即时通信 IM

插件市场



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

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

【商标声明】

腾讯云

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

【服务声明】

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

【联系我们】

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

文档目录

插件市场
概述及开通指引
云端搜索插件
效果展示
集成指引
Android&iOS&Windows&Mac
Web&小程序&uni-app
常见问题
投票插件
效果展示
集成指引
Android & iOS
flutter
联系我们
接龙插件
效果展示
集成指引
Android & iOS
联系我们
文本消息翻译插件
效果展示
集成指引
Android & iOS
联系我们
语音转文字插件
效果展示
集成指引
Android & iOS
Web & H5 & Uniapp
联系我们
会话分组插件
效果展示
集成指引
Android & iOS
联系我们
会话标记插件

版权所有：腾讯云计算（北京）有限责任公司

效果展示
集成指引
Android & iOS
联系我们

插件市场
概述及开通指引

最近更新时间：2024-12-26 10:22:13

插件市场简介

插件市场是集合了即时通信 IM 可扩展功能的平台，您可根据需要从插件市场中选择插件使用，以帮助您更高效、便捷地集成业务所需能力。
您可以在插件市场中浏览提供插件的功能介绍、支持平台、支持语言、版本大小等信息，也可以领取免费试用、付费购买插件服务，根据集成指引完成插件功能配置。

插件功能概述

插件名称	功能描述	体验链接
推送服务	推送服务 Push，提供稳定、及时、多样化的推送服务。接入便捷，3 行代码 2 分钟，即可完成多个厂商的推送接入；支持普通消息推送和全员 / 标签推送、漫游，支持落地和不落地两种推送方式；提供完整的推送生命周期查询、数据统计、问题排查等服务。	立即体验
云端搜索	云端搜索插件（CloudMessageSearch），支持云端全局/指定会话消息（Message）搜索、账号（User)搜索、群组（Group）搜索、群成员搜索，解锁更多业务功能玩法，提升用户使用体验。	立即体验
文本消息翻译	依赖于 TUIChat，集成后手动长按消息列表中的文本消息 item，在出现的菜单中，单击翻译按钮，即可翻译文本。	立即体验
语音转文字	依赖于 TUIChat，集成后长按语音消息时，在出现的菜单中，单击转文字，即可实现语音转文字。	立即体验
会话分组	依赖于 TUIConversation，集成后可以创建分组、编辑分组、删除分组、展示分组列表。	立即体验
会话标记	依赖于 TUIConversation，集成后可以标记会话、取消标记会话、展示标记分组。	立即体验
投票	依赖于 TUIChat，集成后可以直接在 TUIChat 中发起实名/匿名投票、查看投票统计结果、参与其他人发起的投票等。	立即体验
接龙	依赖于 TUIChat，集成后可以直接在 TUIChat 中发起接龙活动、参与接龙、设置截止时间、查看接龙结果。	立即体验

插件市场开通指引

版权所有：腾讯云计算（北京）有限责任公司

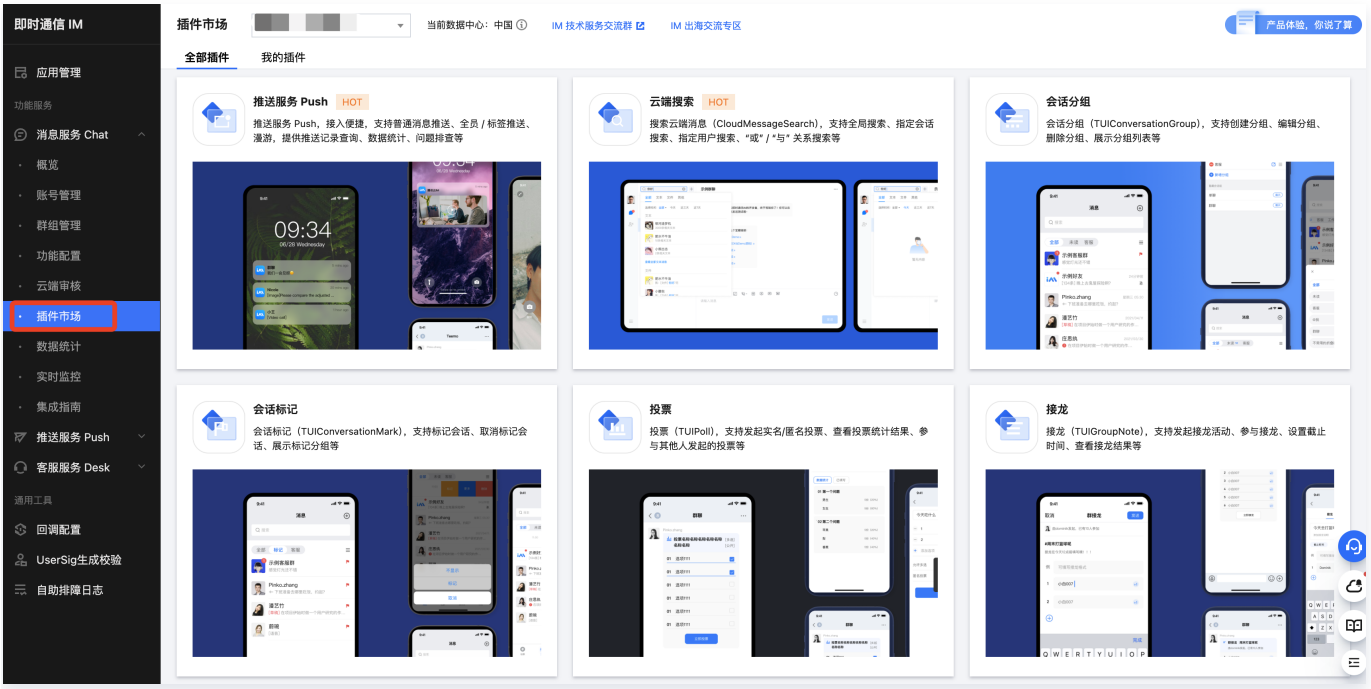
前提条件

1. 已 注册腾讯云 账号并完成 实名认证 。
2. 已有可选 IM 应用。如果没有可选应用，请登录 即时通信 IM 控制台 ，单击**创建新应用**，完成创建操作。

> **说明：**
> 新建应用的业务版本默认为体验版（免费），状态默认为启用。 同一个腾讯云账号，最多可创建300个即时通信 IM 应用。若已有300个应用，您可以先 停用并删除 无需使用的应用后再创建新的应用。**应用删除后，该 SDKAppID 对应的所有数据和服务不可恢复，请谨慎操作。**

进入插件市场

在 即时通信 IM 控制台 总览页，选择需要安装插件功能的 IM 应用，单击进入所选应用的配置页面。
单击左侧**消息服务Chat**目录，选择**插件市场**即可进入，查看所有可使用插件。



免费试用插件

单击任一插件，即可进入对应插件的插件详情页面。单击**免费试用**，并在弹窗中单击**确认**，即可免费试用对应插件功能。



> **注意：**
>
> 每个 IM 应用下，每个插件限免费试用2次，每次有效期7天且不支持延期。未购买插件前可申请免费试用，试用结束立即停服，请提前购买。

购买插件使用

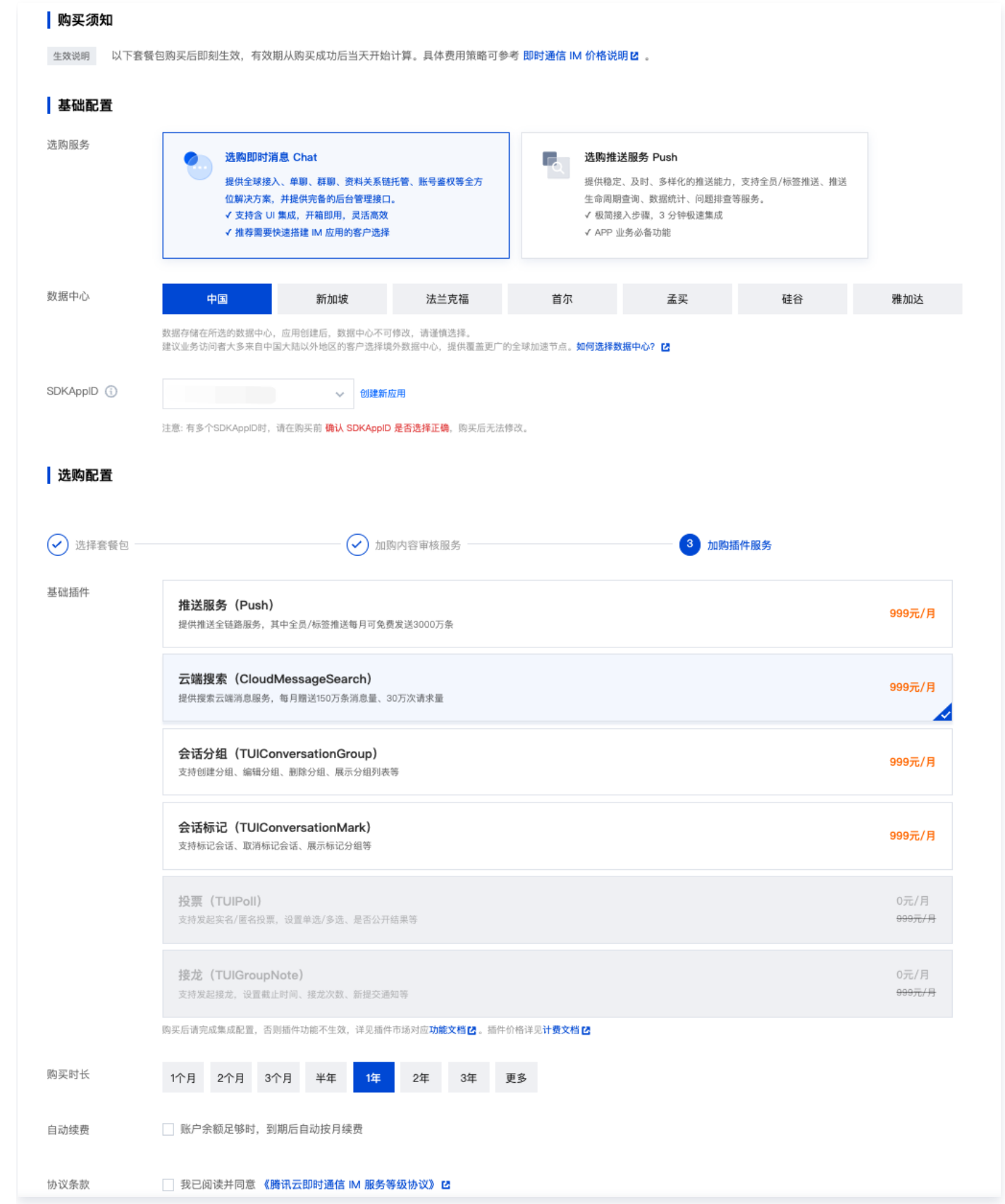
在插件详情页面，单击**立即购买**，跳转至购买页面。



> **注意**
> 购买正式插件需先购买 IM 专业版或旗舰版套餐包。

在购买页面选择**需要购买的插件**，选择**购买时长**，按需勾选**自动续费**，勾选**腾讯云即时通信 IM 服务等级协议**，单击**立即购买**，并完成付费操作。

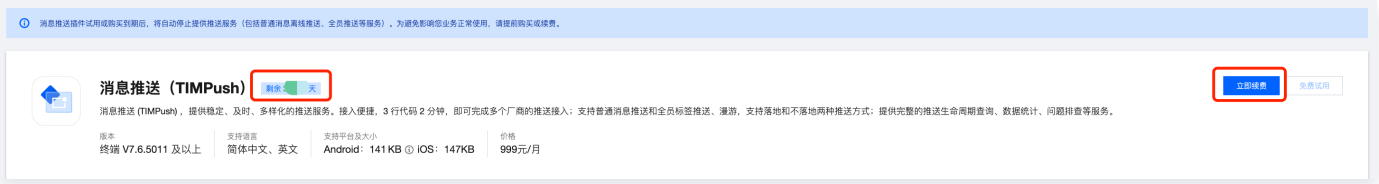
版权所有：腾讯云计算（北京）有限责任公司



注意

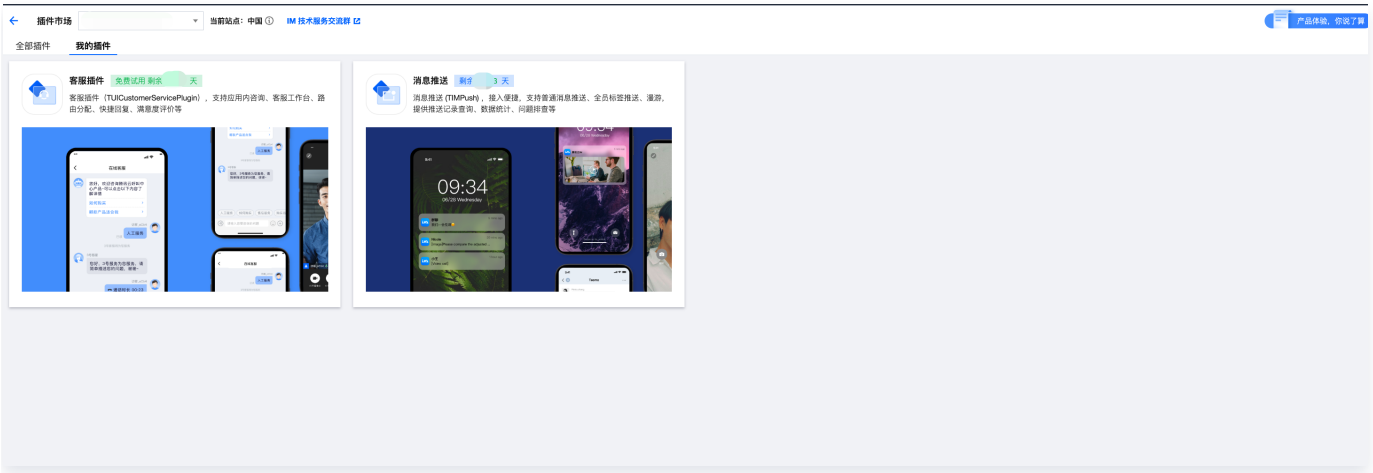
支付成功后，请回到插件市场控制台，按照集成指引完成功能配置，否则插件功能不生效。

回到控制台插件市场，可在插件详情页面看到插件的付费状态和到期时间，为保证您业务正常使用，请及时续费。



查看可用插件

在**我的插件**中，可看到所有在有效期内可使用的插件，为保证您业务正常使用，请在到期前及时续费。



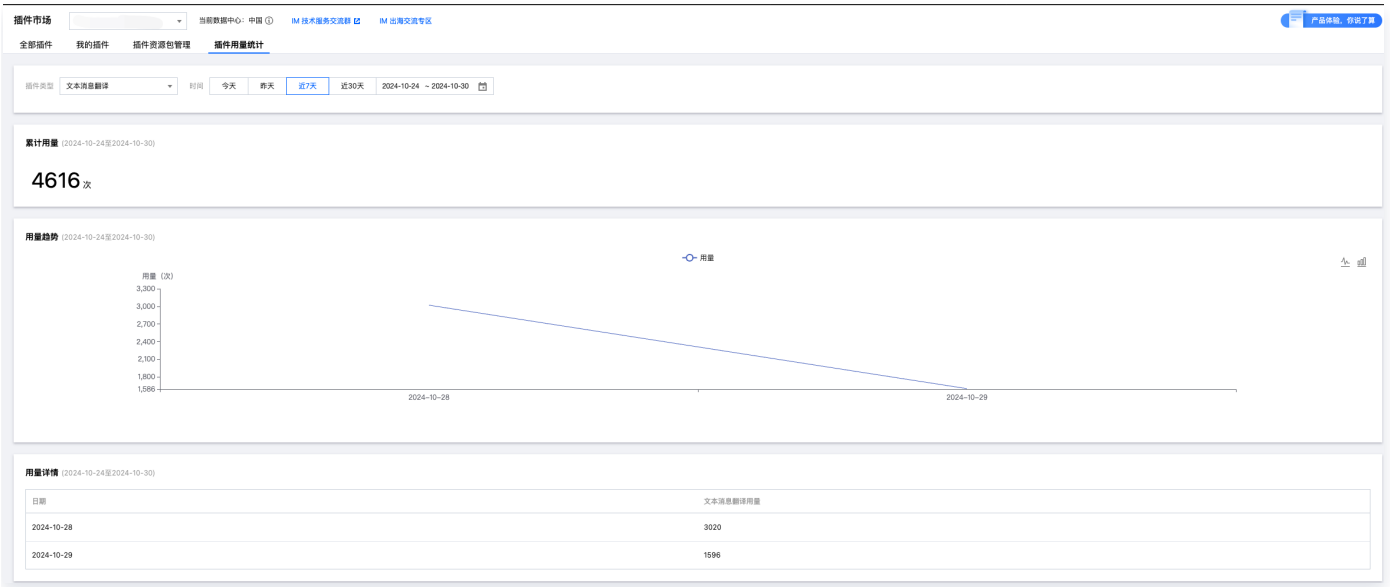
插件资源包管理

在**插件资源包管理**中，可看到当前应用下，所有插件会赠送的资源包以及用量情况。为保证您业务正常使用，请及时续费。



插件用量统计

您可以在**插件用量统计**页面中看到具体插件的累计用量与用量趋势。



联系我们

加入 – 腾讯云即时通信 IM 交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

直接单击进入交流群 。

版权所有：腾讯云计算（北京）有限责任公司

云端搜索插件
效果展示

最近更新时间：2025-03-17 17:20:22

使用须知

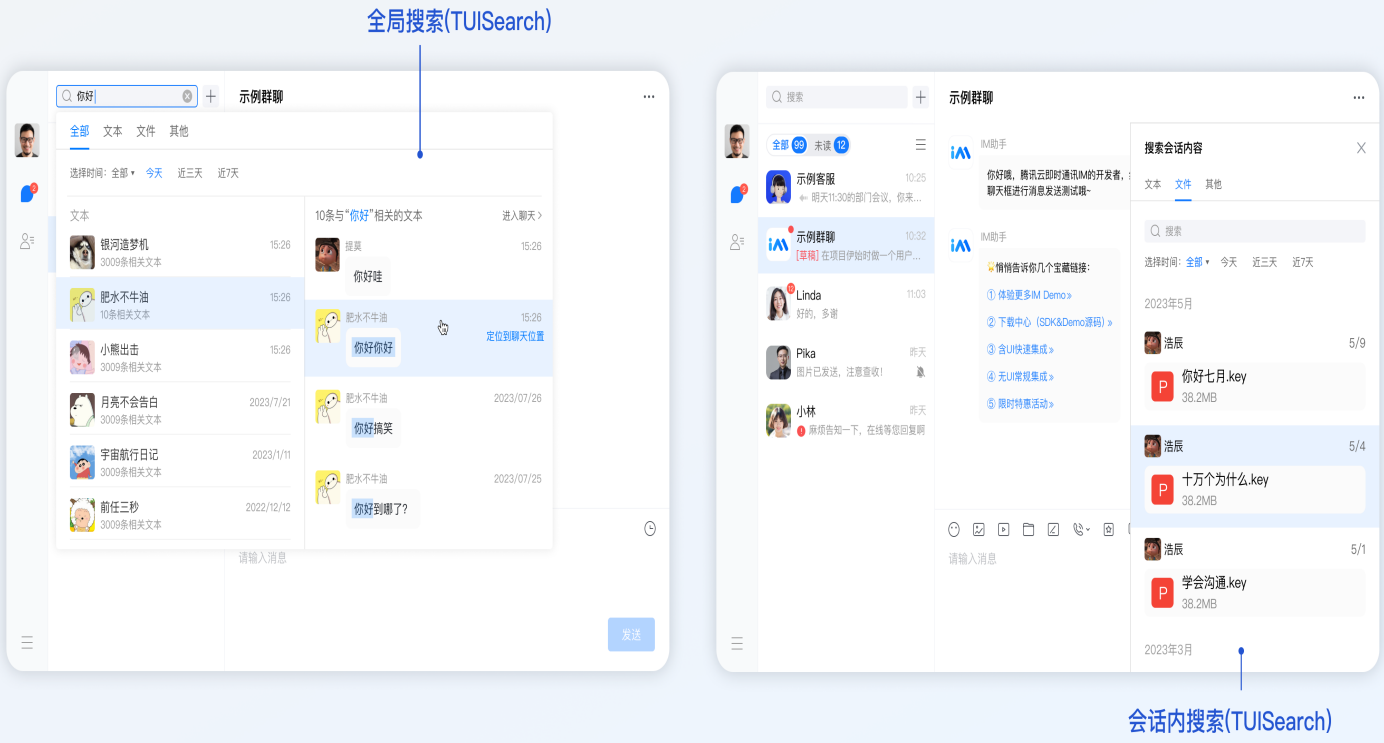
1. 云端搜索插件能力全新升级，在搜索消息基础上，新增**云端用户/群组/群成员搜索**。
2. 调用接口前，请前往 即时通信 IM 控制台 购买正式云端搜索插件或领取体验版并开通对应搜索功能，否则将调用失败。



