Tencent Smart Advisor-Tencent RTC Copilot

시나리오 기반 솔루션

제품 문서

목록:

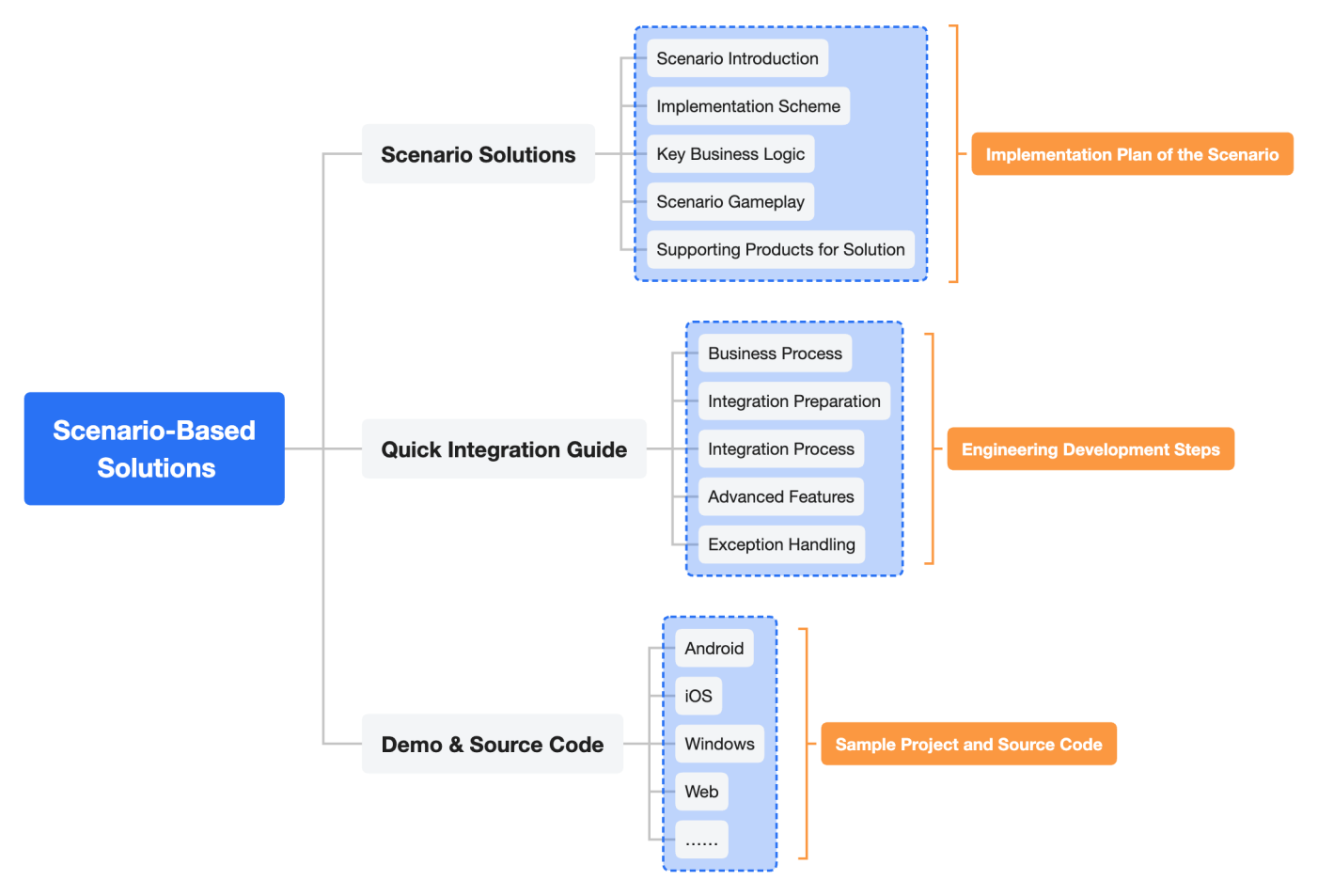
Web

시나리오 기반 솔루션
Overview of Scenario-Based Solutions

최종 업데이트 날짜:: 2024-07-22 14:26:43

Overview

The scenario-based solutions systematically summarize the scenario implementation scheme and engineering development steps to help you quickly familiarize yourself with the relevant scenarios and complete the TRTC integration and launch work. Covers voice chat room, online karaoke, live show streaming, and other social entertainment scenarios, and live streaming marketing and 1V1 audio and video call, and other scenario solutions and quick integration guide. It can help improve integration development efficiency and facilitate the rapid implementation of business scenarios.



Project Planning Phase

At this stage, you can see the experience demo and scenario-based solutions we provided to complete the project planning.

- Experience Demo: Quickly get started with related scenarios and gameplay and experience the project

implementation effect.

- Scenario Solution: Be familiar with the relevant scenarios, functional modules, and technical architecture, and finalize the project implementation scheme.

Project Development Phase

At this stage, you can see the demo source code and Quick Integration Guide we provided to complete project development.

- Demo Source Code: Provides complete engineering source code for the relevant scenarios. You can refer to the code implementation of related modules.
- Quick Integration Guide: Provides full-process integration guides for the relevant scenarios to help you quickly complete project development.

소셜 엔터테인먼트 음성 채팅방 시나리오의 솔루션

최종 업데이트 날짜:: 2025-10-28 11:32:36

시나리오 소개

음성 채팅방은 순수 오디오 방식으로 온라인 인터랙티브 소셜을 하는 가상 공간을 의미합니다.방 내에는 일반적으로 여러 개의 마이크가 설치되어 있습니다. 스트리머와 마이크로 연결된 청취자들이 마이크에서 체팅을 나누고 다른 청취자들은 방에 들어가서 청취할 수 있습니다. 유형이 다른 방은 마이크 설치 수량과 최대 수용 가능 청취자 인원수이 다릅니다. RTC Engine는 최대 50명이 동시에 마이크에서 채팅할 수 있도록 지원하며 마이크의 연결/해제를 원활하게 전환하며 음성 채팅 지연 시간이 300ms 미만입니다. 변성, 분위기 음향, 리버브 등 다양한 오디오 효과를 지원하여 음성 채팅 체험을 더욱 풍부하게 합니다. Chat과 결합하여 공개 채팅, 개인 채팅, 그룹 채팅, 좋아요, 선물 보내기 등 다양한 메시지 인터랙티브 형태를 지원하여 우수한 채팅 체험을 만들어줍니다.

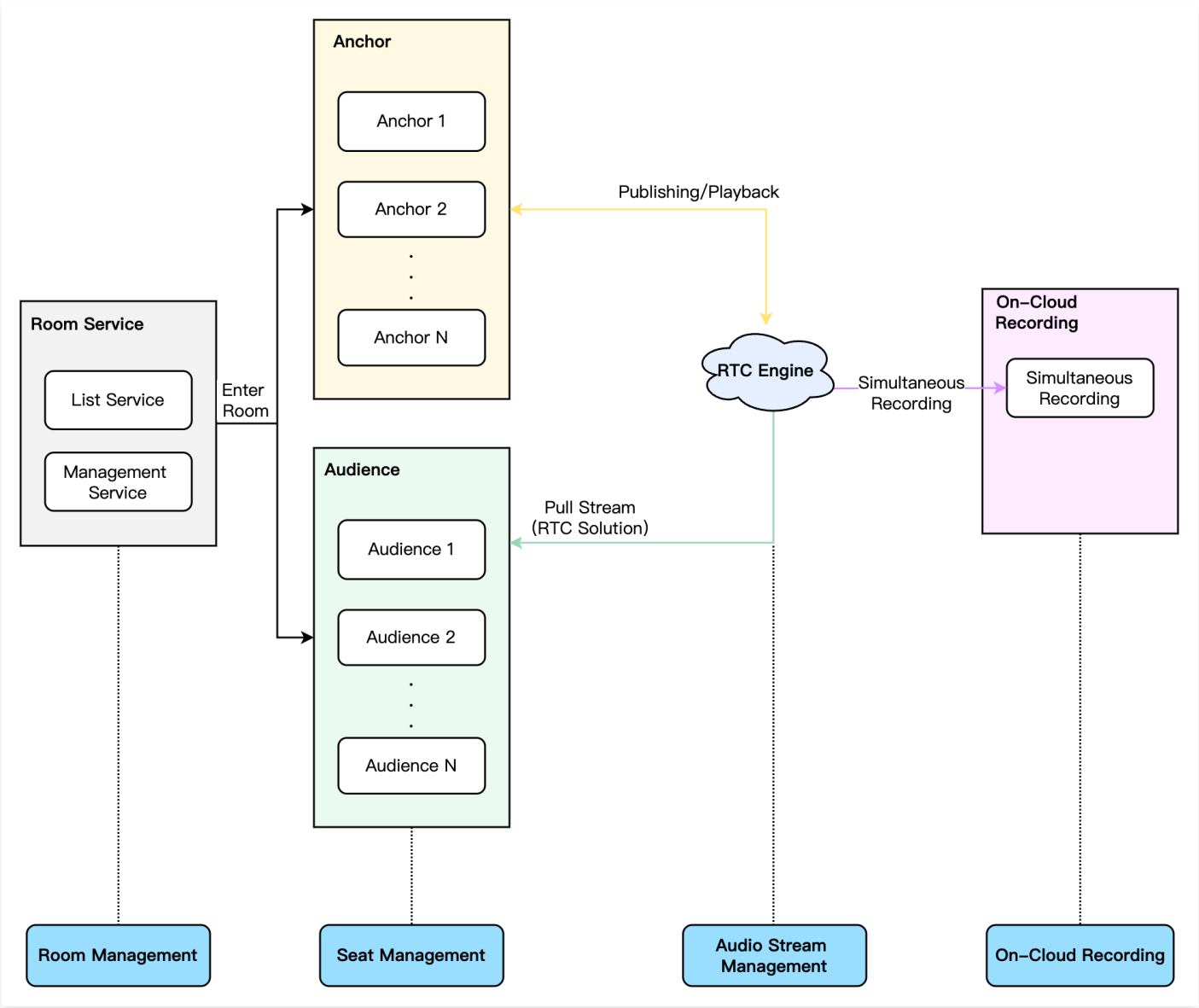
구현 방안

일반적으로 완전한 음성 채팅방 시나리오를 구현하려면 여러 기능 모듈이 필요합니다.방 관리, 마이크 관리, 오디오 스트림 관리, 클라우드 레코딩 등 있습니다.각 기능 모듈의 주요 작업 및 기능 포인트는 아래 표와 같습니다.

기능 모듈	주요 업무 및 기능 포인트
방 관리	방 목록, 방 생성, 방 입장, 방 나가기, 방 파기
마이크 관리	마이크 자동 사용, 마이크 사용 초청, 마이크 끄기, 마이크의 사용 강제 끄기, 마이크 금언, 마이크 잠금, 마이크 이동
오디오 스트림 관리	푸시-풀 스트림 아키텍처 솔루션, 실시간 스트림 구독 모드

클라우드 레코딩	RTC Engine 클라우드 레코딩

음성 채팅방 시나리오의 전체 업무 아키텍처는 아래 그림과 같습니다. 방주인이 음성 채팅방을 생성하면 사용자는 관심 있는 방을 선택해서 입장할 수 있습니다. 방에 입장한 사용자는 마이크를 켜고 사용할 수 있으며 스트리머와 채팅할 수 있으며 규정 준수를 위해 방 내 음성 내용은 녹음되어 심사됩니다.



방 관리

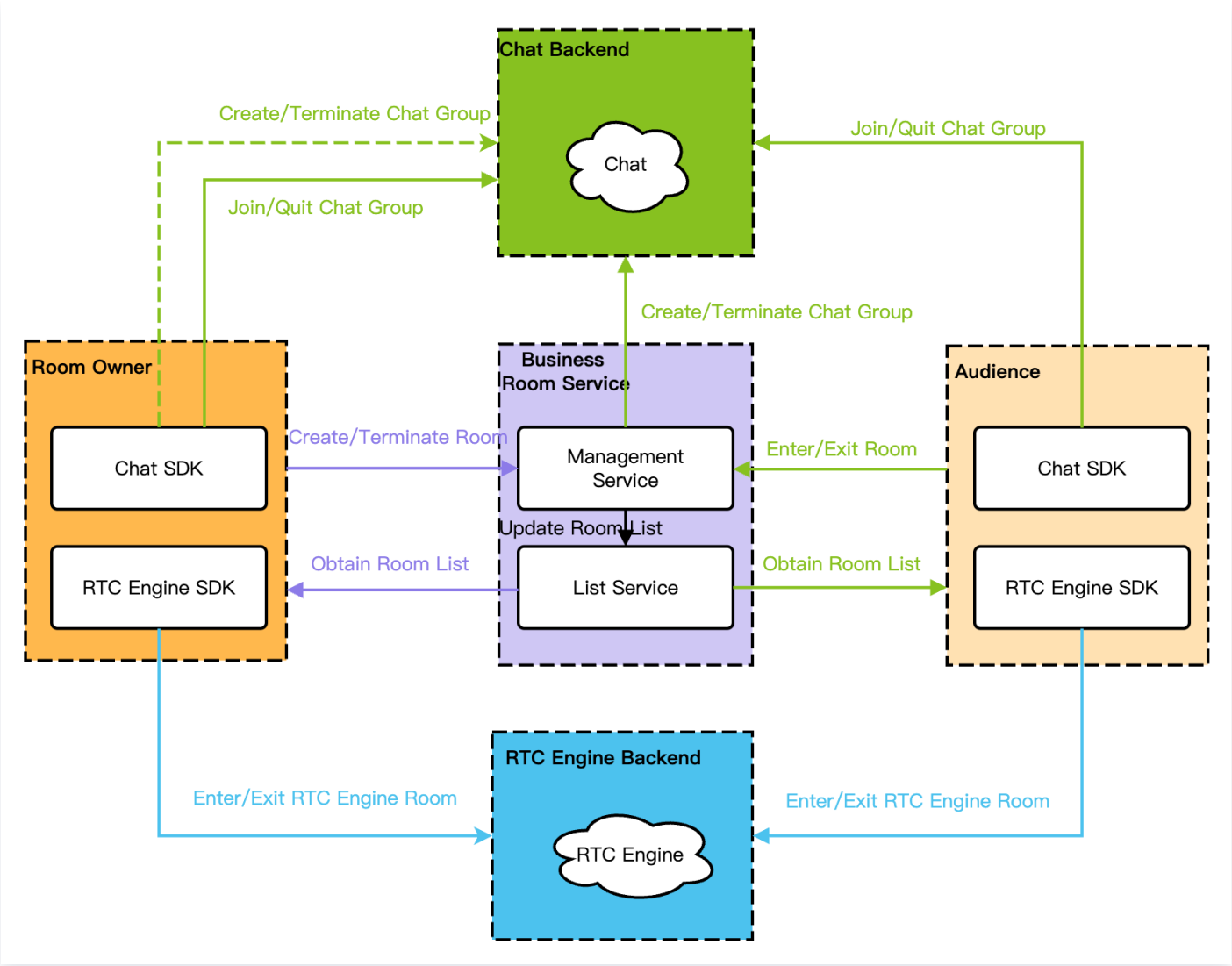
방 관리 모듈은 주로 방 목록 유지 관리를 담당하며 주요 기능은 다음과 같습니다.

- 방 생성: 사용자가 업무 시스템에 로그인한 후 방을 생성할 수 있으며 방 생성 후 방 목록에 추가 작업이 이루어집니다.
- 방 입장: 사용자는 기존 방에 입장할 수 있으며, 방 입장 후 해당 방 인원 목록에 추가 작업이 이루어집니다.
- 방 나가기: 사용자는 현재 방에서 나갈 수 있으며, 방 나간 후 해당 방 인원 목록에서 삭제 작업이 이루어집니다.
- 방 파기: 모든 사용자가 방에서 나간 후 방을 파기해야 하며, 방 파기 후 방 목록에서 삭제 작업이 이루어집니다.

솔루션 아키텍처

전체 방 관리의 아키텍처에서는 주로 세 가지 모듈의 방 관리가 포함됩니다.

- 업무 측 방 관리: 주로 방 목록의 유지 및 관리를 위해 사용되며 예를 들어 업무 방의 속성과 상태를 동기화하는 것입니다. 기능으로는 방 목록 조회, 방 입장 및 퇴장, 방 생성 및 파기가 있습니다.
- 채팅 그룹 관리: 주로 방 멤버 목록, 시그널링의 송부및 수신, 메시지 인터랙티브에 사용되며 예를 들어 마이크 사용 신청의 허락/거부, 마이크 사용 초청/취소, 마이크 무음/해제, 마이크 사용 금지/해제 등이 있습니다.또한 그룹 차원으로 구분되어 있으며 그룹 생성, 그룹 가입, 그룹 퇴장, 그룹 파기 등의 기능이 있습니다.
- RTC Engine 방 관리: 주로 오디오 스트림의 인터랙티브와 전송을 위해 사용되며 예를 들어 스트리머/청취자의 음성/음악의 송부및 수신등 있습니다. 또한 방 차원으로 구분되며 RTC Engine 방 입장 및 퇴장 기능이 있습니다.



구체적인 구현

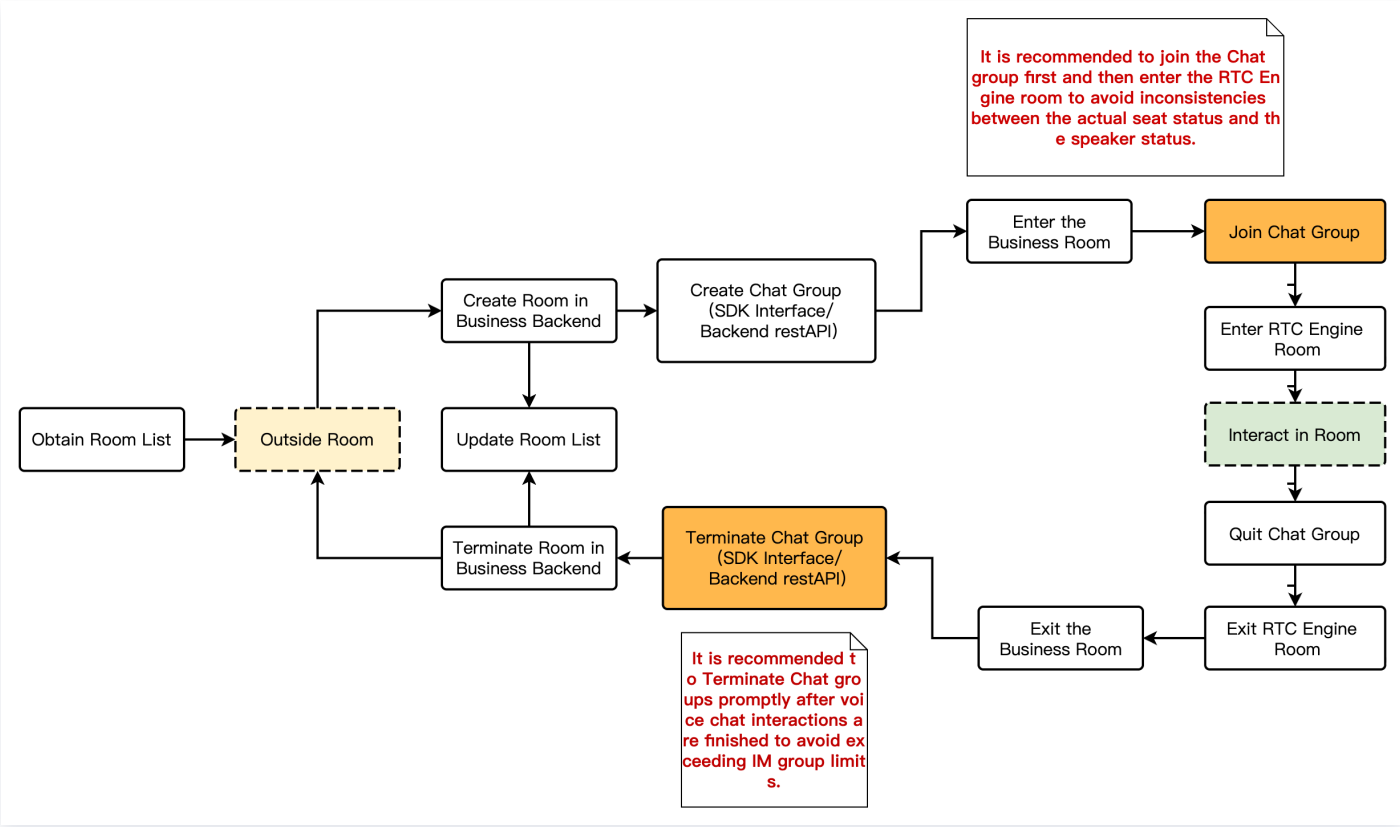
방 관리에서 서로 다른 사용자 역할은 각자의 권한 및 구현 프로세스를 가집니다. 음성 채팅방에서는 주로 방주인과 청취자 두 가지 역할이 존재하며 역할 설명 및 차이점은 아래 표를 참조하십시오.

역할	설명	차이점
방	방의 최고 권한을 가진 소유자는 방을 생성하	• 역할은 반드시 스트리머여야 합니다.

주인	거나 파기할 수 있습니다.	• 업무 룸/채팅 그룹/RTC 룸의 생성 또는 폐기
청취자	방의 참여자도 마이크를 켜서 사용하여 스트리머가 될 수 있습니다.	• 역할은 청취자/스트리머일 수 있습니다. • 방 입장및 퇴장

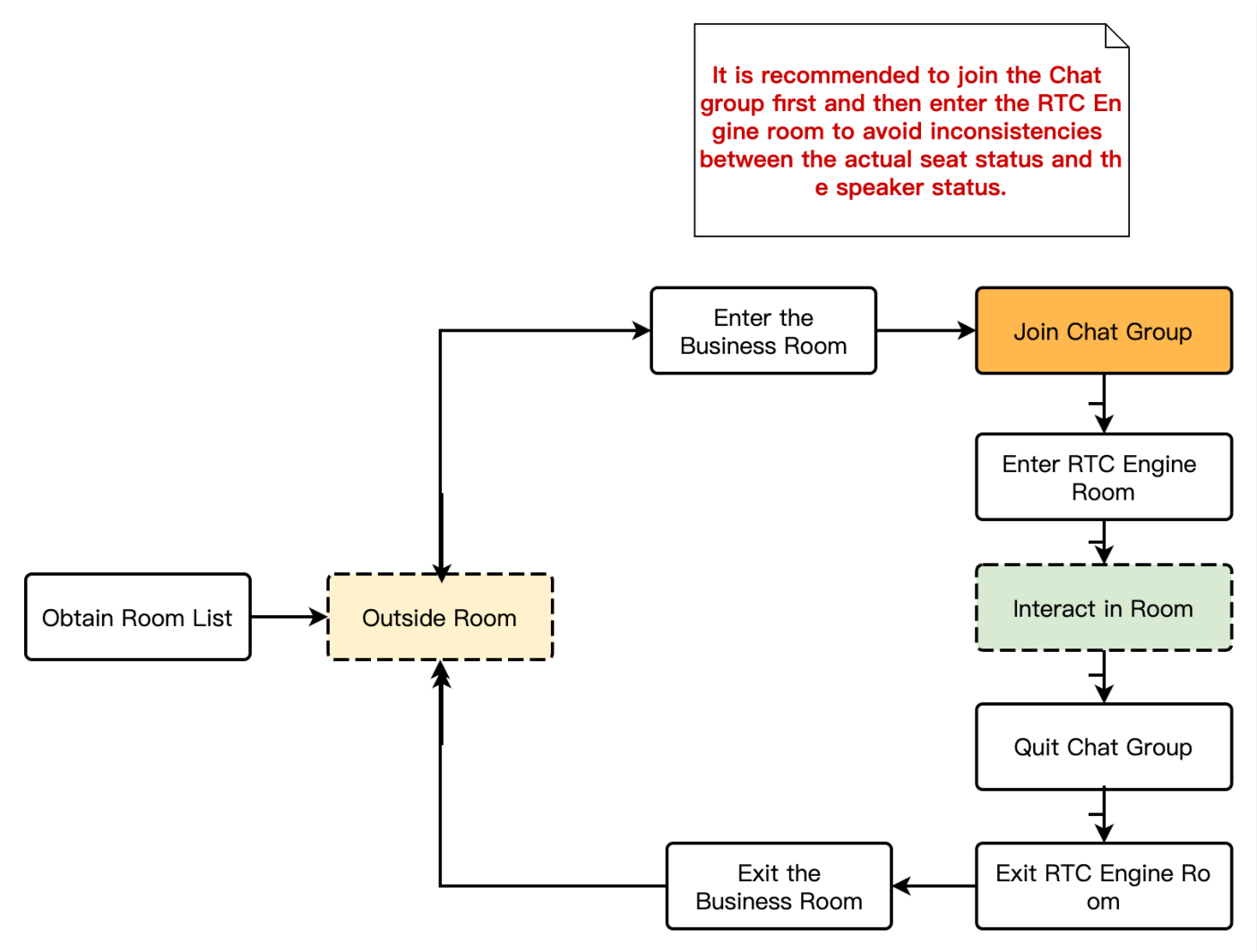
구현 프로세스

- **방주인**



1. 방 목록 가져오기
2. 업무 인터페이스를 통해 해당 방을 생성합니다.
3. Chat 그룹을 생성합니다.
4. 업무 방/Chat 방/RTC 방에 들어가서 다른 사람들과 인터랙티브합시다
5. Chat 그룹/RTC 방/업무 방에서 나갑니다.
6. Chat 그룹을 파기합니다.

- **청취자**



1. 방 목록 가져오기
2. 업무 방/Chat 그룹/RTC 방에 들어가서 다른 사람들과 인터랙티브합시다
3. Chat 그룹/RTC 방/업무 방에서 나갑니다.

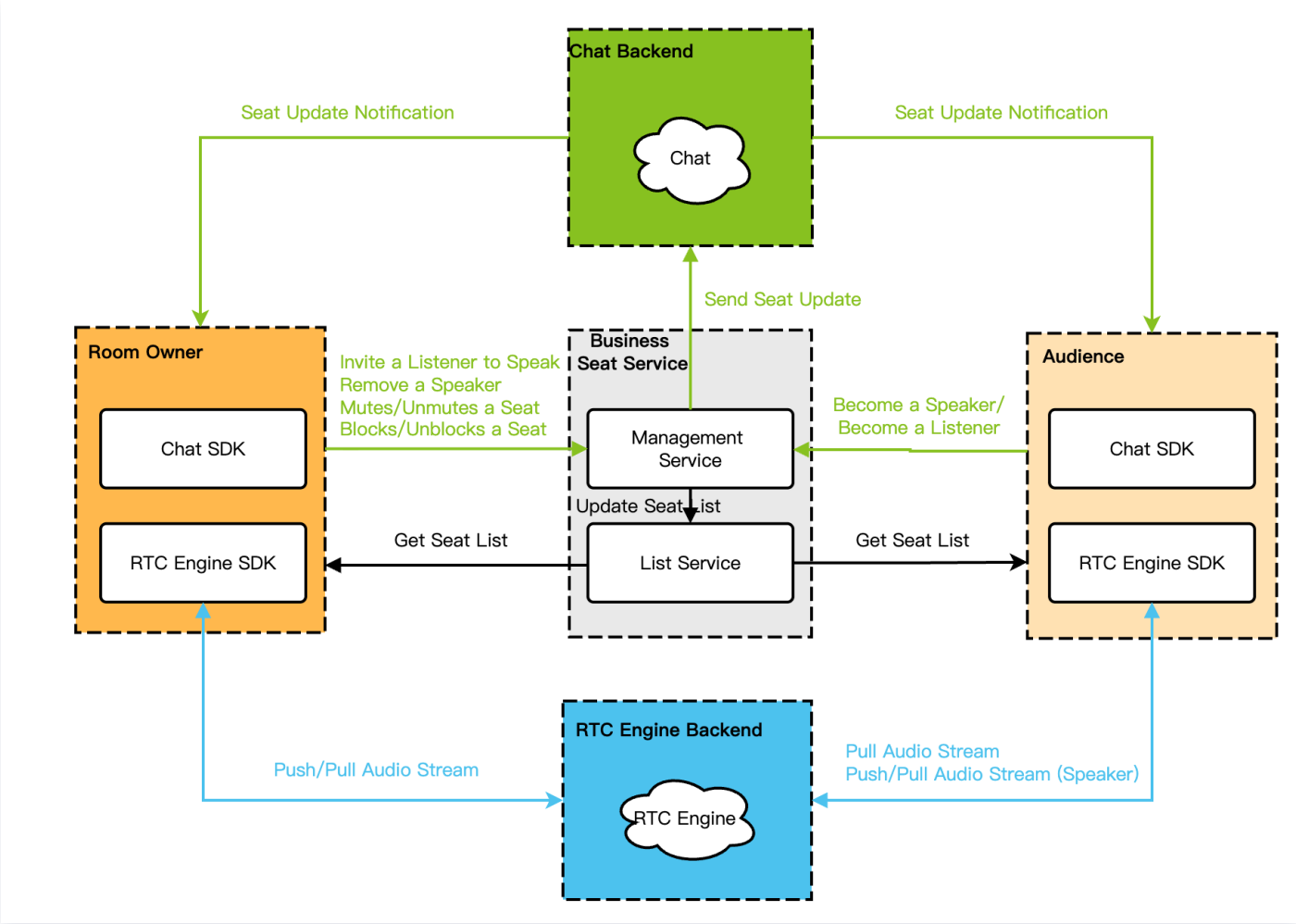
마이크 관리

음성 채팅방의 마이크는 일반적으로 순서가 정해져 있고 제한적입니다. 예를 들어 방 내의 청취자가 마이크를 사용하려면 방주인의 동의를 받아 순서대로 마이크를 사용합니다. 방 내 마이크 수는 일반적으로 10개를 넘지 않습니다. 마이크 관리는 주로 업무 시나리오에 따라 방 내 마이크 수를 정의하고 현재 방의 모든 마이크 상태를 관리하는 역할을 합니다. 마이크 관리에 자동 마이크 사용, 마이크 사용 초청, 자동 마이크 끄기, 마이크의 사용 강제 끄기, 마이크 금언, 마이크 잠금, 마이크 이동 등 기능이 포함됩니다

- 사용자가 방에 입장한 후, 빈 상태의 마이크가 있을 때만 사용자가 마이크 사용을 신청할 수 있습니다.
- 방주인이 사용자의 마이크 사용 신청을 허락한 후, 마이크 상태를 비어 있지 않음으로 수정해야 합니다.
- 사용자가 스트리밍을 중지하고 마이크의 사용을 끈 후, 마이크 상태를 재설정해야 합니다.
- 방주인은 마이크 잠금, 마이크 사용 초청, 강제로 마이크 끄기, 마이크 금언 등의 권한을 가집니다.

솔루션 아키텍처

아래에서는 RTC Engine과 Chat을 결합하여 마이크 관리의 솔루션 아키텍처를 구성합니다. 전체 방 관리 아키텍처에서 방주인은 최고의 권한을 가지며 마이크 사용 초청/강제로 마이크 끄기/마이크 무음및 해제/마이크 금언및 해제를 할 수 있습니다. 청취자들은 마이크의 사용을 신청을 통해 스트리머가 되어 방 내 다른 스트리머들과 인터랙티브할 수 있습니다.



구체적인 구현

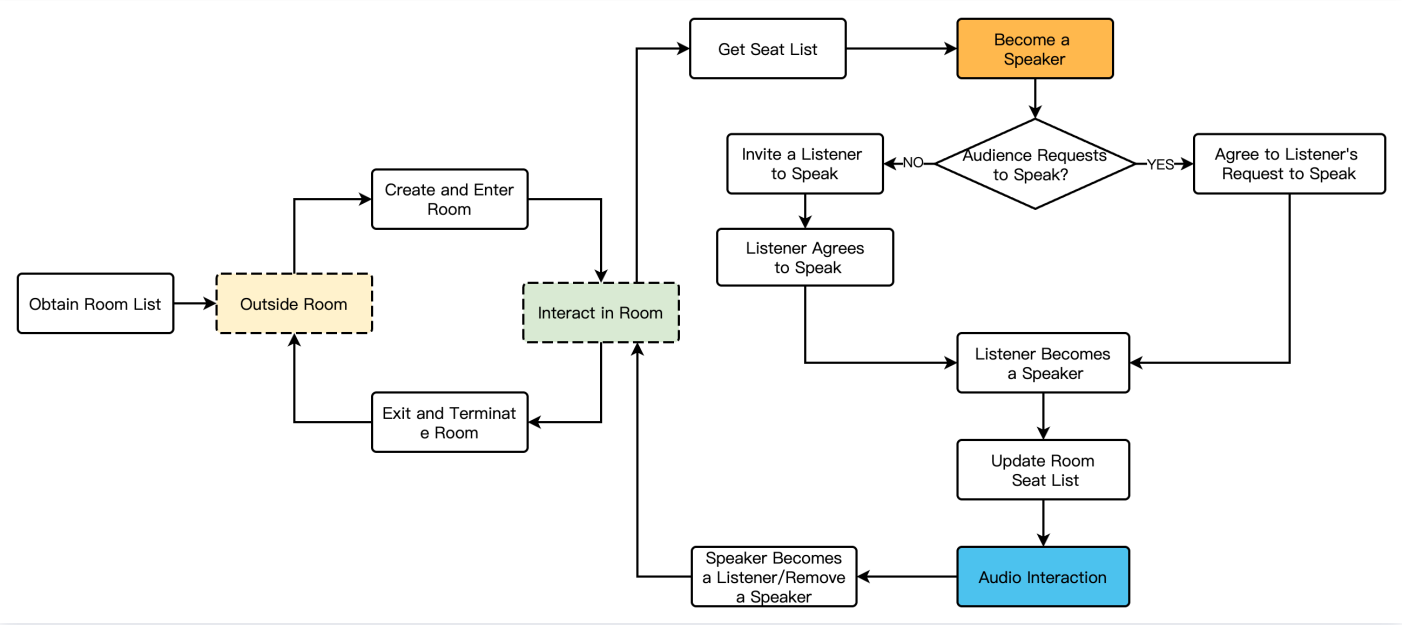
마이크 관리에서 서로 다른 사용자 역할은 각자의 권한 및 구현 프로세스를 가지며, 주로 방주인과 청취자 두 가지 역할이 존재합니다. 역할 설명 및 차이점은 아래 표를 참조하십시오.

역할	설명	차이점

방주인	방주인은 마이크 관리의 최고 권한자이고 모든 마이크를 관리합니다. 방주인이 방을 나가면 모든 마이크가 자동으로 해산됩니다.	• 역할은 반드시 스트리머여야 합니다. • 방 입장 시 자동으로 마이크에 연결됩니다 • 마이크 신청의 수락/거절 • 마이크 사용의 초청/ 마이크 끄기 • 마이크 무음/해제 • 마이크 사용 금지/해제
청취자	방 내 마이크에 연결된 참여자는 마이크를 켜서 사용하여 인터랙티브할 수 있습니다.	• 역할은 청취자/방송자일 수 있습니다. • 마이크 사용/끄기 초청

구현 프로세스

- **방주인**

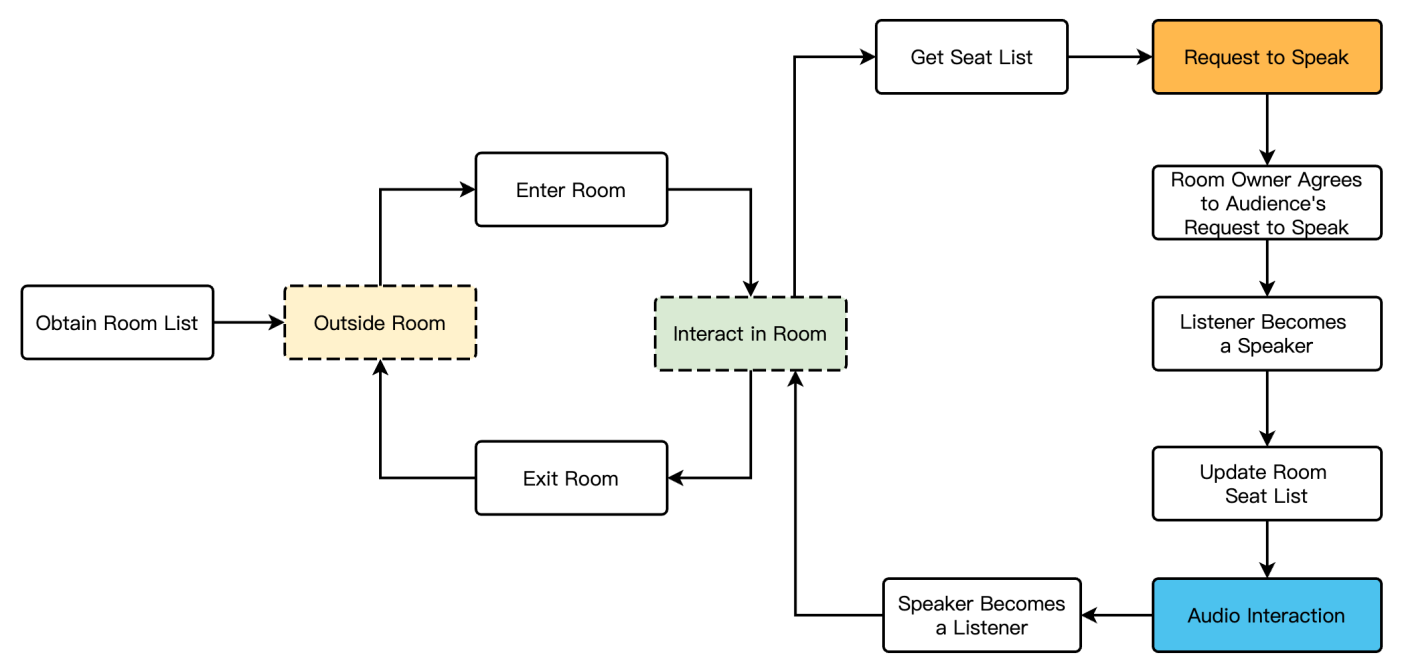


1. 스트리머가 방 홀에 들어가면 방 목록을 가져옵니다.
2. 스트리머가 방주인으로 방을 생성하고 방에 입장합니다.
3. 방주인은 그룹 속성에 의해 마이크 순위 목록을 받고 자동으로 마이크 켜서 사용합니다
4. 청취자가 마이크 켜서 사용합니다.마이크 사용한 후 마이크에 연결중의 다른 사용자와 인터랙티브할 수 있습니

다. 청취자가 마이크 사용하는 방식은 두 가지 있습니다. 청취자가 직접 마이크 사용의 신청을 하고 방주인이 동의하고 나서 사용하는 것과 방주인이 청취자를 직접 초청하고 청취자가 동의한 후 사용하는 것입니다.

5. 청취자가 마이크 끄고 사용을 중지합니다. 마이크 끄고 사용을 중지하는 방식은 두 가지 있습니다.청취자가 직접 마이크 끄고 사용을 중지하는 방식과 방주인이 청취자의 마이크 사용을 강제로 끄는 방식입니다.
6. 방주인이 방에서 나가고 방을 파기합니다(방이 해산되고 모든 사용자가 강제로 마이크의 사용이 중단되어 방을 나갑니다).

- **청취자**

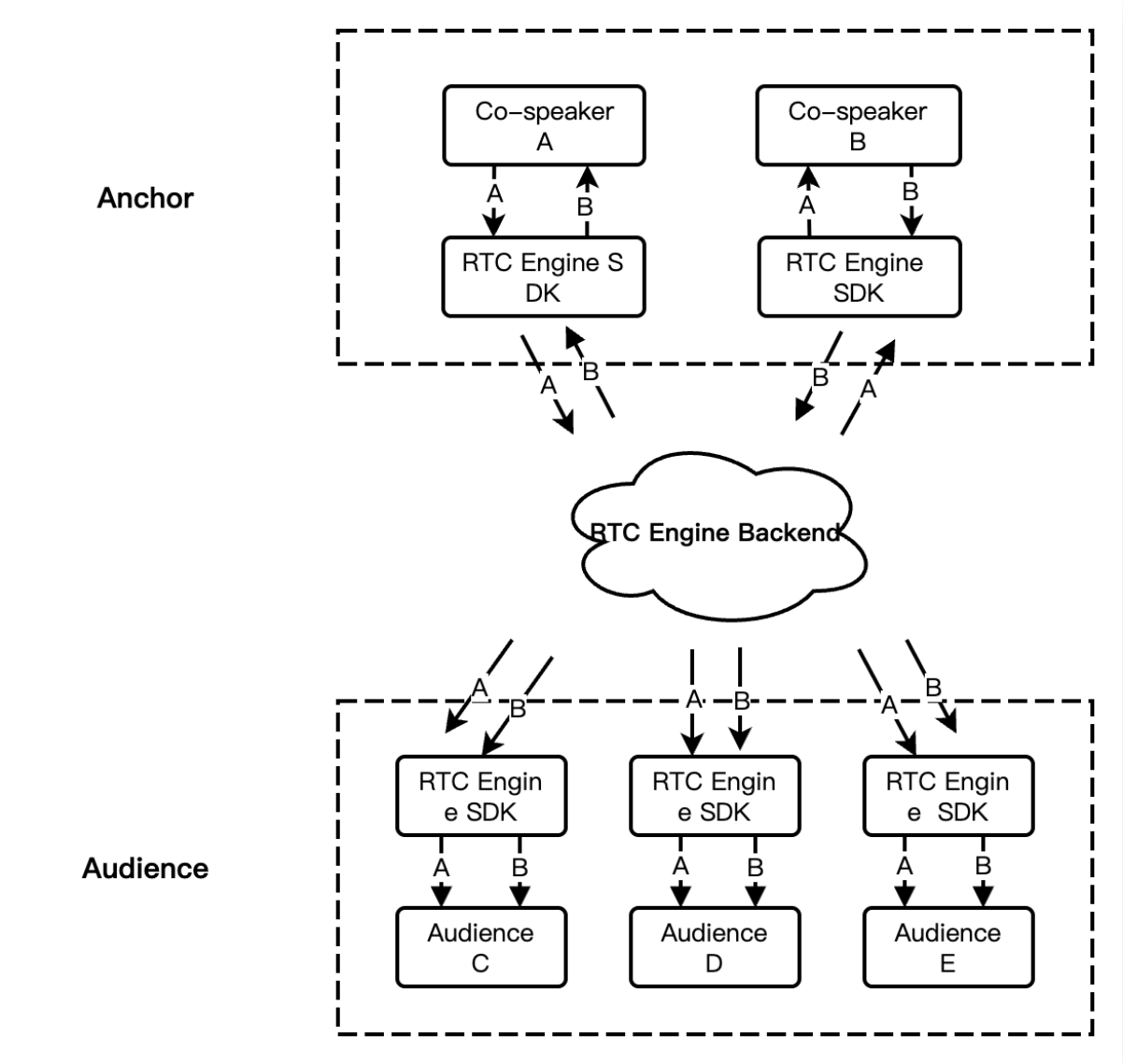


1. 청취자가 방 홀에 들어가서 방 목록을 가져옵니다.
2. 청취자가 방을 선택하고 방에 들어갑니다.
3. 청취자는 그룹 속성에 의해서 마이크 순위 목록을 받습니다.
4. 청취자가 마이크 사용의 신청을 하고 방주인이 동의한 후 청취자는 마이크에 연결된 다른 사용자와 인터랙티브할 수 있습니다
5. 청취자가 마이크 끄고 사용을 중지하고 방을 나갑니다.

오디오 스트림 관리

음성 채팅 인터랙티브 시나리오는 일반적으로 RTC 스트림 접근 방식을 선택하며, 접근이 간단하고 빠르며 실시간 인터랙티브의 낮은 지연 특성을 체험할 수 있습니다. 아래 그림과 같이 마이크 사용중의 사용자와 마이크 사용하지

않는 청취자 두 가지 역할로 실시간 음성 채팅의 푸시 및 풀 스트림 아키텍처 솔루션을 보여줍니다.



방 내 실시간 스트림 구독을 위해 RTC Engine은 자동 구독과 수동 구독 두 가지 모드를 제공합니다.

- 자동 구독: 사용자가 방에 들어가면 해당 방의 오디오 및 비디오 스트림을 즉시 수신하며 오디오는 자동으로 재생되고 비디오는 자동으로 디코딩됩니다.
- 수동 구독: 사용자가 방에 들어간 후 `startRemoteView` 를 수동으로 호출하여 비디오 스트림 구독 및 디코딩을 시작시키고, `muteRemoteAudio` 를 수동으로 호출하여 오디오 재생을 시작시켜야 합니다.

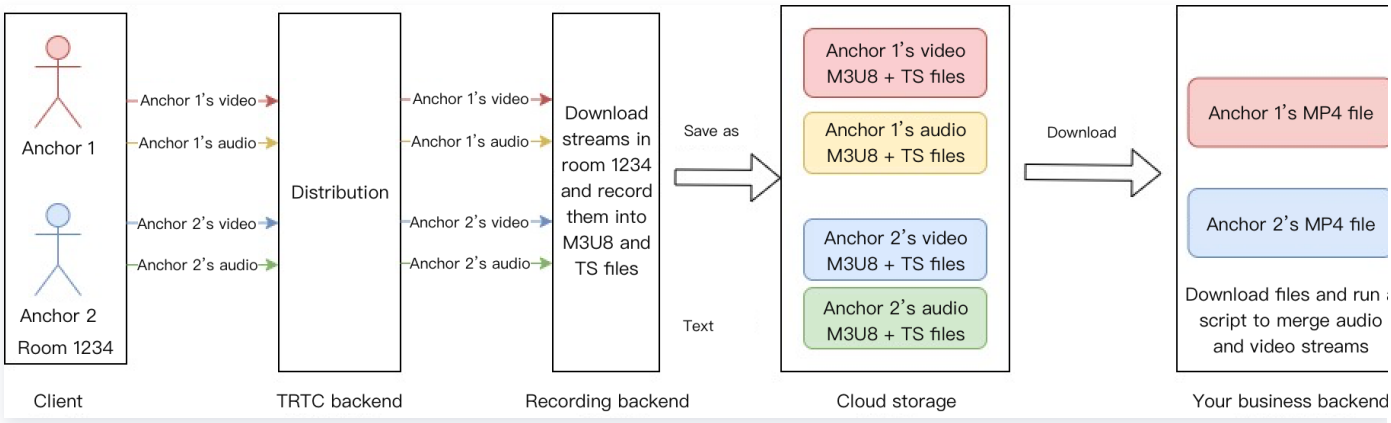
대부분의 시나리오에서 RTC Engine은 자동 구독 모드를 기본으로 사용하며 사용자가 방에 입장하면 방의 모든 스트리머의 오디오 및 비디오 스트림을 구독하여 더 나은 '즉시 시작 체험'을 제공합니다. 반면, 수동 구독 모드는 더 나은 유연성과 맞춤 설정이 가능하며 사용자가 선택적으로 오디오 및 비디오 스트림을 구독할 수 있습니다.

> **설명:**
> 자동 구독 모드는 수동 구독 모드에 비해 복잡한 미디어 스트림 구독 관리가 필요 없으며 음성 채팅 시나리오에서는 특별한 요구 사항이 없는 경우 자동 구독 모드를 사용하는 것이 좋습니다.

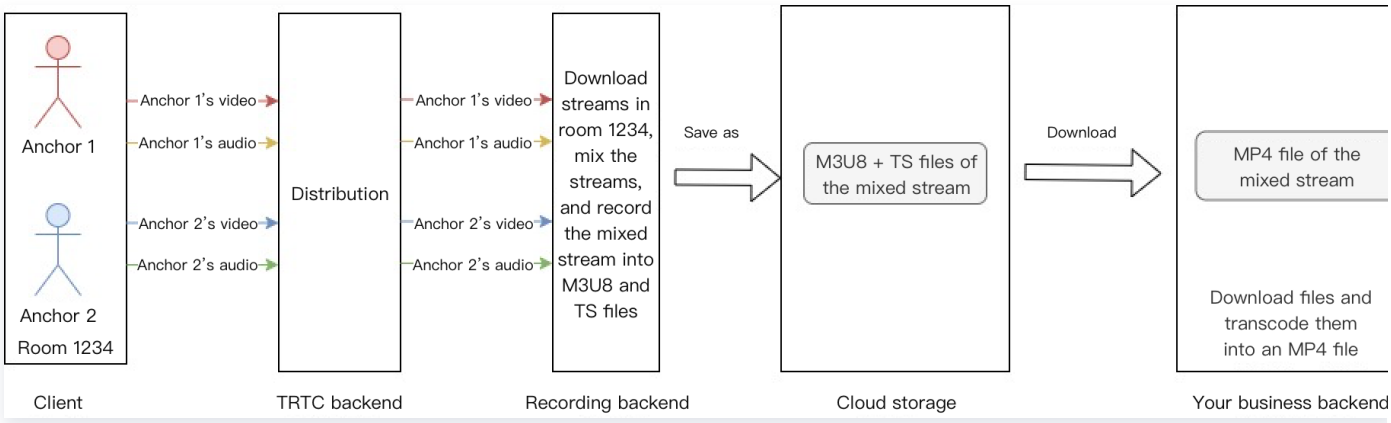
클라우드 레코딩

RTC Engine 최신 업그레이드된 클라우드 레코딩은 RTC Engine 내부의 실시간 레코딩 클러스터를 사용하여 오디오 및 비디오를 레코딩하며, 더 완전하고 통일된 레코딩 체험을 제공합니다.

- 싱글 스트림 레코딩: RTC Engine의 클라우드 레코딩 기능을 통해 방 안의 각 사용자의 오디오 스트림을 개별 파일로 레코딩할 수 있습니다.



- 혼합 스트림 레코딩: 동일한 방의 오디오 미디어 스트림을 하나의 파일로 혼합하여 레코딩합니다.



설명:

> RTC Engine 클라우드 레코딩의 상세 소개 및 개설 안내는 RTC Engine 클라우드 레코딩 설명을 참조하십시오.

핵심 업무 로직

유령 마이크 처리 방안

유령 마이크는 또 폭발 마이크나 블랙 마이크라고도 하며 마이크에 있지 않은 사용자가 말할 수 있고 다른 사용자가 해당 마이크의 사용자의 목소리를 들을 수 있는 현상을 말합니다. 유령 마이크 현상이 발생하는 근본적인 원인은 업무의 마이크 상태와 RTC Engine의 사용자 역할 상태가 일치하지 않기 때문입니다. 이 문제가 발생하는 데는 다음과 같은 몇 가지 가능한 원인이 있습니다.

