即时通信
IMLib / IMKit
iOS 5.X

2024-08-30

开发指导

更新时间:2024-08-30

欢迎使用融云即时通讯。本页面简单介绍了融云即时通讯架构、服务能力和 SDK 产品。

架构与服务

融云提供的即时通讯服务，不需要在 App 之外建立并行的用户体系，不用同步 App 下用户信息到融云，不影响 App 现有的系统架构与帐号体系，与现有业务体系能够实现完美融合。

融云的架构设计特点：

- 无需改变现有 App 的架构，直接嵌入现有代码框架中；
- 无需改变现有 App Server 的架构，独立部署一份用于用户授权的 Service 即可；
- 专注于提供通讯能力，使用私有的二进制通信协议，消息轻量、有序、不丢消息；
- 安全的身份认证和授权方式，无需担心 SDK 能力滥用（盗用身份的垃圾消息、垃圾群发）问题。

融云即时通讯产品支持单聊、群聊、超级群、聊天室 多种业务形态，提供丰富的客户端和服务端接口，大部分能力支持开箱即用。

业务类型介绍

单聊（Private）业务即一对一聊天。普通群组（Group）业务类似微信的群组。超级群与聊天室业务均不设用户总数上限。超级群（UltraGroup）[1] 类似 Discord，提供了一种新的群组业务形态，在超级群中提供公有/私有频道、用户组等功能，适用于构建超级社区。聊天室（Chatroom）只有在线用户可接收消息，广泛适用于直播、社区、游戏、广场交友、兴趣讨论等场景。融云的 IMKit 为 Android/iOS/Web 平台的单聊、普通群组业务提供了开箱即用的 UI 组件，其他情况下可以使用 IMLib SDK 构建您的业务体验。

单聊、群组、超级群、聊天室的主要差异如下：

功能	单聊（Private）	普通群组（Group）	超级群（UltraGroup）[1]	聊天室（Chatroom）
场景类比	类似微信私聊	类似微信群组	类似 Discord	聊天室
特性/优势	支持离线消息推送和历史消息记录漫游	支持离线消息推送和历史消息记录漫游，可用于兴趣群、办公群、客服服务沟通等	不限成员数量；支持修改已发消息；提供公有/私有频道、用户组等社群功能	不限成员数量；只有在线用户可接收消息，退出时清除本地历史消息
开通服务	不需要	不需要	需要	不需要
UI 组件	IMKit[2]	IMKit[2]	不提供	不提供
创建方式	无需创建	服务端 API	服务端 API	服务端 API；客户端加入时可自动创建
销毁/解散方式	不适用	服务端 API	服务端 API	服务端 API；具有自动销毁机制[3]
成员数量限制	不适用	群成员数上限 3000	不限	不限
用户加入限制	不适用	不限	最多加入 100 个群，每个群中可加入 50 个频道	默认仅可加入 1 个聊天室，可自行关闭限制[4]
获取加入前的消息	不适用	默认不允许，可关闭限制	默认不允许，可关闭限制	客户端加入聊天室即可获取最新消息，最多 50 条
客户端发送消息频率	每个客户端 5 条/秒[5]	每个客户端 5 条/秒[5]	每个客户端 5 条/秒[5]	每个客户端 5 条/秒[5]
服务端发送消息频率	6000 条/分钟[6]	20 条/秒[6]	100 条/秒[6]	100 条/秒[6]
扩展消息	支持	支持	支持	不支持
修改消息	不支持	不支持	支持	不支持

功能	单聊（Private）	普通群组（Group）	超级群（UltraGroup）	聊天室（Chatroom）
消息可靠度	100% 可靠	100% 可靠	100% 可靠	超出服务端消费上限的消息将被主动抛弃[7]
消息本地存储	移动端、PC 端支持	移动端、PC 端支持	移动端、PC 端支持	不支持
消息云端存储	需开通，可存储 6 - 36 个月[8]	需开通，可存储 6 - 36 个月[8]	默认存储 7 天，提供 3 - 36 个月存储服务[8]	需开通，可存储 2 - 36 个月[8]
离线缓存消息	默认 7 天离线消息缓存	默认 7 天离线消息缓存	不支持	不支持
消息本地搜索	支持	支持	支持	不支持
离线推送通知	支持	支持	支持，可调整推送频率	不支持

脚注：

1. 超级群业务仅限 IM 尊享版使用。
2. IMKit 已支持 Android/iOS/Web 端。
3. 聊天室具有自动销毁机制。默认情况下，如果聊天室在指定时间内（默认 1 个小时）没有人说话，且没有人加入聊天室时，会把聊天室内所有成员踢出聊天室并销毁聊天室。您可以灵活调整聊天室的存活条件与存活时间。
4. 可允许单个用户加入多个聊天室，参考知识库文档：开通单个用户加入多个聊天室。
5. 客户端不区分业务类型整体限制 5 条消息/秒，可付费上调。
6. 此处为服务端 API 默认频率，可付费上调。详细限频信息参见 API 接口列表。
7. 聊天室消息量较大时，超出服务端消费上限的消息将被主动抛弃。您可通过用户白名单、消息白名单、自定义消息级别等服务，改变消息抛弃策略。如果用户在聊天室的用户白名单内，该用户所发送的消息在消息量大时也不会被抛弃。如需了解服务端消费上限与如何改变消息抛弃策略，可参见服务端文档消息优先级服务、聊天室白名单服务。
8. 参考知识库文档：单聊、群聊、聊天室、超级群在融云端历史消息存储时间分别是多长？。

> 前往融云产品文档 · 即时通讯 »

高级与扩展功能

IM 服务支持的高级与扩展功能，包括但不限于以下项目：

- **用户管理**：例如用户封禁、用户黑名单（拉黑）、用户白名单，群组及聊天室禁言、聊天室成员封禁等。
- **在线状态订阅**：将用户每一个终端在线、离线或登出后的状态，同步给应用开发者指定的服务器地址。
- **多设备在线消息同步**：同时支持桌面端、移动端、以及多个 Web 端之间的消息在线同步。
- **全量消息路由**：支持将单聊、群组、聊天室、超级群等的消息数据同步到应用开发者指定的服务器地址。
- **内容审核**：支持设置敏感词列表，过滤或替换消息中的敏感词。利用消息回调服务，可将消息先转发到应用开发者指定的服务器地址，由应用服务器判定是否可发送给目标接收者。
- **推送服务**：融云负责对接厂商推送平台，已覆盖小米、华为、荣耀、OPPO（适用于一加、realme）、vivo、魅族、FCM、APNs 手机系统级推送通道。支持标签推送、多种推送场景、推送统计、全量用户通知等特性。

> 部分功能需要在控制台开通服务后方可使用。部分为收费增值服务，详见即时通讯计费细则。

客户端 SDK

融云即时通讯（IM）客户端 SDK 提供丰富的组件与接口，大部分能力支持开箱即用。配合 IM 服务端 API 接口，可满足丰富的业务特性要求。

在集成融云 SDK 之前，我们建议使用快速上手教程与示例项目进行评估。

如何选择 SDK

IMLib 与 IMKit 是融云 IM 服务提供的两款经典的客户端 SDK。客户端功能在不同平台间基本保持一致。

- **IMLib** 是即时通讯能力库，封装了通信能力和会话、消息等对象。不含任何 UI 界面组件。

  IMLib 已支持绝大部分主流平台及框架，如 Android、iOS、Web、Flutter、React Native、Unity、微信小程序等。

- **IMKit** 是即时通讯界面库，集成了会话界面，并且提供了丰富的自定义功能。

  IMKit 已支持 Android、iOS 与 Web（要求 Web 5.X 版本）。

您可以根据业务需求进行选择：

- 基于 IMLib 开发应用，将融云即时通讯能力嵌入应用中，并自行开发产品的 UI 界面。
- 基于 IMKit 开发应用，将 IMKit 提供的界面组件直接集成到产品中，自定义界面组件功能，节省开发时间。您还可以使用融云提供的独立功能插件扩展 IMKit 的功能。

> 前往融云产品文档 · 客户端 SDK 体系 · IMLib · IMKit »

平台兼容性

IM 客户端 SDK 支持主流移动操作平台，客户端功能在多端基本保持一致，支持多平台互通。以下数据基于 5.X 版本 SDK。

平台/框架	接口语种	支持架构	说明
Android	Java	armeabi-v7a、arm64-v8a、x86、x86-64	系统版本 4.4 及以上
iOS	Objective-C	真机：arm64、armv7。模拟器： arm64（5.4.7+）、x86_64	系统版本 9.0 及以上
Web	Javascript	---	---
Electron	Javascript	详见下方 **Electron 版本与架构支持**	Electron 11.1.x、14.0.0、16.0.x、20.0.x
Flutter	dart	---	Flutter 2.0.0 及以上
React Native	Typescript	-	react-native 0.60 及以上
uni-app	Javascript	---	uni-app 2.8.1 及以上
Unity	C#	armeabi-v7a、arm64-v8a	---

- **Electron 版本与架构支持：**

  Electron 框架需要通过 Web 端 SDK 的 Electron 模块支持（详见 Electron 集成方案），适用于开发运行在 Windows、Linux、MacOS 平台的桌面版即时通讯应用。下表列出了目前已支持的 Electron 版本、桌面操作系统版本及 CPU 架构：

Electron 版本	平台	支持架构	备注
Electron 11.1.x	Windows	ia32 (x86)	`win32-ia32`
Electron 11.1.x	Linux	x64	`linux-x64`
Electron 11.1.x	Linux	arm64	`linux-arm64`
Electron 11.1.x	Mac	x64	`darwin-x64`
Electron 14.0.0	Windows	ia32 (x86)	`win32-ia32`
Electron 14.0.0	Mac	x64	`darwin-x64`
Electron 16.0.x	Windows	ia32 (x86)	`win32-ia32`
Electron 16.0.x	Mac	x64	`darwin-x64`
Electron 20.0.x	Windows	ia32 (x86)	`win32-ia32`

Electron 版本	平台	支持架构	备注
Electron 20.0.x	Mac	x64	`darwin-x64`
Electron 20.0.x	Mac	arm64	`darwin-arm64`

版本支持

IM 客户端 SDK 针对各平台/框架提供的最新版本如下（--- 表示暂未支持）：

SDK/平台	Android	iOS	Web	Electron	Flutter	React Native	Unity	uni-app	小程序
IMLib	5.6.x	5.6.x	5.9.x	5.9.x	5.4.x	5.2.x	5.1.x	5.4.x	5.9.x
IMKit	5.6.x	5.6.x	5.9.x	---	---	---	---	---	---
Global IM UIKit	1.0.x	1.0.x	1.0.x	1.0.x	---	---	---	---	---

集成 SDK 后包体积增量

- 集成 IM SDK 对应用安装包体积大小的增量（Android）
- 集成 IM SDK 对应用安装包体积大小的增量（iOS）

即时通讯服务端

即时通讯服务端提供一套 API 接口与多种语言的开源 SDK。

服务端 API

您可以使用服务端 API 将融云服务集成到您的即时通讯服务体系中，构建您即时通讯 App 的后台服务系统。例如，向融云获取用户身份令牌 (Token)，从 App 产品服务端向用户发送/撤回消息，或管理禁言用户列表。

> 前往融云即时通讯服务端 API 文档 · 集成必读 »

服务端 SDK

融云提供提供多个语言版本的开源服务端 SDK：

- server-sdk-java (GitHub) · (Gitee)
- server-sdk-php (GitHub) · (Gitee)
- server-sdk-go (GitHub) · (Gitee)

控制台

使用控制台，您可以对开发者账户和应用进行管理，开通高级服务，查看应用数据报表，和计费数据。

> 部分 IM 功能必须开通服务后方可使用。详见控制台服务管理页面。

即时通讯数据

如需在融云服务端长期存储单聊会话、群聊会话、聊天室会话的历史消息，您可以开通消息云存储服务。默认的长期存储时长与业务类型相关，可按需调整。该服务存储的数据仅供客户端获取历史消息时使用。

如果需要获取全部用户的消息历史，请开通 Server API 历史消息日志下载。开通后可使用服务端 API 获取最多三天的消息日志。

除此之外，您还可以开通全量消息路由服务，实时将消息同步到您的业务服务器。

您可以前往控制台的数据统计页面，查看即时通讯用户统计、业务统计、消息统计、业务健康检查等数据。开通相应服务后，还能获取如业务数据分析等数据。

融云不会利用客户的数据。同时融云提供完善的数据隐私保护策略。参见 SDK 隐私政策。

快速上手（OC） 快速上手（OC）

更新时间:2024-08-30

本教程是为了让新手快速了解融云即时通讯界面库（IMKit）。在本教程中，您可以体验集成 IMKit SDK 的基本流程和 IMKit 提供的 UI 界面。

前置条件

- 注册开发者账号。注册成功后，控制台会默认自动创建您的首个应用，默认生成开发环境下的 App Key，使用国内数据中心。
- 获取开发环境的应用 App Key。如不使用默认应用，请参考 如何创建应用，并获取对应环境 App Key 和 App Secret。

> 提示
>
> 每个应用具有两个不同的 App Key，分别对应开发环境与生产环境，两个环境之间数据隔离。在您的应用正式上线前，可切换到使用生产环境的 App Key，以便上线前进行测试和最终发布。

环境要求

名称	版本
Xcode	11 +
iOS	9.0 +
CocoaPods	1.10.0 +

SDK 5.1.1 及其以后要求使用 CocoaPods 1.10.0 +，具体请参见知识库文档。

开始集成

IMKit 支持通过 CocoaPods 或手动导入的方式集成。请提前在融云官网 SDK 下载页面或 CocoaPods 仓库查询最新版本。安装 IMKit 将同时集成即时通讯能力库 IMLib。其他插件可按需集成。

导入 SDK

以下介绍如何使用 CocoaPods 导入 IMKit 的 Framework。

1. 在 `podfile` 中添加如下内容：

```
pod 'RongCloudIM/IMKit', '~> x.y.z'
```

> 提示
>
> `x.y.z` 代表具体版本，请在融云官网 SDK 下载页面或 CocoaPods 仓库查询最新版本。

2. 请在终端中运行以下命令：

```
pod install
```

> 提示
>
> 如果出现找不到相关版本的问题，可先执行 pod repo update，再执行 pod install。

3. 上一步完成后，CocoaPods 会在您的工程根目录下生成一个 xcworkspace 文件，只需通过 XCode 打开该文件即可加载工程。

使用 App key 初始化

融云即时通讯客户端 SDK 核心类为 RCIM。在初始化 SDK 时，需要传入生产或开发环境的 App Key。

导入 IMKit SDK 头文件。

```
#import <RongIMKit/RongIMKit.h>
```

如果 SDK 版本 ≧ 5.4.2，请使用以下初始化方法。

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = nil;

[[RCIM sharedRCIM] initWithAppKey:appKey option:initOption];
```

初始化配置（RCInitOption）中封装了区域码（RCAreaCode），导航服务地址（naviServer）、文件服务地址（fileServer）、数据统计服务地址（statisticServer）配置。不作设置表示全部使用默认配置。SDK 默认连接北京数据中心。

如果 App Key 不属于中国（北京）数据中心，则必须传入有效的初始化配置。初始化详细说明参见初始化。

获取用户 Token

用户 Token 是与用户 ID 对应的身份验证令牌，是应用程序的用户在融云的唯一身份标识。应用客户端在使用融云即时通讯功能前必须与融云建立 IM 连接，连接时必须传入 Token。

在实际业务运行过程中，应用客户端需要通过应用的服务端调用 IM Server API 申请取得 Token。详见 Server API 文档 注册用户。

在本教程中，为了快速体验和测试 SDK，我们将使用控制台「北极星」开发者工具箱，从 API 调试页面调用 获取 Token 接口，获取到 **userId** 为 1 的用户的 Token。提交后，可在返回正文中取得 Token 字符串。

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"code":200,"userId":"1","token":"gxld6GHx3t1eDxof1qtxxYrQcjkbhl1V@sgyu.cn.example.com;sgyu.cn.example.com"}
```

建立 IM 连接

1. 监听 IM 连接状态的变化。建议代理对象设置为 AppDelegate 之类的单例对象，保证 APP 整个生命周期都可以监听到代理方法。详见连接状态监听。

```
/// 添加代理委托
[[RCIM sharedRCIM] addConnectionStatusDelegate:self];
```

2. 在自定义登录页面，使用用户 John Joe 的 token 融云建立 IM 连接。

```
[[RCIM sharedRCIM] connectWithToken:@"后台获取的 token"
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
}success:^(NSString *userId) {
//连接成功,可跳转至会话列表页
}error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else {
//无法连接到 IM 服务器，请根据相应的错误码作出对应处理
}
}];
```

SDK 已实现自动重连机制，请参见连接。

展示会话列表

IMKit SDK 已默认提供会话列表页面 RCConversationListViewController 和会话页面 RCConversationViewController。会话列表页面上可查看当前所有的聊天会话，在会话页面可进行消息编辑、查看、发送等活动。

在 IM 连接成功后可跳转至 SDK 会话列表页。您可以直接初始化 SDK 内置会话列表页面 RCConversationListViewController，或通过继承创建一个子类来构建会话列表页面。首次连接时一般没有会话，因此会显示一个空会话列表。客户端接收到消息后，会自动在会话列表页面展示新会话。

推荐应用程序继承 SDK 内置的 RCConversationListViewController，例如 RCDChatListViewController 类：

```
@interface RCDChatListViewController : RCConversationListViewController

@end
```

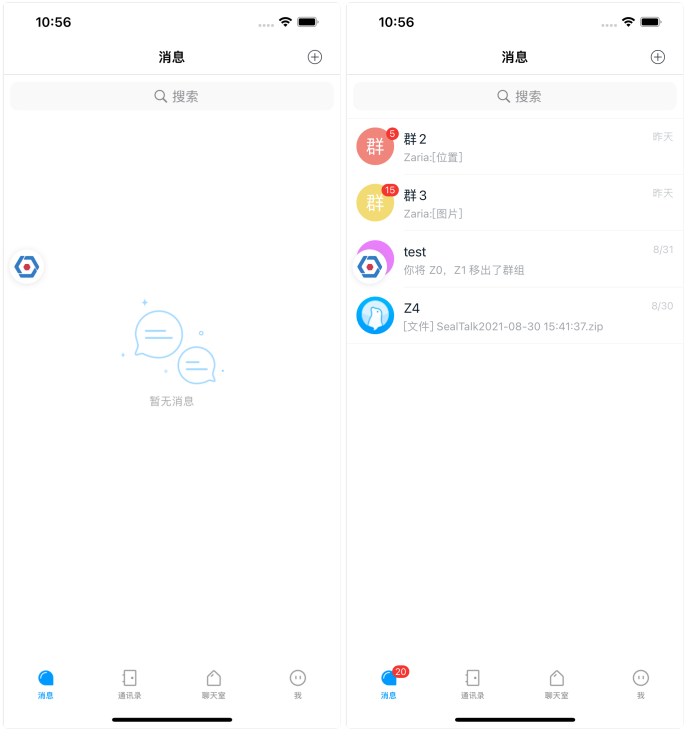
初始化自定义的会话列表页面 RCDChatListViewController，以下代码示例中会话列表进展示单聊会话（ConversationType_PRIVATE）。页面导航器请自行创建。

```
NSArray *displayConversationTypeArray = @[@(ConversationType_PRIVATE)];

RCDChatListViewController *conversationListVC = [[RCDChatListViewController alloc]
initWithDisplayConversationTypes:displayConversationTypeArray
collectionConversationType:nil];

[self.navigationController pushViewController:conversationListVC animated:YES];
```

> 提示
>
> 下图为融云 SealTalk 应用截图。IMKit SDK 默认提供的会话列表页面实现仅包含会话列表，不含搜索、不含底部 Tab Bar 等。如需显示会话列表的昵称和头像，您需要为 IMKit 设置一个用户信息提供者，详见用户概述。



详细使用方法，参见会话列表页面。

展示会话页面

重写会话列表页面点击会话 Cell 的方法，跳转到 SDK 默认内置的会话页面 RCConversationViewController 。

```
- (void)onSelectedTableRow:(RCConversationModelType)conversationModelType
conversationModel:(RCConversationModel *)model
atIndexPath:(NSIndexPath *)indexPath {

RCConversationViewController *conversationVC = [[RCConversationViewController alloc]
initWithConversationType:model.conversationType
targetId:model.targetId];
conversationVC.title = @"想显示的会话标题";
[self.navigationController pushViewController:conversationVC animated:YES];

}
```

详细使用方法，参见会话页面。

测试收发消息

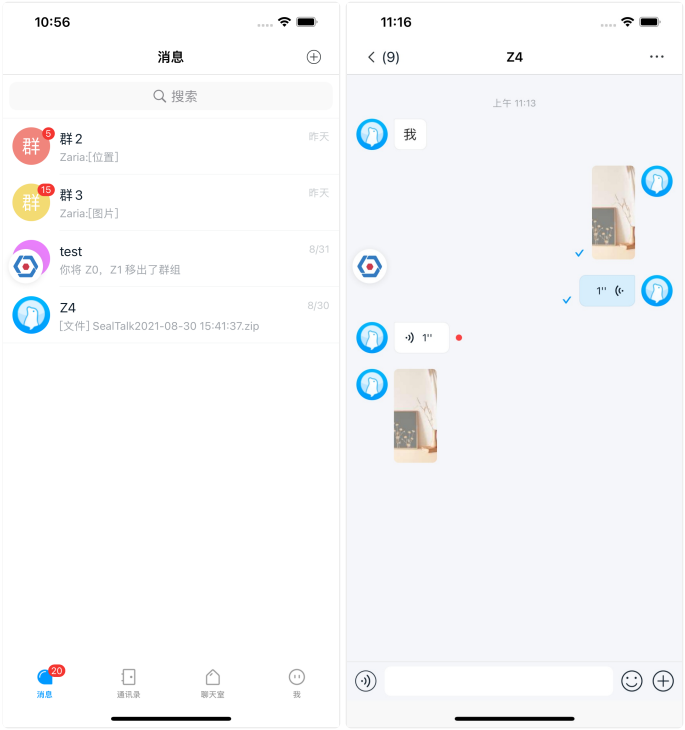
对融云来说，只要提供对方的 userId，融云就可支持跟对方发起聊天。例如，A 需要 发送消息给 B，只需要将 B 的 userId 告知融云服务即可发送消息。

在本教程中，为了快速体验和测试 SDK，我们从控制台「北极星」开发者工具箱 IM Server API 调试 页面向当前登录的用户发送一条文本消息，模拟单聊会话。在实际业务运行过程中，应用客户端可以通过用户 ID、群聊会话 ID、或聊天室 ID 等接收消息。

1. 访问控制台「北极星」开发者工具箱的 IM Server API 调试 页面。
2. 在消息标签下，找到 消息服务 > 发送单聊消息 接口。

   以下模拟了从 UserId 为 2 的用户向 UserId 为 1 的用户发送一条文本消息。

3. 客户端接收到消息后，自动在会话列表页面展示新的单聊会话。点击会话，即可进入消息列表页面，发送消息。



后续步骤

以上步骤即 IMKit SDK 的快速集成与新手体验流程，您体验了基础 IM 通信能力和 UI 界面，更多详细介绍请参考后续各章节详细说明。

- 用户概述：了解 IMKit 与用户相关的服务，以及 IMKit 如何展示头像、昵称、群组成员等
- 群组概述：了解如何使用 IMKit 构建群聊体验
- 会话列表页面：如何构建会话列表页面以及定制化
- 会话页面：如何构建会话页面以及定制化

快速上手（Swift） 快速上手（Swift） 前置条件

更新时间:2024-08-30

- 注册开发者账号。注册成功后，控制台会默认自动创建您的首个应用，默认生成开发环境下的 App Key，使用国内数据中心。
- 获取开发环境的应用 App Key。如不使用默认应用，请参考 如何创建应用，并获取对应环境 App Key 和 App Secret。

> 提示
>
> 每个应用具有两个不同的 App Key，分别对应开发环境与生产环境，两个环境之间数据隔离。在您的应用正式上线前，可切换到使用生产环境的 App Key，以便上线前进行测试和最终发布。

环境要求

名称	版本
Xcode	11 +
iOS	9.0 +
Swift	5.0 +
CocoaPods	1.10.0 +

SDK 5.1.1 及其以后要求使用 CocoaPods 1.10.0 +，具体请参见知识库文档。

开始集成

IMKit 支持通过 CocoaPods 或手动导入的方式集成。请提前在融云官网 SDK 下载页面或 CocoaPods 仓库查询最新版本。安装 IMKit 将同时集成即时通讯能力库 IMLib。其他插件可按需集成。

> 提示
>
> 要求使用 5.4.2 或更高版本。

步骤 1 创建一个项目

打开 Xcode 并创建一个新项目。`RongCloudIM/IMKit` 支持 swift。

步骤 2 安装 IMKit

您可以通过 CocoaPods 来安装 iOS 版本的 IMKit (含 UI SDK)。

1. 在 `podfile` 中添加如下内容：

```
pod 'RongCloudIM/IMKit', '~> x.y.z'
```

> 提示
>
> `x.y.z` 代表具体版本，请在融云官网 SDK 下载页面或 CocoaPods 仓库查询最新版本。

2. 请在终端中运行以下命令：

```
pod install
```

> 提示
>
> 如果出现找不到相关版本的问题，可先执行 pod repo update，再执行 pod install。

3. 上一步完成后，CocoaPods 会在您的工程根目录下生成一个 xcworkspace 文件，只需通过 XCode 打开该文件即可加载工程。

步骤 3 用 App Key 初始化

要在您的应用中集成和运行 Rongcloud IMKit (含 UI SDK)，您需要首先引入 RongIMKit 然后通过 AppDelegate 初始化 RCIM。

```
import RongIMKit

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey:
Any]?) -> Bool {

let appKey: String = "Your_AppKey" // example: bos9p5rlcm2ba
let option: RCInitOption? = nil
RCIM.shared().initWithAppKey(appKey, option: option)

return true
}
```

> 提示
>
> 即使您不使用 AppDelegate，您仍然应该在启动聊天服务之前初始化 RCIM。

步骤 4 获取用户 Token

用户 Token 是与用户 ID 对应的身份验证令牌，是应用程序的用户在融云的唯一身份标识。应用客户端在使用融云即时通讯功能前必须与融云建立 IM 连接，连接时必须传入 Token。

在实际业务运行过程中，应用客户端需要通过应用的服务端调用 IM Server API 申请取得 Token。详见 Server API 文档 注册用户。

在本教程中，为了快速体验和测试 SDK，我们将使用控制台「北极星」开发者工具箱，从 API 调试页面调用 获取 Token 接口，获取到 **userId** 为 1 的用户的 Token。提交后，可在返回正文中取得 Token 字符串。

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"code":200,"userId":"1","token":"gxld6GHx3t1eDxof1qtxxYrQcjkbhl1V@sgyu.cn.example.com;sgyu.cn.example.com"}
```

步骤 5 连接与连接状态监听

1. 监听 IM 连接状态的变化。建议代理对象设置为 AppDelegate 之类的单例对象，保证 APP 整个生命周期都可以监听到代理方法。详见连接状态监听。
2. 在自定义登录页面，使用上一步获取的 `token` 与融云建立 IM 连接，即模拟 `userId` 为 1 的用户连接到融云服务器。
3. 在需要与 `RongcloudIM` 建立连接的页面或者模块中调用 `connectRongcloud` 方法。

```
import UIKit
import RongIMKit

class ViewController: UIViewController {

override func viewDidLoad() {
super.viewDidLoad()
connectRongcloud()
}

func connectRongcloud() {
/*
请在初始化之后，连接之前设置连接监听代理委托。
*/
RCIM.shared().addConnectionStatusDelegate(self)

RCIM.shared().connect(withToken: "后台获取的 token") { dbErrorCode in
// 消息数据库打开，可以进入到主页面

} success: { (userId: String?) in
/* 连接成功,可跳转至会话列表页 */
DispatchQueue.main.async { [weak self] in
/* `openRongcloudConversationList`方法可在**Step 5**中寻找 */
self?.openRongcloudConversationList()
}

} error: { (connectStatus: RCConnectErrorCode) in
if connectStatus == .RC_CONN_TOKEN_INCORRECT {
/* 从 APP 服务获取新 token，并重连 */
} else {
/* 无法连接到 IM 服务器，请根据相应的错误码作出对应处理 */
}
}
}
}

/*
IMKit连接状态的监听器

@param status SDK与融云服务器的连接状态

@discussion 在您设置IMKit连接监听之后，当IMKit与融云服务器的连接状态发生变化时，会回调此方法。
*/
extension ViewController: RCIMConnectionStatusDelegate {
func onRCIMConnectionStatusChanged(_ status: RCConnectionStatus) {

}
}
```

RCConnectErrorCode 定义了连接时可能返回的连接异常状态码。以下的状态码需要 APP 进行处理。

错误码	值	说明	处理方案
RC_CONN_TOKEN_INCORRECT	31004	Token 错误。常见于客户端 SDK 与 App 服务器所使用的 App Key 不一致的错误情况。融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。由测试环境切换到生产环境时，很容易发生应用客户端与应用服务端 App Key 不一致的问题。	检查 SDK 与 App 服务器使用的 App Key 是否一致。
RC_CONN_TOKEN_EXPIRE	31020	Token 已过期。常见于控制台设置了 Token 过期时间。Token 过期之后需要由您的 App 服务器请求融云服务端重新获取 Token，并再次用新的 Token 建立连接	客户端需要请求 App 服务端，向融云获取新的 token。客户端收到新的 Token 后，使用新 Token 发起连接。
RC_CONNECT_TIMEOUT	34006	自动重连超时。常见于无网络的环境，且仅发生在调用连接接口并设置了有效超时时间的情况。超时后 SDK 会放弃重连。开发者需要自行给用户提示，可提示用户等待网络恢复时再重新连接。	重新调用连接手动重连
RC_CONN_APP_BLOCKED_OR_DELETED	31008	Appkey 被封禁	请检查您使用的 AppKey 是否被封禁或已删除
RC_DISCONN_KICK	31010	用户连接成功后被踢下线	退回到登录页面，给用户提示被踢掉线

错误码	值	说明	处理方案
RC_CONN_OTHER_DEVICE_LOGIN	31023	用户连接过程中又在其它设备上登录，导致当前设备被踢	退回到登录页面，给用户提示其他设备登录了当前账号
RC_CONN_USER_BLOCKED	31009	用户被封禁	退回到登录页面，给用户提示被封禁
RC_CLIENT_NOT_INIT	33001	SDK 没有初始化	在使用 SDK 任何功能之前，必须先 Init
RC_INVALID_PARAMETER	33003	开发者接口调用时传入的参数错误	请检查接口调用时传入的参数类型和值
DATABASE_ERROR	33002	数据库错误	检查用户 userId 是否包含特殊字符，SDK userId 支持大小写英文字母、数字的组合方式，最大长度 64 字节

RCConnectionStatus 定义了连接状态监听的可能返回的状态。以下的状态码需要 APP 进行处理。

错误码	值	说明
ConnectionStatus_KICKED_OFFLINE_BY_OTHER_CLIENT	6	当前用户在其他设备上登录，此设备被踢下线
ConnectionStatus_Timeout	14	自动连接超时，SDK 将不会继续连接，用户需要做超时处理，再自行调用 connectWithToken 接口进行连接
ConnectionStatus_TOKEN_INCORRECT	15	Token 无效。 • **Token 错误**：常见于客户端 SDK 与 App 服务器所使用的 App Key 不一致的错误情况。融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。由测试环境切换到生产环境时，很容易发生应用客户端与应用服务端 App Key 不一致的问题。 • **Token 已过期**：常见于控制台设置了 Token 过期时间。Token 过期之后需要由您的 App 服务器请求融云服务端重新获取 Token，并再次用新的 Token 建立连接
ConnectionStatus_DISCONN_EXCEPTION	16	与服务器的连接已断开，用户被封禁

步骤 6 展示会话列表

IMKit SDK 已默认提供会话列表页面 RCConversationListViewController 和会话页面 RCConversationViewController。会话列表页面上可查看当前所有的聊天会话，在会话页面可进行消息编辑、查看、发送等活动。

在 RCIM 连接成功后可跳转至 SDK 会话列表页。您可以直接初始化 SDK 内置会话列表页面 RCConversationListViewController，或通过继承创建一个子类来构建会话列表页面。首次连接时一般没有会话，因此会显示一个空会话列表。客户端接收到消息后，会自动在会话列表页面展示新会话。

以下示例中创建了页面导航器。如果您已经在使用导航控制器，也可以使用 `pushViewController` 功能。

```
private func openRongcloudConversationList() {
let displayConversationTypeArray = [
RCConversationType.ConversationType_PRIVATE.rawValue,
RCConversationType.ConversationType_GROUP.rawValue,
]
let collectionConversationType = [
RCConversationType.ConversationType_SYSTEM.rawValue
]
guard let conversationList = RCConversationListViewController(displayConversationTypes: displayConversationTypeArray,
collectionConversationType: collectionConversationType) else { return }
let naviVC = UINavigationController(rootViewController: conversationList)
naviVC.modalPresentationStyle = .fullScreen
present(naviVC, animated: false)
}
```

> 提示
>
> 推荐继承 IMKit 的 RCConversationListViewController 创建子类会话话列表页面，例如 RCDChatListViewController，方

便应用程序在其中进行一些自定义操作。

```
//
// RCDChatListViewController.swift
// RongcloudApplication
//

import Foundation
import RongIMKit

class RCDChatListViewController: RCConversationListViewController {

}
```

详细使用方法，参见会话列表页面、会话页面。

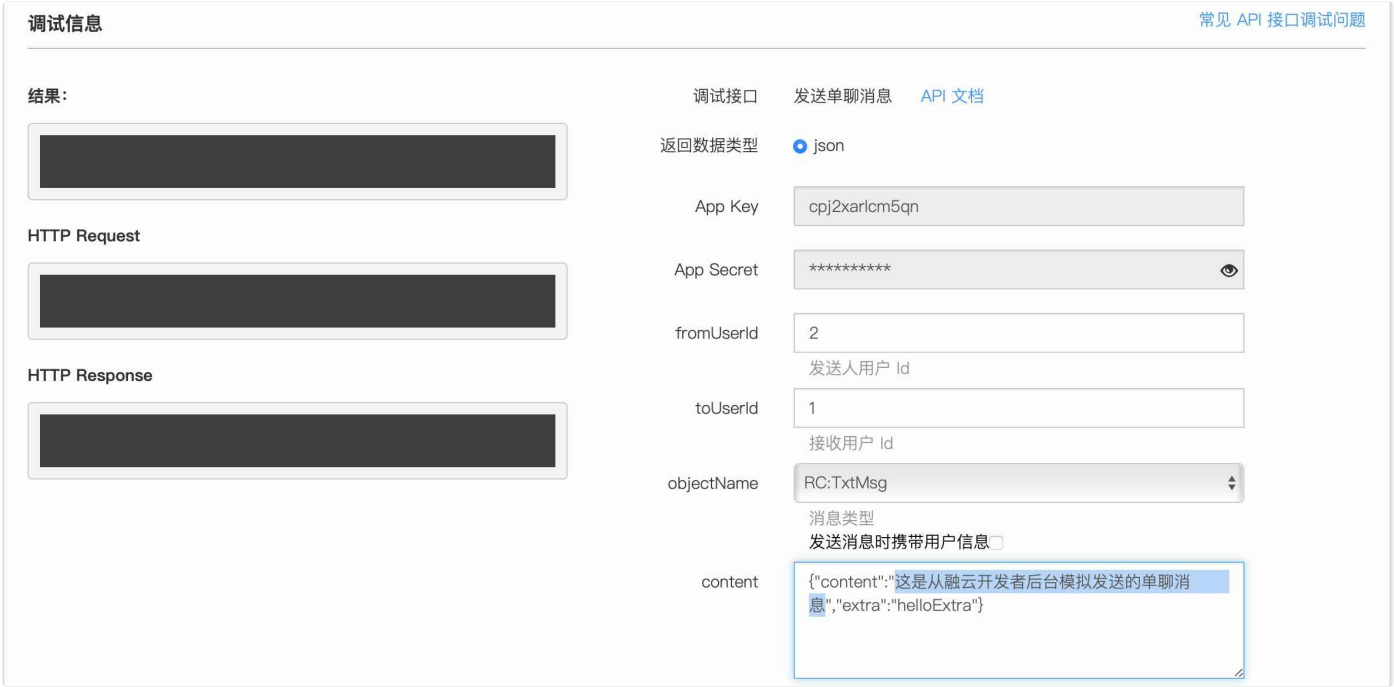
步骤 7 测试收发消息

对融云来说，只要提供对方的 userId，融云就可支持跟对方发起聊天。例如，A 需要 发送消息给 B，只需要将 B 的 userId 告知融云服务即可发送消息。

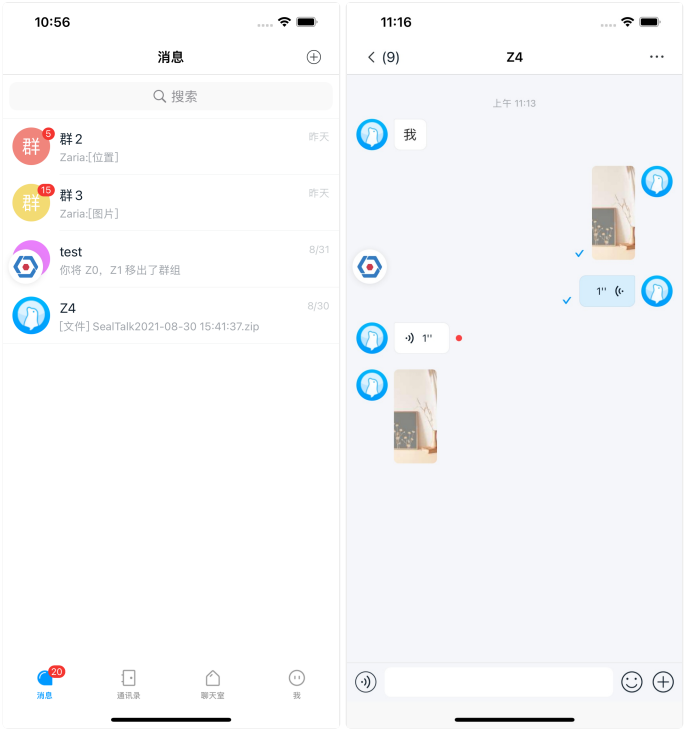
在本教程中，为了快速体验和测试 SDK，我们从控制台「北极星」开发者工具箱 IM Server API 调试 页面向当前登录的用户发送一条文本消息，模拟单聊会话。在实际业务运行过程中，应用客户端可以通过用户 ID、群聊会话 ID、或聊天室 ID 等接收消息。

1. 访问控制台「北极星」开发者工具箱的 IM Server API 调试 页面。
2. 在消息标签下，找到 **消息服务 > 发送单聊消息** 接口。

   以下模拟了从 UserId 为 2 的用户向 UserId 为 1 的用户发送一条文本消息。



3. 客户端接收到消息后，自动在会话列表页面展示新的单聊会话。点击会话，即可进入消息列表页面，发送消息。



> 提示
>
> 上图为融云 SealTalk 应用截图。IMKit SDK 默认提供的会话列表页面实现仅包含会话列表，不含搜索、不含底部 Tab Bar 等。如需显示会话列表的昵称和头像，您需要为 IMKit 设置一个用户信息提供者，详见用户概述。

后续步骤

以上步骤即 IMKit SDK 的快速集成与新手体验流程，您体验了基础 IM 通信能力和 UI 界面，更多详细介绍请参考后续各章节详细说明。

- 用户概述：了解 IMKit 与用户相关的服务，以及 IMKit 如何展示头像、昵称、群组成员等
- 群组概述：了解如何使用 IMKit 构建群聊体验
- 会话列表页面：如何构建会话列表页面以及定制化
- 会话页面：如何构建会话页面以及定制化

导入 SDK 导入 SDK

更新时间:2024-08-30

您可以使用 CocoaPods 导入或手动导入，将 IMKit SDK 集成到您的应用工程中。您可以选择如下任意一种方式：

- 使用 CocoaPods 添加远程依赖项，导入 Framework
- 使用 CocoaPods 添加远程依赖项，导入 IMKit SDK 源码
- 手动导入 Framework（手动集成不支持导入源码）

环境要求

名称	版本
Xcode	11 +
iOS	9.0 +
CocoaPods	1.10.0 +

如需安装 CocoaPods 环境，请参照 安装 CocoaPods 。CocoaPod 版本必须为 1.10.0 或更新版本。具体请查看知识库文档：

https://help.rongcloud.cn/t/topic/747

检查版本

在导入 SDK 前，可以前往融云官网 SDK 下载页面 确认当前最新版本号。

CocoaPods

CocoaPods 用户可以导入 Framework 或导入 IMKit 源码。

提示

请选用同一种方式导入 IMKit 及其插件，不要混用 Framework 和源码。详见 SDK 开源代码说明。

导入 Framework

1. 在 `podfile` 中添加如下内容：

```
pod 'RongCloudIM/IMKit','~> x.y.z' #IMKit
pod 'RongCloudIM/Sight','~> x.y.z' #小视频（可选）
pod 'RongCloudIM/RongSticker','~> x.y.z' #表情（可选）
pod 'RongCloudIM/ContactCard','~> x.y.z' #名片（可选）
pod 'RongCloudIM/LocationKit','~> x.y.z' #位置（可选）
```

提示

- `x.y.z` 代表具体版本，请在融云官网 SDK 下载页面 或 CocoaPods 仓库查询最新版本。
- 5.2.3 及之后的 SDK，不再内置位置插件（LocationKit），您可以根据需要手动集成。5.2.3 之前的 SDK 已内置位置插件，无需手动集成。

2. 请在终端中运行以下命令：

```
pod install
```

> 提示
>
> 如果出现找不到相关版本的问题，可先执行 pod repo update，再执行 pod install。

3. 上一步完成后，CocoaPods 会在您的工程根目录下生成一个 xcworkspace 文件，只需通过 XCode 打开该文件即可加载工程。

4. 引用库参考

```
#import <RongIMKit/RongIMKit.h>
#import <RongSight/RongSight.h>
#import <RongSticker/RongSticker.h>
#import <RongContactCard/RongContactCard.h>
#import <RongLocationKit/RongLocationKit.h>
```

导入源码

IMKit SDK 5.0 及以上版本 UI 相关库开放了源代码，支持在 CocoaPods 中以源码形式进行调试与集成。

1. 在 podfile 中添加如下内容：

```
pod 'RongCloudOpenSource/IMKit','x.y.z' #IMKit，含会话列表页面，会话页面，输入工具栏
pod 'RongCloudOpenSource/Sight','x.y.z' #小视频（可选）
pod 'RongCloudOpenSource/RongSticker','x.y.z' #表情（可选）
pod 'RongCloudOpenSource/IFly','x.y.z' #语音输入（可选）
pod 'RongCloudOpenSource/ContactCard','x.y.z' #名片（可选）
pod 'RongCloudOpenSource/LocationKit','x.y.z' #位置（可选）
```

> 提示
>
> - `x.y.z` 代表具体版本，各个 SDK 的最新版本号可能不同，在融云官网 SDK 下载页面或 CocoaPods 仓库能查询到。
> - `IFly`（语音输入）插件仅支持以源码方式导入。如需使用该插件，请务必以源码方式集成 IMKit SDK。
> - IMKit 5.1.8 之前，名片插件（ContactCard）仅支持以源码方式导入，且要求必须同样以源码方式集成 IMKit SDK。
> - IMKit 5.1.8 及之后，名片插件（ContactCard）支持以源码方式或以 Framework 方式导入。请选用同一种方式导入 IMKit 及其插件，不要混用 Framework 和源码。

2. 请在终端中运行以下命令：

```
pod install
```

> 提示
>
> 如果出现找不到相关版本的问题，可先执行 pod repo update，再执行 pod install。

3. 上一步完成后，CocoaPods 会在您的工程根目录下生成一个 xcworkspace 文件，只需通过 XCode 打开该文件即可加载工程。

4. 引用库参考：

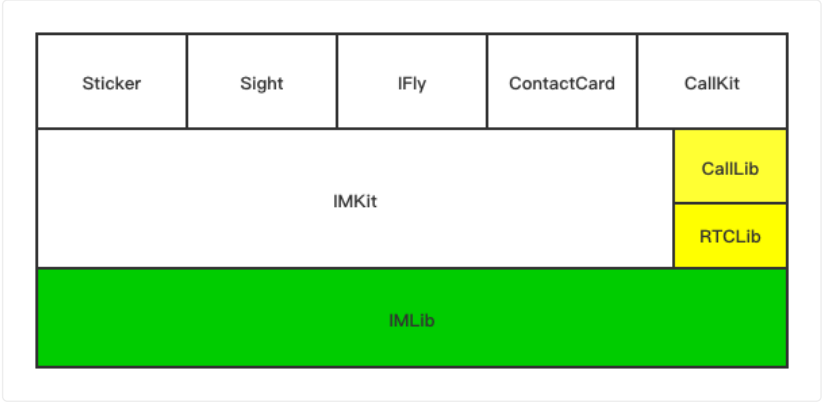
```
#import <RongCloudOpenSource/RongIMKit.h>
#import <RongCloudOpenSource/RongSight.h>
#import <RongCloudOpenSource/RongSticker.h>
#import <RongCloudOpenSource/RongiFlyKit.h>
#import <RongCloudOpenSource/RongContactCard.h>
#import <RongCloudOpenSource/RongLocationKit.h>
```

> 提示
>
> 对于开发者自定义 UI 的需求，有以下建议：
>
> 1. 不建议直接修改源码内容，防止后续源码升级将修改内容覆盖。
> 2. 建议通过继承重写某些类与自身逻辑不一致的方法，来增加新方法以扩展自身的业务逻辑。
> 3. 建议使用 SDK 对外暴露的接口，如果调用私有接口可能会出现版本升级引起私有接口变更。

SDK 开源代码说明

- IMLib（绿色）以 framework 形式存在，属于 RongCloudIM。
- RTCLib 和 CallLib（黄色）以 framework 形式存在，属于 RongCloudRTC。
- IMKit 及其插件与 CallKit（白色）可以源码形式存在，只属于 RongCloudOpenSource。

> 图中白色部分的 SDK 或插件可能有对应 Framework。但在您的应用工程中，白色部分（IMKit、Sticker、IFly、ContacCard、CallKit）必须同为源码或同为 Framework，否则可能会发生冲突。



手动导入（仅支持 Framework）

手动集成仅支持以 Framework 方式进行集成。不支持以源码方式集成。

1. 前往融云官网 SDK 下载页面，将 IMKit SDK 下载到本地。

2. 将下载下来的 SDK 导入您的项目中，所需的 SDK 如下：

	Framework	资源文件
IMLib	• RongIMLib.xcframework • RongIMLibCore.xcframework • RongChatRoom.xcframework • RongCustomerService.xcframework • RongDiscussion.xcframework • RongLocation.xcframework 5.2.5 及之后版本如不使用位置插件 LocationKit，无需导入 • RongPublicService.xcframework	• RCConfig.plist
IMKit	• RongIMKit.xcframework	• Emoji.plist • RCColor.plist • RongCloud.bundle • zh-Hans.lproj • ar.lproj • en.lproj
LocationKit（可选）	• RongLocationKit.xcframework	
Translation（可选）	• RongTranslation.xcframework	
Sticker（可选）	• RongSticker.xcframework	• RongSticker.bundle • en.lproj • zh-Hans.lproj • ar.lproj
Sight（可选）	• RongSight.xcframework	

> 提示
>
> 仅 5.2.2 及之后版本的 SDK 支持翻译插件（Translation），该插件暂仅适用于使用新加坡数据中心的应用。

3. 修改您的项目配置。在 `General -> Frameworks,Libraries,and Embedded Binaries` 中，将 IMKit SDK 所需的 Framework 全部改为 `Embed & Sign`。

外部链接

- IMKit 开源代码仓库（GitHub）
- IMKit 开源代码仓库（Gitee）

初始化 初始化

更新时间:2024-08-30

在使用 SDK 其它功能前，必须先进行初始化。本文中将详细说明 IMKit SDK 初始化的方法。

首次使用融云的用户，我们建议先阅读 IMKit SDK 快速上手，以完成开发者账号注册等工作。

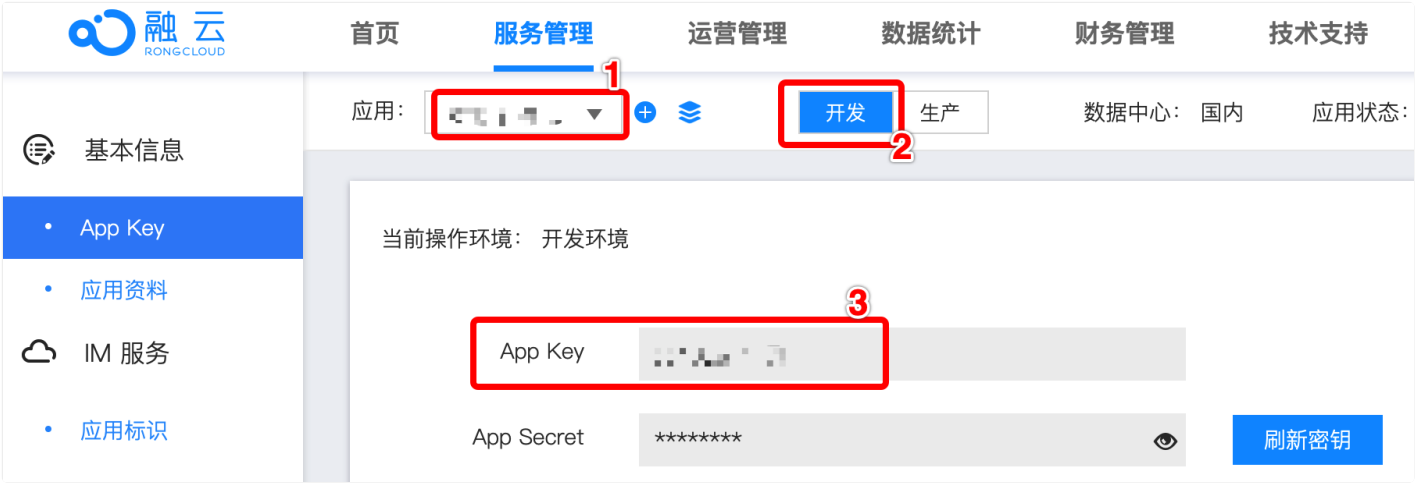
准备 App Key

您必须拥有正确的 App Key，才能进行初始化。

您可以控制台，查看您已创建的各个应用的 App Key。

如果您拥有多个应用，请注意选择应用名称（下图中标号 1）。另外，融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。在获取应用的 App Key 时，请注意区分环境（生产 / 开发，下图中标号 2）。

> 提示
>
> - 如果您并非应用创建者，我们建议在获取 App Key 时确认页面上显示的数据中心是否符合预期。
> - 如果您尚未向融云申请应用上线，仅可使用开发环境。



海外数据中心

- 如果您使用海外数据中心，且使用 5.4.2 及更新版本的开发版（dev）SDK，请注意在初始化配置中传入正确的区域码（RCAreaCode）。
- 如果您使用海外数据中心，且使用稳定版（stable） SDK，或使用早于 5.4.2 版本的开发版（dev）SDK，必须在初始化之前修改 IMLib SDK 连接的服务地址为海外数据中心地址。否则 SDK 默认连接中国国内数据中心服务地址。详细说明请参见配置海外数据中心服务地址。

初始化 SDK

> 提示
>
> 以下初始化方法要求 SDK 版本 ≧ 5.4.2。您可前往 融云官网 SDK 下载页面 查询最新开发版（Dev）和稳定版（Stable）的版本号。

导入 SDK 头文件

```
#import <RongIMKit/RongIMKit.h>
```

请在初始化方法中传入生产或开发环境的 App Key。

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = nil;

[[RCIM sharedRCIM] initWithAppKey:appKey option:initOption];
```

初始化配置（RCInitOption）中封装了区域码（RCAreaCode）配置。SDK 将通过区域码获取有效的导航服务地址、文件服务地址、数据统计服务地址、和日志服务地址等配置。

- 如果 App Key 属于中国（北京）数据中心，您无需配置额外配置 `RCInitOption`，SDK 将全部使用默认配置。
- 请务必在控制台核验当前 App Key 所属海外数据中心后，找到 RCAreaCode 中对应的枚举值进行配置。

例如，如果使用新加坡数据中心，配置如下：

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = [[RCInitOption alloc] init];
initOption.areaCode = RCAreaCodeSG;

[[RCIM sharedRCIM] initWithAppKey:appKey option:initOption];
```

除区域码外，初始化配置（InitOption）中还封装了以下配置：

- 导航服务地址（`naviServer`）：一般情况下不建议单独配置。SDK 内部默认使用与区域码对应的地址。
- 文件服务地址（`fileServer`）：仅限私有云使用。
- 数据统计服务地址（`statisticServer`）：一般情况下不建议单独配置。SDK 内部默认使用与区域码对应的地址。

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = [[RCInitOption alloc] init];
initOption.areaCode = RCAreaCodeSG;
initOption.naviServer = @"http(s)://naviServer";
initOption.fileServer = @"http(s)://fileServer";
initOption.statisticServer = @"http(s)://statsServer";
[[RCIM sharedRCIM] initWithAppKey:appKey option:initOption];
```

初始化 SDK（旧版）

> 提示
>
> 如果您使用的开发版 SDK 版本号小于 5.4.2，或者稳定版 SDK 版本号小于等于 5.3.8，只能使用以下方式初始化。建议您及时升级 SDK 到最新版本。

导入 SDK 头文件。

```
#import <RongIMKit/RongIMKit.h>
```

在初始化方法中传入应用的开发环境或生产环境 AppKey，请勿混淆。

```
[[RCIM sharedRCIM] initWithAppKey:@"融云 AppKey"];
```

配置指南 配置指南

更新时间:2024-08-30

IMKit 全局配置旨在提供易于使用的功能配置，帮助您快速构建聊天应用程序。

配置说明

IMKit 在 RCKitConfig.h 文件中定义了全局配置，并按照以下模块进行划分。您可以在相应的文件中了解 IMKit 提供的所有全局配置。

类别	描述	API 文档	源码
消息配置	控制消息的自动重发、前台提示音、合并转发、输入状态、已读回执、消息撤回等行为。	RCKitMessageConf	RCKitMessageConf.h
UI 配置	控制暗黑模式、布局方向、会话列表、会话页面的头像和消息、标题大小等。	RCKitUIConf	RCKitUIConf.h
字体配置	控制字体大小 。	RCKitFontConf	RCKitFontConf.h

修改 IMKit 配置

对于 iOS，可使用 RCKitConfig 单例（或者使用 RCKitConfigCenter 宏）修改 IMKit 功能配置。每个应用程序仅有一个 IMKit 全局配置。

配置示例：

```
//消息可撤回的最大时长（秒）
RCKitConfigCenter.message.maxRecallDuration = 120;

//设为选择媒体资源时包含视频文件
RCKitConfigCenter.message.isMediaSelectorContainVideo = YES;

//头像显示默认为矩形，可修改为圆角显示。
RCKitConfigCenter.ui.globalMessageAvatarStyle = RC_USER_AVATAR_CYCLE;

// 二级标题，默认 fontSize 为 17（文本消息，引用消息内容，会话列表 title）
RCKitConfigCenter.font.secondLevel = 20;

// 修改文件消息中文件类型（扩展名）对应显示的图标（Since 5.3.4）。
[RCKitConfigCenter.ui registerFileSuffixTypes:types];
```

检查 IMKit 配置

IMKit 配置是实时应用的，修改后的配置将在下一次 UI 刷新或者操作时生效。建议在初始化 IMKit 后完成所有配置。

监听连接状态　　监听连接状态

更新时间:2024-08-30

SDK 提供了 IM 连接状态变化委托协议 RCIMConnectionStatusDelegate。通过监听 IM 连接状态的变化，App 可以进行不同业务处理，或在页面上给出提示。

连接状态变化委托协议

RCIMConnectionStatusDelegate 协议定义如下：

```
@protocol RCIMConnectionStatusDelegate <NSObject>

/*!
IMKit连接状态的监听器

@param status SDK与融云服务器的连接状态

@discussion 如果您设置了IMKit 连接监听之后，当SDK与融云服务器的连接状态发生变化时，会回调此方法。
*/
- (void)onRCIMConnectionStatusChanged:(RCConnectionStatus)status;

@end
```

> 提示
>
> 代理对象请设置给 AppDelegate 之类的单例对象，保证 APP 整个生命周期都可以监听到代理方法。

当连接状态发生变化时，SDK 会通过 - (void)onRCIMConnectionStatusChanged:(RCConnectionStatus)status; 方法，将当前的连接状态回调给开发者。RCConnectionStatus 中定义了连接过程中的可能的状态变化，下表具体说明了需要 App 处理的的状态码。

错误码	值	说明
ConnectionStatus_KICKED_OFFLINE_BY_OTHER_CLIENT	6	当前用户在其他设备上登录，此设备被踢下线
ConnectionStatus_Timeout	14	自动连接超时，SDK 将不会继续连接，用户需要做超时处理，再自行调用 connectWithToken 接口进行连接
ConnectionStatus_TOKEN_INCORRECT	15	Token 无效。 • **Token 错误**：常见于客户端 SDK 与 App 服务器所使用的 App Key 不一致的错误情况。融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。由测试环境切换到生产环境时，很容易发生应用客户端与应用服务端 App Key 不一致的问题。 • **Token 已过期**：常见于控制台设置了 Token 过期时间。Token 过期之后需要由您的 App 服务器请求融云服务端重新获取 Token，并再次用新的 Token 建立连接
ConnectionStatus_DISCONN_EXCEPTION	16	与服务器的连接已断开，用户被封禁

添加或移除代理委托

支持添加多个代理。请在初始化之后，连接之前设置连接监听代理委托。

为了避免内存泄露，请在不需要监听时，将设置的代理移除。

```objc
/// 添加代理委托
[[RCIM sharedRCIM] addConnectionStatusDelegate:self];


/// 移除代理委托
[[RCIM sharedRCIM] removeConnectionStatusDelegate:self];
```

连接 连接

更新时间:2024-08-30

应用客户端成功连接到融云服务器后，才能使用融云即时通讯 SDK 的收发消息功能。

融云服务端在收到客户端发起的连接请求后，会根据连接请求里携带的用户身份验证令牌（Token 参数），判断是否允许用户连接。

前置条件

- 通过服务端 API 注册用户（获取 Token）。融云客户端 SDK 不提供获取 Token 方法。应用程序可以调用自身服务端，从融云服务端获取 Token。
  - 取得 Token 后，客户端可以按需保存 Token，供后续连接时使用。具体保存位置取决于应用程序客户端设计。如果 Token 未失效，就不必再向融云请求 Token。
  - Token 有效期可在控制台进行配置，默认为永久有效。即使重新生成了一个新 Token，未过期的旧 Token 仍然有效。Token 失效后，需要重新获取 Token。如有需要，可以主动调用服务端 API 作废 Token。
- 建议应用程序在连接之前设置连接状态监听。
- SDK 已完成初始化。

> 提示
>
> **请不要在客户端直接调用服务端 API 获取 Token。获取 Token 需要提供应用的 App Key 和 App Secret。客户端如果保存这些凭证，一旦被反编译，会导致应用的 App Key 和 App Secret 泄露。所以，请务必确保在应用服务端获取 Token。**

连接聊天服务器

请根据应用的业务需求与设计，自行决定合适的时机（登陆、注册、或其他时机以免无法进入应用主页），向融云聊天服务器发起连接请求。

请注意以下几点：

- 必须在 SDK 初始化之后，调用 `connect` 方法进行连接。否则可能没有回调。
- SDK 内部实现了重连机制，在网络异常导致 IM 连接断开时，会自动重连，直到应用主动断开连接（见断开连接）。

接口（可设置超时）

客户端用户首次连接聊天服务器时，建议调用带连接超时时间（timeLimit）的接口，并设置超时秒数。在网络较差等导致连接超时的情况下，你可以利用接口的超时错误回调，并在 UI 上提醒用户，例如建议客户端用户等待网络正常的时候再次连接。

一旦连接成功，SDK 的重连机制将立即开始生效，并接管所有的重连处理。当因为网络原因断线时，SDK 会不停重连直到连接成功为止，不需要您做额外的连接操作。应用主动断开连接，将退出重连机制，详见断开连接。其他情况请参见重连机制与重连互踢。

如果免密登录，建议将 timeLimit 参数设置为 0，可以在 dbOpenedBlock 回调中进行页面跳转，优先展示本地历史数据。连接逻辑则完全托管给 SDK。

调用示例

```
[[RCIM sharedRCIM] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值" timeLimit:5
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
} success:^(NSString *userId) {
//连接成功
} error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//Token 错误，可检查客户端 SDK 初始化与 App 服务端获取 Token 时所使用的 App Key 是否一致
} else if(status == RC_CONNECT_TIMEOUT) {
//连接超时，弹出提示，可以引导用户等待网络正常的时候再次点击进行连接
} else {
//无法连接 IM 服务器，请根据相应的错误码作出对应处理
}
}];
```

参数	类型	必填	说明
token	NSString	是	首次连接时必须由 App 服务端调用融云服务端 API 获取。参见服务端 API 文档 注册用户 。
timeLimit	int	是	SDK 连接超时时间，单位：秒。SDK 最多连接 `timeLimit` 秒，超时则返回 `RC_CONNECT_TIMEOUT` 错误，并不再重连。如果需要不需要设置超时时间，可将 `timeLimit` 设置为小于 0 的数字。`timeLimit` < 0 时，SDK 会一直连接，直到连接成功或者出现 SDK 无法处理的错误（如 token 非法）。
dbOpenedBlock	Block	-	本地消息数据库打开的回调。`code` 中返回 RCDBErrorCode 。0 表示打开成功。
successBlock	Block	-	连接建立成功的回调。`userId` 中返回当前连接成功的用户 ID。
errorBlock	Block	-	连接建立失败的回调，该回调表示 SDK 遇到了无法自动重连的错误，例如 Token 无效。`status` 中返回连接状态码 RCConnectErrorCode 。

接口（无超时设置）

如果客户端用户已成功登录过你的应用，且连接过融云聊天服务器，后续离线登录时建议使用此接口。

此时因用户已有历史数据，并不需要强依赖于连接成功。可以在 dbOpenedBlock 回调中进行页面跳转，优先展示本地历史数据。连接逻辑完全托管给 SDK 即可。

> 提示
>
> 当网络较差时，有可能长时间不会回调。

调用此接口后，SDK 的重连机制将立即开始生效，并接管所有的重连处理。SDK 会不停重连直到连接成功为止，不需要您做额外的连接操作。应用主动断开连接，将退出重连机制，详见断开连接。其他情况请参见重连机制与重连互踢。

```
[[RCIM sharedRCIM] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值"
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
}success:^(NSString *userId) {
//连接成功
}error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else {
//无法连接到 IM 服务器，请根据相应的错误码作出对应处理
}
}];
```

参数	类型	必填	说明
token	NSString	是	首次连接时必须由 App 服务端调用融云服务端 API 获取。参见服务端 API 文档 注册用户 。
dbOpenedBlock	Block	-	本地消息数据库打开的回调。`code` 中返回 RCDBErrorCode 。0 表示打开成功。
successBlock	Block	-	连接建立成功的回调。`userId` 中返回当前连接成功的用户 ID。

参数	类型	必填	说明
errorBlock	Block	-	连接建立失败的回调，该回调表示 SDK 遇到了无法自动重连的错误，例如 Token 无效。`status` 中返回连接状态码 RCConnectErrorCode 。

连接状态码

具体可以参考下表 `RCConnectErrorCode` 具体错误码的说明和处理方案建议

> 提示
>
> 关于 `RC_CONN_TOKEN_INCORRECT` 的说明：
>
> 1. 在 `RC_CONN_TOKEN_INCORRECT` 的情况下，您需要请求您的服务器重新获取 Token 并建立连接，但是注意避免无限循环，以免影响 App 用户体验。
> 2. 此方法的回调线程并非为原调用线程，需要进行 UI 操作时请注意切换到主线程。

错误码	值	说明	处理方案
RC_CONN_TOKEN_INCORRECT	31004	Token 错误。常见于客户端 SDK 与 App 服务器所使用的 App Key 不一致的错误情况。融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。由测试环境切换到生产环境时，很容易发生应用客户端与应用服务端 App Key 不一致的问题。	检查 SDK 与 App 服务器使用的 App Key 是否一致。
RC_CONN_TOKEN_EXPIRE	31020	Token 已过期。常见于控制台设置了 Token 过期时间。Token 过期之后需要由您的 App 服务器请求融云服务端重新获取 Token，并再次用新的 Token 建立连接	客户端需要请求 App 服务端，向融云获取新的 token。客户端收到新的 Token 后，使用新 Token 发起连接。
RC_CONNECT_TIMEOUT	34006	自动重连超时。常见于无网络的环境，且仅发生在调用连接接口并设置了有效超时时间的情况。超时后 SDK 会放弃重连。开发者需要自行给用户提示，可提示用户等待网络恢复时再重新连接。	重新调用连接手动重连
RC_CONN_APP_BLOCKED_OR_DELETED	31008	Appkey 被封禁	请检查您使用的 AppKey 是否被封禁或已删除
RC_DISCONN_KICK	31010	用户连接成功后被踢下线	退回到登录页面，给用户提示被踢掉线
RC_CONN_OTHER_DEVICE_LOGIN	31023	用户连接过程中又在其它设备上登录，导致当前设备被踢	退回到登录页面，给用户提示其他设备登录了当前账号
RC_CONN_USER_BLOCKED	31009	用户被封禁	退回到登录页面，给用户提示被封禁
RC_CLIENT_NOT_INIT	33001	SDK 没有初始化	在使用 SDK 任何功能之前，必须先 Init
RC_INVALID_PARAMETER	33003	开发者接口调用时传入的参数错误	请检查接口调用时传入的参数类型和值
DATABASE_ERROR	33002	数据库错误	检查用户 userId 是否包含特殊字符，SDK userId 支持大小写英文字母、数字的组合方式，最大长度 64 字节

断开连接　　断开连接

更新时间:2024-08-30

在 App 需要执行切换用户登录、注销登录的操作时，需要断开与融云的 IM 连接。SDK 支持设置断开 IM 连接之后是否允许向用户发送消息推送通知。

> 提示
>
> SDK 在前后台切换或者网络出现异常都会自动重连，会保证连接的可靠性。除非 App 逻辑需要登出，否则不需要调用此方法进行手动断开。

断开连接（允许推送）

主动断开与融云服务端的 IM 连接，并设置断开连接后允许融云服务端进行远程推送。以下两种方式效果一致：

```
[[RCIM sharedRCIM] disconnect];
```

或者：

```
[[RCIM sharedRCIM] disconnect:YES];
```

参数	类型	说明
isReceivePush	BOOL	断开连接后，是否允许融云服务端进行远程推送。 YES 表示接收远程推送。NO 表示不接收远程推送。

如果融云服务端发现 App 客户端不在线（默认要求全部设备已下线），在接收新消息时，融云服务端会为该用户记录一条离线消息，并触发融云服务端的推送服务。融云服务端会通过推送通道下发一条提醒到客户端 SDK。该提醒一般以通知形式展示在通知面板，提示用户有离线消息。

断开连接（不允许推送）

主动断开与融云服务端的 IM 连接，并设置断开连接后不允许融云服务端进行远程推送。

需要注销登录（登出）或切换 App 用户账号时，推荐使用以下任一方式，效果一致。

```
[[RCIM sharedRCIM] logout];
```

或者：

```
[[RCIM sharedRCIM] disconnect:NO];
```

参数	类型	说明
isReceivePush	BOOL	断开连接后，是否允许融云服务端进行远程推送。 YES 表示接收远程推送。NO 表示不接收远程推送。

断开连接且不允许推送的情况下，融云服务端仅记录离线消息，但不会为当前设备触发推送服务。如果用户登录了多个设备，则在其他设备中最后一个登录的设备上可正常接收推送。在多设备场景下，App 可能需要保证设备间消息记录一致，可通过开启多设备消息同步实现。详见多设备消息同步。

重连机制与重连互踢　　重连机制与重连互踢 自动重连机制

更新时间:2024-08-30

SDK 内已实现自动重连机制，一旦连接成功，SDK 的重连机制将立即开始生效，并接管所有的重连处理。当因为网络原因断线时，SDK 内部会尝试重新建立连接，不需要您做额外的连接操作。

可能导致 SDK 断线重连的异常情况如下：

- **弱网环境**：可能出现 SDK 不停重连的情况。因为客户端 SDK 和融云服务端之间存在连接保活机制，一旦因如果网络太差导致心跳超时，SDK 就会触发重连操作，尝试重连直到连接成功。
- **无网环境**：SDK 的重连机制会暂停。一旦网络恢复，SDK 会进行重连操作。

> 提示
>
> 一旦触发连接错误的回调，SDK 将退出重连机制。请根据具体的状态码自行处理。

重连时间间隔

SDK 尝试重连时，时间间隔逐次变大，分别是 0.05s（5.6.2 之前为 0s）, 0.25s, 0.5s, 1s, 2s, 4s, 8s, 16s, 32s。之后每 64s 重试一次。

当 APP 切换到前台或者网络状态发生变化，重连时间会按照上面的时间间隔从头开始，保证这种情况下能尽快的连接成功。

主动退出重连机制

应用主动断开连接后，SDK 将退出重连机制，不再尝试重连。

重连互踢策略

重连互踢策略用于控制 SDK 自动重连成功时是否需要下线的设备。

即时通讯业务默认仅允许同一用户账号在单台移动端设备上登录。后登录的移动端设备一旦连接成功，则自动踢出之前登录的设备。在部分情况下，SDK 的重连机制可能会导致后登录设备无法正常在线。

例如，默认的重连互踢策略可能导致以下情况：

1. 用户张三尝试在移动设备 A 上登录，但因 A 设备网络不稳定导致未连接成功，触发了 SDK 的自动重连机制。
2. 用户此时尝试换到移动设备 B 上登录。B 设备连接成功，且用户可通过 B 设备正常使用即时通讯业务。
3. A 设备网络稳定之后，SDK 重连成功。因此时 A 设备为后上线设备，导致 B 设备被踢出。

修改 App 用户的重连互踢策略

如果 App 用户希望在以上场景中将重连成功的 A 设备下线，同时保持 B 设备登录，可以通过修改自己的重连互踢策略实现。

> 提示
>
> IMKit 不直接提供对应 API。如有需要，请调用 IMLib SDK 的 `setReconnectKickEnable` 方法。详见 IMLib 文档 重连机制与重连互踢。

多端同时在线 多端同时在线

更新时间:2024-08-30

多端同时在线是指同一用户账号从多个平台同时连接到融云即时通讯服务的功能。默认情况下，融云即支持多端设备同时在线。该功能无需开通即可使用。

> 默认多端设备之间不会进行消息同步。如有需要，请开通多设备消息同步服务。

多端登录限制说明

默认的情况下，同一用户账号可在移动端、Web 端、桌面端、小程序端最多一个设备上同时在线。

平台类别	限制	IM SDK 支持平台列表
移动端	默认仅支持一个移动端设备连接。如需支持移动端多设备登录，请提交工单申请开通移动多端服务。	Android、iOS、Flutter、React Native、uni-app、Unity
桌面端	默认仅支持一个桌面端设备连接。	Electron 框架（通过 Web 端 SDK 的 Electron 模块支持）
Web 端	默认仅支持一个 Web 页面连接（每个浏览器标签页认为是一个连接）。在控制台自助开通多设备消息同步服务后，自动支持多 Web 页面连接。	Web
小程序端	默认仅支持一个小程序连接。如需支持小程序多设备登录，请提交工单申请开通小程序多端服务。	小程序

多设备消息同步

更新时间:2024-08-30

在即时通讯业务中，同一用户账号可能在多个设备上登录。**多设备消息同步**是融云服务端提供的一项服务，可用于同一用户账号的多个设备之间同步收发消息。

默认情况下，融云不会在设备之间同步消息。新消息被某一端设备收取后，其他端无法收取该消息。

适用场景

在融云即时通讯业务中，多设备消息同步适用于以下情况：

- 同一用户账号在多设备上同时在线（无论是否为同一端），希望同步收发消息。例如，用户可能拥有多个移动端设备，如两个 Android 设备、一个 iOS 设备。

  > 注意：融云默认已支持多端同时在线，同一用户账号可在移动端、Web 端、桌面端、小程序端最多一个设备上同时在线。但是如果需要允许 App 用户同时在多个移动端设备或多个小程序端上在线，需要分别提交工单申请，详见多端同时在线。

- 同一用户账号换设备登录（无论是否曾在该设备登录过），希望同步收发的消息记录。例如用户从 Android 设备下线后，换到另一个设备从 Web 端登录。
- 同一用户账号在当前设备卸载重装 App，希望同步收发消息记录。

支持在多设备间同步的消息

并非所有消息均支持多设备消息同步。**状态消息**仅支持在多设备同时在线时同步接收，不在线的设备无法通过**多设备消息同步**收到消息。

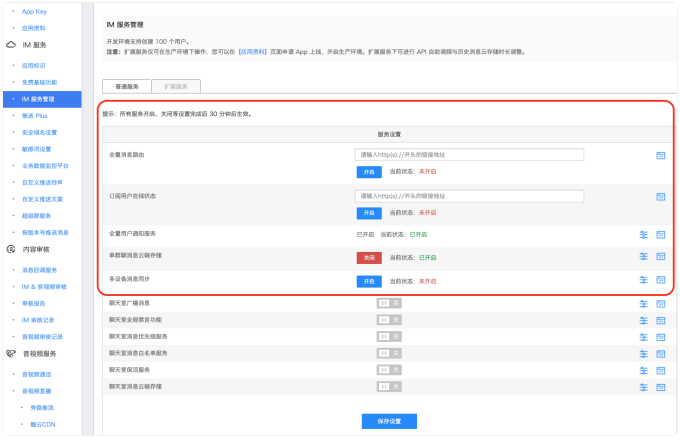
以下情况均属于状态消息：

- 融云内置消息类型中定义为状态消息类型的消息。内置状态类型消息的具体包括：正在输入状态消息（`RC:TypSts`）
- 自定义的状态消息类型的消息。详见各个客户端「自定义消息」文档。
- 使用服务端 API 状态消息接口发送的所有消息（不区分消息类型）均不支持同步。具体的 API 接口为发送单聊状态消息（`/statusmessage/private/publish.json`）、发送群聊状态消息（`/message/group/publish.json`）。

开通服务

请前往控制台，在 IM 服务管理 页面的**普通服务**标签下开通**多设备消息同步**服务。该服务在开发环境免费使用，默认为关闭状态。生产环境预存费用后才可开通服务。

> 服务开启、关闭设置完成后 30 分钟内生效。



对其他功能或业务的影响

多设备消息同步服务的状态对即时通讯业务中的离线补偿[?]、撤回消息、聊天室业务等有影响。

对离线补偿的影响

控制台的多设备消息同步服务包含了融云服务端离线补偿机制[?]的开关。

开通多设备消息同步服务后，融云服务端自动为 App 启用离线补偿机制。离线补偿机制会在以下场景触发：

- **换设备登录**。用户在新设备上登录后（无论是否曾在该设备登录过），SDK 可获取最近指定天数（默认离线补偿天数为 1 个自然日）在其他终端上发送和接收过的消息。
- **应用卸载重装**。消息与会话列表是存储在本地数据库，用户卸载 App 时会删除本地数据库。用户重新安装 App 后并再次连接时，会触发融云服务端离线补偿机制，SDK 可获取最近指定天数（默认离线补偿天数为 1 个自然日）在其他终端上发送和接收过的消息。

> **注意：在 换设备登录 或 应用卸载重装 场景下，离线补偿机制仅可获取到最近（默认离线补偿天数为 1 天，最大 7 天）的会话。早于该天数的会话无法通过离线补偿机制获取。因此，离线补偿后的会话列表可能与原设备上或卸载前的会话列表并不一致（您可能会有丢失部分会话的错觉）。**

如需修改离线消息补偿的天数，可提交工单。建议谨慎设置离线补偿天数，当单用户消息量超大时，可能会因为补偿消息量过大，造成端上处理压力较大。

对 Web 平台连接数的影响

开通多设备消息同步服务后，可额外支持多 Web 页面连接（每个浏览器标签页也认为是一个连接），最多 10 个。

对撤回消息的影响

- 未开通多设备消息同步服务时，多端之间无法同步撤回的消息。
- 开通多设备消息同步服务后，消息发送端一旦撤回消息，如果用户在其他端在线，则其他端同步撤回该条已发送消息。如果用户在其他端不在线时，则在用户登录后同步撤回已发送的消息。

对服务端 API 发送消息的影响

通过服务端 API 发送消息时，部分接口可通过 `isIncludeSender` 指定消息发送者可否在客户端接收该已发消息。

- 未开通多设备消息同步服务时，仅在发送者已登录客户端（在线）的情况下，通过 Server API 发送的消息可即时同步到发送者的在线客户端，无法同步到离线的客户端。
- 开通多设备消息同步服务后，如果发送者未登录客户端（离线），通过 Server API 发送的消息可在再次上线时同步到发送者的在线客户端。

用户概述　　用户概述

更新时间:2024-08-30

App 用户需要接入融云服务，才能使用即时通讯服务。对于融云来说，用户是指持有由融云分发的有效 Token，接入并使用即时通讯服务的 App 用户。

注册用户

应用服务端（App Server）应向融云服务端提供 App 用户的用户 ID（`userId`），以向融云换取唯一用户 Token。对融云来说，这个以 `userId` 获取 Token 的步骤即注册用户，且必须通过调用 Server API 来完成。

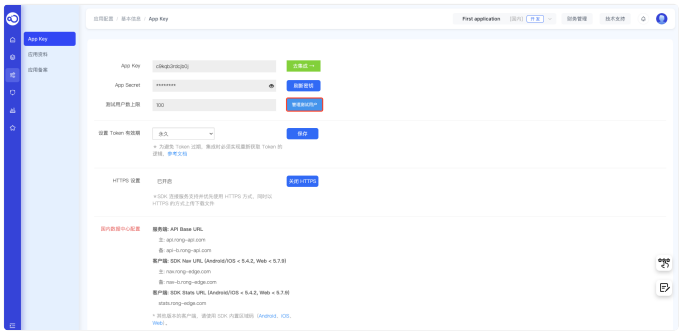
应用客户端必须持有有效 Token，才能成功连接到融云服务端，使用融云即时通讯服务。当 App 客户端用户向服务器发送登录请求时，服务器会查询数据库以检查连接请求是否匹配。

注册用户数限制

- 开发环境[?]中的注册用户数上限为 100 个。
- 在 生产环境[?]中，升级为 **IM 旗舰版**或 **IM 尊享版**后不限制注册用户数。

删除用户

删除用户是指在应用的开发环境中，通过控制台删除已注册的测试用户，以控制开发环境中的测试用户总数。生产环境不支持该操作。



注销用户

注销用户是指在融云服务中删除用户数据。App 可使用该能力实现自身的用户销户功能，满足 App 上架或合规要求。

融云返回注销成功结果后，与用户 ID 相关数据即被删除。您可以向融云查询所有已注销用户的 ID。如有需要，您可以重新激活已被注销的用户 ID（注意，用户个人数据无法被恢复）。

仅 IM Server API 提供上述能力。

用户信息

用户信息泛指用户的昵称、头像，以及群组的群昵称、群头像等数据。融云服务端不提供用户信息托管维护服务。

在 IMKit SDK UI 中，如果需要在会话页面、好友列表等处显示用户及群组的头像、昵称等信息，需要由应用层提供相关数据。为方便 App 开发者，IMKit SDK 设计并提供了多个信息提供者接口，用于 SDK 向应用层获取用户信息。

好友关系

App 用户之间的好友关系需要由应用服务器（App Server）自行维护。融云不会同步或保存 App 端的好友关系数据。

如果需要对客户端用户之间的消息收发行为进行限制（例如，App 的所有 `userId` 泄漏，导致某个恶意用户可越过好友关系向任意用户发送消息），可以考虑使用用户白名单服务。用户一旦开启并设置白名单，则仅可接收来自该白名单中用户的消息。

用户管理接口

功能分类	功能描述	客户端 API	服务端 API
注册用户	使用 App 用户的用户 ID 向融云换取 Token。	不提供该 API	注册用户
删除用户	参见上文删除用户。	不提供该 API	不提供该 API
废弃 Token	废弃在特定时间点之前获取的 Token。	不提供该 API	作废 Token
注销用户	注销用户是指在融云服务中停用用户 ID，并删除用户个人数据。	不提供该 API	注销用户
查询已注销用户	获取已注销的用户 ID 列表。	不提供该 API	查询已注销用户
重新激活用户 ID	在融云服务中重新启用已注销用户的 ID。	不提供该 API	重新激活用户 ID
设置客户端本地的用户信息	设置用户信息提供者，由应用层负责提供数据。	设置用户信息提供者	不提供该 API
设置融云服务端的用户信息	设置在融云推送服务中使用的用户名称与头像。	不提供该 API	未提供单独的设置接口。在注册用户时必须提供用户信息。
获取客户端本地的用户信息	获取在会话页面、好友列表等处显示的用户头像、昵称等信息。	获取用户信息	不提供该 API
获取融云服务端的用户信息	获取用户在融云注册的信息，包括用户创建时间和服务端的推送服务使用的用户名称、头像 URL。	不提供该 API	获取信息
修改客户端本地的用户信息	修改在客户端本地数据库中保存的用户昵称、头像等信息。	刷新用户信息	不提供该 API
修改融云服务端的用户信息	修改在融云推送服务中使用的用户名称与头像。	不提供该 API	修改信息
封禁用户	禁止用户连接到融云即时通讯服务，并立即断开连接。可按时长解封或主动解封。查询被封禁用户的用户 ID、封禁结束时间。	不提供该 API	添加封禁用户、解除封禁用户、查询封禁用户
查询用户在线状态	查询某用户的在线状态。	不提供该 API	查询在线状态
用户黑名单	在用户的黑名单列表中添加、移除用户。在 A 用户黑名单的用户无法向 A 发送消息。查询黑名单相关信息。 IMKit 未提供该 API 接口。此处客户端 API 为 IMLib 的 API 接口。	加入黑名单、移出黑名单、查询用户是否在黑名单中、获取黑名单列表	加入黑名单、移出黑名单、查询黑名单
用户白名单	用户一旦开启并设置白名单，则仅可接收来自该白名单中用户的消息。	不提供该 API	开启用户白名单、用户白名单状态查询、添加白名单、移出白名单、查询白名单

群组概述 群组概述

更新时间:2024-08-30

群聊是即时通讯类应用中常见的多人通讯方式，一般包含两个及以上的用户。融云的群组业务支持丰富的群组成员管理、禁言管理等特性，支持离线消息推送和历史消息记录漫游，可用于兴趣群、办公群、客服服务沟通等。IMKit 提供开箱即用的群聊会话 UI 组件。

服务配置

客户端 SDK 默认支持群组业务，不需要申请开通。部分基础功能与增值服务可以在控制台的免费基础功能和 IM 服务管理页面进行开通和配置。

- App Key 下可创建的群组数量无限制。单个用户可加入的群组数量无限制。
- 群组有容量上限，默认群组成员数量上限为 3000 人，可提交工单修改。
- 默认情况下，App Key 未开通单群聊消息云端存储服务。您可以自助开通，详见 开通单群聊消息云存储服务。如果是生产环境的 App Key，仅 **IM 旗舰版**、**IM 尊享版**可开通该服务。
- 默认情况下，用户只能查看他们加入群组后的群聊消息。开启服务后，新入群用户可以获取他们加入群组之前的群聊历史消息。详见开通新用户获取加入群组前历史消息服务。

客户端 SDK 使用须知

- 客户端 SDK（IMKit/IMLib）均不提供群组管理的 API。如需创建群组，必须由 App 服务器请求融云服务端 API 实现。其他操作例如解散群组、加入群组、退出群组等群组管理操作，均须由 App 服务器请求融云服务端 API 实现。详见下方群组管理功能。
- 群主、群管理员、群公告、邀请入群、群号搜索等均为群组业务逻辑，需在 App 侧自行实现。
- 融云只负责将消息传达给群组中的所有用户，不维护群组成员的资料（头像、名称、群成员名片等）。App 需要自行在业务服务器上维护相关数据，并实现 IMKit 的相关接口，向 IMKit 提供数据。参见用户信息。

群组管理功能

对于客户端开发人员来说，创建群组、解散等基础管理操作只需要与 App 自身的业务服务端交互即可，由 App 服务端负责调用相应的融云服务端 API（Server API）接口完成相关操作。

服务端 API	功能描述
创建群组、解散群组	提供创建者用户 ID、群组 ID、和群名称，向融云服务端申请建群。如解散群组，则群成员关系不复存在。
加入群组、退出群组	加入群组后，默认可查看入群以后产生的新消息。退出群组后，不再接收该群的新消息。
刷新群组信息	修改在融云推送服务中使用的群组信息。
查询群组成员	查询指定群组所有成员的用户 ID 信息。
查询用户所在群组	根据用户 ID 查询该用户加入的所有群组，返回群组 ID 及群组名称。融云不存储群组资料信息，群组资料及群成员信息需要开发者在应用服务器自行维护，如应用服务端维护的用户群组关系有缺失时，可通过此接口来核对校验。
同步用户所在群组	向融云服务端同步指定用户当前所加入的所有群组，防止应用中的用户群组信息与融云服务端的用户所属群信息不一致。如果在集成融云服务前 App Server 上已有群组及成员数据，第一次连接融云服务器时，可使用此接口向融云同步已有的用户与群组对应关系。
禁言指定群成员	在指定的单个群组中或全部群组中，禁言一个或多个用户。被禁言用户可以接收查看群组中其他用户消息，但不能通过客户端 SDK 发送消息。
设置群组全体禁言	将群组全体成员禁言。被禁言群组的所有成员均不能发送消息，需要某些用户可以发言时，可将此用户加入到群禁言用户白名单中。
加入群组全体禁言白名单	群组被整体禁言后，禁言白名单中用户可以发送群消息。

用户信息 用户信息

更新时间:2024-08-30

本文主要描述如何设置用户、群组、群成员的昵称、头像等信息，具体涵盖以下内容：

- 设置当前登录的用户的昵称与头像。
- 通过多种信息提供者协议，从 App 层获取用户、群组、群成员、群成员列表等信息，提供给 SDK，用于设置与显示。
- 如何持久化存储用户信息、群组信息、群成员信息。

启用持久化存储

IMKit SDK 支持将用户信息、群组信息、群成员信息持久化存储到本地数据库中。不支持持久化存储群成员列表信息。

当需要将用户信息、群组信息、群成员信息在本地持久化保存的时候，可以设置 RCIM 类的 enablePersistentUserInfoCache 属性为 YES，信息将被同时缓存到内存和数据库，App 下次启动时缓存仍然有效。

> 提示
>
> 请在初始化之后，连接之前设置持久化存储属性。

```
@property (nonatomic, assign) BOOL enablePersistentUserInfoCache;
```

默认为 NO。信息只缓存到内存，在 App 生命周期结束时会被清除，App 下次启动时将再次通过对应的信息提供者协议获取信息。

了解信息提供者协议

IMKit SDK 设计并提供了以下信息提供者协议，用于向应用层获取信息。开发者需要实现该协议，提供正确的数据。

应用层仅需负责提供数据源，SDK 获取数据后会自动设置、刷新用户头像与昵称，以及相关 UI 展示。

- **用户信息提供者**：RCIMUserInfoDataSource，用于获取非本端登录用户的用户信息，例如昵称、头像等。
- **群组信息提供者**：RCIMGroupInfoDataSource，用于获取群组的昵称、头像。
- **群成员信息提供者**：RCIMGroupUserInfoDataSource，用于获取群成员在群组内的昵称（群名片）。
- **群成员列表信息提供者**：RCIMGroupMemberDataSource，用于获取群成员列表。

> 提示
>
> 建议在 AppDelegate 或者其他单例中遵循以上协议。

在 App 的生命周期中，如果 SDK 获取过用户或群组的信息，便会在内存中缓存该信息。只要 SDK 可从缓存中获取到所需信息，便不会再触发获取该信息的回调。

在处理用户、群组信息时，SDK 的默认行为如下：

1. 当 SDK 需要在 UI 上显示用户或者群组信息时，首先从内存中查询已获取的数据。
2. 如果 SDK 可从缓存或本地数据库中查询到所需信息，将直接将数据返回 UI 层并刷新 UI；该过程不会触发从应用层获取信息的回调方法。
3. 如果 SDK 未能从缓存或本地数据库查询到所需信息，则将触发相应信息提供者接口的回调方法，并尝试从应用层获取信息。收到应用层提供的相应信息后，SDK 将刷新 UI。

设置当前登录的用户信息

开发者与融云服务器建立连接之后，应该设置当前登录用户的用户信息，已确保 SDK 可正常在界面上显示本端用户头像。

如果传入的用户信息中的用户 ID 与当前登录的用户 ID 不匹配，SDK 会将其忽略。

```
RCUserInfo *_currentUserInfo = [[RCUserInfo alloc] initWithUserId:userId name:userNickName portrait:userPortraitUri];

[RCIM sharedRCIM].currentUserInfo = _currentUserInfo;
```

> 提示
>
> 请在成功与融云服务端建立连接之后设置，您可以在连接成功的回调中进行设置。

设置其他用户的用户信息

用户信息提供者（RCIMUserInfoDataSource 协议）用于获取非本端登录用户的用户信息，例如昵称、头像等。

代理回调

SDK 通过代理向应用层获取用户信息。

设置代理委托

请在 SDK 初始化之后，连接融云服务器之前设置代理委托。

```
[RCIM sharedRCIM].userInfoDataSource = self;
```

代理方法

```
@protocol RCIMUserInfoDataSource <NSObject>

/*!
SDK 的回调，用于向 App 获取用户信息

@param userId 用户ID
@param completion 获取用户信息完成之后需要执行的Block [userInfo:该用户ID对应的用户信息]

@discussion SDK通过此方法获取用户信息并显示，请在completion中返回该用户ID对应的用户信息。
在您设置了用户信息提供者之后，SDK在需要显示用户信息的时候，会调用此方法，向您请求用户信息用于显示。
*/
- (void)getUserInfoWithUserId:(NSString *)userId completion:(void (^)(RCUserInfo *userInfo))completion;

@end
```

代码示例

以下代码示例来自 sealtalk-ios 项目（GitHub · Gitee）。

```
- (void)getUserInfoWithUserId:(NSString *)userId completion:(void (^)(RCUserInfo *userInfo))completion{
//开发者调自己的服务器接口，根据 userID 异步请求用户信息
[RCDUserInfoManager getUserInfoFromServer:userId
complete:^(RCUserInfo *userInfo) {
//将请求回来的用户信息通过 completion 回调给 SDK
return completion(userInfo);
}];
}
```

获取用户信息

可以调用以下方法获取用户信息。SDK 将首先尝试从本地缓存获取数据。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会尝试从本地数据库中获取用户信息。

如果本地没有相关信息的数据，则会触发相应的回调方法，从应用层获取数据。

调用示例

```
[[RCIM sharedRCIM] getUserInfoCache:@"userId"];
```

刷新用户信息

您可以通过下面的方法更新 SDK 在本地缓存的用户信息。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会更新本地数据库中的用户信息。

调用示例

```
RCUserInfo *user = [[RCUserInfo alloc] initWithUserId:@"userId" name:@"userName" portrait:@"http://portrait"];

[[RCIM sharedRCIM] refreshUserInfoCache:user withUserId:user.userId];
```

清除用户信息

您可以通过下面的方法清除 SDK 在本地缓存的用户信息。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会清除本地数据库中的用户信息。

调用示例

```
[[RCIM sharedRCIM] clearUserInfoCache];
```

设置群组信息

群组信息提供者（RCIMGroupInfoDataSource 协议）用于获取群组的昵称、头像。

代理回调

SDK 通过代理向应用层获取群组信息。

设置代理委托

```
[RCIM sharedRCIM].groupInfoDataSource = self;
```

代理方法

```
@protocol RCIMGroupInfoDataSource <NSObject>

/*!
SDK 的回调，用于向 App 获取群组信息

@param groupId 群组ID
@param completion 获取群组信息完成之后需要执行的Block [groupInfo:该群组ID对应的群组信息]

@discussion SDK通过此方法获取群组信息并显示，请在completion的block中返回该群组ID对应的群组信息。
在您设置了群组信息提供者之后，SDK在需要显示群组信息的时候，会调用此方法，向您请求群组信息用于显示。
*/
- (void)getGroupInfoWithGroupId:(NSString *)groupId completion:(void (^)(RCGroup *groupInfo))completion;

@end
```

代码示例

以下代码示例来自 sealtalk-ios 项目（GitHub · Gitee）。

```
- (void)getGroupInfoWithGroupId:(NSString *)groupId completion:(void (^)(RCGroup *))completion {
//开发者调自己的服务器接口根据userID异步请求数据
[RCDGroupManager getGroupInfoFromServer:groupId
complete:^(RCGroup * groupInfo) {
return completion(groupInfo);
}];
}
```

获取群组信息

可以调用以下方法获取群组信息。SDK 将首先尝试从本地缓存获取数据。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会尝试从本地数据库中获取群组信息。

如果本地没有相关信息的数据，则会触发相应的回调方法，从应用层获取数据。

调用示例

```
[[RCIM sharedRCIM] getGroupInfoCache:@"groupId"];
```

刷新群组信息

构造群组信息（RCGroup），方法如下：

```
RCGroup *group = [[RCGroup alloc] initWithGroupId:@"groupId" groupName:@"groupName" portraitUri:@"http://portraitUri"];
/// 5.4.1 及之后，新增 NSString 类型的 extra 字段
group.extra = @"groupExtra";
```

您可以通过下面的方法更新 SDK 在本地缓存的群组信息。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会刷新本地数据库中的群组信息。

```
[[RCIM sharedRCIM] refreshGroupInfoCache:group withGroupId:group.groupId];
```

清除群组信息

您可以通过下面的方法清除 SDK 在本地缓存的群组信息。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会清除本地数据库中的群组信息。

调用示例

```
[[RCIM sharedRCIM] clearGroupInfoCache];
```

群成员信息

群成员信息提供者（RCIMGroupUserInfoDataSource协议）用于获取群组成员的昵称、头像。如果未遵循该协议，SDK 默认使用用户信息协议。

代理回调

SDK 通过代理向应用层获取群成员信息（昵称、头像）。

设置代理委托

请在 SDK 初始化之后，连接融云服务器之前设置代理委托。

```
[RCIM sharedRCIM].groupUserInfoDataSource = self;
```

代理方法

```
@protocol RCIMGroupUserInfoDataSource <NSObject>

/*!
SDK 的回调，用于向 App 获取用户在群组中的群名片信息

@param userId 用户ID
@param groupId 群组ID
@param completion 获取群名片信息完成之后需要执行的Block [userInfo:该用户ID在群组中对应的群名片信息]

@discussion 如果您使用了群名片功能，SDK需要通过您实现的群名片信息提供者，获取用户在群组中的名片信息并显示。
*/
- (void)getUserInfoWithUserId:(NSString *)userId
inGroup:(NSString *)groupId
completion:(void (^)(RCUserInfo *userInfo))completion;

@end
```

代码示例

以下代码示例来自 sealtalk-ios 项目（GitHub · Gitee）。

```
- (void)getUserInfoWithUserId:(NSString *)userId
inGroup:(NSString *)groupId
completion:(void (^)(RCUserInfo *userInfo))completion {
[RCDGroupManager getGroupMemberDetailInfoFromServer:userId
groupId:groupId
complete:^(RCUserInfo *memberInfo) {
return completion(memberInfo);
}];
}
```

获取群成员信息

可以调用以下方法获取群成员信息（昵称、头像）。SDK 将首先尝试从本地缓存获取数据。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会尝试从本地数据库中获取群成员信息。

如果本地没有相关信息的数据，则会触发相应的回调方法，从应用层获取数据。

调用示例

```
[[RCIM sharedRCIM] getGroupUserInfoCache:@"userId" withGroupId:@"groupId"];
```

刷新群成员信息

您可以通过下面的方法更新 SDK 在本地缓存的群成员信息（昵称、头像）。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会更新本地数据库中的群成员信息。

调用示例

```
RCUserInfo *user = [[RCUserInfo alloc] initWithUserId:@"userId" name:@"userName" portrait:@"http://portrait"];

[[RCIM sharedRCIM] refreshGroupUserInfoCache:user withUserId:user.userId withGroupId:@"groupId"];
```

清除群成员信息

您可以通过下面的方法清除 SDK 在本地缓存的群成员信息（昵称、头像）。如已允许 SDK 进行持久化存储（即已在 RCIM.h 里开启 enablePersistentUserInfoCache），SDK 还会清除本地数据库中的群成员信息。

调用示例

```
[[RCIM sharedRCIM] clearGroupUserInfoCache];
```

设置群成员列表信息

在群组会话页面中使用 @ 等功能时，SDK 需要应用层提供群成员列表。

> 注意，SDK 不会缓存群成员列表信息。

代理回调

设置代理委托

请在 SDK 初始化之后，连接融云服务器之前设置代理委托。

```
[RCIM sharedRCIM].groupMemberDataSource = self;
```

代理方法

```
@protocol RCIMGroupMemberDataSource <NSObject>
@optional

/*!
SDK 的回调，用于向 App 获取当前群组成员列表（需要实现用户信息提供者 RCIMUserInfoDataSource）

@param groupId 群ID
@param resultBlock 获取成功之后需要执行的Block [userIdList:群成员ID列表]

@discussion SDK通过此方法群组中的成员列表，请在resultBlock中返回该群组ID对应的群成员ID列表。
在您设置了群组成员列表提供者之后，SDK在需要获取群组成员列表的时候，会调用此方法，向您请求群组成员用于显示。
*/
- (void)getAllMembersOfGroup:(NSString *)groupId result:(void (^)(NSArray<NSString *> *userIdList))resultBlock;
@end
```

代码示例

以下代码示例来自 sealtalk-ios 项目（GitHub · Gitee）。

```
- (void)getAllMembersOfGroup:(NSString *)groupId result:(void (^)(NSArray<NSString *> *))resultBlock {
[RCDGroupManager getGroupMembersFromServer:groupId
complete:^(NSArray<NSString *> *_Nonnull memberIdList) {
return resultBlock(memberIdList);
}];
}
```

会话列表页面 会话列表页面

更新时间:2024-08-30

会话列表页面展示了当前用户设备上的所有会话。一旦 SDK 的本地消息数据库生成消息，SDK 就会生成会话列表，并按照时间倒序排列，置顶会话会排在最前。IMKit 提供基于 UIKit `UITableView` 的会话页面类 RCConversationListViewController 。

会话列表页面

会话列表页面一般由标题栏和会话列表两部分组成。

> 提示
>
> 如需显示会话列表的昵称和头像，您需要为 IMKit 设置一个用户信息提供者。IMKit 通过用户信息提供者获取需要显示的资料数据。详情请参见用户信息。

初始化

> 提示
>
> 基于 IMKit 开发时，推荐继承使用 RCConversationListViewController 类，创建自定义的会话列表页面。在排查或复现与会话页面相关的问题时，可以直接使用 RCConversationListViewController 类。

调用 RCConversationListViewController 类的初始化方法构建会话列表页面，设置会话列表中需要包含的会话的类型。注意，您需要将 `RCConversationType` 转为 NSNumber构建 Array。

```
NSArray *displayConversationTypeArray = @[@(ConversationType_PRIVATE), @(ConversationType_GROUP),
@(ConversationType_SYSTEM)];
NSArray *collectionConversationTypeArray = nil;

RCConversationListViewController *conversationListVC = [[RCDChatListViewController alloc]
initWithDisplayConversationTypes:displayConversationTypeArray collectionConversationType:collectionConversationTypeArray];

[self.navigationController pushViewController:conversationListVC animated:YES];
```

参数	类型	说明
displayConversationTypeArray	NSArray (NSNumber *)	列表中需要显示的会话类型数组。需要将 RCConversationType 转为 NSNumber构建 Array。
collectionConversationTypeArray	NSArray (NSNumber *)	列表中需要聚合为一条显示的会话类型数组。您需要将 RCConversationType 转为 NSNumber构建 Array。详见按类型聚合会话。

用法

标题栏

IMKit 的 RCConversationListViewController 使用了系统的导航栏，可用于显示会话列表的标题，自行通过 UIViewController 的 title 属性设置标题。详情请参见用户信息。

会话列表

会话列表组件按时间倒序显示所有会话的列表，置顶会话会排在最前。IMKit 默认已实现了左滑会话 Cell 删除会话功能。

会话列表的每个项目的视图是在 RCConversationCell 中创建和自定义的。您还可以创建自定义会话 Cell。

> 提示
>
> 如果开启了多设备消息同步，在 换新设备登录 或 应用卸载重装 场景下，离线补偿机制仅可获取到最近（默认离线补偿天数为 1 天，最大 7 天）的单聊、群聊会话消息。早于该天数的会话无法通过离线补偿机制获取。因此，离线补偿后的会话列表可能与原设备上或卸载前的会话列表并不一致（您可能会有丢失部分会话的错觉）。

定制化

IMKit 会话列表界面的样式可供自定义修改。

修改会话头像形状

会话列表中的每个会话项目上均会显示一个头像图标，即会话头像（不是聊天页面内中消息列表中的头像）。单聊会话显示对方用户头像，群聊会话显示群组头像，聚合会话显示默认头像或应用程序主动设置的头像。IMKit 支持通过 IMKit 全局配置单独控制会话列表中的会话头像样式。

会话头像显示默认为矩形，可修改为圆形。头像显示大小默认值为 46*46。请在 App 初始化之后调用以下代码进行设置：

```
RCKitConfigCenter.ui.globalConversationAvatarStyle = RC_USER_AVATAR_CYCLE;
```

RCUserAvatarStyle 说明:

类型名	说明
RC_USER_AVATAR_RECTANGLE	矩形
RC_USER_AVATAR_CYCLE	圆形

会话头像高度必须大于或者等于 36，修改方法如下：

```
RCKitConfigCenter.ui.globalConversationPortraitSize = CGSizeMake(46, 46);
```

是否展示网络连接状态提示

当连接断开或者重连时，SDK 会在会话列表页面顶部展示连接状态提示栏。该功能默认开启。可通过 RCConversationListViewController 的属性关闭该提示。

```
@property (nonatomic, assign) BOOL isShowNetworkIndicatorView;
```

如需修改提示文字，可修改语言包中的 `en.lproj` 和 `zh-Hans.lproj`。

自定义空视图

RCConversationListViewController 中提供了会话列表空视图 `emptyConversationView`（即无会话需要显示）。如果您需要自定义空视图 View，赋值给 `emptyConversationView` 即可。

```
- (void)viewDidLoad {
[super viewDidLoad];

//视图的 frame 需要开发者根据需求设定
UIImageView *empty = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 100, 100)];
empty.center = self.view.center;
empty.backgroundColor = [UIColor redColor];
self.emptyConversationView = empty;
}
```

自定义会话 Cell

您可以参考融云的开源项目 **SealTalk**（GitHub · Gitee）中会话列表 `RCDChatListViewController` 类中相关方法实现和自定义 Cell（`RCDChatListCell`）。SealTalk 示例项目具体是针对系统会话（`ConversationType_SYSTEM`）的好友请求消息（`RCContactNotificationMessage`）制作了自定义 Cell。

1. 重写数据源方法。

   在该方法内筛选数据源 `dataSource` 中具体的会话类型及消息的 `model`。`model` 类型必须修改为 `model.conversationModelType = RC_CONVERSATION_MODEL_TYPE_CUSTOMIZATION`。

   ```
   -(NSMutableArray *)willReloadTableData:(NSMutableArray *)dataSource;
   ```

2. 重写自定义 Cell。

   ```
   // 自定义 cell
   -(RCConversationBaseCell *)rcConversationListTableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
   *)indexPath;
   ```

3. 重写自定义 Cell 高度的方法。

   ```
   // 高度
   -(CGFloat)rcConversationListTableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath;
   ```

4. 重写收到消息处理方法。

   在方法里生成新的 model，插入会话列表数据源 conversationListDataSource，更新页面。生成的 model 类型 conversationModelType 必须是 RC_CONVERSATION_MODEL_TYPE_CUSTOMIZATION。

```
#pragma mark - 收到消息监听
-(void)didReceiveMessageNotification:(NSNotification *)notification;
```

卸载重装或换设备登录后的处理方案

如果您的用户卸载重装或换设备登录，可能会发现会话列表为空，或者有部分会话丢失的错觉。

原因如下：

- 在卸载的时候会删除本地数据库，本地没有任何历史消息，导致重新安装后会话列表为空。
- 如果换设备登录，可能本地没有历史消息数据，导致会话列表为空。
- 如果您的 App Key 开启了多设备消息同步 功能，服务端会同时启用离线消息补偿功能。服务端会在 SDK 连接成功后自动同步当天 0 点后的消息，客户端 SDK 接收到服务端补偿的消息后，可生成部分会话和会话列表。与卸载前或换设备前比较，可能会有部分会话丢失的错觉。

如果您希望在卸载重装或换设备登录后，获取到之前的会话列表，可以参考如下方案：

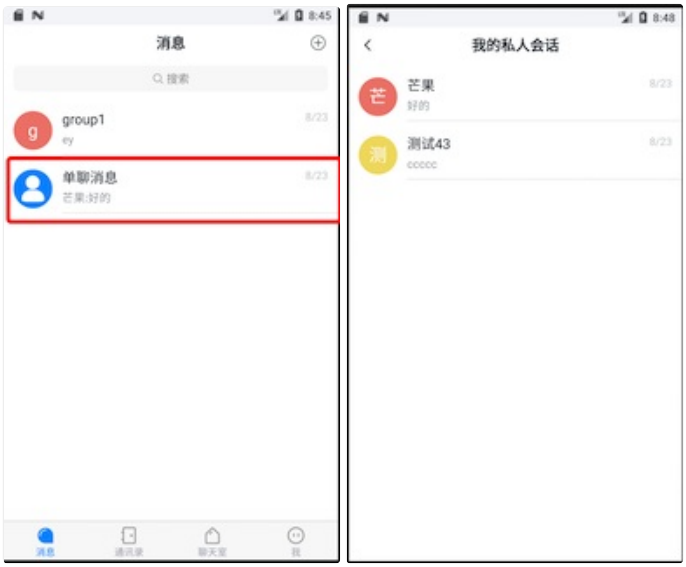
- 申请增加离线消息补偿的天数，最大可修改为 7 天。**注意，设置时间过长，当单用户消息量超大时，可能会因为补偿消息量过大，造成端上处理压力的问题。**如有需要，请提交工单 。
- 在您的服务器中自行维护会话列表，并通过 API 向服务端获取需要展示的历史消息。

按类型聚合会话 按类型聚合会话

更新时间:2024-08-30

IMKit 支持在会话列表中按照会话类型进行聚合（折叠）。设置聚合显示的会话类型后，该类型的会话在会话列表中仅被展示为一个聚合条目。默认情况下，IMKit 的会话列表页面会以平铺方式展示所有本地会话。

> 提示
>
> 下图展示了设置了单聊类型会话聚合显示的效果。在会话列表中，所有单聊会话折叠为“单聊消息”（左侧），点击后跳转到聚合会话列表（右侧）。IMKit 支持通过全局配置修改聚合会话页面的标题与头像。



用法

- 会话列表中的聚合会话项目不支持通过会话置顶功能设置为置顶项目。

设置聚合显示的会话类型

在初始化会话列表 RCConversationListViewController 或子类时，设置需要聚合的会话类型。支持单聊、群聊、系统会话类型。注意，您需要将 RCConversationType 转为 NSNumber构建 Array。

使用 collectionConversationTypeArray 参数传入聚合显示的会话类型。下例中我们聚合显示所有系统会话。

调用 RCConversationListViewController 类的初始化方法构建会话列表页面，在 displayConversationTypeArray 中设置会话列表中需要包含的会话的类型，在 collectionConversationTypeArray 中设置需要聚合（折叠显示）的会话类型。

```
NSArray *displayConversationTypeArray = @[@(ConversationType_PRIVATE), @(ConversationType_GROUP),
@(ConversationType_SYSTEM)];
NSArray *collectionConversationTypeArray = @[@(ConversationType_SYSTEM)];

RCConversationListViewController *conversationListVC = [[RCDChatListViewController alloc]
initWithDisplayConversationTypes:displayConversationTypeArray collectionConversationType:collectionConversationTypeArray];

[self.navigationController pushViewController:conversationListVC animated:YES];
```

参数	类型	说明
displayConversationTypeArray	NSArray (NSNumber *)	列表中需要显示的会话类型数组。需要将 RCConversationType 转为 NSNumber构建 Array。
collectionConversationTypeArray	NSArray (NSNumber *)	列表中需要聚合为一条显示的会话类型数组。您需要将 RCConversationType 转为 NSNumber构建 Array。

点击进入聚合会话列表

1. 您需要处理会话列表页面的点击事件，在点击聚合会话条目时，进入子会话列表。

```
/*!
点击会话列表中Cell的回调

@param conversationModelType 当前点击的会话的Model类型
@param model 当前点击的会话的Model
@param indexPath 当前会话在列表数据源中的索引值

@discussion 您需要重写此点击事件，跳转到指定会话的会话页面。
如果点击聚合Cell进入具体的子会话列表，在跳转时，需要将isEnteredToCollectionViewController设置为YES。
*/
- (void)onSelectedTableRow:(RCConversationModelType)conversationModelType
conversationModel:(RCConversationModel *)model
atIndexPath:(NSIndexPath *)indexPath;
```

2. 在进入子会话列表前，设置 RCConversationListViewController 子类的 isEnteredToCollectionViewController 属性为 YES。

```
//聚合会话类型，此处自定设置。
if (conversationModelType == RC_CONVERSATION_MODEL_TYPE_COLLECTION) {
YouConversationListViewController *temp = [[YouConversationListViewController alloc] init];
NSArray *array = [NSArray arrayWithObject:[NSNumber numberWithInteger:model.conversationType]];
[temp setDisplayConversationTypes:array];
[temp setCollectionConversationType:nil];
temp.isEnteredToCollectionViewController = YES;
[self.navigationController pushViewController:temp animated:YES];
}
```

定制化

您可以通过 IMKit 全局配置修改聚合会话条目的标题与头像。

自定义聚合会话标题

> 提示
>
> IMKit SDK 从 5.2.3 开始支持该功能。

IMKit 会话列表中，聚合条目的标题与会话类型相关。下表列出了各会话类型的聚合显示条目的标题字符串：

聚合会话类型	会话标题	资源名称
单聊	"聊天助手"	conversation_private_collection_title
群聊	"群助手"	conversation_group_collection_title
系统	"系统消息助手"	conversation_systemMessage_collection_title
讨论组	"讨论组助手"	conversation_discussion_collection_title

如果需要更改聚合会话标题，可以在进入会话列表之前，调用下面方法配置：

```
/*!
SDK会话列表界面聚合会话的标题

@discussion 如果不设置此项会使用内置默认头像。
key: 聚合会话类型 RCConversationType
value: 聚合会话标题 NSString
*/
@property (nonatomic, strong) NSDictionary<NSNumber *, NSString *> *globalConversationCollectionTitleDic;

RCKitConfigCenter.ui.globalConversationCollectionTitleDic = @{
@(ConversationType_PRIVATE): @"自定义聚合标题"
};
```

属性	类型	说明
globalConversationCollectionTitleDic	NSDictionary	聚合显示的会话标题配置
key	NSNumber(RCConversationType)	自定义的聚合会话类型
value	NSString	自定义的聚合会话标题

自定义聚合会话头像

> 提示
>
> IMKit SDK 从 5.2.3 开始支持该功能。

IMKit 中聚合会话的头像统一使用融云默认头像。调用下面的方法可以自定义聚合条目的头像。

```
/*!
SDK会话列表界面聚合会话的头像

@discussion 如果不设置此项会使用内置默认头像。
key: 聚合会话类型 RCConversationType
value: 图片路径（支持本地路径或者远端url）
*/
@property (nonatomic, strong) NSDictionary<NSNumber *, NSString *> *globalConversationCollectionAvatarDic;

// 远端图片
RCKitConfigCenter.ui.globalConversationCollectionAvatarDic = @{
@(ConversationType_PRIVATE): @"http://example.pic"
};

// 本地图片
RCKitConfigCenter.ui.globalConversationCollectionAvatarDic = @{
@(ConversationType_PRIVATE): [[NSBundle mainBundle] pathForResource:@"example" ofType:@"png"]
};
```

属性	类型	说明
globalConversationCollectionAvatarDic	NSDictionary	聚合显示的会话头像配置
key	NSNumber(RCConversationType)	自定义的聚合会话类型
value	NSString	自定义的聚合会话的头像 uri

