即时通信
IMLib / IMKit
iOS 4.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

功能	单聊（Private）	普通群组（Group）	超级群（UltraGroup）	聊天室（Chatroom）
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
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 »

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。

前提条件

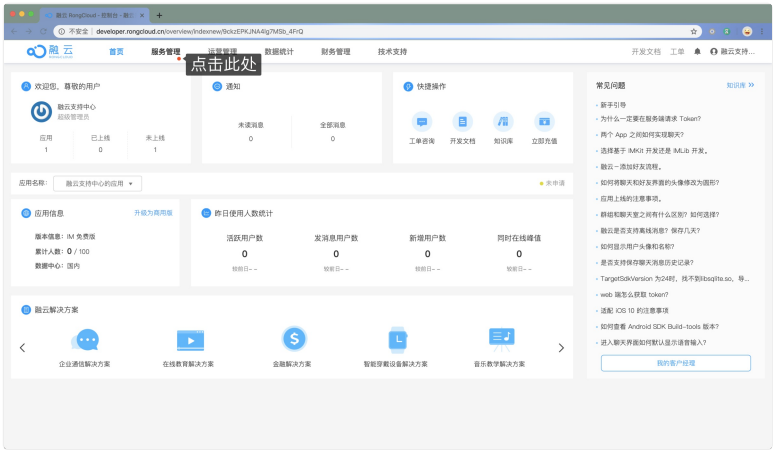
更新时间:2024-08-30

1. 访问 控制台 ，使用手机号注册账户。

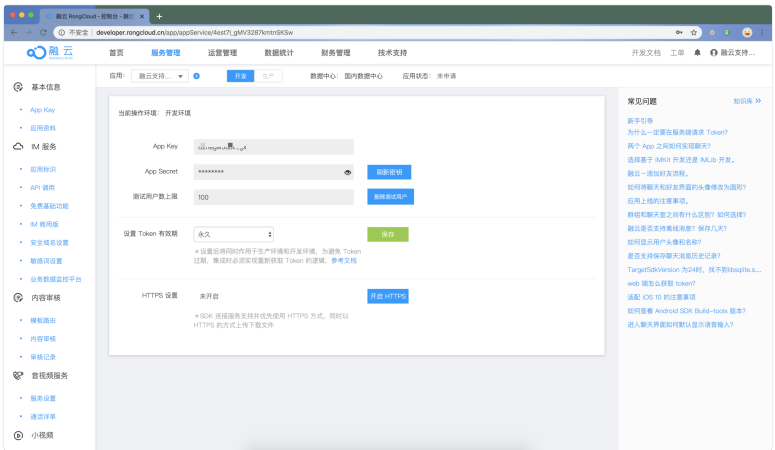
2. 填写企业名称和邮箱地址。

3. 进行邮箱验证。

4. 在控制台点击 **服务管理**。



5. 选择对应环境的 AppKey。



SDK 快速集成

更新时间:2024-08-30

> 提示
>
> 以下内容为快速集成摘要，具体细节可依次参考全局配置

导入 SDK

环境要求

名称	版本
Xcode	11 +
手机系统	8.0 +
CocoaPods	1.10.0 +

SDK 支持使用 CocoaPods 导入并管理 SDK。 需安装 CocoaPods 环境，请参照 CocoaPods 安装

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. cd 至项目根目录
2. 执行 pod init
3. 执行 open -e Podfile
4. 在 Podfile 文件中添加下面内容

```
pod 'RongCloudIM/IMKit', '~> 4.0.3.3'
pod 'RongCloudIM/IMLib', '~> 4.0.3.3'
```

   如需指定具体版本，请注意 pod 使用规范。
   请点击 SDK下载页 获知最新版本。

5. 执行 pod install
6. 双击打开 .xcworkspace

初始化

1. 引入

使用 SDK 功能前，需要 import 下面的头文件，Swift 项目需要在工程的 Bridging-Header.h 文件中添加 SDK 的引用。

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

2. **调用初始化方法**

> ⚠ 警告
>
> 1. 请使用从 融云开发者管理后台 注册得到的 `AppKey`，通过 `RCIM` 的单例，传入 `initWithAppKey:` 方法初始化 SDK。
> 2. 开发者在使用融云 SDK 所有功能之前，必须先调用此方法初始化 SDK。 在 App 的整个生命周期中，开发者只需要将 SDK 初始化一次。

```
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session options:(UISceneConnectionOptions *)connectionOptions {
[[RCIM sharedRCIM] initWithAppKey:@"获取到的 AppKey"];
}
```

连接 IM

将通过 Server 获取到的 Token，通过 RCIM 的单例，调用下面的方法，即可建立与服务器的连接。

在 App 整个生命周期，开发者只需要调用一次此方法与融云服务器建立连接。之后无论是网络出现异常或者 App 有前后台的切换等，SDK 都会负责自动重连。除非已经手动将连接断开，否则不需要自己再手动重连。

当出现 SDK 无法处理的错误（如 SDK 没有初始化、token 非法）时，SDK 将停止连接，并通过 errorBlock 进行回调。

```
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session options:(UISceneConnectionOptions *)connectionOptions {
[[RCIM sharedRCIM] initWithAppKey:@"获取到的 AppKey"];
[[RCIM sharedRCIM] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值"
dbOpened:^(RCDBErrorCode code) {}
success:^(NSString *userId) {}
error:^(RCConnectErrorCode status) {}];
}
```

构建会话列表

新建一个继承于 RCConversationListViewController 的控制器 CustomConversationListViewController，初始化控制器对象并设置需要显示的会话类型和需要聚合显示的会话类型。此处将会话列表页当做 Window 的根视图，实际开发中开发者需根据业务场景决定。

1. **导入会话列表页子类头文件**

```
#import "CustomConversationListViewController.h"
```

2. **替换根视图**

```
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session options:(UISceneConnectionOptions
*)connectionOptions {
[[RCIM sharedRCIM] initWithAppKey:@"获取到的 AppKey"];
[[RCIM sharedRCIM] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值"
dbOpened:^(RCDBErrorCode code) {}
success:^(NSString *userId) {}
error:^(RCConnectErrorCode status) {}];

UIWindowScene *windowScene = (UIWindowScene *)scene;
self.window = [[UIWindow alloc] initWithWindowScene:windowScene];
self.window.frame = [UIScreen mainScreen].bounds;
CustomConversationListViewController *conversationListViewController = [[CustomConversationListViewController
alloc]
initWithDisplayConversationTypes:@[@(ConversationType_PRIVATE),@(ConversationType_GROUP),@(ConversationType_SYSTE
M)] collectionConversationType:@[@(ConversationType_SYSTEM)]];
UINavigationController *navigationController = [[UINavigationController alloc]
initWithRootViewController:conversationListViewController];
self.window.rootViewController = navigationController;
[self.window makeKeyAndVisible];
}
```

> ⚠ 警告
>
> 会话列表页需要从融云数据库中读取会话信息进行展示，实际使用中应该在连接接口的 dbOpenedBlock 中进行页面跳转，这里为了演示方便使用了顺序调用的方式。连接接口的回调不是在主线程，开发者在 dbOpenedBlock 中进行页面跳转时请记得先切换到主线程。

构建会话页面

第一次连接成功时，由于没有任何消息，会话列表页显示空白。开发者可以在 `CustomConversationListViewController` 的导航栏添加按钮，用来开启一个单聊会话。

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

UIBarButtonItem *rightButton = [[UIBarButtonItem alloc] initWithTitle:@"开启单聊" style:UIBarButtonItemStylePlain
target:self action:@selector
(rightBarButtonItemPressed:)];
self.navigationItem.rightBarButtonItem = rightButton;
}

- (void)rightBarButtonItemPressed:(id)sender {
RCConversationViewController *conversationVC = [[RCConversationViewController alloc] init];
conversationVC.conversationType = ConversationType_PRIVATE;
conversationVC.targetId = @"888";
[self.navigationController pushViewController:conversationVC animated:YES];
}
```

在 `CustomConversationListViewController` 中重写 `RCConversationListViewController` 的 `onSelectedTableRow` 事件并传入 `conversationType` 和 `targetId`，即可点击进入聊天会话界面

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

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

设置用户信息

此处用户信息仅用于演示，开发者需要根据自身业务更换为真实的用户信息。

```
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session options:(UISceneConnectionOptions
*)connectionOptions {
[[RCIM sharedRCIM] initWithAppKey:@"获取到的 AppKey"];
[[RCIM sharedRCIM] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值"
dbOpened:^(RCDBErrorCode code) {}
success:^(NSString *userId) {}
error:^(RCConnectErrorCode status) {}];

UIWindowScene *windowScene = (UIWindowScene *)scene;
self.window = [[UIWindow alloc] initWithWindowScene:windowScene];
self.window.frame = [UIScreen mainScreen].bounds;
CustomConversationListViewController *conversationListViewController = [[CustomConversationListViewController
alloc]
initWithDisplayConversationTypes:@[@(ConversationType_PRIVATE),@(ConversationType_GROUP),@(ConversationType_SYSTE
M)] collectionConversationType:@[@(ConversationType_SYSTEM)]];
UINavigationController *navigationController = [[UINavigationController alloc]
initWithRootViewController:conversationListViewController];
self.window.rootViewController = navigationController;
[self.window makeKeyAndVisible];

//添加用户信息代理
[RCIM sharedRCIM].userInfoDataSource = self;
//设置用户信息在本地持久化存储。SDK 获取过的用户信息将保存在数据库中，即使 App 重新启动也能再次读取。
[RCIM sharedRCIM].enablePersistentUserInfoCache = YES;
}

// 实现用户信息提供者的代理函数
- (void)getUserInfoWithUserId:(NSString *)userId completion:(void (^)(RCUserInfo *))completion {
//开发者需要将 userId 对应的用户信息返回，下列仅为示例
//实际项目中，开发者有可能需要到 App Server 获取 userId 对应的用户信息，再通过 completion 返回给 SDK。
if ([userId isEqualToString:@"888"]) {
RCUserInfo *userInfo = [[RCUserInfo alloc] initWithUserId:@"888" name:@"融融"
portrait:@"https://timgsa.baidu.com/timg?
image&quality=80&size=b9999_10000&sec=1568875876191&di=2933bebe9b22899f59840c33b24a1f98&imgtype=0&src=http%3A%2F
g.mp.itc.cn%2Fupload%2F20161119%2F7205fcb7ca4149c1920c01524112cbd9_th.jpg"];
if (completion) {
completion(userInfo);
}
}
}
```

引入 SDK

环境要求

更新时间:2024-08-30

名称	版本
Xcode	11 +
手机系统	8.0 +
CocoaPods	1.10.0+

导入 SDK

SDK 支持使用 CocoaPods 导入并管理 SDK。 需安装 CocoaPods 环境，请参照 CocoaPods 安装

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

1. cd 至项目根目录
2. 执行 pod init
3. 执行 open -e Podfile
4. 在 Podfile 文件中添加下面内容

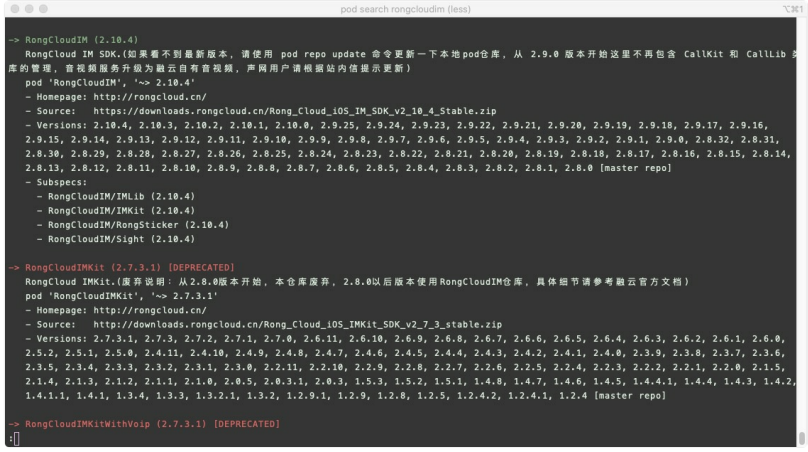
```
pod 'RongCloudIM/IMKit', '~> 4.0.3.3'
pod 'RongCloudIM/IMLib', '~> 4.0.3.3'
```

   如需指定具体版本，请注意 pod 使用规范。
   请点击 下载页 获知最新版本。

5. 执行 pod install
6. 双击打开 .xcworkspace

升级

1. 执行 pod repo update
2. 执行 pod search RongCloudIM 查询最新版本



3. 替换 Podfile 文件内容

4. 执行 pod install

查看当前导入版本

1. 打开根目录
2. 到 `Pods/RongCloudIM/RongCloudIM/RongIMLib.framework` 目录下
3. 打开 `info.plist` 文件
4. 找到 `Bundle versions string, short` 字段，其 `value` 值为当前集成版本号

清除 Pod 缓存

1. 移除项目目录 `Pods` 文件下的 `RongCloudIM`。
2. 删除 `CocoaPods` 目录缓存，找到 `~/Library/Caches/CocoaPods/Pods/Release`，删除此目录。
3. 点击 `Xcode -> Preferences -> Locations -> Derived Data` 点击下方的右箭头跳转到相应目，删除此目录即可。
4. 项目根目录执行 `pod repo update`。
5. 项目根目录执行 `pod install`。

常见问题

> Q1: 集成 SDK 必须使用 Xcode 11 吗？
>
> A1: 融云 SDK 2.9.25 及之后的版本兼容了 iOS 13 ，需要兼容 iOS 13，则需使用新版本的 Xcode；不需要兼容 iOS 13 或者不想升级到 Xcode 11，请使用 2.9.24 版本的 SDK。 不兼容的原因

初始化 功能描述

更新时间:2024-08-30

融云 SDK 需要开发者在工程中调用下面方法来初始化 SDK。

> ⚠ 警告
>
> 1. 在使用融云 SDK 所有功能之前，必须先调用此方法初始化 SDK，可将此方法写到 `AppDelegate.m` 或 `SceneDelegate` 中。
> 2. Appkey 区分 开发环境 与 生产环境，使用时注意当前环境是否匹配。
> 3. 在 App 的整个生命周期内，** SDK 只能 init 一次 **，不可重复调用此接口。

参数说明

参数	类型	必填	说明
appKey	NSString	是	开发者在 控制台 获取的 Appkey

代码示例

1. 导入头文件

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

2. 初始化 SDK

```
[[RCIM sharedRCIM] initWithAppKey:@"此处填写获取的 Appkey "];
```

设置监听

设置状态监听

更新时间:2024-08-30

当 SDK 与融云服务器的连接状态发生变化时，开发者可通过下面方法进行处理。实现此功能需要开发者遵守 `RCIMConnectionStatusDelegate` 协议。

参数说明

代理对象参数说明：

参数	类型	必填	说明
connectionStatusDelegate	id <RCIMConnectionStatusDelegate>	是	连接状态的监听器

协议方法参数说明：

参数	类型	说明
status	RCConnectionStatus	SDK 与融云服务器的连接状态

代码示例

1. 设置代理对象:

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

2. 实现协议方法:

```
- (void)onRCIMConnectionStatusChanged:(RCConnectionStatus)status{

}
```

设置消息监听

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。实现此功能需要开发者遵守 `RCIMReceiveMessageDelegate` 协议。

参数说明

代理对象参数说明：

参数	类型	必填	说明
receiveMessageDelegate	id <RCIMReceiveMessageDelegate>	是	消息接收的监听器

协议方法参数说明：

参数	类型	说明
message	RCMessage	当前接收到的消息
left	int	还剩余的未接收的消息数

代码示例

1. 设置代理对象:

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

2. 实现协议方法:

```
- (void)onRCIMReceiveMessage:(RCMessage *)message left:(int)left {

}
```

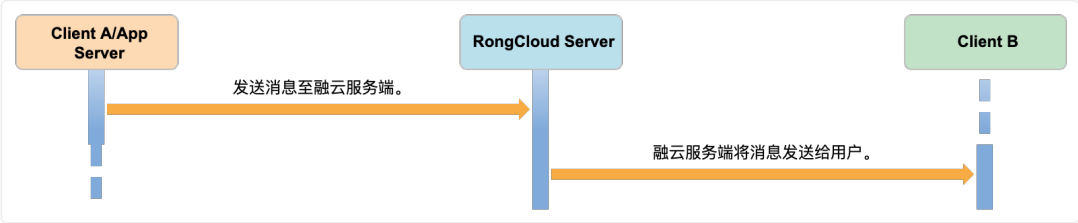
> ⚠ 警告
>
> 通过此方法可以获取到每条消息，left 会依次递减直到 0。开发者可以根据 left 数量来优化 App 体验和性能，比如收到大量消息时等待 left 为 0 再刷新 UI。

本地通知提醒 功能描述

更新时间:2024-08-30

当 App 进入后台，SDK 处于后台活跃状态，App 会使用融云的长连接通道接收消息，此时 SDK 收到消息会弹出本地通知。

流转图：



> ⚠ 警告
>
> iOS 端会在 **2** 分钟之后主动断开，之后的消息会通过 APNs 通道进行发送。

配置说明

您可以通过将此属性设置为 YES，关闭所有的本地通知。

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

回调方法

如果您设置了IMKit消息监听之后，当App处于后台，收到消息时弹出本地通知之前，会执行此方法。如果App没有实现此方法，SDK会弹出默认的本地通知提示。

如果App在后台想使用SDK默认的本地通知提醒，需要实现用户/群组/群名片信息提供者，并返回正确的用户信息或群组信息。

回调流程：

参数说明

参数说明：

参数	类型	说明
message	RCMessage	接收到的消息

参数	类型	说明
senderName	NSString	消息发送者的用户名称

返回参数说明

类型	说明
BOOL	是否弹出本地通知提示

> 当返回值为 NO 时，SDK 会弹出默认的本地通知提示；当返回值为 YES 时，SDK 针对此消息不再弹本地通知提示

代码示例

```
- (BOOL)onRCIMCustomLocalNotification:(RCMessage *)message withSenderName:(NSString *)senderName{
return YES;
}
```

获取通知内容

内容格式

SDK 默认的本地通知 `UILocalNotification` 字段信息如下:

参数	类型	说明
alertBody	String	在通知栏显示的内容
userInfo	NSDictionary	消息的附加信息

`userInfo` 说明：

参数	类型	说明
cType	String	会话类型。PR 指单聊、 DS 指讨论组、 GRP 指群组、 CS 指客服、SYS 指系统会话、 MC 指应用内公众服务、 MP 指跨应用公众服务、PH 指不落地通知。
fId	String	消息发送者的用户 ID。
oName	String	消息类型，参考融云消息类型表.消息标志；可自定义消息类型。
tId	String	为 Target ID。

内容获取

点击通知栏的本地通知时，如果 App 未被系统冻结，开发者可使用下面方式获取:

```
- (void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification
{

NSDictionary *msgDic = [notification.userInfo objectForKey:@"rc"];
NSString *cType = [msgDic objectForKey:@"cType"];
NSString *fId = [msgDic objectForKey:@"fId"];
NSString *oName = [msgDic objectForKey:@"oName"];
NSString *targetId = [msgDic objectForKey:@"tId"];//Target ID 。

/*
cType根据得到字段转换成相应的conversationType
其余信息可根据需求进行相应的操作
*/
RCConversationViewController *chat = [[RCConversationViewController alloc]init];
//设置会话的类型，如单聊、群聊、客服,为转换后的值
chat.conversationType = cType_Value;
//设置会话的目标会话 ID。（单聊、客服、公众服务会话为对方的ID，群聊为会话的ID）
chat.targetId = targetId;
//设置聊天会话界面要显示的标题
chat.title = @"想显示的会话标题";
//显示聊天会话界面
if ([self.window.rootViewController isKindOfClass: [RCDNavigationViewController class]]) {
[(RCDNavigationViewController *)self.window.rootViewController pushViewController:chat animated:YES];
} else {
//........
}
}
```

点击通知栏的本地通知时，如果此时 App 已经被系统冻结，开发者可使用下面方式获取:

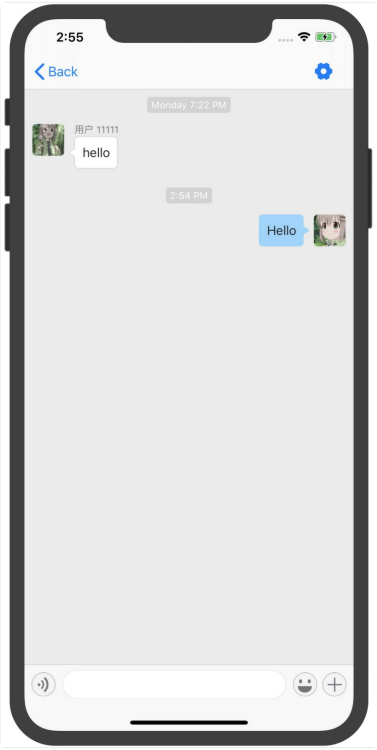
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

// 本地通知的内容
UILocalNotification *localNotification = launchOptions[UIApplicationLaunchOptionsLocalNotificationKey];
//解析同理

return YES;
}
```

用户信息 设置当前登录的用户信息

更新时间:2024-08-30

开发者与融云服务器建立连接之后，应该设置当前用户的用户信息，用于 SDK 显示和发送。可使用下面方式设置当前登录的用户信息。

效果展示



属性说明

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。

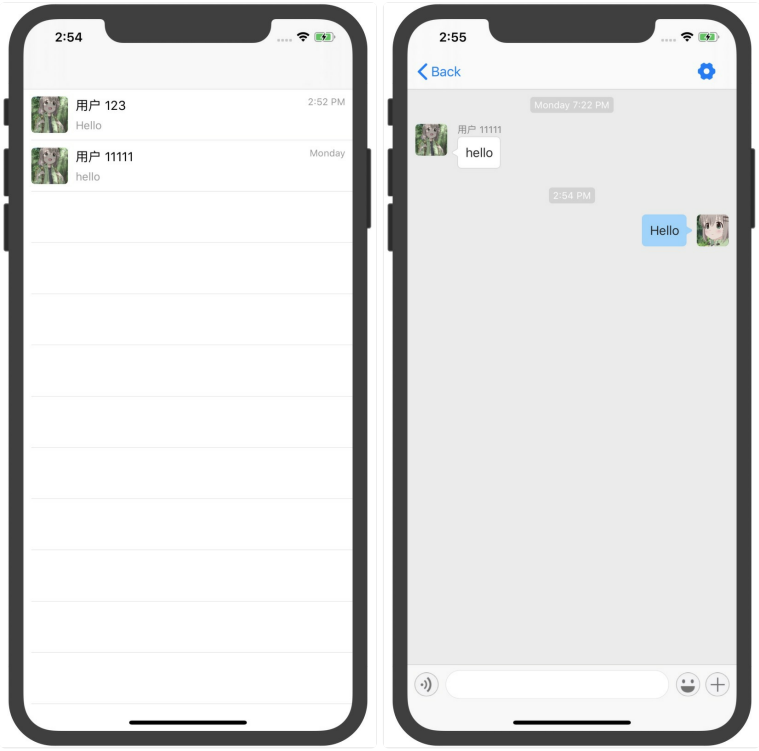
```
@property (nonatomic, strong) RCUserInfo *currentUserInfo;
```

代码示例

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

设置其他用户的用户信息

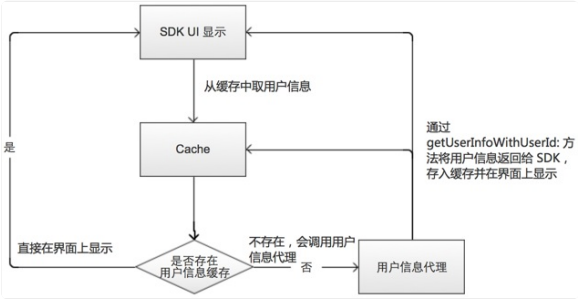
效果展示



设置用户信息

SDK 设计了用户信息提供者，开发者需要实现该协议，提供正确的数据。SDK 在展示用户头像和昵称的时候，会调用协议的代理函数并根据返回的信息进行显示。**请确保此方法是全局有效的，您可以在某个单例里实现此协议。用户信息提供者获取过的用户信息会在内存中缓存，SDK 展示用户信息时优先从缓存里获取。**

逻辑说明：



实现此功能需要开发者遵守 `RCIMUserInfoDataSource` 协议。

参数说明

代理对象参数说明：

参数	类型	必填	说明
userInfoDataSource	id <RCIMUserInfoDataSource>	是	用户信息提供者

协议方法参数说明：

参数	类型	说明
userId	NSString	用户 ID

参数	类型	说明
completion	Block	获取用户信息完成之后需要执行的 Block

回调参数说明：

参数	类型	必填	说明
userInfo	RCUserInfo	是	用户 ID 对应的用户信息

代码示例

1. 设置代理对象:

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

2. 实现协议方法:

```
- (void)getUserInfoWithUserId:(NSString *)userId completion:(void (^)(RCUserInfo *))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
completion(userInfo);

}];
}
```

获取用户信息

开发者可通过下面方法获取 SDK 中缓存的用户信息。此处获取的用户信息是开发者通过 设置用户信息 接口传递给 SDK 的，如果没有传递会返回空。

参数说明

输入参数说明：

参数	类型	必填	说明
userId	NSString	是	用户 ID

返回参数说明：

返回值	返回类型	说明
userInfo	RCUserInfo	SDK 中缓存的用户信息

代码示例

```
RCUserInfo *userInfo = [[RCIM sharedRCIM] getUserInfoCache:self.userId];
```

更新用户信息

开发者可通过下面方法更新 SDK 缓存的用户信息。

出于性能和使用场景的权衡，SDK 不会在当前 View 立即自动刷新（会在切换到其他 View 的时候再刷新该用户的显示信息）。如果需要立即刷新，可以在会话列表或者会话页面 reload 强制刷新。

参数说明

参数	类型	必填	说明
userInfo	RCUserInfo	是	需要更新的用户信息
userId	NSString	是	需要更新的用户 ID

代码示例

```
[[RCIM sharedRCIM] refreshUserInfoCache:userInfo withUserId:userId];
```

清除用户信息

开发者可通过下面方法清空 SDK 缓存的所有用户信息。

出于性能和使用场景的权衡，SDK 不会在当前 View 立即自动刷新（会在切换到其他 View 的时候再刷新该用户的显示信息）。如果需要立即刷新，可以在会话列表或者会话页面 reload 强制刷新。

代码示例

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

持久化存储

当需要将用户信息在本地持久化保存的时候，可以开启 RCIM 类的此属性。

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

默认为 NO。用户信息只缓存到内存，在 App 生命周期结束时会被清除，App 下次启动时将再次通过用户信息提供者获取信息。

设置为 YES，用户信息将被同时缓存到内存和数据库，App 下次启动时用户信息缓存仍然有效。

监听连接状态

更新时间:2024-08-30

当 SDK 与融云服务器的连接状态发生变化时，开发者可通过下面代理方法监听 SDK 连接状态。

设置连接监听

```
//其中 self 最好为单例类(如 AppDelegate)，以此保证在整个 APP 生命周期，该类都能够接收到 SDK 连接状态的变更。
[[RCIM sharedRCIM] setConnectionStatusDelegate: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
}
}
```

连接异常状态码说明

RCConnectionStatus 中定义了连接过程中的可能的状态变化。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，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

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。在应用的整个生命周期，此方法只需要调用一次，之后无论是网络异常或者 App 有前后台的切换，SDK 都会自动重连，直到开发者主动断开连接。

准备 Token

用户身份验证令牌 (Token) 代表一个用户在融云的唯一身份标识。获取 Token 必须调用融云服务端 API 的接口。请提供准确的 App Key / Secret，并使用用户 ID 换取对应的 Token。

融云客户端 SDK 不提供获取 Token 的接口。请从您的应用服务端发起 API 调用请求。具体操作请参见融云服务端文档注册用户 。

> 为什么不要在客户端获取 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。

获取 Token 后，您可以按需保存 Token，供后续连接时使用。如果 Token 未失效，就不必再向融云请求 Token。具体保存位置取决于应用客户端设计。

Token 有效期

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。

您可以调用服务端 API `/user/token/expire.json` 主动废弃 Token。详见服务端 API 文档 作废 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。

接口调用建议：

- **用户初次登录**：建议调用带超时设置（`timeLimit`） 的接口，并为其设置有效值，保证初次登录如遇到异常情况（如网络较差等）SDK 可以及时回调连接超时，由 APP 自行处理重连。
- **用户后续免密登录**：后续登录说明 token 已经是合法可用的，那么 APP 直接调用不带超时设置的接口，IM 的连接完全由 IM SDK 接口，APP 不需要额外处理。

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。
- 在应用的整个生命周期，此方法只需要调用一次，之后无论是网络异常或者 App 有前后台的切换，SDK 都会自动重连，直到开发者主动断开连接。
- SDK 本身有重连机制，在一个应用生命周期内不要多次调用 `connect` 。否则可能触发多个回调，触发导致回调被清除。

建立 IM 连接（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。在网络较差等导致连接超时的情况下，App 可以利用接口的超时错误回调，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。

```
[[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) {
//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
}
}]
```

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 。

建立 IM 连接（无超时设置）

如果客户端用户已成功登录过 App ，且已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。

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

连接状态码

当回调 errorBlock 时，并且 SDK 不会再进行重连，返回 RCConnectErrorCode 错误码。

错误码	值	说明	处理方案
RC_CONN_TOKEN_INCORRECT	31004	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	1. 检查 SDK 与 App 服务器使用的 App Key 是否一致。 2. 从 APP 服务器获取新的 token ，再重新进行连接。
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_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_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

错误码	值	说明	处理方案
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
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
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
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

1. 在 `31004` 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. 此方法的回调并非为原调用线程，需要进行 UI 操作，请注意切换到主线程。
3. 如果 App 集成了 IMKit，请使用 RCIM 中的同名方法建立与融云服务器的连接，具体请参考 IMKit 对应文档。

重连机制与重连互踢

自动重连机制

更新时间:2024-08-30

SDK 已实现重连机制，在 App 的整个生命周期内，开发者只需要调用一次连接服务，在需要退出的时候再调用断开连接即可。

重连时间间隔

SDK 会根据网络状态和 App 的行为（前后台等），在连接中断的情况下不断进行重连。

重连的时间间隔是 0s, 0.25s, 0.5s, 1s, 2s, 4s, 8s, 16s, 32s, 64s, 64s, 64s...

当 **APP 切换到前台** 或者 **网络变化** 时，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。

> 一般来说，App 同时在线，不等于 IM 消息通道需要同时在线。因为用户同时只会操作一个手机，只需开通 多设备消息同步，保证各端消息一致即可。

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。

> - IMKit 不支持提供该接口。请使用 IMLib 核心类 `RCIMClient` 中的方法。
> - 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。

断开连接 功能描述

更新时间:2024-08-30

开发者连接融云服务之后，在进行用户切换、用户注销的操作时，可通过下面方法断开连接，并可根据此方法来指定在用户断开连接之后是否接收消息推送。

SDK 在前后台切换或者网络出现异常时都会自动重连。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。

参数说明

参数	类型	必填	说明
isReceivePush	BOOL	是	App 在断开连接之后，是否还接收远程推送。 YES:表示接收远程推送，NO:表示不接收远程推送

代码示例

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

多端同时在线

更新时间:2024-08-30

默认的情况下，融云仅支持 1 个 Web 端、1 个 桌面端、1 个移动端同时在线。

开通多设备消息同步功能后，可以支持 Web 端、桌面端和移动端之间的消息同步。且开通此功能后，可以同时支持多个 Web 端同时在线。重新登录时，获取当天收发的所有消息。

开通方式

- 开发环境，默认为关闭状态，开启后 30 分钟内生效。
- 生产环境，**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。服务开启、关闭设置完成后 30 分钟内生效。

连接服务

更新时间:2024-08-30

融云 IMKit 已经实现了一个默认的会话列表视图控制器，直接使用或继承此类，即可快速启动和使用会话列表界面。

构建会话列表

效果展示

图中未展示发送者昵称和头像是因为未设置用户信息，具体如何设置请参考 设置用户信息

```
/*!
会话列表界面类
*/
@interface RCConversationListViewController : RCBaseViewController <UITableViewDataSource, UITableViewDelegate>
```

构建说明

开发者可通过两种方式使用会话列表页面。

1. 直接使用此类

   建议仅在排查会话列表的相关错误时使用，如果在使用此页面期间出现了相关问题，可以直接 push 到此类排查是否能复现相关问题。

2. 继承使用此类

开发者可继承此类，并跳转到此页面来开发相关内容。

参数说明

参数	类型	必填	说明
displayConversationTypeArray	NSArray	是	列表中需要显示的会话类型数组
collectionConversationTypeArray	NSArray	否	列表中需要聚合为一条显示的会话类型数组

> ⚠ 警告
>
> displayConversationTypeArray 和 collectionConversationTypeArray 都需要先将 RCConversationType 转为 NSNumber 再构建 Array，直接传 RCConversationType 会导致无法显示列表数据。

方法原型

```
- (instancetype)initWithDisplayConversationTypes:(NSArray *)displayConversationTypeArray
collectionConversationType:(NSArray *)collectionConversationTypeArray;
```

代码示例

```
NSArray *displayConversationTypeArray = @[@(ConversationType_PRIVATE)];
RCDConversationListViewController *conversationListVC = [[RCDConversationListViewController alloc]
initWithDisplayConversationTypes:displayConversationTypeArray collectionConversationType:nil];
[self.navigationController pushViewController:conversationListVC animated:YES];
```

构建聚合会话列表

开发者只需在构建会话列表的构建方法中中传入需要聚合的会话类型即可。

常见问题

> Q1: 当用户卸载重新安装后发现会话列表为空或者部分会话丢失
>
> A1: 由于会话列表是从本地数据库获取的, 是在 SDK 内部数据库存储的，所以当用户在卸载的时候会删除本地数据库，导致重新安装后会话列表为空，而会出现部分会话的原因是因为开启了 离线消息补偿 功能，具体操作是在后台购买 多设备消息同步 功能，这个功能里默认涵盖了最近 1 个自然日内（含当天）收发的消息补偿，所以当用户在新设备登陆会触发离线消息补偿功能，从而获取到部分会话，造成部分会话丢失的错觉。如果想要更多天的离线消息补偿，可 提交工单 修改，最多支持 7 天，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。

事件设置 数据相关

更新时间:2024-08-30

收到新消息

参数说明：

参数	类型	说明
notification	NSNotification	收到新消息的 notification

回调原型：

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

> ⚠ 警告
>
> SDK 在此方法中有针对消息接收有默认的处理（如刷新等），如果开发者重写此方法，请注意调用 super。

删除会话

参数说明：

参数	类型	说明
model	RCConversationModel	会话 Cell 的数据模型

回调原型：

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

即将加载数据源

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。

参数说明：

传入参数说明:

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

返回参数说明:

参数	类型	说明
dataSource	NSMutableArray	修改后的数据源

回调原型：

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

> ⚠ 警告
>
> 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

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，您可以在此回调中执行未读消息数相关的操作。

回调原型：

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

> ⚠ 警告
>
> 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。

Cell 显示

即将显示 Cell

开发者可以在此回调中修改 Cell 的一些显示属性。会话列表自带 Cell 样式如字体颜色，字体大小的修改，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 Cell。

参数说明：

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

回调原型：

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

> ⚠ 警告
>
> 数据源中存放的元素为会话 Cell 的数据模型，即 RCConversationModel 对象。

Cell 点击

点击 Cell

参数说明：

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

回调原型：

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

点击头像

参数说明：

参数	类型	说明
model	RCConversationModel	会话 Cell 的数据模型

回调原型：

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

长按头像

参数说明：

参数	类型	说明
model	RCConversationModel	会话 Cell 的数据模型

回调原型：

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

自定义 Cell

自定义会话 Cell 显示

参数说明：

参数	类型	说明
tableView	UITableView	当前TabelView
indexPath	NSIndexPath	该 Cell 对应的会话 Cell 数据模型在数据源中的索引值

参数	类型	说明

返回参数说明

参数	类型	说明
dataSource	RCConversationBaseCell	自定义会话需要显示的Cell

回调原型：

```
- (RCConversationBaseCell *)rcConversationListTableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath;
```

左滑删除自定义会话

参数说明：

参数	类型	说明
tableView	UITableView	当前TabelView
editingStyle	UITableViewCellEditingStyle	当前的 Cell操作，默认为UITableViewCellEditingStyleDelete
indexPath	NSIndexPath	该 Cell 对应的会话 Cell 数据模型在数据源中的索引值

回调原型：

```
- (void)rcConversationListTableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath;
```

会话搜索 功能描述

更新时间:2024-08-30

开发者可根据下面方法实现会话搜索功能。

实现方法

SDK 未提供内置的 UI 组件，开发者需根据业务需求参考 SealTalk 开源项目 中的示例代码实现 UI 功能，之后调用下面接口即可。

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array
objectNameList	objectNameList	是	需要搜索的消息类型名列表(即每个消息类方法getObjectName的返回值)
keyword	NSString	是	关键字

返回参数说明

返回值	返回类型	说明
messsageResult	NSArray	匹配的会话搜索结果列表

代码示例

```
NSArray *messsageResult = [[RCIMClient sharedRCIMClient] searchConversations:@[ @(ConversationType_GROUP),
@(ConversationType_PRIVATE) ]
messageType:@[
[RCTextMessage getObjectName],
[RCRichContentMessage getObjectName],
[RCFileMessage getObjectName]
]
keyword:searchText];
```

自定义 UI 自定义头像

更新时间:2024-08-30

开发者可根据业务场景自行定义会话页面头像的显示形状和大小

头像显示形状

属性所属类为 RCIM，默认值为矩形

```
@property (nonatomic) RCUserAvatarStyle globalConversationAvatarStyle;
```

RCUserAvatarStyle 说明：

类型名	说明
RC_USER_AVATAR_RECTANGLE	矩形
RC_USER_AVATAR_CYCLE	圆形

头像显示大小

属性所属类为 RCIM，高度必须大于或者等于 **36**

```
@property (nonatomic) CGSize globalConversationPortraitSize;
```

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，将 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

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;
```

参考 demo 会话列表的 RCDChatListViewController 中上述相关方法实现和自定义 Cell（RCDChatListCell），demo 具体是针对系统会话（ConversationType_SYSTEM）的好友请求消息（RCContactNotificationMessage）做了自定义 Cell

自定义空视图

会话列表 RCConversationListViewController 中提供了设置无会话需要显示的 View，只需要自己生成一个 View，设置好坐标和需要显示的内容即可。

```
/*!
列表为空时显示的 View
*/
@property(nonatomic, strong) UIView *emptyConversationView;
```

代码示例：

```
- (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;
}
```

自定义未读数

开发者可通过 获取未读数 接口获取到所有未读数，然后在需要显示的地方自行显示即可。

自定义网络连接提示

开发者可使用下面属性设置是否显示网络连接提示。

属性所属类 RCConversationListViewController，默认为 YES。

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

如果开发者需要修改提示文字，可修改语言包中的 en.lproj 和 zh-Hans.lproj。

会话标签

更新时间:2024-08-30

警告

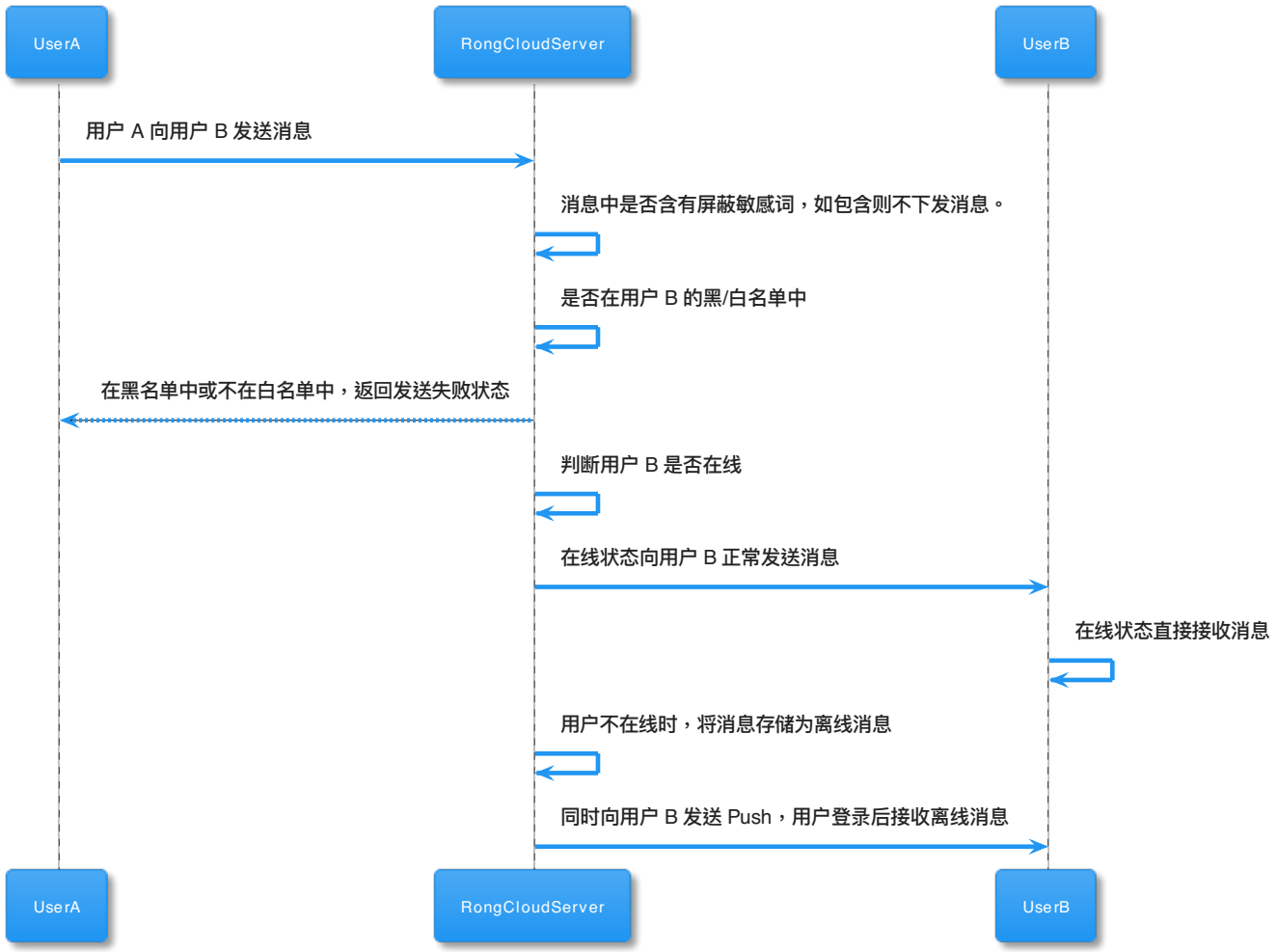
该功能仅在5.0以上版本SDK中提供！

单聊介绍 概述

更新时间:2024-08-30

适用于应用内两个用户之间一对一聊天方式，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。

消息发送时序图：



主要功能

功能	描述
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。
消息提醒	离线状态，单聊中有新消息时，支持 Push 通知。
本地存储	存储在移动端本地，提供本地消息搜索功能。
历史消息	提供服务端消息存储存功能，需开通单群聊消息云存储，默认存储时长为 6 个月。
消息删除	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。
消息搜索	支持按关键字或用户搜索本地指定会话的消息内容。
消息阅读回执	发送单聊消息后如需要查看消息的阅读状态，可以使用此功能来发送阅读回执请求。

功能	描述
消息撤回	消息发送成功后，在有效时间内可撤回该条消息，默认可撤回时间为 2 分钟，时间可配置。
单聊会话免打扰	可设置指定的单聊会话，收到新的消息后是否进行提醒，默认进行新消息提醒。
单聊黑名单	不想接收到某一用户的消息时，可将此用户加入到黑名单中，应用中的每个用户都可以设置自己的黑名单列表
单聊白名单	对用户之间相互发送消息有限制要求的客户，可使用用户白名单功能，将用户加入白名单后，才能收到该用户发送的单聊消息

注：用户白名单服务与用户黑名单服务不能同时使用，融云默认开启的是用户黑名单服务，如需要开通白名单服务请提交工单申请开通。服务开通 30 分钟后生效，同时黑名单服务不再生效。

消息类型

消息类型	描述
文字消息	用来发送文字类消息，其中可以包括表情、超链接（会自动识别），客户端收到消息后计入未读消息数、进行存储。
语音消息	发送高质量的短语音消息，录制的语音文件存储到融云服务端，语音文件格式为 AAC，时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。
图片消息	用来发送图片类消息，客户端收到消息后计入未读消息数、进行存储。图片缩略图格式为 JPG，大小建议不超过 100k。
GIF 图片消息	用来发送 GIF 动态图片消息，客户端收到消息后计入未读消息数、进行存储。
图文消息	用来发送图文消息，包含一个标题，一段文字内容和一张图片，客户端收到消息后计入未读消息数、进行存储。
文件消息	用来发送文件类消息，客户端收到消息后计入未读消息数、进行存储。
位置消息	用来发送地理位置消息，客户端收到消息后计入未读消息数、进行存储。
小视频消息	用来发送小视频消息，支持录制发送及选择本地视频文件发送两种方式，录制时长不超过 10 秒，本地选择视频文件方式时长不超过 2 分钟，小视频消息小视频文件格式为 .mp4，客户端收到消息后计入未读消息数、进行存储。
合并转发消息	IMKit SDK 中支持将多条消息合并为一条消息进行发送，合并后的消息以 HTML 文件的方式存储到融云服务端，客户端收到消息后计入未读消息数、进行存储。红包、阅后即焚及自定义消息的合并转发功能
命令消息	用来发送通用的指令通知消息，消息内可以定义任意 JSON 内容，与通用命令通知消息的区别是不存储、不计数，此类型消息没有 Push 通知。
自定义消息	融云内置消息类型，无法满足客户业务需求时，可通过自定义消息类型进行实现，接收自定义消息的格式解析及展示处理需要开发者自行实现

好友关系

更新时间:2024-08-30

融云为了客户隐私考虑，既不同步又不保存用户的好友关系。所以，所有用户的好友关系都需要开发者自己实现、管理维护。

如果使用的是 IMKit 进行集成，则会话及好友列表中显示好友的昵称及头像信息，需要 App 设置一个用户信息提供者给 IMKit，以便 IMKit 通过用户信息提供者，来实现在聊天界面和会话列表页中显示好友的昵称和头像。

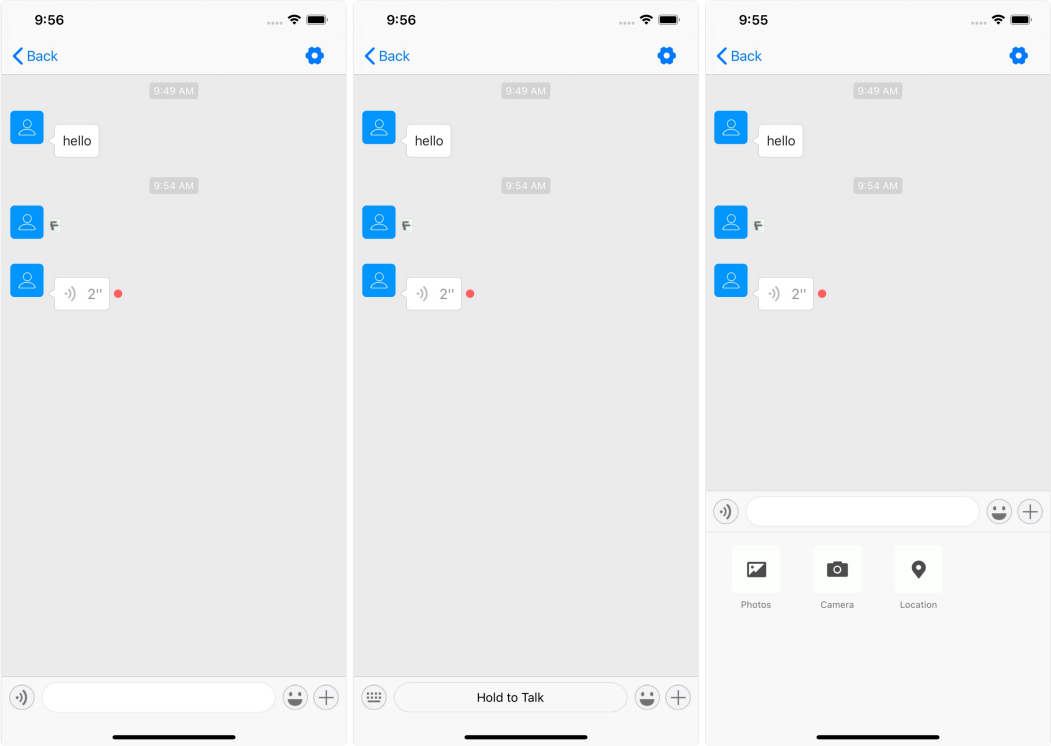
会话界面 功能描述

更新时间:2024-08-30

融云 IMKit 已经实现了一个默认的会话视图控制器，直接使用或继承此类，即可快速启动和使用会话界面。

效果展示

图中未展示发送者昵称和头像是因为未设置用户信息，具体如何设置请参考 设置用户信息



构建会话界面