- 청취자가 마이크 끄고 사용을 중지하여 마이크 리스트가 업데이트되었지만 마이크 정보 콜백이 도달하지 않거나 차단되어 청취자의 로컬에서 RTC Engine의 관객 역할 전환 및 마이크 종료 작업이 실행되지 않아, 마이크에 있지 않아도 여전히 발언할 수 있는 상태가 됩니다.
- 청취자가 마이크 끄고 사용을 중지하여 마이크 리스트가 업데이트되었지만 마이크 정보 콜백을 받은 후 청취자의 로컬에서 RTC Engine의 관객 역할 전환 인터페이스 호출이 실패하여 마이크에 있지 않아도 여전히 발언할 수 있는 상태가 됩니다.
- App이 무차별 대입 공격을 당해 UserSig가 해커에게 탈취되었고, 이로 인해 해커가 스트리머 역할로 RTC Engine 방에 침입해 자유롭게 발언할 수 있었습니다.

유령 마이크의 감지 및 처리

유령 마이크를 감지함으로써 능동적으로 식별하고 신속하게 처리할 수 있습니다. 여기서는 실시간 스트리머 목록 비교및 감지를 하는 서버 측의 감지 솔루션을 권장합니다.
솔루션 원리: 음성 채팅방 시나리오에서 사용자 역할은 스트리머와 시청자로 구분되며 스트리머 역할만 로컬 오디오를 업로드할 수 있습니다. 따라서 업무 마이크 목록과 RTC Engine 역할 목록을 비교하여 유령 마이크를 감지할 수 있습니다. RTC Engine은 서버 측의 방 및 미디어 이벤트 콜백을 제공하며, 방 입장, 역할 전환, 방 퇴장 등의 이벤트를 모니터링하여 현재 방의 실시간 스트리머 목록을 유지할 수 있습니다. 그런 다음 RTC Engine 실시간 스트리머 목록과 업무 전체 마이크 목록을 비교하면 유령 마이크를 쉽게 감지하고 식별할 수 있으며, 이후 방에서 나가게 시키거나 금언등 조치를 취할 수 있습니다.

1. RTC Engine 콘솔은 콜백 정보를 자체 구성할 수 있도록 지원하며, 구성이 완료되면 이벤트 콜백 알림을 받을 수 있습니다. 자세한 내용은 콜백 구성을 참조하십시오.
2. 콜백 이벤트 패킷을 수신하고 파싱하며 103/104/105 이벤트를 주시하여 현재 방의 실시간 온라인 스트리머 역할 사용자 목록을 통계합니다. 자세한 내용은 콜백 이벤트을 참조하십시오.

103

```
{
    "EventGroupId": 1,                          #방 이벤트 그룹
    "EventType":    103,                        #방 입장 이벤트
```

```
    "CallbackTs":   1687679847972,          #콜백 시간, 단위 밀리초
    "EventInfo":    {
        "RoomId":   "123456",               #방 번호
        "EventTs":  1687679847,             #이벤트 발생 시간, 단위 초
        "EventMsTs":  1687679847899,        #이벤트 발생 시간, 단위 밀리초
        "UserId":  "1a99b0a9",              #사용자 이름
        "Role":  20,                        #사용자 역할 20:스트리머; 21:
시청자
        "TerminalType": 2,                  #터미널 유형
        "UserType": 3,                      #사용자 유형
        "Reason": 1                         #구체적인 사유
    }
}
```

104

```
{
    "EventGroupId": 1,                      #방 이벤트 그룹
    "EventType":    104,                    #방 퇴장 이벤트”
    "CallbackTs":   1687679847972,          #콜백 시간, 단위 밀리초
    "EventInfo":    {
        "RoomId":   "123456",               #방 번호
        "EventTs":  1687679847,             #이벤트 발생 시간, 단위 초
        "EventMsTs":  1687679847899,        #이벤트 발생 시간, 단위 밀리초
        "UserId":  "1a99b0a9",              #사용자 이름
        "Role":  20,                        #사용자 역할 20:스트리머; 21:
시청자
        "Reason": 1                         #구체적인 사유
    }
}
```

105

```
{
    "EventGroupId": 1,                      #방 이벤트 그룹
    "EventType":    105,                    #역할 전환 이벤트
    "CallbackTs":   1687679847972,          #콜백 시간, 단위 밀리초
    "EventInfo":    {
```

```
        "RoomId":   "123456",                  #방 번호
        "EventTs":  1687679847,                #이벤트 발생 시간, 단위 초
        "EventMsTs":  1687679847899,           #이벤트 발생 시간, 단위 밀리초
        "UserId":  "1a99b0a9",                 #사용자 이름
        "Role":  20,                           #사용자 역할 20:스트리머; 21:
시청자
    }
}
```

> ⚠ 주의:
> **105-역할 전환 이벤트**은 사용자가 방에 입장한 후 역할 변경 시에만 트리거되므로, **103-방 입장 이벤트**의 초기 역할 정보를 기반으로 사용자 역할 목록을 보완하고 **104-방 퇴장 이벤트**를 통해 사용자 역할 목록을 삭제하여 유지 관리되는 방 사용자 역할 목록을 더 정확하게 만들어야 합니다.

3. 일정한 빈도로 각 방의 업무 마이크 목록과 RTC Engine 실시간 스트리머 목록을 순환적으로 비교하여 유령 마이크를 식별하고 채팅 금지 또는 방에서 나가게 시킴을 합니다.

마이크 연결/해제 시 끊김 방지 솔루션

문제 설명

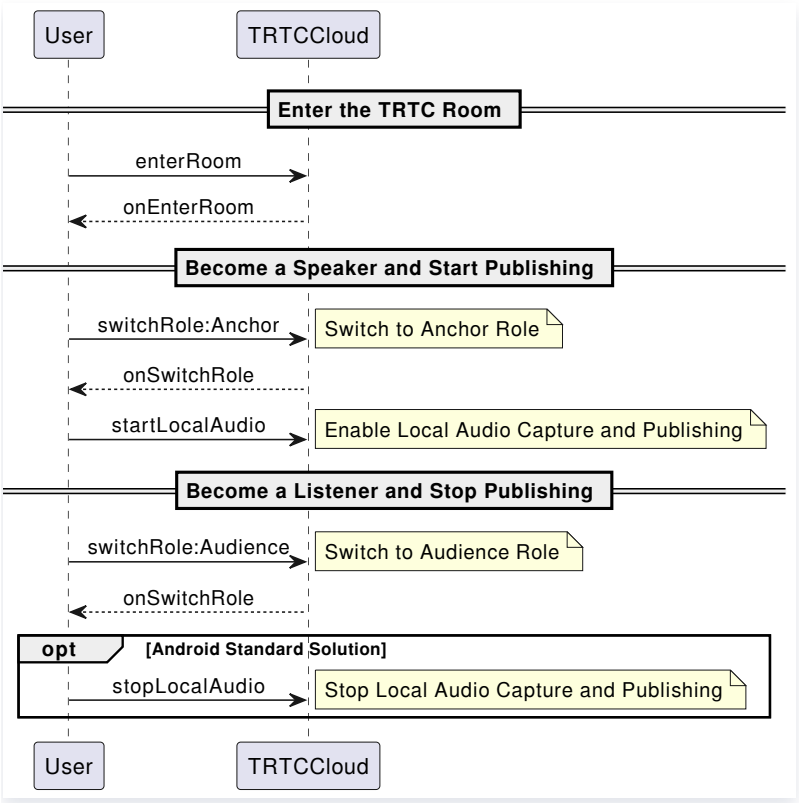
모바일 기기 시스템 메커니즘의 차이로 인해 Android와 iOS는 음성 채팅 시나리오에서 마이크 연결/해제 시의 동작이 일치하지 않습니다. iOS 단말에서는 마이크 연결/해제 시 잠시적인 오디오 끊김이 발생할 수 있습니다.

원인 분석

이는 iOS 시스템 오디오 메커니즘과 관련이 있습니다. `startLocalAudio` 및 `stopLocalAudio` 작업은 마이크 장치 권한을 획득하고 릴리스하며, SDK의 오디오 재수집으로 인해 AVAudioSession이 오디오 드라이버를 재시작시켜 마이크를 켜고 끌 때 오디오가 잠시 끊기는 현상이 발생합니다.

솔루션

RTC Engine의 일반적인 마이크 연결/해제 시나리오의 타이밍은 아래 그림과 같으며, 역할을 전환하는 동시에 로컬 오디오의 수집 및 발행을 시작하거나 중지합니다. 이 시나리오는 Android에서 정상적으로 사용할 수 있습니다. iOS에서는 마이크를 끄고 사용을 중지하는 작업에서 `stopLocalAudio` 를 호출하여 오디오 수집을 중단하고 마이크 권한을 해제하지 않아도 시청자의 역할를 전환하는 것만으로 푸시 스트림을 중지할 수 있어 마이크 켜고 끌 때 끊김 현상을 피할 수 있습니다.



> ⚠ 주의:
> 마이크 켜고 끌 때 끊김 방지 솔루션에서 마이크를 끄고 사용을 중지하는 작업에서 `stopLocalAudio` 를 호출하지 않으면 시스템이 계속 마이크 수집 상태를 유지하여 사용자 오해를 일으킬 수 있습니다.

오디오 구성의 최적화 사례

오디오 구성에서 오디오 음질과 볼륨 유형은 서로 다른 개념입니다. RTC Engine에서 오디오 음질은 로컬 오디오 수집 및 게시를 시작할 때 `startLocalAudio(TRTCAudioQuality)` 를 설정하거나 `setAudioQuality(TRTCAudioQuality)` 를 통해 개별적으로 음질를 설정할 수 있습니다. 볼륨 유형은 방 입장 시나리오와 오디오 음질 설정 등 다양한 요소의 조합에 의해 결정되며, 또한 `setSystemVolumeType(TRTCSystemVolumeType)` 를 통해 특정 볼륨 유형을 강제로 지정할 수도 있습니다.

오디오 음질 구성의 최적화 사례

RTC Engine SDK는 현재 다양한 수직 시나리오에서 오디오 음질에 대한 차별화된 요구를 충족하기 위해 세 가지 신중하게 조정된 오디오 음질 모드를 제공합니다.

오디오 음질 모드	오디오 음질 열거 값	오디오 음질의 매개 변수	오디오 음질의 설명

인간 음성 모드	TRTCAudioQuality Speech	샘플링 레이트: 16k; 모노; 인코딩 비트레이트: 16kbps	네트워크 저항성이 강해서 약한 네트워크 환경에서도 원활한 흐름을 유지하며, 온라인 회의나 음성 통화 등 인간 음성 커뮤니케이션이 주가 되는 애플리케이션 시나리오에 적합합니다.
기본 모드	TRTCAudioQuality Default	샘플링 레이트: 48k; 모노; 인코딩 비트레이트: 50kbps	SDK 기본 모드는 음악 복원도가 인간 음성 모드보다 우수하며 전송 데이터량이 음악 모드보다 훨씬 적어서 다양한 시나리오에 적합합니다.
음악 모드	TRTCAudioQuality Music	샘플링 레이트: 48k; 풀 밴드 스테레오; 인코딩 비트레이트: 128kbps	이 모드에서 오디오 전송 데이터량이 매우 커서 음악 신호가 모든 주파수 대역에서 고음질의 디테일 복원도를 보장하며, 고음질 음악 전송이 필요한 시나리오에 적합합니다.

위 표를 통해서 인간음성 모드에서 음악 모드로 음질 효과가 점차 높아지지만 오디오 전송 데이터량이 점차 늘어납니다.

- 음성 채팅방 시나리오에서 순수 인간 음성 커뮤니케이션은 인간음성 모드를 선택하는 것이 좋으며, 네트워크 조건에서 더 나은 유연성을 얻을 수 있기 때문입니다.
- 좋은 오디오 디테일 복원도를 얻기 위해 배경 음악 재생이 필요한 음성 채팅방은 기본 모드 또는 음악 모드를 선택하는 것을 권장합니다
- 하행 시청자의 네트워크 대역폭 부담을 고려하여 원활한 체험을 보장하기 위해, 10명 이상의 마이크 사용자가 있는 업무 시나리오에서는 음악 모드의 사용을 신중하게 고려해야 합니다.

> **설명:**
> RTC Engine 오디오 음질은 동적 조정을 지원하며 스트리밍 과정에서 `setAudioQuality(TRTCAudioQuality)` 를 호출하여 오디오 음질을 동적으로 조정할 수 있습니다.

볼륨 유형 구성의 최적화 사례

RTC Engine SDK는 현재 다양한 시나리오에서 볼륨 유형에 대한 차별화된 요구를 충족하기 위해 세 가지 시스템 볼륨 유형 제어 모드를 제공합니다.

볼륨 유형 모드	볼륨 유형 모드의 열거값	볼륨 유형 모드의 설명
전체 통화 볼륨	TRTCSystemVolumeTypeVOIP	이 방안의 장점은 사용자가 마이크를 켜고 끌 때 오디오 모듈이 작업 모드를 전환할 필요 없이 원활하게 마이크를 켜고 끌 수 있어서 사용자가 자주 마이크를 켜고 끄는 애플리케이션 시나리오에 적합합니다. 방에 입장할 때 선택한 시나리오가 TRTCAppSceneVideoCall 또는 TRTCAppSceneAudioCall인 경우 SDK가 자동으로 이 모드를 사용합니다.

자동 전환 모드	TRTCSystemVolumeTypeAuto	"마이크 켜면 통화하고, 마이크 끄면 미디어로 전환"이라고도 하며 스트리머가 마이크를 켤 때는 통화 볼륨을 사용하고 시청자가 마이크를 끄고 사용하지 않으면 미디어 볼륨을 사용하는 방식으로 온라인 라이브방송 시나리오에 적합합니다. 방에 입장할 때 선택한 시나리오가 TRTCAppSceneLIVE 또는 TRTCAppSceneVoiceChatRoom인 경우 SDK가 자동으로 이 모드를 사용합니다.
전체 미디어 볼륨	TRTCSystemVolumeTypeMedia	통화 시 미디어 볼륨만 사용하며 음질 요구 사항이 까다로운 음악 시나리오에 적합합니다. 사용자 대부분이 외부 장치(예: 외장 사운드 카드)를 사용하는 경우 이 모드를 사용할 수 있습니다.

- 통화 시나리오에서는 기본 전체 통화 볼륨을 사용하는 것을 권장하며, 이 경우 오디오 모드 전환이 필요 없습니다.
- 음성 채팅방 시나리오에서 순수 음성 커뮤니케이션은 기본 자동 전환 모드(마이크 켜면 통화하고 마이크 끄면 미디어로 전환)를 선택하는 것이 좋습니다.
- 배경 음악을 재생해야 하는 음성 채팅방은 전체 미디어 볼륨을 설정하면 사용자가 마이크를 켜고 끌 때 원격 음악의 끊김 및 볼륨 급변을 방지할 수 있습니다.

설명:
- 특정 볼륨 유형을 지정하려면 방 입장 후 스트리밍 전에 `setSystemVolumeType` 을 한 번 호출하는 것이 좋으며, 마이크를 켜고 끌 때 호출하지 마십시오.
- 통화 음량은 휴대폰 자체의 AEC 기능을 사용할 수 있으며 블루투스 이어폰의 마이크를 통한 음성 수신도 지원됩니다. 단점은 음질이 별로입니다.
- 미디어 음량은 휴대폰 자체의 AEC 기능을 사용할 수 없으며 블루투스 이어폰의 마이크를 통한 음성 수신도 지원되지 않습니다. 하지만 더 나은 음악 재생 효과를 제공합니다.

단일 스트림 음량 크기의 평가

음성 채팅방 시나리오에서 일부 고객은 대역폭을 줄여 비용을 절감하기 위해 마이크 사용자가 RTC 단일 스트림을 푸시하고 시청자가 방 내의 혼합 스트림을 풀링하는 방식을 선택합니다. 그러나 음성 채팅방 시나리오에서는 일반적으로 마이크 사용자의 음량 크기에 따라 UI에 "음파 그래프" 또는 "볼륨 바"와 같은 해당 피드백을 표시해야 합니다. 단일 오디오 스트림의 음량 크기 평가 피드백 기능은 RTC Engine 방 내에서 쉽게 구현할 수 있지만, 순수 오디오 혼합 스트림에서 구현하려면 몇 가지 특별한 방법이 필요합니다. 아래에서는 두 가지 솔루션의 구체적인 구현 방법을 각각 소개하겠습니다.

RTC 방 내 단일 스트림 음량의 평가

단계1: 볼륨 크기 알림의 활성화

`enableAudioVolumeEvaluation` 인터페이스를 통해 음량 크기 콜백을 활성화하고 선택적으로 로컬 인간음성 감지 기능을 켭니다. 이 기능을 활성화하면 SDK는 `onUserVoiceVolume` 콜백에서 로컬 사용자와 원격 스

트리밍 사용자의 음량 크기, 최대 음량 값 및 로컬 인간음성 감지 결과를 피드백합니다.

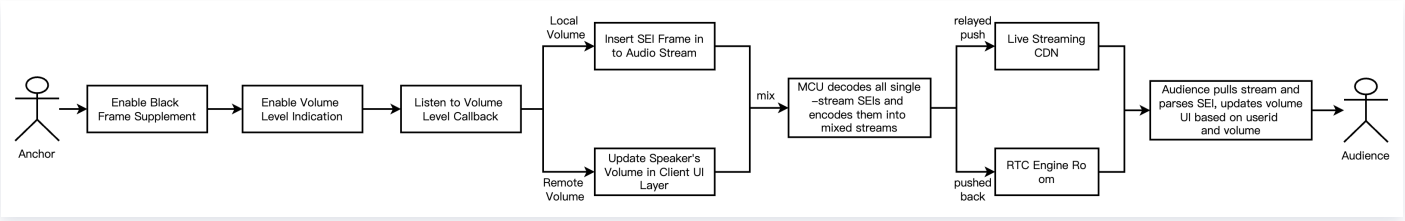
> ⚠ 주의:
> RTC Engine SDK 10.2및 그의 이상 버전에서는 로컬 인간음성 감지 기능이 추가되었으며, 활성화한 후 `TRTCVolumeInfo.vad` 에서 로컬 인간음성 감지 결과를 표시합니다(스트리머 역할이어야 함). `muteLocalAudio` 및 `setAudioCaptureVolume(0)` 작업은 인간음성 감지 결과에 영향을 주지 않아 사용자에게 마이크 켜기를 알리기에 편리합니다.

단계 2: 볼륨 크기 콜백의 모니터링

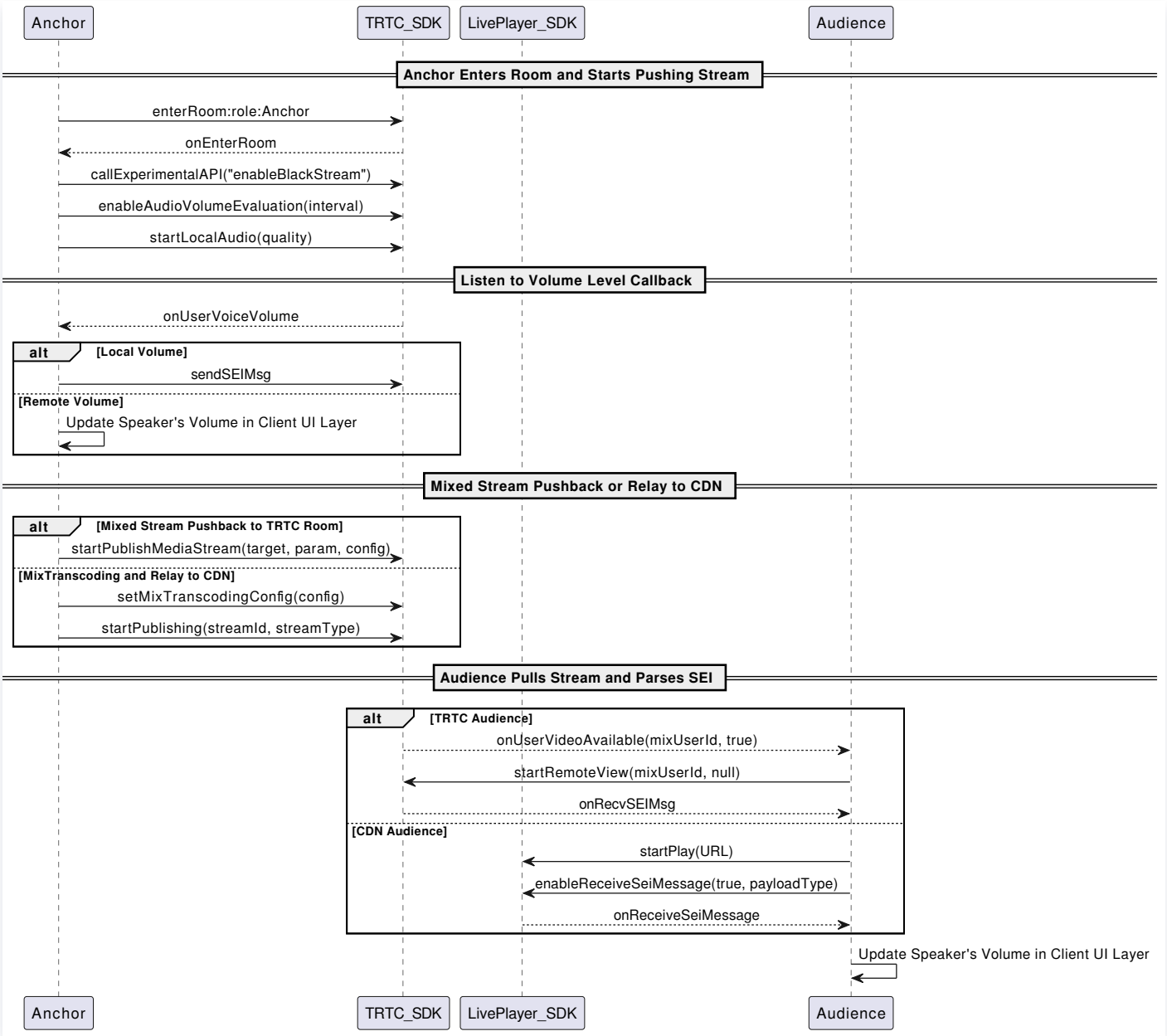
`TRTCCloudListener` 에서 `onUserVoiceVolume` 콜백을 모니터링하며 콜백에서 로컬 사용자와 원격 스트리밍 사용자의 볼륨 크기 및 원격 최대 볼륨 값을 피드백합니다. 볼륨 크기에 따라 UI에서 해당하는 음파 표시를 할 수 있습니다.

> ⚠ 주의:
> 마이크 사용자의 음파 애니메이션 렌더링은 onUserVoiceVolume 콜백의 볼륨 크기에 따라 결정할 수 있으며, 음파 애니메이션의 활성화 및 비활성화(사용자의 마이크 상태)는 onUserAudioAvailable 콜백에 따라 결정하는 것이 좋습니다.

순수 오디오및 혼합 스트림 음량의 평가



순수 오디오및 혼합 스트림 음량의 구현 프로세스는 위 그림과 같습니다. 마이크 사용중의 스트리머는 음량 크기 콜백을 모니터링하고 로컬 음량과 원격 음량을 판단해야 하며, 로컬 음량 값 및 사용자 정보를 SEI 메시지 형태로 오디오 스트림에 삽입한 후 혼합 스트림을 거쳐 마이크 미사용중의 시청자에게 전달합니다. **혹은 방주인 한 명이 모든 마이크 사용중의 스트리머의 콜백 음량 값을 SEI 방식으로 전송할 수도 있습니다**. 아래 그림은 전체 프로세스의 타이밍 다이어그램을 보여줍니다:



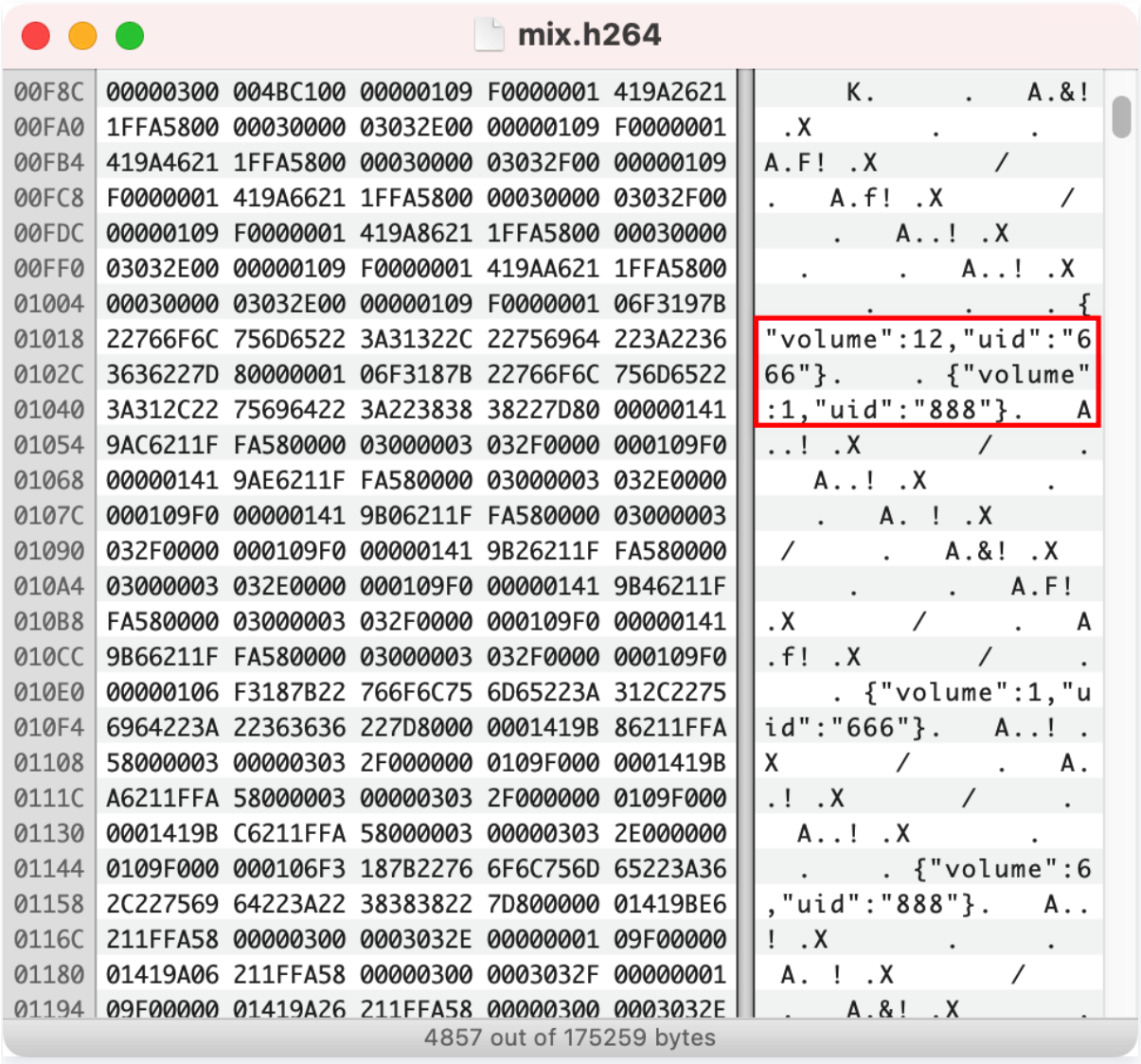
주의:

혼합 스트림 전환 CDN을 통해 SEI를 전달해야 하는 경우:

- 입장 시나리오에서는 반드시 LIVE를 선택해야 하고 순수 음성 입장 시나리오를 선택해서는 안 됩니다. 그렇지 않으면 SEI 메시지가 전달되지 않습니다.

- 혼합 스트림 인터페이스가 `setMixTranscodingConfig` 를 사용하는 경우, 혼합 스트림 모드는 `PureAudio` 순수 오디오 모드를 선택할 수 없습니다.
- 혼합 스트림 인터페이스가 `startPublishMediaStream` 를 사용하는 경우, 미디어 스트림 트랜스코딩 구성 매개변수에 `TRTCVideoLayout` 매개변수를 반드시 포함해야 합니다.

아래 그림과 같이, 시청자 측에서 혼합 스트림을 풀링하여 분석된 SEI 메시지에는 해당 발화자의 음량 크기가 표시됩니다.



시나리오 플레이

음성 채팅방 시나리오에서 방주인과 몇 명의 마이크 사용자는 음성으로 온라인에서 인터랙티브하며, 일부 시청자는 발언은 할 수 없으며 청취만 가능할 수도 있습니다. 선물 증정과 채팅 메시지를 통해 인터랙티브할 수 있습니다. 일반적으로 동일한 취향을 가진 사용자들이 시청하고 인터랙티브할 수 있도록 다양한 방 테마를 설정하며, 일반적인 테마로는 FM 라디오, 노래방 채팅, 게임 인터랙티브, 경기 생방송 등이 있습니다.

FM 라디오 방

스트리머가 단독으로 생방송을 진행하거나 몇 명의 게스트와 함께 방송하는 방식입니다. 배경 음악과 음향 효과를 재생하는 동시에, 시청자들은 선물을 보내고 마이크를 켜서 음성 인터랙티브에 참여할 수 있습니다.
이 시나리오는 일반적으로 시청자 수가 많습니다. RTC Engine 은 단일 방에서 동일 시점에 최대 10만 명을 지원할 수 있으며, 스트리머가 RTC 스트림을 푸시하고 시청자가 RTC 단일 스트림을 풀하는 방식을 채택하는 것에 적합합니다.

설명:
- 이 시나리오의 추천: 스트리머가 RTC 스트림을 푸시/풀하고 시청자가 CDN 혼합 스트림을 풀하는 방안입니다.

KTV 음성 채팅방

일반적으로 한 명의 관리자가 있으며, 노래 신청, 댓글, 노래 맞추기, 노래 이어부르기 등을 할 수 있습니다. 주로 다인 연맥과 다인 론맥 두 가지 모드로 나뉩니다. 다인 연맥은 한 사람이 메인 보컬을 맡고 다른 사용자는 들으면서 말할 수 있지만 메인 보컬은 다른 사용자들의 목소리를 들을 수 없습니다. 방의 청취자들은 모든 소리를 들을 수 있습니다. 다인 론맥 모드는 노래 신청 후 한 사람이 한 구절씩 부르고, 부르기가 끝나면 자동으로 다음 사람에게 차례가 넘어갑니다. 다른 사용자들은 대기 시간 동안에 듣기만 할 수 있으며 댓글로 소통할 수 있지만 음성 채팅은 할 수 없습니다.
온라인 노래방 시나리오는 지연 동기화에 대한 요구 사항이 높으며, 시청자가 언제든지 마이크 켜서 노래를 같이 부르려는 요구가 있으므로, 마이크 사용중의 스트리머가 RTC 스트림을 푸시/풀하고 시청자가 RTC 혼합 스트림을 풀하는 솔루션에 적합합니다. 여기서는 혼합 스트림 봇이 혼합 스트림 명령을 시작하고 혼합 스트림을 RTC Engine 룸으로 다시 푸시하여 시청자들이 스트림을 풀어 볼 수 있도록 해야 합니다.

설명:
- 이 시나리오의 추천: 스트리머가 RTC 스트림을 푸시/풀하고 시청자가 RTC 혼합 스트림을 풀하는 방안입니다.
- KTV 음성 채팅방 구현의 구체적인 기술 세부 사항 및 주의 사항에 대해서는 온라인 노래방 시나리오 솔루션 을 참조하십시오.

인터랙티브 게임룸

늑대인간 게임, 시나리오 게임, PIA 연기, 진실말 게임, 그림 맞추기 등 시나리오에서는 게임 진행에 따라 방을 생성하고 게임 진행 상황에 따라 말하는 플레이어의 권한을 제어하여 순서대로 발언합니다.
인터랙티브 게임 시나리오는 일반적으로 참여 인원이 제한적이며, 자주 마이크를 켜고 끄며 게임에 참여하려는 요구가 있어서 스트리머가 RTC 스트림을 푸시/풀하고 시청자가 RTC 단일 스트림을 풀하는 일반적인 방안에 적합합니다. 게임 참여자는 언제든지 마이크를 켜고 발언하거나 마이크를 끌 수 있으며, 캐릭터가 사망되면 강제로 마이크를 끄고 시청만 하거나 방을 나갑니다.

설명:
- 이 시나리오의 추천: 스트리머가 RTC 스트림을 푸시/풀하고 시청자가 RTC 단일 스트림을 풀하는 방

안입니다.

- 인터랙티브 게임룸은 일반적으로 로컬 게임 음향 효과 재생을 포함하며, AEC 처리 및 볼륨 유형 선택에 주의해야 합니다.

방안 관련 제품

시스템 계층	제품명	시나리오 용도
접근 계층	RTC Engine	저지연, 고품질의 다중 사용자 음성 실시간 인터랙티브 라이브방송 솔루션을 제공하며 음성 채팅 소셜 시나리오의 기반 인프라 능력입니다.
접근 계층	Chat	기반으로 된 그룹 기능의 방 관리 및 마이크 관리 능력을 제공하며, 라이브 방송 참가자 전체 메시지, 공개 화면 메시지 등의 풍부한 미디어 메시지 송부/수신 및 사용자 자체 정의 시그널링 등 통신 요구를 구현합니다.
클라우드 서비스	클라우드 VOD	오디오 및 비디오 미디어를 위해 제작 업로드, 저장, 트랜스코딩, 미디어 처리, 미디어 AI, 가속 분배 재생, 저작권 보호 등 일체형의 고품질 미디어 서비스를 제공합니다.
데이터 저장	객체 저장 COS	오디오 레코딩 파일 및 오디오 슬라이스 파일의 저장 서비스를 제공합니다.

고속 접속 가이드
Android

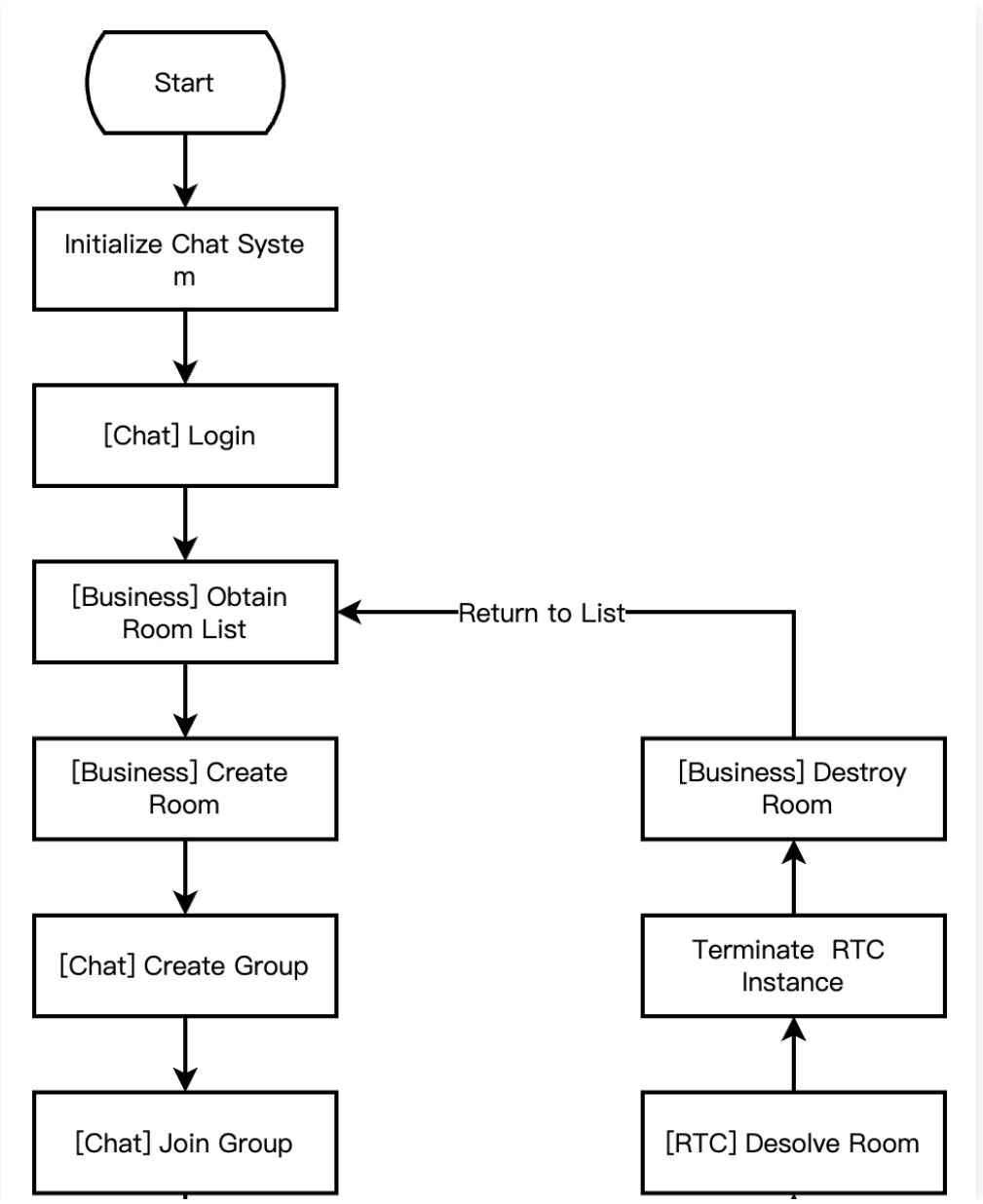
최종 업데이트 날짜:: 2025-10-28 11:32:37

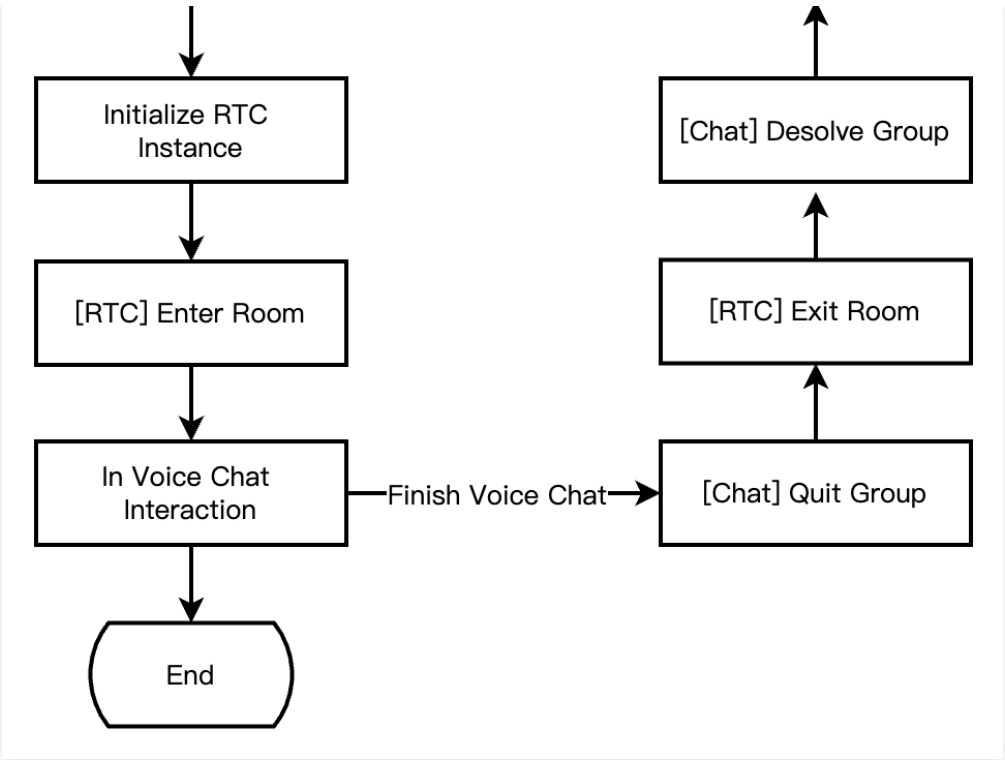
혼합 스트리밍의 전환 및 푸시백 업무 프로세스

본 섹션에서는 음성 채팅방의 몇 가지 일반적인 업무 프로세스를 요약하여 전체 시나리오 구현 프로세스를 더 잘 이해할 수 있도록 도와줍니다.

방 관리 프로세스

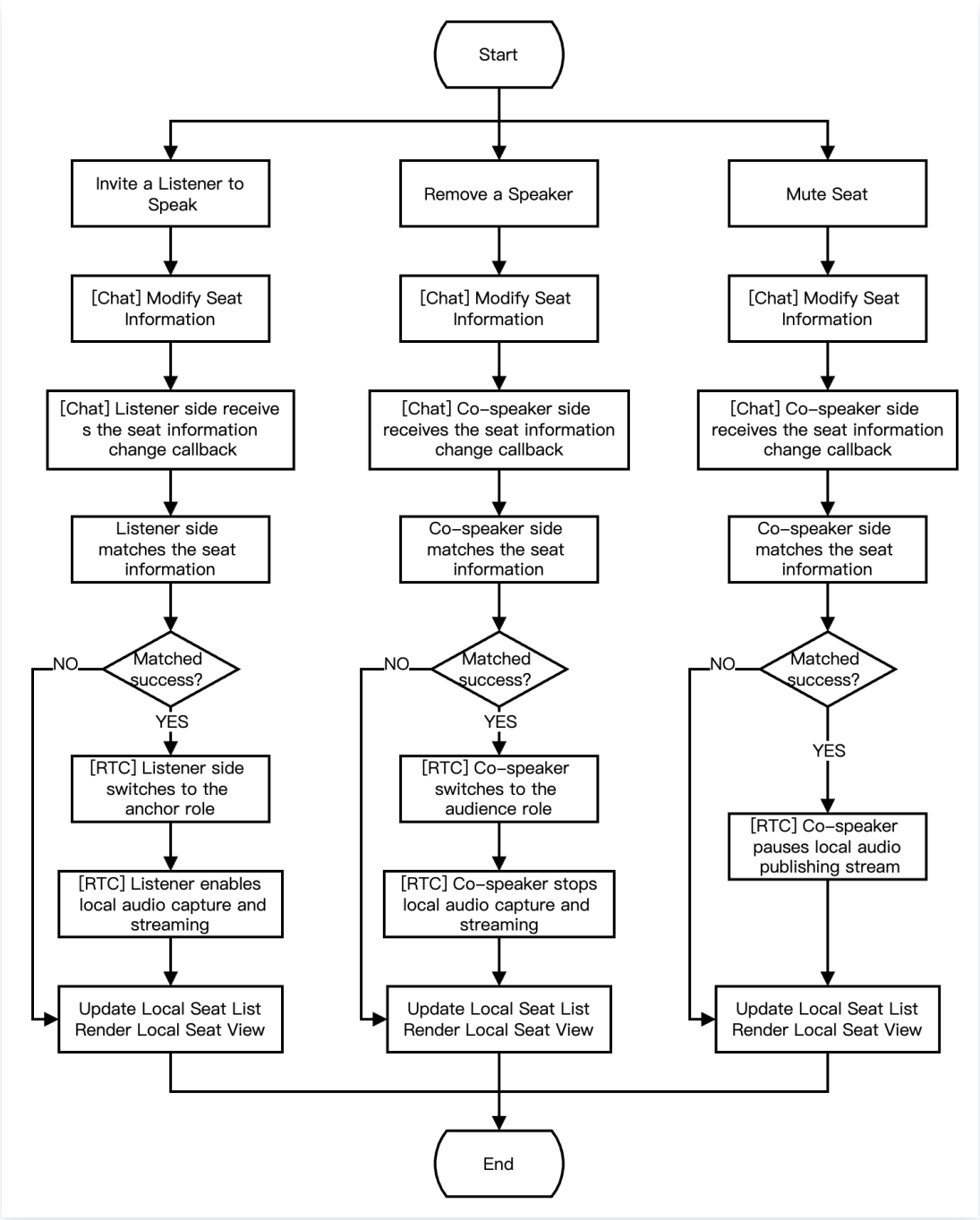
아래 그림은 방 생성, 입장, 퇴장, 해산 등의 구현 프로세스를 포함한 방 관리 프로세스를 보여줍니다.





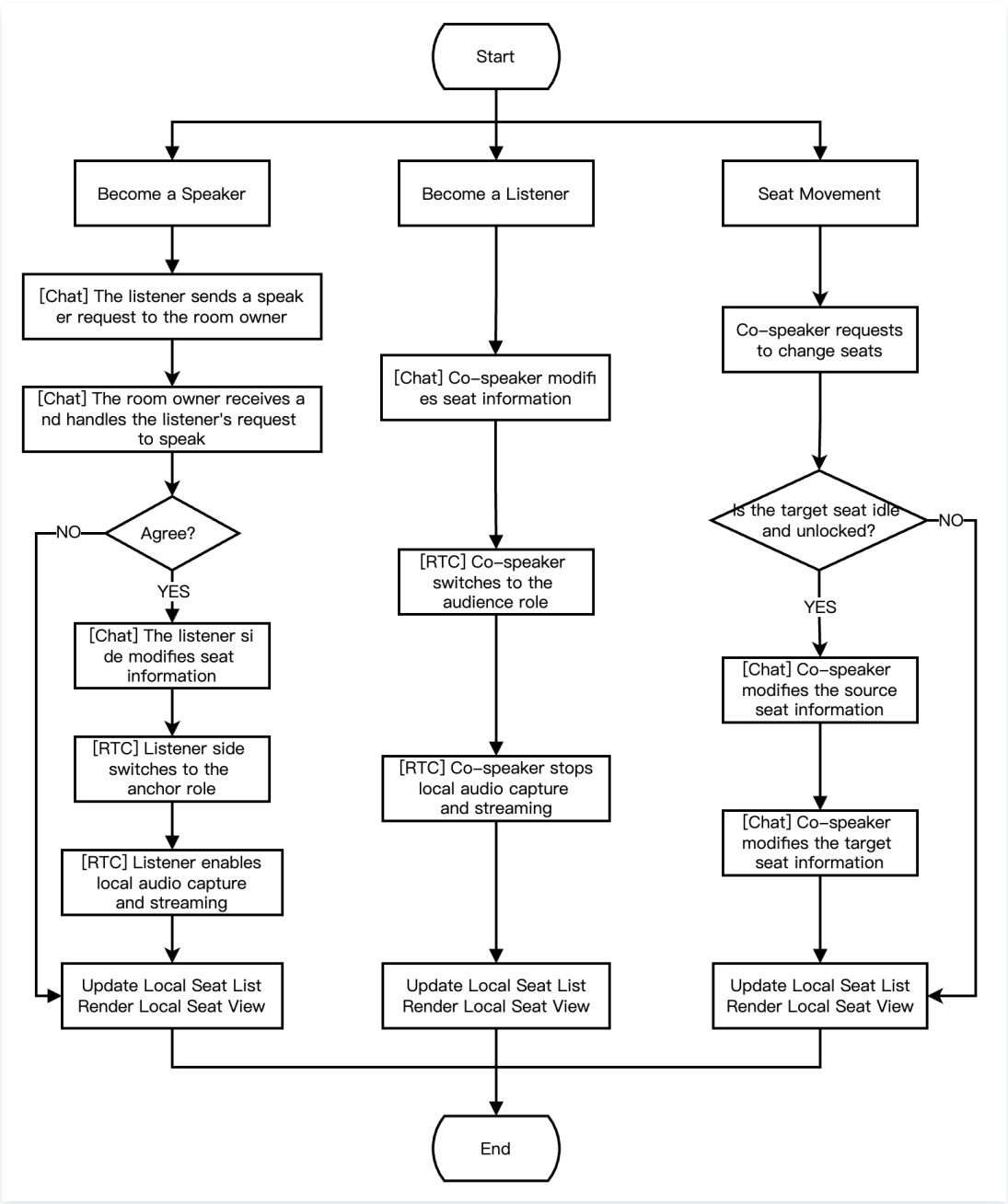
방장 마이크 순위 관리 프로세스

아래 그림은 방주인이 마이크를 관리하는 프로세스를 보여줍니다. 마이크 사용 초청, 마이크 사용 강제 끄기, 마이크 금언등의 구현 프로세스를 포함합니다.



청취자 마이크 순위 관리 프로세스

아래 그림은 청취자가 마이크를 관리하는 프로세스를 보여줍니다. 마이크 자동 사용, 마이크 자동 끄기, 마이크 순위 이동 등의 구현 프로세스를 포함합니다.

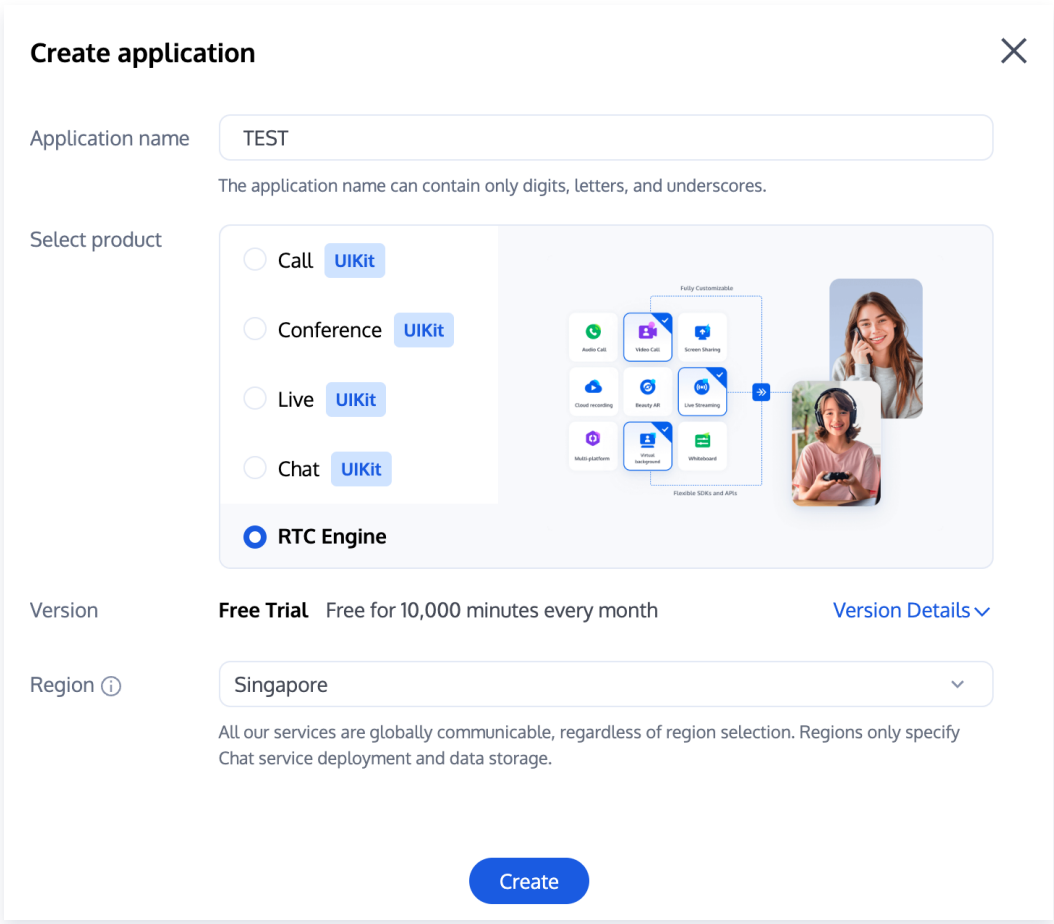


접수 준비

단계1: 서비스 개통

음성 채팅방 시나리오는 일반적으로 Chat과 RTC Engine 두 가지 유료 PaaS 서비스 구축에 의존해야 합니다.

