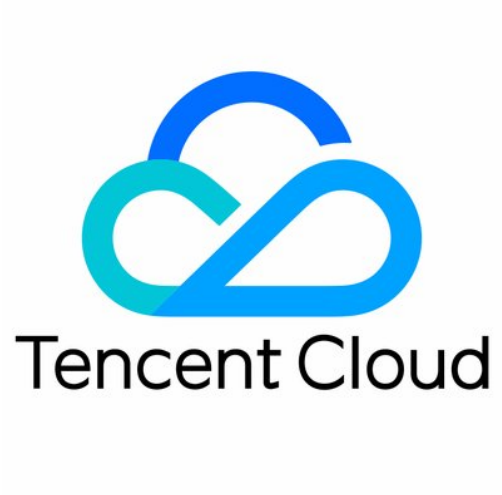
Chat

シナリオプラン

製品ドキュメント



カタログ：

シナリオプラン
Live Room
ライブストリーミングルーム構築ガイド

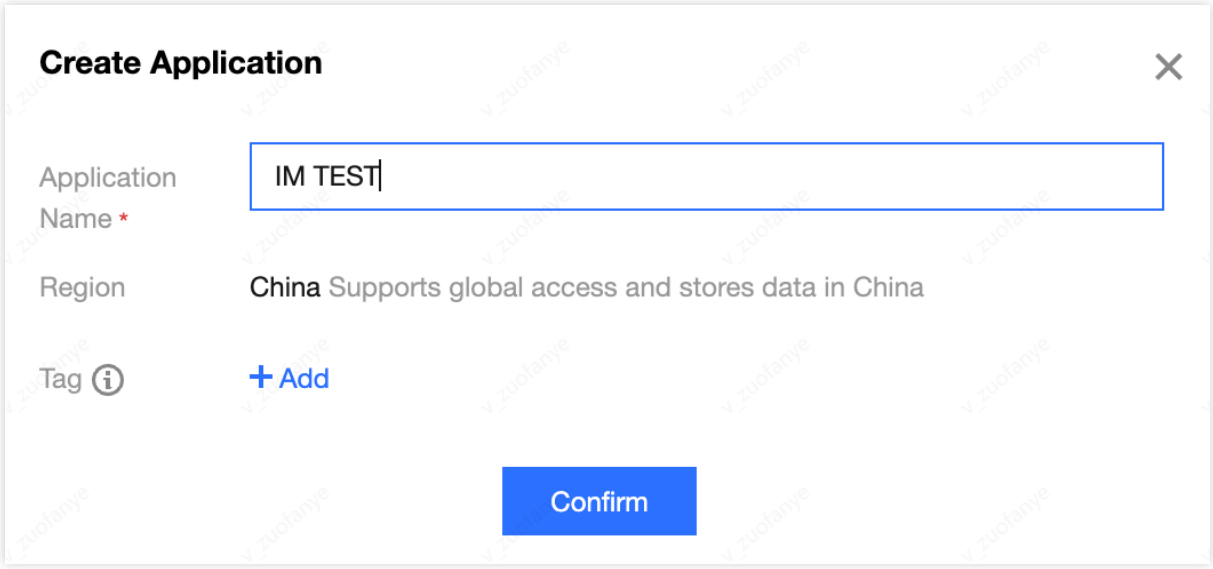
最終更新日：：2024-04-11 17:29:00

ライブストリーミングは生活のいたるところに存在すると言ってよく、自身のライブストリーミングプラットフォームを持つ企業や開発者はますます増えています。ライブストリーミングプラットフォーム自体が関与する、ライブストリームのプッシュ・プル、ライブストリーミングチャットルーム、ライブルームのインタラクション（「いいね」、ギフト、マイク接続）、ライブルームのステータス管理などに対するニーズは複雑であることが多いです。そのため、ここでは、Tencent Cloudの関連製品（Tencent Cloud IM、Tencent Cloud TRTC、Tencent Cloud CSSなど）を基本として、ライブルーム構築の過程でよくあるニーズの実現方法や、発生する可能性がある問題、注意が必要な点などについて整理します。開発者の皆さんがスピーディーに業務を理解し、ニーズを実現する上でお役に立てれば幸いです。

1、準備作業

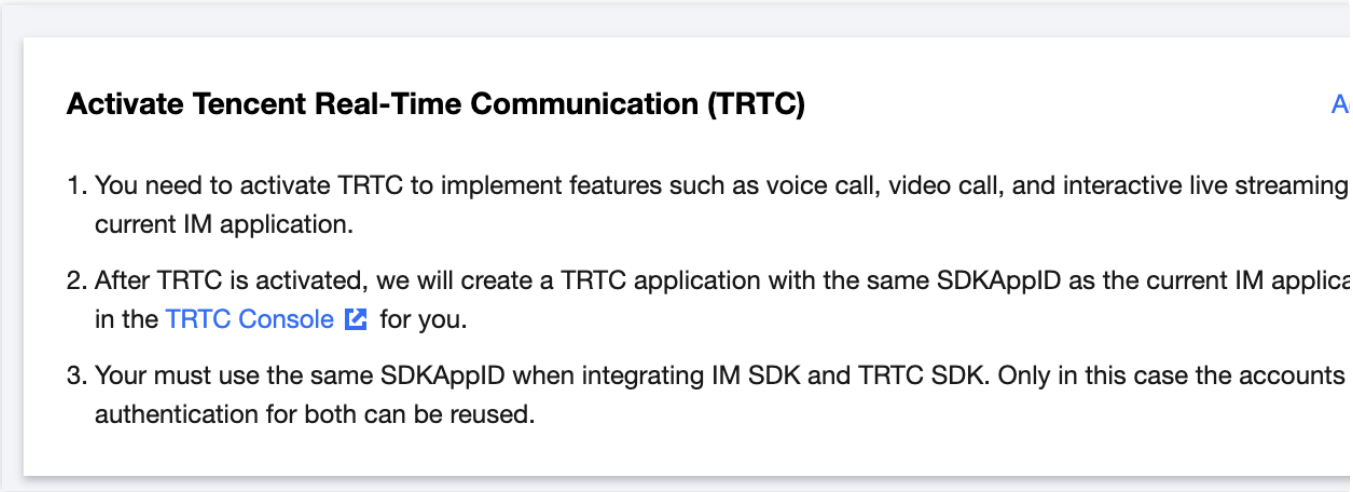
アプリケーションの作成

Tencent Cloudを使用してライブルームを構築するには、下図に示すように、まずコンソールでIMアプリケーションを作成する必要があります。

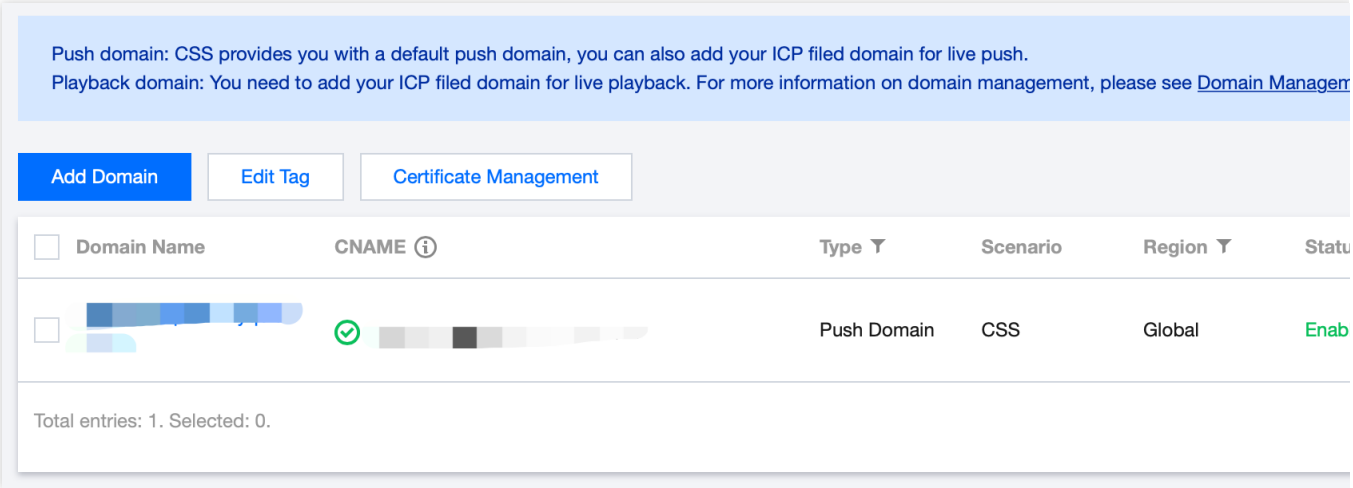


TRTCまたはライブストリーミングアプリケーションの作成

ライブルームの機能はIMの機能に依存するほか、ライブストリーミング機能にも依存しています。ライブストリーミング機能はTencent Cloud TRTCまたはCSSを使用して実装できます。下図に示すように、TRTCアプリケーションはIMアプリケーションの作成後に有効にできます。



CSSを使用したい場合も、下図のようにアクティブ化したプッシュおよび再生ドメイン名を使用することができま：

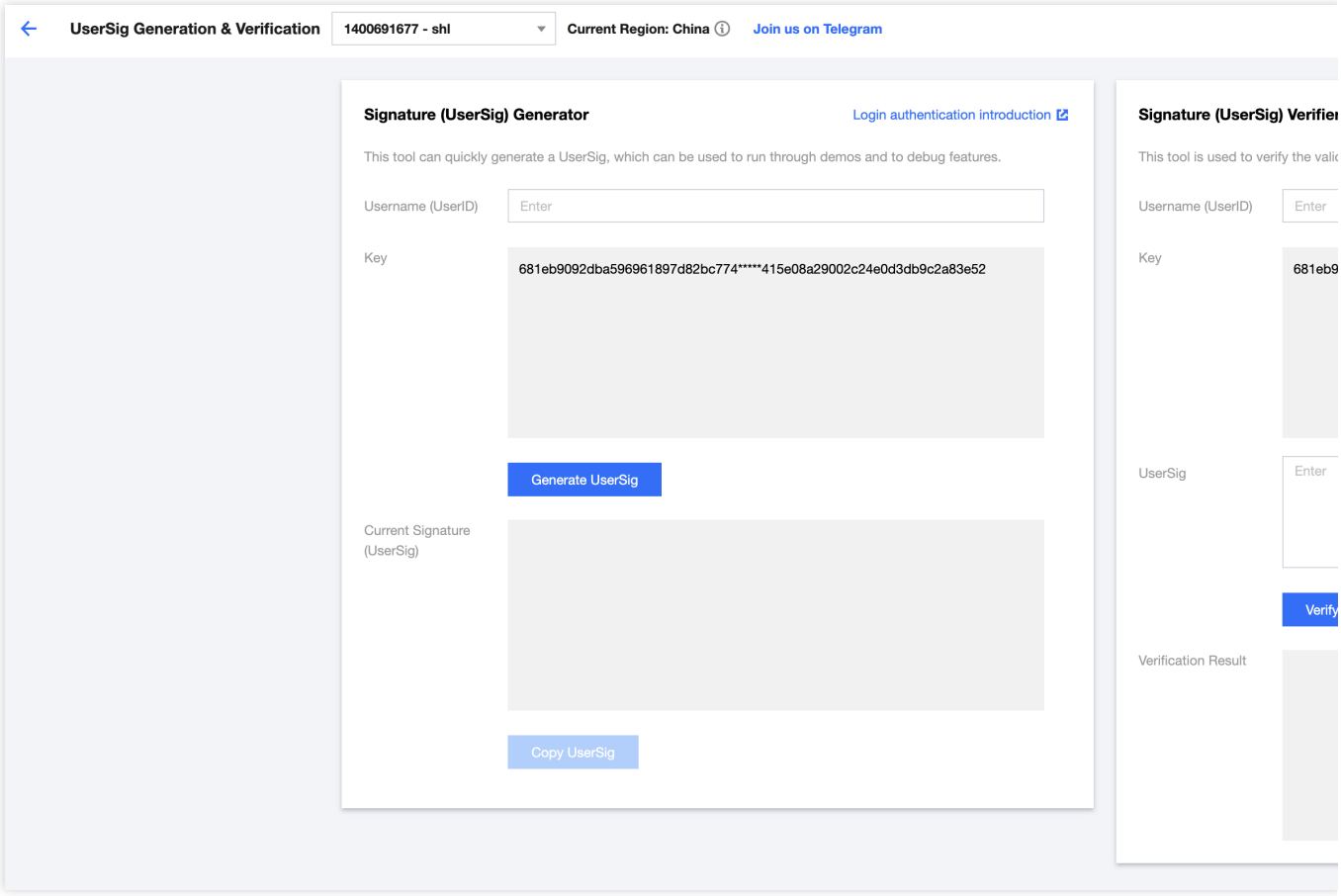


関連基本設定の完了

準備作業のステップ1で作成したアプリケーションは体験版であり、開発段階での使用にのみ適しています。本番環境のユーザーはご自身の業務ニーズに合わせて、プロフェッショナル版またはフラッグシップ版をアクティブ化してください。バージョン間の違いについてはIM価格ドキュメントをご参照ください。
ライブストリーミングのシーンでは、アプリケーションの作成以外にもいくつかの追加設定が必要です：

キーを使用したUserSigの計算

IMのアカウントシステムでは、ユーザーログインに必要なパスワードはユーザーのサーバーでIMが提供するキーを使用して計算されます。ユーザーはドキュメントUserSigの計算をご参照ください。下図に示すように、開発段階では、クライアントでの開発に支障がないよう、コンソールでのUserSigの計算を行うこともできます。

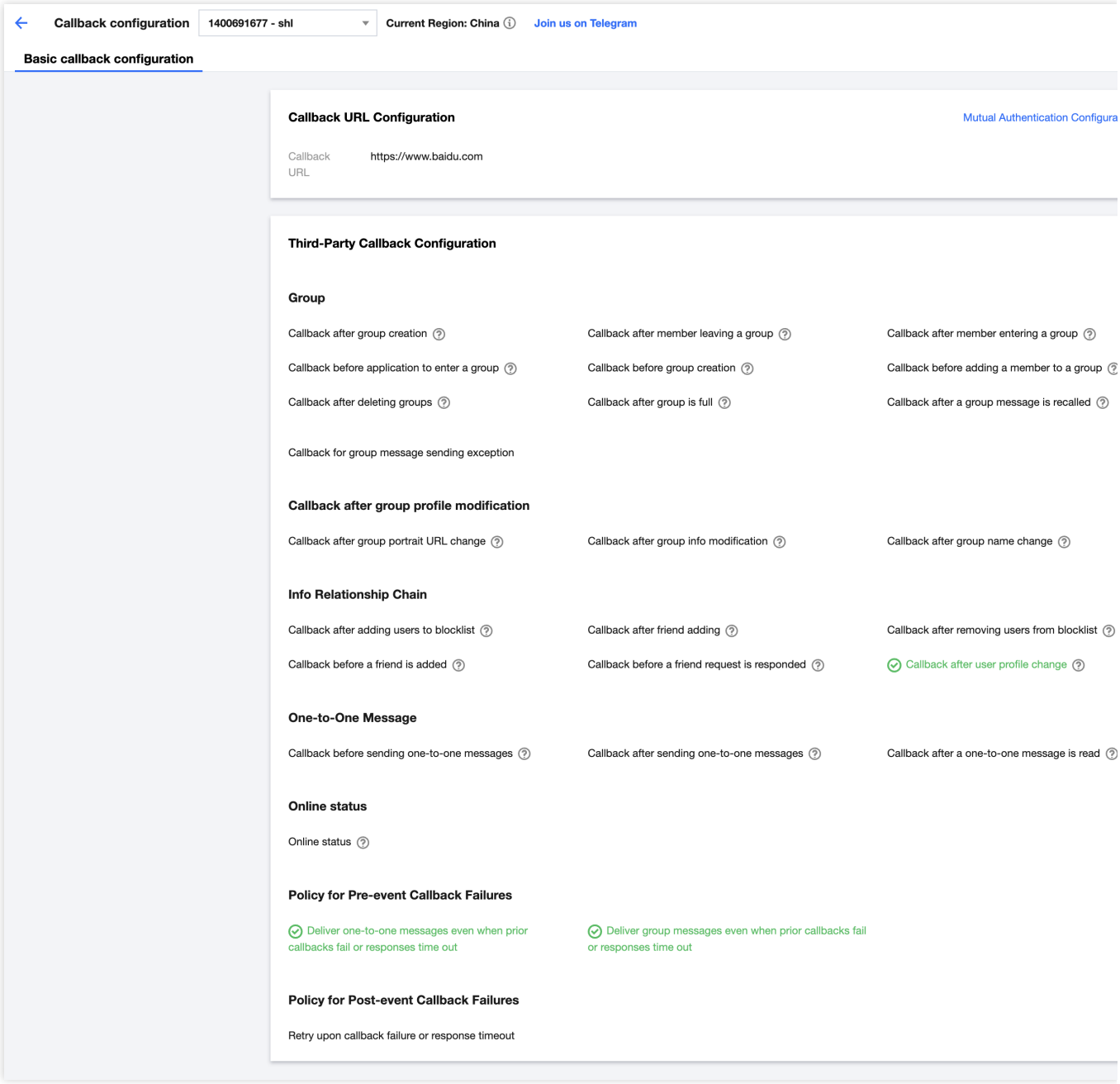


管理者アカウントの設定

ライブストリーミング中に、管理者がライブルームに対してメッセージの送信、違反ユーザーのミュート（強制退室）などを行う必要がある場合があります。この場合はIMサーバーAPIを使用して必要な処理を行わなければなりません。サーバーAPIを呼び出す前に、IM管理者アカウントの作成を行っておく必要があります。IMはデフォルトで1つのUserIDをadministratorアカウントとして開発者にご提供しています。開発者は業務のシーンに応じて複数の管理者アカウントを作成することもできますが、IMで作成できる管理者アカウントは最大5つまでであることにご注意ください。

コールバックアドレスの設定およびコールバックの有効化

ライブルームで弾幕抽選、メッセージ統計、センシティブコンテンツチェックなどのニーズを実現したい場合は、IMのコールバックモジュールを使用する必要があります。これはIMバックエンドがいくつかの特定のシーンで、開発者の業務バックエンドにコールバックを行うものです。開発者は下図のように、HTTPのインターフェースを1つ提供し、コンソール > コールバック設定モジュールに設定するだけで済みます。



クライアントSDKの統合

準備が完了したら、IMとTRTCのクライアントSDKをユーザープロジェクトに統合する必要があります。開発者は、業務ニーズに応じてさまざまな統合ソリューションを選択できます。
続いて、ライブルームによくある機能のポイントについて整理し、開発者のご参照用としてベストプラクティスソリューションを提供します。関連の実装コードも併せてご参照ください。

2、ライブルームの各機能の開発ガイド

ライブルームのステータス

ライブルームのステータスには一般的に次の数種類があります：

番号	ライブルームのステータス
1	ライブストリーミング開始待機中
2	ライブストリーミング中
3	ライブストリーミング一時停止
4	ライブストリーミング終了
5	ライブストリーミングリプレイ中

ライブルームのステータスには次のような特徴があります：

番号	特徴
1	ライブルームのステータス変更はライブルームのユーザーにリアルタイムで通知する必要がある
2	ライブルームの新規入室ユーザーは現在のライブルームのステータスを取得する必要がある

これらについては2つの実現方法を提供しています。この2つの方法のメリットとデメリットについて分析します。

2種類の実現方法	メリット	デメリット
業務バックエンドでライブルームのステータスを保守し、IMサーバーAPIを使用してグループカスタムメッセージ を送信してグループ内のユーザーに通知します。	ライブルームのステータスを何度も頻繁に取得する必要がある場合は、ライブルームのステータスをIMのグループプロファイルに保存する場合に比べて、業務バックエンドに保存することでIM SDKの呼び出し頻度を低下させることができます。IM SDKを統合していない場所に、取得したライブルームのステータスを提供できる可能性があります。	業務バックエンドがライブルームステータス読み取り/書き込みモジュールを追加で提供する必要があります。ライブルームデータは業務バックエンドとIMのグループプロファイルの両方から取得するため、ライブルームデータのエラー確率が上昇します。送信したカスタムメッセージが消失する可能性があります。メッセージ量が特に多い場合、優先度の低いメッセージが破棄され、それによってライブルームステータスの表示に影響することがあります。このため、ここでは高優先度のカスタムメッセージを使用することをお勧めします。

グループカスタムフィールドまたはグループ属性によってライブルームのステータスを保存し、クライアントSDKのグループ属性変更コールバックによってグループ内のユーザーに通知します。	開発者は追加のライブルームステータス読み取り/書き込みモジュールを提供する必要がありません。グループ属性変更コールバックは理論上は消失の可能性がありません。ライブルームデータはグループプロファイルから取得し、データソースが1つのため、エラーが少なくなります。	公開性の高いモジュールではグループプロファイルを頻繁に取得する必要があり、IMの負荷が増大します。IM SDKモジュールを統合していない場合は、業務バックエンドでIMサーバーSDKを呼び出してグループプロファイルを取得する必要があり、呼び出し頻度にも制限があります。

上記の分析の結果、方法1と方法2を組み合わせて使用し、ライブルームのステータスを保守することをお勧めします：
1. ライブルーム内では、方法1によってライブルームのステータスを取得する
2. ライブルーム外では、方法2によってライブルームのステータスを取得する
3. 業務バックエンドはライブルームのステータスが変更されると、速やかにIMサーバーインターフェースを通じてデータをIMグループプロファイルに同期する（グループ属性、グループカスタムフィールド）

グループタイプの選択

ライブストリーミングのシーンでは、ユーザーチャットエリアには次のような特徴があります：
1. ユーザーのグループへの出入りが頻繁で、なおかつこれらのグループのセッション情報（未読、lastMessageなど）を管理する必要がない
2. ユーザーは自動的にグループに参加し、監査の必要がない
3. ユーザーの発言は一時的であり、グループの過去のチャット記録をフォローしていない
4. グループの人数は一般的に比較的多い
5. グループメンバー情報を保存しなくてもよい
そのため、IMのグループ特性に基づいて、ここではAVChatRoomを選択してライブルームのグループタイプとします。
IMライブストリーミンググループ（AVChatRoom）には、次の特徴があります：
人数制限なし、数千万人のインタラクティブライブストリーミングのシナリオを実現。
全オンラインユーザーへのプッシュメッセージ（グループシステム通知）をサポート。
グループへの参加申請後、管理者の承認を受けずに、直接参加が可能。
説明：
IM Web SDKでは、同一のユーザーは同じ時間に1つのAVChatRoomにしか入れないという制限があります。IMの複数端末ログインのシーンでは、ユーザーのログイン端末1がライブルームAでライブストリーミングを視聴している場合、コンソールの設定で複数端末ログインを許可する状態で、このユーザーのログイン端末2がライブルームBに入って視聴した場合、その時点でライブルームA内の端末1はグループから退出させられます。

ライブルームのお知らせ

ライブルームのお知らせ（テーマ）は各ライブルームが必ず備えるべきコンテンツであり、ユーザーはライブルームに入室すると、そのライブルームの基本情報を見ることができます。また、ライブルームのお知らせは、変更後にリアルタイムでライブストリーミンググループのグループメンバーにも通知する必要があります。グループライブストリーミングのステータスと同じように、ライブストリーミングのステータスもお客様の業務バックエンドまたはIMのグループプロファイルに保存することができます。ライブストリーミングのステータスと異なる点は、ライブルームのお知らせには次のような、開発者が注意すべきいくつかの追加事項があることです：

1. グループプロファイルのnotificationおよびintroductionフィールドに保存できる長さに制限があります。お客様は長い通知を送信できません（グループ概要は**240**文字まで、グループのお知らせは**300**文字までです）。
2. グループプロファイルのnotificationおよびintroductionフィールドは現在stringタイプのみサポートしています。ユーザーが画像とテキストの混在するお知らせを希望する場合は、json string形式で設定し、画像はアクセス可能なオンライン画像urlとすることができます。
3. Web端末でeditorツールを使用してお知らせを設定した場合、htmlタグが含まれる場合はnative端末での解析が行えない可能性があります。

ライブルームのお知らせはサーバーAPIまたはクライアントSDKのグループ属性設定によって設定できます。クライアントはグループ属性を取得することで現在のグループ情報を取得し、GroupListener内のOnGroupInfoChangeを監視することで、変更されたグループ属性を取得できます。

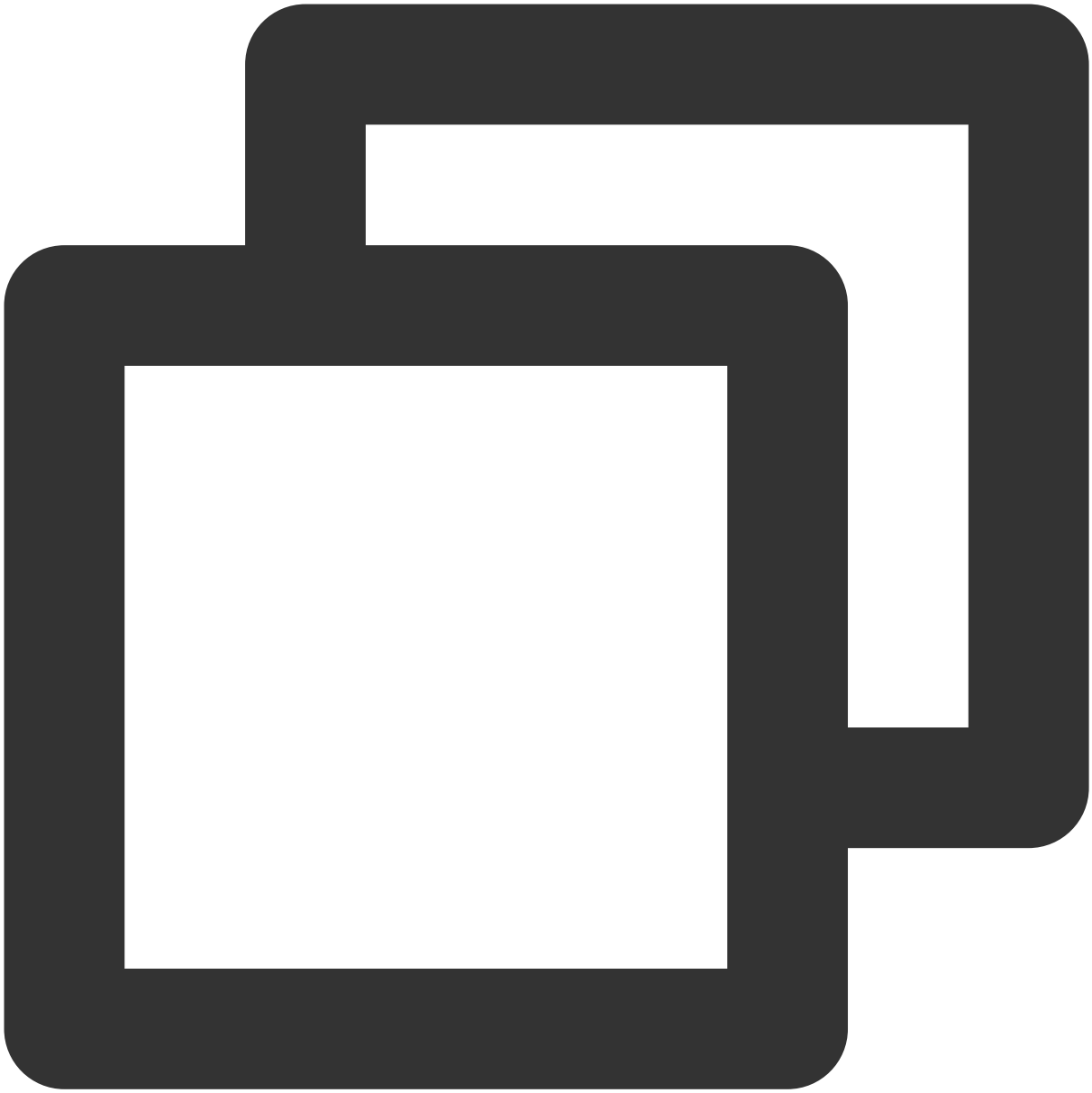
関連サンプルコード：

Android

iOS & Mac

Flutter

Web

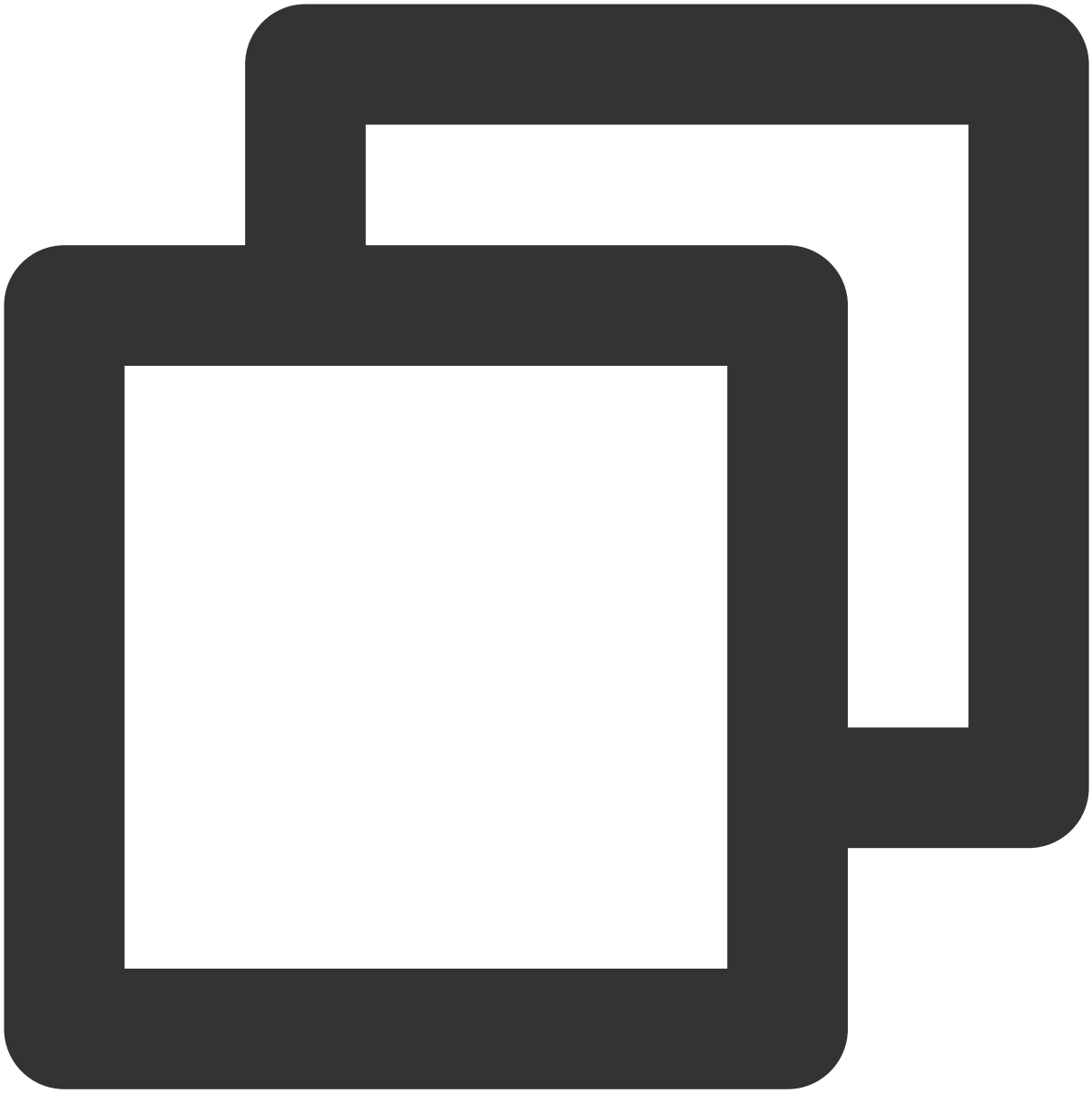


```
// クライアントがグループプロファイルを取得
V2TIMManager.getGroupManager().getGroupsInfo(groupIDList, new V2TIMValueCallback<Li
  @Override
  public void onSuccess(List<V2TIMGroupInfoResult> v2TIMGroupInfoResults) {
      // グループプロファイルの取得に成功
  }

  @Override
  public void onError(int code, String desc) {
      // グループプロファイルの取得に失敗
  }
```

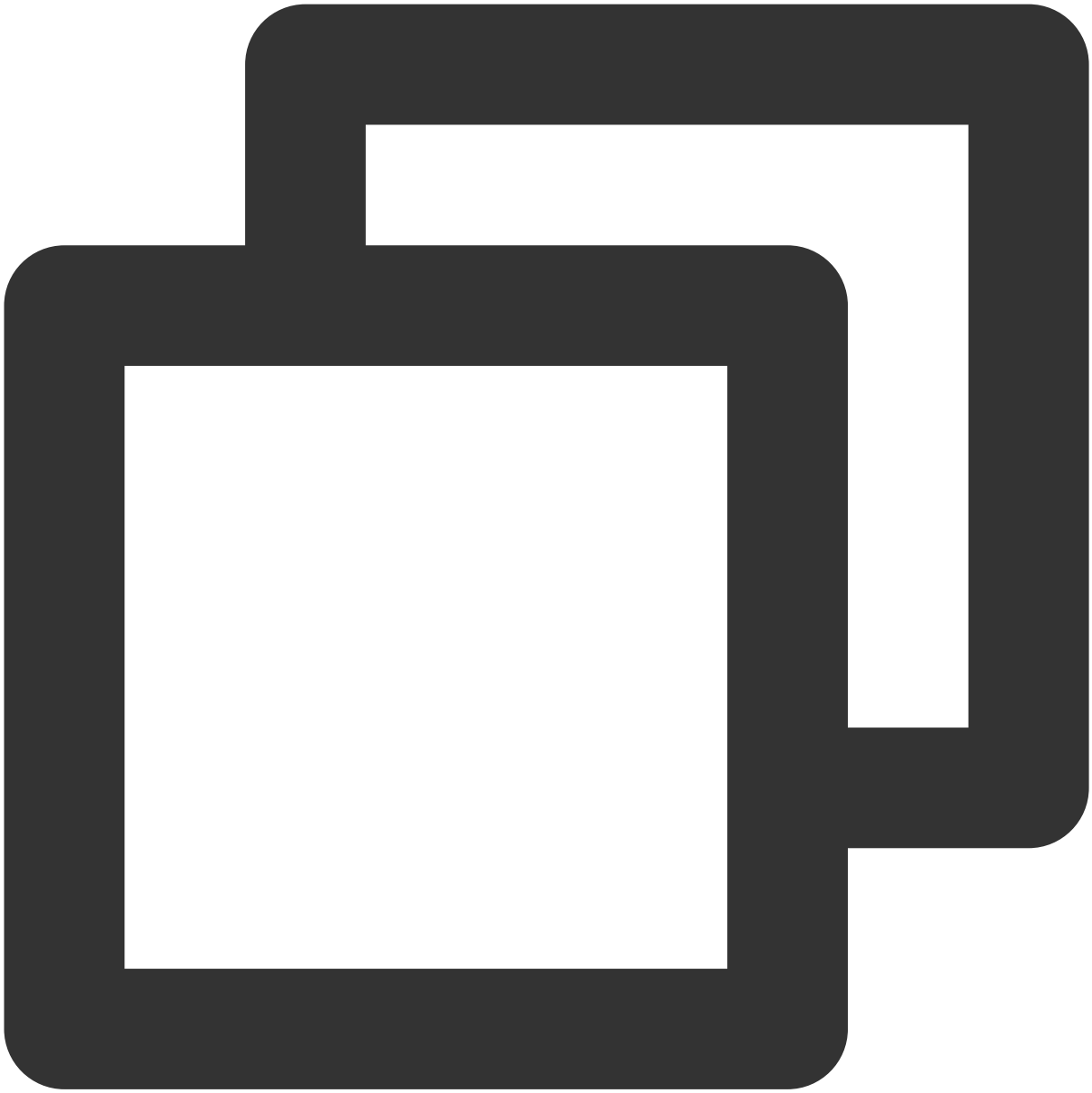
```
});
// クライアントがグループプロファイルを変更
V2TIMGroupInfo v2TIMGroupInfo = new V2TIMGroupInfo();
v2TIMGroupInfo.setGroupID("変更したいグループID");
v2TIMGroupInfo.setFaceUrl("http://xxxx");
V2TIMManager.getGroupManager().setGroupInfo(v2TIMGroupInfo, new V2TIMCallback() {
  @Override
  public void onSuccess() {
      // グループプロファイルの変更に成功
  }

  @Override
  public void onError(int code, String desc) {
      // グループプロファイルの変更に失敗
  }
});
```



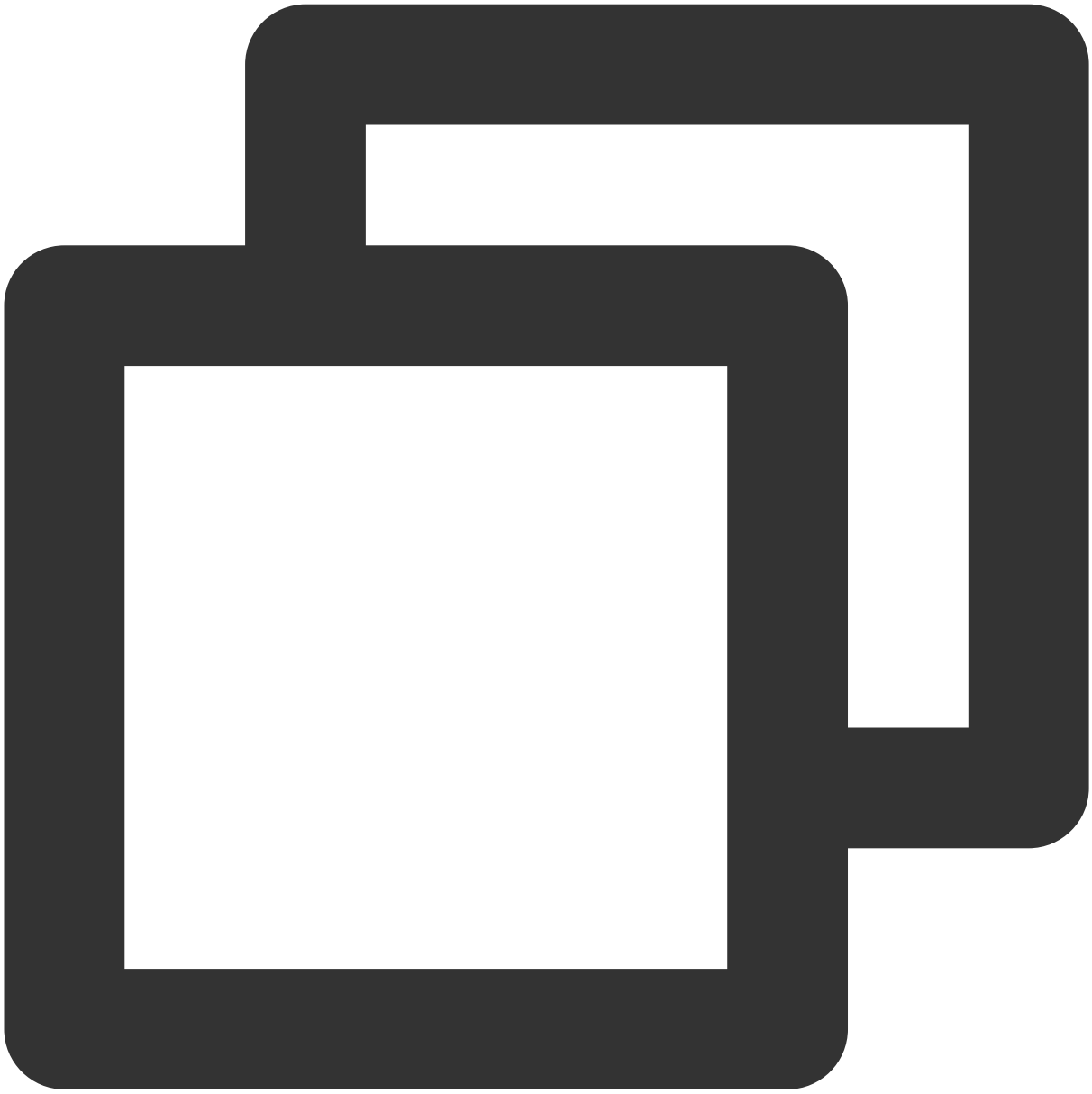
```
[[V2TIMManager sharedInstance] getGroupsInfo:@[@"groupA"] succ:^(NSArray<V2TIMGroup
    // グループプロファイルの取得に成功
} fail:^(int code, NSString *desc) {
    // グループプロファイルの取得に失敗
}];
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"変更したいグループID";
info.faceURL = @"http://xxxx";
[[V2TIMManager sharedInstance] setGroupInfo:info succ:^{
    // グループプロファイルの変更に成功
} fail:^(int code, NSString *desc) {
```

```
    // グループプロファイルの変更に失敗
}];
```



```
// グループプロファイルの取得
V2TimValueCallback<List<V2TimGroupInfoResult>> groupinfos = await groupManager.getG
// グループプロファイルの変更
groupManager.setGroupInfo(info: V2TimGroupInfo.fromJson({
    "groupAddOpt":GroupAddOptTypeEnum.V2TIM_GROUP_ADD_AUTH
    // ...その他のプロファイル
  }));
// コールバック
```

```
TencentImSDKPlugin.v2TIMManager.addGroupListener(listener: V2TimGroupListener(onGro
    // グループ情報変更コールバック
  }));
```



```
// グループプロファイルの取得
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); // 詳細なグループプロファイル取得失敗の
});
```

```
// グループプロファイルの変更
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  name: 'new name', // グループ名の変更
  introduction: 'this is introduction.', // グループ概要の変更
  // v2.6.0からは、グループメンバーがグループカスタムフィールド変更のグループプロンプトメッセージ
  groupCustomField: [{ key: 'group_level', value: 'high'}] // グループディメンションカス
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 変更後のグループの詳細データ
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // グループ情報変更失敗の関連情報
});
// v2.6.2からは、全員ミュートおよびミュート解除機能が利用できます。現在はグループ全員をミュートにす
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  muteAllMembers: true, // trueは全員ミュート、falseは全員ミュートの解除を表します
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 変更後のグループの詳細データ
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // グループ情報変更失敗の関連情報
});
```

グループプロファイル処理モジュールのその他のSDKコードの例

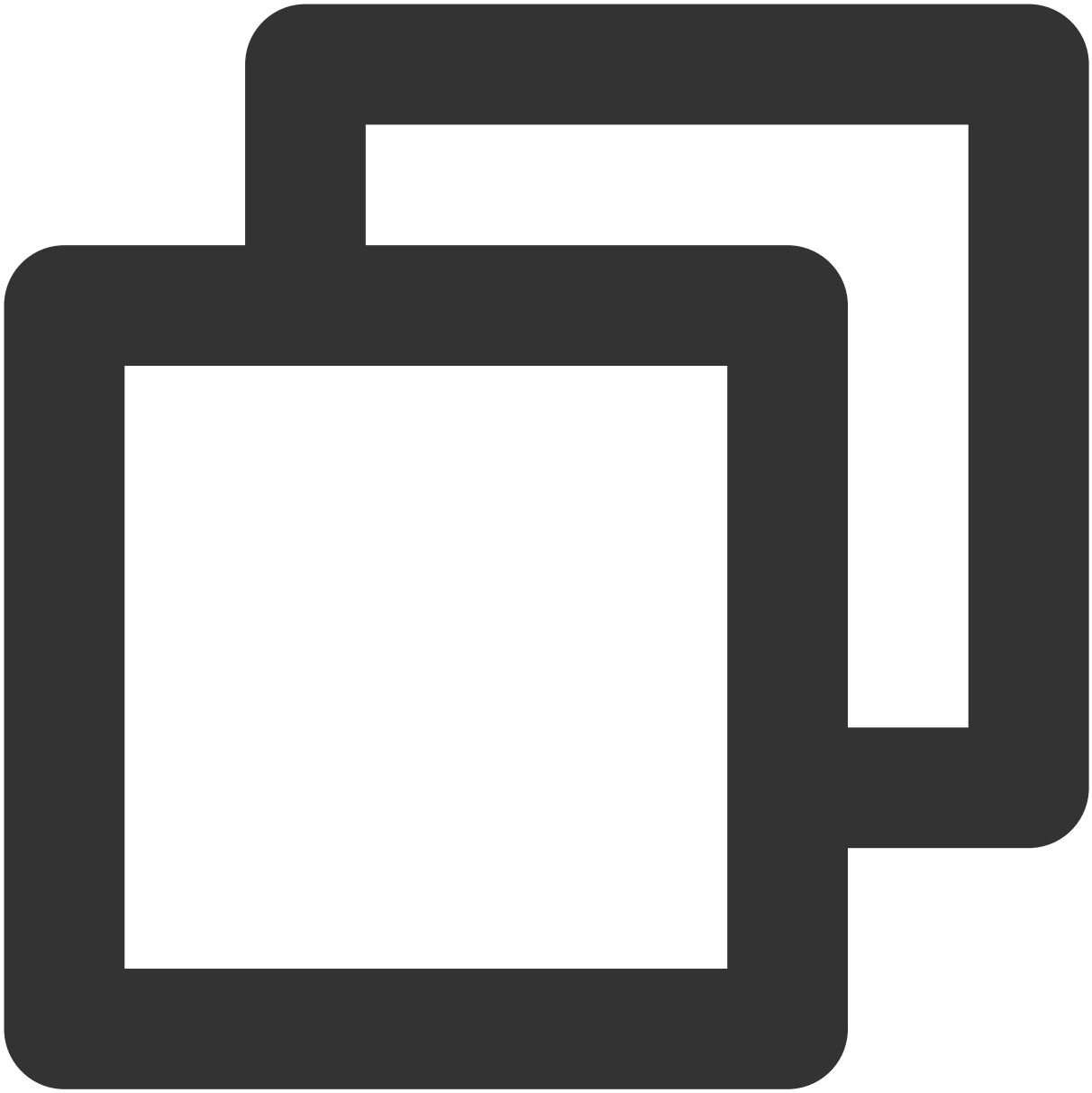
メッセージの優先度

ライブルームの特色の一つに、ユーザーのメッセージ量が非常に多く、各ユーザーが頻繁にメッセージの送受信を行う可能性がある、ということがあげられます。アップストリームメッセージがIMの頻度管理閾値に達すると、IMバックエンドは一部のメッセージを破棄することでシステムの安定した実行を保証します。実際、メッセージがクライアントに届く頻度が高すぎるとメッセージの可読性も低下するため、ライブストリーミンググループメッセージに対する頻度管理の必要性は非常に高いです。
グループメッセージの優先度は3段階に分かれ、バックエンドは優先度の高いメッセージを優先的に送信します。このため、ユーザーはメッセージの重要度に応じて、適切な優先度を選択しなければなりません。
3段階の優先度は高い方から順に、それぞれ次のようになりま：

優先度	意味
High	高優先度
Normal	通常の優先度
Low	低優先度

メッセージ破棄ポリシーは次のとおりです。

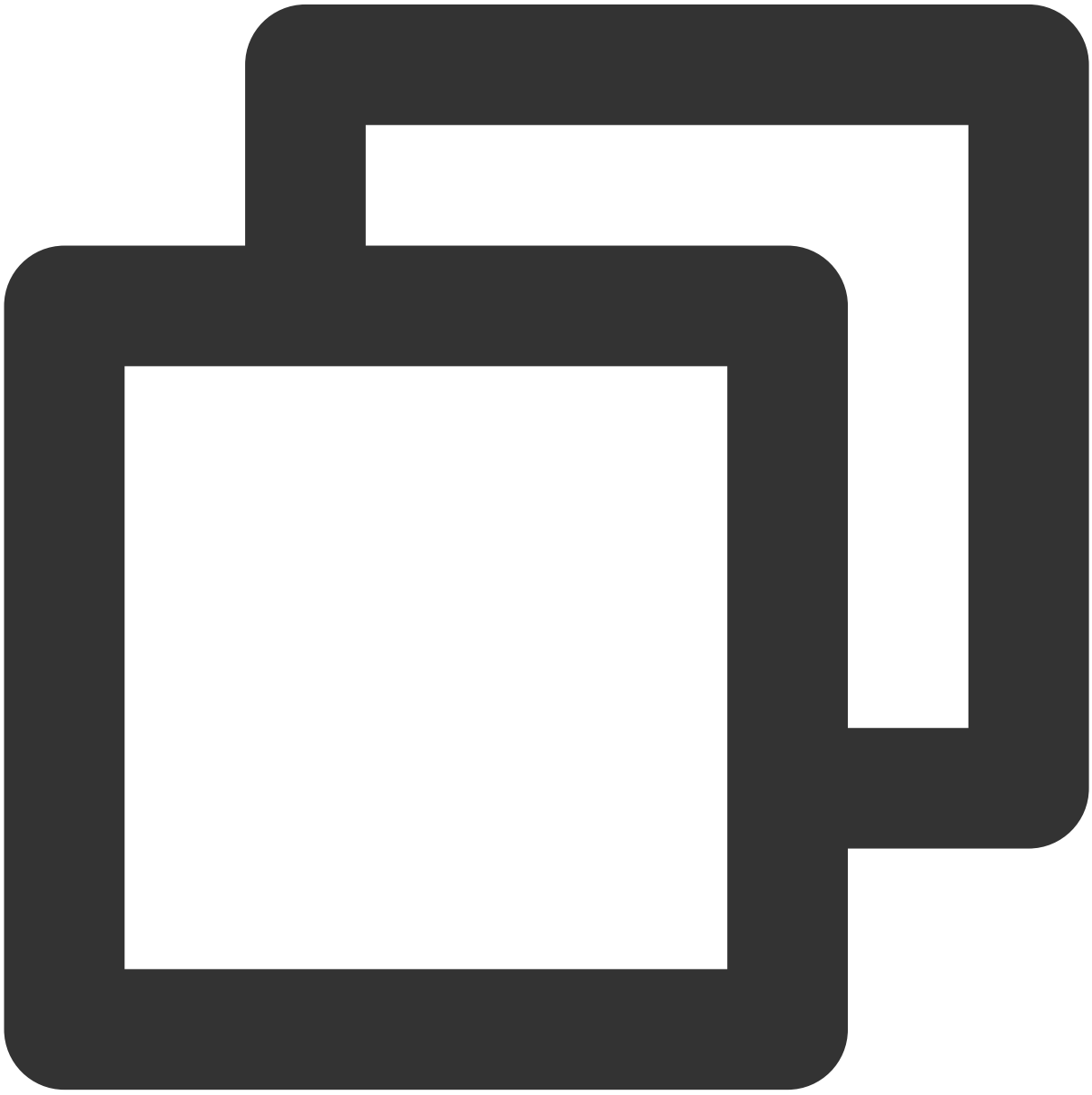
総メッセージ数頻度管理とは、1つのグループが1秒間に送信できる最大メッセージ数の制限であり、デフォルト値は40通/秒で、毎秒平均の頻度制限を採用します。メッセージ数が制限を超過すると、バックエンドは優先度が相対的に高いメッセージを優先的に送信します。優先度が同等のメッセージの順序はランダムとなります。
頻度制限の対象となったメッセージは送信されず、メッセージ履歴にも保存されませんが、送信者には成功と返されます。グループ内での発言前のコールバックはトリガーされますが、グループ内発信後コールバックはトリガーされません。
つまり、ライブルームでメッセージの優先度ルールを制定することは非常に重要です。
ライブルームのメッセージについて、Tencentがユーザーにとっての重要性に基づいて行う優先度の区分は、高い方から順に次のようになります：
1. ユーザーの投げ銭、ギフトメッセージ。このタイプのメッセージは、ライブルームの全ユーザーが見ることをユーザーが望むものです
2. 通常のテキスト、音声、画像などのメッセージ
3. 「いいね」メッセージ
説明：
通常、消費者側のユーザー自身が送信したメッセージは、優先度の高低を問わず、必ず画面に表示させる必要があります。メッセージ量が特に多い場合、ユーザーは他のユーザーのメッセージの一部を受け取らないことが可能です。
メッセージの優先度設定コードの例は次のとおりです：
Android
iOS&Mac
Flutter
Web



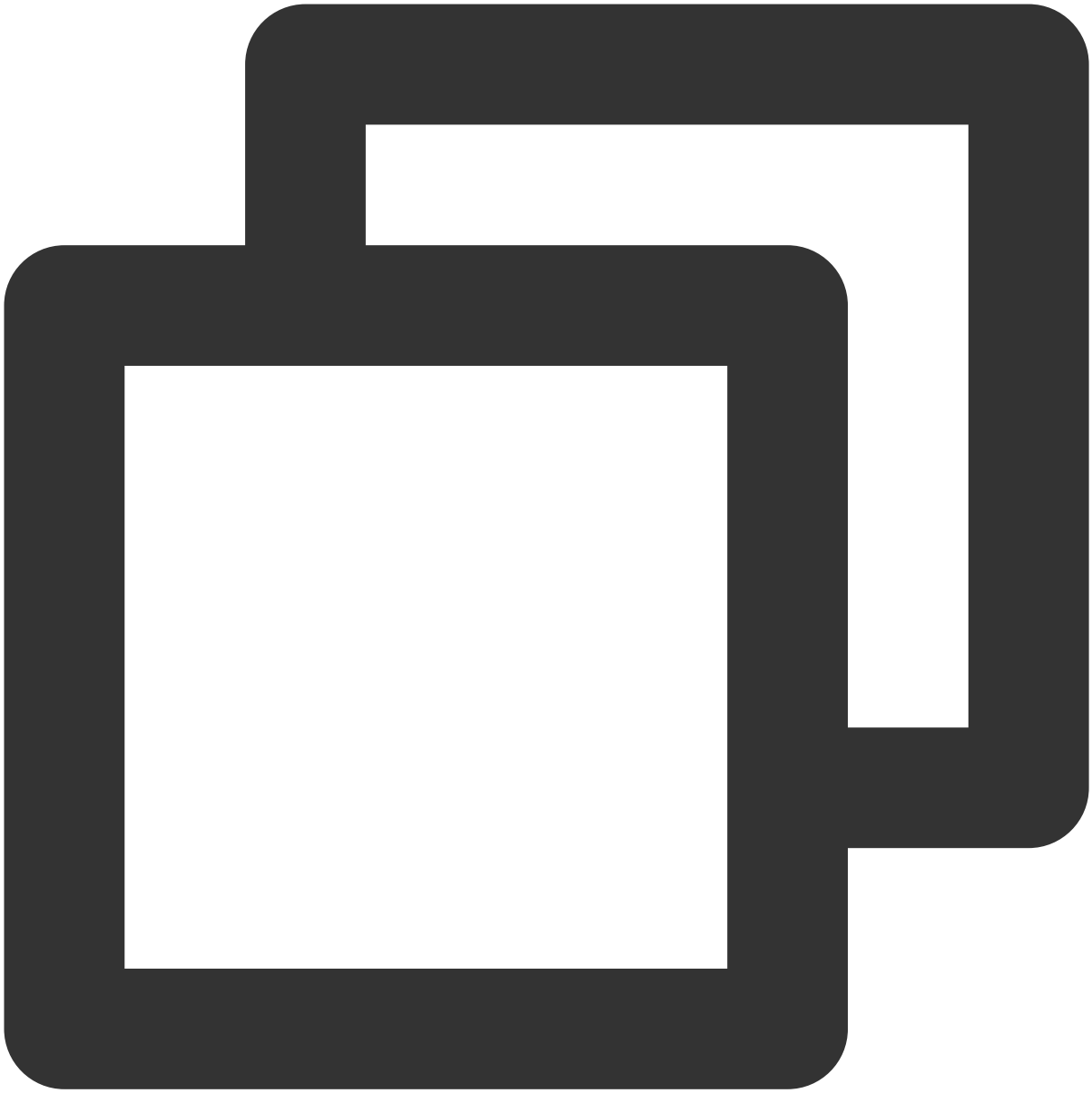
```
// カスタムメッセージの作成
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createCustomMessage("シ
// メッセージの優先度を高優先度メッセージに設定します
v2TIMMessage.setPriority(V2TIMMessage.V2TIM_PRIORITY_HIGH)
// メッセージを送信します
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // カスタムメッセージは進捗をコールバックしません
    }
```

```
    @Override
    public void onSuccess(V2TIMMessage message) {
        // グループチャットカスタムメッセージの送信に成功しました
    }

    @Override
    public void onError(int code, String desc) {
        // グループチャットカスタムメッセージの送信に失敗しました
    }
});
```



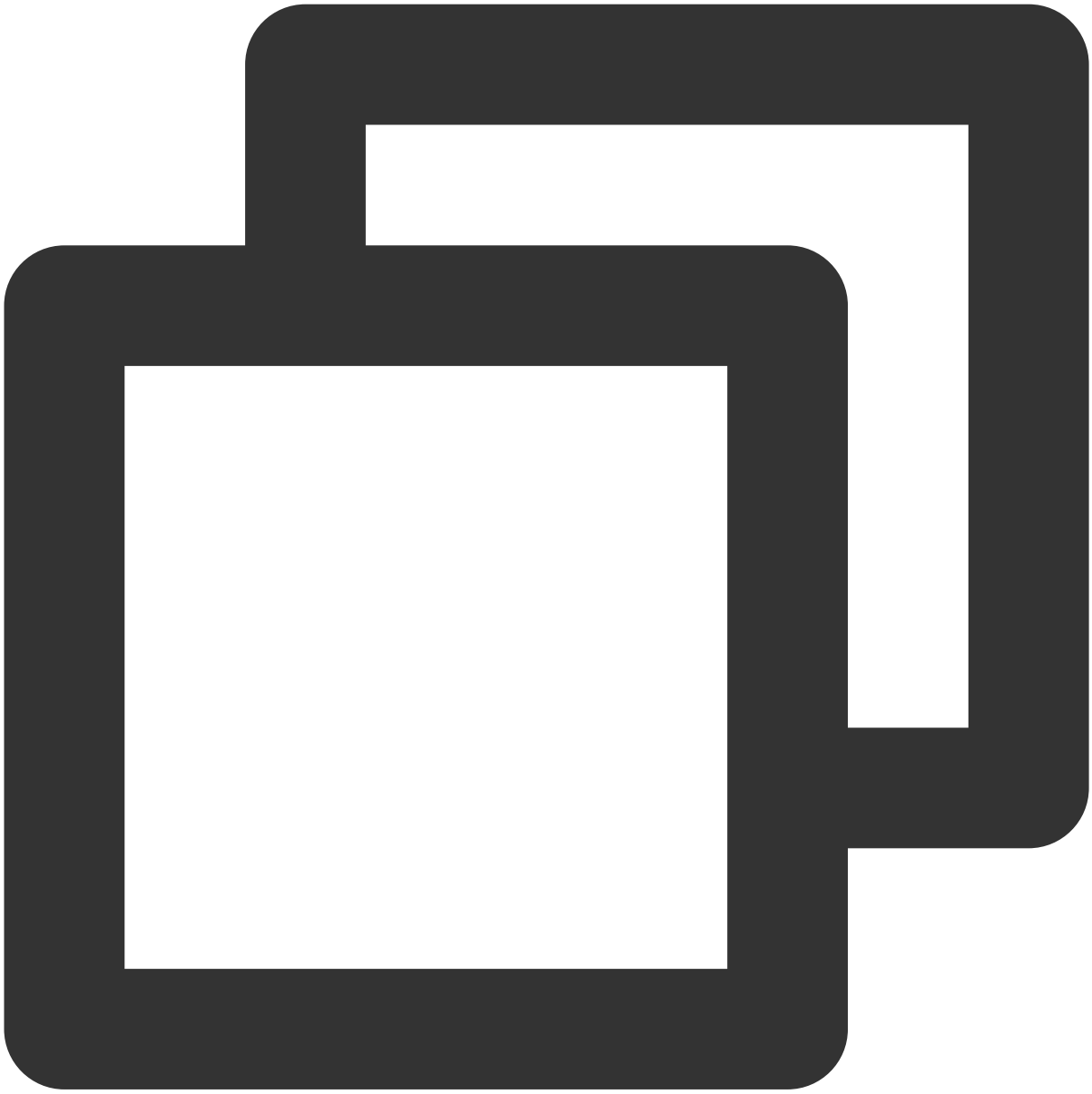
```
/ テキストメッセージを作成します
V2TIMMessage *message = [[V2TIMManager sharedInstance] createTextMessage:@"content"
// メッセージを送信します
[V2TIMManager.sharedInstance sendMessage:message
                                receiver:@"userID"
                                 groupID:nil
                                priority:V2TIM_PRIORITY_NORMAL
                          onlineUserOnly:NO
                         offlinePushInfo:nil
                                progress:nil
                                succ:^{
    // テキストメッセージ送信に成功しました
}
                                fail:^(int code, NSString *desc) {
    // テキストメッセージ送信に失敗しました
}];
```



```
/ テキストメッセージを作成します
V2TimValueCallback<V2TimMsgCreateInfoResult> createTextAtMessageRes = await Tencent
    text: "test",
    atUserList: [],
  );
 if(createTextAtMessageRes.code == 0){
    String id =  createTextAtMessageRes.data.id;

        // テキストメッセージを送信します
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin.v2TI
    if(sendMessageRes.code == 0){
```

```
        // 送信に成功
      }
    }
```

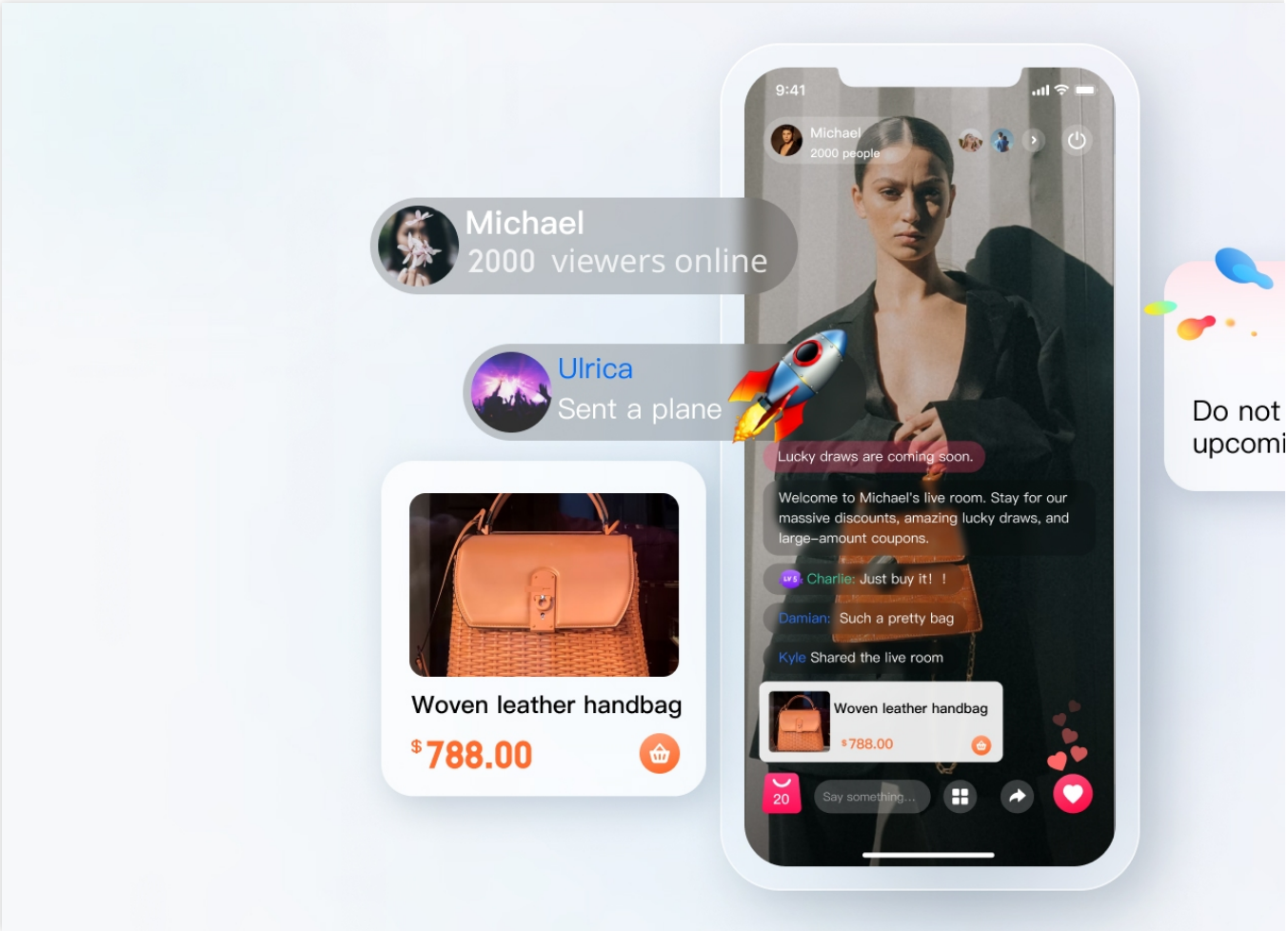


```
// テキストメッセージを送信します
// 1. メッセージインスタンスを作成します。インターフェースから返されたインスタンスを画面に表示でき
let message = tim.createTextMessage({
  to: 'user1',
  conversationType: TIM.TYPES.CONV_C2C,
  // メッセージの優先度をグループチャットに適用します（v2.4.2からサポート）。あるグループのメッセ
  // サポートされる列挙値：TIM.TYPES.MSG_PRIORITY_HIGH, TIM.TYPES.MSG_PRIORITY_NORMAL（
```

```
  // priority: TIM.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    text: 'Hello world!'
  },
  // v2.20.0からは、C2Cメッセージの開封確認機能がサポートされました。送信したメッセージの開封確認
  needReadReceipt: true
  // メッセージのカスタムデータ（クラウドに保存され、相手側に送信されます。プログラムをアンインスト
  // cloudCustomData: 'your cloud custom data'
});
// 2. メッセージの送信
let promise = tim.sendMessage(message);
promise.then(function(imResponse) {
  // 送信に成功
  console.log(imResponse);
}).catch(function(imError) {
  // 送信に失敗
  console.warn('sendMessage error:', imError);
});
```

メッセージ優先度設定のその他のSDKバージョンコードの例

ギフトと「いいね」メッセージのベストプラクティス



ギフトメッセージ

1. クライアントの短時間接続リクエストが自身の業務サーバーに届き、課金ロジックに関与します。
2. 課金後、送信者は「XXXがXXX（ギフト）を贈りました」というメッセージを直接見ることができます。（送信者が自身の贈ったギフトを確認できるようにするためのものです。メッセージ量が多い場合は、破棄ポリシーがトリガーされる可能性があります）
3. 料金の決済後、サーバーのインターフェースを呼び出してカスタムメッセージ（ギフト）を送信します
4. ギフトを連続して贈るケースではメッセージの一括化が必要です

4.1 ギフトの数を直接選択して贈る場合、例えば99個のギフトを選択した場合は、メッセージを1通送信し、パラメータにギフト数を99と入力します

4.2 ギフトを連続送付し、いくつで止めるか決めていない場合は、20個（数は自身で調整）ごとに、または連続送付の間隔が1秒を超えた場合に1通送信することができます。このロジックの場合、例えば99個のギフトを連続送付する場合、最適化後は5通の送信で済みます

「いいね」メッセージ

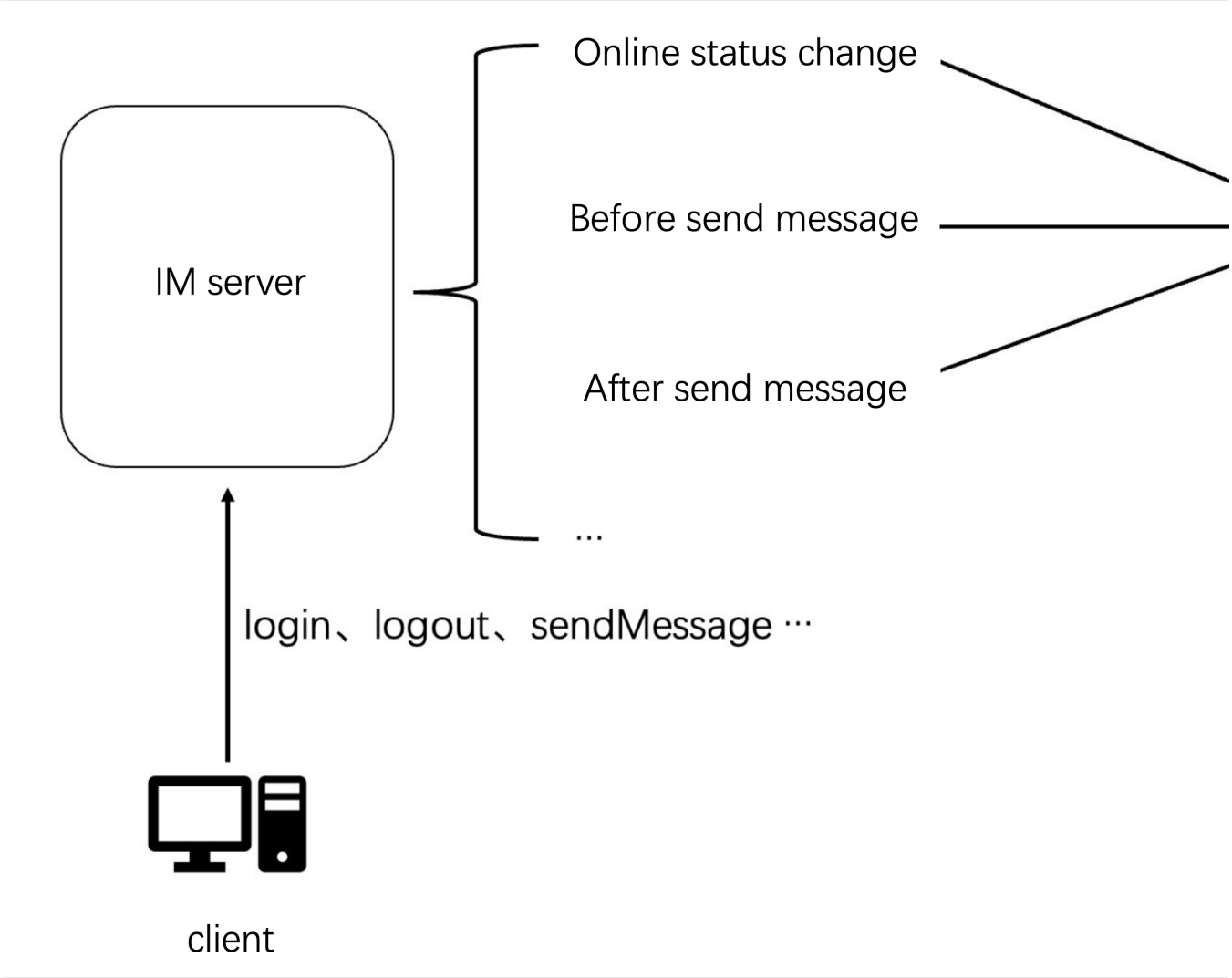
1. 「いいね」とギフトではやや異なり、「いいね」は課金の必要がない場合が多いため、通常は端末からの直接送信方式を用います

2. サーバー側でカウントする必要がある「いいね」メッセージについては、クライアントで流れを制限した後、クライアントで「いいね」回数を統計し、短時間内の複数の「いいね」メッセージを一括化し、メッセージを1回だけ送信すればよいようにします。業務側サーバーはメッセージ送信前コールバックから「いいね」回数を取得し、統計を行います
3. カウントの必要がない「いいね」メッセージについては、ステップ2のロジックと同様に、業務側がクライアントで「いいね」メッセージの流れを制限してから送信します。メッセージ送信前コールバックからカウントする必要はありません。

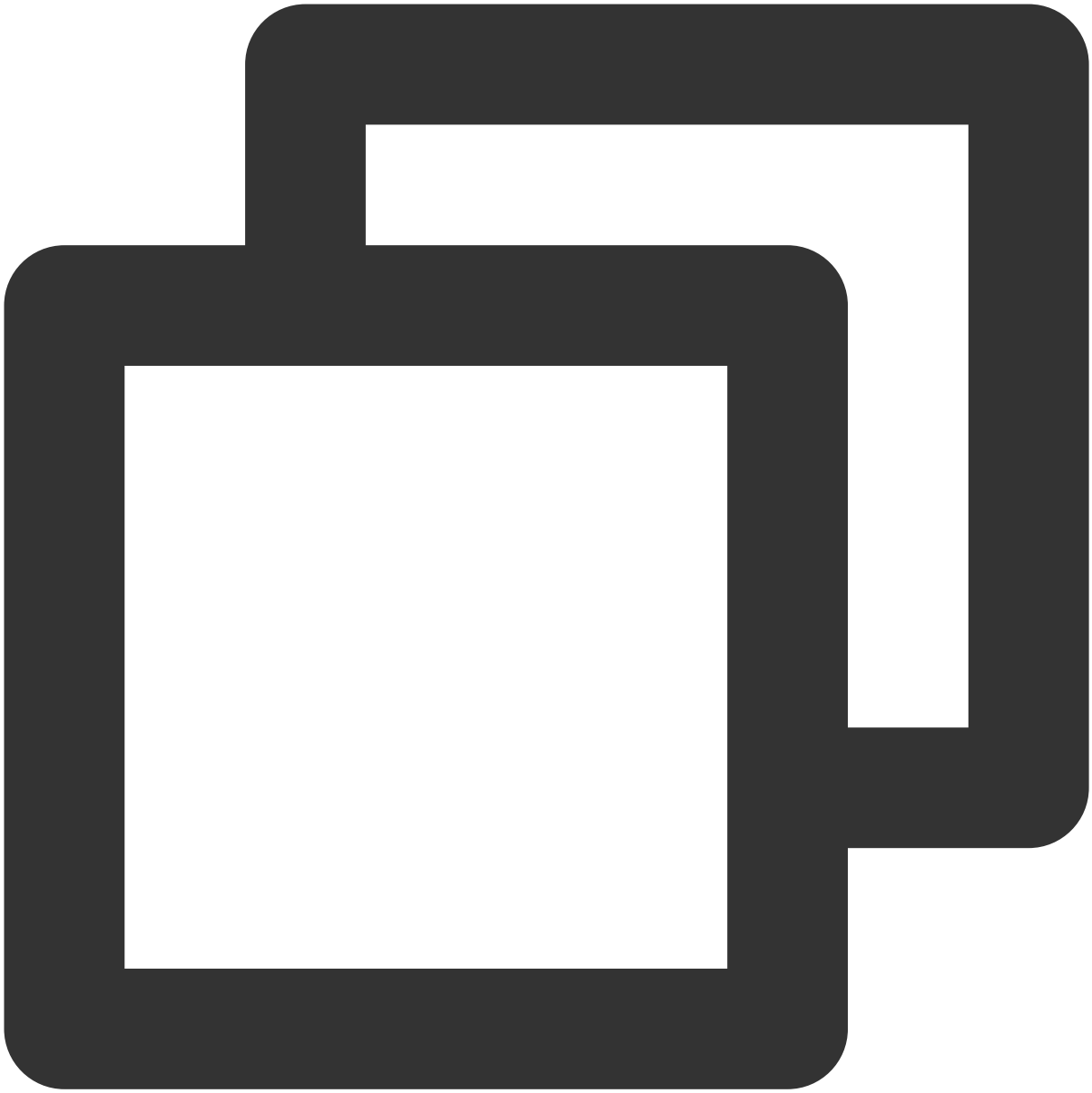
ライブルームユーザーの身分、レベル

ほとんどのライブルームにはユーザーレベルの概念があります。開発者は次の数点に基づき、一定の重み付けに従ってユーザーのレベルを計算することができます。
1. ライブルームユーザーのオンライン時間の長さ
2. ライブルームユーザーが送信に成功した一般ライブルームメッセージ数
3. ライブルームユーザーの「いいね」、ギフト送付数
4. ライブルームユーザーがライブルームメンバーになったかどうか
5. ...
このうち、IMに関連する、オンライン時間の長さの統計、メッセージ量の統計などはすべてIMのコールバックを使用して実現します。最初の準備作業のモジュールに、コールバック設定の関連ガイドがあります。コンソールにあるコールバックの詳細は次のとおりです：

統計関連情報のフローチャートは次のとおりです：



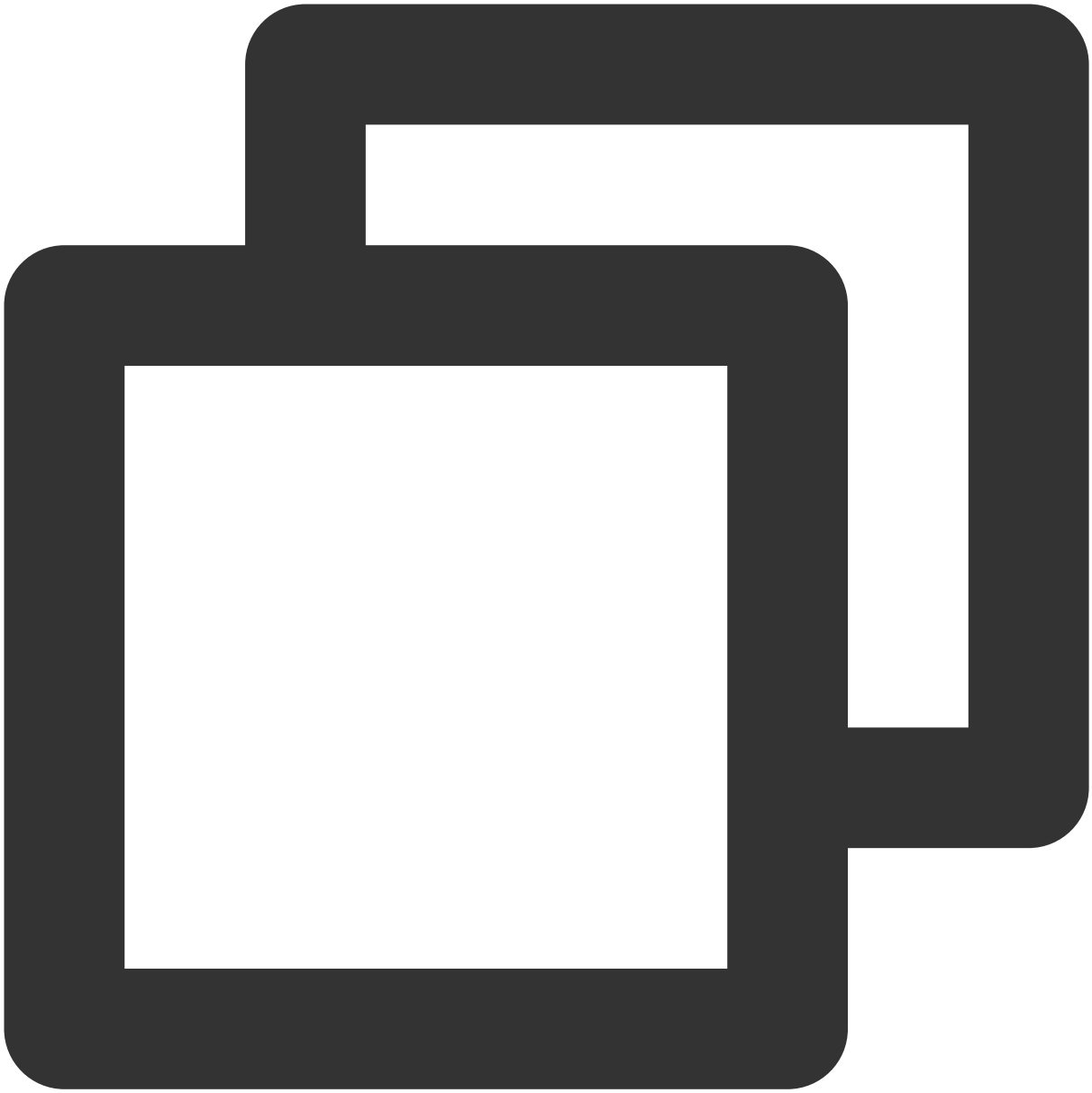
メッセージ送信後コールバックデータの例です：



```
{
    "CallbackCommand": "Group.CallbackAfterSendMsg", // コールバックコマンド
    "GroupId": "@TGS#2J4SZEAEL", // グループID
    "Type": "Public", // グループタイプ
    "From_Account": "jared", // 送信者
    "Operator_Account":"admin", // リクエストの送信者
    "Random": 123456, // 乱数
    "MsgSeq": 123, // メッセージ番号
    "MsgTime": 1490686222, // メッセージの時間
    "OnlineOnlyFlag": 1, //オンラインメッセージは1、そうでない場合は0となります。ライブストリー
    "MsgBody": [ // メッセージボディです。TIMMessageのメッセージオブジェクトをご参照ください
```

```
            {
                "MsgType": "TIMTextElem", // テキスト
                "MsgContent": {
                    "Text": "red packet"
                }
            }
        ],
        "CloudCustomData": "your cloud custom data"
    }
```

開発者はデータ内のMsgBodyのメッセージタイプに基づいて一般メッセージと「いいね」ギフトメッセージを区別することができます。すべてのフィールドの意味については、ドキュメントサーバーAPIをご参照ください。
ユーザーのオンラインステータス変更コールバックデータの例です：



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

```
            },
            {
                "Platform": "Android"
            }
        ]
    }
```

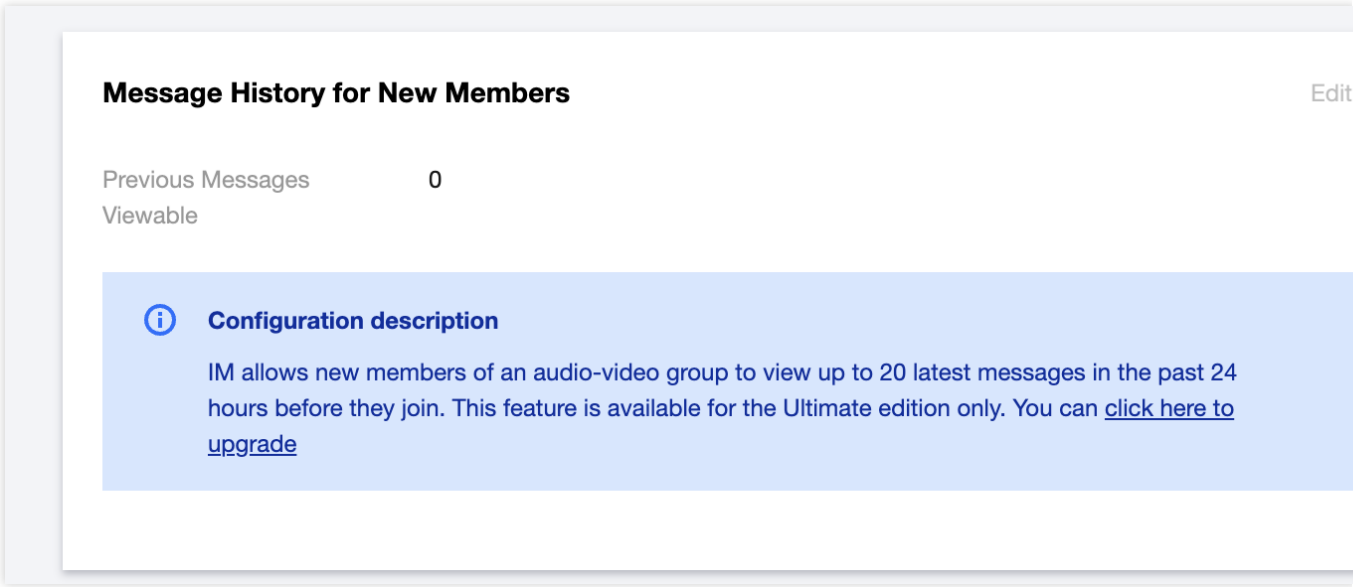
開発者はInfo内のフィールドに基づいてユーザーのオンラインステータスを判断することができます。すべてのフィールドの意味については、ドキュメントサーバーAPIをご参照ください。

説明：

また、多くのライブルームではライブルームの人気度を表示し、人気度に応じてライブルームをユーザーにおすすめしています。ライブルームの人気度の統計方法はユーザーレベルの統計方法と似ており、こちらもIMが提供するコールバックシステムによって実現できます。ここでは詳しくはご説明しません。

ライブルームのメッセージ履歴

AVChatRoomを使用する際、デフォルトではライブルームのメッセージ履歴は保存されず、ユーザーが新たにライブルームに入室した場合は、入室後にユーザーが送信したメッセージしか見ることができません。グループの新規参加ユーザーの体験を最適化するため、次のように、ライブストリーミンググループユーザーが取得できるグループ参加前のメッセージの数をコンソールで設定することができます：



説明：

この機能はフラッグシップ版のユーザーのみアクティブ化することができます。またグループで24時間に最大20通までのメッセージ履歴の取得のみサポートします。

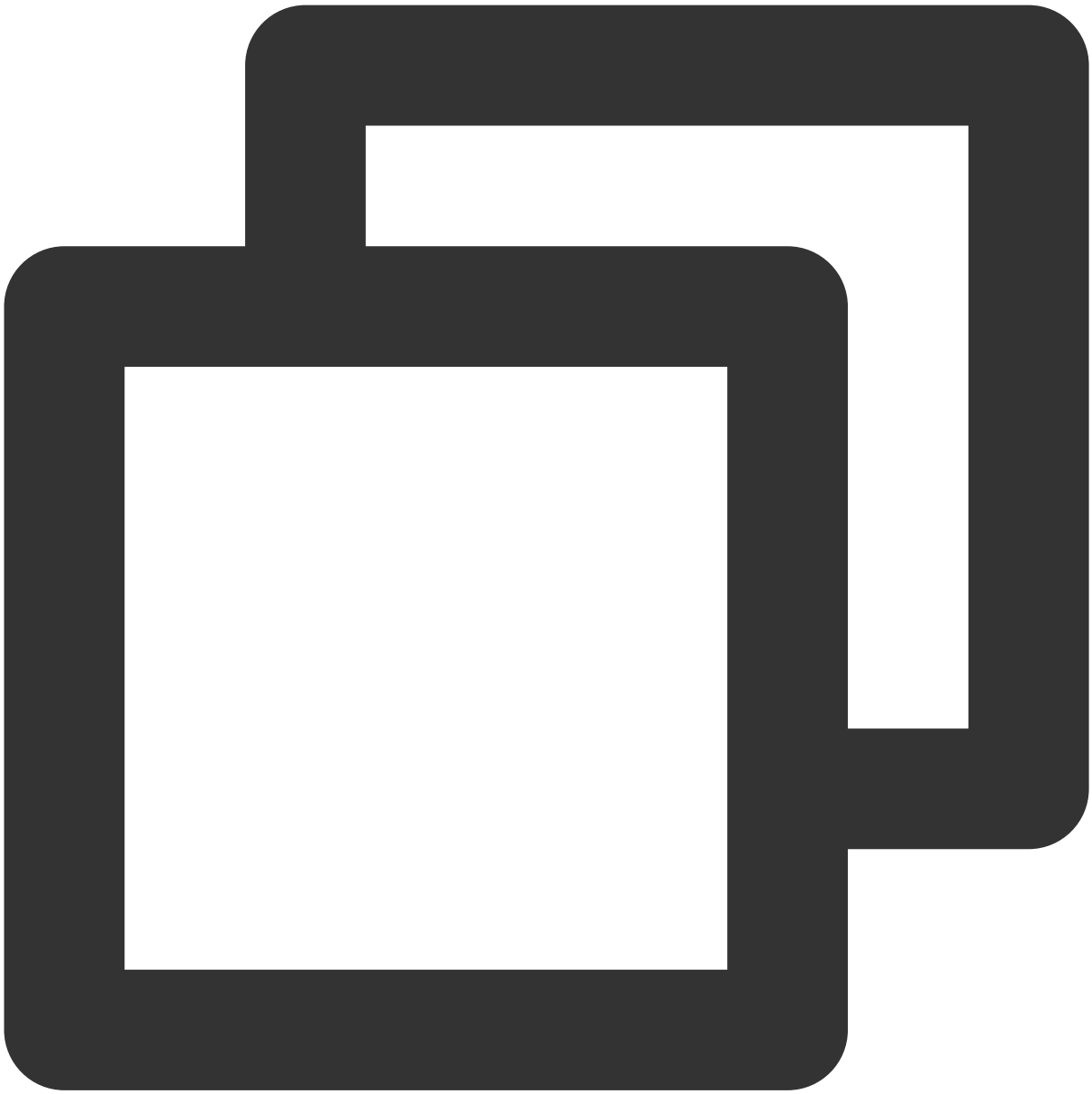
ライブストリーミンググループユーザーによるグループ参加前のメッセージの取得は、他グループのメッセージ履歴の取得と同様です。コードの例は次のとおりです：

Android

iOS&Mac

Flutter

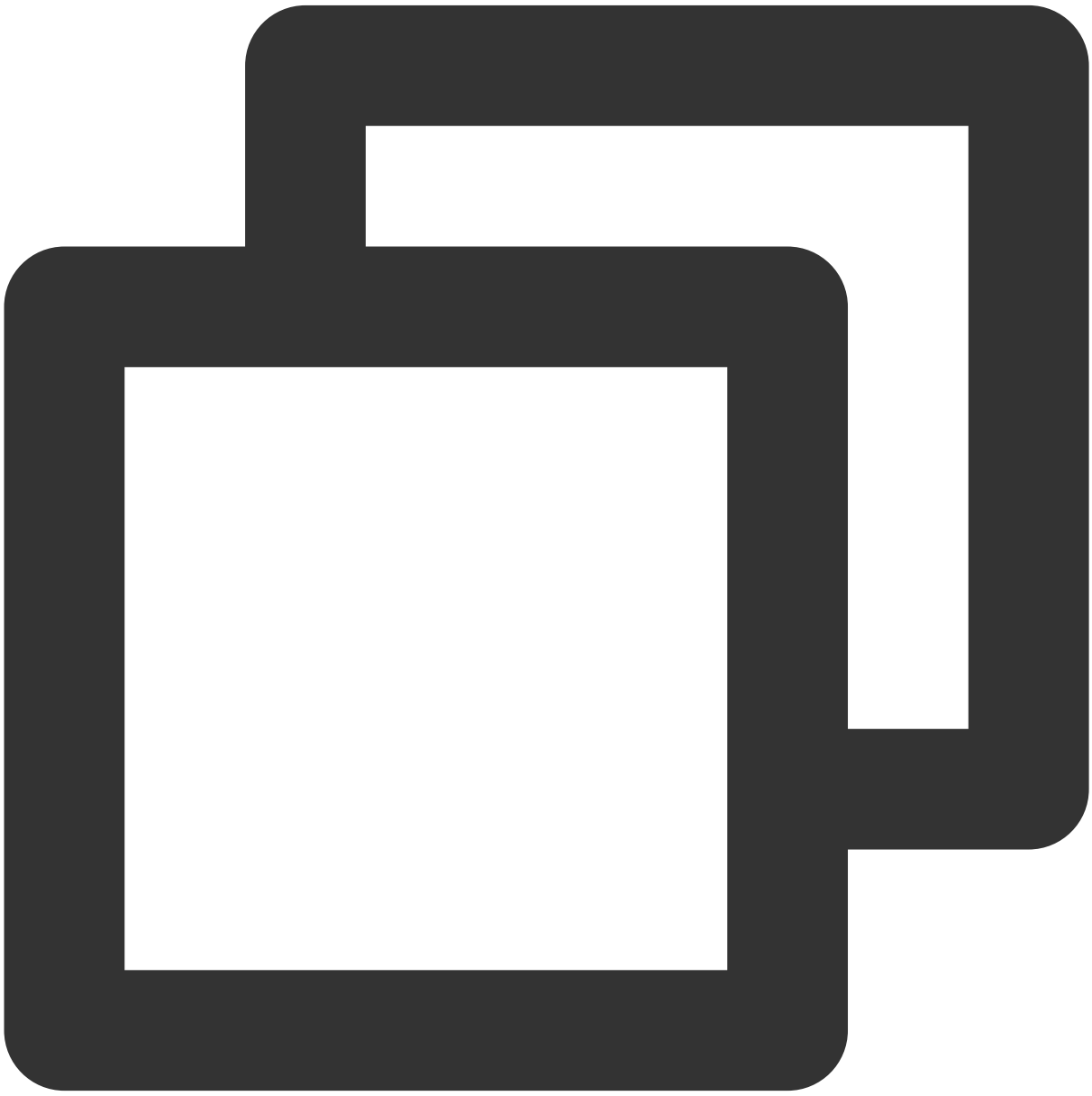
Web



```
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // クラウドの
option.setGetTimeBegin(1640966400);          // 2022-01-01 00:00:00から開始
option.setGetTimePeriod(1 * 24 * 60 * 60);   // 1日全体のメッセージを取得
option.setCount(Integer.MAX_VALUE);          // 時間範囲内の全メッセージを返す
option.setGroupID(#you group id#);           // グループチャットメッセージを取得
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
```

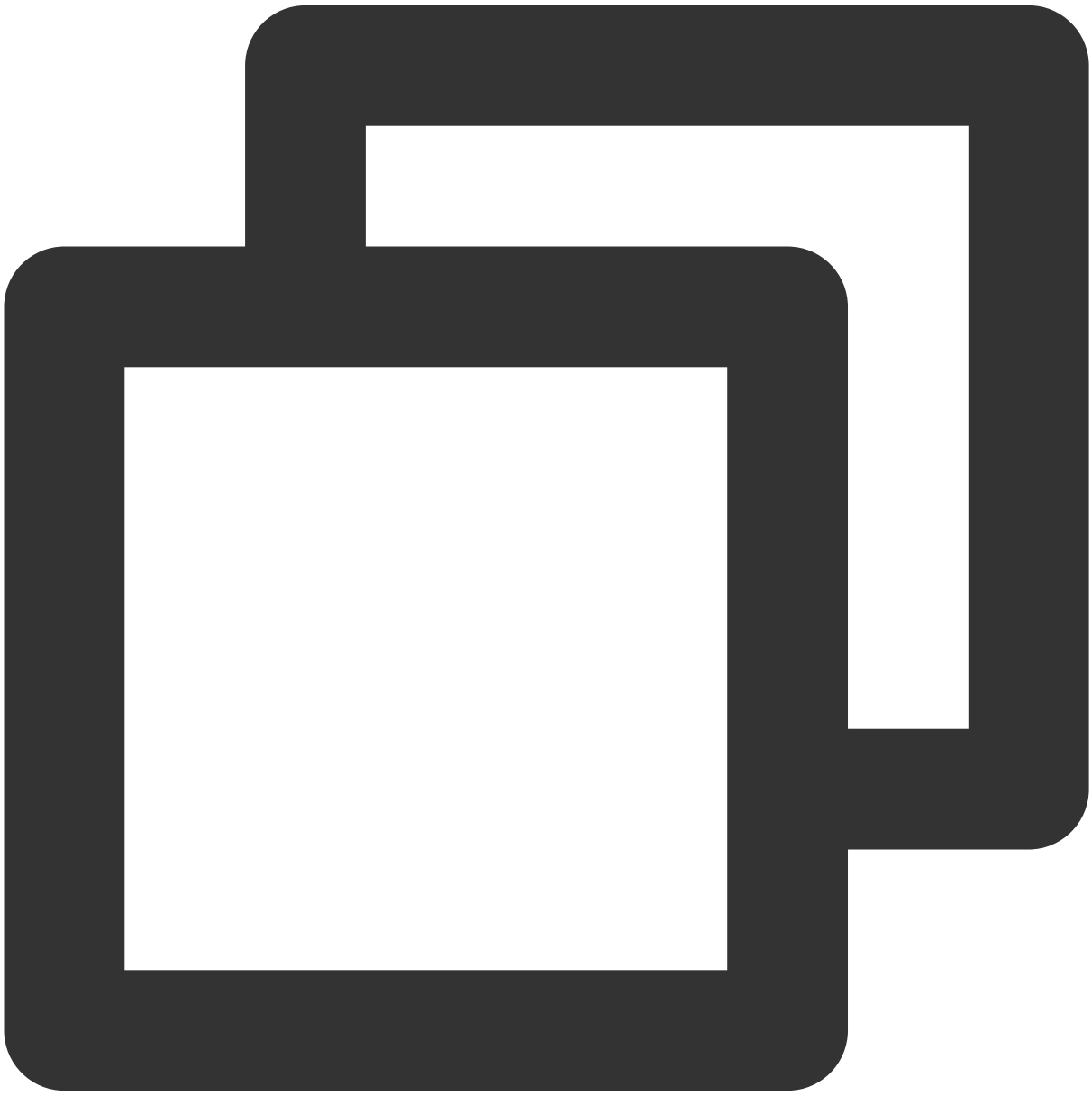
```
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        Log.i("imsdk", "success");
    }

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



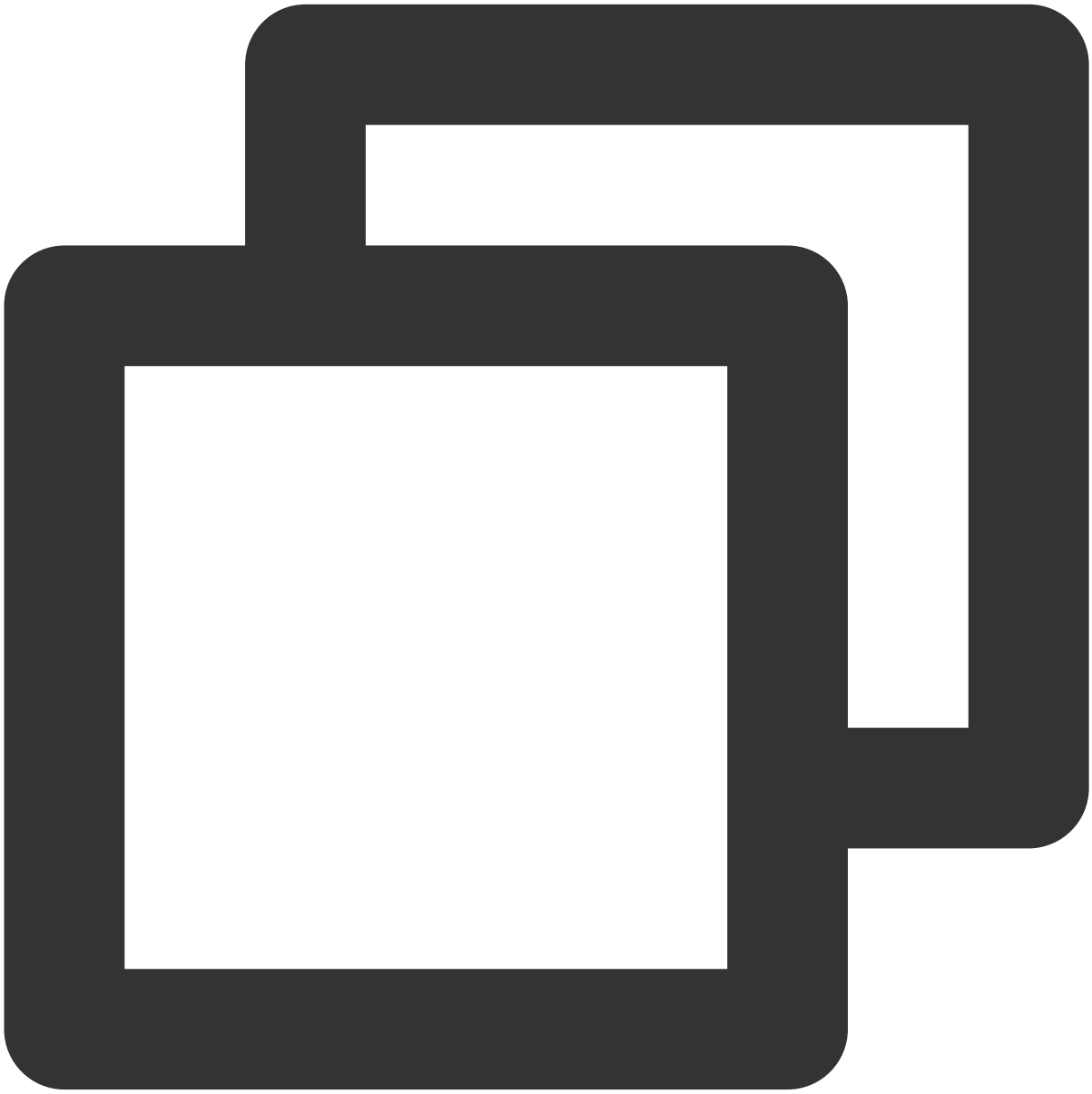
```
// シングルチャットメッセージ履歴の取得
```

```
// 初回の取得では、lastMsgをnilに設定します
// 再度取得する際、lastMsgは返されたメッセージリストの最後の1通を使用することができます
[V2TIMManager.sharedInstance getC2CHistoryMessageList:#your user id# count:20 lastM
    // 次回取得するlastMsgを記録し、次回の取得に使用します
    V2TIMMessage *lastMsg = msgs.lastObject;
    NSLog(@"success, %@", msgs);
} fail:^(int code, NSString *desc) {
    NSLog(@"fail, %d, %@", code, desc);
}];
```



```
// シングルチャットメッセージ履歴の取得
```

```
// 初回の取得では、lastMsgIDをnullに設定します
// 再度取得する際、lastMsgIDは返されたメッセージリストの最後の1通のidを使用することができます
TencentImSDKPlugin.v2TIMManager.getMessageManager().getC2CHistoryMessageList(
        userID: "userId",
        count: 10,
        lastMsgID: null,
);
```



```
// あるセッションを開いた際、最初に取得するメッセージリスト
let promise = tim.getMessageList({conversationID: 'C2Ctest', count: 15});
promise.then(function(imResponse) {
```

```
  const messageList = imResponse.data.messageList; // メッセージリスト。
  const nextReqMessageID = imResponse.data.nextReqMessageID; // 続けて取得する場合に使用
  const isCompleted = imResponse.data.isCompleted; // すべてのメッセージの取得が完了したか
});
// その他のメッセージをドロップダウンして確認
let promise = tim.getMessageList({conversationID: 'C2Ctest', nextReqMessageID, coun
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // メッセージリスト。
  const nextReqMessageID = imResponse.data.nextReqMessageID; // 続けて取得する場合に使用
  const isCompleted = imResponse.data.isCompleted; // すべてのメッセージの取得が完了したか
});
```

メッセージ履歴取得のその他のSDKバージョンコードの例

ライブルームオンライン人数

ライブルームでオンライン人数を表示することは、ライブストリーミングのシーンでのごく一般的なニーズの一つです。2種類の実現方法がありますが、それぞれにメリットとデメリットがあります。

1. クライアントSDKが提供するgetGroupOnlineMemberCountAPIによる定期ポーリング方式でグループのオンライン人数を取得します。

2. ユーザーのグループ参加・退出のバックエンドコールバックによって統計を行い、サーバーAPIのグループシステム通知またはグループカスタムメッセージなどによってグループ内の全メンバーに送信します。

クライアントSDKが提供するインターフェースによってオンライン人数を取得する方式は、シングルライブルームモードのユーザーにとっては基本的なニーズを満たすものです。ただし、Appに複数のライブルームがあり、なおかつ公開性の高い位置にライブルームのオンライン人数を表示する必要がある場合は、2つ目の方法でオンライン人数の統計を行うことをお勧めします。

説明：

開発者サーバーからクライアントに対しオンライン人数統計メッセージを送信する際は、例えば5秒に1回などの定時送信方式を用いることができますが、この方式では、ライブルームの人数にあまり変化がない場合には余分なネットワークオーバーヘッドが発生することになります。人数の変化率をモニタリングする方式で更新を行うことを開発者にはお勧めします。

ライブルームオンライン人数の正確性とリアルタイム性の優先度については、開発者が業務に応じて設定することができます。

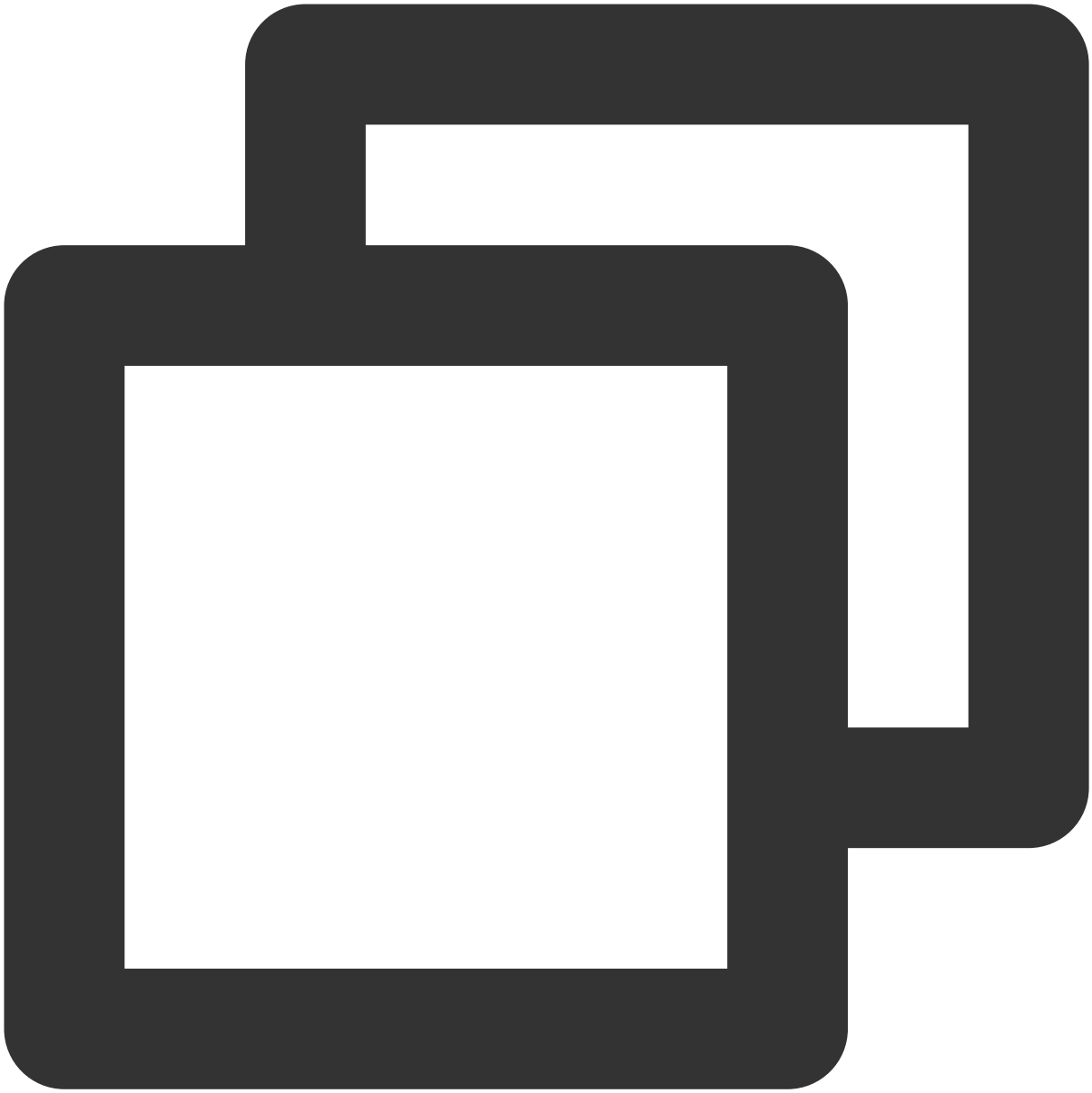
ライブルームでオンラインになっている人数を取得するコードは次のとおりです：

Android

iOS&Mac

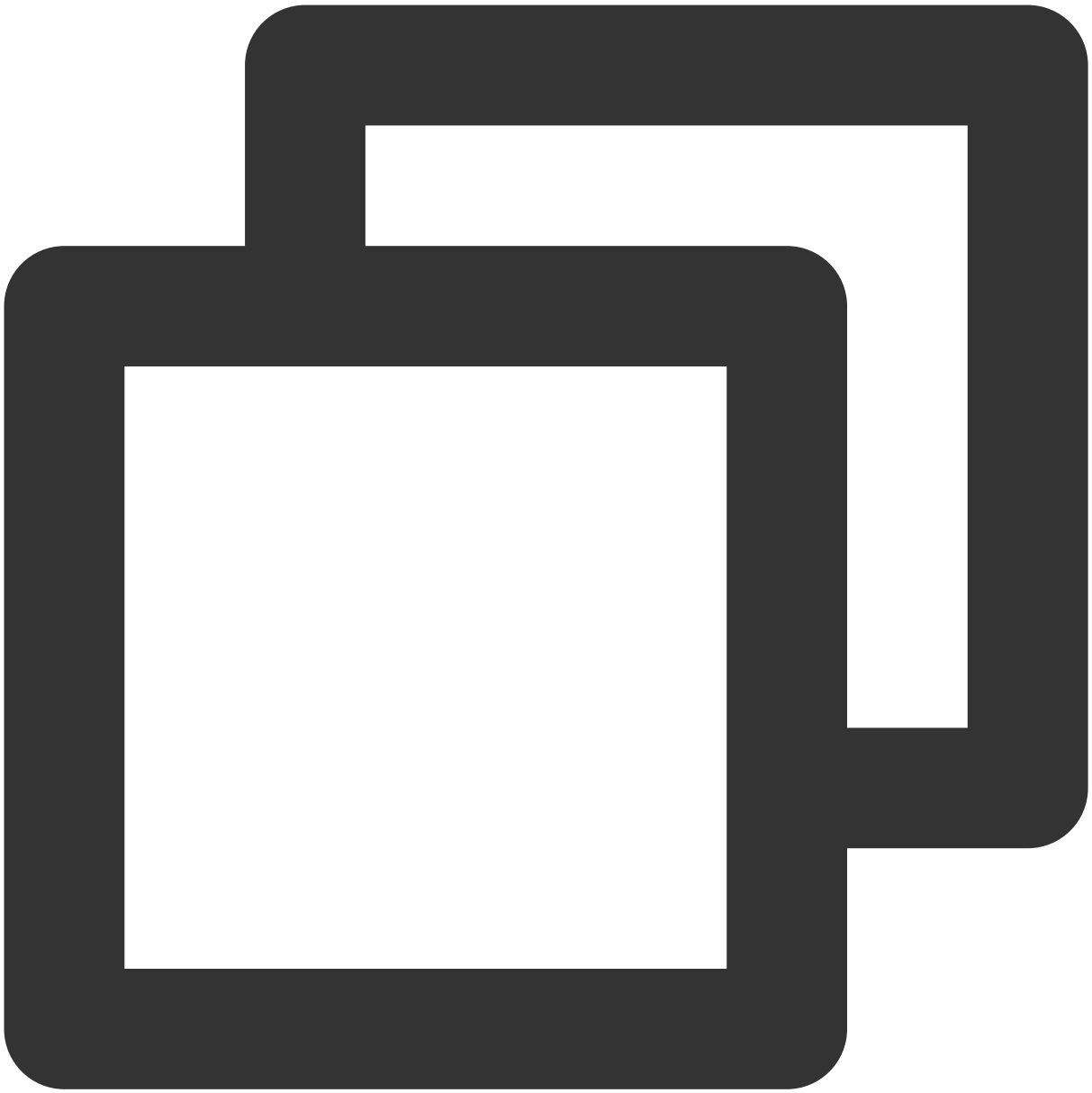
Flutter

Web

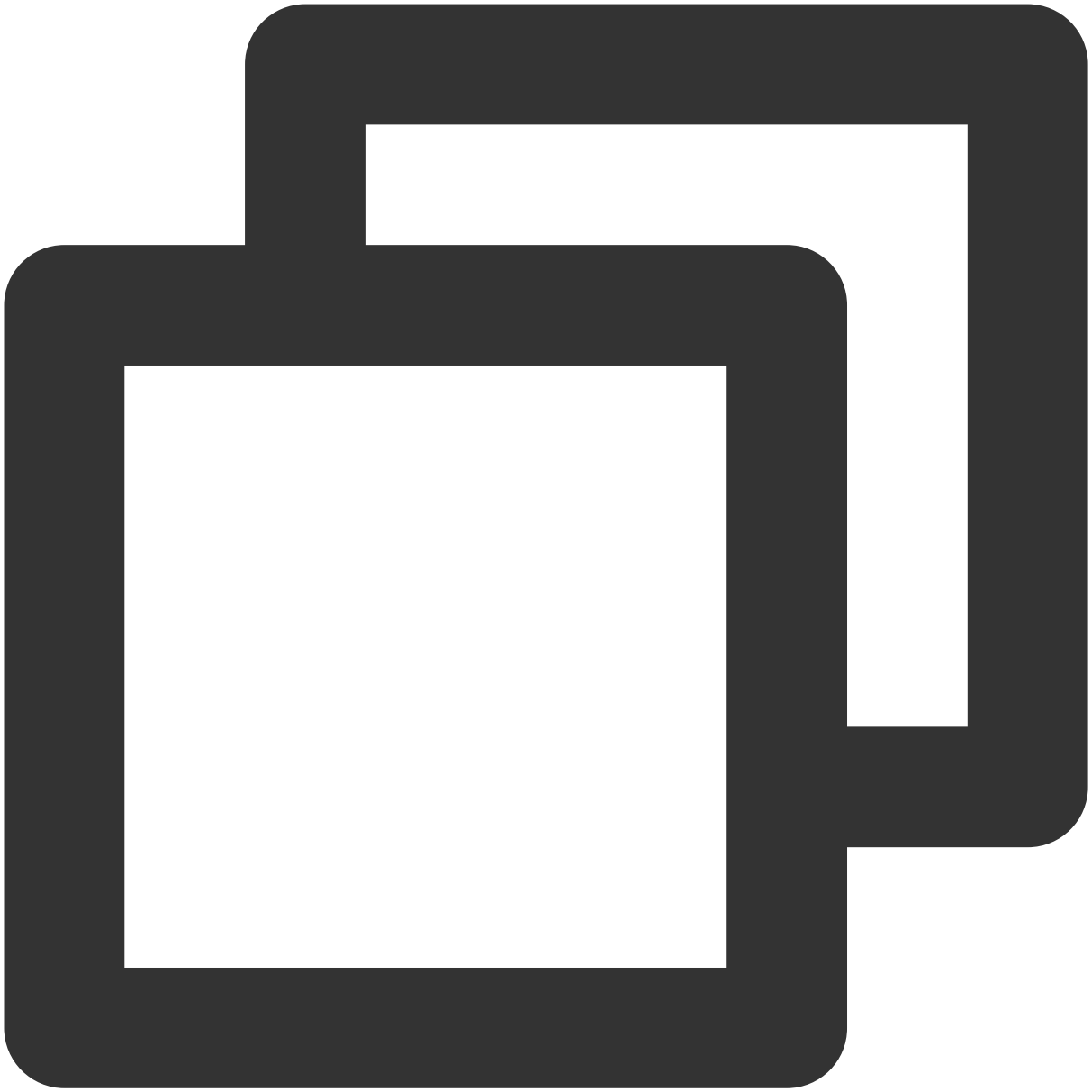


```
2TIMManager.getGroupManager().getGroupOnlineMemberCount("group_avchatroom", new V2T
  @Override
  public void onSuccess(Integer integer) {
      // ライブストリーミンググループのオンライン数の取得に成功
  }