会话页面　　会话页面

更新时间:2024-08-30

会话页面即应用程序中的聊天页面，主要由消息列表和输入区两部分组成。IMKit 提供基于 UIKit `UIViewController` 类实现的会话页面类 RCConversationViewController 。

聊天界面

聊天界面一般由三个部分组成：标题栏、消息列表、输入区域。

> 提示
>
> IMKit 默认会话页面 RCConversationViewController 未包含标题栏的实现，您需要自行设置会话标题。

初始化

> 提示
>
> 基于 IMKit 开发时，推荐继承 RCConversationViewController 类，创建自定义的会话页面。在排查或复现与会话页面相关的问题时，可以直接使用 RCConversationViewController 类。

调用 RCConversationViewController 类的初始化方法构建会话页面，设置 `conversationType` 和 `targetId` 的值来创建一个单聊会话或群聊会话页面。

```
if (self.navigationController) {
RCConversationViewController *conversationVC = [[RCConversationViewController alloc]
initWithConversationType:conversationType targetId:targetId];
[self presentViewController:navigationController animated:YES completion:nil];
}
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID

用法

在 RCConversationViewController 类中，会话页面主要由三个部分组成：

属性	类型	说明
navigationBar	UINavigationBar	iOS 系统的导航栏
conversationMessageCollectionView	RCBaseCollectionView	消息列表
chatSessionInputBarControl	RCChatSessionInputBarControl	输入区域

标题栏

IMKit 的 RCConversationViewController 使用了系统的导航栏，可用于显示会话的标题，例如在群聊时显示群组名称，在一对一聊天中显示另一个用户的名称。IMKit 默认不设置会话标题，应用程序在实现了 IMKit 的信息提供者协议后，可以调用 IMKit 的方法获取存储的用户资料数据，自行通过 `UIViewController` 的 `title` 属性设置标题。详情请参见用户信息。

开启输入状态功能后，在单聊会话中对方正在输入时，标题栏会被 SDK 修改为对方的输入状态。在搜索时，IMKit 会设置 `UINavigationItem` 的 `titleView`。

标题栏组件左侧提供一个按钮，当点击左按钮时，会调用 `leftBarButtonItemPressed` 方法退出当前页面，应用程序可以重写该方法。标题栏右侧默认无按钮，您可以在自定义页面上添加按钮。您可以参考 SealTalk 源码中 RCDChatViewController.m 中的 `rightBarButtonItemClicked` 方法。

消息列表

IMKit 的 RCConversationViewController 的消息列表按时间顺序显示所有消息。

消息列表的视图是在各类型消息的展示模板中创建和自定义的，分别使用了 `RCTextMessageCell`、`RCFileMessageCell`、`RCImageMessageCell` 等继承自 `RCMessageCell` 的类来显示会话中不同类型的消息。您可以在页面提供的回调方法中自定义列表视图中的每个项目，或者重写 `registerCustomCellsAndMessages` 方法，在其中调用 `registerClass:(Class)cellClass forMessageClass:(Class)messageClass;` 方法注册自定义的消息展示模板。

属性	类型	说明
conversationMessageCollectionView	RCBaseCollectionView	消息列表
conversationDataRepository	NSMutableArray	数据源中存放的元素为聊天页面中消息 Cell 的数据模型，即 `RCMessageModel` 对象。

> 提示
>
> 要了解与消息列表有关的事件，请转到 页面事件监听。

输入组件

`RCChatSessionInputBarControl` 是 IMKit 的消息输入组件，用户可以在其中键入和发送消息，支持文本、文件、语音、图像、视频等消息类型。输入区域的视图是在 RCChatSessionInputBarControl 中创建和控制的。

IMKit 已在 RCConversationViewController 中提供了部分输入组件相关的事件回调。

定制化

您可以通过修改 IMKit 全局配置和 RCConversationViewController 来自定义聊天页面。

在开始定制化之前，建议您首先继承 SDK 内置会话页面 RCConversationViewController，创建并实现自己的会话页面 View Controller。

```
#import <RongIMKit/RongIMKit.h>

@interface RCDChatViewController : RCConversationViewController
@property (nonatomic, assign) BOOL needPopToRootView;
@end
```

> 提示
>
> 会话页面中部分样式与行为受 IMKit 全局配置影响。如需了解全局配置，参见 配置指南。

修改用户头像形状与大小

会话页面中显示的用户头像（指 RCMessageCell 上显示的头像）可以通过 IMKit 全局配置修改。

头像显示大小默认值为 40*40。修改方法如下：

```
RCKitConfigCenter.ui.globalMessagePortraitSize = CGSizeMake(40, 40);
```

头像形状默认为矩形，可修改为圆角显示。您还可以通过 IMKit 全局配置修改圆角曲率。

```
RCKitConfigCenter.ui.globalMessageAvatarStyle = RC_USER_AVATAR_CYCLE;
```

> 提示
>
> 修改 IMKit 全局配置会影响 IMKit 中所有用户头像的显示。

显示用户昵称

您可以配置 IMKit 会话页面收到的消息是否显示用户昵称。

该功能默认值为 YES，即显示。应用程序可根据不同会话类型，在进入会话页面前设置为 YES 或 NO。

```
@property (nonatomic, assign) BOOL displayUserNameInCell;
```

隐藏输入区的 Emoji 表情按钮

> 提示
>
> SDK 从 5.2.3 开始支持禁用内置 Emoji。禁用后，表情面板不再显示 Emoji 标签页。

配置方式详见表情区域。

修改默认拉取历史消息数

IMKit 在进入会话页面时和下拉页面时获取历史消息，默认拉取 10 条。优先从本地拉取，本地消息拉取完之后，如果已开通单群聊历史消息云存储功能，IMKit 会再从远端拉取历史消息。您可以修改默认拉取的历史消息数。

请在进入会话页面前设置。

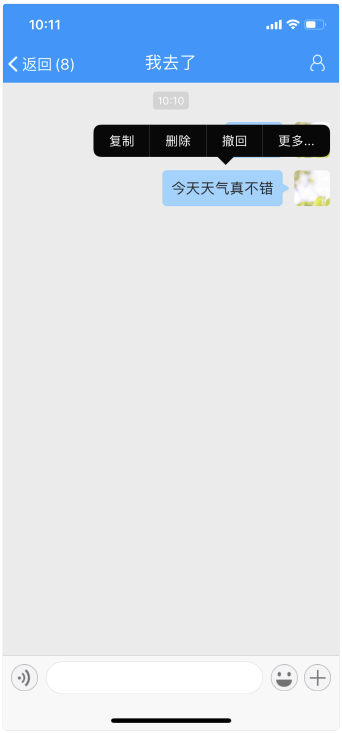
```
//拉取本地会话。此属性已被弃用，请使用 defaultMessageCount
@property (nonatomic, assign) int defaultLocalHistoryMessageCount;
//拉取远端会话。此属性已被弃用，请使用 defaultMessageCount
@property (nonatomic, assign) int defaultRemoteHistoryMessageCount;

/*!
设置进入会话页面后以及每次拉取时获取消息的条数，默认是 10，defaultMessageCount 传入 1 < count <= 100 之间的数值。
@discussion 此属性需要在viewDidLoad之前进行设置。
*/
@property (nonatomic, assign) int defaultMessageCount;
```

> 提示
>
> **您可以自助开通单群聊历史消息云存储功能，详见开通单群聊消息云存储服务。**

修改长按消息菜单中的删除行为

IMKit 会话页面默认已在长按消息弹出的菜单中实现了删除消息的选项。



- 如果 IMKit 版本 ≧ 5.6.3，默认实现的删除功能会删除本地消息，如果 App Key 已开通单群聊消息云端存储服务，则同步删除远端消息。
- 如果 IMKit 版本 < 5.6.3，默认实现的删除功能仅删除本地消息，如果 App Key 已开通单群聊消息云端存储服务，不会删除服务端历史消息记录中的该条消息。用户再次获取历史消息、或触发离线消息补偿（卸载重装、换设备登录）时，该消息可能会再次出现在聊天页面中。如需删除远端消息，应用程序可修改删除按钮的默认行为。

如需修改 IMKit 会话页面中删除消息选项的默认行为，方法如下：

- 如果 SDK 版本 ≧ 5.2.3，可设置 RCConversationViewController 的 `needDeleteRemoteMessage` 属性。YES 表示需要 IMKit 同步删除本地与远端消息。NO 表示仅从本地消息数据库中删除消息。

```
/*!
默认值为 NO 长按只删除本地消息，设置为 YES 时长按删除消息，会把远端的消息也进行删除
*/
@property (nonatomic, assign) BOOL needDeleteRemoteMessage;
```

- 如果 SDK 版本 < 5.2.3，需重写 RCConversationViewController 的 `deleteMessage` 方法，调用 super 删除本地消息，再调用

deleteRemoteMessage 删除远端消息接口，实现同时删除本地与远端消息。

```
/*!
删除消息并更新UI

@param model 消息Cell的数据模型
@discussion
v5.2.3 之前 会话页面只删除本地消息，如果需要删除远端历史消息，需要
1.重写该方法，并调用 super 删除本地消息
2.调用删除远端消息接口，删除远端消息
*/
- (void)deleteMessage:(RCMessageModel *)model;
```

> 提示
>
> 如果已有实现无法满足您的需求，可以直接使用 IMLib 提供的能力。具体的核心类、API 与 使用方法，详见 IMLib 文档 删除消息。注意：IMLib 中的方法并不提供页面刷新能力，您需要根据业务需求自定义通知机制进行页面刷新。

隐藏消息 Cell 上的用户头像

IMKit SDK 默认显示用户头像。从 5.3.0 版本开始，IMKit SDK 支持隐藏 RCMessageCell 子类消息 Cell 上的用户头像。

重写会话页面的 willDisplayMessageCell 方法，修改 RCMessageCell 的 showPortrait 属性为 NO，即可隐藏头像。

```
- (void)willDisplayMessageCell:(RCMessageBaseCell *)cell atIndexPath:(NSIndexPath *)indexPath {
...
if ([cell isKindOfClass:[RCMessageCell class]]) {
RCMessageCell *c = (RCMessageCell *)cell;
// 隐藏头像
c.showPortrait = NO;
}
...
[super willDisplayMessageCell:cell atIndexPath:indexPath];
}
```

自定义消息 Cell 显示

应用程序可以重写 RCConversationViewController 类的 willDisplayMessageCell 方法，在消息 Cell 展示前修改 UI。

例如，SDK 会话页面中文本消息已读的 UI 默认是一个“对勾”图标，如果希望修改为“已读”或“未读”，可以在 Cell 显示的时候，将 SDK 默认的图标移除，在对应位置添加文本“已读”或“未读”。应用程序还需要监听消息发送状态更新的通知，更新 cell。

下面代码仅以修改单聊文本消息 Cell 为例：

1. 重写 cell 即将显示的方法，修改 UI：

```
- (void)willDisplayMessageCell:(RCMessageBaseCell *)cell atIndexPath:(NSIndexPath *)indexPath {
RCMessageModel *model = [self.conversationDataRepository objectAtIndex:indexPath.row];
if ([cell isKindOfClass:[RCTextMessageCell class]] && model.conversationType == ConversationType_PRIVATE) {
if (model.content && (model.sentStatus == SentStatus_READ ||model.sentStatus == SentStatus_SENT) && model.messageDirection
== MessageDirection_SEND) {
for (UIView *view in [((RCMessageCell *)cell).statusContentView subviews]) {
[view removeFromSuperview];
}
CGRect statusContentViewFrame = ((RCMessageCell *)cell).statusContentView.frame;
UILabel *hasReadView = [[UILabel alloc]
initWithFrame:CGRectMake(statusContentViewFrame.size.width-30,statusContentViewFrame.size.height-16, 30, 16)];
hasReadView.textAlignment = NSTextAlignmentRight;
hasReadView.font = [UIFont systemFontOfSize:12];
hasReadView.textColor = [UIColor blackColor];
hasReadView.tag = 1001;
((RCMessageCell *)cell).statusContentView.hidden = NO;
if (model.sentStatus == SentStatus_READ) {
hasReadView.text = @"已读";
[((RCMessageCell *)cell).statusContentView addSubview:hasReadView];
} else if (model.sentStatus == SentStatus_SENT) {
hasReadView.text = @"未读";
[((RCMessageCell *)cell).statusContentView addSubview:hasReadView];
}
}else{
UIView *tagView = [((RCMessageCell *)cell).statusContentView viewWithTag:1001];
if (tagView) {
[tagView removeFromSuperview];
}
}
}
}
```

2. 监听消息发送状态更新的通知，执行刷新 cell 的方法:

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(messageCellUpdateSendingStatusEvent:)
name:KNotificationMessageBaseCellUpdateSendingStatus
object:nil];
```

```
- (void)messageCellUpdateSendingStatusEvent:(NSNotification *)notification {
RCMessageCellNotificationModel *notifyModel = notification.object;
if (notifyModel && ![notifyModel.actionName isEqualToString:CONVERSATION_CELL_STATUS_SEND_PROGRESS]) {
[self reloadMessageCell:notifyModel.messageId];
}
}
```

```
- (void)reloadMessageCell:(long)messageId {
__weak typeof(self) __weakself = self;
dispatch_async(dispatch_get_main_queue(), ^{
for (int i = 0; i < __weakself.conversationDataRepository.count; i++) {
RCMessageModel *model = (__weakself.conversationDataRepository)[i];
if (messageId == model.messageId) {
NSIndexPath *indexPath = [NSIndexPath indexPathForItem:i inSection:0];
[__weakself hideCellReceiptView:indexPath withMessageModel:model];
break;
}
}
});
}
```

```
- (void)hideCellReceiptView:(NSIndexPath *)indexPath withMessageModel:(RCMessageModel *)model {
UICollectionViewCell *__cell = [self.conversationMessageCollectionView cellForItemAtIndexPath:indexPath];
//如果是空说明被回收了，重新dequeue一个cell，这里用来解决已读人数闪的问题
if (__cell) {
[self.conversationMessageCollectionView reloadItemsAtIndexPaths:@[ indexPath ]];
if ([__cell isKindOfClass:[RCMessageCell class]]) {
dispatch_async(dispatch_get_main_queue(), ^{
((RCMessageCell *)__cell).statusContentView.hidden = YES;
});
}
}
}
```

定位到指定消息

您可以配置在进入 IMKit 会话页面时定位到指定消息，用于点击消息搜索结果后进入会话页面等场景。

请在进入会话页面前设置需要定位的消息的 sendTime。

```
@property (nonatomic, assign) long long locatedMessageSentTime;
```

其他定制化

您可以在以下文档中继续了解如何自定义会话页面：

- 未读消息数
- 修改消息的展示样式
- 表情区域
- 输入区域
- 页面事件监听

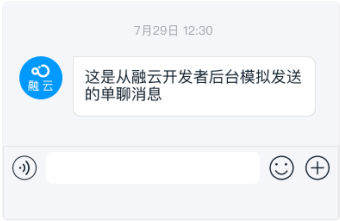
> 提示
>
> 您可以直接参考 IMKit 源码中的 RCConversationViewController.h 了解有关 RCConversationViewController 类的更多属性和方法。

输入区域 输入区域

更新时间:2024-08-30

IMKit 的输入区域是通过 RCChatSessionInputBarControl 统一创建和控制的，支持自定义输入模式、自定义扩展区域（插件）、以及自定义表情。

> 提示
>
> 下图输入区从左至右依次是语音/文本切换按钮、内容输入框、表情面板按钮、扩展面板按钮。



修改输入栏组合模式

IMKit 输入栏提供语音/文本切换、内容输入、扩展区域功能，并支持修改输入组合模式。例如，您可以移除语音/文本切换按钮和扩展面板，仅保留内容输入功能。

IMKit 在 RCChatSessionInputBarControlStyle 中定义了多种排列组合模式：

组合模式	枚举
语音/文本切换-输入框-扩展	RC_CHAT_INPUT_BAR_STYLE_SWITCH_CONTAINER_EXTENTION
扩展-输入框-切换	RC_CHAT_INPUT_BAR_STYLE_EXTENTION_CONTAINER_SWITCH
输入框-语音/文本切换-扩展	RC_CHAT_INPUT_BAR_STYLE_CONTAINER_SWITCH_EXTENTION
输入框-扩展-切换	RC_CHAT_INPUT_BAR_STYLE_CONTAINER_EXTENTION_SWITCH
语音/文本切换-输入框	RC_CHAT_INPUT_BAR_STYLE_SWITCH_CONTAINER
输入框-切换	RC_CHAT_INPUT_BAR_STYLE_CONTAINER_SWITCH
扩展-输入框	RC_CHAT_INPUT_BAR_STYLE_EXTENTION_CONTAINER
输入框-扩展	RC_CHAT_INPUT_BAR_STYLE_CONTAINER_EXTENTION
输入框	RC_CHAT_INPUT_BAR_STYLE_CONTAINER

您可以在会话页面 RCConversationViewController 的 viewDidLoad 之后改变输入栏的样式。使用 chatSessionInputBarControl 的 setInputBarType 方法调整顺序或禁用部分输入模式等。IMKit 在 RCChatSessionInputBarControlStyle 提供了多种排列组合。

```
RCChatSessionInputBarControlStyle style = RC_CHAT_INPUT_BAR_STYLE_SWITCH_CONTAINER_EXTENTION;
[self.yourChatVC.chatSessionInputBarControl setInputBarType:RCChatSessionInputBarControlDefaultType style:style];
```

隐藏输入栏中的表情面板按钮

> 提示
>
> IMKit SDK 从 5.3.2 版本开始提供该功能。

输入栏中的表情按钮是表情面板的入口。在 App 不需要提供表情输入功能时，可隐藏该入口。在会话页面显示前设置 hideEmojiButton 隐藏表情

面板入口。

```
[self.yourChatVC.chatSessionInputBarControl.inputContainerView.hideEmojiButton = YES];
```

输入区域扩展面板

融云将点击输入栏 + 号按钮时展示的面板称为扩展面板，扩展面板上展示当前可用的插件。

扩展面板插件列表

IMKit 完整插件列表如下所示。扩展面板中默认仅包含内置插件（照片、文件）。如需其他插件，请集成 IMKit 提供的插件库。

所属模块名称	插件名称	说明
内置插件	Photo	含相册与拍摄插件，默认支持拍摄照片发送，和从相册发送照片。集成小视频插件后可从相册发送小视频和录制小视频。
内置插件	File	文件插件，需要手动启用。启用后可发送文件。
RongLocationKit	LocationKit	发送位置消息，集成 `LocationKit` 库可用。
RongSight	Sight	小视频插件，集成 `Sight` 库后可用。
RongContactCard	ContactCard	名片插件，集成 `ContactCard` 库后可用。需配置联系人列表。
RongIFlyKit	IFly	语音输入插件，集成 `IFly` 库后可用。需要替换为您自己的讯飞 Framework（包含您自己的讯飞 App Key）。
RCCallKitExtensionModule	CallKit 语音通话插件	语音通话插件，集成 CallKit 后可用。
RCCallKitExtensionModule	CallKit 音视频通话插件	视频通话插件，集成 CallKit 后可用。

添加语音转文字插件

IMKit 基于讯飞 SDK 开发了语音转文字插件库 `IFly`。IMKit 集成 `IFLY` 库后，在扩展面板会自动生成语音输入功能入口。用户点击语音输入可以将录入的语音转成文字并展示在输入栏。

IMKit 的 `IFly` 库依赖于讯飞语音库 SDK `iflyMSC.framework`。讯飞语音库是收费业务，因此插件中不含讯飞 App Key。由于讯飞的 App Key 和 `iflyMSC.framework` 是强绑定关系，所以 App 在集成时必须替换 `iflyMSC.framework`。

请自行去讯飞官网下载 `iflyMSC.framework` SDK，用于替换语音输入插件中的 `iflyMSC.framework`。

语音插件（IFly）仅支持以源码方式导入。如需使用语音插件，必须同样以源码方式集成 IMKit SDK。参见导入SDK。

```
pod 'RongCloudOpenSource/IFly','~> x.y.z' # 语音输入
```

> 提示
>
> **x.y.z 代表具体版本，请通过融云官网 SDK 下载页面 或 CocoaPods 仓库等方式查询最新版本。**

集成步骤如下:

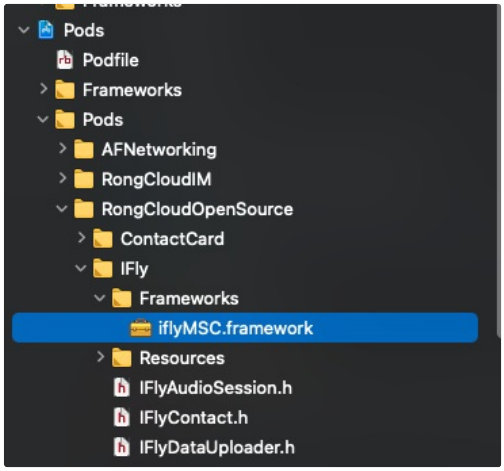
1. 前往讯飞官网，申请账号并下载 SDK。
2. 设置讯飞 App Key。
   1. 引用语音输入模块。

```
#import <RongCloudOpenSource/RongiFlyKit.h>
```

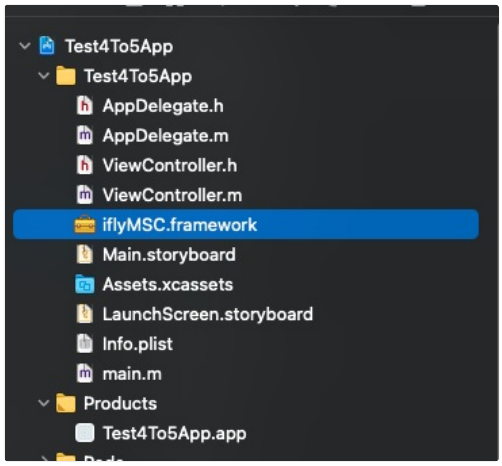
   2. 请在 RCIM 的 `initWithAppKey` 前调用下面接口。

```
[RCiFlyKit setiFlyAppKey:@"讯飞 appkey"];
```

3. 删除 `RongCloudOpenSource` 里的 `iflyMSC.framework`。`iflyMSC.framework` 在 Xcode 项目路径为 ：Pods -> Pods -> RongCloudOpenSource -> IFly -> Frameworks -> iflyMSC.framework



4. 配置 App 依赖您自行下载的 `iflyMSC.framework`。



添加其他插件

要了解如何在扩展面板添加其他插件，请参阅以下文档：

- 语音消息
- 小视频消息
- 位置消息
- 名片消息
- 集成 CallKit 通话功能

动态配置扩展面板插件

App 可以添加、删除、更新扩展面板上展示的项目，或自定义点击事件。

SDK 默认的扩展项的唯一标识符为 **1XXX**。如果您添加自定义扩展项，tag 值请勿使用 **1XXX**，否则会与 SDK 预留的扩展项唯一标识符重复。

插件	扩展项定义名	Tag 值
照片	PLUGIN_BOARD_ITEM_ALBUM_TAG	1001
拍摄	PLUGIN_BOARD_ITEM_CAMERA_TAG	1002
位置	PLUGIN_BOARD_ITEM_LOCATION_TAG	1003
文件	PLUGIN_BOARD_ITEM_FILE_TAG	1006
语音通话	PLUGIN_BOARD_ITEM_VOIP_TAG	1101
视频通话	PLUGIN_BOARD_ITEM_VIDEO_VOIP_TAG	1102
语音输入	PLUGIN_BOARD_ITEM_VOICE_INPUT_TAG	1105
个人名片	PLUGIN_BOARD_ITEM_CARD_TAG	1107

更多标识定义请参考 `RCChatSessionInputBarControl` 类。

添加扩展项

您可以在 RCConversationViewController 子类的 `viewdidload` 触发后，添加自定义的扩展项。向扩展功能面板中插入扩展项。请勿使用 **1XXX** 范围的 tag 值，否则会与 SDK 预留的扩展项唯一标识符重复。

```
[self.chatSessionInputBarControl.pluginBoardView insertItem:[UIImage imageNamed:@"plugin_item_poke"]
highlightedImage:[UIImage imageNamed:@"plugin_item_poke_highlighted"] title:RCDLocalizedString(@"Poke") tag:20090];
```

参数	类型	说明
normalImage	UIImage	扩展项的展示图片
highlightedImage	UIImage	扩展项的触摸高亮图片
title	NSString	扩展项的展示标题
index	NSInteger	需要添加到的索引值
tag	NSInteger	扩展项的唯一标识符

更新指定扩展项

根据 `tag` 更新扩展功能面板中的指定扩展项，支持更新融云提供的扩展项或 App 自定义的扩展项。

```
[self.chatSessionInputBarControl.pluginBoardView updateItemWithTag:101 normalImage:[UIImage imageNamed:@"plugin_item_poke"]
highlightedImage:[UIImage imageNamed:@"plugin_item_poke_highlighted"] title:newTitle];
```

参数	类型	说明
tag	NSInteger	扩展项的唯一标识符
normalImage	UIImage	扩展项的展示图片
highlightedImage	UIImage	扩展项的触摸高亮图片
title	NSString	扩展项的展示标题

删除指定扩展项

根据 tag 删除扩展功能面板中的指定扩展项，支持删除融云提供的扩展项或 App 自定义的扩展项。

```
[self.chatSessionInputBarControl.pluginBoardView removeItemWithTag:PLUGIN_BOARD_ITEM_LOCATION_TAG];
```

参数	类型	说明
tag	NSInteger	扩展项的唯一标识符

处理扩展项点击事件

在会话页面重写下面方法，根据 tag 判断是否为自定义的扩展项点击事件。如果不是，则需要调用 super，否则会影响默认扩展功能的点击功能。

```
- (void)pluginBoardView:(RCPluginBoardView *)pluginBoardView clickedItemWithTag:(NSInteger)tag;
```

参数	类型	说明
pluginBoardView	RCPluginBoardView	输入扩展功能板View
tag	NSInteger	输入扩展功能(Item)的唯一标识

修改消息的展示样式　　修改消息的展示样式

更新时间:2024-08-30

IMKit SDK 通过「消息展示模板」控制会话页面中消息的展示样式，IMKit 中所有消息模板都继承自 RCMessageCell。App 可以按需修改指定类型消息的展示模板，实现对消息展示样式的个性化配置。

- SDK 默认为会话页面中需要展示的内置消息类型（详见消息类型概述）提供了展示模板，App 可以按需创建模板，替换默认模板。
- App 创建的自定义消息类型（详见自定义消息类型）默认没有对应的消息展示模板。如果 App 需要在会话界面中展示该自定义类型的消息，则必须创建对应的消息展示模板，提供给 SDK。否则 SDK 无法正常展示该类型消息。

通过替换 Bundle 资源修改消息 UI

替换消息背景图

IMKit 会话页面中每条消息都有气泡背景，蓝色气泡为发送的消息，白色气泡为接收的消息。

以下为 RongCloud.bundle 内的部分资源：

资源名称	描述
chat_from_bg_normal	接收的消息背景。
chat_to_bg_normal	发出的消息背景。
chat_to_bg_white	发出的位置消息、名片消息、文件消息专用背景。

注意拉伸位置。SDK 内拉伸比例为：

```
UIEdgeInsetsMake(image.size.height * 0.5, image.size.width * 0.5, image.size.height * 0.5, image.size.width * 0.5)
```

修改内置文本消息的字体颜色或字体大小

IMKit SDK 支持通过全局配置修改文本字体大小。使用 RCKitConfigCenter 宏（或者使用 [RCKitConfig defaultConfig]）修改 IMKit 功能配置。您可以查看 [RCKitFontConf.h] 了解更多配置。

```
// 二级标题，默认 fontSize 为 17（文本消息，引用消息内容，会话列表 title）
RCKitConfigCenter.font.secondLevel = 20;
```

如果修改字体颜色，可以使用重写会话页面的 willDisplayMessageCell 方法，根据不同消息类型修改 cell 控件颜色。详见页面事件监听。

替换内置消息默认展示模板

IMKit 为每种类型的消息都封装了对应的消息展示模板。所有的消息展示模板都继承自 RCMessageCell。

替换内置文本消息的展示模板

> 提示
>
> IMKit 从 5.2.5 版本开始提供 RCComplexTextMessageCell。

IMKit 默认使用 RCTextMessageCell 显示 RCTextMessage 类型的消息。 RCTextMessageCell 为同步绘制 UI 的消息 Cell，在部分场景下，可能影响会话页面展示消息的流畅性。

- 字符比较多的复杂文本消息
- 含较多换行符的文本消息

从 5.2.5 版本开始，IMKit 支持将 RCTextMessage 类型消息的 Cell 替换为异步绘制 UI 的 RCComplexTextMessageCell，以更好地适应复杂文本的展示，减少会话页面上的卡顿和加载时间。

重写 RCConversationViewController 的 registerCustomCellsAndMessages 方法，在该方法中调用手动注册 RCComplexTextMessageCell。注册成功后，RCComplexTextMessageCell 将取代 RCTextMessageCell 用于显示文本消息。

```
// 注册自定义消息和cell
- (void)registerCustomCellsAndMessages {
[super registerCustomCellsAndMessages];

...
[self registerClass:[RCComplexTextMessageCell class] forMessageClass:[RCTextMessage class]];
....
}
```

自定义消息展示模板

如果您需要更改 SDK 内置消息的展示效果，且自定义需求较高，可以继承 RCMessageCell，创建新的消息展示模板，并将该自定义模板提供给 SDK，替换 SDK 默认模板。

IMKit 中会话页面中所有消息 Cell 继承自 RCMessageCell。您可以创建自定义消息 Cell，用于控制消息在会话页面的展示形式。自定义 Cell 可继承 RCMessageCell 或 RCMessageBaseCell。

继承 RCMessageBaseCell

RCMessageBaseCell 是 RCMessageCell 的父类，不支持显示头像和昵称。

RCMessageBaseCell 结构图：

如果继承 RCMessageBaseCell，请在初始化 Cell 时将在 baseContentView 上添加自定义控件。

继承 RCMessageCell

RCMessageCell 继承自 RCMessageBaseCell，增加了显示头像和昵称的特性。

如果继承 RCMessageCell，请在初始化 Cell 时将在 messageContentView 上添加自定义控件。请务必根据自定义控件大小调整消息内容区域（messageContentView）的 contentSize。

返回自定义 Cell 的 Size

重写 RCMessageCell 或 RCMessageBaseCell 的以下方法返回 Cell 的 Size。

```
+ (CGSize)sizeForMessageModel:(RCMessageModel *)model
withCollectionViewWidth:(CGFloat)collectionViewWidth
referenceExtraHeight:(CGFloat)extraHeight;
```

参数	类型	说明
model	RCMessageModel	要显示的消息 model
collectionViewWidth	CGFloat	Cell 所在的 collectionView 的宽度
extraHeight	CGFloat	Cell 内容区域之外的高度，就是上面 Cell 结构图中红色箭头的总高度 ，返回的 cell 的 size 的高 等于图中标注的 height + extraHeight ，size 的宽就是 collectionViewWidth

设置视图布局和数据

重写 RCMessageCell 或 RCMessageBaseCell 的设置 Cell 的视图布局和数据。请注意调整 messageContentView 的 contentSize，避免消息内容区域显示错乱。

```
- (void)setDataModel:(RCMessageModel *)model {
[super setDataModel:model];
// 修改布局与数据
}
```

以 IMKit 的 RCFileMessageCell 中的 setDataModel 方法为例：

```
- (void)setDataModel:(RCMessageModel *)model {
[super setDataModel:model];
RCFileMessage *fileMessage = (RCFileMessage *)self.model.content;
self.nameLabel.text = fileMessage.name;
self.sizeLabel.text = [RCKitUtility getReadableStringForFileSize:fileMessage.size];
self.typeIconView.image = [RCKitUtility imageWithFileSuffix:fileMessage.type];
[self setAutoLayout];
}
```

注册消息 Cell 并绑定消息类型

您需要向 IMKit 注册自定义的消息 Cell，同时与消息类型绑定。被绑定的消息类型可以是内置消息类型，或者应用开发者创建的自定义消息类型。如果绑定内置消息类型，则会替换默认的消息 Cell。

> 提示
>
> 关于如何创建自定义消息类型，详见创建自定义消息与注册自定义消息。如果您创建了自定义消息类型且需要展示在会话界面中，必须创建对应的消息展示模板，否则 SDK 无法正常展示该类型消息。

如果 IMKit >= 5.2.4，可重写会话页面的 `registerCustomCellsAndMessages` 方法，并在方法内注册自定义的消息 Cell ，并绑定消息类型。

```
- (void)registerCustomCellsAndMessages {
[super registerCustomCellsAndMessages];
///注册自定义测试消息Cell
[self registerClass:[RCDTestMessageCell class] forMessageClass:[RCDTestMessage class]];
}
```

如果 IMKit < 5.2.4，需要在会话页面 `viewDidLoad` 中注册自定义的消息 Cell ，并绑定消息类型。

```
[self registerClass:[RCDTestMessageCell class] forMessageClass:[RCDTestMessage class]];
```

> 提示
>
> 您还可以参考融云的 SealTalk 应用中的 RCDTestMessage.m 和 RCDTestMessageCell.m 。

隐藏消息 Cell 上的用户头像

IMKit SDK 默认显示用户头像。从 5.3.0 版本开始，IMKit SDK 支持隐藏 `RCMessageCell` 子类消息 Cell 上的用户头像。详见会话页面。

页面事件监听　　页面事件监听 监听会话列表页面事件

更新时间:2024-08-30

您可以设置会话列表操作监听，实现自定义需求。以下列出了 RCConversationListViewController.h 提供的常用事件，您也可以直接参考 IMKit 源码，查看所有事件。

即将显示会话 Cell

重写 RCConversationListViewController 的此方法，可修改 Cell 的一些显示属性，如对会话列表自带 Cell 样式如字体颜色，字体大小进行修改。不建议修改 Cell 的布局。如果对 UI 比较高的定制需求，建议自定义会话列表中的会话 Cell。

```
- (void)willDisplayConversationTableCell:(RCConversationBaseCell *)cell atIndexPath:(NSIndexPath *)indexPath;
```

参数	类型	说明
cell	RCConversationBaseCell	即将显示的 Cell
indexPath	NSIndexPath	该 Cell 对应的会话 Cell 数据模型在数据源中的索引值

点击会话 Cell

重写 RCConversationListViewController 的此方法，可跳转到您自定义的会话页面。

```
- (void)onSelectedTableRow:(RCConversationModelType)conversationModelType
conversationModel:(RCConversationModel *)model
atIndexPath:(NSIndexPath *)indexPath;
```

参数	类型	说明
conversationModelType	RCConversationModelType	当前点击的会话的 Model 类型
model	RCConversationModel	当前点击的会话的 Model
indexPath	NSIndexPath	当前会话在列表数据源中的索引值

点击会话头像

会话列表中的每个会话项目上均会显示一个头像图标，即会话头像（不是聊天页面内中消息列表中的头像）。单聊会话显示对方用户头像，群聊会话显示群组头像，聚合会话显示默认头像或应用程序主动设置的头像。

重写 RCConversationListViewController 的此方法，可自定义处理该事件。

```
- (void)didTapCellPortrait:(RCConversationModel *)model;
```

参数	类型	说明
model	RCConversationModel	会话 Cell 的数据模型

长按会话头像

会话列表中的每个会话项目上均会显示一个头像图标，即会话头像（不是聊天页面内中消息列表中的头像）。单聊会话显示对方用户头像，群聊会话显示群组头像，聚合会话显示默认头像或应用程序主动设置的头像。

重写 RCConversationListViewController 的此方法，可自定义处理该事件。

```
- (void)didLongPressCellPortrait:(RCConversationModel *)model;
```

参数	类型	说明
model	RCConversationModel	会话 Cell 的数据模型

删除会话

重写 RCConversationListViewController 的此方法，可自定义处理该事件。

```
- (void)didDeleteConversationCell:(RCConversationModel *)model;
```

参数	类型	说明
model	RCConversationModel	会话 Cell 的数据模型

即将加载数据源

重写 RCConversationListViewController 的此方法，可修改、添加、删除数据源的元素来定制显示的内容，会话列表会根据您返回的修改后的数据源进行显示。数据源中存放的元素为会话 Cell 的数据模型，即 RCConversationModel 对象。

```
- (NSMutableArray *)willReloadTableData:(NSMutableArray *)dataSource;
```

参数	类型	说明
dataSource	NSMutableArray	即将加载的增量数据源

> 提示
>
> dataSource 为增量数据，conversationListDataSource += dataSource。如果需要更改全量数据的内容，可以更改 conversationListDataSource。

即将更新未读消息数

当收到消息或删除会话时，会调用此回调。重写 RCConversationListViewController 的此方法，可以执行未读消息数相关的操作。

```
- (void)notifyUpdateUnreadMessageCount;
```

> 提示
>
> 该方法在非主线程回调，如果想在本方法中操作 UI，请手动切换到主线程。

在会话列表中收到新消息

重写 RCConversationListViewController 的此方法，可自定义处理该事件。

```
- (void)didReceiveMessageNotification:(NSNotification *)notification;
```

参数	类型	说明
notification	NSNotification	收到新消息的 notification。notification 的 object 为 RCMessage 消息对象。userInfo 为 NSDictionary 对象，其中 key 值为 @"left"，value 为还剩余未接收的消息数的 NSNumber 对象。

> 提示
>
> SDK 在此方法中针对消息接收有默认的处理（如刷新等），如果重写此方法，请注意调用 super。

监听会话页面事件

您可以设置会话页面操作监听，实现自定义需求。以下列出了 RCConversationViewController.h 提供的常用事件，您也可以直接参考 IMKit 源码，查看所有事件。

输入框内容发生变化

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)inputTextView:(UITextView *)inputTextView shouldChangeTextInRange:(NSRange)range replacementText:(NSString *)text;
```

参数	类型	说明
inputTextView	UITextView	文本输入框
range	NSRange	当前操作的范围
text	NSString	插入的文本

输入框高度发生变化

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)chatInputBar:(RCChatSessionInputBarControl *)chatInputBar shouldChangeFrame:(CGRect)frame;
```

参数	类型	说明
chatInputBar	RCChatSessionInputBarControl	输入工具栏
frame	CGRect	输入工具栏最终需要显示的 Frame

> 提示
>
> 如重写此方法，请先调用父类方法。

准备发送消息

准备向外发送消息时会触发该回调（不支持通过插件发送的位置消息、小视频、融云贴纸表情消息）。重写 RCConversationViewController 的此方法，可自定义处理该事件。此回调的返回值不为 `nil` 时，SDK 会对外发送返回的消息内容。

```
- (RCMessageContent *)willSendMessage:(RCMessageContent *)messageContent;
```

参数	类型	说明
messageContent	RCMessageContent	消息内容

> 提示

> 通过位置插件、名片插件、融云贴纸表情发送的消息不会触发该回调。如果希望对消息进行拦截、过滤、修改等操作，建议使用 RCIMMessageInterceptor 协议设置拦截器，详见拦截消息。

发送消息完成

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didSendMessage:(NSInteger)status content:(RCMessageContent *)messageContent;
```

参数	类型	说明
status	NSInteger	发送状态，0表示成功，非0表示失败
messageContent	RCMessageContent	消息内容

取消消息发送

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didCancelMessage:(RCMessageContent *)messageContent;
```

参数	类型	说明
messageContent	RCMessageContent	消息内容

即将将消息插入数据源

在消息准备插入数据源的时候会回调。重写 RCConversationViewController 的此方法，可对消息进行过滤和修改操作。 如果此回调的返回值不为 nil，SDK 会将返回消息实体对应的消息 Cell 数据模型插入数据源，并在会话页面中显示。

```
- (RCMessage *)willAppendAndDisplayMessage:(RCMessage *)message;
```

参数	类型	说明
message	RCMessage	消息内容

即将显示消息

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)willDisplayMessageCell:(RCMessageBaseCell *)cell atIndexPath:(NSIndexPath *)indexPath;
```

参数	类型	说明
cell	RCMessageBaseCell	消息 Cell
indexPath	NSIndexPath	该 Cell 对应的消息Cell数据模型在数据源中的索引值

显示未注册消息的 Cell

重写 RCConversationViewController 的此方法，可设置未知消息类型（未注册的消息类型）的 Cell。

```
- (RCMessageBaseCell *)rcUnkownConversationCollectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath;
```

返回的 RCMessageBaseCell 及未注册消息需要显示的 Cell。

参数	类型	说明
collectionView	UICollectionView	当前 CollectionView
indexPath	NSIndexPath	该Cell对应的消息Cell数据模型在数据源中的索引值

显示未注册消息的 Cell 的高度

重写 RCConversationViewController 的此方法，可设置未知消息类型（未注册的消息类型）的 Cell 的显示高度。

```
- (CGSize)rcUnkownConversationCollectionView:(UICollectionView *)collectionView
layout:(UICollectionViewLayout *)collectionViewLayout
sizeForItemAtIndexPath:(NSIndexPath *)indexPath;
```

返回的 CGSize 为未注册消息 Cell 需要显示的高度。

参数	类型	说明
collectionView	UICollectionView	当前CollectionView
collectionViewLayout	UICollectionViewLayout	当前CollectionView Layout
indexPath	NSIndexPath	该Cell对应的消息Cell数据模型在数据源中的索引值

点击消息 Cell 头像

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didTapCellPortrait:(NSString *)userId;
```

参数	类型	说明
userId	NSString	点击头像对应的用户ID

长按消息 Cell 头像

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didLongPressCellPortrait:(NSString *)userId;
```

参数	类型	说明
userId	NSString	点击头像对应的用户 ID

点击消息 Cell 内容

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didTapMessageCell:(RCMessageModel *)model;
```

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型

> 提示

IMKit 在此点击事件中融云定义的图片、语音、位置等消息有默认的处理，如查看、播放等。重写此回调时，如果想保留 SDK 原有的功能，需要注意调用 super。

长按消息 Cell 内容

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didLongTouchMessageCell:(RCMessageModel *)model inView:(UIView *)view;
```

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型
view	UIView	长按区域的View

提示

重写此回调时，如果想保留 SDK 原有的功能，需要注意调用 super。

点击消息 Cell 中的 URL

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didTapUrlInMessageCell:(NSString *)url model:(RCMessageModel *)model;
```

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型
url	NSString	点击的URL

提示

重写此回调时，如果想保留 SDK 原有的功能，需要注意调用 super。

点击消息 Cell 中的电话号码

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)didTapPhoneNumberInMessageCell:(NSString *)phoneNumber model:(RCMessageModel *)model;
```

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型
phoneNumber	NSString	点击的电话号码

获取长按消息 Cell 的菜单

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (NSArray<UIMenuItem *> *)getLongTouchMessageCellMenuList:(RCMessageModel *)model;
```

参数	类型	说明
model	RCMessageModel	消息 Cell 的数据模型

> 提示
>
> 重写此回调时，如果想保留 SDK 原有的功能，需要注意调用 super。

开始录制录音

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)onBeginRecordEvent;
```

结束录制录音

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)onEndRecordEvent;
```

取消录制录音

重写 RCConversationViewController 的此方法，可自定义处理该事件。

```
- (void)onCancelRecordEvent;
```

点击常用语按钮事件

> 提示
>
> 要求 IMKit 版本 ≧ 5.6.3。

如果启用了 IMKit 快捷回复 功能，用户在会话页面点击常用语按钮后会弹出快捷回复。

您可以重写 RCConversationViewController 的以下方法，返回 YES 表示拦截，您可以自定义点击常用语按钮后的逻辑；否则返回 NO，继续执行 SDK 默认逻辑。

```
- (BOOL)didTapCommonPhrasesButton;
```

消息全部拉取完成事件

> 提示
>
> 此功能要求 IMKit SDK 版本 ≧ 5.8.2。

在实现消息拉取功能时，您可能需要知道何时已经拉取完所有可用的远端消息。为了获取这样的通知事件，您可以重写 RCConversationViewController 类中的 noMoreMessageToFetch 方法。

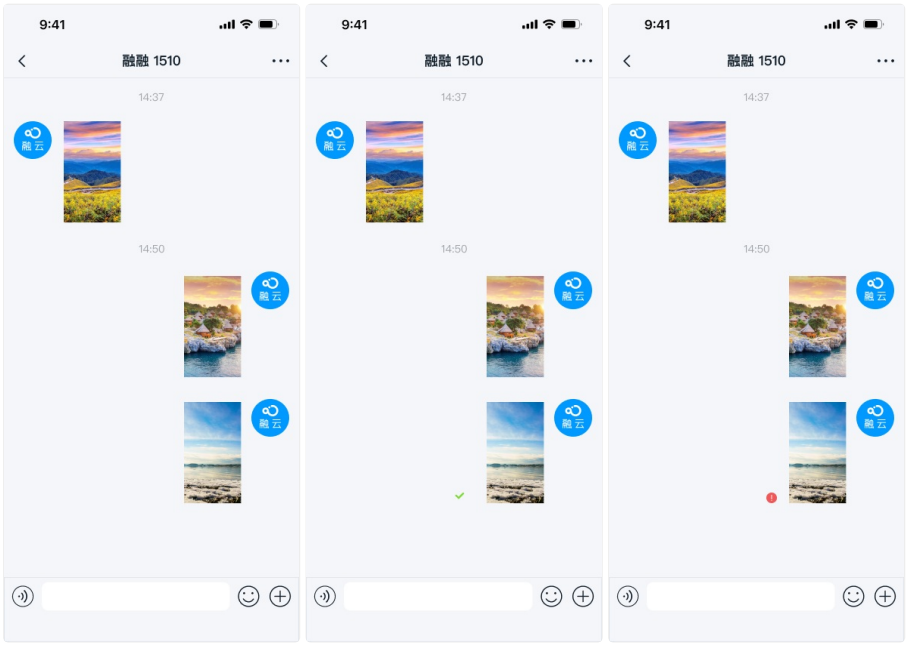
```
- (void)noMoreMessageToFetch;
```

图片和 GIF 消息 图片和 GIF 消息

更新时间:2024-08-30

用户可以通过 IMKit 内置的图片插件发送图片消息和 GIF 消息。消息将出现在会话页面的消息列表组件中。SDK 默认发送消息包含以下消息内容对象：

- 图片消息内容类为 RCImageMessage（类型标识：`RC:ImgMsg`）
- GIF 消息内容类为 RCGIFMessage（类型标识：`RC:GIFMsg`）



局限

- 仅支持发送本地图片和 GIF。
- GIF 文件大小上限为 2 MB。
- 图片消息和 GIF 消息中的文件默认会上传到融云的服务器。如需上传到自己的服务器，您需要拦截消息，自行上传。详见拦截消息。

用法

扩展面板里默认带有发送图片消息入口，由 IMKit 内置的 Photo 实现。用户点击输入栏右侧 + 号按钮可展开扩展面板，点击图片图标，即可打开本地相册，选择图片、GIF 文件进行发送。

定制化

调整图片压缩质量

在发送前，图片会被压缩质量，以及生成缩略图，在聊天界面中展示。GIF 无缩略图，也不会被压缩。

- **图片消息的缩略图**：SDK 会以原图 30% 质量生成符合标准大小要求的大图后再上传和发送。压缩后最长边不超过 240 px。缩略图用于在聊天界面中展示。
- **图片**：发送消息时如未选择发送原图，SDK 会以原图 85% 质量生成符合标准大小要求的大图后再上传和发送。压缩后最长边不超过 1080 px。

一般情况下不建议修改 SDK 默认压缩配置。如需调整 SDK 压缩质量，详见知识库文档如何修改 SDK 默认的图片与视频压缩配置 。

自定义图片、GIF 消息的 UI

图片消息与 GIF 消息默认使用以下模板展示在消息列表中。

- `RCImageMessageCell`
- `RCGIFMessageCell`

如果需要调整内置消息样式，建议自定义消息 Cell，并将该自定义 Cell 提供给 SDK。IMKit 中所有消息模板都继承自 `RCMessageCell`，自定义消息 Cell 也需要继承 `RCMessageCell`。详见修改消息的展示样式。

您也可以直接替换 `RongCloud.bundle` 中消息展示模板中引用的样式资源、字符串资源和图标资源。详见 IMKit 源码 RCImageMessageCell.m 、RCGIFMessageCell.m 。

自定义相册列表页 UI

资源	修改方法
左上/右上按钮	可设置 `UIBarButtonItem` 的全局 tintColor 来更改。
图片右上角点击选中按钮	可替换图片资源中的 `photopicker_state_normal` 和 `photopicker_state_selected` 来更改。
其他按钮	可通过 `color.plist`中 photoPicker 字典里的相关字段来更改。文件路径:`IMKit/Resources/RCColor.plist`

自定义相册预览页 UI

资源	修改方法
左上角返回按钮	可替换图片资源中的 `navigator_btn_back` 来更改。
右上角选择按钮	可替换图片资源中的 `photo_preview_unselected` 和 `photo_preview_selected` 来更改。
其它按钮	可通过 `color.plist` 中 `photoPreview` 字典里的相关字段来更改。
选中发送原图时文字前面的选中状态	可替换图片资源中的 `selected_full` 和 `unselected_full` 来更改。

隐藏扩展面板中的图片入口

IMKit 默认在扩展面板中启用了图片入口。如需动态修改，可创建自定义的扩展面板配置类 `MyExtensionConfig`，继承自 `DefaultExtensionConfig`，重写其中的 `getPluginModules()` 方法。详见输入区域。

小视频消息 小视频消息

更新时间:2024-08-30

用户可以通过 IMKit 图库（本地相册）或小视频插件发送小视频消息。消息将出现在会话页面的消息列表组件中。插件默认发送的消息包含小视频消息内容对象 RCSightMessage（类型标识：RC:SightMsg）。

局限性

小视频功能目前存在以下限制：

- IMKit 仅单聊会话和群聊会话支持发送小视频消息。
- 如果使用小视频插件进行录制，支持录制长度不超过 10 秒的小视频。
- 如果从本地相册中选择视频文件，请注意服务端的默认视频时长上限为 2 分钟。如需调整上限，请联系商务。
- 仅支持 `H.264` + `AAC` 编码的视频文件，因为 IMKit 的短视频录制、播放只实现了该编码组合的支持。
- 如果 App Key 使用 **IM 旗舰版**或 **IM 尊享版**，文件存储时长默认为 180 天（不含小视频文件，小视频文件存储 7 天）。注意，**IM 商用版（已下线）默认存储 7 天。如需了解IM 旗舰版或 IM 尊享版**的具体功能与费用，请参见融云官方价格说明页面及计费说明。

用法

建议通过集成 IMKit 小视频插件使用小视频消息功能。

集成小视频插件

IMKit 的小视频插件实现了小视频消息的消息注册、录制、播放等功能。集成小视频模块后，在单聊、群组会话输入区域的扩展面板中自动出现发送小视频消息的入口。

请根据应用程序集成 IMKit 的方式，选择使用 Framework 和源码导入小视频插件。请务必不要混用 Framework 和源码集成方式。

- 导入小视频插件 Framework

```
pod 'RongCloudIM/Sight','~> x.y.z' #小视频
```

- 导入小视频插件源码

```
pod 'RongCloudOpenSource/Sight','~> x.y.z' #小视频
```

> 提示
>
> **x.y.z 代表具体版本，请通过融云官网 SDK 下载页面 或 CocoaPods 仓库等方式查询最新版本。**

从本地相册选择小视频

> 提示
>
> **前提条件：集成 IMKit 小视频插件后，插件内部会向 IMLib 注册（RCSightMessage）类型的消息。请先集成 IMKit 小视频插件，再进行以下配置，否则 SDK 无法识别小视频消息。**

通过 IMKit 输入区域中的照片插件打开本地相册时，默认不包含视频文件，用户无法选择视频文件进行发送。

您可以修改全局配置，设置在本地相册中包含视频文件。

```
//设为选择媒体资源时包含视频文件
RCKitConfigCenter.message.isMediaSelectorContainVideo = YES;
```

如果您的项目中不使用 IMKit 小视频插件，但仍希望支持从本地发送小视频文件，请自行向 IMLib 注册 RCSightMessage，否则 SDK 无法发送小视频消息。您还需要自行实现小视频录制、播放（在会话页面消息点击事件中处理播放）功能。如果 IMKit ≧ 5.2.4，重写会话页面的 registerCustomCellsAndMessages 方法，注册消息类型同时绑定展示模板。

```
- (void)registerCustomCellsAndMessages {
[super registerCustomCellsAndMessages];
///注册自定义测试消息Cell
[self registerClass:[RCSightMessageCell class] forMessageClass:[RCSightMessage class]];
}
```

详见修改消息的展示样式。

发送小视频消息

用户点击输入栏右侧 + 号按钮可展开扩展面板，点击照片或拍摄图标，即可发送小视频消息。

定制化

调整小视频压缩质量

小视频文件会被压缩为分辨率 544 ＊ 960 的文件。小视频首帧画面会被用于生成缩略图，在聊天界面中展示。SDK 默认以原图 30% 质量生成符合标准大小要求的缩略图后再上传和发送，缩略图最长边不超过 240 px。

一般情况下不建议修改 SDK 默认压缩配置。如需调整 SDK 压缩质量，详见知识库文档如何修改 SDK 默认的图片与视频压缩配置 。

自定义小视频消息的 UI

IMKit 默认生成和发送小视频消息（RC:SightMsg），使用 RCFileMessageCell 展示在消息列表中。如果需要调整内置消息样式，建议自定义消息 Cell，并将该自定义 Cell 提供给 SDK。

IMKit 中所有消息模板都继承自 RCMessageCell，自定义消息 Cell 也需要继承 RCMessageCell。详见修改消息的展示样式。

您也可以直接替换 RongCloud.bundle 中小视频消息展示模板中引用的样式资源、字符串资源和图标资源。详见 IMKit 源码 RCSightMessageCell.m 中引用的资源。

如果希望修改录制、播放 UI，可以参考 IMKit 小视频插件源码 Sight 。

动态隐藏小视频入口

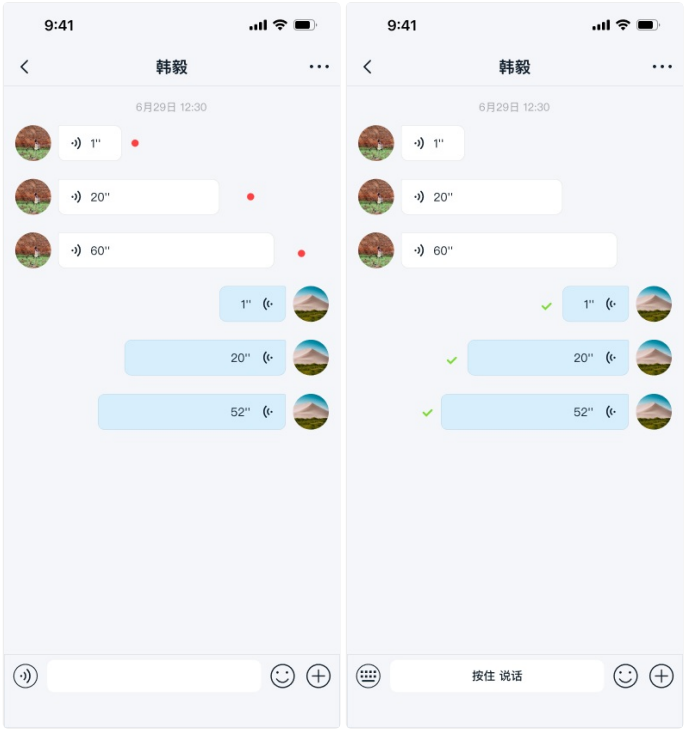
如需动态修改，可在会话页面显示前，通过 RCConversationViewController 的 chatSessionInputBarControl.pluginBoardView 删除指定扩展项目。

详见输入区域。

语音消息　　语音消息

更新时间:2024-08-30

用户可以通过 IMKit 内置的输入组件录制并发送语音消息。消息将出现在会话页面的消息列表组件中。SDK 默认生成和发送的消息包含高清语音消息内容对象 RCHQVoiceMessage（类型标识：RC:HQVcMsg）。



局限性

语音输入功能目前存在以下限制：

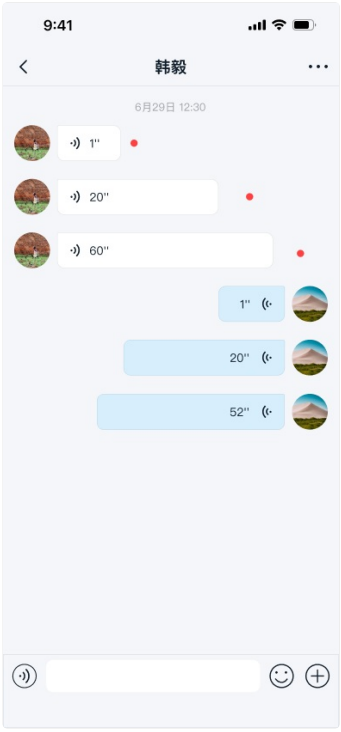
- IMKit 仅单聊会话和群聊会话支持发送语音消息。
- 用户必须录制至少为 1 秒的音频内容，且必须短于 60 秒钟。
- 用户在录制语音消息时无法暂停。
- 正在视频通话和语音通话中不能进行语音消息发送。

用法

IMKit 默认在输入栏组件中启用语音消息输入功能。输入栏中默认带有切换语音输入按钮。

发送语音消息

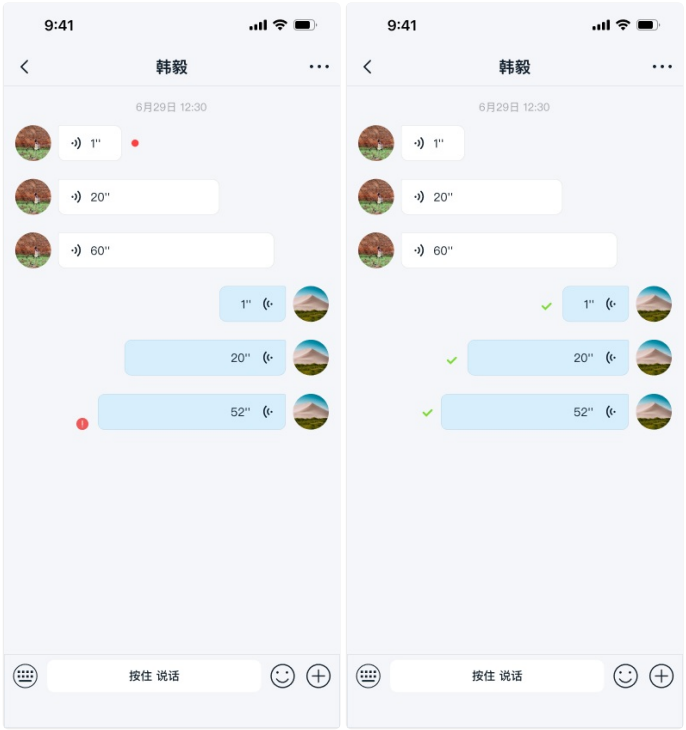
为了发送语音消息，用户必须首先在输入栏中通过 RCVoiceRecorder 录制消息。默认情况下，语音消息图标显示在输入字段的左侧。点击此图标后，就会出现录制按钮（“按住说话”）。用户可以通过点击录制按钮来录制语音消息。长度必须至少为一秒且短于 60 秒钟。如果在点击停止按钮之前消息不到一秒，则不会保存该消息。录制过程中可以上滑取消录制或放弃取消。一旦松开按钮，SDK 默认发送到目前为止录制的内容。不支持在发送语音消息之前预览。在播放语音消息中的音频文件时不可暂停。



消息列表中的语音消息

您可以在单聊会话、群聊会话、系统会话中接收语音消息。语音消息显示在消息列表中。

用户可以通过点击播放按钮查看和播放语音消息。未播放的语音消息旁边会显示一个红点，消息可多次播放。但是，用户只能在客户端应用程序中收听语音消息，并且无法将其保存到自己的设备中。您在频道中一次只能收听一条音频文件。如果您在收听消息时尝试播放另一条消息，则先播放的消息将暂停。



IMKit 默认下载高清语音消息，并且默认点击播放后连续播放消息下方未收听的语音消息。您可以在会话页面显示前，通过 RCConversationViewController 的 `enableContinuousReadUnreadVoice` 属性设置未听的语音消息不连续播放。

NO 表示禁止连续播放。

```
chatVC.enableContinuousReadUnreadVoice = NO;
[self.navigationController pushViewController:chatVC animated:YES];
```

定制化

语音消息的定制化涉及到输入栏的语音输入切换图标、音频录制界面、和高清语音消息展示 UI。

自定义语音消息的 UI

IMKit 默认生成和发送高清语音消息（RC:HQVoiceMsg），使用 RCHQVoiceMessageCell 模板展示在消息列表中。

IMKit 中所有消息模板都继承自 RCMessageCell，自定义消息 Cell 也需要继承 RCMessageCell。详见修改消息的展示样式。

您也可以直接替换 RongCloud.bundle 中小视频消息展示模板中引用的样式资源、字符串资源和图标资源。详见 IMKit 源码 RCHQVoiceMessageCell.m 中引用的资源。

输入栏与录音弹窗样式资源

您也可以直接替换 RongCloud.bundle 中输入栏、录音弹窗相关样式资源。下表列出了部分图标。

图标	图像	描述
inputbar_voice		用作语音输入按钮的图标，用于在消息输入时切换至语音消息录制视图。inputbar_voice_dark 对应暗黑模式下的图标。
voice_unread		本端用户发送的高清语音消息未读状态。

更多图标资源，请参考 IMKit 源码中的 RCHQVoiceMessageCell.m 、RCVoiceCaptureControl.m 等。

处理录音相关事件

IMKit 在 RCConversationViewController 中提供了录音相关的事件：

- `(void)onBeginRecordEvent;`：开始录制语音消息
- `(void)onEndRecordEvent;`：结束录制语音消息
- `(void)onCancelRecordEvent;`： 取消录制语音消息的回调，不会再触发 `onEndRecordEvent`

详见页面事件监听。

隐藏切换语音输入按钮

您可以在会话页面 viewDidLoad 之后，设置，通过 RCConversationViewController 的 chatSessionInputBarControl 属性改变输入工具栏的样式 RCChatSessionInputBarControlStyle，移除语音/文本切换按钮。

```
RCChatSessionInputBarControlStyle style = RC_CHAT_INPUT_BAR_STYLE_CONTAINER_EXTENTION;
[self.yourChatVC.chatSessionInputBarControl setInputBarType:RCChatSessionInputBarControlDefaultType style:style];
```

文件消息　　文件消息

更新时间:2024-08-30

用户可以通过 IMKit 内置的文件插件发送文件消息。消息将出现在会话页面的消息列表组件中。文件插件默认发送的消息包含文件消息内容对象 RCFileMessage（类型标识：RC:FileMsg）

局限

- 仅支持发送本地文件。
- 文件消息中的文件默认会上传到融云的服务器。如需上传到自己的服务器，您需要拦截消息，自行上传。详见拦截消息。
- 不支持在 IMKit 中预览文件，请在 UI 中选择用其他应用打开。

用法

IMKit 内置的 File 插件实现了扩展面板中的文件消息功能。但是为了向后兼容，目前 SDK 默认不开启该功能入口。

您可以在会话页面 viewDidLoad 中添加以下代码，以支持发送文件功能。

```
[self.chatSessionInputBarControl.pluginBoardView insertItem:RCResourceImage(@"plugin_item_file")
highlightedImage:RCResourceImage(@"plugin_item_file_highlighted")
title:RCLocalizedString(@"File")
atIndex:3
tag:PLUGIN_BOARD_ITEM_FILE_TAG];
```

发送文件消息

启用文件消息功能后，扩展面板会出现发送文件消息入口。用户点击输入栏右侧 + 号按钮可展开扩展面板，点击文件图标，即可发送文件消息。

定制化

替换文件消息默认的文件图标

文件消息（RCFileMessage）在会话界面中显示时，会根据消息携带的文件类型展示匹配的图标。SDK 默认为以下类型的文件提供了匹配的图标，如果为其他类型文件，则默认显示统一的默认图标。

- **图片类**：bmp、cod、gif、ief、jpe、jpeg、jpg、jfif、svg、tif、tiff、ras、ico、pbm、pgm、png、pnm、ppm、xbm、xpm、xwd、rgb
- **文本类**：txt、log、html、stm、uls、bas、c、h、rtx、sct、tsv、htt、htc、etx、vcf
- **视频类**：rmvb、avi、mp4、mp2、mpa、mpe、mpeg、mpg、mpv2、mov、qt、lsf、lsx、asf、asr、asx、avi、movie、wmv
- **音频类**：mp3、au、snd、mid、rmi、aif、aifc、aiff、m3u、ra、ram、wav、wma
- **Word 类**：doc、dot、docx
- **Excel 类**：xla、xlc、xlm、xls、xlt、xlw、xlsx

SDK 从 5.3.4 版本开始，支持 App 修改文件类型（扩展名）对应显示的图标。App 可以按需更新指定图标，替换全部图标，或增加文件类型（扩展名）及图标。注意，统一的默认图标暂无法替换。

```
[RCKitConfigCenter.ui registerFileSuffixTypes:types];
```

参数	类型	说明
types	NSDictionary	文件消息图标配置。key 是不带 . 的文件扩展名（例：png、pdf）。value 是本地文件路径。如果文件路径为空或者路径下的文件不存在，会使用 RongCloud.bundle 中的默认图标。本地文件路径下的图片尺寸请参考 RongCloud.bundle 中对应的文件图标。

自定义文件消息的 UI

文件消息使用 RCFileMessageCell 展示在消息列表中。如果需要调整内置消息样式，建议自定义消息 Cell，并将该自定义 Cell 提供给 SDK。IMKit 中所有消息模板都继承自 RCMessageCell，自定义消息 Cell 也需要继承 RCMessageCell。详见修改消息的展示样式。

您也可以直接替换 RongCloud.bundle 中文件消息展示模板中引用的样式资源、字符串资源和图标资源。详见 IMKit 源码 RCFileMessageCell.m 中引用的资源。

自定义发送文件页 UI

- **右上角按钮未选择文件时颜色**：可通过 color.plist 中 fileSelect 字典里的相关字段来更改。
- **左上、右上角按钮**：设置 SDK 全局导航按钮颜色。

```
RCKitConfigCenter.ui.globalNavigationBarTintColor = [UIColor whiteColor];
```

- **右上角按钮选中文件时的颜色**：设置 SDK 全局导航按钮颜色。

```
RCKitConfigCenter.ui.globalNavigationBarTintColor = [UIColor whiteColor];
```

隐藏文件消息入口

如需动态修改，可在会话页面显示前，通过 RCConversationViewController 的 chatSessionInputBarControl.pluginBoardView 删除指定扩展项目。

详见输入区域。

位置消息　　位置消息

更新时间:2024-08-30

IMKit 提供了位置插件，实现了发送位置消息、位置缩略图功能。本文还描述了如何自行实现了应用内位置共享。

- SDK 默认发送的消息包含位置消息内容对象 [RCLocationMessage]（类型标识：`RC:LBSMsg`）。
- 实时位置共享也是基于消息实现的。SDK 默认使用类型标识为 `RC:RL`、`RC:RLStart`、`RC:RLJoin`、`RC:RLQuit` 的消息。

> 提示
>
> IMKit 默认会话页面未启用位置功能。如需要使用位置功能，可集成 IMKit 位置插件并配置您自己的高德地图 SDK 帐号。

局限

- IMKit 的位置插件基于 MapKit。如需使用其他地图服务，您可以自定义插件，自行构造位置消息并发送。添加自定义插件的方法详见输入区域。
- 位置插件默认仅支持点击发送位置。

用法

IMKit 从 5.2.3 及之后开始支持 `LocationKit` 插件。如果从低于 5.2.3 的 IMKit 版本升级，请参见下文升级旧版位置插件。

集成位置插件

请根据 IMKit 的导入方式，选择集成 `LocationKit` 插件的方式。

- 使用 CocoaPods

  Framework（要求 SDK >= 5.2.3）

```
pod 'RongCloudIM/LocationKit',' x.y.z' #位置插件
```

源码（要求 IMKit 同为源码，要求 SDK >= 5.2.3）

```
pod 'RongCloudOpenSource/LocationKit', 'x.y.z' #位置插件
```

- 手动集成。如果项目中 IMKit 源码为手动导入，请通过融云官网 SDK 下载页面 下载 LocationKit 插件。由于 IMKit 的 LocationKit 插件依赖 IMLib SDK 提供的 Location 库，您还需要额外添加 Location 库。您可以选择通过 CocoaPods 导入 Location Framework，或使用下载包中的 RongLocation.xcframework 文件。

```
pod 'RongCloudIM/Location',' x.y.z' # IMLib SDK 位置基础库
```

> 提示
>
> x.y.z 代表具体版本，请通过融云官网 SDK 下载页面 或 CocoaPods 仓库等方式查询最新版本。

发送位置消息

位置插件集成完毕后，在扩展面板里会自动生成位置消息入口。用户点击输入栏右侧 + 号按钮可展开扩展面板，点击位置图标，即可发送位置消息。

集成位置插件后，默认仅支持点击发送位置。

使用实时位置共享

IMKit SDK 没有提供实时位置共享功能插件。您可以参考融云示例应用 SealTalk 中的代码，在您应用中实现实时位置共享。

> 提示
>
> 图中参与位置实时共享人员的用户头像、昵称需要 App 提供给 IMKit，具体方法参考 设置用户信息。

下面以 SealTalk 示例应用开源代码为例，说明在单聊会话中实现位置共享的步骤。操作完成后，就可以在单聊会话中使用位置实时共享功能。您也可以参考在 SealTalk 的 RCDChatViewController.m 。

1. 下载 SealTalk 项目源码，将源码中的 Sections/Chat/RealTimeLocation 导入工程。

```
https://github.com/rongcloud/sealtalk-ios
```

2. 在会话页面中导入下面头文件。

```
#import <objc/runtime.h>
#import "RealTimeLocationEndCell.h"
#import "RealTimeLocationStartCell.h"
#import "RealTimeLocationStatusView.h"
#import "RealTimeLocationViewController.h"
#import "RealTimeLocationDefine.h"
static const char *kRealTimeLocationKey = "kRealTimeLocationKey";
static const char *kRealTimeLocationStatusViewKey = "kRealTimeLocationStatusViewKey";
```

3. 设置相关属性。

```
@interface RCDChatViewController () < RCRealTimeLocationObserver,
RealTimeLocationStatusViewDelegate>

@property(nonatomic, weak) id<RCRealTimeLocationProxy> realTimeLocation;

@property(nonatomic, strong) RealTimeLocationStatusView *realTimeLocationStatusView;

@end
```

4. 获取位置共享服务并注册消息。

```
- (void)viewDidLoad{
[super viewDidLoad];
[self registerRealTimeLocationCell];
[self getRealTimeLocationProxy];
}
```

```
- (void)registerRealTimeLocationCell {
[self initRealTimeLocationStatusView];
[self registerClass:[RealTimeLocationStartCell class] forMessageClass:[RCRealTimeLocationStartMessage class]];
[self registerClass:[RealTimeLocationEndCell class] forMessageClass:[RCRealTimeLocationEndMessage class]];
}

- (void)getRealTimeLocationProxy {
__weak typeof(self) weakSelf = self;
[[RCRealTimeLocationManager sharedManager] getRealTimeLocationProxy:self.conversationType
targetId:self.targetId
success:^(id<RCRealTimeLocationProxy> realTimeLocation){
weakSelf.realTimeLocation = realTimeLocation;
[weakSelf.realTimeLocation addRealTimeLocationObserver:weakSelf];
[weakSelf updateRealTimeLocationStatus];
} error:^(RCRealTimeLocationErrorCode status) {
NSLog(@"get location share failure with code %d",(int)status);
}];
}

- (void)initRealTimeLocationStatusView {
self.realTimeLocationStatusView =
[[RealTimeLocationStatusView alloc] initWithFrame:CGRectMake(0, 62, self.view.frame.size.width, 0)];
self.realTimeLocationStatusView.delegate = self;
[self.view addSubview:self.realTimeLocationStatusView];
}
```

5. 点击位置时弹出位置实时共享选项。

```
//RCDChatViewController Class
- (void)pluginBoardView:(RCPluginBoardView *)pluginBoardView clickedItemWithTag:(NSInteger)tag {
switch (tag) {
case PLUGIN_BOARD_ITEM_LOCATION_TAG: {
if (self.realTimeLocation) {
[RCActionSheetView showActionSheetView:nil cellArray:@[RTLLocalizedString(@"send_location"),
RTLLocalizedString(@"location_share")]
cancelTitle:RTLLocalizedString(@"cancel")
selectedBlock:^(NSInteger index) {
if (index == 0) {
[super pluginBoardView:self.chatSessionInputBarControl.pluginBoardView
clickedItemWithTag:PLUGIN_BOARD_ITEM_LOCATION_TAG];
}else{
[self showRealTimeLocationViewController];
}
} cancelBlock:^{

}];
} else {
[super pluginBoardView:pluginBoardView clickedItemWithTag:tag];
}
}break;
default:
[super pluginBoardView:pluginBoardView clickedItemWithTag:tag];
break;
}
}
```

6. 实时位置共享监听代理方法。

```
- (void)onRealTimeLocationStatusChange:(RCRealTimeLocationStatus)status {
__weak typeof(self) weakSelf = self;
dispatch_async(dispatch_get_main_queue(), ^{
[weakSelf updateRealTimeLocationStatus];
});
}

- (void)onReceiveLocation:(CLLocation *)location type:(RCRealTimeLocationType)type fromUserId:(NSString *)userId {
__weak typeof(self) weakSelf = self;
dispatch_async(dispatch_get_main_queue(), ^{
[weakSelf updateRealTimeLocationStatus];
});
}


- (void)onParticipantsJoin:(NSString *)userId {
__weak typeof(self) weakSelf = self;
if ([userId isEqualToString:[RCCoreClient sharedCoreClient].currentUserInfo.userId]) {
[self notifyParticipantChange:RTLLocalizedString(@"you_join_location_share")];
} else {
[[RCIM sharedRCIM]
.userInfoDataSource
getUserInfoWithUserId:userId
completion:^(RCUserInfo *userInfo) {
if (userInfo.name.length) {
[weakSelf notifyParticipantChange:[NSString
stringWithFormat:RTLLocalizedString(@"someone_join_share_location"),userInfo.name]];
} else {
[weakSelf notifyParticipantChange:[NSString stringWithFormat:RTLLocalizedString(@"user_join_share_location"),userId]];
}
}];
}
}

- (void)onParticipantsQuit:(NSString *)userId {
__weak typeof(self) weakSelf = self;
if ([userId isEqualToString:[RCCoreClient sharedCoreClient].currentUserInfo.userId]) {
[self notifyParticipantChange:RTLLocalizedString(@"you_quit_location_share")];
} else {
[[RCIM sharedRCIM]
.userInfoDataSource
getUserInfoWithUserId:userId
completion:^(RCUserInfo *userInfo) {
if (userInfo.name.length) {
[weakSelf
notifyParticipantChange:[NSString stringWithFormat:RTLLocalizedString(@"someone_quit_location_share"),userInfo.name]];
} else {
[weakSelf
notifyParticipantChange:[NSString stringWithFormat:RTLLocalizedString(@"user_quit_location_share"),userId]];
}
}];
}
}

- (void)onRealTimeLocationStartFailed:(long)messageId {
dispatch_async(dispatch_get_main_queue(), ^{
for (int i = 0; i < self.conversationDataRepository.count; i++) {
RCMessageModel *model = [self.conversationDataRepository objectAtIndex:i];
if (model.messageId == messageId) {
model.sentStatus = SentStatus_FAILED;
}
}
NSArray *visibleItem = [self.conversationMessageCollectionView indexPathsForVisibleItems];
for (int i = 0; i < visibleItem.count; i++) {
NSIndexPath *indexPath = visibleItem[i];
RCMessageModel *model = [self.conversationDataRepository objectAtIndex:indexPath.row];
if (model.messageId == messageId) {
[self.conversationMessageCollectionView reloadItemsAtIndexPaths:@[ indexPath ]];
}
}
});
}

- (void)notifyParticipantChange:(NSString *)text {
__weak typeof(self) weakSelf = self;
dispatch_async(dispatch_get_main_queue(), ^{
```

```
[weakSelf.realTimeLocationStatusView updateText:text];
[weakSelf performSelector:@selector(updateRealTimeLocationStatus) withObject:nil afterDelay:0.5];
});
}

- (void)onFailUpdateLocation:(NSString *)description {

}


#pragma mark - 实时位置共享状态 view 代理方法
- (void)onJoin {
[self showRealTimeLocationViewController];
}

- (RCRealTimeLocationStatus)getStatus {
return [self.realTimeLocation getStatus];
}

- (void)onShowRealTimeLocationView {
[self showRealTimeLocationViewController];
}

- (void)setRealTimeLocation:(id<RCRealTimeLocationProxy>)realTimeLocation{
objc_setAssociatedObject(self, kRealTimeLocationKey, realTimeLocation, OBJC_ASSOCIATION_ASSIGN);
}

- (id<RCRealTimeLocationProxy>)realTimeLocation {
return objc_getAssociatedObject(self, kRealTimeLocationKey);
}

- (void)setRealTimeLocationStatusView:(RealTimeLocationStatusView *)realTimeLocationStatusView {
objc_setAssociatedObject(self, kRealTimeLocationStatusViewKey,
realTimeLocationStatusView,OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (RealTimeLocationStatusView *)realTimeLocationStatusView {
return objc_getAssociatedObject(self, kRealTimeLocationStatusViewKey);
}

//弹出实时位置共享页面
- (void)showRealTimeLocationViewController {
RealTimeLocationViewController *lsvc = [[RealTimeLocationViewController alloc] init];
lsvc.realTimeLocationProxy = self.realTimeLocation;
if ([self.realTimeLocation getStatus] == RC_REAL_TIME_LOCATION_STATUS_INCOMING) {
[self.realTimeLocation joinRealTimeLocation];
} else if ([self.realTimeLocation getStatus] == RC_REAL_TIME_LOCATION_STATUS_IDLE) {
[self.realTimeLocation startRealTimeLocation];
}
lsvc.modalPresentationStyle = UIModalPresentationFullScreen;
[self.navigationController presentViewController:lsvc
animated:YES
completion:^{

}];
}


//更新实时位置共享状态
- (void)updateRealTimeLocationStatus {
if (self.realTimeLocation) {
[self.realTimeLocationStatusView updateRealTimeLocationStatus];
__weak typeof(self) weakSelf = self;
NSArray *participants = nil;
switch ([self.realTimeLocation getStatus]) {
case RC_REAL_TIME_LOCATION_STATUS_OUTGOING:
[self.realTimeLocationStatusView updateText:RTLLocalizedString(@"you_location_sharing")];
break;
case RC_REAL_TIME_LOCATION_STATUS_CONNECTED:
case RC_REAL_TIME_LOCATION_STATUS_INCOMING:
participants = [self.realTimeLocation getParticipants];
if (participants.count == 1) {
NSString *userId = participants[0];
[weakSelf.realTimeLocationStatusView
updateText:[NSString stringWithFormat:RTLLocalizedString(@"user_location_sharing"), userId]];
[[RCIM sharedRCIM]
.userInfoDataSource
getUserInfoWithUserId:userId
```

```
completion:^(RCUserInfo *userInfo) {
if (userInfo.name.length) {
dispatch_async(dispatch_get_main_queue(), ^{
[weakSelf.realTimeLocationStatusView updateText:[NSString
stringWithFormat:RTLLocalizedString(@"someone_location_sharing"),userInfo.name]];
});
}
}];
} else {
if (participants.count < 1)
[self.realTimeLocationStatusView removeFromSuperview];
else
[self.realTimeLocationStatusView
updateText:[NSString stringWithFormat:@"%d人正在共享地理位置", (int)participants.count]];
}
break;
default:
break;
}
}
}
```

7. （可选）在 IMLibCore 的 RCConfig.plist 文件中，添加如下配置，开启群组会话的实时位置共享功能。群聊最多支持5人同时位置共享。

```
<key>RealTimeLocationShare</key>
<dict>
<key>SupportConversationTypes</key>
<array>
<string>1</string>
<string>3</string>
</array>
</dict>
```

升级旧版位置插件

> 提示
>
> IMKit 5.2.3 之前版本升级到 5.2.3 时，旧版位置插件即失效。

- 5.2.3 之前，IMKit 内置了位置插件。
- 5.2.3 及之后，IMKit 不再内置位置插件。不需要位置功能时，可不必集成位置插件（LocationKit），避免 App 上架因获取位置权限需要审核的问题。
- IMKit 5.2.3 之前版本升级到 5.2.3 时，原发送位置功能即失效。

定制化

调整位置缩略图压缩比例

从客户端发送位置消息时，SDK 会自动生成预览地图图片，自动上传到文件服务器（默认上传到七牛云存储），并将云存储服务返回的图片的远程地址放入消息体对应字段后进行发送。SDK 会以宽度不超过 408 像素、高度不超过 240 进行压缩。

一般情况下不建议修改 SDK 默认压缩配置。如需调整 SDK 压缩质量，详见知识库文档如何修改 SDK 默认的图片与视频压缩配置 。

自定义位置消息的 UI

位置消息（非实时位置共享消息）使用 RCLocationMessageCell 展示在消息列表中。如果需要调整内置消息样式，建议自定义消息 Cell，并将该自定义 Cell 提供给 SDK。IMKit 中所有消息模板都继承自 RCMessageCell，自定义消息 Cell 也需要继承 RCMessageCell。详见修改消息的展示样式。

您也可以直接替换 RongCloud.bundle 中文件消息展示模板中引用的样式资源、字符串资源和图标资源。详见 IMKit 源码 RCLocationMessageCell.m 中引用的资源。

自定义位置选取页 UI

您可以通过设置 IMKit 全局导航按钮颜色来调整左上、右上角按钮：

```
RCKitConfigCenter.ui.globalNavigationBarTintColor = [UIColor whiteColor];
```

https://doc.rongcloud.cn/apidoc/locationlib-ios/latest/zh_CN/documentation/ronglocation/rclocationmessage?language=objc

名片消息　　名片消息

更新时间:2024-08-30

用户可以通过 IMKit 名片插件发送个人名片。消息将出现在会话页面的消息列表组件中。插件默认发送的消息中包含名片消息内容对象 RCContactCardMessage（类型标识：RC:CardMsg）。

> 提示
>
> IMKit 默认会话页面未启用名片消息功能。如需要使用该功能，可集成 IMKit 名片插件，并提供需要展示和发送的数据。

用法

IMKit 支持名片插件。导入名片插件模块后，扩展面板中会自动出现该功能入口。点击扩展面板中的个人名片会跳转至联系人列表页面（App 需要提供联系人列表）。

- IMKit 5.1.8 之前，名片插件（ContactCard）仅支持以源码方式导入。
- IMKit 5.1.8 及之后，名片插件（ContactCard）支持以源码方式或以 Framework 方式导入。

请根据应用程序集成 IMKit 的方式，选择使用 Framework 和源码导入小视频插件。请务必不要混用 Framework 和源码集成方式。

- 导入名片插件 Framework（要求 SDK >= 5.1.8）

```
pod 'RongCloudIM/ContactCard','~> x.y.z' #名片 framework
```

- 导入名片插件源码

```
pod 'RongCloudOpenSource/ContactCard','~> x.y.z' #名片源码
```

> 提示
>
> **x.y.z 代表具体版本，请通过融云官网 SDK 下载页面 或 CocoaPods 仓库等方式查询最新版本。**

在导入名片插件模块后，您还需要为名片插件提供联系人列表。由于融云不提供用户信息托管维护服务，您需要通过 RCCCContactsDataSource 协议为 SDK 提供联系人列表。详见 RCContactCardKit.h 。

具体实现可参考以下步骤：

1. 设置代理委托。

```
[RCContactCardKit shareInstance].contactsDataSource = self;
```

2. 实现代理方法。

```
- (void)getAllContacts:(void (^)(NSArray<RCCCUserInfo *> *))resultBlock{
//开发者调自己的服务器接口，获取到联系人信息回调给名片模块
[RCDUserInfoManager getAllFriendsFromServer:^(NSArray<RCDFriendInfo *> *_Nonnull userList) {
NSMutableArray *contacts = [NSMutableArray new];
for (RCDFriendInfo *friend in userList) {
RCCCUserInfo *contact = [RCCCUserInfo new];
contact.userId = friend.userId;
contact.name = friend.name;
if (friend.portraitUri.length <= 0) {
friend.portraitUri = [RCDUtilities defaultUserPortrait:friend];
}
contact.portraitUri = friend.portraitUri;
contact.displayName = friend.displayName;
[contacts addObject:contact];
}
resultBlock(contacts);
}];
}
```

3. 重写 RCConversationViewController 的消息点击方法，来实现名片消息点击事件。

```
- (void)didTapMessageCell:(RCMessageModel *)model {
if ([model.content isKindOfClass:[RCContactCardMessage class]]) {
RCContactCardMessage *cardMSg = (RCContactCardMessage *)model.content;
// 此处执行 APP 业务逻辑
return;
}

[super didTapMessageCell:model];
}
```

发送名片消息

用户点击输入栏右侧 + 号按钮可展开扩展面板，点击个人名片图标，即可发送名片。

定制化

名片插件内部定义了名片消息内容类 RCContactCardMessage.h ，名片消息展示模板 RCContactCardMessageCell.h 。

自定义名片消息的 UI

文件消息使用 RCContactCardMessageCell 展示在消息列表中。如果需要调整内置消息样式，建议自定义消息 Cell，并将该自定义 Cell 提供给 SDK。IMKit 中所有消息模板都继承自 RCMessageCell，自定义消息 Cell 也需要继承 RCMessageCell。详见修改消息的展示样式。

您也可以直接替换 RongCloud.bundle 中文件消息展示模板中引用的样式资源、字符串资源和图标资源。详见 IMKit 源码 RCContactCardMessageCell.m 中引用的资源。

自定义个人名片页 UI

您可以通过 IMKit 全局导航按钮颜色修改左上角按钮：

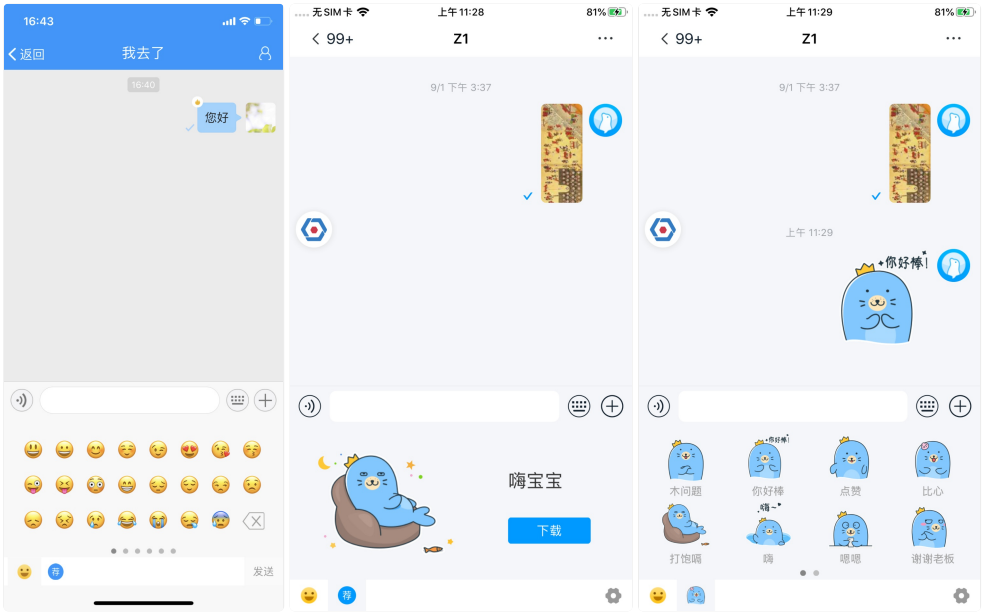
```
RCKitConfigCenter.ui.globalNavigationBarTintColor = [UIColor whiteColor];
```

Emoji 与贴纸表情 Emoji 与贴纸表情

更新时间:2024-08-30

用户可以 IMKit 输入区域发送 Emoji 表情、贴纸表情。点击输入栏的表情（☺）按钮，即可展开表情面板，支持发送 emoji 表情、贴纸表情。表情消息将出现在会话页面的消息列表组件中。

> 提示
>
> IMKit 输入区域的表情面板中默认仅包含 emoji。图中的贴纸表情需要集成 `rcsticker` 库。支持添加自定义表情。



Emoji 符号表情

IMKit 输入区域的表情面板中默认显示内置 Emoji 表情，由 SDK 默认访问 `RongIMKit` 资源文件中的 `Emoji.plist` 生成。用户点击后可在文本输入区域插入 Emoji 符号表情（原生字符）。

禁用表情面板中的内置 Emoji 表情

> 提示
>
> SDK 从 5.2.3 开始支持禁用内置 Emoji 表情。

在会话页面显示前，通过 RCConversationViewController 的 `disableSystemEmoji` 属性设置是否包含内置的 Emoji 表情标签页。禁用后，会话页面的表情面板中不会显示 Emoji 标签页。

YES 表示禁用。

```
chatVC.disableSystemEmoji = YES;
[self.navigationController pushViewController:chatVC animated:YES];
```

贴纸表情

IMKit 表情面板中可支持贴纸表情，您可以集成融云表情贴纸库 `RongSticker`，也可以添加自定义贴纸表情。

集成融云贴纸表情

IMKit 可集成融云表情贴纸库 RongSticker，其中包含一套“嗨豹”贴纸表情。

> 提示
>
> 融云贴纸（RongSticker）支持以源码或 Framework 方式导入。请务必选用同一种方式导入 IMKit 及其插件，不要混用 Framework 和源码。

以集成 RongSticker 源码为例：

```
# 插件版本需要与主 SDK 版本保持一致。
pod 'RongCloudOpenSource/RongSticker','~> x.y.z'
```

x.y.z 代表具体版本，请通过融云官网 SDK 下载页面 或 CocoaPods 仓库等方式查询最新版本。

集成成功后，RCStickerModule 模块会向表情面板中添加融云贴纸标签页。首次点击后请根据提示下载贴纸。下载成功后会在面板里展示所有贴纸。

点击融云贴纸表情后，RongSticker 库会直接向会话中发送消息。如需了解更多细节，可参考 IMKit 源码中的 RCStickerCollectionView.m 。

```
- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)indexPath {
RCStickerSingle *model = self.stickers[indexPath.row];
RCStickerMessage *stickerMsg = [RCStickerMessage messageWithPackageId:self.packageId
stickerId:model.stickerId
digest:model.digest
width:model.width
height:model.height];
[[RCIM sharedRCIM] sendMessage:[RCStickerModule sharedModule].conversationType
targetId:[RCStickerModule sharedModule].currentTargetId
content:stickerMsg
pushContent:nil
pushData:nil
success:nil
error:nil];
}
```

自定义贴纸表情

自定义表情页需遵守 RCEmoticonTabSource 协议。IMKit 的 RongSticker 库中的 RCStickerModule 实现了 RongIMKitExtensionModule 协议，通过实现其中的 getEmoticonTabList 方法，向 IMKit 提供扩展面板中添加了自定义表情页。

要添加自定义贴纸表情，有两种方式：

- 使用会话页面的 `chatSessionInputBarControl.emojiBoardView` 属性直接添加本地图片或 GIF。这种方式添加的表情标签右下角不含发送按钮，需要您自行实现点击发送消息。
- 自行实现一个完整的表情页模块。该模块需要遵循 `RongIMKitExtensionModule` 协议，通过 `getEmoticonTabList` 将表情页面加入表情面板，这种方式自行实现从远端下载表情数据。建议您参考 IMKit 源码的 RCStickerModule.m 和 RCStickerDataManager.m 。

以下步骤介绍了如何通过使用会话页面的 chatSessionInputBarControl.emojiBoardView 属性直接添加本地图片或 GIF。

1. 实现 RCEmoticonTabSource 协议的 loadEmoticonView 返回表情标签页的 View。返回的 view 大小必须等于 contentViewSize （宽度 = 屏幕宽度，高度 = 186）

```
- (UIView *)loadEmoticonView:(NSString *)identify index:(int)index;
```

参数	类型	说明
identify	NSString	表情面板 tab 的标识符
index	int	表情面板 tab 的第几页

2. 在会话页面通过 emojiBoardView 的 addEmojiTab 方法添加表情包（不支持动态移除）。您需要自行处理单个表情上的手势和事件。如果表情是图片，需自行构建 RCImageMessage 消息内容。如果表情是动图，需要自行构建 RCGifMessage 消息内容。发送方法详见发送消息。

```
UIImage *icon = [RCKitUtility imageNamed:@"imageName"
ofBundle:@"RongCloud.bundle"];
RCDCustomerEmoticonTab *emoticonTab1 = [RCDCustomerEmoticonTab new];
emoticonTab1.identify = @"标识符";
emoticonTab1.image = icon;
emoticonTab1.pageCount = 页数;
[self.emojiBoardView addEmojiTab:emoticonTab1];
```

自定义表情标签页面支持分页展示。因 SDK 暂不支持刷新 pageCount，请一次性设置准确的分页数量。

隐藏表情面板入口

> 提示
>
> IMKit SDK 从 5.3.2 版本开始提供该功能。

在 App 不需要提供表情输入功能时，可隐藏输入栏中的表情按钮，隐藏后用户无法展开表情面板。详见输入区域。

@ 消息 @ 消息

更新时间:2024-08-30

提及（@）是群聊会话中常见功能，允许用户在会话中提及指定用户，或全部群成员，以增强消息的提示作用。使用 @ 功能后，消息内容中会额外携带 RCMentionedInfo 对象。IMKit 默认启用了 @ 功能。

> 提示
>
> IMKit 未实现 @所有人功能，图中「@所有人」仅作为设计参考。选择联系人页面的数据需要由应用程序提供，否则展示空列表。

局限

- 仅支持群聊会话。
- IMKit 默认仅实现了在发送文本消息、引用消息时使用 @ 功能。
- IMKit 未实现 @ 所有人功能。
- @ 消息可以被转发，但转发的只是纯文本，不再具备 @ 功能。

用法

> 提示
>
> 在使用 @ 功能前，请先实现 `RCIMGroupMemberDataSource`，并给 IMKit 设置群成员列表信息代理。详见用户信息。

IMKit 默认在配置中启用了 @ 功能，用法如下：

- 在会话页面长按用户头像可触发消息编辑，提及（@）该用户。
- 在会话页面输入 @ 符号之后，IMKit 会跳转到成员列表选择页面。如果应用程序设置群成员列表信息代理，该页面会显示一个空列表。设置群成员列表信息代理后，IMKit 会通过 `RCIMGroupMemberDataSource` 协议的 `getAllMembersOfGroup` 方法取得群成员数据，并展示在该列表页

面中。

定制化

自定义选择成员界面

如果想更换选人界面，可以重写下面方法，弹出自定义的选人界面，选人结束之后，调用 selectedBlock 传入选中的用户列表即可。

```
- (void)showChooseUserViewController:(void (^)(RCUserInfo *selectedUserInfo))selectedBlock
cancel:(void (^)(void))cancelBlock;
```

您可以参考融云 SealTalk 应用的实现方式。详见RCDChatViewController.m 的 showChooseUserViewController 方法。

实现 @ 所有人

IMKit 未实现 @ 所有人功能的页面逻辑，您可以自行实现。您需要构建 type 属性为RC_Mentioned_All 的 RCMentionedInfo 对象，写入消息内容中。构建 RCMessage 后，使用 sendMessage（普通消息）或 sendMediaMessage 方法发送到会话中。

```
RCTextMessage *txtMsg = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMentionedInfo *mentionedInfo = [[RCMentionedInfo alloc] initWithMentionedType:RC_Mentioned_All userIdList:nil
mentionedContent:nil];
txtMsg.mentionedInfo = mentionedInfo;

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:txtMsg];

[[RCIM sharedRCIM] sendMessage:message
pushContent:nil
pushData:nil
successBlock:^(RCMessage *successMessage) {
//成功
} errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}];
```

您参考可以融云 SealTalk 应用的实现方式。详见RCDChooseUserController.m 。

关闭 @ 功能

IMKit 的 @ 功能默认开启。您可以通过修改全局配置关闭 @ 功能。

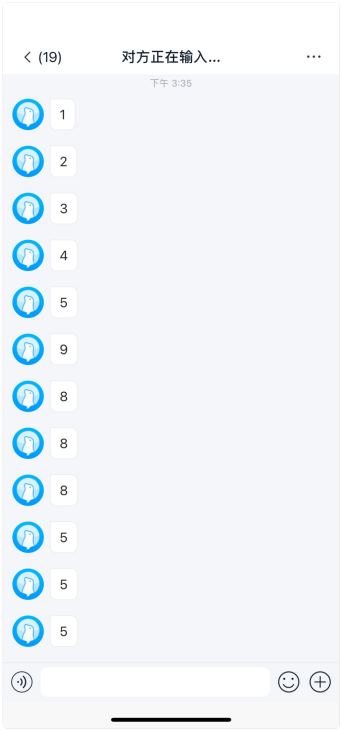
```
RCKitConfigCenter.message.enableMessageMentioned = NO;
```

输入状态　　输入状态

更新时间:2024-08-30

输入状态可让用户直观地了解其他用户是否正在键入消息。在对方用户键入内容时，标题栏会一直显示「对方正在输入」，直到用户发送消息或完全删除文本。如果用户停止打字超过 6 秒，该提示也会消失。SDK 在输入框中有内容变化时，默认向对端用户发送一条正在输入的状态消息，包含消息内容对象 RCTypingStatusMessage（类型标识：RC:TypSts）。

> 提示
>
> IMKit 的 RCConversationViewController 使用了系统的导航栏，开启输入状态功能后，在单聊会话中对方正在输入时，标题栏会被 SDK 修改为对方的输入状态。



局限

- 只支持单聊会话。
- 因无法确定用户的输入操作，该功能可能会产生大量状态消息，为防止消息发送频繁，默认在 6 秒钟内的多次状态变化，只产生一条输入状态消息。
- 该功能可能会导致大量状态消息，如不需要此功能建议关闭。

用法

IMKit 输入状态功能默认可用，无需额外处理。

定制化

设置发送输入状态消息的默认时间间隔

IMKit 不提供调整默认间隔时间的方法。您可以需要使用 IMLib 的配置方法，单位为秒。

```
[[RCCoreClient sharedCoreClient].typingUpdateSeconds = 6];
```

关闭输入状态功能

IMKit 默认开启输入状态功能。您可以通过 IMKit 全局配置关闭输入状态功能。

```
RCKitConfigCenter.message.enableTypingStatus = NO;
```

已读回执 已读回执

更新时间:2024-08-30

IMKit 提供了单聊、群聊的已读回执功能。App 用户通过已读回执获知对方是否阅读该消息。

在 IMKit 内置页面中已默认实现并启用已读回执功能。在单聊会话中，消息默认更新已读状态。在群聊会话中，消息发送者需要在页面上主动请求获取已读状态。

阅读回执开关

IMKit 中回执功能默认开启，在单聊和群聊中默认会展示消息回执。

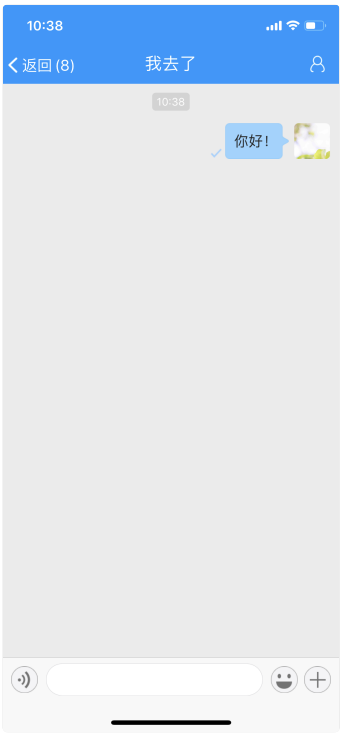
您可以修改支持的会话类型。下面的代码将阅读回执功能配置为仅在单聊中支持，群聊中将不启用阅读回执功能。

```
RCKitConfigCenter.message.enabledReadReceiptConversationTypeList = @[ @(ConversationType_PRIVATE) ];
```

单聊阅读回执

在单聊会话中，发送方会实时收到消息的已读状态更新。在 IMKit 内置页面中，单聊已读状态显示在两处：

- **单聊会话页面（消息列表页面）**：在发送方的单聊会话页面，消息的左下角会显示对号，表示对方已读。
- **会话列表页面**：会话列表的每条会话会显示会话中的最后一条消息的预览。如果单聊会话最后一条消息被对方阅读，发送方的会话列表页中对应会话条目的右下角也会显示对号，表示对方已读。



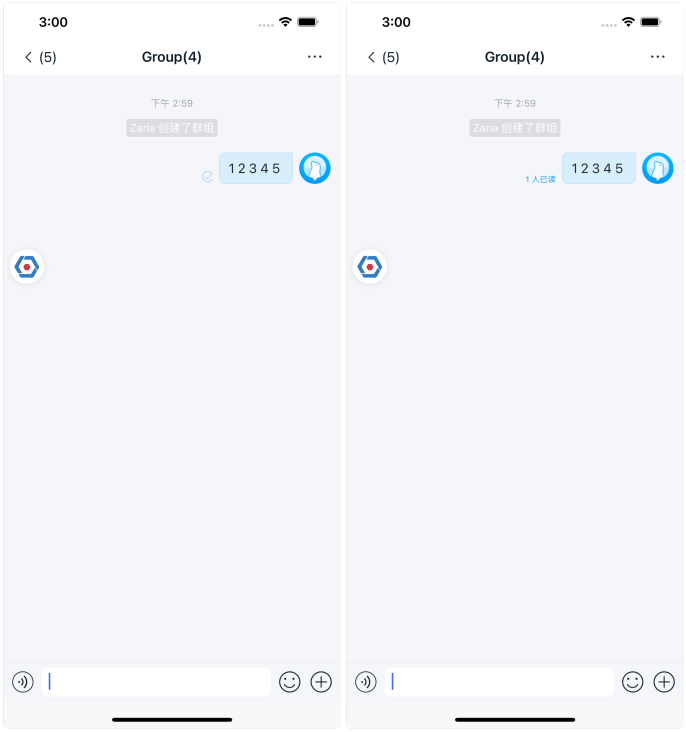
SDK 会话页面中文本消息已读的 UI 默认是一个“对勾”图标，如果展示为文本“已读”或“未读”，可以在消息 Cell 展示前修改。详见会话页面中的定制化。

群聊阅读回执

提示

> IMKit 的群聊已读回执功能仅支持文本消息类型。

在群聊会话中发送消息后 120 秒之内，发送者可以在会话页面上主动请求获取已读人数数据。在会话页面上，消息的左下角会显示对号按钮（请求阅读回执的按钮）。点击按钮后，IMKit 才会请求已读回执。IMKit 在收到已读回执结果后会刷新页面，显示为 “n 人已读”。

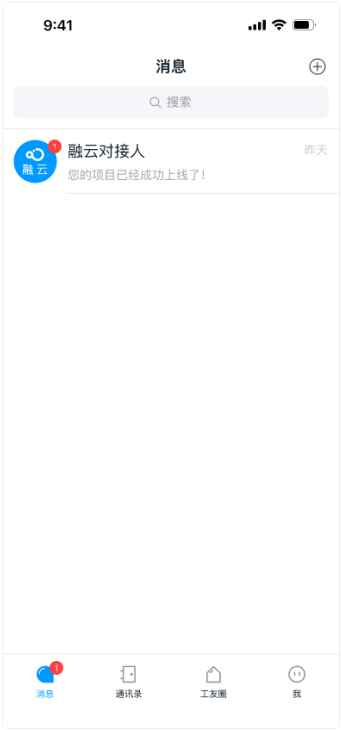


消息未读数　　消息未读数

更新时间:2024-08-30

未读消息计数是 IMKit 默认提供的一项功能，可告知用户每个会话中未读消息的数量。未读消息计数显示在会话列表类 RCConversationListViewController 的 `RCConversationCell` 中。每个会话的未读消息数显示在会话图标右上角。如果未读消息数超过 100 条，则会显示为 99+ 。

> 提示
>
> 为了使用未读消息计数功能，您必须首先构建会话列表页面。IMKit 默认未实现在 Tab Bar 中显示未读消息数。如有需要您可以参考 SealTalk 项目（GitHub · Gitee）中的实现。



用法

IMKit SDK 默认已经实现了一整套会话未读消息数的获取和展示逻辑，使用默认会话列表和会话页面时，不需要额外调用会话相关 API。

IMKit 会在用户进入单聊、群聊、系统会话页面时将会话未读数清零。在用户多端登录时，IMKit 会在设备间同步会话的阅读状态，您也可以按业务需求选择关闭该功能，详见下文多端同步阅读状态。

定制化

如果 IMKit 已有实现无法满足您的需求，可以使用 IMKit 或 IMLib SDK 中相关 API。

获取或清除会话未读数

IMKit 未直接提供获取、清除会话未读数的 API。如果您有自定义需求，可以调用 IMLib SDK 相关方法。例如：

- 获取所有会话未读数
- 按会话类型获取未读数
- 清除单个会话未读数

具体的核心类、API 与 使用方法，详见 IMLib 文档 处理会话未读消息数 。

> 提示
>
> IMLib 中的方法并不提供页面刷新能力，您需要根据业务需求自定义通知机制进行页面刷新。

设置为仅显示红点

如果希望仅在会话列表中的 RCConversationCell 上显示红点，而不显示具体数字，您可以重写 RCConversationListViewController 的willDisplayConversationTableCell:atIndexPath: 方法，设置 isShowNotificationNumber 属性值为 NO。

```
- (void)willDisplayConversationTableCell:(RCConversationBaseCell *)cell atIndexPath:(NSIndexPath *)indexPath {
//会话有新消息通知的时候显示数字提醒，设置为NO,不显示数字只显示红点
((RCConversationCell *)cell).isShowNotificationNumber = NO;
}
```

自定义未读消息角标 UI

会话列表中的 RCConversationCell 上的未读消息角标 UI 由 RCConversationListViewController 的 bubbleTipView 属性控制。您可以重写 RCConversationListViewController 的willDisplayConversationTableCell:atIndexPath: 方法修改。

下表仅列出了 bubbleTipView 中的部分属性。完整属性列表请参见 API 文档中的 RCMessageBubbleTipView 。

项目	属性	类型	默认值
角标位置	bubbleTipAlignment	RCMessageBubbleTipViewAlignment	RC_MESSAGE_BUBBLE_TIP_VIEW_ALIGNMENT_TOP_RIGHT
角标文本颜色	bubbleTipTextColor	UIColor	whiteColor
角标背景颜色	bubbleTipBackgroundColor	UIColor	redColor

修改已读未读状态图标为文本

SDK 会话页面中文本消息已读的 UI 默认是一个“对勾”图标，如果希望修改为「已读」或「未读」，可以在 Cell 显示的时候，将 SDK 默认的图标移除。

修改步骤详见会话页面 文档中的自定义消息 Cell 显示 。

多端同步阅读状态

> 提示
>
> 如果 SDK 版本 ≦ 5.6.2，不支持多端同步 系统会话的已读、未读状态。

在即时通讯业务中，同一用户账号可能在多个设备上登录。仅在开通多设备消息同步服务后，融云会在多个设备之间同步消息数据，但设备上的会话中消息的已读/未读状态仅存储在本地。

IMKit SDK 已默认实现了会话多端阅读状态同步功能，在一端发起同步后，其他端可接收通知，按要求同步阅读状态。您可以按业务需求决定是否使用该功能，该功能默认开启。

```
RCKitConfigCenter.message.enableSyncReadStatus = YES;
```

未读消息气泡提醒

IMKit 支持在会话页面（RCConversationViewController ）中显示未读消息气泡提醒。

是否显示未读消息数提醒

如果会话的未读消息数已超过 10，可在进入会话页面后在右下角显示提醒气泡。用户点击提醒气泡后，页面会跳转到最开始的未读消息。

该功能默认关闭。如需开启提醒，请在进入会话页面前设置。

```
@property (nonatomic, assign) BOOL enableUnreadMessageIcon;
```

是否显示未读 @ 消息数提醒

当一个会话收到大量消息（超过一个屏幕能显示的内容），且收到的消息中有 @ 消息时，进入会话页面后，会话页面右上角会提示未读 @ 消息数。用户点击该提醒按钮，会跳转到最早的未读 @ 消息处，同时未读 @ 消息数量减 1。再次点击，未读 @ 消息数量会根据当前屏幕内看到的个数相应减少。

该功能默认开启。如果需要关闭提醒，请在进入会话页面前设置。

```
@property (nonatomic, assign) BOOL enableUnreadMentionedIcon;
```

是否显示新消息提醒

如果用户在查看会话页面中的历史消息，且当前视图未显示会话最新消息，此时如果收到新消息，会话页面右下角可显示提醒，例如「15 条新消息」。用户点击提醒按钮，会滚动到会话最新消息数。

该功能默认关闭。如需开启提醒，请在进入会话页面前设置。

```
@property (nonatomic, assign) BOOL enableNewComingMessageIcon;
```

自定义右上角气泡提示

开启该提示功能之后，当一个会话收到大量消息时（超过一个屏幕能显示的内容），进入该会话后，会在右上角提示用户上方存在的未读消息数，用户点击该提醒按钮，会跳转到最开始的未读消息。

- **提示的 UILabel**：

  当 unReadMessage > 10 右上角会显示未读消息数。

```
@property (nonatomic, strong) UILabel *unReadMessageLabel;
```

- **提示的 UIButton：**

```
@property (nonatomic, strong) UIButton *unReadButton;
```

- **自定义：**

  您可自定义聊天页面右上角未读消息控件的字体颜色，背景图片和箭头。在聊天页子类添加以下方法，具体图片名称请根据自身业务添加。

```
- (void)viewWillAppear:(BOOL)animated {
[super viewWillAppear:animated];

//修改文本颜色
[self.unReadMessageLabel setTextColor:[UIColor redColor]];

//修改按钮整体背景图片
[self.unReadButton setBackgroundImage:[UIImage imageNamed:@"这里添加想替换的图片名称"]
forState:UIControlStateNormal];
//修改向上箭头图片
[self.unReadButton.subviews enumerateObjectsUsingBlock:^(__kindof UIView * _Nonnull obj, NSUInteger idx, BOOL * _Nonnull
stop) {
if ([obj isKindOfClass:[UIImageView class]]) {
UIImageView *imageView = (UIImageView *)obj;
UIImage *image = [UIImage imageNamed:@"这里添加想替换的图片名称"];
image = [image resizableImageWithCapInsets:UIEdgeInsetsMake(image.size.width * 0.2, image.size.width * 0.8,
image.size.width * 0.2, image.size.width * 0.2)
resizingMode:UIImageResizingModeStretch];
imageView.image = image;
*stop = YES;
}
}];
}
```

自定义右下角气泡提示

开启该提示功能之后，当会话页面滑动到最下方时，此会话中收到消息会自动更新；当用户停留在上方某个区域阅读时，如果此会话收到新消息，右下角会显示新消息提醒与计数，但不会自动滚动到最下方。用户点击该提醒按钮后，页面滚动到最下方。

- **提示的 UILabel：**

  当 unReadMessage > 10 右上角会显示未读消息数。

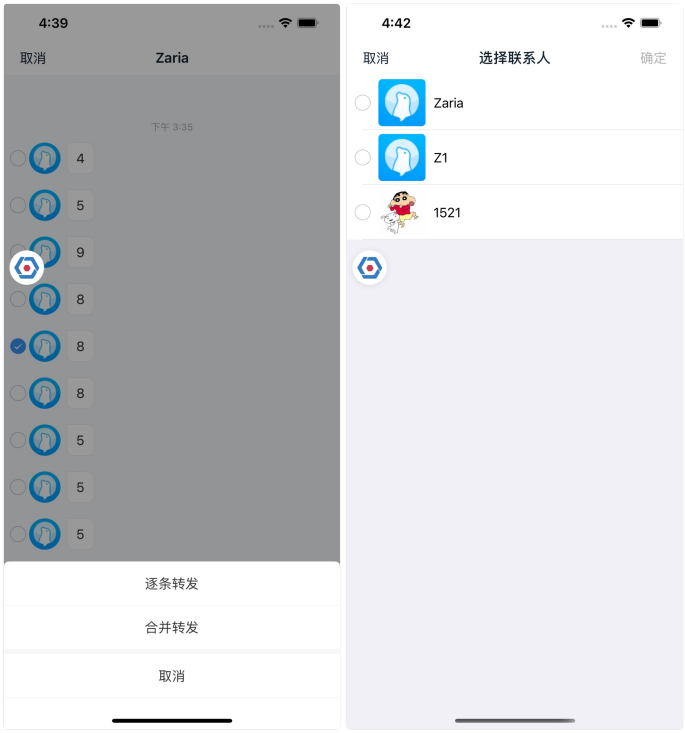
```
@property (nonatomic, strong) UILabel *unReadNewMessageLabel;
```