```
/*!
会话页面类
*/
RCConversationViewController
```

构建说明

开发者可通过两种方式使用会话列表页面。

1. 直接使用此类

   建议仅在排查会话列表的相关错误时使用，如果在使用此页面期间出现了相关问题，可以直接 push 到此类排查是否能复现相关问题。

2. 继承使用此类

开发者可继承此类，并跳转到此页面来开发相关内容。

参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId

方法原型

```
- (id)initWithConversationType:(RCConversationType)conversationType targetId:(NSString *)targetId;
```

代码示例

```
RCDConversationViewController *conversationVC = [[RCDConversationViewController alloc]
initWithConversationType:ConversationType_PRIVATE targetId:@"targetId"];
[self.navigationController pushViewController:conversationVC animated:YES];
```

界面事件回调

输入框相关

更新时间:2024-08-30

内容发生变化

参数说明：

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

回调原型：

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

高度发生变化

参数说明：

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

回调原型：

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

> ⚠ 警告
>
> 如重写此方法，请先调用父类方法。

消息操作相关

准备发送消息

此回调在消息准备向外发送时会回调，可以在此回调中对消息内容进行过滤和修改等操作。此回调的返回值不为 nil 时，SDK 会对外发送返回的消息内容。

参数说明：

参数	类型	说明
messageContent	RCMessageContent	消息内容

返回参数说明

参数	类型	说明
messageContent	RCMessageContent	修改后的消息内容

回调原型：

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

发送消息完成

参数说明：

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

回调原型：

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

取消消息发送

参数说明：

参数	类型	说明
messageContent	RCMessageContent	消息内容

回调原型：

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

即将插入消息

此回调在消息准备向外发送时会回调，可以在此回调中对消息内容进行过滤和修改等操作。此回调的返回值不为 nil 时，SDK 会对外发送返回的消息内容。

参数说明：

参数	类型	说明
messageContent	RCMessageContent	消息内容

返回参数说明

参数	类型	说明
messageContent	RCMessageContent	修改后的消息内容

回调原型：

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

Cell 显示

即将显示消息 Cell

参数说明：

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

回调原型：

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

未注册消息相关

显示内容

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

返回参数说明

类型	说明
RCMessageBaseCell	未注册消息需要显示的Cell

回调原型：

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

显示高度

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

返回参数说明

类型	说明
CGSize	未注册消息Cell需要显示的高度

回调原型：

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

Cell 点击事件

点击头像

参数说明：

参数	类型	说明
userId	NSString	点击头像对应的用户ID

回调原型：

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

长按头像

参数说明：

参数	类型	说明
userId	NSString	点击头像对应的用户ID

回调原型：

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

点击消息内容

参数说明：

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型

回调原型：

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

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

长按消息内容

参数说明：

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

回调原型：

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

> ⚠ 警告
>
> 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 URL

参数说明：

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

回调原型：

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

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

点击电话号码

参数说明：

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

回调原型：

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

获取长按时的菜单

参数说明：

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型

返回参数说明

类型	说明
NSArray <UIMenuItem *>	未注册消息Cell需要显示的高度

回调原型：

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

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

语音相关

开始录制

回调原型：

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

结束录制

回调原型：

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

取消录制

回调原型：

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

历史消息获取

功能描述

更新时间:2024-08-30

SDK 页面内已经内置了获取历史消息的功能，无需用户再次开发。

消息发送 属性描述

更新时间:2024-08-30

消息属性	描述
消息类名	各端消息名
存储属性	存储 / 不存储
离线属性	缓存 / 不缓存
推送属性	是/否
ObjectName	传输层名称，与消息类名一一对应
计数属性	计数 / 不计数
消息尺寸	128 KB
推送内容	详见各消息类送方式

存储属性

存储属性	存储分类	支持平台	详细描述
存储	客户端	Android、iOS	发送、接收该消息后，本地数据库存储 Web 端 和 小程序端因本地存储不可靠，不支持客户端消息存储，但可通过历史消息云存储服务获取历史记录
存储	云端	Android、iOS、Web	默认不在云端进行存储，需开通单群聊消息云存储服务，开通后，可在融云服务端存储 6 个月的历史消息，供客户端按需拉取
不存储	客户端	Android、iOS	发送、接收该消息后，本地数据库不存储
不存储	云端	Android、iOS、Web	无论是否开通历史消息云存储服务，该消息均不存储

计数属性

接收收到消息时，会话是否累计未读数。

计数属性	支持平台	详细描述
计数	iOS、Android、Web	会话未读消息数 + 1，该属性只影响会话列表未读数计数，App 应用角标可根据每个会话列表未读数累加获得
不计数	iOS、Android、Web	会话未读消息数不变

离线属性

接收人当前不在线时，是否进行离线缓存。

离线属性	详细描述
存储	消息进行离线缓存，默认 7 天。接收人在 7 天内上线，均可接收到该消息。超过 7 天后，消息被离线缓存淘汰。如有需要，可通过云端存储拉取到该消息
不存储	消息不进行离线缓存，所以只有接收人在线时，才可收到该消息。该消息不进行历史消息云存储、不进入云端存储（Log 日志）、不进行消息同步（消息路由）

推送属性

接收人是否接收推送，当离线属性为 存储 时，该属性生效。离线属性为 不存储 时属性无效。

由于 Web、小程序、PC 端没有推送平台，无法收到推送提醒。

推送属性	平台	推送方式	详细描述
推送	iOS、Android	APNs、华为、小米、魅族、OPPO、vivo、FCM、融云	当有离线缓存消息时，进行远程推送提醒，内容为该推送提醒显示的内容
不推送	iOS、Android	--	当有离线缓存消息时，不进行远程推送提醒

发送普通消息

发送入口

通过消息内容发送

接口原型：

```
- (RCMessage *)sendMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode nErrorCode, long messageId))errorBlock;
```

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

通过消息对象发送

> ⚠ 警告
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock;
```

RCMessage 添加消息配置的对象

```
/*!
消息配置，可用来设置静默消息（不进行通知提醒）
*/
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
消息推送配置，详细查看 配置消息推送
*/
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息对象
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

error 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIM sharedRCIM] sendMessage:message pushContent:pushContent pushData:nil successBlock:nil errorBlock:nil];
```

> ⚠ 警告
>
> 注意 messageid 无特殊情况应传 **0**

文本消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCTextMessage	RCMessageContent	RC:TxtMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
content	NSString	文本消息的内容

代码示例

```
RCTextMessage *textMessage = [RCTextMessage messageWithContent:@"消息内容"];

[[RCIM sharedRCIM] sendMessage:ConversationType_PRIVATE targetId:@"接收方的 userId" content:textMessage
pushContent:nil pushData:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

位置消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCLocationMessage	RCMessageContent	RC:LBSMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
image	UIImage	地理位置的缩略图
location	CLLocationCoordinate2D	地理位置的二维坐标
locationName	NSString	地理位置的名称

代码示例

```
RCLocationMessage *locationMessage = [RCLocationMessage messageWithLocationImage:image location:location
locationName:locationName];

[[RCIM sharedRCIM] sendMessage:ConversationType_PRIVATE targetId:@"接收方的 userId" content:locationMessage
pushContent:nil pushData:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

发送媒体消息

发送入口

通过消息内容发送

接口原型：

```
- (RCMessage *)sendMediaMessage:(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
cancel:(void (^)(long messageId))cancelBlock;
```

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调

参数	类型	必填	说明
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
messageId	long	消息的ID

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

`cancel` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

通过消息对象发送

> **警告**
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMediaMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
progress:(void (^)(int progress, RCMessage *progressMessage))progressBlock
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock
cancel:(void (^)(RCMessage *cancelMessage))cancelBlock;
```

RCMessage 添加消息配置的对象

```
/*!
 消息配置，可用来设置静默消息（不进行通知提醒）
 */
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
 消息推送配置，详细查看 配置消息推送
 */
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息实体
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
progressMessage	RCMessage	消息实体

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

`cancel` 说明：

回调参数	回调类型	说明
cancelMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIM sharedRCIM] sendMediaMessage:message pushContent:pushContent pushData:nil progress:nil successBlock:nil
errorBlock:nil cancel:nil];
```

> ⚠ 警告
>
> 注意 messageid 无特殊情况应传 **0**

图片消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCImageMessage	RCMediaMessageContent	RC:ImgMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
imageData	NSData	图片的原始数据

> ⚠ 警告
>
> 如果想发送原图，请设置属性 full 为 YES。

代码示例

```
RCImageMessage *message = [RCImageMessage messageWithImageData:data];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId" content:message
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

语音消息

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCHQVoiceMessage	RCMediaMessageContent	RC:HQVCMsg	存储	计数	存储	推送	消息内容

构建参数说明：

参数	类型	说明
localPath	NSString	语音的本地路径
duration	long	语音消息的时长

示例代码

```
RCHQVoiceMessage *hqVoiceMsg = [RCHQVoiceMessage messageWithPath:path duration:duration];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId" content:hqVoiceMsg
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

文件消息

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCFileMessage	RCMediaMessageContent	RC:FileMsg	存储	计数	存储	推送	消息内容

构建参数说明：

参数	类型	说明
localPath	NSString	文件的本地路径

代码示例

```
RCFileMessage *fileMessage = [RCFileMessage messageWithFile:path]

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId" content:fileMessage
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

小视频消息

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 。

集成方式

请您参考 **全局配置 > 引入 SDK**，在 `Podfile` 文件中新增下面内容

```
pod 'RongCloudIM/Sight', '~> 2.10.4'
```

> 必须保持 RongCloudIM 所有导入的模块版本号一致。

发送本地视频

开启小视频服务后，SDK 支持从本地相册中选择视频文件发送，需要开发者设置选择媒体资源是否包含视频文件，默认为不包含，设为后支持发送本地视频文件，时长上限为 **2** 分钟，设置方法如下：

属性所属类为 `RCIM`，默认为 `NO`。

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

发送小视频消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCSightMessage	RCMediaMessageContent	RC:SightMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
path	NSData	视频文件本地路径
image	UIImage	视频首帧缩略图

参数	类型	说明
duration	NSUInteger	视频时长， 单位秒

代码示例

```
RCSightMessage *sightMessage = [RCSightMessage messageWithLocalPath:url thumbnail:image duration:duration];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId" content:sightMessage
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

GIF 消息

> ⚠ 警告
>
> 从 SDK 2.9.21 版本开始支持发送 GIF 图片消息，默认可选择发送的 GIF 图片大小为 2M，接收于小于 1M 的消息时自动下载显示，大于 1M 时需要手动点击下载。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCGIFMessage	RCMediaMessageContent	RC:GIFMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
gifImageData	NSData	GIF 图的数据
width	long	GIF 图的宽
height	long	GIF 图的高

代码示例

```
RCGIFMessage *gifMsg = [RCGIFMessage messageWithGIFImageData:gifImageData
width:gifImage.size.width
height:gifImage.size.height];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId" content:gifMsg
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

发送自定义消息

除了使用 SDK 内置消息外，开发者可根据自己的业务需求来自定义消息。

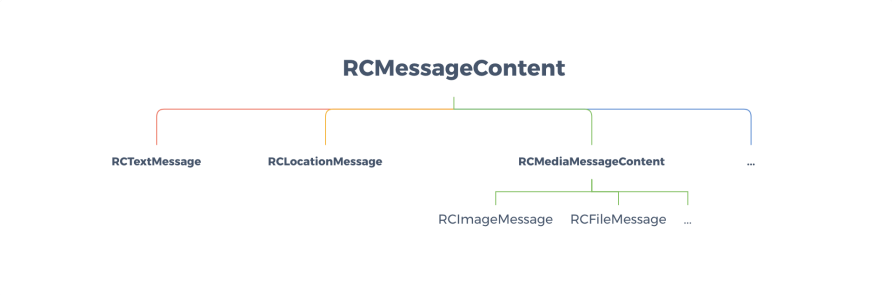
注册消息

注册消息共分为四步。

1. 定义消息类型
2. 注册消息类型
3. 定义消息 Cell
4. 注册消息 Cell

定义消息类型

开发者可根据业务需求来选择继承消息基类 `RCMessageContent` 或者 `RCMediaMessageContent`，两者之前的区别可参考下图:



定义消息类型需要该类遵守下面三个协议

一. 编解码协议

协议原型：

```
@protocol RCMessageCoding <NSObject>
```

`RCMessageCoding` 主要有三个功能：

1. 提供消息唯一标识符
2. 消息发送时将消息中的所有信息编码为 JSON 数据传输

3. 消息接收时将 JSON 数据解码还原为消息对象。

该协议为必须遵守的协议，遵守该协议需实现下面三个方法。

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：

```
+ (NSString *)getObjectName;
```

> ⚠ 警告
>
> 1. 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
> 2. 请勿使用 `RC:` 开头的类型名，以免和SDK默认的消息名称冲突
> 3. 所有自定义消息必须实现此协议，否则将无法正常传输和使用。

二. 存储协议

协议原型：

```
@protocol RCMessagePersistentCompatible <NSObject>
```

`RCMessageCoding` 主要有两个功能:

1. 指明此消息类型在本地和服务端是否存储
2. 指明此消息类是否计入未读消息数

该协议为必须遵守的协议，遵守该协议需实现下面方法:

消息的存储计数策略：

```
+ (RCMessagePersistent)persistentFlag;
```

RCMessagePersistent 说明:

类型	说明
MessagePersistent_NONE	表示消息不存储、不计数。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_ISCOUNTED	表示客户端收到消息后，要进行未读消息计数（未读消息数增加 1）。
MessagePersistent_ISPERSISTED	表示消息会存储到数据库。
MessagePersistent_STATUS	表示状态消息，该类型消息不存储不计数。状态类消息表示的是即时的状态，如果接收方不在线，则无法再收到该状态消息。

自定义消息持久化标识案例分析：

1. 需要存储和未读计数的消息（类似文本消息）：`MessagePersistent_ISCOUNTED | MessagePersistent_ISPERSISTED`
2. 需要确保收到，但不需要展示的消息，多为命令消息（类似撤回命令消息）：`MessagePersistent_NONE`
3. 对方在线接收，不在线就不用接收的消息（类似输入状态消息）：`MessagePersistent_STATUS`

三. 内容摘要协议

协议原型：

```
@protocol RCMessageContentView
```

`RCMessageContentView` 主要有一个功能:

1. 设置此消息在会话列表和本地通知中显示消息的摘要

该协议为非必须遵守的协议，遵守该协议需实现下面方法。

设置消息摘要：

```
- (NSString *)conversationDigest;
```

注册消息类型

开发者在进行完自定义消息类后，需要在 SDK init 之后，注册此自定义消息类。

```
- (void)registerMessageType:(Class)messageClass;
```

⚠ 警告

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。

显示消息

定义消息 Cell

开发者可根据自己的业务需求是否需要显示用户信息来选择继承消息 Cell。

1. 不包含用户头像和昵称

类名：

```
RCMessageBaseCell
```

结构图：

控件需要添加在 `baseContentView` 上，建议在 `baseContentView` 上方预留 **10**

2. 含用户头像和昵称

类名：

```
RCMessageCell
```

结构图：

控件需要添加在 `messageContentView` 上，根据需求在画 Cell 视图布局的时候调整 `messageContentView` 的 frame ，如果是接收方，只需要修改 `messageContentView` 的 `width` 和 `height`，如果是发送方，需要修改 `messageContentView` 的 `x` 、 `width` 和 `height`。

开发者需要在 Cell 的初始化中布局，并重写下面方法来返回 Cell 的 `Size`。当显示自定义消息时，必须实现该方法来返回 Cell 的 `Size`。

参数说明：

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

参数	类型	说明
extraHeight	CGFloat	Cell 内容区域之外的高度

方法原型：

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

注：`extraHeight` 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`

注册消息 Cell

SDK 需要开发者在会话页面注册您定义的消息 Cell 并绑定您的消息类型

```
- (void)registerClass:(Class)cellClass forMessageClass:(Class)messageClass;
```

发送消息

自定义消息继承 `RCMessageContent`，请使用 发送普通消息 的接口进行发送。

自定义消息继承 `RCMediaMessageContent`，请使用 发送媒体消息 的接口进行发送。

配置消息推送

RCMessagePushConfig 属性介绍

> 提示
>
> 从 SDK 4.0.4 版本开始支持此功能，针对每条 Message 都可以设置此属性，详细查看以下参数说明

参数	类型	说明
pushTitle	NSString	推送标题，如果没有设置，会使用 SDK 默认的标题显示规则
pushContent	NSString	推送内容，如果没有，则使用发送消息的 pushContent，最后则会使用 SDK 默认的标题显示规则
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
iOSConfig	RCiOSConfig	iOS 平台相关配置
androidConfig	RCAndroidConfig	Android 平台相关配置

RCiOSConfig 属性介绍

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）

参数	类型	说明
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）

RCAndroidConfig 属性介绍

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
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
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
typeVivo	NSString	VIVO 推送通道类型，开发者集成了 VIVO 推送，需要指定推送类型时，可进行设置。 目前可选值 "0"(运营消息) 和 "1"(系统消息)

Channel ID 需要由开发者进行创建，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

示例代码

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
RCMessagePushConfig *pushConfig = [[RCMessagePushConfig alloc] init];
pushConfig.pushTitle = @"通知标题";
pushConfig.pushContent = @"通知内容";
pushConfig.pushData = @"通知的 pushData";
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.androidConfig.notificationId = @"Android 的通知 id";
pushConfig.androidConfig.channelIdMi = @"小米的 channelId";
pushConfig.androidConfig.channelIdHW = @"华为的 channelId";
pushConfig.androidConfig.channelIdOPPO = @"OPPO 的 channelId";
pushConfig.androidConfig.typeVivo = @"vivo 的类型";
pushConfig.forceShowDetailContent = true;
message.messagePushConfig = pushConfig;
[[RCIMClient sharedRCIMClient] sendMessage:message pushContent:nil pushData:nil successBlock:nil
errorBlock:nil];
```

消息接收

功能描述

更新时间:2024-08-30

开发者可通过此接口拦截到 SDK 接收到的消息，并进行响应的业务操作。

实现方法

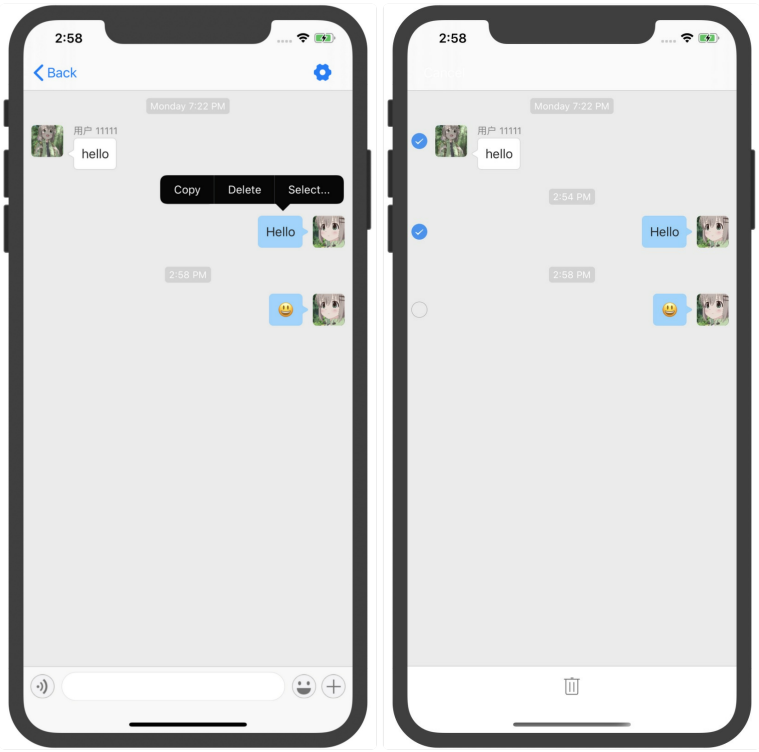
SDK 通过消息监听接收消息，详细可参考 设置消息监听 文档

消息多选 功能描述

更新时间:2024-08-30

开发者在长按消息后出现更多选项，能够多选消息，可以自定义点击事件，SDK 默认实现多选消息转发和删除功能。

效果展示



自定义多选

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，可通过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,才会有有效值

```
@property(nonatomic, strong, readonly) NSArray<RCMessageModel *> *selectedMessages;
```

3. 底部视图

进入多选状态后，页面底部出现的工具视图,开发者可以添加对多选消息的操作事件，如转发、收藏等，可以在 viewDidLoad 中 通过给 messageSelectionToolbar 添加 UIBarButtonItem 的方式添加事件。

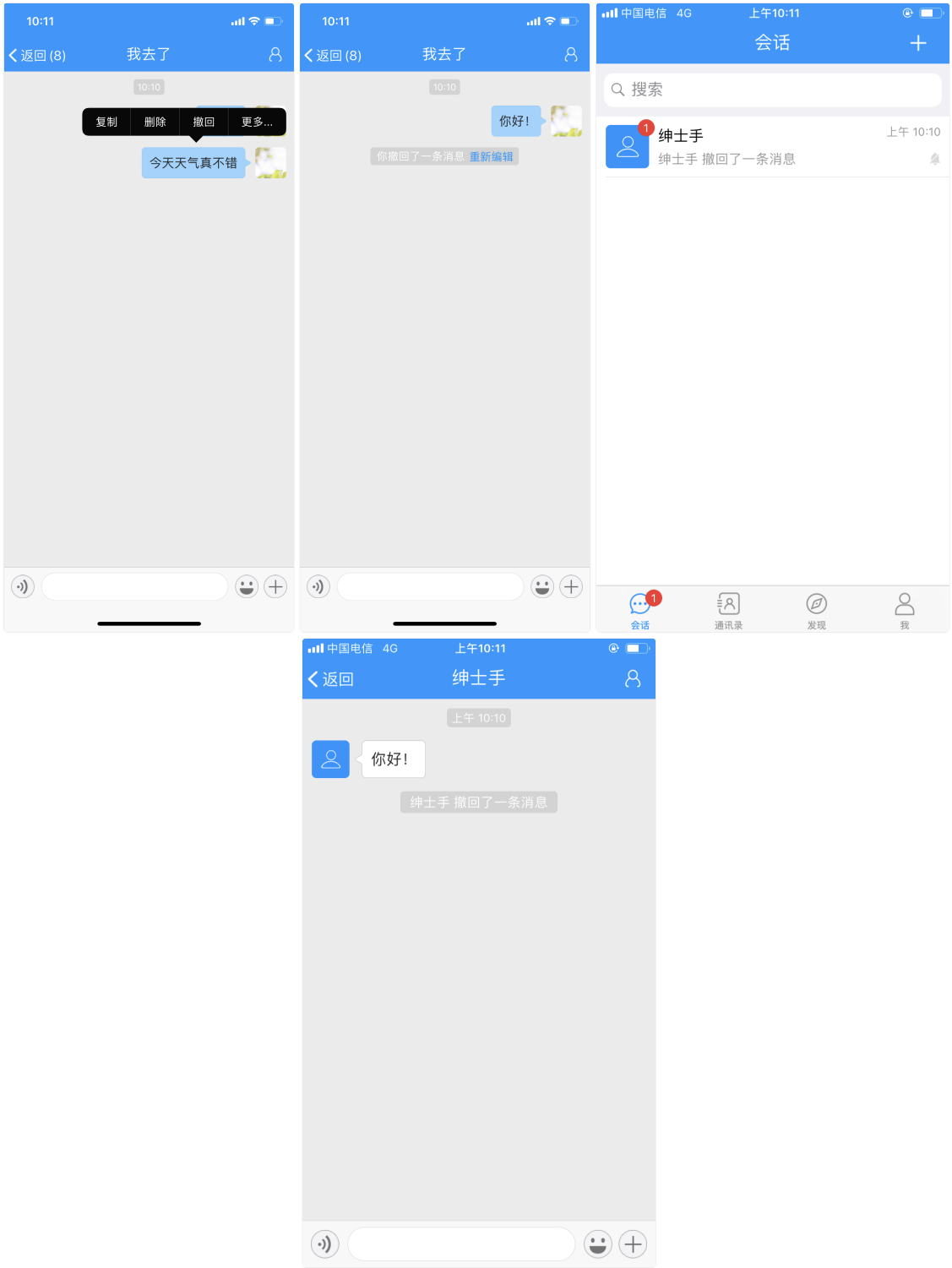
```
@property(nonatomic, strong) UIToolbar *messageSelectionToolbar;
```

消息撤回 功能描述

更新时间:2024-08-30

用户在消息发送成功后的有效时间内长按消息，在弹出菜单中选择“撤回消息”来将这条消息撤回。这条消息将在发送端删除，并提示“你撤回了一条消息”，同时发送撤回指令给接收端，接收端收到撤回指令后，也会把同一条消息删除，提示“XX撤回了一条消息”。

效果展示



配置说明

1. **是否开启撤回功能**

默认值是 YES

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

2. **消息可撤回的最大时间**

单位是秒，默认值是 120 秒。

```
@property (nonatomic, assign) NSUInteger maxRecallDuration;
```

3. **消息撤回后可重新编辑的时间**

单位是秒，默认值是 300 秒。

```
@property (nonatomic, assign) NSUInteger reeditDuration;
```

> ⚠ 警告
>
> 目前消息撤回后重新编辑仅为本地操作，卸载重装或者更换设备不会同步。

通知回调

1. **消息被撤回的通知**

消息被撤回后，SDK 会分发此通知，`Notification` 的 `object` 为 `NSNumber` 的 `messageId` 。

通知原型：

```
FOUNDATION_EXPORT NSString *const RCKitDispatchRecallMessageNotification;
```

2. **消息被撤回的回调**

被撤回的消息会变更为 `RCRecallNotificationMessage` ，App 需要在 UI 上刷新这条消息。

需要您遵守 `RCIMReceiveMessageDelegate` 协议

参数说明：

参数	类型	说明
messageId	long	被撤回的消息ID

协议方法：

```
- (void)onRCIMMessageRecalled:(long)messageId;
```

3. **点击撤回消息Cell中重新编辑的回调**

点击撤回消息 Cell 中重新编辑，会调用此回调，且不会再触发 `didTapMessageCell`。

参数说明：

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型

回调原型：

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

自定义撤回

开发者也可通过下面接口根据业务需求进行撤回消息

参数说明：

参数	类型	必填	说明
messageId	long	是	被撤回消息的消息Id

方法原型：

```
- (void)recallMessage:(long)messageId;
```

此方法为 `RCConversationViewController` 类的实例方法。

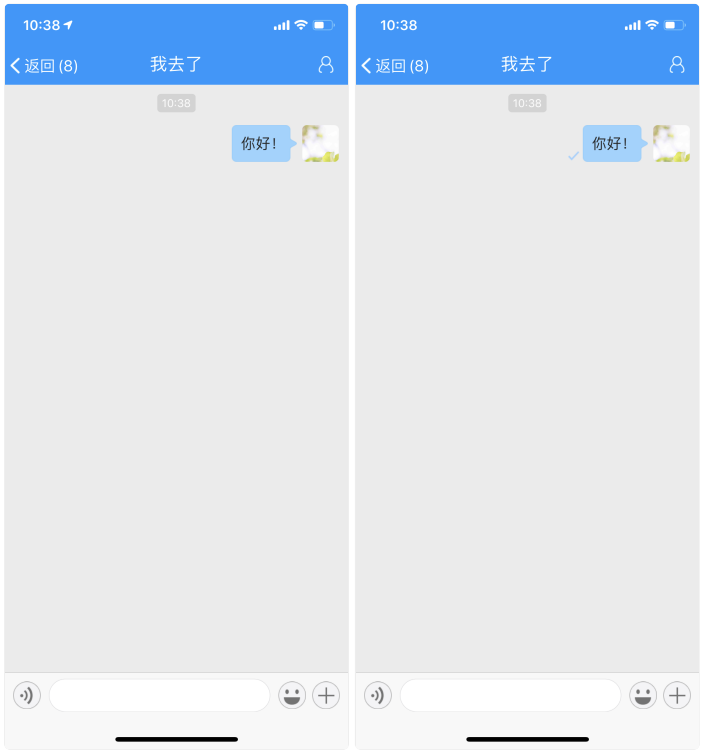
> ⚠ 警告
>
> 只有存储并发送成功的消息才可以撤回。

消息回执 功能描述

更新时间:2024-08-30

用户通过回执消息，可以知道对方是否阅读该消息。

效果展示



配置说明

属性所属类为 RCIM，默认为关闭状态。

```
@property(nonatomic, copy) NSArray* enabledReadReceiptConversationTypeList;
```

代码示例

```
[RCIM sharedRCIM].enabledReadReceiptConversationTypeList = @[ @(ConversationType_PRIVATE) ];
```

自定义显示

SDK 会话页面中文本消息已读的 UI 默认是一个“对勾”图标，开发者可参考下面内容修改为文本“已读”或“未读”。

思路：

1. 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”。

2. 需要监听消息发送状态更新的通知，更新 cell。

解决方案：

下面代码以单聊文本消息为例，如需多更多消息类型修改，可以自行添加判断修改 UI。

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) {
for (UIView *view in [((RCMessageCell *)cell).messageHasReadStatusView subviews]) {
[view removeFromSuperview];
}
UILabel *hasReadView = [[UILabel alloc] initWithFrame:CGRectMake(-18, 5, 30, 20)];
hasReadView.textAlignment = NSTextAlignmentRight;
hasReadView.font = [UIFont systemFontOfSize:12];
hasReadView.textColor = [UIColor blackColor];
((RCMessageCell *)cell).messageHasReadStatusView.hidden = NO;
if (model.sentStatus == SentStatus_READ) {
CGRect hasReadViewFrame = hasReadView.frame;
hasReadViewFrame.origin.y -= 19;
hasReadView.frame = hasReadViewFrame;
hasReadView.text = @"已读";
[((RCMessageCell *)cell).messageHasReadStatusView addSubview:hasReadView];
} else if (model.sentStatus == SentStatus_SENT) {
hasReadView.text = @"未读";
[((RCMessageCell *)cell).messageHasReadStatusView addSubview:hasReadView];
}
}
} else {
for (UIView *view in [((RCMessageCell *)cell).messageHasReadStatusView subviews]) {
[view 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;
}
}
});
}
```

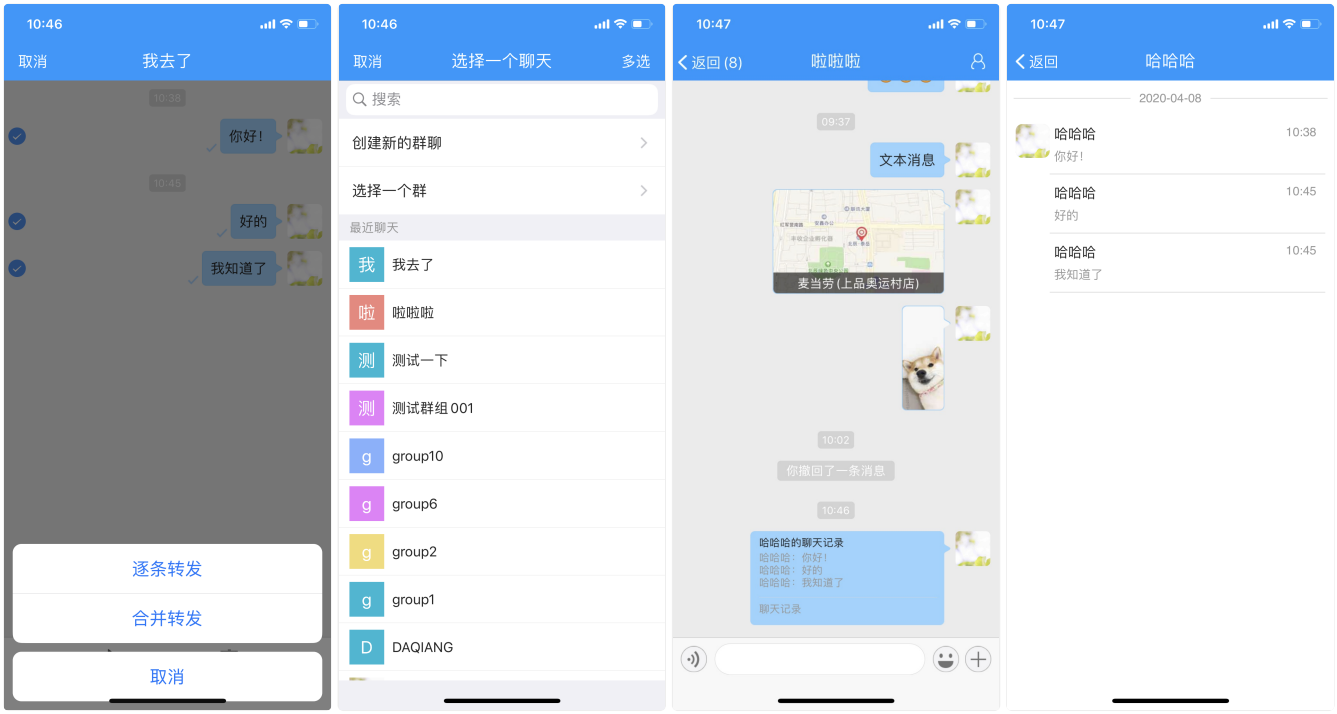
```
- (void)hideCellReceiptView:(NSIndexPath *)indexPath withMessageModel:(RCMessageModel *)model {
UICollectionViewCell *__cell = [self.conversationMessageCollectionView cellForItemAtIndexPath:indexPath];
[self.conversationMessageCollectionView reloadItemsAtIndexPaths:@[ indexPath ]];
//如果是空说明被回收了，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

消息转发 功能描述

更新时间:2024-08-30

从 2.9.25 版本开始 IMKit SDK 支持合并转发功能，目前不支持红包、阅后即焚及自定义消息的合并转发功能。

效果展示



配置说明

属性所属类为 RCIM，默认为 NO。

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

> ⚠ 警告
>
> 目前不支持红包、阅后即焚及自定义消息的合并转发功能。

方法说明

开启合并转发功能之后，RongIMKit 默认的选择会话的界面取的数据是数据库中会话表的全部会话。如果不满足需求，可以通过重写 RCConversationViewController.h 中的如下方法，来自行实现选择会话界面，并把选中后的会话列表回传给 RongIMKit。

参数说明：

参数	类型	说明
index	NSInteger	0 是逐条转发消息, 1 是合并转发消息。
completedBlock	Block	返回需要转发到的会话的列表。

回调参数说明：

参数	类型	说明
completedBlock	NSArray<RCConversation *>	返回需要转发到的会话的列表。

方法原型：

```
- (void)forwardMessage:(NSInteger)index completed:(void (^)(NSArray<RCConversation *>
*conversationList))completedBlock;
```

代码示例：

```
- (void)forwardMessage:(NSInteger)index completed:(void (^)(NSArray<RCConversation *> *))completedBlock {

// conversationList 为您返回的 list 数组
completedBlock(conversationList);
}
```

消息引用

功能描述

更新时间:2024-08-30

开发者可通过下面配置来管理是否支持消息引用。

> ⚠ 警告
>
> 目前仅支持文本消息、文件消息、图文消息、图片消息、引用消息的引用

配置说明

属性所属类为 RCIM，默认为开启状态。

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

代码示例

```
[RCIM sharedRCIM].enableMessageReference = YES;
```

消息删除 功能描述

更新时间:2024-08-30

SDK 内部已经集成了删除消息。长按消息后会弹出功能选择的菜单，点击删除后，SDK 会将此消息在本地数据库删除。

效果展示

自定义删除

开发者也可自定义删除功能。删除消息的接口使用可参考 消息删除 。

消息搜索 功能描述

更新时间:2024-08-30

开发者可根据下面方法实现消息搜索功能。

实现方法

SDK 未提供内置的 UI 组件，开发者需根据业务需求参考 SealTalk 开源项目 中的示例代码实现 UI 功能，之后调用 消息搜索接口 接口即可。

消息插入 功能描述

更新时间:2024-08-30

开发者可通过下面方法在会话页面内插入一条消息。通过此方法插入的消息，会将消息实体对应的内容插入数据源中，并更新 UI。

插入消息

参数说明

参数	类型	必填	说明
message	RCMessage	是	消息实体

代码示例

此方法为 `RCConversationViewController` 的实例方法。

```
[self appendAndDisplayMessage:message];
```

常见问题

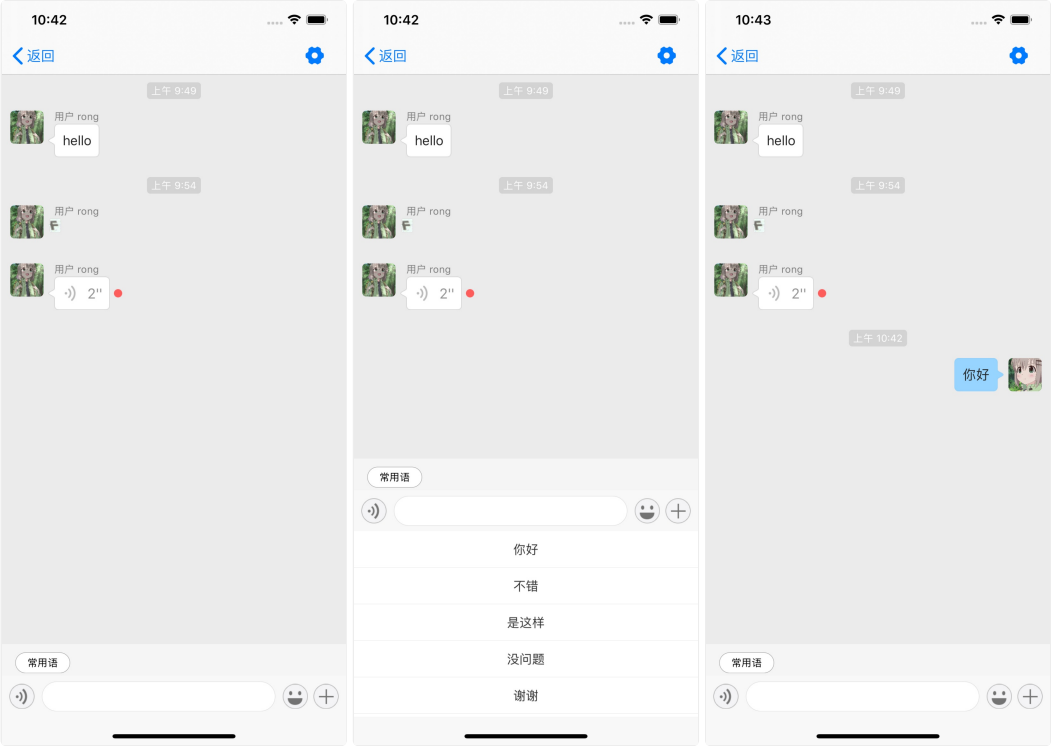
Q1: 插入的消息退出页面再进来就不在了。
A1: 这条消息只会在 UI 上插入，并不会存入数据库，当退出会话页面再次进入的时候，这条消息将不再显示。用户需要在数据库中同时进行存储，可先调用 消息插入 的接口在数据库插入消息，然后将返回的 RCMessage 对象再次调用下面方法插入到 UI。

快捷回复 功能描述

更新时间:2024-08-30

> ⚠ 警告
>
> 从 SDK **2.9.21** 版本开始支持单聊消息快捷回复功能。

效果展示



代码示例

会话界面直接添加如下代码即可：

```
[self.chatSessionInputBarControl setCommonPhrasesList:@[@"你好", @"不错", @"是这样", @"没问题", @"谢谢"]];
```

输入状态 功能描述

更新时间:2024-08-30

输入消息的时候对方可以看到正在输入的提示，当输入框中有内容变化时，向目标用户发送一条正在输入的状态消息，为防止消息发送频繁 6 秒钟内变化多次时，只发送一条消息。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，如不需要此功能可进行关闭。

属性说明

属性所属类为 RCIM，默认为 YES。

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

代码示例

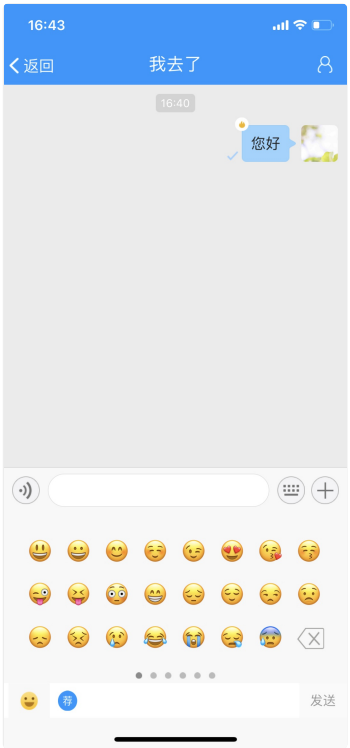
```
[RCIM sharedRCIM].enableTypingStatus = YES;
```

表情区域 Emoji 表情

更新时间:2024-08-30

Emoji 是 SDK 默认访问资源文件中的 `Emoji.plist` 文件生成的且不支持移除。

效果展示



自定义表情

自定义表情需遵守 `RCEmoticonTabSource` 协议

1. 需要用户返回对应的表情View

参数说明：

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

代理方法：

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

2. 在会话页面添加自定义表情

开发者需要通过 `emojiBoardView` 的 `addEmojiTab` 方法添加表情包。

```
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];
```

相关示例代码可参考 SealTalk 开源项目 中的 RCDCustomerEmoticonTab。

加号区域 功能描述

更新时间:2024-08-30

SDK 支持添加、修改、删除扩展板上的每一个扩展项，包括修改扩展项的图片、文案以及点击之后的响应事件处理。

> ⚠ 警告
>
> 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。

内置扩展项

扩展项介绍

常用扩展项介绍：

定义名	Tag 值	说明
PLUGIN_BOARD_ITEM_ALBUM_TAG	1001	相册
PLUGIN_BOARD_ITEM_CAMERA_TAG	1002	拍照
PLUGIN_BOARD_ITEM_LOCATION_TAG	1003	位置
PLUGIN_BOARD_ITEM_FILE_TAG	1006	文件

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

常用扩展项

1. **添加文件功能**

加号区域增加发送文件功能，SDK 中已经默认实现了该功能，但是为了 SDK 向后兼容性，目前 SDK 默认不开启该入口，可以参考以下代码在加号区域中增加发送文件功能。

```
UIImage *imageFile = [RCKitUtility imageNamed:@"actionbar_file_icon" ofBundle:@"RongCloud.bundle"];
RCPluginBoardView *pluginBoardView = self.chatSessionInputBarControl.pluginBoardView;
[pluginBoardView insertItemWithImage:imageFile
title:NSLocalizedStringFromTable(@"File", @"RongCloudKit", nil)
atIndex:3
tag:PLUGIN_BOARD_ITEM_FILE_TAG];
```

2. **添加音视频功能**

音视频功能需要在 控制台 开通音视频服务，SDK 会自动增加音视频通话扩展项。

添加扩展项

参数说明

参数	类型	必填	说明
image	UIImage	是	扩展项的展示图片
title	NSString	是	扩展项的展示标题
index	NSInteger	是	需要添加到的索引值
tag	NSInteger	是	扩展项的唯一标示符

代码示例

```
[self.chatSessionInputBarControl.pluginBoardView insertItemWithImage:[UIImage imageNamed:@"poke_plugin_item"]
title:@"示例1"
atIndex:4
tag:2001];
```

更新指定扩展项

参数说明

参数	类型	必填	说明
image	UIImage	是	扩展项的展示图片
title	NSString	是	扩展项的展示标题
tag	NSInteger	是	扩展项的唯一标示符

代码示例

```
[self.chatSessionInputBarControl.pluginBoardView updateItemWithTag:101 image:newImage title:newTitle];
```

删除指定扩展项

参数说明

参数	类型	必填	说明
tag	NSInteger	是	扩展项的唯一标示符

代码示例

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

点击扩展项

参数说明

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)的唯一标示

回调方法

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

用户名片 功能描述

更新时间:2024-08-30

为满足客户发送个人名片的功能需求，融云实现了个人名片的开源插件，开发者可下载插件直接集成使用。

效果展示

集成方式

参考 plugin-contactcard-ios

自定义 UI

自定义头像大小

更新时间:2024-08-30

开发者可根据业务场景自行定义会话页面头像的显示形状和大小

头像显示形状

属性所属类为 RCIM，默认值为矩形

```
@property (nonatomic) RCUserAvatarStyle globalMessageAvatarStyle;
```

RCUserAvatarStyle 说明：

类型名	说明
RC_USER_AVATAR_RECTANGLE	矩形
RC_USER_AVATAR_CYCLE	圆形

头像显示大小

属性所属类为 RCIM，默认值为 40*40

```
@property (nonatomic) CGSize globalMessagePortraitSize;
```

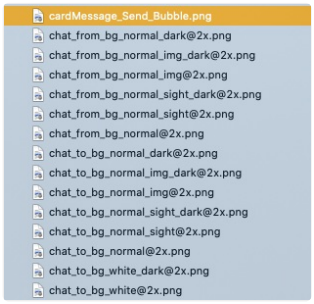
自定义聊天气泡

开发者可根据业务场景自行定义聊天气泡的样式，可以在 SDK 的资源包 (IMKit/Resources/Rongcloud.bundle) 中找到对应气泡图片，然后替换成您自己的图片即可。另外需要注意拉伸位置。

SDK 内拉升比例为：

```
UIEdgeInsetsMake(image.size.height * 0.8, image.size.width * 0.2, image.size.height * 0.2, image.size.width *
0.8)
```

bundle 内的图片资源：



自定义按导航按钮

相册列表页

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。

color.plist 文件路径:`IMKit/Resources/RCColor.plist`

相册预览页

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 `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。

位置选取页

左上、右上角按钮:

可设置 `RCIM` 中的 `globalNavigationBarTintColor` 来更改。

发送文件页

左上、右上角按钮:

可设置 `RCIM` 中的 `globalNavigationBarTintColor` 来更改。

右上角按钮未选择文件时颜色：

可 `color.plist` 中 `fileSelect` 字典里的相关字段来更改。

右上角按钮选中文件时的颜色：

可设置 RCIM 中的 globalNavigationBarTintColor 来更改。

个人名片页

左上角按钮：

可设置 RCIM 中的 globalNavigationBarTintColor 来更改。

未读数显示

右上角提示

当收到的消息超过一个屏幕时，进入会话之后，是否在右上角提示上方存在的未读消息数，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。

是否开启：

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

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：

开发者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;
}
}];
}
```

右下角提示

当前阅读区域的下方收到消息时，是否在会话页面的右下角提示下方存在未读消息，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，此会话收到消息时，会在右下角显示未读消息提示，而不会自动滚动到最下方，用户点击该提醒按钮，会滚动到最下方。

是否开启：

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

提示的 UILabel ：

当 unReadMessage > 10 右上角会显示未读消息数。

```
@property (nonatomic, strong) UILabel *unReadNewMessageLabel;
```

位置实时共享 功能描述

更新时间:2024-08-30

开发者可根据下面步骤集成位置共享功能。

效果展示

集成步骤

下面以 Demo 中的代码为例，说明集成步骤。以下代码，可以在 SealTalk 的 `RCDChatViewController.m` 中找到。

SealTalk 源码下载地址

1. **导入功能包**

将源码中的 Sections/Chat/RealTimeLocation 导入工程

2. **引入头文件**

在会话页面中导入下面头文件

```
#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. **设置协议和代理**

```
@interface RCDChatViewController () < RCRealTimeLocationObserver,
RealTimeLocationStatusViewDelegate>
@property(nonatomic, weak) id<RCRealTimeLocationProxy> realTimeLocation;
@property(nonatomic, strong)
RealTimeLocationStatusView *realTimeLocationStatusView;
```

4. **初始化状态 View**

```
- (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. **注册消息**

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

6. **获取服务**

获取实时位置共享服务

```
//RCDChatViewController Class

//获取实时位置共享代理
- (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);
}];
}
```