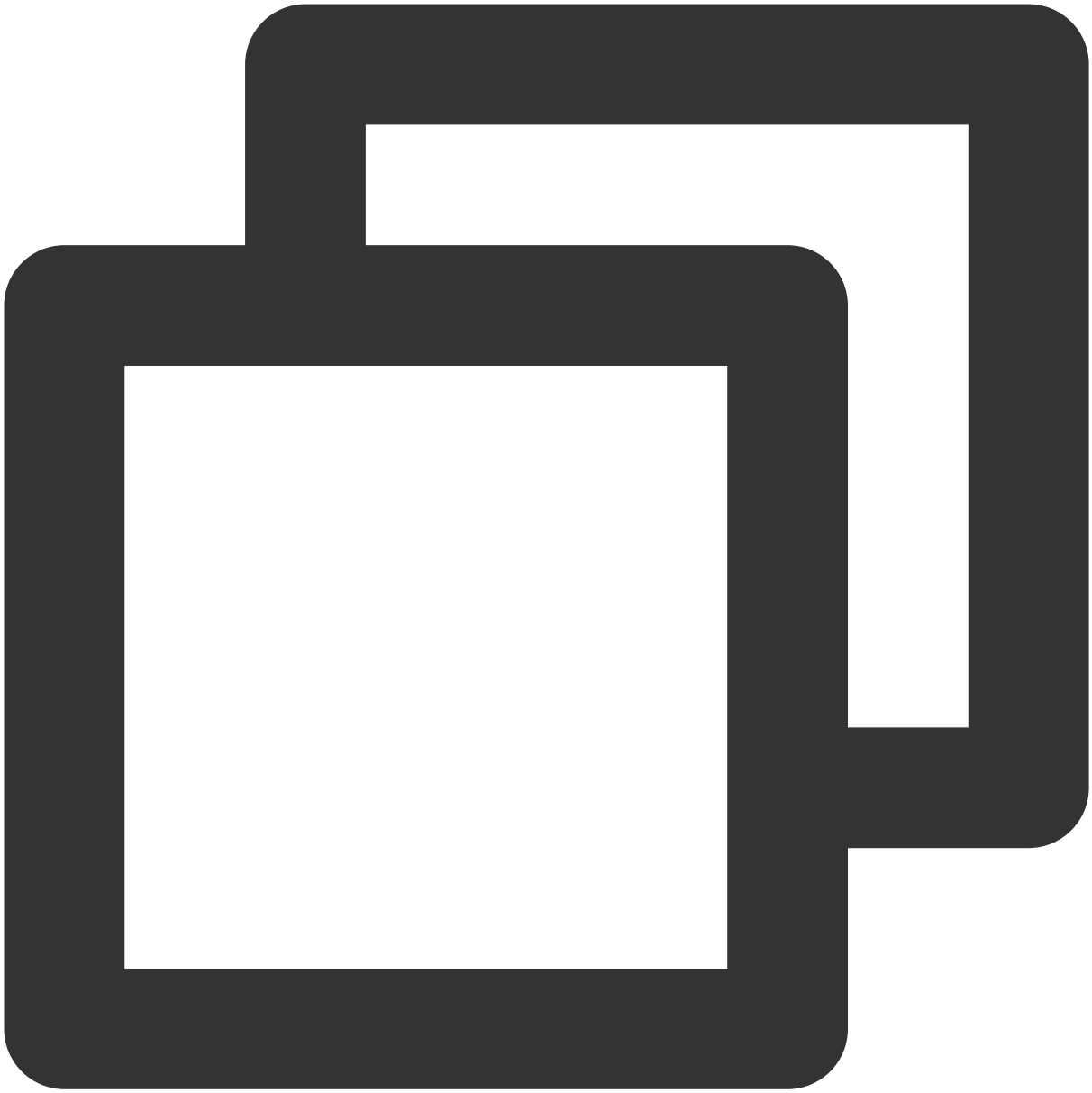
  @Override
  public void onError(int code, String desc) {
      // ライブストリーミンググループのオンライン数の取得に失敗
  }
});
```



```
[[V2TIMManager sharedInstance] getGroupOnlineMemberCount:@"group_avchatroom" succ:^
    // ライブストリーミンググループのオンライン数の取得に成功
} fail:^(int code, NSString *desc) {
    // ライブストリーミンググループのオンライン数の取得に失敗
}];
```



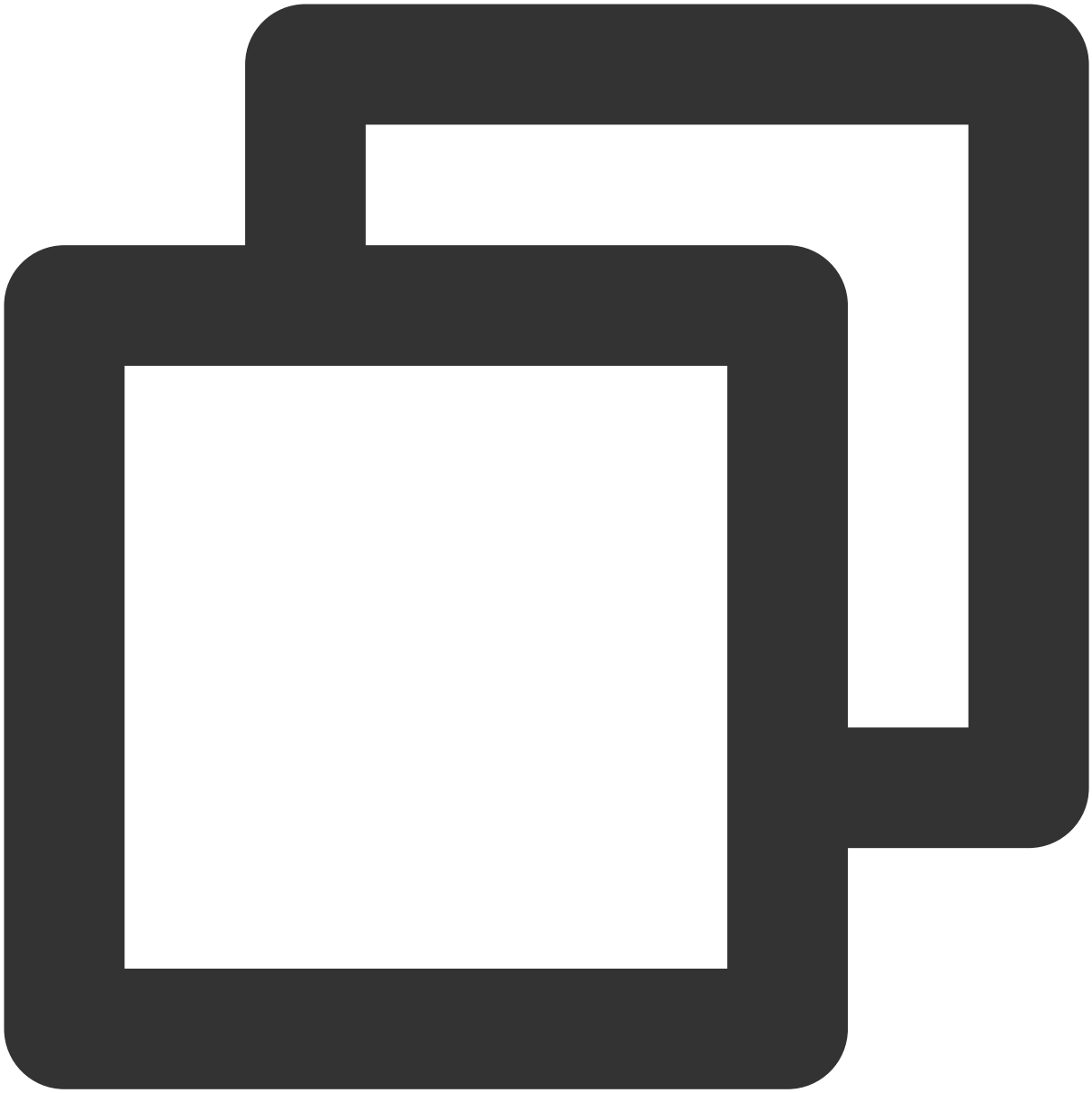
```
groupManager.getGroupOnlineMemberCount(groupID: '');
```



```
// v2.8.0より、ライブストリーミンググループのオンライン数のクエリをサポート
let promise = tim.getGroupOnlineMemberCount('group1');
promise.then(function(imResponse) {
  console.log(imResponse.data.memberCount);
}).catch(function(imError) {
  console.warn('getGroupOnlineMemberCount error:', imError); // ライブストリーミンググ
});
```

ライブルームのミュート

ライブルームのミュートには2種類あり、1つはライブルーム全体のミュート、もう1つは特定のユーザーに対するミュートです。2種類のミュートはそれぞれの業務シナリオで使用されます。
ミュートの設定は一般的にサーバーSDKを使用して行います。ライブルーム全体のミュートはグループ属性の設定の全員ミュートフィールドで設定します。単独のグループメンバーに対するミュートは、グループメンバー属性の設定で設定します。
ライブルーム管理者がバックエンドでグループミュートを設定すると、クライアントは対応するコールバックイベントを受信した後、ユーザーの入力ボックスをdisable状態に設定しなければなりません。これはユーザーがメッセージを送信した際に、メッセージの送信に失敗したと表示されることを防ぐためです。ミュートを解除した後は、入力ボックスはenable状態に設定しなければなりません。
クライアントの対応するコールバックコードは次のとおりです：
Android
iOS&Mac
Flutter
Web



```
// グループメンバーuserBを1分間ミュート
V2TIMManager.getGroupManager().muteGroupMember("groupA", "userB", 60, new V2TIMCall
  @Override
  public void onSuccess() {
      // グループメンバーのミュートに成功
  }

  @Override
  public void onError(int code, String desc) {
      // グループメンバーのミュートに失敗
  }
```

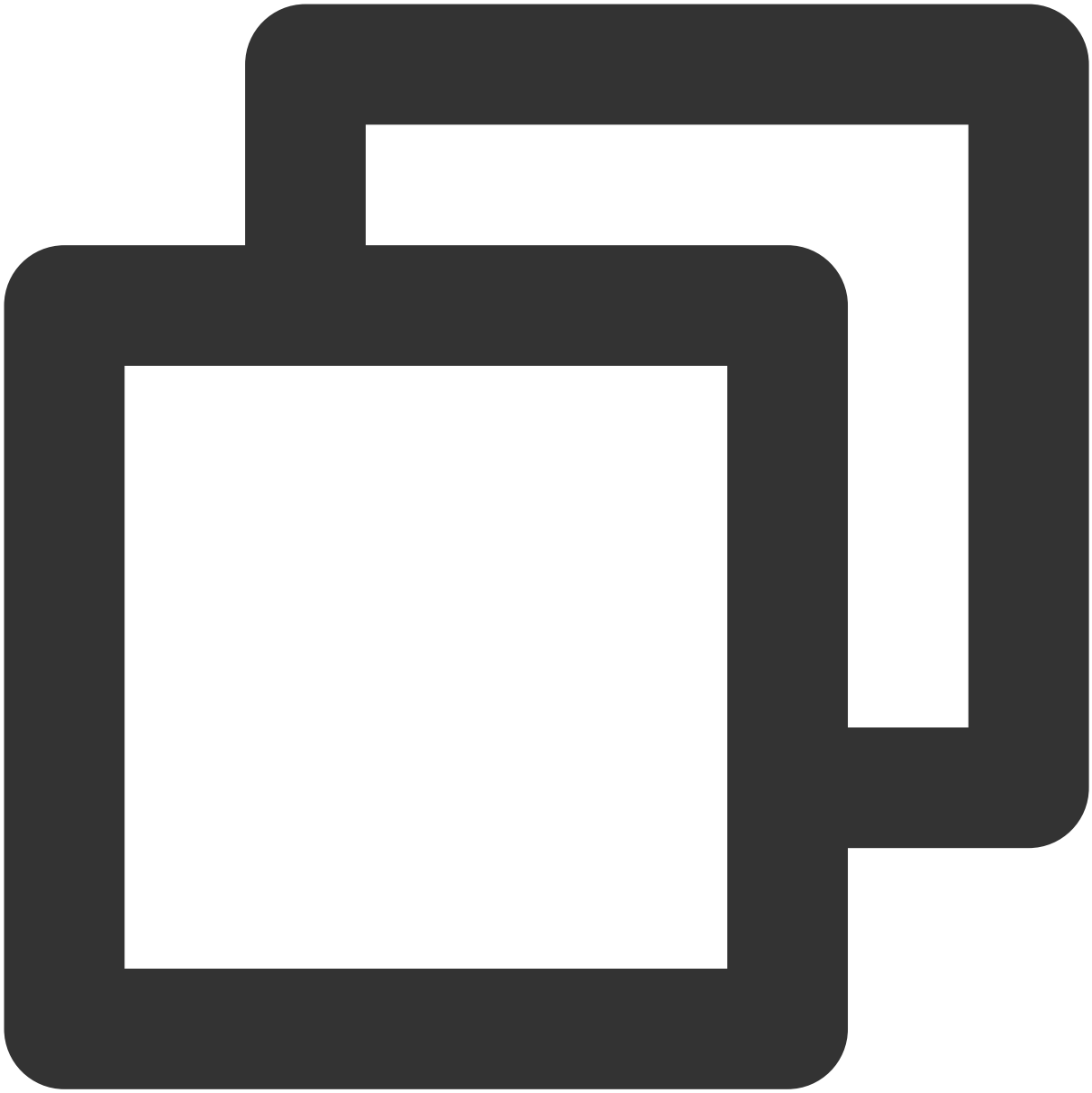
```
});

// 全員ミュート
V2TIMGroupInfo info = new V2TIMGroupInfo();
info.setGroupID("groupA");
info.setAllMuted(true);
V2TIMManager.getGroupManager().setGroupInfo(info, new V2TIMCallback() {
  @Override
  public void onSuccess() {
      // 全員ミュートに成功
  }

  @Override
  public void onError(int code, String desc) {
      // 全員ミュートに失敗
  }
});

V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
  @Override
  public void onMemberInfoChanged(String groupID, List<V2TIMGroupMemberChangeInfo>
    // グループメンバーミュートをリッスン
    for (V2TIMGroupMemberChangeInfo memberChangeInfo : v2TIMGroupMemberChangeInfoLi
      // ミュートされたユーザーID
      String userID = memberChangeInfo.getUserID();
      // ミュート時間
      long muteTime = memberChangeInfo.getMuteTime();
    }
  }

  @Override
  public void onGroupInfoChanged(String groupID, List<V2TIMGroupChangeInfo> changeI
    // 全員ミュートをリッスン
    for (V2TIMGroupChangeInfo groupChangeInfo : changeInfos) {
      if (groupChangeInfo.getType() == V2TIMGroupChangeInfo.V2TIM_GROUP_INFO_CHANGE
        // 全員ミュートされるかどうか
        boolean isMuteAll = groupChangeInfo.getBoolValue();
      }
    }
  }
});
```



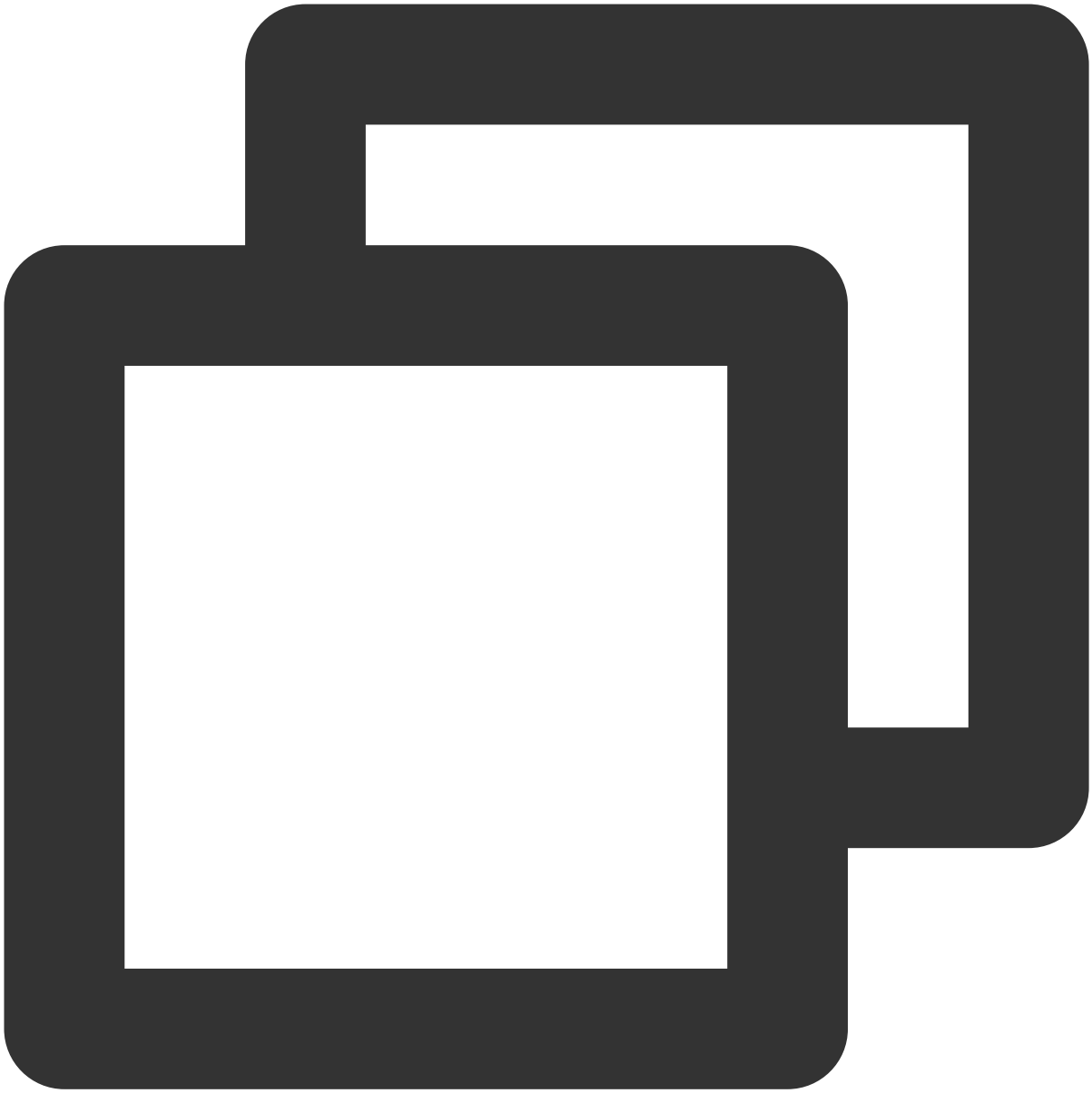
```
// グループメンバーuser1を1分間ミュート
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA" member:@"user1" muteTime:6
    // グループメンバーのミュートに成功
} fail:^(int code, NSString *desc) {
    // グループメンバーのミュートに失敗
}];

// 全員ミュート
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"groupA";
info.allMuted = YES;
```

```
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA" member:@"user1" muteTime:6
    // 全員ミュートに成功
} fail:^(int code, NSString *desc) {
    // 全員ミュートに失敗
}];

[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberInfoChanged:(NSString *)groupID changeInfoList:(NSArray <V2TIMGroup
    // グループメンバーミュートをリッスン
    for (V2TIMGroupMemberChangeInfo *memberChangeInfo in changeInfoList) {
      // ミュートされたユーザーID
      NSString *userID = memberChangeInfo.userID;
      // ミュート時間
        uint32_t muteTime = memberChangeInfo.muteTime;
    }
}

- (void)onGroupInfoChanged:(NSString *)groupID changeInfoList:(NSArray <V2TIMGroupC
    // 全員ミュートをリッスン
    for (V2TIMGroupChangeInfo groupChangeInfo in changeInfoList) {
      if (groupChangeInfo.type == V2TIM_GROUP_INFO_CHANGE_TYPE_SHUT_UP_ALL) {
        // 全員ミュートされるかどうか
        BOOL isMuteAll = groupChangeInfo.boolValue;
      }
    }
}
```

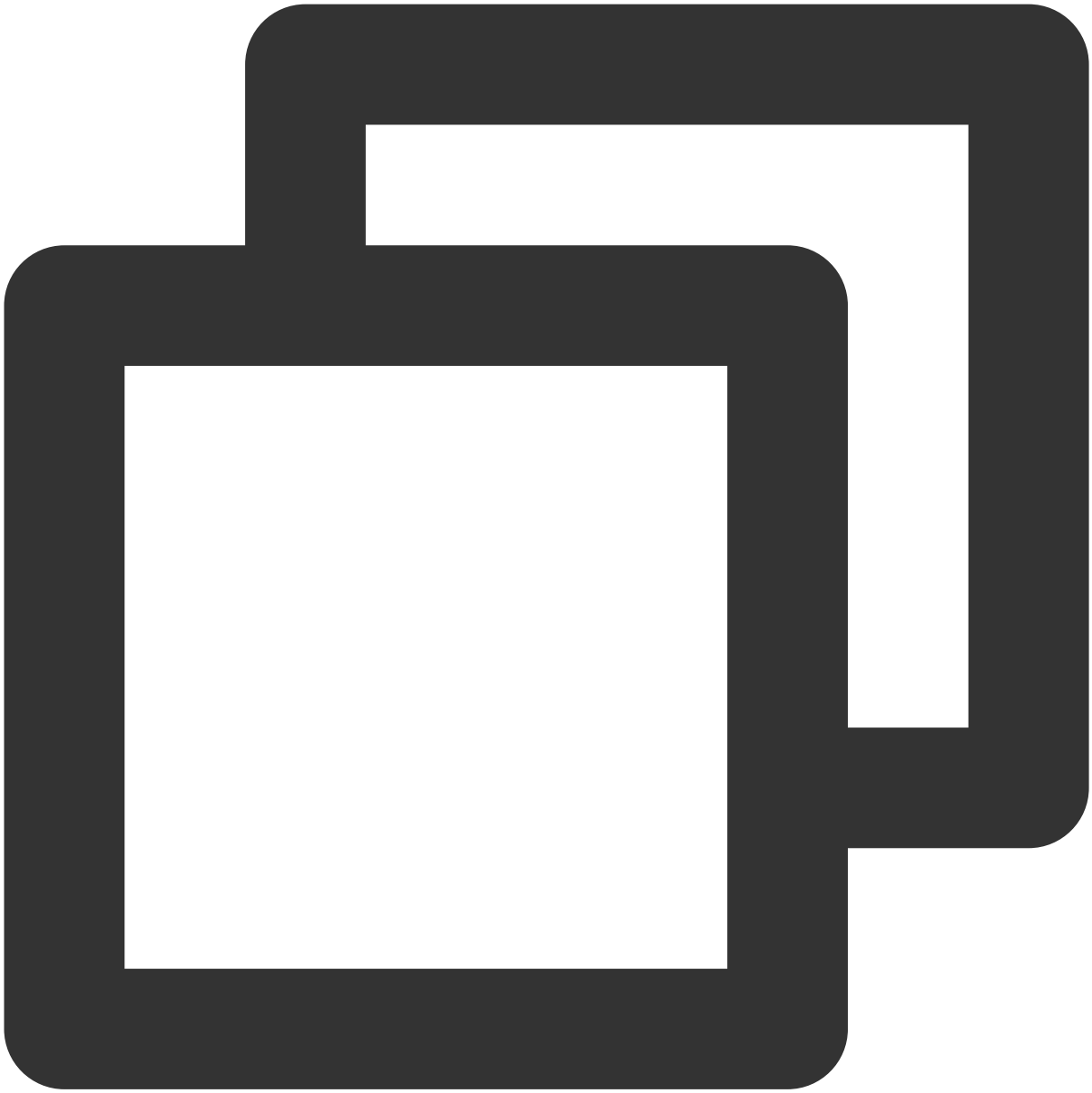


```
// グループメンバーuserBを10分間ミュート
groupManager.muteGroupMember(groupID: '',userID: 'userB',seconds: 10);

// 全員ミュート
groupManager.setGroupInfo(info: V2TimGroupInfo(isAllMuted: true,groupID: '',groupTy

TencentImSDKPlugin.v2TIMManager.addGroupListener(listener: V2TimGroupListener(onMem
    //グループメンバー情報変更
  },
  onGroupInfoChanged: (groupID,info){
    // グループメッセージ変更
```

```
  }
  ));
```

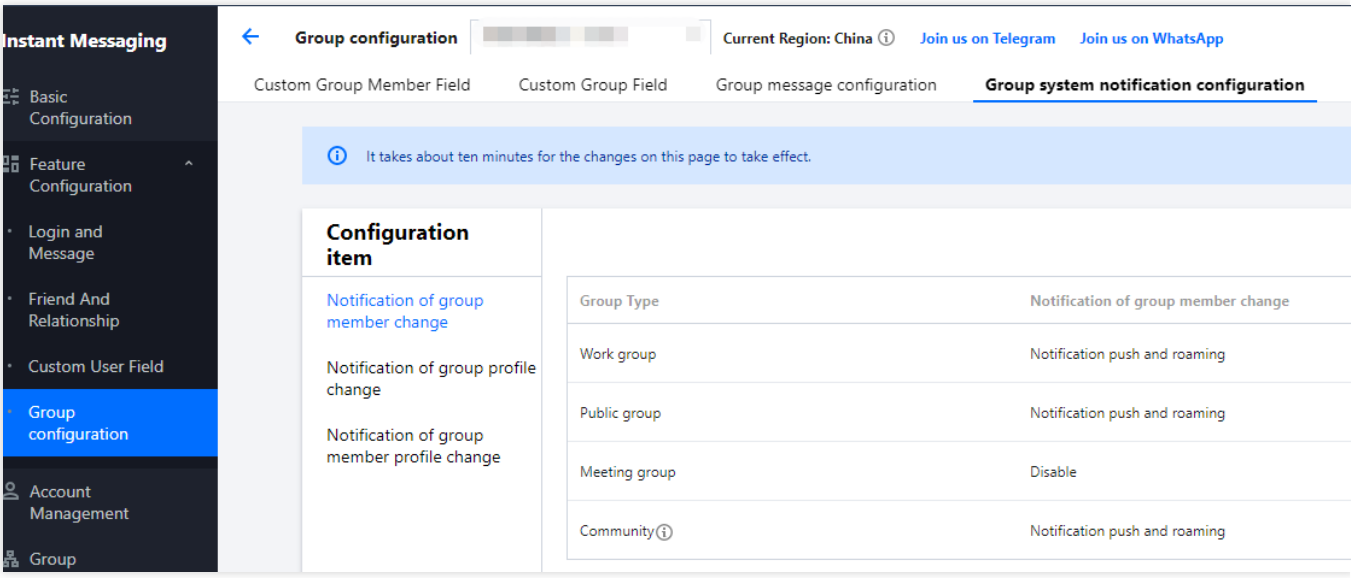


```
tim.setGroupMemberMuteTime(options);
let promise = tim.setGroupMemberMuteTime({
  groupID: 'group1',
  userID: 'user1',
  muteTime: 600 // 10分間ミュート。ミュートを解除するには0を設定
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group); // 変更後のグループデータ
```

```
  console.log(imResponse.data.group); // 変更後のグループメンバーのデータ
}).catch(function(imError) {
  console.warn('setGroupMemberMuteTime error:', imError); // ミュート失敗の関連情報
});
// トピック内のグループメンバーのミュート時間を設定
let promise = tim.setGroupMemberMuteTime({
  groupID: 'topicID',
  userID: 'user1',
  muteTime: 600 // 10分間ミュート。ミュートを解除するには0を設定
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group); // 変更後のグループデータ
  console.log(imResponse.data.group); // 変更後のグループメンバーのデータ
}).catch(function(imError) {
  console.warn('setGroupMemberMuteTime error:', imError); // ミュート失敗の関連情報
});
// v2.6.2からは、全員ミュートおよびミュート解除機能が利用できます。現在はグループ全員をミュートにす
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  muteAllMembers: true, // trueは全員ミュート、falseは全員ミュートの解除を表します
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 変更後のグループの詳細データ
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // グループ情報変更失敗の関連情報
});
```

グループ監視のその他のSDKバージョンコードの例

グループメンバーのミュート状態の変更は、デフォルトではクライアントに通知されず、コンソールで設定しなければならないことに注意が必要です。



説明：

クライアントSDKは現在、ライブルームでのミュートをサポートしていません。サーバー側APIを介してミュートとミュート解除を実行することができます。

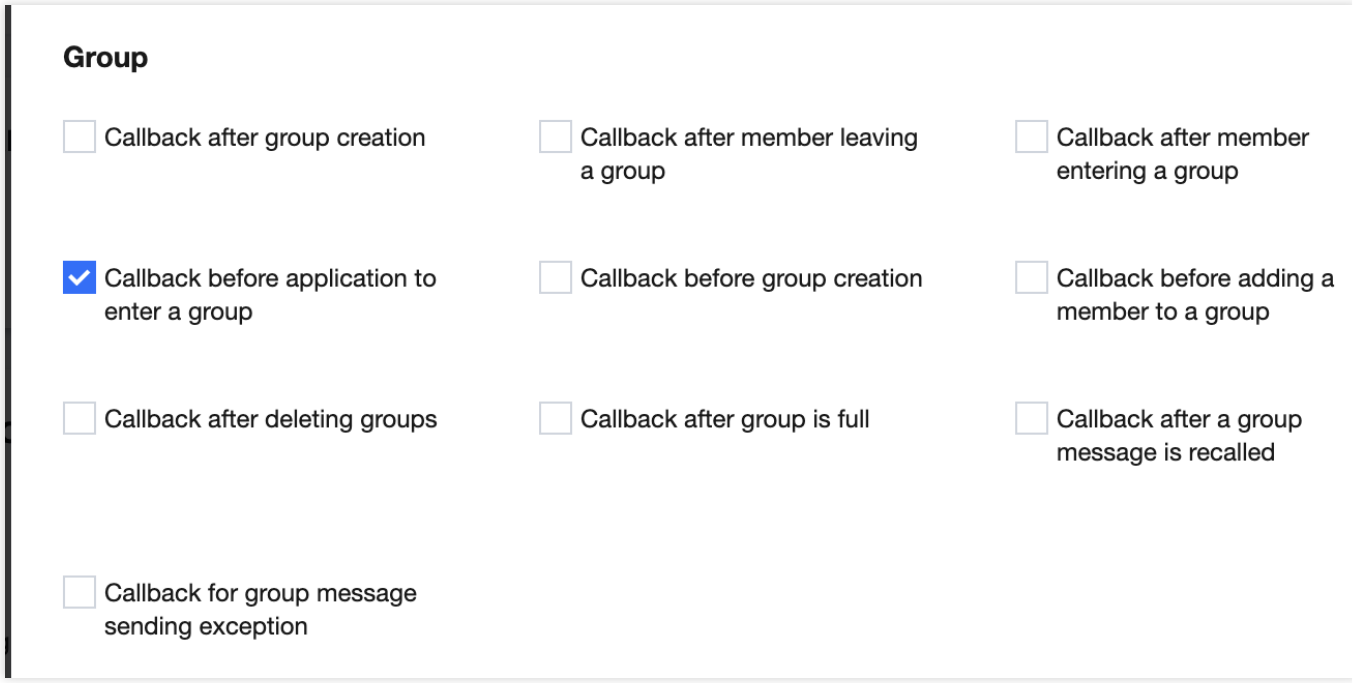
ライブルームからの強制退室

ライブルームのミュートと同様に、開発者はサーバーのグループ強制退出インターフェースを通じて、グループメンバーを強制退出させることができます。ただし、AVChatRoomはこのインターフェースをサポートしていないため、他の方法で実現する必要があります。

方法は次のとおりです。

1. バックエンドがグループカスタムメッセージを送信し、カスタムメッセージの内容を強制退出ユーザーのuserIDにします。一度に強制退出させる人数が多すぎる場合はメッセージボディのサイズに注意し、何度かに分けて送信します。
2. 業務バックエンドでグループ参加ブラックリストを保守します
3. クライアントがカスタムメッセージを受信して解析し、userIDに自身のuserIDが含まれる場合は自主的にグループ退出インターフェースを呼び出します
4. ユーザーのグループ参加前コールバックを設定し、現在グループに参加しようとしているユーザーがステップ2のブラックリストにないかどうかをチェックします。

コンソールの関連設定は図のとおりです：



ご注意：

クライアントSDK6.6.X以降およびFlutter SDK4.1.1以降では、ライブルームからの強制退室をサポートします

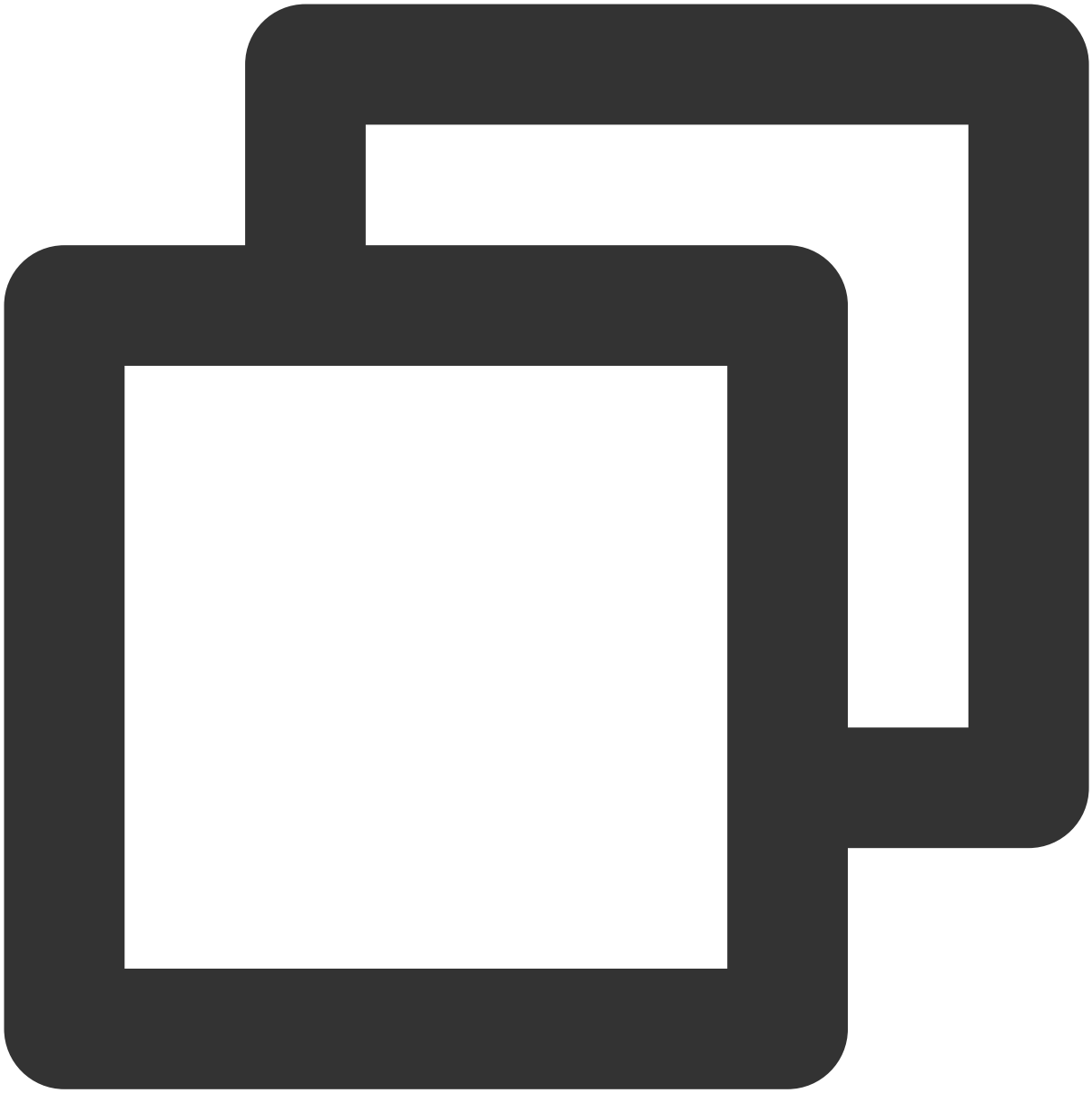
以下のコードを参照できます：

Android

iOS&Mac

Flutter

Web

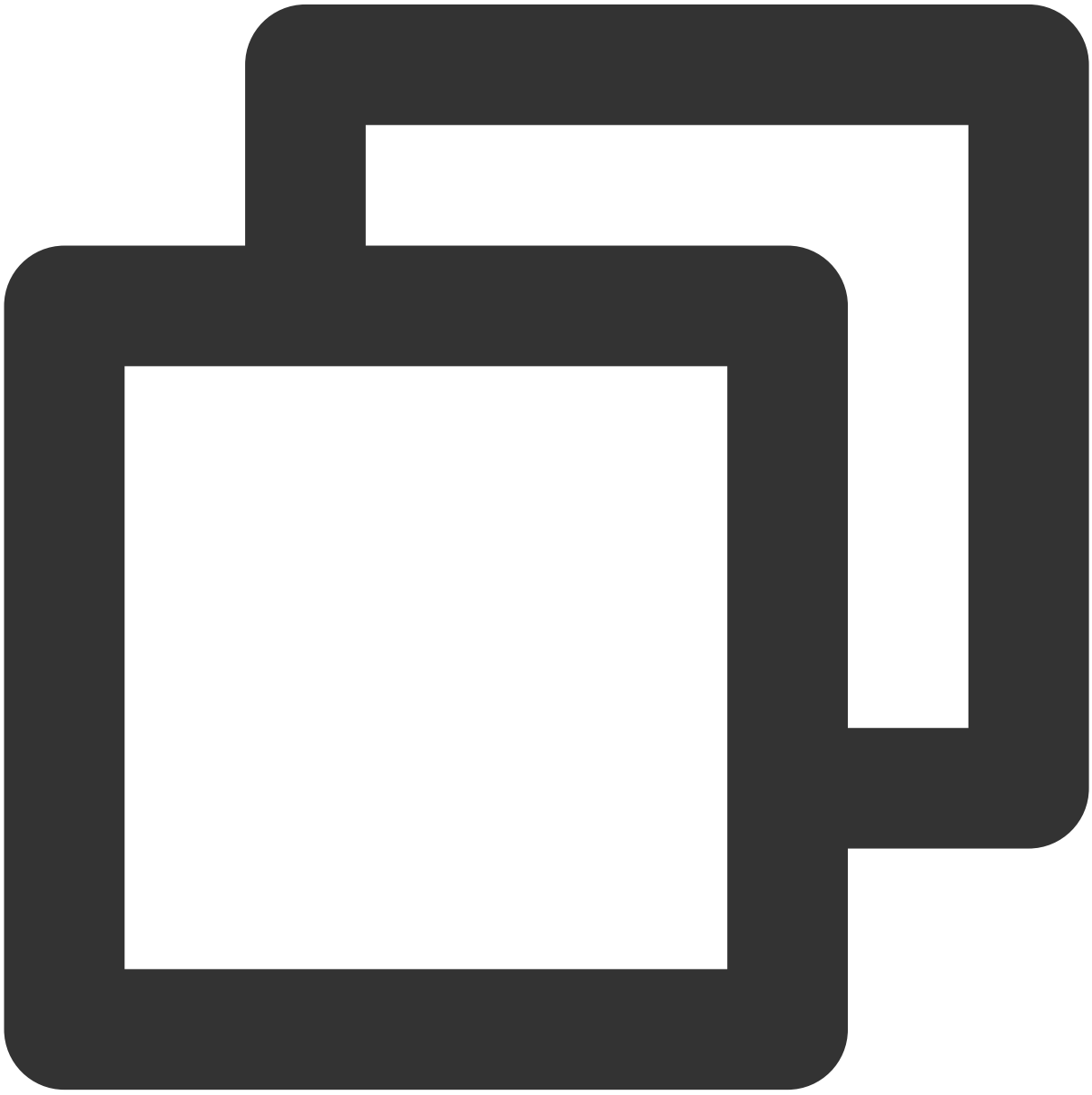


```
List<String> userIDList = new ArrayList<>();
userIDList.add("userB");
V2TIMManager.getGroupManager().kickGroupMember("groupA", userIDList, "", new V2TIMV
  @Override
  public void onSuccess(List<V2TIMGroupMemberOperationResult> v2TIMGroupMemberOpera
      // 強制退室に成功
  }

  @Override
  public void onError(int code, String desc) {
      // 強制退室に失敗
```

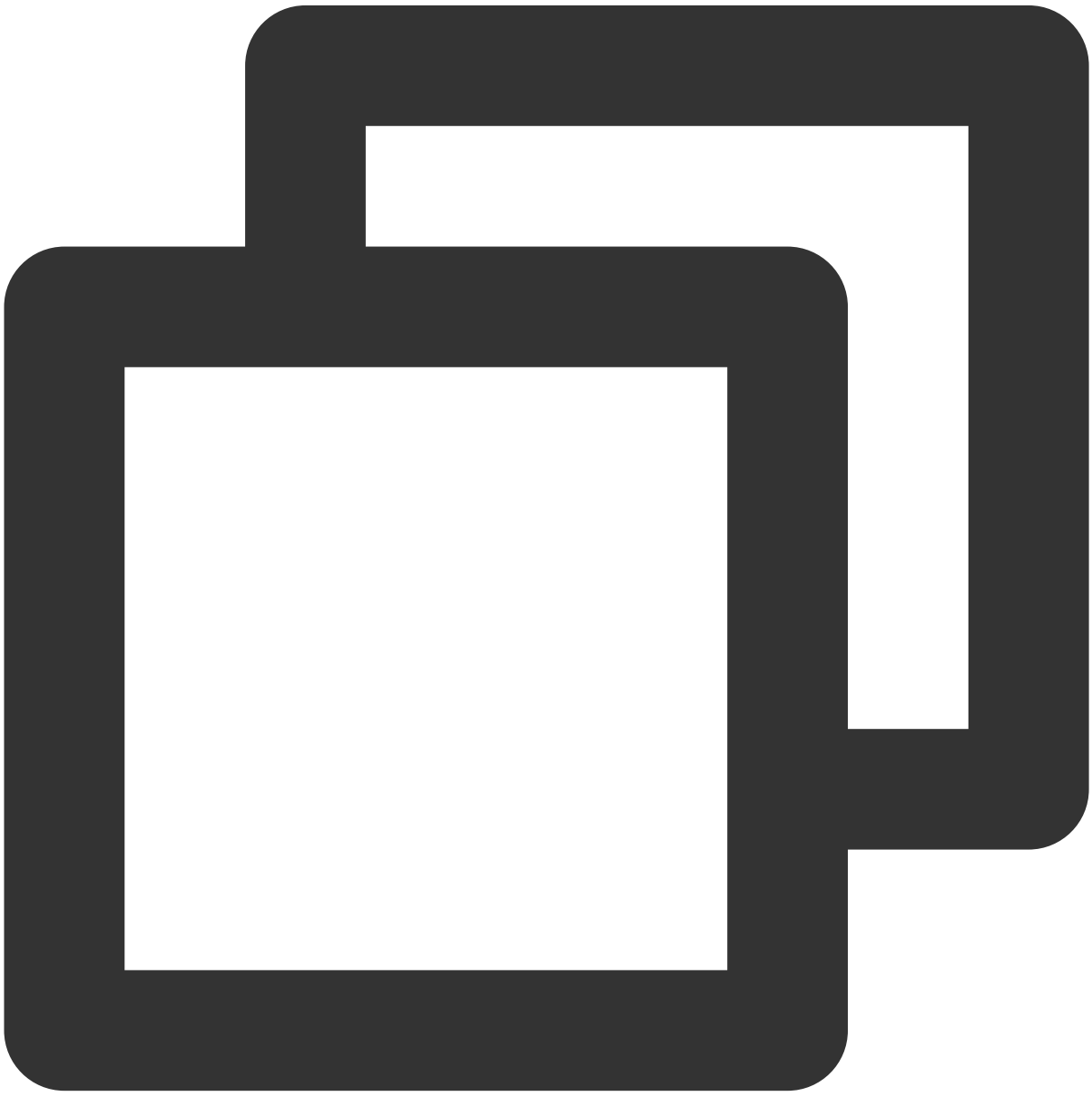
```
    }
});

V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
    @Override
    public void onMemberKicked(String groupID, V2TIMGroupMemberInfo opUser,
    List<V2TIMGroupMemberInfo> memberList) {
        // グループメンバー強制退室の通知
    }
});
```

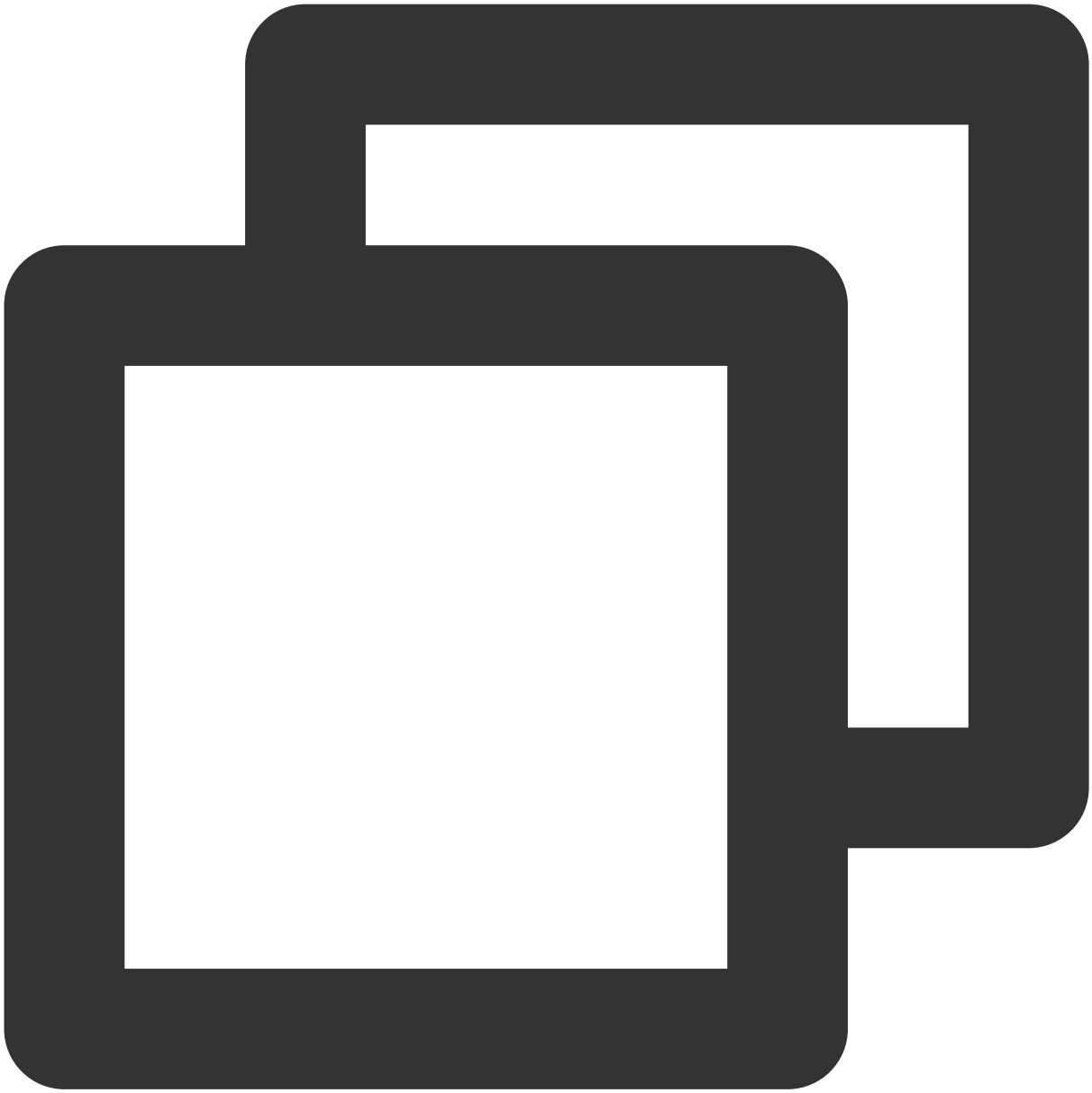


```
[[V2TIMManager sharedInstance] kickGroupMember:@"groupA" memberList:@[@"user1"] rea
    // 強制退室に成功
} fail:^(int code, NSString *desc) {
    // 強制退室に失敗
}];

[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberKicked:(NSString *)groupID opUser:(V2TIMGroupMemberInfo *)opUser me
    // グループメンバー強制退室の通知
}
```



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



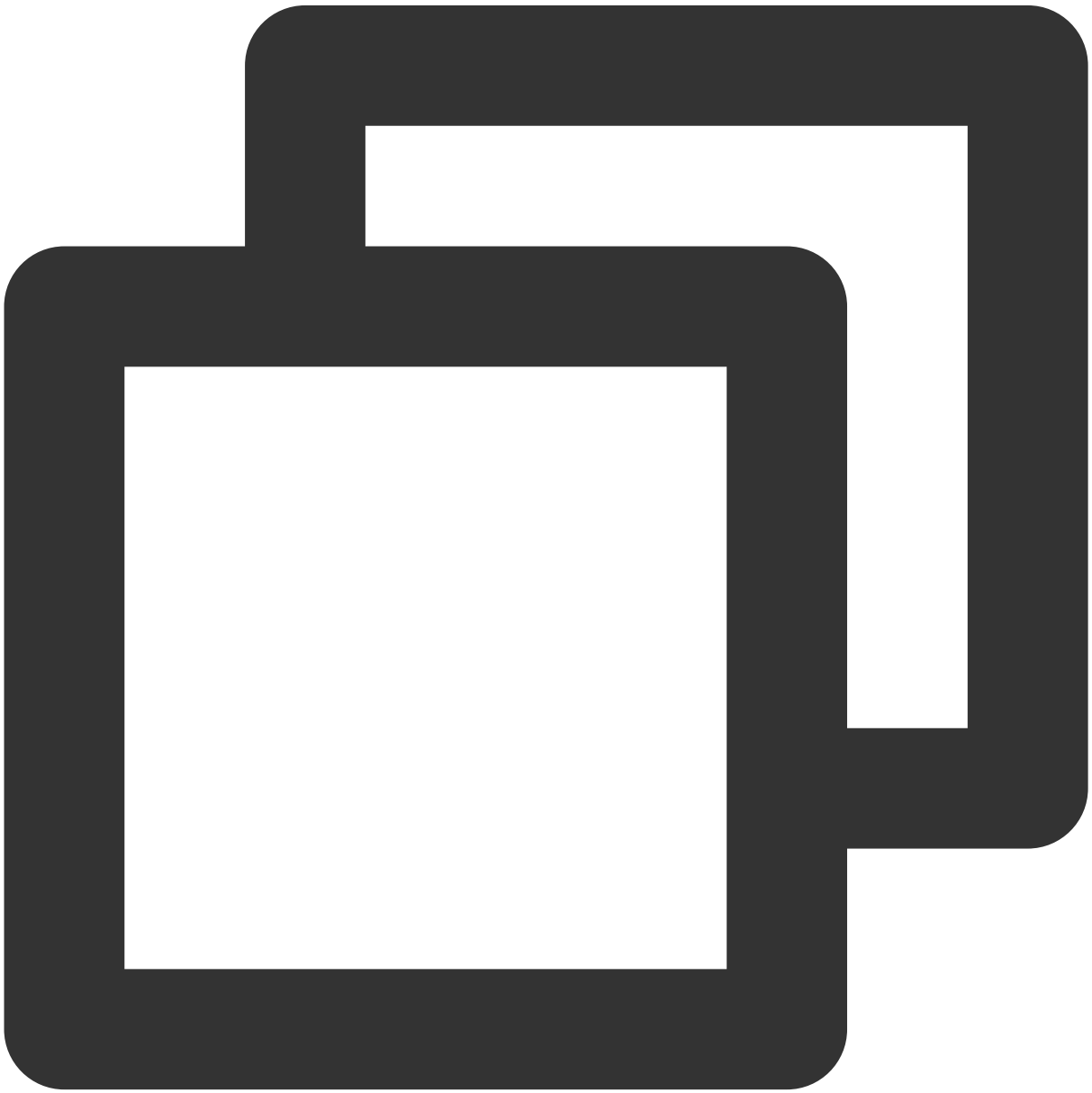
```
tim.deleteGroupMember(options);
```

ライブルームのセンシティブコンテンツフィルタリング

ライブルームのセンシティブコンテンツをフィルタリングすることも、ライブストリーミング業務の非常に重要な機能です。実現方法は次のとおりです：

1. ユーザーグループにメッセージ送信前コールバックをバインドします

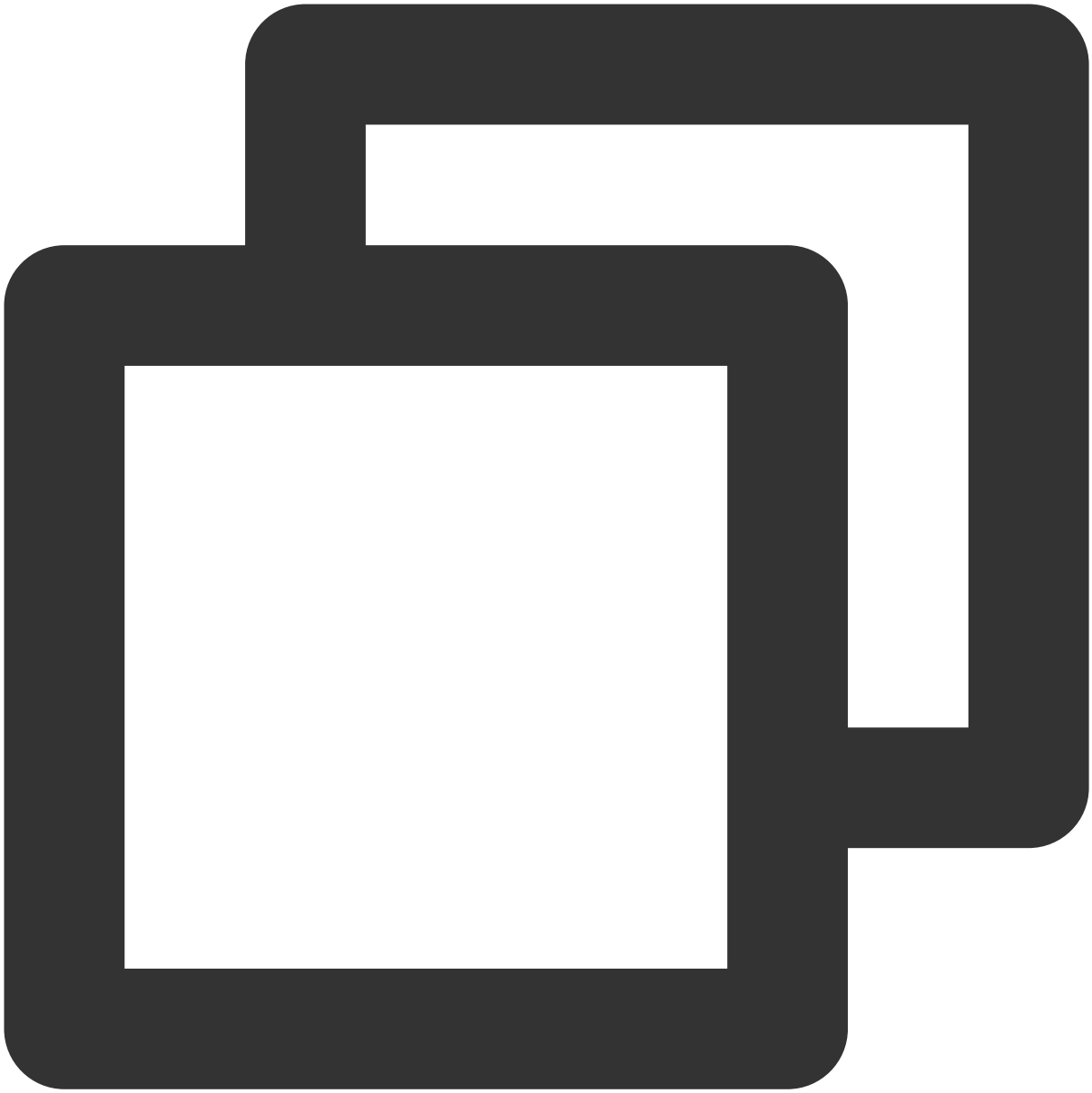
2. コールバックデータからメッセージタイプを判断し、メッセージデータを天御またはその他のサードパーティチェックサービスに転送します
3. メッセージが一般テキストメッセージの場合は、天御のチェックを待って同期的に返すことができ、メッセージを送信するかどうかのデータパケットをIMバックエンドに返します
4. メッセージがマルチメディアメッセージの場合は、メッセージ送信のデータパケットを直接IMバックエンドに返すことができます。さらに天御から非同期的に返された結果を返した後、メッセージに違反が発見された場合は、メッセージ編集インターフェースまたはグループカスタムメッセージインターフェースを通じてメッセージ変更通知を送信することができます。クライアントは通知を受信した後、違反メッセージをブロックします。
メッセージ送信前コールバックデータの例です。



```
{
    "CallbackCommand": "Group.CallbackBeforeSendMsg", // コールバックコマンド
    "GroupId": "@TGS#2J4SZEAEL", // グループID
    "Type": "Public", // グループタイプ
    "From_Account": "jared", // 送信者
    "Operator_Account":"admin", // リクエストの送信者
    "Random": 123456, // 乱数
    "OnlineOnlyFlag": 1, //オンラインメッセージは1、そうでない場合は0となります。ライブストリー
    "MsgBody": [ // メッセージボディです。TIMMessageのメッセージオブジェクトをご参照ください
        {
            "MsgType": "TIMTextElem", // テキスト
            "MsgContent": {
                "Text": "red packet"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data"
}
```

開発者はMsgBody内のMsgTypeフィールドによってメッセージタイプを判断することができます。フィールド全体の説明についてはコールバックドキュメントをご参照ください。

開発者は違法なメッセージに対して異なる処理を選択することができ、コールバック内のリターンパケットによってIMバックエンドで制御することができます。

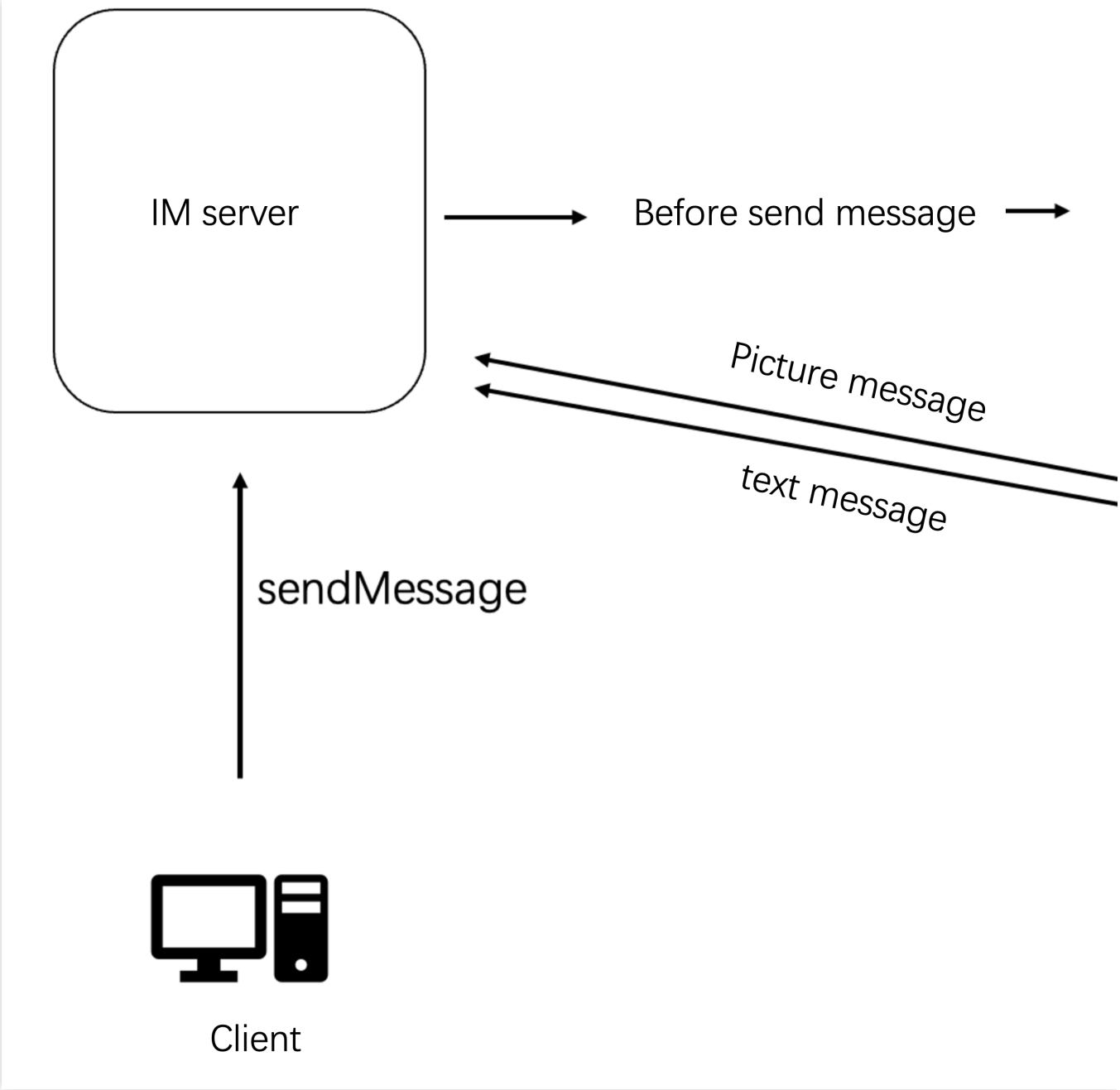


```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // ErrorCodeごとに意味が異なります
}
```

ErrorCode	意味
0	発言を許可し、メッセージは正常に送信されます
1	発言を拒否し、クライアントは10016を返します

2	サイレントで破棄し、クライアントは正常を返します

開発者は自身の業務ニーズに応じて選択し、使用できます
センシティブコンテンツチェックのフローチャートは次のとおりです。



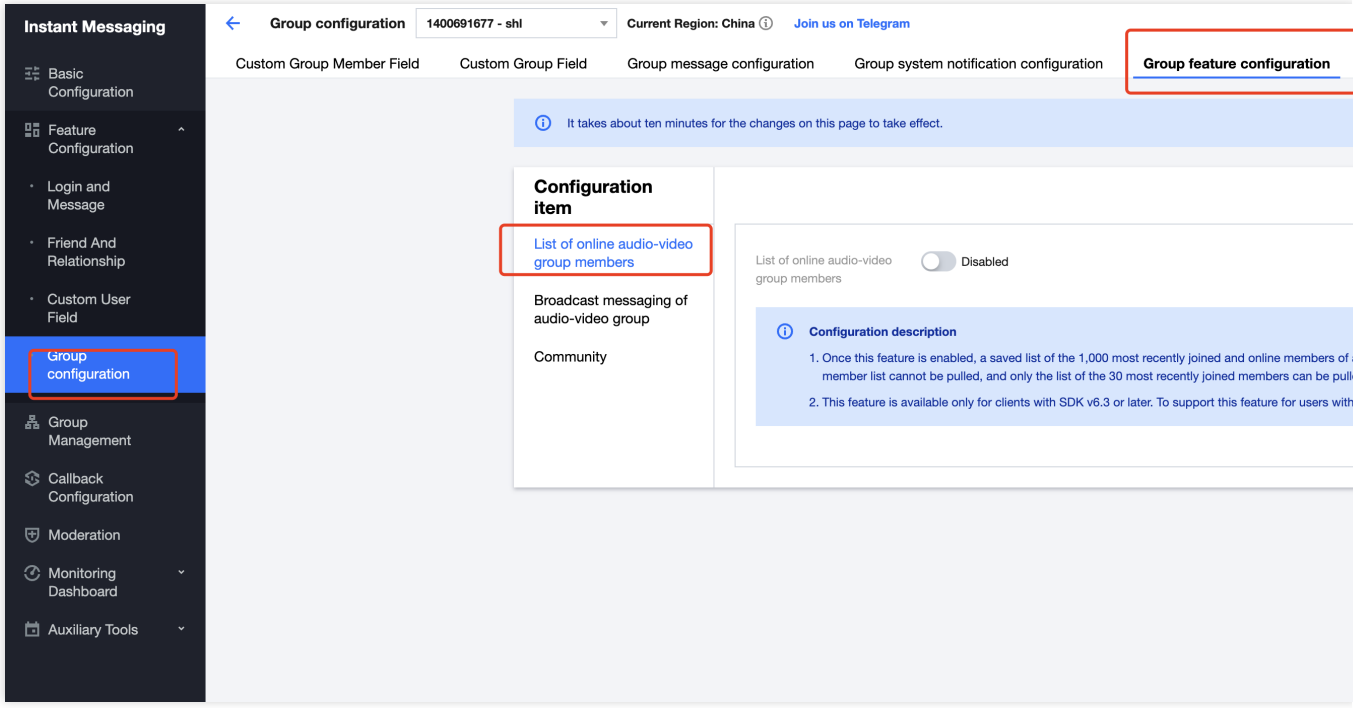
ライブルームのグループメンバーリスト

ライブルームにオンラインのグループメンバーリストを表示する必要がある場合
は、 `getGroupMemberList` インターフェースによってグループメンバーリストを取得することができます

が、AVChatRoomでは人数が多いため、全量のメンバーリストを取得する機能は提供していません。機能はフラッグシップ版か非フラッグシップ版かによって異なります。

1. フラッグシップ版以外のお客様は、`getGroupMemberList`を呼び出して、直近でグループに参加した30名のグループメンバーを取得することができます。
2. フラッグシップ版のお客様は、`getGroupMemberList`を呼び出して、直近でグループに参加した1000名のグループメンバーを取得することができます。この機能はIMコンソールで有効化する必要があり、有効化しなければデフォルトで非フラッグシップ版と同様に、直近でグループに参加した30名のグループメンバーのみを取得します。

コンソールの設定は図2.6のとおりです：



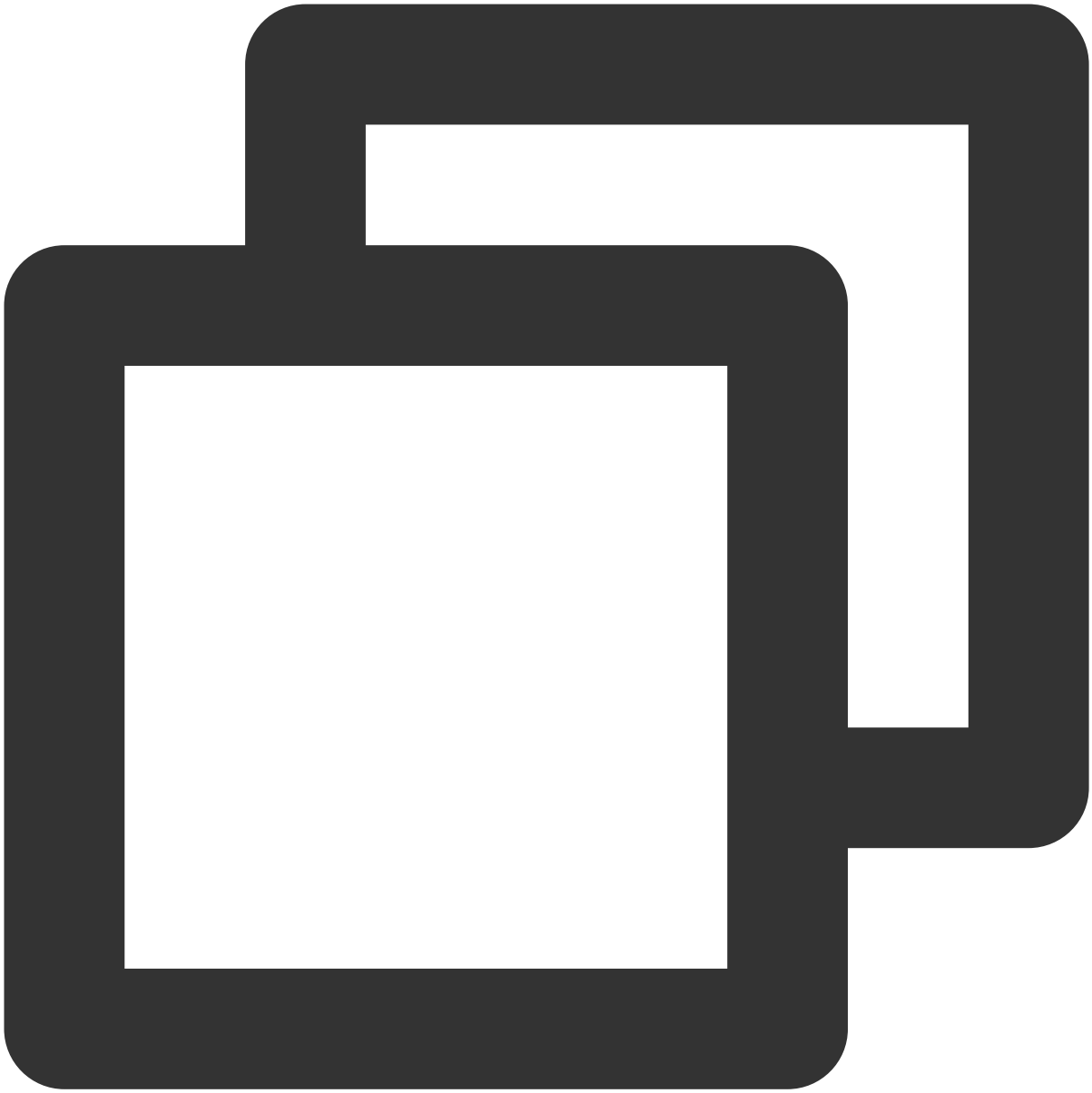
フラッグシップ版以外の場合、開発者は`getGroupMemberList`とグループ監視中のonGroupMemberEnterおよびonGroupMemberQuitコールバックによって、現在オンラインのグループメンバーリストをクライアントで保守することもできます。ただし、この方法では、ユーザーがライブルームから退出して入り直した場合も、最新の30名のグループメンバーしか取得できません。

Android

iOS&Mac

Flutter

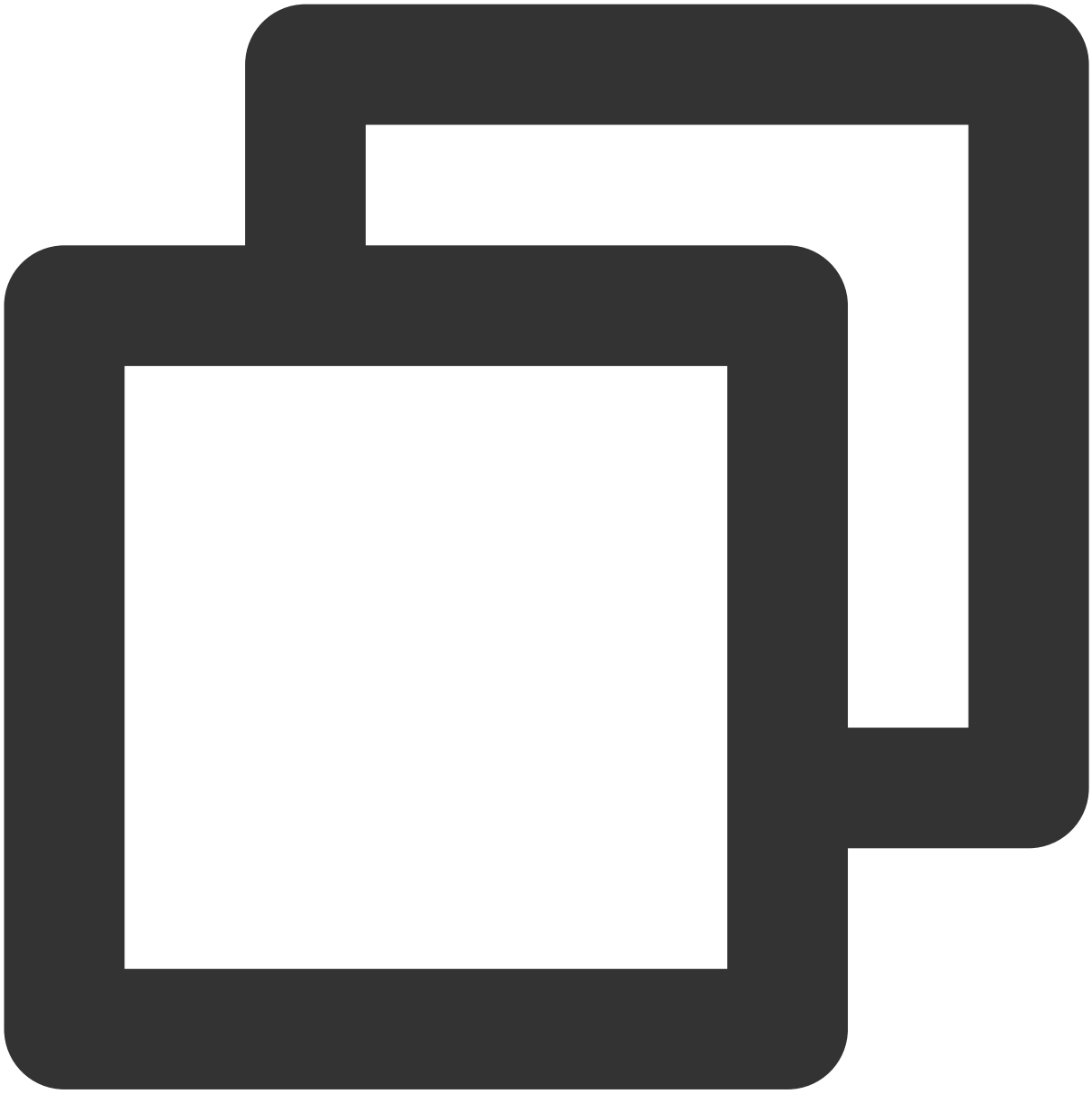
Web



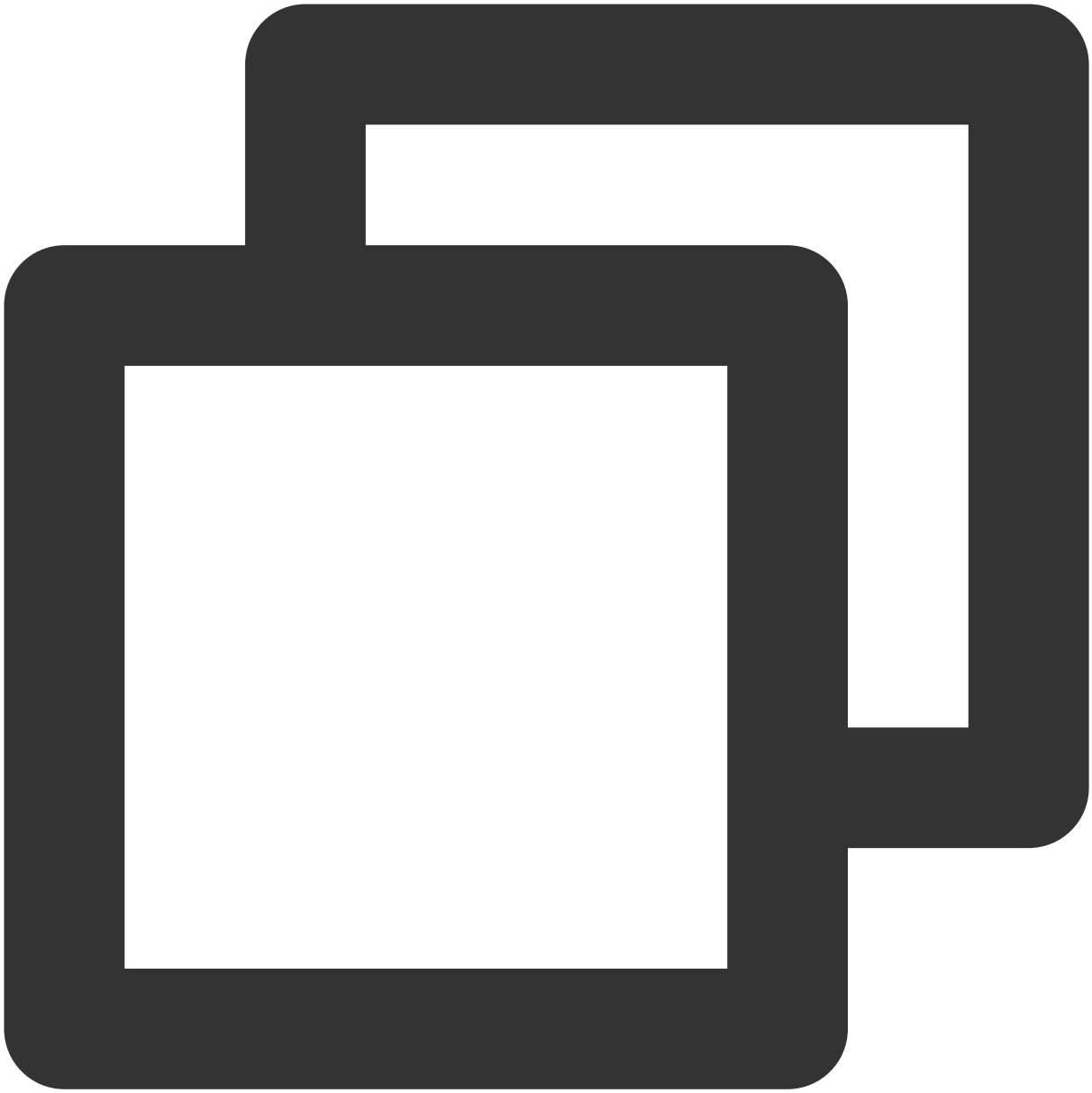
```
// filterパラメータを指定することで、グループ所有者のデータのみを取得します
int role = V2TIMGroupMemberFullInfo.V2TIM_GROUP_MEMBER_FILTER_OWNER;
V2TIMManager.getGroupManager().getGroupMemberList("testGroup", role, 0,
    new V2TIMValueCallback<V2TIMGroupMemberInfoResult>() {
    @Override
    public void onError(int code, String desc) {
        // 取得に失敗しました
    }

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

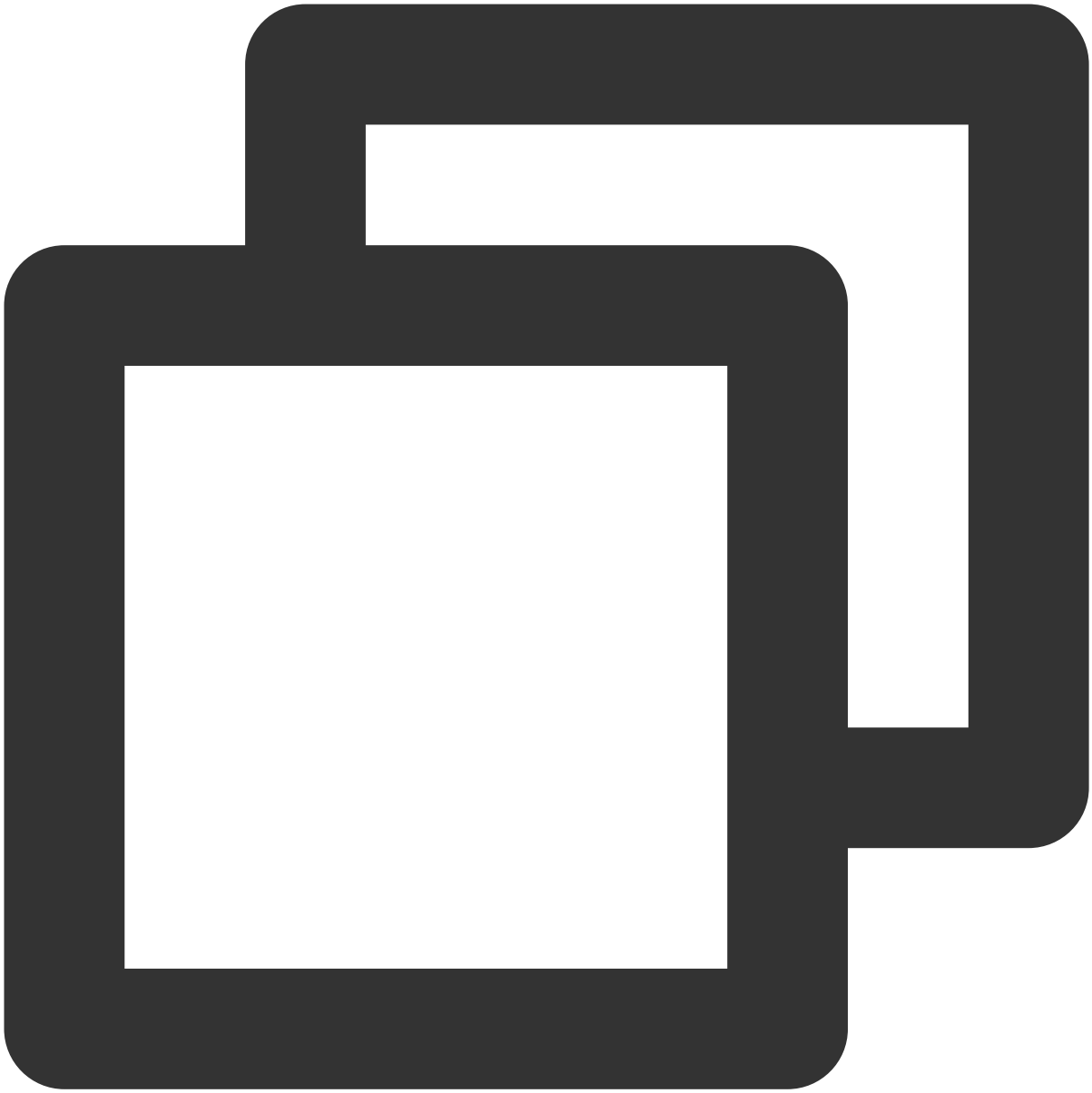
```
        // 取得に成功しました
    }
});
```



```
[[V2TIMManager sharedInstance] getGroupMemberList:@"groupA" filter:V2TIM_GROUP_MEMB
    // 取得に成功しました
} fail:^(int code, NSString *desc) {
    // 取得に失敗しました
}];
```



```
// filterパラメータを指定することで、グループ所有者のデータのみを取得します。ALLを指定することで、
groupManager.getGroupMemberList(count: 10,filter: GroupMemberFilterTypeEnum.V2TIM_G
```



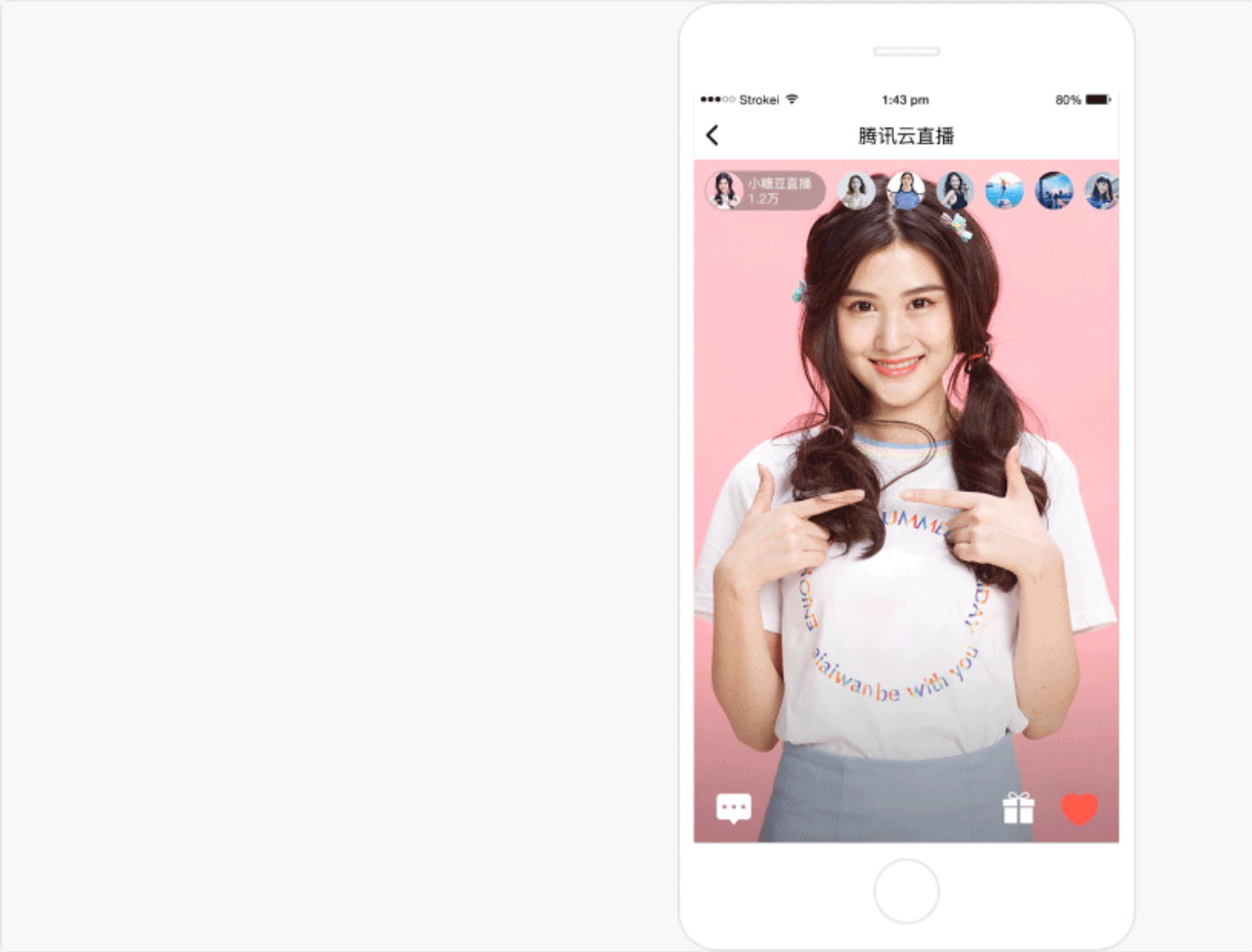
```
tim.getGroupMemberList(options);
```

ライブルームの弾幕抽選

ライブストリーミング抽選はメッセージ統計に似ており、どちらもメッセージ送信後コールバックを使用する必要があります。メッセージの内容をチェックして、抽選キーワードにヒットしたユーザーを抽選プールに加えます。ここではこれ以上の補足説明は行いません。

ライブストリーミング弾幕

AVChatRoomは、弾幕、ギフト、「いいね」など複数のメッセージタイプをサポートし、優れたライブストリーミングチャットのインタラクティブな体験を手軽に構築できます。



ライブルーム内のブロードキャスト

ブロードキャスト機能はライブルームでのシステムからのお知らせ機能に相当しますが、お知らせとは異なり、ブロードキャスト機能はメッセージに分類されます。システム管理者がブロードキャストメッセージを送信すると、SDKAppID下のすべてのライブルームでこのメッセージを受信することができます。
ブロードキャスト機能は現在はフラッグシップ版のみの機能であり、コンソールでアクティブ化する必要があります。
業務バックエンドからのブロードキャストメッセージ送信については、ドキュメントライブストリーミンググループのブロードキャストメッセージをご参照ください。
説明：

開発者のバージョンがフラッグシップ版ではない場合は、サーバーからグループカスタムメッセージを送信して実現できます。

3、ライブストリーミングのオーディオビデオストリーム部分

キャスター側（プッシュ）

現在、Tencent Cloud CSSプッシュには、次のいくつかの方法があります：
1.1 クライアントプッシュにはOBSツールプッシュを使用できます
1.2 Web端末ではWebプッシュを使用できます
1.3 App端末ではモバイルライブストリーミングSDKプッシュを使用できます
説明：
キャスター側でのプッシュの前にはプッシュアドレスの設定が必要です。プッシュ設定またはアドレスジェネレーターを参照してアドレスを生成してください。

ユーザー側（プル）

ユーザー側でのCSSストリームの取得方法にも、次のようないくつかの方法があります。
1.1 PC端末ではVLCツールの再生を使用できます
1.2 Web端末ではPlayer+での再生を使用できます
1.3 App端末ではPlayer+での再生を使用できます
説明：
プルの前にも再生アドレスの設定が必要です。再生設定またはアドレスジェネレーターを参照してアドレスを生成してください。

Live Video Caster

ライブストリーミングシーンのより多くのニーズを満たすために、開発者はクラウドのLive Video Casterを使用してCSSストリームの処理を行うことができます。現在Live Video Casterは次の機能をサポートしています：
1.1 CSSウォーターマーク、テキスト、トランジション
1.2 VODからライブストリーミングへの変換
1.3 ライブストリーミング画質、ビットレート、解像度の設定
1.4 ライブストリーミングのレイアウト調整
1.5 CSSレコーディング
1.6 ライブストリーミング弾幕の追加
1.7 CSSストリームの監視

TRTCライブストリーミング

Tencent Real-Time Communication (TRTC)の使用によっても、同様にライブストリーミングの機能を実装することができます。CSSと比べて、TRTCは次のようなより高度な機能を備えています：

4、よくあるご質問

1. 自身がメッセージを送信する際、メッセージ送信状態の `Message.nick` と `Message.avatar` フィールドがどちらも空である場合、インターフェース上でニックネームとプロフィール画像を正常に表示するにはどうすればよいですか？

getUserInfoインターフェースによって、ご自身のユーザー情報のnickおよびavatarフィールドを、メッセージ送信のnickおよびavatarフィールドにすることができます。

2. TRTCとCSSはどのように選択すればよいです？

TRTCライブストリーミングは遅延が少なく、キャスターとのインタラクションをより便利に実現できますが、価格がやや高いです。CSSは遅延がわずかに大きいですが、価格は安いです。

3. メッセージが紛失するのはなぜですか？

メッセージの紛失原因として以下が考えられます：
ライブストリーミンググループは、40通/秒の頻度制限があります。メッセージ送信前のコールバックとメッセージ送信後のコールバックを介して判断できます。紛失したメッセージが、メッセージ送信前にコールバックを受信し、メッセージ送信後にコールバックを受信していない場合、このメッセージは制限を受けています。
よくあるご質問4を参考に、Web端末から退出したことによって、Android、iOS、PCで同期的な退出が生じているかどうかを判断することができます。
Webで問題が発生した場合は、使用しているSDKがV2.7.6より以前のバージョンかどうかを確認してください。V2.7.6より以前の場合は、最新版にアップグレードしてください。
上記の可能性をすでに排除している場合は、チケットを送信して弊社にご連絡ください。

4.ビデオとチャットを直接使用できるオープンソースのライブストリーミングコンポーネントはありますか？

はい、コードはオープンソースです。詳細については、 Tencent Cloud TWebLIVBコンポーネントをご参照ください。

5. お問い合わせ

Tencent Cloud IMテクノロジーコミュニケーショングループに参加するメリットは以下の通りです：
信頼できる技術サポートを得る
詳細な製品情報を入手できる
業界の有識者と密にコミュニケーションできる

Telegramコミュニケーショングループ：［クリックして参加](https://t.me/tencent_imsdk)

Live Chat Room

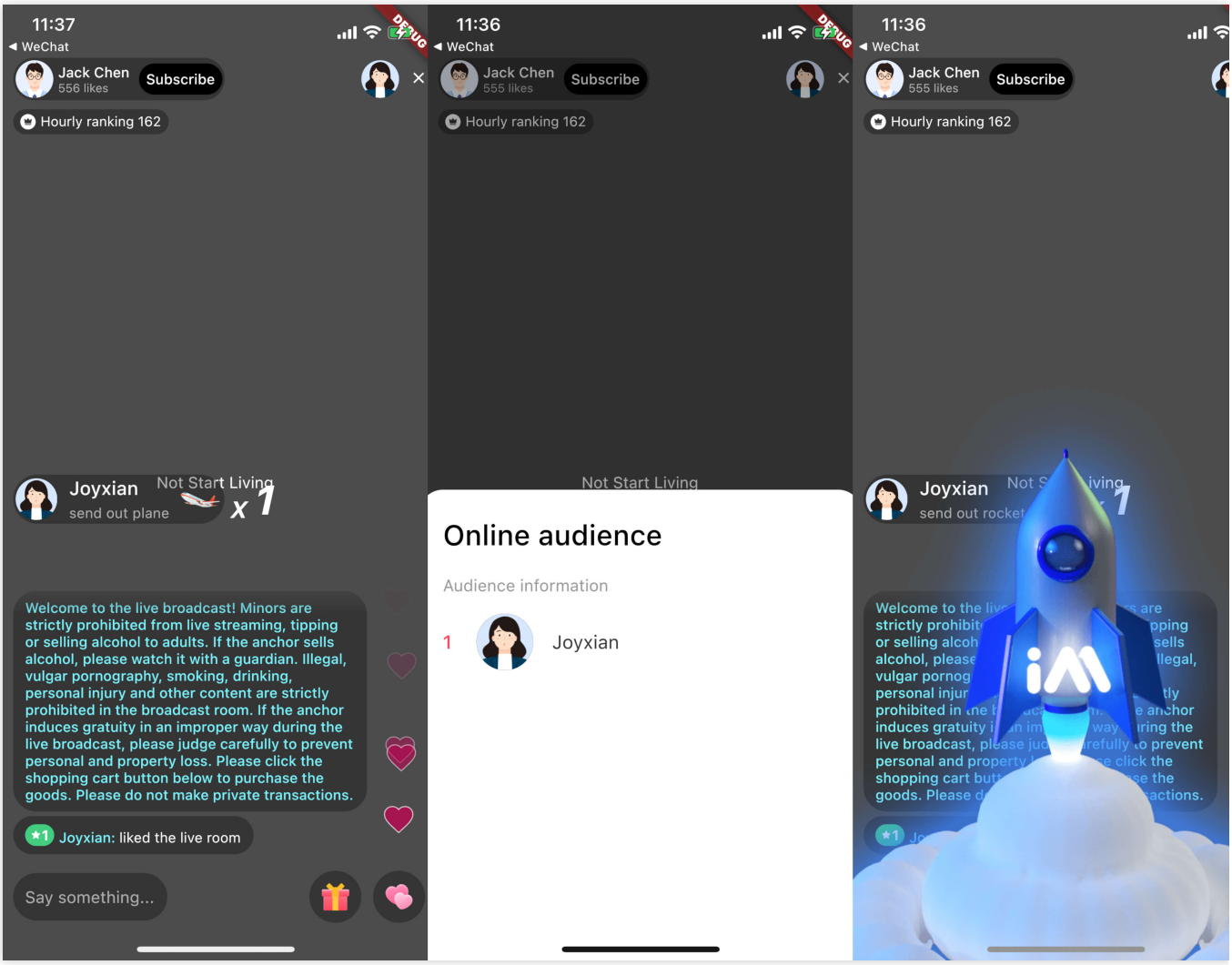
最終更新日：：2024-05-13 17:58:07

Foreword

This article is based on the Tencent Cloud AV Chat Room plug-in, combined with the Live Flutter SDK Realized live broadcast business scenario.

Tencent Cloud AV Chat Room

`Tencent Cloud AV Chat Room` is a UI component based on Tencent Cloud Chat SDK and implemented around chat interaction scenarios.
You can use this component to quickly implement an application with **tens of millions of interactions**.



Introduction

Preconditions

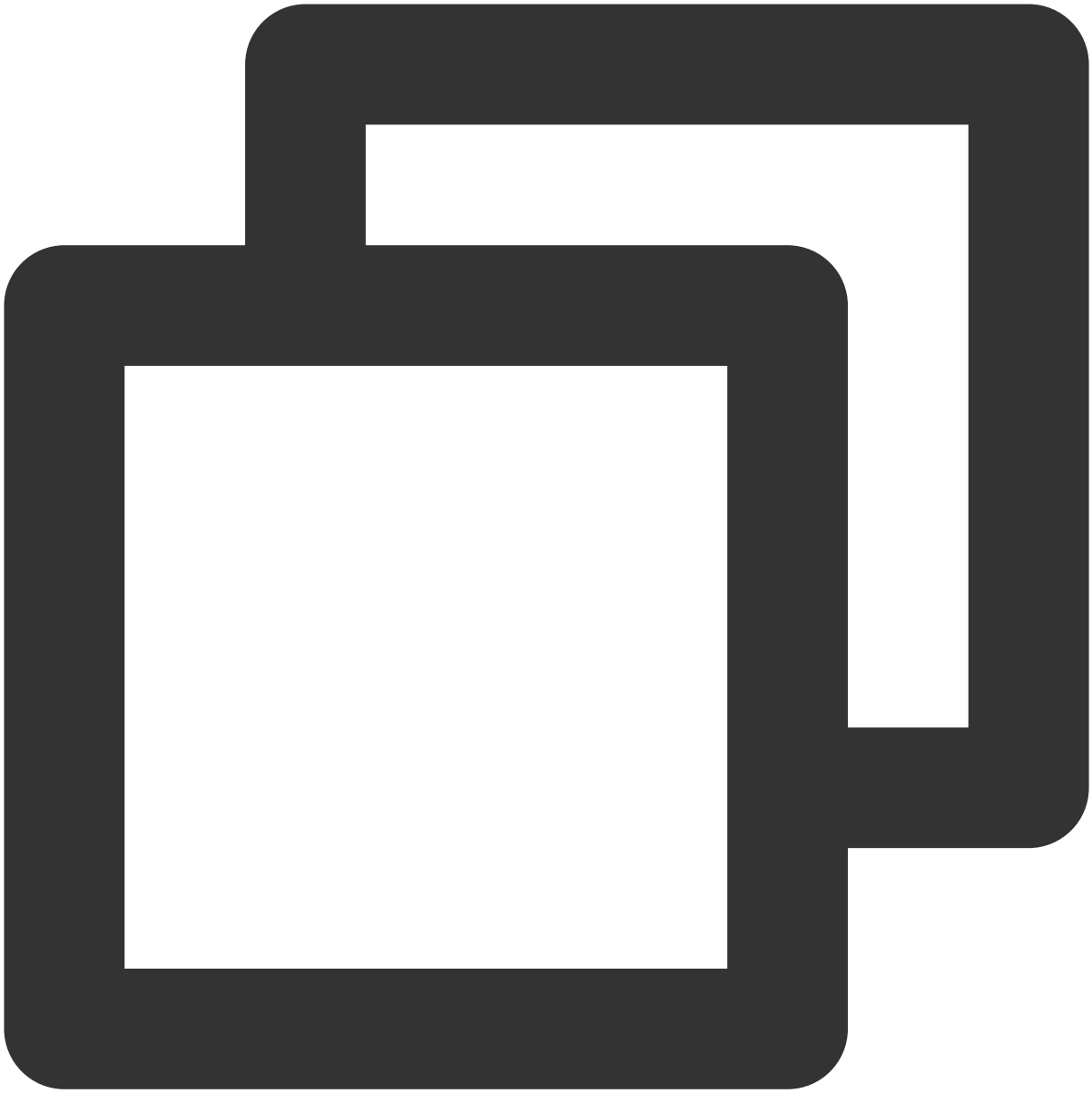
Have registered Tencent Cloud account and completed [identity verification] (https://intl.cloud.tencent.com/zh/document/ product/378/3629).

Create an application by referring to Create and upgrade an application, and record `SDKAppID` .

Select your application in IM Console, click Accessibility Tools -> UserSig Generation & Verification in the left navigation bar, and create a UserID and its corresponding `UserSig` , copy the three `UserID` , `Signature (Key)` , `UserSig` , which will be used in subsequent logins.

Create a Flutter app.

Add tencent_cloud_av_chat_room in the pubspec.yaml file under dependencies. Or execute the following command:



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

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

Step 1: Initialize and login to IM

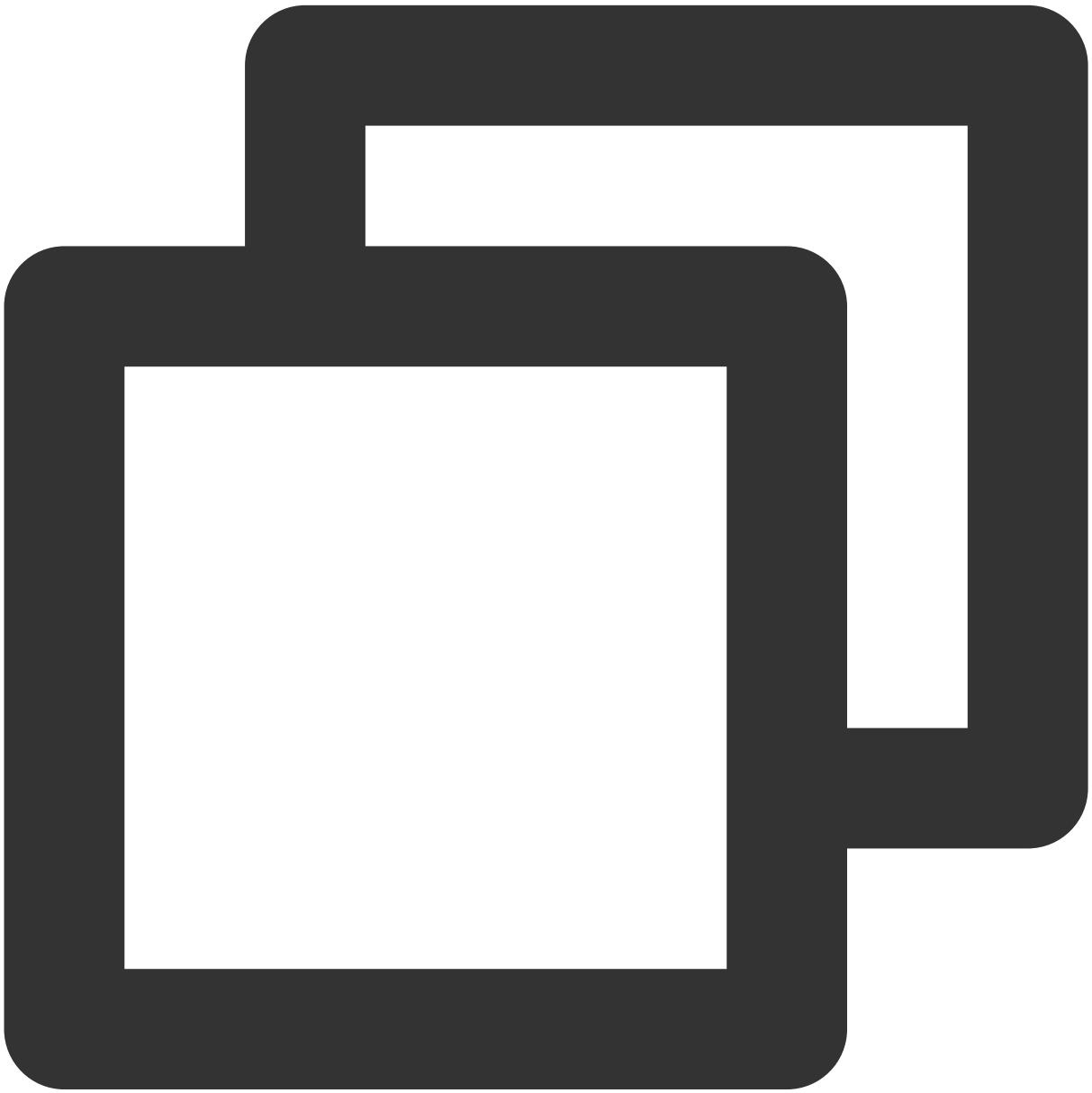
There are two ways to initialize and log in to IM:

External component: The entire application is initialized and logged in only once.

Inside the component: pass parameters into the component through configuration.
If you are integrating this plugin in an existing IM flutter project, you can skip this step.

Outside the component (recommended)

Initialize IM in the flutter app you created, note that the IM app only needs to be initialized once. This step can be skipped if integrating in an existing IM project.



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

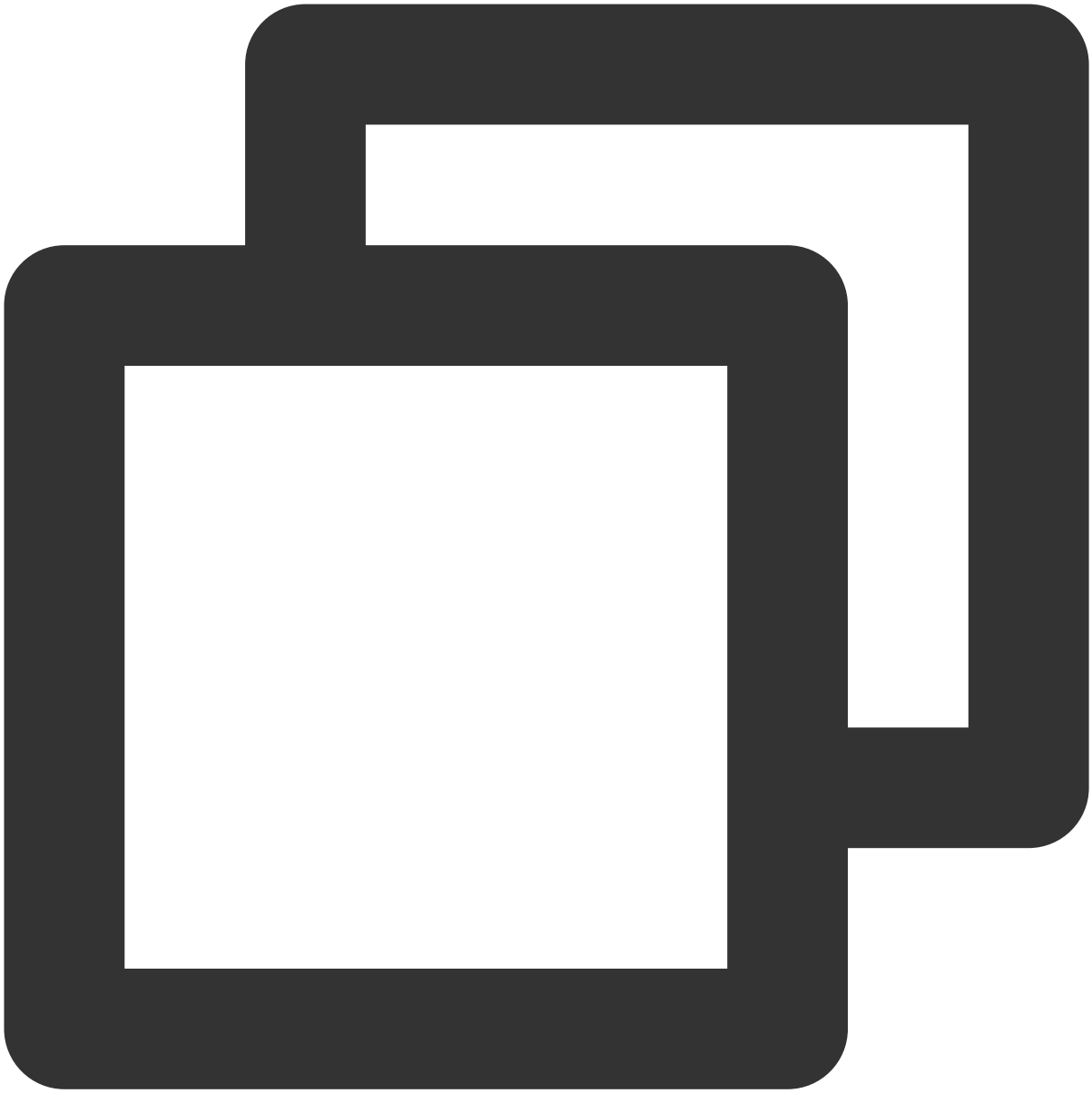
class _MyHomePageState extends State<MyHomePage> {
```

```
 final int _sdkAppID = 0; // SDKAppID of the IM application created in the prec
 final String _loginUserID = ""; // UserID in precondition
 final String _userSig = ""; // UserSig in preconditions
@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);
}
}
```

Inside the component

You can also pass `SDKAppID`, `UserSig`, `UserID` into the component through configuration to initialize and log in IM.



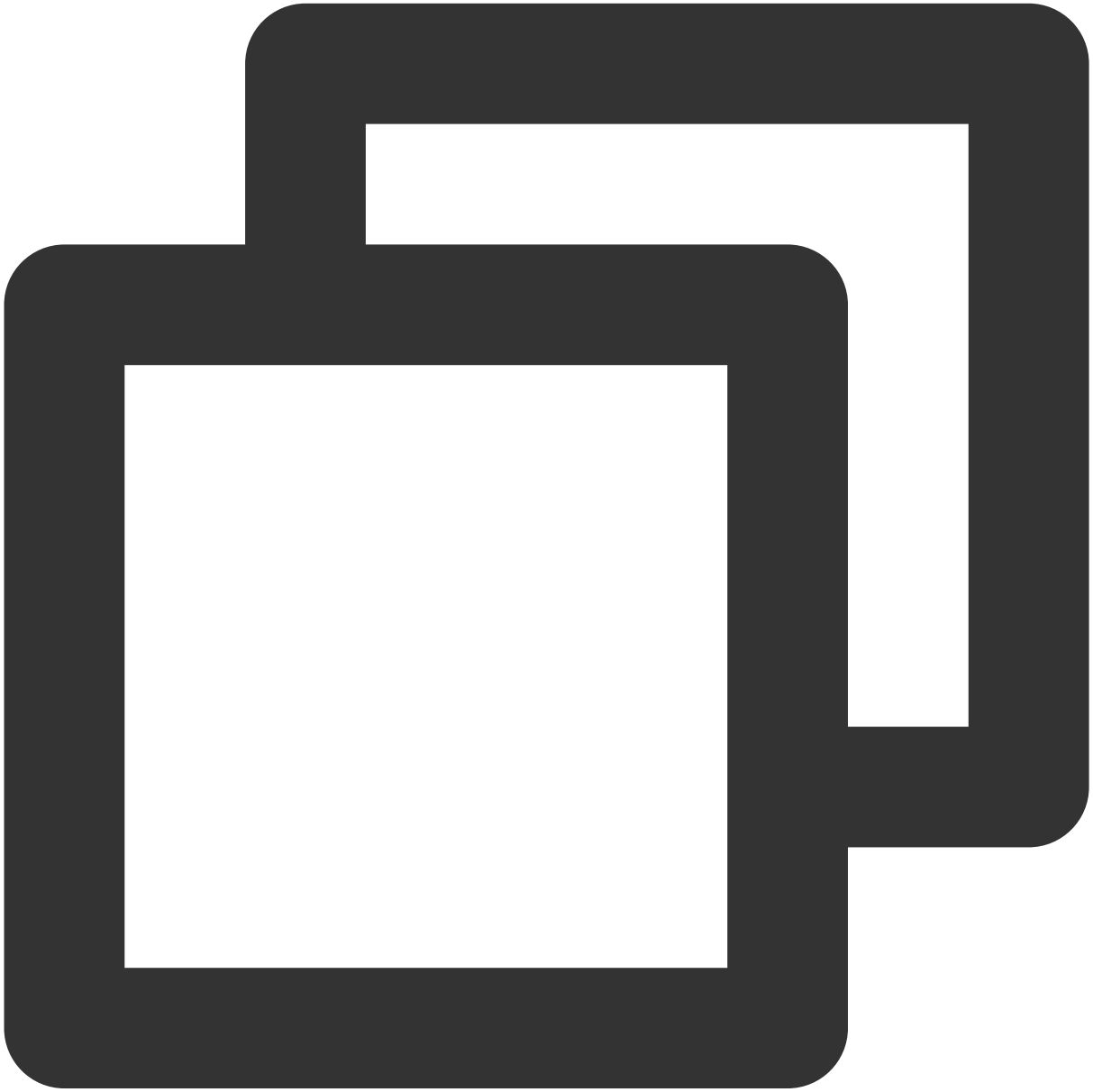
```
import 'package:tencent_cloud_av_chat_room/tencent_cloud_av_chat_room.dart';

class _TencentAVChatRoomKitState extends State<TencentCloudAVChatRoom> {
    final int _sdkAppID = 0; // SDKAppID of the IM application created in the preco
    final String _loginUserID = ""; // UserID in precondition
    final String _userSig = ""; // UserSig in preconditions
    @override
      Widget build(BuildContext context) {
        return TencentCloudAVChatRoom(
            config: TencentCloudAvChatRoomConfig(
                loginUserID: _loginUserID, sdkAppID: _sdkAppID, userSig: _userSig))
```

```
    }
}
```

Step 2: Using Components

Use that component in your appropriate module.



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

class _TencentAVChatRoomKitState extends State<TencentCloudAVChatRoom> {
    final int _sdkAppID = 0; // SDKAppID of the IM application created in the preco
```

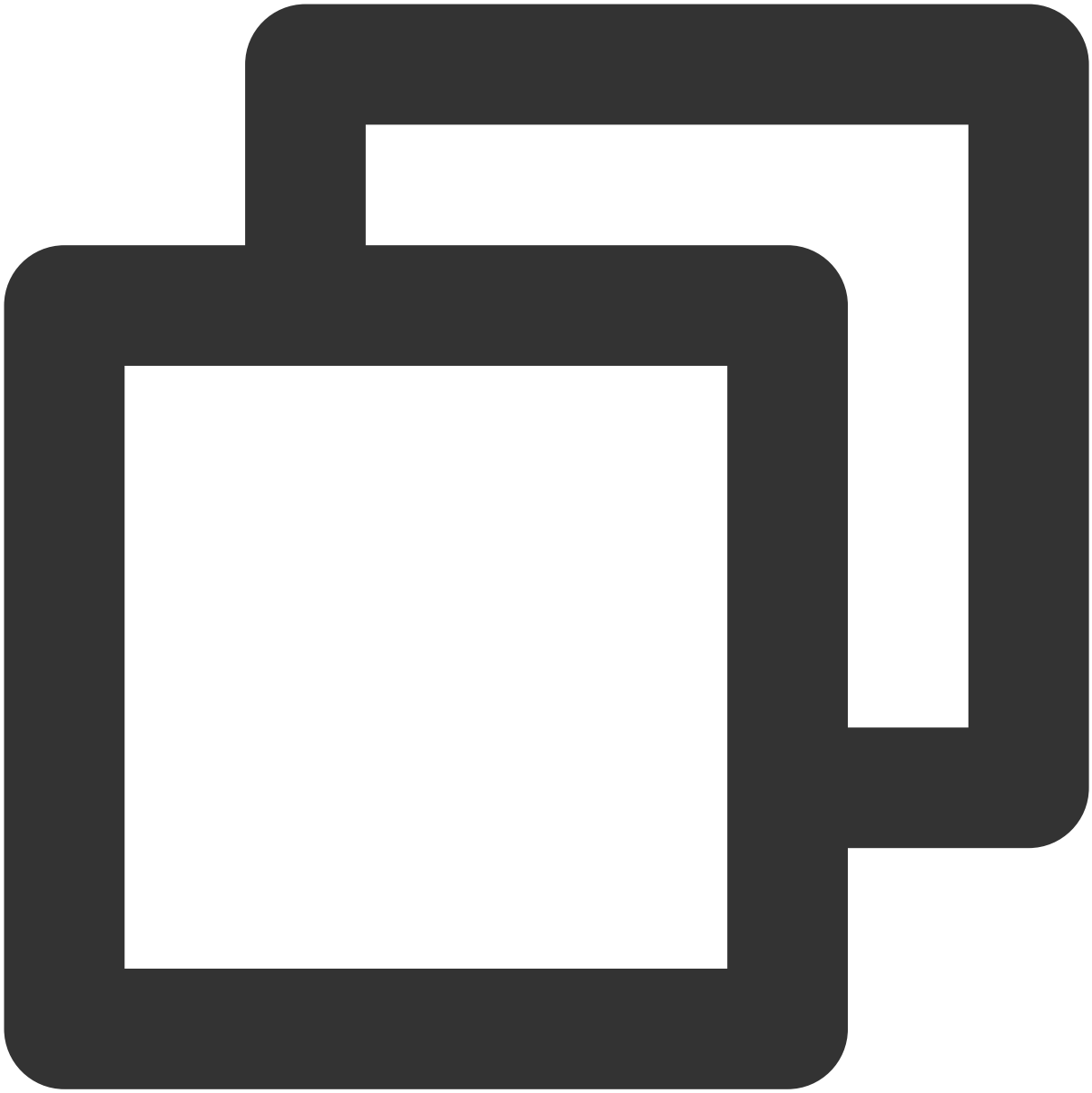
```
    final String _loginUserID = ""; // UserID in precondition
    final String _userSig = ""; // UserSig in preconditions
    @override
      Widget build(BuildContext context) {
        return TencentCloudAVChatRoom(
            data: TencentCloudAvChatRoomData(anchorInfo: AnchorInfo()),
      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

Cooperate with Tencent Cloud View Cube to build a live broadcast room

Live streaming is ubiquitous in life, and more and more companies and developers are building their own live streaming platforms. The following will introduce how to build a live broadcast room with Tencent Cloud View Cube. The live broadcast room mainly includes two parts: live broadcast and interaction. In the live broadcast part, we will use Tencent Cloud View Cube mobile live broadcast to realize live streaming, screen playback, etc. Interaction uses this plug-in to achieve live broadcast likes, gift giving, barrage sending and receiving, etc.
The relevant code of this article can be downloaded and viewed in Github.

Live streaming and screen playback

The live broadcast SDK is one of the sub-products of the Tencent Cloud View Cube product family. The live broadcast SDK supports live streaming push, pull streaming, host-audience interaction with hosts, host cross-room PK and other capabilities, providing users with stable and extremely fast live broadcast terminal services.
See Live Streaming Flutter SDK Standard Live Streaming to realize the functions of live streaming and screen playback.
1: Set License



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

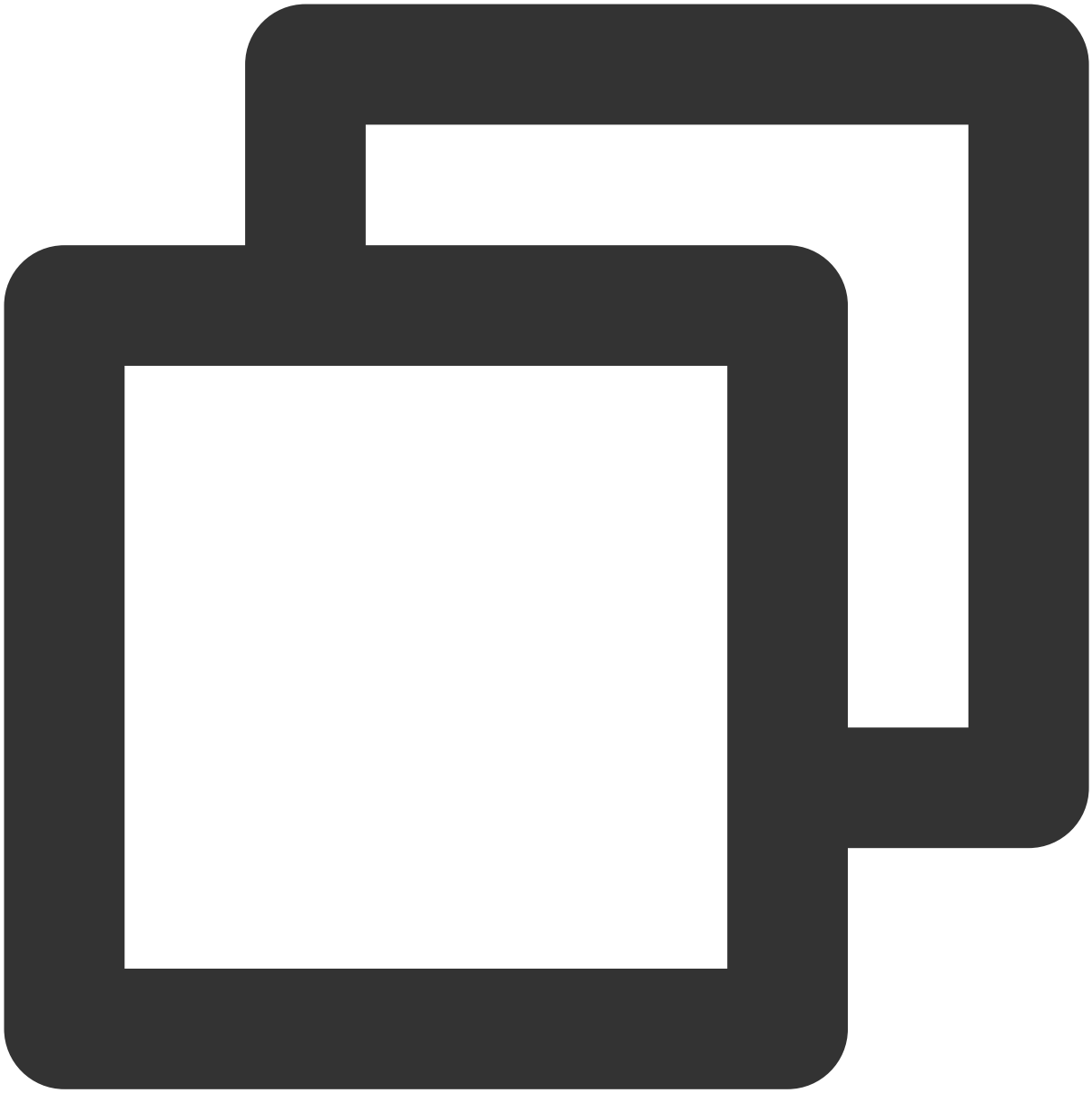
  @override
  State<Live> createState() => _LiveState();
}

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

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

    /// Tencent Cloud License Management Page (https://console.intl.cloud.tencent.c
    setupLicense() {
        // The currently applied License LicenseUrl
        final licenseUrl = "";
        // License Key currently applied
        final licenseKey = "";
        V2TXLivePremier.setLicence(licenseUrl, licenseKey);
    }
    ...
}
```

2: Live streaming, screen playback



```
// 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;
        });
```

```
        }
    }
```

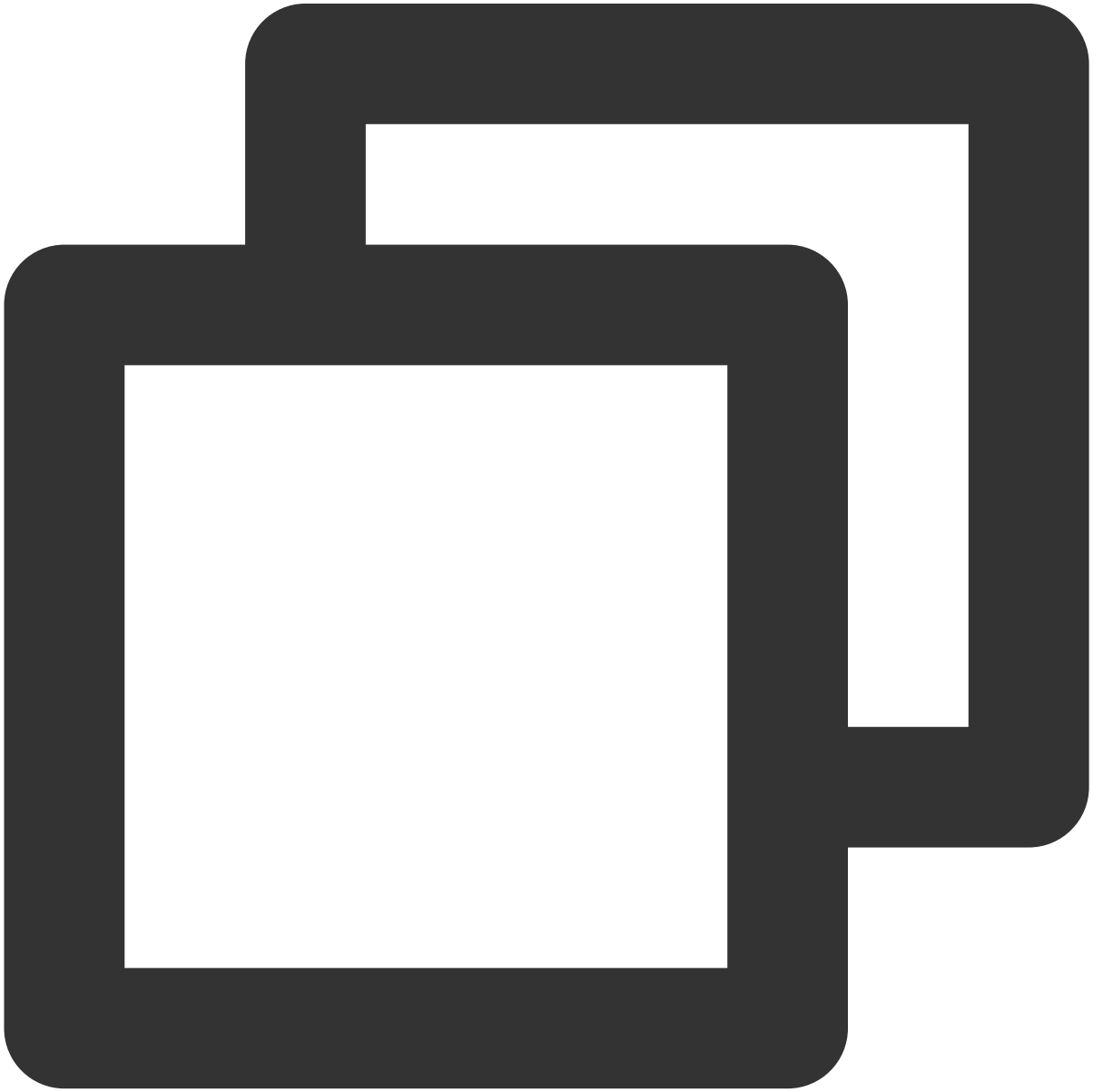
Live interaction (like, give gifts, send and receive barrage)

Live interaction is particularly important in live broadcast scenarios. Users interact with the anchor through barrage, gift giving, etc.