功能体验

单击体验消息云端搜索功能： Web&H5 Demo

TUIKit 消息云端搜索

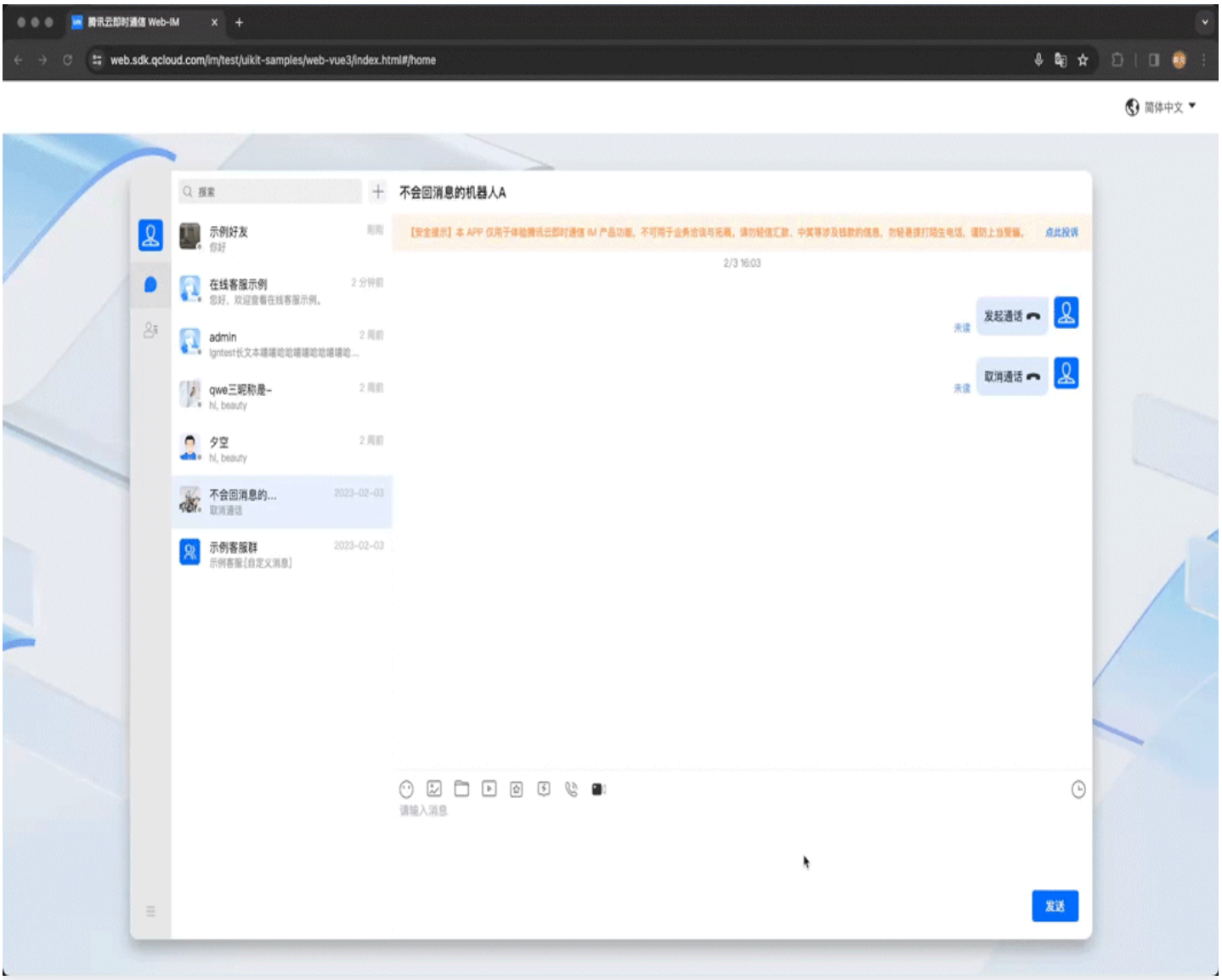




使用场景

场景一：全局消息搜索

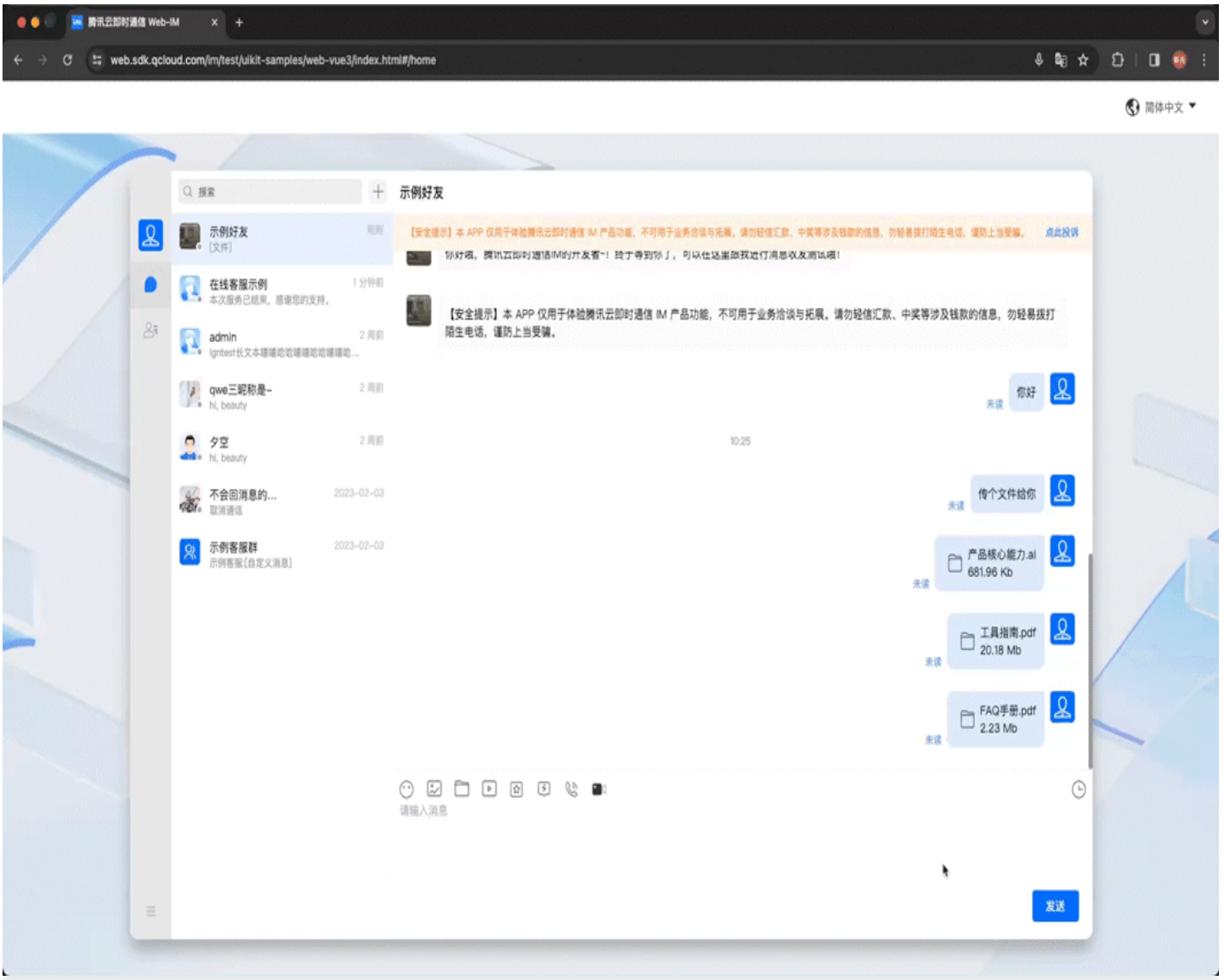
通过消息搜索功能，用户可全局搜索所有包含指定关键词的会话，还可在搜索栏中自由选择搜索的时间范围。用户单击搜索到的会话，可以直接定位至对应聊天位置，并高亮显示该消息。操作演示如下：



场景二：指定会话内搜索文件/图片/视频/其他

通过消息搜索功能，用户可直接搜索指定会话内的全部文件/图片/视频/其他消息，还可在指定的时间段内搜索。用户点击搜索到的文件/图片/视频/其他，同样可以直接定位至对应聊天位置，并高亮显示该文件/图片/视频/其他消息。以下演示以文件为例：

版权所有：腾讯云计算（北京）有限责任公司



场景三：全局账号/群组/群成员搜索

可用于搜索全局应用下的账号（user），群组（group）以及群成员（group member），帮助客户高效检索，拓展陌生人交友玩法等。



集成指引

Web&小程序&uni-app：集成指引
Android&iOS&Windows&Mac：集成指引

集成指引
Android&iOS&Windows&Mac

最近更新时间：2025-01-21 17:44:52

搜索云端消息

功能描述

消息云端搜索，提升 App 使用体验必备功能，可以帮助用户从纷繁复杂的信息中直接找到预期内容，快捷方便；也可以作为运营工具，增加相关内容的引导，简洁高效。

> **说明：**
> - 消息云端搜索功能仅 7.3.4358 及以上版本支持，最大支持搜索近360天内的消息数据，与历史消息存储时间保持一致。
> - 该功能为增值功能，详细参见 价格说明 。
> - 如果您没有开通该服务，调用接口会返回 60020 错误码。

消息搜索类介绍

消息搜索参数类

消息搜索参数类为 `V2TIMMessageSearchParam` （Android / iOS & Mac / Windows）。搜索消息时，SDK 会按照该对象的设置，执行不同的搜索逻辑。
`V2TIMMessageSearchParam` 的参数说明如下：

参数	含义	说明
keywordList	关键字列表	最多支持 5 个。当消息发送者以及消息类型均未指定时，必须设置关键字列表；否则，关键字列表可以为空。
keywordListMatchType	指定关键字列表匹配类型	可设置为 “或” 关系搜索，或 “与” 关系搜索。取值分别为 `V2TIM_KEYWORD_LIST_MATCH_TYPE_OR` 和 `V2TIM_KEYWORD_LIST_MATCH_TYPE_AND` 。默认为 “或” 关系搜索。
senderUserIDList	指定 userID 发送的消息	最多支持 5 个。
messageTypeList	指定搜索的消息类型集合	传空表示搜索支持的全部类型消息（ `V2TIMFaceElem` 和 `V2TIMGroupTipsElem` 不支持搜索）。其他类型取值参考

		`V2TIMElemType`（Android / iOS & Mac / Windows）。
conversationID	搜索“全部会话”还是搜索“指定的会话”	`conversationID` 为空，搜索全部会话；`conversationID` 不为空，搜索指定会话。
searchTimePosition	搜索的起始时间点	默认为 0（从现在开始搜索）。UTC 时间戳，单位：秒。
searchTimePeriod	从起始时间点开始的过去时间范围	默认为 0（不限制时间范围）。24 x 60 x 60 代表过去一天，单位：秒。
searchCount	搜索的数量	搜索的数量，最大支持100。
searchCursor	搜索的游标	搜索的起始位置，第一次填写空字符串，续拉时填写上一次返回的 `V2TIMMessageSearchResult` 中的 `searchCursor`。

消息搜索结果类

消息搜索结果类为 `V2TIMMessageSearchResult`（Android / iOS & Mac / Windows）。参数说明如下：

参数	含义	说明
totalCount	搜索结果总数	• 如果搜索指定会话，返回满足搜索条件的**消息总数**； • 如果搜索全部会话，返回满足搜索条件的消息所在的所有**会话总数量**。
messageSearchResultItems	搜索结果列表	• 如果搜索指定会话，返回结果列表只包含该会话结果； • 如果搜索全部会话，会对满足搜索条件的消息根据会话 ID 分组，分页返回分组结果。
searchCursor	续拉的游标	调用搜索接口续拉时需要填的游标

其中 `messageSearchResultItems` 是个列表，内含 `V2TIMMessageSearchResultItem`（Android / iOS & Mac / Windows）对象，参数说明如下：

参数	含义	说明
conversationID	会话 ID	-

messageCount	消息数量	当前会话一共搜索到了多少条符合要求的消息。

messageList	满足搜索条件的消息列表	• 如果搜索指定会话，`messageList` 中装载的是本会话中所有满足搜索条件的消息列表。
		• 如果搜索全部会话，`messageList` 中装载的消息条数会有如下两种可能：
		  ○ 如果某个会话中匹配到的消息条数 > 1，则 `messageList` 为空，您可以在 UI 上显示 “{ `messageCount` } 条相关记录”。
		  ○ 如果某个会话中匹配到的消息条数 = 1，则 `messageList` 为匹配到的那条消息，您可以在 UI 上显示之，并高亮匹配关键词。

搜索全部会话的消息

当用户在搜索框输入关键字搜索消息时，您可以调用 `searchCloudMessages`（Android / iOS & Mac / Windows）搜索消息。
如果您希望在全部会话范围内搜索，只需要将 `V2TIMMessageSearchParam` 中的 `conversationID` 设置为空或者不设置即可。
示例代码如下：

Android

```
List<String> keywordList = new ArrayList<>();
keywordList.add("abc");
keywordList.add("123");
V2TIMMessageSearchParam searchParam = new V2TIMMessageSearchParam();
// 设置搜索关键字
searchParam.setKeywordList(keywordList);
// 搜索20条数据
searchParam.setSearchCount(20);
// 从最新的会话开始搜索
searchParam.setSearchCursor("");
// 从当前时间开始搜索
searchParam.setSearchTimePosition(0);
// 搜索10分钟之内的消息
searchParam.setSearchTimePeriod(600);
V2TIMManager.getMessageManager().searchCloudMessages(searchParam, ne
wV2TIMValueCallback<V2TIMMessageSearchResult>() {
    @Override
```

```
public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchRes
ult) {
      // 搜索成功
    }

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

iOS & Mac

```
V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam
alloc] init];
// 设置搜索关键字
param.keywordList = @[@"abc", @"123"];
param.messageTypeList = nil;
param.conversationID = nil;
param.searchTimePosition = 0;
param.searchTimePeriod = 0;
// 搜索20条数据
param.searchCount = 20;
// 从最新的会话开始搜索
param.searchCursor = @"";
[V2TIMManager.sharedInstance searchCloudMessages:param

succ:^(V2TIMMessageSearchResult *searchResult) {
    // 搜索成功，searchResult 中返回搜索结果
} fail:^(int code, NSString *desc) {
    // 搜索失败
}];
```

Windows

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

using ErrorCallback = std::function<void(int, const V2TIMString&)>;

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

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

    void OnSuccess(const T& value) override {
      if (success_callback_) {
      success_callback_(value);
    }
    }

void OnError(int error_code, const V2TIMString& error_message) overr
ide {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageSearchParam searchParam;
// 设置搜索关键字
searchParam.keywordList.PushBack("abc");
searchParam.keywordList.PushBack("123");
// 搜索20条数据
searchParam.searchCount = 20;
// 从最新的会话开始搜索
```

```
searchParam.searchCursor = "";

auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
callback->SetCallback(
    [=](const V2TIMMessageSearchResult& messageSearchResult) {
      // 搜索成功
      delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
      // 搜索失败
      delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()-
>SearchCloudMessages(searchParam, callback);
```

搜索指定会话的消息

当用户在搜索框输入关键字搜索消息时，您可以调用 `searchCloudMessages`（Android / iOS & Mac / Windows）搜索消息。
示例代码如下：

Android

```
List<String> keywordList = new ArrayList<>();
keywordList.add("abc");
keywordList.add("123");
V2TIMMessageSearchParam searchParam = new V2TIMMessageSearchParam();
// 搜索与用户 user1 的单聊消息
searchParam.setConversationID("c2c_user1");
// 设置搜索关键字
searchParam.setKeywordList(keywordList);
// 搜索20条数据
searchParam.setSearchCount(20);
// 从最新的会话开始搜索
searchParam.setSearchCursor("");
// 从当前时间开始搜索
searchParam.setSearchTimePosition(0);
// 搜索10分钟之内的消息
searchParam.setSearchTimePeriod(600);
```

```
V2TIMManager.getMessageManager().searchCloudMessages(searchParam, ne
wV2TIMValueCallback<V2TIMMessageSearchResult>() {
    @Override

public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchRes
ult) {
      // 搜索成功
    }

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

iOS & Mac

```
V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam
alloc] init];
// 设置搜索关键字
param.keywordList = @[@"abc", @"123"];
param.messageTypeList = nil;
// 搜索与用户 user1 的单聊消息
param.conversationID = @"c2c_user1";
param.searchTimePosition = 0;
param.searchTimePeriod = 0;
// 搜索20条数据
param.searchCount = 20;
// 从最新的会话开始搜索
param.searchCursor = @"";
[V2TIMManager.sharedInstance searchCloudMessages:param

succ:^(V2TIMMessageSearchResult *searchResult) {
    // 搜索成功，searchResult 中返回搜索结果
} fail:^(int code, NSString *desc) {
    // 搜索失败
}];
```

Windows

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

using ErrorCallback = std::function<void(int, const V2TIMString&)>;

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

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

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }

void OnError(int error_code, const V2TIMString& error_message) overr
ide {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageSearchParam searchParam;
// 搜索与用户 user1 的单聊消息
searchParam.conversationID = "c2c_user1";
// 设置搜索关键字
searchParam.keywordList.PushBack("abc");
```

```
searchParam.keywordList.PushBack("123");
// 搜索20条数据
searchParam.searchCount = 20;
// 从最新的会话开始搜索
searchParam.searchCursor = "";

auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
callback->SetCallback(
    [=](const V2TIMMessageSearchResult& messageSearchResult) {
        // 搜索成功
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // 搜索失败
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()-
>SearchCloudMessages(searchParam, callback);
```