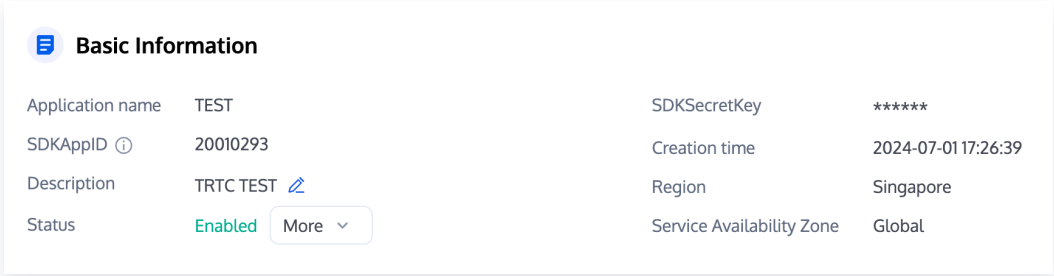
1. 먼저 콘솔에 로그인하여 애플리케이션을 생성해야 합니다. RTC Engine 애플리케이션을 생성한 후 Chat 애플리케이션을 생성하세요.



설명:

- 두 개의 애플리케이션을 생성하고 각각 테스트 환경과 프로덕션 환경에 적용할 것을 권장하며 1년 동안 각 계정(UIN)당 매월 10,000분의 무료 사용 시간이 제공합니다.
- RTC Engine 월정액 요금제는 체험판(기본), 라이트, 스탠다드, 프로페셔널로 구분되며, 다양한 부가 기능 서비스를 사용할 수 있습니다. 자세한 내용은 버전 기능 및 월정액 요금제 설명을 참조하세요.

2. 애플리케이션 생성이 완료되면 **애플리케이션 관리-애플리케이션 개요** 항목에서 해당 애플리케이션의 기본 정보를 확인할 수 있습니다. 이후 사용을 위해 **SDKAppID**와 **SDKSecretKey**를 안전하게 보관해야 하며 키 유출로 인한 트래픽 도용되지 않도록 주의해야 합니다.



단계2: SDK 임포트하기

RTC Engine SDK 와 Chat SDK는 **mavenCentral** 저장소에 릴리스되었으며 gradle을 구성하여 자동 다운로드 및 업데이트가 가능합니다.

1. dependencies에 적합한 버전의 SDK 종속성을 추가합니다.

```
dependencies {
    // RTC Engine 라이트 버전 SDK는 TRTC 및 라이브 방송 재생 두가지 기능을 포함합니다
    implementation 'com.tencent.liteav:LiteAVSDK_TRTC:latest.release'

    // Chat SDK 추가하며 최신 버전 번호를 입력하는 것을 권장합니다
    implementation 'com.tencent.imsdk:imsdk-plus:Version number'

    // Quic 플러그인을 추가하려면 다음 줄의 주석을 취소하세요(주의: 플러그인 버전 번호와 Chat SDK 버전 번호가 동일해야 함)
    // implementation 'com.tencent.imsdk:timquic-plugin:Version number'
}
```

설명:

- 추천하는 자동 로드 방식 외에도 SDK를 다운로드하고 수동으로 임포트할 수도 있습니다. 자세한 내용은 RTC Engine SDK 수동 통합 및 Chat SDK 수동 통합을 참조하세요.
- Quic 플러그인은 axp-Quic 멀티플렉싱 전송 프로토콜을 제공하며 약한 네트워크 환경에서도 더 우수한 성능을 발휘합니다. 네트워크 패킷 손실률이 70%에 달하는 조건에서도 서비스를 제공할 수 있습니다. 이 기능은 프로페셔널, 프로페셔널 플러스 및 엔터프라이즈 버전 사용자에게만 제공되며, 프로페셔널, 프로페셔널 플러스 또는 엔터프라이즈 버전을 구매한 후 사용할 수 있습니다. 기능의 정상적인 사용을 위해 터미널 SDK를 7.7.5282 및 그의 이상 버전으로 업데이트하세요.

2. defaultConfig에서 앱이 사용하는 CPU 아키텍처를 지정합니다.

```
defaultConfig {
     ndk {
             abiFilters "armeabi-v7a", "arm64-v8a"
     }
}
```

설명:

- RTC Engine SDK는 armeabi/armeabi-v7a/arm64-v8a 아키텍처를 지원하며 에뮬레이터 전용의

x86/x86_64 아키텍처도 지원합니다.

- Chat SDK는 armeabi-v7a/arm64-v8a/x86/x86_64 아키텍처를 지원하며 설치 패키지 크기를 줄이기 위해 일부 아키텍처의 SO 파일만 패키징할 수 있습니다.

단계 3: 엔지니어링 구성

1. AndroidManifest.xml에서 앱 권한을 구성하며 음성 채팅 시나리오에서 RTC Engine SDK 및 Chat SDK에는 다음 권한이 필요합니다.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission
android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

⚠ 주의:

- RTC Engine SDK에는 내장된 권한 요청 로직이 없으므로 해당 권한과 기능을 직접 선언해야 하며 일부 권한(예: 저장, 녹음 등)은 운전 시 동적으로 요청해야 합니다.
- Android 프로젝트의 `targetSdkVersion` 이 31이거나 대상 기기가 Android 12및 그 이상 시스템 버전과 관련된 경우, 블루투스 기능을 정상적으로 사용하기 위해 코드에서 `android.permission.BLUETOOTH_CONNECT` 권한을 동적으로 신청해야 합니다. 자세한 내용은 블루투스 권한를 참조하십시오.

2. SDK 내부에서 Java의 리플렉션 기능을 사용하기 때문에 proguard-rules.pro 파일에서 RTC Engine SDK 관련 클래스를 난독화 제외 목록에 추가해야 합니다.

```
-keep class com.tencent.** { *; }
```

접속 과정

단계1: 인증 자격 증명의 생성

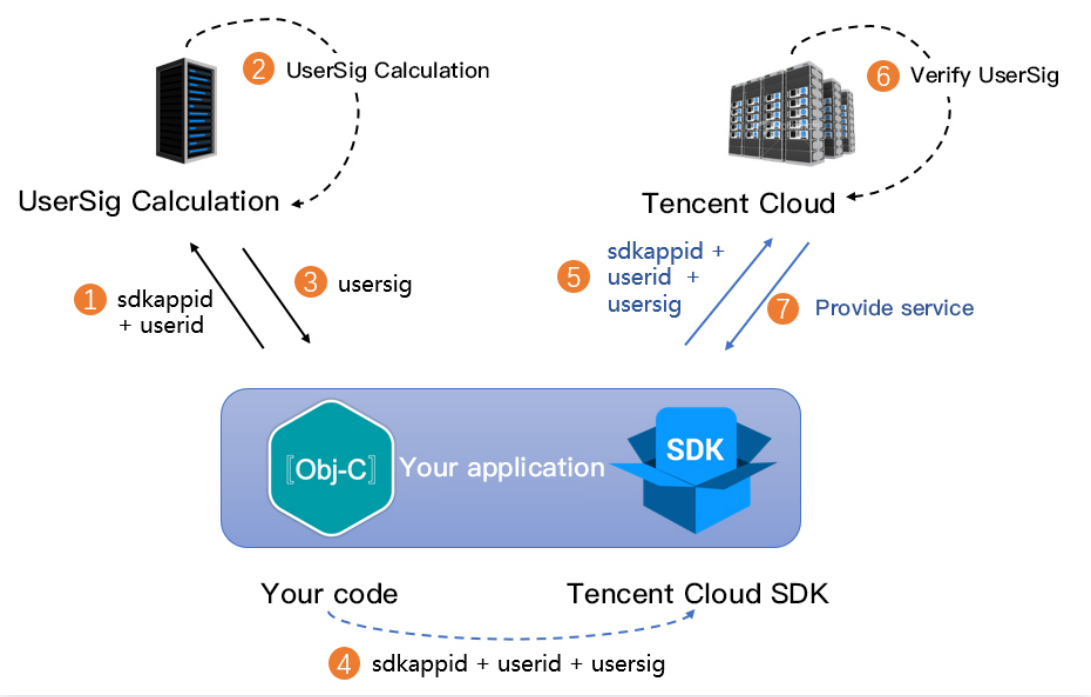
UserSig는 텐센트 실시간 통신 서비스에서 설계한 보안 서명이고 악의적인 공격자가 클라우드 서비스 사용 권한을 도용하는 것을 방지하기 위한 목적입니다. RTC Engine 및 Chat 서비스는 모두 이 보안 메커니즘을 채택하고

있으며 RTC Engine은 방 입장 시 인증을 수행하고 Chat은 로그인 시 인증을 수행합니다.

- 디버깅 및 테스트 단계: 클라이언트 예시 코드와 콘솔에서 가져오기 두 가지 방법으로 UserSig를 계산 및 생성할 수 있으며, 디버깅 및 테스트 용도로만 사용됩니다.
- 정식 운영 단계: 클라이언트가 역공학으로 키가 유출되는 것을 방지하기 위해 보안 등급이 더 높은 서버 UserSig 계산 방식의 사용을 권장합니다.

구체적인 구현 프로세스는 다음과 같습니다.

1. 손님의 App은 SDK 초기화 함수를 호출하기 전에 먼저 서버에 UserSig를 요청해야 합니다.
2. 손님의 서버는 SDKAppID와 UserID에 따라 UserSig를 계산합니다.
3. 서버는 계산된 UserSig를 손님의 App에 반환합니다.
4. 손님의 App은 획득한 UserSig를 특정 API를 통해 SDK에 전달합니다.
5. SDK는 SDKAppID + UserID + UserSig를 텐센트 클라우드 서버에 제출하여 검증합니다.
6. 텐센트 클라우드는 UserSig를 검증하여 합법성을 확인합니다.
7. 검증이 통과되면 Chat SDK에 인스턴트 메시징 서비스를 제공하고 RTC Engine SDK에 실시간 음성/영상 서비스를 제공합니다.

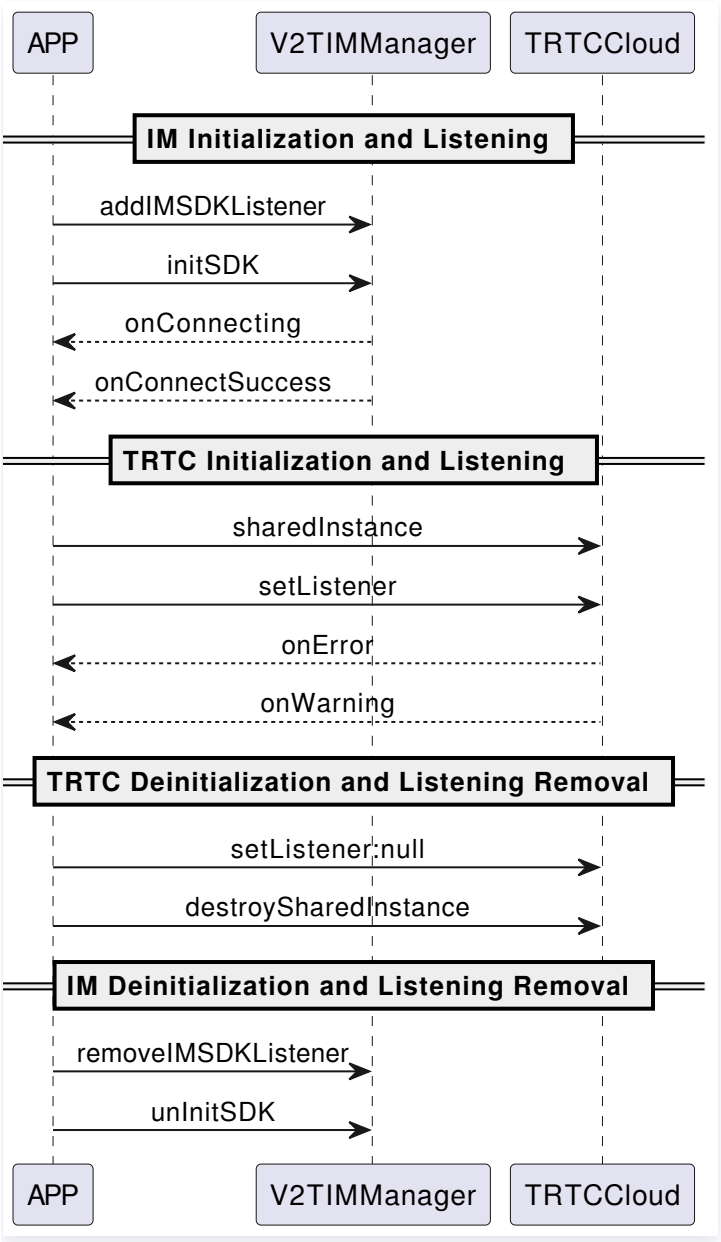


⚠ 주의:

- 디버깅 단계의 로컬 UserSig 계산 방식은 온라인 환경에 적용하는 것을 권장하지 않으며, 역공학으로 인해 키가 유출될 수 있기 때문입니다.
- 여러 언어 버전(Java/Go/PHP/Node.js/Python/C#/C++)의 UserSig 서버 계산원 코드를 제공하며 자세한 내용은 서버에서 UserSig 계산 을 참조하세요.

단계 2: 초기화 및 리스닝

시퀀스 다이어그램



1. Chat SDK 초기화 및 이벤트 리스너의 추가

```
// 이벤트 리스너의 추가
V2TIMManager.getInstance().addIMSDKListener(imSdkListener);
// Chat SDK 초기화하며 이 인터페이스를 호출한 후 즉시 로그인 인터페이스를 호출할
수 있습니다
V2TIMManager.getInstance().initSDK(context, sdkAppID, null);

// SDK 초기화 후 연결 상태, 로그인 티켓 만료 등 일부 이벤트가 발생할 수 있습니다
private V2TIMSDKListener imSdkListener = new V2TIMSDKListener() {
    @Override
```

```
    public void onConnecting() {
        Log.d(TAG, "Chat SDK가 텐센트 클라우드 서버에 연결 중입니다");
    }

    @Override
    public void onConnectSuccess() {
        Log.d(TAG, "Chat SDK가 Tencent Cloud 서버에 성공적으로 연결되었습
니다");
    }
};

// 이벤트 리스너의 제거
V2TIMManager.getInstance().removeIMSDKListener(imSdkListener);
// Chat SDK 초기화의 해제
V2TIMManager.getInstance().unInitSDK();
```

설명:
앱의 라이프사이클이 SDK 라이프사이클과 일치하는 경우, 앱을 종료하기 전에 초기화 해제를 수행하지 않아도 됩니다. 특정 화면에 진입한 후에만 SDK를 초기화하고 화면을 나간 후 더 이상 사용하지 않는 경우, SDK를 초기화 해제할 수 있습니다.

2. RTC Engine SDK 인스턴스의 생성 및 이벤트 리스너의 설정

```
// RTC Engine SDK 인스턴스의 생성(싱글톤 모드)
TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(context);
// 이벤트 리스너의 설정
mTRTCCloud.setListener(trtcSdkListener);

// SDK의 다양한 이벤트 알림(예: 오류 코드, 경고 코드, 오디오/비디오 상태 매개변
수 등)
private TRTCCloudListener trtcSdkListener = new TRTCCloudListener() {
    @Override
    public void onError(int errCode, String errMsg, Bundle extraInfo)
{
        Log.d(TAG, errCode + errMsg);
    }

    @Override
```

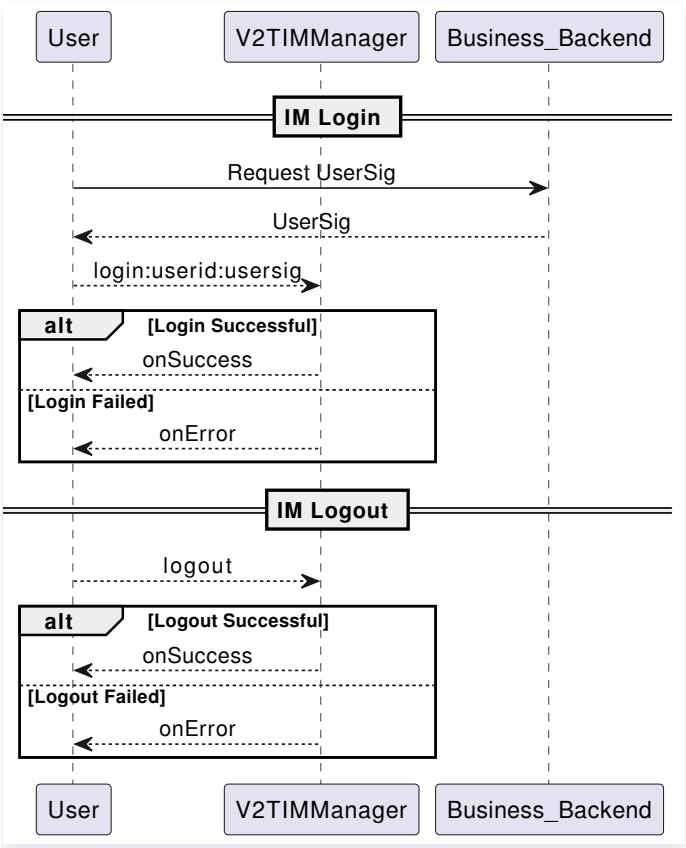
```
    public void onWarning(int warningCode, String warningMsg, Bundle
extraInfo) {
        Log.d(TAG, warningCode + warningMsg);
    }
};

// 이벤트 리스너의 제거
mTRTCCloud.setListener(null);
// RTC Engine SDK 인스턴스(싱글톤 모드)를 파기합니다.
TRTCCloud.destroySharedInstance();
```

설명:
SDK 이벤트 알림을 모니터링하고 일반적인 오류에 대한 로그 출력 및 처리를 권장합니다. 자세한 내용은 오류 코드 표을 참조하세요.

단계3: 로그인 및 로그아웃

Chat SDK 초기화 후, 계정 인증 및 기능 사용 권한을 얻기 위해 SDK 로그인 인터페이스를 호출해야 합니다. 따라서 다른 기능을 사용하기 전에 반드시 로그인이 성공했는지 확인하세요. 그렇지 않으면 기능이 비정상적으로 작동하거나 사용할 수 없을 수 있습니다. RTC Engine 오디오/비디오 서비스만 필요한 경우 이 단계를 무시해도 됩니다.

시퀀스 다이어그램



1. 로그인.

```
// 로그인: userID는 사용자 자체 정의 가능하고 userSig는 단계1 참조하여 생성 및 획득합니다
V2TIMManager.getInstance().login(userID, userSig, new V2TIMCallback()
{
    @Override
    public void onSuccess() {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        // 다음 오류 코드가 반환되면 UserSig가 만료되었음을 의미하므로 새로 발급된 UserSig를 사용하여 다시 로그인하십시오.
        // 1. ERR_USER_SIG_EXPIRED(6206)
        // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED(70001)
        // 주의: 다른 오류 코드의 경우 여기에서 로그인 인터페이스를 호출하지 마십시오. Chat SDK 로그인이 무한 루프에 빠지는 것을 방지하기 위한 것입니다
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
```

```
});
```

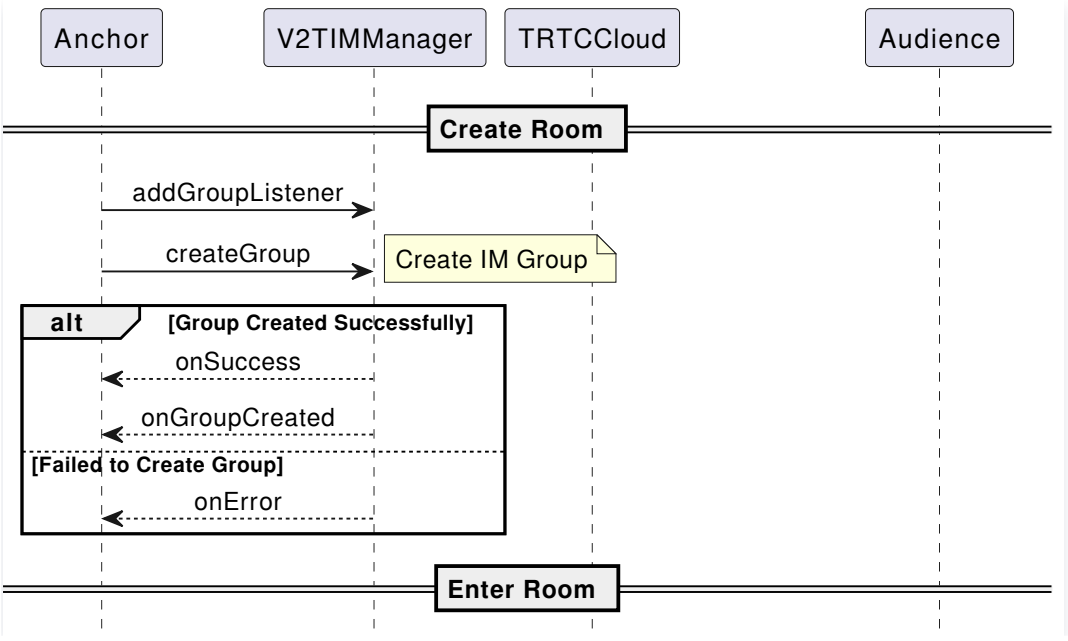
2. 로그아웃

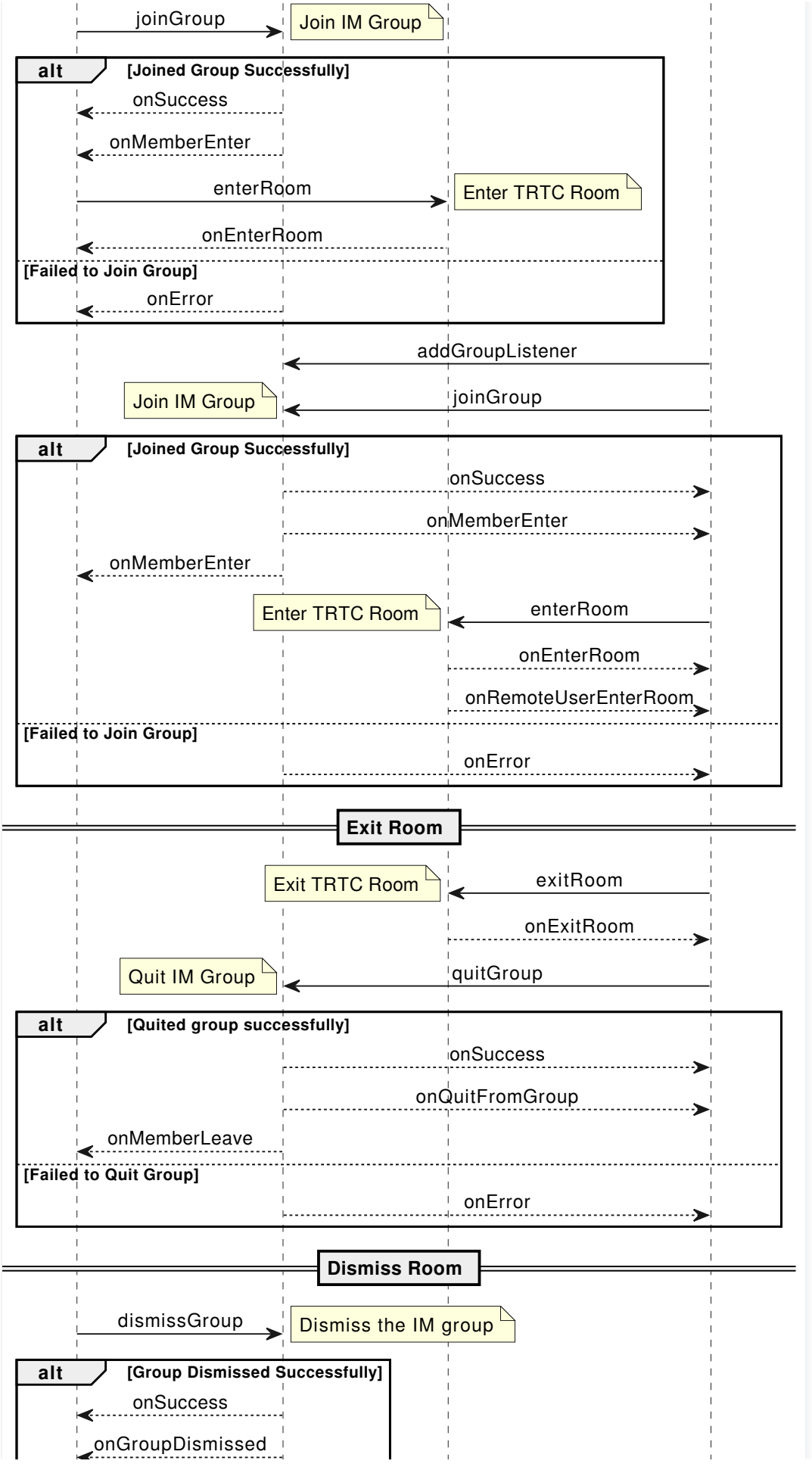
```
// 로그아웃
V2TIMManager.getInstance().logout(new V2TIMCallback() {
    @Override
    public void onSuccess() {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

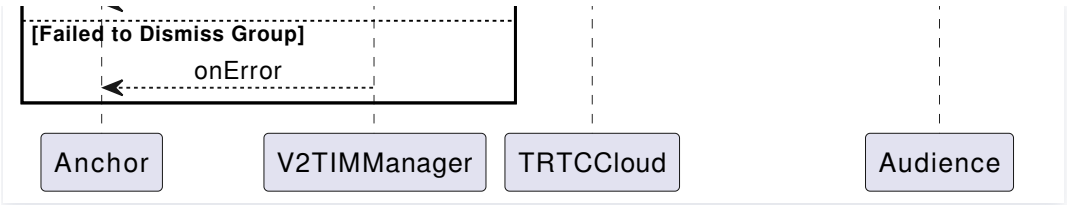
> **설명:**
> Chat SDK의 라이프사이클이 애플리케이션의 라이프사이클과 일치하는 경우, 애플리케이션을 종료하기 전에 로그아웃하지 않아도 됩니다. 특정 화면에 진입한 후에만 Chat SDK를 사용하고 화면을 나간 후에는 더 이상 사용하지 않는 경우, 로그아웃을 수행하고 Chat SDK를 초기화 해제할 수 있습니다.

단계4: 방 관리

시퀀스 다이어그램







1. 방 생성.

   스트리머(방주인)는 방송을 시작할 때 방을 생성해야 하며 여기서 '방' 의 개념은 Chat의 '그룹'에 해당합니다. 이 예시는 클라이언트에서 채팅 그룹을 생성하는 방식만 보여주며 실제로는 서버에서 그룹 생성할 수도 있습니다

```
V2TIMManager.getInstance().createGroup(V2TIMManager.GROUP_TYPE_AVCHAT
ROOM, groupID, groupName, new V2TIMValueCallback<String>() {
  @Override
  public void onSuccess(String s) {
      // 그룹 생성 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 그룹 생성 실패
  }
});

// 그룹 생성 알림을 감시합니다
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
  @Override
  public void onGroupCreated(String groupID) {
      // 그룹 생성 콜백하고 groupID는 새로 생성된 그룹의 ID입니다
  }
});
```

   ⚠ 주의:
   - 음성 채팅방 시나리오에서 Chat 그룹을 생성하려면 라이브방송 그룹 유형 `GROUP_TYPE_AVCHATROOM` 을 선택해야 합니다.
   - RTC Engine에는 방 생성 API가 없으며 사용자가 입장하려는 방이 존재하지 않을 경우 백엔드에서 자동으로 방을 생성합니다.

2. 방 입장.

- Chat 그룹 가입.

```
V2TIMManager.getInstance().joinGroup(groupID, message, new
V2TIMCallback() {
  @Override
  public void onSuccess() {
      // 그룹 가입 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 그룹 가입 실패
  }
});

// 그룹 가입 이벤트의 감시
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
  @Override
  public void onMemberEnter(String groupID,
List<V2TIMGroupMemberInfo> memberList) {
      // 누군가 그룹에 가입했습니다
  }
});
```

- RTC Engine 방에 가입합니다.

```
private void enterRoom(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // 문자열 방 번호를 예로 들면 IM 그룹 번호와 일치하는 것이 좋습니다
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = getUserSig(userId);
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = SDKAppID;
    // 음성 채팅 상호 작용 시나리오에서 방에 들어가려면 사용자 역할을 지정해야
합니다
    params.role = TRTCCloudDef.TRTCRoleAudience;
```

```
    // 음성 채팅 인터랙티브 방 입장 시나리오를 예로 들어
    mTRTCCloud.enterRoom(params,
TRTCCloudDef.TRTC_APP_SCENE_VOICE_CHATROOM);
}

// 방 입장 결과 이벤트의 콜백
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result는 방에 입장하는 데 소요된 시간(밀리초)을 나타냅니다.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result는 방 입장 실패의 오류 코드를 나타냅니다
        Log.d(TAG, "Enter room failed");
    }
}
```

⚠ 주의:

- RTC Engine 방 번호는 정수형 `roomId` 와 문자열 형식 `strRoomId` 로 나뉘며 두 유형의 방은 서로 통하지 않으므로 방 번호 유형을 통일하는 것이 좋습니다.
- 음성 채팅 인터랙티브 시나리오에서 방에 입장 시 사용자 역할(스트리머/시청자)을 지정해야 하며 스트리머만 스트리밍 권한이 있습니다. 지정하지 않으면 기본적으로 스트리머 역할로 설정됩니다.
- 음성 채팅 인터랙티브 방 입장 시나리오에서는 `TRTC_APP_SCENE_VOICE_CHATROOM` 을 선택하는 것을 권장합니다

3. 방에서 나가기
   - Chat 그룹 나가기.

```
V2TIMManager.getInstance().quitGroup(groupID, new V2TIMCallback() {
  @Override
  public void onSuccess() {
      // 그룹 나가기 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 그룹 나가기 실패
```

```
    }
});

V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
    @Override
    public void onMemberLeave(String groupID, V2TIMGroupMemberInfo
member) {
        // 그룹 멤버 나가기의 콜백
    }
});
```

⚠ 주의:
라이브 방송 그룹(AVChatRoom)에서는 그룹 주인이 그룹에서 나갈 수 없으며 `dismissGroup` 을 호출하여 그룹을 해산할 수만 있습니다.

○ RTC Engine 방에서 나가기.

```
private void exitRoom() {
    mTRTCCloud.stopLocalAudio();
    mTRTCCloud.exitRoom();
}

// 방 나가기 이벤트의 콜백
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "exitRoom을 호출하여 방에서 나가기");
    } else if (reason == 1) {
        Log.d(TAG, "서버에 의해 현재 방에서 나가게 됩니다");
    } else if (reason == 2) {
        Log.d(TAG, "현재 방 전체가 해체됩니다");
    }
}
```

⚠ 주의:
- SDK가 점유한 모든 리소스가 릴리스된 후, SDK는 `onExitRoom` 콜백을 통해 알려줍니다.
- `enterRoom` 을 다시 호출하거나 다른 음성/영상 SDK로 전환하려면 `onExitRoom` 콜백이

> 발생한 후 관련 작업을 수행하십시오. 그렇지 않으면 카메라, 마이크 장치가 점유됨 등 다양한 문제가 발생할 수 있습니다.

4. 방 해산하기.

- Chat 그룹 해산.

  본 예시는 클라이언트에서 Chat 그룹 해산하는 방법만 보여주고 실제로는 서버에서 그룹 해산할 수도 있습니다.

```
V2TIMManager.getInstance().dismissGroup(groupID, new V2TIMCallback()
{
  @Override
  public void onSuccess() {
      // 그룹 해산 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 그룹 해산 실패
  }
});

V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
  @Override
  public void onGroupDismissed(String groupID, V2TIMGroupMemberInfo
opUser) {
      // 그룹 해산 콜백
  }
});
```

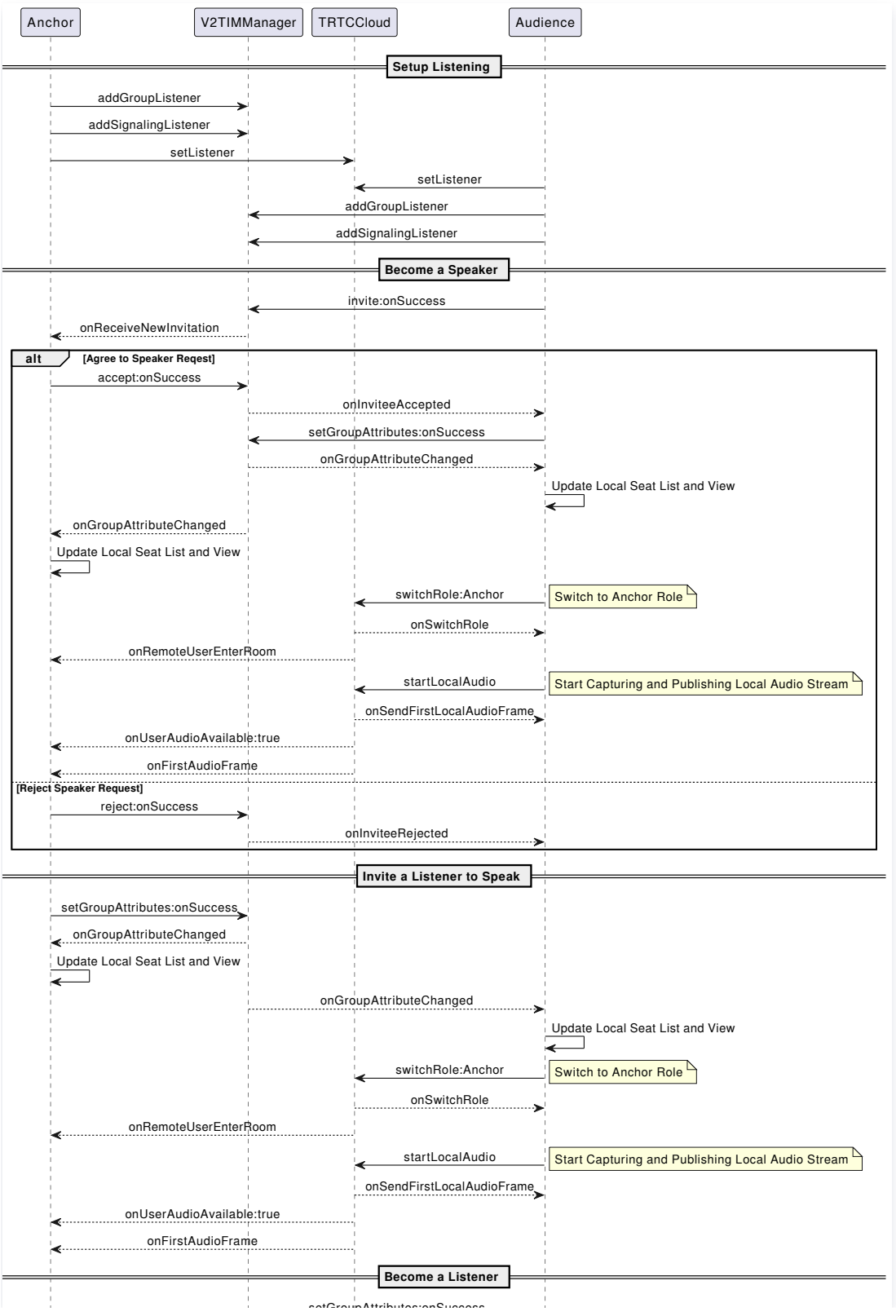
- RTC Engine 방 해산
  - **서버 측 해산**: RTC Engine은 서버 측 방 해산 API `DismissRoom` (숫자 방 ID와 문자열 방 ID 구분)를 제공하며 이 인터페이스를 호출하여 방의 모든 사용자를 방에서 나가게 하고 방을 해산할 수 있습니다.
  - **클라이언트 측 해산**: 각 클라이언트의 방 나가기 `exitRoom` 인터페이스를 통해 방 내 모든 스트리머와 청취자의 퇴장을 완료시키고 퇴장 후 RTC Engine 방 라이프사이클 규칙에 따라 방이 자동으로 해산됩니다. 자세한 내용은 방에서 나가기를 참조하세요.

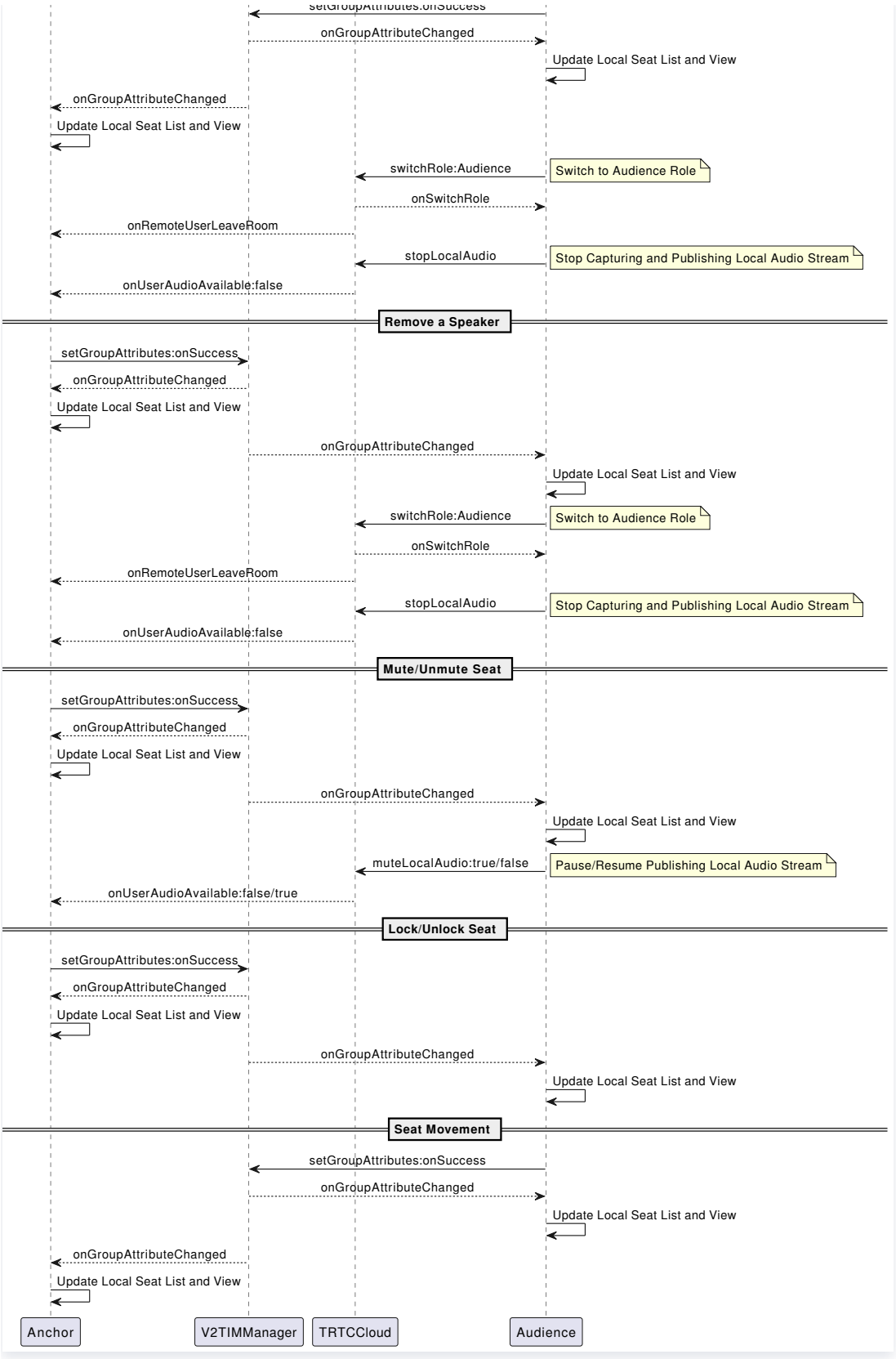
> **경고:**

> 라이브 방송 작업이 종료되면 방 해산 API를 호출하여 방이 해산되도록 하여 청취자가 의외로 방에 들어가서 원치 않는 비용이 발생하는 것을 방지하는 것을 권장합니다.

단계5: 마이크 순위 관리

시퀀스 다이어그램





먼저, 마이크 순위 정보를 저장하기 위한 JavaBean을 생성할 수 있습니다.

```
public class SeatInfo implements Serializable {
    public static final transient int STATUS_UNUSED   = 0;
    public static final transient int STATUS_USED     = 1;
```

```
    public static final transient int STATUS_LOCKED  = 2;

    // 좌석 상태는 세 가지 상태에 대응합니다
    public int status;
    // 좌석 금언 여부
    public boolean mute;
    // 좌석 점유돼었을 때 사용자 정보 저장합니다
    public String userId;

    @Override
    public String toString() {
        return "TXSeatInfo{"
                + "status=" + status
                + ", mute=" + mute
                + ", user='" + userId + '''
                + '}';
    }
}
```

1. 자동으로 마이크 사용.

   자동으로 마이크의 사용을 켜는 것은 방송실에 있는 청취자가 방주인 또는 관리자에게 마이크의 사용 켜기 신청을 보내고 동의한다는 시그널링을 받은 후 마이크를 켜서 사용하는 것을 의미합니다. 자유 마이크 모드인 경우 시그널링 요청 부분은 무시할 수 있습니다.

- 청취자가 마이크 사용 켜기 요청을 보냅니다.

```
// 청취자가 마이크 요청을 보내며 userId는 스트리머 Id이고  data는 시그널링 식별
을 위한 json을 전달할 수 있습니다
private String sendInvitation(String userId, String data) {
    return V2TIMManager.getSignalingManager().invite(userId, data,
true, null, 0, new V2TIMCallback() {
        @Override
        public void onError(int i, String s) {
            Log.e(TAG, "sendInvitation error " + i);
        }

        @Override
        public void onSuccess() {
            Log.i(TAG, "sendInvitation success ");
        }
```

```
    });
}

// 스트리머가 마이크 사용의 요청을 수신하며  inviteID는 해당 요청 Id이고
inviter는 요청자 ID입니다
V2TIMManager.getSignalingManager().addSignalingListener(new
V2TIMSignalingListener() {
    @Override
    public void onReceiveNewInvitation(String inviteID, String
inviter,
                                       String groupId, List<String>
inviteeList, String data) {
        Log.i(TAG, "received invitation: " + inviteID + " from " +
inviter);
    }
});
```

- 스트리머가 마이크 사용 요청을 처리합니다.

```
// 마이크 사용의 요청을 동의합니다
private void acceptInvitation(String inviteID, String data) {
    V2TIMManager.getSignalingManager().accept(inviteID, data, new
V2TIMCallback() {
        @Override
        public void onError(int i, String s) {
            Log.e(TAG, "acceptInvitation error " + i);
        }

        @Override
        public void onSuccess() {
            Log.i(TAG, "acceptInvitation success ");
        }
    });
}

// 마이크 사용의 요청 거절합니다
private void rejectInvitation(String inviteID, String data) {
    V2TIMManager.getSignalingManager().reject(inviteID, data, new
V2TIMCallback() {
```

```
        @Override
        public void onError(int i, String s) {
            Log.e(TAG, "rejectInvitation error " + i);
        }

        @Override
        public void onSuccess() {
            Log.i(TAG, "rejectInvitation success ");
        }
    });
}
```

- 청취자가 마이크를 켜서 사용합니다

  스트리머가 청취자의 마이크 사용 요청을 동의하면 청취자는 그룹 속성을 수정하는 방식으로 마이크 정보를 추가할 수 있으며,다른 사용자는 그룹 속성 변경 콜백을 수신하고 로컬 마이크 정보가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
private List<SeatInfo> mSeatInfoList;

// 마이크 요청 동의의 콜백
V2TIMManager.getSignalingManager().addSignalingListener(new
V2TIMSignalingListener() {
    @Override
    public void onInviteeAccepted(String inviteID, String invitee,
String data) {
        Log.i(TAG, "received accept invitation: " + inviteID + " from
" + invitee);
        takeSeat(seatIndex);
    }
});

// 청취자가 마이크를 켜서 사용하기 시작합니다
private void takeSeat(int seatIndex) {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfo localInfo = mSeatInfoList.get(seatIndex);
    SeatInfo seatInfo = new SeatInfo();
    seatInfo.status = SeatInfo.STATUS_USED;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = mUserId;
```

```
    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    Gson gson = new Gson();
    String json = gson.toJson(seatInfo, SeatInfo.class);
    HashMap<String, String> map = new HashMap<>();
    map.put("seat" + seatIndex, json);

    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    V2TIMManager.getGroupManager().setGroupAttributes(groupId, map,
new V2TIMCallback() {
        @Override
        public void onError(int code, String message) {
            // 그룹 속성의 수정이 실패되고 마이크 연결이 실패됩니다
        }

        @Override
        public void onSuccess() {
            // 그룹 속성의 수정이 성공되고 TRTC 역할의 전환 및 스트리밍 시작합
니다
            mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);

mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
        }
    });
}
```

2. 마이크의 사용을 초대합니다.

   스트리머가 마이크의 사용을 초대합니다(청취자의 동의 필요없이).직접 그룹 속성을 수정하여 저장된 마이크 순위 정보를 변경하며, 해당 청취자는 그룹 속성 변경 콜백을 수신한 후 userId 매칭에 성공하면 RTC Engine 역할을 전환하고 스트리밍을 시작할 수 있습니다. 마이크 사용 초대 모드인 경우, 자동으로 마이크를 켜기 구현 로직을 참조하고 시그널링의 발신자와 수신자를 바꾸기만 하면 됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
private List<SeatInfo> mSeatInfoList;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
private void pickSeat(String userId, int seatIndex) {
```

```
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfo localInfo = mSeatInfoList.get(seatIndex);
    SeatInfo seatInfo = new SeatInfo();
    seatInfo.status = SeatInfo.STATUS_USED;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = userId;

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    Gson gson = new Gson();
    String json = gson.toJson(seatInfo, SeatInfo.class);
    HashMap<String, String> map = new HashMap<>();
    map.put("seat" + seatIndex, json);

    // 그룹 속성을 설정하며 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    V2TIMManager.getGroupManager().setGroupAttributes(groupId, map,
new V2TIMCallback() {
        @Override
        public void onError(int code, String message) {
            // 그룹 속성의 수정이 실패되며 마이크 사용의 초대가 실패됩니다
        }

        @Override
        public void onSuccess() {
            // 그룹 속성의 수정이 성공되면 onGroupAttributeChanged 콜백 트
리거됩니다
        }
    });
}

// 청취자 측에서 그룹 속성 변경 콜백을 수신하여 자체 정보에 매칭 성공 후 스트리밍
시작합니다
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
    @Override
    public void onGroupAttributeChanged(String groupID, Map<String,
String> groupAttributeMap) {
        // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
        final List<SeatInfo> oldSeatInfoList = mSeatInfoList;
        // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
```

```
        final List<SeatInfo> newSeatInfoList =
getSeatListFromAttr(groupAttributeMap, seatSize);
        // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니
다.
        for (int i = 0; i < seatSize; i++) {
            SeatInfo oldInfo = oldSeatInfoList.get(i);
            SeatInfo newInfo = newSeatInfoList.get(i);
            if (oldInfo.status != newInfo.status && newInfo.status ==
SeatInfo.STATUS_USED) {
                if (newInfo.userId.equals(mUserId)) {
                    // 자신의 정보에 매칭 성공 후, RTC Engine 역할 전환 및
스트리밍 시작합니다

mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);

mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
                } else {
                    // 로컬 마이크 순위 목록을 업데이트하고 로컬 마이크 순위
뷰 렌더링합니다
                }
            }
        }
    }
});
```

3. 자동으로 마이크 끄기.

   연결된 청취자는 그룹 속성을 수정하는 방식으로 마이크 순위 정보를 재설정할 수 있으며 다른 사용자는 그룹 속성 변경 콜백을 받고 로컬 마이크 정보가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
private List<SeatInfo> mSeatInfoList;

private void leaveSeat(int seatIndex) {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfo localInfo = mSeatInfoList.get(seatIndex);
    SeatInfo seatInfo = new SeatInfo();
    seatInfo.status = SeatInfo.STATUS_UNUSED;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = "";
```

```
    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    Gson gson = new Gson();
    String json = gson.toJson(seatInfo, SeatInfo.class);
    HashMap<String, String> map = new HashMap<>();
    map.put("seat" + seatIndex, json);

    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    V2TIMManager.getGroupManager().setGroupAttributes(groupId, map,
new V2TIMCallback() {
        @Override
        public void onError(int code, String message) {
            // 그룹 속성의 수정이 실패되고 마이크 끄기가 실패됩니다
        }

        @Override
        public void onSuccess() {
            // 그룹 속성의 수정이 성공되고 RTC Engine 역할 전환 및 스트리밍
중지합니다
            mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
            mTRTCCloud.stopLocalAudio();
        }
    });
}
```

4. 강제로 마이크 끄기.