Next, see Introduction to integrate this plugin.

1: Initialize, log in to IM

We log in to IM outside the plugin, and usually the entire application only needs to initialize and log in to IM once.

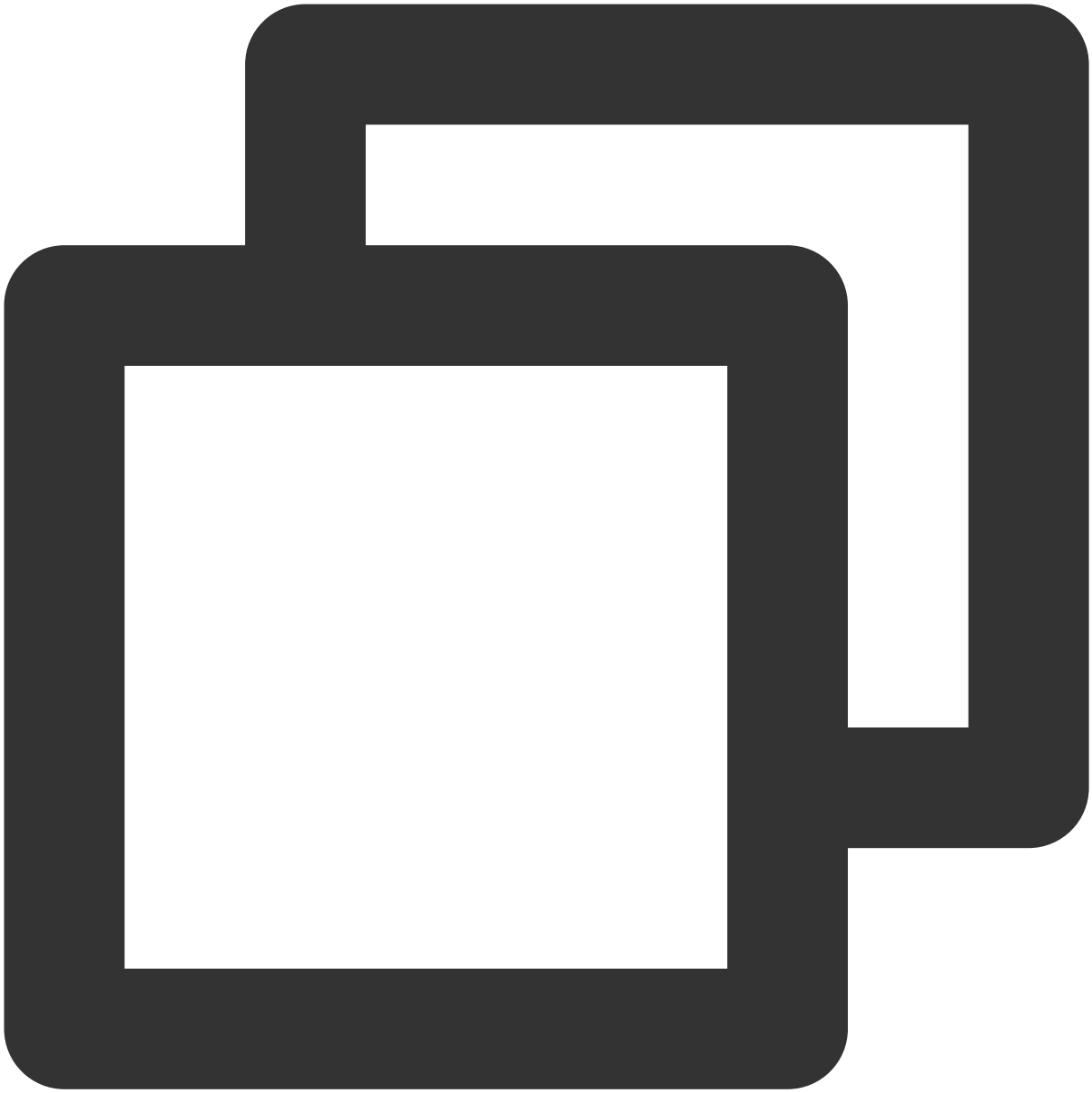


```
import 'package:tencent_av_chat_room_kit/tencent_cloud_chat_sdk_type.dart';

class _MyHomePageState extends State<MyHomePage> {
     final int _sdkAppID = 0; // SDKAppID of the IM application created in the prec
    final String _loginUserID = ""; // UserID in precondition
    final String _userSig = ""; // UserSig in preconditions
    @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: Use plugins



```
// live_room.dart

class LiveRoom extends StatelessWidget {
    final String loginUserID;
    final String playUrl;
    final String avChatRoomID = ''; // A live broadcast room is essentially all use

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

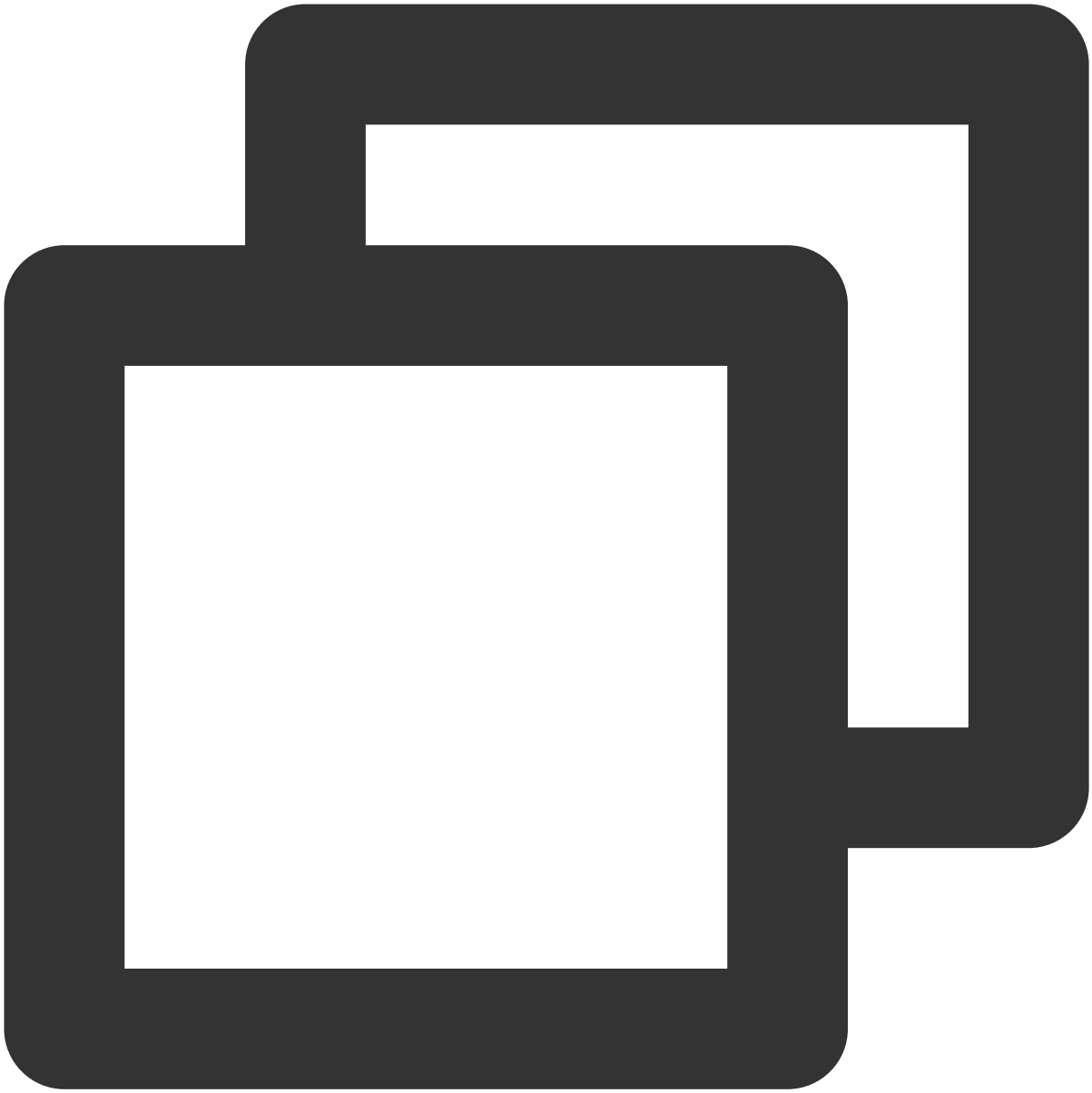
    @override
```

```
    Widget build(BuildContext context) {
        return Stack(
            children: [
                LivePlayer(playUrl: playUrl), // Live streaming and screen playback
                TencentCloudAVChatRoom(
                    config: TencentCloudAvChatRoomConfig(
                        avChatRoomID: avChatRoomID,
                        loginUserID: loginUserID, //Login user ID
                    ),
                    data: TencentCloudAvChatRoomData(
                        isSubscribe: false,
                        notification: "Broadcast Room Announcement",
                        anchorInfo: AnchorInfo(
                            subscribeNum: 200,
                            fansNum: 5768,
                            nickName: "stormy life",
                            avatarUrl:""
                        ),
                    )
                )
            ]
        )
    }
}
```

Above we use `Stack Widget` to combine `LivePlayer (live broadcast)` and `LiveRoom (interactive)` through cascading.

3: How to customize

The plug-in itself provides a default UI, but users often have different UIs during actual use. Next, we will introduce how to customize this plug-in.



```
...
@override
Widget build(BuildContext context) {
return Stack(
children: [
LivePlayer(playUrl: playUrl), // Live streaming and screen playback
TencentCloudAVChatRoom(
...
TencentCloudAvChatRoomCustomWidgets(
roomHeaderAction: Container(), // The area custom component is displayed in the upp
roomHeaderLeading: Container(), // The area custom component is displayed in the up
```

```
roomHeaderTag: Container(), // Below roomHeaderAction and roomHeaderLeading, it is
onlineMemberListPanelBuilder: (context, id) { // Customize the panel that expands a
return Container();
},
anchorInfoPanelBuilder: (context, id) { // Customize the expanded panel after the a
return Container();
},
giftsPanelBuilder: (context) { // Customize the panel displayed after clicking the
return Container();
},
messageItemBuilder: (context, message, child) { // Customize bullet chat messages
return Container();
},
messageItemPrefixBuilder: (context, message) { // Customize the prefix of bullet ch
return Container();
},
giftMessageBuilder: (context, message) { // Custom gift message, gift message that
return Container();
},
textFieldActionBuilder: (// Customize the lower right area of the screen
context,
) {
return [Container()];
},
textFieldDecoratorBuilder: (context) { // Customize the input box at the bottom lef
return Container();
}
)
)
]
)
}
```

4: Gift

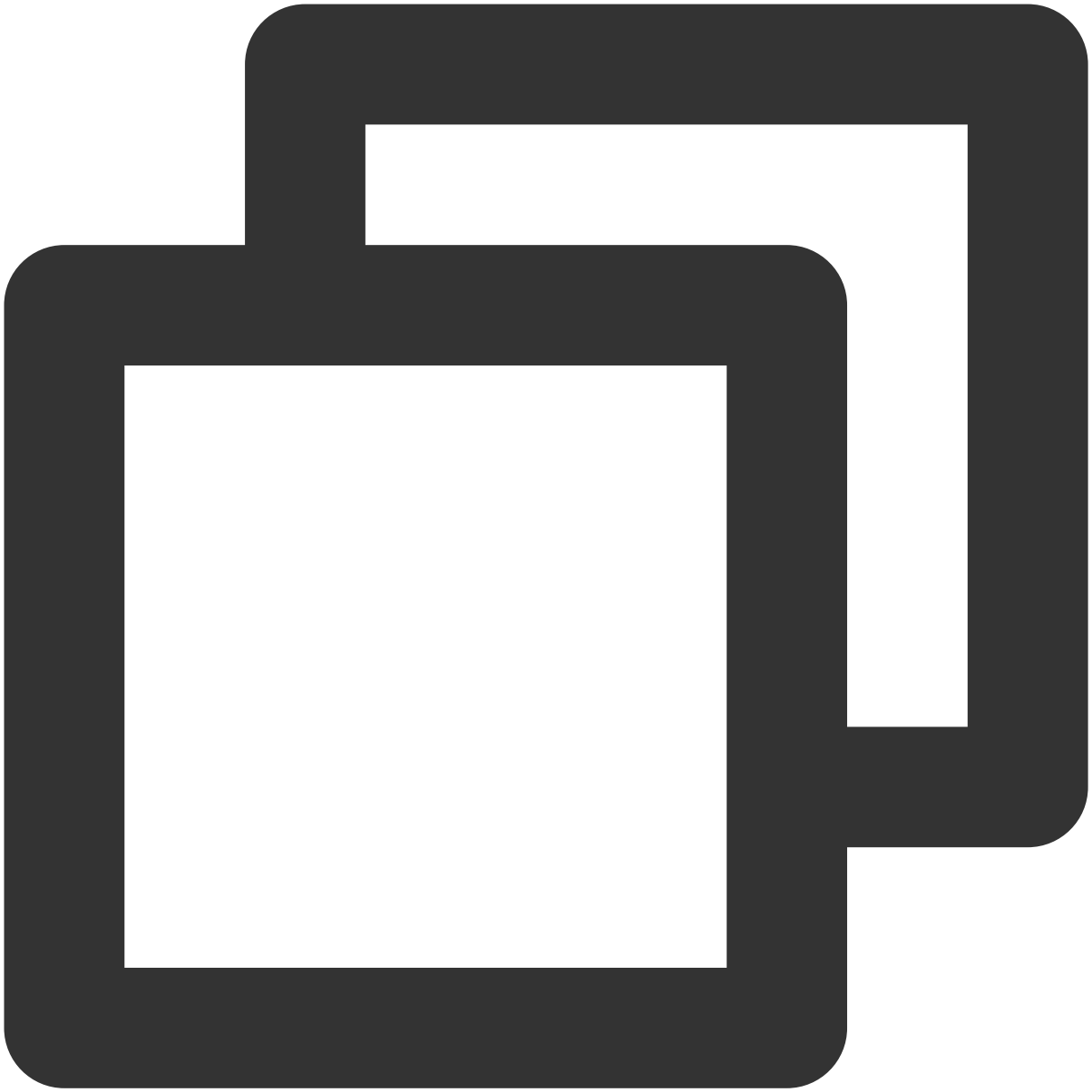
Giving gifts is a very important scene in the live broadcast scene. This plugin provides analysis of three types of gifts by default, users only need to send a custom message in a specific format to display the gift message on the interface. The default gift panel of this plug-in is only used to display the gift message parsing function. Usually, gifts will involve the logic of user billing and measurement, so users need to call the server interface to send gifts according to their business needs. information. The gift giving process is as follows:

The short connection request from the client to its own business server involves billing logic.

After billing, the sender directly sees that XXX sent XXX a gift. (To ensure that the sender sees the gift they sent, when there is a large amount of messages, the abandonment strategy may be triggered)

After billing and settlement, call the server interface to send a custom message (gift)

Gift Messages We do this by sending custom messages. Three gift formats are defined as follows:



```
// Gifts with special effects (the details of the gift will slide into the left sid
final customInfoRocket = {
"version": 1.0, // protocol version number
"businessID": "flutter_live_kit", // Business ID field
"data": {
"cmd":
"send_gift_message", // must be send_gift_message
"cmdInfo": {
"type": 3, // gift type
"giftUrl": "", // URL of gift image
"giftCount": 1, // number of gifts
```

```
"giftSEUrl": "assets/live/rocket.json", // gift special effect address, if it start
"giftName": "Super Rocket", // gift name
},
}
};

// The gift does not have special effects (it will slide in on the left side of the
final customInfoPlane = {
    "version": 1.0, // protocol version number
    "businessID": "flutter_live_kit", // Business ID field
    "data": {
      "cmd":
          "send_gift_message", // must be send_gift_message
      "cmdInfo": {
        "type": 2, // gift type
        "giftUrl": "", // URL of gift image
        "giftCount": 1, // number of gifts
        "giftName": "Airplane", // gift name
      },
    }
};

// Ordinary gift (the gift will be displayed in the barrage, and will not slide in
final normalGift = {
    "version": 1.0, // protocol version number
    "businessID": "flutter_live_kit", // business ID field
    "data": {
      "cmd":
          "send_gift_message", // must be send_gift_message
      "cmdInfo": {
        "type": 1, //Ordinary gift
        "giftUrl": "", // URL of gift image
        "giftCount": 1, // number of gifts
        "giftName": "flower", // gift name
        "giftUnits": "Duo", // gift units
      },
    }
};
```

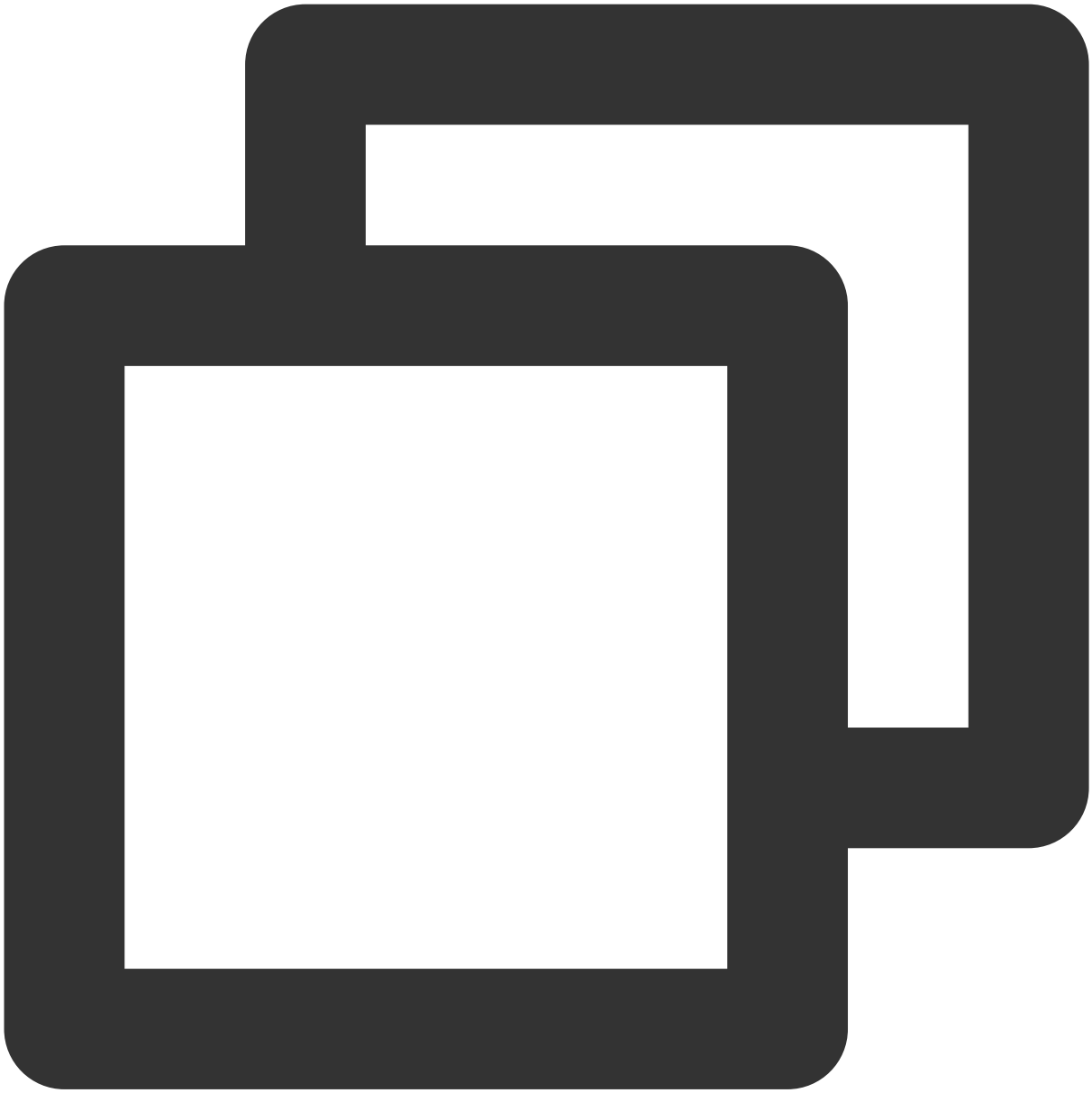
5: Theme

In addition to customization, this plugin also provides the capability of `theme`. The theme is divided into two parts: `color` and `font`. Theme can be customized according to your needs.

API Docs

TencentCloudAvChatRoomData

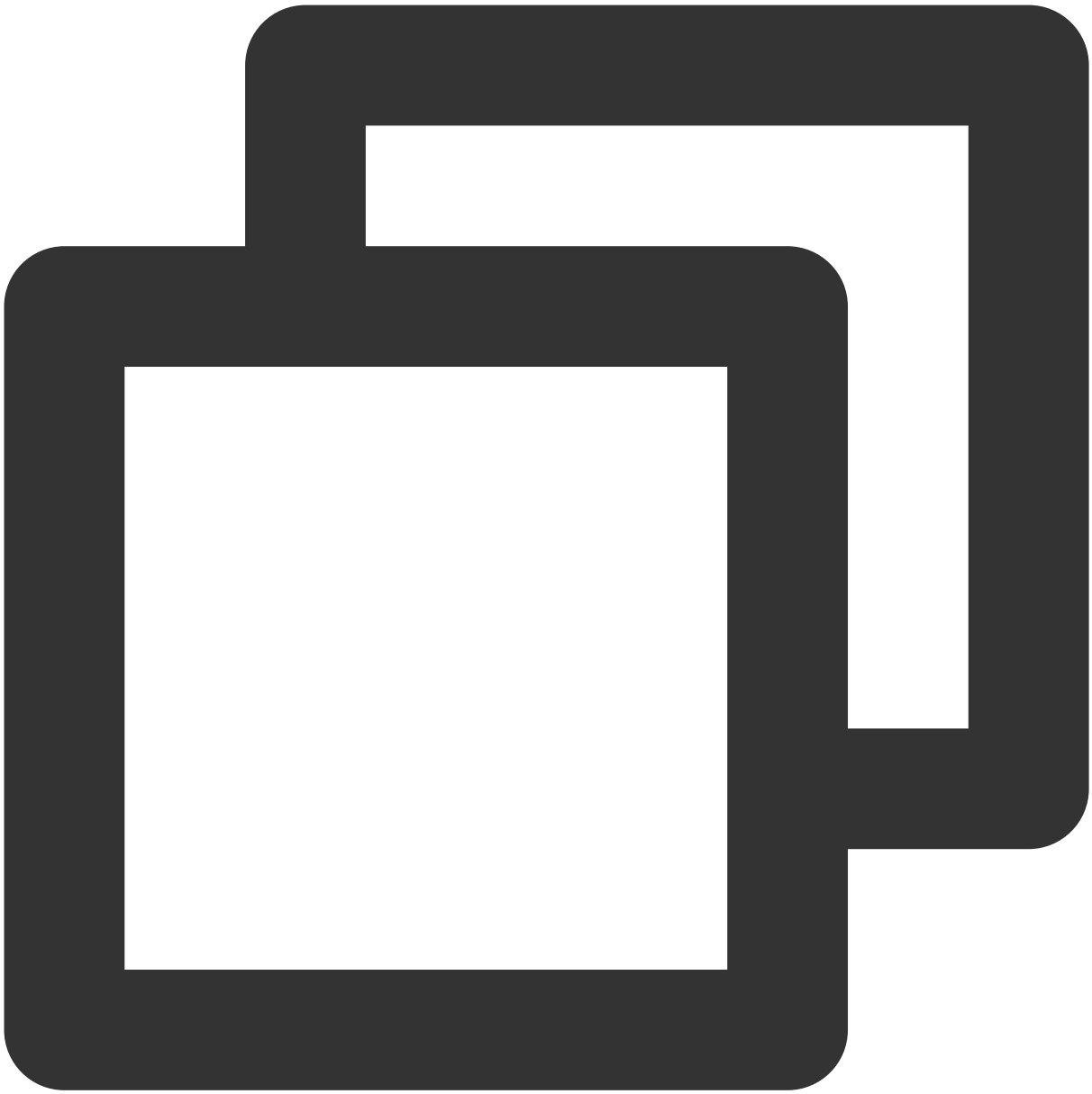
The data that the component needs to use, such as anchor information, broadcast room announcements, etc.



```
TencentCloudAvChatRoomData(
anchorInfo: AnchorInfo(), // anchor information
isSubscribe: false, // whether to subscribe
notification: "Live Room Announcement" // Live Room Announcement
)
```

TencentCloudAvChatRoomConfig

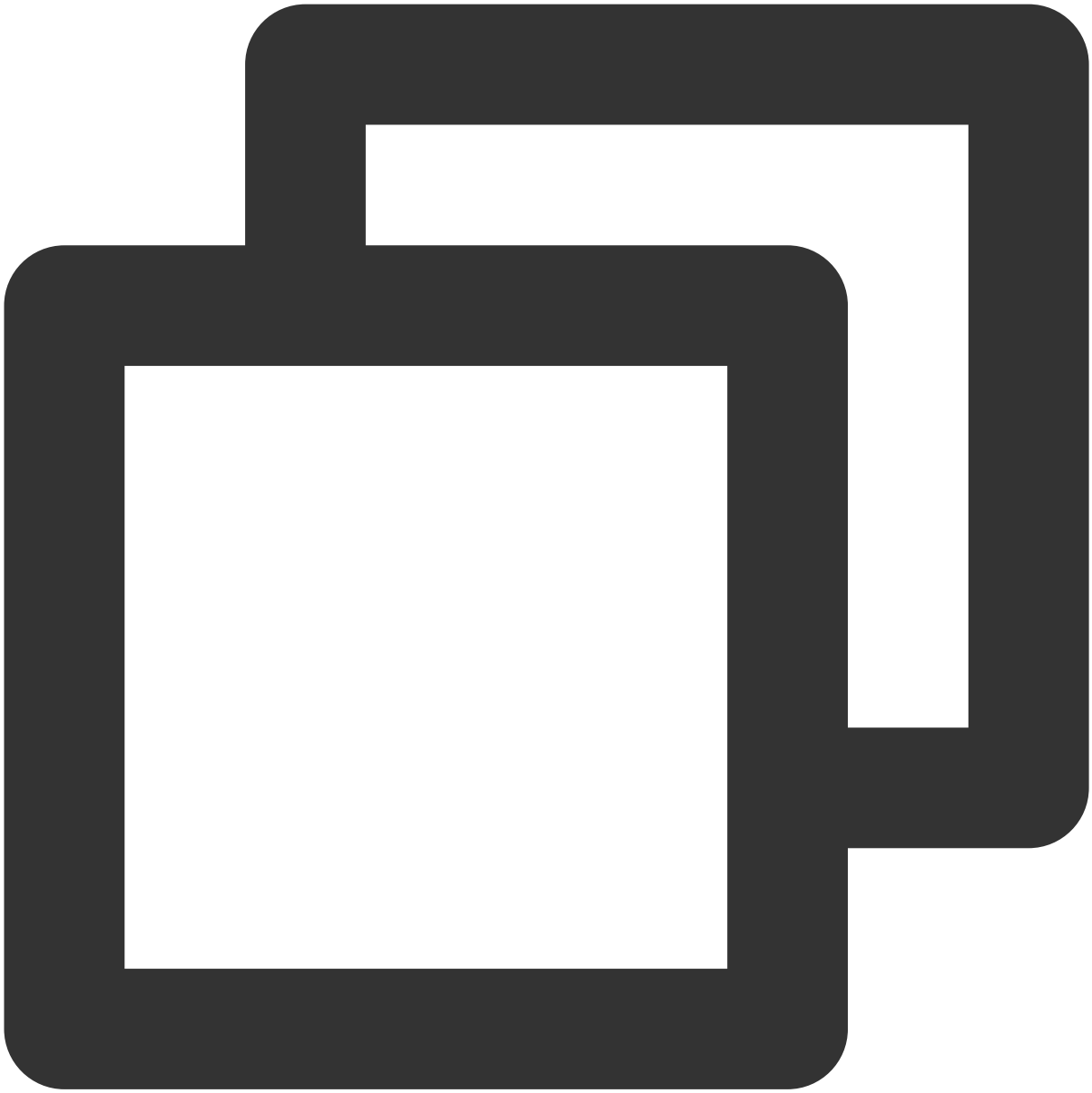
Component configuration information.



```
TencentCloudAvChatRoomConfig(
  avChatRoomID: '', // AV Chat Group. Group ID of AV Chat Room type.[https://cloud.
  loginUserID: '', // login user ID
  sdkAppID: 0, // IM application ID
  userSig: '', // sig generated by user ID and secretKey
  barrageMaxCount: 200, // The maximum number of barrage. The default is 200. When
  giftHttpBase: '', // Gift message http base.
  displayConfig: DisplayConfig() // Can control the display and hiding of some comp
  )
```

TencentCloudAvChatRoomController

Component controller, which can be called outside the component to update data, send messages, etc.



```
final controller = TencentCloudAvChatRoomController();
final _needUpdateData = TencentCloudAvChatRoomData(
        anchorInfo: AnchorInfo(), // anchor information
isSubscribe: false, // whether to subscribe
notification: "Live Room Announcement" // Live Room Announcement
);
final _textString = "I am a text message";
```

```
final customInfoRocket = {
"version": 1.0, // protocol version number
"businessID": "flutter_live_kit", // Business ID field
"data": {
"cmd":
"send_gift_message", // command
"cmdInfo": { // Information carried by the command
"type": 3, // gift type
"giftUrl": "1e8913f8c6d804972887fc179fa1fbd7.png", // gift image address
"giftCount": 1, // number of gifts
"giftSEUrl": "assets/live/rocket.json", // gift special effect address
"giftName": "Super Rocket", // gift name
},
}
};

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

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

```
// Update the data information of the incoming component.
controller. updateData(_needUpdateData);

// send text message
controller. sendTextMessage(_textString);

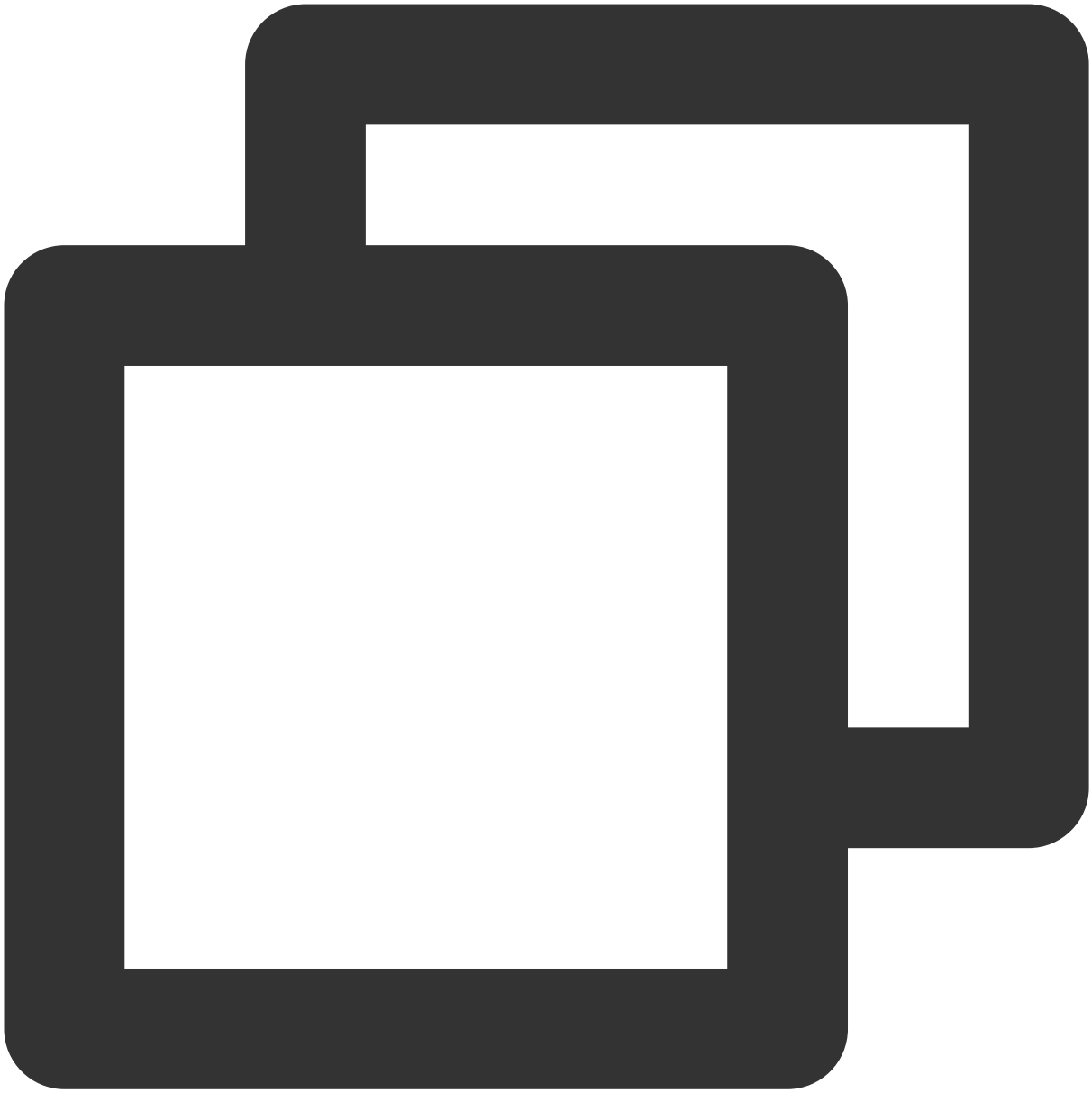
// Send a gift message, the gift message needs to follow the specific format above.
controller.sendGiftMessage(jsonEncode(customInfoFlower));

// Send any type of message, [message] needs to be created by yourself
controller. sendMessage(message);

// Play special effects animation (Lottie, SVGA).
controller.playAnimation("assets/live/rocket.json"):
```

TencentCloudAvChatRoomCallback

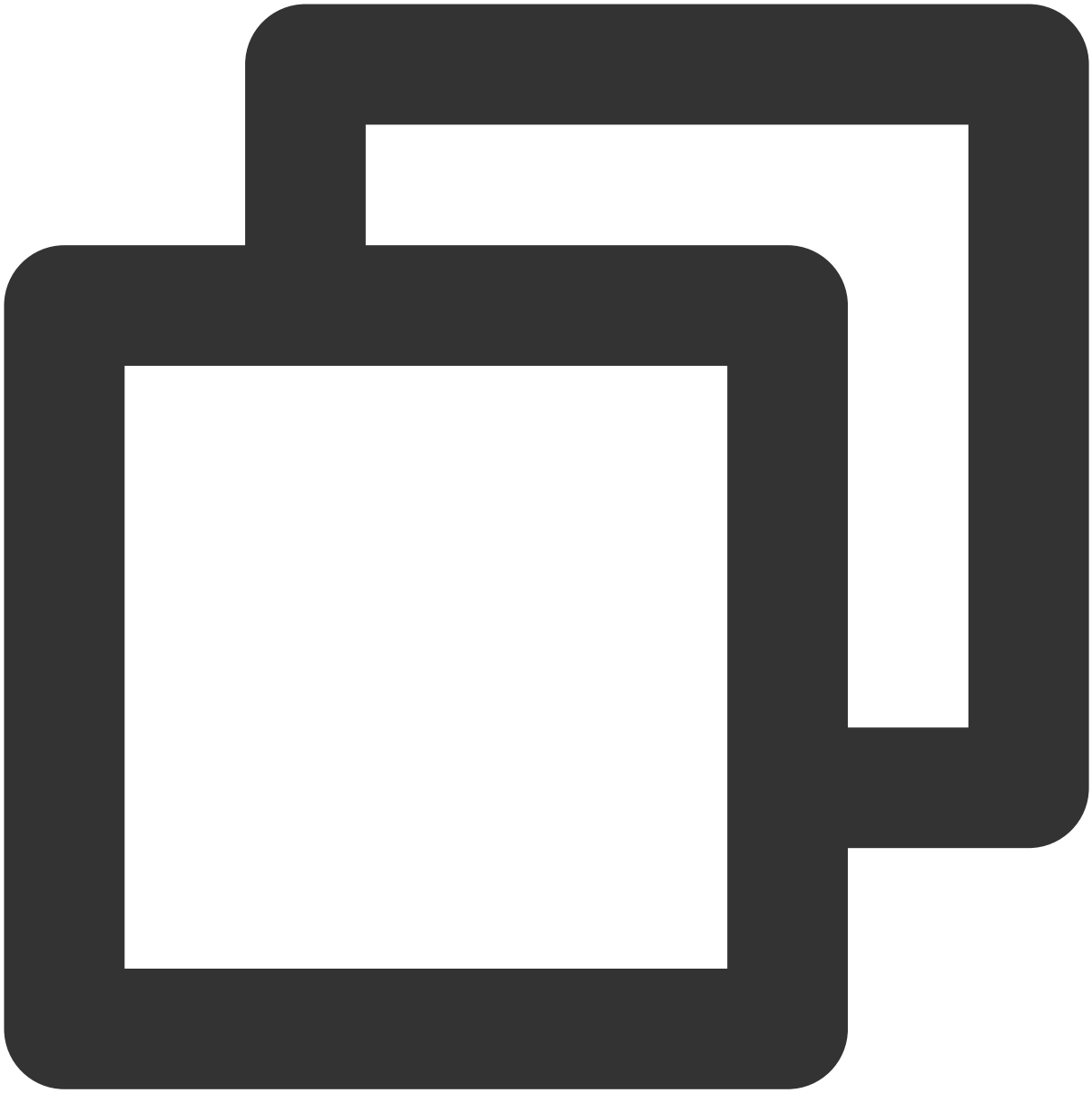
Event callback。



```
TencentCloudAvChatRoomCallback(
onMemberEnter: (memberInfo) {}, // someone enters the live room
onRecvNewMessage: (message) {} // received barrage message
)
```

TencentCloudAvChatRoomCustomWidgets

custom components

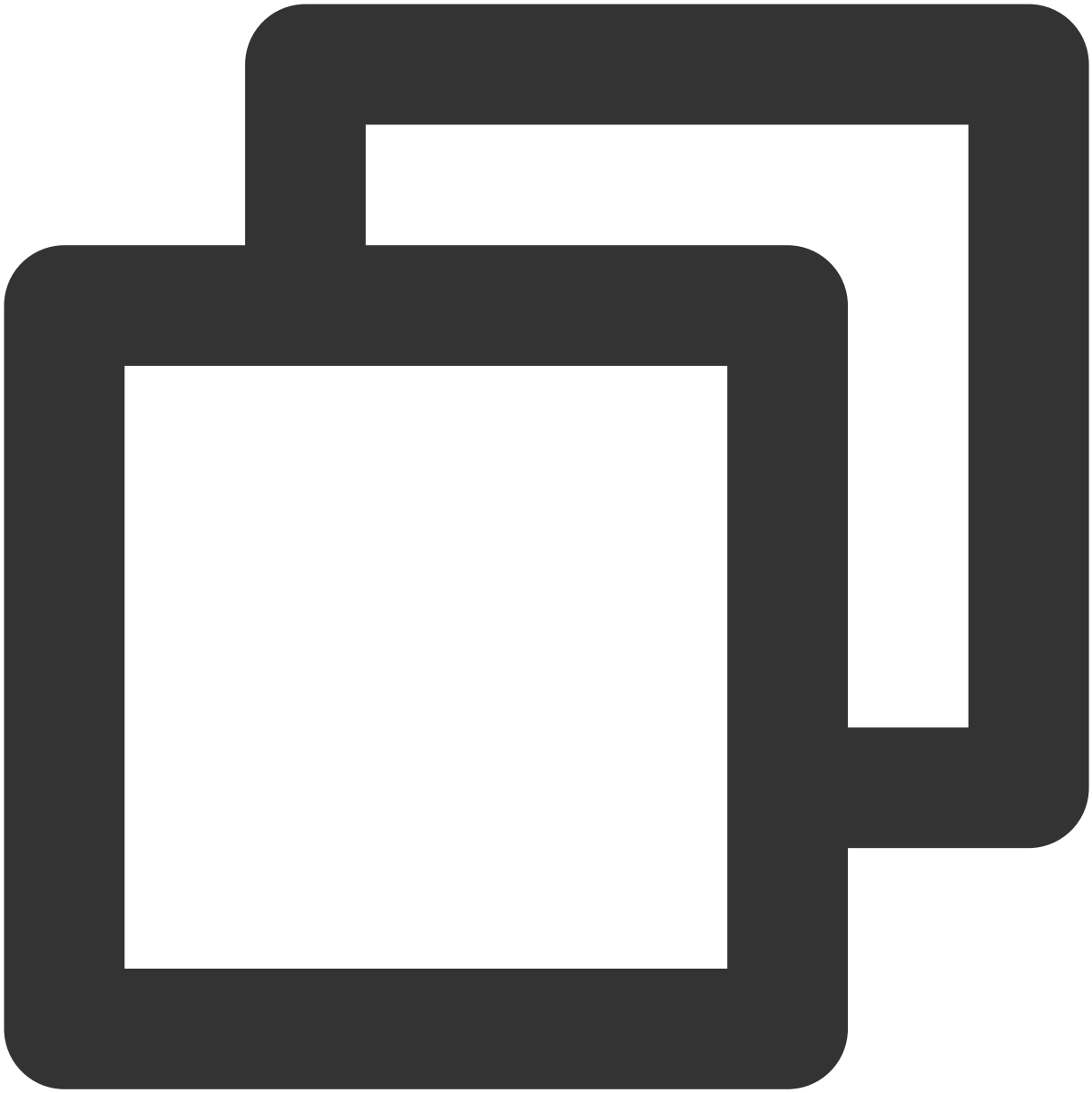


```
TencentCloudAvChatRoomCustomWidgets(
roomHeaderAction: Container(), // The area custom component is displayed in the upp
roomHeaderLeading: Container(), // The area custom component is displayed in the up
roomHeaderTag: Container(), // Below roomHeaderAction and roomHeaderLeading, it is
onlineMemberListPanelBuilder: (context, id) { // Customize the panel that expands a
return Container();
},
anchorInfoPanelBuilder: (context, id) { // Customize the expanded panel after the a
return Container();
},
giftsPanelBuilder: (context) { // Customize the panel displayed after clicking the
```

```
return Container();
},
messageItemBuilder: (context, message, child) { // Customize bullet chat messages
return Container();
},
messageItemPrefixBuilder: (context, message) { // Customize the prefix of bullet ch
return Container();
},
giftMessageBuilder: (context, message) { // Custom gift message, gift message that
return Container();
},
textFieldActionBuilder: (// Customize the lower right area of the screen
context,
) {
return [Container()];
},
textFieldDecoratorBuilder: (context) { // Customize the input box at the bottom lef
return Container();
}
)
```

TencentCloudAvChatRoomTheme

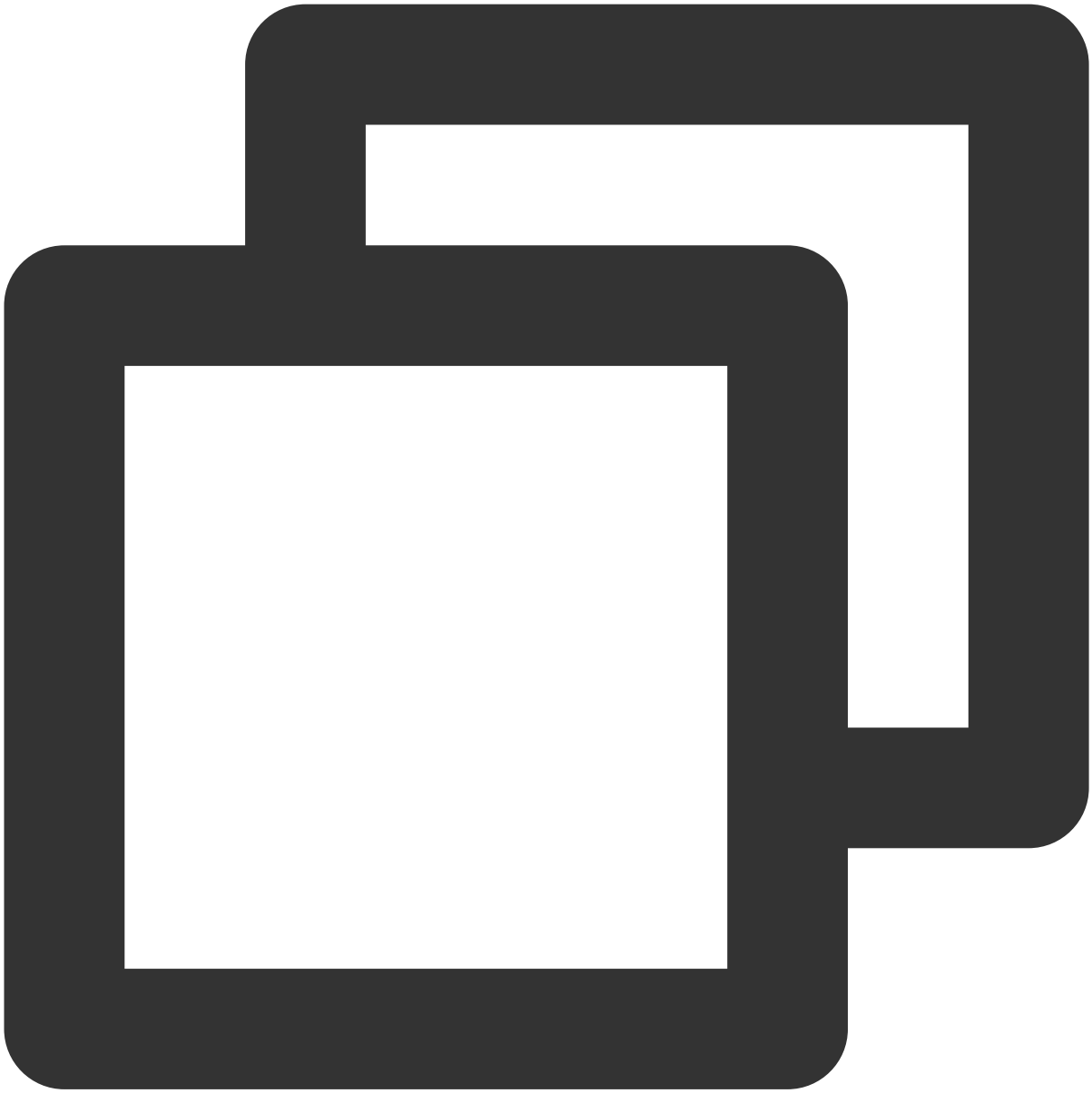
Theme



```
TencentCloudAvChatRoomTheme(
backgroundColor: Colors.black, // The background color of the widget,
hintColor: Colors.red, // hint text color
highlightColor: Colors.orange, // highlight color
accentColor: Colors.white, // foreground color
textTheme: TencentCloudAvChatRoomTextTheme(), // font theme
secondaryColor: Colors.grey, // secondary color
inputDecorationTheme: InputDecorationTheme() // input box theme
)
```

TencentCloudAvChatRoomTextTheme

font theme



```
TencentCloudAvChatRoomTextTheme(
giftBannerSubTitleStyle: TextStyle(), // gift message drawn on the left side of the
giftBannerTitleStyle: TextStyle(), // The title font theme of the gift message draw
anchorTitleStyle: TextStyle(), // Anchor name font theme
anchorSubTitleStyle: TextStyle(), // like font theme
barrageTitleStyle: TextStyle(), // barrage message sender name subject
barrageTextStyle: TextStyle() // Barrage message content theme
)
```

contact us

If there's anything unclear or you have more ideas, feel free to contact us!

Telegram Group

WhatsApp Group

AI Chatbot

最終更新日：：2024-02-07 17:30:51

With the global popularity of ChatGPT, artificial intelligence (AI) has become the focus of developers today, and mainstream vendors in China have launched their own big model (BM) applications and products. Many vendors have combined their applications with AI to discover new opportunities. The powerful conversational communication capabilities of next-generation large language models (LLMs) are naturally compatible with all kinds of instant messaging scenarios, which brings broad imagination space for the combination of Tencent Cloud Chat and AI.

In office scenarios, users can chat with conversational AI to efficiently make work notes, write documents, collect information, and more. In customer service scenarios, AI-powered smart customer service can provide a conversational experience similar to human customer service and guide users to purchase and use products more effectively. In social scenarios, AI chatbots can provide users with 24-hour online psychological counseling and emotional companionship, increasing user engagement and more. Tencent Cloud Chat, the world's leading provider of communication cloud services, has also seen the huge potential of AI in the instant messaging scenario, and quickly released AI capability call APIs. Based on the communication base provided by Tencent Cloud Chat, developers can freely call industry-leading BM capabilities and empower themselves with rich AI capabilities to efficiently implement scenario-specific innovations.

This document describes how to integrate AI service capabilities into Tencent Cloud Chat through the webhook feature of Chat to build an AI chatbot for users to implement features such as intelligent customer service, creative assistance, and work assistant. (The procedure in this document takes the MiniMax LLM as an example. You can use the same method to integrate other ChatGPT-like services.)

Preparations

Creating a Tencent Cloud Chat account

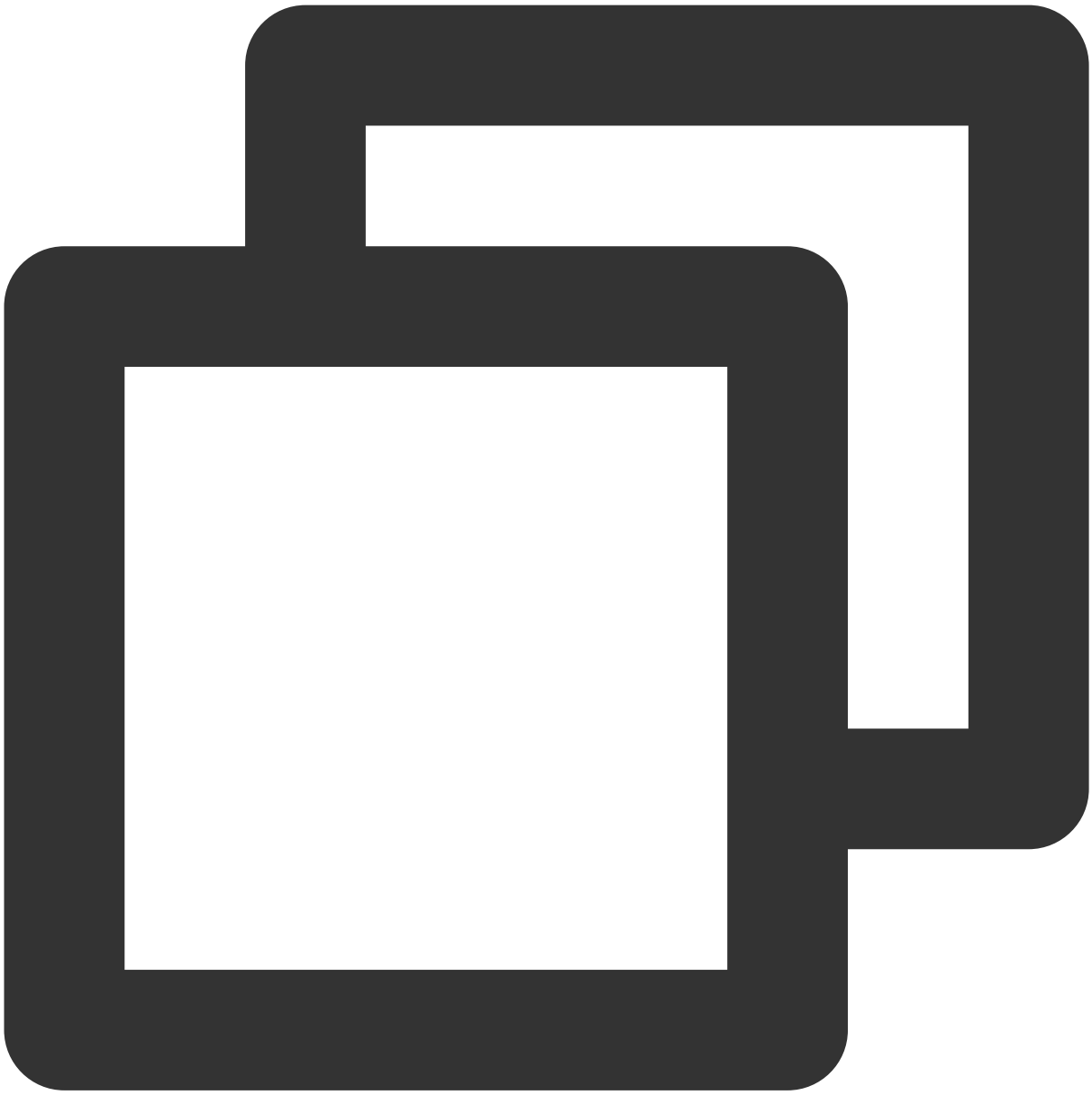
Log in to your Tencent Cloud account, go to the Tencent Cloud Chat console, create an application, get the application's SDKAppID and key (Tencent Cloud Chat key), and create an admin account `administrator`.

Signing up for an account with the corresponding AI service provider

Sign up for an account with the provider of the AI service to be integrated, log in, and get the API key (`AI_SECRET_KEY`).

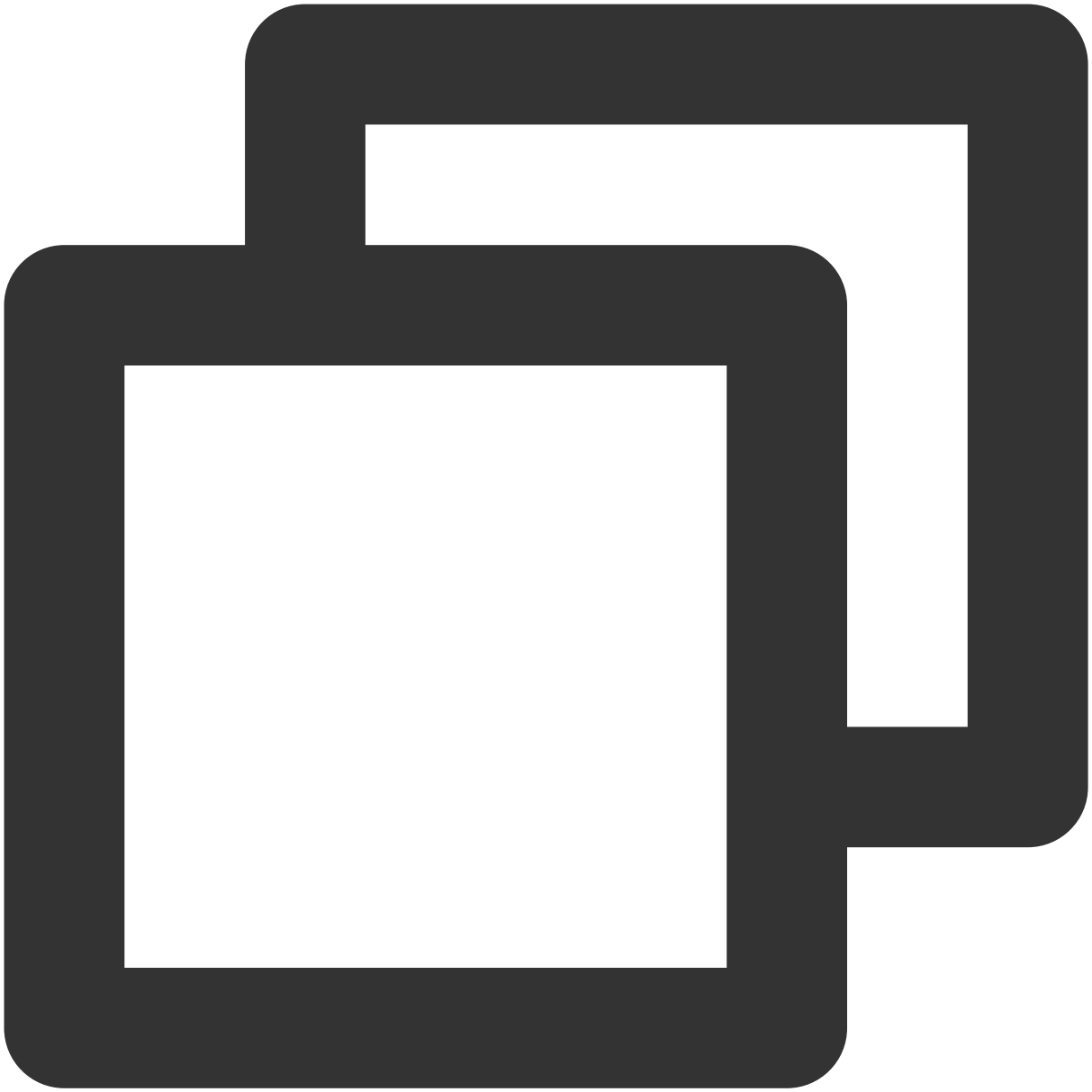
Creating a Tencent Cloud Chat chatbot account

Create a Tencent Cloud Chat chatbot account through the RESTful API. The Tencent Cloud Chat chatbot is a special user whose user ID begins with `@RBT#` .



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

Replace `sdkappid={}` and `usersig={}` in the command above with your SDKAppID and the UserSig generated based on the Tencent Cloud Chat key. For more information, see Generating UserSig. After you run the command in Linux, the Tencent Cloud server returns the following information:



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

The information above indicates that the chatbot `@RBT#001` with the nickname `MyRobot` was created successfully.

Configuring Tencent Cloud Chat webhooks

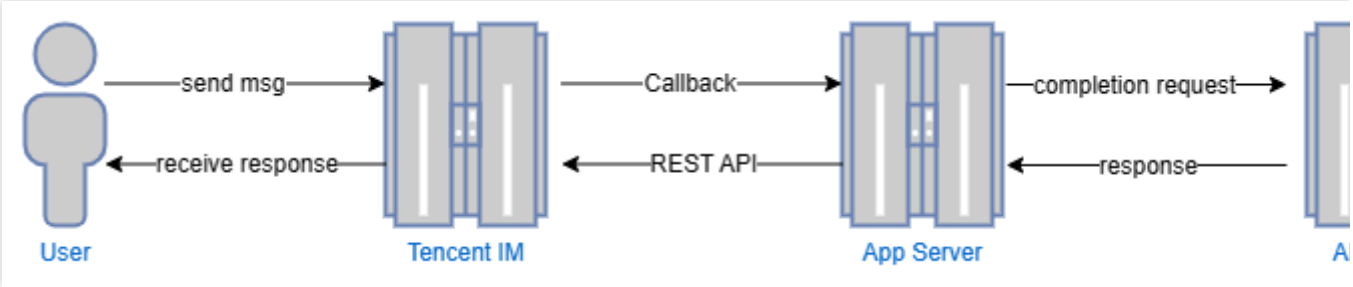
A Tencent Cloud Chat webhook is a request sent by the Tencent Cloud Chat backend to the backend server of the corresponding application before or after an event. The application backend can then perform the necessary data synchronization or intervene in the subsequent processing of the event. We will use a "robot event webhook" to listen

for and react to user messages sent to the chatbot or @RBT# events in group chats. You need to locate and click "Robot Event Webhook" in the Tencent Cloud Chat console to enable the feature and save the settings.

Writing the Application Backend Service

Taking a one-to-one chat as an example, the overall working process is as follows:

1. The `user1` user sends the "hello" message to the chatbot `@RBT#001` .
2. The Tencent Cloud Chat backend sends a webhook to notify the application backend of the event.
3. The application backend receives the event notification which contains information such as the message sender `user1` , message recipient `@RBT#001` , and message content `hello` .
4. The application backend calls the AI service API (MiniMax API) and receives the response containing the reply message, such as "nice to meet you".
5. The application backend calls the Tencent Cloud Chat RESTful API (API `sendmsg` for a one-to-one chat and API `send_group_msg` for a group chat) to send the reply message to `user1` as `@RBT#001` .



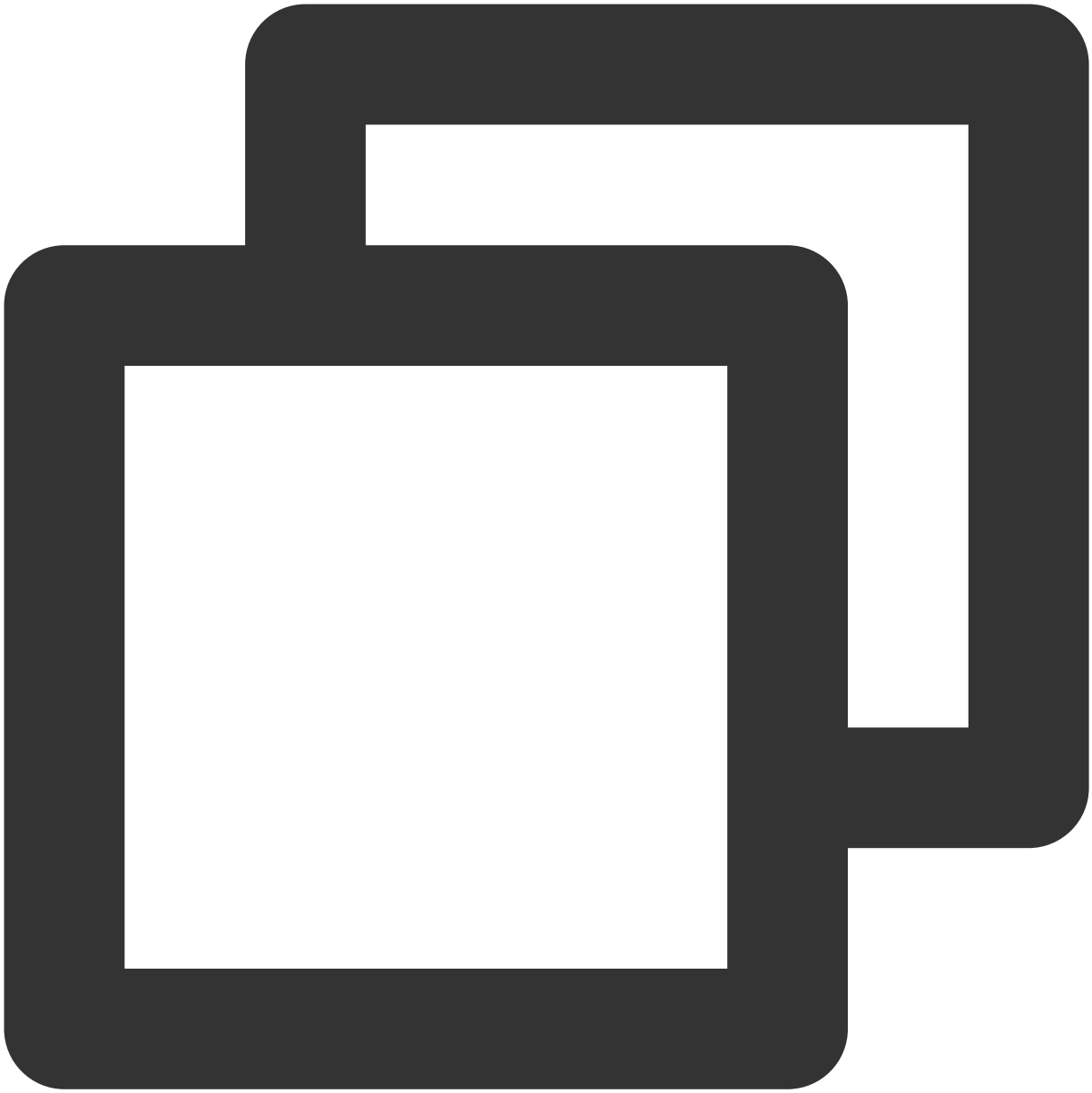
Take the Go programming language as an example, the key code of the application backend is as follows.

Note:

The following code is for demonstration only and omits a lot of exception handling code. It cannot be used directly in production environments.

Distributing and processing the webhook command

We create an HTTP service which is listened to on port 80 and register a handler with the `/im` URL that handles all requests sent to `http://im` . All webhook requests sent by Tencent Cloud Chat contain a `CallbackCommand` parameter, with different values representing different webhook commands. The handler performs processing according to the `CallbackCommand` parameter set by Tencent Cloud Chat.



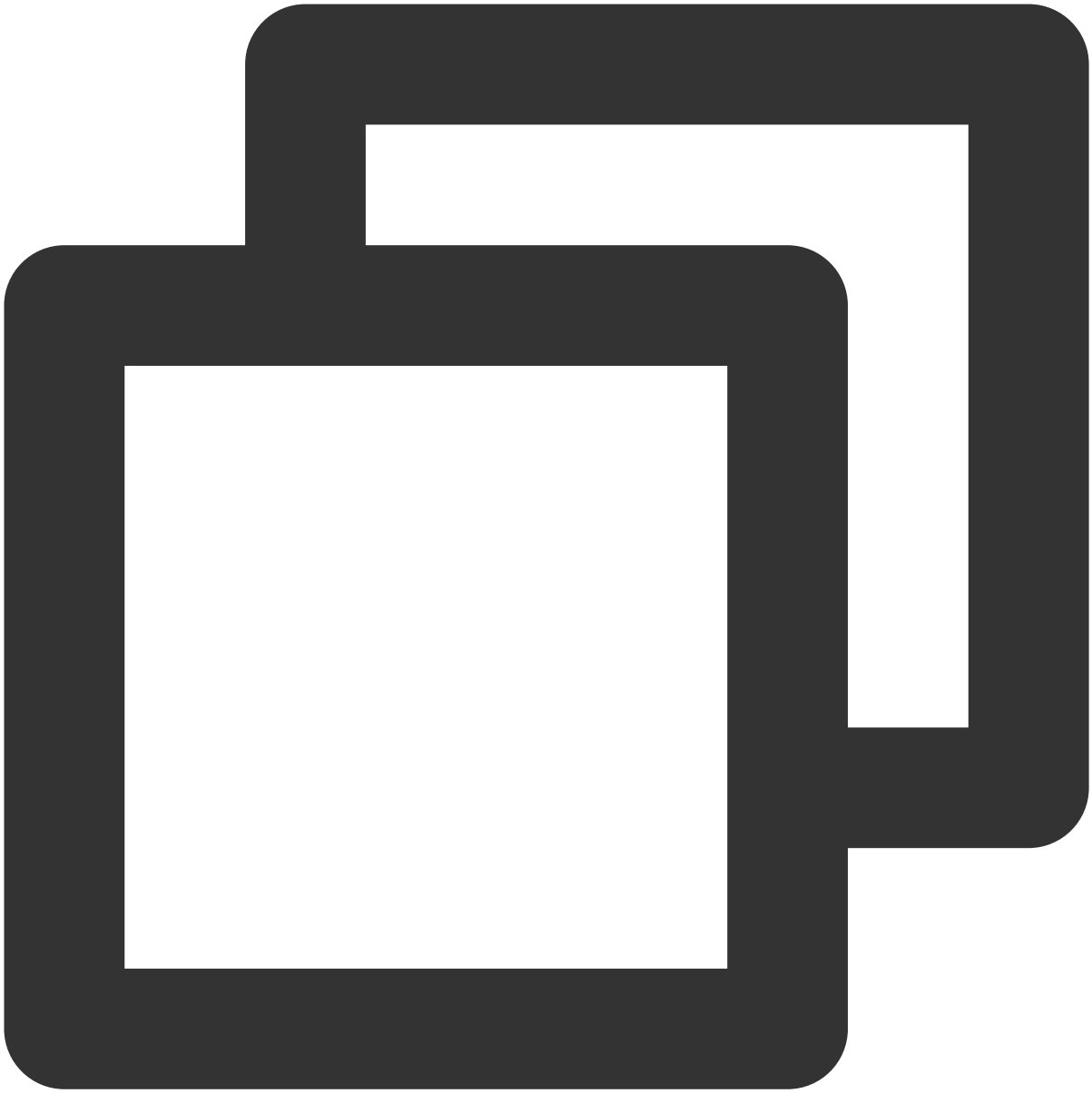
```
func handler(w http.ResponseWriter, r *http.Request) {
   command := r.URL.Query().Get("CallbackCommand")
   reqbody, _ := io.ReadAll(r.Body)
   var rspbody []byte
   switch command {
   case "Bot.OnC2CMessage": // Chatbot's webhook command word for a one-to-one mess
       dealC2c(context.Background(), reqbody)
       rspbody = []byte("{\\"ActionStatus\\": \\"OK\\", \\"ErrorCode\\": 0, \\"Erro
   default:
       rspbody = []byte("invalid CallbackCommand.")
   }
```

```
    w.Write(rspbody)
}

func main() {  // Register a handler to process the webhook command sent to the app
    http.HandleFunc("/im", handler)
    http.ListenAndServe(":80", nil)
}
```

Processing the one-to-one message received by the chatbot

When processing a one-to-one message, we first check that the sender is not a chatbot (generally a chatbot does not send a message to another chatbot) to prevent infinite webhook loops. We then parse the body of the message to obtain the content text of the message sent by the user to the chatbot, save the sender's UserID into the context to facilitate the subsequent action of calling the RESTful API to reply, and finally call `askAI` to request the AI service.

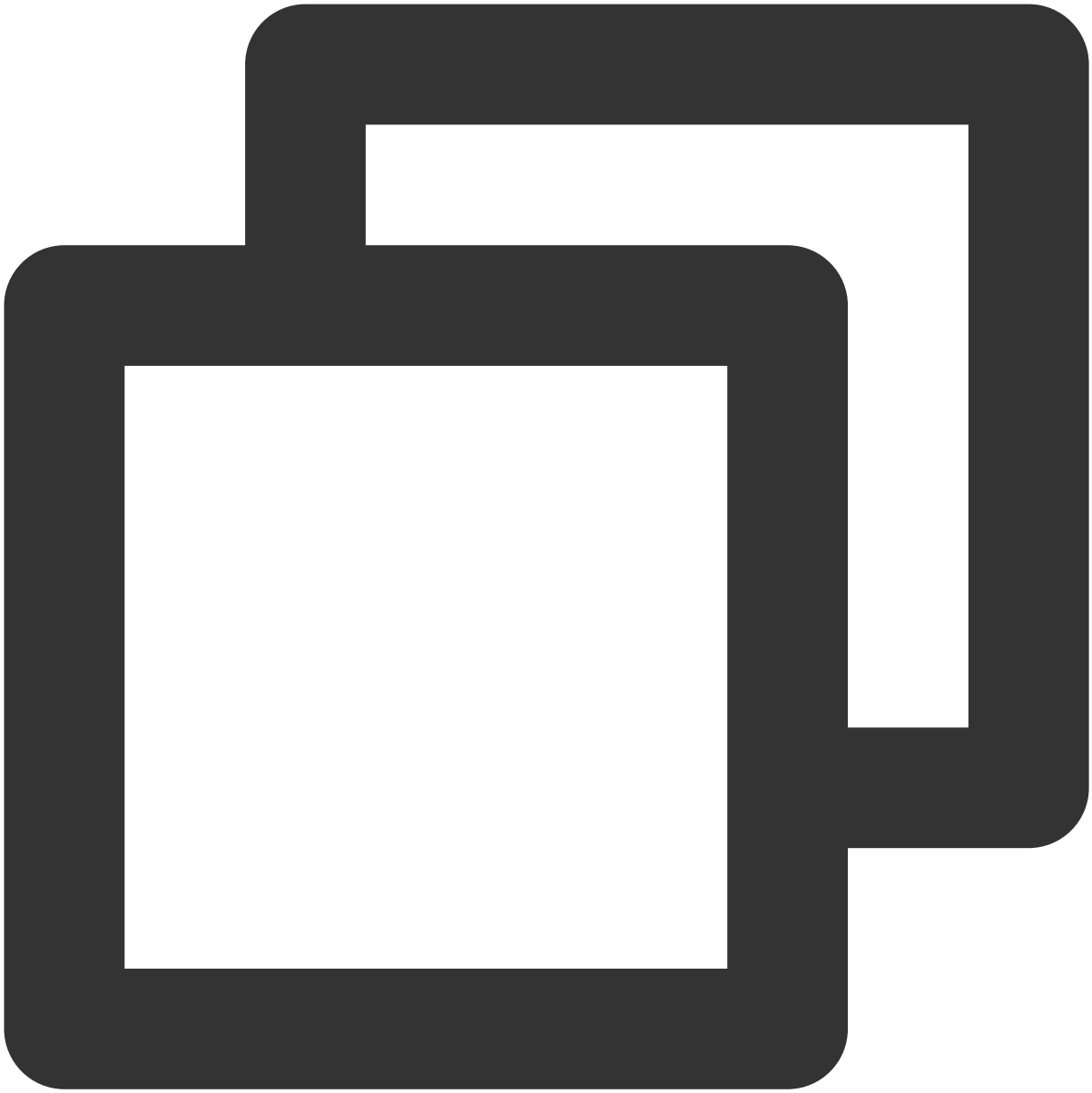


```
func dealC2c(ctx context.Context, reqbody []byte) error {
  root, _ := simplejson.NewJson(reqbody)
  jFromAccount := root.Get("From_Account")
  fromAccount, _ = jFromAccount.String()
  // Check the sender's ID to avoid processing requests sent by one chatbot to anot
  if strings.HasPrefix(fromAccount, "@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
  }
```

Calling the AI service API

In this step, we use a third-party AI service, MiniMax LLM, to implement intelligent chat. Any other AI service can be used instead of the MiniMax LLM service. Note that here we demonstrate a simple `completion` API that does not contain the context of the conversation, and you can see the MiniMax documentation for details on other APIs as needed.



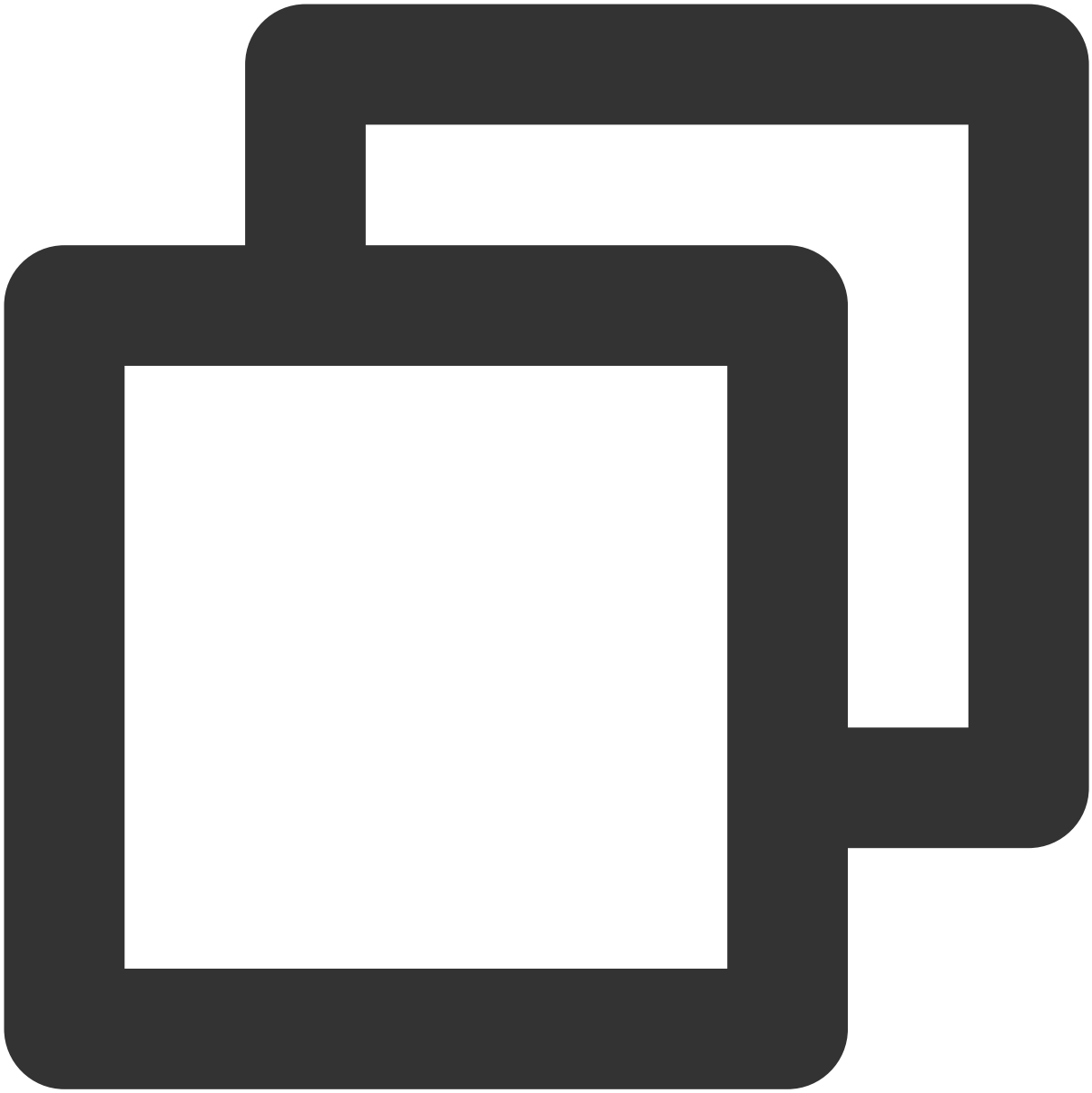
```
type MiniMaxRsp struct {
  Reply string `json:"reply"`
}

// Send a request to MiniMax and get a reply
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) // Send the content replied by the AI service to the user
}
```

Returning the result replied by the AI service to the user

After receiving the reply from the AI service, we only need to call the `sendmsg` RESTful API of Tencent Cloud Chat to simulate the chatbot to reply to the user, specifying the sender of the message as `@RBT#001` and the recipient as `user1` .



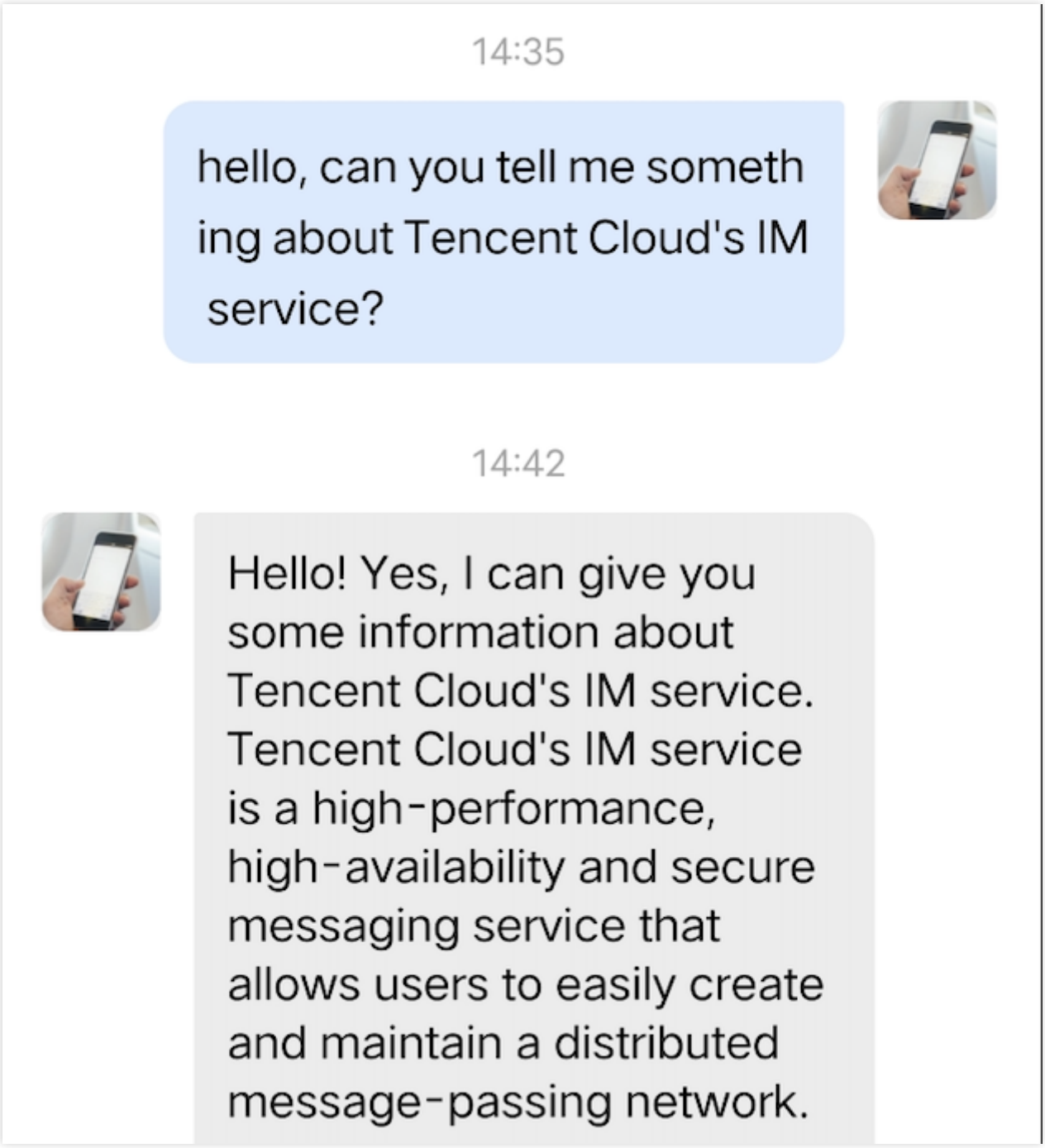
```
// Send a RESTful API request
func doRestAPI(host string, sdkappid int, admin, usersig, command, body string) {
  url := fmt.Sprintf("https://%s/v4/%s?sdkappid=%d&identifier=%s&usersig=%s&random=
    host, command, sdkappid, admin, usersig, rand.Uint32())
  req, _ := http.NewRequest("POST", url, 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)
}

// Call the RESTful API of Tencent Cloud Chat to reply to the user
func reply(ctx context.Context, text string) {
    rsp := make(map[string]interface{})
    msgbody := []map[string]interface{}{{
        "MsgType":    "TIMTextElem",
        "MsgContent": map[string]interface{}{"Text": text},
    }}
    // For the implementation of `GenUserSig`, see the Tencent Cloud documentation.
    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
}
```

Effect Demonstration

The following demonstrates the final implementation effect of the Tencent Cloud Chat chatbot demo:



With the above steps, we have implemented one-to-one chat connectivity between the Tencent Cloud Chat server side and the MiniMaxAI open platform. It is also possible to integrate AI services from another AI service provider following the above steps by simply replacing the `askAI` function with the corresponding API call from that AI service provider. For group chat chatbots, only the implementation of the `Bot.OnGroupMessage` webhook command processing needs to be supplemented.

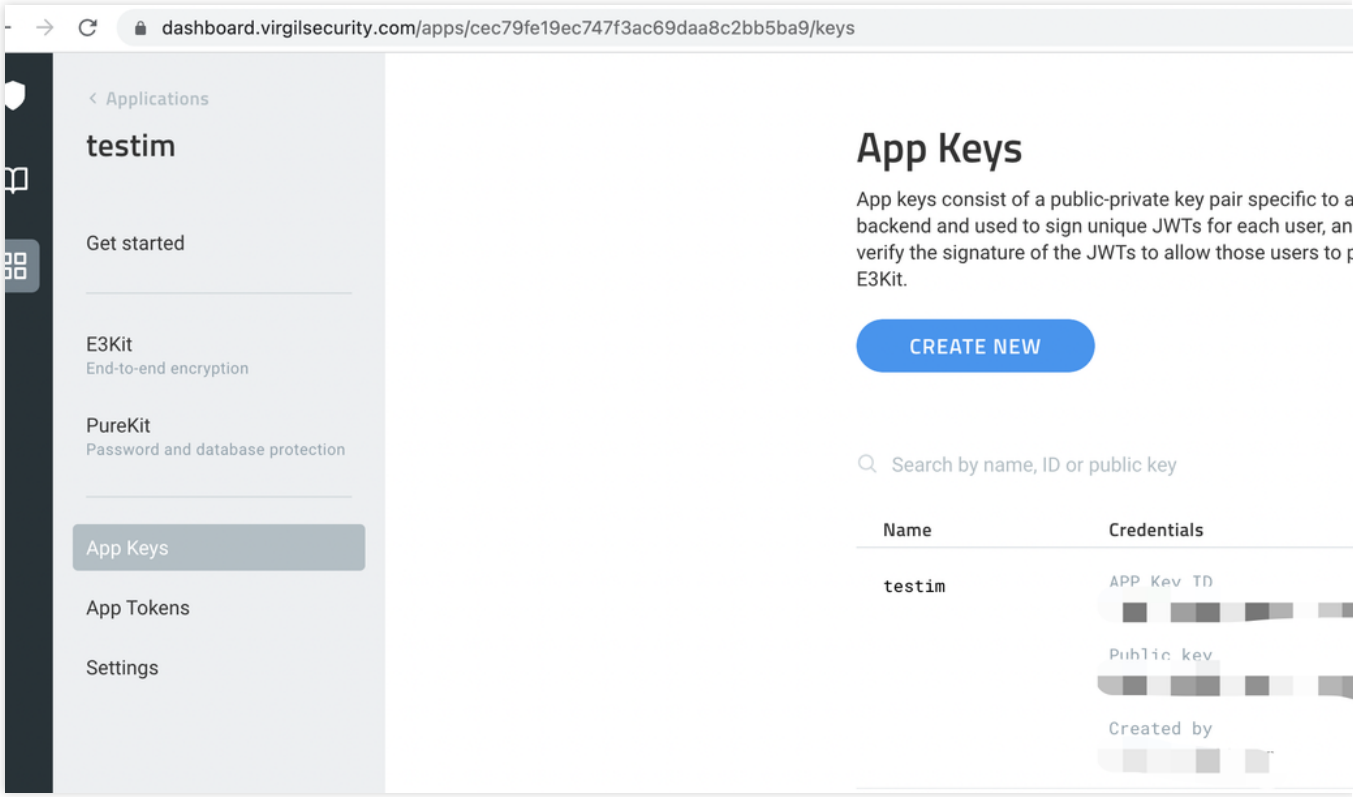
End-to-end encrypted chat with Virgil

最終更新日：：2024-02-07 17:30:51

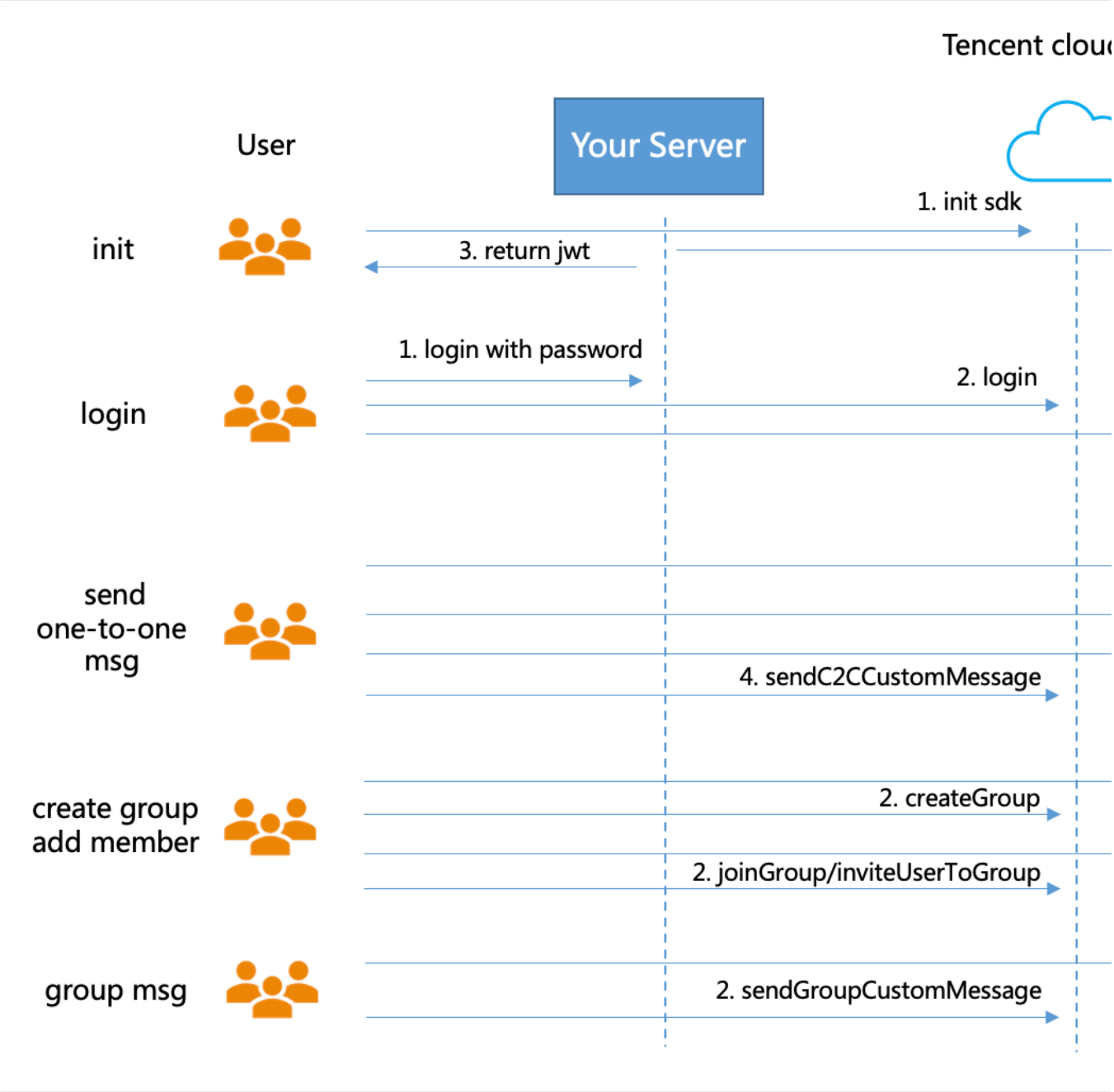
End-to-end encrypted chat with Virgil

Overview

This solution should be implemented by the customer's application based on the E3Kit of virgin security and Tencent Cloud Chat Sdk. The E3Kit document link: Virtual gild E3Kit|Virtual gild Security
Learn about the GroupEncryption-End-to-End-Encryption-E3Kit|Virtual Security documentation before reading the following solutions, especially the JWT token（JSON Web Token）, create channel/create group, encrypt and decrypt messages.
Open application in virgin security console before using. This product is a paid product. See Pricing|Virgin Security for specific pricing.