7. **弹出位置实时共享**

在点击位置时弹出位置实时共享

```
//RCDChatViewController Class

- (void)pluginBoardView:(RCPluginBoardView *)pluginBoardView
clickedItemWithTag:(NSInteger)tag {
switch (tag) {
case PLUGIN_BOARD_ITEM_LOCATION_TAG: {
if (self.realTimeLocation) {
UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:RCDLocalizedString(@"cancel")
style:UIAlertActionStyleCancel
handler:nil];
UIAlertAction *sendLocationAction =
[UIAlertAction actionWithTitle:RCDLocalizedString(@"send_location")
style:UIAlertActionStyleDefault
handler:^(UIAlertAction *_Nonnull action) {
[super pluginBoardView:self.chatSessionInputBarControl.pluginBoardView
clickedItemWithTag:PLUGIN_BOARD_ITEM_LOCATION_TAG];
}];
UIAlertAction *locationShareAction = [UIAlertAction actionWithTitle:RCDLocalizedString(@"location_share")
style:UIAlertActionStyleDefault
handler:^(UIAlertAction *_Nonnull action) {
[self showRealTimeLocationViewController];
}];

[RCKitUtility showAlertController:nil
message:nil
preferredStyle:UIAlertControllerStyleActionSheet
actions:@[ cancelAction, sendLocationAction, locationShareAction ]
inViewController:self];
} else {
[super pluginBoardView:pluginBoardView clickedItemWithTag:tag];
}
} break;
```

8. **代理方法**

实时位置共享监听代理方法

```
//RCDChatViewController Class

- (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:[RCIMClient sharedRCIMClient].currentUserInfo.userId]) {
[self notifyParticipantChange:RCDLocalizedString(@"you_join_location_share")];
} else {
```

```
[[RCIM sharedRCIM]
.userInfoDataSource
getUserInfoWithUserId:userId
completion:^(RCUserInfo *userInfo) {
if (userInfo.name.length) {
[weakSelf notifyParticipantChange:[NSString
stringWithFormat:RCDLocalizedString(@"someone_join_share_location"),userInfo.name]];
} else {
[weakSelf notifyParticipantChange:[NSString
stringWithFormat:RCDLocalizedString(@"user_join_share_location"),userId]];
}
}];
}
}

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

当前类中相关私有方法

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

⚠ 警告

1. 实时位置共享功能仅支持单聊。
2. 将上面所有方法以及方法中相关的代码都拷贝到工程中，就可以正常使用位置实时共享功能了。
3. 参与位置实时共享人员的用户信息需要提前设置好，具体方法参考 设置用户信息。

多端阅读消息同步

功能描述

更新时间:2024-08-30

开启之后，用户在其他端上阅读过的消息，当前客户端会清掉该消息的未读数。

配置说明

默认值为 YES。

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

代码示例

```
[RCIM sharedRCIM].enableSyncReadStatus = YES;
```

群聊介绍 概述

更新时间:2024-08-30

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。

主要功能

功能	描述
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。
消息提醒	离线状态，群组中有新消息时，支持 Push 通知。
本地存储	存储在移动端本地，提供本地消息搜索功能。
历史消息	提供服务端消息存储存功能，需开通单群聊消息云存储，默认存储时长为 6 个月。
消息删除	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。
消息搜索	支持按关键字或用户搜索本地指定会话的消息内容。
群消息阅读回执	发送群消息后如需要查看消息的阅读状态，可以使用此功能来发送阅读回执请求。
消息撤回	消息发送成功后，在有效时间内可撤回该条消息，默认可撤回时间为 2 分钟，时间可配置。
群聊会话免打扰	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。
创建群组	App 内的群组数量没有限制，默认一个群上限为 3000 人，可调整群上限，需要提交工单申请开通。
加入群组	每个群最大至 3000 人，一个用户可加入多个群组，没有限制。默认加入群组后，只能查看加入后群组中产生的消息，如需要查看加入前的消息，则需要开通单群聊消息云存储后，再开通查看加入前群消息功能
退出群组	将用户从群中移除，不再接收该群组的消息。
解散群组	将指定群组解散，所有成员都无法再接收该群的消息。
群成员查询	获取指定群组中群成员用户 Id。
刷新群组信息	目前支持更新群组名称。
同步用户所属群组	在集成融云前 App Server 已有群组数据，可使用此服务进行同步。
群组成员禁言	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。
群组整体禁言	指定群组所有成员不能发送消息，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。
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。
发送群组消息	向群组中所有成员发送消息。
发送群组定向消息	向群中指定的一个或多个用户发送消息，群中其他用户无法收到该消息。
发送群组@消息	发送群组中指定群成员需要特别关注的消息。
发送群组状态消息	群组中在线用户会收到此条消息，离线用户不会再收到此条消息，状态消息不计数、不存储。

与聊天室的区别

融云提供群组与聊天室业务，其主要区别如下，客户可根据自己的业务场景进行选择：

功能	群组（group）	聊天室（Chatroom）
场景	类似微信的群组，无论是否在线都会接收消息	只有在线用户可接收消息，可用于直播、社区、游戏、广场交友、兴趣讨论等场景。
离线缓存消息	支持离线消息存储，存储时间可设置（1 ～ 7 天），默认存储 7 天。	无离线消息，只有在线用户才可收到聊天室消息
人数限制	默认一个群上限为 3000 人	聊天室人数无上限
消息提醒	离线状态，群组中有新消息时，支持远程推送（PUSH）通知	离开聊天室后不再接收消息
本地存储	移动端本地数据库存储，提供本地消息搜索接口	退出聊天室后同时删除本地聊天室消息，不支持消息搜索功能
云端存储	需开通单群聊消息云存储，可以提供 6 - 36 个月存储服务	需开通聊天室消息云存储，可以提供 2 - 36 个月存储服务
用户加入限制	一个用户可加入多个群组，无限制	默认一个用户只能加入一个聊天室，加入多个聊天室功能可在控制台自行开通
加入后消息获取逻辑	默认加入群组后，只能查看加入后群组中产生的消息。如需要查看群历史消息，则需要开通单群聊消息云存储后，再开通“查看加入前群消息”功能	加入后可获取聊天室中最新的 50 条消息。
销毁/解散逻辑	需要通过 AppServer 自行调用解散群组接口。	提供销毁聊天室接口，可通过 AppServer 调用。同时聊天室中 1 小时内没有消息产生时，将自动销毁聊天室。
消息可靠度	100% 可靠，不丢消息。	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。
相关接口调用	SDK 不提供群组管理功能接口，通过 Server API 提供群组功能接口。	SDK 和 Server API 同时提供功能接口，销毁聊天室操作只能通过 Server API 方式调用。
发送消息频率	每个客户端 5 条/秒；服务端调用，20 条/秒	每个客户端 5 条/秒；服务端调用，100 条/秒

消息类型

消息类型	描述
文字消息	用来发送文字类消息，其中可以包括表情、超链接（会自动识别），客户端收到消息后计入未读消息数、进行存储。
语音消息	发送高质量的短语音消息，录制的语音文件存储到融云服务端，语音文件格式为 AAC，时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。
图片消息	用来发送图片类消息，客户端收到消息后计入未读消息数、进行存储。图片缩略图格式为 JPG，大小建议不超过 100k。
GIF 图片消息	用来发送 GIF 动态图片消息，客户端收到消息后计入未读消息数、进行存储。
图文消息	用来发送图文消息，包含一个标题，一段文字内容和一张图片，客户端收到消息后计入未读消息数、进行存储。
文件消息	用来发送文件类消息，客户端收到消息后计入未读消息数、进行存储。
位置消息	用来发送地理位置消息，客户端收到消息后计入未读消息数、进行存储。
小视频消息	用来发送小视频消息，支持录制发送及选择本地视频文件发送两种方式，录制时长不超过 10 秒，本地选择视频文件方式时长不超过 2 分钟，小视频消息小视频文件格式为 .mp4，客户端收到消息后计入未读消息数、进行存储。
合并转发消息	IMKit SDK 中支持将多条消息合并为一条消息进行发送，合并后的消息以 HTML 文件的方式存储到融云服务端，客户端收到消息后计入未读消息数、进行存储。红包、阅后即焚及自定义消息的合并转发功能
命令消息	用来发送通用的指令通知消息，消息内可以定义任意 JSON 内容，与通用命令通知消息的区别是不存储、不计数，此类型消息没有 Push 通知。

消息类型	描述
自定义消息	融云内置消息类型，无法满足客户业务需求时，可通过自定义消息类型进行实现，接收自定义消息的格式解析及展示处理需要开发者自行实现

群信息 设置群信息

更新时间:2024-08-30

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。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，会调用此方法请求群组信息用于显示。

实现此功能需要开发者遵守 `RCIMGroupInfoDataSource` 协议。

参数说明

代理对象参数说明：

参数	类型	必填	说明
groupInfoDataSource	id <RCIMGroupInfoDataSource>	是	群组信息提供者

协议方法参数说明：

参数	类型	说明
groupId	NSString	群组ID
completion	void	获取群组信息完成之后需要执行的Block

回调参数说明:

参数	类型	必填	说明
userInfo	RCGroup	是	该群组ID对应的群组信息

代码示例

1. 设置代理对象:

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

2. 实现协议方法:

```
- (void)getGroupInfoWithGroupId:(NSString *)groupId completion:(void (^)(RCGroup *))completion {
if ([groupId length] == 0)
return;
//开发者调自己的服务器接口根据 userID 异步请求数据
[RCDHTTPTOOL getGroupByID:groupId
successCompletion:^(RCDGroupInfo *group) {
completion(group);
}];
}
```

群成员列表

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。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，会调用此方法请求群组成员用于显示。

实现此功能需要开发者遵守 `RCIMGroupMemberDataSource` 协议。

参数说明

代理对象参数说明：

参数	类型	必填	说明
groupMemberDataSource	id <RCIMGroupMemberDataSource>	是	群成员信息提供者

协议方法参数说明：

参数	类型	说明
groupId	NSString	群组ID
resultBlock	void	获取群组信息完成之后需要执行的Block

回调参数说明:

参数	类型	必填	说明
userIdList	NSArray<NSString *>	是	该群组ID对应的群组信息

代码示例

1. 设置代理对象:

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

2. 实现协议方法:

```
- (void)getAllMembersOfGroup:(NSString *)groupId result:(void (^)(NSArray<NSString *> *))resultBlock {
//开发者回调自己服务器获取信息
[RCDGroupManager getGroupMembersFromServer:groupId
complete:^(NSArray<NSString *> *_Nonnull memberIdList) {
if (resultBlock) {
resultBlock(memberIdList);
}
}];
}
```

群名片信息

SDK 通过此方法群组中的成员列表，请在 `resultBlock` 中返回该群组中某个群成员具体信息。设置了群名片信息提供者之后，SDK 在需要获取群成员信息的时候，会调用此方法请求群成员信息用于显示。

实现此功能需要开发者遵守 `RCIMGroupUserInfoDataSource` 协议。

参数说明

代理对象参数说明：

参数	类型	必填	说明
groupUserInfoDataSource	id <RCIMGroupUserInfoDataSource>	是	群成员信息提供者

协议方法参数说明：

参数	类型	说明
userId	NSString	用户ID
groupId	NSString	群组ID
completion	block	获取群名片信息完成之后需要执行的Block

回调参数说明:

参数	类型	必填	说明
userInfo	RCUserInfo	是	获取群成员信息完成之后需要执行的Block

代码示例

1. 设置代理对象:

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

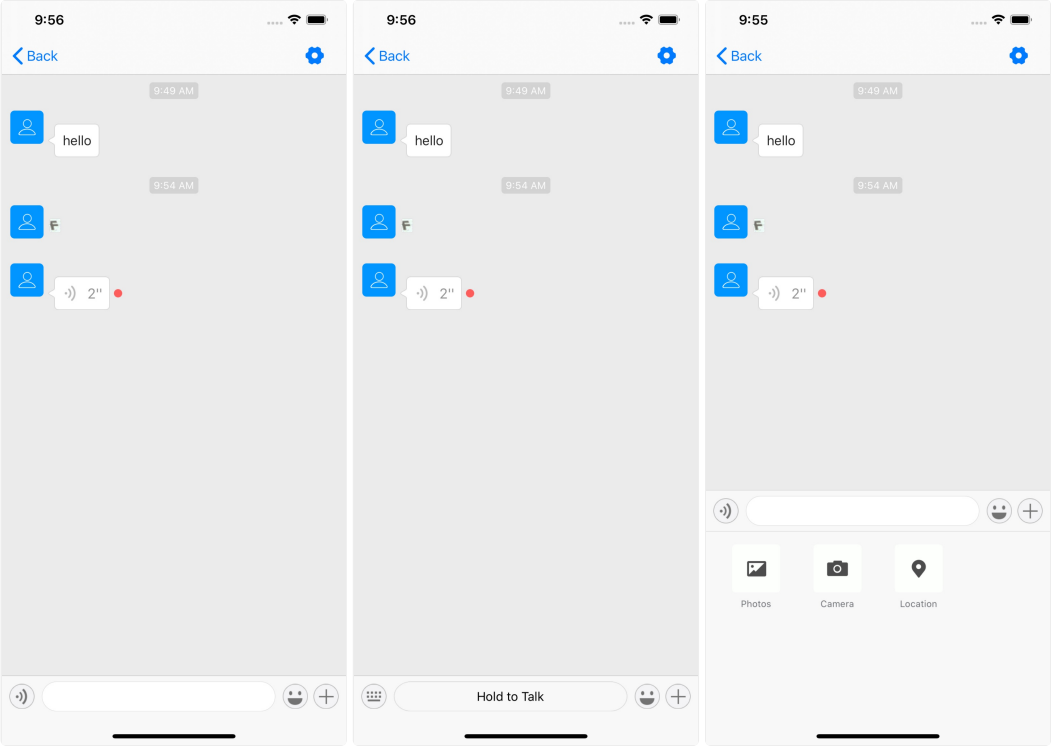
2. 实现协议方法:

```
- (void)getUserInfoWithUserId:(NSString *)userId
inGroup:(NSString *)groupId
completion:(void (^)(RCUserInfo *userInfo))completion {
//开发者回调自己服务器获取信息
[RCDGroupManager getGroupMemberDetailInfoFromServer:userId
groupId:groupId
complete:^(RCUserInfo *userInfo) {
return completion(userInfo);
}];
}
```

会话界面 效果展示

更新时间:2024-08-30

图中未展示发送者昵称和头像是因为未设置用户信息，具体如何设置请参考 设置用户信息



构建会话界面

```
/*!
会话页面类
*/
RCConversationViewController
```

构建说明

开发者可通过两种方式使用会话列表页面。

1. 直接使用此类

   建议仅在排查会话列表的相关错误时使用，如果在使用此页面期间出现了相关问题，可以直接 push 到此类排查是否能复现相关问题。

2. 继承使用此类

   开发者可继承此类，并跳转到此页面来开发相关内容。

参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID

方法原型

```
- (id)initWithConversationType:(RCConversationType)conversationType targetId:(NSString *)targetId;
```

代码示例

```
RCDConversationViewController *conversationVC = [[RCDConversationViewController alloc]
initWithConversationType:ConversationType_PRIVATE targetId:@"targetId"];
[self.navigationController pushViewController:conversationVC animated:YES];
```

界面事件回调

输入框相关

更新时间:2024-08-30

内容发生变化

参数说明：

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

回调原型：

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

高度发生变化

参数说明：

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

回调原型：

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

> ⚠ 警告
>
> 如重写此方法，请先调用父类方法。

消息操作相关

准备发送消息

此回调在消息准备向外发送时会回调，可以在此回调中对消息内容进行过滤和修改等操作。此回调的返回值不为 nil 时，SDK 会对外发送返回的消息内容。

参数说明：

参数	类型	说明
messageContent	RCMessageContent	消息内容

返回参数说明

参数	类型	说明
messageContent	RCMessageContent	修改后的消息内容

回调原型：

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

发送消息完成

参数说明：

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

回调原型：

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

取消消息发送

参数说明：

参数	类型	说明
messageContent	RCMessageContent	消息内容

回调原型：

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

即将插入消息

此回调在消息准备向外发送时会回调，可以在此回调中对消息内容进行过滤和修改等操作。此回调的返回值不为 nil 时，SDK 会对外发送返回的消息内容。

参数说明：

参数	类型	说明
messageContent	RCMessageContent	消息内容

返回参数说明

参数	类型	说明
messageContent	RCMessageContent	修改后的消息内容

回调原型：

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

Cell 显示

即将显示消息 Cell

参数说明：

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

回调原型：

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

未注册消息相关

显示内容

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

返回参数说明

类型	说明
RCMessageBaseCell	未注册消息需要显示的Cell

回调原型：

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

显示高度

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

返回参数说明

类型	说明
CGSize	未注册消息Cell需要显示的高度

回调原型：

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

Cell 点击事件

点击头像

参数说明：

参数	类型	说明
userId	NSString	点击头像对应的用户ID

回调原型：

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

长按头像

参数说明：

参数	类型	说明
userId	NSString	点击头像对应的用户ID

回调原型：

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

点击消息内容

参数说明：

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型

回调原型：

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

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

长按消息内容

参数说明：

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

回调原型：

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

> ⚠ 警告
>
> 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 URL

参数说明：

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

回调原型：

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

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

点击电话号码

参数说明：

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

回调原型：

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

获取长按时的菜单

参数说明：

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型

返回参数说明

类型	说明
NSArray <UIMenuItem *>	未注册消息Cell需要显示的高度

回调原型：

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

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

语音相关

开始录制

回调原型：

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

结束录制

回调原型：

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

取消录制

回调原型：

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

历史消息获取

功能描述

更新时间:2024-08-30

SDK 页面内已经内置了获取历史消息的功能，无需用户再次开发。

消息发送 属性描述

更新时间:2024-08-30

消息属性	描述
消息类名	各端消息名
存储属性	存储 / 不存储
离线属性	缓存 / 不缓存
推送属性	是/否
ObjectName	传输层名称，与消息类名一一对应
计数属性	计数 / 不计数
消息尺寸	128 KB
推送内容	详见各消息类送方式

存储属性

存储属性	存储分类	支持平台	详细描述
存储	客户端	Android、iOS	发送、接收该消息后，本地数据库存储 Web 端 和 小程序端因本地存储不可靠，不支持客户端消息存储，但可通过历史消息云存储服务获取历史记录
存储	云端	Android、iOS、Web	默认不在云端进行存储，需开通单群聊消息云存储服务，开通后，可在融云服务端存储 6 个月的历史消息，供客户端按需拉取
不存储	客户端	Android、iOS	发送、接收该消息后，本地数据库不存储
不存储	云端	Android、iOS、Web	无论是否开通历史消息云存储服务，该消息均不存储

计数属性

接收收到消息时，会话是否累计未读数。

计数属性	支持平台	详细描述
计数	iOS、Android、Web	会话未读消息数 + 1，该属性只影响会话列表未读数计数，App 应用角标可根据每个会话列表未读数累加获得
不计数	iOS、Android、Web	会话未读消息数不变

离线属性

接收人当前不在线时，是否进行离线缓存。

离线属性	详细描述
存储	消息进行离线缓存，默认 7 天。接收人在 7 天内上线，均可接收到该消息。超过 7 天后，消息被离线缓存淘汰。如有需要，可通过云端存储拉取到该消息
不存储	消息不进行离线缓存，所以只有接收人在线时，才可收到该消息。该消息不进行历史消息云存储、不进入云端存储（Log 日志）、不进行消息同步（消息路由）

推送属性

接收人是否接收推送，当离线属性为 存储 时，该属性生效。离线属性为 不存储 时属性无效。

由于 Web、小程序、PC 端没有推送平台，无法收到推送提醒。

推送属性	平台	推送方式	详细描述
推送	iOS、Android	APNs、华为、小米、魅族、OPPO、vivo、FCM、融云	当有离线缓存消息时，进行远程推送提醒，内容为该推送提醒显示的内容
不推送	iOS、Android	--	当有离线缓存消息时，不进行远程推送提醒

发送普通消息

发送入口

通过消息内容发送

接口原型：

```
- (RCMessage *)sendMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode nErrorCode, long messageId))errorBlock;
```

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

通过消息对象发送

> ⚠ 警告
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock;
```

RCMessage 添加消息配置的对象

```
/*!
消息配置，可用来设置静默消息（不进行通知提醒）
*/
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
消息推送配置，详细查看 配置消息推送
*/
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息对象
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

error 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIM sharedRCIM] sendMessage:message pushContent:pushContent pushData:nil successBlock:nil errorBlock:nil];
```

> ⚠ 警告
>
> 注意 messageid 无特殊情况应传 0

文本消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCTextMessage	RCMessageContent	RC:TxtMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
content	NSString	文本消息的内容

代码示例

```
RCTextMessage *textMessage = [RCTextMessage messageWithContent:@"消息内容"];

[[RCIM sharedRCIM] sendMessage:ConversationType_GROUP targetId:@"群组 ID" content:textMessage pushContent:nil
pushData:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

位置消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCLocationMessage	RCMessageContent	RC:LBSMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
image	UIImage	地理位置的缩略图
location	CLLocationCoordinate2D	地理位置的二维坐标
locationName	NSString	地理位置的名称

代码示例

```
RCLocationMessage *locationMessage = [RCLocationMessage messageWithLocationImage:image location:location
locationName:locationName];

[[RCIM sharedRCIM] sendMessage:ConversationType_GROUP targetId:@"群组 ID" content:locationMessage
pushContent:nil pushData:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

发送媒体消息

发送入口

通过消息内容发送

接口原型：

```
- (RCMessage *)sendMediaMessage:(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
cancel:(void (^)(long messageId))cancelBlock;
```

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调

参数	类型	必填	说明
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
messageId	long	消息的ID

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

`cancel` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

通过消息对象发送

> **警告**
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMediaMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
progress:(void (^)(int progress, RCMessage *progressMessage))progressBlock
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock
cancel:(void (^)(RCMessage *cancelMessage))cancelBlock;
```

RCMessage 添加消息配置的对象

```
/*!
 消息配置，可用来设置静默消息（不进行通知提醒）
 */
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
 消息推送配置，详细查看 配置消息推送
 */
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息实体
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
progressMessage	RCMessage	消息实体

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

`cancel` 说明：

回调参数	回调类型	说明
cancelMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIM sharedRCIM] sendMediaMessage:message pushContent:pushContent pushData:nil progress:nil successBlock:nil
errorBlock:nil cancel:nil];
```

> ⚠ 警告
>
> 注意 messageid 无特殊情况应传 **0**

图片消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCImageMessage	RCMediaMessageContent	RC:ImgMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
imageData	NSData	图片的原始数据

> ⚠ 警告
>
> 如果想发送原图，请设置属性 full 为 YES。

代码示例

```
RCImageMessage *message = [RCImageMessage messageWithImageData:data];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:message pushContent:nil
pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

语音消息

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCHQVoiceMessage	RCMediaMessageContent	RC:HQVCMsg	存储	计数	存储	推送	消息内容

构建参数说明：

参数	类型	说明
localPath	NSString	语音的本地路径
duration	long	语音消息的时长

示例代码

```
RCHQVoiceMessage *hqVoiceMsg = [RCHQVoiceMessage messageWithPath:path duration:duration];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:hqVoiceMsg
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

文件消息

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCFileMessage	RCMediaMessageContent	RC:FileMsg	存储	计数	存储	推送	消息内容

构建参数说明：

参数	类型	说明
localPath	NSString	文件的本地路径

代码示例

```
RCFileMessage *fileMessage = [RCFileMessage messageWithFile:path]

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:fileMessage
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

小视频消息

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 。

集成方式

请您参考 **全局配置 > 引入 SDK**，在 `Podfile` 文件中新增下面内容

```
pod 'RongCloudIM/Sight', '~> 2.10.4'
```

> 必须保持 RongCloudIM 所有导入的模块版本号一致。

发送本地视频

开启小视频服务后，SDK 支持从本地相册中选择视频文件发送，需要开发者设置选择媒体资源是否包含视频文件，默认为不包含，设为后支持发送本地视频文件，时长上限为 **2** 分钟，设置方法如下：

属性所属类为 `RCIM`，默认为 `NO`。

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

发送小视频消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCSightMessage	RCMediaMessageContent	RC:SightMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
path	NSData	视频文件本地路径
image	UIImage	视频首帧缩略图

参数	类型	说明
duration	NSUInteger	视频时长， 单位秒

代码示例

```
RCSightMessage *sightMessage = [RCSightMessage messageWithLocalPath:url thumbnail:image duration:duration];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:sightMessage
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

GIF 消息

> ⚠ 警告
>
> 从 SDK 2.9.21 版本开始支持发送 GIF 图片消息，默认可选择发送的 GIF 图片大小为 2M，接收于小于 1M 的消息时自动下载显示，大于 1M 时需要手动点击下载。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCGIFMessage	RCMediaMessageContent	RC:GIFMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
gifImageData	NSData	GIF 图的数据
width	long	GIF 图的宽
height	long	GIF 图的高

代码示例

```
RCGIFMessage *gifMsg = [RCGIFMessage messageWithGIFImageData:gifImageData
width:gifImage.size.width
height:gifImage.size.height];

[[RCIM sharedRCIM] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:gifMsg pushContent:nil
pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

发送自定义消息

除了使用 SDK 内置消息外，开发者可根据自己的业务需求来自定义消息。

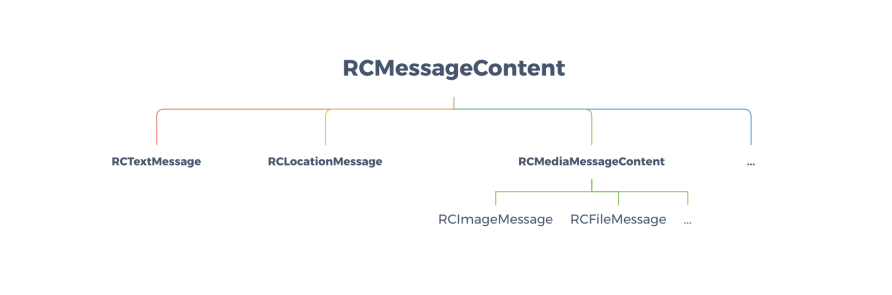
注册消息

注册消息共分为四步。

1. 定义消息类型
2. 注册消息类型
3. 定义消息 Cell
4. 注册消息 Cell

定义消息类型

开发者可根据业务需求来选择继承消息基类 `RCMessageContent` 或者 `RCMediaMessageContent`，两者之前的区别可参考下图:



定义消息类型需要该类遵守下面三个协议

一. 编解码协议

协议原型：

```
@protocol RCMessageCoding <NSObject>
```

`RCMessageCoding` 主要有三个功能：

1. 提供消息唯一标识符
2. 消息发送时将消息中的所有信息编码为 JSON 数据传输

3. 消息接收时将 JSON 数据解码还原为消息对象。

该协议为必须遵守的协议，遵守该协议需实现下面三个方法。

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：

```
+ (NSString *)getObjectName;
```

> ⚠ 警告
>
> 1. 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
> 2. 请勿使用 `RC:` 开头的类型名，以免和SDK默认的消息名称冲突
> 3. 所有自定义消息必须实现此协议，否则将无法正常传输和使用。

二. 存储协议

协议原型：

```
@protocol RCMessagePersistentCompatible <NSObject>
```

`RCMessageCoding` 主要有两个功能:

1. 指明此消息类型在本地和服务端是否存储
2. 指明此消息类是否计入未读消息数

该协议为必须遵守的协议，遵守该协议需实现下面方法:

消息的存储计数策略：

```
+ (RCMessagePersistent)persistentFlag;
```

RCMessagePersistent 说明:

类型	说明
MessagePersistent_NONE	表示消息不存储、不计数。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_ISCOUNTED	表示客户端收到消息后，要进行未读消息计数（未读消息数增加 1）。
MessagePersistent_ISPERSISTED	表示消息会存储到数据库。
MessagePersistent_STATUS	表示状态消息，该类型消息不存储不计数。状态类消息表示的是即时的状态，如果接收方不在线，则无法再收到该状态消息。

自定义消息持久化标识案例分析：

1. 需要存储和未读计数的消息（类似文本消息）：`MessagePersistent_ISCOUNTED | MessagePersistent_ISPERSISTED`
2. 需要确保收到，但不需要展示的消息，多为命令消息（类似撤回命令消息）：`MessagePersistent_NONE`
3. 对方在线接收，不在线就不用接收的消息（类似输入状态消息）：`MessagePersistent_STATUS`

三. 内容摘要协议

协议原型：

```
@protocol RCMessageContentView
```

`RCMessageContentView` 主要有一个功能:

1. 设置此消息在会话列表和本地通知中显示消息的摘要

该协议为非必须遵守的协议，遵守该协议需实现下面方法。

设置消息摘要：

```
- (NSString *)conversationDigest;
```

注册消息类型

开发者在进行完自定义消息类后，需要在 SDK init 之后，注册此自定义消息类。

```
- (void)registerMessageType:(Class)messageClass;
```

⚠ 警告

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。

显示消息

定义消息 Cell

开发者可根据自己的业务需求是否需要显示用户信息来选择继承消息 Cell。

1. 不包含用户头像和昵称

类名：

```
RCMessageBaseCell
```

结构图：

控件需要添加在 `baseContentView` 上，建议在 `baseContentView` 上方预留 **10**

2. 含用户头像和昵称

类名：

```
RCMessageCell
```

结构图：

控件需要添加在 `messageContentView` 上，根据需求在画 Cell 视图布局的时候调整 `messageContentView` 的 frame ，如果是接收方，只需要修改 `messageContentView` 的 `width` 和 `height`，如果是发送方，需要修改 `messageContentView` 的 `x` 、 `width` 和 `height`。

开发者需要在 Cell 的初始化中布局，并重写下面方法来返回 Cell 的 `Size`。当显示自定义消息时，必须实现该方法来返回 Cell 的 `Size`。

参数说明：

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

参数	类型	说明
extraHeight	CGFloat	Cell 内容区域之外的高度

方法原型：

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

注：`extraHeight` 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`

注册消息 Cell

SDK 需要开发者在会话页面注册您定义的消息 Cell 并绑定您的消息类型

```
- (void)registerClass:(Class)cellClass forMessageClass:(Class)messageClass;
```

发送消息

自定义消息继承 `RCMessageContent`，请使用 发送普通消息 的接口进行发送。

自定义消息继承 `RCMediaMessageContent`，请使用 发送媒体消息 的接口进行发送。

消息接收

功能描述

更新时间:2024-08-30

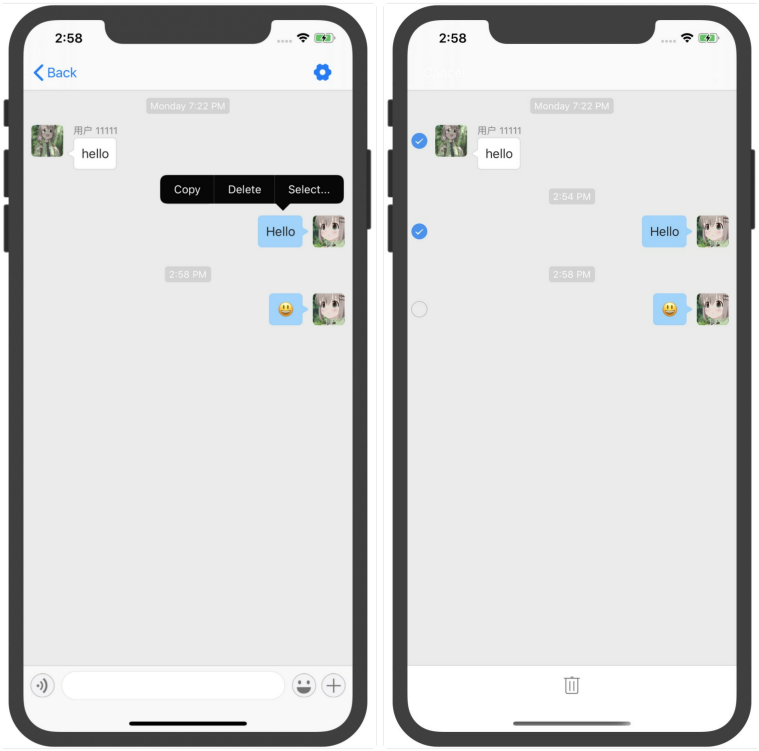
开发者可通过此接口拦截到 SDK 接收到的消息，并进行响应的业务操作。

实现方法

SDK 通过消息监听接收消息，详细可参考 设置消息监听 文档

消息多选 效果展示

更新时间:2024-08-30



自定义多选

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，可通过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,才会有有效值

```
@property(nonatomic, strong, readonly) NSArray<RCMessageModel *> *selectedMessages;
```

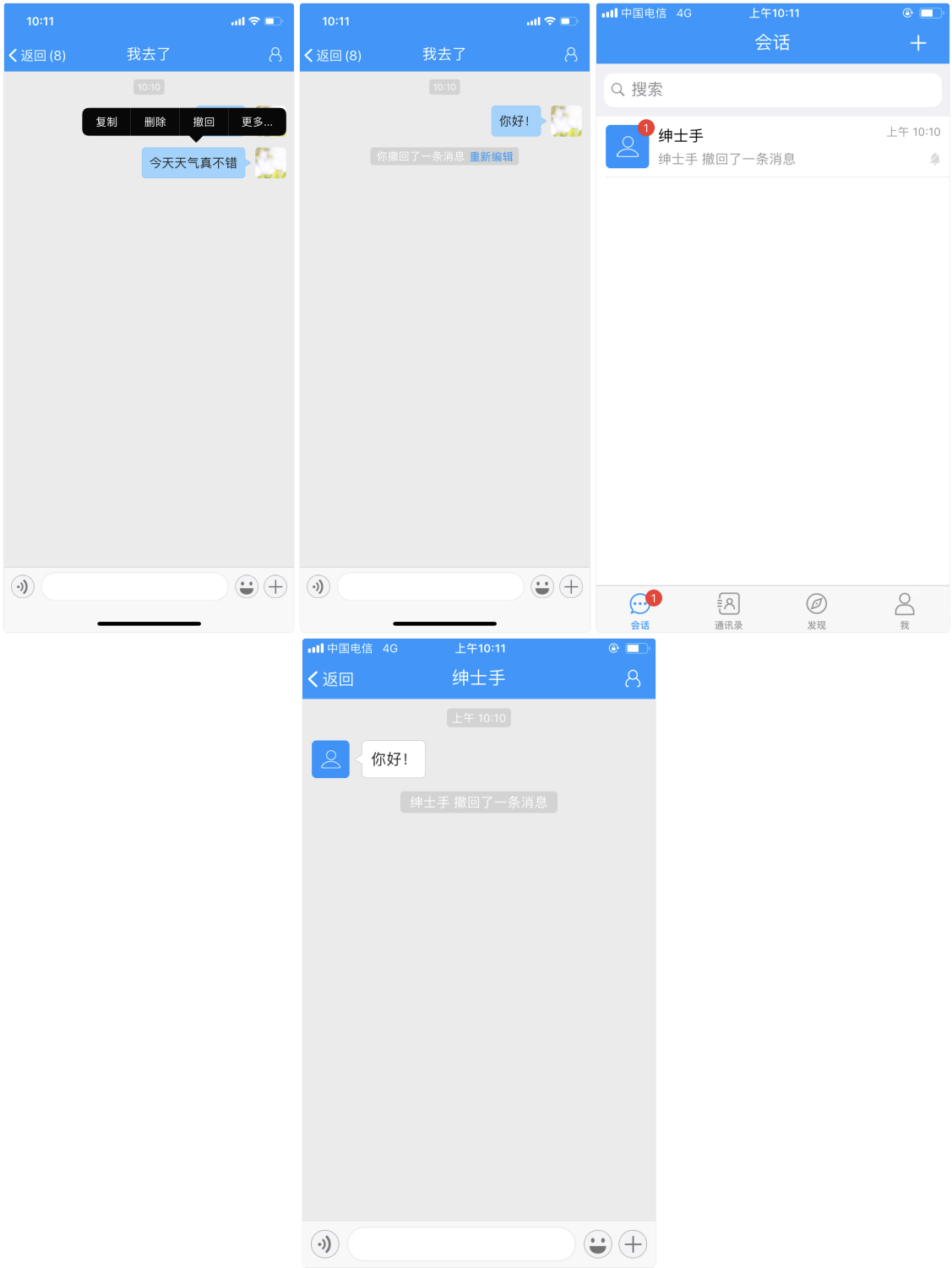
3. 底部视图

进入多选状态后，页面底部出现的工具视图.开发者可以添加对多选消息的操作事件，如转发、收藏等，可以在 viewDidLoad 中 通过给 messageSelectionToolbar 添加 UIBarButtonItem 的方式添加事件。

```
@property(nonatomic, strong) UIToolbar *messageSelectionToolbar;
```

消息撤回 效果展示

更新时间:2024-08-30



配置说明

1. **是否开启撤回功能**

默认值是 YES

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

2. **消息可撤回的最大时间**

单位是秒，默认值是 120 秒。

```
@property (nonatomic, assign) NSUInteger maxRecallDuration;
```

3. **消息撤回后可重新编辑的时间**

单位是秒，默认值是 300 秒。

```
@property (nonatomic, assign) NSUInteger reeditDuration;
```

> ⚠ 警告
>
> 目前消息撤回后重新编辑仅为本地操作，卸载重装或者更换设备不会同步。

通知回调

1. **消息被撤回的通知**

消息被撤回后，SDK 会分发此通知，`Notification` 的 `object` 为 `NSNumber` 的 `messageId` 。

通知原型：

```
FOUNDATION_EXPORT NSString *const RCKitDispatchRecallMessageNotification;
```

2. **消息被撤回的回调**

被撤回的消息会变更为 `RCRecallNotificationMessage` ，App 需要在 UI 上刷新这条消息。

需要您遵守 `RCIMReceiveMessageDelegate` 协议

参数说明：

参数	类型	说明
messageId	long	被撤回的消息ID

协议方法：

```
- (void)onRCIMMessageRecalled:(long)messageId;
```

3. **点击撤回消息Cell中重新编辑的回调**

点击撤回消息 Cell 中重新编辑，会调用此回调，且不会再触发 `didTapMessageCell`。

参数说明：

参数	类型	说明
model	RCMessageModel	消息Cell的数据模型

回调原型：

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

自定义撤回

开发者也可通过下面接口根据业务需求进行撤回消息

参数说明：

参数	类型	必填	说明
messageId	long	是	被撤回消息的消息Id

方法原型：

```
- (void)recallMessage:(long)messageId;
```

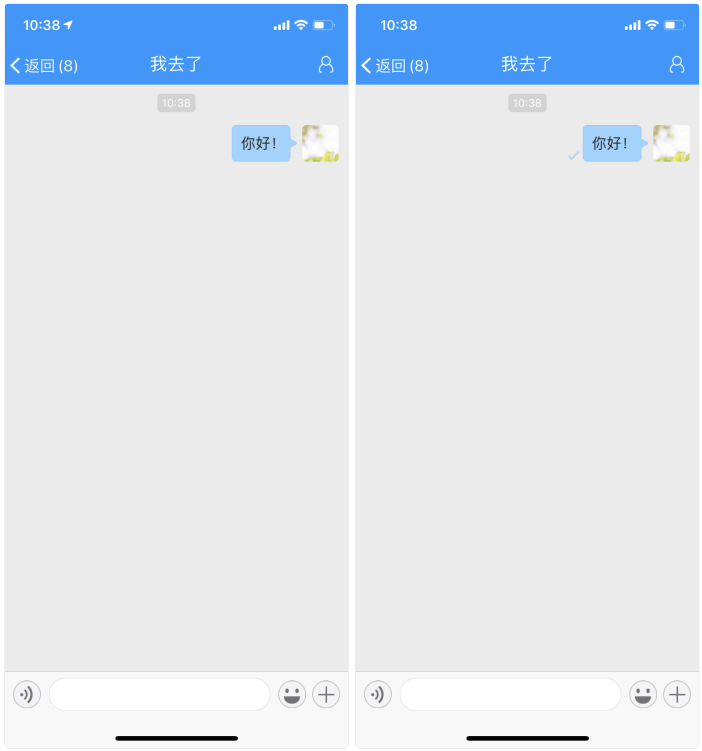
此方法为 `RCConversationViewController` 类的实例方法。

> ⚠ 警告
>
> 只有存储并发送成功的消息才可以撤回。

消息回执

更新时间:2024-08-30

效果展示



配置说明

属性所属类为 RCIM，默认为关闭状态。

```
@property(nonatomic, copy) NSArray* enabledReadReceiptConversationTypeList;
```

代码示例

```
[RCIM sharedRCIM].enabledReadReceiptConversationTypeList = @[ @(ConversationType_PRIVATE) ];
```

自定义显示

SDK 会话页面中文本消息已读的 UI 默认是一个“对勾”图标，开发者可参考下面内容修改为文本“已读”或“未读”。

思路：

1. 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”。
2. 需要监听消息发送状态更新的通知，更新 cell。

解决方案：

下面代码以单聊文本消息为例，如需多更多消息类型修改，可以自行添加判断修改 UI。

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) {
for (UIView *view in [((RCMessageCell *)cell).messageHasReadStatusView subviews]) {
[view removeFromSuperview];
}
UILabel *hasReadView = [[UILabel alloc] initWithFrame:CGRectMake(-18, 5, 30, 20)];
hasReadView.textAlignment = NSTextAlignmentRight;
hasReadView.font = [UIFont systemFontOfSize:12];
hasReadView.textColor = [UIColor blackColor];
((RCMessageCell *)cell).messageHasReadStatusView.hidden = NO;
if (model.sentStatus == SentStatus_READ) {
CGRect hasReadViewFrame = hasReadView.frame;
hasReadViewFrame.origin.y -= 19;
hasReadView.frame = hasReadViewFrame;
hasReadView.text = @"已读";
[((RCMessageCell *)cell).messageHasReadStatusView addSubview:hasReadView];
} else if (model.sentStatus == SentStatus_SENT) {
hasReadView.text = @"未读";
[((RCMessageCell *)cell).messageHasReadStatusView addSubview:hasReadView];
}
}
} else {
for (UIView *view in [((RCMessageCell *)cell).messageHasReadStatusView subviews]) {
[view 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];
```

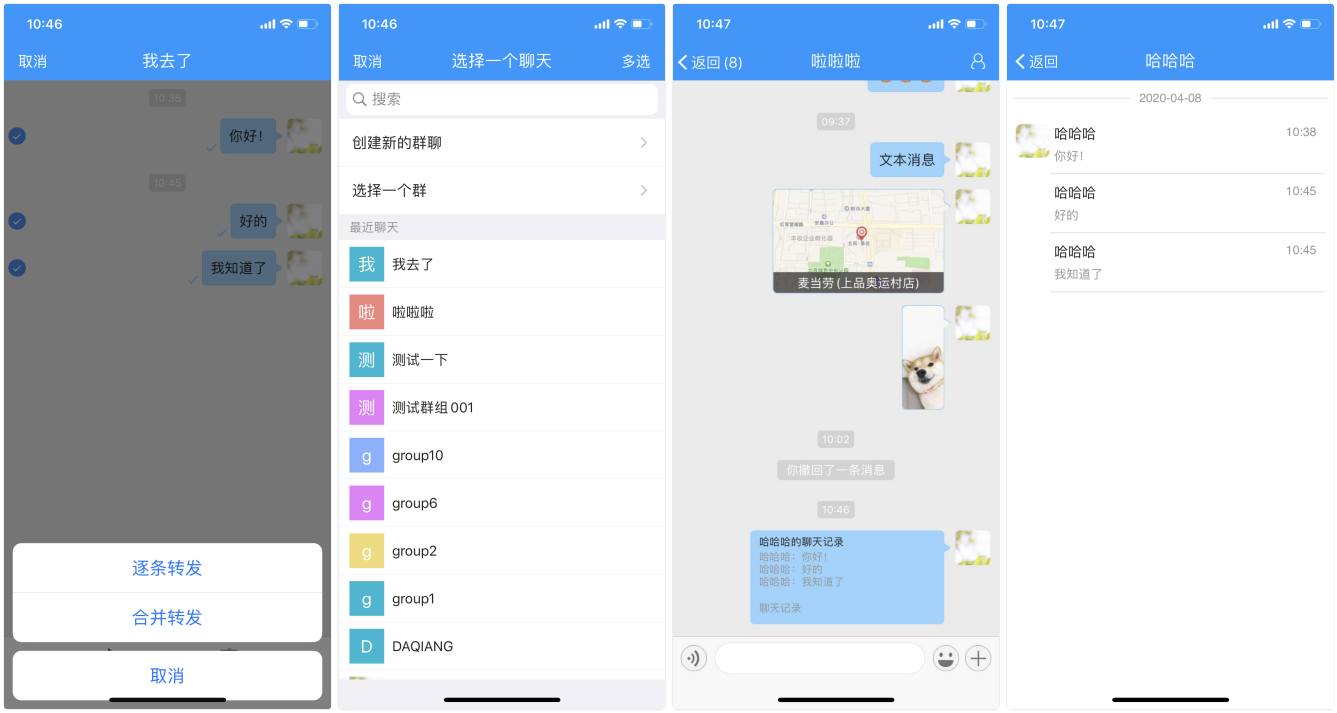
```
- (void)messageCellUpdateSendingStatusEvent:(NSNotification *)notification {
RCMessageCellNotificationModel *notifyModel = notification.object;
if (notifyModel && ![notifyModel.actionName isEqualToString:CONVERSATION_CELL_STATUS_SEND_PROGRESS]) {
[self reloadMessageCell:notifyModel.messageId];
}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

```
- (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];
[self.conversationMessageCollectionView reloadItemsAtIndexPaths:@[ 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) {
if ([__cell isKindOfClass:[RCMessageCell class]]) {
dispatch_async(dispatch_get_main_queue(), ^{
((RCMessageCell *)__cell).receiptCountLabel.hidden = YES;
});
}
}
}
```

消息转发 效果展示

更新时间:2024-08-30



配置说明

属性所属类为 RCIM，默认为 NO。

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

> ⚠ 警告
>
> 目前不支持红包、阅后即焚及自定义消息的合并转发功能。

方法说明

开启合并转发功能之后，RongIMKit 默认的选择会话的界面取的数据是数据库中会话表的全部会话。如果不满足需求，可以通过重写 RCConversationViewController.h 中的如下方法，来自行实现选择会话界面，并把选中后的会话列表回传给 RongIMKit。

参数说明：

参数	类型	说明
index	NSInteger	0 是逐条转发消息, 1 是合并转发消息。
completedBlock	Block	返回需要转发到的会话的列表。

回调参数说明：

参数	类型	说明
completedBlock	NSArray<RCConversation *>	返回需要转发到的会话的列表。

方法原型：

```
- (void)forwardMessage:(NSInteger)index completed:(void (^)(NSArray<RCConversation *>
*conversationList))completedBlock;
```

代码示例：

```
- (void)forwardMessage:(NSInteger)index completed:(void (^)(NSArray<RCConversation *> *))completedBlock {

// conversationList 为您返回的 list 数组
completedBlock(conversationList);
}
```

消息引用 配置说明

更新时间:2024-08-30

属性所属类为 RCIM，默认为开启状态。

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

代码示例

```
[RCIM sharedRCIM].enableMessageReference = YES;
```

消息删除 功能描述

更新时间:2024-08-30

SDK 内部已经集成了删除消息。长按消息后会弹出功能选择的菜单，点击删除后，SDK 会将此消息在本地数据库删除。

效果展示

自定义删除

开发者也可自定义删除功能。删除消息的接口使用可参考 消息删除 。

消息搜索

实现方法

更新时间:2024-08-30

SDK 未提供内置的 UI 组件，开发者需根据业务需求参考 SealTalk 开源项目 中的示例代码实现 UI 功能，之后调用 消息搜索接口 接口即可。

消息插入 插入消息

更新时间:2024-08-30

参数说明

参数	类型	必填	说明
message	RCMessage	是	消息实体

代码示例

此方法为 RCConversationViewController 的实例方法。

```
[self appendAndDisplayMessage:message];
```

常见问题

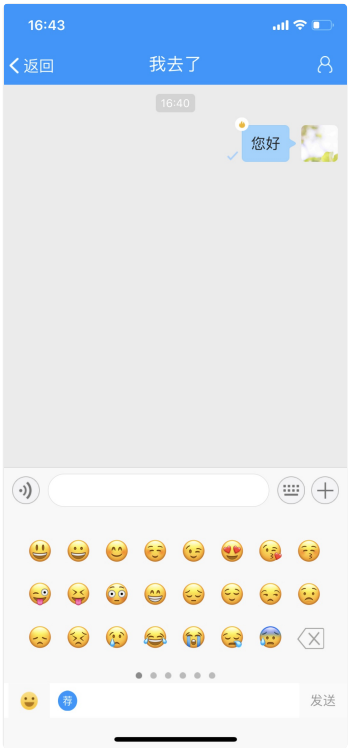
Q1: 插入的消息退出页面再进来就不在了。
A1: 这条消息只会在 UI 上插入，并不会存入数据库，当退出会话页面再次进入的时候，这条消息将不再显示。用户需要在数据库中同时进行存储，可先调用 消息插入 的接口在数据库插入消息，然后将返回的 RCMessage 对象再次调用下面方法插入到 UI。

