即时通信
IMLib / IMKit
Web 5.X

2024-08-30

即时通讯开发指导

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 »

平台兼容性

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

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

- **Electron 版本与架构支持**：

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

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

版本支持

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

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

即时通讯服务端

即时通讯服务端提供一套 API 接口与多种语言的开源 SDK。

服务端 API

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

> 前往融云即时通讯服务端 API 文档 · 集成必读 »

服务端 SDK

融云提供提供多个语言版本的开源服务端 SDK：

- server-sdk-java (GitHub) · (Gitee)

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

控制台

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

> 部分 IM 功能必须开通服务后方可使用。详见控制台服务管理页面。

即时通讯数据

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

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

除此之外，您还可以开通全量消息路由服务，实时将消息同步到您的业务服务器。

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

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

快速上手

更新时间:2024-08-30

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

> 提示
>
> - IMKit 不支持 H5 界面搭建，仅支持 Web 浏览器界面搭建。
> - 目前仅支持单群聊会话类型，暂不支持聊天室和超级群会话。
> - 目前仅支持 NPM 方式集成，暂不支持 CDN 方式集成。
> - 推荐使用 Vue3 集成。文档中示例与讲解基于 Vue 项目。

IMKit 简介

IMKit 基于 IMLib SDK 开发，在 IMLib 基础上提供会话和消息相关的 UI 界面，可用于快速实现 Web 聊天界面的搭建。IMKit 提供了易于使用的构建组件和 UI 交互。

- 会话列表界面：聊天界面中单、群聊会话的列表。
- 消息列表界面：聊天界面中收发的消息，即消息列表。
- 消息编辑界面：消息输入界面。

浏览器兼容说明

> 提示
>
> 当前仅支持 Chrome、Safari、Edge 浏览器，不支持移动端浏览器。

Chrome	Safari	Edge	微信 浏览器
68 +	13 +	100 +	×

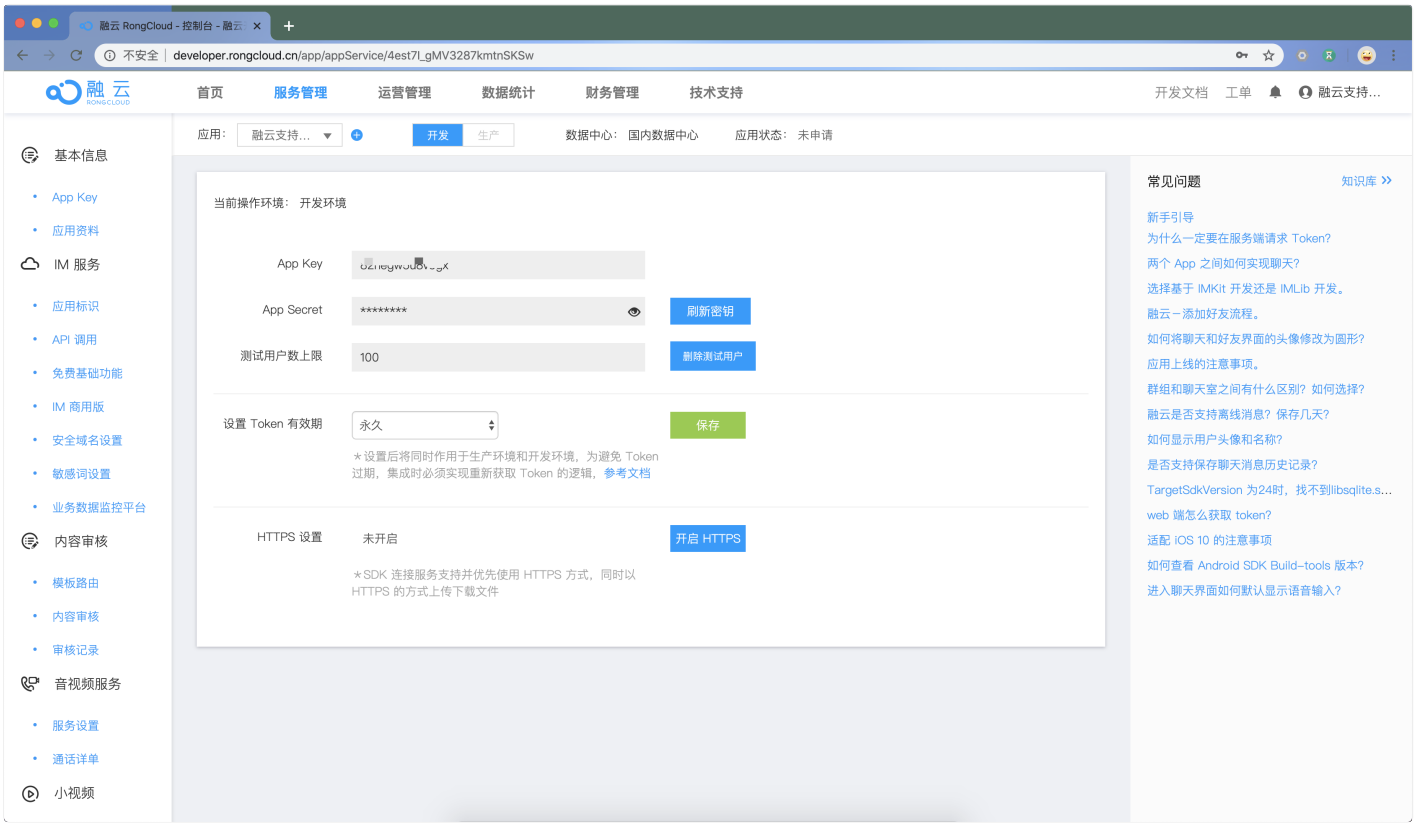
前置条件

创建融云开发者账号，获取 App Key。

控制台将自动为新账号创建一个应用。默认使用国内数据中心，默认提供开发环境。如果您已拥有融云开发者账户，您可以直接创建应用。

> 提示
>
> - 同一个应用的开发环境与生产环境提供不同的 App Key，两个环境之间数据隔离。
> - App Secret 用于生成数据签名，仅在请求融云服务端 API 接口时使用。本教程中暂不涉及。应用的 App Key /

Secret 是获取连接融云服务器身份凭证的必要条件，请注意不要泄露。



Demo 项目

融云 IMKit SDK 提供了一个基于 Vue 框架的 Demo 项目。

https://github.com/rongcloud-community/web-imkit-vue-demo

导入 SDK

> 提示
>
> IMKit 基于 IMLib SDK 开发，@rongcloud/imlib-next 包为 @rongcloud/imkit 提供了能力支持，开发者无需关心其作用。

本章节以 Vue 项目为例引导您快速接入。您可以通过 NPM 方式将 IMKit SDK 集成到您的 Web 项目中。详细说明可参考 导入 SDK。

```
npm install @rongcloud/engine --save
npm install @rongcloud/imlib-next --save
npm install @rongcloud/imkit --save
```

在 Vue 项目的 `main.js` 中引入模块。注意，此处使用 IMKit 中的 `defineCustomElements()` 方法引入了 IMKit 所有界面组件。

```
import * as RongIMLib from '@rongcloud/imlib-next'
// imkit 为核心模块
import { defineCustomElements, imkit } from '@rongcloud/imkit'

defineCustomElements()
```

在 vue.config.js 中添加 isCustomElement 配置，让 IMKit 的自定义标签跳过 Vue 组件解析。

```
// 此配置适用于 vue-cli5 中的 vue.config.js 配置，由于 vue-cli 不同版本配置可能存在差异，请根据您具体项目实现来增加 '自定义标签不处理' 配置
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
chainWebpack(config) {
config.module
.rule('vue')
.use('vue-loader')
.tap(options => {
options.compilerOptions = {
...options.compilerOptions,
isCustomElement: tag => {
return ['conversation-list', 'message-list', 'message-editor'].indexOf(tag) !== -1
}
}
return options
})
.end()
}
})
```

初始化 SDK

初始化 IMKit 需要传递两个必传参数 service 和 libOption。

- `service`：用于获取用户侧的用户、群组信息展示在页面相关位置。详见用户信息。
- `libOption`：IMKit 依赖 IMLib，因此需要传入 IMLib 的初始化配置信息。详见 IMLib 初始化参数说明。

IMKit 依赖即时通讯能力库 IMLib SDK，建议在初始化 IMKit 时同时获取 RongIMLib 实例对象。

```
// 接入时需要将 '请更换您应用的 appkey' 替换为您的应用的 appkey
let libOption = {appkey: '请更换您应用的 appkey'}
let custom_service = {
// 获取用户详情
getUserProfile: (userId) => {
// 需要通过 userId 向应用服务器获取 user 信息，拼接成如下格式
// 注意：userInfo 的 Key 不可修改
const userInfo = {
id: userId,
name: "用户姓名",
portraitUri: "用户头像 URI",
displayName: "别名"
};
return Promise.resolve(userInfo);
},

// 获取会话详情
getConversationProfile: (conversations) => {
// SDK 返回 conversations 为会话列表，可根据返回的 conversations 向应用服务器请求会话详情信息。
// 请根据具体 conversation 信息匹配 name、portraitUri 拼接到 conversationInfo 信息中。

// 方式 1 为了减少请求次数，可以批量请求（推荐），需服务端支持批量请求接口
return new Promise((resolve) => {
// 请将 mockBatchFetchGroupInfo 方法替换成真实请求方法
mockBatchFetchGroupInfo(conversations).then(res => {
const list = conversations.map(item => {
// 代码示例，可根据真实接口返回的数据处理
const info = res.find(con => con.targetId === item.targetId)
```

```
return {
...item,
name: info.name,
portraitUri: info.portraitUri,
// 如果是群组会话，则需要群组成员数量
memberCount: con.conversationType === RongIMLib.ConversationType.GROUP ? info.memberCount : 0
}
})
resolve(list)
})
})

// 方式 2 可以通过 forEach 方式遍历请求
const promises = [];
conversations.forEach((conversation) => {
promises.push(new Promise(resolve => {
// 请将 mockFetchGroupInfo 方法替换成真实请求方法
mockFetchGroupInfo(conversation).then((res) => {
resolve({
...conversation,
name: res.name,
portraitUri: res.portraitUri,
// 如果是群组会话，则需要群组成员数量
memberCount: conversation.conversationType === RongIMLib.ConversationType.GROUP ? res.memberCount : 0
})
})
}))
});
return Promise.all(promises);
},

// 获取群组详情
getGroupMembers: (conversation) => {
// 通过 conversation 的 targetid 获取群组成员信息
// groupMembers 为群组成员 list，需要构建成对象数组。
// 特别注意：如果传递的群组成员信息不准确会影响 @ 信息的发送和群组成员昵称的展示
const groupMembers = [
{
id: `【成员】成员 ID`,
name: `【成员】name`,
portraitUri: `【成员】头像 URI`,
},
];
return Promise.resolve(groupMembers);
},
};

// 应用初始化以获取 RongIMLib 实例对象，请务必保证此过程只被执行一次
RongIMLib.init(libOption);

// 初始化
imkit.init({
appkey: '请填入 appkey'
service: custom_service,
libOption: libOption,
});
```

以上提供了一个简化的初始化示例，关于初始化的更多配置请参见初始化。

获取用户 Token

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

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

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

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

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

建立 IM 连接

拿到 token 后可在 `main.js` 中调用 `connect` 方法进行连接融云

> 提示
>
> **特别提醒：connect 方法需要等待页面加载完成后进行调用，例如：vue 的 mounted 声明周期中。**

```
/**
* 请替换 'token' 为上一步拿到的测试 token
*/
RongIMLib.connect('当前用户 TOKEN').then((res) => {
console.info('连接结果打印：', res);
// 加载会话列表 CoreEvent 可通过 import { CoreEvent } from '@rongcloud/imkit' 获取
imkit.emit(CoreEvent.CONVERSATION, true);
})
```

SDK 已实现自动重连机制，请参见连接。

UI 界面

IMKit SDK 已默认提供会话列表页面和消息列表等页面。客户端用户在会话列表页面可查看到当前所有的聊天会话，在点击某一个会话可查看到该会话的消息列表和消息编辑区。

> 提示
>
> **引入 UI 界面必读：**
>
> 1. base-size: **用来设置会话列表、消息列表、消息编辑界面的 fontSize，默认 16px。**
> 2. **为保证会话列表和消息列表大小展示一致，建议 `conversation-list`、`message-list` 组件设置的 base-size 值保持一致。**
> 3. `message-editor` **中 base-size 设置的仅是字体图标和发送按钮文字大小，并非输入框输入文字大小。**

引入标签

- `<conversation-list/>`：展示会话列表界面，即聊天界面中单、群聊会话的列表。
- `<message-list/>`：展示消息列表界面，即聊天界面中收发的消息。
- `<message-editor/>`：展示消息编辑界面，即消息输入界面。

> 提示
>
> **特别提醒：组件使用必须包裹在 div 内，div 需要设置组件的宽高等样式信息，如不设置会导致会话列表或消息列表无法滚动，消息编辑组件展示位置异常。**

```
<template>
<div class="chat">
<div class="chat-conversation">
<conversation-list ref="conversationList" base-size="10px" />
</div>
<div class="chat-message">
<div class="chat-massage-list">
<message-list ref="messageList" base-size="10px"></message-list>
</div>
<div class="chat-massage-editor">
<message-editor ref="messageEditor" base-size="10px"></message-editor>
</div>
</div>
</div>
</template>

<style scoped>
.chat {
width: 100%;
height: 100%;
}

.chat-conversation {
float: left;
width: 30vw;
height: 100%;
}

.chat-message {
float: left;
width: 70vw;
height: 100%;
}

.chat-massage-list {
height: calc(100% - 220px);
}

.chat-massage-editor {
height: 220px;
}

</style>
```

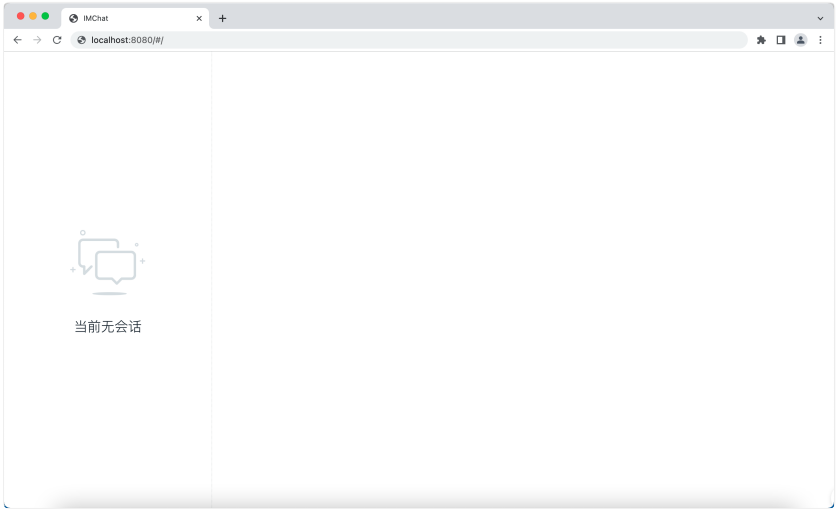
实现事件监听

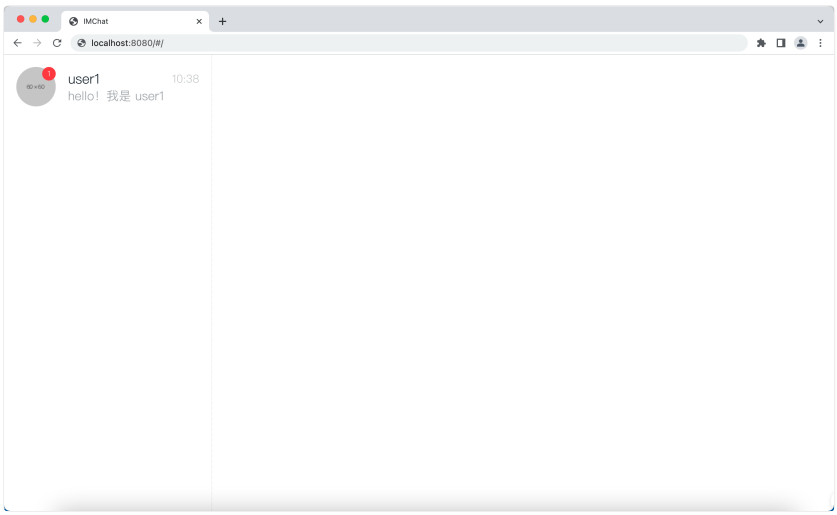
```
export default {
name: 'Chat',
mounted() {
// 获取到标签元素
const conversationList = this.$refs.conversationList;
// 获取到标签元素
const messageList = this.$refs.messageList;

/**
* 添加点击会话监听
*/
conversationList.addEventListener("tapConversation", this.handleTapConversation);

/**
* 添加删除会话监听
*/
conversationList.addEventListener("deleteConversation", this.handleDeleteConversation);

/**
* 添加点击消息监听
* 注意：仅有点击图片消息和文件消息会触发监听
*/
messageList.addEventListener("tapMessage", this.handleTapMessage);
},
methods: {
handleTapConversation(){
console.info('handleTapConversation')
},
handleDeleteConversation(){
console.info('handleDeleteConversation')
},
handleTapMessage(e){
const data = e.detail;
// 处理点击查看大图或文件消息下载等功能
console.log("点击消息触发监听:",data)
}
}
}
```

UI 界面简介

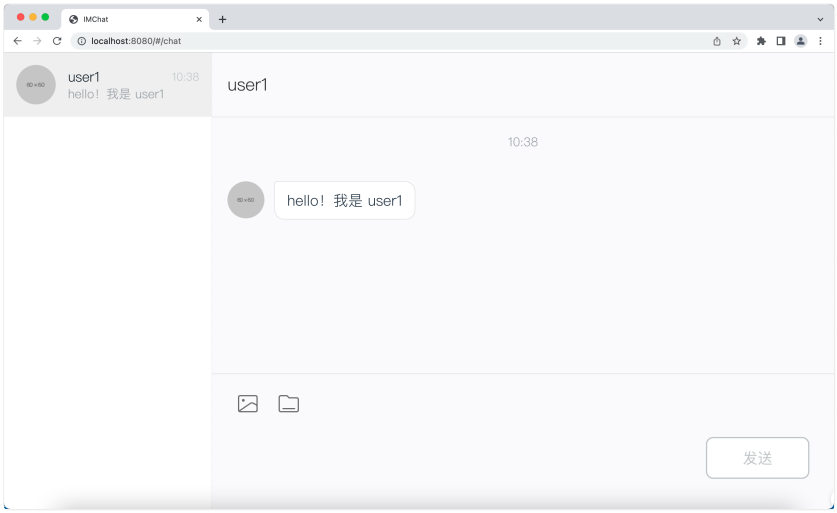
首次登录成功后，如该用户无任何会话和消息，会展示空会话列表，客户端接收到消息后，会自动在会话列表页面展示新会话。

下图展示了 IMKit SDK 默认提供的会话列表页面。以下直接以默认会话列表为例。





点击会话列表中的会话，将进入会话页面。在会话页面可发送消息。



显示昵称及头像信息，您需要在初始化 init 方法中传递 `service` 参数构造用户信息。IMKit 通过您提供的信息显示的用户资料数据。详情请参见用户信息。

测试收发消息

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

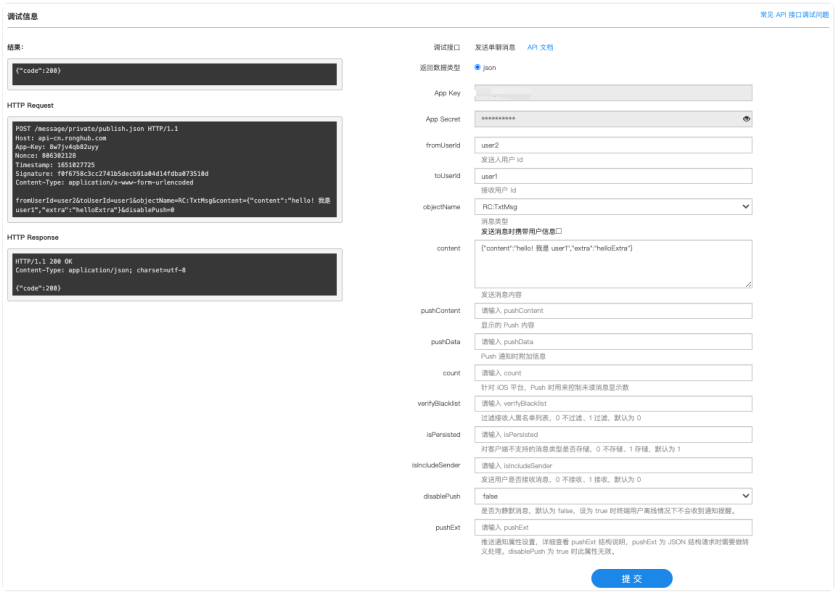
> 提示
>
> - 融云服务器提供消息发送能力，消息发送过程中默认不会做任何权限校验。
> - 好友关系由开发者的应用服务器自行维护。

在实际业务运行过程中，应用客户端可以通过用户 ID、群聊会话 ID、或聊天室 ID 等接收消息。

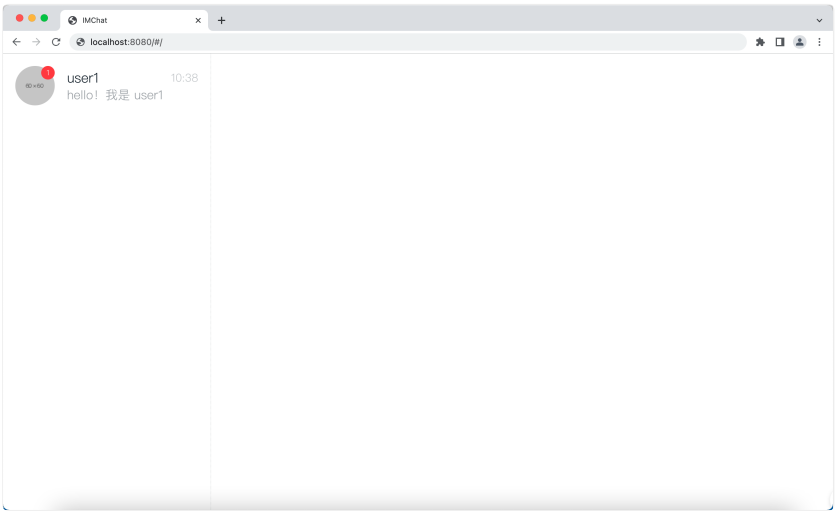
在本教程中，为了快速体验和测试 SDK，我们从控制台「北极星」开发者工具箱 IM Server API 调试 页面向当前登录的用户发送一条文本消息，模拟单聊会话。

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

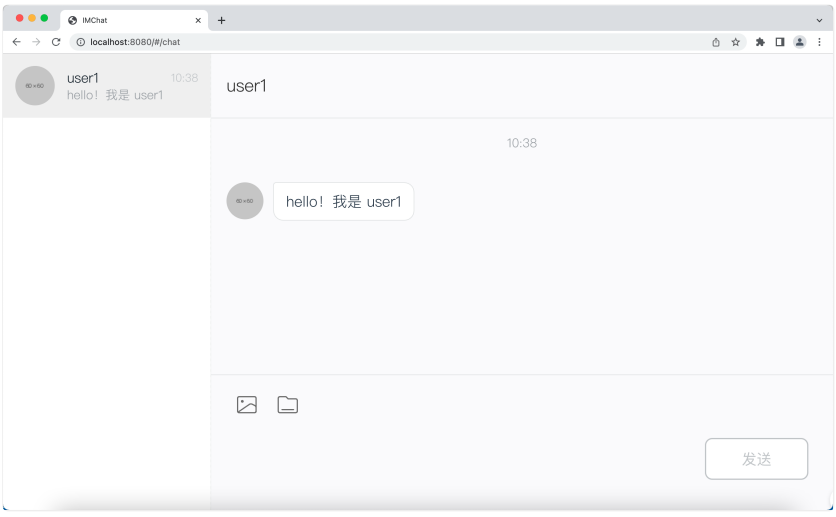
以下模拟了从 UserId 为 user2 的用户向 UserId 为 user1 的用户发送一条文本消息。



3. 客户端接收到消息后，自动在会话列表页面展示新的单聊会话。



4. 点击会话，即可进入消息列表页面，发送消息。



后续步骤

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

导入 SDK 依赖安装

更新时间:2024-08-30

融云目前仅提供通过 NPM 方式将 IMKit SDK 集成到您的 Web 项目中，暂不支持 CDN 方式集成

> 提示
>
> IMKit 基于 IMLib SDK 开发，@rongcloud/imlib-next 包为 @rongcloud/imkit 提供了能力支持，开发者无需关心其作用。

```
npm install @rongcloud/engine --save
npm install @rongcloud/imlib-next --save
npm install @rongcloud/imkit --save
```

导入 SDK

> 提示
>
> - 新集成客户或升级 IMKit 的客户，建议使用 IMKit 5.6.1 及之后版本。
> - IMKit 版本 < 5.6.1，可适配的 IMLib SDK 最高版本为 5.5.5。如遇到问题，请注意限制 IMLib SDK 版本，例如，`npm install @rongcloud/engine@5.5.5 @rongcloud/imlib-next@5.5.5 -S`。

依赖安装完成，导入 IMKit 核心模块，以及由 IMKit 提供的自定义元素组件。目前 IMKit 以自定义元素方式提供组件，包括 `<conversation-list/>`、`<message-list/>`、`<message-editor/>`。

```
import * as RongIMLib from '@rongcloud/imlib-next'
// imkit 为核心模块
import { defineCustomElements, imkit } from '@rongcloud/imkit'

// 引入自定义组件
defineCustomElements()
```

Vue 项目配置示例

Vue 项目需要添加过滤自定义组件配置。本文仅提供了 vue-cli5 与 webpack 的配置示例，仅供参考。具体配置可根据您的项目构建进行修改。

vue-cli5 中 vue.config.js 添加配置

```
// 此配置适用于 vue-cli5 中的 vue.config.js 配置，由于 vue-cli 不同版本配置可能存在差异，请根据您具体项目实现来增加 '自定义标签不处理' 配置
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
chainWebpack(config) {
config.module
.rule('vue')
.use('vue-loader')
.tap(options => {
options.compilerOptions = {
...options.compilerOptions,
isCustomElement: tag => {
return ['conversation-list', 'message-list', 'message-editor'].indexOf(tag) !== -1
}
}
return options
})
.end()
}
})
```

webpack.config.js 添加配置

```
module: {
rules: [
{
test: /\.vue$/,
loader: "vue-loader",
options: {
compilerOptions: {
isCustomElement: (tag) =>
["conversation-list", "message-list", "message-editor"].includes(tag),
},
},
},
]
}
```

初始化

更新时间:2024-08-30

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

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

准备 App Key

您必须拥有正确的 App Key，才能进行初始化。

您可以控制台，查看您已创建的各个应用的 App Key。

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

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

海外数据中心

- 如果您使用海外数据中心，必须在 IMKit 的初始化参数 `LibOption` 参数中将 SDK 连接的服务地址为海外数据中心地址。否则 SDK 默认连接中国国内数据中心服务地址。详细说明请参见配置海外数据中心服务地址。

初始化 SDK

IMKit 初始化流程如下：

1. 在 IMKit 初始化前，请先初始化 RongIMLib，获取 RongIMLib 实例。此处需传入 IMLib 的初始化配置 IInitOption。详见 IMLib 初始化参数说明。

2. 实现应用层向 IMKit SDK 传入用户、群组、会话的名称和头像等信息的方法，并在初始化参数 `service` 中传入，否则 IMKit 无法展示头像和昵称。
3. 调用 IMKit SDK 的 `init` 方法进行初始化，在应用生命周期内，只需要调用一次。

以下示例仅简单呈现了 IMKit 初始化的流程，该示例单独在 `custom_service.js` 中实现了提供用户信息、群组信息的方法，相关代码可参见用户信息。

```
// 用于向 IMKit SDK 提供用户信息、群组信息等数据
import {custom_service} from './custom_service.js'

// IMKit 初始化参数 appkey，从控制台获取
const yourAppKey = "Your own App Key";

// IMLib 的初始化配置，appkey 为必填参数，其他字段参见 IInitOption
let libOption = {appkey: yourAppKey};

// 获取 RongIMLib 实例对象，请务必保证此过程只被执行一次
RongIMLib.init(libOption);

imkit.init({
appkey: yourAppKey,
service: custom_service,
libOption: libOption
});
```

参数	类型	必填	说明
appkey	String	是	从控制台获取的应用的唯一标识。请注意区分生产/开发环境的 App Key。
libOption	IInitOption	是	IMKit 依赖 IMLib，此处需传入 IMLib 的初始化配置 `IInitOption`。详见 IMLib 初始化参数说明。
service	Object	是	获取用户信息、群组信息等接口配置。IMKit 需要在 UI 上展示用户和群组的头像、昵称等信息时，需要应用层（App）提供相关数据。您需要实现向 SDK 提供用户、群组、会话的 name 和 portraitUri 信息的相关方法。详见用户信息。
conversationPullCount	Number	否	获取会话数量，默认值 30，数值范围 [1 - 1000]。
customMessage	Object	否	设置自定义消息展示。详见展示自定义消息。
customIntercept	Object	否	会话过滤回调，可设置会话是否展示，返回值中 true 为不展示，false 为正常展示。详见会话过滤。

连接

更新时间:2024-08-30

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

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

前置条件

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

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

连接聊天服务器

请根据应用的业务需求与设计，自行决定合适的时机向融云聊天服务器发起连接请求。

请注意以事项：

- 必须在 SDK 初始化之后，调用 `connect()` 方法进行连接。
- SDK 本身有重连机制，无需手动多次调用连接。
- 应用调用了断开连接相关方法后，SDK 不再进行重连。

在页面加载完成后，使用 connect 方法建立连接。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。

> 提示
>
> **特别提醒：**connect 方法需要等待页面加载完成后进行调用。

从 5.6.1 版本开始，废弃 IMKit 的 connect 方法。请参考下方示例，使用 IMLib SDK 提供的 connect 方法。

```
// 请替换真实 Token
RongIMLib.connect('当前用户 TOKEN').then((res) => {
console.info('连接结果打印：', res);
// 加载会话列表 CoreEvent 可通过 import { CoreEvent } from '@rongcloud/imkit' 获取
imkit.emit(CoreEvent.CONVERSATION, true);
})
```

断开连接

更新时间:2024-08-30

App 用户退出登录，切换账号时，可以断开 IM 连接。

从 5.6.1 版本开始，废弃 IMKit 的 disconnect 方法。请参考下方示例，使用 IMLib SDK 提供的 disconnect 方法。

```
RongIMLib.disconnect().then(() => {
console.log('成功断开')
})
```

多端同时在线

更新时间:2024-08-30

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

> 默认多端设备之间不会进行消息同步。如有需要，请开通多设备消息同步服务。

多端登录限制说明

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

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

多设备消息同步

更新时间:2024-08-30

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

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

适用场景

在融云即时通讯业务中，多设备消息同步适用于以下情况：

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

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

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

支持在多设备间同步的消息

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

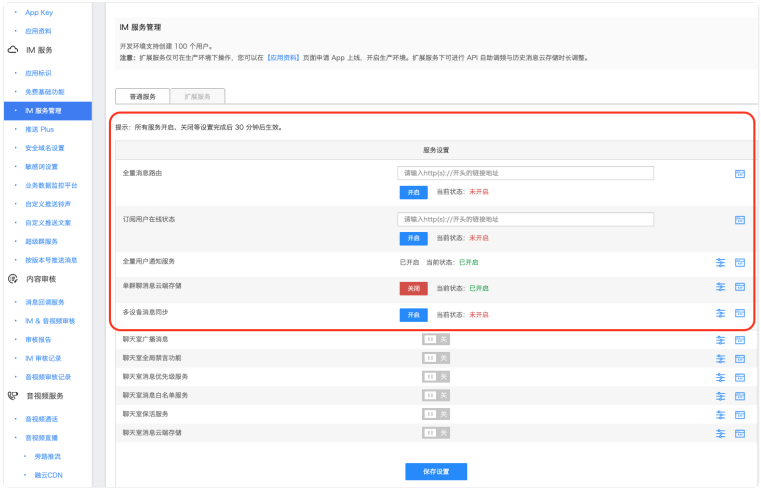
以下情况均属于状态消息：

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

开通服务

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

> 服务开启、关闭设置完成后 30 分钟内生效。



对其他功能或业务的影响

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

对离线补偿的影响

控制台的多设备消息同步服务包含了融云服务端离线补偿机制[?]的开关。

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

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

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

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

对 Web 平台连接数的影响

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

对撤回消息的影响

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

对服务端 API 发送消息的影响

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

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

用户概述

更新时间:2024-08-30

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

注册用户

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

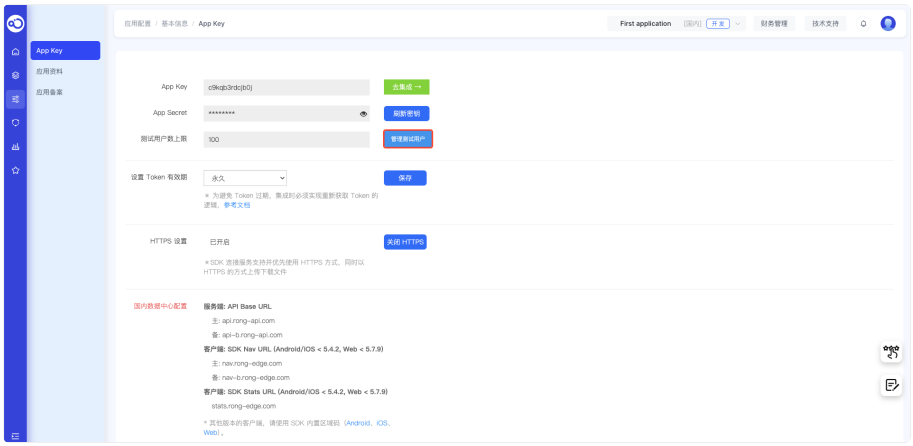
应用客户端必须持有有效 Token，才能成功连接到融云服务端，使用融云即时通讯服务。当 App 客户端用户向服务器发送登录请求时，服务器会查询数据库以检查连接请求是否匹配。

注册用户数限制

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

删除用户

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



注销用户

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

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

仅 IM Server API 提供上述能力。

用户信息

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

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

好友关系

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

展示用户信息

更新时间:2024-08-30

在 IMKit UI 上展示用户信息数据需要由应用程序提供。本文描述了应用程序如何为 IMKit SDK 提供用户信息数据，具体包含：

- 用户头像、昵称
- 群组名称、头像
- 群组成员数量
- 群组内成员的头像、昵称

> 提示
>
> 融云服务端不向 SDK 提供用户与群组信息托管服务。因此 SDK 所需要的用户信息必须由应用开发者主动从 App 服务端获取，并提供给 SDK。

实现向 SDK 提供用户信息的方法

在项目中创建 `custom_service.js` 文件，在适当的调用位置导入 `custom_service.js` 文件，供 IMKit 的初始化时在 `service` 中传入。

```
imkit.init({
appkey: yourAppKey,
service: custom_service,
libOption: libOption
});
```

应用程序需要在 `custom_service.js` 中实现以下方法：

参数	类型	必填	说明
getUserProfile	Function	是	获取用户的昵称、头像。支持异步返回用户信息对象。
getConversationProfile	Function	是	获取会话的昵称、头像。支持异步返回会话信息。
getGroupMembers	Function	否	获取群组成员的昵称、头像。支持异步返回群组信息，返回值为群成员对象数组，数组对象包含字段请参考 `getGroupMembers` 返回值说明。

> 提示
>
> 返回值中的属性名称不可自定义或者修改。必填返回值属性不可缺失，如缺失会引起报错。详见下文返回值说明。

在根据 SDK 返回的会话列表获取会话信息（`conversationInfo`）时，建议您实现向 App 服务端批量请求数据的方法，以减少请求次数。`custom_service.js` 的代码示例：

```
export default {
```

```
// 获取用户详情
getUserProfile: (userId) => {
// 需要通过 userId 向应用服务器获取 user 信息，拼接成如下格式
// 注意：userInfo 的 Key 不可修改
const userInfo = {
id: userId,
name: "用户姓名",
portraitUri: "用户头像 URI",
};
return Promise.resolve(userInfo);
},

// 获取会话详情
getConversationProfile: (conversations) => {
// SDK 返回 conversations 为会话列表，可根据返回的 conversations 向应用服务器请求会话详情信息。
// 请根据具体 conversation 信息匹配 name、portraitUri 拼接到 conversationInfo 信息中。

// 方式 1 为了减少请求次数，可以批量请求（推荐），需服务端支持批量请求接口
return new Promise((resolve) => {
// 请将 mockBatchFetchGroupInfo 方法替换成真实请求方法
mockBatchFetchGroupInfo(conversations).then(res => {
const list = conversations.map(item => {
// 代码示例，可根据真实接口返回的数据处理
const info = res.find(con => con.targetId === item.targetId)

return {
...item,
name: info.name,
portraitUri: info.portraitUri,
// 如果是群组会话，则需要群组成员数量
memberCount: con.conversationType === RongIMLib.ConversationType.GROUP ? info.memberCount : 0
}
})
resolve(list)
})
})

// 方式 2 可以通过 forEach 方式遍历请求
const promises = [];
conversations.forEach((conversation) => {
promises.push(new Promise(resolve => {
// 请将 mockFetchGroupInfo 方法替换成真实请求方法
mockFetchGroupInfo(conversation).then((res) => {
resolve({
...conversation,
name: res.name,
portraitUri: res.portraitUri,
// 如果是群组会话，则需要群组成员数量
memberCount: conversation.conversationType === RongIMLib.ConversationType.GROUP ? res.memberCount : 0
})
})
}))
});
return Promise.all(promises);
},

// 获取群组详情
getGroupMembers: (conversation) => {
// 通过 conversation 的 targetid 获取群组成员信息
// groupMembers 为群组成员 list，需要构建成对象数组。
// 特别注意：如果传递的群组成员信息不准确会影响 @ 信息的发送和群组成员昵称的展示
const groupMembers = [
{
id: `【成员】成员 ID`,
name: `【成员】name`,
portraitUri: `【成员】头像 URI`,
},
];
return Promise.resolve(groupMembers);
```

```
},
};
```

返回值说明

- **getUserProfile** 返回值说明

参数	类型	必填	说明
id	string	是	用户 UserId
name	string	否	用户昵称
portraitUri	string	否	用户头像 URI

- **getConversationProfile** 返回值说明

参数	类型	必填	说明
conversationType	ConversationType	是	会话类型，可在参数 conversation 获取
targetId	string	是	目标 Id，可在参数 conversation 获取
name	string	否	会话昵称
portraitUri	string	否	会话头像 URI
memberCount	number	否	群成员数量，此属性仅对群组有效

- **getGroupMembers** 返回值说明

参数	类型	必填	说明
id	string	是	群成员 UserId
name	string	否	群成员昵称
portraitUri	string	否	群成员头像 URI

修改用户信息

更新时间:2024-08-30

融云服务端不提供用户信息托管维护服务。如果应用程序需要修改在 IMKit SDK UI 显示的用户及群组的头像、昵称等信息，必须主动调用 IMKit 提供的方法更新信息。

本文详细描述了如何修改 IMKit SDK 设置的用户、群组的头像、昵称等信息。

更新当前登录用户昵称和头像

updateUserProfile 用于更新用户的昵称、头像。支持异步返回用户信息对象。

```
const info = {
id: '当前用户 id', // 可在 connect 连接成功后获取
name: '用户名称',
portraitUri: '用户头像 URL'
}
imkit.updateUserProfile(info)
```

info 参数说明

参数	类型	必填	说明
id	string	是	用户 UserId
name	string	是	用户昵称
portraitUri	string	否	用户头像 URI

更新当前页面会话的信息

> 提示
>
> 仅支持修改当前页面上会话的信息。修改当前页面不存在或未加载的会话则更新无效。当前没有选中会话则仅修改内存态对应会话信息。

updateConversationProfile 用于修改当前页面中会话的昵称、头像、群成员数量，适用于单聊会话、群聊会话。注意，群成员数量（memberCount）仅在群组会话时生效。

```
const conversation = {
// 会话类型，单聊为:PRIVATE，群聊为:GROUP，可从 conversation 中获取
conversationType: ConversationType.PRIVATE,
// 群聊为群组 Id，单聊为 userId，可从 conversation 中获取
targetId: '目标 Id'
}

const info = {
conversationType: 1,
targetId: '目标 Id', // 群组为群组 Id,单聊为 userId
name: '用户名称',
portraitUri: '用户头像 URL'
}
imkit.updateConversationProfile(conversation, info)
```

conversation 属性说明

名称	参数	必传	说明
conversationType	string	是	会话类型
targetId	string	是	目标 Id

info 参数说明

参数	类型	必填	说明
conversationType	ConversationType	是	会话类型，可在参数 `conversation` 中获取。
targetId	string	是	目标 ID，可在参数 `conversation` 中获取。
name	string	否	会话昵称
portraitUri	string	否	会话头像 URI
memberCount	number	否	群成员数量，此属性仅对群组有效。

修改全部群成员信息

`updateGroupMembers` 用于修改群聊会话的全部群组成员的昵称、头像。仅支持全量修改群组信息。

- 如果已选中待修改的群聊会话，修改当前选中的群聊会话的成员信息，SDK 会刷新页面，显示修改后的数据。
- 如果修改时没有任何群聊会话被选中，则修改群聊会话成员信息不会触发页面刷新。修改后的数据会保存在内存中，只要内存数据未丢失，后续切换到该会话时不需要再次修改，页面直接展示最新数据。

`updateGroupMembers` 仅支持全量修改群组成员信息，调用必须传递全部群成员信息。如果仅需修改部分数据，调用时也必须传递全部群成员信息。假设群组中有 `groupMemberUserId1` 和 `groupMemberUserId2` 两位用户：

```
import { ConversationType } from '@rongcloud/imlib-next'
const conversation = {
// 会话类型，群聊为:GROUP，可从 conversation 中获取
conversationType: ConversationType.GROUP,
// 群聊为群组 Id，单聊为 userId，可从 conversation 中获取
targetId: '目标 Id'
}

const info = [{
id: 'groupMemberUserId1',
name: '成员昵称',
portraitUri: '成员头像',
},
{
id: 'groupMemberUserId2',
name: '成员昵称',
portraitUri: '成员头像',
}
]
imkit.updateGroupMembers(conversation, info)
```

参数说明

参数	类型	必填	说明
conversation	Object	是	会话信息
info	Array	是	全量群成员信息

- **info 属性说明**

参数	类型	必填	说明
id	string	是	群成员 UserId
name	string	否	群成员昵称
portraitUri	string	否	群成员头像 URI

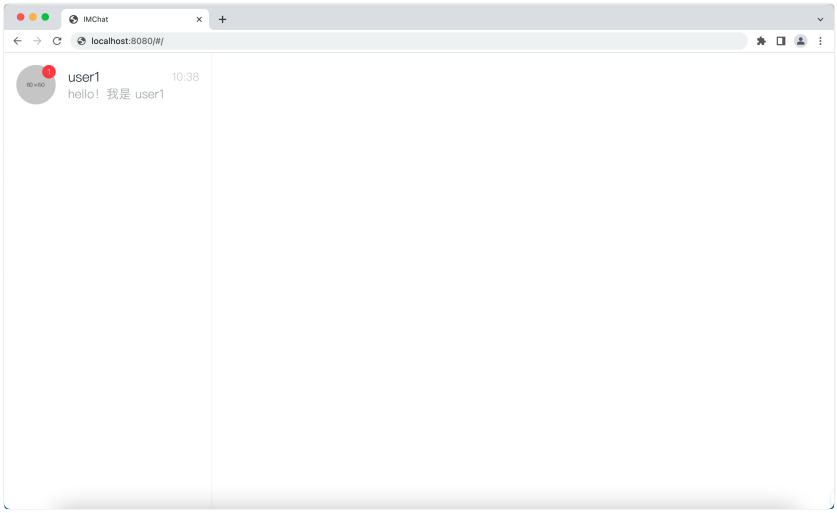
- **conversation 属性说明**

名称	参数	必传	说明
conversationType	string	是	会话类型
targetId	string	是	目标 ID

会话列表组件

更新时间:2024-08-30

会话列表组件 conversation-list 显示当前用户当前用户的会话列表。客户端应用程序必须连接到融云服务器，才可以显示会话列表。



> 提示
>
> Web 端不具备持久化的数据存储能力，无法在本地持久化存储历史消息记录与会话列表，因此需要从融云服务端获取数据。从远端获取单群聊历史消息要求您已在控制台 IM 服务管理 页面为当前使用的 App Key 开启单群聊消息云端存储服务。**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。开通 30 分钟后生效，如果 App Key 未开通，则 IMKit 无法从融云服务端获取会话。

组件用法

导入 IMKit 时，请通过 IMKit 的 defineCustomElements() 统一引入所有组件。Vue 项目需要添加过滤自定义组件配置，详见导入 SDK。

```
import * as RongIMLib from '@rongcloud/imlib-next'
// imkit 为核心模块
import { defineCustomElements, imkit } from '@rongcloud/imkit'

defineCustomElements()
```

按上述方式引入组件后，您可以直接使用 conversation-list。

属性列表

下面列出了 conversation-list 组件的属性。

属性名称	类型	描述
baseSize	string	设置组件大小，接受以 px 为单位的数字或百分比（根据父级 font-size 百分比），例如 18px、62.5%。默认值：16px。
customMenu	{ name: (conversation: IConversationOption) => string; callback: (conversation: IConversationOption) => any; }[]	应用程序在会话右键菜单中添加的自定义功能。要求 IMKit SDK ≧ 5.3.0。
disableMenu	DisabledConversationontextMenu[]	应用程序在会话右键菜单中隐藏的 IMKit 预置功能。要求 IMKit SDK ≧ 5.5.1。
hideNotificUnreadCount	boolean	是否隐藏未读数数字，只显示小红点。该功能只针对免打扰状态下的会话生效。非免打扰状态下的会话正常显示会话未读数。要求 IMKit SDK ≧ 5.6.4。详见未读消息数。

组件样式

> 提示
>
> 如不设置 div 样式，会导致会话列表无法滚动或其他异常。

使用 conversation-list 组件时，必须将组件包裹在 div 元素，并通过 div 的样式控制组件的宽高等样式。

```
<template>
<div class="chat-conversation">
<conversation-list ref="conversationList" base-size="10px" />
</div>
</template>

//

<style scoped>
.chat-conversation {
float: left;
width: 30vw;
height: 100%;
}
</style>
```

事件处理

事件名称	返回值	描述
updateConversation		会话条目被更新。
tapConversation		会话条目被点击。
deleteConversation		会话条目被删除。

> 提示
>
> 以下示例代码以 Vue 框架为例。

```
<template>
<div class="chat-conversation">
<conversation-list ref="conversationList" base-size="10px" />
</div>
</template>

<script>
// Vue 代码
export default {
mounted() {
const conversationList = this.$refs.conversationList;

//添加点击会话监听
conversationList.addEventListener(
"tapConversation",
this.handleTapConversation //回调处理函数
);
//添加删除会话监听
conversationList.addEventListener(
"deleteConversation",
this.handleDeleteConversation //回调处理函数
)
},
beforeUnmount() {
// 注意：需要 removeEventListener 防止多次绑定造成异常
const conversationList = this.$refs.conversationList;

conversationList.removeEventListener("tapConversation", this.handleTapConversation);

conversationList.removeEventListener("deleteConversation", this.handleDeleteConversation);
},
methods: {
handleTapConversation() {
//处理点击会话后的操作
console.info('处理点击会话后的操作');
},
handleDeleteConversation(){
//处理删除会话后的操作
console.info('处理点击会话后的操作');
}
}
}
</script>

<style scoped>
.chat-conversation {
float: left;
width: 30vw;
height: 100%;
}
</style>
```

自定义会话右键菜单

IMKit SDK 会话右键菜单默认已提供会话置顶、免打扰、删除会话等功能。为满足更多客户个性化业务需求，IMKit 支持自定义会话右键菜单。

- 添加自定义功能按钮，您可以在业务侧自行实现并添加拉黑、删除好友等功能选项。
- 移除 IMKit 预置功能按钮，包括置顶、删除、免打扰功能。

添加自定义会话右键菜单选项

> 提示
>
> IMKit 5.3.0 版本开始支持该功能。

应用程序可以使用 conversationList 组件的 conversationCustomMenu 属性添加自定义功能选项。

```
// 拿到会话列表组件
const conversationList = this.$refs.conversationList;

const conversationCustomMenu = [{
name: function (conversation) {
return '自定义按钮'
},
callback: function (conversation) {
// 通过 callback 拿到点击动作后进行业务处理
console.info('点击自定义按钮拿到会话:', JSON.stringify(conversation))
}
}]

conversationList.customMenu = conversationCustomMenu
```

隐藏预置会话菜单按钮

> 提示
>
> IMKit 5.5.1 版本开始支持该功能。

应用程序可以使用 conversationList 组件的 disableMenu 属性隐藏 IMKit SDK 会话右键菜单默认已提供的会话置顶、免打扰、删除会话功能按钮。

> 提示
>
> 置顶功能与免打扰功能均提供的一组按钮，设置隐藏功能按钮后，所有相关按钮都会被隐藏。例如，设置隐藏会话免打扰功能按钮后，IMKit 页面上将隐藏“免打扰”和“取消免打扰”按钮。

```
// 引入会话按钮枚举
import { DisabledConversationontextMenu } from "@rongcloud/imkit";
// 拿到会话列表组件
const conversationList = this.$refs.conversationList;

// 示例：隐藏会话置顶、会话免打扰、会话删除功能按钮，
const disableMenu =
[DisabledConversationontextMenu.Top,DisabledConversationontextMenu.Notification,DisabledConversationontextMenu
.Delete];

conversationList.disableMenu = disableMenu;
```

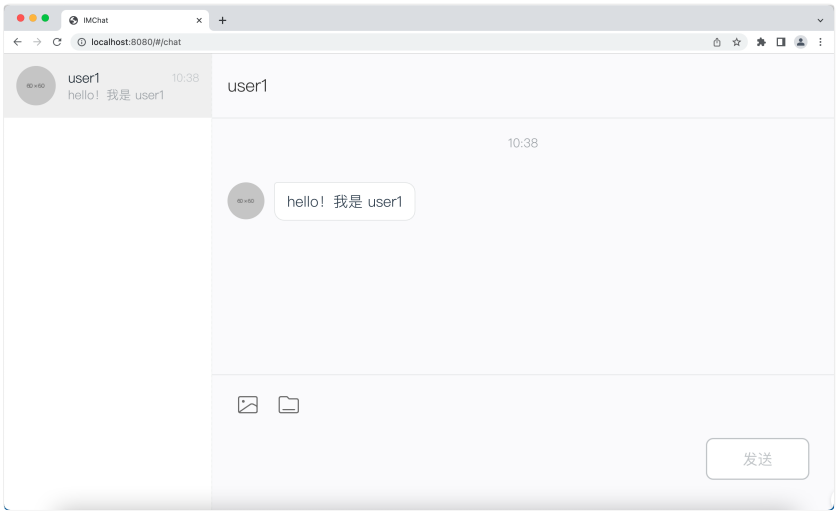
消息列表组件

更新时间:2024-08-30

消息列表组件 `message-list` 负责显示用户收发的消息。目前支持展示的消息类型包括：

- **普通消息**：文本、引用
- **媒体消息（支持自定义点击事件）**：图片、文件、小视频、GIF 图片、富文本、高质量语音。图片消息默认展示缩略图，如需查看大图可通过监听 `tapMessage` 事件来实现。

点击会话列表中的会话，将进入会话页面。在会话页面可发送消息。



> 提示
>
> Web 端不具备持久化的数据存储能力，无法在本地持久化存储历史消息记录与会话列表，因此需要从融云服务端获取数据。从远端获取单群聊历史消息要求您已在控制台 IM 服务管理 页面为当前使用的 App Key 开启单群聊消息云端存储服务。**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。开通 30 分钟后生效，如果 App Key 未开通，则 IMKit 无法从融云服务端获取会话。**

组件用法

导入 IMKit 时，请通过 IMKit 的 `defineCustomElements()` 统一引入所有组件。Vue 项目需要添加过滤自定义组件配置，详见导入 SDK。

```
import * as RongIMLib from '@rongcloud/imlib-next'
// imkit 为核心模块
import { defineCustomElements, imkit } from '@rongcloud/imkit'

defineCustomElements()
```

按上述方式引入组件后，您可以直接使用 `message-list`。

属性列表

下面列出了 message-list 组件的属性。

属性名称	类型	描述
base-size	string	设置组件大小，接受以 px 为单位的数字或百分比（根据父级 font-size 百分比），例如 18px、62.5%。默认值：16px。
customMenu	{name: (message: IAReceivedMessage) => string, callback: (message: IAReceivedMessage) => any }[];	应用程序在消息右键菜单中添加的自定义功能。要求 IMKit SDK ≧ 5.3.0。
disableMenu	DisabledMessageContextMenu[]	应用程序在消息右键菜单中隐藏的 IMKit 预置功能。要求 IMKit SDK ≧ 5.6.0。
forwardList	{callback: (message: IAReceivedMessage) => IConversationOption[] };	转发消息的目标会话。要求 IMKit SDK ≧ 5.3.0。
maxRecallDuration	number	消息撤回的最大有效期，单位为 s，默认值：120s。要求 IMKit SDK ≧ 5.6.0。

组件样式

> 提示
>
> 如不设置 div 样式，会导致消息列表无法滚动或其他异常。

使用 message-list 组件时，必须将组件包裹在 div 元素，并通过 div 的样式控制组件的宽高等样式。

```
<template>
<div class="chat-massage-list">
<message-list ref="messageList" base-size="10px"></message-list>
</div>
</template>

<style scoped>
.chat-massage-list {
height: calc(100% - 220px);
}
</style>
```