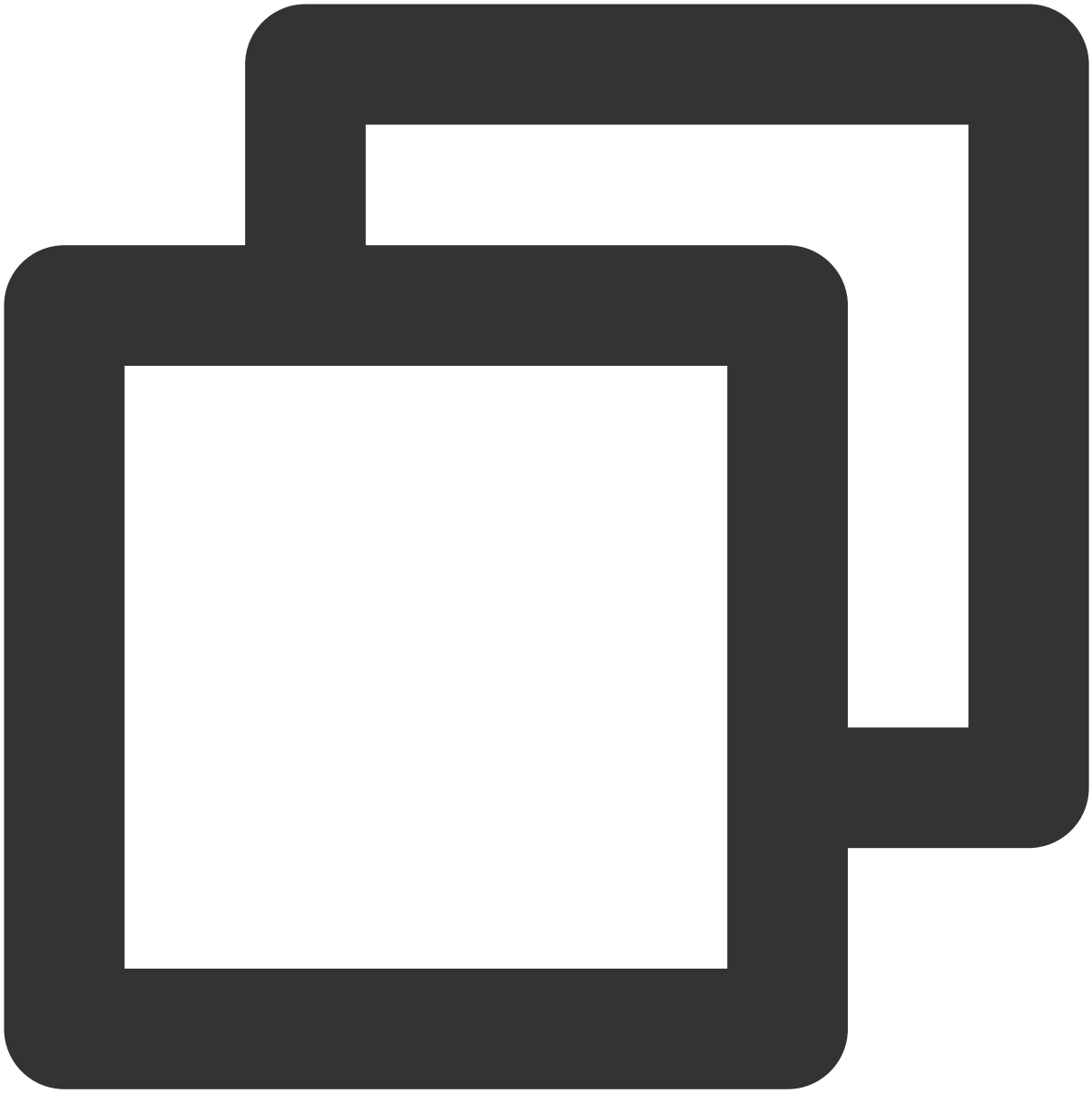
The diagram below shows the high-level communication flow：



Broad implementation overview(example:Android)

Initialization

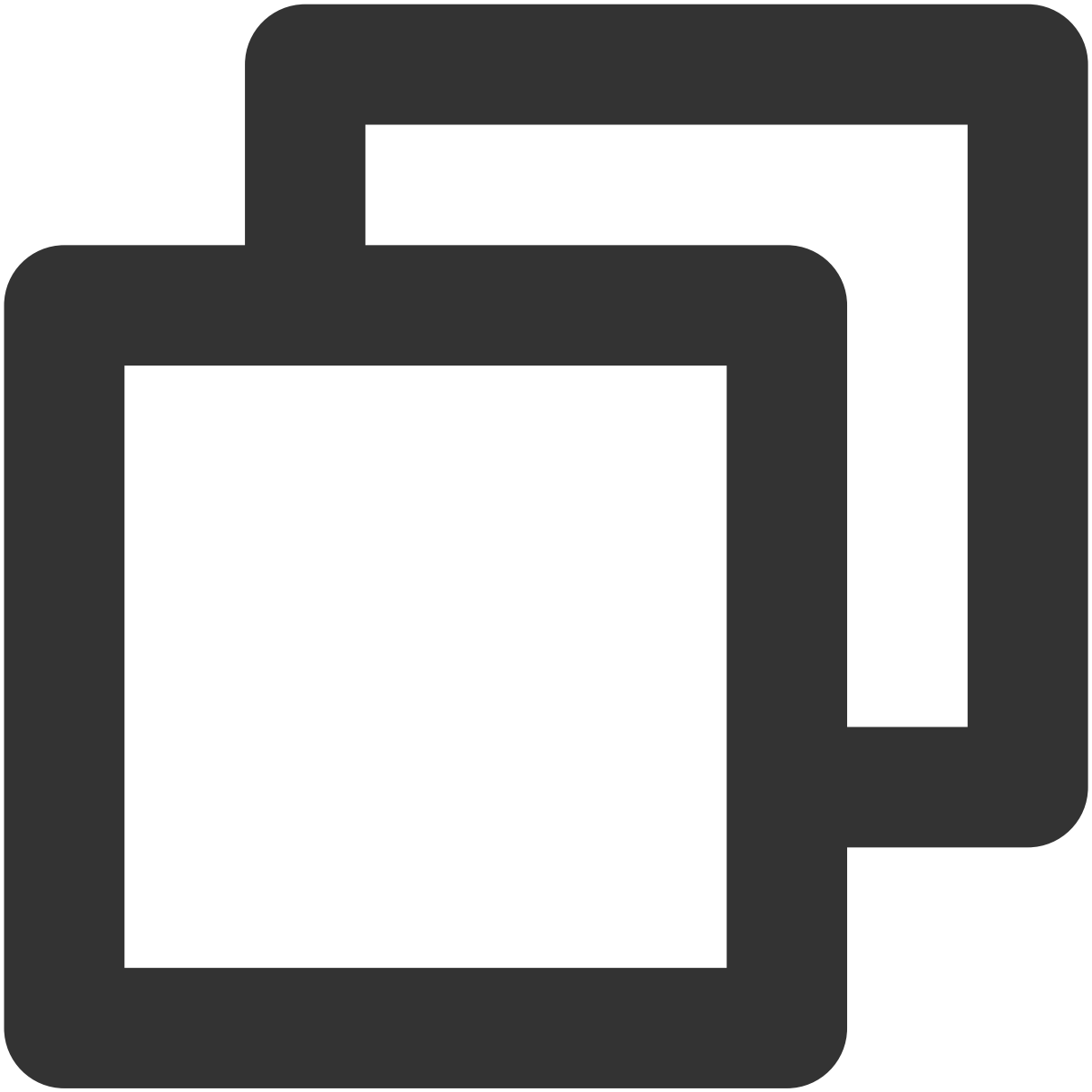
1.Tencent Cloud imsdk initialization



```
// 1. Get the `SDKAppID` from the Chat console.
// 2. Initialize the `config` object.
V2TIMSDKConfig config = new V2TIMSDKConfig();
// 3. Specify the log output level.
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_INFO);
// 4. Add the `V2TIMSDKListener` event listener. `sdkListener` is the implementatio
V2TIMManager.getInstance().addIMSDKListener(sdkListener);
// 5. Initialize the IM SDK. You can call the login API as soon as you call this AP
V2TIMManager.getInstance().initSDK(context, sdkAppID, config);
```

2.E3Kit initialization, passing in the console configuration information, and generating jwt. Corresponding operation document link: GenerateClientTokens-GetStarted-E3Kit | VirgilSecurity

The server generates jwttoken and sends it to the client



```
// generate 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();

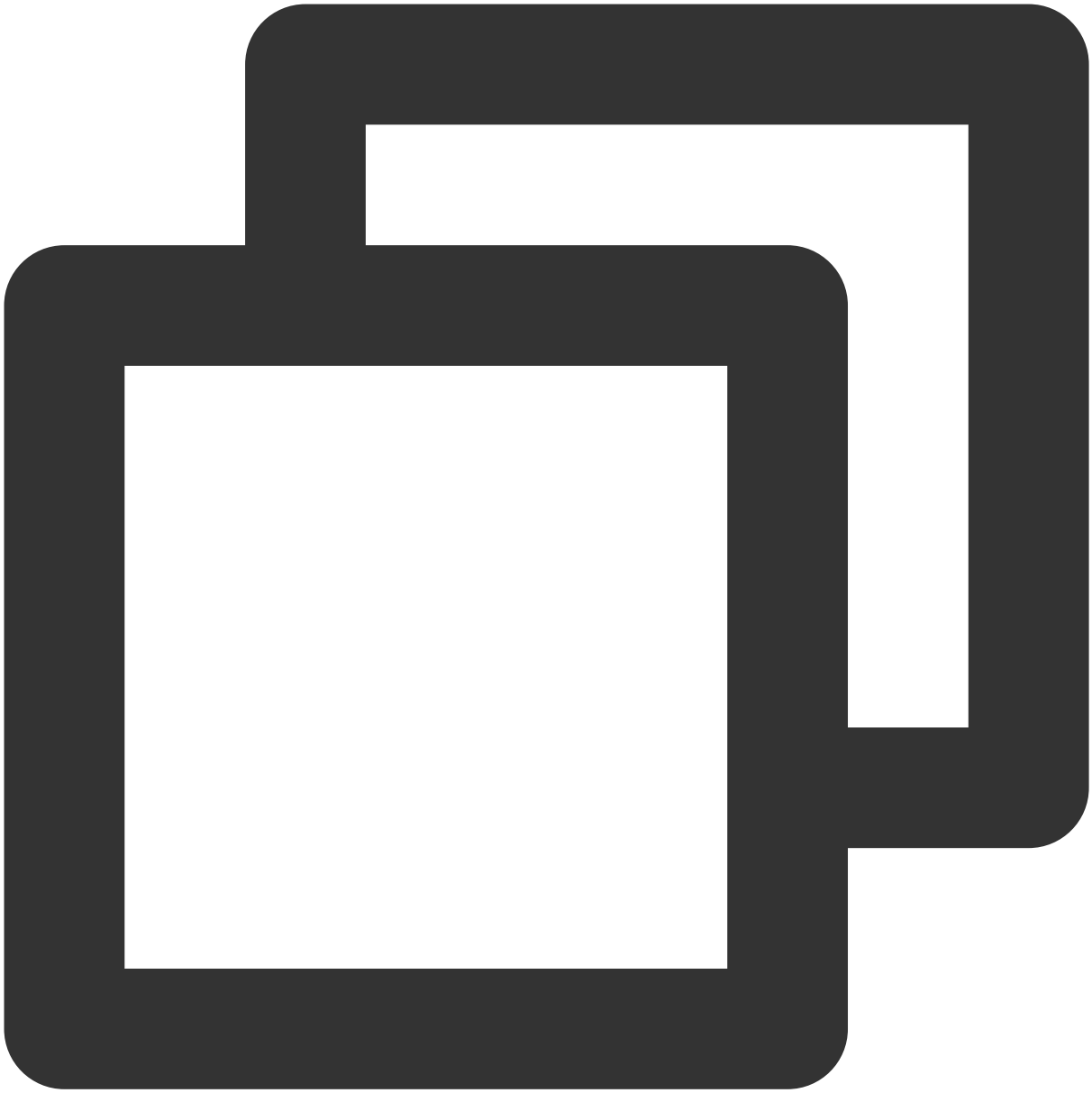
// Use your App Credentials you got at the Virgil Dashboard:
String appId = "be00e10e4e1f4bf58f9b4dc85d79c77a";
String appKeyId = "70b447e321f3a0fd";
TimeSpan ttl = TimeSpan.fromTime(1, TimeUnit.HOURS); // 1 hour - JWT's lifetime

// 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();
```

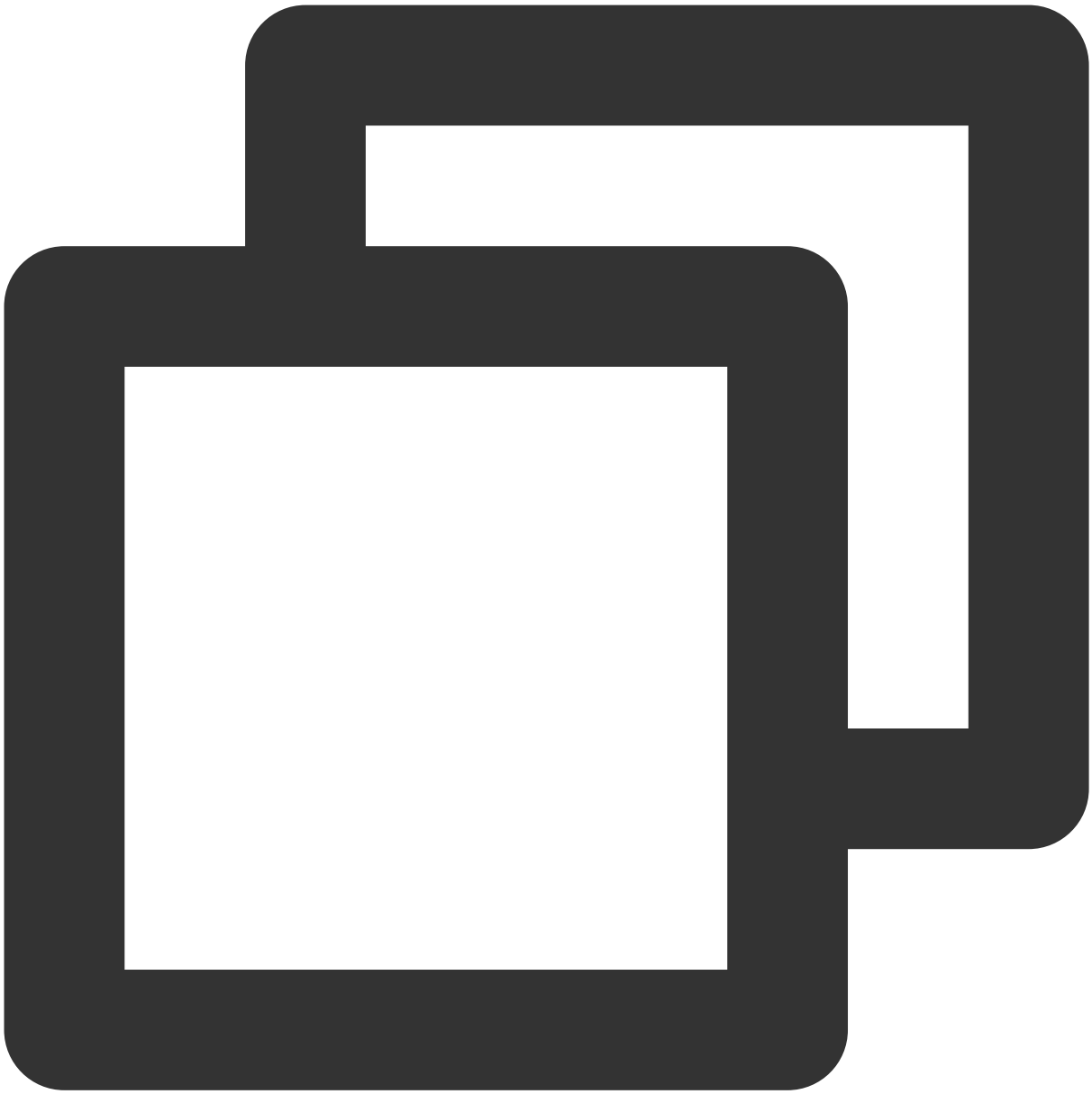
The Android client initializes the E3Kit, tokenCallback is the jwttoken returned by the above server, and User1 is the user ID



```
// initialization 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);
```

User Login

1.Call Tencent Cloud Chat Sdk login method for account login



```
String userID = "your user id";
//Generating UserSig | Tencent Cloud
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) {
        // The following error codes indicate an expired `userSig`, and you need to
        // 1. ERR_USER_SIG_EXPIRED (6206)
        // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)
        // Note: Do not call the login API in case of other error codes; otherwise,
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

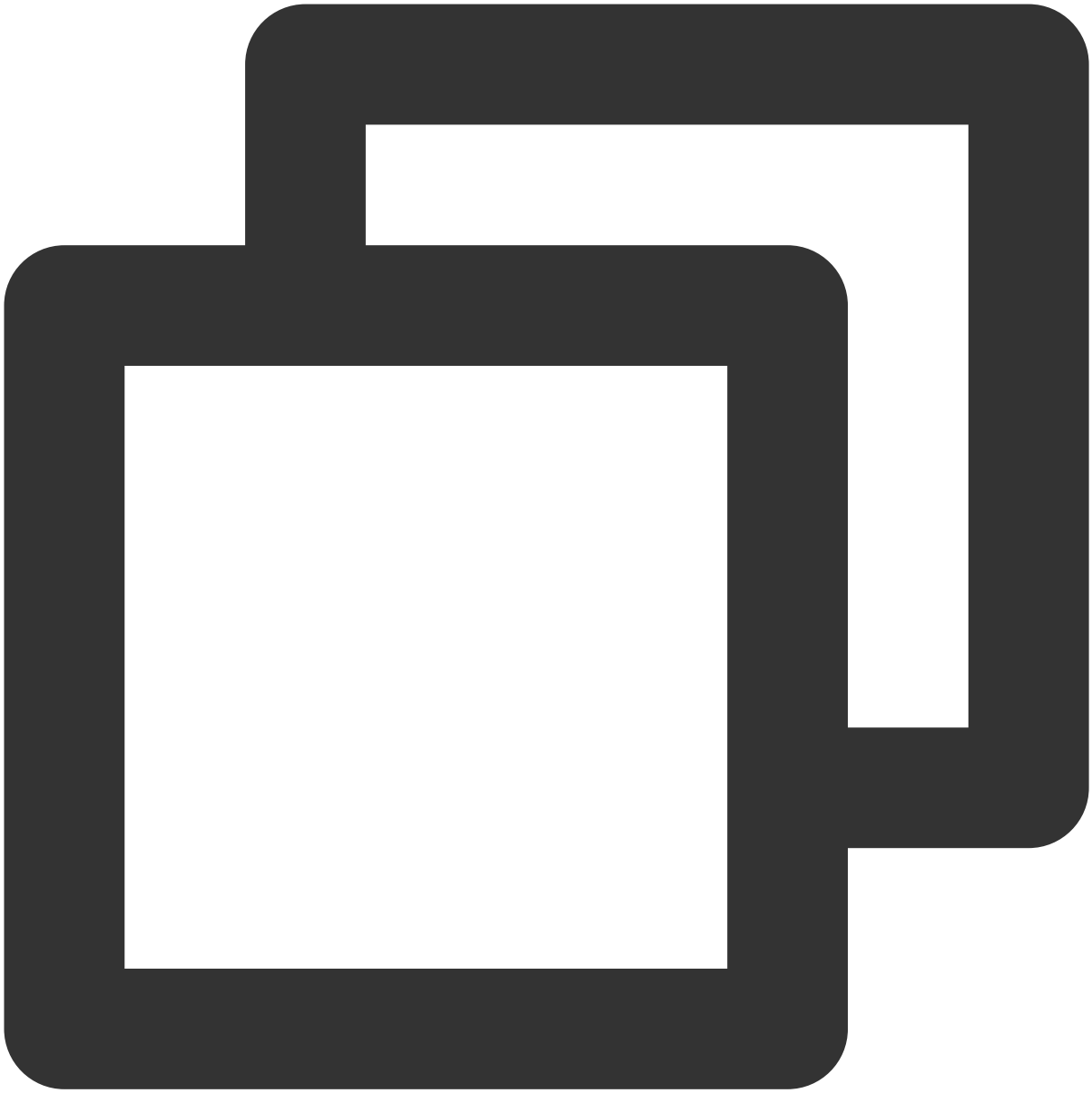
2.Call the eThree.register method to register the user to virgilsecurity. The corresponding operation document link: UserAuthentication-E3Kit | VirgilSecurity

Note：User of im_ ID and user registered on virtilsecurity_ The id should be consistent

Start a one-to-one chat

1.Call the ethree.createRatchetChannel method in the E3Kit to create a one to one session (User1 and User2), and the corresponding operation document link: https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/double-ratchet/?#create -channel

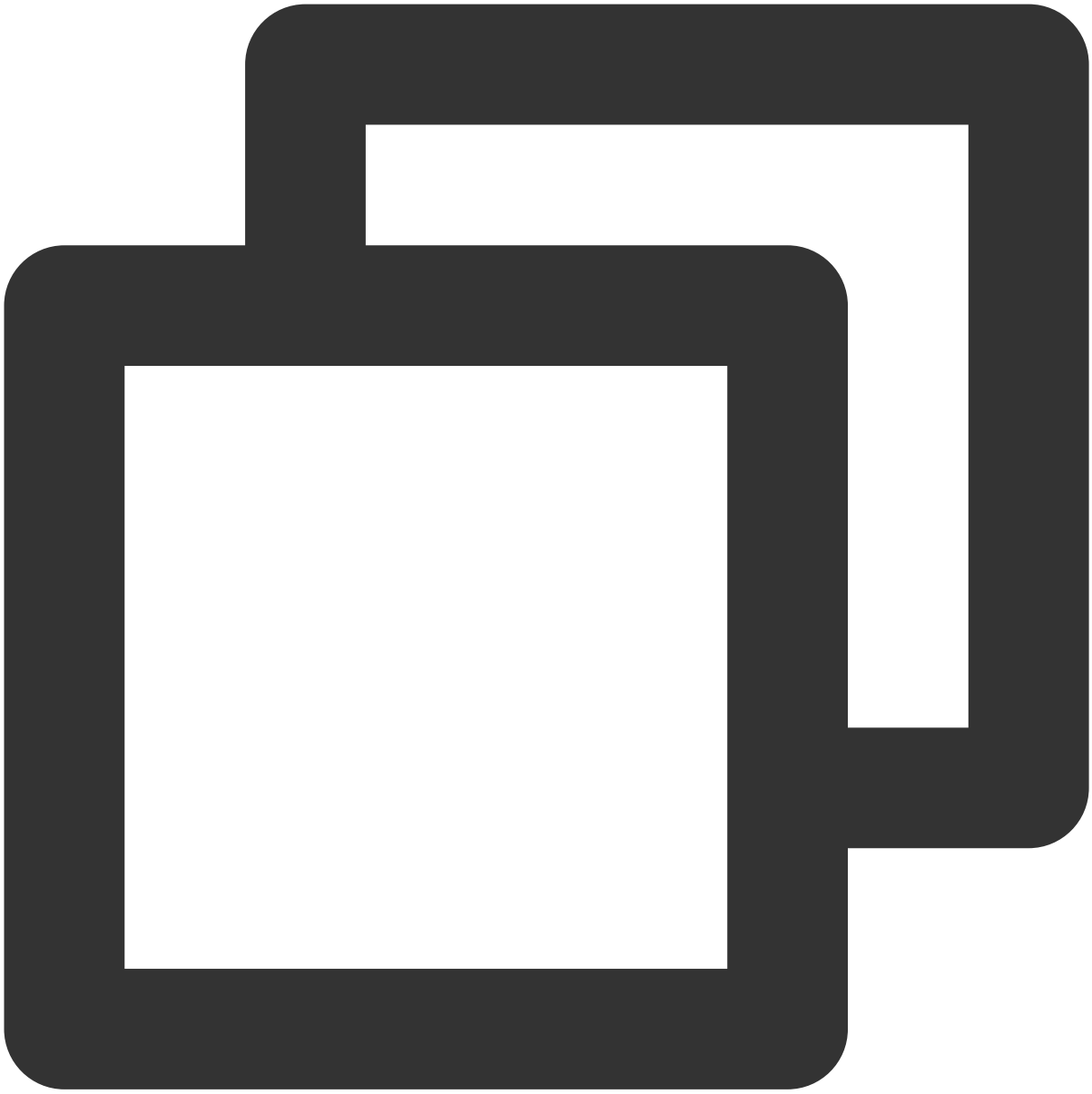
User1 creates a channel with User2



```
// create 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
          }
      });
```

User2 can join the channel



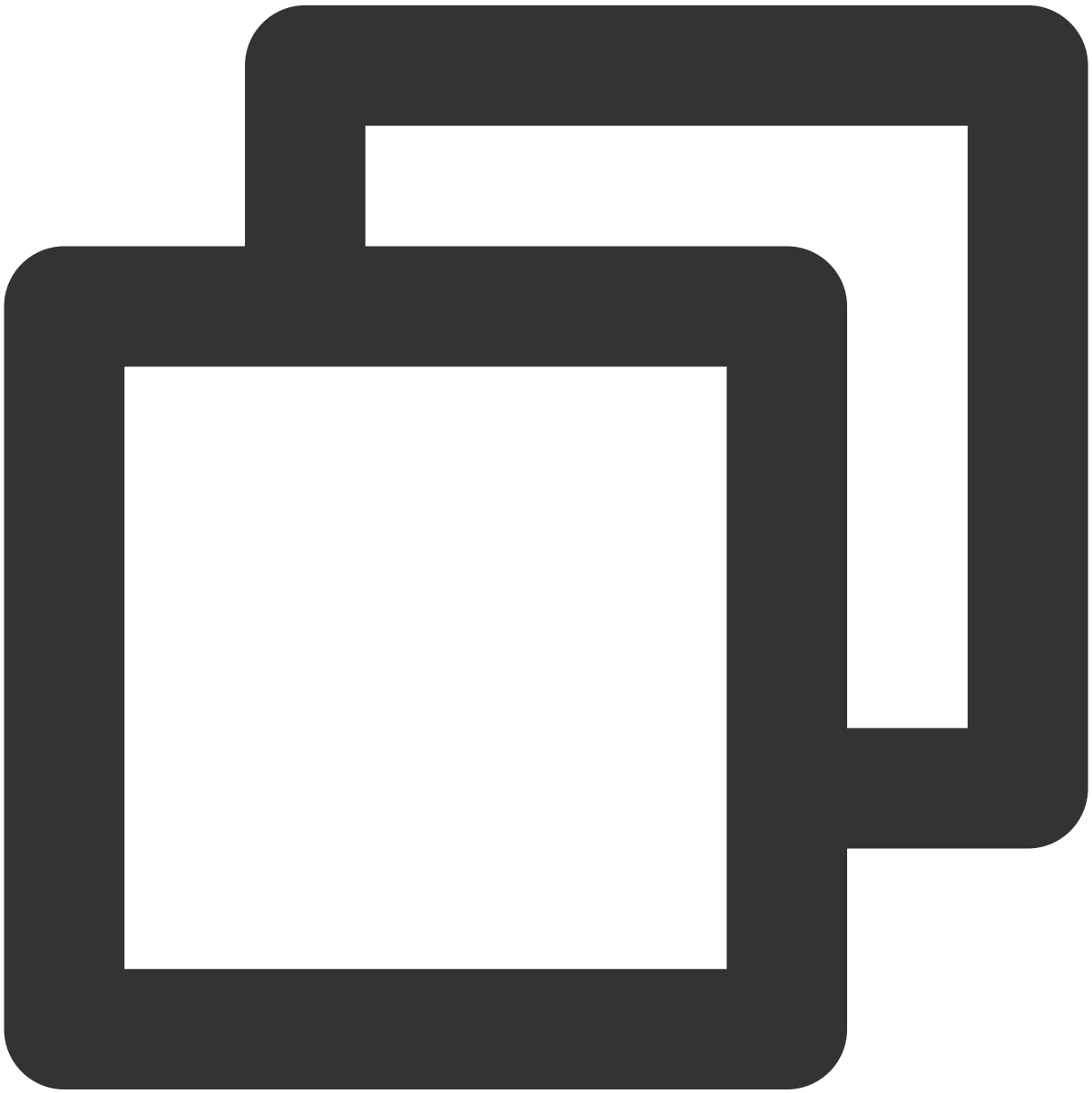
```
// join channel
ethree.joinRatchetChannel(users.get("User1"))
      .addCallback(new OnResultListener<RatchetChannel>() {
    @Override public void onSuccess(RatchetChannel ratchetChannel) {
        // Channel joined and saved locally!
    }

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

```
    }
});
```

One-to-one chat message encryption and decryption

1.Use the channel created above to encrypt messages and link documents：
https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/double-ratchet/?#encrypt-and-decrypt-messages

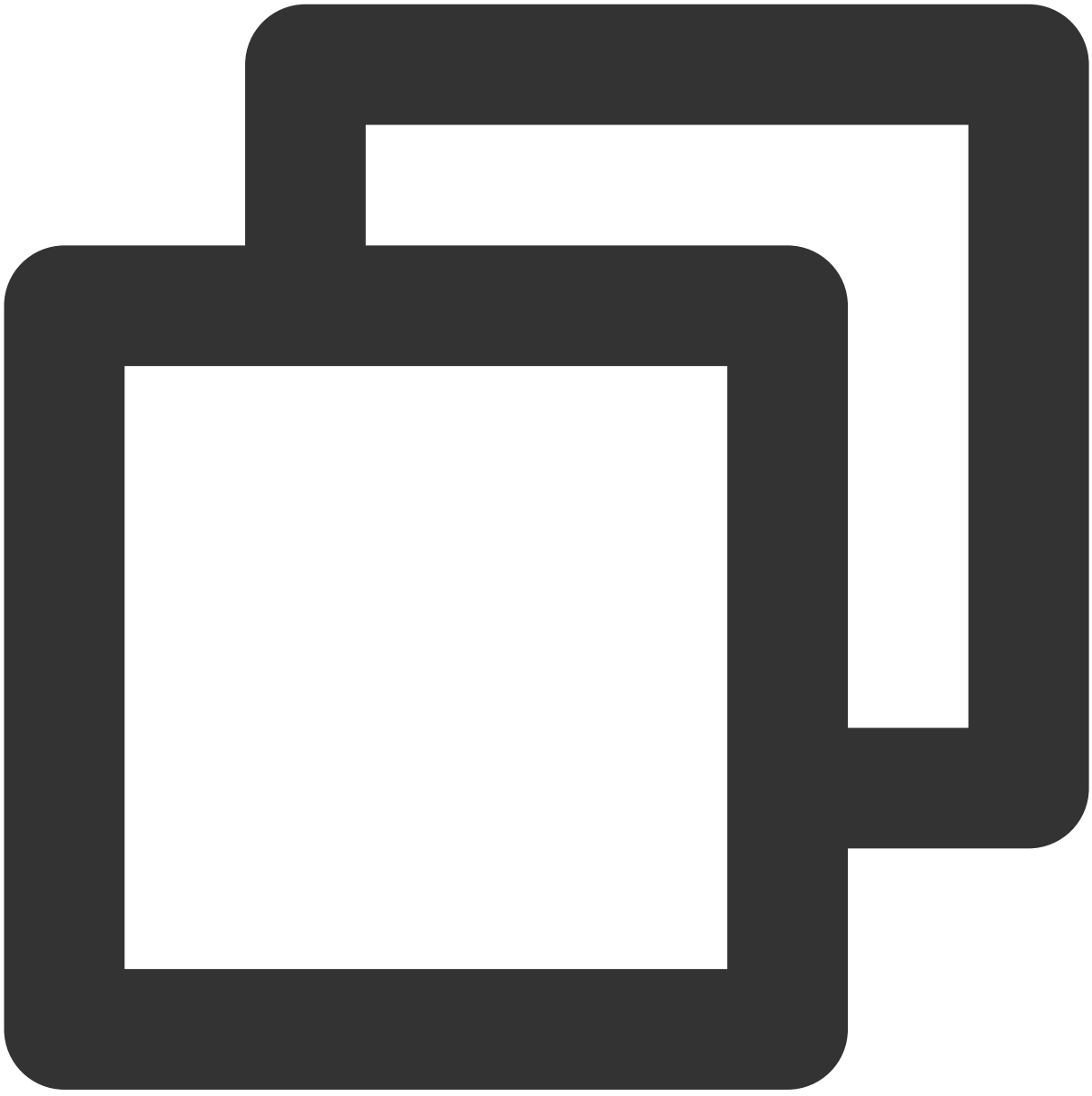


```
// one-to-one chat message encryption
// prepare a message
```

```
String messageToEncrypt = "Hello, User2!";

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

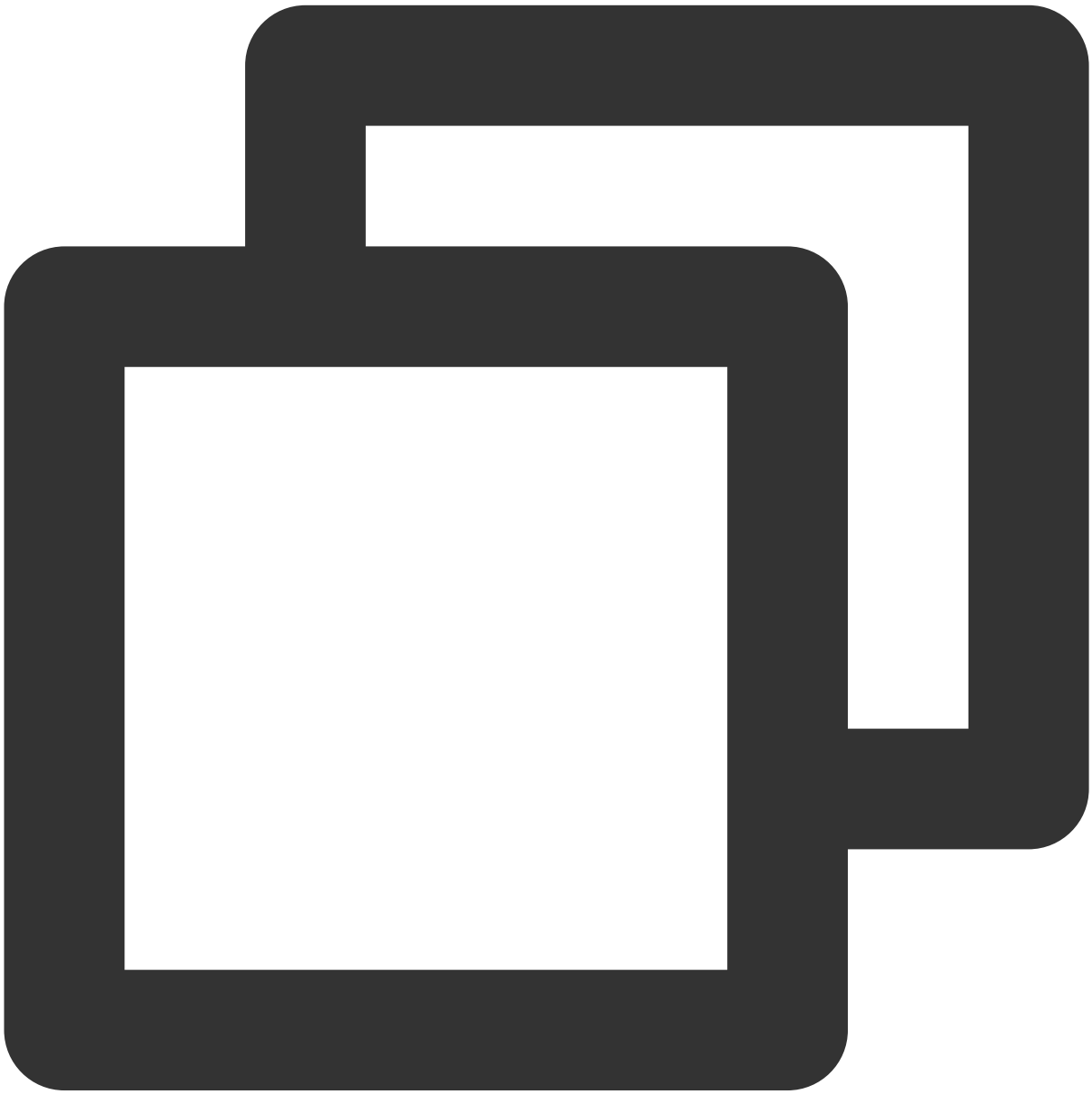
2.The encrypted message content is sent to Tencent Cloud Chat Sdk and sent with a customized message



```
// `msgID` returned by the API for on-demand use
String msgID = V2TIMManager.getInstance().sendC2CCustomMessage("virgil encrypted ms
@Override
public void onSuccess(V2TIMMessage message) {
    // The one-to-one text message sent successfully
}
```

```
    @Override
public void onError(int code, String desc) {
    // Failed to send the one-to-one text message
}
});
```

3.After the peer receiving the customized message

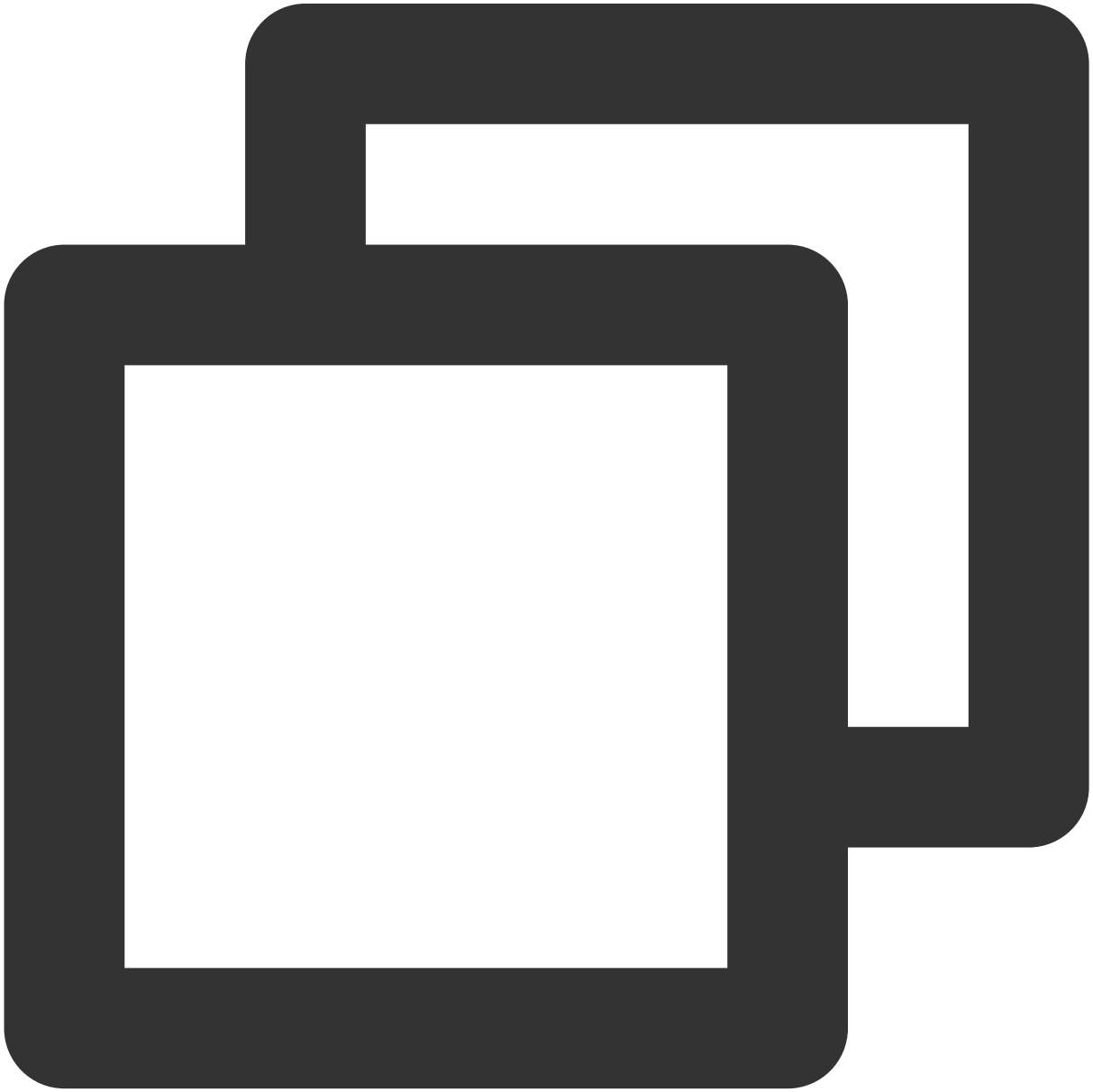


```
// Set the event listener
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);

/**
```

```
* Receive the custom one-to-one message
* @param msgID Message ID
* @param sender Sender information
* @param customData The sent content
*/
public void onRecvC2CCustomMessage(String msgID, V2TIMUserInfo sender, byte[] custo
Log.i("onRecvC2CCustomMessage", "msgID:" + msgID + ", from:" + sender.getNickName()
//call E3Kit to decrypt msg
}
```

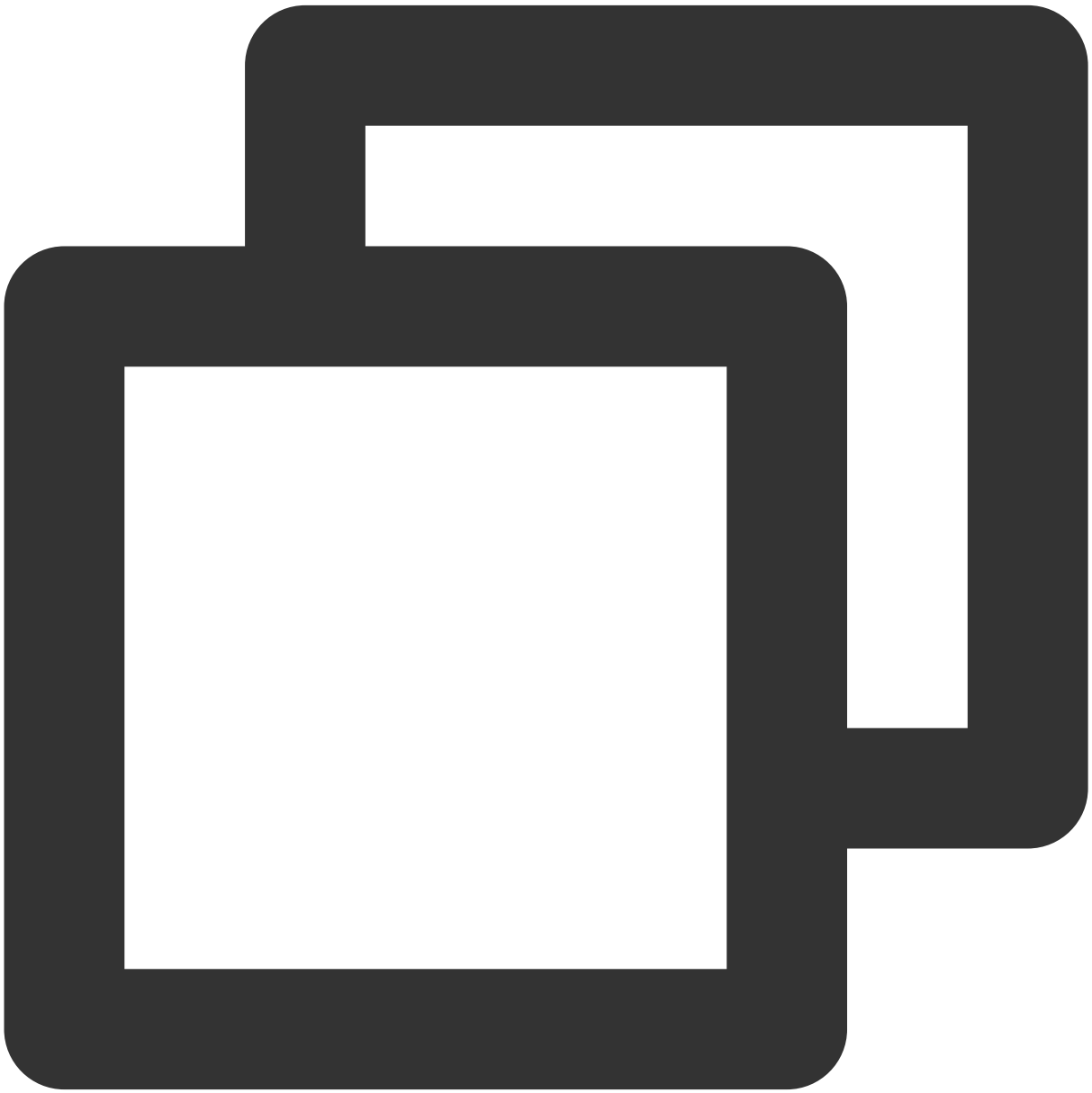
4.the peer calls E3Kit to decrypt and render it, as follows:



```
// Decrypt message
String decrypted = channel.decrypt(encrypted);
```

Start a channel(group)

1.Call the ethree.createGroup to create group， document link：https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#create-group-chat

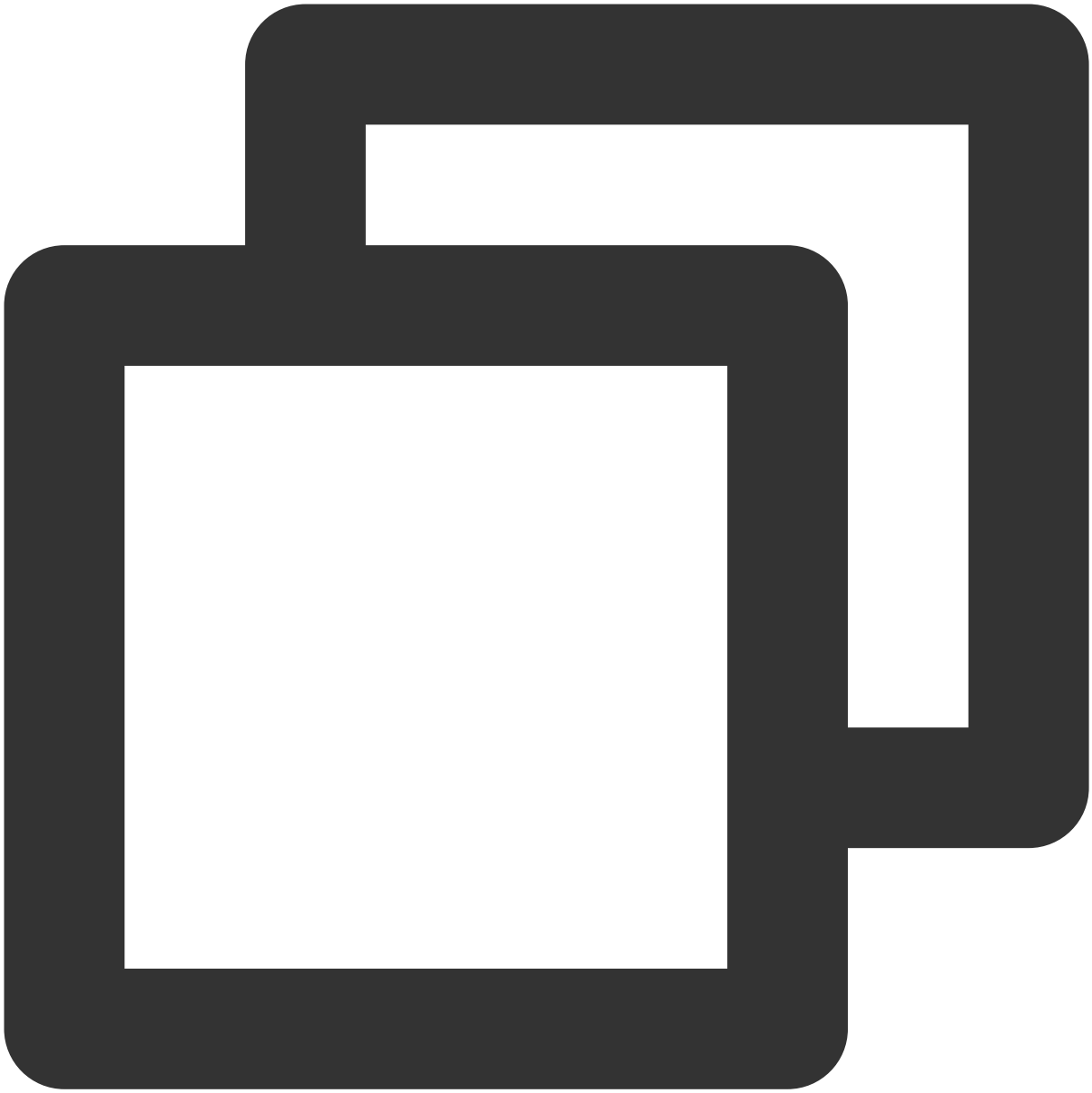


```
// create group
ethree.createGroup(groupId, users).addCallback(new OnResultListener<Group>() {
```

```
    @Override public void onSuccess(Group group) {
        // Group created and saved locally!
    }

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

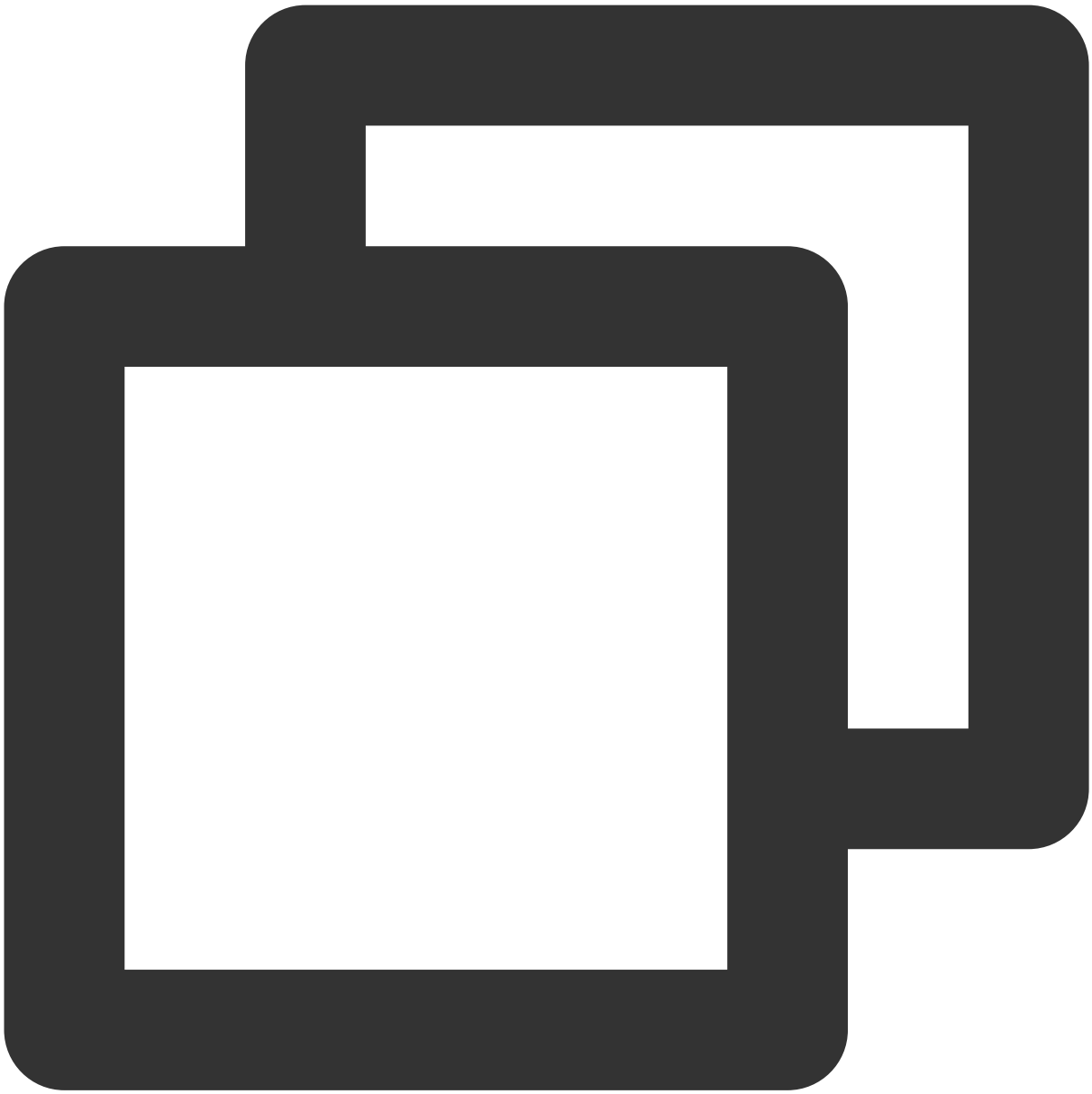
2.If you need to add group members, the code below is as follows. You can call the remove method to delete a group member. Document link：https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#add-new-participant



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

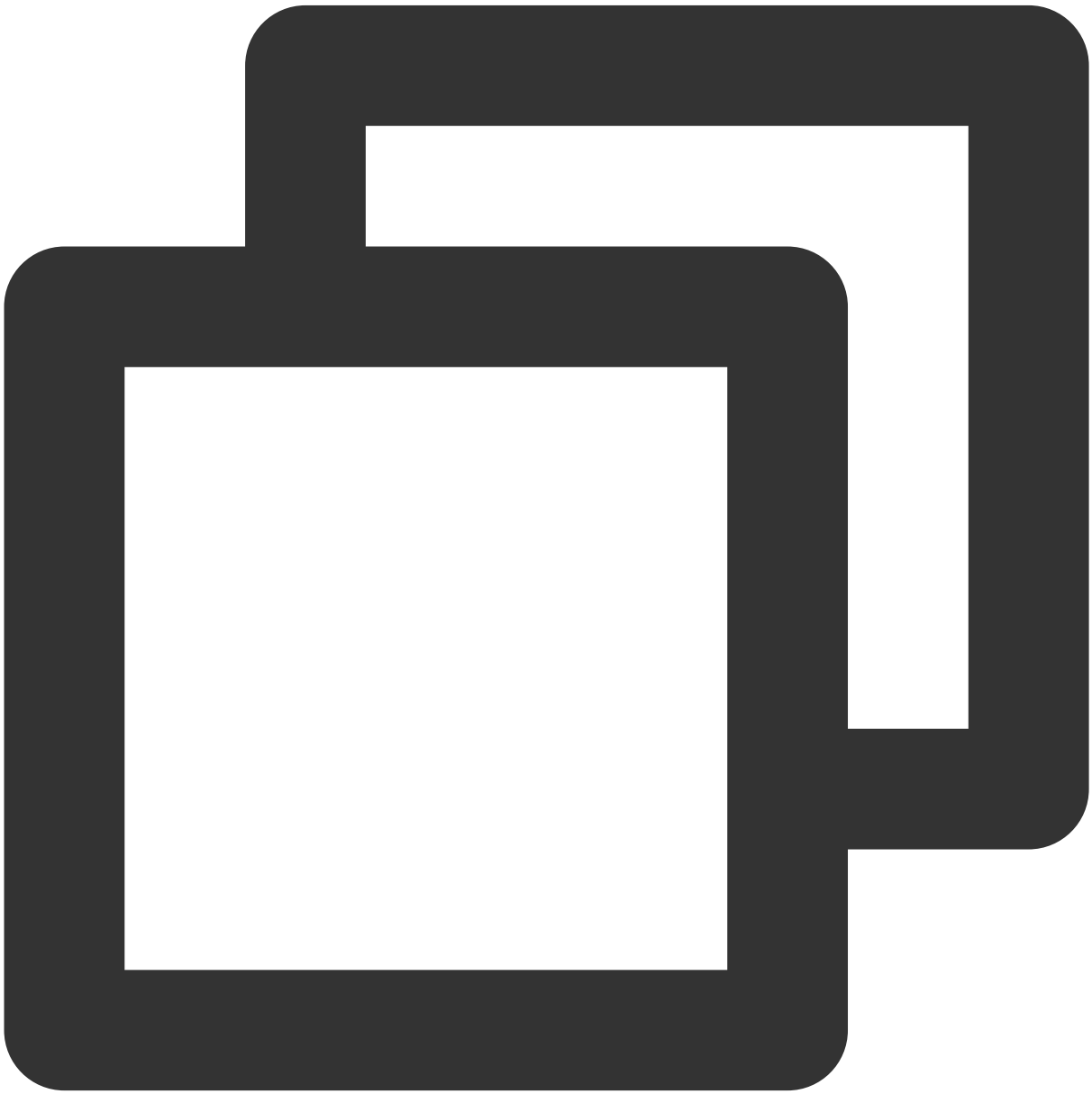
    @Override public void onError(@NotNull Throwable throwable) {
        // Error handling
    }
});
```

3.Call createGroup of Tencent Cloud Chat Sdk to create a group, joinGroup or inviteUserToGroup to add group members



```
V2TIMManager.getInstance().createGroup(V2TIMManager.GROUP_TYPE_WORK, null, "groupA"
 @Override
 public void onSuccess(String s) {
     // Group created successfully
 }
  @Override
 public void onError(int code, String desc) {
     // Failed to create the group
```

```
 }
});
// Listen for the group creation notification
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
 @Override
 public void onGroupCreated(String groupID) {
     // A group was created. `groupID` is the ID of the created group.
 }
});
```



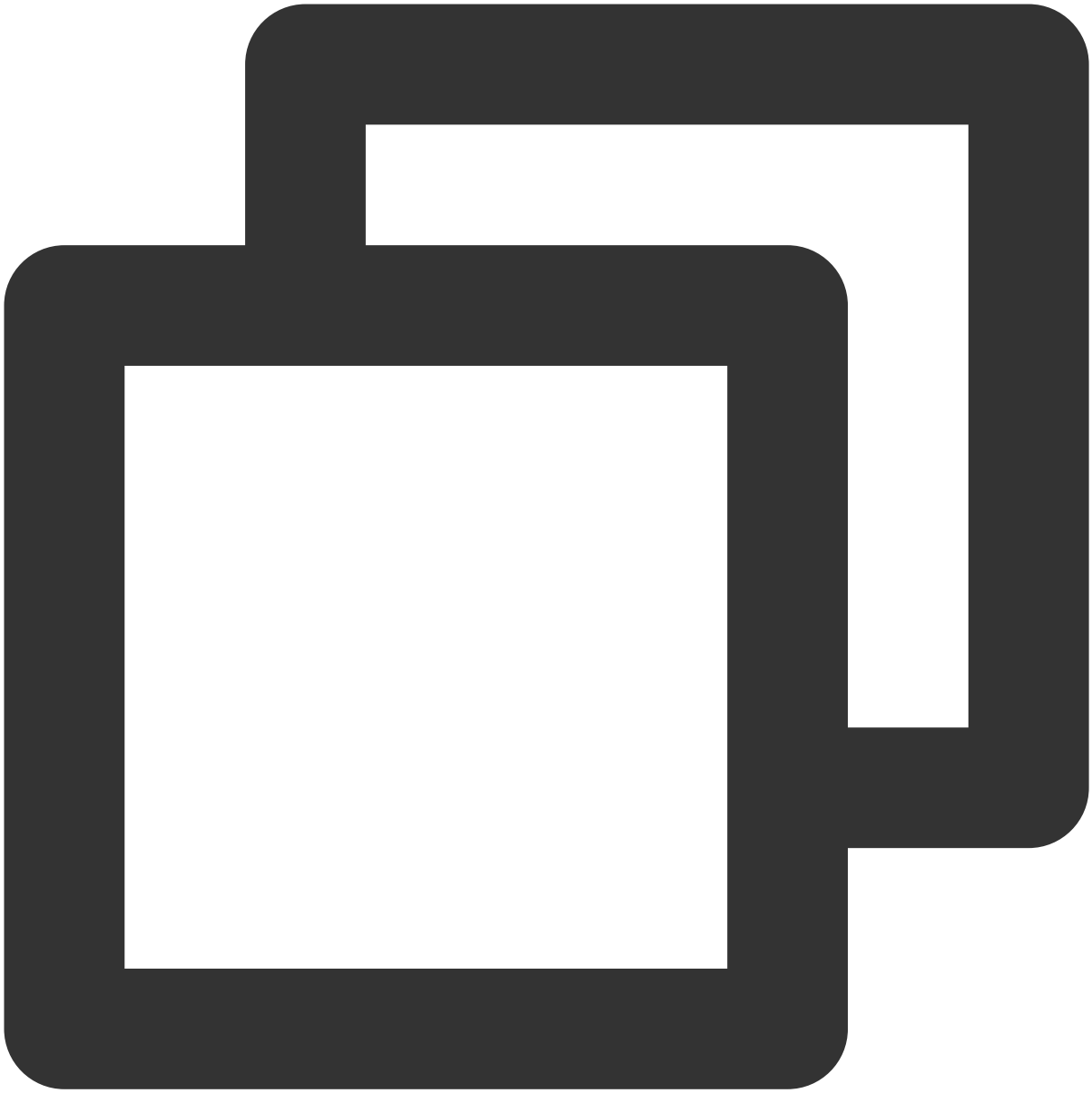
```
// Invite the `userA` user to join the `groupA` group
```

```
List<String> userIDList = new ArrayList<>();
userIDList.add("userA");
V2TIMManager.getGroupManager().inviteUserToGroup("groupA", userIDList, new V2TIMVal
 @Override
 public void onSuccess(List<V2TIMGroupMemberOperationResult> v2TIMGroupMemberOperat
     // Invited the user to the group successfully
 }
  @Override
 public void onError(int code, String desc) {
     // Failed to invite the user to the group
 }
});
// Listen for the group invitation event
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
 @Override
 public void onMemberInvited(String groupID, V2TIMGroupMemberInfo opUser, List<V2TI
     // A user was invited to the group. This callback can contain some UI tips.
 }
});
```

Group chat message encryption and decryption

1.Use the group created above to encrypt messages and link documents：

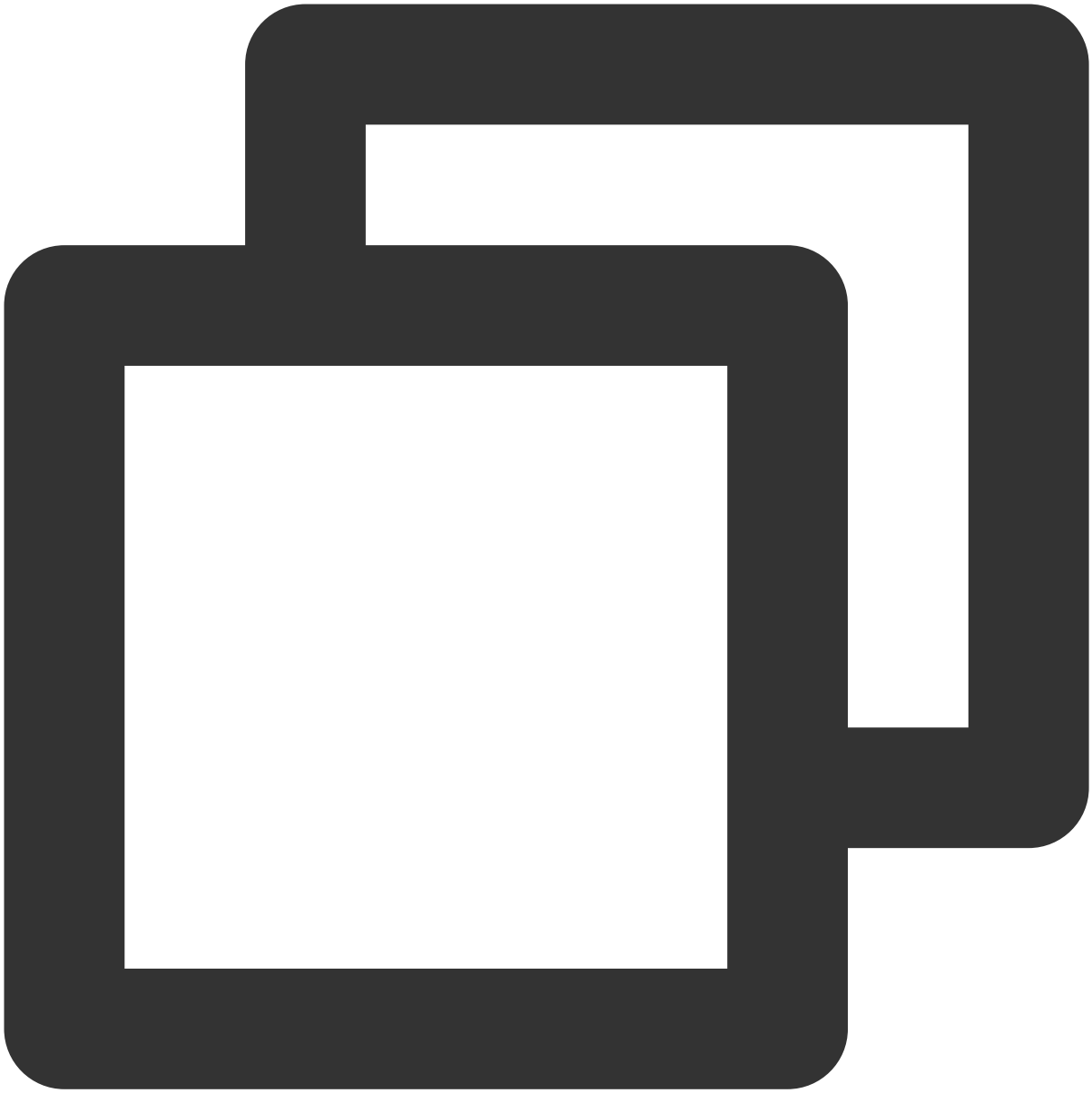
https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#encrypt-and-decrypt-messages



```
//Group message encryption
// prepare a message
String messageToEncrypt = "Hello, Bob and Carol!";

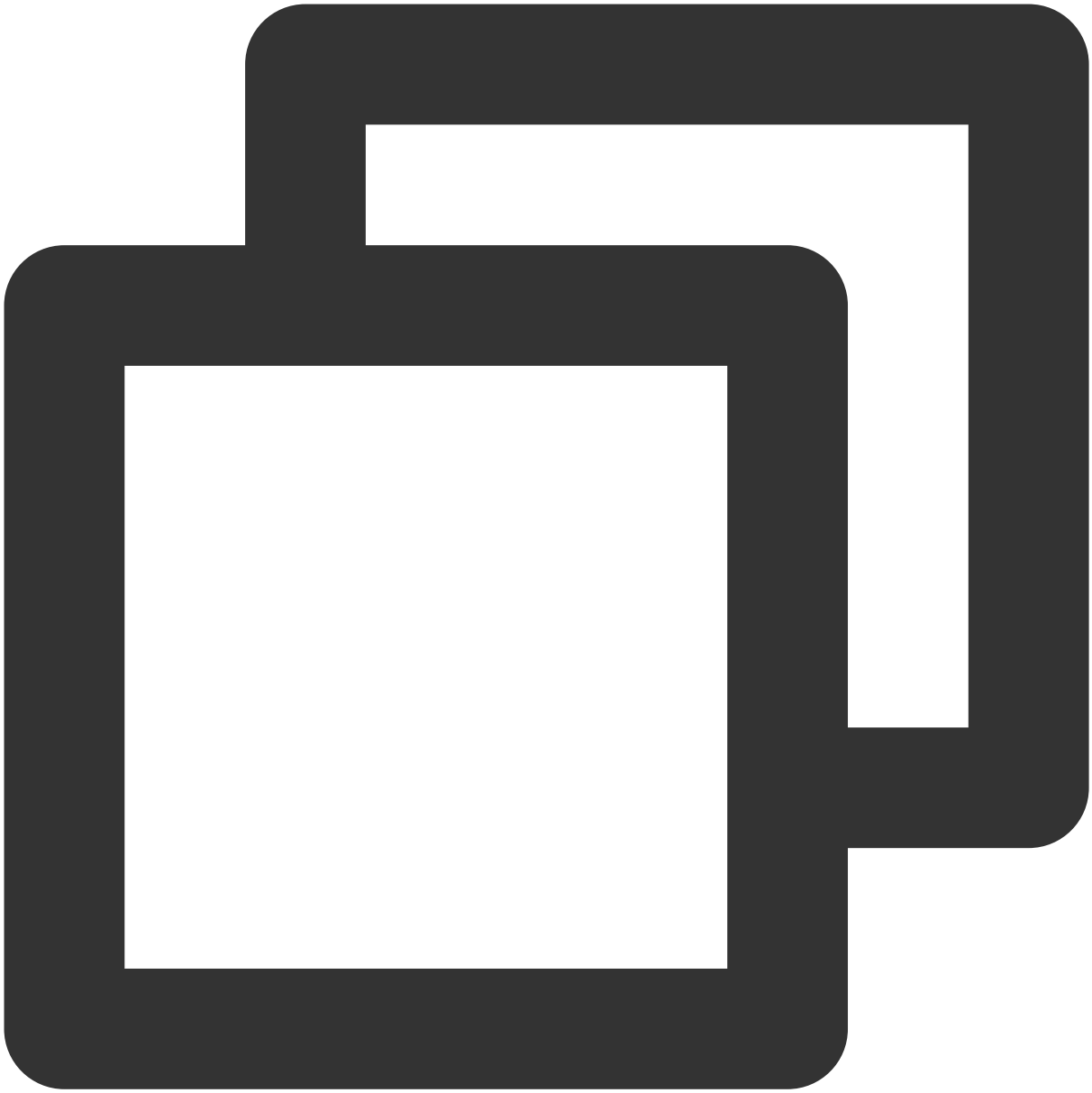
String encrypted = group.encrypt(messageToEncrypt);
```

2.The encrypted message content is sent to tencent Chat Sdk and sent with a customized message



```
String msgID = V2TIMManager.getInstance().sendGroupCustomMessage("virgil encrypted
@Override
public void onSuccess(V2TIMMessage message) {
    // The custom group message sent successfully
}
```

3.After the peer tencent cloud Chat Sdk receiving the customized message,

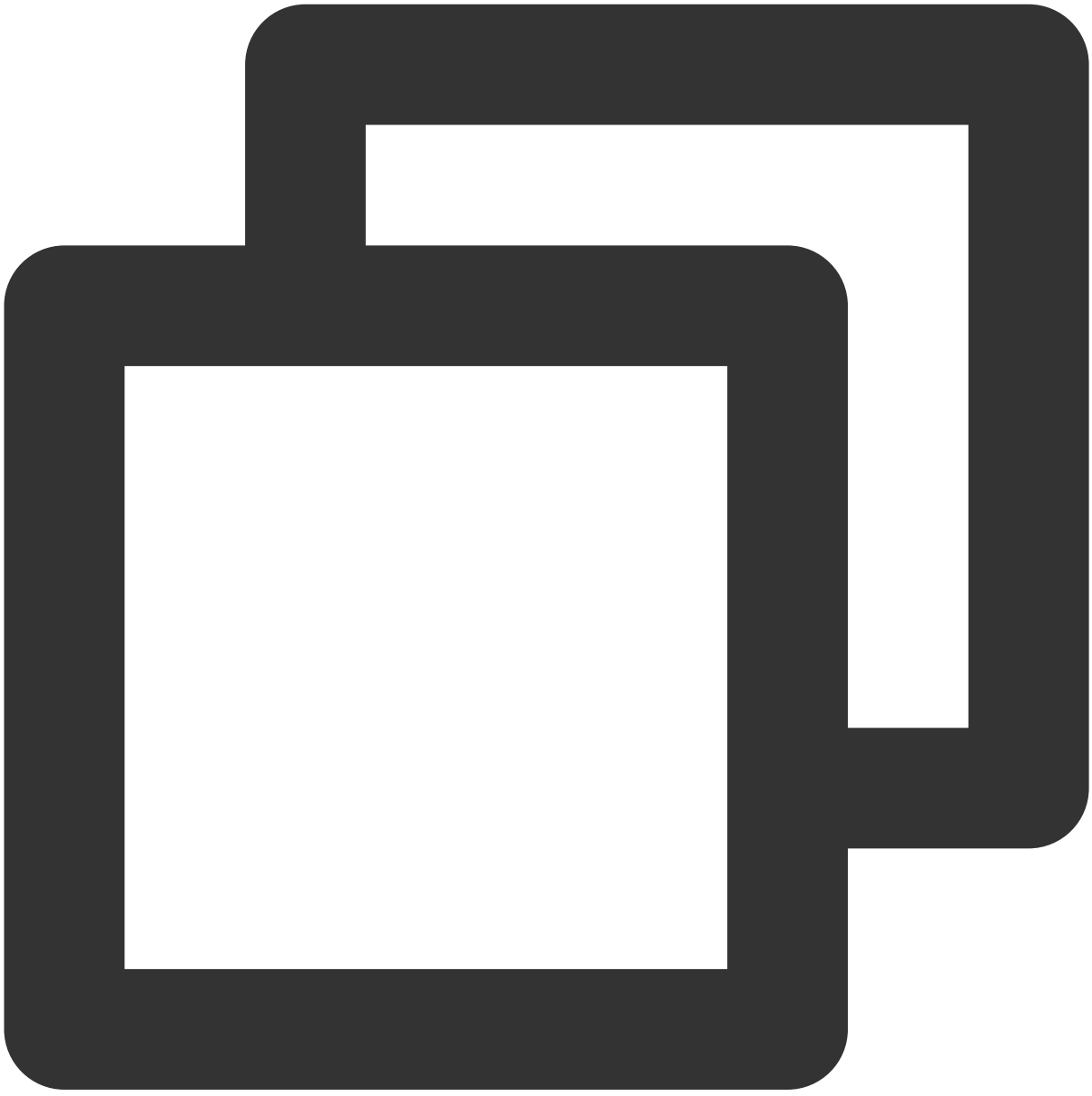


```
// Set the event listener
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);

/**
* Receive the custom group message
* @param msgID Message ID
* @param groupID Group ID
* @param sender The group member information of the sender
* @param customData The sent content
*/
public void onRecvGroupCustomMessage(String msgID, String groupID, V2TIMGroupMember
```

```
Log.i("onRecvGroupCustomMessage", "msgID:" + msgID + ", groupID:" + groupID + ", fr
//call E3Kit to decrypt msg
}
```

4.The peer decrypt and render it, as follows：

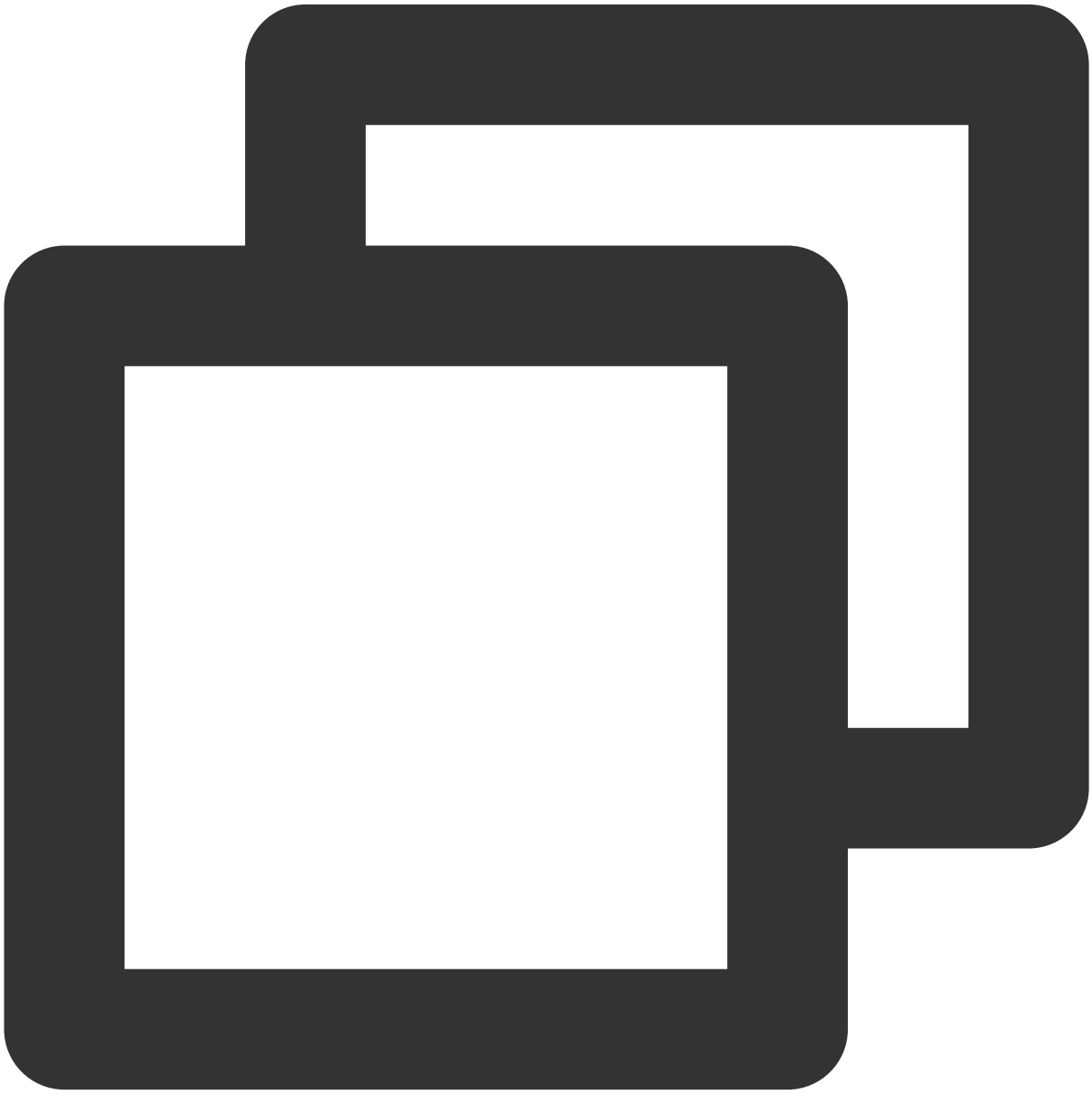


```
//Group message decryption
String decrypted = group.decrypt(encrypted, users.get("Alice"));
```

Note: After using this end-to-end encryption scheme, the local chat record search function of imsdk will not be available

IM - Developer Groups

Join a Tencent Cloud IM developer group for:



```
        ● Reliable technical support
        ● Product details
        ● Constant exchange of ideas
```

Telegram group (EN): join

WhatsApp group (EN): join

Telegram group (ZH): join

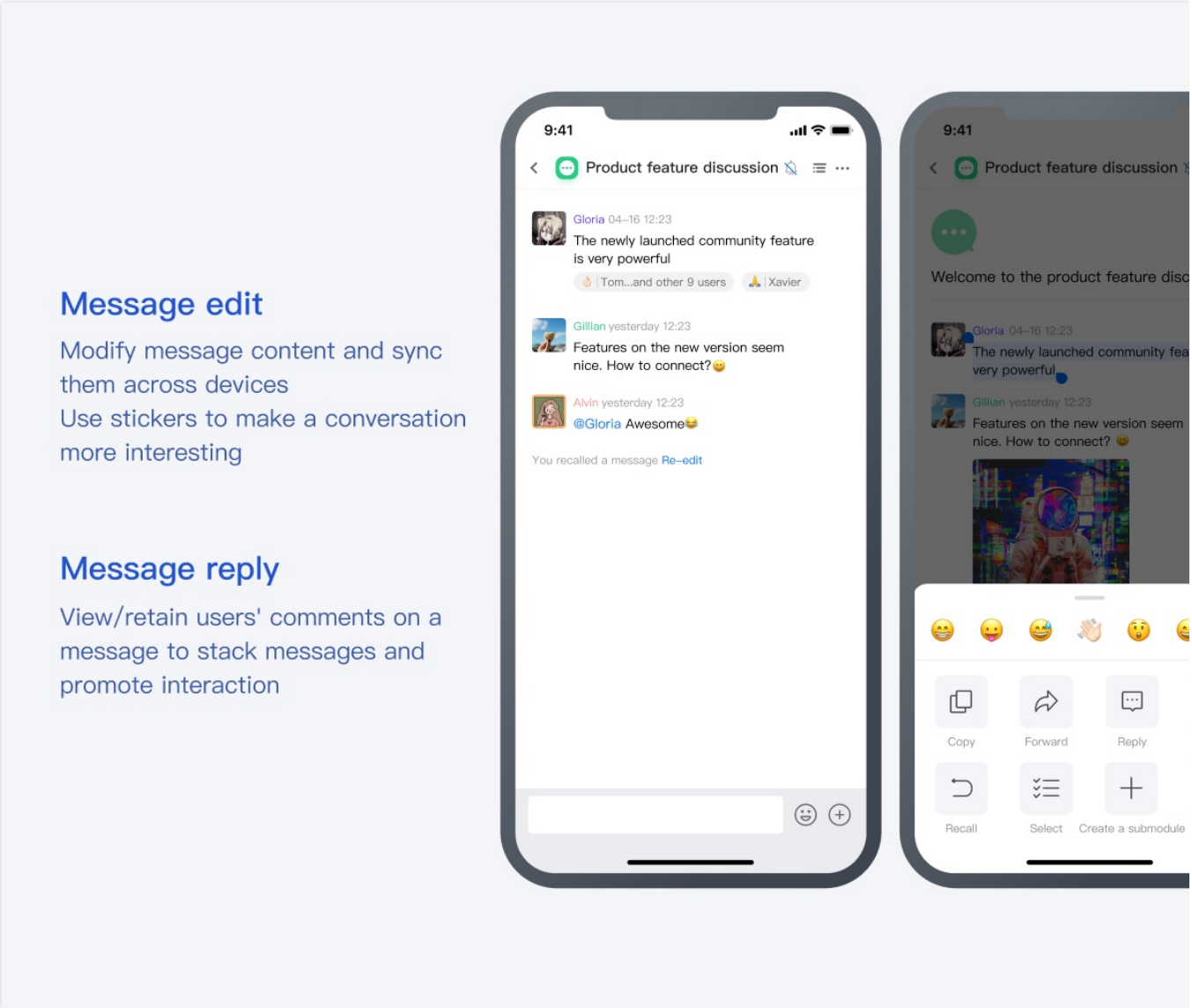
WhatsApp group (ZH): join

極めて大規模なエンターテインメントコラボレーションコミュニティ

最終更新日：：2024-04-11 17:28:06

機能の説明

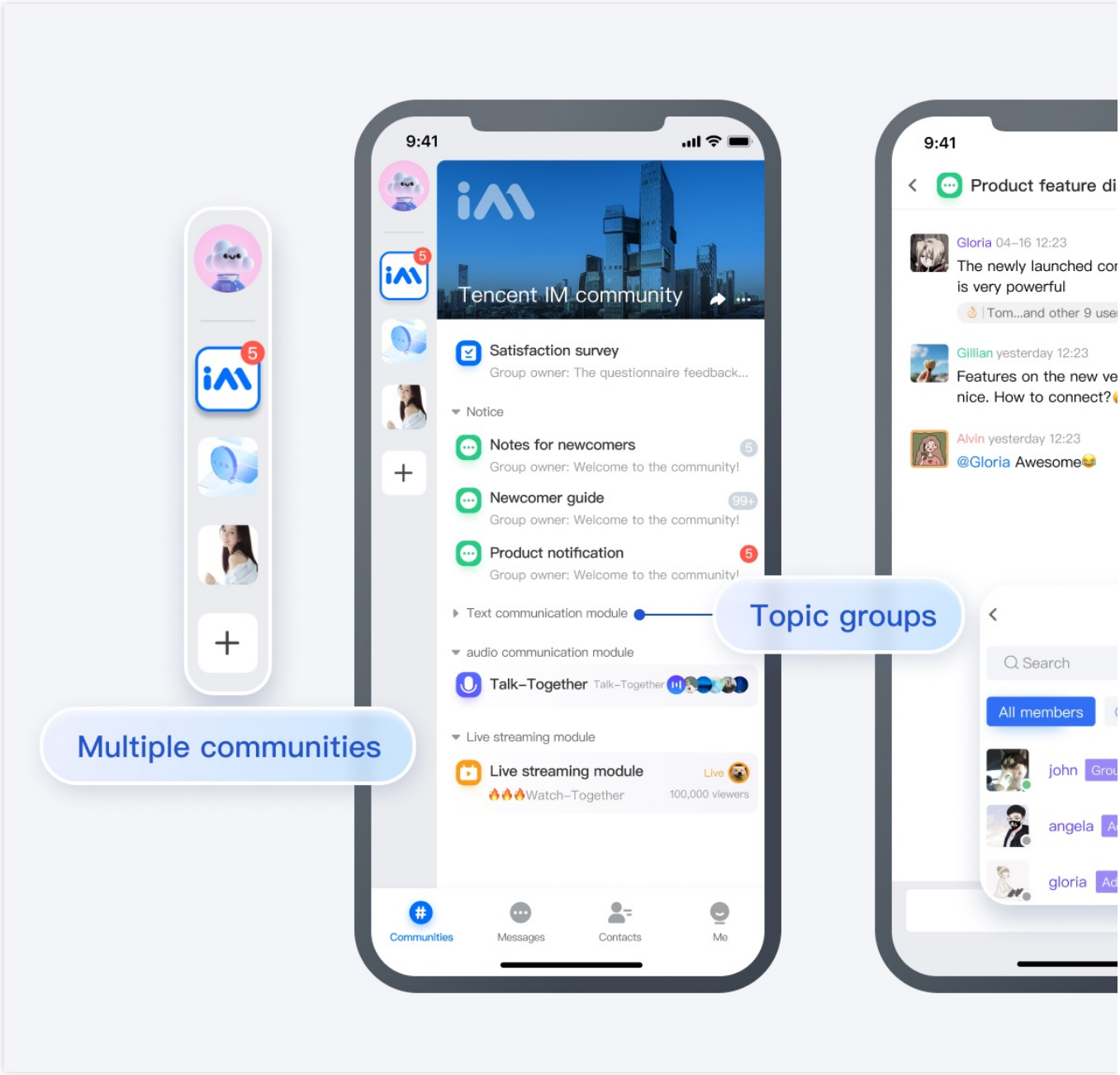
コミュニティモード（エンターテインメントコラボレーションの新しいツール）は、**コミュニティ-分類-トピック**の3つのレベルの構造をサポートし、メッセージを相互に分離するほか、極めて大規模なメンバーを運営し、同じセットの友達関係を使用することができます。また、メンバーをグループ別に分類し、メンバーグループの表示、発言、および管理の権限を設定することもできます。



ユースケース

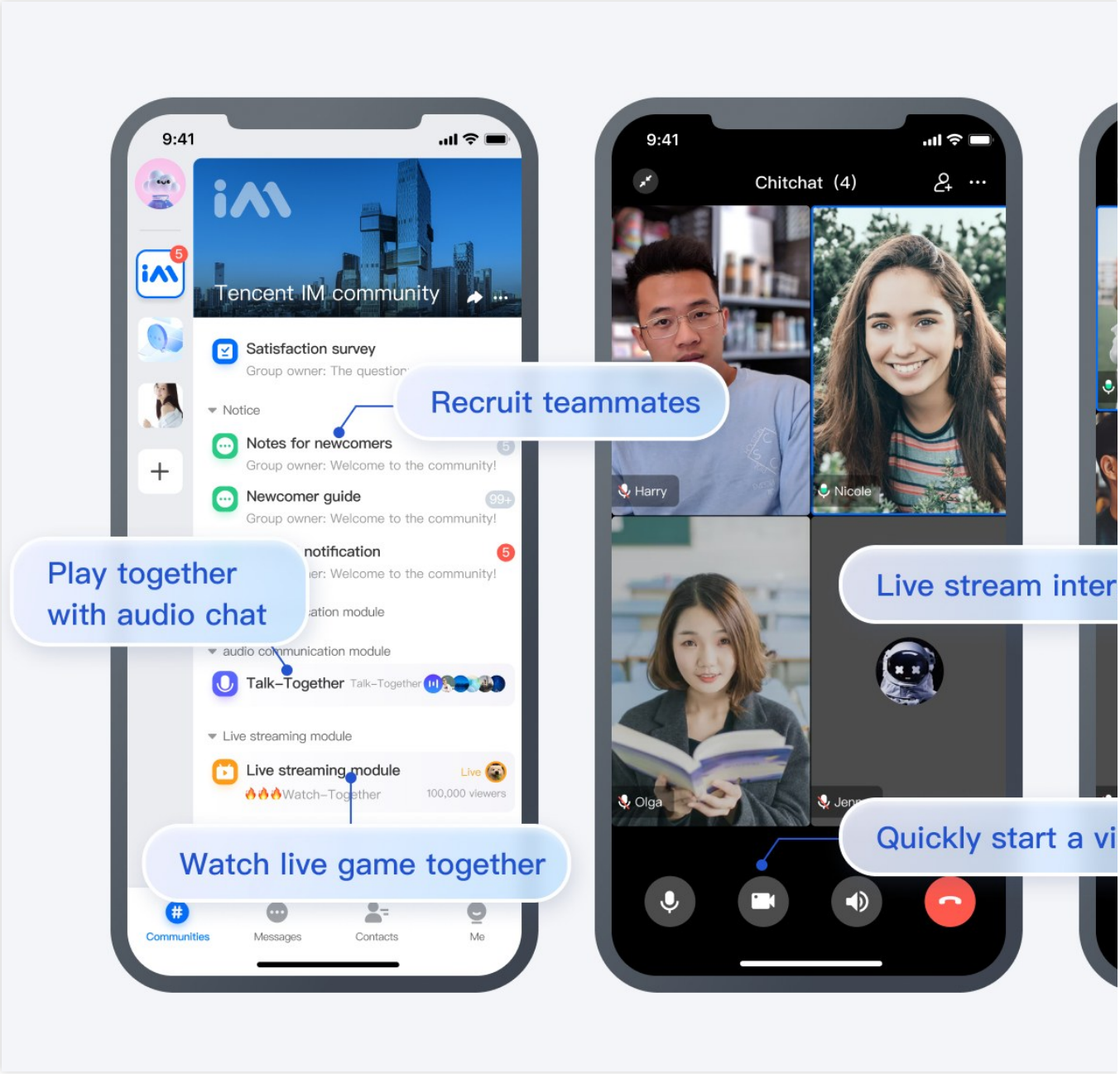
趣味によって友達を作る：ユーザーを増やすための革新的な方法

極めて大規模な同好者の集まりをサポートし、**コミュニティ-分類-トピック**で趣味サークルを垂直に細分化します。
大規模なオープンコミュニティでは、ユーザーに閉鎖的で小さなトピックが提供されます。このような快適な中間領域では、ユーザーが自由にコミュニケーションできるトピックを選択できるため、メンバーの参加意欲が高まります。



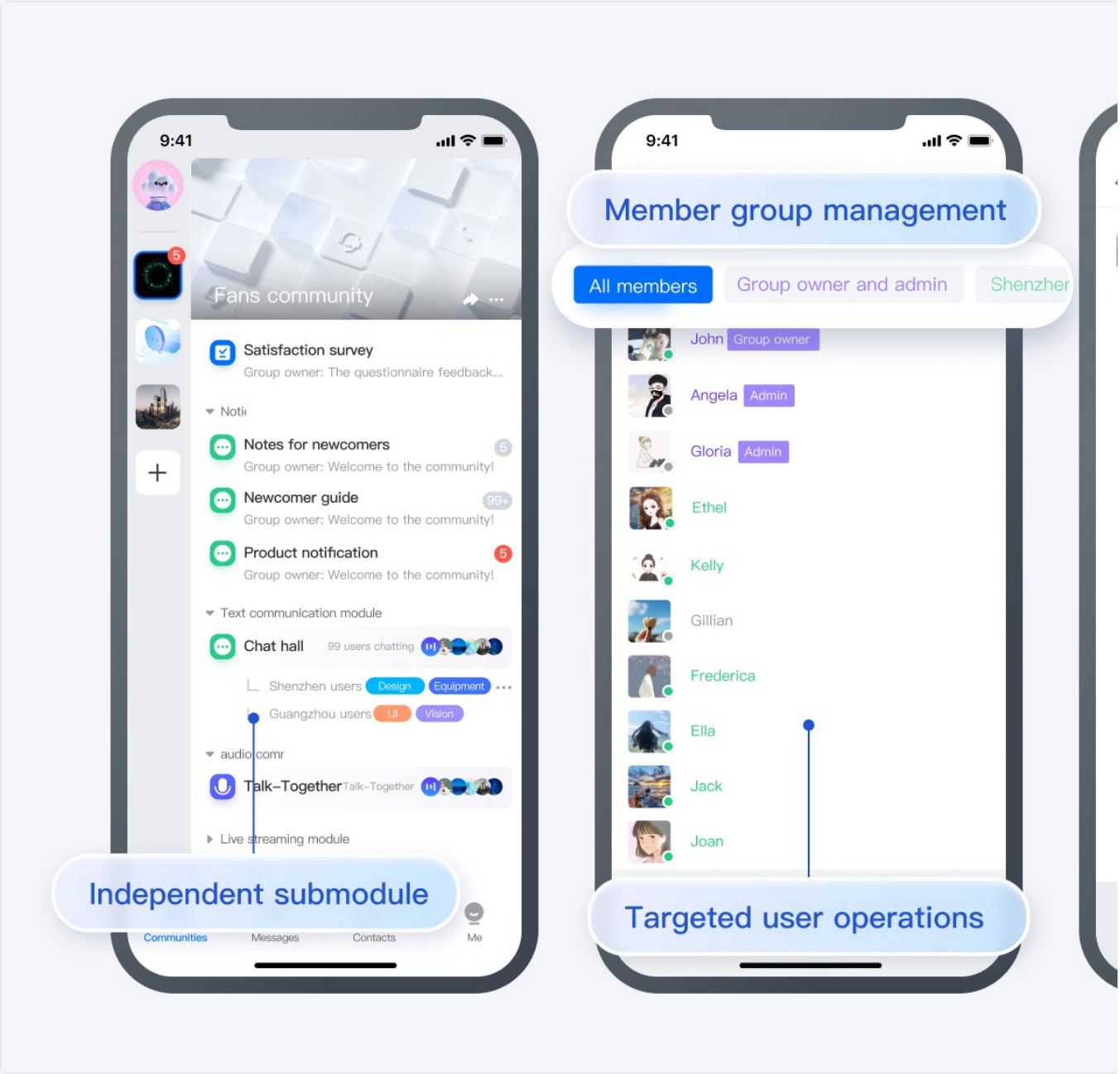
ゲームソーシャル：ユーザーの利用率とアクティブ率を向上させる

1つのコミュニティ内で複数のトピックが提供され、情報の入手、チームメイトの募集、ストーリーの議論、攻略の共有など、多くの情報に対するプレーヤーのニーズを完璧に解決できます。ゲーム前に理解し、問い合わせることができます。ゲーム中はいつでも音声で話す（チャットルームは常にオンラインです）ことができます。ゲーム終了後もトピックでコミュニケーションを続けることができます。



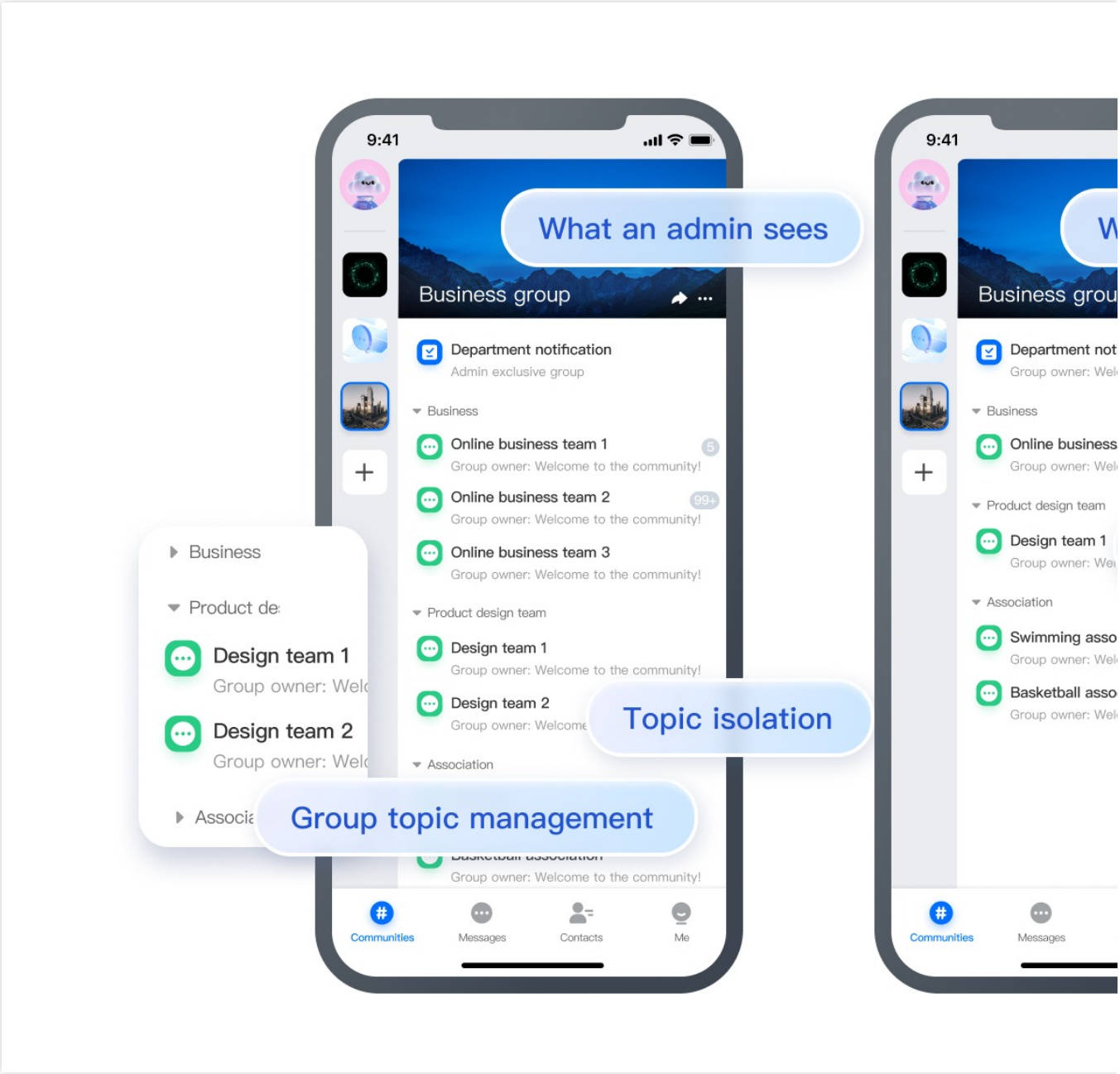
フォロワーの運営：効率的な運営ツールがある

それぞれ独立したグループの運営の代わりに（「深センユーザーグループ1」、「深センユーザーグループ2」、「広州ユーザーグループ1」、「上海ユーザーグループ1」などの複数のグループを置き換える）、1つのコミュニティで複数のトピックを提供することで、複数のグループで操作を行う必要がなく、より簡単かつ正確にフォロワーを運営できるようになります。



組織管理：明確な階層的コミュニケーションを実現

組織のすべてのメンバーを同じコミュニティに引き込み、コミュニティの階層的な機能と権限設定に依存して、階層的コミュニケーションを実現できます。



技術的優位性

極めて大規模なグループメンバー

Tencent IMコミュニティの容量は約10,000倍に拡大し、友達作り、フォロワーの運営、ゲームソーシャル、組織管理などのシナリオで多数のメンバーのニーズに対応しています。

メッセージの信頼性

Tencent IMコミュニティはメンバーの容量を大幅に拡大しましながら、Tencentの強力なメッセージ機能を継承し、20年以上の技術蓄積に基づいて完全で信頼性の高いメッセージシステムを構築しました。99.99%を超える

メッセージの送受信成功率とサービスの信頼性をお客様に提供し、お客様が億レベルの大規模な並列処理タスクに簡単に対処できるようにします。

メッセージプッシュのパフォーマンス

Tencent IMコミュニティは、「高速チャネルと低速チャネル」+「2つのレベルの複合プッシュ」という新しいメッセージプッシュアーキテクチャを採用しています。これにより、時間とスペースのバランスを効果的に取り、システムプッシュのパフォーマンスを向上させるほか、端末のパフォーマンス消費を削減して、極めて大規模なグループでも、通常のグループと同じようなメッセージインタラクション体験を提供できます。

メッセージステータスとユーザー権限の維持

Tencent IMコミュニティは、メッセージの編集、取り消し、転送などの豊富な拡張機能を提供します。発言禁止、通知のミュート、未読メッセージの数、プロフィール編集などはすべて、ユーザーによる全般、コミュニティ、トピックのレベルでのカスタマイズをサポートします。

端末SDKの統合ガイド

ご注意：
コミュニティ(Community)トピック(Topic)機能は、IM端末SDK 6.2.2363拡張バージョン以降でのみサポートされています。Ultimate Editionを購入して、有効化を申請した後にのみ使用できます。
以下では、Androidを例として、コミュニティトピックのインターフェース機能について説明します。

コミュニティトピックに関するAPI使用

1. まず、インターフェースcreateGroupを呼び出して、トピックをサポートするコミュニティを作成します。これを実現するには、コミュニティ管理の「コミュニティの作成」ステップをご参照ください。
2. 次に、getJoinedCommunityListインターフェースを呼び出すことにより、作成・参加したこのタイプのコミュニティリストを取得できます。

ご注意：
コミュニティはグループメンバーを管理するために使用されます。ただし、コミュニティでメッセージを送受信することはできません。コミュニティのその他の機能については、「その他の管理インターフェース」リストをご参照ください。

3. コミュニティが正常に作成されたら、コミュニティの下でcreateTopicInfoCommunityを呼び出して、複数のトピックを作成できます。これを実現するには、トピック管理の「トピックの作成」のステップをご参照ください。
4. トピック管理には、「トピックの削除」、「トピック情報の変更」、「トピックリストの取得」、および「トピックコールバックの監視」の機能も含まれます。同時に、トピックはユーザーがコミュニケーションをとるための場所でもあるため、メッセージを送受信できます。関連するインターフェースについては、トピックメッセージの説明をご参照ください。

5. グループメンバーのカスタムフィールドにサブグループ情報を設定することで、コミュニティメンバーのサブループを行い、サブグループ表示の効果を得ることができます。これは、すべてのグループメンバーをローカルにプルし、グループ別に並べ替える必要があります。グループ内の人数が多い場合は、サーバー側で実装することをお勧めします。

Web SDKの統合ガイド

ご注意：
コミュニティ(Community)トピック(Topic)機能は、IM Web SDK v2.19.1バージョン以降でのみサポートされています。Ultimate Editionを購入して、**コンソール>グループ機能設定>コミュニティ**でスイッチをオンにした後にのみ使用できます。
コミュニティトピックのインターフェース機能は次のとおりです：

Demoのソースコードのダウンロードと設定によるクイック体験

クイックスタートを参照すると、IMの機能をすばやく体験できます。
以下では、API呼び出しの観点からコミュニティトピックの使用について説明します：

コミュニティトピックに関するAPI使用

1. まず、インターフェースcreateGroupを呼び出して、トピックをサポートするコミュニティを作成します。これを実現するには、コミュニティ管理の「コミュニティの作成」ステップをご参照ください。
2. 次に、getJoinedCommunityListインターフェースを呼び出すことにより、作成・参加したこのタイプのコミュニティリストを取得できます。コミュニティはグループメンバーを管理するために使用されます。ただし、トピックをサポートするコミュニティでメッセージを送受信することはできません。コミュニティのその他の機能については、通常のグループAPIをご参照ください。
3. コミュニティが正常に作成されたら、コミュニティの下でcreateTopicInfoCommunityを呼び出して、複数のトピックを作成できます。これを実現するには、トピック管理の「トピックの作成」のステップをご参照ください。
4. トピック管理には、「トピックの削除」、「トピック情報の変更」、「トピックリストの取得」、および「トピックコールバックの監視」の機能も含まれます。同時に、トピックはユーザーがコミュニケーションをとるための場所でもあるため、メッセージを送受信できます。関連するインターフェースについては、メッセージの作成の説明をご参照ください。
5. グループメンバーのカスタムフィールドにサブグループ情報を設定することで、コミュニティメンバーのサブループを行い、サブグループ表示の効果を得ることができます。これは、すべてのグループメンバーをローカルにプルし、グループ別に並べ替える必要があります。グループ内の人数が多い場合は、サーバー側で実装することをお勧めします。

関連ドキュメント

グループ管理
グループシステム
SDKダウンロード
SDKマニュアル
SDKの統合(Android)
SDKの統合(iOS)
[SDKの統合（Web、ミニプログラムとuni-app）(https://intl.cloud.tencent.com/document/product/1047/34309)

お問い合わせ

ご不明の点があれば、いつでもお気軽にお問い合わせください。

How to integrate Tencent IM with Salesforce

最終更新日：：2024-02-07 17:30:51

Introduction

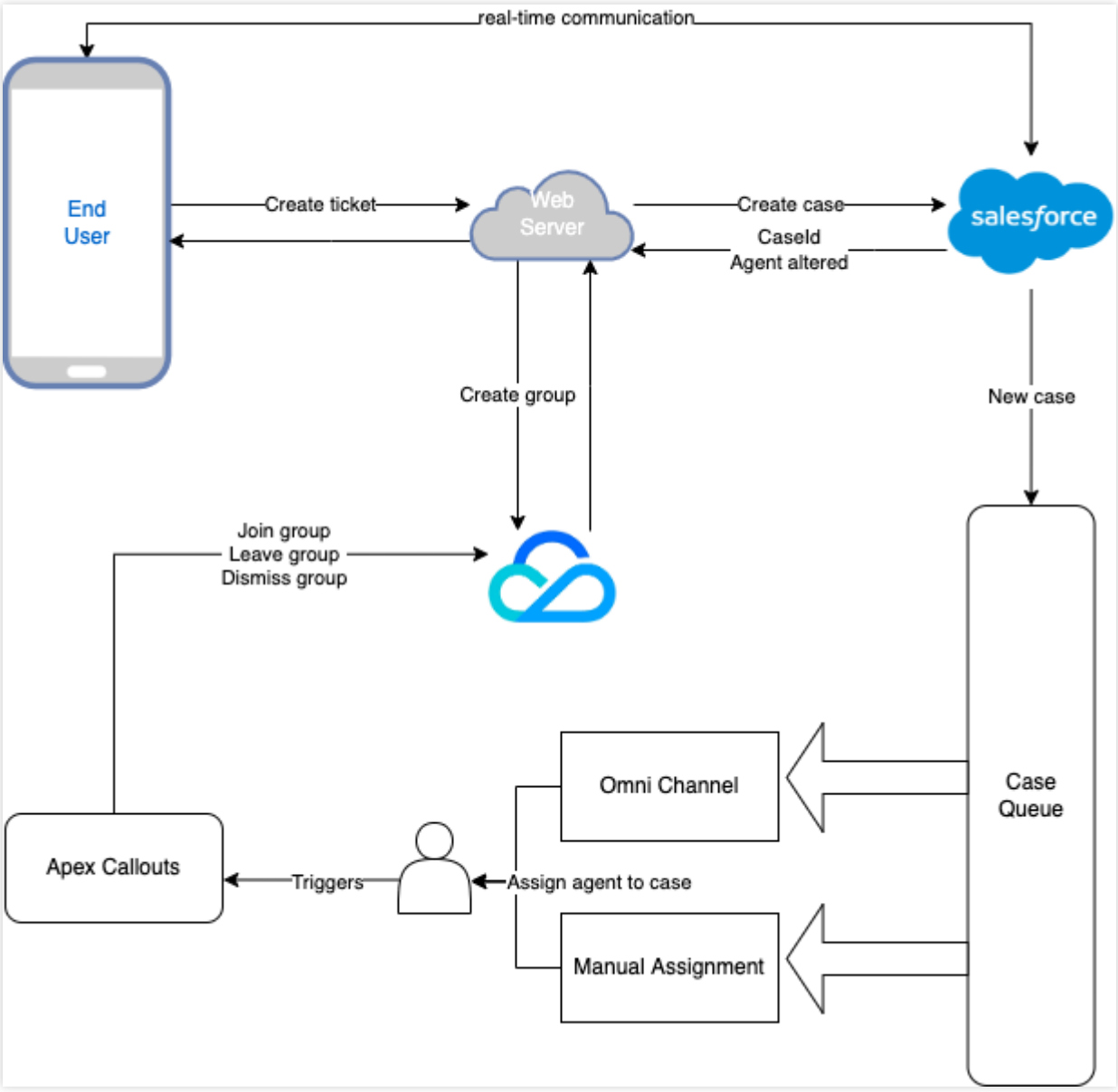
This tutorial aims to demonstrate an approach to integrate the Tencent Cloud IM SDK into Salesforce's workflow to leverage the communication between a Salesforce agent and an end user.

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

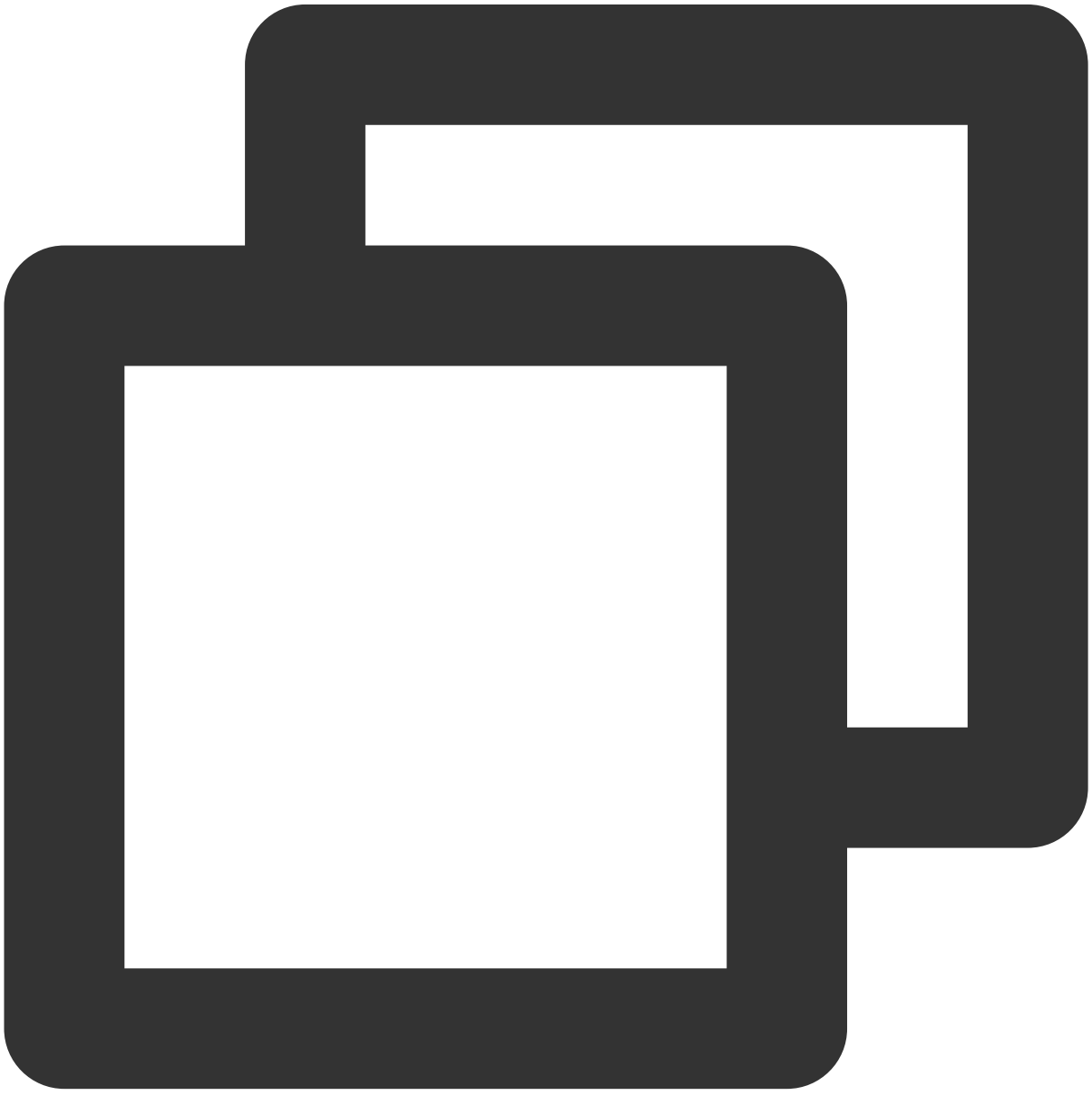
In here, we will guide you step by step to create a web server to allow an end user to submit a Salesforce case and create an Tencent Cloud IM chat group, also allow Salesforce agents to be invited/removed from Tencent Cloud IM chat group and also delete the group when case is closed.

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 }
  }
}

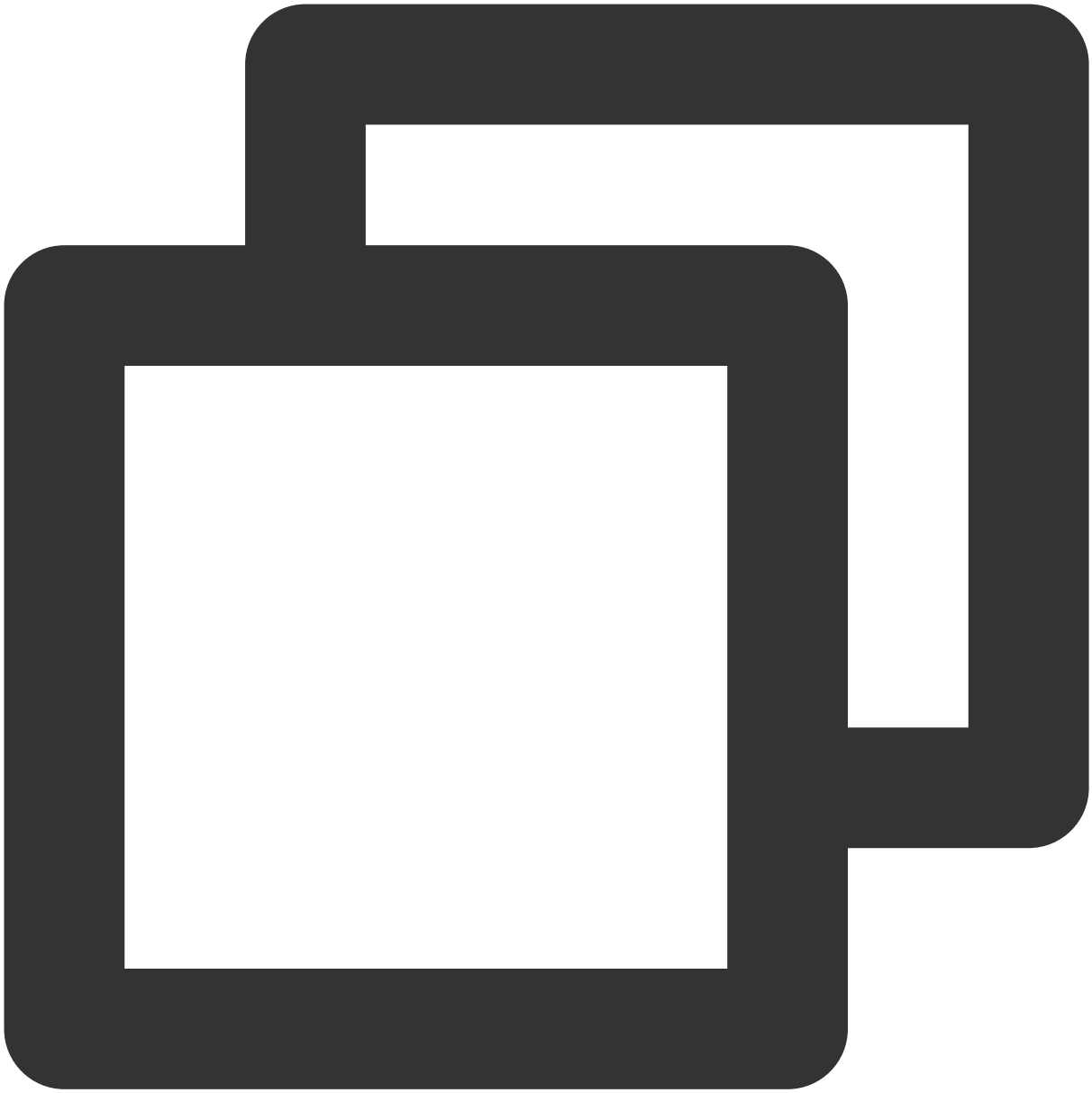
const leaveGroup = async function (groupId, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId, MemberToDel_Account: [userId] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/delete_group_membe
  try {
    const groupRes = await 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}!`)
)
```

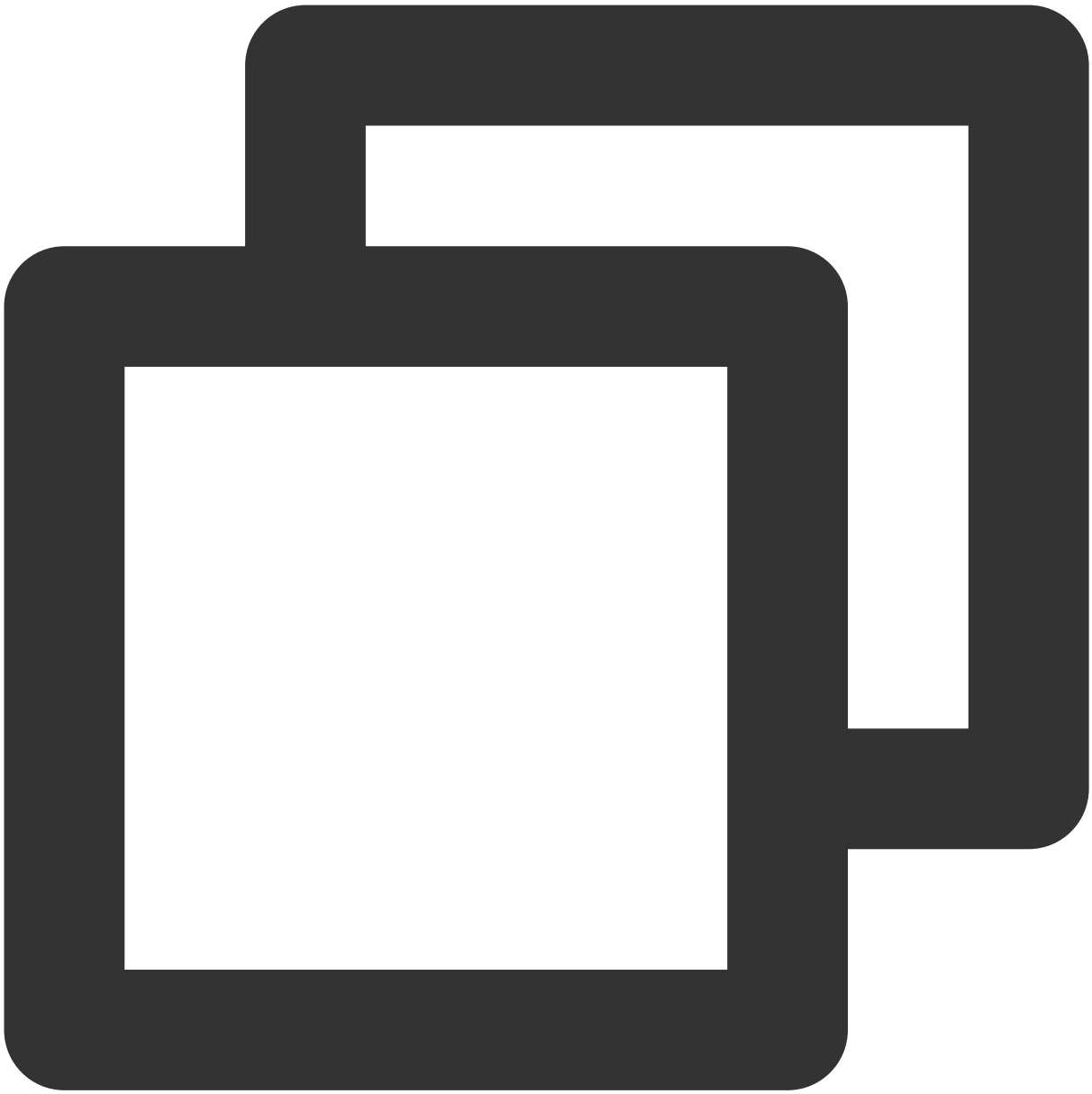
The server supports a route /createticket for the end user to create a case in Salesforce and a chat group in Tencent Cloud IM. Here's what we do here:

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 }
  }
}
```

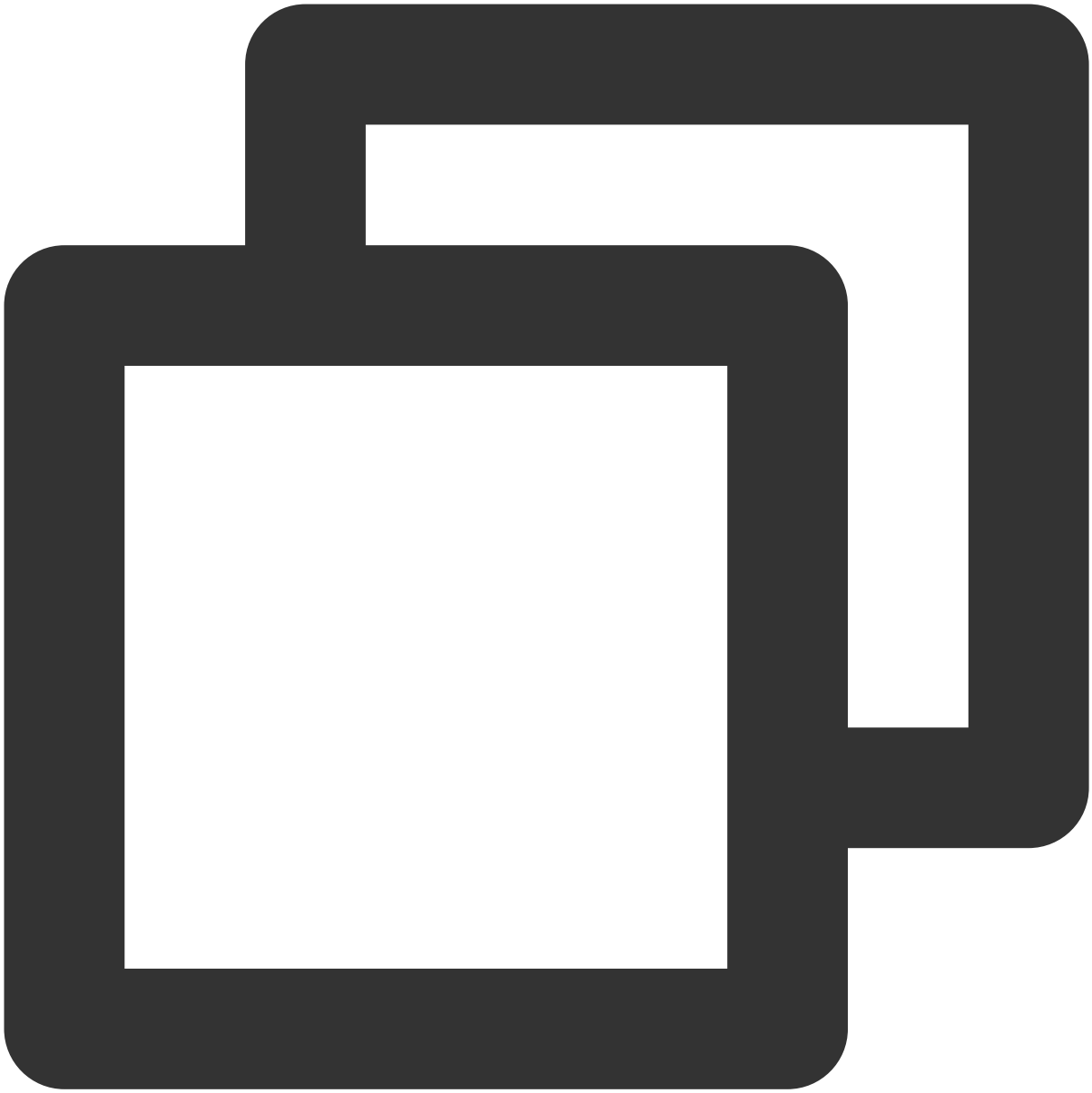
2. Create a Salesforce case.



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

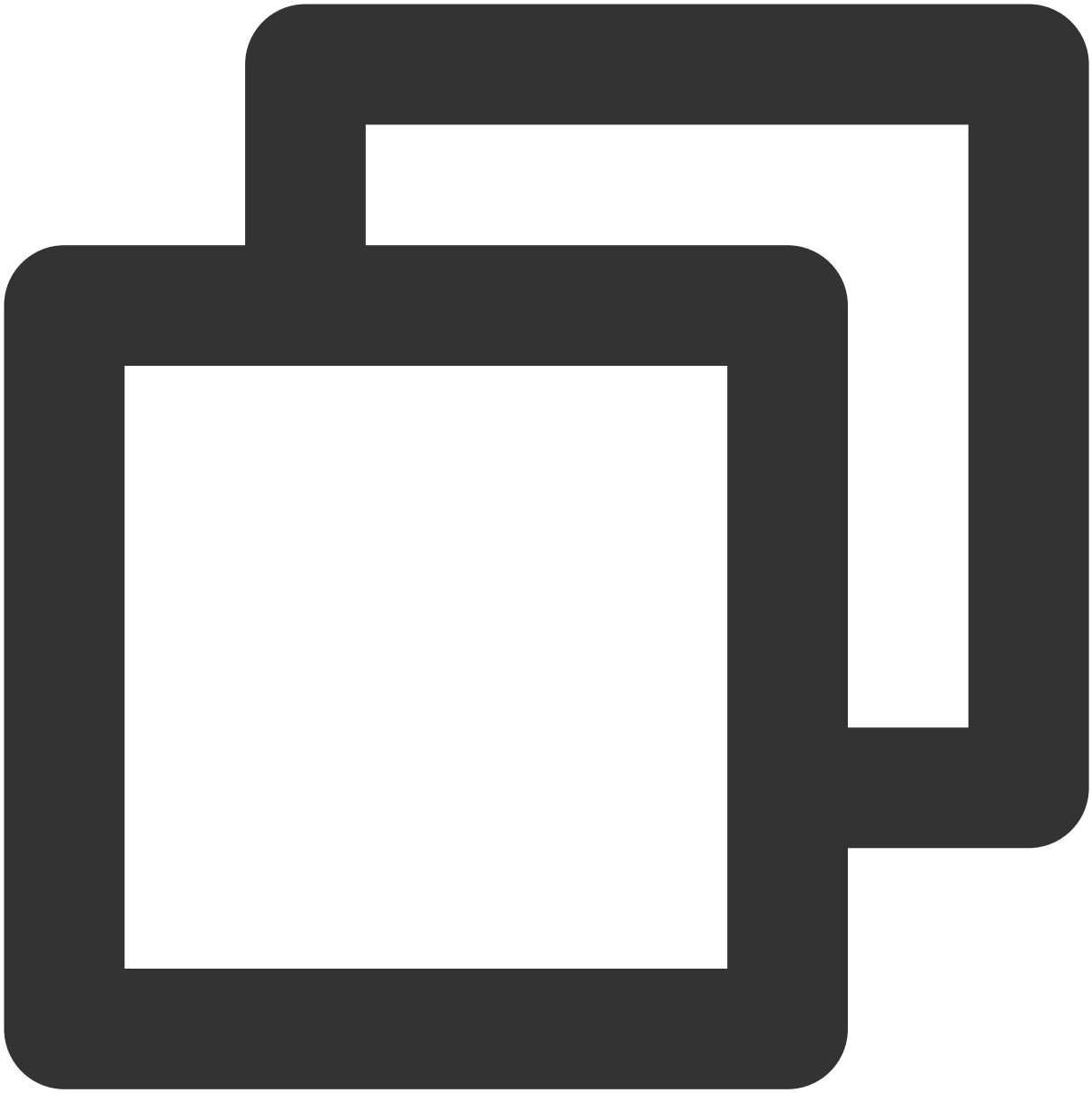
```
  headers: {
    "Content-Type": "application/json",
    Authorization: "Bearer " + token,
  },
   }
   const url =
  "https://{your_instance}.salesforce.com/services/data/v{api_version}/sobjects/Case
   try {
  const result = await axios.post(url, body, headers)
  return result.data
   } catch (e)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

