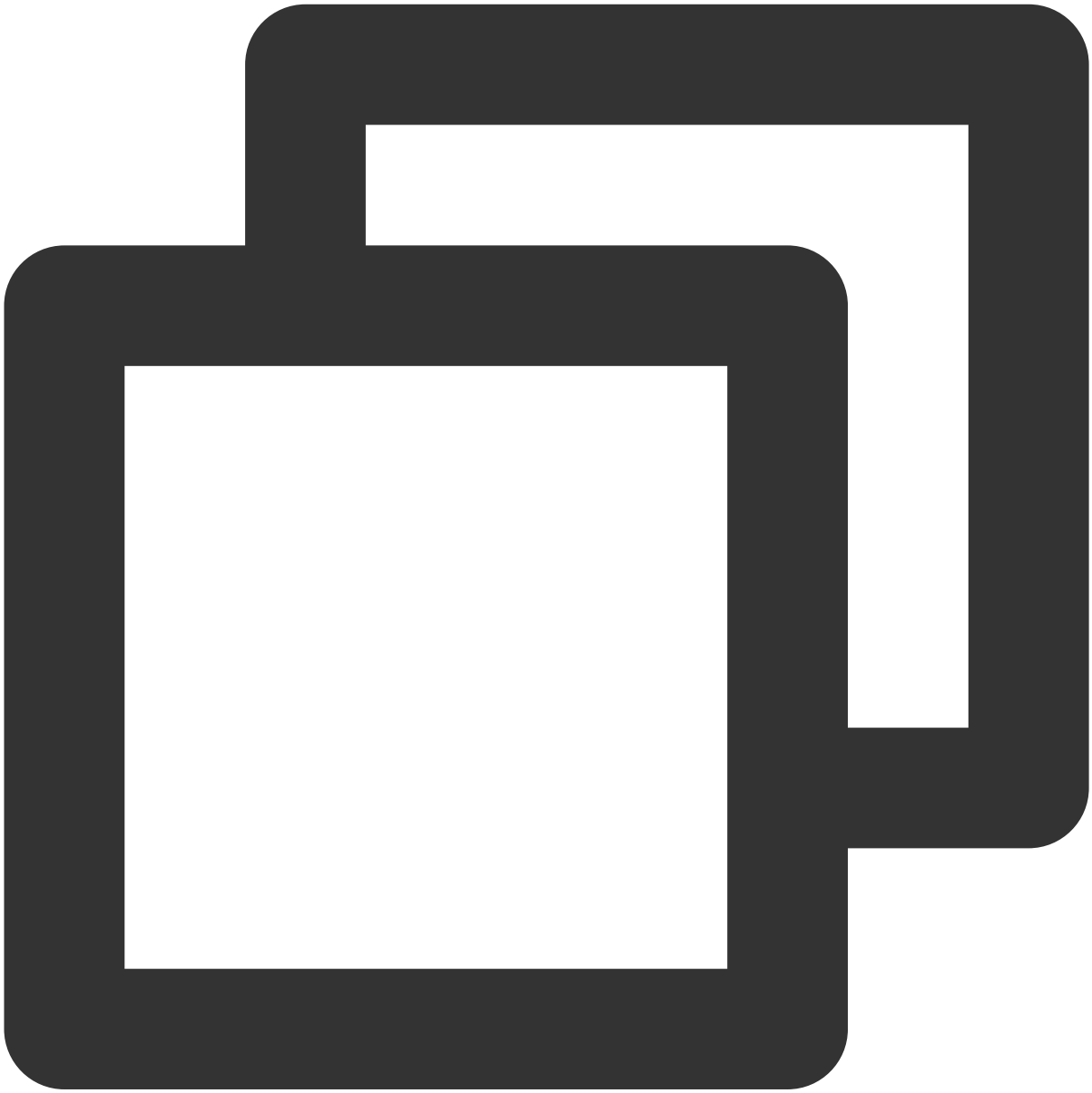
```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new V2TIMCompleteCa
    @Override
    public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
        // 消息编辑完成
    }
});
```

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。

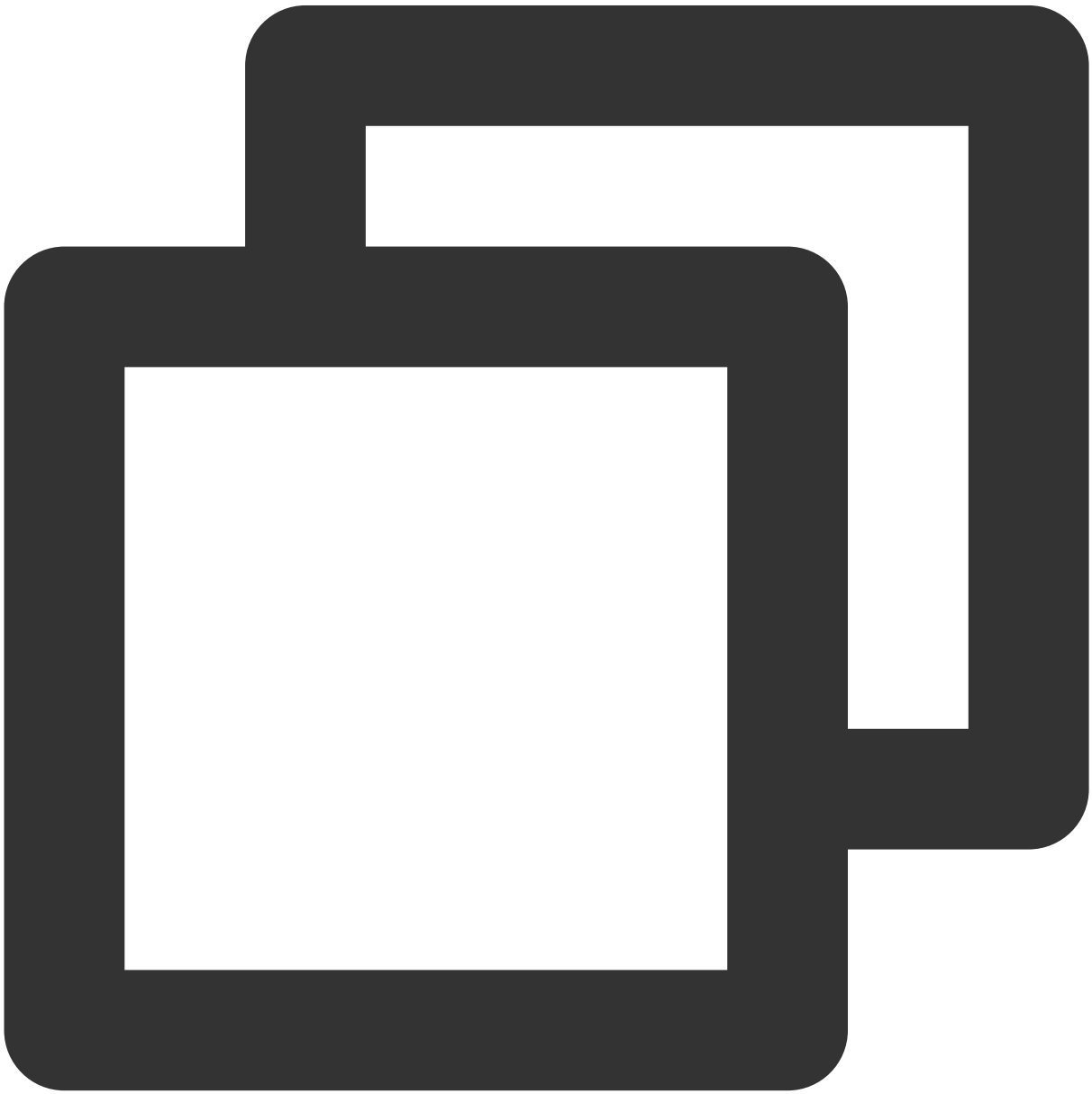
其次进行设置：



```
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<Li
    @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，如上如所示，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。

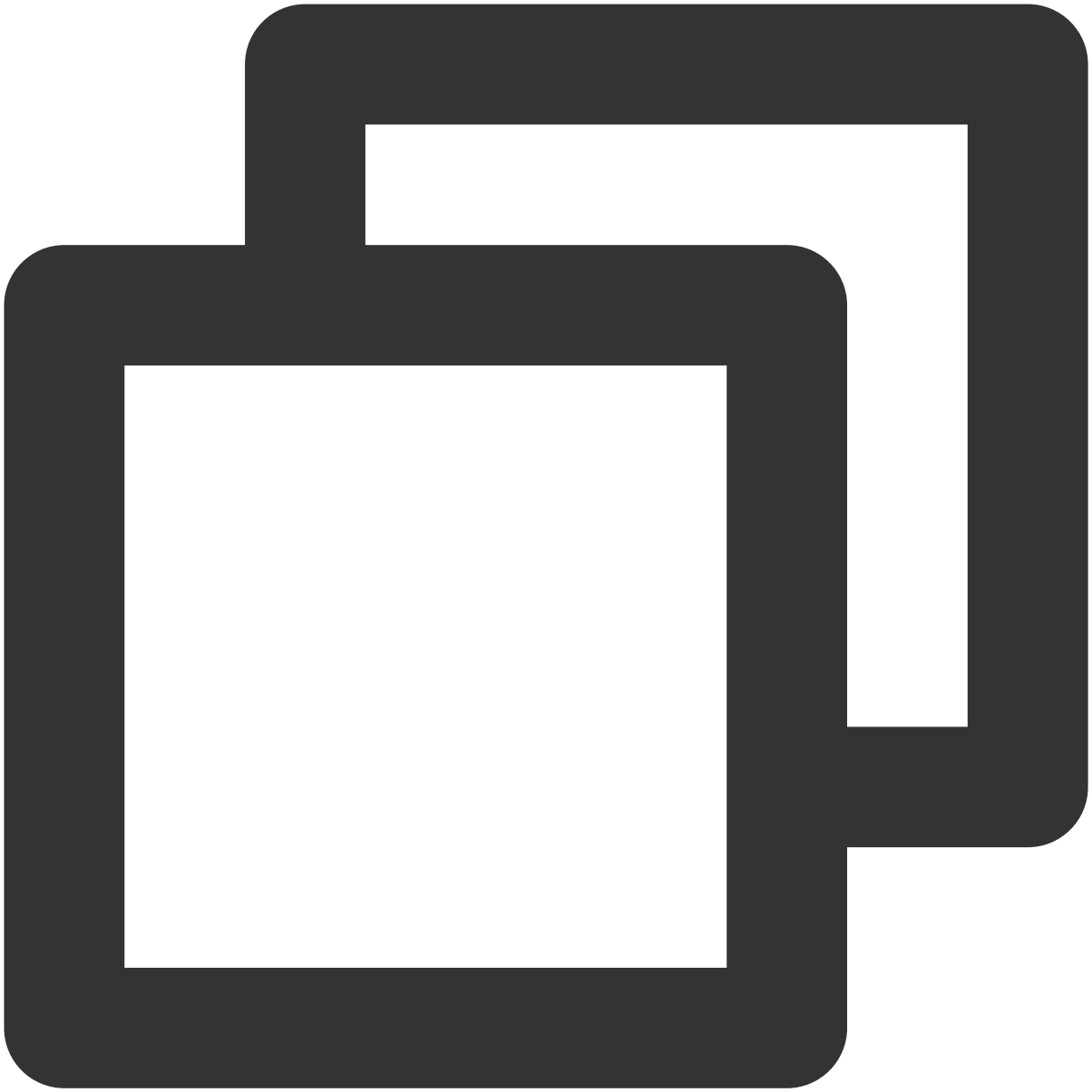
私信

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。

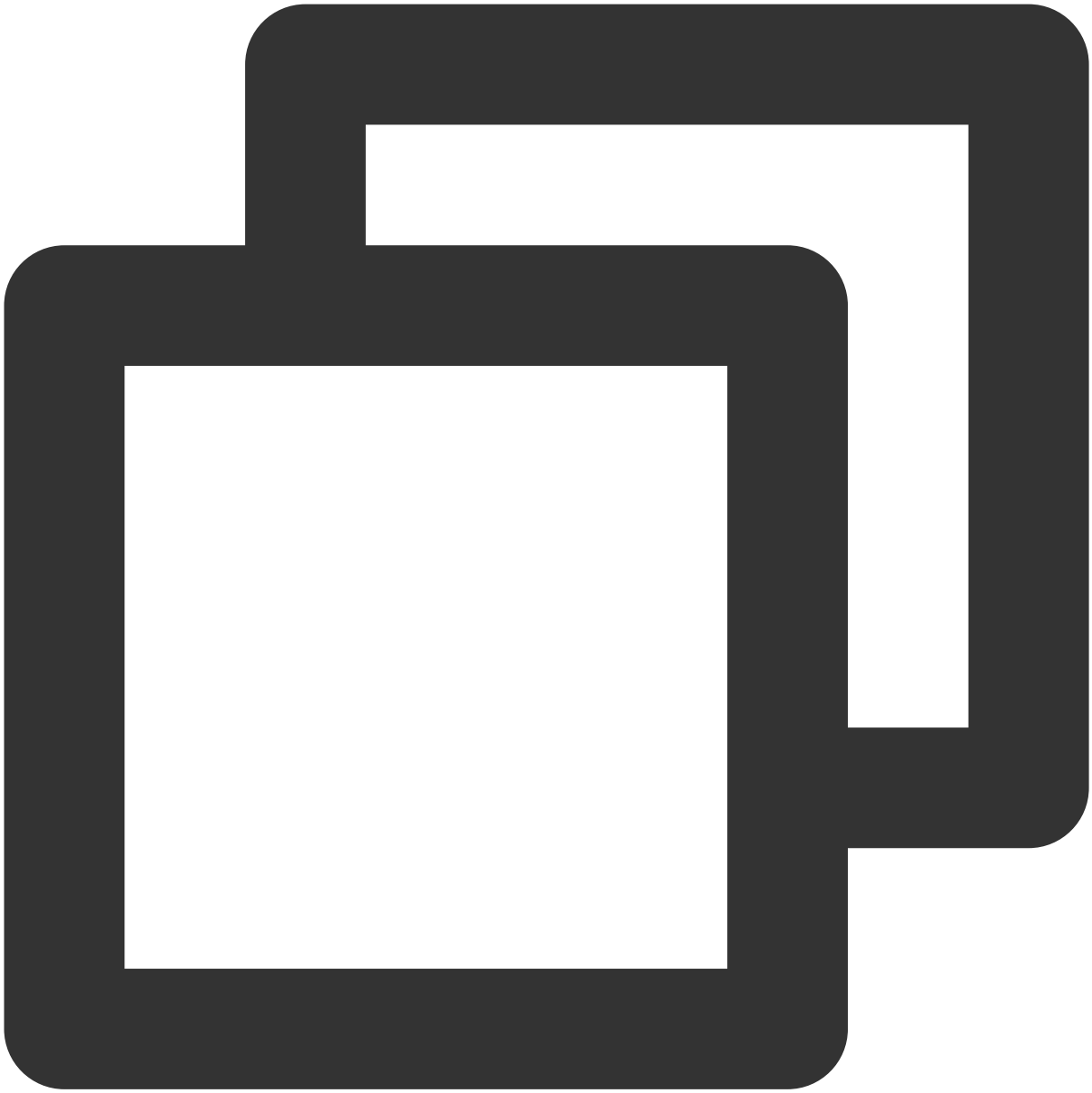
用户也可以先通过 addFriend 接口添加好友，通过 getFriendList 来获取自己的好友列表。

相关代码如下：



```
// 添加好友
V2TIMManager.getFriendshipManager().addFriend(info, new V2TIMValueCallback<V2TIMFri
            @Override
            public void onError(int i, String s) {
                // 添加好友失败
            }

            @Override
            public void onSuccess(V2TIMFriendOperationResult v2TIMFriendOperationRe
               // 添加好友成功
            }
```

```
});
```



```
// 获取好友列表
V2TIMManager.getFriendshipManager().getFriendList(new V2TIMValueCallback<List<V2TIM
            @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