转发消息　　转发消息

更新时间:2024-08-30

IMKit 支持对单条消息转发，和对多条消息的逐条转发/合并转发的功能，允许用户在聊天页面中将消息转发到其他会话中。转发后消息将出现在目标会话页面的消息列表组件中。

> 提示
>
> IMKit 默认未启用合并转发功能。您可以按需启用合并转发功能。下图中 App 已打开合并转发功能，因此可看到逐条转发和合并转发两个功能选项。



局限

- 并非所有消息类型均支持合并转发。
  - 支持的消息类型：文本、图片、图文、GIF、动态表情（`RC:StkMsg`）、名片、位置、小视频、文件、普通语音、高清语音、音视频通话（`RC:VCSummary`）。
  - 不支持的情况：未在支持列表中的消息类型，例如引用消息，以及未发送成功的消息等特殊情况不支持转发。自定义消息均不支持合并转发。

用法

IMKit 会话页面默认已启用转发功能。用户在会话页面长按消息，在弹框里选择更多，即可展示转发消息选项。

- 逐条转发：默认开启。可转发单聊或多条消息至目标会话。
- 合并转发：默认关闭，不展示。如果合并转发，SDK 会将选中的消息合并为一条合并转发消息，包含消息内容对象 `RCCombineMessage`（类型标识：`RC:CombineMsg`）。合并转发的消息默认折叠显示，可点击展开。

启用合并转发功能

IMKit 支持合并转发功能，该功能默认关闭。您可以修改全局配置，打开 IMKit 的合并转发功能。

```
RCKitConfigCenter.message.enableSendCombineMessage = YES;
```

定制化

自定义转发会话列表

IMKit 中选中逐条转发或合并转发后，在转发会话列表页面默认展示 SDK 本地存储的最近会话列表。

如果需要 SDK 在转发选择会话时跳转应用程序自定义页面，可以重写 RCConversationViewController 的以下方法，弹出自定义的选择会话界面，选择结束之后，把选中的会话数组回传给 IMKit SDK。

```
@param index 0 是逐条转发消息, 1 是合并转发消息。
@param completedBlock 返回需要转发到的会话的列表。

@discussion
开发者如果想更换转发消息的选择会话界面，可以重写此方法，弹出自定义的选择会话界面，选择结束之后，调用completedBlock传入选中的会话即可。
*/
- (void)forwardMessage:(NSInteger)index completed:(void (^)(NSArray<RCConversation *> *conversationList))completedBlock;
@end
```

您还可以参考融云 SealTalk 应用中的实现方式。详见 RCDChatViewController.m 中 forwardMessage 方法。

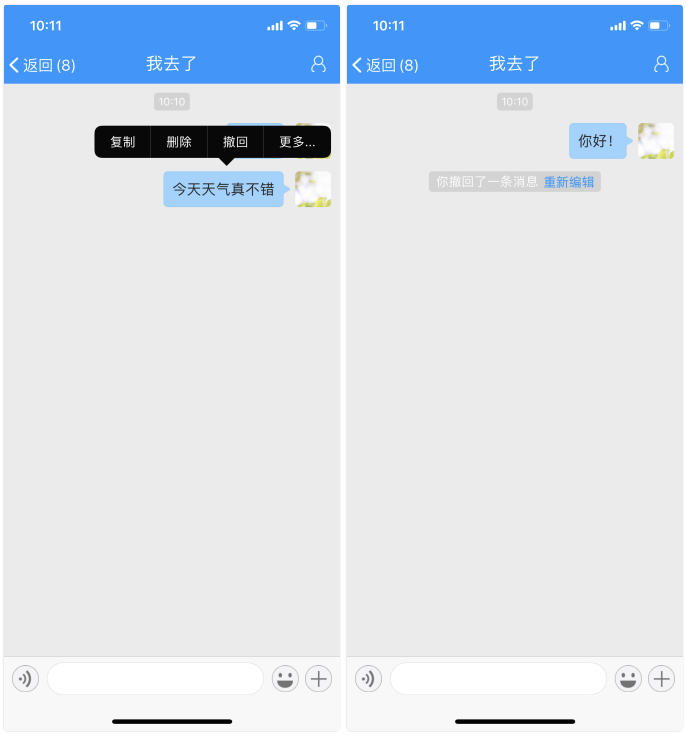
```
- (void)forwardMessage:(NSInteger)index completed:(void (^)(NSArray<RCConversation *> *))completedBlock {
RCDForwardSelectedViewController *forwardSelectedVC = [[RCDForwardSelectedViewController alloc] init];
UINavigationController *navi = [[UINavigationController alloc] initWithRootViewController:forwardSelectedVC];
navi.modalPresentationStyle = UIModalPresentationFullScreen;
[forwardSelectedVC setSelectConversationCompleted:^(NSArray<RCConversation *> *_Nonnull conversationList) {
completedBlock(conversationList);
}];
[self.navigationController presentViewController:navi animated:YES completion:nil];
}
```

撤回消息　　撤回消息

更新时间:2024-08-30

用户通过 App 成功发送了一条消息之后，可能发现消息内容错误等情况，希望将消息撤回，同时从接收者的消息记录中移除该消息。IMKit 默认实现了消息撤回功能。

> 提示
>
> IMKit 在撤回消息后，会替换聊天记录中的原始消息为一条 objectName 为 RC:RcNtf 的撤回通知消息（RCRecallNotificationMessage），可参见服务端文档通知类消息格式。



用法

IMKit 默认启用撤回功能。用户在会话页面长按消息（已发送成功的消息）可打开弹窗，选择撤回。消息撤回后在一定时间内可以“重新编辑”。

定制化

限制撤回操作权限

默认情况下，融云对撤回消息的操作者不作限制。这意味着任何人都可以撤回他人发送的消息。如需限制，可考虑以下方案：

- App 客户端自行限制撤回消息的操作者。例如，不允许 App 业务中的普通用户撤回他人发送的消息，允许 App 业务中的管理员角色撤回他人发送的消息。
- 如需避免用户撤回非本人发送的消息，可以提交工单申请打开**IMLib SDK 只允许撤回自己发送的消息**。从融云服务端进行限制，禁止用户撤回非本人发送的消息。

处理点击重新编辑按钮事件

撤回消息后，会话页面中会展示 RCRecallNotificationMessage，消息 Cell 中会展示「重新编辑」按钮。用户点击重新编辑后触发此回调，且不会再触发消息点击回调 `didTapMessageCell` 方法。

```
- (void)didTapReedit:(RCMessageModel *)model;
```

参数	类型	说明
model	RCMessageModel	消息 Cell 的数据模型

修改消息可撤回的最大时间

IMKit 默认允许在消息发送后 120 秒内撤回。您可以通过全局配置调整该上限。

```
RCKitConfigCenter.message.maxRecallDuration = 120;
```

修复撤回后可重新编辑的时间

IMKit 默认允许在消息撤回后 300 秒内可点击重新编辑，仅文本消息支持撤回再编辑。您可以通过全局配置调整该上限。

```
RCKitConfigCenter.message.reeditDuration = 300;
```

其他定制化

IMKit SDK 默认已经实现了一套消息撤回和展示逻辑，不需要额外调用会话相关 API。如果已有实现无法满足您的需求，可以使用 IMKit 中相关 API。详见撤回消息。

关闭撤回功能

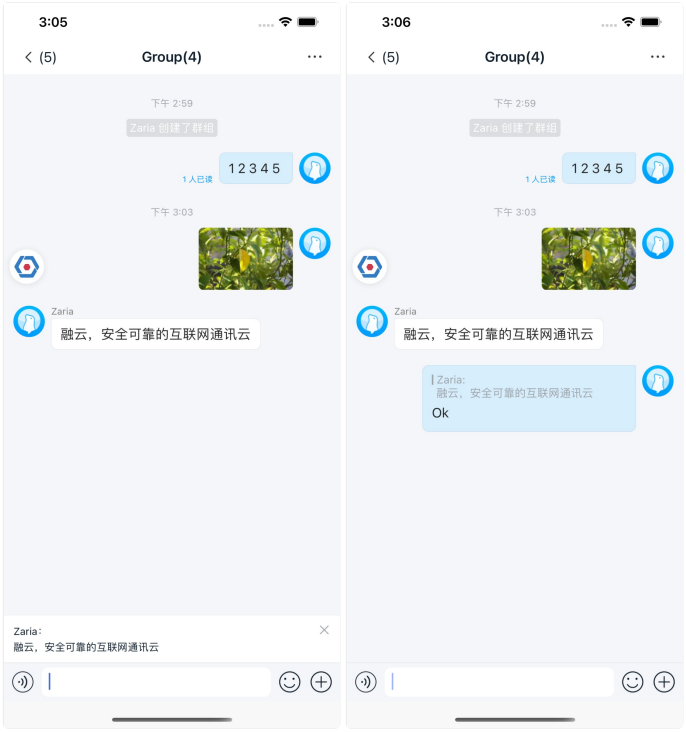
IMKit 默认开启撤回消息功能。您可以通过 IMKit 全局配置关闭撤回功能。

```
RCKitConfigCenter.message.enableMessageRecall = NO;
```

引用回复 引用回复

更新时间:2024-08-30

IMKit 支持引用回复功能，允许用户在聊天页面中回复彼此的消息。消息将出现在会话页面的消息列表组件中。引用回复功能默认发送的消息包含引用消息内容对象 RCReferenceMessage（类型标识：RC:ReferenceMsg）。



局限

引用回复功能目前有以下限制：

- 仅支持文本消息、文件消息、图文消息、图片消息、引用消息的引用。
- 引用深度仅支持一度，即只能引用回复原始消息。如果多重引用，只展示上一层被引消息内容。

用法

IMKit 会话页面默认已启用引用回复功能。用户在会话页面长按消息，在弹框里选择**引用消息**，即可引用该消息。在输入区添加消息内容后，SDK 默认会将输入内容与被引消息组合为 RCReferenceMessage，并发送到会话中。

定制化

自定义应用消息的 UI

IMKit 默认生成和发送引用消息（RC:ReferenceMsg），使用 RCReferenceMessageCell 模板展示在消息列表中。

IMKit 中所有消息模板都继承自 RCMessageCell，自定义消息 Cell 也需要继承 RCMessageCell。详见修改消息的展示样式。

关闭引用回复功能

您可以通过全局配置关闭引用回复功能。

```
RCKitConfigCenter.message.enableMessageReference = NO;
```

快捷回复 快捷回复

更新时间:2024-08-30

IMKit 支持为单聊、群聊会话页面设置常用回复语。

> 提示
>
> IMKit 默认未启用该功能。

局限

- 单条常用语超过 30 个字可能无法完全展示。

用法

在进入会话界面前设置常用语：

```
[self.chatSessionInputBarControl setCommonPhrasesList:@[@"你好", @"不错", @"是这样", @"没问题", @"谢谢"]];
```

如需再次设置常用回复语，新的常用语列表会覆盖已有常用语。如果 IMKit ≧ 5.6.1，设置常用语后页面会立即刷新。

如果 IMKit < 5.6.1，必须主动调用会话页面 `chatSessionInputBarControl` 属性的以下方法刷新常用语页面，否则无法显示最新常用语。

```
- (void)updateStatus:(KBottomBarStatus)status animated:(BOOL)animated
```

点击常用语按钮事件

> 提示

> 要求 IMKit 版本 ≧ 5.6.3。

用户在会话页面点击常用语按钮后会弹出快捷回复。您可以重写 RCConversationViewController 的以下方法，返回 YES 表示拦截，您可以自定义点击常用语按钮后的逻辑；否则返回 NO，继续执行 SDK 默认逻辑。

```
- (BOOL)didTapCommonPhrasesButton;
```

集成 CallKit 通话功能　　集成 CallKit 通话功能 语音/视频通话插件

更新时间:2024-08-30

IMKit SDK 提供的扩展面板中可添加音视频通话功能入口。语音通话、视频通话插件由融云音视频通话（呼叫）SDK CallKit 提供。您需要完成以下工作：

- 集成音视频通话（呼叫）SDK。详见 CallKit 文档实现音视频通话。
- 在控制台开通音视频服务。前往音视频通话服务页面。

请参考您的 App 集成 IMKit 的方式，选择同一种方式导入 CallKit 插件。不要混用 Framework 和源码导入方式。成功集成音视频模块之后在扩展面板会自动展示这两个插件。

> 提示
>
> 扩展面板展示音/视频通话按钮，需要等待 IM 服务下发的配置生效以及客户端本地缓存更新，最多可能需要等待两小时。如长时间未展示，请尝试退出当前账号，清除缓存，卸载重装应用，然后查看是否解决问题。

如果 App 集成 CallKit SDK 后，IMKit 的扩展面板中未出现语音通话、视频通话入口，请检查以下项目：

1. 是否已在控制台开通音视频服务。前往音视频通话服务页面。
2. 扩展面板展示音/视频通话按钮，需要等待 IM 服务下发的配置生效以及客户端本地缓存更新，最多可能需要等待两小时。如长时间未展示，请尝试退出当前账号，清除缓存，卸载重装应用，然后查看是否解决问题。

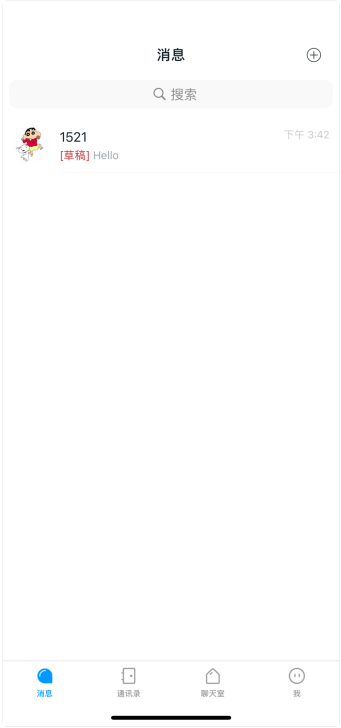
如果 App 集成了 CallKit SDK，但不希望自动展示相关按钮，可参见输入区域。

会话草稿 会话草稿

更新时间:2024-08-30

IMKit 支持会话草稿功能。

> 提示
>
> 用户在会话页面输入框中输入文本后没有发送，退出会话页面到会话列表，会话列表会显示草稿提示及草稿内容。



用法

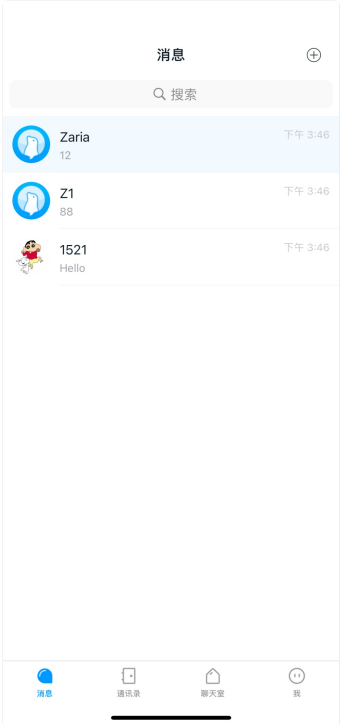
IMKit 中默认已实现了获取会话草稿、删除草稿的功能和页面刷新，您不需要额外调用 API。

会话置顶 会话置顶

更新时间:2024-08-30

IMKit 支持展示置顶会话。

> 提示
>
> IMKit 可根据会话的置顶属性在会话列表中置顶展示会话，但未在 UI 上实现设置置顶功能。



局限

- IMKit 未在 UI 上实现设置会话置顶功能。

用法

您需要自行在 UI 上实现设置会话置顶的功能。用户设置会话置顶后，该状态将会被同步到服务端。融云会为用户自动在设备间同步会话置顶的状态数据。客户端可以通过监听器获取同步通知。

设置是否置顶

设置会话置顶后，会话将在会话列表页面置顶显示。所有置顶会话按照会话时间降序排列。

IMKit SDK 未直接提供该方法，您需要使用 IMLib 中的方法。

```
[[RCCoreClient sharedCoreClient] setConversationToTop:ConversationType_PRIVATE targetId:@"targetId" isTop:YES
completion:^(BOOL ret) {

}];
```

客户端一般通过本地消息数据自动生成会话与会话列表。如果需要置顶的会话在本地尚不存在， SDK 会自动创建该会话。

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
isTop	BOOL	是否置顶

返回 YES 说明设置成功。

监听置顶状态同步

即时通讯业务支持会话状态（置顶状态数据和免打扰状态数据）同步机制。设置会话状态同步监听器后，如果会话状态改变，可在本端收到通知。

会话的置顶和免打扰状态数据同步后，SDK 会分发下面通知。

```
FOUNDATION_EXPORT NSString *const RCKitDispatchConversationStatusChangeNotification;
```

Notification 的 object 是 RCConversationStatusInfo 对象的数组，userInfo 为 nil。

注册通知监听器：

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onConversationStatusChanged:)
name:RCKitDispatchConversationStatusChangeNotification
object:nil];
```

收到通知之后可以更新您的会话的状态。

```
- (void)onConversationStatusChanged:(NSNotification *)notification {
NSArray<RCConversationStatusInfo *> *conversationStatusInfos = notification.object;
}
```

获取会话置顶状态与置顶会话

您可以从客户端主动获取会话置顶状态数据和置顶会话。

IMKit SDK 未直接提供该方法，您需要使用 IMLib 中的方法。详见 IMLib 文档会话置顶 中的获取会话置顶状态与获取置顶会话列表。

控制置顶会话展示顺序

如果需要在控制会话列表最顶部的置顶会话展示顺序，可通过修改数据源实现。重写 RCConversationListViewController 的以下方法，重组数据源并返回，根据您的需求调整置顶会话顺序。

```
/*!

即将加载增量数据源的回调



@param dataSource 即将加载的增量数据源（元素为RCConversationModel对象）

@return 修改后的数据源（元素为RCConversationModel对象）



@discussion 您可以在回调中修改、添加、删除数据源的元素来定制显示的内容，会话列表会根据您返回的修改后的数据源进行显示。

数据源中存放的元素为会话Cell的数据模型，即RCConversationModel对象。

2.9.21 及其以前版本，dataSource 为全量数据，conversationListDataSource = dataSource

2.9.22 及其以后版本，dataSource 为增量数据，conversationListDataSource += dataSource，如果需要更改全量数据的内容，可以更改
conversationListDataSource

*/

- (NSMutableArray *)willReloadTableData:(NSMutableArray *)dataSource;
```

发送消息　　发送消息

更新时间:2024-08-30

IMKit 内置会话页面已实现了发送各类型消息的功能和 UI（部分消息类型需要插件支持）。当您在自定义页面需要发送消息时，可使用 IMKit 核心类 RCIM 下发送消息的方法。这些方法除了提供发送消息的功能外，还会触发 IMKit 内置页面的更新。

IMKit 支持发送普通消息和媒体类消息（参考消息介绍），普通消息父类是 RCMessageContent，媒体消息父类是 RCMediaMessageContent。发送媒体消息和普通消息本质的区别为是否有上传数据过程。

> 重要
>
> - 请务必使用 IMKit 核心类 RCIM 下发送消息的方法，否则不会触发页面刷新。发送普通消息使用 `sendMessage` 方法，发送媒体消息使用 `sendMediaMessage` 方法。
> - 客户端 SDK 发送消息存在频率限制，每秒最多只能发送 5 条消息。

发送普通消息

在发送消息前，需要构造 RCMessage 对象。RCMessage 对象中包含要发送的普通消息内容，即 RCMessageContent 的子类，例如文本消息（RCTextMessage）。

调用 RCIM 的发送消息方法时，SDK 会触发内置的会话列表和会话页面的更新。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:messageContent];


[[RCIM sharedRCIM]
sendMessage:message
pushContent:nil
pushData:nil
successBlock:^(RCMessage *successMessage) {
//成功
}
errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}];
```

sendMessage 中直接提供了用于控制推送通知内容（pushContent）和推送附加信息（pushData）的参数。不过，如果您需要更精细地控制离线推送通知，例如标题、内容、图标、或其他第三方厂商个性化配置，请使用消息推送属性进行配置，详见自定义消息推送通知。请注意，聊天室会话不支持离线消息机制，因此也不支持离线消息转推送。

- 如果发送的消息属于 SDK 内置消息类型，例如 RCTextMessage，这两个参数可设置为 `nil`。一旦消息触发离线推送通知时，融云服务端会使用各个内置消息类型默认的 `pushContent` 值。关于各类型消息的默认推送通知内容，详见用户内容类消息格式。
- 如果消息类型为自定义消息类型，且需要支持离线推送通知，则必须向融云提供 `pushContent` 字段，否则用户无法收到离线推送通知。
- 如果消息类型为自定义消息类型，但不需要支持远程推送通知（例如通过自定义消息类型实现的 App 业务层操作指令），可将 `pushContent` 字段留空。
- `RCMessage` 的推送属性配置 `MessagePushConfig` 的 `pushContent` 和 `pushData` 会覆盖此处配置，并提供更多配置能力，例如自定义推送

通知的标题。详见自定义消息推送通知。

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（conversationType），会话 ID（targetId），消息内容（content）。详见消息介绍 中对 RCMessage 的结构说明。
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性 。 • 如果希望使用融云默认推送通知内容，可以填 nil。注意自定义消息类型无默认值。 • 如果要为该条消息指定个性化的离线推送通知，必须在以上任一处向融云提供 pushContent 字段内容。 • 如果您自定义的消息类型需要离线推送服务，必须在以上任一处向融云提供 pushContent 字段内容，否则用户无法收到离线推送通知。
pushData	NSString	Push 附加信息。可设置为 nil。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性 。
successBlock	Block	消息发送成功回调。
errorBlock	Block	消息发送失败回调。

发送媒体消息

媒体消息 RCMessage 对象的 content 字段必须传入 RCMediaMessageContent 的子类对象，表示媒体消息内容。例如图片消息（RCImageMessage ）、GIF 消息（RCGIFMessage ）等。其他内置媒体消息类型包括文件消息（RCFileMessage ）、位置消息（RCLocationMessage ）、高清语音消息（RCHQVoiceMessage ）、小视频消息（RCSightMessage ），建议先集成相应的 IMKit 插件。

图片消息内容（RCImageMessage ）支持设置为发送原图。

```
RCImageMessage *mediaMessageContent = [RCImageMessage messageWithImageURI:@"path/to/image"];
mediaMessageContent.full YES; // 图片消息支持设置以原图发送

RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:mediaMessageContent];
```

发送媒体消息需要使用 sendMediaMessage 方法。SDK 会为图片、小视频等生成缩略图，根据默认压缩配置 进行压缩，再将图片、小视频等媒体文件上传到融云默认的文件服务器（文件存储时长 ），上传成功之后再发送消息。图片消息如已设置为发送原图，则不会进行压缩。

调用 RCIM 的发送消息方法时，SDK 会触发内置的会话列表和会话页面的更新。

```
[[RCIM sharedRCIM] sendMediaMessage:message
pushContent:nil
pushData:nil
progress:^(int progress, RCMessage *progressMessage) {
//媒体上传进度
}
successBlock:^(RCMessage *successMessage) {
//成功
}
errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}
cancel:^(RCMessage *cancelMessage) {
//取消
}];
```

sendMediaMessage 中直接提供了用于控制推送通知内容（pushContent）和推送附加信息（pushData）的参数。不过，如果您需要更精细地控制离线推送通知，例如标题、内容、图标、或其他第三方厂商个性化配置，请使用消息推送属性进行配置，详见自定义消息推送通知。请注意，聊天室会话不支持离线消息机制，因此也不支持离线消息转推送。

- 如果发送的消息属于 SDK 内置消息类型，例如 RCImageMessage ，这两个参数可设置为 nil。一旦消息触发离线推送通知时，融云服务端会使

用各个内置消息类型默认的 `pushContent` 值。关于各类型消息的默认推送通知内容，详见用户内容类消息格式 。

- 如果消息类型为自定义消息类型，且需要支持离线推送通知，则必须向融云提供 `pushContent` 字段，否则用户无法收到离线推送通知。
- 如果消息类型为自定义消息类型，但不需要支持远程推送通知（例如通过自定义消息类型实现的 App 业务层操作指令），可将 `pushContent` 字段留空。
- `RCMessage` 的推送属性配置 `MessagePushConfig` 的 `pushContent` 和 `pushData` 会覆盖此处配置，并提供更多配置能力，例如自定义推送通知的标题。详见自定义消息推送通知。

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（`conversationType`），会话 ID（`targetId`），消息内容（`content`）。详见消息介绍 中对 `RCMessage` 的结构说明。
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 `RCMessage` 的推送属性（`RCMessagePushConfig`）中配置，会覆盖此处配置，详见配置消息的推送属性 。 • 如果希望使用融云默认推送通知内容，可以填 `nil`。注意自定义消息类型无默认值。 • 如果要为该条消息指定个性化的离线推送通知，必须在以上任一处向融云提供 `pushContent` 字段内容。 • 如果您自定义的消息类型需要离线推送服务，必须在以上任一处向融云提供 `pushContent` 字段内容，否则用户无法收到离线推送通知。
pushData	NSString	Push 附加信息。可设置为 `nil`。您也可以在 `RCMessage` 的推送属性（`RCMessagePushConfig`）中配置，会覆盖此处配置，详见配置消息的推送属性 。
progressBlock	Block	媒体上传进度的回调。
successBlock	Block	消息发送成功回调。
errorBlock	Block	消息发送失败回调。
cancel	Block	取消发送的回调。

> 提示
>
> 如果媒体文件已经成功上传，可以直接当做普通消息发送。

发送多媒体消息并且上传到自己的服务器

> 提示
>
> SDK 从 5.3.5 版本开始提供该功能。

IMKit SDK 从 5.3.5 开始，支持由 App 自行处理媒体文件上传逻辑（可上传到自己的服务器），再发送消息。同时 SDK 会更新 UI 状态。您需要在 `sendMessageMessage` 的 `uploadPrepareBlock` 中自行实现媒体文件上传。

- 上传媒体文件的过程中，可调用 RCUploadMediaStatusListener 的 `updateBlock`、`errorBlock`，通知 IMKit SDK 当前上传媒体文件的进度和状态，SDK 会更新 UI。
- 上传完毕后，取得网络文件 URL。通过 RCUploadMediaStatusListener 的 `currentMessage` 属性获取当前 `RCMessage` 对象，将 `RCMessage` 对象的 `content` 中的对应 URL 字段设置成上传成功的网络文件 URL。
- 调用 RCUploadMediaStatusListener 的 `successBlock`，通知 SDK 文件上传完成。

```
RCImageMessage *mediaMessageContent = [RCImageMessage messageWithImageURI:@"path/to/image"];
mediaMessageContent.full = YES; // 图片消息支持设置以原图发送

RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:mediaMessageContent];

[[RCIM sharedRCIM] sendMediaMessage:message
pushContent:nil
pushData:nil
uploadPrepare:^(RCUploadMediaStatusListener *uploadListener) {
RCMessage *currentMessage = uploadListener.currentMessage;
// App 在上传媒体文件时，需要在监听中调用 updateBlock、successBlock 与 errorBlock，通知 IMKit SDK 当前上传媒体文件的进度和状态，SDK 会更新 UI。
if ([currentMessage.content isKindOfClass:[RCImageMessage class]]) {
RCImageMessage *content = (RCImageMessage *)currentMessage.content;
content.remoteUrl = remoteUrl;
uploadListener.successBlock(content)};
}
progress:^(int progress, RCMessage *progressMessage) {
//媒体上传进度
}
successBlock:^(RCMessage *successMessage) {
//成功
}
errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}
cancel:^(RCMessage *cancelMessage) {
//取消
}];
```

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（conversationType），会话 ID（targetId），消息内容（content）。详见消息介绍 中对 RCMessage 的结构说明。
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。 • 如果希望使用融云默认推送通知内容，可以填 nil。注意自定义消息类型无默认值。 • 如果要为该条消息指定个性化的离线推送通知，必须在以上任一处向融云提供 pushContent 字段内容。 • 如果您自定义的消息类型需要离线推送服务，必须在以上任一处向融云提供 pushContent 字段内容，否则用户无法收到离线推送通知。
pushData	NSString	Push 附加信息。可设置为 nil。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。
uploadPrepareBlock	Block	发送多媒体消息并上传到自己服务器的回调。
progressBlock	Block	媒体上传进度的回调。
successBlock	Block	消息发送成功回调。
errorBlock	Block	消息发送失败回调。
cancel	Block	取消发送的回调。

为消息禁用推送通知

在接收者未上线时，融云服务端默认触发推送服务，将消息通过推送通道下发（服务端是否触发推送也会收到应用级别、用户级别免打扰设置的影响）。

有时 App 用户可能希望在发送消息时就指定该条消息不需要触发推送。IMKit 提供该能力，但并未在 UI 上实现。如有需要，可通过以下方式实现：

1. 在 App 继承的会话页面 RCConversationViewController 子类中，通过 willSendMessage 回调拦截消息，详见 IMKit 会话页面的页面事件监听。如果不在会话页面拦截，您也可以通过单独设置消息拦截代理实现，详见消息拦截。

2. 获取 RCMessage 消息对象。

   - 如果 SDK < 5.3.5，您需要利用回调中获取的消息内容 `RCMessageContent`，组装 RCMessage 对象。
   - 如果 SDK ≧ 5.3.5，您可以直接通过消息拦截获取 RCMessage 对象。

3. 将 `messageConfig` 的 `disableNotification` 设置为 YES 禁用该条消息的推送通知。接收方再次上线时会通过融云服务端的离线消息缓存（最多缓存 7 天）自动收取单聊、群聊、系统会话消息。

```
RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:someMsg];

message.messageConfig.disableNotification = YES;
```

4. 调用发送消息方法重新发送消息。

自定义消息推送通知

IMKit 支持对单条消息的推送行为添加个性化配置（RCMessagePushConfig），但并未在 UI 上实现。如有需要，您可以在发送时拦截消息添加配置。拦截方式与为消息禁用推送通知一致。

`RCMessagePushConfig` 可提供以下能力，具体说明详见「APNs 推送开发指南」中的配置消息的推送属性。

- 自定义推送标题、推送通知内容
- 自定义通知栏图标
- 添加远程推送附加信息
- 越过接收客户端的配置，强制在推送通知内显示通知内容
- 其他 APNs 或 Android 推送通道支持的个性化配置

接收消息　　接收消息

更新时间:2024-08-30

您可通过设置代理监听或通知监听拦截 SDK 接收的消息，并进行相应的业务操作。

消息接收代理委托协议

IMKit SDK 提供了消息接收监听协议 RCIMReceiveMessageDelegate，可接收实时消息或离线消息。该协议提供两个代理方法，监听消息处理只需要在其中一个代理方法内实现。SDK 会通过此方法接收包含单聊、群聊、聊天室、系统类型的所有消息。只需全局设置一次即可，多次设置会导致其他代理失效。

如果您设置了IMlib消息监听之后，SDK在接收到消息时候会执行此方法。您可以根据 nLeft 的数量来优化您的 App 体验和性能，比如收到大量消息时等待 nLeft 为 0 再刷新 UI。

```
/*!
接收消息的回调方法

@param message 当前接收到的消息
@param nLeft 还剩余的未接收的消息数，left>=0

*/
- (void)onRCIMReceiveMessage:(RCMessage *)message left:(int)left;
```

与第一个代理方法相比，第二个代理方法额外暴露了 offline 和 hasPackage 参数。您可以根据 nLeft、offline、hasPackage 选择合适的时机刷新 UI。建议当 hasPackage=0 并且 nLeft=0 时刷新 UI。如果使用此方法，那么就不能再使用 RCIM 中的 - (void)onRCIMReceived:(RCMessage *)message left:(int)nLeft ，否则会出现重复操作的情形。

```
/**
接收消息的回调方法

@param message 当前接收到的消息
@param nLeft 还剩余的未接收的消息数，left>=0
@param offline 是否是离线消息
@param hasPackage SDK 拉取服务器的消息以包(package)的形式批量拉取，有 package 存在就意味着远端服务器还有消息尚未被 SDK
拉取
*/
- (void)onRCIMReceived:(RCMessage *)message
left:(int)nLeft
offline:(BOOL)offline
hasPackage:(BOOL)hasPackage;
```

参数	类型	说明
message	RCMessage	接收的消息对象。
nLeft	int	当客户端连接成功后，服务端会将所有离线消息?以消息包（Package）的形式下发给客户端，每个 Package 中最多含 200 条消息。客户端会解析 Package 中的消息，逐条上抛并通知应用。nLeft 为当前正在解析的消息包（Package）中还剩余的消息条数。
offline	boolean	当前消息是否离线消息。
hasPackage	boolean	是否在服务端还存在未下发的消息包（Package）。

添加消息接收代理

SDK 支持设置多个消息接收代理。所有接收到的消息都会在此接口方法中回调。建议在应用生命周期内注册消息监听。

```
[[RCIM sharedRCIM] addReceiveMessageDelegate:self];
```

移除消息接收代理

SDK 支持移除监听器。为了避免内存泄露，请在不需要监听时将监听器移除。

```
[[RCIM sharedRCIM] removeReceiveMessageDelegate:self];
```

通知监听

开发者可以在任何时机使用通知监听消息接收。

收到消息后，SDK 会分发下面通知。

```
FOUNDATION_EXPORT NSString *const RCKitDispatchMessageNotification;
```

Notification 的 object 为 RCMessage 消息对象。 userInfo 为 NSDictionary 对象。

userInfo 字典对象说明:

key	value	说明
left	NSNumber 类型	剩余待接收消息数

```
[[NSNotificationCenter defaultCenter]
addObserver:self
selector:@selector(didReceiveMessageNotification:)
name:RCLibDispatchReadReceiptNotification
object:nil];
```

```
- (void)didReceiveMessageNotification:(NSNotification *)notification {
RCMessage *message = notification.object;
int left = [[notification.userInfo objectForKey:@"left"] intValue];
}
```

禁用消息排重机制

消息排重机制会在 SDK 接收单聊、群聊、系统消息、聊天室时自动去除内容重复消息。当 App 本地存在大量消息，SDK 默认的排重机制可能会因性能问题导致收消息卡顿。因此在接收消息发生卡顿问题时，可尝试关闭 SDK 的排重机制。

为什么接收消息可能出现消息重复

发送端处于弱网情况下可能出现该问题。A 向 B 发送消息后，消息成功到达服务端，并成功下发到接收者 B。但 A 由于网络等原因可能未收到服务端返回的 ack，导致 A 认为没有发送成功。此时如果 A 重发消息（IMKit 默认具有失败自动重发机制，建议同时关闭），此时 B 就会收到与之前重复的消息（消息内容相同，但 Message UID 不同）。

关闭消息排重机制

单聊、群聊、系统消息使用 IMLib 的核心类 `RCCoreClient` 中的 `setCheckDuplicateMessage` 方法（要求 SDK 版本 ≧ 5.3.4），禁用消息排重行为。请在 SDK 初始化之后，建立 IM 连接之前完成以下配置：

```
BOOL enableCheck = NO; // 关闭消息排重
[[RCCoreClient sharedCoreClient] setCheckDuplicateMessage:enableCheck];
```

聊天室消息排重使用 RongChatRoom 的核心类 RCChatRoomClient 中的 setCheckChatRoomDuplicateMessage 方法（要求 SDK 版本 ≧ 5.8.2），禁用消息排重行为。请在 SDK 初始化之后，建立 IM 连接之前完成以下配置：

```
BOOL enableCheck = NO; // 关闭消息排重
[[RCChatRoomClient sharedChatRoomClient] setCheckChatRoomDuplicateMessage:enableCheck];
```

关闭失败重发机制

IMKit SDK 默认启用了消息失败自动重发机制。在禁用消息排重机制后，为避免收到 UID 重复的消息，建议同时在 App 中禁用 IMKit 的失败重发机制。

请在 SDK 初始化之后，建立 IM 连接之前完成以下配置：

```
RCKitConfigCenter.message.enableMessageResend = NO;
```

拦截消息　　拦截消息

更新时间:2024-08-30

IMKit SDK 可在消息发送前、发送后、接收时进行拦截。

消息拦截功能提供以下协议：

- RCIMMessageInterceptor（要求 SDK ≧ 5.3.5）：支持在发送前、发送后拦截消息，可获取消息对象（RCMessage）。
- `RCIMSendMessageDelegate`（已废弃）：支持在发送前、发送后拦截消息，但仅可获取RCMessage 的 `content` 字段，即 RCMessageContent 或 RCMediaMessageContent 的子类对象。从 SDK 5.3.5 版本开始废弃。
- RCIMReceiveMessageDelegate：支持在实时收到消息时进行拦截。

> 提示
>
> IMKit 的会话页面（RCConversationViewController）也提供与消息相关的回调方法，可用于拦截消息。详见会话页面的页面事件监听。

在消息发送前后拦截消息（SDK ≧ 5.3.5）

通过 `RCIMMessageInterceptor` 协议提供的回调方法，App 可以拦截待发送和已发送的消息，可获取 RCMessage 对象。

在消息发送前、后进行拦截，可能的适用场景如下：

- 在发送前拦截并修改 `RCMessage`，后续由 SDK 继续发送（注意，如果 5.3.5 ≦ SDK < 5.4.5，这种方式仅支持修改 `RCMessage` 的 `content` 字段；如果 SDK ≧ 5.4.5，支持修改 `RCMessage`）。返回 `NO` 表示 App 需要拦截，但需要 SDK 发送修改后的 `RCMessage`。
- 在发送前拦截并修改 `RCMessage`，后续由 App 再次发送。这种方式支持修改 `RCMessage`。返回 `YES` 表示 App 需要拦截并自行处理后续流程。App 需要在处理完毕后自行发送修改后的消息。
- 在发送前拦截 `RCMessage`，将媒体消息中的文件上传至 App 指定的文件服务器，后续由 App 再次发送。返回 `YES` 表示 App 需要拦截并自行处理后续流程。注意，App 需要调用上传文件到自定义服务器的发送消息方法。详见发送消息。
- 在发送后进行拦截，可修改数据库中 `RCMessage` 对象的部分属性，例如消息的附加信息 `setMessageExtra`。详见 RCCoreClient 或 RCChannelClient 中的设置方法。注意，您可能需要先检查被拦截消息的发送状态（`RCMessage.sentStatus`），再判断下一步如何处理。

设置代理委托

只需全局设置一次即可，多次设置会导致其他代理失效。

```
[RCIM sharedRCIM].messageInterceptor = self;
```

实现此功能需要开发者遵守 RCIMMessageInterceptor 协议。

```
#pragma mark -- RCIMMessageInterceptor

- (BOOL)interceptWillSendMessage:(RCMessage *)message {
if ([message.content isKindOfClass:[RCCombineMessage class]]) {
// 示例1：拦截合并转发消息，由 APP 负责发送
[[RCIM sharedRCIM] sendMediaMessage:message pushContent:nil pushData:nil uploadPrepare:^(RCUploadMediaStatusListener
*uploadListener) {

RCCombineMessage *msgContent = (RCCombineMessage *)uploadListener.currentMessage.content;
msgContent.remoteUrl = @"https://your_file_server.com.com/test.html";
uploadListener.successBlock(msgContent);

} progress:nil successBlock:nil errorBlock:nil cancel:nil];
// 返回 YES，SDK 不会继续发送，需要您自己调用发送
return YES;
}
else if ([message.content isKindOfClass:[RCImageMessage class]]) {
// 示例2.1：过滤不需要拦截的图片消息，条件按需自定义
RCImageMessage *msgContent = (RCImageMessage *)message.content;
if ([msgContent.remoteUrl containsString:@"your_file_server"]) {
// 图片已经上传到 App File Server，不需要拦截
return NO;
}

// 示例2.2：拦截图片消息，由 APP 负责发送
[[RCIM sharedRCIM] sendMediaMessage:message pushContent:nil pushData:nil uploadPrepare:^(RCUploadMediaStatusListener
*uploadListener) {

RCImageMessage *msgContent = (RCImageMessage *)uploadListener.currentMessage.content;
// 去实现自己上传附件
// 获取到上传的 url 地址， 并赋值给 remoteUrl
msgContent.remoteUrl = @"http://your_file_server.com/test.jpg";
uploadListener.successBlock(msgContent);

} progress:nil successBlock:nil errorBlock:nil cancel:nil];

// 返回 YES，SDK 不会继续发送，需要您自己调用发送
return YES;
}

else if ([message.content isKindOfClass:[RCTextMessage class]]) {
// 示例3：拦截并继续使用 SDK 方法发送，只做文本消息内容更新
RCTextMessage *msgContent = (RCTextMessage *)message.content;
msgContent.content = @"文本内容被替换了";

// 返回 NO, 继续使用 SDK 方法发送
return NO;
}


return NO;
}
```

拦截接收的消息

IMKit 的接收消息监听协议 RCIMReceiveMessageDelegate 中提供了接收消息时的拦截方法。

代理设置方法详见消息监听。设置代理委托后，实现如下方法。返回 YES 表示需要拦截接收到的消息，拦截后不再触发接收消息的回调。

```
@protocol RCIMReceiveMessageDelegate <NSObject>

/*!
当 Kit 收到消息回调的方法
@param message 接收到的消息
@return YES 拦截，不显示 ; NO: 不拦截, 显示此消息。
*/

- (BOOL)interceptMessage:(RCMessage *)message;

@end
```

提示

只处理接收消息后是否在界面上是否显示。如果重新加载会话页面，消息仍会显示。如有需要，可删除该消息。详见删除消息。

在消息发送前后拦截消息（已废弃）

提示

从 SDK 5.3.5 版本开始废弃 `RCIMSendMessageDelegate` 协议。

当 IMKit 在发送消息时，开发者可以通过代理监听拦截消息，开发者只需全局设置一次即可，多次设置会导致其他代理失效。

实现此功能需要开发者遵守 RCIMSendMessageDelegate 协议。

设置代理委托：

```
[RCIM sharedRCIM].sendMessageDelegate = self;
```

提示

如果开发者在聊天页面拦截消息，也可以使用 RCConversationViewController 中的 `willSendMessage:` 方法和 `didSendMessage:content:` 方法。

删除消息　　删除消息

更新时间:2024-08-30

IMKit 会话页面默认已实现了长按删除消息的功能，支持仅删除单条本地消息，或同步删除本地和远端的单条消息。

您可以修改 IMKit 会话页面长按消息菜单中删除按钮的行为，详见会话页面。

如果 IMKit 的已有实现无法满足您的需求，可以直接使用 IMLib 提供的以下能力：

- 仅从本地删除指定消息（消息 ID）
- 仅从本地删除会话全部历史消息
- 删除会话内指定消息（消息对象）
- 删除会话历史消息（时间戳），可选仅本地删除、或者同时从本地和服务端删除消息
- 仅从服务端删除会话历史消息（时间戳）

> 提示
>
> 具体的核心类、API 与 使用方法，详见 IMLib 文档 删除消息。注意：IMLib 中的方法并不提供页面刷新能力，您需要根据业务需求自定义通知机制进行页面刷新。

撤回消息　　撤回消息

更新时间:2024-08-30

IMKit SDK 默认已经实现了一套消息撤回和展示逻辑，不需要额外调用会话相关 API。如果已有实现无法满足您的需求，可以使用 IMKit 中相关 API。

撤回消息

RCConversationViewController 直接提供了撤回消息的 API。只有存储并发送成功的消息才可以撤回。

```
- (void)recallMessage:(long)messageId;
```

参数	类型	说明
messageId	long	被撤回消息的消息Id

监听他人撤回消息事件

撤回监听支持通知监听和代理监听两种方式。

通知监听

消息被撤回后，SDK 会分发通知下面通知。

```
FOUNDATION_EXPORT NSString *const RCKitDispatchRecallMessageNotification;
```

Notification 的 object 为 消息 Id 的 NSNumber 值。

```
[[NSNotificationCenter defaultCenter]
addObserver:self
selector:@selector(didRecallMessageNotification:)
name:RCKitDispatchRecallMessageNotification
object:nil];
```

```
- (void)didRecallMessageNotification:(NSNotification *)notification {
long long messageId = [notification.object longLongValue];
}
```

代理监听

设置一个遵循 RCIMReceiveMessageDelegate 协议的代理后，可监听消息撤回事件。该协议即接收消息的协议，可参考 接收消息

```objc
@protocol RCIMReceiveMessageDelegate <NSObject>
/*!
消息被撤回的回调方法

@param message 被撤回的消息

@discussion 被撤回的消息会变更为RCRecallNotificationMessage，App需要在UI上刷新这条消息。
@discussion 和上面的 - (void)onRCIMMessageRecalled:(long)messageId 功能完全一致，只能选择其中一个使用。
*/
- (void)messageDidRecall:(RCMessage *)message;
@end
```

插入消息　　插入消息

更新时间:2024-08-30

插入消息的接口只会将消息插入本地数据库，不会向远端发送，所以卸载重装或者换端登录时插入的消息不会从远端同步到本地数据库。

插入发送消息

在本地数据库被插入一条对外发送的消息。返回插入成功的 RCMessage。

> 提示
>
> 此方法如果 sentTime 有问题会影响消息排序，慎用！

```
RCTextMessage *content = [RCTextMessage messageWithContent:@"测试文本消息"];

[[RCCoreClient sharedCoreClient] insertOutgoingMessage:ConversationType_PRIVATE
targetId:@"targetId"
sentStatus:SentStatus_SENT
content:content
sentTime:sentTime
completion:^(RCMessage * _Nullable message) {

}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。
targetId	NSString	会话 ID。
sentStatus	RCSentStatus	发送状态。
content	RCMessageContent	消息的内容。
sentTime	long long	消息发送的 Unix 时间戳，单位为毫秒（传 0 会按照本地时间插入）。
completion	block	完成回调。

插入接收消息

在本地数据库被插入一条接收的消息。返回插入成功的 RCMessage。

> 提示
>
> - SDK 从 5.6.8 版本开始，支持在插入接收消息时指定 RCReceivedStatusInfo 属性。如果低于 5.6.8 版本，请选用指定 [RCReceivedStatus] 属性的接口。
> - sentTime 会影响消息排序，请务必保证 sentTime 正确性。您也可以使用不指定 sentTime 的接口。

```
RCTextMessage *content = [RCTextMessage messageWithContent:@"测试文本消息"];

[[RCCoreClient sharedCoreClient] insertIncomingMessage:ConversationType_PRIVATE
targetId:@"targetId"
senderUserId:@"senderUserId"
receivedStatusInfo:receivedStatusInfo
content:content
sentTime:sentTime
completion:^(RCMessage * _Nullable message) {
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	Target ID 用于标识会话，称为目标 ID 或会话 ID。注意，因为单聊业务中始终使用对端用户 ID 作为标识本端会话的 Target ID，因此在单聊会话中插入本端接收的消息时，Target ID 始终是单聊对端用户 ID。群聊、超级群的会话 ID 分别为群组 ID、超级群 ID。详见消息介绍中关于 Target ID 的说明。
senderUserId	NSString	发送者ID
receivedStatusInfo	RCReceivedStatusInfo	接收状态
content	RCMessageContent	消息的内容
sentTime	long long	消息发送的 Unix 时间戳，单位为毫秒（传 0 会按照本地时间插入）。
completion	block	完成回调。

会话页面显示插入的消息

如果是在会话页面 RCConversationViewController 中插入消息，需要调用下面方法才会实时更新到页面显示。

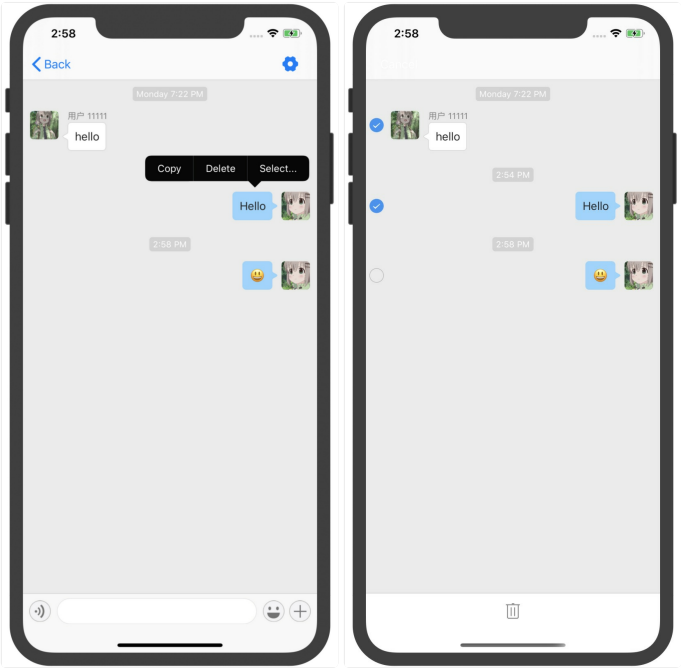
```
RCMessage *insertMessage;
[self appendAndDisplayMessage:insertMessage];
```

消息多选 消息多选

更新时间:2024-08-30

IMKit 会话页面支持长按消息多选功能。

效果展示



自定义多选

您可参考 SealTalk 开源项目 中 RCDChatViewController 的 messageSelectionToolbar 操作实例。

1. 是否为多选状态。

   通过下面属性可设置会话页面多选状态和初始状态，如果为 YES，消息 cell 会变为多选样式。如果为 NO，页面恢复初始状态。

```
@property (nonatomic, assign) BOOL allowsMessageCellSelection;
```

   如果需要修改 SDK 内置消息是否允许多选，可在会话页面重写下面方法，修改消息 cell 的 allowsSelection 属性。自定义消息如果允许多选，也可以设置此属性。

```
- (void)willDisplayMessageCell:(RCMessageBaseCell *)cell atIndexPath:(NSIndexPath *)indexPath;
```

2. 消息数据。

   已经选择的所有消息，只有在 allowsMessageCellSelection 为 YES 时，才有有效值。

```
@property(nonatomic, strong, readonly) NSArray<RCMessageModel *> *selectedMessages;
```

3. 底部视图。

   进入多选状态后，页面底部会出现的工具视图。您可以添加对多选消息的操作事件，如转发、收藏等。您可以在 viewDidLoad 中通过给 messageSelectionToolbar 添加 UIBarButtonItem 的方式添加事件。

```
@property(nonatomic, strong) UIToolbar *messageSelectionToolbar;
```

自定义消息类型 自定义消息类型

更新时间:2024-08-30

IMKit 支持自定义的消息类型（区别于内置消息类型），并支持修改内置消息类型与自定义消息类型在 IMKit SDK 会话页面的展示形式。

创建自定义消息类型

除了使用 SDK 内置消息类型外，还可以根据自己的业务需求自定义消息。

> 提示
>
> 关于如何创建自定义消息类型，详见 IMLib SDK 的自定义消息。

仅当自定义消息的 `persistentFlag` 为以下值时，可在 IMKit 的会话页面中展示：

- `MessagePersistent_ISCOUNTED`
- `MessagePersistent_ISPERSISTED`

如果自定义消息类型带有以上属性，则必须为该自定义消息创建展示模板，否则 IMKit SDK 无法正常展示该类型消息。

为自定义消息创建和注册展示模版

IMKit 支持创建自定义消息 Cell，用于控制消息在会话页面的展示形式。如果您创建了自定义消息类型，且需要将消息展示在会话界面中，必须创建对应的消息展示模板，否则 SDK 无法正常展示该类型消息。详见修改消息展示样式中的新建消息展示模版。

参考资源

您可以参考融云的开源项目 SealTalk 中自定义消息的例子 `RCDTestMessage` 和 `RCDTestMessageCell`。

获取会话 获取会话

更新时间:2024-08-30

IMKit SDK 默认已经实现了一整套会话获取和展示逻辑，使用默认会话列表和会话页面时，不需要额外调用会话相关 API。如果已有实现无法满足您的需求，可以使用 IMLib SDK 中 `RCIMClient` 或 `RCCoreClient` 的相关 API，例如：

- `getConversationList`：获取本地会话列表
- `getConversation`：获取指定会话：

> 提示
>
> 具体使用方法请参见 IMLib 文档获取会话。注意，IMLib 的方法并不提供页面刷新能力，您需要根据业务需求自定义通知机制进行页面刷新。

删除会话 删除会话

更新时间:2024-08-30

IMKit 会话列表默认实现了左滑删除会话的功能。

如果已有实现无法满足您的需求，例如您需要删除全部会话，可以使用 IMLib SDK 中 `RCIMClient` 或 `RCCoreClient` 的相关 API。

> 提示
>
> 具体使用方法请参见 IMLib 文档删除会话。注意，IMLib 的方法并不提供页面刷新能力，您需要根据业务需求自定义通知机制进行页面刷新。

设置会话免打扰　　设置会话免打扰

更新时间:2024-08-30

- 本文仅描述了 SDK 版本 < 5.2.2 时，使用 `RCIMClient` 下接口设置免打扰的方法。
- 如果 SDK 版本 ≧ 5.2.2，推荐使用 `RCChannelClient` 下支持设置免打扰级别的全局免打扰接口。请移步至 IMLib 文档按会话设置免打扰 。

即时通讯服务支持会话免打扰设置。IMKit SDK 可根据会话类型、会话 ID 设置消息提醒状态为「免打扰」。设置后如果客户端在后台运行时，会话中有新的消息，将不会进行通知提醒，可以收到消息内容。如果客户端为离线状态，将不会收到远程通知提醒。

会话的免打扰状态将会被同步到服务端。融云会为用户自动在设备间同步会话免打扰状态数据。客户端可以通过监听器获取同步通知，也可以主动获取最新数据。

设置会话的免打扰状态（< 5.2.2）

`RCIMClient` 类提供 `setConversationNotificationStatus` 方法，可根据会话类型、会话 ID 设置消息提醒状态为「免打扰」。设置成功后，客户端在后台运行时或处于用户离线状态时，均不会收到该会话的新消息通知。修改完成后，您可以主动重新获取数据源刷新 UI。

IMKit SDK 未直接提供该方法，您需要使用 IMLib 中获取会话免打扰状态的方法。

```
[[RCIMClient sharedRCIMClient] setConversationNotificationStatus:ConversationType_PRIVATE
targetId:self.userId
isBlocked:YES
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
isBlocked	BOOL	是否屏蔽消息提醒
successBlock	BOOL	设置成功的回调
errorBlock	BOOL	设置失败的回调。回调参数 `status` 包含错误码，参见 RCErrorCode 。

监听会话的免打扰状态同步

即时通讯业务支持会话状态（置顶状态数据和免打扰状态数据）同步机制。设置会话状态同步监听器后，如果会话状态改变，可在本端收到通知。

会话的置顶和免打扰状态数据同步后，SDK 会分发下面通知。

```
FOUNDATION_EXPORT NSString *const RCKitDispatchConversationStatusChangeNotification;
```

Notification 的 object 是 `RCConversationStatusInfo` 对象的数组，`userInfo` 为 nil。

注册通知监听器：

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onConversationStatusChanged:)
name:RCKitDispatchConversationStatusChangeNotification
object:nil];
```

收到通知之后可以更新您的会话的状态。

```
- (void)onConversationStatusChanged:(NSNotification *)notification {
NSArray<RCConversationStatusInfo *> *conversationStatusInfos = notification.object;
}
```

获取会话的免打扰状态（< 5.2.2）

客户端可以主动获取最新的会话免打扰状态数据。

IMKit SDK 未直接提供该方法，您需要使用 IMLib 中获取会话免打扰状态的方法。

```
[[RCIMClient sharedRCIMClient] getConversationNotificationStatus:ConversationType_PRIVATE
targetId:self.userId
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
success	BOOL	设置成功的回调。回调参数 `nStatus` 为会话设置的消息提醒状态。参见RCConversationNotificationStatus
error	BOOL	设置失败的回调。回调参数 `status` 包含错误码，参见 RCErrorCode 。

获取免打扰状态的会话列表（< 5.2.2）

获取所有设置了免打扰的会话。返回会话列表不包含具体免打扰级别信息，不包含频道信息。

IMKit SDK 未直接提供该方法，您需要使用 IMLib 中获取会话免打扰状态的方法。

```
NSArray *array = [[RCIMClient sharedRCIMClient] getBlockedConversationList:@[@(ConversationType_PRIVATE)]];
```

参数	类型	说明
conversationTypeList	NSArray	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

获取成功后，返回设置了屏蔽消息提醒的会话 RCConversation 的列表。

设置全局免打扰 设置全局免打扰

更新时间:2024-08-30

客户端可以设置通知静默时段，以实现全局免打扰的效果。该功能可设置一个免打扰时间窗口。在再次设置或删除用户免打扰时间段之前，当次设置的免打扰时间窗口会每日重复生效。

IMKit SDK 未直接提供该方法，您需要使用 IMLib 中的方法。详见 IMLib 文档设置全局免打扰 。

> 提示
>
> 推荐集成 IMKit 的客户直接使用 IMLib 的 RCChannelClient 中可同时配置「免打扰级别」的全局免打扰接口。

在经 SDK 设置的全局免打扰时段内：

- 如果客户端在后台运行时，会话中有新的消息，将不会进行通知提醒，可以收到消息内容。
- 如果客户端为离线状态，将不会收到远程通知提醒。
- 例外情况：@ 消息属于高优先级消息，不受全局免打扰逻辑控制，会始终进行通知提醒。

本地通知 本地通知

更新时间:2024-08-30

IMKit 已实现本地通知的创建、弹出行为，方便开发者快速构建应用。

什么是本地通知

本地通知指应用在前台或后台运行时，由 IMKit 或应用客户端直接调用系统接口创建并发送的通知。IMKit SDK 内部已经实现了本地通知功能，当应用处于后台接收到新消息时，IMKit 默认会在通知面板弹出通知提醒，即本地通知。

IMKit 的本地通知已支持以下场景：

- 当 App 刚进入后台时（仍处于后台活跃状态）：App 进入后台，SDK 的长连接最多存活 2 分钟，超时会主动断开。在进入后台 2 分钟内，IMKit 仍可通过长连接通道接收到新消息（撤回消息也会产生撤回信令消息）。接到新消息后，IMKit 默认会创建并弹出通知，即本地通知。

> 提示
>
> App 在后台进入非活跃状态后（例如，被系统杀死），IMKit 与服务端断开连接。如果应用已集成 APNs 推送服务，此时可通过接收推送通知。推送通知由系统直接创建并弹出，不属于本文所述的本地通知。

- 当 App 处于前台，且未打开任何会话页面时（未与任何人聊天），接收新消息后默认会响铃并震动，但不弹通知。

> 提示
>
> 如果需要在 App 处于后台且存活时使用 SDK 默认的本地通知提醒，您必须实现用户/群组/群名片信息提供者相关协议，并返回正确的用户信息或群组信息。详见用户信息。

设置前台铃声提示

当 App 处于前台时，默认会播放消息提示音，您可以通过将此属性设置为 YES，关闭所有的前台消息提示音。

```
RCKitConfigCenter.message.disableMessageAlertSound = YES;
```

关闭本地通知

IMKit 默认本地通知功能是开启的，如果需要关闭请参考：

```
RCKitConfigCenter.message.disableMessageNotificaiton = YES;
```

自定义本地通知

IMKit 在弹出本地通知前会触发消息接收监听器的回调方法。实现消息接收监听器的如下代理方法并返回 YES，SDK 针对此消息就不再弹默认的本地通知提示，您可以自行处理。

> 提示
>
> 如果 App 未实现用户/群组/群名片信息提供者，则 IMKit SDK 无法获取消息的用户/群组/群名片信息，无法触发该回调方法。

```
@protocol RCIMReceiveMessageDelegate <NSObject>
@optional
- (BOOL)onRCIMCustomLocalNotification:(RCMessage *)message withSenderName:(NSString *)senderName;

@end
```

参数	类型	描述
message	RCMessage	接收到的消息
senderName	NSString	消息发送者的用户名称

快速上手 快速上手

更新时间:2024-08-30

本教程是为了让新手快速了解融云即时通讯能力库（IMLib）。在本教程中，你可以体验集成 SDK 的基本流程和 IMLib 的基础通信能力。

前置条件

- 注册开发者账号。注册成功后，控制台会默认自动创建您的首个应用，默认生成开发环境下的 App Key，使用国内数据中心。
- 获取开发环境的应用 App Key。如不使用默认应用，请参考 如何创建应用，并获取对应环境 App Key 和 App Secret。

> 提示
>
> 每个应用具有两个不同的 App Key，分别对应开发环境与生产环境，两个环境之间数据隔离。在您的应用正式上线前，可切换到使用生产环境的 App Key，以便上线前进行测试和最终发布。

环境要求

名称	版本
Xcode	11 +
iOS	9.0 +
CocoaPods	1.10.0 +

SDK 5.1.1 及其以后要求使用 CocoaPods 1.10.0 +，具体请参见知识库文档。

导入 SDK

1. 在 podfile 中添加如下内容：

```
pod 'RongCloudIM/IMLib', '~> x.y.z'
```

> 提示
>
> x.y.z 代表具体版本，请在融云官网 SDK 下载页面或 CocoaPods 仓库查询最新版本。

2. 请在终端中运行以下命令：

```
pod install
```

> 提示
>
> 如果出现找不到相关版本的问题，可先执行 pod repo update，再执行 pod install。

3. 上一步完成后，CocoaPods 会在您的工程根目录下生成一个 xcworkspace 文件，只需通过 XCode 打开该文件即可加载工程。

初始化 SDK

融云即时通讯客户端 SDK 核心类为 RCCoreClient 和 RCIMClient。在初始化 SDK 时，需要传入生产或开发环境的 App Key。

导入 SDK 头文件。

```
#import <RongIMLib/RongIMLib.h>
```

如果 SDK 版本 ≧ 5.4.2，请使用以下初始化方法。

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = nil;

[[RCCoreClient sharedCoreClient] initWithAppKey:appKey option:initOption];
```

初始化配置（RCInitOption）中封装了区域码（RCAreaCode ），导航服务地址（naviServer）、文件服务地址（fileServer）、数据统计服务地址（statisticServer）配置。不作设置表示全部使用默认配置。SDK 默认连接北京数据中心。

如果 App Key 不属于中国（北京）数据中心，则必须传入有效的初始化配置。初始化详细说明参见初始化。

获取用户 Token

用户 Token 是与用户 ID 对应的身份验证令牌，是应用程序的用户在融云的唯一身份标识。应用客户端在使用融云即时通讯功能前必须与融云建立 IM 连接，连接时必须传入 Token。

在实际业务运行过程中，应用客户端需要通过应用的服务端调用 IM Server API 申请取得 Token。详见 Server API 文档 注册用户 。

在本教程中，为了快速体验和测试 SDK，我们将使用控制台「北极星」开发者工具箱，从 API 调试页面调用 获取 Token 接口，获取到 **userId** 为 1 的用户的 Token。提交后，可在返回正文中取得 Token 字符串。

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"code":200,"userId":"1","token":"gxld6GHx3t1eDxof1qtxxYrQcjkbhl1V@sgyu.cn.example.com;sgyu.cn.example.com"}
```

建立 IM 连接

使用以上步骤中获取的 Token，模拟 userId 为 1 的用户连接到融云服务器。

```
[[RCCoreClient sharedCoreClient] connectWithToken:@"融云 token" dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
} success:^(NSString *userId) {
//连接成功
} error:^(RCConnectErrorCode errorCode) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else {
//无法连接到 IM 服务器，请根据相应的错误码作出对应处理
}
}];
```

SDK 已实现自动重连机制，请参见连接。

调用完连接后，你可以设置连接状态监听来实时监听 IM 连接状态，以便 UI 上给用户以提示，提高体验。

监听消息

设置消息接收监听器，用于接收所有类型的实时或者离线消息。实现此功能需要开发者遵守 RCIMClientReceiveMessageDelegate 协议。

1. 在初始化之后，连接之前设置监听器的代理委托。

```
[[RCCoreClient sharedCoreClient] setReceiveMessageDelegate:self object:nil];
```

2. 实现代理方法

当 SDK 在接收到消息时，开发者可通过下面方法进行处理。 SDK 会通过此方法接收包含单聊、群聊、聊天室、系统类型的所有消息，开发者只需全局设置一次即可，多次设置会导致其他代理失效。

```
- (void)onReceived:(RCMessage *)message left:(int)nLeft object:(id)object {

}
```

发送消息

向 userId 为 2 的用户发送一条文本消息。下面示例中先构造了文本消息内容 *messageContent，随后构造了 RCMessage 消息实例，消息实例中指定了收件人的用户 ID 为 2，以及当前的会话类型为私聊 ConversationType_PRIVATE。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_PRIVATE
targetId:@"2"
direction:MessageDirection_SEND
content:messageContent];

[[RCCoreClient sharedCoreClient] sendMessage:message
pushContent:nil
pushData:nil
attached:^(RCMessage *message) {
// 消息已存入本地数据库
}
successBlock:^(RCMessage *successMessage) {
//成功
}
errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}];
```

导入 SDK 导入 SDK

更新时间:2024-08-30

融云支持使用 CocoaPods 添加远程依赖项、和手动导入 Framework 的方式将 IMLib SDK 集成到您的应用工程中。

> RongIMLib 是融云即时通讯的核心能力库，功能包括：消息，会话，推送，聊天室等，以下简称 IMLib
>
> - 如果您需要自行编写 UI 界面，可以使用 IMLib
> - 如果您需要会话列表和会话页面等 UI 功能，请转至 IMKit 的文档。

环境要求

名称	版本
Xcode	11 +
iOS	9.0 +
CocoaPods	1.10.0 +

> 如需安装 CocoaPods 环境，请参照 安装 CocoaPods 。

检查版本

在导入 SDK 前，可以前往融云官网 SDK 下载页面 确认当前最新版本号。

CocoaPods

融云支持使用 CocoaPods 导入并管理 IMLib SDK 。

> pod 版本必须为 1.10.0 或更新版本。具体请查看知识库文档：
>
> https://help.rongcloud.cn/t/topic/747

具体操作步骤如下：

1. 在 `podfile` 中添加如下内容：

```
pod 'RongCloudIM/IMLib', '~> x.y.z'
```

> 提示
>
> - `x.y.z` 代表具体版本，请在融云官网 SDK 下载页面 或 CocoaPods 仓库查询最新版本。

2. 请在终端中运行以下命令：

```
pod install
```

> 提示
>
> 如果出现找不到相关版本的问题，可先执行 pod repo update，再执行 pod install。

3. 上一步完成后，CocoaPods 会在您的工程根目录下生成一个 xcworkspace 文件，只需通过 XCode 打开该文件即可加载工程。

手动集成

1. 前往融云官网 SDK 下载页面，将 IMLib SDK 下载到本地。

2. 将下载下来的 SDK 导入您的项目中，所需的 SDK 如下：

	Framework	资源文件
IMLib	• RongIMLib.xcframework • RongIMLibCore.xcframework • RongChatRoom.xcframework • RongCustomerService.xcframework • RongLocation.xcframework（5.2.5 及之后版本：无需导入） • RongDiscussion.xcframework • RongPublicService.xcframework	• RCConfig.plist
Translation（可选）	• RongTranslation.xcframework	

> 提示
>
> 仅 5.2.2 及之后版本的 SDK 支持翻译插件（Translation），该插件暂仅适用于使用新加坡数据中心的应用。

3. 修改您的项目配置。在 General -> Frameworks,Libraries,and Embedded Binaries 中，将 IMLib SDK 所需的 Framework 全部改为 Embed & Sign。

初始化 初始化

更新时间:2024-08-30

在使用 SDK 其它功能前，必须先进行初始化。本文中将详细说明 IMLib SDK 初始化的方法。

首次使用融云的用户，我们建议先阅读 IMLib SDK 快速上手，以完成开发者账号注册等工作。

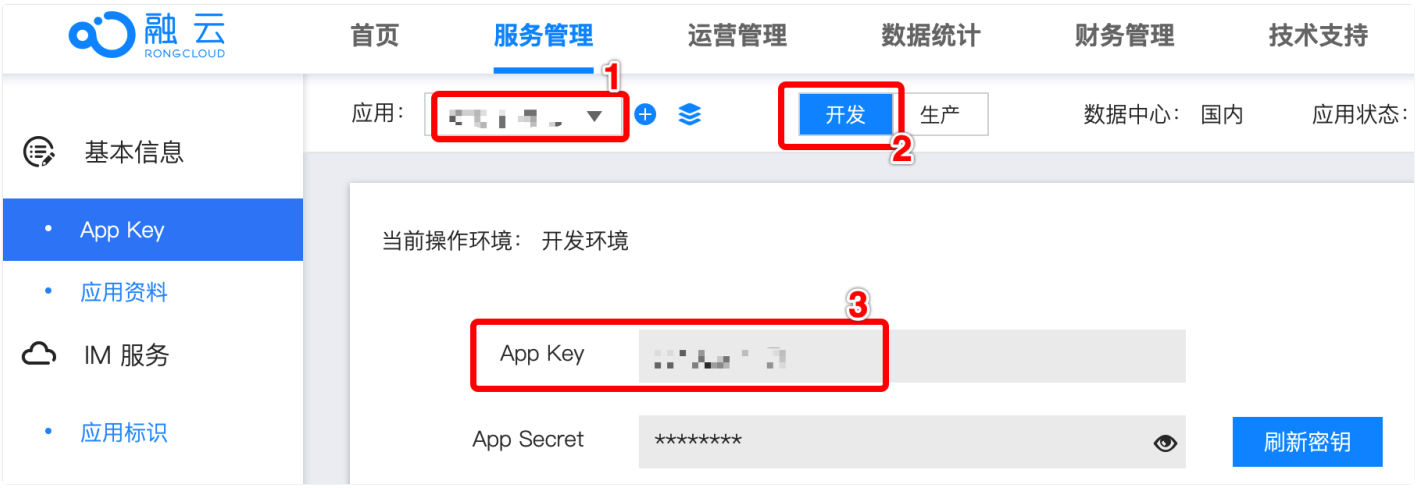
准备 App Key

您必须拥有正确的 App Key，才能进行初始化。

您可以控制台，查看您已创建的各个应用的 App Key。

如果您拥有多个应用，请注意选择应用名称（下图中标号 1）。另外，融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。在获取应用的 App Key 时，请注意区分环境（生产 / 开发，下图中标号 2）。

> 提示
>
> - 如果您并非应用创建者，我们建议在获取 App Key 时确认页面上显示的数据中心是否符合预期。
> - 如果您尚未向融云申请应用上线，仅可使用开发环境。



海外数据中心

- 如果您使用海外数据中心，且使用 5.4.2 及更新版本的开发版（dev）SDK，请注意在初始化配置中传入正确的区域码（RCAreaCode）。
- 如果您使用海外数据中心，且使用稳定版（stable） SDK，或使用早于 5.4.2 版本的开发版（dev）SDK，必须在初始化之前修改 IMLib SDK 连接的服务地址为海外数据中心地址。否则 SDK 默认连接中国国内数据中心服务地址。详细说明请参见配置海外数据中心服务地址。

初始化 SDK

> 提示
>
> 以下初始化方法要求 SDK 版本 ≧ 5.4.2。您可前往 融云官网 SDK 下载页面 查询最新开发版（Dev）和稳定版（Stable）的版本号。

导入 SDK 头文件

```
#import <RongIMLib/RongIMLib.h>
```

请在初始化方法中传入生产或开发环境的 App Key。

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = nil;

[[RCCoreClient sharedCoreClient] initWithAppKey:appKey option:initOption];
```

初始化配置（RCInitOption）中封装了区域码（RCAreaCode）配置。SDK 将通过区域码获取有效的导航服务地址、文件服务地址、数据统计服务地址、和日志服务地址等配置。

- 如果 App Key 属于中国（北京）数据中心，您无需配置额外配置 `RCInitOption`，SDK 将全部使用默认配置。
- 请务必在控制台核验当前 App Key 所属海外数据中心后，找到 RCAreaCode 中对应的枚举值进行配置。

例如，如果使用新加坡数据中心，配置如下：

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = [[RCInitOption alloc] init];
initOption.areaCode = RCAreaCodeSG;

[[RCCoreClient sharedCoreClient] initWithAppKey:appKey option:initOption];
```

除区域码外，初始化配置（InitOption）中还封装了以下配置：

- 导航服务地址（`naviServer`）：一般情况下不建议单独配置。SDK 内部默认使用与区域码对应的地址。
- 文件服务地址（`fileServer`）：仅限私有云使用。
- 数据统计服务地址（`statisticServer`）：一般情况下不建议单独配置。SDK 内部默认使用与区域码对应的地址。

```
NSString *appKey = @"Your_AppKey"; // example: bos9p5rlcm2ba
RCInitOption *initOption = [[RCInitOption alloc] init];
initOption.areaCode = RCAreaCodeSG;
initOption.naviServer = @"http(s)://naviServer";
initOption.fileServer = @"http(s)://fileServer";
initOption.statisticServer = @"http(s)://statsServer";
[[RCCoreClient sharedCoreClient] initWithAppKey:appKey option:initOption];
```

初始化 SDK

> 提示
>
> 如果您使用的开发版 SDK 版本号小于 5.4.2，或者稳定版 SDK 版本号小于等于 5.3.8，只能使用以下方式初始化。建议您及时升级 SDK 到最新版本。

导入 SDK 头文件

```
#import <RongIMLib/RongIMLib.h>
```

请在初始化 SDK 时传入生产或开发环境的 App Key。您可以使用 RCCoreClient 或 RCIMClient 的初始化方法。

```
[[RCIMClient sharedRCIMClient] initWithAppKey:@"融云 AppKey"];
```

监听连接状态　　监听连接状态

更新时间:2024-08-30

SDK 提供了 IM 连接状态变化委托协议 RCConnectionStatusChangeDelegate。通过监听 IM 连接状态的变化，App 可以进行不同业务处理，或在页面上给出提示。

连接状态变化委托协议

RCConnectionStatusChangeDelegate 协议定义如下：

```
@protocol RCConnectionStatusChangeDelegate <NSObject>

/*!
IMLib连接状态的监听器

@param status SDK与融云服务器的连接状态

@discussion 如果您设置了IMLib 连接监听之后，当SDK与融云服务器的连接状态发生变化时，会回调此方法。
*/
- (void)onConnectionStatusChanged:(RCConnectionStatus)status;
@end
```

> 提示
>
> 代理对象请设置给 AppDelegate 之类的单例对象，保证 APP 整个生命周期都可以监听到代理方法。

当连接状态发生变化时，SDK 会通过 - (void)onConnectionStatusChanged:(RCConnectionStatus)status; 方法，将当前的连接状态回调给开发者。RCConnectionStatus 中定义了连接过程中的可能的状态变化，下表具体说明了需要 App 处理的的状态码。

错误码	值	说明
ConnectionStatus_KICKED_OFFLINE_BY_OTHER_CLIENT	6	当前用户在其他设备上登录，此设备被踢下线
ConnectionStatus_Timeout	14	自动连接超时，SDK 将不会继续连接，用户需要做超时处理，再自行调用 connectWithToken 接口进行连接
ConnectionStatus_TOKEN_INCORRECT	15	token 无效，请从 APP 服务重新获取 token。原因有两种： 一是 token 错误，请检查客户端初始化使用的 AppKey 和服务器获取 token 使用的 AppKey 是否一致； 二是 token 过期，是因为在控制台设置了 token 过期时间，需要请求开发者的服务器重新获取 token 并再次用新的 token 建立连接。
ConnectionStatus_DISCONN_EXCEPTION	16	与服务器的连接已断开,用户被封禁

添加或移除代理委托

设置连接状态监听器，支持设置多个监听器。

为了避免内存泄露，请在不需要监听时，将设置的代理移除。

```
/// 添加代理委托
[[RCCoreClient sharedCoreClient] addConnectionStatusChangeDelegate:self];


/// 移除代理委托
[[RCCoreClient sharedCoreClient] removeConnectionStatusChangeDelegate:self];
```

建立连接 建立连接

更新时间:2024-08-30

应用客户端成功连接到融云服务器后，才能使用融云即时通讯 SDK 的收发消息功能。

融云服务端在收到客户端发起的连接请求后，会根据连接请求里携带的用户身份验证令牌（Token 参数），判断是否允许用户连接。

前置条件

- 通过服务端 API 注册用户（获取 Token）。融云客户端 SDK 不提供获取 Token 方法。应用程序可以调用自身服务端，从融云服务端获取 Token。
  - 取得 Token 后，客户端可以按需保存 Token，供后续连接时使用。具体保存位置取决于应用程序客户端设计。如果 Token 未失效，就不必再向融云请求 Token。
  - Token 有效期可在控制台进行配置，默认为永久有效。即使重新生成了一个新 Token，未过期的旧 Token 仍然有效。Token 失效后，需要重新获取 Token。如有需要，可以主动调用服务端 API 作废 Token。
- 建议应用程序在连接之前设置连接状态监听。
- SDK 已完成初始化。

> 提示
>
> **请不要在客户端直接调用服务端 API 获取 Token。获取 Token 需要提供应用的 App Key 和 App Secret。客户端如果保存这些凭证，一旦被反编译，会导致应用的 App Key 和 App Secret 泄露。所以，请务必确保在应用服务端获取 Token。**

连接聊天服务器

请根据应用的业务需求与设计，自行决定合适的时机（登陆、注册、或其他时机以免无法进入应用主页），向融云聊天服务器发起连接请求。

请注意以下几点：

- 必须在 SDK 初始化之后，调用 `connect` 方法进行连接。否则可能没有回调。
- SDK 内部实现了重连机制，在网络异常导致 IM 连接断开时，会自动重连，直到应用主动断开连接（见断开连接）。

接口（可设置超时）

客户端用户首次连接聊天服务器时，建议调用带连接超时时间（timeLimit）的接口，并设置超时秒数。在网络较差等导致连接超时的情况下，你可以利用接口的超时错误回调，并在 UI 上提醒用户，例如建议客户端用户等待网络正常的时候再次连接。

一旦连接成功，SDK 的重连机制将立即开始生效，并接管所有的重连处理。当因为网络原因断线时，SDK 会不停重连直到连接成功为止，不需要您做额外的连接操作。应用主动断开连接，将退出重连机制，详见断开连接。其他情况请参见下面重连机制详解。

如果免密登录，建议将 timeLimit 参数设置为 0，可以在 dbOpenedBlock 回调中进行页面跳转，优先展示本地历史数据。连接逻辑则完全托管给 SDK。

调用示例

```
[[RCIMClient sharedRCIMClient] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值"
timeLimit:5
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
} success:^(NSString *userId) {
//连接成功
} error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//Token 错误，可检查客户端 SDK 初始化与 App 服务端获取 Token 时所使用的 App Key 是否一致
} else if(status == RC_CONNECT_TIMEOUT) {
//连接超时，弹出提示，可以引导用户等待网络正常的时候再次点击进行连接
} else {
//无法连接 IM 服务器，请根据相应的错误码作出对应处理
}
}]
```

输入参数说明

参数	类型	说明
token	NSString	首次连接时必须由 App 服务端调用融云服务端 API 获取。参见服务端 API 文档 注册用户
timeLimit	int	SDK 连接超时时间，单位：秒
dbOpenedBlock	Block	本地消息数据库打开的回调
successBlock	Block	连接建立成功的回调
errorBlock	Block	连接建立失败的回调

- `timeLimit` 参数说明：

参数	取值范围	说明
timeLimit	<=0	SDK 会一直连接，直到连接成功或者出现 SDK 无法处理的错误（如 token 非法），等效于上面的连接接口。
timeLimit	>0	SDK 最多连接 timeLimit 秒，超时则返回 RC_CONNECT_TIMEOUT 错误，并不再重连。

返回参数说明

- `dbOpenedBlock` 说明：

回调参数	回调类型	说明
code	RCDBErrorCode	数据库是否已打开

- `successBlock` 说明：

回调参数	回调类型	说明
userId	NSString	当前连接成功的用户 ID

- `errorBlock` 说明：

回调参数	回调类型	说明
status	RCConnectErrorCode	连接失败的错误码。详见下文连接状态码。

接口（无超时设置）

如果客户端用户已成功登录过你的应用，且连接过融云聊天服务器，后续离线登录时建议使用此接口。

此时因用户已有历史数据，并不需要强依赖于连接成功。可以在 `dbOpenedBlock` 回调中进行页面跳转，优先展示本地历史数据。连接逻辑完全托管给 SDK 即可。

提示

当网络较差时，有可能长时间不会回调。

调用此接口后，SDK 的重连机制将立即开始生效，并接管所有的重连处理。SDK 会不停重连直到连接成功为止，不需要您做额外的连接操作。应用主动断开连接，将退出重连机制，详见断开连接。其他情况请参见下面重连机制详解。

调用示例

```
[[RCIMClient sharedRCIMClient] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值"
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
}
success:^(NSString *userId) {
//连接成功
}
error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else {
//无法连接到 IM 服务器，请根据相应的错误码作出对应处理
}
}]
```

输入参数说明

参数	类型	说明
token	NSString	需要您的 Server 访问融云服务获取 Token
dbOpenedBlock	Block	本地消息数据库打开的回调
successBlock	Block	连接建立成功的回调
errorBlock	Block	连接建立失败的回调

返回参数说明

- `dbOpenedBlock` 说明：

回调参数	回调类型	说明
code	RCDBErrorCode	数据库是否已打开

- `successBlock` 说明：

回调参数	回调类型	说明
userId	NSString	当前连接成功的用户 ID

- `errorBlock` 说明：

回调参数	回调类型	说明
status	RCConnectErrorCode	连接失败的错误码。详见下文连接状态码。

连接状态码

具体可以参考下表 `RCConnectErrorCode` 具体错误码的说明和处理方案建议

提示

关于 RC_CONN_TOKEN_INCORRECT 的说明：

1. 在 `RC_CONN_TOKEN_INCORRECT` 的情况下，您需要请求您的服务器重新获取 Token 并建立连接，但是注意避免无限循环，以免影响 App 用户体验。
2. 此方法的回调线程并非为原调用线程，需要进行 UI 操作时请注意切换到主线程。

错误码	值	说明	处理方案
RC_CONN_TOKEN_INCORRECT	31004	Token 错误。常见于客户端 SDK 与 App 服务器所使用的 App Key 不一致的错误情况。融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。由测试环境切换到生产环境时，很容易发生应用客户端与应用服务端 App Key 不一致的问题。	检查 SDK 与 App 服务器使用的 App Key 是否一致。
RC_CONN_TOKEN_EXPIRE	31020	Token 已过期。常见于控制台设置了 Token 过期时间。Token 过期之后需要由您的 App 服务器请求融云服务端重新获取 Token，并再次用新的 Token 建立连接	客户端需要请求 App 服务端，向融云获取新的 token。客户端收到新的 Token 后，使用新 Token 发起连接。
RC_DISCONN_KICK	31010	用户连接成功后被踢下线	退回到登录页面，给用户提示被踢掉线
RC_CONN_OTHER_DEVICE_LOGIN	31023	用户连接过程中又在其它设备上登录，导致当前设备被踢	退回到登录页面，给用户提示其他设备登录了当前账号
RC_CONN_USER_BLOCKED	31009	用户在线时被封禁导致连接断开。	退回到登录页面，给用户提示被封禁
RC_CONNECT_TIMEOUT	34006	自动重连超时。常见于无网络的环境，且仅发生在调用连接接口并设置了有效超时时间的情况。超时后 SDK 会放弃重连。开发者需要自行给用户提示，可提示用户等待网络恢复时再重新连接。	重新调用连接手动重连
RC_CONN_APP_BLOCKED_OR_DELETED	31008	Appkey 被封禁	请检查您使用的 AppKey 是否被封禁或已删除
RC_CONN_PROXY_UNAVAILABLE	31028	代理服务不可访问	
RC_CLIENT_NOT_INIT	33001	SDK 没有初始化	在使用 SDK 任何功能之前，必须先 Init
RC_INVALID_PARAMETER	33003	开发者接口调用时传入的参数错误	请检查接口调用时传入的参数类型和值
DATABASE_ERROR	33002	数据库错误	检查用户 userId 是否包含特殊字符，SDK userId 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式，最大长度 64 字节

断开连接　　断开连接

更新时间:2024-08-30

在 App 需要执行切换用户登录、注销登录的操作时，需要断开与融云的 IM 连接。SDK 支持设置断开 IM 连接之后是否允许向用户发送消息推送通知。

> 提示
>
> SDK 在前后台切换或者网络出现异常都会自动重连，会保证连接的可靠性。除非 App 逻辑需要登出，否则不需要手动断开连接。

断开连接（允许推送）

主动断开与融云服务端的 IM 连接，并设置断开连接后允许融云服务端进行远程推送。以下两种方式效果一致。

```
[[RCIMClient sharedRCIMClient] disconnect];
```

或者：

```
[[RCIMClient sharedRCIMClient] disconnect:YES];
```

参数	类型	说明
isReceivePush	BOOL	断开连接后，是否允许融云服务端进行远程推送。 YES 表示接收远程推送。NO 表示不接收远程推送。

如果融云服务端发现 App 客户端不在线（默认要求全部设备已下线），在接收新消息时，融云服务端会为该用户记录一条离线消息[?]，并触发融云服务端的推送服务。融云服务端会通过推送通道下发一条提醒到客户端 SDK。该提醒一般以通知形式展示在通知面板，提示用户有离线消息。

断开连接（不允许推送）

主动断开与融云服务端的 IM 连接，并设置断开连接后不允许融云服务端进行远程推送。

需要注销登录（登出）或切换 App 用户账号时，推荐使用以下任一方式，效果一致。

```
[[RCIMClient sharedRCIMClient] logout];
```

或者：

```
[[RCIMClient sharedRCIMClient] disconnect:NO];
```

参数	类型	说明
isReceivePush	BOOL	断开连接后，是否允许融云服务端进行远程推送。 YES 表示接收远程推送。NO 表示不接收远程推送。

断开连接且不允许推送的情况下，融云服务端仅记录离线消息，但不会为当前设备触发推送服务。如果用户登录了多个设备，则在其他设备中最后一个登录的设备上可正常接收推送。在多设备场景下，App 可能需要保证设备间消息记录一致，可通过开启多设备消息同步实现。详见多设备消息同步。

重连机制与重连互踢　　重连机制与重连互踢 自动重连机制

更新时间:2024-08-30

SDK 内已实现自动重连机制，一旦连接成功，SDK 的重连机制将立即开始生效，并接管所有的重连处理。当因为网络原因断线时，SDK 内部会尝试重新建立连接，不需要您做额外的连接操作。

可能导致 SDK 断线重连的异常情况如下：

- **弱网环境**：可能出现 SDK 不停重连的情况。因为客户端 SDK 和融云服务端之间存在连接保活机制，一旦因如果网络太差导致心跳超时，SDK 就会触发重连操作，尝试重连直到连接成功。
- **无网环境**：SDK 的重连机制会暂停。一旦网络恢复，SDK 会进行重连操作。

> 提示
>
> 一旦触发连接错误的回调，SDK 将退出重连机制。请根据具体的状态码自行处理。

重连时间间隔

SDK 尝试重连时，时间间隔逐次变大，分别是 0.05s（5.6.2 之前为 0s）, 0.25s, 0.5s, 1s, 2s, 4s, 8s, 16s, 32s。之后每 64s 重试一次。

当 APP 切换到前台或者网络状态发生变化，重连时间会按照上面的时间间隔从头开始，保证这种情况下能尽快的连接成功。

主动退出重连机制

应用主动断开连接后，SDK 将退出重连机制，不再尝试重连。

重连互踢策略

重连互踢策略用于控制 SDK 自动重连成功时是否需要下线的设备。

即时通讯业务默认仅允许同一用户账号在单台移动端设备上登录。后登录的移动端设备一旦连接成功，则自动踢出之前登录的设备。在部分情况下，SDK 的重连机制可能会导致后登录设备无法正常在线。

例如，默认的重连互踢策略可能导致以下情况：

1. 用户张三尝试在移动设备 A 上登录，但因 A 设备网络不稳定导致未连接成功，触发了 SDK 的自动重连机制。
2. 用户此时尝试换到移动设备 B 上登录。B 设备连接成功，且用户可通过 B 设备正常使用即时通讯业务。
3. A 设备网络稳定之后，SDK 重连成功。因此时 A 设备为后上线设备，导致 B 设备被踢出。

修改 App 用户的重连互踢策略

如果 App 用户希望在以上场景中将重连成功的 A 设备下线，同时保持 B 设备登录，可以修改当前用户（User ID）的重连互踢策略。

> 提示
>
> 使用该接口要求该 App Key 已启用登录时判断用户其他端登录状态提示（移动端）。如需启用该服务，请提交工单。

设置断线重连时是否踢出重连设备。此方法需要在 **init** 之前调用。

```
[[RCIMClient sharedRCIMClient] setReconnectKickEnable:YES];
```

参数	类型	说明
enable	boolean	是否踢出当前正在重连的设备。详见下方 enable 参数详细说明。

enable 参数详细说明：

- 设置 setReconnectKickEnable 为 YES

  如果重连时发现已有别的移动端设备在线，将不再重连，不影响已正常登录的移动端设备。

- 设置 setReconnectKickEnable 为 NO

  如果重连时发现已有别的移动端设备在线，将踢出已在线的移动端设备，使当前设备上线。

多端同时在线

更新时间:2024-08-30

多端同时在线是指同一用户账号从多个平台同时连接到融云即时通讯服务的功能。默认情况下，融云即支持多端设备同时在线。该功能无需开通即可使用。

> 默认多端设备之间不会进行消息同步。如有需要，请开通多设备消息同步服务。

多端登录限制说明

默认的情况下，同一用户账号可在移动端、Web 端、桌面端、小程序端最多一个设备上同时在线。

平台类别	限制	IM SDK 支持平台列表
移动端	默认仅支持一个移动端设备连接。如需支持移动端多设备登录，请提交工单申请开通移动多端服务。	Android、iOS、Flutter、React Native、uni-app、Unity
桌面端	默认仅支持一个桌面端设备连接。	Electron 框架（通过 Web 端 SDK 的 Electron 模块支持）
Web 端	默认仅支持一个 Web 页面连接（每个浏览器标签页认为是一个连接）。在控制台自助开通多设备消息同步服务后，自动支持多 Web 页面连接。	Web
小程序端	默认仅支持一个小程序连接。如需支持小程序多设备登录，请提交工单申请开通小程序多端服务。	小程序

多设备消息同步

更新时间:2024-08-30

在即时通讯业务中，同一用户账号可能在多个设备上登录。**多设备消息同步**是融云服务端提供的一项服务，可用于同一用户账号的多个设备之间同步收发消息。

默认情况下，融云不会在设备之间同步消息。新消息被某一端设备收取后，其他端无法收取该消息。

适用场景

在融云即时通讯业务中，多设备消息同步适用于以下情况：

- 同一用户账号在多设备上同时在线（无论是否为同一端），希望同步收发消息。例如，用户可能拥有多个移动端设备，如两个 Android 设备、一个 iOS 设备。

  > 注意：融云默认已支持多端同时在线，同一用户账号可在移动端、Web 端、桌面端、小程序端最多一个设备上同时在线。但是如果需要允许 App 用户同时在多个移动端设备或多个小程序端上在线，需要分别提交工单申请，详见多端同时在线。

- 同一用户账号换设备登录（无论是否曾在该设备登录过），希望同步收发的消息记录。例如用户从 Android 设备下线后，换到另一个设备从 Web 端登录。
- 同一用户账号在当前设备卸载重装 App，希望同步收发消息记录。

支持在多设备间同步的消息

并非所有消息均支持多设备消息同步。**状态消息**仅支持在多设备同时在线时同步接收，不在线的设备无法通过**多设备消息同步**收到消息。

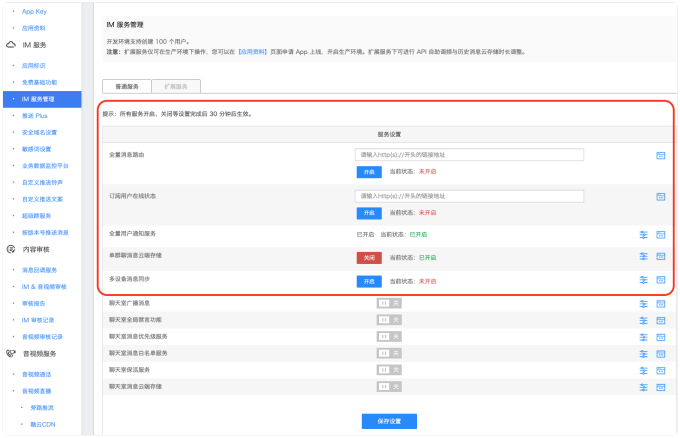
以下情况均属于状态消息：

- 融云内置消息类型中定义为状态消息类型的消息。内置状态类型消息的具体包括：正在输入状态消息（`RC:TypSts`）
- 自定义的状态消息类型的消息。详见各个客户端「自定义消息」文档。
- 使用服务端 API 状态消息接口发送的所有消息（不区分消息类型）均不支持同步。具体的 API 接口为发送单聊状态消息（`/statusmessage/private/publish.json`）、发送群聊状态消息（`/message/group/publish.json`）。

开通服务

请前往控制台，在 IM 服务管理 页面的**普通服务**标签下开通**多设备消息同步**服务。该服务在开发环境免费使用，默认为关闭状态。生产环境预存费用后才可开通服务。

> 服务开启、关闭设置完成后 30 分钟内生效。



对其他功能或业务的影响

多设备消息同步服务的状态对即时通讯业务中的离线补偿[?]、撤回消息、聊天室业务等有影响。

对离线补偿的影响

控制台的多设备消息同步服务包含了融云服务端离线补偿机制[?]的开关。

开通多设备消息同步服务后，融云服务端自动为 App 启用离线补偿机制。离线补偿机制会在以下场景触发：

- 换设备登录。用户在新设备上登录后（无论是否曾在该设备登录过），SDK 可获取最近指定天数（默认离线补偿天数为 1 个自然日）在其他终端上发送和接收过的消息。
- 应用卸载重装。消息与会话列表是存储在本地数据库，用户卸载 App 时会删除本地数据库。用户重新安装 App 后并再次连接时，会触发融云服务端离线补偿机制，SDK 可获取最近指定天数（默认离线补偿天数为 1 个自然日）在其他终端上发送和接收过的消息。

> 注意：在 换设备登录 或 应用卸载重装 场景下，离线补偿机制仅可获取到最近（默认离线补偿天数为 1 天，最大 7 天）的会话。早于该天数的会话无法通过离线补偿机制获取。因此，离线补偿后的会话列表可能与原设备上或卸载前的会话列表并不一致（您可能会有丢失部分会话的错觉）。

如需修改离线消息补偿的天数，可提交工单。建议谨慎设置离线补偿天数，当单用户消息量超大时，可能会因为补偿消息量过大，造成端上处理压力较大。

对 Web 平台连接数的影响

开通多设备消息同步服务后，可额外支持多 Web 页面连接（每个浏览器标签页也认为是一个连接），最多 10 个。

对撤回消息的影响

- 未开通多设备消息同步服务时，多端之间无法同步撤回的消息。
- 开通多设备消息同步服务后，消息发送端一旦撤回消息，如果用户在其他端在线，则其他端同步撤回该条已发送消息。如果用户在其他端不在线时，则在用户登录后同步撤回已发送的消息。

对服务端 API 发送消息的影响

通过服务端 API 发送消息时，部分接口可通过 `isIncludeSender` 指定消息发送者可否在客户端接收该已发消息。

- 未开通多设备消息同步服务时，仅在发送者已登录客户端（在线）的情况下，通过 Server API 发送的消息可即时同步到发送者的在线客户端，无法同步到离线的客户端。
- 开通多设备消息同步服务后，如果发送者未登录客户端（离线），通过 Server API 发送的消息可在再次上线时同步到发送者的在线客户端。

用户概述 用户概述

更新时间:2024-08-30

App 用户需要接入融云服务，才能使用即时通讯服务。对于融云来说，用户是指持有由融云分发的有效 Token，接入并使用即时通讯服务的 App 用户。

注册用户

应用服务端（App Server）应向融云服务端提供 App 用户的用户 ID（`userId`），以向融云换取唯一用户 Token。对融云来说，这个以 `userId` 获取 Token 的步骤即注册用户，且必须通过调用 Server API 来完成。

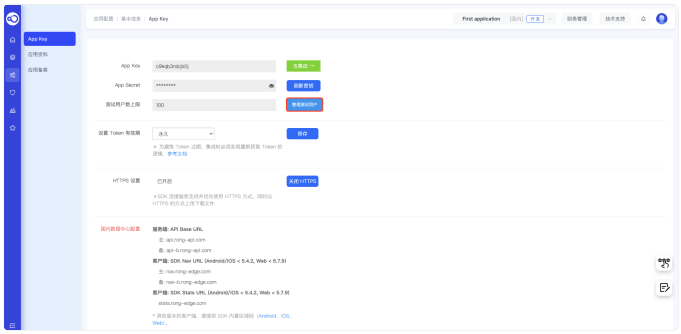
应用客户端必须持有有效 Token，才能成功连接到融云服务端，使用融云即时通讯服务。当 App 客户端用户向服务器发送登录请求时，服务器会查询数据库以检查连接请求是否匹配。

注册用户数限制

- 开发环境[?]中的注册用户数上限为 100 个。
- 在 生产环境[?]中，升级为 **IM 旗舰版**或 **IM 尊享版**后不限制注册用户数。

删除用户

删除用户是指在应用的开发环境中，通过控制台删除已注册的测试用户，以控制开发环境中的测试用户总数。生产环境不支持该操作。



注销用户

注销用户是指在融云服务中删除用户数据。App 可使用该能力实现自身的用户销户功能，满足 App 上架或合规要求。

融云返回注销成功结果后，与用户 ID 相关数据即被删除。您可以向融云查询所有已注销用户的 ID。如有需要，您可以重新激活已被注销的用户 ID（注意，用户个人数据无法被恢复）。

仅 IM Server API 提供上述能力。

用户信息

用户信息泛指用户的昵称、头像，以及群组的群昵称、群头像等数据。融云服务端不提供用户信息托管维护服务。

IMLib SDK 中未提供与用户信息相关的客户端接口。

好友关系

App 用户之间的好友关系需要由应用服务器（App Server）自行维护。融云不会同步或保存 App 端的好友关系数据。

如果需要对客户端用户之间的消息收发行为进行限制（例如，App 的所有 userId 泄漏，导致某个恶意用户可越过好友关系向任意用户发送消息），可以考虑使用用户白名单服务。用户一旦开启并设置白名单，则仅可接收来自该白名单中用户的消息。

用户管理接口

功能分类	功能描述	客户端 API	服务端 API
注册用户	使用 App 用户的用户 ID 向融云换取 Token。	不提供该 API	注册用户
删除用户	参见上文删除用户。	不提供该 API	不提供该 API
废弃 Token	废弃在特定时间点之前获取的 Token。	不提供该 API	作废 Token
注销用户	注销用户是指在融云服务中停用用户 ID，并删除用户个人数据。	不提供该 API	注销用户
查询已注销用户	获取已注销的用户 ID 列表。	不提供该 API	查询已注销用户
重新激活用户 ID	在融云服务中重新启用已注销用户的 ID。	不提供该 API	重新激活用户 ID
设置融云服务端的用户信息	设置在融云推送服务中使用的用户名称与头像。	不提供该 API	未提供单独的设置接口。在注册用户时必须提供用户信息。
获取融云服务端的用户信息	获取用户在融云注册的信息，包括用户创建时间和服务端的推送服务使用的用户名称、头像 URL。	不提供该 API	获取信息
修改融云服务端的用户信息	修改在融云推送服务中使用的用户名称与头像。	不提供该 API	修改信息
封禁用户	禁止用户连接到融云即时通讯服务，并立即断开连接。可按时长解封或主动解封。查询被封禁用户的用户 ID、封禁结束时间。	不提供该 API	添加封禁用户、解除封禁用户、查询封禁用户
查询用户在线状态	查询某用户的在线状态。	不提供该 API	查询在线状态
加入黑名单	在用户的黑名单列表中添加用户。在 A 用户黑名单的用户无法向 A 发送消息。	加入黑名单	加入黑名单
移出黑名单	在用户的黑名单中移除用户。	移出黑名单	移出黑名单
查询黑名单	查询某用户（userId）是否已被当前用户加入黑名单。	查询用户是否在黑名单中	不提供该 API
获取黑名单列表	获取用户的黑名单列表。	获取黑名单列表	查询黑名单
用户白名单	用户一旦开启并设置白名单，则仅可接收来自该白名单中用户的消息。	不提供该 API	开启用户白名单、用户白名单状态查询、添加白名单、移出白名单、查询白名单

用户黑名单

更新时间:2024-08-30

将用户加入黑名单之后，将不再收到对方发来的任何单聊消息。

- 加入黑名单为单向操作，例如：用户 A 拉黑用户 B，代表 B 无法给 A 发消息（错误码 `405`）。但 A 向 B 发消息，B 仍然能正常接收。
- 单个用户的黑名单总人数存在上限，具体与计费套餐有关。IM 旗舰版与 IM 尊享版上限为 3000 人，其他套餐详见功能对照表中的服务限制。
- 调用服务端 API 发送单聊消息默认不受黑名单限制。如需启用限制，请在调用 API 时设置 `verifyBlacklist` 为 `1`。

加入黑名单

将指定用户（userId）加入当前用户的黑名单。操作成功后，当前用户仍可向被拉黑的用户发送消息，但被拉黑的用户无法给当前用户发送单聊消息。

```
[[RCIMClient sharedRCIMClient] addToBlacklist:@"userId" success:^{

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
userId	NSString	用户 id
successBlock	Block	添加成功的回调
errorBlock	Block	添加失败的回调。`status` 中返回错误码 RCErrorCode。

移出黑名单

将指定用户（userId）从当前用户的黑名单中移出。

```
[[RCIMClient sharedRCIMClient] removeFromBlacklist:@"userId" success:^{

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
userId	NSString	用户 id
successBlock	Block	移除成功的回调
errorBlock	Block	移除失败的回调。`status` 中返回错误码 RCErrorCode。

查询用户是否在黑名单中

根据用户 ID（userId）查询指定用户是否在当前用户的黑名单中。

```
[[RCIMClient sharedRCIMClient] getBlacklistStatus:@"userId" success:^(int bizStatus) {

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
userId	NSString	用户 id
successBlock	Block	查询成功的回调。`bizStatus` 表示该用户是否在黑名单中。0 表示已经在黑名单中，101 表示不在黑名单中。
errorBlock	Block	查询失败的回调。`status` 中返回错误码 RCErrorCode 。

查询黑名单列表

获取当前用户的黑名单列表。

```
[[RCIMClient sharedRCIMClient] getBlacklist:^(NSArray *blockUserIds) {

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
successBlock	Block	查询成功的回调。`blockUserIds` 返回 `NSArray` 类型数据，为已经设置的黑名单中的用户 ID 列表
errorBlock	Block	查询失败的回调。`status` 中返回错误码 RCErrorCode 。

订阅用户在线状态

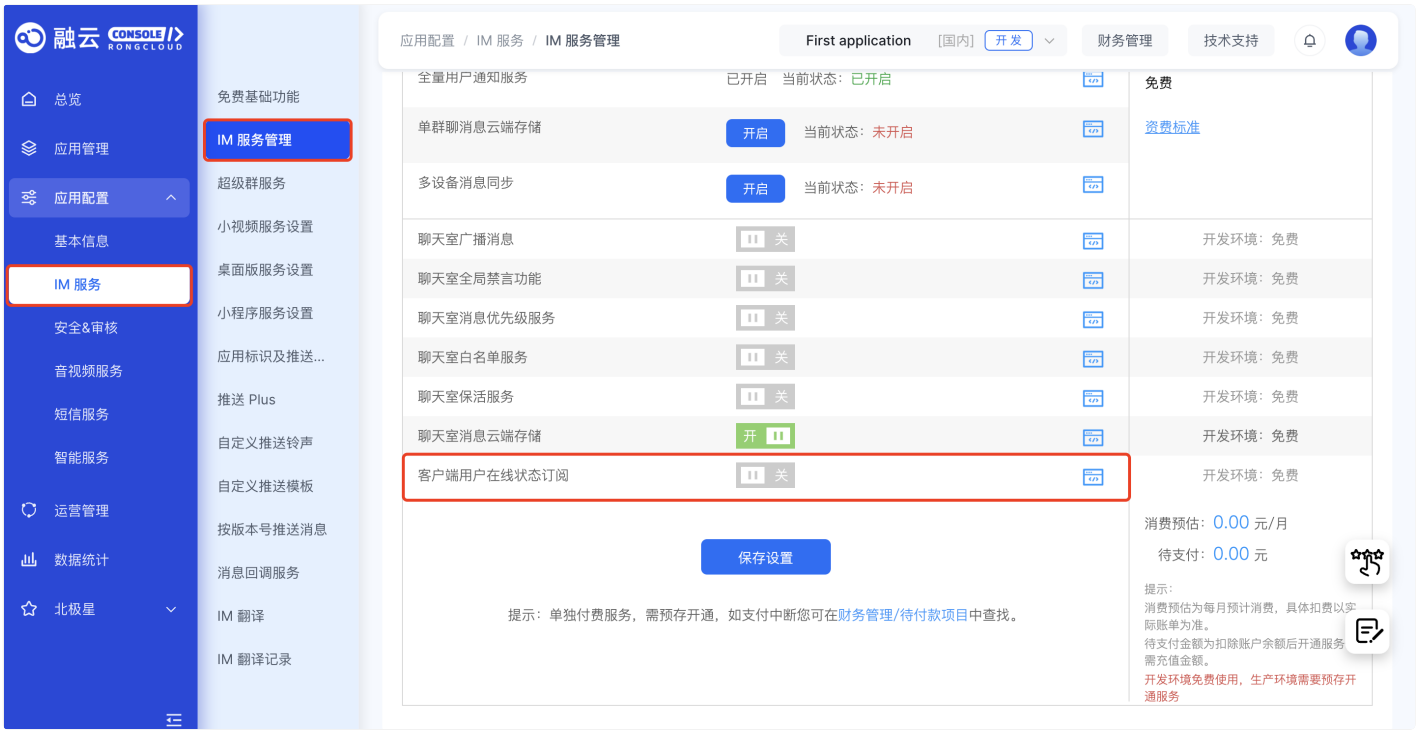
更新时间:2024-08-30

本文档旨在指导开发者如何在融云即时通讯 iOS 客户端 SDK 中实现用户在线状态的订阅、查询和监听。通过本文档，开发者将了解如何获取和跟踪用户在线状态，以及如何在状态变更时接收通知。

> 提示
>
> 此功能在 5.8.1 版本开始支持。

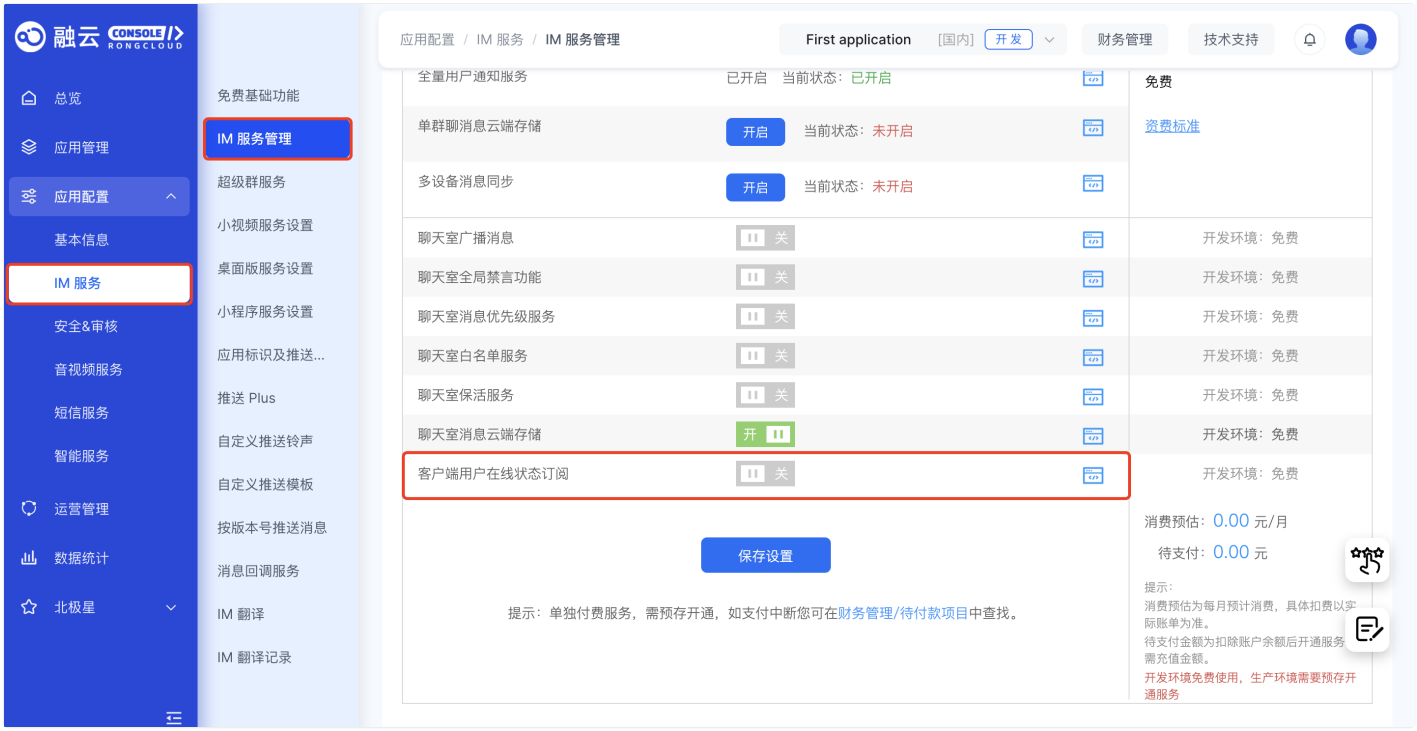
开通服务

您可以通过控制台开通服务。在融云控制台，选择 **IM 服务 > IM 服务管理 > 客户端用户在线状态订阅** ，开启服务。



开通服务

您可以通过控制台开通服务。在融云控制台，选择 **IM 服务 > IM 服务管理 > 客户端用户在线状态订阅** ，开启服务。



订阅用户在线状态

1. 导入 SDK 头文件。

```
#import <RongIMLibCore/RongIMLibCore.h>
```

2. 创建 `RCSubscribeEventRequest` 对象，设置订阅类型为 `RCSubscribeTypeOnlineStatus`，订阅用户 ID 列表和订阅时长。一次订阅用户的上限为 200 个，订阅的用户数最多为 1000 个，一个用户最多可以被 5000 个用户订阅。

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
request.subscribeType = RCSubscribeTypeOnlineStatus;
request.userIds = @[@"test1",@"test2",@"test3"];
request.expiry = 180000;

3. 执行订阅操作。

[[RCCoreClient sharedCoreClient] subscribeEvent:request
completion:^(RCErrorCode status, NSArray<NSString *> * _Nullable failedUserIds) {
if (RC_SUCCESS == status) {
// 订阅成功。
}
}];
```

取消订阅用户在线状态

当您不再需要跟踪用户的在线状态时，可以使用 `unSubscribeEvent:completion:` 方法取消订阅。

示例代码

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
request.subscribeType = RCSubscribeTypeOnlineStatus;
request.userIds = @[@"test1",@"test2",@"test3"];

[[RCCoreClient sharedCoreClient] unSubscribeEvent:request
completion:^(RCErrorCode status, NSArray<NSString *> * _Nullable failedUserIds) {
if (RC_SUCCESS == status) {
// 取消订阅成功。
}
}];
```

查询订阅状态信息

您可以使用querySubscribeEvent:completion:方法查询指定用户和订阅类型的状态信息。

示例代码

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
request.subscribeType = RCSubscribeTypeOnlineStatus;
request.userIds = @[@"test1",@"test2",@"test3"];

[[RCCoreClient sharedCoreClient] querySubscribeEvent:request
completion:^(RCErrorCode status, NSArray<RCSubscribeInfoEvent *> * _Nullable subscribeEvents) {
}];
```

分页查询已订阅用户的状态信息

如果您需要分页获取已订阅的所有事件状态信息，可以使用 `querySubscribeEvent:pageSize:startIndex:completion:`方法, 并指定分页大小和起始索引。

- `pageSize`: 分页大小，取值范围为[1~200]。
- `startIndex`: 起始索引。第一页传 0， 下一页取返回所有数据的数组数量（比如 pageSize = 20，第二页传 20，第三页传40）。

示例代码

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
request.subscribeType = RCSubscribeTypeOnlineStatus;

[[RCCoreClient sharedCoreClient] querySubscribeEvent:request
pageSize:2
startIndex:index
completion:^(RCErrorCode status, NSArray<RCSubscribeInfoEvent *> * _Nullable subscribeEvents) {
if (RC_SUCCESS == status && subscribeEvents.count > 0) {
}
}];
```

监听订阅事件

为了接收订阅事件的变更通知,您需要设置订阅监听器。实现此功能需要开发者遵守 `RCSubscribeEventDelegate` 协议。

> 提示
>
> 从 5.10.0 版本开始，订阅事件包含了用户在线状态和用户信息托管两种种类型，订阅事件的变更，需要根据 RCSubscribeType 的类型来处理对应的业务。

