场景融合
服务端集成

2023-12-22

服务端开发指南

更新时间:2023-12-22

欢迎使用融云融合场景产品。

语聊房/视频直播房的业务服务端（App server）指您自己的 **App 业务服务器**，即为您自身的业务提供接口的后端，您需要自行实现。为帮助客户快速接入，融云提供了最佳实践方案、和 Demo server 供您参考。

语聊房/视频直播客户端 SDK 仅依赖融云提供的 IM 和 RTC 能力。因此，App 服务端可以使用融云 IM 服务端 API 与 RTC 服务端 API 所提供的全部能力构建 App 后台服务系统。包括但不限于：

- 用户注册
- 用户封禁、黑/白名单
- 群组禁言、聊天室禁言
- 消息回调、在线状态订阅
- 云端录制（增值服务）

> 如需了解融合场景下全部服务端能力，请前往服务端文档 · 即时通讯 · 实时音视频 »

概述 语聊房/视频直播房最佳实践

更新时间:2023-12-22

针对与场景化 SDK 相关，且涉及到客户端和服务端配合的常见产品功能，融云给出了以下最佳实践方案。

- PK 场景：包含了 PK 倒计时、PK 中送礼物、惩罚游戏等主流功能。
- 异常退出场景：包含了麦位主播异常退出的解决方案。
- 换设备登录：如何保证用户换设备登录后可正常使用业务。
- 数美审核: 音视频通话中的音视频流进行审核。
- 创建房间: 服务端创建房间并设置房间属性。

创建房间与设置属性

更新时间:2023-12-22

音视频 App 中经常有房间的概念。融云语聊房、视频直播房等产品均提供了从服务端 API 或客户端 SDK 创建房间、设置房间属性的能力。一般情况下，可由服务端创建房间成功后，再由客户端 SDK 去设置房间属性，但是也有一些场景需要由 App 服务端创建房间成功后再由 App 服务端预置一些房间属性。

本文描述了从 App 服务端创建房间、设置房间属性的基本流程。我们的 Demo Server 中有具体实现，您可以参考 Demo Server 接口 `/mic/room/create` 中的具体实现。

本文涉及的融云服务端 API 接口如下：

- **创建房间**: App 服务端需调用融云服务端 API 创建房间 接口创建房间。
- **设置房间属性**: App 服务端可调用融云融云服务端 API 属性管理 接口管理房间属性。

基本实现流程

1. 创建房间： 创建房间需要两个参数。一个是唯一的 roomId（可以用 UUID 生成 64位 的字符串当作 roomId）；另一个是房间名称，由客户端传过来，用来在房间列表中展示等。
2. 设置房间属性： 创建房间成功后，可以基于房间的 roomId 设置房间属性（Key-value）。请注意，不同产品的房间需要设置的房间属性 KV 不同。根据您选用的场景化客户端 SDK，设置不同的房间属性 KV。

语聊房需要设置的 KV

```
{
"RCRoomInfoKey" = "
{\"seatCount\":9,\"roomName\":\"We\",\"isMuteAll\":false,\"isFreeEnterSeat\":false,\"totalMemberCount\":
sLockAll\":false}";
"RCSeatInfoSeatPartPrefixKey_0" = "{\"status\":0,\"mute\":false}";
"RCSeatInfoSeatPartPrefixKey_2" = "{\"status\":0,\"mute\":true}";
"RCSeatInfoSeatPartPrefixKey_2" = "{\"status\":0,\"mute\":true,\"userId\":xxxx,\"extra\":""}";
}
```

- `RCSeatInfoSeatPartPrefixKey_*` 由 `seatCount` 决定。
- `userId` 为当前上麦用户 ID，默认为空。
- `extra` 为开发者自定义信息，一般为 JSON，默认为空。

视频直播房需要设置的 KV

```
{
"LIVE_VIDEO_ROOM_MIX_TYPE" = 1;
"LIVE_VIDEO_ROOM_USER_ID" = "8916bc0c-ee30-43d5-a51c-7b8c89780e4d";
"LIVE_VIDEO_SEAT_INFO_PRE_0" = "
{\"userEnableAudio\":true,\"enableTiny\":false,\"userEnableVideo\":true,\"index\":0,\"lock\":false,\"mut
alse,\"userId\":\"8916bc0c-ee30-43d5-a51c-7b8c89780e4d\"}";
"LIVE_VIDEO_SEAT_INFO_PRE_1" = "
{\"userEnableAudio\":true,\"enableTiny\":true,\"userEnableVideo\":true,\"index\":1,\"lock\":false,\"mute
lse}";
}
```

LIVE_VIDEO_SEAT_INFO_PRE_* 会根据 LIVE_VIDEO_ROOM_MIX_TYPE 的不同有所增减。

PK 场景

更新时间:2023-12-22

PK 场景在语聊房中的应用是比较常见的，我们的PK 场景包含了PK倒计时、PK 中送礼物、惩罚游戏等主流功能。

概览

下图展示了我们PK 场景的App 端截图:

场景描述

A主播向B主播发起PK，PK 开始，观众向主播送礼物，三分钟后，获得礼物积分最多的主播胜，胜利的主播可以要求PK失败的主播做三分钟的惩罚。至此PK 结束。

痛点问题

A主播向B主播发起PK，PK 开始，PK 3分钟后进入惩罚阶段，惩罚3分钟后PK 结束，这个过程如果在客户端做的话，会存在客户端PK 状态的不同步，比如说观众C看到的PK已经进入惩罚阶段，但是D观众可能还在PK 阶段，这种情况我们称为“多端倒计时同步问题”。

解决办法

因存在“多端倒计时同步问题”，PK 的各个阶段由服务端来把控，客户端向服务端只发起PK 开始的接口即可，客户端发起PK 开始接口成功后，服务端依赖IM 的即时通信能力来向客户端发送各个阶段的PK 状态来解决该问题。

详细方案

PK 场景由客户端发起，但是PK 阶段的各个状态由服务端来把控,PK 一共分三种状态，包括：

- PK 开始状态
- PK 惩罚状态
- PK 结束状态
  这三种状态由服务端发送给IM聊天室消息，客户端监听聊天室消息，收到PK 状态消息后做对应的业务处理

时序图

PK 流程

正常流程：首先客户端向服务端发起PK,服务端接收到PK 请求后，向IM 聊天室发送PK 开始状态消息，并开启一个“任务”，这个“任务” 的处理逻辑是三分钟后向IM聊天室发送PK 惩罚状态消息，再过3分钟向IM 发送PK 结束状态。该任务可以被打断，当PK 的房主主动结束PK 时，客户端向服务端发起PK 结束接口调用，该调用会停止该“任务”，并向聊天室发送停止消息。

异常流程：客户端发起PK，并进入PK 阶段，PK 中途想主动停止PK，这时需要由客户端调用PK 结束接口。PK 结束接口会停止向IM发送PK消息状态任务。

流程图

其他

在PK 阶段，后进入房间的用户需要获取到当前PK 的一些信息，比如PK 开始时间,两个房间的积分情况等，这时可以调用获取PK 详情接口

接口文档

PK 接口

请求地址: `http://localhost:8080/mic/room/pk`

请求方式: POST

输入参数:

参数名	类型	必选	说明
roomId	string	是	当前房间ID
toRoomId	string	是	pk房间ID
status	int	是	pk 状态 0 开始 2 结束

返回结果:

参数名	类型	说明
code	int	10000代表正常
msg	String	失败消息
data	String	

Request:

```
{
"roomId": "10000",
"toRoomId": "20000",
"status": 0
}
```

Response:

```
{
"code": 10000,
"msg": "success",
"data":
}
```

获取PK 详细信息接口

请求地址: `http://localhost:8081/mic/room/pk/detail/{roomId}`

请求方式: GET

输入参数:

参数名	类型	必选	说明
roomId	string	是	房间ID

返回结果:

参数名	类型	说明
code	int	返回码，10000为正常
msg	string	错误信息
data.timeDiff	int	pk开始距离现在当前时间的时间差
data.statusMsg	int	pk 状态信息，0:开始pk，1:惩罚阶段，2:pk结束
data.roomScores	List	房间信息
roomScores[i].roomId	string	房间Id
roomScores[i].userId	string	房主Id
roomScores[i].score	long	房间积分
roomScores[i].userInfoList	List	房间送礼物前三名列表
userInfoList[i].userId	string	用户id

参数名	类型	说明
userInfoList[i].userName	string	用户名
userInfoList[i].portrait	string	用户头像

Request:

```
{
"roomId": "10000"
}
```

Response:

```
{
"code": 10000,
"msg": "success",
"data": {
"timeDiff": 23456789,
"roomScores": [
{
"roomId": "34654",
"score": 21,
"userInfoList": [
{
"userId": "1",
"userName": "张三",
"portrait": "http://url"
},
{
"userId": "1",
"userName": "张三",
"portrait": "http://url"
}
]
},
{
"roomId": "34655",
"score": 22,
"userInfoList": [
{
"userId": "1",
"userName": "张三",
"portrait": "http://url"
},
{
"userId": "1",
"userName": "张三",
"portrait": "http://url"
}
]
}
]
}
}
```

IM聊天室三种消息

开始PK消息示例

```
{
"fromUserId": 0,
"toChatroomId":["24","45"],
"objectName":"RCMic:chrmPkStatusMsg",
"content": {
"stopPkRoomId": "",
"statusMsg": 0,
"timeDiff": ,
"roomScores": {
[
"userId":"3888",
"roomId":"345678",
"score":1234
]
}
}
}
```

惩罚消息示例

```
{
"fromUserId": 0,
"toChatroomId":["24","45"],
"objectName":"RCMic:chrmPkStatusMsg",
"content": {
"stopPkRoomId": "",
"statusMsg": 1,
"timeDiff": ,
"roomScores": {
[
"userId":"3888",
"roomId":"345678",
"score":1234
]
}
}
}
```

停止消息示例

```
{
"fromUserId": 0,
"toChatroomId":["24","45"],
"objectName":"RCMic:chrmPkStatusMsg",
"content": {
"stopPkRoomId": "",
"statusMsg": 2,
"timeDiff": 23456789,
"roomScores": {
[
"userId":"3888",
"roomId":"345678",
"score":1234
]
}
}
}
```