表情区域 Emoji 表情

更新时间:2024-08-30

Emoji 是 SDK 默认访问资源文件中的 `Emoji.plist` 文件生成的且不支持移除。

效果展示



自定义表情

自定义表情需遵守 `RCEmoticonTabSource` 协议

1. 需要用户返回对应的表情View

参数说明：

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

代理方法：

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

2. 在会话页面添加自定义表情

开发者需要通过 `emojiBoardView` 的 `addEmojiTab` 方法添加表情包。

```
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];
```

相关示例代码可参考 SealTalk 开源项目 中的 RCDCustomerEmoticonTab。

加号区域

更新时间:2024-08-30

> ⚠ 警告
>
> 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。

内置扩展项

扩展项介绍

常用扩展项介绍：

定义名	Tag 值	说明
PLUGIN_BOARD_ITEM_ALBUM_TAG	1001	相册
PLUGIN_BOARD_ITEM_CAMERA_TAG	1002	拍照
PLUGIN_BOARD_ITEM_LOCATION_TAG	1003	位置
PLUGIN_BOARD_ITEM_FILE_TAG	1006	文件

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

常用扩展项

1. **添加文件功能**

加号区域增加发送文件功能，SDK 中已经默认实现了该功能，但是为了 SDK 向后兼容性，目前 SDK 默认不开启该入口，可以参考以下代码在加号区域中增加发送文件功能。

```
UIImage *imageFile = [RCKitUtility imageNamed:@"actionbar_file_icon" ofBundle:@"RongCloud.bundle"];
RCPluginBoardView *pluginBoardView = self.chatSessionInputBarControl.pluginBoardView;
[pluginBoardView insertItemWithImage:imageFile
title:NSLocalizedStringFromTable(@"File", @"RongCloudKit", nil)
atIndex:3
tag:PLUGIN_BOARD_ITEM_FILE_TAG];
```

2. **添加音视频功能**

音视频功能需要在 控制台 开通音视频服务，SDK 会自动增加音视频通话扩展项。

添加扩展项

参数说明

参数	类型	必填	说明
image	UIImage	是	扩展项的展示图片
title	NSString	是	扩展项的展示标题
index	NSInteger	是	需要添加到的索引值
tag	NSInteger	是	扩展项的唯一标示符

代码示例

```
[self.chatSessionInputBarControl.pluginBoardView insertItemWithImage:[UIImage imageNamed:@"poke_plugin_item"]
title:@"示例1"
atIndex:4
tag:2001];
```

更新指定扩展项

参数说明

参数	类型	必填	说明
image	UIImage	是	扩展项的展示图片
title	NSString	是	扩展项的展示标题
tag	NSInteger	是	扩展项的唯一标示符

代码示例

```
[self.chatSessionInputBarControl.pluginBoardView updateItemWithTag:101 image:newImage title:newTitle];
```

删除指定扩展项

参数说明

参数	类型	必填	说明
tag	NSInteger	是	扩展项的唯一标示符

代码示例

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

点击扩展项

参数说明

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)的唯一标示

回调方法

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

@ 消息 功能描述

更新时间:2024-08-30

群组 @ 功能，可以 @ 指定用户 或 @ 所有人。

效果展示

配置说明

开启功能

属性所属类为 RCIM，默认为 YES。

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

设置成员信息

此功能需要用户实现群成员数据源。可参考 设置群成员信息

自定义

用户可通过设置 RCMentionedInfo 自定义发送 @ 消息，然后将构建的对象赋值给 RCMessageContent 的 `mentionedInfo` 即可。

@ 单人

代码示例：

```
RCMentionedInfo *mentionedInfo = [[RCMentionedInfo alloc] initWithMentionedType:RC_Mentioned_Users
userIdList:mentionedUserIdList
mentionedContent:nil];
```

@ 所有人

具体内容可参考 SealTalk 开源项目 中的 RCDChatViewController。

代码示例：

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

自定义选人界面

开发者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UserInfo即可。

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

您可参考 SealTalk 开源项目 中 RCDChatViewController 的重写示例进行参考。

用户名片

效果展示

更新时间:2024-08-30

集成方式

参考 plugin-contactcard-ios

自定义 UI

自定义头像大小

更新时间:2024-08-30

开发者可根据业务场景自行定义会话页面头像的显示形状和大小

头像显示形状

属性所属类为 RCIM，默认值为矩形

```
@property (nonatomic) RCUserAvatarStyle globalMessageAvatarStyle;
```

RCUserAvatarStyle 说明：

类型名	说明
RC_USER_AVATAR_RECTANGLE	矩形
RC_USER_AVATAR_CYCLE	圆形

头像显示大小

属性所属类为 RCIM，默认值为 40*40

```
@property (nonatomic) CGSize globalMessagePortraitSize;
```

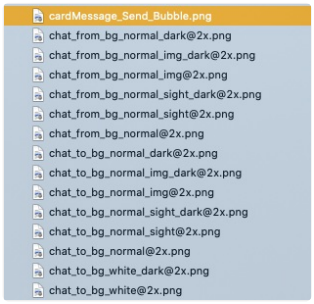
自定义聊天气泡

开发者可根据业务场景自行定义聊天气泡的样式，可以在 SDK 的资源包 (IMKit/Resources/Rongcloud.bundle) 中找到对应气泡图片，然后替换成您自己的图片即可。另外需要注意拉伸位置。

SDK 内拉升比例为：

```
UIEdgeInsetsMake(image.size.height * 0.8, image.size.width * 0.2, image.size.height * 0.2, image.size.width *
0.8)
```

bundle 内的图片资源：



自定义按导航按钮

相册列表页

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。

color.plist 文件路径:`IMKit/Resources/RCColor.plist`

相册预览页

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 `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。

位置选取页

左上、右上角按钮:

可设置 `RCIM` 中的 `globalNavigationBarTintColor` 来更改。

发送文件页

左上、右上角按钮:

可设置 `RCIM` 中的 `globalNavigationBarTintColor` 来更改。

右上角按钮未选择文件时颜色：

可 `color.plist` 中 `fileSelect` 字典里的相关字段来更改。

右上角按钮选中文件时的颜色：

可设置 RCIM 中的 globalNavigationBarTintColor 来更改。

个人名片页

左上角按钮：

可设置 RCIM 中的 globalNavigationBarTintColor 来更改。

未读数显示

右上角提示

当收到的消息超过一个屏幕时，进入会话之后，是否在右上角提示上方存在的未读消息数，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。

是否开启：

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

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：

开发者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;
}
}];
}
```

右下角提示

当前阅读区域的下方收到消息时，是否在会话页面的右下角提示下方存在未读消息，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，此会话收到消息时，会在右下角显示未读消息提示，而不会自动滚动到最下方，用户点击该提醒按钮，会滚动到最下方。

是否开启：

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

提示的 UILabel ：

当 unReadMessage > 10 右上角会显示未读消息数。

```
@property (nonatomic, strong) UILabel *unReadNewMessageLabel;
```

多端阅读消息同步

功能描述

更新时间:2024-08-30

开启之后，用户在其他端上阅读过的消息，当前客户端会清掉该消息的未读数。

配置说明

默认值为 YES。

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

代码示例

```
[RCIM sharedRCIM].enableSyncReadStatus = YES;
```

系统通知介绍 概述

更新时间:2024-08-30

一般系统消息特指会话类型（ConversationType）为“系统（SYSTEM）”的会话中的消息。

系统消息（System Message）是一种业务概念，是指利用系统帐号（非用户帐号，用户不可登录）向用户发送的消息，既可以是通过调用全量通知接口发送给所有人的消息，也可以是加好友提醒等单个用户通知消息。

> ⚠ 警告
>
> 系统通知消息，只能通过 Server API 进行发送，终端用户收到系统消息后，不支持消息回复功能。
> 具体消息发送方式详见：服务端集成-系统通知

主要功能

功能	描述
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。
消息提醒	离线状态，系统会话有新消息时，支持 Push 通知。
本地存储	存储在移动端本地，提供本地消息搜索功能。
历史消息	提供服务端消息存储存功能，需开通单群聊消息云存储，默认存储时长为 6 个月。
单个用户通知	向单个或多个用户发送系统消息。
标签用户通知	向标签下的所有用户发送系统消息，用户标签属性需要开发者通过 Server API 接口进行设置。
全量用户落地通知	向应用下的所有用户发送广播消息，用户在线情况下会直接收到消息内容，未在线时此条消息会转为 Push 进行推送，同时该条消息做为离线消息进行存储，用户下次登录后会通过离线消息获取。
全量用户不落地通知	向应用下的所有用户发送远程 Push，无论用户是否正在使用应用，该通知只会展示在通知栏，通知内容不会做为消息存储到本地。

消息类型

消息类型	描述
文字消息	用来发送文字类消息，其中可以包括表情、超链接（会自动识别），客户端收到消息后计入未读消息数、进行存储。
语音消息	发送高质量的短语音消息，录制的语音文件存储到融云服务端，语音文件格式为 AAC，时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。
图片消息	用来发送图片类消息，客户端收到消息后计入未读消息数、进行存储。图片缩略图格式为 JPG，大小建议不超过 100k。
GIF 图片消息	用来发送 GIF 动态图片消息，客户端收到消息后计入未读消息数、进行存储。
图文消息	用来发送图文消息，包含一个标题，一段文字内容和一张图片，客户端收到消息后计入未读消息数、进行存储。
文件消息	用来发送文件类消息，客户端收到消息后计入未读消息数、进行存储。
小视频消息	用来发送小视频消息，支持录制发送及选择本地视频文件发送两种方式，录制时长不超过 10 秒，本地选择视频文件方式时长不超过 2 分钟，小视频消息小视频文件格式为 .mp4，客户端收到消息后计入未读消息数、进行存储。

消息类型	描述
命令消息	用来发送通用的指令通知消息，消息内可以定义任意 JSON 内容，与通用命令通知消息的区别是不存储、不计数，此类型消息没有 Push 通知。
自定义消息	融云内置消息类型，无法满足客户业务需求时，可通过自定义消息类型进行实现，接收自定义消息的格式解析及展示处理需要开发者自行实现

消息结构示例

系统通知场景下融云内置消息类型说明：

文本消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:TxtMsg	存储	计数	存储	推送	消息内容

消息结构：

发送文本消息时 content 参数的 JSON 结构如下：

```
{
"content":"Hello world!",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
content	String	是	文字消息的文字内容，包括表情。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

图片消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:ImgMsg	存储	计数	存储	推送	[图片]

消息结构：

发送图片消息时 content 参数的 JSON 结构如下：

```
{
"content":"bhZPzJXimRwrtvc=",
"imageUri":"http://p1.cdn.com/fds78ruhi.jpg",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
content	String	是	图片缩略图，格式为 JPG，Base64 字符串长度建议为 5k，最大不超过 10k，注意在 Base64 进行 Encode 后需要将所有 `\r\n` 和 `\r` 和 `\n` 替换成空。
imageUri	String	是	图片上传到图片存储服务器后的地址。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题：

1、缩略图最大尺寸为：240 x 240 像素，以宽度和高度中较长的边不超过 240 像素等比压缩。

2、大图最大尺寸为：960 x 960 像素，以宽度和高度中较长的边不超过 960 像素等比压缩。

3、图片消息包括两个主要部分：缩略图和大图，如设置为原图发送则为缩略图和原图，缩略图直接 Base64 编码后放入 `content` 中，大图或原图首先上传到文件服务器（融云 SDK 中默认上传到七牛云存储），然后将云存储上的大图或原图地址放入消息体中。

4、发送图片消息时，需要自行上传图片文件到应用的文件服务器，生成地址后进行发送。

GIF 图片消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:GIFMsg	存储	计数	存储	推送	[图片]

消息结构：

发送图片消息时 content 参数的 JSON 结构如下：

```
{
"gifDataSize":34563,
"height":246,
"localPath":"/var/mobile/.../GIF_53",
"remoteUrl":"https://rongcloud-image.cn.ronghub.com/image_jpe64562665566.jpg",
"width":263,
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
gifDataSize	Int	是	GIF 图片文件大小，单位为字节 Byte。
localPath	String	是	下载 GIF 图片后存储在本地的图片地址。
remoteUrl	String	是	GIF 图片的服务器地址。
width	Int	是	GIF 图片宽度。
height	Int	是	GIF 图片高度。
user	String	是	消息中携带的用户信息，IMKit SDK 会话界面中优先显示消息中携带的用户信息，可去掉此属性。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

语音消息

> ⚠ 警告
>
> 从 SDK 2.9.19 版本开始支持 `RC:HQVCMsg` 语音消息功能，`RC:HQVCMsg` 语音消息与旧版本 SDK 不兼容，旧版本 SDK 无法收听新的语音消息。

新语音消息 `RC:HQVCMsg` 和旧版本语音消息 `RC:VcMsg` 不同的是将录制的音频数据存储到服务端，而消息体内只保存 URL。摆脱了消息体 128K 的大小限制，所以拥有更高音质。语音时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:HQVCMsg	存储	计数	存储	推送	[语音]

消息结构：

发送高质量语音消息时 content 参数的 JSON 结构如下：

```
{
"localPath":"/9j/4AAQSkZ/2wBaSiimB//9k=",
"remoteUrl":"http://p1.cdn.com/fds78ruhi.aac",
"duration":7,
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

参数说明:

参数	类型	必传	说明
localPath	String	否	采用 AAC 格式进行编码录制的媒体内容本地路径。
remoteUrl	String	是	媒体内容上传服务器后的网络地址。
duration	Int	是	语音消息的时长，最长为 60 秒（单位：秒）。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 参数说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题

1、发送高质量语音消息时，需要自行生成 AAC 格式文件并上传文件到应用的文件服务器，生成地址后进行发送。

文件消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:FileMsg	存储	计数	存储	推送	[文件] + 文件名，如：[文件] 123.txt

消息结构:

发送文件消息时 content 参数的 JSON 结构如下：

```
{
"name":"file.txt",
"size":190184,
"type":"txt",
"fileUrl":"http://www.demo.com/am.ind",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
name	String	是	文件名称。
size	String	是	文件大小，单位：Byte。
type	String	是	文件类型。
fileUrl	String	是	文件地址。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题

1、通过 Server API 发送文件消息时，需要自行上传文件到应用的文件服务器，生成文件地址后进行发送。

小视频消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:SightMsg	存储	计数	存储	推送	[小视频]

消息结构：

发送小视频消息时 content 参数的 JSON 结构如下：

```
{
"sightUrl":"http://rongcloud...com/video...",
"content":"/9j/4AAQSkZ/2wB...hDSaSiimB//9k=",
"duration":2,
"size":734320,
"name":"video_xx.mp4",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":"extra"
}
```

属性说明：

名称	类型	必传	说明
sightUrl	String	是	上传到文件服务器的小视频地址。
content	String	是	小视频首帧的缩略图进行 Base64 编码的结果值，格式为 JPG，注意在 Base64 进行 Encode 后需要将所有 \r\n 和 \r 和 \n 替换成空。
duration	Int	是	视频时长，单位：秒。 • 如使用 IMKit 小视频插件进行录制，默认最长可录制 10 秒。 • 如选择本地视频文件，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。
size	String	是	视频大小单位 Byte。
name	String	是	发送端视频的文件名，小视频文件格式为 `MP4`（H.264+AAC）。
user	String	否	消息中携带的用户信息，详见 `user` 属性说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

- `user` 属性说明：

名称	说明
id	发送用户 ID。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题

1. 通过 Server API 发送视频消息时，需要自行上传视频文件到应用的文件服务器，生成文件地址后进行发送。
2. IMKit SDK 中目前支持播放的视频文件格式为 mp4，IMLib SDK 中播放功能需要开发者自行实现。

命令消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:CmdMsg	不存储	不计数	存储	不推送	无

消息结构：

运营平台向终端发送指令信息时可使用此命令消息，消息中 content 参数的 JSON 结构如下：

```
{
"name":"AtPerson",
"data":"{\"sourceId\":\"9527\"}"
}
```

属性说明：

名称	类型	必传	说明
name	String	是	命令名称，可以自行定义。
data	String	是	命令的内容。

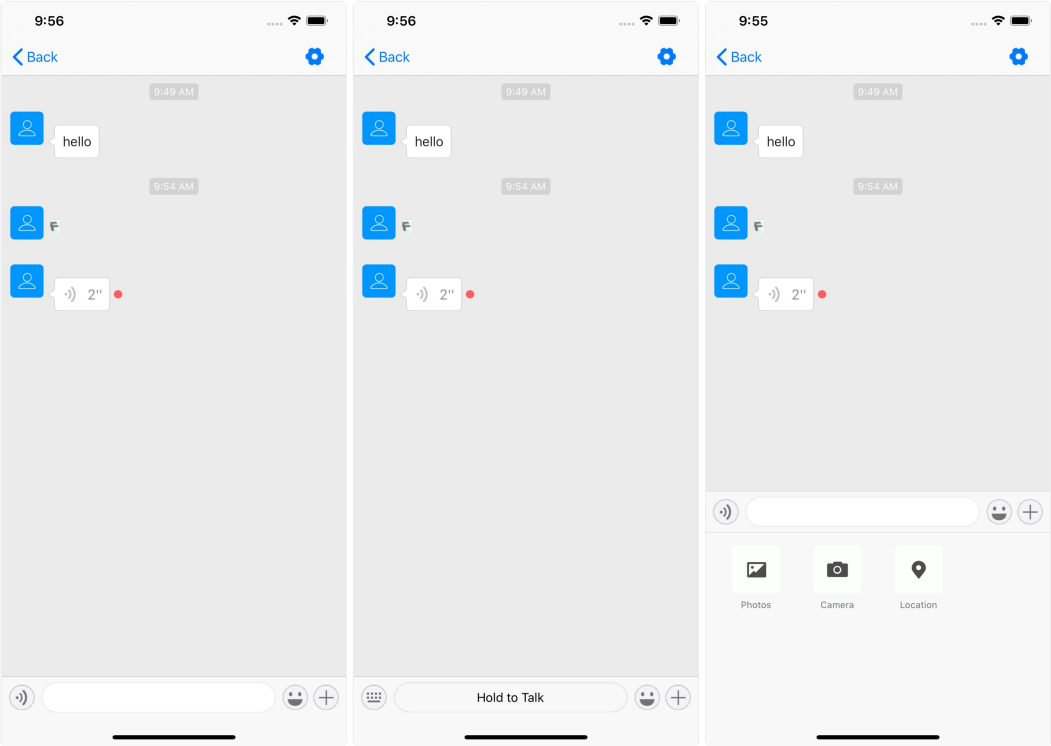
会话界面 功能描述

更新时间:2024-08-30

融云 IMKit 已经实现了一个默认的会话视图控制器，直接使用或继承此类，即可快速启动和使用会话界面。

效果展示

图中未展示发送者昵称和头像是因为未设置用户信息，具体如何设置请参考 设置用户信息



构建会话界面

```
/*!
会话页面类
*/
RCConversationViewController
```

构建说明

开发者可通过两种方式使用会话列表页面。

1. 直接使用此类

   建议仅在排查会话列表的相关错误时使用，如果在使用此页面期间出现了相关问题，可以直接 push 到此类排查是否能复现相关问题。

2. 继承使用此类

开发者可继承此类，并跳转到此页面来开发相关内容。

参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId

方法原型

```
- (id)initWithConversationType:(RCConversationType)conversationType targetId:(NSString *)targetId;
```

代码示例

```
RCDConversationViewController *conversationVC = [[RCDConversationViewController alloc]
initWithConversationType:ConversationType_PRIVATE targetId:@"targetId"];
[self.navigationController pushViewController:conversationVC animated:YES];
```

历史消息获取

功能描述

更新时间:2024-08-30

SDK 页面内已经内置了获取历史消息的功能，无需用户再次开发。

消息接收

功能描述

更新时间:2024-08-30

开发者可通过此接口拦截到 SDK 接收到的消息，并进行响应的业务操作。

实现方法

SDK 通过消息监听接收消息，详细可参考 设置消息监听 文档

升级说明

更新时间:2024-08-30

本文会详细描述 4.0.0 版本相对于 2.X 的修改，如果您只关心升级适配步骤，请参考文末的旧版本快速兼容方案。

一、接口变更说明

4.0.0 相较于旧版本，只有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

4.0.0 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

> 或者直接参考 二、示例代码。

1. error 回调中判断是否是 token 非法和连接超时超时。

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。

二、示例代码

注：示例代码按照 IMKit 的相关接口进行演示，如果您使用 IMLib，SDK 的业务处理逻辑是一致的，根据 IMLib 的相关接口进行接口替换即可。

```
//1.连接处理
[[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
[[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] setConnectionStatusDelegate: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
}
}
```

三、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 4.0.0 新版本建议参见上面的详细处理流程。

::: 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。:::

以下代码以 IMKit 为例，IMLib 修改对应的核心类即可(注意代码注释)。

```
[[RCIM sharedRCIM] 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
}
}];
```

四、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。

五、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。");
```

IMKit 库 RCIM 类以下废弃接口均被删除

1.收到已读回执通知

```
FOUNDATION_EXPORT NSString *const
RCKitDispatchReadReceiptNotification __deprecated_msg("已废弃，请使用RCLibDispatchReadReceiptNotification通知。");
```

2.图片消息发送

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。");
```

3.开启已读回执功能

```
@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。");
```

前提条件

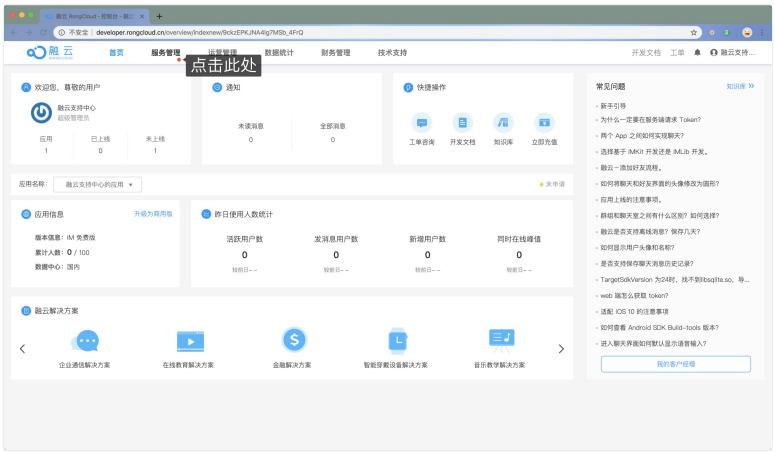
更新时间:2024-08-30

1. 访问 控制台 ，使用手机号注册账户。

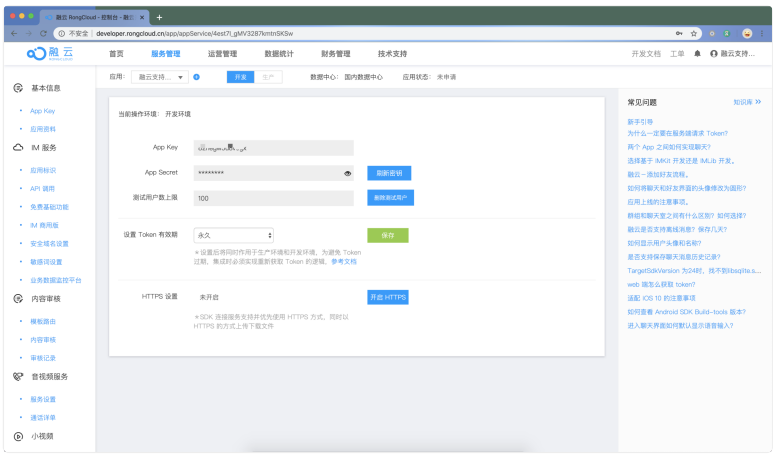
2. 填写企业名称和邮箱地址。

3. 进行邮箱验证。

4. 在控制台点击 服务管理。



5. 选择对应环境的 AppKey。



SDK 快速集成

更新时间:2024-08-30

> 提示
>
> 以下内容为快速集成摘要，具体细节可依次参考全局配置

导入 SDK

SDK 支持使用 CocoaPods 导入并管理 SDK。 需安装 CocoaPods 环境，请参照 CocoaPods 安装

1. `cd` 至项目根目录
2. 执行 `pod init`
3. 执行 `open -e Podfile`
4. 在 `Podfile` 文件中添加下面内容

```
pod 'RongCloudIM/IMLib', '~> 4.0.3.3'
```

如需指定具体版本，请注意 pod 使用规范 。

5. 执行 `pod install`
6. 双击打开 `.xcworkspace`

初始化

1. 使用 SDK 功能前，请 `import` 下面的头文件，`Swift` 项目需要在工程的 `Bridging-Header.h` 文件中添加 SDK 的引用。

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

2. 请使用开发功能之前从控制台注册得到的 `Appkey`，通过 `RCIMClient` 的单例，传入 `initWithAppKey:` 方法，初始化 SDK。
3. 开发者在使用融云 SDK 所有功能之前，开发者必须先调用此方法初始化 SDK。 在 App 的整个生命周期中，开发者只需要将 SDK 初始化一次。

```
[[RCIMClient sharedRCIMClient] initWithAppKey:@"Your Appkey"];
```

连接融云

1. `Token` 即用户令牌，相当于您 APP 上当前用户连接融云的身份凭证。在您连接融云服务器之前，您需要请求您的 App Server，您的 App Server 通过融云 Server API 获取 Token 并返回给您的客户端，客户端获取到这个 Token 即可连接融云服务器。

2. `dbOpened` 即消息数据库是否打开的回调，会返回具体业务码。
3. `success` 即连接成功回调，会返回 `Token` 对应的 `UserId` 。
4. `error` 即连接失败回调，请您检查客户端初始化使用的 `AppKey` 和您服务器获取 `token` 用的 `AppKey` 是否一致。当走到该回调时说明 SDK 无法继续重连，需要 APP 再次连接

```
[[RCIMClient sharedRCIMClient] connectWithToken:@"开发者的 server 通过请求 server api 获取到的 token 值"
dbOpened:^(RCDBErrorCode code) {}
success:^(NSString *userId) {}
error:^(RCConnectErrorCode status) {}];
```

获取会话列表

1. 获取到所有单聊会话，如果需要获取更多会话类型的会话，可以增加数组中的会话类型对象。
2. 可以将获取到的会话列表数用 `UITableView` 来展示。

```
NSArray *conversationList = [[RCIMClient sharedRCIMClient] getConversationList:@[@(ConversationType_PRIVATE)]];
```

发送消息

1. 构造文本消息对象。
2. 调用 `RCIMClient` 中 `sendMessage` 方法发送。
3. `success` 即发送成功回调，会返回消息的 `messageId` 。
4. `error` 即发送失败回调，会返回消息的 `messageId` 和 发送失败的错误码 `nErrorCode` 。

```
RCTextMessage *txtMessage = [RCTextMessage messageWithContent:@"测试文本消息"];
[[RCIMClient sharedRCIMClient]
sendMessage:ConversationType_PRIVATE
targetId:@"tester1"
content:txtMessage
pushContent:@"远程推送显示的内容"
pushData:@"远程推送的附加信息"
success:^(long messageId) {

}
error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

接收消息

1. 设置接收消息监听代理。

```
// 设置消息接收监听，建议设置给 AppDelegate 之类的单利对象，保证在 APP 整个生命周期内都有效
[[RCIMClient sharedRCIMClient] setReceiveMessageDelegate:self object:nil];
```

2. 实现接收消息回调方法。

```
- (void)onReceived:(RCMessage *)message
left:(int)nLeft
object:(id)object {
if ([message.content isMemberOfClass:[RCTextMessage class]]) {
RCTextMessage *testMessage = (RCTextMessage *)message.content;
NSLog(@"消息内容: %@", testMessage.content);
}

NSLog(@"还剩余的未接收的消息数: %d", nLeft);
}
```

获取历史消息

1. 从本地数据库中获取历史消息。
2. `oldestMessageId` 传 **-1** 即从最新的一条消息开始获取。
3. `count` 即获取消息的数量。
4. `RCIMClient` 类中还有其他获取历史消息的接口，可根据自己的需求调用。
5. 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。

```
[[RCIMClient sharedRCIMClient]
getHistoryMessages:ConversationType_PRIVATE
targetId:@"tester1"
oldestMessageId:-1
count:10];
```

断开连接

1. 不同接口决定断开融云连接后收不收推送，视情况调用。
2. 在下次连接融云成功后，会收取上次离线后的消息，离线消息最多可以保存 **7** 天。

```
//断开连接接收推送
[[RCIMClient sharedRCIMClient] disconnect];

//断开连接不收推送
[[RCIMClient sharedRCIMClient] logout];
```

引入 SDK

环境要求

更新时间:2024-08-30

名称	版本
Xcode	11 +
手机系统	8.0 +
CocoaPods	1.10.0 +

导入 SDK

SDK 支持使用 CocoaPods 导入并管理 SDK。 需安装 CocoaPods 环境，请参照 CocoaPods 安装

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

1. cd 至项目根目录
2. 执行 `pod init`
3. 执行 `open -e Podfile`
4. 添加导入配置 `pod 'RongCloudIM/IMLib', '~> 2.10.4'`

   `2.10.4` 为当前最新版本，如需指定具体版本，请注意 pod 使用规范。
5. 执行 `pod install`
6. 双击打开 `.xcworkspace`

升级

1. 执行 `pod repo update`
2. 执行 `pod search RongCloudIM` 查询最新版本

```
pod search rongcloudim (less)

-> RongCloudIM (2.10.4)
   RongCloud IM SDK.(如果看不到最新版本，请使用 pod repo update 命令更新一下本地pod仓库，从 2.9.0 版本开始这里不再包含 CallKit 和 CallLib 多
库的管理，音视频服务升级为融云自有音视频，严网用户请根据站内信提示更新)
   pod 'RongCloudIM', '~> 2.10.4'
   - Homepage: http://rongcloud.cn/
   - Source:   https://downloads.rongcloud.cn/Rong_Cloud_iOS_IM_SDK_v2_10_4_Stable.zip
   - Versions: 2.10.4, 2.10.3, 2.10.2, 2.10.1, 2.10.0, 2.9.25, 2.9.24, 2.9.23, 2.9.22, 2.9.21, 2.9.20, 2.9.19, 2.9.18, 2.9.17, 2.9.16,
   2.9.15, 2.9.14, 2.9.13, 2.9.12, 2.9.11, 2.9.10, 2.9.9, 2.9.8, 2.9.7, 2.9.6, 2.9.5, 2.9.4, 2.9.3, 2.9.2, 2.9.1, 2.9.0, 2.8.32, 2.8.31,
   2.8.30, 2.8.29, 2.8.28, 2.8.27, 2.8.26, 2.8.25, 2.8.24, 2.8.23, 2.8.22, 2.8.21, 2.8.20, 2.8.19, 2.8.18, 2.8.17, 2.8.16, 2.8.15, 2.8.14,
   2.8.13, 2.8.12, 2.8.11, 2.8.10, 2.8.9, 2.8.8, 2.8.7, 2.8.6, 2.8.5, 2.8.4, 2.8.3, 2.8.2, 2.8.1, 2.8.0 [master repo]
   - Subspecs:
     - RongCloudIM/IMLib (2.10.4)
     - RongCloudIM/IMKit (2.10.4)
     - RongCloudIM/RongSticker (2.10.4)
     - RongCloudIM/Sight (2.10.4)

-> RongCloudIMKit (2.7.3.1) [DEPRECATED]
   RongCloud IMKit.(废弃说明：从2.8.0版本开始，本仓库废弃，2.8.0以后版本使用RongCloudIM仓库，具体细节请参考融云官方文档)
   pod 'RongCloudIMKit', '~> 2.7.3.1'
   - Homepage: http://rongcloud.cn/
   - Source:   http://downloads.rongcloud.cn/Rong_Cloud_iOS_IMKit_SDK_v2_7_3_stable.zip
   - Versions: 2.7.3.1, 2.7.3, 2.7.2, 2.7.1, 2.7.0, 2.6.11, 2.6.10, 2.6.9, 2.6.8, 2.6.7, 2.6.6, 2.6.5, 2.6.4, 2.6.3, 2.6.2, 2.6.1, 2.6.0,
   2.5.2, 2.5.1, 2.5.0, 2.4.11, 2.4.10, 2.4.9, 2.4.8, 2.4.7, 2.4.6, 2.4.5, 2.4.4, 2.4.3, 2.4.2, 2.4.1, 2.4.0, 2.3.9, 2.3.8, 2.3.7, 2.3.6,
   2.3.5, 2.3.4, 2.3.3, 2.3.2, 2.3.1, 2.3.0, 2.2.11, 2.2.10, 2.2.9, 2.2.8, 2.2.7, 2.2.6, 2.2.5, 2.2.4, 2.2.3, 2.2.2, 2.2.1, 2.2.0, 2.1.5,
   2.1.4, 2.1.3, 2.1.2, 2.1.1, 2.1.0, 2.0.5, 2.0.3.1, 2.0.3, 1.5.3, 1.5.2, 1.5.1, 1.4.8, 1.4.7, 1.4.6, 1.4.5, 1.4.4.1, 1.4.4, 1.4.3, 1.4.2,
   1.4.1.1, 1.4.1, 1.3.4, 1.3.3, 1.3.2.1, 1.3.2, 1.2.9.1, 1.2.9, 1.2.8, 1.2.5, 1.2.4.2, 1.2.4.1, 1.2.4 [master repo]

-> RongCloudIMKitWithVoip (2.7.3.1) [DEPRECATED]
```

3. 替换 Podfile 文件内容

4. 执行 pod install

查看当前导入版本

1. 打开根目录
2. 到 `Pods/RongCloudIM/RongCloudIM/RongIMLib.framework` 目录下
3. 打开 `info.plist` 文件
4. 找到 `Bundle versions string, short` 字段，其 `value` 值为当前集成版本号

清除 Pod 缓存

1. 移除项目目录 `Pods` 文件下的 `RongCloudIM`。
2. 删除 `cocoaPods` 目录缓存，找到 `~/Library/Caches/CocoaPods/Pods/Release`，删除此目录。
3. 点击 `Xcode -> Preferences -> Locations -> Derived Data` 点击下方的右箭头跳转到相应目，删除此目录即可。
4. 项目根目录执行 `pod repo update`。
5. 项目根目录执行 `pod install`。

常见问题

> Q1: 集成 SDK 必须使用 Xcode 11 吗？
>
> A1: 融云 SDK 2.9.25 及之后的版本兼容了 iOS 13 ，需要兼容 iOS 13，则需使用新版本的 Xcode；不需要兼容 iOS 13 或者不想升级到 Xcode 11，请使用 2.9.24 版本的 SDK。 不兼容的原因

初始化 功能描述

更新时间:2024-08-30

融云 SDK 需要开发者在工程中调用下面方法来初始化 SDK，**请务必保证该接口在应用全生命周期内仅被调用一次。**

功能描述

融云 SDK 需要开发者在工程中调用下面方法来初始化 SDK。

> ⚠ 警告
>
> 1. 开发者在使用融云 SDK 所有功能之前，开发者必须先调用此方法初始化 SDK，可将此方法写到 `AppDelegate.m` 中。
> 2. Appkey 区分 开发环境 与 生产环境，开发者在使用时注意当前环境是否匹配。
> 3. **在应用的整个生命周期内，SDK 只能 init 一次，不可重复调用此接口。**

参数说明

参数	类型	必填	说明
appKey	NSString	是	开发者在 控制台 获取的 Appkey

代码示例

1. 导入头文件

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

2. 初始化 SDK

```
[[RCIMClient sharedRCIMClient] initWithAppKey:@"此处填写获取的 Appkey "];
```

设置监听　　设置状态监听

更新时间:2024-08-30

当 SDK 与融云服务器的连接状态发生变化时，开发者可通过下面方法进行处理。
实现此功能需要开发者遵守 `RCConnectionStatusChangeDelegate` 协议。

参数说明

参数	类型	必填	说明
delegate	id	是	IMLib 连接状态监听器

参数	类型	说明
status	RCConnectionStatus	SDK 与融云服务器的连接状态

代码示例

1. 设置代理对象:

```
[[RCIMClient sharedRCIMClient] setRCConnectionStatusChangeDelegate:self];
```

2. 实现协议方法:

```
- (void)onConnectionStatusChanged:(RCConnectionStatus)status {

}
```

设置消息监听

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。
实现此功能需要开发者遵守 `RCIMClientReceiveMessageDelegate` 协议。

参数说明

代理对象参数说明：

传入参数	类型	必填	说明
delegate	id	是	IMLib 消息接收监听器
userData	id	否	用户自定义的监听器 Key 值，可以为 nil

协议方法参数说明：

参数	类型	说明

参数	类型	说明
message	RCMessage	当前接收到的消息
nLeft	int	还剩余的未接收的消息数
object	id	消息监听设置的 key 值
offline	BOOL	是否是离线消息
hasPackage	BOOL	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

代码示例

1. 设置代理对象:

```
[[RCIMClient sharedRCIMClient] setReceiveMessageDelegate:self object:nil];
```

2. 实现协议方法:

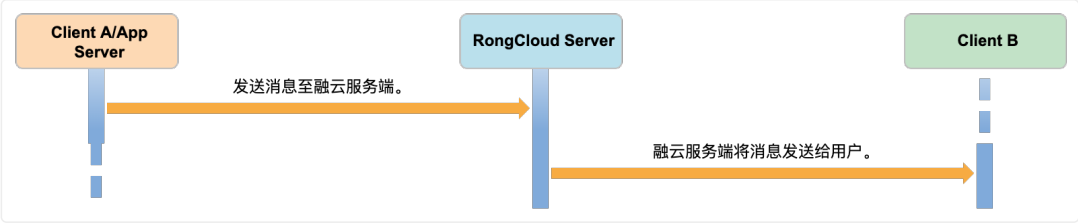
```
- (void)onReceived:(RCMessage *)message left:(int)nLeft object:(id)object offline:(BOOL)offline
hasPackage:(BOOL)hasPackage {

}
```

本地通知提醒 功能描述

更新时间:2024-08-30

当 App 进入后台，SDK 处于后台活跃状态，App 会使用融云的长连接通道接收消息，此时 SDK 收到消息会触发 消息监听，开发者可以在此处弹出本地通知。

流转图：



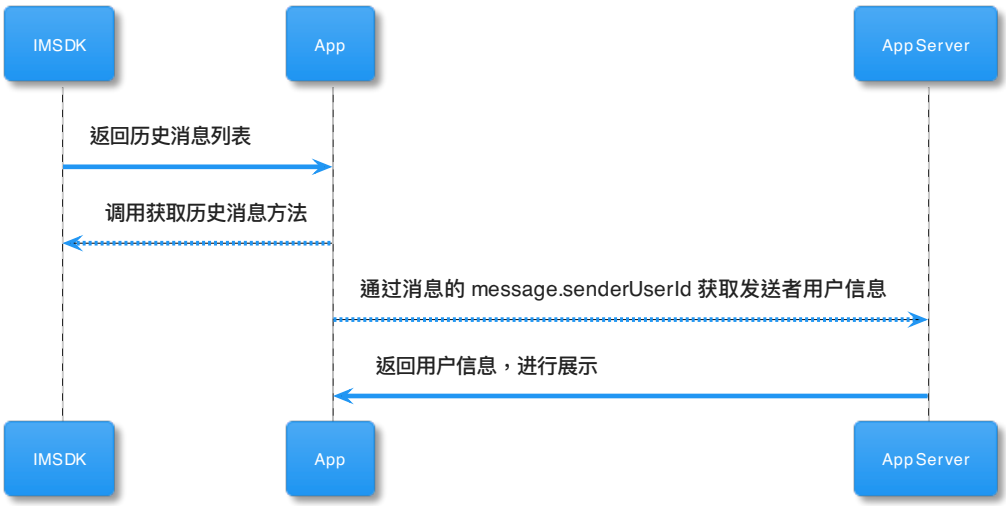
> ⚠ 警告
>
> iOS 端会在 2 分钟之后主动断开，之后的消息会通过 APNs 通道进行发送。

显示用户信息　　在历史消息中显示

更新时间:2024-08-30

从 App Server 获取用户信息

1. 您的 App Server 封装获取用户信息接口。
2. 通过 message.senderUserId 获取发送者 ID。
3. 将发送者 ID 传入 App Server 暴露的接口中，以获取对应的用户信息。
4. 将用户信息展示到页面中。



发消息时携带用户信息

> 携带的用户信息保存在消息中。如果用户修改了用户信息，已经发送的消息携带的用户信息不会同步更新。

1. 获取当前用户(也就是发送者)的用户信息。
2. 发消息时携带当前用户信息。
3. 展示消息时，通过消息体内的用户信息进行展示。

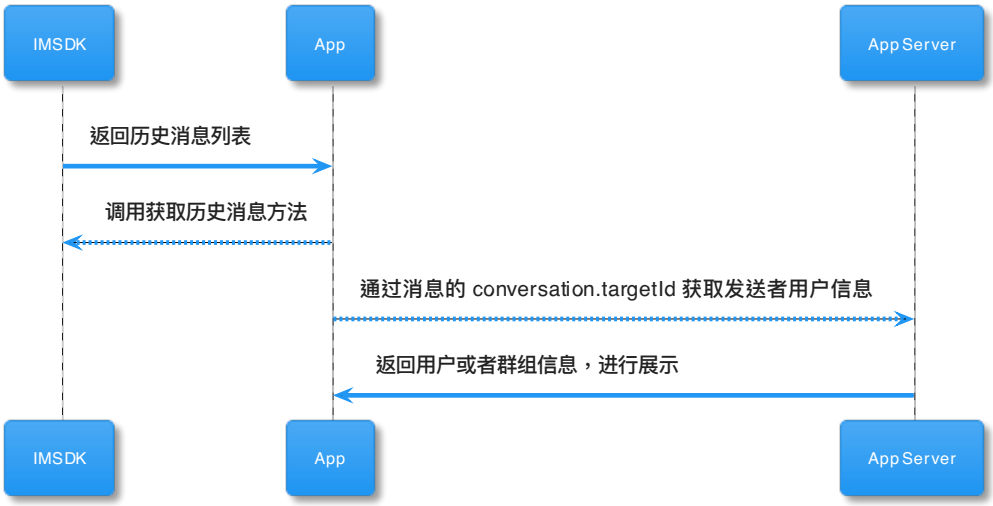
代码示例

```
RCTextMessage *textMsg = [RCTextMessage messageWithContent:@"Hello,RongCloud"];
textMsg.senderUserInfo = [[RCUserInfo alloc] init];
textMsg.senderUserInfo.userId = [RCIMClient sharedRCIMClient].currentUserInfo.userId;
textMsg.senderUserInfo.name = [RCIMClient sharedRCIMClient].currentUserInfo.name;
if ([RCUtilities isLocalPath:[RCIMClient sharedRCIMClient].currentUserInfo.portraitUri]) {
textMsg.senderUserInfo.portraitUri = nil;
} else {
textMsg.senderUserInfo.portraitUri = [RCIMClient sharedRCIMClient].currentUserInfo.portraitUri;
}
```

在会话列表中显示

通过 App Server 获取用户信息。

1. 您的 App Server 封装获取用户或群组信息接口
2. 通过 conversation.targetId 获取用户 ID 或者群组 ID。
3. 将用户或者群组 ID 传入 App Server 暴露的接口中，获取对应用户或群组信息
4. 将信息展示到页面中。



监听连接状态

更新时间:2024-08-30

当 SDK 与融云服务器的连接状态发生变化时，开发者可通过下面代理方法监听 SDK 连接状态。

设置连接监听

```
//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
[[RCIMClient sharedRCIMClient] setConnectionStatusDelegate: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
}
}
```

连接异常状态码说明

RCConnectionStatus 中定义了连接过程中的可能的状态变化。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，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

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。在应用的整个生命周期，此方法只需要调用一次，之后无论是网络异常或者 App 有前后台的切换，SDK 都会自动重连，直到开发者主动断开连接。

准备 Token

用户身份验证令牌 (Token) 代表一个用户在融云的唯一身份标识。获取 Token 必须调用融云服务端 API 的接口。请提供准确的 App Key / Secret，并使用用户 ID 换取对应的 Token。

融云客户端 SDK 不提供获取 Token 的接口。请从您的应用服务端发起 API 调用请求。具体操作请参见融云服务端文档注册用户。

> 为什么不要在客户端获取 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。

获取 Token 后，您可以按需保存 Token，供后续连接时使用。如果 Token 未失效，就不必再向融云请求 Token。具体保存位置取决于应用客户端设计。

Token 有效期

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。

您可以调用服务端 API `/user/token/expire.json` 主动废弃 Token。详见服务端 API 文档 作废 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。

接口调用建议：

- **用户初次登录**：建议调用带超时设置（`timeLimit`） 的接口，并为其设置有效值，保证初次登录如遇到异常情况（如网络较差等）SDK 可以及时回调连接超时，由 APP 自行处理重连。
- **用户后续免密登录**：后续登录说明 token 已经是合法可用的，那么 APP 直接调用不带超时设置的接口，IM 的连接完全由 IM SDK 接口，APP 不需要额外处理。

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。
- 在应用的整个生命周期，此方法只需要调用一次，之后无论是网络异常或者 App 有前后台的切换，SDK 都会自动重连，直到开发者主动断开连接。
- SDK 本身有重连机制，在一个应用生命周期内不要多次调用 `connect` 。否则可能触发多个回调，触发导致回调被清除。

建立 IM 连接（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。在网络较差等导致连接超时的情况下，App 可以利用接口的超时错误回调，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。

```
[[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) {
//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
}
}]
```

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 。

建立 IM 连接（无超时设置）

如果客户端用户已成功登录过 App ，且已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。

> 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。

```
[[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
}
}]
```

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

连接状态码

当回调 errorBlock 时，并且 SDK 不会再进行重连，返回 RCConnectErrorCode 错误码。

错误码	值	说明	处理方案
RC_CONN_TOKEN_INCORRECT	31004	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	1. 检查 SDK 与 App 服务器使用的 App Key 是否一致。 2. 从 APP 服务器获取新的 token ，再重新进行连接。
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_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_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

错误码	值	说明	处理方案
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
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
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
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

1. 在 `31004` 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. 此方法的回调并非为原调用线程，需要进行 UI 操作，请注意切换到主线程。
3. 如果 App 集成了 IMKit，请使用 RCIM 中的同名方法建立与融云服务器的连接，具体请参考 IMKit 对应文档。

重连机制与重连互踢

自动重连机制

更新时间:2024-08-30

SDK 已实现重连机制， 在 App 的整个生命周期内，开发者只需要调用一次连接服务，在需要退出的时候再调用断开连接即可。

重连时间间隔

SDK 会根据网络状态和 App 的行为（前后台等），在连接中断的情况下不断进行重连。

重连的时间间隔是 0s, 0.25s, 0.5s, 1s, 2s, 4s, 8s, 16s, 32s, 64s, 64s, 64s...

当 **APP 切换到前台** 或者 **网络变化** 时，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。

> 一般来说，App 同时在线，不等于 IM 消息通道需要同时在线。因为用户同时只会操作一个手机，只需开通 多设备消息同步，保证各端消息一致即可。

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。

> 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。

断开连接 功能描述

更新时间:2024-08-30

开发者在连接融云服务之后，在需要用户切换、用户注销的操作时，可通过下面方法断开连接，并可根据此方法来指定在用户断开连接之后是否接收消息推送。 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。

参数说明

参数	类型	必填	说明
isReceivePush	BOOL	是	App 在断开连接之后，是否还接收远程推送

代码示例