事件处理

事件名称	返回值	描述
tapMessage	event	点击图片、文件消息触发监听事件，可处理业务层相关操作，例如：查看大图或文件下载。默认值：null。

> 提示
>
> 以下示例代码以 Vue 框架为例。

```
<template>
<div class="chat-massage-list">
<message-list ref="messageList" base-size="10px"></message-list>
</div>
</template>

<script>
// Vue 代码
export default {
mounted() {
const messageList = this.$refs.messageList;
//添加点击消息触发监听
messageList.addEventListener("tapMessage", this.handleTapMessage);
},
beforeUnmount() {
// 注意：需要 removeEventListener 防止多次绑定造成异常
const messageList = this.$refs.messageList;
messageList.removeEventListener("tapMessage", this.handleTapMessage);
},
methods| {
handleTapMessage(e) {
const data = e.detail;
// 处理点击查看大图或文件消息下载等功能
console.log("点击消息触发监听:",data)
}
}
}
</script>

<style scoped>
.chat-massage-list {
height: calc(100% - 220px);
}
</style>
```

自定义消息右键菜单

IMKit SDK 消息右键菜单默认提供以下功能选项：

1. 删除消息
2. 引用消息
3. 转发消息（需要 App 侧提供转发目标用户的信息）
4. 撤回消息，消息撤回操作仅支持撤回己方发送消息。从 5.6.0 开始，限制消息发送后固定时长内（默认 120 秒）可撤回。过期后撤回按钮自动隐藏。

应用程序可以自定义 IMKit 的消息右键菜单，具体支持以下操作：

- 添加自定义功能按钮。例如，您可以自行实现消息收藏、消息设置为公告等功能。
- 移除 IMKit 预置功能按钮，例如转发、撤回等功能按钮。

添加自定义消息右键菜单选项

> 提示
>
> IMKit 5.3.0 版本开始支持该功能。

IMKit 支持在消息的右键菜单中添加自定义功能按钮。您可以在业务侧自行实现并添加消息收藏、消息设置为公告等功能选项。

```
// 拿到消息列表组件
const messageList = this.$refs.messageList;

const messageCustomMenu = [{
name: function (message) {
// 可通过添加判断条件给特定的消息增加按钮，下方示例为：给非文本类型消息添加 '自定义' 按钮
if(message.messageType !== 'RC:TxtMsg')
return '自定义'
},
callback: function (message) {
console.info('点击自定义按钮拿到消息:', JSON.stringify(message))
}
}]

messageList.customMenu = messageCustomMenu;
```

隐藏消息右键菜单选项

> 提示
>
> IMKit 5.5.1 版本开始支持该功能。

默认情况下，IMKit 会展示所有预置功能按钮。应用程序可以按需隐藏预置功能按钮。

```
// 引入消息右键菜单按钮枚举
import { DisabledMessageContextMenu } from "@rongcloud/imkit";
// 拿到消息列表组件
const messageList = this.$refs.messageList;

// 示例：隐藏转发按钮
const disableMenu = [DisabledMessageContextMenu.Forward];

messageList.disableMenu = disableMenu;
```

设置撤回时间限制

> 提示
>
> IMKit 5.6.0 版本开始支持该功能。IMKit 在 5.6.0 版本以前对于撤回消息并没有做时间限制，现在主流的社交软件都会进行撤回时间限制，建议开发者自行做撤回时间限制。

IMKit 默认提供消息撤回功能，撤回按钮默认有效期为 120s。默认时间不满足需求时可通过设置消息组件中的 `maxRecallDuration` 属性。过期后撤回按钮自动隐藏。

```
// 拿到消息列表组件
const messageList = this.$refs.messageList;

// maxRecallDuration 设置单位为s，示例中设置为 120 秒。发送时间超过 120s 的消息将不可被撤回。设置为 0 时使用默认值。
messageList.maxRecallDuration = 120;
```

消息编辑组件

更新时间:2024-08-30

消息编辑组件 message-editor 负责编辑发送消息的内容。客户端应用程序必须连接到融云服务器，才可以发送消息显示会话列表。

输入框中的内容会由 IMKit SDK 直接发送，不支持外部获取。目前支持发送的消息类型包括：

- 文本消息
- 引用消息
- 图片消息
- 文件消息
- @ 消息（输入 @ 符号，会弹出群成员列表，选中 @ 成员即可）

未列出的消息类型（详见消息类型概述）均不支持通过 IMKit 的输入框发送。

组件用法

导入 IMKit 时，请通过 IMKit 的 defineCustomElements() 统一引入所有组件。Vue 项目需要添加过滤自定义组件配置，详见导入 SDK。

```
import * as RongIMLib from '@rongcloud/imlib-next'
// imkit 为核心模块
import { defineCustomElements, imkit } from '@rongcloud/imkit'

defineCustomElements()
```

按上述方式引入组件后，您可以直接使用 message-editor。

属性列表

下面列出了 message-editor 组件的属性。

属性名称	类型	描述
base-size	string	设置字体图标和发送按钮文字大小（并非输入框输入文字大小）。接受以 px 为单位的数字或百分比（根据父级 font-size 百分比），例如 18px、62.5%。默认值：16px。

组件样式

> 提示
>
> 如不设置 div 样式，会导致消息编辑组件展示位置异常。

使用 message-editor 组件时，必须将组件包裹在 div 元素，并通过 div 的样式控制组件的宽高等样式。

```
<template>
<div class="chat-massage-editor">
<message-editor ref="messageEditor" base-size="10px"></message-editor>
</div>
</template>

<style scoped>
.chat-massage-editor {
height: 220px;
}
</style>
```

选中会话

更新时间:2024-08-30

应用程序可以通过 IMKit SDK 的设置选中会话功能，跳转到指定的会话页面。例如：

- 首次加载时会出现未选中任何会话的状态。应用程序可以使用选中会话功能跳转到指定会话中。如果选中群组会话（`ConversationType.GROUP`），建议应用程序先判断当前用户是否在群组内。如用户不在群组建议不要跳转，防止用户向未加入的群组发消息（会导致发送失败，控制台可查看到失败信息）。
- 如果需要发送消息的目标会话未展示在会话列表中，可以通过选中会话跳转到该会话页面。此时 IMKit SDK 默认会直接生成一个内存态会话（页面刷新后该会话将丢失）添加到会话列表中。如果 App Key 已开通**开通单群聊消息云存储**，SDK 会在会话中产生消息后将该会话存入会话列表，会话列表由服务端存储。

使用 IMKit 实例的 `selectConversation` 选中会话：

```
import { ConversationType } from '@rongcloud/imlib-next'
imkit.selectConversation({
conversationType: ConversationType.PRIVATE, // 会话类型
targetId: "目标 ID"
});
```

名称	参数	必传	说明
conversationType	string	是	会话类型。
targetId	string	是	目标 ID。

删除会话

更新时间:2024-08-30

> 提示
>
> IMKit 从 5.5.0 版本开始支持该功能。

IMKit 提供删除会话接口，传入会话参数可清除掉对应会话。

- 业务中有踢出群聊、把某个用户拉入黑名单的需求时，可清除对应会话。
- 业务中其他需要去除掉某个特定会话的场景。

使用 IMKit 实例的 removeConversation 删除会话。

```
const conversation = {
// 会话类型，单聊为:PRIVATE，群聊为:GROUP，可从 conversation 中获取
conversationType: ConversationType.PRIVATE,
// 群聊为群组 Id，单聊为 userId，可从 conversation 中获取
targetId: '目标 Id'
}
imkit.removeConversation(conversation)
```

拦截过滤会话

更新时间:2024-08-30

IMKit SDK 支持过滤会话列表中的数据，由应用程序决定是否展示部分会话。

> 提示
>
> 过滤会话设置需要在 IMKit 初始化时传入，设置此回调后会影响会话列表展示数据，请慎重设置。

在 IMKit SDK 的 `init` 方法中添加 `customIntercept`，接收会话信息并返回该会话是否需要在会话列表中展示。

```
// 如已引入可忽略
import { ConversationType } from '@rongcloud/imlib-next'

// 代码示例中展示过滤掉系统类型会话，使用时可根据您的需要按需过滤，过滤后不会在会话列表中展示，请知晓！！！
const customIntercept = {
// 过滤会话 false / true
interceptConversation: conversation => {
if (!conversation) return true;

// 匹配过滤 - 系统会话
if (conversation.conversationType === ConversationType.SYSTEM) {
return true; // 返回 true 为不展示该会话
}

// 正常会话 - 不过滤
return false;// 返回 false 正常展示
}
};

/** 特别注意：
* 1. 设置此回调后会影响会话列表展示数据，请慎重设置。
* 2. 此处 init 仅为展示自定义消息设置，应用内不需要多次进行初始化
*/
imkit.init({
customIntercept:customIntercept
});
```

转发消息

更新时间:2024-08-30

> 提示
>
> IMKit 5.3.0 版本开始支持该功能。

IMKit SDK 消息右键菜单默认已提供转发功能，具体支持能力如下：

- 仅支持逐条转发
- 单次转发消息的目标会话不得超过 5 个
- 支持转发的消息类型为：文本消息，图片消息，文件消息

用户点击转发按钮后，SDK 会触发回调，由应用程序负责提供转发会话类型和会话 ID。

```
// 拿到消息列表组件
const messageList = this.$refs.messageList;

const forwardList = {
//点击转发按钮触发回调
callback: () => {
return [{
conversationType: '转发会话的类型',
targetId: '转发会话的目标 ID'
}]
}
};

messageList.forwardList = forwardList;
```

未读消息数

更新时间:2024-08-30

IMKit SDK 中默认已实现展示、清除未读消息数的能力，应用程序不需要额外操作：

- 获取会话未读数并展示。
- 从会话列表进入会话后会跳转到最新消息，此时 SDK 会清除会话未读数（如需支持将阅读状态同步至其他平台的设备，请升级至 5.6.0 版本）。
- 接收用户在其他平台的设备的未读数更新，并清除会话未读数。

隐藏免打扰会话的未读数

> 提示
>
> IMKit 从 5.6.4 版本开始支持该功能。只针对免打扰状态下的会话生效。是否隐藏未读数状态值由业务层来维护，所以在每次进入界面前需先获取该状态值再进行设置。

会话的未读数默认显示为具体数值。在 IMKit 中可以设置隐藏未读数数字，只显示小红点。该功能只针对免打扰状态下的会话生效。非免打扰状态下的会话正常显示会话未读数。

```
const conversationList = this.$refs.conversationList;
// true 表示隐藏数字，false 为显示数字
const hideNotificUnreadCount = true
conversationList.hideNotificUnreadCount = hideNotificUnreadCount;
```

自定义消息

更新时间:2024-08-30

IMKit 支持自定义消息类型，并通过 IMKit 发送、接收和展示自定义消息。

注册自定义消息

注册消息是发送自定义消息的关键步骤，如不注册则无法发送自定义消息。

在建立 IM 连接之前，使用 `imkit` 实例的 `registerMessageType` 方法来注册自定义消息，该方法会返回自定义消息构造函数。

```
// registerMessageType 中参数设置需要按照参数说明顺序传递。不可改变顺序，
// 例如：如需发送状态消息 searchProp 参数不可不设置，可设置为 []
const PersonMessage = imkit.registerMessageType('kit:person', true, true);
```

参数	类型	说明	是否必传
messageType	string	消息类型。请勿使用 RC: 开头的类型名，以免和 SDK 默认的消息名称冲突。如需多端互通（例如 Web 端与 Android、iOS 等移动端互通消息），注册的消息类型名必须保持多端一致，否则无法互通。	是
isPersited	boolean	是否存储。	是
isCounted	boolean	是否计数。	是
searchProp	string[]	搜索字段，web 端无需设置，搜索字段值设置为数字时取值范围为 (-Math.pow(2, 64), Math.pow(2, 64)) 且为整数	否
isStatusMessage	boolean	是否是状态消息。状态消息不存储、不计数，接收方在线时才能收到	否

提示

注意事项：

1. 注册消息类型（`imkit.registerMessageType`）必须在 `connect` 之前调用，必须放在发送、接收该自定义消息之前。
2. 推荐将所有注册自定义消息代码放在 `init` 方法之后。
3. 请尽量把应用中所有涉及到的自定义消息一次性统一注册，且同类型消息仅调用一次注册。

发送自定义消息

应用程序注册自定义消息后，可获取自定义消息体构造函数。在发送自定义消息前，使用该函数构建自定义消息体，例如 `new PersonMessage({})`。消息属性 key 可自定义，保持多端一致即可。

```
// 如已引入可忽略
import { ConversationType } from '@rongcloud/imlib-next'

// PersonMessage 中的属性需要多端保持一致，可根据用户侧需要进行定义。
const message = new PersonMessage({ key1: 'value1', key2: 'value2' })
const option = null;
const conversation = {
conversationType: ConversationType.PRIVATE,
targetId: "<targetId>",
};

imkit.sendMessage(conversation, message, option).then(res => {
console.info('发送消息结果：'，res);
})
```

参数	说明	必传
conversation	消息发送目标会话信息	是
message	待发送的消息内容，通过 new 自定义消息得到消息实例	是
options	定义发送行为中的一些可选项，如是否可拓展，推送等	否

- **conversation**

参数	类型	说明	必传
conversationType	ConversationType	会话类型	是
targetId	string	接收方 Id	是

- **options**

> 提示
>
> options 为可选项，options 中所有属性均为可选项，非必传。

参数	类型	说明
isStatusMessage	boolean	是否为状态消息
disableNotification	boolean	是否发送静默消息
pushContent	string	Push 信息
pushData	string	Push 通知携带的附加信息
isMentioned	boolean	是否为 @ 消息，只当 conversationType 值为 `ConversationType.GROUP` 时有效
directionalUserIdList	string[]	用于发送群定向消息，只当 conversationType 值为 `ConversationType.GROUP` 时有效
isVoipPush	boolean	当对方为 iOS 设备且未在线时，其将收到 Voip Push. 此配置对 Android 无影响
canIncludeExpansion	boolean	是否允许消息被拓展

参数	类型	说明
expansion	[key: string]: string	消息拓展内容数据
isFilerWhiteBlacklist	boolean	黑/白名单
pushConfig	IPushConfig	移动端推送配置（与 Android、iOS 端的 MessagePushConfig 作用相似）

设置自定义消息展示样式

应用程序需要在 IMKit SDK 初始化时传入 customMessage 对象实例，在该实例中设置自定义消息的样式。SDK 在会根据初始化时在 设置好的样式展示已发送或接收的自定义消息。

```
// 构造 IMKit 初始化参数
const customMessage = {
// 普通消息展示
userMessage: {
// key 为自定义消息的 messageType，return 的元素中暂不支持设置 class，如需设置样式可设置为行内样式。
'kit:person': (message) => {
const content = message.content;
return `<div style='padding: 0.5em 0.8333em;'>来自 ${`content.name`} 的好友请求</div>`;
}
},
// 通知类消息展示
notifyMessage: {
// key 为自定义消息的 messageType，return 的元素中暂不支持设置 class，如需设置样式可设置为行内样式。
'kit:notify': (message) => {
const content = message.content
const string = `<div>来自 ${`content.name`} 的好友请求</div>`
return string;
}
},
// 会话最后一条消息展示
lastMessage:{
// key 为自定义消息的 messageType，return 的元素中暂不支持设置 class，如需设置样式可设置为行内样式。
'kit:person': (message) => {
const content = message.content;
return `[好友请求]`;
},
'kit:notify': (message) => {
const content = message.content
const string = `[不带头像好友请求]`
return string;
}
}
};

// 特别注意：此处 init 仅为展示自定义消息设置，应用内不需要多次进行初始化
imkit.init({
customMessage:customMessage
});
```

提示

特别注意：

1. 此处示例代码中的 init 调用仅为展示自定义消息设置，应用内不需要多次进行初始化。
2. 存在自定义消息但未设置展示格式，会导致消息无法展示或展示为空。

customMessage 中传入的对象可包含以下参数：

参数	说明
userMessage	普通消息展示
notifyMessage	通知类消息展示
lastMessage	会话列表最后一条消息展示

上述参数所对应的展示位置可参见下图：

提示

如存在自定义消息但未设置展示格式，会导致消息无法展示或展示为空。

语言设置

更新时间:2024-08-30

语言设置功能目前支持中英文两种语言。SDK 会根据设置语种展示 IMKit 界面组件文案。默认展示中文。

```
// 切换语言
imKit.changeLanguage({
lang: 'en', // 语言标识
});
```

名称	参数	必需	说明
lang	string	是	语言标识，支持传递 'zh_CN','en' 字符串，默认为 'zh_CN'

提示

如传递的 lang 无法识别，则使用默认的 zh_CN，展示中文。

更新日志 v5.6.6

更新时间:2024-08-30

发布日期：2023/8/07

优化：

1. 优化切换会话逻辑

v5.6.5

发布日期：2023/5/09

优化：

1. 优化后，在切换路由或者删除会话时，SDK 不再会拉取远端会话

Bug：

1. 修复进入页面立即选中会话时,会话列表会出现重复会话的问题
2. 修复在多端登录情况下，会话列表上单聊会话最后一条已发消息的“对方已读/未读”状态错误地显示为已读的问题
3. 修复拉取离线消息导致会话历史消息错乱

v5.6.4

发布日期：2023/3/30

新增功能：

1. 增加会话设置为免打扰时，是否显示小红点增加开关设置

Bug：

1. 修复会话没有消息时,消息列表能一直下拉加载问题
2. 修复路由切换后历史消息拉取存不全问题

v5.6.3

发布日期：2023/2/23

Bug：

1. 修复离线消息量大时无法拉取全部离线消息的问题
2. 修复离线消息的会话过多时，会频繁获取会话详情的问题
3. 修复头像变形问题
4. 修复切换用户时未清理内存状态的问题

5. 修复切换路由后聊天消息列表中本人头像不展示的问题

v5.6.1

发布日期：2023/1/05

问题修复：

1. 修改路由切换无法加载会话列表的问题

其他：

1. 适配 IMLib 5.7.0 版本

不兼容变更：

1. 废弃 IMKit 的连接与断开连接方法（`connect`、`disconnect`）。连接与断开连接时，请使用 IMLib 的 `connect`、`disconnect` 方法
2. IMKit init 接口增加 appkey 必传参数

v5.6.0

发布日期：2022/12/01

新增功能：

1. 新增向其他端同步本端阅读状态功能，本端未读数清零后，其他端会收到通知
2. 新增消息撤回时间默认限制及配置，超过时间限制后隐藏撤回按钮

问题修复：

1. 修复多端已读未读状态异常

v5.5.1

发布日期：2022/11/10

新增功能：

1. 新增右键按钮展示配置功能

问题修复：

1. 修复内置消息类型数据解析异常问题
2. 修复发消息触发 getConversationProfile 接口问题
3. 修复更新 updateUserProfile 信息失败问题

v5.5.0

发布日期：2022/10/14

新增功能：

1. 新增会话删除接口
2. 新增在 UI 上展示提示条通知消息（RC:InfoNtf），群组通知消息（RC:GrpNtf）的能力。默认展示，无需配置。

问题修复：

1. 修复会话列表用户信息 UI 样式遮挡的问题
2. 修复在未选中会话上右键删除会话后错误地清空了当前选中会话的问题
3. 修复输入 @ 消息后删除消息某个字符导致删全部内容的问题
4. 修复右键复制按钮失效问题

v5.4.1

发布日期：2022/9/15

问题修复：

1. 修复会话列表中 @ 消息展示异常问题
2. 修复会话列表中文本消息展示溢出问题
3. 修复解析移动端引用消息异常问题

v5.4.0

发布日期：2022/8/18

新增功能：

1. 新增当前用户信息修改接口 `updateUserProfile`
2. 新增群成员信息修改接口 `updateGroupMembers`
3. 新增会话信息修改接口 `updateConversationProfile`

问题修复：

1. 修改滚动加载历史消息后滚动条无法拉取到底部
2. 修复单聊消息已读未读展示异常

其他：

1. IMkit 适配弹窗模式展示

v5.3.0

发布日期：2022/7/18

新增功能：

1. 新增消息转发功能
2. 新增自定义消息右键功能

3. 新增自定义会话右键功能

问题修复：

1. 修复取消置顶后会话展示异常问题
2. 修复删除会话后 `deleteConversation` 监听未触发问题

v5.2.2

发布日期：2022/6/24

问题修复：

1. 修复切换用户，获取详情接口回调未触发问题
2. 修复群聊消息中 @所有人 功能中英文切换问题
3. 修复清缓存已读未读状态展示异常问题
4. 优化自定义消息单聊中已读未读展示
5. 优化下拉消息滚动

v5.2.1

发布日期：2022/6/10

新增功能：

1. 新增会话草稿功能
2. 新增自定义消息注册接口
3. 新增自定义消息展示配置回调
4. 增加对外暴露 sendMessage 接口发送自定义消息
5. 增加提供设置会话展示回调，可通过回调设置会话是否展示

问题修复：

1. 修复user-name 展示遮挡问题
2. 修复未选中会话时发送消息异常问题

v5.1.0

发布日期：2022/5/27

新增功能：

1. 增加对 imlib 初始化链接的兼容处理
2. 增加消息发送失败重发，点击叹号重发功能
3. 增加切换会话全局事件监听
4. 增加多语言功能，目前支持中英文

问题修复：

1. 修复单聊撤回消息昵称异常问题

其他：

1. 优化会话列表中 @ 消息展示逻辑

v5.0.0

发布日期：2022/05/16

- 首次发布 IMKit（NPM 地址）。Web 端 IMKit 基于 IMLib 5.X SDK 开发，提供了易于使用的构建组件和 UI 交互，包括：会话列表界面、消息列表界面、消息编辑界面，可用于快速搭建 Web 聊天应用界面。目前已支持单聊、群聊会话，不支持聊天室、超级群会话。如需开发聊天室、超级群应用，建议使用 IMLib。

升级说明　　版本依赖变更

更新时间:2024-08-30

IMKit 升级至 5.6.1 及以上版本时，需要注意更换依赖的 IMLib 的版本。IMLib SDK 版本需使用 5.7.0 及以上版本。

接口废弃说明

- **废弃 IMKit 的 connect 接口**：原 `imkit.connect` 接口废弃，连接请替换使用 IMLib SDK 中的 `connect` 接口。
- **废弃 IMKit 的 disconnect 接口**：原 `imkit.disconnect` 接口废弃，连接请替换使用 IMLib SDK 中的 `disconnect` 接口。

快速上手

更新时间:2024-08-30

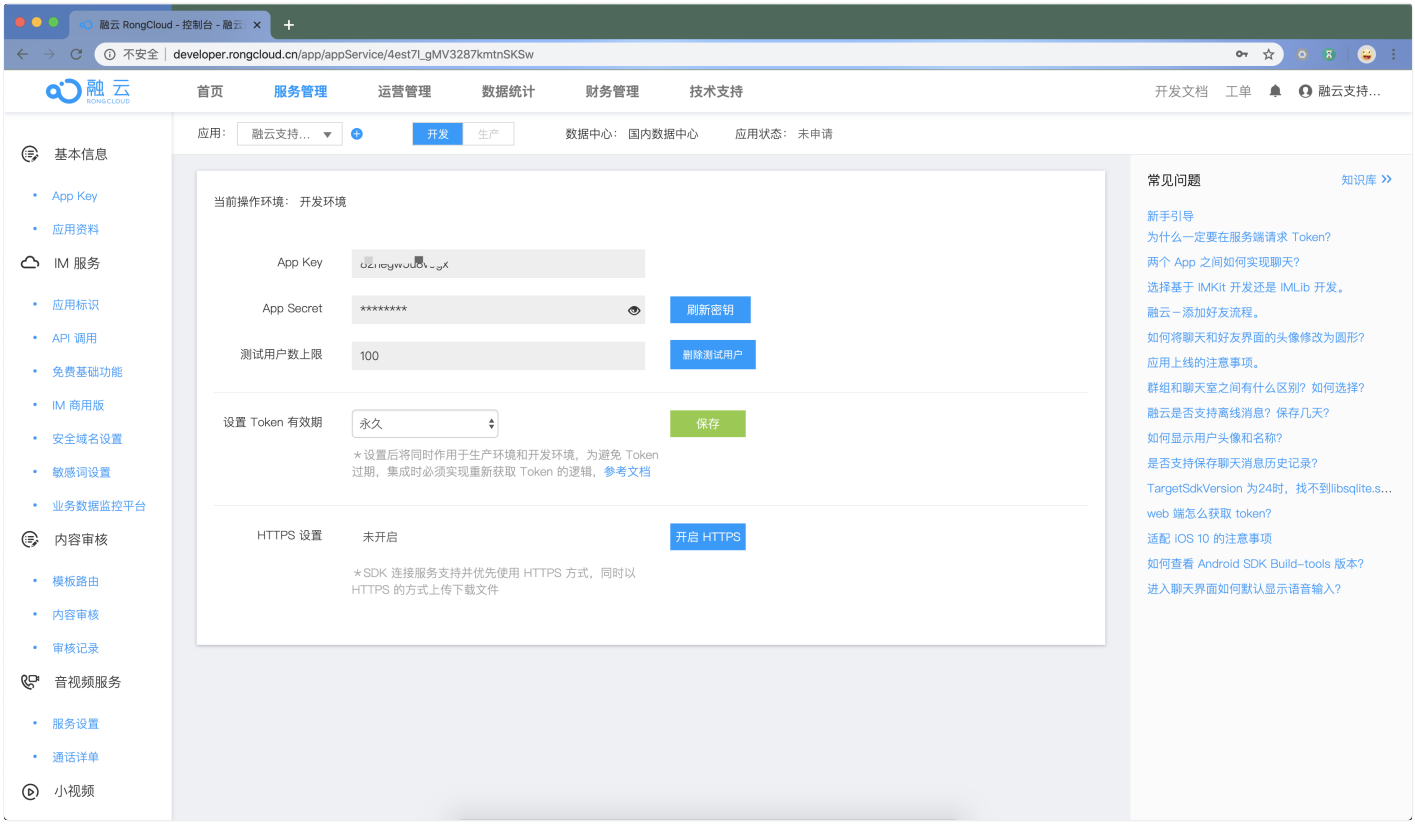
本教程是为了让新手快速了解融云即时通讯能力库（IMLib）。在本教程中，你可以体验集成 SDK 的基本流程和 IMLib 的基础通信能力。

前置条件

创建融云开发者账号，获取 App Key。

控制台将自动为新账号创建一个应用。默认使用国内数据中心，默认提供开发环境。如果您已拥有融云开发者账户，您可以直接创建应用。

> 提示
>
> - 同一个应用的开发环境与生产环境提供不同的 App Key，两个环境之间数据隔离。
> - App Secret 用于生成数据签名，仅在请求融云服务端 API 接口时使用。本教程中暂不涉及。应用的 App Key / Secret 是获取连接融云服务器身份凭证的必要条件，请注意不要泄露。



导入 SDK

> 提示

> IMLib 对 Typescript 的使用者提供了友好的类型化支持，推荐开发者使用 Typescript 进行业务开发以提升代码健壮性及可维护性。

NPM 引入（推荐）

1. 依赖安装

```
npm install @rongcloud/engine@latest @rongcloud/imlib-next@latest -S
```

2. 代码集成

```
// CMD
const RongIMLib = require('@rongcloud/imlib-next')

// ES
import * as RongIMLib from '@rongcloud/imlib-next'
```

CDN 链接引入

```
<script src="https://cdn.ronghub.com/RongIMLib-5.9.5.prod.js"></script>
```

初始化

在使用 IMLib 的能力之前，**必须先调用 IMLib 的初始化接口，且务必保证该接口在应用全生命周期内仅被调用一次。**

App Key 是使用 IMLib 进行即时通讯功能开发的必要条件，也是应用的唯一性标识。您必须拥有正确的 App Key，才能进行初始化。您可以登录融云控制台，查看您已创建的各个应用的 App Key。

只有在 App Key 相同的情况下，不同用户之间的消息才能互通。

```
// 应用初始化以获取 RongIMLib 实例对象，请务必保证此过程只被执行一次
RongIMLib.init({ appkey: '<Your-App-Key>' });
```

设置监听

初始化完成后，添加事件监听器，及时获取相关事件通知。

```
// 添加事件监听
const Events = RongIMLib.Events

RongIMLib.addEventListener(Events.CONNECTING, () => {
console.log('正在链接服务器')
})

RongIMLib.addEventListener(Events.CONNECTED, () => {
console.log('已经链接到服务器')
})

RongIMLib.addEventListener(Events.MESSAGES, (evt) => {
console.log(evt.messages)
})
```

获取用户 Token

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

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

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

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

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

建立 IM 连接

使用以上步骤中获取的 Token, 模拟 userId 为 1 的用户连接到融云服务器。

```
RongIMLib.connect('<Your-Token>').then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('链接成功, 链接用户 id 为: ', res.data.userId);
} else {
console.warn('链接失败, code:', res.code)
}
})
```

获取会话列表

Web 端不具备持久化的数据存储能力，无法在本地持久化存储历史消息记录与会话列表，因此需要从融云服务端获取数据。从远端获取单群聊历史消息要求您已在控制台 IM 服务管理 页面为当前使用的 App Key 开启单群聊消息云端存储服务。**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。

> 提示
>
> 该功能需要在调用 RongIMLib.connect() 并且建立连接成功之后执行。

IMLib 通过会话数据中的 conversationType 与 targetId 两个属性值来标识会话的唯一性，对于两个属性的定义如下：

1. conversationType 用来标识会话类型（如：单聊、群聊...），其值为 RongIMLib.ConversationType 中的常量定义

2. targetId 用来标识与本端进行对话的人员或群组 Id：

- 当 `conversationType` 值为 `RongIMLib.ConversationType.PRIVATE`，`targetId` 为对方用户 Id
- 当 `conversationType` 值为 `RongIMLib.ConversationType.GROUP`，`targetId` 为当前群组 Id
- 当 `conversationType` 值为 `RongIMLib.ConversationType.CHATROOM`，`targetId` 为聊天室 Id

```
// 获取会话列表
RongIMLib.getConversationList().then(({ code, data: conversationList }) => {
if (code === 0) {
console.log('获取会话列表成功', conversationList);
} else {
console.log('获取会话列表失败: ', error.code, error.msg);
}
});
```

发送消息

> 提示
>
> 该功能需要在调用 RongIMLib.connect() 并且建立连接成功之后执行。

以发送文本消息为例：

```
// 指定消息发送的目标会话
const conversation = {
// targetId
targetId: '<TargetId>',
// 会话类型：RongIMLib.ConversationType.PRIVATE | RongIMLib.ConversationType.GROUP
conversationType: RongIMLib.ConversationType.PRIVATE
};

// 构建文本消息
const message = new RongIMLib.TextMessage({ content: 'message content' })

// 发送消息
RongIMLib.sendMessage(conversation, message).then(({ code, data }) => {
if (code === 0) {
console.log('消息发送成功：', data)
} else {
console.log('消息发送失败：', code)
}
});
```

接收消息

当本端作为消息接收的一方，所接收的消息将通过 RongIMLib.addEventListener 和 Events.MESSAGES 注册的消息监听向业务层抛出。具体可参考上述 设置监听 部分

获取历史消息

Web 端不具备持久化的数据存储能力，无法在本地持久化存储历史消息记录与会话列表，因此需要从融云服务端获取数据。从远端获取单群聊历史消息要求您已在控制台 IM 服务管理 页面为当前使用的 App Key 开启单群聊消息云端存储服务。**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。**

> 提示
>
> 该功能需要在调用 RongIMLib.connect() 并且建立连接成功之后执行。

```
const conversation = {
targetId: '<TargetId>',
conversationType: RongIMLib.ConversationType.PRIVATE
};
const option = {
// 获取历史消息的时间戳，默认为 0，表示从当前时间获取
timestamp: +new Date(),
// 获取条数，有效值 1-100，默认为 20
count: 20,
};
RongIMLib.getHistoryMessages(conversation, option).then(result => {
const list = result.data.list; // 获取到的消息列表
const hasMore = result.data.hasMore; // 是否还有历史消息可获取
console.log('获取历史消息成功', list, hasMore);
}).catch(error => {
console.log('获取历史消息失败', error.code, error.msg);
});
```

断开连接

> 提示
>
> 断开当前用户连接，连接断开后无法接收消息、发送消息、获取历史消息、获取会话列表...
> 在下次连接融云成功后，会收取上次离线后的消息，离线消息默认保存 7 天。

```
RongIMLib.disconnect().then(() => console.log('断开链接成功'));
```

后续步骤

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

导入 SDK 兼容说明

更新时间:2024-08-30

Chrome	Firefox	Safari	IE	Edge	QQ 浏览器	微信 浏览器	Android
✔	✔	✔	10+	✔	✔	✔	4.4+

导入 SDK

> 提示
>
> IMLib 对 Typescript 的使用者提供了友好的类型化支持，推荐开发者使用 Typescript 进行业务开发以提升代码健壮性及可维护性。

NPM 引入（推荐）

1. 依赖安装

> 提示
>
> 提示：@rongcloud/engine 包为 imlib-next 提供了能力支持，开发者无需关心其作用，只需保持和 imlib-next 版本一致即可。

```
# 安装 IM 5.0 最新版本
npm install @rongcloud/engine@latest @rongcloud/imlib-next@latest -S
```

2. 代码集成

```
// CMD
const RongIMLib = require('@rongcloud/imlib-next')

// ES
import * as RongIMLib from '@rongcloud/imlib-next'
```

CDN 链接引入

```
<script src="https://cdn.ronghub.com/RongIMLib-5.9.5.prod.js"></script>
```

初始化

更新时间:2024-08-30

在使用 IMLib 的能力之前，必须先调用 **IMLib** 的初始化接口，且务必保证该接口在应用全生命周期内仅被调用一次。

App Key 是使用 IMLib 进行即时通讯功能开发的必要条件，也是应用的唯一性标识。在集成使用 IMLib 之前，请务必先通过 控制台 注册并获取开发者的专属 App Key。

> 提示
>
> 请勿混淆开发/生产环境的 App Key。只有在 App Key 相同的情况下，不同用户之间的消息才能互通。

使用 init 方法进行初始化，并直接传入 App Key。

```
// 应用初始化，请务必保证此过程只被执行一次
RongIMLib.init({ appkey: '<Your-App-Key>' });
```

参数	类型	必填	说明	最低版本	废弃版本
appkey	String	是	应用的唯一标识	3.0.0	
areaCode	AreaCode	否	区域码。不传入任何配置时，SDK 默认连接北京数据中心。如果您 App Key 属于新加坡数据中心或北美数据中心等，则必须传入有效的区域码配置。详见知识库文档海外数据中心使用指南。	5.7.9	
logOutputLevel	EnableLogL	否	日志打印级别，默认为 EnableLogL.WARN	5.6.0	

集成方案（Electron）

更新时间:2024-08-30

即时通讯业务支持使用 Electron 框架开发桌面端应用，实现与传统桌面通讯软件匹配的能力。

> 提示
>
> 使用 Electron 解决方案要求应用已开通桌面版服务。

Electron 解决方案概述

融云 Web IMLib SDK 从 `5.4.0` 版本开始，支持配合使用 @rongcloud/electron 与 @rongcloud/electron-renderer 模块，开发基于 Electron 的桌面版即时通讯应用。融云通过自身 PaaS 能力，解决了多进程连接共享、多进程消息同步问题，降低了开发者构建多窗口、多进程桌面端应用的复杂度。

相较于基于 Web IMLib/IMKit SDK 的 Web 应用，Electron 框架解决方案主要提供了基于本地存储的一系列接口，可用于实现本地消息/会话的获取、搜索、删除等特性。

> 提示
>
> Electron 解决方案暂不支持超级群业务。

Electron 版本支持

Electron 解决方案支持 Windows、Linux、Mac 平台。

Windows 支持版本

Electron 版本	平台	支持架构	备注	支持版本
11.1.X	Windows	x86	`win32-ia32`	
14.0.X	Windows	x86	`win32-ia32`	
16.0.X	Windows	x86	`win32-ia32`	
20.0.X	Windows	x86	`win32-ia32`	5.6.0
20.0.X	Windows	x64	`win32-x64`	5.8.2
不依赖 Electron 版本	Windows	x86	`win32-ia32`	5.8.3
不依赖 Electron 版本	Windows	x64	`win32-x64`	5.8.3

> 提示
>
> 在 windows 平台需使用相应架构的 node。检查方法：
>
> 1. 在 CMD 等命令行界面运行 `node -p process.arch`。

2. 查看当前 node 版本，需与列表中相匹配。

Linux 支持版本

Electron 版本	平台	支持架构	备注	支持版本
11.1.X	Linux	x64	`linux-x64`	
11.1.X	Linux	arm64	`linux-arm64`	
不依赖 Electron 版本	Linux	x64	`linux-x64`	5.8.3
不依赖 Electron 版本	Linux	arm64	`linux-arm64`	5.8.3

MacOS 支持版本

Electron 版本	平台	支持架构	备注	支持版本
11.1.X	MacOS	x64	`darwin-x64`	
14.0.X	MacOS	x64	`darwin-x64`	
16.0.X	MacOS	x64	`darwin-x64`	
20.0.X	MacOS	x64	`darwin-x64`	5.6.0
20.0.X	MacOS	arm64	`darwin-arm64`	5.6.0
不依赖 Electron 版本	MacOS	x64	`darwin-x64`	5.8.3
不依赖 Electron 版本	MacOS	arm64	`darwin-arm64`	5.8.3

安装 SDK

您需要安装 Web IMLib 最新版，以及 Electron 解决方案（@rongcloud/electron 与 @rongcloud/electron-renderer 。）

```
npm install @rongcloud/engine@latest -S
npm install @rongcloud/imlib-next@latest -S
npm install @rongcloud/electron@latest -S
npm install @rongcloud/electron-renderer@latest -S
```

安装 .node 文件

自 5.6.0 版本起，在 `@rongcloud/electron` SDK 中将默认不携带全部平台的 .node 文件，安装完该 SDK 后会自动下载匹配当前环境的 .node 文件到 `node_modules/@rongcloud/electron/binding` 目录下，但可能因未匹配到当前环境的 .node 文件而导致下载失败，您可使用 `npx rc-install` 命令主动下载：

```
// 5.8.2 版本之前
// npx rc-install --electron-version <electron-version>[ --platform <platform>][ --arch <arch>]
npx rc-install --electron-version=16.0.1 --platform=darwin --arch=x64

// 5.8.3 版本之后
// npx rc-install [ --platform <platform>][ --arch <arch>]
npx rc-install --platform=darwin --arch=x64
```

您可在 electron node 支持列表 页面查看各版本支持的全部 .node 文件, .node 文件名称说明：`electron-v[Electron版本]-[平台]-[CPU架构].node`。

提示

1. 在下载完 SDK 后需检查下 `node_modules/@rongcloud/electron/binding` 目录下是否有当前环境下的 .node 文件。
2. 在切换 SDK 版本后需重新下载 .node 文件。
3. 查看当前环境的 CPU 架构信息：在 CMD 等命令行界面运行 node -p process.arch
4. 查看当前环境的系统平台信息：在 CMD 等命令行界面运行 node -p process.platform

主进程初始化

1. 在主进程中引用 `@rongcloud/electron` 包，并在 app 的 `ready` 事件通知后进行初始化。

   在 `main.js` 中：

```
// main.js
const { app, BrowserWindow } = require('electron')
const RCInit = require('@rongcloud/electron')

let rcService

app.on('ready', () => {
// 在 app 的 ready 事件通知后进行初始化
rcService = RCInit({
/**
* 【必填】Appkey , 自 5.6.0 版本起，必须填该参数
* [option]
*/
appkey: '<appkey>',
/**
* 【选填】消息数据库的存储位置，不推荐修改
* [option]
*/
dbpath: app.getPath('userData'),
/**
* 【选填】日志等级
* [option] 4 - DEBUG, 3 - INFO, 2(default) - WARN, 1 - ERROR
*/
logOutputLevel: 2,
/**
* 【选填】是否同步空的置顶会话，默认值为 `false`
* [option]
*/
enableSyncEmptyTopConversation: false
})

// 初始化 UI 窗口
const browserWin = new BrowserWindow({
webPreferences: {
// 指定预加载的 preload.js 文件，在其中引用 @rongcloud/electron-renderer
preload: '<path/to/preload.js>',
// 需要将 contextIsolation 设置为 false
contextIsolation: false,
nodeIntegration: true
}
})

app.on('before-quit', () => {
// 在 app 退出时清理状态
rcService.destroy()
})
})
```

2. 在初始化渲染进程窗口时，通过设置 webPreferences.preload 来添加预加载的 js 文件，并在 js 中引用 @rongcloud/electron-renderer。

   在 preload.js 文件中：

```
// preload.js
const renderer = require('@rongcloud/electron-renderer');
```

   如果开启了上下文隔离，则还需要加入以下代码（5.9.6 版本开始支持）：

```
// preload.js
renderer.initContextBridge()
```

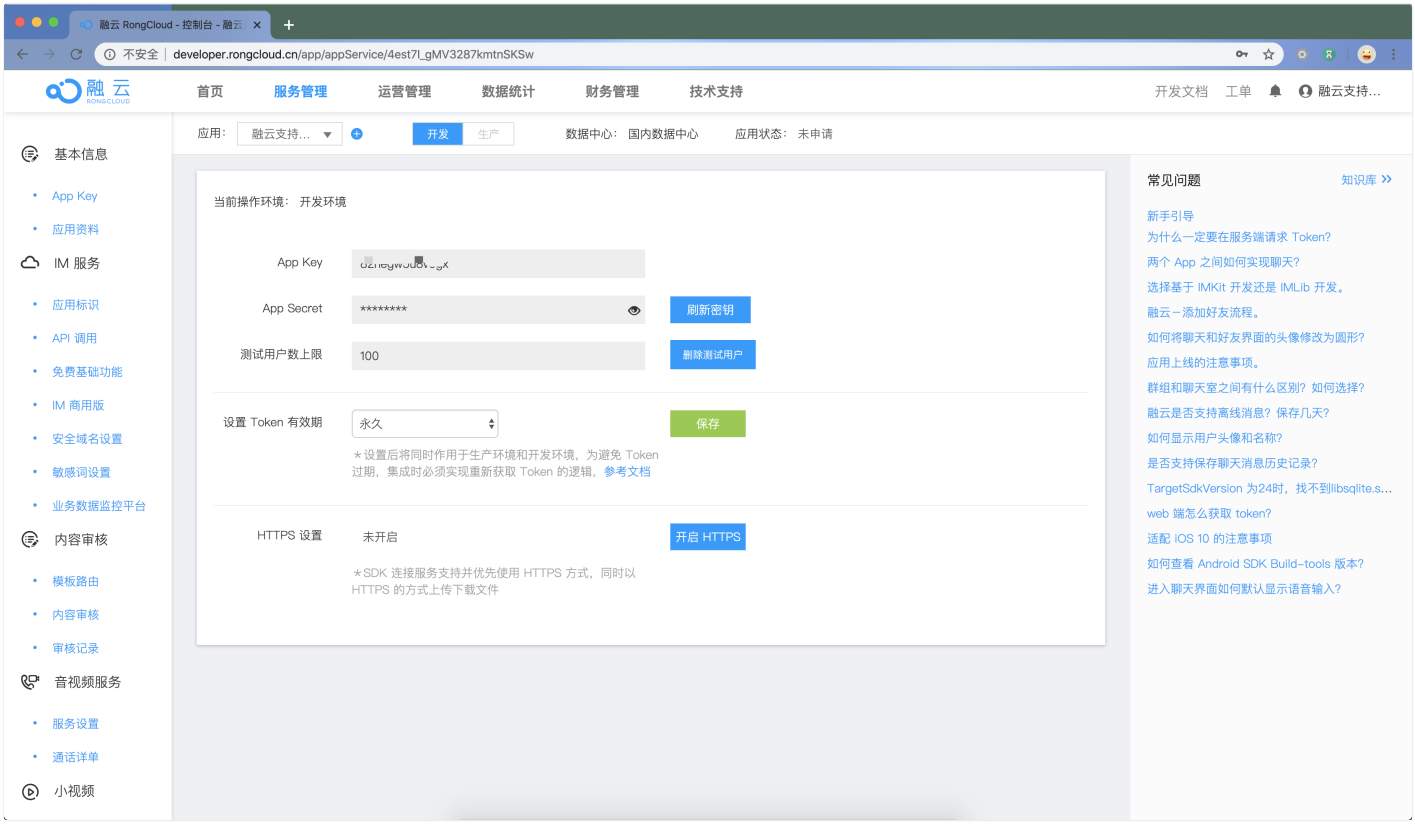
3. 在 Web 页面中引入 @rongcloud/imlib-next。

```
import RongIMLib from '@rongcloud/imlib-next'
```

在渲染进程中初始化 IMLib

初始化 IMLib SDK 时需要传入 App Key。

请登录控制台，记录下图所示的应用 App Key，在初始化时使用。



> 提示
>
> 应用的 AppKey 与 Secret 是获取连接融云服务器身份凭证的必要条件，请注意不要泄露。

1. 初始化 IMLib。

```
// 应用初始化，请务必保证此过程只被执行一次
RongIMLib.init({ appkey: '<Your-App-Key>' });
```

2. 初始化完成后，添加事件监听器，及时获取相关事件通知。

```
// 添加事件监听
const Events = RongIMLib.Events

RongIMLib.addEventListener(Events.CONNECTING, () => {
console.log('正在连接服务器')
})

RongIMLib.addEventListener(Events.CONNECTED, () => {
console.log('已经连接到服务器')
})

RongIMLib.addEventListener(Events.MESSAGES, (evt) => {
console.log(evt.messages)
})
```

获取用户 Token

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

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

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

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

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

建立 IM 连接

使用以上步骤中获取的 Token，模拟 userId 为 1 的用户连接到融云服务器。

```
RongIMLib.connect('<Your-Token>').then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('连接成功, 连接用户 id 为: ', res.data.userId);
} else {
console.warn('连接失败, code:', res.code)
}
})
```

后续步骤

以上步骤即 IMLib SDK 在 Electron 平台的快速集成流程。

文档目录中标注了（**Electron**）的页面均为 Electron 解决方案专属的功能接口文档。

因在 Electron 平台提供了本地存储的能力，所以部分接口可能与 Web 平台中使用有些差异，具体信息可参考各个接口详细说明。

事件监听　　添加监听器

更新时间:2024-08-30

addEventListener 方法用来接收来自于 IMLib 内的各种事件通知，同类型事件可以多次添加不同的监听函数。绑定仅执行一次的事件
可以使用 onceEventListener 。

```
const Events = RongIMLib.Events
RongIMLib.addEventListener(Events.MESSAGES, (evt) => {
console.log(evt.messages)
})
```

移除监听

业务层需注意在必要时使用 removeEventListener 或 removeEventListeners 移除对指定事件的监听函数，以免造成内存泄露。
`removeEventListeners` 将移除对某一特定事件的所有监听函数。

```
const Events = RongIMLib.Events

const listener = (evt) => console.log(evt.messages)

// 添加具体事件监听器
RongIMLib.addEventListener(Events.MESSAGES, listener)

// 移除具体事件监听器
RongIMLib.removeEventListener(Events.MESSAGES, listener)

// 移除具体事件的所有监听器
RongIMLib.removeEventListeners(Events.MESSAGES)
```

事件

事件名称	事件返回值类型	事件说明	版本
CONNECTING	void	链接中	
CONNECTED	void	已链接	
DISCONNECT	ErrorCode	已断开 注意：5.7.0 版本起，回调参数的类型由 ConnectionStatus 变更为 ErrorCode	
SUSPEND	ErrorCode	链接异常断开(自动重连) 注意：5.7.0 版本起，回调参数的类型由 ConnectionStatus & ErrorCode 变更为 ErrorCode	
MESSAGES	IMessagesEvent	收到消息	
READ_RECEIPT_RECEIVED	IReadReceiptReceivedEvent	收到已读回执（单聊）	
MESSAGE_RECEIPT_REQUEST	IMessageReceiptRequestEvent	收到已读回执请求（群聊）	

事件名称	事件返回值类型	事件说明	版本
MESSAGE_RECEIPT_RESPONSE	IMessageReceiptResponseEvent	收到已读回执响应（群聊）	
CONVERSATION	IConversationEvent	收到会话变更通知	
CHATROOM	IChatroomListenerData	收到聊天室变更通知。注意：如需监听聊天室成员变化，请提交工单申请开通服务。	
EXPANSION	IExpansionListenerData	收到扩展变更通知	
PULL_OFFLINE_MESSAGE_FINISHED	void	离线消息拉取完成	
TAG	void	（适用于多端登录场景）收到标签列表标签通知，表示用户可能在其他端创建、移除或编辑某个标签。收到此通知时需主动调用 `getTags` 获取全部标签以更新 UI 渲染。该标签列表包含了所有用于标记会话的标签。	
CONVERSATION_TAG	void	（适用于多端登录场景）收到会话标签变动通知，表示用户可能在其他端修改了会话上所加的标签。收到此通知时需根据本地已渲染的会话列表，主动调用 `getTagsFromConversation` 逐个获取会话的所有标签，以更新 UI 渲染。	
TYPING_STATUS	ITypingStatusEvent	收到正在输入状态通知	
MESSAGE_BLOCKED	IBlockedMessageInfo	敏感词回调通知。需要提交工单开启服务后才能使用。默认关闭状态。	5.0.2
ULTRA_GROUP_ENABLE	IAReceivedConversation	超级群会话列表同步完成，可以调用超级群相关接口	5.2.0
OPERATE_STATUS	IOperateStatusNotify	超级群输入状态通知	5.2.0
ULTRA_GROUP_MESSAGE_EXPANSION_UPDATED	IAReceivedMessage	超级群消息扩展更新通知	5.2.0
ULTRA_GROUP_MESSAGE_MODIFIED	IAReceivedMessage	超级群消息被修改通知	5.2.0
ULTRA_GROUP_MESSAGE_RECALLED	IAReceivedMessage	超级群消息被撤回通知	5.2.0
ULTRA_GROUP_CHANNEL_TYPE_CHANGE	IUltraChannelChangeInfo	超级群频道类型变更	5.4.2
ULTRA_GROUP_CHANNEL_DELETE	IUltraChannelDeleteInfo	超级群频道被删除	5.4.2
ULTRA_GROUP_CHANNEL_USER_KICKED	IUltraChannelUserKickedInfo	超级群私有频道成员被移除	5.4.2
SUBSCRIBED_USER_STATUS_CHANGE	ISubscribeUserStatusInfo	被订阅者发生状态变更	5.9.8
SUBSCRIBED_RELATION_CHANGE	ISubscribeRelationInfo	用户在其他设备上的订阅信息发生变更	5.9.8
SYNC_SUBSCRIBED_USER_STATUS_FINISHED	void	订阅数据同步完成	5.9.8

监听连接状态　　功能描述

更新时间:2024-08-30

使用 addEventListener 监听 IM 连接状态，可根据连接状态进行不同业务处理，或在页面上给出提示。

- 建议在应用生命周期内设置。
- 应用销毁时，应清理这些事件监听。

调用示例

```
const Events = RongIMLib.Events
/**
* 正在链接的事件状态
*/
RongIMLib.addEventListener(Events.CONNECTING, () => {
console.log('正在链接...')
})

/**
* 链接到服务器会触发这个事件
*/
RongIMLib.addEventListener(Events.CONNECTED, () => {
console.log('连接成功')
})

/**
* 手动调用 disconnect 方法或者用户被踢下线 会触发这个事件
*/
RongIMLib.addEventListener(Events.DISCONNECT, (code) => {
console.log('连接中断，需要业务层进行重连处理 ->', code)
})

/**
* 链接出问题时，内部进行重新链接，会触发这个事件
*/
RongIMLib.addEventListener(Events.SUSPEND, (code) => {
console.log('链接中断，SDK 会尝试重连，业务层无需关心')
// 5.1.2 版本开始，事件回调中会引起中断的 code 状态码
console.log(`code -> ${`code`}`)
})
```

提示

- 5.6.1 版本之前，事件 Events.DISCONNECT 与 Events.SUSPEND 回调函数中的 code 类型为 ConnectionStatus
- 5.7.0 版本之后，事件 Events.DISCONNECT 与 Events.SUSPEND 回调函数中的 code 类型为 ErrorCode

连接事件说明

事件	说明
Events.CONNECTING	IM 连接中
Events.CONNECTED	IM 连接完成
Events.DISCONNECT	IM 连接已断开，业务层需进行重连处理
Events.SUSPEND	IM 连接中断，SDK 负责尝试恢复重连

连接

更新时间:2024-08-30

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

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

前置条件

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

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

连接聊天服务器

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

请注意以下几点：

- 必须在 SDK 初始化之后，调用 `connect()` 方法进行连接。否则可能没有回调。
- SDK 本身有重连机制，在一个应用生命周期内不须多次调用 `connect()` 。
- 在应用生命周期调用一次即可。

使用 connect 方法发起连接，传入从用户身份令牌（Token）。

```
RongIMLib.connect('<Your-token>').then((res) => {
if (res.code === 0) {
console.log(res.data.userId)
}
})
```

断开连接

更新时间:2024-08-30

SDK 提供了 disconnect 方法用于断开 IM 连接。

调用示例

```
RongIMLib.disconnect().then(() => {
console.log('成功断开')
})
```

适用场景

注销账号，切换账号时，推荐使用此方法。

重连机制与重连互踢　　自动重连机制

更新时间:2024-08-30

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

可能导致 SDK 断线重连的异常情况如下：

因为客户端 SDK 和融云服务端之间存在连接保活机制，一旦因如果网络太差或无网络导致心跳超时，SDK 就会触发重连操作，尝试重连直到连接成功。

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

重连时间间隔

SDK 尝试重连时，时间间隔是 5s。

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（仅限 Electron）

> 提示
>
> 用户级别的重连互踢策略功能仅限在 Electron 解决方案 中使用。Web 端应用不支持设置重连互踢策略。

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桌面端设备上登录。后登录的桌面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. 用户张三尝试在桌面端设备 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. 用户此时尝试换到桌面端设备 B 上登录。B 设备连接成功，且用户可通过 B 设备正常使用即时通讯业务。
3. A 设备网络稳定之后，SDK 重连成功。因此时 A 设备为后上线设备，导致 B 设备被踢出。