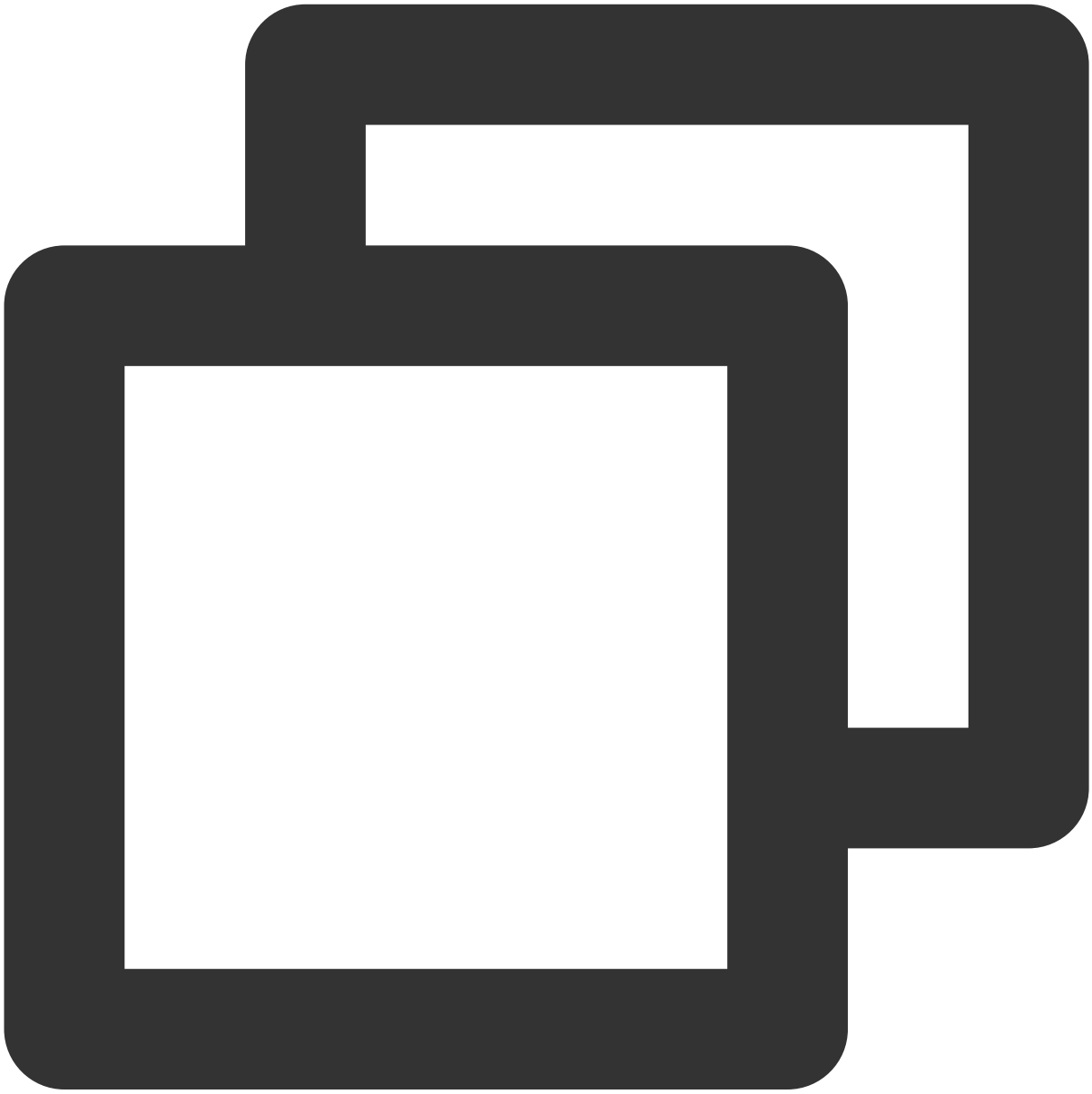
4. Create a Tencent Cloud IM chat group by case ID and user ID



```
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.

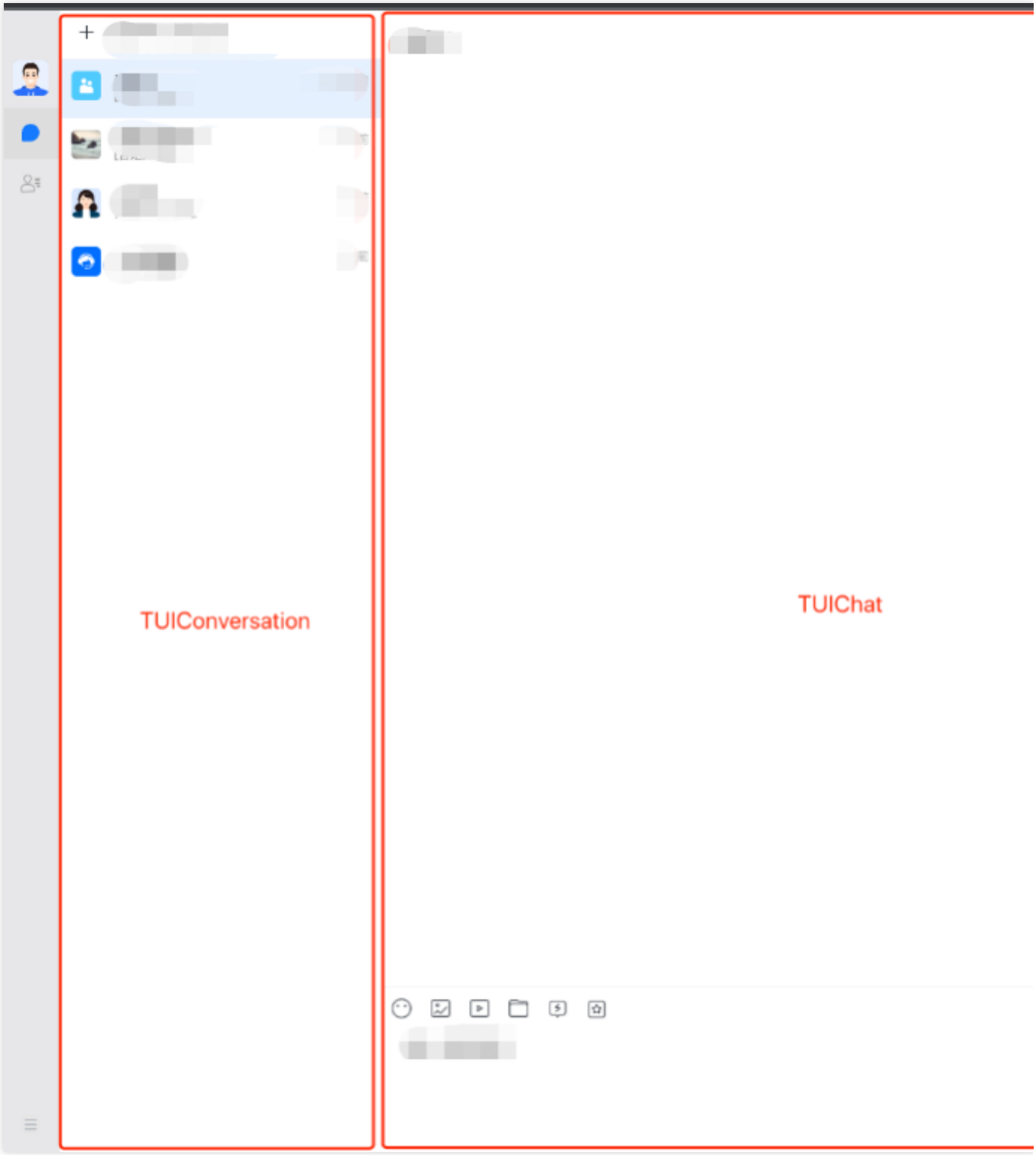
Check in the Tencent Cloud IM console for the group with the case ID as the group ID.

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.

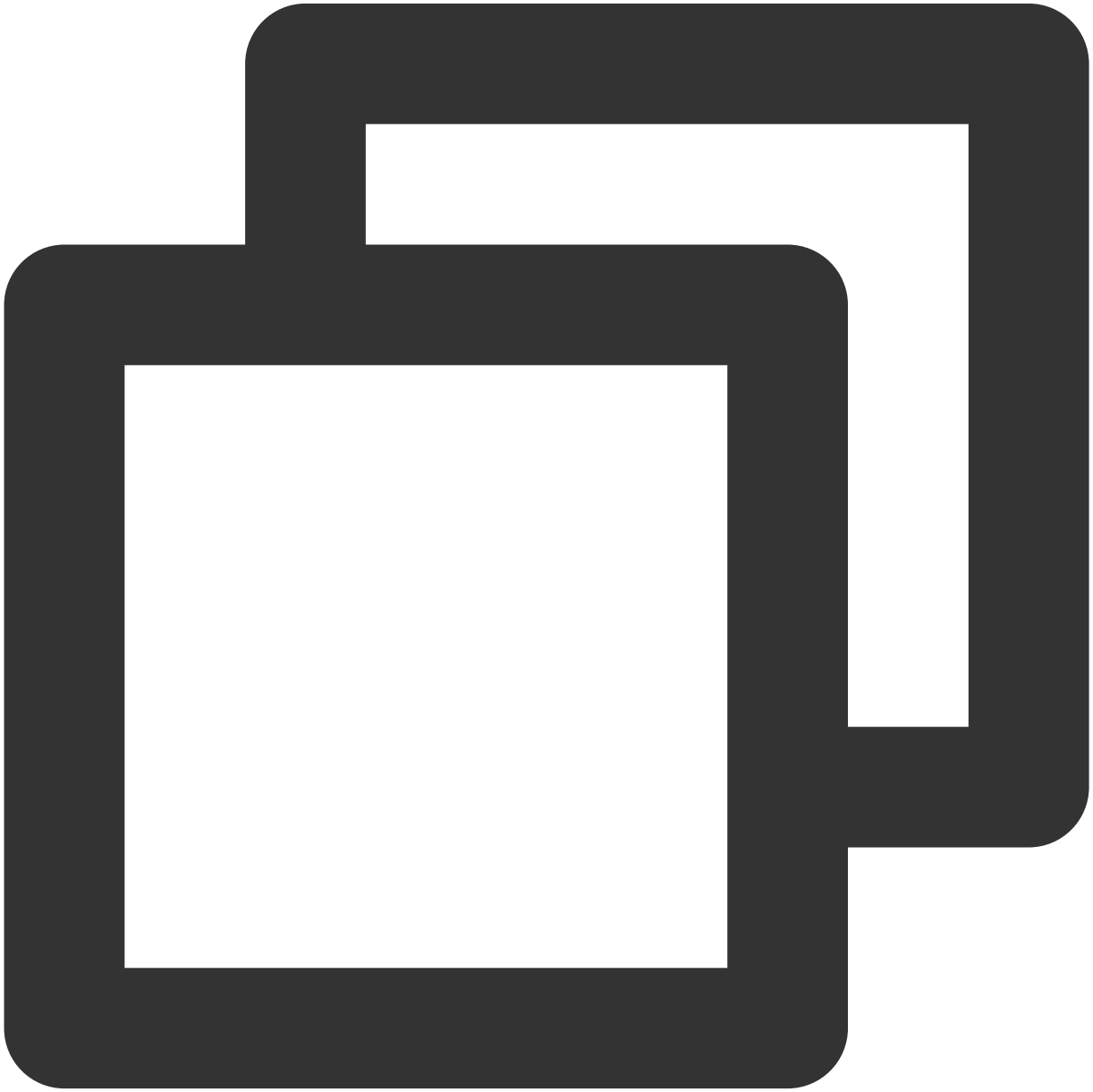
2.1 Go to Salesforce Developer Console

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.

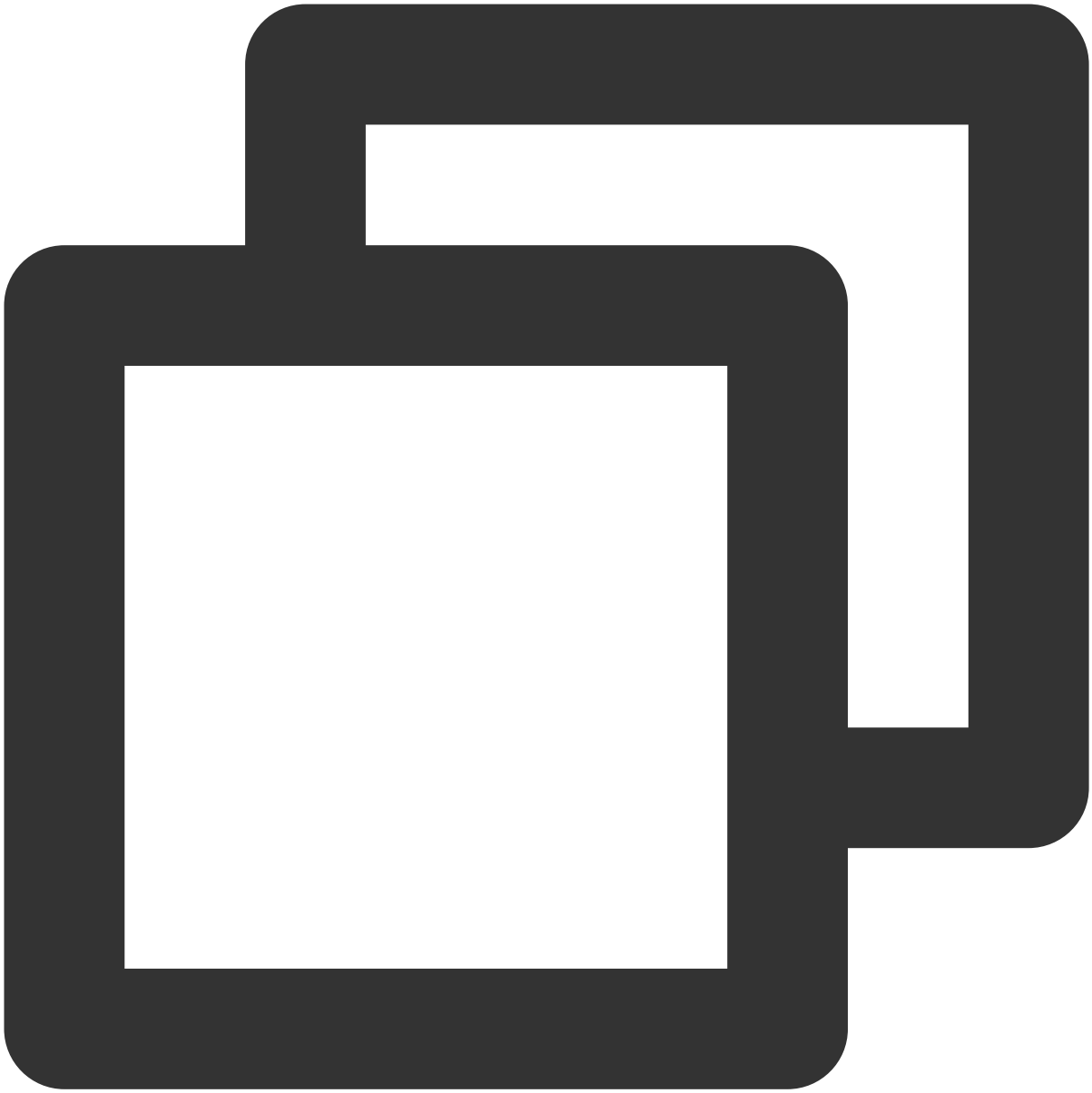
Salesforce saves the resource name, not the name of the .zip file.

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!

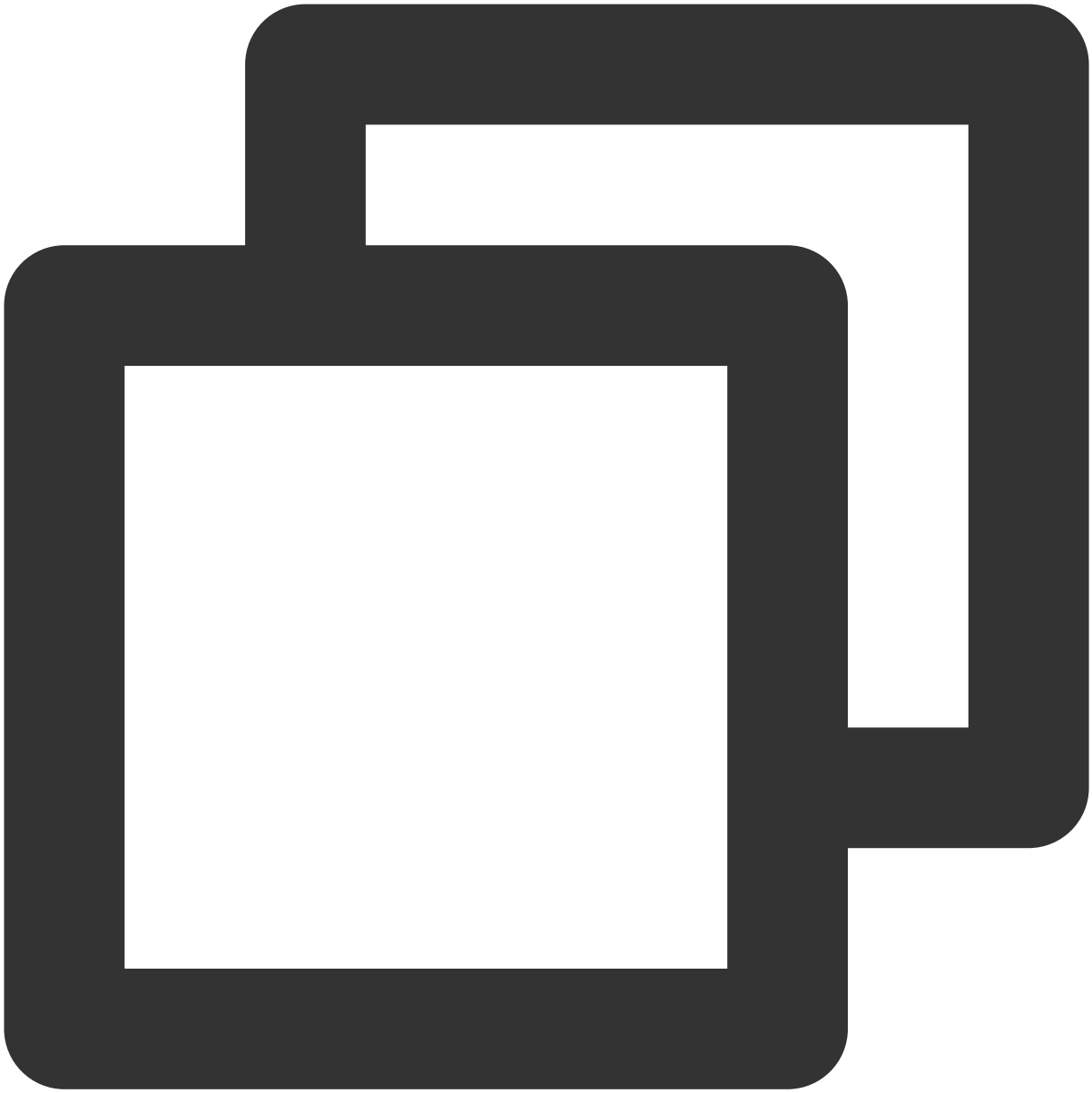
4. Initialize the Lightning Container

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

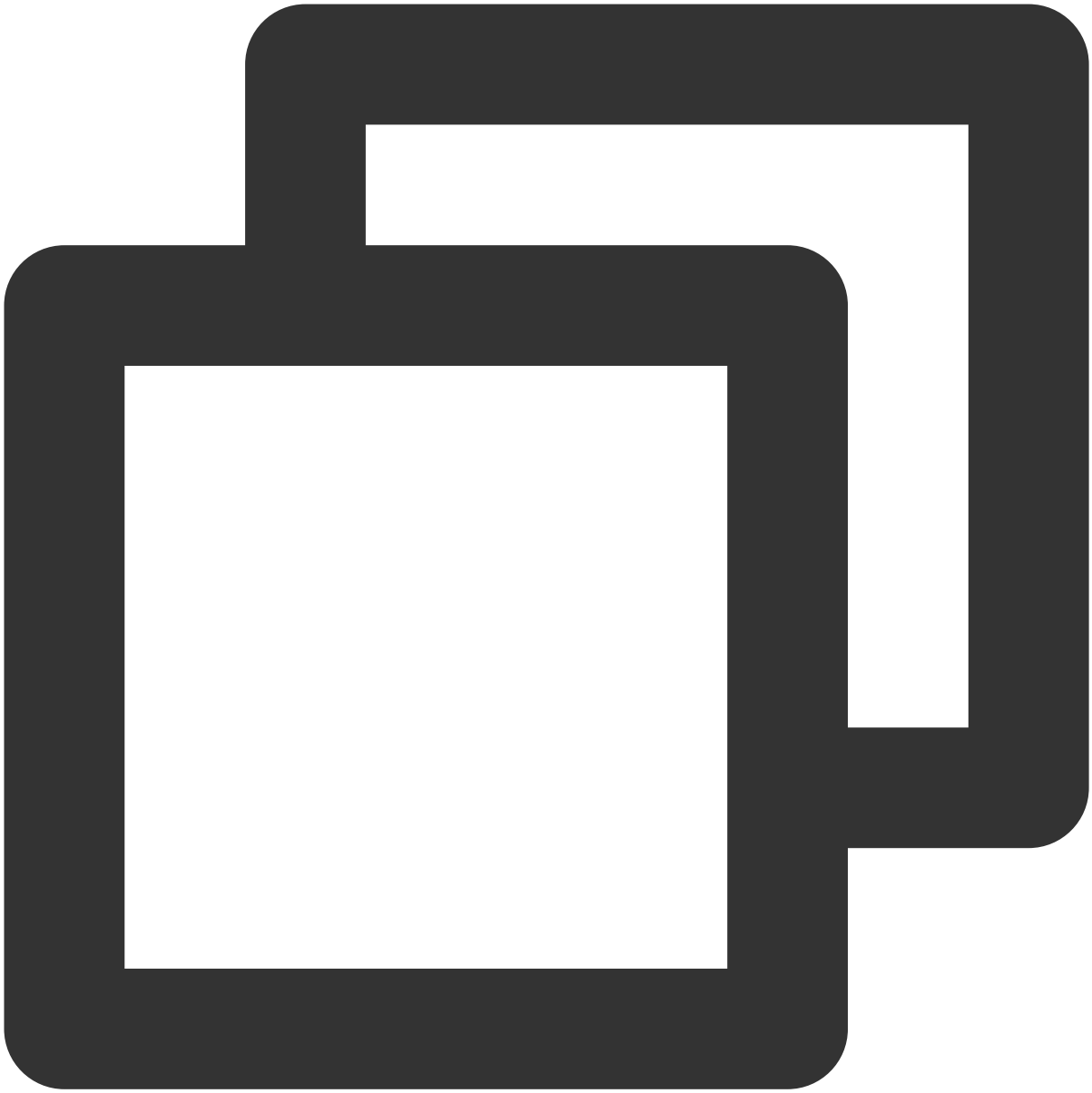
Follow the code shown below:



```
// 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
```

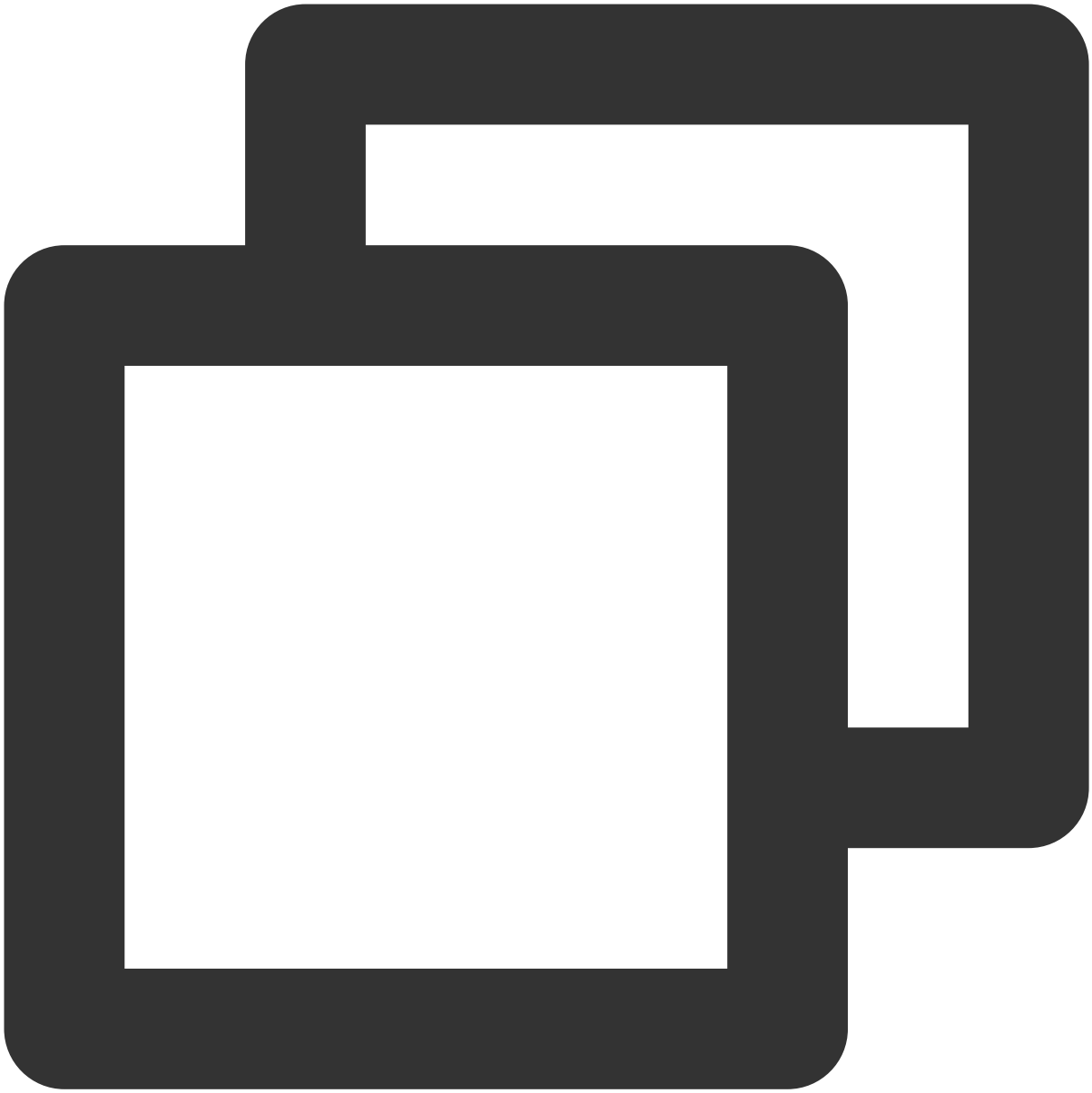
```
var userId = $A.get("$SObjectType.CurrentUser.Id")
var message = { userId: userId }
try {
// Send the ID to the component
component.find("TIM_Bar").message(message)
} catch (err) {
console.error("Error from Utilities Bar:", err)
}
},
})
```

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);
}
```

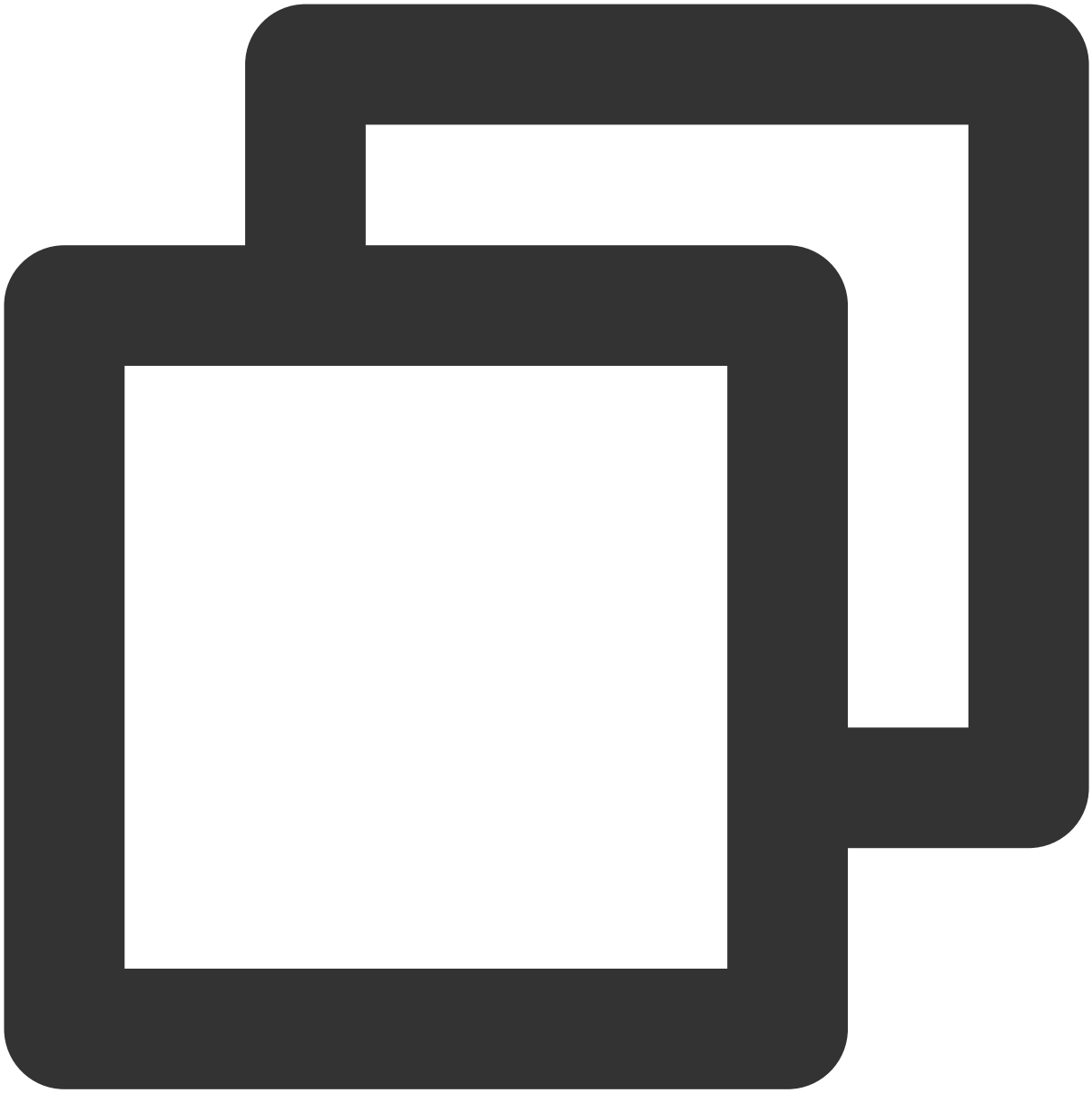
Step 3. Listen for Salesforce Case assignment and set IM Chat Group members

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"



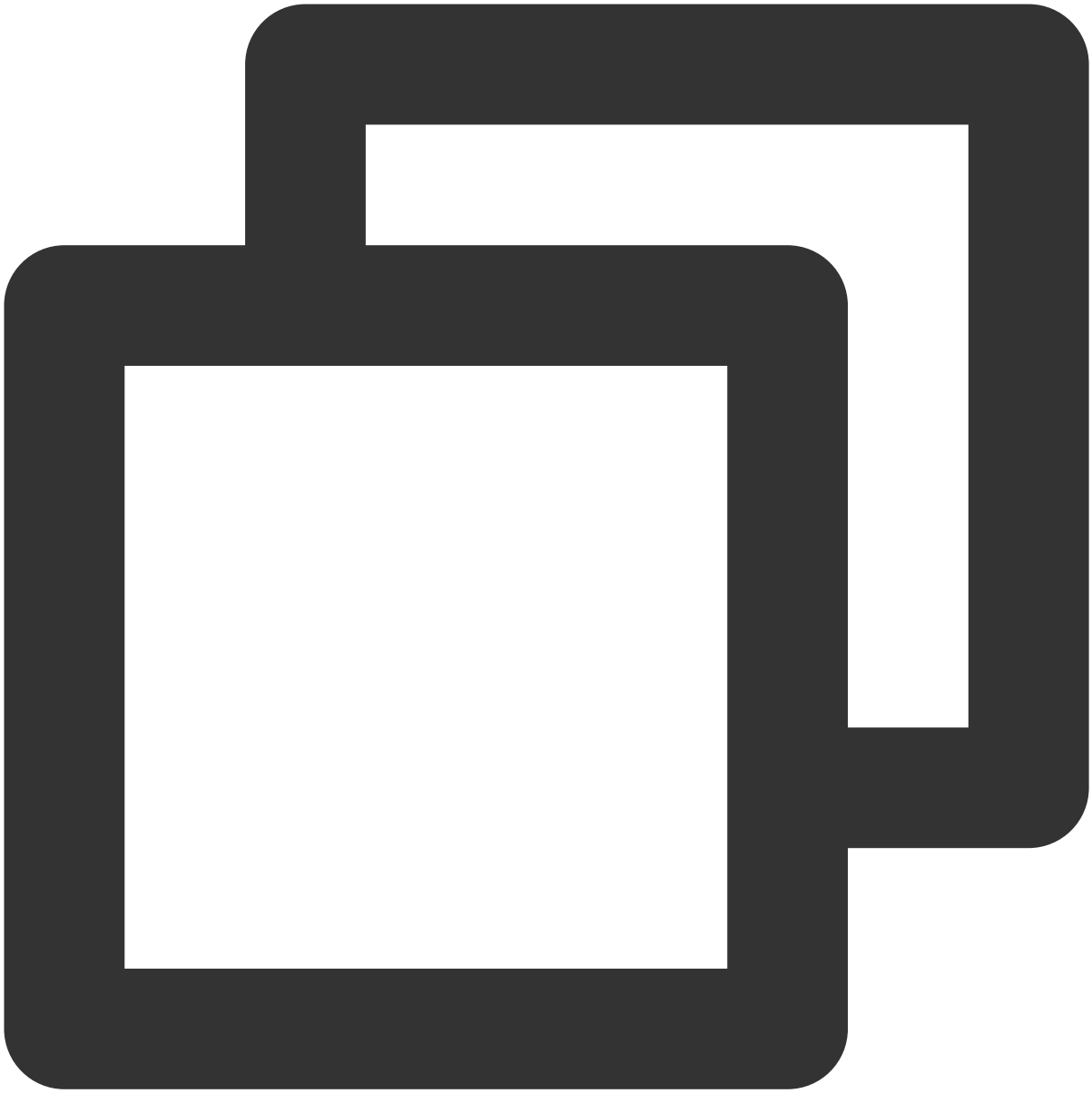
```
// AssignAgent.apxt
trigger AssignAgent on Case (after update, after delete) {
     if(trigger.isUpdate){
       // Case is updating
       System.debug('Case Update Fired:');
       for(Case a : trigger.new){
         Case oldCase = trigger.oldMap.get(a.ID);
         if(String.valueOf(a.OwnerId).substring(0, 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"

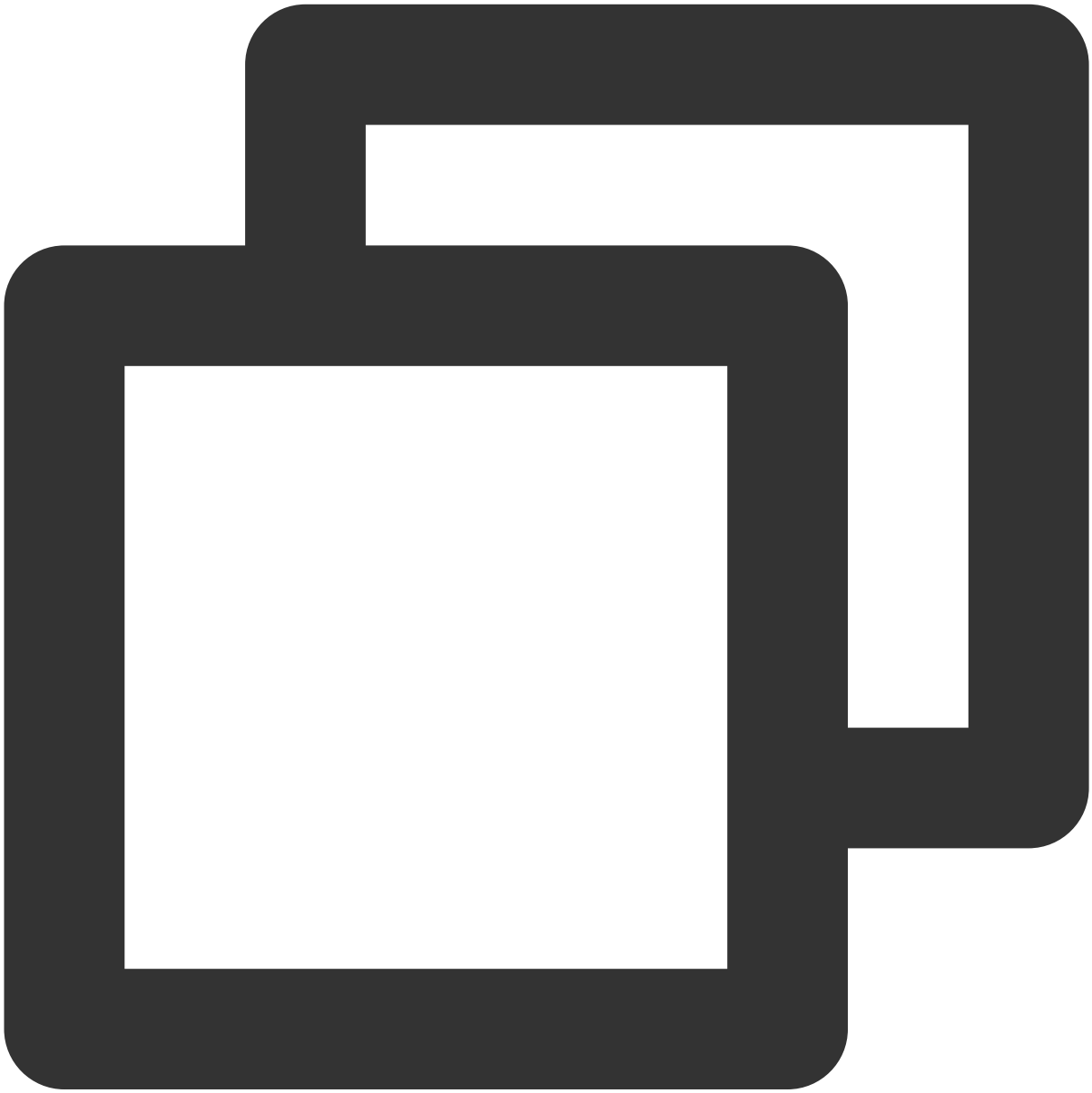


```
// AgentOmniChanne.apxt
trigger AgentOmniChannel on AgentWork (after update, after insert) {
    if(Trigger.isUpdate){
      for(AgentWork a : Trigger.new){
            AgentWork oldCase = Trigger.oldMap.get(a.ID);
            if(a.Status == 'Opened' && String.valueOf(a.OwnerId).substring(0, 3) ==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

最終更新日：：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.

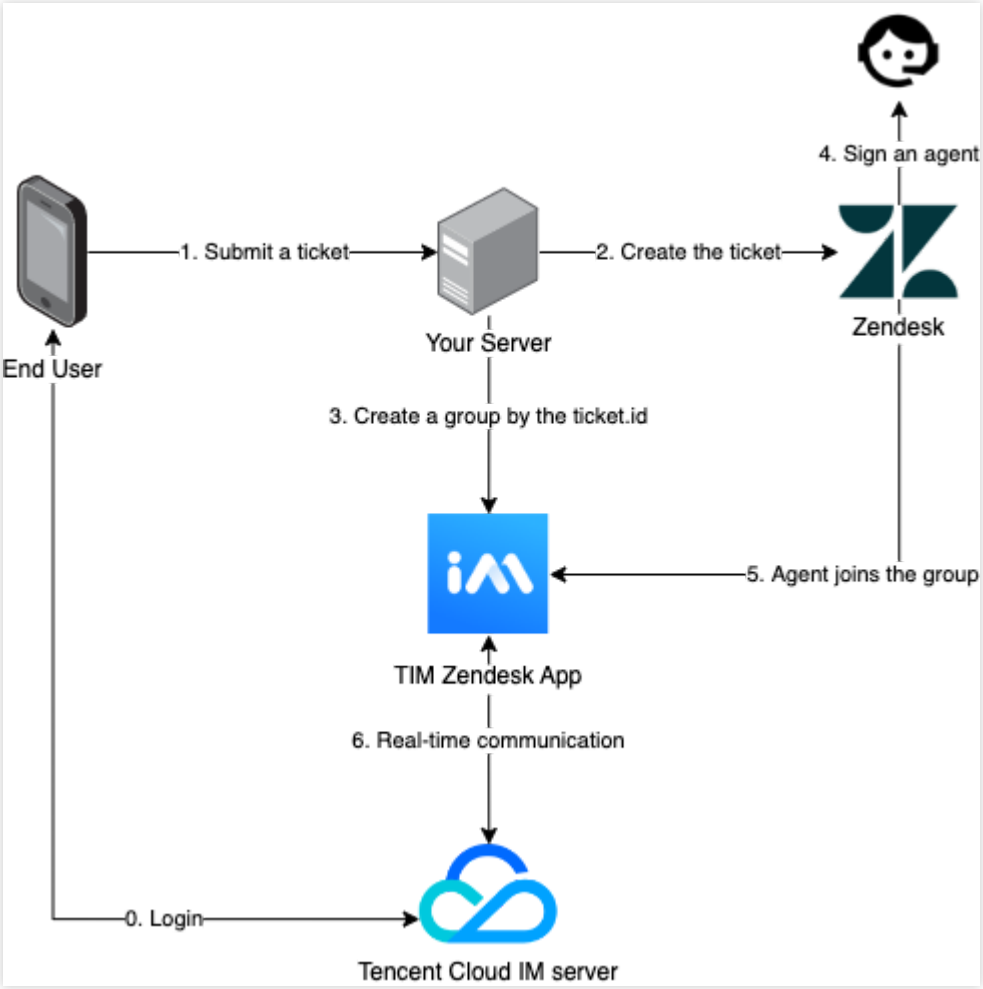
Prerequisites

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.

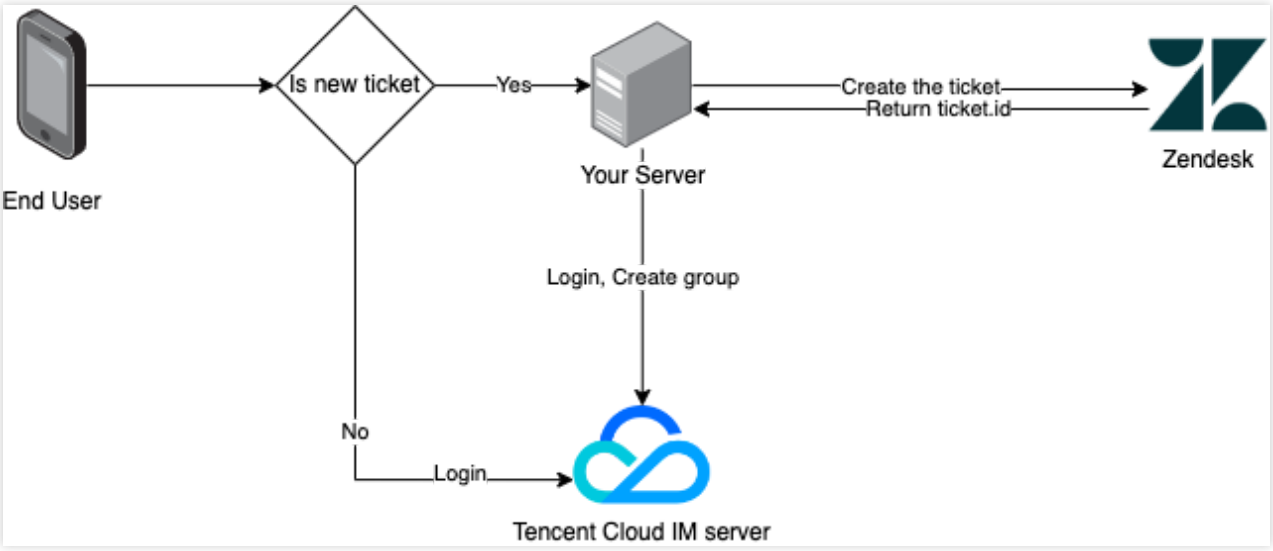
3. Create a group in Tencent Cloud IM, waiting for agent to join the group.

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:

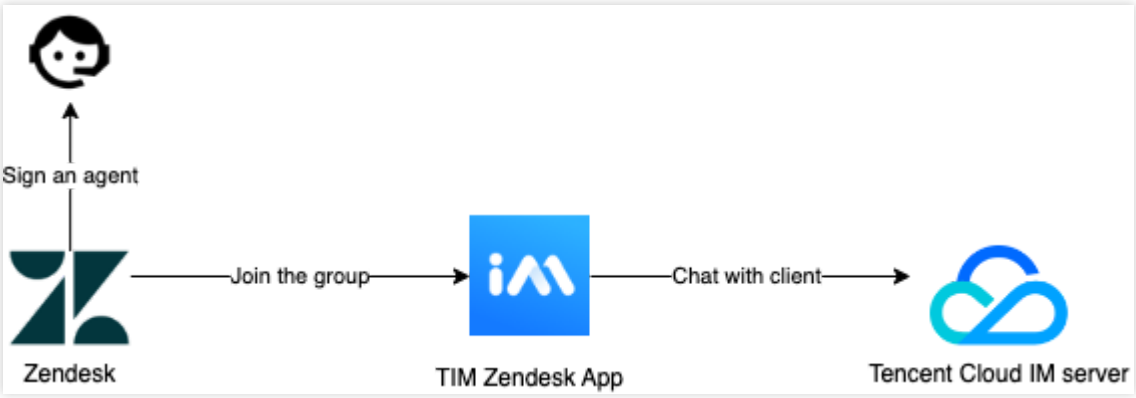


Before logged in, client may choose to reuse the ticket created last time. And for that, just login to Tencent Cloud IM server.

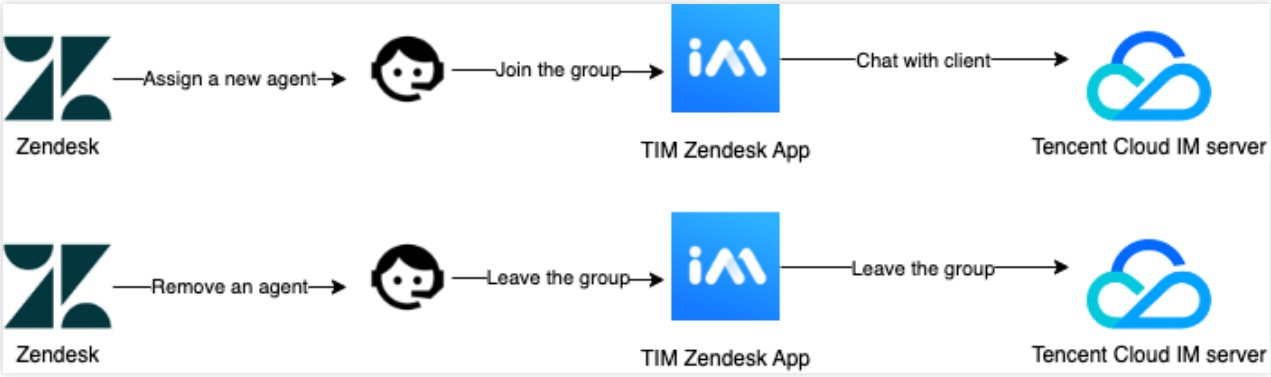
Otherwise submit a new ticket to Zendesk.

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

The process of Tencent Cloud IM for Zendesk is:



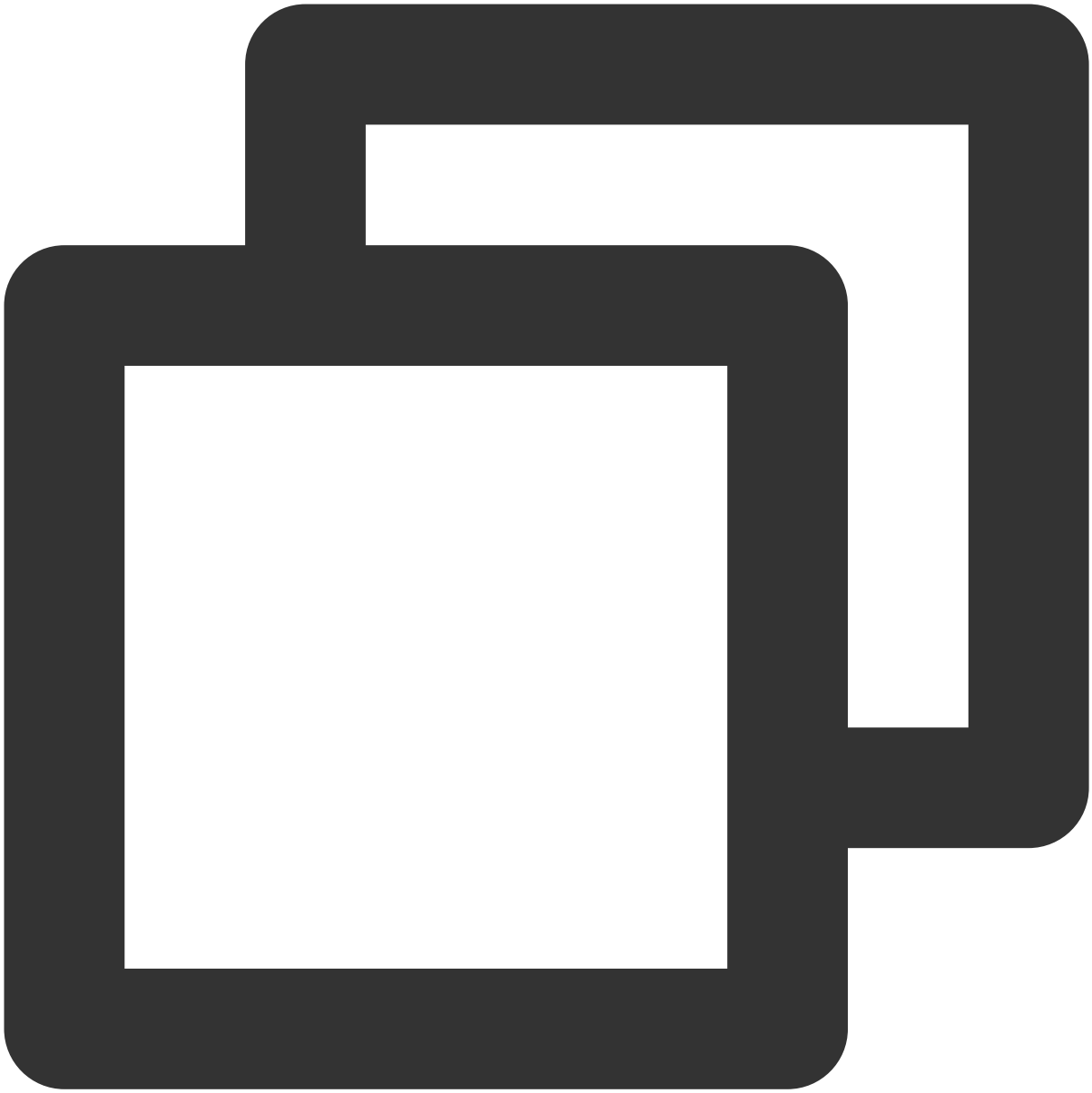
The process of updating the ticket:



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);
};

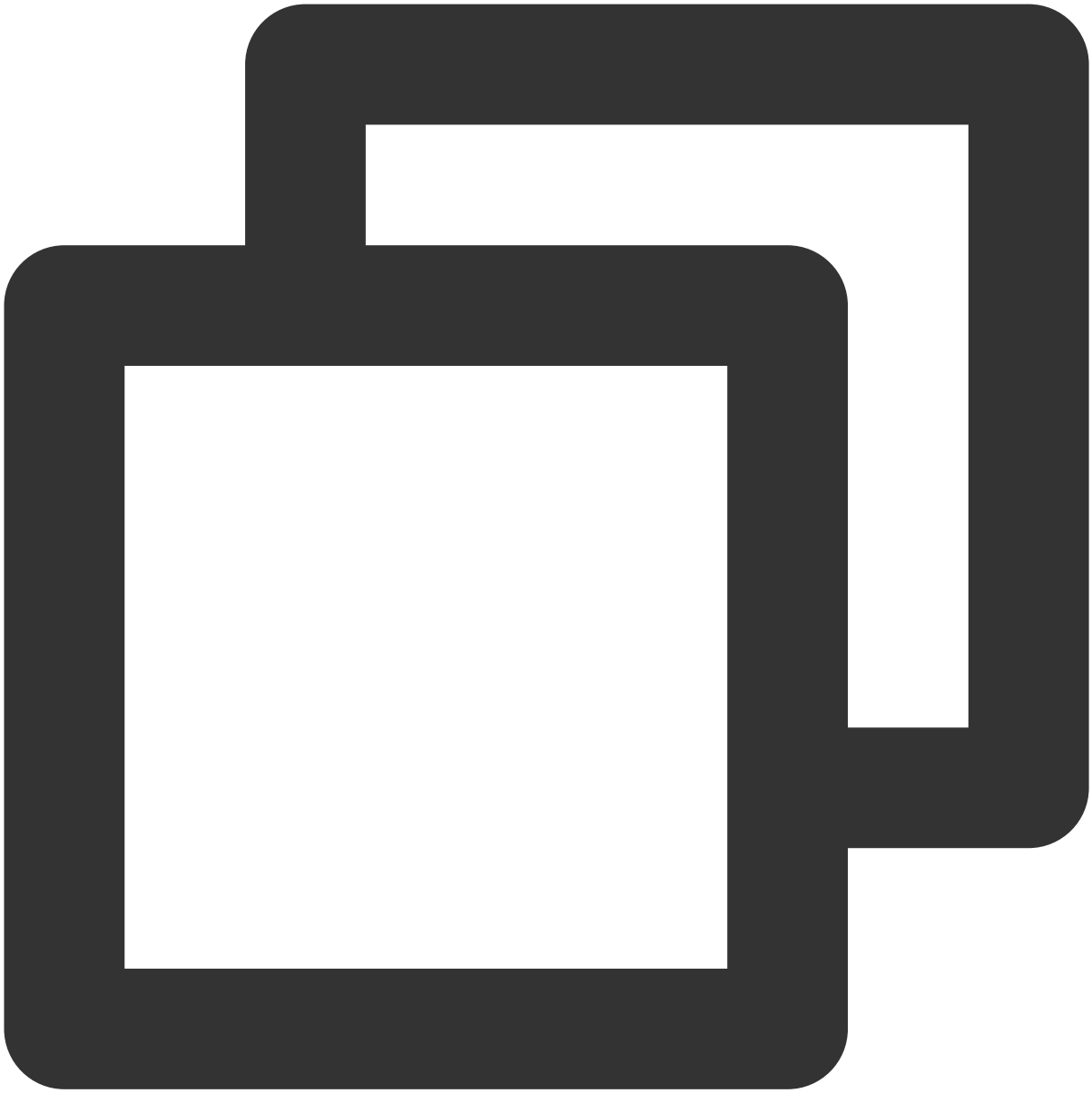
const createGroup = async function (groupId, userId, groupName) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    Type: 'Public',
    Name: groupName,
    GroupId: `ZENDESK#${groupId}`,
    ApplyJoinOption: 'FreeAccess',
    MemberList: [{ Member_Account: userId }]
  };
  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}!`)
);
```

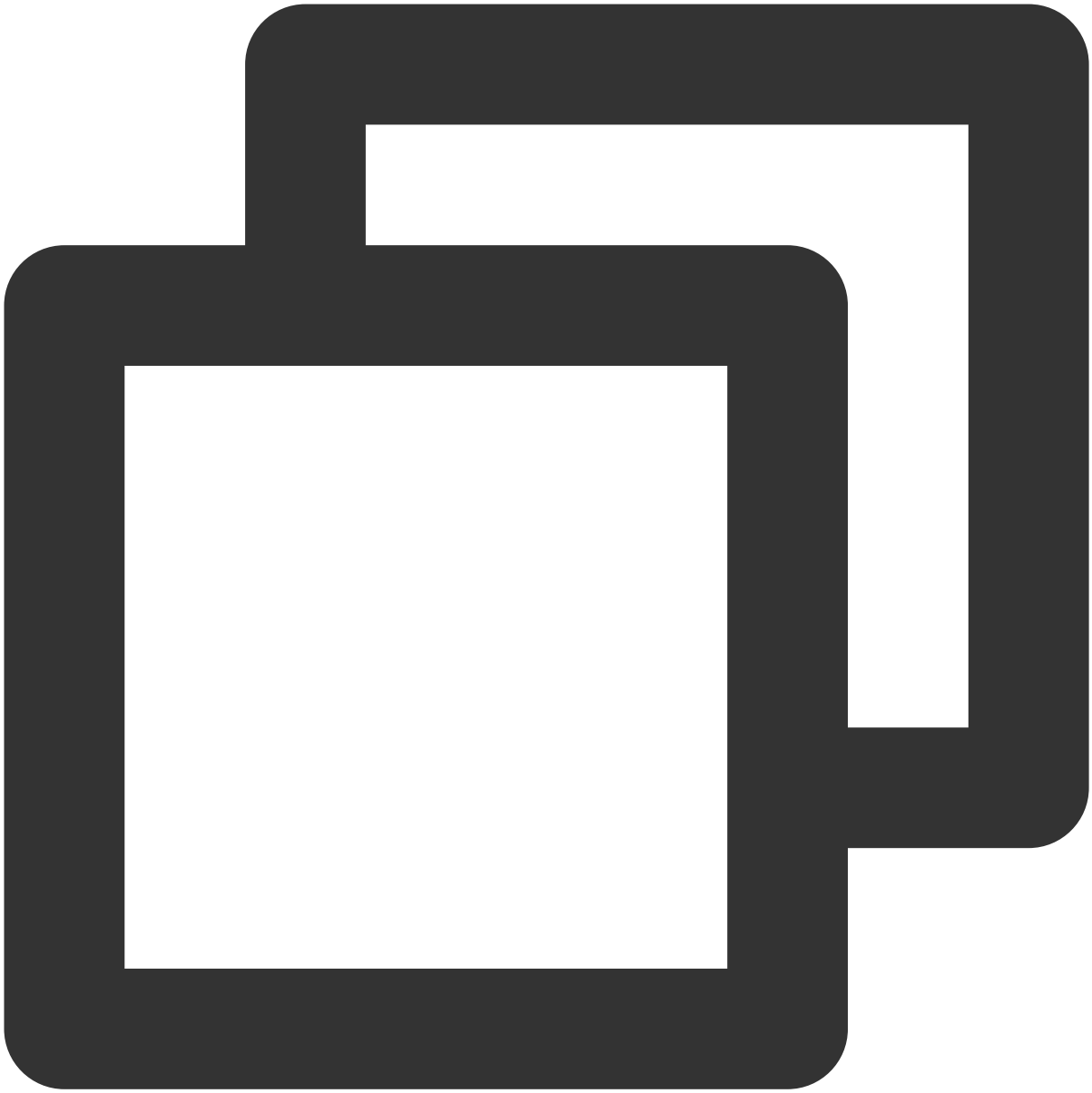
Let's break down the code.



```
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 || '';
```

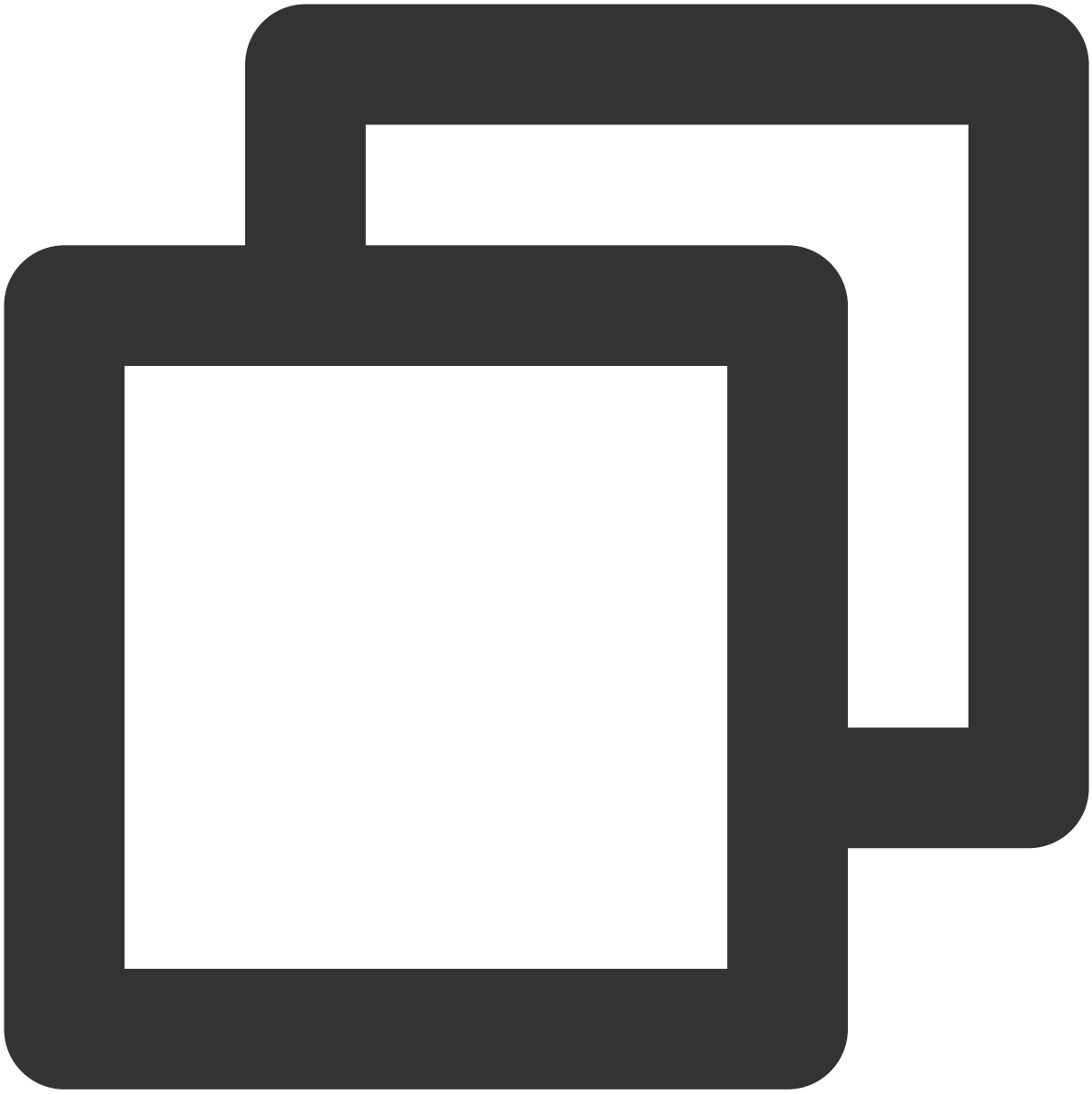
The first part requires you to set your Tencent Cloud IM identification information as the environment variables. You can find all the information here.

`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 };
  }
};

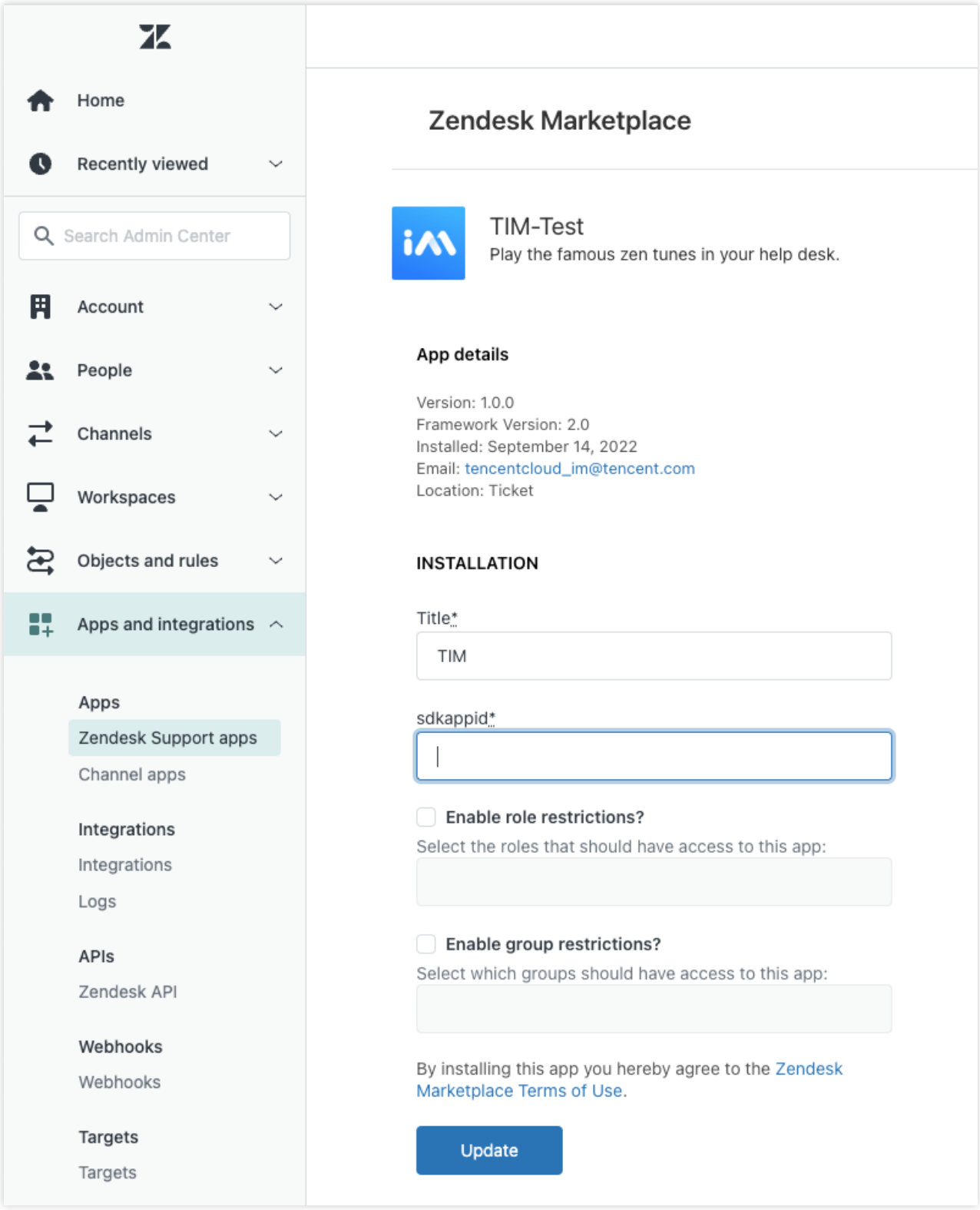
// Create IM group
const createGroup = async function (groupId, userId, groupName) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    Type: 'Public',
    Name: groupName,
    GroupId: `ZENDESK#${groupId}`,
    ApplyJoinOption: 'FreeAccess',
    MemberList: [{ Member_Account: 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.

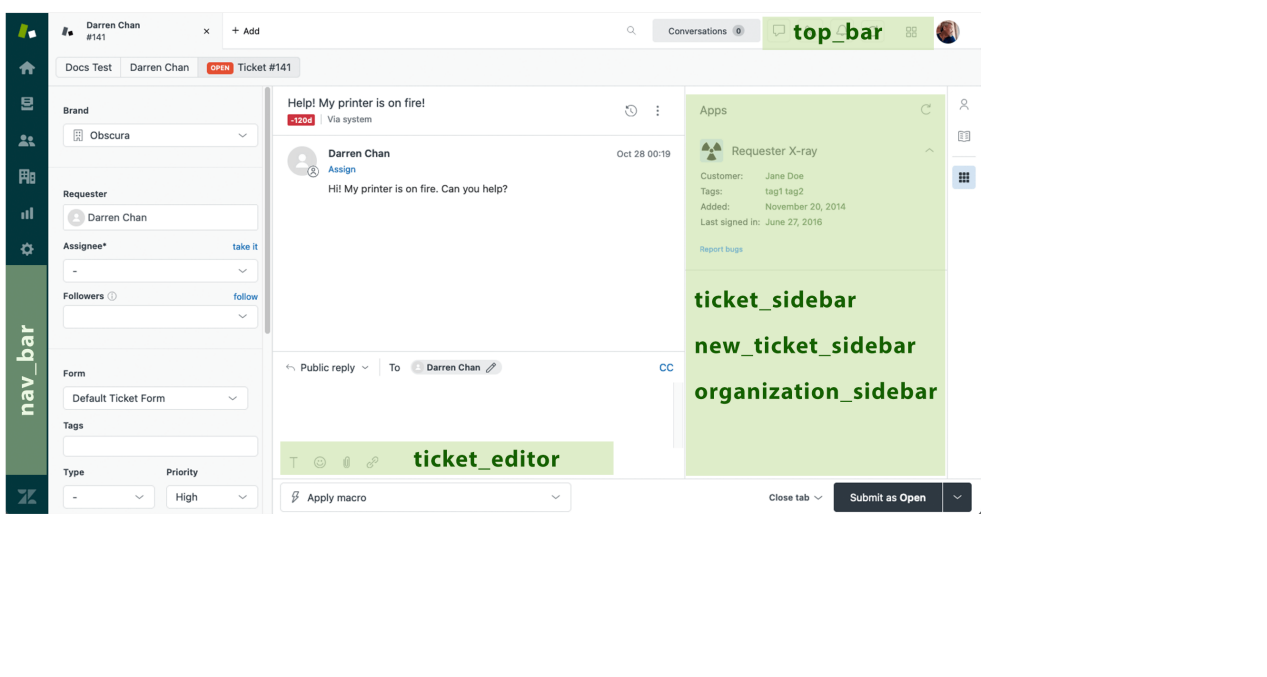
Test and publish the server code online for the client side.

TIM for Zendesk Installment

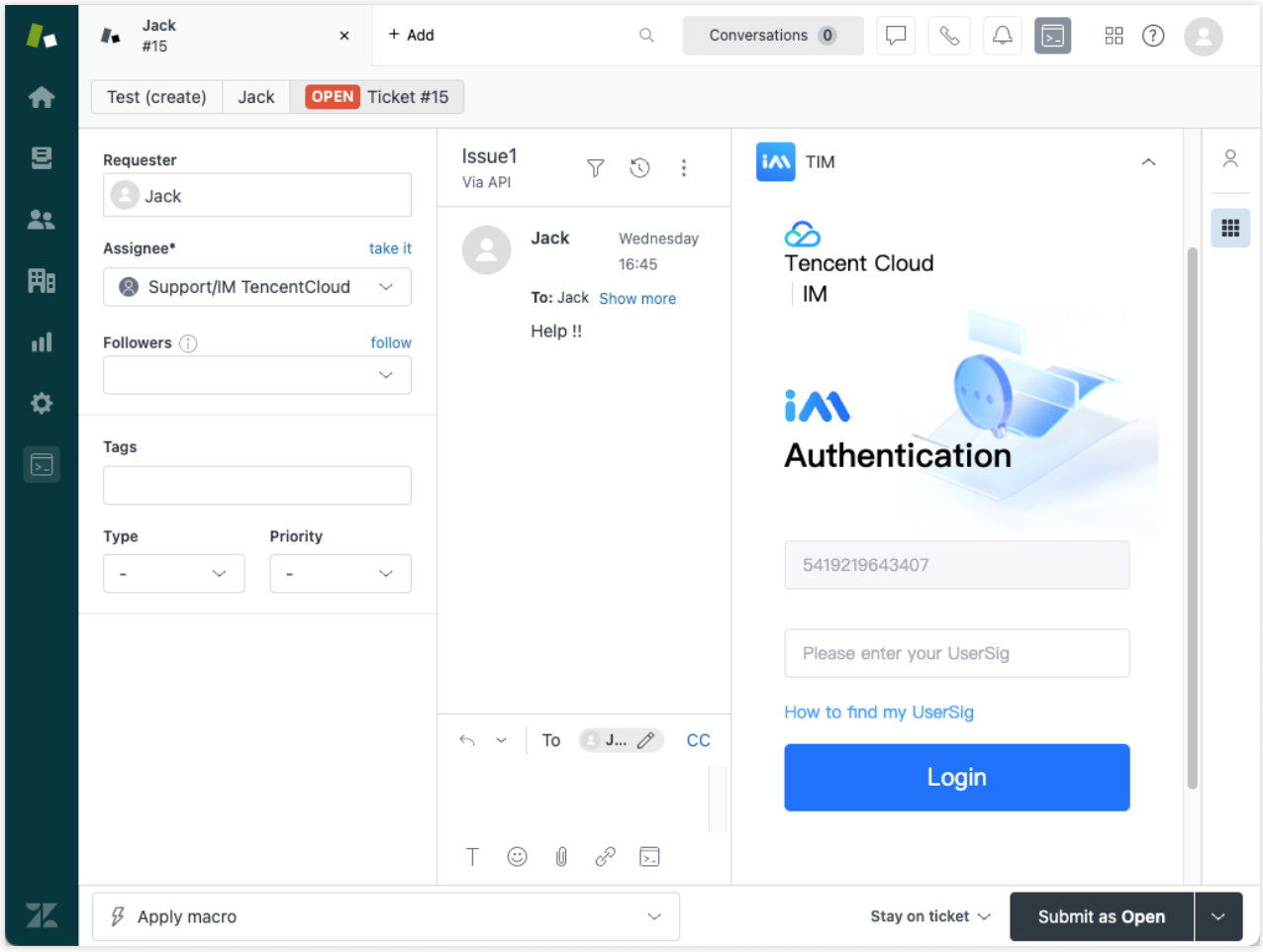
Install Tencent Cloud IM for Zendesk by searching Tencent Cloud IM for Zendesk on Zendesk marketplace.
Next, type your SDKAppId for the App to complete the installation.



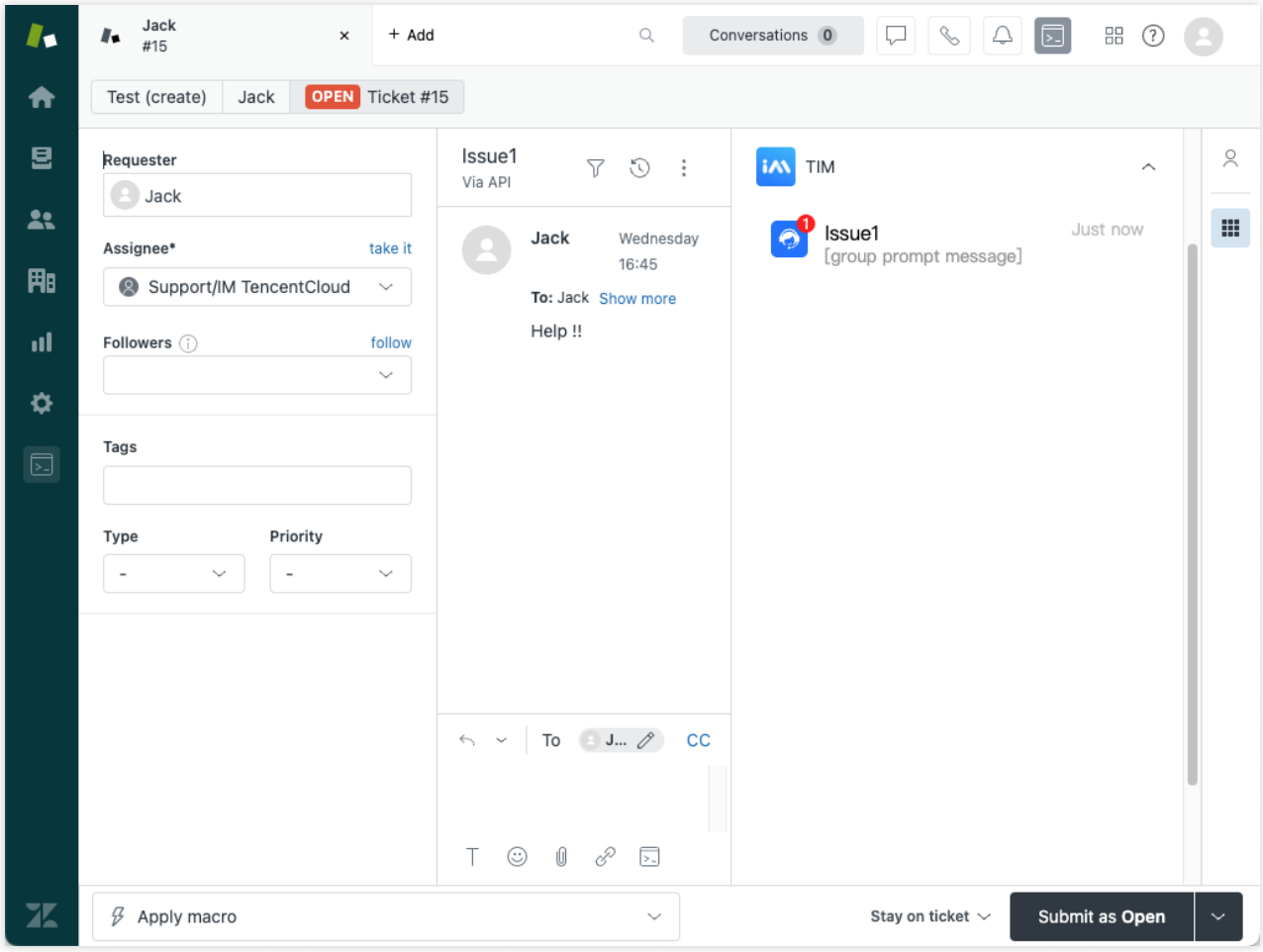
The App will show on the ticket bar area.

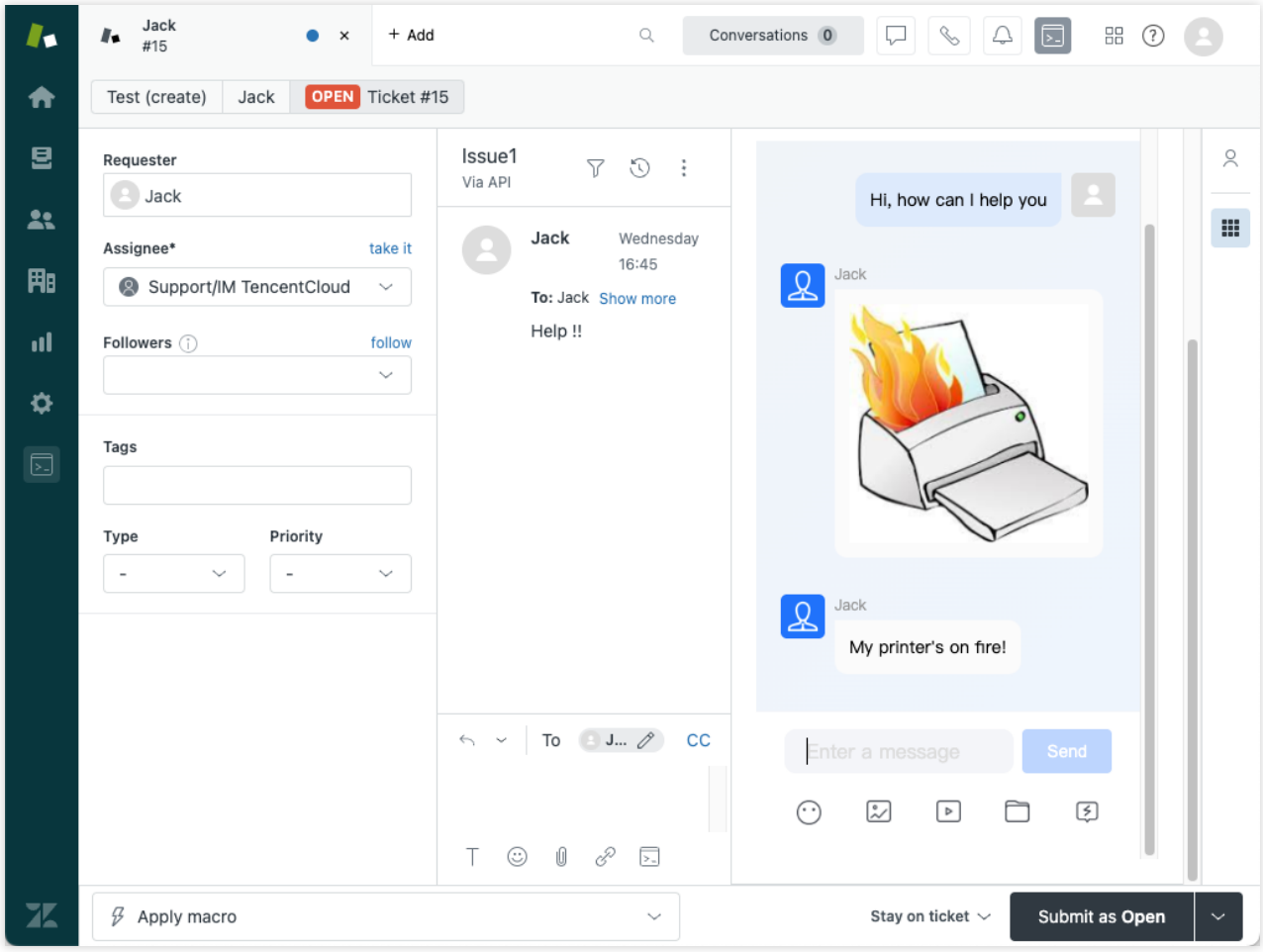


It uses agent's ID as UserID for Tencent Cloud IM, and you need to acquire UserSig for this UserID. See how to retrieve UserSig. Once logged in, the username in IM is automatically updated by the agent's name, which can be updated with other profile information on the profile page. Once logged in, you'll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





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!

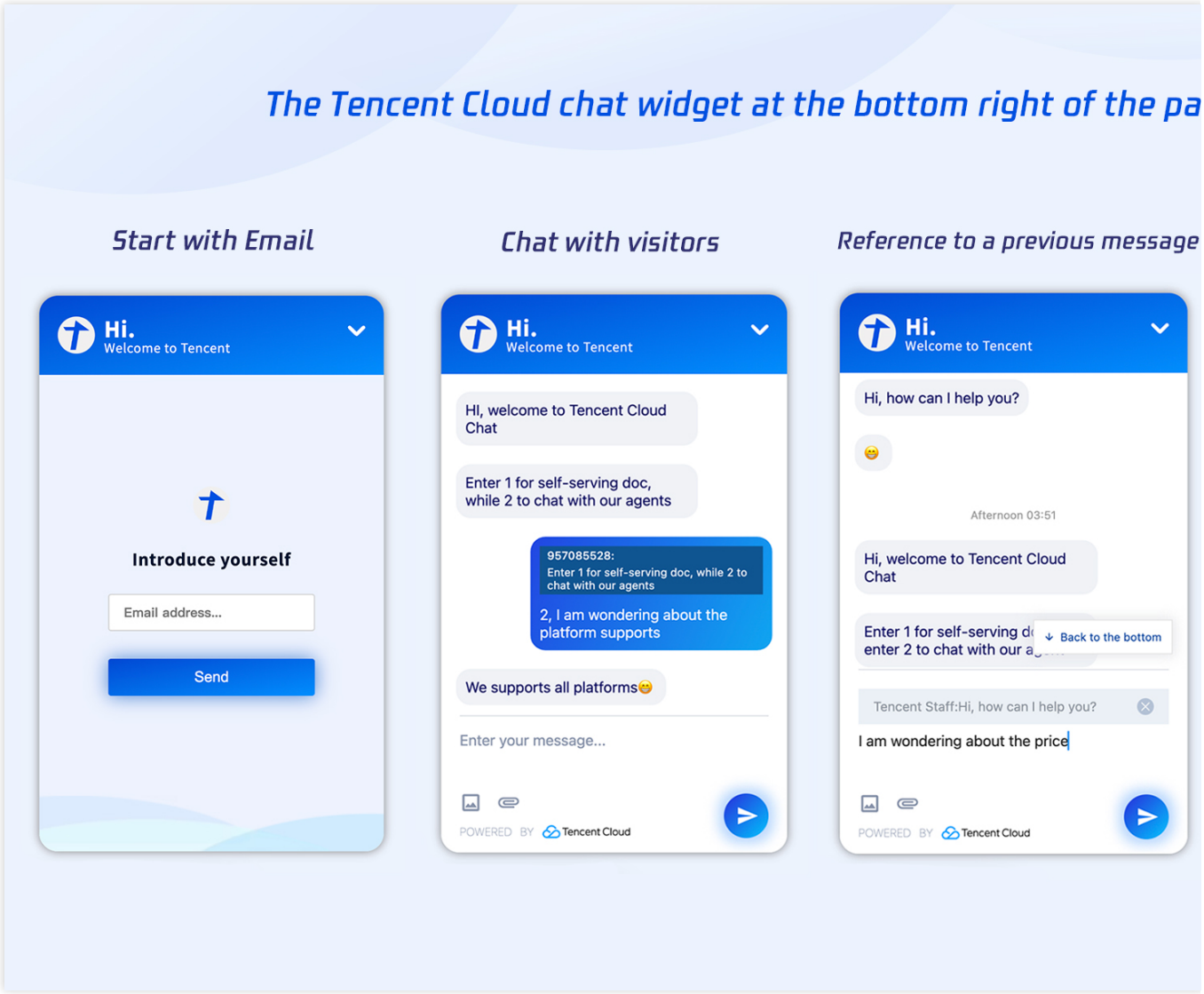
How to integrate chat widget to your Shopify online store

最終更新日：：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

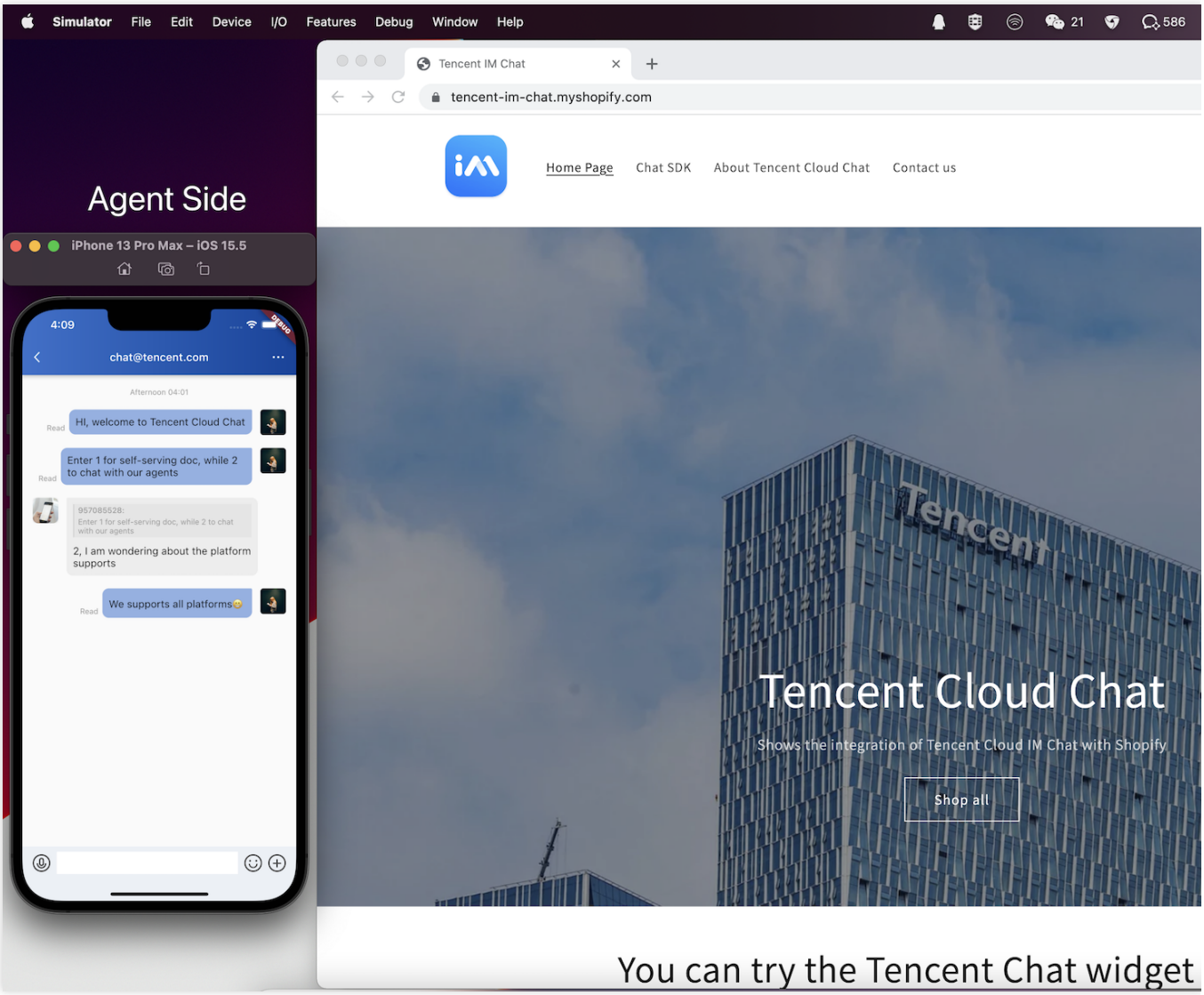
In this tutorial, we will discuss how to build this chat box on your Shopify online store with the codes we provided.

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.

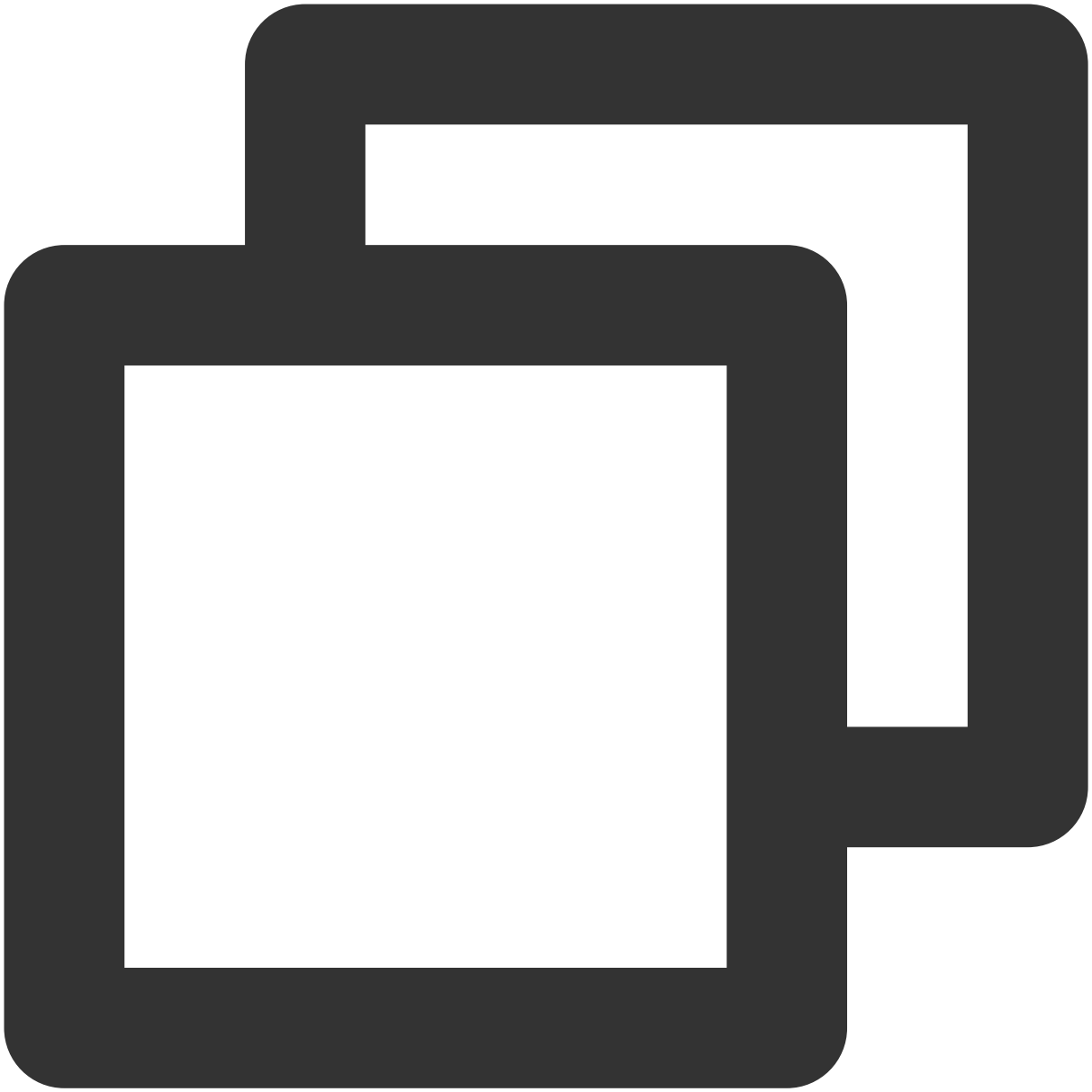
The development store is only for testing and previewing your app while developing, we will guide you to install your app to your online store by the end of this tutorial.

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.

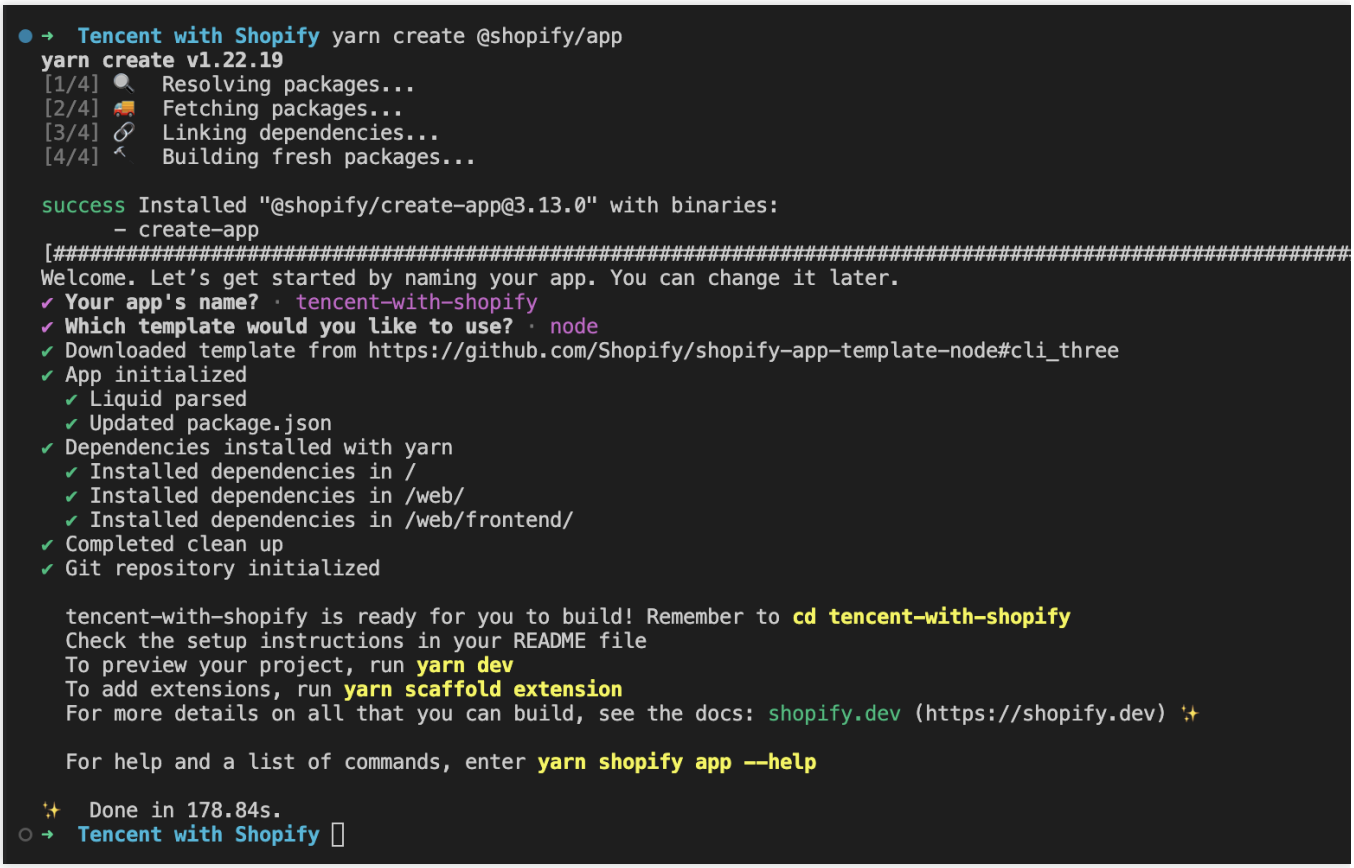
1. Navigate to the directory where you want to create your app. Your app will be created in a new subdirectory.
2. Run one of the following commands to create a new app, and select `node` as the template.



```
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:

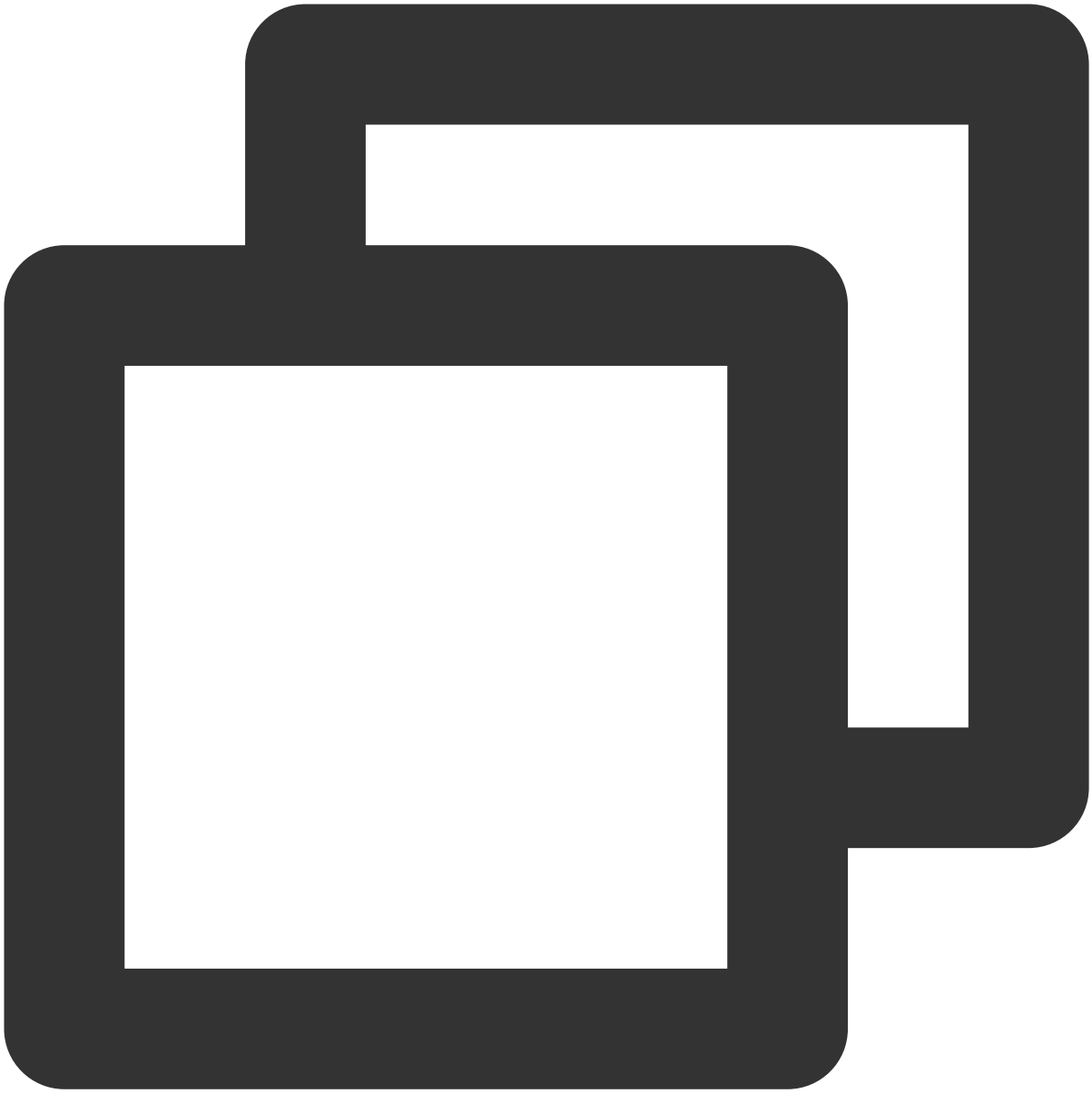


Step 2: Start a local development server

After your app is created, you can work with it by building the app and starting a local development server.

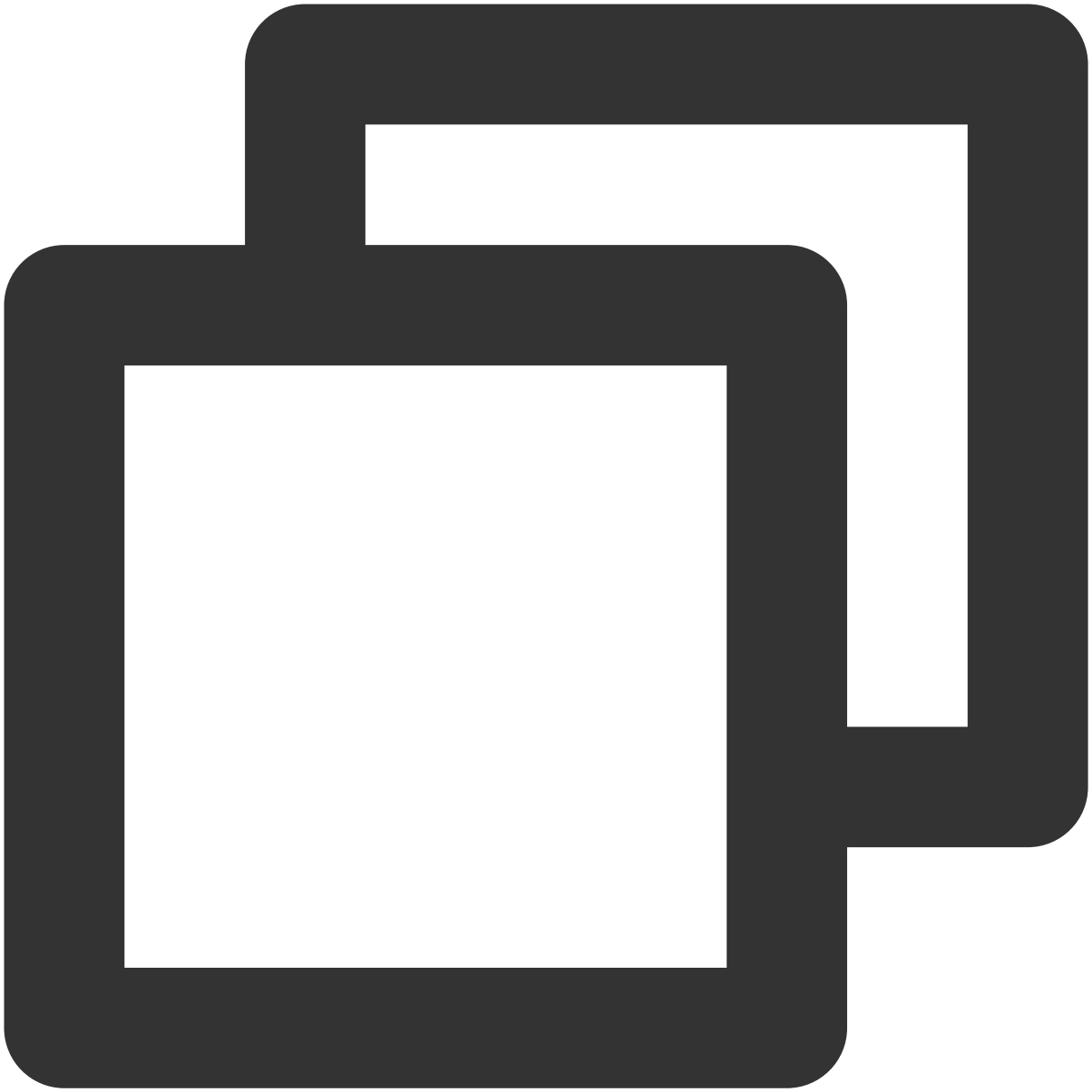
Shopify CLI uses ngrok to create a tunnel that allows your app to be accessed using a unique HTTPS URL. You need to create an ngrok account and auth token to preview your app using Shopify CLI.

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

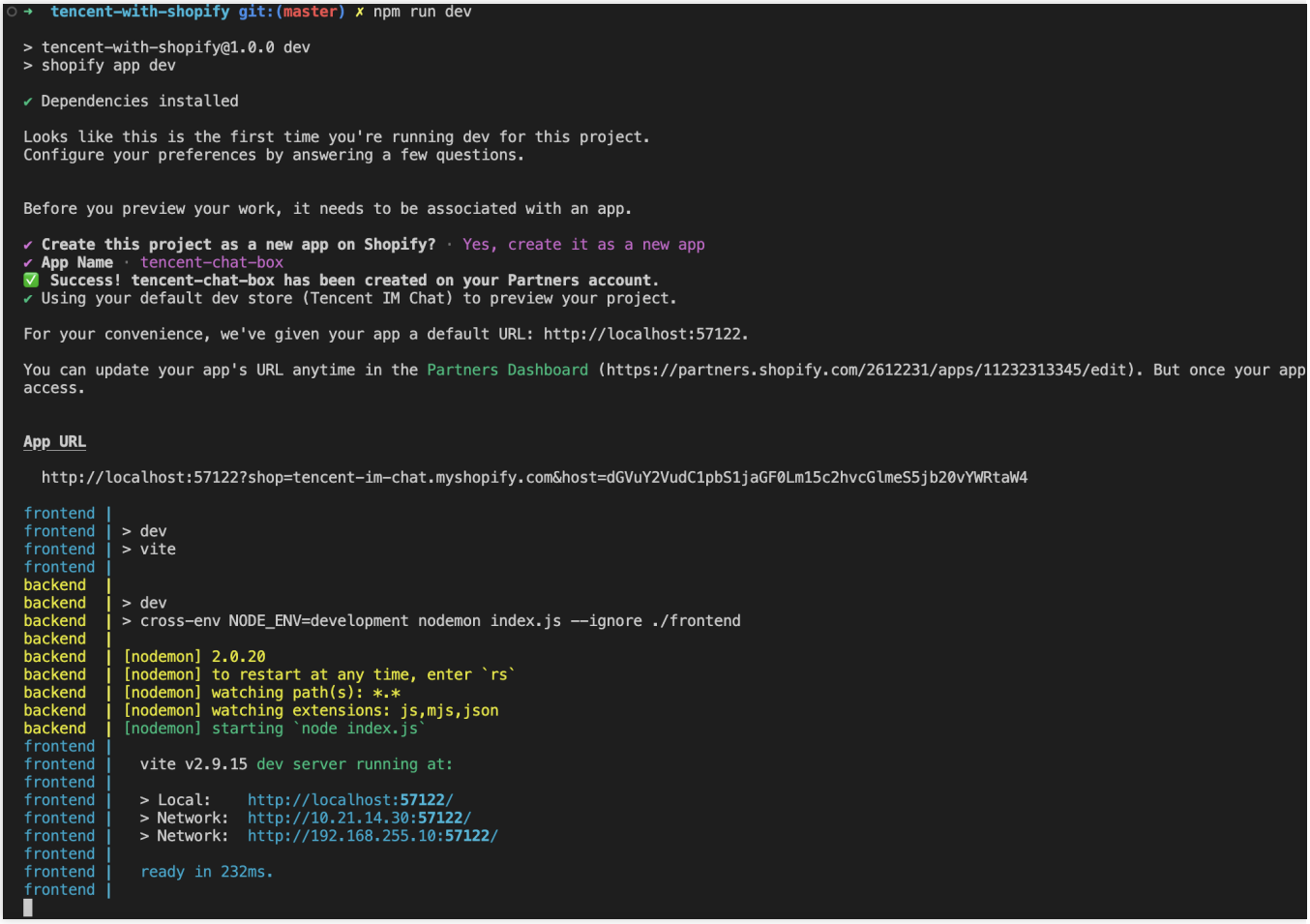
Storing your ngrok token, and then creating a tunnel between your local environment and the development store using ngrok

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` :



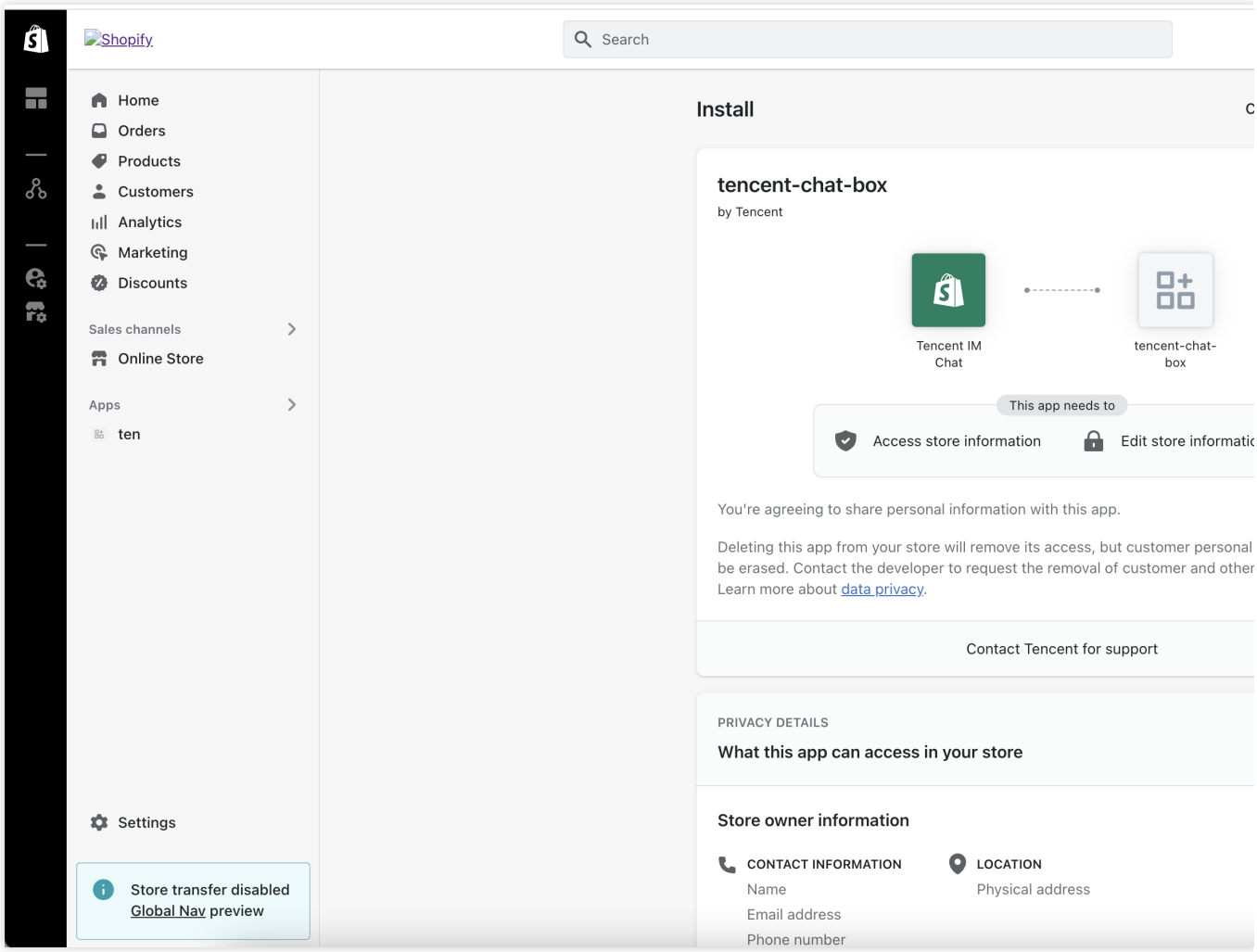
Step 3: Install your app on your development store

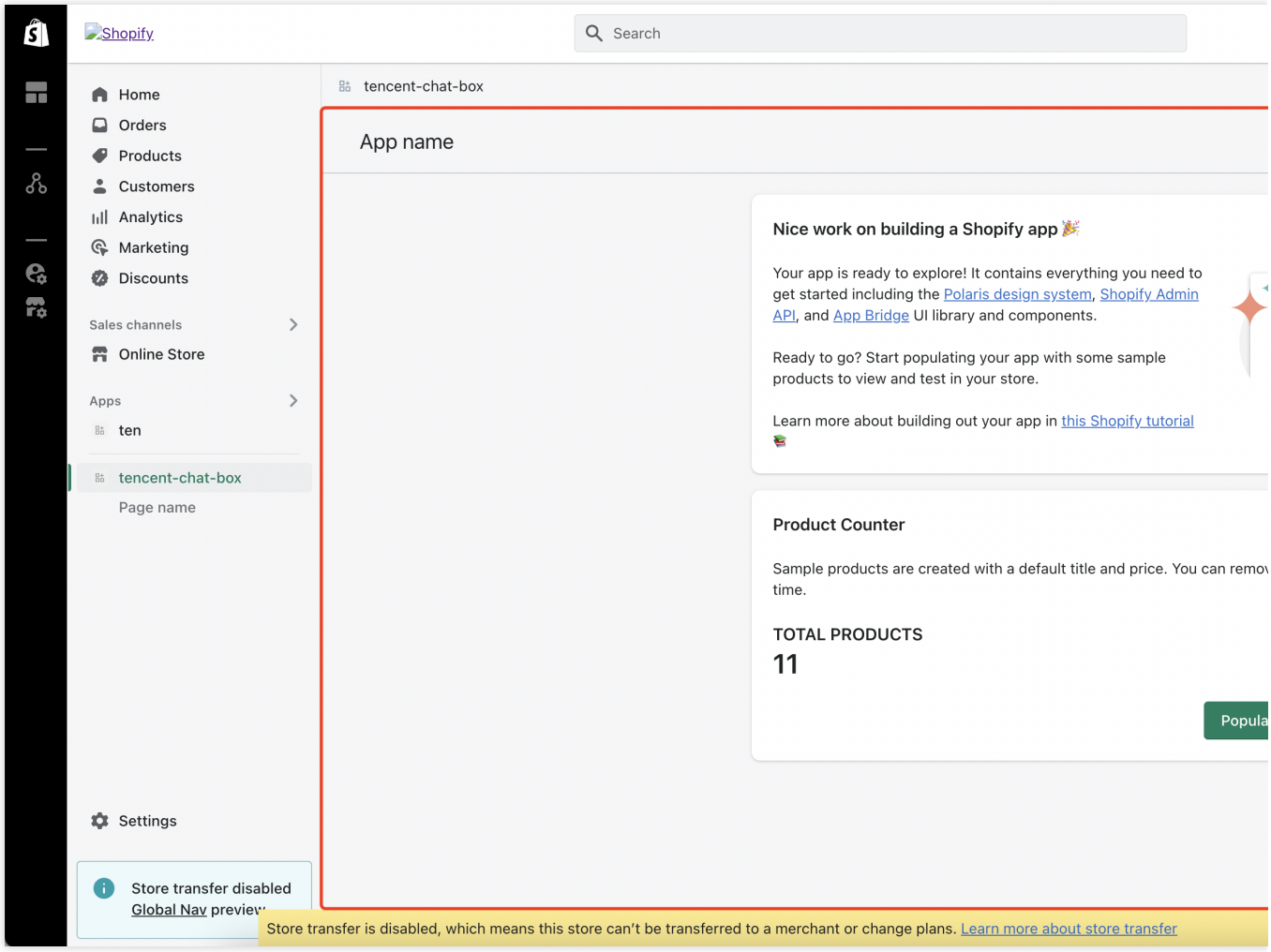
With the server running, open the URL in the App URL section of the terminal output in the previous step.

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.