```
[[RCIMClient sharedRCIMClient] disconnect:YES];
```

多端同时在线

更新时间:2024-08-30

默认的情况下，融云仅支持 1 个 Web 端、1 个 桌面端、1 个移动端同时在线。

开通多设备消息同步功能后，可以支持 Web 端、桌面端和移动端之间的消息同步。且开通此功能后，可以同时支持多个 Web 端同时在线。重新登录时，获取当天收发的所有消息。

开通方式

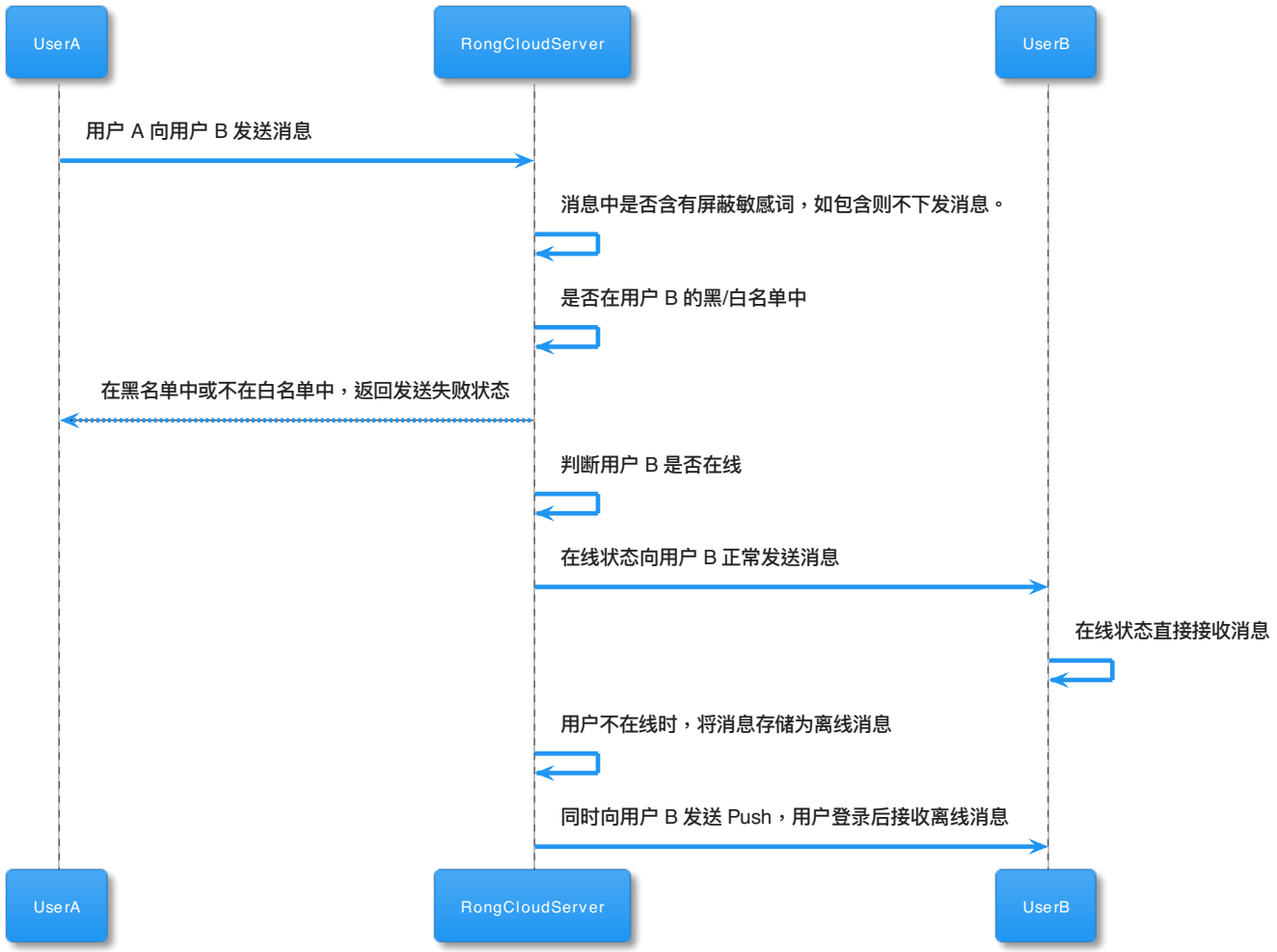
- 开发环境，默认为关闭状态，开启后 30 分钟内生效。
- 生产环境，**IM 旗舰版**或**IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。服务开启、关闭设置完成后 30 分钟内生效。

单聊介绍 概述

更新时间:2024-08-30

适用于应用内两个用户之间一对一聊天方式，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。

消息发送时序图：



主要功能

功能	描述
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。
消息提醒	离线状态，单聊中有新消息时，支持 Push 通知。
本地存储	存储在移动端本地，提供本地消息搜索功能。
历史消息	提供服务端消息存储存功能，需开通单群聊消息云存储，默认存储时长为 6 个月。
消息删除	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。
消息搜索	支持按关键字或用户搜索本地指定会话的消息内容。
消息阅读回执	发送单聊消息后如需要查看消息的阅读状态，可以使用此功能来发送阅读回执请求。

功能	描述
消息撤回	消息发送成功后，在有效时间内可撤回该条消息，默认可撤回时间为 2 分钟，时间可配置。
单聊会话免打扰	可设置指定的单聊会话，收到新的消息后是否进行提醒，默认进行新消息提醒。
单聊黑名单	不想接收到某一用户的消息时，可将此用户加入到黑名单中，应用中的每个用户都可以设置自己的黑名单列表
单聊白名单	对用户之间相互发送消息有限制要求的客户，可使用用户白名单功能，将用户加入白名单后，才能收到该用户发送的单聊消息

注：用户白名单服务与用户黑名单服务不能同时使用，融云默认开启的是用户黑名单服务，如需要开通白名单服务请提交工单申请开通。服务开通 30 分钟后生效，同时黑名单服务不再生效。

消息类型

消息类型	描述
文字消息	用来发送文字类消息，其中可以包括表情、超链接（会自动识别），客户端收到消息后计入未读消息数、进行存储。
语音消息	发送高质量的短语音消息，录制的语音文件存储到融云服务端，语音文件格式为 AAC，时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。
图片消息	用来发送图片类消息，客户端收到消息后计入未读消息数、进行存储。图片缩略图格式为 JPG，大小建议不超过 100k。
GIF 图片消息	用来发送 GIF 动态图片消息，客户端收到消息后计入未读消息数、进行存储。
图文消息	用来发送图文消息，包含一个标题，一段文字内容和一张图片，客户端收到消息后计入未读消息数、进行存储。
文件消息	用来发送文件类消息，客户端收到消息后计入未读消息数、进行存储。
位置消息	用来发送地理位置消息，客户端收到消息后计入未读消息数、进行存储。
小视频消息	用来发送小视频消息，支持录制发送及选择本地视频文件发送两种方式，录制时长不超过 10 秒，本地选择视频文件方式时长不超过 2 分钟，小视频消息小视频文件格式为 .mp4，客户端收到消息后计入未读消息数、进行存储。
合并转发消息	IMKit SDK 中支持将多条消息合并为一条消息进行发送，合并后的消息以 HTML 文件的方式存储到融云服务端，客户端收到消息后计入未读消息数、进行存储。红包、阅后即焚及自定义消息的合并转发功能
命令消息	用来发送通用的指令通知消息，消息内可以定义任意 JSON 内容，与通用命令通知消息的区别是不存储、不计数，此类型消息没有 Push 通知。
自定义消息	融云内置消息类型，无法满足客户业务需求时，可通过自定义消息类型进行实现，接收自定义消息的格式解析及展示处理需要开发者自行实现

内置消息说明

请参见融云服务端文档内置消息类型说明。

好友关系

更新时间:2024-08-30

融云为了客户隐私考虑，既不同步又不保存用户的好友关系。所以，所有用户的好友关系都需要开发者自己实现、管理维护。

如果使用的是 IMKit 进行集成，则会话及好友列表中显示好友的昵称及头像信息，需要 App 设置一个用户信息提供者给 IMKit，以便 IMKit 通过用户信息提供者，来实现在聊天界面和会话列表页中显示好友的昵称和头像。

获取全部会话　　获取本地会话列表

更新时间:2024-08-30

开发者可通过下面接口获取 SDK 在本地数据库生成的会话列表，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。

参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array
count	int	是	获取的数量
startTime	long long	是	会话的时间戳，获取的是这个时间戳之前的会话列表，0 表示从最新开始获取

代码示例

```
NSArray *conversationList = [[RCIMClient sharedRCIMClient] getConversationList:@[@(ConversationType_PRIVATE)]
count:100 startTime:0];
```

获取远端会话列表

为提高性能和获取效率，SDK 的会话列表会根据消息的收发直接在本地数据库生成，不需要从服务端获取。

常见问题

Q1: 当用户卸载重新安装后发现会话列表为空或者部分会话丢失
A1: 由于会话列表是从本地数据库获取的, 是在 SDK 内部数据库存储的，所以当用户在卸载的时候会删除本地数据库，导致重新安装后会话列表为空，而会出现部分会话的原因是因为开启了 离线消息补偿 功能，具体操作是在后台购买 多设备消息同步 功能，这个功能里默认涵盖了今天(当天 0 点)的离线消息补偿，所以当用户在新设备登陆会触发离线消息补偿功能，从而获取到部分会话，造成部分会话丢失的错觉。如果想要更多天的离线消息补偿，可 提交工单 修改，最多支持 7 天，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。

删除全部会话

清除本地会话列表

更新时间:2024-08-30

开发者可调用 删除指定类型 会话接口，并传入业务中用到的所有的会话类型即可清除 SDK 在本地数据库内生成的会话列表数据。

清除远端会话列表

由于客户端的会话列表是根据消息在本地生成的，融云服务端并不存储会话列表，因此无需清除远端会话列表。

获取指定会话

功能描述

更新时间:2024-08-30

开发者可以通过下面接口来获取某个会话的详细信息。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId

返回参数说明

返回值	返回类型	说明
conversation	RCConversation	会话的对象

代码示例

```
RCConversation *conversation = [[RCIMClient sharedRCIMClient] getConversation:ConversationType_PRIVATE
targetId:@"接收方的 userId"];
```

删除指定会话

删除指定会话

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] removeConversation:ConversationType_PRIVATE targetId:@"接收方的
userId"];
```

按会话类型删除

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient]
clearConversations:@[@(ConversationType_PRIVATE),@(ConversationType_SYSTEM),@(ConversationType_GROUP)]];
```

会话草稿信息

获取草稿

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	接收方的 userId

返回参数说明

返回值	返回类型	说明
draft	NSString	该会话中的草稿

代码示例

```
NSString *draft = [[RCIMClient sharedRCIMClient] getTextMessageDraft:ConversationType_PRIVATE targetId:@"接收方的
userId"];
```

保存草稿

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	接收方的 userId
content	NSString	是	草稿信息

返回参数说明

返回值	返回类型	说明
success	BOOL	是否保存成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] saveTextMessageDraft:ConversationType_PRIVATE targetId:@"接收方的
userId" content:@"这个是草稿内容"];
```

删除草稿

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	接收方的 userId

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] clearTextMessageDraft:ConversationType_PRIVATE targetId:@"接收方的
userId"];
```

会话未读数

获取所有会话未读数

更新时间:2024-08-30

参数说明

返回参数说明

返回值	返回类型	说明
count	int	所有的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient] getTotalUnreadCount];
```

获取单个会话未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId

返回参数说明

返回值	返回类型	说明
count	int	该会话内的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient] getUnreadCount:ConversationType_PRIVATE targetId:@"接收方的 userId"];
```

按会话类型获取未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypes	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array
isContain	BOOL	是	是否包含免打扰消息的未读数

返回参数说明

返回值	返回类型	说明
count	int	该类型的会话中所有的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient]
getUnreadCount:@[@(ConversationType_PRIVATE),@(ConversationType_GROUP)] containBlocked:YES];
```

清除单个会话未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
timestamp	long long	是	此会话已阅读的最后一条消息的发送时间

返回参数说明

返回值	返回类型	说明
success	BOOL	是否清除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] clearMessagesUnreadStatus:ConversationType_PRIVATE targetId:@"接收方的 userId" time:0];
```

会话免打扰 设置免打扰状态

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
isBlocked	BOOL	是	是否屏蔽消息提醒
success	BOOL	是	设置成功的回调
error	BOOL	是	设置失败的回调

回调参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] setConversationNotificationStatus:ConversationType_PRIVATE
targetId:self.userId
isBlocked:swch.on
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

获取免打扰状态

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
success	BOOL	是	设置成功的回调
error	BOOL	是	设置失败的回调

回调参数说明

`success` 说明：

回调参数	回调类型	说明
nStatus	RCConversationNotificationStatus	会话设置的消息提醒状态

error 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] getConversationNotificationStatus:ConversationType_PRIVATE
targetId:self.userId
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

获取免打扰状态列表

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
array	NSArray	屏蔽消息提醒的会话RCConversation的列表

代码示例

```
NSArray *array = [[RCIMClient sharedRCIMClient] getBlockedConversationList:@[@(ConversationType_PRIVATE)]];
```

设置免打扰时间

参数说明

输入参数说明

参数	类型	必填	说明
startTime	NSString	是	开始消息免打扰时间，格式为 HH:MM:SS
spanMins	int	是	需要消息免打扰分钟数， 0 < spanMins < 1440 您设置的起始时间是 00：00， 结束时间为 23：59，则 spanMins 为 23 * 60 + 59 = 1339 分钟
success	BOOL	是	屏蔽成功的回调
error	BOOL	是	屏蔽失败的回调

回调参数说明

error 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] setNotificationQuietHours:@"00:00:00"
spanMins:1439
success:^{}
error:^(RCErrorCode status) {}];
```

会话置顶 功能描述

更新时间:2024-08-30

开发者可通过下面接口来设置会话是否置顶，置顶的状态将会被同步到服务端，切换设备后置顶状态也会一并同步下来。

设置是否置顶

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
isTop	BOOL	是	是否置顶

返回参数说明

返回值	返回类型	说明
success	BOOL	设置是否成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] setConversationToTop:ConversationType_PRIVATE targetId:@"接收方的
userId" isTop:YES];
```

获取置顶列表

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypes	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
conversations	NSArray <RCConversation *>	置顶的会话RCConversation的列表

代码示例

```
NSArray *conversations = [[RCIMClient sharedRCIMClient]
getTopConversationList:@[@(ConversationType_PRIVATE),@(ConversationType_GROUP)]];
```

会话标签

更新时间:2024-08-30

警告

该功能仅在5.0以上版本SDK中提供！

消息发送 属性描述

更新时间:2024-08-30

消息属性	描述
消息类名	各端消息名
存储属性	存储 / 不存储
离线属性	缓存 / 不缓存
推送属性	是/否
ObjectName	传输层名称，与消息类名一一对应
计数属性	计数 / 不计数
消息尺寸	128 KB
推送内容	详见各消息类送方式

存储属性

存储属性	存储分类	支持平台	详细描述
存储	客户端	Android、iOS	发送、接收该消息后，本地数据库存储 Web 端 和 小程序端因本地存储不可靠，不支持客户端消息存储，但可通过历史消息云存储服务获取历史记录
存储	云端	Android、iOS、Web	默认不在云端进行存储，需开通单群聊消息云存储服务，开通后，可在融云服务端存储 6 个月的历史消息，供客户端按需拉取
不存储	客户端	Android、iOS	发送、接收该消息后，本地数据库不存储
不存储	云端	Android、iOS、Web	无论是否开通历史消息云存储服务，该消息均不存储

计数属性

接收收到消息时，会话是否累计未读数。

计数属性	支持平台	详细描述
计数	iOS、Android、Web	会话未读消息数 + 1，该属性只影响会话列表未读数计数，App 应用角标可根据每个会话列表未读数累加获得
不计数	iOS、Android、Web	会话未读消息数不变

离线属性

接收人当前不在线时，是否进行离线缓存。

离线属性	详细描述
存储	消息进行离线缓存，默认 7 天。接收人在 7 天内上线，均可接收到该消息。超过 7 天后，消息被离线缓存淘汰。如有需要，可通过云端存储拉取到该消息
不存储	消息不进行离线缓存，所以只有接收人在线时，才可收到该消息。该消息不进行历史消息云存储、不进入云端存储（Log 日志）、不进行消息同步（消息路由）

推送属性

接收人是否接收推送，当离线属性为 存储 时，该属性生效。离线属性为 不存储 时属性无效。

由于 Web、小程序、PC 端没有推送平台，无法收到推送提醒。

推送属性	平台	推送方式	详细描述
推送	iOS、Android	APNs、华为、小米、魅族、OPPO、vivo、FCM、融云	当有离线缓存消息时，进行远程推送提醒，内容为该推送提醒显示的内容
不推送	iOS、Android	--	当有离线缓存消息时，不进行远程推送提醒

发送普通消息

通过消息体发送

接口原型：

```
- (RCMessage *)sendMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
option:(RCSendMessageOption *)option
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode nErrorCode, long messageId))errorBlock;
```

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
option	RCSendMessageOption	否	消息的相关配置
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

error 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

通过消息发送

> ⚠ 警告
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock;
```

RCMessage 添加消息配置的对象

```
/*!
消息配置，可用来设置静默消息（不进行通知提醒）
*/
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
消息推送配置，详细查看 配置消息推送
*/
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

参数说明

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息对象
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIMClient sharedRCIMClient] sendMessage:message pushContent:pushContent pushData:nil successBlock:nil
errorBlock:nil];
```

> ⚠ 警告
>
> 注意 messageid 无特殊情况应传 **0**

文本消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCTextMessage	RCMessageContent	RC:TxtMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
content	NSString	文本消息的内容

代码示例

```
RCTextMessage *textMessage = [RCTextMessage messageWithContent:@"消息内容"];

[[RCIMClient sharedRCIMClient] sendMessage:ConversationType_PRIVATE targetId:@"接收方的 userId"
content:textMessage pushContent:nil pushData:nil option:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

Emoji 消息

iOS 端发送 Emoji 消息，开发者直接使用 文本消息 发送即可。如有特殊需求需要开发者自己 自定义消息 实现。

位置消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCLocationMessage	RCMessageContent	RC:LBSMsg	存储	计数	存储	推送	[位置]

属性说明

构建参数说明：

参数	类型	说明
image	UIImage	地理位置的缩略图
location	CLLocationCoordinate2D	地理位置的二维坐标
locationName	NSString	地理位置的名称

代码示例

```
RCLocationMessage *locationMessage = [RCLocationMessage messageWithLocationImage:image location:location
locationName:locationName];

[[RCIMClient sharedRCIMClient] sendMessage:ConversationType_PRIVATE targetId:@"接收方的 userId"
content:locationMessage pushContent:nil pushData:nil option:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

发送媒体消息

通过消息体发送

接口原型：

```
- (RCMessage *)sendMediaMessage:(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
cancel:(void (^)(long messageId))cancelBlock;
```

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
content	RCMessageContent	是	消息的内容要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
messageId	long	消息的ID

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

`cancel` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

通过消息发送

> 提示
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMediaMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
progress:(void (^)(int progress, RCMessage *progressMessage))progressBlock
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock
cancel:(void (^)(RCMessage *cancelMessage))cancelBlock;
```

RCMessage 添加消息配置的对象

```
/*!
消息配置，可用来设置静默消息（不进行通知提醒）
*/
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
消息推送配置，详细查看 配置消息推送
*/
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息实体
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
progressMessage	RCMessage	消息实体

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

`cancel` 说明：

回调参数	回调类型	说明
cancelMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIM sharedRCIM] sendMediaMessage:message pushContent:pushContent pushData:nil progress:nil successBlock:nil
errorBlock:nil cancel:nil];
```

> ⚠ 警告
>
> **注意 messageid 无特殊情况应传 0**

图片消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCImageMessage	RCMediaMessageContent	RC:ImgMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
imageData	NSData	图片的原始数据

代码示例

```
RCImageMessage *message = [RCImageMessage messageWithImageData:data];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId"
content:message pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

语音消息

> ⚠ 警告
>
> SDK 从 2.9.19 版本开始支持发送 RCHQVoiceMessage。

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCHQVoiceMessage	RCMediaMessageContent	RC:HQVCMsg	存储	计数	存储	推送	[语音]

构建参数说明：

参数	类型	说明
localPath	NSString	语音的本地路径
duration	long	语音消息的时长，单位为秒

示例代码

```
RCHQVoiceMessage *hqVoiceMsg = [RCHQVoiceMessage messageWithPath:path duration:duration];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId"
content:hqVoiceMsg pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

文件消息

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCFileMessage	RCMediaMessageContent	RC:FileMsg	存储	计数	存储	推送	[文件] + 文件名，如：[文件] 123.txt

构建参数说明：

参数	类型	说明
localPath	NSString	文件的本地路径

代码示例

```
RCFileMessage *fileMessage = [RCFileMessage messageWithFile:path]

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId"
content:fileMessage pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

小视频消息

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 。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCSightMessage	RCMediaMessageContent	RC:SightMsg	存储	计数	存储	推送	[小视频]

属性说明

构建参数说明：

参数	类型	说明
path	NSData	视频文件本地路径
image	UIImage	视频首帧缩略图
duration	NSUInteger	视频时长， 单位为秒

代码示例

```
RCSightMessage *sightMessage = [RCSightMessage messageWithLocalPath:url thumbnail:image duration:duration];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId"
content:sightMessage pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

GIF 消息

> ⚠ 警告
>
> 从 SDK 2.9.21 版本开始支持发送 GIF 图片消息，默认可选择发送的 GIF 图片大小为 2M。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCGIFMessage	RCMediaMessageContent	RC:GIFMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
gifImageData	NSData	GIF 图的数据
width	long	GIF 图的宽
height	long	GIF 图的高

代码示例

```
RCGIFMessage *gifMsg = [RCGIFMessage messageWithGIFImageData:gifImageData
width:gifImage.size.width
height:gifImage.size.height];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_PRIVATE targetId:@"接收方的 userId"
content:gifMsg pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

发送自定义消息

除了使用 SDK 内置消息外，开发者可根据自己的业务需求来自定义消息，自定义消息的类型、消息结构需要确保多端一致，否则将出现无法互通的问题。

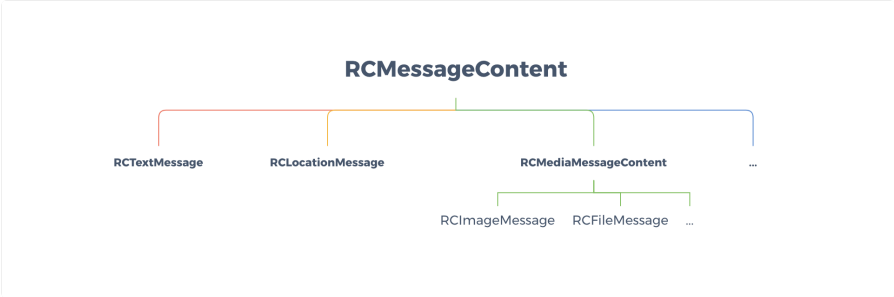
注册消息

注册消息共分为两步。

> 提示
>
> 1. 定义消息类型
> 2. 注册消息类型

1.定义消息类型

开发者可根据业务需求来选择继承消息基类 `RCMessageContent` 或者 `RCMediaMessageContent`，两者之前的区别可参考下图:



定义消息类型需要该类遵守下面三个协议

一. 编解码协议

协议原型：

```
@protocol RCMessageCoding <NSObject>
```

`RCMessageCoding` 主要有三个功能：

1. 提供消息唯一标识符
2. 消息发送时将消息中的所有信息编码为 JSON 数据传输
3. 消息接收时将 JSON 数据解码还原为消息对象。

该协议为必须遵守的协议，遵守该协议需实现下面三个方法。

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：

```
+ (NSString *)getObjectName;
```

> ⚠ 警告
>
> 1. 定义的消息类型名（objectname），需要在各个平台上保持一致，否则消息无法互通
> 2. 请勿使用 `RC:` 开头的类型名，以免和SDK默认的消息名称冲突
> 3. 所有自定义消息必须实现此协议，否则将无法正常传输和使用。

二. 存储协议

协议原型：

```
@protocol RCMessagePersistentCompatible <NSObject>
```

`RCMessageCoding` 主要有两个功能:

1. 指明此消息类型在本地和服务端是否存储
2. 指明此消息类是否计入未读消息数

该协议为必须遵守的协议，遵守该协议需实现下面方法:

消息的存储计数策略：

```
+ (RCMessagePersistent)persistentFlag;
```

RCMessagePersistent 说明:

类型	说明
MessagePersistent_NONE	表示消息不存储、不计数。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_ISCOUNTED	表示客户端收到消息后，要进行未读消息计数（未读消息数增加 1）。
MessagePersistent_ISPERSISTED	表示消息会存储到数据库。
MessagePersistent_STATUS	表示状态消息，该类型消息不存储不计数。状态类消息表示的是即时的状态，如果接收方不在线，则无法再收到该状态消息。

自定义消息持久化标识案例分析：

1. 需要存储和未读计数的消息（类似文本消息）：`MessagePersistent_ISCOUNTED | MessagePersistent_ISPERSISTED`
2. 需要确保收到，但不需要展示的消息，多为命令消息（类似撤回命令消息）：`MessagePersistent_NONE`
3. 对方在线接收，不在线就不用接收的消息（类似输入状态消息）：`MessagePersistent_STATUS`

三. 内容摘要协议

协议原型：

```
@protocol RCMessageContentView
```

`RCMessageContentView` 主要有一个功能:

1. 设置此消息在会话列表和本地通知中显示消息的摘要

该协议为非必须遵守的协议，遵守该协议需实现下面方法。

设置消息摘要：

```
- (NSString *)conversationDigest;
```

2.注册消息类型

开发者在进行完自定义消息类后，需要在 SDK 初始化之后，注册此自定义消息类。

```
- (void)registerMessageType:(Class)messageClass;
```

> ⚠ 警告
>
> 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。

发送消息

继承 RCMessageContent，请使用 发送普通消息 的接口进行发送。

继承 RCMediaMessageContent，请使用 发送媒体消息 的接口进行发送。

配置消息推送

RCMessagePushConfig 属性介绍

> ⚠ 警告
>
> 从 SDK 4.0.4 版本开始支持此功能，针对每条 Message 都可以设置此属性，详细查看以下参数说明。

参数	类型	说明
pushTitle	NSString	推送标题，如果没有设置，会使用 SDK 默认的标题显示规则
pushContent	NSString	推送内容，如果没有，则使用发送消息的 pushContent，最后则会使用 SDK 默认的标题显示规则
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
iOSConfig	RCiOSConfig	iOS 平台相关配置
androidConfig	RCAndroidConfig	Android 平台相关配置

RCiOSConfig 属性介绍

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）

RCAndroidConfig 属性介绍

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
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
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
typeVivo	NSString	VIVO 推送通道类型，开发者集成了 VIVO 推送，需要指定推送类型时，可进行设置。 目前可选值 "0"(运营消息) 和 "1"(系统消息)

Channel ID 需要由开发者进行创建，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

示例代码

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
RCMessagePushConfig *pushConfig = [[RCMessagePushConfig alloc] init];
pushConfig.pushTitle = @"通知标题";
pushConfig.pushContent = @"通知内容";
pushConfig.pushData = @"通知的 pushData";
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.androidConfig.notificationId = @"Android 的通知 id";
pushConfig.androidConfig.channelIdMi = @"小米的 channelId";
pushConfig.androidConfig.channelIdHW = @"华为的 channelId";
pushConfig.androidConfig.channelIdOPPO = @"OPPO 的 channelId";
pushConfig.androidConfig.typeVivo = @"vivo 的类型";
pushConfig.forceShowDetailContent = true;
message.messagePushConfig = pushConfig;
[[RCIMClient sharedRCIMClient] sendMessage:message pushContent:nil pushData:nil successBlock:nil
errorBlock:nil];
```

消息接收

功能描述

更新时间:2024-08-30

开发者可通过此接口拦截到 SDK 接收到的消息，并进行响应的业务操作。

实现方法

SDK 通过消息监听接收消息，详细可参考 设置消息监听 文档

历史消息获取 本地获取

更新时间:2024-08-30

开发者可通过下面接口获取 SDK 存储在本地数据库的历史消息。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
oldestMessageId	long	是	最后一条消息的 messageId，获取此消息之前的 count 条消息，第一次调用应设置为 -1。
count	int	是	需要获取的消息数量，每次最多获取 20 条

返回参数说明

返回值	返回类型	说明
history	NSArray	消息实体RCMessage对象列表

返回参数说明

代码示例

```
NSArray *history = [[RCIMClient sharedRCIMClient] getHistoryMessages:ConversationType_PRIVATE
targetId:@"接收方的 userId"
oldestMessageId:lastMessageID
count:count];
```

远端获取

开发者可通过下面接口获取 SDK 存储在服务端的历史消息。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。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	目标会话 ID

参数	类型	必填	说明
option	RCRemoteHistoryMsgOption	是	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

回调参数说明

`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

代码示例

```
RCRemoteHistoryMsgOption *option = [RCRemoteHistoryMsgOption new];
option.recordTime = recordTime;
option.count = 10;
option.order = RCRemoteHistoryOrderDesc;

[[RCIMClient sharedRCIMClient] getRemoteHistoryMessages:ConversationType_PRIVATE targetId:@"接收方的 userId"
option:option success:^(NSArray *messages, BOOL isRemaining) {

} error:^(RCErrorCode status) {

}];
```

消息回执 功能描述

更新时间:2024-08-30

开发者可使用此功能实现消息已读未读功能的展示。

当 A 给 B 发送了一条消息，B 在未阅读之前 A 用户显示未读，当 B 用户阅读并调用发送回执接口之后，A 用户可在监听回执中收到通知，此时可根据对应的数据内容将发送的消息显示为已读。

发送回执

当收到一条消息时，可以通过以下方法发送一条回执给发送方。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	消息发送方的 userId
timestamp	long long	是	可配置的参数
successBlock	Block	是	发送成功的回调
errorBlock	Block	是	发送失败的回调

返回参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	发送失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] sendReadReceiptMessage:ConversationType_PRIVATE
targetId:@"targetId"
time:sentTime
success:nil
error:nil];
```

接收回执

收到消息已读回执之后，IMLib 会分发一个名为 `RCLibDispatchReadReceiptNotification` 的通知，开发者收到这个消息之后可以更新这个会话中 messageTime 以前的消息 UI 为已读（底层数据库消息状态已经改为已读）。

参数说明

Notification 的 object 为 nil，userInfo 为 NSDictionary 对象。

NSDictionary 对象的 key 值说明:

key值	类型	说明
cType	NSNumber	会话类型
messageTime	NSNumber	已阅读的最后一条消息的sendTime
tId	NSString	A 的 targetID
fId	NSString	B 的 targetId

代码示例

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

```
- (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"];
}
```

消息撤回 消息撤回

更新时间:2024-08-30

消息发送方可通过下面方法撤回已发送成功的消息。

参数说明

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	需要撤回的消息
pushContent	NSString	否	push 显示的内容
successBlock	Block	-	撤回成功的回调
errorBlock	Block	-	撤回失败的回调

返回参数说明

`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	撤回失败错误码

代码示例

```
[[RCIMClient sharedRCIMClient] recallMessage:msg pushContent:nil success:^(long messageId) {

} error:^(RCErrorCode errorcode) {

}];
```

监听撤回

消息接收方可通过下面方法监听到被撤回的消息。

实现此功能需要开发者遵守 `RCIMClientReceiveMessageDelegate` 协议。

参数说明

代理对象参数说明：

传入参数	类型	必填	说明
delegate	id	是	IMLib 消息接收监听器
userData	id	否	用户自定义的监听器 Key 值，可以为 nil

协议方法参数说明：

参数	类型	说明
messageId	long	被撤回的消息ID

代码示例

1. 设置代理对象:

```
[[RCIMClient sharedRCIMClient] setReceiveMessageDelegate:self object:nil];
```

2. 实现协议方法:

```
- (void)onMessageRecalled:(long)messageId{

}
```

消息转发

更新时间:2024-08-30

消息转发调用发送消息接口发送即可，调用 SDK 发送消息接口需考虑 SDK 每秒 5 条消息的限制

超过每秒 5 条限制可使用 Server API 进行发送

消息删除 本地删除

更新时间:2024-08-30

开发者可通过下面接口对本地数据库的消息进行删除。

通过消息 ID 删除

输入参数说明：

参数	类型	必填	说明
messageIds	NSArray<NSNumber *>	是	消息ID的列表，元素需要为 NSNumber 类型

返回参数说明：

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例：

```
BOOL success = [[RCIMClient sharedRCIMClient] deleteMessages: @[@(message.messageId)]];
```

通过时间戳删除

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
recordTime	long long	是	清除消息时间戳，传入的时间戳必须小于等于当前会话最后一条消息的 sentTime 传 **0** 表示清除所有消息， 传入其他值表示清除小于等于 recordTime 的消息
clearRemote	BOOL	是	是否同时删除服务端消息
successBlock	Block	-	获取成功的回调
errorBlock	Block	-	获取失败的回调

回调参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	撤回失败错误码

代码示例：

```
[[RCIMClient sharedRCIMClient] clearHistoryMessages:ConversationType_PRIVATE
targetId:@"接收方的 userId"
recordTime:recordTime
clearRemote:NO
success:^{}
error:^(RCErrorCode status) {}];
```

通过会话删除

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId

返回参数说明：

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例：

```
BOOL success = [[RCIMClient sharedRCIMClient] clearMessages:ConversationType_PRIVATE targetId:@"接收方的
userId"];
```

远端删除

开发者可通过下面接口对远端的消息进行删除。

通过时间戳删除

通过时间戳删除该时间之前的服务端历史消息，删除后不可恢复。

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
recordTime	long long	是	清除消息时间戳，传入的时间戳必须小于等于当前会话最后一条消息的 sentTime 传 **0** 表示清除所有消息， 传入其他值表示清除小于等于 recordTime 的消息
successBlock	Block	-	删除成功的回调
errorBlock	Block	-	删除失败的回调

回调参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	删除失败错误码

代码示例：

```
[[RCIMClient sharedRCIMClient]clearRemoteHistoryMessages:ConversationType_PRIVATE
targetId:@"接收方的 userId"
recordTime:sentTime
success:^{}
error:^(RCErrorCode status) {}];
```

通过消息删除

仅删除服务端指定一条或多条历史消息。

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
messages	NSArray<RCMessage *>	是	将被删除的消息列表
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

回调参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	删除失败错误码

代码示例：

```
[[RCIMClient sharedRCIMClient] deleteRemoteMessage:ConversationType_PRIVATE
targetId:@"接收方的 userId"
messages:messages success:^{}
error:^(RCErrorCode status) {}];
```

消息搜索

通过关键字搜索

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	会话ID
keyword	NSString	是	关键字
count	int	是	最大的查询数量
startTime	long long	是	查询 startTime 之前的消息（传 0 表示不限时间）

返回参数说明

返回值	返回类型	说明
searchResult	NSArray<RCMessage *>	匹配的消息列表

代码示例

```
NSArray *searchResult = [[RCIMClient sharedRCIMClient] searchMessages:ConversationType_PRIVATE
targetId:@"接收方的 userId"
keyword:searchText
count:50
startTime:0];
```

通过用户 ID 搜索

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	会话ID
userId	NSString	是	搜索用户ID
count	int	是	最大的查询数量
startTime	long long	是	查询 startTime 之前的消息（传 0 表示不限时间）

返回参数说明

返回值	返回类型	说明
searchResult	NSArray<RCMessage *>	匹配的消息列表

代码示例

```
NSArray *searchResult = [[RCIMClient sharedRCIMClient] searchMessages:ConversationType_PRIVATE
targetId:@"接收方的 userId"
userId:@"userId"
count:50
startTime:0];
```

消息插入　　插入发送消息

更新时间:2024-08-30

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。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId
sentStatus	RCSentStatus	是	发送状态
content	RCMessageContent	是	消息的内容
sentTime	long long	是	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）

返回参数说明

返回值	返回类型	说明
message	RCMessage	插入的消息实体

代码示例

> ⚠ 警告
>
> 此方法如果 sentTime 有问题会影响消息排序，慎用！

```
RCMessage *message = [[RCIMClient sharedRCIMClient] insertOutgoingMessage:ConversationType_PRIVATE
targetId:@"接收方的 userId"
sentStatus:SentStatus_SENT
content:content
sentTime:sentTime];
```

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。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，单聊传入 ConversationType_PRIVATE
targetId	NSString	是	接收方的 userId

参数	类型	必填	说明
senderUserId	NSString	是	发送者ID
sentStatus	RCReceivedStatus	是	接收状态
content	RCMessageContent	是	消息的内容
sentTime	long long	是	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）

返回参数说明

返回值	返回类型	说明
message	RCMessage	插入的消息实体

代码示例

> ⚠ 警告
>
> 此方法如果 sentTime 有问题会影响消息排序，慎用！

```
RCMessage *tempMessage = [[RCIMClient sharedRCIMClient] insertIncomingMessage:ConversationType_PRIVATE
targetId:@"接收方的 userId"
senderUserId:@"senderUserId"
receivedStatus:ReceivedStatus_READ
content:content];
```

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。

- 仅支持单聊、群聊会话类型。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。

设置消息是否支持扩展

发送消息时，该条消息设置了可扩展属性后，才能设置扩展信息，消息可扩展属性相关功能如下：

```
/*!
消息是否可以包含扩展信息

@discussion 该属性在消息发送时确定，发送之后不能再做修改
@discussion 扩展信息只支持单聊和群组，其它会话类型不能设置扩展信息
*/
@property (nonatomic, assign) BOOL canIncludeExpansion;

/*!
消息扩展信息列表

@discussion 扩展信息只支持单聊和群组，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 64 ，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;
```

更新扩展信息

- 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。
- 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。

输入参数说明：

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。 • Value 最大 64 个字符。
messageUId	NSString	消息 messageUId
successBlock	Block	成功的回调
errorBlock	Block	失败的回调

返回参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	失败的错误码

```
[[RCIMClient sharedRCIMClient] updateMessageExpansion:expansionDic messageUId:messageUId success:^{

} error:^(RCErrorCode status) {

}];
```

删除扩展信息

输入参数说明：

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

返回参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	失败的错误码

```
[[RCIMClient sharedRCIMClient] removeMessageExpansionForKey:keyArray messageUId:messageUId success:^{

} error:^(RCErrorCode status) {

}];
```

设置回调

实现此功能需要开发者遵守 `RCMessageExpansionDelegate` 协议

代理对象参数说明：

参数	类型	必填	说明
delegate	id	是	IMLib 消息扩展监听器

```
[RCIMClient sharedRCIMClient].messageExpansionDelegate = self;
```

协议方法参数说明：

参数	类型	说明
expansionDic	NSDictionary	消息扩展信息中更新的键值对
message	RCMessage	消息

```
- (void)messageExpansionDidUpdate:(NSDictionary<NSString *,NSString *> *)expansionDic message:(RCMessage
*)message{

}
```

参数	类型	说明
keyArray	NSArray	消息扩展信息中删除的键值对 key 列表
message	RCMessage	消息

```
- (void)messageExpansionDidRemove:(NSArray<NSString *> *)keyArray message:(RCMessage *)message{

}
```

群聊介绍 概述

更新时间:2024-08-30

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。

主要功能

功能	描述
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。
消息提醒	离线状态，群组中有新消息时，支持 Push 通知。
本地存储	存储在移动端本地，提供本地消息搜索功能。
历史消息	提供服务端消息存储存功能，需开通单群聊消息云存储，默认存储时长为 6 个月。
消息删除	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。
消息搜索	支持按关键字或用户搜索本地指定会话的消息内容。
群消息阅读回执	发送群消息后如需要查看消息的阅读状态，可以使用此功能来发送阅读回执请求。
消息撤回	消息发送成功后，在有效时间内可撤回该条消息，默认可撤回时间为 2 分钟，时间可配置。
群聊会话免打扰	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。
创建群组	App 内的群组数量没有限制，默认一个群上限为 3000 人，可调整群上限，需要提交工单申请开通。
加入群组	每个群最大至 3000 人，一个用户可加入多个群组，没有限制。默认加入群组后，只能查看加入后群组中产生的消息，如需要查看加入前的消息，则需要开通单群聊消息云存储后，再开通查看加入前群消息功能
退出群组	将用户从群中移除，不再接收该群组的消息。
解散群组	将指定群组解散，所有成员都无法再接收该群的消息。
群成员查询	获取指定群组中群成员用户 Id。
刷新群组信息	目前支持更新群组名称。
同步用户所属群组	在集成融云前 App Server 已有群组数据，可使用此服务进行同步。
群组成员禁言	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。
群组整体禁言	指定群组所有成员不能发送消息，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。
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。
发送群组消息	向群组中所有成员发送消息。
发送群组定向消息	向群中指定的一个或多个用户发送消息，群中其他用户无法收到该消息。
发送群组@消息	发送群组中指定群成员需要特别关注的消息。
发送群组状态消息	群组中在线用户会收到此条消息，离线用户不会再收到此条消息，状态消息不计数、不存储。

与聊天室的区别

融云提供群组与聊天室业务，其主要区别如下，客户可根据自己的业务场景进行选择：

功能	群组（group）	聊天室（Chatroom）
场景	类似微信的群组，无论是否在线都会接收消息	只有在线用户可接收消息，可用于直播、社区、游戏、广场交友、兴趣讨论等场景。
离线缓存消息	支持离线消息存储，存储时间可设置（1 ～ 7 天），默认存储 7 天。	无离线消息，只有在线用户才可收到聊天室消息
人数限制	默认一个群上限为 3000 人	聊天室人数无上限
消息提醒	离线状态，群组中有新消息时，支持远程推送（PUSH）通知	离开聊天室后不再接收消息
本地存储	移动端本地数据库存储，提供本地消息搜索接口	退出聊天室后同时删除本地聊天室消息，不支持消息搜索功能
云端存储	需开通单群聊消息云存储，可以提供 6 - 36 个月存储服务	需开通聊天室消息云存储，可以提供 2 - 36 个月存储服务
用户加入限制	一个用户可加入多个群组，无限制	默认一个用户只能加入一个聊天室，加入多个聊天室功能可在控制台自行开通
加入后消息获取逻辑	默认加入群组后，只能查看加入后群组中产生的消息。如需要查看群历史消息，则需要开通单群聊消息云存储后，再开通“查看加入前群消息”功能	加入后可获取聊天室中最新的 50 条消息。
销毁/解散逻辑	需要通过 AppServer 自行调用解散群组接口。	提供销毁聊天室接口，可通过 AppServer 调用。同时聊天室中 1 小时内没有消息产生时，将自动销毁聊天室。
消息可靠度	100% 可靠，不丢消息。	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。
相关接口调用	SDK 不提供群组管理功能接口，通过 Server API 提供群组功能接口。	SDK 和 Server API 同时提供功能接口，销毁聊天室操作只能通过 Server API 方式调用。
发送消息频率	每个客户端 5 条/秒；服务端调用，20 条/秒	每个客户端 5 条/秒；服务端调用，100 条/秒

消息类型

消息类型	描述
文字消息	用来发送文字类消息，其中可以包括表情、超链接（会自动识别），客户端收到消息后计入未读消息数、进行存储。
语音消息	发送高质量的短语音消息，录制的语音文件存储到融云服务端，语音文件格式为 AAC，时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。
图片消息	用来发送图片类消息，客户端收到消息后计入未读消息数、进行存储。图片缩略图格式为 JPG，大小建议不超过 100k。
GIF 图片消息	用来发送 GIF 动态图片消息，客户端收到消息后计入未读消息数、进行存储。
图文消息	用来发送图文消息，包含一个标题，一段文字内容和一张图片，客户端收到消息后计入未读消息数、进行存储。
文件消息	用来发送文件类消息，客户端收到消息后计入未读消息数、进行存储。
位置消息	用来发送地理位置消息，客户端收到消息后计入未读消息数、进行存储。
小视频消息	用来发送小视频消息，支持录制发送及选择本地视频文件发送两种方式，录制时长不超过 10 秒，本地选择视频文件方式时长不超过 2 分钟，小视频消息小视频文件格式为 .mp4，客户端收到消息后计入未读消息数、进行存储。
合并转发消息	IMKit SDK 中支持将多条消息合并为一条消息进行发送，合并后的消息以 HTML 文件的方式存储到融云服务端，客户端收到消息后计入未读消息数、进行存储。红包、阅后即焚及自定义消息的合并转发功能
命令消息	用来发送通用的指令通知消息，消息内可以定义任意 JSON 内容，与通用命令通知消息的区别是不存储、不计数，此类型消息没有 Push 通知。

消息类型	描述
自定义消息	融云内置消息类型，无法满足客户业务需求时，可通过自定义消息类型进行实现，接收自定义消息的格式解析及展示处理需要开发者自行实现

内置消息说明

请参见融云服务端文档内置消息类型说明。

获取全部会话　　获取本地会话列表

更新时间:2024-08-30

开发者可通过下面接口获取 SDK 在本地数据库生成的会话列表，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。

参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array
count	int	是	获取的数量
startTime	long long	是	会话的时间戳，获取的是这个时间戳之前的会话列表，0 表示从最新开始获取

代码示例

```
NSArray *conversationList = [[RCIMClient sharedRCIMClient] getConversationList:@[@(ConversationType_GROUP)]
count:100 startTime:0];
```

获取远端会话列表

为提高性能和获取效率，SDK 的会话列表会根据消息的收发直接在本地数据库生成，不需要从服务端获取。

常见问题

Q1: 当用户卸载重新安装后发现会话列表为空或者部分会话丢失
A1: 由于会话列表是从本地数据库获取的, 是在 SDK 内部数据库存储的，所以当用户在卸载的时候会删除本地数据库，导致重新安装后会话列表为空，而会出现部分会话的原因是因为开启了 离线消息补偿 功能，具体操作是在后台购买 多设备消息同步 功能，这个功能里默认涵盖了今天(当天 0 点)的离线消息补偿，所以当用户在新设备登陆会触发离线消息补偿功能，从而获取到部分会话，造成部分会话丢失的错觉。如果想要更多天的离线消息补偿，可 提交工单 修改，最多支持 7 天，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。

删除全部会话

清除本地会话列表

更新时间:2024-08-30

开发者可调用 删除指定类型 会话接口，并传入业务中用到的所有的会话类型即可清除 SDK 在本地数据库内生成的会话列表数据。

清除远端会话列表

由于客户端的会话列表是根据消息在本地生成的，融云服务端并不存储会话列表，因此无需清除远端会话列表。

获取指定会话

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID

返回参数说明

返回值	返回类型	说明
conversation	RCConversation	会话的对象

代码示例

```
RCConversation *conversation = [[RCIMClient sharedRCIMClient] getConversation:ConversationType_GROUP
targetId:@"群组 ID"];
```

删除指定会话

删除指定会话

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] removeConversation:ConversationType_GROUP targetId:@"群组 ID"];
```

按会话类型删除

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient]
clearConversations:@[@(ConversationType_GROUP),@(ConversationType_SYSTEM),@(ConversationType_GROUP)]];
```

会话草稿信息 获取草稿

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	群组 ID

返回参数说明

返回值	返回类型	说明
draft	NSString	该会话中的草稿

代码示例

```
NSString *draft = [[RCIMClient sharedRCIMClient] getTextMessageDraft:ConversationType_GROUP targetId:@"群组
ID"];
```

保存草稿

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	群组 ID
content	NSString	是	草稿信息

返回参数说明

返回值	返回类型	说明
success	BOOL	是否保存成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] saveTextMessageDraft:ConversationType_GROUP targetId:@"群组 ID"
content:@"这个是草稿内容"];
```

删除草稿

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	群组 ID

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] clearTextMessageDraft:ConversationType_GROUP targetId:@"群组 ID"];
```

会话未读数

获取所有会话未读数

更新时间:2024-08-30

参数说明

返回参数说明

返回值	返回类型	说明
count	int	所有的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient] getTotalUnreadCount];
```

获取单个会话未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID

返回参数说明

返回值	返回类型	说明
count	int	该会话内的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient] getUnreadCount:ConversationType_GROUP targetId:@"群组 ID"];
```

按会话类型获取未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypes	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array
isContain	BOOL	是	是否包含免打扰消息的未读数

返回参数说明

返回值	返回类型	说明
count	int	该类型的会话中所有的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient] getUnreadCount:@[@(ConversationType_GROUP),@(ConversationType_GROUP)]
containBlocked:YES];
```

清除单个会话未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
timestamp	long long	是	此会话已阅读的最后一条消息的发送时间

返回参数说明

返回值	返回类型	说明
success	BOOL	是否清除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] clearMessagesUnreadStatus:ConversationType_GROUP targetId:@"群组
ID" time:0];
```

会话免打扰 设置免打扰状态

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 id
isBlocked	BOOL	是	是否屏蔽消息提醒
success	BOOL	是	设置成功的回调
error	BOOL	是	设置失败的回调

回调参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] setConversationNotificationStatus:ConversationType_GROUP
targetId:self.userId
isBlocked:swch.on
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

获取免打扰状态

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 id
success	BOOL	是	设置成功的回调
error	BOOL	是	设置失败的回调