修改 App 用户的重连互踢策略

如果 App 用户希望在以上场景中将重连成功的 A 设备下线，同时保持 B 设备登录，可以通过修改自己的重连互踢策略实现。设置断线重连时是否踢出重连设备需要在建立连接时进行设置。

提示

该功能要求该 App Key 已开通用户级别功能设置。如需开通，请联系客服。

调用示例

```
RongIMLib.connect('<Your-Token>', true).then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('连接成功, 连接用户 id 为: ', res.data.userId);
} else {
console.warn('连接失败, code:', res.code)
}
})
```

参数说明

参数	类型	说明
reconnectKickEnable	boolean	是否踢出正在重连的设备。详见下方 reconnectKickEnable 参数详细说明。

reconnectKickEnable 参数详细说明：

- 设置 reconnectKickEnable 为 true：

  如果重连时发现已有别的桌面端设备在线，将不再重连，不影响已正常登录的桌面端设备。

- 设置 reconnectKickEnable 为 false：

  如果重连时发现已有别的桌面端设备在线，将踢出已在线的桌面端设备，使当前设备上线。

多端同时在线

更新时间:2024-08-30

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

> 默认多端设备之间不会进行消息同步。如有需要，请开通多设备消息同步服务。

多端登录限制说明

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

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

多设备消息同步

更新时间:2024-08-30

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

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

适用场景

在融云即时通讯业务中，多设备消息同步适用于以下情况：

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

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

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

支持在多设备间同步的消息

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

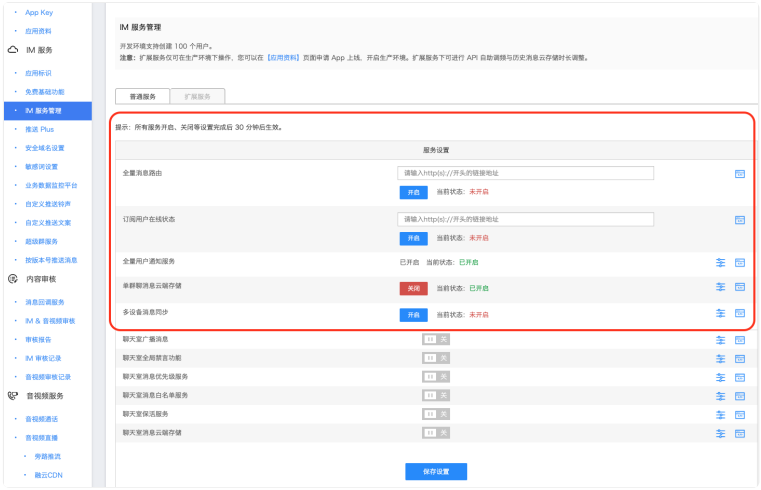
以下情况均属于状态消息：

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

开通服务

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

> 服务开启、关闭设置完成后 30 分钟内生效。



对其他功能或业务的影响

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

对离线补偿的影响

控制台的多设备消息同步服务包含了融云服务端离线补偿机制[?]的开关。

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

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

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

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

对 Web 平台连接数的影响

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

对撤回消息的影响

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

对服务端 API 发送消息的影响

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

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

用户概述

更新时间:2024-08-30

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

注册用户

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

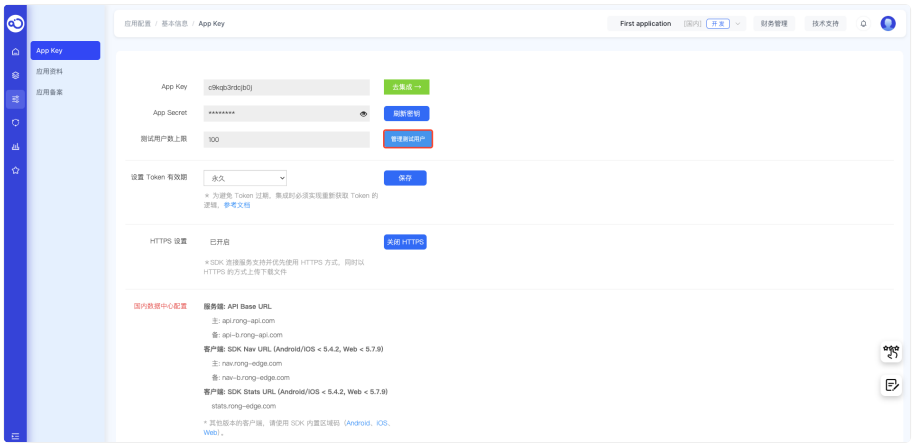
应用客户端必须持有有效 Token，才能成功连接到融云服务端，使用融云即时通讯服务。当 App 客户端用户向服务器发送登录请求时，服务器会查询数据库以检查连接请求是否匹配。

注册用户数限制

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

删除用户

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



注销用户

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

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

仅 IM Server API 提供上述能力。

用户信息

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

好友关系

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

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

用户管理接口

功能分类	功能描述	客户端 API	服务端 API
注册用户	使用 App 用户的用户 ID 向融云换取 Token。	不提供该 API	注册用户
删除用户	参见上文删除用户。	不提供该 API	不提供该 API
废弃 Token	废弃在特定时间点之前获取的 Token。	不提供该 API	作废 Token
注销用户	注销用户是指在融云服务中停用用户 ID，并删除用户个人数据。	不提供该 API	注销用户
查询已注销用户	获取已注销的用户 ID 列表。	不提供该 API	查询已注销用户
重新激活用户 ID	在融云服务中重新启用已注销用户的 ID。	不提供该 API	重新激活用户 ID
设置融云服务端的用户信息	设置在融云推送服务中使用的用户名称与头像。	不提供该 API	未提供单独的设置接口。在注册用户时必须提供用户信息。
获取融云服务端的用户信息	获取用户在融云注册的信息，包括用户创建时间和服务端的推送服务使用的用户名称、头像 URL。	不提供该 API	获取信息
修改融云服务端的用户信息	修改在融云推送服务中使用的用户名称与头像。	不提供该 API	修改信息
封禁用户	禁止用户连接到融云即时通讯服务，并立即断开连接。可按时长解封或主动解封。查询被封禁用户的用户 ID、封禁结束时间。	不提供该 API	添加封禁用户、解除封禁用户、查询封禁用户
查询用户在线状态	查询某用户的在线状态。	不提供该 API	查询在线状态
加入黑名单	在用户的黑名单列表中添加用户。在 A 用户黑名单的用户无法向 A 发送消息。	加入黑名单（暂仅支持 Electron）	加入黑名单
移出黑名单	在用户的黑名单中移除用户。	移出黑名单（暂仅支持 Electron）	移出黑名单
查询黑名单	查询某用户（userId）是否已被当前用户加入黑名单。	查询用户是否在黑名单中（暂仅支持 Electron）	不提供该 API
获取黑名单列表	获取用户的黑名单列表。	获取黑名单列表（暂仅支持 Electron）	查询黑名单
用户白名单	用户一旦开启并设置白名单，则仅可接收来自该白名单中用户的消息。	不提供该 API	开启用户白名单、用户白名单状态查询、添加白名单、移出白名单、查询白名单

黑名单管理（Electron）

更新时间:2024-08-30

> 本功能暂仅限于在 Electron 解决方案中使用。

将用户加入黑名单之后，将不再收到对方发来的任何单聊消息。

- 加入黑名单为单向操作，例如：用户 A 拉黑用户 B，代表 B 无法给 A 发消息（错误码 `405`）。但 A 向 B 发消息，B 仍然能正常接收。
- 单个用户的黑名单总人数存在上限，具体与计费套餐有关。IM 旗舰版与 IM 尊享版上限为 3000 人，其他套餐详见功能对照表中的服务限制。
- 调用服务端 API 发送单聊消息默认不受黑名单限制。如需启用限制，请在调用 API 时设置 `verifyBlacklist` 为 `1`。

加入黑名单

调用 addToBlacklist 将某个用户加入黑名单。

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。本功能暂仅限于在 Electron 解决方案中使用。

```
RongIMLib.addToBlacklist('<userId>').then(res => {
console.log(res.code)
})
```

参数	类型	说明
userId	string	用户 ID

移出黑名单

调用 removeFromBlacklist 将某个用户移除黑名单。

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。本功能暂仅限于在 Electron 解决方案中使用。

```
RongIMLib.removeFromBlacklist('<userId>').then(res => {
console.log(res.code)
})
```

参数	类型	说明
userId	string	用户 Id

获取黑名单列表

调用 getBlacklist 获取黑名单列表。

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。本功能暂仅限于在 Electron 解决方案中使用。

```
RongIMLib.getBlacklist().then(res => {
console.log(res)
})
```

查询用户是否在黑名单中

调用 getBlacklistStatus 获取指定人员是否在黑名单中。

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。本功能暂仅限于在 Electron 解决方案中使用。

```
RongIMLib.getBlacklistStatus('<userId>').then(res => {
console.log(res)
})
```

参数	类型	说明
userId	string	用户 Id

订阅用户在线状态

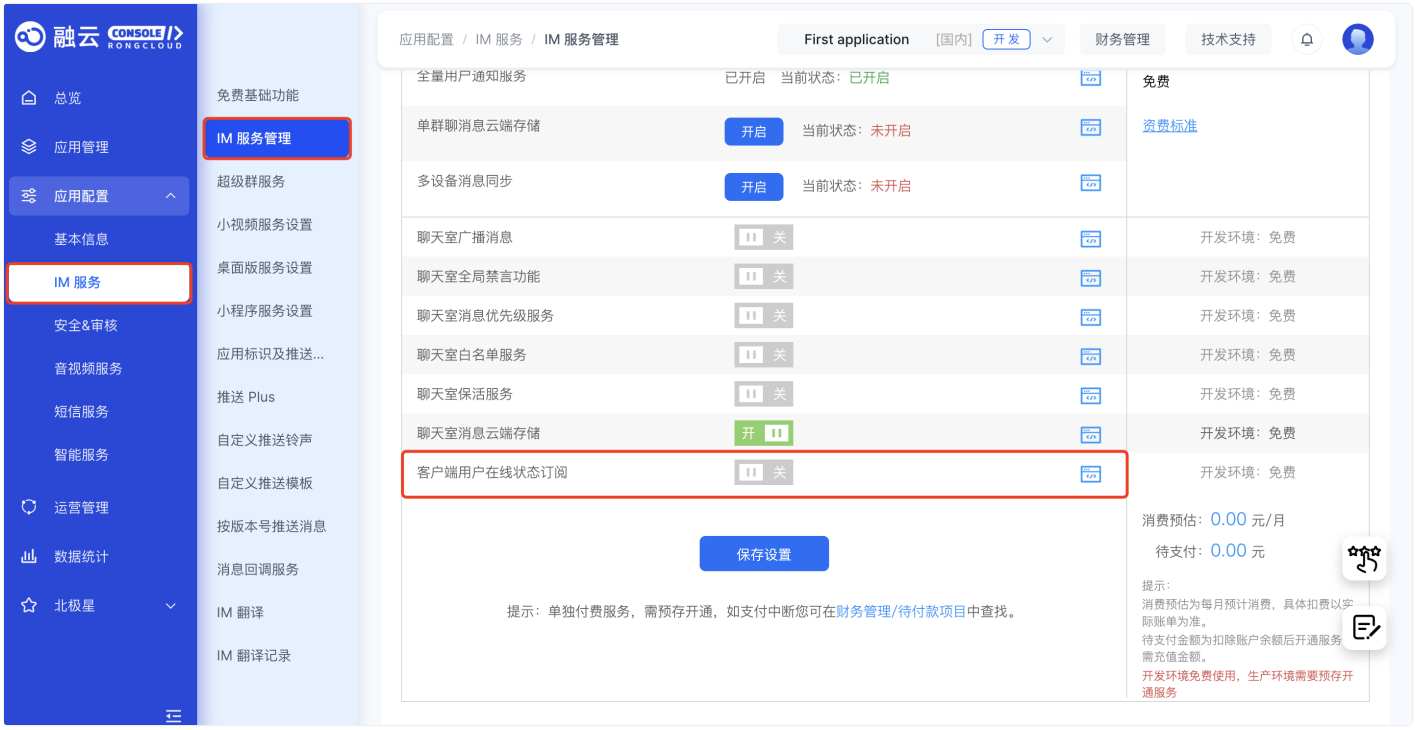
更新时间:2024-08-30

本文档旨在指导开发者如何在融云即时通讯客户端 SDK 中实现用户在线状态的订阅、查询和监听。通过本文档，开发者将了解如何获取和跟踪用户在线状态，以及如何在状态变更时接收通知。

> 提示
>
> 此功能在 5.9.8 版本开始支持。

开通服务

您可以通过控制台开通服务。在融云控制台，选择 **IM 服务 > IM 服务管理 > 客户端用户在线状态订阅** ，开启服务。



订阅用户在线状态

为了跟踪特定用户的在线状态，您需要使用 subscribeUserStatus 方法进行订阅。以下是具体的操作步骤和示例代码：

1. 定义订阅用户 ID 列表：创建一个包含您希望订阅用户 ID 的数组，即单聊的 `targetId`。一次订阅用户的上限为 200 个，订阅的用户数最多为 1000 个，一个用户最多可以被 5000 个用户订阅。
2. 设置订阅时间：定义一个时间值，该值表示订阅的持续时间，订阅时长的范围为 60 秒到 2592000 秒。
3. 指定订阅类型：使用 `RongIMLib.SubscribeType.ONLINE_STATUS` 来指定订阅的是用户在线状态。
4. 使用 `subscribeUserStatus` 方法执行订阅操作。

```
// 订阅订阅用户userId列表，也是单聊的 targetId（一次最多订阅 200 个）。
const userIds = ['user1', 'user2']
// 设置订阅时间，取值范围为[60,2592000]（单位:秒）。
const expiry=180000;
// 指定订阅类型为用户在线状态。
const type = RongIMLib.SubscribeType.ONLINE_STATUS

RongIMLib.subscribeUserStatus(userIds, type, expiry).then((res) => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('订阅成功')
} else if (res.code === RongIMLib.ErrorCode.RC_BESUBSCRIBED_USERIDS_COUNT_EXCEED_LIMIT) {
console.log('用户被订阅达到上限', res.code, res.data)
} else {
console.log('订阅失败', res.code, res.msg)
}
})
```

取消订阅用户在线状态

当您不再需要跟踪用户的在线状态时，可以使用 unSubscribeUserStatus 方法取消订阅：

```
// 取消订阅用户userId，也是单聊的 targetId(一次最多取消订阅 200 个)
const userIds = ['user1', 'user2']
// 取消订阅类型
const type = RongIMLib.SubscribeType.ONLINE_STATUS

RongIMLib.unSubscribeUserStatus(userIds, type).then((res) => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('取消订阅成功')
} else {
console.log('取消订阅失败', res.code, res.msg)
}
})
```

查询订阅状态信息

您可以使用 getSubscribeUserStatus 方法查询指定用户和订阅类型的状态信息。 一次最多查询 200 个用户的状态信息。

```
// 指定要查询用户的 userId，即单聊的 targetId。 一次最多查询 200 个用户的状态数据。
const userIds = ['user1', 'user2']
// 指定要查询的订阅类型。
const type = RongIMLib.SubscribeType.ONLINE_STATUS

RongIMLib.getSubscribeUserStatus(type, userIds).then((res) => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('获取成功', res.data)
} else {
console.log('获取失败', res.code, res.msg)
}
})
```

分页查询已订阅用户的状态信息

如果您需要分页获取已订阅的所有事件状态信息，可以使用 `getSubscribeUserList` 方法.

```
// 设置要查询的订阅类型。
const type = RongIMLib.SubscribeType.ONLINE_STATUS
// 设置分页大小，取值范围为 [1~200]。
const pageSize = 20
// 设置分页偏移量。offset 常量用于控制查询的起始位置。对于第一次查询（即第一页），我们将其设置为 0。对于后续的查询（例如获取第二页或第三页的数据），offset 应该设置为之前一页返回的数据数组数量的累加值。比如第一页的偏移量为 0，pageSize 为 20，则第二页的偏移量为 20， 第三页为 40.
const offset = 0

RongIMLib.getSubscribeUserList(type, pageSize, offset).then((res) => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('获取成功', res.data)
} else {
console.log('获取失败', res.code, res.msg)
}
})
```

监听订阅事件

为了接收订阅事件的变更通知，您需要设置订阅监听器。使用 addEventListener 来添加 SUBSCRIBED_USER_STATUS_CHANGE 事件的监听器。监听器需要在连接之前调用。

> 提示
>
> 从 5.10.1 版本开始，订阅事件包含了用户在线状态和用户信息托管两种种类型，订阅事件的变更，需要根据 subscribeType 的类型来处理对应的业务。

订阅用户状态变更

```
/**
* 所订阅的用户的状态变更时会收到此通知。
* 订阅过期后，融云 SDK不会主动通知您，请您自行关注过期时间。
*/
RongIMLib.addEventListener(Events.SUBSCRIBED_USER_STATUS_CHANGE, (event) => {
// 5.10.1 版本开始需根据数据中的 subscribeType 来判断是用户资料还是在线状态
console.log('被订阅者状态变更', event)
})
```

订阅关系变更（多端同步）

```
/**
* 当用户在其他设备上的订阅信息发生变更时会收到此通知。
* 这可以用于更新当前设备上的用户状态，确保订阅信息的一致性。
*/
RongIMLib.addEventListener(Events.SUBSCRIBED_RELATION_CHANGE, (event) => {
// 5.10.1 版本开始需根据数据中的 subscribeType 来判断是用户资料还是在线状态
console.log('订阅关系变更', event)
})
```

订阅数据同步完成

```
/**
* 订阅数据同步完成。
* 在订阅数据成功同步到设备或系统后会受到此通知，用于执行后续的业务操作。
*/
RongIMLib.addEventListener(Events.SYNC_SUBSCRIBED_USER_STATUS_FINISHED, (event) => {
// 5.10.1 版本开始需根据数据中的 subscribeType 来判断是用户资料还是在线状态
console.log('订阅数据同步', event)
})
```

用户信息托管

更新时间:2024-08-30

本文档旨在指导开发者如何在融云即时通讯客户端 SDK 中实现用户信息订阅、查询和监听，同时支持用户信息与权限的修改和查询。
通过本文档，开发者将了解如何获取和跟踪用户信息，以及如何在状态变更时接收通知。

> 提示
>
> 此功能在 5.10.1 版本开始支持。

开通服务

使用此功能前，您须在控制台开通信息托管服务。

用户信息管理

SDK 提供了用户信息修改、查询、订阅的相关接口。

更新当前用户信息

您可以使用 updateMyUserProfile 方法来更新当前用户信息。

IUserProfileInfo 属性介绍

属性名	类型	描述
name	string	昵称，长度不超过 32 个字符。
portraitUri	string	头像地址，长度不超过 128 个字符。
uniqueId	string	用户应用号，支持大小写字母、数字，长度不超过 32 个字符。请注意 SDK 不支持设置此字段。
email	string	Email，长度不超过 128 个字符。 |
birthday	string	生日，长度不超过 32 个字符 |
gender	RCUserGender	性别 |
location	string	所在地，长度不超过 32 个字符。
role	number	角色，支持 0~100 以内数字。
level	number	级别，支持 0~100 以内数字。
userExtProfile	{ [key: string]: string }	自定义扩展信息，默认最多可以设置 20 个用户信息（以 Key、Value 方式设置，扩展用户信息的 Key 需通过开发者后台进行设置） 1. Key：支持大小写字母、数字，长度不超过 32 个字符，需要保障 AppKey 下唯一。Key 不存在时，设置不成功返回错识提示。 2. Value：字符串，不超过 256 个字符。 。

```
const profile = {
name: 'name',
portraitUri: 'portraitUri',
email: 'email',
birthday: 'birthday',
gender: 1,
location: 'location',
role: 1,
level: 1,
extraProfile: {
key1: 'value1',
key2: 'value2',
},
}
const res = await RongIMLib.updateMyUserProfile(profile);
console.log('更新用户资料结果：', res);
```

获取当前用户信息

使用 getMyUserProfile 方法来获取当前用户信息，用于展示。

```
const res = await RongIMLib.getMyUserProfile();
console.log('获取当前用户资料结果：', res);
```

批量获取他人用户信息

可以使用 getUserProfiles 方法来批量获取他人用户信息，一次最多查询 20 个用户的用户信息

```
const userIds = ['user1', 'user2'];
const res = await RongIMLib.getUserProfiles(userIds);
console.log('批量获取用户资料结果：', res);
```

用户权限

客户端 SDK 提供了用户权限的设置和获取接口，通过 RCUserProfileVisibility 枚举来表示用户权限。

UserProfileVisibility 枚举介绍：

枚举值	用户权限
NOT_SET	未设置：以 AppKey 权限设置为准，默认是此状态。
INVISIBLE	都不可见：任何人都不能搜索到我的用户信息，名称、头像除外。
EVERYONE	所有人：应用中任何用户都可以查看到我的用户信息。

用户权限设置

可以使用 UserProfileVisibility 方法来设置当前用户的用户信息访问权限。

```
const visibility = RongIMLib.UserProfileVisibility.EVERYONE;
const res = await RongIMLib.updateMyUserProfileVisibility(visibility);
console.log('用户权限设置结果：', res);
```

获取用户权限

可以使用 getMyUserProfileVisibility 方法来获取当前用户信息的访问权限。

```
const res = await RongIMLib.getMyUserProfileVisibility();
console.log('用户权限获取结果：', res);
```

用户权限说明

AppKey 默认用户信息访问权限为 **都不可见**，用户级别默认为 **未设置**。都为默认值时， SDK 仅可查看他人的用户名和头像信息。

下面列举了 AppKey 级 权限与 用户级 权限综合说明：

AppKey 级权限	用户级权限	结果
都不可见、仅好友可见、所有人可见	未设置	以 AppKey 级权限设置为准
都不可见、仅好友可见、所有人可见	设置为（都不可见、所有人可见）	以用户级权限设置为准

用户搜索

按用户应用号精确搜索用户信息

可以使用 searchUserProfileByUniqueId 方法来按应用号搜索用户信息。

```
const uniqueId = 'uniqueId';
const res = await RongIMLib.searchUserProfileByUniqueId(uniqueId);
console.log('按用户应用号精确搜索用户信息结果：', res);
```

监听用户信息变更事件

为了接收订阅事件的变更通知，您需要设置订阅监听器。使用 addEventListener 来添加事件监听器。监听器需要在连接之前调用。

订阅用户信息变更

```
/**
* 所订阅的用户的信息变更时会收到此通知。
* 订阅过期后，融云 SDK不会主动通知您，请您自行关注过期时间。
*/
RongIMLib.addEventListener(Events.SUBSCRIBED_USER_STATUS_CHANGE, (event) => {
// 需根据数据中的 subscribeType 来判断是用户资料还是在线状态
console.log('被订阅者状态变更', event)
})
```

订阅关系变更（多端同步）

```
/**
* 当用户在其他设备上的订阅信息发生变更时会收到此通知。
* 这可以用于更新当前设备上的用户状态，确保订阅信息的一致性。
*/
RongIMLib.addEventListener(Events.SUBSCRIBED_RELATION_CHANGE, (event) => {
// 需根据数据中的 subscribeType 来判断是用户资料还是在线状态
console.log('订阅关系变更', event)
})
```

订阅数据同步完成

```
/**
* 订阅数据同步完成。
* 在订阅数据成功同步到设备或系统后会受到此通知，用于执行后续的业务操作。
*/
RongIMLib.addEventListener(Events.SYNC_SUBSCRIBED_USER_STATUS_FINISHED, (event) => {
// 需根据数据中的 subscribeType 来判断是用户资料还是在线状态
console.log('订阅数据同步', event)
})
```

用户资料变更

```
/**
* 用户资料变更。
* 在其他端修改用户资料后会受到此通知，用于执行后续的业务操作。
*/
RongIMLib.addEventListener(Events.OWN_USER_PROFILE_CHANGED, (event) => {
console.log('用户资料变更', event)
})
```

发送消息

更新时间:2024-08-30

本文介绍了如何从客户端发送消息，适用于单聊、群聊、聊天室。

请注意，客户端 SDK 发送消息存在频率限制，每秒最多只能发送 5 条消息。

发送消息

sendMessage 是发送消息的基础接口，可用来发送 IMLib 中的内置类型消息或自定义消息。针对不同的特定消息类型，IMLib 还提供了多个便于调用的语法糖方法。

以调用 sendMessage 往群聊中发送 @张三的文本消息为例。

1. 定义目标群聊会话，并实例化待发送消息。因为发送的是群聊 @ 消息，所以需要添加 `mentionedInfo`。

```
// 定义消息投送目标会话，这里定义一个群组类型会话
const conversation = { conversationType: RongIMLib.ConversationType.GROUP, targetId: '<目标 Id>' }
// 实例化待发送消息，RongIMLib.TextMessage 为内置文本型消息
const message = new RongIMLib.TextMessage({
// 文本内容
content: '文本内容',
//  （可选）消息中附加信息，透传到对端
extra: '消息中附加信息',
// 群组消息中，如果需要发送 @ 消息，可添加 mentionedInfo 字段
mentionedInfo: {
// @ 类型：全部、个人
type: RongIMLib.MentionedType.SINGAL,
// @ 用户列表
userIdList: [zhangsan],
// @ 内容
mentionedContent: ''
}
})
```

2. 构造发送选项 ISendMessageOptions，用于定义发送行为中的一些可选配置。因为发送的是群聊 @ 消息，必须将 `options` 中的 `isMentioned` 设置为 `true`。

```
// 配置属性
const options = {
// 如果需要发送 @ 消息，isMentioned 需设置为 true
isMentioned: true,
// 消息发送前的回调，可以使用此回调返回的 message 用于列表渲染
onSendBefore: (message) => {
console.log('消息发送前的回调', message)
}
}
```

3. 发送消息。如果消息发送成功，可以根据返回结果中的 messageId 字段将列表中的该消息状态改为发送成功。

```
// 发送消息
RongIMLib.sendMessage(conversation, message, options).then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
// 消息发送成功，可以根据返回结果中的 messageId 字段将列表中的该消息状态改为发送成功。
console.log('消息发送成功', res.data)
} else {
console.log('消息发送失败', res.code, res.msg)
}
})
```

4. 如果消息发送失败，可以根据返回结果中的 messageId 字段将列表中的该消息状态改为发送失败，并重发消息。

   客户端遇到消息发送失败时，可能有以下情况：

   1. 消息未送达融云服务器，或送达融云服务端后返回发送失败的情况（其他人也没有收到）。
   2. 某些极端情况下（例如弱网环境），消息可能已成功送达收件方。但发送端未能及时收到融云服务端回执，超时后认为发送失败。此时客户端重发消息后会导致对方重复收到消息。关于该问题的详细解释可参考 FAQ。

5. 重新发送消息。重发消息需与原始消息保持一致。

> 提示
>
> SDK 从 5.5.1 版本开始，支持重发消息时在 options 中传入原消息的 messageId。融云服务端不会去除重复消息，应用层可通过该 ID 自行判断消息是否属于重复消息，并进行相应处理。该 ID 可以从 onSendBefore 回调返回的 message 对象或返回结果中获取。

```
// 定义消息投送目标会话, 这里定义一个群组类型会话
const conversation = { conversationType: RongIMLib.ConversationType.GROUP, targetId: '<目标 Id>' }
// 实例化待发送消息，RongIMLib.TextMessage 为内置文本型消息
const message = new RongIMLib.TextMessage({ content: '文本内容' })
// 配置属性
const options = {
// 重发消息的 messageId, 可以从 onSendBefore 回调返回的 message 对象中 或 返回结果中获取
messageId: 0
}

RongIMLib.sendMessage(conversation, message, options).then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
// 消息发送成功，可以根据返回结果中的 messageId 字段将列表中的该消息状态改为发送成功。
console.log('消息发送成功', res.data)
} else {
// 消息发送失败，可以根据返回结果中的 messageId 字段将列表中的该消息状态改为发送失败。
console.log('消息发送失败', res.code, res.msg)
}
})
```

sendMessage 接口说明

sendMessage 方法接收 conversation、message、options 三个参数。

- conversation 定义目标会话。详见 IConversationOption 。

参数	类型	说明
conversationType	ConversationType	会话类型
targetId	string	接收方 Id

- message 是待发送的消息内容，支持 IMLib 内置消息（例如 RongIMLib.TextMessage）实例，或者通过 RongIMLib.registerMessageType() 实现的自定义消息实例。

- options 定义发送行为中的一些可选配置，如是否可拓展，推送等。参见 ISendMessageOptions

参数	类型	说明
~~isStatusMessage~~	boolean	（已废弃）是否为状态消息（可选项）
disableNotification	boolean	是否发送静默消息（可选项）
pushContent	string	Push 信息（可选项）
pushData	string	Push 通知携带的附加信息（可选项）
isMentioned	boolean	是否为 @ 消息。仅在 conversationType 取值为群组或超级群类型时有效（可选项）
~~mentionedType~~	1 | 2	（已废弃） @ 消息类型，1: @ 所有人 2: @ 指定用户（可选项） 已废弃
~~mentionedUserIdList~~	string[]	（已废弃）被 @ 的用户 Id 列表（可选项）
directionalUserIdList	string[]	用于发送定向消息，仅在 conversationType 取值为群组或超级群类型时有效。注意，如果 SDK 版本 < 5.9.4，仅群组支持定向消息。（可选项）
isVoipPush	boolean	当对方为 iOS 设备且未在线时，其将收到 Voip Push. 此配置对 Android 无影响（可选项）
canIncludeExpansion	boolean	是否允许消息被拓展
expansion	[key: string]: string	消息拓展内容数据（可选项）
isFilerWhiteBlacklist	boolean	黑/白名单（可选项）
pushConfig	IPushConfig	移动端推送配置（可选项）（与 Android、iOS 端的 MessagePushConfig 作用相似）
onSendBefore	Function	消息发送前的回调，支持版本：5.4.0
messageId	number	重发消息时，传要重发消息的 messageId，重发消息的所有参数需与原始消息一致，支持版本：5.5.1

- IPushConfig 参数说明

参数	类型	说明
pushTitle	string	推送标题，在没有设置的情况下：单聊通知标题显示为发送者名称，群聊通知标题显示为群名称；自定义消息，默认不显示标题；
pushContent	string	推送内容
pushData	string	远程推送附加信息
iOSConfig	IiOSPushConfig	（可选项）
androidConfig	IAndroidPushConfig	（可选项）
disablePushTitle	boolean	是否显示推送标题. 仅针对 iOS 平台有效
forceShowDetailContent	boolean	是否强制推送
templateId	string	推送模板id

内置消息类型

IMLib 提供了预定义的文本、语音、图片、Gif、文件等消息类型。

文本消息

ITextMessageBody 参数说明

Key	类型	必填	说明
content	string	是	文本内容

代码示例

```
new RongIMLib.TextMessage({ content: '' })
```

图片消息

IImageMessageBody 参数说明

Key	类型	必填	说明
content	string	是	图片缩略图，应为 Base64 字符串。自行生成缩略图（建议最长边不超过 240 像素），进行 Base64 编码后放入 `content` 中。Base64 字符串长度建议为 5k，最大不超过 10k。注意，部分工具图片转 Base64 输出结果会携带 Data URI 前缀。例如：``。请丢弃其中 Data URI 前缀，仅保留数据部分，例如：`/9j/4AAQSkZJRgABAgAAZABkAAD`。
imageUri	string	是	图片的远程访问地址

代码示例

```
new RongIMLib.ImageMessage({
content: '', // 图片缩略图，应为 Base64 字符串，且不可超过 80KB
imageUri: '' // 图片的远程访问地址
})
```

文件消息

IFileMessageBody 参数说明

Key	类型	必填	说明
name	string	是	文件名称
size	number	是	图片尺寸，单位 Byte
type	string	是	文件类型
fileUrl	string	是	文件远程下载地址

代码示例

```
new RongIMLib.FileMessage({
name: '',
size: 1000,
type: '',
fileUrl: ''
})
```

高清语音

IHQVoiceMessageBody 参数说明

Key	类型	必填	说明
remoteUrl	string	是	远程媒体资源地址
duration	number	是	语音时长，单位 s。请务必保证待发送的语音时长保持在 60s 以内
type	string	否	编码类型，默认 aac。如果您的业务同时使用了 Android/iOS 端的 IMKit SDK，请使用默认的 aac 格式，因为 IMKit 的音频录制、播放只实现了对 aac 格式的支持，默认无法播放其他格式。如果您的 Android/iOS App 会自行处理音频录制、播放，可以按需选用格式。

代码示例

```
new RongIMLib.HQVoiceMessage({
remoteUrl: '<aac 文件地址>',
duration: 60,
})
```

短视频消息

ISightMessageBody 参数说明

Key	类型	必填	说明

Key	类型	必填	说明
sightUrl	string	是	远程视频资源地址。如果您的业务同时使用了 Android/iOS 端的 IMKit SDK，必须使用 H.264 + AAC 编码的文件，因为 IMKit 的短视频录制、播放只实现了该编码组合的支持。
content	string	是	小视频首帧缩略图的 Base64 字符串。自行生成缩略图（建议最长边不超过 240 像素），进行 Base64 编码后放入 content 中。Base64 字符串长度建议为 5k，最大不超过 10k。注意，部分工具图片转 Base64 输出结果会携带 Data URI 前缀。例如：。请丢弃其中 Data URI 前缀，仅保留数据部分，例如：/9j/4AAQSkZJRgABAgAAZABkAAD。
duration	number	是	视频时长。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	number	是	视频大小，单位 Byte
name	number	是	视频名称

代码示例

```
new RongIMLib.SightMessage({
sightUrl: "<视频资源的远程地址>",
content: "<缩略图base64>"
duration: 10,
size: 100,
name: "视频名称"
})
```

GIF 消息

IGIFMessageBody **参数说明**

Key	类型	必填	说明
gifDataSize	number	是	GIF 图片文件大小，单位 Byte
remoteUrl	string	是	GIF 图片的远程存储地址
width	number	是	图片宽度
height	number	是	图片高度

代码示例

```
new RongIMLib.GIFMessage({
gifDataSize: 30,
remoteUrl: '<图片地址>',
width: 300,
height: 200
})
```

引用消息

IReferenceMessageBody **参数说明**

参数	类型	必填	说明
referMsgUserId	string	是	被引用消息的发送用户 Id
referMsg	any	是	引用消息对象
content	string	是	输入的文本消息内容

参数	类型	必填	说明
objName	string	是	被引用的消息类型
referMsgUid	string	是	被引用消息的 messageUId，从 5.9.6 版本开始支持

代码示例

```
new RongIMLib.ReferenceMessage({
referMsgUserId: '<引用消息的用户ID>',
referMsg: {
content: '引用消息文本'
},
referMsgUid: '引用消息的 messageUId',
content: '发送的消息内容',
objName: RongIMLib.MessageType.TEXT
})
```

位置消息

ILocationMessageBody 参数说明

参数	类型	必填	说明
longitude	number	是	经度
latitude	number	是	维度
poi	string	是	位置信息
content	string	是	位置缩略图，图片需要是不带前缀的 Base64 字符串

代码示例

```
new RongIMLib.LocationMessage({
latitude: '<位置的经度>',
longitude: '<位置的纬度>',
poi: '位置信息',
content: '<base64>'
})
```

富文本消息

IRichContentMessageBody 参数说明

参数	类型	必填	说明
title	string	是	标题
content	string	是	简介内容
imageUri	string	是	展示的图片地址
url	string	是	文章链接地址

代码示例

```
new RongIMLib.RichContentMessage({
title: '标题',
content: '内容简介',
imageUri: '<图片地址>',
url: '<文章链接地址>'
})
```

小灰条消息

> 提示
>
> 从 5.8.1 开始支持

IInformationNotificationMessageBody **参数说明**

参数	类型	必填	说明
message	string	是	提示条消息内容
extra	string	否	提示条附加信息

代码示例

```
new RongIMLib.InformationNotificationMessage({
message: '提示条消息内容',
extra: '提示条附加信息',
})
```

命令消息

> 提示
>
> 从 5.8.1 开始支持

ICommandMessageBody **参数说明**

参数	类型	必填	说明
name	string	是	命令名称
data	string	是	命令内容

代码示例

```
new RongIMLib.CommandMessage({
name: '命令名称',
data: '命令内容',
})
```

群组通知消息

> 提示

> 从 5.8.1 开始支持

IGroupNotificationMessageBody 参数说明

参数	类型	必填	说明
operatorUserId	string	是	操作人用户 ID
operation	string	是	群组中各种通知的操作名称
data	string	是	操作数据
message	string	是	消息内容
extra	string	否	扩展信息

代码示例：

```
new RongIMLib.GroupNotificationMessage({
operatorUserId: '操作人用户 ID',
operation: '群组中各种通知的操作名称',
data: '操作数据',
message: '消息内容'
})
```

Emoji 消息

Web 端发送 Emoji 消息时，直接以文本消息类型（TextMessage）发送即可。

- 融云提供 Emoji 插件，内置了 128 个 Emoji 表情的图片。可用于消息输入框的表情选项，也可自行扩展配置。
- 发消息时，必须直接发送 Emoji 原生字符。如：□，转换方法： `symbolToEmoji`。
- Web SDK 接收消息时，接收到的 Emoji 为 Unicode 编码格式，例如：`ef600`。转换后才能正确显示为原生 Emoji。

代码示例：

```
// 定义消息投送目标会话
const conversation = { conversationType: RongIMLib.ConversationType.PRIVATE, targetId: '<目标 Id>' }
// 实例化待发送消息，RongIMLib.TextMessage 为内置文本型消息
const message = new RongIMLib.TextMessage({ content: '□' })
// 发送
RongIMLib.sendMessage(conversation, message).then(res => {})
```

Emoji 插件

- **插件兼容性**

Chrome	Firefox	Safari	IE	Edge	iPhone	Android
30+	30+	10+	7+	✔	iOS 8.0+ 的Safari浏览器以及微信浏览器	4.4+系统的Chrome浏览器以及微信浏览器

- **引入 Emoji 插件**

```
<!-- 非压缩版 -->
<script src="https://cdn.ronghub.com/RongEmoji-2.2.10.js"></script>
<!-- 压缩版 -->
<script src="https://cdn.ronghub.com/RongEmoji-2.2.10.min.js"></script>
```

> 提示
>
> Emoji 代码示例：https://rongcloud.github.io/web-emoji-demo/src/index.html

> 警告
>
> 1. 使用 `import * as RongIMLib from '@rongcloud/imlib-next'` 方式引入 SDK 时表情插件调用需要使用 `window` 前缀。例如：`window.RongIMLib.RongIMEmoji.init()`
> 2. RongEmoji 插件仅支持 cdn 引入方式，暂不支持 npm 引入

- **Emoji 初始化（默认参数）**

```
RongIMLib.RongIMEmoji.init();
```

- **Emoji 初始化（自定义表情配置）**

  `config` 参数说明：

参数	类型	必填	说明	最低版本
size	Number	否	表情大小，默认 24，建议 18 - 58	2.2.6
url	String	否	Emoji 背景图片 url	2.2.6
lang	String	否	Emoji 对应名称语言，默认 zh	2.2.6
extension	Object	否	扩展表情	2.2.6

```
// 表情信息可参考 http://unicode.org/emoji/charts/full-emoji-list.html
var config = {
size: 25,
url: '//f2e.cn.ronghub.com/sdk/emoji-48.png',
lang: 'en',
extension: {
dataSource: {
u1F914: { // 自定义 u1F914 对应的表情
en: 'thinking face', // 英文名称
zh: '思考', // 中文名称
tag: '', // 原生 Emoji
position: '0 0' // 所在背景图位置坐标
}
},
url: '//cdn.ronghub.com/thinking-face.png' // 新增 Emoji 背景图 url
}
};
RongIMLib.RongIMEmoji.init(config);
```

- **获取列表**

```
var list = RongIMLib.RongIMEmoji.list;
/*list => [{
unicode: 'u1F600',
emoji: '',
node: span,
symbol: '[笑嘻嘻]'
}]
*/
```

- **Emoji 转文字**

  在不支持原生 Emoji 渲染时，可显示对应名称，适用于消息输入。

```
var message = '测试 Emoji';
// 将 message 中的原生 Emoji 转化为对应名称
RongIMLib.RongIMEmoji.emojiToSymbol(message);
// => '[笑嘻嘻][露齿而笑]测试 Emoji'
```

- **文字转 Emoji**

  发送消息时，消息体里必须使用原生 Emoji 字符。

```
var message = '[笑嘻嘻][露齿而笑]测试 Emoji';
// 将 message 中的 Emoji 对应名称转化为原生 Emoji
RongIMLib.RongIMEmoji.symbolToEmoji(message);
// => '测试 Emoji'
```

- **Emoji 转 HTML**

  Web SDK 接收消息后，消息体内的原生 Emoji 字符会被解码为对应 Unicode 码，需调用转化方法才能正确显示。

```
var message = '\uf600测试 Emoji';
// 将 message 中的原生 Emoji（包含 Unicode ）转化为 HTML
RongIMLib.RongIMEmoji.emojiToHTML(message);
// => "<span class='rong-emoji-content' name='[笑嘻嘻]'>□</span>测试 Emoji"
```

- **文字 转 HTML**

```
var message = '[露齿而笑]测试 Emoji';
// 将 message 中的 Emoji 对应名称转化为 HTML
RongIMLib.RongIMEmoji.symbolToHTML(message);
// => "<span class='rong-emoji-content' name='[露齿而笑]'>😁</span>测试 Emoji"
```

拓展属性

所有消息均支持 user 与 extra 两种拓展属性，便于业务层基于自身需求透传业务数据信息。

- `user`: 用于在消息中携带用户数据，目前仅支持 `user.id`、`user.name`、`user.portraitUri`、`user.extra` 四个属性定义。
- `extra`: 字符串类型数据，融云不解析，由业务层自行定义序列化与反序列化逻辑。

代码示例

```
new RongIMLib.TextMessage({
content: '',
extra: '',
user: {
id: '',
name: '',
portraitUri: '',
extra: ''
}
})
```

语法糖

发送 FileMessage、ImageMessage、HQVoiceMessage、SightMessage、GIFMessage 五种消息时，在构建消息实例过程中均需要一个远程资源地，这意味着业务层需要将待发送的本地资源先行上传，再构建消息实例，过程较为复杂。

为此，IMLib 实现了相应的语法糖方法以封装本地资源的上传过程、以及消息构建过程，便于业务层更易于集成。

发送文件消息

参数说明

参数	类型	必填	说明
conversation	IConversationOption	是	会话
msgBody	ISendFileMessageOptions	是	要发送的消息内容
hooks	IUploadHooks	否	上传过程中的回调钩子
sendOptions	IUploadMessageOption	否	消息配置

代码示例

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: ''
}
const msgBody = {
file, // 待上传文件
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，非必填
extra: '' // 消息中携带的透传信息，非必填
}
const hooks = {
onProgress (progress) {}, // 上传进度监听，可选
onComplete (fileInfo) { // 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 FileMessage
}
}
const options = {
contentDisposition: 'attachment' // 文件链接在浏览器中展示形式（仅 aws、stc 上传有效） 'inline': 在浏览器中预览，
'attachment': 直接下载。如果不传，html 类型文件会预览，其他类型为直接下载
// ... 其他配置项，可选
},

RongIMLib.sendFileMessage(
conversation,
msgBody,
hooks,
options
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

发送图片消息

参数说明

参数	类型	必填	说明
conversation	IConversationOption	是	会话
msgBody	ISendImageMessageOptions	是	要发送的消息内容
hooks	IUploadHooks	否	上传过程中的回调钩子
sendOptions	IImageMessageOption	否	消息配置

代码示例

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: ''
}
const msgBody = {
file, // 待上传文件
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，非必填
extra: '' // 消息中携带的透传信息，非必填
}
const hooks = {
onProgress (progress) {}, // 上传进度监听，可选
onComplete (fileInfo) { // 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 FileMessage
}
}
const options = {
contentDisposition: 'inline' // 'inline' | 'attachment' , 使用 aws 上传时返回链接在浏览器中的展示形式
// ... 其他配置项，可选
},

RongIMLib.sendImageMessage(
conversation,
msgBody,
hooks,
options
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

发送高清语音消息

> 提示
>
> 如果您的业务同时使用了 Android/iOS 端的 IMKit SDK，请使用默认的 aac 格式，因为 IMKit 的音频录制、播放只实现了对 aac 格式的支持，默认无法播放其他格式。如果您的 Android/iOS App 会自行处理音频录制、播放，可以按需选用格式。

参数说明

参数	类型	必填	说明
conversation	IConversationOption	是	会话
msgBody	ISendHQVoiceMessageOptions	是	要发送的消息内容
hooks	IUploadHooks	否	上传过程中的回调钩子
sendOptions	IUploadMessageOption	否	消息配置

代码示例

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: ''
}
const msgBody = {
file, // 待上传文件
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，非必填
extra: '' // 消息中携带的透传信息，非必填
}
const hooks = {
onProgress (progress) {}, // 上传进度监听，可选
onComplete (fileInfo) { // 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 FileMessage
}
}
const options = {
contentDisposition: 'attachment' // 文件链接在浏览器中展示形式（仅 aws、stc 上传有效） 'inline': 在浏览器中预览，
'attachment': 直接下载。如果不传，html 类型文件会预览，其他类型为直接下载
// ... 其他配置项，可选
},

RongIMLib.sendHQVoiceMessage(
conversation,
msgBody,
hooks,
options
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

发送短视频消息

> 提示
>
> 如果您的业务同时使用了 Android/iOS 端的 IMKit SDK，必须使用 H.264 + AAC 编码的文件，因为 IMKit 的短视频录制、播放只实现了该编码组合的支持。

参数说明

参数	类型	必填	说明
conversation	IConversationOption	是	会话
msgBody	ISendSightMessageOptions	是	要发送的消息内容
hooks	IUploadHooks	否	上传过程中的回调钩子
sendOptions	IUploadMessageOption	否	消息配置

代码示例

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: ''
}
const msgBody = {
file, // 待上传文件
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，非必填
extra: '', // 消息中携带的透传信息，非必填
duration: 10, // 时长
thumbnail: '' // 缩略图
}
const hooks = {
onProgress (progress) {}, // 上传进度监听，可选
onComplete (fileInfo) { // 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 FileMessage
}
}
const options = {
contentDisposition: 'attachment' // 文件链接在浏览器中展示形式（仅 aws、stc 上传有效） 'inline': 在浏览器中预览，
'attachment': 直接下载。如果不传，html 类型文件会预览，其他类型为直接下载
// ... 其他配置项，可选
},

RongIMLib.sendSightMessage(
conversation,
msgBody,
hooks,
options
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

发送输入状态

当需要同步输入状态到指定会话，可以通过调用 sendTypingStatusMessage 方法实现。

代码示例

```
RongIMLib.sendTypingStatusMessage({
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: '<targetId>'
}, RongIMLib.MessageType.TEXT)
```

为什么发送失败后重发消息可能导致消息重复

发送端处于弱网情况下可能出现该问题。

问题场景

A 向 B 发送消息后，消息成功到达服务端，并成功下发到接收者 B。但 A 由于网络等原因可能未收到服务端返回的 ack，导致 A 认为没有发送成功。此时如果 A 重发消息，此时 B 就会收到与之前重复的消息。

解决方案

融云服务端不处理消息重复。从 SDK 5.5.1 版本开始，支持在重发消息携带原消息的 `messageId`。应用层可在消息重复时自行处理。

发送消息后和接收消息时，消息中会携带 `messageId` 字段，这个字段是由客户端生成的消息 ID。从 Web 端重发消息时，在 `options` 参数中传入发送失败消息的 `messageId`。应用层在展示消息前，可根据 `senderUserId` 和 `messageId` 滤除重复消息后再进行展示。

遗留问题

1. 撤回重发的消息时，会出现只能撤回一条，另一条还存在的问题。
2. 收到的消息重复时，即时页面上通过过滤方式使消息显示一条，但未读数依然会显示 2 个。

如何实现转发、群发消息

转发消息：IMLib SDK 未提供转发消息接口。App 实现转发消息效果时，可调用发送消息接口发送。

群发消息：群发消息是指向多个用户发送消息。更准确地说，是向多个 Target ID[?]发送消息，一个 Target ID 可能代表一个用户，一个群组，或一个聊天室。客户端 SDK 未提供直接实现群发消息效果的 API。您可以考虑以下实现方式：

- App 循环调用客户端的发送消息 API。请注意，客户端 SDK 限制发送消息的频率上限为每秒 5 条。
- App 在业务服务端接入即时通讯服务端 API（IM Server API），由 App 业务服务端群发消息。IM Server API 的 发送单聊消息 接口支持单次向多个用户发送单聊消息。您也可以通过 发送群聊消息 、发送聊天室消息 接口实现单次向多个群组或多个聊天室发送消息。

接收消息

更新时间:2024-08-30

App 可监听 SDK 接收的消息，并进行相应的业务操作。

- 建议在应用生命周期内注册消息监听。
- 建议在应用生命周期结束时移除消息监听。

设置消息监听

调用 addEventListener 设置消息接收监听器。所有接收到的消息都会在此接口方法中回调。可在任意位置多次监听。

```
const Events = RongIMLib.Events
const listener = (evt) => {
console.log(evt.messages)
};
RongIMLib.addEventListener(Events.MESSAGES, listener)
```

上方示例的 listener 中返回 IMessageEvent 对象，其 messages 属性中包含接收到的消息（IAReceivedMessage）的列表。

IAReceivedMessage 参数说明

参数	类型	说明
messageType	string	消息类型标识（Object Name），例如 RC:TxtMsg。融云内置消息类型的 Object Name 值参见 消息类型概述。
content	Object	消息内容。例如，如果收到文本消息，content 字段值为 { content: 'text message inner content' }，详见发送消息文档。
senderUserId	string	消息发送者的用户 ID
targetId	string	会话 ID（或称目标 ID），用于标识会话对端。 1. 针对单聊会话，用对端用户的 ID 作为 Target ID，因此发送与接收的所有消息的 Target ID 一定为对端的用户 ID。请注意，本端用户所接收的消息在本地消息数据中存储的 Target ID 也不是当前用户 ID，而是对端用户（消息发送者）的用户 ID。 2. 在群组、聊天室、超级群会话中，Target ID 为对应的群组、聊天室、超级群 ID。 3. 针对系统会话，因客户端用户无法回复系统会话消息，因此不需要 Target ID。
channelId	string	会话的业务标识。
conversationType	ConversationType	会话类型。
sentTime	number	消息在服务器端的发送时间。
receivedTime	number	消息接收时间。
messageUId	string	服务端存储的消息 ID。
messageDirection	MessageDirection	消息方向： 1: 发送，2: 接收。
isPersited	boolean	是否存储。

参数	类型	说明
isCounted	boolean	是否计数。
isOffLineMessage	boolean	是否为离线消息。
isMentioned	boolean	是否为 @ 消息。
disableNotification	boolean	消息是否静默。
isStatusMessage	boolean	是否是状态消息。
canIncludeExpansion	boolean	是否支持消息扩展。
expansion	{ [key: string]: any } | null	消息扩展。
receivedStatus	ReceivedStatus	消息接收状态（Electron 独有字段，5.9.3 版本开始废弃，推荐使用 receivedStatusInfo）。
receivedStatusInfo	IReceivedStatusInfo	消息接收状态详情（Electron 独有字段，支持多种状态并存）。
pushConfig	IPushConfig	推送扩展（与 Android、iOS 端的 MessagePushConfig 作用相似）。
messageId	number	消息本地 ID（Electron 独有字段）。
directedUserIds	string[]	定向消息目标用户列表，如果该列表为空，表示此消息不是定向消息。SDK 从 5.9.6 版本开始支持获取定向消息的目标用户列表。

移除消息监听

调用 removeEventListener 移除消息接收监听器。

```
RongIMLib.removeEventListener(Events.MESSAGES, listener)
```

禁用消息排重机制（仅限 Electron）

消息排重机制会在 SDK 接收单聊、群聊、系统消息时自动去除内容重复消息。当 App 本地存在大量消息，SDK 默认的排重机制可能会因性能问题导致收消息卡顿。因此在接收消息发生卡顿问题时，可尝试关闭 SDK 的排重机制。

为什么接收消息可能出现消息重复

发送端处于弱网情况下可能出现该问题。A 向 B 发送消息后，消息成功到达服务端，并成功下发到接收者 B。但 A 由于网络等原因可能未收到服务端返回的 ack，导致 A 认为没有发送成功。此时如果 A 重发消息，此时 B 就会收到与之前重复的消息（消息内容相同，但 Message UID 不同）。

关闭消息排重

> 提示
>
> SDK 从 5.7.1 版本开始支持关闭消息排重。不支持为聊天室、超级群会话类型关闭消息排重。

请在 SDK 初始化之后，建立 IM 连接之前调用。多次调用以最后一次为准。

```
const enableCheck = false;
RongIMLib.electronExtension.setCheckDuplicateMessage(enableCheck);
```

获取历史消息　　开通服务

更新时间:2024-08-30

从远端获取单群聊历史消息是指从融云服务端获取历史消息，该功能要求 App Key 已启用融云提供的单群聊消息云端存储服务。您可以在控制台 IM 服务管理 页面为当前使用的 App Key 开启服务。如果使用生产环境的 App Key，请注意仅 IM 旗舰版或 IM 尊享版可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。

提示：请注意区分历史消息记录与离线消息[?]。融云针对单聊、群聊、系统消息默认提供最多 7 天（可调整）的离线消息缓存服务。客户端上线时 SDK 会自动收取离线期间的消息，无需 App 层调用 API。详见管理离线消息存储配置。

获取历史消息

> 提示
>
> Web 端不具备持久化的数据存储能力，无法在本地持久化存储历史消息记录与会话列表，因此需要从融云服务端获取数据。请确保已开通单群聊消息云端存储服务。

调用 getHistoryMessages 可以拉取指定某个会话的历史消息记录。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户Id>"
}