消息参数说明

参数名	类型	说明
stopPkRoomId	String	停止消息有意义，如果有值，则代表房主主动停止pK,无值代表自动停止PK
status	int	状态，0：开始PK；1:惩罚阶段；2：PK停止
timeDiff	Long	当前时间和pk开始时间的差值
roomId	int	房间id
userId	int	房主id
score	double	积分值

异常退出场景 概述

更新时间:2023-12-22

异常退出场景通常是因为用户网络出现问题，导致长时间与融云 RTC Server 断开连接，从而被 RTC Server 踢出房间。

这种非正常退出可能会造成以下情况：

- 如果断线前该用户在房间中是主播

  用户断开连接后，无法正常调用 `leaveSeat` 接口，导致该断线用户依然在麦位上。

- 如果断线前该用户是普通用户，并且该用户一直没有退出 App 客户端

  网络恢复时，在 App 客户端上，用户会发现此时仍然在房间界面，但是接收不到房间的任何信息与语音。

解决思路

用融云的实时音视频中提供的 Server 端 房间状态同步 来解决。

在用户异常退出时：

1. 融云的 RTC Server 会通过您在开发者后台设置的回调接口，向 App 服务器发送回调。
2. 当 App 服务器接收到回调时（可参考音视频房间状态同步接口，利用融云的 IM Server 中的 聊天室管理-属性管理 接口拉取房间最新的聊天室属性，判断该用户是否还在麦位上。
3. 如果该用户还在麦位上，则清除麦位上的用户信息。如果用户不在麦位上，则不需要清除麦位信息，需发送被踢出房间的信息。
4. 给 App 客户端发送一条被踢出房间的的自定义系统消息，当 App 客户端恢复连接时，会收到该条消息，之后根据用户前端此时是否在房间，给出相应的提示及动作。

详细方案

大致流程如下：

1. 应用服务端收到房间状态同步后，取得 `userId`、`roomId`、`event`、和 `exitType` 信息。
2. 如果 `event=12` 并且 `exitType=3`，说明房间里面的用户异常下线了。这种情况需要根据 `roomId` 去拿到房间属性信息，如果房间属性信息中的麦位信息中包含该 `userId`，并且 `status=1` 的话，我们把 `status` 设置为 0，并清除 `userId` 并重新设置房间属性。
3. 向 App 客户端发送条被踢出房间的的自定义系统消息来告诉 App 客户端用户已经被踢出房间。

异常退出流程图

最佳实践

> ⚠ 警告
>
> 下面是我们在 Demo 和 Demo server 中的实现，SDK 中没有封装。

音视频房间状态同步接口

请求地址：`http://localhost:8080/mic/video/room/status/sync`

请求方式：POST

输入参数：

参数名	类型	必选	说明
roomId	string	是	当前房间 ID
userId	string	是	用户 ID
event	int	是	事件类型，12 代表成员退出
exitType	int	是	`exitType` 表示退出房间的原因。1：主动离开房间；2：被踢出房间；3：断网超时被移出房间

返回结果

参数名	类型	说明
code	int	10000 代表正常
msg	String	失败消息
data	String	

Request:

```
{
"roomId": "10000",
"userId": "20000",
"event": 12,
"exitType":3
}
```

Response:

```
{
"code": 10000,
"msg": "success",
"data": ""
}
```

提示用户被踢出房间消息示例

```
{
"fromUserId": 0,
"toChatroomId":["24","45"],
"objectName":"RCMic:chrmSeatRemoveMsg",
"content": {
"userId": "10000",
"roomId": "20000",
"timestamp": 56747787
}
}
```

消息参数说明

参数名	类型	说明
roomId	int	房间 ID
userId	int	房主 ID
timestamp	long	移除房间的时间戳

App 客户端处理流程

1. 当用户手机恢复网络连接，会在 IMLib 的消息回调中收到被踢出房间的消息。
2. 前端判断当前用户所在页面是否在语聊房内，并判断该语聊房的 `roomId` 是否与消息体内被踢出房间的 `roomId` 一致。
3. 如果一致，说明该用户此时被踢出房间内。但在 App 客户端上，该用户依然在房间中，所以需要提示用户「已断开连接请重新进入」。
4. 如果不一致，说明用户已退出刚才的房间，不用特殊处理。

换设备登录 概述

更新时间:2023-12-22

换设备登录是指用户在终端设备登录后不执行登出，直接换另一台设备，并再次登录。在这种情况下，如果用户已经进入语聊房内，我们需要保证用户正常离开语聊房，否则可能会发生以下异常行为：

- 用户在设备 A 加入其它语聊房间时报错。此时由于底层 IM 设备互踢机制，用户在设备 A 已被下线，导致不能再加入或离开房间；
- 房主向该用户发送踢出、下麦、邀请等指令时，由于 RCVoiceRoomLib 的代理未实现，导致指令失效。