搜索典型场景示例

普通的 IM 聊天软件，搜索界面的展示通常分这几种场景：

图 1：搜索聊天记录	图 2：搜索更多聊天记录	图 3：搜索指定会话的消息



下文我们将依次向您展示如何利用 IM SDK 的搜索 API 实现上图的典型场景。

展示最近几个活跃的会话

如图 1 所示，最下方是搜索到的消息所属的最近 3 个会话列表，实现方式如下：

1. 设置搜索参数 `V2TIMMessageSearchParam`
   - `conversationID` 设置为 `null`，表示搜索所有会话的消息。
   - `searchCursor` 设置为 ""，表示搜索最新的数据。
   - `searchCount` 设置为 3，表示返回最近的会话数量，UI 上一般显示 3 条。
2. 处理搜索回调结果 `V2TIMMessageSearchResult`
   - `totalCount` 表示匹配到的消息所属的所有会话数量。
   - `messageSearchResultItems` 列表为最近 3（即入参 `searchCount`）个会话信息。其中元素 `V2TIMMessageSearchResultItem` 的 `messageCount` 表示当前会话搜索到的消息总数量；
   - 搜索到的消息条数 > 1，则 `messageList` 为空，您可以在 UI 上显示 “4 条相关聊天记录”，其中的 4 为 `messageCount`。
   - 搜索到的消息条数 = 1，则 `messageList` 为匹配到的那条消息，您可以在 UI 上显示消息内容并高亮搜索关键词，例如 搜索典型场景示例 图中的 “test”。

示例代码如下：

Android

```
List<String> keywordList = new ArrayList<>();
keywordList.add("test");
V2TIMMessageSearchParam v2TIMMessageSearchParam = new
V2TIMMessageSearchParam();
// conversationID 设置为 null 表示搜索所有会话中的消息，结果会按照会话分类
v2TIMMessageSearchParam.setConversationID(null);
v2TIMMessageSearchParam.setKeywordList(keywordList);
v2TIMMessageSearchParam.setSearchCursor("");
v2TIMMessageSearchParam.setSearchCount(3);
V2TIMManager.getMessageManager().searchCloudMessages(v2TIMMessageSea
rchParam, new V2TIMValueCallback<V2TIMMessageSearchResult>() {
    @Override
    public void onSuccess(V2TIMMessageSearchResult
v2TIMMessageSearchResult) {
      // 匹配到的消息所属的所有会话数量
      int totalCount = v2TIMMessageSearchResult.getTotalCount();
      // 最近3个根据消息会话分类的信息
      List<V2TIMMessageSearchResultItem> resultItemList =
v2TIMMessageSearchResult.getMessageSearchResultItems();
      for (V2TIMMessageSearchResultItem resultItem :
resultItemList) {
        // 会话 ID
        String conversationID = resultItem.getConversationID();
        // 该会话匹配到的所有消息数量
        int totalMessageCount = resultItem.getMessageCount();
        // 消息列表：如果 totalMessageCount > 1，该列表为空；如果
totalMessageCount = 1，该列表元素为此消息
        List<V2TIMMessage> v2TIMMessageList =
resultItem.getMessageList();
      }
    }

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

iOS & Mac

```
V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam
alloc] init];
param.keywordList = @[@"test"];
// conversationID 设置为 nil 表示搜索所有会话中的消息，结果会按照会话分类
param.conversationID = nil;
param.searchCursor = @"";
param.searchCount = 3;
[V2TIMManager.sharedInstance searchCloudMessages:param
succ:^(V2TIMMessageSearchResult *searchResult) {
// 匹配到的消息所属的所有会话数量
    NSInteger totalCount = searchResult.totalCount;
    // 最近3个根据消息会话分类的信息
    NSArray<V2TIMMessageSearchResultItem
*> *messageSearchResultItems = searchResult.messageSearchResultItems
;
    for (V2TIMMessageSearchResultItem
*searchItem in messageSearchResultItems) {
      // 会话 ID
      NSString *conversationID = searchItem.conversationID;
      // 该会话匹配到的所有消息数量
      NSUInteger messageCount = searchItem.messageCount;
      // 消息列表
      NSArray<V2TIMMessage *> *messageList = searchItem.messageList
?: @[];
    }
} fail:^(int code, NSString *desc) {
    // fail
}];
```

Windows

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

using ErrorCallback = std::function<void(int, const V2TIMString&)>;
```

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

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

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }

void OnError(int error_code, const V2TIMString& error_message) overr
ide {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
  SuccessCallback success_callback_;
  ErrorCallback error_callback_;
};

V2TIMMessageSearchParam searchParam;
// conversationID 设置为空表示搜索所有会话中的消息，结果会按照会话分类
searchParam.conversationID = "";
searchParam.keywordList.PushBack("test");
searchParam.searchCursor = "";
searchParam.searchCount = 3;

auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
callback->SetCallback(
    [=](const V2TIMMessageSearchResult& messageSearchResult) {
        // 匹配到的消息所属的所有会话数量
        uint32_t totalCount = messageSearchResult.totalCount;
        // 最近3个根据消息会话分类的信息
        V2TIMMessageSearchResultItemVector
messageSearchResultItems =
```

```
        messageSearchResult.messageSearchResultItems;
        for (size_t
i = 0; i < messageSearchResultItems.Size(); ++i) {
          // 会话 ID
          V2TIMString
conversationID = messageSearchResultItems[i].conversationID;
          // 该会话匹配到的所有消息数量

uint32_t messageCount = messageSearchResultItems[i].messageCount;
          // 消息列表：如果 messageCount > 1，该列表为空；如果
messageCount = 1，该列表元素为此消息
          V2TIMMessageVector
messageList = messageSearchResultItems[i].messageList;
        }
        delete callback;
      },
      [=](int error_code, const V2TIMString& error_message) {
        // 搜索失败
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()-
>SearchCloudMessages(searchParam, callback);
```

展示所有搜索到的消息所属会话列表

点击搜索典型场景示例图 1 中的 “更多聊天记录”，会跳转到图 2，展示所有搜索到的消息所属的会话列表。搜索参数和搜索结果描述跟上文的场景类似。

为了防止内存膨胀，我们强烈建议您分页加载会话列表。

举个例子，分页加载，每页展示 10 条会话结果，搜索参数 `V2TIMMessageSearchParam` 可以参考如下设置：

1. 首次调用：设置参数 `searchCount` = 10，`searchCursor` = ""。调用 `searchCloudMessages` 获取消息搜索结果，解析并展示到首页，并且从结果回调中获取会话总数量 `totalCount` 以及下次请求的游标 `searchCursor` 。
2. 当界面滑动快到底部后根据上一次请求结果中的游标 `searchCursor` 继续拉取下一页的数据 。

示例代码如下：

Android

```
......
```

```
// 记录搜索游标
String searchCursor = "";
......

private void searchConversation(String cursor) {
    List<String> keywordList = new ArrayList<>();
    keywordList.add("test");

V2TIMMessageSearchParam v2TIMMessageSearchParam = new V2TIMMessageSe
archParam();
    v2TIMMessageSearchParam.setConversationID(null);
    v2TIMMessageSearchParam.setKeywordList(keywordList);
    v2TIMMessageSearchParam.setSearchCount(10);
    v2TIMMessageSearchParam.setSearchCursor(cursor);

V2TIMManager.getMessageManager().searchCloudMessages(v2TIMMessageSea
rchParam, newV2TIMValueCallback<V2TIMMessageSearchResult>() {
      @Override

public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchRes
ult) {
        // 匹配到的消息所属的所有会话数量
        int totalCount = v2TIMMessageSearchResult.getTotalCount();
        // 下一页的游标
        searchCursor = v2TIMMessageSearchResult.getSearchCursor();
        // 该页的根据消息会话分类的信息

List<V2TIMMessageSearchResultItem> resultItemList = v2TIMMessageSear
chResult.getMessageSearchResultItems();

for (V2TIMMessageSearchResultItem resultItem : resultItemList) {
        // 会话 ID
        String conversationID = resultItem.getConversationID();
        // 该会话匹配到的所有消息数量
        int totalMessageCount = resultItem.getMessageCount();
        // 消息列表：如果 totalMessageCount > 1，该列表为空；如果
totalMessageCount = 1，该列表元素为此消息

List<V2TIMMessage> v2TIMMessageList = resultItem.getMessageList();
      }
    }

    @Override
```

```
    public void onError(int code, String desc) {}
    });
}

// 当需要加载下一页时
public void loadMore() {
    searchConversation(searchCursor);
}
```

iOS & Mac

```
......
// 记录搜索游标
NSString *searchCursor = @"";
......

- (void)searchConversation:(NSString *)cursor {
    V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam
alloc] init];
    param.keywordList = @[@"test"];
    param.conversationID = nil;
    param.searchCursor = cursor;
    param.searchCount = 10;
    [V2TIMManager.sharedInstance searchCloudMessages:param
succ:^(V2TIMMessageSearchResult *searchResult) {
      // 匹配到的消息所属的所有会话数量
      NSUInteger totalCount = searchResult.totalCount;
      // 下一页的游标
      searchCursor = searchResult.searchCursor;
      // 该页的根据消息会话分类的信息

NSArray<V2TIMMessageSearchResultItem *> *messageSearchResultItems =
searchResult.messageSearchResultItems;

for (V2TIMMessageSearchResultItem *searchItem in messageSearchResult
Items) {
        // 会话 ID
        NSString *conversationID = searchItem.conversationID;
        // 该会话匹配到的所有消息数量
        NSUInteger totalMessageCount = searchItem.messageCount;
```

```
        // 消息列表：如果 totalMessageCount > 1，该列表为空；如果
totalMessageCount = 1，该列表元素为此消息

NSArray<V2TIMMessage *> *messageList = searchItem.messageList ?: @[]
;
      }
    } fail:^(int code, NSString *desc) {
    // fail
    }];
}

// 当需要加载下一页时
- (void)loadMore {
    [self searchConversation:searchCursor];
}
```

Windows

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

using ErrorCallback = std::function<void(int, const V2TIMString&)>;

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

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

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }
```

```
void OnError(int error_code, const V2TIMString& error_message) overr
ide {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
  SuccessCallback success_callback_;
  ErrorCallback error_callback_;
};

// 记录搜索游标
V2TIMString searchCursor = "";

void SearchConversation(V2TIMString cursor) {
    V2TIMMessageSearchParam searchParam;
    searchParam.keywordList.PushBack("test");
    searchParam.searchCursor = cursor;
    searchParam.searchCount = 10;

    auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
      callback->SetCallback(
        [=](const V2TIMMessageSearchResult& messageSearchResult) {
        // 匹配到的消息所属的所有会话数量
        uint32_t totalCount = messageSearchResult.totalCount;
        // 下一页的游标
        searchCursor = messageSearchResult.searchCursor;
        // 该页的根据消息会话分类的信息
        V2TIMMessageSearchResultItemVector
messageSearchResultItems =
        messageSearchResult.messageSearchResultItems;
        for (size_t
i = 0; i < messageSearchResultItems.Size(); ++i) {
          // 会话 ID
          V2TIMString
conversationID = messageSearchResultItems[i].conversationID;
          // 该会话匹配到的所有消息数量

uint32_t messageCount = messageSearchResultItems[i].messageCount;
          // 消息列表：如果 messageCount > 1，该列表为空；如果
messageCount = 1，该列表元素为此消息
```

```
            V2TIMMessageVector
messageList = messageSearchResultItems[i].messageList;
          }
          delete callback;
        },
        [=](int error_code, const V2TIMString& error_message) {
        // 搜索失败
        delete callback;
      });

      V2TIMManager::GetInstance()->GetMessageManager()-
>SearchLocalMessages(searchParam, callback);
}

// 当需要加载下一页时
void LoadMore() { SearchConversation(searchCursor); }
```

展示搜索指定会话的消息

与 搜索典型场景示例 图 2 展示会话列表不同的是，图 3 所示在指定会话中搜索到的消息列表。
为了防止内存膨胀，我们强烈建议您分页加载消息。举个例子，分页加载，每页展示 10 条消息结果：

1. 搜索参数 `V2TIMMessageSearchParam` 可以参考如下设置：
   - 设置搜索参数 `V2TIMMessageSearchParam` 的 `conversationID` 为搜索的会话 ID。
   - 首次调用：设置参数 `searchCount` = 10，`searchCursor` = ""。调用 `searchCloudMessages` 获取消息搜索结果，解析并展示到首页，并且从结果回调中获取会话总数量 `totalCount` 以及下一页的游标 `searchCursor`。
   - 再次调用：更新参数 `searchCursor` 的值为上一步调用结果中的返回值。
2. 处理搜索结果 `V2TIMMessageSearchResult`：
   - `totalCount` 表示该会话匹配到的所有消息数量。
   - `messageSearchResultItems` 列表只有该会话的结果。列表中的元素 `V2TIMMessageSearchResultItem` 的 `messageCount` 为该分页的消息数量，`messageList` 为该分页的消息列表。
   - `searchCursor` 表示下一页搜索的起始游标。

示例代码如下：

Android

```
......
// 记录搜索游标
String searchCursor = "";
......

private void searchMessage(String cursor) {
    List<String> keywordList = new ArrayList<>();
    keywordList.add("test");

V2TIMMessageSearchParam v2TIMMessageSearchParam = new V2TIMMessageSe
archParam();
    v2TIMMessageSearchParam.setConversationID(conversationID);
    v2TIMMessageSearchParam.setKeywordList(keywordList);
    v2TIMMessageSearchParam.setSearchCount(10);
    v2TIMMessageSearchParam.setSearcuCursor(cursor);

V2TIMManager.getMessageManager().searchCloudMessages(v2TIMMessageSea
rchParam, newV2TIMValueCallback<V2TIMMessageSearchResult>() {
      @Override

public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchRes
ult) {
        // 该会话匹配到的所有消息数量

int totalMessageCount = v2TIMMessageSearchResult.getTotalCount();
        // 下一页的游标
        searchCursor = v2TIMMessageSearchResult.getSearchCursor();
        // 该页消息信息

List<V2TIMMessageSearchResultItem> resultItemList = v2TIMMessageSear
chResult.getMessageSearchResultItems();

for (V2TIMMessageSearchResultItem resultItem : resultItemList) {
          // 会话 ID
          String conversationID = resultItem.getConversationID();
          // 该页的消息数量
          int totalMessageCount = resultItem.getMessageCount();
          // 该页的消息数据列表

List<V2TIMMessage> v2TIMMessageList = resultItem.getMessageList();
        }
```

```
        }

        @Override
        public void onError(int code, String desc) {
        }
    });
}
// 当需要加载下一页时
public void loadMore() {
    searchMessage(searchCursor);
}
```

iOS & Mac

```
......
// 记录搜索游标
NSString *searchCursor = @"";
......

- (void)searchMessage:(NSString *)cursor {
    V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam
alloc] init];
    param.keywordList = @[@"test"];
    // conversationID 是要搜索的会话 ID
    param.conversationID = conversationID;
    param.searchCursor = cursor;
    param.searchCount = 10;
    [V2TIMManager.sharedInstance searchCloudMessages:param
succ:^(V2TIMMessageSearchResult *searchResult) {
        // 该会话匹配到的所有消息数量
        NSUInteger totalMessageCount = searchResult.totalCount;
        // 下一页的游标
        searchCursor = searchResult.searchCursor;
        // 该页消息信息

NSArray<V2TIMMessageSearchResultItem *> *messageSearchResultItems =
searchResult.messageSearchResultItems;

for (V2TIMMessageSearchResultItem *searchItem in messageSearchResult
Items) {
```

```
        // 会话 ID
        NSString *conversationID = searchItem.conversationID;
        // 该页的消息数量
        NSUInteger totalMessageCount = searchItem.messageCount;
        // 该页的消息数据列表

NSArray<V2TIMMessage *> *messageList = searchItem.messageList ?: @[]
;
      }
    } fail:^(int code, NSString *desc) {
      // fail
    }];
}

// 当需要加载下一页时
- (void)loadMore {
    [self searchMessage:searchCursor];
}
```

Windows

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

using ErrorCallback = std::function<void(int, const V2TIMString&)>;

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

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

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
```

```
      }
    }

void OnError(int error_code, const V2TIMString& error_message) overr
ide {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

// 记录搜索游标
V2TIMString searchCursor = "";

void SearchConversation(V2TIMString cursor) {
    V2TIMMessageSearchParam searchParam;
    searchParam.conversationID = conversationID;
    searchParam.keywordList.PushBack("test");
    searchParam.searchCursor = cursor;
    searchParam.searchCount = 10;

    auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
    callback->SetCallback(
      [=](const V2TIMMessageSearchResult& messageSearchResult) {
        // 该会话匹配到的所有消息数量
        uint32_t totalCount = messageSearchResult.totalCount;
        // 下一页的游标
        searchCursor = messageSearchResult.searchCursor;
        // 该页消息信息
        V2TIMMessageSearchResultItemVector
messageSearchResultItems =
        messageSearchResult.messageSearchResultItems;
        for (size_t
i = 0; i < messageSearchResultItems.Size(); ++i) {
          // 会话 ID
          V2TIMString
conversationID = messageSearchResultItems[i].conversationID;
          // 该页的消息数量
```

```
uint32_t messageCount = messageSearchResultItems[i].messageCount;
            // 该页的消息数据列表
            V2TIMMessageVector
messageList = messageSearchResultItems[i].messageList;
        }
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // 搜索失败
        delete callback;
    });

    V2TIMManager::GetInstance()->GetMessageManager()-
>SearchCloudMessages(searchParam, callback);
}

// 当需要加载下一页时
void LoadMore() { SearchConversation(SearchCursor); }
```

搜索自定义消息

通常情况下，如果您使用接口 `createCustomMessage(data)`（Android / iOS & Mac / Windows）创建自定义消息，该消息无法被搜到，因为 SDK 将该自定义消息保存为二进制数据流。
如果您希望自定义消息可以被搜到，需要使用接口 `createCustomMessage(data, description, extension)`（Android / iOS & Mac / Windows）来创建并发送自定义消息，把需要搜索的文本放到 `description` 参数中。
如果您配置了离线推送功能，设置参数 `description` 后，自定义消息也会有离线推送且通知栏展示该参数内容。
如果不需要离线推送可以用发消息接口 `sendMessage`（Android / iOS & Mac / Windows）的参数 `V2TIMOfflinePushInfo` 中的 `disablePush` 来控制。
如果推送的通知栏内容不想展示为被搜索的文本，可以用参数 `V2TIMOfflinePushInfo` 中的 `desc` 来另外设置推送内容。

搜索富媒体消息

富媒体消息包含文件、图片、语音、视频消息。
对于文件消息，界面通常显示文件名。如果调用 `createFileMessage` 创建文件消息时传入 `fileName` 参数，`fileName` 会作为文件消息被搜索的内容，与搜索关键词进行匹配。如果未设置 `fileName`，SDK 则会自动从 `filePath` 提取文件名作为搜索内容。
`fileName` 和 `filePath` 信息会保存到本地和服务器，换设备拉取相关信息后均可搜索。

对于图片、语音、视频消息，并没有类似 `fileName` 这种名称，界面通常显示缩略图或时长，此时指定 `keywordList` 搜索无效。
如果您希望搜索出此类消息，可以指定 `messageTypeList` 为 `V2TIM_ELEM_TYPE_IMAGE` / `V2TIM_ELEM_TYPE_SOUND` / `V2TIM_ELEM_TYPE_VIDEO` 做分类搜索，此时会搜索出所有指定类型的消息。

搜索云端用户

功能描述

您可以通过用户 ID、昵称、性别等信息来搜索云端用户，以便快速找到所需的用户资料。此功能适用于需要查找特定用户信息的场合，例如在熟人社交场景中查找用户以加好友，或者在陌生人社交场景中查找用户以关注。

> **说明：**
> - 用户云端搜索功能仅 8.4 及以上版本支持。
> - 该功能为增值功能，详细参见 价格说明 。
> - 如果您没有开通该服务，调用接口会返回 60020 错误码。

搜索云端用户接口

调用接口 `searchUsers`（Java / Swift / Objective-C / C++）可以搜索云端用户资料，该接口返回的是云端存储的用户资料，包括好友和非好友资料，您可以调用 `checkFriend`（Java / Swift / Objective-C / C++）接口来判断是否为好友。
`V2TIMUserSearchParam` 的参数说明如下：

参数	含义	说明
keywordList	关键字列表	关键字列表最多支持 5 个，keyword 会自动匹配用户 ID、昵称。
keywordListMatchType	指定关键字列表匹配类型	可设置为 “或” 关系搜索，或 “与” 关系搜索。取值分别为 `V2TIM_KEYWORD_LIST_MATCH_TYPE_OR` 和 `V2TIM_KEYWORD_LIST_MATCH_TYPE_AND` 。默认为 “或” 关系搜索。
gender	用户性别	如果不设置，默认男性和女性都会返回。
minBirthday	用户最小生日	如果不设置，默认值为 0。
maxBirthday	用户最大生日	如果不设置，默认 birthday >= minBirthday 的用户都会返回。
searchCount	搜索的数量	必须大于 0，最大支持 100，默认 20。

searchCursor	搜索的游标	搜索的起始位置，第一次填写空字符串，续拉时填写上一次返回的 `V2TIMUserSearchResult` 中的 `searchCursor` 。

用户搜索结果类

消息搜索结果类为 `V2TIMUserSearchResult` （ Java / Swift / Objective-C / C++ ）。参数说明如下：

参数	含义	说明
isFinished	搜索是否已完成	是否已经返回全部满足搜索条件的用户列表。
totalCount	搜索结果总数	满足搜索条件的用户总数量。
nextCursor	续拉的游标	调用搜索接口续拉时需要填的游标。
userList	用户列表	当前一次云端搜索返回的用户列表。

示例代码如下：

Java