   스트리머가 사용자의 마이크 사용을 끄게 합니다. 그룹 속성에 저장된 마이크 순위 정보를 직접 수정하여 저장하고 해당 연결된 청취자는 그룹 속성 변경 콜백을 받은 후 userId와 일치하면 RTC Engine 역할을 전환하고 스트리밍을 중지합니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
private List<SeatInfo> mSeatInfoList;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
private void kickSeat(int seatIndex) {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfo localInfo = mSeatInfoList.get(seatIndex);
```

```
    SeatInfo seatInfo = new SeatInfo();
    seatInfo.status = SeatInfo.STATUS_UNUSED;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = "";

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    Gson gson = new Gson();
    String json = gson.toJson(seatInfo, SeatInfo.class);
    HashMap<String, String> map = new HashMap<>();
    map.put("seat" + seatIndex, json);

    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    V2TIMManager.getGroupManager().setGroupAttributes(groupId, map,
new V2TIMCallback() {
        @Override
        public void onError(int code, String message) {
            // 그룹 속성의 수정이 실패되고 마이크  끄기 실패됩니다
        }

        @Override
        public void onSuccess() {
            // 그룹 속성의 수정이 성공되고 onGroupAttributeChanged 콜백 트
리거됩니다
        }
    });
}

// 연결된 청취자 측에서 그룹 속성 변경 콜백을 수신하여 자체 정보에 매칭 성공 후
스트리밍 중지합니다
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
    @Override
    public void onGroupAttributeChanged(String groupID, Map<String,
String> groupAttributeMap) {
        // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
        final List<SeatInfo> oldSeatInfoList = mSeatInfoList;
        // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
        final List<SeatInfo> newSeatInfoList =
getSeatListFromAttr(groupAttributeMap, seatSize);
```

```
        // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니
다.
        for (int i = 0; i < seatSize; i++) {
            SeatInfo oldInfo = oldSeatInfoList.get(i);
            SeatInfo newInfo = newSeatInfoList.get(i);
            if (oldInfo.status != newInfo.status && newInfo.status ==
SeatInfo.STATUS_UNUSED) {
                if (oldInfo.userId.equals(mUserId)) {
                    // 자신의 정보에 매칭 성공 후 RTC Engine 역할을 전환하고
스트리밍을 중지합니다.

mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
                    mTRTCCloud.stopLocalAudio();
                } else {
                    // 로컬 마이크 순위 목록을 업데이트하고 로컬 마이크 순위
뷰 렌더링합니다
                }
            }
        }
    }
});
```

5. 마이크 금언.

   스트리머가 특정 마이크 순위를 금언/해제하고 그룹 속성에 저장된 마이크 순위 정보를 직접 수정하며 해당 연결된 청취자는 그룹 속성 변경 콜백을 수신한 후 userId 매칭에 성공하면 로컬 스트리밍을 일시 중지/재개합니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
private List<SeatInfo> mSeatInfoList;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
private void muteSeat(int seatIndex, boolean mute) {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfo localInfo = mSeatInfoList.get(seatIndex);
    SeatInfo seatInfo = new SeatInfo();
    seatInfo.status = localInfo.status;
    seatInfo.mute = mute;
    seatInfo.userId = localInfo.userId;
```

```
    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    Gson gson = new Gson();
    String json = gson.toJson(seatInfo, SeatInfo.class);
    HashMap<String, String> map = new HashMap<>();
    map.put("seat" + seatIndex, json);

    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    V2TIMManager.getGroupManager().setGroupAttributes(groupId, map,
new V2TIMCallback() {
        @Override
        public void onError(int code, String message) {
            // 그룹 속성의 수정이 실패되고 마이크 금언 실패됩니다
        }

        @Override
        public void onSuccess() {
            // 그룹 속성의 수정이 성공되고 onGroupAttributeChanged 콜백 트
리거됩니다
        }
    });
}

// 연결된 청취자 측에서 그룹 속성 변경 콜백을 수신하여 자체 정보에 매칭 성공 후
스트리밍 일시 중지/재개합니다
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
    @Override
    public void onGroupAttributeChanged(String groupID, Map<String,
String> groupAttributeMap) {
        // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
        final List<SeatInfo> oldSeatInfoList = mSeatInfoList;
        // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
        final List<SeatInfo> newSeatInfoList =
getSeatListFromAttr(groupAttributeMap, seatSize);
        // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니
다.
        for (int i = 0; i < seatSize; i++) {
            SeatInfo oldInfo = oldSeatInfoList.get(i);
```

```
            SeatInfo newInfo = newSeatInfoList.get(i);
            if (oldInfo.mute != newInfo.mute) {
                if (oldInfo.userId.equals(mUserId)) {
                    // 자신의 정보에 매칭 성공 후 로컬 스트리밍을 일시 중지/
재개합니다.
                    mTRTCCloud.muteLocalAudio(newInfo.mute);
                } else {
                    // 로컬 마이크 순위 목록을 업데이트하고 로컬 마이크 순위
뷰 렌더링합니다
                }
            }
        }
    }
});
```

6. 마이크 순위 잠그기.

   스트리머가 특정 마이크를 잠그거나 잠금 해제합니다. 그룹 속성에 저장된 마이크 순위 정보를 직접 수정하며 청취자는 그룹 속성 변경 콜백을 수신한 후 해당 마이크 뷰가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
private List<SeatInfo> mSeatInfoList;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
private void lockSeat(int seatIndex, boolean isLock) {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfo localInfo = mSeatInfoList.get(seatIndex);
    SeatInfo seatInfo = new SeatInfo();
    seatInfo.status = isLock ? SeatInfo.STATUS_LOCKED :
SeatInfo.STATUS_UNUSED;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = "";

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    Gson gson = new Gson();
    String json = gson.toJson(seatInfo, SeatInfo.class);
    HashMap<String, String> map = new HashMap<>();
    map.put("seat" + seatIndex, json);
```

```
    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    V2TIMManager.getGroupManager().setGroupAttributes(groupId, map,
new V2TIMCallback() {
        @Override
        public void onError(int code, String message) {
            // 그룹 속성의 수정이 실패되고 마이크 잠금이 실패됩니다
        }

        @Override
        public void onSuccess() {
            // 그룹 속성의 수정이 성공되고 onGroupAttributeChanged 콜백 트
리거됩니다
        }
    });
}

// 청취자 측에서 그룹 속성 변경 콜백을 수신하여 해당 마이크 뷰가 업데이트됩니다
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
    @Override
    public void onGroupAttributeChanged(String groupID, Map<String,
String> groupAttributeMap) {
        // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
        final List<SeatInfo> oldSeatInfoList = mSeatInfoList;
        // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
        final List<SeatInfo> newSeatInfoList =
getSeatListFromAttr(groupAttributeMap, seatSize);
        // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니
다.
        for (int i = 0; i < seatSize; i++) {
            SeatInfo oldInfo = oldSeatInfoList.get(i);
            SeatInfo newInfo = newSeatInfoList.get(i);
            if (oldInfo.status == SeatInfo.STATUS_LOCKED &&
newInfo.status == SeatInfo.STATUS_UNUSED) {
                // 마이크 잠금의 해제
            } else if (oldInfo.status != newInfo.status &&
newInfo.status == SeatInfo.STATUS_LOCKED) {
                // 마이크 순위 잠그기
            }
```

```
        }
    }
});
```

7. 마이크 순위의 이동.

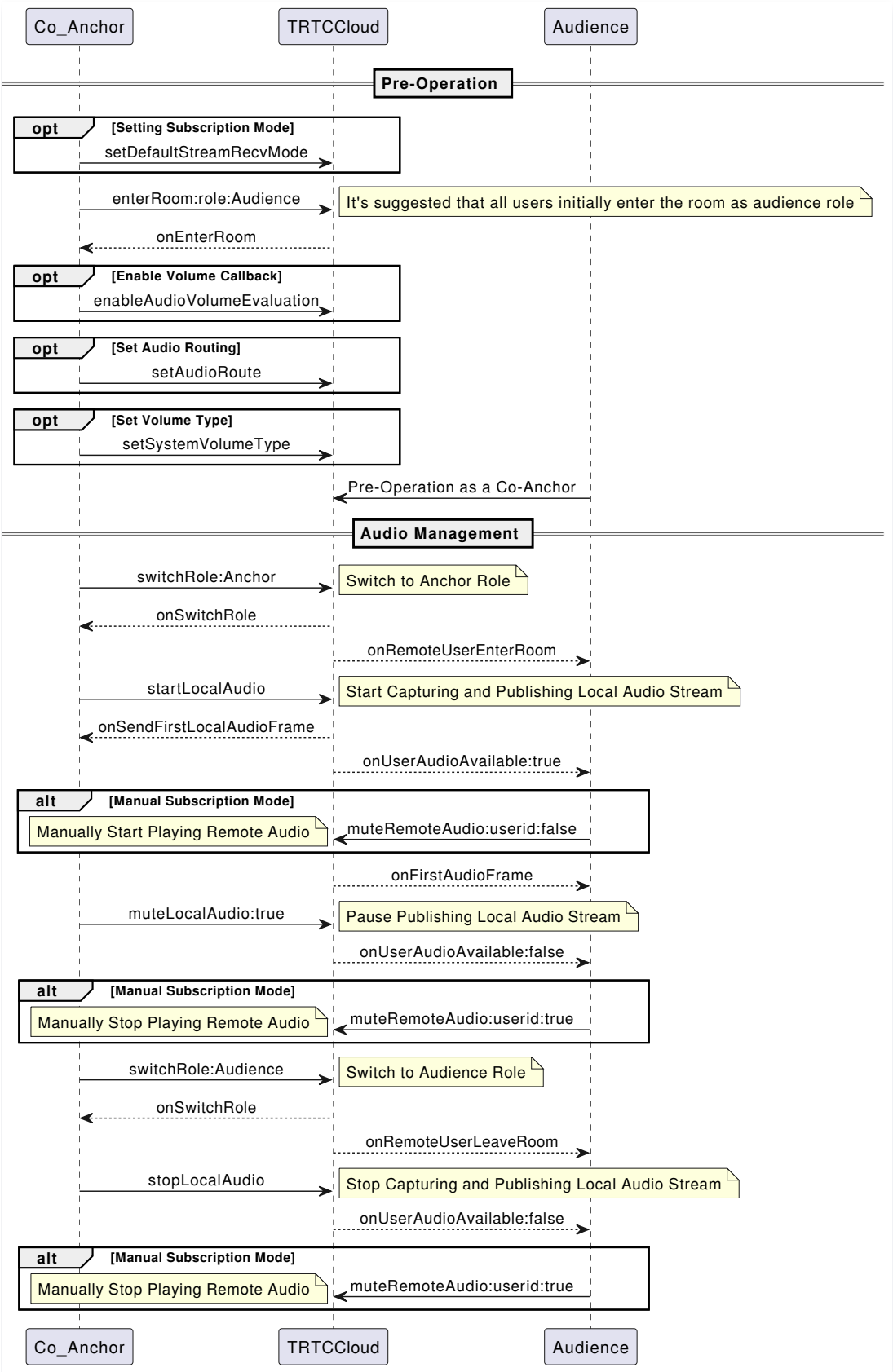
   마이크 사용중의 스트리머가 마이크 순위를 이동하면 그룹 속성에 저장된 소스 및 대상 마이크 정보를 각각 수정해야 하며 청취자는 그룹 속성 변경 콜백을 수신한 후 해당 마이크 뷰가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
private List<SeatInfo> mSeatInfoList;

// 마이크 사용중의 스트리머가 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이
크 정보를 수정합니다.
private void moveSeat(int dstIndex) {
    // userId로 소스 마이크 번호를 받습니다
    int srcIndex = -1;
    for (int i = 0; i < mSeatInfoList.size(); i++) {
        SeatInfo seatInfo = mSeatInfoList.get(i);
        if (seatInfo != null && mUserId.equals(seatInfo.userId)) {
            srcIndex = i;
            break;
        }
    }

    // 마이크 번호로 해당 마이크 순위를 받습니다
    SeatInfo srcSeatInfo = mSeatInfoList.get(srcIndex);
    SeatInfo dstSeatInfo = mSeatInfoList.get(dstIndex);

    // 마이크 위치 정보 인스턴스를 생성하고 수정된 소스 마이크 순위 정보를 저장
합니다
    SeatInfo srcChangeInfo = new SeatInfo();
    srcChangeInfo.status = SeatInfo.STATUS_UNUSED;
    srcChangeInfo.mute = srcSeatInfo.mute;
    srcChangeInfo.userId = "";

    // 마이크 순위 정보 인스턴스를 생성하고 수정된 대상 마이크 순위 정보를 저장
합니다
    SeatInfo dstChangeInfo = new SeatInfo();
    dstChangeInfo.status = SeatInfo.STATUS_USED;
```

```
    dstChangeInfo.mute = dstSeatInfo.mute;
    dstChangeInfo.userId = mUserId;

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    Gson gson = new Gson();
    HashMap<String, String> map = new HashMap<>();
    String json = gson.toJson(srcChangeInfo, SeatInfo.class);
    map.put("seat" + srcIndex, json);
    json = gson.toJson(dstChangeInfo, SeatInfo.class);
    map.put("seat" + dstIndex, json);

    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    V2TIMManager.getGroupManager().setGroupAttributes(groupId, map,
new V2TIMCallback() {
        @Override
        public void onError(int code, String message) {
            // 그룹 속성의 수정이 실패되고 마이크 이동이 실패됩니다
        }

        @Override
        public void onSuccess() {
            // 그룹 속성의 수정이 성공되며 마이크 이동이 성공됩니다
        }
    });
}
```

단계6: 오디오 관리

시퀀스 다이어그램



1. 구독 모드.

   RTC Engine SDK는 기본적으로 오디오 스트림을 자동으로 구독하는 로직을 사용하며, 사용자가 방에 입장하면 원격 사용자의 소리가 자동으로 재생됩니다. 수동으로 오디오 스트림을 구독해야 하는 경우,

`muteRemoteAudio(userId, mute)` 을 추가로 호출하여 원격 사용자의 오디오 스트림을 구독하고 재생해야 합니다.

```
// 자동 구독 모드(기본값)
mTRTCCloud.setDefaultStreamRecvMode(true, true);

// 수동 구독 모드(자체 정의)
mTRTCCloud.setDefaultStreamRecvMode(false, false);
```

> ⚠ **주의:**
> 구독 모드 설정 `setDefaultStreamRecvMode` 은 방 입장 `enterRoom` 전에 호출해야만 적용됩니다.

2. 수집 및 게시.

```
// 로컬 오디오의 수집 및 게시를 활성화합니다
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT)

// 로컬 오디오의 수집 및 게시를 중지합니다
mTRTCCloud.stopLocalAudio();
```

> ⓘ **설명:**
> `startLocalAudio` 는 마이크 사용 권한을 요청하며, `stopLocalAudio` 는 마이크 사용 권한을 릴리스합니다.

3. 마이크 끄기와 켜기.

```
// 로컬 오디오 스트림 게시를 일시 중지합니다(마이크 끄기)
mTRTCCloud.muteLocalAudio(true);
// 로컬 오디오 스트림 게시를 재개합니다(마이크 켜기)
mTRTCCloud.muteLocalAudio(false);

// 특정 원격 사용자의 오디오 스트림 구독 및 재생을 일시 중지합니다
mTRTCCloud.muteRemoteAudio(userId, true);
// 특정 원격 사용자의 오디오 스트림 구독 및 재생을 재개합니다
mTRTCCloud.muteRemoteAudio(userId, false);
```

```
// 모든 원격 사용자의 오디오 스트림 구독 및 재생을 일시 중지합니다
mTRTCCloud.muteAllRemoteAudio(true);
// 모든 원격 사용자의 오디오 스트림 구독 및 재생을 재개합니다
mTRTCCloud.muteAllRemoteAudio(false);
```

설명:
반면에, `muteLocalAudio` 는 소프트웨어 측면에서 데이터 스트림을 일시 중지하거나 허용하기만 하면 되므로 효율적이고 원활합니다.자주 마이크를 켜고 끄는 시나리오에 더 적합합니다.

4. 음질 및 음량 유형.

- 음질의 설정

```
// 로컬 오디오 수집 및 게시 시 음질을 설정합니다
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);

// 오디오 스트리밍 중 동적으로 음질 설정합니다
mTRTCCloud.setAudioQuality(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
```

설명:
RTC Engine 사전 설정된 음질은 세 가지 등급(Speech/Default/Music)으로 나뉘며 각각 다른 오디오 매개변수에 해당합니다. 자세한 내용은 TRTCAudioQuality 을 참조하세요.

- 음량 유형의 설정

RTC Engine 각 음질 등급에는 기본 음량 유형이 설정되어 있으며 음량 유형을 강제로 지정하려면 다음 인터페이스를 사용할 수 있습니다.

```
// 음량 유형의 설정
mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeAuto)
;
```

설명:
RTC Engine 음량 유형은 세 가지 등급(VOIP/Auto/Media)으로 나뉘며 각각 다른 음량 채널에 해당합니다. 자세한 내용은 TRTCSystemVolumeType 을 참조하세요.

- 오디오 라우팅의 설정

휴대폰과 같은 모바일 장치에는 일반적으로 스피커와 수화기 두 가지 재생 위치가 있습니다. 오디오 라우팅을 강제로 지정하려면 다음 인터페이스를 사용할 수 있습니다.

```
// 오디오 라우팅의 설정
mTRTCCloud.setAudioRoute(TRTCCloudDef.TRTC_AUDIO_ROUTE_SPEAKER);
```

> **설명:**
> RTC Engine 오디오 라우팅은 두 가지 유형(Speaker/Earpiece)으로 나뉘며, 각각 다른 발음 위치에 해당합니다. 자세한 내용은 TRTCAudioRoute을 참조하세요.

고급 기능

탄막 메시지 인터랙티브

음성 채팅 라이브 방송에는 일반적으로 텍스트 형태의 탄막 메시지 인터랙티브가 있으며, 여기서는 Chat을 통해 그룹 채팅 일반 텍스트 메시지를 보내고 받는 방식으로 구현할 수 있습니다.

```
// 공개 탄막 메시지의 전송
V2TIMManager.getInstance().sendGroupTextMessage(text, groupID,
V2TIMMessage.V2TIM_PRIORITY_NORMAL, new V2TIMValueCallback<V2TIMMessage>
() {
    @Override
    public void onError(int i, String s) {
        // 탄막 메시지의 전송이 실패됩니다
    }

    @Override
    public void onSuccess(V2TIMMessage v2TIMMessage) {
        // 탄막 메시지의 전송이 성공됩니다
    }
});

// 공개 탄막 메시지의 수신
V2TIMManager.getInstance().addSimpleMsgListener(new
V2TIMSimpleMsgListener() {
    @Override
    public void onRecvGroupTextMessage(String msgID, String groupID,
V2TIMGroupMemberInfo sender, String text) {
        Log.i(TAG, sender.getNickName + ": " + text);
```

```
    }
});
```

볼륨 크기 콜백

RTC Engine 정해진 주기로 마이크에 연결된 스트리머의 음량 크기를 콜백할 수 있으며 일반적으로 음파나 음량을 표시하거나 발언 중인 스트리머를 알리는 데 사용됩니다.

```
// 볼륨 크기 콜백 활성화하고 방 입장 성공 후 즉시 활성화하는 것을 권장합니다
// interval: 콜백 간격(ms); enable_vad: 인간 음성  감지 활성화 여부
mTRTCCloud.enableAudioVolumeEvaluation(int interval, boolean
enable_vad);

private class TRTCCloudImplListener extends TRTCCloudListener {
    public void onUserVoiceVolume(ArrayList<TRTCCloudDef.TRTCVolumeInfo>
userVolumes, int totalVolume) {
        super.onUserVoiceVolume(userVolumes, totalVolume);
        // userVolumes는 로컬 사용자 및 원격 스트리밍 사용자를 포함한 모든 말하
는 사용자의 볼륨 크기를 표시하는 데 사용됩니다
        // totalVolume는 원격 스트리밍 사용자 중 최대 볼륨 값을 피드백하는 데 사
용됩니다
        ...
        // 볼륨 크기에 따라 UI에 해당하는 음파 디스플레이를 표시합니다.
        ...
    }
}
```

⚠ 주의:

- 인간 음성 감지는 로컬 인간 음성 감지 결과만 피드백하며 사용자에게 마이크 켜기를 알리기 위해 자신의 역할이 반드시 스트리머여야 합니다.
- `userVolumes` 는 한 배열리고 배열의 각 요소에서 userId가 자신일 경우 로컬 마이크에서 수집된 볼륨 크기를 나타내고 다른 경우는 원격 사용자의 볼륨 크기를 나타냅니다.

음악 및 음향 효과의 재생

배경 음악 및 음향 효과 재생은 음성 채팅방 시나리오에서 자주 사용되는 것입니다. 아래에서는 일반적으로 사용되는 배경 음악 관련 인터페이스의 사용 방법과 주의 사항에 대해 설명합니다.

1. 재생 시작/중지/일시 정지/재개.

```
// 배경 음악, 짧은 음향 효과 및 인간음성 이펙트를 설정하는 데 사용되는 관리 클래
스 얻습니다
TXAudioEffectManager mTXAudioEffectManager =
mTRTCCloud.getAudioEffectManager();

TXAudioEffectManager.AudioMusicParam param = new
TXAudioEffectManager.AudioMusicParam(musicID, musicPath);
// 음악을 원격으로 공유할지?(그렇지 않으면 로컬에서만 재생)
param.publish = true;
// 재생 중인 파일이 짧은 음향 효과 파일인지?
param.isShortFile = false;

// 배경 음악의 재생 시작합니다
mTXAudioEffectManager.startPlayMusic(param);
// 배경 음악의 재생 중지합니다
mTXAudioEffectManager.stopPlayMusic(musicID);
// 배경 음악의 재생을 일시 정지합니다
mTXAudioEffectManager.pausePlayMusic(musicID);
// 배경 음악의 재생 재개합니다
mTXAudioEffectManager.resumePlayMusic(musicID);
```

⚠ 주의:

- RTC Engine 여러 음악을 동시에 재생할 수 있으며 musicID로 고유하게 식별됩니다. 동시에 한 곡만 재생하려면 새 음악을 재생하기 전에 다른 음악을 중지하거나 동일한 musicID를 사용하여 다른 음악을 재생할 수 있습니다. 이 경우 SDK는 기존 음악을 먼저 중지한 후 새 음악을 재생합니다.
- RTC Engine 로컬 및 네트워크 오디오 파일 재생을 지원하며 `musicPath` 를 통해 로컬 절대 경로 또는 URL 주소로 전달할 수 있습니다. MP3/AAC/M4A/WAV 형식을 지원합니다.

2. 음악 및 인간 음성 볼륨 비율을 조정합니다.

```
// 특정 백그라운드 음악의 로컬 재생 볼륨 크기를 설정합니다
mTXAudioEffectManager.setMusicPlayoutVolume(musicID, volume);
// 특정 백그라운드 음악의 원격 재생 볼륨 크기를 설정합니다
mTXAudioEffectManager.setMusicPublishVolume(musicID, volume);
// 모든 백그라운드 음악의 로컬 볼륨 및 원격 볼륨 크기를 설정합니다
mTXAudioEffectManager.setAllMusicVolume(volume);
// 인간 음성의 수집 음량 크기를 설정합니다
mTXAudioEffectManager.setVoiceCaptureVolume(volume);
```

⚠ 주의:

- 볼륨 값 volume의 정상 범위는 0–100이며 기본값은 60입니다. 최대 150까지 설정할 수 있지만 폭음 위험이 있습니다.
- 배경 음악이 인간 음성보다 큰 경우 음악 재생 볼륨을 적절히 낮추고 인간 음성의 수집 볼륨을 높일 수 있습니다.
- **마이크 끄지만 배경 음악 여전히 재생하기**: `muteLocalAudio(true)` 아니고 `setVoiceCaptureVolume(0)` 를 사용하세요.

3. 음악 재생 이벤트 콜백을 설정합니다.

```
mTXAudioEffectManager.setMusicObserver(mCurPlayMusicId, new
TXAudioEffectManager.TXMusicPlayObserver() {
    @Override
    // 배경 음악 재생을 시작합니다
    public void onStart(int id, int errCode) {
        // -4001: 경로 열기 실패
        // -4002: 디코딩 실패
        // -4003: URL 주소가 무효합니다
        // -4004: 재생 중지되지 않습니다
        if (errCode < 0) {
            // 재생 실패 후 다시 재생하기 전에 현재 음악 재생을 먼저 중지해야
합니다.
            mTXAudioEffectManager.stopPlayMusic(id);
        }
    }

    @Override
    // 배경 음악의 재생 진도
    public void onPlayProgress(int id, long curPtsMs, long
durationMs) {
        // curPtsMS 현재 재생 시간(밀리초)
        // durationMs 현재 음악의 총 시간(밀리초)
    }

    @Override
    // 배경 음악 재생이 완료되었습니다
    public void onComplete(int id, int errCode) {
```

```
        // 약한 네트워크로 인해 재생 중간에 실패한 경우에도 이 콜백이 발생하며
이때 errCode < 0입니다
        // 중간에 재생을 일시 중지하거나 중단해도 onComplete 콜백이 발생하지
않습니다
    }
});
```

⚠ 주의:

- 배경 음악 재생 전에 재생 이벤트 콜백을 설정하여 재생 진행 상황을 인지할 수 있도록 이 인터페이스를 사용하십시오.
- MusicId를 재사용할 필요가 없는 경우 재생 완료 후 `setMusicObserver(musicId, null)` 을 수행하여 Observer를 완전히 릴리스할 수 있습니다.

4. 배경 음악 및 음향 효과를 반복 재생합니다

- 방안 1: `AudioMusicParam` 의 `loopCount` 매개변수를 사용하여 반복 재생 횟수를 설정합니다.
값의 범위는 0부터 임의의 양의 정수입니다. 기본값은 0이고 0은 음악을 한 번 재생함을 의미하고, 1은 음악을 두 번 재생함을 의미하며 이와 같이 반복됩니다.

```
private void startPlayMusic(int id, String path, int loopCount) {
    TXAudioEffectManager.AudioMusicParam param = new
TXAudioEffectManager.AudioMusicParam(id, path);
    // 음악을 원격으로 공유할지?
    param.publish = true;
    // 재생 중인 파일이 짧은 음향 효과 파일인지?
    param.isShortFile = true;
    // 반복 재생 횟수를 설정하고 음수는 무한 반복을 의미합니다
    param.loopCount = loopCount < 0 ? Integer.MAX_VALUE : loopCount;
    mTRTCCloud.getAudioEffectManager().startPlayMusic(param);
}
```

⚠ 주의:
방안 1은 매번 반복 재생이 끝나도 `onComplete` 콜백이 트리거되지 않으며 설정된 반복 횟수가 모두 재생된 후에야 해당 콜백이 트리거됩니다.

- 방안 2: "배경 음악이 이미 재생 완료됨" 이벤트 콜백 `onComplete` 을 통해 반복 재생을 구현하며, 일반적으로 리스트 반복 또는 단일 곡 반복에 사용됩니다.

```
// 반복 재생 여부를 나타내는 멤버 변수입니다
private boolean loopPlay;

private void startPlayMusic(int id, String path) {
    TXAudioEffectManager.AudioMusicParam param = new
TXAudioEffectManager.AudioMusicParam(id, path);
    mTXAudioEffectManager.setMusicObserver(id, new
MusicPlayObserver(id, path));
    mTXAudioEffectManager.startPlayMusic(param);
}

private class MusicPlayObserver implements
TXAudioEffectManager.TXMusicPlayObserver {
    private final int mId;
    private final String mPath;

    public MusicPlayObserver(int id, String path) {
        mId = id;
        mPath = path;
    }

    @Override
    public void onStart(int i, int i1) {

    }

    @Override
    public void onPlayProgress(int i, long l, long l1) {

    }

    @Override
    public void onComplete(int i, int i1) {
        mTXAudioEffectManager.stopPlayMusic(i);
        if (i1 >= 0 && loopPlay) {
            // 여기서 반복 재생 리스트 음악의 ID, Path를 대체할 수 있습니다
            startPlayMusic(mId, mPath);
        }
    }
```

```
}
```

혼합 스트리밍 전환 및 푸시백

1. 혼합 스트리밍 푸시백 RTC Engine 방.

```
private void startPublishMediaToRoom(String roomId, String userId) {
    // TRTCPublishTarget 객체 생성
    TRTCCloudDef.TRTCPublishTarget target = new
TRTCCloudDef.TRTCPublishTarget();
    // 혼합 스트리밍 후 방으로 푸시백하기
    target.mode = TRTCCloudDef.TRTC_PublishMixStream_ToRoom;
    target.mixStreamIdentity.strRoomId = roomId;
    // 혼합 스트리밍 로봇의 userid는 방 내 다른 사용자의 userid와 중복될 수 없
습니다
    target.mixStreamIdentity.userId = userId + MIX_ROBOT;

    // 트랜스코딩된 오디오 스트림의 인코딩 매개변수를 설정합니다(자체 정의 가능)
    TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new
TRTCCloudDef.TRTCStreamEncoderParam();
    trtcStreamEncoderParam.audioEncodedChannelNum = 2;
    trtcStreamEncoderParam.audioEncodedKbps = 64;
    trtcStreamEncoderParam.audioEncodedCodecType = 2;
    trtcStreamEncoderParam.audioEncodedSampleRate = 48000;

    // 트랜스코딩된 비디오 스트림의 인코딩 매개변수를 설정합니다(순수 오디오 혼합
스트리밍은 무시해도 됩니다)
    trtcStreamEncoderParam.videoEncodedFPS = 15;
    trtcStreamEncoderParam.videoEncodedGOP = 3;
    trtcStreamEncoderParam.videoEncodedKbps = 30;
    trtcStreamEncoderParam.videoEncodedWidth = 64;
    trtcStreamEncoderParam.videoEncodedHeight = 64;

    // 오디오/비디오 혼합 스트리밍 매개변수를 설정합니다
    TRTCCloudDef.TRTCStreamMixingConfig trtcStreamMixingConfig = new
TRTCCloudDef.TRTCStreamMixingConfig();
    // 기본적으로 빈 값을 입력하면 방의 모든 오디오가 믹싱됨을 의미합니다
    trtcStreamMixingConfig.audioMixUserList = null;
```

```
    // 비디오 혼합 스트리밍 템플릿을 설정합니다(오디오 혼합 스트리밍은 무시해도
됩니다)
    TRTCCloudDef.TRTCVideoLayout videoLayout = new
TRTCCloudDef.TRTCVideoLayout();
    trtcStreamMixingConfig.videoLayoutList.add(videoLayout);

    // 혼합 스트리밍 푸시백 시작합니다
    mTRTCCloud.startPublishMediaStream(target,
trtcStreamEncoderParam, trtcStreamMixingConfig);
}
```

2. 이벤트 콜백 및 업데이트 작업을 중지합니다

- 작업 결과 이벤트의 콜백.

```
private class TRTCCloudImplListener extends TRTCCloudListener {
    @Override
    public void onStartPublishMediaStream(String taskId, int code,
String message, Bundle extraInfo) {
        // taskId: 요청이 성공하면 RTC Engine 백엔드는 콜백에서 이 작업의
taskId를 제공하며 이후 이 taskId를 updatePublishMediaStream 및
stopPublishMediaStream과 함께 사용하여 업데이트 및 중지할 수 있습니다.
        // code: 콜백 결과이고  0은 성공을 나타내며 그 외의 값은 실패를 나타냅
니다.
    }

    @Override
    public void onUpdatePublishMediaStream(String taskId, int code,
String message, Bundle extraInfo) {
        // 미디어 스트림 발행 인터페이스(updatePublishMediaStream)를 호출할
때 전달한 taskId가 이 콜백을 통해 다시 전달되며 이 콜백이 어떤 업데이트 요청에
속하는지 식별하는 데 사용됩니다.
        // code: 콜백 결과이고  0은 성공을 나타내며 그 외의 값은 실패를 나타냅
니다.
    }

    @Override
    public void onStopPublishMediaStream(String taskId, int code,
String message, Bundle extraInfo) {
```

```
        // 미디어 스트림 발행 중지(stopPublishMediaStream)를 호출할 때 전달
한 taskId가 이 콜백을 통해 다시 전달되며 이 콜백이 어떤 중지 요청에 속하는지 식
별하는 데 사용됩니다.
        // code: 콜백 결과이고  0은 성공을 나타내며 그 외의 값은 실패를 나타냅
니다.
    }
}
```

- 미디어 스트림 발행의 업데이트

  이 인터페이스는 RTC Engine 서버에 명령을 보내고 `startPublishMediaStream` 로 시작된 미디어 스트림을 업데이트합니다.

```
// taskId: onStartPublishMediaStream 콜백의 작업 ID입니다
// target: 예를 들어 발행된 CDN URL 추가 또는 삭제합니다
// params: 생 측의 스트림 중단을 방지하기 위해 미디어 스트림 인코딩 출력 매개변
수를 일치시키는 것을 권장합니다
// config: 예를 들어 크로스 룸 PK와 같은 혼합 트랜스코딩에 참여하는 사용자 목록
을 업데이트하는 것입니다
mTRTCCloud.updatePublishMediaStream(taskId, target,
trtcStreamEncoderParam, trtcStreamMixingConfig);
```

> ⚠ 주의:
> 동일한 작업은 순수 오디오, 오디오/비디오, 순수 비디오 간의 전환을 지원하지 않습니다.

- 미디어 스트림 게시를 중지합니다

  이 인터페이스는 RTC Engine 서버에 명령을 보내고 `startPublishMediaStream` 로 시작된 미디어 스트림을 중지합니다.

```
// taskId: onStartPublishMediaStream 콜백의 작업 ID입니다
mTRTCCloud.stopPublishMediaStream(taskId);
```

> ⚠ 주의:
> taskId가 빈 문자열이면 해당 사용자의 `startPublishMediaStream` 로 시작한 모든 미디어 스트림이 중지됩니다. 하나의 미디어 스트림만 시작했거나 손님을 통해 시작된 모든 미디어 스트림을 중지하려는 경우 이 방법을 사용하는 것이 좋습니다.

네트워크 품질 실시간 콜백

`onNetworkQuality` 를 통해 로컬 및 원격 사용자의 네트워크 품질을 실시간으로 모니터링할 수 있으며 이 콜백은 2초마다 한 번씩 발생합니다.

```
private class TRTCCloudImplListener extends TRTCCloudListener {
    @Override
    public void onNetworkQuality(TRTCCloudDef.TRTCQuality localQuality,
                                 ArrayList<TRTCCloudDef.TRTCQuality>
remoteQuality) {
        // localQuality userId가 비어 있으면 로컬 사용자의 네트워크 품질 평가
결과를 나타냅니다.
        // remoteQuality는 원격 사용자의 네트워크 품질 평가 결과를 나타내며 그
결과는 원격 및 로컬의 영향을 받습니다.
        switch (localQuality.quality) {
            case TRTCCloudDef.TRTC_QUALITY_Excellent:
                Log.i(TAG, "현재 네트워크 상태가 매우 좋습니다");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Good:
                Log.i(TAG, "현재 네트워크 상태가 비교적 좋습니다");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Poor:
                Log.i(TAG, "현재 네트워크 상태가 일반입니다");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Bad:
                Log.i(TAG, "현재 네트워크 상태가 좋지 않습니다");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Vbad:
                Log.i(TAG, "현재 네트워크 상태가 매우 나쁩니다");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Down:
                Log.i(TAG, "현재 네트워크가 TRTC 최소 요구 사항을 충족하지 못합
니다");
                break;
            default:
                Log.i(TAG, "정의되지 않음");
                break;
        }
    }
}
```

고급 권한의 관리

RTC Engine 고급 권한 관리는 다양한 방에 대해 서로 다른 입장 권한을 설정할 수 있습니다.예를 들어 고급 VIP 방과 같은 경우. 또한 청취자가 마이크 사용 권한을 관리하는 데 사용할 수 있습니다.예를 들어 유령 마이크를 처리하는 경우.
단계1: RTC Engine 콘솔의 앱 기능 설정 페이지에서 고급 권한 관리 스위치를 켭니다.
단계2: 업무 백엔드에서 PrivateMapKey 생성하고 코드 예시는 PrivateMapKey 계산원 코드을 참조하세요
단계3: 방 입장 검증 및 마이크 연결 검증 PrivateMapKey

- 입장 검증

```
TRTCCloudDef.TRTCParams mTRTCParams = new TRTCCloudDef.TRTCParams();
mTRTCParams.sdkAppId = SDKAPPID;
mTRTCParams.userId = mUserId;
mTRTCParams.strRoomId = mRoomId;
// 업무 백엔드에서 가져온 UserSig
mTRTCParams.userSig = getUserSig();
// 업무 백엔드에서 가져온 PrivateMapKey
mTRTCParams.privateMapKey = getPrivateMapKey();
mTRTCParams.role = TRTCCloudDef.TRTCRoleAudience;
mTRTCCloud.enterRoom(mTRTCParams,
TRTCCloudDef.TRTC_APP_SCENE_VOICE_CHATROOM);
```

- 마이크 연결 검증

```
// 업무 백엔드에서 가져온 최신 PrivateMapKey를 역할 전환 인터페이스로 전달합니
다
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor,
getPrivateMapKey());
```

이상 처리

고장및 오류 처리

RTC Engine SDK에서 복구할 수 없는 오류가 발생하면 `onError` 콜백에서 나오며, 자세한 내용은 오류 코드표 을 참조하십시오.

- UserSig 관련

  UserSig 검증 실패로 인해 방 입장에 실패할 수 있으며 UserSig 도구를 사용하여 검증할 수 있습니다.

열거형	값	설명

ERR_TRTC_INVALID_USER_SIG	-3320	방 입장 매개변수 userSig가 올바르지 않습니다. `TRTCParams.userSig` 이 비어 있는지 확인하세요.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	UserSig 검증에 실패했습니다. 매개변수 `TRTCParams.userSig` 이 올바르게 입력되었거나 만료되었는지 확인하세요.

- 방 입장/퇴장 관련

  방 입장 실패 시 먼저 방 입장 매개변수가 올바른지 확인하고 방 입장 및 퇴장 인터페이스는 반드시 쌍으로 호출해야 합니다. 방 입장에 실패한 경우에도 퇴장 인터페이스를 호출해야 합니다.

열거형	값	설명
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	입장 요청이 시간 초과했습니다, 네트워크 연결이 끊겼는지 또는 VPN이 켜져 있는지 확인하세요. 4G로 전환하여 테스트할 수도 있습니다.
ERR_TRTC_INVALID_SDK_APPID	-3317	입장 매개변수 sdkAppId가 잘못되었습니다. `TRTCParams.sdkAppId` 이 비어 있는지 확인하세요.
ERR_TRTC_INVALID_ROOM_ID	-3318	입장 매개변수 roomId가 잘못되었습니다. `TRTCParams.roomId` 또는 `TRTCParams.strRoomId` 이 비어 있는지 확인하세요. roomId와 strRoomId는 혼용할 수 없습니다.
ERR_TRTC_INVALID_USER_ID	-3319	입장 매개변수 userId가 올바르지 않습니다. `TRTCParams.userId` 이 비어 있는지 확인하세요.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	입장 요청이 거부되었습니다. `enterRoom` 을 연속적으로 호출하여 동일한 ID의 방에 입장했는지 확인하세요.

- 장치 관련

  장치 관련 오류를 감지할 수 있으며 관련 오류 발생 시 UI에서 사용자에게 알립니다.

열거형	값	설명
ERR_MIC_START_FAIL	-1302	Windows 또는 Mac 장치에서 마이크 구성 프로그램(드라이버)이 비정상일 경우, 마이크 열 수가 없습니다. 장치를 비활성화한 후 다시 활성화하거나, 기기를 재시작하거나, 구성 프로그램을 업데이트하세요.
ERR_SPEAKER_START_FAIL	-1321	스피커 열기 실패했습니다. 예를 들어 Windows 또는 Mac 장치에서 스피커 구성 프로그램(드라이버)이 이상이 있는 경우, 장치를 비활

		성화한 후 다시 활성화하거나 기기를 재시작하거나 구성 프로그램을 업데이트하세요.
ERR_MIC_OCCUPY	-1 31 9	마이크가 사용 중입니다. 예를 들어 모바일 장치에서 통화 중일 때 마이크를 열 수가 없습니다.

비정상 종료 처리

1. 네트워크 끊김 감지 및 타임아웃 퇴장.

   다음 콜백을 통해 Real-Time Communication Engine (RTC Engine)의 네트워크 연결 끊김 및 재연결 이벤트 알림을 모니터링할 수 있습니다.
   `onConnectionLost` 콜백을 수신한 후 로컬 마이크 순위 UI에 네트워크 끊김 표시를 표시하여 사용자에게 알릴 수 있으며 동시에 로컬에서 타이머를 시작합니다. 설정된 시간의 임계값을 초과해도 `onConnectionRecovery` 콜백을 수신하지 못하면 네트워크가 계속 끊긴 상태입니다. 이때 로컬에서 마이크 종료 및 퇴장 절차를 시작하며 동시에 팝업으로 사용자에게 방을 나갔음을 알리고 페이지를 파기합니다. 네트워크가 90초(기본값) 이상 끊긴 경우 타임아웃 퇴장이 트리거되며, RTC Engine 서버는 해당 사용자를 방에서 강제 퇴장시킵니다. 해당 사용자가 스트리머 역할인 경우 방 내 다른 사용자는 `onRemoteUserLeaveRoom` 콜백을 수신합니다.

```
private class TRTCCloudImplListener extends TRTCCloudListener {
    @Override
    public void onConnectionLost() {
        // SDK와 클라우드 간의 연결이 끊어졌습니다.
    }

    @Override
    public void onTryToReconnect() {
        // SDK가 클라우드에 재연결을 시도 중입니다.
    }

    @Override
    public void onConnectionRecovery() {
        // SDK가 클라우드와의 연결이 복구되었습니다.
    }
}
```

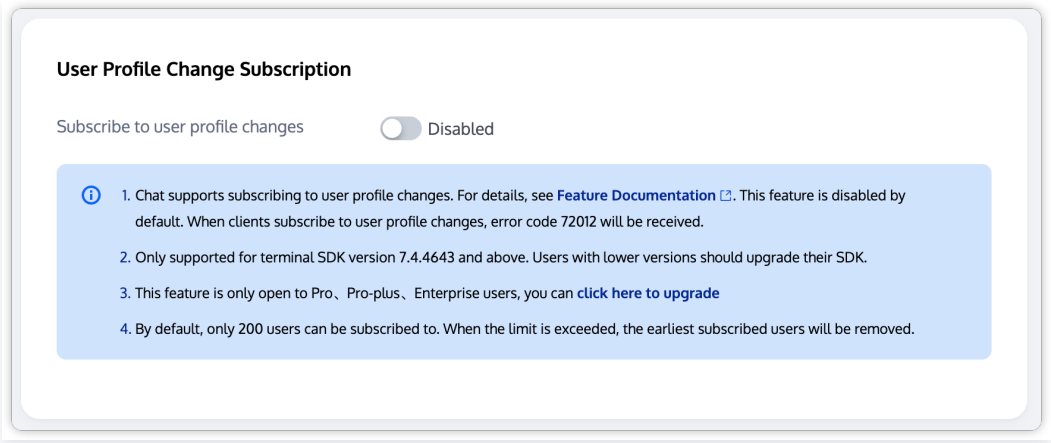
2. 오프라인 상태에서 자동으로 마이크의 사용을 종료합니다.

   Chat 사용자의 일반 상태는 온라인(ONLINE), 오프라인(OFFLINE), 미로그인(UNLOGGED)으로 구분되며 오프라인 상태는 일반적으로 사용자가 프로세스를 강제 종료하거나 네트워크가 비정상적으로 중단된 경우에 발생

합니다. 스트리머 구독을 통해 마이크 연결된 청취자 상태를 모니터링하여 오프라인 상태의 마이크 연결 청취자를 감지하고 강제 퇴장시킬 수 있습니다.

```
// 스트리머 구독을 통해 마이크 연결 청취자 상태를 감지합니다
V2TIMManager.getInstance().subscribeUserStatus(userList, new
V2TIMCallback() {
    @Override
    public void onSuccess() {
        // 사용자 상태의 구독이 성공됩니다
    }

    @Override
    public void onError(int code, String message) {
        // 사용자 상태의 구독이 실패됩니다
    }
});

// 스트리머가 마이크 해제 청취자 상태를 구독하는 것을 취소합니다
V2TIMManager.getInstance().unsubscribeUserStatus(userList, new
V2TIMCallback() {
    @Override
    public void onSuccess() {
        // 사용자 상태의 구독을 취소하는 것이 성공됩니다
    }

    @Override
    public void onError(int code, String message) {
        // 사용지 상태의 구독을 취소하는 것이 실패됩니다
    }
});

// 사용자 상태 변경의 알림 및 처리
V2TIMManager.getInstance().addIMSDKListener(new V2TIMSDKListener() {
    @Override
    public void onUserStatusChanged(List<V2TIMUserStatus>
userStatusList) {
        for (V2TIMUserStatus userStatus : userStatusList) {
            final String userId = userStatus.getUserID();
            int status = userStatus.getStatusType();
```

```
            if (status == V2TIMUserStatus.V2TIM_USER_STATUS_OFFLINE)
{
                오프라인 상태에서 마이크 끄기를 수행합니다
                kickSeat(getSeatIndexFromUserId(userId));
            }
        }
    }
});
```



⚠ 주의:

- 사용자 상태를 구독하는 것은 프로 버전 패키지로 업그레이드해야 합니다. 자세한 내용은 기본 서비스 상세 정보 을 참조하세요.
- 사용자 상태를 구독하는 것은 사전에 Chat 콘솔 에서 **사용자 상태 조회 및 상태 변경 알림 구성**을 활성화해야 합니다. 활성화되지 않은 경우 `subscribeUserStatus` 호출 시 오류가 발생합니다.

서버에서 사용자 나가게 시키고 방을 해산합니다.

1. 서버에서 사용자 나가게 시킵니다.

   먼저 RTC Engine 서버에서 사용자 나가게 시키는 인터페이스 RemoveUser(정수형 방 번호) 또는 RemoveUserByStrRoomId(문자열 방 번호)를 호출하여 대상 사용자를 RTC Engine 방에서 나가게 시킵니다. 입력 예시는 다음과 같습니다.

```
https://trtc.tencentcloudapi.com/?Action=RemoveUser
&SdkAppId=1400000001
&RoomId=1234
&UserIds.0=test1
&UserIds.1=test2
<공개 요청 매개변수>
```

강제 퇴장시킴이 성공적으로 실행되면 대상 사용자는 클라이언트에서 `onExitRoom()` 콜백을 받으며 `reason` 값은 1입니다. 이때 해당 콜백에서 마이크 끄기, Chat 그룹 퇴장 등의 작업을 처리할 수 있습니다.

```
// RTC Engine 방 퇴장 이벤트의 콜백
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "exitRoom을 호출하여 방에서 나가기");
    } else {
        // reason 1: 서버에 의해 현재 방에서 나가게 됩니다
        // reason 2: 현재 방이 완전히 해산됩니다
        Log.d(TAG, "서버에 의해 방에서 나가게 되거나 현재 방이 해산됩니다");
        // 마이크 끄기
        leaveSeat(seatIndex);
        // IM 그룹 나가기
        quitGroup(groupID, new V2TIMCallback() {});
    }
}
```

2. 서버에서 방을 해산합니다.

   먼저 Chat 서버의 그룹 해산 인터페이스 destroy_group을 호출하여 대상 그룹을 해산합니다. 요청 URL 예시는 다음과 같습니다.

```
https://xxxxxx/v4/group_open_http_svc/destroy_group?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&conten
ttype=json
```

   그룹 해산이 성공적으로 실행되면 대상 그룹의 모든 멤버는 클라이언트에서 `onGroupDismissed()` 콜백을 받게 됩니다. 이때 해당 콜백에서 RTC Engine 방 나가기 등의 작업을 처리할 수 있습니다.

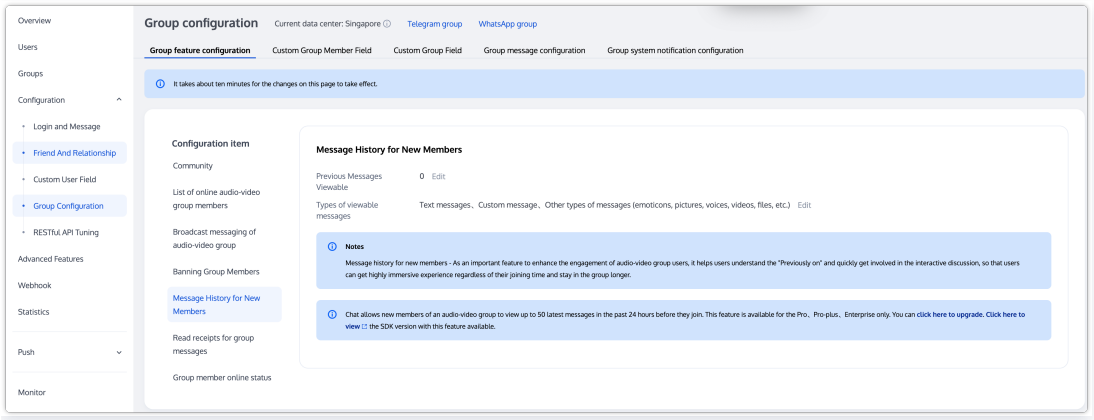
```
// 그룹 해산 콜백
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener()
{
  @Override
  public void onGroupDismissed(String groupID, V2TIMGroupMemberInfo
opUser) {
      // RTC Engine 방 나가기
```

```
        mTRTCCloud.stopLocalAudio();
        mTRTCCloud.exitRoom();
    }
});
```

설명:
방 내 모든 사용자가 `exitRoom()` 을 호출하여 퇴장을 완료하면 RTC Engine 방이 자동으로 해산됩니다. 물론 서버 인터페이스 DismissRoom(정수형 방 번호) 또는 DismissRoomByStrRoomId(문자열 방 번호)를 호출하여 RTC Engine 방을 강제로 해산할 수도 있습니다.

방 입장하여 라이브 방송의 이전 메시지를 확인합니다

AVChatRoom은 기본적으로 라이브 방송의 이전 메시지를 저장하지 않고 새 사용자가 라이브 방송에 입장한 후에는 입장 후 사용자가 보낸 메시지만 볼 수 있습니다. 새로 입장한 사용자의 체험을 최적화하기 위해 콘솔에서 라이브방송 그룹 사용자가 입장 전의 메시지를 구성할 수 있으며, 그림과 같습니다.



라이브방송 그룹 사용자가 입장 전의 메시지를 가져오는 것은 다른 그룹의 이전 메시지를 가져오는 것과 동일하며, 코드 예시는 하기와 같습니다.

```
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG);
// 클라우드의 이전 메시지 가져오기
option.setGetTimeBegin(1640966400);          // 2022-01-01 00:00:00부터 시
작
option.setGetTimePeriod(1 * 24 * 60 * 60);   // 하루 동안의 메시지 가져오기
option.setCount(Integer.MAX_VALUE);          // 시간 범위 내 모든 메시지 가져
오기
option.setGroupID(#you group id#);           // 그룹 채팅 메시지 가져오기
V2TIMManager.getMessageManager().getHistoryMessageList(option, new
V2TIMValueCallback<List<V2TIMMessage>>() {
```

```
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

> ⚠ 주의:
> 이 기능은 프리미엄 버전 사용자만 이용할 수 있으며, 그룹 생성 후 24시간 이내에 최대 20개의 이전의 메시지만 지원합니다.

방 입장 시 스트리머의 무음 상태를 감지합니다

방안 1: 입장 시 모든 스트리머를 기본적으로 무음상태로 설정한 후 `onUserAudioAvailable(userId, true)` 콜백을 통해 해당 스트리머의 무음 상태를 해제합니다.

```
private class TRTCCloudImplListener extends TRTCCloudListener {
    @Override
    public void onUserAudioAvailable(String userId, boolean available) {
        if (available) {
            // 해당 스트리머의 무음 상태를 해제합니다
        }
    }
}
```

방안 2: 스트리머의 뮤음 상태를 RTC Engine 그룹 속성에 저장하고 청취자가 방에 입장하면 전체 그룹 속성을 가져와서 마이크 사용중의 스트리머의 무음 상태를 파악합니다.