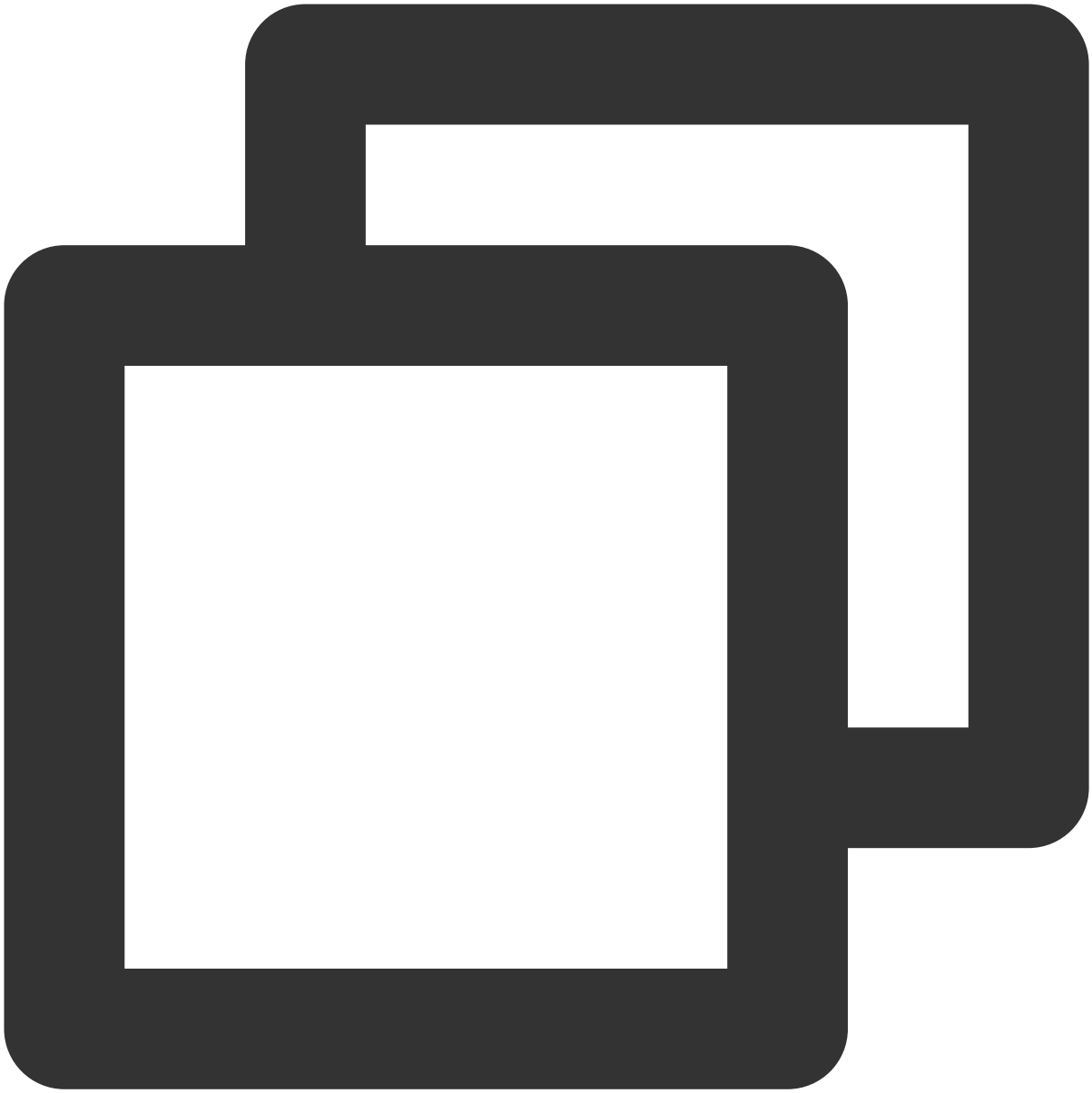
Discord 用户面板的在线状态能力，可以通过腾讯云 IM 提供的在线状态 API 来实现：

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。

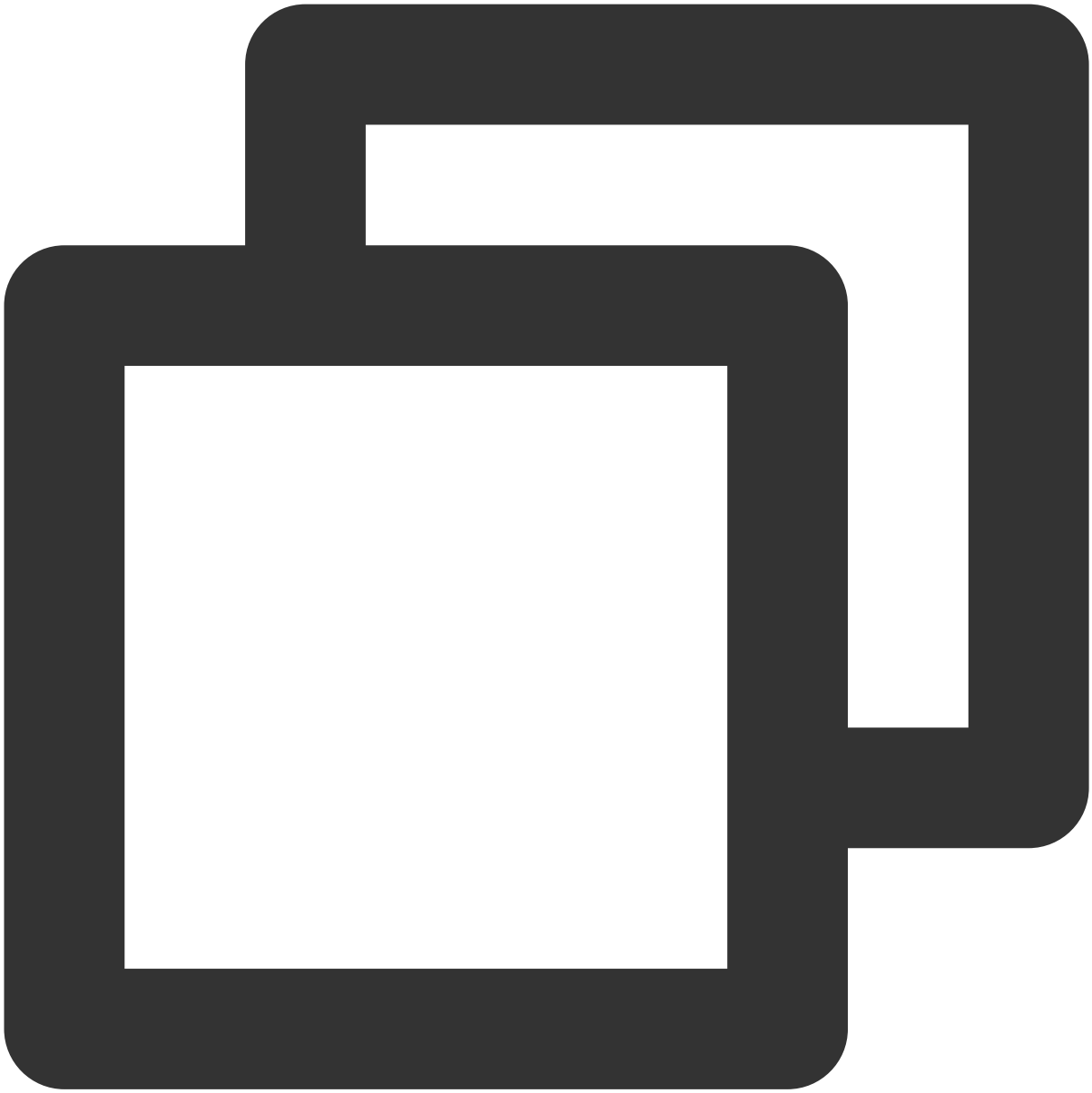
通过 onUserStatusChanged 来监听好友的在线状态改变。

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<V2
    @Override
    public void onSuccess(List<V2TIMUserStatus> v2TIMUserStatuses) {

    }
```

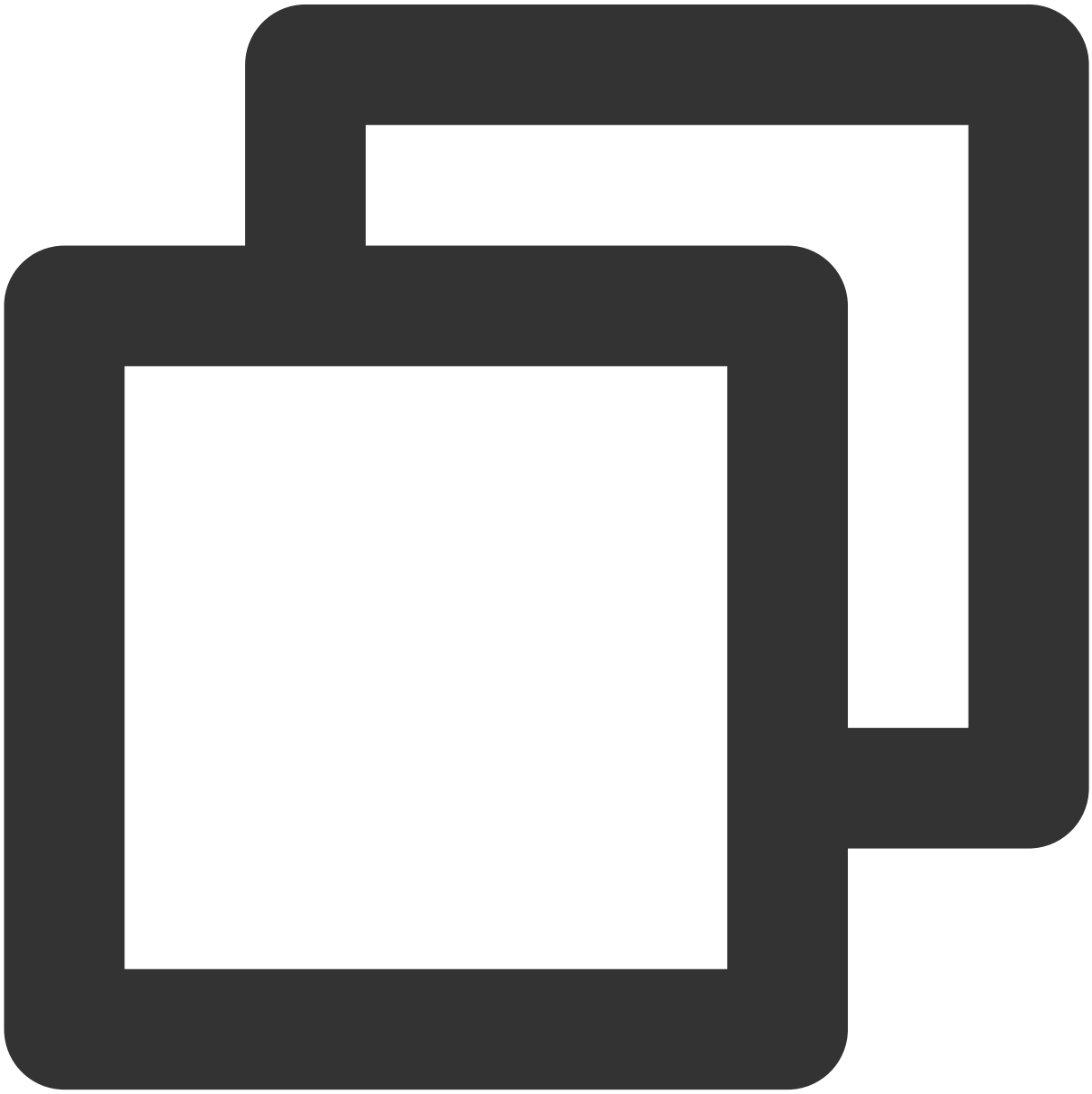
```
    @Override
    public void onError(int i, String s) {

    }
});
```

个人信息相关

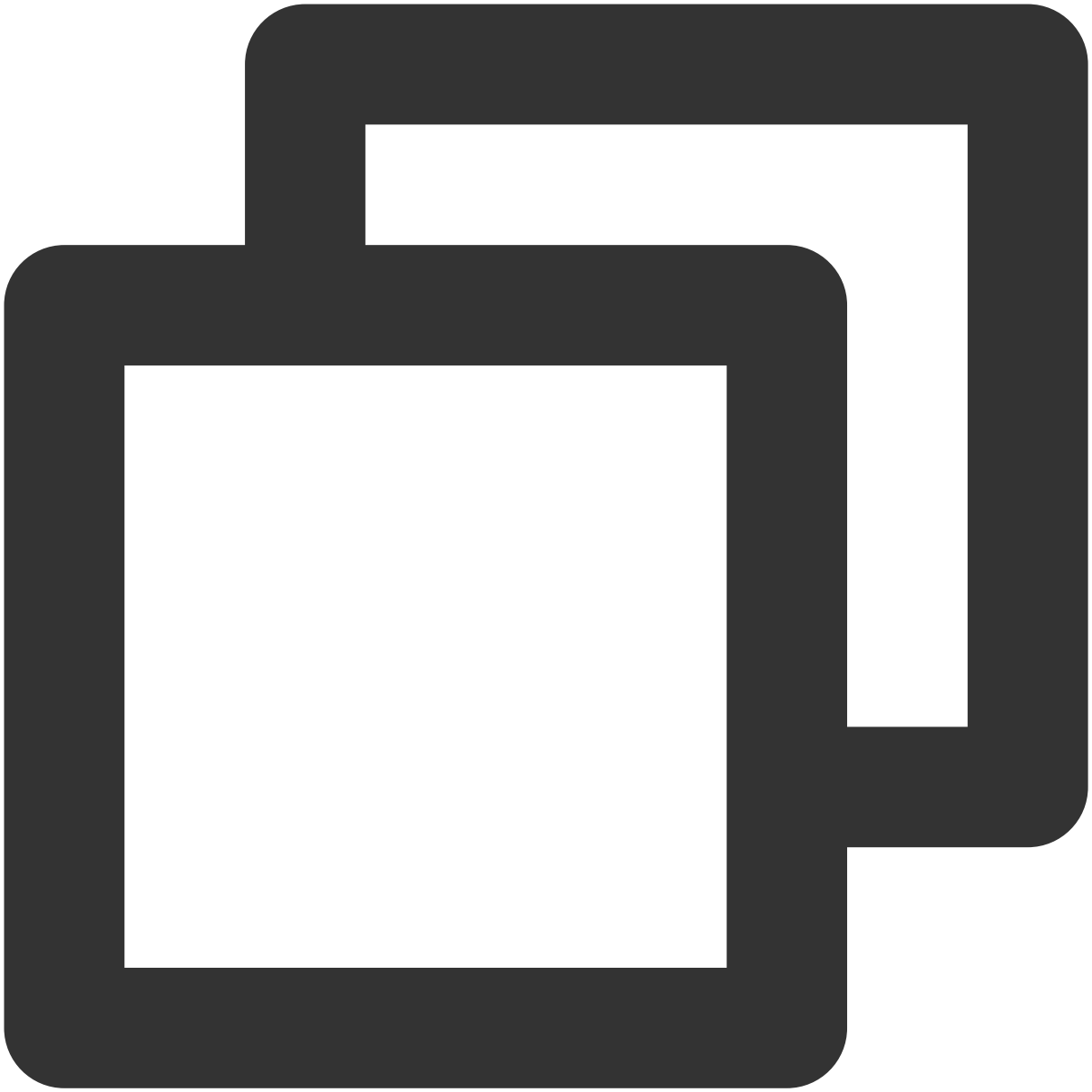
个人信息相关的功能可通过腾讯云 IM 提供的资料关系链相关的 API 实现：

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<V2T
    @Override
    public void onError(int i, String s) {

    }

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