```
V2TIMUserSearchParam searchParam = new V2TIMUserSearchParam();
searchParam.setKeywordList(keywordList);
searchParam.setKeywordListMatchType(param.V2TIM_KEYWORD_LIST_MATCH_TYPE_
OR);
searchParam.setSearchCount(20);
searchParam.setSearchCursor("");

V2TIMManager.getInstance().searchUsers(searchParam, new
V2TIMValueCallback<V2TIMUserSearchResult>() {
  @Override
  public void onSuccess(V2TIMUserSearchResult userSearchResult) {
      // 搜索云端用户资料成功
  }

  @Override
  public void onError(int code, String desc) {
      // 搜索云端用户资料失败
  }
});
```

Swift

```
let param = V2TIMUserSearchParam()
param.gender = .V2TIM_GENDER_UNKNOWN;
param.keywordList = ["keyword1", "keyword2"];
param.keywordListMatchType = .V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.searchCount = 20;
param.searchCursor = "";
V2TIMManager.shared.searchUsers(param: param) { searchResult in
    // 搜索云端用户资料成功
} fail: { code, desc in
    // 搜索云端用户资料失败
}
```

Objective-C

```
V2TIMUserSearchParam *param = [[V2TIMUserSearchParam alloc] init];
param.gender = V2TIM_GENDER_UNKNOWN;
param.keywordList = @[@"keyword1", @"keyword2"];
param.keywordListMatchType = V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.searchCount = 20;
param.searchCursor = @"";
[[V2TIMManager sharedInstance] searchUsers:param
succ:^(V2TIMUserSearchResult *searchResult) {
    // 搜索云端用户资料成功
} fail:^(int code, NSString *desc) {
    // 搜索云端用户资料失败
}];
```

C++

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

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

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

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

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMUserSearchParam searchParam;
searchParam.keywordList = keywordList;
param.keywordListMatchType = V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.searchCount = 20;
param.searchCursor = "";

auto callback = new ValueCallback<V2TIMUserSearchResult>{};
callback->SetCallback(
    [=](const V2TIMUserSearchResult& userSearchResult) {
        // 搜索云端用户资料成功
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // 搜索云端用户资料失败
        delete callback;
    });

V2TIMManager::GetInstance()->SearchUsers(searchParam, callback);
```

搜索云端群组

功能描述

您可以通过群 ID、群名称等信息搜索云端群组，以便快速找到所需的群组资料。这一功能特别适用于需要查找特定群组的场景，例如在社交场景中寻找兴趣群以加入。

> **说明：**
> - 群组云端搜索功能仅 8.4 及以上版本支持。
> - 该功能为增值功能，详细参见 价格说明 。
> - 如果您没有开通该服务，调用接口会返回 60020 错误码。

搜索云端群组接口

您可以调用接口 `searchCloudGroups`（Java / Swift / Objective-C / C++）搜索云端群组。`V2TIMGroupSearchParam` 的参数说明如下：

参数	含义	说明
keywordList	关键字列表	关键字列表最多支持 5 个，keyword 会自动匹配群 ID、群名称。
keywordListMatchType	指定关键字列表匹配类型	可设置为 “或” 关系搜索，或 “与” 关系搜索。取值分别为 `V2TIM_KEYWORD_LIST_MATCH_TYPE_OR` 和 `V2TIM_KEYWORD_LIST_MATCH_TYPE_AND` 。默认为 “或” 关系搜索。
searchCount	搜索的数量	必须大于 0，最大支持 100，默认 20。
searchCursor	搜索的游标	搜索的起始位置，第一次填写空字符串，续拉时填写上一次返回的 `V2TIMGroupSearchResult` 中的 `searchCursor` 。

群组搜索结果类

群组搜索结果类为 `V2TIMGroupSearchResult`（ Java / Swift / Objective-C / C++ ）。参数说明如下：

参数	含义	说明
isFinished	搜索是否已完成	满足搜索条件的群列表是否已经全部返回。
totalCount	搜索结果总数	满足搜索条件的群总数量。
nextCursor	续拉的游标	调用搜索接口续拉时需要填的游标。

groupList	群列表	当前一次云端搜索返回的群列表。

示例代码如下：

Java

```
V2TIMGroupSearchParam searchParam = new V2TIMGroupSearchParam();
searchParam.setKeywordList(keywordList);
searchParam.setKeywordListMatchType(param.V2TIM_KEYWORD_LIST_MATCH_TYPE_
OR);
searchParam.setSearchCount(20);
searchParam.setSearchCursor("");

V2TIMManager.getGroupManager().searchCloudGroups(searchParam, new
V2TIMValueCallback<V2TIMGroupSearchResult>() {
  @Override
  public void onSuccess(V2TIMGroupSearchResult groupSearchResult) {
        // 搜索云端群组成功
  }

  @Override
  public void onError(int code, String desc) {
      // 搜索云端群组失败
  }
});
```

Swift

```
let param = V2TIMGroupSearchParam()
param.keywordList = ["keyword1", "keyword2"];
param.keywordListMatchType = .V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.searchCount = 20;
param.searchCursor = "";
V2TIMManager.shared.searchCloudGroups(searchParam: param) { searchResult
in
    // 搜索云端群组成功
} fail: { code, desc in
    // 搜索云端群组失败
}
```

Objective-C

```
V2TIMGroupSearchParam *param = [[V2TIMGroupSearchParam alloc] init];
param.keywordList = @[@"keyword1", @"keyword2"];
param.keywordListMatchType = V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.searchCount = 20;
param.searchCursor = @"";
[[V2TIMManager sharedInstance] searchCloudGroups:param
succ:^(V2TIMGroupSearchResult *searchResult) {
    // 搜索云端群组成功
} fail:^(int code, NSString *desc) {
    // 搜索云端群组失败
}];
```

C++

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

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

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

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

```
private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMGroupSearchParam searchParam;
searchParam.keywordList = keywordList;
searchParam.keywordListMatchType = V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
searchParam.searchCount = 20;
searchParam.searchCursor = "";

auto callback = new ValueCallback<V2TIMGroupSearchResult>{};
callback->SetCallback(
    [=](const V2TIMGroupSearchResult& groupSearchResult) {
        // 搜索云端群组成功
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // 搜索云端群组失败
        delete callback;
    });

V2TIMManager::GetInstance()->GetGroupManager()-
>SearchCloudGroups(searchParam, callback);
```

搜索云端群成员

功能描述

您可以通过群成员 ID、昵称、名片等信息搜索云端群成员。这一功能特别适用于需要快速查找特定群成员的场景，例如在大型群组中快速查找某个群成员信息。

> **说明：**
> - **群成员云端搜索功能仅 8.4 及以上版本支持。**
> - **该功能为增值功能，详细参见 价格说明 。**
> - **如果您没有开通该服务，调用接口会返回 60020 错误码。**

搜索云端群成员接口

您可以调用接口 `searchCloudGroupMembers` **（Java / Swift / Objective-C / C++）搜索云端群成员。** `V2TIMGroupMemberSearchParam` **的参数说明如下：**

参数	含义	说明
keywordList	关键字列表	关键字列表最多支持 5 个，keyword 会自动匹配群成员 ID、昵称、群名片。
keywordListMatchType	指定关键字列表匹配类型	可设置为 “或” 关系搜索，或 “与” 关系搜索。取值分别为 `V2TIM_KEYWORD_LIST_MATCH_TYPE_OR` 和 `V2TIM_KEYWORD_LIST_MATCH_TYPE_AND` 。默认为 “或” 关系搜索。
groupIDList	指定群 ID 列表	• 如果设置 `groupIDList` 为空，代表搜索全部群中的群成员，返回的结果会按照 `groupID` 进行分类； • 如果设置 `groupIDList` 不为空，代表搜索指定群中的群成员。
searchCount	搜索的数量	必须大于 0，最大支持 100，默认 20。
searchCursor	搜索的游标	搜索的起始位置，第一次填写空字符串，续拉时填写上一次返回的 `V2TIMGroupMemberSearchResult` 中的 `searchCursor` 。

群成员搜索结果类

群成员搜索结果类为 `V2TIMGroupMemberSearchResult` (Java / Swift / Objective-C / C++) 。参数说明如下：

参数	含义	说明
isFinished	搜索是否已完成	满足搜索条件的群成员列表是否已经全部返回。
totalCount	搜索结果总数	满足搜索条件的群成员总数量。
nextCursor	续拉的游标	调用搜索接口续拉时需要填的游标。
memberList	群成员列表	当前一次云端搜索返回的群成员列表。

示例代码如下：

Java

```
V2TIMGroupMemberSearchParam searchParam = new
V2TIMGroupMemberSearchParam();
searchParam.setKeywordList(keywordList);
```

```
searchParam.setKeywordListMatchType(param.V2TIM_KEYWORD_LIST_MATCH_TYPE_
OR);
searchParam.setGroupIDList(groupIDList);
searchParam.setSearchCount(20);
searchParam.setSearchCursor("");

V2TIMManager.getGroupManager().searchCloudGroupMembers(searchParam, new
V2TIMValueCallback<V2TIMGroupMemberSearchResult>() {
  @Override
  public void onSuccess(V2TIMGroupMemberSearchResult
groupMemberSearchResult) {
      // 搜索云端群成员成功
  }

  @Override
  public void onError(int code, String desc) {
      // 搜索云端群成员失败
  }
});
```

Swift

```
let param = V2TIMGroupMemberSearchParam()
param.keywordList = ["keyword1", "keyword2"];
param.keywordListMatchType = .V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.groupIDList= ["指定群 ID"];
param.searchCount = 20;
param.searchCursor = "";
V2TIMManager.shared.searchCloudGroupMembers(searchParam: param) {
searchResult in
    // 搜索云端群成员成功
} fail: { code, desc in
    // 搜索云端群成员失败
}
```

Objective-C

```
V2TIMGroupMemberSearchParam *param = [[V2TIMGroupMemberSearchParam
alloc] init];
param.keywordList = @[@"keyword1", @"keyword2"];
```

```
param.keywordListMatchType = V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.groupIDList= @[@"指定群 ID"];
param.searchCount = 20;
param.searchCursor = "";
[[V2TIMManager sharedInstance] searchCloudGroupMembers:param
succ:^(V2TIMGroupMemberSearchResult *searchResult) {
    // 搜索云端群成员成功
} fail:^(int code, NSString *desc) {
    // 搜索云端群成员失败
}];
```

C++

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

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

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

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

private:
    SuccessCallback success_callback_;
```

```
    ErrorCallback error_callback_;
};

V2TIMGroupMemberSearchParam param;
param.keywordList = keywordList;
param.keywordListMatchType = V2TIM_KEYWORD_LIST_MATCH_TYPE_OR;
param.groupIDList = groupIDList;
param.searchCount = 20;
param.searchCursor = "";

auto callback = new ValueCallback<V2TIMGroupMemberSearchResult>{};
callback->SetCallback(
    [=](const V2TIMGroupMemberSearchResult& groupMemberSearchResult) {
        // 搜索云端群成员成功
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // 搜索云端群成员失败
        delete callback;
    });

V2TIMManager::GetInstance()->GetGroupManager()-
>SearchCloudGroupMembers(param, callback);
```

交流与反馈

单击进入 IM 社群，享有专业工程师的支持，解决您的难题。

Web&小程序&uni-app

最近更新时间：2025-01-08 15:56:23

使用须知

1. **云端搜索插件能力全新升级，支持消息/用户/群/群成员搜索。**

   Web 端无 UI 集成

   - 搜索云端消息：searchCloudMessages
   - 搜索云端用户：searchCloudUsers
   - 搜索云端群组：searchCloudGroups
   - 搜索云端群成员：searchCloudGroupMembers

   Web 端含 UI 集成

   - 含 UI 集成目前仅支持搜索云端消息，其他类型搜索功能正在加急研发中，敬请期待。

   Android、iOS、Mac、Windows 云端搜索功能正在加急研发中，敬请期待。

2. **调用接口前，请前往 控制台 购买正式云端搜索插件或领取体验版并开通对应搜索功能，否则将调用失败。**



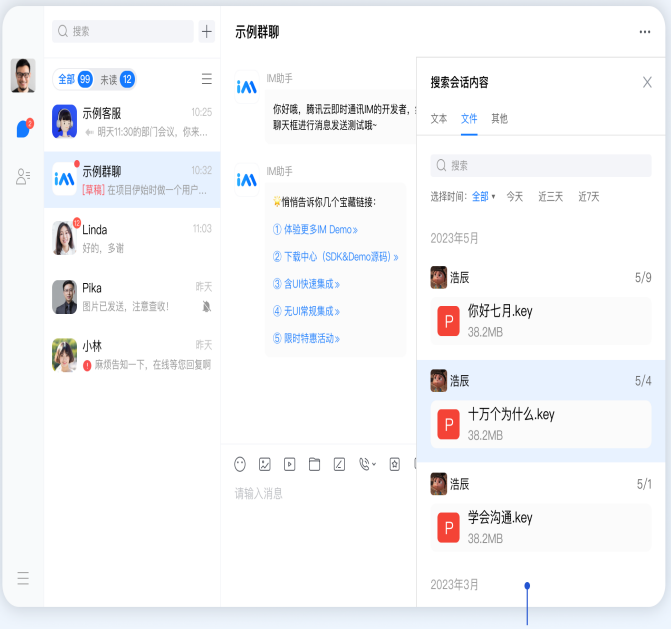
功能体验

点击体验消息云端搜索功能： Web&H5 Demo 。

TUIKit 消息云端搜索

全局搜索(TUISearch)





会话内搜索(TUISearch)



无 UI 集成

无 UI 使用消息云端搜索功能请参见 Chat SDK 接口文档：

- 搜索云端消息：searchCloudMessages
- 搜索云端用户：searchCloudUsers
- 搜索云端群组：searchCloudGroups
- 搜索云端群成员：searchCloudGroupMembers

含 UI 集成

快速集成消息云端搜索

Web&H5 Vue2&Vue3

步骤1: 集成TUIKit

- @tencentcloud/chat-uikit-vue ≥ 2.0.0，如未集成，请务必先根据 Vue2 & Vue3 TUIKit 快速集成指引 进行集成。

步骤2：控制台开通云端搜索插件

开通指引： 插件市场 > 概述与开通指引 。

> ⚠ 注意：
> 1. 每个插件限免费试用 1 次，有效期 7 天，试用结束后将停服，请提前购买。试用时，仅支持搜索开通云端搜索功能后产生的消息内容，不支持历史消息搜索；购买插件后，将自动同步历史消息，支持历史消息搜索。
> 2. 最大支持搜索近360天内的消息数据，与历史消息存储时间保持一致。

步骤3：搜索您的第一条消息

在完成 Vue2 & Vue3 TUIKit 快速集成指引 – 步骤6: 发送您的第一条消息 后，搜索您刚才发送的消息。

Uniapp Vue2&Vue3

步骤1: 集成TUIKit

- @tencentcloud/chat-uikit-uniapp ≥ 2.0.6，如未集成，请请务必先根据 Uniapp TUIKit 快速集成指引 进行集成。

步骤2：控制台开通云端搜索插件

开通指引： 插件市场 > 概述与开通指引 。

> ⚠ 注意：
> 每个插件限免费试用 1 次，有效期 7 天，试用结束后将停服，请提前购买。试用时，仅支持搜索开通云端搜索功能后产生的消息内容，不支持历史消息搜索；购买插件后，将自动同步历史消息，支持历史消息搜索。

步骤3：搜索您的第一条消息

在完成 Uniapp TUIKit 快速集成指引 – 步骤6: 发送您的第一条消息 后，搜索您刚才发送的消息。

独立引入消息云端搜索

> ⓘ 说明：

以上 快速集成消息云端搜索 中已默认包含消息云端搜索全部功能，无需重复引入。
如果您想独立引入 <TUISearch> 消息云端搜索，请参考以下教程。

Web&H5 Vue2&Vue3

前提条件

- @tencentcloud/chat-uikit-vue ⩾ 2.0.0，如未集成，请务必先根据 Vue2 & Vue3 TUIKit 快速集成指引 进行集成。

引入 <TUISearch>

在您所需要使用 **消息云端搜索** 功能的 `.vue` 界面，引入 **<TUISearch>** 。

<TUISearch> 参数说明

参数名	类型	说明
searchType	String	global：全局搜索（default）
		conversation：会话内搜索

<TUISearch> 效果展示

```
<TUISearch
searchType="global" />
```

```
<TUISearch
searchType="conversation
" />
```





使用 TUISearch

```
import { TUISearch } from "@tencentcloud/chat-uikit-vue";
// 全局搜索
<TUISearch searchType="global" />
// 会话内搜索
<TUISearch searchType="conversation" />
```

删除默认引入的 TUISearch

TUIKit 中默认集成 `<TUISearch>`，如您不按照默认集成方式使用，可在 `TUIKit/index.vue` 中，注释掉 `<TUISearch>` 即可。

Uniapp Vue2&Vue3

Uniapp TUISearch 支持两种方式引入：组件方式引入与界面方式引入。

前提条件

- **@tencentcloud/chat-uikit-uniapp ≥ 2.0.6**，如未集成，请请务必先根据 Uniapp TUIKit 快速集成指引 进行集成。

组件方式引入

在您所需要使用 **消息云端搜索** 功能的 `.vue` 界面，引入 **<TUISearch>** 。

<TUISearch> 参数说明

参数名	类型	说明
search Type	String	global：全局搜索
		conversation：会话内搜索（default）

<TUISearch> 效果展示

```
<TUISearch
searchType="global" />
```

```
<TUISearch
searchType="conversati
on" />
```





使用 TUISearch

```
// 以下路径仅为示例路径，请根据您项目自身路径进行调整
```

```
import { TUISearch } from
"/TUIKit/components/TUISearch/index.vue";
// 全局搜索
<TUISearch searchType="global" />
// 会话内搜索
<TUISearch searchType="conversation" />
```

删除默认引入的 TUISearch

TUIKit 中默认集成 `<TUISearch>` ，如您不按照默认集成方式使用，可在 `TUIKit/components/TUIConversation/index.vue` 中，注释掉 `<TUISearch>` 即可。

界面方式引入

在 pages.json 新增 TUISearch 页面

```
{
  "pages": [
    ...,
    {
      "path": "TUIKit/components/TUISearch/index",
      "style": {
        "navigationBarTitleText": "聊天记录"
      }
    }
  ],
  ...
}
```

跳转到 TUISearch 界面