在应用初始化之后，建立连接之前，您需要设置监听器的代理委托。

```objectivec
[[RCCoreClient sharedCoreClient] addSubscribeEventDelegate:self];
[[RCCoreClient sharedCoreClient] removeSubscribeEventDelegate:self];
```

实现代理：

1. 当订阅事件发生变化时，以下方法将被调用。您可以通过实现此方法来处理事件的变化，例如更新用户界面或处理新的数据。通常，该方法在后台线程中被调用，因此更新UI时需要切换到主线程。subscribeEvents 是订阅事件的列表，包含所有发生变化的事件。
   请注意，订阅过期没有通知， 您需自行关注过期时间。

```objectivec
/// 注意：从 5.10.0 版本开始，需要判断 RCSubscribeInfoEvent 的 subscribeType，等于 RCSubscribeTypeOnlineStatus 时代表在线状态订阅事件。
- (void)onEventChange:(NSArray<RCSubscribeInfoEvent *> *)subscribeEvents;
```

2. 订阅数据同步完成。当订阅数据成功同步到本地设备或系统后，将调用此方法来执行后续业务逻辑。

```objectivec
/// 已废弃，请使用 onSubscriptionSyncCompleted: 代替
- (void)onSubscriptionSyncCompleted;

/// 订阅数据同步完成
/// 使用 type 区分不同的订阅类型，等于 RCSubscribeTypeOnlineStatus 时代表在线状态订阅事件。
/// - Since: 5.10.0
- (void)onSubscriptionSyncCompleted:(RCSubscribeType)type;
```

3. 远程设备订阅信息变更。当用户在其他设备上的订阅信息发生变更时，此方法将被调用。您可以利用这个方法来更新当前设备上的状态信息，以保证订阅信息的一致性。 subscribeEvents 包含发生变化的订阅事件的列表。

```objectivec
/// 从 5.10.0 版本开始，需要判断 RCSubscribeEvent 的 subscribeType，等于 RCSubscribeTypeOnlineStatus 时代表在线状态订阅事件。
- (void)onSubscriptionChangedOnOtherDevices:(NSArray<RCSubscribeEvent *> *)subscribeEvents;
```

用户信息托管

更新时间:2024-08-30

本文档旨在指导开发者如何在融云即时通讯 iOS 客户端 SDK 中实现用户信息订阅、查询和监听，同时支持用户信息与权限的修改、查询。通过本文档，iOS 开发者将了解如何获取和跟踪用户信息，以及如何在用户信息变更、订阅状态变更时接收通知。

> 提示
>
> 此功能在 5.10.0 版本开始支持。

开通服务

使用此功能前，您须在控制台开通信息托管服务。

用户信息管理

SDK 提供了用户信息修改、查询、订阅的相关接口。但客户端 SDK 仅支持修改自己的用户信息。

用户信息设置

使用 `updateMyUserProfile:success:error:` 接口可以修改自己的用户信息。

RCUserProfile 属性介绍：

属性名	类型	描述
name	NSString	昵称，长度不超过 32 个字符。
portraitUri	NSString	头像地址，长度不超过 128 个字符。
uniqueId	NSString	用户应用号，支持大小写字母、数字，长度不超过 32 个字符。请注意 SDK 不支持设置此字段。
email	NSString	Email，长度不超过 128 个字符。

属性名	类型	描述
birthday	NSString	生日，长度不超过 32 个字符
gender	RCUserGender	性别
location	NSString	所在地，长度不超过 32 个字符。
role	NSUInteger	角色，支持 0~100 以内数字。
level	NSUInteger	级别，支持 0~100 以内数字。
userExtProfile	NSDictionary<NSString *, NSString *>	自定义扩展信息，默认最多可以设置 20 个用户信息（以 Key、Value 方式设置，扩展用户信息的 Key 需通过开发者后台进行设置） 1. Key：支持大小写字母、数字，长度不超过 32 个字符，需要保障 AppKey 下唯一。Key 不存在时，设置不成功返回错识提示。 2. Value：字符串，不超过 256 个字符。 。

代码示例

```
RCUserProfile *userProfile = [[RCUserProfile alloc] init];
userProfile.name = @"name";

[[RCCoreClient sharedCoreClient] updateMyUserProfile:userProfile success:^{
// 更新成功
} error:^(RCErrorCode errorCode, NSString * _Nullable errorKey) {
// 更新失败
}];
```

获取当前用户信息

您可以使用 getMyUserProfile:error: 方法获取当前用户信息。

代码示例

```
[[RCCoreClient sharedCoreClient] getMyUserProfile:^(RCUserProfile *userProfile) {
// 获取成功
} error:^(RCErrorCode errorCode) {
// 获取失败
}];
```

批量获取他人用户信息

您可以使用 getUserProfiles:success:error: 方法查询指定用户的用户信息。一次最多查询 20个 用户的用户信息。

代码示例

```
NSArray *userIds = @[@"userId1", @"userId2"];
[[RCCoreClient sharedCoreClient] getUserProfiles:userIds success:^(NSArray<RCUserProfile *> *userProfiles) {
// 获取成功
} error:^(RCErrorCode errorCode) {
// 获取失败
}];
```

用户权限

客户端 SDK 提供了用户权限的设置和获取接口，通过 RCUserProfileVisibility 枚举来表示用户权限。

RCUserProfileVisibility 枚举介绍

枚举值	用户权限
RCUserProfileVisibilityNotSet	未设置：以 AppKey 权限设置为准，默认是此状态。
RCUserProfileVisibilityInvisible	都不可见：任何人都不能搜索到我的用户信息，名称、头像除外。
RCUserProfileVisibilityEveryone	所有人：应用中任何用户都可以查看到我的用户信息。

用户权限设置

您可以使用 updateMyUserProfileVisibility:success:error 方法设置当前用户的用户信息访问权限。

代码示例

```
[[RCCoreClient sharedCoreClient] updateMyUserProfileVisibility:RCUserProfileVisibilityEveryone success:^{
// 修改成功
} error:^(RCErrorCode errorCode) {
// 修改失败
}];
```

用户权限获取

您可以使用 getMyUserProfileVisibility: 方法获取当前用户的用户信息访问权限。

代码示例

```
[[RCCoreClient sharedCoreClient] getMyUserProfileVisibility:^(RCUserProfileVisibility visibility) {
// 获取成功
} error:^(RCErrorCode errorCode) {
// 获取失败
}];
```

用户权限说明

AppKey 默认用户信息访问权限为 **都不可见**，用户级别默认为 **未设置**。都为默认值时， SDK 仅可查看他人的用户名和头像信息。

下面列举了 AppKey 级 权限与 用户级 权限综合说明：

AppKey 级权限	用户级权限	结果
都不可见、仅好友可见、所有人可见	未设置	以 AppKey 级权限设置为准
都不可见、仅好友可见、所有人可见	设置为（都不可见、所有人可见）	以用户级权限设置为准

用户搜索

按用户应用号精确搜索

支持按用户应用号搜索用户信息，可以使用 searchUserProfileByUniqueId:success:error 方法

```
// 使用应用号查询用户信息
[[RCCoreClient sharedCoreClient] searchUserProfileByUniqueId:@"uniqueId" success:^(RCUserProfile * _Nonnull userProfile) {
// 查询成功
} error:^(RCErrorCode errorCode) {
// 查询失败
}];
```

用户信息&权限变更订阅

订阅用户信息&权限变更

当您需要跟踪用户信息和权限的变更时，可以使用 subscribeEvent:completion: 方法订阅用户信息变更事件。

1. 导入 SDK 头文件。

```
#import <RongIMLibCore/RongIMLibCore.h>
```

2. 创建 `RCSubscribeEventRequest` 对象，设置订阅类型为 `RCSubscribeTypeUserProfile`，订阅用户 ID 列表和订阅时长。一次订阅用户的上限为 200 个，订阅的用户数最多为 1000 个，一个用户最多可以被 5000 个用户订阅。

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
// 订阅的用户列表
request.userIds = @[@"userId1", @"userId2"];
// 本次订阅的类型
request.subscribeType = RCSubscribeTypeUserProfile;
// 订阅时长，单位秒
request.expiry = 180000;

[[RCCoreClient sharedCoreClient] subscribeEvent:request completion:^(RCErrorCode status, NSArray<NSString *> * _Nullable
failedUserIds) {
if (RC_SUCCESS == status) {
// 订阅成功
}
}];
```

取消订阅用户信息&权限变更

当您不再需要跟踪用户信息和权限的变更时，可以使用 `unSubscribeEvent:completion:` 方法取消订阅。

示例代码

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
request.subscribeType = RCSubscribeTypeUserProfile;
request.userIds = @[@"userId1", @"userId2"];

[[RCCoreClient sharedCoreClient] unSubscribeEvent:request completion:^(RCErrorCode status, NSArray<NSString *> * _Nullable
failedUserIds) {
if (RC_SUCCESS == status) {
// 取消订阅成功
}
}];
```

查询订阅用户信息&权限变更状态

您可以使用 `querySubscribeEvent:completion:` 方法查询指定用户和订阅类型的状态信息。

示例代码

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
request.subscribeType = RCSubscribeTypeUserProfile;
request.userIds = @[@"userId1", @"userId2"];

[[RCCoreClient sharedCoreClient] querySubscribeEvent:request completion:^(RCErrorCode status, NSArray<RCSubscribeInfoEvent
*> * _Nullable subscribeEvents) {
if (RC_SUCCESS == status) {
// 查询成功
}
}];
```

分页查询订阅用户信息&权限变更状态

如果您需要分页获取已订阅的所有事件状态信息，可以使用 querySubscribeEvent:pageSize:startIndex:completion:方法，并指定分页大小和起始索引。

- pageSize：分页大小，最大值为 100。
- startIndex：起始索引。第一页传 0， 下一页取返回所有数据的数组数量（比如 pageSize = 20，第二页传 20，第三页传40）。

示例代码

```
RCSubscribeEventRequest *request = [[RCSubscribeEventRequest alloc] init];
request.subscribeType = RCSubscribeTypeUserProfile;
request.userIds = @[@"userId1", @"userId2"];

[[RCCoreClient sharedCoreClient] querySubscribeEvent:request
pageSize:20
startIndex:0
completion:^(RCErrorCode status, NSArray<RCSubscribeInfoEvent *> * _Nullable subscribeEvents) {
if (RC_SUCCESS == status && subscribeEvents.count > 0) {
// 查询成功
}
}];
```

监听订阅事件

为了接收订阅事件的变更通知,您需要设置订阅监听器。实现此功能需要开发者遵守 RCSubscribeEventDelegate 协议。

在应用初始化之后，建立连接之前，您需要设置监听器的代理委托。

```
// 添加监听代理
[[RCCoreClient sharedCoreClient] addSubscribeEventDelegate:self];

// 移除监听代理
[[RCCoreClient sharedCoreClient] removeSubscribeEventDelegate:self];
```

实现代理：

1. 当订阅事件发生变化时，以下方法将被调用。您可以通过实现此方法来处理事件的变化，例如更新用户界面或处理新的数据。通常，该方法在后台线程中被调用，因此更新UI时需要切换到主线程。subscribeEvents 是订阅事件的列表，包含所有发生变化的事件。
   请注意，订阅过期没有通知， 您需自行关注过期时间。

   ```
   // 需要判断 RCSubscribeInfoEvent 的 subscribeType，等于 RCSubscribeTypeUserProfile 时代表用户信息订阅事件。
   - (void)onEventChange:(NSArray<RCSubscribeInfoEvent *> *)subscribeEvents;
   ```

2. 订阅数据同步完成。当订阅数据成功同步到本地设备或系统后，将调用此方法来执行后续业务逻辑。

   ```
   /// 订阅数据同步完成
   /// 使用 type 区分不同的订阅类型，等于 RCSubscribeTypeUserProfile 时代表用户信息订阅事件。
   - (void)onSubscriptionSyncCompleted:(RCSubscribeType)type;
   ```

3. 远程设备订阅信息变更。当用户在其他设备上的订阅信息发生变更时，此方法将被调用。您可以利用这个方法来更新当前设备上的状态信息，以保证订阅信息的一致性。 subscribeEvents 包含发生变化的订阅事件的列表。

```
/// 需要判断 RCSubscribeEvent 的 subscribeType，等于 RCSubscribeTypeUserProfile 时代表用户信息订阅事件。
- (void)onSubscriptionChangedOnOtherDevices:(NSArray<RCSubscribeEvent *> *)subscribeEvents;
```

消息介绍　　消息介绍

更新时间:2024-08-30

IMLib SDK 定义了 RCMessage 类，用于进行消息传输和管理。

RCMessage 模型

RCMessage 类中封装了以下关键数据：

- **用于消息传输的属性**：发送者 ID、接收者 ID、所属会话类型等。
- **消息内容体**： 用于封装一条消息携带的的具体内容，分为普通消息内容体和媒体消息内容体。例如，文本消息内容（RCTextMessage）属于普通消息内容体，图片消息内容（RCImageMessage）属于媒体消息内容体。消息内容体的类型，常称为「消息类型」，决定了使用发送普通消息还是发送媒体的接口。

> 提示
>
> 文档里出现的「消息」，如文本消息、语音消息等，有时指继承自 RCMessageContent 或 RCMediaMessageContent 的消息具体内容。

下表描述 RCMessage 类的关键属性，完整的属性列表可参考 API 文档。

RCMessage 的数据结构如下：

属性名	类型	描述
objectName	NSString	消息类型的标识名，通过消息体类方法 `+getObjectName` 获取的值决定。
content	RCMessageContent	消消息携带的具体内容，必须为 RCMessageContent 或 RCMediaMessageContent 的子类对象，其中封装了不同类型消息的具体数据字段。即时通讯服务已提供预定义的消息类型，并规定了跨平台一致的消息内容体结构（参见消息类型概述），如文本、语音、图片、GIF 等。您也可以通过自定义消息内容体创建自定义消息类型。
conversationType	RCConversationType	会话类型枚举，例如单聊、群聊、聊天室、超级群、系统会话等。参考会话介绍。
senderUserId	NSString	消息发送者的用户 ID。
targetId	NSString	会话 ID（或称目标 ID），用于标识会话对端。 1. 针对单聊会话，用对端用户的 ID 作为 Target ID，因此发送与接收的所有消息的 Target ID 一定为对端的用户 ID。请注意，本端用户所接收的消息在本地消息数据中存储的 Target ID 也不是当前用户 ID，而是对端用户（消息发送者）的用户 ID。 2. 在群组、聊天室、超级群会话中，Target ID 为对应的群组、聊天室、超级群 ID。 3. 针对系统会话，因客户端用户无法回复系统会话消息，因此不需要 Target ID。
channelId	NSString	超级群频道 ID。
messageId	long	消息 ID，消息在本地存储中的唯一 ID（数据库索引唯一值）。
messageUId	String	消息 UID，在同一个 App Key 下全局唯一。只有发送成功的消息才有唯一 ID。
messageDirection	RCMessageDirection	消息方向枚举，分为发送和接收。
sentTime	long long	消息发送时间。发送时间为消息从发送客户端到达服务器时服务器的本地时间。使用 UNIX 时间戳，单位为毫秒。
receivedTime	long long	消息接收时间。接收时间为消息到达接收客户端时客户端的本地时间。使用 UNIX 时间戳，单位为毫秒。
sentStatus	RCSentStatus	发送消息的状态。如发送中、发送成功、发送失败、取消发送。

属性名	类型	描述
receivedStatus	RCReceivedStatus	接收到的消息的状态，如是否已读、是否下载等。详见消息接收状态。
readReceiptInfo	RCReadReceiptInfo	群聊消息已读回执信息。详细请参考群聊消息回执。
canIncludeExpansion	BOOL	是否允许消息扩展。详情请参考 消息扩展。
expansionDic	NSDictionary	消息扩展信息。详情请参考 消息扩展。
extra	NSString	消息的附加信息，该字段为本地操作的字段，不会发往服务端。请区别于 RCMessage.content.extra，后者会随消息一并发送到对端。
isOffLine	BOOL	是否是离线消息，只在接收消息的回调方法中有效，如果消息为离线消息，则为 YES ，其他情况均为 NO。

RCMessageContent

RCMessage 类中封装了 content 属性，代表一条消息携带的具体内容。消息内容体的类型必须继承以下基类：

- RCMessageContent - 即普通消息内容体。例如，SDK 内置消息类型中的文本消息内容体（RCTextMessage）即继承自 `RCMessageContent`。
- RCMediaMessageContent - 即媒体消息内容体，继承自 `MessageContent` 基类，并在其基础上增加了对媒体文件的处理逻辑。。例如，SDK 内置消息类型中的文本消息内容体（RCImageMessage）即继承自 `MediaMessageContent`。在发送和接收消息时，SDK 会判断消息类型是否为媒体类型消息，如果是媒体类型，则会触发上传或下载媒体文件流程。

即时通讯服务已提供预定义的、跨平台一致的消息内容体结构（参见消息类型概述），如文本、语音、图片、GIF 等。如果您需要实现自定义消息，可以继承 RCMessageContent 或 RCMediaMessageContent，创建自定义消息内容体。

消息存储策略

客户端 SDK 和即时通讯服务端通过消息存储策略（RCMessagePersistent）识别消息的类型、在本地和服务端的存储策略、是否计入未读消息数等属性。继承自 RCMessageContent 或 RCMediaMessageContent 的消息具体内容都遵守 RCMessagePersistentCompatible 协议，设置与存储相关属性。

如果发送 SDK 内置消息类型的消息，则 RCMessagePersistent 由 SDK 默认自动处理，无需额外操作。关于内置消息的存储策略，详见消息类型概述。

如果您需要创建自定义消息类型，则需要注意，自定义消息必须遵守 RCMessagePersistentCompatible 协议。详见自定义消息类型。

发送消息　　发送消息

更新时间:2024-08-30

本文主要描述了如何使用 IMLib SDK 向单聊会话、群聊会话、聊天室会话中发送消息。

> 如发送超级群消息，请参见超级群群文档 收发消息。

消息内容类型简介

IMLib SDK 定义的 RCMessage 对象的 content 属性中可包含两大类消息内容：普通消息内容和媒体消息内容。普通消息内容父类是 RCMessageContent，媒体消息内容父类是 RCMediaMessageContent。发送媒体消息和普通消息本质的区别为是否有上传数据过程。

功能	消息内容的类型	父类	描述
文本消息	RCTextMessage	RCMessageContent	文本消息的内容。
引用回复	RCReferenceMessage	RCMessageContent	引用消息的内容，用于实现引用回复功能。
图片消息	RCImageMessage	RCMediaMessageContent	图片消息的内容，支持发送原图。
GIF 消息	RCGIFMessage	RCMediaMessageContent	GIF 消息的内容。
文件消息	RCFileMessage	RCMediaMessageContent	文件消息的内容。
语音消息	RCHQVoiceMessage	RCMediaMessageContent	高清语音消息的内容。
提及他人（@ 消息）	不适用	不适用	@消息并非预定义的消息类型。详见如何发送 @ 消息。

以上为 IMLib SDK 内置的部分消息内容类型。您还可以创建自定义的消息内容类型，并使用 sendMessage 方法或sendMediaMessage 方法发送。详见自定义消息类型。

> 重要
>
> - 发送普通消息使用 `sendMessage` 方法，发送媒体消息使用 `sendMediaMessage` 方法。
> - 客户端 SDK 发送消息存在频率限制，每秒最多只能发送 5 条消息。

本文使用 IMLib SDK 核心类 RCIMClient（也可以用 RCCoreClient）的发送消息方法。

普通消息

普通消息指文本消息、引用消息等不涉及媒体文件上传的消息。普通消息的消息内容为 RCMessageContent 的子类的消息，例如文本消息内容（RCTextMessage），或自定义类型的普通消息内容。

构造普通消息

在发送前，需要先构造 RCMessage 对象，指定会话类型（单聊、群聊、聊天室），会话的 Target ID，消息方向（发送或接收），消息内容。以下示例中构造了一条包含文本消息内容（RCTextMessage）的消息对象。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"测试文本消息"];
RCConversationType conversationType = ConversationType_PRIVATE

RCMessage *message = [[RCMessage alloc]
initWithType:conversationType
targetId:@"targetId"
direction:MessageDirection_SEND
content:messageContent];
```

发送普通消息

如果 RCMessage 对象包含普通消息内容，使用 IMLib SDK 核心类 RCCoreClient（也可以用 RCIMClient）的 sendMessage 方法发送消息。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:messageContent];


[[RCCoreClient sharedRCCoreClient]
sendMessage:message
pushContent:nil
pushData:nil
attached:^(RCMessage *successMessage) {
//入库成功
}
successBlock:^(RCMessage *successMessage) {
//成功
}
errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}];
```

sendMessage 方法中直接提供了用于控制推送通知内容（pushContent）和推送附加信息（pushData）的参数。另一种方式是使用消息推送属性配置 RCMessagePushConfig，其中包含了 pushContent 和 pushData，并提供更多推送通知配置能力，例如标题、内容、图标、或其他第三方厂商个性化配置。详见远程推送通知。

关于是否需要配置输入参数或 RCMessage 的 messagePushConfig 属性中 pushContent，请参考以下内容：

- 如果消息类型为即时通讯服务预定义消息类型中的用户内容类消息格式，例如 [TextMessage]，`pushContent` 可设置为 `null`。一旦消息触发离线推送通知时，远程推送通知默认使用服务端预置的推送通知内容。关于各类型消息的默认推送通知内容，详见用户内容类消息格式。
- 如果消息类型为即时通讯服务预定义消息类型中通知类、信令类（"撤回命令消息" 除外），且需要支持远程推送通知，则必须填写 `pushContent`，否则收件人不在线时无法收到远程推送通知。如无需触发远程推送，可不填该字段。
- 如果消息类型为自定义消息类型，请参考自定义消息如何支持远程推送。
- 请注意，聊天室会话不支持离线消息机制，因此也不支持离线消息转推送。

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（conversationType），会话 ID（targetId），消息内容（content）。详见[消息介绍] 中对 RCMessage 的结构说明。
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。
pushData	NSString	Push 附加信息。可设置为 nil。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。
attachedBlock	Block	消息已成功存入消息数据库的回调。
successBlock	Block	消息发送成功回调。
errorBlock	Block	消息发送失败回调。

通过 RCCoreClient 的 sendMessage 方法的回调，融云服务器始终会通知您的消息是否已发送成功。当因任何问题导致发送失败时，可通过回调方法返回异常。

注意：

- 关于如何个性化配置接收方离线时收到的远程推送通知，详见下文远程推送通知。
- 自定义消息类型默认不支持离线消息转推送机制。如需支持，详见下文自定义消息如何支持远程推送。

媒体消息

媒体消息指图片、GIF、文件等需要处理媒体文件上传的消息。媒体消息的消息内容类型为 RCMeidaMessageContent 的子类，例如高清语音消息内容（RCHQVoiceMessage ），或您自定义类型的媒体消息内容。

构造媒体消息

在发送前，需要先构造 RCMessage 对象。媒体消息 RCMessage 对象的 content 字段必须传入 RCMediaMessageContent 的子类对象，表示媒体消息内容。例如图片消息内容（RCImageMessage ）、GIF 消息内容（RCGIFMessage ），或继承自 RCMediaMessageContent 的自定义媒体消息内容。

图片消息内容（RCImageMessage ）支持设置为发送原图。

```
RCImageMessage *mediaMessageContent = [RCImageMessage messageWithImageURI:@"path/to/image"];
mediaMessageContent.full YES; // 图片消息支持设置以原图发送

RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:mediaMessageContent];
```

在发送前，图片会被压缩质量，以及生成缩略图，在聊天界面中展示。GIF 无缩略图，也不会被压缩。

- 图片消息的缩略图：SDK 会以原图 30% 质量生成符合标准大小要求的大图后再上传和发送。压缩后最长边不超过 240 px。缩略图用于在聊天界面中展示。
- 图片：发送消息时如未选择发送原图，SDK 会以原图 85% 质量生成符合标准大小要求的大图后再上传和发送。压缩后最长边不超过 1080 px。

一般情况下不建议修改 SDK 默认压缩配置。如需调整 SDK 压缩质量，详见知识库文档如何修改 SDK 默认的图片与视频压缩配置 。

发送媒体消息

发送媒体消息需要使用 sendMediaMessage 方法。SDK 会为图片、小视频等生成缩略图，根据默认压缩配置 进行压缩，再将图片、小视频等媒体文件上传到融云默认的文件服务器（文件存储时长 ），上传成功之后再发送消息。图片消息如已设置为发送原图，则不会进行压缩。

```
[[RCCoreClient sharedCoreClient] sendMediaMessage:message
pushContent:nil
pushData:nil
attached:^(RCMessage *successMessage) {
//入库成功
}
progress:^(int progress, RCMessage *progressMessage) {
//媒体上传进度
}
successBlock:^(RCMessage *successMessage) {
//成功
}
errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}
cancel:^(RCMessage *cancelMessage) {
//取消
}];
```

sendMediaMessage 方法中直接提供了用于控制推送通知内容（pushContent）和推送附加信息（pushData）的参数。另一种方式是使用消息推送属性配置 RCMessagePushConfig ，其中包含了 pushContent 和 pushData，并提供更多推送通知配置能力，例如标题、内容、图标、或其他第三方厂商个性化配置。详见远程推送通知。

关于是否需要配置输入参数或 RCMessage 的 messagePushConfig 属性中 pushContent，请参考以下内容：

- 如果消息类型为即时通讯服务预定义消息类型中的用户内容类消息格式，例如 RCTextMessage，`pushContent` 可设置为 `nil`。一旦消息触发离线推送通知时，远程推送通知默认使用服务端预置的推送通知内容。关于各类型消息的默认推送通知内容，详见用户内容类消息格式。
- 如果消息类型为即时通讯服务预定义消息类型中通知类、信令类（"撤回命令消息" 除外），且需要支持远程推送通知，则必须填写 `pushContent`，否则收件人不在线时无法收到远程推送通知。如无需触发远程推送，可不填该字段。
- 如果消息类型为自定义消息类型，请参考自定义消息如何支持远程推送。
- 请注意，聊天室会话不支持离线消息机制，因此也不支持离线消息转推送。

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（`conversationType`），会话 ID（`targetId`），消息内容（`content`）。详见[消息介绍] 中对 `RCMessage` 的结构说明。
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 `RCMessage` 的推送属性（`RCMessagePushConfig`）中配置，会覆盖此处配置，详见配置消息的推送属性。
pushData	NSString	Push 附加信息。可设置为 `nil`。您也可以在 `RCMessage` 的推送属性（`RCMessagePushConfig`）中配置，会覆盖此处配置，详见配置消息的推送属性。
attachedBlock	Block	消息已成功存入消息数据库的回调。
progressBlock	Block	媒体上传进度的回调。
successBlock	Block	消息发送成功回调。
errorBlock	Block	消息发送失败回调。
cancel	Block	取消发送的回调。

通过 `RCCoreClient` 的 `sendMediaMessage` 方法的回调方法，融云服务器始终会通知媒体文件上传进度，以及您的消息是否已发送成功。当因任何问题导致发送失败时，可通过回调方法返回异常。

发送媒体消息的方法默认将媒体文件上传到融云的文件服务器，您可以在客户端应用程序 下载媒体消息文件。融云对上传媒体文件大小进行了限制，GIF 大小限制为 2 MB，文件上传限制为 100 MB。如果您需要提高上限，可提交工单。

> **注意：**
>
> - 关于如何个性化配置接收方离线时收到的远程推送通知，详见下文远程推送通知。
> - 自定义消息类型默认不支持离线消息转推送机制。如需支持，详见下文自定义消息如何支持远程推送。

发送媒体消息并且上传到自己的服务器

您可以直接发送您服务器上托管的文件。将媒体文件的 URL（表示其位置）作为参数，在构建媒体消息内容时传入。在这种情况下，您的文件不会托管在融云服务器上。当您发送带有远程 URL 的文件消息时，文件大小没有限制，您可以直接使用 `sendMessage` 方法发送消息。

如果您希望 SDK 在您上传成功后发送消息，您可以使用 `RCCoreClient` 的 sendMediaMessage 方法，具体操作如下：

- 上传媒体文件的过程中，可调用 RCUploadMediaStatusListener 的 `updateBlock`、`errorBlock`，通知 SDK 当前上传媒体文件的进度和状态。
- 上传完毕后，取得网络文件 URL。通过 RCUploadMediaStatusListener 的 `currentMessage` 属性获取当前 `RCMessage` 对象，将 `RCMessage` 对象的 `content` 中的对应 URL 字段设置成上传成功的网络文件 URL。
- 调用 RCUploadMediaStatusListener 的 `successBlock`，通知 SDK 文件上传完成。

```
RCImageMessage *mediaMessageContent = [RCImageMessage messageWithImageURI:@"path/to/image"];
mediaMessageContent.full = YES; // 图片消息支持设置以原图发送

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:mediaMessageContent];

[[RCCoreClient sharedCoreClient] sendMediaMessage:message
pushContent:nil
pushData:nil
attached:^(RCMessage * _Nullable msg) {
} uploadPrepare:^(RCUploadMediaStatusListener * _Nonnull uploadListener) {
RCMessage *currentMessage = uploadListener.currentMessage;
// App 在上传媒体文件时，需要在监听中调用 updateBlock、successBlock 与 errorBlock
if ([currentMessage.content isKindOfClass:[RCImageMessage class]]) {
RCImageMessage *content = (RCImageMessage *)currentMessage.content;
// 上传自己服务器后，获取到的图片 url 地址
content.remoteUrl = @"https://www.test.com/img/test.jpg";
uploadListener.successBlock(content);
}
} progress:^(int progress, long messageId) {
//媒体上传进度，需要自行刷新 UI
} success:^(long messageId) {
} error:^(RCErrorCode errorCode, long messageId) {
} cancel:^(long messageId) {
}];
```

如何发送 @ 消息

@消息在融云即时通讯服务中不属于一种预定义的消息类型（详见服务端文档 消息类型概述 ）。融云通过在消息中携带 mentionedInfo 数据，帮助 App 实现提及他人（@）功能。

消息的 RCMessageContent 中的 RCMentionedInfo 字段存储了携带所 @ 人员的信息。无论是 SDK 内置消息类型，或者您自定义的消息类型，都直接或间接继承了 RCMessageContent 类。

融云支持在向群组和超级群中发消息时，在消息内容中添加 mentionedInfo。消息发送前，您可以构造 RCMentionedInfo 对象，并设置到消息的 RCMessageContent 中。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"Test"];

messageContent.mentionedInfo = [[RCMentionedInfo alloc] initWithMentionedType:RC_Mentioned_Users
userIdList:mentionedUserIdList mentionedContent:nil];
```

MentionedInfo 参数：

参数	类型	说明
type	RCMentionedType	（必填）指定 MentionedInfo 的类型。RC_Mentioned_All 表示需要提及（@）所有人。RC_Mentioned_Users 表示需要 @ 部分人，被提及的人员需要在 userIdList 中指定。
userIdList	NSArray	被提及（@）用户的 ID 集合。当 type 为 RC_Mentioned_Users 时必填。从 5.3.1 版本开始，支持当 type 为 RC_Mentioned_All 时同时在 userIdList 中提及部分人。接收端可通过 RCMentionedInfo 获取 userIdList 数据。
mentionedContent	NSString	触发离线消息推送时，通知栏显示的内容。如果是 nil，则显示默认提示内容（“有人 @ 你”）。@消息携带的 mentionedContent 优先级最高，会覆盖所有默认或自定义的 pushContent 数据。

以下示例展示了发送一条提及部分用户的文本消息，该消息发往一个群聊会话。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"测试文本消息"];

messageContent.mentionedInfo = [[RCMentionedInfo alloc]
initWithMentionedType:RC_Mentioned_Users
userIdList:@[@"userId1", @"userId2"]
mentionedContent:nil];

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_GROUP
targetId:@"targetId"
direction:MessageDirection_SEND
content:messageContent];

[[RCCoreClient sharedRCCoreClient]
sendMessage:message
pushContent:nil
pushData:nil
successBlock:^(RCMessage *successMessage) {
//成功
}
errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}];
```

IMLib SDK 接收消息后，您需要处理 @ 消息中的数据。您可以在获取 RCMessage(消息对象)后获取到消息对象携带的 RCMentionedInfo 对象。

远程推送通知

> 提示
>
> 聊天室会话不支持离线消息机制，因此也不支持离线消息转推送。

如果您的应用已经在融云配置第三方推送，在消息接收方离线时，融云服务端会根据消息类型、接收方支持的推送通道、接收方的免打扰设置等，决定是否触发远程推送。

远程推送通知一般会展现在系统的通知栏。融云内置的消息类型默认会在通知栏展现通知标题和通知内容。关于各类型消息的默认推送通知内容，详见用户内容类消息格式 。

如果您需要个性化的离线推送通知，可以通过以下方式，修改或指定远程推送的通知标题、通知内容其他属性。

- sendMessage 和 sendMediaMessage 方法的输入参数中直接提供了用于控制推送通知内容（`pushContent`）参数。
- 如果您需要控制离线推送通知的更多属性，例如标题、内容、图标、或根据第三方厂商通道作个性化配置，请使用 RCMessage 的推送属性（RCMessagePushConfig）进行配置。消息推送属性中的 `pushContent` 配置会覆盖发送消息接口中的 `pushContent`。

配置消息推送属性

在发送消息时，您可以通过设置 RCMessage 的 messagePushConfig 属性，对单条消息的推送行为进行个性化配置。例如：

- 自定义推送标题、推送通知内容
- 自定义通知栏图标
- 添加远程推送附加信息
- 越过接收客户端的配置，强制在推送通知内显示通知内容
- 其他 APNs 或 Android 推送通道支持的个性化配置

相对于发送消息时输入参数中的 pushContent 和 pushData，MessagePushConfig 中的配置具有更高优先级。发送消息时，如已配置 RCMessage 的 messagePushConfig 属性，则优先使用 messagePushConfig 中的配置。

```
RCTextMessage *txtMsg = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:txtMsg];

RCMessagePushConfig *pushConfig = [[RCMessagePushConfig alloc] init];
pushConfig.disablePushTitle = NO;
pushConfig.pushTitle = @"通知标题";
pushConfig.pushContent = @"通知内容";
pushConfig.pushData = @"通知的 pushData";
pushConfig.templateId = @"templateId";
pushConfig.iosConfig.threadId = @"iOS 用于通知分组的 id";
pushConfig.iosConfig.apnsCollapseId = @"iOS 用于通知覆盖的 id";
pushConfig.iosConfig.richMediaUri = @"iOS 推送自定义的通知栏消息右侧图标 URL";
pushConfig.androidConfig.notificationId = @"Android 的通知 id";
pushConfig.androidConfig.channelIdMi = @"小米的 channelId";
pushConfig.androidConfig.channelIdHW = @"华为的 channelId";
pushConfig.androidConfig.categoryHW = @"华为的 Category";
pushConfig.androidConfig.channelIdOPPO = @"OPPO 的 channelId";
pushConfig.androidConfig.typeVivo = @"vivo 的 classification";
pushConfig.androidConfig.categoryVivo = @"vivo 的 Category";
pushConfig.forceShowDetailContent = YES;
message.messagePushConfig = pushConfig;

/// 调用 IMLib 发送消息方法
```

消息推送属性说明（RCMessagePushConfig）提供以下参数：

参数	类型	说明
disablePushTitle	BOOL	是否屏蔽通知标题，此属性只针目标用户为 iOS 平台时有效，Android 第三方推送平台的通知标题为必填项，所以暂不支持。
pushTitle	NSString	推送标题，此处指定的推送标题优先级最高。如不设置，可参考 用户内容类消息格式 中对各内置消息类型默认推送通知标题与推送通知内容的说明。
pushContent	NSString	推送内容。此处指定的推送内容优先级最高。如不设置，可参考 用户内容类消息格式 中对各内置消息类型默认推送通知标题与推送通知内容的说明。
pushData	NSString	远程推送附加信息，如果没有，则使用发送消息的 `pushData`
forceShowDetailContent	BOOL	是否强制显示通知详情，当目标用户通过 RCPushProfile 中的 - (void)updateShowPushContentStatus:(BOOL)isShowPushContent success:(void (^)(void))successBlock error:(void (^)(RCErrorCode status))errorBlock; 设置推送不显示消息详情时，可通过此参数，强制设置该条消息显示推送详情
templateId	NSString	推送模板 ID，设置后根据目标用户通过 SDK RCPushProfile 中的 setPushLauguageCode 设置的语言环境，匹配模板中设置的语言内容进行推送，未匹配成功时使用默认内容进行推送。模板内容在“控制台-自定义推送文案”中进行设置，具体操作请参见 配置和使用自定义多语言推送模板。
iOSConfig	RCiOSConfig	iOS 平台相关配置。详见 **RCiOSConfig 属性说明**。
androidConfig	RCAndroidConfig	Android 平台相关配置。**RCiOSConfig 属性说明**。

- **RCiOSConfig 属性说明**

参数	类型	说明
threadId	NSString	iOS 平台通知栏分组 ID，相同的 threadId 推送分为一组（iOS10 开始支持）
apnsCollapseId	NSString	iOS 平台通知覆盖 ID，apnsCollapseId 相同时，新收到的通知会覆盖老的通知，最大 64 字节（iOS10 开始支持）
richMediaUri	NSString	iOS 推送自定义的通知栏消息右侧图标 URL，需要 App 自行解析 `richMediaUri` 并实现展示。图片要求：大小 120 * 120px，格式为 png 或者 jpg 格式。SDK 从 5.2.4 版本开始支持携带该字段。
interruptionLevel	NSString	适用于 iOS 15 及之后的系统。取值为 `passive`，`active`（默认），`time-sensitive`，或 `critical`，取值说明详见对应的 APNs 的 interruption-level 字段。在 iOS 15 及以上版本中，系统的“定时推送摘要”、“专注模式”都可能导致重要的推送通知（例如余额变化）无法及时被用户感知的情况，可考虑设置该字段。SDK 5.6.7 及以上版本支持该字段。

- **RCAndroidConfig 属性说明**

参数	类型	说明
notificationId	NSString	Android 平台 Push 唯一标识，目前支持小米、华为推送平台，默认开发者不需要进行设置，当消息产生推送时，消息的 messageUId 作为 notificationId 使用
channelIdMi	NSString	小米的渠道 ID，该条消息针对小米使用的推送渠道，如开发者集成了小米推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，channelId 由开发者自行创建
miLargeIconUrl	NSString	（由于小米官方已停止支持该能力，该字段已失效）小米通知类型的推送所使用的通知图片 url。图片要求：大小120 * 120px，格式为 png 或者 jpg 格式。 此属性 5.1.7 及以上版本支持。支持 MIUI 国内版（国内版要求为 MIUI 12 及以上）和国际版。
channelIdHW	NSString	华为的渠道 ID，该条消息针对华为使用的推送渠道，如开发者集成了华为推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，channelId 由开发者自行创建
hwImageUrl	NSString	华为推送通知中自定义的通知栏消息右侧小图片 URL，如果不设置，则不展示通知栏右侧图片。图标文件须小于 512 KB，图标建议规格大小：40dp x 40dp，弧角大小为 8dp，超出建议规格大小的图标会存在图片压缩或显示不全的情况。 此属性 5.1.7 及以上版本支持
categoryHW	NSString	华为推送通道的消息自分类标识，默认为空。category 取值必须为大写字母，例如 `IM`。App 根据华为要求完成华为自分类权益申请 或 申请特殊权限 后可传入该字段有效。详见华为推送官方文档华为消息分类标准。该字段优先级高于控制台为 App Key 下的应用标识配置的华为推送 Category。SDK 5.4.0 及以上版本支持该字段。
importanceHW	RCImportanceHw	华为推送的消息提醒级别。`RCImportanceHwLow` 表示通知栏消息预期的提醒方式为静默提醒，消息到达手机后，无铃声震动。`RCImportanceHwNormal` 表示通知栏消息预期的提醒方式为强提醒，消息到达手机后，以铃声、震动提醒用户。终端设备实际消息提醒方式将根据 `categoryHW` 字段取值、或者控制台配置的 `category` 字段取值，或者华为智能分类结果进行调整。SDK 5.1.3 及以上版本支持该字段。
imageUrlHonor	NSString	荣耀推送通知中用户自定义的通知栏右侧大图标 URL，如果不设置，则不展示通知栏右侧图标。图标文件须小于 512 KB，图标建议规格大小：40dp x 40dp，弧角大小为 8dp，超出建议规格大小的图标会存在图片压缩或显示不全的情况。SDK 5.6.7 及以上版本支持该字段。
importanceHonor	RCimportanceHonor	荣耀推送的 Android 通知消息分类，决定用户设备消息通知行为。`RCImportanceHonorLow` 表示资讯营销类消息。`RCImportanceHonorNormal`（默认值）表示服务与通讯类消息。SDK 5.6.7 及以上版本支持该字段。
typeVivo	NSString	VIVO 推送服务的消息类别。可选值 `0`（运营消息） 和 `1`（系统消息）。该参数对应 VIVO 推送服务的 `classification` 字段，详见 VIVO 推送消息分类说明
categoryVivo	NSString	VIVO 推送服务的消息二级分类。例如 `IM`（即时消息）。该参数对应 VIVO 推送服务的 `category` 字段。详细的 category 取值请参见 VIVO 推送消息分类说明。如果指定二级分类 `categoryVivo`，必须同时指定 `typeVivo`（系统消息或运营消息）。请注意遵照 VIVO 官方要求，确保二级分类属于 VIVO 系统消息场景或运营消息场景下允许发送的内容。`categoryVivo` 字段优先级高于控制台为 App Key 下的应用标识配置的 VIVO 推送 Category。SDK 5.4.2 及以上版本支持该字段。
channelIdOPPO	NSString	OPPO 的渠道 ID，该条消息针对 OPPO 使用的推送渠道，如开发者集成了 OPPO 推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，channelId 由开发者自行创建
fcmCollapseKey	NSString	FCM 推送的通知分组 ID。SDK 5.1.3 及以上版本支持该字段。注意，如使用该字段，请确保控制台的 FCM 推送配置中推送方式为通知消息方式。
fcmImageUrl	NSString	FCM 推送的通知栏右侧图标 URL。如果不设置，则不展示通知栏右侧图标。SDK 5.1.3 及以上版本支持该字段。注意，如使用该字段，请确保控制台的 FCM 推送配置鉴权方式为证书，推送方式为通知消息方式。
fcmChannelId	NSString	FCM 的渠道 ID，该条消息针对 FCM 推送渠道，如开发者集成了 FCM 推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，`channelId` 由开发者自行创建

Channel ID 需要由 Android 端开发者进行创建，创建方式如下：

推送通道	配置说明
华为	App 端，调用 Android SDK 创建 Channel ID 接口创建 Channel ID
小米	在小米开放平台管理台上创建 Channel ID 或通过小米服务端 API 创建
OPPO	App 端，调用 Android SDK 创建 Channel ID；在 OPPO 管理台登记该 Channel ID，保持一致性
vivo	调用服务端 API 创建 Channel ID

自定义消息如何支持远程推送

融云为内置的消息类型默认支提供了远程通知标题和通知内容（详见用户内容类消息格式 ）。不过，如果您发送的是自定义类型的消息，则需要您自行提供 pushContent 字段，否则用户无法收到离线推送通知。具体如下：

- 如果消息类型为自定义消息类型，且需要支持离线推送通知，则必须向融云提供 `pushContent` 字段，否则用户无法收到离线推送通知。
- 如果消息类型为自定义消息类型，但不需要支持远程推送通知（例如通过自定义消息类型实现的 App 业务层操作指令），可将 `pushContent` 字段留空。

> 提示
>
> 如果要自定义消息启用离线推送通知能力，请务必发送消息的入参或消息推送属性向融云提供 pushContent 字段内容，否则接收方用户无法收到离线推送通知。

为消息禁用推送通知

在接收者未上线时，融云服务端默认触发推送服务，将消息通过推送通道下发（服务端是否触发推送也会收到应用级别、用户级别免打扰设置的影响）。

有时 App 用户可能希望在发送消息时就指定该条消息不需要触发推送，该需求可通过 RCMessage 对象的 MessageConfig 配置实现。

以下示例中，我们将 messageConfig 的 disableNotification 设置为 YES 禁用该条消息的推送通知。接收方再次上线时会通过融云服务端的离线消息缓存（最多缓存 7 天）自动收取单聊、群聊、系统会话消息。超级群消息因不支持离线消息机制，需要主动拉取。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:MessageContent];

message.messageConfig.disableNotification = YES;
```

如何透传自定义数据

如果应用需要将自定义数据透传到对端，可通过以下方式实现：

- 消息内容 RCMessageContent 的附加信息字段，该字段会随即时消息一并发送到对端。接收方在线接收消息后，可从消息内容中获取该字段。请区别于 `Message.extra`，`Message.extra` 为本地操作的字段，不会被发往服务端。
- 远程推送附加信息 `pushData`。您可以在 sendMessage 和 sendMediaMessage 的输入参数中直接设置 `pushData`，也可以使用消息推送属性中的同名字段，后者中的 `pushData` 会覆盖前者。pushData 仅会在消息触发离线远程推送时下发到对端设备。对端收到远程推送通知时，可通过推送数据中的 `appData` 字段获取透传的数据内容。详见获取推送数据 。

如何处理消息发送失败

对于客户端本地会存储的消息类型（参见消息类型概述），如果触发（errorBlock）回调，App 可以在 UI 临时展示这条发送失败的消息，并缓存 errorBlock 抛出的 RCMessage 实例，在合适的时机重新调用 sendMessage / sendMediaMessage 发送。请注意，如果不复用该消息实例，而是重发相同内容的新消息，本地消息列表中会存储内容重复的两条消息。

对于客户端本地不存储的消息，如果发送失败（errorBlock），App 可缓存消息实例再重试发送，或直接新建消息实例进行发送。

如何实现转发、群发消息

转发消息：IMLib SDK 未提供转发消息接口。App 实现转发消息效果时，可调用发送消息接口发送。

群发消息：群发消息是指向多个用户发送消息。更准确地说，是向多个 Target ID[?]发送消息，一个 Target ID 可能代表一个用户，一个群组，或一个聊天室。客户端 SDK 未提供直接实现群发消息效果的 API。您可以考虑以下实现方式：

- App 循环调用客户端的发送消息 API。请注意，客户端 SDK 限制发送消息的频率上限为每秒 5 条。
- App 在业务服务端接入即时通讯服务端 API（IM Server API），由 App 业务服务端群发消息。IM Server API 的 发送单聊消息 接口支持单次向多个用户发送单聊消息。您也可以通过 发送群聊消息、发送聊天室消息接口实现单次向多个群组或多个聊天室发送消息。

接收消息 接收消息

更新时间:2024-08-30

您可通过设置消息监听器拦截 SDK 接收的消息，并进行相应的业务操作。

监听消息接收

应用程序可以通过 addReceiveMessageDelegate 方法设置多个消息接收代理。所有接收到的消息都会在此 RCIMClientReceiveMessageDelegate 协议的代理方法中回调。建议在应用生命周期内注册消息监听。

```
[[RCCoreClient sharedCoreClient] addReceiveMessageDelegate:self];
```

RCIMClientReceiveMessageDelegate 可用于处理接收实时消息或离线消息。该协议提供两个代理方法，监听消息处理只需要在其中一个代理方法内实现。SDK 会通过此方法接收包含单聊、群聊、聊天室、系统类型的所有消息。

代理方法一提供了还剩余的未接收的消息数 nLeft 参数。您可以根据 nLeft 的数量来优化您的 App 体验和性能，比如收到大量消息时等待 nLeft 为 0 再刷新 UI。

```
/*!
接收消息的回调方法

@param message 当前接收到的消息
@param nLeft 还剩余的未接收的消息数，left>=0
@param object 消息监听设置的 key 值

*/
- (void)onReceived:(RCMessage *)message left:(int)nLeft object:(id)object;
```

当客户端连接成功后，服务端会将所有离线消息[?]以消息包（Package）的形式下发给客户端，每个 Package 中最多含 200 条消息。客户端会解析 Package 中的消息，逐条上抛并通知应用。第二个代理方法额外暴露了 offline 和 hasPackage 参数。您可以根据 nLeft、offline、hasPackage 选择合适的时机刷新 UI。

```
/**
接收消息的回调方法

@param message 当前接收到的消息
@param nLeft 还剩余的未接收的消息数，left>=0
@param object 消息监听设置的 key 值
@param offline 是否是离线消息
@param hasPackage SDK 拉取服务器的消息以包(package)的形式批量拉取，有 package 存在就意味着远端服务器还有消息尚未被 SDK
拉取
*/
- (void)onReceived:(RCMessage *)message
left:(int)nLeft
object:(id)object
offline:(BOOL)offline
hasPackage:(BOOL)hasPackage;
```

参数	类型	说明
message	RCMessage	接收的消息对象。
nLeft	int	当客户端连接成功后，服务端会将所有离线消息[?]以消息包（Package）的形式下发给客户端，每个 Package 中最多含 200 条消息。客户端会解析 Package 中的消息，逐条上抛并通知应用。nLeft 为当前正在解析的消息包（Package）中还剩余的消息条数。

参数	类型	说明
offline	boolean	当前消息是否离线消息。
hasPackage	boolean	是否在服务端还存在未下发的消息包（Package）。

同时满足以下条件，表示离线消息已收取完毕：

- `hasPackage` 为 `NO`：表示当前正在解析最后一包消息。
- `nLeft` 为 0：表示最后一个消息包中最后一条消息已接收完毕。

从 5.2.3 版本开始，每次连接成功后，离线消息收取完毕时会触发 RCIMClientReceiveMessageDelegate 中的以下回调方法。如果没有离线消息，连接成功后会立即触发。

```
/*!
离线消息接收完成
*/
- (void)onOfflineMessageSyncCompleted;
```

SDK 支持移除监听器。为了避免内存泄露，请在不需要监听时将监听器移除。

```
[[RCCoreClient sharedCoreClient] removeReceiveMessageDelegate:self];
```

消息接收状态

> 提示
>
> 在 5.6.8 版本前，IMLib SDK 使用 RCMessage 的 `receivedStatus` 表示接收到的消息的状态。该属性在多设备场景下无法正常更新消息的已读状态，已被废弃。

从 5.6.8 版本开始，RCMessage 类中封装了 `receivedStatusInfo` 属性，使用以下属性表示接收到的消息的状态。

状态	描述
`isRead`	是否已读。当前设备或其他设备上已读后，该状态值会变为已读。如果消息在当前设备上被阅读，该状态会变为已读。SDK 5.6.8 版本开始，只要在其他设备上阅读过该消息，当前设备的该状态值会也变为已读。
`isListened`	是否已被收听，仅用于语音消息。
`isDownloaded`	是否已被下载，仅适用于媒体消息。
`isRetrieved`	该消息是否已被同时在线或之前登录的其他设备接收。只要其他设备先收到该消息，该状态值都会变为已接收。

接收消息处理建议

1. 如果接收消息量比较小，建议当 `nLeft == 0` 时刷新 UI，即消息是分批进行刷新。
2. 如果接收消息量比较大，建议当 `hasPackage == NO && nLeft == 0`（即所有的远端消息已经全部接收）时整体刷新一次。
3. 如果接收消息量较大，还可以考虑使用 iOS 函数节流(throttle) 方式，即在一定时间段内，丢弃掉其它触发，就做一次执行。大致思路是接收大量消息的时候起定时器，固定时间刷新一次，等条件满足 `hasPackage == NO && nLeft == 0` 时关闭定时器再刷新一次。

下面分按消息数量大小分两种情况讨论以上方案。

- **消息数量小**：假设远端服务器有 200 条消息需要接收，接收总过程耗时 0.1 秒。
  - 方案 1：刷新 UI 次数为 200/200 = 1 次，即 0.1 秒钟之内刷新了 1 次。
  - 方案 2：刷新 UI 次数为 1 次，不管多少消息只有全部刷新一次，即 0.1 秒之内刷新了 1 次。
  - 方案 3：假设设置的是 0.1 秒固定刷新，那么次数为 0.1/0.1 + 1 = 2 次，即 0.1 秒之内刷新了 2 次。

- **消息数量大**：假设远端服务器有 1 万条消息需要接收，接收总过程耗时 1 秒。

  - 方案 1：刷新 UI 次数为 10000/200 = 50 次，即 1 秒钟之内刷新了 50 次
  - 方案 2：刷新 UI 次数为 1 次，不管多少消息只有全部刷新一次，即 1 秒之内刷新了 1 次
  - 方案 3：假设设置的是 0.5 秒固定刷新，那么次数为 1/0.5 + 1 = 3 次，即 1 秒之内刷新了 3 次。

整体来说方案 1、2 的开发难度小，方案 3 的开发难度大。您需要根据接收消息数量大小的实际情况，选择合适的方案。

上述方案都需要您自行实现。您也可以参考 IMKit SDK 中的处理方法。

禁用消息排重机制

消息排重机制会在 SDK 接收单聊、群聊、系统消息、聊天室时自动去除内容重复消息。当 App 本地存在大量消息，SDK 默认的排重机制可能会因性能问题导致收消息卡顿。因此在接收消息发生卡顿问题时，可尝试关闭 SDK 的排重机制。

为什么接收消息可能出现消息重复

发送端处于弱网情况下可能出现该问题。A 向 B 发送消息后，消息成功到达服务端，并成功下发到接收者 B。但 A 由于网络等原因可能未收到服务端返回的 ack，导致 A 认为没有发送成功。此时如果 A 重发消息，此时 B 就会收到与之前重复的消息（消息内容相同，但 Message UID 不同）。

关闭消息排重

> 提示
>
> 单聊、群聊、系统消息从 5.3.4 版本开始支持关闭消息排重。仅在 RCCoreClient 中提供。

请在 SDK 初始化之后，建立 IM 连接之前调用。多次调用以最后一次为准。

```
BOOL enableCheck = NO; // 关闭消息排重
[[RCCoreClient sharedCoreClient] setCheckDuplicateMessage:enableCheck];
```

> 聊天室消息从 5.8.2 版本开始支持关闭消息排重。在 RCChatRoomClient 中提供。

请在 SDK 初始化之后，建立 IM 连接之前调用。多次调用以最后一次为准。

```
BOOL enableCheck = NO; // 关闭消息排重
[[RCChatRoomClient sharedChatRoomClient] setCheckChatRoomDuplicateMessage:enableCheck];
```

获取历史消息 获取历史消息

更新时间:2024-08-30

获取历史消息可以仅从本地数据中获取，仅从远端获取，和同时从本地与远端获取。

开通服务

从远端获取单群聊历史消息是指从融云服务端获取历史消息，该功能要求 App Key 已启用融云提供的**单群聊消息云端存储**服务。您可以在控制台 IM 服务管理 页面为当前使用的 App Key 开启服务。如果使用生产环境的 App Key，请注意仅 **IM 旗舰版**或 **IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。

提示：请注意区分历史消息记录与离线消息[7]。融云针对单聊、群聊、系统消息默认提供最多 7 天（可调整）的离线消息缓存服务。客户端上线时 SDK 会自动收取离线期间的消息，无需 App 层调用 API。详见管理离线消息存储配置。

从本地数据库中获取消息

使用 `getHistoryMessages` 方法可分页查询指定会话存储在本地数据库中的历史消息，并返回消息对象列表。列表中的消息按发送时间从新到旧排列。

获取会话中所有类型的消息

返回消息实体 RCMessage 对象列表。

```
NSArray *history = [[RCIMClient sharedRCIMClient]
getHistoryMessages:ConversationType_PRIVATE
targetId:@"targetId"
oldestMessageId:lastMessageID
count:count];
```

`count` 参数表示返回列表中应包含多少消息。`oldestMessageId` 参数用于控制分页的边界。每次调用 `getHistoryMessages` 方法时，SDK 会以 `oldestMessageId` 参数指向的消息为界，继续在下一页返回指定数量的消息。如果需要获取会话中最新的 `count` 条消息，可以将 `oldestMessageId` 设置为 -1。

建议获取返回结果中最早一条消息的 ID，并在下一次调用时作为 `oldestMessageId` 传入，以便遍历整个会话的消息历史记录。

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 Id
oldestMessageId	long	以此 messageId 为界，获取发送时间更小的 count 条消息。ID 不存在时，设置为 -1，表示获取最新的 count 条消息。
count	int	需要获取的消息数量，每次最多获取 20 条，按照消息发送时间从新到旧排列

获取会话中指定类型的消息

返回消息实体 RCMessage 对象列表。

```
NSArray *history = [[RCIMClient sharedRCIMClient] getHistoryMessages:ConversationType_PRIVATE targetId:@"targetId"
objectName:@"RC:TxtMsg" oldestMessageId:lastMessageID count:count];
```

`count` 参数表示返回列表中应包含多少消息。`oldestMessageId` 参数用于控制分页的边界。每次调用 `getHistoryMessages` 方法时，SDK 会以

oldestMessageId 参数指向的消息为界，继续在下一页返回指定数量的消息。如果需要获取会话中最新的 count 条消息，可以将 oldestMessageId 设置为 -1。

建议获取返回结果中最早一条消息的 ID，并在下一次调用时作为 oldestMessageId 传入，以便遍历整个会话的消息历史记录。

参数	类型	说明
conversationType	RCConversationType	会话类型。
targetId	NSString	会话 ID。
objectName	NSString	消息类型标识。内置消息类型的标识可参见消息类型概述。
oldestMessageId	long	以此 messageId 为界，获取发送时间更小的 count 条消息。ID 不存在时，设置为 -1，表示获取最新的 count 条消息。
count	int	需要获取的消息数量，每次最多获取 20 条，按照消息发送时间从新到旧排列

通过消息 UID 获取消息

> 提示
>
> - SDK 从 5.2.5.1 版本开始支持通过 UID 批量获取消息的接口。仅在 RCChannelClient 类中提供。支持的会话类型包括单聊、群聊、聊天室、超级群。
> - 如果 SDK 版本低于 5.2.5.1，可以使用 RCCoreClient 类中的 getMessageByUId 方法，该方法一次仅支持传入一个 UID。

消息的 UID 是由融云服务端生成的全局唯一 ID。消息存入本地数据库后，App 可能需要再次提取特定消息。例如，您的用户希望收藏聊天记录中的部分消息，App 可以先记录消息的 UID，在需要展示时调用 getBatchLocalMessages，传入收藏消息的 UID，从本地数据库中提取消息。

```
[[RCChannelClient sharedChannelManager] getBatchLocalMessages:conversationType_PRIVATE
targetId: @"targetId"
channelId: nil
messageUIDs:@[@"C3GC-8VAA-LJQ4-TPPM", @"B5GC-3VAA-LJQ4-TXXA"]]
success:^(NSArray<RCMessage *> *messages, NSArray<NSString *> *mismatch)) {}
error:^(RCErrorCode status) {};
```

只要持有消息 UID（messageUIDs），并且本地数据库中已存有消息，即可以使用该方法从本地数据库提取消息。单次仅可从一个会话（targetId）或超级群频道（channelId）中提取消息。

参数	类型	说明
conversationType	RCConversationType	会话类型，支持单聊、群聊、聊天室、超级群。
targetId	NSString	会话 ID
channelId	NSString	超级群的频道 ID。非超级群会话类型时，传入 null。
messageUIDs	NSArray	消息的 UID，即由融云服务端生成的全局唯一 ID。必须传入有效的 UID，最多支持 20 条。
successBlock	Block	成功回调。返回结果中包含消息对象列表和提取失败的消息的 UID 列表。
errorBlock	Block	失败回调。

在会话类型为超级群或聊天室时，请注意以下情况：

- 超级群会话默认只同步会话最后一条消息。如果您直接调用该方法（例如您传入了通过融云服务端回调全量消息路由获取的 UID），可能 SDK 无法在本地找到对应消息。建议先调用 getBatchRemoteUltraGroupMessages 从服务端获取消息。
- 聊天室会话自动在用户退出清空本地消息。如果用户退出聊天室后再调用该接口，则无法取得本地消息。

获取指定时间戳前后的 N 条消息

您可以通过 getHistoryMessages 方法，获取指定时间戳前后的 N 条本地消息。如果指定的消息数量大于实际的消息数，则返回实际数量的消息。

该方法将返回一个包含 RCMessage 对象的列表。

```
[[RCCoreClient sharedCoreClient] getHistoryMessages:ConversationType_GROUP targetId:@"g1" sentTime:sentTime beforeCount:10
afterCount:10 completion:^(NSArray<RCMessage *> * _Nullable messages) {

}];
```

在此方法中，beforeCount 参数用于指定您希望获取的、发送时间早于 sentTime 的消息数量；afterCount 参数则用于指定您希望获取的、发送时间晚于 sentTime 的消息数量。sentTime 参数是确定消息检索范围的关键时间点。每次调用 getHistoryMessages 方法时，将获取到会话中发送时间为 sentTime 的消息，以及该消息之前 beforeCount 条和之后 afterCount 条的消息。

以下是参数的详细说明：

参数	类型	说明
conversationType	RCConversationType	指定的会话类型。
targetId	NSString	会话 ID。
sentTime	long long	消息的发送时间戳。
beforeCount	int	发送时间早于 sentTime 的消息数量。
afterCount	int	发送时间晚于 sentTime 的消息数量。
completion	Block	当消息检索完成时，将调用此回调函数，返回检索到的消息列表。

获取远端历史消息

使用 getRemoteHistoryMessages 方法可直接查询指定会话存储在单群聊消息云端存储中的历史消息。

> 提示
>
> 用户是否可以获取在加入群组之前的群聊历史消息取决于 App 在控制台的设置。您可以在控制台的免费基础功能页面，启用新用户获取加入群组前历史消息。启用此选项后，新入群用户可以获取在他们加入群组之前发送的所有群聊消息。如不启用，新入群用户只能看到他们入群后的群聊消息。

获取会话的远端历史消息

SDK 按照指定条件直接查询并获取单群聊消息云端存储中的满足查询条件的历史消息。查询结果与本地数据库对比，排除重复的消息后，返回消息对象列表。返回的消息列表中的消息按发送时间从新到旧排列。

因为默认该接口返回的消息会跟本地消息排重后返回，建议先使用 getHistoryMessages，在本地数据库消息全部获取完之后，再获取远端历史消息。否则可能会获取不到指定的部分或全部消息。

```
[[RCIMClient sharedRCIMClient] getRemoteHistoryMessages:ConversationType_PRIVATE targetId:@"targetId" recordTime:recordTime
count:count success:^(NSArray *messages, BOOL isRemaining) {

} error:^(RCErrorCode status) {

}];
```

count 参数表示返回列表中应包含多少消息。recordTime 参数用于控制分页的边界。每次调用 getRemoteHistoryMessages 方法时，SDK 会以 recordTime 为界，继续在下一页返回指定数量的消息。如果需要获取会话中最新的 count 条消息，可以将 recordTime 设置为 0。

建议获取返回结果中最早一条消息的 sentTime，并在下一次调用时作为 recordTime 的值传入，以便遍历整个会话的消息历史记录。

参数	类型	说明
conversationType	RCConversationType	会话类型

参数	类型	说明
targetId	NSString	会话 ID
recordTime	long long	时间戳，获取发送时间早于 `recordTime` 的历史消息。传 `0` 表示获取最新 `count` 条消息。默认值为 `0`。
count	int	要获取的消息数量。如果 SDK < 5.4.1，范围为 [2-20]；如果 SDK ≧ 5.4.1，范围为 [2-100]。
successBlock	Block	获取成功的回调
errorBlock	Block	获取失败的回调

- `success` 说明：

回调参数	回调类型	说明
messages	NSArray	获取到的历史消息数组
isRemaining	BOOL	是否还有剩余消息

- `error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

自定义获取会话的远端历史消息

通过 `RCRemoteHistoryMsgOption` 可以自定义 `getRemoteHistoryMessages` 方法获取远端历史消息的行为。SDK 会按照指定条件直接查询单群聊消息云端存储中的满足查询条件的历史消息。

`RCRemoteHistoryMsgOption` 中包含多个配置项，其中 `count` 与 `recordTime` 参数分别时获取历史消息的数量与分页查询时间戳。`order` 参数用于控制获取历史消息的时间方向，可选择获取早于或者晚于给定的 `recordTime` 的消息。`includeLocalExistMessage` 参数控制返回消息列表中是否需要包含本地数据库已存在的消息。

```
RCRemoteHistoryMsgOption *option = [RCRemoteHistoryMsgOption new];
option.recordTime = recordTime;
option.count = 10;
option.order = RCRemoteHistoryOrderDesc;
option.includeLocalExistMessage = NO;

[[RCIMClient sharedRCIMClient] getRemoteHistoryMessages:ConversationType_PRIVATE targetId:@"targetId" option:option
success:^(NSArray *messages, BOOL isRemaining) {

} error:^(RCErrorCode status) {

}];
```

`RCRemoteHistoryMsgOption` 默认按消息发送时间降序查询会话中的消息，且默认会并将查询结果与本地数据库对比，排除重复的消息后，再返回消息对象列表。在 `includeLocalExistMessage` 设置为 NO 的情况下，建议 App 层先使用 `getHistoryMessages`，在本地数据库消息全部获取完之后，再获取远端历史消息，否则可能会获取不到指定的部分或全部消息。

如需按消息发送时间升序查询会话中的消息，建议获取返回结果中最新一条消息的 `sentTime`，并在下一次调用时作为 `recordTime` 的值传入，以便遍历整个会话的消息历史记录。升序查询消息一般可用于跳转到会话页面指定消息位置后需要查询更新的历史消息的场景。

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
option	RCRemoteHistoryMsgOption	可配置的参数，包括拉取数量、拉取顺序等
successBlock	Block	获取成功的回调
errorBlock	Block	获取失败的回调

- RCRemoteHistoryMsgOption 说明：

参数	说明
recordTime	时间戳，用于控制分页查询消息的边界。默认值为 0。
count	要获取的消息数量。如果 SDK < 5.4.1，范围为 [2-20]；如果 SDK ≧ 5.4.1，范围为 [2-100]；默认值为 0，表示不获取。
order	拉取顺序。RCRemoteHistoryOrderDesc：降序，按消息发送时间递减的顺序，获取发送时间早于 recordTime 的消息，返回的列表中的消息按发送时间从新到旧排列。RCRemoteHistoryOrderAsc：升序，按消息发送时间递增的顺序，获取发送时间晚于 recordTime 的消息，返回的列表中的消息按发送时间从旧到新排列。默认值为 1。
includeLocalExistMessage	是否包含本地数据库中的已有消息。YES：包含，查询结果不与本地数据库排除重复，直接返回。NO：不包含，查询结果先与本地数据库排除重复，只返回本地数据库中不存在的消息。默认值为 NO。

- success 说明：

回调参数	回调类型	说明
messages	NSArray	获取到的历史消息数组
isRemaining	BOOL	是否还有剩余消息

- error 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

获取本地与远端历史消息

> 提示
>
> 该接口在 RCCoreClient 类中提供。注意：用户是否可以获取在加入群组之前的群聊历史消息取决于 App 在控制台的设置。您可以在控制台的免费基础功能页面，启用新用户获取加入群组前历史消息。启用此选项后，新入群用户可以获取在他们加入群组之前发送的所有群聊消息。如不启用，新入群用户只能看到他们入群后的群聊消息。

getMessages 方法与 getRemoteHistoryMessages 的区别是 getMessages 会先查询指定会话存储本地数据库的消息，当本地消息无法满足查询条件时，再查询在单群聊消息云端存储中的历史消息，以返回连续且相邻的消息对象列表。

通过 getMessages 的 RCHistoryMessageOption 配置，可以自定义 getMessages 方法获取远端历史消息的行为。

```
RCHistoryMessageOption *option = [[RCHistoryMessageOption alloc] init];
option.order = RCHistoryMessageOrderDesc;
option.count = 20;
option.recordTime = message.sentTime; // 如果获取最新的 20 条消息，可以传 0。
[[RCCoreClient sharedCoreClient] getMessages:ConversationType_PRIVATE targetId:@"会话 id" option:option complete:^(NSArray
*messages, RCErrorCode code) {
if (code == 0) {
// 成功
} else {
// 失败
}
}];
```

RCHistoryMessageOption 中包含多个配置项，count 参数表示返回列表中应包含多少消息。recordTime 参数用于控制分页的边界。每次调用 getMessages 方法时，SDK 会以 recordTime 为界，继续在下一页返回指定数量的消息。如果需要获取会话中最新的 count 条消息，可以将

recordTime 设置为 0。Order 参数用于控制获取历史消息的时间方向，可选择获取早于或者晚于给定的 recordTime 的消息。

RCHistoryMessageOption 默认降序查询会话中的消息，建议获取返回结果中最早一条消息的 sentTime，并在下一次调用时作为 recordTime 的值传入，以便遍历整个会话的消息历史记录。升序查询消息一般可用于跳转到会话页面指定消息位置后需要查询更新的历史消息的场景。

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
option	RCHistoryMessageOption	可配置的参数，包括拉取数量、拉取顺序等
complete	Block	获取消息的回调

- RCHistoryMessageOption 说明：

参数	说明
recordTime	时间戳，用于控制分页查询消息的边界。默认值为 0。
count	要获取的消息数量。如果 SDK < 5.4.1，范围为 [2-20]；如果 SDK ≧ 5.4.1，范围为 [2-100]；默认值为 0，表示不获取。
order	拉取顺序。RCHistoryMessageOrderDesc：降序，按消息发送时间递减的顺序，获取发送时间早于 recordTime 的消息，返回的列表中的消息按发送时间从新到旧排列。RCHistoryMessageOrderAsc： 升序，按消息发送时间递增的顺序，获取发送时间晚于 recordTime 的消息，返回的列表中的消息按发送时间从旧到新排列。默认值为降序。

下载媒体消息文件　　下载媒体消息文件

更新时间:2024-08-30

SDK 提供多媒体文件的下载功能。

媒体消息中的媒体文件

消息 RCMessage 对象的 content 属性中可能包含媒体消息内容，其中携带了媒体文件地址。常见的媒体消息内容如下：

- RCFileMessage：文件消息内容
- RCImageMessage：图片消息内容
- RCGIFMessage：GIF 消息内容
- RCHQVoiceMessage：高清语音消息
- RCSightMessage：小视频消息内容

在收到此类消息时，您可以使用 downloadMediaMessage 下载其中的媒体文件。

下载媒体消息中的媒体文件

> 提示
>
> SDK 版本 从 5.6.6 开始，提供传入 RCMessage 下载媒体文件的方法，同时废弃传入消息 ID 下载媒体文件的方法。

（SDK 版本 ≥ 5.6.6）下载媒体文件至本地后，消息内容体 RCMessageContent 中的媒体文件本地路径 localPath 会被更新。

```
[[RCCoreClient sharedCoreClient] downloadMediaMessage:message
progressBlock:^(int progress) {
// Update UI with download progress
}
successBlock:^(NSString *mediaPath) {
// Handle successful download
}
errorBlock:^(RCErrorCode errorCode) {
// Handle download error
}
cancelBlock:^{
// Handle download cancellation
}];
```

参数	类型	说明
message	RCMessage	消息对象。
progressBlock	Block	当前的下载进度的回调，返回 int 类型的下载进度，范围 [0, 100]。
successBlock	Block	下载消息成功的回调，返回 NSString 类型的本地路径，同时消息内容体中的 localPath 会被更新为返回的 mediaPath。
errorBlock	Block	下载消息失败的回调，返回错误码 RCErrorCode。
cancelBlock	Block	取消下载的回调。

（SDK 版本 < 5.6.6）下载媒体文件至本地后，消息内容体 RCMessageContent 中的媒体文件本地路径 localPath 会被更新。

```
[[RCCoreClient sharedCoreClient] downloadMediaMessage:messageId
progress:^(int progress) { }
success:^(NSString *mediaPath) { }
error:^(RCErrorCode errorCode) { }
cancel:^{ }];
```

暂停下载媒体消息中的媒体文件

> 提示
>
> 暂停下载功能仅在 5.6.6 及之后版本支持。

在下载过程中可以暂停下载。如需恢复下载，重新调用下载方法，支持断点续传。

```
[[RCCoreClient sharedCoreClient] pauseDownloadMediaMessage:message
successBlock:^{
// Handle pause success
}
errorBlock:^(RCErrorCode errorCode) {
// Handle pause error
}];
```

取消下载媒体消息中的媒体文件

> 提示
>
> SDK 版本 从 5.6.6 开始，提供异步方法，原同步方法废弃。

（SDK 版本 ≧ 5.6.6）异步取消下载媒体消息中的媒体文件。如果 RCErrorCode 为 -3/OPERATION_MEDIA_NOT_FOUND，表示下载任务已经结束，或者消息不存在。

```
[[RCCoreClient sharedCoreClient] cancelDownloadMediaMessage:message
successBlock:^{
// Handle cancellation success
}
errorBlock:^(RCErrorCode errorCode) {
// Handle cancellation error
}];
```

（SDK 版本 < 5.6.6）取消下载媒体消息中的媒体文件。返回 YES 表示取消成功。返回 NO 表示取消失败，即已经下载完成或者消息不存在。

```
BOOL cancelResult = [[RCCoreClient sharedCoreClient] cancelDownloadMediaMessage:messageId];
```

网络媒体文件

SDK 提供一个媒体文件下载器，可通过网络 URL 下载文件，不会操作消息体。

通过 URL 下载网络媒体文件

使用此方法通过 URL 下载文件，该方法不会操作消息体。返回的 mediaPath 会包含入参中指定的文件名。

```
[[RCCoreClient sharedCoreClient] downloadMediaFile:@"filename.png"
mediaUrl:@"http://remote.url"
progress:^(int progress) { }
success:^(NSString *mediaPath) { }
error:^(RCErrorCode errorCode) { }
cancel:^{ } ];
```

参数	类型	说明
fileName	mediaUrl	指定的文件名称，必须指定文件后缀，例如 `rongCloud.mov`。
mediaUrl	NSString	媒体文件远端 URL。
progressBlock	Block	当前的下载进度的回调，返回 `int` 类型的下载进度，范围 [0, 100]。
successBlock	Block	下载消息成功的回调，返回 `NSString` 类型的本地路径。
errorBlock	Block	下载消息失败的回调，返回错误码 RCErrorCode。
cancelBlock	Block	取消下载的回调。

通过文件的网络 URL 下载图片。此方法仅仅是文件下载器，不会操作消息体。SDK 会通过拼接输入参数中的会话类型、会话 ID、媒体类型生成文件名，在 `mediaPath` 中返回。该方法仅支持下载图片。

```
[[RCCoreClient sharedCoreClient] downloadMediaFile:ConversationType_GROUP
targetId:@"targetId"
mediaType:MediaType_IMAGE
mediaUrl:@"remoteurl"
progress:^(int progress) { }
success:^(NSString *mediaPath) { }
error:^(RCErrorCode errorCode) { }
cancel:^{ }];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
mediaType	RCMediaType	多媒体类型，该方法仅支持下载图片（`MediaType_IMAGE`）。
mediaUrl	NSString	媒体文件远端 URL。
progressBlock	Block	当前的下载进度的回调，返回 `int` 类型的下载进度，范围 [0, 100]。
successBlock	Block	下载消息成功的回调，返回 `NSString` 类型的本地路径。
errorBlock	Block	下载消息失败的回调，返回错误码 RCErrorCode。
cancelBlock	Block	取消下载的回调。

暂停下载网络媒体文件

> 提示
>
> 暂停下载功能仅在 5.6.6 及之后版本支持。

在下载过程中可以暂停下载。如需恢复下载，重新调用下载方法，支持断点续传。

```
[[RCCoreClient sharedCoreClient] pauseDownloadMediaUrl:mediaUrl
successBlock:^{
// Handle pause success
}
errorBlock:^(RCErrorCode errorCode) {
// Handle pause error
}];
```

取消下载网络媒体文件

> 提示
>
> SDK 版本 从 5.6.6 开始，提供异步取消下载方法，原同步方法废弃。

（SDK 版本 ≧ 5.6.6）异步取消下载网络媒体文件。如果 RCErrorCode 为 -3/OPERATION_MEDIA_NOT_FOUND，表示下载任务已经结束。

```
[[RCCoreClient sharedCoreClient] cancelDownloadMediaUrl:mediaUrl
successBlock:^{
// Handle cancellation success
}
errorBlock:^(RCErrorCode errorCode) {
// Handle cancellation error
}];
```

（SDK 版本 < 5.6.6）取消下载网络媒体文件。返回 YES 表示取消成功。返回 NO 表示取消失败，即已经下载完成或者消息不存在。

```
BOOL cancelResult = [[RCCoreClient sharedCoreClient] cancelDownloadMediaUrl:@"remoteurl"];
```

插入消息 插入消息

更新时间:2024-08-30

SDK 支持在本地数据库中插入消息。本地插入的消息不会实际发送给服务器和对方。

> 提示
>
> - 请确保所有插入消息的均为支持客户端本地存储的消息，否则会报错。详见了解消息存储策略。
> - 插入消息的接口仅将消息插入本地数据库，所以 App 卸载重装或者换端登录时插入的消息不会从远端同步到本地数据库。

插入发送消息

开发者可通过下面接口在本地数据库被插入一条对外发送的消息。不支持聊天室会话类型。以下示例使用 RCChannelClient 下的 insertOutgoingMessage 方法。

> 提示
>
> 如果 sentTime 有问题会影响消息排序，慎用！

```
RCTextMessage *content = [RCTextMessage messageWithContent:@"测试文本消息"];

[[RCChannelClient sharedChannelManager]
insertOutgoingMessage:ConversationType_PRIVATE
targetId:@"targetId"
channelId:@"channelId"
canIncludeExpansion:YES
sentStatus:SentStatus_SENT
content:content
sentTime:sentTime
completion:completion];
```

插入本地数据库的消息如需支持消息扩展功能，必须使用 RCChannelClient 下的插入方法，否则无法打开消息的可扩展属性（canIncludeExpansion）。本地插入消息时不支持设置消息扩展信息数据。App 可以选择合适的时机为消息设置扩展数据。详见消息扩展。

参数	类型	说明
conversationType	RCConversationType	会话类型。
targetId	NSString	会话 ID
channelId	NSString	所属会话的业务标识。
canIncludeExpansion	BOOL	是否支持消息扩展。YES表示可扩展；NO 表示不可扩展。
sentStatus	RCSentStatus	发送状态。
content	RCMessageContent	消息的内容。
sentTime	long long	消息发送的 Unix 时间戳，单位为毫秒（传 0 会按照本地时间插入）。
completion	Block	异步回调，返回已插入的消息实体 RCMessage。

插入接收消息

开发者可通过下面接口在本地数据库被插入一条接收的消息。

> 提示

如果 sentTime 有问题会影响消息排序，慎用！

```
RCTextMessage *content = [RCTextMessage messageWithContent:@"测试文本消息"];

[[RCCoreClient sharedCoreClient]
insertIncomingMessage:ConversationType_PRIVATE
targetId:@"targetId"
senderUserId:@"senderUserId"
receivedStatus:ReceivedStatus_READ
content:content
sentTime:sentTime
completion:completion];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	Target ID 用于标识会话，称为目标 ID 或会话 ID。注意，因为单聊业务中始终使用对端用户 ID 作为标识本端会话的 Target ID，因此在单聊会话中插入本端接收的消息时，Target ID 始终是单聊对端用户 ID。群聊、超级群的会话 ID 分别为群组 ID、超级群 ID。详见消息介绍中关于 Target ID 的说明。
senderUserId	NSString	发送者ID
sentStatus	RCReceivedStatus	接收状态
content	RCMessageContent	消息的内容
sentTime	long long	消息发送的Unix时间戳，单位为毫秒（传 0 会按照本地时间插入）
completion	Block	异步回调，返回已插入的消息实体 RCMessage

批量插入消息

提示

- 从 5.1.1 版本开始支持该功能。
- 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

在本地数据库批量插入消息。由于批量插入失败时数据会使数据整体插入失败，建议分批插入。该接口不支持聊天室会话类型，不支持超级群会话类型。

RCMessage 的下列属性会被入库，其余属性会被抛弃：

- `conversationType`：会话类型。
- `messageUId`：消息全局唯一 ID，消息成功收发后会带有由服务端生成的全局唯一 ID。SDK 从 5.3.5 开始支持入库该属性，该字段一般用于迁移数据。
- `targetId`：会话 ID。
- `messageDirection`：消息方向。
- `senderUserId`：发送者 ID。
- `receivedStatus`：接收状态；如果消息方向为接收方，并且 `receivedStatus` 为 `ReceivedStatus_UNREAD` 时，该条消息未读。未读消息数会累加到会话的未读消息数上。
- `sentStatus`：发送状态。
- `content`：消息的内容。
- `sentTime`：消息发送的 Unix 时间戳，单位为毫秒，会影响消息排序。
- `extra`：消息附加信息

```
/*!
异步批量插入接收的消息（该消息只插入本地数据库，实际不会发送给服务器和对方）

@discussion 此方法不支持聊天室的会话类型。不支持超级群会话类型。每批最多处理 500 条消息，超过 500 条返回 NO
@discussion 消息的未读会累加到会话的未读数上

@remarks 消息操作
*/
- (void)batchInsertMessage:(NSArray<RCMessage *> *)msgs completion:(nullable void(^)(BOOL))completion;
```

参数	类型	说明
msgs	NSArray	消息实体 RCMessage 对象列表。单次操作最多插入 500 条消息，建议单次插入 10 条。注意，`message` 中必须包含正确有效的 `sentTime`（消息发送的 Unix 时间戳，单位为毫秒），否则无法通过获取历史消息接口从数据库中获取该消息。
completion	Block	异步回调，返回是否插入成功

批量插入消息提供以下方法，设置 `checkDuplicate` 为 YES 后，SDK 会在插入时检查当前 UID 是否与数据库中 UID 重复，并移除重复项。注意，App 应自行保证插入的消息数组内部无重复的 UID。

```
- (void)batchInsertMessage:(NSArray<RCMessage *> *)msgs checkDuplicate:(BOOL)checkDuplicate completion:(nullable
void(^)(BOOL ret))completion;
```

删除消息 删除消息

更新时间:2024-08-30

针对单聊会话、群聊会话、系统会话，融云支持 App 用户通过客户端 SDK 删除自己的历史消息，支持仅从本地数据库删除消息、仅从融云服务端删除自己的历史消息、或从两处同时删除。

SDK 的删除消息操作（下表中的 API）均指从当前登录用户的历史消息记录中删除一条或一组消息，不影响会话中其他用户的历史消息记录。

功能	本地/服务端	API
仅从本地删除指定消息（消息 ID）	仅从本地删除	deleteMessages
仅从本地删除会话全部历史消息	仅从本地删除	deleteMessages
删除会话内指定消息（消息对象）	同时从本地和服务端删除消息	deleteRemoteMessage
删除会话历史消息（时间戳）	可选仅本地删除、或者同时从本地和服务端删除消息	clearHistoryMessages
仅从服务端删除会话历史消息（时间戳）	仅从服务端删除	clearRemoteHistoryMessages

提示

- App 用户的单聊会话、群聊会话、系统会话的消息默认仅存储在本地数据库中，仅支持从本地删除。如果 App（App Key/环境）已开通单群聊消息云端存储，该用户的消息还会保存在融云服务端（默认 6 个月），可从远端历史消息记录中删除消息。
- 针对单聊会话、群聊会话，如果通过任何接口以传入时间戳的方式删除远端消息，服务端默认不会删除对应的离线消息补偿（该机制仅会在打开多设备消息同步开关后生效）。此时如果换设备登录或卸载重装，仍会因为消息补偿机制获取到已被删除的历史消息。如需彻底删除消息补偿，请提交工单，申请开通**删除服务端历史消息时同时删除多端补偿的离线消息**。如果以传入消息对象的方式删除远端消息，则服务端一定会删除消息补偿中的对应消息。
- 针对单聊会话、群聊会话，如果 App 的管理员或者某普通用户希望在所有会话参与者的历史记录中彻底删除一条消息，应使用撤回消息功能。消息成功撤回后，原始消息内容会在所有用户的本地与服务端历史消息记录中删除。
- 客户端 SDK 不提供删除聊天室消息的接口。当前用户的聊天室本地消息在退出聊天室时会被自动删除。开通**聊天室消息云端存储服务**后，如需清除全部用户的聊天室历史消息，可使用服务端 API 清除消息。
- 客户端 SDK 提供删除超级群会话的消息的 API，详见「超级群管理」下的删除消息。

仅从本地删除指定消息（消息 ID）

App 用户可以按消息 ID 删除存储在本地数据库内的消息。待删除消息可以属于不同会话。

如果 App 已经开通**单群聊历史消息云存储**服务，服务端保存的该用户的历史消息记录不受影响。如果该用户从服务端获取历史消息，可能会获取到在本地已删除的消息。

```
BOOL success = [[RCIMClient sharedRCIMClient] deleteMessages: @[@(message.messageId)]];
```

删除时需要提供待删除消息 ID 数组。

参数	类型	说明
messageIds	NSArray<NSNumber *>	messageId 的列表，元素需要为 NSNumber 类型。详见消息介绍中的 `MessageId` 属性。

仅从本地删除会话全部历史消息

如果 App 用户希望本地清空自己的单聊、群聊或系统会话的历史记录，可以删除指定会话保存在本地数据库中的全部消息。如果 App 已经开通单群

聊历史消息云存储服务，服务端保存的该用户的历史消息记录不受影响。如果该用户从服务端获取历史消息，可能会获取到在本地已删除的消息。

该接口一次仅允许删除指定的单个会话的消息，会话由会话类型和会话 ID 参数指定。

```
BOOL success = [[RCIMClient sharedRCIMClient] clearMessages:ConversationType_PRIVATE
targetId:@"targetId"];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID

删除会话内指定消息（消息对象）

> 提示
>
> 如果 App 已经开通单群聊历史消息云存储服务，您可以调用以下接口删除 App 用户在融云服务端的历史消息记录。该接口同时从本地和服务端删除消息。

如果 App 用户希望从自己的单聊、群聊或系统会话的历史记录中彻底删除一条消息，可以使用 deleteRemoteMessage 从本地和服务端同时删除消息。删除成功后，该用户无法从本地数据库获取消息。如果从服务端获取历史消息，也无法获取到已删除的消息。

该接口允许一次删除指定的单个会话内的一条或一组消息。请确保所提供的消息均属于同一会话（由会话类型和会话 ID 指定）。

```
[[RCIMClient sharedRCIMClient]
deleteRemoteMessage:ConversationType_PRIVATE
targetId:@"targetId"
messages:messages
success:^{}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
messages	NSArray<RCMessage *>	将被删除的消息对象 RCMessage 列表
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 参数返回错误码 RCErrorCode。

删除会话历史消息（时间戳）

> 提示
>
> 该如果 App 已经开通单群聊历史消息云存储服务，您可以调用以下接口并通过参数指定需要同时删除 App 用户在融云服务端的历史消息记录。

如果 App 用户希望从自己的单聊、群聊或系统会话的历史记录中删除一段历史消息记录，可以使用 clearHistoryMessages 删除会话中早于某个时间点（recordTime）的消息。clearRemote 参数控制是否同时删除服务端对应的历史消息。

```
[[RCIMClient sharedRCIMClient]
clearHistoryMessages:ConversationType_PRIVATE
targetId:@"targetId"
recordTime:recordTime
clearRemote:YES
success:^{}
error:^(RCErrorCode status) {}];
```

如果 clearRemote 参数设置为 YES，表示需要同时删除服务端对应消息，该接口会从本地和服务端同时删除早于 （recordTime）的消息。服务端消息删除后，该用户无法再从服务端获取到已删除的消息。该接口一次仅允许删除指定的单个会话的消息，会话由会话类型和会话 ID 参数指定。

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
recordTime	long long	消息时间戳。默认删除小于等于 recordTime 的消息。如果传 0，则删除所有消息。
clearRemote	BOOL	是否同时删除服务端消息
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 参数返回错误码 RCErrorCode 。

仅从服务端删除会话历史消息（时间戳）

> 提示
>
> - 如果 App 已经开通单群聊历史消息云存储服务，您可以调用以下接口删除 App 用户在融云服务端的历史消息记录。该接口仅从服务端删除消息。
> - 使用融云即时通讯服务端 API 也可以直接删除服务端消息。具体操作请参阅服务端文档消息清除。

App 用户可以仅删除指定会话保存在服务端的历史消息。该接口提供时间戳参数（recordTime），支持删除早于指定时间的消息。如果 recordTime 设置为 0 则删除该会话保存在服务端的全部历史消息。

```
[[RCIMClient sharedRCIMClient]
clearRemoteHistoryMessages:ConversationType_PRIVATE
targetId:@"targetId"
recordTime:sentTime
success:^{}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
recordTime	long long	消息时间戳。默认删除小于等于 recordTime 的消息。如果传 0，则删除所有消息。
successBlock	Block	删除成功的回调
errorBlock	Block	删除失败的回调。status 参数返回错误码 RCErrorCode 。

撤回消息 撤回消息

更新时间:2024-08-30

您的用户通过 App 成功发送了一条消息之后，可能发现消息内容错误等情况，希望将消息撤回，同时从接收者的消息记录中移除该消息。您可以使用撤回消息功能实现该需求。

`recallMessage` 会替换聊天记录中的原始消息为一条 objectName 为 `RC:RcNtf` 的撤回通知消息（RCRecallNotificationMessage ）。同时消息接收方可通过 `RCIMClientReceiveMessageDelegate` 的 `messageDidRecall` 回调方法获取服务端通知，在收到对方已撤回通知时进行相应操作并刷新界面。

撤回通知消息的结构说明可参见服务端文档通知类消息格式。

默认情况下，融云对撤回消息的操作者不作限制。如需限制，可考虑以下方案：

- App 客户端自行限制撤回消息的操作者。例如，不允许 App 业务中的普通用户撤回他人发送的消息，允许 App 业务中的管理员角色撤回他人发送的消息。
- 如需避免用户撤回非本人发送的消息，可以提交工单 申请打开**IMLib SDK 只允许撤回自己发送的消息**。从融云服务端进行限制，禁止用户撤回非本人发送的消息。

IMLib 对于撤回消息并没有做时间限制，现在主流的社交软件都会进行撤回时间限制，建议开发者自行做撤回时间限制。

撤回消息

撤回指定消息，只有已发送成功的消息可被撤回。撤回成功结果中返回消息的 `messageId`，但此时该消息已被替换为撤回提示小灰条消息（`RCRecallNotificationMessage`）。您可以根据 `messageId` 从数据库中获取撤回提示小灰条消息并在界面展示。

```
[[RCIMClient sharedRCIMClient] recallMessage:msg pushContent:nil success:^(long messageId) {

} error:^(RCErrorCode errorcode) {

}];
```

参数	类型	说明
message	RCMessage	需要撤回的消息
pushContent	NSString	push 显示的内容
successBlock	Block	撤回成功的回调。返回撤回消息的 `messageId`，该消息已经变更为新的撤回提示小灰条消息。
errorBlock	Block	撤回失败的回调。`RCErrorCode` 为错误码。

如果需要通过消息 ID （数据库索引唯一值）或者 UID（服务器消息唯一 ID）撤回消息，可先通过以下方法获取待撤回的消息，再使用 `recallMessage` 撤回消息。

- getMessage
- getMessageByUId

撤回消息时携带用户信息（userInfo）和附加信息（extra）

某些情况下，您可能希望在撤回消息的同时附加一些用户信息和其他的一些额外信息。为了实现这个功能，您可以在调用撤回消息的 API 时，为消息内容中的 `senderUserInfo` 和 `extra` 字段设置相应的值。以下是如何在 Objective-C 代码中实现上述功能的示例：

```
// 获取消息实例。
RCMessage *msg = [[RCCoreClient sharedCoreClient] getMessage:messageId];
// 设置用户信息，包括用户 ID，用户名，和头像 URL。
msg.content.senderUserInfo = [[RCUserInfo alloc] initWithUserId:@"userId" name:@"name" portrait:@""];
// 设置附加信息。
msg.content.extra = @"user's extra information";
// 执行撤回操作，设置撤回后的消息推送内容为空，并定义成功和失败的回调。
[[RCIMClient sharedRCIMClient] recallMessage:msg pushContent:nil success:^(long messageId) {
// 撤回成功处理逻辑。
} error:^(RCErrorCode errorcode) {
// 撤回失败处理逻辑。
}];
```

撤回操作完成后，消息的 senderUserInfo 和 extra 字段将更新为撤回时所附带的信息。

监听消息被撤回事件

消息发送方调用 recallMessage 后，消息接收方可通过 RCIMClientReceiveMessageDelegate 的 messageDidRecall 回调方法获取服务端通知，在收到对方已撤回通知时进行相应操作并刷新界面，或进行相应处理。

实现此功能需要开发者遵守 RCIMClientReceiveMessageDelegate 协议。

设置代理委托

```
[[RCIMClient sharedRCIMClient] setReceiveMessageDelegate:self object:nil];
```

代理方法

消息接收方可通过下面方法监听到被撤回的消息。

```
@protocol RCIMClientReceiveMessageDelegate <NSObject>
/*!
消息被撤回的回调方法

@param message 被撤回的消息

@discussion 被撤回的消息会变更为RCRecallNotificationMessage，App需要在UI上刷新这条消息。
@discussion 和上面的 - (void)onMessageRecalled:(long)messageId 功能完全一致，只能选择其中一个使用。
*/
- (void)messageDidRecall:(RCMessage *)message;
@end
```

搜索消息　　搜索消息

更新时间:2024-08-30

SDK 提供了本地消息搜索功能，允许 App 用户通过关键词、用户 ID 等条件搜索指定的单个会话中的消息，支持按时间段搜索。消息搜索仅查询本地数据库中的消息，返回包含指定关键字或符合全部搜索条件的消息列表。

并非所有消息类型均支持关键字搜索：

- 内置的消息类型中文本消息（RCTextMessage），文件消息（RCFileMessage），和图文消息`RCRichContentMessage`类型默认实现了`RCMessageCoding`协议的`getSearchableWords`方法。
- 自定义消息类型也可以支持关键字搜索，需要您参考文档自行实现。详见自定义消息类型。

如何实现基于关键字的全局搜索：

1. 根据关键字搜索本地存储的全部会话，获取包含关键字的会话列表。
2. 根据搜索会话返回的会话列表数据，调用搜索单个会话的方法，搜索符合条件的消息。

搜索全部会话

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

按关键字搜索本地存储的所有会话，获取符合条件的搜索结果（RCSearchConversationResult）列表。

```
NSArray *conversationTypeList = @[@(ConversationType_PRIVATE)];
NSArray *objectNameList = @[@"RC:TXT"];

[[RCCoreClient sharedCoreClient] searchConversations:conversationTypeList
messageType:objectNameList
keyword:@"搜索的关键字"
completion:^(NSArray<RCSearchConversationResult *> * _Nullable results) {
//异步回调
}];
```

参数	类型	说明
conversationTypeList	NSArray	会话类型列表，包含 RCConversationType
objectNameList	NSArray	消息类型列表，默认仅支持内置类型`RC:TxtMsg`（文本消息）、`RC:FileMsg`（文件消息）、`RC:ImgTextMsg`（图文消息）。
keyword	NSString	关键字。不可为空。
completion	Block	异步回调。`results`中返回 RCSearchConversationResult 列表。

在指定单个会话中搜索

获取包含关键词的会话列表后，可以搜索指定单个会话中符合条件的消息。

根据关键字搜索消息

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

在本地存储中根据关键字搜索指定会话中的消息，支持搜索指定时间点之前的历史消息记录。回调中分页返回包含指定关键字的消息列表。

```
[[RCCoreClient sharedCoreClient] searchMessages:ConversationType_PRIVATE
targetId:@"会话 ID"
keyword:searchText
count:50
startTime:0
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
keyword	NSString	关键字。传空默认为是查全部符合条件的消息。
count	int	最大的查询数量
startTime	long long	询记录的起始时间。传 `0` 时从最新消息开始搜索。非 `0` 时从该时间往前搜索。
completion	Block	异步回调。`messages` 中返回匹配的 RCMessage 列表。

根据关键字搜索指定时间段的消息

> 提示
>
> 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

SDK 支持将关键字搜索的范围限制在指定时间段内。回调中分页返回包含指定关键字和时间段要求的消息列表。

```
[[RCCoreClient sharedCoreClient] searchMessages:ConversationType_PRIVATE
targetId:@"会话 ID"
keyword:searchText
startTime:startTime
endTime:endTime
offset:0
limit:100
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

`limit` 参数控制返回的搜索结果数量，取值范围为 [1-100]。超过 100 默认使用最大值 100。

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
keyword	NSString	关键字。传空默认为是查全部符合条件的消息。
startTime	long long	查询记录的起始时间。传 `0` 时从最新消息开始搜索。非 `0` 时从该时间往前搜索。
endTime	long long	结束时间
offset	int	偏移量。要求 ≥ 0。
limit	int	返回的搜索结果数量，limit 需大于 0。最大值为 100。超过 100 时默认返回 100 条。
completion	Block	异步回调。`messages` 中返回匹配的 RCMessage 列表。

根据用户 ID 搜索消息

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

在本地存储中根据搜索来自指定用户 ID 的消息，支持搜索指定时间点之前的历史消息记录。回调中分页返回包含符合条件的消息列表。

```
[RCCoreClient sharedCoreClient] searchMessages:ConversationType_PRIVATE
targetId:@"接收方 id"
userId:@"userId"
count:50
startTime:0
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID
userId	NSString	搜索用户ID
count	int	最大的查询数量
startTime	long long	查询记录的起始时间。传 `0` 时从最新消息开始搜索。非 `0` 时从该时间往前搜索。
completion	Block	异步回调。`messages` 中返回匹配的 RCMessage 列表。

在指定会话中的指定消息类型中搜索

您可以在指定会话中，按关键字搜索指定消息类型的历史消息，搜索结果将分页展示。

参数表格

参数	类型	说明
conversationIdentifier	RCConversationIdentifier	会话标识，用于指定要搜索的会话类型和会话ID的组合。
keyword	String	搜索的关键字，不能为空。
objectNameList	NSArray<NSString *>	需要索引的消息类型名列表，非空。每个消息类型名称是其 `getObjectName` 方法的返回值。
limit	int	最大的搜索数量，最大为 100，超过则使用 100。
startTime	long long	搜索起始时间点（毫秒），传 0 表示从最新消息开始搜索。
success	void (^)(NSArray<RCMessage *> * _Nonnull messages)	异步成功回调，返回匹配的消息列表。
error	void (^)(RCErrorCode status)	异步失败回调，返回错误码。

示例代码

```
// 创建会话标识，指定会话类型和目标会话ID。
RCConversationIdentifier *cider = [[RCConversationIdentifier alloc] initWithType:self.conversationType
targetId:self.targetId channelId:@""];

// 定义要搜索的消息类型列表，这里包括文本消息、图文消息和文件消息
NSArray *objnamelist = @[
[RCTextMessage getObjectName], // 文本消息类型
[RCRichContentMessage getObjectName], // 图文消息类型
[RCFileMessage getObjectName] // 文件消息类型
];

// 调用搜索方法，传入会话标识、搜索关键字、消息类型列表、结果数量上限、搜索起始时间点
[[RCCoreClient sharedCoreClient] searchMessages:cider
keyword:@"搜索的关键字" // 指定搜索的关键字
objectNameList:objnamelist // 指定要搜索的消息类型
limit:50 // 设置返回结果的数量上限为50
startTime:0 // 从最新消息开始搜索
success:^(NSArray<RCMessage *> * _Nonnull messages) {
// 搜索成功，处理返回的消息列表。
// 此处应添加处理消息列表的逻辑。
} error:^(RCErrorCode status) {
// 搜索失败，处理错误。
// 此处应添加错误处理逻辑。
}];
```

单群聊已读回执 单群聊已读回执

更新时间:2024-08-30

已读回执功能可以让发送方知道消息是否被对方已读。

- 在单聊场景下，当接收方已读消息后，可以主动发送已读回执给发送方，发送方通过监听已读回执消息来获得已读通知。
- 在群聊场景下，当一条消息发送到群组后，发送方可以在消息发送完成后主动发起已读回执请求。群组中的其他成员在读到这条消息后，可以对请求进行响应。发送方通过监听已读回执响应结果来获知哪些群成员已读了这条消息。

SDK 针对单聊、群聊已读回执相关的事件提供了不同的事件处理机制：

- 单聊已读回执采用观察者模式，应用程序需要接收由 SDK 分发的 RCLibDispatchReadReceiptNotification 通知。
- 群聊已读回执采用委托模式，使用接收消息的协议 RCIMClientReceiveMessageDelegate 中的代理方法。

单聊已读回执

单聊已读回执功能基于用户在 1 对 1 会话中上次阅读位置的消息的时间戳。通过将该时间戳传递给对方，对方可获得已发送的消息的阅读进度。SDK 定义了 RCSentStatus ，用于记录单聊会话中的一条消息是否已被对方阅读，应用程序可以通过访问本端发出的消息的发送状态（sentStatus）属性查询具体状态。

发送单聊已读回执

消息接收方在阅读过某条消息后，需要主动发送已读回执给发送方。调用 sendReadReceiptMessage:targetId:time:success:error: 方法时需要传入一个时间戳。可以传入指定消息的发送时间（message.sentTime），或者传入会话最后一条消息的发送时间（conversation.sentTime）。

```
[[RCCoreClient sharedCoreClient]
sendReadReceiptMessage:ConversationType_PRIVATE
targetId:@"targetId"
time:sentTime
success:nil
error:nil];
```

单聊已读回执实际是一条已读通知消息，由 SDK 内部构建并发出。

接收单聊已读回执

SDK 收到单聊消息的已读回执后，会读取其中携带的时间戳，将本地消息数据库中该单聊会话的早于该时间戳的所有消息的 sentStatus 属性改为 SentStatus_READ，同时 IMLib SDK 会分发一个 RCLibDispatchReadReceiptNotification 的通知。应用程序需要注册为已读回执通知的观察者，才能收取到该类通知。

这段代码将当前对象 (self) 注册为 RCLibDispatchReadReceiptNotification 通知的观察者。当收到此通知时，将调用 didReceiveReadReceiptNotification: 方法。

```
[[NSNotificationCenter defaultCenter]
addObserver:self
selector:@selector(didReceiveReadReceiptNotification:)
name:RCLibDispatchReadReceiptNotification
object:nil];
```

didReceiveReadReceiptNotification: 方法在收到读取回执通知时调用。该方法接受一个 NSNotification 对象作为参数。可以使用此对象来获取读取回执通知的信息。

```
- (void)didReceiveReadReceiptNotification:(NSNotification *)notification {

RCConversationType conversationType = (RCConversationType)[notification.userInfo[@"cType"] integerValue];
long long readTime = [notification.userInfo[@"messageTime"] longLongValue];
NSString *targetId = notification.userInfo[@"tId"];
NSString *senderUserId = notification.userInfo[@"fId"];
}
```

这段代码从通知的 `userInfo` 字典中获取以下信息：

- 会话类型（`conversationType`）
- 读取时间（`readTime`）
- 目标 ID（`targetId`）
- 发送者 ID（`senderUserId`）

上面的 `readTime` 即单聊已读回执消息中携带的时间戳。此时消息数据库内发送时间小于或等于 `readTime` 的消息的发送状态属性（`sentStatus`）均已被改为对方已读（`SentStatus_READ`）。应用程序在 UI 展示本端已发送的消息时，可以将早于该时间戳的消息均展示为对方已读。

群聊已读回执

群聊已读回执功能基于消息的全局唯一 ID（UID）。通过在群聊会话中传递消息的 UID，消息发送者可获得一条指定消息的阅读进度和已读用户列表。SDK 定义了 RCReadReceiptInfo，用于记录群聊会话中的一条消息群组已读回执数据，应用程序可以通过访问本端发出的消息的 `readReceiptInfo` 属性查询具体状态。

群聊已读回执的工作流程如下：

1. 群成员 Alice 往群组发送了一条消息。消息将到达即时通讯服务端，并由即时通讯服务端发送到目标群组中。
2. Alice 希望得知已阅读该条消息的群成员列表。作为该条消息的发送者，她可以发起群组消息已读回执请求。
3. 群组中其他成员监听到 Alice 发起的群消息已读回执请求。
4. 群成员 Bob 阅读了 Alice 发送的群消息，于是在群组中响应 Alice 的已读回执请求。
5. 作为消息发送者，Alice 可以通过监听接收该条消息的已读回执响应。收到 Bob 在群组中发送的响应后，即可调整消息已读数。

发起群聊已读回执请求

发送者向群组中发送消息后，必须先主动发起已读回执请求，获取群成员的主动响应，才能获取该条消息的阅读状态。在一个群组会话中，只有消息发送者可以发起群聊已读回执请求。

消息发送完成后，消息发送者使用 sendReadReceiptRequest:success:error: 方法，传入已发送的 RCMessage 对象，对这条消息发起已读回执请求，SDK 内部会构建并发出一条群聊已读回执请求消息。

```
[[RCCoreClient sharedCoreClient] sendReadReceiptRequest:message
success:^{
}error:^(RCErrorCode nErrorCode) {
}];
```

接收群聊已读回执请求

SDK 在接收消息的委托协议 RCIMClientReceiveMessageDelegate 中定义了与群聊已读回执的相关代理方法。

应用程序可以通过 `addReceiveMessageDelegate` 方法设置多个消息接收代理。

```
[[RCCoreClient sharedCoreClient] addReceiveMessageDelegate:self];
```

在任意群成员发起已读回执请求后，其他群成员会通过 RCIMClientReceiveMessageDelegate 的 onMessageReceiptRequest 代理方法收到已读回执请求。

```
/*!
请求消息已读回执（收到需要阅读时发送回执的请求，收到此请求后在会话页面已经展示该 messageUId 对应的消息或者调用
getHistoryMessages 获取消息的时候，包含此 messageUId 的消息，需要调用 sendMessageReadReceiptResponse
接口发送消息阅读回执）

@param messageUId 请求已读回执的消息ID
@param conversationType conversationType
@param targetId targetId
*/
- (void)onMessageReceiptRequest:(RCConversationType)conversationType
targetId:(NSString *)targetId
messageUId:(NSString *)messageUId;
```

onMessageReceiptRequest 方法中会返回消息的 UID，您可以通过 getMessageByUId:completion: 方法获取消息对象，用于后续响应已读回执请求。如果需要批量获取消息，可以使用 RCChannelClient 下的 getBatchLocalMessages:targetId:channelId:messageUIDs:success:error: 方法。

响应群聊已读回执请求

应用程序可以根据用户的阅读进度，对已读回执请求进行响应。如果一次接收到多个请求，可以批量响应已读回执请求，但一次只能批量响应同一个会话中的已读回执请求。

使用 sendReadReceiptResponse:targetId:messageList:success:error: 方法，传入会话类型、群组 ID 和 RCMessage 对象列表，发起已读回执响应。注意会话类型一定是 ConversationType_GROUP。

```
[[RCCoreClient sharedCoreClient] sendReadReceiptResponse:ConversationType_GROUP
targetId:@"targetId"
messageList:msgList
success:^{

}
error:^(RCErrorCode nErrorCode){

}];
```

接收群聊已读回执响应

群组内有用户响应成功后，请求发起方会通过 RCIMClientReceiveMessageDelegate 的 onMessageReceiptResponse 收到通知及响应结果。注意只有请求发起者可以收到响应。

```
/*!
消息已读回执响应（收到阅读回执响应，可以按照 messageUId 更新消息的阅读数）
@param messageUId 请求已读回执的消息ID
@param conversationType RCConversationType
@param targetId targetId
@param userIdList 已读userId列表，字典类型 key:userId,value:readTime，例如：{"uid100001":1683708891523}
*/
- (void)onMessageReceiptResponse:(RCConversationType)conversationType
targetId:(NSString *)targetId
messageUId:(NSString *)messageUId
readerList:(NSMutableDictionary *)userIdList;
```

应用程序可以从 onMessageReceiptResponse 方法中获取消息的 UID 和所有已响应的用户列表。在 UI 展示本端发送的该条消息时，可以展示已读人数和具体的已读用户列表。SDK 会在消息的 readReceiptInf）属性中存储该条群聊消息的已读回执数据。

消息扩展 消息扩展

更新时间:2024-08-30

消息扩展功能可为消息对象（RCMessage）增加基于 Key/Value 的状态标识。消息的扩展信息可在发送前、后设置或更新，可用于实现消息评论、礼物领取、订单状态变化等业务需求。

一条消息是否可携带或可设置扩展信息，由发送消息时 RCMessage 的可扩展（canIncludeExpansion）属性决定，该属性必须在发送前设置，发送后无法修改。单条消息单次最多可设置 20 个扩展信息 KV 对，总计不可超过 300 个扩展信息 KV 对。在并发情况下如出现设置超过 300 个的情况，超出部分会被丢弃。

为 RCMessage 消息对象添加的 Key、Value 扩展信息会被存储。如已开通历史消息云存储功能，从服务端获取的历史消息也会携带已设置的扩展信息。

> 提示
> - 消息扩展仅支持单聊、群聊、超级群会话类型。不支持聊天室和系统会话。
> - 4.x SDK 从 4.0.3 版本开始支持消息扩展功能。

实现思路

以订单状态变化为例，可通过消息扩展改变消息显示状态。以订单确认为例：

1. 当用户购买指定产品下单后，商家需要向用户发送订单确认信息。可在发送消息时，将消息对象中的 `canIncludeExpansion` 属性设置为可扩展，同时设置用于标识订单状态的 Key 和 Value。例如，在用户未确认前，可用一对 Key/Value 表示该订单状态为「未确认」。
2. 用户点击确认（或其他确认操作）该订单消息后，订单消息状态需要变更为「已确认」。此时，可通过 `updateMessageExpansion` 更新此条消息的扩展信息，标识为已确认状态，同时更改本地显示的消息样式。
3. 发送方通过消息扩展状态监听，获取指定消息的状态变化，根据最新扩展信息显示最新的订单状态。

消息评论、礼物领取可参照以上实现思路：

- **礼物领取**：可通过消息扩展改变消息显示状态实现。例如，向用户发送礼物，默认为未领取状态，用户点击后可设置消息扩展为已领取状态。
- **消息评论**：可通过设置原始消息扩展信息的方式添加评论信息。

打开消息的可扩展属性（仅发送前）

构建新消息后，设置 RCMessage 的 canIncludeExpansion 属性打开或关闭某条消息的可扩展属性。必须在发送消息前设置该属性。

```
RCTextMessage *txt = [RCTextMessage messageWithContent:text];

RCMessage *msg = [[RCMessage alloc] initWithType:self.conversationType targetId:self.targetId
direction:(MessageDirection_SEND) messageId:-1 content:txt];
msg.canIncludeExpansion = YES;
```

参数	类型	说明
canIncludeExpansion	BOOL	该条消息是否可扩展。YES：该消息允许设置扩展信息。NO：该消息不允许设置扩展信息。消息发送之后不允许再更改该开关状态。

如果 App 先在本地插入消息，再发送本地已插入的消息（例如 App 业务要求在发送前审核消息内容），则必须在插入消息时设置此属性（详见插入消息）。本地插入成功后返回的消息不支持再通过修改 canIncludeExpansion 属性修改可扩展属性开关状态。

设置消息扩展数据（仅发送前）

如果消息已打开可扩展属性，可调用 RCMessage 的 expansionDic 属性设置扩展数据。该接口仅可在消息发送前调用。

```
/*!
消息扩展信息列表

@discussion 扩展信息支持单聊、群组、超级群，其它会话类型不能设置扩展信息
@discussion 默认消息扩展字典 key 长度不超过 32 ，value 长度不超过 4096 ，单次设置扩展数量最大为 20，消息的扩展总数不能超过 300
*/
@property (nonatomic, strong) NSDictionary<NSString *, NSString *> *expansionDic;
```

以下示例为之前构建的消息 msg 设置了扩展信息数据，并通过 sendMessage 发送了这条文本消息。

```
msg.expansionDic = @{@"key1":@"value1",@"key2":@"value2"};

[[RCIMClient sharedRCIMClient] sendMessage:msg pushContent:nil pushData:nil successBlock:^(RCMessage *successMessage) {

} errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {

}];
```

参数	类型	说明
expansionDic	NSDictionary	消息扩展信息。单次最大设置扩展信息键值对 20 对。单条消息可设置最多 300 个扩展信息。 • Key 支持大小写英文字母、数字、 特殊字符`+ = - _` 的组合方式，不支持汉字。最大 32 个字符。 • （SDK < 5.2.0）Value 最大 64 个字符 • （SDK ≧ 5.2.0）Value 最大 4096 个字符

上例中发送的消息会携带 expansionDic 属性中的扩展数据。消息发送成功后，SDK 会将消息及扩展数据存入本地数据库。

如果发送的是本地数据库中已存在的消息（详见插入消息），请注意：

- （SDK ≧ 5.3.4）发送成功后 SDK 会刷新本地数据库中消息的扩展数据。
- （SDK < 5.3.4）发送成功后 SDK 无法刷新本地数据库中消息的扩展数据。建议 App 先发送消息，再通过调用 `updateMessageExpansion` 更新本地与远端的扩展信息。对端可通过监听收到扩展数据更新。