```
    }
});
```

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 可参考官文档 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，这是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。

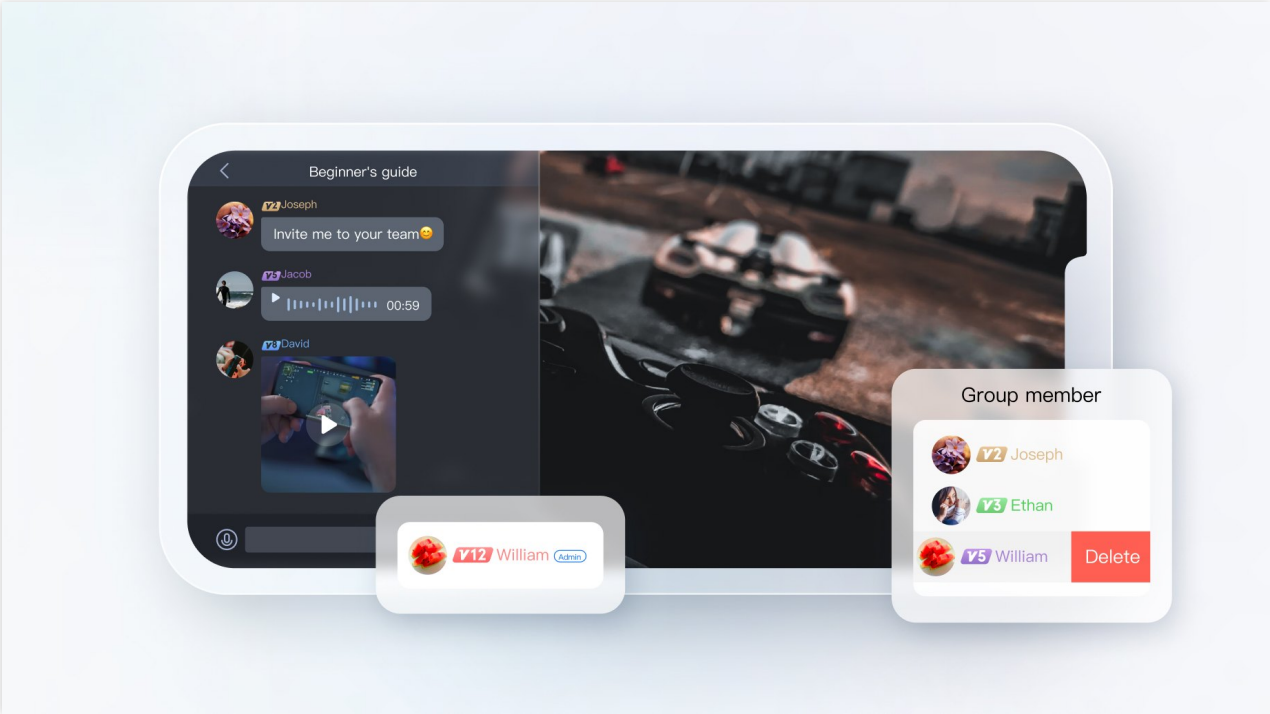
敏感词校验

在发送信息、配置资料时，需要对内容进行过滤，腾讯云IM也提供了这样的方案，帮助用户更加合规的使用 IM。

游戏内集成 IM 指南

最近更新时间：2024-02-19 11:22:55

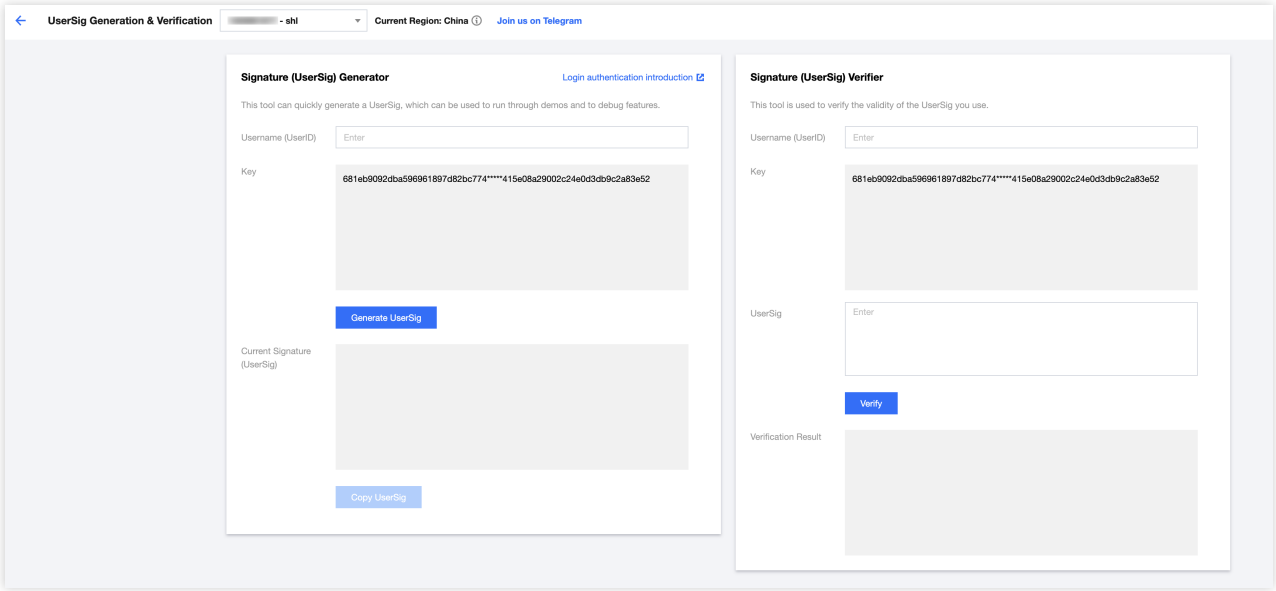
在游戏中，即时通信需求很常见，在多人游戏中即时聊天也成为了必不可少的功能。由于游戏平台本身所涉及到的多种群组类型、自定义消息类型（如游戏内道具赠送、交易等业务）、全球接入等需求往往比较复杂，本文梳理了在搭建游戏聊天时过程中常见的需求的实现方法，以及可能遇到的问题、需要注意的细节等，希望能帮助开发者们快速的理解业务，实现需求。



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，提供最佳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 类型的恶存储长度不得超过500自解

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：单次最多倒入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

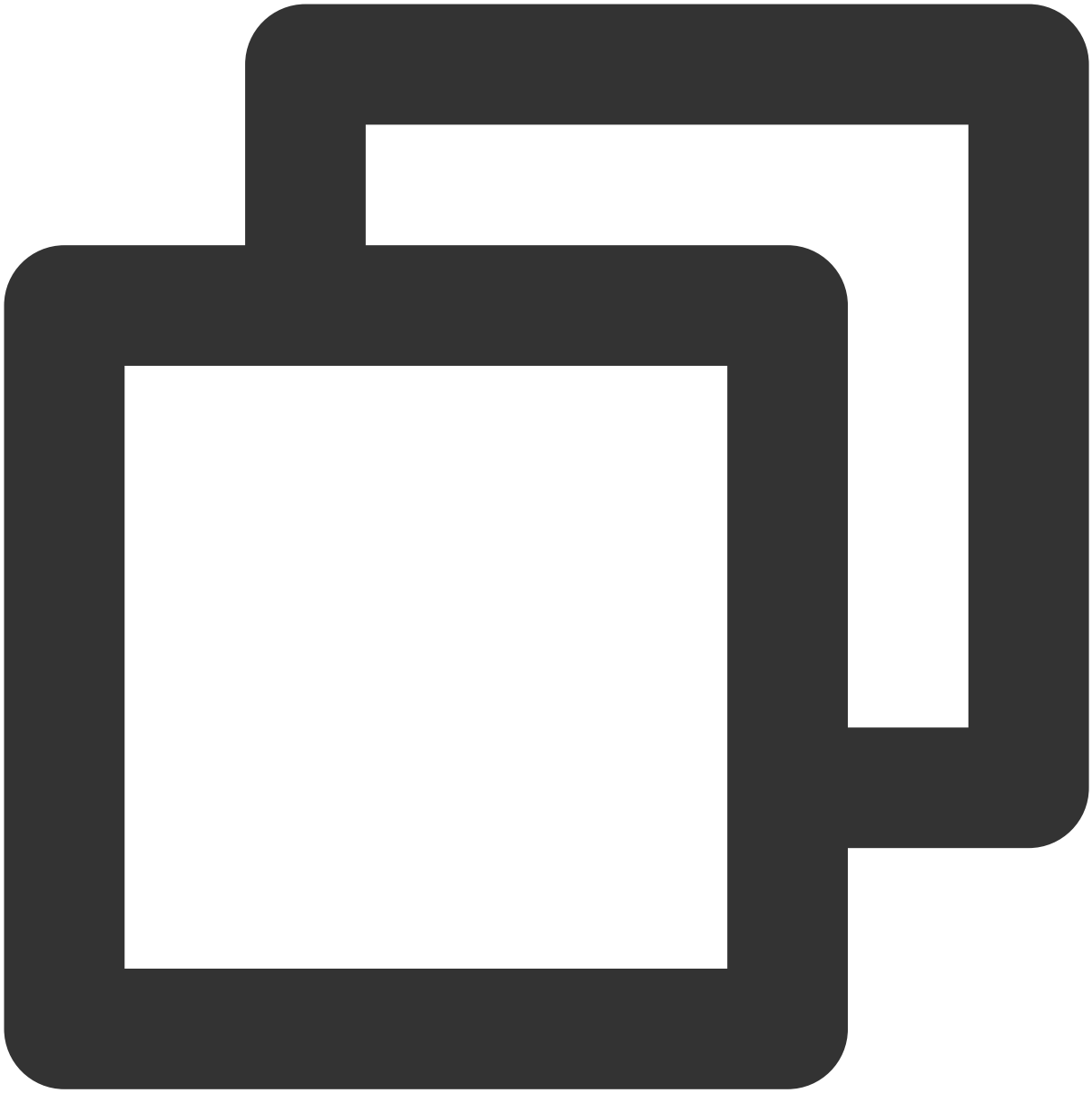
更多关于使用限制请参见 使用限制。

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`移除舰艇。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。

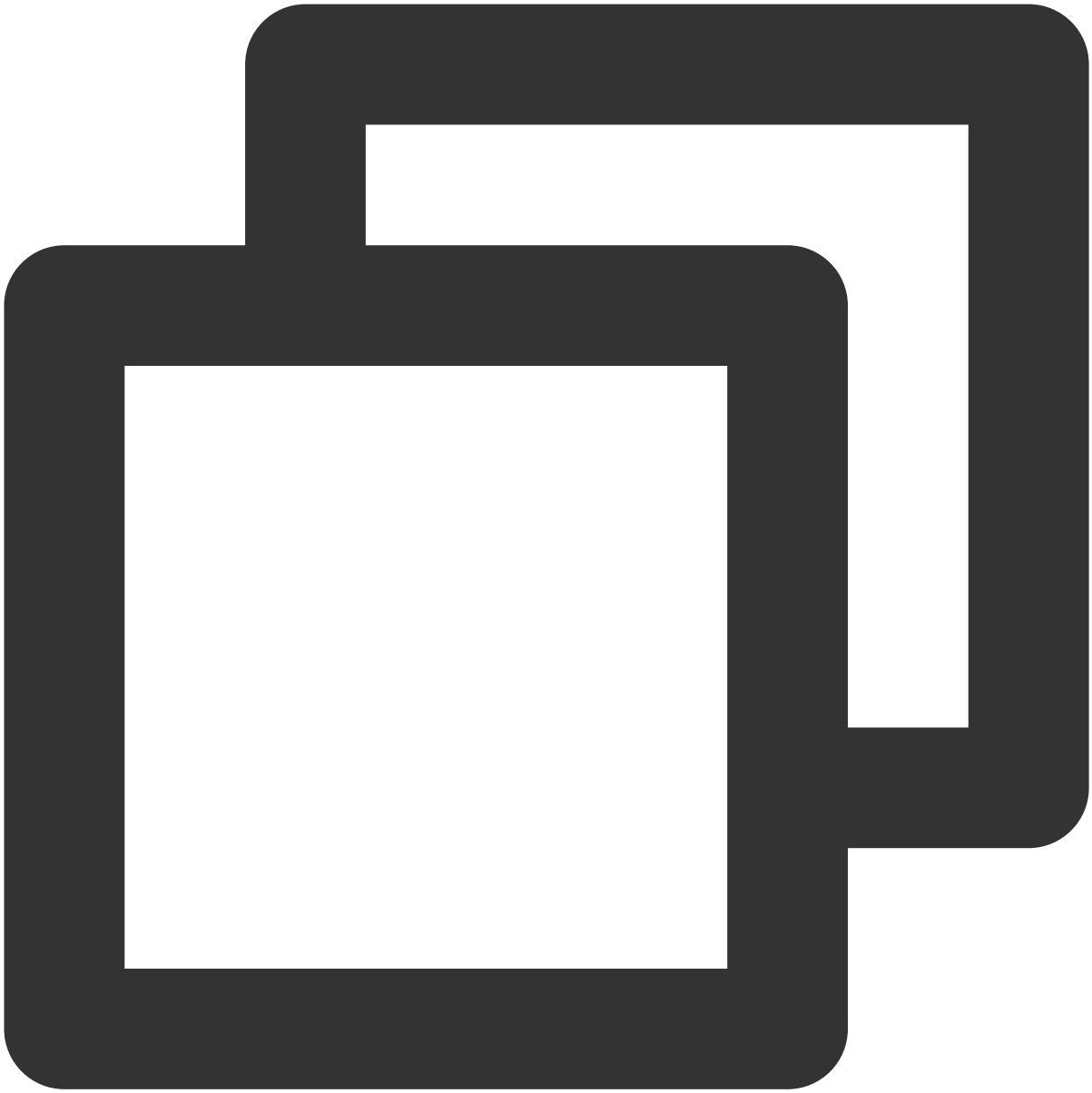
系统发送邮件：系统向用户发系统邮件可以通过 服务端 API 的几种方式，下面列出不同方法的特点：

方法	特点	应用场景
单发单聊消息	向指定账号发消息，接收方看到的发送着不是管理员，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 IM 旗舰版功能，需 购买旗舰版 并在 控制台 > 功能配置 > 登录与消息 > 全员推送设置，打开开关后方可使用。

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。

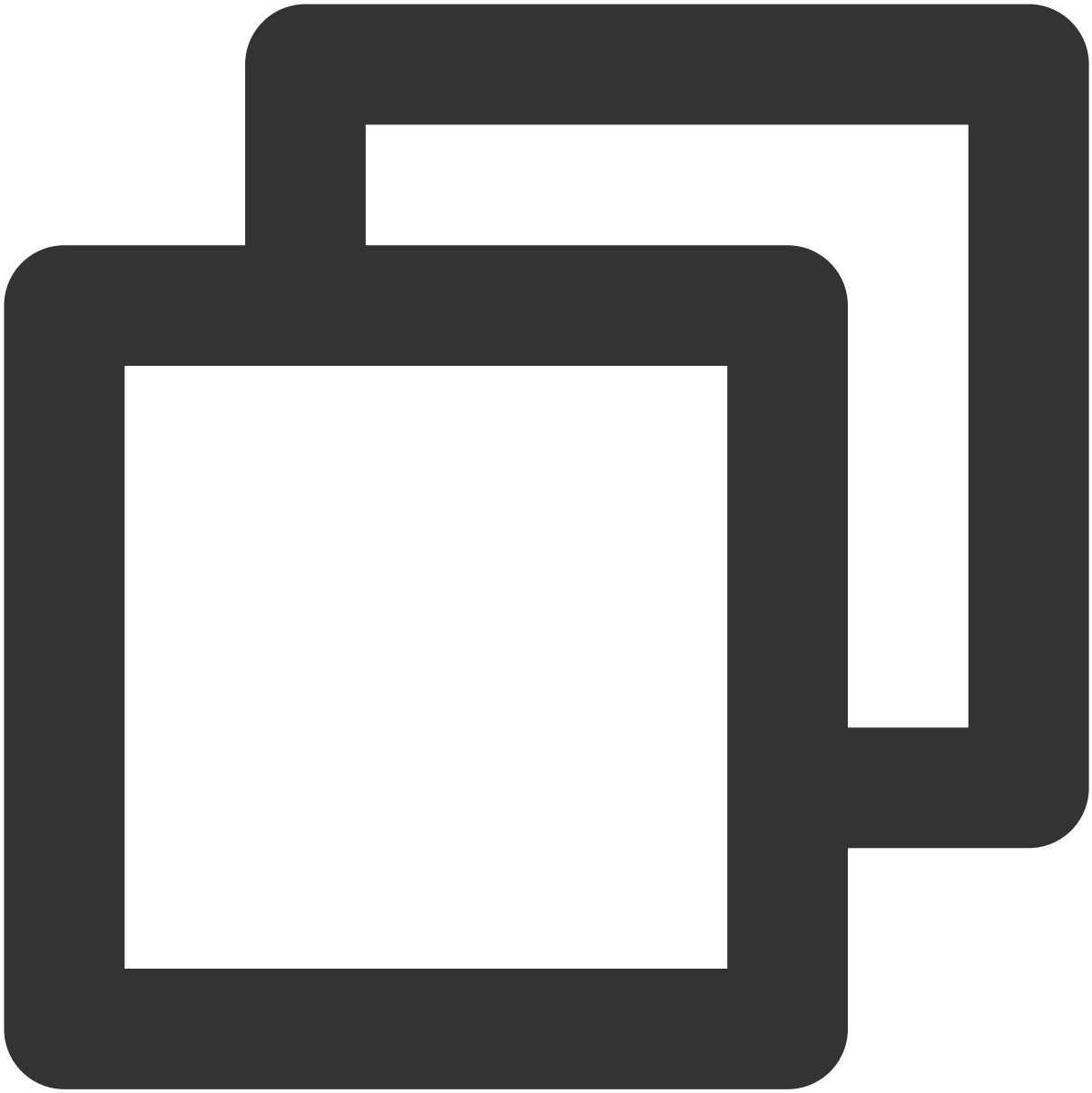
邮件列表

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，旗舰版为 30 天。专业版和旗舰版支持延长时长。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 增值服务资费。

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：

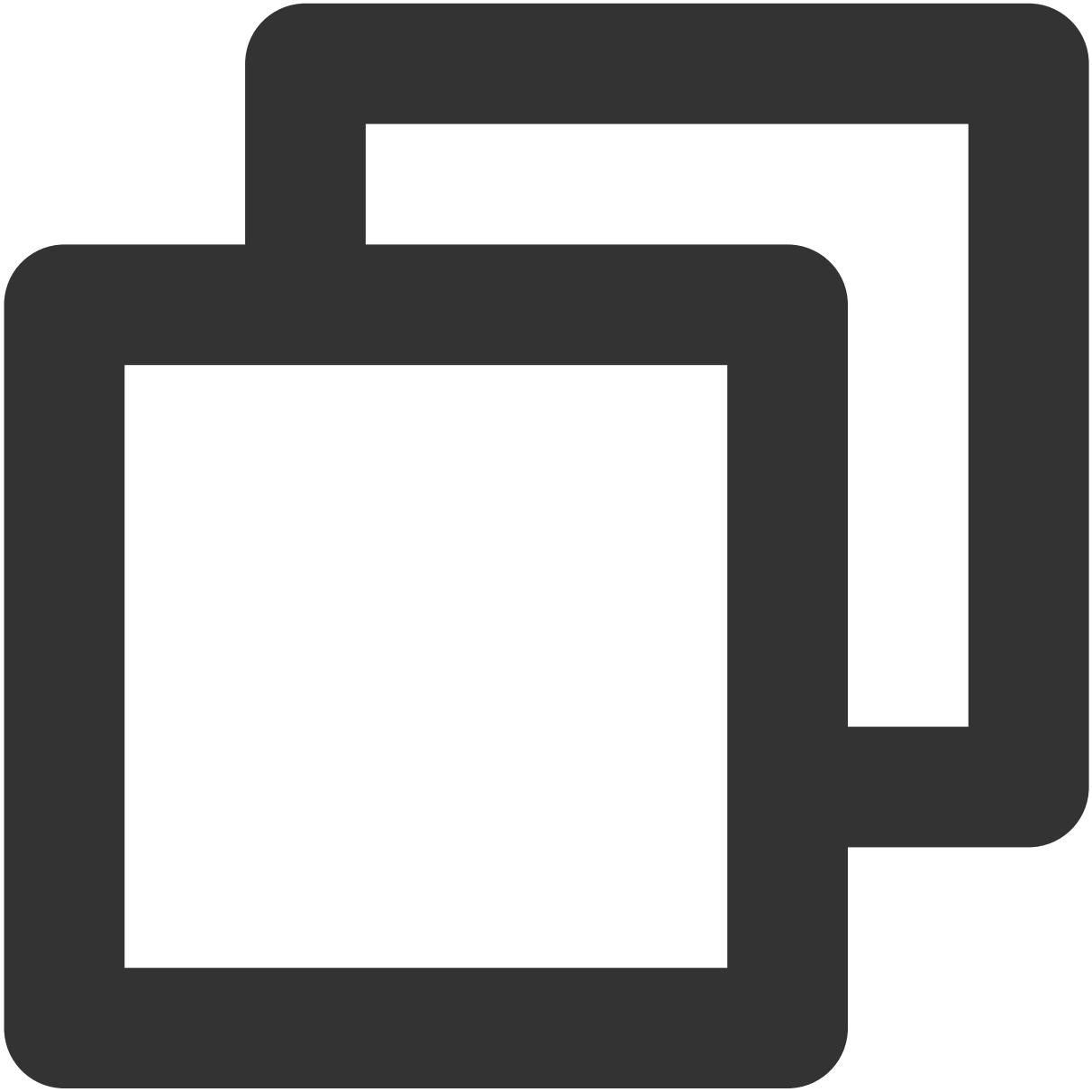