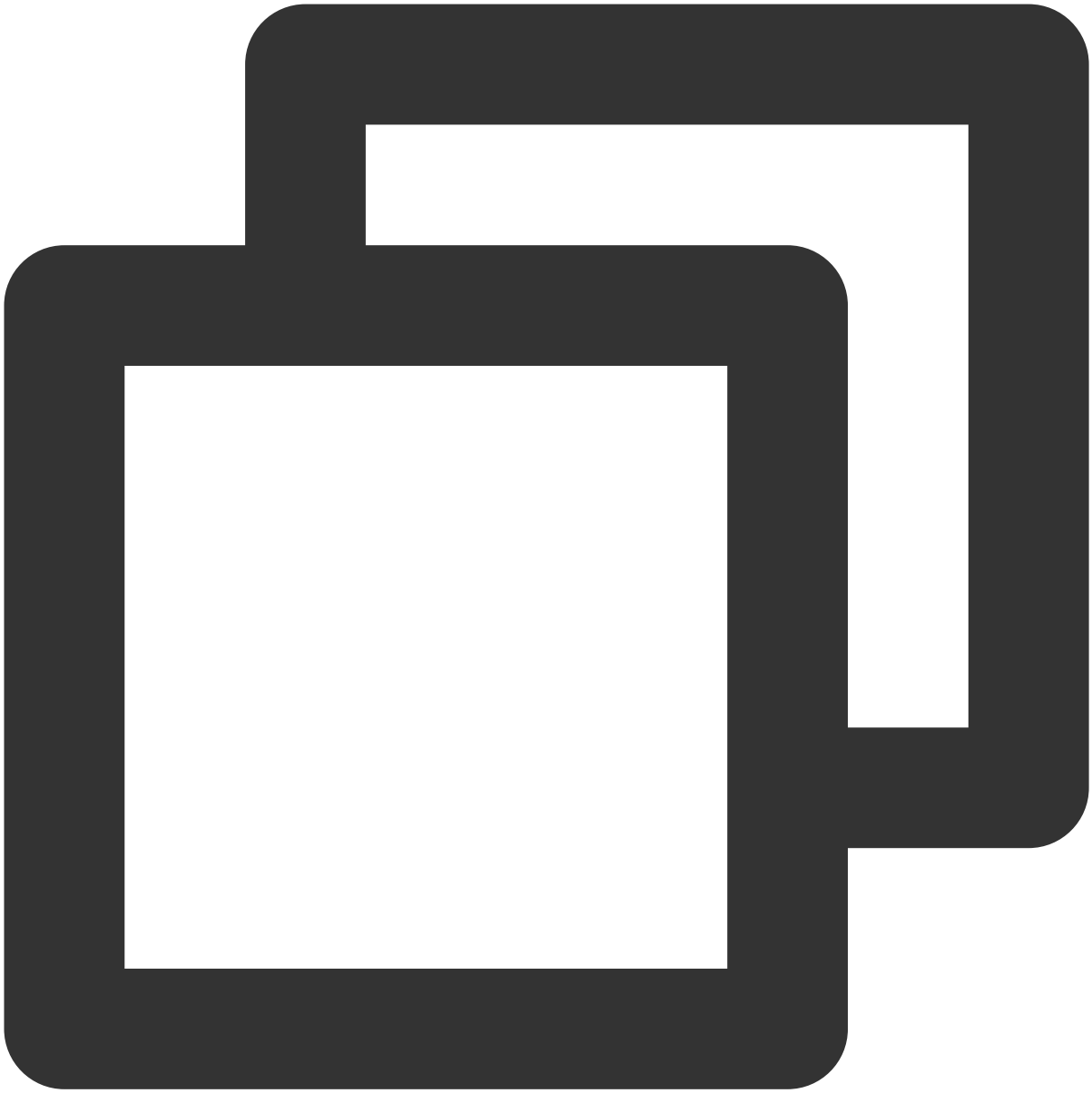
1. Replace the `web/frontend/pages/index.jsx` file.

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`.



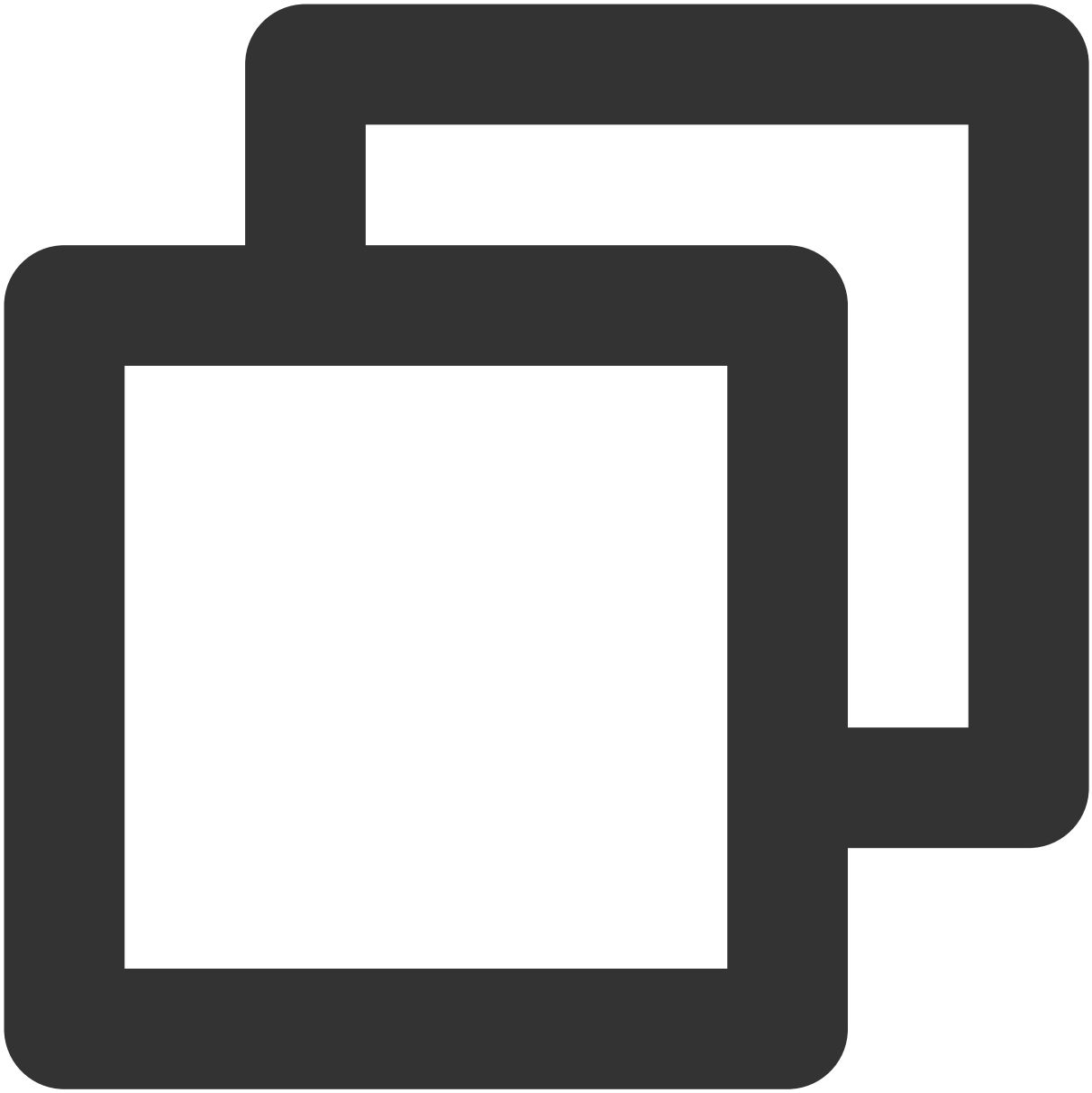
```
export { Guide } from "./Guide";
```

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

Logo, named as `logo.png` .

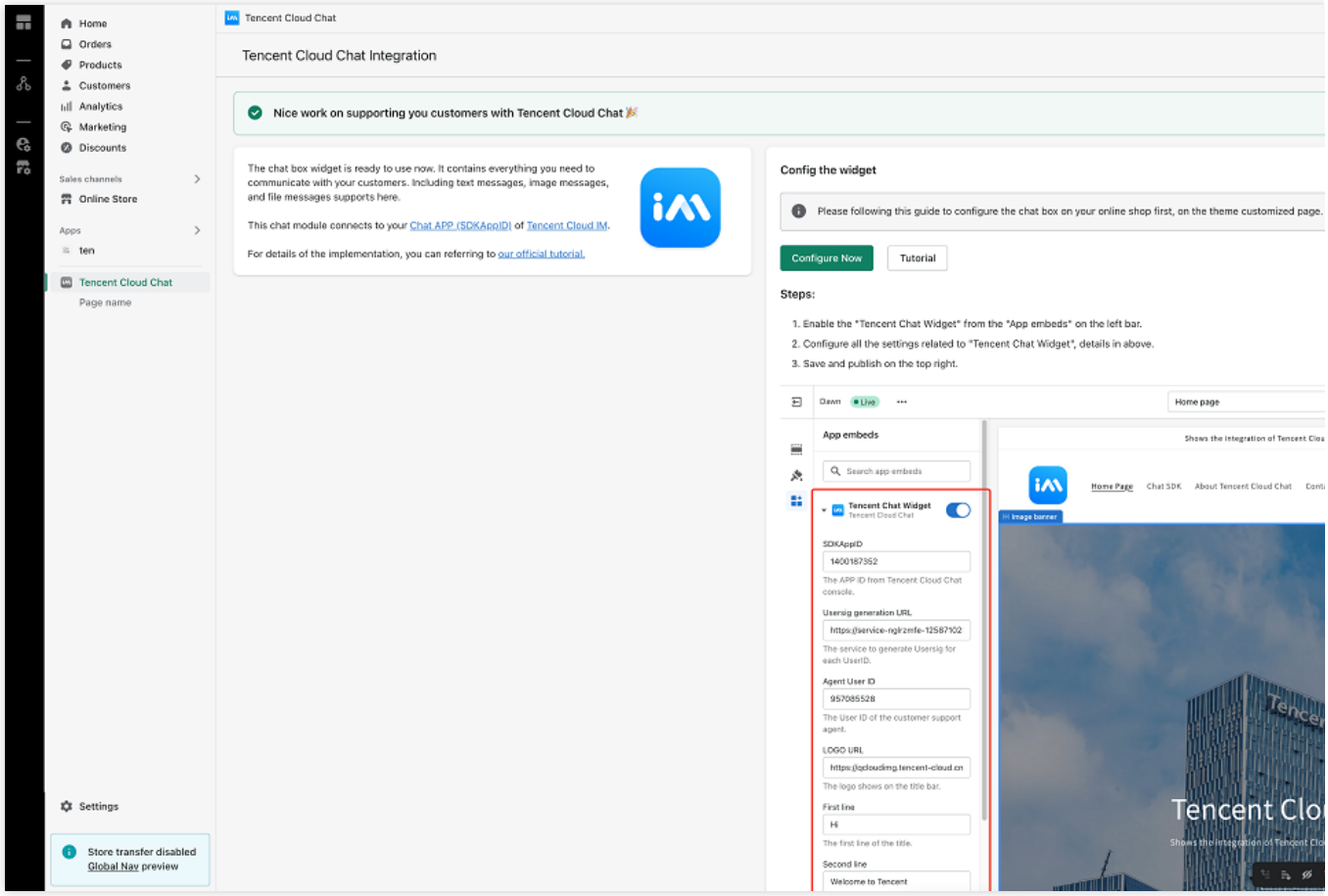
Demo instructions, named as `demo.png` .

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.



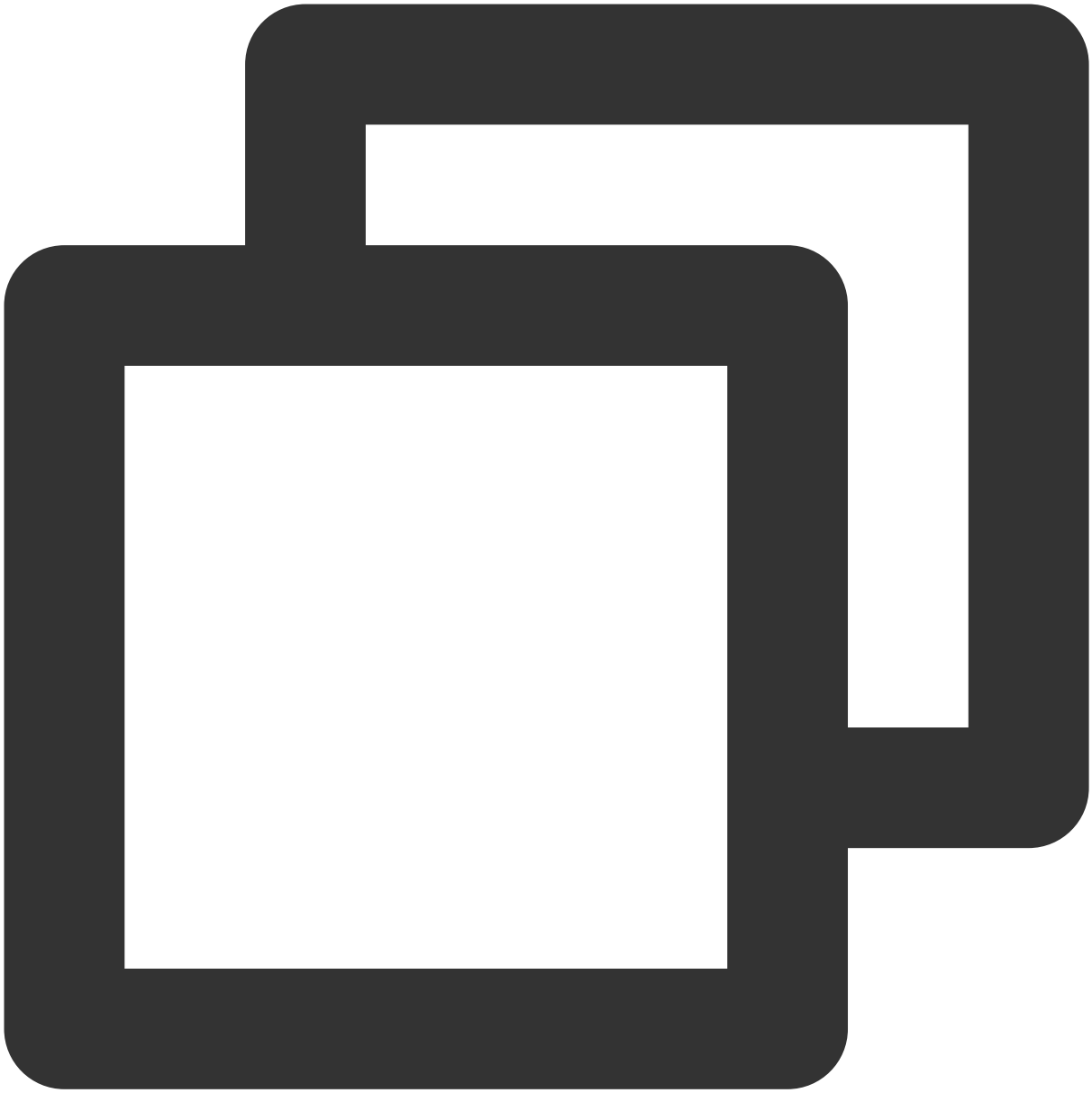
Step 5: Create a new extension

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.

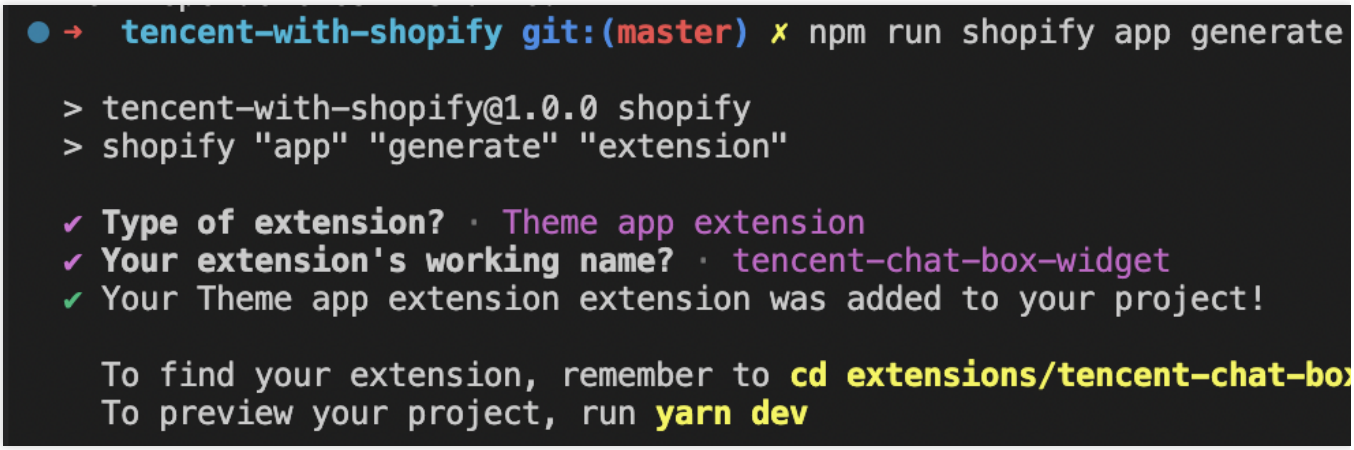
2. Run the following command to start creating the extension, choose `Theme app extension` as the `Type of extension` .



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

Note:

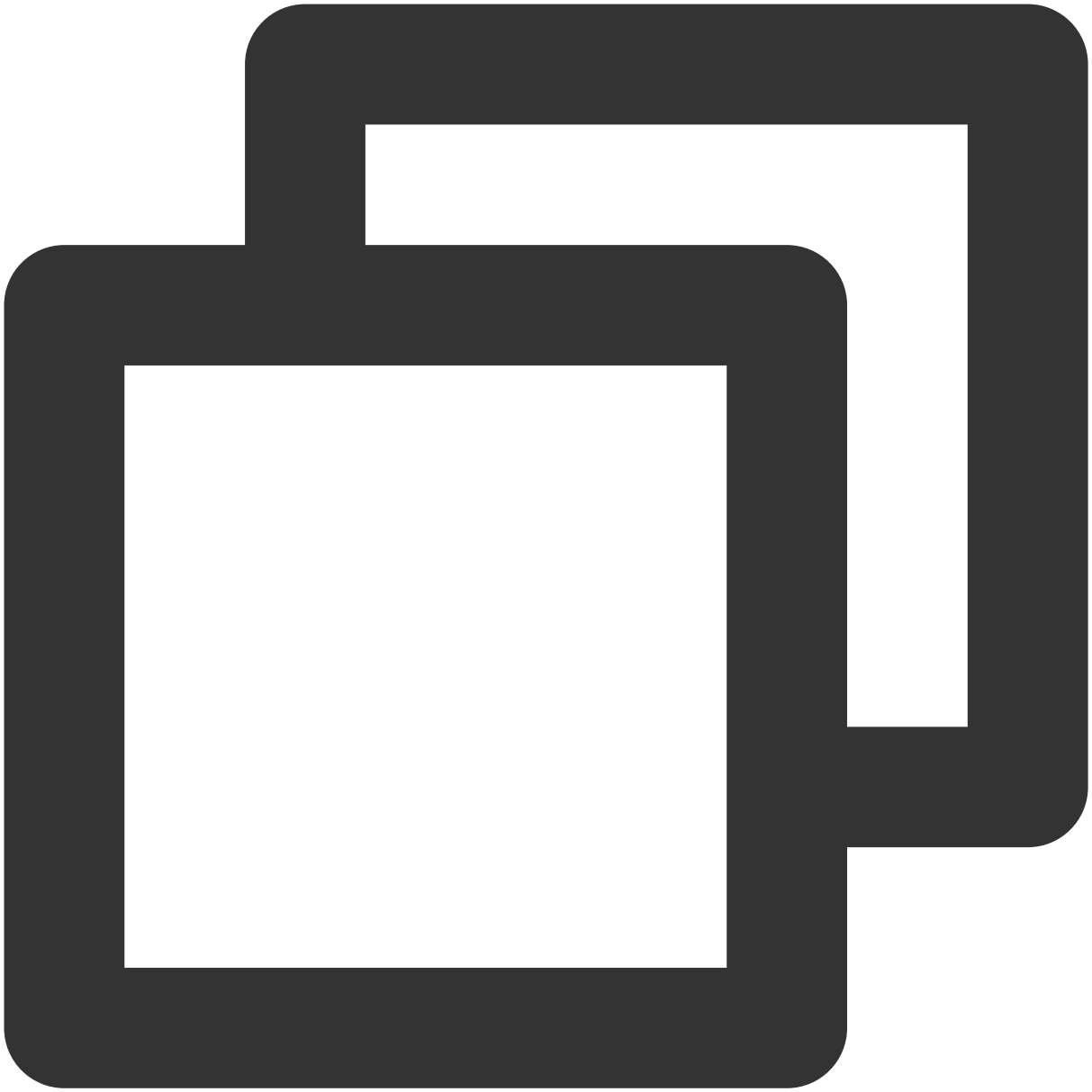
Please make sure you use Shopify CLI 3.0 or higher, if this command does not work.



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

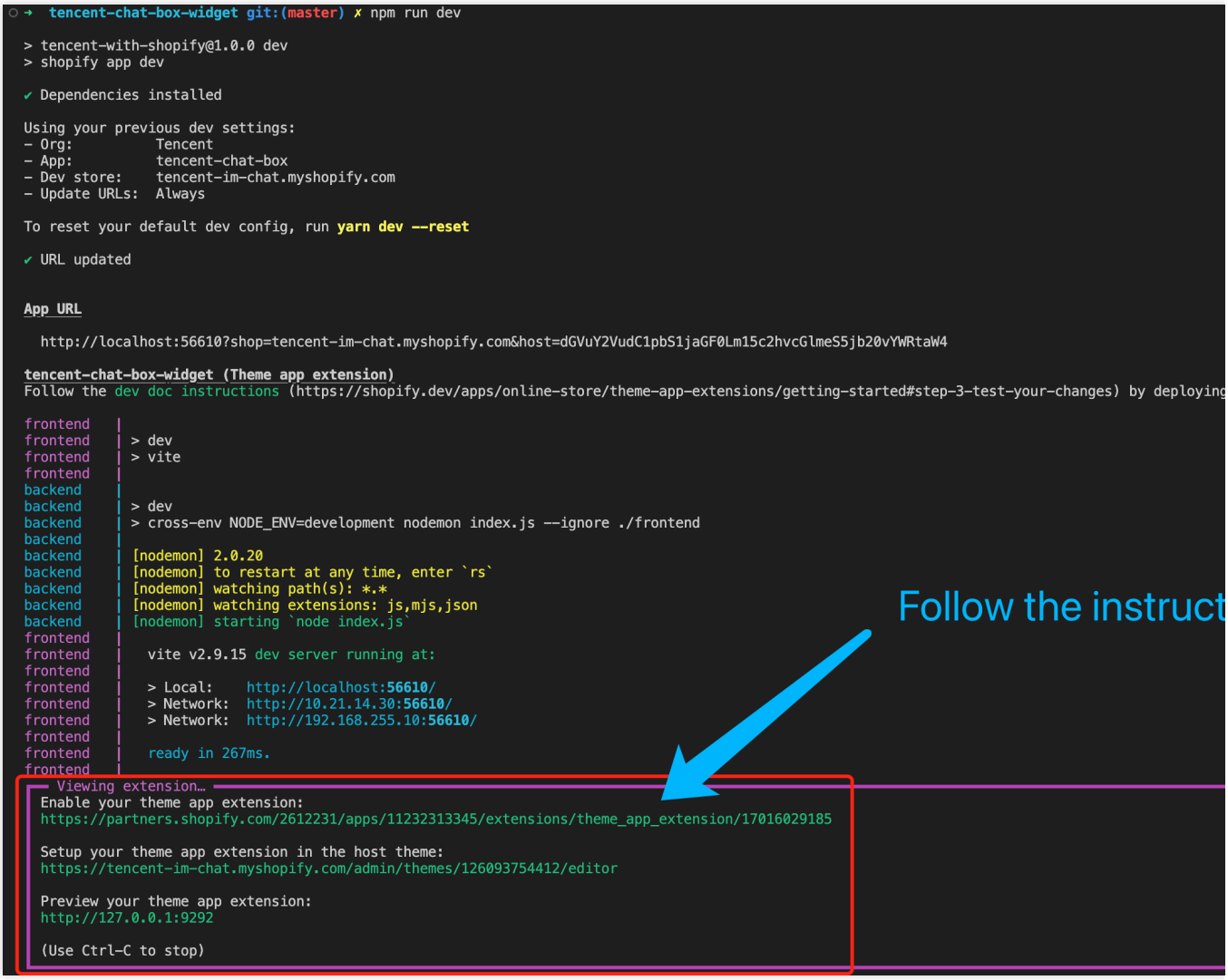
The `dev` command returns a preview URL that reloads local changes, allowing you to preview changes in real time using the store's data. This preview is only available in Google Chrome.

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



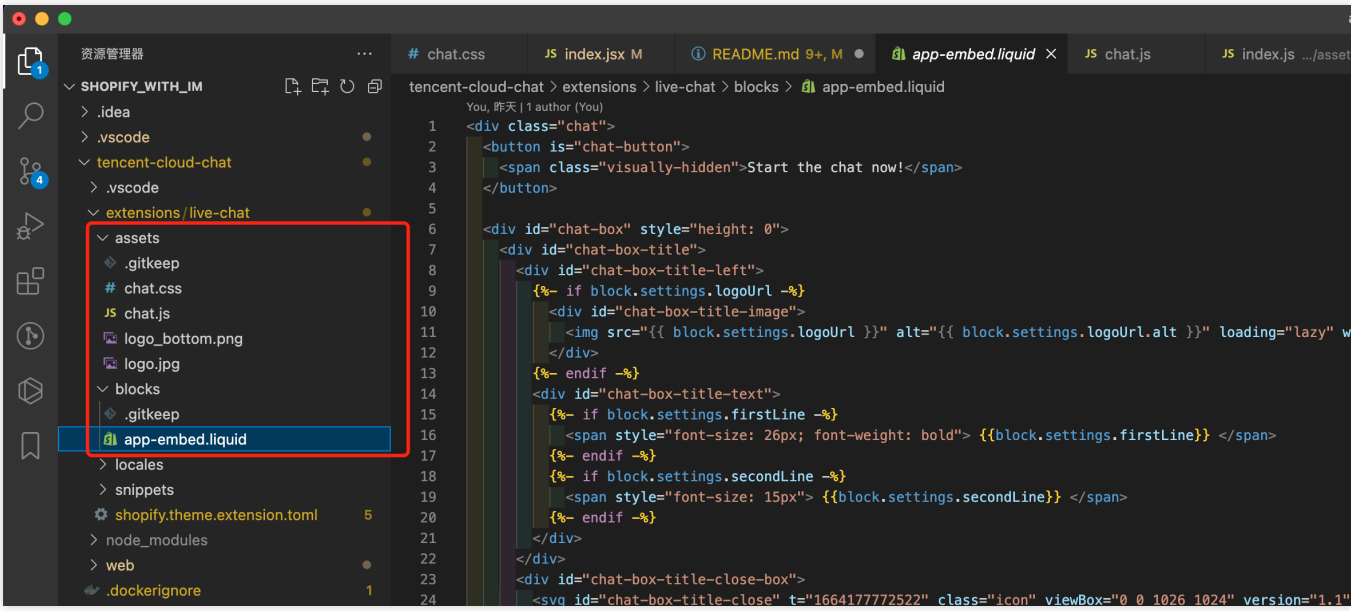
```
npm run dev
```

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.

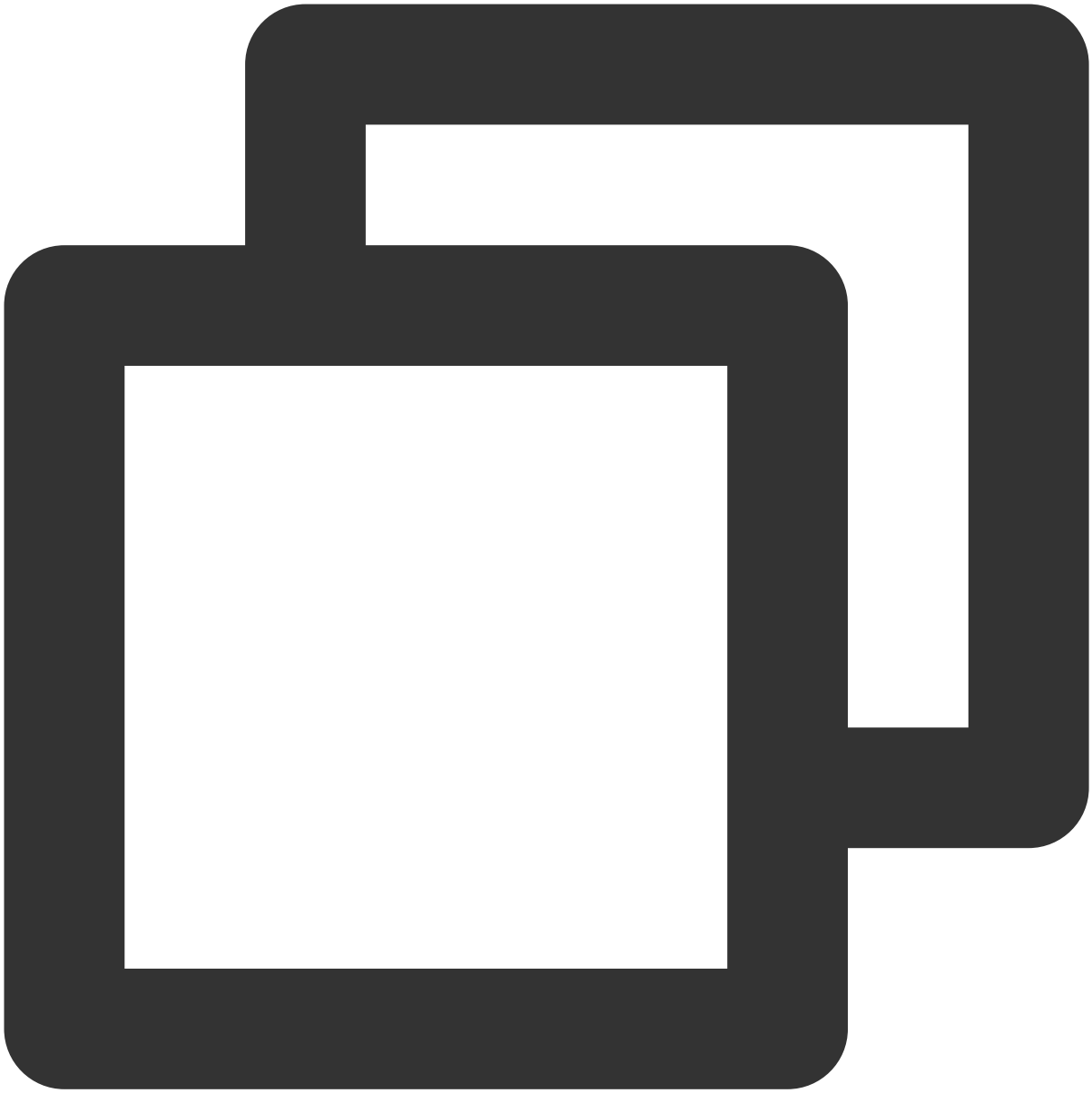
Both `UserID` and `UserSig` are needed for login before chatting, while the `UserSig` is generated with `UserID`, `SECRETKEY` and `SDKAppID` dynamically.

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.

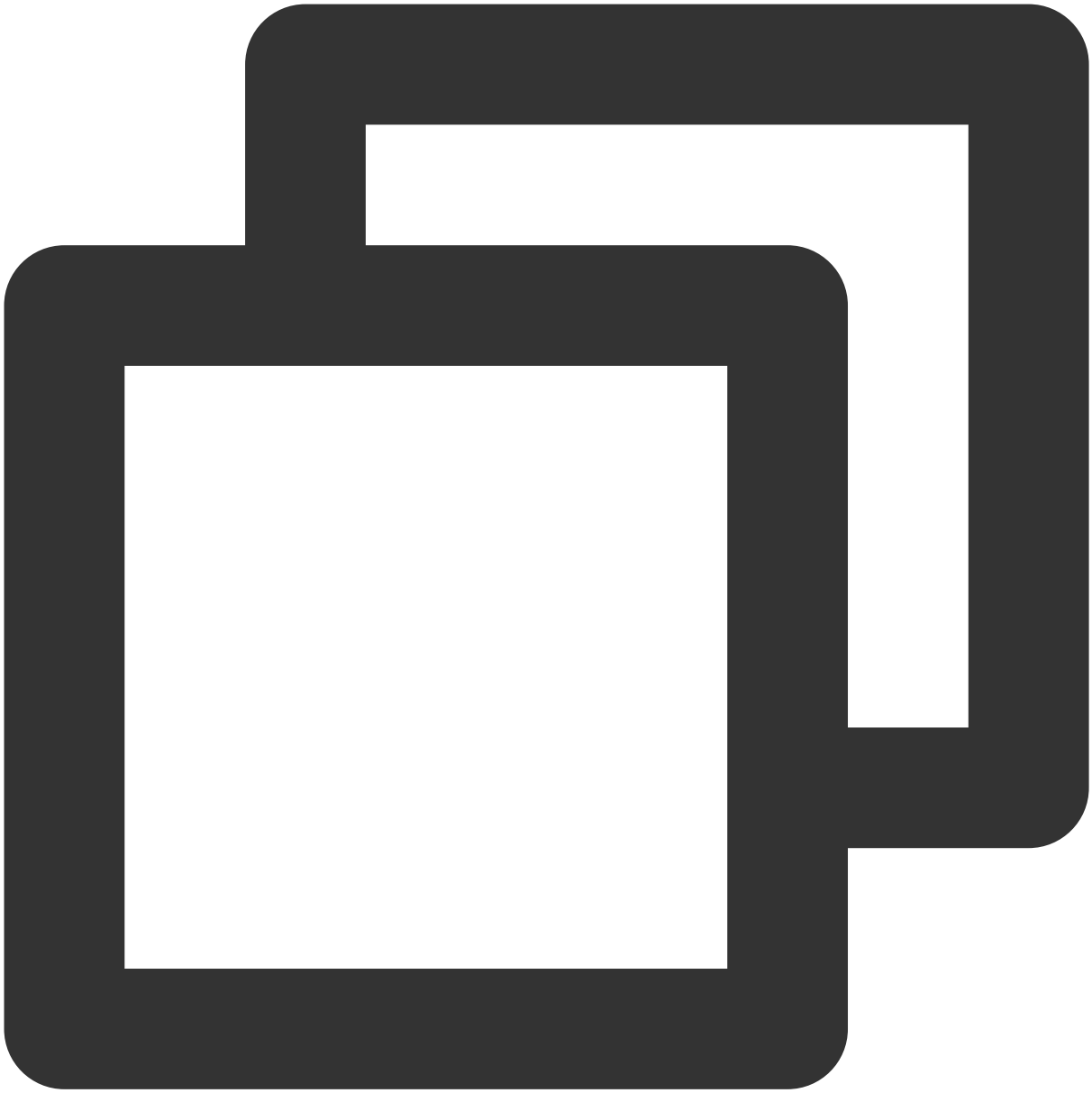
The communication between the chat box and your server should use RESTful API, with POST request.

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:

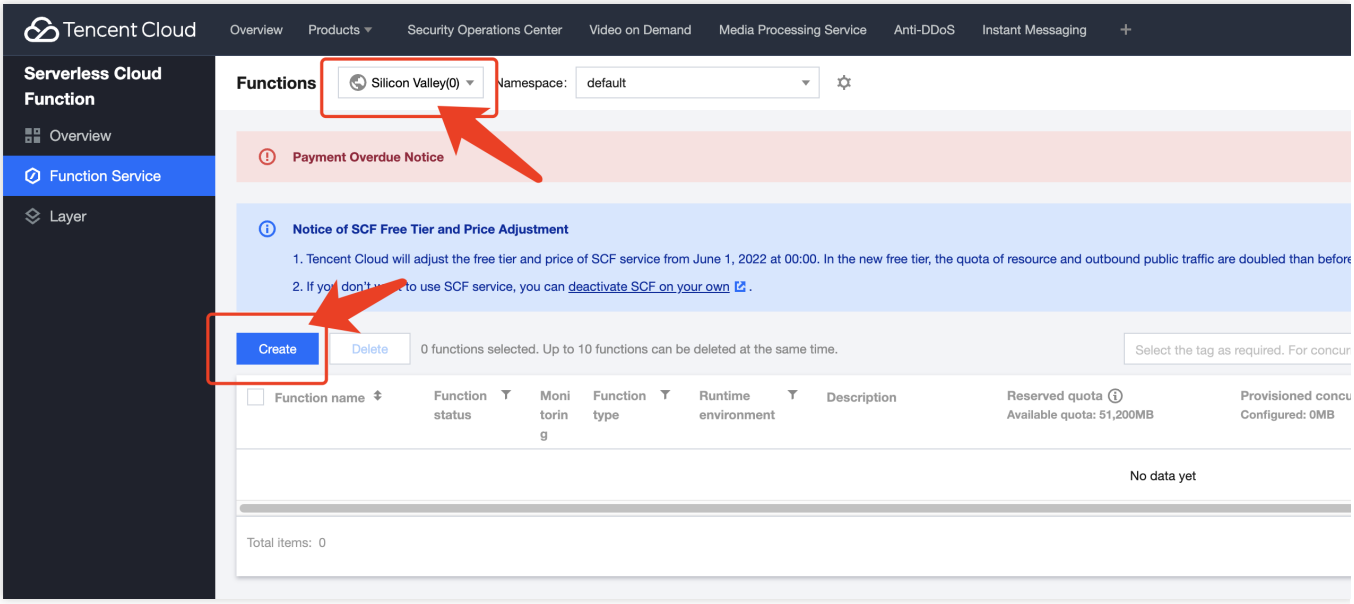


```
{
   "userID": "",
   "userSig": ""
}
```

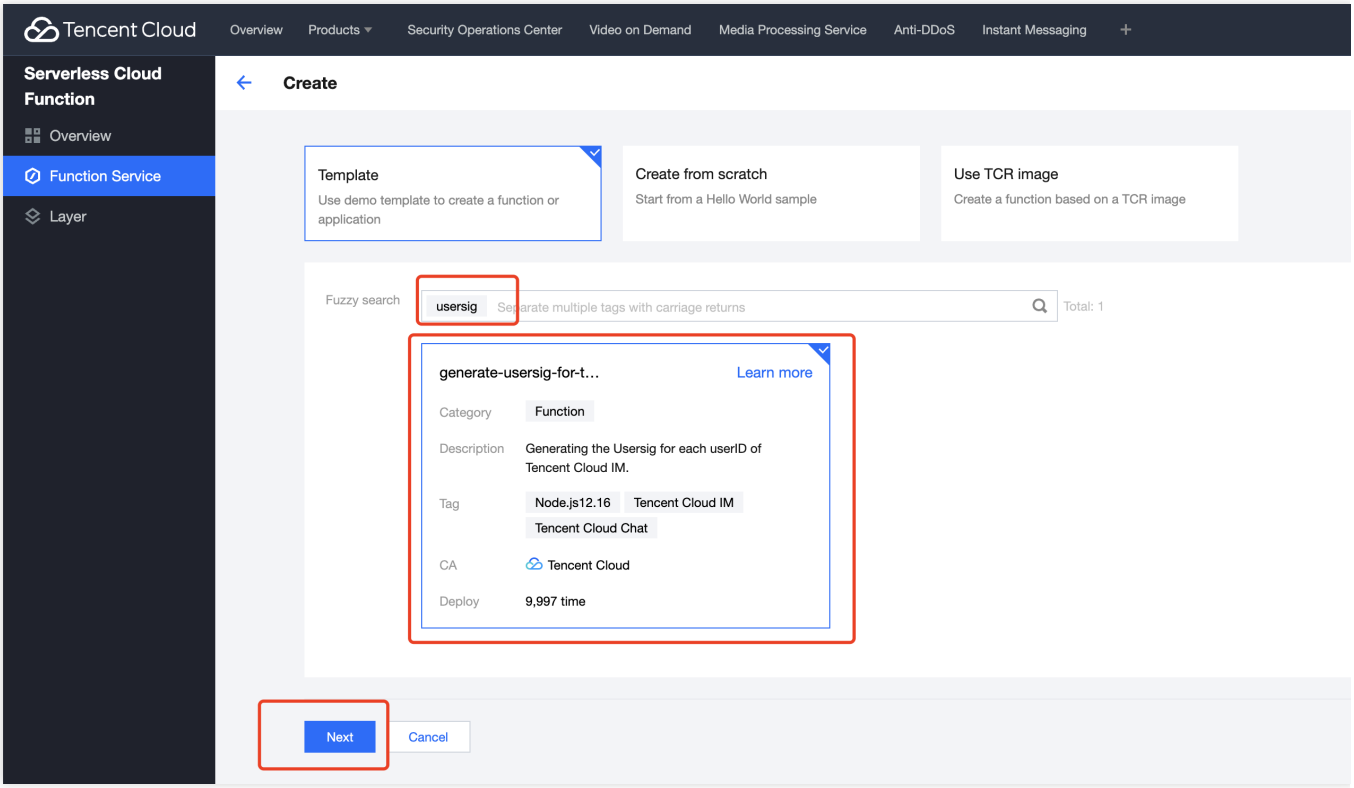
Set up the `UserSig` generation service with serverless function

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` .

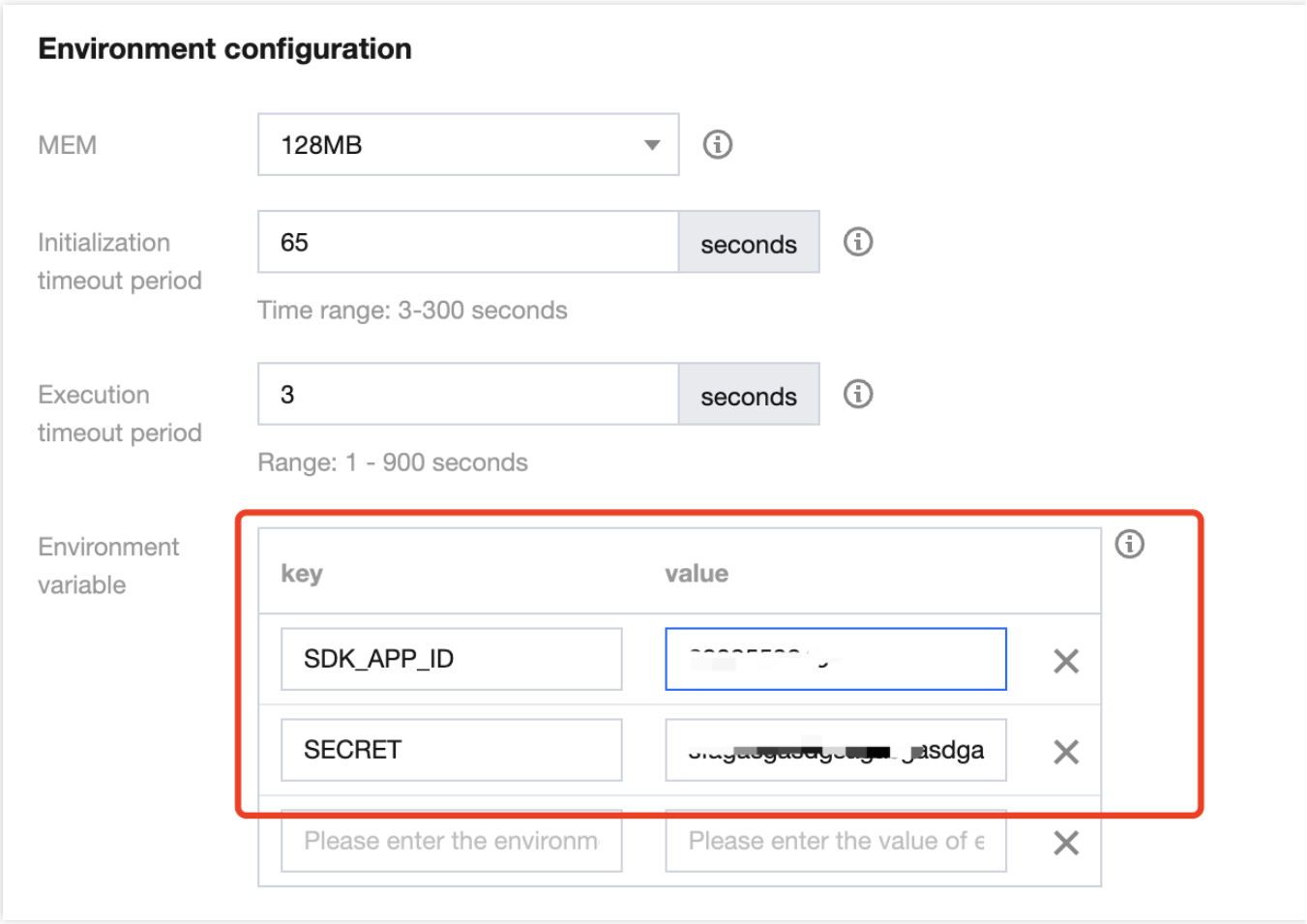


3. Specify the `Function name` based on your needs.

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

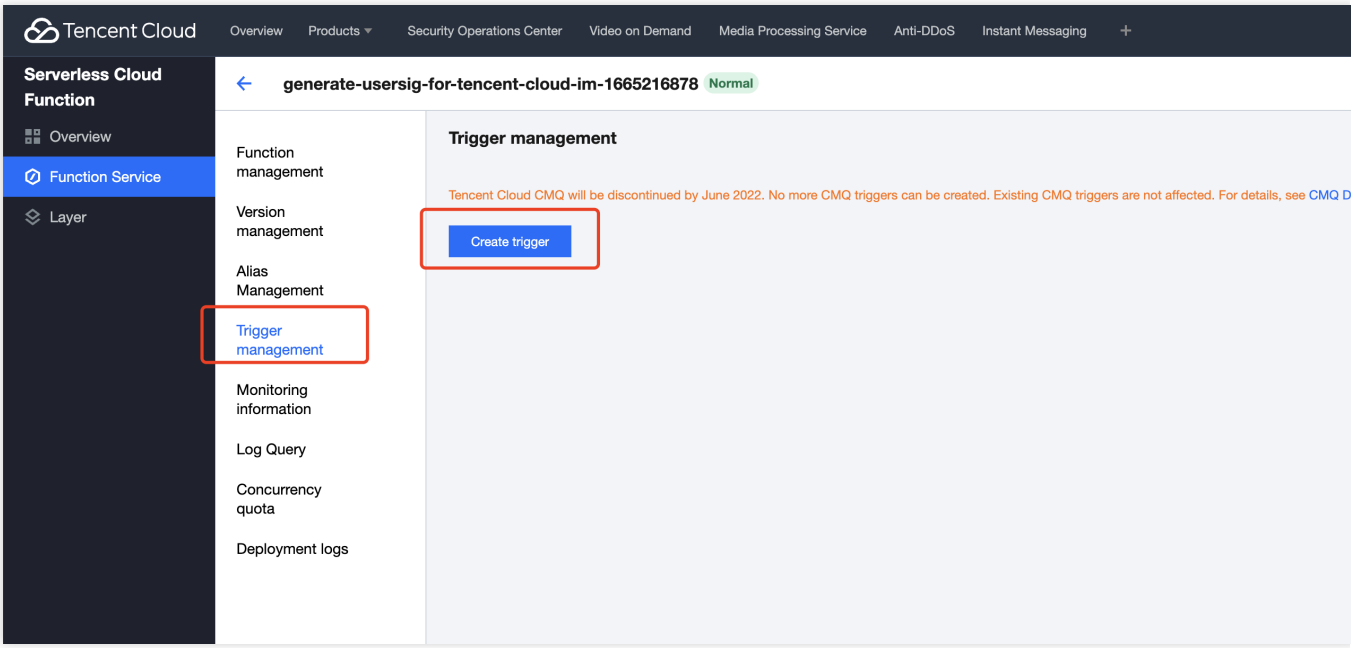
key	value

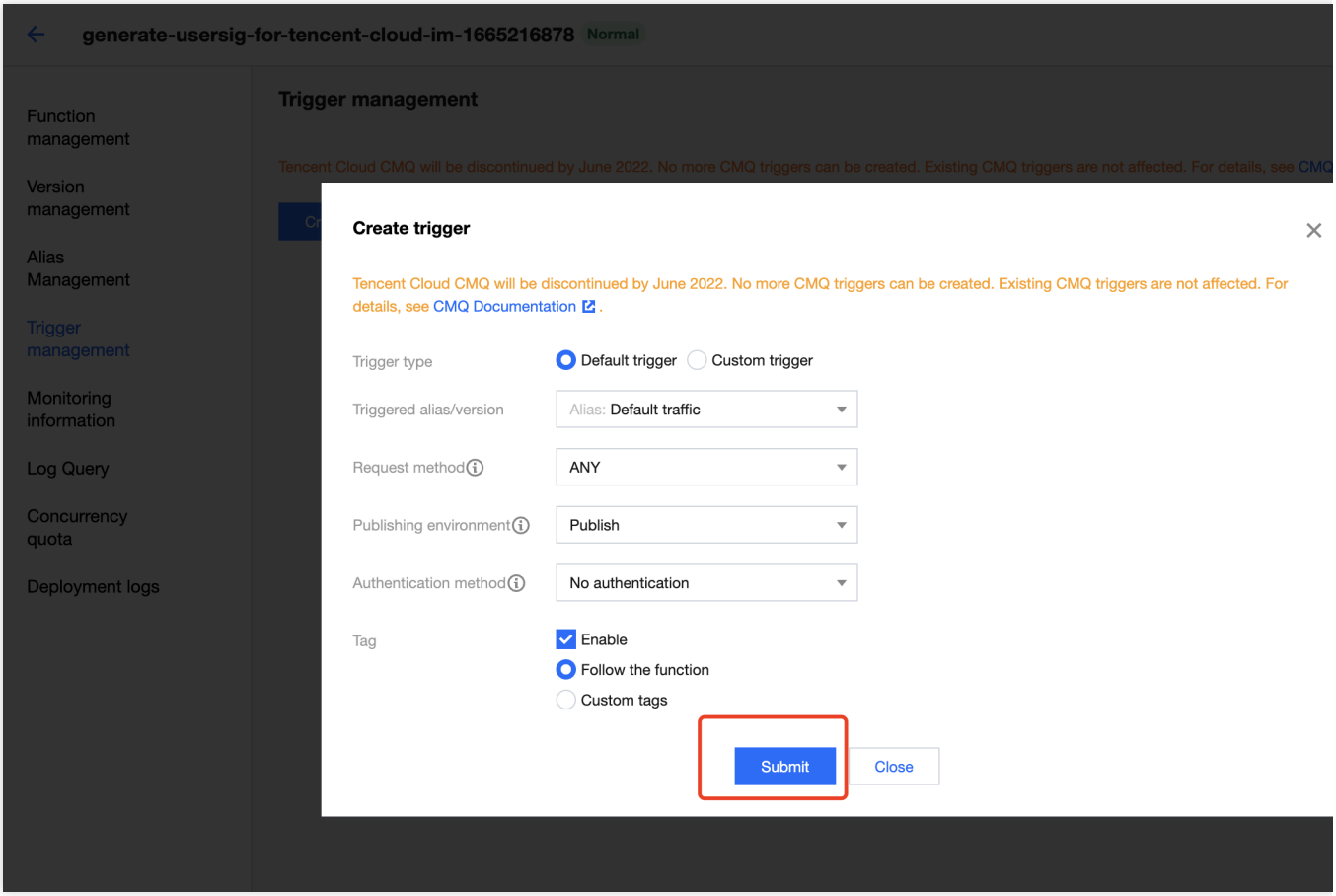
SDK_APP_ID	(Your SDKAppID)
SECRET	(The `Key` shows on the main page of the IM console)



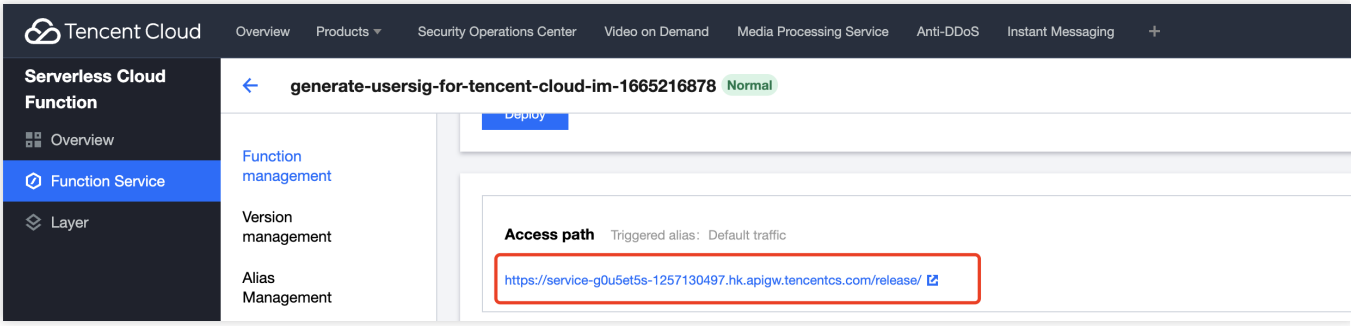
5. After deployment, create a trigger for it by clicking `Trigger management` on the left bar, then click `Create trigger`. Use the default settings on the modal opened, then click `Submit`.

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

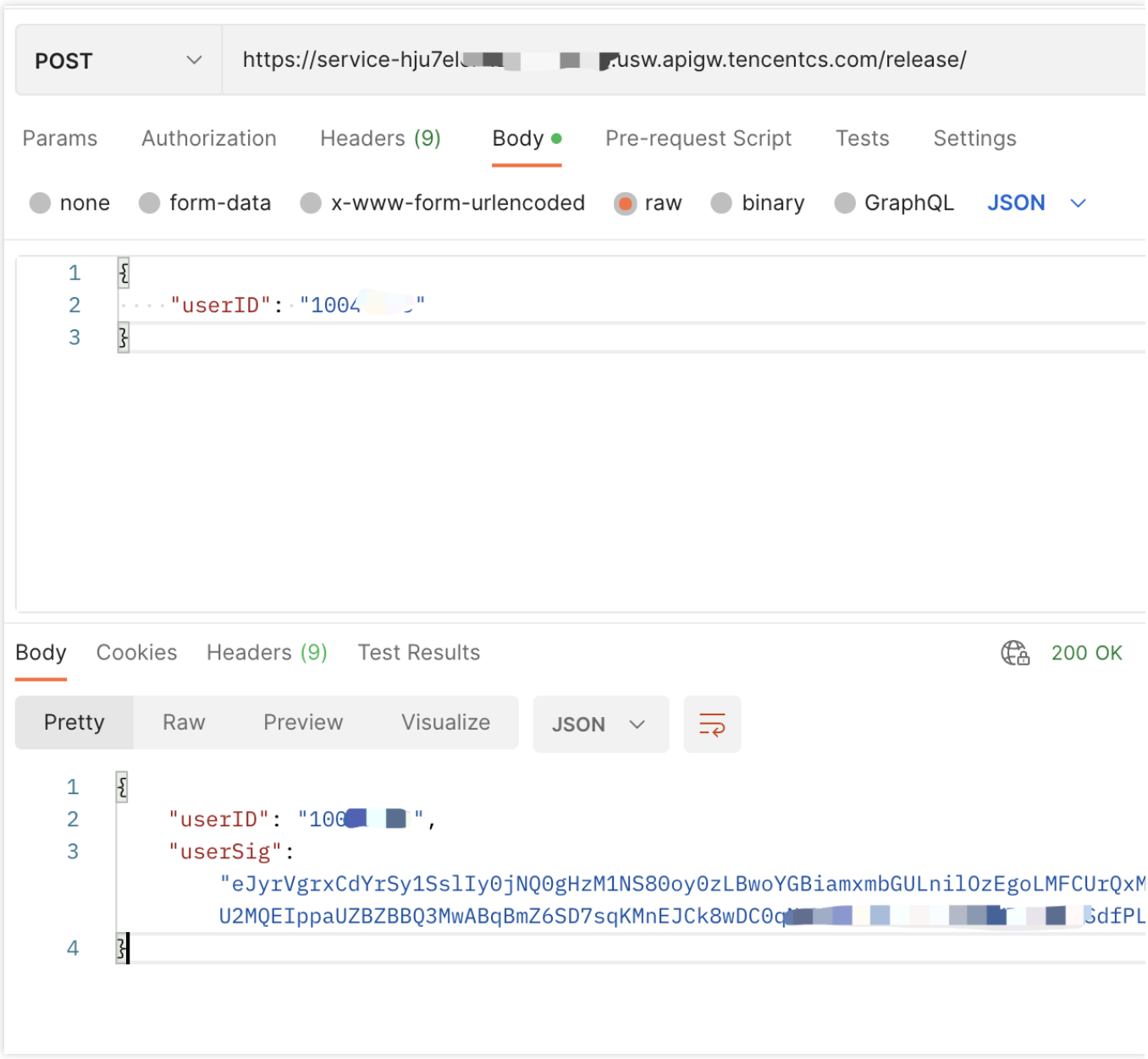




6. Find the URL of the API created from `Function management` => `Function codes` => `Access path` . This URL will be used in the following steps.

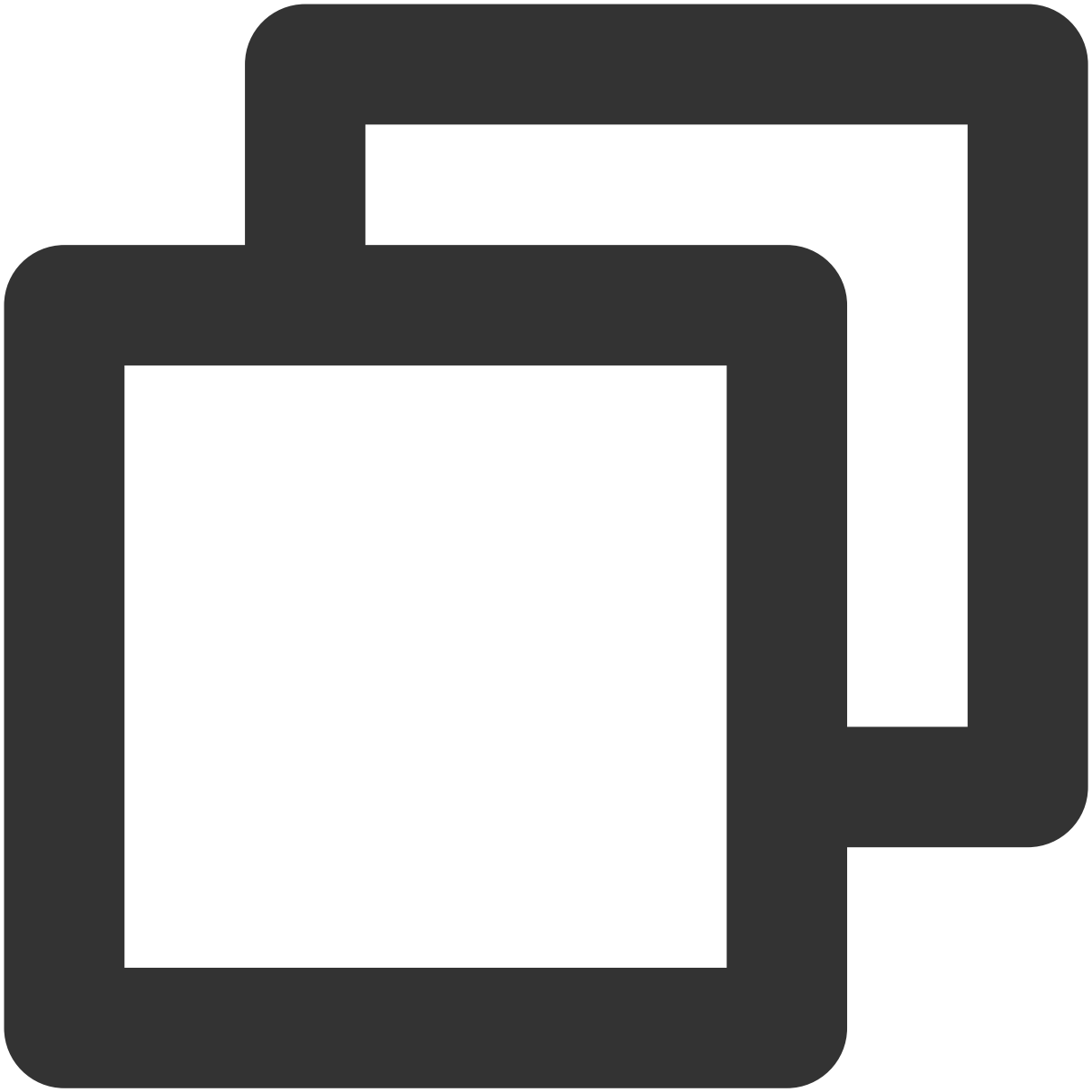


7. [Optional] You can test it by `Postman` .



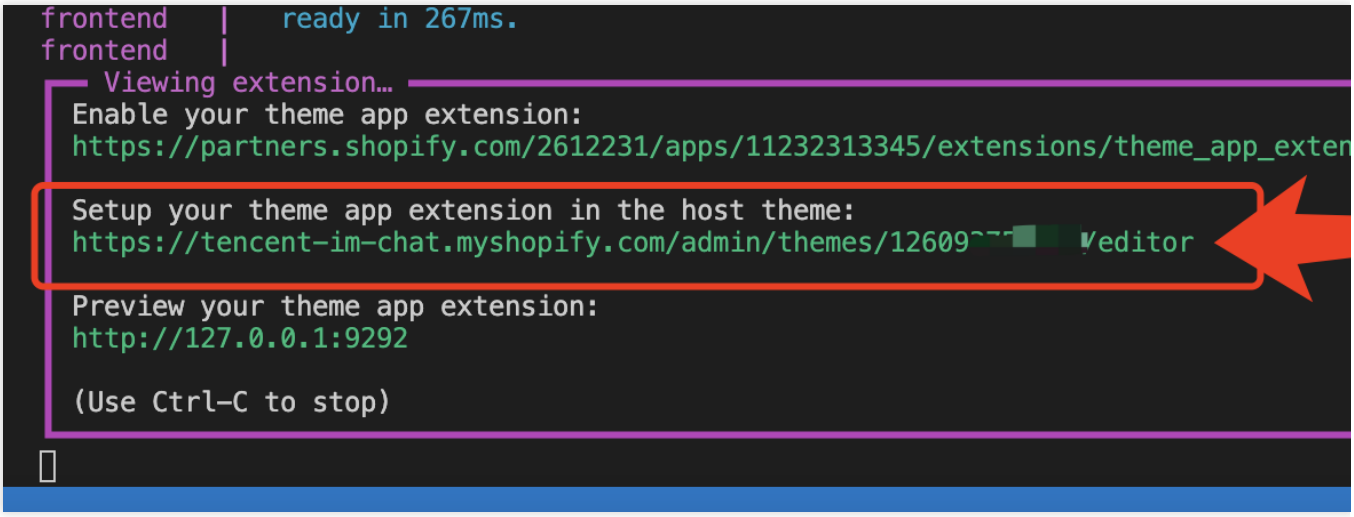
Step 8: Configure this extension

Run the following commands to start the extension:



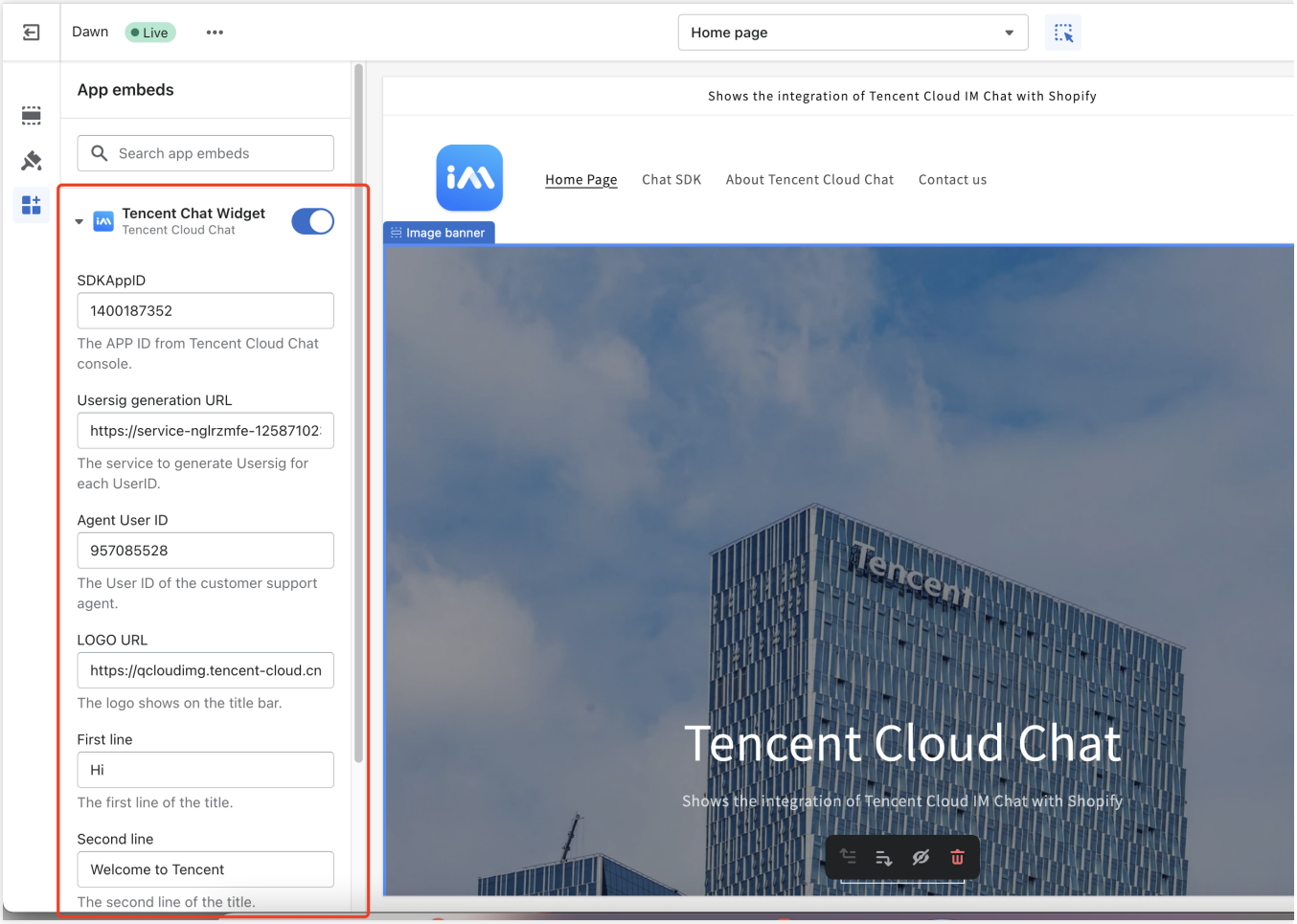
```
npm run dev
```

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.

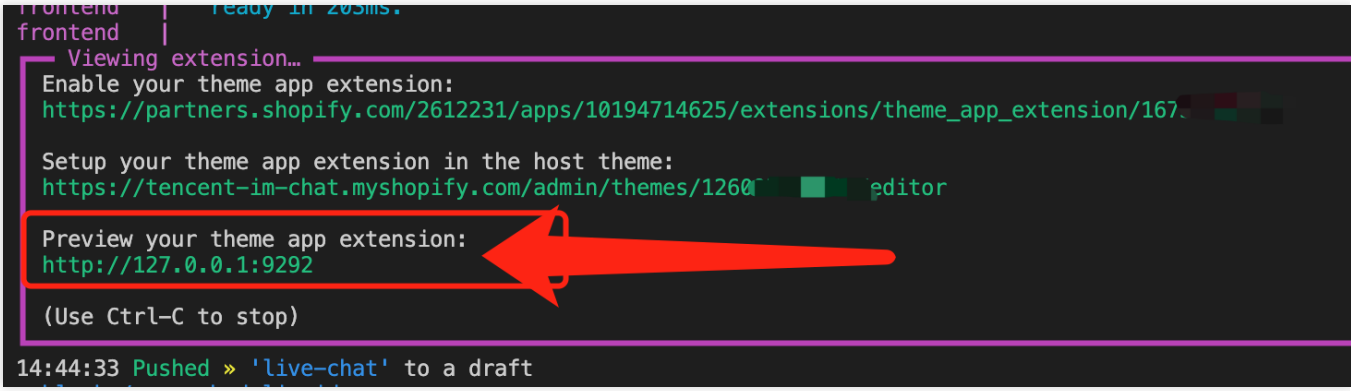


Field	Description	Required

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.

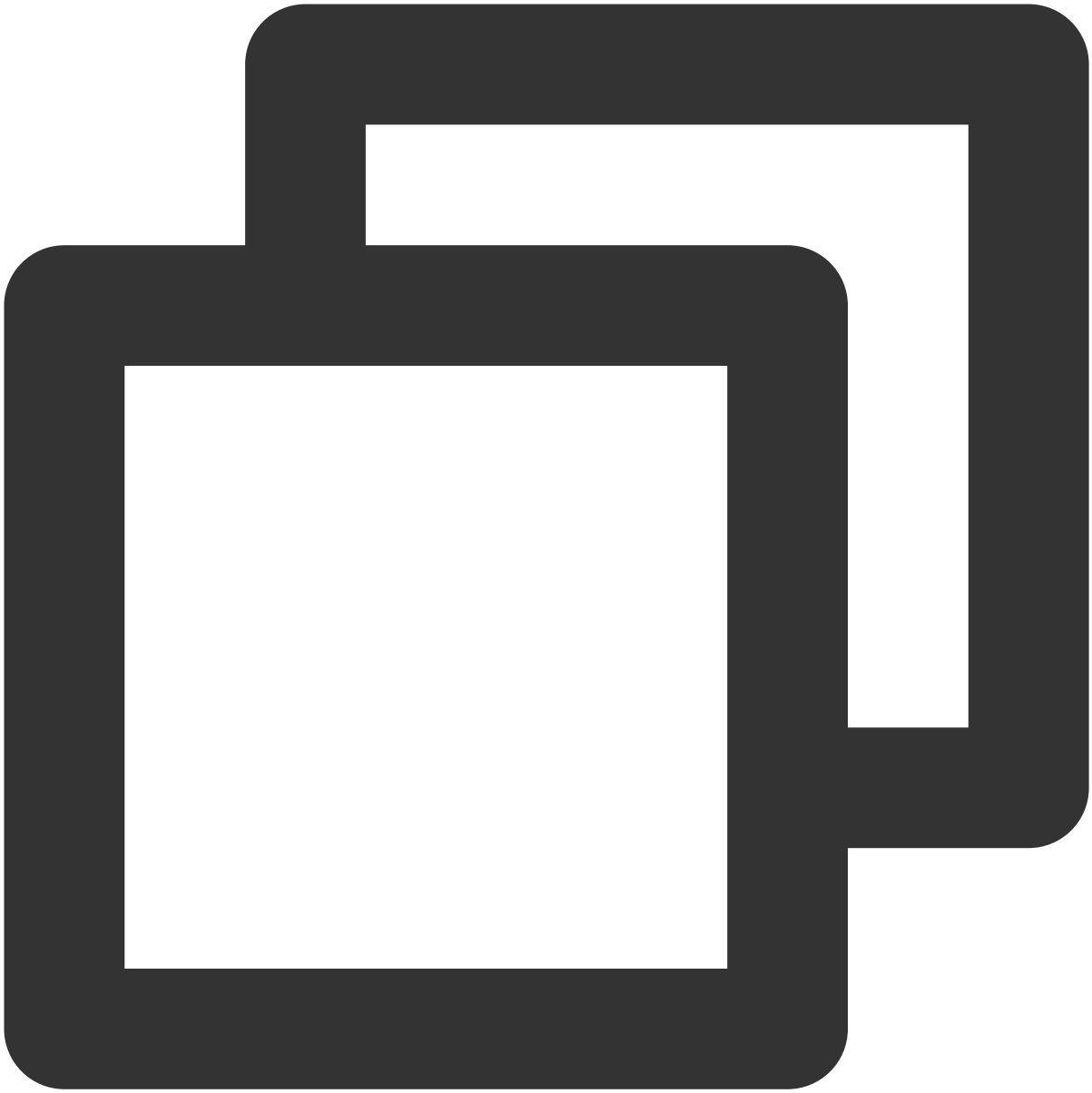


Make sure the chat widget works well. You can refer to our sample codes, if errors are encountered.

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:



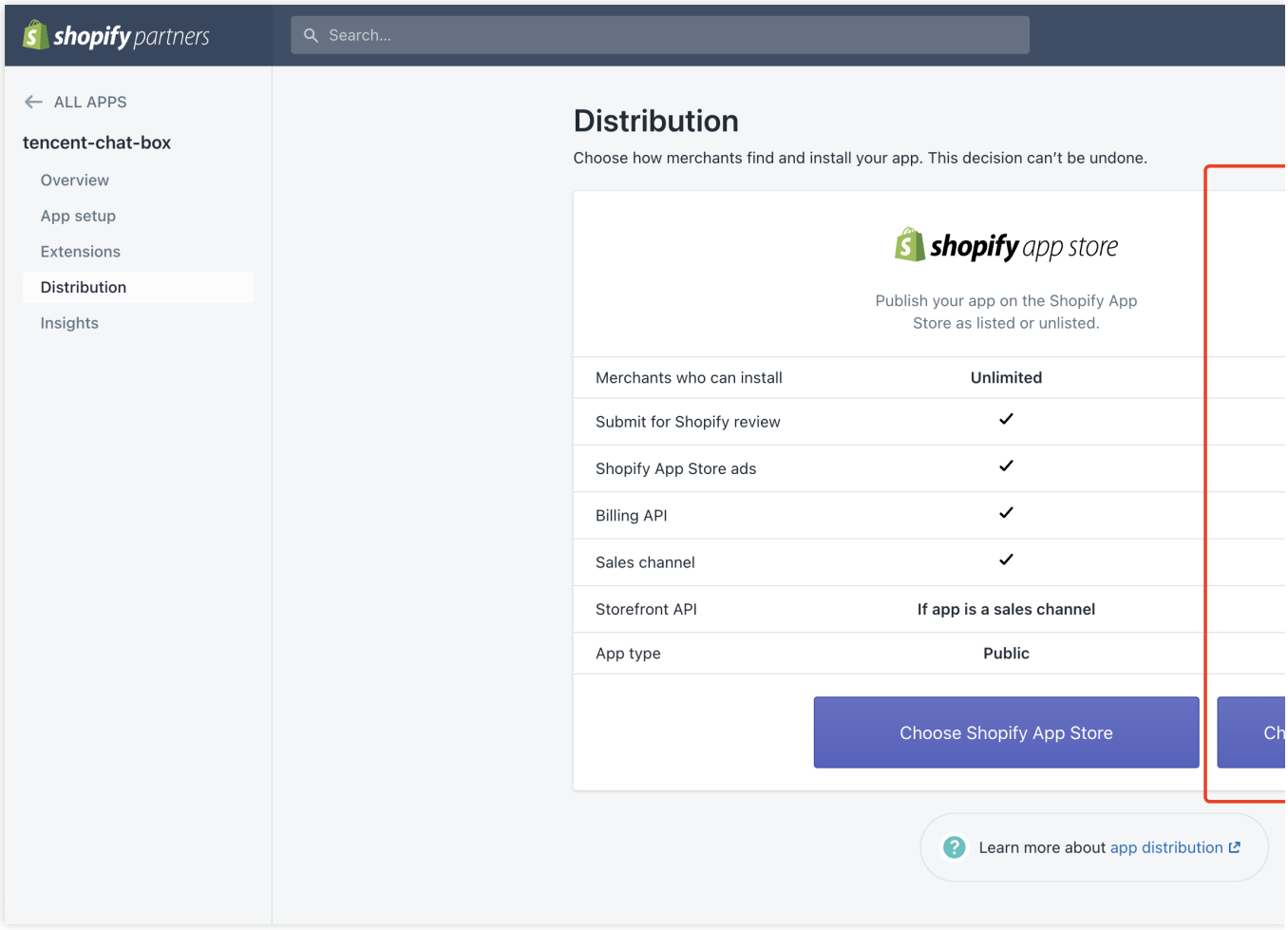
```
npm run deploy
```

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.

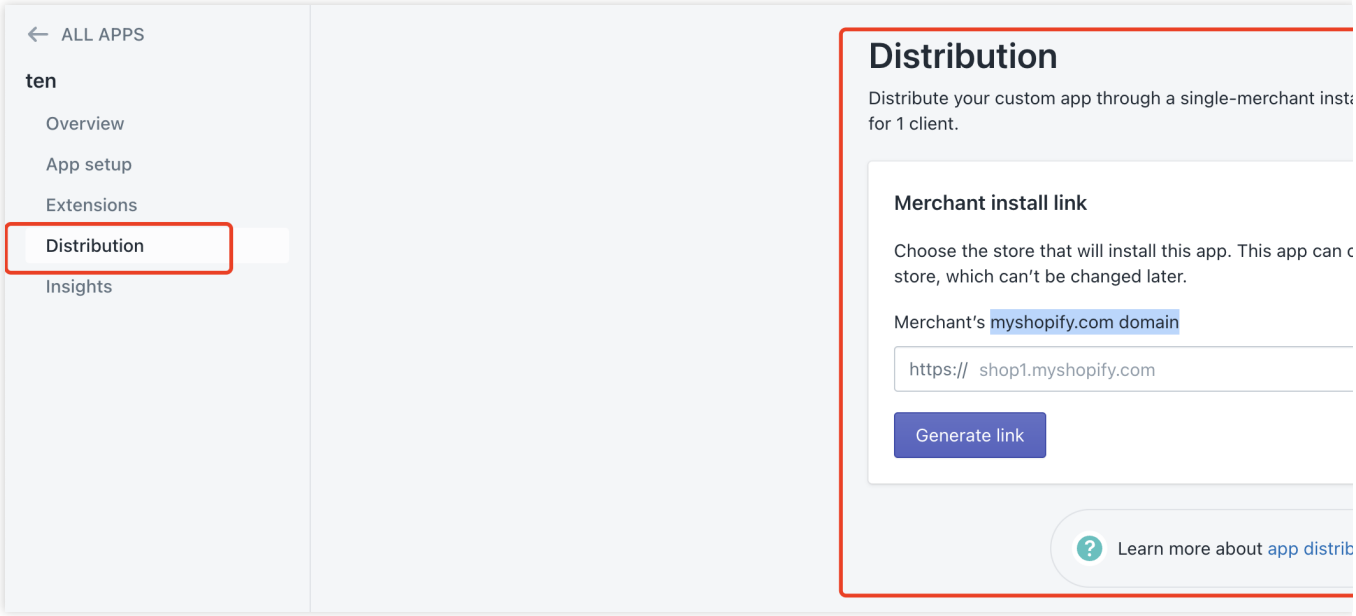
Step 10: Install this app to your online store

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

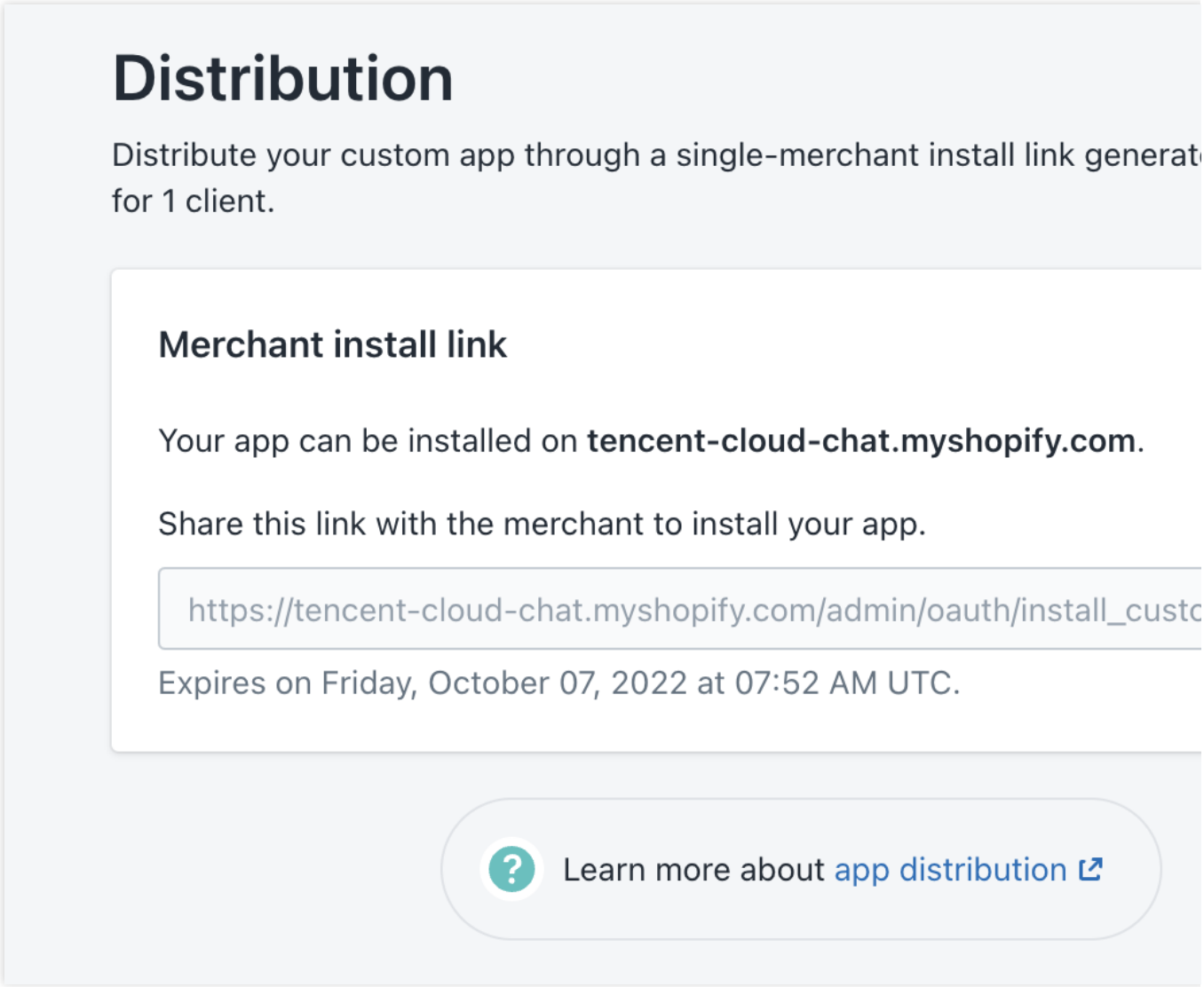
2. In the sidebar, click **Distribution**.
3. Select "Single-merchant install link" as the distribution method.



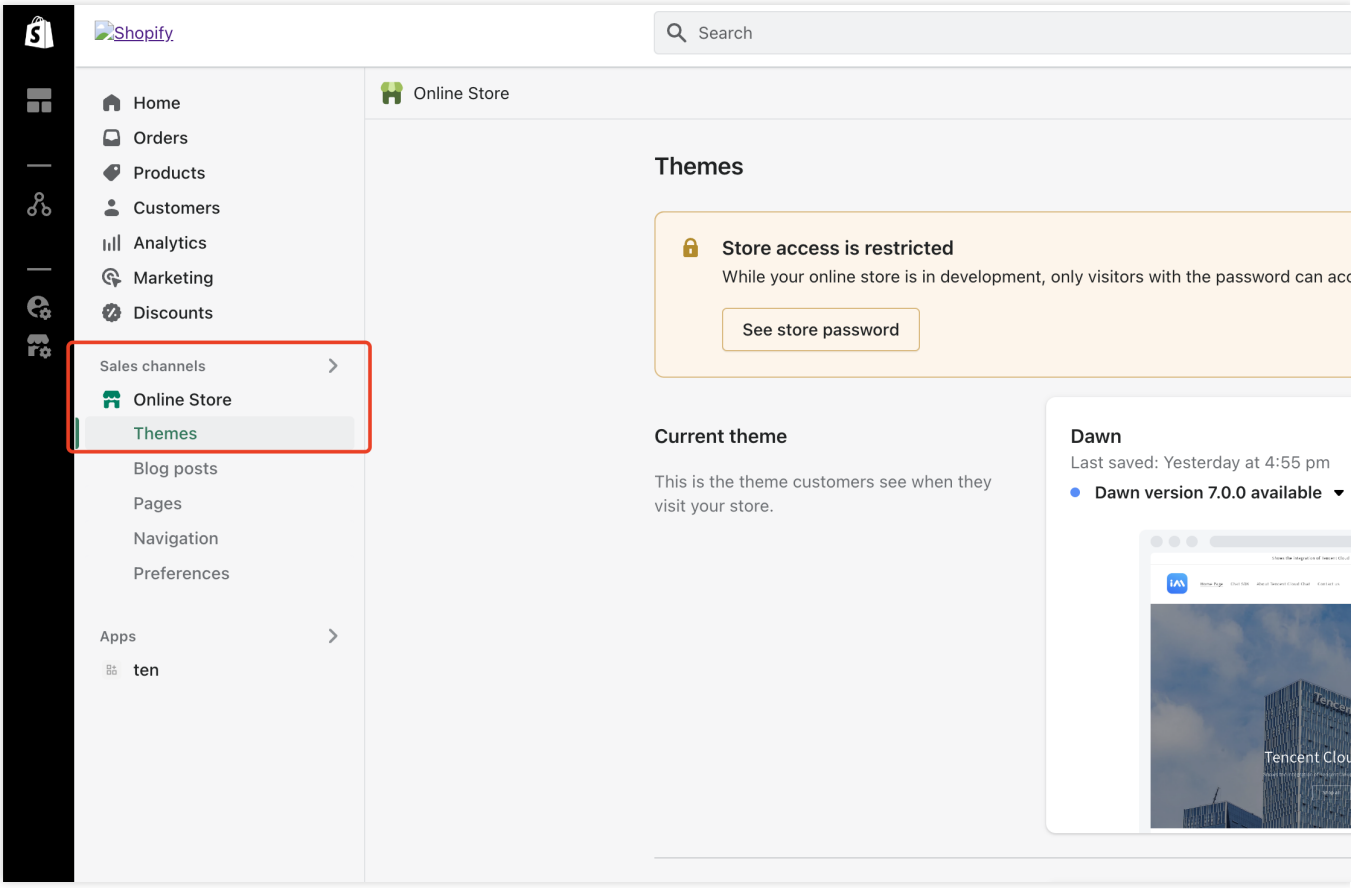
4. Generate the install link for your Shopify store, by entering your `myshopify.com` domain here.



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実装ガイド

最終更新日：：2024-04-11 17:29:57

Discordの紹介

Discordはコミュニティのために設計された無料のインターネットインスタント通信ソフトとデジタル配信プラットフォームで、主にゲーマー、教育関係者、友人、ビジネス関係者を対象としており、ユーザー同士がソフトのチャットチャンネルでメッセージ、画像、ムービー、音声でコミュニケーションを取ります。

Discordの概念

サーバー

Discordには、一般的な通信ソフトのグループとは別のグループチャットとして、サーバー（コミュニティのようなもの）と呼ばれるものがあり、サーバーの所有者はサーバーの中に自分だけのコミュニティを作ることができます。

チャネル

サーバーにチャンネルと呼ばれるチャットパイプを構築し、音声と文字に分けられます。音声チャンネルはゲームのライブ配信やチャットなどに利用されます。チャンネルでアイデンティティグループに様々な権限を設定することで、Discordコミュニティシステムをさらに多様化します。

サブチャンネル

ユーザーは特定のトピックについてサブチャンネル内で議論します。

準備作業

Tencent Cloud IMアプリの作成

このチュートリアルはTencent Cloud IMをベースにしているため、Tencent Cloud IM Consoleでアプリケーションを作成する必要があります。次の図を参照してください：
アプリケーションを作成したら、Tencent Cloud IM Consoleアプリケーション基本情報ページで、アプリケーションの基本情報を確認します。

関連する設定と機能について

Tencent Cloud IMを使用してDiscord関連機能を実装するには、Tencent Cloud IMに関連する基本的な概念と、このチュートリアルの後述する専門用語を事前に理解しておく必要があります。
SDKAppID：Tencent Cloud IMは各アプリケーションにSDKAppIDを割り当て、コンソールでアプリケーションを作成した後にアプリ詳細ページで確認できます。開発者は、Tencent Cloud IMクライアントのSDKを初期化し、ユーザーログイン用チケットを計算する際に使用できます。詳細については、UISDKなしの初期化およびログインのドキュメントをご参照ください。
キー：現在のアプリケーションキーはTencent Cloud IMコンソールのアプリケーション詳細ページに表示されます。SDKへのユーザーログイン用チケットの計算に使用されます。
ユーザーアカウント：Tencent Cloud IMへログインするユーザーはTencent Cloud IMのアカウントシステムに含まれなければなりません。ユーザーがクライアントSDKを使用して正常にログインすると、Tencent Cloud IMバックグラウンドで自動的にIMユーザーが作成されます。また、Tencent Cloud IMが提供するサーバー側APIを使ってユーザーをIMのユーザーシステムにインポートすることができます。
グループ：これまでのIMでは、さまざまな場面で必要に応じて、5種類のグループを提供しています。ユーザーはグループ内で発言すると、グループのメンバーはメッセージを受け取ることができます。
コールバックの設定：開発者は、IMが提供するクライアントSDKとサーバー側APIを積極的に統合するだけでなく、IMは特定のビジネスロジックの開始時に必要な情報を開発者サーバー側に自動的に戻します。開発者がTencent Cloud IMコンソールのコールバック設定モジュールで適切な設定を完了するだけで済みます。Tencent Cloud IMは豊富なコールバック設定を提供し、信頼性の高いコールバックを保証します。開発者はコールバックを通じて多くのカスタム要件を実現できます。
カスタムフィールド：デフォルトでは、Tencent Cloud IMが開発者に提供するフィールドはほとんどのニーズに対応しています。フィールドを拡張するユーザーに対し、Tencent Cloud IMは、次のようなカスタムフィールドを各モジュールに提供しています。
ユーザーカスタムフィールド
友達カスタムフィールド
グループカスタムフィールド
グループメンバーのカスタムフィールド
説明：
ユーザーはカスタムフィールドを使用するには、コンソールで設定し、SDKのAPIを用いて読み書きしてください。

クライアント側とサーバー側の統合SDK

Discord関連の機能を実装する際には、IMSDKを統合する必要があります。Tencent Cloud　IMは豊富で使いやすいSDKおよびサーバー側APIを提供しており、同じSDKAppIDでログインしたアプリケーションではすべての端末でメッセージの相互通信ができます。開発者は、ビジネスニーズのシナリオとテクノロジースタックに基づいて、適切なSDKを選択できます。

Discord機能の分析

上図に示すように、Discordの機能は主にサーバー、チャンネルおよびサブチャンネルに分けられており、サーバーとサーバーの間には王者栄耀サーバーや和平精英サーバーなど、コンテンツ上の違いがあります。サーバー内には文字チャンネル、音声チャンネル、ワールドチャンネルなど、さまざまなチャンネルを作れます。ユーザーのコミュニケーションは実際に各チャンネルで行われています。ユーザーはコミュニケーションしているコンテンツに対し他のアイデアを持っている場合、そのコンテンツについてサブチャンネルを作成してコミュニケーションを続けます。Discordのコアプレイはここで説明したとおりです。このチュートリアルでは、Tencent Cloud IMを使用して関連機能を実装する方法を1つずつ分析します。

説明：

デモコードについては、このチュートリアルでAndroid側（Java SDK）の例を示しています。他のSDKインターフェースの呼び出しについては、UIなしの統合方法をご参照ください。

サーバー

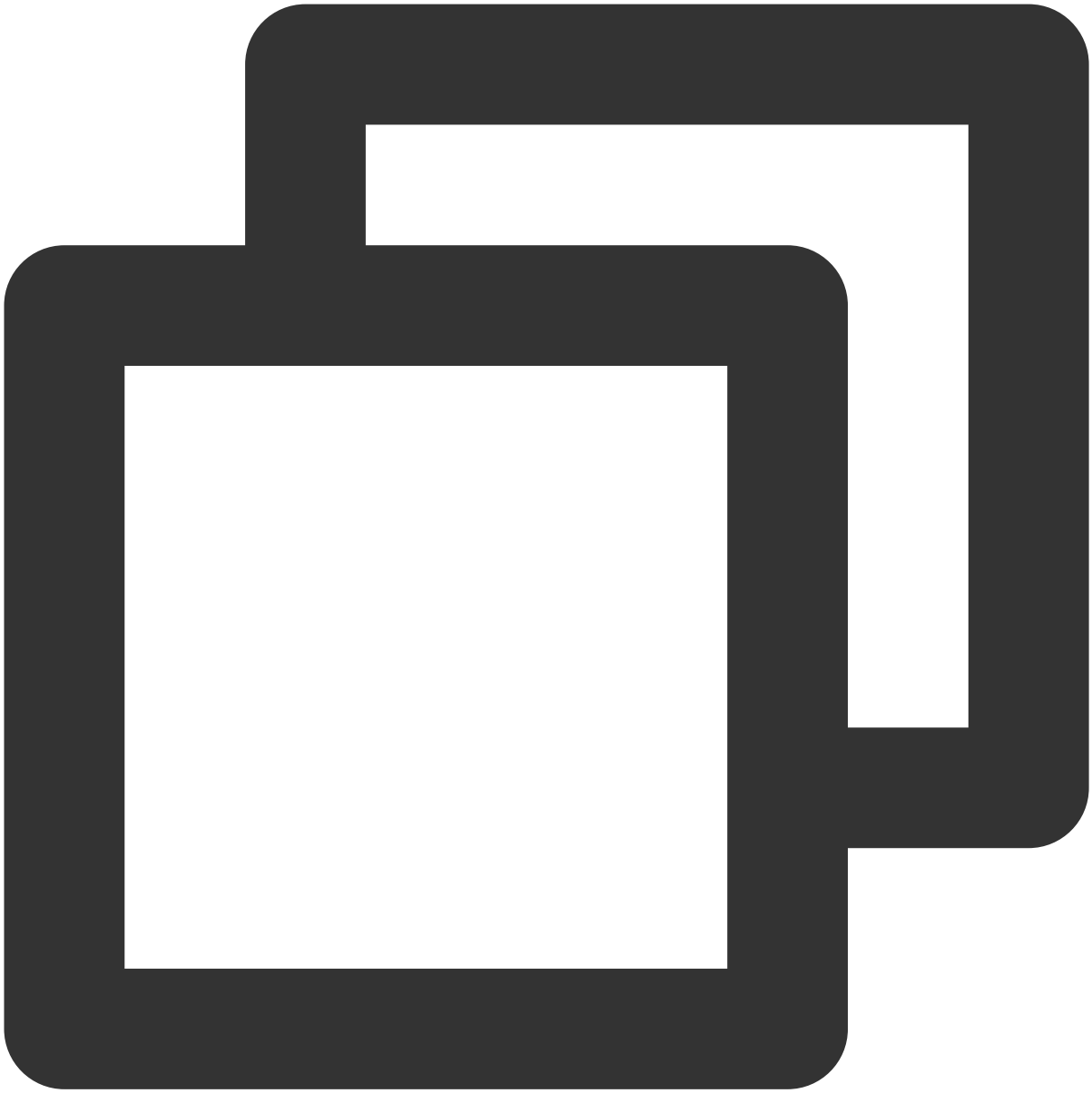
サーバーの作成

分析の結果、Discordのサーバーリストには次のような特徴があります：

1. サーバーの人数が非常に多い場合があります。
2. ユーザーは実際にサーバーでコミュニケーションをとることはなく、サーバー内のチャンネルやサブチャンネルでのみチャットを行います。
3. サーバー内のチャット履歴にはローミングが必要です。
4. 自由に入ることができます。
5. サーバーにチャンネルを作成できます。

IMは5種類のグループを提供しているが、Discordのサーバーの特徴を見ると、コミュニティ（Community）だけがサーバーの特徴に合致しており、Tencent Cloud IMコミュニティは作成後に自由に出入りでき、最大10万人の同時オンラインと履歴メッセージの保存が可能で、ユーザーがグループIDを検索してグループ参加を申請すると、管理者の承認なしにグループに入れます。

サーバー（グループ）は、Tencent Cloud IMが提供するcreateGroupインターフェースで作成されます。作成したグループのタイプはCommunityであり、サーバーにチャンネルを作成するにはsetSupportTopicをtrueに設定する必要があります。デモコードは次のとおりです：



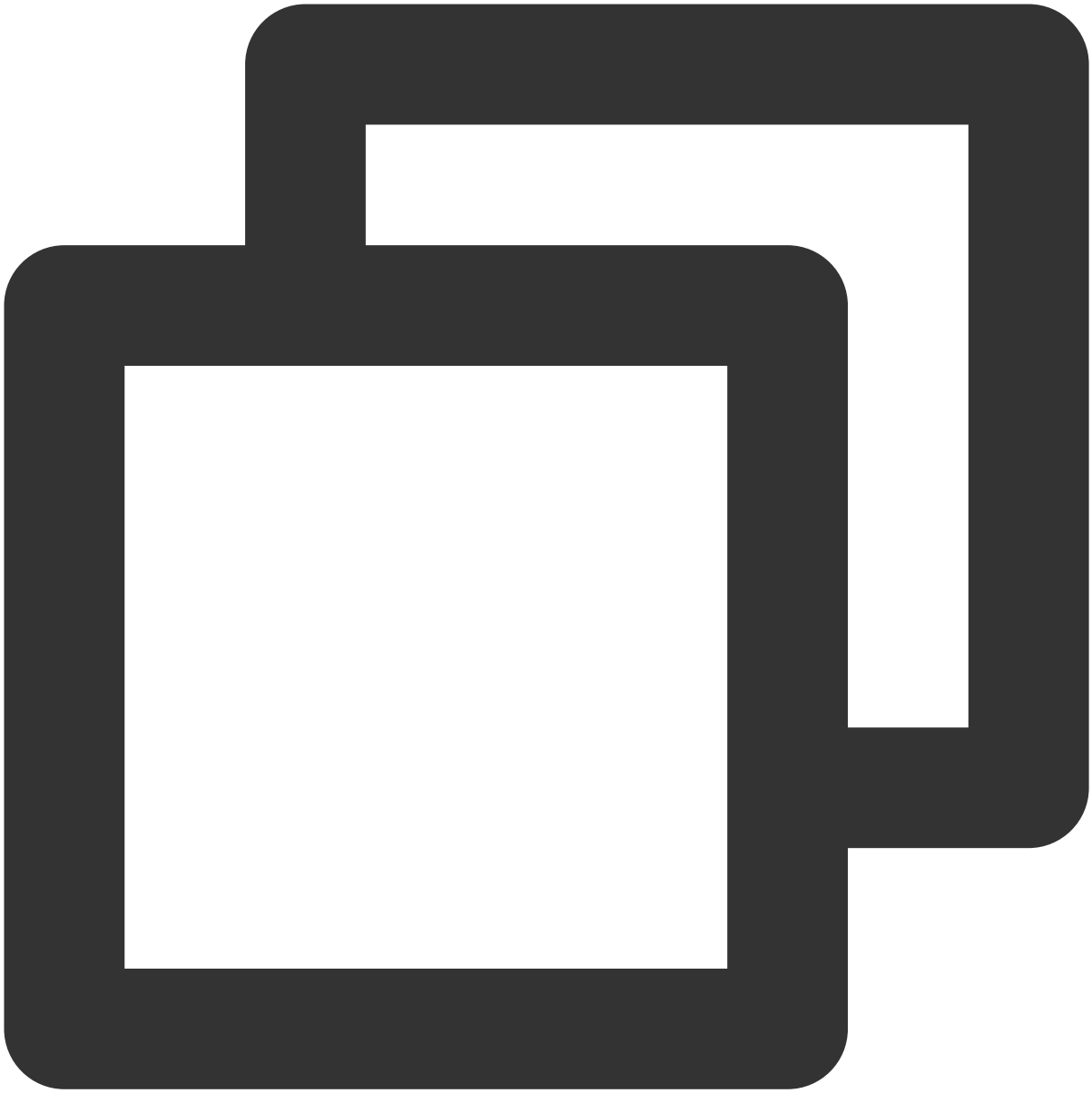
```
V2TIMGroupInfo  groupinfo = new V2TIMGroupInfo();
groupinfo.setGroupName("テストサーバー");
groupinfo.setSupportTopic(true);
/// 初期グループメンバー
List<V2TIMCreateGroupMemberInfo> memberList = new LinkedList<V2TIMCreateGroupMember
// サーバーのアバターなどのその他の設定
V2TIMManager.getGroupManager().createGroup(groupinfo, memberList, new V2TIMValueCal
            @Override
            public void onError(int i, String s) {
                //  作成失敗
            }
```

```
            @Override
            public void onSuccess(String s) {
                // 正常に作成されるとサーバーIDを返します
            }
});
```

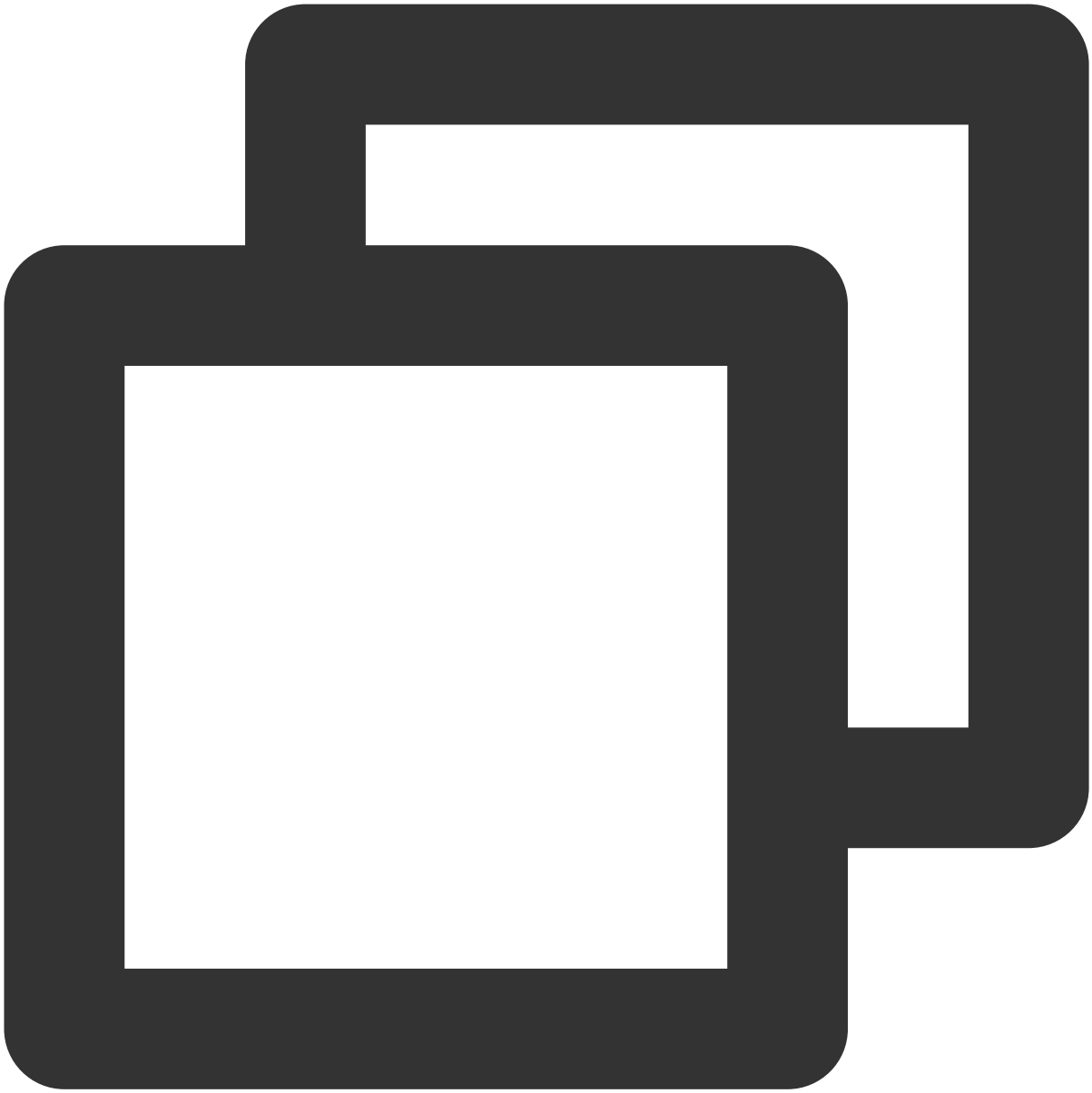
インターフェースを正常に呼び出した後、onSuccessのコールバックでサーバーIDが返され、サーバーでのチャンネル作成機能で使用されます。

ご注意：

開発者は、IMが提供するサーバー側APIを使って、サーバー側でサーバーを作成できます。主なパラメータは次のとおりです：



```
{
    "Type": "Community",      // グループタイプ（必須）
    "Name"："TestCommunityGroup",  //  コミュニティ名(必須)
    "SupportTopic"：1                    //  トピックオプションへの対応。対応の場合は1、非対
}
```



```
###### サーバーリスト

Discordの左端にはサーバーリスト機能があり、ユーザーが参加しているサーバーのリストを示しています。こ

```java
V2TIMManager.getGroupManager().getJoinedCommunityList(new V2TIMValueCallback<List<V
            @Override
            public void onSuccess(List<V2TIMGroupInfo> v2TIMGroupInfos) {
                // サーバーリストの取得に成功する場合は、返されたList<V2TIMGroupInfo>はサー
            }
```
```

```
            @Override
            public void onError(int i, String s) {
              //  サーバーリストの取得に失敗する場合は、
            }
});
```

返されたV2TIMGroupInfoリストはサーバーの基本情報を示します。ただし、基本情報には、サーバーの未読数やカスタマイズ状態などに関する情報はありません。したがって、Discordのような効果を実現するには、IMが提供する別のAPIを使用する必要があります。サーバーリストの未読数は、後述のセクションで説明します。

サーバー分類

サーバーを作成すると、それぞれのサーバーがデフォルトの分類を作成し、作成後に新しい分類を作成することもできます。サーバーはTencent Cloud IMで基本的にコミュニティであるため、コミュニティのカスタムフィールドを設定することで実現できます。グループのカスタムフィールドを使用するには、次の2つのステップがあります：

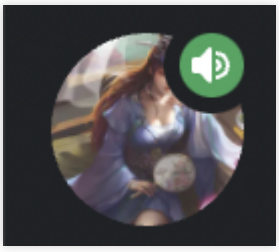
1. コンソールでカスタムフィールドkeyを有効にします。
2. クライアントSDK/サーバー側APIで読み書きを実行します。

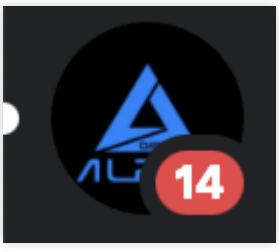
クラスタカスタムフィールドのサービス側APIおよびクライアント側SDKを設定します。

ご注意：

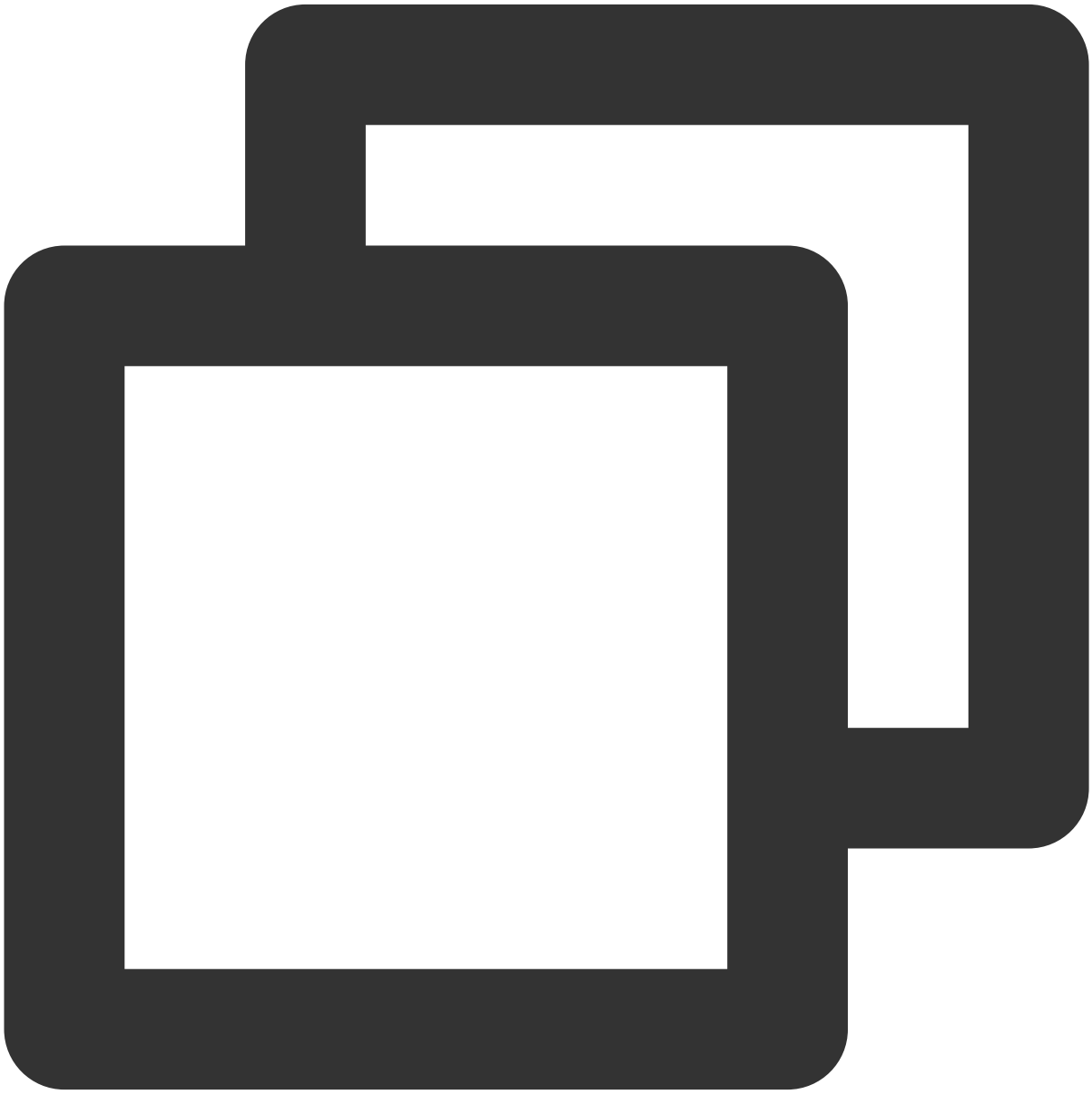
コミュニティはUltimate機能であり、コミュニティのカスタムフィールドを設定するには、まずUltimateを購入する必要があります。

サーバーメッセージ未読数&カスタムステータスの表示





前のセクションでは、参加済みサーバーリスト取得APIでは、未読数やサーバーの状態などの情報が返されないことを説明しました。注意すべき点として、このデータを取得するだけでなく、このデータの変化を監視してクライアントUIを更新する必要があります。サーバーはIMコミュニティを使用して実装され、IM内のコミュニティでセッションを生成しないため、すべてのパブリックチャンネルのセッションとプライベートチャンネルのセッションの合計を集計する必要があります。パブリックチャンネルの未読数はV2TIMTopicInfoのgetUnreadCountによって取得され、プライベートチャンネルはworkグループによって実現されるので、プライベートチャンネルの未読数はgetConversationによって取得されます。



```
// パブリックチャンネル
List<String> conversationIDList = new LinkedList();
```