// 从当前时间开始向前查询
const option = {
timestamp: 0,
count: 20,
order: 0
}

RongIMLib.getHistoryMessages(conversation, option).then(res => {
if (res.code === 0) {
console.log(res.data.list)
console.log(res.data.hasMore)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话
options	GetHistoryMessageOption	可选项。详见下方 GetHistoryMessageOption 说明

- **GetHistoryMessageOption** 说明

参数	类型	说明
timestamp	number	（可选项）用于控制分页查询消息的边界。查询以此时间戳为基准，根据 `order` 决定查询早于或晚于该时间戳的消息。`timestamp` 传 `0` 表示从当前时间开始获取。
count	number	（可选项）获取消息的数量。如果 SDK < 5.7.4，范围为 [1-20]；如果 SDK ≧ 5.7.4，范围为 [1-100]。默认值 `20`。
order	number	（可选项）查询消息的方向，值为 `0` 或 `1`。`0` 表示降序，即按消息发送时间（`timestamp`）递减的顺序，获取发送时间早于 timestamp 的消息。`1` 表示升序，即按消息发送时间（`timestamp`）递增的顺序，获取发送时间晚于 timestamp 的消息。

获取第一条未读消息信息

> 提示
>
> 本功能在 5.9.0 版本开始支持。
> 当没有未读消息时，返回的 data 为 null, 当第一条未读消息被撤回时，此信息不会更新。

调用 getFirstUnreadMessageInfo 可以获取第一条未读消息信息。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户Id>"
}

RongIMLib.getFirstUnreadMessageInfo(conversation).then(res => {
if (res.code === 0) {
console.log(res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话

获取历史消息（Electron）

更新时间:2024-08-30

> 本文档仅适用于 Electron 解决方案，仅限于配合 Electron 模块 （@rongcloud/electron 与 @rongcloud/electron-renderer）使用。

开通服务

从远端获取单群聊历史消息是指从融云服务端获取历史消息，该功能要求 App Key 已启用融云提供的单群聊消息云端存储服务。您可以在控制台 IM 服务管理 页面为当前使用的 App Key 开启服务。如果使用生产环境的 App Key，请注意仅 **IM 旗舰版**或 **IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。

提示：请注意区分历史消息记录与离线消息[?]。融云针对单聊、群聊、系统消息默认提供最多 7 天（可调整）的离线消息缓存服务。客户端上线时 SDK 会自动收取离线期间的消息，无需 App 层调用 API。详见管理离线消息存储配置。

从本地数据库中获取消息

获取会话中所有类型的消息

调用 getHistoryMessages 可以拉取指定某个会话的历史消息记录。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户Id>"
}

// 从当前时间开始向前查询
const option = {
timestamp: 0,
count: 20,
order: 0
}

RongIMLib.getHistoryMessages(conversation, option).then(res => {
if (res.code === 0) {
console.log(res.data.list)
console.log(res.data.hasMore)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话
options	GetHistoryMessageOption	参数配置。详见下方 `GetHistoryMessageOption` 说明

- **GetHistoryMessageOption 说明**

参数	类型	说明
timestamp	number	（可选项）用于控制分页查询消息的边界。查询以此时间戳为基准，根据 `order` 决定查询早于或晚于该时间戳的消息。`timestamp` 传 `0` 表示从当前时间开始获取。
count	number	（可选项）获取消息的数量。如果 SDK < 5.7.4，范围为 [1-20]；如果 SDK ≧ 5.7.4，范围为 [1-100]。默认值 `20`。
order	number	（可选项）查询消息的方向，值为 `0` 或 `1`。`0` 表示降序，即按消息发送时间（`timestamp`）递减的顺序，获取发送时间早于 timestamp 的消息。`1` 表示升序，即按消息发送时间（`timestamp`）递增的顺序，获取发送时间晚于 timestamp 的消息。

获取会话中指定类型的消息

调用 electronExtension.getHistoryMessagesByMessageTypes 可以拉取指定某个会话指定消息类型的历史消息记录。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户Id>"
}

// 从当前时间开始向前查询
const options = {
timestamp: 0,
count: 20,
order: 0,
messageTypes: ['RC:TxtMsg']
}

RongIMLib.electronExtension.getHistoryMessagesByMessageTypes(conversation, option).then(res => {
if (res.code === 0) {
// res.data.messages 从 5.6.1 开始废弃，5.6.1 及之后版本请使用 res.data.list
console.log(res.data.list)
console.log(res.data.hasMore)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话
options	IGetHistoryMessagesByTypesOption	参数配置

按时间戳获取消息列表

> 提示
>
> 本方法 5.9.8 版本支持

调用 [electronExtension.getMessagesAroundTimestamp] 可以拉取会话某个时间点的 beforeCount 条和后 afterCount 条数量的消息。

传递的时间戳如果为消息的时间则可以拉取到这条消息及这条消息前 beforeCount 条和后 afterCount 条消息。获得的总数量为 beforeCount + afterCount + 1，1 是根据传递时间戳查询到的消息。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户Id>"
}

const options = {
timestamp: 0, // 传递时间戳，0 为当前时间
afterCount: 10,
beforeCount: 10,
}

RongIMLib.electronExtension.getMessagesAroundTimestamp(conversation, option).then(res => {
if (res.code === 0) {
console.log(res.data)
}
})
```

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话
options	Object	查询参数配置

option 说明

参数	类型	说明
timestamp	number	查询时间戳
beforeCount	number	向前查询消息的条数
afterCount	number	向后查询消息的条数

获取远端历史消息

调用 getRemoteHistoryMessages 可以拉取指定某个会话的远端历史消息记录。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户Id>"
}

// 从当前时间开始向前查询
const option = {
timestamp: 0,
count: 20,
order: 0
}

RongIMLib.getRemoteHistoryMessages(conversation, option).then(res => {
if (res.code === 0) {
console.log(res.data.list)
console.log(res.data.hasMore)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话

参数	类型	说明
options	GetHistoryMessageOption	参数配置

获取本地与远端历史消息

getContinuousMessages 方法与 getRemoteHistoryMessages 的区别是 getContinuousMessages 会先查询指定会话存储本地数据库的消息，当本地消息无法满足查询条件时，再查询在单群聊消息云端存储中的历史消息，以返回连续且相邻的消息对象列表。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户Id>"
}

// 从当前时间开始向前查询
const option = {
timestamp: 0,
count: 20,
order: 0
}

RongIMLib.electronExtension.getContinuousMessages(conversation, option).then(res => {
if (res.code === 0) {
console.log(res.data.list)
console.log(res.data.hasMore)
// 如果您想继续查询更多历史消息，建议使用 res.data.timestamp 作为下次调用的 option.timestamp
console.log(res.data.timestamp)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话
options	GetHistoryMessageOption	参数配置

插入消息（Electron）

更新时间:2024-08-30

> 本文档仅适用于 Electron 解决方案，仅限于配合 Electron 模块 （@rongcloud/electron 与 @rongcloud/electron-renderer）使用。

SDK 支持在 Electron 的本地数据库中插入消息。本地插入的消息不会实际发送给服务器和对方。

插入单条消息

在本地会话中插入一条发送方向或接收方向的消息。消息进入本地数据库，但不会发送给服务器和对方。不支持聊天室会话类型。以下示例使用 electronExtension.insertMessage 方法，插入 IAReceivedMessage 。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>"
}

const message = {
senderUserId: '<userId>',
messageType: RongIMLib.MessageType.TEXT,
content: {
content: '插入一条消息'
},
messageDirection: RongIMLib.MessageDirection.SEND
}

RongIMLib.electronExtension.insertMessage(conversation, message).then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
// 消息插入成功，返回 IAReceivedMessage 类型的消息数据。
console.log('消息插入成功', res.data)
} else {
console.log('消息插入失败', res.code, res.msg)
}
})
```

- 插入 IAReceivedMessage 时，可通过 `messageDirection` 控制消息方向。
- 从 5.6.1 开始，SDK 支持插入 BaseMessage 。插入 `BaseMessage` 时不支持控制消息方向，SDK 默认插入发送方向的消息。
- 插入本地数据库的消息如需支持消息扩展功能（详见消息扩展），必须使用 IAReceivedMessage ，并打开消息的可扩展属性（`canIncludeExpansion`）。

参数	类型	说明
conversation	IConversationOption	会话
message	BaseMessage （推荐） 或 IAReceivedMessage	插入的消息。从 5.6.1 开始支持插入 `BaseMessage`。
options	IInsertOptions	选项

批量插入消息

提示

- 从 5.7.2 版本开始支持该功能。
- 每次最多插入 500 条消息，不支持聊天室和超级群。

在本地数据库批量插入消息。

```
const messages = [{
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: '<目标用户ID>'
senderUserId: '<userId>',
messageType: RongIMLib.MessageType.TEXT,
content: {
content: '插入一条消息'
},
messageDirection: RongIMLib.MessageDirection.SEND,
messageUId: '<消息唯一值>',

}]

const checkDuplicate = false

RongIMLib.electronExtension.batchInsertMessage(messages, checkDuplicate).then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
// 消息插入成功
console.log('批量插入成功', res.data)
} else {
// 如果消息格式不正确，res.msg 返回该消息在数组中的下标号
console.log('批量插入失败', res.code, res.msg)
}
})
```

参数	类型	说明
messages	IInsertMessage[]	需要批量插入的消息数组
checkDuplicate	Boolean	消息是否排重

删除消息

更新时间:2024-08-30

SDK 支持从客户端直接删除单聊会话、群聊会话存储在融云服务端的历史消息。您可以删除指定的一条或一组消息，或删除会话中早于指定时间点的消息。

提示

- 仅适用于 App Key 已开通单群聊消息云端存储 的 App。开通消息云存储服务后，用户的消息会保存在融云服务端（默认 6 个月）。
- 针对单聊会话、群聊会话，如果通过任何接口以传入时间戳的方式删除远端消息，服务端默认不会删除对应的离线消息补偿（该机制仅会在打开多设备消息同步开关后生效）。此时如果换设备登录或卸载重装，仍会因为消息补偿机制获取到已被删除的历史消息。如需彻底删除消息补偿，请提交工单，申请开通**删除服务端历史消息时同时删除多端补偿的离线消息**。如果以传入消息对象的方式删除远端消息，则服务端一定会删除消息补偿中的对应消息。
- 针对单聊会话、群聊会话，如果 App 的管理员或者某普通用户希望在所有会话参与者的历史记录中彻底删除一条消息，应使用撤回消息功能。消息成功撤回后，原始消息内容会在所有用户的本地与服务端历史消息记录中删除。
- 客户端 SDK 不提供删除聊天室消息的接口。当前用户的聊天室本地消息在退出聊天室时会被自动删除。开通**聊天室消息云端存储服务**后，如需清除全部用户的聊天室历史消息，可使用服务端 API 清除消息。

通过消息删除

调用 deleteMessages 可根据消息的 ID、时间戳和方向（发送或接收）从服务端删除指定单个会话中的一条或一组消息。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>"
}
RongIMLib.deleteMessages(conversation, [
{
messageUId: "BS4O-P5AO-D1O6-9GPP",
sentTime: 1632728405345,
messageDirection: RongIMLib.MessageDirection.SEND
},
{
messageUId: "BS4O-QEBR-VJM6-9GPP",
sentTime: 1632728573423,
messageDirection: RongIMLib.MessageDirection.SEND
}
]).then(res => {
if (res.code === 0) {
console.log('删除成功')
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversationOption	会话。不支持聊天室。
messages	消息数组	要删除的消息数组。详见 messages 结构说明。

- messages 结构说明

参数	类型	说明
messageUId	string	消息 UId
sentTime	number	消息发送时间
messageDirection	MessageDirection	消息方向

通过时间戳删除

调用 clearHistoryMessages 可根据时间戳从服务端删除指定会话中早于该时间戳的消息。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>"
}
RongIMLib.clearHistoryMessages(conversation, <时间戳>).then(res => {
if (res.code === 0) {
console.log('清除成功')
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversationOption	会话。不支持聊天室。
timestamp	number	时间点，该时间点前的消息将被删除

删除消息（Electron）

更新时间:2024-08-30

本文档仅适用于 Electron 解决方案，仅限于配合 Electron 模块 （@rongcloud/electron 与 @rongcloud/electron-renderer）使用。

提示

- App 用户的单聊会话、群聊会话、系统会话的消息默认仅存储在本地数据库中，仅支持从本地删除。如果 App（App Key/环境）已开通单群聊消息云端存储，该用户的消息还会保存在融云服务端（默认 6 个月），可从远端历史消息记录中删除消息。
- 针对单聊会话、群聊会话，如果通过任何接口以传入时间戳的方式删除远端消息，服务端默认不会删除对应的离线消息补偿（该机制仅会在打开多设备消息同步开关后生效）。此时如果换设备登录或卸载重装，仍会因为消息补偿机制获取到已被删除的历史消息。如需彻底删除消息补偿，请提交工单，申请开通删除服务端历史消息时同时删除多端补偿的离线消息。如果以传入消息对象的方式删除远端消息，则服务端一定会删除消息补偿中的对应消息。
- 针对单聊会话、群聊会话，如果 App 的管理员或者某普通用户希望在所有会话参与者的历史记录中彻底删除一条消息，应使用撤回消息功能。消息成功撤回后，原始消息内容会在所有用户的本地与服务端历史消息记录中删除。
- 客户端 SDK 不提供删除聊天室消息的接口。当前用户的聊天室本地消息在退出聊天室时会被自动删除。开通聊天室消息云端存储服务后，如需清除全部用户的聊天室历史消息，可使用服务端 API 清除消息。

本文档描述了如何删除客户端本地消息。

通过会话删除

提示

从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.clearMessages 从本地消息数据库中删除指定会话所有消息。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>"
}
RongIMLib.electronExtension.clearMessages(conversation).then(res => {
if (res.code === 0) {
console.log('成功清理此会话下的消息');
} else {
console.log(res.code, res.msg);
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversationOption	会话

通过时间戳删除

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.deleteMessagesByTimestamp 根据时间戳，从本地消息数据库中删除单个会话中早于该时间戳的消息数据。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>"
}
const timestamp = 1632728573423

RongIMLib.electronExtension.deleteMessagesByTimestamp(conversation, timestamp, false).then(res => {
if (res.code === 0) {
console.log('成功清理此会话下的消息');
} else {
console.log(res.code, res.msg);
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversationOption	会话
timestamp	number	指定删除该时间戳之前的消息
cleanSpace	boolean	是否清理数据条目所使用的磁盘空间

> 提示
>
> 清理磁盘空间会阻塞进程且耗时较长，不推荐使用。 数据在被抹除的情况下，未清理的磁盘空间会在后续存储操作中复用，且对数据查询无影响。

通过消息 ID 删除

提示

从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.deleteMessages 通过消息 ID 从本地消息数据库中删除指定的一条或一组消息数据。删除多条消息时，请确保消息 ID 均属于同一会话。

```
const messageIds = [<messageId>]

RongIMLib.electronExtension.deleteMessages(messageIds).then(res => {
if (res.code === 0) {
console.log('成功删除消息');
} else {
console.log(res.code);
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
messageIds	number[]	消息 Id 数组，必须是同一会话的消息

撤回消息

更新时间:2024-08-30

撤回已发送成功的消息。

默认情况下，融云对撤回消息的操作者不作限制。如需限制，可考虑以下方案：

- App 客户端自行限制撤回消息的操作者。例如，不允许 App 业务中的普通用户撤回他人发送的消息，允许 App 业务中的管理员角色撤回他人发送的消息。
- 如需避免用户撤回非本人发送的消息，可以提交工单 申请打开**IMLib SDK 只允许撤回自己发送的消息**。从融云服务端进行限制，禁止用户撤回非本人发送的消息。

调用 recallMessage 撤回指定消息。

> 提示
>
> Electron 端
>
> 融云定义了 ObjectName 为 RC:RcNtf 的撤回通知消息。
>
> recallMessage 会替换聊天记录中的原始消息为一条 objectName 为 RC:RcNtf 的撤回通知消息

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: '<目标用户ID>',
}
RongIMLib.recallMessage(conversation, {
messageUId: 'BS4O-QEBR-VJM6-9GPP',
sentTime: 1632728573423,
})
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversation	会话。不支持系统会话。
options	消息相关	要撤回消息的相关信息。详见下方 `options` 参数说明。

- `options` 参数说明

参数	类型	是否必填	说明
messageUId	string	是	消息的 UId
sentTime	number	是	消息的发送时间
user	IUserProfile	否	撤回消息携带用户信息
disableNotification	boolean	否	是否发送静默消息
pushConfig	IPushConfig	否	移动端推送配置（与 Android、iOS 端的 MessagePushConfig 作用相似）
extra	string	否	撤回消息携带扩展信息。Web SDK 从 5.3.0 版本开始支持该参数。
isDelete	boolean	否	指定移动端接收方是否需要从本地删除原始消息记录。为 `false` 时，移动端不会删除原始消息记录，会将消息内容替换为撤回提示（小灰条通知）。为 `true` 时，移动端会删除原始消息记录，不显示撤回提示（小灰条通知）。Web SDK 从 5.3.1 版本开始支持该参数。

搜索消息（Electron）

更新时间:2024-08-30

本文档仅适用于 Electron 解决方案，仅限于配合 Electron 模块 （@rongcloud/electron 与 @rongcloud/electron-renderer ）使用。

本文档描述了如何搜索存储在客户端本地的消息。

根据关键字搜索本地消息

提示

从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.searchMessages 根据关键字搜索指定会话中的消息

- 文本类型消息只支持搜索 content，文件类型消息只支持搜索 name
- 自定义消息根据 registerMessageType 的 searchProps 参数决定
- 引用消息不支持搜索

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>",
channelId: ""
}
const keyword = '搜索关键字'
const startTime = Date.now()
const count = 10

RongIMLib.electronExtension.searchMessages(conversation, keyword, startTime, count).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversationOption	会话
keyword	string	关键字
startTime	number	搜索时间, 搜索该时间之前的消息
count	number	获取的数量

参数	类型	说明
messageTypes	string[]	指定消息类型，支持：文本（RC:TxtMsg）、文件（RC:FileMsg）、引用消息（RC:ReferenceMsg）、自定义消息，5.9.8 版本开始支持

- conversation 字段说明

参数	类型	说明
conversationType	IConversationType	会话类型
targetId	string	会话 ID
channelId	number	频道 ID，限定搜索指定频道内的消息，如果不传，则在所有频道中进行搜索

在指定时间范围内搜索本地消息

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.searchMessageInTimeRange 在指定会话的所有频道中搜索时间范围内的消息

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>"
}
const option = {
keyword: 'hello',
startTime: 0,
endTime: 1632728573423,
offset: 0,
limit: 5
}
RongIMLib.electronExtension.searchMessageInTimeRange(conversation, option).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	Object	会话，详见下方 conversation 参数说明。
option	ISearchMessageInTimeRangeOption	搜索参数，详见下方 option 参数说明。

- conversation 参数说明

参数	类型	说明
conversationType	IConversationType	会话类型
targetId	string	会话 ID

- option 参数说明

参数	类型	说明
keyword	string	搜索关键字
startTime	number	开始时间
endTime	number	截止时间
offset	number	分页搜索时的偏移量，默认为0
limit	number	每页的数量，默认为 5

根据用户 ID 搜索本地消息

> 提示
>
> 从 SDK 5.7.10 开始支持该接口。

调用 electronExtension.searchMessagesByUser 在指定会话的所有频道中搜索时间范围内的消息

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<会话 ID>",
channelId: ""
}
const userId = '用户 ID'
const startTime = Date.now()
const count = 10

RongIMLib.electronExtension.searchMessagesByUser(conversation, userId, startTime, count).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversationOption	会话
userId	string	用户 id
startTime	number	搜索时间, 搜索该时间之前的消息
count	number	获取的数量

单聊消息回执

更新时间:2024-08-30

当 A 给 B 发送一条消息，B 用户调用发送阅读回执接口之后，A 用户可在回执监听中收到已读通知。

实现思路

通过以下步骤，即可以实现消息已读回执功能。Web 平台与 Electron 平台的实现流程有差异：

- Web

1. 在用户查看了某单聊会话的未读消息后，调用 sendReadReceiptMessage 发送已读回执。
2. 通过 addEventListener 设置消息回执监听器。
3. SDK 在收到消息已读回执时派发 `READ_RECEIPT_RECEIVED` 事件。在 Web 平台，SDK 不会记录每个会话和消息的已读状态，您需要在 localStorage 中记录每个会话的已读时间。
4. 在渲染消息列表时，根据 localStorage 中时间判断消息是否已读，更新对应的消息界面。

- Electron

1. 在用户查看了某单聊会话的未读消息后，调用 sendReadReceiptMessage 发送已读回执。
2. 通过 addEventListener 设置消息回执监听器。
3. SDK 在收到消息已读回执时派发 `READ_RECEIPT_RECEIVED` 事件。
4. 在 Electron 平台，收到 `READ_RECEIPT_RECEIVED` 通知时，您需要调用 `electronExtension.setMessageStatusToRead` 方法更新消息已读状态，并更新对应的消息界面。

> 提示
>
> 调用 sendReadReceiptMessage 接口不会清除未读数，需单独调用清除未读数接口。

发送回执

通过 sendReadReceiptMessage 发送某个会话中的消息阅读回执。

```
const messageUId = 'BS4O-QEBR-VJM6-9GPP';
const timestamp = 1632728573423;

RongIMLib.sendReadReceiptMessage('targetId', messageUId, timestamp).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
targetId	string	接收回执方的用户 ID
messageUId	string	已读消息的UId
timestamp	number	会话中已读的最后一条消息的发送时间戳, 即 Message 的 sentTime

设置消息回执监听器

通过 addEventListener 设置消息回执监听器，用于接收消息回执。

```
const Events = RongIMLib.Events
function onReadReceiptReceived({conversation, messageUId, sentTime}) {
console.log(conversation, messageUId, sentTime)
}
RongIMLib.addEventListener(Events.READ_RECEIPT_RECEIVED, onReadReceiptReceived);
```

移除消息回执监听

应用生命周期结束时，应当通过 removeEventListener 移除已经绑定的事件。

```
RongIMLib.removeEventListener(Events.READ_RECEIPT_RECEIVED, onReadReceiptReceived);
```

群聊消息回执

更新时间:2024-08-30

群聊支持已读回执功能。本端用户向群组中发送消息后，可请求获取该条消息的已读数据。

实现思路

1. 群成员 A 往群组发送了一条消息。消息将到达融云服务端，并由融云服务端发送到目标群组中。
2. 在消息发出后，建议您在 UI 上向用户提供一个请求查看已读状态的按钮（展示 1～2 分钟）。发件人点击该按钮，表示需要获取消息已读数据，即发起群组消息已读回执请求（sendReadReceiptRequest）。

> 提示
>
> 在群聊场景中，建议让您的用户自行发起请求，按需获取消息的已读数据。如果针对每条消息都发起已读回执请求，且每个群成员都发送回执响应，可能会导致您的应用下出现大量的已读回执响应，影响用户的使用体验。

3. 群组中其他成员监听到 A 发起的群消息已读回执请求（MESSAGE_RECEIPT_REQUEST），并在该群组中发送回执响应（sendReadReceiptResponseV2）。例如，群成员 B 阅读 A 发送的群消息后即可发起响应。
4. A 用户监听群组中的回执响应。

- 5.9.3 版本之前，用户需在本地维护该信息，比如 `user_targetId_type:{'messageUid':['userId1', 'userId2']}`。在 App 展示历史消息时，可先与缓存的数据进行匹配后展示，例如调整消息已读数。具体方案可根据实际场景实现。
- 5.9.4 版本之后，SDK 内部维护了已读回执请求信息，并在历史消息列表里携带已读回执信息：readReceiptInfo, 在 Web 平台和 Electron 平台实现不同之处：
  - 在 Web 平台，SDK 内部会缓存每个群会话的最新 30 个已读回执信息（以已读回执请求的发送时间为准），缓存有效期为 24H。在发送消息回执响应时无需传 messageList 字段，SDK 会把该会话下的所有请求组织好然后发送响应。
  - 在 Electron 平台，用户需先获取历史消息，取出 message.messageDirection = MessageDirection.RECEIVE , readReceiptInfo.isReceiptRequestMessage = true 并且 readReceiptInfo.hasRespond = false 的数据组织成参数 messageList 来发送回执响应。

群聊已读回执时序图

发起已读回执请求

群成员希望获取已发消息的阅读状态数据，可发起群组消息已读回执请求。

调用 sendReadReceiptRequest 方法发起消息已读回执请求。

```
RongIMLib.sendReadReceiptRequest('targetId', 'BS4S-U34I-T4G6-9GPP')
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

参数	类型	说明
targetId	string	消息所属的会话 targetId，即群组 Id
messageUId	string	待查询的消息 messageUId，例如 BS4S-U34I-T4G6-9GPP。

响应已读回执请求

> 提示
>
> 从 5.1.1 版本开始，废弃 sendReadReceiptResponse() 方法，新增 sendReadReceiptResponseV2 方法。

群组中其他成员监听到群消息已读回执请求后，消息接收者需要响应该请求，通知对方（消息发送者）已阅读此消息。可以一次响应同一会话中某个用户的多条消息。

调用 sendReadReceiptResponseV2 方法响应消息已读回执请求。

```
const messageList = {
'<用户Id1>': ['messageUId1', 'messageUId2'],
'<用户Id2>': ['messageUId3', 'messageUId4']
};

RongIMLib.sendReadReceiptResponseV2('<targetId>', messageList).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	必填	说明
targetId	string	是	群 ID
messageList	{[senderUserId: string]: string[]}	是	会话中需要发送回执的消息列表, 在 5.9.4 版本起，Web 平台无需传此参数
channelId	string	否	

监听消息回执请求事件

通过 addEventListener 设置消息回执监听器，用于接收消息回执请求。

```
const Events = RongIMLib.Events
function onMessageReceiptRequest({conversation, messageUId, senderUserId}) {
// conversation 群组会话
// messageUId 需要回执的消息UId
// senderUserId 消息发送者Id
console.log(conversation, messageUId, senderUserId)
}
RongIMLib.addEventListener(
Events.MESSAGE_RECEIPT_REQUEST,
onMessageReceiptRequest
)
```

监听消息回执响应事件

通过 addEventListener 设置针对消息回执响应的监听器，用于接收消息回执。

```
const Events = RongIMLib.Events
function onMessageReceiptResponse({conversation, receivedUserId, messageUIdList}) {
// conversation 群组会话
// receivedUserId 为消息接收者，即谁响应了之前发送的消息回执请求
// messageUIdList 为 消息接收者已查看了的消息UId列表
console.log(conversation, receivedUserId, messageUIdList)
}
RongIMLib.addEventListener(
Events.MESSAGE_RECEIPT_RESPONSE,
onMessageReceiptResponse
)
```

消息扩展

更新时间:2024-08-30

消息扩展功能可以对已发送的消息增加、修改、删除扩展信息。

消息扩展功能可为消息增加基于 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，由发送消息时的可扩展（canIncludeExpansion）属性决定，发送后无法修改该属性。因此消息发送前需要将消息设置为可扩展，才能对该条消息进行扩展信息添加、删除等操作。

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，超出部分会被丢弃删除。

消息扩展（Key、Value 扩展信息）会被存储。如已开通**历史消息云存储功能**，从服务端获取的历史消息也会携带已设置的扩展信息。

> 提示
>
> 消息扩展仅支持单聊、群聊、超级群会话类型。不支持聊天室和系统会话。

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 `canIncludeExpansion` 属性设置为可扩展，同时设置用于标识订单状态的 Key 和 Value。例如，在用户未确认前，可用一对 Key/Value 表示该订单状态为「未确认」。
2. 用户点击确认（或其他确认操作）该订单消息后，订单消息状态需要变更为「已确认」。此时，可通过 `updateMessageExpansion` 更新此条消息的扩展信息，标识为已确认状态，同时更改本地显示的消息样式。
3. 发送方通过消息扩展状态监听，获取指定消息的状态变化，根据最新扩展信息显示最新的订单状态。

消息评论、礼物领取可参照以上实现思路：

- **礼物领取**：可通过消息扩展改变消息显示状态实现。例如，向用户发送礼物，默认为未领取状态，用户点击后可设置消息扩展为已领取状态。
- **消息评论**：可通过设置原始消息扩展信息的方式添加评论信息。

打开消息的可扩展属性（仅发送前）

发送新消息前，可通过 SendMessage 的 `options.canIncludeExpansion` 属性打开或关闭该条消息的可扩展属性。必须在发送消息前设置该属性。

您还可以在发送消息前设置扩展信息数据 `options.expansion`。下例中发送的消息会携带 `expansion` 属性中的扩展信息数据。

```
// 定义消息投送目标会话，这里定义一个群组类型会话
const conversation = { conversationType: RongIMLib.ConversationType.GROUP, targetId: '<目标 Id>' }
// 实例化待发送消息，RongIMLib.TextMessage 为内置文本型消息
const message = new RongIMLib.TextMessage({ content: '文本内容' })
// 配置属性
const options = {
// 打开消息的可扩展属性
canIncludeExpansion: true,
expansion: { key1: 'tom', key2: 'jerry' }
}

RongIMLib.sendMessage(conversation, message, options).then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
// 消息发送成功，可以根据返回结果中的 messageId 字段将列表中的该消息状态改为发送成功。
console.log('消息发送成功', res.data)
} else {
// 消息发送失败，可以根据返回结果中的 messageId 字段将列表中的该消息状态改为发送失败。
console.log('消息发送失败', res.code, res.msg)
}
})
```

针对 Electron 平台，可能存在 App 先在本地插入消息，再发送本地已插入的消息的业务场景（例如 App 业务要求在发送前审核消息内容）。如需使用消息扩展功能，请注意以下事项：

- 插入消息时，需要设置 `canIncludeExpansion` 属性。插入成功后不支持再修改可扩展属性开关的状态。详见插入消息。
- 发送时，需要从成功插入的消息中（IAReceivedMessage）获取本地消息 ID（`messageId`）。发送成功后，SDK 会根据 `SendMessage` 的 `options.expansion` 中的扩展数据刷新本地数据库中消息的扩展数据。

更新消息扩展信息

调用 updateMessageExpansion 设置、更新消息扩展信息。对端可通过监听收到扩展数据更新。

- 仅支持单聊、群聊会话类型，不支持聊天室类型。
- 每次设置消息扩展将会产生内置通知消息，频繁设置扩展会产生大量消息。
- 仅当发送消息时指定 `canIncludeExpansion` 值为 `true`，才可对消息进行拓展

```
RongIMLib.updateMessageExpansion({ key1: 'val1', key2: 'val2' }, message).then(res => {
if (res.code === 0) {
console.log(res.code, '更新成功')
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
expansion	Map	要更新的消息扩展信息键值对，类型是 HashMap。 • Key 支持大小写英文字母、数字、 特殊字符`+ = - _` 的组合方式，不支持汉字。最大 32 个字符。 • （SDK < 5.3.0）Value 最大 64 个字符 • （SDK ≧ 5.3.0）Value 最大 4096 个字符
message	IAReceivedMessage	通过接收在线消息或拉取历史消息从 IMLib 取得的消息实例。

删除消息扩展信息

调用 removeMessageExpansionForKey 删除消息扩展信息。

```
RongIMLib.removeMessageExpansionForKey(['key1', 'key2'], messsage).then(res => {
if (res.code === 0) {
console.log(res.code, '删除成功')
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
keyArray	List	消息扩展信息中待删除的 key 的列表，类型是 ArrayList。
message	IAReceivedMessage	通过接收在线消息或拉取历史消息从 IMLib 取得的消息实例

监听消息扩展通知

可通过监听 Events.EXPANSION 事件来捕获消息的拓展信息变更通知

```
RongIMLib.addEventListener(RongIMLib.Events.EXPANSION, ({ updatedExpansion, deletedExpansion }) => {
console.log('拓展信息更新：', updatedExpansion);
console.log('拓展信息删除：', deletedExpansion);
})
```

自定义消息

更新时间:2024-08-30

除了使用 SDK 内置消息外，开发者可根据自己的业务需求自定义消息，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。

注意事项：

- 注册自定义消息代码必须在发送、接收该自定义消息之前
- 推荐将所有注册自定义消息代码放在 `init` 方法之后，`connect` 方法之前
- 注册的消息类型名，必须多端一致，否则消息无法互通
- 请勿使用 `RC:` 开头的类型名，以免和 SDK 默认的消息名称冲突

注册自定义消息

使用 RongIMLib.registerMessageType 方法来注册自定义消息，该方法会返回自定义消息构造函数。

```
const PersonMessage = RongIMLib.registerMessageType('s:person', true, true, [], false)
```

参数	类型	说明
messageType	string	消息类型
isPersited	boolean	是否存储
isCounted	boolean	是否计数
searchProps	string[]	搜索字段，web 端无需设置，搜索字段值设置为数字时取值范围为 (-Math.pow(2, 64), Math.pow(2, 64)) 且为整数
isStatusMessage	boolean	是否是状态消息。状态消息不存储、不计数，接收方在线时才能收到。

> 提示
>
> 1. `RongIMLib.registerMessageType` 必须在 `connect` 之前调用，否则可能造成收取消息的行为异常。
> 2. 请尽量把应用中所有涉及到的自定义消息统一一次注册，且同类型消息仅调用一次注册，便于管理。

发送自定义消息

API 参考：sendMessage

```
// 构建要发送的自定义消息
const message = new PersonMessage({ name: 'someone', age: 18 })

// 发送消息
RongIMLib.sendMessage({
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: '<targetId>'
}, message).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code)
}
})
```

设置消息状态（Electron）

更新时间:2024-08-30

消息状态功能仅限于配合 Electron 模块 （@rongcloud/electron 与 @rongcloud/electron-renderer）使用。

本文档仅适用于 Electron 解决方案，描述了如何客户端本地数据库中维护消息的接收、发送与已读状态。

设置消息的接收状态

支持多状态同时设置

提示

从 SDK 5.9.3 开始支持该接口。

调用 electronExtension.setMessageReceivedStatusInfo 设置对方发送的消息，自己的接收状态(支持同时设置多种状态)。

```
const messageId = 1 // 取消息中的 messageId 字段
const receivedStatusInfo = {
isRead: true,
isListened: true,
isDownload: false,
isRetrieved: false,
}
RongIMLib.electronExtension.setMessageReceivedStatusInfo(messageId, receivedStatusInfo).then(res => {
console.log(res.code)
})
```

参数	类型	说明
messageId	number	消息 Id
receivedStatusInfo	IReceivedStatusInfo	消息的接收状态详情

支持单一状态设置

提示

SDK 从 5.4.0 版本开始支持该接口，于 5.9.3 版本废弃。如果 IMLib SDK 版本 ≧ 5.9.3，建议使用 setMessageReceivedStatusInfo 接口替换。

调用 electronExtension.setMessageReceivedStatus 设置对方发送的消息，自己的接收状态。

```
const messageId = 1 // 取消息中的 messageId 字段
RongIMLib.electronExtension.setMessageReceivedStatus(messageId, RongIMLib.ReceivedStatus.READ).then(res => {
console.log(res.code)
})
```

参数	类型	说明
messageId	number	消息 Id
receivedStatus	ReceivedStatus	消息的接收状态

设置消息的发送状态

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.setMessageSentStatus 设置自己发送的消息的发送状态。

```
const messageId = 1 // 取消息中的 messageId 字段
RongIMLib.electronExtension.setMessageSentStatus(messageId, RongIMLib.SentStatus.SENT).then(res => {
console.log(res.code)
})
```

参数	类型	说明
messageId	number	消息 Id
sentStatus	SentStatus	消息的发送状态

通过时间戳设置消息状态为对方已读

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.setMessageStatusToRead 将自己发送的指定时间之前的消息设置为对方已读，状态值为 SentStatus.READ。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>",
channelId: ""
}
const timestamp = 1632728573423
RongIMLib.electronExtension.setMessageStatusToRead(conversation, timestamp).then(res => {
console.log(res.code)
})
```

参数	类型	说明
conversation	IConversationOption	会话
timestamp	number	消息的发送时间，将该时间之前的消息设置为已读

管理离线消息存储配置

更新时间:2024-08-30

即时通讯业务支持修改 App 级别的离线消息配置。

> 提示
>
> 离线消息配置仅适用于单聊、群聊。聊天室、超级群因业务特性不支持离线消息，因此无离线消息配置。

了解离线消息

离线消息?是指当用户不在线时收到的消息。融云服务端会自动为用户保留离线期间接收的消息，默认的离线消息保留时长为 7 天。7 天内客户端如果上线，服务端会直接将离线消息发送到该接收端。如果 7 天内客户端都没有上线，服务端将抛弃掉过期的消息。

即时通讯业务下并非所有会话类型都支持离线消息：

- **支持离线消息**：单聊、群聊、系统消息
- **不支持离线消息**：聊天室、超级群

App 级别离线消息配置

如需修改 App 级别设置，请提交工单。

App 级别的离线消息配置如下：

- 单聊离线消息存储时长：默认存储 7 天。设置范围为 1 - 7 天。配置修改将影响 App 下所有单聊会话。
- 群聊离线消息存储时长：默认存储 7 天。设置范围为 1 - 7 天。配置修改将影响 App 下所有群聊会话。
- 群组离线消息存储数量：默认存储 7 天内的所有群消息。配置修改将影响 App 下所有群聊会话。

会话介绍

更新时间:2024-08-30

会话是指融云 SDK 根据每条消息的发送方、接收方以及会话类型等信息，自动建立并维护的逻辑关系，是一种抽象概念。

会话类型

融云支持多种会话类型，以满足不同业务场景需求。客户端 SDK 通过 ConversationType 枚举来表示各类型会话，各枚举值代表的含义参考下表：

会话类型	描述
ConversationType.PRIVATE	单聊会话
ConversationType.GROUP	群组会话
ConversationType.CHATROOM	聊天室会话
ConversationType.SYSTEM	系统会话
ConversationType.ULTRA_GROUP	超级群会话

单聊会话

指两个用户一对一进行聊天，两个用户间可以是好友也可以是陌生人，融云不对用户的关系进行维护管理，会话关系由融云负责建立并保持.

群组会话

群组指两个以上用户一起进行聊天，群组成员信息由 App 提供并进行维系，融云只负责将消息传达给群组中的所有用户。每个群最大人数上限为 3000 人，App 内的群组数量没有限制。

超级群会话

超级群会话指无成员上限的多人聊天服务，海量消息并发即时到达，支持消息推送服务。群组成员信息由 App 提供并维系，融云负责将消息传达给群成员。App 内超级群数量没有限制，超级群无人数上限，一个用户可加入 100 个超级群。更多内容请参见超级群概述。

聊天室会话

聊天室成员不设用户上限，海量消息并发即时到达，用户退出聊天室后不会再接收到任何聊天室中的消息，没有推送通知功能。会话关系由融云负责建立并保持连接，通过 SDK 相关接口，可以让用户加入或者退出聊天室。

SDK 不保存聊天室消息，在退出聊天室时会清空此聊天室所有数据。

系统会话

系统会话是指利用系统帐号向用户发送消息从而建立的会话关系，此类型会话可以是通过调用广播接口发送广播来建立，也可以是加好友等单条通知消息而建立的会话。

获取会话

更新时间:2024-08-30

> 提示
>
> - Web 端不具备持久化的数据存储能力，无法在本地持久化存储历史消息记录与会话列表，因此需要从融云服务端获取数据。从远端获取单群聊历史消息要求您已在控制台 IM 服务管理 页面为当前使用的 App Key 开启单群聊消息云端存储服务。**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。
> - 获取远端会话必须在调用 `RongIMLib.connect()` 并且成功建立连接之后执行。
> - 服务器会话列表存储上限为 `1000` 条会话，会话存储有效期与单群聊消息云端存储的有效期一致，默认为 6 个月。

获取会话列表

> 提示
>
> 该接口在 Electron 平台调用时会忽略参数传值，固定返回所有会话列表。建议 Electron 平台客户考虑使用其他获取会话列表的接口。

调用 getConversationList 获取会话，返回 IAReceivedConversation 列表。

```
const startTime = 0;
const count = 10;
const order = 0;

RongIMLib.getConversationList({
count: count,
startTime: startTime,
order: order
}).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

服务端按照会话最后一条消息的时间进行排序。获取方式如下：

- 如需获取最新的会话列表，应设置 `order` 为 `0`、`startTime` 为 `0`，表示以当前时间点为分界，查询早于当前时间的会话，接口将返回最新的会话列表。
- 如需获取服务端最早的会话，应设置 `order` 为 `1`，`startTime` 为 `0`，此时 `startTime = 0` 表示从服务端最早一条会话开始查询，获取指定数量会话，接口将返回服务端 1000 条会话中最早的会话列表。
- 如果分页获取会话，第二次应传入返回结果数组中最后一条数据的 IAReceivedConversation 的 `latestMessage` 的

`sendTime`。

参数	类型	必填	默认值	说明
count	Number	否	300	会话列表数量，默认值 `300`，最大值 `1000`。
startTime	Number	否	0	获取会话的时间边界，为精确到毫秒的时间戳，与 `order` 配合使用。分页获取时，第二次可传入返回结果数组中最后一条数据的 IAReceivedConversation 的 `latestMessage` 的 `sendTime`。
order	Number	否	0	获取会话的方向。`0` 表示获取早于时间边界（`startTime`）的会话。`1` 表示获取晚于时间边界（`startTime`）的会话。

返回结果中的 IAReceivedConversation 属性说明：

字段名	类型	说明
conversationType	Number	会话类型
targetId	String	接收方的 userId
unreadMessageCount	Number	当前会话的未读消息数
latestMessage	Object	会话中最后一条消息
hasMentioned	Boolean	是否包含 @ 自己的消息,此数据仅在 `conversationType` 为 `ConversationType.GROUP` 时有效
mentionedInfo	MentionedInfo	@ 信息。
notificationStatus	Number	当前会话免打扰状态，web 平台在 5.3.0 版本废弃该字段，ELectron 平台在 5.8.4 版本废弃该字段，请使用 notificationLevel。
notificationLevel	NotificationLevel	当前会话免打扰级别， web 平台在 5.3.0 版本支持， ELectron 平台在 5.8.4 版本支持
isTop	Boolean	当前会话免置顶状态
lastUnreadTime	Number	会话中消息的最后未读时间
draft	String	草稿信息，5.9.0 版本新增

获取指定会话

调用 getConversation 获取指定会话 IAReceivedConversation。

> 提示
>
> 通过该方法获取的会话可能并不存在于当前的会话列表中，此处只作为功能性封装语法糖。

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = '接收方的 userId';

RongIMLib.getConversation({
conversationType,
targetId,
}).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。

获取置顶会话列表

调用 getTopConversationList 获取置顶会话列表。

> 提示
>
> 从 SDK 5.6.0 开始支持该接口。注意：该接口在 Web 平台返回数据中将不包含 latestMessage 字段。

```
const conversationTypes = [RongIMLib.ConversationType.PRIVATE, RongIMLib.ConversationType.GROUP,
RongIMLib.ConversationType.SYSTEM];

RongIMLib.getTopConversationList(conversationTypes).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
conversationTypes	ConversationType[]	否	会话类型，参考 ConversationType ， 默认是全部会话类型

获取未读会话列表

> 提示
>
> 从 SDK 5.7.0 开始支持该接口。注意：该接口在 Web 平台返回数据中将不包含 latestMessage 字段。

调用 getUnreadConversationList 获取含有未读消息的会话列表。

```
const conversationTypes = [RongIMLib.ConversationType.PRIVATE, RongIMLib.ConversationType.GROUP,
RongIMLib.ConversationType.SYSTEM];

RongIMLib.getUnreadConversationList(conversationTypes).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
conversationTypes	ConversationType[]	否	会话类型。参考 ConversationType ， 仅支持 单聊、群聊、系统会话类型

获取会话（Electron）

更新时间:2024-08-30

> 本文档仅适用于 Electron 解决方案，仅限于配合 Electron 模块 （@rongcloud/electron 与 @rongcloud/electron-renderer）使用。

本文档描述了如何从客户端本地获取会话。

获取本地全部会话

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。注意：获取本地会话必须在调用 RongIMLib.connect() 并且成功建立连接之后执行。

调用 electronExtension.getAllConversationList 获取本地全部会话。

```
RongIMLib.electronExtension.getAllConversationList().then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
channelId	String	否	频道Id， 默认获取所有频道的会话列表

分页获取本地会话

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。注意：获取本地会话必须在调用 RongIMLib.connect() 并且成功建立连接之后执行。

调用 electronExtension.getConversationList 分页获取本地会话列表。

```
const startTime = 0
const count = 10

RongIMLib.electronExtension.getConversationList(startTime, count).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
startTime	Number	是	时间戳，需要精确到毫秒，0表示当前时间
count	Number	是	数量
channelId	string	否	频道Id，默认获取所有频道的会话列表
topPriority	Boolean	否	获取会话列表是否按置顶状态排序，默认 false

批量获取会话信息

> 提示
>
> 从 SDK 5.9.9 开始支持该接口。注意：获取本地会话必须在调用 RongIMLib.connect() 并且成功建立连接之后执行。

```
// 最多支持获取 100
const conversations = [{ targetId:"user1", conversationType:1 }];

RongIMLib.electronExtension.getConversations(conversations).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
conversations	IConversationOption[]	是	会话 IConversationOption 的数组，一次最多获取 100 个会话。IConversationOption 必须提供会话类型（ConversationType）和会话 ID（targetId）

搜索会话（Electron）

更新时间:2024-08-30

> 本文档仅适用于 Electron 解决方案，仅限于配合 Electron 模块 （@rongcloud/electron 与 @rongcloud/electron-renderer ）使用。

本文档仅描述了如何从客户端本地搜索会话。

搜索本地会话

> 提示
>
> 从 SDK 5.4.0 开始支持该接口。

调用 electronExtension.searchConversationByContent 根据消息内容搜索本地会话列表。

```
const keyword = '<keyword>'
const messageTypes = [RongIMLib.MessageType.TEXT]
RongIMLib.electronExtension.searchConversationByContent(keyword, messageTypes).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
keyword	string	是	搜索关键字
messageTypes	string[]	是	消息类型
channelId	string	否	频道Id，不传则获取全部频道 ID 类型

会话未读数

更新时间:2024-08-30

> ⚠ 警告
>
> 1. 清除浏览器缓存会导致会话未读数不准确
> 2. 会话消息未读数存储在 localStorage 中, 若浏览器不支持或禁用 localStorage，未读消息数将不会保存，浏览器页面刷新未读消息数将不会存在

获取所有会话未读数

获取所有会话类型的未读数。

> ⓘ 提示
>
> 不支持聊天室、超级群会话

API 参考：getTotalUnreadCount

```
const includeMuted = false
const conversationTypes = [RongIMLib.ConversationType.PRIVATE, RongIMLib.ConversationType.GROUP]
RongIMLib.getTotalUnreadCount(includeMuted, conversationTypes).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	必填	说明
includeMuted	Boolean	否	是否包含免打扰会话，默认为 false
conversationTypes	Array	否	会话类型，参考 ConversationType 。

获取所有群会话未读 @ 消息数

获取所有会话类型的未读 @ 消息数。

> ⓘ 提示
>
> 仅支持普通群会话
> SDK 从 5.9.0 版本开始支持该接口

API 参考：getAllUnreadMentionedCount

```
RongIMLib.getAllUnreadMentionedCount().then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

按会话免打扰级别获取未读数（仅限 Web 端）

> 提示
>
> SDK 从 5.5.1 版本开始支持该接口，暂不支持 Electron 平台。

查找免打扰配置符合传入的免打扰级别的会话，返回所有会话中未读消息的总数。

API 参考：getTotalUnreadCountByLevels

```
const conversationTypes = [
RongIMLib.ConversationType.PRIVATE,
RongIMLib.ConversationType.GROUP,
RongIMLib.ConversationType.ULTRA_GROUP,
]
const levels = [
RongIMLib.NotificationLevel.AT_MESSAGE_NOTIFICATION,
RongIMLib.NotificationLevel.AT_USER_NOTIFICATION
]
RongIMLib.getTotalUnreadCountByLevels(conversationTypes, levels).then(res => {
if (res.code === 0) {
console.log(res.data)
} else {
console.log(res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	必填	说明
conversationTypes	ConversationType[]	是	会话类型，参考 ConversationType 。 不支持聊天室。
levels	NotificationLevel[]	是	会话的免打扰级别设置。SDK 会根据该参数查找匹配的会话，并计算未读消息数。参考免打扰级别 。传空数组则统计全部免打扰级别会话中的未读数。

按会话免打扰级别获取未读 @ 消息数（仅限 Web 端）

> 提示
>
> SDK 从 5.5.1 版本开始支持该接口，暂不支持 Electron 平台。

查找免打扰配置符合传入的免打扰级别的会话，返回所有会话中所有未读 @ 消息的总数。

API 参考：getTotalUnreadMentionedCountByLevels

```
const conversationTypes = [
RongIMLib.ConversationType.PRIVATE,
RongIMLib.ConversationType.GROUP,
RongIMLib.ConversationType.ULTRA_GROUP,
]
const levels = [
RongIMLib.NotificationLevel.AT_MESSAGE_NOTIFICATION,
RongIMLib.NotificationLevel.AT_USER_NOTIFICATION
]
RongIMLib.getTotalUnreadMentionedCountByLevels(conversationTypes, levels).then(res => {
if (res.code === 0) {
console.log(res.data)
} else {
console.log(res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	必填	说明
conversationTypes	ConversationType[]	是	会话类型，参考 ConversationType 。不支持聊天室。
levels	NotificationLevel[]	是	会话的免打扰级别设置。SDK 会根据该参数查找匹配的会话，并计算未读 @ 消息数。参考免打扰级别 。传空数组则统计全部免打扰级别会话中的全部未读 @ 消息数。

获取指定会话未读数

API 参考：getUnreadCount

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = " 会话 Id ";

RongIMLib.getUnreadCount({ conversationType, targetId }).then(res => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。

清除指定会话未读数

清除指定会话(除聊天室外)的未读数。

API 参考：clearMessagesUnreadStatus

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = " 会话 Id ";

RongIMLib.clearMessagesUnreadStatus({ conversationType, targetId }).then(res => {
if (res.code === 0) {
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。

清除全部会话未读数

> 提示
>
> 最低版本：5.0.2

```
RongIMLib.clearAllMessagesUnreadStatus()
```

多端同步未读数

未读消息存在 localStorage 中，未读消息数是针对当前端的未读消息数，服务器不存未读消息数量。

API 参考：sendSyncReadStatusMessage

```
// 清除未读数
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = " 会话 Id ";

RongIMLib.clearMessagesUnreadStatus({ conversationType, targetId }).then(res => {
if (res.code === 0) {
// 清理成功
console.log(res)
// 发送多端同步未读数消息
RongIMLib.sendSyncReadStatusMessage({ conversationType, targetId }, Date.now()).then(() => {})
}
})
```

删除会话 删除指定会话

更新时间:2024-08-30

调用 removeConversation 删除指定会话。

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = '接收方的 userId';

RongIMLib.removeConversation({
conversationType,
targetId: targetId,
}).then(res => {
// 删除指定会话成功
if(res.code === 0){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。

会话草稿 保存草稿

更新时间:2024-08-30

调用 saveTextMessageDraft 保存消息草稿。

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = '接收方的 userId';
const draft = '草稿内容'

RongIMLib.saveTextMessageDraft({
conversationType,
targetId,
}, draft).then(res => {
// 保存成功
if(res.code === 0){
console.log(res)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。
draft	String	是	草稿内容

获取草稿

调用 getTextMessageDraft 获取消息草稿。

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = '接收方的 userId';

RongIMLib.getTextMessageDraft({
conversationType,
targetId,
}).then(res => {
// 获取草稿成功
if( res.code === 0 ) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。

删除草稿

调用 clearTextMessageDraft 删除消息草稿。

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = '接收方的 userId';

RongIMLib.clearTextMessageDraft({
conversationType,
targetId,
}).then(res => {
// 获取草稿成功
if( res.code === 0 ) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。

输入状态　　发送输入状态消息

更新时间:2024-08-30

用户正在输入的时候，向对方发送正在输入的状态。

API 参考：sendTypingStatusMessage 向对方发送正在输入的状态。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "targetId"
}
RongIMLib.sendTypingStatusMessage(conversation, RongIMLib.MessageType.TEXT).then(res => {
// 发送输入状态成功
if ( res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	说明
conversation	Object	会话对象。详见下方 conversation 参数说明
typingContentType	String	正在输入的消息的类型名。如文本消息，应该传类型名"RC:TxtMsg"。融云内置消息的类型名称详见消息类型概述 。

- conversation 参数说明

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。

设置输入状态事件监听

API 参考：addEventListener

```
const Events = RongIMLib.Events;
function onTyping({ status }) {
// 同一时刻，可能存在多个会话均有人在输入状态中
status.forEach((item) => {
const { targetId, conversationType, channelId, list } = item;
console.log(`当前正在输入的会话：`, targetId, conversationType, channelId);

// 同一时刻可能存在多人在同一个会话中同时输入，如群组；
// list 为该会话内处于输入状态的人员列表
list.forEach(({ userId, timestamp, messageType }) => {
// ...
})
})
}
RongIMLib.addEventListener(Events.TYPING_STATUS, onTyping);
```

会话置顶　　设置会话置顶

更新时间:2024-08-30

调用 setConversationToTop 设置会话置顶。

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = '接收方的 userId';
const isTop = true

RongIMLib.setConversationToTop({
conversationType,
targetId,
}, isTop).then(({code}) => {
// 设置会话置顶成功
if( !code ){
}
})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型，参考 ConversationType 。
isTop	Boolean	否	是否置顶

免打扰功能概述

更新时间:2024-08-30

即时通讯客户端 SDK 支持多维度、多级别精细化的免打扰设置。

IM 的免打扰设置维度

即时通讯业务支持对免打扰功能进行多维度地控制。开发者可从 App Key、指定细分业务（仅超级群）、用户级别进行免打扰功能配置。

融云服务端会遵照以下优先级搜索免打扰配置，判断当前消息是否需要触发推送：用户级别设置 > 指定超级群频道的默认配置 > 指定超级群会话的默认配置 > App Key 级设置。