```
uni.navigateTo({
   url: "/TUIKit/components/TUISearch/index",
});
```

高级指引

新增搜索消息类型

原“全局搜索”消息类型列表	新增后“全局搜索”消息类型列表
	

目录位置： `src/TUIKit/components/TUISearch/search-type-list.ts`

`searchMessageTypeList` **中包含了所有“搜索消息类型” Tab 定义，如需新增** `searchMessageTypeList` **未定义的搜索消息类型，请按照以下结构在** `searchMessageTypeList` **中进行新增：**

```
[keyName: string]: {
    key: string;// 消息搜索类型 key，请保持唯一性
    label: string;// 消息搜索类型渲染 label
    value: Array<string>;// 消息搜索类型实际搜索列表
 };

 // 例如，定义搜索自定义类型消息
export const searchMessageTypeList = {
    ...
    customMessage: {
      key: "customMessage",// 消息搜索类型 key，请保持唯一性
      label: "自定义",// 消息搜索类型渲染 label
      value: [TUIChatEngine.TYPES.MSG_CUSTOM],// 消息搜索类型实际搜索列表
    }
};
```

因为 TUIKit 使用 i18next 支持国际化，如您声明新的 label，请在 `src/TUIKit/locales/zh_cn/TUISearch.ts` **以及** `src/TUIKit/locales/en/TUISearch.ts` **增加相应的国际化词条进行翻译。**

如需将已定义的 `searchMessageTypeList` 中某类型增加到**全局搜索类型列表**或者**会话内搜索类型列表**，仅需将其 `key` 填入 `globalSearchTypeKeys` 或 `conversationSearchTypeKeys` 即可。

```
// 例如，将以上新增的 自定义消息 customMessage 应用到“全局搜索”消息类型列表
export const globalSearchTypeKeys = [..., "customMessage"];
// 例如，将以上新增的 自定义消息 customMessage 应用到“会话内搜索”消息类型列表
export const conversationSearchTypeKeys = [..., "customMessage"];
```

新增消息云端搜索时间范围

原“全局搜索”时间范围列表	新增后“全局搜索”时间范围列表
	

目录位置： `src/TUIKit/components/TUISearch/search-time-list.ts`

`searchMessageTimeList` 中包含了所有“搜索时间范围” Tab 定义，如需新增 `searchMessageTimeList` 未定义的搜索时间范围类型，请按照以下结构在 `searchMessageTimeList` 中进行新增：

```
[keyName: string]: {
    key: string;// 消息搜索时间范围 key，请保持唯一性
    label: string;// 消息搜索时间范围渲染 label
    value: {
      timePosition: number; // 消息搜索时间范围起始位置，默认为 0，从当前时间开始搜索
      timePeriod: number; // 从 timePosition 向前搜索的时间范围
    };
};

 // 例如，定义搜索“近两天“时间范围
export const searchMessageTimeList = {
```

```
    ...
    twoDay: {
      key: "twoDay",// 消息搜索时间范围 key，请保持唯一性
      label: "近两天",// 消息搜索时间范围渲染 label
      value: {
        timePosition: 0,// 消息搜索时间范围起始位置，默认为 0，从当前时间开始搜索
        timePeriod: 2 * oneDay,// 从 timePosition 向前搜索的时间范围
      },
  },
};
```

因为 TUIKit 使用 i18next 支持国际化，如您声明新的 label，请在 `src/TUIKit/locales/zh_cn/TUISearch.ts` 以及 `src/TUIKit/locales/en/TUISearch.ts` 增加相应的国际化词条进行翻译。

常见问题

最近更新时间：2025-02-20 10:35:23

计费说用

云端搜索插件为增值能力，使用前需购买云端搜索插件同时赠送用量，使用中会根据实际使用情况，产生对应用量，默认先抵扣赠送用量，超额产生后付费，具体费用详情请参见 增值服务计费说明 。

购买后如何开启?

购买后请前往 IM 控制台 > 消息服务 Chat > 插件市场，找到云端搜索插件，开通您希望搜索的功能。



> **注意：**
> 请按需开启对应开关，开启搜索开关后，会进行历史数据同步，为避免频繁开关影响后台数据同步，每个开关每7天限制开关一次。

云端搜索插件支持哪些内容搜索?

最新版本云端搜索插件，再除消息外，支持账号、群组、群成员搜索，同时提供 SDK 接入和 RestAPI 接入方式，全面满足用户的搜索场景和搜索需求。

云端搜索 RestAPI 和 SDK 接入区别是什么?

主要区别在于群成员搜索的范围，RestAPI 是针对全局群成员数据搜索，而 SDK 接入是用户加入的群组里面群成员数据搜索。

如果想让一些特殊用户和群组，不被搜索出来，应该怎么做?

RestAPI 里面提供了 隐藏搜索对象接口 ，通过该接口，可以将这些特殊用户和群组隐藏起来，不被搜索出来。

云端搜索存量的消息/用户/群组/群成员数据同步，一般要多久完成?

消息/用户/群组/群成员的存量数据，一般会在2天同步完成，同步完成后，控制台可以查看到已经同步标识。如果没有如期完成，请通过 提交工单 联系我们。

投票插件
效果展示

最近更新时间：2024-05-08 17:55:43

发起投票

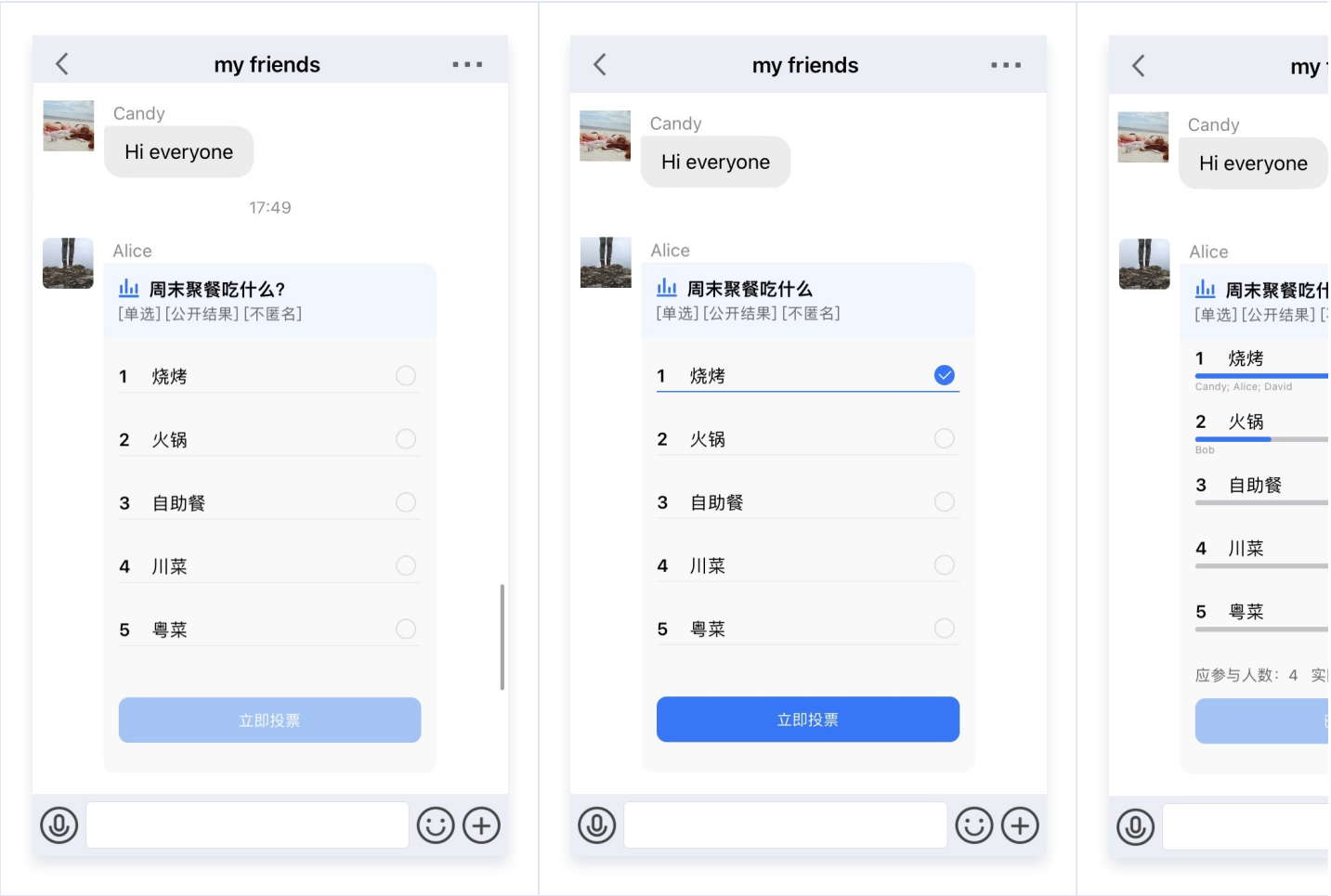
集成 `TUIPollPlugin` 成功后，`TUIChat` 输入栏的 + 菜单中会自动展示投票插件入口。发起人单击**投票**后，可以在接下来的发起投票界面，设置投票主题、选项，并进行相关设置，如下图所示：



参与人投票

发起人创建投票成功后，`TUIChat` 消息列表中会自动展示投票消息。群组里的其他成员，收到投票消息后，可以参与到投票中，如下图所示：

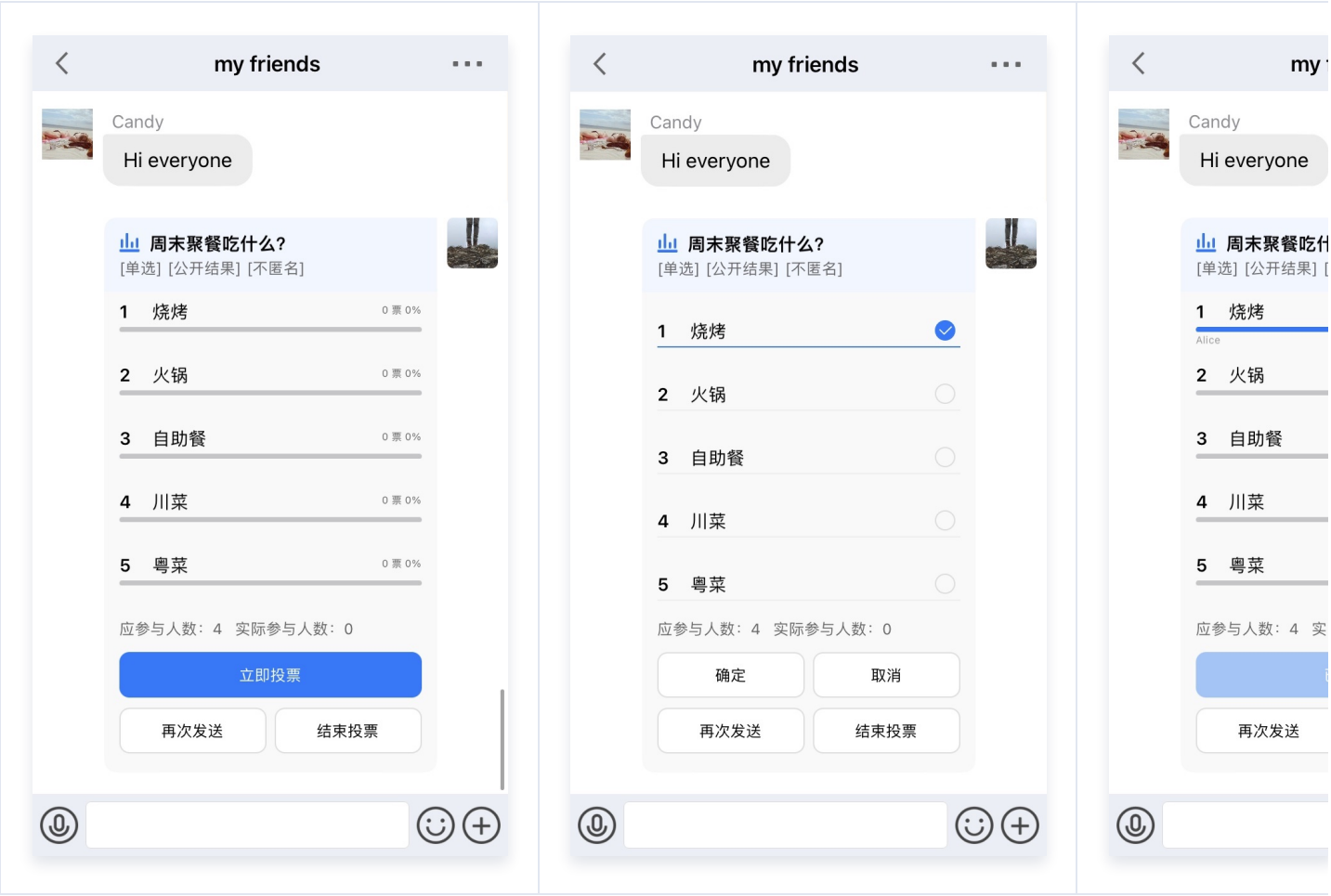
参与人新收到投票消息	参与人准备投票	参与人



发起人投票

发起人自己也可参与投票，如下图所示：

发起人新收到投票消息	发起人准备投票	发起人

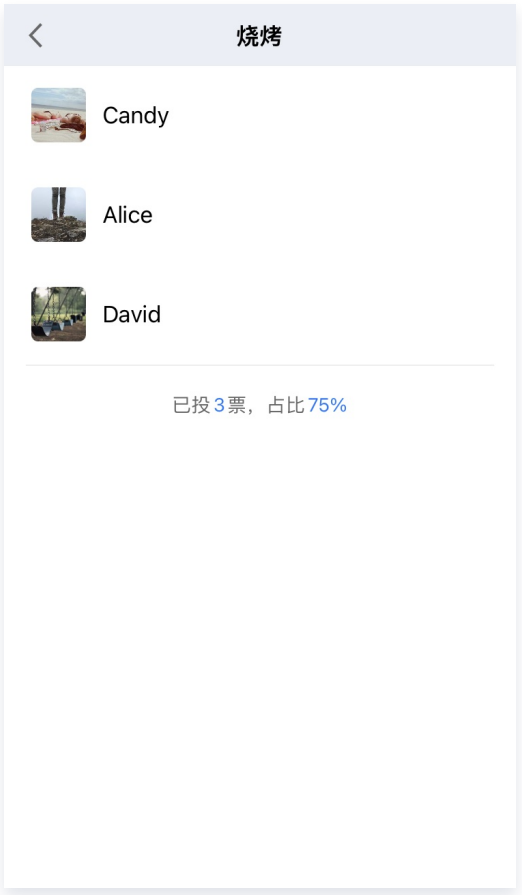


投票数据统计

投票结束或本人完成投票后，可以看到当前的投票数据统计。单击某一选项可查看详细的投票成员列表，如下图所示：

投票数据统计	选项成员列表





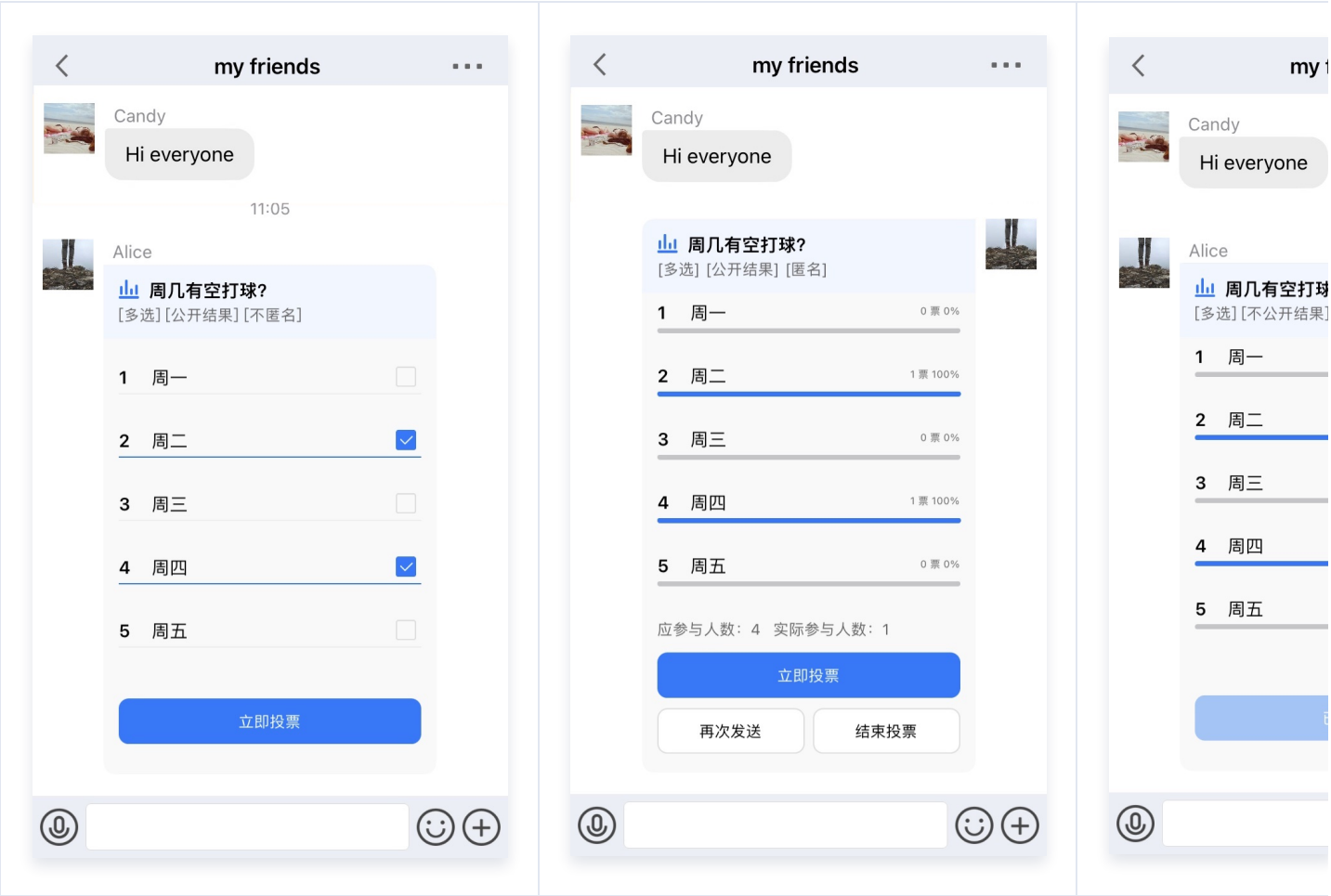
投票设置

为了满足不同的投票需求，`TUIPollPlugin` 开放了几个投票设置。不同的设置项组合，会产生不同的效果。例如：

- 开启**允许多选**，一个参与人可投多个选项。
- 开启**匿名投票**，投票发起人无法看到投票者的具体信息，只能看到票数统计。
- 关闭**公开结果**，只有投票发起人能看到投票数据统计，参与人无法得知。

如下图所示：

允许多选	发起人视角，匿名投票	参与人视角



说明

发起人不可以在发起投票后修改设置。

集成指引
Android & iOS

最近更新时间：2025-05-28 17:45:53

插件功能

从 7.1 版本开始，您可以集成投票插件 `TUIPollPlugin` 。投票插件是一个 UI 闭源库，依赖于 `TUIChat` 。集成后，您可以直接在 `TUIChat` 中发起群组投票、查看投票统计结果、参与其他人发起的投票等。

Android 集成

`TUIPollPlugin` 是闭源插件，需要通过 gradle 集成。找到 App 的 build.gradle，在 dependencies 中添加投票插件的依赖。

经典版

```
  dependencies {
    ...
  # 集成聊天功能
  api project(':tuichat')
  ...
  # 集成投票插件，从 7.1 版本开始支持
  api "com.tencent.imsdk:tuipoll-plugin:7.3.4358"
  ...
}
```

简约版

`TUIPollPlugin` 暂不支持简约版，敬请期待。

iOS 集成

`TUIPollPlugin` 和普通的 TUIKit 组件一样，都可以通过 CocoaPods 快速集成。详细的集成步骤可以参见 集成基础功能 。

对于投票插件来说，您只需要向 Podfile 中添加一行即可完成：

经典版

```
# Uncomment the next line to define a global platform for your
project
# ...
  # OC 语言版本
  # 集成聊天功能
  pod 'TUIChat/UI_Classic'
  ...
  # 集成投票插件，从 7.1 版本开始支持
  pod 'TUIPollPlugin'
  ...