更新扩展数据

调用 RCIMClient 的 updateMessageExpansion 方法更新消息扩展信息。仅支持已打开可扩展属性的消息。更新消息扩展信息后，更新发起者应在成功回调里处理 UI 数据刷新，会话对端可通过消息扩展监听器收到更新的数据。

> 提示
>
> - 每次更新（或删除）消息扩展时，SDK 内部将向会话对端发送一条类型标识为 `RC:MsgExMsg` 的消息扩展信令消息。因此，频繁更新消息扩展会导致产生大量消息。
> - 每个终端在设置扩展信息时，如未达到上限都可以进行设置；在并发情况下，会出现设置超过 300 的情况，超出部分会被丢弃。

```
// 更新消息扩展信息
[[RCCoreClient sharedCoreClient] updateMessageExpansion:dic messageUId:message.messageUId success:^{
RCMessage *msg = [[RCCoreClient sharedCoreClient] getMessageByUId:message.messageUId];
// 更新发起者在这里处理更新扩展后的 UI 数据刷新
} error:^(RCErrorCode status) {
[self showToastMsg:[NSString stringWithFormat:@"msgUid:%@的KV更新失败%zd",message.messageUId,status]];
}];
```

参数	类型	说明
expansionDic	NSDictionary	要更新的消息扩展信息键值对。 • Key 支持大小写英文字母、数字、 特殊字符`+ = - _` 的组合方式，不支持汉字。最大 32 个字符。 • （SDK < 5.2.0）Value 最大 64 个字符 • （SDK ≧ 5.2.0）Value 最大 4096 个字符
messageUId	NSString	消息 messageUId
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。`status` 包含失败的错误码 RCErrorCode

删除扩展数据

消息发送后调用 `RCIMClient` 的 `removeMessageExpansionForKey` 方法删除消息扩展信息中特定的键值对。仅支持已打开可扩展属性的消息。删除消息扩展信息后，发起者应在成功回调里处理删除后的 UI 数据刷新，会话对端可通过消息扩展监听器收到通知。

```
[[RCIMClient sharedRCIMClient] removeMessageExpansionForKey:keyArray messageUId:messageUId success:^{

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
keyArray	NSArray	消息扩展信息中待删除的 key 的列表
messageUId	NSString	消息 messageUId
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。`status` 包含失败的错误码 RCErrorCode

监听消息扩展数据变更

消息扩展变更发起方调用 API 更新、删除扩展数据后，接收方可通过 `RCMessageExpansionDelegate` 协议中的代理方法监听扩展数据变更，并进行相应处理。

实现此功能需要开发者遵守 `RCMessageExpansionDelegate` 协议。

```
[RCIMClient sharedRCIMClient].messageExpansionDelegate = self;
```

代理方法

```objc
@protocol RCMessageExpansionDelegate <NSObject>
/**
消息扩展信息更改的回调

@param expansionDic 消息扩展信息中更新的键值对
@param message 消息

@discussion expansionDic 只包含更新的键值对，不是全部的数据。如果想获取全部的键值对，请使用 message 的 expansionDic 属性。
*/
- (void)messageExpansionDidUpdate:(NSDictionary<NSString *, NSString *> *)expansionDic
message:(RCMessage *)message;

/**
消息扩展信息删除的回调

@param keyArray 消息扩展信息中删除的键值对 key 列表
@param message 消息

*/
- (void)messageExpansionDidRemove:(NSArray<NSString *> *)keyArray
message:(RCMessage *)message;

@end
```

自定义消息类型 自定义消息类型

更新时间:2024-08-30

除了使用 SDK 内置消息外，还可以创建自定义消息类型。

您需要根据业务需求来选择自定义消息类型继承的消息基类：

- `RCMessageContent`，即普通类型消息内容。例如在 SDK 内置消息类型中的文本消息和位置消息。
- `RCMediaMessageContent`，即多媒体类型消息。多媒体类型消息内容继承自 `RCMessageContent`，并在其基础上增加了对多媒体文件的处理逻辑。在发送和接收消息时，SDK 会判断消息类型是否为多媒体类型消息，如果是多媒体类型，则会触发上传或下载多媒体文件流程。

关于消息实体类与消息内容的更多介绍，可参见消息介绍。

创建自定义消息类型

自定义消息类型类遵守下面三个协议：

- 编解码协议（必须遵守）：`RCMessageCoding` 指定消息类型名称、消息收发过程中的编码与解码行为，以及提供消息搜索所需的关键词。该协议是所有自定义消息类型必须实现的协议，否则将无法正常传输和使用。
- 存储协议（必须遵守）：`RCMessagePersistentCompatible` 指定此消息类型在客户端与服务端是否计入未读消息数、是否存储的行为。
- 内容摘要协议（非必须遵守）：`RCMessageContentView` 设置如何显示该类型消息的摘要。

编解码协议

> 提示
>
> 任何自定义消息类型都必须实现 RCMessageCoding 协议，否则将无法正常传输和使用。

协议原型：

```
@protocol RCMessageCoding <NSObject>
```

编解码协议（RCMessageCoding）主要有以下功能：

- 提供该自定义消息类型的唯一 ID。
- 在消息发送时，将消息中的所有信息编码为 JSON 数据传输。
- 在消息接收时，将 JSON 数据解码还原为消息对象。
- 提供消息搜索数据。

遵守编解码协议需实现下面的方法：

- **序列化**：消息内容通过此方法，将消息中的所有数据，编码成为 JSON 数据，返回的 JSON 数据将用于网络传输。

```
- (NSData *)encode;
```

- **反序列化**：网络传输的 JSON 数据，会通过此方法解码，获取消息内容中的所有数据，生成有效的消息内容。

```
- (void)decodeWithData:(NSData *)data;
```

- **定义消息类型名**：定义的消息类型名，需要在各个平台上保持一致，否则消息无法互通。为避免和 SDK 默认的消息类型名称冲突，请勿使用 RC: 开头的类型名。

```
+ (NSString *)getObjectName;
```

- **提供消息搜索数据**：如果需要自定义消息需要被搜索，需要将关键字返回。如果不需要被搜索，可直接返回 nil。

```
- (NSArray<NSString *> *)getSearchableWords;
```

存储协议

存储协议（RCMessagePersistentCompatible）为必须遵守的协议。

协议原型：

```
@protocol RCMessagePersistentCompatible <NSObject>
```

存储协议主要有两个功能:

- 指明此消息类型在本地和服务端是否存储
- 指明此消息类是否计入未读消息数

遵守存储协议需实现控制消息的存储计数策略的方法：

```
+ (RCMessagePersistent)persistentFlag;
```

persistentFlag 属性说明	客户端是否存储	服务端是否存储	是否计入消息未读数
MessagePersistent_NONE	客户端不存储	支持离线消息机制	不计入未读消息数
MessagePersistent_ISCOUNTED	客户端存储	支持离线消息机制，且存入服务端历史消息	计入未读消息数
MessagePersistent_ISPERSISTED	客户端存储	支持离线消息机制，且存入服务端历史消息	不计入未读消息数
MessagePersistent_STATUS	客户端不存储	服务端不存储	不计入未读消息数

> 提示
>
> - `MessagePersistent_NONE` 一般用于需要确保收到，但不需要展示的消息，例如运营平台向终端发送的指令信息。如果消息接收方不在线，再次上线时可通过离线消息收到。
> - `MessagePersistent_STATUS` 用于状态消息。状态消息表示的是即时的状态，例如输入状态。因为状态消息在客户端与服务端均不会存储，如果接收方不在线，则无法再收到该状态消息。

内容摘要协议

内容摘要协议（RCMessageContentView）为非必须遵守的协议。

协议原型：

```
@protocol RCMessageContentView
```

消息的内容摘要显示在以下几处：

- 在会话列表中
- 本地通知中

遵守内容摘要协议需实现设置消息摘要的方法：

```
- (NSString *)conversationDigest;
```

注册自定义的消息类型

在进行完自定义消息类后，需要在 SDK init 之后 connect 之前，注册此自定义消息类。

> 提示
>
> 只有注册了该消息类型之后，SDK 才能正确识别和编码、解码该类型的消息。

```
- (void)registerMessageType:(Class)messageClass;
```

发送自定义消息

自定义消息类型可直接使用发送内置消息类型的方法。请注意根据当前使用的 SDK、业务、消息类型选择合适的核心类与方法：

- 如果自定义消息类型继承 `RCMessageContent`，请使用发送普通消息的接口发送。
- 如果自定义消息类型继承 `RCMediaMessageContent`，请使用发送媒体消息的接口发送。

如果自定义消息类型需要支持推送，必须在发送自定义消息时额外指定推送内容（pushContent）。推送内容在接收方收到推送时显示在通知栏中。

- 在发送消息时，可直接通过 `pushContent` 参数指定推送内容。
- 您也可以通过设置 RCMessage 的 `messagePushConfig` 中的 `pushContent` 及其他字段，对消息的推送进行个性化配置。优先使用 `messagePushConfig` 中的配置。详见配置消息的推送属性。

发送消息的具体方法与配置方式，请参考以下文档：

- **App 仅集成 IMLib SDK**：发送消息（单聊、群聊、聊天室）、收发消息（超级群）
- **App 集成 IMKit SDK**：发送消息（单聊、群聊、聊天室）

> 提示
>
> - 如果融云服务端无法获取自定义消息的 `pushContent`，则无法触发消息推送。例如，在接收方在离线等情况无法收到消息推送通知。
> - 如果自定义的消息类型为状态消息（`MessagePersistent_STATUS`），则无法支持推送，不需要额外指定推送内容。

实例

自定义消息范例

管理离线消息存储配置 管理离线消息存储配置

更新时间:2024-08-30

即时通讯业务支持修改 App 级别与用户级别的离线消息配置。

> 提示
>
> 离线消息配置仅适用于单聊、群聊。聊天室、超级群因业务特性不支持离线消息，因此无离线消息配置。

了解离线消息

离线消息[?]是指当用户不在线时收到的消息。融云服务端会自动为用户保留离线期间接收的消息，默认的离线消息保留时长为 7 天。7 天内客户端如果上线，服务端会直接将离线消息发送到该接收端。如果 7 天内客户端都没有上线，服务端将抛弃掉过期的消息。

即时通讯业务下并非所有会话类型都支持离线消息：

- **支持离线消息**：单聊、群聊、系统消息
- **不支持离线消息**：聊天室、超级群

App 级别离线消息配置

如需修改 App 级别设置，请提交工单。

App 级别的离线消息配置如下：

- 单聊离线消息存储时长：默认存储 7 天。设置范围为 1 - 7 天。配置修改将影响 App 下所有单聊会话。
- 群聊离线消息存储时长：默认存储 7 天。设置范围为 1 - 7 天。配置修改将影响 App 下所有群聊会话。
- 群组离线消息存储数量：默认存储 7 天内的所有群消息。配置修改将影响 App 下所有群聊会话。

用户级别离线消息配置

> 提示
>
> 设置、获取用户的离线消息存储时长功能均要求已开通用户级别功能设置。如需开通，请提交工单。

即时通讯业务支持用户级别的离线消息配置，仅支持修改离线消息存储时长。未修改的情况下，用户的离线消息存储时长为 7 天。设置范围为 1 - 7 天。

App Key 开通用户级别功能设置功能后，客户端 SDK 支持修改当前登录用户的离线消息存储时长。

设置用户的离线消息存储时长

设置当前用户的离线消息存储时长，以天为单位。

```
[[RCCoreClient sharedCoreClient] setOfflineMessageDuration:7
success:^{

} failure:^(RCErrorCode nErrorCode) {

}];
```

参数	类型	说明
duration	int	离线消息存储时长，范围为 1-7天。
successBlock	Block	设置成功的回调。
errorBlock	Block	设置失败的回调。nErrorCode 返回错误码 RCErrorCode 。

获取用户的离线消息存储时长

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取当前用户的离线消息存储时长，以天为单位。

```
[[RCCoreClient sharedCoreClient] getOfflineMessageDuration:^(int duration) {
//成功
}
```

会话介绍 会话介绍

更新时间:2024-08-30

会话是指融云 SDK 根据每条消息的发送方、接收方以及会话类型等信息，自动建立并维护的逻辑关系，是一种抽象概念。

会话类型

融云支持多种会话类型，以满足不同业务场景需求。客户端 SDK 通过 RCConversationType 枚举来表示各类型会话，各枚举值代表的含义参考下表：

枚举值	会话类型
ConversationType_PRIVATE	单聊会话
ConversationType_GROUP	群组会话
ConversationType_ULTRAGROUP	超级群会话
ConversationType_CHATROOM	聊天室会话
ConversationType_SYSTEM	系统会话

RCConversationType 枚举中还定义了其它会话类型，目前已废弃，不再维护。

单聊会话

指两个用户一对一进行聊天，两个用户间可以是好友也可以是陌生人，融云不对用户的关系进行维护管理，会话关系由融云负责建立并保持。

单聊类型会话里的消息会保存在客户端本地数据库中。

群组会话

群组指两个以上用户一起进行聊天，群组成员信息由 App 提供并进行维系，融云只负责将消息传达给群组中的所有用户。每个群最大人数上限为 3000 人，App 内的群组数量没有限制。

群组类型会话里的消息会保存在客户端本地数据库中。

超级群会话

超级群会话指无成员上限的多人聊天服务，海量消息并发即时到达，支持消息推送服务。群组成员信息由 App 提供并维系，融云负责将消息传达给群成员。App 内超级群数量没有限制，超级群无人数上限，一个用户可加入 100 个超级群。

超级群类型会话里的消息会保存在客户端本地数据库中，更多内容请参见超级群概述。

聊天室会话

聊天室成员不设用户上限，海量消息并发即时到达，用户退出聊天室后不会再接收到任何聊天室中的消息，没有推送通知功能。会话关系由融云负责建立并保持连接，通过 SDK 相关接口，可以让用户加入或者退出聊天室。

SDK 不保存聊天室消息，在退出聊天室时会清空此聊天室所有数据，更多内容请参见聊天室概述。

系统会话

系统会话是指利用系统帐号向用户发送消息从而建立的会话关系，此类型会话可以是通过调用广播接口发送广播来建立，也可以是加好友等单条通知消息而建立的会话。

会话实体类

客户端 SDK 中封装的会话实体类是 RCConversation，所有会话相关的信息都从该实体类中获取。

下表列出了 RCConversation 中提供的主要属性。

属性	类型	描述
targetId	NSString	会话 ID（或目标 ID），用于标识会话对端。 • 单聊时，会话 ID 直接使用对方的用户 ID。 • 在群组、聊天室、超级群中，为对应的群组、聊天室、超级群 ID。 • 系统会话中，为开发者指定的系统账号 ID。
channelId	NSString	该会话的业务标识，长度限制为 20 字符。仅适用于超级群。
conversationTitle	NSString	会话标题。
conversationType	RCConversationType	会话类型，参考上文详细描述。
unreadMessageCount	Int	会话中未读消息数。
isTop	BOOL	会话是否置顶。
lastestMessage	RCMessageContent	会话中在客户端本地存储的最后一条消息的消息内容。关于消息的存储属性请参考 消息介绍 中的说明。
lastestMessageId	long	会话中最后一条在客户端本地存储的消息的 ID。
draft	NSString	会话里保存的草稿信息，参考草稿详细说明。
objectName	NSString	会话中最后一条消息的类型名，与消息内容体对应。预定义消息类型的 objectName 参见消息类型概述。自定义消息类型的 ObjectName 为您自行指定的值。
receivedTime	long long	会话最后一条消息接收时间。 1. 返回值为 unix 时间戳，单位毫秒。 2. 接收时间为消息到达接收端时客户端的本地时间。
sentTime	long long	会话最后一条消息发送时间，为 Unix 时间戳，单位毫秒 1. 当会话里最后一条消息为发送成功或者接收到的消息时，此方法返回的是该消息到达融云服务器的时间。 2. 当会话里最后一条消息为发送失败的消息时，此方法返回此条消息的本地发送时间。 3. 当会话有草稿信息，且草稿保存时间大于最后一条消息时间时，此方法返回草稿保存时间。
receivedStatus	RCReceivedStatus	会话中最后一条消息的接收状态。
sentStatus	RCSentStatus	会话中最后一条消息的发送状态。
senderUserId	NSString	会话中最后一条消息发送者 ID。
lastestMessageDirection	RCMessageDirection	会话中最后一条消息的方向，分为发送和接收。
lastestMessageUId	NSString	会话中最后一条消息的唯一 ID。 1. 只有发送成功的消息才有唯一 Id。 2. 在同一个 Appkey 下全局唯一。
latestMessageReadReceiptInfo	RCReadReceiptInfo	会话中最后一条消息的阅读回执状态，仅适用于群聊。
latestMessageMessageConfig	RCMessageConfig	会话中最后一条消息的配置信息。
latestMessageCanIncludeExpansion	BOOL	会话中最后一条消息是否可以包含扩展信息。 1. 该属性在消息发送时确定，发送之后不能再做修改； 2. 扩展信息只支持单聊、群组和超级群，其它会话类型不能设置扩展信息。
latestExpansion	NSDictionary<NSString *, NSString *>	会话中最后一条消息的扩展信息列表，详情请参考消息扩展。
hasUnreadMentioned	BOOL	会话中是否存在被 @ 的消息
mentionedCount	int	本会话里自己被 @ 的消息数量
blockStatus	RCConversationNotificationStatus	会话的免打扰状态
notificationLevel	RCPushNotificationLevel	会话的免打扰级别，详见免打扰功能概述。
channelType	RCUltraGroupChannelType	超级群频道类型，只有 ConversationType 为超级群时该字段有效，非超级群时为 0
firstUnreadMsgSendTime	long long	从 5.2.5 开始，支持会话中第一条未读消息发送时间属性，为 Unix 时间戳，单位毫秒。仅支持超级群会话。

获取会话　　获取会话

更新时间:2024-08-30

客户端 SDK 会根据用户收发的消息，在本地数据库中生成对应会话，并维护会话列表。应用程序可以获取本地数据库中的会话列表。

获取指定单个会话

使用 getConversation:targetId:completion: 获取某个会话的详细信息。

```
[[RCCoreClient sharedCoreClient] getConversation:ConversationType_PRIVATE
targetId:@"targetId"
completion:^(RCConversation *conversation) {

}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID

批量获取会话信息

除了单个获取会话信息外，客户端 SDK 还支持批量获取会话的详细信息。

> 注意：
> 客户端 SDK 从 5.8.2 版本开始支持批量获取会话信息。
> 支持的会话类型：单聊、群聊、系统。

使用 getConversations:success:error: 方法查询多个会话的详细信息。如果本地数据库中只有部分会话信息，successBlock 结果只返回本地存在的会话；如果查询的多个会话信息在本地都查不到，将触发 errorBlock 回调。

```
// 假设我们有会话标识 conIden1 和 conIden2，它们代表想要查询的会话。
RCConversationIdentifier *conIden1 = [[RCConversationIdentifier alloc]
initWithConversationIdentifier:ConversationType_PRIVATE targetId:@"tId1"];

RCConversationIdentifier *conIden2 = [[RCConversationIdentifier alloc]
initWithConversationIdentifier:ConversationType_PRIVATE targetId:@"tId2"];

// 创建包含会话标识符的数组。
NSArray *conversationIdentifiers = @[
conIden1,
conIden2
// 如果有更多的会话标识符，可以继续添加到这个数组中。
];

// 使用 getConversations:success:error: 方法批量获取会话信息
[[RCCoreClient sharedCoreClient] getConversations:conversationIdentifiers
success:^(NSArray<RCConversation *> * _Nonnull conversations) {
// 成功获取会话信息的处理逻辑。
}
error:^(RCErrorCode status) {
// 错误处理逻辑，status 为错误码。
// 根据 status 进行相应的错误处理。
NSLog(@"Error fetching conversations: %@", @(status));
}];
```

参数	类型	说明
conversationIdentifiers	NSArray	需要查询的会话标识符数组，每次获取会话个数最大为 100。
success	Block	获取会话信息成功的回调，返回包含会话信息的数组。
error	Block	获取会话信息失败的错误回调，返回错误码。

获取会话列表

会话列表是 SDK 在本地生成和维护的。如果未发生卸载重载或换设备登录，您可以获取本地设备上存储的所有历史消息生成的会话列表。

分页获取会话列表

使用 getConversationList:count:startTime:completion: 分页获取 SDK 在本地数据库生成的会话列表。返回的会话列表按照时间倒序排列。如果返回结果含有被设置为置顶状态的会话，则置顶会话默认排在最前。

时间戳（`startTime`）首次可传 `0`，后续可以使用返回的 RCConversation 对象的 `operationTime` 属性值为下一次查询的 `startTime`。

> 提示
>
> 从 5.6.8 版本开始，需要使用会话的 `operationTime`；5.6.8 之前的版本，还需使用会话的 `sentTime`。

```
[[RCCoreClient sharedCoreClient] getConversationList:@[@(ConversationType_PRIVATE)]
count:100
startTime:0
completion:^(NSArray<RCConversation *> *conversationList) {
}];
```

如果希望返回的会话列表严格按照时间倒序排列，请使用带 `topPriority` 参数的方法，并将该参数设置为 NO。该方法仅在 5.6.9 及之后版本提供。

```
[[RCCoreClient sharedCoreClient] getConversationList:@[@(ConversationType_PRIVATE)]
count:100
startTime:0
topPriority:NO
completion:^(NSArray<RCConversation *> *conversationList) {
}];
```

参数	类型	说明
conversationTypeList	NSArray	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array。
count	int	获取的数量
startTime	long long	指定时间戳，以获取早于这个时间戳的会话列表。首次可传 0，表示从最新开始获取。后续使用真实时间戳。
topPriority	boolean	是否优先显示置顶消息。要求 SDK 版本 ≧ 5.6.9。
completion	Block	获取会话列表成功的回调

获取未读会话列表

> 提示
>
> SDK 从 5.3.2 版本开始提供该接口。

使用 getUnreadConversationList:completion: 获取指定类型的含有未读消息的会话列表，支持单聊、群聊、系统会话，返回 RCConversation 列表，获取到的会话列表按照时间倒序排列，置顶会话会排在最前。

```
[[RCCoreClient sharedCoreClient]
getUnreadConversationList:@[@(ConversationType_GROUP),@(ConversationType_SYSTEM),@(ConversationType_PRIVATE)]
completion:^(NSArray<RCConversation *> * _Nullable conversationList) {

}];
```

参数	类型	说明
conversationTypeList	NSArray	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array
completion	Block	获取会话列表成功的回调

卸载重装或换设备登录后的处理方案

如果您的用户卸载重装或换设备登录，可能会发现会话列表为空，或者有部分会话丢失的错觉。

原因如下：

- 在卸载的时候会删除本地数据库，本地没有任何历史消息，导致重新安装后会话列表为空。
- 如果换设备登录，可能本地没有历史消息数据，导致会话列表为空。
- 如果您的 App Key 开启了多设备消息同步 功能，服务端会同时启用离线消息补偿功能。服务端会在 SDK 连接成功后自动同步当天 0 点后的消息，客户端 SDK 接收到服务端补偿的消息后，可生成部分会话和会话列表。与卸载前或换设备前比较，可能会有部分会话丢失的错觉。

如果您希望在卸载重装或换设备登录后，获取到之前的会话列表，可以参考如下方案：

- 申请增加离线消息补偿的天数，最大可修改为 7 天。**注意，设置时间过长，当单用户消息量超大时，可能会因为补偿消息量过大，造成端上处理压力的问题。**如有需要，请提交工单 。
- 在您的服务器中自行维护会话列表，并通过 API 向服务端获取需要展示的历史消息。

处理会话未读消息数　　处理会话未读消息数

更新时间:2024-08-30

即时通讯客户端常常需要对会话进行未读消息计数。

您可以使用 IMLib SDK 提供的接口直接获取会话中的未读消息数。具体能力如下：

- 获取所有会话（不含聊天室）中的未读消息总数（`getTotalUnreadCountWith`）
- 获取指定会话中的总未读消息数，或指定会话中指定消息类型的总未读消息数，或按会话类型获取总未读消息总数（`getUnreadCount`）

在用户使用您的 App 时，UI 上未读计数可能需要发生变化，此时您可以清除会话中的未读数（clearMessagesUnreadStatus）。

获取所有会话总未读消息数

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

您可以使用 getTotalUnreadCountWith 获取所有类型会话（不含聊天室）中未读消息的总数。获取成功后，会返回未读消息数（unreadcount）。

```
[[RCCoreClient sharedCoreClient] getTotalUnreadCountWith:^(int unreadCount) {

}];
```

获取指定会话的总未读消息数

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取指定会话中的未读消息总数。获取成功后，会返回未读消息数（count）。不适用于聊天室、超级群。

```
[[RCCoreClient sharedCoreClient] getUnreadCount:ConversationType_PRIVATE
targetId:@"targetId"
completion:^(int count) {
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。不适用于聊天室、超级群。
targetId	NSString	会话 ID。
completion	Block	异步回调。

获取指定会话中指定消息类型的总消息未读数

> 提示
>
> - 从 5.1.5 版本开始提供该功能；仅在 `RCCoreClient` 中提供。

- 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取指定会话内指定的某一个或多个消息类型的未读数。

```
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:@"targetId"];
[[RCCoreClient sharedCoreClient] getUnreadCount:iden messageClassList:@[RCTextMessage.class, RCImageMessage.class]
completion:^(int count){
//异步回调，返回该会话内的未读消息数
}];
```

参数	类型	说明
conversationIdentifier	RCConversationIdentifier	会话标识，需要指定会话类型（RCConversationType）和 Target ID。
messageClassList	NSArray	消息类型数组，例 @[RCTextMessage.class, RCImageMessage.class]
completion	Block	异步回调。

按会话类型获取总未读消息数

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取多个指定会话类型的未读数。获取成功后，会返回未读消息数（count）。

```
[[RCCoreClient sharedCoreClient] getUnreadCount:@[@(ConversationType_PRIVATE),@(ConversationType_GROUP)]
containBlocked:NO
completion:^(int count) {
}];
```

参数	类型	说明
conversationTypes	NSArray	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array。不适用于聊天室、超级群。
isContain	BOOL	是否包含免打扰消息的未读数。
completion	Block	异步回调。

按会话免打扰级别获取总未读消息数

> 提示
>
> SDK 从 5.2.5 版本开始支持该接口。仅在 RCChannelClient 中提供。

获取已设置指定免打扰级别的会话的总未读消息数。SDK 将按照传入的免打扰级别配置查找会话，再返回这些所有会话的总未读消息数。

```
NSArray *conversationTypes = @[@(ConversationType_PRIVATE)];
NSArray * pushNotificationLevels = @[@(RCPushNotificationLevelMention)];
[[RCChannelClient sharedChannelManager] getUnreadCount:conversationTypes
levels: pushNotificationLevels
success:^(NSInteger unReadCount) {}
error:^(RCErrorCode status) {}
];
```

获取成功后，successBlock 中会返回未读消息数。

参数	类型	必填
conversationTypes	RCConversationType []	会话类型数组。不适用于聊天室。
levels	RCPushNotificationLevel[]	免打扰类型数组。详见免打扰功能概述。
successBlock	Block	成功回调
errorBlock	Block	失败回调

按会话免打扰级别获取总未读 @ 消息数

> 提示
>
> SDK 从 5.2.5 版本开始支持该接口。仅在 RCChannelClient 中提供。

获取已设置指定免打扰级别的会话的总未读 @ 消息数。SDK 将按照传入的免打扰级别配置查找会话，再返回这些所有会话的总未读 @ 消息数。

```
NSArray *conversationTypes = @[@(ConversationType_GROUP)];
NSArray * pushNotificationLevels = @[@(RCPushNotificationLevelMention)];
[[RCChannelClient sharedChannelManager] getUnreadMentionedCount:conversationTypes
levels: pushNotificationLevels
success:^(NSInteger unReadCount) {}
error:^(RCErrorCode status) {}
];
```

获取成功后，successBlock 中会返回未读 @ 消息数。

参数	类型	必填
conversationTypes	RCConversationType []	会话类型数组。不适用于聊天室。
levels	RCPushNotificationLevel[]	免打扰类型数组。详见免打扰功能概述。
successBlock	Block	成功回调
errorBlock	Block	失败回调

清除指定会话未读数

按时间戳清除指定会话的未读数。SDK 会将该时间戳之前的消息的未读状态全部清除。

```
BOOL success = [[RCIMClient sharedRCIMClient] clearMessagesUnreadStatus:ConversationType_PRIVATE targetId:@"targetId"
time:0];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。不适用于聊天室、超级群。
targetId	NSString	会话 ID
timestamp	long long	时间戳，此时间戳之前的未读状态都清除。

获取会话未读消息 获取会话未读消息

更新时间:2024-08-30

IMLib SDK 支持从指定会话中获取未读消息，可满足 App 跳转到第一条未读消息、展示全部未读 @ 消息的需求。

获取会话中第一条未读消息

获取会话中的最早一条未读消息。

```
RCMessage *theFirstUnreadMessage = [[RCIMClient sharedRCIMClient]
getFirstUnreadMessage:ConversationType_PRIVATE
targetId:@"targetId"];
```

获取成功后，返回消息对象（RCMessage）。

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 ID

获取会话中未读的 @ 消息

> 提示
>
> - 低于 5.2.5 版本仅提供不带 `count` 与 `desc` 参数的 `getUnreadMentionedMessages` 方法，每次最多返回 10 条数据。
> - 从 5.2.5 版本开始，`getUnreadMentionedMessages` 支持 `count` 与 `desc` 参数。该方法仅在 `RCCoreClient` 中提供。
> - 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取会话中最早或最新的未读 @ 消息，最多返回 100 条。

```
[[RCCoreClient sharedRCCoreClient]
getUnreadMentionedMessages:ConversationType_PRIVATE
targetId:@"targetId"
count:count
desc: NO
completion:completion];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
count	int	消息条数。最大 100 条。
desc	BOOL	`YES`：拉取最新的 `count` 条数据。`NO`：拉取最早的 `count` 条数据。
completion	Block	异步回调，返回消息对象（RCMessage）列表。

删除会话　　删除会话

更新时间:2024-08-30

App 用户可能需要从会话列表中删除一个会话或多个会话，可以通过 SDK 删除会话功能实现。客户端的会话列表是根据本地消息生成的，删除会话操作指的是删除本地会话。

删除指定会话

调用 removeConversation 可实现软删除的效果。该接口实际不会删除会话内的消息，仅将该会话项目从 SDK 的会话列表中移除。成功删除会话后，App 可以刷新 UI，不再向用户展示该会话项目。

```
BOOL success = [[RCIMClient sharedRCIMClient] removeConversation:ConversationType_PRIVATE
targetId:@"targetId"];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	会话 id

如果会话内再来一条消息，该会话会重新出现在会话列表中，App 用户可查看会话内的历史消息和最新消息。

SDK 未提供同时删除指定会话项目和会话历史消息的接口。如果需要同时删除会话内的消息，您可以在删除指定会话时同时调用删除消息的接口。详见删除消息。

按会话类型删除会话

App 用户可能需要清空某一类型的所有会话，例如清空所有群聊会话。SDK 支持按指定会话类型清空所有会话及会话信息，一次支持清空多个类型的会话。

```
BOOL success = [[RCIMClient sharedRCIMClient]
clearConversations:@[@(ConversationType_PRIVATE),@(ConversationType_SYSTEM),@(ConversationType_GROUP)]];
```

参数	类型	说明
conversationTypeList	NSArray	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

删除全部会话

SDK 内部没有清除全部会话的方法，如有需要可通过以下任一方式实现：

- 如果 App 不涉及超级群业务，您可使用 按会话类型删除会话，传入所有会话类型。这种方式会清除本地消息。
- 先获取会话列表，循环删除指定会话：

```
NSArray *array = [[RCIMClient sharedRCIMClient] getConversationList:@[@(ConversationType_GROUP),
@(ConversationType_PRIVATE)]];
for (RCConversation *con in array) {
[[RCIMClient sharedRCIMClient] removeConversation:con.conversationType targetId:con.targetId];
}
```

会话草稿　　会话草稿 保存草稿

更新时间:2024-08-30

使用 saveTextMessageDraft:targetId:content:completion: 保存一条草稿内容至指定会话。保存草稿会更新会话 sentTime，该会话会排在列表前部。

```
[[RCCoreClient sharedCoreClient] saveTextMessageDraft:ConversationType_PRIVATE
targetId:@"targetId"
content:@"这个是草稿内容"
completion:^(BOOL success) {

}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
content	NSString	草稿信息

获取草稿

使用 getTextMessageDraft:targetId:completion: 获取草稿内容。

```
[[RCCoreClient sharedCoreClient] getTextMessageDraft:ConversationType_PRIVATE
targetId:@"targetId"
completion:^(NSString *draft) {
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID

删除草稿

使用 clearTextMessageDraft:targetId:completion: 删除草稿。

```
[[RCCoreClient sharedCoreClient] clearTextMessageDraft:ConversationType_PRIVATE
targetId:@"targetId"
completion:^(BOOL success) {

}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID

输入状态 输入状态

更新时间:2024-08-30

应用程序可以在单聊会话中发送当前用户输入状态。对端收到通知后可以在 UI 展示 “xxx 正在输入”。

发送输入状态消息

在当前用户输入文本时调用 sendTypingStatus，发送当前用户输入状态。

```
[[RCIMClient sharedRCIMClient] sendTypingStatus:self.conversationType
targetId:self.targetId
contentType:[RCTextMessage getObjectName]];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。该接口仅支持单聊会话类型。
targetId	NSString	会话 Id
objectName	NSString	正在输入的消息的类型名。如文本消息，通过 [RCTextMessage getObjectName] 获取类型名。

监听输入状态

应用程序可以使用 setRCTypingStatusDelegate 设置 RCTypingStatusDelegate 代理，监听在单聊类型的会话收到的输入状态通知。在收到对端发来的输入状态通知时，SDK 会通过 onTypingStatusChanged 回调方法返回当前正在输入的用户列表和消息类型。

设置代理委托

```
[[RCIMClient sharedRCIMClient] setRCTypingStatusDelegate:self];
```

代理方法

```
@protocol RCTypingStatusDelegate <NSObject>

/*!
用户输入状态变化的回调

@param conversationType 会话类型
@param targetId 会话目标ID
@param userTypingStatusList 正在输入的RCUserTypingStatus列表（nil 表示当前没有用户正在输入）

@discussion
当客户端收到用户输入状态的变化时，会回调此接口，通知发生变化的会话以及当前正在输入的RCUserTypingStatus列表。

@warning 目前仅支持单聊。
*/
- (void)onTypingStatusChanged:(RCConversationType)conversationType
targetId:(NSString *)targetId
status:(NSArray *)userTypingStatusList;

@end
```

管理标签信息数据　　管理标签信息数据

更新时间:2024-08-30

SDK 从 5.1.1 版本开始支持创建标签。

本文描述如何 App 如何使用 RCCoreClient 下的接口创建和管理标签信息数据。客户端 SDK 支持用户创建标签信息（RCTagInfo ），用于对会话进行标记分组。每个用户最多可以创建 20 个标签。App 用户创建的标签信息数据会同步融云服务端。

标签信息（RCTagInfo ）的定义如下：

参数	类型	说明
tagId	NSString	标签唯一标识，字符型，长度不超过 10 个字。
tagName	NSString	长度不超过 15 个字，标签名称可以重复。
count	NSInteger	标签下会话个数
timestamp	long long	时间戳由 SDK 内部协议栈提供。

提示

本文仅描述如何管理标签信息数据。关于如何为会话设置标签、以及如何按标签获取会话数据，请参见设置与使用会话标签。

创建标签信息

创建标签，每个用户最多可以创建 20 个标签。

```
RCTagInfo *tag = [[RCTagInfo alloc] initWithTagInfo:@"tagId" tagName:@"tagName"];

[[RCCoreClient sharedCoreClient] addTag:tag success:^{

} error:^(RCErrorCode errorCode) {

}];
```

移除标签信息

移除标签。移除标签信息时只需要传入 RCTagInfo 中的 tagId。

```
[[RCCoreClient sharedCoreClient] removeTag:@"tagId" success:^{

} error:^(RCErrorCode errorCode) {

}];
```

编辑标签信息

更新标签信息。仅支持修改 RCTagInfo 中的标签名称（tagName）字段。长度不超过 15 个字，标签名称可以重复。

```
RCTagInfo *tag = [[RCTagInfo alloc] initWithTagInfo:@"tagId" tagName:@"tagName"];

[[RCCoreClient sharedCoreClient] updateTag:tag success:^{

} error:^(RCErrorCode errorCode) {

}];
```

获取标签信息列表

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取当前用户已创建的标签信息。成功回调中会返回 RCTagInfo 的列表。

```
[[RCCoreClient sharedCoreClient] getTags:^(NSArray<RCTagInfo *> * _Nonnull tags) {
NSArray *array = tags;
}];
```

参数	类型	说明
completion	Block	异步回调，返回标签信息（RCTagInfo）的列表。

多端同步标签信息修改

即时通讯支持同一用户账号在多端登录。如果您的 App 用户在当前设备上修改了标签信息，SDK 会负责通知该用户的其他设备。其他设备收到通知后，需要调用 getTags 从融云服务端获取最新标签信息。

设置代理委托后可在当前设备上接收到来自其他设备的标签信息修改通知。

> 提示
>
> - 请在初始化之后，连接之前调用该方法。
> - 在当前设备上修改标签信息不会触发该回调方法。服务端仅会通知 SDK 在同一用户账号登录的其他设备上触发回调。

设置代理委托

```
[RCCoreClient sharedCoreClient].tagDelegate = self;
```

代理方法

```
@protocol RCTagDelegate <NSObject>

/*!
标签变化

@discussion 本端添加删除更新标签，不会触发不会触发此回调方法，在相关调用方法的 block 块直接回调
*/
- (void)onTagChanged;

@end
```

当用户在其它端添加、移除、编辑标签时，SDK 会触发 onTagChanged 回调。请在收到通知后调用 getTags 从融云服务端获取最新标签信息。

设置与使用会话标签 设置与使用会话标签

更新时间:2024-08-30

- SDK 从 5.1.1 版本开始支持会话标签功能，相关接口仅在 `RCCoreClient` 中提供。
- 在为会话设置标签前，请确保已创建标签信息。详见管理标签信息数据。
- 本功能不适用于聊天室、超级群。

每个用户最多可以创建 20 个标签，每个标签下最多可以添加 1000 个会话。如果标签下已添加 1000 个会话，继续在该标签下添加会话仍会成功，但会导致最早添加标签的会话被移除标签。

场景描述

会话标签常实现 App 用户对会话进行分组的需求。创建标签信息（RCTagInfo）后，App 用户可以为会话设置一个或多个标签。

设置标签后，可以利用会话的标签数据实现会话的分组获取、展示、删除等特性。还可以获取指定标签下所有会话的消息未读数，或在特定标签下设置某个会话置顶。

- **场景 1**：对会话列表中的每个会话打 tag，类似企业微信会话列表中的外部群，部门群，个人群等 tag。
- **场景 2**：通讯录根据 tag 来分组，类似 QQ 好友列表中的家人，朋友，同事分组等。
- **场景 3**：前两个场景的结合，按照 tag 来进行会话列表分组，类似 Telegram 的会话列表分组。

使用标签标记会话

在创建标签信息（RCTagInfo）后，App 用户可以使用标签标记会话。SDK 将用标签标记会话的操作视为将会话添加到标签中。

支持以下操作：

- 标记会话，即将一个或多个会话添加到指定标签
- 从标签中移除一个或多个会话
- 为指定会话移除一个或多个标签

将一个或多个会话添加到指定标签

SDK 将用标签标记会话的操作视为将会话添加到标签中。您可以将多个会话添加到一个标签。指定标签时只需要传入 RCTagInfo 中的 `tagId`。

```
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:@"targetId"];

[[RCCoreClient sharedCoreClient] addConversationsToTag:@"tagId" conversationIdentifiers:@[iden] success:^{

} error:^(RCErrorCode errorCode) {

}];
```

参数	类型	说明
tagId	NSString	标签 ID。
conversationIdentifiers	NSArray	会话列表，数组元素是 RCConversationIdentifier。
successBlock	Block	成功回调。

参数	类型	说明
errorBlock	Block	失败回调。`errorCode` 参数返回 RCErrorCode 。

从指定标签下移除会话

App 用户可能需要携带指定标签的会话中移除一个或多个会话。例如，在所有添加了「培训班」标签的会话中移除与「Tom」的私聊会话。SDK 将该操作视为从指定标签中移除会话。移除成功后，会话仍然存在，但不再携带该标签。

```
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:@"targetId"];

[[RCCoreClient sharedCoreClient] removeConversationsFromTag:@"tagId" conversationIdentifiers:@[iden] success:^{

} error:^(RCErrorCode errorCode) {

}];
```

参数	类型	说明
tagId	NSString	标签 id
conversationIdentifiers	NSArray	会话列表，数组元素是 RCConversationIdentifier 。
successBlock	Block	成功回调。
errorBlock	Block	失败回调。`errorCode` 参数返回 RCErrorCode 。

为指定会话中移除标签

App 用户可能为指定会话中添加了多个标签。SDK 支持一次移除单个或多个标签。移除时需要传入所有待移除 RCTagInfo 的 `tagId` 列表。

```
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:@"targetId"];

[[RCCoreClient sharedCoreClient] removeTagsFromConversation:iden tagIds:@[@"tagId1",@"tagId2"] success:^{

} error:^(RCErrorCode errorCode) {

}];
```

参数	类型	说明
tagIds	NSArray	标签 ID 列表。
conversationIdentifiers	NSArray	会话列表，数组元素是 RCConversationIdentifier 。
successBlock	Block	成功回调。
errorBlock	Block	失败回调。`errorCode` 参数返回 RCErrorCode 。

获取指定会话的所有标签

> 提示
>
> 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取指定会话携带的所有标签。获取成功后，回调中返回 RCConversationTagInfo 的列表。每个 RCConversationTagInfo 中包含对应的标签信息 RCTagInfo 和置顶状态信息（会话是否在携带该标签信息的所有会话中置顶）。

```
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:@"targetId"];

[[RCCoreClient sharedCoreClient] getTagsFromConversation:iden completion:^(NSArray<RCConversationTagInfo *> *){
//异步回调，返回标签列表
}];
```

参数	类型	说明
conversationIdentifier	RCConversationIdentifier	会话标识，需要指定会话类型（RCConversationType）和 Target ID。
completion	Block	异步回调，返回标签信息（RCConversationTagInfo）列表。

RCConversationTagInfo 说明：

参数	类型	说明
tagInfo	RCTagInfo	标签信息。
isTop	BOOL	会话是否置顶。

多端同步会话标签修改

即时通讯支持同一用户账号在多端登录。如果您的 App 用户在当前设备上修改了会话标签，SDK 会负责通知该用户的其他设备。其他设备收到通知后，可以调用 getTagsFromConversation 从融云服务端获取指定会话的最新标签数据。

设置代理委托后后可在当前设备上接收到来自其他设备的会话标签修改通知。

> 提示
>
> - 请在初始化之后，连接之前调用该方法。
> - 在当前设备上修改标签信息不会触发该回调方法。服务端仅会通知 SDK 在同一用户账号登录的其他设备上触发回调。

设置代理委托

```
//RCCoreClient.h
@property (nonatomic, weak) id<RCConversationTagDelegate> conversationTagDelegate;
```

代理方法

```
@protocol RCConversationTagDelegate <NSObject>
/*!
会话标签变化

@discussion 本端添加删除更新会话标签，不会触发此回调方法，在相关调用方法的 block 块直接回调
*/
- (void)onConversationTagChanged;
@end
```

当 App 用户在其它端添加、移除、编辑会话上的标签时，SDK 会触发 onConversationTagChanged 回调。请在收到通知后调用 getTagsFromConversation 从融云服务端获取指定会话的最新标签数据。

按标签操作会话数据

SDK 支持对携带指定标签的会话进行操作。App 用户为会话添加标签后，可以实现以下操作：

- 配合使用会话置顶功能，可以实现在携带指定标签的会话中置顶会话
- 按标签获取会话列表，即获取携带指定标签的所有会话

- 按标签获取未读消息数
- 清除标签对应会话的未读消息数
- 删除标签对应的会话

在携带指定标签的会话中置顶

App 可以使用会话标签按照业务需求对会话进行分类和展示。如果需要在同一类会话（携带同一标签的所有会话）中置顶显示会话，可以使用会话置顶功能。

详细实现方式请参见会话置顶的**在标签下置顶会话。

分页获取本地指定标签下会话列表

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

以会话中最后一条消息时间戳为界，分页获取本地指定标签下会话列表。该方法仅从本地数据库中获取数据。从 5.6.4 版本开始，该接口返回的 RCConversation 对象新增 isTopForTag 属性，如果为 YES，表示该会话在当前 tagId 下为置顶会话。

```
[[RCCoreClient sharedCoreClient] getConversationsFromTagByPage:texts[0] timestamp:0 count:20
completion:^(NSArray<RCConversation *> *){
//异步回调，返回会话列表
}];
```

参数	类型	说明
tagId	NSString	标签 ID。
timestamp	long long	会话的时间戳。获取这个时间戳之前的会话列表。首次可传 0，后续可以使用返回的 RCConversation 对象的 sentTime 或 operationTime 属性值，作为下一次查询的 startTime。推荐使用 operationTime（该属性仅在 5.6.8 及之后版本提供）。
count	int	获取的数量。当实际取回的会话数量小于 count 值时，表明已取完数据。
completion	Block	异步回调，返回会话（RCConversation）列表。

按标签获取未读消息数

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

获取携带指定标签的所有会话的未读消息数。

```
[[RCCoreClient sharedCoreClient] getUnreadCountByTag:@"tagId" containBlocked:isBlocked completion:^(int count){
//异步回调
}];
```

参数	类型	说明
tagId	NSString	标签 ID。
isContain	BOOL	是否包含免打扰会话。
completion	Block	异步回调，返回会话消息未读数。

清除标签对应会话的未读消息数

提示

- 从 5.1.5 版本开始支持该功能。
- 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

清除携带指定标签的所有会话的未读消息数。

```
[[RCCoreClient sharedCoreClient] clearMessagesUnreadStatusByTag:@"tagId" completion:^(BOOL){
//异步回调
}];
```

参数	类型	说明
tagId	NSString	标签 ID。
completion	Block	异步回调，返回是否清除成功。

删除标签对应的会话

提示

- 从 5.1.5 版本开始支持该功能。
- 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

删除指定标签下的全部会话，同时解除这些会话和标签的绑定关系。删除成功后，会话不再携带指定的标签。这些会话收到新消息时，会产生新的会话。

```
RCClearConversationOption *option = [[RCClearConversationOption alloc] init];
option.isDeleteMessage = YES;
[[RCCoreClient sharedCoreClient] clearConversationsByTag:@"tagId" option:option success:^{

} error:^(RCErrorCode errorCode) {

}];
```

参数	类型	说明
tagId	NSString	标签 ID。
option	RCClearConversationOption	清除会话的相关配置。
successBlock	Block	成功回调。
errorBlock	Block	失败回调。`errorCode` 参数返回 RCErrorCode 。

您可以通过 `option` 参数的 `isDeleteMessage` 属性配置是否同时清除这些会话对应的本地消息。

- 如果不删除会话对应的本地消息，再接收到新消息时，可以看到历史聊天记录。
- 如果删除会话对应的本地消息，再接收到新消息时，无法看到历史聊天记录。如果开通了单群聊消息云存储服务，服务端仍保存有消息历史。如需删除，请使用删除服务端历史消息接口。

会话置顶 会话置顶

更新时间:2024-08-30

会话置顶功能提供以下能力：

- 在会话列表中置顶会话：通过会话（`RCConversation`）的置顶（`isTop`）属性控制。
- 在携带同一标签的会话中置顶（需配合使用会话标签功能）：通过 RCConversationTagInfo 类的 `isTop` 属性控制。

在会话列表中置顶会话

设置指定会话在会话列表中置顶后，SDK 将修改 RCConversation 的 isTop 字段，该状态将会被同步到服务端。融云会在为用户自动同步会话置顶的状态数据。客户端可以主动获取或通过监听器获取到最新数据。

设置会话置顶

使用 setConversationToTop:targetId:isTop:completion: 设置会话置顶。

```
// Set the conversation type and target ID
RCConversationType conversationType = ConversationType_PRIVATE;
NSString *targetId = @"your_target_id";

// Set the conversation to top or not
BOOL isTop = YES;

[[RCCoreClient sharedCoreClient] setConversationToTop:conversationType
targetId:targetId
isTop:isTop
completion:^(BOOL success) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，支持单聊、群聊、系统会话。
targetId	NSString	会话 id
isTop	BOOL	是否置顶
completion	Block	设置置顶结果回调

客户端通过本地消息数据自动生成会话与会话列表，并会在用户登录的多个设备之间同步置顶状态。如果在调用该 API 时，要置顶的会话在本地或该用户登录的其他设备上不存在（会话尚未生成，或者已被移除），SDK SDK 将直接创建会话并置顶。

设置会话置顶可选择是否更新会话时间

使用 [setConversationToTop:targetId:isTop:needUpdateTime:completion:] 设置会话置顶。

```
// Set the conversation type and target ID
RCConversationType conversationType = ConversationType_PRIVATE;
NSString *targetId = @"your_target_id";

// Set the conversation to top or not
BOOL isTop = YES;

[[RCCoreClient sharedCoreClient] setConversationToTop:conversationType
targetId:targetId
isTop:isTop
needUpdateTime:YES
completion:^(BOOL success) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型，支持单聊、群聊、系统会话。
targetId	NSString	会话 id
isTop	BOOL	是否置顶
needUpdateTime	BOOL	是否更新时间
completion	Block	设置置顶结果回调

客户端通过本地消息数据自动生成会话与会话列表，并会在用户登录的多个设备之间同步置顶状态。如果在调用该 API 时，要置顶的会话在本地或该用户登录的其他设备上不存在（会话尚未生成，或者已被移除），SDK 将直接创建会话并置顶。

获取是否置顶

> 提示
>
> - 从 5.1.5 版本开始支持该功能。
> - 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

主动获取指定会话的置顶状态。

```
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:@"targetId"];
[[RCCoreClient sharedCoreClient] getConversationTopStatus:iden completion:^(BOOL){
//异步回调，是否获取成功
}];
```

参数	类型	说明
conversationIdentifier	RCConversationIdentifier	会话标识，需要指定会话类型（RCConversationType）和 Target ID。
completion	Block	返回是否设置成功。

获取置顶会话列表

主动获取指定会话类型的所有置顶会话。

```
NSArray *conversations = [[RCIMClient sharedRCIMClient]
getTopConversationList:@[@(ConversationType_PRIVATE),@(ConversationType_GROUP)]];
```

参数	类型	说明
conversationTypes	NSArray	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array

获取成功后，返回 会话 `RCConversation` 的列表。

在携带标签的所有会话中置顶会话

> 提示
>
> 该功能相关接口仅在 `RCCoreClient` 中提供。在标签标记的所有会话中置顶是通过修改 `RCConversationTagInfo.isTop` 字段实现的，不影响 RCConversation 的 `isTop` 字段。

如果 App 实现了会话标签功能，App 用户可能会使用同一个标签标记多个会话，并且需要将其中一个会话置顶。SDK 在 RCConversationTagInfo 中提供 `isTop` 字段，用于控制会话在携带同样标签的会话中的置顶状态。

在标签下置顶会话

在携带指定标签的所有会话中设置指定会话置顶。例如，在所有添加了「培训班」标签的会话中将与「Tom」的私聊会话置顶。

```
NSString targetId = @"useridoftom";
NSString *tagId = @"peixunban";
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:targetId];

[[RCCoreClient sharedCoreClient] setConversationToTopInTag:tagId conversationIdentifier:iden isTop:isTop success:^{

} error:^(RCErrorCode errorCode) {

}];
```

参数	类型	说明
tagId	String	标签 ID
conversationIdentifier	RCConversationIdentifier	会话标识，需要指定会话类型（RCConversationType）和 Target ID。
isTop	BOOL	是否置顶
successBlock	Block	成功回调
errorBlock	Block	失败回调。`errorCode` 参数返回 RCErrorCode 。

获取会话在标签下的置顶状态

> 提示
>
> 从 5.3.0 版本 `RCCoreClient` 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

查询指定会话是否在携带同一标签的所有会话中置顶。获取成功后会返回是否已置顶。

```
RCConversationIdentifier *iden = [[RCConversationIdentifier alloc] initWithConversationIdentifier:ConversationType_PRIVATE
targetId:@"targetId"];

[[RCCoreClient sharedCoreClient] getConversationTopStatusInTag:iden tagId:@"tagId" completion:^(BOOL){
// 异步回调，返回置顶状态
}];
```

参数	类型	说明
tagId	NSString	标签 ID
conversationIdentifier	RCConversationIdentifier	会话标识，需要指定会话类型（RCConversationType）和 Target ID。
completion	Block	异步回调，返回是否置顶

是否开启同步空置顶会话（SDK 版本 ≧ 5.10.0）

卸载重装或者换设备登录后，置顶状态的会话会在会话列表中显示，包括已置顶的空会话。

从 5.10.0 版本开始，支持是否开启同步空置顶会话功能。

可在 SDK 初始化时通过 `RCInitOption` 的 `enableSyncEmptyTopConversation` 属性设置是否开启。如不设置，SDK 默认不自动同步空置顶会话。

> 提示
>
> 如果开启同步空置顶会话功能，那么分页获取会话数据时，第一页 startTime 传 0，后续传会话对象 `operationTime` 的值。

```
RCInitOption *initOption = [[RCInitOption alloc] init];
// 打开同步空置顶会话
initOption.enableSyncEmptyTopConversation = YES;
// 初始化 SDK
[[RCCoreClient sharedCoreClient] initWithAppKey:@"your_appkey" option:initOption];
```

免打扰功能概述 免打扰功能概述

更新时间:2024-08-30

「免打扰功能」用于控制用户在客户端设备离线时，是否可针对离线消息接收推送通知。

- **客户端为离线状态**：会话中有新离线消息时，用户默认通过推送通道收到消息且默认弹出通知。设置免打扰后，融云服务端不会为相关消息触发推送。
- **客户端在后台运行**：会话中有新消息时，用户直接收到消息。如果使用 IMLib，您需要自行判断 App 是否在后台运行，并根据业务需求自行实现本地通知弹窗。

前提条件

请在使用「免打扰功能」前检查是否已集成 APNs 推送。

免打扰设置维度

客户端 SDK 支持对单聊、群聊、系统会话业务进行以下多个维度的免打扰设置：

- App 的免打扰设置
- 按指定会话类型设置免打扰级别
- 按会话设置免打扰级别
- 全局免打扰

App 的免打扰设置

以 App Key 为单位，设置整个应用所有用户的默认免打扰级别。默认未设置，等同于全部消息都接收通知。该级别的配置暂未在控制台开放，如有需要，请提交工单。

- **全部消息均通知**：当前 App 下的用户可针对任何消息接收推送通知。
- **未设置**：默认全部消息都通知。
- **仅 @ 消息通知**：当前 App 下，仅针对提及（@）指定用户和群组全体成员的消息向离线用户发送推送通知。
- **仅 @ 指定用户通知**：当前 App 下，用户仅针对提及（@）当前用户的消息接收推送通知。例如：仅张三会接收且仅接收 “@张三 Hello” 的消息的通知。
- **仅 @群全员通知**：当前 App 下，用户仅针对提及（@）群组全体成员的消息接收推送通知。
- **都不接收通知**：当前 App 下，用户不针对任何消息接收推送通知，即任何离线消息都不会触发推送通知。
- **除 @ 消息外群聊消息不发推送**：当前 App 下，用户针对单聊消息、提及（@）指定用户的消息、和提及（@）群组全体成员的消息接收推送通知。

融云服务端判断是否需要推送时，App 级别的免打扰配置的优先级最低。如果存在以下任何一种用户级别的免打扰配置，以用户级别配置为准：

- 按指定会话类型设置免打扰级别
- 按会话设置免打扰级别
- 全局免打扰

按会话类型设置免打扰级别

> 提示
>
> 客户端 SDK 从 5.2.2.1 开始支持该功能。该功能属于用户级别设置。

客户端 SDK 提供 RCPushNotificationLevel，允许用户为会话类型（单聊、群聊、超级群、系统会话）配置触发推送通知的消息类别，或完全关闭通知。提供以下六个级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	与融云服务端断开连接后，当前用户可针对指定类型会话中的所有消息接收通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。
RCPushNotificationLevelMention	1	与融云服务端断开连接后，当前用户仅针对指定类型的会话中提及（@）当前用户和全体群成员的消息接收通知。
RCPushNotificationLevelMentionUsers	2	与融云服务端断开连接后，当前用户仅针对指定类型的会话中提及（@）当前用户的消息接收通知。例如：张三只会接收 “@张三 Hello” 的消息的通知。
RCPushNotificationLevelMentionAll	4	与融云服务端断开连接后，当前用户仅针对指定类型的会话中提及（@）全部群成员的消息接收通知。
RCPushNotificationLevelBlocked	5	当前用户针对指定类型的会话中的任何消息都不接收推送通知。

具体设置方法详见按会话类型设置免打扰。

融云服务端判断是否需要为用户发送推送通知时，如果同时存在以下任何一种用户级别的免打扰配置，以下列配置为准：

- 按会话设置免打扰级别
- 全局免打扰

按会话设置免打扰级别

> 提示
>
> 客户端 SDK 从 5.2.2 开始支持该功能。该功能属于用户级别设置。

客户端 SDK 提供 RCPushNotificationLevel，允许用户为会话配置触发通知的消息类别，或完全关闭通知。提供以下六个级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	与融云服务端断开连接后，当前用户可针对指定类型会话中的所有消息接收通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。
RCPushNotificationLevelMention	1	与融云服务端断开连接后，当前用户仅针对指定会话中提及（@）当前用户和全体群成员的消息接收通知。
RCPushNotificationLevelMentionUsers	2	与融云服务端断开连接后，当前用户仅针对接收指定会话中提及（@）当前用户的消息接收通知。例如：张三只会接收 “@张三 Hello” 的消息的通知。
RCPushNotificationLevelMentionAll	4	与融云服务端断开连接后，当前用户仅针对指定会话中提及（@）全部群成员的消息接收通知。
RCPushNotificationLevelBlocked	5	当前用户针对指定会话中中的任何消息都不接收推送通知。

具体设置方法详见按会话设置免打扰。

融云服务端判断是否需要为用户发送推送通知时，如果该用户已配置全局免打扰，则已全局免打扰的配置细节为准。

全局免打扰

客户端 SDK 从 5.2.2 开始提供 RCPushNotificationQuietHoursLevel，允许用户配置何时接收通知以及触发通知的消息类别。提供了以下三个级别：

枚举值	数值	说明
RCPushNotificationQuietHoursLevelDefault	0	未设置。如未设置，SDK 会依次查询消息所属群的用户级别免打扰设置及其他非用户级别设置，再判断是否需要推送通知。
RCPushNotificationQuietHoursLevelMention	1	与融云服务端断开连接后，当前用户仅在指定时段内针对指定会话中提及（@）当前用户和全体群成员的消息接收通知。

枚举值	数值	说明
RCPushNotificationQuietHoursLevelBlocked	5	当前用户在指定时段内针对任何消息都不接收推送通知。

具体设置方法详见全局免打扰。

早于 5.2.2 的 SDK 版本不支持设置触发通知的消息类别，仅支持设置为接收或不接收推送通知。

免打扰设置的优先级

针对单聊、群聊、系统会话、融云服务端会遵照以下顺序搜索免打扰配置。优先级从左至右依次降低，以优先级最高的配置为准判断是否需要触发推送：

全局免打扰设置（用户级）> 指定会话类型的免打扰设置（用户级）> 指定会话的免打扰设置（用户级）> App 级的免打扰设置

API 接口列表

下表描述了适用于单聊、群聊、系统会话的免打扰配置 API 接口。

免打扰配置维度	客户端 API	服务端 API
设置指定时段内，应用全局的免打扰级别。	详见全局免打扰。	详见 设置用户免打扰时段。
设置指定类型会话的免打扰级别	详见按会话类型设置免打扰。	详见设置会话类型免打扰。
设置指定会话的免打扰级别	详见按会话设置免打扰。	详见设置会话免打扰。
设置 App 级免打扰级别	客户端 SDK 不提供 API。	服务端不提供该 API。

按会话设置免打扰 按会话设置免打扰

更新时间:2024-08-30

本文描述如何为指定会话（targetId）设置免打扰级别。

> 提示
>
> 即时通讯客户端 SDK 支持多维度、多级别的免打扰设置。
>
> - App 开发者可实现从 App Key、指定细分业务（仅超级群）、用户级别多个维度的免打扰功能配置。在融云服务端决定是否触发推送通知时，不同维度的优先级如下：用户级别设置 > 指定超级群频道的默认配置（仅超级群支持） > 指定超级群会话的默认配置（仅超级群支持） > App Key 级设置。
> - 用户级别设置下包含多个细分维度。在融云服务端决定是否触发推送通知时，如存在用户级别配置，不同细分维度的优先级如下：全局免打扰 > 按频道设置的免打扰 > 按会话设置的免打扰 > 按会话类型设置的免打扰。详见免打扰功能概述。

在免打扰设置生效时，客户端收到新消息时行为如下：

- 客户端在后台运行：会话中有新消息时，可以收到消息内容。您需要自行实现本地通知弹窗。
- 客户端为离线状态：会话中有新消息时，不会收到远程通知提醒，再次上线时可收取消息内容

支持的免打扰级别

免打扰级别提供了针对不同 @ 消息的免打扰控制。从 SDK 5.2.2 开始，指定会话的免打扰配置支持以下级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	所有消息均可进行通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。 注意：在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人
RCPushNotificationLevelMentionUsers	2	仅针对 @ 指定用户消息进行通知，且仅针对被 @ 的指定的用户进行通知。 如：@张三，则张三可以收到推送； @所有人不会触发推送通知。
RCPushNotificationLevelMentionAll	4	仅针对 @群全员进行通知，即只接收 @所有人的推送信息。
RCPushNotificationLevelBlocked	5	不接收通知，即使为 @ 消息也不推送通知。

早于 5.2.2 的 SDK 版本仅支持设置为免打扰状态（不接收推送通知）或提醒状态（接收推送通知）。

管理会话的免打扰设置

即时通讯业务用户（userId）可为指定会话（targetId）设置免打扰级别，支持单聊、群聊、超级群会话。

设置指定会话免打扰级别 （SDK >= 5.2.2）

> 提示
>
> 该接口在 RCChannelClient 中，从 5.2.2 版本开始支持。

为当前用户设置指定会话的（targetId）免打扰级别。

```
[[RCChannelClient sharedChannelManager] setConversationNotificationLevel:" 会话类型 "
targetId:" 会话 Id "
level:RCPushNotificationLevelDefault
success:^() {}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。请注意以下限制： • 超级群会话类型：如在 2022.09.01 之前开通超级群业务，默认不支持为单个超级群会话所有消息设置免打扰级别（“所有消息”指所有频道中的消息和不属于任何频道的消息）。该接口仅设置指定超级群会话（`targetId`）中不属于任何频道的消息的免打扰状态级别。如需修改请提交工单。 • 聊天室会话类型：不支持，因为聊天室消息默认不支持消息推送提醒。
targetId	NSString	会话 ID
level	RCPushNotificationLevel	• `-1`：全部消息通知 • `0`：未设置（用户未设置情况下，默认以群 或者 APP级别的默认设置为准，如未设置则全部消息都通知） • `1`：仅针对 @ 消息进行通知 • `2`：仅针对 @ 指定用户进行通知 如：@张三 则张三可以收到推送，@所有人 时不会收到推送。 • `4`：仅针对 @ 群全员进行通知，只接收 @所有人 的推送信息。 • `5`：不接收通知
successBlock	block	回调接口
errorBlock	block	回调接口

移除指定会话免打扰级别 （SDK >= 5.2.2）

如需移除指定会话类型的免打扰设置，请调用设置接口，并将 `level` 参数传入 RCPushNotificationLevelDefault。

查询指定会话免打扰级别 （SDK >= 5.2.2）

> 提示
>
> 该接口在 RCChannelClient 中，从 5.2.2 版本开始支持。

查询当前用户为指定会话（targetId）设置的免打扰级别。

```
[[RCChannelClient sharedChannelManager] getConversationNotificationLevel: " 会话类型 "
targetId: " 会话 Id "
success:^(RCPushNotificationLevel level) {}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	String	会话 Id
successBlock	block	回调接口

参数	类型	说明
errorBlock	block	回调接口

获取免打扰状态列表

> 提示
>
> 该接口在 RCIMClient 中。

获取所有设置了免打扰的会话。返回 RCConversation 列表不包含具体免打扰级别信息，不包含频道信息。

```
NSArray *array = [[RCIMClient sharedRCIMClient] getBlockedConversationList:@[@(ConversationType_PRIVATE)]];
```

参数	类型	说明
conversationTypeList	NSArray	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

按频道设置免打扰 按频道设置免打扰

更新时间:2024-08-30

本文描述如何为超级群业务的指定频道（channelId）设置免打扰级别。

> 提示
>
> 即时通讯客户端 SDK 支持多维度、多级别的免打扰设置。
>
> - App 开发者可实现从 App Key、指定细分业务（仅超级群）、用户级别多个维度的免打扰功能配置。在融云服务端决定是否触发推送通知时，不同维度的优先级如下：用户级别设置 > 指定超级群频道的默认配置（仅超级群支持） > 指定超级群会话的默认配置（仅超级群支持） > **App Key** 级设置。
> - 用户级别设置下包含多个细分维度。在融云服务端决定是否触发推送通知时，如存在用户级别配置，不同细分维度的优先级如下：全局免打扰 > 按频道设置的免打扰 > 按会话设置的免打扰 > 按会话类型设置的免打扰。详见免打扰功能概述。

在免打扰设置生效时，客户端收到新消息时行为如下：

- 客户端在后台运行：会话中有新消息时，可以收到消息内容。您需要自行实现本地通知弹窗。
- 客户端为离线状态：会话中有新消息时，不会收到远程通知提醒，再次上线时可收取消息内容。

支持的免打扰级别

免打扰级别提供了针对不同 @ 消息的免打扰控制。从 SDK 5.2.2 开始，指定频道的免打扰配置支持以下级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	所有消息均可进行通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。 注意：在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人
RCPushNotificationLevelMentionUsers	2	仅针对 @ 指定用户消息进行通知，且仅针对被 @ 的指定的用户进行通知。 如：@张三，则张三可以收到推送； @所有人不会触发推送通知。
RCPushNotificationLevelMentionAll	4	仅针对 @群全员进行通知，即只接收 @所有人的推送信息。
RCPushNotificationLevelBlocked	5	不接收通知，即使为 @ 消息也不推送通知。

早于 5.2.2 的 SDK 版本仅支持设置为免打扰状态（不接收推送通知）或提醒状态（接收推送通知）。

管理频道的免打扰设置

超级群（UltraGroup）支持在超级群的会话下创建独立的频道（channel），对消息数据（会话、消息、未读数）和群组成员分频道进行聚合。SDK 支持在超级群业务中的用户（userId）设置指定群频道（channelId）的免打扰级别。

设置指定频道免打扰级别 （SDK >= 5.2.2）

> 提示
>
> 该接口在 RCChannelClient 中，从 5.2.2 版本开始支持。

为当前用户设置超级群频道中的消息的免打扰级别。

```
[[RCChannelClient sharedChannelManager] setConversationChannelNotificationLevel:"会话类型"
targetId:" 会话 Id "
channelId:" 频道 Id "
level:RCPushNotificationLevelDefault
success:^() {}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。
targetId	NSString	会话 ID
channelId	NSString	超级群的会话频道 ID。 • 如果传入频道 ID，则针对该指定频道设置消息免打扰级别。 • 注意：2022.09.01 之前开通超级群业务的客户，如果不指定频道 ID，则默认传 "" 空字符串，即仅针对指定超级群会话（`targetId`）中不属于任何频道的消息设置免打扰状态级别。如需修改请提交工单。
level	RCPushNotificationLevel	• `-1`：全部消息通知 • `0`：未设置（用户未设置情况下，默认以群 或者 APP级别的默认设置为准，如未设置则全部消息都通知） • `1`：仅针对 @ 消息进行通知 • `2`：仅针对 @ 指定用户进行通知 如：@张三 则张三可以收到推送，@所有人 时不会收到推送。 • `4`：仅针对 @ 群全员进行通知，只接收 @所有人 的推送信息。 • `5`：不接收通知
successBlock	block	回调接口
errorBlock	block	回调接口

获取指定频道免打扰级别 （SDK >= 5.2.2）

获取为当前用户设置的超级群频道免打扰级别。

```
[[RCChannelClient sharedChannelManager] getConversationChannelNotificationLevel::"会话类型"
targetId:" 会话 Id "
channelId:" 频道 Id "
success:^(RCPushNotificationLevel level) {}
error:^(RCErrorCode status) { }];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 Id
channelId	NSString	超级群的会话频道 ID，获取会话指定频道的设置。
successBlock	block	回调接口
errorBlock	block	回调接口

按会话类型设置免打扰 按会话类型设置免打扰

更新时间:2024-08-30

本文描述如何为指定类型（conversationType）的会话设置免打扰级别。

> 提示
>
> 即时通讯客户端 SDK 支持多维度、多级别的免打扰设置。
>
> - App 开发者可实现从 App Key、指定细分业务（仅超级群）、用户级别多个维度的免打扰功能配置。在融云服务端决定是否触发推送通知时，不同维度的优先级如下：用户级别设置 > 指定超级群频道的默认配置（仅超级群支持） > 指定超级群会话的默认配置（仅超级群支持） > **App Key** 级设置。
> - 用户级别设置下包含多个细分维度。在融云服务端决定是否触发推送通知时，如存在用户级别配置，不同细分维度的优先级如下：全局免打扰 > 按频道设置的免打扰 > 按会话设置的免打扰 > 按会话类型设置的免打扰。详见免打扰功能概述。

在免打扰设置生效时，客户端收到新消息时行为如下：

- 客户端在后台运行：会话中有新消息时，可以收到消息内容。您需要自行实现本地通知弹窗。
- 客户端为离线状态：会话中有新消息时，不会收到远程通知提醒，再次上线时可收取消息内容。

支持的免打扰级别

免打扰级别提供了针对不同 @ 消息的免打扰控制。从 SDK 5.2.2.1 开始，指定会话类型的免打扰配置支持以下级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	所有消息均可进行通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。 注意：在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人
RCPushNotificationLevelMentionUsers	2	仅针对 @ 指定用户消息进行通知，且仅针对被 @ 的指定的用户进行通知。 如：@张三，则张三可以收到推送； @所有人不会触发推送通知。
RCPushNotificationLevelMentionAll	4	仅针对 @群全员进行通知，即只接收 @所有人的推送信息。
RCPushNotificationLevelBlocked	5	不接收通知，即使为 @ 消息也不推送通知。

早于 5.2.2.1 的 SDK 版本仅支持设置为免打扰状态（不接收推送通知）或提醒状态（接收推送通知）。

管理会话类型的免打扰级别

从 SDK 5.2.2.1 版本开始，支持在即时通讯业务中由用户（userId）为指定类型的会话（conversationType）设置免打扰级别，支持单聊、群聊、超级群会话。

设置指定会话类型免打扰级别

> 提示
>
> 该接口在 RCChannelClient 中，从 5.2.2.1 版本开始支持。

为用户设置指定会话类型（conversationType）的免打扰级别，支持单聊、群聊、超级群会话。

```
[[RCChannelClient sharedChannelManager] setConversationTypeNotificationLevel:"会话类型"
level:RCPushNotificationLevelDefault
success:^() {}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。不支持聊天室类型，因为聊天室默认就是不接受消息提醒的。
level	RCPushNotificationLevel	• `-1`： 全部消息通知 • `0`： 未设置（用户未设置情况下，默认以群 或者 APP级别的默认设置为准，如未设置则全部消息都通知） • `1`： 仅针对 @ 消息进行通知 • `2`： 仅针对 @ 指定用户进行通知 如：@张三 则张三可以收到推送，@所有人 时不会收到推送。 • `4`： 仅针对 @ 群全员进行通知，只接收 @所有人 的推送信息。 • `5`： 不接收通知
successBlock	block	回调接口
errorBlock	block	回调接口

移除指定会话类型的免打扰级别

如需移除指定会话类型的免打扰级别设置，请调用设置接口，并将 `level` 参数传入 RCPushNotificationLevelDefault。

查询指定会话类型的免打扰级别

> 提示
>
> 该接口在 RCChannelClient 中，从 5.2.2.1 版本开始支持。

查询当前用户为指定会话类型（conversationType）设置的免打扰级别。

```
[[RCChannelClient sharedChannelManager] getConversationTypeNotificationLevel:"会话类型"
success:^(RCPushNotificationLevel level) {}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。不支持聊天室类型，因为聊天室默认就是不接受消息提醒的。

参数	类型	说明
level	RCPushNotificationLevel	• `-1`： 全部消息通知 • `0`： 未设置（用户未设置情况下，默认以群 或者 APP级别的默认设置为准，如未设置则全部消息都通知） • `1`： 仅针对 @ 消息进行通知 • `2`： 仅针对 @ 指定用户进行通知 如：@张三 则张三可以收到推送，@所有人 时不会收到推送。 • `4`： 仅针对 @ 群全员进行通知，只接收 @所有人 的推送信息。 • `5`： 不接收通知
successBlock	block	回调接口
errorBlock	block	回调接口

多端同步免打扰/置顶 多端同步免打扰/置顶

更新时间:2024-08-30

SDK 提供了会话状态（置顶或免打扰）同步机制，通过设置会话状态同步监听器，当在其它端修改会话状态时，可在本端实时监听到会话状态的改变。

实现此功能需要遵守 RCConversationStatusChangeDelegate 协议。

设置代理委托

```
[[RCIMClient sharedRCIMClient] setRCConversationStatusChangeDelegate:self];
```

代理方法

```
@protocol RCConversationStatusChangeDelegate <NSObject>

/**
IMLib 会话状态同步的回调

@param conversationStatusInfos 改变过的会话状态的数组
*/
- (void)conversationStatusDidChange:(NSArray<RCConversationStatusInfo *> *)conversationStatusInfos;

@end
```

RCConversationStatusInfo 说明

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
channelId	NSString	所属会话的业务标识
conversationStatusType	RCConversationStatusType	会话状态改变的类型（RCConversationStatusType_Mute = 1//免打扰; RCConversationStatusType_Top = 2//置顶）
conversationStatusvalue	int	会话状态改变的值，详见 conversationStatusvalue 说明
notificationLevel	RCPushNotificationLevel	会话免打扰类型，详见 RCPushNotificationLevel 说明

- conversationStatusvalue 说明
  - 如果 conversationStatusType == RCConversationStatusType_Mute，conversationStatusvalue = 1 是提醒，conversationStatusvalue = 0 是免打扰。
  - 如果 conversationStatusType == RCConversationStatusType_Top，conversationStatusvalue = 0 是不置顶，conversationStatusvalue = 1 是置顶。
- RCPushNotificationLevel 说明

  当 conversationStatusType = RCConversationStatusType_Mute 时，notificationLevel 值为有效值。

```
typedef NS_ENUM(NSInteger, RCPushNotificationLevel) {
/*!
全部消息通知（接收全部消息通知 -- 显示指定关闭免打扰功能）
@discussion 超级群设置全部消息通知时
@ 消息一定收到推送通知
普通消息的推送频率受到超级群服务端默认推送频率设置的影响，无法做到所有普通消息都通知
*/
RCPushNotificationLevelAllMessage = -1,
/*!
未设置（向上查询群或者APP级别设置）,存量数据中0表示未设置
*/
RCPushNotificationLevelDefault = 0,
/*!
群聊，超级群 @所有人 或者 @成员列表有自己 时通知；单聊代表消息不通知
*/
RCPushNotificationLevelMention = 1,
/*!
群聊，超级群 @成员列表有自己时通知，@所有人不通知；单聊代表消息不通知
*/
RCPushNotificationLevelMentionUsers = 2,
/*!
群聊，超级群 @所有人通知，其他情况都不通知；单聊代表消息不通知
*/
RCPushNotificationLevelMentionAll = 4,
/*!
消息通知被屏蔽，即不接收消息通知
*/
RCPushNotificationLevelBlocked = 5,
};
```

示例代码

```
- (void)conversationStatusDidChange:(NSArray<RCConversationStatusInfo *> *)conversationStatusInfos {

for (RCConversationStatusInfo *statusInfo in conversationStatusInfos) {
/*!
如果 conversationStatusType = RCConversationStatusType_Mute，conversationStatusvalue = 1 是提醒，conversationStatusvalue = 0 是免打扰。
*/
if (RCConversationStatusType_Mute == statusInfo.conversationStatusType) {
int blockStatus = statusInfo.conversationStatusvalue;//1 是提醒，0 是免打扰
}
/*!
如果 conversationStatusType = RCConversationStatusType_Top，conversationStatusvalue = 0 是不置顶，conversationStatusvalue = 1 是置顶。
*/
else if (RCConversationStatusType_Top == statusInfo.conversationStatusType) {
BOOL isTop = (statusInfo.conversationStatusvalue == 1);//0 是不置顶， 1 是置顶
}
}
}
```

多端同步阅读状态 多端同步阅读状态

更新时间:2024-08-30

在即时通讯业务中，同一用户账号可能在多个设备上登录。仅在开通多设备消息同步服务后，融云会在多个设备之间同步消息数据，但设备上的会话中消息的已读/未读状态仅存储在本地。因此，应用程序可能希望将用户当前登录设备上指定会话的已读/未读状态同步给其他终端。

> 提示
>
> 如果 SDK 版本 ≦ 5.6.2，不支持多端同步 系统会话的阅读状态。

发起会话阅读状态同步

```
[[RCIMClient sharedRCIMClient] syncConversationReadStatus:ConversationType_PRIVATE targetId:@"targetId" time:1626422416809
success:^{

} error:^(RCErrorCode nErrorCode) {

}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 Id
timestamp	long long	消息时间戳，表示小于或等于此时间戳之前的消息为已读

接收多端同步阅读状态的通知

IMLib 在收到同步阅读状态通知时会分发一个 RCLibDispatchReadReceiptNotification 的通知。此时消息数据库中消息状态已经改为已读，应用程序可根据通知更新会话 UI，将 messageTime 以前的消息 UI 为已读。

```
[[NSNotificationCenter defaultCenter]
addObserver:self
selector:@selector(didReceiveReadReceiptNotification:)
name:RCLibDispatchReadReceiptNotification
object:nil];
```

Notification 的 object 为 nil，userInfo 为 NSDictionary 对象。

```
- (void)didReceiveReadReceiptNotification:(NSNotification *)notification {

RCConversationType conversationType = (RCConversationType)[notification.userInfo[@"cType"] integerValue];
long long readTime = [notification.userInfo[@"messageTime"] longLongValue];
NSString *targetId = notification.userInfo[@"tId"];
NSString *senderUserId = notification.userInfo[@"fId"];
}
```

NSDictionary 对象的 key 值说明:

key值	类型	说明
cType	NSNumber	会话类型
messageTime	NSNumber	消息时间戳，表示小于或等于此时间戳之前的消息为已读

key值	类型	说明
tId	NSString	A 的 targetID
fId	NSString	B 的 targetId

群组业务概述 群组业务概述

更新时间:2024-08-30

群聊是即时通讯类应用中常见的多人通讯方式，一般包含两个及以上的用户。融云的群组业务支持丰富的群组成员管理、禁言管理等特性，支持离线消息推送和历史消息记录漫游，可用于兴趣群、办公群、客服服务沟通等。

群组业务要点如下：

- 融云只负责将消息传达给群组中的所有用户，不维护群组成员的资料（头像、名称、群成员名片等），需要由开发者应用服务器维护。
- 创建/解散/加入/退出群组等群组管理操作，必须由 App 服务器请求融云服务端 API 实现。融云客户端 SDK 不提供相应方法。详见下方群组管理功能。
- App Key 下可创建的群组数量没有限制，单个群组默认成员上限为 3000 人。可提交工单修改群成员人数上限。
- 单个用户可加入的群组数量无限制。
- 从控制台 IM 服务管理页面为 App Key 开启单群聊消息云端存储服务后，可使用融云提供的消息存储服务，实现消息历史记录漫游。**IM 旗舰版**或**IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明页面及计费说明文档为准。

服务配置

客户端 SDK 默认支持群组业务，不需要申请开通。

群组业务的部分基础功能与增值服务可以在控制台的免费基础功能和 IM 服务管理页面进行开通和配置。

客户端 SDK 使用须知

> 提示
>
> - 仅 IMKit 提供开箱即用的群聊会话 UI 组件。IMLib 不提供开箱即用的群聊会话 UI 组件。IMKit 依赖 IMLib，因此 IMKit 具备 IMLib 的全部能力。
> - 客户端不提供群组管理的 API。群组管理需要由 App 服务端调用相应的融云服务端 API（Server API）接口完成。

群组管理功能

融云不会托管用户，也不管理群组的业务逻辑，因此群的业务逻辑全部需要在 App 服务器进行实现。

> 提示
>
> 群主、群管理员、群公告、邀请入群、群号搜索等均为群组业务逻辑，需在 App 侧自行实现。

对于客户端开发人员来说，创建群组、解散等基础管理操作只需要与 App 自身的业务服务端交互即可，由 App 服务端负责调用相应的融云服务端 API（Server API）接口完成相关操作。

下表列出了融云服务端提供的群组基础管理接口。注意，客户端不提供群组管理的 API。

功能分类	功能描述	融云服务端 API
创建、解散群组	提供创建者用户 ID、群组 ID、和群名称，向融云服务端申请建群。如解散群组，则群成员关系不复存在。	创建群组、解散群组

功能分类	功能描述	融云服务端 API
加入、退出群组	加入群组后，默认可查看入群以后产生的新消息。退出群组后，不再接收该群的新消息。	加入群组、退出群组
修改融云服务端的群组信息	修改在融云推送服务中使用的群组信息。	刷新群组信息
查询群组成员	查询指定群组所有成员的用户 ID 信息。	查询群组成员
查询用户所在群组	根据用户 ID 查询该用户加入的所有群组，返回群组 ID 及群组名称。融云不存储群组资料信息，群组资料及群成员信息需要开发者在应用服务器自行维护，如应用服务端维护的用户群组关系有缺失时，可通过此接口来核对校验。	查询用户所在群组
同步用户所在群组	向融云服务端同步指定用户当前所加入的所有群组，防止应用中的用户群组信息与融云服务端的用户所属群信息不一致。如果在集成融云服务前 App Server 上已有群组及成员数据，第一次连接融云服务器时，可使用此接口向融云同步已有的用户与群组对应关系。	同步用户所在群组
群组单人禁言	在指定的单个群组中或全部群组中，禁言一个或多个用户。被禁言用户可以接收查看群组中其他用户消息，但不能通过客户端 SDK 发送消息。	单人禁言
群组全体禁言	将群组全体成员禁言。被禁言群组的所有成员均不能发送消息，需要某些用户可以发言时，可将此用户加入到群禁言用户白名单中。	全体禁言
群组禁言用户白名单	群组被整体禁言后，禁言白名单中用户可以发送群消息。	全体成员禁言白名单

群聊消息功能

群聊消息功能与单聊业务类似，共用部分 API 及配置。

功能	描述	客户端 API	融云服务端 API
发送消息	可发送普通消息与媒体消息，例如文本、图片、GIF 等，或自定义消息。支持在发送消息时添加 @ 信息。	发送消息	发送群聊消息
发送群聊定向消息	可发送普通消息与媒体消息给群组中的指定的一个或多个成员，其他成员不会收到该消息。	发送群定向消息	发送群聊定向消息
接收消息	监听并实时接收消息，或在客户端上线时接收离线消息。	接收消息	不适用
群聊已读回执	发送群消息后如需要查看消息的阅读状态，需要先发送回执请求，在通过接受者的响应获取已读数据。	群聊已读回执	不适用
离线消息	支持离线消息存储，存储时间可设置（1 ～ 7 天），默认存储 7 天内的所有群消息，支持调整存储时长与存储的群消息数量。	管理离线消息存储配置	不提供该 API
离线消息推送	离线状态下，群组中有新消息时，支持 Push 通知。	APNs 推送开发指南	不提供该 API
撤回消息	消息发送成功后可撤回该条消息。	撤回消息	撤回消息
本地搜索消息	消息存储在本地（移动端），支持按关键字或用户搜索本地指定会话的消息内容。	搜索消息	不提供该 API
获取历史消息	从本地数据库或远端获取历史消息。注意，从远端获取历史消息需要开通单群聊消息云存储服务，默认存储时长为 6 个月。	获取历史消息	不提供该 API
获取历史消息日志	融云服务端可以保存 APP 内所有会话的历史消息记录，历史消息记录以日志文件方式提供，并已经过压缩。您可以使用服务端 API 获取、删除指定 App 的历史消息日志	不提供该 API	获取历史消息日志
本地插入消息	在本地数据库中插入消息。本地插入的消息不会实际发送给服务器和对方。	插入消息	不适用
删除消息	支持按会话删除本地和存储在服务器的指定消息或会话中全部历史消息。	删除消息	消息清除
单群聊消息扩展	为原始消息增加状态标识（扩展数据为 KV 键值对），提供添加、删除、查询扩展信息的接口。	消息扩展	单/群聊消息扩展
自定义消息类型	如果内置消息类型满足不了您的需求，可以自定义消息类型。支持自定义普通消息类型与自定义媒体消息类型。	自定义消息类型	不适用

默认新入群的成员的用户仅可接收加入群组后产生的消息。如果需要查看入群之前的历史消息，请为 App Key 开启以下两项服务（请注意区分开发/生产环境）：

从控制台 IM 服务管理 页面开启单群聊消息云端存储服务。开启该服务后，可使用融云提供的消息存储服务，实现消息历史记录漫游。**IM 旗舰版或IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。

- 从控制台免费基础功能页面开启新用户获取加入群组前历史消息。

群聊会话功能

群聊会话功能与单聊业务类似，共用部分 API 及配置。

功能	描述	客户端 API	融云服务端 API
获取会话	SDK 会根据收发的消息在本地数据库中生成对应会话。您可以从本地数据库获取 SDK 生成的会话列表。	获取会话	不提供该 API
获取会话未读消息	从指定会话中获取未读消息，可满足 App 跳转到第一条未读消息、展示全部未读 @ 消息的需求。	获取会话未读消息	不提供该 API
处理会话未读消息数	获取或清除会话中的未读消息数，可用于 UI 展示。	处理会话未读消息数	不提供该 API
删除会话	从 SDK 生成的会话列表中删除一个会话或多个会话	删除会话	不提供该 API
会话草稿	保存一条草稿内容至指定会话。	会话草稿	不提供该 API
输入状态	可设置指定的群聊会话，收到新的消息后是否进行提醒，默认进行新消息提醒。	输入状态	不提供该 API
管理会话标签	创建和管理标签信息数据，用于对会话进行标记分组。每个用户最多可以创建 20 个标签。App 用户创建的标签信息数据会同步融云服务端。	管理标签信息数据	不提供该 API
设置与使用会话标签	使用会话标签对会话进行分组。	设置与使用会话标签	不提供该 API
会话置顶	在会话列表中将指定会话置顶。	会话置顶	会话置顶
会话免打扰	控制用户在客户端设备离线时，是否可针对离线消息接收推送通知。支持按照会话或按会话类型设置免打扰。	免打扰功能概述	免打扰功能概述
多端同步会话免打扰/置顶状态	SDK 提供了会话状态（置顶或免打扰）同步机制，通过设置会话状态同步监听器，当在其它端修改会话状态时，可在本端实时监听到会话状态的改变。	多端同步免打扰/置顶	不适用
多端同步阅读状态	在同一用户账户的多个设备间主动同步会话的阅读状态。	多端同步阅读状态	不适用

与聊天室和超级群的区别

您可以通过以下文档了解业务类型之间的区别及所有功能：

- 即时通讯开发指导 · 业务类型介绍
- IM 尊享版、IM 旗舰版功能对照表

发送群定向消息 发送群定向消息

更新时间:2024-08-30

SDK 支持往群聊会话中发送定向消息。定向消息只会发送给指定用户，群聊会话中的其它用户不会收到这条消息。

开通服务

使用发送群组定向消息功能无需开通服务。注意，如需将群组定向消息存入服务端历史消息记录，需要开通以下服务：

- 单群聊历史消息云存储服务，可前往控制台 IM 服务管理 页面为当前使用的 App Key 开启服务。IM 旗舰版或IM 尊享版可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。
- 群定向消息云存储服务，需要提交工单 申请开通。

默认情况下，客户端发送与接收的群定向消息默认都不会存入历史消息服务，因此客户端调用获取历史消息的 API 时，从融云服务端返回的结果中不会包含当前用户发送、接收的群组定向消息。

发送定向普通消息

在群组中发送普通消息给指定的单个或多个用户。

以发送文本消息为例，您需要构造 RCMessage，然后调用 sendDirectionalMessage:toUserIdList:pushContent:pushData:attached:successBlock:errorBlock: 发送消息。注意，RCMessage 中仅保存群组 ID（Target ID），不会保存接收用户 ID 列表。

```
RCTextMessage *text = [RCTextMessage messageWithContent:@"你好"];
RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_GROUP targetId:@"Group1"
direction:MessageDirection_SEND messageId:-1 content:text];

[[RCCoreClient sharedCoreClient] sendDirectionalMessage:message
toUserIdList:@[@"user1",@"user2"]
pushContent:nil
pushData:nil
successBlock:^(RCMessage *successMessage) {

} errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {

}];
```

参数	类型	说明
message	RCMessage	消息对象
userIdList	NSArray	将要发送的用户列表
pushContent	NSString	自定义推送显示内容
pushData	NSString	远程推送的附加信息
successBlock	Block	发送消息成功的回调
errorBlock	Block	发送消息失败的回调，其中包含错误码 RCErrorCode 和发送失败的消息

发送定向媒体消息

> 提示
>
> SDK 从 5.2.5 版本开始支持往群聊中发送定向媒体消息。

在群聊中发送多媒体消息给指定的单个或多个用户。

以发送图片消息为例，您需要构造 RCMessage，然后调用 sendDirectionalMediaMessage:toUserIdList:pushContent:pushData:attached:progress:successBlock:errorBlock:cancel: 发送消息。注意，RCMessage 中仅保存群组 ID（Target ID），不会保存接收用户 ID 列表。

```
RCImageMessage *imageMessage = [RCImageMessage messageWithImageURI:@"https://test.png"];
RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_GROUP targetId:@"Group1"
direction:MessageDirection_SEND messageId:-1 content:imageMessage];

[[RCCoreClient sharedCoreClient] sendDirectionalMediaMessage:message toUserIdList:@[@"user1",@"user2"] pushContent:nil
pushData:nil progress:^(int progress, RCMessage * _Nonnull progressMessage) {
// 多媒体上传进度回调
} successBlock:^(RCMessage * _Nonnull successMessage) {
// 发送成功
} errorBlock:^(RCErrorCode nErrorCode, RCMessage * _Nonnull errorMessage) {
// 发送失败
} cancel:^(RCMessage * _Nonnull cancelMessage) {
// 用户取消发送
}];
```

参数	类型	说明
message	RCMessage	消息对象
userIdList	NSArray	将要发送的用户列表
pushContent	NSString	自定义推送显示内容
pushData	NSString	远程推送的附加信息
progress	Block	发送消息进度更新的回调
successBlock	Block	发送消息成功的回调
errorBlock	Block	发送消息失败的回调，其中包含错误码 RCErrorCode 和发送失败的消息
cancel	Block	用户取消消息发送的回调

获取定向消息的目标用户列表

> 提示
>
> 从 5.8.0 版本起，SDK 支持获取定向消息的目标用户列表。这个功能仅适用于普通群和超级群消息。

通过检查消息 RCMessage 对象的 directedUserIds 属性，您可以获取定向消息的目标用户列表。如果该列表为空，表示此消息不是定向消息。

```
// 获取定向消息的目标用户列表
NSArray<NSString *> *directedUserIds = message.directedUserIds;
```

聊天室概述 聊天室概述

更新时间:2024-08-30

聊天室（Chatroom）提供了一种不设用户上限，支持高并发消息处理的业务形态，可用于直播、社区、游戏、广场交友、兴趣讨论等场景。聊天室业务要点如下：

- App Key 下可创建的聊天室数量没有限制，单个聊天室成员数量没有限制。
- 聊天室具有自动销毁机制，默认情况下所有聊天室会在不活跃（连续时间段内无成员进出且无新消息）达到 1 小时后踢出所有成员并自动销毁，可延长该时间，也可配置为定时自动销毁。详见服务端文档聊天室销毁机制。
- 聊天室具有离线成员自动退出机制。满足默认预设条件时，融云服务端会踢出聊天室成员，详见退出聊天室。
- 聊天室本地消息会在退出聊天室时删除。**IM 旗舰版** 与 **IM 尊享版**客户可选择启用**聊天室消息云端存储**功能，将消息存储在融云服务端。具体功能与费用以融云官方价格说明页面及计费说明文档为准。
- 聊天室不具备离线消息转推送功能，只有在线的聊天室成员可接收聊天室消息。

客户端 UI 框架参考设计

聊天室产品暂不提供聊天室会话专用的 UI 组件。您可以参考以下 UI 框架设计了解聊天室的设计思路。

- 下图**聊天室**标签中为聊天室消息列表。
- 下图**聊天室管理**窗口中展示了聊天室支持的部分能力，如禁言、封禁、白名单等。

服务配置

客户端 SDK 默认支持聊天室，不需要申请开通。

聊天室的部分基础功能与增值服务可以在控制台的免费基础功能和 IM 服务管理页面进行开通和配置。

客户端 SDK 使用须知

提示

IMKit 依赖 IMLib，因此 IMKit 具备 IMLib 的全部能力，包括聊天室。但请注意 IMKit 不提供开箱即用的聊天室会话 UI 组件。

聊天室功能接口

聊天室会话关系由融云负责建立并保持连接。SDK 提供加入、退出等部分聊天室管理接口。更多聊天室管理功能需要配合使用即时通讯服务端 API。下表描述了融云聊天室主要的功能接口。

功能分类	功能描述	客户端 API	融云服务端 API
创建与销毁聊天室	手动创建聊天室，或手动销毁聊天室。**注意**：客户端 SDK 无单独的创建聊天室 API。客户端不提供手动销毁聊天室 API。	不提供该 API	创建房间、销毁房间
加入与退出聊天室	加入已存在的聊天室，请确保聊天室 ID 已存在。加入与退出聊天室仅客户端提供 API。**注意**：客户端有废弃接口可支持在聊天室不存在时创建聊天室再加入，但已不推荐使用。	加入聊天室、退出聊天室	不提供该 API
查询聊天室房间与用户信息	• 查询聊天室房间的基础信息，包括聊天室 ID、名称、创建时间。 • 查询聊天室成员信息，支持获取聊天室成员用户 ID、加入时间，最多返回 500 个成员信息，支持按加入时间排序。	查询聊天室信息	查询房间信息
聊天室保活	添加一个或多个聊天室到聊天室保活列表。在保活列表中的聊天室不会被融云服务端自动销毁。	不提供该 API	保活房间
聊天室属性管理	在指定聊天室中设置自定义属性。比如在语音直播聊天室场景中，利用此功能记录聊天室中各麦位的属性；或在狼人杀等卡牌类游戏场景中记录用户的角色和牌局状态等。 聊天室属性以 Key-Value 的方式进行存储，支持设置、删除与查询属性，支持批量和强制操作。	聊天室属性	属性管理（KV）
封禁/解封聊天室用户	封禁一个或多个聊天室成员。被封禁成员将被踢出指定聊天室，并在封禁时间内不能再进入此聊天室中。	不提供该 API	成员封禁
聊天室用户白名单	需在 IM 服务管理 页面普通服务下开通后使用。**IM 旗舰版或IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。 用户被加入某个聊天室的白名单后，在该聊天室消息量较大的情况下，该用户发送的消息不会被丢弃；并且用户也不会被融云服务端自动踢出该聊天室。	不提供该 API	聊天室白名单服务
发送聊天室消息	发送聊天室消息。	发送消息	发送聊天室消息
撤回聊天室消息	撤回聊天室消息。	撤回消息	消息撤回
获取聊天室历史消息	获取聊天室历史消息。	获取聊天室历史消息	历史消息日志
聊天室低级别消息	需在 IM 服务管理 页面普通服务下开通后使用。**IM 旗舰版或IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。 如果消息类型在低级别消息列表中，该类型的消息全部视为低级别消息。当服务器负载高时，高级别的消息优先保留，低级别消息则优先丢弃。默认情况下，所有消息均为高级别消息。	不提供该 API	聊天室消息优先级服务
聊天室消息白名单	需在 IM 服务管理 页面普通服务下开通后使用。**IM 旗舰版或IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。 如果消息类型在聊天室消息白名单中，该类型的消息全部受到保护，在聊天室消息量较大的情况下也不会被丢弃。	不提供该 API	聊天室白名单服务
聊天室成员禁言	在指定的某个聊天室中，禁言一个或多个成员。聊天室成员被禁言后，可以接收并查看聊天室中用户聊天信息，但不能通过往该聊天室内发送消息。	不提供该 API	单人禁言
全体成员禁言	设置某一聊天室全部成员禁言，或取消指定聊天室全部成员禁言状态。设置全体群成员禁言后，该聊天室的所有成员均不能通过客户端 SDK 往该群组内发送消息。	不提供该 API	全体禁言

功能分类	功能描述	客户端 API	融云服务端 API
全体禁言白名单	添加一个或多个群成员到聊天室全体成员禁言白名单。聊天室成员被添加到白名单后，即使该聊天室处于全体成员禁言状态，该成员仍可通过客户端 SDK 往该聊天室发送消息。	不提供该 API	全体禁言
全局禁言聊天室成员	需在 IM 服务管理 页面普通服务下开通后使用。**IM 旗舰版或IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。 添加一个或多个用户到聊天室全局禁言列表中，列表中的用户在应用下的所有聊天室中都无法发送消息。	不提供该 API	全局禁言

与群组和超级群的区别

您可以通过以下文档了解业务类型之间的区别及所有功能：

- 即时通讯开发指导 · 业务类型介绍
- IM 尊享版、IM 旗舰版功能对照表

聊天室服务配置

更新时间:2024-08-30

聊天室业务本身不需要单独申请开通，但部分聊天室服务需要在控制台开通与配置，例如**聊天室广播消息**、**聊天室消息云端存储**、以及与聊天室相关的回调地址等。

聊天室服务配置主要在**免费基础功能**和 **IM 服务管理**页面。

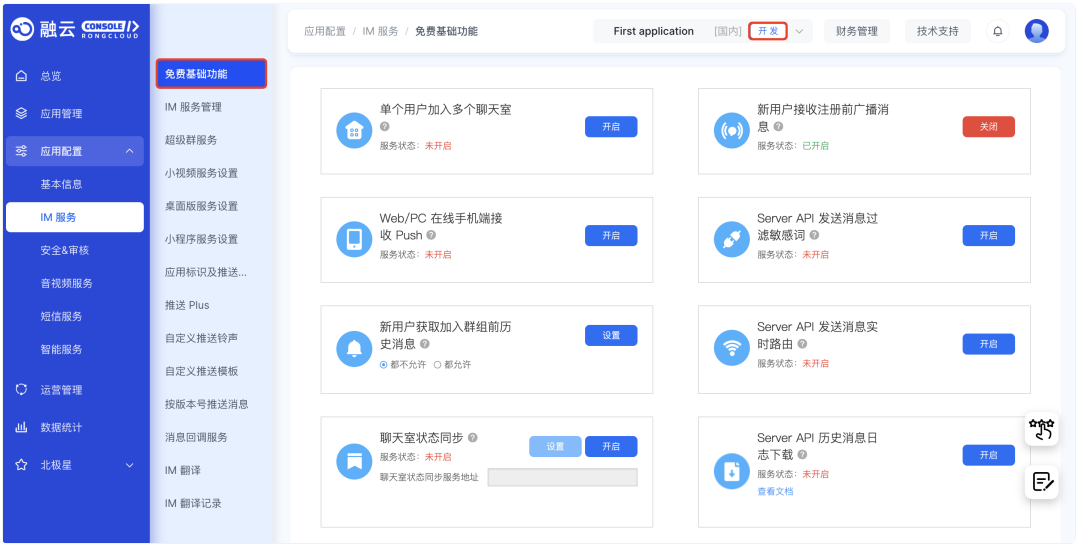
免费基础功能

以下是聊天室业务提供的免费基础功能：

- **单个用户加入多个聊天室**：默认一个用户只能加入一个聊天室中，开启后一个用户可以同时加入到多个聊天室中。
- **聊天室状态同步**：聊天室状态同步是融云提供的服务端回调服务，需同时提供可正常访问的回调地址。配置成功后，在应用下聊天室发生状态变化时，将实时同步到开发者的应用服务器地址，目前支持的同步状态包括：创建、销毁、成员加入、成员退出聊天室。详见 IM 服务端文档「聊天室管理」下的聊天室状态同步。
- **加入聊天室获取指定消息设置**：默认加入聊天室时可最多获取全部消息类型的最近 50 条消息，开启后可设置指定消息类型获取。
- **聊天室销毁等待时间**：
  1. 可以支持配置不活跃聊天室销毁的等待时间，默认等待时间为1小时，即超过1小时不活跃即被销毁，客户可以按需调整这个时间，最长可设置24小时。
  2. 聊天室销毁时，会向聊天室成员发送聊天室销毁通知，以方便客户可以在聊天室销毁后，在终端自定义一些操作(依赖 5.1.1 及以后版本)。
  3. 聊天室增加 sessionid，在聊天室生存周期内保持不变，聊天室重建后重新生成。用于使用相同聊天室ID，多次开播时，客户端能区分出来。
- **聊天室属性自定义设置**：可在指定聊天室中设置自定义属性，用于语音直播聊天室场景的会场属性同步或狼人杀等卡牌类游戏场景中记录用户的角色和牌局状态等，详见「聊天室业务」下的**聊天室属性管理（KV）**文档。如果 App 业务服务端需要融云提供聊天室属性变更数据同步，需要提供可正常访问的回调地址，配置成功后，自动开启融云提供的服务端回调服务，详见 IM 服务端文档「聊天室管理」下的聊天室属性同步。

修改服务配置

访问控制台免费基础功能页面，可调整聊天室业务相关的免费基础功能配置。



IM 旗舰版/尊享版功能

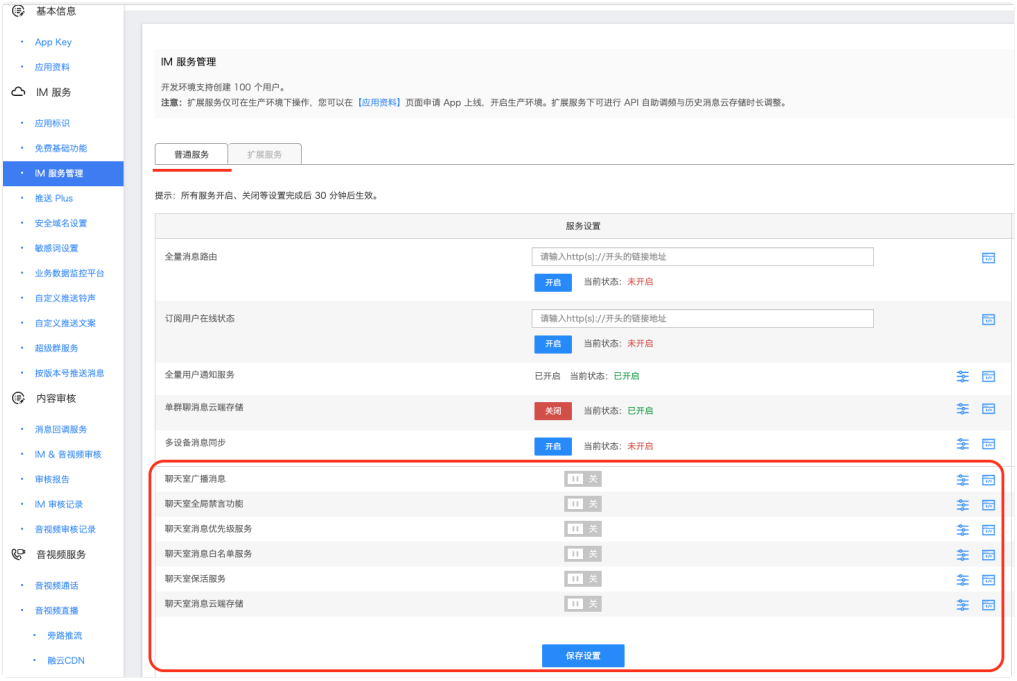
下图显示了控制台 IM 服务管理 页面与聊天室业务相关的普通服务配置。

> 开发环境下可以免费使用。生产环境下，IM 旗舰版或 IM 尊享版才能使用以下服务。

- **聊天室广播消息**：向应用中的所有聊天室发送一条消息，单条消息最大 128k。详见 IM 服务端文档「消息管理」下的发送全体聊天室广播消息。
- **聊天室全局禁言功能**：当不想让某一用户在所有聊天室中发言时，可将此用户添加到聊天室全局禁言中，被禁言用户可接收查看聊天室中用户聊天信息，但不能发送消息。详见 IM 服务端文档「聊天室用户管理」下的全局禁言用户。
- **聊天室消息优先级服务**：在指定聊天室中设置指定类型的消息为低级别消息。当服务器负载高时低级别消息优先被丢弃，这样可以确保重要的消息不被丢弃。详见 IM 服务端文档「聊天室消息优先级服务」下的添加低级别消息。
- **聊天室白名单服务**：开通后，可以使用以下功能对应的 Server API：
  - 聊天室用户白名单：可用于保护指定聊天室中的重要用户，支持按聊天室设置白名单用户。例如，App 业务中指定聊天室中的管理员、主播等重要角色的用户。
  - 聊天室消息白名单：可用于保护 App 下所有聊天室中的指定消息类型。例如 App 业务中自定义的红包消息。
- **聊天室保活服务**：当聊天室中 1 小时无人说话，同时没有人加入聊天室时，融云服务端会自动把聊天室内所有成员踢出聊天室并销毁聊天室。保活的聊天室不会被自动销毁，可以调用 API 接口销毁聊天室。详见 IM 服务端文档「聊天室管理」下的保活聊天室。
- **聊天室消息云端存储**：聊天消息保存在云端，用户进入聊天室后，可以查看聊天室中以前的消息，历史消息默认保存 2 个月。
- **加入聊天室获取指定消息配置**：加入聊天室时只返回指定类型的消息，不返回其他类型的消息。

修改服务配置

访问开发后台 IM 服务管理页面，切换到普通服务标签下，可启用以下聊天室服务配置开关。



您还可以在扩展服务标签下对部分服务的具体配置进行调整。

查询聊天室房间信息 查询聊天室房间信息

更新时间:2024-08-30

获取聊天室的信息，可返回以下数据：

- 聊天室成员总数
- 指定数量（最多 20 个）的聊天室成员的列表，包括该成员的用户 ID 以及加入聊天室的时间

> 提示
>
> **频率限制：单个设备每秒钟支持调用一次，每分钟单个设备最多调用 20 次。**

您可以使用 RCIMClient 或 RCChatRoomClient 下的 getChatRoomInfo 方法：

```
[[RCIMClient sharedRCIMClient] getChatRoomInfo:@"chatroomId"
count:0
order:RC_ChatRoom_Member_Asc
success:^(RCChatRoomInfo *chatRoomInfo) {
// Get ChatRoomInfo properties
NSString *targetId = chatRoomInfo.targetId;
int totalMemberCount = chatRoomInfo.totalMemberCount;
NSArray<RCChatRoomMemberInfo *> *memberInfoArray = chatRoomInfo.memberInfoArray;

// Get ChatRoomMemberInfo properties
for (RCChatRoomMemberInfo *memberInfo in memberInfoArray) {
NSString *userId = memberInfo.userId;
long long joinTime = memberInfo.joinTime;

// TODO
}
}
error:^(RCErrorCode status) {
// Handle error
}];
```

参数	类型	说明
targetId	NSString	聊天室 ID。有效值最大长度为 64 个字符。
count	int	需要获取的聊天室成员数量。范围 0-20。因为聊天室一般成员数量巨大，权衡效率和用户体验，获取的聊天室成员数上限为 20。如果 count 为 0，则返回的聊天室信息仅包含成员总数，不包含具体的成员列表。
order	RCChatRoomMemberOrder	需要获取的成员列表的顺序（最早加入或是最晚加入的部分成员）。RC_ChatRoom_Member_Asc（1） 表示升序，从最早加入聊天室成员开始，按加入时间递增的顺序获取成员列表；RC_ChatRoom_Member_Desc（2）表示降序，从最晚加入聊天室成员开始，按加入时间递减的顺序获取成员列表。
successBlock	Block	获取成功的回调。返回 RCChatRoomInfo，其中包含按要求获取的聊天室成员列表结果。详见下方 **RCChatRoomInfo 类说明**。
errorBlock	Block	获取失败的回调。 status 中返回错误码 RCErrorCode 。

- **RCChatRoomInfo 类说明：**

参数	类型	说明
targetId	NSString	聊天室信息
memberOrder	RCChatRoomMemberOrder	获取的成员列表的顺序

参数	类型	说明
memberInfoArray	NSArray	聊天室成员列表，数组元素为聊天室成员对象 RCChatRoomMemberInfo，内部包含用户 ID（`userId`）和Unix 时间戳格式的加入时间（`joinTime`），单位为毫秒。 列表中的成员按加入时间从旧到新排列。
totalMemberCount	int	当前聊天室的成员总数

加入聊天室　　加入聊天室

更新时间:2024-08-30

加入聊天室分为以下几种情况：

- （推荐）应用程序后端通过即时通讯服务端 API 创建聊天室，将聊天室 ID 下分发至客户端。客户端获取聊天室 ID 后可加入聊天室。
- （已废弃）应用程序通过直接调用客户端 SDK 加入聊天室方法，创建并加入该聊天室。这种方式已不推荐，相关 API 已于 5.6.3 版本废弃，未来可能会删除。
- SDK 断网重连后会自动重新加入聊天室，不需要应用程序处理。

应用程序后端可以通过提前注册服务端回调聊天室状态同步，收取聊天室创建成功与成员加入等事件通知。客户端可以通过监听聊天室事件收取相关通知。

局限

- 默认同一用户不能同时加入多个聊天室。用户加入新的聊天室后会自动退出之前的聊天室。您可以在控制台启用单个用户加入多个聊天室。
- 客户端的加入聊天室方法允许在加入时获取最新的历史消息（默认 10 条，最多 50 条），但不支持指定消息类型。您可以在控制台启用加入聊天室获取指定消息配置，限制加入聊天室时只获取指定类型的消息。

加入已存在的聊天室

> 提示
>
> IMLib 从 5.6.3 开始新增方法，支持在成功回调中返回 RCJoinChatRoomResponse，其中包含聊天室的创建时间、成员数量、聊天室禁言状态、用户禁言状态等信息。

如果 IMLib 版本 ≧ 5.6.3，可使用 RCChatRoomClient 的 joinExistChatRoom:messageCount:successBlock:errorBlock: 方法加入一个已存在的聊天室。

```
[[RCChatRoomClient sharedChatRoomClient] joinExistChatRoom:@"your_chat_room_id"
messageCount:-1
successBlock:^(RCJoinChatRoomResponse *response) {
NSLog(@"Successfully joined the chat room");
// 创建时间（毫秒时间戳）
long long createTime = response.createTime;
// 成员数量
NSInteger memberCount = response.memberCount;
// 是否全局禁言
BOOL isAllChatRoomBanned = response.isAllChatRoomBanned;
// 是否当前用户被禁言
BOOL isCurrentUserBanned = response.isCurrentUserBanned;
// 当前用户是否在此聊天室被禁言
BOOL isCurrentChatRoomBanned = response.isCurrentChatRoomBanned;
// 当前用户是否在此聊天室的白名单中
BOOL isCurrentChatRoomInWhitelist = response.isCurrentChatRoomInWhitelist;

}
errorBlock:^(RCErrorCode status) {
NSLog(@"Failed to join the chat room. Error code: %ld", (long)status);
}];
```

参数	类型	说明
targetId	NSString	聊天室会话 ID，最大长度为 64 个字符。
messageCount	int	进入聊天室时获取历史消息的数量，数量范围：1-50。如果传 -1，表示不获取任何历史消息。如果传 0，表示使用 SDK 默认设置（默认为获取 10 条）。

参数	类型	说明
successBlock	Block	加入聊天室成功的回调。`response` 中返回 `RCJoinChatRoomResponse` 对象，其中包含聊天室的基础信息，以及当前用户在聊天室的状态信息。
errorBlock	Block	加入聊天室失败的回调。`status` 中返回错误码 RCErrorCode 。

如果 IMLib 版本 < 5.6.3，可使用 `RCChatRoomClient` 的 joinExistChatRoom:messageCount:success:error: 方法加入已存在的聊天室。该方法在 5.6.3 版本上已废弃。