```
// 拉取单聊历史消息
// 首次拉取，msg_getmsglist_param_last_msg 设置为 null
// 再次拉取时，msg_getmsglist_param_last_msg 可以使用返回的消息列表中的最后一条消息
var get_message_list_param = new MsgGetMsgListParam
{
    msg_getmsglist_param_last_msg = LastMessage
};
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
// 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：



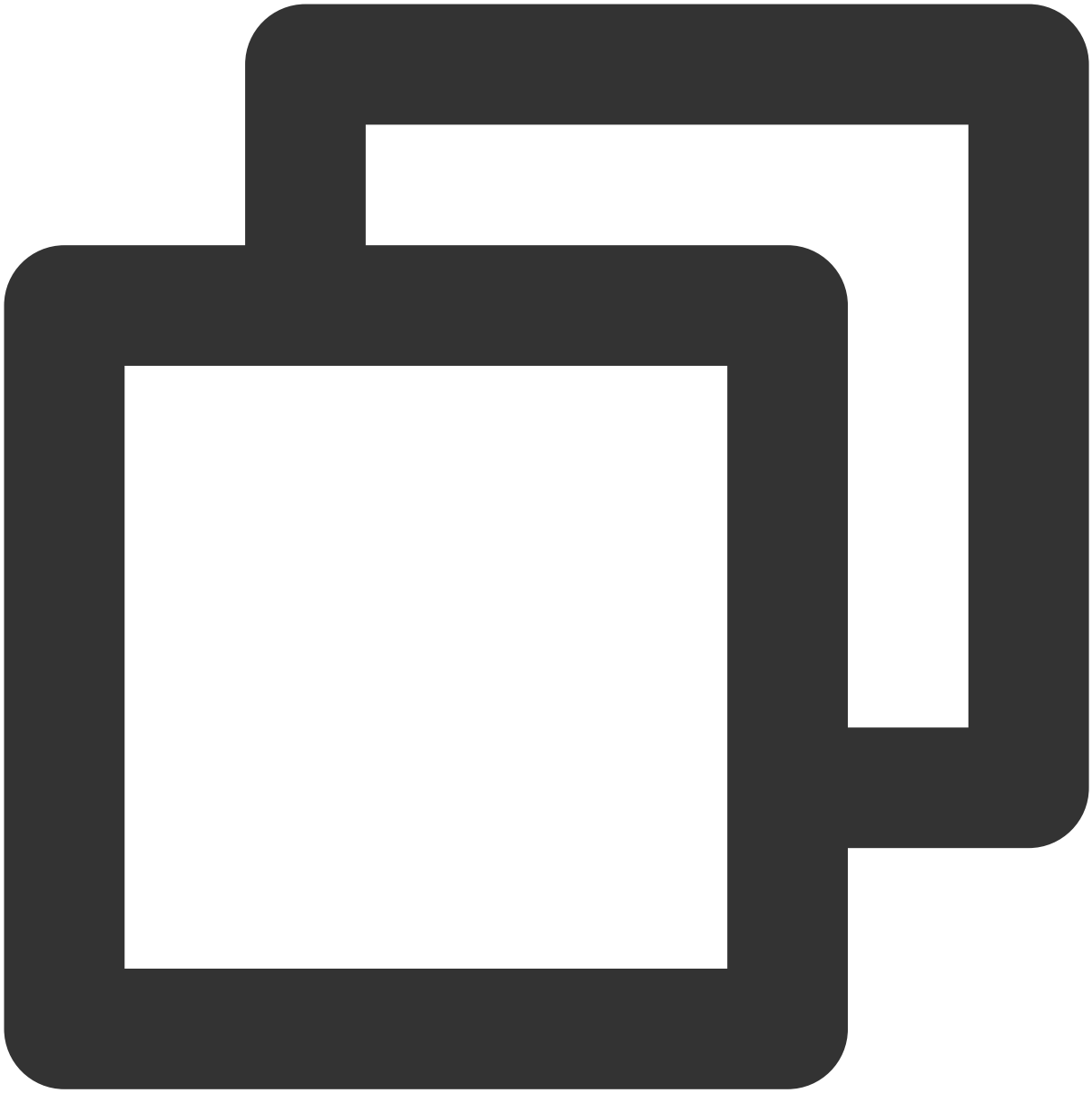
```
// 获取全部未读数
TIMResult res = TencentIMSDK.ConvGetTotalUnreadMessageCount((int code, string desc,
 // 处理异步逻辑
});
```

```
// 未读计数变更通知
TencentIMSDK.SetConvTotalUnreadMessageCountChangedCallback((int total_unread_count,
 // 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
 // 处理异步逻辑
});
```

更多内容请参见 Unity-会话未读消息数。

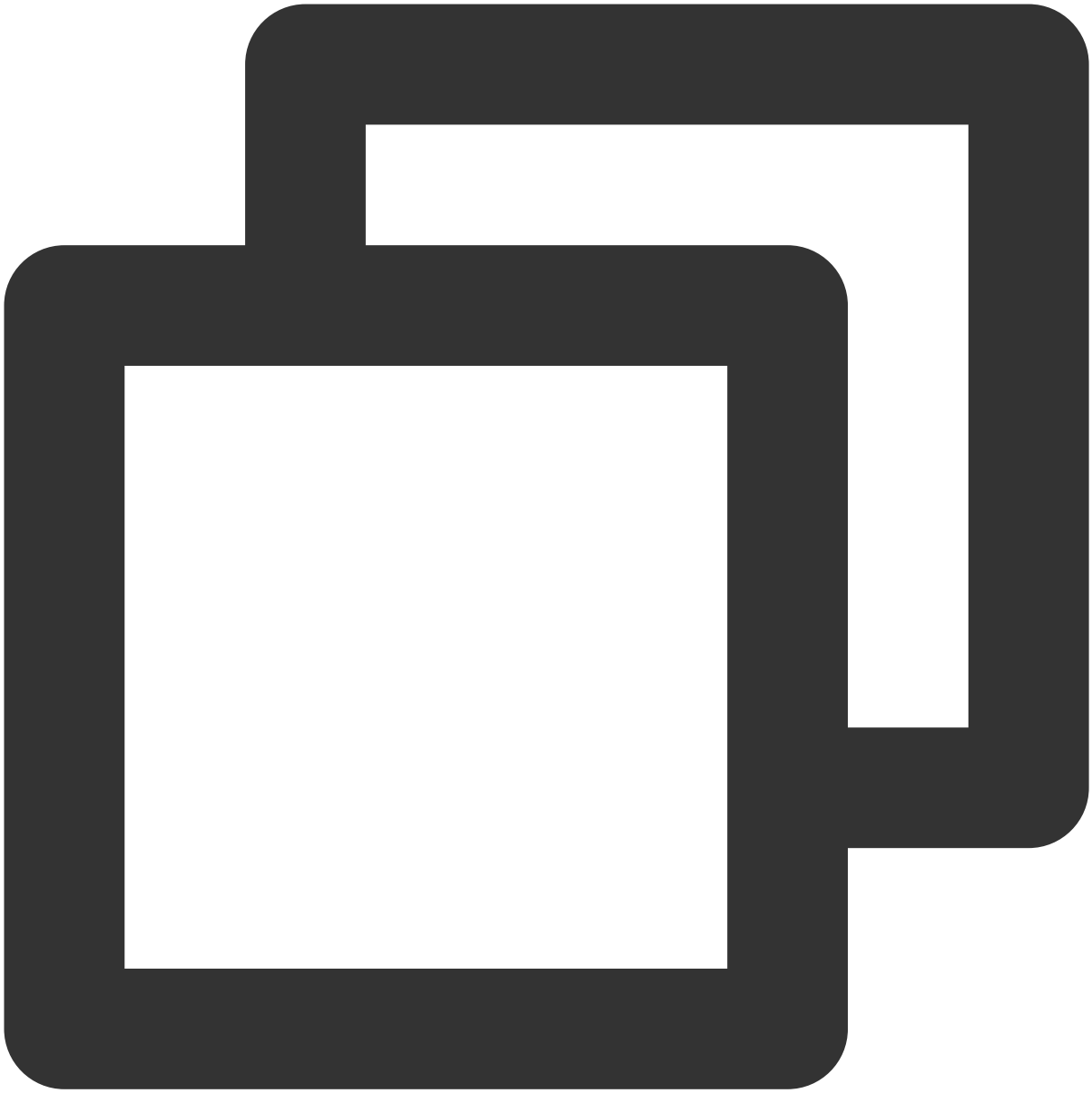
全员邮件

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&
```

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

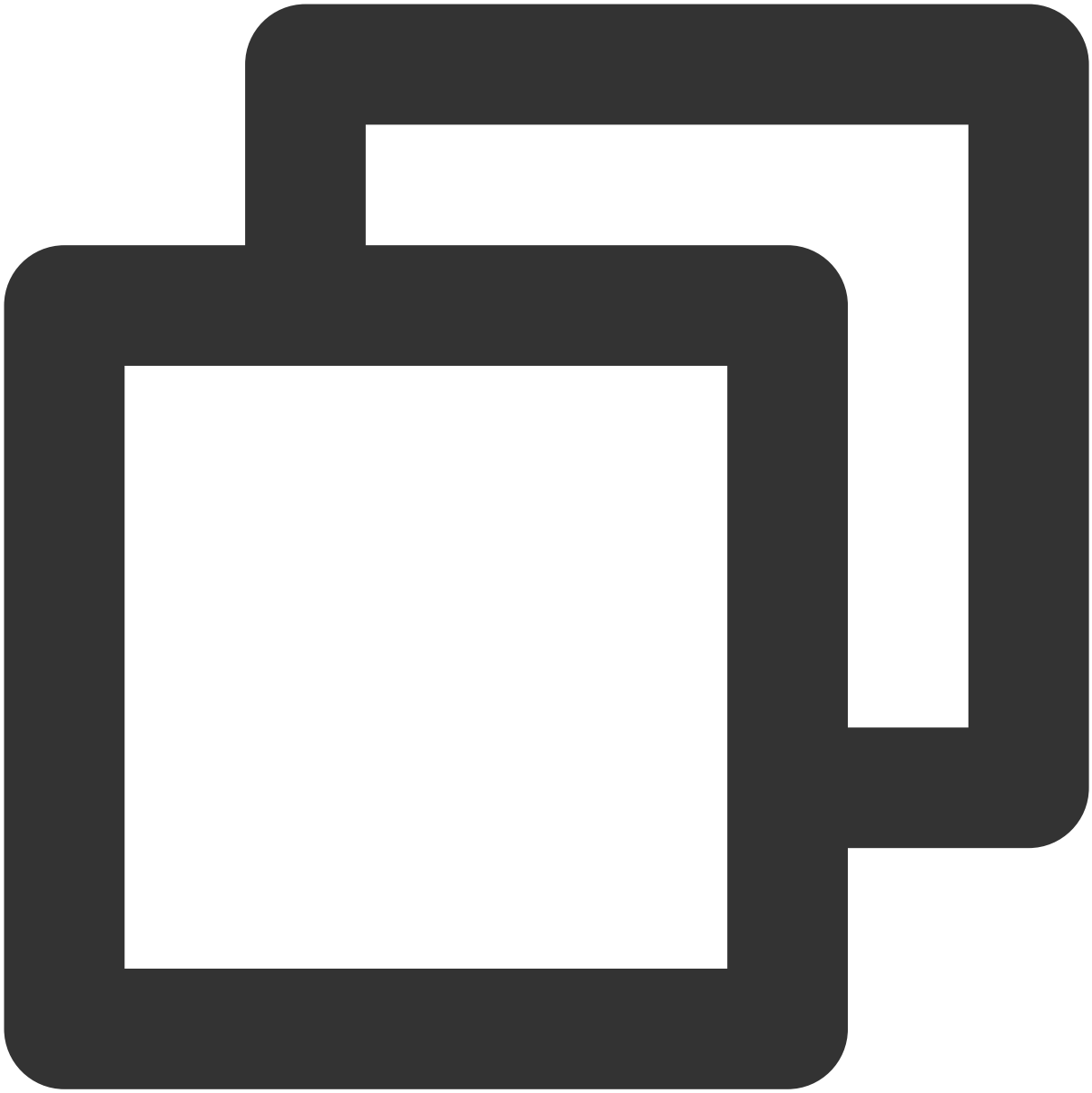
对于历史邮件，邮件存储有效期为：体验版/专业版为7天，旗舰版为30天，专业版和旗舰版支持延长时长。对于历史存储的更多内容请参见 历史消息存储时长配置。
此外，在服务端发送消息时，可通过在请求包内设置 `MsgLifeTime` 设置消息离线保存时长（最长为7天）。若该字段为0，则表示消息只发在线用户，不离线保存。 更多内容请参见 服务端 API。

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。

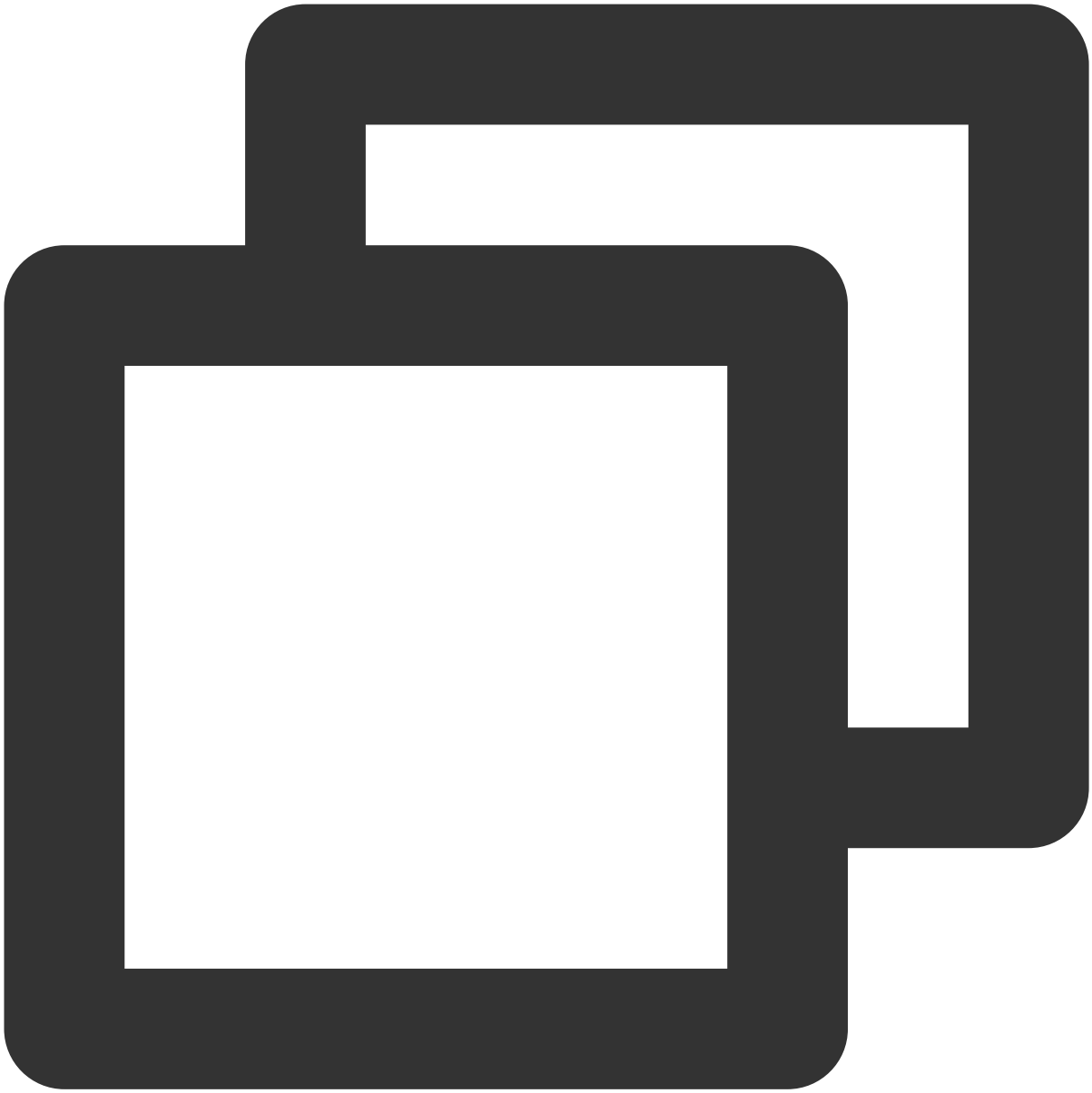
组队场景

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&id
```