> 提示
>
> 客户端 SDK 从 5.3.0 开始提供对以下维度免打扰设置的完整支持。

免打扰配置的维度	适用场景	说明	客户端 API	服务端 API
App Key 级设置	单聊、群聊、超级群	以 App Key 为单位，设置整个应用的默认免打扰级别。默认未设置，等同于全部消息都接收通知。该级别的配置暂未在控制台开放，如有需要，请提交工单。	客户端 SDK 不提供 API。	服务端不提供该 API。
指定超级群默认设置	仅限超级群业务	可为指定的超级群设置默认的免打扰级别。	详见「超级群管理」下的设置群/频道默认免打扰。	详见「超级群管理」下的设置群/频道默认免打扰
指定超级群频道的默认设置	仅限超级群业务	可为指定的超级群设置默认的免打扰级别。	详见「超级群管理」下的设置群/频道默认免打扰。	详见「超级群管理」下的设置群/频道默认免打扰
用户级设置	单聊、群聊、超级群	用户级别免打扰设置如下（优先级从高到低）： 1. 设置指定时段内，应用全局的免打扰级别。 2. （仅超级群）设置指定频道的免打扰级别。 3. 设置指定会话的免打扰级别。 在融云服务端判断是否需要推送通知时，如存在用户级别免打扰配置，则遵照以上优先级判断是否需要推送。	1. Web 端不支持该配置。 2. 详见按频道设置免打扰。 3. 详见设置免打扰。	1. 详见 设置用户免打扰时段。 2. 详见设置会话免打扰。 3. 详见设置会话免打扰。

设置免打扰

更新时间:2024-08-30

本文描述如何为指定会话（targetId）配置免打扰。将会话或部分消息设置为免打扰后，如果移动平台客户端为离线状态，将不会收到远程通知提醒。如果使用 IMKit 客户端，App 在后台运行时将不会为免打扰的会话或消息触发通知提醒，可以收到消息内容。

会话的免打扰配置将会被同步到服务端。融云会为用户自动在设备间同步会话免打扰配置数据。客户端可以通过监听器获取同步通知，也可以主动获取最新数据。

使用免打扰级别（SDK ≧ 5.3.0）

> 提示
>
> - SDK 从 5.3.0 开始提供免打扰级别配置（notificationLevel），App 可以实现精细化的通知控制策略。Android/iOS 移动端从 SDK 5.2.2 开始支持免打扰级别。
> - 如果您的应用基于 Electron 解决方案，暂不支持使用免打扰级别（notificationLevel），请使用免打扰状态配置（notificationStatus）。详见使用免打扰状态。

由于即时通讯本身的业务特性，App 可能需要实现精细化的通知控制策略。当前 Web 端 IMLib SDK 已支持多维度、多级别的免打扰设置。

- 支持 App Key、指定细分业务（仅超级群）、用户级别的免打扰功能配置。在融云服务端决定是否触发推送通知时，不同维度的优先级如下：**用户级别设置 > 指定超级群频道的默认配置（仅超级群支持） > 指定超级群会话的默认配置（仅超级群支持） > App Key 级设置**。
- **用户级别设置** 是指可由 App 用户控制的免打扰配置，包含多个细分维度。在融云服务端决定是否触发推送通知时，如存在用户级别配置，不同细分维度的优先级如下：**全局免打扰 > 按频道设置的免打扰 > 按会话设置的免打扰**。详见免打扰功能概述。
- 使用超级群业务的客户请参见**超级群管理**下的「设置群/频道默认免打扰」与「设置指定群/频道免打扰」文档。

免打扰级别（notificationLevel）提供了针对不同 @ 消息的免打扰控制。从 SDK 5.3.0 开始，免打扰配置支持以下级别：

notificationLevel 的枚举值	数值	说明
NotificationLevel.ALL_MESSAGE	-1	全部消息均接收通知，即关闭免打扰功能
NotificationLevel.NOT_SET	0	未设置（用户未设置时为此状态，为全部消息都通知，在此状态下，如设置了超级群默认状态以超级群的默认设置为准）
NotificationLevel.AT_MESSAGE_NOTIFICATION	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人
NotificationLevel.AT_USER_NOTIFICATION	2	仅针对 @ 指定用户消息进行通知，且仅通知被 @ 的指定的用户进行通知 如：@张三 则张三可以收到推送，@所有人 时不会收到推送
NotificationLevel.AT_GROUP_ALL_USER_NOTIFICATION	4	仅针对 @群全员进行通知，只接收 @所有人 的推送信息
NotificationLevel.NOT_MESSAGE_NOTIFICATION	5	不接收通知，即使为 @ 消息也不推送通知

设置免打扰级别

> 提示
>
> Web 平台从 5.3.0 开始支持 setConversationNotificationLevel 接口，同时废弃 setConversationNotificationStatus。
>
> Electron 平台从 5.8.4 版本开始支持 setConversationNotificationLevel 接口。

调用 setConversationNotificationLevel 设置免打扰级别

```
const conversationType = RongIMLib.ConversationType.GROUP;
const targetId = '目标会话 ID';
const notificationLevel = RongIMLib.NotificationLevel.NOT_MESSAGE_NOTIFICATION

RongIMLib.setConversationNotificationLevel({
conversationType,
targetId,
}, notificationLevel).then(( {code} ) => {
// 设置免打扰状态成功
})
```

参数	类型	必填	说明
targetId	String	是	目标会话 ID
conversationType	Number	是	会话类型。参考 ConversationType 。
notificationLevel	Number	是	免打扰级别。详见上文支持的免打扰级别 对 notificationLevel 的说明。

获取免打扰级别

> 提示
>
> 从 5.3.0 开始支持 getConversationNotificationLevel，同时废弃 setConversationNotificationStatus。

调用 getConversationNotificationLevel 获取免打扰级别

参数	类型	必填	说明
targetId	String	是	目标会话 ID
conversationType	Number	是	会话类型。参考 ConversationType 。

```
const conversationType = RongIMLib.ConversationType.GROUP;
const targetId = '目标会话 ID';

RongIMLib.getConversationNotificationLevel({
conversationType,
targetId,
}).then(({ code, data }) => {

})
```

使用免打扰状态（适用于 Electron）

> 提示
>
> - 如果您实现 Web 应用，且 SDK 版本 < 5.3.0，可使用 `notificationStatus` 相关接口。如果 SDK 版本 ≧ 5.3.0，推荐使用免打扰级别功能。
> - 如果您的应用基于 Electron 解决方案，当前仅支持使用以下设置免打扰状态（`notificationStatus`）的接口，暂不支持配置免打扰级别。

SDK 支持为会话设置免打扰状态（notificationStatus），支持设置两种状态。

值	说明
NotificationStatus.OPEN	免打扰已开启（不接收推送通知）。
NotificationStatus.CLOSE	免打扰已关闭（接收推送通知）。

App 可以调用 setConversationNotificationStatus 设置免打扰状态。调用 getConversationNotificationStatus 查询免打扰状态。

```
const conversationType = RongIMLib.ConversationType.PRIVATE;
const targetId = '接收方的 userId';
const notificationStatus = NotificationStatus.OPEN

// 设置会话为免打扰状态
RongIMLib.setConversationNotificationStatus({
conversationType,
targetId,
}, notificationStatus).then(( {code} ) => {
// 设置免打扰状态成功
if( !code ){

}
})

// 获取会话的免打扰状态
RongIMLib.getConversationNotificationStatus({
conversationType,
targetId,
}).then(({ code, data }) => {

})
```

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	会话类型。参考 ConversationType。
notificationStatus	Number	否	免打扰状态。

获取免打扰会话列表

> 提示
>
> 从 5.1.1 开始支持此接口。

调用 getBlockedConversationList 获取已设置免打扰状态的所有会话。

```
RongIMLib.getBlockedConversationList().then(( {code, data} ) => {
// 获取免打扰会话成功
if( code === RongIMLib.ErrorCode.SUCCESS ){
console.log("blocked conversation list",data)
}
})
```

多端同步阅读状态 多端同步阅读状态

更新时间:2024-08-30

调用 sendSyncReadStatusMessage 同步多个客户端的消息阅读状态。

```
const conversation = {
conversationType: RongIMLib.ConversationType.PRIVATE,
targetId: "<目标用户ID>",
channelId: ""
};
RongIMLib.sendSyncReadStatusMessage(conversation, '1641350006895').then(res => {
if (res.code === 0) {
console.log('多端同步成功！')
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
});
```

参数	类型	说明
conversation	IConversationOption	会话
lastMessageSendTime	number	最后一条消息的时间

多端同步免打扰/置顶

更新时间:2024-08-30

SDK 提供了会话状态（置顶或免打扰）同步机制，通过设置会话状态同步监听器，当在其它端修改会话状态时，可在本端实时监听到会话状态的改变。

设置会话状态多端同步监听器

调用 addEventListener 设置会话状态（置顶和免打扰）多端同步监听器。

```
const Events = RongIMLib.Events
RongIMLib.addEventListener(Events.CONVERSATION, ({ conversationList }) => {
console.log(conversationList)
})
```

管理标签信息数据

更新时间:2024-08-30

SDK 支持创建标签系统，用于对会话进行管理。SDK 创建的标签会被同步到服务端。

本文描述了 SDK 标签管理的接口，包括如何创建标签、编辑标签，获取标签列表，以及移除标签。

> 提示
>
> 关于如何使用标签管理会话，详见设置与使用会话标签。

创建标签

调用 addTag 创建一个标签。每个用户最多可以创建 20 个标签。

```
RongIMLib.addTag({
tagId: '<标签Id>',
tagName: '<标签名字>'
}).then(({ code }) => {
console.log(code);
});
```

参数	类型	必填	说明
tag	ITagParam	是	要创建的标签
tag.tagId	string	是	标签 ID，长度不能超过 10
tag.tagName	string	是	标签名称，长度不能超过 15

移除标签

调用 removeTag 删除标签。

```
RongIMLib.removeTag('<标签Id>').then(({code}) => {
console.log(code)
});
```

参数	类型	必填	说明
tagId	string	是	要移除的标签 ID

编辑标签

调用 updateTag 更新标签。

```
RongIMLib.updateTag({
tagId: '<标签Id>',
tagName: '<新的标签名字>'
}).then(({code}) => {
console.log(code)
});
```

参数	类型	必填	说明
tag	ITagParam	是	要编辑的标签
tag.tagId	string	是	标签 ID，长度不能超过 10
tag.tagName	string	是	标签名称，长度不能超过 15

获取标签列表

调用 getTags 获取标签列表。

```
RongIMLib.getTags().then(({code, data}) => {
console.log(code, data)
});
```

设置与使用会话标签

更新时间:2024-08-30

> 在使用标签管理会话之前，请确保您已创建了标签。详见 创建与管理会话标签。

SDK 支持使用一个或多个标签标记会话，可用于会话的分组显示、获取会话列表等操作，并提供了多个接口用于移除会话上的标签。

会话标签还支持一系列高级功能，包括按照标签获取会话列表、按照标签获取标签下所有会话的消息未读数，以及在特定标签下设置某个会话置顶。

获取指定会话的标签

调用 getTagsFromConversation 获取指定会话上的标签。

```
const conversation = {
conversationType: 1,
targetId: '<会话Id>',
channelId: ''
}
RongIMLib.getTagsFromConversation(conversation).then(({code, tags}) => {
console.log(code, tags);
});
```

参数	类型	必填	说明
conversation	IConversationOption	是	会话

添加会话到指定标签

调用 addConversationsToTag 可使用单个标签标记一个或多个会话。

> 提示
>
> - 每个标签下最多可以添加 1000 个会话。
> - 如果标签下已添加 1000 个会话，继续在该标签下添加会话仍会成功，但会导致最早添加标签的会话被移除标签。
> - 同一个会话可以设置多个不同的标签。

```
const conversations = [{
conversationType: 1,
targetId: '<会话Id_1>',
channelId: ''
}, {
conversationType: 1,
targetId: '<会话Id_2>',
channelId: ''
}];

RongIMLib.addConversationsToTag('<标签Id>', conversations).then(({ code }) => {
console.log(code);
})
```

参数	类型	必填	说明
tagId	string	是	标签Id
conversations	IConversationOption[]	是	会话 IConversationOption 的数组。

从指定标签中删除多个会话

> 提示
>
> 此操作仅解除会话与标签的关系，并不会删除会话数据。

调用 removeConversationsFromTag 从指定标签中删除多个会话。

```
const conversations = [{
conversationType: 1,
targetId: '<会话Id_1>',
channelId: ''
}, {
conversationType: 1,
targetId: '<会话Id_2>',
channelId: ''
}];

RongIMLib.removeConversationsFromTag('<标签Id>', conversations).then(({ code }) => {
console.log(code);
});
```

参数	类型	必填	说明
tagId	string	是	标签Id
conversations	IConversationOption[]	是	会话 IConversationOption 的数组。

从指定会话中删除多个标签

> 提示
>
> 此操作仅解除会话与标签的关系，并不会删除标签数据。

调用 removeTagsFromConversation 从指定会话中删除多个标签。

```
const conversation = {
conversationType: 1,
targetId: '<会话Id>',
channelId: ''
}
RongIMLib.removeTagsFromConversation(conversation, ['tagId_1', 'tagId_2']).then(({code}) => {
console.log(code);
});
```

参数	类型	必填	说明
conversation	IConversationOption	是	会话
tagIds	string[]	是	标签 ID 集合

从多个会话中删除指定的标签

> 提示
>
> 此操作仅解除会话与标签的关系，并不会删除标签数据。

调用 removeTagFromConversations 从多个会话中删除指定的标签。

```
const conversations = [{
conversationType: 1,
targetId: '<会话Id_1>',
channelId: ''
}, {
conversationType: 1,
targetId: '<会话Id_2>',
channelId: ''
}];

RongIMLib.removeTagFromConversations('<标签Id>', conversations).then(({code}) => {
console.log(code);
})
```

参数	类型	必填	说明
tagId	string	是	标签Id
conversations	`IConversationOption[]`	是	会话 IConversationOption 的数组。

分页获取标签下会话列表

> 提示
>
> 从 SDK 版本 5.7.0 开始，该接口的返回数据类型由 IReceivedConversationByTag 变更为 IAReceivedConversationByTag 。

调用 getConversationsFromTagByPage 分页获取标签下会话列表。

```
RongIMLib.getConversationsFromTagByPage('<标签Id>', 10, 1640869971655).then(({code, data}) => {
console.log(code, data);
})
```

参数	类型	必填	说明
tagId	string	是	标签 ID
count	number	是	获取数量
startTime	number	是	会话中消息的时间戳

根据标签获取未读消息数

调用 getUnreadCountByTag 根据标签获取未读消息数。

```
RongIMLib.getUnreadCountByTag('<标签Id>', true).then(({code, data}) => {
console.log(code, data);
})
```

参数	类型	必填	说明
tagId	string	是	标签Id
containMuted	boolean	是	是否包含免打扰

设置标签中会话置顶

调用 setConversationToTopInTag 设置在指定标签下置顶的会话。如果根据标签获取会话，可在获取的会话中可以看到该属性。请注意与会话列表中的会话置顶相区分。

```
const conversation = {
conversationType: 1,
targetId: '<会话Id>',
channelId: ''
}
RongIMLib.setConversationToTopInTag('<标签Id>', conversation, true).then(({code}) => {
console.log(code);
})
```

参数	类型	必填	说明
tagId	string	是	标签 ID
conversation	IConversationOption	是	会话
isTop	boolean	是	置顶状态

群组业务概述

更新时间:2024-08-30

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

群组业务要点如下：

- 融云只负责将消息传达给群组中的所有用户，不维护群组成员的资料（头像、名称、群成员名片等），需要由开发者应用服务器维护。
- 创建/解散/加入/退出群组等群组管理操作，必须由 App 服务器请求融云服务端 API 实现。融云客户端 SDK 不提供相应方法。详见下方群组管理功能。
- App Key 下可创建的群组数量没有限制，单个群组默认成员上限为 3000 人。可提交工单修改群成员人数上限。
- 单个用户可加入的群组数量无限制。
- 从控制台 IM 服务管理页面为 App Key 开启单群聊消息云端存储服务后，可使用融云提供的消息存储服务，实现消息历史记录漫游。**IM 旗舰版**或**IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明页面及计费说明文档为准。

服务配置

客户端 SDK 默认支持群组业务，不需要申请开通。

群组业务的部分基础功能与增值服务可以在控制台的免费基础功能和 IM 服务管理页面进行开通和配置。

客户端 SDK 使用须知

> 提示
>
> - 仅 IMKit 提供开箱即用的群聊会话 UI 组件。IMLib 不提供开箱即用的群聊会话 UI 组件。
> - 客户端不提供群组管理的 API。群组管理需要由 App 服务端调用相应的融云服务端 API（Server API）接口完成。

群组管理功能

融云不会托管用户，也不管理群组的业务逻辑，因此群的业务逻辑全部需要在 App 服务器进行实现。

> 提示
>
> 群主、群管理员、群公告、邀请入群、群号搜索等均为群组业务逻辑，需在 App 侧自行实现。

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

下表列出了融云服务端提供的群组基础管理接口。注意，客户端不提供群组管理的 API。

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

群聊消息功能

群聊消息功能与单聊业务类似，共用部分 API 及配置。

功能	描述	客户端 API	融云服务端 API
发送消息	可发送普通消息与媒体消息，例如文本、图片、GIF 等，或自定义消息。支持在发送消息时添加 @ 信息。	发送消息	发送群聊消息
发送群聊定向消息	可发送普通消息与媒体消息给群组中的指定的一个或多个成员，其他成员不会收到该消息。	发送群定向消息	发送群聊定向消息
接收消息	监听并实时接收消息，或在客户端上线时接收离线消息。	接收消息	不提供该 API
群聊消息已读回执	发送群消息后如需要查看消息的阅读状态，需要先发送回执请求，在通过接受者的响应获取已读数据。	群聊消息回执	不提供该 API
离线消息	支持离线消息存储，存储时间可设置（1 ～ 7 天），默认存储 7 天内的所有群消息，支持调整存储时长与存储的群消息数量。	不提供该 API	不提供该 API
控制离线消息推送属性	接收方离线状态下，群消息可触发推送通知。通过配置消息推送属性（IPushConfig）可控制移动端接收的推送通知的展示效果及行为。	发送消息	不提供该 API
撤回消息	消息发送成功后可撤回该条消息。	撤回消息	撤回消息
获取历史消息	从本地数据库或远端获取历史消息。注意，从远端获取历史消息需要开通单群聊消息云存储服务，默认存储时长为 6 个月。	获取历史消息	不提供该 API
获取历史消息日志	融云服务端可以保存 APP 内所有会话的历史消息记录，历史消息记录以日志文件方式提供，并已经过压缩。您可以使用服务端 API 获取、删除指定 App 的历史消息日志	不提供该 API	获取历史消息日志

功能	描述	客户端 API	融云服务端 API
删除远端消息	支持从服务端删除指定单个会话中指定消息、指定时间戳之前的消息、或会话中全部历史消息。	删除消息	消息清除
单群聊消息扩展	为原始消息增加状态标识（扩展数据为 KV 键值对），提供添加、删除、查询扩展信息的接口。	消息扩展	单/群聊消息扩展
自定义消息类型	如果内置消息类型满足不了您的需求，可以自定义消息类型。支持自定义普通消息类型与自定义媒体消息类型。	自定义消息	不提供该 API
本地搜索消息（仅限 Electron）	消息存储在本地（移动端），支持按关键字或用户搜索本地指定会话的消息内容。	搜索消息（Electron）	不提供该 API
本地插入消息（仅限 Electron）	在本地数据库中插入消息。本地插入的消息不会实际发送给服务器和对方。	插入消息（Electron）	不提供该 API
删除本地消息（仅限 Electron）	支持从本地删除指定单个会话中指定消息、指定时间戳之前的消息、或会话中全部历史消息。	删除消息（Electron）	不提供该 API
设置消息状态（仅限 Electron）	支持从客户端本地数据库中维护消息的接收、发送与已读状态。	设置消息状态（仅限 Electron）	不提供该 API

默认新入群的成员的用户仅可接收加入群组后产生的消息。如果需要查看入群之前的历史消息，请为 App Key 开启以下两项服务（请注意区分开发/生产环境）：

从控制台 IM 服务管理 页面开启**单群聊消息云端存储**服务。开启该服务后，可使用融云提供的消息存储服务，实现消息历史记录漫游。**IM 旗舰版**或**IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。

- 从控制台免费基础功能 页面开启**新用户获取加入群组前历史消息**。

群聊会话功能

群聊会话功能与单聊业务类似，共用部分 API 及配置。

功能	描述	客户端 API	融云服务端 API
获取远端会话	服务端获取会话列表，最多 1000 条。支持获取指定会话、设置为置顶的会话、未读会话。	获取会话	不提供该 API
处理会话未读消息数	获取或清除会话中的未读消息数，可用于 UI 展示。	会话未读数	不提供该 API
删除会话	从会话列表中删除指定的单个会话。	删除会话	不提供该 API
会话草稿	保存一条草稿内容至指定会话。	会话草稿	不提供该 API
输入状态	可设置指定的群聊会话，收到新的消息后是否进行提醒，默认进行新消息提醒。	输入状态	不提供该 API
管理会话标签	创建和管理标签信息数据，用于对会话进行标记分组。每个用户最多可以创建 20 个标签。App 用户创建的标签信息数据会同步融云服务端。	管理标签信息数据	不提供该 API
设置与使用会话标签	使用会话标签对会话进行分组。	设置与使用会话标签	不提供该 API
会话置顶	在会话列表中将指定会话置顶。	会话置顶	会话置顶
会话免打扰	控制用户在客户端设备离线时，是否可针对离线消息接收推送通知。支持按照会话或按会话类型设置免打扰。	免打扰功能概述	免打扰功能概述

功能	描述	客户端 API	融云服务端 API
多端同步会话免打扰/置顶状态	SDK 提供了会话状态（置顶或免打扰）同步机制，通过设置会话状态同步监听器，当在其它端修改会话状态时，可在本端实时监听到会话状态的改变。	多端同步免打扰/置顶	不提供该 API
多端同步阅读状态	在同一用户账户的多个设备间主动同步会话的阅读状态。	多端同步阅读状态	不提供该 API
获取本地会话（仅限 Electron）	SDK 会根据收发的消息在本地数据库中生成对应会话。您可以从本地数据库获取 SDK 生成的会话列表。	获取会话（Electron）	不提供该 API
搜索本地会话（仅限 Electron）	按关键字、消息类型搜索客户端本地生成的会话列表。	搜索会话（Electron）	不提供该 API

与聊天室和超级群的区别

您可以通过以下文档了解业务类型之间的区别及所有功能：

- 即时通讯开发指导 · 业务类型介绍
- IM 尊享版、IM 旗舰版功能对照表

发送群定向消息

更新时间:2024-08-30

SDK 支持往群聊会话中发送定向消息。定向消息只会发送给指定用户，群聊会话中的其它用户不会收到这条消息。

开通服务

使用发送群组定向消息功能无需开通服务。注意，如需将群组定向消息存入服务端历史消息记录，需要开通以下服务：

- 单群聊历史消息云存储服务，可前往控制台 IM 服务管理 页面为当前使用的 App Key 开启服务。IM 旗舰版或IM 尊享版可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。
- 群定向消息云存储服务，需要提交工单 申请开通。

默认情况下，客户端发送与接收的群定向消息默认都不会存入历史消息服务，因此客户端调用获取历史消息的 API 时，从融云服务端返回的结果中不会包含当前用户发送、接收的群组定向消息。

发送群组定向消息

调用 sendMessage 发送群定向消息。

```
//发送群定向消息，只当 conversationType 值为 `ConversationType.GROUP` 时有效
const conversationType = RongIMLib.ConversationType.GROUP
const targetId = 'GroupID'

const message = new RongIMLib.TextMessage({
content: '我是一条文字信息'
})

RongIMLib.sendMessage({
conversationType,
targetId
}, message, {
directionalUserIdList: ['user1'], // 用户 id 列表
}).then(res => {
if(res.code === 0){
// 发送群@消息成功
console.log(res.data)
}
})
```

发送群 @ 消息

更新时间:2024-08-30

在群组中可发送群 @ 消息。

构造 @ 消息

待发送的消息内容，可以是如 RongIMLib.TextMessage 般的 IMLib 内置消息实例，也可以是通过 RongIMLib.registerMessageType() 实现的自定义消息实例。

```
const details = {
content: '我是一条文字信息',
mentionedInfo: {
type: 1, //1: @ 所有人 2: @ 指定用户
userIdList: ['user1'],//被 @ 的用户 Id 列表
mentionedContent: '有人@你'//携带扩展信息，例如 `有人@你`
}
}
const message = new RongIMLib.TextMessage(details)
```

参数	类型	必填	说明
content	String	是	文本内容
mentionedInfo	Object	否	@ 信息详情。
mentionedInfo.type	Number	否	@ 消息类型 1: @ 所有人 2: @ 指定用户
mentionedInfo.userIdList	String []	否	被 @ 的用户 Id 列表
mentionedInfo.mentionedContent	String	否	推送通知内容，例如 '有人@你'。@消息携带的 mentionedContent 优先级最高，会覆盖所有默认或自定义的 pushContent 数据。

发送 @ 消息

调用 sendMessage 在群组中发送 @ 消息，请务必设置 isMentioned 为 true。

```
const details = {
content: '我是一条文字信息',
mentionedInfo: {
type: 1,
userIdList: ['user1'],
mentionedContent: '有人@你'
}
}
const message = new RongIMLib.TextMessage(details)
//发送 @ 消息，只当 conversationType 值为 `ConversationType.GROUP` 时有效
const conversation = {
conversationType: RongIMLib.ConversationType.GROUP,
targetId: '目标ID'
}
const options = {
isMentioned: true
}
RongIMLib.sendMessage(conversation, message, options).then(res => {
if(res.code === 0){
// 发送群@消息成功
console.log(res.data)
} else {
console.log(res.code, res.msg);
}
})
```

sendMessage 方法具有以下参数：

- conversation： - 消息投送目标会话，API 参考 IConversationOption
- message：待发送的消息内容，可以是如 RongIMLib.TextMessage 般的 IMLib 内置消息实例，也可以是通过 RongIMLib.registerMessageType() 实现的自定义消息实例
- options：定义发送行为中的一些可选项，如是否可拓展，推送等。

参数	类型	必填	说明
isMentioned	boolean	否	是否为 @ 消息，只当 conversationType 值为 ConversationType.GROUP 时有效
disableNotification	boolean	否	当值为 true 时，服务器将不会发送 Push 信息，移动端也不会弹出本地通知提醒

聊天室概述

更新时间:2024-08-30

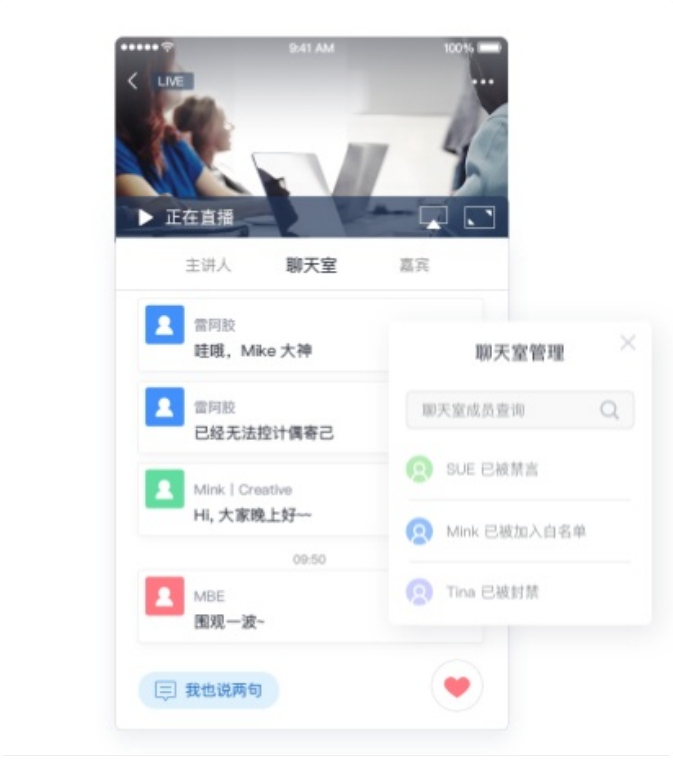
聊天室（Chatroom）提供了一种不设用户上限，支持高并发消息处理的业务形态，可用于直播、社区、游戏、广场交友、兴趣讨论等场景。聊天室业务要点如下：

- App Key 下可创建的聊天室数量没有限制，单个聊天室成员数量没有限制。
- 聊天室具有自动销毁机制，默认情况下所有聊天室会在不活跃（连续时间段内无成员进出且无新消息）达到 1 小时后踢出所有成员并自动销毁，可延长该时间，也可配置为定时自动销毁。详见服务端文档聊天室销毁机制。
- 聊天室具有离线成员自动退出机制。满足默认预设条件时，融云服务端会踢出聊天室成员，详见退出聊天室。
- 聊天室本地消息会在退出聊天室时删除。**IM 旗舰版** 与 **IM 尊享版**客户可选择启用**聊天室消息云端存储**功能，将消息存储在融云服务端。具体功能与费用以融云官方价格说明页面及计费说明文档为准。
- 聊天室不具备离线消息转推送功能，只有在线的聊天室成员可接收聊天室消息。

客户端 UI 框架参考设计

聊天室产品暂不提供聊天室会话专用的 UI 组件。您可以参考以下 UI 框架设计了解聊天室的设计思路。

- 下图**聊天室**标签中为聊天室消息列表。
- 下图**聊天室管理**窗口中展示了聊天室支持的部分能力，如禁言、封禁、白名单等。



服务配置

客户端 SDK 默认支持聊天室，不需要申请开通。

聊天室的部分基础功能与增值服务可以在控制台的免费基础功能和 IM 服务管理页面进行开通和配置。

客户端 SDK 使用须知

> 提示
>
> IMKit 依赖 IMLib，因此 IMKit 具备 IMLib 的全部能力，包括聊天室。但请注意 IMKit 不提供开箱即用的聊天室会话 UI 组件。

聊天室功能接口

聊天室会话关系由融云负责建立并保持连接。SDK 提供加入、退出等部分聊天室管理接口。更多聊天室管理功能需要配合使用即时通讯服务端 API。下表描述了融云聊天室主要的功能接口。

功能分类	功能描述	客户端 API	融云服务端 API
创建与销毁聊天室	手动创建聊天室，或手动销毁聊天室。注意：客户端 SDK 无单独的创建聊天室 API。客户端不提供手动销毁聊天室 API。	不提供该 API	创建房间、销毁房间
加入与退出聊天室	加入已存在的聊天室，请确保聊天室 ID 已存在。加入与退出聊天室仅客户端提供 API。注意：客户端有废弃接口可支持在聊天室不存在时创建聊天室再加入，但已不推荐使用。	加入聊天室、退出聊天室	不提供该 API
查询聊天室房间与用户信息	• 查询聊天室房间的基础信息，包括聊天室 ID、名称、创建时间。 • 查询聊天室成员信息，支持获取聊天室成员用户 ID、加入时间，最多返回 500 个成员信息，支持按加入时间排序。	查询聊天室信息	查询房间信息
聊天室保活	添加一个或多个聊天室到聊天室保活列表。在保活列表中的聊天室不会被融云服务端自动销毁。	不提供该 API	保活房间
聊天室属性管理	在指定聊天室中设置自定义属性。比如在语音直播聊天室场景中，利用此功能记录聊天室中各麦位的属性；或在狼人杀等卡牌类游戏场景中记录用户的角色和牌局状态等。 聊天室属性以 Key-Value 的方式进行存储，支持设置、删除与查询属性，支持批量和强制操作。	聊天室属性	属性管理（KV）
封禁/解封聊天室用户	封禁一个或多个聊天室成员。被封禁成员将被踢出指定聊天室，并在封禁时间内不能再进入此聊天室中。	不提供该 API	成员封禁
聊天室用户白名单	需在 IM 服务管理页面**普通服务**下开通后使用。**IM 旗舰版**或**IM 尊享版**可开通该服务。具体功能与费用以融云官方价格说明页面及计费说明文档为准。 用户被加入某个聊天室的白名单后，在该聊天室消息量较大的情况下，该用户发送的消息不会被丢弃；并且用户也不会被融云服务端自动踢出该聊天室。	不提供该 API	聊天室白名单服务
发送聊天室消息	发送聊天室消息。	发送消息	发送聊天室消息
撤回聊天室消息	撤回聊天室消息。	撤回消息	消息撤回
获取聊天室历史消息	获取聊天室历史消息。	获取聊天室历史消息	历史消息日志

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。IM 旗舰版或IM 尊享版可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。 如果消息类型在低级别消息列表中，该类型的消息全部视为低级别消息。当服务器负载高时，高级别的消息优先保留，低级别消息则优先丢弃。默认情况下，所有消息均为高级别消息。	不提供该 API	聊天室消息优先级服务
聊天室消息白名单	需在 IM 服务管理 页面普通服务下开通后使用。IM 旗舰版或IM 尊享版可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。 如果消息类型在聊天室消息白名单中，该类型的消息全部受到保护，在聊天室消息量较大的情况下也不会被丢弃。	不提供该 API	聊天室白名单服务
聊天室成员禁言	在指定的某个聊天室中，禁言一个或多个成员。聊天室成员被禁言后，可以接收并查看聊天室中用户聊天信息，但不能通过往该聊天室内发送消息。	不提供该 API	单人禁言
全体成员禁言	设置某一聊天室全部成员禁言，或取消指定聊天室全部成员禁言状态。设置全体群成员禁言后，该聊天室的所有成员均不能通过客户端 SDK 往该群组内发送消息。	不提供该 API	全体禁言
全体禁言白名单	添加一个或多个群成员到聊天室全体成员禁言白名单。聊天室成员被添加到白名单后，即使该聊天室处于全体成员禁言状态，该成员仍可通过客户端 SDK 往该聊天室发送消息。	不提供该 API	全体禁言
全局禁言聊天室成员	需在 IM 服务管理 页面普通服务下开通后使用。IM 旗舰版或IM 尊享版可开通该服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。 添加一个或多个用户到聊天室全局禁言列表中，列表中的用户在应用下的所有聊天室中都无法发送消息。	不提供该 API	全局禁言

与群组和超级群的区别

您可以通过以下文档了解业务类型之间的区别及所有功能：

- 即时通讯开发指导 · 业务类型介绍
- IM 尊享版、IM 旗舰版功能对照表

聊天室服务配置

更新时间:2024-08-30

聊天室业务本身不需要单独申请开通，但部分聊天室服务需要在控制台开通与配置，例如聊天室广播消息、聊天室消息云端存储、以及与聊天室相关的回调地址等。

聊天室服务配置主要在免费基础功能和 IM 服务管理页面。

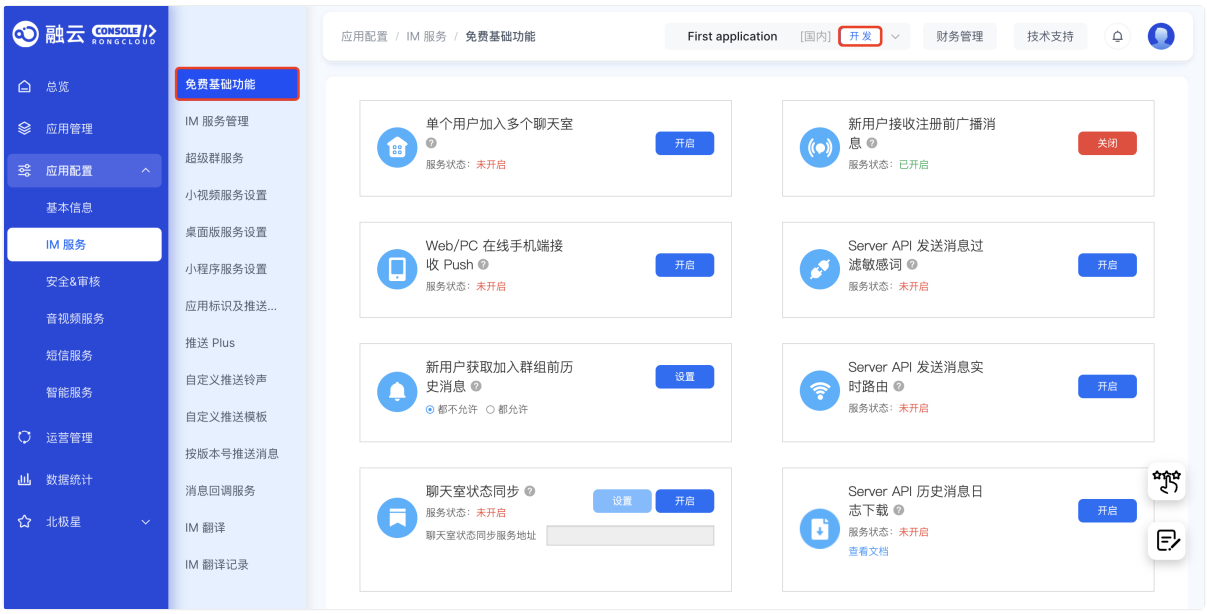
免费基础功能

以下是聊天室业务提供的免费基础功能：

- 单个用户加入多个聊天室：默认一个用户只能加入一个聊天室中，开启后一个用户可以同时加入到多个聊天室中。
- 聊天室状态同步：聊天室状态同步是融云提供的服务端回调服务，需同时提供可正常访问的回调地址。配置成功后，在应用下聊天室发生状态变化时，将实时同步到开发者的应用服务器地址，目前支持的同步状态包括：创建、销毁、成员加入、成员退出聊天室。详见 IM 服务端文档「聊天室管理」下的聊天室状态同步。
- 加入聊天室获取指定消息设置：默认加入聊天室时可最多获取全部消息类型的最近 50 条消息，开启后可设置指定消息类型获取。
- 聊天室销毁等待时间：
  1. 可以支持配置不活跃聊天室销毁的等待时间，默认等待时间为1小时，即超过1小时不活跃即被销毁，客户可以按需调整这个时间，最长可设置24小时。
  2. 聊天室销毁时，会向聊天室成员发送聊天室销毁通知，以方便客户可以在聊天室销毁后，在终端自定义一些操作(依赖 5.1.1 及以后版本)。
  3. 聊天室增加 sessionid，在聊天室生存周期内保持不变，聊天室重建后重新生成。用于使用相同聊天室ID，多次开播时，客户端能区分出来。
- 聊天室属性自定义设置：可在指定聊天室中设置自定义属性，用于语音直播聊天室场景的会场属性同步或狼人杀等卡牌类游戏场景中记录用户的角色和牌局状态等，详见「聊天室业务」下的**聊天室属性管理（KV）**文档。如果 App 业务服务端需要融云提供聊天室属性变更数据同步，需要提供可正常访问的回调地址，配置成功后，自动开启融云提供的服务端回调服务，详见 IM 服务端文档「聊天室管理」下的聊天室属性同步。

修改服务配置

访问控制台免费基础功能页面，可调整聊天室业务相关的免费基础功能配置。



IM 旗舰版/尊享版功能

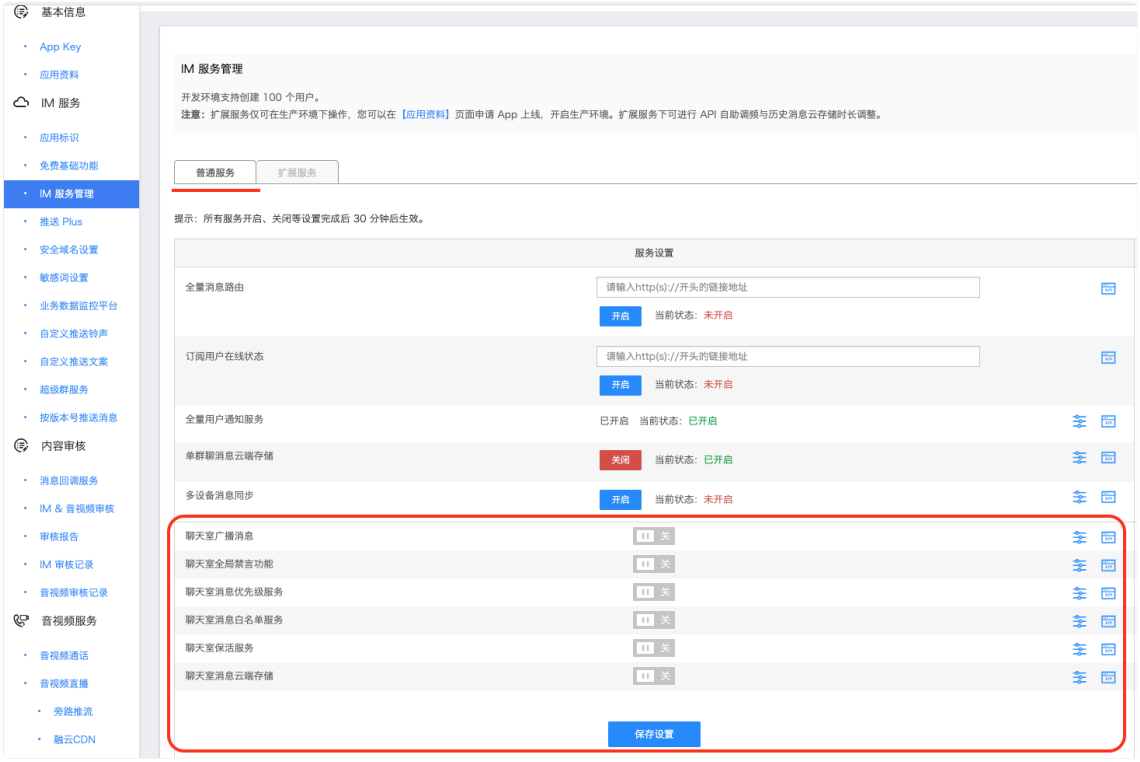
下图显示了控制台 IM 服务管理 页面与聊天室业务相关的普通服务配置。

> 开发环境下可以免费使用。生产环境下，IM 旗舰版或 IM 尊享版才能使用以下服务。

- **聊天室广播消息**：向应用中的所有聊天室发送一条消息，单条消息最大 128k。详见 IM 服务端文档「消息管理」下的发送全体聊天室广播消息 。
- **聊天室全局禁言功能**：当不想让某一用户在所有聊天室中发言时，可将此用户添加到聊天室全局禁言中，被禁言用户可接收查看聊天室中用户聊天信息，但不能发送消息。详见 IM 服务端文档「聊天室用户管理」下的全局禁言用户 。
- **聊天室消息优先级服务**：在指定聊天室中设置指定类型的消息为低级别消息。当服务器负载高时低级别消息优先被丢弃，这样可以确保重要的消息不被丢弃。详见 IM 服务端文档「聊天室消息优先级服务」下的添加低级别消息 。
- **聊天室白名单服务**：开通后，可以使用以下功能对应的 Server API：
  - 聊天室用户白名单 ：可用于保护指定聊天室中的重要用户，支持按聊天室设置白名单用户。例如，App 业务中指定聊天室中的管理员、主播等重要角色的用户。
  - 聊天室消息白名单 ：可用于保护 App 下所有聊天室中的指定消息类型。例如 App 业务中自定义的红包消息。
- **聊天室保活服务**：当聊天室中 1 小时无人说话，同时没有人加入聊天室时，融云服务端会自动把聊天室内所有成员踢出聊天室并销毁聊天室。保活的聊天室不会被自动销毁，可以调用 API 接口销毁聊天室。详见 IM 服务端文档「聊天室管理」下的保活聊天室 。
- **聊天室消息云端存储**：聊天消息保存在云端，用户进入聊天室后，可以查看聊天室中以前的消息，历史消息默认保存 2 个月。
- **加入聊天室获取指定消息配置**：加入聊天室时只返回指定类型的消息，不返回其他类型的消息。

修改服务配置

访问开发后台 IM 服务管理 页面，切换到普通服务标签下，可启用以下聊天室服务配置开关。



您还可以在扩展服务标签下对部分服务的具体配置进行调整。

加入聊天室

更新时间:2024-08-30

应用程序服务端创建聊天室后，可将聊天室 ID 下分发至客户端。客户端获取聊天室 ID 后可加入聊天室。

- 应用程序的服务端可以调用 IM Server API 创建聊天室。详见服务端文档创建聊天室 。
- 应用程序的服务端可以通过聊天室状态同步 收到聊天室创建成功与成员加入等事件通知。
- 默认同一用户不能同时加入多个聊天室。App 用户加入新的聊天室后会自动退出之前的聊天室。您可以在控制台启用单个用户加入多个聊天室，详见聊天室服务配置。
- IMLib SDK 可在断网重连后自动重新加入聊天室。

加入已存在的聊天室

> 提示
>
> 1. Electron 平台从 5.7.0 开始提供该接口。
> 2. 从 5.8.3 开始，成功回调的 data 数据中返回聊天室信息和禁言状态，详见 IChatroomJoinResponse 。

如果聊天室已存在，在获取 chatRoomId 后可以调用 joinExistChatRoom 加入聊天室。该方法只能加入已存在的聊天室。

```
const chatRoomId = "聊天室 ID";
const count = 50;

RongIMLib.joinExistChatRoom(chatRoomId, {
count: count
}).then(res => {
// 加入聊天室成功
const { code, data } = res;
if(code === RongIMLib.ErrorCode.SUCCESS){
console.log(code, data)
} else {
console.log(code)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	string	聊天室 ID。应用程序服务端应调用 IM Server API 创建聊天室，详见创建聊天室 。
count	number	进入聊天室时获取历史消息的数量，数量范围：1-50。如果传 -1，表示不获取任何历史消息。如果传 0，表示使用 SDK 默认设置（默认为获取 10 条）。

加入聊天室

> 提示

IMLib 从 5.8.3 开始废弃该方法

joinChatRoom 接口会创建并加入聊天室。如果聊天室已存在，则直接加入。如果您使用了服务端回调聊天室状态同步，融云会将聊天室创建成功的通知发送到您指定的服务器地址。

```
const chatRoomId = "聊天室 ID";
const count = 50;

RongIMLib.joinChatRoom(chatRoomId, {
count: count
}).then(res => {
// 加入聊天室成功
if (res.code === RongIMLib.ErrorCode.SUCCESS){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	string	聊天室 ID
count	number	进入聊天室时获取历史消息的数量，数量范围：1-50。如果传 `-1`，表示不获取任何历史消息。如果传 `0`，表示使用 SDK 默认设置（默认为获取 10 条）。

断线重连后重新加入聊天室

SDK 具备断线重连机制。重连成功后，如果当前登录用户曾经加入过聊天室，且没有退出，则 SDK 会自动重新加入聊天室，不需要 App 处理。App 可以通过聊天室状态回调收到通知。

提示

断网重连场景下，SDK 重新加入聊天室成功后，会按照加入聊天室时传入的 `count` 值拉取固定数量的消息。拉取的消息有可能在本地已存在，App 可能需要进行排重后再显示。

- Electron 从 5.7.0 开始，支持断网重连情况下，SDK 重新加入聊天室后拉取消息的行为。低于 5.7.0 版本的 SDK 不会拉取消息。

退出聊天室

更新时间:2024-08-30

退出聊天室支持以下几种情况：

- **被动退出聊天室**：聊天室具有离线成员自动踢出机制。该机制被触发时，融云服务端会将用户踢出聊天室。用户如被封禁，也会被踢出聊天室。
- **主动退出聊天室**：客户端提供 API，支持由用户主动退出聊天室。

聊天室离线成员自动退出机制

聊天室具有离线成员自动退出机制。用户离线后，如满足以下默认预设条件，融云服务端会自动将该用户踢出聊天室：

- 从用户离线开始 30 秒内，聊天室中产生第 31 条消息时，触发自动踢出。
- 或用户已离线 30 秒后，聊天室有新消息产生时，触发自动踢出。

> 提示
>
> - 默认预设条件均要求聊天室中必须要有新消息产生，否则无法触发踢出动作。如果聊天室中没有消息产生，则无法将异常用户踢出聊天室。
> - 如需修改默认行为对新消息的依赖，请提交工单申请开通**聊天室成员异常掉线实时踢出**。开通该服务后，服务端会通过 SDK 行为（要求 Android/iOS IMLib SDK 版本 ≧ 5.1.6，Web IMLib 版本 ≧ 5.3.2）判断用户是否处于异常状态，最迟 5 分钟可以将异常用户踢出聊天室。
> - 如需保护特定用户，即不自动踢出指定用户（如某些应用场景下可能希望用户驻留聊天室），可使用 Server API 提供的聊天室用户白名单功能。

主动退出聊天室

客户端用户可主动退出聊天室。调用 quitChatRoom 退出聊天室。

```
const chatRoomId = "<聊天室 ID>";

RongIMLib.quitChatRoom(chatRoomId).then(res => {
// 退出聊天室成功
if(res.code === 0){
console.log('退出聊天室 ' + chatRoomId)
}
})
```

参数	类型	说明
chatRoomId	string	聊天室 ID

监听聊天室事件

更新时间:2024-08-30

聊天室业务支持客户端 App 监听以下类型的聊天室操作事件：聊天室房间的状态变化与本端聊天室相关操作、用户在当前及其他客户端加入退出聊天室的状态、聊天室中成员进出的事件通知、以及聊天室中成员禁言、封禁相关的信息。

设置聊天室操作监听器

通过 addEventListener 可以设置监听器。

调用示例： `addEventListener(Events.CHATROOM, listener)`

```
const listener = (event) => {
if (event.rejoinedRoom) {
console.log('SDK 内部重连聊天室信息:', event.rejoinedRoom)
}
if (event.updatedEntries) {
console.log('监听到的聊天室 KV 更新:', event.updatedEntries)
}
if (event.userChange) {
console.log('加入退出的用户通知:', event.userChange)
}
if (event.chatroomDestroyed) {
console.log('聊天室销毁:', event.chatroomDestroyed)
}
/* since 5.7.9 */
if (event.chatroomNotifyMultiLoginSync) {
console.log('加入退出多端同步通知:', event.chatroomNotifyMultiLoginSync)
}
/* since 5.7.9 */
if (event.chatroomNotifyBlock) {
console.log('聊天室用户封禁通知:', event.chatroomNotifyBlock)
}
/* since 5.7.9 */
if (event.chatroomNotifyBan) {
console.log('聊天室用户禁言通知:', event.chatroomNotifyBan)
}
}
RongIMLib.addEventListener(Events.CHATROOM, listener)
```

聊天室销毁事件

下方列出了聊天室销毁事件。

事件	触发时机	说明
`event.chatroomDestroyed`	当前用户在线，且所在聊天室被销毁	返回数据包括：聊天室 ID。

聊天室成员变化事件

本端用户加入、退出聊天室时默认会触发 `event.userChange` 事件。

> 提示
>
> 如果需要监听聊天室中其他人员的变动，需要开通服务，请提交工单，申请开通聊天室成员变化监听。

事件	触发时机	说明
event.userChange	用户加入、退出聊天室	IChatroomUserChangeInfo 封装了当前聊天室中加入或退出聊天室的成员信息。如果当前用户由于网络原因和聊天室断开连接，则无法监听到断开连接期间的其他成员的加入、退出行为。

多端登录事件通知

> 提示
>
> SDK 从 5.4.5 版本开始支持监听以下事件通知。

在用户在多端登录情况下，在其他客户端加入、退出聊天室时，触发 event.chatroomNotifyMultiLoginSync 事件。返回数据详细定义参见 IChatroomNotifyMultiLoginSync。

触发场景	通知范围	说明
用户多端登录，在一台设备上加入聊天室	当前加入的用户	当前用户在一端加入聊天室时，其他端的在线设备上会接收通知，不在线的设备不会通知。返回数据包括：聊天室 ID、加入时间。
用户多端登录，在一台设备上退出聊天室	当前退出的用户	当前用户在一端退出聊天室时，其他端的在线设备上会接收通知，不在线的设备不会通知。 返回数据包括：聊天室 ID、退出时间。
用户多端登录且已在聊天室中，加入一个新的聊天室导致被踢出上一个聊天室	当前用户，和被踢出的聊天室所有成员	如果在控制台开通了单个用户加入多个聊天室，则不会通知。返回数据包括：聊天室 ID、被踢出的时间。

封禁相关事件通知

> 提示
>
> SDK 从 5.4.5 版本开始支持监听以下事件通知。

用户所在聊天室发生了封禁、解封的事件，且调用相关 Server API 时指定了需要通知（指定 needNotify 为 true），触发 event.chatroomNotifyBlock 事件。返回数据详细定义参见 IChatroomNotifyBlock。

触发场景	通知范围	说明
封禁聊天室用户	聊天室中所有成员	返回数据包括：聊天室 ID、被封禁成员用户 ID 列表、封禁时间与持续时长、附加信息。
解除封禁聊天室用户	被解除封禁的成员	返回数据包括：聊天室 ID、当前用户 ID、附加信息。

禁言相关事件通知

> 提示
>
> SDK 从 5.4.5 版本开始支持监听以下事件通知。

用户所在聊天室发生了禁言、解除禁言相关的事件，且调用相关 Server API 时指定了需要通知（指定 needNotify 为 true），触发 event.chatroomNotifyBan 事件。返回数据详细定义参见 IChatroomNotifyBan。

触发场景	通知范围	说明
禁言指定聊天室用户	聊天室中所有成员	返回数据包括：聊天室 ID、被封禁成员用户 ID 列表、禁言时间与持续时长、附加信息。
取消禁言指定聊天室用户	聊天室中所有成员	返回数据包括：聊天室 ID、被封禁成员用户 ID 列表、解除禁言时间、附加信息。
设置聊天室全体禁言	聊天室中所有成员	返回数据包括：聊天室 ID、禁言时间、附加信息。
取消聊天室全体禁言	聊天室中所有成员	返回数据包括：聊天室 ID、解除禁言时间、附加信息。
加入聊天室全体禁言白名单	聊天室中所有成员	返回数据包括：聊天室 ID、被添加白名单的用户 ID 列表、设置白名单时间、附加信息。
移出聊天室全体禁言白名单	聊天室中所有成员	返回数据包括：聊天室 ID、被移出白名单的用户 ID 列表、移出白名单时间、附加信息。
全局禁言用户	被全局禁言的用户	返回数据包括：聊天室 ID、禁言时间与持续时长、附加信息。
取消全局禁言用户	被解除全局禁言的用户	返回数据包括：聊天室 ID、解除禁言时间、附加信息。