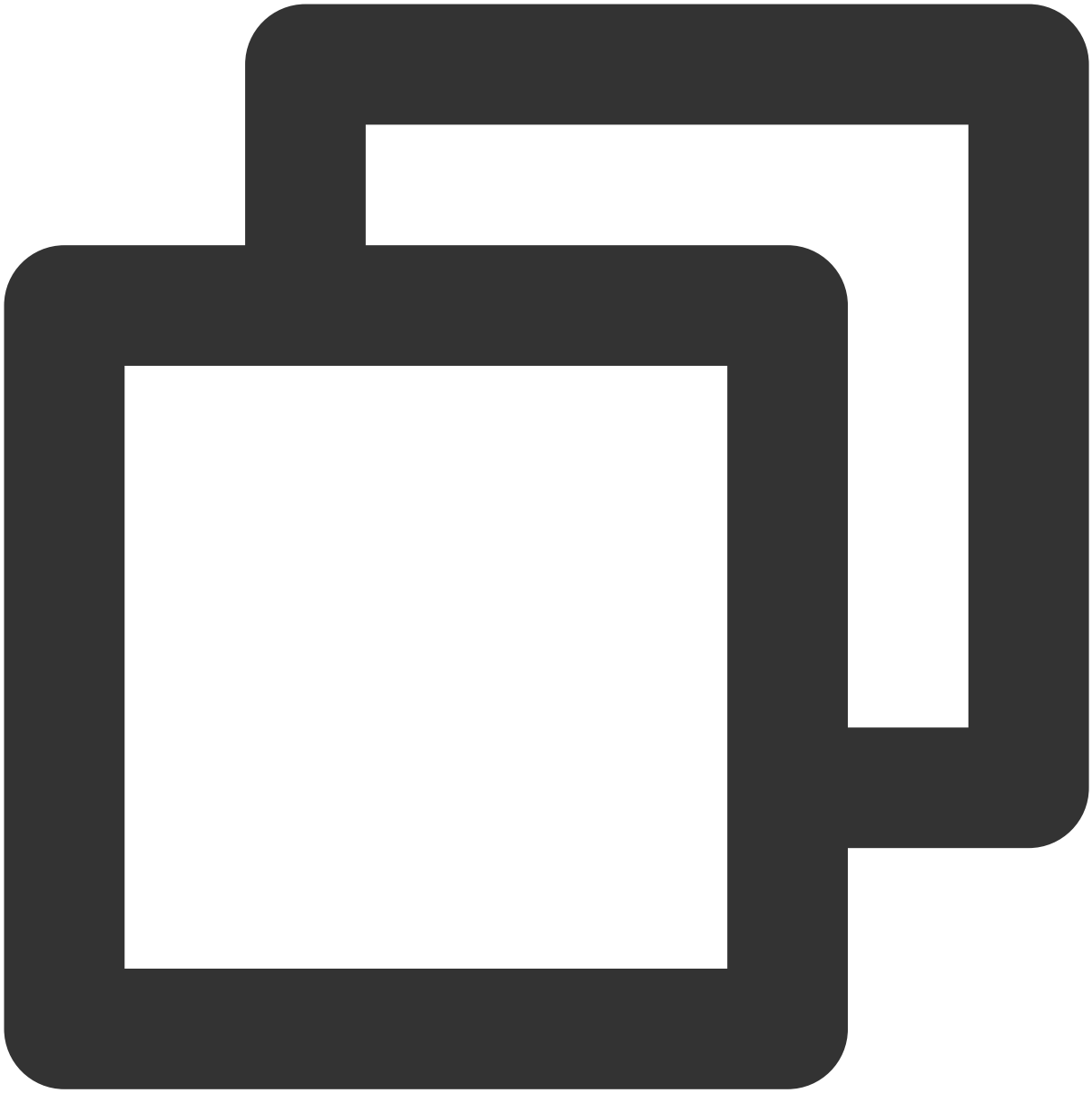
```
conversationIDList.add("GROUP_$GROUPID");
V2TIMManager.getConversationManager().getConversationList(conversationIDList, new V
            @Override
            public void onError(int i, String s) {
                // サーバーに対応するセッション情報の取得に失敗しました
            }

            @Override
            public void onSuccess(V2TIMConversationList List<V2TIMConversation>) {
               // サーバーに対応するセッション情報の取得に成功しました
            }
});
// プライベートチャンネル
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList, new V2TIMValu
                @Override
                public void onSuccess(List<V2TIMTopicInfoResult> v2TIMTopicInfoResu

                }

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

サーバーのカスタム状態機能は、サーバーセッションのカスタムデータを設定することで実現できます。APIは setConversationCustomDataです。



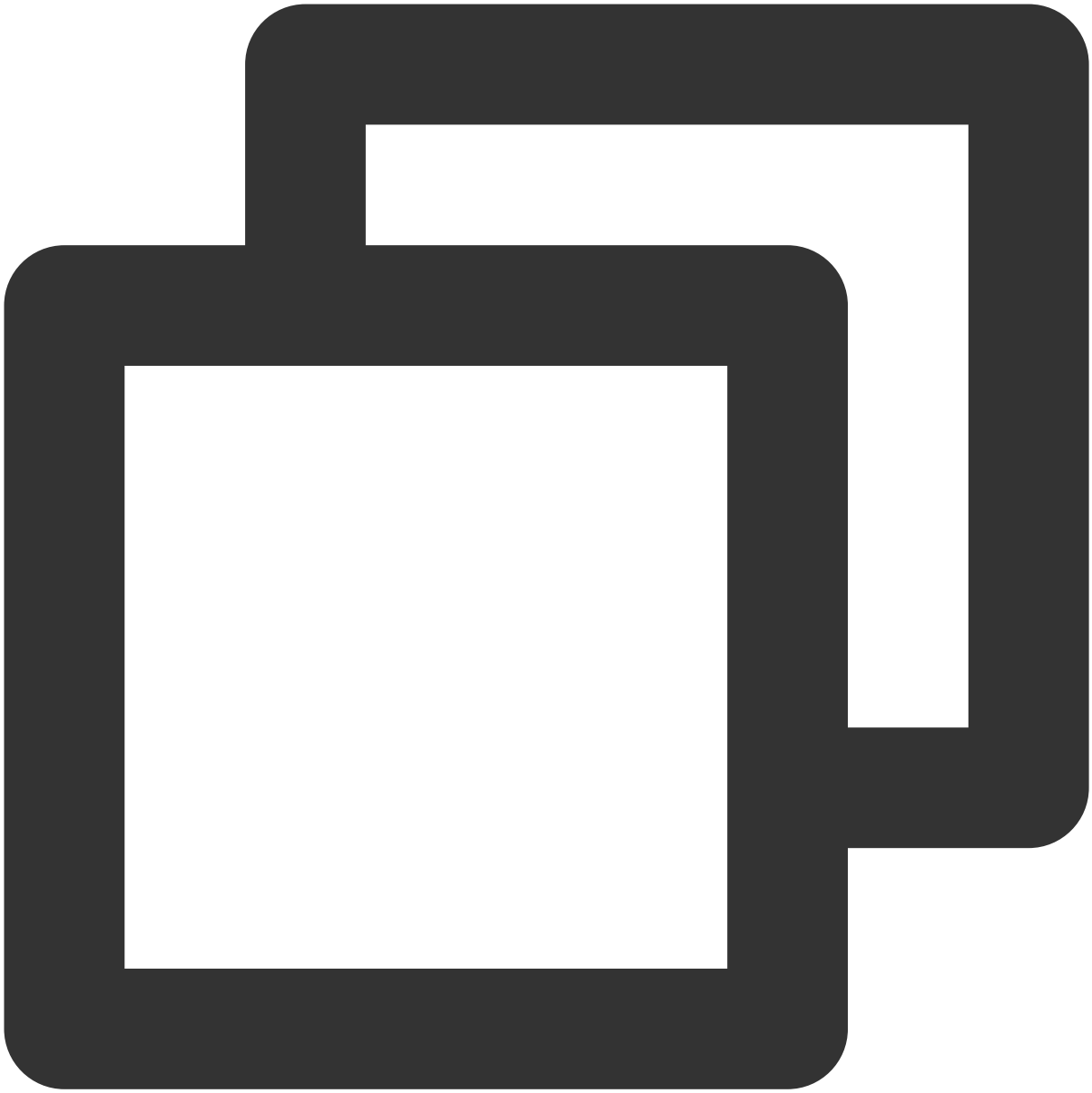
```
List<String> conversationIDList = new LinkedList();
String customData = "通話中"
V2TIMManager.getConversationManager().setConversationCustomData(conversationIDList,
            @Override
            public void onSuccess(List<V2TIMConversationOperationResult> v2TIMConve
                // グループセッションのカスタムデータの設定に成功しました
            }

            @Override
            public void onError(int i, String s) {
                // グループセッションのカスタムデータの設定に失敗しました
```

```
            }
});
```

サーバーセッションの関連データが変更された場合、クライアントはUIの更新を表示しようとする場合、サーバーセッションの変更をリスニングするために、IMはイベントリスニング関数addConversationListenerを提供し、以下の情報が変更された場合にこのコールバック関数がトリガされます。

1. サーバーメッセージの追加・削除・変更
2. サーバーメッセージ未読数の変更
3. サーバーのカスタマイズ情報の変更
4. サーバートップ表示
5. サーバーのメッセージ受信設定の変更
6. サーバータグの変更
7. サーバーグループの変更
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);
```

以上から、getJoinedCommunityList、getConversationListインターフェースおよびaddConversationListenerコールバックにより、サーバーリストを表示するDiscord機能を実現できます。

チャネル

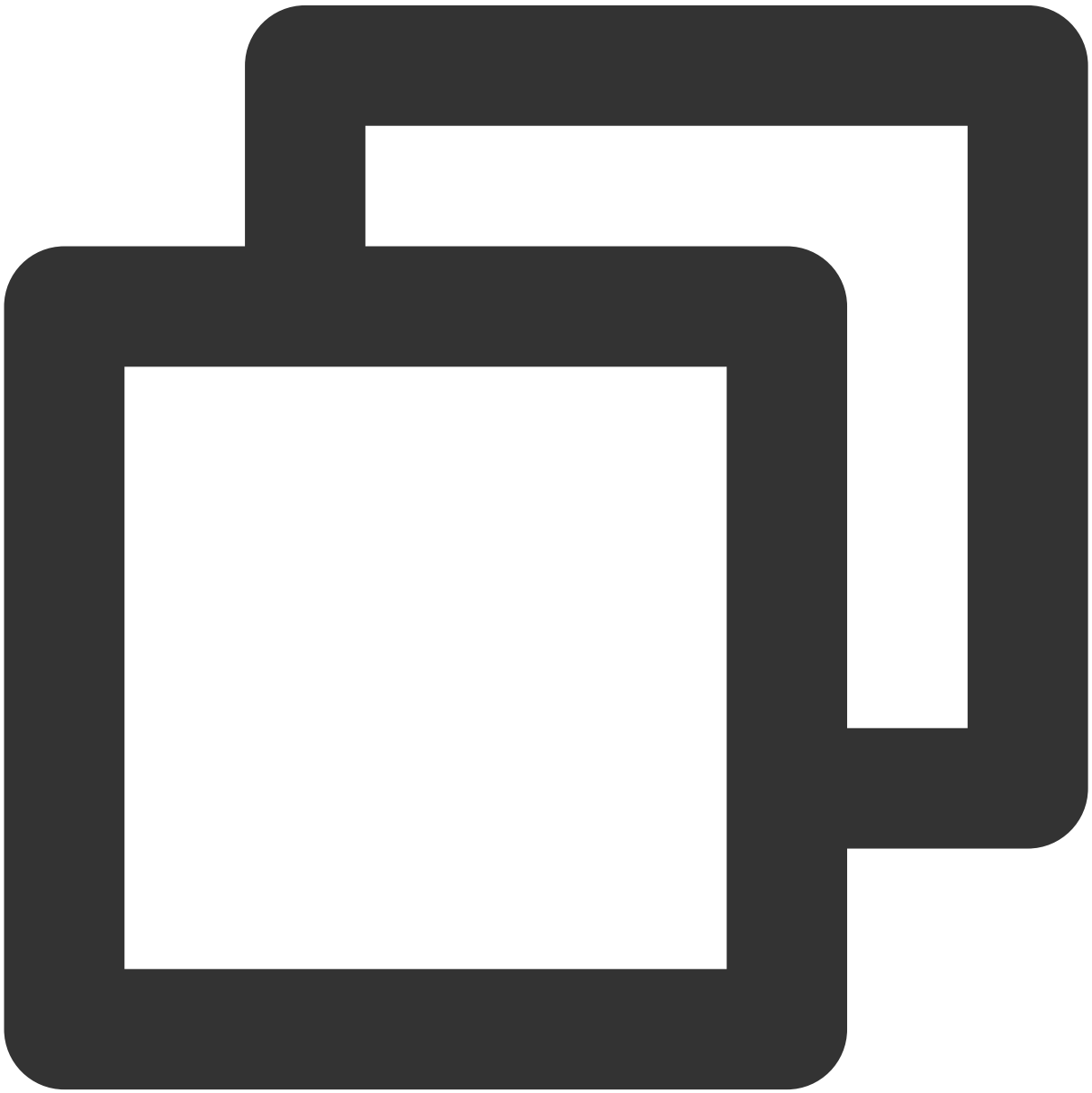
サーバー内では複数のチャンネルを作成できます。図に示すように、このサーバーの下に4つのチャンネルが作成され、4つのチャンネルが2つのカテゴリに配置されています。
チャンネルの場合、ユーザーは別のユーザを招待することや、チャンネルの基本的な設定を行うことができます。ユーザーのチャットの多くはチャンネル内で行われるため、チャンネルの能力がDiscordでは最も重要となります。Tencent Cloud IMでは、つまりトピックの能力に相当します。IMコミュニティのコミュニティ内でトピックを新規作成する機能をサポートします。

デフォルトチャンネル

Discordはサーバーを作成する際に、デフォルトで4つのチャンネルを作成しますが、Tencent Cloud IMを使用しても、次のようなプロセスで同様な機能を実現できます。

1. グループ作成後コールバックを使用して、サーバーが正常に作成されたことをビジネスサーバー側に通知します。
2. 他のグループに関連する業務に影響を及ぼさないように、ビジネス側が作成したコミュニティを確認します。
3. サーバープロパティに基づいて、サーバー側でトピックを作成します。

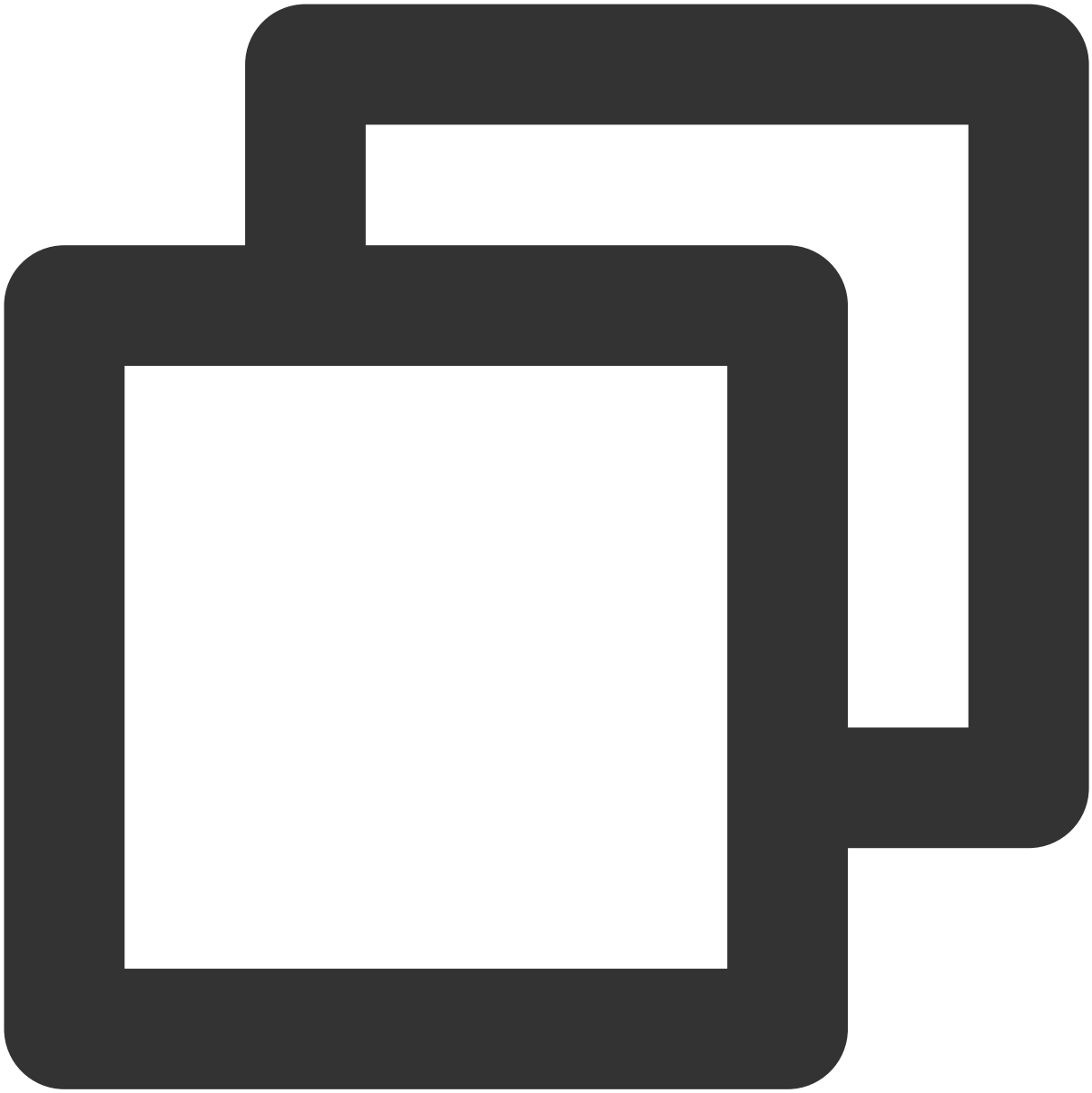
サーバー側でトピックを作成するとき主なパラメータは次のとおりです。



```
{
    "GroupId"：@TGS#_@TGS#cQVLVHIM62CJ//トピックが属するグループID(必須)
    "TopicId"："@TGS#_@TGS#cQVLVHIM62CJ@TOPIC#_TestTopic",//ユーザー定義のトピックID(オ
    "TopicName"："TestTopic",//トピックの名前(必須)
    "From_Account"："1400187352",//トピックを作成したメンバー
    "CustomString"："This is a custom string",//カスタム文字列
    "FaceUrl"："http://this.is.face.url",//トピックのアバターURL(オプション)
    "Notification"："This is topic Notification",//トピックの掲示（オプション入力）
    "Introduction"："This is topic Introduction"//トピックの概要（オプション入力）
}
```

チャネル作成

Discordクライアントでは、ユーザーが次のようなチャンネル分類でチャンネルを作成できます。
以上のように、チャンネルを作成することはIMのコミュニティでトピックを作成することに相当します。また、作成時には、トピックを分類し、トピックの基本情報を設定することができます。
関連する機能は、createTopicInCommunityを使用して実装されます。



```
String groupID="サーバーID"
V2TIMTopicInfo info = new V2TIMTopicInfo();
info.setCustomString("{'categray'：'game','type'：'text'}")//チャンネル分類とタイプを設定
// ここでV2TIMTopicInfoの具体的な情報を設定できます
V2TIMManager.getGroupManager().createTopicInCommunity(groupID, info, new V2TIMValue
            @Override
            public void onSuccess(String s) {
                // チャンネル作成成功
            }

            @Override
```

```
            public void onError(int i, String s) {
                //  チャンネル作成失敗
            }
});
```

チャンネル作成時には、V2TIMTopicInfoのメンバーメソッドを呼び出すことでチャンネルの情報、チャンネル分類およびチャンネルタイプを設定できます。setCustomStringで設定されます。
チャンネルを作成する際には、通常のチャンネルとは異なるプライベートチャンネルであるか否かを設定できます。
1. ユーザーはサーバーに参加した後、プライベートチャンネルに参加しません。
2. プライベートチャンネルへの参加は、サーバー管理者の招待が必要です。
そのため、workグループを使ってプライベートチャンネルの機能を実現しますが、サーバーがどのプライベートチャンネルをバインドしているのかという情報をビジネス側が保存する必要があります。

チャンネルタイプ

Discordはチャンネル作成時に、音声チャンネルと文字チャンネルのいずれかを選択できます。文字チャンネルは通常の文字、顔文字、画像のチャットであり、音声チャンネルは音声ビデオチャットです。注意すべきこととして、同じユーザーが同じ時間に1つの音声チャンネル内に入ることです。ユーザーが新しい音声チャンネルに参加するには、現在参加している音声チャンネルを終了する必要があります。
次のように4つの注意点があります：
1. チャンネルの作成時にチャンネルタイプを設定します。チャンネルタイプの設定方法については、「チャンネルの作成」で説明しました。
2. ユーザーが音声チャンネルに参加する際には、すでに他の音声チャンネルに入っているかどうかを判断しなければなりません。
3. 音声チャンネルはグローバルです。
4. Tencent Cloud IMは、ユーザーが音声チャンネルにいるかどうか、どの音声チャンネルにいるかのAPIを一時的に提供していないため、この部分のデータは事業側で管理されます。
第4の点について、開発者は、Tencent Cloud IMが提供するグループ参加後のコールバックおよびグループ退出後のコールバックを使用して、ユーザーが音声チャンネルにいるかどうかの状態を判断し、その状態をビジネス側ストレージ内に保存するなどのメンテナンスを行います。注意すべきこととして、Tencent Cloud IMのコールバックには遅延が存在する可能性があり、つまり、ユーザーが音声チャンネルを退出した後も、短時間は他の音声チャンネルに参加できない可能性があります。このため、ユーザーが音声チャンネルを参加・退出する状態をリアルタイムで業務サーバー側に報告するように、クライアントに報告する方法でも対応します。

ユーザーをチャンネルに招待

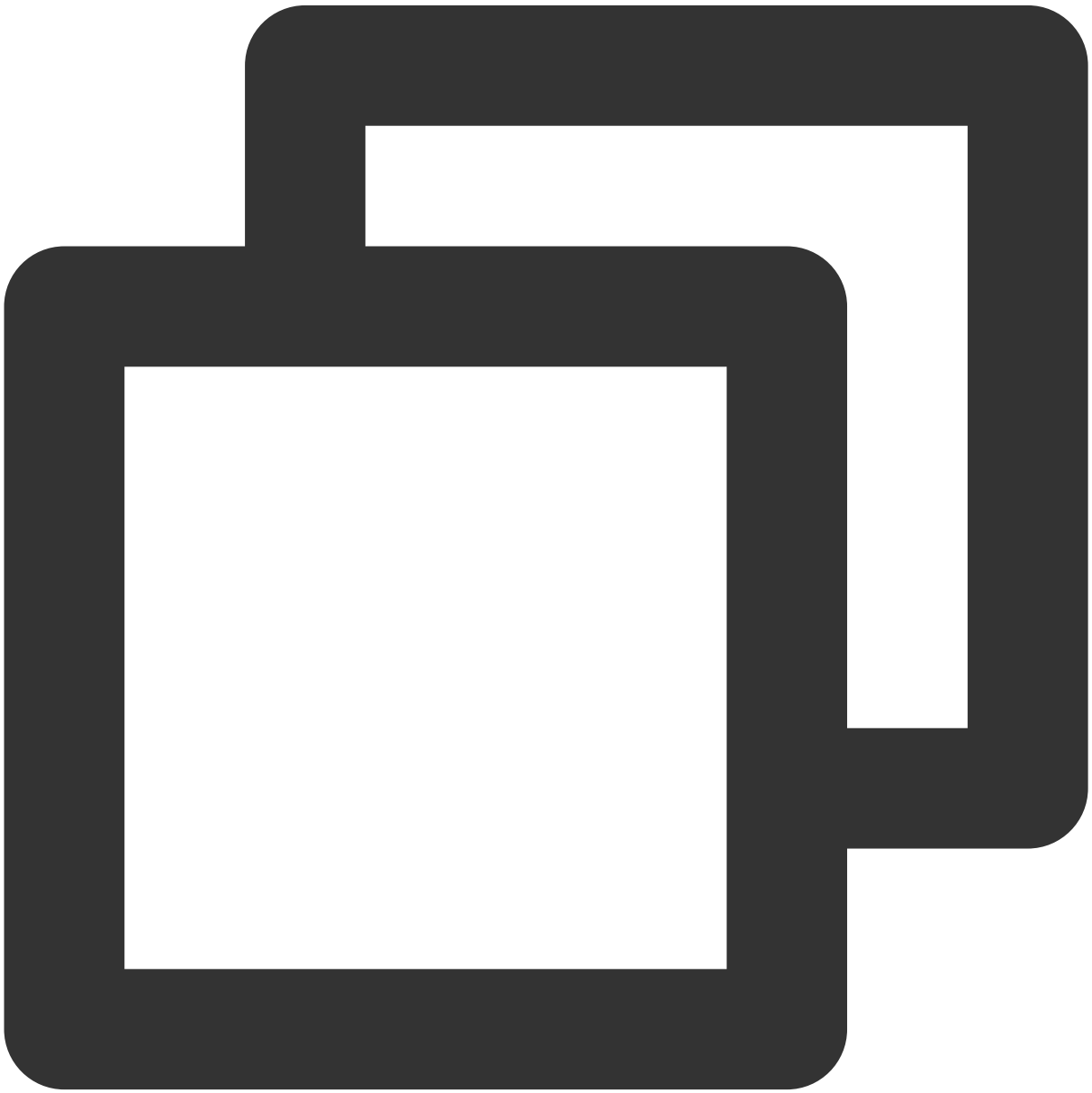
ユーザーがチャンネルに入る方法は次の3つがあります：
1. ユーザをサーバーに招待するとき、サーバーに入ったユーザーはすべての公開されたチャンネルに入ります。
2. ユーザーは検索したサーバーに入ります。
3. サーバー管理者からの招待を受けてプライベートチャンネルに入ります。

コミュニティの特性から、ユーザーがワールドチャンネルに入るには、サーバーに参加すればよいことが分かります。

1. グループIDを検索して参加できます。
2. 招待者が入場できます。
3. 特に承認を得る必要なく、申請すれば入れます。

チャンネル設定

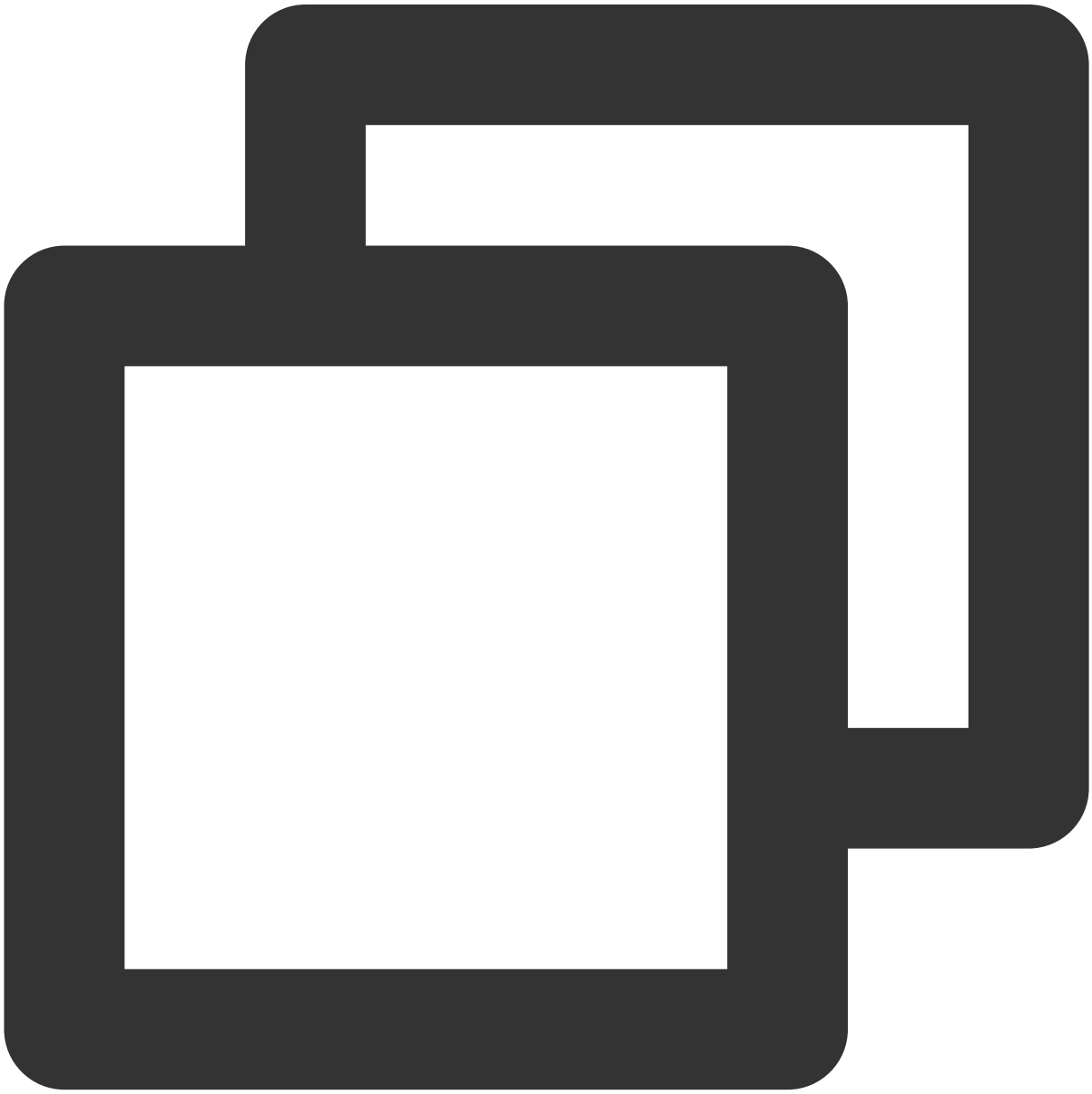
チャンネルのミュート設定や通知設定は、APIsetAllMuteおよびsetGroupReceiveMessageOptを使用します。



```
// チャネル基本情報の設定
```

```
V2TIMManager.getGroupManager().setTopicInfo(topicInfo, new V2TIMCallback() {
    @Override
    public void onSuccess() {
        //  設定成功
    }

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



```
// チャンネルがメッセージを受け取る方法を設定します
String groupID = "topicid"
int opt = 0;
V2TIMManager.getMessageManager().setGroupReceiveMessageOpt(groupID, opt, new V2TIMC
    @Override
    public void onSuccess() {

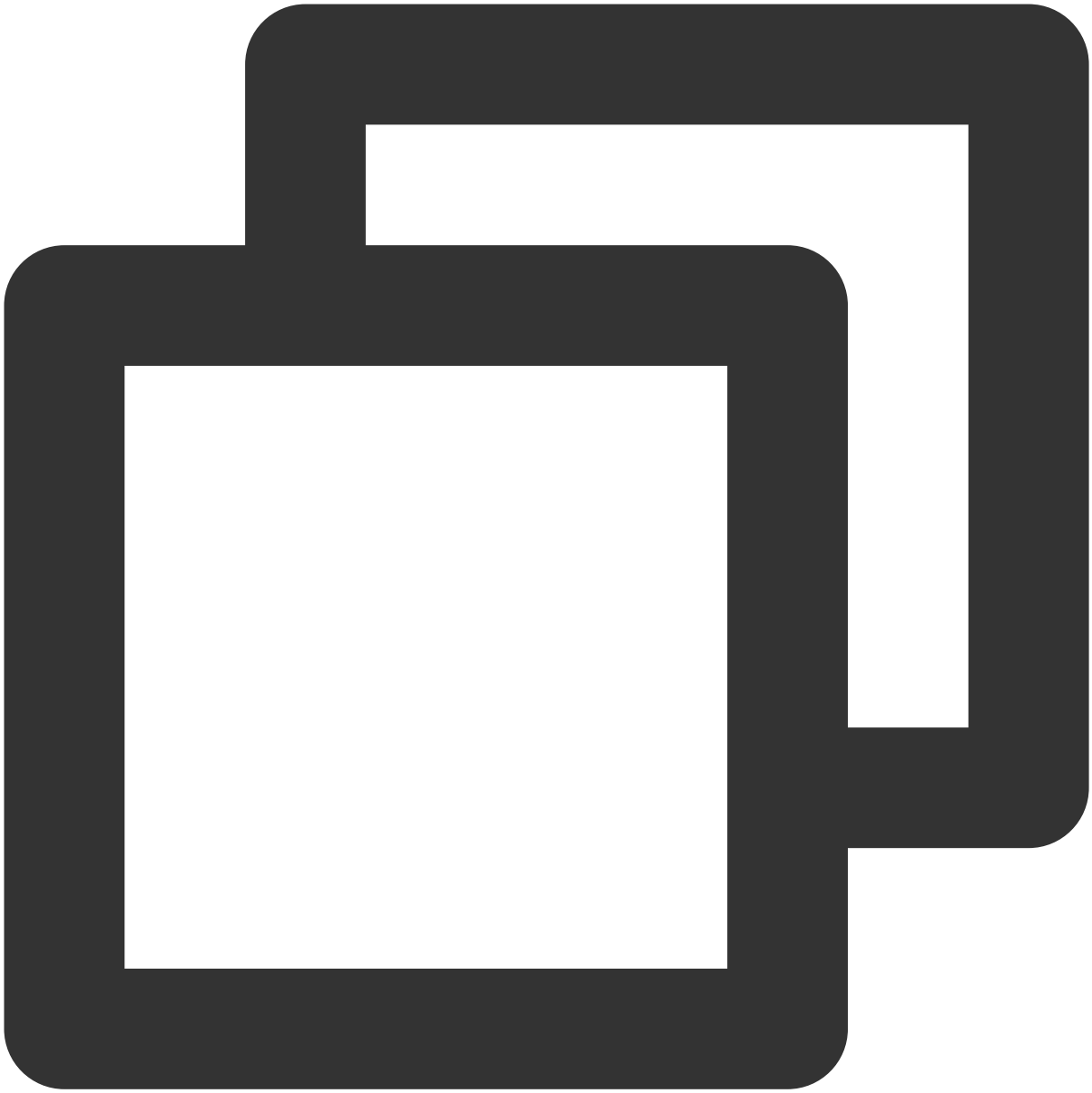
    }

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

    }
});
```

チャネルメッセージリスト

チャンネルの履歴メッセージレコードは、getHistoryMessageListを使用して取得できます。



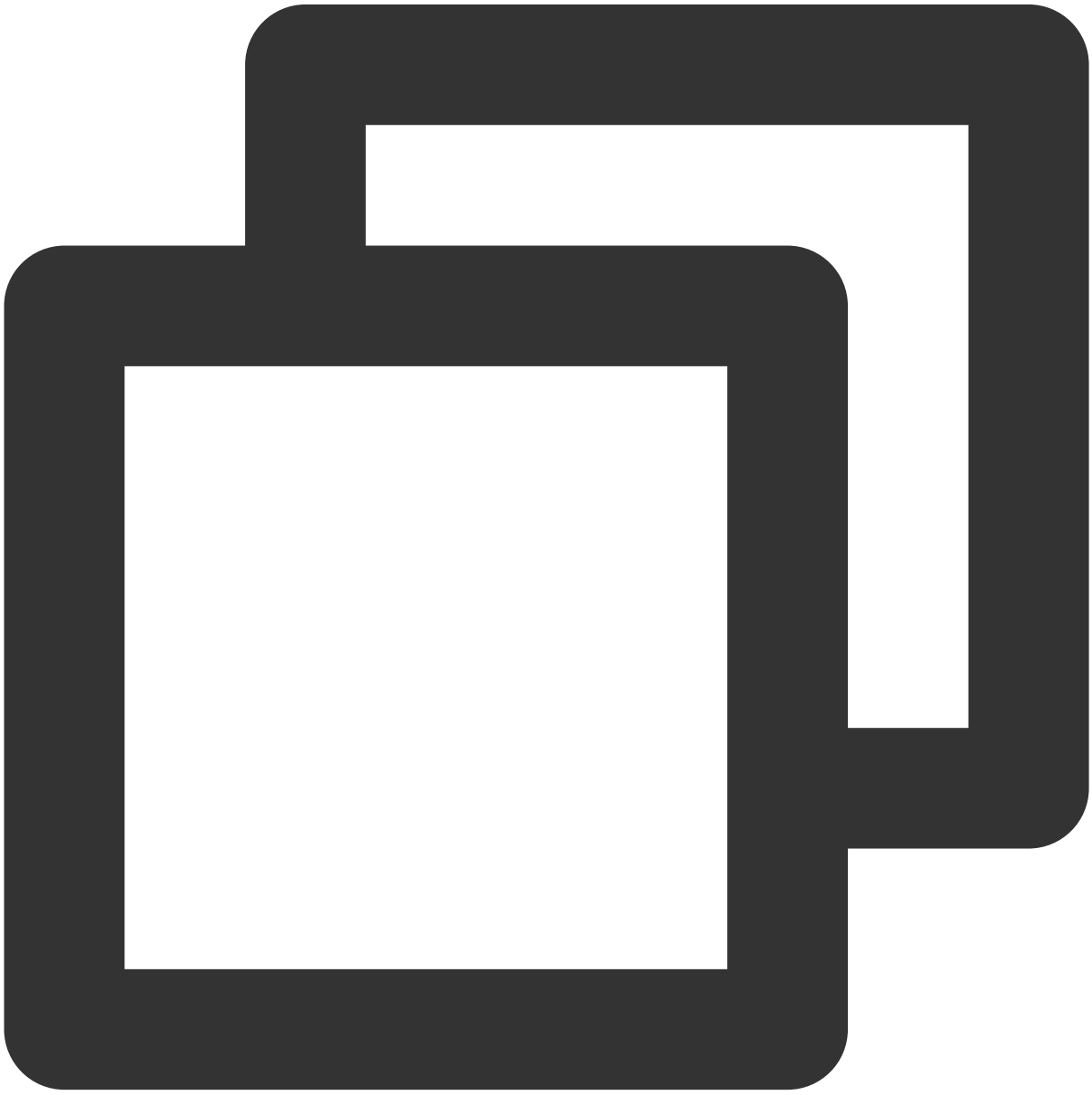
```
final V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGroupID("チャネルID");
option.setCount(20);
// その他の設定
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // チャンネル履歴メッセージの取得に成功
    }

    @Override
```

```
    public void onError(int code, String desc) {
        // チャンネル履歴メッセージの取得に失敗
    }
});
```

チャンネルメッセージ未読数

チャンネル内メッセージ未読数とサーバーメッセージ未読数は異なり、サーバー未読数はセッション情報の中に含まれ、チャンネルの未読数はチャンネルの基本情報の中に含まれます。Tencent Cloud IMのグループコールバックonTopicInfoChangedによって未読数をリアルタイムで取得します。



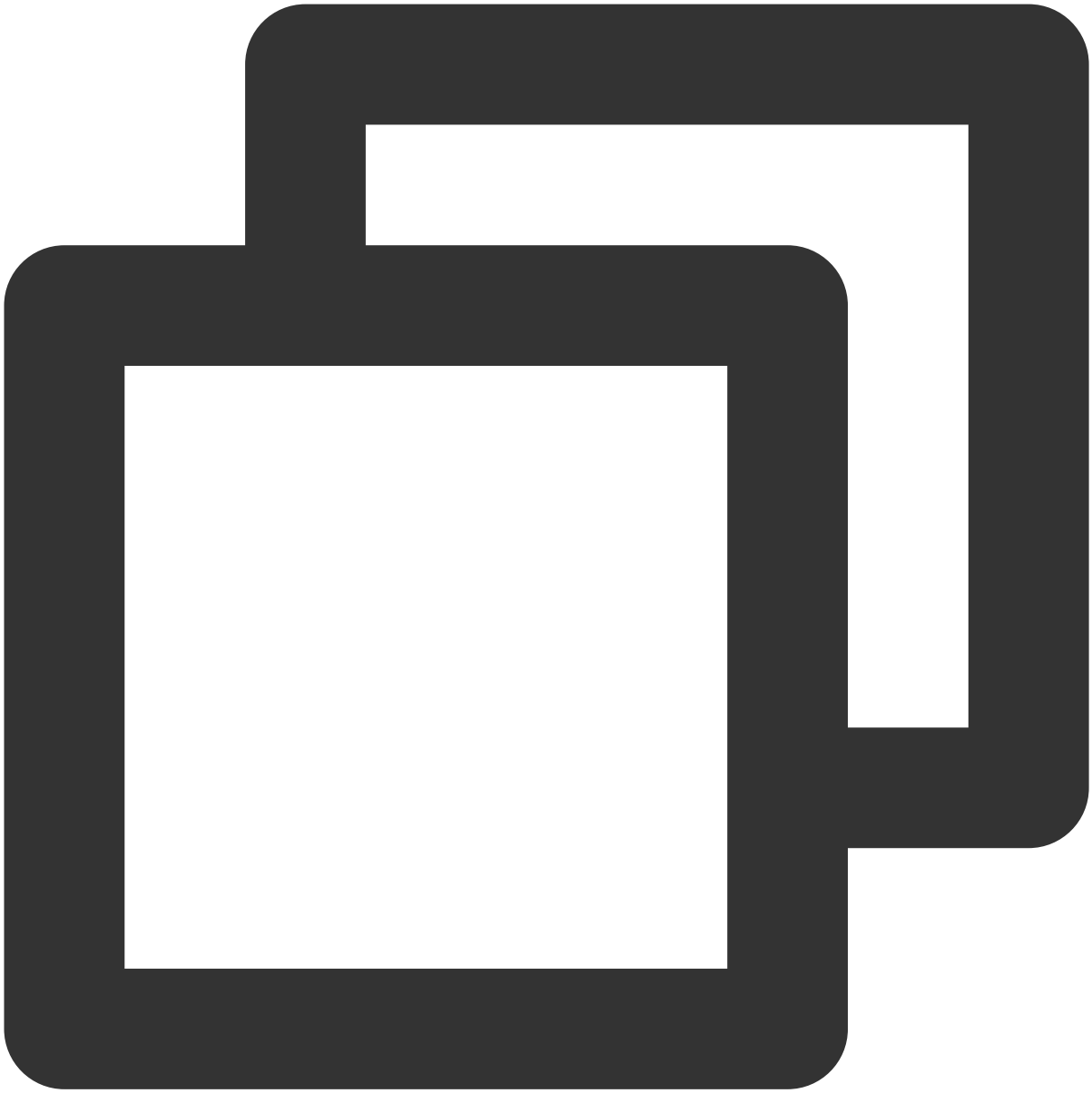
```
String groupID = "サーバーID";
List< String > topicIDList= new LinkedList(); // チャネルメッセージリスト
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList, new V2TIMValu
    @Override
    public void onSuccess(List<V2TIMTopicInfoResult> v2TIMTopicInfoResults) {
        // チャンネルID、チャンネル名、未読数などのチャンネル情報の取得
    }

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

    }
});
```

チャンネルメンバーリスト

サーバーに参加しているユーザーはデフォルトですべてのパブリックチャンネルに参加するため、サーバーのグループメンバリストを確認すればよい。機能チャンネルからメンバーリストの取得：



```
String groupID = "サーバーID";
int filter=0;//グループメンバー、管理者、一般メンバー...
long nextSeq=0;//ページングパラメータ
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) {
```

```
    }
});
```

プライベートチャンネルのグループメンバーリスト取得もgetGroupMemberListを呼び出し、プライベートチャンネルのグループIDを渡せばよいです。

サブチャンネル

サブチャンネルは、チャンネル内のメッセージについてさらに議論を行うグループであり、ユーザーは1つのチャンネル内で議論しているポイントをまとめて閲覧することができ、チャンネル内メッセージリストでは、サブチャンネルの概要を確認することもできます。サブチャネルはTencent Cloud IMでもグループの概念であり、1つのサブチャンネルが1つのグループとなります。

サブチャンネルの作成

サブチャンネルの作成は、チャンネル内の1つのメッセージにバインドすることも、Tencent Cloud IMが提供するCreateGroupインターフェースを介して個別に作成することもできますが、以下の点に注意する必要があります。

1. サーバー内のすべてのメンバーは、デフォルトで新しく作成されたサブチャンネルに配置されます。
2. サブチャンネルが作成されると、サーバーに参加しているメンバもサブチャンネルに参加します。
3. その後にサブチャンネルに参加したユーザーはサブチャンネル作成後のチャット履歴を閲覧できます。

以上のように、サブチャンネルを作成した後、グループを作成すると、サーバーのグループメンバーをサブチャンネルに加え、ユーザーがサーバーに参加する場合には、グループに参加した後のコールバックにおいて、サーバーのすべてのサブチャンネルにユーザーを加えます。この2つのステップは、ビジネスサーバー側で実行することをお勧めします。使用するAPIは次のとおりです：

1. サーバー内のメンバーを問い合せます。
2. グループの作成とメンバーの設定。
3. ユーザーをグループに招待する。

サーバー側のコールバックはグループ参加後のコールバック。

サブチャンネルの数

チャンネルリストでは、チャンネルの下にあるサブチャンネルのリストを取得できますので、サブチャンネルとサーバーの多対1の関係をバインドする必要があります。サブチャンネルに履歴メッセージが存在し、サブチャンネルとメッセージの1対1の関係もバインドする場合、Tencent Cloud IMは上記の関係をバインドする能力を現時点で提供していないため、ビジネス側が別途で対応しなければなりません。サービスが提供するHTTPインターフェースを介して、サブチャンネルの数とサブチャンネルリストを取得します。オンラインメッセージを送信して、サブチャンネルリストをリアルタイムでリフレッシュします。

サブチャンネルの概要

メッセージのサブチャンネルを作成すると、そのサブチャンネルのメッセージ概要がメッセージリストに表示されます。

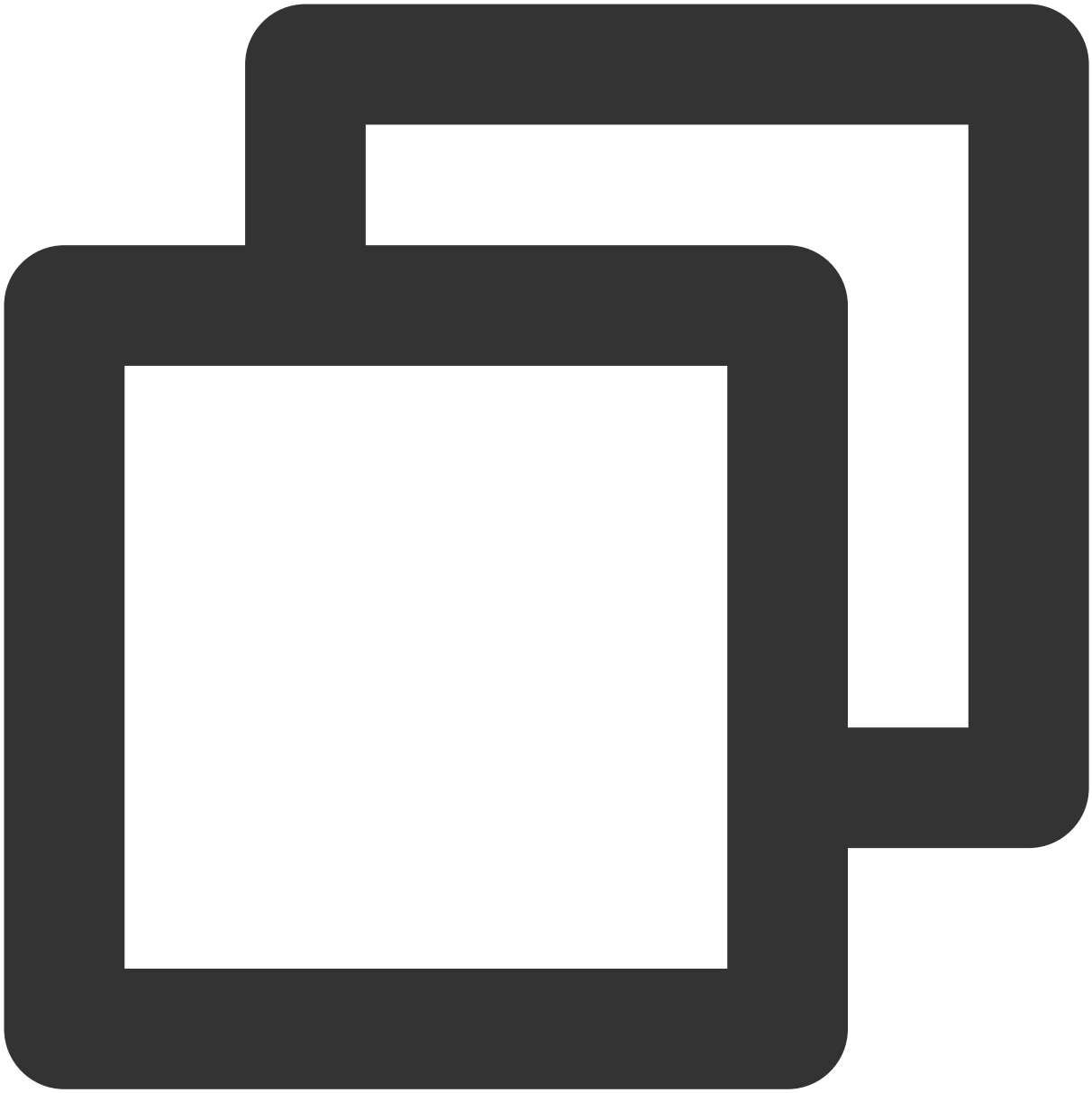
1. サブチャンネルにあるメッセージの数。
2. このサブチャンネル内のlastMessageに関する情報。
3. 情報はリアルタイムでメッセージリストに表示される必要があります。

サブチャンネルのメッセージ概要能力を実現するには、グループ内メッセージ送信後のコールバック、およびメッセージ編集の能力を組み合わせる必要があります。ユーザーがサブチャンネルでメッセージを送信した後、IMサービスはメッセージ送信後のコールバックをトリガし、関連情報をビジネス側に同期させ、ビジネス側でメッセージカウントを行い、lastMessage関連情報を処理し、メッセージ編集APIを介してメッセージに保存します。

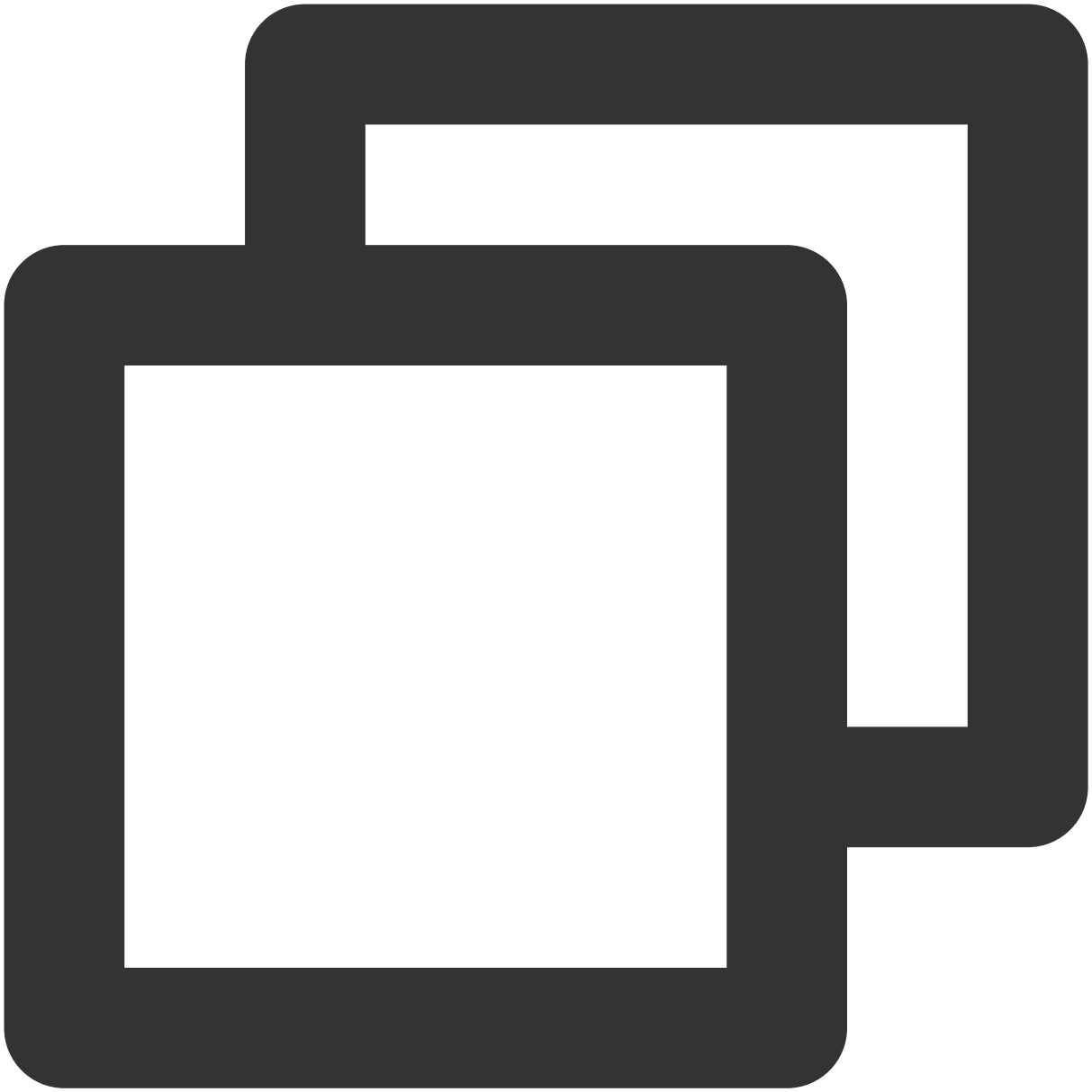
メッセージ関連

メッセージフィードバック

メッセージフィードバックは、次の図に示すように、ユーザーがメッセージを拡張するために使用されます。
すべてのメッセージのタイプは編集およびフィードバックが可能であり、コミュニティをサポートする必要があるため、データをcloudCustomDataフィールドに保存することをお勧めします。詳細なデータストレージフォーマットは次のとおりです：



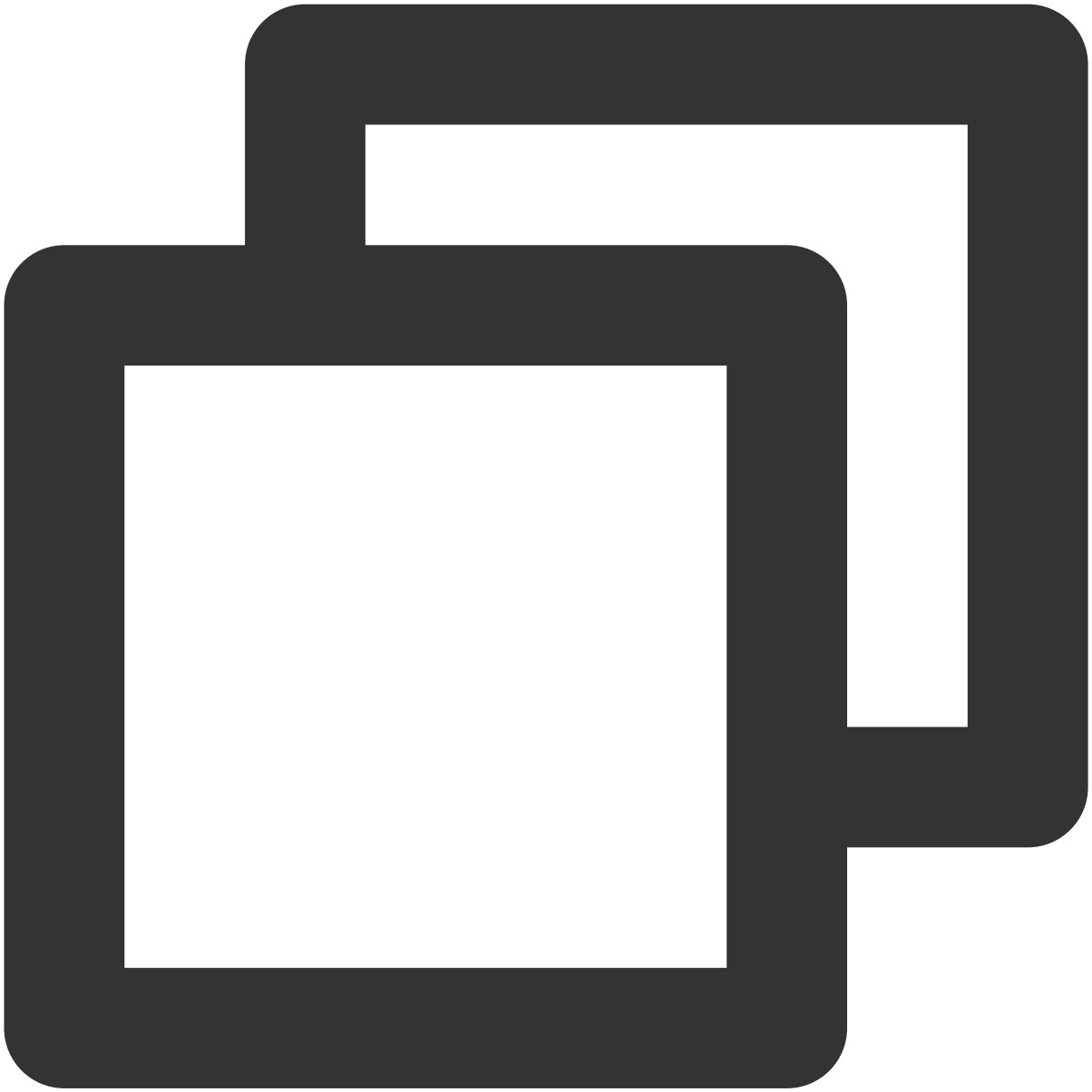
```
{
  "reaction": {
    "simle":["user1","user2"]
  }
}
```



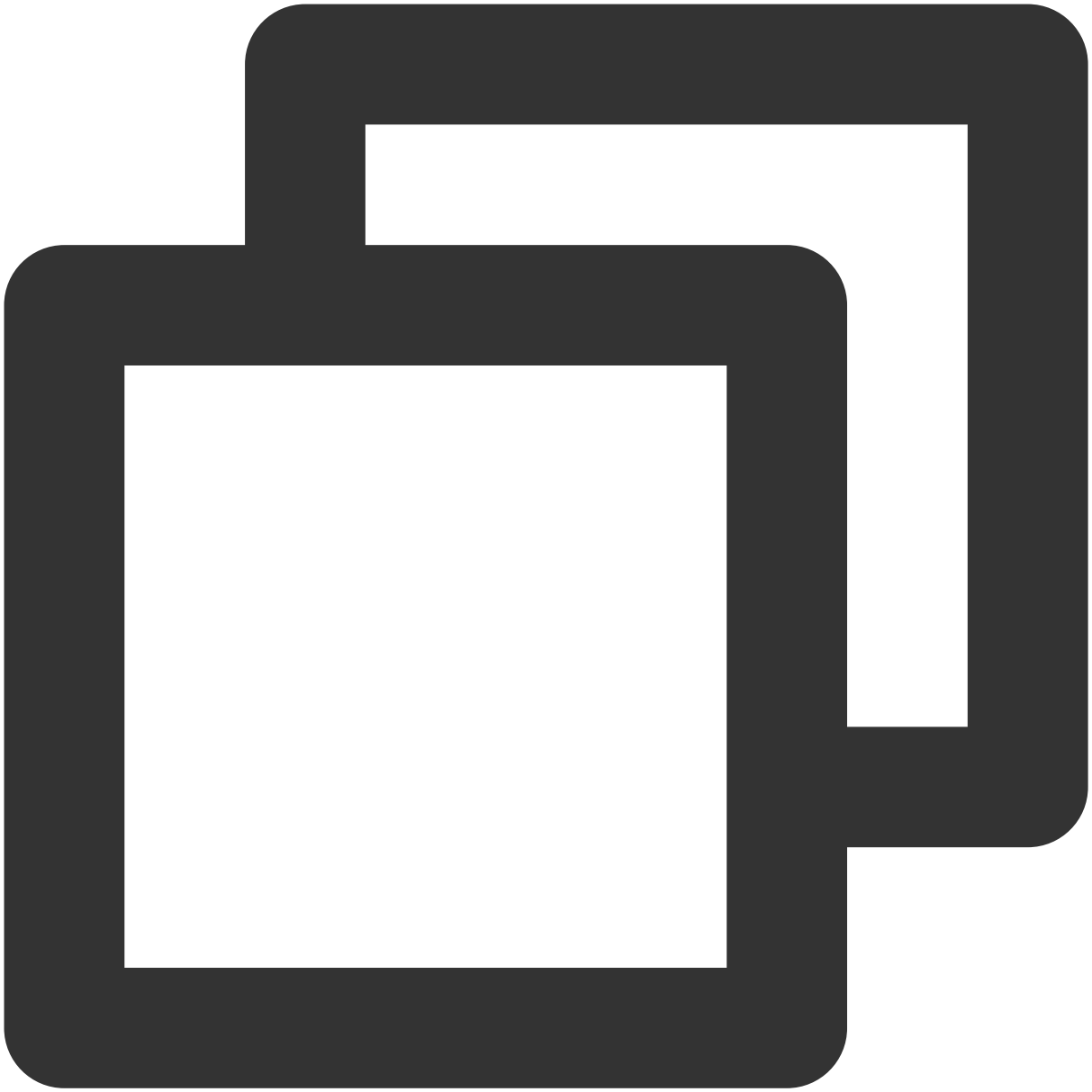
```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new V2TIMCompleteCa
    @Override
    public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
        //  メッセージの変更完了
    }
});
```

メッセージ編集

メッセージ編集は、メッセージフィードバックの原理と同様に、設定されたカスタムデータだけが異なります。同様に、すべてのメッセージにメッセージ編集を適用するために、cloudCustomDataフィールドを次の形式で編集することをお勧めします。



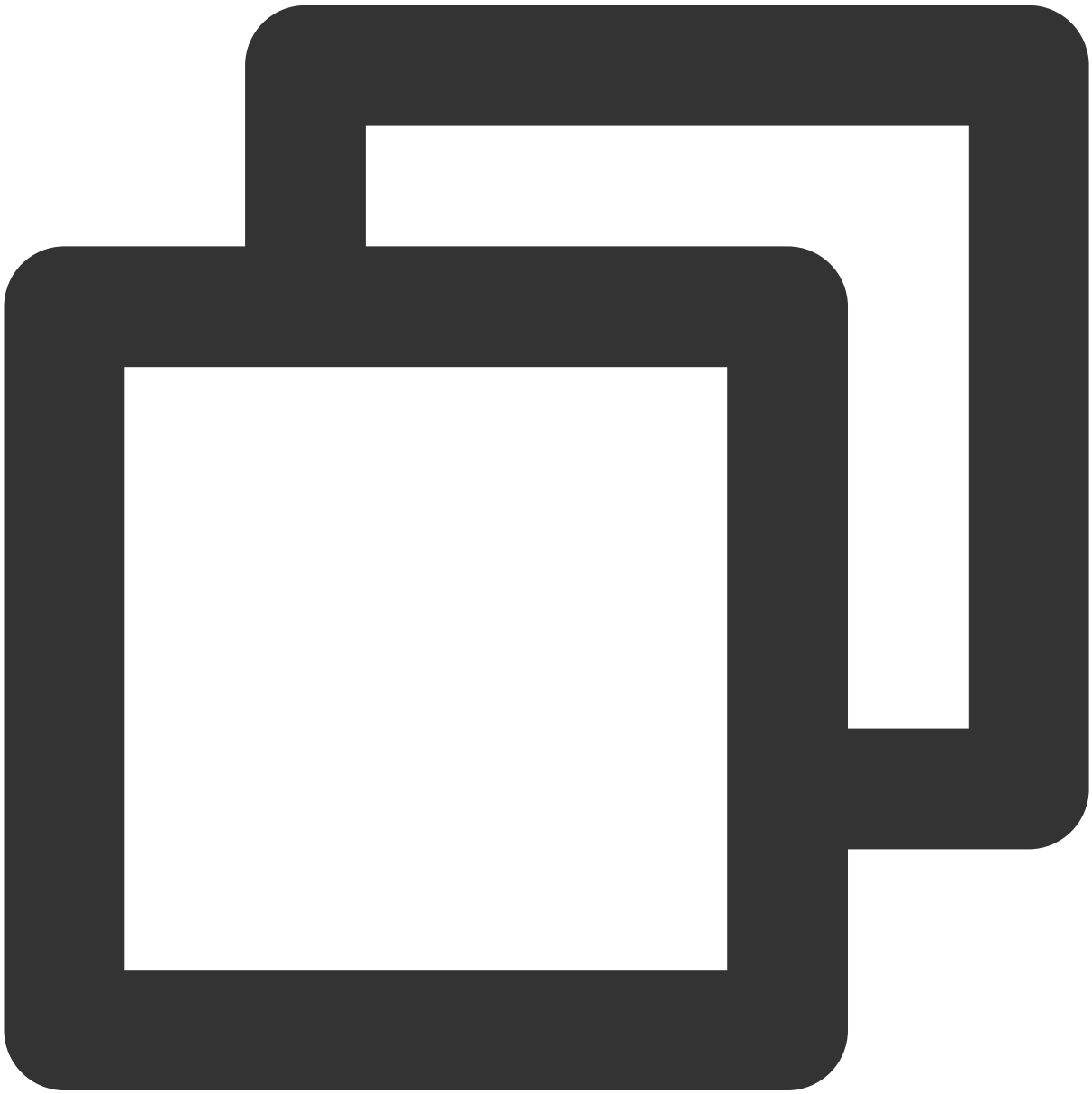
```
{
  "isEdited": true
}
```



```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new V2TIMCompleteCa
    @Override
    public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
        //  メッセージ編集完了
    }
});
```

メッセージのタグ付け

メッセージのタグ付けとは、ユーザーがグループチャットでメッセージにタグ付けを行い、他のユーザーがタグ付けられたメッセージを確認できるようにすることです。コミュニティはグループ属性をサポートしていないので、ここではカスタムメッセージを使用してメッセージタグ付け機能を実装します。
グループのカスタムメッセージを使用するには、Tencent Cloud IM Consoleを設定する必要があります。
次のように設定します：

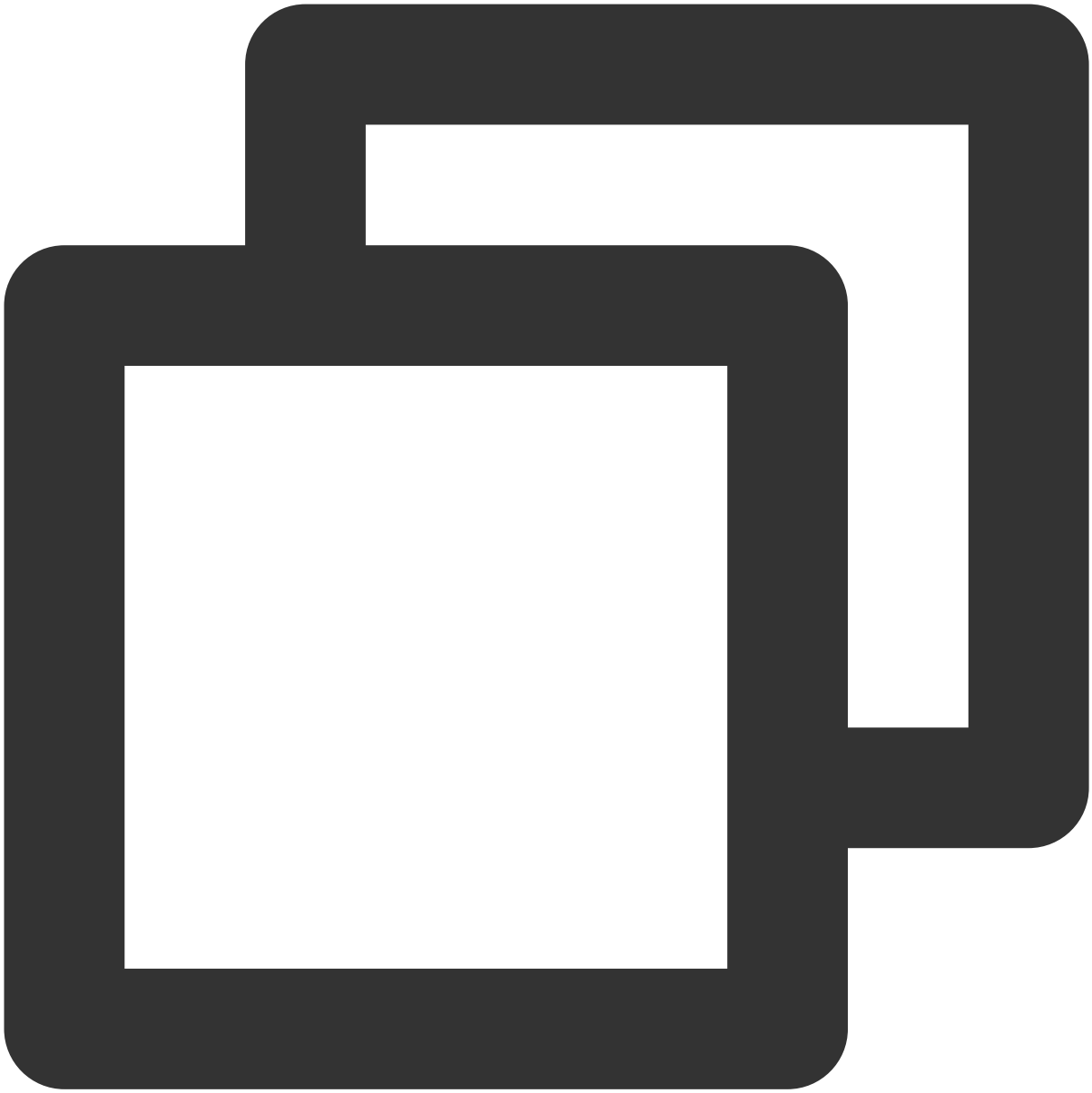


```
V2TIMGroupInfo info = new V2TIMGroupInfo();
info.setCustomInfo("pin data");
V2TIMManager.getGroupManager().setGroupInfo(info, new V2TIMCallback() {
    @Override
```

```
    public void onError(int i, String s) {
        // 設定失敗
    }

    @Override
    public void onSuccess() {
        //  設定成功
    }
});
```

設定されると、グループメンバーはonMemberInfoChangedイベントを受信し、オンラインにいないユーザーは、タグ付けられたメッセージの内容をグループ情報から取得できます。



```
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) {

    }
```

```
});
```

現時点では、カスタムフィールドはサーバー上にのみ設定でき、チャンネル上には設定できないことに注意してください。

入力中のステータス

友達が入力している間、他のユーザー側は入力中の状態を見ることができます。前述のように、Tencent Cloud IMが提供するオンラインメッセージを使用してこのメッセージを表示します。

1. オンラインユーザーのみが受け取ることができます。
2. メッセージローミングは保存されません。

また、パフォーマンスを最適化するために、次のような最適化を行うこともできます。

20s以内にメッセージを送信したユーザー双方が入力中状態メッセージの送信をトリガします。

同じテキストメッセージについて、複数回のオンラインメッセージ送信がトリガーされません。

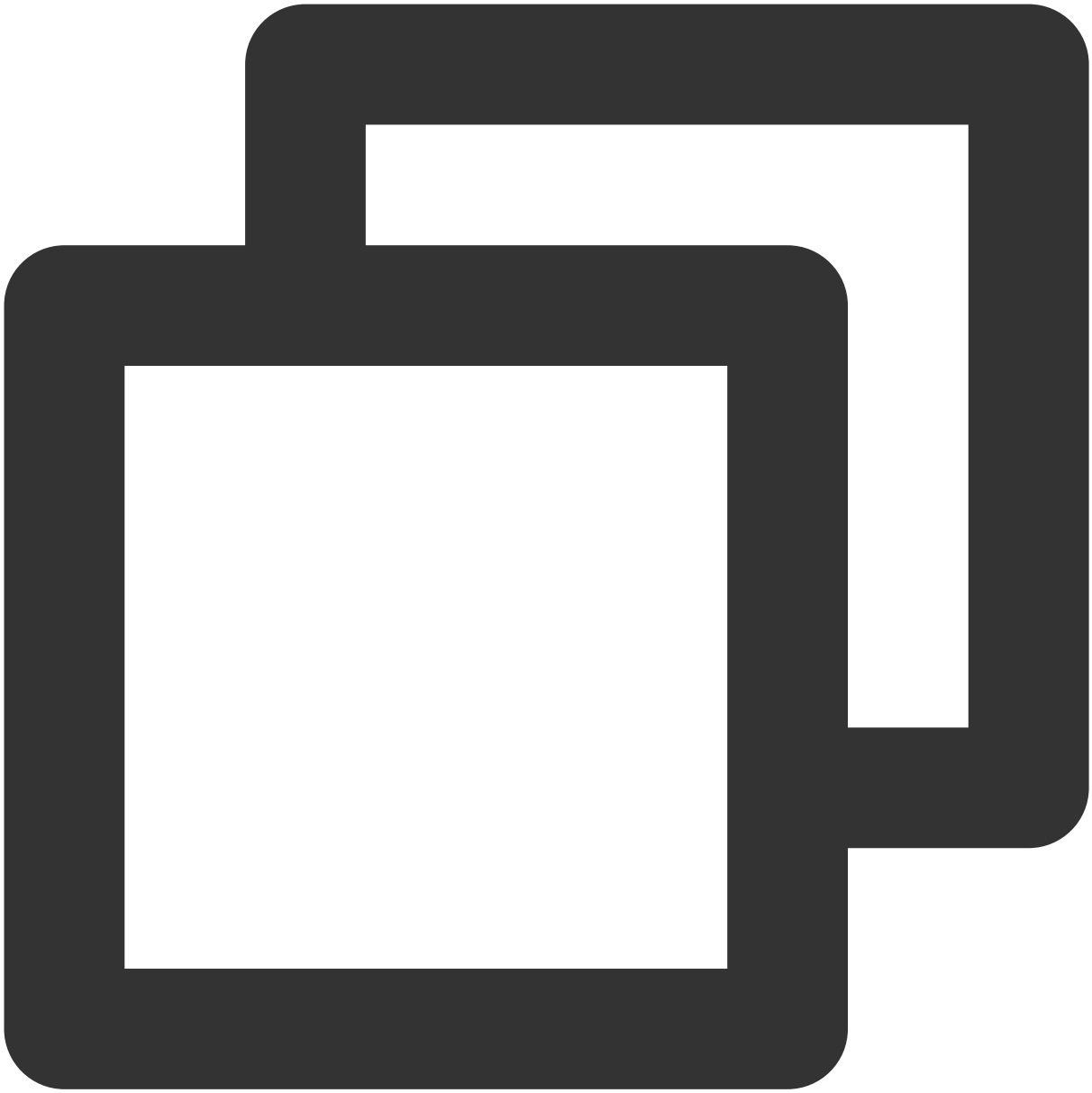
プライベートメッセージ

プライベートメッセージとは、Discordユーザーとユーザーとの間で、友達関係があるかどうかに関係なくメッセージを送信することができます。

友達関係がない場合、メッセージを送信するにはユーザーIDを知るほか、Tencent CloudのIMコンソールでメッセージ送信検出リレーションシップチェーンのオプションをオフにする必要があります。オフにしないと、メッセージの送信に失敗します。

ユーザーは、最初にaddFriendインターフェースを使用して友達を追加し、getFriendListを使用して自分の友達リストを取得することもできます。

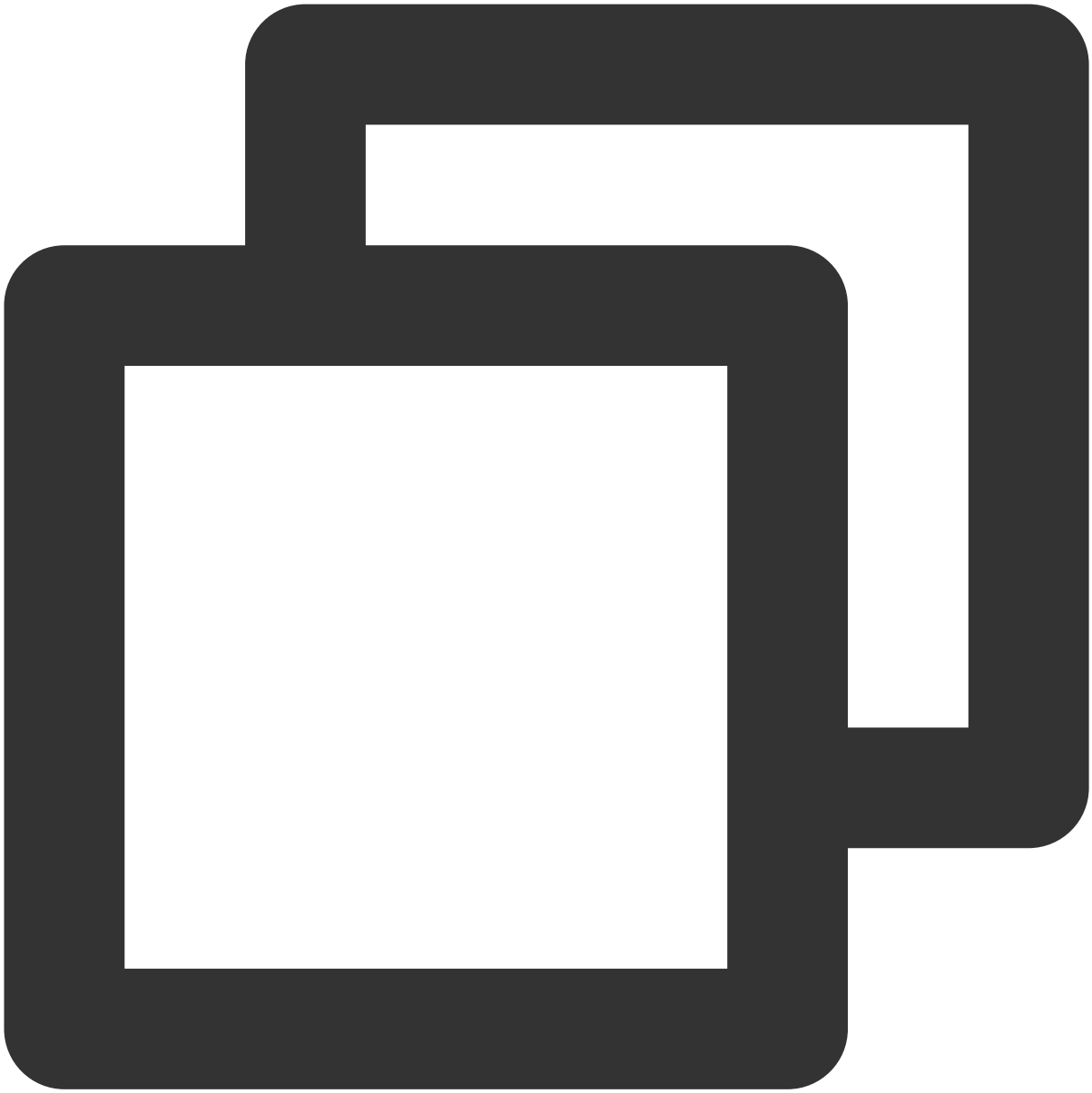
関連コードは次のとおりです：



```
// 友達を追加
V2TIMManager.getFriendshipManager().addFriend(info, new V2TIMValueCallback<V2TIMFri
            @Override
            public void onError(int i, String s) {
                // 友達追加失敗
            }

            @Override
            public void onSuccess(V2TIMFriendOperationResult v2TIMFriendOperationRe
               // 友達追加成功
            }
```

```
});
```



```
// 友達リストを取得する
V2TIMManager.getFriendshipManager().getFriendList(new V2TIMValueCallback<List<V2TIM
            @Override
            public void onError(int i, String s) {
                // 友達リスト取得失敗
            }

            @Override
            public void onSuccess(List<V2TIMFriendInfo> v2TIMFriendInfos) {
```

```
                // 友達リストを正常に取得する
            }
});
```

個人センター

オンラインステータス

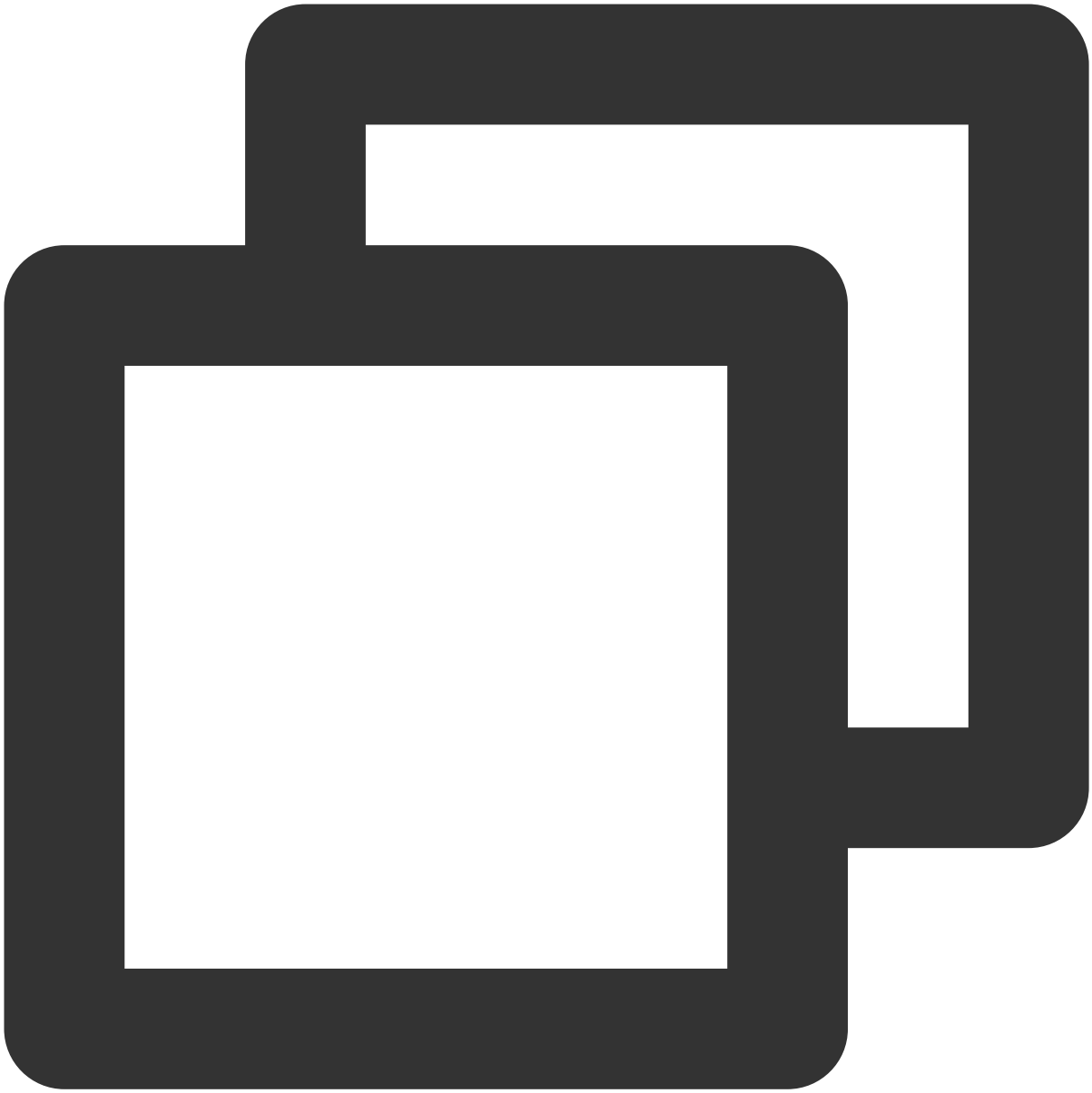
Discordユーザーパネルのプレゼンス状態機能は、Tencent Cloud IMが提供するプレゼンス状態APIによって実現されます。

setSelfStatusを使用して、ユーザー独自のオンライン/オフライン状態を設定します。ユーザー独自のオンライン/オフライン状態はIMによって設定され、開発者は変更できません。

getUserStatusを使用して友達のプレゼンス状態を取得します。

onUserStatusChangedを使用して、友達のプレゼンス状態の変更を受信します。

関連コードは次のとおりです：

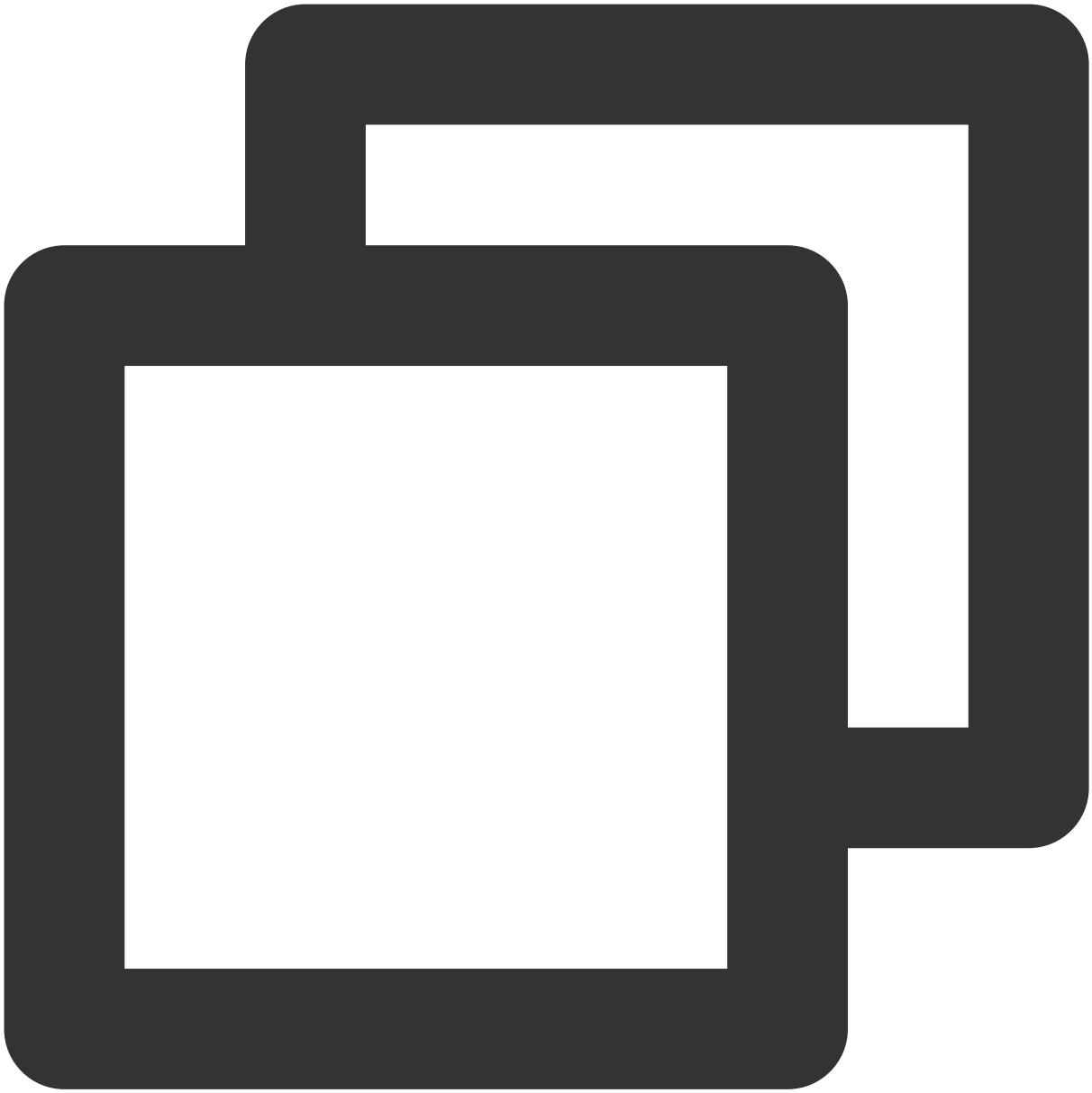


```
//  個人のプレゼンス状態の設定
V2TIMUserStatus customStatus = new V2TIMUserStatus();
V2TIMManager.getInstance().setSelfStatus(customStatus, new V2TIMCallback() {
    @Override
    public void onSuccess() {

    }

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

```
    }
});
```



```
//  友達のプレゼンス状態を取得
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) {

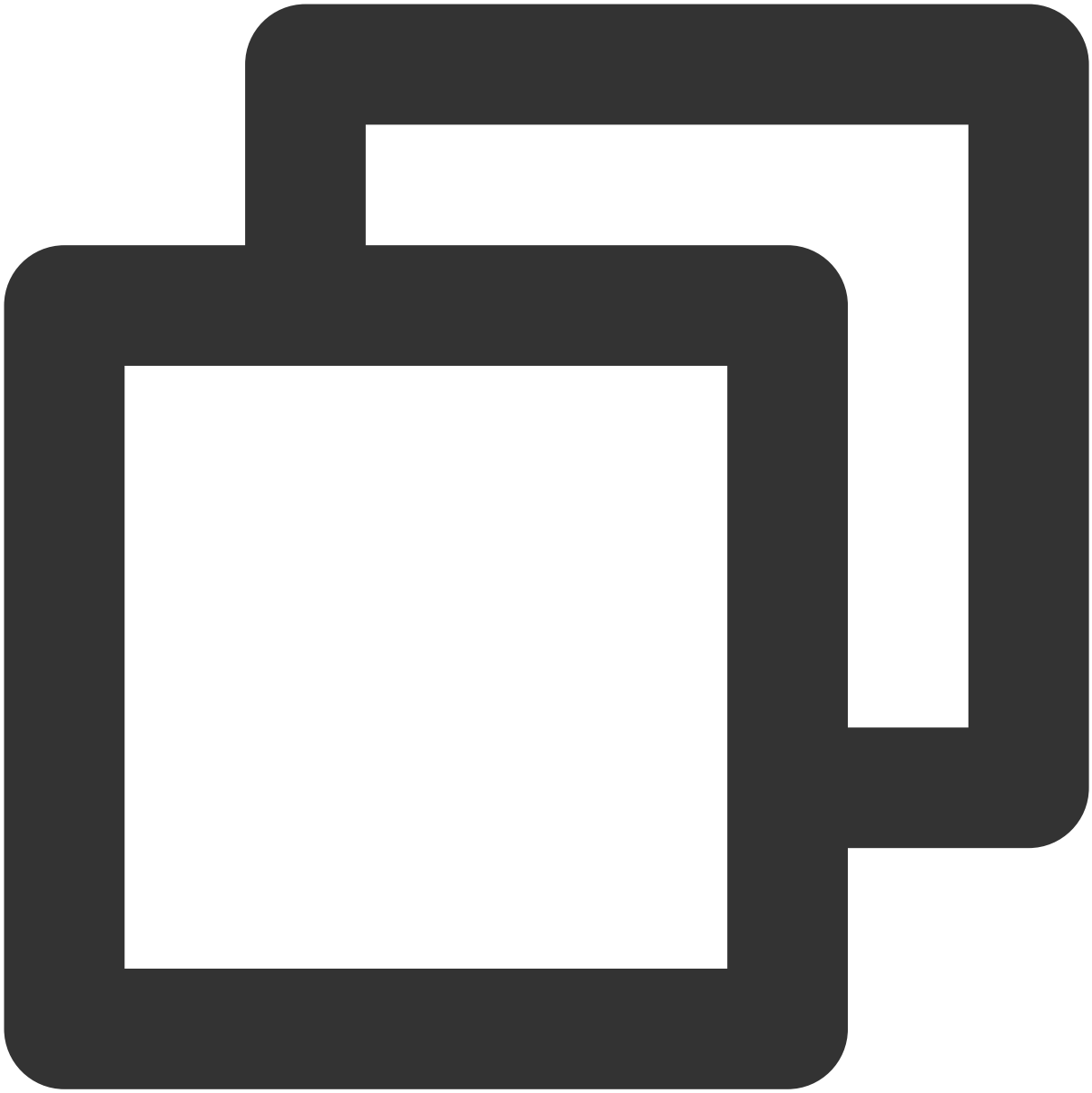
    }
});
```

個人情報関連

個人情報関連の機能は、Tencent Cloud IMが提供する情報リレーションシップチェーン関連のAPIを介して実装できます。

個人プロファイルgetUsersInfoを取得します。

個人プロファイルsetSelfInfoを設定します。

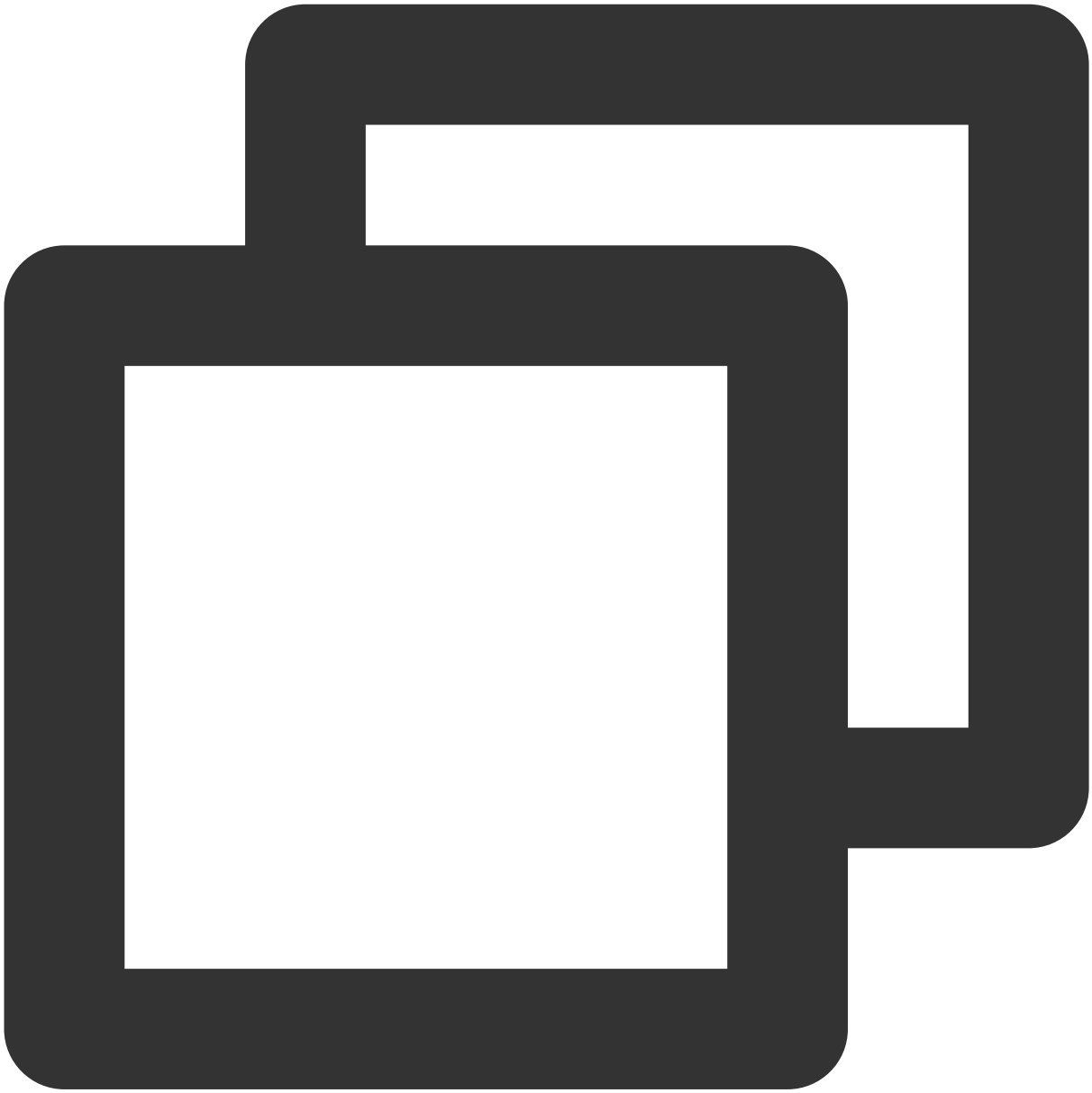


```
// 個人情報を設定する
final V2TIMUserFullInfo userFullInfo = new V2TIMUserFullInfo();
V2TIMManager.getInstance().setSelfInfo(userFullInfo, new V2TIMCallback() {
    @Override
    public void onError(int i, String s) {

    }

    @Override
    public void onSuccess() {
```

```
    }
});
```



```
// ユーザープロファイルの取得
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) {

    }
});
```

その他

検索機能

Discordの検索機能は次のとおりです：

Tencent Cloud IMは、次のような豊富な検索機能を提供します。

メッセージの検索searchLocalMessages。

グループの検査searchGroups。

グループメンバーの検索searchGroupMembers。

友達の検索searchFriends。

API利用の詳細については、公式ドキュメント検索についてをご参照ください。

オフラインプッシュ機能

ユーザーがオフラインになった場合、Tencent Cloud IMのオンラインメッセージはユーザーに届かません。これは端末機器メーカーが提供するオフラインプッシュ機能が必要で、Tencent Cloud IMはオフラインプッシュを導入するのにも非常に便利です。詳細についてはオフラインプッシュをご参照ください。

センシティブワード検証

情報を送信したり、プロファイルを設定したりする際には、内容をフィルタリングする必要があります。Tencent Cloud　IMもこうした解決策を提供し、ユーザーが正しくIMを利用できるよう支援しています。

ゲーム内IM統合ガイド

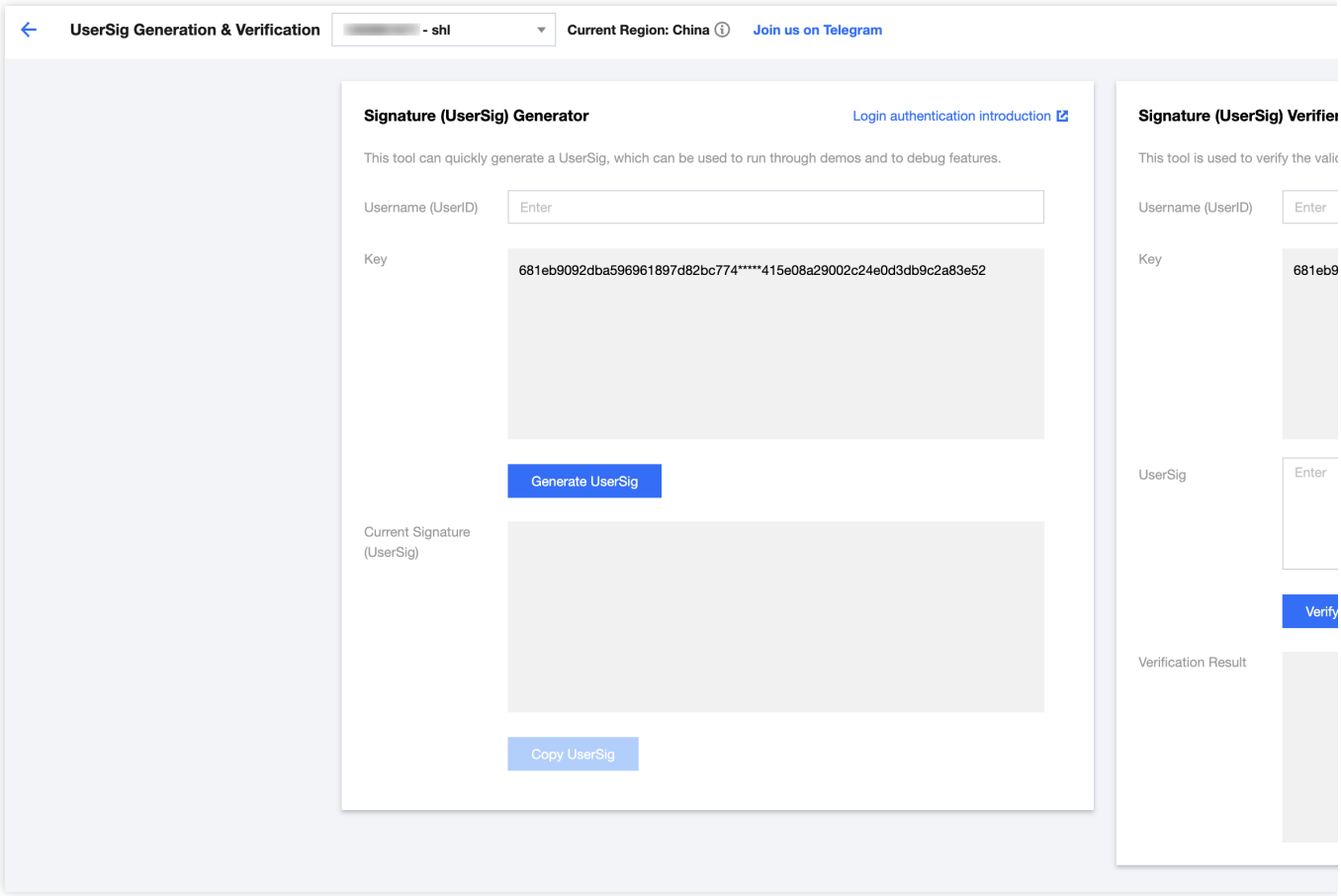
最終更新日：：2024-04-11 17:31:14

ゲームにおいて、インスタントメッセージングのニーズがよく発生します。マルチプレイヤーゲームでは、インスタントチャットが不可欠な機能となっています。ゲームプラットフォーム自体に関わる、グループタイプ、カスタムメッセージタイプ(例えば、ゲーム内で道具を贈るとか取引をするなどの業務)、グローバルアクセスなどに対するニーズは比較的複雑です。開発者が迅速に業務を理解し、ニーズを実装するように、本書では、ゲームチャットを構築中によく見られるニーズの実装方法、発生しうる問題、注意事項などを整理し、参考として提供します。

準備作業

キーを使用したUserSigの計算

IMのアカウントシステムでは、ユーザーログインに必要なパスワードはユーザーのサーバーでIMが提供するキーを使用して計算されます。ユーザーはドキュメントUserSigの計算をご参照ください。下図に示すように、開発段階では、クライアントでの開発に支障がないよう、コンソールでのUserSigの計算を行うこともできます。

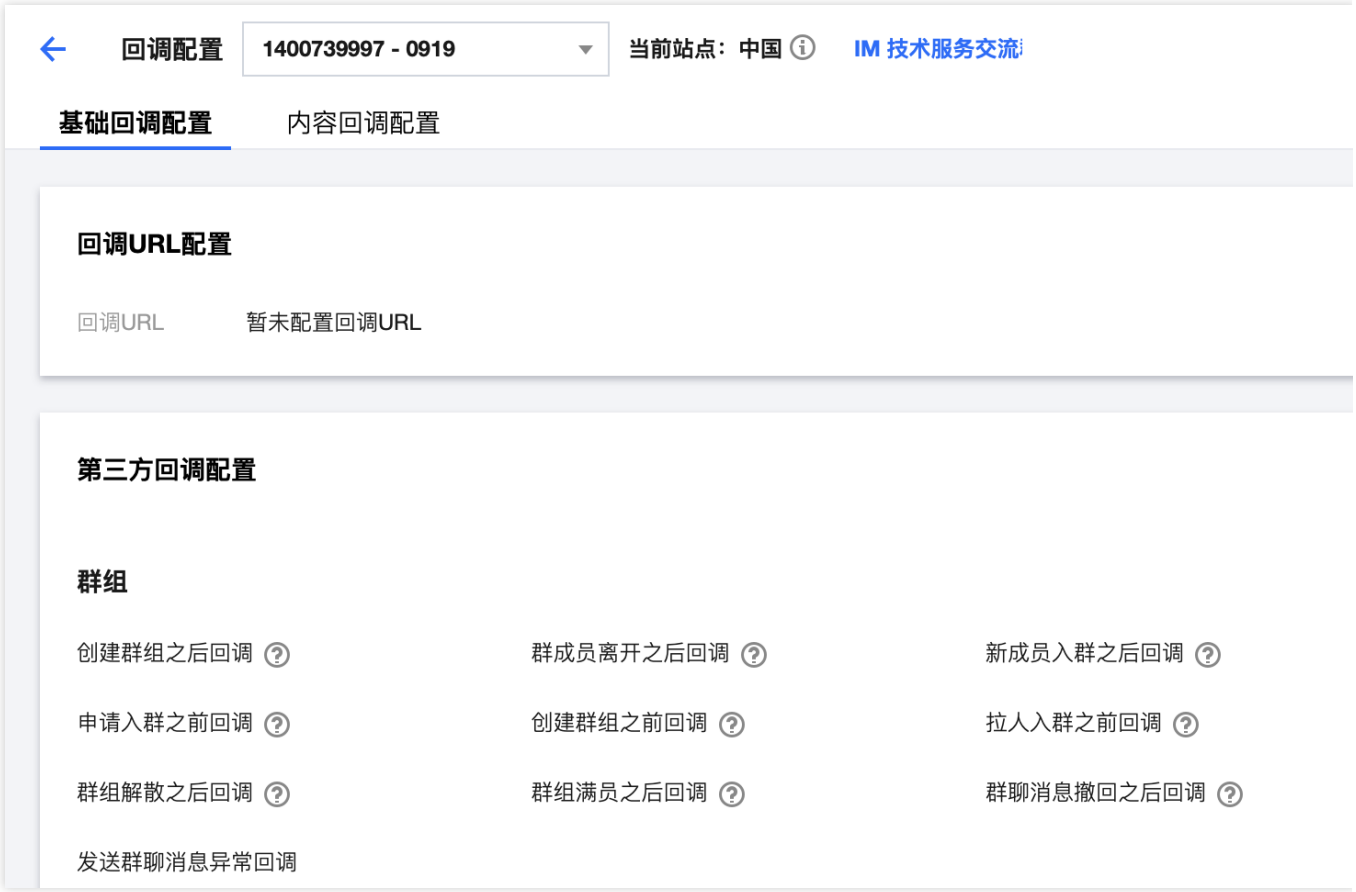


管理者アカウントの設定

ゲーム内IMの管理において、管理者がゲームに対して、メールで公告を送信したり、一時的にチームを組むメッセージを管理したりすることがあります。この場合、IMサーバーAPIを使用してください。サーバー側のAPIを呼び出すには、IM管理者アカウントの作成を行ってください。IMはデフォルトでUserIDがadministratorのアカウントを開発者に提供します。開発者は運用シーンに応じて複数の管理者アカウントを作成することもできます。IMで最大5つの管理者アカウントを作成できることにご注意ください。

コールバックアドレスの設定およびコールバックの有効化

ゲーム内でチームを組むなどのニーズを実装する時、IMのコールバックモジュールを使用する必要があります。具体的には、運用シーンによって、IMバックグラウンドで開発者の業務バックグラウンドをコールバックすることがあります。開発者は下図のように、HTTPのインターフェースを提供し、コンソール > コールバック設定モジュールに設定するだけでよいです。



クライアントSDKの統合

準備作業をすべて完了した後、IMクライアントSDKをユーザーのプロジェクトに統合する必要があります。開発者はご自身のニーズに応じて、統合ソリューションを選択することができます。詳しくは、クイックインテグレーションシリーズドキュメントをご参照ください。
以下で、ゲームへのIM統合においてよく使用される機能をまとめ、開発者のご参照用としてベストプラクティスソリューションを提供します。関連の実装コードも併せてご参照ください。

ゲームチャットルームに関する機能の開発ガイド

ユーザープロフィール

よくあるユーザープロフィール

ゲーム業務で保存される一般的なユーザープロフィールは、基本情報プロフィールとその他情報プロフィールに分類できます。

基本情報	その他情報
ユーザー名、性別、誕生日、レベル、キャラクター、携帯電話番号など	その他のゲームで必要な情報

プロフィールの保存

ゲーム業務に数多くのユーザーが存在するため、膨大なユーザープロフィールを保存するのは簡単なことではありません。Tencent Cloud IMは、ユーザープロフィールホスティング機能を実装し、プロフィールに関連する完全なソリューションを提供します。Tencent Cloud IMへの保存と業務バックグラウンドへの保存の違いを以下に示します。

比較項目	IM	業務バックグラウンド
ストレージ容量	自動スケーリング	容量が限られており、スケールリングが難しい
ユーザープロフィール	標準フィールドとカスタムフィールドを提供し、フィールドの長さと名前に制限がある	カスタマイズ可能で柔軟性が高い
プロフィールの読み書き	使いやすいサービスインターフェースとヘルプガイドを提供する	自分で開発する必要がある
インターフェース	インターフェース呼出し頻度に制限あり:最大200回/秒	必要に応じて、インターフェース呼出しなどの機能を自分で開発できる
セキュリティ	リモート災害復旧・マルチサイトデプロイが可能	セルフメンテナンス

上記により、IMユーザープロフィールホスティングサービスを使用すれば、プロフィールの保存、読取り/書込み機能を活用できるほか、次のメリットもあります。
1. IMは、リモート障害復旧、マルチサイトデプロイ及び自動スケーリングを提供し、サーバーのダウンタイム、マルチマスター/スレーブレプリケーション、スケーリングなど複雑な処理プロセスからお客様を完全に解放します。
2. IMは、業界共通の業務処理フローを提供し、お客様がユーザープロフィールの業務ロジックから完全に解放されるようお手伝いします。
3. IMは、プロフェッショナルな運営フローと運営チームを提供し、年間を通じて99.99%という安定したサービス品質をお約束し、ユーザーに評判の高い安定したサービスを提供するお手伝いをします。
4. IMは、使いやすいサービスインターフェースとクイックアクセスヘルプガイドを提供し、プロセス全体で優れたサービスを提供します。

IMでユーザープロフィールを保存する方法

IMストレージソリューションには、プロフィールの保存、読取り/書込み機能が含まれています。ここでは、IMでユーザープロフィール、友達プロフィールおよび拡張プロフィールを保存する方法を説明します。プロフィールはすべて `Key-Value` の形式で表します。そのうち、 `Key` は `String` 型で、アルファベット大文字・小文字、数字、アンダーバーのみサポートしています。 `Value` には以下のタイプがあります：

タイプ	説明
uint64_t	整数(カスタムフィールドではサポートされていない)
string	文字列で、500バイト以内
bytes	bufferの一部で、500バイト以内
string 配列	文字列配列で、各文字列は500バイト以内で、友達テーブルのTag_SNS_IM_Groupフィールドのみで使用可能

ユーザープロフィール:ユーザープロフィールには、標準フィールドとカスタムフィールドが含まれています。カスタムフィールドの詳細は、後述する拡張プロフィールをご参照ください。IMがサポートしているユーザープロフィールの標準フィールドについては、ユーザープロフィール標準フィールドをご参照ください。
友達プロフィール:友達プロフィールは、標準友達フィールドとカスタム友達フィールドをサポートします。IM友達リストには、最大3000人の友達を追加できます。そのうち、標準友達フィールドは次のとおりです：

フィールド名	タイプ	説明
Tag_SNS_IM_Group	Array	友達グループ。文字列配列
Tag_SNS_IM_Remark	string	友達ノート
Tag_SNS_IM_AddSource	string	友達追加ルート
Tag_SNS_IM_AddWording	string	友達追加コメント
Tag_SNS_IM_AddTime	Integer	友達追加タイムスタンプ

詳細については、
友達標準フィールド
をご参照ください。- **カスタムプロフィール**：カスタムプロフィールフィールドは、IMコンソール > アプリケーション設定 > 機能設定 で申請できます。カスタムプロフィールフィールドは申請して5分以内に有効になります。 - ユーザープロフィール管理の詳細については、ユーザープロフィール管理をご参照ください。 - 友達リレーションシップチェーンのカスタムプロフィールの管理については、友達カスタムフィールドをご参照ください。

IM ユーザープロフィールの保存制限

業務特徴の制限
ストレージデータ：stringとbufferタイプのストレージ長は500バイト以内にする必要があります
カスタムフィールド:カスタムフィールドのキーワードは8バイト以内のアルファベットを使用する必要があります。カスタムフィールドの値は500バイト以内にする必要があります
友達リレーションシップチェーン：ユーザー1人あたり3000人の友達をサポートします
インターフェース関連の制限

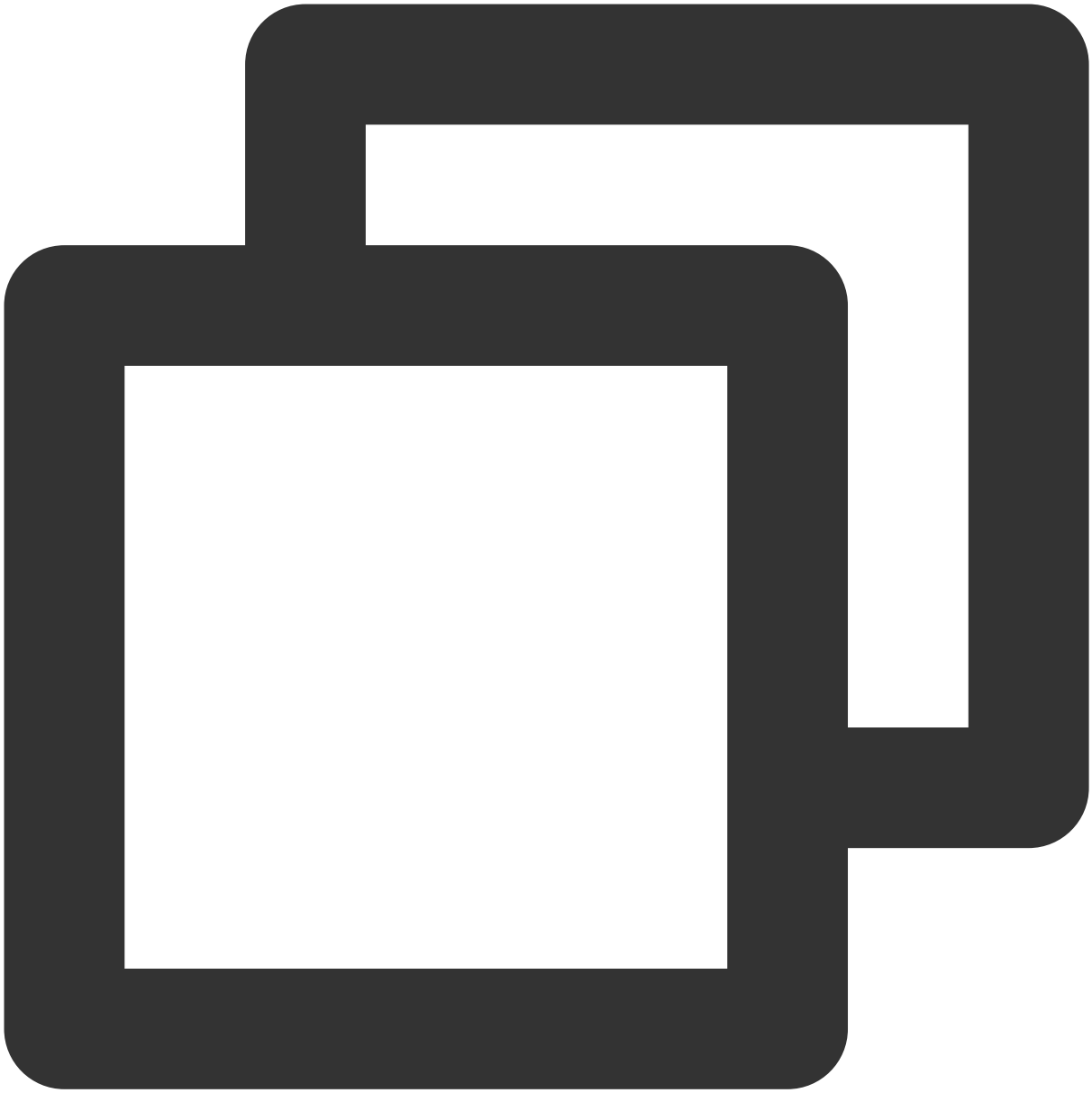
アカウント管理：1回でユーザー名を100個までインポートできます。1回のリクエストで最大500人のユーザー状態を検索できます
その他の呼出し頻度：最大200回/秒
使用制限の詳細については、使用制限をご参照ください。

メールシステム

現在のゲームでは、メールシステムは必須機能と言えるでしょう。メールにテキストメッセージを書いたり、ゲーム内の道具や報酬などを添付ファイルとして付けたりできます。メールは1人への送信が可能で、イベントの報酬を付与するように多数人への送信もできます。以下では、プレイヤのメール受信、メーリングリスト、メールの未読カウント、全員向けメール、メールの有効期限などから、メールシステムの機能の実装を詳しく説明します。

メールの送受信

プレイヤーのメール受信:システムメールが正常に送信され、プレイヤーがネットワークに接続している場合、プレイヤーはシステムメールを正常に受信できます。受信したメールは、メールセッションのメッセージリストを取得することで履歴/最新のメールを取得できます。また、新規メッセージを受信するコールバックを追加/削除することで、すべてのタイプのメッセージ(テキスト、カスタマイズ、リッチメディアメッセージなどを含む)の受信を監視できます。Unityを例としたサンプルコードは次のとおりです：



```
// メッセージイベントリスナーを設定する
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
foreach(Message message in messages)
{
  foreach (Elem elem in message.message_elem_array)
  {
    // 次のメッセージがある
    if (elem.elem_type == TIMElemType.kTIMElem_Text)
    {
       string text = elem.text_elem_content;
    }
```

```
    }
  }
  })
  // `RecvNewMsgCallback`コールバックを監視し、その中でメッセージを受信する
  // メッセージを受信したくない場合、`RemoveRecvNewMsgCallback`を呼び出してリスナーをリムーブしま
```