请求包基础形式示例如下：

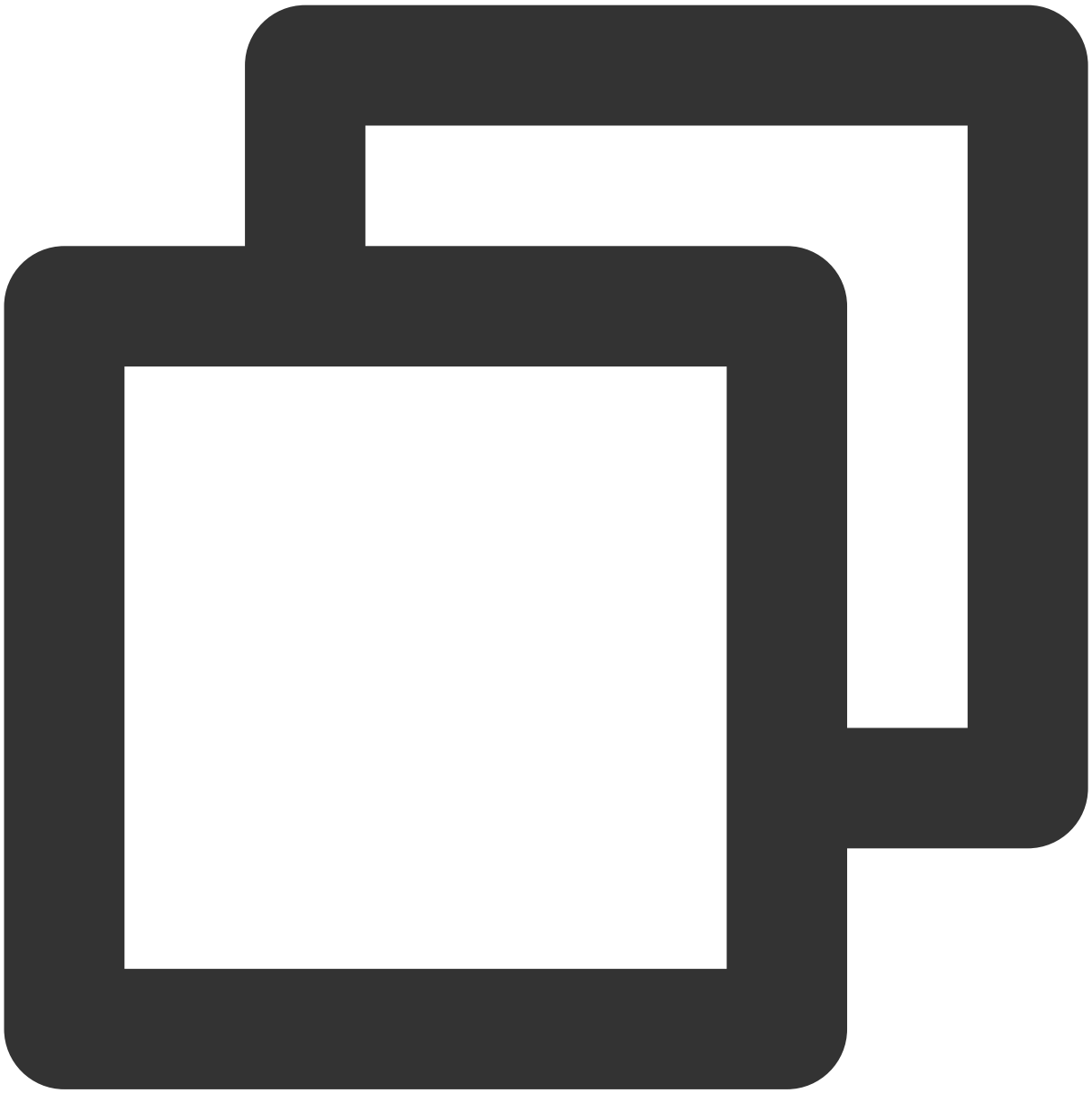


```
{
  "Owner_Account": "leckie", // 群主的 UserId（选填）
  "Type": "Public", // 群组类型：Private/Public/ChatRoom/AVChatRoom/Community
  "Name": "TestGroup", // 群名称（必填）
  "MaxMemberCount":5 // 最大群成员数量（选填）
}
```

说明：

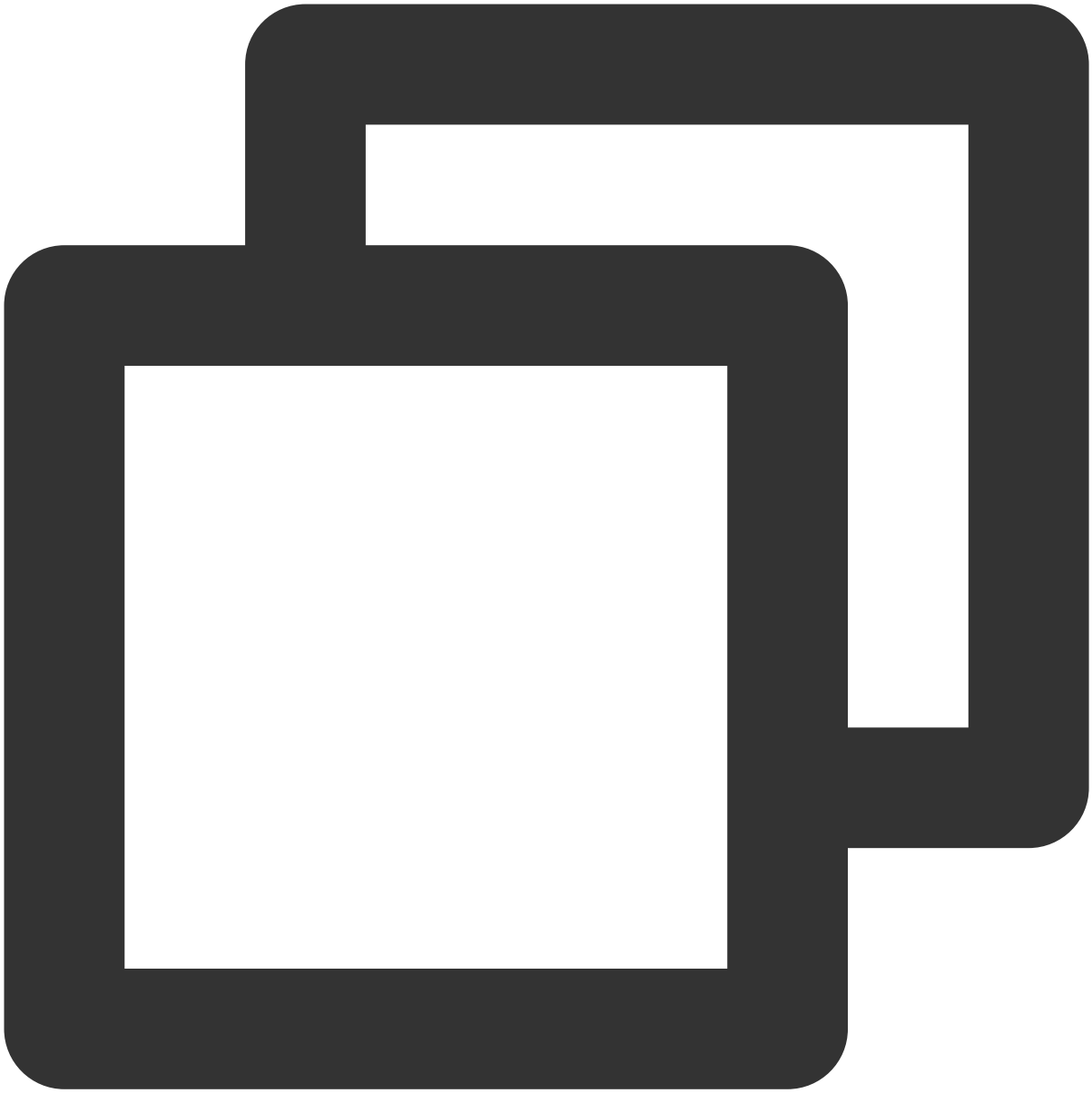
App 中同时存在的所有群组数量最多为10万，若超过10万则需要支付一定费用。详情请参见 价格说明。更多关于服务端创建群组请参见 创建群组。

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=8888888
```

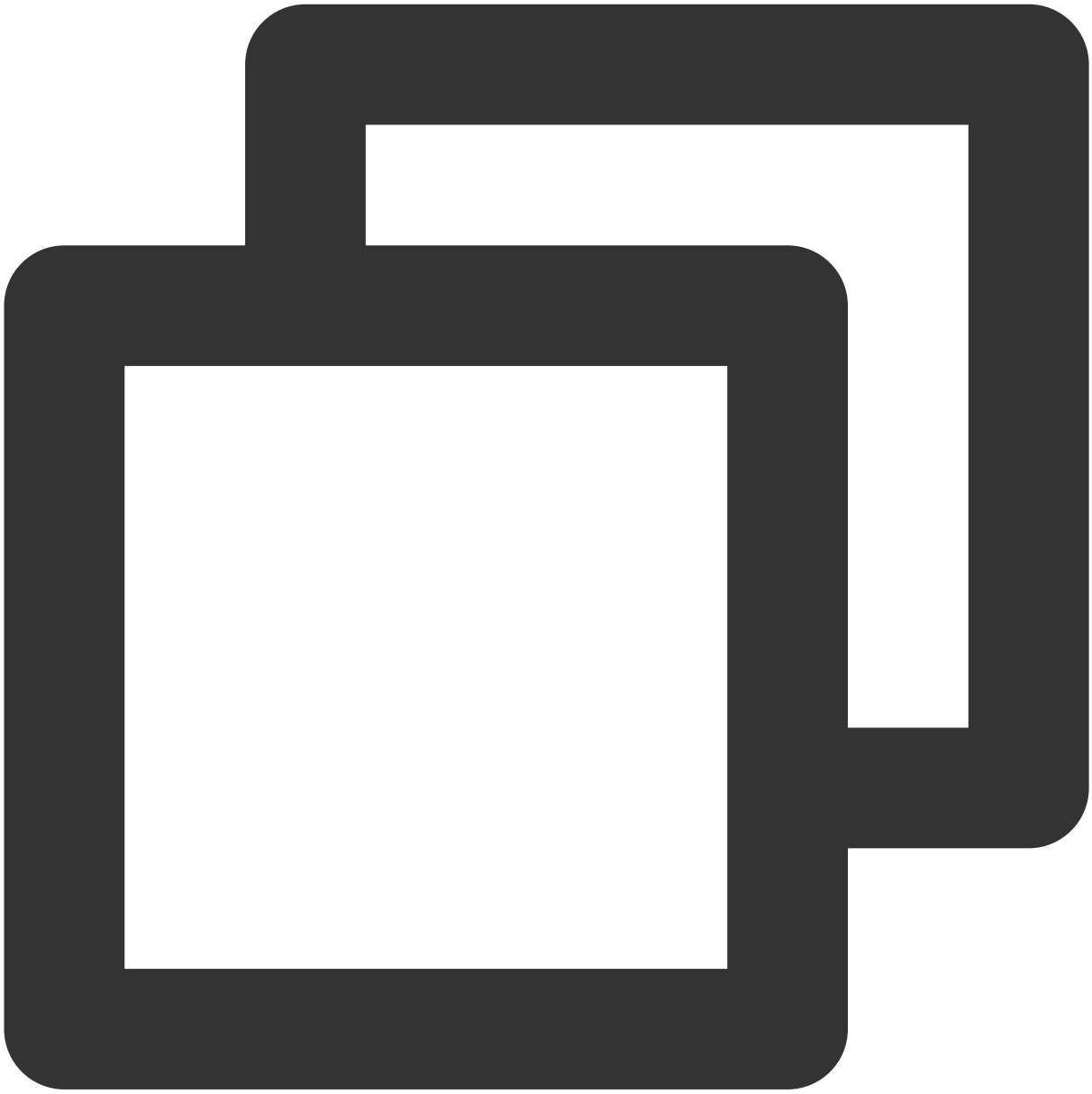
请求包如下：



```
{
  "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"
  }]
}
```

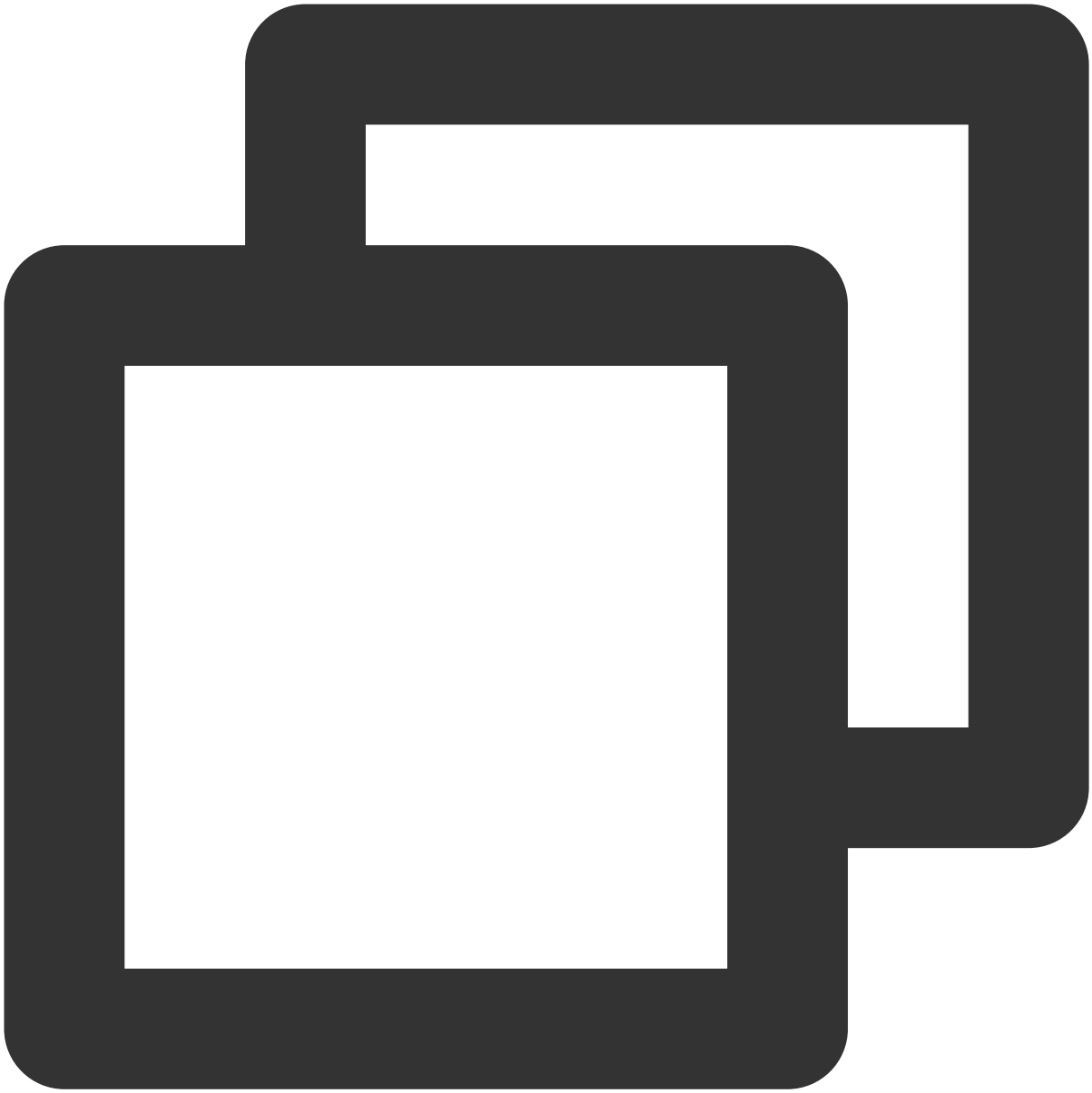
更多内容请参见 增加群成员。

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&content
```