```
[[RCIMClient sharedRCIMClient] joinExistChatRoom:@"chatroomId" messageCount:20 success:^{

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
targetId	NSString	聊天室会话 ID，最大长度为 64 个字符。
messageCount	int	进入聊天室时获取历史消息的数量，数量范围：1-50。如果传 `-1`，表示不获取任何历史消息。如果传 `0`，表示使用 SDK 默认设置（默认为获取 10 条）。
successBlock	Block	加入聊天室成功的回调
errorBlock	Block	加入聊天室失败的回调。`status` 中返回错误码 RCErrorCode 。

加入聊天室

> 提示
>
> IMLib 从 5.6.3 开始废弃该方法。

`joinChatRoom` 接口会创建并加入聊天室。如果聊天室已存在，则直接加入。如果您使用了服务端回调聊天室状态同步，融云会将聊天室创建成功的通知发送到您指定的服务器地址。

```
[[RCIMClient sharedRCIMClient] joinChatRoom:@"chatroomId" messageCount:20 success:^{

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
targetId	NSString	聊天室 ID，最大长度为 64 个字符。
messageCount	int	进入聊天室时获取历史消息的数量，数量范围：1-50。如果传 `-1`，表示不获取任何历史消息。如果传 `0`，表示使用 SDK 默认设置（默认为获取 10 条）。
successBlock	Block	加入聊天室成功的回调
errorBlock	Block	加入聊天室失败的回调。`status` 中返回错误码 RCErrorCode 。

断线重连后重新加入聊天室

SDK 具备断线重连机制。重连成功后，如果当前登录用户曾经加入过聊天室，且没有退出，则 SDK 会自动重新加入聊天室，不需要 App 处理。App 可以通过监听聊天室状态收到通知。

> 提示
>
> 断网重连的场景下，一旦 SDK 重新加入聊天室成功，会自动收取一定数量的聊天室消息。
>
> - 如果 SDK 版本 < 5.3.1，SDK 不会拉取消息。

- 如果 SDK 版本 ≧ 5.3.1，SDK 会按照加入聊天室时传入的 `messageCount` 拉取固定数量的消息。拉取的消息有可能在本地已存在，App 可能需要进行排重后再显示。

监听聊天室事件 监听聊天室事件

更新时间:2024-08-30

聊天室业务为客户端 App 提供三种类型的事件委托协议：聊天室状态委托、聊天室成员变化委托、聊天室事件通知。

通过 SDK 提供的以上委托，客户端 App 可以从融云服务端获取聊天室销毁状态、用户在当前及其他客户端加入退出聊天室的状态、聊天室中成员进出的事件通知、以及聊天室中成员禁言、封禁相关的信息。

委托协议	名称	说明
RCChatRoomStatusDelegate	聊天室状态委托	接收在当前客户端上用户加入、退出聊天室的事件与聊天室销毁状态的信息。
RCChatRoomMemberDelegate	聊天室成员变化委托	接收当前所在聊天室中用户加入、退出聊天室的信息。
ChatRoomNotifyEventDelegate	聊天室事件通知委托	接收当前所在聊天室中成员禁言、封禁相关的信息；接收当前用户在其他端加入、退出聊天室相关的信息。

监听聊天室状态

要从融云服务器接收当前客户端上用户加入、退出聊天室的事件与聊天室销毁状态，您可以通过 RCChatRoomClient（或 RCIMClient） 的 setChatRoomStatusDelegate 或 addChatRoomStatusDelegate 添加一个 RCChatRoomStatusDelegate 代理。

聊天室状态事件

下方列出了聊天室状态委托协议 RCChatRoomStatusDelegate 提供的事件列表及代理方法。

```
@protocol RCChatRoomStatusDelegate <NSObject>

/*!
开始加入聊天室的回调

- Parameter chatroomId: 聊天室ID
*/
- (void)onChatRoomJoining:(NSString *)chatroomId;

/*!
加入聊天室成功的回调

- Parameter chatroomId: 聊天室ID
- Parameter response: 加入成功返回的信息
- Since: 5.6.3
*/
- (void)onChatRoomJoined:(NSString *)chatroomId response:(RCJoinChatRoomResponse *)response;

/*!
加入聊天室成功的回调

- Parameter chatroomId: 聊天室ID
*/
- (void)onChatRoomJoined:(NSString *)chatroomId __deprecated_msg("Use [RCChatRoomStatusDelegate onChatRoomJoined:response:]
instead");

/*!
加入聊天室失败的回调

- Parameter chatroomId: 聊天室ID
- Parameter errorCode: 加入失败的错误码

如果错误码是KICKED_FROM_CHATROOM或RC_CHATROOM_NOT_EXIST，则不会自动重新加入聊天室，App需要按照自己的逻辑处理。
*/
- (void)onChatRoomJoinFailed:(NSString *)chatroomId errorCode:(RCErrorCode)errorCode;

/*!
加入聊天室成功，但是聊天室被重置。接收到此回调后，还会收到 onChatRoomJoined：回调。

- Parameter chatroomId: 聊天室ID
*/
- (void)onChatRoomReset:(NSString *)chatroomId;

/*!
退出聊天室成功的回调

- Parameter chatroomId: 聊天室ID
*/
- (void)onChatRoomQuited:(NSString *)chatroomId;

/*!
聊天室被销毁的回调，用户在线的时候房间被销毁才会收到此回调。

- Parameter chatroomId: 聊天室ID
- Parameter type: 聊天室销毁原因

*/
- (void)onChatRoomDestroyed:(NSString *)chatroomId type:(RCChatRoomDestroyType)type;

@end
```

设置聊天室状态代理委托

```
// 设置 IMLib 的聊天室状态监听器
[[RCChatRoomClient sharedChatRoomClient] setChatRoomStatusDelegate:self];
```

SDK 从 5.2.5 版本开始支持添加多代理，添加代理委托的方法如下：

```
// 添加 IMLib 的聊天室状态监听器 @Since 5.2.5
[[RCChatRoomClient sharedChatRoomClient] addChatRoomStatusDelegate:self];
```

删除聊天室状态代理委托

SDK 从 5.2.5 版本开始支持添加多代理，删除代理委托的方法如下：

```
// 移除 IMLib 的聊天室状态监听器 @Since 5.2.5
[[RCChatRoomClient sharedChatRoomClient] removeChatRoomStatusDelegate:self];
```

监听聊天室成员变化

> 提示
>
> SDK 从 5.1.4 版本开始支持监听聊天室成员变化。

SDK 在 RCChatRoomClient 中提供 RCChatRoomMemberDelegate 协议，支持在当前客户端监听所在聊天室中其它成员的加入、退出行为。

开通服务

此功能需要开通服务后方可使用。如有需求，请提交工单，申请开通聊天室成员变化监听。

设置聊天室成员变化代理委托

以下代码示例显示了如何设置聊天室成员变化代理委托。从 5.6.7 版本 开始，新增返回 RCChatRoomMemberActionModel 对象的代理方法，其中增加了当前聊天室人数。

```
[RCChatRoomClient sharedChatRoomClient].memberDelegate = self;

#pragma mark - RCChatRoomMemberDelegate

- (void)memberDidChange:(NSArray<RCChatRoomMemberAction *> *)members inRoom:(NSString *)roomId {
// Handle the member change with the array of RCChatRoomMemberAction objects and the roomId
}

/**
- Since: 5.6.7
*/
- (void)memberDidChange:(RCChatRoomMemberActionModel *)actionModel {
// Handle the member change with the RCChatRoomMemberActionModel object
NSString *roomId = actionModel.roomId;
NSArray<RCChatRoomMemberAction *> *chatRoomMemberActions = actionModel.chatRoomMemberActions;
NSUInteger memberCount = actionModel.memberCount;

for (RCChatRoomMemberAction *action in chatRoomMemberActions) {
// Access the properties of RCChatRoomMemberAction object
NSString *memberId = action.memberId;
RCChatRoomMemberActionType actionType = action.action;

if (actionType == RCChatRoomMemberActionType_Join) {
NSLog(@"Action Type: Join");
} else if (actionType == RCChatRoomMemberActionType_Quit) {
NSLog(@"Action Type: Leave");
}
}
}
```

聊天室中有其他用户加入或退出时，SDK 会同时触发两个 memberDidChange 方法，都在子线程回调。RCChatRoomMemberAction 列表中包含了当前聊天室中加入或退出聊天室的成员信息。RCChatRoomMemberActionModel 额外封装了聊天室当前人数（指已加入未退出的人数）。推荐使用返回 RCChatRoomMemberActionModel 对象的代理方法。

如果当前用户由于网络原因和聊天室断开连接，则无法监听到断开连接期间的其他成员的加入、退出行为。

监听聊天室事件通知

> 提示
>
> SDK 从 5.4.5 版本开始支持监听聊天室事件通知。

要从融云服务端接收当前聊天室中成员禁言、封禁相关的信息，以及当前用户在其他端加入、退出聊天室相关的信息，您可以通过 RCChatRoomClient 的 addChatRoomNotifyEventDelegate 方法添加一个RCChatRoomNotifyEventDelegate 代理。

代理方法	触发时机	说明
chatRoomNotifyMultiLoginSync:(RCChatRoomSyncEvent *)event;	在用户在多端登录情况下，在其他客户端加入、退出聊天室	接收在当前客户端上用户加入、退出聊天室的事件
chatRoomNotifyBlock:(RCChatRoomMemberBlockEvent *)event;	用户所在聊天室中发生了禁言、解除禁言相关的事件	是否接收通知取决于调用服务端封禁、解封的相关 API 时是否指定了 needNotify 为 true。
chatRoomNotifyBan:(RCChatRoomMemberBanEvent *)event;	用户所在聊天室发生了封禁用户、解除封禁相关的事件	是否接收通知取决于调用服务端封禁、解封的相关 API 时是否指定了 needNotify 为 true。

多端登录事件通知

在用户在多端登录情况下，在其他客户端加入、退出聊天室时，触发 chatRoomNotifyMultiLoginSync:(RCChatRoomSyncEvent *)event 方法。

触发场景	通知范围	说明
用户多端登录，在一台设备上加入聊天室	当前加入的用户	当前用户在一端加入聊天室时，其他端的在线设备上会接收通知，不在线的设备不会通知。返回数据包括：聊天室 ID、加入时间。
用户多端登录，在一台设备上退出聊天室	当前退出的用户	当前用户在一端退出聊天室时，其他端的在线设备上会接收通知，不在线的设备不会通知。 返回数据包括：聊天室 ID、退出时间。
用户多端登录且已在聊天室中，加入一个新的聊天室导致被踢出上一个聊天室	当前用户，和被踢出的聊天室所有成员	如果在控制台开通了单个用户加入多个聊天室，则不会通知。返回数据包括：聊天室 ID、被踢出的时间。

RCChatRoomSyncEvent 具体定义如下：

```
@interface RCChatRoomSyncEvent : NSObject
/*!
聊天室 ID
*/
@property (nonatomic, copy) NSString *chatroomId;

/**
同步通知的变更状态
*/
@property (nonatomic, assign) RCChatRoomSyncStatus status;

/**
如果status是0的情况，区分离开类型：
1，自己主动离开，
2，多端加入互踢导致离开
*/
@property (nonatomic, assign) NSInteger reason;

/**
同步通知的变更时间
用户加入/退出/被踢的时间(毫秒时间戳)
*/
@property (nonatomic, assign) long long time;

/**
附加信息
*/
@property (nonatomic, copy, nullable) NSString *extra;

@end
```

封禁相关事件通知

用户所在聊天室发生了封禁、解封的事件，且调用相关 Server API 时指定了需要通知（指定 needNotify 为 true），触发 chatRoomNotifyBlock:(RCChatRoomMemberBlockEvent *)event 方法。

触发场景	通知范围	说明
封禁聊天室用户	聊天室中所有成员	返回数据包括：聊天室 ID、被封禁成员用户 ID 列表、封禁时间与持续时长、附加信息。
解除封禁聊天室用户	被解除封禁的成员	返回数据包括：聊天室 ID、当前用户 ID、附加信息。

RCChatRoomMemberBlockEvent 具体定义如下：

```
@interface RCChatRoomMemberBlockEvent : NSObject
/*!
聊天室 ID
*/
@property (nonatomic, copy) NSString *chatroomId;

/**
封禁类型，0是解封，1是封禁
*/
@property (nonatomic, assign) RCChatRoomMemberOperateType operateType;

/**
封禁的总时间，封禁时有此字段(毫秒)，最大值为43200分钟（1个月），最小值1分钟
*/
@property (nonatomic, assign) NSInteger durationTime;

/**
操作时间(毫秒时间戳)
*/
@property (nonatomic, assign) long long operateTime;

/**
被封禁用户 ID 列表；解封时为当前用户 ID
*/
@property (nonatomic, copy) NSArray<NSString *> *userIdList;

/**
附加信息
*/
@property (nonatomic, copy, nullable) NSString *extra;

@end
```

禁言相关事件通知

用户所在聊天室发生了禁言、解除禁言相关的事件，且调用相关 Server API 时指定了需要通知（指定 needNotify 为 true），触发 chatRoomNotifyBan:(RCChatRoomMemberBanEvent *)event 方法。

触发场景	通知范围	说明
禁言指定聊天室用户	聊天室中所有成员	返回数据包括：聊天室 ID、被封禁成员用户 ID 列表、禁言时间与持续时长、附加信息。
取消禁言指定聊天室用户	聊天室中所有成员	返回数据包括：聊天室 ID、被封禁成员用户 ID 列表、解除禁言时间、附加信息。
设置聊天室全体禁言	聊天室中所有成员	返回数据包括：聊天室 ID、禁言时间、附加信息。
取消聊天室全体禁言	聊天室中所有成员	返回数据包括：聊天室 ID、解除禁言时间、附加信息。
加入聊天室全体禁言白名单	聊天室中所有成员	返回数据包括：聊天室 ID、被添加白名单的用户 ID 列表、设置白名单时间、附加信息。
移出聊天室全体禁言白名单	聊天室中所有成员	返回数据包括：聊天室 ID、被移出白名单的用户 ID 列表、移出白名单时间、附加信息。
全局禁言用户	被全局禁言的用户	返回数据包括：聊天室 ID、禁言时间与持续时长、附加信息。
取消全局禁言用户	被解除全局禁言的用户	返回数据包括：聊天室 ID、解除禁言时间、附加信息。

RCChatRoomMemberBanEvent 具体定义如下：

```
@interface RCChatRoomMemberBanEvent : NSObject
/*!
聊天室 ID
*/
@property (nonatomic, copy) NSString *chatroomId;

/**
禁言操作类型，详见枚举值
禁言/解除禁言操作类型：
0：解除指定聊天室中用户禁言；
1：禁言指定聊天室中用户；
2：解除聊天室全体禁言；
3：聊天室全体禁言
4: 移出禁言用户白名单
5: 添加禁言用户白名单
6: 解除用户聊天室全局禁言
7：用户聊天室全局禁言
*/
@property (nonatomic, assign) RCChatRoomMemberBanType banType;

/**
禁言的总时间（禁言的操作有此字段）（毫秒），最大值为43200分钟（1个月），最小值1分钟
*/
@property (nonatomic, assign) NSInteger durationTime;

/**
操作时间(毫秒时间戳)
*/
@property (nonatomic, assign) long long operateTime;

/**
禁言/解禁言用户 ID 列表
*/
@property (nonatomic, copy) NSArray<NSString *> *userIdList;

/**
附加信息
*/
@property (nonatomic, copy, nullable) NSString *extra;

@end
```

设置聊天室事件通知代理委托

SDK 从 5.4.5 版本开始支持 RCChatRoomNotifyEventDelegate 委托协议。以下代码示例显示了如何设置聊天室事件通知代理委托。

```
[[RCChatRoomClient sharedChatRoomClient] addChatRoomNotifyEventDelegate:self];
```

删除聊天室事件通知代理委托

SDK 从 5.4.5 版本开始支持 RCChatRoomNotifyEventDelegate 委托协议。以下代码示例显示了如何删除聊天室事件通知代理委托。

```
[[RCChatRoomClient sharedChatRoomClient] removeChatRoomNotifyEventDelegate:self];
```

获取聊天室历史消息 获取聊天室历史消息

更新时间:2024-08-30

客户端 SDK 支持获取聊天室历史消息。具体能力如下：

- 通过 getHistoryMessages 获取聊天室保存在本地的历史消息。请注意，聊天室业务默认在用户退出聊天室时会清除聊天室保存在本地的历史消息。
- 通过 getRemoteChatroomHistoryMessages 获取聊天室保存在服务端的历史消息。请注意，该功能依赖聊天室消息云端存储服务。

开通服务

使用 getRemoteChatroomHistoryMessages 要求开通 聊天室消息云端存储服务。使用前请确认已开通服务。开通后聊天室历史消息保存在云端，默认保存 2 个月。

获取聊天室远端历史消息

您可以通过 getRemoteChatroomHistoryMessages 获取聊天室远端历史记录。如果本地数据库存在相同时段内的历史消息，那么调用这个接口会返回一个 size 为 0 的数组。因此请先调用 getHistoryMessages 获取本地的历史记录，如果为空的话，再调用 getRemoteChatroomHistoryMessages 来获取远端历史记录。

```
NSArray *history = [[RCIMClient sharedRCIMClient]
getHistoryMessages:ConversationType_CHATROOM
targetId:@"chatroomId"
oldestMessageId:lastMessageID
count:count];
if (history.count == 0) {
[[RCIMClient sharedRCIMClient]
getRemoteChatroomHistoryMessages:@"chatroomId"
recordTime:recordTime
count:50
order:RC_Timestamp_Desc
success:^(NSArray *messages, long long syncTime) {

} error:^(RCErrorCode status) {

}];
}
```

获取成功的 successBlock 中会返回获取到的历史消息数组和 syncTime。 如果拉取顺序为 RC_Timestamp_Desc，为拉取结果中最早一条消息的时间戳（即最小的时间戳）。如果拉取顺序为 RC_Timestamp_Asc，为拉取结果中最晚消息的时间戳（即最大的时间戳）。在拉取顺序不变的情况下，当次返回的 syncTime 的值可以作为下次拉取时的 recordTime 传入，方便连续拉取。

参数	类型	说明
targetId	NSString	聊天室 ID，最大长度为 64 个字符。
recordTime	long long	消息发送时间戳（毫秒），表示获取 recordTime 之前或者之后的消息。
count	int	需要获取的消息数量，最大不超过 200 条。
order	RCTimestampOrder	拉取顺序。RC_Timestamp_Desc：降序，按消息发送时间递减的顺序，获取发送时间早于 recordTime 的消息。RC_Timestamp_Asc： 升序，按消息发送时间递增的顺序，获取发送时间晚于 recordTime 的消息。默认值为降序。
successBlock	Block	获取成功的回调。
errorBlock	Block	获取失败的回调。status 中返回错误码 RCErrorCode

聊天室属性管理（KV） 聊天室属性管理（KV）

更新时间:2024-08-30

SDK 在聊天室业务核心类 RCChatRoomClient 中提供了聊天室属性（KV）管理接口（也可以使用 RCIMClient），用于在指定聊天室中设置自定义属性。

在语音直播聊天室场景中，可利用此功能记录聊天室中各麦位的属性；或在狼人杀等卡牌类游戏场景中记录用户的角色和牌局状态等。

开通服务

使用聊天室属性（KV）接口要求开通聊天室属性自定义设置服务。您可以前往控制台的免费基础功能页面开启服务。

聊天室业务还提供服务端回调功能，支持由融云服务端将应用下的全部聊天室属性变化（设置，删除，全部删除等操作）同步到您指定的地址，方便 App 业务服务端了解聊天室属性变化。详见服务端文档聊天室属性同步（KV）。

功能局限

提示

- 聊天室销毁后，聊天室中的自定义属性同时销毁。
- 每个聊天室中，最多允许设置 **100** 个属性信息，以 `Key-Value` 的方式进行存储。
- 客户端 SDK 未针对聊天室属性 KV 的操作频率进行限制。建议每个聊天室，每秒钟操作 `Key-Value` 频率保持在 **100** 次及以下（一秒内单次操作 100 个 KV 等同于操作 100 次）。

监听聊天室属性变化

SDK 在聊天室业务核心类 RCChatRoomClient 中提供了 RCChatRoomKVStatusChangeDelegate 协议，用于监听聊天室属性（KV）变化。

实现此功能需要开发者遵守 RCChatRoomKVStatusChangeDelegate 协议。

```
@protocol RCChatRoomKVStatusChangeDelegate <NSObject>

/**
IMLib 刚加入聊天室时 KV 同步完成的回调

@param roomId 聊天室 Id
*/
- (void)chatRoomKVDidSync:(NSString *)roomId;

/**
IMLib 聊天室 KV 变化的回调

@param roomId 聊天室 Id
@param entry KV 字典，如果刚进入聊天室时存在 KV，会通过此回调将所有 KV 返回，再次回调时为其他人设置或者修改 KV
*/
- (void)chatRoomKVDidUpdate:(NSString *)roomId entry:(NSDictionary<NSString *, NSString *> *)entry;

/**
IMLib 聊天室 KV 被删除的回调

@param roomId 聊天室 Id
@param entry KV 字典
*/
- (void)chatRoomKVDidRemove:(NSString *)roomId entry:(NSDictionary<NSString *, NSString *> *)entry;

@end
```

设置聊天室 KV 监听器

设置聊天室 KV 监听器代理。为了避免内存泄露，请在不需要监听时将监听器移除。

使用 addChatRoomKVStatusChangeDelegate 方法，支持设置多个监听器：

```
[[RCChatRoomClient sharedChatRoomClient] addChatRoomKVStatusChangeDelegate:self];
```

移除聊天室 KV 监听器

SDK 支持移除监听器。为了避免内存泄露，请在不需要监听时将监听器移除。

使用 removeChatRoomKVStatusChangeDelegate：

```
[[RCChatRoomClient sharedChatRoomClient] removeChatRoomKVStatusChangeDelegate:self];
```

获取单个属性

通过属性的 Key 获取指定聊天室中的单个属性 KV 对。

```
[[RCChatRoomClient sharedChatRoomClient] getChatRoomEntry:chatroomId
key:key
success:^(NSDictionary *entry) {

} error:^(RCErrorCode nErrorCode) {

}];
```

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符
key	NSString	聊天室属性名称
successBlock	Block	成功的回调。entry 中返回 NSDictionary 类型的属性内容。
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode 。

获取所有属性

获取指定聊天室中所有属性 KV 对。

```
[[RCChatRoomClient sharedChatRoomClient] getAllChatRoomEntries:self.roomId
success:^(NSDictionary *entry) {

} error:^(RCErrorCode nErrorCode) {

}];
```

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符。
successBlock	Block	成功的回调。entry 中返回 NSDictionary 类型的属性内容。
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode 。

设置单个属性

设置聊天室自定义属性，当 key 不存在时，代表增加属性； 当 key 已经存在时，代表更新属性的值，且只有 key 的创建者可以更新属性的值。

```
[[RCChatRoomClient sharedChatRoomClient] setChatRoomEntry:chatroomId
key:key
value:value
sendNotification:isNotice
autoDelete:isDelete
notificationExtra:extra
success:^{

} error:^(RCErrorCode nErrorCode) {

}];
```

在设置 KV 时，可指定 SDK 发送一条聊天室属性通知消息（RCChatroomKVNotificationMessage）。消息内容结构说明可参见通知类消息格式。

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符。
key	NSString	聊天室属性名称，Key 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式，最大长度 128 个字符
value	NSString	聊天室属性对应的值，最大长度 4096 个字符
sendNotification	BOOL	是否需要发送通知
autoDelete	BOOL	用户掉线或退出时，是否自动删除该 Key、Value 值；自动删除时不会发送通知
notificationExtra	NSString	通知的自定义字段，聊天室属性通知消息 RCChatroomKVNotificationMessage 中会包含此字段，最大长度 2048 个字符
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode。

强制设置单个属性

强制设置聊天室自定义属性，当 key 不存在时，代表增加属性； 当 key 已经存在时，代表更新属性的值。

```
[[RCChatRoomClient sharedChatRoomClient] forceSetChatRoomEntry:chatroomId
key:key
value:value
sendNotification:isNotice
autoDelete:isDelete
notificationExtra:extra
success:^{

} error:^(RCErrorCode nErrorCode) {

}];
```

在设置 KV 时，可指定 SDK 发送一条聊天室属性通知消息（RCChatroomKVNotificationMessage）。消息内容结构说明可参见通知类消息格式。

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符。
key	NSString	聊天室属性名称，Key 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式，最大长度 128 个字符
value	NSString	聊天室属性对应的值，最大长度 4096 个字符
sendNotification	BOOL	是否需要发送通知
autoDelete	BOOL	用户掉线或退出时，是否自动删除该 Key、Value 值；自动删除时不会发送通知
notificationExtra	NSString	通知的自定义字段，聊天室属性通知消息 RCChatroomKVNotificationMessage 中会包含此字段，最大长度 2048 个字符
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode。

批量设置属性

> 提示
>
> IMLib 从 5.1.4 版本开始支持该接口。

批量设置聊天室属性。通过 isForce 参数可控制是否强制覆盖他人设置的属性。

```
NSDictionary *entries = @{@"key1":@"value1", @"key2":@"value2"};
[[RCChatRoomClient sharedChatRoomClient] setChatRoomEntries:chatroomId
entries:entries
isForce:isForce
autoDelete:isDelete
success:^{

} error:^(RCErrorCode nErrorCode) {

}];
```

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符。
entries	NSDictionary	聊天室属性，字典中 key 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式，最大长度 128 个字符，value 聊天室属性对应的值，最大长度 4096 个字符，最多一次设置 10 条
isForce	BOOL	是否强制覆盖
autoDelete	BOOL	用户掉线或退出时，是否自动删除该 Key、Value 值；自动删除时不会发送通知
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode 。

删除单个属性

删除指定单个聊天室的单个属性。该接口仅支持删除当前用户所创建的属性。

```
[[RCChatRoomClient sharedChatRoomClient] removeChatRoomEntry:chatroomId
key:key
sendNotification:isNotice
notificationExtra:extra
success:^{

} error:^(RCErrorCode nErrorCode) {

}];
```

在删除 KV 时，可指定 SDK 发送一条聊天室属性通知消息（RCChatroomKVNotificationMessage ）。消息内容结构说明可参见通知类消息格式 。

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符。
key	NSString	聊天室属性名称，Key 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式，最大长度 128 个字符
sendNotification	BOOL	是否需要发送通知
notificationExtra	NSString	通知的自定义字段，聊天室属性通知消息 RCChatroomKVNotificationMessage 中会包含此字段，最大长度 2048 个字符
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode 。

强制删除单个属性

强制删除聊天室自定义属性。可删除他人所创建的属性。

```
[[RCChatRoomClient sharedChatRoomClient] forceRemoveChatRoomEntry:chatroomId
key:key
sendNotification:isNotice
notificationExtra:extra
success:^{}
error:^(RCErrorCode nErrorCode) {}];
```

在删除 KV 时，可指定 SDK 发送一条聊天室属性通知消息（RCChatroomKVNotificationMessage ）。消息内容结构说明可参见通知类消息格式 。

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符。
key	NSString	聊天室属性名称，Key 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式，最大长度 128 个字符
sendNotification	BOOL	是否需要发送通知
notificationExtra	NSString	通知的自定义字段，聊天室属性通知消息 RCChatroomKVNotificationMessage 中会包含此字段，最大长度 2048 个字符
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode 。

批量删除属性

> 提示
>
> IMLib 从 5.1.4 版本开始支持该接口。

从指定单个聊天室中批量删除属性。单次最多删除 10 个属性。通过 isForce 参数可控制是否强制删除他人设置的属性。

```
NSArray *keys = @[@"key1", @"key2"];
[[RCChatRoomClient sharedChatRoomClient] removeChatRoomEntries:chatroomId
keys:keys
isForce:isForce
success:^{

} error:^(RCErrorCode nErrorCode) {

}];
```

参数	类型	说明
chatroomId	NSString	聊天室 ID，最大长度为 64 个字符。
keys	NSDictionary	聊天室属性 keys, 最多一次删除 10 条
isForce	BOOL	是否强制覆盖
successBlock	Block	成功的回调
errorBlock	Block	失败的回调。status 中返回错误码 RCErrorCode 。

绑定音视频房间 绑定音视频房间

更新时间:2024-08-30

聊天室与音视频房间绑定成功后，只要音视频房间仍存在，则阻止聊天室自动销毁。

适用场景：

聊天室具有自动销毁机制。在使用融云 RTC 业务的 App 中，可能配合使用 IM SDK 的聊天室业务实现直播聊天、弹幕、属性记录等功能。这种情况下，可以考虑将聊天室与音视频房间绑定，确保聊天室不会在语聊、直播结束前销毁，以免丢失关键数据。

在单独使用聊天室业务情况的下，无需调用该接口。

> 提示
>
> **关于聊天室自动销毁逻辑的说明：**
>
> 聊天室具有自动销毁机制，默认情况下所有聊天室会在不活跃（连续时间段内无成员进出且无新消息）达到 1 小时后踢出所有成员并自动销毁，可延长该时间，也可配置为定时自动销毁。详见服务端文档聊天室销毁机制。

绑定音视频房间

> 提示
>
> - 客户端 SDK 5.2.1 版本开始支持绑定音视频房间接口。
> - 该接口在 `RCChatRoomClient` 中。

绑定音视频房间后，当聊天室达到预设的自动销毁条件时，服务端会先检测已绑定的音视频房间（RTCRoomId）是否仍存在：

- 如果绑定的音视频房间仍存在，则聊天室不会销毁。
- 如果绑定的音视频房间已销毁，则直接销毁聊天室。

该接口仅创建从聊天室到音视频房间的单向绑定关系。因此在绑定音视频房间后，音视频房间的主动销毁或自动销毁，并不会直接触发聊天室房间的销毁。关于音视频房间的销毁机制，详见音视频房间销毁机制。

接口说明

调用该接口必须传入聊天室 ID，因此必须在聊天室房间已创建成功之后调用。客户端调用加入聊天室接口时会自动完成创建与加入动作。

该接口不具备用户权限控制功能，建议由业务侧的房主或者主播角色用户加入聊天室房间成功后调用一次，其他用户无需调用。

```
- (void)bindChatRoom:(NSString *)chatRoomId withRTCRoom:(NSString *)rtcRoomId success:(void(^)(void))successBlock
error:(void (^)(RCErrorCode nErrorCode))errorBlock
```

参数说明

参数	类型	说明
chatRoomId	NSString	聊天室 ID，非空，聊天室 ID 必须已存在。需要 App 传入，最大长度为 64 个字符。
rtcRoomId	NSString	音视频房间 ID，非空。需要 App 传入。
successBlock	Block	成功回调

参数	类型	说明
errorBlock	Block	失败回调

退出聊天室 退出聊天室

更新时间:2024-08-30

退出聊天室支持以下几种情况：

- **被动退出聊天室**：聊天室具有离线成员自动踢出机制。该机制被触发时，融云服务端会将用户踢出聊天室。用户如被封禁，也会被踢出聊天室。
- **主动退出聊天室**：客户端提供 API，支持由用户主动退出聊天室。

聊天室离线成员自动退出机制

聊天室具有离线成员自动退出机制。用户离线后，如满足以下默认预设条件，融云服务端会自动将该用户踢出聊天室：

- 从用户离线开始 30 秒内，聊天室中产生第 31 条消息时，触发自动踢出。
- 或用户已离线 30 秒后，聊天室有新消息产生时，触发自动踢出。

> 提示
>
> - 默认预设条件均要求聊天室中必须要有新消息产生，否则无法触发踢出动作。如果聊天室中没有消息产生，则无法将异常用户踢出聊天室。
> - 如需修改默认行为对新消息的依赖，请提交工单申请开通聊天室成员异常掉线实时踢出。开通该服务后，服务端会通过 SDK 行为（要求 Android/iOS IMLib SDK 版本 ≧ 5.1.6，Web IMLib 版本 ≧ 5.3.2）判断用户是否处于异常状态，最迟 5 分钟可以将异常用户踢出聊天室。
> - 如需保护特定用户，即不自动踢出指定用户（如某些应用场景下可能希望用户驻留聊天室），可使用 Server API 提供的 聊天室用户白名单功能。

主动退出聊天室

客户端用户可主动退出聊天室。

调用示例

```
[[RCIMClient sharedRCIMClient] quitChatRoom:@"chatroomId"
success:^{

} error:^(RCErrorCode status) {

}];
```

输入参数

参数	类型	说明
targetId	NSString	聊天室 ID，最大长度为 64 个字符。
successBlock	Block	退出聊天室成功的回调
errorBlock	Block	退出聊天室失败的回调

返回参数

`errorBlock` 说明:

回调参数	回调类型	说明
status	RCErrorCode	退出聊天室失败的错误码

超级群概述 超级群概述

更新时间:2024-08-30

- 客户端 IMLib SDK 从 SDK 5.2.0 开始支持超级群。IMKit 暂不支持超级群业务。
- 除部分接口另有说明外，超级群 Android 客户端接口大部分均在 `RCChannelClient` 中。

融云超级群（UltraGroup）提供了一种新的群组业务形态。超级群不设置群成员人数上限，允许用户在超级社群中建立社交关系、在海量信息中聚焦自己感兴趣的内容，帮助开发者打造高用户黏性的群体。超级群组成员最多可加入 100 个超级群，每个超级群下的不同频道之间共享一份超级群成员关系。App 内的超级群数量没有限制。

超级群业务的会话类型（ConversationType）为 ConversationType_ULTRAGROUP，用 targetId 表示超级群 ID，channelId 表示超级群频道 ID。除部分接口另有说明外，超级群 Android 客户端接口大部分均在 RCChannelClient 中。

开通服务

超级群功能需要在控制台超级群服务 页面开通。仅 **IM 尊享版**支持开通超级群服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。

如何使用频道

超级群支持在群会话中创建独立的频道（客户端由 channelId 指定、对应服务端的 busChannel），超级群的会话、消息、未读数等消息数据和群组成员支持分频道进行聚合，各个频道之间消息独立。

频道按类型区分为公有频道与私有频道。公有频道对所有超级群成员开放（无需加入）。该超级群的所有成员都会接收公有频道下的消息。私有频道仅对该频道成员列表上的用户开放。有关私有频道的详细介绍，可参见超级群私有频道概述。

超级群业务提供一个 ID 为 RCDefault 的默认频道。RCDefault 频道对所有超级群成员开放，不可转为私有频道。

对于 App 业务来说，如果仅需实现类似群聊的业务，可以利用超级群无成员上限的特性构建大于 3000 人的超大群。这种场景下，可以让所有消息都在 RCDefault 默认频道中进行收发。建议在调用客户端、服务端 API 时指定频道 ID 为 RCDefault。

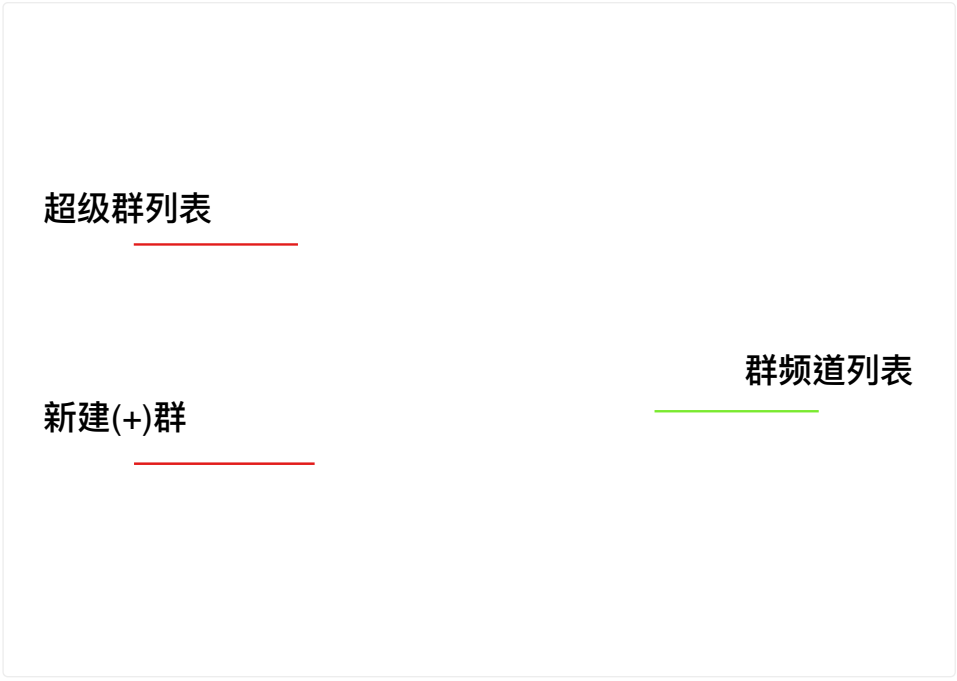
如果仅需实现类似 Discord 类业务，通过超级群频道功能构建子社区，推荐全部使用您自行创建的频道实现您的业务特性。默认频道（RCDefault）与自建频道的行为存在差异，全部使用自建频道可避免这种差异在实现 App 业务逻辑时造成限制。

> 提示
>
> 如果您的 App / 环境在 2022.10.13 日之前开通超级群服务，则您的超级群服务中不存在 RCDefault 频道。在调用客户端、服务端 API 时如果不指定频道 ID，一般仅作用于不属于任何频道的消息，具体行为需参见各功能文档。融云支持客户调整服务至最新行为。该行为调整将影响客户端、服务端收发消息、获取会话、清除历史消息、禁言等多个功能。如有需要，请提交工单咨询详细方案。

客户端 UI 框架参考设计

超级群产品暂不提供 UI 组件。您可以参考以下 UI 框架设计了解超级群 App 的设计思路。

- 下图左侧红框中为超级群列表，即当前登录用户的超级群列表。红框底部的加号（+）按钮代表新建超级群。
- 上图绿框中为超级群频道（Channel）列表。超级群的每一个频道由频道 ID（`channelId`）指定。



超级群管理接口

融云不会托管用户，也不管理群组的业务逻辑，因此超级群的业务逻辑全部需要在 App 服务器进行实现。

对于客户端开发人员来说，创建群组、频道等基础管理操作只需要与 App 服务端交互即可。App 服务端需要调用相应的融云服务端 API（Server API）接口相关接口创建超级群、创建频道等其他管理操作。

下表列出了融云提供的超级群基础管理接口。

> 提示
>
> Server API 还提供超级群全体禁言、超级群用户禁言等更多管理接口。具体请参见 融云服务端超级群文档。

功能分类	功能描述	客户端 API	融云服务端 API
创建、解散超级群	提供创建者用户 ID、超级群 ID、和群名称，向融云服务端申请建群。如解散超级群，则群成员关系不复存在。	不提供该 API	创建超级群、解散超级群
加入、退出超级群	加入超级群后，默认可查看入群以后产生的新消息。退出超级群后，不再接受该群的新消息。	不提供该 API	加入超级群、退出超级群
修改融云服务端的超级群信息	修改在融云推送服务中使用的超级群信息。	不提供该 API	更新超级群信息
创建、删除群频道	在超级群会话中创建独立沟通的频道。如删除频道，将无法在频道中发送消息。	不提供该 API	创建频道、删除频道
查询群频道列表	加入超级群后，默认可查看入群以后产生的新消息。退出超级群后，不再接受该群的新消息。	不提供该 API	查询频道列表
变更群频道类型	超级群频道可以随时切换为公有频道或私有频道。	不提供该 API	变更频道类型
添加、删除私有频道成员	将超级群成员加入或移出指定频道的私有频道成员列表。在频道类型为私有频道时启用该成员列表的数据。	不提供该 API	添加私有频道成员、删除私有频道成员

私有频道概述 私有频道概述

更新时间:2024-08-30

超级群服务支持创建私有频道，满足社区场景中只有指定用户可以在频道中沟通的业务需求。

在使用私有频道功能前，建议先阅读超级群概述，了解在 App 业务中如何使用频道。

已知限制

> 提示
>
> **私有频道功能目前有以下限制：**
>
> **用户即使不在私有频道成员列表中，发送消息会失败，但消息仍会进入本地数据库。如果用户未在私有频道成员列表中，但仍往频道中发送消息，此时消息无法成功发送到服务端，但会进入本地数据库，并可生成会话。**
>
> **建议解决方案：开发者在 App 业务侧维护一份私有频道成员列表数据，如果用户未在私有频道成员列表中，禁止用户往私有频道发送消息。**

了解私有频道

> 提示
>
> 客户端不提供创建私有频道、变更频道类型的 API。请通过服务端 API 实现。

超级群的频道按类型区分为公有频道与私有频道。公有频道对所有超级群成员开放（无需加入）。该超级群的所有成员都会接收公有频道下的消息。私有频道仅对该频道的成员列表开放，仅频道成员可在该频道中收发消息、接收频道状态变化的通知。私有频道可以随时切换为公有频道，公有频道也可以切换为私有频道。

例如，管理员在社区中新创建一个频道，并定义为私有频道。接着邀请社区中部分成员加入私有频道。只有加入频道的成员才可以浏览此频道，并在频道中接收、发送消息。

频道变更会影响该频道下服务端历史消息的拉取权限，具体如下：

- 公有频道转换为私有频道：仅当前私有频道成员列表中的用户可以拉取该频道下的历史消息（含原公有频道消息）；
- 私有频道转换为公有频道：所有超级群用户均可拉取该频道下的历史消息（含原私有频道消息）。

您可以通过以下方式管理私有频道：

功能描述	客户端 API	融云服务端 API
创建私有频道	不提供该 API	创建频道
删除私有频道	不提供该 API	删除频道
查询频道列表	不提供该 API	查询频道列表
变更频道类型	不提供该 API	变更频道类型

了解私有频道成员列表

> 提示
>
> 客户端 SDK 不提供管理私有频道成员列表的 API。您可以调用服务端 API 查询私有频道成员列表、或将超级群成员加入、移出

成员列表。

频道类型为私有频道时，只有该列表中用户可在频道中的收发消息、接收频道内状态通知。

私有频道转为公有频道时，频道变更为对所有超级群成员开放。但该列表数据不会被删除。假设频道类型再次变更为私有，则启用该成员列表。

您也可以为公有频道创建私有频道成员列表。一旦该公有频道变更类型为私有频道，该列表即生效。

提示

- 删除频道时，服务端会清除该频道的私有频道成员列表。
- 一旦超级群成员退群，服务端会自动将用户从私有频道成员列表移除。

您可以通过以下方式管理私有频道成员列表：

功能描述	客户端 API	融云服务端 API
加入私有频道成员列表	不提供该 API	添加私有频道成员
移出私有频道成员列表	不提供该 API	删除私有频道成员
查询私有频道成员列表	不提供该 API	查询私有频道成员列表

用户组概述 用户组概述

更新时间:2024-08-30

超级群用户组（User Group）功能是融云超级群业务提供的群成员管理工具，结合超级群私有频道功能，可以帮助 App 实现更高效的超级群成员管理，沟通管理，和更精细的用户通知能力。

前提条件

> 提示
>
> 客户端 SDK 从 5.4.0 版本开始支持超级群用户组功能。

超级群用户组功能主要通过与私有频道的绑定操作，提供了对用户在社区私有频道中沟通权限（收发消息、通知）的批量管理能力，提升了 App 集成效率与对超级群的运营管理能力。

在了解与使用超级群用户组功能前，需要先了解超级群私有频道功能。请参见超级群私有频道概述。

如何使用超级群用户组

App 服务端可以通过调用融云服务端 API，在超级群中创建最多 50 个用户组（userGroup），每个用户组成员最多由 100 个超级群成员组成。单个用户可以存在于多个用户组中。

用户组创建后，可以与超级群频道绑定。如果用户组绑定了一个或多个私有频道，该用户组的所有成员即具有在绑定的私有频道中收发消息、接收通知的能力。

- App 将用户组绑定私有频道后，可认为组中所有用户均加入了该私有频道。与该私有频道成员列表中的用户类似，只有加入频道的成员才可以浏览此频道，并在频道中接收消息，发送消息，和接收通知。
- App 将用户组绑定公有频道后，不会影响组中用户可收发消息的范围。但一旦该公有频道转为私有频道，该组用户将具有在该私有频道中收发消息、获取通知的能力。
- App 可以在一个频道上绑定最多 10 个用户组，一个用户组可以与多个频道绑定（一个超级群最多 50 个频道）。

> 提示
>
> 超级群业务默认提供 RCDefault 频道，对所有超级群成员开放，不可转为私有频道。建议不要将用户组绑定到 RCDefault 频道。

App 客户端无法进行用户组管理操作，仅可通过 SDK 提供的回调方法监听用户组相关变更的通知，具体包括：

- **删除用户组**：用户组下所有用户均可收到客户端回调
- **用户组成员变更**：被加入或移出用户组的用户可收到客户端回调
- **频道与用户组绑定关系变更**：用户组下所有用户均可收到客户端回调

混合使用用户组与私有频道成员列表

只要 App 用户在指定私有频道绑定的任意一个用户组中，或者在有该私有频道成员列表中，该用户就能在私有频道中收发消息接收通知。

如果混合使用私有频道成员列表与用户组，在 App 业务中可能存在以下情况：

- 私有频道配置了私有成员列表，并添加了多位用户。
- 该私有频道绑定了多个用户组。

在上述使用场景中，某个用户可能既在私有频道成员列表中，又同时在该频道绑定的多个用户组中。如果该用户不再使用该私有频道，App 进行以下操作，确保该用户无法继续在私有频道收发消息：

- 从该私有频道的成员列表中移除该用户。
- 检查该私有频道绑定的所有用户组，从绑定的所有用户组中移除该用户，或解绑用户组。

用户组管理接口

即时通讯（IM）服务端提供了超级群用户组的基础管理接口。用户组的业务逻辑需要 App 服务端自行实现，例如申请加入用户组、审核用户组加入申请等。建议 App 服务端同时维护一份用户、用户组、频道之间对应关系的数据。

对于客户端开发人员来说，创建用户组等基础管理操作只需要与 App 服务端交互即可。App 服务端需要调用相应的融云服务端 API（Server API）接口相关接口创建用户组等其他管理操作。

下表列出了超级群用户组基础管理接口。更多相关接口可参见 IM 服务端文档 API 接口列表 。

功能分类	功能描述	融云服务端 API
创建、删除用户组	在指定超级群下创建用户组。如删除用户组，则用户、用户组、频道之间的关系不复存在。	创建用户组 、删除用户组
查询用户组列表	分页查询指定超级群下的用户组，返回用户组 ID 列表。	查询用户组列表
添加、删除用户	在超级群用户组中添加、删除用户。	添加用户 、移出用户
查询用户所属用户组	分页查询指定单个用户在超级群下所属的用户组列表，返回用户组 ID 列表。	查询用户所属用户组
绑定频道与用户组	将指定单个超级群频道与用户组绑定，可绑定多个用户组，单个频道最多支持与 10 个用户组绑定。	绑定频道与用户组
解绑频道与用户组	将指定单个超级群频道与用户组解除绑定，单次请求可解绑最多 10 个用户组。	解绑频道与用户组
查询频道绑定的用户组	分页查询指定的超级群频道绑定的用户组列表，返回用户组 ID 列表。	查询频道绑定的用户组
查询用户组绑定的频道	分页查询指定单个用户组绑定的超级群频道列表，返回超级群频道 ID 列表。	查询用户组绑定的频道

创建超级群与频道 创建超级群与频道

更新时间:2024-08-30

客户端 SDK 不提供创建超级群与创建群频道的接口。请使用融云服务端 API（Server API）的相关接口创建超级群、群频道，或进行其他管理操作。

> 提示
>
> 单个用户最多可以加入 100 个超级群。单个用户在每个群中最多可以加入或者创建 50 个频道。

本文仅简单介绍创建超级群与创建频道的基本流程。

创建超级群

创建超级群必须使用融云服务端 API（Server API），具体接口使用方法请参见服务端文档创建超级群。

基本流程

1. App 客户端请求 App 服务端（AppServer）创建超级群。
2. App 服务端调用融云 Server API 接口创建超级群，群组 ID 由 App 服务器自行生成
3. 超级群创建成功后，由 App 服务端返回给 App 客户端。

如何处理与展示超级群列表

AppServer 需要保存当前用户的超级群列表，并下发给 App，然后进行展示。

> 考虑到 App 由于自身业务需求，需要知晓当前用户的超级群列表（例如用户所在的超级群有等级权重等排序规则），而融云的超级群列表是通过消息产生的，因而两个列表可能不完全一样，建议由 App 按照自身业务保存用户的超级群列表。

创建群频道

创建超级群必须使用融云服务端 API（Server API），具体接口使用方法请参见服务端文档创建频道。

基本流程

如何处理与展示超级群频道列表

为方便在同一超级群下的不同用户按需看到自己需要的列表（例如用户对特定频道标星需要特别展示，APP 服务则需要按用户记录），APP 服务应采取更为灵活的方式维护超级群的频道列表。

APP 服务端也可以按照需求将超级群分组（如超级群概述中的 UI 框架设计所示）。

获取频道列表 获取频道列表

更新时间:2024-08-30

App 可按需使用客户端 SDK 与融云服务端 API 提供的能力，采取灵活的方式维护超级群的频道列表。

> 提示
>
> - 如果您的应用/环境在 2022.10.13 日及以后开通超级群服务，超级群业务中会包含一个 ID 为 `RCDefault` 的默认频道。如果发消息时不指定频道 ID，则该消息会发送到 `RCDefault` 频道中。在获取 `RCDefault` 频道的历史消息时，需要传入该频道 ID。
> - 如果您的应用/环境在 2022.10.13 日前已开通超级群服务，在发送消息时如果不指定频道 ID，则该消息不属于任何频道。获取历史消息时，如果不传入频道 ID，可获取不属于任何频道的消息。融云支持客户调整服务至最新行为。该行为调整将影响客户端、服务端收发消息、获取会话、清除历史消息、禁言等多个功能。如有需要，请提交工单咨询详细方案。

如何获取超级群全部频道列表

超级群下全部频道的列表可通过融云服务端 API (`/ultragroup/channel/get.json`) 获取。

注意，对单个用户来说，最多可以加入 100 个超级群，在每个超级群中最多可以加入或者创建 50 个频道。

> 提示
>
> 客户端 SDK 会通过频道中收发的消息在本地数据库中生成一个超级群频道列表。该列表仅包含了已在本地产生消息的频道，可能并非该超级群下全部频道的列表。

建议从 App 业务服务端维护超级群的频道列表。App 业务一般需要知晓当前用户的所加入的超级群，以及超级群下的频道列表，才能实现特定的业务功能。假设同一超级群下的不同用户需要展示个性化的频道列表（例如 App 需要在 UI 上展示用户标星的特定频道），则需要为用户保存超级群频道列表。

APP 服务端也可以按照需求将超级群分组（如超级群概述中的 UI 框架设计所示）。

获取本地指定超级群下的频道列表

客户端 SDK 会根据频道中收发的消息在本地数据库中生成对应频道的会话。您可以从本地数据库获取 SDK 生成的频道列表。获取到的频道列表按照时间倒序排列。

```
NSArray *conversationList = [[RCChannelClient sharedChannelManager]
getConversationListForAllChannel:ConversationType_ULTRAGROUP targetId:self.ultraGroup.groupId];
for (RCConversation *conversation in conversationList) {
NSString *totalCountInfo = [NSString stringWithFormat:@"未读数：%d",conversation.unreadMessageCount]
NSString *totalMentionCountInfo = [NSString stringWithFormat:@"未读@总数：%d",conversation.mentionedCount]
/// since 5.4.5
NSString *mentionMeCountInfo = [NSString stringWithFormat:@"未读@我个数：%d",conversation.mentionedMeCount]
}
```

获取本地指定超级群特定类型频道列表

> 提示
>
> SDK 从 5.2.4 开始支持按类型获取频道列表。

客户端 SDK 会根据频道中收发的消息在本地数据库中生成对应频道的会话。您可以从本地数据库获取 SDK 生成的频道列表。获取到的频道列表按照时间倒序排列。

频道类型（channelType）支持超级群公有频道或私有频道。

```
/*!
可以获取本地指定超级群下公有频道或者私有频道的列表
@param targetId 会话ID
@param channelType 频道类型
@param successBlock 获取成功的回调 [会话列表]
@param errorBlock 获取失败的回调 [status:清除失败的错误码]
@remarks 超级群消息操作
*/
- (void)getUltraGroupChannelList:(NSString *)targetId
channelType:(RCUltraGroupChannelType)channelType
success:(nullable void (^)(NSArray<RCConversation *>* list))successBlock
error:(nullable void (^)(RCErrorCode status))errorBlock;
```

- **参数说明：**
  - 频道类型（channelType）

```
typedef NS_ENUM(NSInteger, RCUltraGroupChannelType) {
/*!
超级群公有频道
*/
RCUltraGroupChannelTypePublic = 0,
/*!
超级群私有频道
*/
RCUltraGroupChannelTypePrivate = 1
};
```

监听频道状态变更　　监听频道状态变更

更新时间:2024-08-30

客户端可设置监听，在超级群频道发生类型变更（仅适用于私有频道）、成员变更、频道删除等情况时收到对应通知。

- 由于频道类型、用户是否在私有频道成员列表等差异，通知用户的范围会有差异。
- 客户端 SDK 接收到频道删除（解散）、私有频道成员列表变更通知后，会根据具体变化清理本地数据。

设置频道状态变化通知

1. 需要首先实现 RCUltraGroupChannelDelegate 协议，包括私有频道成员变更通知，频道类型变更通知，频道解散通知。

```
/// 频道类型变更的通知
- (void)ultraGroupChannelTypeDidChanged:(NSArray<RCUltraGroupChannelChangeTypeInfo *> *)infoList;

/// 频道已删除（解散）的通知
- (void)ultraGroupChannelDidDisbanded:(NSArray<RCUltraGroupChannelDisbandedInfo *> *)infoList;

/// 私有频道用户列表变更的通知
- (void)ultraGroupChannelUserDidKicked:(NSArray<RCUltraGroupChannelUserKickedInfo *> *)infoList;
```

2. 通过调用 [[RCChannelClient sharedChannelManager] setUltraGroupChannelDelegate:代理] 方法，添加频道变更代理:

```
- (void)setUltraGroupChannelDelegate:(id<RCUltraGroupChannelDelegate>)delegate;
```

频道类型变更的通知

超级群的频道按类型区分为公有频道与私有频道。超级群频道类型变更仅可通过 App 服务端调用服务端 API 实现。客户端 SDK 不提供接口。

频道类型发生变更时，SDK 通过以下方法通知 App：

```
- (void)ultraGroupChannelTypeDidChanged:(NSArray<RCUltraGroupChannelChangeTypeInfo *> *)infoList;
```

- 参数说明

返回值	返回类型	说明
infoList	RCUltraGroupChannelChangeTypeInfo	频道变更信息

- 频道变更类型说明

枚举值	数值	说明
RCUltraGroupChannelChangeTypePublicToPrivate	2	超级群公有频道变成了私有频道
RCUltraGroupChannelChangeTypePrivateToPublic	3	超级群私有频道变成了公有频道

枚举值	数值	说明
RCUltraGroupChannelChangeTypePublicToPrivateUserNotIn	6	超级群公有频道变成了私有频道，但是当前用户不在该私有频道中

频道类型的变更的通知范围

- **公有频道变私有频道**：公有频道变私有频道时，所有用户都会收到通知。但根据用户是否在私有频道成员列表中，收到的通知有差异：
  - 在私有频道成员列表内的用户，收到的变更类型是 `RCUltraGroupChannelChangeTypePublicToPrivate`。
  - 不在私有频道成员列表的用户，收到的变更类型为 `RCUltraGroupChannelChangeTypePublicToPrivateUserNotIn`。

  如有需要，App 可在公有频道变私有频道前，提前指定用户加入私有频道成员列表。
- **私有频道变公有频道**：私有频道变公有频道时，仅在私有频道成员列表的用户会收到通知，变更类型为 `RCUltraGroupChannelChangeTypePrivateToPublic`。

频道已删除（解散）的通知

超级群频道删除（解散）仅可通过 App 服务端调用服务端 API 实现。客户端 SDK 不提供接口。

删除频道时，SDK 通过以下方法通知 App：

```
- (void)ultraGroupChannelDidDisbanded:(NSArray<RCUltraGroupChannelDisbandedInfo *> *)infoList;
```

- 参数说明

返回值	返回类型	说明
infoList	RCUltraGroupChannelDisbandedInfo	频道变更信息

频道已删除的通知范围

- **删除公有频道的通知**：所有用户会收到通知。
- **删除私有频道的通知**：仅在私有频道成员列表的用户会收到通知。

如何清理本地数据

客户端 SDK 收到通知后会清理会删除用户本地会话，但会保留本地会话的消息。

私有频道成员列表变更的通知

超级群私有频道成员列表仅可通过服务端 API 变更。客户端 SDK 不提供接口。

私有频道成员列表发生变化时，SDK 通过以下方法通知 App：

```
- (void)ultraGroupChannelUserDidKicked:(NSArray<RCUltraGroupChannelUserKickedInfo *> *)infoList;
```

- 参数说明：

返回值	返回类型	说明
infoList	RCUltraGroupChannelUserKickedInfo	频道变更信息

私有频道成员列表变更的通知范围

- 如果频道类型为私有频道，将用户从私有频道成员列表移除时，仅通知被移除的用户
- 如果频道类型为公有频道，将用户从私有频道成员名单移除时，不发送通知。注意，该列表在该公有频道变更类型为私有频道时才会生效。

如何清理本地数据

- **当前登录用户被移出私有频道成员列表**
  - （SDK < 5.4.0）：客户端 SDK 收到通知后会从会话列表中删除本地会话，但会保留本地会话的消息。
  - （SDK ≧ 5.4.0）：客户端 SDK 收到通知后会不从会话列表中删除本地会话，App 可以自行决定是否需要删除会话及会话中的消息。
- **其他情况**：如果当被移出列表的用户非本端登录用户，客户端 SDK 收到通知后不做任何处理。

监听用户组状态变更 监听用户组状态变更

更新时间:2024-08-30

SDK 从 5.4.0 版本开始支持超级群用户组功能。

客户端可设置监听，在超级群用户组发生变更时收到取对应通知。

客户端 SDK 针对以下操作提供回调。回调的通知范围与回调数据有差异具体如下：

- **用户加入用户组**：被加入用户组的用户可收到通知，通知中携带超级群 ID、用户组 ID。
- **用户被移出用户组**：被移出的用户可收到通知，通知中携带超级群 ID、用户组 ID。
- **频道与用户组绑定**：用户组下所有用户均可收到通知，通知中携带超级群 ID、频道 ID、频道类型、用户组 ID。
- **频道与用户组解除绑定**：用户组下所有用户均可收到通知，通知中携带超级群 ID、频道 ID、频道类型、用户组 ID。
- **删除用户组**：用户组下所有用户均可收到通知，通知中携带超级群 ID、用户组 ID。

创建用户组时，SDK 不会收到回调。

监听用户组变更通知

SDK 在 `RCUserGroupStatusDelegate` 协议中提供了与用户组变更相关的回调方法。您可以从返回的 `RCConversationIdentifier` 中获取超级群 ID（`targetId`）数据，从返回的 `RCChannelIdentifier` 中获取超级群 ID（`targetId`）和频道 ID（`channelId`）数据。

```
@protocol RCUserGroupStatusDelegate <NSObject>
/*!
当前用户收到超级群下的用户组中解散通知

@param identifier 会话标识[identifier.type:ConversationType_ULTRAGROUP ]
@param userGroupIds 用户组ID列表

@discussion
@warning
@since
*/
- (void)userGroupDisbandFrom:(RCConversationIdentifier *)identifier
userGroupIds:(NSArray<NSString *> *)userGroupIds;

/*!
当前用户被添加到超级群下的用户组

@param identifier 会话标识[identifier.type:ConversationType_ULTRAGROUP ]
@param userGroupIds 用户组ID列表

@discussion
@warning
@since
*/
- (void)userAddedTo:(RCConversationIdentifier *)identifier
userGroupIds:(NSArray<NSString *> *)userGroupIds;

/*!
当前用户从到超级群下的用户组中被移除

@param identifier 会话标识
@param userGroupIds 用户组ID列表

@discussion
@warning
@since
*/
- (void)userRemovedFrom:(RCConversationIdentifier *)identifier
userGroupIds:(NSArray<NSString *> *)userGroupId;


/*!
频道中绑定用户组回调

@param identifier 频道标识
@param channelType 频道类型
@param userGroupIds 用户组ID列表

@discussion
@warning
@since
*/
- (void)userGroupBindTo:(RCChannelIdentifier *)identifier
channelType:(RCUltraGroupChannelType)channelType
userGroupIds:(NSArray<NSString *> *)userGroupIds;

/*!
频道解绑用户组回调

@param identifier 频道标识
@param channelType 频道类型
@param userGroupIds 用户组ID列表

@discussion
@warning
@since
*/
- (void)userGroupUnbindFrom:(RCChannelIdentifier *)identifier
channelType:(RCUltraGroupChannelType)channelType
userGroupIds:(NSArray<NSString *> *)userGroupIds;
@end
```

App 可通过以下方法添加频道变更代理：

```
[[RCChannelClient sharedChannelManager] setUserGroupStatusDelegate:self];
```

关于更新 UI 的提示

考虑到同一用户既在私有频道成员列表中，又在私有频道绑定的（多个）用户组中，App 可以在收到回调时向 App 自身业务服务端查询当前用户是否仍可继续访问该私有频道，并根据该结果刷新 UI。

对于未在通知范围内的用户，可以在用户进入相应页面时向 App 自身业务服务端查询数据，并决定是否需要刷新 UI。

监听会话同步状态 监听会话同步状态

更新时间:2024-08-30

在每一次连接 IM 服务时，客户端 SDK 都会自动从服务端拉取超级群会话列表与消息。

通过设置超级群会话同步监听器，您可以获取超级群会话列表与会话最后一条消息同步完成的通知，从而进行不同业务处理，例如刷新 UI 界面。

建议在应用生命周期内设置。

设置超级群会话同步监听器

> 提示
>
> 从 SDK 5.2.2 版本开始支持该回调接口。

```
- (void)setUltraGroupConversationDelegate:(id<RCUltraGroupConversationDelegate>)delegate;
```

收发消息 收发消息

更新时间:2024-08-30

本文介绍了如何从客户端发送超级群消息。超级群收发消息需要使用 RCCoreClient 下的方法。

前置条件

建议先阅读超级群概述和超级群私有频道概述，了解在 App 业务中如何使用频道和超级群频道功能特性。

- 通过服务端 API 创建超级群
- 通过服务端 API 创建频道，或使用默认频道 ID `RCDefault`
- 通过服务端 API 将发件人加入超级群
- 如不确定发件人是否在超级群中，请通过服务端 API 查询用户是否为群成员
- 如向超级群私有频道中发送消息，请确认已通过服务端 API 添加私有频道成员

> 提示
>
> 当前频道聊天页面发送与接收消息，需要同时检查超级群 ID 和频道 ID。如果超级群 ID 和频道 ID 和当前频道聊天页面对应，才能在当前频道页面进行展示处理，否则就不处理。如果消息出现在其他聊天页面，一般是因为超级群 ID 或者频道 ID 发生错误。

构造消息对象

在发送消息前，需要构造 RCMessage 对象。消息的 conversationType 字段必须填写超级群业务的会话类型 ConversationType_ULTRAGROUP。消息的 targetId 字段表示超级群 ID，channelId 表示超级群频道 ID。

RCMessage 对象中可包含普通消息内容或媒体消息内容。普通消息内容指 RCMessageContent 的子类，例如文本消息（RCTextMessage）。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"测试超级群消息"];

RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_ULTRAGROUP targetId:@"超级群 id" channelId:@"频道 id"
direction:MessageDirection_SEND content:messageContent];
```

媒体消息内容指 RCMediaMessageContent 的子类，例如图片消息（RCImageMessage）、GIF 消息（RCGIFMessage）等。

```
RCImageMessage *mediaMessageContent = [RCImageMessage messageWithImageURI:@"path/to/image"];
mediaMessageContent.full YES; // 图片消息支持设置以原图发送

RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_ULTRAGROUP targetId:@"超级群 id" channelId:@"频道 id"
direction:MessageDirection_SEND content:mediaMessageContent];
```

> 提示
>
> - 如果您的应用/环境在 2022.10.13 日及以后开通超级群服务，超级群业务中会包含一个 ID 为 `RCDefault` 的默认频道。如果发消息时不指定频道 ID，则该消息会发送到 `RCDefault` 频道中。
> - 如果您的应用/环境在 2022.10.13 日前已开通超级群服务，在发送消息时如果不指定频道 ID，则该消息不属于任何频道。融云支持客户调整服务至最新行为。该行为调整将影响客户端、服务端收发消息、获取会话、清除历史消息、禁言等多个功能。如有需要，请提交工单咨询详细方案。

发送普通消息

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

发送超级群普通消息需要使用 RCCoreClient 中的提供的 sendMessage 方法。

```
[[RCCoreClient sharedCoreClient]
sendMessage:message
pushContent:nil
pushData:nil
attached:^(RCMessage * _Nullable message) {
// 消息发出前已存入数据库的消息实体
}
success:^(RCMessage * _Nonnull successMessage) {
//成功
}
error:^(RCErrorCode nErrorCode, RCMessage * _Nonnull errorMessage) {
//失败
}];
```

sendMessage 中直接提供了用于控制推送通知内容（pushContent）和推送附加信息（pushData）的参数。不过，如果您需要更精细地控制离线推送通知，例如标题、内容、图标、或其他第三方厂商个性化配置，请使用消息推送属性进行配置，详见自定义消息推送通知。

- 如果发送的消息属于 SDK 内置消息类型，例如 RCTextMessage，这两个参数可设置为 `nil`。一旦消息触发离线推送通知时，融云服务端会使用各个内置消息类型默认的 `pushContent` 值。关于各类型消息的默认推送通知内容，详见用户内容类消息格式。
- 如果消息类型为自定义消息类型，且需要支持离线推送通知，则必须向融云提供 `pushContent` 字段，否则用户无法收到离线推送通知。
- 如果消息类型为自定义消息类型，但不需要支持远程推送通知（例如通过自定义消息类型实现的 App 业务层操作指令），可将 `pushContent` 字段留空。
- `RCMessage` 的推送属性配置 `MessagePushConfig` 的 `pushContent` 和 `pushData` 会覆盖此处配置，并提供更多配置能力，例如自定义推送通知的标题。详见自定义消息推送通知。

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（conversationType），会话 ID（targetId），消息内容（content）。详见消息介绍 中对 RCMessage 的结构说明。
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。 • 如果希望使用融云默认推送通知内容，可以填 `nil`。注意自定义消息类型无默认值。 • 如果要为该条消息指定个性化的离线推送通知，必须在以上任一处向融云提供 `pushContent` 字段内容。 • 如果您自定义的消息类型需要离线推送服务，必须在以上任一处向融云提供 `pushContent` 字段内容，否则用户无法收到离线推送通知。
pushData	String	远程推送附加信息。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。
attachedBlock	Block	入库回调
successBlock	Block	消息发送成功回调
errorBlock	Block	消息发送失败回调

发送媒体消息

> 提示
>
> 从 5.3.0 版本 RCCoreClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

媒体消息 RCMessage 对象的 content 字段必须传入 RCMediaMessageContent 的子类对象，表示媒体消息内容。例如图片消息内容（RCImageMessage）、GIF 消息内容（RCGIFMessage），或继承自 RCMediaMessageContent 的自定义媒体消息内容。

图片消息内容（RCImageMessage）支持设置为发送原图。

```
RCImageMessage *mediaMessageContent = [RCImageMessage messageWithImageURI:@"path/to/image"];
mediaMessageContent.full YES; // 图片消息支持设置以原图发送

RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_ULTRAGROUP
targetId:@"targetId"
channelId:@"channelId"
direction:MessageDirection_SEND
content:mediaMessageContent];
```

向超级群中发送媒体消息需要使用 sendMediaMessage 方法。SDK 会为图片、小视频等生成缩略图，根据默认压缩配置进行压缩，再将图片、小视频等媒体文件上传到融云默认的文件服务器（文件存储时长），上传成功之后再发送消息。图片消息如已设置为发送原图，则不会进行压缩。

```
[[RCCoreClient sharedCoreClient] sendMediaMessage:message
pushContent:nil
pushData:nil
attached:^(RCMessage * _Nullable message) {
// 消息发出前已存入数据库的消息实体
}
progress:^(int progress, RCMessage *progressMessage) {
//媒体上传进度
}
success:^(RCMessage *successMessage) {
//成功
}
error:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {
//失败
}
cancel:^(RCMessage *cancelMessage) {
//取消
}];
```

sendMediaMessage 中直接提供了用于控制推送通知内容（pushContent）和推送附加信息（pushData）的参数。不过，如果您需要更精细地控制离线推送通知，例如标题、内容、图标、或其他第三方厂商个性化配置，请使用消息推送属性进行配置，详见自定义消息推送通知。

- 如果发送的消息属于 SDK 内置消息类型，例如 RCImageMessage，这两个参数可设置为 `nil`。一旦消息触发离线推送通知时，融云服务端会使用各个内置消息类型默认的 `pushContent` 值。关于各类型消息的默认推送通知内容，详见用户内容类消息格式。
- 如果消息类型为自定义消息类型，且需要支持离线推送通知，则必须向融云提供 `pushContent` 字段，否则用户无法收到离线推送通知。
- 如果消息类型为自定义消息类型，但不需要支持远程推送通知（例如通过自定义消息类型实现的 App 业务层操作指令），可将 `pushContent` 字段留空。
- `RCMessage` 的推送属性配置 `MessagePushConfig` 的 `pushContent` 和 `pushData` 会覆盖此处配置，并提供更多配置能力，例如自定义推送通知的标题。详见自定义消息推送通知。

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（conversationType），会话 ID（targetId），消息内容（content）。详见消息介绍 中对 RCMessage 的结构说明。
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。 • 如果希望使用融云默认推送通知内容，可以填 `nil`。注意自定义消息类型无默认值。 • 如果要为该条消息指定个性化的离线推送通知，必须在以上任一处向融云提供 `pushContent` 字段内容。 • 如果您自定义的消息类型需要离线推送服务，必须在以上任一处向融云提供 `pushContent` 字段内容，否则用户无法收到离线推送通知。
pushData	String	远程推送附加信息。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。

参数	类型	说明
attachedBlock	Block	入库回调
progressBlock	Block	发送进度的回调
successBlock	Block	消息发送成功回调
errorBlock	Block	消息发送失败回调
cancel	Block	取消发送的回调

接收消息

> 提示
>
> 消息接收监听的设置在 RCCoreClient 中

设置监听

```
- (void)addReceiveMessageDelegate:(id<RCIMClientReceiveMessageDelegate>)delegate
```

实现监听代理方法

```
- (void)onReceived:(RCMessage *)message left:(int)nLeft object:(id)object
```

如何发送 @ 消息

@消息在融云即时通讯服务中不属于一种预定义的消息类型（详见服务端文档 消息类型概述 ）。融云通过在消息中携带 mentionedInfo 数据，帮助 App 实现提及他人（@）功能。

消息的 RCMessageContent 中的 RCMentionedInfo 字段存储了携带所 @ 人员的信息。无论是 SDK 内置消息类型，或者您自定义的消息类型，都直接或间接继承了 RCMessageContent 类。

融云支持在向群组和超级群中发消息时，在消息内容中添加 mentionedInfo。消息发送前，您可以构造 MentionedInfo，并设置到消息的 RCMessageContent 中。

```
RCTextMessage *messageContent = [RCTextMessage messageWithContent:@"Test"];

messageContent.mentionedInfo = [[RCMentionedInfo alloc] initWithMentionedType:RC_Mentioned_Users
userIdList:mentionedUserIdList mentionedContent:nil];
```

MentionedInfo 参数：

参数	类型	说明
type	RCMentionedType	（必填）指定 MentionedInfo 的类型。RC_Mentioned_All 表示需要提及（@）所有人。RC_Mentioned_Users 表示需要 @ 部分人，被提及的人员需要在 userIdList 中指定。
userIdList	NSArray	被提及（@）用户的 ID 集合。当 type 为 MentionedType.PART 时必填。
mentionedContent	NSString	触发离线消息推送时，通知栏显示的内容。如果是 nil，则显示默认提示内容（“有人 @ 你”）。@消息携带的 mentionedContent 优先级最高，会覆盖所有默认或自定义的 pushContent 数据。

IMLib SDK 接收消息后，您需要处理 @ 消息中的数据。您可以在获取 RCMessage(消息对象)后获取到消息对象携带的 RCMentionedInfo 对象。

为消息禁用推送通知

在接收者未上线时，融云服务端默认触发推送服务，将消息通过推送通道下发（服务端是否触发推送也会收到应用级别、用户级别免打扰设置的影响）。

有时 App 用户可能希望在发送消息时就指定该条消息不需要触发推送，该需求可通过 RCMessage 对象的 MessageConfig 配置实现。

以下示例中，我们将 messageConfig 的 disableNotification 设置为 YES 禁用该条消息的推送通知。接收方再次上线时需要主动拉取。

```
RCTextMessage *txtMsg = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_ULTRAGROUP
targetId:@"targetId"
channelId:@"targetId"
direction:MessageDirection_SEND
content:txtMsg];

message.messageConfig.disableNotification = YES;
```

自定义消息推送通知

您可以在发送消息时提供 RCMessagePushConfig 配置，对单条消息的推送行为进行个性化配置。例如：

- 自定义推送标题、推送通知内容
- 自定义通知栏图标
- 添加远程推送附加信息
- 越过接收客户端的配置，强制在推送通知内显示通知内容
- 其他 APNs 或 Android 推送通道支持的个性化配置

关于 RCMessagePushConfig 的配置方式详见配置消息的推送属性。

如何处理消息发送失败

对于客户端本地会存储的消息类型（参见消息类型概述），如果触发（attached）回调，表示此时该消息已存入本地数据库，并已进入本地消息列表。

- 如果入库失败，您可以检查参数是否合法，设备存储是否可正常写入。
- 如果入库成功，但消息发送失败（`error`），App 可以在 UI 临时展示这条发送失败的消息，并缓存 `error` 抛出的 `RCMessage` 实例，在合适的时机重新调用 `sendMessage` / `sendMediaMessage` 发送。请注意，如果不复用该消息实例，而是重发相同内容的新消息，本地消息列表中会存储内容重复的两条消息。

对于客户端本地不存储的消息，如果发送失败（error），App 可缓存消息实例再重试发送，或直接新建消息实例进行发送。

发送超级群定向消息 发送超级群定向消息

更新时间:2024-08-30

可发送普通消息与媒体消息给超级群频道中的指定的一个或多个成员，其他成员不会收到该消息。

局限

- 5.6.9 版本开始支持该能力。
- 定向目标用户上限 300 个用户 ID。
- 如果为 @ 消息，不支持 @所有人。

发送定向普通消息

使用 sendDirectionalMessage:toUserIdList:pushContent:pushData:attached:successBlock:errorBlock: 在超级群中发送普通消息给指定频道中的指定用户。不在接收列表的用户不会收到这条消息。

请在消息对象中设置超级群会话类型、超级群 ID 和 频道 ID。频道 ID 为空时默认向 RCDefault 频道发送消息。注意，RCMessage 中不会保存接收用户 ID 列表。

```
RCTextMessage *text = [RCTextMessage messageWithContent:@"你好"];
RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_ULTRAGROUP targetId:@"Group1"
direction:MessageDirection_SEND messageId:-1 content:text];

[[RCCoreClient sharedCoreClient] sendDirectionalMessage:message
toUserIdList:@[@"user1",@"user2"]
pushContent:nil
pushData:nil
successBlock:^(RCMessage *successMessage) {

} errorBlock:^(RCErrorCode nErrorCode, RCMessage *errorMessage) {

}];
```

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（conversationType），频道 ID（channelId），会话 ID（targetId），消息内容（content）。
userIdList	NSArray	将要发送的用户列表
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。 • 如果希望使用融云默认推送通知内容，可以填 nil。注意自定义消息类型无默认值。 • 如果要为该条消息指定个性化的离线推送通知，必须在以上任一处向融云提供 pushContent 字段内容。 • 如果您自定义的消息类型需要离线推送服务，必须在以上任一处向融云提供 pushContent 字段内容，否则用户无法收到离线推送通知。
pushData	String	远程推送附加信息。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。
successBlock	Block	发送消息成功的回调
errorBlock	Block	发送消息失败的回调，其中包含错误码 RCErrorCode 和发送失败的消息。

发送定向媒体消息

使用 sendDirectionalMediaMessage:toUserIdList:pushContent:pushData:attached:progress:successBlock:errorBlock:cancel: 在超级群中发送多媒体消息给指

定的频道中的单个或多个用户。

请在消息对象中设置超级群会话类型、超级群 ID 和 频道 ID。频道 ID 为空时默认向 RCDefault 频道发送消息。注意，RCMessage 中不会保存接收用户 ID 列表。

```
RCImageMessage *imageMessage = [RCImageMessage messageWithImageURI:@"https://test.png"];
RCMessage *message = [[RCMessage alloc] initWithType:ConversationType_ULTRAGROUP targetId:@"Group1"
direction:MessageDirection_SEND messageId:-1 content:imageMessage];

[[RCCoreClient sharedCoreClient] sendDirectionalMediaMessage:message toUserIdList:@[@"user1",@"user2"] pushContent:nil
pushData:nil progress:^(int progress, RCMessage * _Nonnull progressMessage) {
// 多媒体上传进度回调
} successBlock:^(RCMessage * _Nonnull successMessage) {
// 发送成功
} errorBlock:^(RCErrorCode nErrorCode, RCMessage * _Nonnull errorMessage) {
// 发送失败
} cancel:^(RCMessage * _Nonnull cancelMessage) {
// 用户取消发送
}];
```

参数	类型	说明
message	RCMessage	要发送的消息体。必填属性包括会话类型（conversationType），频道 ID（channelId），会话 ID（targetId），消息内容（content）。
userIdList	NSArray	将要发送的用户列表
pushContent	NSString	修改或指定远程消息推送通知栏显示的内容。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。 • 如果希望使用融云默认推送通知内容，可以填 nil。注意自定义消息类型无默认值。 • 如果要为该条消息指定个性化的离线推送通知，必须在以上任一处向融云提供 pushContent 字段内容。 • 如果您自定义的消息类型需要离线推送服务，必须在以上任一处向融云提供 pushContent 字段内容，否则用户无法收到离线推送通知。
pushData	String	远程推送附加信息。您也可以在 RCMessage 的推送属性（RCMessagePushConfig）中配置，会覆盖此处配置，详见配置消息的推送属性。
progress	Block	发送消息进度更新的回调
successBlock	Block	发送消息成功的回调
errorBlock	Block	发送消息失败的回调，其中包含错误码 RCErrorCode 和发送失败的消息。
cancel	Block	用户取消消息发送的回调

获取历史消息 获取历史消息

更新时间:2024-08-30

- 如果您的应用/环境在 2022.10.13 日及以后开通超级群服务，超级群业务中会包含一个 ID 为 `RCDefault` 的默认频道。如果发消息时不指定频道 ID，则该消息会发送到 `RCDefault` 频道中。在获取 `RCDefault` 频道的历史消息时，需要传入该频道 ID。
- 如果您的应用/环境在 2022.10.13 日前已开通超级群服务，在发送消息时如果不指定频道 ID，则该消息不属于任何频道。获取历史消息时，如果不传入频道 ID，可获取不属于任何频道的消息。融云支持客户调整服务至最新行为。该行为调整将影响客户端、服务端收发消息、获取会话、清除历史消息、禁言等多个功能。如有需要，请提交工单咨询详细方案。

获取本地与远端历史消息

getMessages 方法先从本地获取历史消息，本地有缺失的情况下会从服务端同步缺失的部分。当本地没有更多消息的时候，会从服务端拉取。

```
- (void)getMessages:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelId:(NSString *)channelId
option:(RCHistoryMessageOption *)option
complete:(void (^)(NSArray *messages,long long timestamp,BOOL isRemaining, RCErrorCode code))complete
error:(void (^)(RCErrorCode status))errorBlock;
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
channelId	NSString	超级群频道 ID
option	RCHistoryMessageOption	可配置的参数，包括拉取数量、拉取顺序等
complete	Block	获取消息的回调
error	Block	获取消息失败的回调

- `RCHistoryMessageOption` 说明：

参数	说明
recordTime	时间戳，用于控制分页查询消息的边界。默认值为 `0`。
count	要获取的消息数量。如果 SDK < 5.4.1，范围为 [2-20]；如果 SDK ≧ 5.4.1，范围为 [2-100]；默认值为 `0`，表示不获取。
order	拉取顺序。`RCHistoryMessageOrderDesc`：降序，按消息发送时间递减的顺序，获取发送时间早于 `recordTime` 的消息，返回的列表中的消息按发送时间从新到旧排列。`RCHistoryMessageOrderAsc`： 升序，按消息发送时间递增的顺序，获取发送时间晚于 `recordTime` 的消息，返回的列表中的消息按发送时间从旧到新排列。默认值为降序。

从本地数据库中获取消息

使用 `getHistoryMessages` 方法可分页查询指定会话存储在本地数据库中的历史消息，并返回消息对象列表。列表中的消息按发送时间从新到旧排列。

获取指定消息 ID 前的消息

异步获取会话中指定消息之前、指定数量的最新消息实体，返回消息实体 RCMessage 对象列表。

```
- (void)getHistoryMessages:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelId:(nullable NSString *)channelId
oldestMessageId:(long)oldestMessageId
count:(int)count
completion:(nullable void(^)(NSArray<RCMessage *> * _Nullable messages))completion;
```

count 参数表示返回列表中应包含多少消息。oldestMessageId 参数用于控制分页的边界。每次调用 getHistoryMessages 方法时，SDK 会以 oldestMessageId 参数指向的消息为界，继续在下一页返回指定数量的消息。如果需要获取会话中最新的 count 条消息，可以将 oldestMessageId 设置为 -1。

建议获取返回结果中最早一条消息的 ID，并在下一次调用时作为 oldestMessageId 传入，以便遍历整个会话的消息历史记录。

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
channelId	NSString	超级群频道 ID
oldestMessageId	long	以此 messageId 为界，获取发送时间更小的 count 条消息。ID 不存在时，设置为 -1，表示获取最新的 count 条消息。
count	int	需要获取的消息数量。
completion	Block	获取的消息的回调。

获取指定时间戳前后的消息

> 提示
>
> 超级群业务从 5.4.5 开始支持该功能。

异步获取会话中指定时间戳之前之后、指定数量的最新消息实体，返回消息实体 RCMessage 对象列表。

```
- (void)getHistoryMessages:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelId:(nullable NSString *)channelId
sentTime:(long long)sentTime
beforeCount:(int)beforeCount
afterCount:(int)afterCount
completion:(nullable void(^)(NSArray<RCMessage *> * _Nullable messages))completion;
```

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
channelId	NSString	超级群频道 ID
sentTime	long long	以此时间戳为界，获取早于或晚于该时间的消息。
beforeCount	int	需要获取的发送时间早于指定时间戳的消息数量。
afterCount	int	需要获取的发送时间晚于指定时间戳的消息数量。
completion	Block	获取的消息的回调。

获取指定单个消息类型的历史消息

以下方法从指定消息之前的特定类型的最新消息实体，异步返回消息实体 RCMessage 对象列表。

```
- (void)getHistoryMessages:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelId:(nullable NSString *)channelId
objectName:(nullable NSString *)objectName
oldestMessageId:(long)oldestMessageId
count:(int)count
completion:(nullable void(^)(NSArray<RCMessage *> * _Nullable messages))completion;
```

count 参数表示返回列表中应包含多少消息。oldestMessageId 参数用于控制分页的边界。每次调用 getHistoryMessages 方法时，SDK 会以 oldestMessageId 参数指向的消息为界，继续在下一页返回指定数量的消息。如果需要获取会话中最新的 count 条消息，可以将 oldestMessageId 设置为 -1。

建议获取返回结果中最早一条消息的 ID，并在下一次调用时作为 oldestMessageId 传入，以便遍历整个会话的消息历史记录。

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
channelId	NSString	超级群频道 ID
objectName	NSString	消息类型标识。内置消息类型的标识可参见消息类型概述。
oldestMessageId	long	以此 messageId 为界，获取发送时间更小的 count 条消息。ID 不存在时，设置为 -1，表示获取最新的 count 条消息。
count	int	需要获取的消息数量。
completion	Block	获取的消息的回调。

如果需要获取早于或晚于指定消息的历史消息，可以使用以下带 isForward 参数的方法：

```
- (void)getHistoryMessages:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelId:(nullable NSString *)channelId
objectName:(nullable NSString *)objectName
baseMessageId:(long)baseMessageId
isForward:(BOOL)isForward
count:(int)count
completion:(nullable void(^)(NSArray<RCMessage *> * _Nullable messages))completion;
```

参数	类型	说明
conversationType	RCConversationType	会话类型。
targetId	NSString	会话 ID
channelId	NSString	超级群频道 ID
objectName	NSString	消息类型标识列表。内置消息类型的标识可参见消息类型概述。
baseMessageId	long	以此消息 ID 为界，获取早于或晚于该消息的历史消息。
isForward	BOOL	YES 表示获取早于传入时间戳的消息。NO 表示获取晚于传入时间戳的消息。
count	int	需要获取的消息数量。
completion	Block	获取的消息的回调。

获取指定多个消息类型的历史消息

获取会话中，从指定消息之前或之后、指定数量的、多个消息类型的最新消息实体，异步返回消息实体 RCMessage 对象列表。

```
- (void)getHistoryMessages:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelId:(nullable NSString *)channelId
objectNames:(NSArray<NSString *> *)objectNames
sentTime:(long long)sentTime
isForward:(BOOL)isForward
count:(int)count
completion:(nullable void(^)(NSArray<RCMessage *> * _Nullable messages))completion;
```

参数	类型	说明
conversationType	RCConversationType	会话类型。
targetId	NSString	会话 ID
channelId	NSString	超级群频道 ID
objectNames	NSArray	消息类型标识列表。内置消息类型的标识可参见消息类型概述。
sentTime	long long	以此时间戳为界，获取早于或晚于该时间的消息。
isForward	BOOL	YES 表示获取早于传入时间戳的消息。NO 表示获取晚于传入时间戳的消息。
count	int	需要获取的消息数量。
completion	Block	获取的消息的回调。

获取远端历史消息

从 5.6.4 开始，SDK 支持使用 RCChannelClient 的 getRemoteHistoryMessages 方法直接查询指定会话存储在超级群消息云端存储中的历史消息。

> 提示
>
> 用户是否可以获取在加入超级群之前的群聊历史消息取决于 App 在控制台的设置。默认新入群用户只能看到他们入群后的群聊消息。配置方法详见开通超级群服务。

获取会话的远端历史消息

SDK 按照指定条件直接查询并获取超级群消息云端存储中的满足查询条件的历史消息。查询结果与本地数据库对比，排除重复的消息后，返回消息对象列表。返回的消息列表中的消息按发送时间从新到旧排列。因为默认该接口返回的消息会跟本地消息排重后返回，建议先使用 getHistoryMessages，在本地数据库消息全部获取完之后，再获取远端历史消息。否则可能会获取不到指定的部分或全部消息。

RCRemoteHistoryMsgOption 中包含多个配置项，其中 count 与 recordTime 参数分别时获取历史消息的数量与分页查询时间戳。order 参数用于控制获取历史消息的时间方向，可选择获取早于或者晚于给定的 recordTime 的消息。includeLocalExistMessage 参数控制返回消息列表中是否需要包含本地数据库已存在的消息。

```
RCRemoteHistoryMsgOption *option = [RCRemoteHistoryMsgOption new];
option.recordTime = recordTime;
option.count = 10;
option.order = RCRemoteHistoryOrderDesc;
option.includeLocalExistMessage = NO;

[[RCChannelClient sharedChannelManager] getRemoteHistoryMessages:ConversationType_ULTRAGROUP targetId:@"targetId"
channelId:@"channelId" option:option success:^(NSArray<RCMessage *> * _Nonnull messages, BOOL isRemaining) {

} error:^(RCErrorCode status) {
///
}];
```

RCRemoteHistoryMsgOption 默认按消息发送时间降序查询会话中的消息，且默认会并将查询结果与本地数据库对比，排除重复的消息后，再返回消息对象列表。在 includeLocalExistMessage 设置为 NO 的情况下，建议 App 层先使用 getHistoryMessages，在本地数据库消息全部获取完之后，再获取远端历史消息，否则可能会获取不到指定的部分或全部消息。

如需按消息发送时间升序查询会话中的消息，建议获取返回结果中最新一条消息的 sentTime，并在下一次调用时作为 recordTime 的值传入，以便遍历整个会话的消息历史记录。升序查询消息一般可用于跳转到会话页面指定消息位置后需要查询更新的历史消息的场景。

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	会话 ID
channelId	NSString	频道 ID
option	RCRemoteHistoryMsgOption	可配置的参数，包括拉取数量、拉取顺序等
successBlock	Block	获取成功的回调，其中包含获取到的历史消息数组。isRemaining 表示是否还有剩余消息。
errorBlock	Block	获取失败的回调。status 中返回错误码 RCErrorCode 。

- RCRemoteHistoryMsgOption 说明：

参数	说明
recordTime	时间戳，用于控制分页查询消息的边界。默认值为 0。
count	要获取的消息数量。如果 SDK < 5.4.1，范围为 [2-20]；如果 SDK ≧ 5.4.1，范围为 [2-100]；默认值为 0，表示不获取。
order	拉取顺序。RCRemoteHistoryOrderDesc：降序，按消息发送时间递减的顺序，获取发送时间早于 recordTime 的消息，返回的列表中的消息按发送时间从新到旧排列。RCRemoteHistoryOrderAsc：升序，按消息发送时间递增的顺序，获取发送时间晚于 recordTime 的消息，返回的列表中的消息按发送时间从旧到新排列。默认值为 1。
includeLocalExistMessage	是否包含本地数据库中的已有消息。YES：包含，查询结果不与本地数据库排除重复，直接返回。NO：不包含，查询结果先与本地数据库排除重复，只返回本地数据库中不存在的消息。默认值为 NO。

强制从服务端获取特定批量消息

getBatchRemoteUltraGroupMessages: 接口会强制从服务端获取对应的消息。

1. 0 < messages 个数 ≦ 20
2. messages 所有数据必须是超级群类型且为同一个会话
3. message 有效值为 ConversationType，targetId，channelId，messageUid，sentTime

```
matchedMsgList 从服务获取的消息列表
notMatchMsgList 非法参数或者从服务没有拿到对应消息

- (void)getBatchRemoteUltraGroupMessages:(NSArray <RCMessage*>*)messages
success:(void (^)(NSArray *matchedMsgList, NSArray *notMatchMsgList))successBlock
error:(void (^)(RCErrorCode status))errorBlock
```

超级群还支持通过消息 UID 从本地数据库中提取消息。详见获取历史消息 中的通过消息 UID 获取消息。

统计本地历史消息数量

> 提示
>
> RCChannelClient 从 5.6.4 开始支持该能力。

您可以获取指定超级群在指定时间范围内的消息数量。该方法支持传入一个频道列表，如果不指定任何频道，则返回全部频道在本地的历史消息数量。

```
- (void)getUltraGroupMessageCountByTimeRange:(NSString *)targetId
channelIds:(NSArray<NSString *> *)channelIds
startTime:(long long)startTime
endTime:(long long)endTime
success:(nullable void (^)(NSInteger count))successBlock
error:(nullable void (^)(RCErrorCode status))errorBlock;
```

搜索本地消息 搜索本地消息

更新时间:2024-08-30

SDK 从 5.3.4 版本开始支持超级群本地消息搜索功能。

SDK 允许 App 通过关键词、用户 ID 等条件搜索指定超级群频道或多个频道中已拉取到本地的消息，支持按时间段搜索。

并非所有消息类型均支持关键字搜索：

- 内置的消息类型中文本消息（RCTextMessage），文件消息（RCFileMessage），和图文消息`RCRichContentMessage`类型默认实现了`RCMessageCoding`协议的`getSearchableWords`方法。
- 自定义消息类型也可以支持关键字搜索，需要您参考文档自行实现。详见自定义消息类型。

请注意，消息搜索仅查询本地数据库。超级群默认每次连接时只同步频道最后一条消息，因此超级群业务本地消息记录可能不完整，需要 App 自行将消息拉取到本地（一般在进入会话页面时主动获取历史消息）。

提示

融云提供搜索超级群历史消息记录的 IM Server API。详见服务端文档搜索超级群消息。

搜索会话

按关键字搜索本地所有会话。返回符合条件的搜索结果（RCSearchConversationResult）列表。请注意，超级群业务中单个会话（conversation）仅对应单个超级群频道。

```
NSArray *conversationTypeList = @[@(ConversationType_ULTRAGROUP)];
NSArray *objectNameList = @[@"RC:TXT"];

[[RCChannelClient sharedChannelManager] searchConversations:conversationTypeList
messageType:objectNameList
keyword:@"搜索的关键字"
completion:^(NSArray<RCSearchConversationResult *> * _Nullable results) {
//异步回调
}];
```

如果搜索条件与搜索结果包含多个会话类型，您可以通过返回结果中 RCConversation 对象的会话类型（conversationType）字段进行分类或筛选。

参数	类型	说明
conversationTypeList	NSArray	会话类型列表，包含 RCConversationType。例如 ConversationType_PRIVATE，ConversationType_GROUP，ConversationType_ULTRAGROUP。
objectNameList	NSArray	消息类型列表，默认仅支持内置类型 RC:TxtMsg（文本消息）、RC:FileMsg（文件消息）、RC:ImgTextMsg（图文消息）。
keyword	NSString	关键字。不可为空。
completion	Block	异步回调。results 中返回 RCSearchConversationResult 列表。请注意，超级群业务中单个会话（conversation）仅对应单个超级群频道。

搜索单个频道的消息

获取包含关键词的会话列表后，可以搜索指定单个会话中符合条件的消息。在实现超级群消息搜索时，App 可以先通过搜索会话获取符合条件的超级群会话列表，再搜索指定频道中符合条件的消息。

搜索单个频道的消息需要 App 同时提供超级群 ID（targetId）与 频道 ID（channelId）。

根据关键字搜索指定频道的消息

在本地存储中根据关键字搜索指定频道会话中的消息，支持搜索指定时间点之前的历史消息记录。回调中分页返回包含指定关键字的消息列表。

```
[[RCChannelClient sharedChannelManager] searchMessages:ConversationType_ULTRAGROUP
targetId:@"超级群 ID"
channelId:@"超级群频道 ID"
keyword:searchText
count:50
startTime:0
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。超级群会话传入 ConversationType_ULTRAGROUP。
targetId	NSString	超级群 ID。
channelId	NSString	超级群频道 ID。
keyword	NSString	关键字。传空默认为是查全部符合条件的消息。
count	int	最大的查询数量
startTime	long long	询记录的起始时间。传 0 时从最新消息开始搜索。非 0 时从该时间往前搜索。
completion	Block	异步回调。messages 中返回匹配的 RCMessage 列表。

根据关键字搜索指定时间段内指定频道的消息

SDK 支持将关键字搜索的范围限制在指定时间段内。回调中分页返回包含指定关键字和时间段要求的消息列表。

```
[[RCChannelClient sharedChannelManager] searchMessages:ConversationType_ULTRAGROUP
targetId:@"超级群 ID"
channelId:@"超级群频道 ID"
keyword:searchText
startTime:startTime
endTime:endTime
offset:0
limit:100
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

limit 参数控制返回的搜索结果数量，取值范围为 [1-100]。超过 100 默认使用最大值 100。

参数	类型	说明
conversationType	RCConversationType	会话类型。超级群会话传入 ConversationType_ULTRAGROUP。
targetId	NSString	超级群 ID。
channelId	NSString	超级群频道 ID。
keyword	NSString	关键字。传空默认为是查全部符合条件的消息。
startTime	long long	查询记录的起始时间。传 0 时从最新消息开始搜索。非 0 时从该时间往前搜索。
endTime	long long	结束时间
offset	int	偏移量。要求 ≥ 0。
limit	int	返回的搜索结果数量，limit 需大于 0。最大值为 100。超过 100 时默认返回 100 条。
completion	Block	异步回调。messages 中返回匹配的 RCMessage 列表。

根据用户 ID 搜索指定频道的消息

在本地存储中根据搜索来自指定用户 ID 的消息，支持搜索指定时间点之前的历史消息记录。回调中分页返回包含符合条件的消息列表。

```
[[RCChannelClient sharedChannelManager] searchMessages:ConversationType_ULTRAGROUP
targetId:@"接收方 id"
channelId:@"超级群频道 ID"
userId:@"userId"
count:50
startTime:0
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。超级群会话传入 `ConversationType_ULTRAGROUP`。
targetId	NSString	超级群 ID。
channelId	NSString	超级群频道 ID。
userId	NSString	搜索用户 ID。
count	int	最大的查询数量
startTime	long long	查询记录的起始时间。传 `0` 时从最新消息开始搜索。非 `0` 时从该时间往前搜索。
completion	Block	异步回调。`messages` 中返回匹配的 RCMessage 列表。

搜索多个频道的消息

App 如需搜索指定超级群的消息，并已持有超级群 ID（`targetId`），可以直接使用以下接口搜索本地已接收、已存储的消息。符合搜索条件的消息可能来自多个频道。

根据关键字搜索指定超级群本地所有频道的消息

搜索指定超级群存储在本地的消息，支持搜索指定时间点之前的历史消息记录。回调中分页返回包含指定关键字的消息列表。

```
[[RCChannelClient sharedChannelManager] searchMessagesForAllChannel:ConversationType_ULTRAGROUP
targetId:@"超级群 ID"
keyword:searchText
count:50
startTime:0
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

参数	类型	说明
conversationType	RCConversationType	会话类型。超级群会话传入 `ConversationType_ULTRAGROUP`。
targetId	NSString	超级群 ID。
keyword	NSString	关键字。传空默认为是查全部符合条件的消息。
count	int	最大的查询数量
startTime	long long	询记录的起始时间。传 `0` 时从最新消息开始搜索。非 `0` 时从该时间往前搜索。
completion	Block	异步回调。`messages` 中返回匹配的 RCMessage 列表。

根据关键字搜索指定超级群指定时间段内本地所有频道的消息

在通过关键字搜索指定超级群存储在本地的消息时，SDK 支持将搜索范围限制在指定时间段内。回调中分页返回包含指定关键字和时间段要求的消息列表。

```
[[RCChannelClient sharedChannelManager] searchMessagesForAllChannel:ConversationType_ULTRAGROUP
targetId:@"超级群 ID"
keyword:searchText
startTime:startTime
endTime:endTime
offset:0
limit:100
completion:^(NSArray<RCMessage *> * _Nullable messages) {
//异步回调
}];
```

limit 参数控制返回的搜索结果数量，取值范围为 [1-100]。超过 100 默认使用最大值 100。

参数	类型	说明
conversationType	RCConversationType	会话类型。超级群会话传入 ConversationType_ULTRAGROUP。
targetId	NSString	超级群 ID。
keyword	NSString	关键字。传空默认为是查全部符合条件的消息。
startTime	long long	查询记录的起始时间。传 0 时从最新消息开始搜索。非 0 时从该时间往前搜索。
endTime	long long	结束时间
offset	int	偏移量。要求 ≥ 0。
limit	int	返回的搜索结果数量，limit 需大于 0。最大值为 100。超过 100 时默认返回 100 条。
completion	Block	异步回调。messages 中返回匹配的 RCMessage 列表。

根据关键字搜索指定超级群下多个频道的本地历史消息

> 提示
>
> SDK 从 5.6.2 版本开始提供该方法。

请确保传入合法的 Channel ID 数组。Channel ID 数组不可为空，ID 数量不可超过 50 个，数组中的 Channel ID 均必须为合法有效的值（大小写英文字母与数字），不支持 Channel ID 为空字符串。支持从最新消息开始搜索，或者搜索早于 startTime 的消息。每次搜索最大 100 条，如果还有更多的消息，可以将 startTime 设为获取到的消息数组最后一条消息的时间戳，然后继续调用接口。

```
/// 根据关键字搜索指定超级群下多个频道的本地历史消息
///
/// - Parameter conversationType: 会话类型
/// - Parameter targetId: 会话 id
/// - Parameter channelIds: 频道列表，支持一次查询多个频道，不能超过 50 个
/// - Parameter keyword: 查询内容关键字
/// - Parameter startTime: 查询 startTime 之前的消息（传 0 表示从最新消息开始搜索）
/// - Parameter limit: 最大的查询数量，limit 需大于 0，最大值为 100，如果大于 100，会默认成 100。
/// - Parameter successBlock: [messages] 满足条件的消息列表，按时间倒序排序
/// - Parameter errorBlock: [errorCode] 错误码
///
/// - Since: 5.6.2
- (void)searchMessagesForChannels:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelIds:(NSArray<NSString *> *)channelIds
keyword:(NSString *)keyword
startTime:(long long)startTime
limit:(int)limit
success:(void (^)(NSArray<RCMessage *> *messages))successBlock
error:(nullable void (^)(RCErrorCode errorCode))errorBlock;
```

根据发送者用户 ID 搜索指定超级群下多个频道的本地历史消息

> 提示
>
> SDK 从 5.6.2 版本开始提供该方法。

请确保传入合法的 Channel ID 数组。Channel ID 数组不可为空，ID 数量不可超过 50 个，数组中的 Channel ID 均必须为合法有效的值（大小写英文字母与数字），不支持 Channel ID 为空字符串。支持从最新消息开始搜索，或者搜索早于 startTime 的消息。每次搜索最大 100 条，如果还有更多的消息，可以将 startTime 设为获取到的消息数组最后一条消息的时间戳，然后继续调用接口。

```
/// 根据发送者用户 ID 搜索指定超级群下多个频道的本地历史消息
///
/// - Parameter conversationType: 会话类型
/// - Parameter targetId: 会话 id
/// - Parameter channelIds: 频道列表，支持一次查询多个频道，不能超过 50 个
/// - Parameter userId: 消息发送者用户 ID
/// - Parameter startTime: 查询 startTime 之前的消息（传 0 表示从最新消息开始搜索）
/// - Parameter limit: 最大的查询数量，limit 需大于 0，最大值为 100，如果大于 100，会默认成 100。
/// - Parameter successBlock: [messages] 满足条件的消息列表，按时间倒序排序
/// - Parameter errorBlock: [errorCode] 错误码
///
/// - Since: 5.6.2
- (void)searchMessagesByUserForChannels:(RCConversationType)conversationType
targetId:(NSString *)targetId
channelIds:(NSArray<NSString *> *)channelIds
userId:(NSString *)userId
startTime:(long long)startTime
limit:(int)limit
success:(void (^)(NSArray<RCMessage *> *messages))successBlock
error:(nullable void (^)(RCErrorCode errorCode))errorBlock;
```

根据发送者用户 ID 搜索指定超级群下所有频道的本地历史消息

> 提示
>
> SDK 从 5.6.2 版本开始提供该方法。

支持从最新消息开始搜索，或者搜索早于 startTime 的消息。每次搜索最大 100 条，如果还有更多的消息，可以将 startTime 设为获取到的消息数组最后一条消息的时间戳，然后继续调用接口。

```
/// 根据发送者用户 ID 搜索指定超级群下所有频道的本地历史消息
///
/// - Parameter conversationType: 会话类型
/// - Parameter targetId: 会话 id
/// - Parameter userId: 消息发送者用户 ID
/// - Parameter startTime: 查询 startTime 之前的消息（传 0 表示从最新消息开始搜索）
/// - Parameter limit: 最大的查询数量，limit 需大于 0，最大值为 100，如果大于 100，会默认成 100。
/// - Parameter successBlock: [messages] 满足条件的消息列表，按时间倒序排序
/// - Parameter errorBlock: [errorCode] 错误码
///
/// - Since: 5.6.2
- (void)searchMessagesByUserForAllChannel:(RCConversationType)conversationType
targetId:(NSString *)targetId
userId:(NSString *)userId
startTime:(long long)startTime
limit:(int)limit
success:(void (^)(NSArray<RCMessage *> *messages))successBlock
error:(nullable void (^)(RCErrorCode errorCode))errorBlock;
```

获取未读消息数　　获取未读消息数

更新时间:2024-08-30

超级群业务支持从客户端 SDK 获取未读消息数，具体如下：

- 当前用户加入的所有超级群、指定超级群、或指定频道的未读消息数。
- 当前用户加入的所有超级群、指定超级群、或指定频道的未读 @ 消息数。
- 按免打扰级别获取总未读消息数

返回的未读数最大值为 999。如果实际未读数超过 999，接口仍返回 999。

获取多个超级群的未读消息数

SDK 支持获取当前用户加入的所有超级群中未读消息数与未读 @ 消息数。

批量获取当前用户的超级群的未读消息数

> 提示
>
> SDK 从 5.4.6 版本开始支持该接口。仅在 ChannelClient 中提供。

在社群应用场景中，App 可能需要实时显示用户所在的多个超级群下所有频道的最新未读消息数据，可以使用 getUltraGroupConversationUnreadInfoList 一次获取最多 20 个超级群下所有频道的未读数据。具体包含：

- 超级群频道的未读消息数
- 超级群频道的未读 @ 消息数
- 超级群频道中仅 @ 当前用户的未读 @ 消息数
- 超级群频道的免打扰级别

```
[[RCChannelClient sharedChannelManager] getUltraGroupConversationUnreadInfoList:targetIds
success:^(NSArray<RCConversationUnreadInfo *> * _Nonnull list) {
for (RCConversationUnreadInfo *info in list) {
NSString *content = [NSString stringWithFormat:@"%@ %@:%@,%@,%@,%@", info.targetId, info.channelId,
@(info.unreadMessageCount), @(info.mentionedCount), @(info.mentionedMeCount), @(info.notificationLevel)];
NSLog(@"%@", content);
}
} error:^(RCErrorCode status) {
// TODO error code
}];
```

获取当前用户的超级群未读消息数总和

获取当前用户加入的所有超级群会话的未读消息数的总和。

```
- (void)getUltraGroupAllUnreadCount:(void (^)(NSInteger count))successBlock
error:(void (^)(RCErrorCode status))errorBlock;
```

获取当前用户的超级群未读 @ 消息数总和

获取当前用户加入的所有超级群会话中的未读 @ 消息数的总和。

```
- (void)getUltraGroupAllUnreadMentionedCount:(void (^)(NSInteger count))successBlock
error:(void (^)(RCErrorCode status))errorBlock;
```

获取单个超级群的未读消息数

SDK 支持获取指定超级群或频道中的未读消息数和未读 @ 消息数。

获取指定单个超级群的未读消息数

获取当前用户在指定超级群中的未读消息数。

```
- (void)getUltraGroupUnreadCount:(NSString *)targetId
success:(void (^)(NSInteger count))successBlock
error:(void (^)(RCErrorCode status))errorBlock
```

获取指定单个超级群的未读 @ 消息数

获取当前用户在指定超级群中的未读 @ 消息数。

```
- (int)getUltraGroupUnreadMentionedCount:(NSString *)targetId
```

获取指定频道的未读消息数

获取当前用户在超级群会话指定的频道中的未读消息数。

```
- (int)getUnreadCount:(RCConversationType)conversationType targetId:(NSString *)targetId channelId:(NSString *)channelId;
```

获取指定频道的未读 @ 消息数

App 可以直接从 RCConversation 对象上获取该频道未读 @ 消息数。

- `mentionedCount`：当前频道中“@所有人”与“@当前用户”的未读消息数之和。
- `mentionedMeCount`：当前频道中“@当前用户”的未读消息数。要求 SDK 版本 ≧ 5.4.5。

具体示例可参见获取频道列表。

按免打扰级别获取超级群的总未读消息数

> 提示
>
> SDK 从 5.2.5 版本开始支持该接口。仅在 RCChannelClient 中提供。

获取已设置指定免打扰级别的会话及频道的总未读消息数。SDK 将按照传入的免打扰级别配置查找，再返回该会话下符合设置的频道的总未读消息数。

```
NSArray * pushNotificationLevels = @[@(RCPushNotificationLevelMention)];
[[RCChannelClient sharedChannelManager] getUltraGroupUnreadCount:targetId
levels: pushNotificationLevels
success:^(NSInteger unReadCount) {}
error:^(RCErrorCode status) {}];
```

获取成功后，successBlock 中会返回未读消息数。

参数	类型	必填
targetId	NSString	会话 ID
levels	RCPushNotificationLevel[]	免打扰类型数组。详见免打扰功能概述。
successBlock	Block	成功回调
errorBlock	Block	失败回调

按免打扰级别获取超级群的总未读 @ 消息数

> 提示
>
> SDK 从 5.2.5 版本开始支持该接口。仅在 RCChannelClient 中提供。

获取已设置指定免打扰级别的会话及频道的总未读 @ 消息数。SDK 将按照传入的免打扰级别配置查找，再返回该会话下符合设置的频道的总未读 @ 消息数。

```
NSArray * pushNotificationLevels = @[@(RCPushNotificationLevelMention)];
[[RCChannelClient sharedChannelManager] getUltraGroupUnreadMentionedCount:targetId
levels: pushNotificationLevels
success:^(NSInteger unReadCount) {}
error:^(RCErrorCode status) {}];
```

获取成功后，successBlock 中会返回未读消息数。

参数	类型	必填
targetId	NSString	会话 ID
levels	RCPushNotificationLevel[]	免打扰类型数组。详见免打扰功能概述。
successBlock	Block	成功回调
errorBlock	Block	失败回调

如果您希望获取多个类型会话的未读消息总数，可参见处理会话未读消息数中介绍的方法。

处理未读 @ 消息 处理未读 @ 消息

更新时间:2024-08-30

本页面描述了在超级群业务如何实现获取全部未读 @ 消息、跳转到指定未读 @ 消息等功能。

> 提示
>
> 本页面功能依赖融云服务端保存的超级群会话「未读 @ 消息摘要」数据。「@所有人」消息的摘要数据固定存储 7 天（不支持调整）。其他类型 @ 消息的摘要数据与超级群历史消息存储时长一致。

获取全部未读 @ 消息

> 提示
>
> 要求 SDK 版本 ≧ 5.2.5。

获取全部未读 @ 消息

SDK 支持从远端获取指定超级群频道中的全部未读 @ 消息的摘要信息（RCMessageDigestInfo）列表。App 可使用返回的摘要信息，从远端取得未读的 @ 消息。

例如，App 希望获取仅展示未读 @ 消息的场景，步骤如下：

1. 获取超级群频道下的未读 @ 消息摘要，最多可获取 50 条（count 取值范围为 [1-50]）。成功回调中会返回 RCMessageDigestInfo 的列表。每个 RCMessageDigestInfo 对象包含一条未读消息的摘要信息。使用 RCMessageDigestInfo 中的摘要信息构造 RCMessage 列表。每个消息对象需包含从 RCMessageDigestInfo 中获取的 ConversationType，targetId，channelId，messageUid，sentTime。

```
[[RCChannelClient sharedChannelManager] getUltraGroupUnreadMentionedDigests:targetID
channelId:channelID
sendTime:time
count:count
success:^(NSArray<RCMessageDigestInfo *> * _Nonnull digests) {
NSMutableArray *messages = [NSMutableArray array];
for (RCMessageDigestInfo *digest in digests) {
// 可使用 RCMessageDigestInfo 的属性筛选 @ 消息摘要
RCMessage *msg = [[RCMessage alloc] init];
msg.conversationType = digest.conversationType;
msg.targetId = digest.targetId;
msg.channelId = digest.channelId;
msg.sentTime = digest.sentTime;
msg.messageUId = digest.messageUid;
// 从 5.6.0 开始，RCMessageDigestInfo 携带消息类型的唯一标识 ObjectName
msg.objectName = digest.objectName;
[messages addObject:msg];}
}
error:^(RCErrorCode status) {}];
```

2. 您可以使用 getBatchRemoteUltraGroupMessages 方法，传入上一步构建的消息列表，从远端批量提取 @ 消息的完整内容。

```
[[RCChannelClient sharedChannelManager] getBatchRemoteUltraGroupMessages:messages
success:^(NSArray<RCMessage *> * _Nonnull matchedMsgList, NSArray<RCMessage *> * _Nonnull notMatchMsgList) {}
error:^(RCErrorCode status) {}];
```

定位到指定未读 @ 消息

> 提示
>
> 要求 SDK 版本 ≧ 5.2.5。

利用获取超级群会话未读 @ 消息摘要列表接口（getUltraGroupUnreadMentionedDigests），可以实现从会话列表进入会话页面后定位到特定 @ 消息的位置。

1. 获取超级群频道下的未读 @ 消息摘要，最多可获取 50 条（count 取值范围为 [1-50]）。成功回调中会返回 RCMessageDigestInfo 的列表。每个 RCMessageDigestInfo 对象包含一条未读消息的摘要信息。