回调参数说明

`success` 说明：

回调参数	回调类型	说明
nStatus	RCConversationNotificationStatus	会话设置的消息提醒状态

error 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] getConversationNotificationStatus:ConversationType_GROUP
targetId:self.userId
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

获取免打扰状态列表

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
array	NSArray	屏蔽消息提醒的会话RCConversation的列表

代码示例

```
NSArray *array = [[RCIMClient sharedRCIMClient] getBlockedConversationList:@[@(ConversationType_GROUP)]];
```

设置免打扰时间

参数说明

输入参数说明

参数	类型	必填	说明
startTime	NSString	是	开始消息免打扰时间，格式为 HH:MM:SS
spanMins	int	是	需要消息免打扰分钟数， 0 < spanMins < 1440 您设置的起始时间是 00：00， 结束时间为 23：59，则 spanMins 为 23 * 60 + 59 = 1339 分钟
success	BOOL	是	屏蔽成功的回调
error	BOOL	是	屏蔽失败的回调

回调参数说明

error 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] setNotificationQuietHours:@"00:00:00"
spanMins:1439
success:^{}
error:^(RCErrorCode status) {}];
```

会话置顶 设置是否置顶

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
isTop	BOOL	是	是否置顶

返回参数说明

返回值	返回类型	说明
success	BOOL	设置是否成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] setConversationToTop:ConversationType_GROUP targetId:@"群组 ID"
isTop:YES];
```

获取置顶列表

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypes	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
conversations	NSArray <RCConversation *>	置顶的会话RCConversation的列表

代码示例

```
NSArray *conversations = [[RCIMClient sharedRCIMClient]
getTopConversationList:@[@(ConversationType_GROUP),@(ConversationType_GROUP)]];
```

会话标签

更新时间:2024-08-30

⚠ 警告

该功能仅在5.0以上版本SDK中提供！

消息发送 属性描述

更新时间:2024-08-30

消息属性	描述
消息类名	各端消息名
存储属性	存储 / 不存储
离线属性	缓存 / 不缓存
推送属性	是/否
ObjectName	传输层名称，与消息类名一一对应
计数属性	计数 / 不计数
消息尺寸	128 KB
推送内容	详见各消息类送方式

存储属性

存储属性	存储分类	支持平台	详细描述
存储	客户端	Android、iOS	发送、接收该消息后，本地数据库存储 Web 端 和 小程序端因本地存储不可靠，不支持客户端消息存储，但可通过历史消息云存储服务获取历史记录
存储	云端	Android、iOS、Web	默认不在云端进行存储，需开通单群聊消息云存储服务，开通后，可在融云服务端存储 6 个月的历史消息，供客户端按需拉取
不存储	客户端	Android、iOS	发送、接收该消息后，本地数据库不存储
不存储	云端	Android、iOS、Web	无论是否开通历史消息云存储服务，该消息均不存储

计数属性

接收收到消息时，会话是否累计未读数。

计数属性	支持平台	详细描述
计数	iOS、Android、Web	会话未读消息数 + 1，该属性只影响会话列表未读数计数，App 应用角标可根据每个会话列表未读数累加获得
不计数	iOS、Android、Web	会话未读消息数不变

离线属性

接收人当前不在线时，是否进行离线缓存。

离线属性	详细描述
存储	消息进行离线缓存，默认 7 天。接收人在 7 天内上线，均可接收到该消息。超过 7 天后，消息被离线缓存淘汰。如有需要，可通过云端存储拉取到该消息
不存储	消息不进行离线缓存，所以只有接收人在线时，才可收到该消息。该消息不进行历史消息云存储、不进入云端存储（Log 日志）、不进行消息同步（消息路由）

推送属性

接收人是否接收推送，当离线属性为 存储 时，该属性生效。离线属性为 不存储 时属性无效。

由于 Web、小程序、PC 端没有推送平台，无法收到推送提醒。

推送属性	平台	推送方式	详细描述
推送	iOS、Android	APNs、华为、小米、魅族、OPPO、vivo、FCM、融云	当有离线缓存消息时，进行远程推送提醒，内容为该推送提醒显示的内容
不推送	iOS、Android	--	当有离线缓存消息时，不进行远程推送提醒

发送普通消息

通过消息体发送

接口原型：

```
- (RCMessage *)sendMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
option:(RCSendMessageOption *)option
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode nErrorCode, long messageId))errorBlock;
```

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
option	RCSendMessageOption	否	消息的相关配置
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

error 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

通过消息发送

> ⚠ 警告
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock;
```

RCMessage 添加消息配置的对象

```
/*!
消息配置，可用来设置静默消息（不进行通知提醒）
*/
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
消息推送配置，详细查看 配置消息推送
*/
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

参数说明

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息对象
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIMClient sharedRCIMClient] sendMessage:message pushContent:pushContent pushData:nil successBlock:nil
errorBlock:nil];
```

> ⚠ 警告
>
> 注意 messageid 无特殊情况应传 **0**

文本消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCTextMessage	RCMessageContent	RC:TxtMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
content	NSString	文本消息的内容

代码示例

```
RCTextMessage *textMessage = [RCTextMessage messageWithContent:@"消息内容"];

[[RCIMClient sharedRCIMClient] sendMessage:ConversationType_GROUP targetId:@"群组 ID" content:textMessage
pushContent:nil pushData:nil option:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

Emoji 消息

iOS 端发送 Emoji 消息，开发者直接使用 文本消息 发送即可。如有特殊需求需要开发者自己 自定义消息 实现。

位置消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCLocationMessage	RCMessageContent	RC:LBSMsg	存储	计数	存储	推送	[位置]

属性说明

构建参数说明：

参数	类型	说明
image	UIImage	地理位置的缩略图
location	CLLocationCoordinate2D	地理位置的二维坐标
locationName	NSString	地理位置的名称

代码示例

```
RCLocationMessage *locationMessage = [RCLocationMessage messageWithLocationImage:image location:location
locationName:locationName];

[[RCIMClient sharedRCIMClient] sendMessage:ConversationType_GROUP targetId:@"群组 ID" content:locationMessage
pushContent:nil pushData:nil option:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

发送媒体消息

通过消息体发送

接口原型：

```
- (RCMessage *)sendMediaMessage:(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
cancel:(void (^)(long messageId))cancelBlock;
```

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
content	RCMessageContent	是	消息的内容要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
messageId	long	消息的ID

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

`cancel` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

通过消息发送

> 提示
>
> 4.0.1 版本之后 SDK 新增发送消息传参为 message 的接口，并增加是否发送推送的配置
>
> 4.0.4 版本开始支持单条消息推送配置功能

接口原型：

```
- (RCMessage *)sendMediaMessage:(RCMessage *)message
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
progress:(void (^)(int progress, RCMessage *progressMessage))progressBlock
successBlock:(void (^)(RCMessage *successMessage))successBlock
errorBlock:(void (^)(RCErrorCode nErrorCode, RCMessage *errorMessage))errorBlock
cancel:(void (^)(RCMessage *cancelMessage))cancelBlock;
```

RCMessage 添加消息配置的对象

```
/*!
消息配置，可用来设置静默消息（不进行通知提醒）
*/
@property (nonatomic, strong) RCMessageConfig *messageConfig;

/*!
消息推送配置，详细查看 配置消息推送
*/
@property (nonatomic, strong) RCMessagePushConfig *messagePushConfig;
```

RCMessagePushConfig 详细说明请查看 配置消息推送

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	消息实体
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
progressMessage	RCMessage	消息实体

`success` 说明：

回调参数	回调类型	说明
successMessage	RCMessage	消息实体

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
errorMessage	RCMessage	消息实体

`cancel` 说明：

回调参数	回调类型	说明
cancelMessage	RCMessage	消息实体

代码示例：

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
message.messageConfig.disableNotification = disableNotification; // disableNotificatio 为 NO 或者 YES
message.messagePushConfig = pushConfig; // pushConfig 为构建的 RCMessagePushConfig 对象
[[RCIM sharedRCIM] sendMediaMessage:message pushContent:pushContent pushData:nil progress:nil successBlock:nil
errorBlock:nil cancel:nil];
```

> ⚠ 警告
>
> **注意 messageid 无特殊情况应传 0**

图片消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCImageMessage	RCMediaMessageContent	RC:ImgMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
imageData	NSData	图片的原始数据

代码示例

```
RCImageMessage *message = [RCImageMessage messageWithImageData:data];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:message
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

语音消息

> ⚠ 警告
>
> SDK 从 2.9.19 版本开始支持发送 RCHQVoiceMessage。

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCHQVoiceMessage	RCMediaMessageContent	RC:HQVCMsg	存储	计数	存储	推送	[语音]

构建参数说明：

参数	类型	说明
localPath	NSString	语音的本地路径
duration	long	语音消息的时长，单位为秒

示例代码

```
RCHQVoiceMessage *hqVoiceMsg = [RCHQVoiceMessage messageWithPath:path duration:duration];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:hqVoiceMsg
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

文件消息

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCFileMessage	RCMediaMessageContent	RC:FileMsg	存储	计数	存储	推送	[文件] + 文件名，如：[文件] 123.txt

构建参数说明：

参数	类型	说明
localPath	NSString	文件的本地路径

代码示例

```
RCFileMessage *fileMessage = [RCFileMessage messageWithFile:path]

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:fileMessage
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

小视频消息

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 。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCSightMessage	RCMediaMessageContent	RC:SightMsg	存储	计数	存储	推送	[小视频]

属性说明

构建参数说明：

参数	类型	说明
path	NSData	视频文件本地路径
image	UIImage	视频首帧缩略图
duration	NSUInteger	视频时长， 单位为秒

代码示例

```
RCSightMessage *sightMessage = [RCSightMessage messageWithLocalPath:url thumbnail:image duration:duration];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:sightMessage
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

GIF 消息

> ⚠ 警告
>
> 从 SDK 2.9.21 版本开始支持发送 GIF 图片消息，默认可选择发送的 GIF 图片大小为 2M。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCGIFMessage	RCMediaMessageContent	RC:GIFMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
gifImageData	NSData	GIF 图的数据
width	long	GIF 图的宽
height	long	GIF 图的高

代码示例

```
RCGIFMessage *gifMsg = [RCGIFMessage messageWithGIFImageData:gifImageData
width:gifImage.size.width
height:gifImage.size.height];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_GROUP targetId:@"群组 ID" content:gifMsg
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

发送自定义消息

除了使用 SDK 内置消息外，开发者可根据自己的业务需求来自定义消息，自定义消息的类型、消息结构需要确保多端一致，否则将出现无法互通的问题。

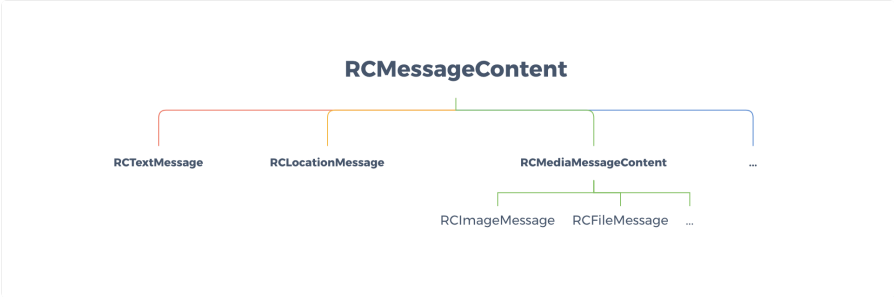
注册消息

注册消息共分为两步。

> 提示
>
> 1. 定义消息类型
> 2. 注册消息类型

1.定义消息类型

开发者可根据业务需求来选择继承消息基类 `RCMessageContent` 或者 `RCMediaMessageContent`，两者之前的区别可参考下图:



定义消息类型需要该类遵守下面三个协议

一. 编解码协议

协议原型：

```
@protocol RCMessageCoding <NSObject>
```

`RCMessageCoding` 主要有三个功能：

1. 提供消息唯一标识符
2. 消息发送时将消息中的所有信息编码为 JSON 数据传输
3. 消息接收时将 JSON 数据解码还原为消息对象。

该协议为必须遵守的协议，遵守该协议需实现下面三个方法。

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：

```
+ (NSString *)getObjectName;
```

> ⚠ 警告
>
> 1. 定义的消息类型名（objectname），需要在各个平台上保持一致，否则消息无法互通
> 2. 请勿使用 `RC:` 开头的类型名，以免和SDK默认的消息名称冲突
> 3. 所有自定义消息必须实现此协议，否则将无法正常传输和使用。

二. 存储协议

协议原型：

```
@protocol RCMessagePersistentCompatible <NSObject>
```

`RCMessageCoding` 主要有两个功能:

1. 指明此消息类型在本地和服务端是否存储
2. 指明此消息类是否计入未读消息数

该协议为必须遵守的协议，遵守该协议需实现下面方法:

消息的存储计数策略：

```
+ (RCMessagePersistent)persistentFlag;
```

RCMessagePersistent 说明:

类型	说明
MessagePersistent_NONE	表示消息不存储、不计数。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_ISCOUNTED	表示客户端收到消息后，要进行未读消息计数（未读消息数增加 1）。
MessagePersistent_ISPERSISTED	表示消息会存储到数据库。
MessagePersistent_STATUS	表示状态消息，该类型消息不存储不计数。状态类消息表示的是即时的状态，如果接收方不在线，则无法再收到该状态消息。

自定义消息持久化标识案例分析：

1. 需要存储和未读计数的消息（类似文本消息）：`MessagePersistent_ISCOUNTED | MessagePersistent_ISPERSISTED`
2. 需要确保收到，但不需要展示的消息，多为命令消息（类似撤回命令消息）：`MessagePersistent_NONE`
3. 对方在线接收，不在线就不用接收的消息（类似输入状态消息）：`MessagePersistent_STATUS`

三. 内容摘要协议

协议原型：

```
@protocol RCMessageContentView
```

`RCMessageContentView` 主要有一个功能:

1. 设置此消息在会话列表和本地通知中显示消息的摘要

该协议为非必须遵守的协议，遵守该协议需实现下面方法。

设置消息摘要：

```
- (NSString *)conversationDigest;
```

2.注册消息类型

开发者在进行完自定义消息类后，需要在 SDK 初始化之后，注册此自定义消息类。

```
- (void)registerMessageType:(Class)messageClass;
```

> ⚠ 警告
>
> 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。

发送消息

继承 RCMessageContent，请使用 发送普通消息 的接口进行发送。

继承 RCMediaMessageContent，请使用 发送媒体消息 的接口进行发送。

配置消息推送

RCMessagePushConfig 属性介绍

> ⚠ 警告
>
> 从 SDK 4.0.4 版本开始支持此功能，针对每条 Message 都可以设置此属性，详细查看以下参数说明。

参数	类型	说明
pushTitle	NSString	推送标题，如果没有设置，会使用 SDK 默认的标题显示规则
pushContent	NSString	推送内容，如果没有，则使用发送消息的 pushContent，最后则会使用 SDK 默认的标题显示规则
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
iOSConfig	RCiOSConfig	iOS 平台相关配置
androidConfig	RCAndroidConfig	Android 平台相关配置

RCiOSConfig 属性介绍

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）

RCAndroidConfig 属性介绍

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
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
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
typeVivo	NSString	VIVO 推送通道类型，开发者集成了 VIVO 推送，需要指定推送类型时，可进行设置。 目前可选值 "0"(运营消息) 和 "1"(系统消息)

Channel ID 需要由开发者进行创建，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

示例代码

```
RCMessage *message = [[RCMessage alloc] initWithType:conversationType targetId:targetId
direction:MessageDirection_SEND messageId:0 content:messageContent];
RCMessagePushConfig *pushConfig = [[RCMessagePushConfig alloc] init];
pushConfig.pushTitle = @"通知标题";
pushConfig.pushContent = @"通知内容";
pushConfig.pushData = @"通知的 pushData";
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.androidConfig.notificationId = @"Android 的通知 id";
pushConfig.androidConfig.channelIdMi = @"小米的 channelId";
pushConfig.androidConfig.channelIdHW = @"华为的 channelId";
pushConfig.androidConfig.channelIdOPPO = @"OPPO 的 channelId";
pushConfig.androidConfig.typeVivo = @"vivo 的类型";
pushConfig.forceShowDetailContent = true;
message.messagePushConfig = pushConfig;
[[RCIMClient sharedRCIMClient] sendMessage:message pushContent:nil pushData:nil successBlock:nil
errorBlock:nil];
```

发送 @ 消息

发送消息

在发送消息的时候如果需要 @ 提示，可以给 RCMessageContent 的 `mentionedInfo` 赋值。@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。

代码示例:

```
RCTextMessage *rcTextMessage = [RCTextMessage messageWithContent:@“Test”];

rcTextMessage.mentionedInfo = [[RCMentionedInfo alloc] initWithMentionedType:RC_Mentioned_Users
userIdList:mentionedUserIdList mentionedContent:nil];

[self sendMessage:rcTextMessage pushContent:nil];
```

接收消息

收到 @ 消息时，SDK 会在 RCConversation 修改 hasUnreadMentioned 的值。

> ⚠ 警告
>
> @ 消息推送会越过所有免打扰逻辑，给用户推送 Push 通知。
> 如果是自定义消息，在使用 @ 功能时，发送消息必须设置 pushContent ，否则不会发送 Push 。

发送定向消息

普通消息

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	发送消息的会话类型
targetId	NSString	是	发送消息的目标会话 ID
userIdList	NSArray	是	接收消息的用户 ID 列表
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
msg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

代码示例

```
RCMessage *msg = [[RCIMClient sharedRCIMClient] sendDirectionalMessage:ConversationType_GROUP
targetId:@"targetId"
toUserIdList:userList
content:content
pushContent:nil
pushData:nil
success:^(long messageId) {

}
error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

媒体消息

SDK iOS 端暂未开放定向发送媒体消息的接口。

消息接收

功能描述

更新时间:2024-08-30

开发者可通过此接口拦截到 SDK 接收到的消息，并进行响应的业务操作。

接口实现

SDK 通过消息监听接收消息，详细可参考 设置消息监听 文档

历史消息获取　　本地获取

更新时间:2024-08-30

开发者可通过下面接口获取 SDK 存储在本地数据库的历史消息。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
oldestMessageId	long	是	最后一条消息的 messageId，获取此消息之前的 count 条消息，第一次调用应设置为 -1。
count	int	是	需要获取的消息数量，每次最多获取 20 条

返回参数说明

返回值	返回类型	说明
history	NSArray	消息实体RCMessage对象列表

返回参数说明

代码示例

```
NSArray *history = [[RCIMClient sharedRCIMClient] getHistoryMessages:ConversationType_GROUP
targetId:@"群组 ID"
oldestMessageId:lastMessageID
count:count];
```

远端获取

开发者可通过下面接口获取 SDK 存储在服务端的历史消息。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。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	目标会话 ID

参数	类型	必填	说明
option	RCRemoteHistoryMsgOption	是	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

回调参数说明

`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

代码示例

```
RCRemoteHistoryMsgOption *option = [RCRemoteHistoryMsgOption new];
option.recordTime = recordTime;
option.count = 10;
option.order = RCRemoteHistoryOrderDesc;

[[RCIMClient sharedRCIMClient] getRemoteHistoryMessages:ConversationType_GROUP targetId:@"群组 ID" option:option
success:^(NSArray *messages, BOOL isRemaining) {

} error:^(RCErrorCode status) {

}];
```

消息回执 功能描述

更新时间:2024-08-30

开发者可使用此功能实现消息已读未读功能的展示。

当 A 给 B 发送了一条消息，B 在未阅读之前 A 用户显示未读，当 B 用户阅读并调用发送回执接口之后，A 用户可在监听回执中收到通知，此时可根据对应的数据内容将发送的消息显示为已读。

此功能目前仅在 GROUP 类型的会话中开放。用户可以对自己发送的消息发起阅读回执请求，发起后，可以看到有多少人阅读过这条消息。

发起方

发送阅读回执请求

提示

"提示"

对于需要阅读之后收到阅读回执的消息，可以调用这个接口来发送阅读回执请求。

参数	类型	说明
message	RCMessage	要求阅读回执的消息
successBlock	void	发送请求成功的回调
errorBlock	void	发送请求失败的回调 [nErrorCode: 失败的错误码]

```
- (void)sendReadReceiptRequest:(RCMessage *)message
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode nErrorCode))errorBlock;
```

接收阅读回执响应

提示

收到消息阅读回执响应的回调函数（收到此响应后，可以从消息中获取获取到阅读回执消息的 `messageUId` 和 阅读消息用户的列表，消息的发送方即可直接知道哪些人已经阅读。）

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	目标会话 Id
messageUId	NSString	阅读回执消息的 messageUId

参数	类型	说明
userIdList	NSMutableDictionary	发送阅读回执响应的用户列表

```
- (void)onMessageReceiptResponse:(RCConversationType)conversationType
targetId:(NSString *)targetId
messageUId:(NSString *)messageUId
readerList:(NSMutableDictionary *)userIdList;
```

接收方

接收阅读回执请求

> 提示
>
> 收到消息已读回执请求的回调函数（收到此请求后，如果用户阅读了对应的消息，需要调用 `sendMessageReadReceiptResponse` 接口发送已读响应。）

参数	类型	说明
conversationType	RCConversationType	会话类型
targetId	NSString	目标会话 Id
messageUId	NSString	收到阅读回执请求消息的 messageUId

```
- (void)onMessageReceiptRequest:(RCConversationType)conversationType
targetId:(NSString *)targetId
messageUId:(NSString *)messageUId;
```

发送阅读回执响应

> 提示
>
> 当收到阅读回执请求之后，如果用户阅读了对应的消息，可以调用此接口来发送消息阅读回执响应。

| 参数 | 类型 | 说明 | | ---------- | --------- | ---------- | | conversationType | RCConversationType | 会话类型 | targetId | NSString | 目标会话 Id | | messageList | NSArray | 已经阅读了的消息列表 | | successBlock | void | 发送成功的回调 | | errorBlock | void | 发送失败的回调 [nErrorCode: 失败的错误码] |

```
- (void)sendReadReceiptResponse:(RCConversationType)conversationType
targetId:(NSString *)targetId
messageList:(NSArray<RCMessage *> *)messageList
success:(void (^)(void))successBlock
error:(void (^)(RCErrorCode nErrorCode))errorBlock;
```

消息撤回 消息撤回

更新时间:2024-08-30

消息发送方可通过下面方法撤回已发送成功的消息。

参数说明

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	需要撤回的消息
pushContent	NSString	否	push 显示的内容
successBlock	Block	-	撤回成功的回调
errorBlock	Block	-	撤回失败的回调

返回参数说明

`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	撤回失败错误码

代码示例

```
[[RCIMClient sharedRCIMClient] recallMessage:msg pushContent:nil success:^(long messageId) {

} error:^(RCErrorCode errorcode) {

}];
```

监听撤回

消息接收方可通过下面方法监听到被撤回的消息。

实现此功能需要开发者遵守 `RCIMClientReceiveMessageDelegate` 协议。

参数说明

代理对象参数说明：

传入参数	类型	必填	说明
delegate	id	是	IMLib 消息接收监听器
userData	id	否	用户自定义的监听器 Key 值，可以为 nil

协议方法参数说明：

参数	类型	说明
messageId	long	被撤回的消息ID

代码示例

1. 设置代理对象:

```
[[RCIMClient sharedRCIMClient] setReceiveMessageDelegate:self object:nil];
```

2. 实现协议方法:

```
- (void)onMessageRecalled:(long)messageId{

}
```

消息转发

更新时间:2024-08-30

消息转发调用发送消息接口发送即可，调用 SDK 发送消息接口需考虑 SDK 每秒 5 条消息的限制

超过每秒 5 条限制可使用 Server API 进行发送

消息删除 本地删除

更新时间:2024-08-30

开发者可通过下面接口对本地数据库的消息进行删除。

通过消息 ID 删除

输入参数说明：

参数	类型	必填	说明
messageIds	NSArray<NSNumber *>	是	消息ID的列表，元素需要为 NSNumber 类型

返回参数说明：

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例：

```
BOOL success = [[RCIMClient sharedRCIMClient] deleteMessages: @[@(message.messageId)]];
```

通过时间戳删除

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
recordTime	long long	是	清除消息时间戳，传入的时间戳必须小于等于当前会话最后一条消息的 sentTime 传 **0** 表示清除所有消息， 传入其他值表示清除小于等于 recordTime 的消息
clearRemote	BOOL	是	是否同时删除服务端消息
successBlock	Block	-	获取成功的回调
errorBlock	Block	-	获取失败的回调

回调参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	撤回失败错误码

代码示例：

```
[[RCIMClient sharedRCIMClient] clearHistoryMessages:ConversationType_GROUP
targetId:@"群组 ID"
recordTime:recordTime
clearRemote:NO
success:^{}
error:^(RCErrorCode status) {}];
```

通过会话删除

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID

返回参数说明：

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例：

```
BOOL success = [[RCIMClient sharedRCIMClient] clearMessages:ConversationType_GROUP targetId:@"群组 ID"];
```

远端删除

开发者可通过下面接口对远端的消息进行删除。

通过时间戳删除

通过时间戳删除该时间之前的服务端历史消息，删除后不可恢复。

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
recordTime	long long	是	清除消息时间戳，传入的时间戳必须小于等于当前会话最后一条消息的 sentTime 传 **0** 表示清除所有消息， 传入其他值表示清除小于等于 recordTime 的消息
successBlock	Block	-	删除成功的回调
errorBlock	Block	-	删除失败的回调

回调参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	删除失败错误码

代码示例：

```
[[RCIMClient sharedRCIMClient]clearRemoteHistoryMessages:ConversationType_GROUP
targetId:@"群组 ID"
recordTime:sentTime
success:^{}
error:^(RCErrorCode status) {}];
```

通过消息删除

仅删除服务端指定一条或多条历史消息。

输入参数说明：

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
messages	NSArray<RCMessage *>	是	将被删除的消息列表
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

回调参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	删除失败错误码

代码示例：

```
[[RCIMClient sharedRCIMClient] deleteRemoteMessage:ConversationType_GROUP
targetId:@"群组 ID"
messages:messages success:^{}
error:^(RCErrorCode status) {}];
```

消息搜索

通过关键字搜索

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	会话ID
keyword	NSString	是	关键字
count	int	是	最大的查询数量
startTime	long long	是	查询 startTime 之前的消息（传 0 表示不限时间）

返回参数说明

返回值	返回类型	说明
searchResult	NSArray<RCMessage *>	匹配的消息列表

代码示例

```
NSArray *searchResult = [[RCIMClient sharedRCIMClient] searchMessages:ConversationType_GROUP
targetId:@"群组 ID"
keyword:searchText
count:50
startTime:0];
```

通过用户 ID 搜索

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	会话ID
userId	NSString	是	搜索用户ID
count	int	是	最大的查询数量
startTime	long long	是	查询 startTime 之前的消息（传 0 表示不限时间）

返回参数说明

返回值	返回类型	说明
searchResult	NSArray<RCMessage *>	匹配的消息列表

代码示例

```
NSArray *searchResult = [[RCIMClient sharedRCIMClient] searchMessages:ConversationType_GROUP
targetId:@"群组 ID"
userId:@"userId"
count:50
startTime:0];
```

消息插入

插入发送消息

更新时间:2024-08-30

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。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
sentStatus	RCSentStatus	是	发送状态
content	RCMessageContent	是	消息的内容
sentTime	long long	是	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）

返回参数说明

返回值	返回类型	说明
message	RCMessage	插入的消息实体

代码示例

> ⚠ 警告
>
> 此方法如果 sentTime 有问题会影响消息排序，慎用！

```
RCMessage *message = [[RCIMClient sharedRCIMClient] insertOutgoingMessage:ConversationType_GROUP
targetId:@"群组 ID"
sentStatus:SentStatus_SENT
content:content
sentTime:sentTime];
```

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。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，群聊传入 ConversationType_GROUP
targetId	NSString	是	群组 ID

参数	类型	必填	说明
senderUserId	NSString	是	发送者ID
sentStatus	RCReceivedStatus	是	接收状态
content	RCMessageContent	是	消息的内容
sentTime	long long	是	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）

返回参数说明

返回值	返回类型	说明
message	RCMessage	插入的消息实体

代码示例

> ⚠ 警告
>
> 此方法如果 sentTime 有问题会影响消息排序，慎用！

```
RCMessage *tempMessage = [[RCIMClient sharedRCIMClient] insertIncomingMessage:ConversationType_GROUP
targetId:@"群组 ID"
senderUserId:@"senderUserId"
receivedStatus:ReceivedStatus_READ
content:content];
```

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。

- 仅支持单聊、群聊会话类型。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。

设置消息是否支持扩展

发送消息时，该条消息设置了可扩展属性后，才能设置扩展信息，消息可扩展属性相关功能如下：

```
/*!
消息是否可以包含扩展信息

@discussion 该属性在消息发送时确定，发送之后不能再做修改
@discussion 扩展信息只支持单聊和群组，其它会话类型不能设置扩展信息
*/
@property (nonatomic, assign) BOOL canIncludeExpansion;

/*!
消息扩展信息列表

@discussion 扩展信息只支持单聊和群组，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 64 ，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;
```

更新扩展信息

- 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。
- 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。

输入参数说明：

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。 • Value 最大 64 个字符。
messageUId	NSString	消息 messageUId
successBlock	Block	成功的回调
errorBlock	Block	失败的回调

返回参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	失败的错误码

```
[[RCIMClient sharedRCIMClient] updateMessageExpansion:expansionDic messageUId:messageUId success:^{

} error:^(RCErrorCode status) {

}];
```

删除扩展信息

输入参数说明：

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

返回参数说明：

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	失败的错误码

```
[[RCIMClient sharedRCIMClient] removeMessageExpansionForKey:keyArray messageUId:messageUId success:^{

} error:^(RCErrorCode status) {

}];
```

设置回调

实现此功能需要开发者遵守 `RCMessageExpansionDelegate` 协议

代理对象参数说明：

参数	类型	必填	说明
delegate	id	是	IMLib 消息扩展监听器

```
[RCIMClient sharedRCIMClient].messageExpansionDelegate = self;
```

协议方法参数说明：

参数	类型	说明
expansionDic	NSDictionary	消息扩展信息中更新的键值对
message	RCMessage	消息

```
- (void)messageExpansionDidUpdate:(NSDictionary<NSString *,NSString *> *)expansionDic message:(RCMessage *)message{

}
```

参数	类型	说明
keyArray	NSArray	消息扩展信息中删除的键值对 key 列表
message	RCMessage	消息

```
- (void)messageExpansionDidRemove:(NSArray<NSString *> *)keyArray message:(RCMessage *)message{

}
```

聊天室概述

更新时间:2024-08-30

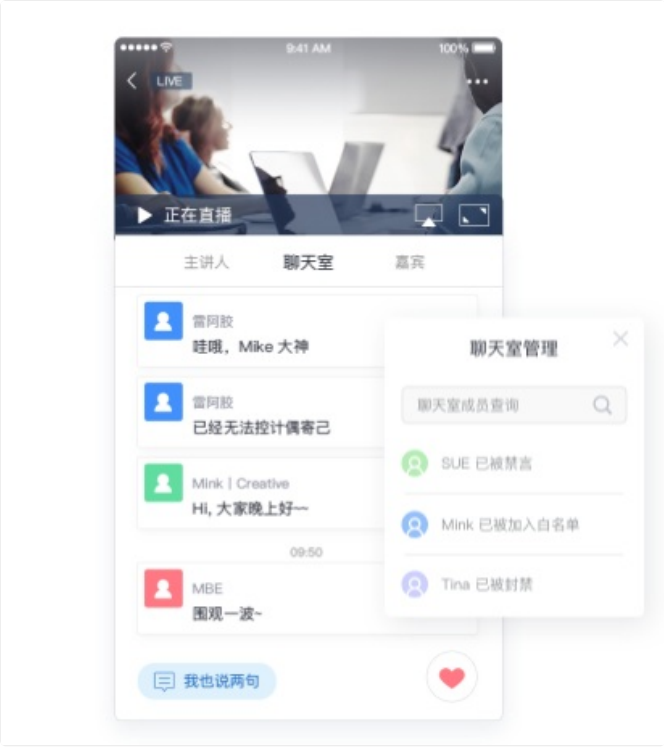
聊天室（Chatroom）提供了一种不设用户上限，支持高并发消息处理的业务形态，可用于直播、社区、游戏、广场交友、兴趣讨论等场景。聊天室业务要点如下：

- 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

> 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	加入聊天室时，如果聊天室未创建，则会创建聊天室再加入。如果已创建，则直接加入。	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。 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	[低级别消息/白名单]
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

功能分类	功能描述	客户端 API	融云服务端 API
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。 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	[低级别消息/白名单]
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。 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	[低级别消息/白名单]
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
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。 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

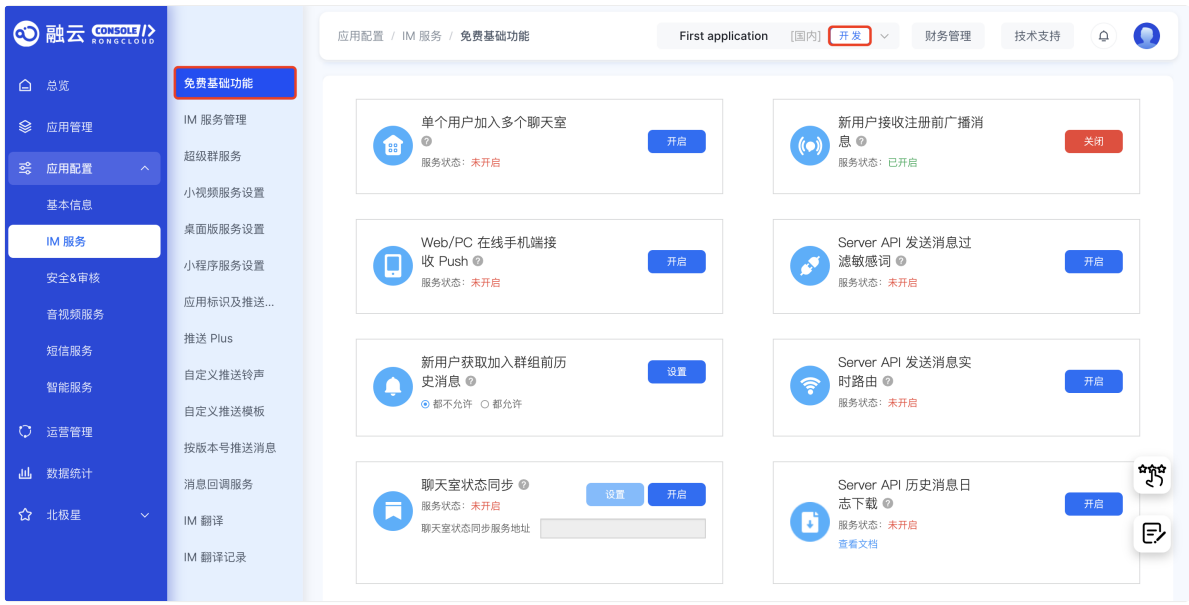
以下是聊天室业务提供的免费基础功能：

- **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

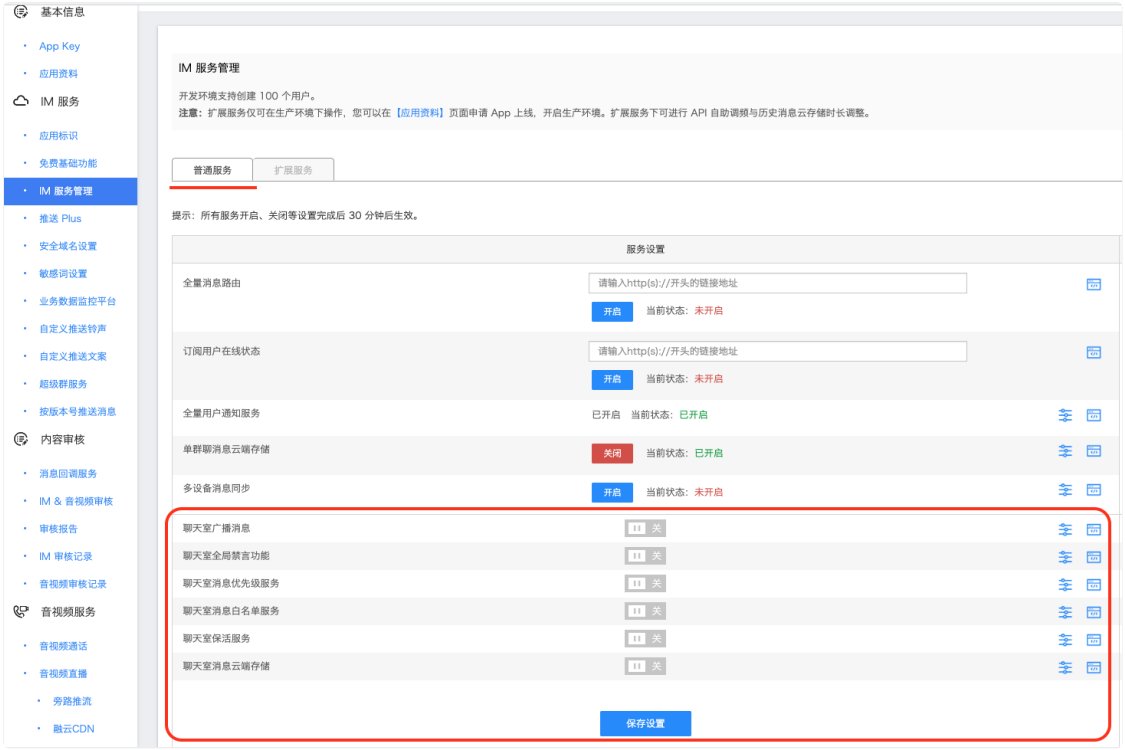
下图显示了控制台 IM 服务管理 页面与聊天室业务相关的普通服务配置。

> 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。

加入

更新时间:2024-08-30

默认同一用户不能同时加入多个聊天室，加入新的聊天室后，会自动退出之前的聊天室。

如需支持单个用户加入多个聊天室，请在控制台免费基础功能页面打开该配置。

加入聊天室

调用此接口时如果聊天室不存在则会创建。

参数说明

输入参数说明

参数	类型	必填	说明
targetId	NSString	是	聊天室的 chatroomId
messageCount	int	是	进入聊天室时获取历史消息的数量 **-1** 表示不获取任何历史消息 **0** 表示不特殊设置而使用 SDK 默认的设置，**默认为获取 10 条** 传入其他值为具体获取的消息数量，**最大值为 50**
successBlock	Block	-	加入聊天室成功的回调
errorBlock	Block	-	加入聊天室失败的回调

返回参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	加入聊天室失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] joinChatRoom:roomId
messageCount:20
success:^{}
error:^(RCErrorCode status) {}];
```

加入存在的聊天室

参数说明

输入参数说明

参数	类型	必填	说明
targetId	NSString	是	聊天室的 chatroomId

参数	类型	必填	说明
messageCount	int	是	进入聊天室时获取历史消息的数量 **-1** 表示不获取任何历史消息 **0** 表示不特殊设置而使用 SDK 默认的设置，默认为获取 **10** 条 传入其他值为具体获取的消息数量，最大值为 **50**
successBlock	Block	-	加入聊天室成功的回调
errorBlock	Block	-	加入聊天室失败的回调

返回参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	加入聊天室失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] joinExistChatRoom:roomId
messageCount:20
success:^{}
error:^(RCErrorCode status) {}];
```

退出

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。

> **注意：**
>
> - 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。

参数说明

输入参数说明

参数	类型	必填	说明
targetId	NSString	是	聊天室的 chatroomId
successBlock	Block	-	退出聊天室成功的回调
errorBlock	Block	-	退出聊天室失败的回调

返回参数说明

`error` 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

代码示例

```
[[RCIMClient sharedRCIMClient] quitChatRoom:roomId
success:^{}
error:^(RCErrorCode status) {}];
```

查询聊天室信息 功能描述

更新时间:2024-08-30

因为聊天室一般成员数量巨大，权衡效率和用户体验，SDK 会根据传入的查询方式返回聊天室信息。

参数说明

输入参数说明

参数	类型	必填	说明
targetId	NSString	是	聊天室 chatroomId
count	int	是	传入 **0** 获取到的聊天室信息将或仅包含成员总数，不包含具体的成员列表 传入其他大于 **0** 的值返回聊天室信息，结果仅包含包含不多于 **20** 人的成员信息和当前成员总数。**最大值为 20**
order	RCChatRoomMemberOrder	是	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）
successBlock	Block	-	获取成功的回调
errorBlock	Block	-	获取失败的回调

返回参数说明

`success` 说明：

回调参数	回调类型	说明
chatRoomInfo	RCChatRoomInfo	聊天室信息

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] getChatRoomInfo:@"roomId" count:0 order:RC_ChatRoom_Member_Asc
success:^(RCChatRoomInfo *chatRoomInfo) {

} error:^(RCErrorCode status) {

}];
```

设置属性

更新时间:2024-08-30

聊天室属性（KV）管理接口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。

功能局限

- 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）。

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。

如果配置了服务端回调 URL，融云服务端会将应用下的聊天室属性变化（设置，删除，全部删除等操作）同步到指定的回调地址。详见服务端文档聊天室属性同步（KV）。

设置属性

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。

参数说明

输入参数说明

参数	类型	必填	说明
chatroomId	NSString	是	聊天室的 chatroomId
key	NSString	是	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	是	聊天室属性对应的值，最大长度 4096 个字符
sendNotification	BOOL	是	是否需要发送通知，
autoDelete	BOOL	是	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	是	通知的自定义字段，RC:chrmKVNotiMsg 通知消息中会包含此字段，最大长度 2048 个字符
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

返回参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] setChatRoomEntry:chatroomId
key:key
value:value
sendNotification:isNotice
autoDelete:isDelete
notificationExtra:extra
success:^{}
error:^(RCErrorCode nErrorCode) {}];
```

强制设置属性

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。

参数说明

输入参数说明

参数	类型	必填	说明
chatroomId	NSString	是	聊天室的 chatroomId
key	NSString	是	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	是	聊天室属性对应的值，最大长度 4096 个字符
sendNotification	BOOL	是	是否需要发送通知，
autoDelete	BOOL	是	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	是	通知的自定义字段，RC:chrmKVNotiMsg 通知消息中会包含此字段，最大长度 2048 个字符
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

返回参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] forceSetChatRoomEntry:chatroomId
key:key
value:value
sendNotification:isNotice
autoDelete:isDelete
notificationExtra:extra
success:^{}
error:^(RCErrorCode nErrorCode) {}];
```

删除属性 删除属性

更新时间:2024-08-30

删除聊天室自定义属性，只有自己设置的属性可以被删除。

参数说明

输入参数说明

参数	类型	必填	说明
chatroomId	NSString	是	聊天室的 chatroomId
key	NSString	是	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	是否需要发送通知
notificationExtra	NSString	是	通知的自定义字段，RC:chrmKVNotiMsg 通知消息中会包含此字段，最大长度 2048 个字符
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

返回参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

> ⚠ 警告
>
> 只有自己设置的属性可以被删除。

```
[[RCIMClient sharedRCIMClient] removeChatRoomEntry:chatroomId
key:key
sendNotification:isNotice
notificationExtra:extra
success:^{}
error:^(RCErrorCode nErrorCode) {}];
```

强制删除属性

强制删除聊天室自定义属性。

参数说明

输入参数说明

参数	类型	必填	说明
chatroomId	NSString	是	聊天室的 chatroomId
key	NSString	是	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	是否需要发送通知
notificationExtra	NSString	是	通知的自定义字段，RC:chrmKVNotiMsg 通知消息中会包含此字段，最大长度 2048 个字符
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

返回参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] forceRemoveChatRoomEntry:chatroomId
key:key
sendNotification:isNotice
notificationExtra:extra
success:^{}
error:^(RCErrorCode nErrorCode) {}];
```

获取属性　　获取单个属性

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
chatroomId	NSString	是	聊天室的 chatroomId
key	NSString	是	聊天室属性名称，
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

返回参数说明

`success` 说明：

回调参数	回调类型	说明
entry	NSDictionary	属性内容

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] getChatRoomEntry:chatroomId
key:key
success:^(NSDictionary *entry) {

}
error:^(RCErrorCode nErrorCode) {

}];
```

获取所有属性

参数说明

输入参数说明

参数	类型	必填	说明
chatroomId	NSString	是	聊天室的 chatroomId
successBlock	Block	-	成功的回调
errorBlock	Block	-	失败的回调

返回参数说明

`success` 说明：

回调参数	回调类型	说明
entry	NSDictionary	属性内容

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] getAllChatRoomEntries:self.roomId
success:^(NSDictionary *entry) {

}
error:^(RCErrorCode nErrorCode) {

}];
```

设置回调

更新时间:2024-08-30

实现此功能需要开发者遵守 RCChatRoomKVStatusChangeDelegate 协议

参数说明

代理对象参数说明：

参数	类型	必填	说明
delegate	id	是	IMLib 消息接收监听器

协议方法参数说明：

参数	类型	说明
roomId	NSString	聊天室 ID
entry	NSDictionary	修改或者删除的 KV 字典

代码示例

```
// 1. 设置代理对象
[[RCIMClient sharedRCIMClient] setRCChatRoomKVStatusChangeDelegate:self];

- (void)chatRoomKVDidSync:(NSString *)roomId {
}

- (void)chatRoomKVDidUpdate:(NSString *)roomId entry:(NSDictionary<NSString *,NSString *> *)entry {
}

- (void)chatRoomKVDidRemove:(NSString *)roomId entry:(NSDictionary<NSString *,NSString *> *)entry {
}
```

消息发送 属性描述

更新时间:2024-08-30

消息属性	描述
消息类名	各端消息名
存储属性	存储 / 不存储
离线属性	缓存 / 不缓存
推送属性	是/否
ObjectName	传输层名称，与消息类名一一对应
计数属性	计数 / 不计数
消息尺寸	128 KB
推送内容	详见各消息类送方式

存储属性

存储属性	存储分类	支持平台	详细描述
存储	客户端	Android、iOS	发送、接收该消息后，本地数据库存储 Web 端 和 小程序端因本地存储不可靠，不支持客户端消息存储，但可通过历史消息云存储服务获取历史记录
存储	云端	Android、iOS、Web	默认不在云端进行存储，需开通单群聊消息云存储服务，开通后，可在融云服务端存储 6 个月的历史消息，供客户端按需拉取
不存储	客户端	Android、iOS	发送、接收该消息后，本地数据库不存储
不存储	云端	Android、iOS、Web	无论是否开通历史消息云存储服务，该消息均不存储

计数属性

接收收到消息时，会话是否累计未读数。

计数属性	支持平台	详细描述
计数	iOS、Android、Web	会话未读消息数 + 1，该属性只影响会话列表未读数计数，App 应用角标可根据每个会话列表未读数累加获得
不计数	iOS、Android、Web	会话未读消息数不变

离线属性

接收人当前不在线时，是否进行离线缓存。

离线属性	详细描述
存储	消息进行离线缓存，默认 7 天。接收人在 7 天内上线，均可接收到该消息。超过 7 天后，消息被离线缓存淘汰。如有需要，可通过云端存储拉取到该消息
不存储	消息不进行离线缓存，所以只有接收人在线时，才可收到该消息。该消息不进行历史消息云存储、不进入云端存储（Log 日志）、不进行消息同步（消息路由）

推送属性

接收人是否接收推送，当离线属性为 存储 时，该属性生效。离线属性为 不存储 时属性无效。

由于 Web、小程序、PC 端没有推送平台，无法收到推送提醒。

推送属性	平台	推送方式	详细描述
推送	iOS、Android	APNs、华为、小米、魅族、OPPO、vivo、FCM、融云	当有离线缓存消息时，进行远程推送提醒，内容为该推送提醒显示的内容
不推送	iOS、Android	--	当有离线缓存消息时，不进行远程推送提醒

发送普通消息

发送消息

接口原型：

```
- (RCMessage *)sendMessage:(RCConversationType)conversationType
targetId:(NSString *)targetId
content:(RCMessageContent *)content
pushContent:(NSString *)pushContent
pushData:(NSString *)pushData
option:(RCSendMessageOption *)option
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode nErrorCode, long messageId))errorBlock;
```

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，聊天室传入 ConversationType_CHATROOM
targetId	NSString	是	聊天室 ID
content	RCMessageContent	是	消息的内容
pushContent	NSString	否	接收方离线时需要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
option	RCSendMessageOption	否	消息的相关配置
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调

返回参数说明

返回值	返回类型	说明
textMsg	RCMessage	发送的消息实体

回调参数说明

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

error 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

文本消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCTextMessage	RCMessageContent	RC:TxtMsg	存储	计数	存储	推送	消息内容