查询聊天室房间信息

更新时间:2024-08-30

获取聊天室的信息，可返回以下数据：

- 聊天室成员总数
- 指定数量（最多 20 个）的聊天室成员的列表，包括该成员的用户 ID 以及加入聊天室的时间

> 提示
>
> **频率限制：单个设备每秒钟支持调用一次，每分钟单个设备最多调用 20 次。**

您可以使用 getChatRoomInfo 方法：

```
const chatroomId = "<聊天室 ID>"
const count = 10 // 获取房间人员列表数量，最大有效值 `20`，最小值为 `0`，默认为 0
const order = 1 // 人员排序方式，`1` 为正序，`2` 为倒序，默认为 0
RongIMLib.getChatRoomInfo(chatRoomId, {
count: count,
order: order
}).then(res => {
// 获取聊天室信息成功
if(res.code === 0){
console.log(res.data)
} else {
console.log(res.code, res.data)
}
})
```

参数	类型	说明
chatRoomId	string	聊天室 ID
options.count	number	需要获取的成员信息的数量，范围：1 - 20。 如果不传或传入默认值（0），则获取到的聊天室信息将仅包含成员总数，不包含具体的成员列表。
options.order	number	按照何种顺序返回聊天室成员信息。升序（1）返回最早加入的用户列表。降序（2）返回最晚加入的用户列表。

获取聊天室历史消息　　开通服务

更新时间:2024-08-30

使用 getChatroomHistoryMessages 要求开通 聊天室消息云端存储 服务。使用前请确认已开通服务。开通后聊天室历史消息保存在云端，默认保存 2 个月。

获取聊天室历史消息

获取保存在服务端的聊天室历史消息。

```
const chatRoomId = "聊天室 ID";
const timestamp = 0;
const count = 10
const order = 1
RongIMLib.getChatroomHistoryMessages(chatRoomId, {
timestamp: timestamp,
count: count,
order: order
}).then(res => {
// 获取聊天室历史信息成功
if (res.code === 0){
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
targetId	string	聊天室 ID
timestamp	number	获取时间戳. 0 为从当前最新时间拉取,单位: 毫秒。
count	number	要获取的消息数量, 范围 1 - 20
order	number	获取顺序，默认为 0，0 表示升序：获取消息发送时间比传入 sentTime 小 的消息;1 表示倒序：获取消息发送时间比传入 sentTime 大 的消息

聊天室属性管理（KV）

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

> 提示
>
> - 聊天室销毁后，聊天室中的自定义属性同时销毁。
> - 每个聊天室中，最多允许设置 **100** 个属性信息，以 `Key-Value` 的方式进行存储。
> - 客户端 SDK 未针对聊天室属性 KV 的操作频率进行限制。建议每个聊天室，每秒钟操作 `Key-Value` 频率保持在 **100** 次及以下（一秒内单次操作 100 个 KV 等同于操作 100 次）。

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）。

设置聊天室 KV 监听器

调用示例： `addEventListener(Events.CHATROOM, listener)`

调用 addEventListener

```
const listener = (event) => {
if (event.rejoinedRoom) {
console.log('SDK 内部重连聊天室信息:', event.rejoinedRoom)
}
if (event.updatedEntries) {
console.log('监听到的聊天室 KV 更新:', event.updatedEntries)
}
if (event.userChange) {
console.log('加入退出的用户通知:', event.userChange)
}
if (event.chatroomDestroyed) {
console.log('聊天室销毁:', event.chatroomDestroyed)
}
}
RongIMLib.addEventListener(Events.CHATROOM, listener)
```

获取单个属性

调用 getChatRoomEntry 获取单个属性。

```
const chatRoomId = "聊天室 ID";
const key = "name";

RongIMLib.getChatRoomEntry(chatRoomId, key).then(res => {
// 获取聊天室单个属性成功
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	string	聊天室 ID
key	string	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

获取所有属性

调用 getAllChatRoomEntries 获取所有属性。

```
const chatRoomId = "聊天室 ID";
RongIMLib.getAllChatRoomEntries(chatRoomId).then(res => {
// 获取聊天室所有属性成功
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

设置单个属性

调用 setChatRoomEntry 设置属性。

```
const entry = {
key: 'key',
value: 'value',
notificationExtra: 'extra',
isAutoDelete: true,
isSendNotification: false
}
RongIMLib.setChatRoomEntry(chatRoomId, entry).then(res => {
// 设置聊天室单个属性成功
if(res.code === 0){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	String	聊天室 ID
key	String	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	String	聊天室属性对应的值，最大长度 4096 个字符
isSendNotification	Boolean	设置成功后是否发送通知消息
isAutoDelete	Boolean	用户退出聊天室时是否清除此属性
notificationExtra	String	RC:chrmKVNotiMsg 消息中携带的附加信息

强制设置单个属性

强制设置单个聊天室属性。可用于修改他人创建的属性值。调用 forceSetChatRoomEntry 强制设置聊天室属性。以 key = value 的形式存储。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。使用强制设置可修改他人创建的属性值。

```
const entry = {
key: 'key',
value: 'value',
notificationExtra: 'extra',
isAutoDelete: true,
isSendNotification: false
}
RongIMLib.forceSetChatRoomEntry(chatRoomId, entry).then(res => {
// 强制设置聊天室单个属性成功
if(res.code === 0){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	String	聊天室 ID
key	String	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

参数	类型	说明
value	String	聊天室属性对应的值，最大长度 4096 个字符
isSendNotification	Boolean	设置成功后是否发送通知消息
isAutoDelete	Boolean	用户退出聊天室时是否清除此属性
notificationExtra	String	RC:chrmKVNotiMsg 消息中携带的附加信息

批量设置属性

> 提示
>
> 从 SDK 5.3.4 版本开始，该接口同时支持通过 isForce 属性强制设置多个属性值。强制设置可直接覆盖他人创建的属性值。

调用 setChatRoomEntries 批量聊天室属性。

```
const entries = [{key: '',value: ''}]
const options = {
entries,
isAutoDelete:true, // 用户退出聊天室时是否清除此属性
isForce: true // 是否强制覆盖
}
RongIMLib.setChatRoomEntries(chatRoomId, options).then(res => {
// 设置聊天室属性成功
if(res.code === 0){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	String	聊天室 ID
options.entries	Array	聊天室属性键值对集合, 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，聊天室属性对应的值，最大长度 4096 个字符
options.isAutoDelete	Boolean	用户退出聊天室时是否清除此属性
options.isForce	Boolean	是否强制覆盖（该属性要求 SDK 版本 ≧ 5.3.4）

删除单个属性

调用 removeChatRoomEntry 删除聊天室自定义属性。只可删除当前用户所创建的属性。

```
const chatRoomId = '聊天室ID'
const entry = {
key: key,
notificationExtra: extra,
isSendNotification: isSendNotification
}
RongIMLib.removeChatRoomEntry(chatRoomId, entry).then(res => {
// 删除聊天室单个属性成功
if(res.code === 0){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	String	聊天室 ID
key	String	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
isSendNotification	Boolean	设置成功后是否发送通知消息
notificationExtra	String	RC:chrmKVNotiMsg 消息中携带的附加信息

强制删除

调用 forceRemoveChatRoomEntry 强制删除聊天室自定义属性。可删除他人所创建的属性。

```
const chatRoomId = '聊天室ID'
const entry = {
key: key,
notificationExtra: extra,
isSendNotification: isSendNotification
}
RongIMLib.forceRemoveChatRoomEntry(chatRoomId, entry).then(res => {
// 强制删除聊天室属性成功
if(res.code === 0){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	String	聊天室 ID
key	String	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
isSendNotification	Boolean	设置成功后是否发送通知消息
notificationExtra	String	RC:chrmKVNotiMsg 消息中携带的附加信息

批量删除

调用 removeChatRoomEntries 批量删除聊天室属性。

```
const chatRoomId = '聊天室ID'
RongIMLib.removeChatRoomEntries(chatRoomId, {
entries: ['']
}).then(res => {
// 删除聊天室属性成功
if(res.code === 0){
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
chatRoomId	String	聊天室 ID
options.entries	Array	聊天室属性名称集合,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

绑定音视频房间

更新时间:2024-08-30

聊天室与音视频房间绑定成功后，只要音视频房间仍存在，则阻止聊天室自动销毁。

适用场景：

聊天室具有自动销毁机制。在使用融云 RTC 业务的 App 中，可能配合使用 IM SDK 的聊天室业务实现直播聊天、弹幕、属性记录等功能。这种情况下，可以考虑将聊天室与音视频房间绑定，确保聊天室不会在语聊、直播结束前销毁，以免丢失关键数据。

在单独使用聊天室业务情况的下，无需调用该接口。

> 提示
>
> **关于聊天室自动销毁逻辑的说明：**
>
> **聊天室具有自动销毁机制，默认情况下所有聊天室会在不活跃（连续时间段内无成员进出且无新消息）达到 1 小时后踢出所有成员并自动销毁，可延长该时间，也可配置为定时自动销毁。详见服务端文档聊天室销毁机制。**

绑定音视频房间

> 提示
>
> - 客户端 SDK 5.2.1 版本开始支持绑定音视频房间接口。

绑定音视频房间后，当聊天室达到预设的自动销毁条件时，服务端会先检测已绑定的音视频房间（RTCRoomId）是否仍存在：

- 如果绑定的音视频房间仍存在，则聊天室不会销毁。
- 如果绑定的音视频房间已销毁，则直接销毁聊天室。

该接口仅创建从聊天室到音视频房间的单向绑定关系。因此在绑定音视频房间后，音视频房间的主动销毁或自动销毁，并不会直接触发聊天室房间的销毁。关于音视频房间的销毁机制，详见音视频房间销毁机制。

接口说明

调用该接口必须传入聊天室 ID，因此必须在聊天室房间已创建成功之后调用。客户端调用加入聊天室接口时会自动完成创建与加入动作。

该接口不具备用户权限控制功能，建议由业务侧的房主或者主播角色用户加入聊天室房间成功后调用一次，其他用户无需调用。

```
RongIMLib.bindRTCRoomForChatroom({
chatRoomId:'chatRoomId',
rtcRoomId:'rtcRoomId'
}).then((res)=>{
// 绑定成功返回值
if(res.code === 0){
//0 表示绑定成功
console.log(res.code)
} else {
//23410 聊天室不存在
console.log(res.code)
}
})
```

参数说明

参数	类型	说明
chatRoomId	string	聊天室 ID
rtcRoomId	string	RTC 房间 ID

超级群概述

更新时间:2024-08-30

> 客户端 SDK 从 SDK 5.2.0 开始支持超级群。仅 Web 端 IMLib 支持超级群。Electron 解决方案暂不适用于超级群业务。

融云超级群（UltraGroup）提供了一种新的群组业务形态。超级群不设置群成员人数上限，允许用户在超级社群中建立社交关系、在海量信息中聚焦自己感兴趣的内容，帮助开发者打造高用户黏性的群体。超级群组成员最多可加入 100 个超级群，每个超级群下的不同频道之间共享一份超级群成员关系。App 内的超级群数量没有限制。

超级群业务的会话类型（ConversationType）为 RongIMLib.ConversationType.ULTRA_GROUP，用 targetId 表示超级群 ID，channelId 表示超级群频道 ID。除部分接口另有说明外，超级群 Web 客户端接口大部分均在 RongIMLib 中。

开通服务

超级群功能需要在控制台超级群服务 页面开通。仅 **IM 尊享版**支持开通超级群服务。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。

如何使用频道

超级群支持在群会话中创建独立的频道（客户端由 channelId 指定、对应服务端的 busChannel），超级群的会话、消息、未读数等消息数据和群组成员支持分频道进行聚合，各个频道之间消息独立。

频道按类型区分为公有频道与私有频道。公有频道对所有超级群成员开放（无需加入）。该超级群的所有成员都会接收公有频道下的消息。私有频道仅对该频道成员列表上的用户开放。有关私有频道的详细介绍，可参见超级群私有频道概述。

超级群业务提供一个 ID 为 RCDefault 的默认频道。RCDefault 频道对所有超级群成员开放，不可转为私有频道。

对于 App 业务来说，如果仅需实现类似群聊的业务，可以利用超级群无成员上限的特性构建大于 3000 人的超大群。这种场景下，可以让所有消息都在 RCDefault 默认频道中进行收发。建议在调用客户端、服务端 API 时指定频道 ID 为 RCDefault。

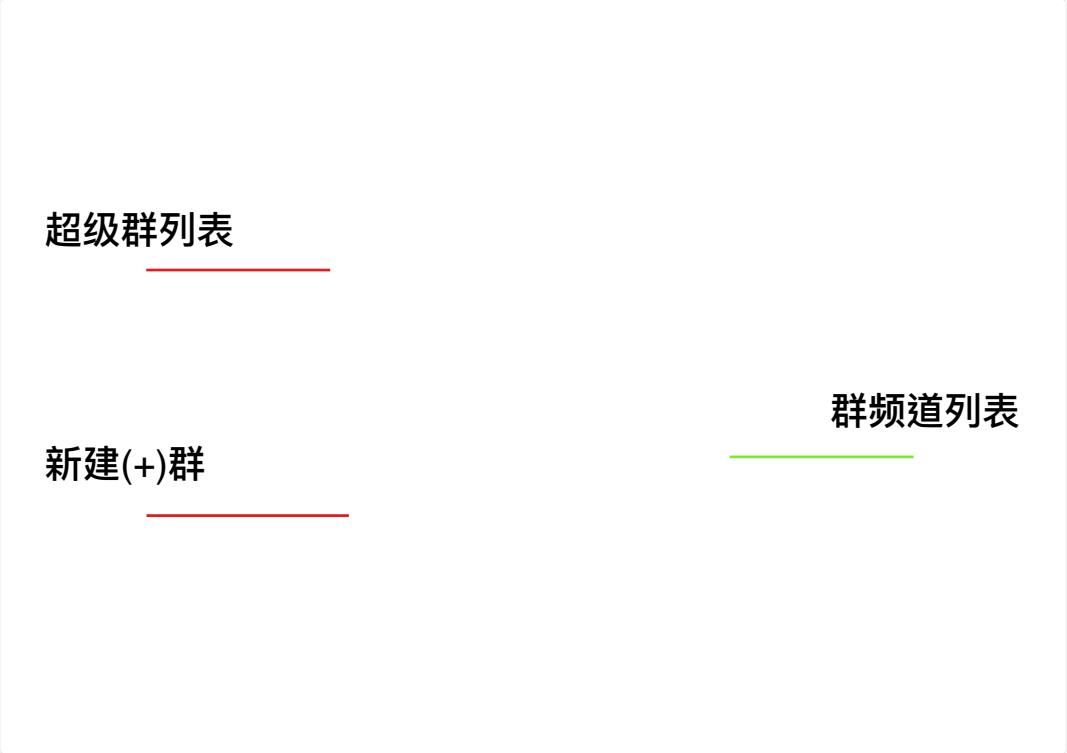
如果仅需实现类似 Discord 类业务，通过超级群频道功能构建子社区，推荐全部使用您自行创建的频道实现您的业务特性。默认频道（RCDefault）与自建频道的行为存在差异，全部使用自建频道可避免这种差异在实现 App 业务逻辑时造成限制。

> 提示
>
> 如果您的 App / 环境在 2022.10.13 日之前开通超级群服务，则您的超级群服务中不存在 RCDefault 频道。在调用客户端、服务端 API 时如果不指定频道 ID，一般仅作用于不属于任何频道的消息，具体行为需参见各功能文档。融云支持客户调整服务至最新行为。该行为调整将影响客户端、服务端收发消息、获取会话、清除历史消息、禁言等多个功能。如有需要，请提交工单咨询详细方案。

客户端 UI 框架参考设计

超级群产品暂不提供 UI 组件。您可以参考以下 UI 框架设计了解超级群 App 的设计思路。

- 下图左侧红框中为超级群列表，即当前登录用户的超级群列表。红框底部的加号（+）按钮代表新建超级群。
- 上图绿框中为超级群频道（Channel）列表。超级群的每一个频道由频道 ID（`channelId`）指定。



超级群管理接口

融云不会托管用户，也不管理群组的业务逻辑，因此超级群的业务逻辑全部需要在 App 服务器进行实现。

对于客户端开发人员来说，创建群组、频道等基础管理操作只需要与 App 服务端交互即可。App 服务端需要调用相应的融云服务端 API（Server API）接口相关接口创建超级群、创建频道等其他管理操作。

下表列出了融云提供的超级群基础管理接口。

> 提示
>
> Server API 还提供超级群全体禁言、超级群用户禁言等更多管理接口。具体请参见融云服务端超级群文档。

功能分类	功能描述	客户端 API	融云服务端 API
创建、解散超级群	提供创建者用户 ID、超级群 ID、和群名称，向融云服务端申请建群。如解散超级群，则群成员关系不复存在。	不提供该 API	创建超级群、解散超级群
加入、退出超级群	加入超级群后，默认可查看入群以后产生的新消息。退出超级群后，不再接受该群的新消息。	不提供该 API	加入超级群、退出超级群
修改融云服务端的超级群信息	修改在融云推送服务中使用的超级群信息。	不提供该 API	更新超级群信息
创建、删除群频道	在超级群会话中创建独立沟通的频道。如删除频道，将无法在频道中发送消息。	不提供该 API	创建频道、删除频道
查询群频道列表	加入超级群后，默认可查看入群以后产生的新消息。退出超级群后，不再接受该群的新消息。	不提供该 API	查询频道列表

功能分类	功能描述	客户端 API	融云服务端 API
变更群频道类型	超级群频道可以随时切换为公有频道或私有频道。	不提供该 API	变更频道类型
添加、删除私有频道成员	将超级群成员加入或移出指定频道的私有频道成员列表。在频道类型为私有频道时启用该成员列表的数据。	不提供该 API	添加私有频道成员、删除私有频道成员

私有频道概述

更新时间:2024-08-30

超级群服务支持创建私有频道，满足社区场景中只有指定用户可以在频道中沟通的业务需求。

已知限制

> 提示
>
> **私有频道功能目前有以下限制：**
>
> **用户即使不在私有频道成员列表中，发送消息会失败，但消息仍会进入本地数据库。如果用户未在私有频道成员列表中，但仍往频道中发送消息，此时消息无法成功发送到服务端，但会进入本地数据库（不适用于 Web 端），并可生成会话。**
>
> **建议解决方案：开发者在 App 业务侧维护一份私有频道成员列表数据，如果用户未在私有频道成员列表中，禁止用户往私有频道发送消息。**

了解私有频道

> 提示
>
> 客户端不提供创建私有频道、变更频道类型的 API。请通过服务端 API 实现。

超级群的频道按类型区分为公有频道与私有频道。公有频道对所有超级群成员开放（无需加入）。该超级群的所有成员都会接收公有频道下的消息。私有频道仅对该频道的成员列表开放，仅频道成员可在该频道中收发消息、接收频道状态变化的通知。私有频道可以随时切换为公有频道，公有频道也可以切换为私有频道。

例如，管理员在社区中新创建一个频道，并定义为私有频道。接着邀请社区中部分成员加入私有频道。只有加入频道的成员才可以浏览此频道，并在频道中接收、发送消息。

频道变更会影响该频道下服务端历史消息的拉取权限，具体如下：

- 公有频道转换为私有频道：仅当前私有频道成员列表中的用户可以拉取该频道下的历史消息（含原公有频道消息）；
- 私有频道转换为公有频道：所有超级群用户均可拉取该频道下的历史消息（含原私有频道消息）。

您可以通过以下方式管理私有频道：

功能描述	客户端 API	融云服务端 API
创建私有频道	不提供该 API	创建频道
删除私有频道	不提供该 API	删除频道
查询频道列表	不提供该 API	查询频道列表
变更频道类型	不提供该 API	变更频道类型

了解私有频道成员列表

> 提示
>
> 客户端 SDK 不提供管理私有频道成员列表的 API。您可以调用服务端 API 查询私有频道成员列表、或将超级群成员加入、移出成员列表。

频道类型为私有频道时，只有该列表中用户可在频道中的收发消息、接收频道内状态通知。

私有频道转为公有频道时，频道变更为对所有超级群成员开放。但该列表数据不会被删除。假设频道类型再次变更为私有，则启用该成员列表。

您也可以为公有频道创建私有频道成员列表。一旦该公有频道变更类型为私有频道，该列表即生效。

> 提示
>
> - 删除频道时，服务端会清除该频道的私有频道成员列表。
> - 一旦超级群成员退群，服务端会自动将用户从私有频道成员列表移除。

您可以通过以下方式管理私有频道成员列表：

功能描述	客户端 API	融云服务端 API
加入私有频道成员列表	不提供该 API	添加私有频道成员
移出私有频道成员列表	不提供该 API	删除私有频道成员
查询私有频道成员列表	不提供该 API	查询私有频道成员列表

用户组概述

更新时间:2024-08-30

超级群用户组（User Group）功能是融云超级群业务提供的群成员管理工具，结合超级群私有频道功能，可以帮助 App 实现更高效的超级群成员管理，沟通管理，和更精细的用户通知能力。

前提条件

> 提示
>
> Web 客户端 IMLib SDK 从 5.7.3 版本开始支持超级群用户组功能。

超级群用户组功能主要通过与私有频道的绑定操作，提供了对用户在社区私有频道中沟通权限（收发消息、通知）的批量管理能力，提升了 App 集成效率与对超级群的运营管理能力。

在了解与使用超级群用户组功能前，需要先了解超级群私有频道功能。请参见超级群私有频道概述。

如何使用超级群用户组

App 服务端可以通过调用融云服务端 API，在超级群中创建最多 50 个用户组（userGroup），每个用户组成员最多由 100 个超级群成员组成。单个用户可以存在于多个用户组中。

用户组创建后，可以与超级群频道绑定。如果用户组绑定了一个或多个私有频道，该用户组的所有成员即具有在绑定的私有频道中收发消息、接收通知的能力。

- App 将用户组绑定私有频道后，可认为组中所有用户均加入了该私有频道。与该私有频道成员列表中的用户类似，只有加入频道的成员才可以浏览此频道，并在频道中接收消息，发送消息，和接收通知。
- App 将用户组绑定公有频道后，不会影响组中用户可收发消息的范围。但一旦该公有频道转为私有频道，该组用户将具有在该私有频道中收发消息、获取通知的能力。
- App 可以在一个频道上绑定最多 10 个用户组，一个用户组可以与多个频道绑定（一个超级群最多 50 个频道）。

> 提示
>
> 超级群业务默认提供 RCDefault 频道，对所有超级群成员开放，不可转为私有频道。建议不要将用户组绑定到 RCDefault 频道。

App 客户端无法进行用户组管理操作，仅可通过 SDK 提供的回调方法监听用户组相关变更的通知，具体包括：

- **删除用户组**：用户组下所有用户均可收到客户端回调
- **用户组成员变更**：被加入或移出用户组的用户可收到客户端回调
- **频道与用户组绑定关系变更**：用户组下所有用户均可收到客户端回调

混合使用用户组与私有频道成员列表

只要 App 用户在指定私有频道绑定的任意一个用户组中，或者在有该私有频道成员列表中，该用户就能在私有频道中收发消息接收通知。

如果混合使用私有频道成员列表与用户组，在 App 业务中可能存在以下情况：

- 私有频道配置了私有成员列表，并添加了多位用户。
- 该私有频道绑定了多个用户组。

在上述使用场景中，某个用户可能既在私有频道成员列表中，又同时在该频道绑定的多个用户组中。如果该用户不再使用该私有频道，App 进行以下操作，确保该用户无法继续在私有频道收发消息：

- 从该私有频道的成员列表中移除该用户。
- 检查该私有频道绑定的所有用户组，从绑定的所有用户组中移除该用户，或解绑用户组。

用户组管理接口

即时通讯（IM）服务端提供了超级群用户组的基础管理接口。用户组的业务逻辑需要 App 服务端自行实现，例如申请加入用户组、审核用户组加入申请等。建议 App 服务端同时维护一份用户、用户组、频道之间对应关系的数据。

对于客户端开发人员来说，创建用户组等基础管理操作只需要与 App 服务端交互即可。App 服务端需要调用相应的融云服务端 API（Server API）接口相关接口创建用户组等其他管理操作。

下表列出了超级群用户组基础管理接口。更多相关接口可参见 IM 服务端文档 API 接口列表 。

功能分类	功能描述	融云服务端 API
创建、删除用户组	在指定超级群下创建用户组。如删除用户组，则用户、用户组、频道之间的关系不复存在。	创建用户组 、删除用户组
查询用户组列表	分页查询指定超级群下的用户组，返回用户组 ID 列表。	查询用户组列表
添加、删除用户	在超级群用户组中添加、删除用户。	添加用户 、移出用户
查询用户所属用户组	分页查询指定单个用户在超级群下所属的用户组列表，返回用户组 ID 列表。	查询用户所属用户组
绑定频道与用户组	将指定单个超级群频道与用户组绑定，可绑定多个用户组，单个频道最多支持与 10 个用户组绑定。	绑定频道与用户组
解绑频道与用户组	将指定单个超级群频道与用户组解除绑定，单次请求可解绑最多 10 个用户组。	解绑频道与用户组
查询频道绑定的用户组	分页查询指定的超级群频道绑定的用户组列表，返回用户组 ID 列表。	查询频道绑定的用户组
查询用户组绑定的频道	分页查询指定单个用户组绑定的超级群频道列表，返回超级群频道 ID 列表。	查询用户组绑定的频道

超级群快速上手

更新时间:2024-08-30

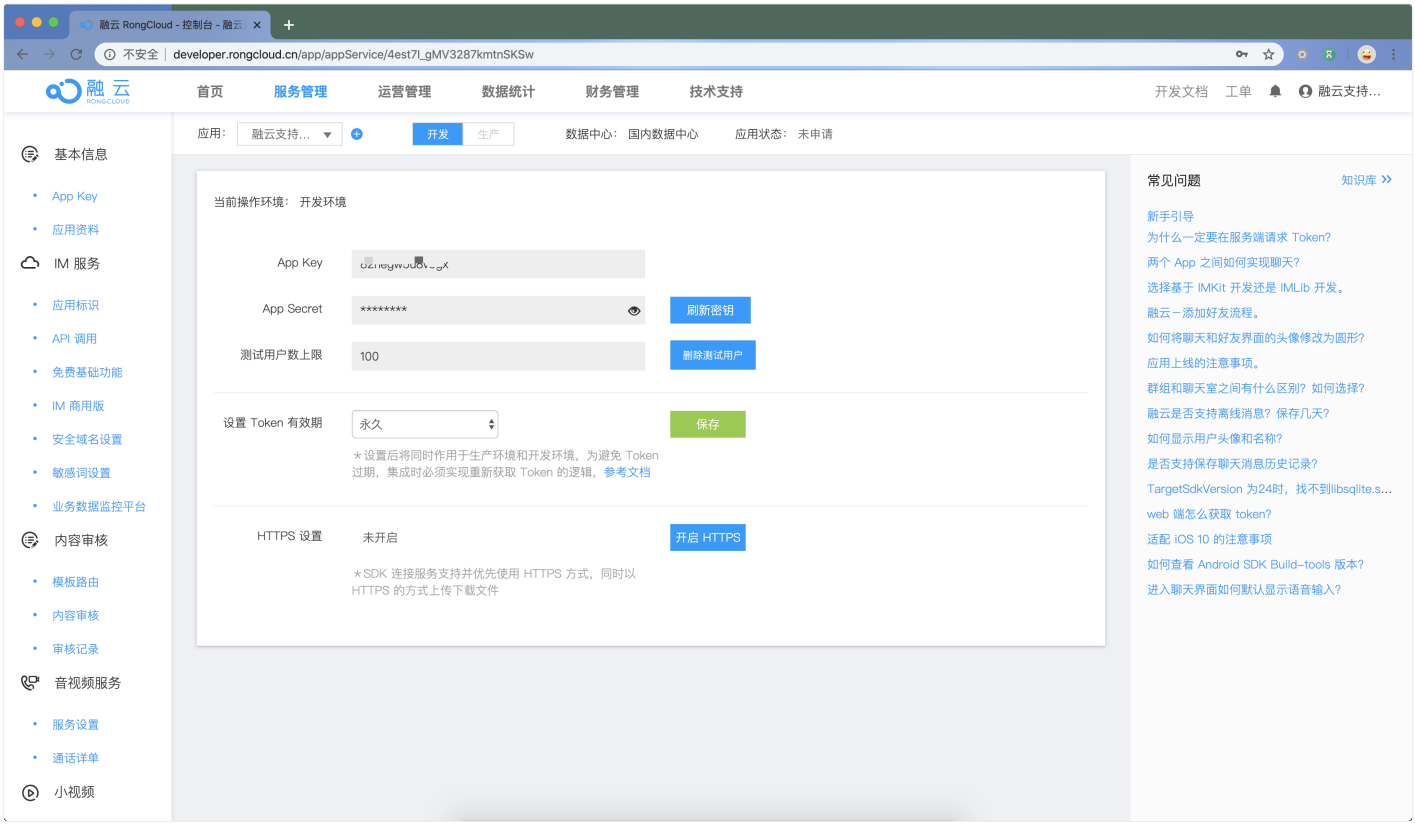
本教程是为了让新手快速了解融云即时通讯能力库（IMLib）。在本教程中，你可以体验集成 SDK 的基本流程和 IMLib 的超级群通信能力。

融云开发者账户

融云开发者账户是使用融云 SDK 产品的必要条件。在开始之前，请先前往融云官网注册开发者账户。注册后，控制台将自动为你创建一个应用，默认为开发环境应用，使用国内数据中心。请获取该应用的 App Key，在本教程中使用。

> 提示
>
> App Secret 用于生成数据签名，仅在请求融云服务端 API 接口时使用。本教程中暂不涉及。

如果您已拥有融云开发者账户，你可以直接选择合适的环境，创建应用。



您需要记录上图所示的应用 App Key，在本教程中使用。

应用的 AppKey 与 Secret 是获取连接融云服务器身份凭证的必要条件，请注意不要泄露。

导入 SDK

> 提示

> IMLib 对 Typescript 的使用者提供了友好的类型化支持，推荐开发者使用 Typescript 进行业务开发以提升代码健壮性及可维护性。

NPM 引入（推荐）

1. 依赖安装

```
npm install @rongcloud/engine -S
npm install @rongcloud/imlib-next -S
```

2. 代码集成

```
// 非 ESModule
const RongIMLib = require('@rongcloud/imlib-next')
// ESModule
import * as RongIMLib from '@rongcloud/imlib-next'
```

CDN 引入

```
<script src="https://cdn.ronghub.com/RongIMLib-5.9.5.prod.js"></script>
```

App Key

App Key 是使用 IMLib 进行即时通讯功能开发的必要条件，也是应用的唯一性标识。在集成使用 IMLib 之前，请务必先通过控制台注册并获取开发者的专属 App Key。

只有在 App Key 相同的情况下，不同用户之间的消息才能互通。

初始化

在使用 IMLib 的能力之前，必须先调用 **IMLib 的初始化接口**，且务必保证该接口在应用全生命周期内仅被调用一次。

```
// 应用初始化以获取 RongIMLib 实例对象，请务必保证此过程只被执行一次
RongIMLib.init({ appkey: '<Your-App-Key>' })
```

设置监听

初始化完成后，添加事件监听器，及时获取相关事件通知。IM 在初次连接成功后，需立即同步消息，获取全量超级群会话列表，列表同步完成前，所有超级群相关功能接口不可用

```
// 添加事件监听
const Events = RongIMLib.Events

RongIMLib.addEventListener(Events.CONNECTING, () => {
console.log('正在连接服务器')
})

RongIMLib.addEventListener(Events.CONNECTED, () => {
console.log('已经连接到服务器')
})

RongIMLib.addEventListener(Events.MESSAGES, (evt) => {
console.log(evt.messages)
})

// 会话列表同步完成，可通过监听 Events.ULTRA_GROUP_ENABLE 事件以通知业务层会话列表同步完成
RongIMLib.addEventListener(Events.ULTRA_GROUP_ENABLE, (conversationList) => {
console.log(conversationList)
})

// 会话状态变动监听 包含：会话未读数、@ 未读数、免打扰状态
RongIMLib.addEventListener(Events.CONVERSATION, ({ conversationList }) => {
console.log(conversationList)
})
```

建立 IM 连接

App Key 是应用的唯一性标识，Token 则是用户的唯一性标识，是用户连接融云 IM 服务所必需的身份凭证。Token 一般由开发者的应用服务器调用融云 Server API 获取 Token 接口获取之后，由应用服务器下发到应用客户端。

> 提示
>
> 以下示例代码假定客户端已获取 Token

```
RongIMLib.connect('<Your-Token>').then((res) => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
console.log('连接成功, 连接用户 id 为: ', res.data.userId)
} else {
console.warn('连接失败, code:', res.code)
}
})
```

获取所有超级群会话列表

> 提示
>
> Web 端不具备持久化的数据存储能力，需要开发者开启超级群消息云端存储（参见开通超级群服务）功能才能生效。
>
> 该功能需要在调用 RongIMLib.connect() 并且建立连接成功之后执行。

IMLib 通过会话数据中的 conversationType 与 targetId 与 channelId 三个属性值来标识会话的唯一性，对于三个属性的定义如下：

1. `conversationType` 用来标识会话类型（如：超级群...），其值为 `RongIMLib.ConversationType` 中的常量定义

2. `targetId` 用来标识与本端进行对话的超级群会话 Id：

- 当 conversationType 值为 RongIMLib.ConversationType.ULTRA_GROUP，targetId 为对方超级群会话 Id

```
// 获取超级群会话列表
RongIMLib.getUltraGroupList().then(({ code, data: conversationList }) => {
if (code === 0) {
console.log('获取超级群会话列表', conversationList)
} else {
console.log('获取超级群会话列表: ', error.code, error.msg)
}
})
```

发送消息

> 提示
>
> 该功能需要在调用 RongIMLib.connect() 并且建立连接成功之后执行。

以发送文本消息为例：

```
// 指定消息发送的目标会话
const conversation = {
// targetId
targetId: '<TargetId>',
// 会话类型：RongIMLib.ConversationType.ULTRA_GROUP
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
// 频道 ID
channelId: '<频道 ID>',
}

// 构建文本消息
const message = new RongIMLib.TextMessage({ content: 'message content' })

// 发送消息
RongIMLib.sendMessage(conversation, message).then(({ code, data }) => {
if (code === 0) {
console.log('消息发送成功：', data)
} else {
console.log('消息发送失败：', code)
}
})
```

接收消息

当本端作为消息接收的一方，所接收的消息将通过 RongIMLib.addEventListener 和 Events.MESSAGES 注册的消息监听向业务层抛出。具体可参考上述 设置监听 部分

获取历史消息

> 提示
>
> Web 端不具备持久化的数据存储能力，需要开发者开启超级群消息云端存储（参见开通超级群服务）功能才

能生效。

该功能需要在调用 RongIMLib.connect() 并且建立连接成功之后执行。

```
const conversation = {
targetId: '<TargetId>',
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
channelId: '<频道ID>',
}
const option = {
// 获取历史消息的时间戳，默认为 0，表示从当前时间获取
timestamp: +new Date(),
// 获取条数，有效值 1-100，默认为 20
count: 20,
}
RongIMLib.getHistoryMessages(conversation, option)
.then((result) => {
const list = result.list // 获取到的消息列表
const hasMore = result.hasMore // 是否还有历史消息可获取
console.log('获取历史消息成功', list, hasMore)
})
.catch((error) => {
console.log('发送文字消息失败', error.code, error.msg)
})
```

断开连接

> 提示
>
> 断开当前用户连接，连接断开后无法接收消息、发送消息、获取历史消息、获取会话列表...
>
> 在下次连接融云成功后，会收取上次离线后的消息，离线消息默认保存 7 天。

```
RongIMLib.disconnect().then(() => console.log('断开连接成功'))
```

后续步骤

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

创建超级群与频道

更新时间:2024-08-30

客户端 SDK 不提供创建超级群与创建群频道的接口。请使用融云服务端 API（Server API）的相关接口创建超级群、群频道，或进行其他管理操作。

> 提示
>
> 单个用户最多可以加入 100 个超级群。单个用户在每个群中最多可以加入或者创建 50 个频道。

本文仅简单介绍创建超级群与创建频道的基本流程。

创建超级群

创建超级群必须使用融云服务端 API（Server API），具体接口使用方法请参见服务端文档创建超级群。

基本流程

1. App 客户端请求 App 服务端（AppServer）创建超级群。
2. App 服务端调用融云 Server API 接口创建超级群，群组 ID 由 App 服务器自行生成
3. 超级群创建成功后，由 App 服务端返回给 App 客户端。

如何处理与展示超级群列表

AppServer 需要保存当前用户的超级群列表，并下发给 App，然后进行展示。

> 考虑到 App 由于自身业务需求，需要知晓当前用户的超级群列表（例如用户所在的超级群有等级权重等排序规则），而融云的超级群列表是通过消息产生的，因而两个列表可能不完全一样，建议由 App 按照自身业务保存用户的超级群列表。

创建群频道

创建超级群必须使用融云服务端 API（Server API），具体接口使用方法请参见服务端文档创建频道。

基本流程

如何处理与展示超级群频道列表

为方便在同一超级群下的不同用户按需看到自己需要的列表（例如用户对特定频道标星需要特别展示，APP 服务则需要按用户记录），APP 服务应采取更为灵活的方式维护超级群的频道列表。

APP 服务端也可以按照需求将超级群分组（如超级群概述中的 UI 框架设计所示）。

获取频道列表

更新时间:2024-08-30

App 可按需使用客户端 SDK 与融云服务端 API 提供的能力，采取灵活的方式维护超级群的频道列表。

> 提示
>
> - 如果您的应用/环境在 2022.10.13 日及以后开通超级群服务，超级群业务中会包含一个 ID 为 `RCDefault` 的默认频道。如果发消息时不指定频道 ID，则该消息会发送到 `RCDefault` 频道中。在获取 `RCDefault` 频道的历史消息时，需要传入该频道 ID。
> - 如果您的应用/环境在 2022.10.13 日前已开通超级群服务，在发送消息时如果不指定频道 ID，则该消息不属于任何频道。获取历史消息时，如果不传入频道 ID，可获取不属于任何频道的消息。融云支持客户调整服务至最新行为。该行为调整将影响客户端、服务端收发消息、获取会话、清除历史消息、禁言等多个功能。如有需要，请提交工单咨询详细方案。

如何获取超级群全部频道列表

超级群下全部频道的列表可通过融云服务端 API (`/ultragroup/channel/get.json`) 获取。

注意，对单个用户来说，最多可以加入 100 个超级群，在每个超级群中最多可以加入或者创建 50 个频道。

> 提示
>
> 服务端会通过频道中收发的消息生成一个超级群频道列表。该列表仅包含了已产生了消息的频道，可能并非该超级群下全部频道的列表。

建议从 App 业务服务端维护超级群的频道列表。App 业务一般需要知晓当前用户的所加入的超级群，以及超级群下的频道列表，才能实现特定的业务功能。假设同一超级群下的不同用户需要展示个性化的频道列表（例如 App 需要在 UI 上展示用户标星的特定频道），则需要为用户保存超级群频道列表。

APP 服务端也可以按照需求将超级群分组（如超级群概述中的 UI 框架设计所示）。

获取当前登录用户的超级群会话列表

getUltraGroupList 获取超级群会话列表。该列表不包含未产生任何消息的会话。

参数说明

参数	类型	说明
options	IGetUltraGroupListOption	可选项。参见下方 `IGetUltraGroupListOption` 参数说明。

- `IGetUltraGroupListOption` 参数说明

参数	类型	说明
targetId	string	（可选项）超级群 ID
channelType	number	（可选项） 频道类型，值为 0 或 1，0 为公有频道，1 为私有频道

代码示例

```
const options = {
target: 'xxx',
channelType: 0
}
// options 不传为获取所有超级群会话列表；
// 获取指定频道类型的超级群频道列表：RongIMLib.getUltraGroupList(options)
RongIMLib.getUltraGroupList()
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

获取当前登录用户免打扰超级群会话列表

getBlockUltraGroupList

代码示例

```
RongIMLib.getBlockUltraGroupList()
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

批量获取超级群会话信息

getUltraGroupUnreadInfoList 根据传入要查询的会话 id，获取该会话下所有频道的 IUltraGroupUnreadInfo 信息

代码示例

```
const targetIds = ['id01', 'id02']
RongIMLib.getUltraGroupUnreadInfoList(targetIds)
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

参数	类型	说明
targetIds	string[]	数组长度大于 0, 小于等于 20

监听频道状态变更　　监听超级群频道类型切换通知

更新时间:2024-08-30

可通过监听 Events.ULTRA_GROUP_CHANNEL_TYPE_CHANGE 事件来捕获超级群频道类型切换的通知。

```
RongIMLib.addEventListener(RongIMLib.Events.ULTRA_GROUP_CHANNEL_TYPE_CHANGE, (list) => {
const { target, channelId, channelType, changeType } = list[0]
// channelType 0 为超级群公有频道；1 为超级群私有频道；
// changeType 2 为 超级群公有频道变成了私有频道；3 为超级群私有频道变成了公有频道；6 为超级群频道变成了私有频道，但是当前用户不再该私有频道中
})
```

list 参数说明

参数	类型	说明
list	IUltraChannelChangeInfo[]	频道类型切换通知列表

IUltraChannelChangeInfo

参数	类型	说明
targetId	string	超级群 ID
channelId	string	超级群频道 ID
channelType	number	超级群频道类型：0 超级群公有频道；1 超级群私有频道
changeType	number	超级群频道类型变更通知：2 超级群公有频道变成了私有频道；3 超级群私有频道变成了公有频道；6 超级群公有频道变成了私有频道，但是当前用户不在该私有频道中；

频道类型的变更的通知范围

- 公有频道变私有频道：公有频道变私有频道时，所有用户都会收到通知。但根据用户是否在私有频道成员列表中，收到的通知有差异：
  - 在私有频道成员列表内的用户，收到的变更类型是 2（超级群公有频道变成了私有频道）。
  - 不在私有频道成员列表的用户，收到的变更类型为 6（超级群公有频道变成了私有频道，但是当前用户不在该私有频道中）。

    如有需要，可在公有频道变私有频道前，提前指定用户加入私有频道成员列表。
- 私有频道变公有频道：私有频道变公有频道时，仅在私有频道成员列表的用户会收到通知，变更类型为 3（超级群私有频道变成了公有频道）。

监听超级群私有频道用户移出白名单通知

可通过监听 Events.ULTRA_GROUP_CHANNEL_USER_KICKED 事件来捕获超级群私有频道用户移出白名单的通知。

```
RongIMLib.addEventListener(RongIMLib.Events.ULTRA_GROUP_CHANNEL_USER_KICKED,  (list) => {
const { target, channelId, channelType, userId } = list[0]
// channelType 0 为超级群公有频道；1 为超级群私有频道；
// userId 被移出超级群的用户 ID
}
```

list 参数说明

参数	类型	说明
list	IUltraChannelUserKickedInfo[]	用户移出白名单通知列表

IUltraChannelUserKickedInfo

参数	类型	说明
targetId	string	超级群 ID
channelId	string	超级群频道 ID
channelType	number	超级群频道类型：0 超级群公有频道；1 超级群私有频道
userId	number	被移出白名单用户的 userId

私有频道成员列表变更的通知范围

- 如果频道类型为私有频道，将用户从私有频道成员列表移除时，仅通知被移除的用户
- 如果频道类型为公有频道，将用户从私有频道成员名单移除时，不发送通知。注意，该列表在该公有频道变更类型为私有频道时才会生效。

监听超级群频道删除通知

可通过监听 Events.ULTRA_GROUP_CHANNEL_DELETE 事件来捕获超级群频道删除的通知。

```
RongIMLib.addEventListener(RongIMLib.Events.ULTRA_GROUP_CHANNEL_DELETE,  (list) => {
const { target, channelId, channelType, deleteTime } = list[0]
// channelType 0 为超级群公有频道；1 为超级群私有频道；
// deleteTime 删除时间戳
})
```

list 参数说明

参数	类型	说明
list	IUltraChannelDeleteInfo[]	删除超级群频道通知列表

IUltraChannelDeleteInfo

参数	类型	说明
targetId	string	超级群 ID
channelId	string	超级群频道 ID
channelType	number	超级群频道类型：0 超级群公有频道；1 超级群私有频道
deleteTime	number	超级群频道删除时间

频道已删除的通知范围

- 删除公有频道的通知：所有用户会收到通知。
- 删除私有频道的通知：仅在私有频道成员列表的用户会收到通知。

监听用户组状态变更

更新时间:2024-08-30

> SDK 从 5.7.3 版本开始支持超级群用户组功能。

客户端可设置监听，在超级群用户组发生变更时收到取对应通知。

客户端 SDK 针对以下操作提供回调。回调的通知范围与回调数据有差异具体如下：

- **用户加入用户组**：被加入用户组的用户可收到通知，通知中携带超级群 ID、用户组 ID。
- **用户被移出用户组**：被移出的用户可收到通知，通知中携带超级群 ID、用户组 ID。
- **频道与用户组绑定**：用户组下所有用户均可收到通知，通知中携带超级群 ID、频道 ID、频道类型、用户组 ID。
- **频道与用户组解除绑定**：用户组下所有用户均可收到通知，通知中携带超级群 ID、频道 ID、频道类型、用户组 ID。
- **删除用户组**：用户组下所有用户均可收到通知，通知中携带超级群 ID、用户组 ID。

监听超级群用户组状态变更通知

可通过监听 `Events.USER_GROUP_STATUS` 事件来捕获超级群用户组状态变更的通知。

```
RongIMLib.addEventListener(RongIMLib.Events.USER_GROUP_STATUS, (info) => {
const { userGroupDisband, userAdded, userRemoved, userGroupBindChannel, userGroupUnBindChannel } = info
if (userGroupDisband) {
// 用户组解散
// 需要将符合条件的用户从符合条件的频道中移除。
// 轮询判断该用户组内的每个用户：
// 1、是否还在该用户组绑定的某个频道中。
// 2、是否还在该用户组绑定的某个频道的其它用户组中。
// 若结果均为否，则可以移除。（解除该用户组下成员的绑定关系。解除该用户组与频道的绑定关系）
}
if (userAdded) {
// 用户被加入用户组
// 该用户组绑定的每个私有频道对该用户显示会话入口。
// 并支持在该部分私有频道进行消息的收发、未读消息的展示等
}
if (userRemoved) {
// 用户被从用户组中移除
// 需要将用户从符合条件的频道中移除。
// 轮询判断该用户：
// 1、是否还在该用户组绑定的每个频道中。
// 2、是否还在该用户组绑定的每个频道的其它用户组中。
// 若结果均为否，则可以移除。（解除用户同该用户组之间的绑定关系。如该用户组绑定了私有频道，则该用户无法在私有频道中收发消息，获取历史消息内容。）
}
if (userGroupBindChannel) {
// 用户组绑定频道
// 需要针对该用户组内的用户显示该频道入口。
// 并支持在该部分私有频道进行消息的收发、未读消息的展示等
}
if (userGroupUnBindChannel) {
// 用户组解绑频道
// 需要针对该用户组内符合条件的用户隐藏该频道入口。
// 轮询判断该用户组内的每个用户：
// 1、是否还在该频道中。
// 2、是否还在该频道的其它用户组中。
// 若结果均为否，则可以隐藏。
}
})
```

info 参数说明

参数	类型	说明
info	IUserGroupStatusInfo	超级群用户组状态数据

- **IUserGroupStatusInfo**：

参数	类型	说明
userGroupDisband	IUserGroupChangeData	用户组解散信息
userAdded	IUserGroupChangeData	用户被加入用户组信息
userRemoved	IUserGroupChangeData	用户被从用户组中移除信息
userGroupBindChannel	IChannelAndUserGroupChangeData	用户组绑定频道信息
userGroupUnBindChannel	IChannelAndUserGroupChangeData	用户组解绑频道信息

关于更新 UI 的提示

考虑到同一用户既在私有频道成员列表中，又在私有频道绑定的（多个）用户组中，App 可以在收到回调时向 App 自身业务服务端查

询当前用户是否仍可继续访问该私有频道，并根据该结果刷新 UI。

对于未在通知范围内的用户，可以在用户进入相应页面时向 App 自身业务服务端查询数据，并决定是否需要刷新 UI。

收发消息

更新时间:2024-08-30

本文介绍了如何从客户端发送超级群消息。

前置条件

建议先阅读超级群概述和超级群私有频道概述，了解在 App 业务中如何使用频道和超级群频道功能特性。

- 通过服务端 API 创建超级群
- 通过服务端 API 创建频道，或使用默认频道 ID `RCDefault`
- 通过服务端 API 将发件人加入超级群
- 如不确定发件人是否在超级群中，请通过服务端 API 查询用户是否为群成员
- 如向超级群私有频道中发送消息，请确认已通过服务端 API 添加私有频道成员

发送消息

基础接口 sendMessage 可用来发送所有自定义消息或 IMLib 中的内置类型消息。如发送文件、图片、高清语音、小视频、动图类型消息，可参见下文描述的语法糖方法。

`sendMessage` 方法接收 `conversation`、`message` 与 `options` 三个参数：

- **conversation**： 消息需要投送的目标会话。参见 IConversationOption。

参数	类型	说明
conversationType	ConversationType	会话类型（RongIMLib.ConversationType.ULTRA_GROUP）
targetId	string	接收方 Id
channelId	string	频道 Id（可选项）

- **message**：待发送的消息内容，可以是如 `RongIMLib.TextMessage` 的 IMLib 内置消息实例，也可以是通过 `RongIMLib.registerMessageType()` 实现的自定义消息实例。

- **options**：可选参数，用于定义发送行为中的一些可选项，如是否可扩展，推送等。参见 ISendMessageOptions。

options 参数	类型	说明
~~isStatusMessage~~	boolean	（已废弃）是否为状态消息（可选项）
disableNotification	boolean	是否发送静默消息（可选项）
pushContent	string	Push 信息（可选项）
pushData	string	Push 通知携带的附加信息（可选项）

options 参数	类型	说明
isMentioned	boolean	是否为 @ 消息（可选项）
mentionedType	1 | 2	@ 消息类型，1: @ 所有人 2: @ 指定用户（可选项）
mentionedUserIdList	string[]	被 @ 的用户 Id 列表（可选项）
directionalUserIdList	string[]	用于发送普通群组定向消息，超级群暂不支持该功能。
isVoipPush	boolean	当对方为 iOS 设备且未在线时，其将收到 Voip Push. 此配置对 Android 无影响（可选项）
canIncludeExpansion	boolean	是否允许消息被拓展
expansion	[key: string]: string	消息拓展内容数据（可选项）
isFilerWhiteBlacklist	boolean	黑/白名单（可选项）
pushConfig	IPushConfig	移动端推送配置（可选项）（与 Android、iOS 端的 MessagePushConfig 作用相似）。详见下方的 **IPushConfig 参数说明**。

- **IPushConfig 参数说明**

参数	类型	说明
pushTitle	string	推送标题，在没有设置的情况下：单聊通知标题显示为发送者名称，群聊通知标题显示为群名称；自定义消息，默认不显示标题；
pushContent	string	推送内容
pushData	string	远程推送附加信息
iOSConfig	IiOSPushConfig	（可选项）
androidConfig	IAndroidPushConfig	（可选项）
disablePushTitle	boolean	是否显示推送标题. 仅针对 iOS 平台有效
forceShowDetailContent	boolean	是否强制推送
templateId	string	推送模板 ID

代码示例

以发送文本消息为例：

```
// 定义消息投送目标会话
const conversation = {
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '<目标 Id>',
channelId: '<频道ID>',
}
// 实例化待发送消息，RongIMLib.TextMessage 为内置文本型消息
const message = new RongIMLib.TextMessage({ content: '' })
// 发送
RongIMLib.sendMessage(conversation, message).then((res) => {})
```

语法糖

发送 FileMessage、ImageMessage、HQVoiceMessage、SightMessage、GIFMessage 五种消息时，在构建消息实例过程中均需要一个远程资源地，这意味着业务层需要将待发送的本地资源先行上传，再构建消息实例，过程较为复杂。

为此，IMLib 实现了相应的语法糖方法以封装本地资源的上传过程、以及消息构建过程，便于业务层更易于集成。

发送文件消息

```
RongIMLib.sendFileMessage(
{
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '',
channelId: '',
},
{
file, // 待上传文件
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，非必填
extra: '', // 消息中携带的透传信息，非必填
},
{
onProgress(progress) {}, // 上传进度监听，可选
onComplete(fileInfo) {
// 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 FileMessage
},
},
{
// ... 其他配置项，可选
}
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

发送图片消息

```
RongIMLib.sendImageMessage(
{
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '',
channelId: '',
},
{
file, // 待上传文件
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，非必填
extra: '', // 消息中携带的透传信息，非必填
},
{
onProgress(progress) {}, // 上传进度监听，可选
onComplete(fileInfo) {
// 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 ImageMessage
},
},
{
// ... 其他配置项，可选
}
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

发送高清语音消息

> 提示
>
> 如果您的业务同时使用了 Android/iOS 端的 IMKit SDK，请使用默认的 aac 格式，因为 IMKit 的音频录制、播放只实现了对 aac 格式的支持，默认无法播放其他格式。如果您的 Android/iOS App 会自行处理音频录制、播放，可以按需选用格式。

```
RongIMLib.sendHQVoiceMessage(
{
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '',
channelId: '',
},
{
file, // 待上传文件
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，非必填
extra: '', // 消息中携带的透传信息，非必填
},
{
onProgress(progress) {}, // 上传进度监听，可选
onComplete(fileInfo) {
// 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 HQVoiceMessage
},
},
{
// ... 其他配置项，可选
}
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

发送短视频消息

> 提示
>
> 如果您的业务同时使用了 Android/iOS 端的 IMKit SDK，必须使用 H.264 + AAC 编码的文件，因为 IMKit 的短视频录制、播放只实现了该编码组合的支持。