```
[[RCChannelClient sharedChannelManager] getUltraGroupUnreadMentionedDigests:targetID
channelId:channelID
sendTime:time
count:count
success:^(NSArray<RCMessageDigestInfo *> * _Nonnull digests) {}
error:^(RCErrorCode status) {}];
```

2. 构造 RCHistoryMessageOption 对象用于获取历史消息。从 RCMessageDigestInfo 中取出 sendTime，作为 RCHistoryMessageOption 对象的 recordTime。

```
RCHistoryMessageOption *option = [[RCHistoryMessageOption alloc] init];
option.recordTime = time;
option.count = count;
option.order = RCHistoryMessageOrderDesc;
```

3. （可选）修正 RCHistoryMessageOption 的 recordTime。getMessages 的查询结果中不会包含 message.sentTime == recordTime 的消息。如有需要，可根据 Order 分别对 recordTime 做以下修正，以保证查询结果包含您想要的消息：

   - RCHistoryMessageOrderDesc: recordTime = message.sentTime + 1
   - RCHistoryMessageOrderAsc: recordTime = message.sentTime - 1

4. 使用超级群 RCChannelClient 的获取历史消息接口 getMessages，可以获取特定 @ 消息前、后的历史消息。

```
RCHistoryMessageOption *option = [[RCHistoryMessageOption alloc] init];
option.recordTime = time;
option.count = count;
option.order = RCHistoryMessageOrderDesc;

[[RCChannelClient sharedChannelManager] getMessages:ConversationType_ULTRAGROUP
targetId:targetID
channelId:channelID
option:option
complete:^(NSArray<RCMessage *> * _Nullable messages, RCErrorCode code) {
}];
```

清空未读 @ 消息的时机

建议 App 在以下时机在会话页面清除未读数：

- 会话页面即将消失时(viewWillDisappear)
- 处于会话页面，app 返回前台时
- 已经加载到最新消息时

> 提示
>
> 当调用 syncUltraGroupReadStatus 成功后，客户端本地 Conversation 的 firstUnreadMsgSendTime 会变为 0。服务端保存的第一条未读时间会变为 0，保存的未读 @ 消息摘要列表也会清零.

清除消息未读状态 清除消息未读状态

更新时间:2024-08-30

超级群业务可在多个客户端之间同步消息阅读状态。

同步消息已读状态

调用同步已读状态接口会同时清除本地与服务端记录的消息的未读状态，同时服务端会将最新状态同步给同一用户账号的其他客户端。

- 如果指定了频道 ID（`channelId`），则标记该频道所有消息为全部已读，并同步其他客户端。
- 如果频道 ID 为空，则标记该超级群会话下所有不属于任何频道的消息为全部已读，并同步其他客户端。

> 提示
>
> 超级群暂不支持按时间戳同步已读状态。调用 `syncUltraGroupReadStatus` 会按指定参数的要求标记全部消息为已读。时间戳参数（`timestamp`）未使用，可传入任意数字。

```
- (void)syncUltraGroupReadStatus:(NSString *)targetId
channelId:(NSString *)channelId
time:(long long)timestamp
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode errorCode))errorBlock
```

监听消息已读时间

```
@protocol RCUltraGroupReadTimeDelegate <NSObject>

/**
超级群已读时间同步

@param targetId 会话 ID
@param channelId 频道 ID
@param readTime 已读时间（服务端将未读数清零的时间）
*/
- (void)onUltraGroupReadTimeReceived:(NSString *)targetId channelId:(NSString *)channelId readTime:(long long)readTime;

@end

- (void)setRCUltraGroupReadTimeDelegate:(id<RCUltraGroupReadTimeDelegate>)delegate
```

删除消息　　删除消息

更新时间:2024-08-30

超级群会话消息存储在服务端（免费存储 7 天）和用户设备本地数据库。App 用户通过客户端 SDK 删除自己的历史消息，支持仅从本地数据库删除消息、或仅从融云服务端删除消息。

> 提示
>
> - 客户端的删除消息的操作均指从当前登录用户的历史消息记录中删除消息，不影响会话中其他用户的历史消息记录。
> - 如果 App 的管理员或者某普通用户希望在该 App 中彻底删除一条消息，例如在所有超级群成员的聊天记录中删除一条消息，应使用客户端或服务端的撤回消息功能。消息成功撤回后，原始消息内容会在所有用户的本地与服务端历史消息记录中删除。

功能	本地/服务端	API
从本地删除全部频道的消息（时间戳）	仅从本地删除	deleteUltraGroupMessagesForAllChannel
从本地删除指定频道的消息（时间戳）	仅从本地删除	deleteUltraGroupMessages
从服务端删除指定频道的消息（时间戳）	仅从服务端删除	deleteRemoteUltraGroupMessages
从本地和远端删除消息（消息对象）	同时从本地和服务端删除	deleteRemoteMessages

从本地删除全部频道的消息（时间戳）

> 提示
>
> 从 5.3.0 版本 RCChannelClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

删除本地数据库删除所有频道指定时间戳之前的历史消息。需提供 Unix 时间戳，精确到毫秒。单次操作仅针对单个超级群，不支持一次删除多个超级群中的消息。

如果 App 用户希望从超级群在当前设备上存储的所有历史记录中删除一段时间的记录，可以使用 deleteUltraGroupMessagesForAllChannel 删除所有频道中早于某个时间点（timestamp）的消息。时间戳为 0 表示使用当前时间戳，会从本地清除全部消息。服务端保存的该用户的历史消息记录不受影响。如果该用户从服务端获取历史消息，可能会获取到在本地已删除的消息。

```
[[RCChannelClient sharedChannelManager]
deleteUltraGroupMessagesForAllChannel:@"targetId"
timestamp:timestamp
completion:^(BOOL){
//异步回调，是否成功
}];
```

从本地删除指定频道的消息（时间戳）

> 提示
>
> 从 5.3.0 版本 RCChannelClient 开始，建议使用下方异步返回结果的接口，原同步接口同时废弃。

删除本地数据库删除指定单个频道指定时间戳之前的历史消息。需提供 Unix 时间戳，精确到毫秒。单次操作仅针对单个超级群，不支持一次删除多个超级群中的消息。

如果 App 用户希望从超级群在当前设备上存储的频道消息历史记录中删除一段时间的记录，可以使用 deleteUltraGroupMessages 删除早于某个

时间点（timestamp）的消息。时间戳为 0 表示使用当前时间戳，会从本地清除指定频道中的全部消息。服务端保存的该用户的历史消息记录不受影响。如果该用户从服务端获取历史消息，可能会获取到在本地已删除的消息。

```
[[RCChannelClient sharedChannelManager]
deleteUltraGroupMessages:@"targetId"
channelId:@"channelId"
timestamp:timestamp
completion:^(BOOL){
//异步回调，是否成功
}];
```

从服务端指定频道的消息（时间戳）

从服务端历史消息记录中删除指定单个频道指定时间戳之前的历史消息。需提供 Unix 时间戳，精确到毫秒。单次操作仅针对单个超级群，不支持一次删除多个超级群中的消息。该接口仅清除当前用户在服务端历史消息，本地保存的用户历史消息记录不受影响。

> 提示
>
> 在控制台为 App 启用超级群服务后，融云会自动启用历史消息存储（免费存储 7 天）。

App 用户从本地删除消息后，如果再从服务端获取历史消息，可能会获取到在本地已删除的消息。如果希望删除服务端历史消息记录，可以使用 deleteRemoteUltraGroupMessages 删除早于某个时间点（timestamp）的消息。时间戳为 0 表示使用当前时间戳，会清除指定频道中的全部消息。

```
- (void)deleteRemoteUltraGroupMessages:(NSString *)targetId
channelId:(NSString *)channelId
timestamp:(long long)timestamp
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock
```

从本地和远端删除指定频道的消息（消息对象）

如果 App 用户希望从自己的超级群会话的历史记录中彻底删除一组消息，可以使用以下方法，传入需要被删除的消息对象 RCMessage 列表。请确保所提供的消息均属于同一超级群频道（RCMessage#channelId）。一次最多删除 100 条消息。

删除成功后，该用户无法从本地数据库获取消息。如果从服务端获取历史消息，也无法获取到已删除的消息。

```
[[RCCoreClient sharedCoreClient]
deleteRemoteMessage:ConversationType_ULTRAGROUP
targetId:@"targetId"
messages:messages
success:^{}
error:^(RCErrorCode status) {}];
```

修改消息 修改消息

更新时间:2024-08-30

用户在成功发送超级群消息后，可以主动修改已发送消息的消息内容。

修改本端用户已发消息的内容

本端用户发送消息成功后服务端会返回消息的 UID。如果需要修改消息内容，可以使用 modifyUltraGroupMessage ，传入待修改消息的 Message UID 和新的消息内容进行修改。消息被修改后，RCMessage 对象的 HasChanged 属性会被更新为 YES。注意：消息类型无法修改。如果改前为文本消息，则传入的新消息内容必须为 RCTextMessage 类型。无法修改他人发送的消息。

```
RCTextMessage *newTextMessageContent = [RCTextMessage messageWithContent:@"修改后的文本消息内容"];

[[RCChannelClient sharedChannelManager] modifyUltraGroupMessage:messageUId
messageContent:textMessage
success:^{ }
error:^(RCErrorCode status) {}];
```

监听远端用户的消息变更

使用 setUltraGroupMessageChangeListener 设置 UltraGroupMessageChangeListener 监听器。远端用户修改消息时，应用程序可以通过 onUltraGroupMessageModified 方法收到通知。消息被修改后，Message#isHasChanged() 返回 true。

```
@protocol RCUltraGroupMessageChangeDelegate <NSObject>

/*!
消息扩展更新，删除

@param messages 消息集合
*/
- (void)onUltraGroupMessageExpansionUpdated:(NSArray<RCMessage*>*)messages;

/*!
消息内容发生变更

@param messages 消息集合
*/
- (void)onUltraGroupMessageModified:(NSArray<RCMessage*>*)messages;

/*!
消息撤回

@param messages 消息集合
*/
- (void)onUltraGroupMessageRecalled:(NSArray<RCMessage*>*)messages;

@end

//设置超级群消息变化监听
- (void)setRCUltraGroupMessageChangeDelegate:(id<RCUltraGroupMessageChangeDelegate>)delegate
```

撤回消息　　撤回消息

更新时间:2024-08-30

超级群业务中，消息发送方可撤回已发送成功的消息。撤回成功后，服务端即删除原始消息。

默认情况下，融云对撤回消息的操作者不作限制。如需限制，可考虑以下方案：

- App 客户端自行限制撤回消息的操作者。例如，不允许 App 业务中的普通用户撤回他人发送的消息，允许 App 业务中的管理员角色撤回他人发送的消息。
- 如需避免用户撤回非本人发送的消息，可以提交工单 申请打开**IMLib SDK 只允许撤回自己发送的消息**。从融云服务端进行限制，禁止用户撤回非本人发送的消息。

撤回指定消息

撤回指定消息，只有已发送成功的消息可被撤回。

方法说明

```
//撤回消息
- (void)recallUltraGroupMessage:(RCMessage *)message
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode status))errorBlock
```

参数说明

参数	类型	说明
message	RCMessage	需要撤回的消息
successBlock	Block	撤回成功的回调
errorBlock	Block	撤回失败的回调

- `success` 说明：

回调参数	回调类型	说明
messageId	long	撤回的消息ID，该消息已经变更为新的消息

- `error` 说明：

回调参数	回调类型	说明
errorCode	RCErrorCode	失败错误码

撤回指定消息并删除原始数据

撤回指定消息，并删除移动端发送方、接收方的原始消息数据。

方法说明

```
//撤回消息
- (void)recallUltraGroupMessage:(RCMessage *)message
isDelete:(BOOL)isDelete
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode errorcode))errorBlock
```

参数说明

参数	类型	说明
message	RCMessage	需要撤回的消息
isDelete	isDelete	指定移动端发送方与接收方是否需要从本地删除原始消息记录。为 `false` 时，移动端不会删除原始消息记录，会将消息内容替换为撤回提示（小灰条通知）。为 `true` 时，移动端会删除原始消息记录，不显示撤回提示（小灰条通知）。
successBlock	Block	撤回成功的回调
errorBlock	Block	撤回失败的回调

- `success` 说明：

回调参数	回调类型	说明
messageId	long	撤回的消息ID，该消息已经变更为新的消息

- `error` 说明：

回调参数	回调类型	说明
errorCode	RCErrorCode	失败错误码

监听消息处理

```
@protocol RCUltraGroupMessageChangeDelegate <NSObject>

/*!
消息扩展更新，删除

@param messages 消息集合
*/
- (void)onUltraGroupMessageExpansionUpdated:(NSArray<RCMessage*>*)messages;

/*!
消息内容发生变更

@param messages 消息集合
*/
- (void)onUltraGroupMessageModified:(NSArray<RCMessage*>*)messages;

/*!
消息撤回

@param messages 消息集合
*/
- (void)onUltraGroupMessageRecalled:(NSArray<RCMessage*>*)messages;

@end

//设置超级群消息变化监听
- (void)setRCUltraGroupMessageChangeDelegate:(id<RCUltraGroupMessageChangeDelegate>)delegate
```

扩展消息　　扩展消息

更新时间:2024-08-30

已发送的超级群消息可增加、修改、删除扩展信息。

适用场景：

原始消息增加状态标识的需求，都可使用消息扩展。

- 消息评论需求，可通过设置原始消息扩展信息的方式添加评论信息。
- 礼物领取、订单状态变化需求，通过此功能改变消息显示状态。 例如：向用户发送礼物，默认为未领取状态，用户点击后可设置消息扩展为已领取状态。

> 提示
>
> - 每次设置消息扩展将会产生内置通知消息，频繁设置扩展会产生大量消息。
> - 仅当发送消息时指定 `canIncludeExpansion` 值为 `YES`，才可对消息进行扩展。

设置、更新消息扩展信息

> 提示
>
> 消息扩展功能要求在发送消息前设置该消息为可扩展消息。关于如何修改消息的的可扩展属性，请参考消息扩展中的对 `canIncludeExpansion()` 的说明。

```
//更新消息扩展
- (void)updateUltraGroupMessageExpansion:(NSString *)messageUId
expansionDic:(NSDictionary<NSString *, NSString *> *)expansionDic
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock

//删除消息扩展
- (void)removeUltraGroupMessageExpansion:(NSString *)messageUId
keyArray:(NSArray *)keyArray
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock
```

监听消息处理

```objc
@protocol RCUltraGroupMessageChangeDelegate <NSObject>

/*!
消息扩展更新，删除

@param messages 消息集合
*/
- (void)onUltraGroupMessageExpansionUpdated:(NSArray<RCMessage*>*)messages;

/*!
消息内容发生变更

@param messages 消息集合
*/
- (void)onUltraGroupMessageModified:(NSArray<RCMessage*>*)messages;

/*!
消息撤回

@param messages 消息集合
*/
- (void)onUltraGroupMessageRecalled:(NSArray<RCMessage*>*)messages;

@end

//设置超级群消息变化监听
- (void)setRCUltraGroupMessageChangeDelegate:(id<RCUltraGroupMessageChangeDelegate>)delegate
```

输入状态 输入状态

更新时间:2024-08-30

应用程序可以向超级群中发送当前用户输入状态。超级群内收到通知的用户可以在 UI 展示 “ xxx 正在输入”。

> 提示
>
> 为保证最佳体验，建议在仅在人数小于 10,000 的超级群中使用该功能。

发送输入状态

应用程序可以在当前用户输入文本时调用 sendUltraGroupTypingStatus，发送当前用户输入状态。

```
typedef NS_ENUM(NSUInteger, RCUltraGroupTypingStatus) {
/*!
正在输入文本
*/
RCUltraGroupTypingStatusText = 0,
}


- (void)sendUltraGroupTypingStatus:(NSString *)targetId
channleId:(NSString *)channelId
typingStatus:(RCUltraGroupTypingStatus)status
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock
```

监听输入状态

为了减少服务端和客户端的压力，服务端会将一段时间内（例如 5 秒）用户输入事件汇总之后统一批量下发，所以回调以数组形式提供。应用程序收到通知时可以在 UI 展示 “ xxx 正在输入”。

```objc
@interface RCUltraGroupTypingStatusInfo : NSObject
/*!
会话 ID
*/
@property (nonatomic, copy) NSString *targetId;

/*!
所属会话的业务标识
*/
@property (nonatomic, copy) NSString *channelId;

/*!
用户id
*/
@property (nonatomic, copy) NSString *userId;

/*!
用户数
*/
@property (nonatomic, assign) NSInteger userNumbers;

/*!
输入状态
*/
@property (nonatomic, assign) RCUltraGroupTypingStatus status;

/*!
服务端收到用户操作的上行时间.
*/
@property (nonatomic, assign) long long timestamp;

@end

@protocol RCUltraGroupTypingStatusDelegate <NSObject>
/*!
用户输入状态变化的回调

@param infoArr 正在输入的RCUltraGroupTypingStatusInfo列表（nil 表示当前没有用户正在输入）

@discussion
当客户端收到用户输入状态的变化时，会回调此接口，通知发生变化的会话以及当前正在输入的RCUltraGroupTypingStatusInfo列表。
*/
- (void)onUltraGroupTypingStatusChanged:(NSArray<RCUltraGroupTypingStatusInfo*>*)infoArr;
@end

- (void)setRCUltraGroupTypingStatusDelegate:(id<RCUltraGroupTypingStatusDelegate>)delegate
```

超级群免打扰功能概述 超级群免打扰功能概述

更新时间:2024-08-30

「免打扰功能」用于控制用户在客户端设备离线时，是否可针对离线消息接收推送通知。

- **客户端为离线状态**：会话中有新离线消息时，用户默认通过推送通道收到消息且默认弹出通知。设置免打扰后，融云服务端不会为相关消息触发推送。
- **客户端在后台运行**：会话中有新消息时，用户直接收到消息。如果使用 IMLib，您需要自行判断 App 是否在后台运行，并根据业务需求自行实现本地通知弹窗。

前提条件

请在使用「免打扰功能」前检查是否已集成 APNs 推送。

免打扰设置维度

客户端 SDK 支持超级群业务进行以下多个维度的免打扰设置：

- App 的免打扰设置
- 按超级群或频道设置默认免打扰级别
- 按指定会话类型设置免打扰级别
- 按会话设置免打扰级别
- 按超级群频道设置免打扰级别
- 全局免打扰

> 提示
>
> 超级群离线消息推送通知还会收到控制台超级群默认推送频率设置影响。详见开通超级群服务 。

App 的免打扰设置

以 App Key 为单位，设置整个应用所有用户的默认免打扰级别。默认未设置，等同于全部消息都接收通知。该级别的配置暂未在控制台开放，如有需要，请提交工单。

- **全部消息均通知**：当前 App 下的用户可针对任何消息接收推送通知。
- **未设置**：默认全部消息都通知。
- **仅 @ 消息通知**：当前 App 下的用户仅针对提及（@）当前用户和提及所在群组全体成员的消息接收推送通知。
- **仅 @ 指定用户通知**：当前 App 下，用户仅针对提及（@）当前用户的消息接收推送通知。例如：仅张三会接收且仅接收 “@张三 Hello” 的消息的通知。
- **仅 @群全员通知**：当前 App 下，用户仅针对提及（@）群组全体成员的消息接收推送通知。
- **都不接收通知**：当前 App 下，用户不针对任何消息接收推送通知，即任何离线消息都不会触发推送通知。
- **除 @ 消息外群聊消息不发推送**：当前 App 下，用户针对单聊消息、提及（@）指定用户的消息、和提及（@）群组全体成员的消息接收推送通知。

融云服务端判断是否需要推送时，如果存在以下任何一种用户级别的免打扰配置，以用户级别配置为准：

- 按指定会话类型设置免打扰级别
- 按会话设置免打扰级别
- 全局免打扰

如果不存在用户级别配置，则以消息所属超级群/频道的默认免打扰级别为准。App 级别的免打扰配置的优先级最低。

按超级群/频道设置默认免打扰级别

客户端 SDK 支持为**指定超级群下的所有成员**配置触发推送通知的消息类别，或完全关闭通知。客户端 SDK 提供 `RCPushNotificationLevel`，支持以下六个级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	与融云服务端断开连接后，当前超级群的所有用户可针对指定超级群（或频道）中的所有消息接收通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	与融云服务端断开连接后，当前超级群的所有用户仅针对指定超级群（或频道）中提及（@）当前用户和全体群成员的消息接收通知。
RCPushNotificationLevelMentionUsers	2	与融云服务端断开连接后，当前用户仅针对指定超级群（或频道）中提及（@）当前用户的消息接收通知。例如：张三只会接收 “@张三 Hello” 的消息的通知。
RCPushNotificationLevelMentionAll	4	与融云服务端断开连接后，当前用户仅针对指定超级群（或频道）中提及（@）全部群成员的消息接收通知。
RCPushNotificationLevelBlocked	5	当前用户针对指定超级群（或频道）中的任何消息都不接收推送通知。

具体设置方法详见设置群/频道默认免打扰。

为指定的超级群设置的默认免打扰逻辑，自动适用于群下的所有频道。如果针对频道另行设置了默认免打扰逻辑，则以该频道的默认设置为准。融云服务端判断是否需要为用户发送推送通知时，如果同时存在以下任何一种用户级别的免打扰配置，以下列配置为准：

- 按会话类型设置免打扰级别
- 按会话设置免打扰级别
- 按超级群频道设置免打扰级别
- 全局免打扰

按会话类型设置免打扰级别

> 提示
>
> 客户端 SDK 从 5.2.2.1 开始支持该功能。该功能属于用户级别设置。

客户端 SDK 提供 `RCPushNotificationLevel`，允许用户为会话类型（单聊、群聊、超级群、系统会话）配置触发推送通知的消息类别，或完全关闭通知。提供以下六个级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	与融云服务端断开连接后，当前用户可针对指定类型会话中的所有消息接收通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。 注意：在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	与融云服务端断开连接后，当前用户仅针对指定类型的会话中提及（@）当前用户和全体群成员的消息接收通知。
RCPushNotificationLevelMentionUsers	2	与融云服务端断开连接后，当前用户仅针对指定类型的会话中提及（@）当前用户的消息接收通知。例如：张三只会接收 “@张三 Hello” 的消息的通知。
RCPushNotificationLevelMentionAll	4	与融云服务端断开连接后，当前用户仅针对指定类型的会话中提及（@）全部群成员的消息接收通知。
RCPushNotificationLevelBlocked	5	当前用户针对指定类型的会话中的任何消息都不接收推送通知。

具体设置方法详见按会话类型设置免打扰。

融云服务端判断是否需要为用户发送推送通知时，如果同时存在以下任何一种用户级别的免打扰配置，以下列配置为准：

- 按会话设置免打扰级别
- 按超级群频道设置免打扰级别
- 全局免打扰

按会话设置免打扰级别

> 提示
>
> 客户端 SDK 从 5.2.2 开始支持该功能。该功能属于用户级别设置。

客户端 SDK 提供 RCPushNotificationLevel，允许用户为会话配置触发通知的消息类别，或完全关闭通知。提供以下六个级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	与融云服务端断开连接后，当前用户可针对指定类型会话中的所有消息接收通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。 注意：在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	与融云服务端断开连接后，当前用户仅针对指定会话中提及（@）当前用户和全体群成员的消息接收通知。
RCPushNotificationLevelMentionUsers	2	与融云服务端断开连接后，当前用户仅针对接收指定会话中提及（@）当前用户的消息接收通知。例如：张三只会接收 “@张三 Hello” 的消息的通知。
RCPushNotificationLevelMentionAll	4	与融云服务端断开连接后，当前用户仅针对指定会话中提及（@）全部群成员的消息接收通知。
RCPushNotificationLevelBlocked	5	当前用户针对指定会话中的任何消息都不接收推送通知。

具体设置方法详见按会话设置免打扰。

从 2022.09.01 开始，为指定的超级群设置的免打扰级别，自动适用于群下的所有频道。融云服务端判断是否需要为用户发送推送通知时，如果同时存在以下任何一种用户级别的免打扰配置，以下列配置为准：

- 按超级群频道设置免打扰级别
- 全局免打扰

按超级群频道设置免打扰级别

> 提示
>
> 客户端 SDK 从 5.2.2 开始支持该功能。该功能属于用户级别设置。

客户端 SDK 提供 RCPushNotificationLevel，允许用户为指定超级群频道配置触发通知的消息类别，或完全关闭通知。提供以下六个级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	与融云服务端断开连接后，当前用户可针对指定超级群频道中的所有消息接收通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。 注意：在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	与融云服务端断开连接后，当前用户可针对指定超级群频道中提及（@）当前用户和全体群成员的消息接收通知。
RCPushNotificationLevelMentionUsers	2	与融云服务端断开连接后，当前用户可针对指定超级群频道中提及（@）当前用户的消息接收通知。例如：张三只会接收 “@张三 Hello” 的消息的通知。
RCPushNotificationLevelMentionAll	4	与融云服务端断开连接后，当前用户可针对指定超级群频道中提及（@）全部群成员的消息接收通知。
RCPushNotificationLevelBlocked	5	当前用户针对指定超级群频道中的任何消息都不接收推送通知。

具体设置方法详见按频道设置免打扰。

融云服务端判断是否需要为用户发送推送通知时，如果该用户已配置全局免打扰，则已全局免打扰的配置细节为准。

全局免打扰

客户端 SDK 从 5.2.2 开始提供 RCPushNotificationQuietHoursLevel，允许用户配置何时接收通知以及触发通知的消息类别。提供了以下三个级别：

枚举值	数值	说明
RCPushNotificationQuietHoursLevelDefault	0	未设置。如未设置，SDK 会依次查询消息所属群的用户级别免打扰设置及其他非用户级别设置，再判断是否需要推送通知。
RCPushNotificationQuietHoursLevelMention	1	与融云服务端断开连接后，当前用户仅在指定时段内针对指定会话中提及（@）当前用户和全体群成员的消息接收通知。
RCPushNotificationQuietHoursLevelBlocked	5	当前用户在指定时段内针对任何消息都不接收推送通知。

具体设置方法详见全局免打扰。

早于 5.2.2 的 SDK 版本不支持设置触发通知的消息类别，仅支持设置为接收或不接收推送通知。

免打扰设置的优先级

针对超级群会话，融云服务端会遵照以下顺序搜索免打扰配置。优先级从左至右依次降低，以优先级最高的配置为准判断是否需要触发推送：

全局免打扰设置（用户级）> 指定超级群频道的免打扰设置（用户级）> 指定会话的免打扰设置（用户级）> 指定会话类型的免打扰设置（用户级）> 指定超级群频道的默认免打扰设置（超级群全员）> 指定超级群的免打扰设置（超级群全员）> App 级的免打扰设置

API 接口列表

下表描述了适用于超级群会话的免打扰配置 API 接口。

免打扰配置维度	客户端 API	服务端 API
设置指定时段内，应用全局的免打扰级别（用户级）	详见全局免打扰。	详见 设置用户免打扰时段。
设置指定超级群频道的免打扰级别（用户级）	详见按频道设置免打扰。	详见设置会话免打扰。
设置指定会话的免打扰级别（用户级）	详见按会话设置免打扰。	详见设置会话免打扰。
设置指定类型会话的免打扰级别（用户级）	详见按会话类型设置免打扰。	详见设置会话类型免打扰。
设置指定超级群/频道的默认免打扰设置（超级群全体成员）	详见「超级群管理」下的设置群/频道默认免打扰。	详见「超级群管理」下的设置群/频道默认免打扰
设置 App 级免打扰级别	客户端 SDK 不提供 API。	服务端不提供该 API。

设置群/频道默认免打扰 设置群/频道默认免打扰

更新时间:2024-08-30

超级群业务支持为指定的群，或群频道设置默认免打扰逻辑。默认免打扰逻辑对所有群成员生效，一般由超级群的管理员进行设置。

如果您希望从 App 服务端控制指定超级群，或指定群频道默认免打扰逻辑，可参考服务端 API 文档设置超级群/频道默认免打扰。

注意事项

- 在融云服务端判断是否需要推送超级群消息时，指定的超级群，或群频道的默认免打扰配置优先级均低于用户级别配置。如果存在任何用户级别的免打扰配置，则优先以用户级别免打扰配置为准进行判断。

> 提示
>
> 即时通讯业务免打扰功能的 用户级别设置 支持控制指定的单聊会话、群聊会话、超级群会话、超级群频道的免打扰级别，并可设置全局免打扰的时间段与级别。用户级别设置优先级如下：全局免打扰 > 按频道设置的免打扰 > 按会话设置的免打扰 > 按会话类型设置的免打扰。详见超级群免打扰功能概述。

- 为指定的超级群设置的默认免打扰逻辑，自动适用于群下的所有频道。如果针对频道另行设置了默认免打扰逻辑，则以该频道的默认设置为准。

支持的免打扰级别

指定超级群或群频道的默认免打扰级别可设置为以下任一级别：

枚举值	数值	说明
RCPushNotificationLevelAllMessage	-1	所有消息均可进行通知。
RCPushNotificationLevelDefault	0	未设置。未设置时均为此初始状态。 注意：在此状态下，如果超级群与群频道均为未设置，则认为超级群与频道的默认免打扰级别为全部消息都通知。
RCPushNotificationLevelMention	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人
RCPushNotificationLevelMentionUsers	2	仅针对 @ 指定用户消息进行通知，且仅针对被 @ 的指定的用户进行通知。 如：@张三，则张三可以收到推送； @所有人不会触发推送通知。
RCPushNotificationLevelMentionAll	4	仅针对 @群全员进行通知，即只接收 @所有人的推送信息。
RCPushNotificationLevelBlocked	5	不接收通知，即使为 @ 消息也不推送通知。

```
typedef NS_ENUM(NSInteger, RCPushNotificationLevel) {
/*!
全部消息通知（接收全部消息通知 -- 显示指定关闭免打扰功能）
*/
RCPushNotificationLevelAllMessage = -1,
/*!
未设置（向上查询群或者APP级别设置），存量数据中0表示未设置
*/
RCPushNotificationLevelDefault = 0,
/*!
群聊，超级群 @所有人 或者 @成员列表有自己 时通知；单聊代表消息不通知
*/
RCPushNotificationLevelMention = 1,
/*!
群聊，超级群 @成员列表有自己时通知，@所有人不通知；单聊代表消息不通知
*/
RCPushNotificationLevelMentionUsers = 2,
/*!
群聊，超级群 @所有人通知，其他情况都不通知；单聊代表消息不通知
*/
RCPushNotificationLevelMentionAll = 4,
/*!
消息通知被屏蔽，即不接收消息通知
*/
RCPushNotificationLevelBlocked = 5,
};
```

设置指定超级群的默认免打扰级别

```
- (void)setUltraGroupConversationDefaultNotificationLevel:(NSString *)targetId
level:(RCPushNotificationLevel)level
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock;
```

查询指定超级群的默认免打扰级别

```
- (void)getUltraGroupConversationDefaultNotificationLevel:(NSString *)targetId
success:(void (^)(RCPushNotificationLevel level))successBlock
error:(void (^)(RCErrorCode status))errorBlock;
```

设置指定群频道的默认免打扰级别

```
- (void)setUltraGroupConversationChannelDefaultNotificationLevel:(NSString *)targetId
channelId:(NSString *)channelId
level:(RCPushNotificationLevel)level
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock;
```

查询指定群频道的默认免打扰级别

```
- (void)getUltraGroupConversationChannelDefaultNotificationLevel:(NSString *)targetId
channelId:(NSString *)channelId
success:(void (^)(RCPushNotificationLevel level))successBlock
error:(void (^)(RCErrorCode status))errorBlock;
```

关于本地通知 关于本地通知

更新时间:2024-08-30

IMLib 未实现本地通知功能，需要您的 App 自行实现本地通知。

> 提示
>
> - 当 App 刚进入后台时， App 处于后台活跃状态，SDK 通过长连接通道接收消息，此时 SDK 收到消息会触发消息接收监听。您可以在收到消息监听通知时弹出本地通知。关于消息监听器的说明，详见接收消息。
> - App 进入后台，SDK 的长连接最多存活 2 分钟，超时会主动断开。断开后如有消息会通过 APNs 推送通道推送通知。

由于超级群产品本身的业务特性，App 可能需要配合免打扰级别（详见超级群免打扰功能概述）功能，实现精细化的本地通知控制策略。当前 IMLib SDK 获取免打扰的 API 是异步的，每次都从数据库中获取，可能会造成一定延迟，无法满足部分 App 对本地通知处理的要求。如果您希望实现从内存中获取免打扰级别数据，可以参考融云 IMKit SDK 中的实现方案。

以下简述了 IMKit SDK 的实现方案，相关代码可见 **IMKit 开源工程**（GitHub · Gitee）。我们建议您参考 IMKit 的方案，自行实现相关业务逻辑。

IMKit SDK 持有一个全局的 NSDictionary 数据表，用于缓存查询过的会话的免打扰信息，如果内存中查询失败，会执行一次数据库查找并更新全局数据表。包含如下：

- RCMessageNotificationHelper：消息通知状态辅助查询工具类
- RCIMNotificationDataContext：会话免打扰数据上下文
- RCIMThreadLock：读写锁

调用 `RCMessageNotificationHelper` 下的 `checkNotifyAbilityWith` 方法，查询该消息是否需要显示本地通知。
`show` 为 `YES` 时，表示正常发送当前消息的本地通知；否则不发送本地通知。

```
/// 验证消息是否可以通知
/// @param message 消息
/// @param completion 回调
+ (void)checkNotifyAbilityWith:(RCMessage *)message
completion:(void (^)(BOOL show))completion;
```

IMkit 仅在发送本地通知时对会话的免打扰级别进行判断，因此在 RCIM 类的 `postLocalNotificationIfNeed` 方法中完成。代码示例如下：

```
- (void)postLocalNotificationIfNeed:(RCMessage *)message {
...
if (message.conversationType == ConversationType_Encrypted) {
[RCMessageNotificationHelper checkNotifyAbilityWith:message completion:^(BOOL show) {
if (show) {
....
}
}];
} else {
[RCMessageNotificationHelper checkNotifyAbilityWith:message completion:^(BOOL show) {
if (show) {
[[RCLocalNotification defaultCenter] postLocalNotificationWithMessage:message
userInfo:dictionary];

}
}];
}
...
}
```

IMLib 中，您可以在接收消息的回调中（onRCIMReceiveMessage）调用 checkNotifyAbilityWith 方法，查询接收到的消息是否需要显示本地通知。

```
- (void)onRCIMReceiveMessage:(RCMessage *)message left:(int)left {
[RCDMessageNotificationHelper checkNotifyAbilityWith:message completion:^(BOOL show) {
if (show) {
NSLog(@"[RC] Show");
} else {
NSLog(@"[RC] Hide");
}
}];
}
```

本地通知判断逻辑：消息配置(1) > 用户应用全局 （2） > 用户指定群组频道（3） > 用户指定群 （4）> 用户会话类型（5）> 超级群默认配置 (6)

全局免打扰　　全局免打扰

更新时间:2024-08-30

SDK 支持为当前用户设置全局免打扰时段与免打扰级别。

- 该功能可设置一个从任意时间点（`HH:MM:SS`）开始的免打扰时间窗口。在再次设置或删除用户免打扰时间段之前，当次设置的免打扰时间窗口会每日重复生效。例如，App 用户希望设置永久全天免打扰，可设置 `startTime` 为 `00:00:00`，`period` 为 `1439`。
- 单个用户仅支持设置一个时间段，重复设置会覆盖该用户之前设置的时间窗口。
- 如果 SDK 版本 < 5.2.2，仅支持设置免打扰时段，不支持同时设置免打扰级别。

> 提示
>
> 在经 SDK 设置的全局免打扰时段内：
>
> - 如果客户端处于离线状态，融云服务端将不会进行推送通知。
> - 「全局免打扰时段」为用户级别的免打扰设置，且具有最高优先级。在用户设置了「全局免打扰时段」时，均以此设置的免打扰级别为准。

（推荐）在 App 自行实现本地通知处理时，如果检测到客户端 App 已转至后台运行，可通过 SDK 提供的全局免打扰接口决定是否弹出本地通知，以实现全局免打扰的效果。

设置免打扰时段与级别（SDK >= 5.2.2）

从 SDK 5.2.2 开始，为当前用户设置免打扰时间段时，可使用以下免打扰级别：

枚举值	数值	说明
RCPushNotificationQuietHoursLevelDefault	0	未设置。如未设置，SDK 会依次查询消息所属群的用户级别免打扰设置及其他非用户级别设置，再判断是否需要推送通知。
RCPushNotificationQuietHoursLevelMention	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人的消息。
RCPushNotificationQuietHoursLevelBlocked	5	不接收通知，即使为 @ 消息也不推送通知。

早于 5.2.2 的 SDK 版本仅支持设置为接收或不接收推送通知。

设置免打扰时段与级别

> 提示
>
> 该接口在 `RCChannelClient` 中，从 5.2.2 版本开始支持。

设置消息通知免打扰时间。在免打扰时间内接收到消息时，会根据该接口设置的免打扰级别判断是否需要推送消息通知。

```
[[RCChannelClient sharedChannelManager] setNotificationQuietHoursLevel:@"00:00:00"
spanMins:1439
level:(RCPushNotificationQuietHoursLevel)level
success:^() {} error:^(RCErrorCode status) {}];
```

参数	类型	说明
startTime	NSString	开始时间，精确到秒。格式为 `HH:MM:SS`，例如 `01:31:17`。

参数	类型	说明
spanMins	int	免打扰时间窗口大小，单位为分钟。支持范围为 [1-1439]。
level	RCPushNotificationQuietHoursLevel	• `1`：仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人的消息。如果消息所属会话类型为单聊，则代表不通知。 • `0`：未设置。如未设置，SDK 会依次查询消息所属群的用户级别免打扰设置及其他非用户级别设置，再判断是否需要推送通知。 • `5`：不接收通知，即使为 @ 消息也不推送通知。
successBlock	Block	成功回调
errorBlock	Block	失败回调。回调参数 `status` 包含错误码，参见 RCErrorCode。

移除免打扰时段与级别

> 提示
>
> 该接口在 RCChannelClient 中，从 5.2.2 版本开始支持。

您可以调用以下方法，将 level 设置为 `RCPushNotificationQuietHoursLevelDefault` 将免打扰时间段设置移除。

```
[[RCChannelClient sharedChannelManager] setNotificationQuietHoursLevel:@"00:00:00"
spanMins:1439
level:(RCPushNotificationQuietHoursLevel)level
success:^() {} error:^(RCErrorCode status)
```

获取免打扰时段与级别

> 提示
>
> 该接口在 RCChannelClient 中，从 5.2.2 版本开始支持。

您可以通过以下方法获取免打扰时间段设置。在免打扰时间内接收到消息时，会根据当前免打扰级别判断是否需要推送消息通知。

```
[[RCChannelClient sharedChannelManager] getNotificationQuietHoursLevel:^(NSString *startTime, int spanMins,
RCPushNotificationQuietHoursLevel level) {

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
successBlock	BOOL	成功的回调。`startTime` 为全局免打扰起始时间（格式为 `HH:MM:SS`）。`spanMins` 为全局免打扰持续时长（分钟）。`level` 为设置时传入的 `RCPushNotificationQuietHoursLevel`
errorBlock	BOOL	失败的回调。`status` 为错误码，参见 RCErrorCode。

设置全局免打扰时段（< 5.2.2）

早于 5.2.2 的 SDK 版本仅支持设置免打扰时段。在全局免打扰时间段内接收到消息时，不会有消息提醒。

例外情况：@ 消息为高优先级消息，会跳过全局免打扰逻辑，仍然进行推送通知。

设置全局免打扰时段

提示

该接口在 RCIMClient 中，从 5.2.2 版本开始废弃。

设置全局免打扰时段，以屏蔽所有通知，包括本地通知以及远程推送通知。

```
[[RCIMClient sharedRCIMClient] setNotificationQuietHours:@"00:00:00"
spanMins:1439
success:^{}
error:^(RCErrorCode status) {}];
```

参数	类型	说明
startTime	NSString	开始时间，精确到秒。格式为 HH:MM:SS，例如 01:31:17。
spanMins	int	免打扰时间窗口大小，单位为分钟。支持范围为 [1-1439]。
successBlock	BOOL	成功的回调
errorBlock	BOOL	失败的回调。回调参数 status 包含错误码，参见 RCErrorCode。

获取全局免打扰时段

提示

该接口在 RCIMClient 中，从 5.2.2 版本开始废弃。

通过以下方法，可以获取当前应用设置的静默时间。

```
[[RCIMClient sharedRCIMClient] getNotificationQuietHours:^(NSString *startTime, int spanMins) {

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
successBlock	BOOL	成功的回调。startTime 为全局免打扰开始时间（格式为 HH:MM:SS）。spanMins 为全局免打扰持续时长（分钟）。
errorBlock	BOOL	失败的回调。 status 为错误码，参见 RCErrorCode。

取消全局免打扰时段

提示

该接口在 RCIMClient 中，从 5.2.2 版本开始废弃。

通过以下方法移除之前设置的通知静默时间，移除成功后，会正常收到本地通知或推送通知。

```
[[RCIMClient sharedRCIMClient] removeNotificationQuietHours:^{

} error:^(RCErrorCode status) {

}];
```

参数	类型	说明
successBlock	BOOL	成功的回调。

参数	类型	说明
errorBlock	BOOL	失败的回调。 `status` 为错误码，参见 RCErrorCode 。

集成 APNs 远程推送 集成 APNs 远程推送

更新时间:2024-08-30

融云服务端已集成 APNs 服务端功能。当 App 被杀进程，或者在后台被挂起，或者在后台存活超过 2 分钟，SDK 长连接通道会断开，此时融云服务端可将消息通过 APNs 通道通知客户端。

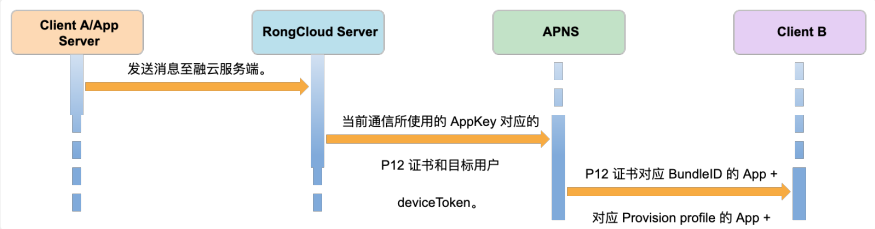
融云服务端对 APNs 的支持

融云服务端支持与 Apple APNs 服务端的以下验证方式：

类别	区别	必要凭证
使用验证令牌（P8）	APNs 侧对基于令牌的推送请求处理更快。同一帐户下的应用程序可以使用同一个 P8 证书，不区分沙盒与生产环境。P8 证书永久有效。	您需要从 Apple 开发人员账户中获取签名密钥（.p8 后缀的文本文件），并提供给融云。融云将使用您的密钥对 APNs 推送请求中的 Token 进行签名。
使用 TSL 证书（P12）	必须与 Apple App ID 绑定，仅可用于该 App。APNs 证书可以只支持沙盒环境（P12 证书），同时支持生产环境和沙盒环境（P12 通用证书）。有效期一年。	您可以从 Apple 开发人员账户获取该证书，并提供给融云。融云将使用该证书与 APNs 进行身份验证。证书在一年后过期，请务必在过期前创建新证书，然后将其提供给融云。

> 提示
>
> 以上内容摘自 Apple 官方开发者文档。如有疑问，详见 Apple 开发者帐户帮助： 使用验证令牌与 APNs 通信 与 使用 TLS 证书与 APN 通信 。

下图以使用 TSL 证书为例，说明融云离线消息触发 APNs 推送的流程：



Apple 开发者账户的操作

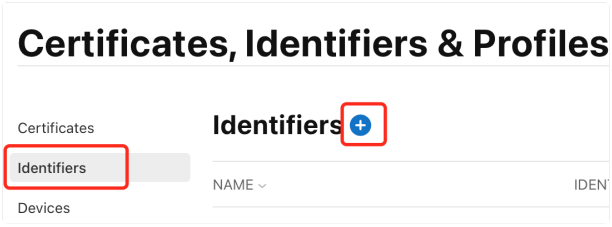
您必须持有 App ID 才能使用 APNs 推送服务。以下描述了如何从 Apple 开发者账户 页面创建 App ID，以及为 App ID 启用 Push Notifications（推送通知功能）。

创建 App ID

> 提示
>
> 如已持有 App ID，请跳过此步骤，直接查看启用 App ID 的推送功能。

以下步骤描述如何从 Apple 开发者账户 创建 App ID：

1. 在 Certificates, Identifiers & Profiles （证书、标识符和描述文件）中，点按边栏中的 **Identifiers** (标识符)，然后点按左上方的添加按钮 (+)。

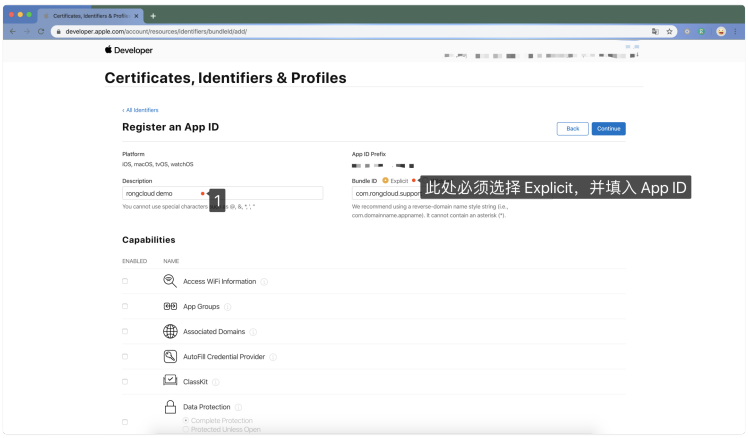


2. 从选项列表中选择 **App IDs**（App ID），点按 **Continue**（继续）。

3. 从选项中，确认已自动选择了 App ID 类型，然后点按 **Continue**（继续）。

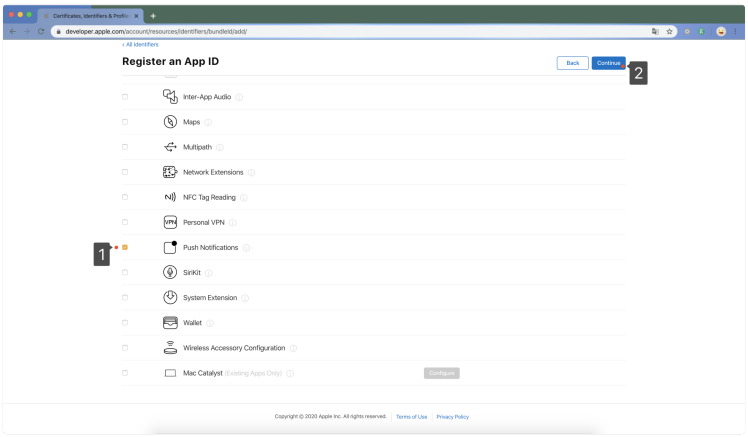
4. 在 **Description**（描述）栏位中输入 App ID 的名称或描述。请选择 **Explicit App ID** （精确 App ID），并在 **Bundle ID** 栏位中输入 App 的 **Bundle ID**。

   您在这里输入的精确 App ID 应该与 Xcode 中目标的 **Summary** （摘要）面板中输入的 **Bundle ID** 一致。



5. **Capabilities** （功能）下面会显示您的 App 类型和可以使用的功能。选中相应的复选框，以启用您想要使用的 App 功能。

   请勾选 **Push Notifications** 以启用推送通知功能。



6. 点按 **Continue**（继续），检查注册信息，然后点按 **Register**（注册）。

关于上述创建 App ID 步骤的详细说明，可参考 Apple 开发者帐户帮助文档：注册 App ID。

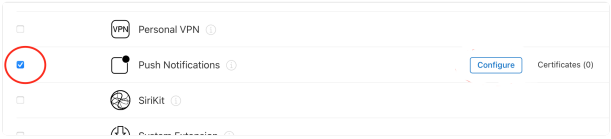
启用 App ID 的推送功能

> 提示
>
> 以下内容来自 Apple 开发者账户帮助文档： 启用推送通知。

以下步骤描述如何从 Apple 开发者账户 为已有的 App ID 启用推送通知功能。如已启用，可跳过该步骤。

1. 在 Certificates, Identifiers & Profiles （证书、标识符和描述文件）中，点按边栏中的 **Identifiers** (标识符)，然后找到对应的 App ID 进行配置。

2. 在 **Capabilities** 中勾选 **Push Notifications**。



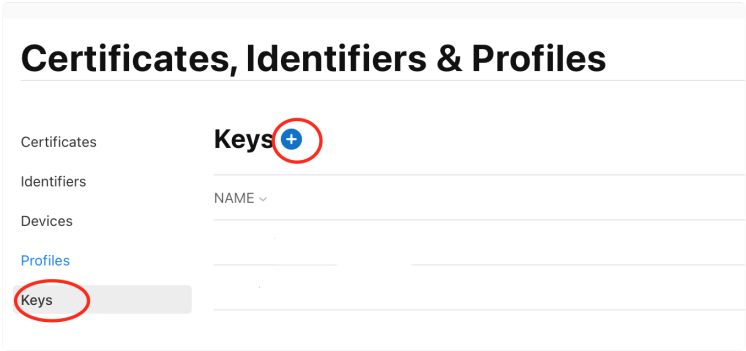
启用 App ID 的推送功能后，请根据需要选择创建 P8 证书或者 P12 证书。

使用验证令牌（P8）

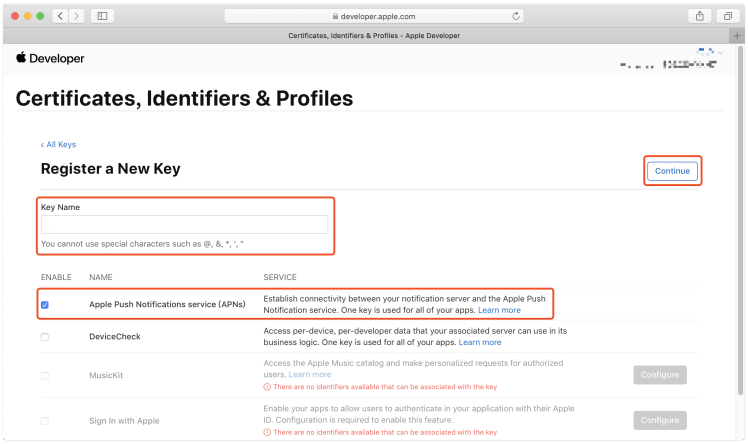
> 提示
>
> 如需了解 “使用验证令牌与 APNs 通信”，可参见 Apple 开发者文档 Establishing a Token-Based Connection to APNs 。以下步骤来自 Apple 开发者账户帮助文档： 创建并下载启用了 APNs 的私钥 。

您需要从 Apple 开发人员账户中创建并下载用于 APNs 推送服务的私钥（.p8 后缀的文本文件，也称为 “P8 证书”）。

1. 在 Certificates, Identifiers & Profiles （证书、标识符和描述文件）中，点按边栏中的 **Keys** (密钥)，再点按左上方的添加按钮 (+)。



2. 在 **Key Name**（密钥名称）下面，为密钥输入唯一的名称。勾选 **Apple Push Notifications service (APNs)**，以启用 APNs 服务，再点按 **Continue**（继续）。



3. 检查密钥配置，再点击 **Confirm**。

4. 点按 **Download**（下载），立即生成并下载密钥文件。密钥会以文件扩展名为 .p8 的文本文件形式下载到本地。

- 注意：请记录下 Key ID，后续需要使用。
- 警告：密钥只可下载一次，下载后将从您的开发者帐户中移除，因此请务必妥善保管。如果 **Download**（下载）按钮处于停用状态，则表示您已经下载过这个密钥。

5. 点按 **Done**（完成）。

获取 .p8 文件后，您可以前往控制台上传文件。详见上传证书到融云。

使用 TSL 证书（P12）

> 提示
>
> 如需了解 p12 证书及“使用 TLS 证书与 APNs 通信”，可参见 Apple 开发者文档 Establishing a Certificate-Based Connection to APNs。

创建 p12 证书需要先从本地创建证书签名请求 (CSR)，再前往 Apple 开发者账户 上传 CSR 生成证书（.cer）文件，下载到本地，在 Mac 上导出为 .p12 格式的证书。

创建证书签名请求

Mac 上的“钥匙串访问”让您可以创建证书签名请求 (CSR)。

1. 打开 Mac 应用中的**钥匙串访问**，选择**证书助理**，再选择**从证书颁发机构请求证书**。

2. 填写证书信息，并保存到磁盘。

- 在**证书助理**对话框中，在“用户电子邮件地址”栏位中输入电子邮件地址。
- 在**常用名称**栏位中，输入密钥的名称 (例如，RongCloud)。
- 将 **CA 电子邮件地址**栏位留空。
- 选取**存储到磁盘**，然后点按**继续**。

生成 cer 格式证书

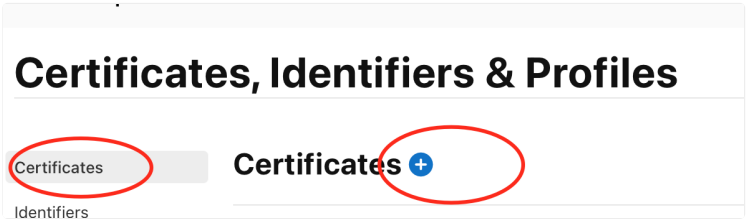
> 提示
>
> 如有多个 App，则需要为每个 App 生成单独的客户端 TLS 证书。

本步骤描述如何从 Apple 开发者账户 生成 `.cer` 后缀的推送证书。注意，本步骤创建的 `.cer` 文件必须转换为 `.p12` 后缀的文件后才能在控制台上传使用。

以下两种方式仅存在 UI 操作上的区别，任选一种即可。

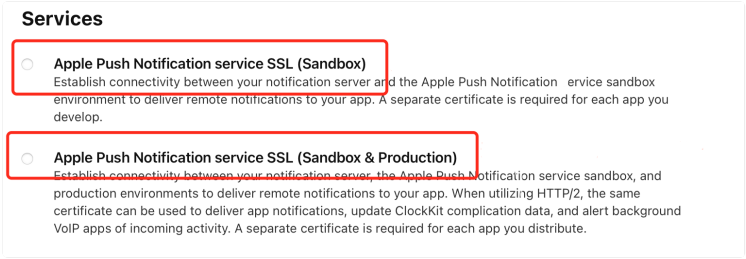
- 方式一：从 Apple 开发账户的 **Certificates**（证书）页面操作，生成 `.cer` 文件。

1. 在 Certificates, Identifiers & Profiles （证书、标识符和描述文件）中，点按边栏中的 **Certificates** (证书)，然后点按左上方的添加按钮 (+)。

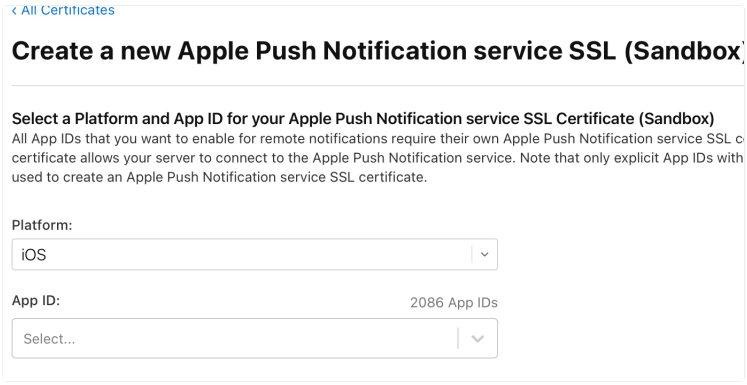


2. 选择证书种类，点按 **Continue**（继续）。

   - **Apple Push Notification service SSL（Sandbox）**：仅可向 APNs Sandbox 环境推送。仅可在控制台的环境使用此类证书，生产环境不允许使用此类证书。
   - **Apple Push Notification service SSL（Sandbox & Production）**：可向 APNs Sandbox 和 Production 环境推送，可在融云应用的开发环境和生产环境中使用。

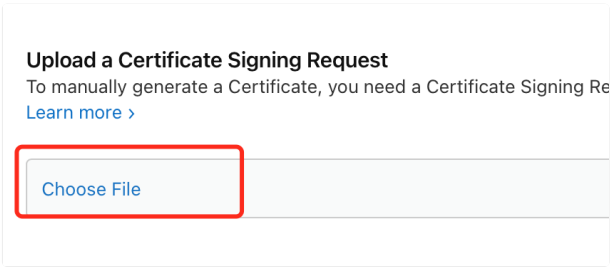


3. 选择需要生成证书的 **App ID**，点按 **Continue**（继续）。



4. 在 Mac 上创建证书签名请求。如果您已按上文指导创建了 CSR 文件，可跳过这一步。

5. 点击 **Choose File**（选取文件），将生成的请求证书（CSR文件）上传，点按 **Continue**（继续）。



6. 点击 **Download** 将证书下载到本地。

- **方式二**：从 Apple 开发账户的 **Identifiers** (标识符)页面操作，生成 `.cer` 文件。

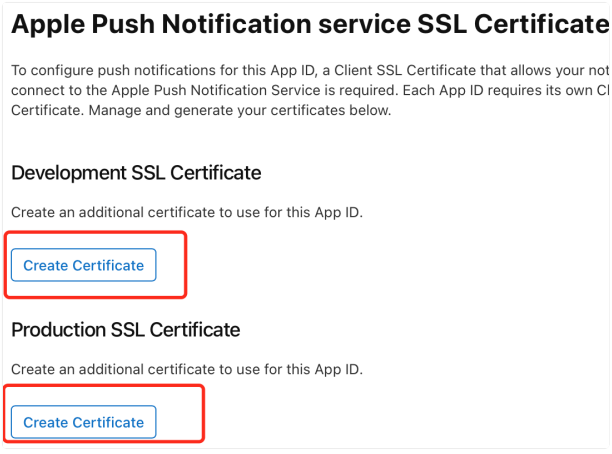
> 提示
>
> 以下步骤来自 Apple 开发者账户帮助文档： 生成 APNs 客户端 TLS 证书 。

1. 在 Certificates, Identifiers & Profiles （证书、标识符和描述文件）中，点按边栏中的 **Identifiers** (标识符)，然后选择要设置的 App ID。

2. 在 **Capabilities**（功能）下，确认已选中 **Push Notifications**（推送通知）复选框。

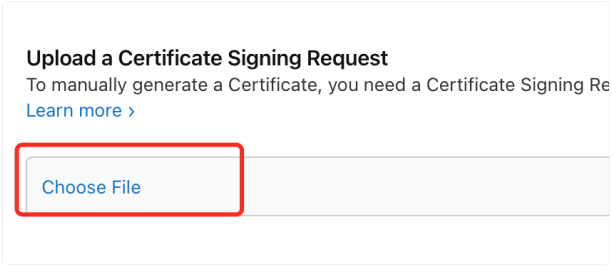
3. 点按 **Configure**（配置），进入创建证书页面。选择需要配置证书的环境，点按 **Create Certificate**（创建证书）。APNs 提供生产与开发两个环境。

   - **Development SSL Certificate**：仅可向 APNs Sandbox 环境推送。仅可在控制台的环境使用此类证书，生产环境不允许使用此类证书。
   - **Production SSL Certificate**：可向 APNs Sandbox 和 Production 环境推送，可在融云应用的开发环境和生产环境中使用。



4. 在 Mac 上创建证书签名请求。如果您已按上文指导创建了 CSR 文件，可跳过这一步。

5. 点击 **Choose File**（选取文件），将生成的请求证书（CSR文件）上传，点按 **Continue**（继续）。



6. 点击 **Download** 将证书下载到本地。

转换 cer 文件为 P12 格式证书

1. 双击下载到本地的证书文件（`.cer`），证书会自动导入钥匙串中。证书的名称为 `Apple [Development/Production] iOS Push Services: [Bundle ID]`，或者 `Apple Push Services: [Bundle ID]`。

2. 在 Mac 应用 **钥匙串访问** 中，左侧点击 **登陆** 和 **证书**，选择刚导入的证书，右键导出为扩展名为 `.p12` 的证书文件。

3. 系统将显示一个对话框，提示输入将用于保护导出项目的密码。您在此输入的密码将在后续将 p12 证书上传到融云后台时使用，请务必记录该密码。

4. 再输入一次证书密码，执行导出操作。

在控制台配置 APNs 推送

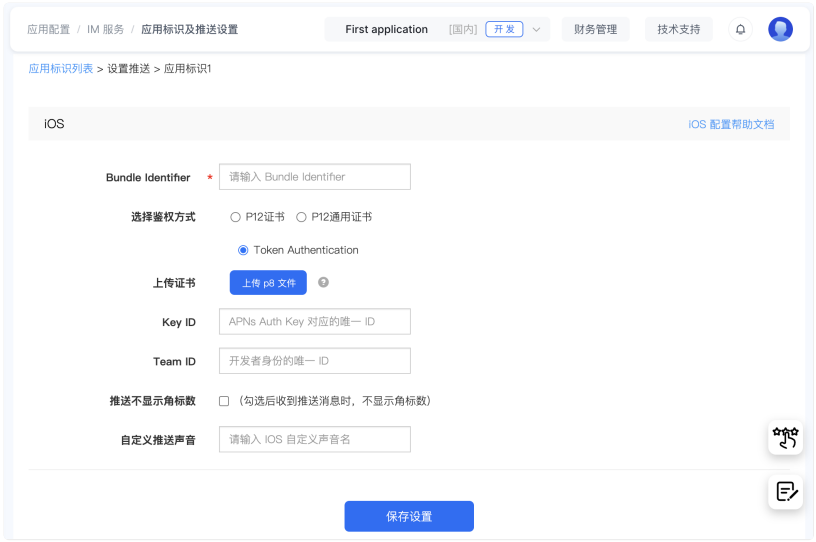
您需要将创建的 APNs Auth Key 文件（P8）或者 TSL 证书（P12）上传到融云后台，融云服务端才能与 APNs 通信，向 iOS 客户端发送推送通知。

1. 前往 控制台，点击 **应用标识** 并找到当前需要集成的项目，然后点击 **设置推送**。如果未创建可点击 **添加**。

2. 填写 **Bundle Identifier**。

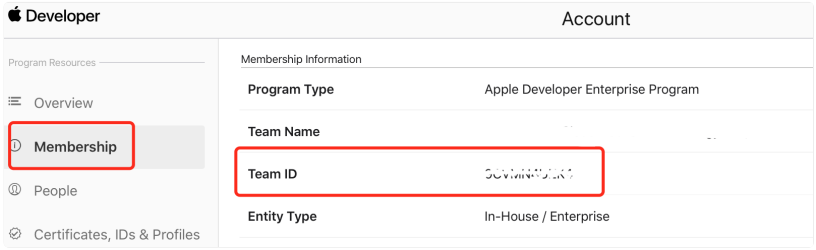
3. 如果您创建了 P8 证书文件，鉴权方式请选择 **Token Authentication**。具体步骤如下：

1. 点击上传 **P8** 文件，选择您创建的 APNs 签名密钥 .p8 文件。



2. 填写 **Key ID**（参见创建 p8 证书）。

3. 填写 **Team ID**。您可以在 Apple 开发者账户 的 Membership 页面获取 Team ID。



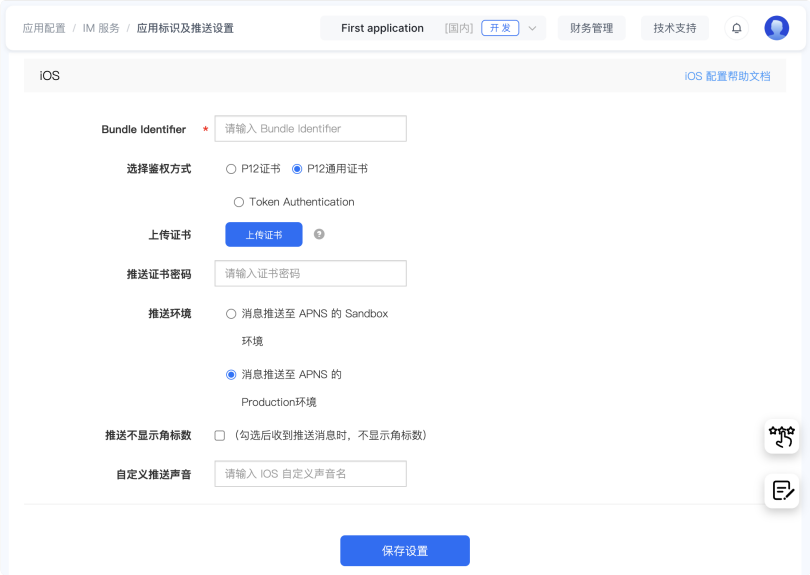
4. 如果您创建了 P12 证书文件，**请注意融云应用的生产环境仅接受 APNs 通用证书（Sandbox & Production）。请根据证书类型和当前环境选择合适的鉴权方式：**

   1. 如果上传 APNs 开发证书（Sandbox），请选择 **P12** 证书。

   2. 如果上传 APNs 通用证书（Sandbox & Production），请选择 **P12** 通用证书。

   3. 上传证书，填写证书密码。

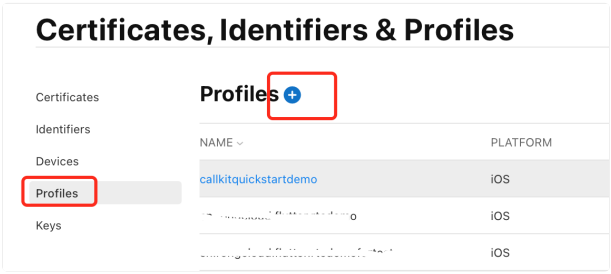
   4. 如果在融云应用的开发环境中使用 APNs 通用证书（Sandbox & Production），请在推送环境中向 APNs 的 Sandbox 或 Production 环境推送。在生产环境使用 APNs 通用证书时，仅支持推送到 APNs 的 Production 环境，无需选择。



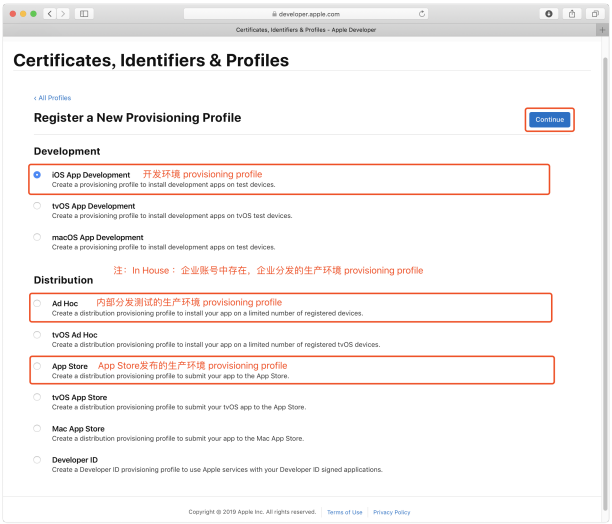
创建 Provisioning Profile 文件

您需要创建一个 Provisioning Profile（配置文件），才能在设备上运行您的应用程序，使用推送通知功能。您需要从 Apple 开发者账户 创建并下载预置描述文件，然后在 Xcode 中安装预置描述文件。

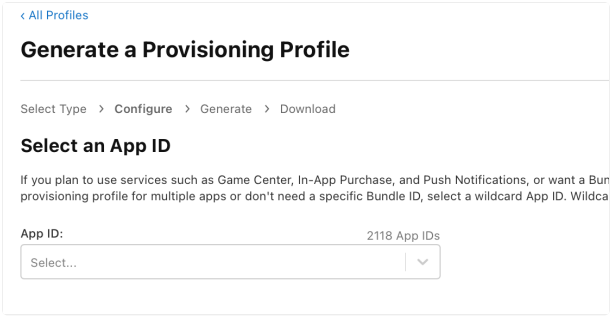
1. 在 Certificates, Identifiers & Profiles （证书、标识符和描述文件）中，点按边栏中的 **Profiles** (标识符)，然后点按左上方的添加按钮 (+)。



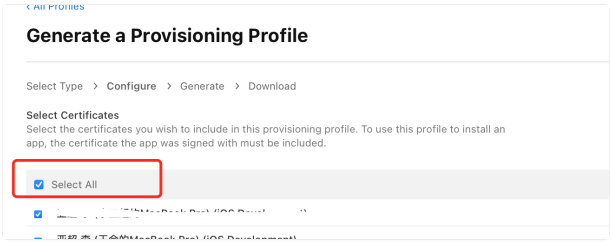
2. 选择 **Provisioning Profile** 的环境后点按 **Continue**



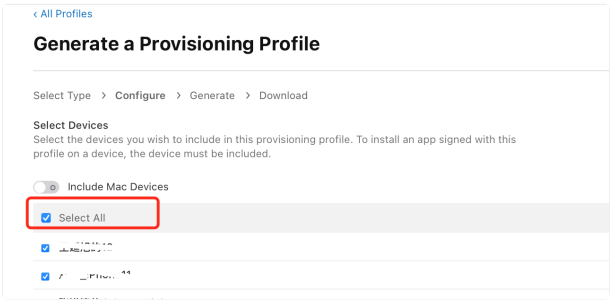
3. 选择要创建 **Provisioning Profile** 的 App ID 后，点按 **Continue**。



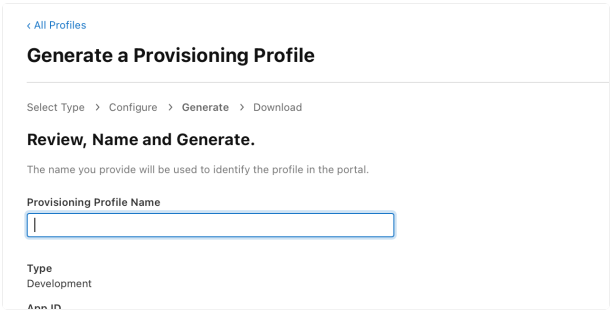
4. 选择所属的开发者证书，如果创建了多个，选择 **Select All**，点按 **Continue**。



5. 为该 **Provisioning Profile** 选择将要安装的设备（一般选择 **Select All**），点按 **Continue**。



6. 填写 **Profile Name**。建议命名为环境 + **AppID**，点击 **generate** 完成创建。



7. 创建成功后，点击 **Download** 下载到本地。

8. 双击下载的 **Provisioning Profile** 文件，添加到 Xcode。

客户端配置

请求推送权限

无论是本地推送还是远程推送，都需要申请权限。

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

if ([[UIDevice currentDevice].systemVersion floatValue] >= 10.0) {
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
center.delegate = self;
[center requestAuthorizationWithOptions:(UNAuthorizationOptionAlert | UNAuthorizationOptionBadge |
UNAuthorizationOptionSound) completionHandler:^(BOOL granted, NSError * _Nullable error) {
if (granted) {
//点击允许
[center getNotificationSettingsWithCompletionHandler:^(UNNotificationSettings * _Nonnull settings) {

}];

} else {

}
}];
}else if ([[UIDevice currentDevice].systemVersion floatValue] >8.0){
//iOS8 - iOS10
[application registerUserNotificationSettings:[UIUserNotificationSettings settingsForTypes:UIUserNotificationTypeAlert |
UIUserNotificationTypeSound | UIUserNotificationTypeBadge categories:nil]];
}
}
```

用户同意后，还需要在用户设备上获取 device token。融云服务器可以使用这个 token 将用向 Apple Push Notification 的服务器提交请求，然后 APNs 通过 token 识别设备和应用，将通知推给用户。

- iOS 10 及以后，向 APNs 推送服务注册，以获取 device Token：

```
// iOS 10 及以后，注册获得device Token
[application registerForRemoteNotifications];
```

- iOS 8-10，在 application:didRegisterUserNotificationSettings: 回调中向 APNs 推送服务注册，以获取 device Token：

```
- (void)application:(UIApplication *)application didRegisterUserNotificationSettings:(UIUserNotificationSettings
*)notificationSettings{
[application registerForRemoteNotifications];
}
```

设置 device token

用户设备向 APNs 注册推送服务后，可获得 NSData 类型的 device Token。

请在 application:didRegisterForRemoteNotificationsWithDeviceToken: 回调中将该 device token 通过 setDeviceTokenData 提供给融云。

```
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {
[[RCIMClient sharedRCIMClient] setDeviceTokenData:deviceToken];
}
```

获取推送数据

如果 App 需要支持 iOS 版本 7+，可使用 didReceiveRemoteNotification:fetchCompletionHandler: 方法获取推送数据（userInfo）。

在 iOS 10 及更高版本上，您需要使用 UNUserNotificationCenterDelegate 的两个 delegate 方法来获取通知信息。

- 如果 App 在前台，可以使用 userNotificationCenter:willPresentNotification:withCompletionHandler: ，获取推送数据（userInfo）。
- App 用户点击通知时可以使用 userNotificationCenter:didReceiveNotificationResponse:withCompletionHandler: ，获取推送数据（userInfo）。

```
/// ios >= 10.0
#pragma mark - UNUserNotificationCenterDelegate

#if __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_10_0
- (void)userNotificationCenter:(UNUserNotificationCenter *)center didReceiveNotificationResponse:(UNNotificationResponse
*)response withCompletionHandler:(void(^)(void))completionHandler NS_AVAILABLE_IOS(10_0){
// userInfo为远程推送的内容
completionHandler();
}
#endif
```

如果点击通知栏的远程推送时，App 已经被系统冻结，可使用下面方式获取：

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
// 远程推送的内容
NSDictionary *remoteNotificationUserInfo = launchOptions[UIApplicationLaunchOptionsRemoteNotificationKey];

...

return YES;
}
```

App 可以通过 getPushExtraFromLaunchOptions 或 getPushExtraFromRemoteNotification 获取远程推送数据。

融云消息的远程推送内容格式如下：

```
{
   "aps" :
{
"alert" : "You got your emails.",
"badge" : 1,
"sound" : "default"
},
"rc":{
"cType":"PR",
"fId":"2121",
"oName":"RC:TxtMsg",
"tId":"3232",
"rId":"3243",
"id":"5FSClm2gQ9V9BZ-kUZn58B",
"rc-dlt-identifier":"2FSClm2gQ9Q9BZ-kUZn54B"
},
"appData":"xxxx"
}
```

参数	类型	说明
alert	String	远程推送显示的内容。自带的消息会有默认显示，如果您使用的是自定义消息，需要在发送时设置。对应于 iOS 发送消息接口中的 pushContent。
cType	String	会话类型。PR（单聊）、DS（讨论组）、GRP（群组）、CS（客服）、SYS（系统会话）、MC（应用内公众服务）、MP（跨应用公众服务）、PH（不落地通知）、UG（超级群）。
bId	String	超级群的频道 ID。仅超级群消息的远程推送内容会携带该字段。如果消息不属于任何频道，则 bId 字段为空字符串。
fId	String	消息发送者的用户 ID。
oName	String	消息类型。融云内置消息类型（详见内置消息类型 中的 ObjectName）或自定义的消息类型。

参数	类型	说明
tId	String	为 Target ID。
rId	String	为接收者的用户 ID。
id	String	当前推送的消息唯一标识，对应消息路由功能中的 msgUID。
rc-dlt-identifier	String	当发送的是一条撤回消息推送时，为需要撤回消息的 ID，对应消息路由功能中的 msgUID。
appData	String	远程推送的附加信息，对应于 iOS 发送消息接口中的 pushData。

自定义推送通知

- 如需自定义远程推送内容，可在发送消息时提供 pushContent 内容。如果使用 IMKit UI 发送消息，可通过 WillSendMessage 回调中拦截并进行设置。
- 如需拦截并修改远程推送内容，App 需要自行创建 `Notification Service Extension`。详见 Apple 官方开发者文档 Modifying Content in Newly Delivered Notifications。
- 如需自定义远程推送通知显示样式，App 需要自行创建 `Notification Content Extension`。详见 Apple 官方开发者文档 Customizing the Appearance of Notifications

测试 APNs 远程推送

完成 APNs 推送集成后，您可以参照测试 APNs 远程推送进行测试。

测试 APNs 远程推送 测试 APNs 远程推送

更新时间:2024-08-30

完成 APNs 集成步骤以后，可直接测试推送是否集成成功。

准备测试 App

安装方式	发布方式	描述文件类型	推送至 APNs Sandbox 环境	推送至 APNs Production 环境
Xcode run（模拟器）	/	/	/	/
Xcode run（真机）	/	开发	支持，必须使用融云应用开发环境的 App Key 和 APNs 配置。	不支持
IPA 文件	Development	开发	支持，必须使用融云开发环境的 App Key 和 APNs 配置。	不支持
第三方 App 平台（如，蒲公英）	Ad Hoc	Ad Hoc 发布	不支持	推荐使用融云应用生产环境的 App Key 和 APNs 配置。
第三方 App 平台（如，蒲公英）	Enterprise	Enterprise 发布	不支持	推荐使用融云应用生产环境的 App Key 和 APNs 配置。
TestFlight	App Store Connect	App Store 发布	不支持	支持，但必须使用融云应用生产环境的 APNs 配置。
App Store	App Store Connect	App Store 发布	不支持	支持，但必须使用融云应用生产环境的 App Key 和 APNs 配置。

准备推送测试环境

融云应用区分开发与生产环境，使用不同的 App Key。融云服务端向 APNs 发送推送请求时可能有多种组合，

必要条件：

- **使用真机**：请使用真机进行测试，模拟器收不到远程推送。
- **使用未越狱的设备**：请勿使用越狱的设备，已越狱的设备可能收不到远程推送。
- **确认 BundleID 中未包含通配符**：使用通配符 BundleID 的 App 将无法使用远程推送。

打包指导：

- **通过 Xcode 在真机运行时**，请使用融云应用的开发环境的 App Key，在融云应用的开发环境中，如果使用 P8、P12 证书，融云服务端仅向 APNs Sandbox 环境推送，如果使用 P12 通用证书，应配置为推送到 APNs Sandbox 环境。
- **通过 Development 方式打包时**，请使用融云应用的开发环境的 App Key，在融云应用的开发环境中，如果使用 P8、P12 证书，融云服务端仅向 APNs Sandbox 环境推送，如果使用 P12 通用证书，应配置为推送到 APNs Sandbox 环境。
- **使用 Ad-Hoc/TestFlight/AppStore 方式打包时**，建议使用融云应用的生产环境的 App Key（需要在控制台申请上线）。只要是 生产环境的 App Key，融云服务端仅向 APNs Production 环境推送。

  例外情况：在融云应用的开发环境中，如果使用 P12 通用证书，允许配置为向 APNs Production 环境推送，以满足测试需求。

如有疑问，您可以参考 Xcode 帮助文档：

- Xcode 帮助文档：Distribute to registered devices (iOS, tvOS, watchOS)。
- Xcode 帮助文档：Distribute an app using TestFlight (iOS, tvOS, watchOS)。

测试接收推送

建议先使用比较简单的单聊场景测试推送。步骤如下：

1. App 连接融云成功之后杀掉 App 进程。
2. 访问控制台的 IM Server API 调试 页面，找到**消息 > 消息服务 >发送单聊消息**，直接给当前 App 用户发送一条单聊消息内容。
3. 查看手机是否收到推送。

故障排除

如果未能收到推送，请先检查以下项目：

- 如果客户端断开连接时设置了不允许推送，例如调用了 `logout` 方法，则会彻底注销在融云服务端的登录信息。融云服务端仅记录离线消息，但不会触发推送服务。
- 如果 App 用户已在 Web/PC 端在线，此时融云认为用户在线，默认不会给移动端发送推送通知。如有需要，您可以在控制台免费基础功能 页面调整 **Web/PC 在线手机端接收 Push** 开关设置。
- 如果 App 用户使用多台移动端设备，融云服务端仅向最后一个登录的设备发送推送通知。

您可以通过控制台的「北极星」查看消息发送与接收状态，以及具体的推送错误：

1. 访问控制台的「北极星」消息流转 页面，切换到正确的环境（开发/生产），根据页面提供的搜索条件搜索消息，点击**查询**。
2. 在查询结果页面，找到需要排障的消息，点击「接收状态」栏中的**查看**。

3. 在**目标用户消息接收状态**下，检查消息已通过推送下发给消息收件人。如果出现问题，该页面会显示具体原因。

4. 根据以下错误，排查是否已参照文档正确集成 APNs：

常见推送错误码	问题原因	解决方案
P16	该消息不属于融云内置类型消息类型（详见消息类型概述 ），缺少推送通知内容（pushContent），因此无法推送。	如果发送自定义消息类型的消息，且需要支持推送，则必须在发送消息时设置 pushContent。
P17	收件人设置了全局免打扰	排查是否设置了对应功能。详见全局免打扰。
P19	收件人设置了单个会话的免打扰	排查是否设置了对应功能。详见免打扰功能概述 或超级群免打扰功能概述。

常见推送错误码	问题原因	解决方案
P23	deviceId 为空	设置 deviceToken。请检查集成步骤是否已完成。详见集成 APNs 远程推送中的设置 **deviceToken**。
P60	加密消息没有推送	无
PUSH_I9	证书加载错误	检查控制台是否上传了正确的推送证书。请检查集成步骤是否已完成。详见集成 APNs 远程推送。如确认证书无误，请联系融云继续排查。
Rong_DeviceToken_Invalid	上传的 deviceToken 有误	设置 deviceToken。请检查集成步骤是否已完成。详见集成 APNs 远程推送中的设置 **deviceToken**。
APNS_4_4_null	证书过期	排查推送证书是否过期，如果过期，需要重新上传到控制台，如确认证书未过期，请联系融云继续排查。
APNS_A2_2400{"reason":"BadDeviceToken"}	打包环境有误	请检查在融云后台上传的证书类型、配置、与您使用的 provisioning file 是否完全匹配。
PUSH_IP8_ERR	打包环境有误	请检查在融云后台上传的证书类型、配置、与您使用的 provisioning file 是否完全匹配。
APNs_XXX	推送至苹果后失败，XXX 为 APNs 错误码	参考 Apple 官方开发者文档的 APNs 错误码说明 Communicating with APNs。

您可以在知识库中查询北极星消息流转返回的全部推送错误码。

5. 如果问题持续，可直接提交工单，提供您的消息 ID 与查询结果。

配置消息的推送属性 配置消息的推送属性

更新时间:2024-08-30

您可以在发送消息时提供 RCMessagePushConfig 配置，对单条消息的推送行为进行个性化配置。例如：

- 自定义推送标题、推送通知内容
- 自定义通知栏图标
- 添加远程推送附加信息
- 越过接收客户端的配置，强制在推送通知内显示通知内容
- 其他 APNs 或 Android 推送通道支持的个性化配置

相对于发送消息时输入参数中的 pushContent 和 pushData，MessagePushConfig 中的配置具有更高优先级。发送消息时，如已配置 RCMessagePushConfig，则优先使用 RCMessagePushConfig 中的配置。

```
RCTextMessage *txtMsg = [RCTextMessage messageWithContent:@"测试文本消息"];

RCMessage *message = [[RCMessage alloc]
initWithType:ConversationType_PRIVATE
targetId:@"targetId"
direction:MessageDirection_SEND
content:txtMsg];

RCMessagePushConfig *pushConfig = [[RCMessagePushConfig alloc] init];
pushConfig.disablePushTitle = NO;
pushConfig.pushTitle = @"通知标题";
pushConfig.pushContent = @"通知内容";
pushConfig.pushData = @"通知的 pushData";
pushConfig.templateId = @"templateId";
pushConfig.iosConfig.threadId = @"iOS 用于通知分组的 id";
pushConfig.iosConfig.apnsCollapseId = @"iOS 用于通知覆盖的 id";
pushConfig.iosConfig.richMediaUri = @"iOS 推送自定义的通知栏消息右侧图标 URL";
pushConfig.androidConfig.notificationId = @"Android 的通知 id";
pushConfig.androidConfig.channelIdMi = @"小米的 channelId";
pushConfig.androidConfig.channelIdHW = @"华为的 channelId";
pushConfig.androidConfig.categoryHW = @"华为的 Category";
pushConfig.androidConfig.channelIdOPPO = @"OPPO 的 channelId";
pushConfig.androidConfig.typeVivo = @"vivo 的 classification";
pushConfig.androidConfig.categoryVivo = @"vivo 的 Category";
pushConfig.forceShowDetailContent = YES;
message.messagePushConfig = pushConfig;

/// 调用 IMKit 或 IMLib 发送消息方法
```

消息推送属性说明

RCMessagePushConfig 提供以下参数：

参数	类型	说明
disablePushTitle	BOOL	是否屏蔽通知标题，此属性只针目标用户为 iOS 平台时有效，Android 第三方推送平台的通知标题为必填项，所以暂不支持。
pushTitle	NSString	推送标题，此处指定的推送标题优先级最高。如不设置，可参考 用户内容类消息格式 中对各内置消息类型默认推送通知标题与推送通知内容的说明。
pushContent	NSString	推送内容。此处指定的推送内容优先级最高。如不设置，可参考 用户内容类消息格式 中对各内置消息类型默认推送通知标题与推送通知内容的说明。
pushData	NSString	远程推送附加信息，如果没有，则使用发送消息的 pushData
forceShowDetailContent	BOOL	是否强制显示通知详情，当目标用户通过 RCPushProfile 中的 - (void)updateShowPushContentStatus:(BOOL)isShowPushContent success:(void (^)(void))successBlock error:(void (^)(RCErrorCode status))errorBlock; 设置推送不显示消息详情时，可通过此参数，强制设置该条消息显示推送详情

参数	类型	说明
templateId	NSString	推送模板 ID，设置后根据目标用户通过 SDK RCPushProfile 中的 setPushLauguageCode 设置的语言环境，匹配模板中设置的语言内容进行推送，未匹配成功时使用默认内容进行推送。模板内容在“控制台-自定义推送文案”中进行设置，具体操作请参见 配置和使用自定义多语言推送模板 。
iOSConfig	RCiOSConfig	iOS 平台相关配置。详见 **RCiOSConfig 属性说明**。
androidConfig	RCAndroidConfig	Android 平台相关配置。**RCiOSConfig 属性说明**。

- **RCiOSConfig 属性说明**

参数	类型	说明
threadId	NSString	iOS 平台通知栏分组 ID，相同的 threadId 推送分为一组（iOS10 开始支持）
apnsCollapseId	NSString	iOS 平台通知覆盖 ID，apnsCollapseId 相同时，新收到的通知会覆盖老的通知，最大 64 字节（iOS10 开始支持）
richMediaUri	NSString	iOS 推送自定义的通知栏消息右侧图标 URL，需要 App 自行解析 `richMediaUri` 并实现展示。图片要求：大小 120 * 120px，格式为 png 或者 jpg 格式。SDK 从 5.2.4 版本开始支持携带该字段。
interruptionLevel	NSString	适用于 iOS 15 及之后的系统。取值为 `passive`，`active`（默认），`time-sensitive`，或 `critical`，取值说明详见对应的 APNs 的 interruption-level 字段。在 iOS 15 及以上版本中，系统的 “定时推送摘要”、“专注模式” 都可能导致重要的推送通知（例如余额变化）无法及时被用户感知的情况，可考虑设置该字段。SDK 5.6.7 及以上版本支持该字段。

- **RCAndroidConfig 属性说明**

参数	类型	说明
notificationId	NSString	Android 平台 Push 唯一标识，目前支持小米、华为推送平台，默认开发者不需要进行设置，当消息产生推送时，消息的 messageUId 作为 notificationId 使用
channelIdMi	NSString	小米的渠道 ID，该条消息针对小米使用的推送渠道，如开发者集成了小米推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，channelId 由开发者自行创建
miLargeIconUrl	NSString	（由于小米官方已停止支持该能力，该字段已失效）小米通知类型的推送所使用的通知图片 url。图片要求：大小120 * 120px，格式为 png 或者 jpg 格式。 此属性 5.1.7 及以上版本支持。支持 MIUI 国内版（国内版要求为 MIUI 12 及以上）和国际版。
channelIdHW	NSString	华为的渠道 ID，该条消息针对华为使用的推送渠道，如开发者集成了华为推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，channelId 由开发者自行创建
hwImageUrl	NSString	华为推送通知中自定义的通知栏消息右侧小图片 URL，如果不设置，则不展示通知栏右侧图片。图标文件须小于 512 KB，图标建议规格大小：40dp x 40dp，弧角大小为 8dp，超出建议规格大小的图标会存在图片压缩或显示不全的情况。 此属性 5.1.7 及以上版本支持
categoryHW	NSString	华为推送通道的消息自分类标识，默认为空。category 取值必须为大写字母，例如 `IM`。App 根据华为要求完成华为自分类权益申请 或 申请特殊权限 后可传入该字段有效。详见华为推送官方文档华为消息分类标准 。该字段优先级高于控制台为 App Key 下的应用标识配置的华为推送 Category。SDK 5.4.0 及以上版本支持该字段。
importanceHW	RCImportanceHw	华为推送的消息提醒级别。`RCImportanceHwLow` 表示通知栏消息预期的提醒方式为静默提醒，消息到达手机后，无铃声震动。`RCImportanceHwNormal` 表示通知栏消息预期的提醒方式为强提醒，消息到达手机后，以铃声、震动提醒用户。终端设备实际消息提醒方式将根据 `categoryHW` 字段取值、或者控制台配置的 `category` 字段取值，或者华为智能分类 结果进行调整。SDK 5.1.3 及以上版本支持该字段。
imageUrlHonor	NSString	荣耀推送通知中用户自定义的通知栏右侧大图标 URL，如果不设置，则不展示通知栏右侧图标。图标文件须小于 512 KB，图标建议规格大小：40dp x 40dp，弧角大小为 8dp，超出建议规格大小的图标会存在图片压缩或显示不全的情况。SDK 5.6.7 及以上版本支持该字段。

参数	类型	说明
importanceHonor	RCimportanceHonor	荣耀推送的 Android 通知消息分类，决定用户设备消息通知行为。`RCImportanceHonorLow` 表示资讯营销类消息。`RCImportanceHonorNormal`（默认值）表示服务与通讯类消息。SDK 5.6.7 及以上版本支持该字段。
typeVivo	NSString	VIVO 推送服务的消息类别。可选值 `0`（运营消息） 和 `1`（系统消息）。该参数对应 VIVO 推送服务的 `classification` 字段，详见 VIVO 推送消息分类说明
categoryVivo	NSString	VIVO 推送服务的消息二级分类。例如 `IM`（即时消息）。该参数对应 VIVO 推送服务的 `category` 字段。详细的 category 取值请参见 VIVO 推送消息分类说明。如果指定二级分类 `categoryVivo`，必须同时指定 `typeVivo`（系统消息或运营消息）。请注意遵照 VIVO 官方要求，确保二级分类属于 VIVO 系统消息场景或运营消息场景下允许发送的内容。`categoryVivo` 字段优先级高于控制台为 App Key 下的应用标识配置的 VIVO 推送 Category。SDK 5.4.2 及以上版本支持该字段。
channelIdOPPO	NSString	OPPO 的渠道 ID，该条消息针对 OPPO 使用的推送渠道，如开发者集成了 OPPO 推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，channelId 由开发者自行创建
fcmCollapseKey	NSString	FCM 推送的通知分组 ID。SDK 5.1.3 及以上版本支持该字段。注意，如使用该字段，请确保控制台的 FCM 推送配置中推送方式为通知消息方式。
fcmImageUrl	NSString	FCM 推送的通知栏右侧图标 URL。如果不设置，则不展示通知栏右侧图标。SDK 5.1.3 及以上版本支持该字段。注意，如使用该字段，请确保控制台的 FCM 推送配置鉴权方式为证书，推送方式为通知消息方式。
fcmChannelId	NSString	FCM 的渠道 ID，该条消息针对 FCM 推送渠道，如开发者集成了 FCM 推送，需要指定 `channelId` 时，可向 Android 端研发人员获取，`channelId` 由开发者自行创建

Channel ID 需要由 Android 端开发者进行创建，创建方式如下：

推送通道	配置说明
华为	App 端，调用 Android SDK 创建 Channel ID 接口创建 Channel ID
小米	在小米开放平台管理台上创建 Channel ID 或通过小米服务端 API 创建
OPPO	App 端，调用 Android SDK 创建 Channel ID；在 OPPO 管理台登记该 Channel ID，保持一致性
vivo	调用服务端 API 创建 Channel ID

用户级推送配置 用户级推送配置

更新时间:2024-08-30

用户级别推送配置是指针对 App 当前登录用户的推送配置，通过 [RCIMClient sharedRCIMClient].pushProfile 来配置。

> 提示
>
> - 用户级别推送配置区别于 App Key 级别推送配置。App Key 级别的推送配置针对 App 下所有用户。您可以在控制台调整部分 App Key 级别的推送服务配置。
> - 用户级别推送配置要求 App Key 已开通用户级别功能设置。如需开通，请提交工单。

以下配置适用于 IMKit 或 IMLib，或其他依赖 IMLib/IMKit 的客户端 SDK。

设置用户推送语言偏好

为当前登录用户设置推送通知的展示语言偏好。在用户未设置偏好的情况下，使用 App Key 级别的 **Push** 语言设置。

融云内置消息类型的默认推送内容中含有部分格式文本字符串。例如，默认情况下用户收到单聊会话的文件消息推送时，推送通知内容中将显示简体中文字符串“[文件]”。如果用户将自己的推送语言偏好修改为美国英语 en_US，则再接收到文件消息时，通知内容中的格式文本字符串将为“[File]”。

上例中的 “[文件]” “[File]” 即格式文本字符串。目前融云服务端为内置消息类型的推送内容提供了格式文本字符串，支持简体中文 zh_CN、美国英语 en_US、阿拉伯语 ar_SA。

```
[[RCIMClient sharedRCIMClient].pushProfile setPushLauguageCode:@"zh_CN" success:^{

} error:^(RCErrorCode status) {

}];

RCPushLauguage lauguage = [RCIMClient sharedRCIMClient].pushProfile.pushLauguage;
```

设置成功后，当前用户接收内置消息类型的推送通知时，推送内容中的格式文本字符串会根据对应语种进行调整。

参数	类型	说明
lauguage	NSString	设置推送通知显示的语言。目前融云支持的内置推送语言为 zh_CN、en_US、ar_SA。自定义推送语言请与 控制台 > 自定义推送文案 中的语言标识保持一致。
successBlock	Block	设置推送语言成功的回调
errorBlock	Block	设置推送语言失败的回调。status 中返回错误码 RCErrorCode 。

目前融云支持的内置推送语言为简体中文 zh_CN、美国英语 en_US、阿拉伯语 ar_SA。App 可以配合使用自定义多语言推送模板功能，可以实现在一条推送通知中支持更多推送语言。

您可以修改 App Key 在融云的默认推送语言配置。如有需要，请提交工单申请更改 App Key 级别的 **Push** 语言。

设置用户推送通知详情偏好

在用户未设置偏好的情况下，默认会展示推送通知内容。该功能支持当前登录用户设置自己的推送通知中是否需要展示推送通知的内容详情。

```
[[RCIMClient sharedRCIMClient].pushProfile updateShowPushContentStatus:NO success:^{

} error:^(RCErrorCode status) {

}];

BOOL isShowPushContent = [RCIMClient sharedRCIMClient].pushProfile.isShowPushContent;
```

如果设置为不显示详情，推送通知将显示为格式文本字符串“您收到了一条通知”（该格式文本字符串支持简体中文 zh_CN、美国英语 en_US、阿拉伯语 ar_SA）。

参数	类型	说明
isShowPushContent	BOOL	是否显示推送的具体内容（YES 显示; NO 不显示）。
successBlock	Block	设置是否显示推送的具体内容成功的回调。
errorBlock	Block	设置是否显示推送的具体内容失败的回调。status 中返回错误码 RCErrorCode。

请注意，发送消息时可以指定 forceShowDetailContent 强制越过消息接收者的该项配置，强制显示推送通知内容详情。以下列出了部分平台的配置：

- Android：Message 的 MessagePushConfig 属性。参见 Android 端「发送消息」文档中的 **MessagePushConfig 属性说明**。
- iOS：RCMessage 的 RCMessagePushConfig 属性。参见 iOS 端「APNs 推送开发指南」下的**配置消息的推送属性**。
- Web：IPushConfig
- IM Server API：若接口提供 forceShowPushContent 参数，则支持该功能。

您也可以修改 App Key 在融云的默认配置。如有需要，请提交工单申请更改 App Key 级别的 **推送通知详情**。关闭后，所有推送通知均默认不显示推送内容详情。

设置用户多端时接收推送偏好

为当前登录用户设置在 Web 端或 PC 端在线时，离线的移动端设备是否需要接收推送通知。请注意，该接口仅在 App Key 已开通 **Web/PC 在线手机端接收 Push** 服务后可用。

您可以前往控制台的免费基础功能页面修改 App Key 级别配置。

- 如果 App Key 未开通 **Web/PC 在线手机端接收 Push**，则所有 App 用户在 Web 端或 PC 端在线时，不在线的移动端不会收到推送。不支持 App 用户修改自己的偏好。
- 如果 App Key 已开通 **Web/PC 在线手机端接收 Push**，当前登录用户可自行关闭或开启该行为。

```
[[RCIMClient sharedRCIMClient].pushProfile setPushReceiveStatus:YES success:^{

} error:^(RCErrorCode status) {

}];

BOOL receiveStatus = [RCIMClient sharedRCIMClient].pushProfile.receiveStatus;
```

参数	类型	说明
receiveStatus	BOOL	设置 Web 端在线时，手机端是否接收推送 （YES 接收; NO 不接收）。
successBlock	Block	设置手机端是否接收推送成功的回调。
errorBlock	Block	设置手机端是否接收推送失败的回调。status 中返回错误码 RCErrorCode。

自定义远程推送铃声 自定义远程推送铃声

更新时间:2024-08-30

远程推送铃声是指收到来自 APNs 推送通道的通知时播放的铃声。App 仅会在关闭状态下收到 APNs 的推送通知。

> 提示
>
> 在 App 退到后台且并未被系统回收的情况下不会触发 APNs 远程推送。如果您集成 IMKit，这种情况下会弹出 IMKit 默认实现的本地通知，播放 IMKit 内置的本地通知铃声，应用程序也可拦截通知播放自定义铃声。详见 IMKit 文档本地通知。如果您集成 IMLib，需要自行弹出本地通知。

自定义推送铃声

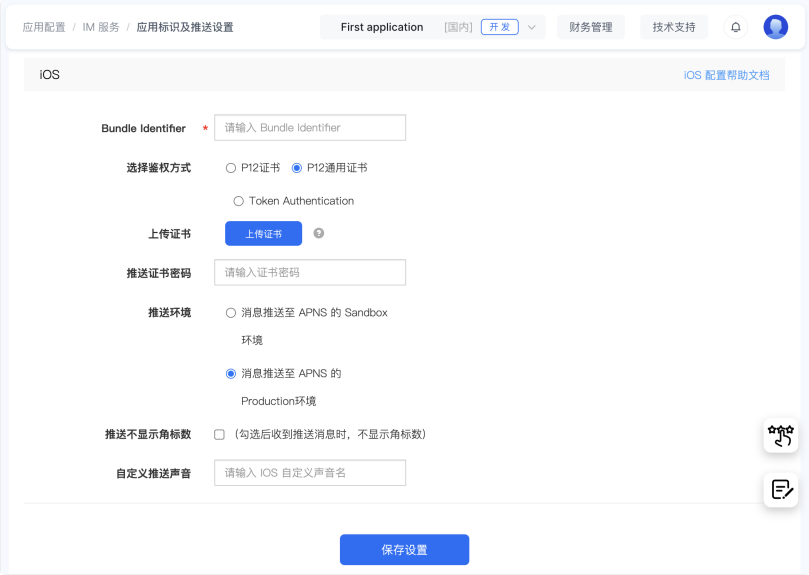
消息推送提示默认使用手机系统设置的声音与振动提示状态。

如需自定义铃声，必须将铃声文件打包到应用程序中。同时需要在控制台配置自定义推送铃声的文件地址。铃声文件地址应该使用应用 ipa 包解压后包内容中的地址。

> 提示
>
> 因 iOS 系统限制，自定义推送铃声时长不能超过 30s。

设置统一推送铃声

1. 访问控制台 应用标识 页面，找到需要设置推送的应用标识。
2. 点击 设置推送，在 iOS 配置区域内找到自定义推送声音，输入自定义铃声文件地址（应用 ipa 包解压后包内容中的文件地址）。



按消息类型设置铃声

融云支持按消息类型设置自定义推送铃声。

1. 访问控制台 自定义推送铃声 页面，在 iOS 标签下点击添加。

2. **输入 Bundle ID、消息类型名称、以及自定义铃声文件地址（应用 ipa 包解压后包内容中的文件地址）。**

上报推送数据 上报推送数据

更新时间:2024-08-30

APNs 推送通道的到达设备、点击通知数据，需要终端主动上报数据后才能获取统计数据，从 SDK 5.1.4 版本开始，融云 iOS SDK 提供了上报数据接口。

> 提示
>
> 推送数据统计功能仅针对已上线应用的生产环境。如果是海外数据中心的应用，请根据 SDK 版本完成配置，确保数据上报到正确的数据中心，详见知识库文档融云海外数据中心使用指南 。

上报推送到达数据

通过 recordReceivedRemoteNotificationEvent: 接口上报推送送达数据。iOS 10 以上系统版本支持此能力。暂仅支持上报单聊、群聊会话类型的推送到达数据，不支持上报超级群推送和不落地推送到达数据。

```
/*!
统计收到远程推送的事件

@param userInfo 远程推送的内容

@discussion 此方法用于统计融云推送服务的到达率。
如果您需要统计推送服务的到达率，需要在 App 中实现通知扩展，并在 NotificationService 的 -didReceiveNotificationRequest:
withContentHandler: 中
先初始化 appkey 再调用此方法并将推送内容 userInfo 传入即可。

@discussion 如果有单独的统计服务地址，还需要在初始化之后设置独立的统计服务地址

如：

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationContent *
_Nonnull))contentHandler {
self.contentHandler = contentHandler;
self.bestAttemptContent = [request.content mutableCopy];

NSDictionary *userInfo = self.bestAttemptContent.userInfo;
[[RCCoreClient sharedCoreClient] initWithAppKey:RONGCLOUD_IM_APPKEY];
if (RONGCLOUD_STATS_SERVER.length > 0) {
[[RCCoreClient sharedCoreClient] setStatisticServer:RONGCLOUD_STATS_SERVER];
}
[[RCCoreClient sharedCoreClient] recordReceivedRemoteNotificationEvent:userInfo];

self.contentHandler(self.bestAttemptContent);
}

@remarks 高级功能
*/
- (void)recordReceivedRemoteNotificationEvent:(NSDictionary *)userInfo;
```

上报推送点击数据

通过 recordRemoteNotificationEvent 接口上报远程推送通知点击数据。

```
/*!
统计远程推送的点击事件

@param userInfo 远程推送的内容

@discussion 此方法用于统计融云推送服务的点击率。
如果您需要统计推送服务的点击率，只需要在 AppDelegate 的-application:didReceiveRemoteNotification:中，
调用此方法并将 launchOptions 传入即可。

@remarks 高级功能
*/
- (void)recordRemoteNotificationEvent:(NSDictionary *)userInfo;
```

自定义多语言推送模板 自定义多语言推送模板

更新时间:2024-08-30

融云支持通过推送模板功能实现多语言推送。服务端会根据 App 用户通过客户端上报的推送语言，从指定推送模板中匹配对应语言的推送内容进行远程推送。

> 提示
> - 客户端 SDK 从 5.0.0 版本开始，支持在发消息时指定使用一个推送模板。
> - App 用户需要通过客户端 SDK 设置接收推送的语言，否则服务端会默认使用 App Key 级别的 **Push** 语言设置为其匹配推送文案。

创建多语言推送模板

在使用推送模板功能前，必须在控制台创建多语言推送模板。

1. 访问控制台 自定义推送文案 页面，创建推送模板，最多可创建 100 个。

- **推送模板 ID**：推送模板的唯一标识。发送消息时，若使用该推送模板，需填入此模板 ID。
- **推送模板名称**：设置推送模板的名称。
- **推送内容设置**：每个推送模板中支持设置多个语言的推送内容，每种语言对应一条推送标题和一条推送内容。

2. 添加推送内容。您可以按**语言标识**逐个设置对应的推送标题与推送内容。例如，模板 `asia` 中可同时添加 zh_CN、zh_TW、zh_HK、ja_JP、ko_KR 等多种语言的对应文案。

- **语言标识**：从下拉菜单中选择一个语言标识，如 en_US，以添加与该语言匹配的文案。
- **推送标题**：设置对应语言的通知栏标题。可选参数，默认单聊为用户名，群聊、超级群为群名称。请注意发消息时指定的推送标题会覆盖模板中的标题，导致推送模板中设置的标题不生效。
- **推送内容**：设置对应语言的通知栏显示内容。请注意发消息时指定的推送内容会覆盖模板中的内容，导致推送模板中设置的内容不生效。

创建模板后 30 分钟生效。

使用自定义推送模板

客户端 SDK 支持在发送消息时，在消息推送属性配置 `MessagePushConfig` 中传入模版 ID（`templateId`），为消息启用指定的多语言推送模板。设置后，如该消息触发推送，服务端会根据指定推送模板中的多语言内容进行推送。

如果接收方客户端设置了推送语言，服务端将从推送模板中匹配用户上报的推送语言进行远程推送。如果用户未设置，或无法匹配用户推送语言，服务端根据 App 在融云服务端的默认推送语言内容进行推送。

客户端 SDK 在设置接收推送的语言时，语言标识需要控制台的语言标识一致，才能匹配成功。例如，客户端设置推送语言为美式英语 en-US，而推送模板中美式英语的表示法为 en_US，则无法成功匹配。

> 提示
>
> - 关于如何指定推送模版 ID（`templateId`），详见 配置消息的推送属性。注意，如需使用推送模板功能，请将消息推送属性中推送标题与推送内容留空。
> - 关于客户端如何设置接收推送的语言，详见 用户级推送配置 > 推送语言。
> - 您可以修改 App Key 在融云的默认推送语言配置。如有需要，请提交工单申请更改 App Key 级别的 **Push 语言**。

内容审核概述 内容审核概述

更新时间:2024-08-30

即时通讯支持对 IM 内容进行审核。

- 即时通讯（IM）服务已内置敏感词机制。注意，敏感词机制仅是一种基础保护机制，且仅限于文本内容（默认最多 50 个敏感词），不可替代专业内容审核服务。
- 融云的内容审核服务产品中的 IM 审核服务，可为 IM 内容提供全面的保障与支持，支持审核文本、图片、语音片段、小视频，精准识别敏感信息。
- 如需自行实现审核或对接第三方审核服务，可以使用**消息回调服务**。

如果消息因被判定违规导致无法下发收件人，默认情况下消息发送者不会收到通知。如果 App 希望通知消息发送者消息已被拦截，可提交工单开通**含敏感词消息屏蔽状态回调发送端**，并在客户端设置监听（要求 Android/iOS SDK 版本 ≧ 5.1.4，Web ≧ 5.0.2）。详见敏感信息拦截回调。

敏感词机制

> 提示
>
> - 客户端不提供针对该功能的管理接口，仅提供回调接口，可在消息被判定为不下发时通知消息发送方。详见敏感信息拦截回调。
> - 如需审核文本（支持语义检测）、图片、语音片段、小视频，建议使用融云提供的内容审核服务产品。

敏感词机制是一种基础保护机制，仅支持对文本消息内容中的敏感词进行识别与过滤。对命中敏感词的消息，您可以选择进行屏蔽该消息（不会下发给接收方），或按指定规则替换消息中的敏感词后再进行下发。

目前支持的敏感词过滤语言包括：中文、英文、日语、德语、俄语、韩语、阿拉伯语。

您可以通过以下方式管理 App Key 下开发环境或生产环境的敏感词：

功能描述	客户端 API	融云服务端 API	控制台
添加敏感词，支持设置替换内容	不提供该 API	添加敏感词	敏感词设置页面
移除敏感词	不提供该 API	移除敏感词	敏感词设置页面
批量移除敏感词	不提供该 API	批量移除敏感词	敏感词设置页面
获取敏感词列表，支持获取设置的替换内容	不提供该 API	获取敏感词列表	敏感词设置页面

默认行为

- 默认最多设置 50 个敏感词。
- 默认仅针对从客户端 SDK 发送的消息生效。
- 默认仅支持识别官方内置的文本消息类型（消息标识为 `RC:TxtMsg`）中的敏感词。支持单聊、群聊、聊天室、超级群会话。超级群中文本消息修改后的内容默认也会敏感词识别、拦截或过滤。

调整配置

- **IM 旗舰版**或 **IM 尊享版**可以在控制台 IM 服务管理 页面的**扩展服务**标签下自行调整敏感词上限数。具体功能与费用以融云官方价格说明页面及计费说明文档为准。
- 如果您对使用服务端 API 发送的消息进行敏感词过滤，可以在控制台的免费基础功能界面打开 **Server API 发送消息过滤敏感词**开关。
- 如果您需要对自定义消息类型启用敏感词机制，可以在敏感词设置页面点击**设置自定义消息**。提供自定义消息的消息类型的 `ObjectName`，及

该消息类型下内容（Content）JSON 结构中对应的键值 Key，即可对该 Key 所对应的 Value 值进行敏感词过滤处理。

IM 内容审核服务

> 提示
>
> 客户端不提供针对该功能的管理接口，仅提供回调接口，可在消息被判定为不下发时通知消息发送方。详见敏感信息拦截回调。

如果您希望全面审核 IM 内容，可以使用融云的内容审核服务产品，该产品提供 **IM 审核服务**与**音视频审核服务**。

IM 审核针对即时通讯业务，具体可提供以下能力：

- 审核文本内容
- 审核图片
- 审核语音片段
- 审核小视频
- 审核自定义消息类型（需要提交工单申请）
- 审核超级群业务中的消息修改
- 从控制台查看审核报告
- 从控制台查询 IM 审核记录
- 审核结果回调

您可以在控制台的 IM & 音视频审核 页面开通 **IM 审核**服务，配置接收审核结果回调的地址。详见服务端文档审核结果回调。

IM 内容审核计费

内容审核服务为付费服务，开发环境可免费体验，生产环境下需预存才能使用服务。具体计费说明详见资费标准 · IM 审核。

消息回调服务

如果您希望对接自己的审核系统或其他第三方内容审核服务，可以使用消息回调服务。

消息回调服务（**原模版路由**）提供一种消息过滤机制。您可以根据发送用户 ID、接收用户 ID、消息类型、会话类型等参数，将相应的消息同步到您指定的服务器。超级群业务中，修改消息内容、更新消息扩展也支持通过消息回调同步到您指定的服务器。

消息同步到您指定的服务器后，可以使用您自己的审核系统执行内容审核，也可以对接其他第三方审核系统。融云服务端会根据您应用服务器返回的响应结果，决定是否将消息下发、是否替换消息中的内容，以及如何进行内容替换。

您可以通过控制台的消息回调服务页面管理 App Key 下开发环境或生产环境的消息回调服务状态和路由规则。

关于如何创建路由规则，以及回调参数的具体说明，请参见消息回调服务文档。

消息回调服务计费

费用以融云官方价格说明页面及计费说明文档为准。

敏感信息拦截回调 敏感信息拦截回调

更新时间:2024-08-30

SDK 从 5.1.4 版本开始支持敏感信息拦截回调。

默认情况下，消息发送方无法感知消息是否已被融云审核服务拦截。如果 App 希望在消息因触发审核规则而无法下发时通知消息发送方，可开通使用敏感信息拦截回调服务。

融云的内容审核服务（包括消息敏感词、IM 审核服务、消息回调服务），可能在以下情况下拦截消息：

- 文本消息内容命中了融云内置的消息敏感词，导致消息不下发给接收方。
- 文本消息内容命中了您自定义的消息敏感词（屏蔽敏感词），导致消息不下发给接收方。
- 消息命中了 **IM 审核服务**，或**消息回调服务**设置的审核规则，导致消息不下发给接收方。

开通服务

如有需求，请提交工单，申请开通**含敏感词消息屏蔽状态回调发送端**。客户端 SDK 5.1.4 及之后版本支持该回调服务。

敏感信息拦截代理委托

在发出的消息被拦截时，SDK 会触发 `RCMessageBlockDelegate` 的以下方法：

```
@protocol RCMessageBlockDelegate <NSObject>

/*!
发送消息被拦截的回调方法
@param blockedMessageInfo 被拦截消息的相关信息
*/
- (void)messageDidBlock:(RCBlockedMessageInfo *)blockedMessageInfo;