属性说明

构建参数说明：

参数	类型	说明
content	NSString	文本消息的内容

代码示例

```
RCTextMessage *textMessage = [RCTextMessage messageWithContent:@"消息内容"];

[[RCIMClient sharedRCIMClient] sendMessage:ConversationType_CHATROOM targetId:@"聊天室 ID" content:textMessage
pushContent:nil pushData:nil option:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

Emoji 消息

iOS 端发送 Emoji 消息，开发者直接使用 文本消息 发送即可。如有特殊需求需要开发者自己 自定义消息 实现。

位置消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCLocationMessage	RCMessageContent	RC:LBSMsg	存储	计数	存储	推送	[位置]

属性说明

构建参数说明：

参数	类型	说明
image	UIImage	地理位置的缩略图
location	CLLocationCoordinate2D	地理位置的二维坐标
locationName	NSString	地理位置的名称

代码示例

```
RCLocationMessage *locationMessage = [RCLocationMessage messageWithLocationImage:image location:location
locationName:locationName];

[[RCIMClient sharedRCIMClient] sendMessage:ConversationType_CHATROOM targetId:@"聊天室 ID"
content:locationMessage pushContent:nil pushData:nil option:nil success:^(long messageId) {

} error:^(RCErrorCode nErrorCode, long messageId) {

}];
```

发送媒体消息

发送媒体消息

接口原型：

```
- (RCMessage *)sendMediaMessage:(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
cancel:(void (^)(long messageId))cancelBlock;
```

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，聊天室传入 ConversationType_CHATROOM
targetId	NSString	是	聊天室 ID
content	RCMessageContent	是	消息的内容要显示的远程推送内容
pushData	NSString	否	接收方离线时需要在远程推送中携带的非显示数据
progress	Block	-	消息发送进度更新的回调
success	Block	-	消息发送成功的回调
error	Block	-	消息发送失败的回调
cancel	Block	-	用户取消了消息发送的回调

返回参数说明

返回值	返回类型	说明
imgMsg	RCMessage	发送的消息实体

回调参数说明

`progress` 说明：

回调参数	回调类型	说明
progress	int	当前的发送进度, 0 <= progress <= 100
messageId	long	消息的ID

`success` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

`error` 说明：

回调参数	回调类型	说明
nErrorCode	RCErrorCode	发送失败的错误码
messageId	long	消息的ID

`cancel` 说明：

回调参数	回调类型	说明
messageId	long	消息的ID

图片消息

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCImageMessage	RCMediaMessageContent	RC:ImgMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
imageData	NSData	图片的原始数据

代码示例

```
RCImageMessage *message = [RCImageMessage messageWithImageData:data];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_CHATROOM targetId:@"聊天室 ID" content:message
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

语音消息

> ⚠ 警告
>
> SDK 从 2.9.19 版本开始支持发送 `RCHQVoiceMessage`。

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCHQVoiceMessage	RCMediaMessageContent	RC:HQVCMsg	存储	计数	存储	推送	[语音]

构建参数说明：

参数	类型	说明
localPath	NSString	语音的本地路径
duration	long	语音消息的时长，单位为秒

示例代码

```
RCHQVoiceMessage *hqVoiceMsg = [RCHQVoiceMessage messageWithPath:path duration:duration];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_CHATROOM targetId:@"聊天室 ID"
content:hqVoiceMsg pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

文件消息

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCFileMessage	RCMediaMessageContent	RC:FileMsg	存储	计数	存储	推送	[文件] + 文件名，如：[文件] 123.txt

构建参数说明：

参数	类型	说明
localPath	NSString	文件的本地路径

代码示例

```
RCFileMessage *fileMessage = [RCFileMessage messageWithFile:path]

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_CHATROOM targetId:@"聊天室 ID"
content:fileMessage pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

小视频消息

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 。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCSightMessage	RCMediaMessageContent	RC:SightMsg	存储	计数	存储	推送	[小视频]

属性说明

构建参数说明：

参数	类型	说明
path	NSData	视频文件本地路径
image	UIImage	视频首帧缩略图
duration	NSUInteger	视频时长， 单位为秒

代码示例

```
RCSightMessage *sightMessage = [RCSightMessage messageWithLocalPath:url thumbnail:image duration:duration];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_CHATROOM targetId:@"聊天室 ID"
content:sightMessage pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

GIF 消息

> ⚠ 警告
>
> 从 SDK 2.9.21 版本开始支持发送 GIF 图片消息，默认可选择发送的 GIF 图片大小为 2M。

消息说明

消息类名	父类名称	ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RCGIFMessage	RCMediaMessageContent	RC:GIFMsg	存储	计数	存储	推送	[图片]

属性说明

构建参数说明：

参数	类型	说明
gifImageData	NSData	GIF 图的数据
width	long	GIF 图的宽
height	long	GIF 图的高

代码示例

```
RCGIFMessage *gifMsg = [RCGIFMessage messageWithGIFImageData:gifImageData
width:gifImage.size.width
height:gifImage.size.height];

[[RCIMClient sharedRCIMClient] sendMediaMessage:ConversationType_CHATROOM targetId:@"聊天室 ID" content:gifMsg
pushContent:nil pushData:nil progress:^(int progress, long messageId) {

} success:^(long messageId) {

} error:^(RCErrorCode errorCode, long messageId) {

} cancel:^(long messageId) {

}];
```

发送自定义消息

除了使用 SDK 内置消息外，开发者可根据自己的业务需求来自定义消息，自定义消息的类型、消息结构需要确保多端一致，否则将出现无法互通的问题。

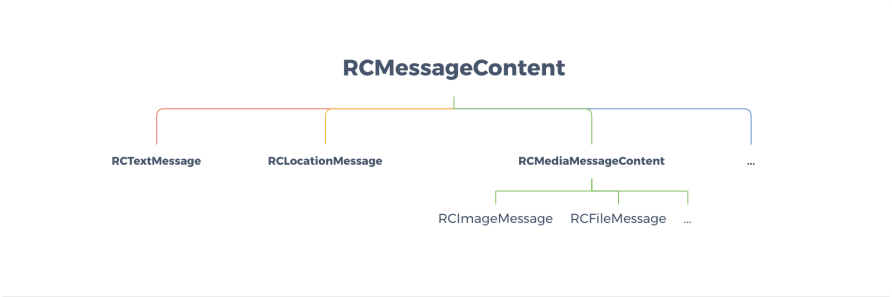
注册消息

注册消息共分为两步。

> 提示
>
> 1. 定义消息类型
> 2. 注册消息类型

1.定义消息类型

开发者可根据业务需求来选择继承消息基类 `RCMessageContent` 或者 `RCMediaMessageContent`，两者之前的区别可参考下图:



定义消息类型需要该类遵守下面三个协议

一. 编解码协议

协议原型：

```
@protocol RCMessageCoding <NSObject>
```

`RCMessageCoding` 主要有三个功能：

1. 提供消息唯一标识符
2. 消息发送时将消息中的所有信息编码为 JSON 数据传输
3. 消息接收时将 JSON 数据解码还原为消息对象。

该协议为必须遵守的协议，遵守该协议需实现下面三个方法。

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：

```
+ (NSString *)getObjectName;
```

> ⚠ 警告
>
> 1. 定义的消息类型名（objectname），需要在各个平台上保持一致，否则消息无法互通
> 2. 请勿使用 `RC:` 开头的类型名，以免和SDK默认的消息名称冲突
> 3. 所有自定义消息必须实现此协议，否则将无法正常传输和使用。

二. 存储协议

协议原型：

```
@protocol RCMessagePersistentCompatible <NSObject>
```

`RCMessageCoding` 主要有两个功能:

1. 指明此消息类型在本地和服务端是否存储
2. 指明此消息类是否计入未读消息数

该协议为必须遵守的协议，遵守该协议需实现下面方法:

消息的存储计数策略：

```
+ (RCMessagePersistent)persistentFlag;
```

RCMessagePersistent 说明:

类型	说明
MessagePersistent_NONE	表示消息不存储、不计数。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_ISCOUNTED	表示客户端收到消息后，要进行未读消息计数（未读消息数增加 1）。
MessagePersistent_ISPERSISTED	表示消息会存储到数据库。
MessagePersistent_STATUS	表示状态消息，该类型消息不存储不计数。状态类消息表示的是即时的状态，如果接收方不在线，则无法再收到该状态消息。

自定义消息持久化标识案例分析：

1. 需要存储和未读计数的消息（类似文本消息）：`MessagePersistent_ISCOUNTED | MessagePersistent_ISPERSISTED`
2. 需要确保收到，但不需要展示的消息，多为命令消息（类似撤回命令消息）：`MessagePersistent_NONE`
3. 对方在线接收，不在线就不用接收的消息（类似输入状态消息）：`MessagePersistent_STATUS`

三. 内容摘要协议

协议原型：

```
@protocol RCMessageContentView
```

`RCMessageContentView` 主要有一个功能:

1. 设置此消息在会话列表和本地通知中显示消息的摘要

该协议为非必须遵守的协议，遵守该协议需实现下面方法。

设置消息摘要：

```
- (NSString *)conversationDigest;
```

2.注册消息类型

开发者在进行完自定义消息类后，需要在 SDK 初始化之后，注册此自定义消息类。

```
- (void)registerMessageType:(Class)messageClass;
```

⚠ 警告

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。

发送消息

继承 RCMessageContent，请使用 发送普通消息 的接口进行发送。

继承 RCMediaMessageContent，请使用 发送媒体消息 的接口进行发送。

消息接收

功能描述

更新时间:2024-08-30

开发者可通过此接口拦截到 SDK 接收到的消息，并进行响应的业务操作。

实现方法

SDK 通过消息监听接收消息，详细可参考 设置消息监听 文档

消息撤回 消息撤回

更新时间:2024-08-30

消息发送方可通过下面方法撤回已发送成功的消息。

参数说明

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	需要撤回的消息
pushContent	NSString	否	push 显示的内容
successBlock	Block	-	撤回成功的回调
errorBlock	Block	-	撤回失败的回调

返回参数说明

`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	撤回失败错误码

代码示例

```
[[RCIMClient sharedRCIMClient] recallMessage:msg pushContent:nil success:^(long messageId) {

} error:^(RCErrorCode errorcode) {

}];
```

监听撤回

消息接收方可通过下面方法监听到被撤回的消息。

实现此功能需要开发者遵守 `RCIMClientReceiveMessageDelegate` 协议。

参数说明

代理对象参数说明：

传入参数	类型	必填	说明
delegate	id	是	IMLib 消息接收监听器
userData	id	否	用户自定义的监听器 Key 值，可以为 nil

协议方法参数说明：

参数	类型	说明
messageId	long	被撤回的消息ID

代码示例

1. 设置代理对象:

```
[[RCIMClient sharedRCIMClient] setReceiveMessageDelegate:self object:nil];
```

2. 实现协议方法:

```
- (void)onMessageRecalled:(long)messageId{

}
```

消息发送方撤回消息

参数说明

输入参数说明

参数	类型	必填	说明
message	RCMessage	是	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

返回参数说明

`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	撤回失败错误码

代码示例

```
[[RCIMClient sharedRCIMClient] recallMessage:msg success:^(long messageId) {

} error:^(RCErrorCode errorcode) {

}];
```

消息接收方监听撤回

实现此功能需要开发者遵守 `RCIMClientReceiveMessageDelegate` 协议。

参数说明

代理对象参数说明：

传入参数	类型	必填	说明
delegate	id	是	IMLib 消息接收监听器
userData	id	否	用户自定义的监听器 Key 值，可以为 nil

协议方法参数说明：

参数	类型	说明
messageId	long	被撤回的消息ID

代码示例

1. 设置代理对象:

```
[[RCIMClient sharedRCIMClient] setReceiveMessageDelegate:self object:nil];
```

2. 实现协议方法:

```
- (void)onMessageRecalled:(long)messageId{

}
```

历史消息获取 功能描述

更新时间:2024-08-30

聊天室会话的历史消息可保存在云端，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。该功能要求您已在控制台 IM 服务管理 页面为当前使用的 App Key 开启**聊天室消息云端存储服务**。**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。

参数说明

输入参数说明

参数	类型	必填	说明
targetId	NSString	是	聊天室 ID
recordTime	long long	是	起始的消息发送时间戳，毫秒
count	int	是	需要获取的消息数量， 0 < count <= 200
order	RCTimestampOrder	是	拉取顺序。 `RC_Timestamp_Desc`：倒序（降序），表示拉取小（早）于 `recordTime` 的消息。 `RC_Timestamp_Asc`：正序（升序），表示拉取大（晚）于 `recordTime` 的消息。
successBlock	Block	-	获取成功的回调
errorBlock	Block	-	获取失败的回调

回调参数说明

`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
syncTime	long long	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。

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	获取失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] getRemoteChatroomHistoryMessages:@"targetId"
recordTime:recordTime
count:50 order:RC_Timestamp_Desc
success:^(NSArray *messages, long long syncTime) {

}
error:^(RCErrorCode status) {

}];
```

聊天室常见问题

聊天室产品问题

更新时间:2024-08-30

融云 IM 聊天室中是否包含视频流？

IMLib 聊天室中不包含视频直播，包含与 IM 相关功能，如消息发送、聊天室控制等。

视频直播视频流的录制、播放可使用融云 RTC SDK 低延迟直播 。

聊天室登录、重连问题

登录聊天室是否要先连接融云？

必须先连接融云，所有接口都是连接融云后才可以调用。

如何实现匿名登录？

从融云 SDK 的角度，都是需要确定一个 UserId 来与融云服务器连接，以保证其后所有的操作都有指向性。

但作为 App 应用层来讲，可以不暴露融云的 UserId，所谓匿名就是以一个随机的或者说不跟你们应用帐号绑定的 UserId 来登录。

如何处理断线情况？

进入聊天室后如果出现断开连接，当与融云服务器的连接恢复后，SDK 会尝试重连聊天室。重连过程不需要用户参与，用户可以设置聊天室状态监听，来获取聊天室重连的状态信息。

聊天室消息问题

如何在进入聊天室时获取最新消息？

对于同一个聊天室，只存储该聊天室的 50 条最新消息，也就是说移动端用户进入聊天室时，最多能够拉取到最新的 50 条消息。

可通过融云加入聊天室方法，设置获取历史消息条数，加入后可获取到对应条数的历史消息。

如何发送用户进入聊天室消息？

如用户加入聊天室时，需要发送“XX加入聊天室”的通知消息，可通过融云 Server API 发送聊天室消息接口，自行向聊天室中发送一条消息，可以在消息体中附加其他需要的信息属性。

如何向应用下所有聊天室发送消息？

在直播聊天场景下，如需要向应用下所有聊天室发送消息时，可使用聊天室广播消息功能 ，发送后所有在聊天室中的用户都将收到此条消息，如：聊天室管理通知、优惠活动系统通知等。

直播聊天室中有哪些常见消息类型？

常见消息类型包含：文字、语音、图片、点赞、礼物、弹幕及自定义消息实现的命令消息，用来辅助直播业务。

聊天室中的哪些功能可以用消息实现？

文字、语音、图片、点赞、礼物、弹幕、开播通知等功能均可使用消息实现。

直播聊天室中如何准确的使用消息类型？

由于当消息密集时，服务会自动抛弃消息，开发者设计消息类型一定要将文字、语音、图片、点赞、礼物等业务消息，单独定义消息类型使用，不要通过一个消息类型的不同属性实现功能。

如何发送展示点赞消息？

通过融云提供的自定义消息功能，定义礼物的消息类型，实现点赞功能，鉴于有些用户会连续点击，建议做消息合并机制处理。比如设置定时器，每 5 秒触发一次，将 5 秒内所有的点赞数一次性发出去，降低服务器压力，保证重要消息的畅达。

如何发送展示礼物消息？

通过融云提供的自定义消息功能，定义礼物的消息类型，实现礼物功能，展示方式就是界面上的一个 View，完全可以根据你们的需求来自定义。因为聊天室类型的应用 UI 都是特殊定制的，作为融云 SDK 来讲，只是负责消息通知，并不负责 UI 绘制。

什么是实时消息路由？

由融云服务端把指定 App Key 下产生所有的消息回调到到开发者服务器的过程叫实时消息路由，消息到达开发者服务器后便于对聊天室消息进行数据实时分析和挖掘。

调用客户端 SDK 接口发送消息为什么自己发的自己收不到？

自己发送的消息是 不会触发消息监听 的，自己将发送成功后的 Message 对象渲染到页面即可。

调用 Server API 接口发送消息为什么自己发的自己收不到？

Server API 发送的消息, 默认发送方是不会触发消息监听的，可以在本地模拟一条 Message 对象渲染到页面即可。

或者在使用 Server API 发消息时，isIncludeSender 填 1，则发送者也可收到发送的消息。

断网后再连接为何有时可收到消息，有时无法收到消息？

聊天室中用户在离线 30 秒后或离线后聊天室中产生 30 条消息时会被自动退出聊天室，如有需求不希望自动退出，可将用户加入到聊天室白名单中。

聊天室用户问题

如何显示用户昵称和头像？

融云不维护用户的信息，所以用户的昵称和头像信息都需要您自己来维护管理。如果您基于 IMKit 用户，可参见「用户信息显示」相关章节。

如何显示用户列表？

用户列表显示逻辑是根据您的需求来确定。举一个例子：聊天室人数从几百到十几万不等，比如需求定为只显示 10 个用户的头像，那就可以根据用户等级或者加入时间的顺序，选取 10 个在线用户的信息来显示。可以由应用自己的服务器来维护这个用户列表，不定时通知给全体成员，也可以每个用户加入/退出聊天室时发送状态消息，由端上接收并维护这个列表。显示的方法也是移动端进行 UI 绘制。

如何显示用户等级信息？

融云不维护用户的信息，所以关于用户信息和等级的显示，都需要您自己进行开发。通常有两种处理方式：

1. 在收到消息需要展示用户信息的时候，您可以通过您的 Server 端接口获取到用户信息来展示。
2. 用户在发送消息时把自己的信息附加到消息中，接收方通过解析消息获得用户信息。

一个用户是否可多个端同时加入聊天室？

融云默认支持一个 Android、iOS、Web、小程序用户同时在线，如果同种设备需要多端登录，需要单独开通。Web 每个浏览器 Tab 算一个端。

开通位置：控制台 -> 选择应用 -> 服务管理 -> IM 服务管理 -> 多设备消息同步

一个用户是否可以加入多个聊天室？

融云默认允许一个用户（不区分平台）同时加入一个聊天室，如有加入多聊天室需求，需至控制台开通。

开通位置: 控制台 -> 选择应用 -> 服务管理 -> 免费基础功能 -> 单个用户加入多个聊天室

什么是用户白名单？

白名单中用户发送的消息受到保护，在聊天室消息量较大的情况下不会被丢弃，可将重要用户加入用户白名单，保证重要用户发送消息不会被抛弃，如：主播、场控等。

聊天室管理功能

如何实现聊天室踢人功能？

可通过调用融云 Server API 聊天室封禁服务接口，实现将用户踢出聊天室。

如何实现在聊天室中禁言用户？

可通过融云 Server API 接口，设置聊天室禁言用户，被禁言用户不能在聊天室中发送消息，详细请查看聊天室成员禁言服务。

如何实现聊天室成员自动退出机制？

聊天室中用户离线后从第一条新消息产生开始计时, 倒数 30 秒退出聊天室。或离线后聊天室中产生 30 条消息时会被自动退出聊天室。如应用场景中有驻留用户，不允许被自动退出聊天室，则需要将用户加入到聊天室白名单中。

如何设置聊天室用户白名单？

可通过融云 Server API 接口，设置聊天室白名单，详细请查看聊天室用户白名单服务。

聊天室如何销毁？

销毁方式有两种:

1. 主动销毁聊天室。该方式要求需要调用融云 Server API 的销毁房间接口

2. 或聊天室中一小时内无人说话，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

如何避免聊天室被自动销毁？

可通过融云 Server API 接口，设置需要保活的聊天室，设置后聊天室不会被自动销毁，详细请查看聊天室保活服务 。

如何实现用户如何同时加入多个聊天室？

默认同一用户不能同时加入多个聊天室，可通过提交工单方式开启用户同时加入多个聊天室功能。同时，在开启多设备消息同步情况下，多端用户可同时加入到多个聊天室。

如何动态获取聊天室在线人数？

调用 Server API 中的查询聊天室内用户 方法，可通过返回值 `total` 获取聊天室中在线人数。

聊天室消息进阶功能

什么是消息优先级？

融云消息服务中，消息存在高级别、低级别之分 API，可设置消息界别的高低，默认全部为高级别，当服务器负载高时低级别消息优先被丢弃，让出资源给 高级别消息，确保高级别消息不被丢弃，消息多时可主动抛弃指定消息，如: 点赞消息

具体可参考文档: [低级别消息/白名单]

消息的分级与抛弃模型是什么？

当聊天室内人数众多，消息量会变得非常大，这时可能会出现服务端超过预设承载能力或者分发的消息量超过客户端消息接收能力的情况，这时，就需要引入消息分级机制。

融云并没有对任何消息进行抛弃，但是在消息量极大的情况下，比如 1 万人到百万左右的聊天室内，消息并发量极大的情况下，每个用户端能收发到的消息和体验已经很有限，因此消息抛弃指的是确保用户端总是能收到最重要的消息，因此不重要的消息看起来就像是被抛弃了。

在开发过程中，除官方的普通文本消息之外，开发者需要针对不同的消息类别定义不同的消息类型，以便通过消息的 ObjectName 设置消息分级。目前融云支持两级消息分类，分别是高优先级消息和低优先级消息。当发生消息抛弃行为时，优先抛弃低优先级消息。

消息量大时的丢弃策略是什么？

聊天室场景下，融云服务端默认单个聊天室中上行消息处理能力是每 200 毫秒 40 条，其中 20 条为高优先级消息使用配额，另外 20 条为高优先级和低优先级消息共同使用。

在聊天室消息量较大的情况下，融云服务器会按消息发送的时间顺序，将超出消费上限的最新消息丢弃，确保服务器稳定。

针对以上情况，为保证聊天室中重要消息不被丢弃，融云提供了以下服务：

1. 聊天室用户白名单功能 ，白名单中用户发送的消息受到保护，在聊天室消息量较大的情况下也不被丢弃。
2. 聊天室消息白名单功能 ，该名单中的消息受到保护，在聊天室消息量较大的情况下也不被丢弃。
3. 聊天室低级别消息设置 ，该功能为设置低优先级的消息类型，在聊天室消息量较大的情况下，此类型的消息将被优先抛弃，确保重要消息不被丢弃。

注：未设置情况下融云的所有消息均为高优先级消息。

以上功能设置后，服务端收到聊天室上行消息时，根据消息类型的设置状态，处理逻辑如下：

1. 上行消息为低优先级消息，则占用高优先级和低优先级共有的 20 条消息配额，如配额已经用完，之后收到的低优先级消息将被抛

弃，不占用高优先级的配额。

2. 上行消息为默认高优先级消息，则先占用高优先级的 20 条消息配额，如配额已经用完，高优先级和低优先级共用的 20 条配额未占用完时，则占用高低优先级消息的共同配额，直到全部占用，之后收到的高、低优先级消息都将被抛弃。

3. 上行消息为设置的聊天室消息白名单中的消息或用户白名单中的用户发送的消息时，该类消息不会丢弃，但会占用每 200 毫秒 40 条的消息配额，优先占用高、低优先级消息共用的 20 条配额，其次占用高优先级消息的 20 条配额。配额被全部占用后，再收到高、低优先级的消息时都将被丢弃，但如收到白名单中的消息时则不会被丢弃，按时间顺序正常下发。

> 单个聊天室可消费的每 200 毫秒 40 条的上行消息配额，开通专有云后可进行配置

用户白名单、消息白名单、消息优先级三者均和消息有关，是否有先后顺序？

用户白名单、消息白名单、消息优先级提供了 3 种设置消息级别的不同维度，级别均为高级别，可根据业务需要按需使用

如何统计聊天室点赞消息数？

您可以通过开通服务端实时消息路由来实现此功能，用户在发送点赞消息时，这条消息同步给您的应用服务器。您的应用服务器通过消息类型来分析统计这些数据。

查看服务端实时消息路由功能介绍。

如何实现消息回看功能？

如果直播聊天室中需要实现消息回看功能，则需要开通聊天室消息云端存储功能。开通成功后，可通过接口获取聊天室中历史消息，消息回看 UI 界面的展示需要由您自己实现。

聊天室中存储的消息类型及自定义消息如何设置存储，请查看聊天室消息云端存储功能介绍。

多端情况下退出聊天室

在多端登录情况下，一端退出聊天室其他终端是否可同步退出聊天室？

在开通了多端同时登录情况下，用户登录多个终端，加入聊天室后，如在一端退出聊天室，其他端不会同步退出，仍然可以收到聊天室中的消息，如果需要一端退出聊天室后，其他已登录的终端也一起退出聊天室，则需要在退出时发送一条命令消息通知其他端退出聊天室，此步骤需要开发者自行实现。

另外，需要注意如果用户登录终端 A 后，加入了聊天室并在聊天室的会话界面中，这时用户又登录了终端 B ，但未进入聊天室会话界面，这时 A 、B 两个终端都会收接聊天室的消息，只是 B 端的聊天室消息不进行展示。如用户在 A 端退出聊天室，这时融云认为用户在 B 端仍然在聊天室中，会继续向 B 端发送聊天室消息。如果需要 B 端同时也退出聊天室，则需要 A 端在退出时发送一条命令消息通知其他端退出聊天室，此步骤需要开发者自行实现。

系统通知介绍 概述

更新时间:2024-08-30

一般系统消息特指会话类型（ConversationType）为“系统（SYSTEM）”的会话中的消息。

系统消息（System Message）是一种业务概念，是指利用系统帐号（非用户帐号，用户不可登录）向用户发送的消息，既可以是通过调用全量通知接口发送给所有人的消息，也可以是加好友提醒等单个用户通知消息。

> ⚠ 警告
>
> 系统通知消息，只能通过 Server API 进行发送，终端用户收到系统消息后，不支持消息回复功能。
> 具体消息发送方式详见：服务端集成-系统通知

主要功能

功能	描述
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。
消息提醒	离线状态，系统会话有新消息时，支持 Push 通知。
本地存储	存储在移动端本地，提供本地消息搜索功能。
历史消息	提供服务端消息存储存功能，需开通单群聊消息云存储，默认存储时长为 6 个月。
单个用户通知	向单个或多个用户发送系统消息。
标签用户通知	向标签下的所有用户发送系统消息，用户标签属性需要开发者通过 Server API 接口进行设置。
全量用户落地通知	向应用下的所有用户发送广播消息，用户在线情况下会直接收到消息内容，未在线时此条消息会转为 Push 进行推送，同时该条消息做为离线消息进行存储，用户下次登录后会通过离线消息获取。
全量用户不落地通知	向应用下的所有用户发送远程 Push，无论用户是否正在使用应用，该通知只会展示在通知栏，通知内容不会做为消息存储到本地。

消息类型

消息类型	描述
文字消息	用来发送文字类消息，其中可以包括表情、超链接（会自动识别），客户端收到消息后计入未读消息数、进行存储。
语音消息	发送高质量的短语音消息，录制的语音文件存储到融云服务端，语音文件格式为 AAC，时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。
图片消息	用来发送图片类消息，客户端收到消息后计入未读消息数、进行存储。图片缩略图格式为 JPG，大小建议不超过 100k。
GIF 图片消息	用来发送 GIF 动态图片消息，客户端收到消息后计入未读消息数、进行存储。
图文消息	用来发送图文消息，包含一个标题，一段文字内容和一张图片，客户端收到消息后计入未读消息数、进行存储。
文件消息	用来发送文件类消息，客户端收到消息后计入未读消息数、进行存储。
小视频消息	用来发送小视频消息，支持录制发送及选择本地视频文件发送两种方式，录制时长不超过 10 秒，本地选择视频文件方式时长不超过 2 分钟，小视频消息小视频文件格式为 .mp4，客户端收到消息后计入未读消息数、进行存储。

消息类型	描述
命令消息	用来发送通用的指令通知消息，消息内可以定义任意 JSON 内容，与通用命令通知消息的区别是不存储、不计数，此类型消息没有 Push 通知。
自定义消息	融云内置消息类型，无法满足客户业务需求时，可通过自定义消息类型进行实现，接收自定义消息的格式解析及展示处理需要开发者自行实现

消息结构示例

系统通知场景下融云内置消息类型说明：

文本消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:TxtMsg	存储	计数	存储	推送	消息内容

消息结构：

发送文本消息时 content 参数的 JSON 结构如下：

```
{
"content":"Hello world!",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
content	String	是	文字消息的文字内容，包括表情。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

图片消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:ImgMsg	存储	计数	存储	推送	[图片]

消息结构：

发送图片消息时 content 参数的 JSON 结构如下：

```
{
"content":"bhZPzJXimRwrtvc=",
"imageUri":"http://p1.cdn.com/fds78ruhi.jpg",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
content	String	是	图片缩略图，格式为 JPG，Base64 字符串长度建议为 5k，最大不超过 10k，注意在 Base64 进行 Encode 后需要将所有 `\r\n` 和 `\r` 和 `\n` 替换成空。
imageUri	String	是	图片上传到图片存储服务器后的地址。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题：

1、缩略图最大尺寸为：240 x 240 像素，以宽度和高度中较长的边不超过 240 像素等比压缩。

2、大图最大尺寸为：960 x 960 像素，以宽度和高度中较长的边不超过 960 像素等比压缩。

3、图片消息包括两个主要部分：缩略图和大图，如设置为原图发送则为缩略图和原图，缩略图直接 Base64 编码后放入 `content` 中，大图或原图首先上传到文件服务器（融云 SDK 中默认上传到七牛云存储），然后将云存储上的大图或原图地址放入消息体中。

4、发送图片消息时，需要自行上传图片文件到应用的文件服务器，生成地址后进行发送。

GIF 图片消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:GIFMsg	存储	计数	存储	推送	[图片]

消息结构：

发送图片消息时 content 参数的 JSON 结构如下：

```
{
"gifDataSize":34563,
"height":246,
"localPath":"/var/mobile/.../GIF_53",
"remoteUrl":"https://rongcloud-image.cn.ronghub.com/image_jpe64562665566.jpg",
"width":263,
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
gifDataSize	Int	是	GIF 图片文件大小，单位为字节 Byte。
localPath	String	是	下载 GIF 图片后存储在本地的图片地址。
remoteUrl	String	是	GIF 图片的服务器地址。
width	Int	是	GIF 图片宽度。
height	Int	是	GIF 图片高度。
user	String	是	消息中携带的用户信息，IMKit SDK 会话界面中优先显示消息中携带的用户信息，可去掉此属性。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

语音消息

> ⚠ 警告
>
> 从 SDK 2.9.19 版本开始支持 `RC:HQVCMsg` 语音消息功能，`RC:HQVCMsg` 语音消息与旧版本 SDK 不兼容，旧版本 SDK 无法收听新的语音消息。

新语音消息 `RC:HQVCMsg` 和旧版本语音消息 `RC:VcMsg` 不同的是将录制的音频数据存储到服务端，而消息体内只保存 URL。摆脱了消息体 128K 的大小限制，所以拥有更高音质。语音时长上限为 60 秒，客户端收到消息后计入未读消息数、进行存储。

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:HQVCMsg	存储	计数	存储	推送	[语音]

消息结构：

发送高质量语音消息时 content 参数的 JSON 结构如下：

```
{
"localPath":"/9j/4AAQSkZ/2wBaSiimB//9k=",
"remoteUrl":"http://p1.cdn.com/fds78ruhi.aac",
"duration":7,
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

参数说明:

参数	类型	必传	说明
localPath	String	否	采用 AAC 格式进行编码录制的媒体内容本地路径。
remoteUrl	String	是	媒体内容上传服务器后的网络地址。
duration	Int	是	语音消息的时长，最长为 60 秒（单位：秒）。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 参数说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题

1、发送高质量语音消息时，需要自行生成 AAC 格式文件并上传文件到应用的文件服务器，生成地址后进行发送。

文件消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:FileMsg	存储	计数	存储	推送	[文件] + 文件名，如：[文件] 123.txt

消息结构:

发送文件消息时 content 参数的 JSON 结构如下：

```
{
"name":"file.txt",
"size":190184,
"type":"txt",
"fileUrl":"http://www.demo.com/am.ind",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":""
}
```

属性说明：

名称	类型	必传	说明
name	String	是	文件名称。
size	String	是	文件大小，单位：Byte。
type	String	是	文件类型。
fileUrl	String	是	文件地址。
user	String	否	消息中携带的用户信息，详细查看 user 参数说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

user 属性说明：

名称	说明
id	发送用户 Id。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题

1、通过 Server API 发送文件消息时，需要自行上传文件到应用的文件服务器，生成文件地址后进行发送。

小视频消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:SightMsg	存储	计数	存储	推送	[小视频]

消息结构：

发送小视频消息时 content 参数的 JSON 结构如下：

```
{
"sightUrl":"http://rongcloud...com/video...",
"content":"/9j/4AAQSkZ/2wB...hDSaSiimB//9k=",
"duration":2,
"size":734320,
"name":"video_xx.mp4",
"user":
{
"id":"4242",
"name":"Robin",
"portrait":"http://example.com/p1.png",
"extra":"extra"
},
"extra":"extra"
}
```

属性说明：

名称	类型	必传	说明
sightUrl	String	是	上传到文件服务器的小视频地址。
content	String	是	小视频首帧的缩略图进行 Base64 编码的结果值，格式为 JPG，注意在 Base64 进行 Encode 后需要将所有 \r\n 和 \r 和 \n 替换成空。
duration	Int	是	视频时长，单位：秒。 • 如使用 IMKit 小视频插件进行录制，默认最长可录制 10 秒。 • 如选择本地视频文件，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。
size	String	是	视频大小单位 Byte。
name	String	是	发送端视频的文件名，小视频文件格式为 `MP4`（H.264+AAC）。
user	String	否	消息中携带的用户信息，详见 `user` 属性说明。
extra	String	否	扩展信息，可以放置任意的数据内容，也可以去掉此属性。

- `user` 属性说明：

名称	说明
id	发送用户 ID。
name	发送用户需要显示的名称。
portrait	发送用户需要显示的头象。
extra	扩展信息，可以放置任意的数据内容。

常见问题

1. 通过 Server API 发送视频消息时，需要自行上传视频文件到应用的文件服务器，生成文件地址后进行发送。
2. IMKit SDK 中目前支持播放的视频文件格式为 mp4，IMLib SDK 中播放功能需要开发者自行实现。

命令消息

ObjectName	存储属性	计数属性	离线属性	推送属性	推送内容
RC:CmdMsg	不存储	不计数	存储	不推送	无

消息结构：

运营平台向终端发送指令信息时可使用此命令消息，消息中 content 参数的 JSON 结构如下：

```
{
"name":"AtPerson",
"data":"{\"sourceId\":\"9527\"}"
}
```

属性说明：

名称	类型	必传	说明
name	String	是	命令名称，可以自行定义。
data	String	是	命令的内容。

获取全部会话　　获取本地会话列表

更新时间:2024-08-30

开发者可通过下面接口获取 SDK 在本地数据库生成的会话列表，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。

参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array
count	int	是	获取的数量
startTime	long long	是	会话的时间戳，获取的是这个时间戳之前的会话列表，0 表示从最新开始获取

代码示例

```
NSArray *conversationList = [[RCIMClient sharedRCIMClient] getConversationList:@[@(ConversationType_SYSTEM)]
count:100 startTime:0];
```

获取远端会话列表

为提高性能和获取效率，SDK 的会话列表会根据消息的收发直接在本地数据库生成，不需要从服务端获取。

常见问题

Q1: 当用户卸载重新安装后发现会话列表为空或者部分会话丢失
A1: 由于会话列表是从本地数据库获取的, 是在 SDK 内部数据库存储的，所以当用户在卸载的时候会删除本地数据库，导致重新安装后会话列表为空，而会出现部分会话的原因是因为开启了 离线消息补偿 功能，具体操作是在后台购买 多设备消息同步 功能，这个功能里默认涵盖了今天(当天 0 点)的离线消息补偿，所以当用户在新设备登陆会触发离线消息补偿功能，从而获取到部分会话，造成部分会话丢失的错觉。如果想要更多天的离线消息补偿，可 提交工单 修改，最多支持 7 天，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。

删除全部会话

清除本地会话列表

更新时间:2024-08-30

开发者可调用 删除指定类型 会话接口，并传入业务中用到的所有的会话类型即可清除 SDK 在本地数据库内生成的会话列表数据。

清除远端会话列表

由于客户端的会话列表是根据消息在本地生成的，融云服务端并不存储会话列表，因此无需清除远端会话列表。

获取指定会话

功能描述

更新时间:2024-08-30

开发者可以通过下面接口来获取某个会话的详细信息。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，系统会话传入 ConversationType_SYSTEM
targetId	NSString	是	系统会话 ID

返回参数说明

返回值	返回类型	说明
conversation	RCConversation	会话的对象

代码示例

```
RCConversation *conversation = [[RCIMClient sharedRCIMClient] getConversation:ConversationType_SYSTEM
targetId:@"系统会话 ID"];
```

删除指定会话

删除指定会话

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，系统传入 ConversationType_SYSTEM
targetId	NSString	是	系统会话 ID

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] removeConversation:ConversationType_SYSTEM targetId:@"系统会话ID"];
```

按会话类型删除

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] clearConversations:@[@(ConversationType_SYSTEM),@(ConversationType_SYSTEM),@(ConversationType_GROUP)]];
```

会话草稿信息

获取草稿

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	系统会话 ID

返回参数说明

返回值	返回类型	说明
draft	NSString	该会话中的草稿

代码示例

```
NSString *draft = [[RCIMClient sharedRCIMClient] getTextMessageDraft:ConversationType_SYSTEM targetId:@"系统会话ID"];
```

保存草稿

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	系统会话 ID
content	NSString	是	草稿信息

返回参数说明

返回值	返回类型	说明
success	BOOL	是否保存成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] saveTextMessageDraft:ConversationType_SYSTEM targetId:@"系统会话ID" content:@"这个是草稿内容"];
```

删除草稿

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	系统会话 ID

返回参数说明

返回值	返回类型	说明
success	BOOL	是否删除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] clearTextMessageDraft:ConversationType_SYSTEM targetId:@"系统会话ID"];
```

会话未读数

获取所有会话未读数

更新时间:2024-08-30

参数说明

返回参数说明

返回值	返回类型	说明
count	int	所有的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient] getTotalUnreadCount];
```

获取单个会话未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_SYSTEM
targetId	NSString	是	系统会话

返回参数说明

返回值	返回类型	说明
count	int	该会话内的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient] getUnreadCount:ConversationType_SYSTEM targetId:@"系统会话"];
```

按会话类型获取未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypes	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array
isContain	BOOL	是	是否包含免打扰消息的未读数

返回参数说明

返回值	返回类型	说明
count	int	该类型的会话中所有的未读消息数

代码示例

```
int count = [[RCIMClient sharedRCIMClient]
getUnreadCount:@[@(ConversationType_SYSTEM),@(ConversationType_GROUP)] containBlocked:YES];
```

清除单个会话未读数

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_SYSTEM
targetId	NSString	是	系统会话
timestamp	long long	是	此会话已阅读的最后一条消息的发送时间

返回参数说明

返回值	返回类型	说明
success	BOOL	是否清除成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] clearMessagesUnreadStatus:ConversationType_SYSTEM targetId:@"系统会话" time:0];
```

会话免打扰 设置免打扰状态

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，系统会话传入 ConversationType_SYSTEM
targetId	NSString	是	系统会话 id
isBlocked	BOOL	是	是否屏蔽消息提醒
success	BOOL	是	设置成功的回调
error	BOOL	是	设置失败的回调

回调参数说明

`error` 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] setConversationNotificationStatus:ConversationType_SYSTEM
targetId:self.userId
isBlocked:swch.on
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

获取免打扰状态

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，系统会话传入 ConversationType_SYSTEM
targetId	NSString	是	系统会话 id
success	BOOL	是	设置成功的回调
error	BOOL	是	设置失败的回调

回调参数说明

`success` 说明：

回调参数	回调类型	说明
nStatus	RCConversationNotificationStatus	会话设置的消息提醒状态

error 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] getConversationNotificationStatus:ConversationType_SYSTEM
targetId:self.userId
success:^(RCConversationNotificationStatus nStatus) {}
error:^(RCErrorCode status){}];
```

获取免打扰状态列表

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypeList	NSArray	是	会话类型的数组 需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
array	NSArray	屏蔽消息提醒的会话RCConversation的列表

代码示例

```
NSArray *array = [[RCIMClient sharedRCIMClient] getBlockedConversationList:@[@(ConversationType_SYSTEM)]];
```

设置免打扰时间

参数说明

输入参数说明

参数	类型	必填	说明
startTime	NSString	是	开始消息免打扰时间，格式为 HH:MM:SS
spanMins	int	是	需要消息免打扰分钟数， 0 < spanMins < 1440 您设置的起始时间是 00：00， 结束时间为 23：59，则 spanMins 为 23 * 60 + 59 = 1339 分钟
success	BOOL	是	屏蔽成功的回调
error	BOOL	是	屏蔽失败的回调

回调参数说明

error 说明：

回调参数	回调类型	说明
status	RCErrorCode	设置失败的错误码

代码示例

```
[[RCIMClient sharedRCIMClient] setNotificationQuietHours:@"00:00:00"
spanMins:1439
success:^{}
error:^(RCErrorCode status) {}];
```

会话置顶 设置是否置顶

更新时间:2024-08-30

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_SYSTEM
targetId	NSString	是	系统会话 id
isTop	BOOL	是	是否置顶

返回参数说明

返回值	返回类型	说明
success	BOOL	设置是否成功

代码示例

```
BOOL success = [[RCIMClient sharedRCIMClient] setConversationToTop:ConversationType_SYSTEM targetId:@"系统会话id" isTop:YES];
```

获取置顶列表

参数说明

输入参数说明

参数	类型	必填	说明
conversationTypes	NSArray	是	会话类型的数组，需要将 RCConversationType 转为 NSNumber 构建 Array

返回参数说明

返回值	返回类型	说明
conversations	NSArray <RCConversation *>	置顶的会话RCConversation的列表

代码示例

```
NSArray *conversations = [[RCIMClient sharedRCIMClient]
getTopConversationList:@[@(ConversationType_SYSTEM),@(ConversationType_GROUP)]];
```

消息接收

功能描述

更新时间:2024-08-30

开发者可通过此接口拦截到 SDK 接收到的消息，并进行响应的业务操作。

接口实现

SDK 通过消息监听接收消息，详细可参考 设置消息监听 文档

历史消息获取　　本地获取

更新时间:2024-08-30

开发者可通过下面接口获取 SDK 存储在本地数据库的历史消息。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型，{{type}}传入 ConversationType_GROUP
targetId	NSString	是	群组 ID
oldestMessageId	long	是	最后一条消息的 messageId，获取此消息之前的 count 条消息，第一次调用应设置为 -1。
count	int	是	需要获取的消息数量，每次最多获取 20 条

返回参数说明

返回值	返回类型	说明
history	NSArray	消息实体RCMessage对象列表

返回参数说明

代码示例

```
NSArray *history = [[RCIMClient sharedRCIMClient] getHistoryMessages:ConversationType_GROUP
targetId:@"群组 ID"
oldestMessageId:lastMessageID
count:count];
```

远端获取

开发者可通过下面接口获取 SDK 存储在服务端的历史消息。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。

参数说明

输入参数说明

参数	类型	必填	说明
conversationType	RCConversationType	是	会话类型
targetId	NSString	是	目标会话 ID

参数	类型	必填	说明
option	RCRemoteHistoryMsgOption	是	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

回调参数说明

`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

代码示例

```
RCRemoteHistoryMsgOption *option = [RCRemoteHistoryMsgOption new];
option.recordTime = recordTime;
option.count = 10;
option.order = RCRemoteHistoryOrderDesc;

[[RCIMClient sharedRCIMClient] getRemoteHistoryMessages:ConversationType_GROUP targetId:@"群组 ID" option:option
success:^(NSArray *messages, BOOL isRemaining) {

} error:^(RCErrorCode status) {

}];
```

集成 APNs 远程推送

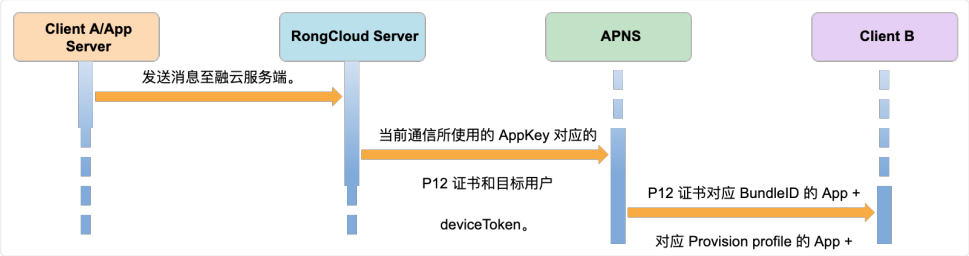
更新时间:2024-08-30

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

如果需要融云向 iOS 应用程序发送推送通知，您需要向融云提供有关 APNs 的信息（App ID、证书等），授权融云服务端与 APNs 通信。

以使用 p12 证书为例:



融云服务端支持使用以下任一种与 APNs 通信的方式（不同方式要求创建对应的证书文件）：

- **使用验证令牌与 APNs 通信：** 您需要创建扩展名为 .p8 的 Key (密钥) 文本文件（也称为 p8 证书文件），并提交融云。

  如果您选择使用签名密钥，对于您发送的每个推送通知，融云都会使用令牌针对 APNs 进行身份验证。借助您的签名密钥，您可以将推送通知发送到 APNs 生产环境和沙盒环境。

  与证书不同，签名密钥不会过期。您只需提供一次您的密钥，以后不用更新。此外，您可以为多个应用程序使用相同的签名密钥。有关更多信息，详见 Apple 开发者帐户帮助：使用验证令牌与 APNs 通信 。

- **使用 TLS 证书与 APNs 通信：**您需要创建文件扩展名为 .p12 的证书文件，并提交融云。

  如果您选择使用 TLS 证书，对于您发送的每个推送通知，融云都会使用该证书针对 APNs 进行身份验证。APN 证书可以同时支持生产环境和沙盒环境，也可以只支持沙盒环境。您可以从 Apple 开发者账户获取该证书。

  证书在一年后过期。出现这种情况时，您必须创建新证书，然后将其提供给融云以更新推送通知送达。有关更多信息，详见 Apple 开发者帐户帮助：使用 TLS 证书与 APN 通信 。

配置 App ID

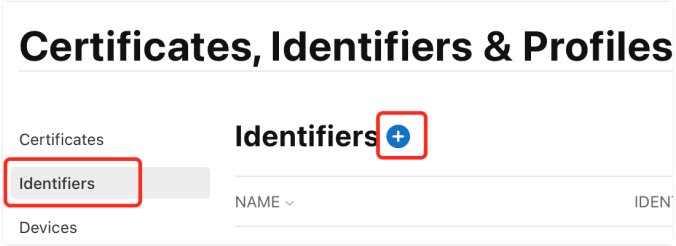
您需要持有一个 Apple 的 App ID 才能使用 Apple 的推送服务。以下描述了如何从 Apple 开发者账户 页面创建 App ID，以及为 App ID 启用 Push Notifications（推送通知功能）。