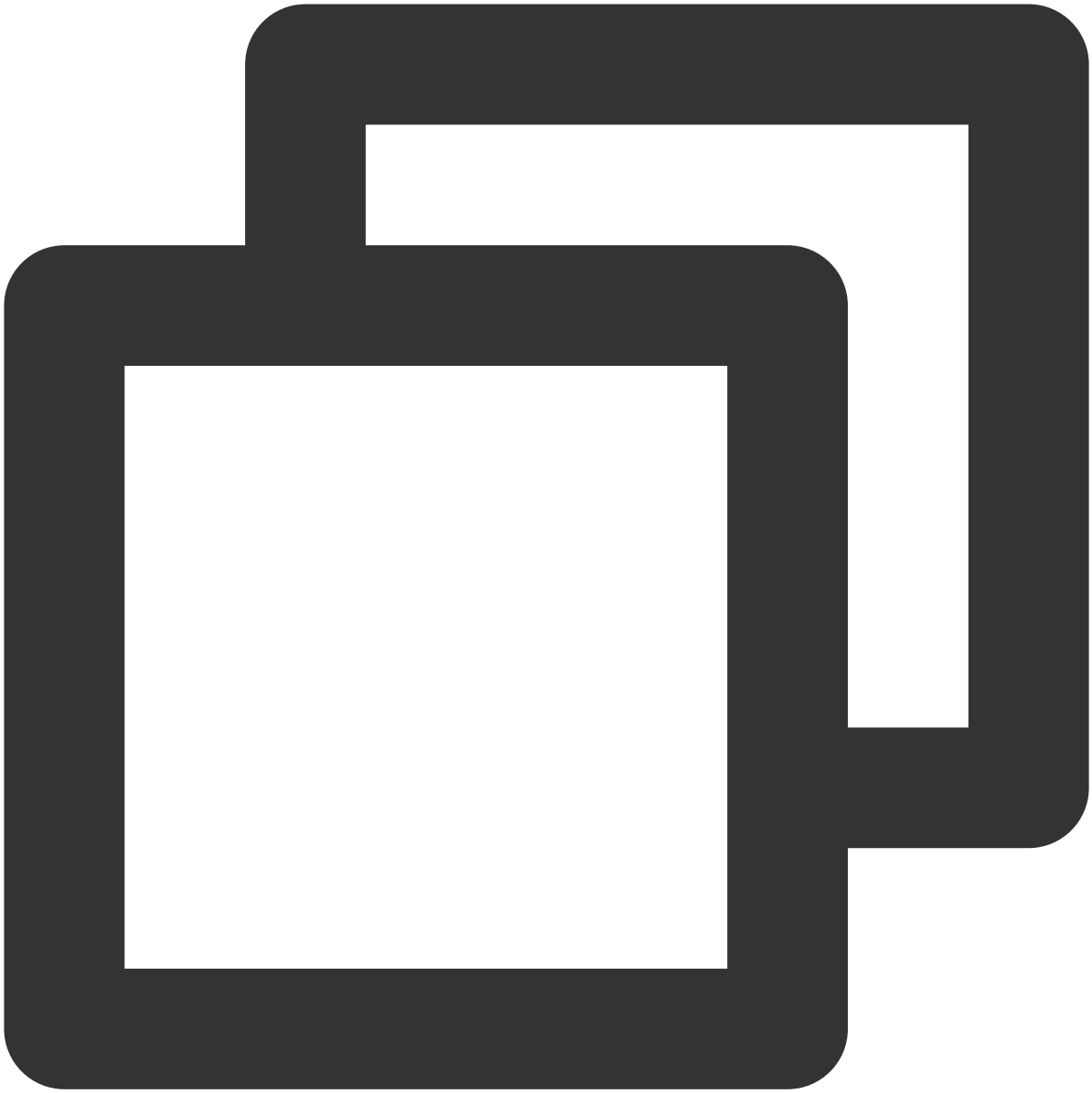
请求包示例如下：



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

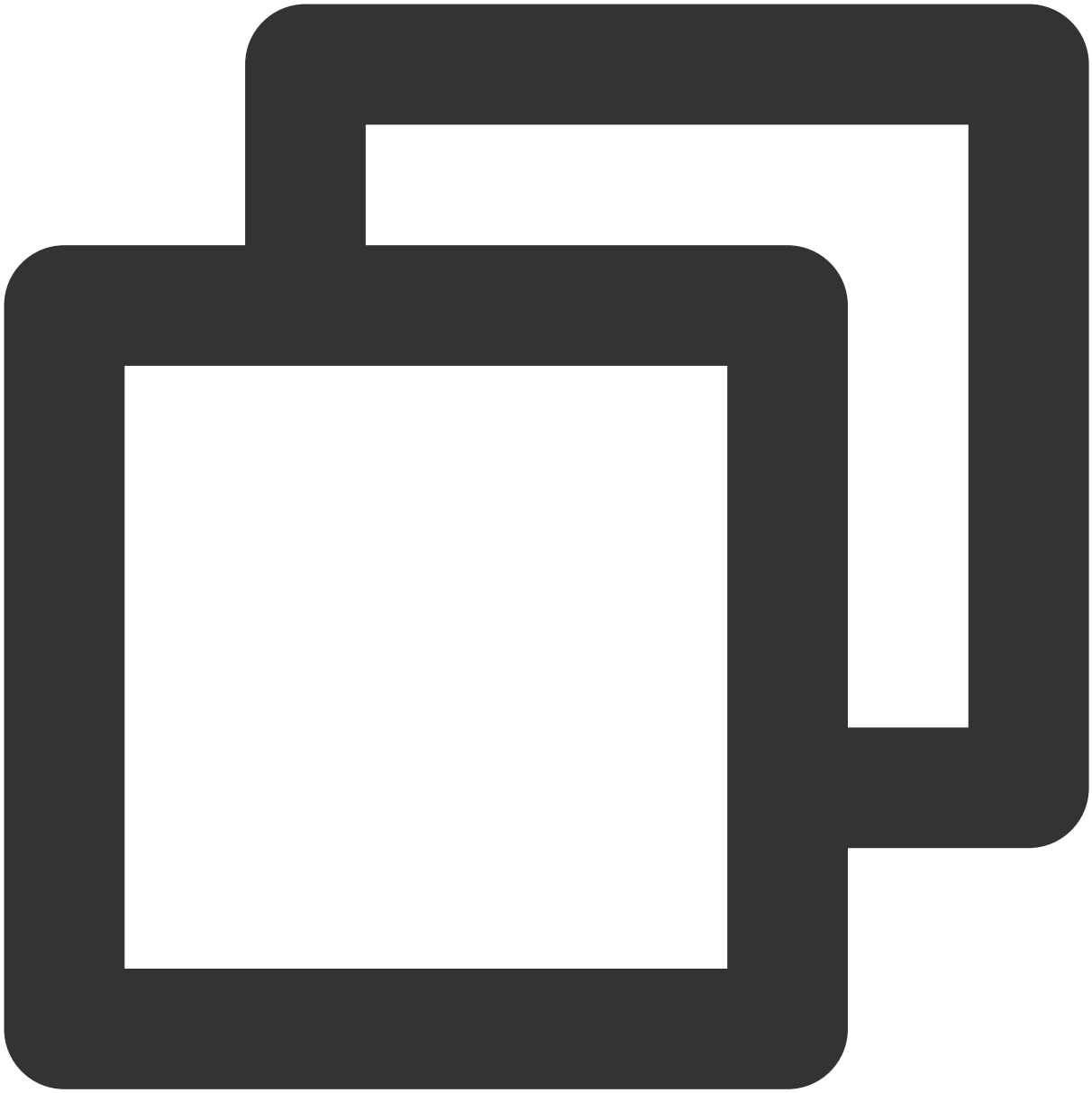
更多相关内容请参见 群组满员之后回调。

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&content
```

请求包示例如下：



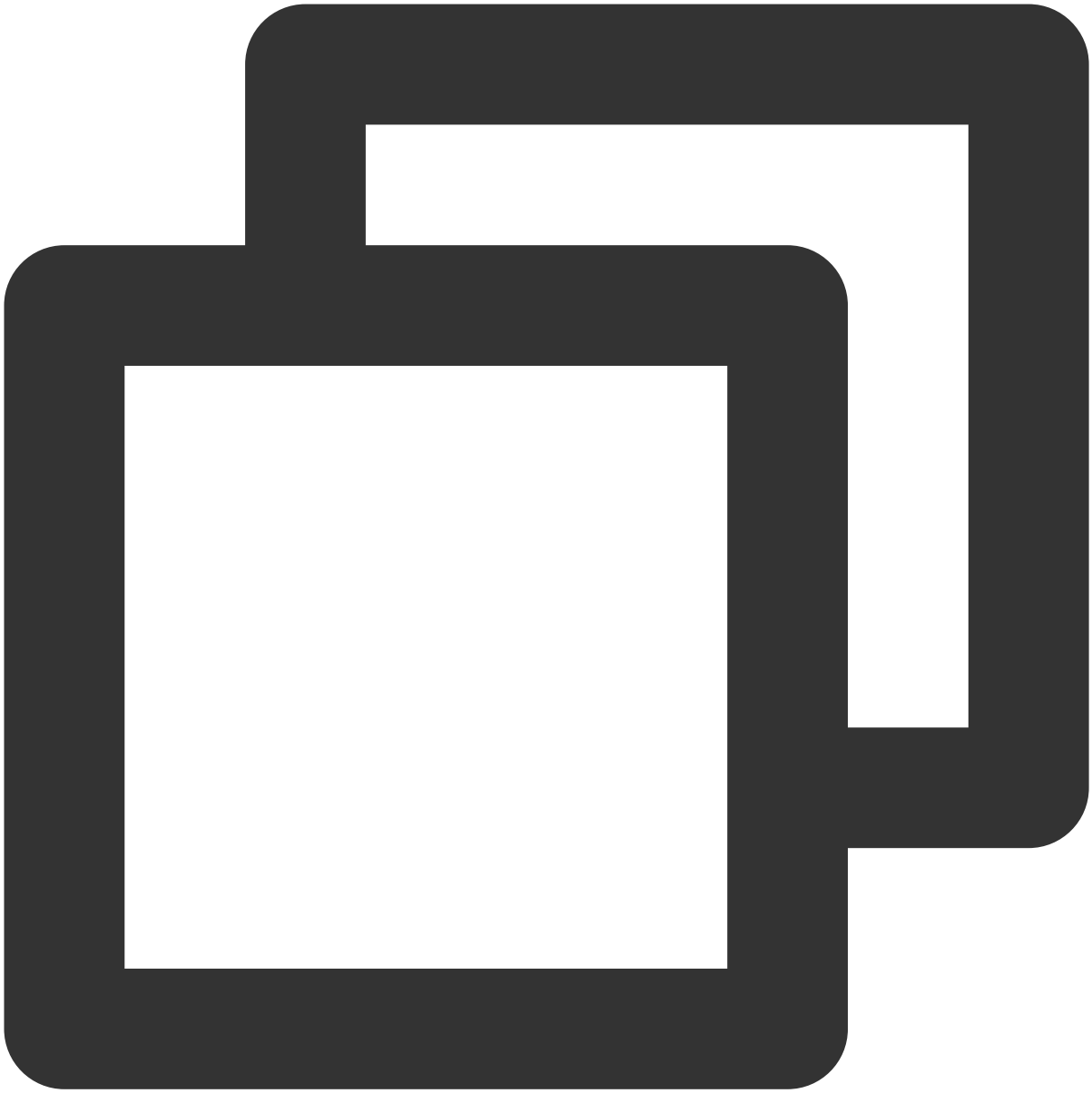
```
{
  "CallbackCommand": "Group.CallbackAfterNewMemberJoin", // 回调命令
  "GroupId" : "@TGS#2J4SZEAEL",
  "Type": "Public", // 群组类型
  "JoinType": "Apply", // 入群方式：Apply（申请入群）；Invited（邀请入群）
  "Operator_Account": "leckie", // 操作者成员
  "NewMemberList": [ // 新入群成员列表
      {
          "Member_Account": "jared"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

```
            "Member_Account": "tommy
        }
    ]
}
```

说明：

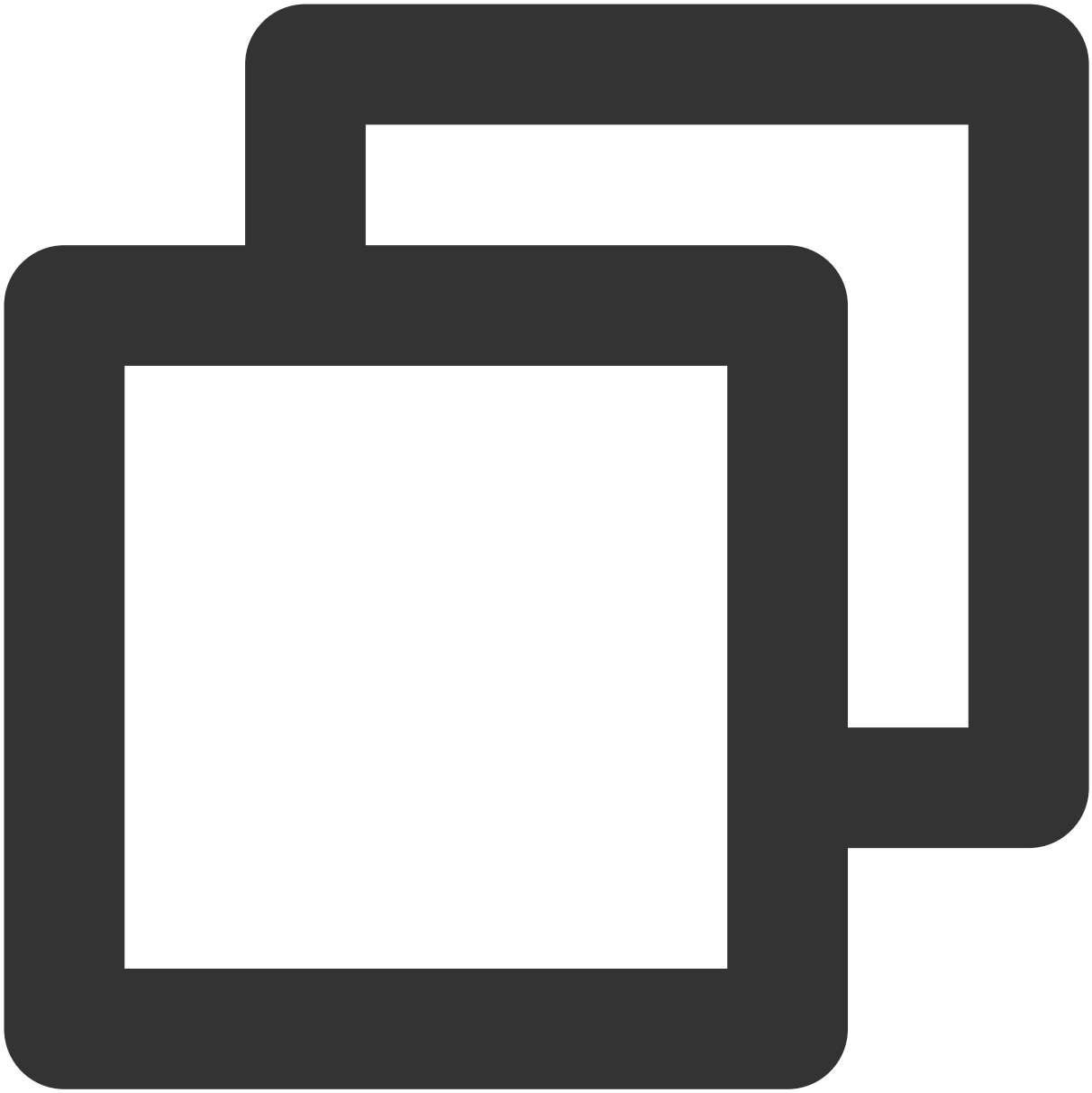
要启用回调，必须配置回调URL，并打开本条回调协议对应的开关。配置方法参见 第三方回调配置 文件。更多相关内容请参见 新成员入群之后回调。

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&content
```

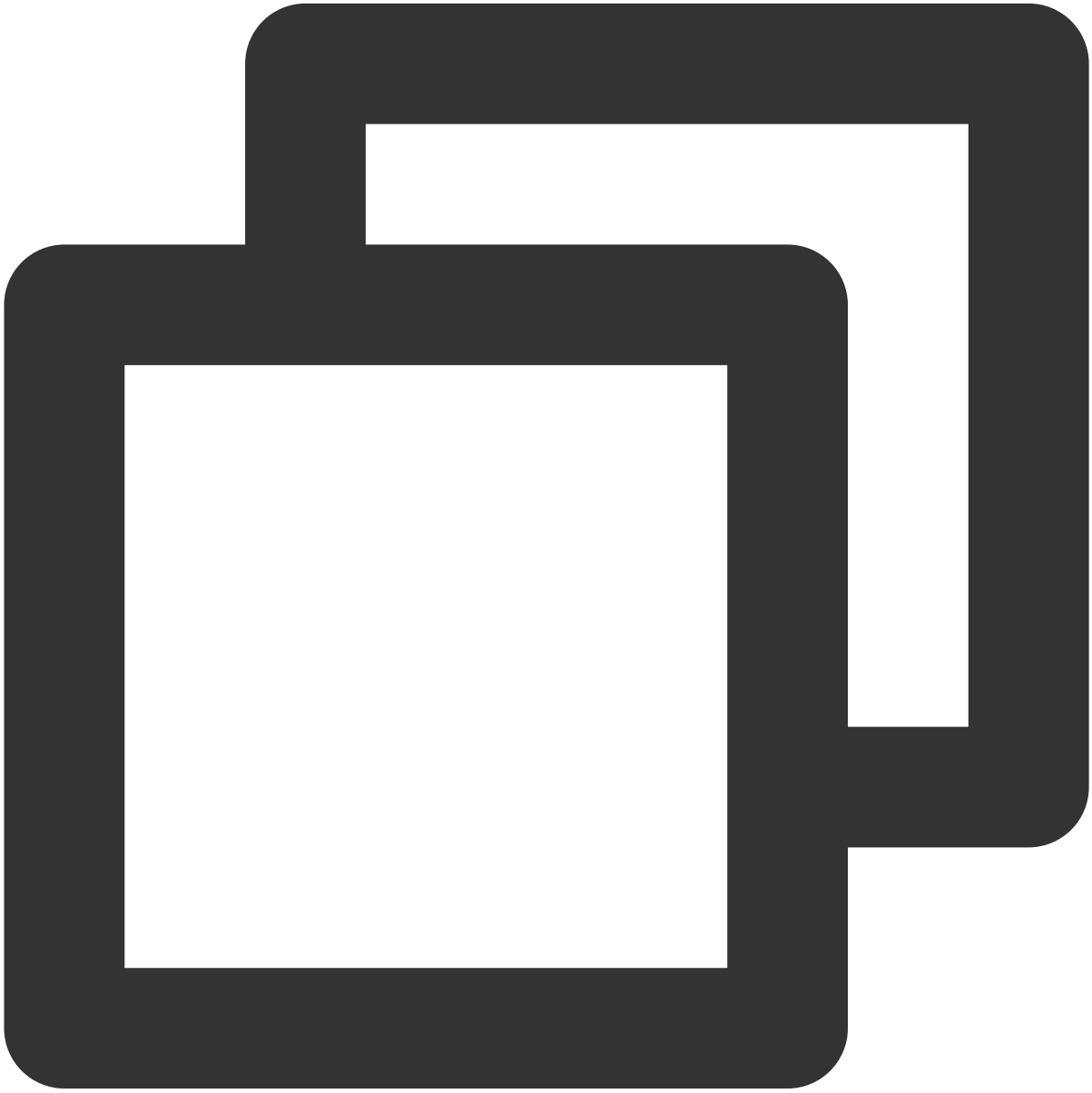
请求包示例如下：