```
RongIMLib.sendSightMessage(
{
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '',
channelId: ''
},
{
file, // 待上传文件
duration: number, // 视频时长
thumbnail: string, // 视频首帧缩略图，应为 Base64 字符串,
name: string // 文件名，可选项，默认读取文件真实名称
user: { id: '', name: '', portraitUri: '', extra: '' }, // 消息中携带的用户信息，可选参
extra: '', // 消息中携带的透传信息，可选参
},
{
onProgress (progress) {}, // 上传进度监听，可选
onComplete (fileInfo) { // 上传完成时的回调钩子，可选
console.log(fileInfo.url) // 文件存储地址
// 如果需要构建自定义消息，return new ABCMesssage('')
// ABCMesssage 定义通过自定义消息实现 `const ABCMesssage = RongIMLib.registerMessageType(...)`
// 无 return 返回值的情况下，SDK 默认发送 SightMessage
}
},
{
// ... 其他配置项，可选
}
).then(({ code, data: message }) => {
if (code === 0) {
// 发送成功
}
})
```

接收消息

超级群会话中通过消息监听器通知业务层。调用 addEventListener 设置消息接收监听器。所有接收到的消息都会在此接口方法中回调。可在任意位置多次监听。

```
const Events = RongIMLib.Events
const listener = (evt) => {
console.log(evt.messages)
};
RongIMLib.addEventListener(Events.MESSAGES, listener)
```

上方示例的 listener 中返回 IMessageEvent 对象，其 messages 属性中包含接收到的消息（IAReceivedMessage）的列表。

获取历史消息

更新时间:2024-08-30

从服务端获取历史消息。

> 提示
>
> Web 没有本地消息存储，消息从融云消息服务器拉取，超级群历史消息默认存储 7 天。如遇到问题，请检查控制台超级群服务页面设置的存储天数。

获取超级群会话历史消息

通过 getHistoryMessages，开发者可以拉取指定某个会话的历史消息记录。

参数说明

参数	类型	说明
conversation	IConversationOption	获取消息所指定的会话。参见下方 `IConversationOption` 参数说明。
options	GetHistoryMessageOption	可选项。参见下方 `GetHistoryMessageOption` 参数说明。

- `IConversationOption` 参数说明

参数	类型	必填	说明
targetId	String	是	接收方的 `userId`
conversationType	Number	是	`RongIMLib.ConversationType.ULTRA_GROUP`。会话类型，参考 ConversationType。
channelId	String	否	接收方的频道 Id

- `GetHistoryMessageOption` 参数说明

参数	类型	说明
timestamp	number	（可选项）获取此时间之前的消息，0 为从当前时间拉取
count	number	（可选项）获取消息的数量。如果 SDK < 5.7.4，范围为 [1-20]；如果 SDK ≧ 5.4.1，范围为 [1-100]。默认值 `20`。
order	number	（可选项）获取消息的排列顺序，值为 0 或 1，0 为升序，1 为降序

代码示例

```
const conversation = {
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '<目标用户Id>',
channelId: '',
}

RongIMLib.getHistoryMessages(conversation).then((res) => {
if (res.code === 0) {
console.log(res.data.list)
console.log(res.data.hasMore)
} else {
console.log(res.code, res.msg)
}
})
```

从服务端获取特定批量消息

> 提示
>
> 从 SDK 版本 5.7.0 开始，该接口的返回数据类型由 IReceivedMessage 变更为 IAReceivedMessage 。

通过 getUltraGroupMessageListByMessageUId 根据消息 ID 获取消息。

参数说明

参数	类型	必填	说明
option	IConversationOption	是	接收方的 `userId`。参见下方 `IConversationOption` 参数说明。
msgs	IMessageDesc[]	是	参见下方 `IMessageDesc` 参数说明。

- `IConversationOption` 参数说明

参数	类型	必填	说明
targetId	String	是	接收方的 `userId`
conversationType	Number	是	`RongIMLib.ConversationType.ULTRA_GROUP`。会话类型，参考 ConversationType 。
channelId	String	否	接收方的频道 Id

- `IMessageDesc` 参数说明

参数	类型	必填	说明
messageUId	String	是	消息 Uid
sendTime	Number	是	消息发送时间

代码示例

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP
const targetId = ' 会话 Id '
const channelId = '<频道ID>'

RongIMLib.getUltraGroupMessageListByMessageUId({ conversationType, targetId, channelId }, msgs)
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

删除消息

更新时间:2024-08-30

超级群会话消息存储在服务端（免费存储 7 天）。App 用户通过客户端 SDK 删除自己的历史消息。

> 提示
>
> - 客户端的删除消息的操作均指从当前登录用户的历史消息记录中删除消息，不影响会话中其他用户的历史消息记录。
> - 如果 App 的管理员或者某普通用户希望在该 App 中彻底删除一条消息，例如在所有超级群成员的聊天记录中删除一条消息，应使用客户端或服务端的撤回消息功能。消息成功撤回后，原始消息内容会在所有用户的本地与服务端历史消息记录中删除。

按时间戳删除历史消息

clearHistoryMessages 支持按时间戳从服务端历史消息记录中删除指定单个频道的历史消息。单次操作仅针对单个超级群，不支持一次删除多个超级群中的消息。

```
const conversation = {
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: "<目标用户ID>",
channelId: "<频道ID>",
}
RongIMLib.clearHistoryMessages(conversation, timestamp).then(res => {
if (res.code === 0) {
console.log('清除成功')
} else {
console.log(res.code, res.msg)
}
}).catch(error => {
console.log(error)
})
```

参数	类型	说明
conversation	IConversationOption	会话
timestamp	number	时间点，该时间点前的消息将被删除

修改消息

更新时间:2024-08-30

用户在成功发送超级群消息后，可以主动修改已发送消息的消息内容。

修改本端用户已发消息的内容

使用 modifyMessage ，传入新的消息内容和待修改消息实例 IAReceivedMessage ，可修改当前用户已发送消息的内容。消息被修改后，IAReceivedMessage 对象的 isModifyMessage 属性会被更新为 true。注意：消息类型无法修改。如果改前为文本消息，则传入的新消息内容必须为 ITextMessageBody 类型。无法修改他人发送的消息。

```
RongIMLib.modifyMessage(content, message);
```

参数说明

参数	类型	说明
content	Object	消息内容，大小不能超过 128k
message	IAReceivedMessage	通过接收在线消息或拉取历史消息从 IMLib 取得的消息实例

代码示例

```
RongIMLib.modifyMessage({ key: 'value' }, {
messageUId: 'BS4O-QEBR-VJM6-9GPP',
sentTime: 1632728573423,
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '<超级群 ID>',
channelId: '<频道 ID>',
})
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

监听消息修改通知

可通过监听 Events.ULTRA_GROUP_MESSAGE_MODIFIED 事件来捕获修改信息的通知。

```
RongIMLib.addEventListener(RongIMLib.Events.ULTRA_GROUP_MESSAGE_MODIFIED, (messageList) => {
console.log(messageList)
})
```

撤回消息

更新时间:2024-08-30

撤回已发送成功的消息。

默认情况下，融云对撤回消息的操作者不作限制。如需限制，可考虑以下方案：

- App 客户端自行限制撤回消息的操作者。例如，不允许 App 业务中的普通用户撤回他人发送的消息，允许 App 业务中的管理员角色撤回他人发送的消息。
- 如需避免用户撤回非本人发送的消息，可以提交工单 申请打开**IMLib SDK 只允许撤回自己发送的消息**。从融云服务端进行限制，禁止用户撤回非本人发送的消息。

撤回指定消息

recallMessage 撤回指定消息。

调用示例

```
RongIMLib.recallMessage(conversation, options);
```

参数说明

参数	类型	说明
conversation	IConversation	会话
options	消息相关	要撤回消息的相关信息。请参见下方对 `options` 的参数说明。

- `options` 的参数说明

参数	类型	说明
messageUId	string	消息的 UId
sentTime	number	消息的发送时间
user	IUserProfile	撤回消息携带用户信息（可选项）
disableNotification	boolean	是否发送静默消息（可选项）
pushConfig	IPushConfig	移动端推送配置（可选项，与 Android、iOS 端的 MessagePushConfig 作用相似）
extra	string	撤回消息携带扩展信息。Web SDK 从 5.3.0 版本开始支持该参数。
isDelete	boolean	指定移动端接收方是否需要从本地删除原始消息记录。为 `false` 时，移动端不会删除原始消息记录，会将消息内容替换为撤回提示（小灰条通知）。为 `true` 时，移动端会删除原始消息记录，不显示撤回提示（小灰条通知）。Web SDK 从 5.3.1 版本开始支持该参数。

代码示例

```
const conversation = {
conversationType: RongIMLib.ConversationType.ULTRA_GROUP,
targetId: '<目标用户ID>',
channelId: '<频道ID>',
}
RongIMLib.recallMessage(conversation, {
messageUId: 'BS4O-QEBR-VJM6-9GPP',
sentTime: 1632728573423,
})
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

监听消息撤回通知

可通过监听 Events.ULTRA_GROUP_MESSAGE_RECALLED 事件来捕获撤回信息的通知。

```
RongIMLib.addEventListener(RongIMLib.Events.ULTRA_GROUP_MESSAGE_RECALLED, (messageList) => {
console.log(messageList)
})
```

扩展消息

更新时间:2024-08-30

已发送的消息可增加、修改、删除扩展信息。

适用场景：

原始消息增加状态标识的需求，都可使用消息扩展。

- 消息评论需求，可通过设置原始消息扩展信息的方式添加评论信息。
- 礼物领取、订单状态变化需求，通过此功能改变消息显示状态。 例如：向用户发送礼物，默认为未领取状态，用户点击后可设置消息扩展为已领取状态。

> 提示
>
> - 每次设置消息扩展将会产生内置通知消息，频繁设置扩展会产生大量消息。
> - 仅当发送消息时指定 `canIncludeExpansion` 值为 `true`，才可对消息进行扩展。

设置、更新消息扩展信息

updateExpansionForUltraGroupMessage

参数说明

参数	类型	说明
expansion	Object	要更新的消息扩展信息键值对，类型是 Object。 • Key 支持大小写英文字母、数字、 特殊字符`+ = - _` 的组合方式，不支持汉字。最大 32 个字符。 • （SDK < 5.3.0）Value 最大 64 个字符 • （SDK ≧ 5.3.0）Value 最大 4096 个字符
message	IAReceivedMessage	通过接收在线消息或拉取历史消息从 IMLib 取得的消息实例

代码示例

```
RongIMLib.updateExpansionForUltraGroupMessage(
{ key1: 'val1', key2: 'val2' },
message
).then((res) => {
if (res.code === 0) {
console.log(res.code, '更新成功')
} else {
console.log(res.code, res.msg)
}
})
```

删除消息扩展信息

removeExpansionForUltraGroupMessage

参数说明

参数	类型	说明
keyArray	Array	消息扩展信息中待删除的 key 的列表，类型是 Array。
message	IAReceivedMessage	通过接收在线消息或拉取历史消息从 IMLib 取得的消息实例

代码示例

```
RongIMLib.removeExpansionForUltraGroupMessage(['key1', 'key2'], message).then(
(res) => {
if (res.code === 0) {
console.log(res.code, '删除成功')
} else {
console.log(res.code, res.msg)
}
}
)
```

监听消息扩展通知

可通过监听 Events.ULTRA_GROUP_MESSAGE_EXPANSION_UPDATED 事件来捕获消息的扩展信息变更通知。

```
RongIMLib.addEventListener(RongIMLib.Events.ULTRA_GROUP_MESSAGE_EXPANSION_UPDATED, (messageList) => {
console.log(messageList)
})
```

多端同步已读状态

更新时间:2024-08-30

超级群业务可在多个客户端之间同步消息阅读状态。

同步消息已读状态

客户端主动调用 clearMessagesUnreadStatus 清除未读数时，SDK 会同时清除本地与服务端记录的消息的未读状态，同时服务端会将最新状态同步给同一用户账号的其他客户端。

- 如果指定了频道 ID（`channelId`），则标记该频道所有消息为全部已读，并同步其他客户端。
- 如果频道 ID 为空，则标记该超级群会话下所有不属于任何频道的消息为全部已读，并同步其他客户端。

> 提示
>
> - 超级群暂不支持按时间戳同步已读状态。调用 `clearMessagesUnreadStatus` 会按指定参数的要求标记全部消息为已读。
> - 超级群的多端未读数同步由融云服务端维护，Web 客户端不需要发送 `RC:SRSMsg` 消息来进行多端未读数同步。
> - 在多端同步消息已读状态时，可通过 `Events.CONVERSATION` 监听消息已读状态。参见快速上手中的「设置监听」。

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP
const targetId = '超级群 ID'
const channelId = '超级群频道 ID'

RongIMLib.clearMessagesUnreadStatus({
conversationType,
targetId,
channelId,
}).then((res) => {
if (res.code === 0) {
console.log(res.code)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	超级群 ID
conversationType	Number	是	`RongIMLib.ConversationType.ULTRA_GROUP`。会话类型，参考 ConversationType 。
channelId	String	否	超级群频道 ID

获取未读消息数

更新时间:2024-08-30

超级群业务支持从客户端 SDK 获取未读消息数，具体如下：

- 当前用户加入的所有超级群、指定超级群、或指定频道未读消息数。
- 当前用户加入的所有超级群、指定超级群、或指定频道的未读 @ 消息数。

返回的未读数最大值为 999。如果实际未读数超过 999，接口仍返回 999。

获取所有会话的未读消息数

通过 getAllUltraGroupUnreadCount 获取当前用户加入的所有超级群会话的未读消息数的总和。

代码示例

```
RongIMLib.getAllUltraGroupUnreadCount().then((res) => {
if (res.code === 0) {
console.log(res.data)
} else {
console.log(res.code, res.msg)
}
})
```

获取所有会话的未读 @ 消息数

通过 getAllUltraGroupUnreadMentionedCount 获取当前用户加入的所有超级群会话中的未读 @ 消息数的总和。

代码示例

```
RongIMLib.getAllUltraGroupUnreadMentionedCount().then((res) => {
if (res.code === 0) {
console.log(res.data)
} else {
console.log(res.code, res.msg)
}
})
```

获取指定会话的未读消息数

通过 getUltraGroupUnreadCountByTargetId 获取当前用户在指定超级群会话中的未读消息数。

参数	类型	必填	说明	支持版本
targetId	String	是	超级群 ID	

参数	类型	必填	说明	支持版本
levels	NotificationLevel[]	否	会话的免打扰级别设置。SDK 会根据该参数查找匹配的会话，并计算未读消息数。参考免打扰级别 。传空数组则统计全部免打扰级别会话中的未读数。	5.5.1

代码示例

```
const targetId = '超级群 ID'

RongIMLib.getUltraGroupUnreadCountByTargetId(targetId).then((res) => {
if (res.code === 0) {
console.log(res.data)
} else {
console.log(res.code, res.msg)
}
})
```

获取指定会话的未读 @ 消息数

通过 getUltraGroupUnreadMentionedCountByTargetId 获取当前用户在指定超级群会话中的未读 @ 消息数。

参数	类型	必填	说明	支持版本
targetId	String	是	超级群 ID	
levels	NotificationLevel[]	否	会话的免打扰级别设置。SDK 会根据该参数查找匹配的会话，并计算未读 @ 消息数。参考免打扰级别 。传空数组则统计全部免打扰级别会话中的全部未读 @ 消息数。	5.5.1

代码示例

```
const targetId = '超级群 ID'

RongIMLib.getUltraGroupUnreadMentionedCountByTargetId(targetId).then((res) => {
if (res.code === 0) {
console.log(res.data)
} else {
console.log(res.code, res.msg)
}
})
```

获取指定频道的未读消息数

通过 getUnreadCount 获取当前用户在超级群会话指定的频道中的未读消息数。注意，获取指定频道的未读消息数要求必须传入超级群频道 ID（channelId）。

参数说明

参数	类型	必填	说明
targetId	String	是	超级群 ID
conversationType	Number	是	RongIMLib.ConversationType.ULTRA_GROUP。会话类型，参考 ConversationType 。
channelId	String	否	超级群频道 ID

代码示例

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP
const targetId = '超级群 ID'
const channelId = '超级群频道 ID'

RongIMLib.getUnreadCount({ conversationType, targetId, channelId })
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

获取指定频道 @ 未读数

通过 getUnreadMentionedCount 获取当前用户在指定超级群会话指定频道的未读 @ 消息数。注意，获取指定频道的未读 @ 消息数要求必须传入超级群频道 ID（channelId）

参数说明

参数	类型	必填	说明
targetId	String	是	超级群 ID
conversationType	Number	是	RongIMLib.ConversationType.ULTRA_GROUP。会话类型，参考 ConversationType 。
channelId	String	否	超级群频道 ID。

代码示例

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP
const targetId = '超级群 ID'
const channelId = '超级群频道 ID'

RongIMLib.getUnreadMentionedCount({ conversationType, targetId, channelId })
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

获取未读消息

更新时间:2024-08-30

本页面描述了在超级群业务如何实现获取全部未读 @ 消息、跳转到指定未读 @ 消息等功能。

> 提示
>
> 本页面功能依赖融云服务端保存的超级群会话「未读 @ 消息摘要」数据。「@所有人」消息的摘要数据固定存储 7 天（不支持调整）。其他类型 @ 消息的摘要数据与超级群历史消息存储时长一致。

获取指定会话的未读 @ 消息列表

> 提示
>
> 该方法从 SDK 5.5.2 版本开始支持。

SDK 支持从远端获取指定超级群频道中的未读 @ 消息的摘要数据。App 可使用返回的摘要数据，从远端取得未读的 @ 消息。

例如，App 希望获取仅展示未读 @ 消息的场景，步骤如下：

1. 使用 getUltraGroupUnreadMentionedMessages 从远端获取指定超级群频道中的未读 @ 消息的摘要数据。最多可获取 50 条（count 取值范围为 [1-50]）

```
const targetId = '超级群 ID'
const channelId = '超级群频道 ID'
const sentTime = 0
const count = 20

RongIMLib.getUltraGroupUnreadMentionedMessages({ targetId, channelId, sentTime, count }).then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
```

参数	类型	必填	说明
targetId	String	是	超级群 ID
channelId	String	否	超级群频道 ID
sentTime	Number	是	融云服务会获取大于这个时间戳的未读 @ 消息列表。首次获取传 0， 后续获取传获取到的未读 @ 消息列表中最后一条消息的时间戳。

参数	类型	必填	说明
count	Number	是	数据个数，最大 50 。

`getUltraGroupUnreadMentionedMessages` 会返回一个 Promise，可解析得到 IUltraUnreadMsg 的数组。`IUltraUnreadMsg` 即每条 @ 消息的摘要数据，但不是完整的消息。从 5.8.0 版本开始，`IUltraUnreadMsg` 中会返回消息类型标识属性 `messageType`，可用于筛选摘要数据。

2. 从上一步得到 IUltraUnreadMsg 数据中可获取每一条 @ 消息的 `messageUId` 与 `sentTime`。使用 getUltraGroupMessageListByMessageUId 方法，在第二个参数传入用于提取消息完整内容的摘要 IMessageDesc 列表，即可从远端批量提取 @ 消息的完整内容。

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP
const targetId = ' 会话 Id '
const channelId = '<频道ID>'

RongIMLib.getUltraGroupMessageListByMessageUId({ conversationType, targetId, channelId }, msgs)
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

获取指定会话的第一条未读消息的时间戳

> 提示
>
> 该方法从 SDK 5.5.2 版本开始支持。

通过 getUltraGroupFirstUnreadMessageTimestamp 获取当前用户在指定超级群会话或指定频道的第一条未读消息的时间戳。

参数说明

参数	类型	必填	说明
targetId	String	是	超级群 ID
channelId	String	否	超级群频道 ID

代码示例

```
const targetId = '超级群 ID'
const channelId = '超级群频道 ID'

RongIMLib.getUltraGroupFirstUnreadMessageTimestamp({ targetId, channelId }).then((res) => {
if (res.code === 0) {
console.log(res.code, res.data) // res.data: { sentTime: <timestamp> } | null
} else {
console.log(res.code, res.msg)
}
})
```

输入状态 发送输入状态

更新时间:2024-08-30

sendUltraGroupTypingStatus 通知服务端正在输入中。

参数说明

参数	类型	必填	说明
targetId	String	是	接收方的 userId
conversationType	Number	是	RongIMLib.ConversationType.ULTRA_GROUP。会话类型，参考 ConversationType 。
channelId	String	否	接收方的频道 Id

代码示例

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP
const targetId = ' 会话 Id '
const channelId = '<频道ID>'

RongIMLib.sendUltraGroupTypingStatus({ conversationType, targetId, channelId })
.then((res) => {
if (res.code === 0) {
console.log(res.code, res.data)
} else {
console.log(res.code, res.msg)
}
})
.catch((error) => {
console.log(error)
})
```

监听用户正在输入状态通知

可通过监听 Events.OPERATE_STATUS 事件来捕获用户正在输入状态通知

```
RongIMLib.addEventListener(RongIMLib.Events.OPERATE_STATUS,  (evt) => {
console.log(evt)
})
```

设置指定群/频道免打扰

更新时间:2024-08-30

本文描述如何为指定超级群会话（targetId）或指定频道（channelId）设置免打扰级别。

> 提示
>
> 即时通讯客户端 SDK 支持多维度、多级别的免打扰设置。
>
> - App 开发者可实现从 App Key、指定细分业务（仅超级群）、用户级别多个维度的免打扰功能配置。在融云服务端决定是否触发推送通知时，不同维度的优先级如下：用户级别设置 > 指定超级群频道的默认配置（仅超级群支持） > 指定超级群会话的默认配置（仅超级群支持） > **App Key** 级设置。
> - 用户级别设置下包含多个细分维度。在融云服务端决定是否触发推送通知时，如存在用户级别配置，不同细分维度的优先级如下：全局免打扰 > 按频道设置的免打扰 > 按会话设置的免打扰。详见会话管理类别下的「免打扰功能概述」。

支持的免打扰级别

免打扰级别（notificationLevel）提供了针对不同 @ 消息的免打扰控制。从 SDK 5.3.0 开始，免打扰配置支持以下级别：

notificationLevel 的枚举值	数值	说明
NotificationLevel.ALL_MESSAGE	-1	全部消息均接收通知，即关闭免打扰功能
NotificationLevel.NOT_SET	0	未设置（用户未设置时为此状态，为全部消息都通知，在此状态下，如设置了超级群默认状态以超级群的默认设置为准）
NotificationLevel.AT_MESSAGE_NOTIFICATION	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人
NotificationLevel.AT_USER_NOTIFICATION	2	仅针对 @ 指定用户消息进行通知，且仅通知被 @ 的指定的用户进行通知 如：@张三 则张三可以收到推送，@所有人 时不会收到推送
NotificationLevel.AT_GROUP_ALL_USER_NOTIFICATION	4	仅针对 @群全员进行通知，只接收 @所有人 的推送信息
NotificationLevel.NOT_MESSAGE_NOTIFICATION	5	不接收通知，即使为 @ 消息也不推送通知

早于 5.3.0 的 SDK 版本仅支持设置为免打扰状态（不接收推送通知）或提醒状态（接收推送通知）。

设置免打扰级别

SDK 支持超级群用户为指定超级群会话（targetId）设置免打扰级别。调用 setConversationNotificationLevel 可设置免打扰级别。

> 注意：超级群业务暂不支持针对单个超级群会话所有消息设置免打扰级别（“所有消息”指所有频道中的消息和不属于任何频道的消息）。当前接口仅设置单个会话中不属于任何频道的消息的免打扰状态级别。

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP;
const targetId = '超级群 ID';
const channelId = '超级群频道 ID';
const notificationLevel = RongIMLib.NotificationLevel.NOT_MESSAGE_NOTIFICATION

RongIMLib.setConversationNotificationLevel({
conversationType,
targetId,
channelId
}, notificationLevel).then(( {code} ) => {

})
```

参数	类型	必填	说明
targetId	String	是	超级群 ID
conversationType	Number	是	会话类型，参考 ConversationType。。请注意超级群会话类型：如在 2022.09.01 之前开通超级群业务，默认不支持为单个超级群会话所有消息设置免打扰级别（“所有消息”指所有频道中的消息和不属于任何频道的消息）。该接口仅设置指定超级群会话（targetId）中不属于任何频道的消息的免打扰状态级别。如需修改请提交工单。
notificationLevel	Number	是	免打扰级别。详见上文支持的免打扰级别 对 **notificationLevel** 的说明。
channelId	String	否	超级群的会话频道 ID。 • 如果传入频道 ID，则针对该指定频道设置消息免打扰级别。 • 注意：如果不指定频道 ID，则仅针对指定超级群会话（targetId）中不属于任何频道的消息设置免打扰状态级别。

获取免打扰级别

调用 getConversationNotificationLevel 获取免打扰级别

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP;
const targetId = '超级群 ID';
const channelId = '超级群频道 ID';

RongIMLib.getConversationNotificationLevel({
conversationType,
targetId,
channelId
}).then(({ code, data }) => {

})
```

参数	类型	必填	说明
targetId	String	是	超级群 ID
conversationType	Number	是	会话类型，参考 ConversationType。
channelId	String	否	超级群频道 ID

设置群/频道默认免打扰

更新时间:2024-08-30

超级群业务支持为指定的群，或群频道设置默认免打扰逻辑。默认免打扰逻辑对所有群成员生效，一般由超级群的管理员进行设置。

如果您希望从 App 服务端控制指定超级群，或指定群频道默认免打扰逻辑，可参考服务端 API 文档设置超级群/频道默认免打扰。

注意事项

- 在融云服务端判断是否需要推送超级群消息时，指定的超级群，或群频道的默认免打扰配置优先级均低于用户级别配置。如果存在任何用户级别的免打扰配置，则优先以用户级别免打扰配置为准进行判断。

> 提示
>
> 即时通讯业务免打扰功能的 用户级别设置 支持控制指定的单聊会话、群聊会话、超级群会话、超级群频道的免打扰级别，并可设置全局免打扰的时间段与级别。用户级别设置优先级如下：全局免打扰 > 按频道设置的免打扰 > 按会话设置的免打扰 。详见「会话管理」下的免打扰功能概述。

- 为指定的超级群设置的默认免打扰逻辑，自动适用于群下的所有频道。如果针对频道另行设置了默认免打扰逻辑，则以该频道的默认设置为准。

支持的免打扰级别

免打扰级别（`notificationLevel`）提供了针对不同 @ 消息的免打扰控制。从 SDK 5.3.0 开始，免打扰配置支持以下级别：

`notificationLevel` 的枚举值	数值	说明
NotificationLevel.ALL_MESSAGE	-1	全部消息均接收通知，即关闭免打扰功能
NotificationLevel.NOT_SET	0	未设置（用户未设置时为此状态，为全部消息都通知，在此状态下，如设置了超级群默认状态以超级群的默认设置为准）
NotificationLevel.AT_MESSAGE_NOTIFICATION	1	仅针对 @ 消息进行通知，包括 @指定用户 和 @所有人
NotificationLevel.AT_USER_NOTIFICATION	2	仅针对 @ 指定用户消息进行通知，且仅通知被 @ 的指定的用户进行通知 如：@张三 则张三可以收到推送，@所有人 时不会收到推送
NotificationLevel.AT_GROUP_ALL_USER_NOTIFICATION	4	仅针对 @群全员进行通知，只接收 @所有人 的推送信息
NotificationLevel.NOT_MESSAGE_NOTIFICATION	5	不接收通知，即使为 @ 消息也不推送通知

早于 5.3.0 的 SDK 版本仅支持设置为免打扰状态（不接收推送通知）或提醒状态（接收推送通知）。

设置指定超级群或频道的默认免打扰级别

调用 setUltraGroupDefaultNotificationLevel 设置指定超级群默认通知配置

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP;
const targetId = '超级群 ID';
const channelId = '超级群频道 ID';
const notificationLevel = RongIMLib.NotificationLevel.NOT_MESSAGE_NOTIFICATION

RongIMLib.setUltraGroupDefaultNotificationLevel({
conversationType,
targetId,
channelId
}, notificationLevel).then(( {code} ) => {

})
```

参数	类型	必填	说明
targetId	String	是	超级群 ID
conversationType	Number	是	会话类型，参考 ConversationType 。
notificationLevel	Number	是	免打扰级别。详见上文支持的免打扰级别 对 **notificationLevel** 的说明。
channelId	String	否	超级群频道 ID

查询指定超级群或频道的默认免打扰级别

调用 getUltraGroupDefaultNotificationLevel 查询指定超级群默认通知配置

```
const conversationType = RongIMLib.ConversationType.ULTRA_GROUP;
const targetId = '超级群 ID';
const channelId = '超级群频道 ID';

RongIMLib.getUltraGroupDefaultNotificationLevel({
conversationType,
targetId,
channelId
}).then(({ code, data }) => {

})
```

参数	类型	必填	说明
targetId	String	是	超级群 ID
conversationType	Number	是	会话类型，参考 ConversationType 。
channelId	String	否	超级群频道 ID

更新日志

v5.10.2

更新时间:2024-08-30

发布日期：2024/07/02

功能优化：

1. 修改置顶空会话同步功能默认为 false。

v5.10.1

发布日期：2024/06/28

新增功能：

1. 新增用户信息托管功能
2. Electron 平台支持关闭置顶空会话同步

问题修复：

1. 修复 searchMessages 接口返回 count 可能为 undefined 的问题
2. 修复非群聊会话中会可能会携带 mentionedInfo 的问题
3. 修复发送 RC:SRSMsg 时，Content 中的 lastMessageSendTime 字段传错的问题

v5.9.9

发布日期：2024/06/05

新增功能：

1. Electron 平台新增了 `批量获取会话信息` 方法。

问题修复：

1. 修复了实时日志请求 URL 有特殊字符导致请求失败的问题。
2. 修复了不能给系统会话发送 RC:SRSMsg 消息的问题。
3. 修复了日志数据库升级可能会报错的问题。
4. 修复了 Electron 本地插入 RC:RcNtf 消息的发送状态异常的问题。
5. 修复了在 Electron 平台，RTC 信令发送和解析失败的问题。
6. 修复了连接时连续收到多个 30021 导致子进程崩溃的问题。
7. 修复了批量插入消息 LastMessage 更新问题。
8. 修复了应用退出，子进程会重启的问题。

v5.9.8

发布日期：2024/04/29

新增功能：

1. 新增了在线状态订阅功能。
2. 新增了撤回消息支持发送定向通知的能力。
3. Electron 平台新增了 `获取本地指定时间戳前后 N 条消息` 方法。
4. Electron 平台`搜索本地指定会话历史消息接口`支持传入消息类型。

问题修复：

1. 修复了 Web 端拉取消息后处理异常时导致不再拉取消息的问题。
2. 修复了导出 CombineV2Message 错误。

v5.9.7

发布日期：2024/04/01

问题修复：

1. 修复了使用 NPM 下载 IMLib-next 包时报错的问题。

v5.9.6

发布日期：2024/03/29

新增功能：

1. 定向消息中增加了指定接收消息的目标用户列表。
2. 引用消息中增加了被引用的唯一标识 `referMsgUid`。
3. 新增了 `electronExtension.getContinuousMessages` 接口，用以同时查询本地与远端历史消息。

问题修复：

1. 修复了重连报 30021 时没有重连的问题。
2. 修复了主动撤回消息后，在消息监听中收到重复的撤回消息通知的问题。

优化功能：

1. 适配 Electron 平台开启上下文隔离场景。

v5.9.5

发布日期：2024/01/31

优化功能：

1. 消息数据中的 `receivedStatusInfo` 改为可选，解决 TS 开发中可能存在的报错。

v5.9.4

发布日期：2024/01/31

新增功能：

1. 超级群支持发送定向消息功能
2. 超级群支持按消息 Uid 批量删除消息
3. 空会话置顶支持多端同步
4. 获取会话列表功能增加参数是否按置顶状态排序
5. Electron 平台新增获取全部 @ 消息未读数接口
6. Electron 平台新增发送状态消息功能

问题修复：

1. 修复发送图片消息时，缩略图质量参数 quality 无效的问题
2. 修复 Electron 平台发送状态消息异常问题

优化功能：

1. 优化群消息已读回执功能

v5.9.3

发布日期：2023/12/18

新增功能：

1. Electron 平台新增 setMessageReceivedStatusInfo 方法

优化功能：

1. 单聊已读回执回调参数中增加 sendUserId 用于判断是自己还是对方发的
2. 优化多设备登录收取离线消息时消息回执监听触发早于消息监听

问题修复：

1. 修复 removeChatRoomEntries 接口参数 ts 类型定义错误
2. 修复会话状态变更通知里的时间不对的问题

v5.9.2

发布日期：2023/12/13

问题修复：

1. 修复 Electron 在 Windows 平台发送消息接口响应延迟高的问题
2. 修复 Electron 在 Windows 平台拉大量离线消息时应用卡顿的问题
3. 接口 `addConversationsToTag` 增加限制，会话列表不能为空

v5.9.1

发布日期：2023/11/28

问题修复：

1. 修复 Node 14 以下版本无法下载 @rongcloud/electron 依赖包的问题

v5.9.0

发布日期：2023/11/23

新增功能：

1. 消息拓展变更通知（Events.EXPANSION）回调数据 `IExpansionListenerData` 中补充会话信息。从 5.9.0 开始，IUpdatedExpansion 与 IDeletedExpansion 均返回 `conversationType` 和 `targetId`。
2. 聊天室成员加入、退出通知 (Events.CHATROOM) 回调 IChatroomUserChangeInfo 中增加当前聊天室人数 `memberCount`。
3. 新增 `getAllUnreadMentionedCount` 接口，支持获取所有会话类型的未读 @ 消息数。
4. 获取会话列表接口返回的会话数据 `IAReceivedConversation` 中增加草稿字段 draft。
5. 新增 `getFirstUnreadMessageInfo` 接口，支持获取第一条未读消息信息。
6. 支持荣耀推送配置，可在发送消息时通过 `IPushConfig` 下的 IAndroidPushConfig 控制荣耀推送的消息提醒级别与通知栏图片。

问题修复：

1. 小程序平台不再请求动态导航地址
2. 修复可能收不到敏感词拦截通知的问题
3. 修复发送@消息时，会话中的@字段错误的问题
4. 修复高频调用 tag 和会话状态相关接口时报 26002 的问题
5. Electron 平台修复发送撤回消息后,再次拉到撤回消息时原始消息被修改两次的问题
6. 修复 Electron 的 Windows 平台退出时卡死的问题
7. 优化消息量大时，在 Windows 平台会导致应用卡顿问题

v5.8.5

发布日期：2023/10/27

优化功能：

1. 优化 SDK 日志上传机制

问题修复：

1. 修复发送 @ 消息时，会话中的 @ 字段错误的问题

v5.8.4

发布日期：2023/09/25

新增功能：

1. Electron 平台支持免打扰级别功能
2. 会话列表中增加 operationTime 字段（仅 Electron 平台支持）

问题修复：

1. 修复加入聊天室后，刷新页面并连接后会自动重新加入的问题
2. 修复插入本地的撤回消息已读状态错误的问题（Electron 平台）
3. 修复并发上传语音消息时报错的问题

v5.8.3

发布日期：2023/08/31

新增功能：

1. 多端同步会话未读数功能支持系统会话。
2. 发送文件消息（上传）支持携带 @ 信息。
3. 加入聊天室接口返回房间信息与用户状态信息，例如是否禁言、是否在禁言白名单中、聊天室人数等。
4. 在 Electron 平台，如果消息被撤回时本地数据库已不存在该消息，仍然插入一条 RC:RCNtf 类型消息。

v5.8.2

发布日期：2023/07/28

新增功能：

1. Electron 平台支持 win32 X64 架构
2. Electron 平台增加处理多端同步消息 RC:ReadNtf
3. Electron 平台适配上传功能

问题修复：

1. 修复服务器重启后可能导致连接不成功的问题
2. 修复支付宝小程序连接报错问题

v5.8.1

发布日期：2023/07/14

新增功能：

1. 新增小灰条消息（RC:InfoNtf）、命令消息（RC:CmdMsg）、群组通知消息（RC:GrpNtf）的发送

v5.8.0

发布日期：2023/07/3

新增功能：

1. 超级群查询未读 @ 消息增加返回消息类型属性

问题修复：

1. 修复断网重连偶现导致触发心跳问题。

其他：

1. 小程序平台安全域名调整：安全域名

v5.7.10

发布日期：2023/06/15

新增功能：

1. 新增接口 `getUltraGroupUnreadInfoList`，支持批量获取超级群会话未读信息
2. Electron 平台新增接口`searchMessagesByUser`，支持根据发送者ID 搜索本地单群聊会话消息

问题修复：

1. 修复撤回超级群 @ 消息时，未读 @ 数减一异常的问题。
2. 修复拉取超级群消息可能陷入死循环的问题。

v5.7.9

发布日期：2023/05/29

新增功能：

1. 新增聊天室状态通知(封禁、禁言、加入/退出多端状态同步)
2. Electron 平台增加在收到 30019、30021 状态码时 SDK 会自动发起重连逻辑
3. 初始化增加区域码设置
4. 超级群获取频道列表接口增加 @ 我的未读消息数属性

问题修复：

1. 修复超级群收到撤回消息时，未读消息数异常问题
2. 修复支付宝小程序平台重新连接时会建立两个 socket 连接的问题

v5.7.8

发布日期：2023/05/11

新增功能：

1. IMLib 的 Electron平台增加对草稿的操作

问题修复：

1. uniapp 打包 app 链接不上
2. IE 浏览器不再支持日志存储，因为 indexDB 不支持 getAllKeys 方法

v5.7.7

发布日期：2023/04/21

问题修复：

1. 修复获取免打扰列表 notificationLevel 值 undefined。
2. 修复 Electron 平台获取全部会话列表无法获取系统会话的问题。

其他：

1. 增加处理多端同步 RC:ReadNtf 消息逻辑。
2. 消息推送属型配置 IPushConfig.androidConfig 新增 `categoryVivo` 字段。如果指定了 `categoryVivo` ，必须同时指定匹配的 `typeVivo`。该字段优先级高于控制台为 App Key 下的应用标识配置的 vivo 推送 Category。
   - `categoryVivo` 字段对应 vivo 推送服务的消息二级分类（`category` 字段）。详细取值请参见 vivo 推送消息分类说明。
   - `typeVivo` 字段对应 vivo 推送服务的消息分类（`classification` 字段，区分系统消息、运营消息）。请注意遵照 VIVO 官方要求，确保二级分类（`category`）取值属于 `classification` 下允许发送的内容。详细取值请参见 vivo 推送消息分类说明。

v5.7.5

发布日期：2023/04/12

问题修复：

1. 优化 5.4.7 之前版本禁用资源 pb 报错。

v5.7.4

发布日期：2023/03/30

问题修复：

1. 修复无法获取到未设置免打扰级别和免打扰状态的未读数的问题。
2. 修复在 web 平台，会收到自己设置的聊天室 kv 的通知的问题。
3. 修复偶现 `Cannot read property 'kvStorage' of null` 的问题。
4. 修复断网重连后再发消息时，偶发消息监听中收到自己发送的消息的问题。

其他：

1. 获取历史消息接口，count 字段范围改为 0-100，超出将报错。
2. 获取指定会话接口（`getConversations`）针对超级群会话类型，返回值增加 `firstUnreadMessage` 和 `channelType` 字段

v5.7.3

发布日期：2023/03/02

新增功能：

1. 增加超级群用户组通知监听

2. 消息推送属型配置 IPushConfig.androidConfig 支持华为推送参数
    1. categoryHW ： 华为推送消息分类
    2. importanceHW ： 华为推送消息级别
    3. imageUrlHW：华为通知类型的推送所使用的通知图片 url
3. 消息推送属型配置 IPushConfig.androidConfig 支持消息推送参数
    1. miLargeIconUrl：小米 Large icon 链接

问题修复：

1. 修复按会话免打扰级别获取未读数接口设置 levels 参数不生效问题
2. 修复发送 @ 消息后，发送方自己收到 @ 消息的会话变更问题
3. 修复获取免打扰的会话列表返回的 notificationlevel 字段值错误问题
4. 修复切换用户后，会话状态还使用的前一个用户的数据问题
5. 修复 ELectron 平台 CMP 连接失败后未重连的问题
6. 修复推送配置中单独设置 iOSConfig 或者 androidConfig 不生效的问题

v5.7.2

发布日期：2023/02/07

新增功能：

1. Electron 平台新增 `batchInsertMessage` 接口，支持批量插入消息到本地
2. Electron 平台新增 `getMessageCount` 接口，支持获取某个会话下所有消息数量

问题修复：

1. 修复 getConnectionStatus 接口返回状态类型错误问题
2. 修复获取免打扰会话列表接口 getBlockedConversationList 无法返回设置了免打扰级别的会话的问题。修复后，免打扰级别（level）大于 0 的会话都会返回。

优化：

1. Web 端本地会话状态缓存上限优化，最大支持存储 1000 条会话状态
2. Electron 平台导航缓存优化，使用本地文件存储代替数据库存储

v5.7.1

发布日期：2023/01/10

新增功能：

1. Electron 平台新增 `setCheckDuplicateMessage` 接口，支持在接收消息时禁用消息排重机制
2. Electron 平台 disconnect 接口增加 closeDB 参数

问题修复：

1. 修复在火狐浏览器中的 indexDB 兼容问题

2. 修复断网重连时调用 disconnect 无法断开连接的问题
3. 修复调用 `removeChatRoomEntry` 后，其他人收到的 KV 数据更新类型（ChatroomEntryType）为 `UPDATE` 的问题。修复后，KV 更新类型为 `DELETE`。
4. 修复 Electron 平台插入消息时设置的消息扩展字段 `canIncludeExpansion`，`expansion` 与返回数据中不一致的问题
5. 修复 Electron 平台发起 http 请求报错的问题

优化：

1. 补齐位置消息相关的消息类型注册
2. `IUserInfo` 中增加 `alias` 字段

v5.7.0

发布日期：2022/12/01

新增功能：

1. 获取未读会话列表 `getUnreadConversationList`，支持单聊、群聊、系统会话。

问题修复：

1. 修复超级群未读数计算将自己发送的超级群消息也计入的问题。
2. 修复 ELectron 平台获取会话列表中 hasMentioned 字段错误的问题
3. 修复 ELectron 平台获取消息中 isMentioned 字段错误的问题

优化：

1. 断网重连时，如果被聊天室封禁，则不再尝试加入该聊天室
2. 断网重连情况下，SDK 内部重新加入聊天室时拉取的历史消息数量为加入时传入的值，默认为 10

非兼容性变更：

1. 连接状态监听回调参数类型变更，Events.DISCONNECT 回调参数类型由 ConnectionStatus 变更为 ErrorCode, Events.SUSPEND 回调参数类型由 `ConnectionStatus | ErrorCode` 变更为 `ErrorCode`。

2. 部分接口类型变更

方法名称	变更前返回类型	变更后返回类型
getUltraGroupMessageListByMessageUId	IReceivedMessage	IAReceivedMessage
getConversationsFromTagByPage	IReceivedConversationByTag	IAReceivedConversationByTag

3. 部分监听事件返回值类型变更

事件名称	变更前返回类型	变更后返回类型	事件描述
ULTRA_GROUP_ENABLE	IReceivedConversation	IAReceivedConversation	超级群会话列表同步完成，可以调用超级群相关接口

事件名称	变更前返回类型	变更后返回类型	事件描述
ULTRA_GROUP_MESSAGE_EXPANSION_UPDATED	IReceivedMessage	IAReceivedMessage	超级群消息扩展更新通知
ULTRA_GROUP_MESSAGE_MODIFIED	IReceivedMessage	IAReceivedMessage	超级群消息被修改通知
ULTRA_GROUP_MESSAGE_RECALLED	IReceivedMessage	IAReceivedMessage	超级群消息被撤回通知

v5.6.1

发布日期：2022/11/18

优化：

1. 在 Electron 平台，收到撤回消息时 SDK 内部将被撤回消息更新为小灰条消息 `RC:RcNtf`
2. RongIMLib.electronExtension.insertMessage 接口支持 `BaseMessage` 类型参数
3. getHistoryMessagesByMessageTypes 返回对象中增加 list 字段，messages 字段标记废弃

其他：

1. 支持京东小程序平台

v5.6.0

发布日期：2022/11/04

新增功能：

1. 新增接口：获取置顶会话列表

问题修复：

1. 修复断网重连时如果 token 过期，应用层收不到状态通知的问题
2. 修复多端登录时 ELectron 端收到消息的 `offLineMessage` 为 true 的问题
3. 修复 Web 端多端登录情况下，本端未加入聊天室时，会收到其他端加入聊天室后发送的消息问题
4. 修复 Electron 平台插入本地消息时（`insertMessage`），因传入的 message 中指定了服务端消息 ID（`messageUId`），导致消息可能重复的问题
5. 修复 Electron 平台引用消息和图文消息无法被搜索的问题
6. 修复 Electron 平台下发送消息无法携带 "@ 信息" 的问题
7. 修复 Web 平台收到位置共享功能的 `RC:RLQuit`、`RC:RLJoin` 消息时，在控制台报错的问题

非兼容性变更：

1. 在 ELectron 平台，主进程 `@rongcloud/electron` 初始化时，强制要求传参 appkey，否则初始化失败，详见 主进程初始化
2. 修改 ELectron 平台扩展 .node 包的下载方式，详见：安装 .node 文件
3. Web 端不再支持 Comet 连接模式，仅支持 Websocket 连接

v5.5.5

发布日期：2022/10/21

问题修复：

1. fix: 修复导航数据变更通知向前兼容报错的问题，该问题可能导致部分使用 RTCLib 或 CallLib 的客户无法正常使用会话标签功能。

v5.5.4

发布日期：2022/09/23

问题修复：

1. 修复解码 RTC 信令时 int64 型数据解码错误，进而可能导致 RTC 房间内的 KV 数据异常。

v5.5.3

发布日期：2022/09/22

问题修复：

1. 修复网络异常时可能无 `Event.SUSPEND` 事件通知的问题
2. 修复弱网情况下，可能无法收到超级群初始化完成通知 `Event.ULTRA_GROUP_ENABLE` 的问题
3. 修正 `Event.MESSAGES`、`Event.CONVERSATION` 等事件的回调参数接口类型声明错误问题

v5.5.2

发布日期：2022/09/09

新增功能：

1. 新增接口 `getUltraGroupUnreadMentionedMessages`，支持获取超级群未读 @ 消息列表
2. 新增接口 `getUltraGroupFirstUnreadMessageTimestamp`，支持获取超级群第一条未读消息时间戳

问题修复：

1. 修复升级到 5.5.0 版本时，Electron 中数据库会话列表丢失的问题

v5.5.1

发布日期：2022/09/01

新增功能：

1. 新增接口 `getTotalUnreadCountByLevels`，支持按照会话的免打扰级别，获取对应会话的全部未读消息数
2. 新增接口 `getTotalUnreadMentionedCountByLevels`，支持按照会话的免打扰级别，获取对应会话的全部未读 @ 消息数
3. `getUltraGroupUnreadCountByTargetId` 支持按照会话的免打扰级别，获取对应会话的全部未读消息数
4. `getUltraGroupUnreadMentionedCountByTargetId`，支持按照会话的免打扰级别，获取对应会话的全部未读 @ 消息数
5. 发送、接收消息增加 `messageId` 字段

问题修复：

1. 修复同时频繁修改超级群 KV 和 发送消息，可能导致接收到自己发送的消息的问题
2. 修复在 Electron 中发送自定义消息时，content 中数字大于 32 位可能导致崩溃的问题

v5.5.0

发布日期：2022/08/25

优化：

1. 架构优化，降低后续 plugin-rtc 升级对 IMLib 的版本依赖

v5.4.5

发布日期：2022/08/18

新增功能：

1. 超级群消息修改和消息扩展中含敏感词时，在敏感词回调中通知

问题修复：

1. 修复多端设置会话状态时会收到重复通知的问题
2. 修复小程序平台 HTTP 请求的 header 字段错误的问题
3. 修复加入多个聊天室时，后加入的聊天室 KV 拉取异常的问题

其他：

1. 支持微信小程序插件平台
2. 支付宝小程序平台支持 Websocket 连接

v5.4.4

发布日期：2022/08/16

问题修复：

1. 修复超级群消息扩展变更通知重复的问题。
2. 修复超级群消息扩展变更的 channelId 为空时无通知的问题

v5.4.3

发布日期：2022/08/04

问题修复：

1. 修复 `getUltraGroupMessageListByMessageUId` 接口和超级群消息变更通知中参数 `senderUserId` 无值的问题。

v5.4.2

发布日期：2022/07/21

新增功能：

1. 新增超级群私有频道功能
2. getUltraGroupList 接口增加 channelType 频道类型参数
3. 新增 超级群类型（私有和公有）变更通知 RongIMLib.Events.ULTRA_GROUP_CHANNEL_TYPE_CHANGE
4. 新增 私有频道白名单用户被移出通知 RongIMLib.Events.ULTRA_GROUP_CHANNEL_USER_KICKED
5. 新增 删除频道通知 RongIMLib.Events.ULTRA_GROUP_CHANNEL_DLELTE

问题修复：

1. 修复内部 logger 引用错误问题

v5.4.1

发布日期：2022/07/02

问题修复：

1. 修复被引用后编译时可能产生错误的问题。

v5.4.0

发布日期：2022/07/01

新增功能：

1. 获取会话列表增加 `notificationLevel` 字段
2. init 方法增加 uploadDomain 参数，支持修改文件上传地址为指定的服务器（仅适用于私有云，暂仅支持七牛地址）
3. 新增 Electron 平台接口：
   - electronExtension.getAllConversationList
   - electronExtension.getConversationList
   - electronExtension.searchConversationByContent
   - electronExtension.searchMessage
   - electronExtension.searchMessageInTimeRange
   - electronExtension.getHistoryMessagesByMessageTypes
   - electronExtension.setMessageStatusToRead
   - electronExtension.setMessageReceivedStatus
   - electronExtension.setMessageSentStatus
   - electronExtension.deleteMessages
   - electronExtension.clearMessages
   - electronExtension.deleteMessagesByTimestamp
   - electronExtension.insertMessage
4. 新增黑名单相关接口

- addToBlacklist
- removeFromBlacklist
- getBlacklist
- getBlacklistStatusForUser

v5.3.4

发布日期：2022/06/20

新增功能：

1. 批量设置聊天室 KV 接口增加 `isForce` 字段。

问题修复：

1. 修复频繁设置会话置顶或会话免打扰状态导致 26002 错误的问题。

v5.3.3

发布日期：2022/06/02

新增功能：

1. 敏感词拦截事件 `Events.MESSAGE_BLOCKED` 新增 extra 字段。

问题修复：

1. 修复撤回超级群会话最后一条消息时，超级群会话中 `latestMessage` 字段未更新的问题。
2. 修复可能会丢失会话类型为 ConversationType.RTC_ROOM 的直发消息的问题。
3. 修复获取会话列表为空时，返回报错的问题。
4. 修复收到超级群会话第一条消息时计数错误的问题。

优化：

1. 优化撤回消息计数。

v5.3.2

发布日期：2022/05/20

优化：

1. 优化聊天室获取消息及扩展属性信息机制。

v5.3.1

发布日期：2022/05/19

新增功能：

1. 撤回消息接口增加 `isDelete` 参数，可控制移动端接收方是否展示撤回消息的提示小灰条。

问题修复：

1. 优化重连逻辑，修复网络异常时可能无法重连的问题
2. 修复 comet 连接时拉取消息报错的问题
3. 修复免打扰级别（`NotificationLevel`）枚举值拼写错误，由 `AT_GROUP_ALL_USER_NOFICATION` 改为 `AT_GROUP_ALL_USER_NOTIFICATION`。
4. 修复 App Key 未开启超级群服务时，SDK 断网重连后会异常拉取超级群消息的问题

其他：

1. 优化连接逻辑

v5.3.0

发布日期：2022/04/29

新增功能：

1. 发送撤回消息的消息体中可携带 extra 字段
2. 发送上传文件消息的接口增加 contentDisposition 参数
3. 新增设置指定超级群默认通知配置接口 setUltraGroupDefaultNotificationLevel
4. 新增查询指定超级群默认通知配置接口 getUltraGroupDefaultNotificationLevel
5. 新增获取指定会话@消息未读数接口 getUnreadMentionedCount
6. 新增获取指定超级群所有频道未读总数接口 getUltraGroupUnreadCountByTargetId
7. 新增获取指定超级群所有频道 @ 消息未读总数接口 getUltraGroupUnreadMentionedCountByTargetId
8. 新增获取所有超级群未读总数接口 getAllUltraGroupUnreadCount
9. 新增获取所有超级群类型 @ 消息未读总数接口 getAllUltraGroupUnreadMentionedCount
10. 新增设置免打扰级别接口 setConversationNotificationLevel
11. 新增获取免打扰级别接口 getConversationNotificationLevel

问题修复：

1. 修复收到广播消息后，断开连接再重复连接，会再次收到广播消息的问题
2. 修复断开连接再重新连接后无法获取最新超级群会话列表的问题
3. 修复在 IE 11 浏览器中调用 disconnect 方法报错的问题

其他：

1. 超级群消息扩展字段 value 长度限制改为 4096

v5.2.4

发布日期：2022/04/15

问题修复：

1. 修复极少数情况下会丢失会话类型为 `ConversationType.RTC_ROOM` 的消息的问题

v5.2.3

发布日期：2022/04/15

问题修复：

1. 修复 https 协议时无法上报日志的问题

v5.2.2

发布日期：2022/04/07

问题修复：

1. 修复接受广播消息可能重复的问题。
2. 修复获取会话列表为空时，返回报错的问题。
3. 修复收到超级群会话第一条消息时计数错误的问题。

v5.2.1

发布日期：2022/03/17

问题修复：

1. 修复收到 RongIMLib.Events.ULTRA_GROUP_ENABLE 监听事件时立即调用超级群相关接口报错的问题
2. 修复在小程序中重连失败时无法继续重连的问题
3. 修复无法收到超级群撤回监听事件 RongIMLib.Events.ULTRA_GROUP_MESSAGE_RECALLED的问题
4. 修复超级群中消息未读数计算不准确的问题

新增功能：

1. 新增聊天室和 rtc 房间绑定接口 bindRTCRoomForChatroom
2. 超级群历史消息中新增消息是否已被修改字段 isModifyMessage
3. 监听事件 RongIMLib.Events.ULTRA_GROUP_MESSAGE_RECALLED 中增加发送者 userId 字段 senderUserId

v5.2.0

发布日期：2022/03/02

新增功能：

1. 新增了融云超级群会话，支持无成员上限的群组聊天
2. 新增了超级群频道功能，可在超级群会话下创建多个频道，成员可随意在不同群频道中发送消息，但不同频道间的消息相互隔离

v5.1.2

发布日期：2022/02/17

问题修复：

1. 修复环境中 console 无法使用时导致 SDK 无法使用的问题
   新增功能：
2. 监听事件 DISCONNECT，SUSPEND 的回调中返回状态码
3. 注册自定义消息时新增 'isStatusMessage' 字段

v5.1.1

发布日期： 2022/01/17

新增功能：

1. 为群聊已读回执功能增加 sendReadReceiptResponseMessageV2 方法，废弃 sendReadReceiptResponseMessage 方法。
2. 增加获取免打扰会话列表接口 getBlockedConversationList

v5.1.0

发布日期：2022/01/07

问题修复：

1. 修复在单聊中发送 @ 消息时，接收方收到该消息时报错的问题
   新增功能：
2. IMLib 提供对 RTC 跨房间连麦功能的底层支持

v5.0.2

发布日期：2021/12/30

问题修复：

1. 修复会话列表中 latestMessage 为 null 时（一般在会话中最新消息在 Web 客户端本地被删除时出现）报错的问题
2. 修复获取聊天室历史消息接口报错的问题
3. 修复群已读回执事件中 messageUId 错误的问题。
4. 修复 @ 消息中缺失 content.mentionedInfo.mentionedContent 字段（向客户端离线推送 @消息时可能会用到该字段）的问题
5. 修复调用 Server API 发送消息且设置了发送者接收该消息时，如果发送方在 Web 客户端在线，会重复收到两次消息的问题。
6. 修复重连失败时，未触发 SUSPEND 事件的问题
   新增功能：
7. 新增清除全部未读数接口
8. 新增 MESSAGE_BLOCKED 事件，通知敏感词回调。需要联系客服开启后才能使用，默认关闭状态
9. 新增对外 Typescript 接口：ISendImageMessageOptions, ISendFileMessageOptions

v5.0.1

发布日期：2021/12/09

问题修复：

1. 修复断线重连时可能收消息延迟的问题
2. 修复连接之前多个 ping 等待造成连接延迟的问题

3. 修复切换用户后，后登录用户使用前一用户的内存数据拉取消息的问题
4. 修复消息体内 user.portraitUri 字段多端不一致问题，推荐使用 portrait 字段
5. 修复单聊已读回执事件响应参数中 messageUId 错误的问题
   新增功能：
6. 新增获取未读 @ 消息数接口
7. 新增获取所有未读 @ 消息数接口

v5.0.0

发布日期：2021/11/16

1. 重新定义接口实现，所有接口平铺设计，易用性更强。
2. 新增定义发送媒体消息接口，将上传、发送过程整合，无需再集成单独的上传插件。
3. 连接状态通知优化，将可重连、不可重连的连接中断分别提醒，便于业务区分处理。
4. 事件监听支持多次添加，便于业务层根据不同的场景 View 监听所需事件通知。
5. 支持批量消息通知，便于业务层针对批量数据进行业务优化，提升渲染性能。

内容审核概述

更新时间:2024-08-30

即时通讯支持对 IM 内容进行审核。

- 即时通讯（IM）服务已内置敏感词机制。注意，敏感词机制仅是一种基础保护机制，且仅限于文本内容（默认最多 50 个敏感词），不可替代专业内容审核服务。
- 融云的内容审核服务产品中的 IM 审核服务，可为 IM 内容提供全面的保障与支持，支持审核文本、图片、语音片段、小视频，精准识别敏感信息。
- 如需自行实现审核或对接第三方审核服务，可以使用消息回调服务。

如果消息因被判定违规导致无法下发收件人，默认情况下消息发送者不会收到通知。如果 App 希望通知消息发送者消息已被拦截，可提交工单开通**含敏感词消息屏蔽状态回调发送端**，并在客户端设置监听（要求 Android/iOS SDK 版本 ≧ 5.1.4，Web ≧ 5.0.2）。详见敏感信息拦截回调。

敏感词机制

> 提示
>
> - 客户端不提供针对该功能的管理接口，仅提供回调接口，可在消息被判定为不下发时通知消息发送方。详见敏感信息拦截回调。
> - 如需审核文本（支持语义检测）、图片、语音片段、小视频，建议使用融云提供的内容审核服务产品。

敏感词机制是一种基础保护机制，仅支持对文本消息内容中的敏感词进行识别与过滤。对命中敏感词的消息，您可以选择进行屏蔽该消息（不会下发给接收方），或按指定规则替换消息中的敏感词后再进行下发。

目前支持的敏感词过滤语言包括：中文、英文、日语、德语、俄语、韩语、阿拉伯语。

您可以通过以下方式管理 App Key 下开发环境或生产环境的敏感词：

功能描述	客户端 API	融云服务端 API	控制台
添加敏感词，支持设置替换内容	不提供该 API	添加敏感词	敏感词设置页面
移除敏感词	不提供该 API	移除敏感词	敏感词设置页面
批量移除敏感词	不提供该 API	批量移除敏感词	敏感词设置页面
获取敏感词列表，支持获取设置的替换内容	不提供该 API	获取敏感词列表	敏感词设置页面

默认行为

- 默认最多设置 50 个敏感词。
- 默认仅针对从客户端 SDK 发送的消息生效。
- 默认仅支持识别官方内置的文本消息类型（消息标识为 `RC:TxtMsg`）中的敏感词。支持单聊、群聊、聊天室、超级群会话。超级群中文本消息修改后的内容默认也会敏感词识别、拦截或过滤。

调整配置

- **IM 旗舰版或 IM 尊享版**可以在控制台 IM 服务管理 页面的**扩展服务**标签下自行调整敏感词上限数。具体功能与费用以融云官方价格说明 页面及计费说明 文档为准。
- 如果您对使用服务端 API 发送的消息进行敏感词过滤，可以在控制台的免费基础功能界面 打开 **Server API 发送消息过滤敏感词**开关。
- 如果您需要对自定义消息类型启用敏感词机制，可以在敏感词设置 页面点击**设置自定义消息**。提供自定义消息的消息类型的 `ObjectName`，及该消息类型下内容（Content）JSON 结构中对应的键值 Key，即可对该 Key 所对应的 Value 值进行敏感词过滤处理。

IM 内容审核服务

> 提示
>
> **客户端不提供针对该功能的管理接口，仅提供回调接口，可在消息被判定为不下发时通知消息发送方。详见敏感信息拦截回调。**

如果您希望全面审核 IM 内容，可以使用融云的内容审核服务产品 ，该产品提供 **IM 审核服务**与**音视频审核服务**。

IM 审核针对即时通讯业务，具体可提供以下能力：

- 审核文本内容
- 审核图片
- 审核语音片段
- 审核小视频
- 审核自定义消息类型（需要提交工单申请）
- 审核超级群业务中的消息修改
- 从控制台查看审核报告
- 从控制台查询 IM 审核记录
- 审核结果回调

您可以在控制台的 IM & 音视频审核 页面开通 **IM 审核服务**，配置接收审核结果回调的地址。详见服务端文档审核结果回调 。

IM 内容审核计费

内容审核服务为付费服务，开发环境可免费体验，生产环境下需预存才能使用服务。具体计费说明详见资费标准 · IM 审核 。

消息回调服务

如果您希望对接自己的审核系统或其他第三方内容审核服务，可以使用消息回调服务 。

消息回调服务（**原模版路由**）提供一种消息过滤机制。您可以根据发送用户 ID、接收用户 ID、消息类型、会话类型等参数，将相应的消息同步到您指定的服务器。超级群业务中，修改消息内容、更新消息扩展也支持通过消息回调同步到您指定的服务器。

消息同步到您指定的服务器后，可以使用您自己的审核系统执行内容审核，也可以对接其他第三方审核系统。融云服务端会根据您应用服务器返回的响应结果，决定是否将消息下发、是否替换消息中的内容，以及如何进行内容替换。

您可以通过控制台的消息回调服务 页面管理 App Key 下开发环境或生产环境的消息回调服务状态和路由规则。

关于如何创建路由规则，以及回调参数的具体说明，请参见消息回调服务文档。

消息回调服务计费

费用以融云官方价格说明页面及计费说明文档为准。

敏感信息拦截回调

更新时间:2024-08-30

> 提示
>
> Web 端 SDK 从 5.0.2 版本开始支持敏感信息拦截回调。

如果您使用了融云的内容审核服务（包括消息敏感词、IM 审核服务、消息回调服务），可能希望在消息因触发审核规则而无法下发时通知消息发送方。从 SDK 5.0.2 版本开始，客户端 SDK 可以在消息被拦截时触发回调，通知消息发送方。

消息因包含敏感信息被拦截可分为以下情况：

- 文本消息内容命中了融云内置的消息敏感词，导致消息不下发给接收方。
- 文本消息内容命中了您自定义的消息敏感词，导致消息不下发给接收方。
- 消息命中了 **IM 审核服务，或消息回调服务设置的审核规则，导致消息不下发给接收方。**

开通服务

如有需求，请提交工单，申请开通含敏感词消息屏蔽状态回调发送端。

设置敏感信息拦截监听器

您可以通过下面的方法设置敏感词拦截监听器，监听到被拦截的消息以及拦截原因。