针对这种场景，融云 RTC 项目通过前端和后端相互配合，制定了一套完善的解决方案，解决方案支持以下两种流程：

- 用户在设备 A 下线，用户在设备 B 恢复加入语聊房；
- 用户在设备 A 下线，用户在设备 B 离开语聊房；

解决方案

根据常见场景业务需求，当用户在换端登录时，如果检测到用户在某个语聊房间时，会恢复或者离开之前房间，确保用户和房间之间的信息同步。为此，我们设计了三个 App 服务端接口，与相应的流程，供您参考：

- 用户登录接口：`user/login` 用于获取用户 `TOKEN`；
- 更新房间接口：`user/change` 更新用户当前所在房间；
- 查询房间接口：`user/check` 查询用户当前所在房间；

本方案以 iOS 来阐述实现流程，安卓和 iOS 的实现方法一致。具体步骤如下：

1. 用户在设备 A 登录，调用 App 服务端接口 `user/login`，本地缓存用户 TOKEN；
2. 用户在设备 A 初始化语聊房，调用 RCVoiceRoomLib SDK 中的 `initWithAppkey:` 和 `connectWithToken:success:error:` 方法；
3. 用户在设备 A 设置连接代理，调用 IMLib SDK 中的 `addConnectionStatusChangeDelegate:` 方法，示例代码：

```
- (void)viewDidLoad {
[super viewDidLoad];
/// 设置 IM 连接状态监听
[[RCCoreClient sharedCoreClient] addConnectionStatusChangeDelegate:self];
}
```

4. 用户在设备 A 加入房间，调用 RCVoiceRoomLib SDK 中的 `joinRoom:success:error:` 方法；

5. 用户在设备 A 向 App 服务端同步用户所在房间，调用 App 服务端接口 user/change，示例代码：

```
- (void)didJoinRoom:(NSString *)roomId {
[self updateCurrentRoom:roomId];
}
/// 更新用户当前所在房间
- (void)updateCurrentRoom:(NSString *)roomId {
[network updateUserCurrentRoom:roomId completion: {}];
}
```

6. 用户在设备 B 登录，设备 A 收到 IMLib SDK 连接状态发生变化 onConnectionStatusChanged 回调，此时，建议清理用户在设备 A 上的信息（TOKEN、用户基本资料等），保护用户的信息隐私。代理方法返回值为枚举类型：连接状态，示例代码：

```
/// 连接状态发生变化
- (void)onConnectionStatusChanged:(RCConnectionStatus)status {
switch (status) {
case ConnectionStatus_KICKED_OFFLINE_BY_OTHER_CLIENT:
/// TODO clear
break;
default:
break;
}
}
```

7. 用户在设备 B 同步房间信息，调用 App 服务端接口 user/check 获取房间信息，示例代码：

```
/// 查询用户当前所在房间
- (void)updateCurrentRoom:(NSString *)roomId {
[network checkUserCurrentRoom:^(VoiceRoom *room) {
if (room) {
/// TODO room restore or clear
}
}];
}
```

8. 用户在设备 B 检测到 App 服务端的房间信息存在，根据需求，一般可以这样做：

- 恢复用户之前加入的房间，确保用户体验一致性。调用 RCVoiceRoomLib 中的 joinRoom:success:error: 方法再次加入该房间，示例代码：

```
/// 恢复房间
- (void)restore:(NSString *)roomId {
[RCVoiceRoomEngine.sharedInstance joinRoom:roomId success:^{
// TODO success
} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {
// TODO error
}];
}
```

- 离开用户之前加入的房间，确保用户能够正常加入其它房间。调用 RCVoiceRoomLib SDK 中的 `leaveRoom:error:` 方法，离开房间。同时，调用 App 服务端接口 `user/change`，清理当前所在房间。示例代码：

```
/// 离开房间
- (void)leave:(NSString *)roomId {
[RCVoiceRoomEngine.sharedInstance leaveRoom:^{
// TODO success
} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {
// TODO error
}];
[network updateUserCurrentRoom:@"" completion: {}];
}
```

详细流程图

客户端相关接口

语聊房客户端 SDK 接口

```
/// 用户根据roomId加入一个语聊房
/// @param roomId 房间id
/// @param successBlock 加入成功回调
/// @param errorBlock 加入失败回调
- (void)joinRoom:(NSString *)roomId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 离开房间
/// @param successBlock 离开房间成功
/// @param errorBlock 离开房间失败
- (void)leaveRoom:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

IM 客户端 SDK 接口

以 iOS IMLib 为例。