```
V2TIMManager.getGroupManager().getGroupAttributes(groupID, null, new
V2TIMValueCallback<Map<String, String>>() {
    @Override
    public void onError(int i, String s) {
        // 그룹 속성 가져오기가 실패됩니다
    }
```

```
    @Override
    public void onSuccess(Map<String, String> attrMap) {
        // 그룹 속성 가져오기가 성공되며 스트리머 무음 상태를 저장하는 키를
muteStatus로 가정합니다
        String muteStatus = attrMap.get("muteStatus");
        // muteStatus를 분석하여 각 마이크를 사용하는 스트리머의 무음 상태를 가져
옵니다
    }
});
```

블루투스 헤드폰의 오디오 입력 및 출력 문제

휴대폰이 블루투스 헤드폰에 성공적으로 연결되었지만 RTC Engine 앱의 오디오 입력 또는 출력은 여전히 휴대폰 마이크 또는 스피커를 사용하고 있습니다.

1. 오디오 출력이 정상적이고 블루투스 헤드폰을 사용하는 데 오디오 입력만 여전히 휴대폰 마이크를 사용하는 경우, 볼륨 유형 설정을 확인하세요. 통화 볼륨 모드에서만 블루투스 헤드폰의 마이크를 통해 소리를 수집할 수 있습니다. 자세한 내용은 오디오 관리 – 음질 및 볼륨 유형을 참조하세요.

   ```
   mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeVOIP)
   ;
   ```

2. 오디오의 입력 및 출력 모두 블루투스 헤드폰을 사용할 수 없는 경우, 앱 권한에서 블루투스 권한이 구성되었는지 확인하십시오. Android 장치의 경우 Android 12 미만 시스템은 최소한 `BLUETOOTH` 권한이 구성되어야 하며, Android 12및 그의 이상 시스템은 최소한 `BLUETOOTH_CONNECT` 권한이 구성되어야 하고 코드에서 동적으로 권한을 신청해야 합니다.

   AndroidManifest.xml에서 블루투스 권한을 구성하고 Android 12 이하 시스템과의 호환성을 위해 다음과 같이 선언하는 것을 권장합니다.

   ```
   <!--일반 권한: 기본 블루투스 연결 권한-->
   <uses-permission android:name="android.permission.BLUETOOTH"
   android:maxSdkVersion="30"/>
   <!--일반 권한: 블루투스 관리, 스캔 권한-->
   <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
   android:maxSdkVersion="30" />

   <!--런타임 권한: Android 12 블루투스 권한 블루투스 장치 검색-->
   <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
   <!--런타임 권한: Android 12 블루투스 권한      현재 장치를 다른 블루투스 장치에
   서 감지할 수 있도록 합니다-->
   ```

```
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE"
/>
<!--런타임 권한: Android 12 블루투스 권한 페어링된 블루투스 장치와 통신 또는 현
재 휴대폰 블루투스 켜짐 여부를 확인합니다-->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT"
/>
```

Android 12및 그의 이상 시스템에 추가된 블루투스 세분화 권한에 대한 동적 요청 방법은 다음과 같습니다.

```
private List<String> permissionList = new ArrayList<>();

protected void initPermission() {
    // Android SDK 버전을 판단하고 Android 12 이상으로 되어야 합니다.
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        // 필요한 경우 동적으로 요청할 권한을 추가합니다
        permissionList.add(Manifest.permission.BLUETOOTH_SCAN);
        permissionList.add(Manifest.permission.BLUETOOTH_ADVERTISE);
        permissionList.add(Manifest.permission.BLUETOOTH_CONNECT);
    }
    if (permissionList.size() != 0) {
        // 동적으로 권한 요청
        ActivityCompat.requestPermissions(this,
permissionList.toArray(new String[0]), REQ_PERMISSION_CODE);
    }
}
```

음악 재생을 지원하는 리소스 경로 문제

RTC Engine SDK API `startPlayMusic` 를 사용하여 배경 음악을 재생할 때, 음악 리소스 경로 매개변수 `path` 에 Android 개발의 assets/raw 등 애플리케이션 리소스 파일을 저장하는 디렉터리의 파일 경로를 전달할 수 없습니다. 이러한 디렉터리의 파일은 APK로 패키징되어 설치 후 휴대폰 파일 시스템에 압축 해제되지 않기 때문입니다. 현재는 네트워크 리소스 URL, Android 장치 외부 저장소 및 애플리케이션 개인 디렉터리에 있는 리소스 파일의 절대 경로만 전달할 수 있습니다.
assets 디렉터리에 있는 리소스 파일을 미리서 복사해서 장치 외부 저장소 또는 애플리케이션 개인 디렉터리에 넣는 방법으로 이 문제를 해결할 수 있습니다. 예시 코드는 다음과 같습니다.

```
public static void copyAssetsToFile(Context context, String name) {
    // 애플리케이션 자체 디렉터리의 files 디렉터리
```

```
    String savePath = ContextCompat.getExternalFilesDirs(context, null)
[0].getAbsolutePath();
    // 애플리케이션 자체 디렉터리의 cache 디렉터리
    // String savePath =
getApplication().getExternalCacheDir().getAbsolutePath();
    // 애플리케이션의 개인 저장 디렉터리의 files 디렉터리
    // String savePath =
getApplication().getFilesDir().getAbsolutePath();
    String filename = savePath + "/" + name;
    File dir = new File(savePath);
    // 디렉터리가 존재하지 않으면 이 디렉터리를 생성하세요.
    if (!dir.exists()) {
        dir.mkdir();
    }
    try {
        if (!(new File(filename)).exists()) {
            InputStream is =
context.getResources().getAssets().open(name);
            FileOutputStream fos = new FileOutputStream(filename);
            byte[] buffer = new byte[1024];
            int count = 0;
            while ((count = is.read(buffer)) > 0) {
                fos.write(buffer, 0, count);
            }
            fos.close();
            is.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

- 애플리케이션 개인 저장 files 디렉터리 경로: `/data/user/0/<package_name>/files/<file_name>`
- 애플리케이션 외부 저장 files 디렉터리 경로:
  `/storage/emulated/0/Android/data/<package_name>/files/<file_name>`
- 애플리케이션 외부 저장 cache 디렉터리 경로:
  `/storage/emulated/0/Android/data/<package_name>/cache/<file_name>`

⚠ 주의:

- 전달한 경로가 애플리케이션 자체의 특정 디렉터리가 아닌 다른 외부 저장 경로인 경우, Android 10 및 그의 이상 기기에서 리소스 액세스가 거부될 수 있습니다. 이는 Google이 도입한 새로운 저장소 관리 시스템인 분할 저장소 때문입니다. AndroidManifest.xml 파일의 <application> 태그 내에 다음 코드를 추가하여 일시적으로 이 문제를 회피할 수 있습니다(`android:requestLegacyExternalStorage="true"` .) 이 속성은 targetSdkVersion이 29(Android 10)인 애플리케이션에서만 유효하며 더 높은 버전의 targetSdkVersion을 가진 애플리케이션의 경우 여전히 애플리케이션의 개인 또는 외부 저장 경로를 사용하는 것을 권장합니다.
- RTC Engine SDK 11.5 미 그의 이상 버전의 Content Provider 컴포넌트의 Content URI를 사용하여 Android 기기의 로컬 음악 리소스를 재생합니다.
- Android 11 및 HarmonyOS 3.0 이상 시스템에서 외부 저장 디렉터리의 리소스 파일에 액세스할 수 없는 경우 `MANAGE_EXTERNAL_STORAGE` 권한을 신청해야 합니다.
  - 먼저, 앱의 AndroidManifest 파일에 다음 항목을 추가해야 합니다.

```
<manifest ...>
    <!-- This is the permission itself -->
    <uses-permission
android:name="android.permission.MANAGE_EXTERNAL_STORAGE" />

    <application ...>
        ...
    </application>
</manifest>
```

  - 다음에 앱에서 이 권한이 필요한 경우 사용자에게 수동으로 권한을 부여하도록 안내하세요.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
    if (!Environment.isExternalStorageManager()) {
        Intent intent = new
Intent(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSIO
N);
        Uri uri = Uri.fromParts("package", getPackageName(),
null);
        intent.setData(uri);
        startActivity(intent);
    }
} else {
    // For Android versions less than Android 11, you can
use the old permissions model
```

```
    ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.WRITE_EXTERNAL_STORAGE}, REQUEST_CODE);
}
```

iOS

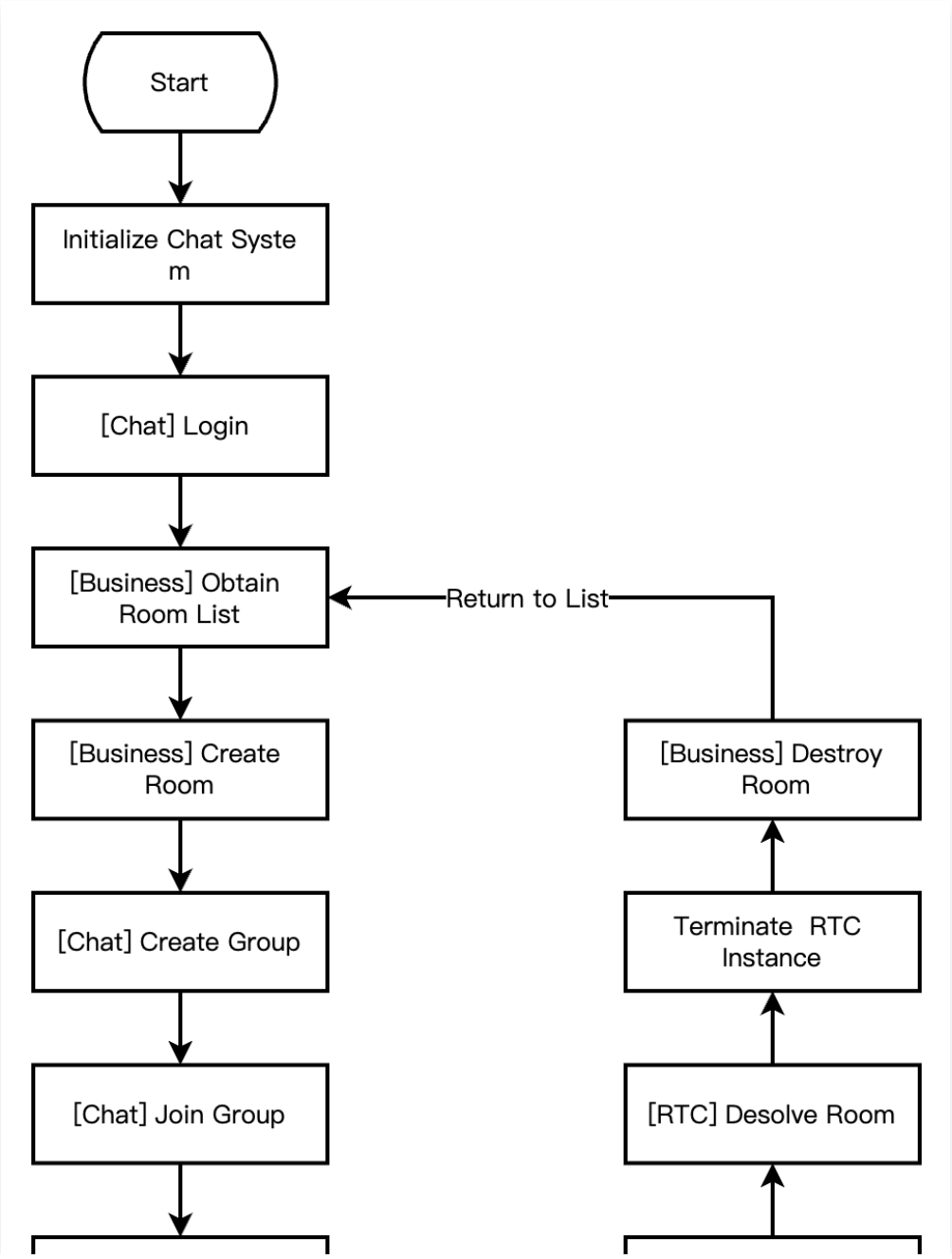
최종 업데이트 날짜:: 2025-10-28 11:32:37

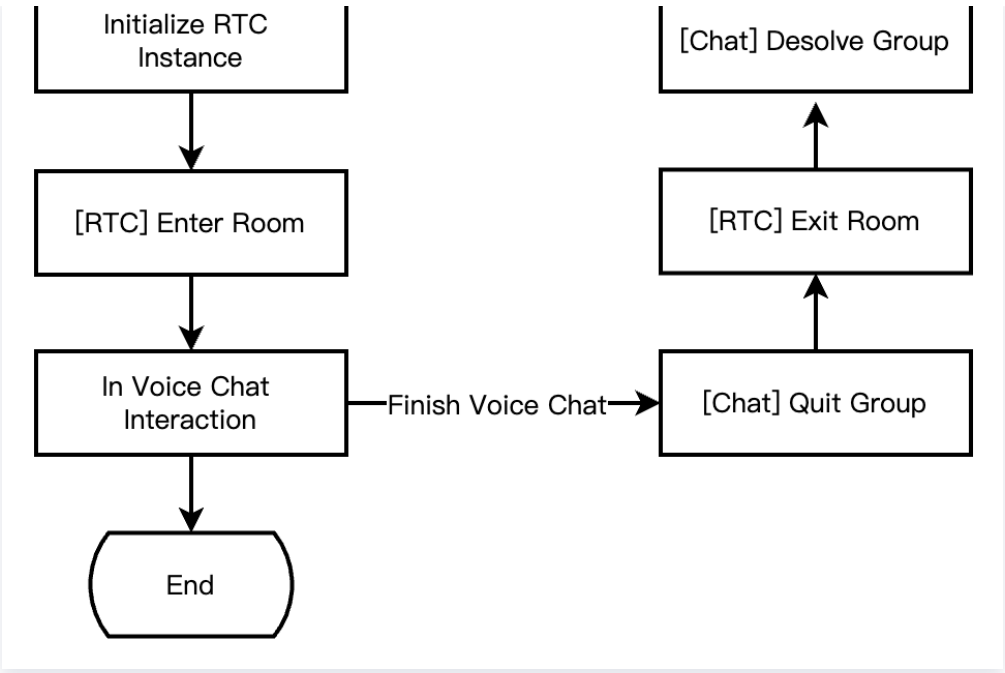
업무 프로세스

본 섹션에서는 음성 채팅방의 몇 가지 일반적인 업무 프로세스를 요약하여 전체 시나리오 구현 프로세스를 더 잘 이해할 수 있도록 도와줍니다.

방 관리 프로세스

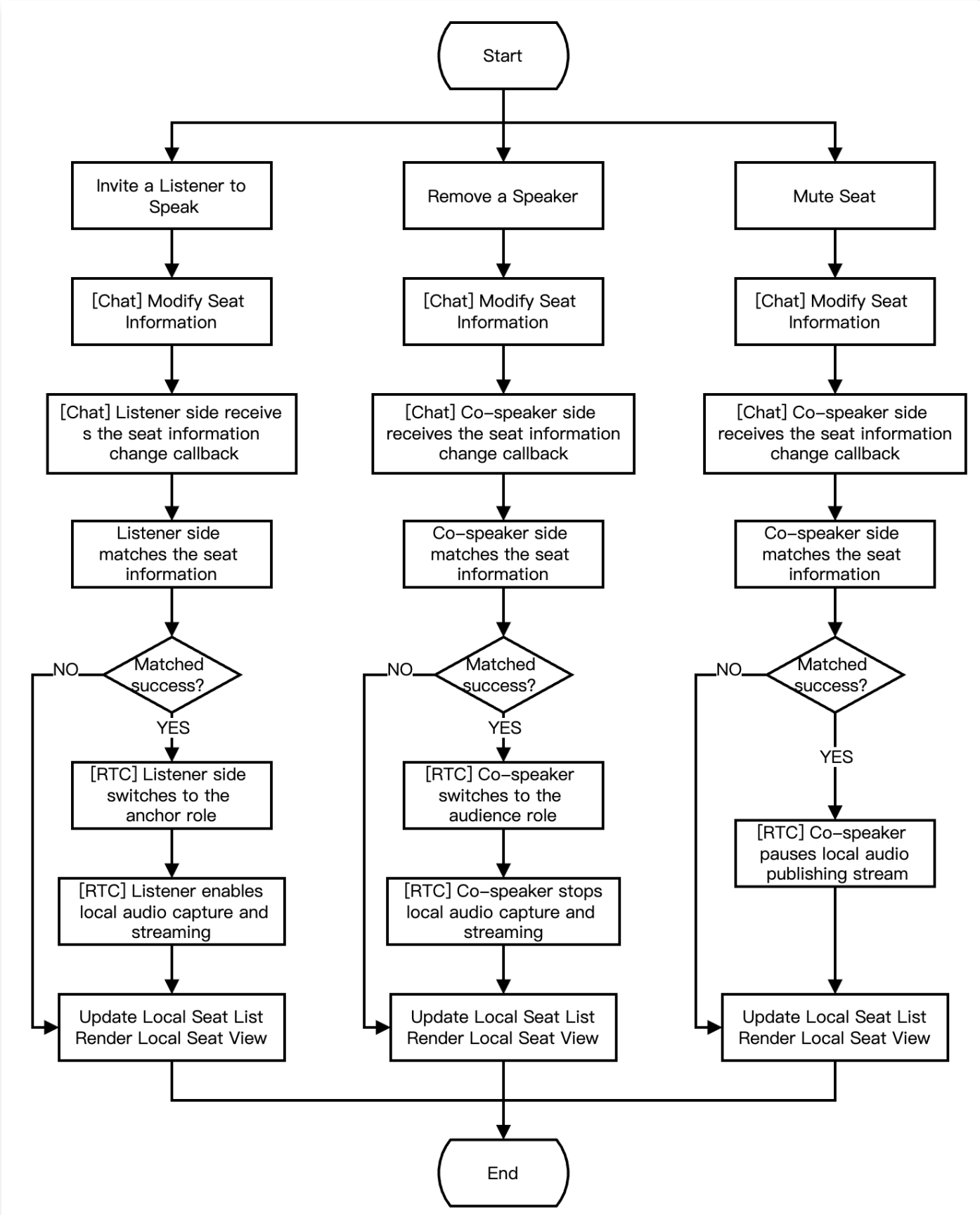
아래 그림은 방 생성, 입장, 퇴장, 해산 등의 구현 프로세스를 포함한 방 관리 프로세스를 보여줍니다.





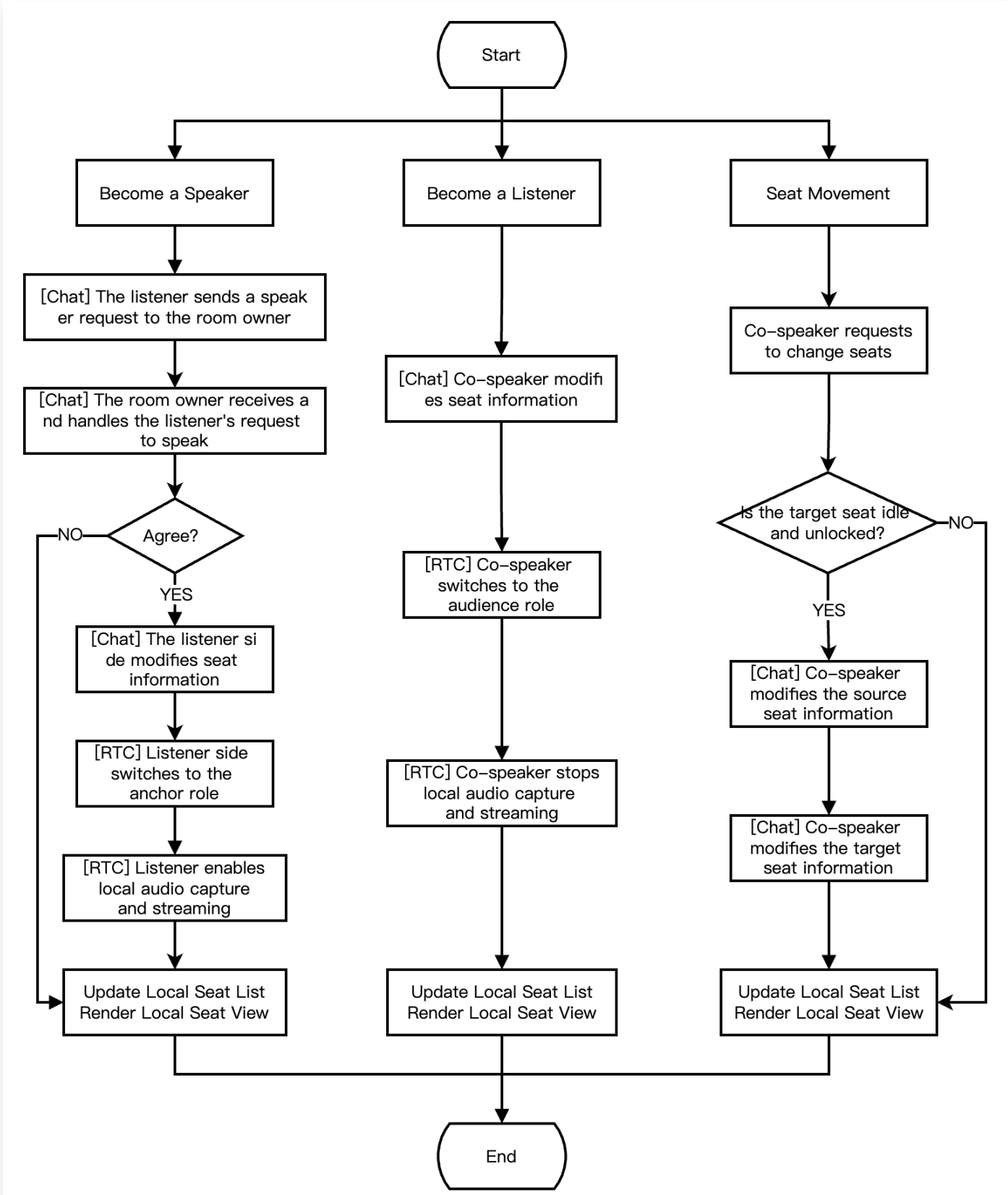
방장 마이크 순위 관리 프로세스

아래 그림은 방주인이 마이크를 관리하는 프로세스를 보여줍니다. 마이크 사용 초청, 마이크 사용 강제 끄기, 마이크 금언등의 구현 프로세스를 포함합니다.



청취자 마이크 순위 관리 프로세스

아래 그림은 청취자가 마이크를 관리하는 프로세스를 보여줍니다. 마이크 자동 사용, 마이크 자동 끄기, 마이크 순위 이동 등의 구현 프로세스를 포함합니다.

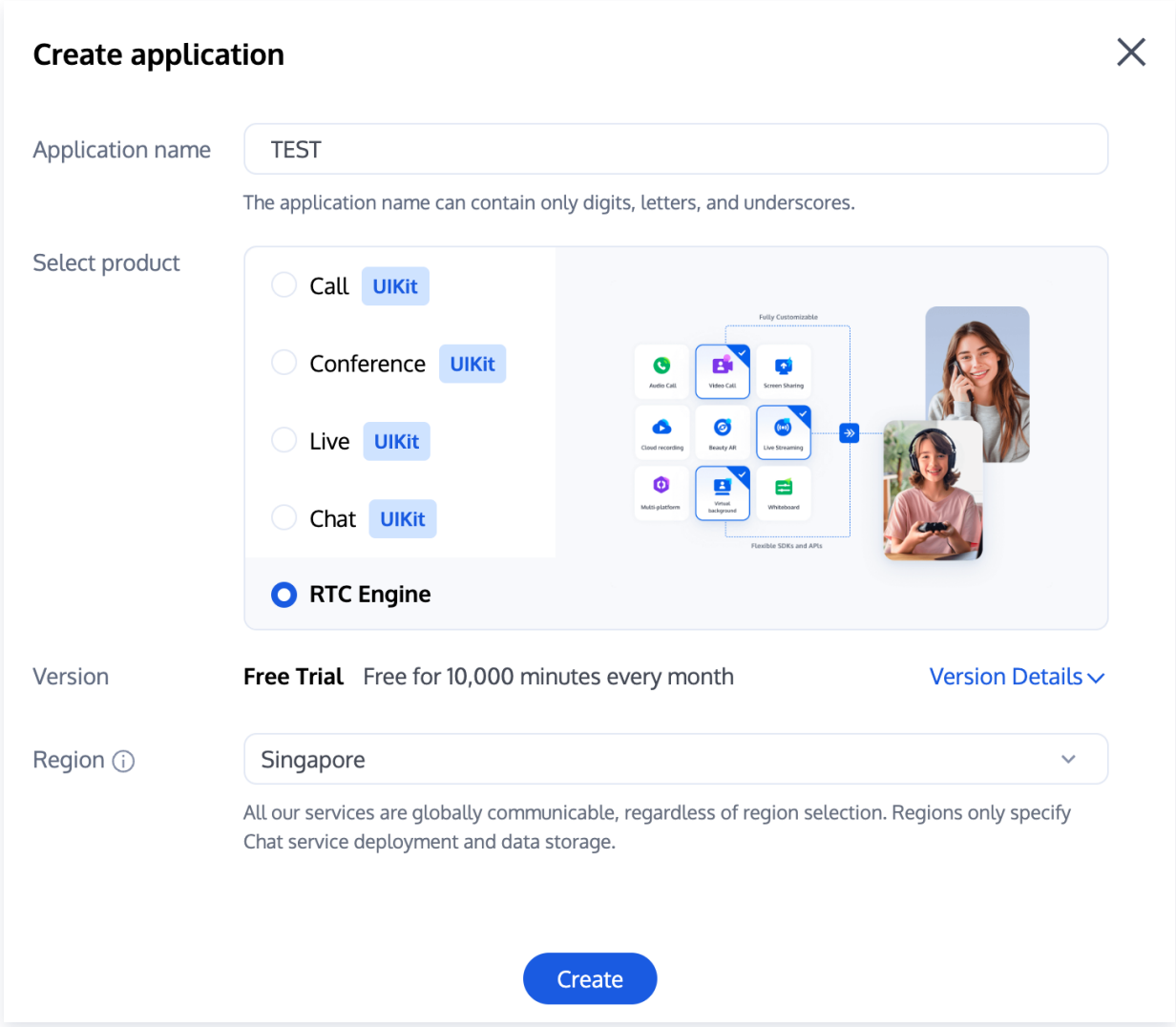


접속 준비

단계1: 서비스 개통

음성 채팅방 시나리오는 일반적으로 Chat과 RTC Engine 두 가지 유료 PaaS 서비스에 의존하여 구축됩니다.

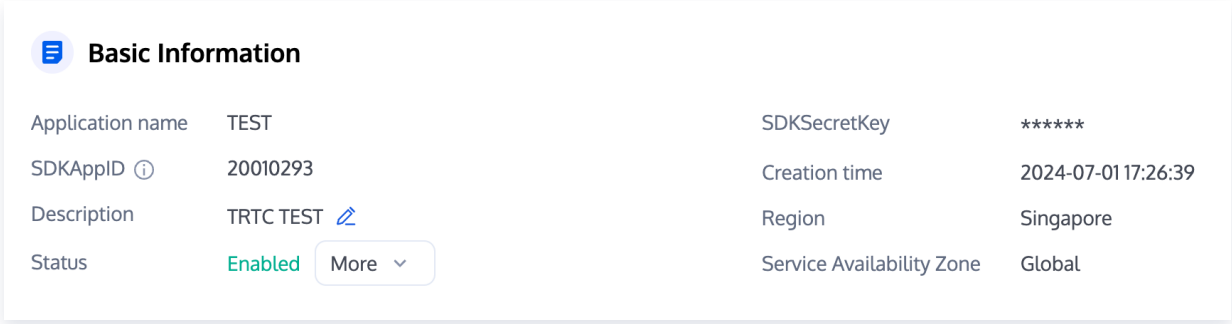
1. 먼저 콘솔에 로그인하여 애플리케이션을 생성해야 합니다. RTC Engine 애플리케이션을 생성한 후 Chat 애플리케이션을 생성해야 합니다.



설명:

- 두 개의 애플리케이션을 생성하고 각각 테스트 환경과 프로덕션 환경에 적용할 것을 권장하며 1년 동안 각 계정(UIN)당 매월 10,000분의 무료 사용 시간이 제공합니다.
- RTC Engine 월정액 요금제는 체험판(기본), 라이트, 스탠다드, 프로페셔널로 구분되며, 다양한 부가 기능 서비스를 이용할 수 있습니다. 자세한 내용은 버전 기능 및 월정액 요금제 설명를 참조하세요.

2. 애플리케이션 생성이 완료되면 앱 관리-앱 개요 메뉴에서 해당 애플리케이션의 기본 정보를 확인할 수 있습니다. SDKAppID와 SDKSecretKey는 향후 사용을 위해 안전하게 보관해야 하며,동시에 키 유출로 인한 트래픽 도용을 방지해야 합니다.



단계2: SDK 임포트하기

RTC Engine SDK와 Chat SDK가 CocoaPods에 이미 게시되었으며, CocoaPods를 통해 SDK를 통합하는 것을 권장합니다.

1. CocoaPods 설치

   터미널 창구에 다음 명령을 입력하세요(Mac에 Ruby 환경을 미리서 설치해야 합니다):

```
sudo gem install cocoapods
```

2. Podfile 파일 생성

   프로젝트 경로로 이동한 후 다음 명령어를 입력하면 프로젝트 경로에 Podfile 파일이 생성됩니다.

```
pod init
```

3. Podfile 파일 편집

   프로젝트 필요에 따라 적합한 버전을 선택하고 Podfile 파일을 편집하세요.

```
platform :ios, '8.0'
target 'App' do

    # RTC Engine 라이트 버전
    # 설치 패키지의 크기 증가가 최소화되어 있어서 RTC Engine 및 라이브방송 플레이어(TXLivePlayer) 두 가지 기능만 지원됩니다.
    pod 'TXLiteAVSDK_TRTC', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_TRTC.podspec'
```

```
    # Add the Chat SDK
    pod 'TXIMSDK_Plus_iOS'
    # pod 'TXIMSDK_Plus_iOS_XCFramework'
    # pod 'TXIMSDK_Plus_Swift_iOS_XCFramework'

    # If you need to add the Quic plugin, please uncomment the next
line.
    # Note: This plugin must be used with the Objective-C edition or
XCFramework edition of the Chat SDK, and the plugin version number
must match the Chat SDK version number.
    # pod 'TXIMSDK_Plus_QuicPlugin'

end
```

4. SDK 업데이트 및 설치

   터미널 창구에 다음 명령을 입력하여 로컬 라이브러리 파일을 업데이트하고 SDK를 설치하세요.

```
pod install
```

   또는 다음 명령을 사용하여 로컬 라이브러리 버전을 업데이트해도 됩니다.

```
pod update
```

   pod 명령 실행 후 SDK가 통합된 .xcworkspace 확장자의 프로젝트 파일이 생성되며, 더블 클릭하여 열 수 있습니다.

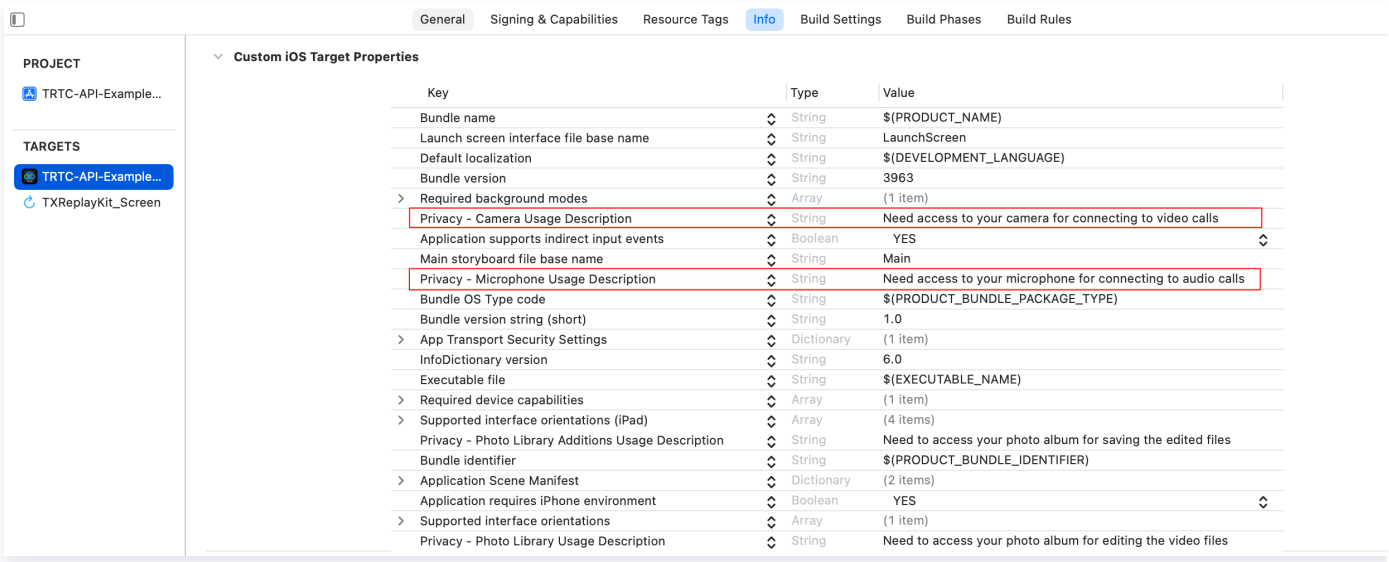
> **설명:**
>
> - pod 검색이 실패할 경우 pod의 로컬 repo 캐시를 업데이트해 볼 것을 권장합니다. 업데이트 명령은 다음과 같습니다.
>
>   ```
>   pod setup
>   pod repo update
>   rm ~/Library/Caches/CocoaPods/search_index.json
>   ```
>
> - CocoaPods 통합 방식 외에도 SDK를 다운로드하여 수동으로 임포트할 수도 있습니다. 자세한 내용은 RTC Engine SDK 수동 통합 및 Chat SDK 수동 통합를 참조하세요.
> - Quic 플러그인은 axp-Quic 멀티플렉싱 전송 프로토콜을 제공하며 약한 네트워크 환경에서도 더 우수한 성능을 발휘합니다. 네트워크 패킷 손실률이 70%에 달하는 조건에서도 서비스를 제공할

> 수 있습니다. 이 기능은 프로페셔널 에디션, 프로페셔널 에디션 플러스 및 엔터프라이즈 에디션의 사용자에게만 개방되어 있습니다. 프로페셔널 에디션, 프로페셔널 에디션 플러스 또는 엔터프라이즈 에디션를 구매한 후 사용할 수 있습니다. 기능이 정상적으로 작동될 수 있도록 하려면 터미널 SDK를 7.7.5282및 그의 이상 버전으로 업데이트하세요.

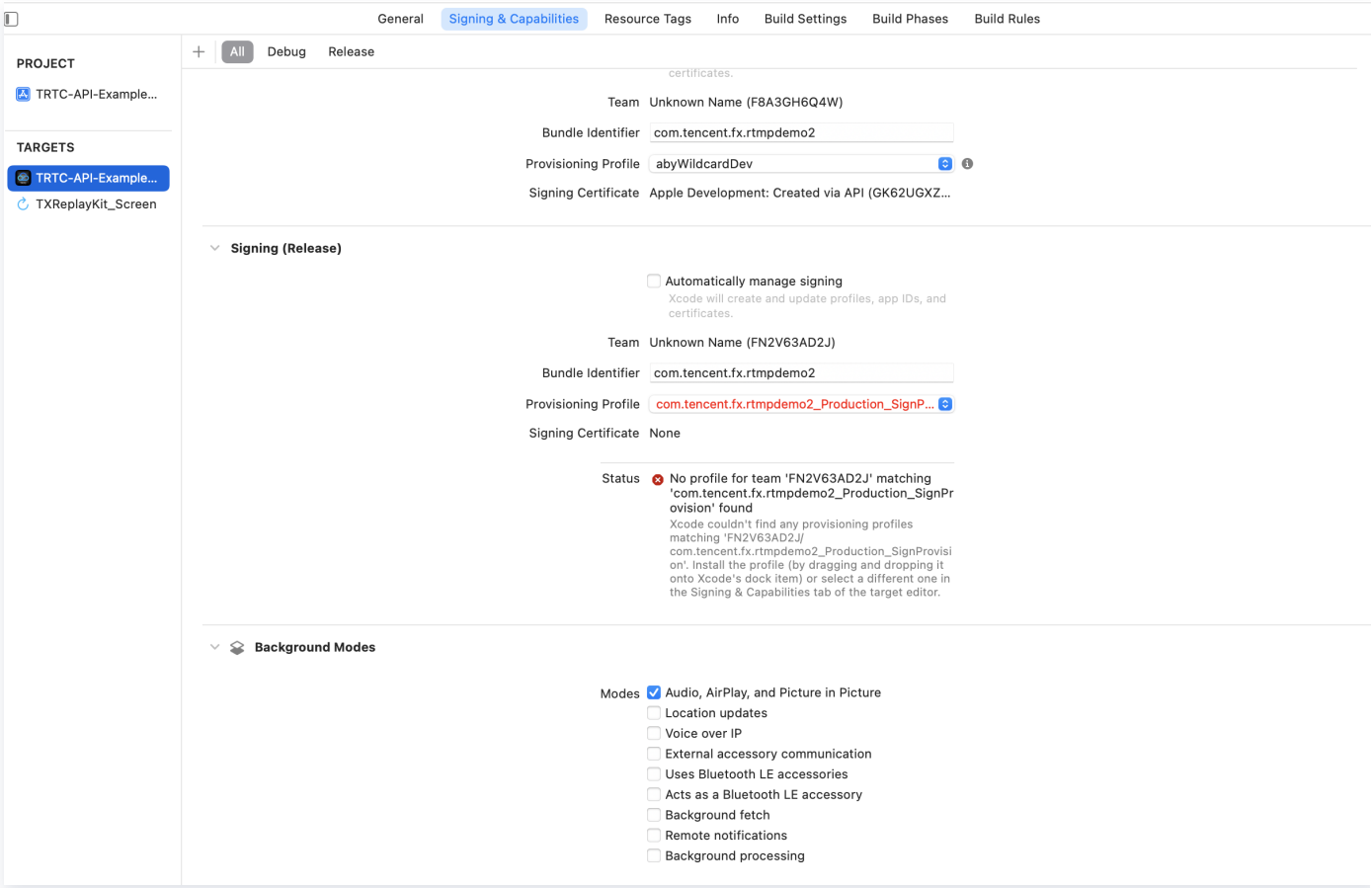
단계 3: 엔지니어링 구성

1. 음성 채팅 시나리오에서 RTC Engine SDK 및 Chat SDK는 앱의 마이크 권한을 요구하며, 앱의 Info.plist에 다음 내용을 추가해야 합니다. 시스템에 나타난 권한 부여 다이얼로 박스에 마이크에 대한 안내가 있습니다.

```
Privacy - Microphone Usage Description,마이크 사용 목적 안내문을 함께 입력하세요
```



2. 앱이 백그라운드에서도 관련 기능을 계속 실행하려면 Xcode에서 현재 프로젝트를 선택하고 Capabilities에서 Background Modes 설정을 ON으로 설정한 후 Audio, AirPlay and Picture in Picture를 선택하세요. 아래 그림과 같습니다:



접속 과정

단계1: 인증 자격 증명의 생성

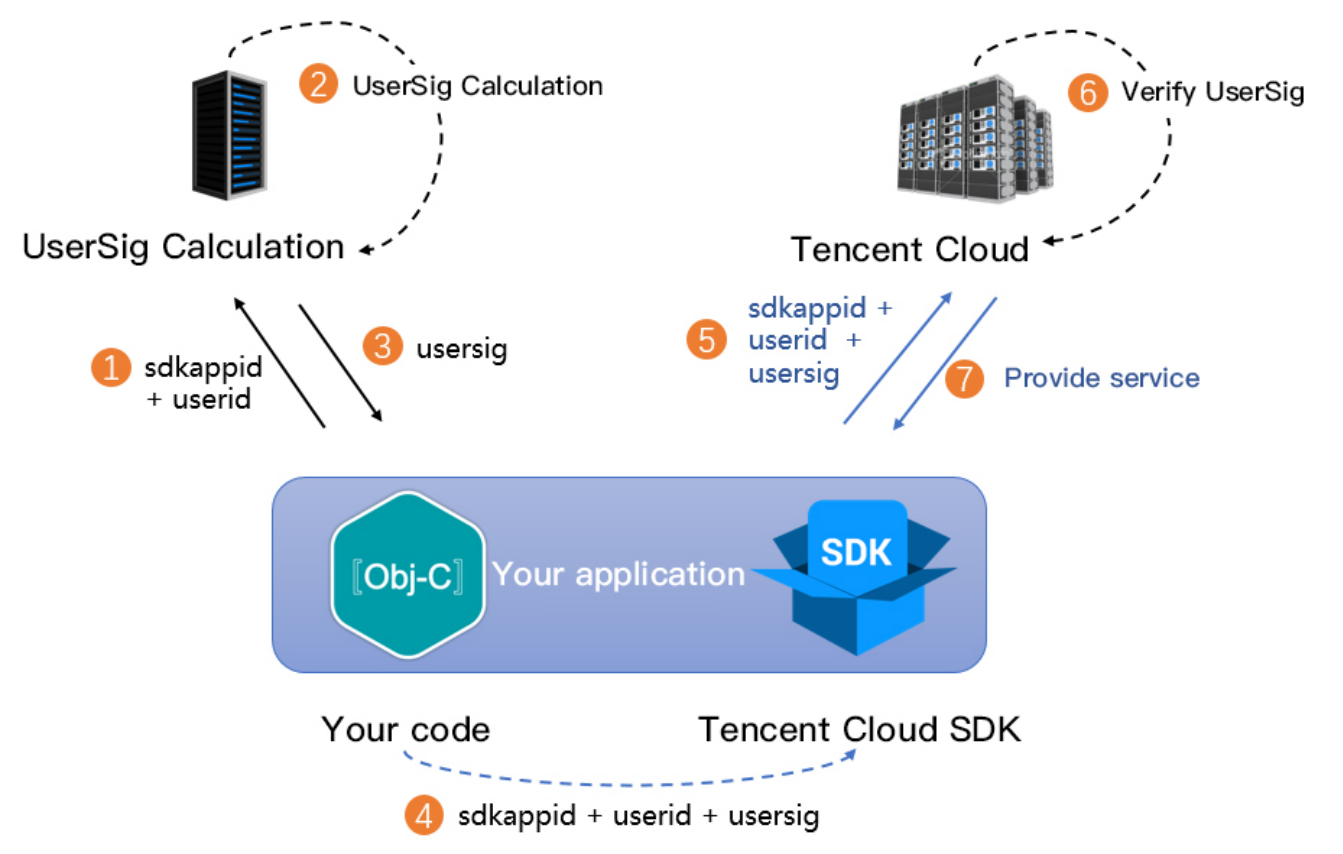
UserSig는 텐센트 실시간 통신 서비스에서 설계한 보안 서명이고 악의적인 공격자가 클라우드 서비스 사용 권한을 도용하는 것을 방지하기 위한 목적입니다. RTC Engine 및 Chat 서비스는 모두 이 보안 메커니즘을 채택하고 있으며 RTC Engine은 방 입장 시 인증을 수행하고 Chat은 로그인 시 인증을 수행합니다.

- 디버깅 및 테스트 단계: 클라이언트 예시 코드와 콘솔에서 가져오기 두 가지 방법으로 UserSig를 계산 및 생성할 수 있으며, 디버깅 및 테스트 용도로만 사용됩니다.
- 정식 운영 단계: 클라이언트가 역공학으로 키가 유출되는 것을 방지하기 위해 보안 등급이 더 높은 서버 UserSig 계산 방식의 사용을 권장합니다.

구체적인 구현 프로세스는 다음과 같습니다.

1. 손님의 App은 SDK 초기화 함수를 호출하기 전에 먼저 서버에 UserSig를 요청해야 합니다.

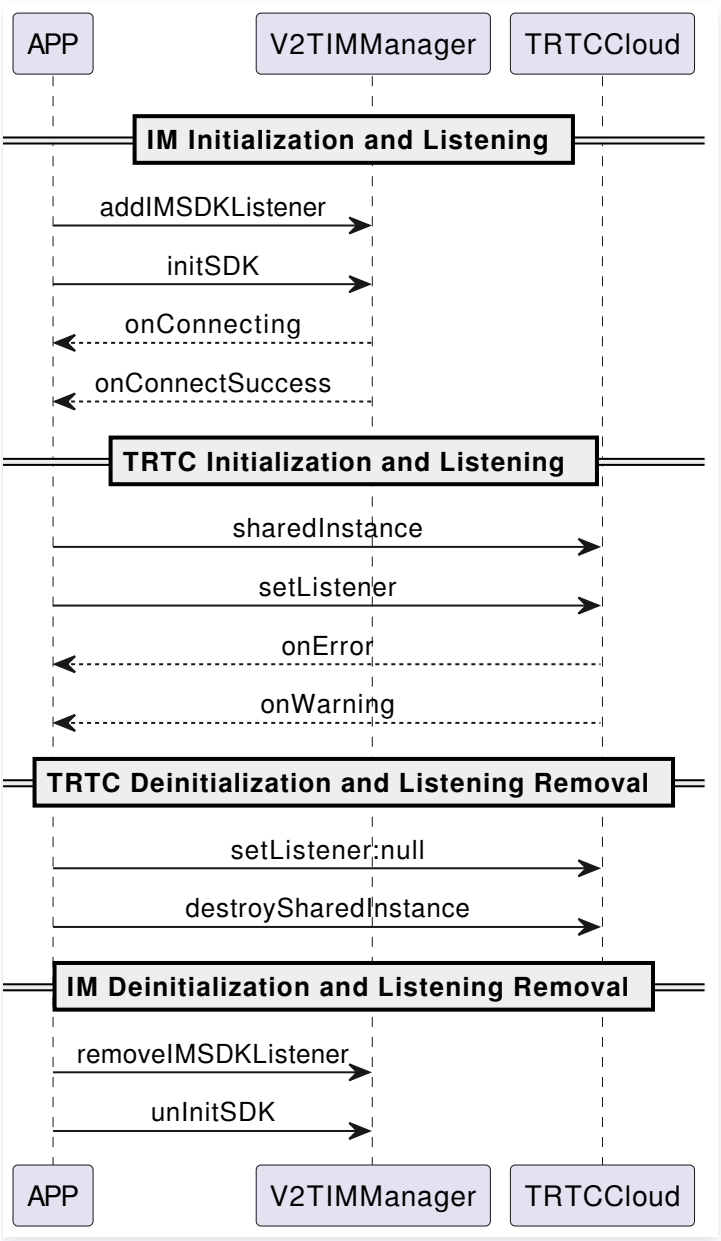
2. 손님의 서버는 SDKAppID와 UserID에 따라 UserSig를 계산합니다.
3. 서버는 계산된 UserSig를 손님의 App에 반환합니다.
4. 손님의 App은 가져온 UserSig를 특정 API를 통해 SDK에 전달합니다.
5. SDK는 SDKAppID + UserID + UserSig를 텐센트 클라우드 서버에 제출하여 검증합니다.
6. 텐센트 클라우드는 UserSig를 검증하여 합법성을 확인합니다.
7. 검증이 통과되면 Chat SDK에 인스턴트 메시징 서비스를 제공하고 RTC Engine SDK에 실시간 음성/영상 서비스를 제공합니다.



> ⚠ 주의:
> - 디버깅 단계의 로컬 UserSig 계산 방식은 온라인 환경에 적용하는 것을 권장하지 않으며, 역공학으로 인해 키가 유출될 수 있기 때문입니다.
> - 여러 언어 버전(Java/Go/PHP/Nodejs/Python/C#/C++)의 UserSig 서버 계산원 코드를 제공하며 자세한 내용은 서버에서 UserSig 계산을 참조하십시오.

단계 2: 초기화 및 리스닝

시퀀스 다이어그램



1. Chat SDK 초기화 및 이벤트 리스너의 추가

```
// Chat IM 콘솔에서 애플리케이션 SDKAppID를 획득합니다.
// V2TIMSDKListener의 이벤트 리스너를 추가합니다. self는
id<V2TIMSDKListener>의 구현 클래스이고 Chat SDK의 이벤트를 감지할 필요가 없는
경우 이 단계는 생략할 수 있습니다.
[[V2TIMManager sharedInstance] addIMSDKListener:self];
// Chat SDK를 초기화합니다. 이 인터페이스를 호출한 후 즉시 로그인 인터페이스를
호출할 수 있습니다.
[[V2TIMManager sharedInstance] initSDK:sdkAppID config:config];

// SDK 초기화 후 연결 상태, 로그인 티켓 만료 등 일부 이벤트가 발생할 수 있습니다
- (void)onConnecting {
```

```
    NSLog(@"Chat SDK가 텐센트 클라우드 서버에 연결 중입니다");
}

- (void)onConnectSuccess {
    NSLog(@"Chat SDK가 텐센트 클라우드 서버에 성공적으로 연결되었습니다");
}

// 이벤트 리스너의 제거
// self는 id<V2TIMSDKListener>의 구현 클래스입니다
[[V2TIMManager sharedInstance] removeIMSDKListener:self];
// SDK 초기화의 해제
[[V2TIMManager sharedInstance] unInitSDK];
```

설명:
앱의 라이프사이클이 SDK 라이프사이클과 일치하는 경우, 앱을 종료하기 전에 초기화 해제를 수행하지 않아도 됩니다. 특정 화면에 진입한 후에만 SDK를 초기화하고 화면을 나간 후 더 이상 사용하지 않는 경우, SDK를 초기화 해제할 수 있습니다.

2. RTC Engine SDK 인스턴스의 생성 및 이벤트 리스너의 설정.