```
const listener = (blockedMessageInfo) => {
console.log('消息含有敏感信息被拦截：', blockedMessageInfo)
}

RongIMLib.addEventListener(RongIMLib.Events.MESSAGE_BLOCKED, listener)
```

移除监听方法及全部事件列表，详见 IMLib 文档 事件监听。

返回值说明

参数	类型	说明
blockedMessageInfo	IBlockedMessageInfo	含敏感信息的消息被拦截的回调参数

- `IBlockedMessageInfo` 说明

属性名称	说明
blockType	获取消息被拦截的原因，详见下方 `MessageBlockType` 说明。
blockedMessageUId	被拦截消息的唯一 ID。
channelId	被拦截消息所在会话的频道 ID（仅适用于会话类型为超级群的消息）。

属性名称	说明
conversationType	被拦截消息所在会话的会话类型。
extra	被拦截消息的附加信息。Web SDK 从 5.3.3 版本开始支持该参数。
targetId	被拦截消息的所在会话的会话 ID。
sourceType	被拦截消息源触发类型，详见下方 `MessageBlockSourceType` 说明。（仅超级群）
sourceContent	被拦截消息源内容对象。`sourceType` 字段为 1 时表示扩展内容，`sourceType` 为 2 时表示消息内容。（仅超级群）

- `MessageBlockType` 说明

枚举值	数值	说明
GLOBAL	1	全局敏感词：命中了融云内置的全局敏感词
CUSTOM	2	自定义敏感词拦截：命中了客户在融云自定义的敏感词
THIRD_PARTY	3	第三方审核拦截：命中了第三方（数美）或消息回调服务（原模板路由服务）决定不下发的状态

- `MessageBlockSourceType` 说明

枚举值	数值	说明
MSG_ORIGINAL	0	原始消息
MSG_EXPANSION	1	消息扩展
MSG_MODIFY	2	消息修改

IM 翻译插件

更新时间:2024-08-30

> 提示
>
> - IMLib 从 5.1.2 版本开始支持翻译插件。
> - 该插件暂仅适用于使用新加坡数据中心的应用。详见海外数据中心。

融云即时通讯业务提供翻译插件，可为 IMLib SDK 快速接入外部翻译服务，由融云服务端负责对接外部翻译服务供应商的鉴权、API 调用、账号管理、计费等流程。翻译插件支持翻译文本。

目前已支持接入 Google 翻译服务。

翻译流程

服务开通

该功能为付费增值服务。如有需求，请前往控制台 IM 翻译 页面开通服务。

关于 IM 翻译服务费用，详见 IM 翻译计费说明 。

客户端鉴权

客户端需要持有有效的 JWT Token，才能向融云请求翻译结果。

您的 App 服务端需要调用融云服务端 API 接口获取 JWT Token，然后返回给客户端。详见服务端文档获取 JWT Token。

> 提示
>
> 翻译插件鉴权专用的 JWT Token 不同于 IM 用户连接 IM 服务的 Token，请注意区分。

JWT

JWT 全称 JSON web Token，是为了在网络应用环境间传递声明而执行的一种基于 JSON 的开放标准。 JWT 包含 header、payload、signature 三部分。通过解析 payload 部分可获取到 Token 有效期和 UserId 等信息。

获取和刷新 JWT Token 流程图

IMKit 使用翻译插件

Web IMKit 暂不支持使用翻译插件。

IMLib 使用翻译插件

1. 安装插件

   示例代码

   - NPM 方式：

   ```
   npm install @rongcloud/engine @rongcloud/imlib-next @rongcloud/plugin-translate -S
   ```

   - CDN 方式：

   ```
   <script src="https://cdn.ronghub.com/RCTranslate-1.0.1.prod.js"></script>
   ```

2. 初始化，并连接融云 IM 服务。

   示例代码

   ```
   import * as RCTranslate from '@rongcloud/plugin-translate'
   import * as RongIMLib from '@rongcloud/imlib-next'

   // 初始化 IM
   RongIMLib.init({ appkey: '<appkey>' })
   const translateClient = RongIMLib.installPlugin(RCTranslate.installer)
   RongIMLib.connect('<token>').then(res => {
   if (res.code === 0) {
   console.log('连接成功')
   }
   })
   ```

3. 在用户登录成功之后，需先判断当前是否已开通翻译服务。

   示例代码

   ```
   translateClient.isSupport()
   ```

4. 在确认支持翻译服务之后，可以开始进行客户端鉴权。App 需要向自身的应用服务器发起请求，由应用服务器调用融云服务端 API 获取 JWT Token。App 获取 JWT Token 后，通过 `setAuthToken` 接口设置到 SDK 中。

   示例代码

```
translateClient.setAuthToken('<鉴权token>')
```

5. 调用 translateText 翻译文本

示例代码

```
const params = {
content: '', // 要翻译的文本
target: RCTranslate.LanguageVerify.zh_CN, // 目标语言类型
source: RCTranslate.LanguageVerify.en // 源语言类型
}
translateClient.translateText(params).then(({code, data, msg}) => {
if (code === RCTranslate.TranslateCode.TranslateCodeSuccess) {
// data： 翻译结果
console.log('翻译成功', data)
} else {
// msg: 错误信息
console.log('翻译失败', code, msg)
}
})
```

> 提示
>
> 源语言类型为可选参数。Google 翻译服务会自动识别待翻译文本的源语言，并仅以识别结果为准。

支持的语言类型

翻译插件支持的语言可参见文档中列出的语言列表。

Google 翻译服务

翻译插件已支持通过 Google Cloud Translation 服务翻译以下语言。更多细节，您可以直接参考 Google Cloud Translation 官方文档：语言列表 。

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
南非荷兰语	af
阿尔巴尼亚语	sq
阿姆哈拉语	am
阿拉伯语	ar
亚美尼亚文	hy
阿萨姆语	as
艾马拉语	ay
阿塞拜疆语	az
班巴拉语	bm
巴斯克语	eu
白俄罗斯语	be

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
孟加拉文	bn
博杰普尔语	bho
波斯尼亚语	bs
保加利亚语	bg
加泰罗尼亚语	ca
宿务语	ceb
中文（简体）	zh_CN 或 zh
中文（繁体）	zh_TW
科西嘉语	co
克罗地亚语	hr
捷克语	cs
丹麦语	da
迪维希语	dv
多格来语	doi
荷兰语	nl
英语	en
世界语	eo
爱沙尼亚语	et
埃维语	ee
菲律宾语（塔加拉语）	fil
芬兰语	fi
法语	fr
弗里斯兰语	fy
加利西亚语	gl
格鲁吉亚语	ka
德语	de
希腊文	el
瓜拉尼人	gn
古吉拉特文	gu
海地克里奥尔语	ht
豪萨语	ha
夏威夷语	haw
希伯来语	he 或 iw
印地语	hi
苗语	hmn
匈牙利语	hu
冰岛语	is
伊博语	ig
伊洛卡诺语	ilo
印度尼西亚语	id

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
爱尔兰语	ga
意大利语	it
日语	ja
爪哇语	jv 或 jw
卡纳达文	kn
哈萨克语	kk
高棉语	km
卢旺达语	rw
贡根语	gom
韩语	ko
克里奥尔语	kri
库尔德语	ku
库尔德语（索拉尼）	ckb
吉尔吉斯语	ky
老挝语	lo
拉丁文	la
拉脱维亚语	lv
林格拉语	ln
立陶宛语	lt
卢干达语	lg
卢森堡语	lb
马其顿语	mk
迈蒂利语	mai
马尔加什语	mg
马来语	ms
马拉雅拉姆文	ml
马耳他语	mt
毛利语	mi
马拉地语	mr
梅泰语（曼尼普尔语）	mni_Mtei
米佐语	lus
蒙古文	mn
缅甸语	my
尼泊尔语	ne
挪威语	no
尼杨扎语（齐切瓦语）	ny
奥里亚语（奥里亚）	or
奥罗莫语	om
普什图语	ps
波斯语	fa

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
波兰语	pl
葡萄牙语（葡萄牙、巴西）	pt
旁遮普语	pa
克丘亚语	qu
罗马尼亚语	ro
俄语	ru
萨摩亚语	sm
梵语	sa
苏格兰盖尔语	gd
塞佩蒂语	nso
塞尔维亚语	sr
塞索托语	st
修纳语	sn
信德语	sd
僧伽罗语	si
斯洛伐克语	sk
斯洛文尼亚语	sl
索马里语	so
西班牙语	es
巽他语	su
斯瓦希里语	sw
瑞典语	sv
塔加路语（菲律宾语）	tl
塔吉克语	tg
泰米尔语	ta
鞑靼语	tt
泰卢固语	te
泰语	th
蒂格尼亚语	ti
宗加语	ts
土耳其语	tr
土库曼语	tk
契维语（阿坎语）	ak
乌克兰语	uk
乌尔都语	ur
维吾尔语	ug
乌兹别克语	uz
越南语	vi
威尔士语	cy
班图语	xh

语言	枚举值（已替换 ISO-639 语言代码中的 - 为 _）
意第绪语	yi
约鲁巴语	yo
祖鲁语	zu

状态码

状态码	原因
26200	翻译成功
26201	翻译失败，融云鉴权失败 鉴权失败或者 token 过期
26202	翻译失败，翻译功能服务商鉴权失败 融云服务器的原因，token 无效
26203	翻译失败，翻译功能服务商返回失败 具体服务商失败码信息
26204	翻译失败，翻译功能未在融云开启
26205	翻译失败，融云限流
26206	翻译失败，Server 没有鉴权 token 的 sercret 需要在控制台开启
34100	没有设置 authToken 或者 authToken 为 空串
34101	待翻译文本内容为空
34102	目标语言为空

2.X 升级到 5.X（Web）

更新时间:2024-08-30

本文描述如何从 IMLib SDK 2.X 版本（`@rongcloud/imlib-v2`） 升级至 5.X 版本的步骤。

Web 平台升级概述

IMLib SDK 5.X 是即时通讯业务客户端 SDK 的最新版本， 对 Typescript 的使用者提供了友好的类型化支持。相对于 2.X 版本，5.X 版本功能更丰富，更稳定，并在之前版本上修复了大量问题，我们建议融云客户尽早升级至新版 IMLib SDK。

> 提示
>
> 融云同时为已集成 IMLib v2 版本（`@rongcloud/imlib-v2`）的客户提供了基于 Adapter 库（`RongIMLib-v2-Adapter`）替换升级选项。请注意，Adapter 库仅会提供问题修复，但不会在旧版 SDK 基础上增加新功能。详见关于 Adapter 的说明。

评估升级工作量

IMLib 5x SDK 与旧版 SDK 不兼容。我们整理了新旧版本 SDK API 对应关系与差异。请开发者根据自身 API 使用情况与 API 差异，合理安排开发周期。

- 新旧版本接口接口的名称和参数差异较大。升级前需要详细对照。
- 新旧版本接口返回值差异较大，可根据接口名称速查对照表比对各个接口返回值差异。

从 v2.5 或更早版本升级

IMLib 2.5 及更早版本，需注意如下变更：

> 提示
>
> 1. SDK 不再支持公众号与客服插件相关接口！
> 2. IM 链接因网络问题意外中断后，SDK 会自动重连，应用层无需再调用 `reconnect` 方法。
> 3. 若 `'RC:ProfileNtf'`、`'RC:CmdNtf'`、`'RC:InfoNtf'` 类型消息的 `content.data` 字段为 Json 字符串，需自行解析。
> 4. SDK 不再兼容 IE 浏览器 6 至 8 版本。

具体废弃功能参见下表：

分类	废弃方法	描述
会话	getConversationUnreadCount	按会话类型获取会话未读数
会话	clearConversations	按会话类型删除会话

分类	废弃方法	描述
会话	clearTotalUnreadCount	清除所有会话未读数
讨论组	createDiscussion	创建讨论组
讨论组	getDiscussion	获取讨论组信息
讨论组	quitDiscussion	退出讨论组
讨论组	addMemberToDiscussion	加入讨论组
讨论组	removeMemberFromDiscussion	将指定成员移除讨论租
讨论组	setDiscussionInviteStatus	设置讨论组邀请状态
讨论组	setDiscussionName	设置讨论组名称

新旧 API 对照速查表

描述	2.x API	5.x API
初始化	init	init
设置监听	setConnectionStatusListener	addEventListener
重连	reconnect	无需应用侧调用重连，详见SDK 重连机制
断开连接	disconnect	disconnect
退出登录	logout	disconnect
获取会话列表	getConversationList	getConversationList
清除会话列表	clearConversations	5.X 版本客户端不提供该接口。
获取指定会话	getConversation	getConversation
删除指定会话	removeConversation	removeConversation
获取草稿	getTextMessageDraft	getTextMessageDraft
保存草稿	saveTextMessageDraft	saveTextMessageDraft
删除草稿	clearTextMessageDraft	clearTextMessageDraft
获取所有会话未读数	getTotalUnreadCount	getTotalUnreadCount
获取单个会话未读数	getUnreadCount	getUnreadCount
按会话类型获取未读数	getConversationUnreadCount	getTotalUnreadCount
清除单个会话未读数	clearUnreadCount	clearMessagesUnreadStatus
清除全部会话未读数	clearAllUnreadCount	clearAllMessagesUnreadStatus
会话置顶	setConversationStatus	setConversationToTop
会话免打扰	setConversationStatus	setConversationNotificationLevel
创建标签	createTag	addTag
移除标签	removeTag	removeTag
编辑标签	updateTag	updateTag
获取标签列表	getTagList	getTags
添加会话到一个标签	addTagForConversations	addConversationsToTag
删除指定标签中某些会话	removeTagForConversations	removeTagsFromConversations
删除指定会话中的某些标签	removeTagsForConversation	removeTagsFromConversation
获取指定会话下的所有标签	getTagsForConversation	getTagsFromConversation

描述	2.x API	5.x API
分页获取指定标签下的会话列表	getConversationListByTag	getConversationsFromTagByPage
按标签获取未读消息数	getUnreadCountByTag	getUnreadCountByTag
设置标签中会话置顶	setConversationStatusInTag	setConversationToTopInTag
发送消息	sendMessage	sendMessage
注册自定义消息	registerMessageType	registerMessageType
历史消息获取	getHistoryMessages	getHistoryMessages
单聊消息回执	实现方案需要整体替换成 5x SDK 的接口	sendReadReceiptMessage
群聊消息回执	实现方案需要整体替换成 5x SDK 的接口	sendReadReceiptRequest ，sendReadReceiptResponseV2
消息撤回	sendRecallMessage	recallMessage
消息删除	deleteRemoteMessages	deleteMessages
更新消息扩展	updateMessageExpansion	updateMessageExpansion
删除消息扩展	removeMessageExpansionForKey	removeMessageExpansionForKey
加入聊天室	joinChatRoom	joinChatRoom
退出聊天室	quitChatRoom	quitChatRoom
查询聊天室信息	getChatRoomInfo	getChatRoomInfo
聊天室设置属性	setChatroomEntry	setChatRoomEntry
聊天室批量设置属性	setChatRoomEntries	setChatRoomEntries
聊天室强制设置属性	forceSetChatroomEntry	forceSetChatRoomEntry
聊天室删除属性	removeChatroomEntry	removeChatRoomEntry
聊天室强制删除属性	forceRemoveChatroomEntry	forceRemoveChatRoomEntry
聊天室获取单个属性	getChatroomEntry	getChatRoomEntry
聊天室获取所有属性	getAllChatroomEntries	getAllChatRoomEntries
获取聊天室历史消息	getChatRoomHistoryMessages	getChatroomHistoryMessages

2.X 升级到 5.X（Electron）

更新时间:2024-08-30

本文描述如何从 IMLib SDK 2.X 版本（`@rongcloud/imlib-v2`） 升级至 5.X 版本的步骤。

Electron 平台升级概述

IMLib SDK 5.X 是即时通讯业务客户端 SDK 的最新版本， 对 Typescript 的使用者提供了友好的类型化支持。相对于 2.X 版本，5.X 版本功能更丰富，更稳定，并在之前版本上修复了大量问题，我们建议融云客户尽早升级至新版 IMLib SDK。

> 提示
>
> 融云同时为已集成 IMLib v2 版本（`@rongcloud/imlib-v2`）的客户提供了基于 Adapter 库（`RongIMLib-v2-Adapter`）替换升级选项。请注意，Adapter 库仅会提供问题修复，但不会在旧版 SDK 基础上增加新功能。详见关于 Adapter 的说明。

评估升级工作量

IMLib 5x SDK 与旧版 SDK 不兼容。我们整理了新旧版本 SDK API 对应关系与差异。请开发者根据自身 API 使用情况与 API 差异，合理安排开发周期。

- 新旧版本接口接口的名称和参数差异较大。升级前需要详细对照。
- 新旧版本接口返回值差异较大，可根据接口名称速查对照表比对各个接口返回值差异。

升级注意事项

在 Electron 框架中使用 IMLib 5.X SDK，需要在主进程中引用 `@rongcloud/electron` 包，并在 app 的 ready 事件通知后进行主进程初始化。在渲染进程中初始化 IMLib，详细请阅读 适配 Electron 框架 相关文档完成升级。

Electron 包变化

需要删除 `electron-solution` 包，重新引入 `@rongcloud/electron` 与 `@rongcloud/electron-renderer`

Electron 独有接口

升级到 5.X 后，如果调用 Electron 独有接口，需要增加 **electronExtension** 属性标识。调用方式示例如下：

```
RongIMLib.electronExtension.getAllConversationList();
```

下表列出了 IMLib 5.X 提供的 Electron 独有接口。

描述	5.x API
获取本地全部会话	getAllConversationList

描述	5.x API
分页获取本地会话	getConversationList
搜索本地会话	searchConversationByContent
根据关键字搜索本地消息	searchMessages
指定时间范围内搜索本地消息	searchMessageInTimeRange
获取指定消息类型的历史消息	getHistoryMessagesByMessageTypes
通过时间戳设置消息状态为对方已读	setMessageStatusToRead
设置消息的接收状态	setMessageReceivedStatus
设置消息的发送状态	setMessageSentStatus
通过消息 ID 删除	deleteMessages
通过会话删除	clearMessages
通过时间戳删除	deleteMessagesByTimestamp
插入单条消息	insertMessage
批量插入消息	batchInsertMessage

从 v2.5 或更早版本升级

IMLib 2.5 及更早版本，需注意如下变更：

> 提示
>
> 1. SDK 不再支持公众号与客服插件相关接口！
> 2. IM 链接因网络问题意外中断后，SDK 会自动重连，应用层无需再调用 `reconnect` 方法。
> 3. 若 `'RC:ProfileNtf'`、`'RC:CmdNtf'`、`'RC:InfoNtf'` 类型消息的 `content.data` 字段为 Json 字符串，需自行解析。
> 4. SDK 不再兼容 IE 浏览器 6 至 8 版本。

具体废弃功能参见下表：

分类	废弃方法	描述
会话	getConversationUnreadCount	按会话类型获取会话未读数
会话	clearConversations	按会话类型删除会话
会话	clearTotalUnreadCount	清除所有会话未读数
讨论组	createDiscussion	创建讨论组
讨论组	getDiscussion	获取讨论组信息
讨论组	quitDiscussion	退出讨论组
讨论组	addMemberToDiscussion	加入讨论组
讨论组	removeMemberFromDiscussion	将指定成员移除讨论租
讨论组	setDiscussionInviteStatus	设置讨论组邀请状态
讨论组	setDiscussionName	设置讨论组名称

新旧 API 对照速查表

描述	2.x API	5.x API
初始化	init	init
设置监听	setConnectionStatusListener	addEventListener
重连	reconnect	无需应用侧调用重连，详见SDK 重连机制
断开连接	disconnect	disconnect
退出登录	logout	disconnect
获取会话列表	getConversationList	getConversationList
清除会话列表	clearConversations	5.X 版本客户端不提供该接口。
获取指定会话	getConversation	getConversation
删除指定会话	removeConversation	removeConversation
获取草稿	getTextMessageDraft	getTextMessageDraft
保存草稿	saveTextMessageDraft	saveTextMessageDraft
删除草稿	clearTextMessageDraft	clearTextMessageDraft
获取所有会话未读数	getTotalUnreadCount	getTotalUnreadCount
获取单个会话未读数	getUnreadCount	getUnreadCount
按会话类型获取未读数	getConversationUnreadCount	getTotalUnreadCount
清除单个会话未读数	clearUnreadCount	clearMessagesUnreadStatus
清除全部会话未读数	clearAllUnreadCount	clearAllMessagesUnreadStatus
会话置顶	setConversationStatus	setConversationToTop
会话免打扰	setConversationStatus	setConversationNotificationLevel
创建标签	createTag	addTag
移除标签	removeTag	removeTag
编辑标签	updateTag	updateTag
获取标签列表	getTagList	getTags
添加会话到一个标签	addTagForConversations	addConversationsToTag
删除指定标签中某些会话	removeTagForConversations	removeTagsFromConversations
删除指定会话中的某些标签	removeTagsForConversation	removeTagsFromConversation
获取指定会话下的所有标签	getTagsForConversation	getTagsFromConversation
分页获取指定标签下的会话列表	getConversationListByTag	getConversationsFromTagByPage
按标签获取未读消息数	getUnreadCountByTag	getUnreadCountByTag
设置标签中会话置顶	setConversationStatusInTag	setConversationToTopInTag
发送消息	sendMessage	sendMessage
注册自定义消息	registerMessageType	registerMessageType
历史消息获取	getHistoryMessages	getHistoryMessages
单聊消息回执	实现方案需要整体替换成 5x SDK 的接口	sendReadReceiptMessage
群聊消息回执	实现方案需要整体替换成 5x SDK 的接口	sendReadReceiptRequest ，sendReadReceiptResponseV2
消息撤回	sendRecallMessage	recallMessage
消息删除	deleteRemoteMessages	deleteMessages
更新消息扩展	updateMessageExpansion	updateMessageExpansion

描述	2.x API	5.x API
删除消息扩展	removeMessageExpansionForKey	removeMessageExpansionForKey
加入聊天室	joinChatRoom	joinChatRoom
退出聊天室	quitChatRoom	quitChatRoom
查询聊天室信息	getChatRoomInfo	getChatRoomInfo
聊天室设置属性	setChatroomEntry	setChatRoomEntry
聊天室批量设置属性	setChatRoomEntries	setChatRoomEntries
聊天室强制设置属性	forceSetChatroomEntry	forceSetChatRoomEntry
聊天室删除属性	removeChatroomEntry	removeChatRoomEntry
聊天室强制删除属性	forceRemoveChatroomEntry	forceRemoveChatRoomEntry
聊天室获取单个属性	getChatroomEntry	getChatRoomEntry
聊天室获取所有属性	getAllChatroomEntries	getAllChatRoomEntries
获取聊天室历史消息	getChatRoomHistoryMessages	getChatroomHistoryMessages

关于 Adapter 的说明　　什么是 Adapter SDK？

更新时间:2024-08-30

使用 5.x 版本 SDK 的客户无需关注 Adapter 版本 SDK。

Adapter 版本是针对 2022年 5 月 5 日前已集成旧版 2.x/4.x SDK 的客户推出的 SDK，仅用于后续维护。

- 我们建议使用融云旧版 SDK 客户尽早升级至 5.X 版本 SDK。5.X 版本更稳定，功能更丰富。
- 您可能会看到 v2/v4 对应的 Adapter SDK 的版本号从 5.x.x 开始迭代，但请注意该版本号与 IMLib 5.x 版本 SDK 无关。

针对 2.x/4.x 版本 SDK 客户的说明

2022年 5 月 5 日前集成 SDK 的客户，可以将旧版 SDK 替换为对应的 Adapter 版本 SDK（`RongIMLib-v2-Adapter`/`RongIMLib-v4-Adapter`）。

具体细节如下：

- 后续将转至 Adapter 版本的 SDK 上进行问题修复。
- 仅用于后续问题修复，不会增加新功能。
- **务必注意修改集成方式**，Adapter 版本 SDK 支持无缝替换升级。详见各版本升级说明：
  - IMLib v2 升级说明
  - IMLib v4 升级说明
- 旧版 IMLib 2.x/4.x SDK 同时停止维护，具体如下：
  - NPM：`@rongcloud/imlib-v2`
  - NPM：`@rongcloud/imlib-v4`
  - CDN：`RongIMLib-2.x.x. ... .js`
  - CDN：`RongIMLib-4.x.x. ... .js`
- v2/v4 对应的 Adapter SDK 的版本号将从 5.x.x 开始迭代。详各版本升级说明。
  - v2 版本描述
  - v4 版本描述
- 替换为 Adapter 版本 SDK 后，请继续使用 v2/v4 开发者文档。
  - IMLib v2 文档
  - IMLib v4 文档

更新日志

更新时间:2024-08-30

IMKit / IMLib 独立发版，请参见对应的版本描述：

- IMKit 版本描述
- IMLib 版本描述

状态码

更新时间:2024-08-30

本文档列出了 RongIMLib 的所有状态码，内容基于枚举 ErrorCode 。

连接状态码

App 可以监听 SDK 连接状态。在连接中可能抛出以下状态码。

> 提示
>
> - 5.6.1 版本之前，事件 Events.DISCONNECT 与 Events.SUSPEND 回调函数中的 code 类型为 ConnectionStatus 。
> - 5.7.0 版本之后，事件 Events.DISCONNECT 与 Events.SUSPEND 回调函数中的 code 类型为 ErrorCode 。

ConnectionStatus 中提供以下连接状态：

状态码	说明
0	连接成功
1	正在连接中
2	用户主动断开连接
3	网络不可用，SDK 内部会自动重连
4	Socket 不可用, SDK 内部会自动重连
6	被其他端踢掉
9	用户被封禁
12	域名错误

业务错误码

状态码	枚举	说明
-1	TIMEOUT	未知错误
0	SUCCESS	成功
2	ACTIVE_DISCONNECT	主动断开连接
405	REJECTED_BY_BLACKLIST	已被对方加入黑名单，消息发送失败
20106		用户已经被设置单聊禁言
20109	SYS_CONVERSATION_NOT_SUPPORT_MESSAGE	系统会话不支持发送该消息
20604	SEND_FREQUENCY_TOO_FAST	发送消息频率过高，1 秒钟最多只允许发送 5 条消息
20605	OPERATION_BLOCKED	操作被禁止，此错误码已被弃用
20606	OPERATION_NOT_SUPPORT	操作不支持，仅私有云有效，服务端禁用了该操作
20607	REQUEST_OVER_FREQUENCY	请求超出了调用频率限制，请稍后再试
21406	NOT_IN_DISCUSSION	不在讨论组
21501	SENSITIVE_SHIELD	发送的消息中包含敏感词 （发送方发送失败，接收方不会收到消息）
21502	SENSITIVE_REPLACE	消息中敏感词已经被替换 （接收方可以收到被替换之后的消息）
22201	MESSAGE_EXPAND_NOT_EXIST	超级群扩展消息，但是原始消息不存在。
22202	MESSAGE_EXPAND_NOT_SUPPORT	超级群扩展消息，但是原始消息不支持扩展
22203	MESSAGE_EXPAND_FORMAT_ERROR	超级群扩展消息，扩展内容格式错误
22204	MESSAGE_EXPAND_NOT_AUTHORIZED	超级群扩展消息，无操作权限
22406	NOT_IN_GROUP	不在该群组中
22408	FORBIDDEN_IN_GROUP	在群组中已被禁言。

状态码	枚举	说明
23406	NOT_IN_CHATROOM	不在该聊天室中。
23407	GET_USERINFO_ERROR	获取用户失败
23408	FORBIDDEN_IN_CHATROOM	在该聊天室中已被禁言
23409	RC_CHATROOM_USER_KICKED	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	RC_CHATROOM_NOT_EXIST	聊天室不存在
23411	RC_CHATROOM_IS_FULL	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	RC_CHATROOM_PATAMETER_INVALID	聊天室接口参数无效。请确认参数是否为空或者有效。
23413	CHATROOM_GET_HISTORYMSG_ERROR	查询聊天室历史消息异常
23414	CHATROOM_NOT_OPEN_HISTORYMSG_STORE	聊天室云存储业务未开通
23423	CHATROOM_KV_EXCEED	聊天室的 KV 属性个数超限，单个聊天室默认上限为 100 个
23424	CHATROOM_KV_OVERWRITE_INVALID	没有权限修改聊天室中已存在的属性值
23425	CHATROOM_SET_PROPERTY_OVER_FREQUENCY	超过聊天室中状态设置频率，1 个聊天室 1 秒钟最多设置和删除状态 100 次
23426	CHATROOM_KV_STORE_NOT_OPEN	聊天室属性自定义设置，您可以在控制台免费基础功能页面中开启该功能。
23427	CHATROOM_KEY_NOT_EXIST	聊天室属性不存在
23428	CHATROOM_KV_STORE_NOT_ALL_SUCCESS	聊天室批量设置或删除KV部分不成功
23429	CHATROOM_KV_STORE_OUT_LIMIT	聊天室批量设置或删除KV数量超限（最多一次10条）
23431	CHATROOM_KV_SET_ERROR	聊天室设置 KV 失败，出现在两人或者多端同时操作一个 KV。 如果出现该错误，为避免和其他端同时操作，请延时一定时间再试
24401	ULTRA_GROUP_SERVICE_UNAVAILABLE	超级群功能未开通
24402	ULTRA_GROUP_SERVICE_ERROR	超级群服务异常

状态码	枚举	说明
24403	ULTRA_GROUP_PARAMETER_ERROR	超级群参数错误
24404	ULTRA_GROUP_UNKNOWN_ERROR	超级群未知异常
24406	NOT_IN_ULTRA_GROUP	非超级群成员
24408	FORBIDDEN_IN_ULTRA_GROUP	超级群成员禁言
24410	ULTRA_GROUP_NOT_EXIST	超级群不存在
24411	ULTRA_GROUP_MEMBERS_OVERSIZE	超级群成员超限制
24412	ULTRA_GROUP_JOINED_OVERSIZE	用户加入超级群数量超限
24413	ULTRA_GROUP_CHANNELS_OVERSIZE	创建超级群频道，频道数超限
24414	ULTRA_GROUP_CHANNEL_ID_NOT_EXIST	超级群频道 ID 不存在
24415	ULTRA_GROUP_MESSAGE_SENT_OVER_FREQUENCY	超级群频道发送消息超限：超级群下每个频道有消息发送频率限制，默认每秒 30 条 频道内每秒发送消息总量超过限制会收到该错误码，建议延时发送或重试发送
24416	ULTRA_GROUP_USER_NOT_IN_PRIVATE_CHANNEL	用户不在超级群私有频道中
25101	RECALL_MESSAGE	撤回消息参数无效，请确认撤回消息参数是否正确的填写
25102	MESSAGE_STORAGE_SERVICE_UNAVAILABLE	未开通单群聊云存储服务
25107	RECALL_MESSAGE_USER_INVALID	IMLib 撤回消息可以撤回自己发送的消息和别人发送的消息，IM 服务有开关，控制只可以撤回自己发送的消息，当服务该开关打开时，撤回别人的消息会报这个错误。
26001	PUSH_PARAMETER_INVALID	远程推送设置参数无效，请确认是否正确的填写了远程推送参数
26002	USER_SETTING_SYNCED_ERROR	表示客户端版本号低，需要同步版本号，可以提交工单申请打开用户级别配置开关
26004	CONVERSATION_TAG_OVERSIZE	用户会话标签个数超限，最多支持添加 20 个标签
30001	RC_NET_CHANNEL_INVALID	当前连接已经被释放
30002	RC_NET_UNAVAILABLE	当前连接不可用

状态码	枚举	说明
30003	RC_MSG_RESP_TIMEOUT	客户端发送消息请求，融云服务端响应超时
30004	RC_HTTP_SEND_FAIL	导航操作时，Http 请求失败
30005	RC_HTTP_REQ_FAIL	请求连接导航地址失败
30006	RC_HTTP_RECV_FAIL	导航操作时，HTTP 接收失败
30007	RC_NAVI_REQ_FAILED	导航返回结果异常
30008	RC_NODE_NOT_FOUND	导航 HTTP 返回数据格式错误
30009	RC_DOMAIN_NOT_RESOLVE	导航数据解析后，其中不存在有效 IP 地址
30010	RC_SOCKET_NOT_CREATED	创建连接失败
30011	RC_SOCKET_DISCONNECTED	链接断开
30012	RC_PING_EXCEED_LIMIT	PING 失败
30013	PONG_RECEIVED_ERROR	PING 超时
30014	RC_MSG_SEND_FAIL	信令发送失败
30015	CONNECT_OVER_FREQUENCY	连接过于频繁
30016	RC_MSG_CONTENT_EXCEED_LIMIT	消息大小超限，消息体（序列化成 json 格式之后的内容）最大 128k bytes
30019	RC_NETWORK_DOWN	网络连接不可用
30021	RC_TCP_DISCONNECTED_NO_RMTP	tcp 连接成功，rmtp 连接失败
31000	RC_CONN_ACK_TIMEOUT	连接ACK超时
31001	RC_CONN_PROTO_VERSION_ERROR	信令版本错误
31002	RC_CONN_IDENTIFIER_REJECTED	客户端 info 字段格式错误，正确格式：{平台类型}-{设备信息}-{sdk版本}。 其中设备信息：{手机类型}

状态码	枚举	说明
31003	RC_CONN_SERVER_UNAVAILABLE	连接服务未开通，需要排查后台小程序（或桌面端）服务是否已开通
31004	RC_CONN_TOKEN_INCORRECT	Token 无效；AppKey 和 Token 不匹配；Token 过期
31005	RC_CONN_NOT_AUTHRORIZED	App 校验未通过（开通了 App 校验功能，但是校验未通过）
31006	RC_CONN_REDIRECTED	连接重定向
31007	RC_CONN_PACKAGE_NAME_INVALID	包名与后台注册信息不匹配
31008	RC_CONN_APP_BLOCKED_OR_DELETED	AppKey 被封禁或已删除
31009	RC_CONN_USER_BLOCKED	用户被封禁
31010	RC_DISCONN_KICK	用户被踢下线
31011	RC_DISCONN_EXCEPTION	与服务器的连接已断开,用户被封禁
31012	CONNECTION_ENCRYPT_AUTHORIZED_ERROR	链路加密认证失败
31020	RC_CONN_TOKEN_EXPIRED	Token 已过期
31021	RC_CONN_DEVICE_ERROR	Token 中携带 deviceId 时，检测 Token 中deviceId 与链接设备 deviceId 不一致
31022	RC_CONN_HOSTNAME_ERROR	页面域名不在安全域名白名单内，需通过控制台添加安全域名配置
31023	RC_DISCONN_SAME_CLIENT_ON_LINE	开启禁止把已在线客户端踢下线开关后，该错误码标识已有同类型端在线
31024	RC_LISCENSE_COUNT_OUT_OF_LIMIT	连接总数量超过服务设定的并发限定值（私有云专属）
31025	RC_CONN_WRONG_CLUSTER	客户端连错环境，引发连接拒绝；如使用开发环境 Appkey 连接到生产环境
31026	RC_APP_AUTH_NOT_PASS	开启AppServer联合鉴权功能后，到AppServer认证失败
31027	RC_OTP_USED	该 token 已经被使用过,无法进行连接 一次性 token 只能连接一次，之后再使用会上报此错误
31028	RC_PLATFORM_ERROR	Token 平台验证失败

状态码	枚举	说明
31029	RC_ACCOUNT_CANCELLATION	用户已销户
31030	RC_LICENSE_EXPIRED	私有云 License 检查不通过;APP License 过期
32001	RC_QUERY_ACK_NO_DATA	协议层内部错误，查询，上传，下载过程中数据错误
32002	RC_MSG_DATA_INCOMPLETE	协议层内部错误
32011	RC_UDP_DISCONNECTED	服务器主动断开连接（仅 quic 协议下触发）
32054	RC_TCP_RESET	链接被服务器中断，可能原因是运营商认为此链接非法或无效，直接断开 出现此错误码后，SDK 会自动触发重连，App 无需处理
32061	CONNECTION_REFUSED	连接被拒绝
33000	PROTOCOL_MESSAGE_SAVED_ERROR	将消息存储到本地数据时失败。 发送或插入消息时，消息需要存储到本地数据库，当存库失败时，会回调此错误码
33001	BIZ_ERROR_CLIENT_NOT_INIT	协议栈未初始化
33002	BIZ_ERROR_DATABASE_ERROR	数据库错误
33003	BIZ_ERROR_INVALID_PARAMETER	开发者接口调用时传入的参数错误
33004	BIZ_ERROR_NO_CHANNEL	通道无效
33005	BIZ_ERROR_RECONNECT_SUCCESS	重新连接成功
33006	BIZ_ERROR_CONNECTING	连接中，再调用 connect 被拒绝
33007	MSG_ROAMING_SERVICE_UNAVAILABLE	历史消息云存储业务未开通
33008	MSG_INSERT_ERROR	消息存入本地数据库失败
33009	MSG_DEL_ERROR	聊天室被回收
33100	TAG_NOT_EXIST	标签不存在
33101	TAG_EXISTS	标签已存在

状态码	枚举	说明
33102	NO_TAG_IN_CONVER	标签不在会话中
34001	RC_CONNECTION_EXIST	连接已存在
34005	PACKAGE_ENVIRONMENT_ERROR	连接环境不正确
34006	CONNECTION_TIMEOUT	连接超时
34008	MESSAGE_KV_NOT_SUPPORT	消息不能被扩展，只支持单群聊，其他类型消息会返回此错误，消息在发送时，Message 对象的属性 canIncludeExpansion 置为 true 才能进行扩展
34009	MESSAGE_EXPANDED_ERROR	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	EXPANSION_LIMIT_EXCEET	消息扩展大小超出限制， 默认消息扩展字典 key 长度不超过 32 个字符，value 长度不超过 4096 个字符，设置的 Expansion 键值对不超过 300 个
34011	UPLOAD_FAIL	媒体消息媒体文件 http 上传失败
34013	CONVER_OUT_LIMIT_ERROR	标签中添加/删除的会话数量超限，最多支持添加/删除 1000 个会话
34014	READ_RECEIPT_ERROR	群已读回执版本不支持
34016	PUSH_CONTENT_CONFIG_SERVICE_UNAVAILABLE	用户级别设置未开通
34017	MESSAGE_INTERCEPTION_PRODUCED_NULL_MESSAGE	消息拦截器处理接口返回的消息为空
34021	MESSAGE_NOT_REGISTERED	消息未被注册
34022	METHOD_NOT_SUPPORT	该接口不支持超级群会话
34024	ULTRA_GROUP_CHANNEL_NOT_EXIST	超级群频道不存在
34025	INCONSISTENT_CONVERSATION_TYPE	扩展消息失败，因为消息中的会话类别与接口支持的会话类别不一致
34201	CONVERSATION_TYPE_NOT_SUPPORT	开发者调用的接口不支持传入的会话类型
34206	MSG_LIMIT_ERROR	开发者接口调用时传入的 messageList 非法
34238	INVALID_PARAMETER_PROXY	非法的代理配置，RongIMProxy 为空或者非法

状态码	枚举	说明
34239	INVALID_PARAMETER_TESTHOST	开发者接口调用 testProxy 时传入的代理测试服务非法
34240	INVALID_CONNECT_TESTHOST_FAILED	开发者接口调用 testProxy 接口时无法联通
34245	RC_CONN_PROXY_UNAVAILABLE	设置的代理地址不可用
35004	NOT_SUPPORT	方法未支持
35005	MAIN_PROCESS_ERROR	主进程内方法错误
35008	SERVER_UNAVAILABLE	无可用的 IM 服务地址
35011	METHOD_NOT_AVAILABLE	IM 在初次连接成功后，需立即同步消息，标记上线，然后获取全量超级群会话列表，列表同步完成前，所有超级群相关功能接口应不可用
35015	DRAFT_SAVE_ERROR	保存草稿失败
35016	DRAFT_REMOVE_ERROR	删除草稿失败
35020	CONVER_REMOVE_ERROR	删除会话失败
35021	CONVER_GET_ERROR	获取会话失败
35022	SEARCH_PROPS_LIMIT_ERROR	搜索字段对应消息错误
-2	UNKNOWN	未知原因失败。
-3	PARAMETER_ERROR	参数错误(该错误码已废弃, 请使用 ErrorCode.BIZ_ERROR_INVALID_PARAMETER)
-4	EXTRA_METHOD_UNDEFINED	未实现的方法定义，在应用层调用 callExtra 传入无法识别的方法名时抛出(该错误已废弃, 请使用 ErrorCode.NOT_SUPPORT)
-6	PARAMETER_CHANGED	参数变更
1	CANCEL	己方取消已发出的通话请求
2	REJECT	己方拒绝收到的通话请求
3	HANGUP	己方挂断

状态码	枚举	说明
4	BUSYLINE	己方忙碌
5	NO_RESPONSE	己方未接听
6	ENGINE_UN_SUPPORTED	己方不支持当前引擎
7	NETWORK_ERROR	己方网络出错
11	REMOTE_CANCEL	对方取消已发出的通话请求
12	REMOTE_REJECT	对方拒绝收到的通话请求
13	REMOTE_HANGUP	通话过程对方挂断
14	REMOTE_BUSYLINE	对方忙碌
15	REMOTE_NO_RESPONSE	对方未接听
16	REMOTE_ENGINE_UN_SUPPORTED	对方不支持当前引擎
17	REMOTE_NETWORK_ERROR	对方网络错误
18	VOIP_NOT_AVALIABLE	VoIP 不可用
21407	JOIN_IN_DISCUSSION	加入讨论失败
21408	CREATE_DISCUSSION	创建讨论组失败
21409	INVITE_DICUSSION	设置讨论组邀请状态失败
24001	HAVNODEVICEID	没有注册DeviveId 也就是用户没有登陆
24002	DEVICEIDISHAVE	已经存在
24009	FEILD	没有对应的用户或token
24013	VOIPISNULL	voip为空

状态码	枚举	说明
24010	NOENGINETYPE	不支持的Voip引擎
24011	NULLCHANNELNAME	channleName 是空
24012	VOIPDYANMICERROR	生成Voipkey失败
24014	NOVOIP	没有配置voip
24015	INTERNALERRROR	服务器内部错误
24016	VOIPCLOSE	VOIP close
35001	GROUP_SYNC_ERROR	群组信息异常
35002	GROUP_MATCH_ERROR	匹配群信息异常
35007	CAN_NOT_RECONNECT	已连接或者内部重连中，不允许调用重连，需先调用 disconnect 方法
35009	HOSTNAME_ERROR	Web 端设置安全域名后，连接端域名不在安全域名范围内
35010	HAS_OHTER_SAME_CLIENT_ON_LINE	开启禁止把已在线客户端踢下线开关后，该错误码标识已有同类型端在线，禁止链接
35014	METHOD_ONLY_SUPPORT_ULTRA_GROUP	该功能仅支持超级群(该错误码已废弃, 请使用 ErrorCode.CONVERSATION_TYPE_NOT_SUPPORT)
36001	CHATROOM_ID_ISNULL	加入聊天室Id为空
36002	CHARTOOM_JOIN_ERROR	加入聊天室失败
36003	CHATROOM_HISMESSAGE_ERROR	拉取聊天室历史消息失败
36004	CHATROOM_KV_NOT_FOUND	聊天室 kv 未找到
37001	BLACK_ADD_ERROR	加入黑名单异常
37002	BLACK_GETSTATUS_ERROR	获得指定人员再黑名单中的状态异常
37003	BLACK_REMOVE_ERROR	移除黑名单异常

状态码	枚举	说明
38001	DRAF_GET_ERROR	获取草稿失败
39001	SUBSCRIBE_ERROR	关注公众号失败
39002	NOT_SUPPORT	方法不支持。该错误码已在 5.8.4 版本移除，5.8.4 及之后版本对应错误码为 35004。
41001	QNTKN_FILETYPE_ERROR	关注公众号失败
41002	QNTKN_GET_ERROR	获取七牛token失败
51001	COOKIE_ENABLE	cookie被禁用