詳細については、Unity-メッセージ受信をご参照ください。

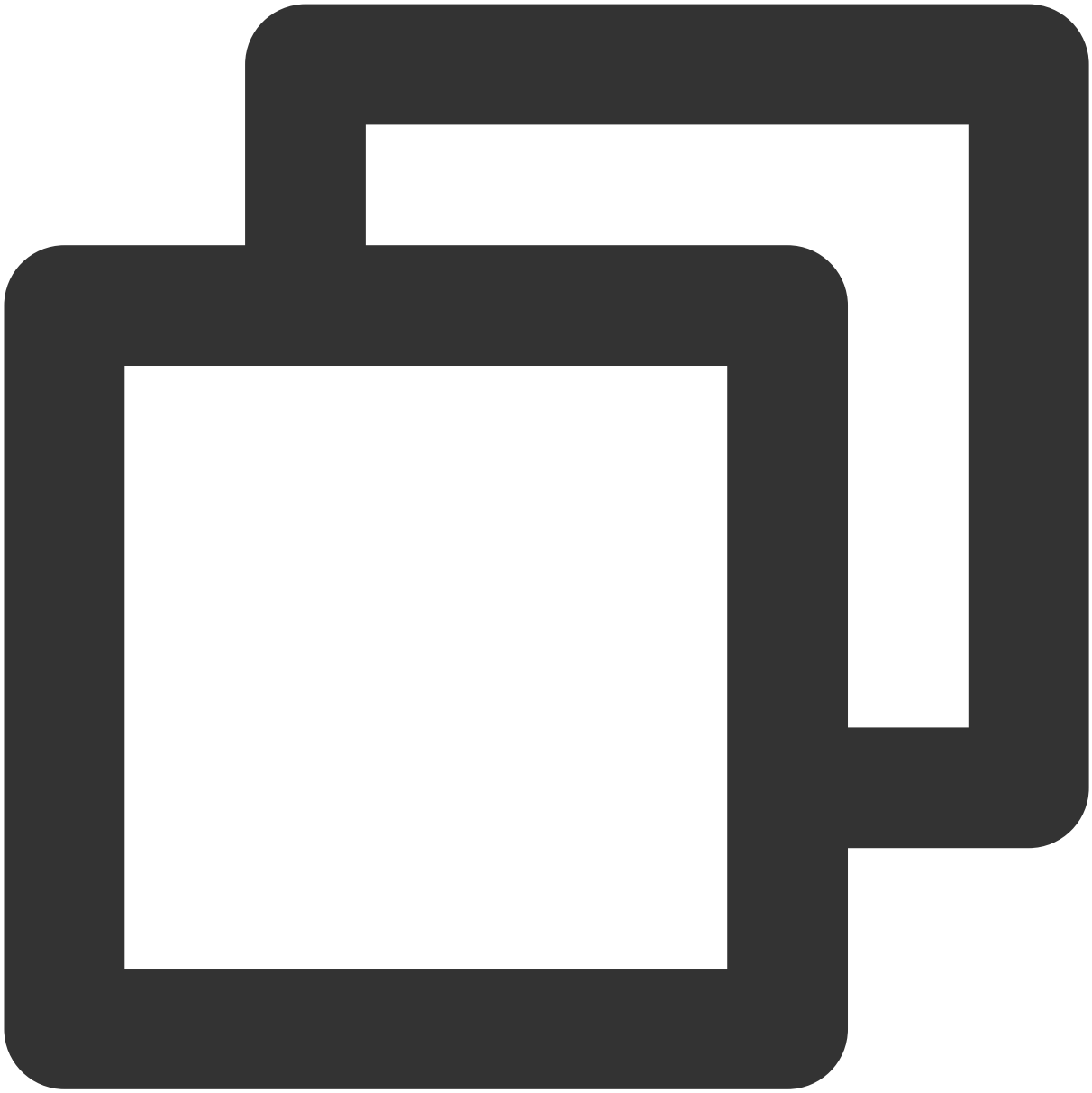
システムからのメール送信:システムはいくつかの方法でサーバーAPIを使用し、ユーザーにシステムメールを送信できます。以下でこれらの方法の特徴を説明します。

メソッド	特徴	運用シーン
シングルチャットメッセージの個別送信	指定したアカウントにメッセージを送信する。受信者が見た送信者は管理者ではなく、管理者が指定したアカウントです	特定のユーザーにメッセージを送信します。例えば、ランクマッチの報酬など
シングルチャットメッセージの一括送信	1回で最大500人のユーザーにシングルメッセージを送信できます。呼出し頻度は最大200回/秒です	グループを作成する必要がないため、特定のユーザーにメッセージを送信できます。グループを作成する必要がないため、柔軟性が高いです。ただし、送信するユーザーが多い場合、数回分けて送信する必要があります
グループにおける一般メッセージの送信	グループ内で一般メッセージを送信するには、ユーザーを同じグループに追加する必要があります。	多くのユーザーにメッセージを送信する場合、グループを作成してから一般メッセージを送信する方法があります。グループの最大メンバー数として、コミュニティ(community)は10万人です
全員プッシュサービス	App内の全員にメッセージをプッシュする時、ユーザータグと属性を指定してメッセージを送信できます	App内の全員にメッセージをプッシュしたり、人数が非常に多く、イベントプッシュメールなどの特徴的な属性がある場合に利用できます

説明：

全員プッシュはIMPremium editionで提供された機能です。使用するには、Premium editionを購入し、コンソール > 機能設定 > ログインとメッセージ > 全員プッシュで有効にするように設定する必要があります。

そのうち、グループ内で一般メッセージを送信するリクエストパケットの基本形式の例を以下に示します。



```
{
    "GroupId": "@TGS#2C5SZEAEF",
    "Random": 8912345, //乱数。5分間で数字が同じな場合、同一メッセージだと判断する
    "MsgBody": [ // メッセージボディ。1つのelement 配列で構成される。詳細については、フィール
        {
            "MsgType": "TIMTextElem", // テキスト
            "MsgContent": {
            "Text": "red packet"
            }
        },
        {
```

```
            "MsgType": "TIMCustomElem", // カスタマイズ
            "MsgContent": {
            "Data": "message",
            "Desc": "notification",
            "Ext": "url",
            "Sound": "dingdong.aiff"
            }
        }
    ],
}
```

`MsgBody` (メッセージボディ)はメッセージ配列です。テキストメッセージとカスタムメッセージをMsgBody に入れて送信できます。

メーリングリスト

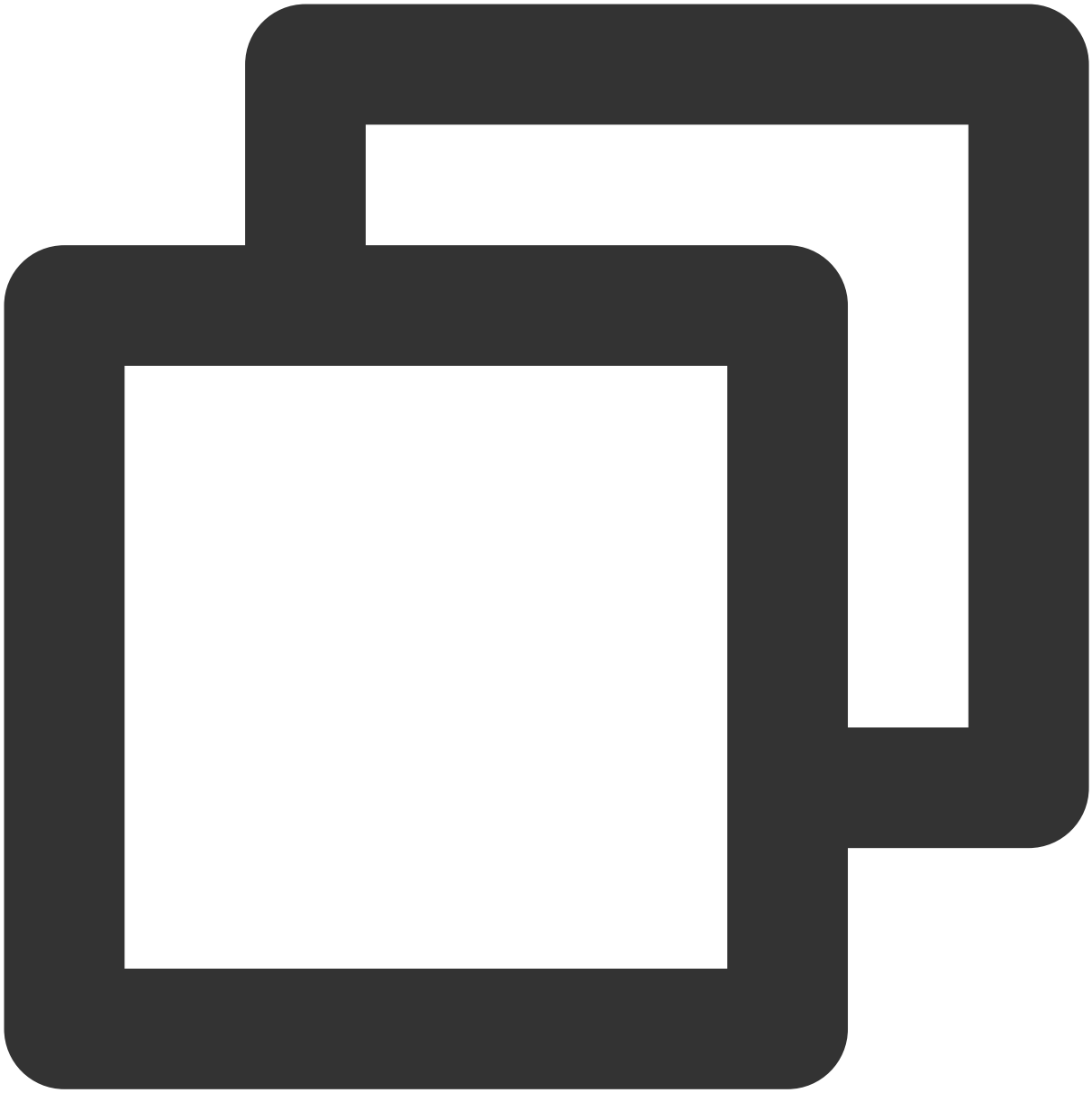
メーリングリスト履歴の保存はメッセージの保存と同じで、「C2Cシングルチャットメッセージ履歴の保存」と「グループチャットメッセージ履歴の保存」 に分けられます。グループチャットの最小人数は2人なので、管理者が指定したアカウントとメールを受信するユーザーを新しいグループとして作成して保存できます。

説明：

体験版とStandard editionの保存期間は7日間、アルティミト版は30日間です。Standard editionとPremium editionは、保存期間の延長をサポートします。コンソールにログインして、関連の設定を変更できます。

メッセージ履歴保存期間の延長は、有料の付加価値サービスです。料金の詳細については、付加価値サービスの料金をご参照ください。

ネットワークが正常な場合、クラウドから最新のデータを取得します。ネットワークが異常な場合、ローカルに保存されたメッセージ履歴を返します。このインターフェースは改ページでの取得をサポートします。メーリングリスト履歴を取得するUnityのサンプルコードは次のとおりです：

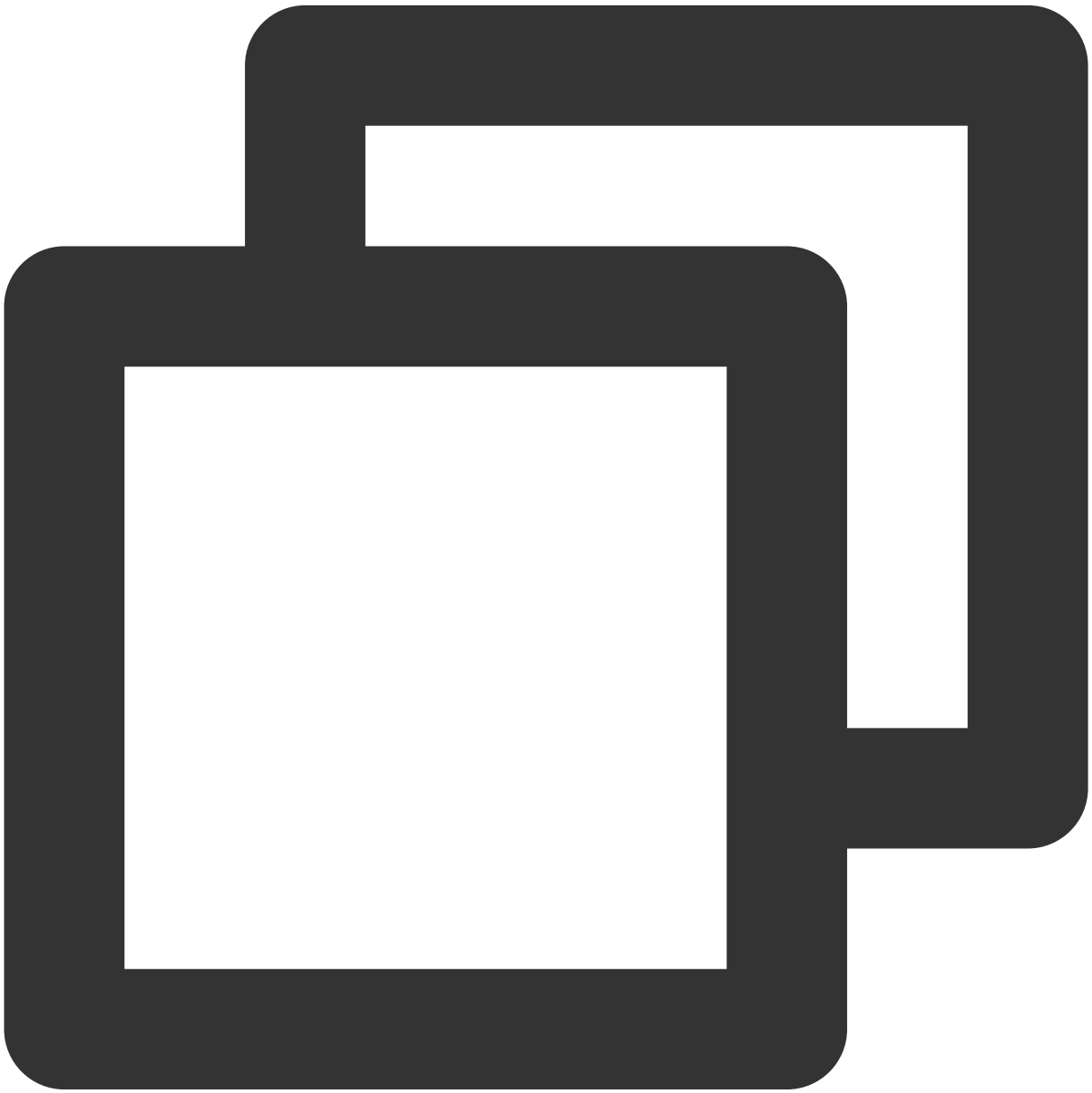


```
// シングルチャットメッセージ履歴の取得
// 初めて取得する場合は、msg_getmsglist_param_last_msgをnullに設定します
// 再度取得する場合は、msg_getmsglist_param_last_msgは、返されたメッセージリストの最後のメッセ
var get_message_list_param = new MsgGetMsgListParam
{
    msg_getmsglist_param_last_msg = LastMessage
};
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
// コールバックロジックを処理します
});
```

より多くのメッセージ履歴を取得するには、メッセージ履歴-Unityをご参照ください。

メール未読数のカウント

ユーザー-システムメッセージの記録は、チャット内のセッションに相当します。Tencent Cloud IMは、セッションの未読数をカウントする機能を提供し、メッセージがまだ読まれていないことをユーザーをリマインドします。ユーザーがセッションに入ってまたセッションリストに戻ると、未読メッセージ数はクリアされます。Unityのサンプルコードは次のとおりです：



```
// すべての未読数を取得する
TIMResult res = TencentIMSDK.ConvGetTotalUnreadMessageCount((int code, string desc,
```

```
 // 非同期処理のロジック
});

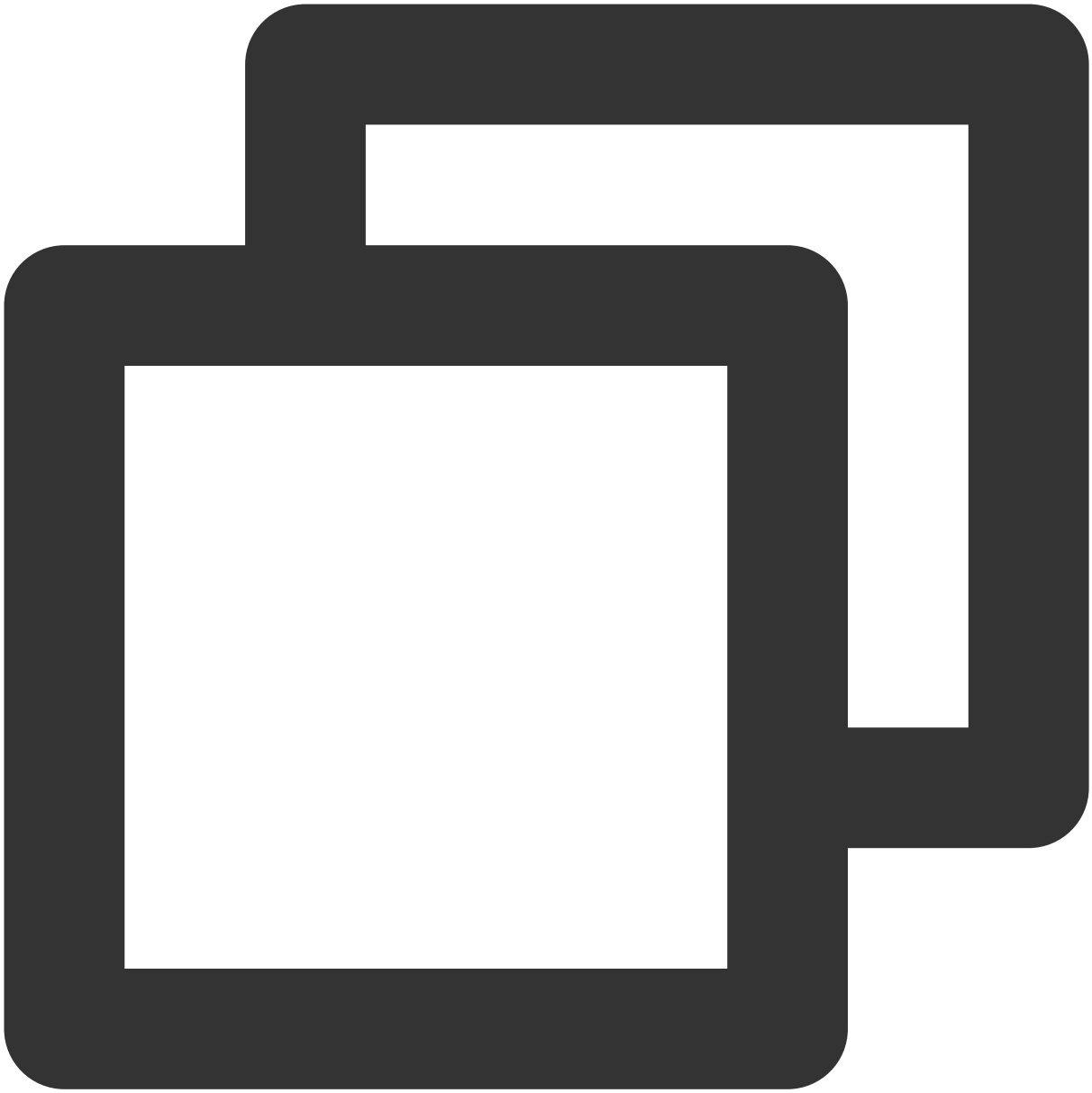
// 未読数変更通知
TencentIMSDK.SetConvTotalUnreadMessageCountChangedCallback((int total_unread_count,
 // コールバックロジックを処理します
});

// すべてのセッションの未読メッセージ数をクリアする
TIMResult res = TencentIMSDK.MsgMarkAllMessageAsRead((int code, string desc, string
 // 非同期処理のロジック
});
```

詳細については、Unity-セッションの未読メッセージ数をご参照ください。

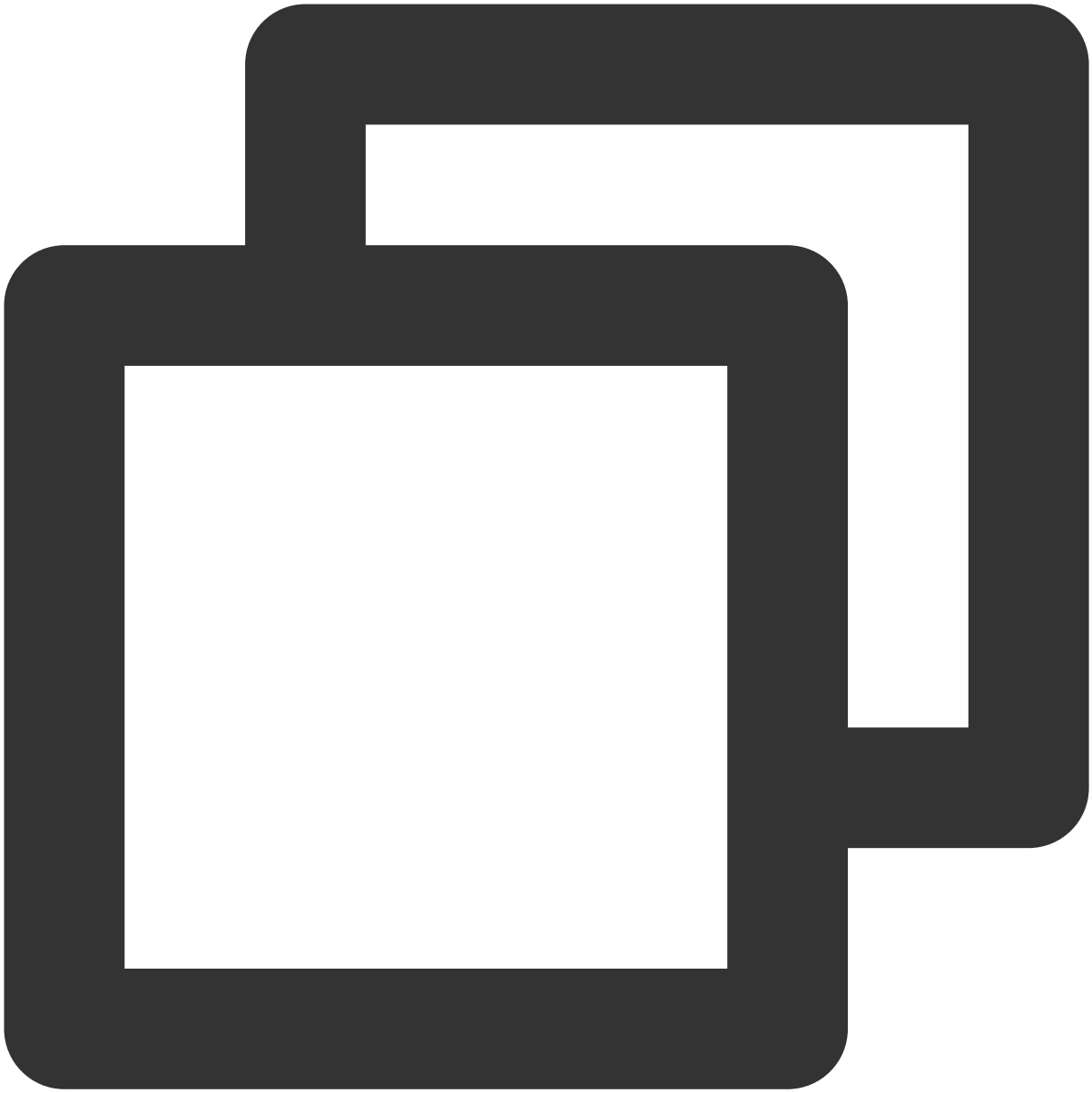
全員メール

全員メールは、ゲーム内のすべてのプレイヤにメールを送信することに相当します。Tencent Cloud IMはサーバー側の 「全員プッシュ」 機能を提供します。サンプルコードは次のとおりです：



```
https://console.tim.qq.com/v4/all_member_push/im_push?usersig=xxx&identifier=admin&
```

リクエストパケットの例は次のとおりです：



```
{
    "From_Account": "admin",
    "MsgRandom": 56512,
    "MsgLifeTime": 120, // オフラインで120秒（2分間）保存
    "MsgBody": [
        {
        "MsgType": "TIMTextElem",
        "MsgContent": {
            "Text": "hi, beauty"
            }
        }
```

```
        ]
    }
```

全員プッシュでは、オフライン時のメッセージ保存期間を設定できます。これにより、ユーザーがオフラインになっても、オフライン保存期間内である限り、メッセージを受信できます。オフライン保存期間を設定するには、`MsgLifeTimé`を秒単位で設定します。最大7日間(604800 s)保存できます。デフォルトは0で、オフライン時に保存しないことを意味します。

全員プッシュの詳細については、全員プッシュをご参照ください。

メールの有効期間

メール履歴の保存期間として、体験版/Standard editionは7日間、Premium editionは30日間です。Standard editionとPremium editionは、保存期間の延長をサポートします。履歴の保存の詳細については、メッセージ履歴保存期間の設定をご参照ください。

また、サーバー側がメッセージを送信する時、リクエストパケットで `MsgLifeTime` を設定することで、メッセージのオフライン保存期間(最大7日間)を設定できます。このフィールドが0の場合、メッセージはオンラインユーザーのみに送信され、オフラインでは保存されません。詳細については、サーバーAPIをご参照ください。

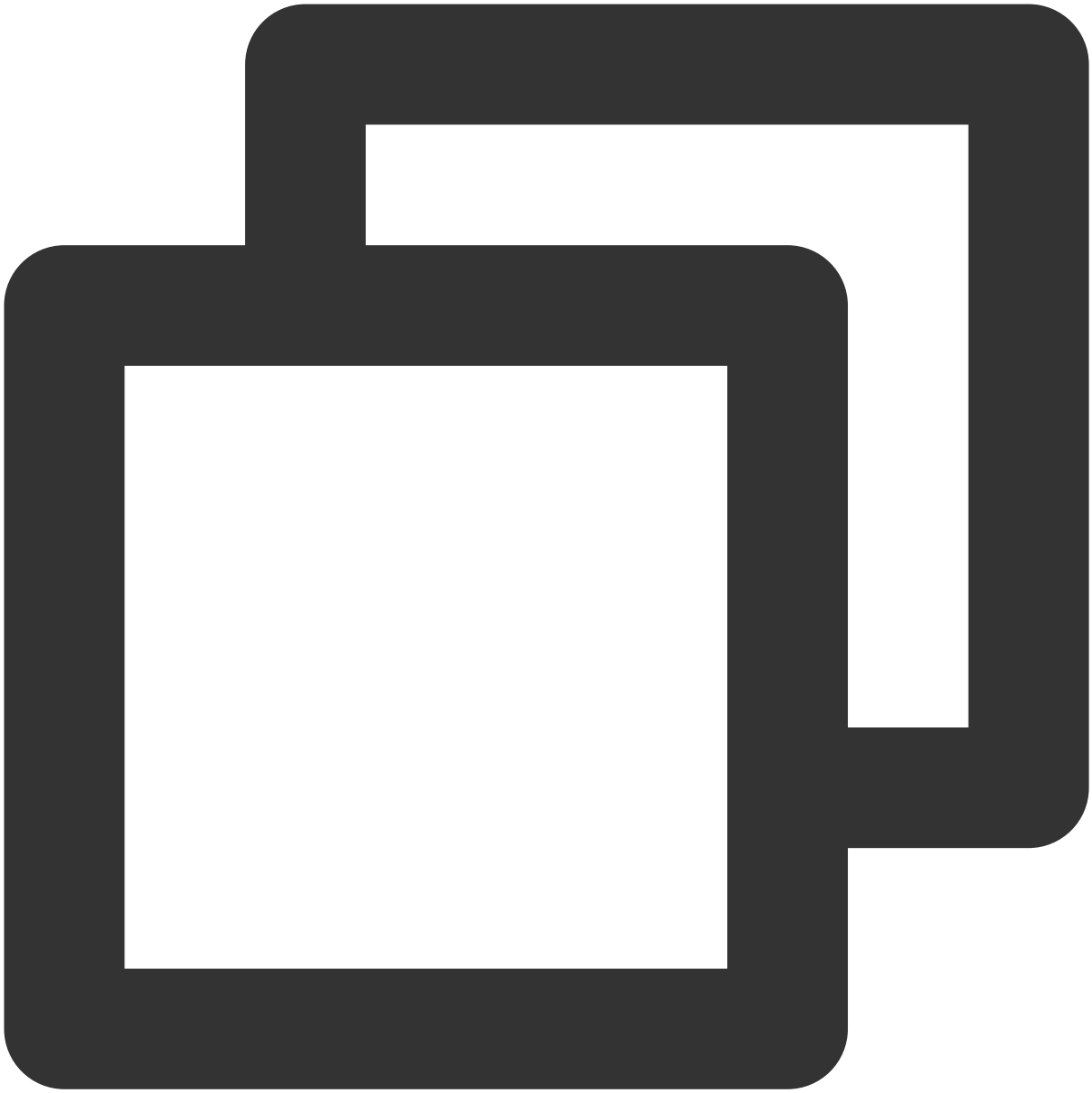
一時的なチーム作成

マルチプレイヤーオンラインゲームでは、一時的なチーム作成が不可欠です。以下では、チーム作成シーン、バックグラウンド、チーム内メンバがそれぞれ必要とするチーム情報から、一時的なチーム作成を説明します。

チーム作成シーン

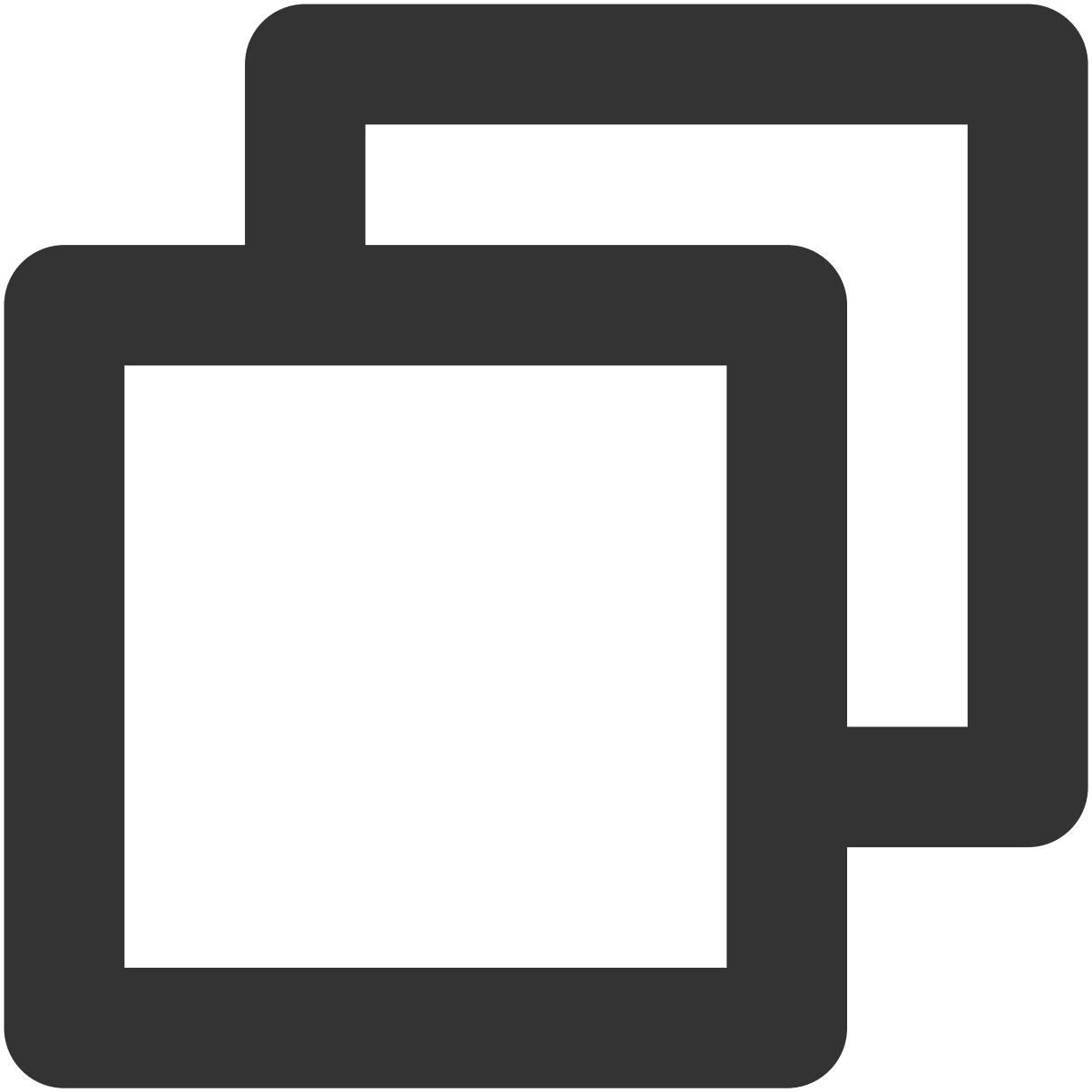
チーム作成において、チーム作成、一時的なチーム作成から退出、キャプテンになる、チームへ招待、チーム解散などの運用シーンがよく見られます。以下では、いくつかのサンプルコードを挙げて、異なる運用シーンの実装方法を説明します。

ゲーム開始前のチーム作成：最初のメンバーがゲームに入ると、サーバーでグループを自動的に作成し、最大グループメンバー数を設定します。リクエスト時にグループまたはグループメンバーを指定した場合、作成時にグループマスターまたはグループメンバーは自動的にグループに参加します。リクエストURLの例は次のとおりです。



```
https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkappid=88888888&id
```

リクエストパケットの基本形式の例は次のとおりです：

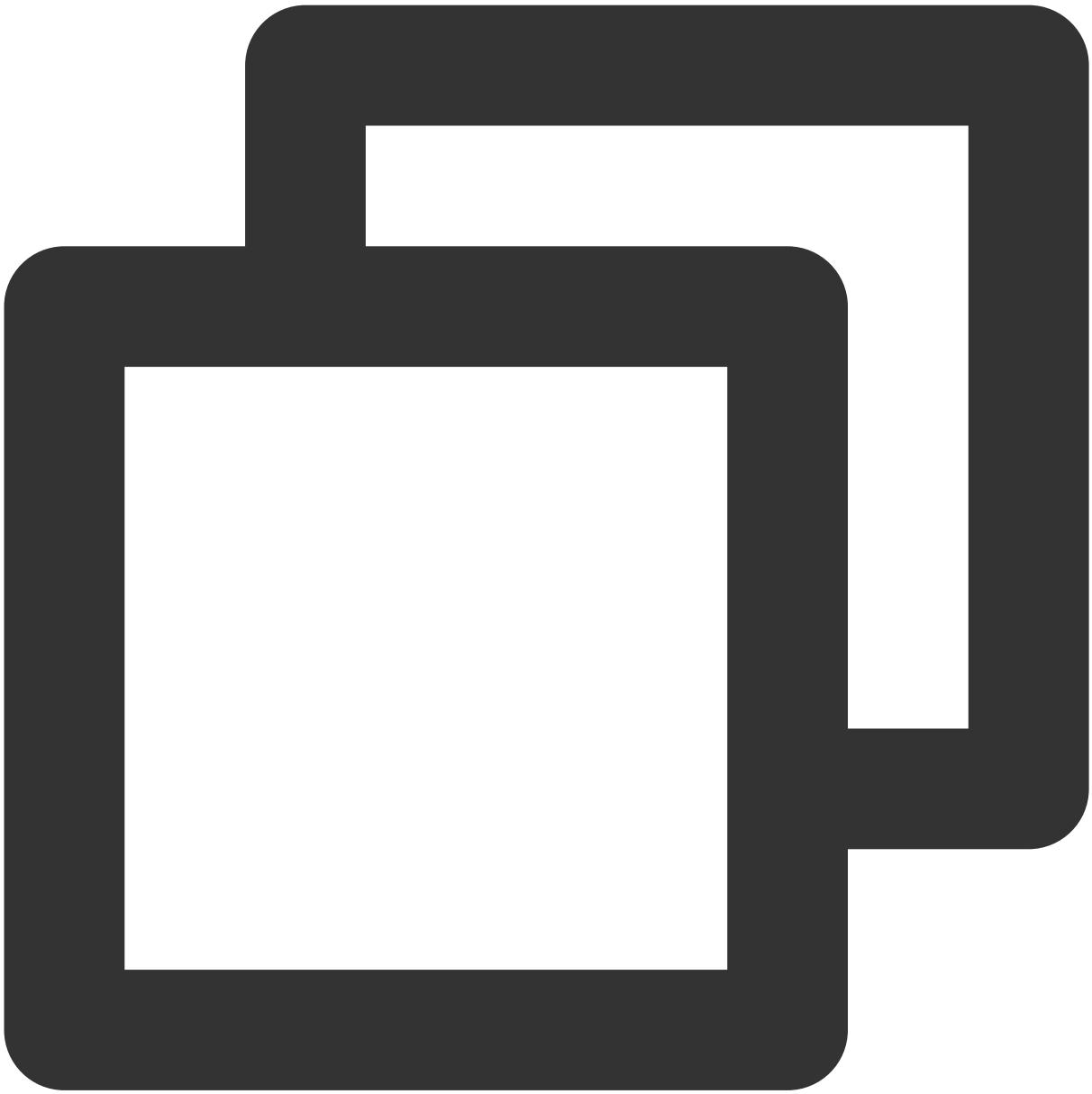


```
{
"Owner_Account": "leckie", // グループマスターのUserId（オプション）
"Type": "Public", // グループタイプ：Private/Public/ChatRoom/AVChatRoom/Community
"Name": "TestGroup", // グループ名（必須）
"MaxMemberCount":5 // 最大グループメンバー数（オプション）
}
```

説明：

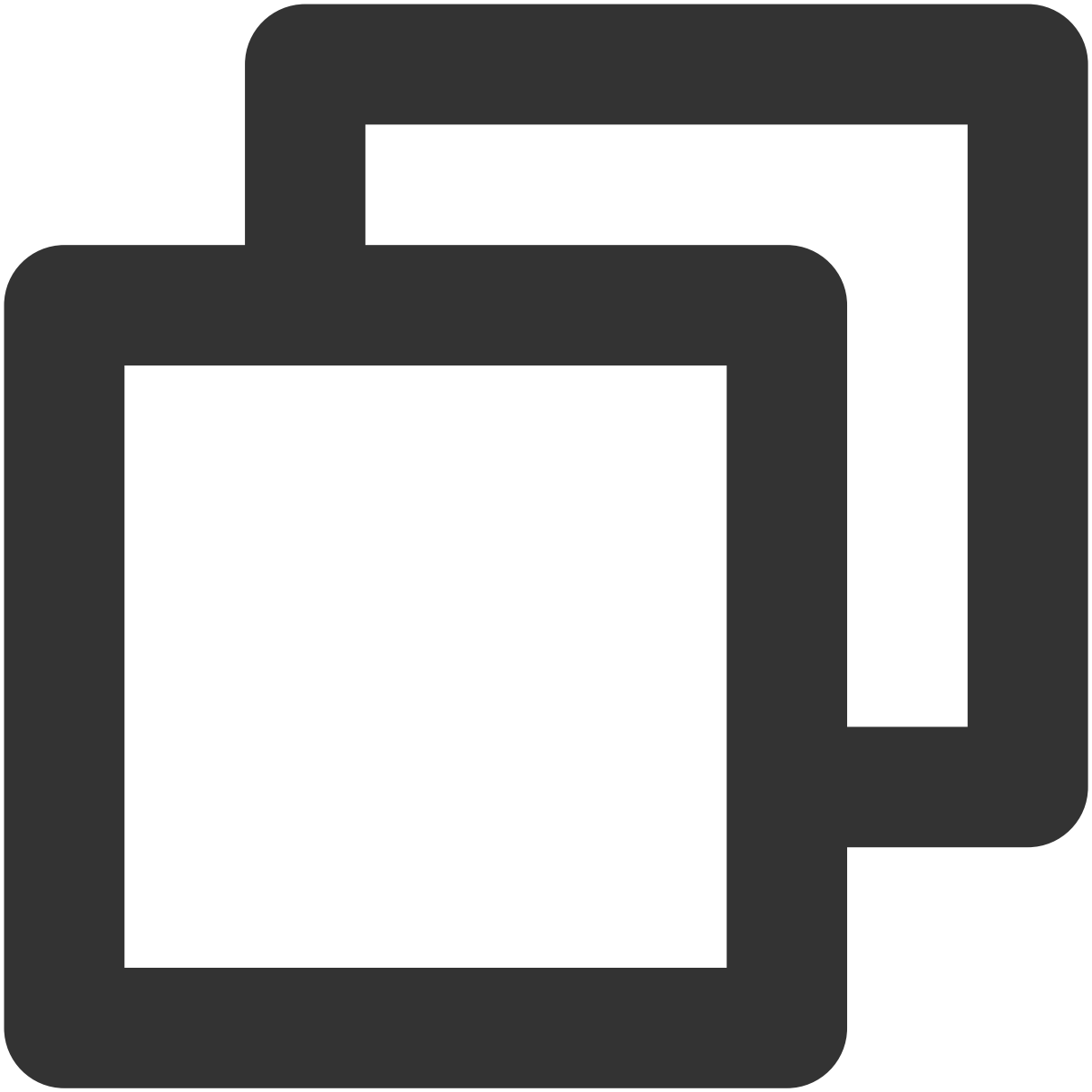
Appに同時に存在するグループは10万までです。10万を超えると、料金が請求されます。詳細については、価格説明をご参照ください。サーバーでグループを作成する詳細については、グループ作成をご参照ください。

グループメンバーの追加：グループチャットを作成した後、新しいプレイヤーが続々とゲームに入る場合、グループに新しいグループメンバーを追加する必要があります。リクエストURLの例は次のとおりです：



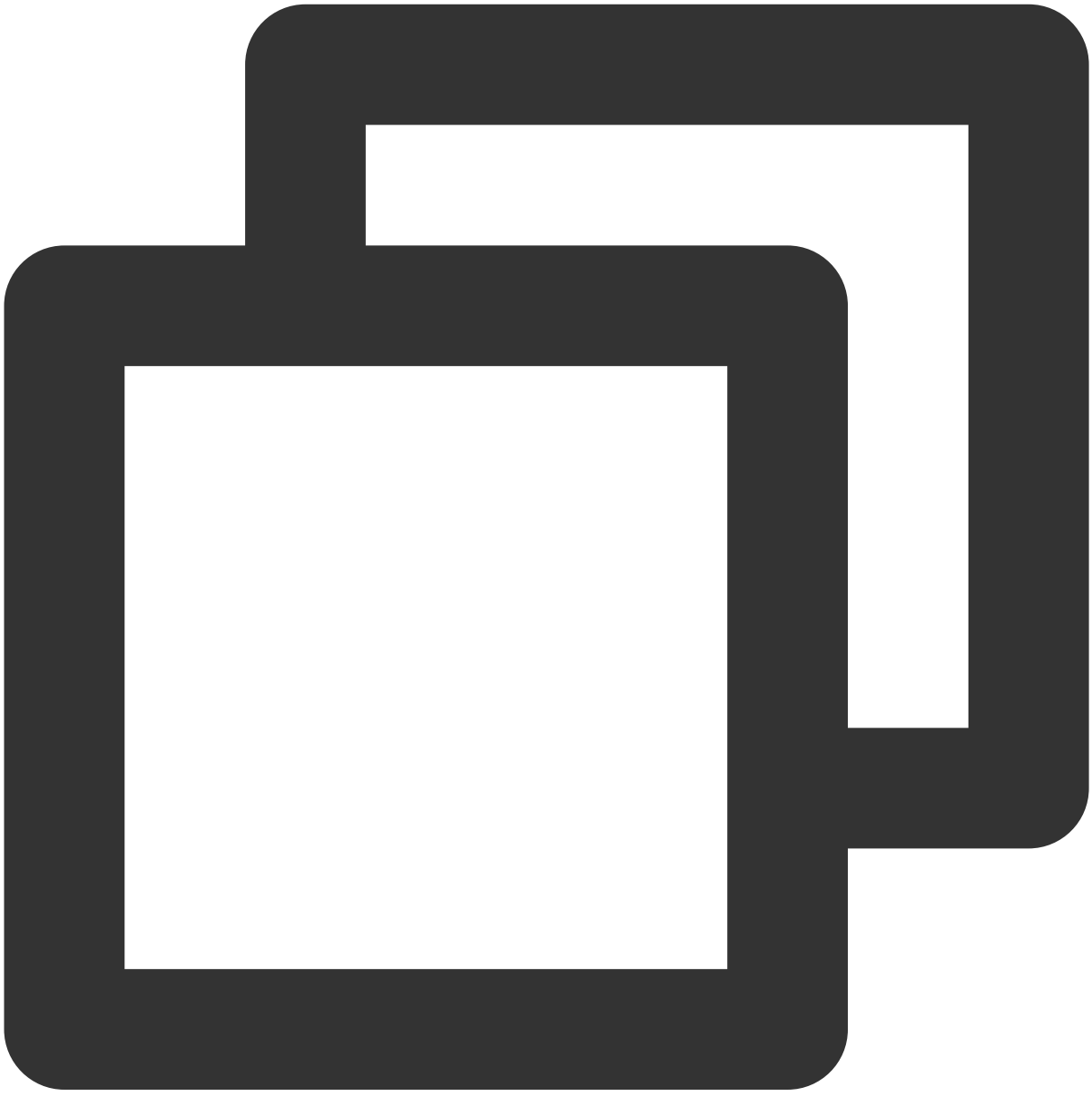
```
https://console.tim.qq.com/v4/group_open_http_svc/add_group_member?sdkappid=8888888
```

リクエストパケットは次のとおりです：



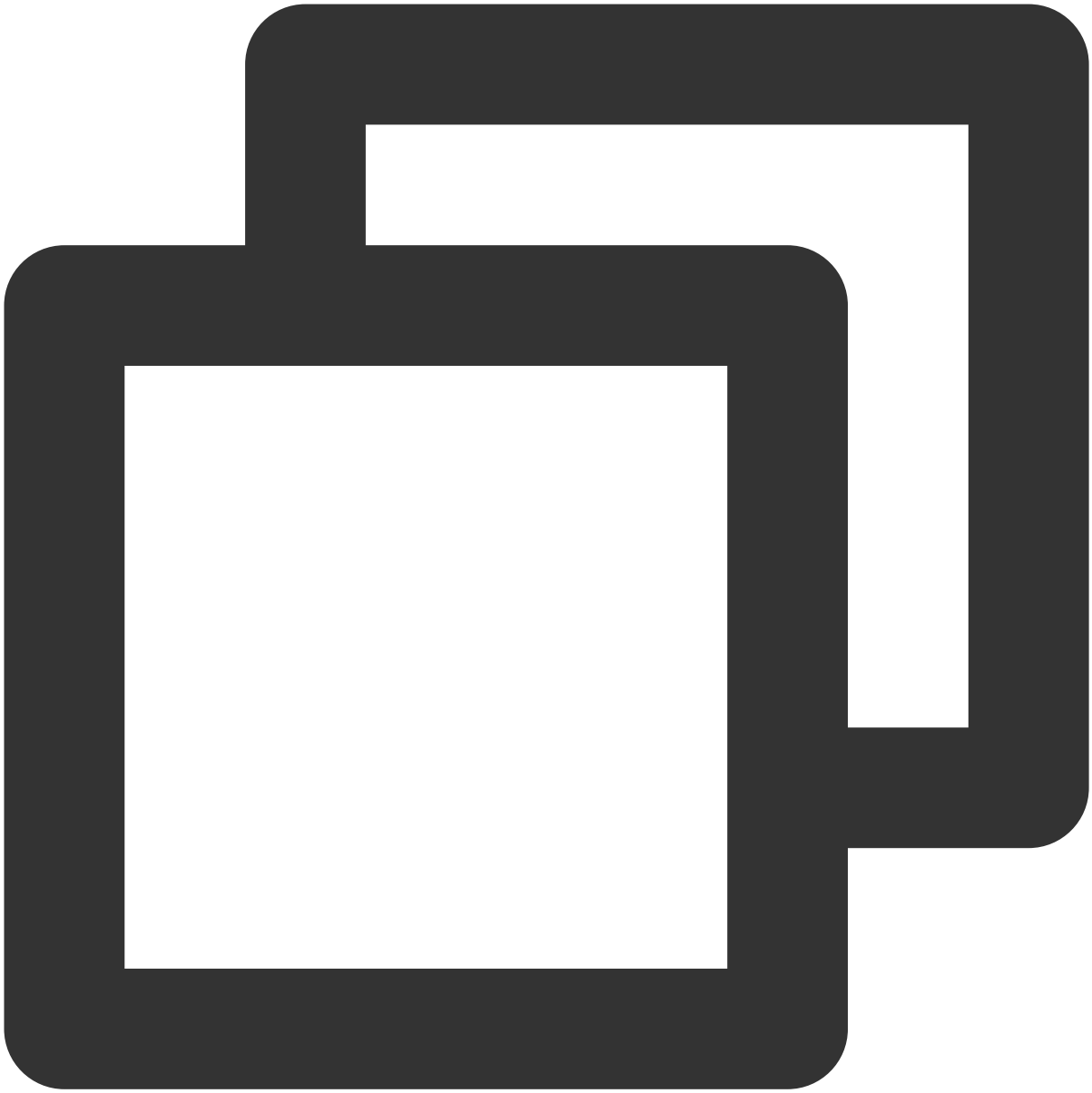
```
{
"GroupId": "@TGS#2J4SZEAEL", // 操作するグループ（必須）
"MemberList": [ // 1回で最大300メンバーを追加できる
    {
        "Member_Account": "tommy" // 追加するグループメンバーのID（必須）
    },
    {
        "Member_Account": "jared"
}]
}
```

詳しくは、グループメンバーの追加のドキュメントをご参照ください。
チーム作成成功コールバック：グループを作成する時に最大グループメンバー数を設定した場合、チームに必要なメンバーが揃っていると、ゲームを開始できます。 `グループ満員後群のコールバック` を呼び出して、ゲームを開始できるかを確認できます。リクエストURLの例は次のとおりです：



```
https://www.example.com?SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&content
```

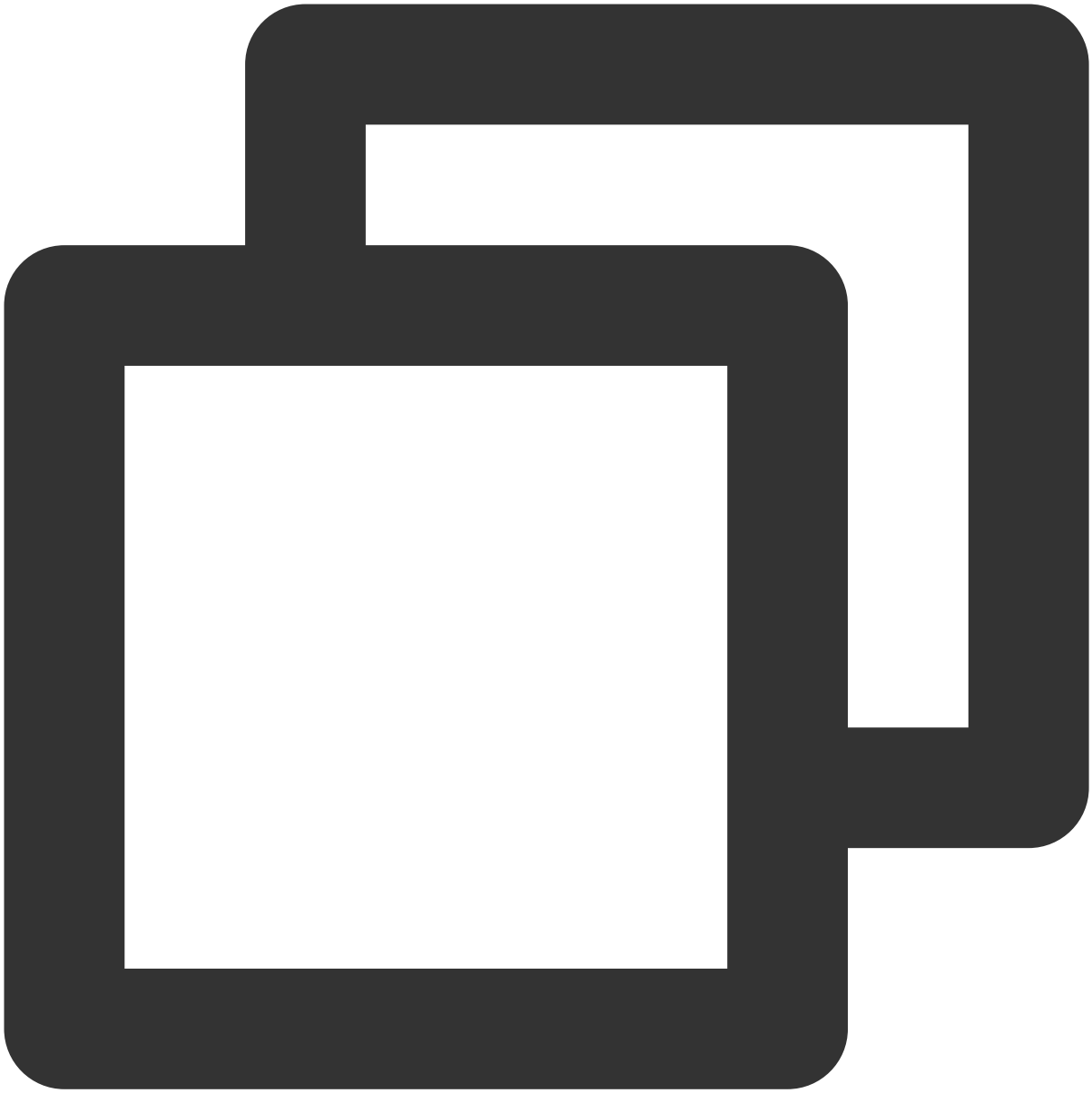
リクエストパケットの例は次のとおりです：



```
{
"CallbackCommand": "Group.CallbackAfterGroupFull", // コールバックコマンド
"GroupId": "@TGS#2J4SZEAEL" // グループID
}
```

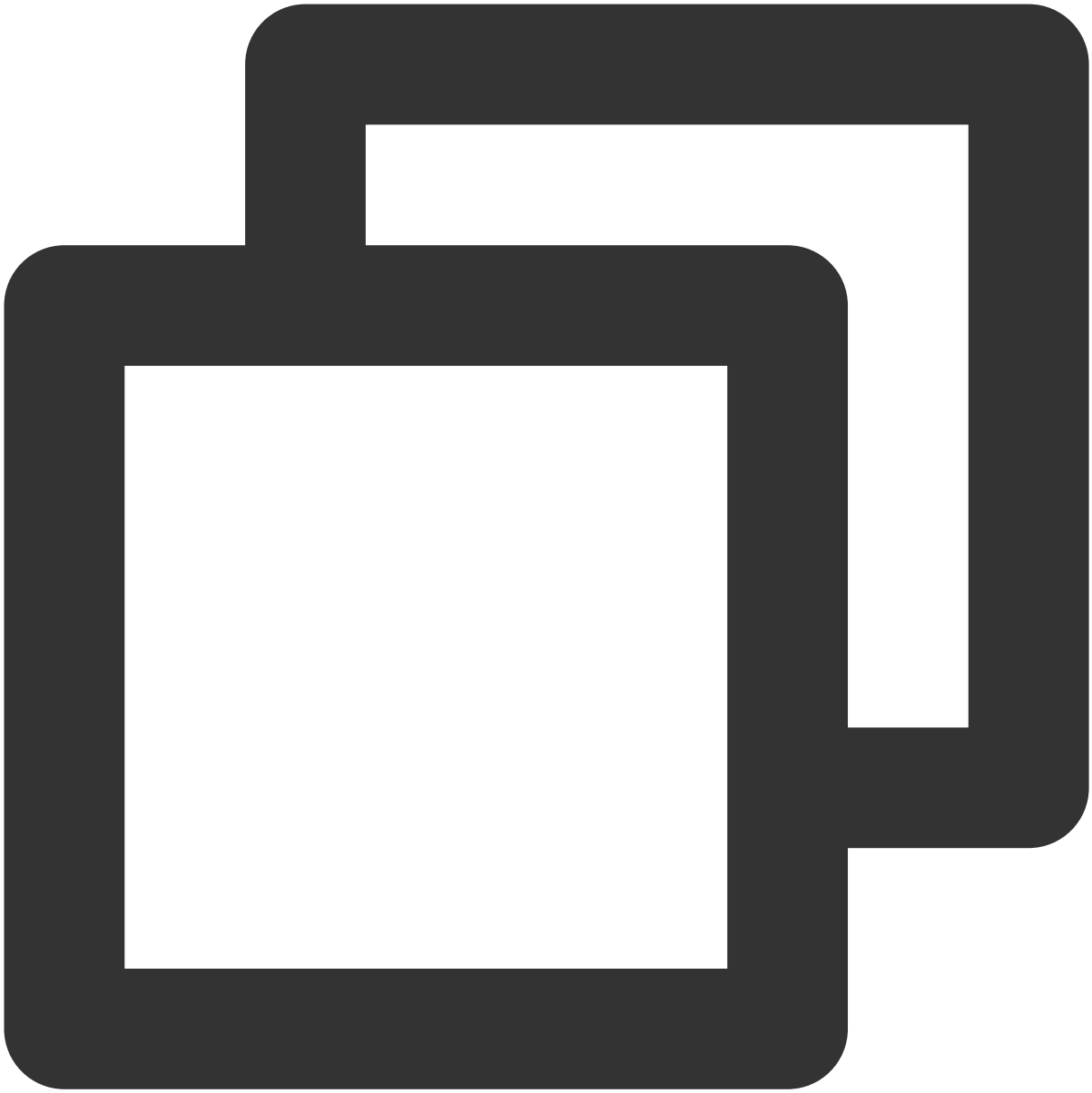
詳しくは、グループ満員後のコールバックをご参照ください。

新しいメンバーがグループに参加した通知：新しいプレイヤーがゲーム（グループチャット）に入ると、`新しいメンバーがグループに参加したコールバック`を呼び出して、その他のグループメンバーに通知します。リクエストURLの例は次のとおりです：



```
https://www.example.com?SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&content
```

リクエストパケットの例は次のとおりです：



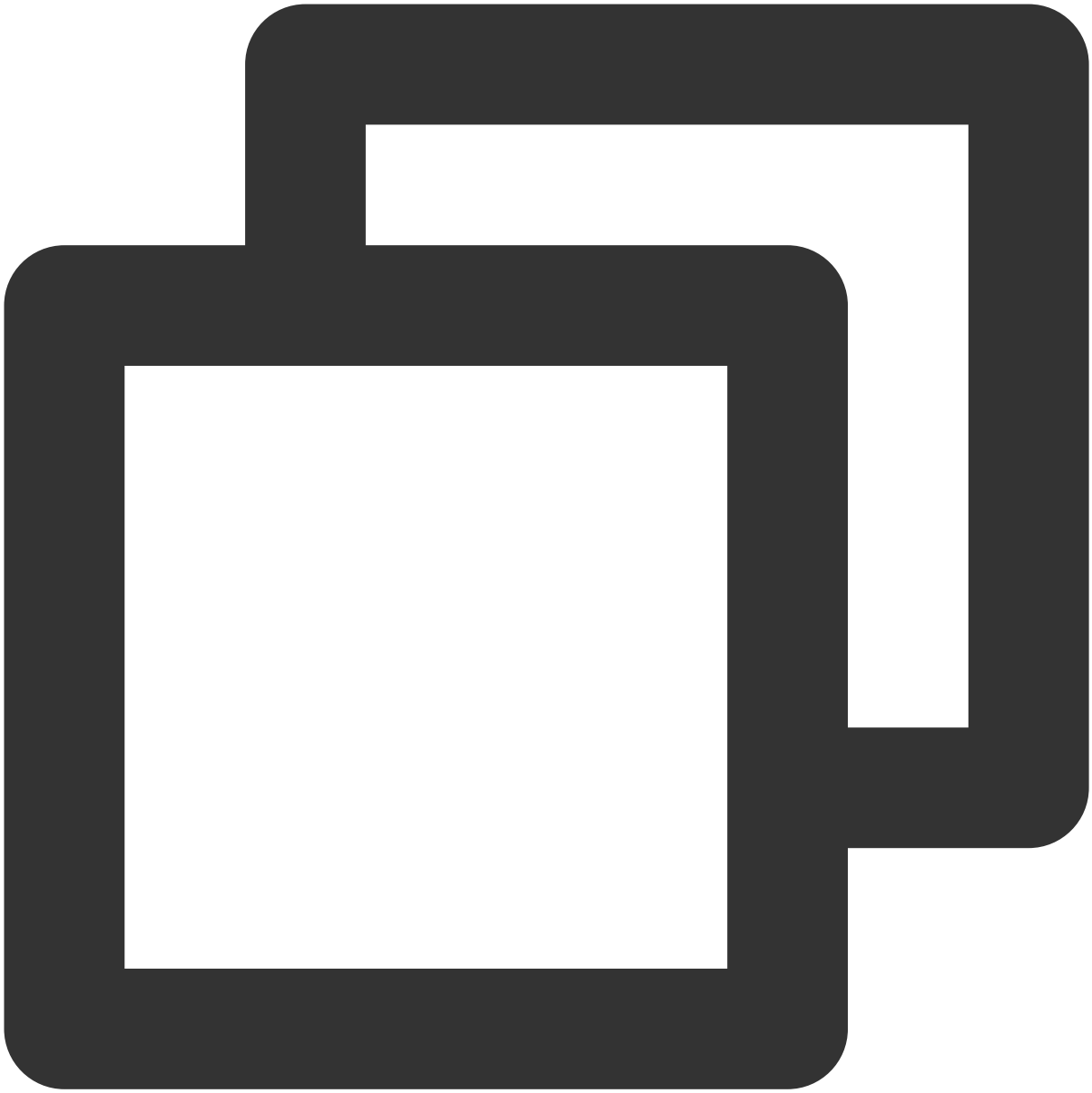
```
{
"CallbackCommand": "Group.CallbackAfterNewMemberJoin", // コールバックコマンド
GroupId": "@TGS#2J4SZEAEL",
"Type": "Public", // グループタイプ
"JoinType": "Apply", // グループへの参加方法：Apply（グループへの参加を申請）、Invited（招待
"Operator_Account": "leckie", // 操作者メンバー
"NewMemberList": [ // グループに参加した新しいメンバーのリスト
    {
        "Member_Account": "jared"
    },
    {
```

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

説明：

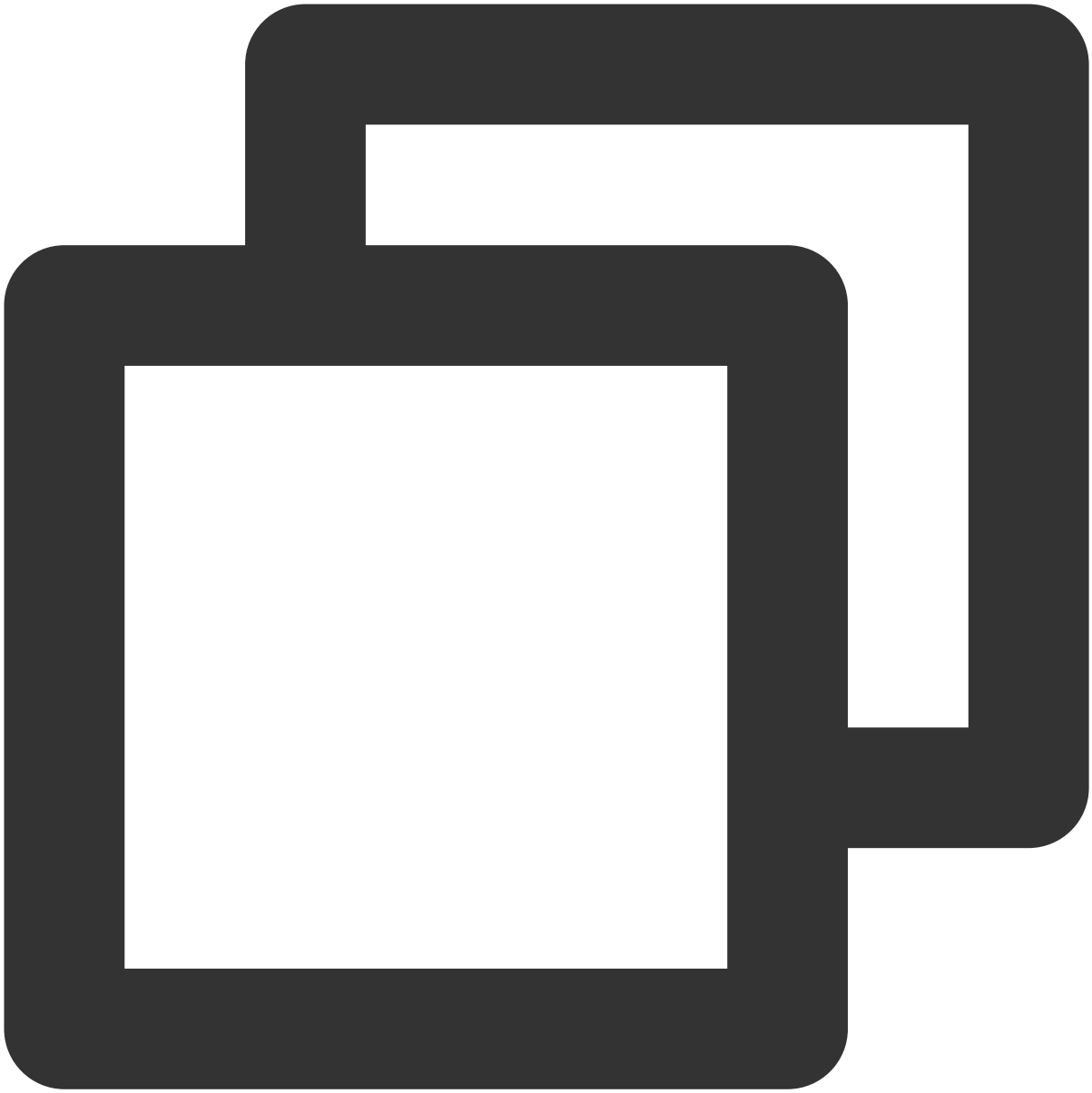
コールバックを有効にするには、コールバックURLを設定し、このコールバックプロトコルに対応するスイッチをオンにする必要があります。設定方法については、サードパーティコールバックの設定ファイルをご参照ください。詳しくは、新しいメンバーがグループに参加したコールバックをご参照ください。

ゲーム中にグループを退出：プレイヤーが自発的ゲームを退出、または、ネットワークが原因でゲームを退出された場合、サーバーで `グループメンバーが退出したコールバック` を呼び出し、グループ内のその他のプレイヤーに通知したり、ネットワークが原因でゲームを退出されたプレイヤーにメッセージを送信したりできます。グループメンバーが退出したコールバックのリクエストURLの例は次のとおりです。



```
https://www.example.com?SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&content
```

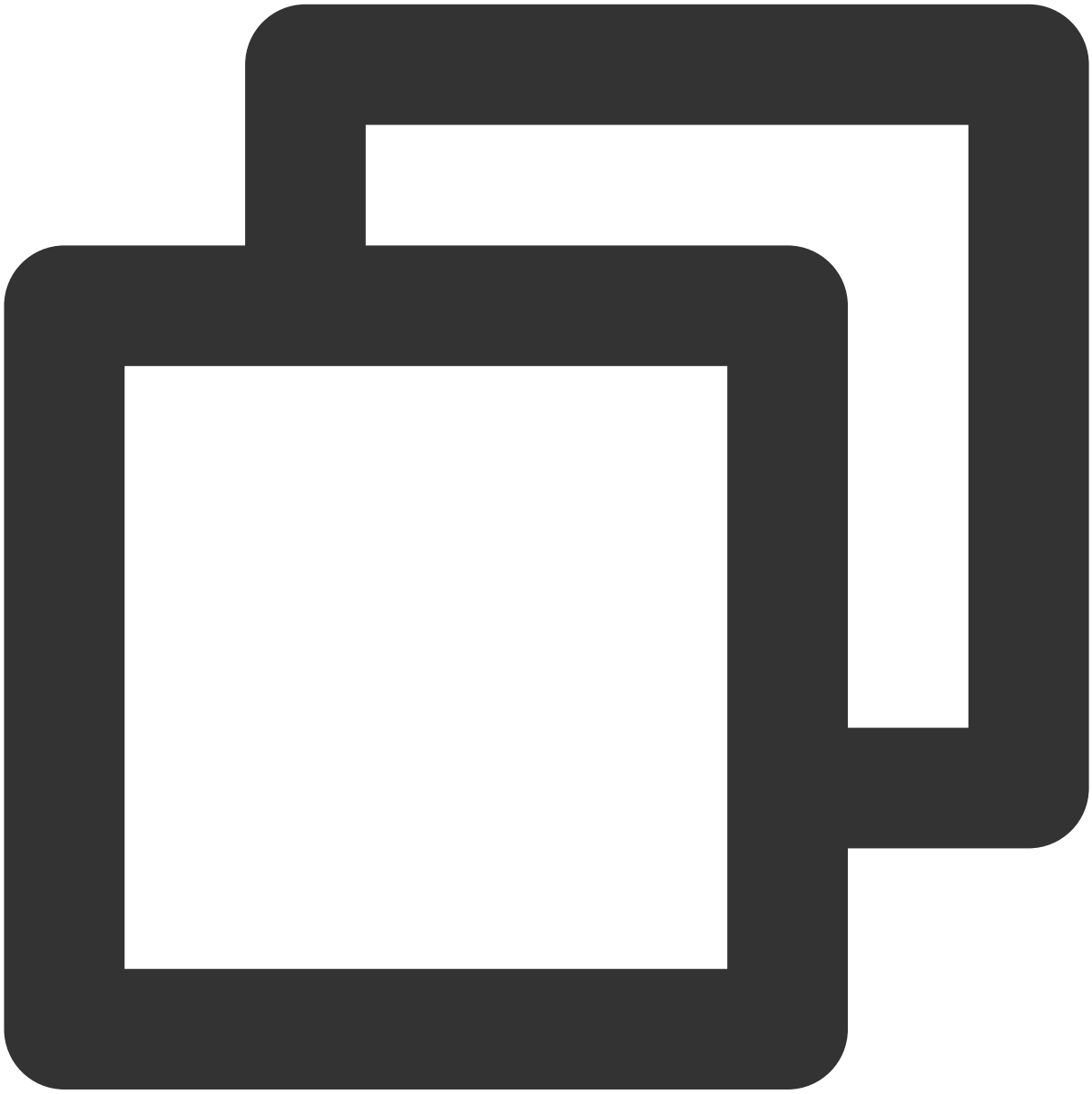
リクエストパケットの例は次のとおりです：



```
{
"CallbackCommand": "Group.CallbackAfterMemberExit", // コールバックコマンド
"GroupId": "@TGS#2J4SZEAEL", // グループID
"Type": "Public", // グループタイプ
"ExitType": "Kicked", // メンバーの退出の形：Kicked-強制退出、Quit-自分がグループから退出
"Operator_Account": "leckie", // 操作者
"ExitMemberList": [ // グループを退出したメンバーのリスト
    {
        "Member_Account": "jared"
    },
    {
```

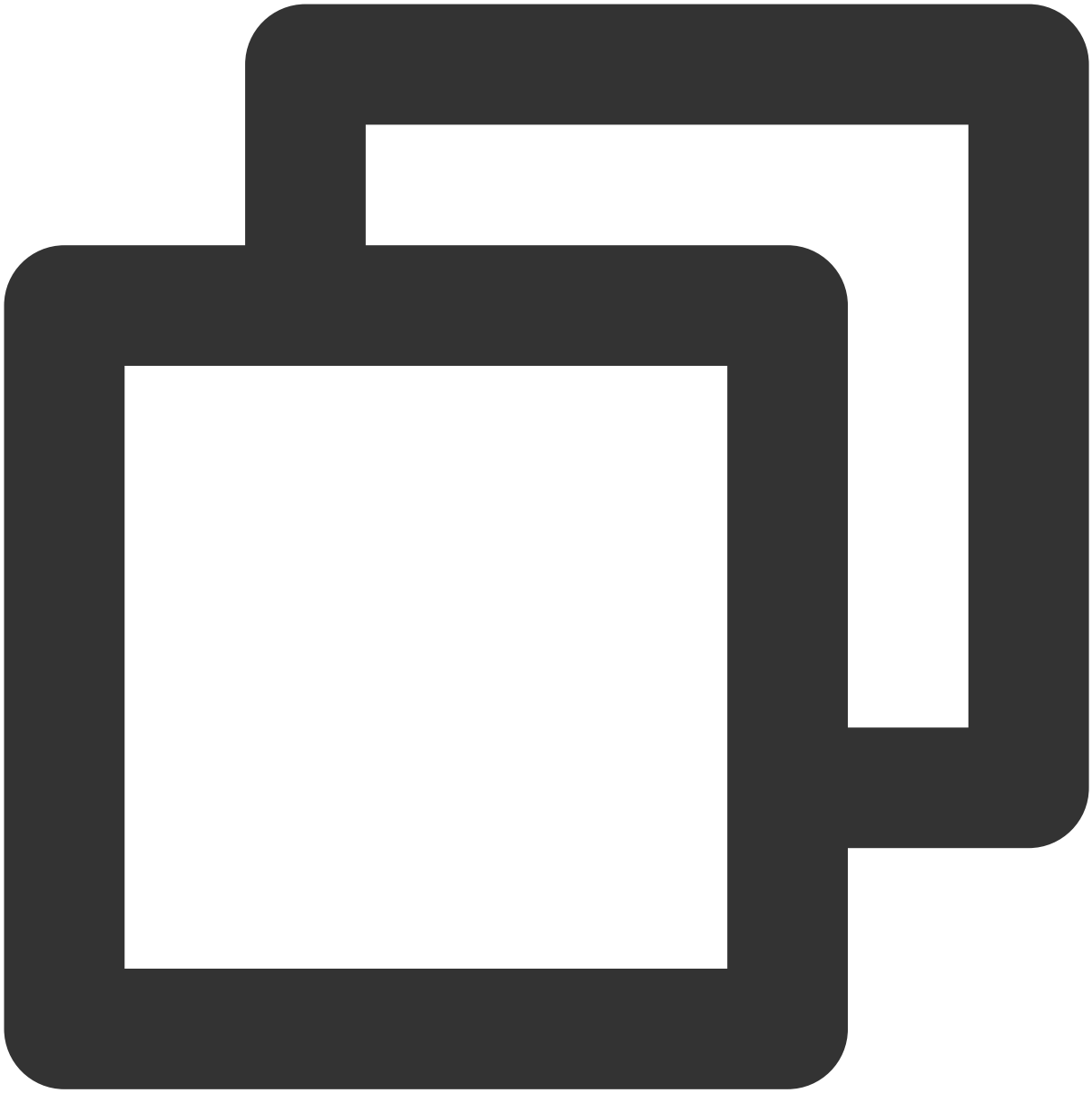
```
            "Member_Account": "tommy"
        }
    ]
    }
```

ゲーム終了後にグループチャットを解散：ゲーム終了後に、サーバーでグループチャットをそのまま解散できます。グループチャットの解散のリクエストURLの例は次のとおりです：



```
https://console.tim.qq.com/v4/group_open_http_svc/destroy_group?sdkappid=88888888&i
```

リクエストパケットの例は次のとおりです：

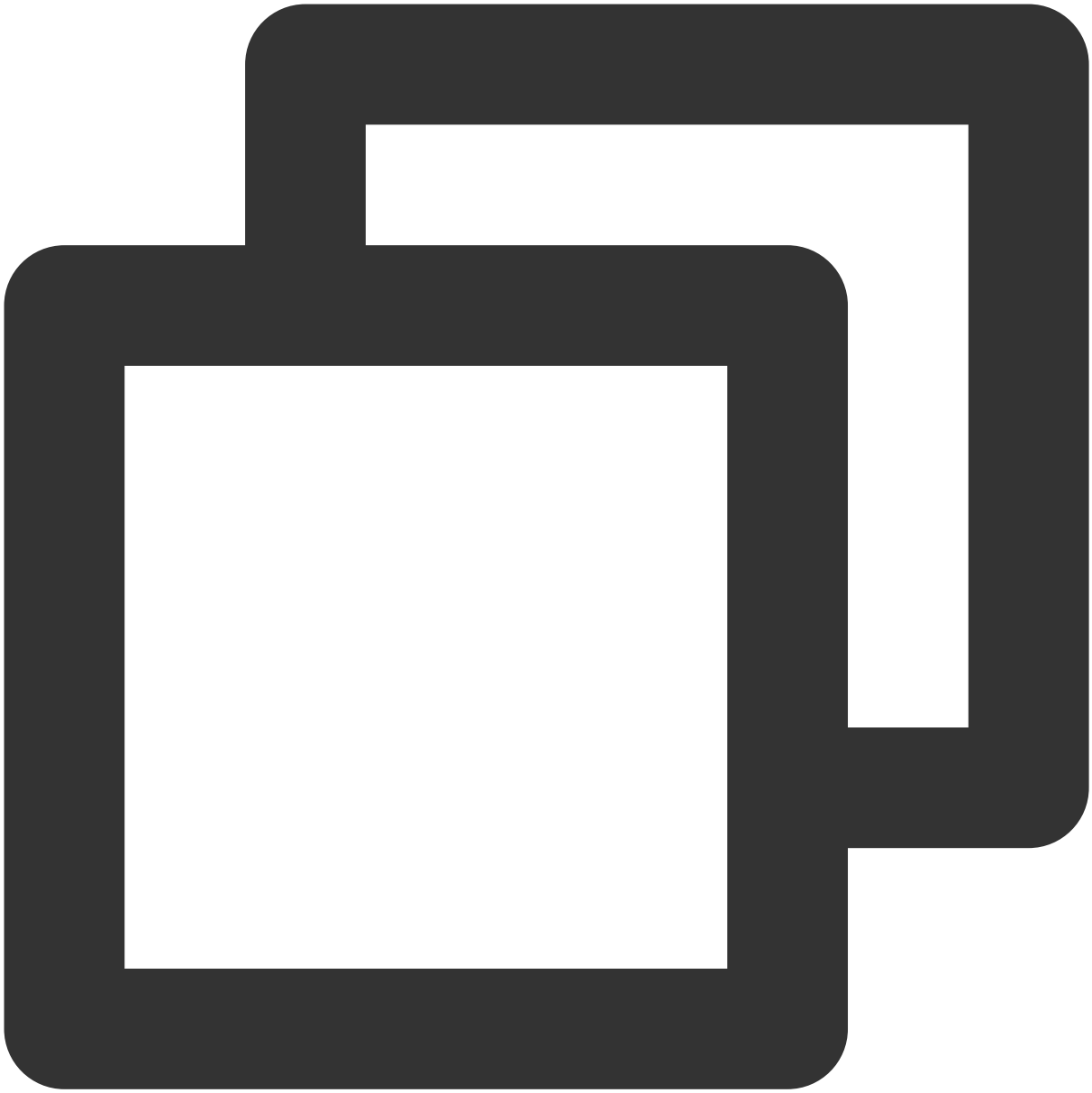


```
{
"GroupId": "@TGS#2J4SZEAEL"
}
```

一時的に作成したチーム内の音声チャットやビデオチャットは、状況が複雑なため、以下で詳しく説明します。

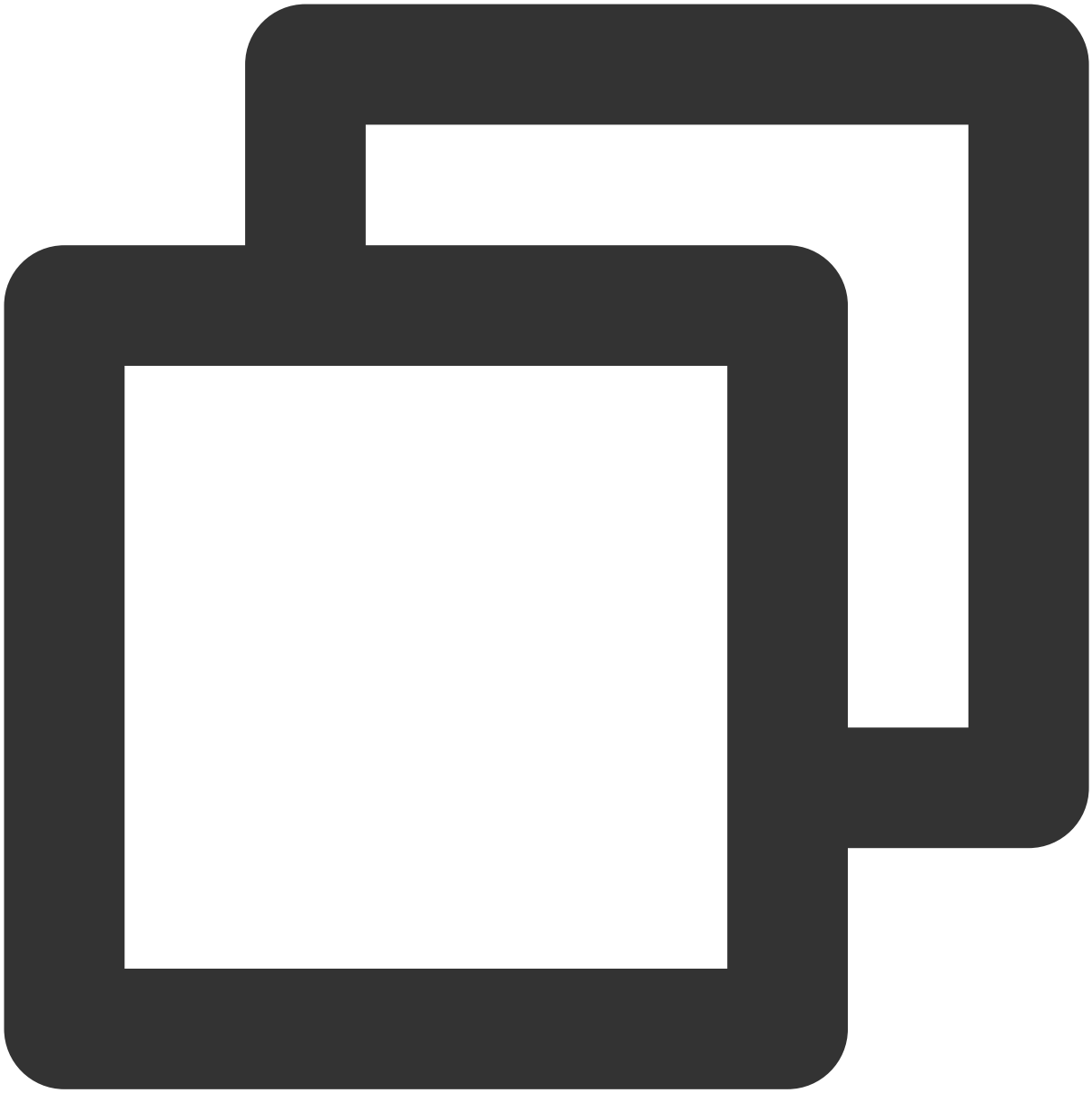
バックグラウンドでチーム内情報の取得

グループ内の詳細情報を取得：グループ内の詳細情報（現在のグループメンバー数、グループメンバーの基本情報などを含む）は、サーバー側のAPI `グループ詳細情報の取得` を使用して取得できます。リクエストパケットの中で、Filterフィールドを設定してトップに表示した情報を取得できます。サンプルコードは次のとおりです：



```
https://console.tim.qq.com/v4/group_open_http_svc/get_group_info?sdkappid=88888888&
```

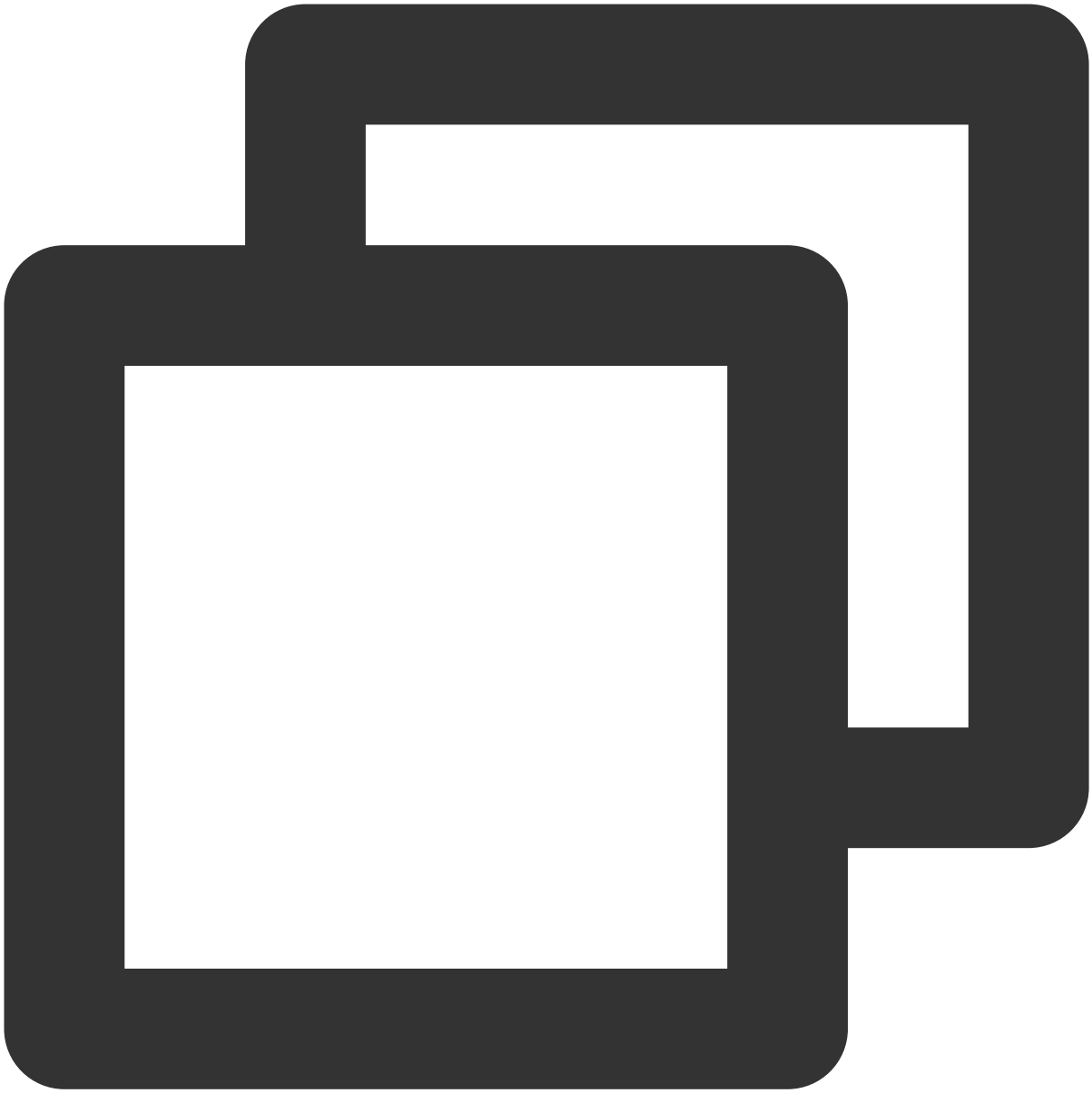
リクエストパケットの例は次のとおりです：



```
{
"GroupIdList": [ // グループリスト（必須）
    "@TGS#1NVTZEAE4",
    "@TGS#1CXTZEAET"
]
}
```

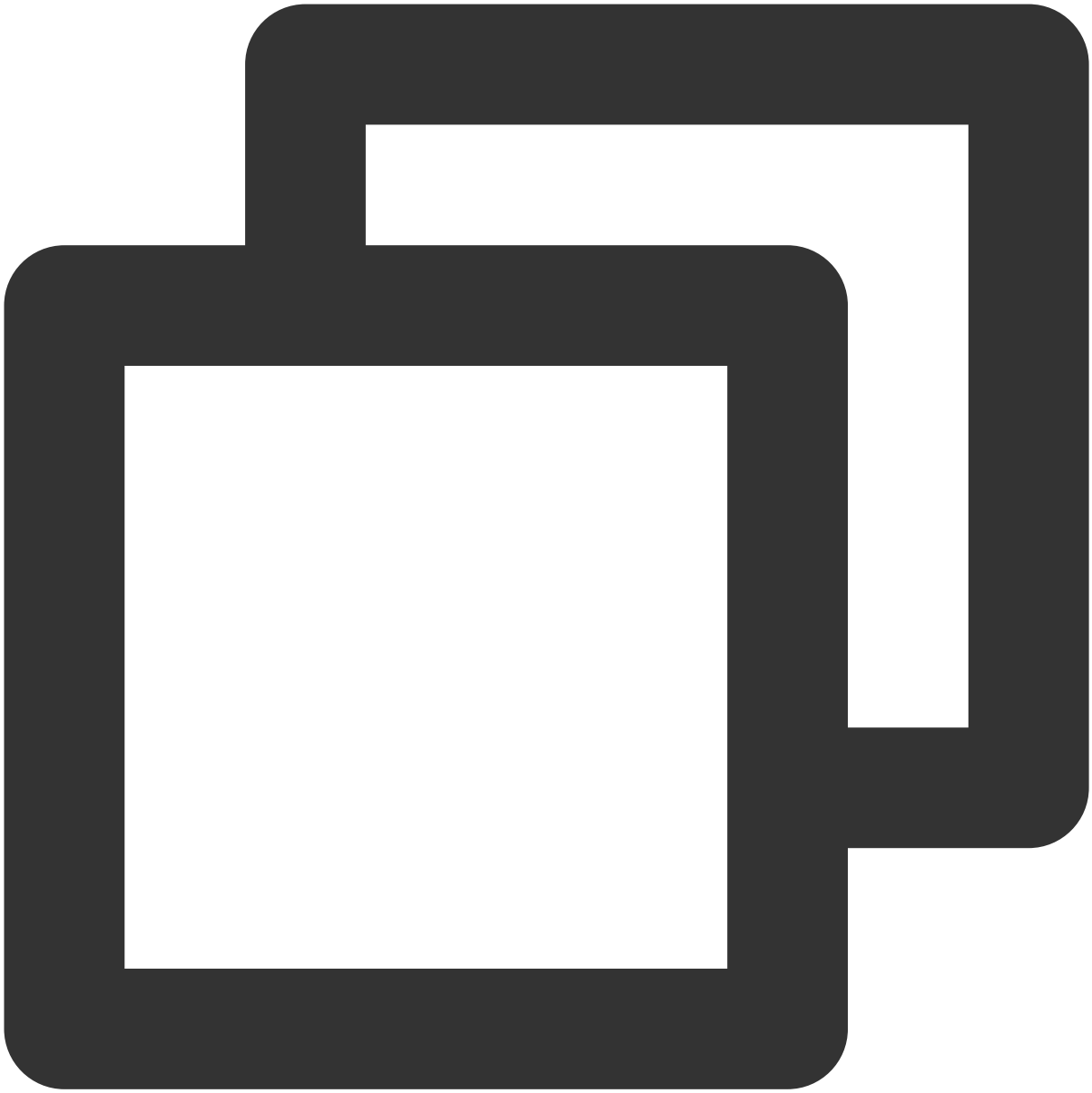
詳細については、グループ詳細情報の取得をご参照ください。

グループメンバーの詳細情報を取得：グループメンバーの情報（カスタムフィールドを含む）のみが必要な場合、 グループメンバー詳細情報の取得 を使用して取得できます。サンプルコードは次のとおりです：



```
https://console.tim.qq.com/v4/group_open_http_svc/get_group_member_info?sdkappid=88
```

リクエストパケットの例は次のとおりです：



```
{
"GroupId":"@TGS#1NVTZEAE4"   // グループID（必須）
}
```

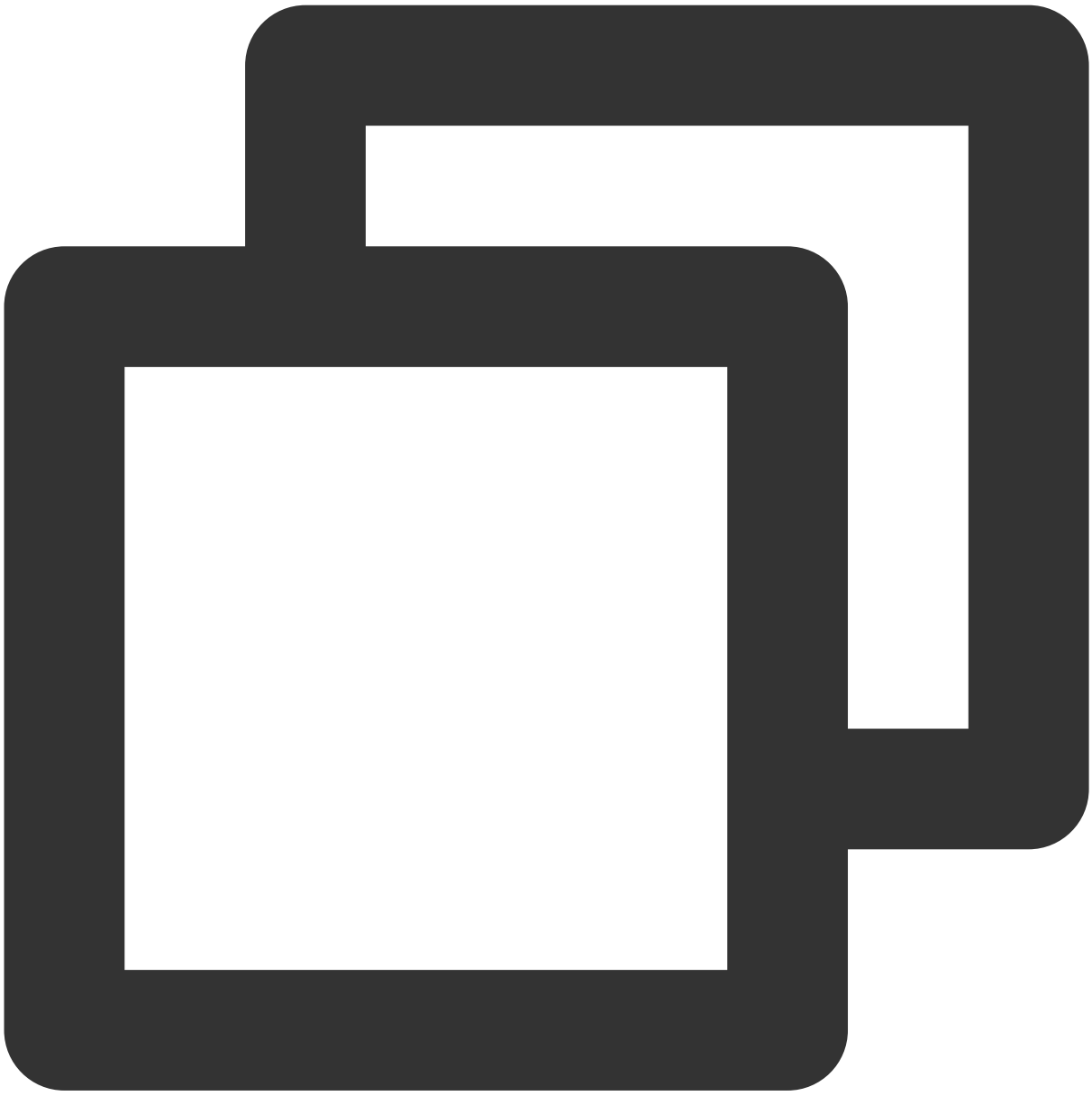
返却されたフィールドの中で、`MemberNum` からこのグループのグループメンバー数、`AppMemberDefinedData` からグループメンバーのカスタムフィールド情報を取得できます。

説明：

このインターフェースはライブストリーミンググループ（AVChatRoom）をサポートしません。詳細については、グループメンバー詳細情報の取得をご参照ください。

チームメンバーのチーム情報取得

—チームメンバーは、 `グループメンバープロフィールの取得` を使用し、その他のチームメンバーのプロフィール、例えば、メンバーのロール、待機状態などを取得できます。サンプルコードは次のとおりです：



```
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,
 // 非同期処理のロジック
});
```

説明：

詳細については、Unity-グループメンバープロフィールの取得をご参照ください。

`グループ満員でゲームが開始します`、`メンバーがグループに参加/退出しました`など、その他のグループ情報は、チームシーンを参照し、コールバックを呼び出してグループに通知できます。

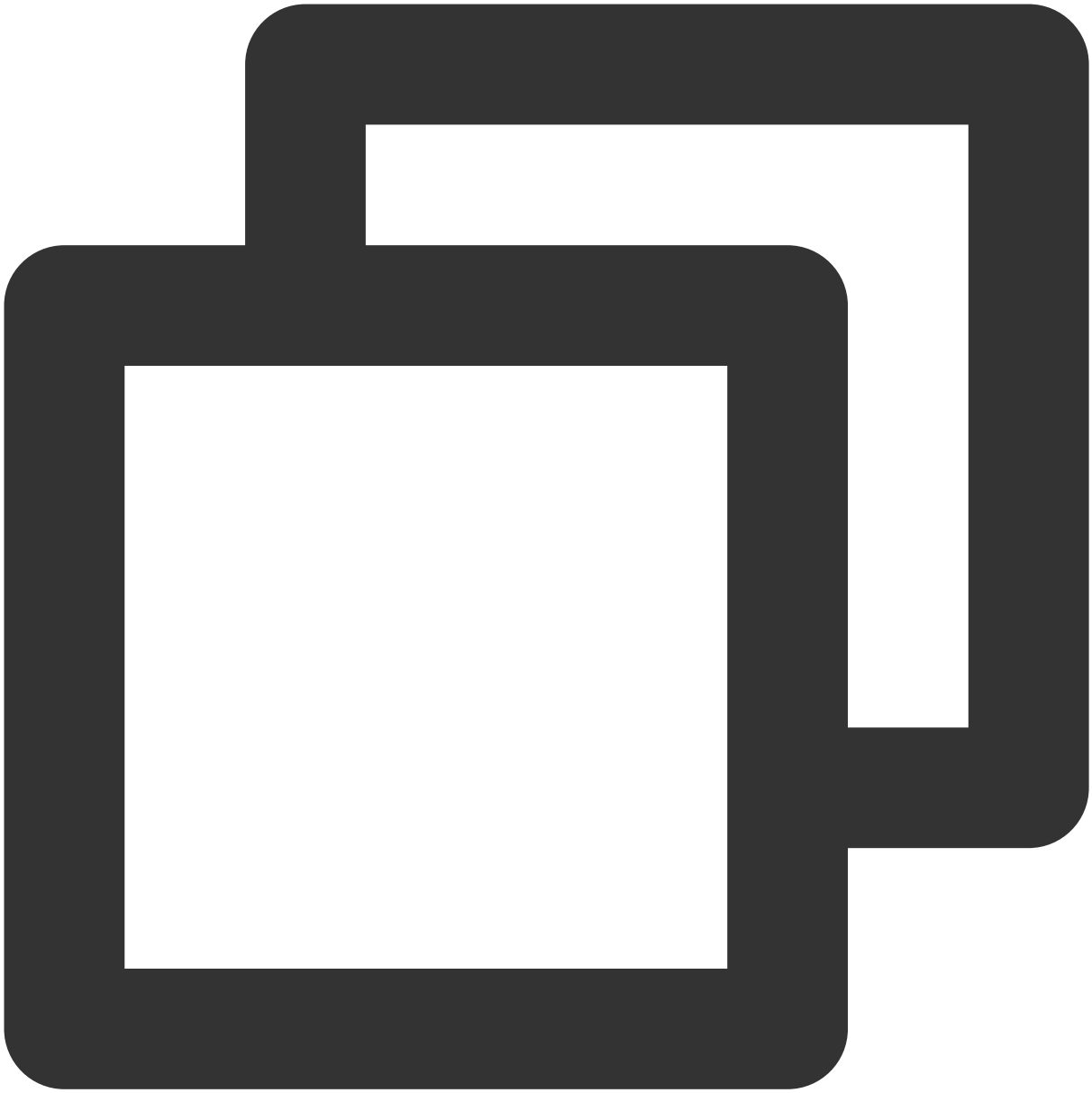
グループの選択などグループの詳細については、グループシステムをご参照ください。**グループ管理コンソールガイド**の詳細については、グループ管理コンソールガイドをご参照ください。

センシティブ情報のブロック

センシティブなコンテンツのブロックはゲームのインスタントメッセージングにおいて非常に重要な機能であり、その実装方法は次のとおりです：

1. ユーザーグループにメッセージ送信前コールバックをバインドします
2. コールバックデータからメッセージタイプを判断し、メッセージデータを天御またはその他のサードパーティチェックサービスに転送します
3. メッセージが一般テキストメッセージの場合、天御のチェック結果が返されてから、メッセージを送信するかのパケットをIMバックグラウンドに返します

メッセージ送信前コールバックデータの例です。



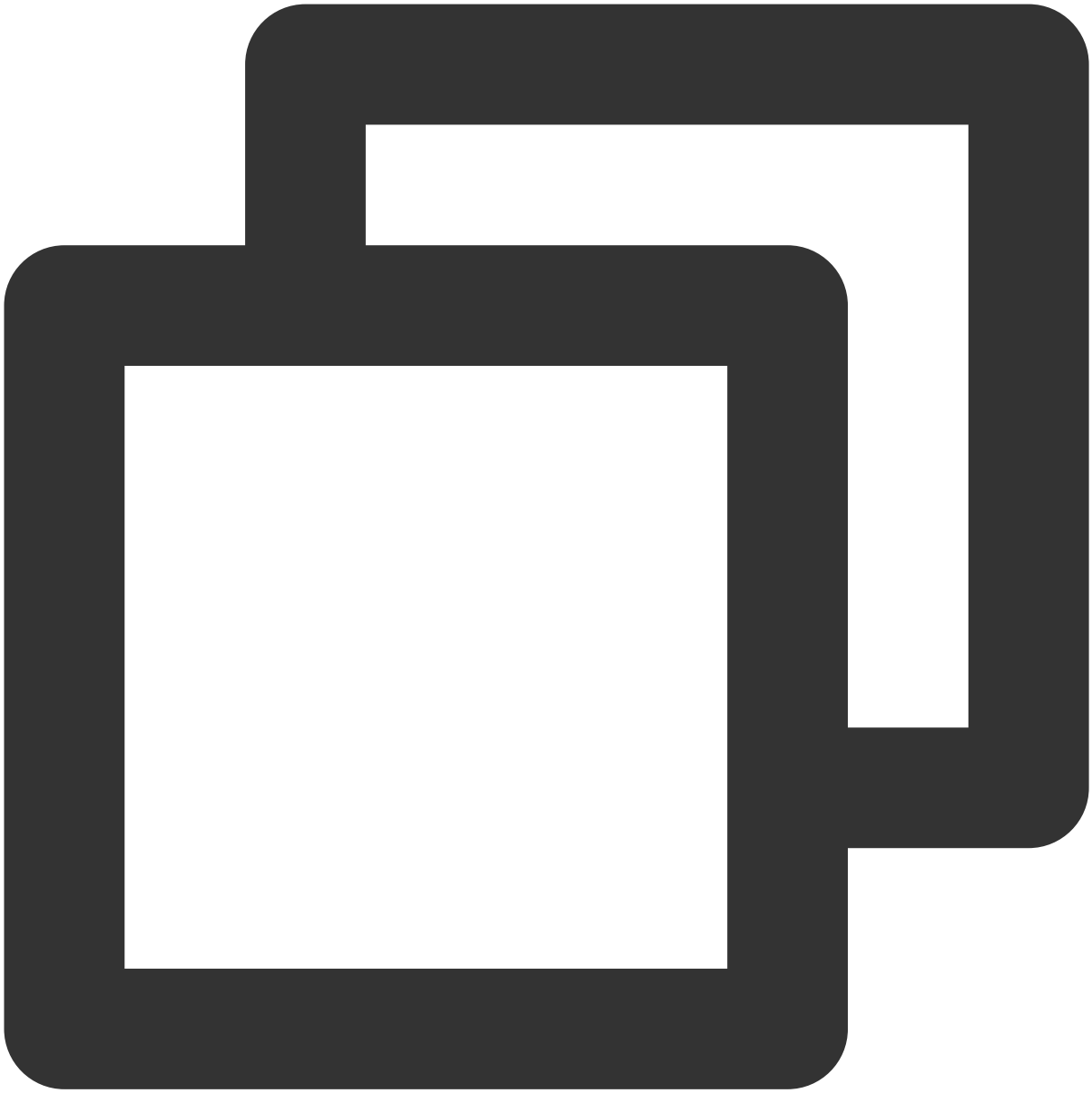
```
{
    "CallbackCommand": "Group.CallbackBeforeSendMsg", // コールバックコマンド
    "GroupId": "@TGS#2J4SZEAEL", // グループID
    "Type": "Public", // グループタイプ
    "From_Account": "jared", // 送信者
    "Operator_Account":"admin", // リクエストの送信者
    "Random": 123456, // 乱数
    "OnlineOnlyFlag": 1, //オンラインメッセージは1、そうでない場合は0となります。ライブストリー
    "MsgBody": [ // メッセージボディです。TIMMessageのメッセージオブジェクトをご参照ください
        {
            "MsgType": "TIMTextElem", // テキスト
```

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

説明：

開発者はMsgBody内のMsgTypeフィールドでメッセージタイプを判断できます。詳細なフィールド説明については、コールバックドキュメントをご参照ください。

開発者は違法なメッセージに対して異なる処理を選択することができ、コールバック内のリターンパケットによってIMバックエンドで制御することができます。



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // ErrorCodeごとに意味が異なります
}
```

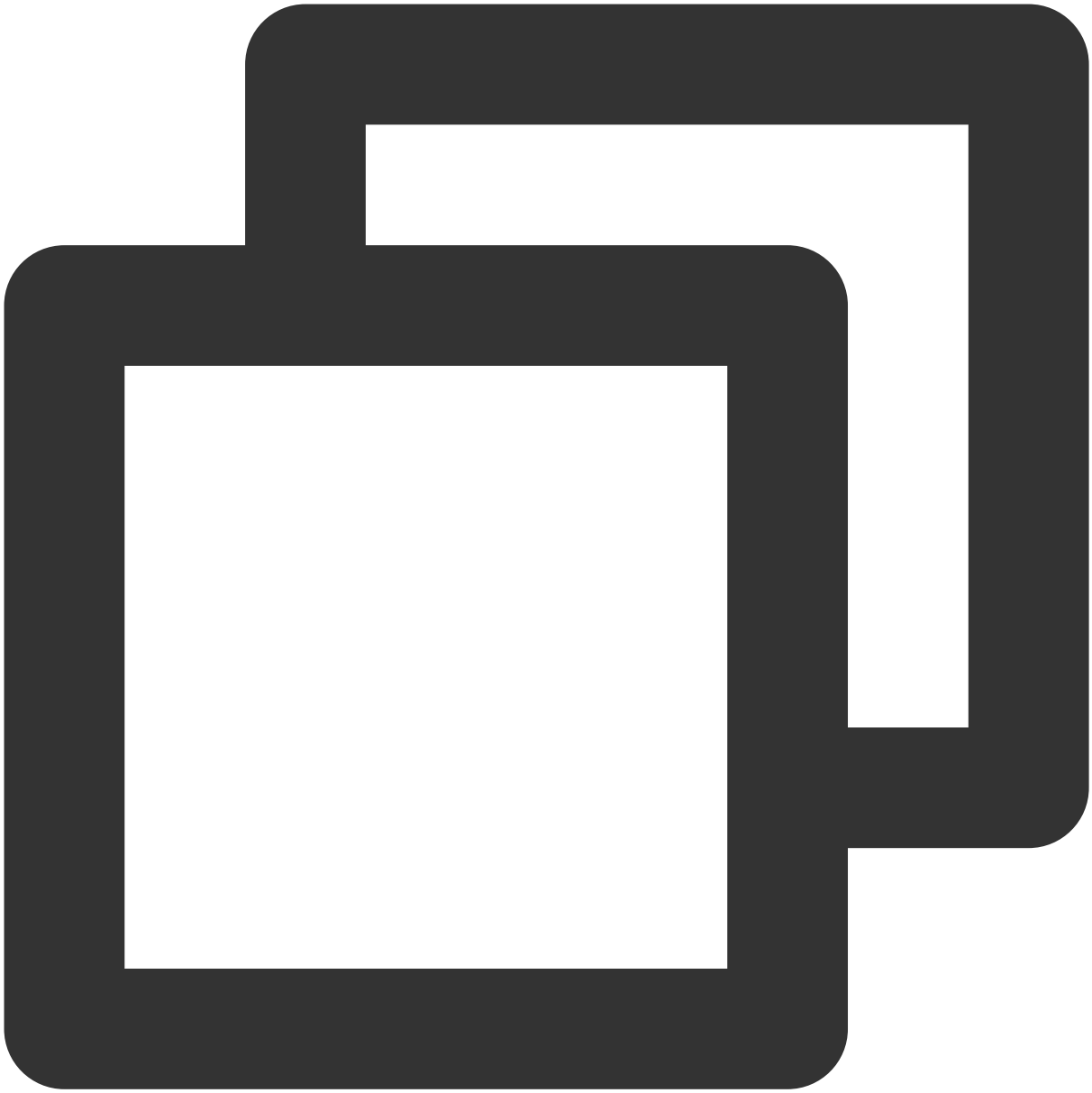
ErrorCode	意味
0	発言を許可し、メッセージを正常に送信しました
1	発言を拒否し、クライアントが10016を返しました

2	サイレントで破棄し、クライアントが正常を返しました

説明：
開発者は自身の業務ニーズに応じて選択して使用できます。

カスタムメッセージタイプ（道具贈り、取引など）

ゲームでのチャットシーンにおいて、テキストメッセージ、スタンプメッセージ、音声メッセージなど簡単なメッセージタイプの他、開発者がメッセージ内容の形式をカスタマイズできるカスタムメッセージタイプも使用されています。カスタムメッセージタイプにより、ゲームチャットルームでゲーム道具を贈ったり、取引をしたりするチャットルームの機能を実装できます。
Tencent Cloud IMは9種類の基本メッセージタイプを提供します。具体的には、テキストメッセージ、スタンプメッセージ、地理位置メッセージ、画像メッセージ、音声メッセージ、ファイルメッセージ、ショート動画メッセージ、システム通知とカスタムメッセージです。カスタムメッセージ以外のメッセージの形式は定義されているため、具体的な情報を記入してください。メッセージタイプの詳細については、メッセージタイプをご参照ください。メッセージタイプの形式の詳細については、メッセージ形式の説明をご参照ください。
サーバーでシングルチャットメッセージの個別送信 、 シングルチャットメッセージの一括送信 、グループ内での一般メッセージの送信を使用し、ユーザーにカスタムメッセージを送信できます。グループ内で一般メッセージを送信する基本的なリクエストパケットの例は次のとおりです：



```
{
    "GroupId": "@TGS#2C5SZEAEF",
    "Random": 8912345, //乱数。5分間で数字が同じな場合、同一メッセージだと判断する
    "MsgBody": [ // メッセージボディ。1つのelement 配列で構成される。詳細については、フィール
        {
            "MsgType": "TIMTextElem", // テキスト
            "MsgContent": {
                "Text": "red packet"
            }
        },
        {
```

```
            "MsgType": "TIMFaceElem", // スタンプ
            "MsgContent": {
            "Index": 6,
            "Data": "abc\\u0000\\u0001"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data",
    "SupportMessageExtension": 0,
}
```

そのうち、`CloudCustomData` を編集することで、メッセージのカスタムデータ（クラウドに保存）を定義し、カスタムメッセージを `MsgBody` （メッセージボディ）に記入して送信できます。

グループ内ボイス/ビデオチャット

ゲーム内のボイス/ビデオチャットは重要機能の1つです。Tencent Cloud IMはInstant Messaging（IM）とTencent Real-Time Communication (TRTC)向けのオーディオビデオ通話機能（TUICallKit）を用意しており、IMでのチャットにリアルタイムの音声通話またはビデオ通話を提供します。

説明：

オーディオビデオ通話機能をより良く体験するために、SDKAppIDごとに7日間のオーディオビデオ通話機能体験版を提供します。IMコンソールでご利用のアプリケーションに対応する体験版を申請し使用することができます。SDKAppIDごとに体験版を1回のみ申請できます。

TUICallKitの詳細については、UIありの統合方法-オーディオビデオ通話をご参照ください。

ゲームチャットルームのタイプ

ゲームチャットルームには以下のタイプがあります：

タイプ	特徴
チャネル	チャットに参加する人数が非常に多く、固定メンバーリストがなく、グループチャットへの参加申請が不要で、いつでも自由に参加/退出できます。オフラインメッセージプッシュは送信不要です。
ロビー	チャットに参加する人数が比較的多く、自由に参加/退出できます。メッセージ履歴は検索可能です。
チーム	チャットに参加する人数が少なく、チャット相手が知らない人の可能性があり、ゲームが終了すると、チャットルームが廃棄されます。オフラインメッセージプッシュは送信不要です。
友達	C2Cチャットであり、チャット履歴が保存され、チャット相手が友達リストに存在します。
私信	C2Cチャットであり、チャット相手が知らない人の可能性があります。

ライブストリーミンググループ	チャットに参加する人数には制限がなく、いつでも自由に参加/退出できます。

IMが提供するゲームチャットルームには以下のタイプがあります：

タイプ	特徴
友達ワークグループ(Work)	作成後はすでにグループ内にいる友達のみがグループ参加に招待できます。被招待者側の同意またはグループの承認は不要です。
知らない人とのソーシャルグループ(Public)	作成後はグループマスターがグループ管理者を指定できます。ユーザーはグループIDを検索してグループ参加申請を送信した後、グループマスターまたは管理者が申請を承認してからでないとグループに参加できません。
臨時ミーティンググループ（Meeting）	作成後は自由に参加・退出でき、かつグループ参加前のメッセージを確認する機能をサポートしています。音声/ビデオ会議のシナリオ、eラーニングのシナリオなどのTRTC製品と連携させたシナリオに適しています。旧バージョンのChatRoomと同じです。
ライブストリーミンググループ（AVChatRoom）	作成後は自由に参加・退出ができ、グループ参加者数の上限はありませんが、メッセージ履歴の保存はサポートしていません。CSS製品との連携に適しており、弾幕チャットのシナリオに使用します。
コミュニティ（Community）	作成後、自由に出入りでき、最大10w人をサポートします。またメッセージ履歴保存、ユーザー検索グループIDをサポートし、グループ追加申請を開始でき、管理者の承認なしにグループに参加できます。

以下では、IMのグループ特徴によって、例とするソリューションを提供します。必要に応じて、ゲーム内で活用してください。

タイプ	ソリューション	特徴
チャネル、ロビー	コミュニティ（Community）	チャットに参加している人が多いため、作成後に自由に参加または退出でき、承認不要です
友達	シングルチャット+権限制御（友達だけにメッセージ送信可能	友達のみへのメッセージ送信をサポートします
私信	シングルチャット+権限制御（Appで任意2人のユーザーの間で送信したシングルチャットメッセージ）	お互い知らない任意2人の間でメッセージを送信するのをサポートします
チーム	知らない人とのソーシャルグループ	ゲーム内のチームメンバーだけがチャットグ

	（Public）、臨時ミーティンググループ（Meeting）	ループに参加できます。オーディオビデでチャットするのをサポートします
ライブストリーミンググループ	ライブストリーミンググループ（AVChatRoom）	グループメンバー数には制限がありません。いつでも自由に参加または退出できます

説明：

友達とのチャット及び私信でのチャットにおける権限制御の詳細については、シングルチャットメッセージの権限制御をご参照ください。

グループシステムの詳細については、グループシステムをご参照ください。