```
// RTC Engine SDK 인스턴스의 생성(싱글톤 모드)
_trtcCloud = [TRTCCloud sharedInstance];
// 이벤트 리스너의 설정
_trtcCloud.delegate = self;

// SDK의 다양한 이벤트 알림(예: 오류 코드, 경고 코드, 오디오/비디오 상태 매개변수 등)
- (void)onError:(TXLiteAVError)errCode errMsg:(nullable NSString *)errMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", errCode, errMsg);
}

- (void)onWarning:(TXLiteAVWarning)warningCode warningMsg:(nullable NSString *)warningMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", warningCode, warningMsg);
}

// 이벤트 리스너의 제거
_trtcCloud.delegate = nil;
```

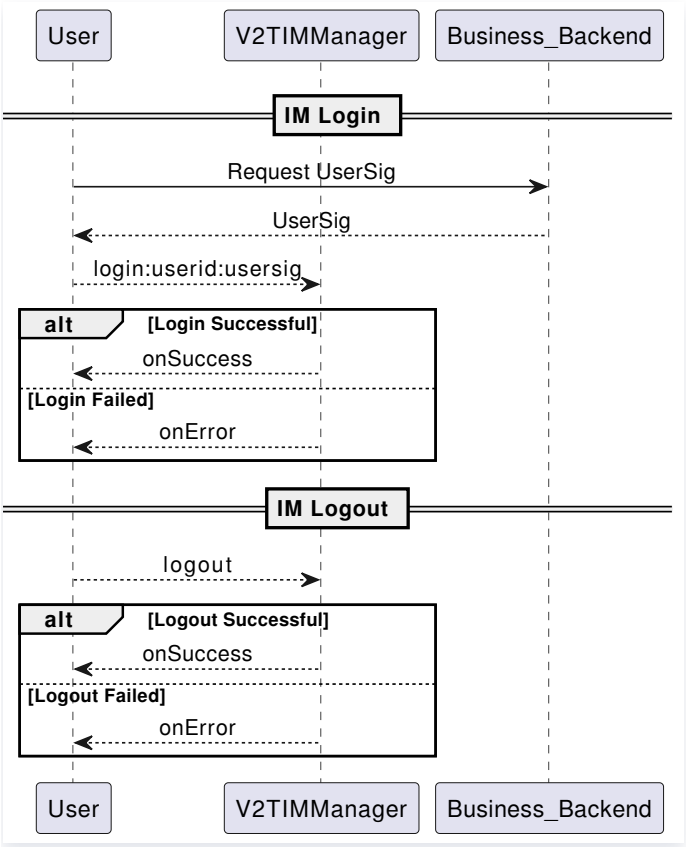
```
// RTC Engine SDK 인스턴스(싱글톤 모드)를 파기합니다.
[TRTCCloud destroySharedIntance];
```

> **설명:**
> SDK 이벤트 알림을 모니터링하고 일반적인 오류에 대한 로그 출력 및 처리를 권장합니다. 자세한 내용은 오류 코드 표를 참조하세요.

단계3: 로그인 및 로그아웃

Chat SDK 초기화 후, 계정 인증 및 기능 사용 권한을 얻기 위해 SDK 로그인 인터페이스를 호출해야 합니다. 따라서 다른 기능을 사용하기 전에 반드시 로그인이 성공했는지 확인하세요. 그렇지 않으면 기능이 비정상적으로 작동하거나 사용할 수 없을 수 있습니다. RTC Engine 오디오/비디오 서비스만 필요한 경우 이 단계를 무시해도 됩니다.

시퀀스 다이어그램



1. 로그인.

```
// 로그인: userID는 사용자 자체 정의 가능하고 userSig는 단계1 참조하여 생성 및 획득합니다
[[V2TIMManager sharedInstance] login:userID userSig:userSig succ:^{
```

```
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    // 다음 오류 코드가 반환되면 UserSig가 만료되었음을 의미하므로 새로 발급된 UserSig를 사용하여 다시 로그인하십시오.
    // 1. ERR_USER_SIG_EXPIRED(6206)
    // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED(70001)
    // 주의: 다른 오류 코드의 경우 여기에서 로그인 인터페이스를 호출하지 마십시오. Chat SDK 로그인이 무한 루프에 빠지는 것을 방지하기 위한 것입니다.
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

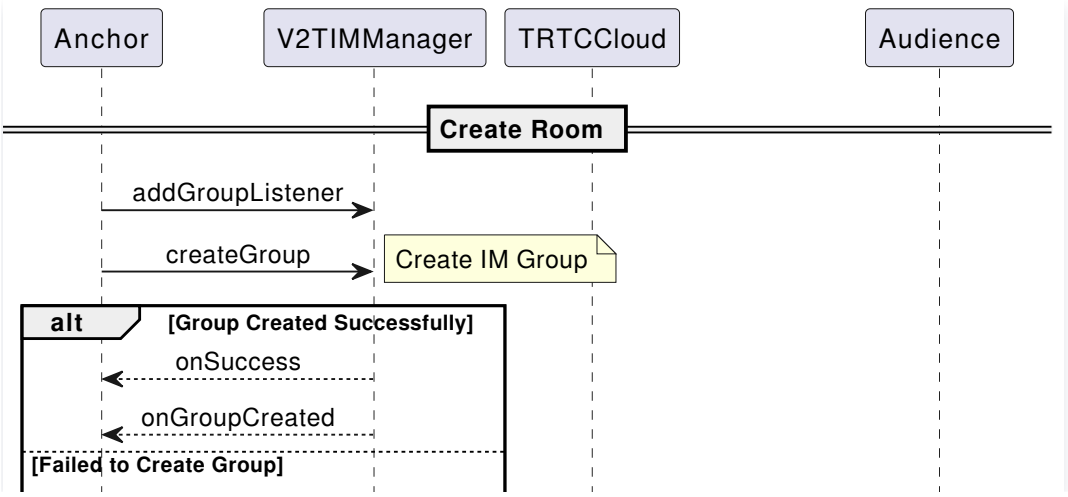
2. 로그아웃

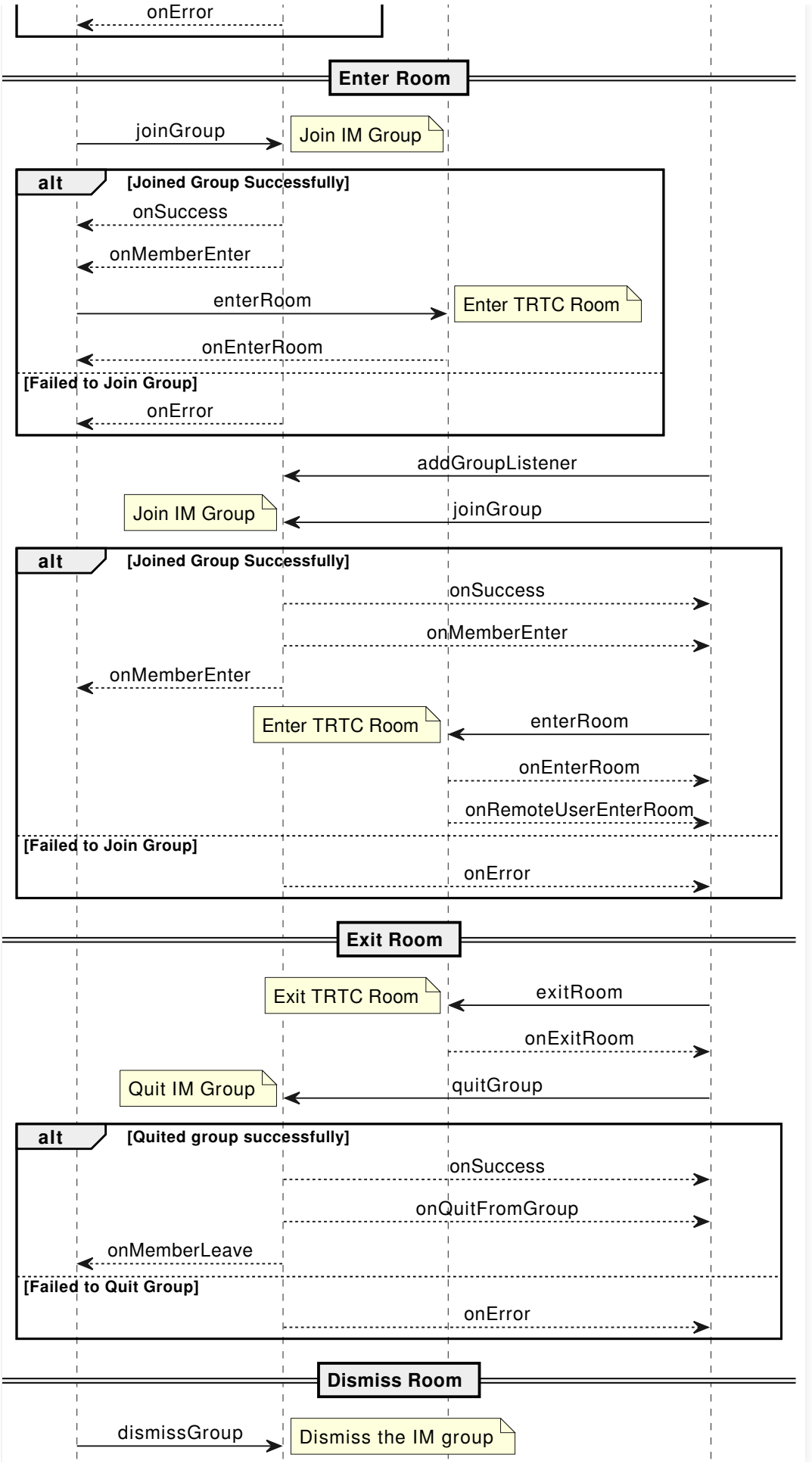
```
// 로그아웃
[[V2TIMManager sharedInstance] logout:^{
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

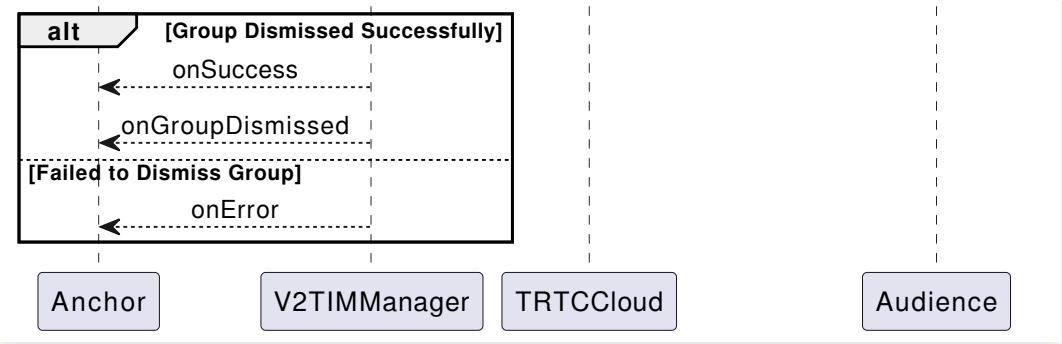
설명:
Chat SDK의 라이프사이클이 애플리케이션의 라이프사이클과 일치하는 경우, 애플리케이션을 종료하기 전에 로그아웃하지 않아도 됩니다. 특정 화면에 진입한 후에만 Chat SDK를 사용하고 화면을 나간 후에는 더 이상 사용하지 않는 경우, 로그아웃을 수행하고 Chat SDK를 초기화 해제할 수 있습니다.

단계4: 방 관리

시퀀스 다이어그램




onError
Enter Room
joinGroup
Join IM Group
alt
[Joined Group Successfully]
onSuccess
onMemberEnter
enterRoom
Enter TRTC Room
onEnterRoom
[Failed to Join Group]
onError
addGroupListener
Join IM Group
joinGroup
alt
[Joined Group Successfully]
onSuccess
onMemberEnter
onMemberEnter
Enter TRTC Room
enterRoom
onEnterRoom
onRemoteUserEnterRoom
[Failed to Join Group]
onError
Exit Room
Exit TRTC Room
exitRoom
onExitRoom
Quit IM Group
quitGroup
alt
[Quited group successfully]
onSuccess
onQuitFromGroup
onMemberLeave
[Failed to Quit Group]
onError
Dismiss Room
dismissGroup
Dismiss the IM group



1. 방 생성.

   스트리머(방주인)이 라이브방송을 시작할 때 방을 생성해야 하며, 여기서 "방"의 개념은 Chat의 "그룹"에 해당합니다. 이 예시는 클라이언트에서 Chat 그룹을 생성하는 방식만 보여주며, 실제로는 서버에서도 그룹를 생성할 수 있습니다.

```
// 그룹 생성
[[V2TIMManager sharedInstance] createGroup:GroupType_AVChatRoom
groupID:groupID groupName:groupName succ:^(NSString *groupID) {
    // 그룹 생성 성공
} fail:^(int code, NSString *desc) {
    // 그룹 생성 실패
}];


// 그룹 생성 알림을 감시합니다
[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onGroupCreated:(NSString *)groupID {
    // 그룹 생성 콜백하고 groupID는 새로 생성된 그룹의 ID입니다
}
```

   ⚠ 주의:
   - 음성 채팅방 시나리오에서 Chat 그룹 생성은 라이브 방송 그룹 유형을 선택해야 합니다. `GroupType_AVChatRoom` .
   - RTC Engine에는 방 생성 API가 없으며 사용자가 입장하려는 방이 존재하지 않을 경우 백엔드에서 자동으로 방을 생성합니다.

2. 방 입장.

- Chat 그룹 입장.

```
// 그룹 입장
```

```
[[V2TIMManager sharedInstance] joinGroup:groupID msg:message succ:^{
    // 그룹 가입 성공
} fail:^(int code, NSString *desc) {
    // 그룹 가입 실패
}];

// 그룹 가입 이벤트의 감시
[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberEnter:(NSString *)groupID memberList:
(NSArray<V2TIMGroupMemberInfo *>*)memberList {
    // 누군가 그룹에 가입했습니다
}
```

- RTC Engine 방에 가입합니다.

```
- (void)enterRoomWithRoomId:(NSString *)roomId userID:(NSString
*)userId {
    TRTCParams *params = [[TRTCParams alloc] init];
    // 문자열 방 번호를 예로 들면 IM 그룹 번호와 일치하는 것이 좋습니다
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = getUserSig(userId);
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = SDKAppID;
    // 음성 채팅 상호 작용 시나리오에서 방에 들어가려면 사용자 역할을 지정해야
합니다
    params.role = TRTCRoleAudience;
    // 음성 채팅 인터랙티브 방 입장 시나리오를 예로 들어,
    [self.trtcCloud enterRoom:params
appScene:TRTCAppSceneVoiceChatRoom];
}

// 방 입장 결과 이벤트의 콜백
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result는 방에 임장하는 데 소요된 시간(밀리초)을 나타냅니다.
        [self toastTip:@"Enter room succeed!"];
    } else {
```

```
        // result는 방 입장 실패의 오류 코드를 나타냅니다.
        [self toastTip:@"Enter room failed!"];
    }
}
```

> ⚠ 주의:
> - RTC Engine 방 번호는 정수형 `roomId` 와 문자열 형식 `strRoomId` 로 나뉘며 두 유형의 방은 서로 통하지 않으므로 방 번호 유형을 통일하는 것을 권장합니다.
> - 음성 채팅 인터랙티브 시나리오에서 방에 입장 시 사용자 역할(스트리머/시청자)을 지정해야 하며 스트리머만 스트리밍 권한이 있습니다. 지정하지 않으면 기본적으로 스트리머 역할로 설정됩니다.
> - 음성 채팅 인터랙티브 방 입장 시나리오에서는 `TRTCAppSceneVoiceChatRoom` 을 사용하는 것이 좋습니다.

3. 방에서 나가기.

- Chat 그룹 나가기.

```
[[V2TIMManager sharedInstance] quitGroup:groupID succ:^{
    // 그룹 나가기 성공
} fail:^(int code, NSString *desc) {
    // 그룹 나가기 실패
}];

[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberLeave:(NSString *)groupID member:
(V2TIMGroupMemberInfo *)member {
    // 그룹 멤버 나가기의 콜백
}
```

> ⚠ 주의:
> 라이브 방송 그룹(AVChatRoom)에서는 그룹 주인이 그룹에서 나갈 수 없으며 `dismissGroup` 을 호출하여 그룹을 해산할 수만 있습니다.

- RTC Engine 방에서 나가기.

```
- (void)exitTrtcRoom {
    self.trtcCloud = [TRTCCloud sharedInstance];
```

```
    [self.trtcCloud stopLocalAudio];
    [self.trtcCloud exitRoom];
}

// onExitRoom 콜백을 수신하여 자신의 퇴장 사유를 확인할 수 있습니다.
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        // exitRoom을 호출하여 방에서 나가기.
        NSLog(@"Exit current room by calling the 'exitRoom' api of sdk
...");
    } else if (reason == 1) {
        //서버에 의해 현재 방에서 나가게 됩니다
        NSLog(@"Kicked out of the current room by server through the
restful api...");
    } else if (reason == 2) {
        // 현재 방 전체가 해체됩니다
        NSLog(@"Current room is dissolved by server through the
restful api...");
    }
}
```

⚠ 주의:

- SDK가 점유한 모든 리소스가 릴리스된 후, SDK는 `onExitRoom` 콜백을 통해 알려줍니다.
- `enterRoom` 을 다시 호출하거나 다른 음성/영상 SDK로 전환하려면 `onExitRoom` 콜백이 발생한 후 관련 작업을 수행하십시오. 그렇지 않으면 카메라, 마이크등 장치의 점유됨과 같은 다양한 이상 현상이 발생할 수 있습니다.

4. 방 해산하기.

- Chat 그룹 해산.

  본 예시는 클라이언트에서 Chat 그룹 해산하는 방법만 보여주고 실제로는 서버에서 그룹 해산할 수도 있습니다.

```
[[V2TIMManager sharedInstance] dismissGroup:groupID succ:^{
    // 그룹 해산 성공
} fail:^(int code, NSString *desc) {
    // 그룹 해산 실패
}];
```

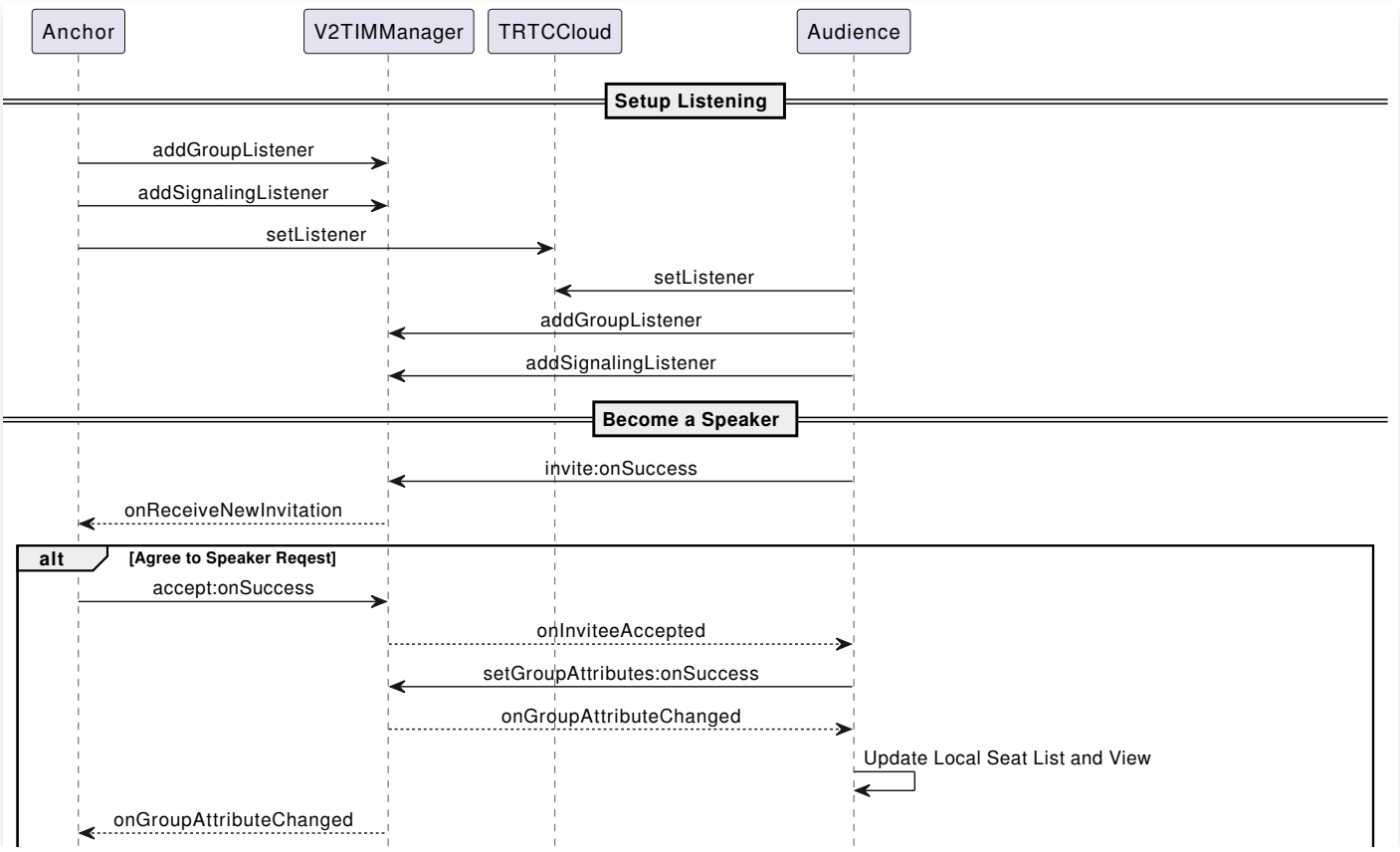
```
[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onGroupDismissed:(NSString *)groupID opUser:
(V2TIMGroupMemberInfo *)opUser {
    // 그룹 해산 콜백
}
```

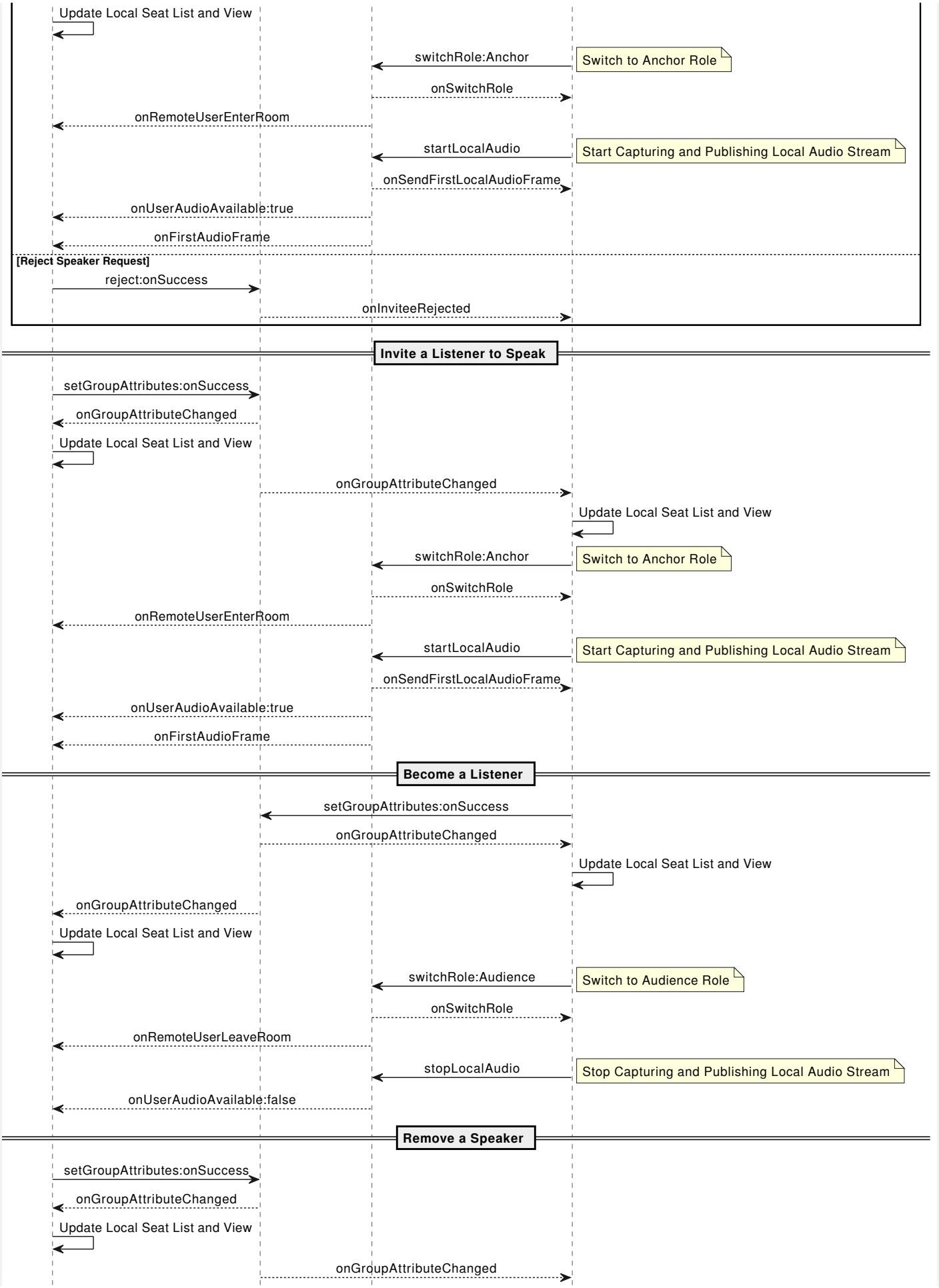
- RTC Engine 방을 해산합니다.
  - **서버측 해산**: RTC Engine 서버 측 방 해산 API `DismissRoom` (숫자 방 ID와 문자열 방 ID 구분)를 제공합니다. 이 인터페이스를 호출하여 방의 모든 사용자를 방에서 제거하고 방을 해산할 수 있습니다.
  - **클라이언트 측 해산**: 각 클라이언트의 방 나가기 `exitRoom` 인터페이스를 통해 방 내의 모든 스트리머와 청취자를 방에서 완전히 나가게 할 수 있습니다. 방을 나간 후, RTC Engine 방의 생명주기 규칙에 따라 방은 자동으로 해산됩니다. 자세한 내용은 방 나가기 를 참조하세요.

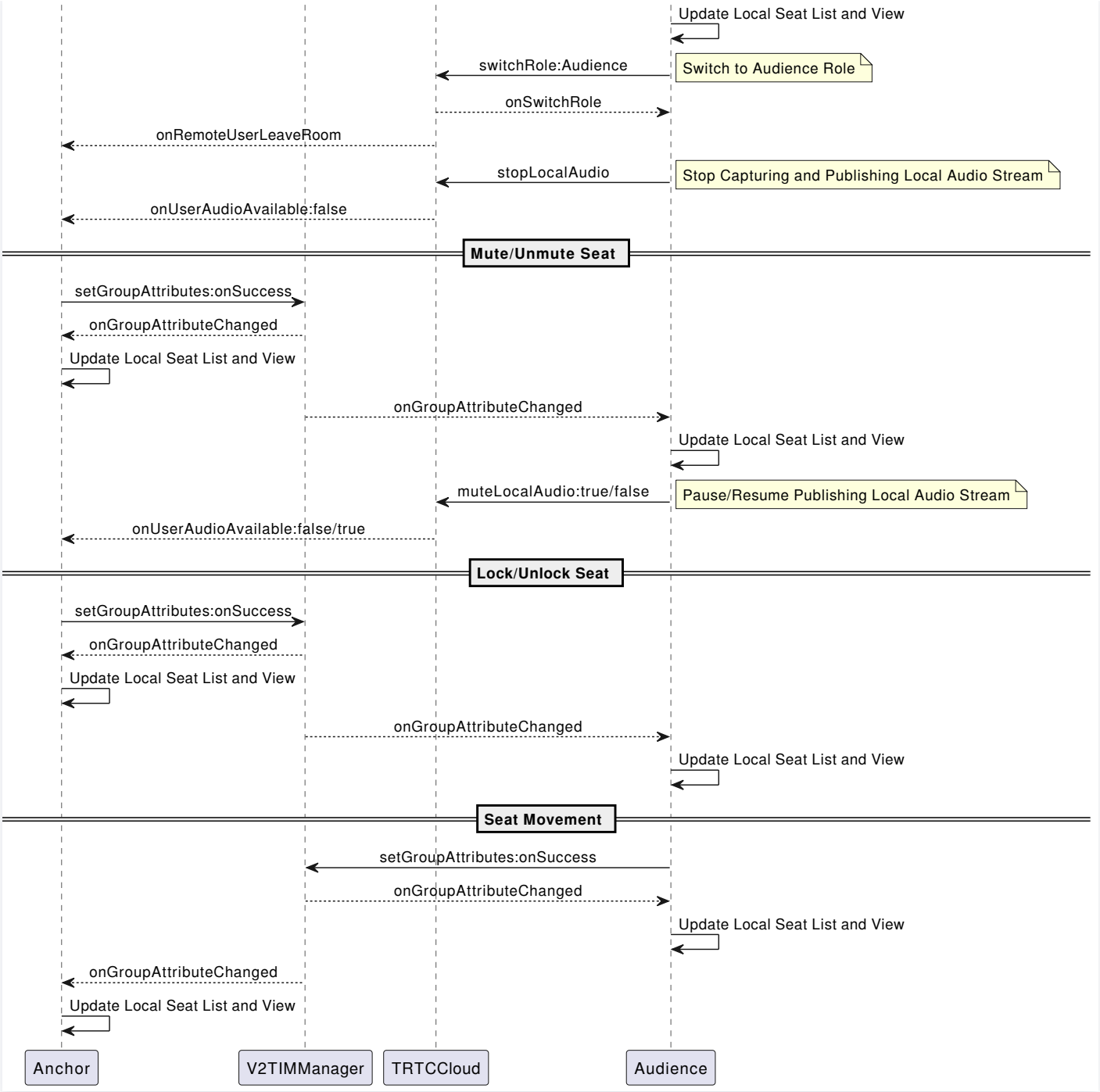
> **경고:**
> 라이브 방송 작업이 종료되면 방 해산 API를 호출하여 방이 해산되도록 하여 청취자가 의외로 방에 들어가서 원치 않는 비용이 발생하는 것을 방지하는 것을 권장합니다.

단계5: 마이크 순위 관리

시퀀스 다이어그램







먼저, 마이크 순위 정보를 저장하기 위한 모델을 생성할 수 있습니다.

```
#import "JSONModel.h"

typedef NS_ENUM(NSUInteger, SeatInfoStatus) {
    SeatInfoStatusUnused = 0,
    SeatInfoStatusUsed = 1,
    SeatInfoStatusLocked = 2,
};

NS_ASSUME_NONNULL_BEGIN

@interface SeatInfoModel : JSONModel

/// 좌석 상태는 세 가지 상태에 대응합니다
@property (nonatomic, assign) SeatInfoStatus status;
/// 죄석 금언 여부
@property (nonatomic, assign) BOOL mute;
/// 좌석이 점유됐을 때, 사용자 정보 저장합니다
@property (nonatomic, copy) NSString *userId;

@end

NS_ASSUME_NONNULL_END
```

1. 자동으로 마이크 사용.

   자동으로 마이크의 사용을 켜는 것은 방송실에 있는 청취자가 방주인 또는 관리자에게 마이크의 사용 켜기 신청을 보내고 동의한다는 시그널링을 받은 후 마이크를 켜서 사용하는 것을 의미합니다. 자유 마이크 모드인 경우 시그널링 요청 부분은 무시할 수 있습니다.
   - 청취자가 마이크 사용의 요청을 보냅니다

   ```
   // 청취자가 마이크 요청을 보내며 userId는 스트리머 Id이고  data는 시그널링 식별을 위한 json을 전달할 수 있습니다
   - (void)sendInvitationWithUserId:(NSString *)userId data:(NSString *)data {
       [[V2TIMManager sharedInstance] invite:userId data:data onlineUserOnly:YES offlinePushInfo:nil timeout:0 succ:^{
           NSLog(@"sendInvitation success");
       } fail:^(int code, NSString *desc) {
   ```

```
        NSLog(@"sendInvitation error %d", code);
    }];
}

// 스트리머가 마이크 사용의 요청을 수신하며  inviteID는 해당 요청 Id이고
inviter는 요청자 ID입니다
[[V2TIMManager sharedInstance] addSignalingListener:self];
- (void)onReceiveNewInvitation:(NSString *)inviteID inviter:(NSString
*)inviter groupID:(NSString *)groupID inviteeList:(NSArray<NSString
*> *)inviteeList data:(NSString * __nullable)data {
    NSLog(@"received invitation: %@ from %@", inviteID, inviter);
}
```

○ 스트리머가 마이크 사용의 요청을 처리합니다

```
// 마이크 사용의 요청을 동의합니다
- (void)acceptInvitationWithInviteID:(NSString *)inviteID data:
(NSString *)data {
    [[V2TIMManager sharedInstance] accept:inviteID data:data succ:^{
        NSLog(@"acceptInvitation success");
    } fail:^(int code, NSString *desc) {
        NSLog(@"acceptInvitation error %d", code);
    }];
}

// 마이크 사용의 요청 거절합니다
- (void)rejectInvitationWithInviteID:(NSString *)inviteID data:
(NSString *)data {
    [[V2TIMManager sharedInstance] reject:inviteID data:data succ:^{
        NSLog(@"rejectInvitation success");
    } fail:^(int code, NSString *desc) {
        NSLog(@"rejectInvitation error %d", code);
    }];
}
```

○ 청취자가 마이크 사용합니다

스트리머가 청취자의 마이크 사용 요청을 동의하면 청취자는 그룹 속성을 수정하는 방식으로 마이크 정보를 추가할 수 있으며,다른 사용자는 그룹 속성 변경 콜백을 수신하고 로컬 마이크 정보가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
@property (nonatomic, copy) NSArray<SeatInfoModel *> *seatInfoArray;

// 마이크 요청 동의의 콜백
- (void)onInviteeAccepted:(NSString *)inviteID invitee:(NSString
*)invitee data:(NSString * __nullable)data {
    NSLog(@"received accept invitation: %@ from %@", inviteID,
invitee);
    NSInteger seatIndex = [self findSeatIndex:inviteID];
    [self takeSeatWithIndex:seatIndex];
}

// 청취자가 마이크를 켜서 사용하기 시작합니다
- (void)takeSeatWithIndex:(NSInteger)seatIndex {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfoModel *localInfo = self.seatInfoArray[seatIndex];
    SeatInfoModel *seatInfo = [[SeatInfoModel alloc] init];
    seatInfo.status = SeatInfoStatusUsed;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = self.userId;

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    NSString *jsonStr = seatInfo.toJSONString;
    NSDictionary *dict = @{ [NSString stringWithFormat:@"seat%ld",
seatIndex]: jsonStr };

    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    [[V2TIMManager sharedInstance] setGroupAttributes:self.groupId
attributes:dict succ:^{
        // 그룹 속성의 수정이 성공되고 TRTC 역할의 전환 및 스트리밍 시작합니다
        [self.trtcCloud switchRole:TRTCRoleAnchor];
        [self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];
    } fail:^(int code, NSString *desc) {
        // 그룹 속성의 수정이 실패되고 마이크 연결이 실패됩니다
    }];
}
```

2. 마이크의 사용을 초대합니다.

스트리머가 마이크의 사용을 초대합니다(청취자의 동의 필요없이).직접 그룹 속성을 수정하여 저장된 마이크 순위 정보를 변경하며, 해당 청취자는 그룹 속성 변경 콜백을 수신한 후 userId 매칭에 성공하면 RTC Engine 역할을 전환하고 스트리밍을 시작할 수 있습니다. 마이크 사용 초대 모드인 경우, 자동으로 마이크를 켜기 구현 로직을 참조하고 시그널링의 발신자와 수신자를 바꾸기만 하면 됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
@property (nonatomic, copy) NSArray<SeatInfoModel *> *seatInfoArray;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
- (void)pickSeatWithUserId:(NSString *)userId seatIndex:
(NSInteger)seatIndex {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfoModel *localInfo = self.seatInfoArray[seatIndex];
    SeatInfoModel *seatInfo = [[SeatInfoModel alloc] init];
    seatInfo.status = SeatInfoStatusUsed;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = self.userId;

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    NSString *jsonStr = seatInfo.toJSONString;
    NSDictionary *dict = @{ [NSString stringWithFormat:@"seat%ld",
seatIndex]: jsonStr };

    // 그룹 속성 설정합니다.해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고 없으면 해당 속성을 추가합니다
    [[V2TIMManager sharedInstance] setGroupAttributes:self.groupId
attributes:dict succ:^{
        // 그룹 속성의 수정이 성공되고 onGroupAttributeChanged 콜백 트리거됩
니다
    } fail:^(int code, NSString *desc) {
        // 그룹 속성의 수정이 실패되며 마이크 연결이 실패됩니다
    }];
}

// 청취자 측에서 그룹 속성 변경 콜백을 수신하여 자체 정보에 매칭 성공 후 스트리밍
시작합니다
[[V2TIMManager sharedInstance] addGroupListener:self];
```

```
- (void)onGroupAttributeChanged:(NSString *)groupID attributes:
(NSMutableDictionary<NSString *,NSString *> *)attributes {
    // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
    NSArray *oldSeatArray = self.seatInfoArray;
    // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
    NSArray *newSeatArray = [self getSeatListFromAttr:attributes
seatSize:self.seatSize];
    // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니다.
    for (int i = 0; i < self.seatSize; i++) {
        SeatInfoModel *oldInfo = oldSeatArray[i];
        SeatInfoModel *newInfo = newSeatArray[i];
        if (oldInfo.status != newInfo.status && newInfo.status ==
SeatInfoStatusUsed) {
            if ([newInfo.userId isEqualToString:self.userId]) {
                // 자신의 정보에 매칭 성공 후, RTC Engine 역할 전환 및 스트
리밍 시작합니다
                [self.trtcCloud switchRole:TRTCRoleAnchor];
                [self.trtcCloud
startLocalAudio:TRTCAudioQualityDefault];
            } else {
                // 로컬 마이크 순위 목록을 업데이트하고 로컬 마이크 순위 뷰 렌
더링합니다
            }
        }
    }
}
```

3. 자동으로 마이크 끄기.

   연결된 청취자는 그룹 속성을 수정하는 방식으로 마이크 순위 정보를 재설정할 수 있으며 다른 사용자는 그룹 속성 변경 콜백을 받고 로컬 마이크 정보가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
@property (nonatomic, copy) NSArray<SeatInfoModel *> *seatInfoArray;

- (void)leaveSeatWithIndex:(NSInteger)seatIndex {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfoModel *localInfo = self.seatInfoArray[seatIndex];
    SeatInfoModel *seatInfo = [[SeatInfoModel alloc] init];
    seatInfo.status = SeatInfoStatusUnused;
```

```
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = @"";

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    NSString *jsonStr = seatInfo.toJSONString;
    NSDictionary *dict = @{ [NSString stringWithFormat:@"seat%ld",
seatIndex]: jsonStr };

    // 그룹 속성 설정합니다.해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고 없으면 해당 속성을 추가합니다
    [[V2TIMManager sharedInstance] setGroupAttributes:self.groupId
attributes:dict succ:^{
        // 그룹 속성의 수정이 성공되고 RTC Engine 역할 전환 및 스트리밍 중지합
니다
        [self.trtcCloud switchRole:TRTCRoleAudience];
        [self.trtcCloud stopLocalAudio];
    } fail:^(int code, NSString *desc) {
        // 그룹 속성의 수정이 실패되고 마이크 끄기가 실패됩니다
    }];
}
```

4. 강제로 마이크 끄기.

   스트리머가 사용자의 마이크 사용을 끄게 합니다. 그룹 속성에 저장된 마이크 순위 정보를 직접 수정하여 저장하고 해당 연결된 청취자는 그룹 속성 변경 콜백을 받은 후 userId와 일치하면 RTC Engine 역할을 전환하고 스트리밍을 중지합니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
@property (nonatomic, copy) NSArray<SeatInfoModel *> *seatInfoArray;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
- (void)kickSeatWithIndex:(NSInteger)seatIndex {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfoModel *localInfo = self.seatInfoArray[seatIndex];
    SeatInfoModel *seatInfo = [[SeatInfoModel alloc] init];
    seatInfo.status = SeatInfoStatusUnused;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = @"";
```

```
    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    NSString *jsonStr = seatInfo.toJSONString;
    NSDictionary *dict = @{ [NSString stringWithFormat:@"seat%ld",
seatIndex]: jsonStr };

    // 그룹 속성 설정합니다.해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고 없으면 해당 속성을 추가합니다
    [[V2TIMManager sharedInstance] setGroupAttributes:self.groupId
attributes:dict succ:^{
        // 그룹 속성의 수정이 성공되고 onGroupAttributeChanged 콜백 트리거됩
니다
    } fail:^(int code, NSString *desc) {
        // 그룹 속성의 수정이 실패되고 마이크  끄기 실패됩니다
    }];
}

// 연결된 청취자 측에서 그룹 속성 변경 콜백을 수신하여 자체 정보에 매칭 성공 후
스트리밍 중지합니다
[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onGroupAttributeChanged:(NSString *)groupID attributes:
(NSMutableDictionary<NSString *,NSString *> *)attributes {
    // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
    NSArray *oldSeatArray = self.seatInfoArray;
    // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
    NSArray *newSeatArray = [self getSeatListFromAttr:attributes
seatSize:self.seatSize];
    // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니다.
    for (int i = 0; i < self.seatSize; i++) {
        SeatInfoModel *oldInfo = oldSeatArray[i];
        SeatInfoModel *newInfo = newSeatArray[i];
        if (oldInfo.status != newInfo.status && newInfo.status ==
SeatInfoStatusUnused) {
            if ([newInfo.userId isEqualToString:self.userId]) {
                // 자신의 정보에 매칭 성공 후 RTC Engine 역할을 전환하고 스
트리밍을 중지합니다.
                [self.trtcCloud switchRole:TRTCRoleAudience];
                [self.trtcCloud stopLocalAudio];
            } else {
                // 로컬 마이크 순위 목록을 업데이트하고 로컬 마이크 순위 뷰 렌
더링합니다
```

```
            }
        }
    }
}
```

5. 마이크 금언.

   스트리머가 특정 마이크 순위를 금언/해제하고 그룹 속성에 저장된 마이크 순위 정보를 직접 수정하며 해당 연결된 청취자는 그룹 속성 변경 콜백을 수신한 후 userId 매칭에 성공하면 로컬 스트리밍을 일시 중지/재개합니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
@property (nonatomic, copy) NSArray<SeatInfoModel *> *seatInfoArray;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
- (void)muteSeatWithIndex:(NSInteger)seatIndex mute:(BOOL)mute {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfoModel *localInfo = self.seatInfoArray[seatIndex];
    SeatInfoModel *seatInfo = [[SeatInfoModel alloc] init];
    seatInfo.status = localInfo.status;
    seatInfo.mute = mute;
    seatInfo.userId = localInfo.userId;

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    NSString *jsonStr = seatInfo.toJSONString;
    NSDictionary *dict = @{ [NSString stringWithFormat:@"seat%ld",
seatIndex]: jsonStr };

    // 그룹 속성 설정합니다. 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고 없으면 해당 속성을 추가합니다
    [[V2TIMManager sharedInstance] setGroupAttributes:self.groupId
attributes:dict succ:^{
        // 그룹 속성의 수정이 성공되고 onGroupAttributeChanged 콜백 트리거됩
니다
    } fail:^(int code, NSString *desc) {
        // 그룹 속성의 수정이 실패되고 마이크 금언 실패됩니다
    }];
}
```

```
// 연결된 청취자 측에서 그룹 속성 변경 콜백을 수신하여 자체 정보에 매칭 성공 후
스트리밍 일시 중지/재개합니다
[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onGroupAttributeChanged:(NSString *)groupID attributes:
(NSMutableDictionary<NSString *,NSString *> *)attributes {
    // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
    NSArray *oldSeatArray = self.seatInfoArray;
    // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
    NSArray *newSeatArray = [self getSeatListFromAttr:attributes
seatSize:self.seatSize];
    // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니다.
    for (int i = 0; i < self.seatSize; i++) {
        SeatInfoModel *oldInfo = oldSeatArray[i];
        SeatInfoModel *newInfo = newSeatArray[i];
        if (oldInfo.mute != newInfo.mute) {
            if ([newInfo.userId isEqualToString:self.userId]) {
                // 자신의 정보에 매칭 성공 후 로컬 스트리밍을 일시 중지/재개합
니다.
                [self.trtcCloud muteLocalAudio:newInfo.mute];
            } else {
                // 로컬 마이크 순위 목록을 업데이트하고 로컬 마이크 순위 뷰 렌
더링합니다
            }
        }
    }
}
```

6. 마이크 순위 잠그기.

   스트리머가 특정 마이크를 잠그거나 잠금 해제합니다. 그룹 속성에 저장된 마이크 순위 정보를 직접 수정하며 청취자는 그룹 속성 변경 콜백을 수신한 후 해당 마이크 뷰가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
@property (nonatomic, copy) NSArray<SeatInfoModel *> *seatInfoArray;

// 스트리머 측에서 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이크 순위 정보
를 수정합니다.
- (void)lockSeatWithIndex:(NSInteger)seatIndex isLock:(BOOL)isLock {
    // 마이크 순위 정보 인스턴스 생성하며 수정된 마이크 순위 정보를 저장합니다
    SeatInfoModel *localInfo = self.seatInfoArray[seatIndex];
```

```
    SeatInfoModel *seatInfo = [[SeatInfoModel alloc] init];
    seatInfo.status = isLock? SeatInfoStatusLocked :
SeatInfoStatusUnused;
    seatInfo.mute = localInfo.mute;
    seatInfo.userId = @"";

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    NSString *jsonStr = seatInfo.toJSONString;
    NSDictionary *dict = @{ [NSString stringWithFormat:@"seat%ld",
seatIndex]: jsonStr };

    // 그룹 속성 설정합니다. 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고 없으면 해당 속성을 추가합니다
    [[V2TIMManager sharedInstance] setGroupAttributes:self.groupId
attributes:dict succ:^{
        // 그룹 속성의 수정이 성공되고 onGroupAttributeChanged 콜백 트리거됩
니다
    } fail:^(int code, NSString *desc) {
        // 그룹 속성의 수정이 실패되고 마이크 잠금이 실패됩니다
    }];
}

// 청취자 측에서 그룹 속성 변경 콜백을 수신하여 해당 마이크 뷰가 업데이트됩니다
[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onGroupAttributeChanged:(NSString *)groupID attributes:
(NSMutableDictionary<NSString *,NSString *> *)attributes {
    // 로컬에 저장된 기존의 전체 마이크 순위 정보 리스트
    NSArray *oldSeatArray = self.seatInfoArray;
    // groupAttributeMap에서 파싱한 최신 전체 마이크 순위 정보 리스트
    NSArray *newSeatArray = [self getSeatListFromAttr:attributes
seatSize:self.seatSize];
    // 전체 마이크 순위 정보 리스트를 순회하며 새/구 마이크 정보를 비교합니다.
    for (int i = 0; i < self.seatSize; i++) {
        SeatInfoModel *oldInfo = oldSeatArray[i];
        SeatInfoModel *newInfo = newSeatArray[i];
        if (oldInfo.status == SeatInfoStatusLocked && newInfo.status
== SeatInfoStatusUnused) {
            // 마이크 잠금의 해제
        } else if (oldInfo.status != newInfo.status && newInfo.status
== SeatInfoStatusLocked) {
```

```
            // 마이크 순위 잠그기
        }
    }
}
```

7. 마이크 순위의 이동.

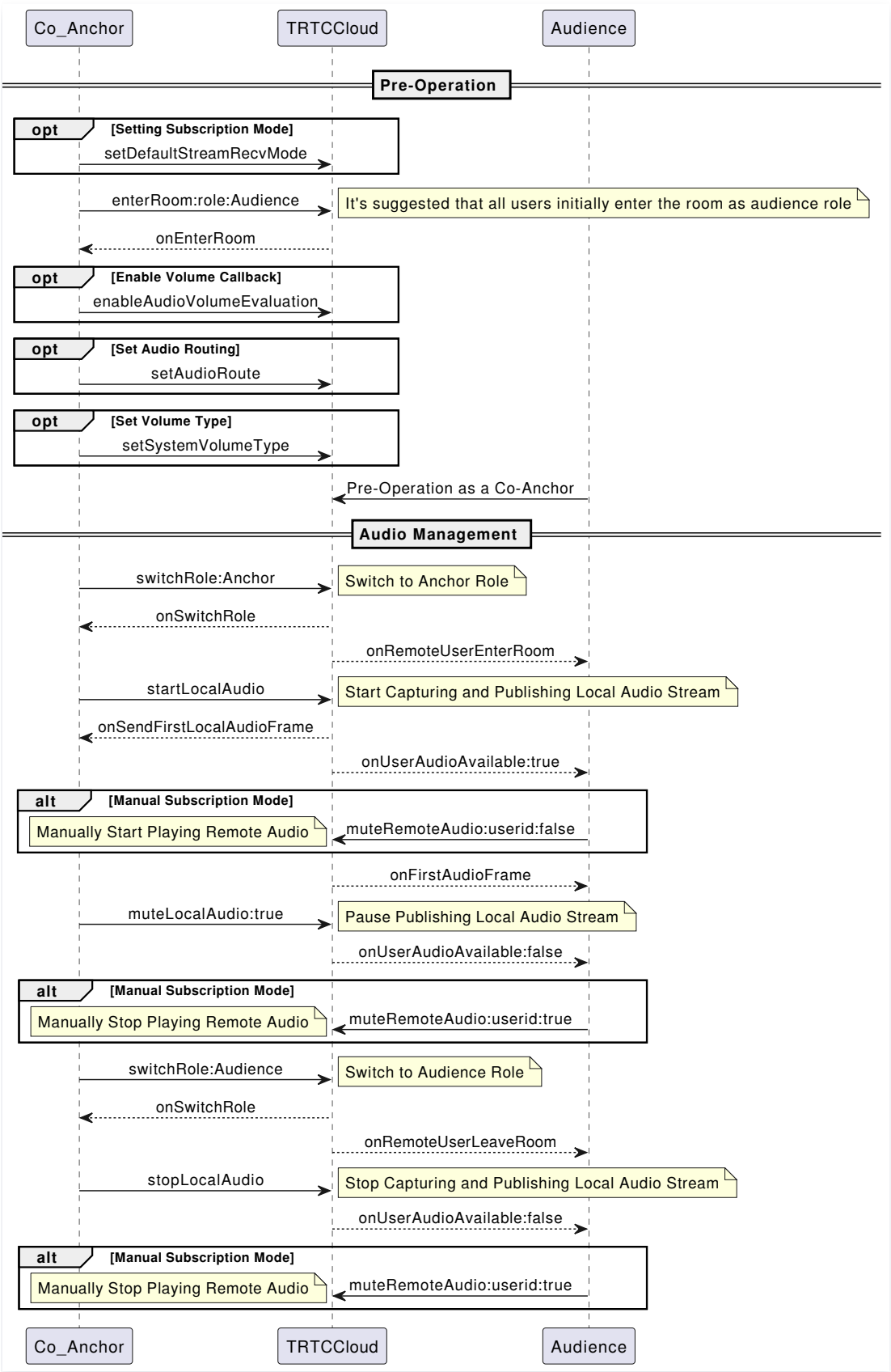
   마이크 사용중의 스트리머가 마이크 순위를 이동하면 그룹 속성에 저장된 소스 및 대상 마이크 정보를 각각 수정해야 하며 청취자는 그룹 속성 변경 콜백을 수신한 후 해당 마이크 뷰가 업데이트됩니다.