创建 App ID

您必须持有 App ID 才能在Apple 开发者账户 创建证书。如已持有 App ID，请跳过此步骤，直接查看创建证书。

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 (+)。

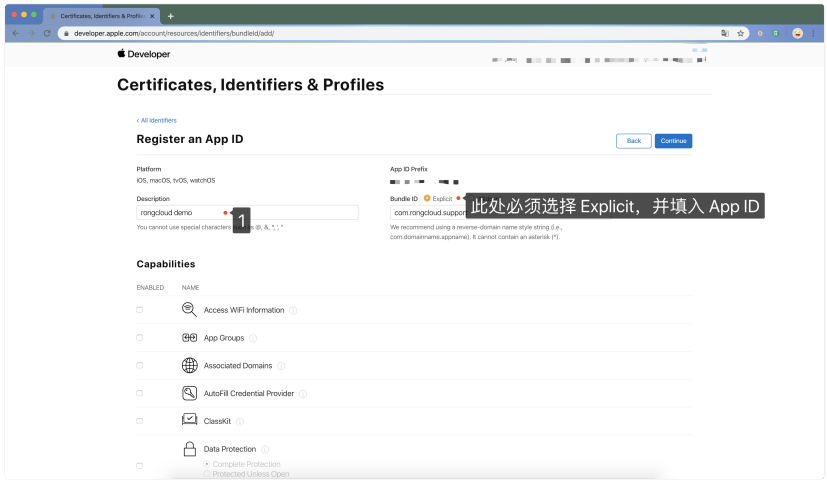


2. 从选项列表中选择 **App IDs**（App ID），点按 **Continue**（继续）。

3. 从选项中，确认已自动选择了 App ID 类型，然后点按 **Continue**（继续）。

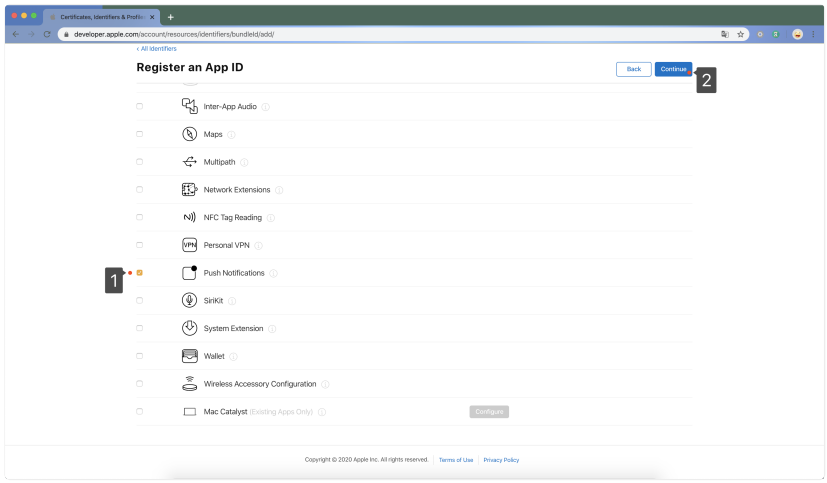
4. 在 **Description**（描述）栏位中输入 App ID 的名称或描述。请选择 **Explicit App ID** （精确 App ID），并在 **Bundle ID** 栏位中输入 App 的 **Bundle ID**。

   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）。

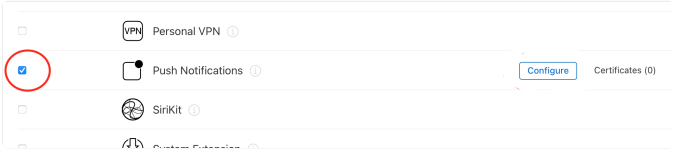
关于上述创建 App ID 步骤的详细说明，可参考 Apple 开发者帐户帮助文档：注册 App ID 。

为 App ID 开启推送功能

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**。



以上步骤也可以参考 Apple 开发者账户帮助文档： 启用推送通知 。

创建证书

融云服务端需要持有您的 APNs 证书才能与 APNs 通信，向 iOS 客户端发送推送通知。

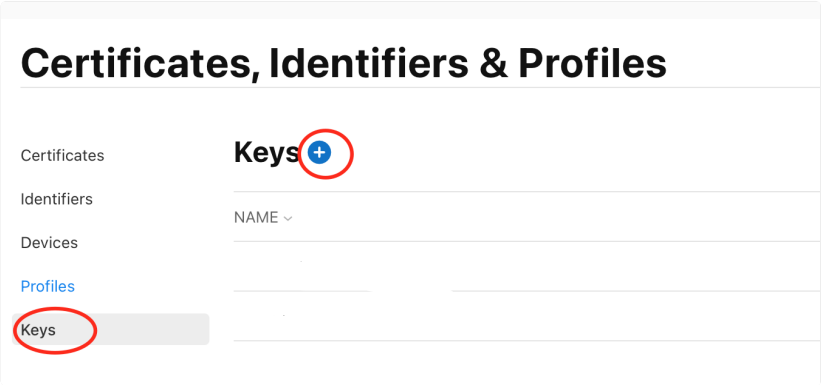
您可以选择创建 p8 证书或者 p12 证书。

创建 p8 证书（APNs Auth Key 鉴权）

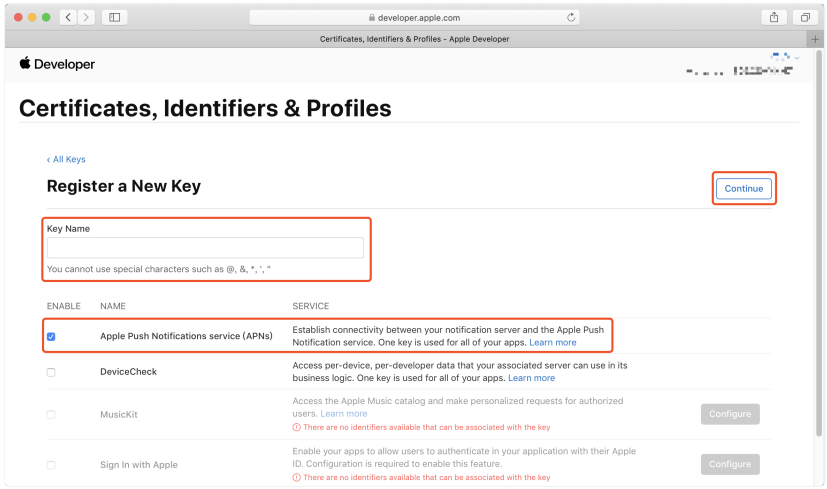
本步骤描述如何从 Apple 开发者账户 创建并下载启用了 APNs 的私钥（也称为 p8 证书文件）。p8 证书不会过期，可用于多个 App ID 服务。

如需了解 p8 证书及“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 。

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）。

以上步骤也可参考 Apple 开发者账户帮助文档： 创建并下载启用了 APNs 的私钥。

创建 p12 证书

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，最后将下载的证书（.cer）文件在 Mac 上导出为 .p12 格式的证书。

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)。在 Apple 开发者账户 中生成证书文件时需要上传该 CSR 文件。

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。

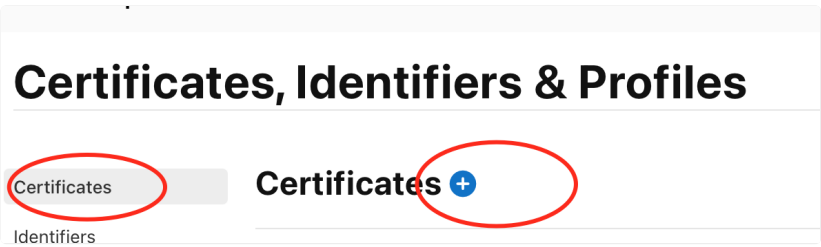
生成推送证书

本步骤描述从 Apple 开发者账户 生成 APNs 客户端 TLS 证书的两种方式。两种方式仅存在 UI 操作上的区别，任选一种即可。

> 注意：如有多个 App，则需要为每个 App 生成单独的客户端 TLS 证书。

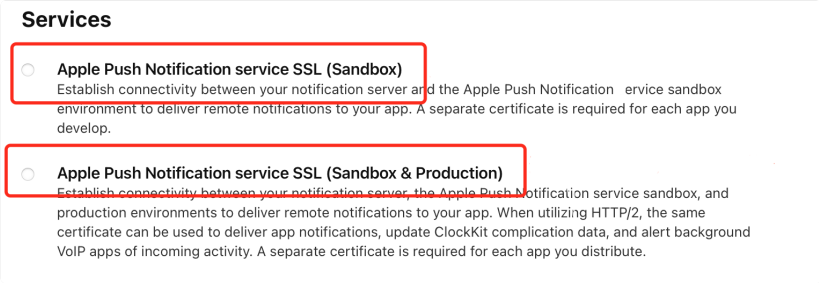
方式一：从 **Certificates**（证书）页面操作

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 (+)。

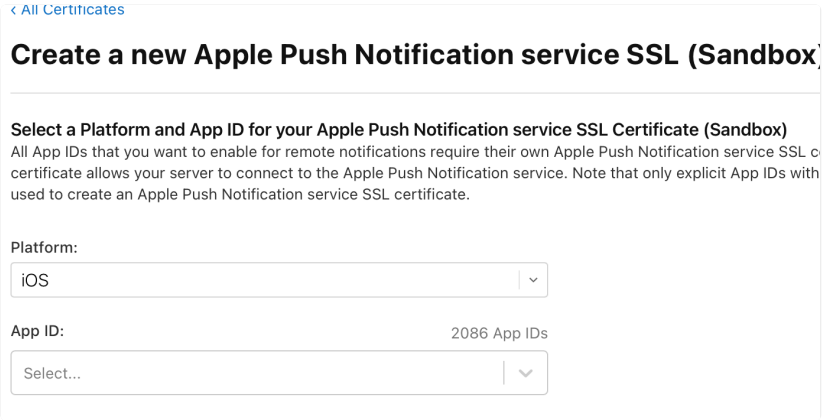


2. 选择证书种类，点按 **Continue**（继续）。

- **Apple Push Notification service SSL（Sandbox）**：供开发调试使用。
- **Apple Push Notification service SSL（Sandbox & Production）**：开发调试和生产环境通用。

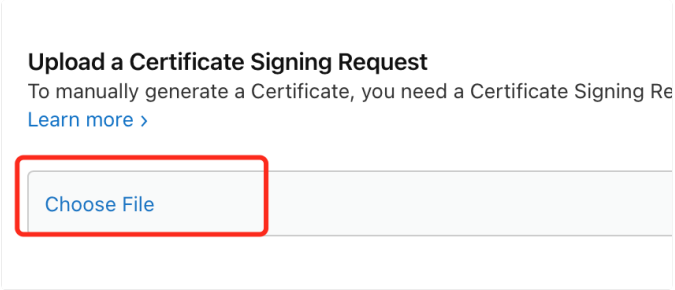


3. 选择需要生成证书的 **App ID**，点按 **Continue**（继续）。



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。

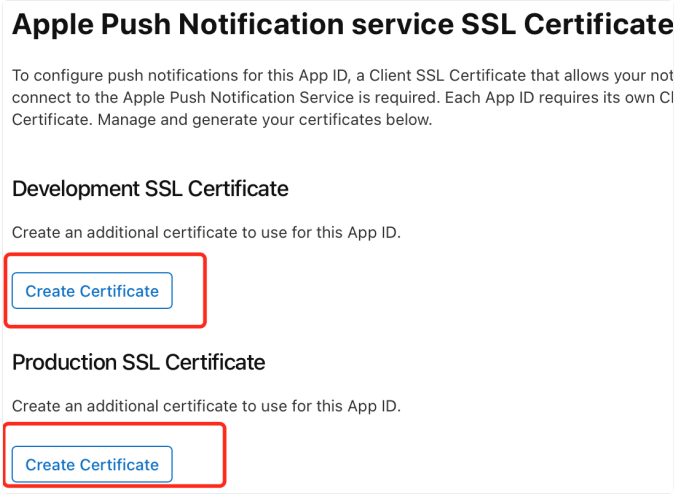
方式二：从 **Identifiers** (标识符)页面操作

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。

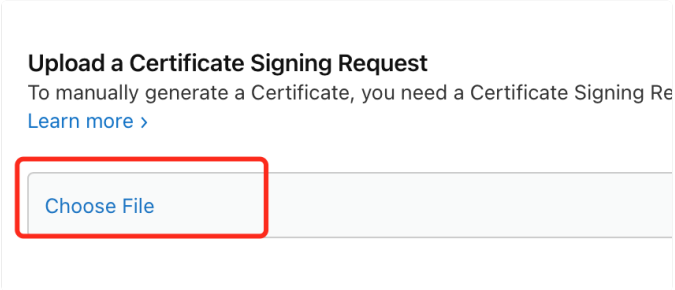
3. 点按 **Configure**（配置），进入创建证书页面。选择需要配置证书的环境，点按 **Create Certificate**（创建证书）。APNs 提供生产与开发两个环境。

   - **Development SSL Certificate**（开发 SSL 证书）
   - **Production SSL Certificate**（生产 SSL 证书）



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。

方式二的步骤也可参考 Apple 开发者账户帮助文档： 生成 APNs 客户端 TLS 证书。

转换证书为 p12 格式

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。

上传证书到融云

您需要将创建的 p8 证书或者 p12 证书上传到融云后台，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**。

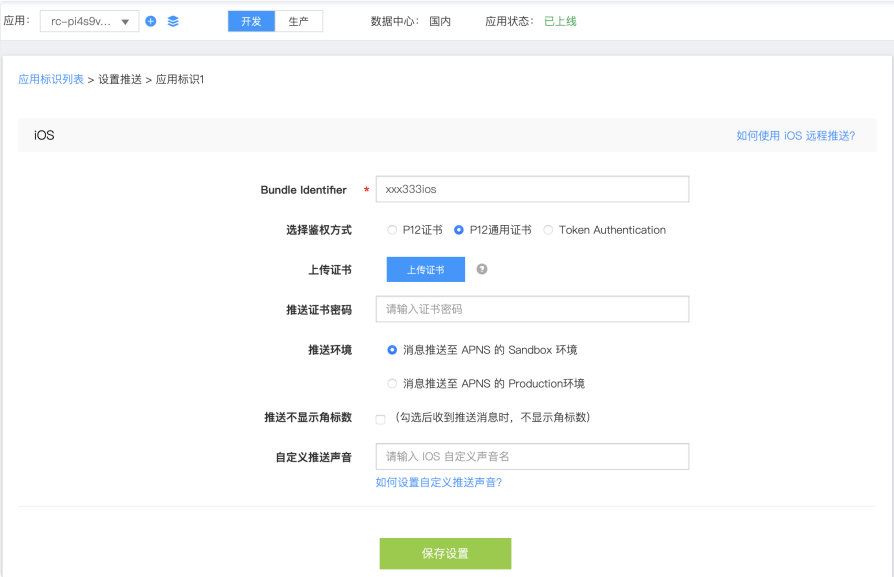
2. 填写 **Bundle Identifier**，选择鉴权方式。

- （p12）根据您需要上传的 P12 证书类型选择鉴权方式，支持 APNs 开发证书（Sandbox）和 APNs 通用证书（Sandbox & Production）。

  1. 如果您需要使用 APNs 开发证书，请选 **P12 证书**。如果使用 APNs 通用证书，请选 **P12 通用证书**。

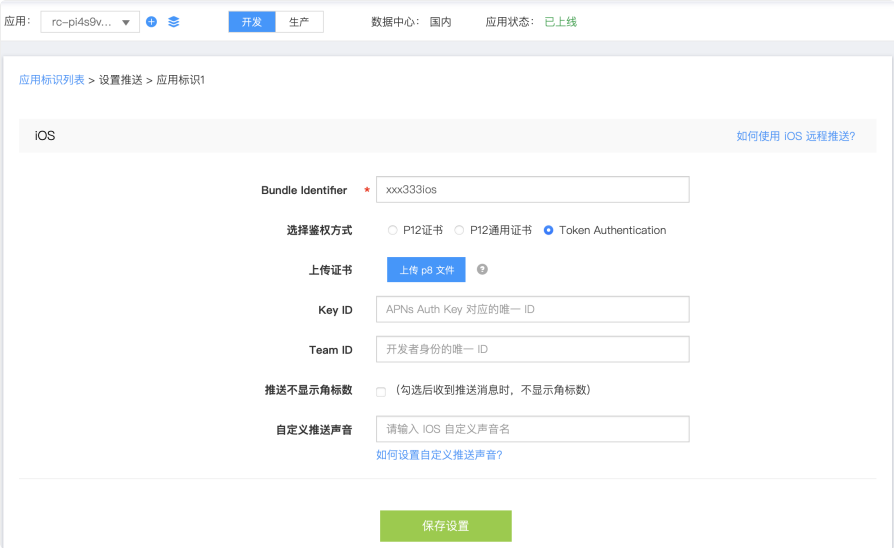
  2. 上传证书，填写证书密码。

  3. 如果使用 APNs 通用证书（Sandbox & Production），还需要选择推送环境。



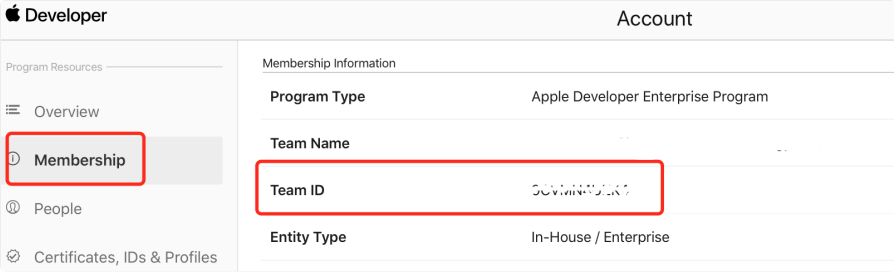
- （p8）如果使用 p8，**鉴权方式请勾选 Token Authentication**。具体步骤如下：

  1. 上传 p8 证书。



2. 填写 **Key ID**（参见创建 p8 证书）。

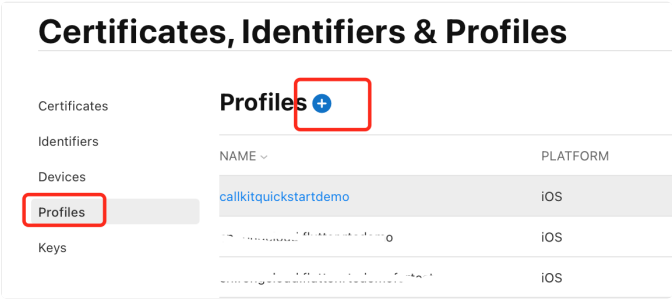
3. 填写 **Team ID**。您可以在 Apple 开发者账户 的 Membership 页面获取 Team ID。



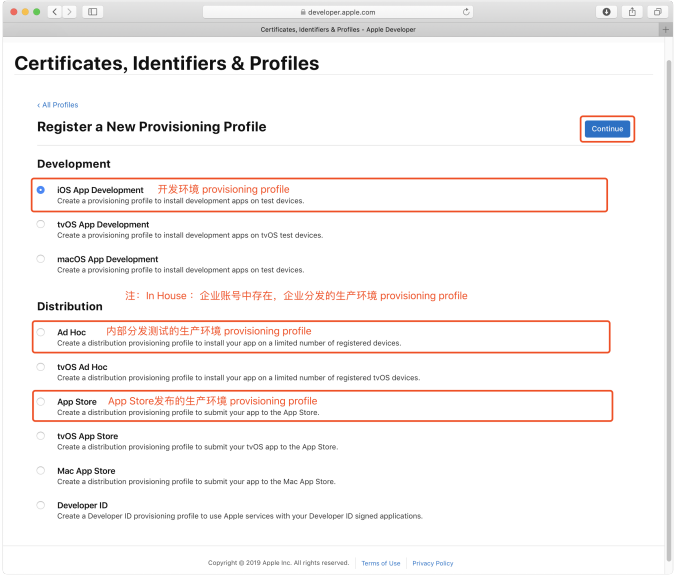
创建 Provisioning Profile 文件

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。

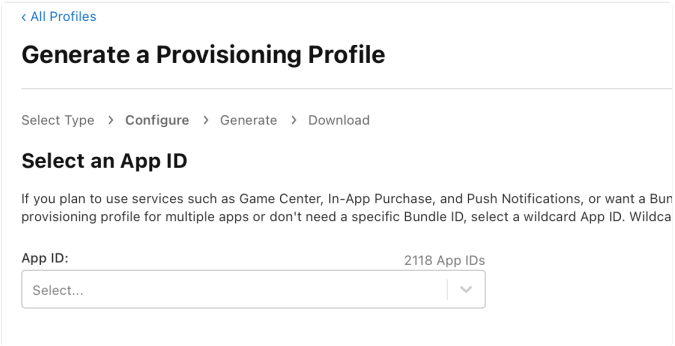
1. 在 Certificates, Identifiers & Profiles （证书、标识符和描述文件）中，点按边栏中的 **Profiles** (标识符)，然后点按左上方的添加按钮 (+)。



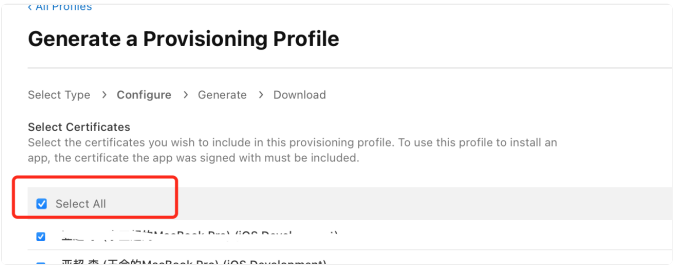
2. 选择 **Provisioning Profile** 的环境后点按 **Continue**



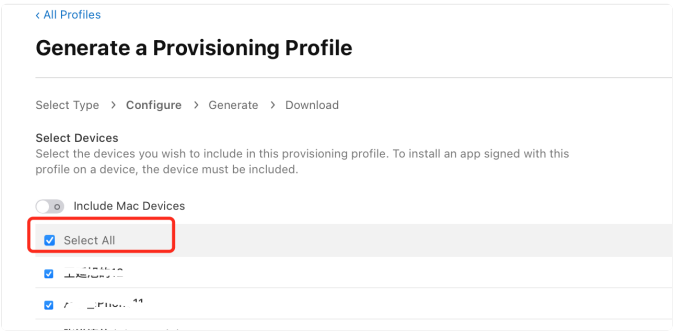
3. 选择要创建 **Provisioning Profile** 的 App ID 后，点按 **Continue**。



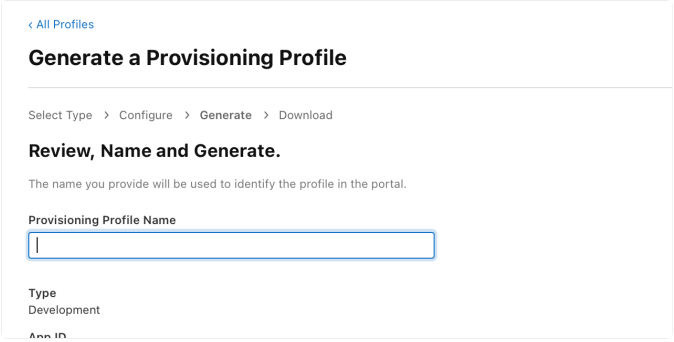
4. 选择所属的开发者证书, 如果创建了多个，选择 **Select All**，点按 **Continue**。



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];
}
```

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。
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

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。

准备推送测试环境

- **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。

准备测试 App

- 如果 App 是 `App-Store` 类型的，但是还没有在 App Store 上架，在上架之前如果想使用生产环境测试远程推送，需要生成一个 `Ad-Hoc` 的 **Provisioning Profile** ，并在 Xcode 中使用其进行打包，导出为 `Ad Hoc Deployment`。 详见 Xcode 帮助文档：Distribute to registered devices (iOS, tvOS, watchOS) 。
- 开发者可使用 `TestFlight` 模式进行测试。详见 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

如果无法收到推送，可通过以下步骤自行排除故障：

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. 根据错误原因排查是否已参照文档正确集成 APNs：

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。
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。
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 。

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。

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。

- 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。

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 。

配置消息的推送属性

更新时间:2024-08-30

在发送消息时，您可以通过设置消息的 RCMessagePushConfig 对象，对单条消息的推送行为进行个性化配置。

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.androidConfig.notificationId = @"Android 的通知 id";
pushConfig.androidConfig.channelIdMi = @"小米的 channelId";
pushConfig.androidConfig.channelIdHW = @"华为的 channelId";
pushConfig.androidConfig.channelIdOPPO = @"OPPO 的 channelId";
pushConfig.androidConfig.typeVivo = @"vivo 的类型";
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：

RCMessagePushConfig 属性介绍

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 (./push-template)。
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）

- **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
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
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
typeVivo	NSString	VIVO 推送通道类型，开发者集成了 VIVO 推送，需要指定推送类型时，可进行设置。 目前可选值 "0"(运营消息) 和 "1"(系统消息)

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。

> 在 App 退到后台且并未被系统回收的情况下不会触发远程推送，仅会触发 SDK 实现的本地通知，此时播放本地通知的铃声。

自定义推送铃声

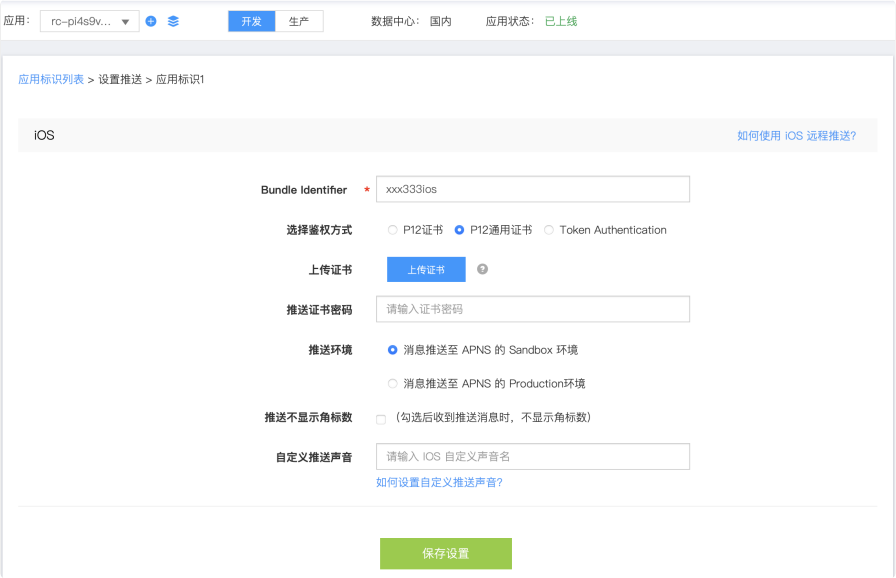
消息推送提示默认使用手机系统设置的声音与振动提示状态。

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。

> 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）。

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。

- 客户端 SDK 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。

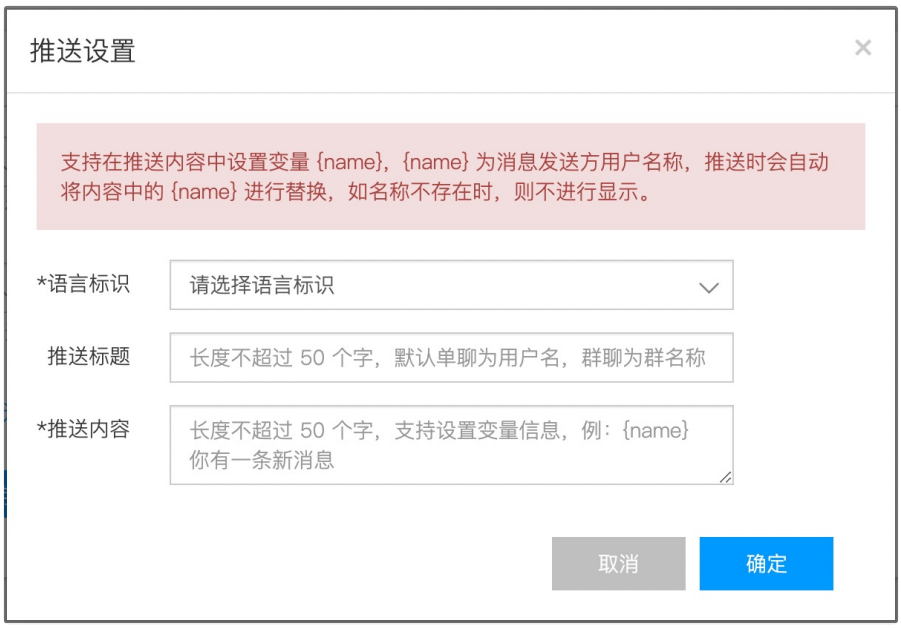
创建多语言推送模板

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。

> - 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**。

更新日志

更新时间:2024-08-30

> 4.X 属于已不再积极开发的历史大版本。请仍使用 4.X 版本的客户尽快升级到 4.1.1.x Stable 版本，或者考虑升级到 5.X 系列的 SDK。当前 iOS 4.X 系列版本中仅维护最新的 4.1.1.X Stable 版本，相关发布说明请转至知识库文档 IMLib/IMKit iOS SDK 4.1.1.X Stable 发布说明 。

4.1.1.6 Stable

发布日期：2021/08/04

- 发布 4.1.1.6 Stable 版本。

4.1.1 Dev

发布日期：2021/01/15

1. 增加了图片缩略图可尺寸可配置功能
2. 优化了处理地理位置缩略图模糊的问题
3. 优化了 IMkit 会话界面下拉消息条数可配置
4. 增加了进入群会话有 @ 我消息时，可点击定位到 @ 我消息处的功能

4.1.0 Dev

发布日期：2020/12/17

问题修复:

1. 修复了会话列表在收取大量新消息卡顿的问题
2. 修复了 disconnect 之后发送定向消息偶现崩溃的问题
3. 修复了断开连接之后，未清空聊天室状态缓存和聊天室属性缓存的问题

4.0.3 Stable

发布日期：2020/11/23

问题修复:

1. 修复了无网状态下进入会话页面设置按钮和发送按钮重叠问题
2. 修复了已读未读样式时未读会刷新两次的问题

3. 修复了设置免打扰之后，杀死打开 APP，免打扰状态异常问题

4. 修复了当协议栈连接过程中出现问题（如 ping 超时），SDK 重连成功之后没有加入聊天室的问题

5. 修复了 RCNaviThread 内存泄漏问题

6. iPhoneXR、iPhone11、iPhoneXS 等设备表情页为 4 页时无法加载 emoji 问题

7. Xcode12 iOS14 选择图片时，点击图片无法跳转到对应的预览页，只能停留在第一张的问题

8. 修复消息扩展功能，setMessageExtra 可以设置为空字符串

9. 断开连接之后，清空聊天室状态缓存和聊天室 kv 缓存

4.0.4 Dev

发布日期：2020/11/12

新增功能：

1. 发送消息时，支持设置该条消息的推送属性，详细如下：

   - 支持设置通知标题
   - 支持设置是否强制显示通知详细，当目标用户设置通知不显示详细时，可通过此属性设置通知显示详细内容，默认为不强制
   - 目标用户未在线时，针对不同 Android 厂商平台，设置消息所使用的 channel_Id，支持小米、华为、OPPO 厂商平台，默认情况下不需要进行设置
   - 目标用户未在线时，针对 vivo 厂商平台，设置消息所使用的推送通道类型，默认为运营消息类型，可设置为系统消息类型。查看 vivo 消息类型说明
   - 支持针对目标用户为 Android 平台的通知 ID 设置功能，目前支持小米、华为厂商平台
   - 支持针对目标用户为 iOS 平台通知分组设置 thread-id
   - 支持针对目标用户为 iOS 平台通知合并功能 apns-collapse-id

功能优化：

1. 优化了通知显示逻辑，默认单聊消息通知标题显示为用户名称，群组消息通知标题显示为群组名称

2. 优化了自定义已读未读样式时，未读会刷新两次的问题

3. 优化 iPad 默认横屏并且分屏时会话列表布局超出当前视图问题

4. 优化了 SDK 心跳机制

5. 会话快捷回复功能，回复内容不做长度限制，开发者可根据情况设置内度，但建议不超过 30 个字

问题修复：

1. 修复了没有网络的情况下进入会话列表，在 emoji 选项时表情面板中显示了表情设置按钮

2. 修复了 connect 时 iOS nil 转成 char* 崩溃问题

3. 修复了设置免打扰之后，杀死应用再打开后，免打扰状态异常问题

4. 修复了当协议栈连接过程中出现问题（如 ping 超时），SDK 重连成功之后没有加入聊天室的问题

5. 修复了 RCNaviThread 内存泄漏问题

6. 修复了 iPhoneXR、iPhone11、iPhoneXS 等设备表情页为 5 页时首次滑动 Emoji 无法加载 Emoji 问题

7. 修复了 Xcode12 iOS14 选择图片时，点击图片无法跳转到对应的预览页，只能停留在第一张的问题

4.0.3.1 Dev

发布日期：2020/09/29

问题修复：

1. 修复了 IMKit 合并转发功能，合并转发的内容中包含手机号或包含 7 位的数字时解析字符乱码的问题

2. 修复了 IMkit 关闭相册后没有释放相册资源的问题

4.0.3 Dev

发布日期：2020/09/18

新增功能：

1. 针对单条消息增加了消息扩展属性设置功能，消息发送前需要设置为可扩展后，才能对该条消息进行扩展信息添加。查看文档

问题修复：

1. 修复了 IMKit 会话界面中上拉加载历史消息时，滑动到最后一个 cell 显示并不触发上拉，接收新消息会加到数据源的中，导致上拉时历史消息断层的问题

2. 修复了 IMKit ConnectionService 的全局状态监听错误的 get 方法

3. 修复了 IMKit 发送合并转发消息，在 Web 端查看详细内容时，显示不正确的问题

4. 修复了 IMLib 多端情况下会话状态免打扰功能，在其他端设置免打扰，端上仍然能收到通知的问题

5. 修复了 IMKib 转发群聊的 @ 消息后，携带 @ 属性的问题，修复后只对消息内容进行转发

4.0.2 Dev

发布日期：2020/08/19

新增功能：

1. IMLib SDK 增加了聊天室 KV 属性变化监听能力，聊天室中属性每次变化时都会同步监听状态

问题修复：

1. 修复了收到小视频消息下载过程中时退到后台，再次进入小视频没有播放也没有显示播放按钮的问题

2. 引用消息支持本地消息搜索功能

3. 修复了因 Hash 碰撞导致点击的图片、视频与打开后的图片、视频不匹配的问题

4. 修复了进入 [有人@我] 的会话界面，无法跳转到的对应消息 Cell 的问题

4.0.1.1 Dev

发布日期：2020/08/03

新增功能：

1. 增加了静默消息功能，发送单条消息时支持设置该条消息没有通知

4.0.1 Dev

发布日期：2020/07/21

问题修复：

1. 修复了合并转发消息 cell 气泡边缘显示问题

2. IMKit SDK 进入会话界面时默认不主动获取历史消息

3. 解决了上传的图片文件包含 / 被转义成 %2F 无法被国外环境识别的问题，图片，小视频，acc 语音等需要上传的文件都受到影响

4. 修复了连续调用两次 joinChatroom，第二次获取不到聊天室自定义属性，一直返回未同步完成的问题

5. 解决了下拉加载更多会话后，从其他页面切回会话列表页面，加载会话变少的问题

6. 修复了 iOS14 上 pageControl 使用图片引起的崩溃问题

功能优化：

1. IMKit SDK 优化了发送消息用户体验

4.0.0.1 Dev

发布日期：2020/06/19

新增功能：

1. 增加了会话免打扰及会话置顶多端状态同步功能

问题修复：

1. 修复了聊天页面长按某条消息时，偶现消息操作菜单栏显示不正确的问题

2. 修复了播放小视频时接收音视频通话，挂断后小视频依然播放

3. 修复了消息发送失败时，会话列表页面和聊天页面 UI 显示不一致的问题

4. 修复了接收小视频消息，下载小视频文件时切到后台，再次启动 App 进度条显示样式的问题。

5. 修复了 iPad 横竖屏切换 Emoji Tab 显示错误的问题

功能优化：

1. 重构了底层通讯协议栈及连接逻辑，使服务运行更加稳定，详细查看 v4 版本升级说明
2. 优化了上传文件服务，当发送富媒体消息时如图片、文件、小视频等，上传文件失败情况下会自动切换到备用服务器进行存储，用户无感知。
3. 删除了废弃的功能接口，详细查看 v4 版本升级说明

2.10.6 Dev

发布日期：2020/05/08

新增功能：

1. 消息引用功能默认打开，会话中长按消息后，可引用该消息进行回复
2. 新增了小视频录制时长配置功能，录制时长上限不超过 2 分钟，默认为 10 秒，RCIM 类文件中通过如下属性进行设置

```
/**
小视频的最长录制时间，单位是秒，默认值是 10s。

@discussion 在集成了融云小视频功能后，可以通过此方法来设置小视频的最长录制时间。录制时间最长不能超过 2 分钟,超过 2 分钟，以 2 分钟为准。
*/
@property (nonatomic, assign) NSUInteger sightRecordMaxDuration;
```

问题修复：

1. 修复了切换暗黑模式偶现 reloadItemsAtIndexPaths 时崩溃问题
2. 修复了 SDK 上若干 Bug，提升了稳定性

功能优化：

1. 优化了合并转发消息中图片、小视频浏览体验
2. 优化了 IMLib 对外接口注释说明，使开发者集成时能够方便对功能的理解

2.10.5 Dev

发布日期：2020/04/10

新增功能：

1. 增加了消息引用功能，默认为关闭状态，RCIM 类文件中设置如下属性进行开通

```
/*!
是否支持消息引用功能，默认值是NO，开启之后，聊天页面长按消息支持引用（目前仅支持文本消息、文件消息、图文消息、图片消息、引用消息的引用）
*/
@property (nonatomic, assign) BOOL enableMessageReference;
```

2. 增加了动态导航功能，使融云 IM 服务在导航地址异常情况下，确保对业务没有影响提升稳定性

问题修复：

1. 修复了 IMKit SDK 特定场景预览视频后，退出了预览界面，还是能听见视频声音的问题

2. 修复了 IMKit SDK 在 iPad 下横竖屏切换时 UI 显示问题

3. 修复了会话列表，聚合会话功能手动加载更多会话时出现重复的问题

功能优化：

1. 开放了 RCHQVoiceMessageCell 类，支持开发者修改高清语音消息显示样式

2. 优化了逐条转发消息过滤功能，音视频通话消息不支持逐条转发功能

2.10.4 Stable

发布日期：2020/02/20

新增功能：

1. 增加了通过 URL 下载多媒体的文件对应的取消下载接口

2. IMKit SDK 增加了视频文件浏览时，支持长按屏幕后，可选择保存视频到本地相册功能

3. 增加了文本消息撤回后重新编辑功能

4. IMLib SDK 增加了聊天室属性自定义存储功能

5. IMKit SDK 增加了对 iOS 13 暗黑 UI 的适配，默认为关闭状态，开启后根据系统设置自动适配。

```
RCIM 中
/*!
是否支持暗黑模式，默认值是 NO，开启之后 UI 支持暗黑模式，可以跟随系统切换
*/
@property (nonatomic, assign) BOOL enableDarkMode;
```

6. CallLib、CallKit SDK 支持了自定义设置接听铃声

7. IMKit SDK 单聊会话支持了阅后即焚功能，该功能需要配置后才能使用

8. IMKit SDK 单聊、群组会话支持了合并转发功能，该功能需要配置后才能使用

9. IMKit SDK 实现了对 iOS 13 的适配

问题修复：

1. 修复了 IMKit SDK 聊天室中不能点击用户头像查看用户信息的问题

2. 修复了消息 cell 暗黑模式下显示错误的问题

3. 修复了发送图片消息引起内存增加，图片消息不释放的问题

4. 修复了被目标用户加入黑名单后，发送消息聊天页面不显示“您的消息已发出，但被对方拒绝”的问题

5. 修复了 iPad 小视频拍摄时横屏按钮被遮挡问题

6. 修复了点击查看图片消息正在下载时，长按保存后点击其他的位置卡死的问题。

7. 修复了大图预览时显示个别大图异常的问题。

8. UIAlertView，UIActionSheet 替换为 UIAlertController 解决了 iOS 13 系统兼容问题。

9. IMKit SDK 修复了接收多张图片消息的会话中预览加载问题

10. IMKit SDK 修复了特殊情况下发送 GIF 消息显示不正确问题

11. 修复了加载大图占用内存过多的崩溃问题

12. 修复了 iPhone 11 录制视频偶现崩溃的问题

13. IMKit SDK 修复了相册中发送小视频收不到 Push 问题

功能优化：

1. 优化了群组中 @群成员时，优先显示用户群昵称逻辑

2. 适配了 iOS 13 Window

3. 优化连接，解决 30013 & 32054 问题，详细查看 [SDK 状态码表]

4. IMKit SDK 支持了会话草稿消息，在会话列表中的排序功能

5. IMKit SDK 优化了刘海屏手机上位置共享界面样式

6. 优化了获取海量离线消息时的获取速度，提升了用户体验

7. 优化了视频文件压缩逻辑，提升了视频清晰度

8. IMLib SDK 优化了下载文件接口

2.9.24 Stable

发布日期：2019/09/17

问题修复：

1. 修复了无网状态下发送消息失败，拉取会话列表出现重复的问题

2. 修复了 CocoaPod 引入 SDK 加入 -all_load 参数导致重复定义 gifData 的报错

3. IMKit SDK UIWebView 替换为 WKWebView

4. 修复了快捷回复功能

功能优化：

1. 优化了每次进入相册选择页面，默认进入到相机胶卷页面

2.9.22 Dev

发布日期：2019/08/16

新增功能：

1. IMLib SDK 中新增获取服务端历史消息接口，支持获取指定时间前 N 条消息或后 N 条消息。

2. 对 iOS 13 公测版进行了适配

功能优化：

1. 优化了 IMKit SDK 图片消息展示规则

2. 实现了 IMKit SDK 会话列表分页加载

2.9.21 Dev

发布日期：2019/08/01

新增功能：

1. 支持了发送 GIF 图片消息功能

2. 会话界面中增加了消息快捷回复功能

问题修复：

1. 修复了文本消息点击链接的问题

2. 修复了加载更多消息时给每条消息添加时间逻辑错误的问题

3. 修复了 @功能中，用户信息的 userId 为 nil 的问题

2.9.20 Dev

发布日期：2019/07/15

问题修复：

1. 解决了小视频录制完成后预览过程中切入后台或者锁屏，再进入应用后的显示问题

2. 修复了音视频消息撤回的 BUG

功能优化：

1. 优化了清除用户信息缓存的问题

2. 优化了后台 SDK 断开连接的处理逻辑

2.9.19 Dev

发布日期：2019/06/28

新增功能：

1. 增加了高质量语音消息功能，需要注意高质量语音消息与旧版本 SDK 不兼容，旧版本 SDK 无法收听新的高质量语音消息。

问题修复：

1. 修复了发送小视频消息加载过程中，取消发送不成功的问题

功能优化：

1. 优化了发送图片消息，加载相册列表时性能问题

2.9.18 Stable

发布日期：2019/06/05

功能优化：

1. 优化了 SDK 会话加载性能

2.9.17 Stable

发布日期：2019/05/24

问题修复：

1. 修复了文本消息中包含表情和网址情况下的显示问题

2. 修复了 iPad 键盘收回问题

功能优化：

1. 优化了 IMKit SDK 中图片、视频消息自定义显示逻辑

2. 优化 SDK 内存问题

2.9.16 Dev

发布日期：2019/05/14

新增功能：

1. 增加了发送 VoIP 推送消息接口

功能优化：

1. 优化了 IM 重连的逻辑

2.9.15 Dev

发布日期：2019/04/23

新增功能：

1. 新增了获取不包含设置消息免打扰会话的，某些会话类型的未读消息数接口

问题修复：

1. 修复了文本消息中包含表情和网址情况下的显示问题

功能优化：

1. 优化了 IMKit SDK 中图片、视频消息自定义显示逻辑

2.9.14 Stable

发布日期：2019/04/11

问题修复：

1. 修复了英文环境下，视频通话页面摄像头按钮显示问题
2. 解决了图片消息占用内存的问题

功能优化：

2. 优化了音视频通讯引擎

2.9.13 Dev

发布日期：2019/03/29

问题修复：

1. 修复了 SDK 性增强了稳定性

功能优化：

1. 对 CallKit、CallLib SDK 中核心音视频引擎进行了升级，新版本与之前版本不兼容。详情请查看知识库文档

2.9.12 Dev

发布日期：2019/03/19

新增功能：

1. 增加了 “ 输入工具栏尺寸（高度）发生变化的回调” 方法

2. 增加图片下载 Token 验证

功能优化：

1. 优化了会话列表和聊天页面针对 iPad 横竖屏切换的适配

2. 优化了 IMKit SDK 国际化英语提示内容

2.9.11 Stable

发布日期：2019/02/22

问题修复：

1. 修复了录制小视频时的显示问题。

功能优化：

1. 聊天页面输入框针对 iPhoneX 适配。

2.9.10 Dev

发布日期：2019/01/25

问题修复：

1. 修复了获取 SDK 连接状态时，如用户被封禁则返回 31011

2. 修复了某种特殊情况点击消息中手机号不灵敏的问题

3. 修复了拍摄小视频未发送，切换到后台再开启应用后小视频被取消的问题

功能优化：

1. 优化了拉取离线消息较多时的内存占用

2. 更新了红包 SDK，对群会话中红包消息的用户昵称显示进行了优化

2.9.9 Dev

发布日期：2019/01/10

新增功能：

1. 增加了推送是否显示详情的接口

2. 增加了批量删除会话中指定服务器端消息接口

3. RCUserInfo 中添加扩展信息字段 extra

问题修复：

1. 修复了本地未提供用户信息时接收到系统消息后仍会弹出本地通知的问题

2. 修复了从相册选取图片资源时显示不全的问题

3. 修复了撤回一条消息后的通知栏的显示问题

2.9.8 Dev

发布日期：2018/11/22

新增功能：

1. 新增了在本地相册中选择视频发送功能，可发送视频文件时长不超过 2 分钟

问题修复：

1. 修复了第一次进入聊天页面语音发送按钮 UI 显示问题

2. 修复了聊天页面中第一次下拉刷新后，页面又滚动到了最底部的问题

功能优化：

1. 优化了录音按钮点击延迟录制的问题

2. 优化了会话列表中偶现头像闪动问题

3. 提升了实时音视频最大码率，提高了视频的清晰度

2.9.7 Stable

发布日期：2018/10/19

问题修复：

1. 修复了播放语音消息时开始录制，录制不成功，1 秒后自动中断的问题

功能优化：

1. 适配了 iPhone XS Max

2. 优化了会话页面加载消息性能，增强稳定性

2.9.6 Stable

发布日期：2018/09/06

问题修复：

1. 修复了发送 PNG 图片因缩略图大小超过限制无法发送的问题

2.9.5 Stable

发布日期：2018/08/25

新增功能：

1. 新增了融云动态表情 SDK

问题修复：

1. 修复了部分 BUG

2.9.4 Stable

发布日期：2018/08/21

问题修复：

1. SDK 多端互踢时增加设备信息提醒能力

2. 修复了上传数据 100% 时仍然无法正常发送的问题

3. 修复了发送图片时原图无法正常发送的问题

2.9.3 Stable

发布日期：2018/07/31

问题修复：

1. 修复了撤回消息时用户信息没有发送到 IM 的 BUG

功能优化：

1. 更新了输入框图片资源，支持输入框的横竖屏切换。

2. 优化了阅读回执功能，解决发送阅读回执时，发送成功以及发送失败方法没有回调的问题

3. 优化了正在输入状态消息状态显示的相关逻辑

2.9.2 Dev

发布日期：2018/07/06

新增功能：

1. 实时音视频增加录音、美颜、分辨率设置功能

问题修复：

1. 修复了在获取历史消息后，UI 显示的问题

功能优化：

1. 更新了 SDK 内置的 Emoji 表情

2. 优化了会话中下拉刷新的逻辑和交互逻辑

3. 优化了消息撤回逻辑

2.9.1 Stable

发布日期：2018/06/13

功能优化：

1. 支持了 WebP 格式图片

2. 修复了不停点击发送语音消息按钮，录音界面不消失问题

3. 优化了修改 isShowNotificationNumber 属性后刷新 UI

2.9.0 Dev

发布日期：2018/05/16

功能优化：

1. 实时音视频引擎升级，新老引警不兼容，老引擎仍然可以使用。详情请查看知识库文档

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】
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。

状态码	说明
407	未在对方的白名单中，消息发送失败。
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。
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
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。

状态码	说明
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
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
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

状态码	说明
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）
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
32061	连接被拒绝, SDK 会自动重连，开发者无须处理。
33001	未调用 init 初始化函数，请参考 初始化 文档
33002	数据库错误

状态码	说明
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	连接已经存在，不需要重复连接
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
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 ，value 长度不超过 64 ，单次设置扩展数量最大为 20，消息的扩展总数不能超过 300
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

状态码	说明
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
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。

客户端 API 参考 IMKit

更新时间:2024-08-30

以下是即时通讯界面库 IMKit 4.X 的 API 参考文档（appledoc）：

- IMKit（含 UI）

IMLib

以下是即时通讯基础能力库 IMLib 4.X 的 API 参考文档（appledoc）：

- IMLib（无 UI）

下列功能以不再继续提供服务，因此不再推荐使用 IMLib 中相关 API。API 文档中仅保留相关内容以作参考：

- 客服：`RongCustomerService*`
- 公众号：`RongPublicService*`
- 讨论组：`RongDiscussion*`