```
/*!
添加 IMLib 连接状态监听

@param delegate 代理
*/
- (void)addConnectionStatusChangeDelegate:(id<RCConnectionStatusChangeDelegate>)delegate;
```

App 服务端接口最佳实践

我们在 Demo server 中实现了以下接口，供您参考。

同步用户所在房间

请求地址：`http://xx.com/user/change`

请求方式：`POST`

请求参数：

参数名	类型	必选	说明
roomId	string	是	当前的房间 Id

返回结果

参数名	类型	说明
code	int	10000 代表正常
msg	String	失败消息

查询用户所在房间

请求URL：`http://xx.com/user/check`

请求方式：`GET`

请求参数：无

返回结果

参数名	类型	说明
code	int	10000 代表正常
data	object	房间信息
data.id	int	房间号
data.roomId	string	房间 id
data.roomName	string	房间名称
data.themePictureUrl	string	房间缩略图
data.backgroundUrl	string	房间背景

参数名	类型	说明
data.isPrivate	bool	房间是否私密
data.password	string	房间密码
data.userId	string	房间房主 id
data.updateDt	double	房间更新时间
data.createUser	object	房间创建者信息
data.createUser.userId	string	房间创建者信息 id
data.createUser.userName	string	房间创建者信息名称
data.createUser.portrait	string	房间创建者信息头像
data.roomType	int	房间类型
data.userTotal	int	房间当前用数量
data.stop	bool	房间是否停止

数美审核

更新时间:2023-12-22

音视频场景常常离不开内容审核。融云提供的实时音视频内容审核的能力，可精准识别多场景违法违规内容，高效防御内容风险，提高审核效率。

本文档介绍的方案如下：

1. 融云后台已对接 数美 的音视频内容审核服务。开通音视频审核服务后，您的应用服务器可实时接收音视频审核结果回调。您需要根据回调结果并做进一步处理。
2. 利用数美的 智能文本识别 审核 App 用户昵称是否合规。
3. 利用数美的 智能图片识别 审核 App 用户头像是否合规。

您可以参照本文档和 Demo server 的实现进行适配，快速集成到您自己的 App 服务端。也可以参照融云 实时音视频内容审核 封装您自己 App 的审核功能。

开通服务

融云的音视频审核服务支持将音视频流的审核结果通过服务端回调实时通知您的应用服务器。审核结果一般包含审核任务命中的违规事件，以及审核任务状态实时的通知。

请在融云开发者后台的 IM & 音视频审核 页面开通音视频审核服务，并填写接收审核结果的回调地址。配置后 30 分钟内生效。

配置完成后，审核结果会通过 HTTP 请求实时回调您的服务器。

音视频审核

对 App 用户在使用 App 服务时产生的音视频流进行审核，可过滤掉不良语音内容和违规视频内容。

基本流程

1. 多位 App 用户在房间内发布音视频流，融云音视频审核服务对其进行审核。
2. 某个音视频流触发违规事件，融云向 App 服务端发送审核结果回调。
3. App 服务端接收并解析回调结果，得到违规事件的类型、房间、用户等数据。可参见实时音视频内容审核中的回调字段说明。
4. App 服务端提示并记录违规事件到数据库。
5. 发送消息提醒用户违规。根据违规事件级别，可用户进行冻结。

内容审核流程图

音视频审核回调接口

您需要在 App 服务端实现一个解析融云音视频审核结果回调的接口，例如根据审核结果进行消息提示、冻结用户等处理。

> 以下接口是我们在 Demo 和 Demo server 中的实现。

请求地址：`http://localhost:8080/mic/callback/audit`

请求方式：POST

具体代码：

```
public RestResult auditSync(AuditCallBackDto dto) {
log.info("audit sync dto:{}",dto);
// 保存到数据库 只保存有问题的
if(dto==null||dto.getType().intValue()==
AuditTypeEnum.AUDIT_START.getValue()||dto.getType().intValue()==AuditTypeEnum.AUDIT_STOP.getValue()){
return RestResult.success();
}

// 将有问题的用户 冻结
if(AuditTypeEnum.AUDIT_CALLBACK.getValue()==dto.getType().intValue()) {
AuditCallBackContentDto contentDto = dto.getContent();
Integer riskLevel = contentDto.getRiskLevel();
if (riskLevel.intValue() ==
RiskLevelEnum.PASS.getValue()||riskLevel.intValue()==RiskLevelEnum.REVIEW.getValue()) {
return RestResult.success();
}
try {
String uid = "_SYSTEM_";
String userId = dto.getUserId();
ShumeiAuditFreezeMessage message = new ShumeiAuditFreezeMessage();
message.setUserId(userId);
boolean go = SlideWindowUtil.isGo(userId, 3, 1000 * 60 * 5L);
if (!go) {
SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日HH时mm分ss秒");
Date freezeTime = new Date();

freezeTime = DateUtils.addMinutes(freezeTime, 5);
log.info("audit sync freezeTime:{}", sdf.format(freezeTime));

userService.freezeUser(userId, UserStatusEnum.FREEZE.getValue(), freezeTime);
//发消息
message.setStatus(2);
message.setMessage(String.format(ShumeiKey.FREEZE_MESSAGE, sdf.format(freezeTime)));
imHelper.publishSysMessage(uid, Lists.newArrayList(userId), message);

}else{
// 如果没有达到冻结标准，那么发送消息，App 端谈提示消息
message.setStatus(1);
message.setMessage(contentDto.getDesc());
imHelper.publishSysMessage(uid, Lists.newArrayList(userId), message);
}
ShumeiAudit shumeiAudit = new ShumeiAudit();
BeanUtils.copyProperties(dto,shumeiAudit);
shumeiAudit.setUid(IdentifierUtils.uuid());
shumeiAudit.setContent(JSON.toJSONString(dto.getContent()));
shumeiAuditService.save(shumeiAudit);
} catch (Exception e) {
log.error("audit sync error message:{}",e.getMessage(),e);
}