```
// 로컬에 저장된 전체 마이크 순위 정보의 목록
@property (nonatomic, copy) NSArray<SeatInfoModel *> *seatInfoArray;

// 마이크 사용중의 스트리머가 해당 인터페이스를 호출하여 그룹 속성에 저장된 마이
크 정보를 수정합니다.
- (void)moveSeatToIndex:(NSInteger)dstIndex {
    // userId로 소스 마이크 번호를 받습니다
    __block NSInteger srcIndex = -1;
    [self.seatInfoArray enumerateObjectsUsingBlock:^(SeatInfoModel *
_Nonnull seatInfo, NSUInteger idx, BOOL * _Nonnull stop) {
        if ([seatInfo.userId isEqualToString:self.userId]) {
            srcIndex = idx;
            *stop = YES;
        }
    }];
    if (srcIndex < 0 || dstIndex < 0 || dstIndex >=
self.seatInfoArray.count) {
        return;
    }

    // 마이크 번호로 해당 마이크 순위를 받습니다
    SeatInfoModel *srcSeatInfo = self.seatInfoArray[srcIndex];
    SeatInfoModel *dstSeatInfo = self.seatInfoArray[dstIndex];

    // 마이크 위치 정보 인스턴스를 생성하고 수정된 소스 마이크 순위 정보를 저장
합니다
    SeatInfoModel *srcChangeInfo = [[SeatInfoModel alloc] init];
    srcChangeInfo.status = SeatInfoStatusUnused;
    srcChangeInfo.mute = srcSeatInfo.mute;
    srcChangeInfo.userId = @"";
```

```
    // 마이크 순위 정보 인스턴스를 생성하고 수정된 대상 마이크 순위 정보를 저장
합니다
    SeatInfoModel *dstChangeInfo = [[SeatInfoModel alloc] init];
    dstChangeInfo.status = SeatInfoStatusUsed;
    dstChangeInfo.mute = dstSeatInfo.mute;
    dstChangeInfo.userId = self.userId;

    // 마이크 순위 정보 객체를 JSON 형식으로 직렬화합니다
    NSString *srcJsonStr = srcChangeInfo.toJSONString;
    NSString *dstJsonStr = dstChangeInfo.toJSONString;
    NSDictionary *dict = @{ [NSString stringWithFormat:@"seat%ld",
srcIndex]: srcJsonStr,
                            [NSString stringWithFormat:@"seat%ld",
dstIndex]: dstJsonStr
    };

    // 그룹 속성을 설정하고 해당 그룹 속성이 이미 있으면 value 값을 업데이트하
고, 없으면 해당 속성을 추가합니다.
    [[V2TIMManager sharedInstance] setGroupAttributes:self.groupId
attributes:dict succ:^{
        // 그룹 속성의 수정이 성공되며 마이크 이동이 성공됩니다
    } fail:^(int code, NSString *desc) {
        // 그룹 속성의 수정이 실패되고 마이크 이동이 실패됩니다
    }];
}
```

단계6: 오디오 관리

시퀀스 다이어그램



1. 구독 모드.

   RTC Engine SDK는 기본적으로 오디오 스트림을 자동으로 구독하는 로직으로 설정되어 있으며 사용자가 방에 입장하면 원격 사용자의 음성이 자동으로 재생됩니다. 수동으로 오디오 스트림을 구독하는 경우,

`muteRemoteAudio(userId, mute)` 를 추가로 호출하여 원격 사용자의 오디오 스트림을 구독하고 재생해야 합니다.

```
// 자동 구독 모드(기본값)
[self.trtcCloud setDefaultStreamRecvMode:YES video:YES];

// 수동 구독 모드(자체 정의)
[self.trtcCloud setDefaultStreamRecvMode:NO video:NO];
```

> ⚠ **주의:**
> 구독 모드인 `setDefaultStreamRecvMode` 을 설정하려면 방 입장 `enterRoom` 전에 호출해야만 적용시켜야 합니다.

2. 수집 및 게시.

```
// 로컬 오디오의 수집 및 게시를 활성화합니다
[self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];

// 로컬 오디오의 수집 및 게시를 중지합니다
[self.trtcCloud stopLocalAudio];
```

> ⓘ **설명:**
> `startLocalAudio` 는 마이크 사용 권한을 요청하며, `stopLocalAudio` 는 마이크 사용 권한을 릴리스합니다.

3. 마이크 끄기와 켜기.

```
// 로컬 오디오 스트림 게시를 일시 중지합니다(마이크 끄기)
[self.trtcCloud muteLocalAudio:YES];
// 로컬 오디오 스트림 게시를 재개합니다(마이크 켜기)
[self.trtcCloud muteLocalAudio:NO];

// 특정 원격 사용자의 오디오 스트림 구독 및 재생을 일시 중지합니다
[self.trtcCloud muteRemoteAudio:userId mute:YES];
// 특정 원격 사용자의 오디오 스트림 구독 및 재생을 재개합니다
[self.trtcCloud muteRemoteAudio:userId mute:NO];
```

```
// 모든 원격 사용자의 오디오 스트림 구독 및 재생을 일시 중지합니다
[self.trtcCloud muteAllRemoteAudio:YES];
// 모든 원격 사용자의 오디오 스트림 구독 및 재생을 재개합니다
[self.trtcCloud muteAllRemoteAudio:NO];
```

설명:
반면에, `muteLocalAudio` 는 소프트웨어 측면에서 데이터 스트림을 일시 중지하거나 허용하기만 하면 되므로 효율적이고 원활합니다.자주 마이크를 켜고 끄는 시나리오에 더 적합합니다.

4. 음질 및 음량 유형.
   - 음질의 설정

```
// 로컬 오디오 수집 및 게시 시 음질을 설정합니다
[self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];

// 오디오 스트리밍 중 종적으로 음질을 설정합니다
[self.trtcCloud setAudioQuality:TRTCAudioQualityDefault];
```

설명:
RTC Engine 사전 설정된 음질은 세 가지 등급(Speech/Default/Music)으로 나뉘며 각각 다른 오디오 매개변수에 해당합니다. 자세한 내용은 TRTCAudioQuality 을 참조하세요.

   - 음량 유형의 설정

     RTC Engine 각 음질 등급에는 기본 음량 유형이 설정되어 있으며 음량 유형을 강제로 지정하려면 다음 인터페이스를 사용할 수 있습니다.

```
// 음량 유형의 설정
[self.trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeAuto];
```

설명:
RTC Engine 음량 유형은 세 가지 등급(VOIP/Auto/Media)으로 나뉘며 각각 다른 음량 채널에 해당합니다. 자세한 내용은 TRTCSystemVolumeType 을 참조하세요.

   - 오디오 라우팅의 설정

휴대폰과 같은 모바일 장치에는 일반적으로 스피커와 수화기 두 가지 재생 위치가 있습니다. 오디오 라우팅을 강제로 지정하려면 다음 인터페이스를 사용할 수 있습니다.

```
// 오디오 라우팅의 설정
[self.trtcCloud setAudioRoute:TRTCAudioModeSpeakerphone];
```

설명:
RTC Engine 오디오 라우팅은 두 가지 유형(Speaker/Earpiece)으로 나뉘며, 각각 다른 발음 위치에 해당합니다. 자세한 내용은 TRTCAudioRoute을 참조하세요.

고급 기능

탄막 메시지 인터랙티브

음성 채팅 라이브 방송에는 일반적으로 텍스트 형태의 탄막 메시지 인터랙티브가 있으며, 여기서는 Chat을 통해 그룹 채팅 일반 텍스트 메시지를 보내고 받는 방식으로 구현할 수 있습니다.

```
// 공개 탄막 메시지의 전송
[[V2TIMManager sharedInstance] sendGroupTextMessage:text to:groupID
priority:V2TIM_PRIORITY_NORMAL succ:^{
    // 탄막 메시지의 전송이 성공됩니다
} fail:^(int code, NSString *desc) {
    // 탄막 메시지의 전송이 실패됩니다
}];

// 공개 탄막 메시지의 수신
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];
- (void)onRecvGroupTextMessage:(NSString *)msgID groupID:(NSString
*)groupID sender:(V2TIMGroupMemberInfo *)info text:(NSString *)text {
    NSLog(@"%@: %@", info.nickName, text);
}
```

볼륨 크기 콜백

RTC Engine 고정된 주기로 마이크 사용중의 스트리머의 볼륨 크기를 콜백할 수 있으며 일반적으로 음파나 음량을 표시하거나 발언 중인 스트리머를 알리는 데 사용됩니다.

```
// 볼륨 크기 콜백 활성화하고 방 입장 성공 후 즉시 활성화하는 것을 권장합니다
// interval: 콜백 간격(ms); enable_vad: 인간 음성 검사의 활성화 여부
[self.trtcCloud enableAudioVolumeEvaluation:interval
enable_vad:enable_vad];
self.trtcCloud.delegate = self;
```

```
- (void)onUserVoiceVolume:(NSArray<TRTCVolumeInfo *> *)userVolumes
totalVolume:(NSInteger)totalVolume {
    // userVolumes는 로컬 사용자와 원격 스트리밍 사용자를 포함한 모든 말하고 있는
사용자의 볼륨 크기를 관리합니다
    // totalVolume는 원격 스트리밍 사용자 중 최대 볼륨 값을 피드백하는 데 사용됩
니다
    ...
    // 볼륨 크기에 따라 UI에 해당하는 음파 디스플레이를 표시합니다.
    ...
}
```

⚠ 주의:

- 인간 음성 검사는 로컬 음성 검사 결과만 피드백하며 자신의 역할은 반드시 스트리머여야 하여 사용자에게 마이크 켜기를 알리기 편리합니다.
- `userVolumes` 는 배열이며 배열의 각 요소에서 userId가 비어 있으면 로컬 마이크에서 수집한 볼륨 크기가 너무 적다는 것을 나타내고, userId가 비어 있지 않으면 원격 사용자의 볼륨 크기가 너무 적다는 것을 나타냅니다.

음악 및 음향 효과의 재생

배경 음악및 음향 효과의 재생은 음성 채팅방 시나리오에서 자주 사용되는 것입니다.아래에서는 일반적으로 사용되는 배경 음악 관련 인터페이스의 사용 방법과 주의 사항에 대해 설명합니다.

1. 재생 시작/중지/일시 정지/재개.

```
// 배경 음악, 짧은 음향 효과 및 인간음성 이펙트를 설정하는 데 사용되는 관리 클래
스 얻습니다
self.audioEffectManager = [self.trtcCloud getAudioEffectManager];

TXAudioMusicParam *param = [[TXAudioMusicParam alloc] init];
param.ID = musicID;
param.path = musicPath;
// 음악을 원격으로 게시할지?(그렇지 않으면 로컬에서만 재생됨)
param.publish = YES;
// 재생 중인 파일이 짧은 음향 효과 파일인지?
param.isShortFile = NO;

// 배경 음악의 재생 시작합니다
__weak typeof(self) weakSelf = self;
```

```
[self.audioEffectManager startPlayMusic:param onStart:^(NSInteger
errCode) {
    __strong typeof(weakSelf) strongSelf = weakSelf;
    // 재생 시작 콜백
    // -4001: 경로 열기 실패
    // -4002: 디코딩 실패
    // -4003: URL 주소 무효
    // -4004: 재생 중지되지 않음
    if (errCode < 0) {
        // 재생 실패 후 다시 재생하기 전에 현재 음악 재생을 먼저 중지해야 합니
다.
        [strongSelf.audioEffectManager stopPlayMusic:musicID];
    }
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // 재생 진도 콜백
    // progressMs 현재 재생 시간(밀리초)
    // durationMs 현재 음악의 총 시간(밀리초)
} onComplete:^(NSInteger errCode) {
    // 재생 종료 콜백
    // 약한 네트워크 상태로 인해 재생 중간에 실패한 경우에도 이 콜백이 발생하며
이때 errCode < 0입니다
    // 재생 중간에 일시 정지하거나 중지해도 onComplete 콜백이 발생하지 않습니다
}];
// 배경 음악의 재생 중지합니다
[self.audioEffectManager stopPlayMusic:musicID];
// 배경 음악의 재생을 일시 정지합니다
[self.audioEffectManager pausePlayMusic:musicID];
// 배경 음악의 재생 재개합니다
[self.audioEffectManager resumePlayMusic:musicID];
```

⚠ 주의:

- RTC Engine 여러 음악을 동시에 재생할 수 있으며 musicID로 고유하게 식별됩니다. 어느 기간에 한 곡만 재생하려면 새 음악 재생 전에 다른 음악을 중지하거나 동일한 musicID를 사용하여 다른 음악을 재생할 수 있습니다. 이 경우에 SDK는 기존 음악을 먼저 중지한 후 새 음악을 재생합니다.
- RTC Engine 로컬 및 네트워크 오디오 파일의 재생을 지원하며 `musicPath` 를 통해 로컬 절대 경로 또는 URL 주소를 전달할 수 있습니다. MP3/AAC/M4A/WAV 형식을 지원합니다.

2. 음악 및 인간 음성의 볼륨 비율을 조정합니다.

```
// 특정 배경 음악의 로컬 재생 음량 크기를 설정합니다
[self.audioEffectManager setMusicPlayoutVolume:musicID
volume:volume];
// 특정 배경 음악의 원격 재생 음량 크기를 설정합니다
[self.audioEffectManager setMusicPublishVolume:musicID
volume:volume];
// 모든 배경 음악의 로컬 음량 및 원격 음량 크기를 설정합니다
[self.audioEffectManager setAllMusicVolume:volume];
// 인간 음성을 수집하는 음량 크기를 설정합니다
[self.audioEffectManager setVoiceVolume:volume];
```

⚠ 주의:

- 음량 값 volume의 정상 범위는 0–100이며 기본값은 60입니다. 최대 150까지 설정할 수 있지만 폭음 위험이 있습니다.
- 인간 음성이 배경 음악에 묻히는 경우, 음악의 재생 음량을 적절히 낮추고 인간 음성을 수집하는 음량을 높여야 합니다.
- **마이크 끄고 배경 음악은 끄지 않는 경우**: `muteLocalAudio(true)` 아니고 `setVoiceVolume(0)` 를 사용합니다.

3. 배경 음악 및 음향 효과를 반복 재생합니다.
   - 방안 1: `AudioMusicParam` 의 `loopCount` 매개변수를 사용하여 반복 재생 횟수를 설정합니다.

값의 범위는 0 – 임의의 양의 정수이며, 기본값은 0입니다. 0은 음악을 한 번 재생함을 나타내고, 1은 음악을 두 번 재생함을 나타내며, 이와 같이 반복됩니다.

```
- (void)startPlayMusicWithId:(int32_t)musicId path:(NSString *)path
loopCount:(NSInteger)loopCount {
    TXAudioMusicParam *param = [[TXAudioMusicParam alloc] init];
    param.ID = musicId;
    param.path = path;
    param.publish = YES;
    // 재생 중인 파일이 짧은 음향 효과 파일인지?
    param.isShortFile = YES;
    // 반복 재생 횟수를 설정합니다. 음수는 무한 반복을 의미합니다
    param.loopCount = loopCount < 0 ? NSIntegerMax : loopCount;
    [self.audioEffectManager startPlayMusic:param onStart:nil
onProgress:nil onComplete:nil];
}
```

> **설명:**
> 방안 1은 매번 반복 재생이 끝나도 `onComplete` 콜백이 트리거되지 않으며 설정된 반복 횟수가 모두 완료된 후에야 해당 콜백이 트리거됩니다.

- 방안 2: "배경 음악이 재생 완료됨" 이벤트 콜백 `onComplete`을 통해 반복 재생을 구현하며 일반적으로 리스트 반복 또는 단곡 반복에 사용됩니다.

```
- (void)repeatPlayMusicWithParam:(TXAudioMusicParam *)param {
    __weak typeof(self) weakSelf = self;
    [self.audioEffectManager startPlayMusic:param onStart:nil
onProgress:nil onComplete:^(NSInteger errCode) {
        __strong typeof(weakSelf) strongSelf = weakSelf;
        // 여기서 재생 인터페이스를 다시 호출하여 음악을 반복 재생할 수 있습니
다.
        if (errCode >= 0) {
            [strongSelf repeatPlayMusicWithParam:param];
        }
    }];
}
```

혼합 스트림 전송 및 리턴 전송

1. 혼합 스트림을 RTC Engine 방으로 리턴 전송합니다.

```
- (void)startPublishMediaToRoom:(NSString *)roomId userID:(NSString
*)userId {
    // 미디어 스트림 발행의 목적지 주소
    TRTCPublishTarget *target = [[TRTCPublishTarget alloc] init];
    // 혼합 스트림 후 방으로 리턴 전송
    target.mode = TRTCPublishMixStreamToRoom;
    target.mixStreamIdentity.strRoomId = roomId;
    // 혼합 스트림 로봇의 userid는 방 내 다른 사용자의 userid와 중복되면 안됩
니다
    target.mixStreamIdentity.userId = [NSString
stringWithFormat:@"%@%@", userId, MIX_ROBOT];

    TRTCStreamEncoderParam* encoderParam = [[TRTCStreamEncoderParam
alloc] init];
```

```
    // 트랜스코딩된 오디오 스트림의 인코딩 매개변수를 설정합니다(사용자 자체 정의
가능)
    encoderParam.audioEncodedSampleRate = 48000;
    encoderParam.audioEncodedChannelNum = 2;
    encoderParam.audioEncodedKbps = 64;
    encoderParam.audioEncodedCodecType = 2;

    // 트랜스코팅된 비디오 스트림의 인코딩 매개변수를 설정합니다(순수 오디오및 믹
싱혼합 스트림의 경우는 무시해도 됩니다)
    encoderParam.videoEncodedWidth = 64;
    encoderParam.videoEncodedHeight = 64;
    encoderParam.videoEncodedFPS = 15;
    encoderParam.videoEncodedGOP = 3;
    encoderParam.videoEncodedKbps = 30;

    // 오디오/비디오 혼합 스트리밍 매개변수를 설정합니다
    TRTCStreamMixingConfig *config = [[TRTCStreamMixingConfig alloc]
init];
    // 기본적으로 빈 값을 입력하면 되고 방의 모든 오디오가 믹싱됨을 의미합니다
    config.audioMixUserList = nil;

    // 비디오 혼합 스트림 템플릿을 설정합니다(순수 오디오및 믹싱혼합 스트림의 경
우는 무시해도 됩니다)
    TRTCVideoLayout *layout = [[TRTCVideoLayout alloc] init];
    config.videoLayoutList = @[layout];

    // 혼합 스트림의 전송이 시작됩니다
    [self.trtcCloud startPublishMediaStream:target
encoderParam:encoderParam mixingConfig:config];
}
```

2. 이벤트 콜백 및 업데이트 작업 중지합니다.
   - 작업 결과 이벤트의 콜백

```
#pragma mark - TRTCCloudDelegate

- (void)onStartPublishMediaStream:(NSString *)taskId code:(int)code
message:(NSString *)message extraInfo:(NSDictionary *)extraInfo {
```

```
    // taskId: 요청이 성공하면 RTC Engine 백엔드는 콜백에서 이 작업의 taskId
를 제공합니다. 다음에 이 taskId를 updatePublishMediaStream 및
stopPublishMediaStream과 함께 사용하여 업데이트 및 중지할 수 있습니다.
    // code:  콜백 결과.0은 성공을 나타내며 그 외의 값은 실패를 나타냅니다.
}

- (void)onUpdatePublishMediaStream:(NSString *)taskId code:(int)code
message:(NSString *)message extraInfo:(NSDictionary *)extraInfo {
    // 미디어 스트림 발생 인터페이스 (updatePublishMediaStream) 를 호출할 때
전달한 taskId는 이 콜백을 통해 다시 전달되며 이 콜백이 어떤 업데이트 요청에 속하
는지 식별하는 데 사용됩니다.
    // code: 콜백 결과.0은 성공을 나타내며 그 외의 값은 실패를 나타냅니다.
}

- (void)onStopPublishMediaStream:(NSString *)taskId code:(int)code
message:(NSString *)message extraInfo:(NSDictionary *)extraInfo {
    // 미디어 스트림 중지 인터페이스(stopPublishMediaStream)를 호출할 때 전
달한 taskId는 이 콜백을 통해 다시 전달되며 이 콜백이 어떤 중지 요청에 속하는지
식별하는 데 사용됩니다.
    // code: 콜백 결과. 0은 성공을 나타내며, 그 외의 값은 실패를 나타냅니다.
}
```

○ 미디어 발생 스트림의업데이트

이 인터페이스는 RTC Engine 서버에 명령을 보내고 `startPublishMediaStream` 로 시작된 미디어 스트림을 업데이트합니다.

```
// taskId: onStartPublishMediaStream 콜백의 작업 ID
// target: 예를 들어 발행된 CDN URL 추가 또는 삭제
// params: 재생 측의 스트림 중단을 방지하기 위해 미디어 스트림 인코딩 출력 매개
변수를 일치하게 유지하세요.
// config: 예를 들어 크로스 룸 PK와 같은혼합 스트림 트랜스코딩에 참여하는 사용
자 목록 업데이트합니다
[self.trtcCloud updatePublishMediaStream:taskId publishTarget:target
encoderParam:trtcStreamEncoderParam
mixingConfig:trtcStreamMixingConfig];
```

⚠ 주의:
동일한 작업은 순수 오디오, 오디오/비디오, 순수 비디오 간의 전환을 지원하지 않습니다.

○ 미디어 스트림의 발생을 중지합니다.

이 인터페이스는 RTC Engine 서버에 명령을 보내고 `startPublishMediaStream` 로 시작된 미디어 스트림을 중지합니다.

```
// taskId: onStartPublishMediaStream 콜백의 작업 ID
[self.trtcCloud stopPublishMediaStream:taskId];
```

> ⚠ 주의:
> taskId가 빈 문자열이면 `startPublishMediaStream` 로 시작된 해당 사용자의 모든 미디어 스트림을 중지합니다. 하나의 미디어 스트림만 시작했거나 손님을 통해 시작된 모든 미디어 스트림을 중지하려는 경우 이 방법을 사용하는 것이 좋습니다.

네트워크 상태의 실시간 콜백

`onNetworkQuality` 를 통해 로컬 및 원격 사용자의 네트워크 상태를 실시간으로 모니터링할 수 있으며 이 콜백은 2초마다 한 번씩 발생합니다.

```
#pragma mark - TRTCCloudDelegate

- (void)onNetworkQuality:(TRTCQualityInfo *)localQuality remoteQuality:
(NSArray<TRTCQualityInfo *> *)remoteQuality {
    // localQuality userId가 비어 있으면 로컬 사용자의 네트워크 상태에 대한 평가
결과를 나타냅니다.
    // remoteQuality는 원격 사용자의 네트워크 상태에 대한 평가 결과를 나타내며 그
결과는 원격 및 로컬 양쪽의 영향을 다 받습니다.
    switch(localQuality.quality) {
        case TRTCQuality_Unknown:
            NSLog(@"정의되지 않음");
            break;
        case TRTCQuality_Excellent:
            NSLog(@"현재 네트워크 상태가 매우 좋음");
            break;
        case TRTCQuality_Good:
            NSLog(@"현재 네트워크 상태가 비교적 좋음");
            break;
        case TRTCQuality_Poor:
            NSLog(@"현재 네트워크 상태가 보통임");
            break;
        case TRTCQuality_Bad:
```

```
            NSLog(@"현재 네트워크 상태가 나쁨");
            break;
        case TRTCQuality_Vbad:
            NSLog(@"현재 네트워크 상태가 매우 나쁨");
            break;
        case TRTCQuality_Down:
            NSLog(@"현재 네트워크가 RTC Engine 최소 요구 사항을 충족하지 못
함");
            break;
        default:
            break;
    }
}
```

고급 권한의 관리

RTC Engine 고급 권한 관리는 다양한 방에 대해 서로 다른 입장 권한을 설정하는 데 사용할 수 있습니다(예: 고급 VIP 방). 또한 청취자의 마이크 권한을 관리하는 데도 사용할 수 있습니다(예: 유령 마이크 처리). 구체적인 조작 단계는 다음과 같습니다.

1. RTC Engine 콘솔의 기능 구성 페이지에서 고급 권한 관리 스위치를 켭니다.
2. 업무 백엔드에서 PrivateMapKey를 생성하고 코드 예시는 PrivateMapKey 계산원 코드를 참조하세요.
3. 입장 검증 & 마이크 연결 검증 PrivateMapKey.
   - 입장 검증

```
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = SDKAppID;
params.roomId = self.roomId;
params.userId = self.userId;
// 업무 백엔드에서 가져온 UserSig
params.userSig = [self getUserSig];
// 업무 백엔드에서 얻은 PrivateMapKey입니다
params.privateMapKey = [self getPrivateMapKey];
params.role = TRTCRoleAudience;
[self.trtcCloud enterRoom:params appScene:TRTCAppSceneVoiceChatRoom];
```

   - 마이크 연결 검증

```
// 업무 백엔드에서 얻은 최신 PrivateMapKey를 역할 전환 인터페이스에 전달합니다
```

```
[self.trtcCloud switchRole:TRTCRoleAnchor privateMapKey:[self
getPrivateMapKey]];
```

이상 처리

고장및 오류 처리

RTC Engine SDK에서 복구할 수 없는 오류가 발생하면 `onError` 콜백에서 나오며, 자세한 내용은 오류 코드표 을 참조하십시오.

- UserSig 관련.

  UserSig 검증 실패로 인해 방 입장에 실패할 수 있으며, UserSig 도구 을 사용하여 검증할 수 있습니다.

열거형	값	설명
ERR_TRTC_INVALID_USER_SIG	-3320	방 입장 매개변수 userSig가 올바르지 않습니다. `TRTCParams.userSig` 이 비어 있는지 확인하세요.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	UserSig 검증에 실패했습니다. 매개변수 `TRTCParams.userSig` 이 올바르게 입력되었는지 또는 만료되지 않았는지 확인하세요.

- 입장 및 퇴장 관련.

  방 입장 실패 시 먼저 입장 매개변수가 올바른지 확인하고 입장 및 퇴장 인터페이스는 반드시 쌍으로 호출해야 합니다. 입장에 실패한 경우에도 퇴장 인터페이스를 호출해야 합니다.

열거형	값	설명
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	방 입장 요청이 시간 초과하는 경우, 네트워크 연결이 끊겼는지 또는 VPN이 켜져 있는지 확인하세요. 4G로 전환하여 테스트할 수도 있습니다.
ERR_TRTC_INVALID_SDK_APPID	-3317	입장 매개변수 sdkAppId가 잘못되었습니다. `TRTCParams.sdkAppId` 이 비어 있는지 확인하세요.
ERR_TRTC_INVALID_ROOM_ID	-3318	입장 매개변수 roomId가 잘못되었습니다. `TRTCParams.roomId` 또는 `TRTCParams.strRoomId` 이 비어 있는지 확인하세요. roomId와 strRoomId는 혼용할 수 없습니다.
ERR_TRTC_INVALID_USER_ID	-3319	입장 매개변수 userId가 올바르지 않습니다. `TRTCParams.userId` 이 비어 있는지 확인하세요.
ERR_TRTC_ENTER_ROOM_RE	-334	입장 요청이 거부되었습니다. `enterRoom` 을 연속적

FUSED	0	으로 호출하여 동일한 ID의 방에 입장했는지 확인하세요.

- 장치 관련.

  장치 관련 오류를 감지할 수 있으며 관련 오류 발생 시 UI에서 사용자에게 알립니다.

열거형	값	설명
ERR_MIC_START_FAIL	-1302	Windows 또는 Mac 장치에서 마이크 구성 프로그램(드라이버)이 비정상일 경우, 마이크 열 수가 없습니다. 장치를 비활성화한 후 다시 활성화하거나, 기기를 재시작하거나, 구성 프로그램을 업데이트하세요.
ERR_SPEAKER_START_FAIL	-1321	스피커 열기가 실패됩니다. 예를 들어 Windows 또는 Mac 장치에서 스피커 구성 프로그램(드라이버)이 비정상일 경우, 장치를 비활성화한 후 다시 활성화하거나 기기를 재시작하거나 구성 프로그램을 업데이트하세요.
ERR_MIC_OCCUPY	-1319	마이크가 사용 중입니다. 예를 들어 모바일 장치에서 통화 중일 때 마이크를 열 수가 없습니다.

오류 종료 처리

1. 네트워크 연결 감지 및 퇴장 타임아웃.

   다음 콜백을 통해 RTC Engine 연결 끊김 및 재연결 이벤트 알림을 모니터링할 수 있습니다. `onConnectionLost` 콜백을 수신한 후 로컬 마이크 순위 UI에서 네트워크 연결 끊김을 표시하여 사용자에게 알려줍니다. 동시에 로컬에서 타이머를 시작하고 설정된 시간 임계값을 초과한 후에도 `onConnectionRecovery` 콜백을 수신하지 못하면 네트워크가 계속 끊긴 상태입니다. 이때 로컬에서 마이크 종료 및 방 나가기 프로세스를 시작하고 사용자에게 방을 나갔음을 알리는 팝업을 보낸 후 페이지를 파기합니다. 네트워크 연결이 90초(기본값) 이상 끊어지면 타임아웃으로 인한 방 나가기가 트리거되며 RTC Engine 서버는 해당 사용자를 방에서 강제 퇴장시킵니다. 해당 사용자가 스트리머 역할인 경우 방 내 다른 사용자는 `onRemoteUserLeaveRoom` 콜백을 받게 됩니다.

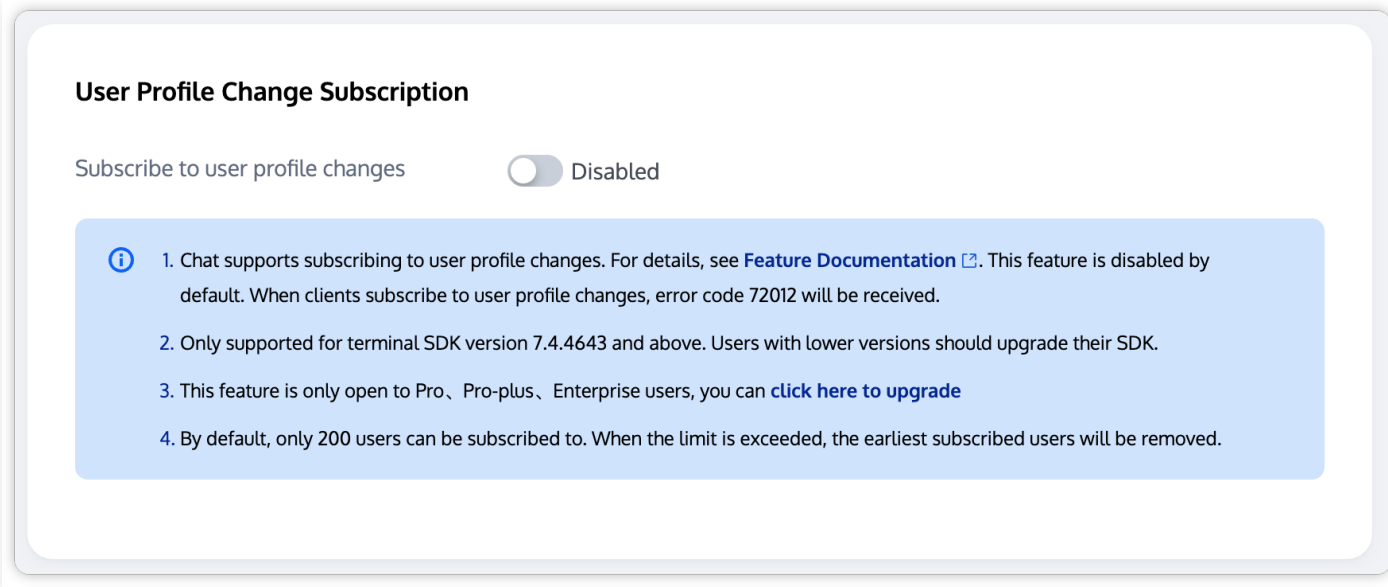
```
#pragma mark - TRTCCloudDelegate

- (void)onConnectionLost {
    // SDK와 클라우드 간의 연결이 끊어졌습니다.
}

- (void)onTryToReconnect {
    // SDK가 클라우드에 다시 연결을 시도 중입니다.
}
```

```
- (void)onConnectionRecovery {
    // SDK와 클라우드 간의 연결이 복구되었습니다.
}
```

2. 오프라인 상태에서 자동으로 마이크의 사용을 끕니다.

   Chat 사용자의 일반 상태는 온라인(ONLINE), 오프라인(OFFLINE), 미로그인(UNLOGINED)으로 구분됩니다.오프라인 상태는 일반적으로 사용자가 프로세스를 강제 종료하거나 네트워크가 비정상적으로 중단된 경우 발생합니다. 스트리머 구독을 통해 오프라인 상태의 마이크 사용자를 감지하여 마이크의 사용을 끄게 시킬 수 있습니다.

```
// 스트리머 구독을 통해 마이크 사용자의 상태를 감지합니다
[[V2TIMManager sharedInstance] subscribeUserStatus:userList succ:^{
    // 스트리머 구독 사용자의 상태가 성공합니다
} fail:^(int code, NSString *desc) {
    // 스트리머 구독 사용자의 상태가 실패합니다
}];

// 스트리머 구독을 통해 마이크 사용자의 상태 취소합니다
[[V2TIMManager sharedInstance] unsubscribeUserStatus:userList succ:^{
    // 스트리머 구독 사용자의 상태를 취소하는 것이 성공합니다
} fail:^(int code, NSString *desc) {
    // 스트리머 구독 사용자의 상태를 취소하는 것이 실패합니다.
}];

// 사용자 상태 변경의 알림 및 처리
[[V2TIMManager sharedInstance] addIMSDKListener:self];

- (void)onUserStatusChanged:(NSArray<V2TIMUserStatus *>
*)userStatusList {
    for (V2TIMUserStatus *userStatus in userStatusList) {
        NSString *userId = userStatus.userID;
        V2TIMUserStatusType status = userStatus.statusType;
        if (status == V2TIM_USER_STATUS_OFFLINE) {
            // 오프라인 상태에서 마이크 사용 끄게 시킵니다
            [self kickSeatWithIndex:[self
getSeatIndexWithUserId:userId]];
        }
    }
}
```



⚠ 주의:

- 스트리머 구독 사용자의 상태는 프로페셔널 버전 패키지로 업그레이드해야 합니다. 자세한 내용은 기본 서비스 내역를 참조하세요.
- 스트리머 구독 사용자의 상태는 사전에 Chat 콘솔에서 **사용자 상태 조회 및 상태 변경 알림 구성**을 활성화해야 합니다. 활성화하지 않으면 `subscribeUserStatus` 호출 시 오류가 발생합니다.

서버에서 사용자를 강제 퇴장 시키고 방 해산시킵니다

1. 서버에서 사용자를 강제 퇴장시킵니다.

   먼저 RTC Engine 서버에서 사용자 강제 퇴장 인터페이스 RemoveUser(정수형 방 번호) 또는 RemoveUserByStrRoomId(문자열 방 번호)를 호출하여 대상 사용자를 RTC Engine 방에서 퇴장시킵니다. 입력 예시는 다음과 같습니다.

```
https://trtc.tencentcloudapi.com/?Action=RemoveUser
&SdkAppId=1400000001
&RoomId=1234
&UserIds.0=test1
&UserIds.1=test2
&<공통 요청 매개변수>
```

사용자 강제 퇴장이 성공적으로 실행되면 대상 사용자는 클라이언트에서 `onExitRoom()` 콜백을 수신하며 `reason` 값은 1입니다. 이때 해당 콜백에서 마이크 사용 종료및 Chat 그룹 퇴장 등의 작업을 처리할 수 있습니다.

```
// RTC Engine 방 이탈 이벤트의 콜백
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        // exitRoom을 호출하여 방에서 나가기.
        NSLog(@"Exit current room by calling the 'exitRoom' api of sdk
...");
    } else {
        // reason 1: 서버에 의해 현재 방에서 나가게 됩니다
        // reason 2: 현재 방 전체가 해산됩니다
        NSLog(@"Kicked out of the current room by server or current
room is dissolved ...");
        // 마이크 사용 종료
        [self leaveSeatWithIndex:seatIndex];
        // Chat 그룹에서 나가기
        [[V2TIMManager sharedInstance] quitGroup:groupID succ:^{
            //그룹 나가기 성공
        } fail:^(int code, NSString *desc) {
            // 그룹 나가기 실패
        }];
    }
}
```

2. 서버에서 방을 해산시킵니다.

   먼저 Chat 서버의 그룹 해산 인터페이스 destroy_group 을 호출하여 대상 그룹을 해산합니다.요청 URL 예시는 다음과 같습니다.

```
https://xxxxxx/v4/group_open_http_svc/destroy_group?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&conten
ttype=json
```

   그룹 해산이 성공적으로 실행되면 대상 그룹의 모든 멤버는 클라이언트에서 `onGroupDismissed()` 콜백을 받게 됩니다. 이때 해당 콜백에서 RTC Engine 방 나가기 등의 작업을 처리할 수 있습니다.

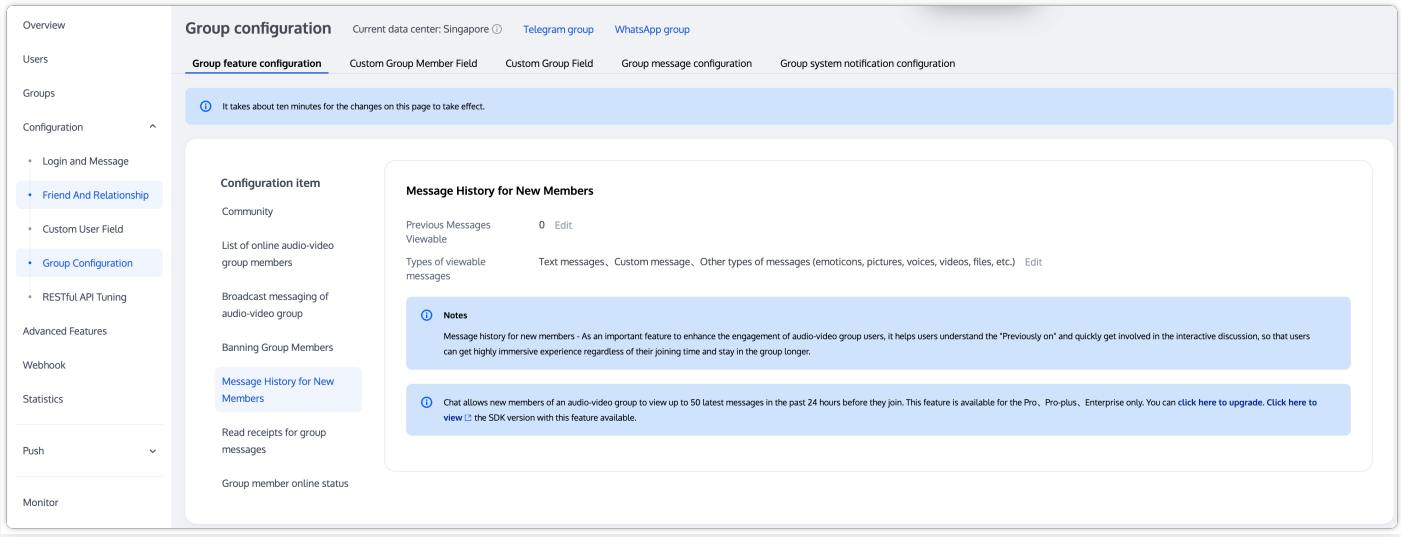
```
// 그룹 해산 콜백
```

```
[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onGroupDismissed:(NSString *)groupID opUser:
(V2TIMGroupMemberInfo *)opUser {
    // RTC Engine 방 나가기
    [self.trtcCloud stopLocalAudio];
    [self.trtcCloud exitRoom];
}
```

설명:
방 내 모든 사용자가 `exitRoom()` 을 호출하여 방 나가기를 완료하면 RTC Engine 방이 자동으로 해산됩니다. 물론 서버 측의 인터페이스 DismissRoom(정수형 방 번호) 또는 DismissRoomByStrRoomId(문자열 방 ID)를 호출하여 RTC Engine 방을 강제로 해산시킬 수도 있습니다.

방에 입장해서 라이브 방송 기록을 확인합니다

AVChatRoom은 기본적으로 라이브 방송실의 이전 메시지를 저장하지 않습니다.새 사용자가 방송실에 입장하면 입장 후의 사용자가 보낸 메시지만 볼 수 있습니다. 새로 입장한 사용자의 체험을 개선하기 위해 콘솔에서 라이브 방송 그룹 사용자가 입장 전의 메시지 수량을 구성할 수 있습니다. 하기 그림과 같습니다.



라이브방송 그룹 사용자가 입장 전의 이전 메시지는 다른 그룹의 이전 메시지와 동일합니다.코드 예시는 하기와 같습니다.

```
V2TIMMessageListGetOption *option = [[V2TIMMessageListGetOption alloc]
init];
option.getType = V2TIM_GET_CLOUD_OLDER_MSG; // 클라우드의 이전 메시지 가져오
기
option.getTimeBegin = 1640966400;         // 2022-01-01 00:00:00부터 시작
option.getTimePeriod = 1 * 24 * 60 * 60; // 하루 종일의 메시지 가져오기
option.count = INT_MAX;                   // 시간 범위 내 모든 메시지로 반환하
기
option.groupID = #your group id#;         // 그룹 채팅 메시지 가져오기
[V2TIMManager.sharedInstance getHistoryMessageList:option
succ:^(NSArray<V2TIMMessage *> *msgs) {
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

> ⚠ 주의:
> 이 기능은 프리미엄 버전 사용자만 이용할 수 있으며 그룹 생성 후 24시간 이내에 최대 20개의 이전 메시지만 지원합니다.

방 입장 시 마이크 사용중의 스트리머의 무음 상태를 감지합니다

방안 1: 방 입장 시 모든 스트리머를 기본적으로 무음 상태로 설정한 후, `onUserAudioAvailable(userId, true)` 콜백을 통해 해당 스트리머의 무음상태를 해제합니다.

```
- (void)onUserAudioAvailable:(NSString *)userId available:
(BOOL)available {
    if (available) {
        // 해당 스트리머의 무음 상태를 해제합니다
    }
}
```

방안 2: 스트리머의 무음상태를 Chat 그룹 속성에 저장하고 청취자가 방에 입장하면 전체 그룹 속성을 받아서 마이크 사용중의 스트리머의 무음 상태를 확인할 수 있습니다.

```
[[V2TIMManager sharedInstance] getGroupAttributes:groupID keys:nil
succ:^(NSMutableDictionary<NSString *,NSString *> *groupAttributeList) {
```

```
    // 그룹 속성 가져오기가 성공되며 스트리머 무음 상태를 저장하는 키를
muteStatus로 가정합니다
    NSString *muteStatus = groupAttributeList[@"muteStatus"];
    // muteStatus를 분석하여 각 마이크를 사용하는 스트리머의 무음 상태를 가져옵니
다
} fail:^(int code, NSString *desc) {
    // 그룹 속성 가져오기가 실패됩니다
}];
```

Online Karaoke Scenario Solution

최종 업데이트 날짜:: 2024-07-25 17:01:48

Scenario Introduction

According to data from iiMedia Research, in 2021, the number of online Karaoke users in China was about 510 million, with a penetration rate of approximately 49.7%. Online Karaoke offers a more immersive experience and its diverse gameplay caters to the personalized needs of different user groups, becoming one of the main projects in the online pan-entertainment field. Based on network technology innovations, online Karaoke apps continue to launch diverse singing patterns and gameplay, and the continuously enriched features have enhanced the practicality and playability of online Karaoke apps. This document will provide a detailed introduction to the online Karaoke scenario-based solution based on Tencent Real-Time Communication (TRTC) in the following sections.

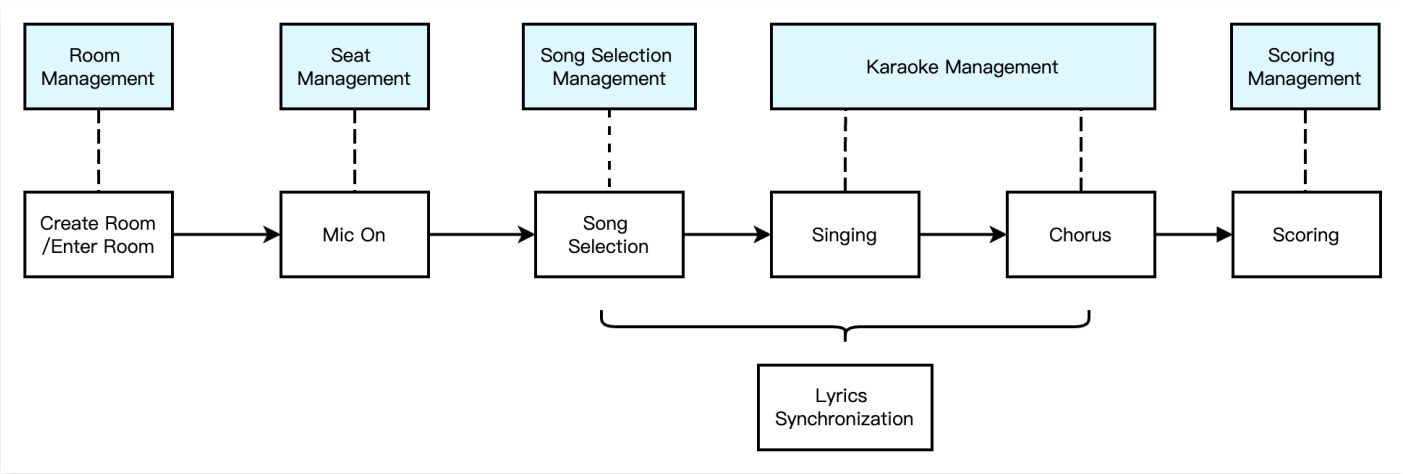
Implementation Scheme

Typically, implementing a complete online Karaoke scenario involves multiple functional modules: room management, seat management, song selection management, karaoke management, scoring management, etc. Key actions and feature points under each functional module are shown in the following table:

Functional Module	Key Actions and Feature Points
Room Management	Room list, create a room, enter a room, exit a room, and terminate a room.
Seat Management	Become a speaker/listener, seat control, change a speaker,

	lock the seat, invite a listener to speak, and mute a speaker.
Song Selection Management	Song list display, searching songs, song selection, queue management, and selected song list.
Karaoke Management	Karaoke play mode, start/stop/switch songs, accompaniment and vocal volume adjustment, reverb/sound effects, original sound and accompaniment switch, and lyrics synchronization
Scoring Management	Singing Scoring and Pitch Line Display

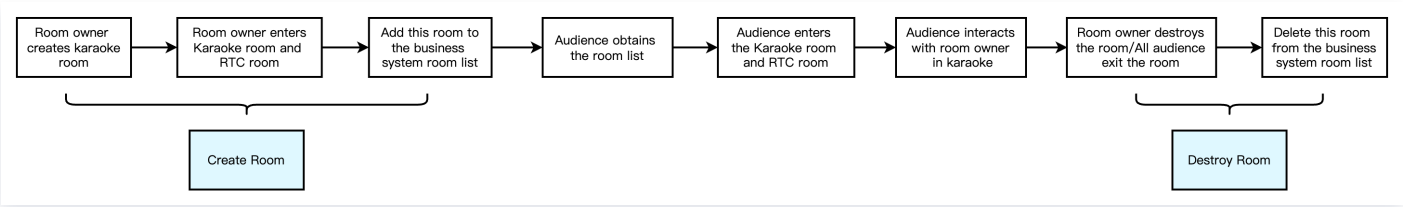
The overall business process of the online Karaoke scene is shown in the following diagram. The room owner creates a Karaoke room, and users can choose the Karaoke room they are interested in to enter. After entering the room, users can request to speak to participate in the interaction. After becoming a speaker, they can also choose their favorite songs to sing and wait in line. When it is their turn, they can sing along with the accompaniment. Of course, users can also choose to become a speaker directly to participate in a chorus. These are two different Karaoke play modes. During the singing process, there will be pitch scoring for individual sentences, and there will also be singing scoring for the entire song after the singing is finished.



Room Management

The Room Management module is primarily responsible for maintaining the room list, which includes functions such as create a room, enter a room, exit a room, and terminate a room. Additionally, a Karaoke room differs from regular rooms in that it requires a separate Karaoke room identifier to initiate related component management: Song Selection Management, Karaoke Management, Scoring Management, etc.

- Create Room: After users log in to the business system, they can create a room. The room list needs to be updated after a room is created.
- Enter a Room: Users can choose to enter an existing room. Upon entering, the current list of room members should be updated.
- Exit a Room: Users can choose to exit the current room. Upon exiting, the current list of room members needs to be updated with a delete operation.
- Terminate Room: After all users exit the room, it needs to be terminateed. Upon destruction, the room list needs to be updated with a delete operation.



Note:
Room Management is a necessary functional module for implementing online Karaoke but is not the main functional module. Specific implementation can be achieved through integration with business systems and IM&TRTC SDKs. For details, see Voice Chat Room > Room Management.

Seat Management

In a Karaoke room, seats are generally orderly and limited. Seat management primarily involves defining the number of seats in the room based on the business scenario, as well as managing the status of all seats in the current room. Seat management includes features such as become a speaker/listener, seat control, change a speaker, lock the seat, invite a listener to speak, and mute a speaker.

- After users enter a room, only idle seats can be applied for.
- After the room owner approves a user's speaker request, the corresponding seat status should change to occupied.
- When a user stops streaming and becomes a listener, the corresponding seat status should revert to idle.
- The room owner has the authority to lock the seat, invite a listener to speak, remove a speaker, mute a speaker, etc.

Note:
Seat management is a necessary functional module for implementing online Karaoke but is not the main functional module. Specific implementation can be achieved through integration with business

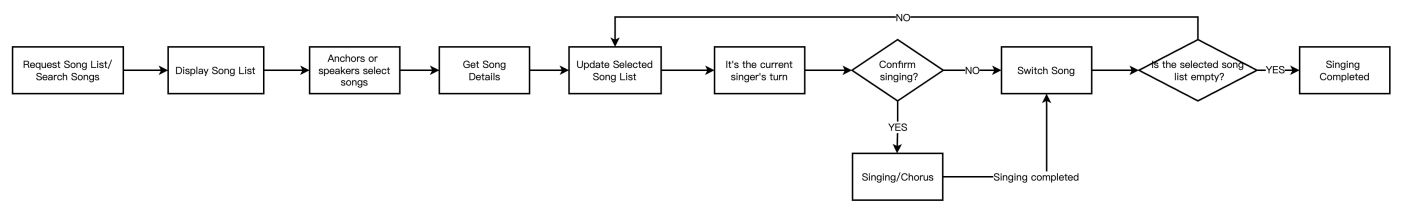
systems and IM&TRTC SDKs. For details, see Voice Chat Room –> Seat Management.

Song Selection Management

Basic Introduction

Song selection management is an important part of the online Karaoke scenario, mainly including features such as song list display, search for songs, song selection, and queue management, selected song list. Each Karaoke room needs to maintain a selected song list and an auto queue management feature, which needs to be implemented by the business backend. Meanwhile, aspects related to accompaniment resources such as song list display and song search are recommended to be implemented with accompaniment library products for overseas users.

Implementation Process



The entire song selection management, mainly involves the business-side app, business backend, and music library, where their respective functions are as follows:

- Business-Side App:
  - Call the song selection API to report song information.
  - Call the change song-switching API to notify the business backend.
  - Call the singing confirmation API to notify the business backend.
- Business Backend:
  - Maintain the selected song list.
  - Send notifications to tell the business-side app to switch the song.
- Music Library:
  - Provide authorized music resources for TRTC to play.
  - Provide lyric files and pitch files matching the music resources.