```
{
  "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"
      },
      {
```

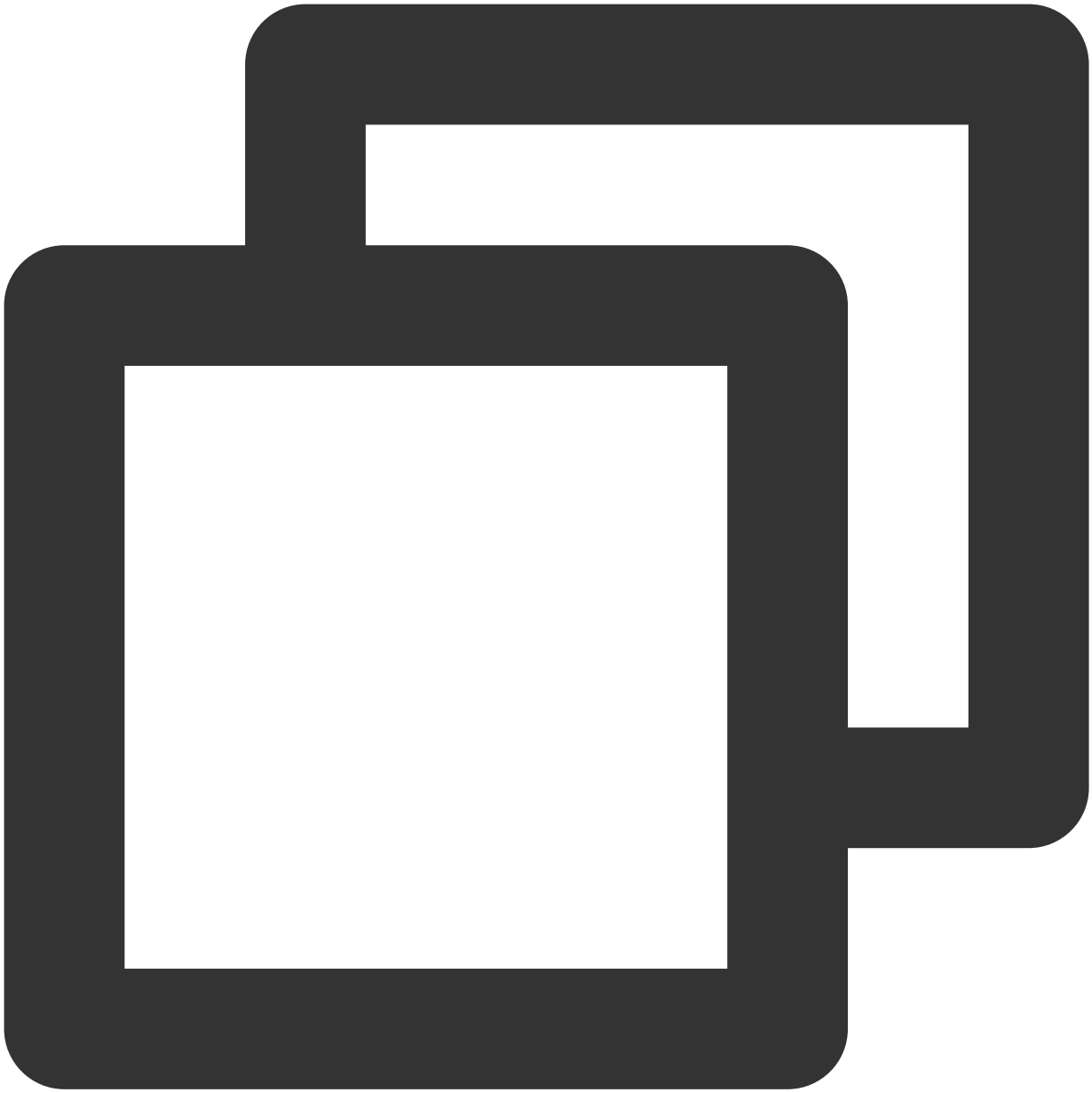
```
            "Member_Account": "tommy"
        }
    ]
}
```

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&i
```

请求包示例如下：

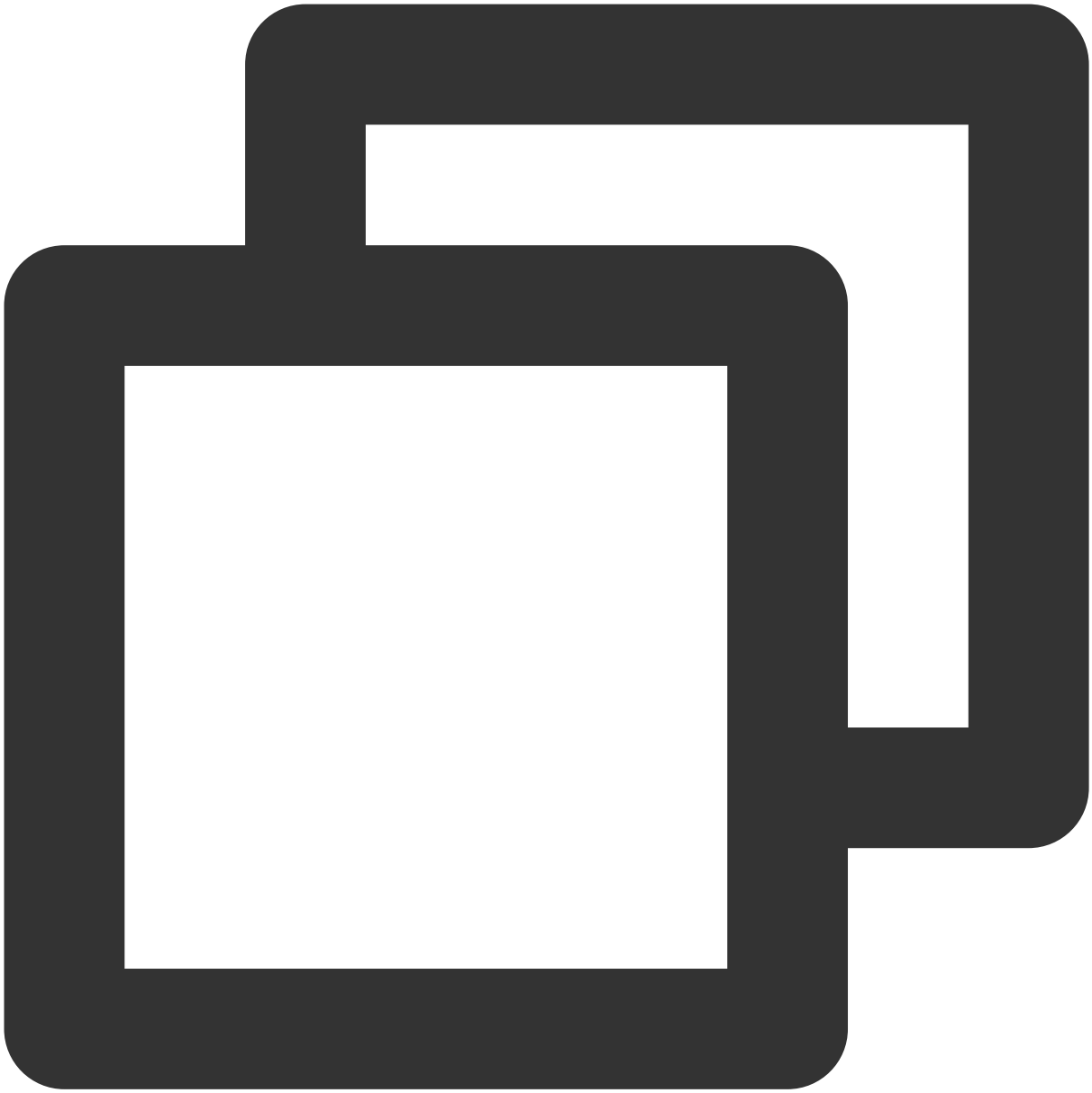


```
{
  "GroupId": "@TGS#2J4SZEAEL"
}
```

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。

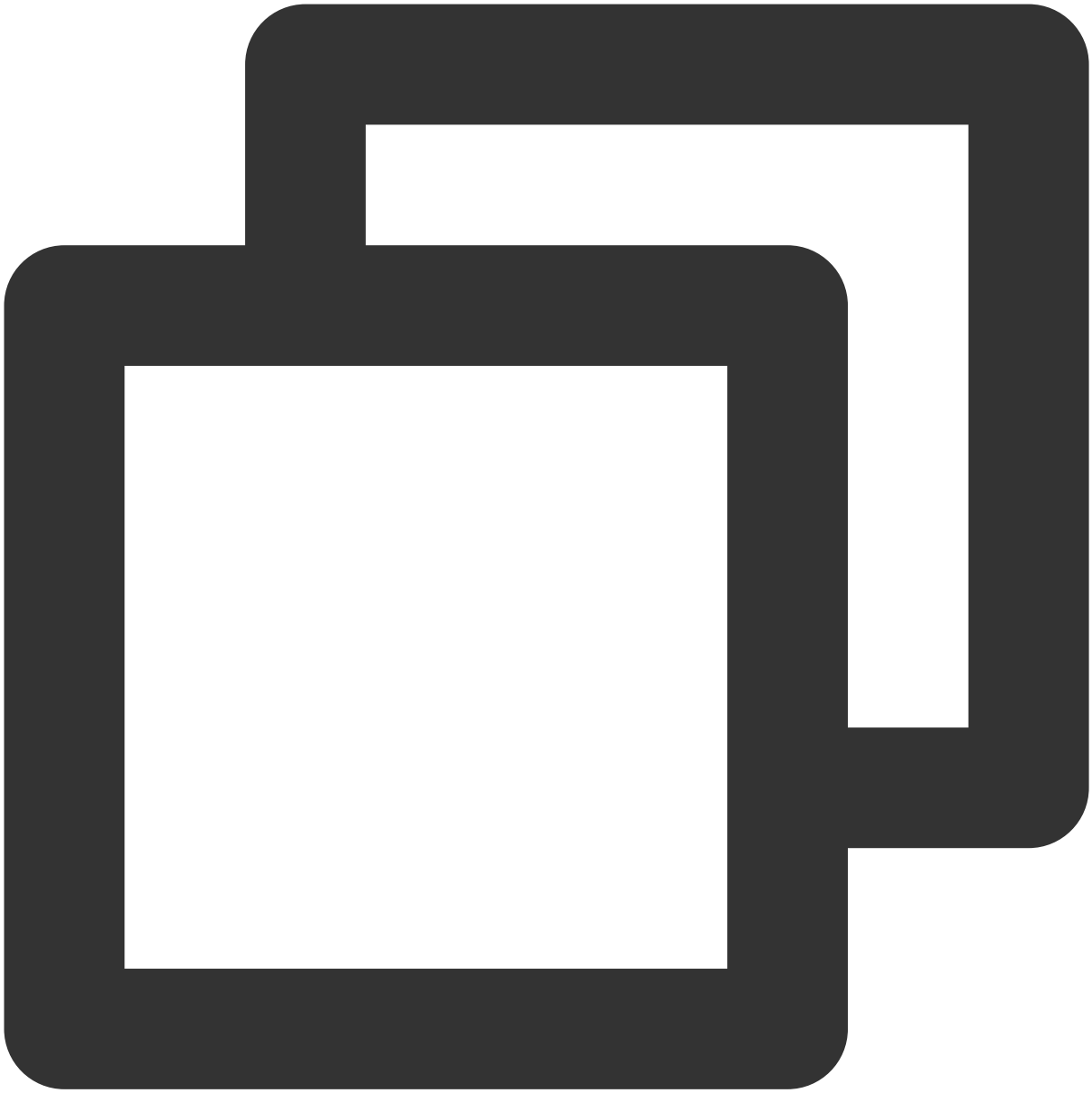
后台获取队内情况

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&
```

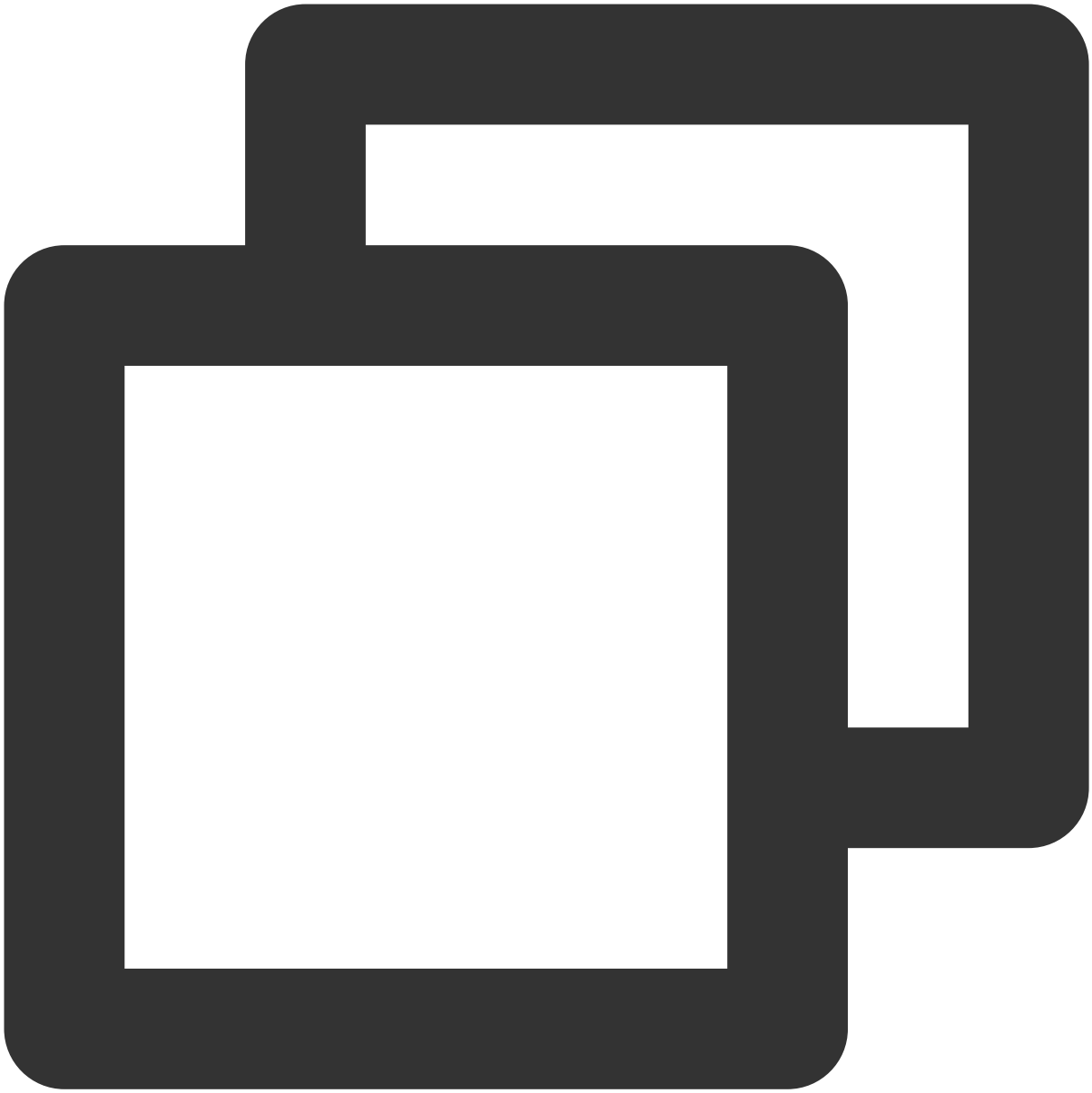
请求包示例如下：



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

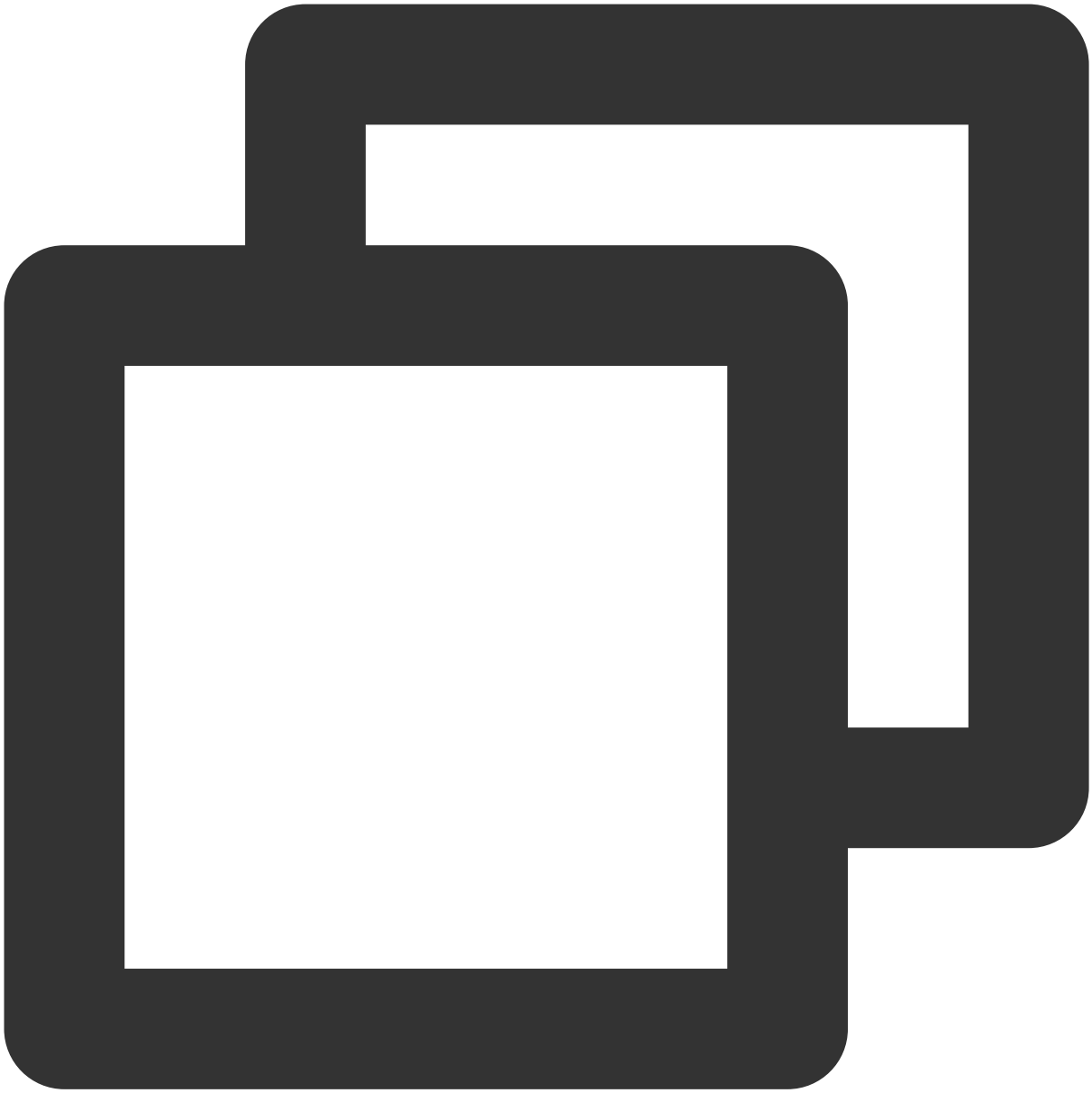
更多详细信息请参见 获取群详细资料。

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_info?sdkappid=88
```