}
return RestResult.success();
}
```

用户昵称与头像审核

在融云开发者后台开通音视频审核服务后，您将收到一封带有数美账号密码的邮件。您可以利用邮件中提供的数美凭证使用数美的 智能文本识别 与 智能图片识别 服务，用于审核 App 用户昵称与头像是否合规。

审核用户昵称

利用数美的 智能文本识别 审核 App 用户昵称是否合规，并同步返回审核结果。如果违规,可以直接返回到用户操作界面。

> 以下接口是我们在 Demo 和 Demo server 中的实现。

```
public ShumeiResp textAudit(String text) {
if(StringUtils.isBlank(text)){
return null;
}
HashMap<String, Object> payload = new HashMap<String, Object>();
payload.put("eventId", "nickname");
payload.put("type", "ALL");

HashMap<String, Object> data = new HashMap<String, Object>();
data.put("text", text);

return execute(payload,data,textUrl);
}

private ShumeiResp execute(HashMap<String, Object> payload,HashMap<String, Object> data,String url){
data.put("tokenId", UUID.randomUUID());
payload.put("accessKey", ACCESS_KEY);
payload.put("appId", "default");
payload.put("data", data);
JSONObject json = JSONUtil.parseObj(payload);
JSONObject result = ShumeiUtils.httpPost(url, json);
ShumeiResp shumeiResp = JSONUtil.toBean(result, ShumeiResp.class);
log.info("shumei audit url:{} param:{}, result:{}",url,json,shumeiResp);
return shumeiResp;
}
```

审核用户头像

利用数美的智能图片识别 服务审核 App 用户昵称与头像是否合规，并同步返回审核结果。如果违规,可以直接返回到用户操作界面。

> 以下接口是我们在 Demo 和 Demo server 中的实现。

```
public ShumeiResp imageAudit(String url){
if(StringUtils.isBlank(url)){
return null;
}
HashMap<String, Object> payload = new HashMap<String, Object>();
payload.put("type", "POLITICS_PORN_OCR_AD_LOGO_MINOR_SCREEN_SCENCE_QR_VIOLENCE_BAN");
payload.put("eventId","headImage");

HashMap<String, Object> data = new HashMap<String, Object>();
data.put("img", url);

return execute(payload,data,imageUrl);
}

private ShumeiResp execute(HashMap<String, Object> payload,HashMap<String, Object> data,String url){
data.put("tokenId", UUID.randomUUID());
payload.put("accessKey", ACCESS_KEY);
payload.put("appId", "default");
payload.put("data", data);
JSONObject json = JSONUtil.parseObj(payload);
JSONObject result = ShumeiUtils.httpPost(url, json);
ShumeiResp shumeiResp = JSONUtil.toBean(result, ShumeiResp.class);
log.info("shumei audit url:{} param:{}, result:{}",url,json,shumeiResp);
return shumeiResp;
}
```

使用 Demo Server 项目概述

更新时间:2023-12-22

rongcloud-scene-server-bestcase 是融云融合场景服务端的 Java 项目。项目代码已在 Github 上开源。

Github 地址：https://github.com/rongcloud/rongcloud-scene-server-bestcase

该项目基于融云的"即时通信" 和 "实时音视频" 能力，封装了一些最佳实践，可以满足大部分需求场景。

主要功能模块有：

- 房间管理
- 麦位管理
- PK 场景

技术介绍

- 该项目基于 SpringBoot 框架实现
- 依赖于 Mysql 数据存储、Redis 数据缓存 、flyway 数据库版本控制
- 依赖于融云 IM 服务收发信令、依赖于融云短信服务收发短信

项目组件

组件	版本	说明
Mysql	5.6+	业务数据，存储用户、房间、音乐、APP 版本等信息
Redis	4.0+	缓存数据库，缓存用户、房间、排麦用户、麦位等频繁访问的信息
Jdk	1.8	Java 版本
Nginx	1.13 +	负载，也可替换为其他负载

项目运行

准备工作

请提前准备好项目所需的软件服务。如 JDK、MySQL、Redis、Flyway 等。

1. 克隆项目

```
git clone https://github.com/rongcloud/rongcloud-scene-server-bestcase.git
```

2. 服务配置

服务用的配置信息统一在 `application.yml` 配置文件中。

请前往 融云官网 注册、申请 AppKey 和 Secret, 并替换 application.yml 文件中的融云 IM 配置，配置如下:

```
# 融云 IM 服务配置
im:
appKey: {您的appKey}
secret: {您的secret}
host: http://api.rong-api.com
```

rongcloud-scene-server-bestcase 登录默认需短信验证，登录是否需短信验证可通过开关配置，配置项在 application.yml 文件中, 配置如下:

```
rongrtc:
login:
sms_verify: false
```

如果您不需要登录短信验证，请直接跳转下一步操作。如您需要，请前往 融云官网 开通短信服务，并注册登录验证短信模板 Id，然后将 AppKey、Secret 和 TemplateId 替换 application.yml 文件中的 融云 SMS 配置，配置如下:

```
# 融云 SMS服务配置
sms:
appKey: {您的appKey}
secret: {您的secret}
host: http://sms.rong-api.com
templateId: {您的templateId}
```

短信模板示例:

```
【融云全球通信云】您的短信验证码是 xxxx，请在 15 分钟内输入使用。超时请重新申请。
```

请您将实际数据库连接配置替换 application.yml 文件中的 数据库连接配置，配置如下：

```
# 数据库连接配置
spring:
datasource:
url: jdbc:mysql://127.0.0.1:3306/您的数据
库?useSSL=false&serverTimezone=Asia/Shanghai&characterEncoding=utf8
username: (您的用户名)
password: (您的密码)
driver-class-name: com.mysql.cj.jdbc.Driver
```

请您将实际 Redis 连接配置替换 application.yml 文件中的 Redis 连接配置，配置如下：

```
# Redis 连接配置
redis:
database: 0
host: 127.0.0.1
port: 6379
pass:
maxIdle: 100
maxTotal: 300
testOnBorrow: true
```

请将 rongcloud-scene-server-bestcase 服务访问地址配置替换 application.yml 文件中的 服务访问地址配置，配置如下：

```
# 访问地址配置
rongrtc:
domain: http://120.92.13.89 # rongcloud-scene-server-bestcase 地址，协议://域名:端口
```

假设服务启动端口为 8080，则该地址需要配: http://127.0.0.1:8080

数据库初始化

执行如下命令，创建数据库:

```
create database rongrtc;
```

服务打包运行

通过 mvn package 编译出 jar 或者 IntelliJ IDE 直接运行工程

- java -jar -Dspring.profiles.active=prod rongcloud-scene-server-bestcase-1.0-SNAPSHOT.jar 启动服务
- 默认启用 8080 端口，默认 http 请求

概述 实时社区描述

更新时间:2023-12-22

融云实时社区产品的主要功能基于即时通讯（IM）超级群业务。实时社区是一种新型的社交形态，用户可以建立或者加入自己感兴趣的社区，每个社区下可以创建独立的频道，频道中的消息独立，互不影响，并且每个社区对于人数不设定上限。

实时社区产品已提供客户端（Android、iOS）与应用服务端 Demo。

业务概述

融云在 RCRTC 项目（GitHub · Gitee）以模块的形式包含了针对实时社区场景的 App Server 端的简单实现，已提供社区管理、社区用户管理、分组和频道管理、消息管理功能。

客户端用户登陆 App 后可以发起创建实时社区请求。由 App Server 创建实时社区后，默认生成两个频道组，每个频道分组下默认生成两个频道。其中一个频道是默认用来发送消息和接收消息的。

创建完社区后，用户可以在发现页看到该社区，可加入或申请加入该社区。如果社区设置为审核后才能加入，那么社区的创建者会收到一个审核消息，可以同意或拒绝该用户加入。加入者可以修改在该社区的用户昵称等一些信息，也可以标注某一条消息。更多玩法可以去官网下载 APP 体验。

具体请求参考服务端实时社区模块代码实现。

整体流程

社区管理

更新时间:2023-12-22

社区管理主要包含了社区的创建、查询、修改、解散功能。

用户可以自己创建社区。已创建的社区可以在客户端的发现页中找到，其他人可以加入社区，并在默认的频道下发消息。社区创建者也可以修改社区的基本信息，也可以改变加入社区是否需要审核,还可以长按分组或频道，改变分组或频道的顺序或改变名称等。

创建社区的时序图

接口设计参考

功能	Demo Server 接口	IM 服务端核心 API
创建社区	`/mic/community/save`	创建超级群
修改社区	`/mic/community/update`	修改超级群
解散社区	`/mic/community/delete/{communityUid}`	解散超级群
社区详情	`/mic/community/detail/{communityUid}`	不涉及 IM 服务端核心 API。
整体保存	`/mic/community/saveAll`	不涉及 IM 服务端核心 API。社区的创建者可以调整分组顺序、频道顺序、分组名称、频道名称。调整之后，客户端需要调用整体保存接口。
分页查询	`/mic/community/page`	不涉及 IM 服务端核心 API。用于实现客户端的发现页。

Demo Server 项目提供了已实现接口的 API 文档。详见 GitHub · Gitee 。

社区用户管理

更新时间:2023-12-22

社区用户管理主要从两个维度考虑：

- **社区创建者**：社区创建者可以修改用户的社区昵称、用户禁言、用户封禁、审核通过/不通过、踢出、在线/离线/禁言/封禁用户列表查询。
- **社区加入者**：加入社区、修改用户昵称、修改通知类型设置、退出、修改频道通知设置、查询已加入社区、标记频道消息、删除已标记、标记消息列表。

社区用户时序图

接口设计参考

功能	Demo Server 接口	IM 服务端核心 API
加入社区	`/mic/community/user/join/{communityUid}`	加入超级群
社区用户修改	`/mic/community/user/update`	不涉及 IM 服务端核心 API。这个接口实现了多个功能，包含：用户昵修改、群通知设置、封禁用户、禁言用户、审核、退出、被踢出等
用户修改频道设置	`/mic/community/user/update/channel`	不涉及 IM 服务端核心 API
在线/离线/禁言/封禁 用户列表	`/mic/community/user/page`	在线和离线的判定标准由 IM 用户在线状态回调实现。依赖用户在线状态回调
用户已加入社区列表	`/mic/community/user/pageCommunity`	不涉及 IM 服务端核心 API
添加频道标记消息	`/mic/channel/message/save`	不涉及 IM 服务端核心 API
分页查询频道标记消息	`/mic/channel/message/page`	不涉及 IM 服务端核心 API
删除频道标记消息	`/mic/channel/message/delete/{messageUid}`	不涉及 IM 服务端核心 API
标记消息详情	`/mic/channel/message/detail/{messageUid}`	不涉及 IM 服务端核心 API

Demo Server 项目提供了已实现接口的 API 文档。详见 GitHub · Gitee 。

分组和频道管理

更新时间:2023-12-22

分组和频道管理主要包含了分组的创建、修改、删除、频道的创建、修改、删除、详情。这个模块比较简单。需要注意的就是分组是业务上的概念，不需要同步给 IM，社区和频道在 IM 上是有对应关系的，所以创建社区和频道的时候，需要同步到 IM。

接口设计参考

下表列出了 Demo Server 中已实现的实时社区相关接口，以及涉及的 IM 服务端核心 API。如果您希望自行实现，可参考 Demo Server 中的代码与相关的 IM 服务端 API 文档。

功能	Demo Server 接口	IM 服务端核心 API
创建分组	`/mic/group/save`	不涉及 IM 服务端核心 API
修改分组	`/mic/group/update`	不涉及 IM 服务端核心 API
删除分组	`/mic/group/delete/{groupUid}`	不涉及 IM 服务端核心 API
创建频道	`/mic/channel/save`	创建频道
修改频道	`/mic/channel/update`	不涉及 IM 服务端核心 API
删除频道	`/mic/channel/delete/{channelUid}`	删除频道
频道详情	`/mic/channel/detail/{channelUid}`	不涉及 IM 服务端核心 API

Demo Server 项目提供了已实现接口的 API 文档。详见 GitHub · Gitee 。

消息管理

更新时间:2023-12-22

实时社区业务要求将业务中的变化及时通知客户端。例如，社区创建者 A 创建的社区中已有用户 B 加入，并在社区中发消息。此时用户 A 如果执行解散社区、删除社区的频道分组或频道，App 业务应保证用户 B 可实时感知该变化，并执行对应操作。

在实时社区中，此类业务变化可通过 App Server 进行通知。您可以使用传统方式实现，比如使用 websocket 与 client 保持长链接进行通信。我们推荐通过集成融云的 IM 业务进行处理。在需要通知时，由 App Server 向 client 发一条自定义消息进行通知。

以下文档描述了实时社区 Demo 中实现的自定义消息。

社区变化通知消息

修改社区、修改分组、修改频道、新增分组、新增频道、删除分组、删除频道 时都会发送消息，端上收到消息后要刷新社区详情，发送的自定义消息的消息题内容如下：