  # Swift 语言版本，从 8.6.6870 版本开始支持
  # 集成聊天功能
  pod 'TUIChat_Swift/UI_Classic'
  ...
  # 集成投票插件
  pod 'TUIPollPlugin_Swift'
  ...

end
```

简约版

`TUIPollPlugin` 暂不支持简约版，敬请期待。

⚠ 注意：

- `TUIPollPlugin` 依赖于 `TUIChat`，单独集成 `TUIPollPlugin` 会使功能异常，无法正常展示投票界面。
- `TUIPollPlugin` 从 7.1 版本开始支持，使用投票插件需要将 `TUIChat` 升级到 7.1 及以上版本。

版权所有：腾讯云计算（北京）有限责任公司

flutter

最近更新时间：2024-05-08 14:48:21

插件功能

从tencent_cloud_chat_sdk 5.1.5版本之后，您可以集成腾讯云即时通信官方提供的投票插件 tencent_cloud_chat_vote_plugin，此插件为闭源插件，集成后，您可以在群里（Community 和 AVChatRoom 除外）集成投票功能。投票功能包括发起（单选、多选）投票、查看投票结果，参与投票等。Flutter 端与 Native 的投票能力互通。

环境与版本

本插件依赖插件以及环境

- Fluter 3.10.0及以上
- tencent_cloud_chat_sdk 5.1.5及以上

插件引入

通过 pub 可将投票插件 tencent_cloud_chat_vote_plugin 1.0.2 引入到项目中：

```
// 集成最新版本
pub add tencent_cloud_chat_vote_plugin

// 集成指定版本，在项目pubspec.yaml中dependencies字段加入
tencent_cloud_chat_vote_plugin: "version"
```

核心组件

- TencentCloudChatVoteCreate 创建投票
- TencentCloudChatVoteMessage 投票消息解析
- TencentCloudChatVoteDetail 投票详情展示

插件集成

初始化插件

```
// 放在登录IM后

await TencentCloudChatVotePlugin.initPlugin();
```

创建投票

用户单击投票按钮，可创建投票。

```
import 'package:example/config.dart';
import 'package:flutter/material.dart';
import
'package:tencent_cloud_chat_vote_plugin/components/vote_create/vote_crea
te.dart';

class VoteCreateExample extends StatefulWidget {
  const VoteCreateExample({super.key});

  @override
  State<StatefulWidget> createState() => VoteCreateExampleState();
}

class VoteCreateExampleState extends State {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("创建投票"),
      ),
      body: Container(
        padding: const EdgeInsets.all(16),
        child: TencentCloudChatVoteCreate(
          groupID: ExampleConfig.testGruopID,
          onCreateVoteSuccess: () {
            Navigator.pop(context);
          },
        ),
      ),
    );
  }
}
```

TencentCloudChatVoteCreate 参数说明

参数	说明
groupID	需要创建投票的群 ID，同 IM 群 ID，Community 和 AVChatRoom 除外。

onCreateVoteSuccess	创建投票成功回调。

投票消息解析

```
import 'package:example/config.dart';
import 'package:example/vote_detail_example.dart';
import 'package:flutter/material.dart';
import
'package:tencent_cloud_chat_sdk/enum/history_msg_get_type_enum.dart';
import 'package:tencent_cloud_chat_sdk/models/v2_tim_message.dart';
import
'package:tencent_cloud_chat_sdk/models/v2_tim_value_callback.dart';
import 'package:tencent_cloud_chat_sdk/tencent_im_sdk_plugin.dart';
import
'package:tencent_cloud_chat_vote_plugin/tencent_cloud_chat_vote_plugin.d
art';

class VoteMessageExample extends StatefulWidget {
  const VoteMessageExample({super.key});

  @override
  State<StatefulWidget> createState() => VoteMessageExampleState();
}

class VoteMessageExampleState extends State {
  V2TimMessage? message;
  getTestV2TimMessage() async {
    V2TimValueCallback<List<V2TimMessage>> messageListRes =
        await TencentImSDKPlugin.v2TIMManager
            .getMessageManager()
            .getHistoryMessageList(
              count: 1,
              groupID: ExampleConfig.testGruopID,
              getType: HistoryMsgGetTypeEnum.V2TIM_GET_CLOUD_OLDER_MSG,
            );
    if (messageListRes.code == 0) {
      if (messageListRes.data != null) {
        if (messageListRes.data!.isNotEmpty) {
          setState(() {
            message = messageListRes.data!.first;
          });
        }
```

```
      }
    }
  }

  bool isEnd = false;
  @override
  void initState() {
    super.initState();
    Future.delayed(
        const Duration(
          milliseconds: 300,
        ), () {
      setState(() {
        isEnd = true;
      });
    });
    // 页面运动结束再显示组件
    getTestV2TimMessage();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("投票消息体"),
      ),
      body: !isEnd
          ? Container()
          : message != null
              ? TencentCloudChatVoteMessage(
                  message: message!,
                  onTap: (
                    TencentCloudChatVoteDataOptoin option,
                    TencentCloudChatVoteLogic data,
                  ) {
                    print(data.voteData.toJson());
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => VoteDetailExample(
                          option: option,
                          data: data,
                        ),
```

```
                        ),
                    );
                  },
                )
              : const Center(
                  child: Text("未获取到正确的Message实例"),
                ),
    );
  }
}
```

TencentCloudChatVoteMessage参数说明

参数	说明
message	投票消息，V2TimMessage 类型。
onTap	单击投票回调，当投票为公开且实名时，可以打开群投票详情。

投票详情查看

```
import 'package:flutter/material.dart';
import
'package:tencent_cloud_chat_vote_plugin/tencent_cloud_chat_vote_plugin.d
art';

class VoteDetailExample extends StatelessWidget {
  final TencentCloudChatVoteDataOptoin option;
  final TencentCloudChatVoteLogic data;
  const VoteDetailExample({
    super.key,
    required this.option,
    required this.data,
  });

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(option.option),
      ),
```

```
        body: Padding(
          padding: const EdgeInsets.all(16),
          child: TencentCloudChatVoteDetail(
            option: option,
            data: data,
          ),
        ),
      );
    }
}
```

TencentCloudChatVoteDetail 参数说明

参数	说明
option	TencentCloudChatVoteDataOption 类型，投票详情数据，由 TencentCloudChatVoteMessage 单击时获取。
data	TencentCloudChatVoteLogic 类型由 TencentCloudChatVoteMessage 点击时获取。

联系我们

最近更新时间：2024-10-14 17:40:02

即时通信 IM 投票插件交流群

加入 - 腾讯云即时通信 IM 投票插件交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

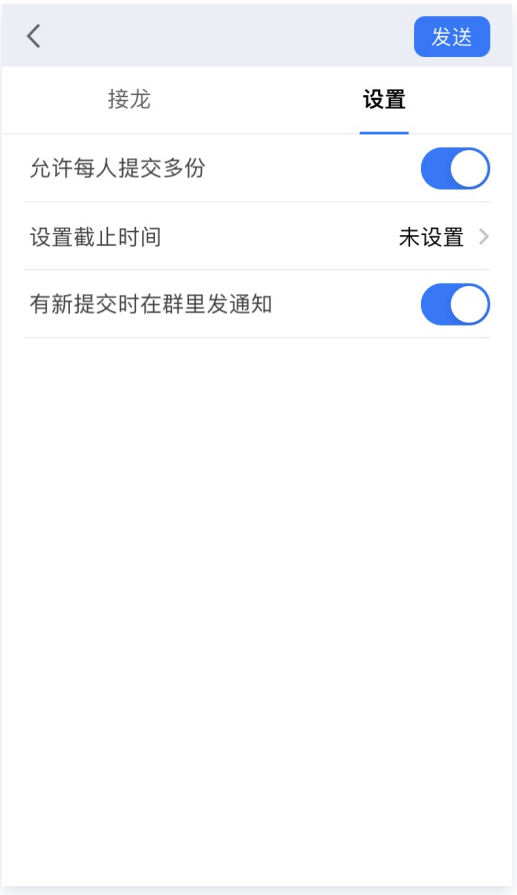
直接 点击进入交流群 。

接龙插件
效果展示

最近更新时间：2024-05-08 17:55:44

发起群接龙

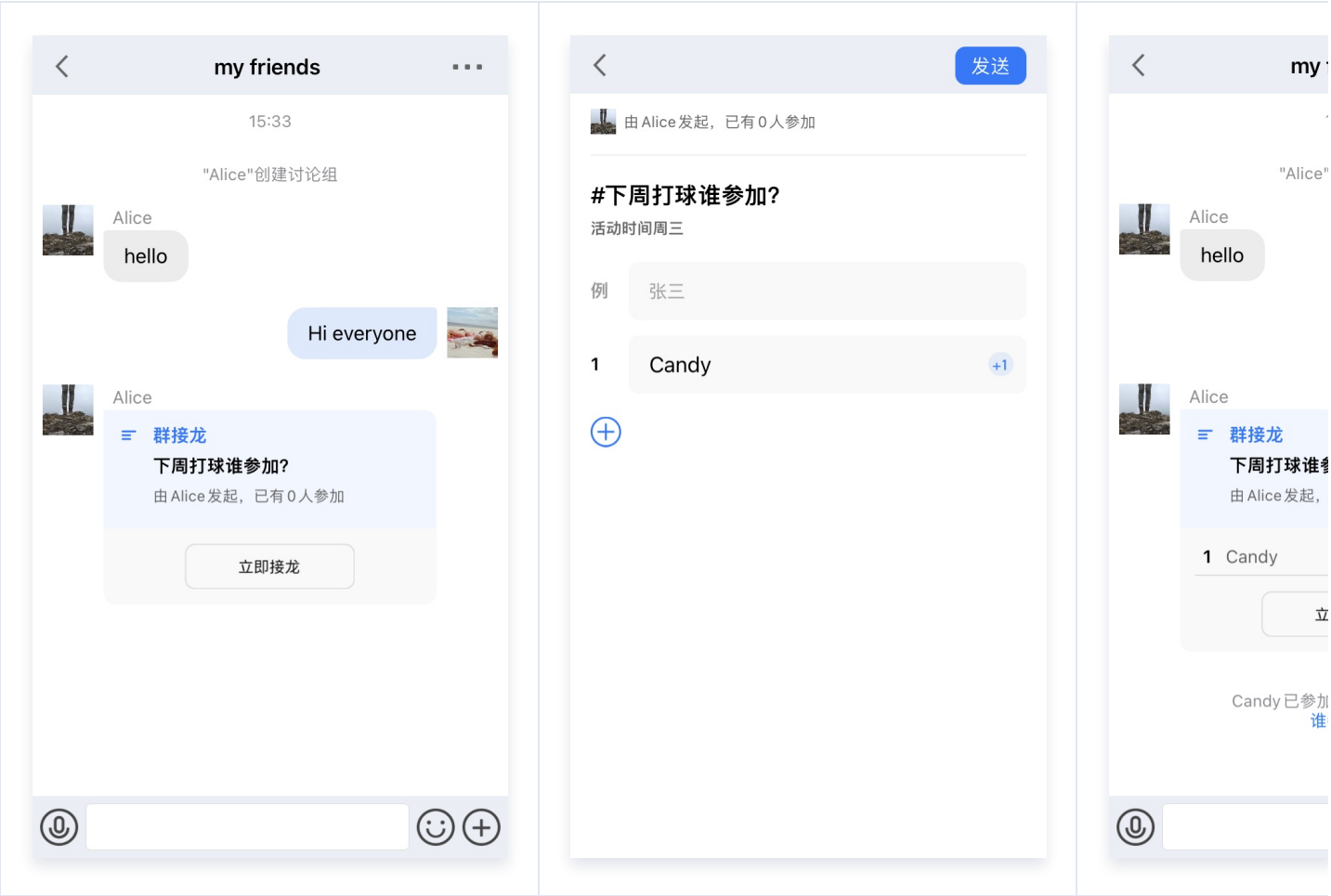
集成 `TUIGroupNotePlugin` 成功后，`TUIChat` 输入栏的 + 菜单中会自动展示群接龙插件入口。发起人单击接龙后，可以在接下来的创建接龙界面，填写接龙主题、说明、截止时间、示例，设置接龙特性等，如下图所示：

接龙信息填写	接龙设置
	

参与人接龙

发起人创建接龙成功后，`TUIChat` 消息列表中会自动展示接龙消息。群组里的其他成员，收到接龙消息后，可以参与到接龙中，如下图所示：

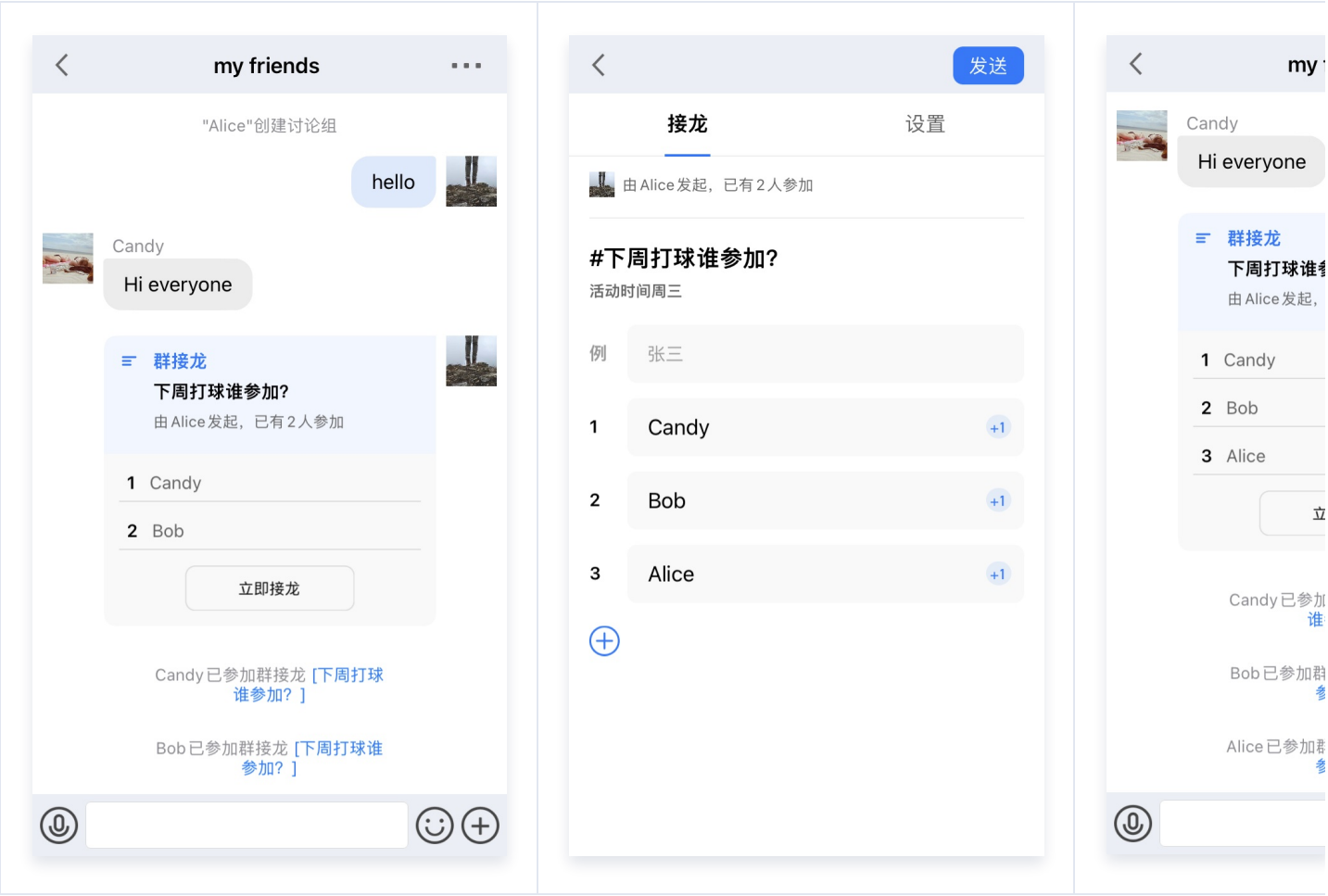
参与人新收到接龙消息	参与人准备接龙	参与人



发起人接龙

发起人自己也可参与接龙，如下图所示：

发起人新收到接龙消息	发起人准备接龙	发起人



接龙设置

为了满足不同的接龙需求，`TUIGroupNotePlugin` 开放了几个设置。例如：

- 开启**允许每人提交多份**，一个参与人可接龙多次。
- 设置**截止时间**，超过截止时间，所有人不能再参与接龙。
- 开启**有新提交时在群里发通知**，有人新增接龙条目后，群里会有通知消息提醒。

> **说明**
> 发起人可以在发起接龙后修改设置。

集成指引
Android & iOS

最近更新时间：2025-05-28 17:45:53

插件功能

从 7.1 版本开始，您可以集成群接龙插件 `TUIGroupNotePlugin` 。群接龙插件是一个 UI 闭源库，依赖于 `TUIChat` 。
集成后，您可以直接在 `TUIChat` 中发起群组接龙活动。

Android 集成

`TUIGroupNotePlugin` 是闭源插件，需要通过 gradle 集成。找到 App 的 build.gradle，在 dependencies 中添加投票插件的依赖。

经典版

```
  dependencies {
    ...
  # 集成聊天功能
  api project(':tuichat')
  ...
  # 集成接龙插件，从 7.1 版本开始支持
  api "com.tencent.imsdk:tuigroupnote-plugin:7.3.4358"
  ...
}
```

简约版

`TUIGroupNotePlugin` 暂不支持简约版，敬请期待。

iOS 集成

`TUIGroupNotePlugin` 和普通的 TUIKit 组件一样，都可以通过 CocoaPods 快速集成。详细的集成步骤可以参见 集成基础功能 。
对于接龙插件来说，您只需要向 Podfile 中添加一行即可完成：

经典版

```
# Uncommment the next line to define a global platform for your
project
# ...
  # OC 语言版本
  # 集成聊天功能
  pod 'TUIChat/UI_Classic'
  # 集成接龙插件，从 7.1 版本开始支持
  pod 'TUIGroupNotePlugin'

  # Swift 语言版本，从 8.6.6870 版本开始支持
  # 集成聊天功能
  pod 'TUIChat_Swift/UI_Classic'
  # 集成接龙插件
  pod 'TUIGroupNotePlugin_Swift'
  ...