请求包示例如下：



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

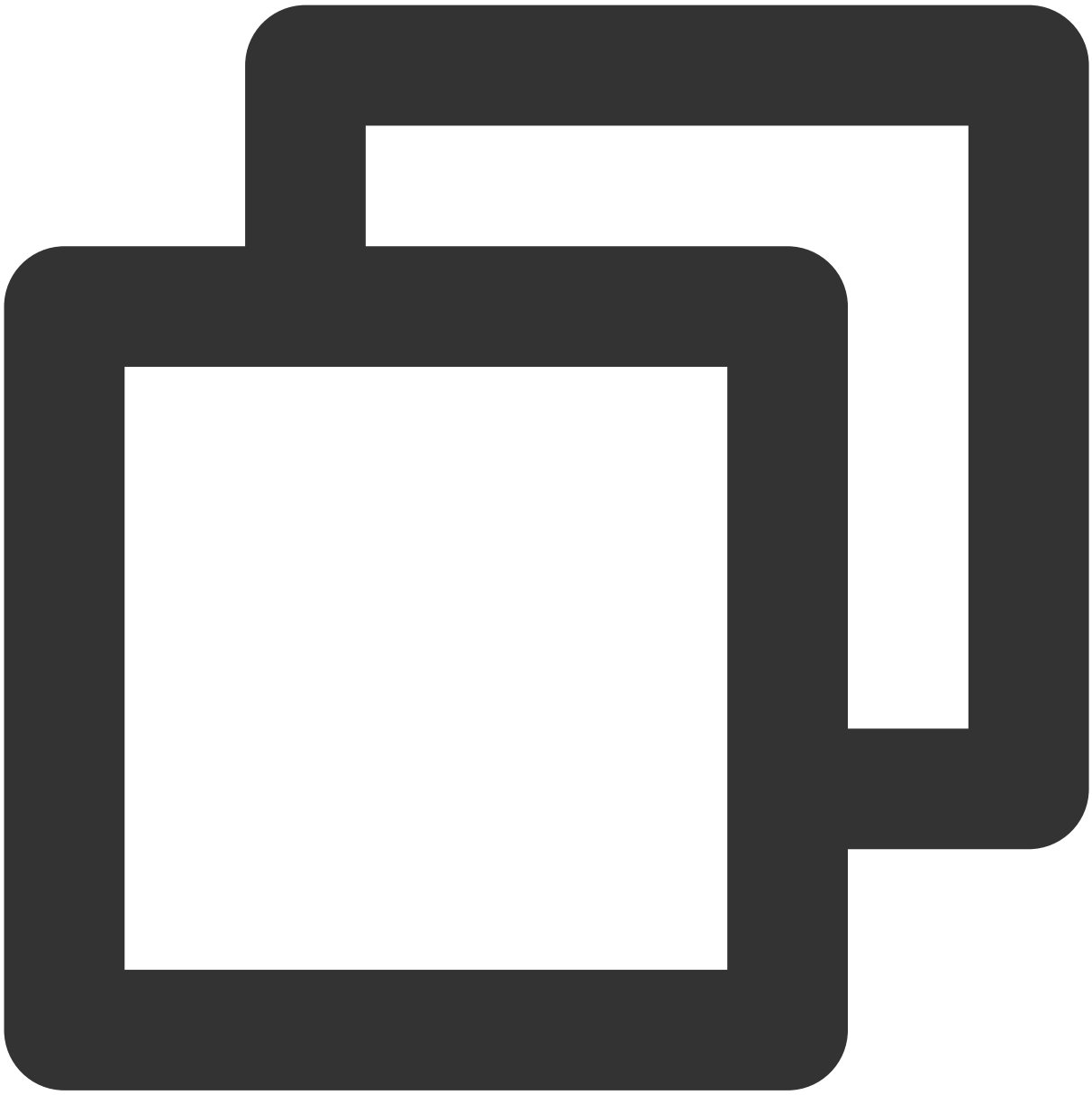
返回字段中 `MemberNum` 可获取本群组的群成员总数，`AppMemberDefinedData` 可获取群成员自定义字段信息。

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,
 // 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。

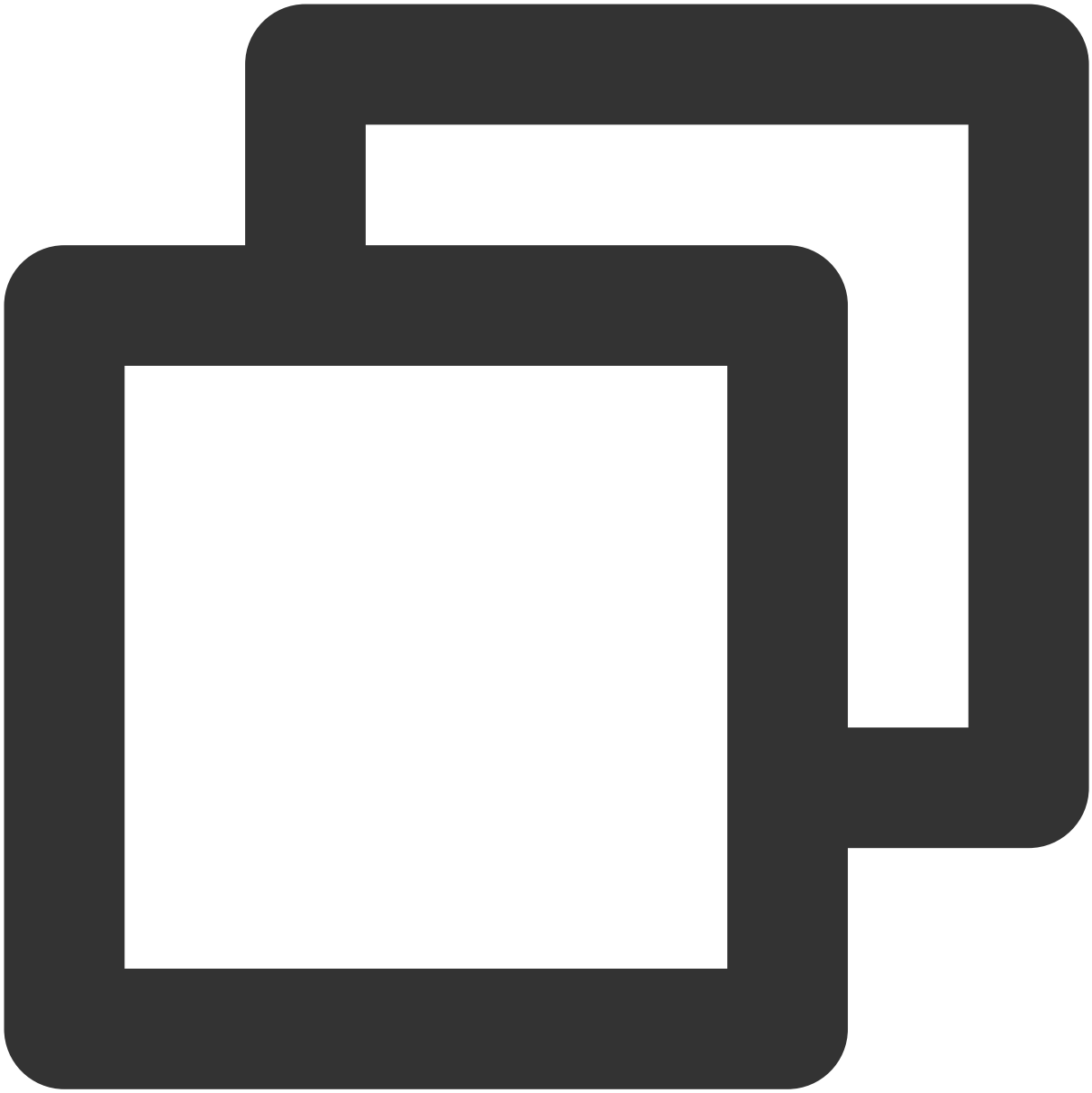
更多有关群组选择和关于群组的其他事项，请参见 群组系统 。更多有关**群组管理的控制台指南**，请参见 群组管理控制台指南。

敏感信息过滤

过滤敏感内容也是游戏即时通信中非常重要的功能，实现方案如下：

1. 绑定用户群发消息前回调
2. 判断通过回调数据判断消息类型，将消息数据传递给天御或者其他第三方检测服务
3. 如消息为普通文本消息，可等天御检测同步返回，再将是否下发消息的数据包返回给 IM 后台

发送消息前回调数据示例：



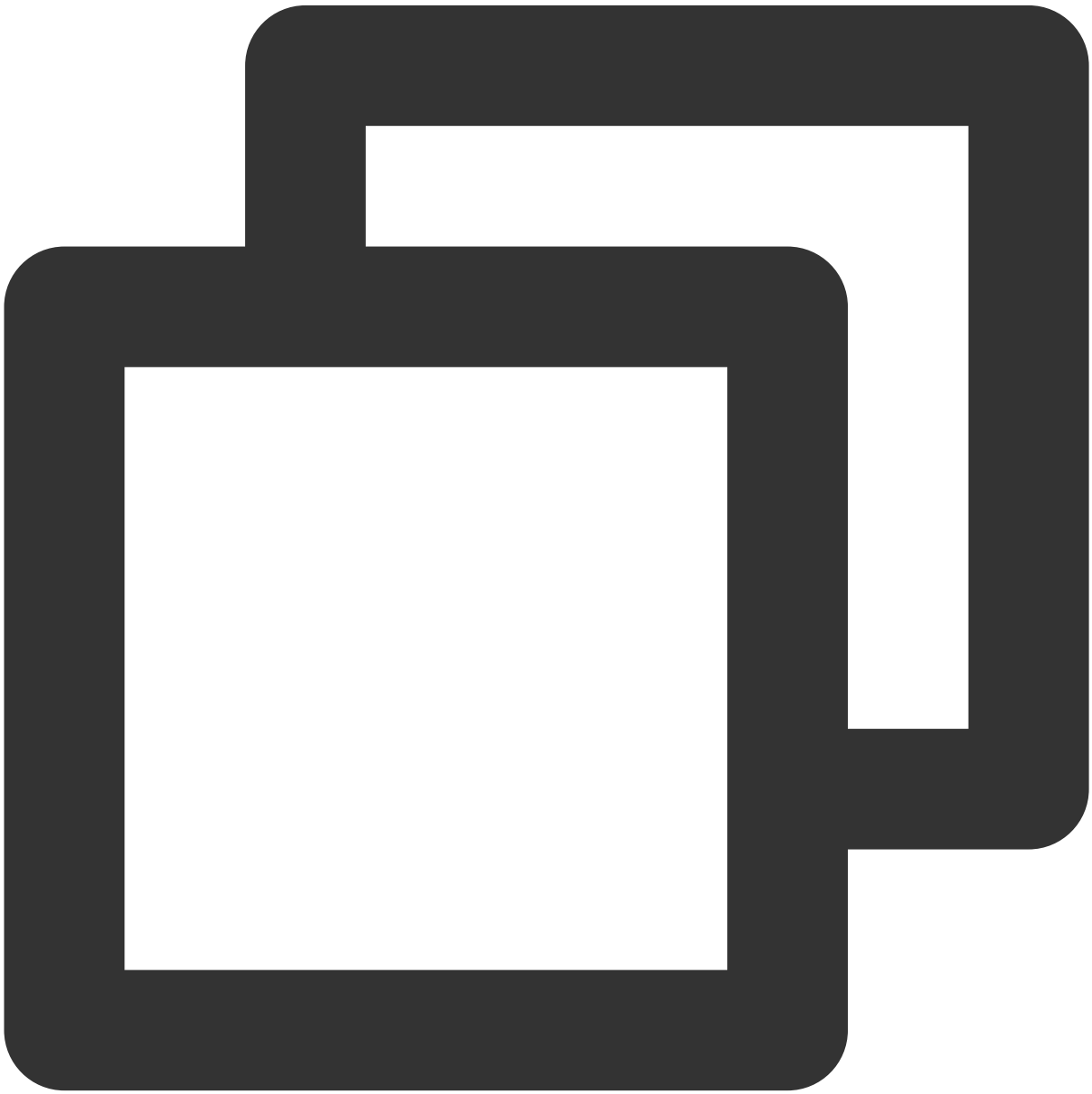
```
{
    "CallbackCommand": "Group.CallbackBeforeSendMsg", // 回调命令
    "GroupId": "@TGS#2J4SZEAEL", // 群组 ID
    "Type": "Public", // 群组类型
    "From_Account": "jared", // 发送者
    "Operator_Account":"admin", // 请求的发起者
    "Random": 123456, // 随机数
    "OnlineOnlyFlag": 1, //在线消息，为1，否则为0；直播群忽略此属性，为默认值0。
    "MsgBody": [ // 消息体，参见 TIMMessage 消息对象
        {
            "MsgType": "TIMTextElem", // 文本
```

```
            "MsgContent": {
                "Text": "red packet"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data"
}
```

说明：

开发者可通过 MsgBody 中 MsgType 字段判断消息类型。完整字段说明可参见 回调文档。

开发者可以选择对不合法的消息进行不一样的处理，可在回调中回包给 IM 后台来控制。



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // 不同的ErrorCode有不同的含义
}
```

ErrorCode	含义
0	允许发言，正常下发消息
1	拒绝发言，客户端返回10016
2	静默丢弃，客户端返回正常

说明：

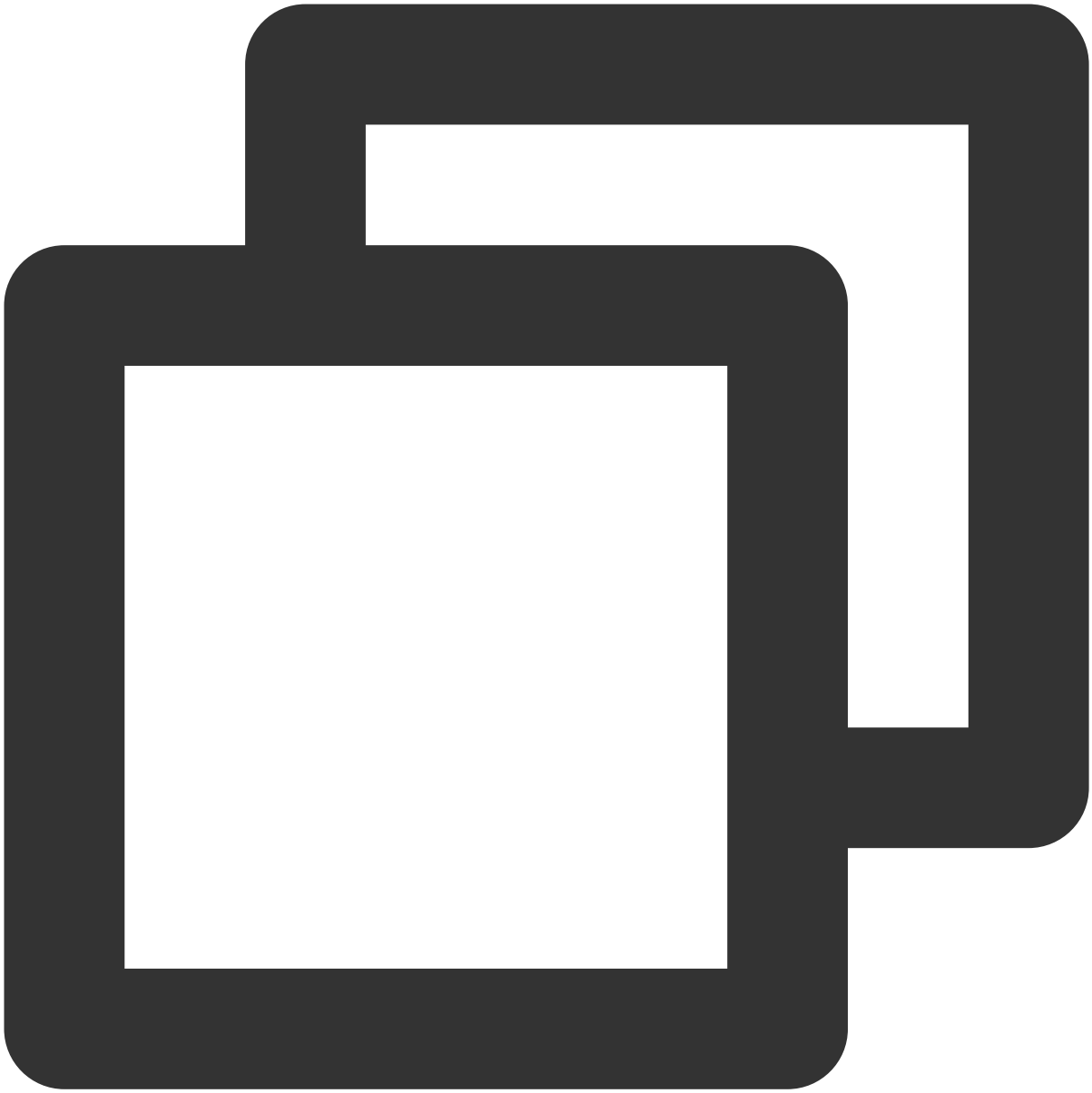
开发者可根据自身业务进行选择使用。

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 提供给予即时通信 IM 和 实施音视频品视频 TRTC 音视频通话功能（TUICallKit），为 IM 聊天提供实施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，我们面为为每个 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）	创建后仅支持已在群内的好友申请加群，无需被邀请访同意或群组审批。
陌生人社交群（Public）	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	社群（Community）	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。