```
{
"fromUserId": 0,
"toGroupIds":["24","45"],
"objectName":"RCMic:CommunityChange",
"content": {
"fromUserId":"用户id",
"communityUid": "1234567",
"message":"频道被删除",
"channelUids":[
"频道1","频道2"
]
}
}
```

代码位置：CommunityMessageUtil.sendCommunityChangeMessage();

社区通知类消息

申请加入社区消息、加入社区、退出社区、被踢出，这类消息是展示在系统通知里面的。

```
{
"fromUserId": 0,
"toUserId":["24","45"],
"objectName":"RCMic:CommunitySysNotice",
"content": {
"fromUserId":"用户id",
"communityUid": "1234567",
"message":"张三申请加入 A社区",
"type":"0代表申请加入社区、1代表加入社区后通知消息、2代表退出社区的通知消息，3代表被踢出社区的通知消息,4代表被禁言，5代表解除禁言，6代表被拒绝加入,7代表解散社区"
}
}
```

代码位置：CommunityMessageUtil.sendCommunityNoticeMessage();

频道通知类消息

频道通知消息,这类消息是展示在某个社区下的默认频道里面的。

```
{
"fromUserId": 0,
"toGroupIds":["24","45"],
"objectName":"RCMic:ChannelNotice",
"content": {
"fromUserId":"用户uid",
"communityUid": "1234567",
"channelUid":"频道id,type=2的时候使用",
"message":"XX 加入XX频道",
"type":"1代表加入消息，2代表标记消息，3代表被禁言，4代表解除禁言,5删除标记消息,6主动退出社区"
}
}
```

代码位置：CommunityMessageUtil.sendChannelNoticeMessage();

用户变更通知消息

用户修改用户昵称和头像发送社区消息，客户端缓存了一些用户的头像和昵称，用来在社区中实时展示，所以如果用户修改了昵称或头像，需要更新客户端的本地缓存。

```
{
"fromUserId": 0,
"toGroupIds":["24","45"],
"objectName":"RCMic:UserUpdate",
"content": {
"portrait":"用户头像",
"nickName": "用户社区昵称",
"userId":"用户id",
"type":"1代表修改用户头像，2代表修改用户社区昵称"
}
}
```

代码位置：CommunityMessageUtil.sendUserUpdateMessage();

Demo Server 项目提供了已实现接口的 API 文档。详见 GitHub · Gitee 。