  ...
end
```

简约版

`TUIGroupNotePlugin` 暂不支持简约版，敬请期待。

⚠ 注意

- `TUIGroupNotePlugin` 依赖于 `TUIChat` ，单独集成 `TUIGroupNotePlugin` 会使功能异常，无法正常展示接龙界面。
- `TUIGroupNotePlugin` 从 7.1 版本开始支持，使用接龙插件需要将 `TUIChat` 升级到 7.1 及以上版本。

版权所有：腾讯云计算（北京）有限责任公司

版权所有：腾讯云计算（北京）有限责任公司

联系我们

最近更新时间：2024-10-14 17:40:02

即时通信 IM 接龙插件交流群

加入 - 腾讯云即时通信 IM 接龙插件交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

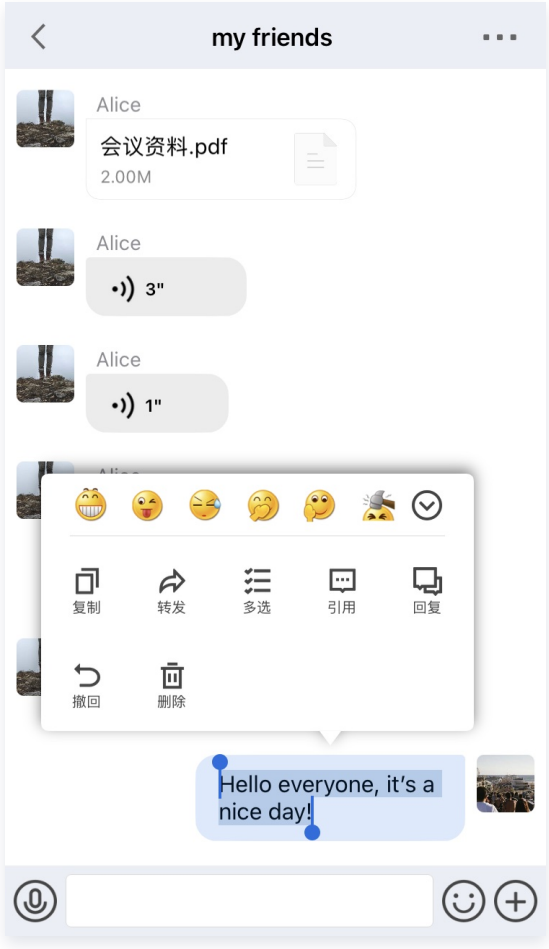
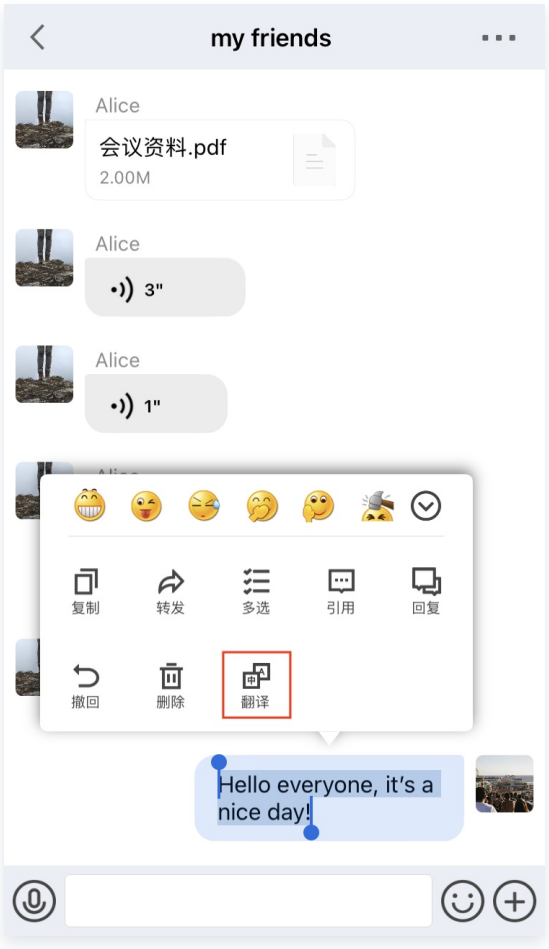
直接 点击进入交流群 。

文本消息翻译插件
效果展示

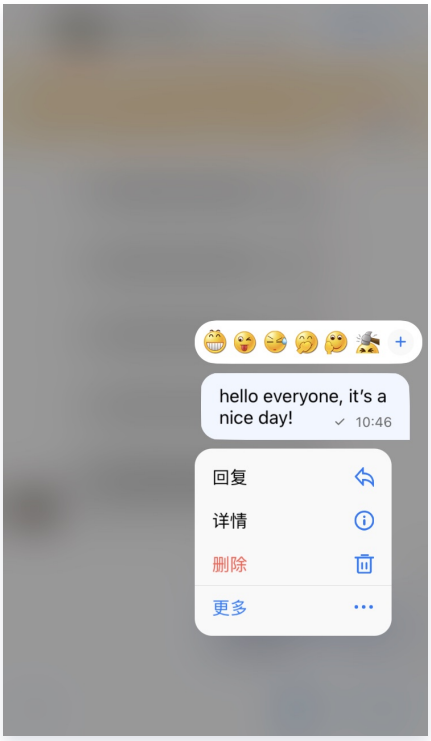
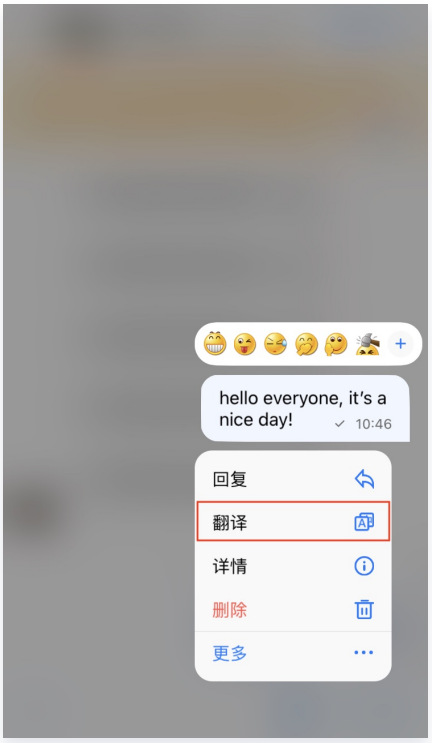
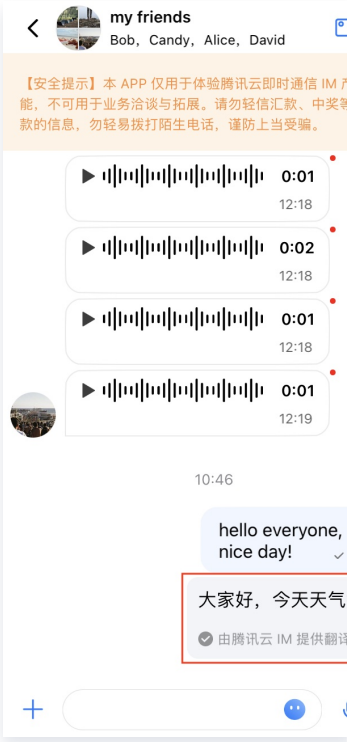
最近更新时间：2024-05-08 17:55:45

“集成前”与“集成后”效果对比

经典版

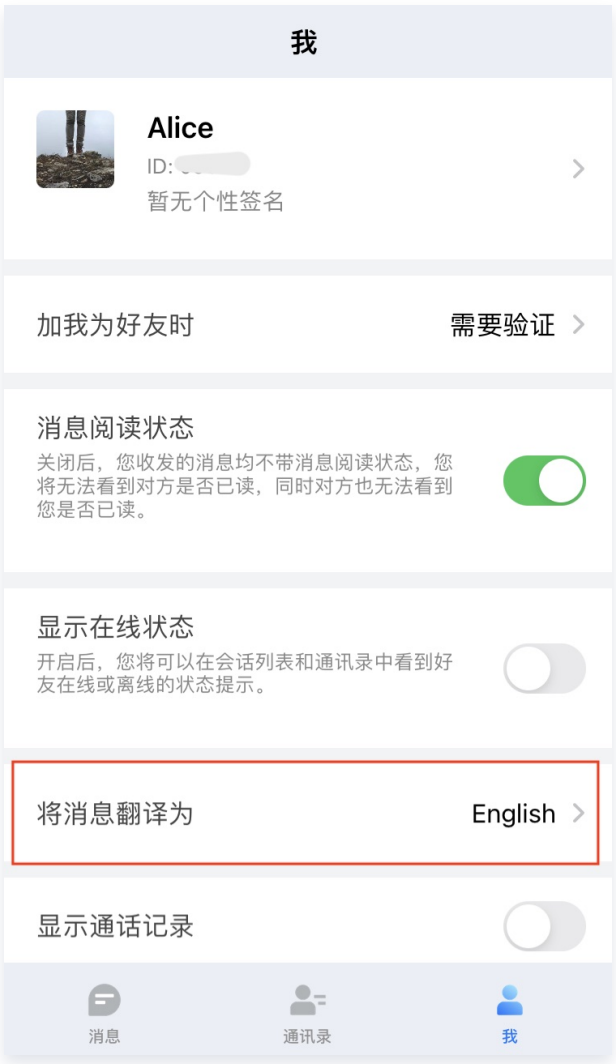
未集成翻译插件，不显示翻译按钮	集成了翻译插件，显示翻译按钮
	

简约版

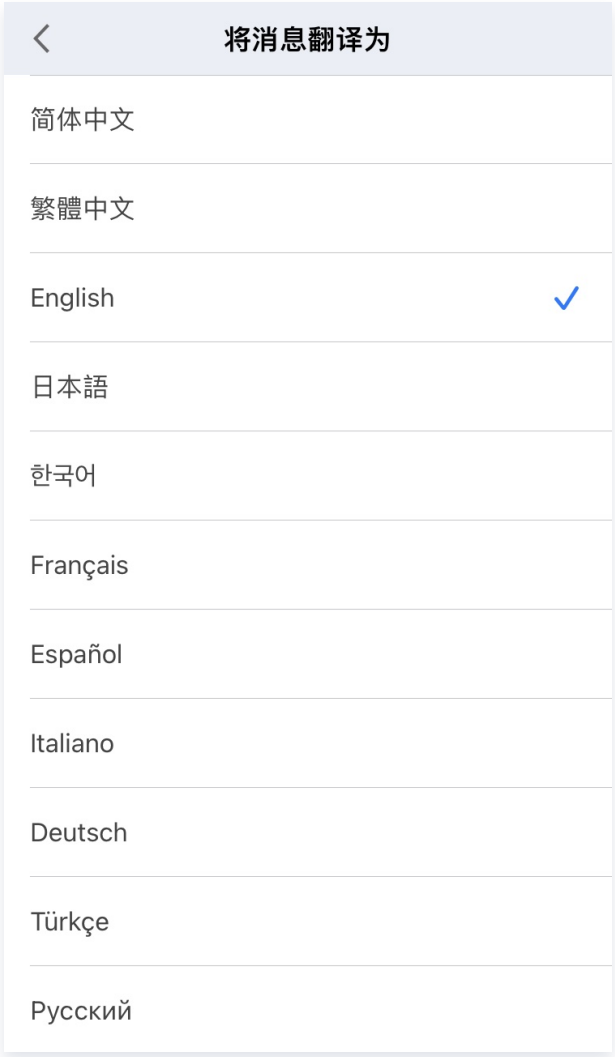
未集成翻译插件，不显示翻译按钮	集成了翻译插件，显示翻译按钮	文本消息翻译效果
		

设置入口及语言选择页面如下所示：

目标语言设置入口


我
Alice
ID:
暂无个性签名
加我为好友时
需要验证
消息阅读状态
关闭后，您收发的消息均不带消息阅读状态，您将无法看到对方是否已读，同时对方也无法看到您是否已读。
显示在线状态
开启后，您将可以在会话列表和通讯录中看到好友在线或离线的状态提示。
将消息翻译为
English
显示通话记录
消息
通讯录
我

目标语言选择页



注意：

- `TUITranslationPlugin` 依赖于 `TUIChat`，不能单独集成。
- 仅支持文本消息、文本类的引用或回复消息，图片、语音、视频、文件、表情、自定义消息等不支持翻译。
- 并不是任意源语言都能翻译成设定的目标语言。例如，英语可以翻译成印地语，但是中文不可以。目前已经支持的翻译语言请参见《文本翻译语言支持》。如果翻译失败，请查询该文档切换源语言或目标语言。

集成指引
Android & iOS

最近更新时间：2025-02-20 10:35:23

插件功能

当您进入了聊天界面后，可以手动长按消息列表中的文本消息 item，在出现的菜单中，单击翻译按钮，翻译文本。

> **注意：**
> - 文本消息翻译功能由插件提供，使用需集成 `TUITranslationPlugin`，7.2 及以上版本支持。
> - 文本翻译费用，请参考文档： 增值服务计费说明 查询详情。申请开通可参见 概述及开通指引 。
> - 为了方便用户测试功能，当您的账号在控制台首次单击**免费试用**时，会赠送2500条消息量，抵扣范围为此账号下首次试用消息翻译的应用用量。
>   - 试用期间免费用量用尽后，超量部分将按照套餐包外超量费用规则收取费用，免费用量有效期为一个月，到期自动失效。
>   - 若您测试多次申请消息翻译测试，或者测试时间超过一个月，超量部分都将按照套餐包外超量费用规则收取费用。

- 如果您不需要翻译功能，不集成该插件即可。长按文本消息时，不会显示翻译按钮。
- 如果您需要翻译功能，需集成 `TUIChat` 和 `TUITranslationPlugin`。集成方法请参见《集成基础功能》（Android/iOS）。集成后不需要进行任何设置，长按文本消息时，自动显示翻译按钮。

集成 `TUITranslationPlugin` 后，您还可以在**我**界面设置翻译的目标语言。默认目标语言是当前 `TUIChat` 所使用的语言。

> **注意：**
> - `TUITranslationPlugin` 依赖于 `TUIChat`，不能单独集成。
> - 仅支持文本消息、文本类的引用或回复消息，图片、语音、视频、文件、表情、自定义消息等不支持翻译。
> - 并不是任意源语言都能翻译成设定的目标语言。例如，英语可以翻译成印地语，但是中文不可以。目前已经支持的翻译语言请参见 《文本翻译语言支持》。如果翻译失败，请查询该文档切换源语言或目标语言。

联系我们

最近更新时间：2024-10-14 17:40:02

即时通信 IM 文本消息翻译插件交流群

加入 - 腾讯云即时通信 IM 文本消息翻译插件交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

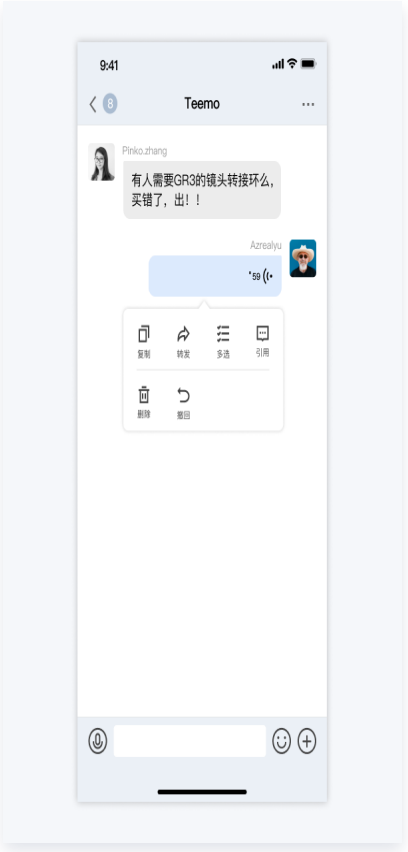
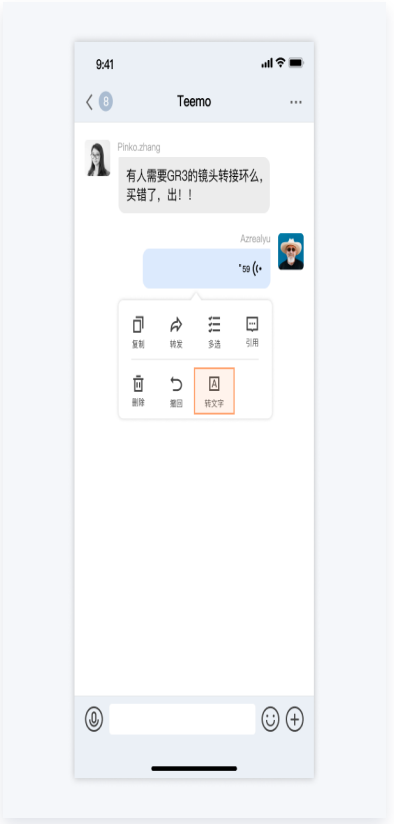
直接 点击进入交流群 。

语音转文字插件
效果展示

最近更新时间：2024-09-03 16:44:01

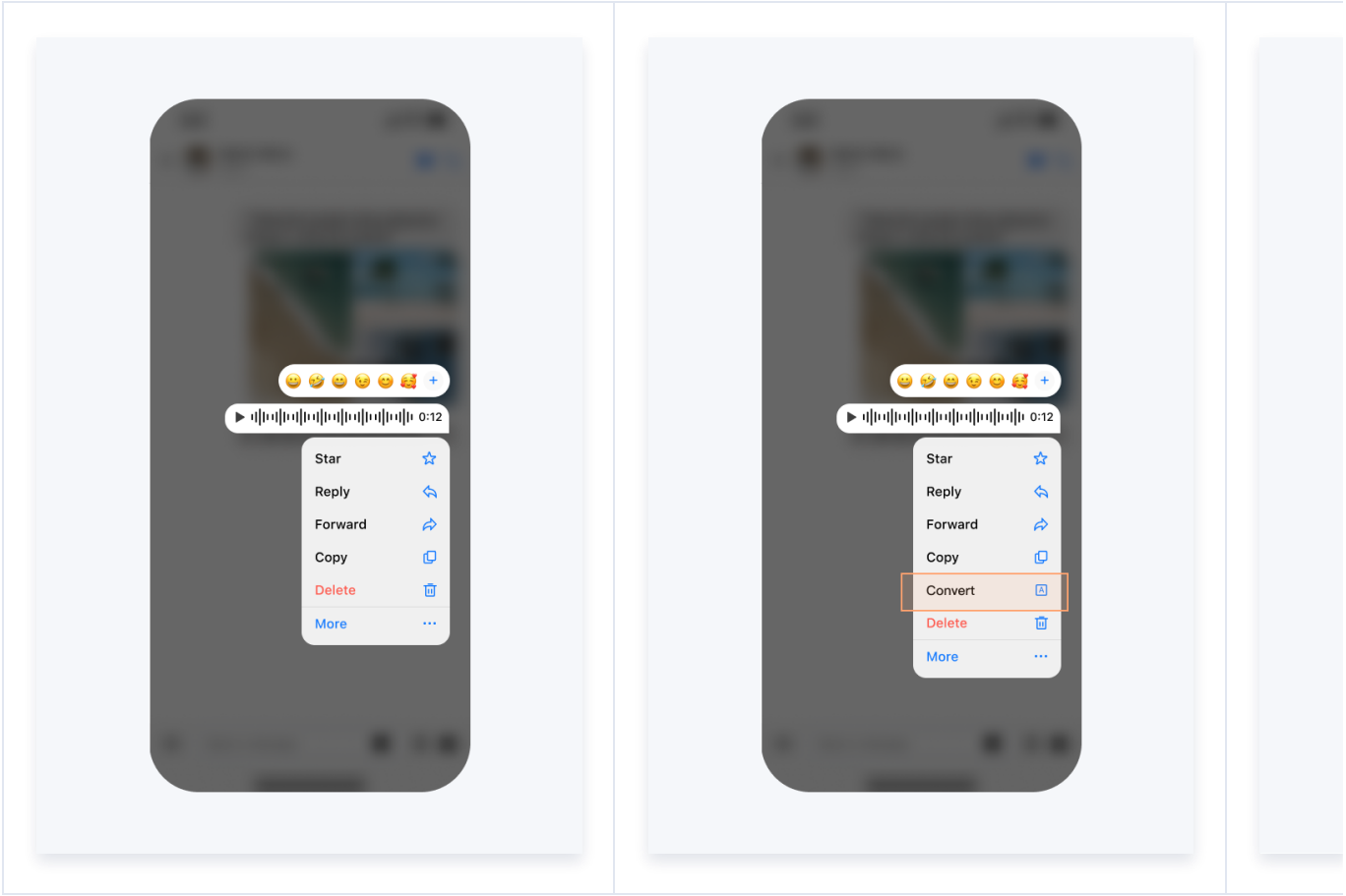
“集成前”与“集成后”效果对比

经典版

未集成转文字插件，不显示转文字按钮	集成了转文字插件，显示转文字按钮	语音转文字效果
		

简约版

未集成转文字插件，不显示转文字按钮	集成了转文字插件，显示转文字按钮	



说明：

- `TUIVoiceToTextPlugin` 依赖于 `TUIChat`，不能单独集成。
- 仅支持语音消息，文本、图片、视频、文件、表情、自定义消息等不支持转文字。
- 目前已经支持的源语言有中文、粤语、英文和日文，具体请参见《语言支持》。

集成指引
Android & iOS

最近更新时间：2025-02-20 10:35:23

插件说明

语音转文字功能：当您进入了聊天界面后，可以手动长按消息列表中的语音消息气泡，在出现的菜单中，单击**转文字**。

> ⚠ **注意：**
> - 语音转文字功能由插件提供，使用需集成 `TUIVoiceToTextPlugin`，7.5 及以上版本支持。
> - “语音转文字”功能已结束内测，处于付费阶段。具体费用请参见文档： 增值服务计费说明 。申请开通可参见 概述及开通指引 。
> - 为了方便用户测试功能，当您的账号在控制台首次单击**免费试用**时，会赠送2500条消息量，抵扣范围为此账号下首次试用语音转文字的应用用量。
>   - 试用期间免费用量用尽后，超量部分将按照套餐包外超量费用规则收取费用，免费用量有效期为一个月，到期自动失效。
>   - 若您测试多次申请语音转文字测试，或者测试时间超过一个月，超量部分都将按照套餐包外超量费用规则收取费用。

集成插件

从 7.5 版本开始，语音转文字功能由插件 `TUIVoiceToTextPlugin` 提供。

- 如果您不需要语音转文字功能，不集成该插件即可。长按语音消息时，不会显示转文字按钮。
- 如果您需要语音转文字功能，需集成 `TUIChat` 和 `TUIVoiceToTextPlugin`。集成方法请参见《集成 TUIKit》（ Android / iOS ）。集成后不需要进行任何设置，长按语音消息时，自动显示转文字按钮。

> ⚠ **注意：**
> - `TUIVoiceToTextPlugin` 依赖于 `TUIChat`，不能单独集成。
> - 仅支持语音消息，文本、图片、视频、文件、表情、自定义消息等不支持转文字。
> - 目前已经支持的源语言有中文、粤语、英文和日文，具体请参见 《语言支持》 。

Web & H5 & Uniapp

最近更新时间：2025-02-20 10:35:23

插件描述

TUIKit 从 v2.1.2 版本之后开始提供**语音转文字功能**，TUIChat 组件中的消息工具支持对语音消息转换成文字，目前可以识别普通话、粤语、英语、日语并转换为对应的文字。
语音转文字后，转换的结果会在本次登录中持续保留，切换会话后转换结果依然保存。

注意：

- “语音转文字”功能已结束内测，处于付费阶段。具体费用请参见文档： 增值服务计费说明 。申请开通可参见 概述及开通指引 。
- 可以通过 在线示例 进行功能抢先体验，更多平台体验可浏览对应的 体验 Demo 。



功能介绍

如何设置语音转文字的源语言

语音转文字的源语言默认情况下，能够自动识别普通话、粤语、英语，因此在简单场景下用户不需要手动设置源语言。当然，为了保证识别的精确性，也可以手动设置识别的源语言，目前支持以下四个选项：

- zh (cmn-Hans-CN)，中文通用。
- en-US，英语。
- yue-Hant-HK，粤语。
- ja-JP，日语。

具体修改方法是在文件 `TUIKit/components/TUIChat/utils/convertVoiceToText.ts` 中，修改 `TUIChatService.convertVoiceToText` 方法调用的入参（以日语为例），添加 `language` 属性：