@end
```

`RCBlockedMessageInfo` 里包含了被拦截消息的相关信息：

参数	类型	说明
type	RCConversationType	被拦截消息所在会话的会话类型
targetId	NSString	被拦截消息所在的会话 Id
channelId	NSString	被拦截消息所在的超级群频道 ID（iOS 端 SDK 从 5.2.4 版本开始支持）
blockedMsgUId	NSString	被拦截消息的唯一 Id
blockType	RCMessageBlockType	消息被拦截的原因，详见下方 `RCMessageBlockType` 说明
extra	NSString	被拦截消息的附加信息
sentTime	long long	被拦截消息的发送时间（Unix 时间戳、毫秒）。iOS 端 SDK 5.2.4 及以后版本支持。
sourceType	NSInteger	被拦截的超级群消息的源类型。`0`：原始消息触发了拦截（默认）。`1`：消息扩展触发了拦截。`2`：修改消息后的消息内容处罚了拦截。iOS 端 SDK 5.2.5 及以后版本支持（仅支持超级群）。
sourceContent	NSString	被拦截的超级群消息或扩展的内容 JSON 字符串。`sourceType` 字段为 1 时表示扩展内容。`sourceType` 为 2 时表示修改后的消息内容。详见下方 `sourceContent` 说明。iOS 端 SDK 5.2.5 及以后版本支持（仅支持超级群）。

- `RCMessageBlockType` 说明

```
typedef NS_ENUM(NSUInteger, RCMessageBlockType) {
/*!
全局敏感词：命中了融云内置的全局敏感词
*/
RCMessageBlockTypeGlobal = 1,

/*!
自定义敏感词拦截：命中了客户在融云自定义的敏感词
*/
RCMessageBlockTypeCustom = 2,

/*!
第三方审核拦截：命中了第三方（数美）或消息回调服务（原模板路由服务）决定不下发的状态
*/
RCMessageBlockTypeThirdParty= 3,
};
```

- `sourceContent` **说明**
  - `sourceType` 为 `0` 时，sourceContent 为 nil。
  - `sourceType` 为 `1` 时，sourceContent 是消息扩展内容，示例 {"mid":"xxx-xxx-xxx-xxx","put":{"key":"敏感词"}}。mid 为通知信息的 ID。
  - `sourceType` 为 `2` 时，sourceContent 是修改后的消息内容，示例 {"content":"含有敏感信息的文字"}。内置消息类型的消息内容格式消息类型概述。

设置敏感信息拦截代理委托

> 提示
>
> 该接口在 RCCoreClient 中。

```
[RCCoreClient sharedCoreClient].messageBlockDelegate = self;
```

IMLib 2.X 升级到 5.X IMLib 2.X 升级到 5.X

更新时间:2024-08-30

本文描述 IMLib SDK 从 2.X 到 5.X 版本的升级步骤。

接口变更说明

连接接口发生了变更，新版连接接口有如下两个：

连接接口 1

```
- (void)connectWithToken:(NSString *)token
dbOpened:(void (^)(RCDBErrorCode code))dbOpenedBlock
success:(void (^)(NSString *userId))successBlock
error:(void (^)(RCConnectErrorCode status))errorBlock
```

用户调用接口之后，如果因为网络原因暂时连接不上，SDK 会一直尝试重连，直到连接成功或者出现 SDK 无法处理的错误（如 token 非法，用户封禁等）。

连接接口 2

```
- (void)connectWithToken:(NSString *)token
timeLimit:(int)timeLimit
dbOpened:(void (^)(RCDBErrorCode code))dbOpenedBlock
success:(void (^)(NSString *userId))successBlock
error:(void (^)(RCConnectErrorCode status))errorBlock
```

用户调用接口之后，SDK 会在 timeLimit 秒内尝试连接，超过时间将会返回超时并停止连接，timeLimit <= 0 行为和没有 timeLimit 的接口一样。

错误回调处理

新增的连接接口，可自行重连的中间错误码将不再触发 onError 回调（如旧版的 RC_NET_CHANNEL_INVALID 等），只有发生 SDK 无法处理的错误才会触发 onError 回调。

跟旧版的另外一个区别是，onError 回调触发后，4.0.0 将不再重连。而旧版 SDK 在出现可自行重连的中间错误码触发 onError 回调后，还会进行自动重连。

连接错误信息

```
31004：Token 无效。
处理方案：从 APP 服务器获取新的 token ，再调用 connect 接口进行连接。

31010： 用户被踢下线。
处理法案：退回到登录页面，给用户提示被踢掉线。

31023：用户在其它设备上登录。
处理方案：退回到登录页面，给用户提示其他设备登录了当前账号。

31009：用户被封禁。
处理方案：退回到登录页面，给用户提示被封禁。

34006：自动重连超时(发生在 timeLimit 为有效值并且网络极差的情况下)。
处理方案：重新调用 connect 接口进行连接。

31008：Appkey 被封禁。
处理方案：请检查您使用的 AppKey 是否被封禁或已删除。

33001：SDK 没有初始化。
处理方案：在使用 SDK 任何功能之前，必须先 Init。

33003：开发者接口调用时传入的参数错误。
处理方案：请检查接口调用时传入的参数类型和值。

33002：数据库错误。
处理方案：检查用户 userId 是否包含特殊字符，SDK userId支持大小写英文字母与数字的组合，最大长度 64 字节。
```

连接错误解决方案

1. error 回调中判断是否是 token 非法和连接超时。
2. 连接状态回调中判断是否是连接状态是否是 token 非法、踢掉线、封禁、自动重连超时等情况，并按照上述处理方案处理。

```
//1.连接处理
[[RCIMClient sharedRCIMClient] connectWithToken:newToken
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
}
success:^(NSString *userId) {
//连接成功
}
error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else {
//无法连接到 IM 服务器，请根据相应的错误码作出对应处理
}
}]
//或者
[[RCIMClient sharedRCIMClient] connectWithToken:token
timeLimit:5
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
} success:^(NSString *userId) {
//连接成功
} error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else if(status == RC_CONNECT_TIMEOUT) {
//连接超时，弹出提示，可以引导用户等待网络正常的时候再次点击进行连接
} else {
//无法连接 IM 服务器，请根据相应的错误码作出对应处理
}
}]
```

由静态库修改为动态库

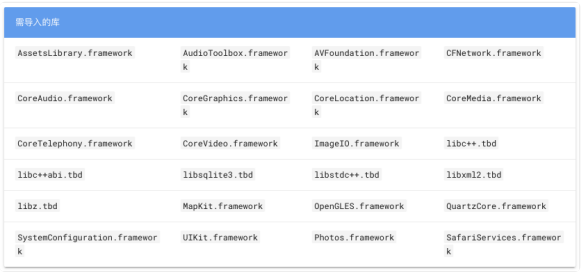
5.0.0 相较于旧版本，SDK 从静态库修改为动态库。

5.1.1 动态库改为 XCFramework 形式。 Cocoapod 版本最低为 1.10.0 ，否则可能会无法加载 SDK 或者报 ld: framework not found

RandomNames.xcframework

通过手动集成时需要修改:

1. 将原先 IMLib SDK 依赖的系统库全部去掉，如果有 App 或者其他 SDK 依赖的系统库除外



需导入的库			
AssetsLibrary.framework	AudioToolbox.framework	AVFoundation.framework	CFNetwork.framework
CoreAudio.framework	CoreGraphics.framework	CoreLocation.framework	CoreMedia.framework
CoreTelephony.framework	CoreVideo.framework	ImageIO.framework	libc++.tbd
libc++abi.tbd	libsqlite3.tbd	libstdc++.tbd	libxml2.tbd
libz.tbd	MapKit.framework	OpenGLES.framework	QuartzCore.framework
SystemConfiguration.framework	UIKit.framework	Photos.framework	SafariServices.framework

2. General -> Frameworks,Libraries,and Embedded Binaries 中将 RongXX.framework 的 Embed 由 Do Not Embed 改为 Embed & Sign

3. 引入的 IMLib 的特定头文件如 <RongIMLib/RCUserInfo.h> 修改，请参考常见问题 3。

4. 打包上架时遇到报错 IPA processing failed，请参考常见问题 4。

通过 pod 集成时需要修改

方式一、全量库导入：

```
pod 'RongCloudIM/IMLib', '5.0.0' # ChatRoom，Discussion、PublicService、CustomerService、Location、IMLibCore 的合集
```

方式二、根据需求对 SDK 选择性导入：

```
pod 'RongCloudIM/IMLibCore', '5.0.0' # 基础通讯库，必须

pod 'RongCloudIM/ChatRoom', '5.0.0' # 聊天室，可选
pod 'RongCloudIM/Discussion', '5.0.0' # 讨论组，可选
pod 'RongCloudIM/PublicService', '5.0.0' # 公众号，可选
pod 'RongCloudIM/CustomerService', '5.0.0' # 客服，可选
pod 'RongCloudIM/Location', '5.0.0' # 实时位置共享，可选
```

RongIMLib 拆分

为减小 IMLib 本身的尺寸，加载速度，性能等指标，对 RongIMLib 进行了拆分，目前对外提供的 IMLib SDK 有两种：

一种是 RongIMLib，包含了所有功能，接口使用方式不用修改

另外一种是 拆分后的 RongIMLibCore（基本通信能力库）、RongCustomerService（客服）、RongPublicService（公众号）、RongDiscussion（讨论组）、RongChatRoom（聊天室） 和 RongLocation（实时位置共享），可以根据需求进行选择

```
// 单例类使用方式需要由原先的 [RCIMClient sharedRCIMClient] 修改为如下：
RongIMLibCore：[RCCoreClient sharedCoreClient]
RongCustomerService：[RCCustomerServiceClient sharedCustomerServiceClient]
RongPublicService：[RCPublicServiceClient sharedPublicServiceClient]
RongDiscussion：[RCDiscussionClient sharedDiscussionClient]
RongChatRoom：[RCChatRoomClient sharedChatRoomClient]
RongLocation：无
```

旧版本快速兼容方案

说明：以下是旧版本快速兼容方案，但是我们依然建议您参考上面的详细建议进行处理；如果您是直接使用 5.0.0 新版本建议参见上面的详细处理流

程。

1. 关键适配点：删除连接接口 token 非法的回调，并将该回调中的处理逻辑，移动至 error 回调中。

```
[[RCIMClient sharedRCIMClient] connectWithToken:token
dbOpened:^(RCDBErrorCode code) {
//如果消息数据库打开，可以进入到主页面
}
success:^(NSString *userId) {
//连接成功
}
error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//将旧版本 token 无效的回调处理代码写到这里
//从 APP 服务获取新 token，并重连
} else {
//无法连接 im 服务器，请根据相应的错误码作出对应处理
}
}];
```

2. 关键适配点：SDK 引入方式修改。

   1. 将原先依赖的系统库全部去掉
   2. General -> Frameworks,Libraries,and Embedded Binaries 中将 RongXX.framework 的 Embed 由 Do Not Embed 改为 Embed & Sign

常见问题

1. 为什么一部分无法重连错误码的处理逻辑并没有在示例代码中写明？

都是开发阶段的问题，不需要代码兼容处理。

31008：Appkey 被封禁

当发生这个问题的时候，您可以自行在控制台查看您的 appkey 使用状态，大多情况是 appkey 被自行删除或者欠费。

33001：SDK 没有初始化

这个错误只会发生在开发阶段，只要您保证先 init 后 connect 就不会有这个问题。

33003：开发者接口调用时传入的参数错误

这个错误只会发生在开发阶段，很可能是传入的 token 为空，只要保证 connect 传入正确合法的 token 就不会有这个问题。

33002：数据库错误

这个问题只会发生在开发阶段，很可能是您的用户 id 体系和我们 SDK 的不一致，一般该情况很少发生。

2. 旧版本连接过程中一旦出现中间错误码就会立即触发 error 回调，新版本中间错误码不会触发 error 回调了，可能会等很长时间没有任何回调，这要怎么处理？

以下的建议，择一选取即可。

建议1.

用户第一次登录，设置 timeLimit 为有效值，网络极差情况下超时回调 error。

用户后续登录，调用没有 timeLimit 的接口，SDK 就会保持旧版本的自动重连。

建议2.

设置 SDK 的连接状态监听，APP 自行做超时的记录，如果超时了，APP 可以自动断开连接再调用 connect 进行连接。

3. 打包上架时报错：IPA processing failed

建议一：在打包上架时添加删除模拟器架构的脚本

1. 清空项目编译缓存：

   选择 Product -> Clean Build Folder，等待清空编译缓存

2. 剔除打包不支持的 x86_64 和 i386 架构：

   1. 选择 TARGETS -> Build Phases。
   2. 单击加号，选择 New Run Script Phase。
   3. 添加如下代码：

```
# 打包上架时需要添加此脚本

echo "------ extract architectures start -----------------"
APP_PATH="${`TARGET_BUILD_DIR`}/${`WRAPPER_NAME`}"

find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecutable)
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"

EXTRACTED_ARCHS=()

echo "----- FRAMEWORK_EXECUTABLE_NAME = $FRAMEWORK_EXECUTABLE_NAME ARCHS = $ARCHS -----"

for ARCH in $ARCHS
do
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o "$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done

lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${`EXTRACTED_ARCHS[@]`}"
rm "${`EXTRACTED_ARCHS[@]`}"

rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"
done
```

建议二：在打包上架时手动删除模拟器架构

1. 备份 SDK
2. 进入到需要删除模拟器架构的 IMLib.framework 目录中
3. 移除支持 x86_64，i386 的二进制文件

```
// 此处以 IMLib 举例

// 剔除模拟器架构
lipo RongIMLib.framework/RongIMLib -remove x86_64 -remove i386 -output RongIMLib

// 替换 framwork 内部二进制文件
mv RongIMLib RongIMLib.framework/RongIMLib

// 查看剥离后的二进制文件支持的 CPU 架构，如果显示 armv7 arm64，就可以重新打包了
lipo -info RongIMLib.framework/RongIMLib
```

4. 如 <RongIMLib/RCUserInfo.h> 等这种特定头文件的导入报错

有两种处理方案：

第一、<RongIMLib/RCUserInfo.h> 改成 <RongIMLib/RongIMLib.h>

第二、<RongIMLib/RCUserInfo.h> 改成 <RongIMLibCore/RCUserInfo.h>

被删除的废弃接口说明

IMLib 库 RCIMClient 类以下废弃接口均被删除

1. SDK 初始化

   升级说明: 从 2.4.1 版本开始，为了兼容 Swift 的风格与便于使用，将此方法升级为 `initWithAppKey:` 方法，方法的功能和使用均不变。

```
- (void)init:(NSString *)appKey __deprecated_msg("已废弃，请勿使用。");
```

2. 获取用户信息

   注意：已废弃，请勿使用

```
//RCIMClient
- (void)getUserInfo:(NSString *)userId
success:(void (^)(RCUserInfo *userInfo))successBlock
error:(void (^)(RCErrorCode status))errorBlock __deprecated_msg("已废弃，请勿使用。");
```

3. 插入消息

   升级说明：如果您之前使用了此接口，可以直接替换为 `insertOutgoingMessage:targetId:sentStatus:content:` 接口，行为和实现完全一致。

```
- (RCMessage *)insertMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
senderUserId:(NSString *)senderUserId
sendStatus:(RCSentStatus)sendStatus
content:(RCMessageContent *)content __deprecated_msg("已废弃，请勿使用。");
```

4. 多媒体消息下载

升级说明：如果您之前使用了此接口，可以直接替换为 downloadMediaFile:mediaUrl:progress:success:error:cancel: 接口行为和实现完全一致。

```
- (void)downloadMediaFile:(NSString *)fileName
mediaUrl:(NSString *)mediaUrl
progress:(void (^)(int progress))progressBlock
success:(void (^)(NSString *mediaPath))successBlock
error:(void (^)(RCErrorCode errorCode))errorBlock __deprecated_msg("已废弃，请勿使用。");
```

5. 多媒体消息下载

升级说明：如果您之前使用了此接口，可以直接替换为 downloadMediaFile:targetId:mediaType:mediaUrl:progress:success:error:cancel: 接口行为和实现完全一致。

```
- (void)downloadMediaFile:(RCConversationType)conversationType
targetId:(NSString *)targetId
mediaType:(RCMediaType)mediaType
mediaUrl:(NSString *)mediaUrl
progress:(void (^)(int progress))progressBlock
success:(void (^)(NSString *mediaPath))successBlock
error:(void (^)(RCErrorCode errorCode))errorBlock __deprecated_msg("已废弃，请勿使用。");
```

6. 消息发送

升级说明：如果您之前使用了此接口，可以直接替换为 sendMessage:targetId:content:pushContent:pushData:success:error: 接口（pushData传为 nil），行为和实现完全一致。

```
- (RCMessage *)sendMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode nErrorCode, long messageId))errorBlock
__deprecated_msg("已废弃，请勿使用。");
```

7. 多媒体消息发送

升级说明：如果您之前使用了下面两个接口，可以直接替换为 sendMediaMessage:targetId:content:pushContent:pushData:progress:success:error:cancel: 接口（pushData传为nil），行为和实现完全一致。

```
- (RCMessage *)sendImageMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
progress:(void (^)(int progress, long messageId))progressBlock
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode errorCode, long messageId))errorBlock
__deprecated_msg("已废弃，请勿使用。");
```

```
- (RCMessage *)sendImageMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
progress:(void (^)(int progress, long messageId))progressBlock
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode errorCode, long messageId))errorBlock
__deprecated_msg("已废弃，请勿使用。");
```

8. 上传图片到指定的服务器的消息发送接口

   升级说明：如果您之前使用了此接口，可以直接替换为 sendMediaMessage:targetId:content:pushContent:pushData:uploadPrepare:progress:success:error:cancel: 接口，行为和实现完全一致。

```
- (RCMessage *)sendImageMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
uploadPrepare:(void (^)(RCUploadImageStatusListener *uploadListener))uploadPrepareBlock
progress:(void (^)(int progress, long messageId))progressBlock
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode errorCode, long messageId))errorBlock
__deprecated_msg("已废弃，请勿使用。");
```

9. 全局屏蔽某个时间段的消息提醒

   升级说明：如果您之前使用了此接口，可以直接替换为 setNotificationQuietHours:spanMins:success:error: 接口，行为和实现完全一致。

```
- (void)setConversationNotificationQuietHours:(NSString *)startTime
spanMins:(int)spanMins
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock
__deprecated_msg("已废弃，请勿使用。");
```

10. 删除已设置的全局时间段消息提醒屏蔽

    升级说明：如果您之前使用了此接口，可以直接替换为 removeNotificationQuietHours:error: 接口，行为和实现完全一致。

```
- (void)removeConversationNotificationQuietHours:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock
__deprecated_msg("已废弃，请勿使用。");
```

11. 群组相关接口

    注意：已废弃，请勿使用

```
- (void)syncGroups:(NSArray *)groupList
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock __deprecated_msg("已废弃，请勿使用。");
```

```
- (void)joinGroup:(NSString *)groupId
groupName:(NSString *)groupName
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock __deprecated_msg("已废弃，请勿使用。");
```

```
- (void)quitGroup:(NSString *)groupId
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode status))errorBlock __deprecated_msg("已废弃，请勿使用。");
```

12. 评价人工客服

升级说明：如果您之前使用了此接口，可以直接替换为 `evaluateCustomerService:dialogId:starValue:suggest:resolveStatus:tagText:extra:` 接口，行为和实现完全一致。

```
- (void)evaluateCustomerService:(NSString *)kefuId
dialogId:(NSString *)dialogId
humanValue:(int)value
suggest:(NSString *)suggest __deprecated_msg("已废弃，请勿使用。");
```

IMLib 4.X 升级到 5.X IMLib 4.X 升级到 5.X

更新时间:2024-08-30

本文描述 IMLib SDK 从 4.X 到 5.X 版本的升级步骤。

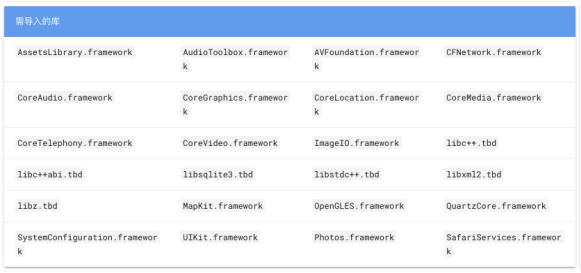
修改集成方式

5.0.0 相较于旧版本，从静态库修改为动态库

5.1.1 动态库改为 XCFramework 形式。 Cocoapod 版本最低为 1.10.0 ，否则可能会无法加载 SDK 或者报 ld: framework not found RandomNames.xcframework

手动集成

1. 将原先 IMLib SDK 依赖的系统库全部去掉，如果 App 或者其他 SDK 有依赖的系统库除外



2. General -> Frameworks,Libraries,and Embedded Binaries 中将 RongXX.framework 的 Embed 由 Do Not Embed 改为 Embed & Sign

3. 引入的 IMLib 的特定头文件如 <RongIMLib/RCUserInfo.h> 修改，请参考常见问题 3。

4. 打包上架时遇到报错 IPA processing failed，请参考常见问题 4。

pod 集成

方式一、全量库导入：

```
pod 'RongCloudIM/IMLib', '5.0.0' # ChatRoom，Discussion、PublicService、CustomerService、Location、IMLibCore 的合集
```

方式二、根据需求对 SDK 选择性导入：

```
pod 'RongCloudIM/IMLibCore', '5.0.0' # 基础通讯库，必须

pod 'RongCloudIM/ChatRoom', '5.0.0' # 聊天室，可选
pod 'RongCloudIM/Discussion', '5.0.0' # 讨论组，可选
pod 'RongCloudIM/PublicService', '5.0.0' # 公众号，可选
pod 'RongCloudIM/CustomerService', '5.0.0' # 客服，可选
pod 'RongCloudIM/Location', '5.0.0' # 实时位置共享，可选
```

RongIMLib 拆分

为减小 IMLib 本身的尺寸，加载速度，性能等指标，对 RongIMLib 进行了拆分，目前对外提供的 IMLib SDK 有两种：

一种是 RongIMLib，包含了所有功能，接口使用方式不用修改

另外一种是 拆分后的 RongIMLibCore（基本通信能力库）、RongCustomerService（客服）、RongPublicService（公众号）、RongDiscussion（讨论组）、RongChatRoom（聊天室） 和 RongLocation（实时位置共享），可以根据需求进行选择

```
// 单例类使用方式需要由原先的 [RCIMClient sharedRCIMClient] 修改为如下：
RongIMLibCore：[RCCoreClient sharedCoreClient]
RongCustomerService：[RCCustomerServiceClient sharedCustomerServiceClient]
RongPublicService：[RCPublicServiceClient sharedPublicServiceClient]
RongDiscussion：[RCDiscussionClient sharedDiscussionClient]
RongChatRoom：[RCChatRoomClient sharedChatRoomClient]
RongLocation：无
```

常见问题

1. 打包上架时报错：IPA processing failed

> 提示
>
> **建议一：升级到 5.1.1 及其以后版本**

5.1.1 开始动态库以 XCFramework 形式存在，在打包时候 Xcode 会自动选择使用真机架构，不需要再脚本或者手动进行动态库模拟器架构的移除

如果 APP 依赖其他的动态库，可能还需要按照下面的步骤处理其他的 SDK

> 提示
>
> **建议二：在打包上架时添加删除模拟器架构的脚本**

1. 清空项目编译缓存：

   选择 Product -> Clean Build Folder，等待清空编译缓存

2. 剔除打包不支持的 x86_64 和 i386 架构：

   1. 选择 TARGETS -> Build Phases。
   2. 单击加号，选择 New Run Script Phase。
   3. 添加如下代码：

```
# 打包上架时需要添加此脚本

echo "------ extract architectures start -----------------"
APP_PATH="${`TARGET_BUILD_DIR`}/${`WRAPPER_NAME`}"

find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecutable)
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"

EXTRACTED_ARCHS=()

echo "----- FRAMEWORK_EXECUTABLE_NAME = $FRAMEWORK_EXECUTABLE_NAME ARCHS = $ARCHS -----"

for ARCH in $ARCHS
do
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o "$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done

lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${`EXTRACTED_ARCHS[@]`}"
rm "${`EXTRACTED_ARCHS[@]`}"

rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"
done
```

> 提示
>
> **建议三：在打包上架时手动删除模拟器架构**

1. 备份 SDK

2. 进入到需要删除模拟器架构的 IMLib.framework 目录中

3. 移除支持 x86_64，i386 的二进制文件

```
// 此处以 IMLib 举例

// 剔除模拟器架构
lipo RongIMLib.framework/RongIMLib -remove x86_64 -remove i386 -output RongIMLib

// 替换 framwork 内部二进制文件
mv RongIMLib RongIMLib.framework/RongIMLib

// 查看剥离后的二进制文件支持的 CPU 架构，如果显示 armv7 arm64，就可以重新打包了
lipo -info RongIMLib.framework/RongIMLib
```

2. 如 <RongIMLib/RCUserInfo.h> 等这种特定头文件的导入报错

有两种处理方案：

第一、<RongIMLib/RCUserInfo.h> 改成 <RongIMLib/RongIMLib.h>

第二、<RongIMLib/RCUserInfo.h> 改成 <RongIMLibCore/RCUserInfo.h>

IMKit 2.X 升级到 5.X IMKit 2.X 升级到 5.X

更新时间:2024-08-30

本文描述 IMKit SDK 从 2.X 到 5.X 版本的升级步骤。

升级概述

从 2.x IMKit 升级到 5.x 版本，涉及到以下相关内容的变更，如果没用到以下内容可以平滑升级:

- RCIM
- RCMessageCell
- RCChatSessionInputBarControl
- RCPluginBoardView
- 会话设置页删除：RCSettingViewController 和RCConversationSettingTableViewController
- RCImagePreviewController
- RCKitUtility
- 资源图片变动
- 消息气泡拉伸比例修改
- 音视频资源图片迁移
- 注册自定义消息 cell 时机

以下是详细的升级说明。

1. RCIM 的接口

1.1 配置属性调用类变更

```
/// RCIM 的废弃接口，配置已被移动到 RCKitConfig 类中
@interface RCIM (Deprecated)

@property (nonatomic, assign) BOOL disableMessageNotificaiton __deprecated_msg("已废弃，请使用
RCKitConfigCenter.message.disableMessageNotificaiton");

@property (nonatomic, assign) BOOL disableMessageAlertSound __deprecated_msg("已废弃，请使用
RCKitConfigCenter.message.disableMessageAlertSound");

@property (nonatomic, assign) BOOL enableTypingStatus __deprecated_msg("已废弃，请使用
RCKitConfigCenter.message.enableTypingStatus");

@property (nonatomic, copy) NSArray *enabledReadReceiptConversationTypeList __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enabledReadReceiptConversationTypeList，设置开启回执的会话类型。");

@property (nonatomic, assign) NSUInteger maxReadRequestDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.maxReadRequestDuration");

@property (nonatomic, assign) BOOL enableSyncReadStatus __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableSyncReadStatus");

@property (nonatomic, assign) BOOL enableMessageMentioned __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableMessageMentioned");

@property (nonatomic, assign) BOOL enableMessageRecall __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableMessageRecall");

@property (nonatomic, assign) NSUInteger maxRecallDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.maxRecallDuration");

@property (nonatomic, assign) BOOL showUnkownMessage __deprecated_msg(
```

```
"已废弃，请使用 RCKitConfigCenter.message.showUnkownMessage");

@property (nonatomic, assign) BOOL showUnkownMessageNotificaiton __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.showUnkownMessageNotificaiton");

@property (nonatomic, assign) NSUInteger maxVoiceDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.maxVoiceDuration");

@property (nonatomic, assign) BOOL isExclusiveSoundPlayer __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.isExclusiveSoundPlayer");

@property (nonatomic, assign) BOOL isMediaSelectorContainVideo __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.isMediaSelectorContainVideo");

@property (nonatomic, assign) NSInteger GIFMsgAutoDownloadSize __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.GIFMsgAutoDownloadSize");

@property (nonatomic, assign) BOOL enableSendCombineMessage __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableSendCombineMessage");

@property (nonatomic, assign) BOOL enableBurnMessage __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableDestructMessage");

@property (nonatomic, assign) NSUInteger reeditDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.reeditDuration");

@property (nonatomic, assign) BOOL enableMessageReference __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableMessageReference");

@property (nonatomic, assign) NSUInteger sightRecordMaxDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.sightRecordMaxDuration");

@property (nonatomic, strong) UIColor *globalNavigationBarTintColor __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalNavigationBarTintColor");

@property (nonatomic, assign) RCUserAvatarStyle globalConversationAvatarStyle __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalConversationAvatarStyle");

@property (nonatomic, assign) CGSize globalConversationPortraitSize __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalConversationPortraitSize");

@property (nonatomic, assign) RCUserAvatarStyle globalMessageAvatarStyle __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalMessageAvatarStyle");

@property (nonatomic, assign) CGSize globalMessagePortraitSize __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalMessagePortraitSize");

@property (nonatomic, assign) CGFloat portraitImageViewCornerRadius __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.portraitImageViewCornerRadius");

@property (nonatomic, assign) BOOL enableDarkMode __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.enableDarkMode");
@end
```

1.2 连接接口变更

从 2.x 版本升级到 5.x， 关键适配点是删除连接接口 token 非法的回调，并将该回调中的处理逻辑，移动至 error 回调中，代码示例如下：

```
//1.连接处理
//用户调用接口之后，如果因为网络原因暂时连接不上，SDK 会一直尝试重连，直到连接成功或者出现 SDK 无法处理的错误（如 token 非法，用户封禁等）
[[RCIM sharedRCIM] connectWithToken:newToken
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
}
success:^(NSString *userId) {
//连接成功
}
error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else {
//无法连接到 IM 服务器，请根据相应的错误码作出对应处理
}
}]
//或者
//用户调用接口之后，SDK 会在 timeLimit 秒内尝试连接，超过时间将会返回超时并停止连接，timeLimit <= 0 行为和没有 timeLimit 的接口一样
[[RCIM sharedRCIM] connectWithToken:token
timeLimit:5
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
} success:^(NSString *userId) {
//连接成功
} error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//从 APP 服务获取新 token，并重连
} else if(status == RC_CONNECT_TIMEOUT) {
//连接超时，弹出提示，可以引导用户等待网络正常的时候再次点击进行连接
} else {
//无法连接 IM 服务器，请根据相应的错误码作出对应处理
}
}]

//2.连接状态监听设置

//其中 self 最好为单例类(如 AppDelegate)，以此保证在整个 APP 生命周期，该类都能够检测到 SDK 连接状态的变更
[[RCIM sharedRCIM] addConnectionStatusDelegate:self];

- (void)onRCIMConnectionStatusChanged:(RCConnectionStatus)status {
if (status == ConnectionStatus_KICKED_OFFLINE_BY_OTHER_CLIENT) {
//当前用户账号在其他端登录，请提示用户并做出对应处理
} else if (status == ConnectionStatus_DISCONN_EXCEPTION) {
//用户被封禁，请提示用户并做出对应处理
}
}
```

> 提示
>
> **新增的连接接口，可自行重连的中间错误码将不再触发 onError 回调（如旧版的 RC_NET_CHANNEL_INVALID 等），只有发生 SDK 无法处理的错误才会触发 onError 回调。**
> **跟旧版的另外一个区别是，onError 回调触发后，4.0.0 及之后版本将不再重连。而旧版 SDK 在出现可自行重连的中间错误码触发 onError 回调后，还会进行自动重连。**

1.3 收到已读回执通知

```
FOUNDATION_EXPORT NSString *const
RCKitDispatchReadReceiptNotification __deprecated_msg("已废弃，请使用RCLibDispatchReadReceiptNotification通知。");
```

1.4 图片消息发送接口废弃

升级说明：如果您之前使用了此接口，可以直接替换
为sendMediaMessage:targetId:content:pushContent:pushData:success:error:cancel:接口，行为和实现完全一致。

```
- (RCMessage *)sendImageMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
progress:(void (^)(int progress, long messageId))progressBlock
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode errorCode, long messageId))errorBlock
__deprecated_msg("已废弃，请使用sendMediaMessage函数。");
```

1.5 开启已读回执功能接口废弃

```
@property (nonatomic, assign) BOOL enableReadReceipt __deprecated_msg(
"已废弃，请使用enabledReadReceiptConversationTypeList，设置开启回执的会话类型。");
```

2. RCMessageCell

```
@property (nonatomic, strong) UIView *messageHasReadStatusView;
```

上面接口已废弃删除，可用原有的 statusContentView 替代

3. RCChatSessionInputBarControl

以下代理方法被删除：

```
- (void)imageDidSelect:(NSArray *)selectedImages fullImageRequired:(BOOL)full __deprecated_msg("已废弃，请勿使用。");
```

请用以下方法替代

```
- (void)imageDataDidSelect:(NSArray *)selectedImages fullImageRequired:(BOOL)full;
```

另外，RCChatSessionInputBarControl 类里的枚举均迁移到 RCExtensionKitDefine 类中。

4. RCPluginBoardView

插入或者更新扩展面板 button的方法均更改，具体是每个方法增加了一个 highlightedImage 参数，主要是给 PluginBoardItem 增加点击状态的图片

```
/*!
向扩展功能板中插入扩展项

@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
@param index 需要添加到的索引值
@param tag 扩展项的唯一标示符

@discussion 您以在RCConversationViewController的viewdidload后，添加自定义的扩展项。
SDK默认的扩展项的唯一标示符为1XXX，我们建议您在自定义扩展功能时不要选用1XXX，以免与SDK预留的扩展项唯一标示符重复。
*/
- (void)insertItem:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage title:(NSString *)title
atIndex:(NSInteger)index tag:(NSInteger)tag;

/*!
添加扩展项到扩展功能板，并在显示为最后一项

@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
@param tag 扩展项的唯一标示符

@discussion 您以在RCConversationViewController的viewdidload后，添加自定义的扩展项。
SDK默认的扩展项的唯一标示符为1XXX，我们建议您在自定义扩展功能时不要选用1XXX，以免与SDK预留的扩展项唯一标示符重复。
*/
- (void)insertItem:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage title:(NSString *)title
tag:(NSInteger)tag;

/*!
更新指定扩展项

@param index 扩展项的索引值
@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
*/
- (void)updateItemAtIndex:(NSInteger)index normalImage:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage
title:(NSString *)title;

/*!
更新指定扩展项

@param tag 扩展项的唯一标示符
@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
*/
- (void)updateItemWithTag:(NSInteger)tag normalImage:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage
title:(NSString *)title;
```

5. 设置页面类删除

RCSettingViewController 、RCConversationSettingTableViewController会话设置类已移除，如果需要，可以参考官网 SealTalk demo 源码中的 RCDSettingViewController，RCDPrivateSettingsTableViewController 及 RCDGroupSettingsTableViewController 来实现会话设置页

6. RCImagePreviewController

RCImagePreviewController 类（会话页面预览单个图片消息控制器）已移除，可以用 RCImageSlideController 类替代，具体实现参考

```
RCImageSlideController *imagePreviewVC = [[RCImageSlideController alloc] init];
imagePreviewVC.messageModel = model;
imagePreviewVC.onlyPreviewCurrentMessage = YES;//是否只预览当前图片消息，默认为 NO，支持当前会话图片消息滑动预览，如果设置为 YES， 只预览当前图片消息
UINavigationController *nav = [[UINavigationController alloc] initWithRootViewController:imagePreviewVC];
nav.modalPresentationStyle = UIModalPresentationFullScreen;
[self presentViewController:nav animated:YES completion:nil];
```

7. RCKitUtility

RCKitUtility 工具类中下面方法已移除，请使用 RCAlertView 中对应的方法

```
+ (void)showAlertController:(NSString *)title
message:(NSString *)message
preferredStyle:(UIAlertControllerStyle)style
actions:(NSArray<UIAlertAction *> *)actions
inViewController:(UIViewController *)controller;
```

8. 资源图片名称修改

会话页面输入栏加号扩展中的图片命名替换为 plugin_item 为前缀类型的命名;

输入框切换语音输入按钮表情按钮加号按钮的 icon 替换为 inputbar_xxx 类型的命名。

如果开发者之前直接替换过 SDK 上述资源图片，请注意修改对应图片名称

9. 消息气泡拉伸比例修改

```
原来的拉伸比例：
UIEdgeInsetsMake(image.size.height * 0.8, image.size.width * 0.2, image.size.height * 0.2, image.size.width * 0.8)

修改后的拉伸比例：
UIEdgeInsetsMake(image.size.height * 0.5, image.size.width * 0.5, image.size.height * 0.5, image.size.width * 0.5)
```

10. 音视频资源图片变更

RongIMKit 中不再包含 RongCallKit 库中所需的图片资源和语言文件，已转移到 RongCallKit 库中

11. 注册自定义消息 cell 时机

如果有自定义消息 Cell，在会话页面 viewDidload 方法中必须优先注册自定义 Cell，再做其他操作

IMKit 4.X 升级到 5.X IMKit 4.X 升级到 5.X 升级到 IMKit 5.4.0

更新时间:2024-08-30

在 IMKit 5.4.0 版本中，SDK 内部移除了对 RongIMLib 的依赖。

如果您从低版本升级至 IMKit 5.4.0，且项目中调用了 RCIMClient 类的方法，可能会出现报错。如果遇到问题，请将 RCIMClient 方法改为调用 RCCoreClient 的同名方法。

IMKit 的以下文档中曾使用 RCIMClient 核心类，现已更新为使用 RCCoreClient 核心类，并提供相关调用示例，供您参考：

- 实时位置共享
- 删除会话
- 会话置顶
- 会话未读数
- 删除消息
- 插入消息

例外情况：如果您在 IMKit 的项目中使用聊天室业务，请将调用 RCIMClient 下方法改为调用 RCChatRoomClient 下的同名方法。

从 2.X/4.X 版本升级到 IMKit 5.X 版本

从 2.x / 4.x 版本的 IMKit 升级到 5.x 版本，均涉及到以下相关内容的变更，如果没用到以下内容可以平滑升级:

- RCIM
- RCMessageCell
- RCChatSessionInputBarControl
- RCPluginBoardView
- 会话设置页删除：RCSettingViewController 和RCConversationSettingTableViewController
- RCImagePreviewController
- RCKitUtility
- 资源图片变动
- 消息气泡拉伸比例修改
- 音视频资源图片迁移
- 注册自定义消息 cell 时机

下面请看详细介绍：

1.RCIM 的接口

从 4.x 版本升级到 5.x， 需要关注配置属性调用类变更。

```
/// RCIM 的废弃接口，配置已被移动到 RCKitConfig 类中
@interface RCIM (Deprecated)

@property (nonatomic, assign) BOOL disableMessageNotificaiton __deprecated_msg("已废弃，请使用
RCKitConfigCenter.message.disableMessageNotificaiton");

@property (nonatomic, assign) BOOL disableMessageAlertSound __deprecated_msg("已废弃，请使用
RCKitConfigCenter.message.disableMessageAlertSound");

@property (nonatomic, assign) BOOL enableTypingStatus __deprecated_msg("已废弃，请使用
RCKitConfigCenter.message.enableTypingStatus");
```

```
@property (nonatomic, copy) NSArray *enabledReadReceiptConversationTypeList __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enabledReadReceiptConversationTypeList，设置开启回执的会话类型。");

@property (nonatomic, assign) NSUInteger maxReadRequestDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.maxReadRequestDuration");

@property (nonatomic, assign) BOOL enableSyncReadStatus __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableSyncReadStatus");

@property (nonatomic, assign) BOOL enableMessageMentioned __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableMessageMentioned");

@property (nonatomic, assign) BOOL enableMessageRecall __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableMessageRecall");

@property (nonatomic, assign) NSUInteger maxRecallDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.maxRecallDuration");

@property (nonatomic, assign) BOOL showUnkownMessage __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.showUnkownMessage");

@property (nonatomic, assign) BOOL showUnkownMessageNotificaiton __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.showUnkownMessageNotificaiton");

@property (nonatomic, assign) NSUInteger maxVoiceDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.maxVoiceDuration");

@property (nonatomic, assign) BOOL isExclusiveSoundPlayer __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.isExclusiveSoundPlayer");

@property (nonatomic, assign) BOOL isMediaSelectorContainVideo __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.isMediaSelectorContainVideo");

@property (nonatomic, assign) NSInteger GIFMsgAutoDownloadSize __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.GIFMsgAutoDownloadSize");

@property (nonatomic, assign) BOOL enableSendCombineMessage __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableSendCombineMessage");

@property (nonatomic, assign) BOOL enableBurnMessage __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableDestructMessage");

@property (nonatomic, assign) NSUInteger reeditDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.reeditDuration");

@property (nonatomic, assign) BOOL enableMessageReference __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.enableMessageReference");

@property (nonatomic, assign) NSUInteger sightRecordMaxDuration __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.message.sightRecordMaxDuration");

@property (nonatomic, strong) UIColor *globalNavigationBarTintColor __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalNavigationBarTintColor");

@property (nonatomic, assign) RCUserAvatarStyle globalConversationAvatarStyle __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalConversationAvatarStyle");

@property (nonatomic, assign) CGSize globalConversationPortraitSize __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalConversationPortraitSize");

@property (nonatomic, assign) RCUserAvatarStyle globalMessageAvatarStyle __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalMessageAvatarStyle");

@property (nonatomic, assign) CGSize globalMessagePortraitSize __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.globalMessagePortraitSize");

@property (nonatomic, assign) CGFloat portraitImageViewCornerRadius __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.portraitImageViewCornerRadius");

@property (nonatomic, assign) BOOL enableDarkMode __deprecated_msg(
"已废弃，请使用 RCKitConfigCenter.ui.enableDarkMode");
@end
```

2.RCMessageCell

```
@property (nonatomic, strong) UIView *messageHasReadStatusView;
```

上面接口已废弃删除，可用原有的 statusContentView 替代

3.RCChatSessionInputBarControl

3.1 方法删除

```
- (void)imageDidSelect:(NSArray *)selectedImages fullImageRequired:(BOOL)full __deprecated_msg("已废弃，请勿使用。");
```

上面代理方法删除，请用以下方法替代

```
- (void)imageDataDidSelect:(NSArray *)selectedImages fullImageRequired:(BOOL)full;
```

3.2 RCChatSessionInputBarControl 类里的枚举均迁移到 RCExtensionKitDefine 类中

4.RCPluginBoardView

插入或者更新扩展面板 button的方法均更改，具体是每个方法增加了一个 highlightedImage 参数，主要是给 PluginBoardItem 增加点击状态的图片

```
/*!
向扩展功能板中插入扩展项

@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
@param index 需要添加到的索引值
@param tag 扩展项的唯一标示符

@discussion 您以在RCConversationViewController的viewdidload后，添加自定义的扩展项。
SDK默认的扩展项的唯一标示符为1XXX，我们建议您在自定义扩展功能时不要选用1XXX，以免与SDK预留的扩展项唯一标示符重复。
*/
- (void)insertItem:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage title:(NSString *)title
atIndex:(NSInteger)index tag:(NSInteger)tag;

/*!
添加扩展项到扩展功能板，并在显示为最后一项

@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
@param tag 扩展项的唯一标示符

@discussion 您以在RCConversationViewController的viewdidload后，添加自定义的扩展项。
SDK默认的扩展项的唯一标示符为1XXX，我们建议您在自定义扩展功能时不要选用1XXX，以免与SDK预留的扩展项唯一标示符重复。
*/
- (void)insertItem:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage title:(NSString *)title
tag:(NSInteger)tag;

/*!
更新指定扩展项

@param index 扩展项的索引值
@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
*/
- (void)updateItemAtIndex:(NSInteger)index normalImage:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage
title:(NSString *)title;

/*!
更新指定扩展项

@param tag 扩展项的唯一标示符
@param normalImage 扩展项的展示图片
@param highlightedImage 扩展项的触摸高亮图片
@param title 扩展项的展示标题
*/
- (void)updateItemWithTag:(NSInteger)tag normalImage:(UIImage *)normalImage highlightedImage:(UIImage *)highlightedImage
title:(NSString *)title;
```

5. 设置页面类删除

RCSettingViewController 、RCConversationSettingTableViewController会话设置类已移除，如果需要，可以参考官网 SealTalk demo 源码中的 RCDSettingViewController，RCDPrivateSettingsTableViewController 及 RCDGroupSettingsTableViewController 来实现会话设置页

6. RCImagePreviewController

RCImagePreviewController 类（会话页面预览单个图片消息控制器）已移除，可以用 RCImageSlideController 类替代，具体实现参考

```
RCImageSlideController *imagePreviewVC = [[RCImageSlideController alloc] init];
imagePreviewVC.messageModel = model;
imagePreviewVC.onlyPreviewCurrentMessage = YES;//是否只预览当前图片消息，默认为 NO，支持当前会话图片消息滑动预览，如果设置为 YES， 只预览
当前图片消息
UINavigationController *nav = [[UINavigationController alloc] initWithRootViewController:imagePreviewVC];
nav.modalPresentationStyle = UIModalPresentationFullScreen;
[self presentViewController:nav animated:YES completion:nil];
```

7.RCKitUtility

RCKitUtility 工具类中下面方法已移除，请使用 RCAlertView 中对应的方法

```
+ (void)showAlertController:(NSString *)title
message:(NSString *)message
preferredStyle:(UIAlertControllerStyle)style
actions:(NSArray<UIAlertAction *> *)actions
inViewController:(UIViewController *)controller;
```

8. 资源图片名称修改

会话页面输入栏加号扩展中的图片命名替换为 plugin_item 为前缀类型的命名;

输入框切换语音输入按钮表情按钮加号按钮的 icon 替换为 inputbar_xxx 类型的命名。

如果开发者之前直接替换过 SDK 上述资源图片，请注意修改对应图片名称

9. 消息气泡拉伸比例修改

```
/// 原来的拉伸比例：
UIEdgeInsetsMake(image.size.height * 0.8, image.size.width * 0.2, image.size.height * 0.2, image.size.width * 0.8)

/// 修改后的拉伸比例：
UIEdgeInsetsMake(image.size.height * 0.5, image.size.width * 0.5, image.size.height * 0.5, image.size.width * 0.5)
```

10. 音视频资源图片变更

RongIMKit 中不再包含 RongCallKit 库中所需的图片资源和语言文件，已转移到 RongCallKit 库中

11.注册自定义消息 cell 时机

如果有自定义消息 Cell，在会话页面 viewDidload 方法中必须优先注册自定义 Cell，再做其他操作

IMKit 5.X 版本升级 IMKit 5.X 版本升级

更新时间:2024-08-30

本文描述 IMKit SDK 5.X 系列低版本升级到高版本的步骤。

升级到 IMKit 5.4.0

在 IMKit 5.4.0 版本中，SDK 内部移除了对 RongIMLib 的依赖。

如果您从低版本升级至 IMKit 5.4.0，且项目中调用了 RCIMClient 类的方法，可能会出现报错。如果遇到问题，请将 RCIMClient 方法改为调用 RCCoreClient 的同名方法。

IMKit 的以下文档中曾使用 RCIMClient 核心类，现已更新为使用 RCCoreClient 核心类，并提供相关调用示例，供您参考：

- 实时位置共享
- 删除会话
- 会话置顶
- 会话未读数
- 删除消息
- 插入消息

例外情况：如果您在 IMKit 的项目中使用聊天室业务，请将调用 RCIMClient 下方法改为调用 RCChatRoomClient 下的同名方法。

IM 翻译插件 IM 翻译插件

更新时间:2024-08-30

- IMLib 与 IMKit 从 5.2.2 版本开始支持翻译插件。
- 该插件暂仅适用于使用新加坡数据中心的应用。详见海外数据中心。

融云即时通讯业务提供翻译插件，可为 IMLib 与 IMKit SDK 快速接入外部翻译服务，由融云服务端负责对接外部翻译服务供应商的鉴权、API 调用、账号管理、计费等流程。翻译插件支持翻译文本。IMKit SDK 提供翻译 UI。

目前已支持接入 Google 翻译服务。

翻译流程

服务开通

该功能为付费增值服务。如有需求，请前往控制台 IM 翻译 页面开通服务。

关于 IM 翻译服务费用，详见 IM 翻译计费说明 。

客户端鉴权

客户端需要持有有效的 JWT Token，才能向融云请求翻译结果。

您的 App 服务端需要调用融云服务端 API 接口获取 JWT Token，然后返回给客户端。详见服务端文档获取 JWT Token。

提示

翻译插件鉴权专用的 JWT Token 不同于 IM 用户连接 IM 服务的 Token，请注意区分。

JWT

JWT 全称 JSON web Token，是为了在网络应用环境间传递声明而执行的一种基于 JSON 的开放标准。 JWT 包含 header、payload、signature 三部分。通过解析 payload 部分可获取到 Token 有效期和 UserId 等信息。

获取和刷新 JWT Token 流程图

集成翻译插件

翻译插件以 xframework 包方式提供。请首先在融云官网 下载 IMLib/IMKit SDK 。

1. 解压下载的 SDK 压缩包后，打开 IMLib 文件夹，将其中的 RongTranslation.xcframework 文件拷贝到项目目录下，并将其添加到项目中。
2. 在项目中，选择 Target > Frameworks and Libraries，找到 RongTranslation.xcframework。在Embed 列中选择 Embed and Sign，将其嵌入工程。
3. 在需要使用翻译功能的地方添加头文件：

```
#import <RongTranslation/RongTranslation.h>
```

IMKit 使用翻译插件

从 IMKit 5.2.2 版本开始，IMKit 集成了翻译功能。在开始使用翻译插件之前，请确认已开通翻译服务。

1. 使用 IMKit，需要提前设置默认源语言和目标语言。以下示例中我们设置简体中文为默认源语言，英语（英国）为默认目标语言。更多支持语言参见下文支持的语言类型。

```
// srcLanguage 源语言
// targetLanguage 目标语言
NSString *srcLanguage = @"zh_CN";
NSString *targetLanguage = @"en";
RCKitTranslationConfig *translationConfig = [[RCKitTranslationConfig alloc] initWithSrcLanguage:srcLanguage
targetLanguage:targetLanguage];
[RCKitConfig defaultConfig].message.translationConfig = translationConfig;
```

2. 在确认支持翻译服务之后，可以开始进行客户端鉴权。App 需要向自身的应用服务器发起请求，由应用服务器调用融云服务端 API 获取 JWT Token。App 获取 JWT Token 后，通过 `updateAuthToken` 接口设置到 SDK 中。

   示例代码

```
[[RCTranslationClient sharedInstance] updateAuthToken:@"YOUR_JWT_TOKEN"];
```

3. IMKit 只处理翻译结果，不会维护 JWT Token 的有效性，所以 App 需要单独调用 `addTranslationDelegate` 注册翻译结果回调。

   示例代码

```
// 添加监听，可多次添加
[[RCTranslationClient sharedInstance] addTranslationDelegate:delegate];

// 不需要时，可移除监听
[[RCTranslationClient sharedInstance] removeTranslationDelegate:delegate];
```

4. 在 JWT Token 失效时，重新获取并设置 JWT Token。

   示例代码

```
- (void)onTranslation:(RCTranslation *)translation
finishedWith:(RCTranslationCode)code {
if (code == RCTranslationCodeAuthFailed
|| code == RCTranslationCodeServerAuthFailed
|| code == RCTranslationCodeInvalidAuthToken) {
[self requestTranslationTokenBy:[RCIM sharedRCIM].currentUserInfo.userId];
}
}
```

   翻译结果会保存在 `RCTranslation` 对象中：

```
@interface RCTranslation : NSObject
/// 消息ID
@property (nonatomic, assign) NSInteger messageId;
/// 源语言
@property (nonatomic, copy) NSString *srcLanguage;
/// 目标语言
@property (nonatomic, copy) NSString *targetLanguage;
/// 原始文本
@property (nonatomic, copy) NSString *text;
/// 翻译文本
@property (nonatomic, copy) NSString *translationString;
```

IMLib 使用翻译插件

1. **在用户登录成功之后，需先判断当前是否开通翻译服务。**

   示例代码

   ```
   [[RCTranslationClient sharedInstance] isTextTranslationSupported];
   ```

2. **在确认支持翻译服务之后，可以开始进行客户端鉴权。**App **需要向自身的应用服务器发起请求，由应用服务器调用融云服务端** API **获取** JWT Token**。**App **获取** JWT Token **后，通过** `updateAuthToken` **接口设置到** SDK **中。**

   示例代码

   ```
   [[RCTranslationClient sharedInstance] updateAuthToken:@"YOUR_JWT_TOKEN"];
   ```

3. **通过** `addTranslationDelegate` **添加翻译结果回调。**

   示例代码

   ```
   // 添加监听，可多次添加
   [[RCTranslationClient sharedInstance] addTranslationDelegate:delegate];

   // 不需要时，可移除监听
   [[RCTranslationClient sharedInstance] removeTranslationDelegate:delegate];
   ```

4. **调用** `translate` **翻译文本。**

   示例代码

   ```
   [[RCTranslationClient sharedInstance] translate:messageId
   text:@"Text"
   srcLanguage:@"zh_CN"
   targetLanguage:@"en"];
   ```

   参数说明

参数名	类型	描述
messageId	Int	消息 ID。`messageId` 大于 0 时 SDK 会在本地缓存翻译结果（推荐）。如果无需缓存，可传入小于等于 0 的 `messageId`。
text	NSString	要翻译的文本。
srcLanguage	NSString	源语言类型。
targetLanguage	NSString	目标语言类型。

> 提示
>
> 接入 Google 翻译时，`srcLanguage` 可传任意值。Google 翻译服务会自动识别待翻译文本的源语言，并仅以识别结果为准。

5. 翻译的结果通过以下函数返回:

```
- (void)onTranslation:(RCTranslation *)translation
finishedWith:(RCTranslationCode)code {
...
显示翻译结果
}
```

翻译结果会保存在 `RCTranslation` 对象中：

```
@interface RCTranslation : NSObject
/// 消息ID
@property (nonatomic, assign) NSInteger messageId;
/// 源语言
@property (nonatomic, copy) NSString *srcLanguage;
/// 目标语言
@property (nonatomic, copy) NSString *targetLanguage;
/// 原始文本
@property (nonatomic, copy) NSString *text;
/// 翻译文本
@property (nonatomic, copy) NSString *translationString;
```

支持的语言类型

翻译插件支持的语言可参见文档中列出的语言列表。

Google 翻译服务

翻译插件已支持通过 Google Cloud Translation 服务翻译以下语言。更多细节，您可以直接参考 Google Cloud Translation 官方文档：语言列表 。

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
南非荷兰语	af
阿尔巴尼亚语	sq
阿姆哈拉语	am
阿拉伯语	ar
亚美尼亚文	hy
阿萨姆语	as

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
艾马拉语	ay
阿塞拜疆语	az
班巴拉语	bm
巴斯克语	eu
白俄罗斯语	be
孟加拉文	bn
博杰普尔语	bho
波斯尼亚语	bs
保加利亚语	bg
加泰罗尼亚语	ca
宿务语	ceb
中文（简体）	zh_CN 或 zh
中文（繁体）	zh_TW
科西嘉语	co
克罗地亚语	hr
捷克语	cs
丹麦语	da
迪维希语	dv
多格来语	doi
荷兰语	nl
英语	en
世界语	eo
爱沙尼亚语	et
埃维语	ee
菲律宾语（塔加拉语）	fil
芬兰语	fi
法语	fr
弗里斯兰语	fy
加利西亚语	gl
格鲁吉亚语	ka
德语	de
希腊文	el
瓜拉尼人	gn
古吉拉特文	gu
海地克里奥尔语	ht
豪萨语	ha
夏威夷语	haw
希伯来语	he 或 iw
印地语	hi
苗语	hmn
匈牙利语	hu
冰岛语	is
伊博语	ig
伊洛卡诺语	ilo
印度尼西亚语	id

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
爱尔兰语	ga
意大利语	it
日语	ja
爪哇语	jv 或 jw
卡纳达文	kn
哈萨克语	kk
高棉语	km
卢旺达语	rw
贡根语	gom
韩语	ko
克里奥尔语	kri
库尔德语	ku
库尔德语（索拉尼）	ckb
吉尔吉斯语	ky
老挝语	lo
拉丁文	la
拉脱维亚语	lv
林格拉语	ln
立陶宛语	lt
卢干达语	lg
卢森堡语	lb
马其顿语	mk
迈蒂利语	mai
马尔加什语	mg
马来语	ms
马拉雅拉姆文	ml
马耳他语	mt
毛利语	mi
马拉地语	mr
梅泰语（曼尼普尔语）	mni_Mtei
米佐语	lus
蒙古文	mn
缅甸语	my
尼泊尔语	ne
挪威语	no
尼杨扎语（齐切瓦语）	ny
奥里亚语（奥里亚）	or
奥罗莫语	om
普什图语	ps
波斯语	fa
波兰语	pl
葡萄牙语（葡萄牙、巴西）	pt
旁遮普语	pa
克丘亚语	qu
罗马尼亚语	ro

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
俄语	ru
萨摩亚语	sm
梵语	sa
苏格兰盖尔语	gd
塞佩蒂语	nso
塞尔维亚语	sr
塞索托语	st
修纳语	sn
信德语	sd
僧伽罗语	si
斯洛伐克语	sk
斯洛文尼亚语	sl
索马里语	so
西班牙语	es
巽他语	su
斯瓦希里语	sw
瑞典语	sv
塔加路语（菲律宾语）	tl
塔吉克语	tg
泰米尔语	ta
鞑靼语	tt
泰卢固语	te
泰语	th
蒂格尼亚语	ti
宗加语	ts
土耳其语	tr
土库曼语	tk
契维语（阿坎语）	ak
乌克兰语	uk
乌尔都语	ur
维吾尔语	ug
乌兹别克语	uz
越南语	vi
威尔士语	cy
班图语	xh
意第绪语	yi
约鲁巴语	yo
祖鲁语	zu

状态码

状态码	原因
26200	翻译成功
26201	翻译失败，融云鉴权失败 鉴权失败或者 token 过期

状态码	原因
26202	翻译失败，翻译功能服务商鉴权失败 融云服务器的原因，token 无效
26203	翻译失败，翻译功能服务商返回失败 具体服务商失败码信息
26204	翻译失败，翻译功能未在融云开启
26205	翻译失败，融云限流
26206	翻译失败，Server 没有鉴权 token 的 secret 需要在控制台开启
26208	短语音转录结果为空, 请核对编解码类型
26209	语言设置错误
26210	编码格式设置错误
34100	没有设置 authToken 或者 authToken 为 空串
34101	待翻译文本内容为空
34102	目标语言为空
34103	源语言为空
34104	翻译服务器地址为空
34105	app key 为空
34106	服务器返回数据无效

状态码 SDK 状态码

更新时间:2024-08-30

状态码	说明
-1000	开发者接口调用时传入的参数错误。
-1	未知状态，具体请【 提交工单 】。
-3	取消或暂停下载媒体文件失败。失败的原因一般为下载任务已经结束，或媒体消息不存在。
0	连接成功。
1	当前设备网络不可用。
2	当前设备切换到飞行模式。
3	当前设备切换到 2G（GPRS、EDGE）低速网络。
4	当前设备切换到 3G 或 4G 高速网s络。
5	当前设备切换到 WIFI 网络。
6	当前用户在其他设备上登录，此设备被踢下线。
7	当前用户在 Web 端登录。
8	服务器异常。
9	连接验证异常。
10	连接中。
11	连接失败或未连接。
12	当前用户已注销。
405	已被对方加入黑名单，消息发送失败。
407	未在对方的白名单中，消息发送失败。

状态码	说明
5004	超时。
20106	该用户处于单聊禁言状态，禁止发送单聊消息。
20604	发送消息频率过高，1 秒钟最多只允许发送 5 条消息，详细请联系商务，电话：13161856839。
20605	信令被封禁。如遇到该错误，请提交工单。
20607	调用超过频率限制，请稍后再试。
22201	消息扩展/修改，但是原始消息不存在。
22202	消息扩展/修改，但是原始消息不支持扩展。
22203	消息扩展/修改，扩展内容格式错误。
22204	消息扩展/修改，无操作权限。
22406	当前用户不在群组中。
22408	当前用户在群组中已被禁言。
23406	当前用户不在聊天室中。
23408	当前用户在聊天室中已被禁言。
23409	当前用户已被踢出并禁止加入聊天室。被禁止的时间取决于服务端调用踢出接口时传入的时间。
23410	聊天室不存在。
23411	聊天室成员超限，开发者可以提交工单申请聊天室人数限制变更。
23412	聊天室接口参数不正确。
23414	聊天室云存储业务未开通，如需开通请提交工单。
23423	聊天室的属性个数超限，单个聊天室默认上限为 100 个。
23424	没有权限修改聊天室中已存在的属性值。
23425	聊天室中属性设置频率超限，单个聊天室每秒上限 100 次。

状态码	说明
23426	未开通聊天室属性自定义设置。请在控制台免费基础功能页面开通聊天室属性自定义设置。
23427	聊天室属性值不存在。
23431	多端操作聊天室同一属性时属性设置失败。
24401	超级群功能没有开通。
24402	超级群服务异常。
24403	超级群参数错误。
24404	超级群未知异常。
24406	当前用户不在超级群中。
24408	当前用户在超级群中已被禁言。
24410	超级群不存在。
24411	超级群成员超限制。
24412	用户加入超级群数量超限。
24413	创建超级群频道，频道数超限。
24414	超级群频道 ID 不存在。
24416	用户不在超级群私有频道成员列表中
25101	撤回参数不正确。
25102	未开通历史消息云存储。请开通单群聊消息云端存储或聊天室消息云端存储服务。
25103	清除历史消息时，传递的时间戳大于当前系统时间。
26001	Push 参数不正确。
25105	清除历史消息时遇到内部异常，请提交工单确认原因。
25107	撤回他人消息失败。服务端可控制消息是否可由他人（非发送者本人）撤回。如果服务端已设置为仅限发送者本人撤回，则在撤回他人消息时报这个错误。

状态码	说明
26002	向服务端同步时出现问题，有可能是操作过于频繁所致。请稍后再试。
26101	没有在控制台开启小视频服务。
30001	连接已被释放。
30002	当前连接不可用。
30002	连接不可用。
30003	客户端发送消息请求，融云服务端响应超时。
30004	导航 HTTP 发送失败。
30007	导航 HTTP 请求失败。
30008	导航 HTTP 返回数据格式错误。
30010	创建 Socket 连接失败。
30011	Socket 断开。
30012	PING 失败。
30013	PING 超时。
30014	信令发送失败。
30015	连接过于频繁。
30016	消息大小超限，消息体最大 128 KB。
31000	连接 ACK 超时。
31002	初始化时填写的 AppKey 不正确，请在 控制台 获取。
31003	服务器当前不可用。在小程序端或桌面端未开通对应服务时也会返回该错误。
31004	Token 无效。
31005	App 校验未通过（开通了 App 校验功能，但是校验未通过）。

状态码	说明
31007	BundleID 不正确。
31008	AppKey 被封禁或已删除。
31009	连接失败，一般因为用户已被封禁。
31010	当前用户在其他移动设备上登录，此设备被踢下线。
31011	用户在线时被封禁导致连接断开。
31020	Token过期。一般是因为在控制台设置了token 过期时间，需要请求您的服务器重新获取 Token 并再次用新的 Token 建立连接。
31023	重连过程中当前用户在其它设备上登录。
31028	配置了 SDK 通过代理连接融云，但代理地址不可用。
32054	TCP 连接被重置，可能原因是运营商认为此链接非法或无效。SDK 会自动触发重连，App 侧无需处理。
32061	连接被拒绝。SDK 会自动重连，开发者无须处理。
33000	将消息存储到本地数据时失败。发送或插入消息时，消息需要存储到本地数据库，当存库失败时，会回调此错误码。
33001	未调用 SDK 的初始化方法。
33002	数据库错误。
33003	调用接口时传入的参数不正确。
33007	未开通历史消息云存储服务。可以在 基础功能 中开启服务。
34001	连接已存在，或正在重连中。
34002	小视频时间长度超出限制，默认小视频时长上限为 2 分钟。
34003	GIF 消息文件大小超出限制， 默认 GIF 文件大小上限是 2 MB。
34004	聊天室状态未同步完成，加入聊天室时立即调用获取聊天室属性接口，极限情况下会存在本地数据和服务器未同步完成的情况，开发者可以设置聊天室属性回调，SDK 同步完成时会在属性回调中通知开发者，开发者可根据回调状态进行获取。
34005	连接环境不正确。
34006	连接超时。当调用 connectWithToken:timeLimit:dbOpened:success:error: 接口，timeLimit 为有效值时，SDK 在 timeLimit 时间内还没连接成功返回此错误。

状态码	说明
34007	查询的公共服务信息不存在。
34008	消息不能被扩展。消息在发送时，RCMessage 对象的属性 canIncludeExpansion 置为 YES 才能进行扩展。
34009	消息扩展失败。一般是网络原因导致的，请确保网络状态良好，并且融云 SDK 连接正常。
34010	消息扩展大小超出限制， 默认消息扩展字典 key 长度不超过 32 个字符，value 长度不超过 4096 个字符（ SDK < 5.2.0 时，value 长度限制最大 64 个字符），设置的 Expansion 键值对不超过 300 个。
34011	媒体消息的媒体文件 http 上传失败。
34012	指定的会话类型不支持标签功能，会话标签仅支持单群聊会话、系统会话。
34013	批量处理指定标签的会话个数超限，批量处理会话个数最大为 1000。
34015	视频消息压缩失败。
34016	用户级别设置未开通。
34017	消息处理失败。一般是消息为 nil。
34018	媒体文件上传异常，媒体文件不存在或文件大小为 0。
34019	上传媒体文件格式不支持。
34020	文件已过期或被清理。如果 App Key 使用 **IM 旗舰版**或 **IM 尊享版**，文件存储时长默认为 180 天（不含小视频文件，小视频文件存储 7 天）。注意，**IM 商用版（已下线）默认存储 7 天。如需了解IM 旗舰版或 IM 尊享版**的具体功能与费用，请参见融云官方价格说明 页面及计费说明 。
34021	消息未注册。发送或者插入自定义消息之前，请确保注册了该类型的消息。
34022	该接口不支持超级群会话。
34023	超级群功能未开通。
34024	超级群频道不存在。
34210	传入的 `targetId` 非法。
34211	传入的 `channelId` 非法。
34212	传入的 `tagId` 非法。
34213	传入的 `tagName` 非法。

状态码	说明
34214	传入的 userId 非法。
34215	传入的 userIdList 非法。
34238	非法的代理配置。请检查代理是否为空或者是否传入了非法参数。
34239	传入的代理测试服务非法。
34240	代理地址或 testHost 地址无法连通。
34241	超级群撤回了不支持的消息类型，请开发者判断当前 RCMessageContent 类型是否支持被撤。
40006	RTC 房间操作时传入参数错误。

APNs 推送状态码

下面列举了在控制台，上传 APNs 推送证书后，推送测试过程中，可能遇上的错误码、原因以及需要进行的排查处理。

错误码	原因	处理。
5, 8	deviceToken 信息有误。	请检查设备是否越狱，setDeviceToken 传入的参数是否正确，Xcode 在打包时使用的provisioning profile 是否与当前环境匹配。
52	当前环境下 UserId 不存在，请确认一下是否连接服务器成功，init 使用的 App Key 是否是对应环境的 App Key。	connectWithToken 的 success Block 会回调当前登陆的 UserId，您可以查一下和您测试 Push 输入的是否一致。
53	您上传的推送证书为空。	检查并重新上传证书，并检查您填写的 BundleID 和证书中的 BundleID 是否对应)。
54, 62	您上传的证书或者证书密码有问题。	请检查您填写的证书密码，检查并重新上传证书，并检查您填写的 BundleID 和证书中的 BundleID 是否对应。
55	该 userId 当前未在 iOS 设备上登录。	请检查是否在 iOS 设备上成功登陆，是否存在多端登陆同一个 userId 的情况。
57	发送苹果 Push 失败。	SSL 解析异常，重新上传证书。
2, 56	该 userId 没有设置 deviceToken 信息。	请检查设备是否越狱，用户是否允许 App 进行通知，是否请求了远程推送权限以及 setDeviceToken 是否设置正确。
58	原始参数解析失败。	请联系我们尽快解决问题。
60	socket 异常。	内部错误，重新发送。
61	该 userId 设置了屏蔽推送。	请检查客户端是否屏蔽了推送功能（如果您有多个客户端或者 App 使用同一个 Appkey，请检查一遍所有的客户端代码是否设置了屏蔽推送）。
3, 4, 6, 7, 64	网络原因，推送失败，请您再次尝试。	网络原因，请重试。
0, 1, 10, 255，59	Apple APNS 服务器服务异常。	请过一段时间再尝试。
66	上传的证书与当前环境不匹配。	请检查并重新上传证书。
67	您上传证书的密码错误，请核对您的证书和密码。	请检查证书和密码，重新上传证书或填写密码。
68	开发者平台填写的Bundle ID和证书中的Bundle ID不匹配，请检查。	请检查填写的BundleID和证书Bundle ID。
70	VoIP Push Token 为空。	请确认 VoIP Push Token 是否为空。
73	没有用户信息，此用户 ID 没有获取过融云 Token。	重新获取用户 Token。
74	上传证书包名与配置包名不一致。	请检查上传证书包名与配置包名是否一致。
75	上传的证书包含的 BundleId 含有通配符。	按照苹果的要求，使用通配符的 App 无法使用APNs远程推送。
1050	APNS 返回超时，您可能延时收到推送消息，也可能推送失败。	网络原因，请重试。

更新日志（开发版）

更新日志（开发版）

更新时间:2024-08-30

> 提示
>
> 仅 Android/iOS 平台的 IM SDK 存在开发版、稳定版区分。开发版（Dev）SDK 首推新功能，同时会得到最快的 bug 修复。

5.10.1 Dev

发布日期：2024/07/02

问题修复

- 是否同步置顶空会话的开关默认值改为 NO，默认不同步置顶的空会话。

5.10.0 Dev

发布日期：2024/06/28

新增功能

- 新增用户信息托管功能，支持修改、查询、订阅托管的用户信息。
- 新增一个置顶空会话的开关配置，开发者可以选择是否同步置顶的空会话。

问题修复

- 修复了 IMKit 合并转发页面文件下载不能取消的问题。

5.8.2 Dev

发布日期：2024/06/05

新增功能

- 新增了 IMKit 会话页面消息全部拉取完的回调。
- 新增了批量获取会话信息的 API。
- 新增了用来控制会话置顶操作是否更新操作时间的 API。
- 新增了聊天室消息排重开关。

优化功能

- 优化了拖拽小视频播放进度条时，播放按钮的显示。
- 优化了合并转发消息的内容显示格式。

问题修复

- 修复了引用消息原文件已下载，但点击引用处的文件依然显示开始下载的问题。
- 修复了文件断点下载后可能出现的不完整的问题.

5.8.1 Dev

发布日期：2024/04/29

新增功能

- 支持了在指定会话中，对指定消息类型的历史消息，按关键字进行搜索的功能。
- 新增了客户端订阅用户在线状态订阅的功能。

优化功能

- 优化了消息撤回机制，现在撤回消息时会同时撤回命令消息中携带的用户信息（`UserInfo`）和额外信息（`extra`）。

5.8.0 Dev

发布日期：2024/03/29

新增功能

- IMLib 和 IMKit 增加了媒体上传下载拦截回调接口。
- 支持获取定向消息的目标用户列表。此功能仅适用于普通群和超级群消息。
- 新增了错误码 34296，针对发送定向消息，当会话类型不是群聊、超级群，且定向消息目标用户列表为空时，返回此错误。

问题修复

- 修复了 IMKit 发送消息携带 `senderUserInfo` 时，会覆盖接收方本地用户缓存中的 `alias` 的问题。
- 修复了会话列表置顶优先的问题.
- 修复了会话页表情切换至海豹后，返回会话列表再进入会话页表情仍是海豹的问题。

5.6.11 Dev

发布日期：2024/03/20

优化功能：

- 优化媒本消息内部处理逻辑

5.6.10 Dev

发布日期：2024/02/26

问题修复

- 修复 5.6.7、5.6.8、5.6.9 版本在特定场景下的重连错误问题。

5.6.9 Dev

发布日期：2024/01/31

新增功能：

- 提供包含 `PrivacyInfo.xcprivacy` 的 Framework。详见 关于 2024 春季 iOS 的隐私清单的通知。
- 超级群支持发送定向消息，可给指定频道中的指定用户发送消息，频道中其他用户不会收到该条消息。
- 超级群支持同时从本地和远端删除用户的历史消息。

优化功能：

- 设置会话置顶后置顶状态会同步给用户登录的其他设备。如果其他设备的本地会话列表中不存在该会话（尚未创建或已被删除），SDK 会自动创建该会话，并将其置顶。

- 获取会话列表时，支持通过参数指定返回结果忽略置顶状态，严格按照时间排序返回会话列表。

问题修复

- 修复调用 `AVAudioSession` 的 `setCategory` 与第三方冲突，导致录制语音消息失败的问题。

5.6.8 Dev

发布日期：2023/12/29

优化功能：

- 优化接收消息的状态处理。接收消息后，无论是否已被同时在线或之前登录的其他设备接收。只要其他设备先收到该消息，该状态值都会变为已接收。如果在其他设备已被阅读，同时还会变为已阅读。
- 新增设置消息接收状态、插入消息的 API，支持使用新的消息接收状态类型 `RCReceivedStatusInfo`。
- 会话（`RCConversation`）新增操作时间（`operationTime`）属性，可在分页获取会话列表时作为传入的时间戳。
- 新增支持携带消息配置（RCSendMessageOption 类型）的发送消息接口，暂仅支持控制是否使用 VOIP 推送。

问题修复：

- 修复 `onReceived` 和 `onOfflineMessageSyncCompleted` 时序错乱的问题。
- 修复 IMKit 列表页搜索有草稿消息的会话，进入会话页后定位错误的问题。
- 修复 IMKit 录入小视频后会内存泄露的问题。
- 修复 IMKit 多次下拉加载相册内容，点击选中视频，无法选中的问题。
- 修复 IMKit 更新到 5.6.7 以后，每次进入聊天界面都闪一下的问题。
- 修复 IMKit 用户发送的图片消息，对方接收空图片的问题。
- 修复 IMKit 发送动图超过限制（2 MB），在相册中选中时无提示的问题。
- 修复 IMKit 地图的定位页面在进入后台后重新回来时位置偏移的问题。
- 修复 RongLocationlib `NSMutableArray *delegateArray` 内存泄漏

5.6.7 Dev

发布日期：2023/11/23

新增功能：

- 消息推送属性中新增荣耀推送配置参数
- 支持 iOS Time Sensitive 时效性通知推送
- IMKit 支持复制引用消息中的被引用内容。
- IMLib 聊天室成员变更功能支持返回当前聊天室人数

优化功能：

- 优化 IMKit 进入相册后，最右边的图片复选框不易点击的问题。

问题修复：

- 修复 `decodeUserInfo` 时 SDK 崩溃的问题。
- 修复 `RCStickerDownloader` 下载超时，导致 `sessionDescription` 为空的问题。

5.6.6 Dev

发布日期：2023/10/27

- 新增暂停下载功能，支持断点续传。

优化功能：

- IMKit 中 `RCFilePreviewViewController` 的 forwardIcon@2x.png 增加黑色图标。
- IMKit 中移除发送小视频并上传到自定义 OSS 的场景下对小视频文件的时长上限限制。
- 新增崩溃收集开关 `crashMonitorEnable`，可关闭 SDK 捕获崩溃事件。
- 优化 SDK 连接逻辑。
- 优化 `getDeltaTime` 的逻辑，避免造成卡顿。
- 优化 SDK 日志上传机制。

问题修复：

- 修复接收消息线程阻塞的问题。
- 修复会话列表数组遍历异常的问题。
- 修复 IMKit `RCComplexTextMessageCell` 多次刷新后 cell 出现闪动的问题。
- 修复 IMKit 合并转发时因未应用程序未提供用户信息，导致应用闪退的问题。

5.6.4 Dev

发布日期：2023/09/25

新增功能：

- 超级群业务可以使用 `RCChannelClient` 下的 `getRemoteHistoryMessages` 方法获取远端历史消息。
- 超级群业务支持使用 getUltraGroupMessageCountByTimeRange 统计本地历史消息数量。

优化功能：

- 获取本地指定标签下的会话（`getConversationsFromTagByPage`）返回的 `RCConversation` 新增 `isTopForTag` 属性，用于标识会话在当前标签下是否已置顶。

问题修复：

- 修复 IMKit 在接收带图片的合并转发消息未下载时断开网络，点击查看图片消息未提示“图片加载失败”的问题。
- 修复 IMKit 中点击查看合并转发页面的小视频，右上角不显示进入到聊天文件页面的按钮的问题
- 修复 IMKit 首次安装未下载动态表情，断开网络后杀掉 App，再次启动 App 后不显示动态表情图标的问题。
- 修复 IMKit 聊天页面的多个小视频和文件消息布局显示紧凑的问题。
- 修复 `addConversationsToTag` 方法中传入超级群类型，应返回 34012，实际返回 33003 的问题。
- 修复偶发的 SDK 上传日志时报 400 错误的问题。

5.6.3 Dev

发布日期：2023/08/31

新增功能：

- IMLib 加入聊天室可返回聊天室当前状态（是否禁言、是否在禁言白名单中、聊天室人数等）信息。聊天室房间事件监听协议中新增对应的回调方法。
- IMKit 会话页面支持拦截点击常用语按钮的事件。

优化功能：

- IMKit 会话页面的长按删除消息功能改为默认同步删除远端历史消息。

- IMKit 优化为在被撤回的消息本地已不存在时，仍然插入小灰条消息。
- IMLib 移除断线重连后延后 2 秒再自动加入聊天室的行为。
- IMLib 优化删除单个会话所有消息耗时较长的问题

问题修复：

- 修复 IMKit 在录制界面中途关闭屏幕，再恢复录制，导致视频无声音的问题。
- 修复 IMKit 在引用回复显示用户名时偶现的崩溃问题。
- 修复 12小时制模式下, 全局免打扰不生效的问题。
- 修复首次安装时出现 `DB Error: 1 "no such table: CONVERSATION_INFO"` 报错的问题。

5.6.2 Dev

发布日期：2023/08/11

新增功能：

- IMKit/IMLib 支持多端同步系统会话阅读状态，新增错误码 20109。
- IMLib 超级群支持搜索本地数据库中指定用户 ID 发送的消息，支持通过关键词搜索所有频道的消息。

优化功能：

- 调整 SDK 重连时间间隔为 0.05s, 0.25s, 0.5s, 1s, 2s, 4s, 8s, 16s, 32s。之后每 64s 重试一次。

问题修复：

- 修复 IMKit 合并转发的消息过长（超过 4 行），没有...省略号展示的问题。
- 修复 IMKit 会话界面启用位置插件，点击位置插件，弹出的视图导致导航栏及状态栏变黑的问题。
- 修复 IMKit 多选按钮没有刷新出来的问题。
- 修复 IMKit 选择图片时展示列表时有滚动，相册列表页面空白的问题。
- 修复 IMKit 件消息发送检查内容错误未返回的问题。

5.6.1 Dev

发布日期：2023/07/14

优化功能：

- 优化 IMKit 单聊、群聊会话页面消息加载速度。
- 去掉 bundle 中的 1x 图片，改为 assets 方式管理资源。
- 加固了 IMKit SDK，防止极少数情况下非法字符导致的崩溃问题

问题修复：

- 修复阿拉伯语文本内容是左对齐的的错误。
- 修复 IMKit 会话页面放大历史图片再关闭时，错误触发消息列表滚动的问题
- 修复 IMKit 会话页面开启动态常用语后，右滑会话页面但不退出该页面，导致页面 UI 混乱的问题

5.6.0 Dev

发布日期：2023/07/03

新增功能：

- 超级群业务中，获取未读 @ 消息的摘要信息 `getUltraGroupUnreadMentionedDigests` 接口返回的 `RCMessageDigestInfo` 中新增消息

类型标识，可用于筛选数据。

问题修复：

- 修复未初始化进入会话页面 Crash 的问题
- 修复用户收取离线的扩展（KV）更新消息不全的问题。

5.4.8 Dev

发布日期：2023/06/25

问题修复：

- 修复 5.4.6、5.4.7 版本上超级群业务的回调方法 `onUltraGroupMessageExpansionUpdated` 返回的 `RCMessage` 的消息 ID 为 0 的问题。
- 修复 5.4.6、5.4.7 版本上 Xlog 库写入冲突的问题。

5.4.7 Dev

发布日期：2023/06/16

问题修复：

- 修复使用 CocoaPods 集成的客户无法使用模拟器 arm64 架构的问题

5.4.6 Dev

发布日期：2023/06/15

新增功能：

- 新增批量获取当前用户的超级群的未读消息数接口 `getUltraGroupConversationUnreadInfoList` 一次获取最多 20 个超级群下所有频道的未读数据。
- 新增 ARM64 模拟器支持

问题修复：

- 修复合并转发消息内邮箱地址不会识别的问题
- 修复调用超级群修改消息内容接口 `ModifyUltraGroupMessage` 后未更新搜索索引的问题
- 修复阿拉伯语下合并转发消息的内容展示方便问题

优化功能：

- 优化后台切换回前台时重连耗时长的问题
- 优化获取指定时间戳前或后消息接口，以实际传入时间戳为准，SDK 内部不做时间戳 +1 或-1 处理
- 优化的上传图片文件名的识别逻辑

5.4.5 Dev

发布日期：2023/05/29

新增功能：

1. 新增聊天室事件通知委托协议 `RCChatRoomNotifyEventDelegate`，支持在聊天室中执行成员封禁、禁言等操作时接收通知（封禁、禁言时需要指定 `needNotify` 为 `true`），支持在用户多端加入/退出接收通知。
2. 获取超级群获取频道列表时，支持通过 `RCConversation` 对象获取 "@我" 的未读消息数。
3. 超级群获取本地历史消息功能支持获取会话中指定时间戳前后、指定数量的消息。

优化功能：

1. 优化 IMKit 中默认发消息方法。优化完成后，在使用 `RCIMMessageInterceptor` 拦截消息时，可修改 `RCMessage` 所有属性，例如消息扩展，并由 SDK 继续发送。

问题修复：

1. 修复超级群未读消息数的问题。当前用户在连接状态下，超级群中有人撤回消息时（包括普通消息和 @消息），如消息在当前用户端为未读状态，未读消息及未读 @消息数没有修改。问题修复后，会对未读消息数做 -1 处理。
2. 修复超级群撤回消息小灰条提示重复的问题。超级群中撤回一条消息后，如本地没有找到原始消息，会插入小灰条消息。在特定情况下，可能出现小灰条消息重复的问题。问题修复后，小灰条消息会携带原始消息 ID，以进行排重。
3. 修复 通过 `unReadMentionedLabel` 自定义字体颜色无效的问题
4. 修复 SDK 的 xlog 与外部 xlog 日志写入冲突的问题

5.4.4 Dev

发布日期：2023/05/11

新增功能

1. 新增支持 VoIP 的 `sendMediaMessage` 接口

功能优化

1. 优化 IMKit SDK 对阿拉伯语言环境下 UI 布局的支持

问题修复：

1. 修复会话列表草稿没有立即刷新的问题
2. 移除[Text] UITextView 警告
3. 修复无法关闭控制台日志的问题
4. 修复加载图片导致内存暴涨的问题
5. 修复会话输入框上方的线重复出现的问题
6. 修复下载一个无效的文件路径，第一次会回调失败，再次下载该路径，不再回调成功或失败状态的问题。
7. 修复自定义消息未实现 `getSearchableWords` 方法输出警告的问题

5.4.2 Dev

发布日期：2023/04/20

新增功能

1. IMLib/IMKit SDK 支持在初始化配置 `InitOption` 中指定区域码。配置成功后，SDK 将使用与区域码对应的服务地址。
2. IMLib/IMKit SDK 支持在消息推送属性配置中指定 vivo 推送 category 参数。

优化功能

1. 优化 SDK 内置 IPluginModule 的稳定性
2. 为聊天室属性相关方法 `forceRemoveChatRoomEntry`、`removeChatRoomEntry`、`forceSetChatRoomEntry`、`setChatRoomEntry` 的 `notificationExtra` 字段增加长度校验

问题修复：

1. 修复拨打系统电话过程中录制语音消息未提示声音通道被占用的问题
2. 修复 iOS 15、16 的系统上，接听电话时进入 IMKit 会话里选择图片拍照或录小视频会卡在最后一帧的问题 。

3. 修复 iOS 阿拉伯语环境下，IMKit 聊天页面不展示对方的昵称的问题

5.4.1 Dev

发布日期：2023/03/31

优化功能

1. IMLib/IMKit SDK 新增支持 `RCInitOption` 的初始化接口
2. IMKit RCGroup 增加 extra 字段
3. IMLib SDK 获取远端历史消息数量上限提升至 100 条。
4. IMKit SDK 优化对阿拉伯语的支持

问题修复：

1. 修复 IMKit SDK 自定义 RCPluginBoardView 导致的崩溃问题
2. 修复 IMKit SDK 消息拦截后，再发送时携带的扩展信息丢失的问题
3. 修复偶现的下载文件失败的问题

5.4.0 Dev

发布日期：2023/03/03

新增功能

1. 发送消息时可在消息推送属性配置中设置华为推送通道的 Category 参数
2. IMLib SDK 新增超级群用户组功能

优化功能

1. **非兼容性更新：**IMKit SDK 内部剥离对 `RongIMLib` 库的依赖。请参照 *升级* **IMKit** 文档进行更新。
2. 优化 SDK 实时日志上传逻辑
3. IMKit SDK 聊天室会话下有新消息时不需要新消息铃声提醒

问题修复：

1. 修复 IMKit SDK 合并转发消息的头像变形的问题
2. 移除 [RCImageMessage getSearchableWords] 警告
3. 修复 IMLib SDK 在弱网情况下点击发送视频消息，偶现消息状态一直为发送中，关闭网络后也没有发送错误状态的回调的问题
4. 修复 IMLib SDK 当前在会话详情页面切到其它应用收到消息后，再切回后新收到的消息不会发送已读回执的问题
5. 修复 IMKit SDK PC 和移动端同时登录，PC 端已读被 @ 消息，但 iOS 端会话列表仍显示「有人@我」提示的问题
6. 修复 IMKit SDK 首次安装时权限选择「选中的照片」，导致图片选择页面向上拖拽滑动后会显示空白的问题

5.3.5 Dev

发布日期：2023/02/07

新增功能

1. IMLib SDK 本地批量插入消息接口支持将消息唯一标识 Message UID 存入数据库，支持针对 UID 进行排重
2. IMKit SDK 支持消息拦截协议，支持在文件上传前拦截媒体消息，并转发到 App 指定的服务器
3. IMKit SDK 新增发送媒体消息方法，支持将文件上传到 App 指定的服务器

优化功能

1. 将 RCCombineMessage.h 改为 public

问题修复：

1. 修复 IMKit SDK 批量发送携带用户信息消息，进入聊天页面出现卡顿的问题
2. 修复 IMKit SDK 合并消息的非媒体文件路径改变导致多次下载的问题
3. 修复 IMKit SDK 播放小视频过程中来系统电话，小视频停止播放但是没有显示暂停的按钮的问题
4. 修复 IMLib SDK 接收消息进行解码时，类型错误程序会崩溃的问题
5. 修复 IMKit SDK 长按消息点击更多选择按钮，查看底部高亮的转发和删除按钮比较暗的问题
6. 修复 IMKit SDK 停留在聊天页面，接收其他会话的撤回消息，查看左上角未读数字没有减 1 的问题
7. 修复 IMKit SDK 录制 1s 的小视频，发送后缩略图显示 1s, 播放页面显示 2s 的问题
8. 修复 IMKit SDK 快速预览勾选的图片，图库一些没有被勾选的图片也显示被勾选状态的问题
9. 修复 IMKit SDK 下载完引用的文件消息后再次打开重新下载的问题
10. 修复 IMKit SDK 在查看引用的文本消息时，对方撤回后未及时更新界面的问题
11. 修复 IMKit SDK 在群组中接收 10 条以上消息且第一条是 @ 消息时，发送端撤回第一条消息后，接收端未撤回的问题
12. 修复 IMKit SDK 发送保存的 GIF 图片，发送后显示的是图片而不是 Gif 的问题
13. 修复 IMKit SDK 下载引用消息中的文件消息时页面会崩溃的问题

5.3.4 Dev

发布日期：2023/01/10

新增功能

1. IMKit SDK 支持配置文件消息的文件图标
2. IMLib SDK 超级群支持搜索本地消息

优化功能

1. 优化 IMKit SDK 发送消息，发送已插入本地的消息同时更新内容、扩展、状态
2. 优化 IMLib SDK 接收消息，在 IMLibCore 中添加了禁用消息排重机制的开关

问题修复：

1. 修复 SDK 内部 RCloudCache 异常导致的崩溃问题
2. 修复 SDK 5.3.2 / 5.3.3 版本调用 `removeChatRoomEntry` 后，其他人错误地触发 `chatRoomKVDidUpdate` 回调的问题。修复后，其他人正常触发 `chatRoomKVDidRemove` 回调。

5.3.3 Dev

发布日期：2022/12/22

问题修复：

1. 修复 IMKit SDK 在撤回消息设置了 isDelete 为 true 时仍显示小灰条提示的问题
2. 修复 IMLib SDK 超级群消息更新扩展时无回调的问题
3. 修复 RCStickerDataManager 多线程调用 crash 的问题。
4. 修复 IMKit SDK 在同一用户账号在多 iOS 设备登录时会话未读数不一致的问题。具体表现为在一台设备上打开会话页面，此时另一台设备的会话列表页面仍显示该会话有未读数。
5. 修复 IMKit SDK 在图片选择页面勾选原图后，左右滑动预览其他的图片时画面闪烁的问题。
6. 修复 IMKit SDK 合并转发的消息在部分场景下无法在 Android 端无法打开的问题。

5.3.2 Dev

发布日期：2022/12/02

新增功能：

1. IMLib SDK 支持获取指定类型的所有未读会话的列表 `getUnreadConversationList`，支持单聊、群聊、系统会话
2. IMKit SDK 支持隐藏输入区表情按钮

优化功能

1. IMKit SDK 优化表情区域功能，禁用内置 emoji 表情后，表情面板不展示分页

问题修复：

1. 修复群组内发送文本消息引发的 App 崩溃问题
2. 修复因文件不存在引起的崩溃问题。
3. 修复 App 打包警告问题（App Store Connect Operation Error）
4. 修复卸载重装后偶现群聊已读回执列表不准的问题
5. 修复 IMLib 超级群业务在多端同步已读消息时间戳时，未清除第一条未读消息时间戳（`firstUnreadMsgSendTime`）的问题。问题修复后，多端同步阅读状态时 `firstUnreadMsgSendTime` 会被置为 0。

5.3.1 Dev

发布日期：2022/11/18

新增功能：

1. IMLib SDK 加入聊天室后，断网重连场景下，重新加入聊天室成功后获取聊天室消息条数与断网前加入聊天室获取的消息条数一致
2. IMLib SDK 接收消息中同时提及（@）所有人和提及部分人时，支持获取 @ 部分人列表

问题修复：

1. 修复特定版本 SDK 升级导致的发送语音和图片消息失败的问题
2. 修复 IMKit SDK 设置 `RCMessageBaseCell` 自定义多选属性无效问题
3. 修复用户被踢下线，切换用户登录后，会话列表消息显示异常，展示暂不支持查看此消息的问题
4. 修复设置文本消息超链接颜色失效问题

5.3.0 Dev

发布日期：2022/11/04

新增功能：

1. IMLib SDK 的 RCCoreClient 和 RCChannelClient 下增加异步数据库接口，已有同步接口废弃
2. IMKit SDK 支持在聊天页面中隐藏消息上的头像

功能优化：

1. IMLib SDK 移除客户端对加入聊天室消息个数最大值的限制
2. IMKit 优化阿拉伯语适配
3. IMLib SDK 超级群撤回消息时禁止撤回不支持的消息类型，新增错误码 34241

问题修复：

1. IMKit SDK 修复 xcode14 打包警告
2. IMLib SDK 修复断网时发送消息再删除消息，重连后发送成功问题

3. IMKit SDK 修复图片发送失败后，重发时 remoteUrl 被赋值为本地路径的问题

5.2.5 Dev

发布日期：2022/09/09

新增功能：

1. IMLib SDK按会话免打扰级别，获取未读消息数
2. IMLib SDK多端会话状态同步支持返回对应的免打扰级别
3. IMLib SDK超级群撤回消息时，即时本地不存在原始消息，自动插入一条撤回小灰条消息
4. IMLib SDK超级群获取未读 @消息列表
5. IMLib SDK含敏感词消息回调信息中增加 sourceType，sourceContent 字段，只针对超级群会话
6. IMLib SDK超级群 `getUnreadMentionedMessages` 支持传入消息数量，拉取顺序参数
7. IMLib SDK聊天室房间状态监听支持多代理
8. IMLib SDK 支持发送群组定向媒体消息
9. IMKit SDK 新增 `RCComplexTextMessageCell`，为长文本消息异步绘制 Cell
10. IMKit 默认使用高清语音消息

问题修复：

1. 修复 IMLib SDK 更新消息扩展后 successBlock 中无法拿到更新的扩展的问题
2. 修复 IMLib SDK 发送媒体消息 如文件、视频、语音、GIF时，errorBlock 回调多次的问题
3. 修复 IMKit SDK 撤回消息后，点击重新编辑按钮，placeholderlabel 没有处理的问题
4. 修复 IMKit SDK 会话列表页面头像大小，昵称 label 位置不对的问题
5. 修复 IMKit SDK 最后一条未读是普通或高清语音消息时，列表页面上的显示问题

5.2.4 Dev

发布日期：2022/07/22

新增功能：

1. IMLib SDK 含敏感词消息回调信息中增加频道 ID 字段，只针对超级群会话。
2. IMLib SDK 超级群会话支持了私有频道功能。通过 Server API 创建私有频道并设置私有频道成员列表。只有在私有频道成员列表中的用户可以在私有频道中收发消息。
3. IMLib SDK 接口规范化，方便 Swift 调用。对于在 5.2.4 之前已使用 Swift 的开发者，可能会遇到部分接口编译失败或者报警。请参见下方注意事项。
4. IMKit SDK 增加自定义消息注册入口 `- (void)registerCustomCellsAndMessages`。
5. IMLib SDK 增加 25107 错误码。服务端可控制消息是否支持他人（非发送者本人）撤回。如果服务端已设置为仅限发送者本人撤回，则在他人尝试撤回消息时报这个错误。

问题修复：

1. 修复 IMLib SDK 偶现会话未读数为 -1 的情况。
2. 修复 IMLib SDK `onOfflineMessageSyncCompleted` 和 `onReceived` 接收离线消息存在时序的问题。
3. 修复 IMLib SDK 超级群调用接口上传媒体消息到自己服务器时，消息发送状态 `sentStatus` 为 `SentStatus_FAILED` 的问题。
4. 修复 IMKit SDK 会话页面 `defaultLocalHistoryMessageCount` 的值影响下拉刷新的问题。
5. 修复 IMKit SDK【群标识】会话列表滑动时，带有标识的群，标识与群名称间距变宽的问题。
6. 修复 IMKit SDK 选择联系人页面群组和单聊头像昵称显示问题。
7. 修复小视频插件（Sight SDK）录制视频失败导致的 UI 错乱的问题。

注意事项

对于在 5.2.4 之前已使用 Swift 的开发者，需要根据规范化要求修改代码，否则可能会遇到部分接口编译失败或者报警。以下提供两个修改示例，以供参考：

- **获取聊天室信息**
  - SDK < 5.2.4

```
RCIMClient.shared().getChatRoomInfo("", count: 1, order: .chatRoom_Member_Asc) { (roomInfo: RCChatRoomInfo?) in
    let targetId = roomInfo?.targetId
} error: { (errorCode: RCErrorCode) in
    print("error code: \(errorCode.rawValue)")
}
```

  - SDK >= 5.2.4

```
RCIMClient.shared().getChatRoomInfo("", count: 1, order: .chatRoom_Member_Asc) { (roomInfo: RCChatRoomInfo) in
    let targetId = roomInfo.targetId
} error: { (errorCode: RCErrorCode) in
    print("error code: \(errorCode.rawValue)")
}
```

- **获取消息**
  - SDK < 5.2.4

```
let message = RCIMClient.shared().getMessage(0)
if nil != message {
    let targetId = message!.targetId
}
```

  - SDK >= 5.2.4

```
let message = RCIMClient.shared().getMessage(0)
if let msg = message {
    let targetId = msg.targetId
}
```

5.2.3.1 Dev

发布日期：2022/06/21

问题修复：

1. 1. **修复 IMLib SDK 图片消息** `+ (instancetype)messageWithImageURI:(NSString *)imageURI` **构造时** remoteUrl **和** localUrl **混用的问题**

5.2.3 Dev

发布日期：2022/06/08

新增功能：

1. IMKit SDK 支持关闭本地通知的提示音或震动。
2. IMKit SDK 支持关闭表情面板中内置的 Emoji表情。
3. IMKit SDK 不再内置地图模块，并推出新 `locationKit` 插件。注意：旧版 SDK 升级后，原有地图功能即失效，请集成地图插件。
4. IMKit 支持修改会话页面删除消息操作的默认行为。支持配置为在删除消息时同时删除本地与服务端的消息。
5. IMKit 增加一种会话列表聚合会话头像集成方式，
6. IMLib SDK 超级群消息撤回支持消息时支持通过设置 `isDelete` 参数同时删除发送端与接收端的原始消息数据。

功能优化：

1. 录制小视频消息时，其他会话有新消息时默认不提醒。

问题修复：

1. 修复了若干 BUG。

5.2.2 Dev

发布日期：2022/05/05

新增功能：

1. 增加了新的全局免打扰功能接口 setNotificationQuietHoursLevel，原接口 setNotificationQuietHours 废弃仍然可以正常使用。
2. 新增了会话免打扰枚举 RCPushNotificationLevel，设置项包括：所有消息都通知、未设置（默认为所有消息都通知）、@消息通知、@指定用户通知、@所有人通知、所有消息都不通知，其中未设置、@消息通知设置项为老版本支持逻辑可兼容老版本，其他设置项需要升级到此版本后才能支持
3. 针对超级群会话增加了默认免打扰状态设置接口 setUltraGroupConversationDefaultNotificationLevel
4. 针对超级群会话增加了查询免打扰默认状态接口 getUltraGroupConversationDefaultNotificationLevel
5. 针对超级群下指定频道增加了默认免打扰状态设置接口 setUltraGroupConversationChannelDefaultNotificationLevel
6. 针对超级群下指定频道增加了查询免打扰默认状态接口 getUltraGroupConversationChannelDefaultNotificationLevel
7. 增加了连接 IM SDK 后超级群会话信息同步完成的回调功能 setUltraGroupConversationDelegate
8. 增加了获取指定超级群下所有频道的未读消息总数接口 getUltraGroupUnreadCount
9. 增加了获取超级群会话类型的所有未读消息数接口 getUltraGroupAllUnreadCount
10. 增加了获取超级群会话类型的@消息未读数接口 getUltraGroupAllUnreadMentionedCount

问题修复：

1. 修复了若干 BUG

5.2.1 Dev

发布日期：2022/03/25

新增功能：

1. 超级群消息结构中增加消息已被修改标识。
2. 调整了超级群获取服务端历史消息接口 getMessages 获取历史消息条数的上限。调整后最多可获取 100 条。
3. 增加 IM 聊天室与 RTC 音视频房间绑定接口。创建绑定关系后，如果 RTC 房间仍存在，则服务端会阻止 IM 聊天室房间自动销毁。

问题修复：

1. 修复了超级群撤回消息的回调结果中 operatorId（撤回该条消息的操作用户）可能出现错误的问题。
2. 修复了敏感词回调不带 channelid 的问题。

5.2.0 Dev

发布日期：2022/03/01

新增功能：

1. 新增了融云超级群会话，支持无成员上限的群组聊天
2. 新增了超级群频道功能，可在超级群会话下创建多个频道，成员可随意在不同群频道中发送消息，但不同频道间的消息相互隔离。
3. 消息扩展功能，可设置的 Value 值长度改为 4096 个字符。

5.1.8 Dev

发布日期：2022/01/20

新增功能：

1. 针对多端操作聊天室同一属性时，偶现属性设置失败的问题，增加单独错误码 23431
2. 不再支持 App 设置消息状态为发送中，保证 SDK 的发送中的状态会被重置为发送失败

问题修复：

1. 修复了接收消息及聊天室成员变化代理设置问题
2. 修复了 IMKit SDK 若干 BUG

5.1.7 Dev

发布日期：2021/12/14

新增功能：

1. 针对小米、华为推送通道，在发送单条消息时，可设置推送时通知栏右侧显示的图片内容
2. 针对 FCM 推送通道通知消息方式，支持设置推送标题、通知栏右侧图片内容及推送 ChannelId

问题修复：

1. 修复了发送消息时如消息未注册，报错不准确的问题（如果自定义消息未注册，SDK 将无法识别）。修复后，如果 SDK 发现消息未注册，将抛出错误码 34021
2. 修复了若干 BUG

5.1.6 Dev

发布日期：2021/11/05

问题修复：

1. IMKit SDK 废弃原重发消息接口 resendMessage，新增对外接口 resendMessageWithModel
2. IMKit SDK 废弃原发送消息回调接口 didSendMessage，新增对外接口 didSendMessageModel
3. 修复了 SDK 中若干 BUG

5.1.5 Dev

发布日期：2021/09/24

新增功能：

1. 新增了清除标签对应会话的未读数接口 clearMessagesUnreadStatusByTag

2. 新增删除标签对应的会话接口 clearConversationsByTag
3. 新增获取会话的置顶状态接口 getConversationTopStatus
4. 新增获取某个会话内的指定消息类型未读消息数接口 getUnreadCount
5. 增加了对 iOS 15 的适配

问题修复：

1. 优化了媒体文件下载进度优化
2. 修复了小视频消息发送中点击查看，消息发送成功后，提示不正确的问题

5.1.4 Dev

发布日期：2021/08/11

新增功能：

1. 新增了批量设置和删除聊天室属性能力
2. 新增了用户未加入聊天室时，支持获取聊天室属性信息
3. 新增了用户加入、退出聊天室回调能力，需要客户开通后支持，可提交工单申请开通

问题修复：

1. 修复了播放小视频时，快速左右滑动切换视频，小视频没有暂停播放的问题

5.1.3 Dev

发布日期：2021/06/25

新增功能：

1. 发送单条消息时，针对华为推送通道，支持设置 LOW、NORMAL 级别消息
2. 优化引用消息结构，增加被引用消息 ID 属性 ReferMsgUid
3. IMKit 会话页面输入框中增加默认提示内容

问题修复：

1. 修复了通过 iCloud 视频下载失败视频发送异常的问题
2. 修复了某些图片缩略图有白边的问题
3. 修复了撤回消息时出现的 objName 和 msgContent 不一致的情况
4. 优化了 IMKit SDK 中小视频横屏显示的逻辑处理

5.1.2 Dev

发布日期：2021/05/21

新增功能：

1. IMLib SDK 支持了媒体消息中文件分片下载功能
2. IMKit SDK 小视频录制实现了防抖优化
3. IMLib SDK 增加了按时间搜索本地会话中历史消息功能

问题修复：

1. 修复了某些指定文本超链接高亮没显示的问题
2. 修复了 IMKit SDK 共享相册视频压缩失败的问题

3. 修复了 IMKit SDK 视频预览页面下载失败内存未释放的问题
4. 修复了偶现的发送视频崩溃的问题

5.1.1 Dev

发布日期：2021/04/09

新增功能：

1. 新增了会话标签设置功能
2. 新增了批量导入本地消息数据接口
3. 群会话中有 @我消息时，进入群会话界面支持点击跳转到 @ 消息功能
4. IMLib SDK 新增了图片缩略图尺寸设置能力
5. SDK 输出日志优化为全英文提示
6. SDK 改为 XCFramework，打包不需要额外删除 IM SDK 模拟器架构。 Cocoapod 版本最低为 1.10.0 ，否则可能会无法加载 SDK 或者报 ld: framework not found RandomNames.xcframework

问题修复：

1. 发送横屏的小视频消息，压缩后保存系统相册显示异常问题

5.1.0 Dev

发布日期：2021/03/05

问题修复：

1. SDK 中日志信息修改成英文提示

5.0.0 Dev

发布日期：2021/01/19

新增功能：

1. 发送消息，支持设置推送模板 ID，模板 ID 及模板中在“控制台-自定义推送文案”中进行设置。设置后根据目标用户通过 RCPushProfile 中的 setPushLauguageCode 设置的语言环境，匹配模板中设置的语言内容进行推送，未匹配成功时使用融云默认内容进行推送
2. 消息撤回功能支持推送属性设置 RCMessagePushConfig 和 isDisableNotification
3. 发送消息时支持设置，通知展示时是否显示 Title 功能

功能优化：

1. 对 IMKit SDK 进行了重构，提升了 UI 品质及用户体验
2. IMKit SDK UI 界面适配了阿拉伯语
3. IMLib SDK 按模块进行代码拆分，提升了初始化速度，减少了不必要的内存占用
4. SDK 由静态库变为动态库

问题修复：

1. 因系统会话不能回复，所以针对系统会话页面去掉内容输入框
2. 修复了输入框输入多行文本消息后，长按点击选择按钮，拖动光标不能向上滑动选中的问题
3. 修复了无论是否设置 forceKeepAlive，都监听系统后台任务，在挂起 APP 的时候主动释放资源并置为 suspend，方便再打开 APP 重连
4. 修改了输入框表情面板 view 被多次重复添加的问题
5. 修复了应用接收消息播放铃声引起后台音乐停止播放的问题
6. 修复了连接时 nil 转 char * 崩溃的问题

7. 修复了断网后发送小视频和文件消息，再次连接网络后依然发送失败的问题
8. 弱网下接收多个小视频预览,查看缩略图和实际的小视频不一致
9. 修复了带中文的超链接消息，点击没反应的问题

更新日志（稳定版） 更新日志（稳定版）

更新时间:2024-08-30

> 提示
>
> 仅 Android/iOS 平台的 IM SDK 存在开发版、稳定版区分。

设计原则

iOS 平台提供稳定版 IMLib SDK 和 IMKit。

- SDK 的稳定版本在线上运行时长、稳定性、使用量等方面满足一定的指标要求。
- 更强调稳定性，而非引入新功能。

发布周期与版本号规则

IM SDK 在 5.4.X 版本前后版本号规则不同。5.4.X 后更方便区分开发版、稳定版。

- 从 5.4.X 版本及以后，Stable 版本占用第二位版本号。第二位为偶数均为开发版，第二位为奇数均为稳定版。例如，5.5.X 为稳定版 SDK 使用的版本号。
- 在 5.4.X 之前，稳定版版本号规则不固定。
- 融云会监控 Stable 版本客户的使用状况，定期更新稳定版，最长更新周期为两个月。

维护说明

1. 如果融云正在积极开发的大版本号（当前为 5.X）下发布了新的 Stable 版本，我们建议使用 Stable 版本的客户升级到新的 Stable 版本。新的 Stable 版本发布后，历史稳定版维护力度相应降低。
2. 针对已不再积极开发的历史大版本（2.X、4.X）SDK，融云仅维护一个 Stable 版本。请仍使用 2.X、4.X 版本的客户尽快升级到相应的 Stable 版本，或者考虑升级到 5.X 系列的 SDK。

> 5.7.x 系列是基于 5.8.0 Dev 版本推出的稳定版本

5.7.0 Stable

发布日期：2024/05/23

优化功能：

- 优化了初始化接口偶现调用卡顿的问题。
- 优化了引用消息中文件名称过长时的显示方式。
- 优化了预览位置消息时位置锚点的显示速度。
- 增加了一个会话消息删除失败的弹窗提示。当用户在没有网络连接的情况下尝试删除会话消息时，系统会弹出此弹窗，告知用户删除操作失败。
- 优化了合并转发消息的内容显示格式。
- 优化了会话列表断网的文字提示。

问题修复：

- 修复了引用消息原文件已下载，但点击引用处的文件依然显示开始下载的问题。

5.5.X 系列是基于 5.4.8 Dev 版本推出的稳定版本。

5.5.4 Stable

发布日期：2024/03/20

优化功能：

- 优化媒本消息内部处理逻辑

5.5.3 Stable

发布日期：2024/02/29

优化功能：

- 提供包含 `PrivacyInfo.xcprivacy` 的 Framework。详见 关于 2024 春季 iOS 的隐私清单的通知 。

问题修复：

- 修复 RongLocationlib `NSMutableArray *delegateArray` 内存泄漏
- 修复 IMKit 录入小视频后会内存泄露的问题。
- 修复调用 `AVAudioSession` 的 `setCategory` 与第三方冲突，导致录制语音消息失败的问题。

5.5.2 Stable

发布日期：2023/12/08

优化功能：

- 消息推送属性（`RCMessagePushConfig`）中的 RCAndroidConfig 增加荣耀推送配置。
- 补全初始化配置 `RCInitOption` 中区域码（`AreaCode`）枚举值。
- 优化 SDK 连接逻辑。

问题修复：

- 修复接收消息线程阻塞的问题。

5.5.0 Stable

发布日期：2023/09/08

优化功能：

- IMKit 优化为在被撤回的消息本地已不存在时，仍然插入小灰条消息。
- IMLib 移除断线重连后延后 2 秒再自动加入聊天室的行为。
- 优化 IMKit 单聊、群聊会话页面消息加载速度。
- 调整 SDK 重连时间间隔为 0.05s, 0.25s, 0.5s, 1s, 2s, 4s, 8s, 16s, 32s。之后每 64s 重试一次。
- 加固了 IMKit SDK，防止极少数情况下非法字符导致的崩溃问题

问题修复：

- 修复 IMKit 在录制界面中途关闭屏幕，再恢复录制，导致视频无声音的问题。
- 修复 IMKit 在引用回复显示用户名时偶现的崩溃问题。

- 修复 12小时制模式下, 全局免打扰不生效的问题。
- IMLib 优化删除单个会话所有消息耗时较长的问题
- 修复 IMKit 选择图片时展示列表时有滚动，相册列表页面空白的问题。
- 修复 IMKit 合并转发的消息过长（超过 4 行），没有...省略号展示的问题。
- 修复 IMKit 会话界面启用位置插件，点击位置插件，弹出的视图导致导航栏及状态栏变黑的问题。
- 修复 IMKit 多选按钮没有刷新出来的问题。
- 修复 IMKit 选择图片时展示列表时有滚动，相册列表页面空白的问题。
- 修复 IMKit 件消息发送检查内容错误未返回的问题。
- 修复阿拉伯语文本内容是左对齐的的错误。
- 修复 IMKit 会话页面开启动态常用语后，右滑会话页面但不退出该页面，导致页面 UI 混乱的问题
- 修复未初始化进入会话页面 Crash 的问题
- 修复用户收取离线的扩展（KV）更新消息不全的问题。

> 5.3.X 系列是基于 5.3.5 Dev 版本推出的稳定版本。5.3.X 系列稳定版本现已过时，请尽快升级到最新 Stable 版本，或最新 dev 版本。

5.3.7 Stable

发布日期：2023/07/07

优化功能

- 弃用 `defaultRemoteHistoryMessageCount` 和 `defaultLocalHistoryMessageCount`，使用 `defaultMessageCount` 设置进入会话页面后下拉刷新从远端获取消息的条数。

问题修复

- 修复转发消息时因消息扩展的值非字符串类型造成的闪退问题
- 优化获取指定时间戳前或后消息接口，以实际传入时间戳为准，SDK 内部不做时间戳 +1 或-1 处理
- 修复用户收取离线的扩展（KV）更新消息不全的问题。
- 修复 通过 `unReadMentionedLabel` 自定义字体颜色无效的问题

5.3.6 Stable

发布日期：2023/05/05

1. IMLib/IMKit SDK 支持在消息推送属性配置中指定 vivo 推送 category 参数。
2. IMLib/IMKit SDK 支持在消息推送属性配置中指定华为推送 category 参数。

> 5.1.9 稳定版是 5.X 系列首个稳定版本。5.1.9 Stable 版本已过时，请尽快升级到最新 Stable 版本，或最新 dev 版本。

5.1.9 Stable

发布日期：2022/08/22

> 请尽快升级到最新 Stable 版本，或最新 dev 版本。

新增功能：

1. 增加获取@未读消息列表接口

客户端 API 参考 IMKit

更新时间:2024-08-30

以下是 IMKit 5.X 的 API 参考文档（appledoc）：

- IMKit（界面库）

IMLib

即时通讯基础能力库 IMLib 5.X 分为 6 个子模块：

LibCore（基础核心功能），chatroom（聊天室），customerservice（客服），discussion（讨论组），location（实时位置），publicserivce（公众号）

IMLib 核心类 `RCIMClient` 直接调用各个子模块的接口，内部没有任何具体实现。

以下是 appledoc 格式的 API 参考文档：

- RongIMLib.framework（通讯库）
- RongIMLibCore.framework（通讯核心库）
- RongChatRoom.framework（聊天室库）
- RongLocation.framework（位置）

下列 IMLib API 不再推荐使用，保留文档仅作为作参考：

- RongCustomerService.framework（客服）
- RongPublicService.framework（公众号）
- RongDiscussion.framework（讨论组）