```
const response: IChatResponese<{ result: string }> = await
TUIChatService.convertVoiceToText({
  message: msg,
  language: 'ja-JP', // add this new line
});
```

扩展资料

以下内容仅为辅助阅读资料，语音转文字以及相关功能已在 TUIKit 2.1.1 之后实现，不需要用户手动实现。

如何查询当前套餐是否支持功能

通过 `TUIStore` 查询 `StoreName.APP` 中 `enabledVoiceToText` 这个 key，得到的布尔值即是开通结果。

```
const enable = TUIStore.getData(StoreName.APP, 'enabledVoiceToText');
if (enable) {
  // ...
}
```

如何移除语音转文字功能

在文件 `TUIKit/components/TUIChat/message-list/message-tool/index.vue` 中，删除 `actionItems` 中 key 为 `convert` 的对象即可。

```
{
    key: 'convert',
    text: TUITranslateService.t('TUIChat.转文字'),
    visible: false,
```

```
    iconUrl: convertText,
    renderCondition() {},
    clickEvent: convertVoiceToText,
}
```

联系我们

最近更新时间：2024-09-03 16:44:01

时通信 IM 语音转文字插件交流群

加入 - 腾讯云即时通信 IM 语音转文字插件交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

直接 点击进入交流群 。

会话分组插件
效果展示

最近更新时间：2024-10-14 14:29:31

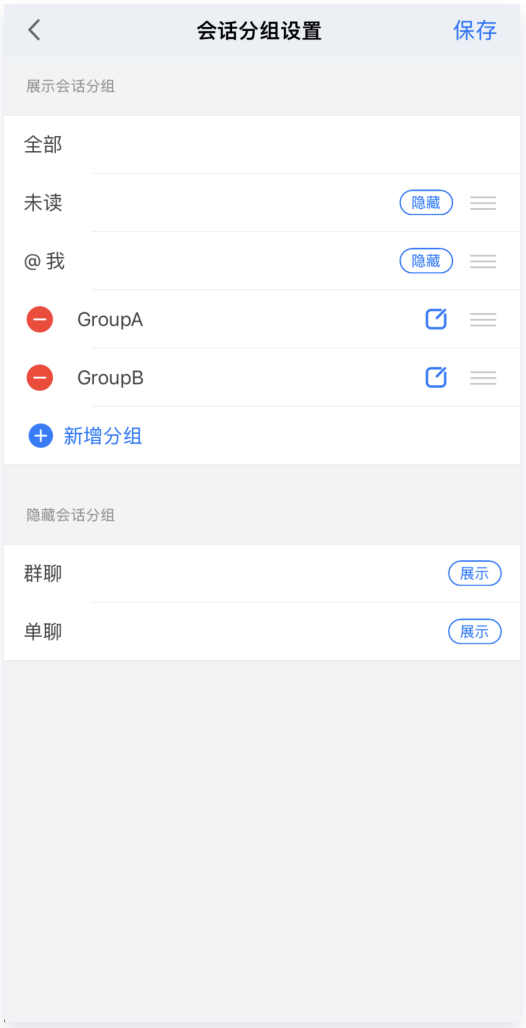
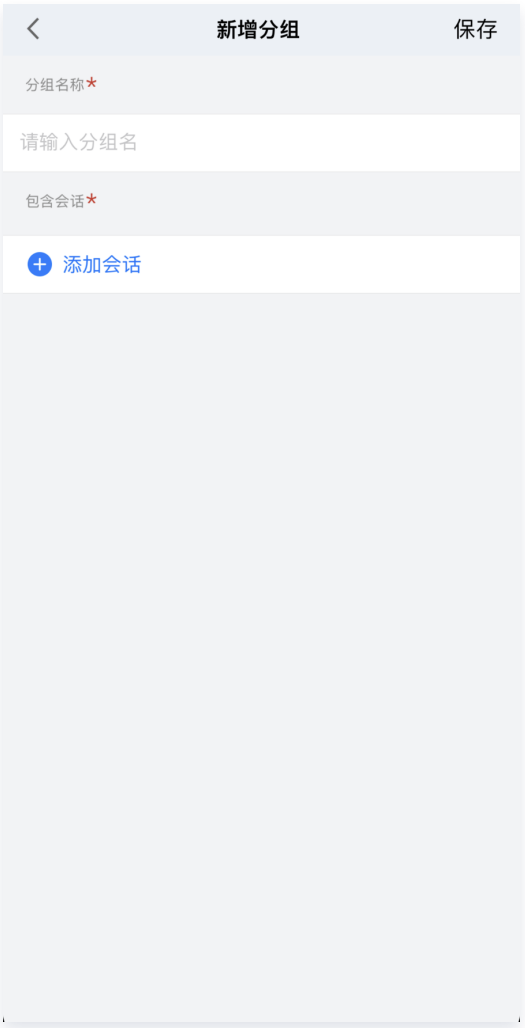
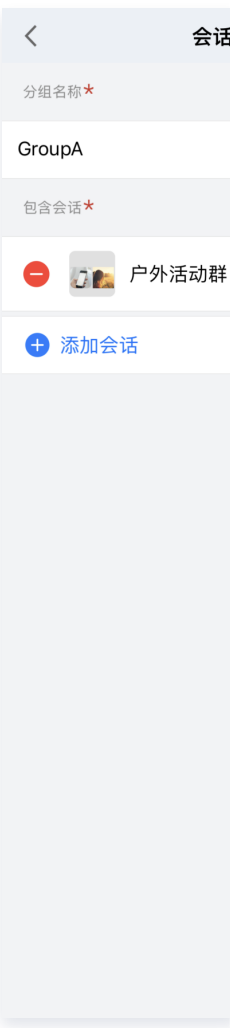
展示会话分组

集成 `TUIConversationGroupPlugin` 成功后，在 `TUIConversation` 会话列表上方会展示会话分组管理 `View`，里面由各个分组 `Item` 组成，每个分组 `Item` 包含了分组名和分组未读数信息。如果点击分组管理 `View` 右边的 `三` 按钮，就会弹出分组列表界面，如下图所示：

会话分组管理 View	会话分组列表
	

管理会话分组

在会话分组列表界面，点击右上角的“管理”按钮，就可以跳转到分组管理界面，在分组管理界面您可以创建、删除、编辑、隐藏分组，也能对分组进行重排序，如下图所示：

管理会话分组	创建会话分组	编辑
		

会话分组操作

在会话列表界面，您可以左滑会话 Cell 对其进行分组操作，例如添加会话到分组，把会话移动到其他分组，创建一个包含该会话的新分组等。

添加会话分组	选择会话分组	创建



注意：

会话分组栏默认会展示“全部”、“未读”、“@我”三个常驻分组，如果不需要某个分组，可以在分组管理界面隐藏对应分组。

集成指引
Android & iOS

最近更新时间：2025-05-28 17:45:53

插件功能

从 7.3 版本开始，您可以集成会话分组插件 `TUIConversationGroupPlugin` 。会话分组插件是一个 UI 闭源库，依赖于 `TUIConversation` 。集成后，您可以创建分组、删除分组、编辑分组、隐藏分组、对分组进行重排序。

> **注意：**
> 1. 本插件仅经典版 UIKit 支持，简约版 UIKit 不支持。
> 2. 本插件依赖于 TUIConversation（需要将 TUIConversation 升级到 7.3 及以上版本），单独集成会使功能异常，无法正常展示会话分组界面。

Android 集成

`TUIConversationGroupPlugin` 是闭源插件，需要通过 gradle 集成。找到 App 的 build.gradle，在 dependencies 中添加分组插件的依赖。

经典版

> **注意：**
> 如果使用会话分组功能，需集成 TUIConversationFragmentContainer 而非 TUIConversationFragment。

```
  dependencies {
    ...
    # 集成会话功能
    api project(':tuiconversation')
    ...
    # 集成会话分组插件，从 7.3 版本开始支持
    api "com.tencent.imsdk:tuiconversationgroup-plugin:7.3.4358"
    ...
}
```

简约版

TUIConversationGroupPlugin 暂不支持简约版，敬请期待。

iOS 集成

TUIConversationGroupPlugin 和普通的 TUIKit 组件一样，都可以通过 CocoaPods 快速集成。详细的集成步骤可以参见 集成基础功能 。
对于会话分组插件来说，您只需要向 Podfile 中添加一行即可完成：

经典版

```
# Uncomment the next line to define a global platform for your
project
# ...

  # OC 语言版本
  # 集成会话功能，仅经典版支持
  pod 'TUIConversation/UI_Classic'
  ...
  # 集成会话分组插件，从 7.3 版本开始支持
  pod 'TUIConversationGroupPlugin'
  ...

  # Swift 语言版本，从 8.6.6870 版本开始支持
  # 集成会话功能，仅经典版支持
  pod 'TUIConversation_Swift/UI_Classic'
  ...
  # 集成会话分组插件
  pod 'TUIConversationGroupPlugin_Swift'
  ...
end
```

简约版

`TUIConversationGroupPlugin` **暂不支持简约版，敬请期待。**

版权所有：腾讯云计算（北京）有限责任公司

联系我们

最近更新时间：2024-10-14 17:40:02

即时通信 IM 会话分组插件交流群

加入 - 腾讯云即时通信 IM 会话分组插件交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

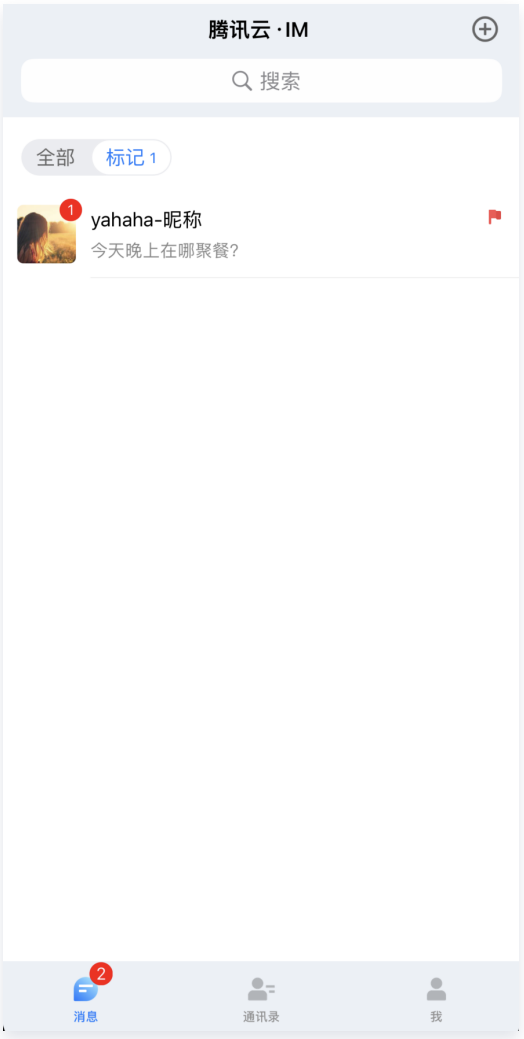
直接 点击进入交流群 。

会话标记插件
效果展示

最近更新时间：2024-10-14 11:01:32

标记会话

集成 `TUIConversationMarkPlugin` 成功后，在 `TUIConversation` 会话列表界面左滑 Cell 就能标记一个会话，当会话被标记后，会在会话列表上方展示“标记”分组栏，如下图所示：

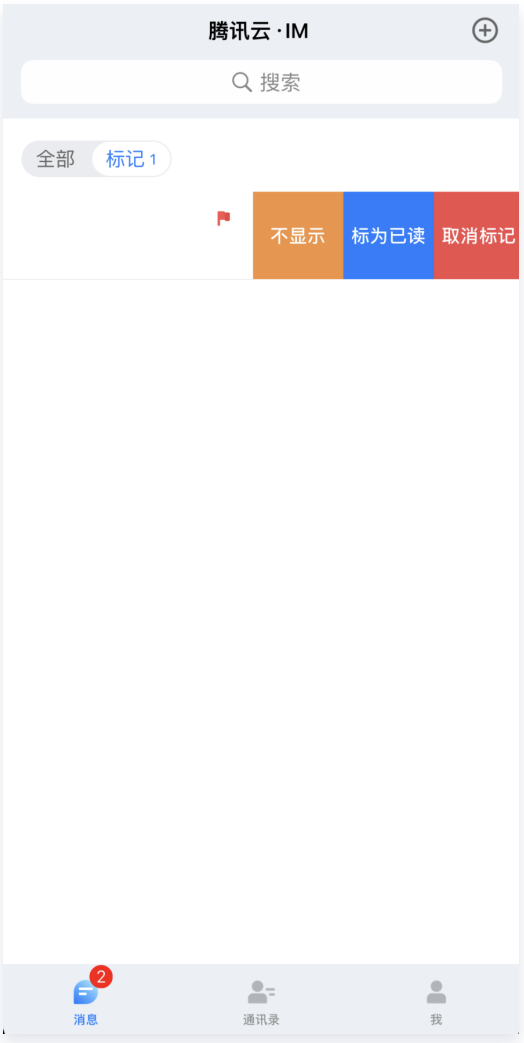
标记会话	“标记”分组
	

取消标记会话

如果一个会话已经被标记，当对该会话 Cell 左滑时，可以取消标记会话，如下图所示：

在“全部”分组取消标记会话	在“标记”分组取消标记会话





注意：

如果从来没有标记过会话，“标记”分组默认不展示，如果标记过会话，即使该会话已经被取消标记，“标记”分组也会一直展示。

集成指引
Android & iOS

最近更新时间：2025-05-28 17:45:53

插件功能

从 7.3 版本开始，您可以集成会话标记插件 `TUIConversationMarkPlugin` 。会话标记插件是一个 UI 闭源库，依赖于 `TUIConversation` 。集成后，您可以在 `TUIConversation` 中标记会话、取消标记会话。

> **注意：**
> 1. 本插件仅经典版 UIKit 支持，简约版 UIKit 不支持。
> 2. 本插件依赖于 TUIConversation（需要将 TUIConversation 升级到 7.3 及以上版本），单独集成会使功能异常，无法正常展示会话标记界面。

Android 集成

`TUIConversationMarkPlugin` 是闭源插件，需要通过 gradle 集成。找到 App 的 build.gradle，在 dependencies 中添加标记插件的依赖。

经典版

```
dependencies {
    ...
    # 集成会话功能
    api project(':tuiconversation')
    ...
    # 集成会话标记插件，从 7.3 版本开始支持
    api "com.tencent.imsdk:tuiconversationmark-plugin:7.3.4358"
    ...
}
```

简约版

TUIConversationMarkPlugin 暂不支持简约版，敬请期待。

iOS 集成

TUIConversationMarkPlugin 和普通的 TUIKit 组件一样，都可以通过 CocoaPods 快速集成。详细的集成步骤可以参见 集成基础功能 。
对于会话标记插件来说，您只需要向 Podfile 中添加一行即可完成：

经典版

```
# Uncomment the next line to define a global platform for your
project
# ...
  # OC 语言版本
  # 集成会话功能
  pod 'TUIConversation/UI_Classic'
  ...
  # 集成会话标记插件，从 7.3 版本开始支持
  pod 'TUIConversationMarkPlugin'
  ...

  # Swift 语言版本，从 8.6.6870 版本开始支持
  # 集成会话功能
  pod 'TUIConversation_Swift/UI_Classic'
  ...
  # 集成会话标记插件
  pod 'TUIConversationMarkPlugin_Swift'
  ...

end
```

简约版

`TUIConversationMarkPlugin` **暂不支持简约版，敬请期待。**

版权所有：腾讯云计算（北京）有限责任公司

联系我们

最近更新时间：2024-10-14 17:40:02

即时通信 IM 会话标记插件交流群

加入 - 腾讯云即时通信 IM 会话标记插件交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

直接 点击进入交流群 。