Karaoke Management

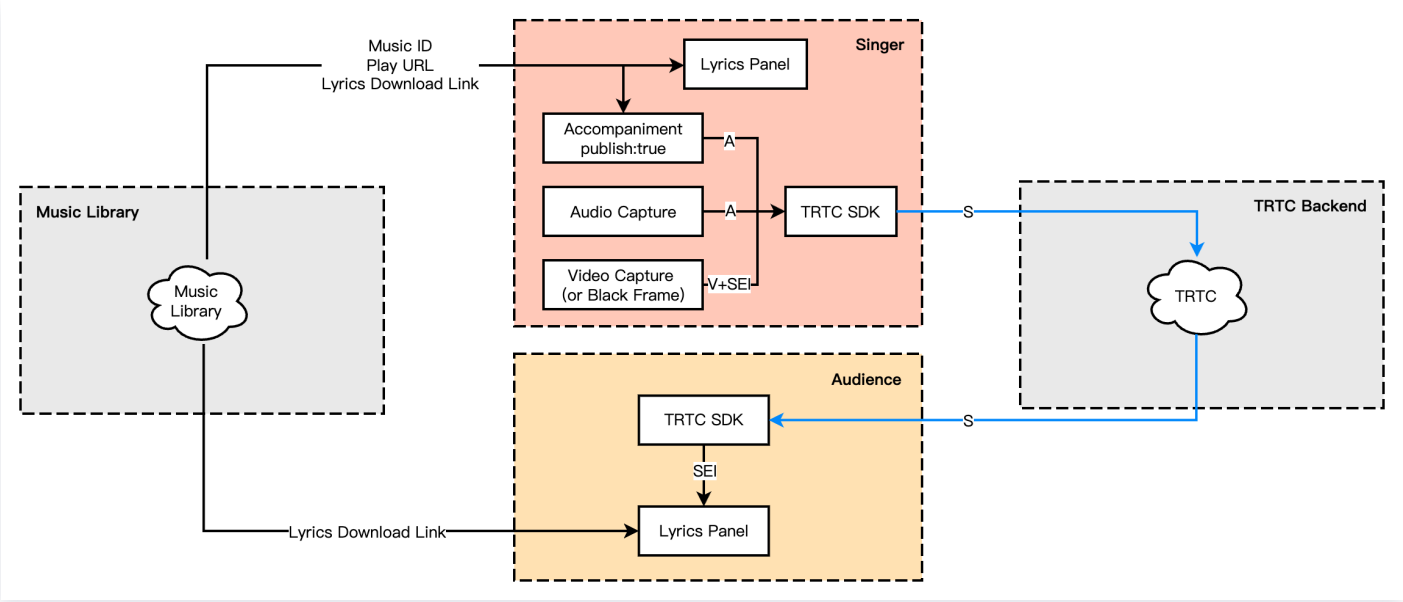
The Karaoke system primarily includes functions such as: Karaoke play mode, start/stop/switch songs, accompaniment and vocal volume adjustment, reverb/sound effects, original sound and accompaniment switch, and lyrics synchronization. Below, we will introduce the implementation process of the Karaoke management module in detail through two typical Karaoke gameplay: solo singing and real-time chorus.

Solo Singing

Solo singing: Primarily in the interactive Karaoke scenario with multiple participants, after the anchor/audience members become speakers, they can proceed to select songs. Once a song selection is successful, it will be displayed collectively on the song selection platform. When it's someone's turn to select a song, the corresponding individual will play the song's accompaniment, start singing, and undergo scoring.

Solution Architecture

The overall solution primarily relies on the music library for song and lyric resources and TRTC for streaming the singer's vocals, song accompaniment, and streaming. The solution architecture is as follows:



Specific Implementation

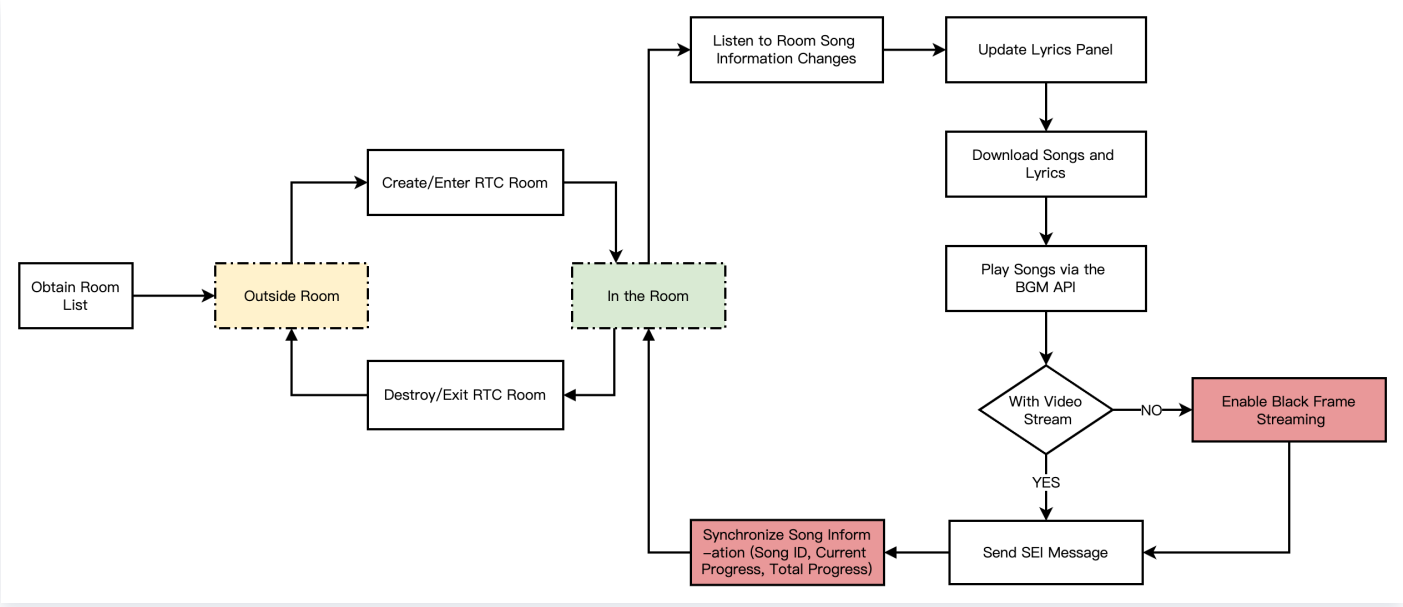
In the solo singing scenario, different roles have different implementation processes. There are two roles: singer and audience. The description and differences of their roles are detailed in the table below:

Roles	Description	Differences
Singer	The singer in the Karaoke room is evolved	• The role must be an anchor.

	from the anchor/audience who selects songs and sings after becoming a speaker. After leaving the room, the room is automatically dissolved and the list of selected songs is automatically cleared.	• Upstream audio and video (no video upstream black frame) • Play BGM • Send SEI information (sending lyric information) • Song Selection
Audience	The audience in the Karaoke room plays the media stream of the singer or other people.	• The role is an audience, but can also become an anchor by becoming a speaker. • Downstream Audio and Video Streams • Receive SEI information (receive lyric information)

Implementation Process

- **Singer**



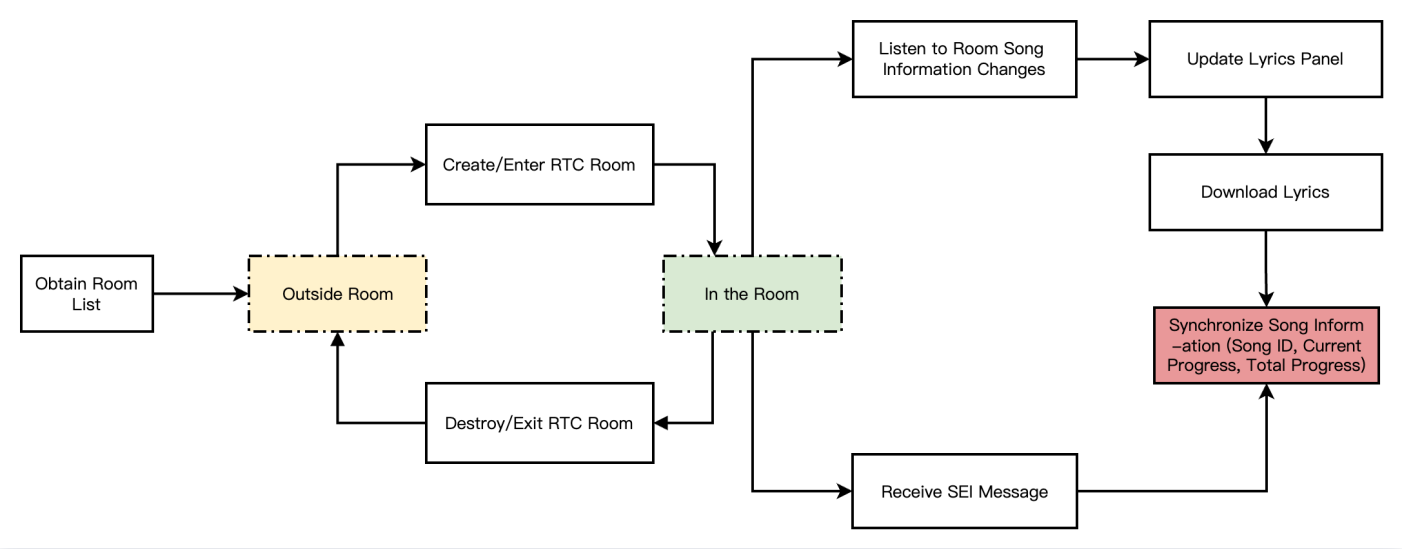
1. The anchor/audience creates/enters a TRTC room, and automatically becomes a singer after selecting a song.
2. After the singer selects a song, the song/lyric is downloaded, and then the song is played through the

BGM interface.

3. If the singer does not bring up the video upstream, they need to enable the video upstream.
4. Synchronize the lyric progress of everyone through SEI information.
5. When the singer becomes a listener, all songs they selected will be cleared, and they revert to their original role.
6. After the anchor/audience exits the room, the TRTC room will be dissolved.

> **Note:**
> Anchors/Audiences on the seat can select songs for themselves or others, but the corresponding singer must play the BGM; otherwise, it may cause asynchrony between the singer and the song due to latency (about 300 ms or more).

- **Audience**



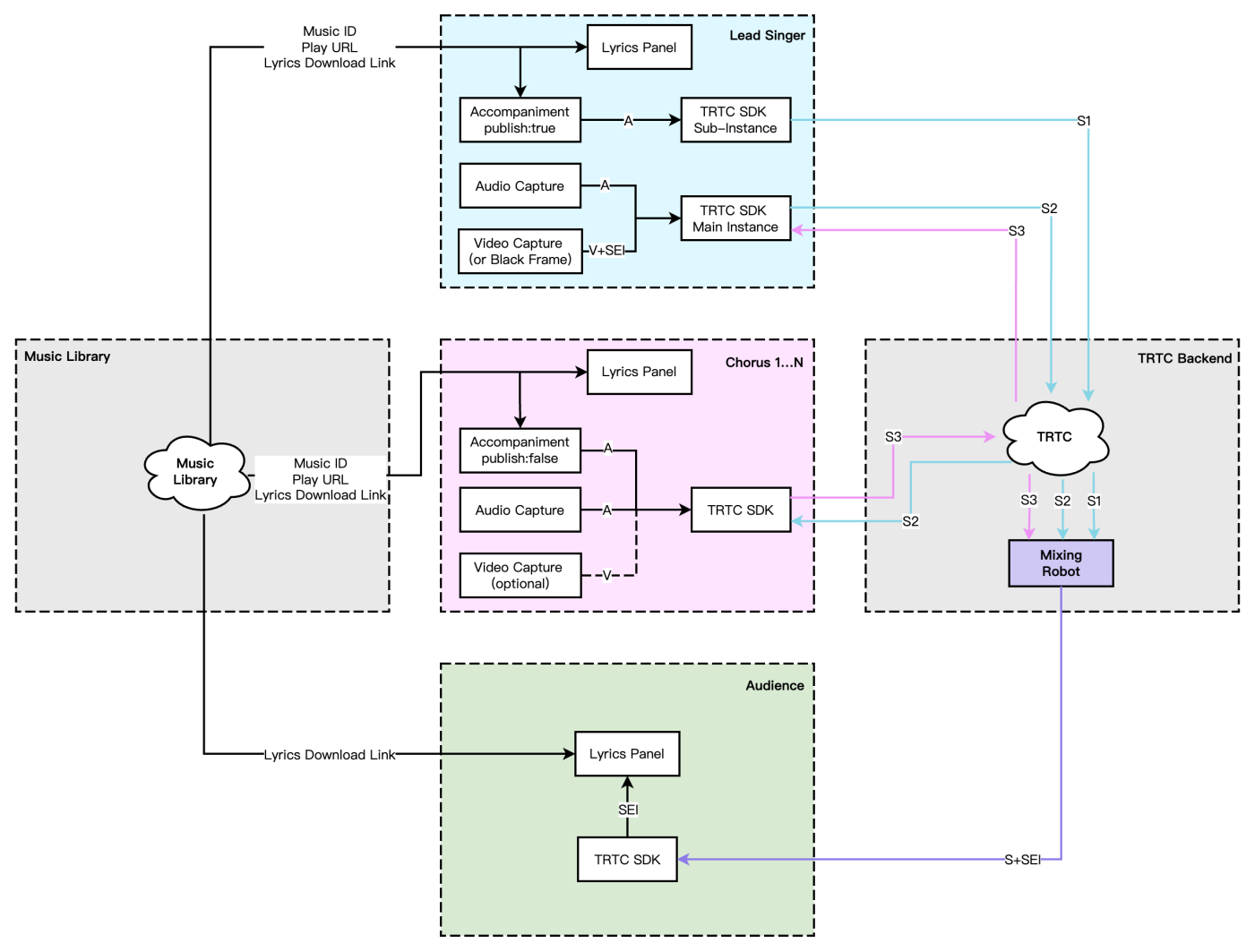
1. Anchor/Co-Anchor/Audience can create/enter a TRTC room.
2. Monitor room song changes and load lyrics.
3. Pull the singer's stream.
4. Parse the SEI messages sent by the singer and synchronize the lyrics.

Real-Time Chorus

Real-time chorus refers to playing the song accompaniment simultaneously on all ends based on co-mic, and then performing the chorus on the mic. In a duo pattern, the lead and backing vocals can hear each other; in a multi-person pattern, all choristers can hear each other with almost no delay, achieving true real-time chorus.

Solution Architecture

In terms of media streams, the singers publish/playback streams to each other, while one leading singer uploads accompaniment, and the other singers play accompaniment locally, synchronized via NTP. Additionally, the accompaniment and all singers' voices are mixed through a mixing bot to form a single stream, which is then pushed back to the TRTC room, allowing the audience to hear the synchronized voices from all ends by pulling a single stream, achieving a multi-person chorus effect. The real-time chorus solution architecture is shown in the figure below:



The advantages of this solution are:

- It reduces end-to-end latency.
- It provides a solution for users to join the chorus midway.
- It accurately synchronizes accompaniment, lyrics, and vocals between different ends.
- It improves the performance of devices on different ends and the accuracy of local time, and reduces

the impact of network environment latency.

> **Note:**
> - Depending on business needs, you can choose a real-time chorus solution for audio-only or audio and video scenarios. If it is a pure audio scenario, black frames need to be added to send SEI messages for lyric synchronization.
> - The lead singer needs to use a sub-instance to upstream both the accompaniment and vocals at the same time; other singers only need to pull each other's vocal streams and play accompaniment locally; the audience only needs to pull one mixed stream.

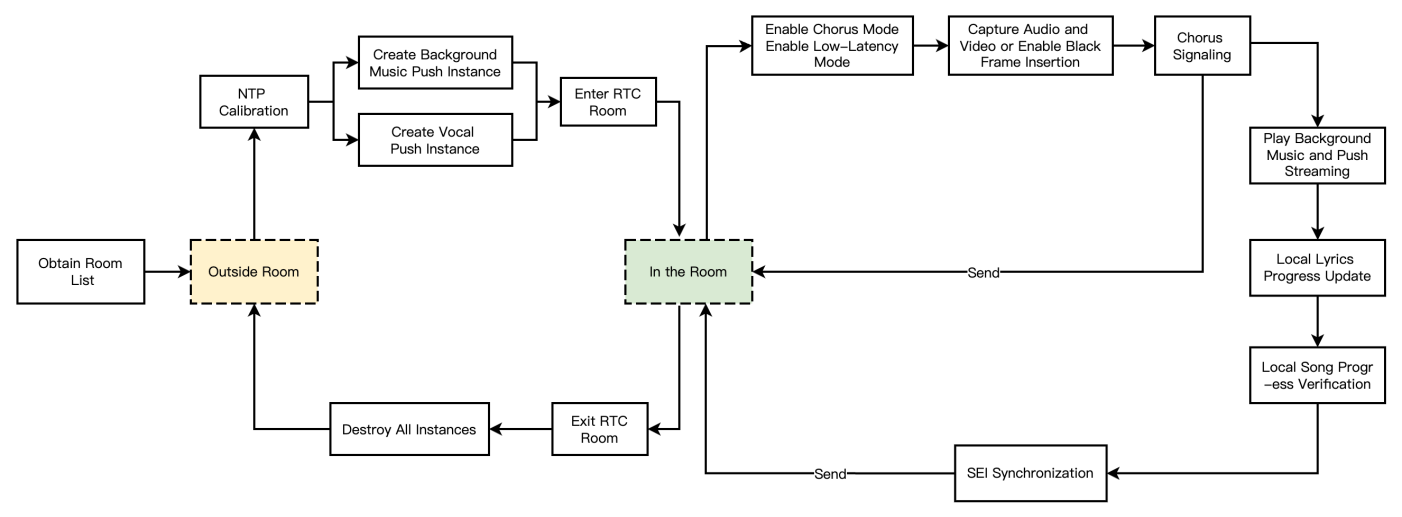
Specific Implementation

In online Karaoke rooms, different roles have different feature permissions and implementation processes, divided into three roles: lead singer, chorus, and audience, as shown in the table below:

Roles	Description	Differences
Lead Singer	The lead singer is responsible for selecting songs, sending chorus signalings, and sending SEI messages.	• The role must be an Anchor. • Upstream accompaniment and vocals. • Song selection and initiating chorus. • Push Back Mixed Stream. • Send SEI Message
Choruss	Chorus can receive and process chorus signalings, and participate in the chorus on the seat.	• The role must be an Anchor. • Upstream Vocals • Play Accompaniment Locally • Receive Chorus Signals
Audience	After entering the Karaoke room, the audience can pull the stream from the seat and also participate in the chorus on the seat.	• The role must be an audience. • Downstream mixed stream • Receive SEI messages • Request to Become an Anchor

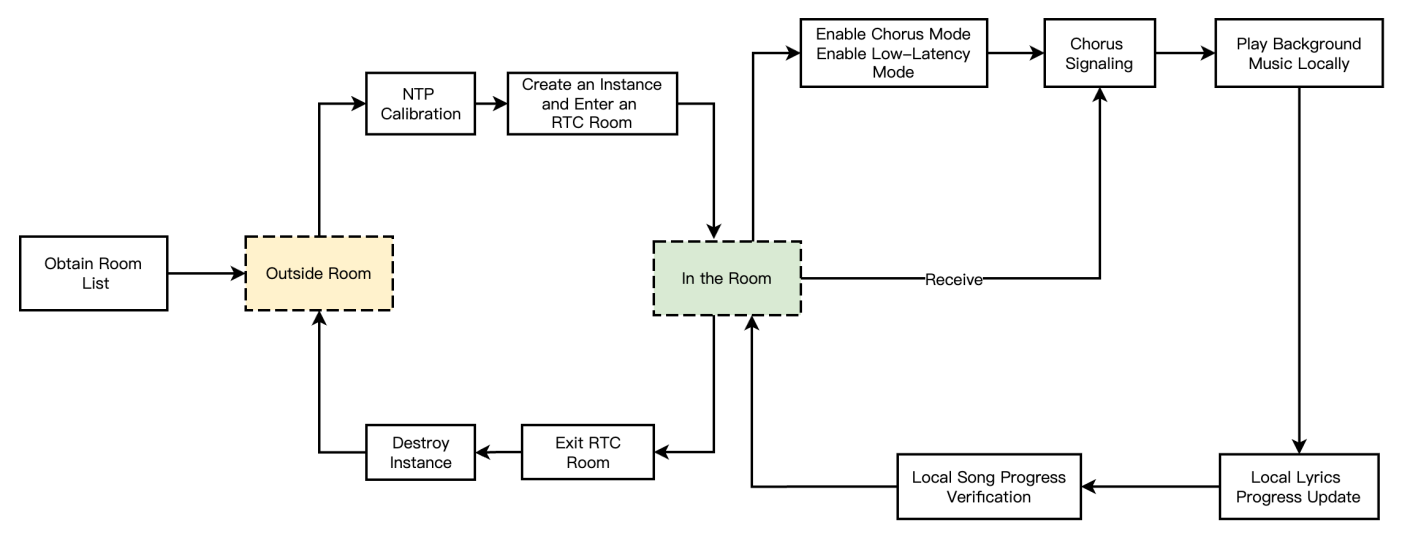
Implementation Process

- **Lead Singer**



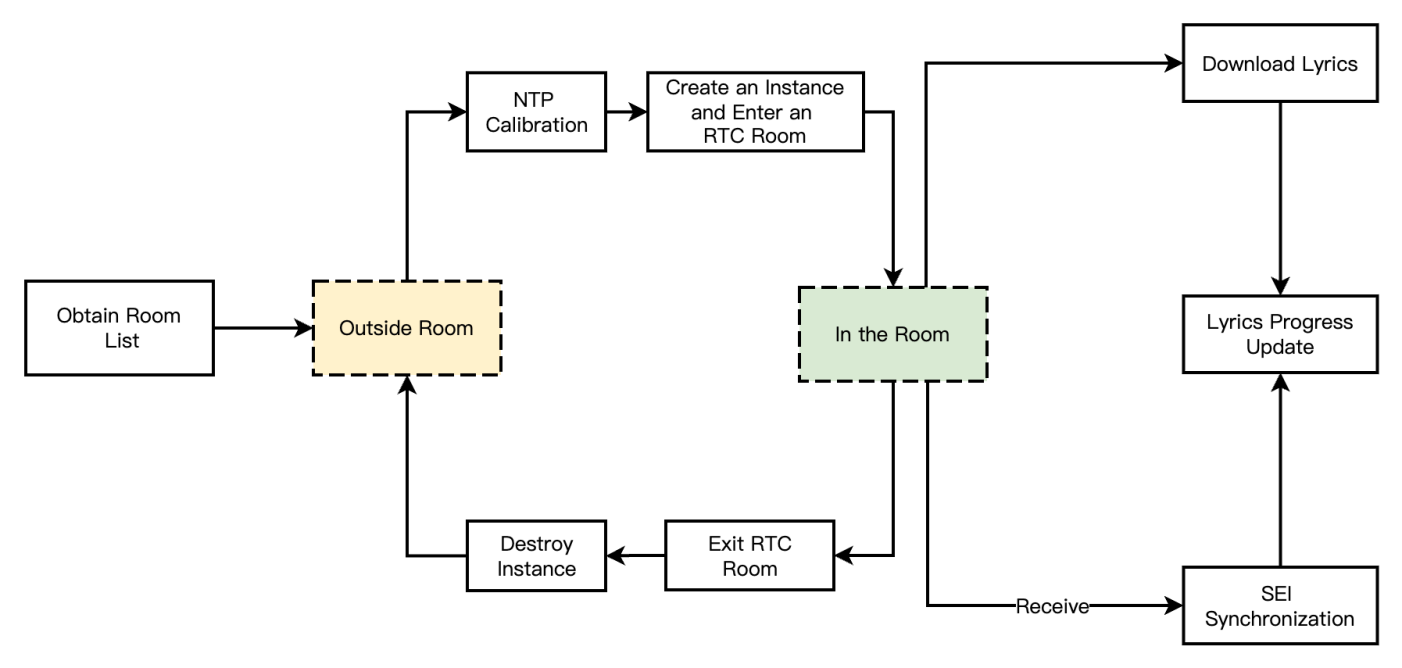
1. The lead singer needs to select songs on-demand and send chorus signalings.
2. The lead singer creates a sub-instance to push vocals and accompaniment and pulls the vocals of other singers.
3. After streaming, the lead singer is responsible for initiating the mixed stream push task.
4. After starting to sing, play the accompaniment and synchronize the lyrics through the playback progress callback.
5. SEI messages need to be sent to synchronize the song progress on the audience end.
6. AI singers need to calibrate the local song playback progress according to NTP.

- **Chorus**



1. The chorus pushes a vocal stream, pulling the vocal stream of chorus users on the seat.
2. The singers need to listen and receive chorus signalings, preloading accompaniment resources.
3. After they start to sing and play accompaniment locally, the singers synchronizes the lyrics through the playback progress callback.
4. AI singers need to calibrate the local song playback progress according to NTP.

- **Audience**



1. Upon entering the TRTC room, the audience receives the mixed chorus stream.
2. Parse the song progress information in the SEI of the mixed stream for lyric synchronization.
3. After the audience becomes a speaker, the mixed stream is stopped and switched to pulling the vocal stream on the seat, and the chorus mode is started.

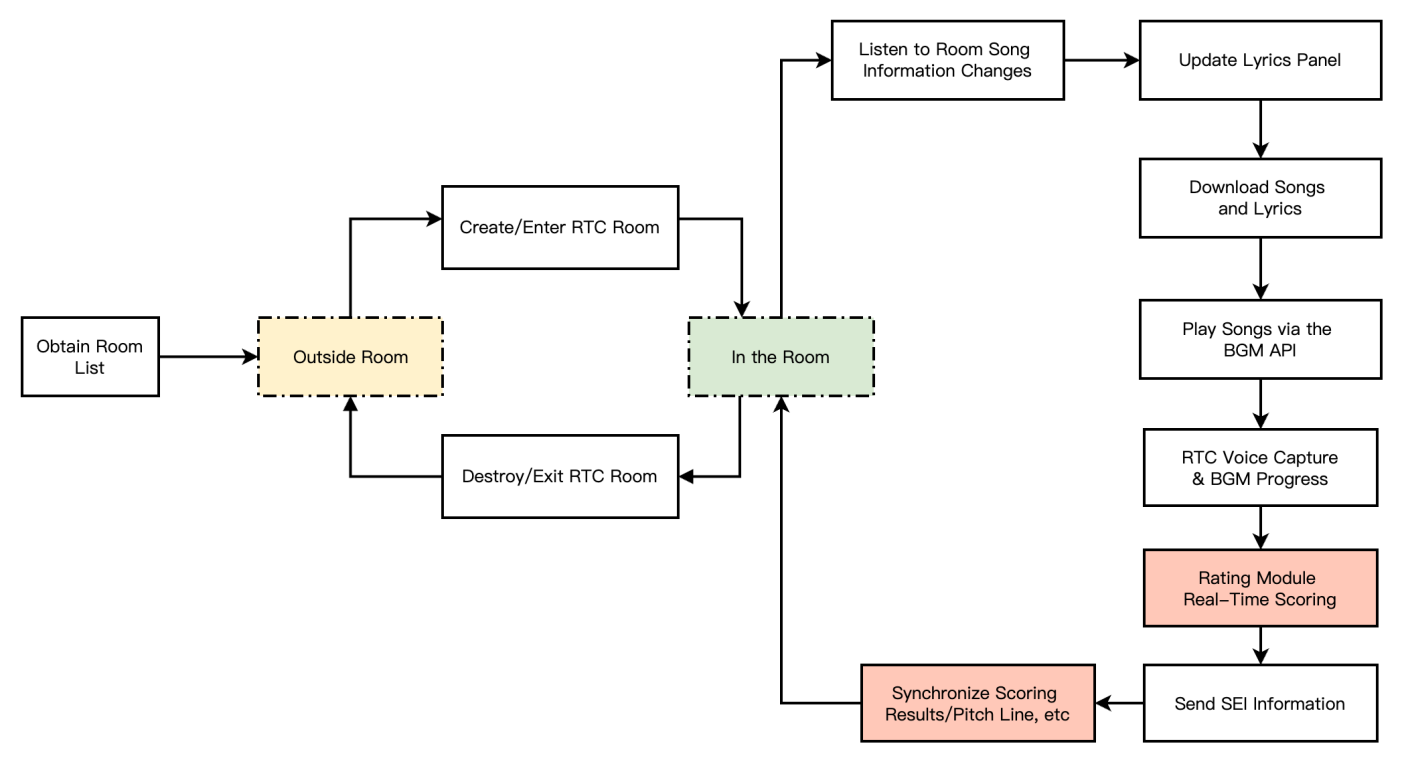
Scoring Management

Basic Introduction

The scoring function is also one of the mainstream play methods in the Karaoke scenario, mainly judging the results of pitch accuracy and sound quality during the actual singing process. It can be used for score comparisons after multi-person singing.

Implementation Process

In the entire scoring management process, singers and audiences have different implementations based on their user roles. Scoring is usually done locally by the singer and synchronized with other people in the room.
Singer



- The anchor/audience creates/enters a TRTC room, and automatically becomes a singer after selecting a song.
- After the singer selects a song, the song/lyric is downloaded, and then the song is played through the BGM interface.
- The vocals captured by TRTC and the progress of the BGM playback are transmitted in real-time to the rating module.
- After the rating module produces the data in real time, it synchronizes with everyone in the room through SEI.

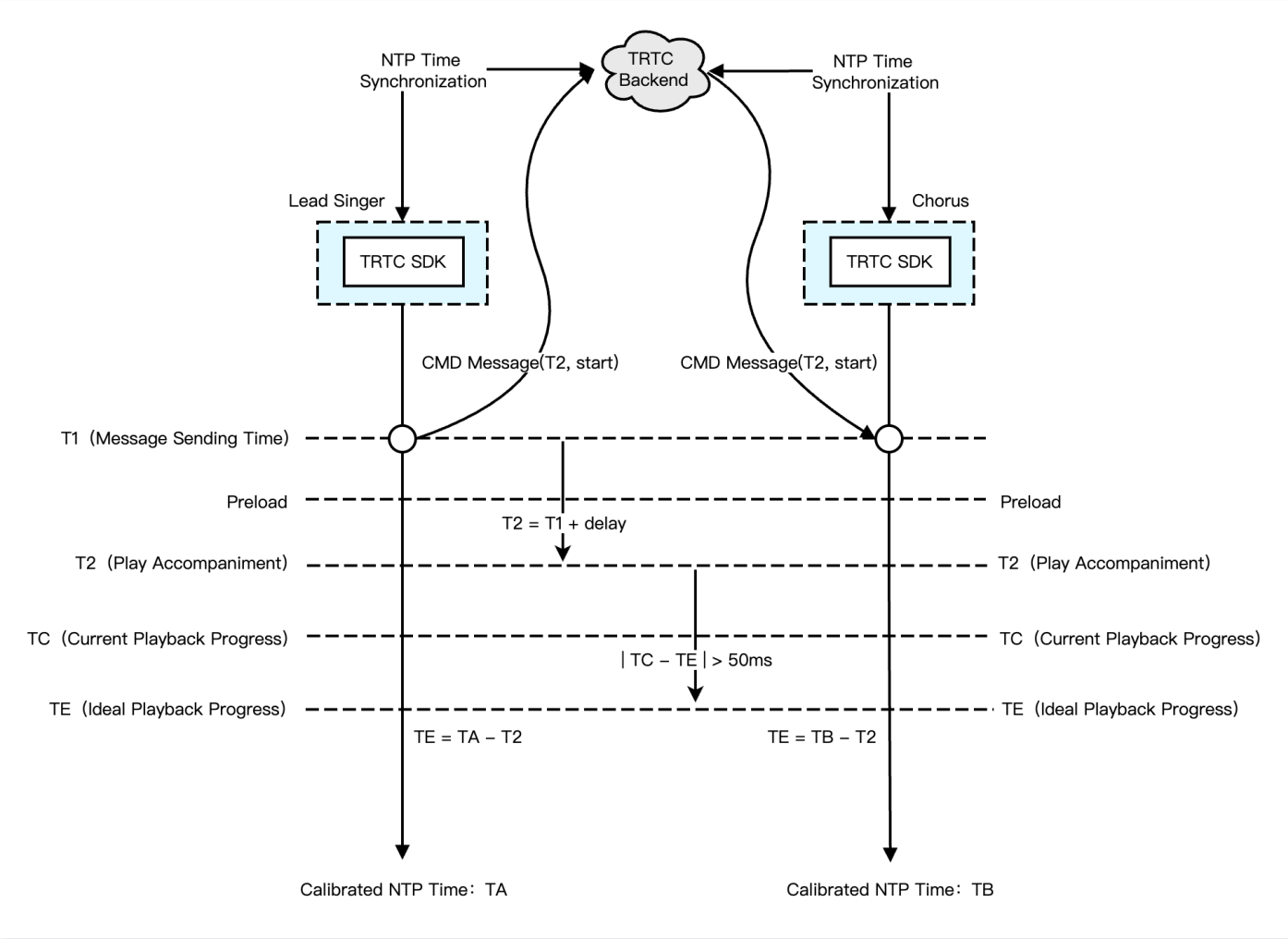
Audience

The audience side process is identical to the solo singing audience role action process; you can refer to the audience implementation process.

Key Business Logic

Accompaniment Synchronization Solution

In real-time scenarios, it is necessary to synchronize the accompaniment progress in real-time after starting the performance to avoid increasing end-to-end latency due to accompaniment errors. Synchronizing the accompaniment requires NTP time-based synchronization because the local clocks of different devices are not consistent, resulting in some errors. Therefore, Tencent Cloud's self-developed NTP service is introduced. Additionally, users who join the ensemble midway also need to synchronize the accompaniment progress. Only after synchronizing the progress can they join in the chorus.
The approach to accompaniment synchronization is: the lead singer convention starts to play the accompaniment at a future point in time (e.g., after a 3-second latency), and other users join in the chorus. All ends' time is based on NTP time, which is synchronized after the TRTC SDK initialization.



The specific process is as follows:

1. All ends calibrate the NTP time, update, and access the latest NTP time T from the TRTC cloud.
2. The lead singer sends the chorus signalings (custom message), agreeing on the chorus start time T2.
3. Preload the accompaniment locally based on T2, and schedule playback.
4. Other chorus participants follow step 3 upon receiving the chorus signalings.
5. During the process, verify the local accompaniment playback progress, and perform seek calibration when the difference between TE and TC exceeds 50 ms.

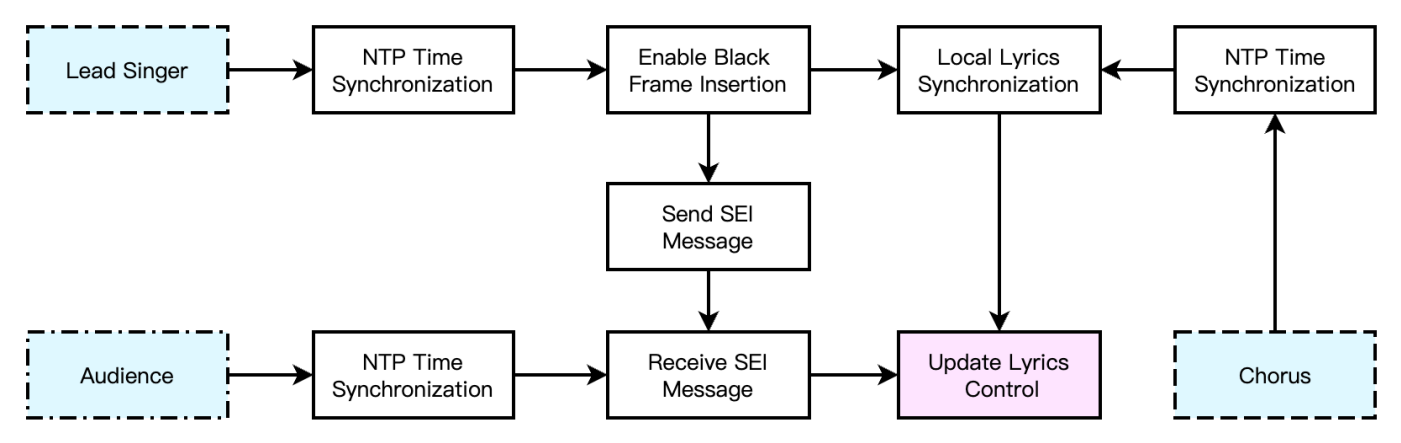
> **Note:**
> The 50 ms deviation here is a typical value, which can be adjusted appropriately based on the business tolerance, with a recommendation to fluctuate around 50 ms.

Lyrics Synchronization Solution

In the lyrics synchronization solution, the actions of three different roles are as follows:

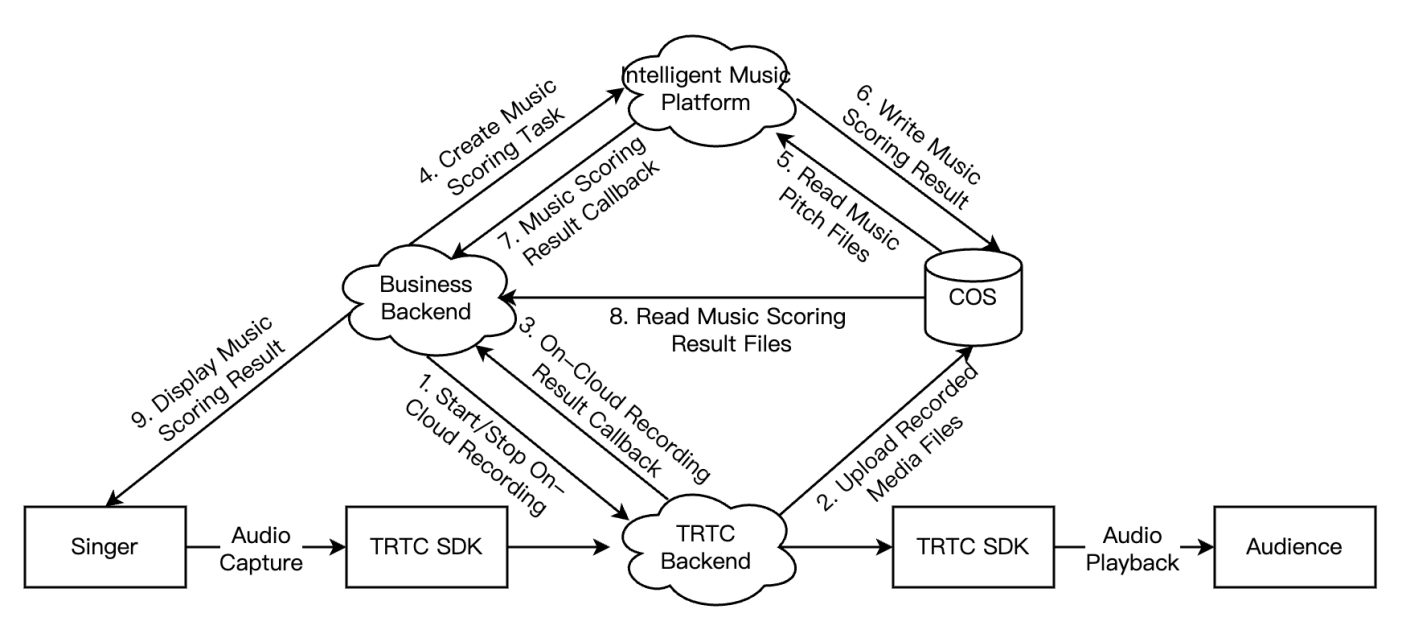
Lead Singer	Chorus	Audience
• NTP Time Synchronization • Enable Black Frame Insertion • Send SEI Message • Local Lyric Synchronization. • Update Lyrics Control	• NTP Time Synchronization • Local Lyric Synchronization. • Update Lyrics Control	• NTP Time Synchronization • Receive SEI messages • Update Lyrics Control

Among them, the lead singer and chorus update the lyric progress locally based on the playback progress of the synchronized accompaniment; the audience needs to receive SEI messages sent from the lead vocalist containing the latest lyric progress to update the local lyric progress. The overall process of the lyrics synchronization solution is shown in the following diagram.



Music Scoring Integration Solution

The accompaniment scoring feature is an indispensable feature in the Online Karaoke scenario. You need to access the standardized audio file and MIDI pitch file of the music resources in advance, and then it is recommended to use the music scoring feature of the Intelligent Music Platform to score the singer's voice from TRTC on-cloud recording. The overall process of the Music Scoring Integration Solution is shown in the following diagram.



1. The business backend starts the on-cloud recording task when the singer begins to sing, and stops the on-cloud recording task when the singer finishes singing.
2. The TRTC backend will upload the recorded singing clip media file to the COS bucket specified when initiating the recording task.
3. After the recording file is uploaded, the TRTC backend will callback the on-cloud recording results to the business backend.
4. The business backend uses the Intelligent Music Platform's music scoring feature to create a music scoring task.
5. The Intelligent Music Platform reads the singing clip and the standard pitch file from the COS bucket for scoring.
6. The Intelligent Music Platform will write the JSON file containing the scoring results into the specified path in COS.
7. After the music scoring is completed, the Intelligent Music Platform will call back the music scoring results to the business backend.
8. The business backend reads the music scoring results JSON file from COS according to the callback path.
9. The business backend analyzes the music scoring results and displays the scoring results on the singer's

App.

> **Note:**
> The input file format for the Intelligent Music Platform's music scoring should use MP3 or WAV. If the on-cloud recording file format is HLS or AAC, audio transcoding is required.

Best Practices for Audio Tuning Strategies

In the entire Karaoke scenario, the audio quality is mainly affected by parameters such as sampling rate, number of channels, bitrate, and 3A. According to different room scenarios, we recommend various audio parameters and volume mixing schemes, as well as commonly used vocal and accompaniment synchronization alignment solutions.

1. **Best Parameter Configuration for Different Scenarios**

Room Scenario	Entry Mode	Audio Quality	Volume Type	Hidden Interface
Solo Singing	• Video or CDN Push Requirements: LIVE • Audio-only or Pure RTC Requirements: VOICE_CHATROOM	MUSIC	TRTCSystemVolumeTypeMedia	enableBlackStream
Real-Time Chorus	• Video or CDN Push Requirements: LIVE • Audio-only or Pure RTC Requirements: VOICE_CHATROOM	MUSIC	TRTCSystemVolumeTypeMedia	• enableBlackStream • enableChorus • setLowLatencyModeEnabled
Chat and Listen to Music	VOICE_CHATROOM	DEFAULT	TRTCSystemVolumeTypeAuto	No

2. **Best Volume Ratio for Different Scenarios**

The TRTC SDK has initial default values for voice collection and music playback. If in the default situation, there is suppression of voice by accompaniment in the live streaming room, leading to the voice being masked by music, you can adjust the voice and music volume ratio according to the recommended values in the table below.

Room Scenario	Recommended Configuration for Voice/Music/Sound Effects
Solo Singing	• Voice Capture Volume: 60 • Music Playback Volume: 50 • Enable Reverb Effect: Yes
Real-Time Chorus	
Chat and Listen to Music	• Vocal Capture Volume: 100 • Music Playback Volume: 30 • Enable Reverb Effect: No

3. **Voice and Accompaniment Synchronization Alignment**

Due to the JitterBuffer for local vocal capture, the JitterBuffer for song playback mixing, and the GAP that exists from when the human ear receives the accompaniment to when singing begins if the singer sings entirely in sync with the lyrics and accompaniment, the remote audience may perceive a certain latency and misalignment between the vocal, accompaniment, and lyrics. This issue can be improved through the following two methods.

- **Enable Chorus Mode**

Android

```
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("api", "enableChorus");
    JSONObject params = new JSONObject();
    params.put("enable", true);
    params.put("audioSource", 0);
    jsonObject.put("params", params);
    mTRTCCloud.callExperimentalAPI(String.format(Locale.ENGLISH,
jsonObject.toString()));
} catch (JSONException e) {
    e.printStackTrace();
}
```

iOS

```
NSDictionary *jsonDic = @{
                                @"api": @"enableChorus",
                                @"params": @{
                                            @"enable": @(YES),
                                            @"audioSource": @(0)
                                        }
                            };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[trtcCloud callExperimentalAPI:jsonString];
```

Note:
audioSource: 0 (Vocal), audioSource: 1 (Accompaniment); If using single instance streaming, set audioSource to 0 for all streams.

- Enable Low Latency Mode

Android

```
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("api", "setLowLatencyModeEnabled");
    JSONObject params = new JSONObject();
    params.put("enable", true);
    jsonObject.put("params", params);
    mTRTCCloud.callExperimentalAPI(String.format(Locale.ENGLISH,
jsonObject.toString()));
} catch (JSONException e) {
    e.printStackTrace();
}
```

iOS

```
NSDictionary *jsonDic = @{
                            @"api": @"setLowLatencyModeEnabled",
                            @"params": @{
                                        @"enable": @(1)
                                      }
                        };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[trtcCloud callExperimentalAPI:jsonString];
```

Scenario Gameplay

Solo Singing

After becoming a speaker, the audience can select songs and wait in line. After the song starts to play, they can sing solo. This game mode is relatively simple and can be achieved using TRTC single-instance mixing and streaming.

Real-Time Chorus

After becoming a speaker, the audience sings a song with the lead singer at the same time. This game mode is relatively complex. The lead singer side needs to use TRTC dual-instance streaming, and all ends also need to pay attention to accompaniment synchronization and lyric synchronization.

Mass Singing Competition

Users can choose song rooms of different categories according to their preferences. The room will randomly play music clips, and users in the room can grab the microphone at any time to sing the music clips.

Segmental Singing

The same song is divided into segments and assigned to different speakers. After the lead singer sings a segment, other speakers sing the assigned music clips respectively.

Cross-Room Singing Competition

Anchors from different rooms sing, and the audience in their respective rooms helps their anchors. In addition to the Karaoke scenario, this game mode also involves TRTC cross-room competition, and it is necessary to pay attention to the subscription logic of audio and video streams in different rooms.

Supporting Products for the Solution

System Level	Product Name	Application Scenarios
Access Layer	Tencent Real-Time Communication (TRTC)	Provides a low-latency, high-quality real-time interactive live streaming solution for multiple people's audio, which is the basic foundation for online Karaoke scenarios.
Access Layer	Instant Messaging (IM)	Provides room management and seat management capabilities based on group features, enables the sending and receiving of rich media messages such as live streaming room-wide messaging, public screen messages, as well as custom signaling and other communication needs.
Access Layer	Intelligent Music Platform	Based on the self-developed music understanding technology of Tencent Media Lab, it helps users to deeply understand, analyze and create music, and provides capabilities such as lyric recognition, intelligent composition, song recognition, and music scoring.

Quick Integration Guide
Android

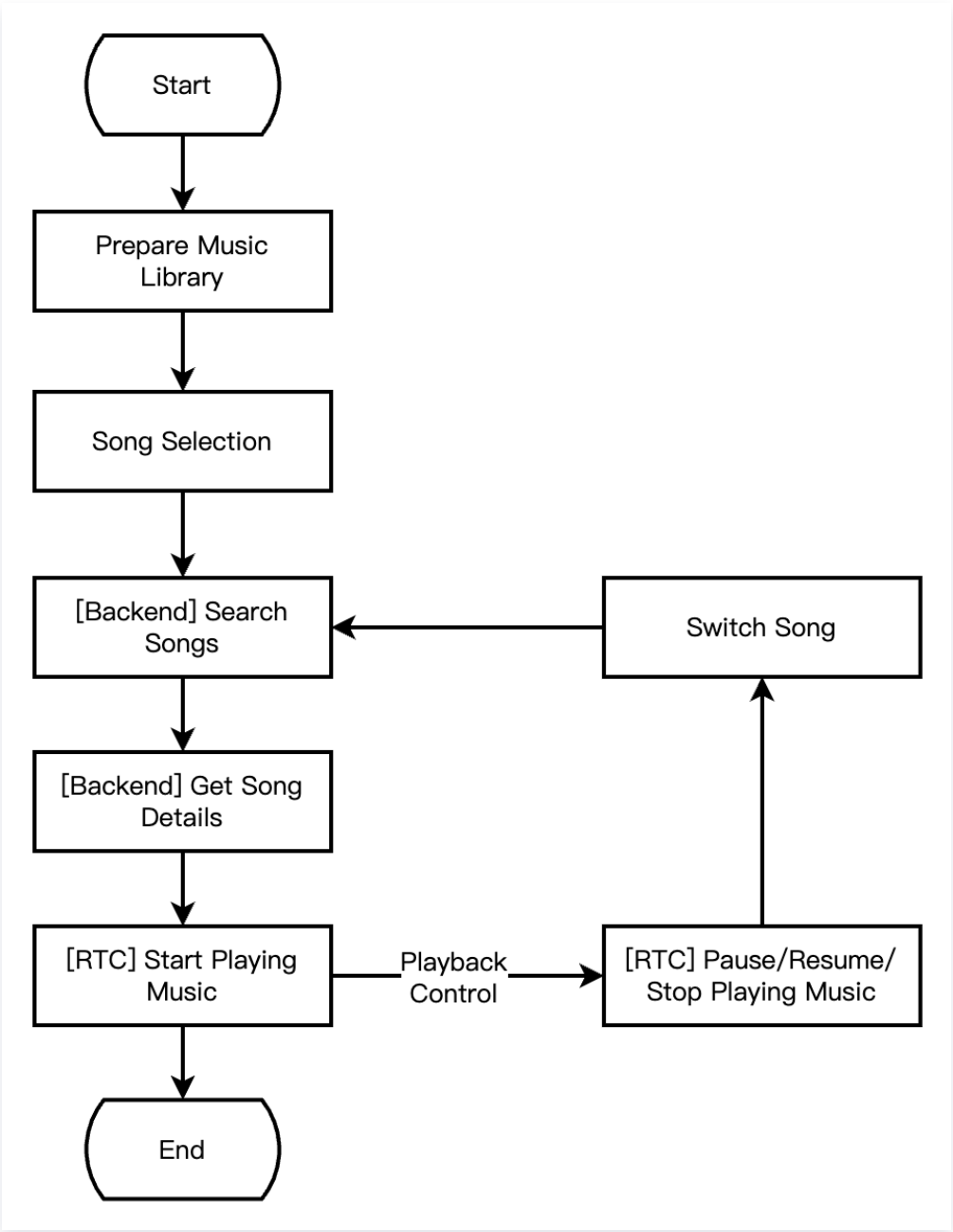
최종 업데이트 날짜:: 2024-07-18 14:26:14

Business Process

This section summarizes some common business processes in online karaoke, helping you better understand the implementation process of the entire scenario.

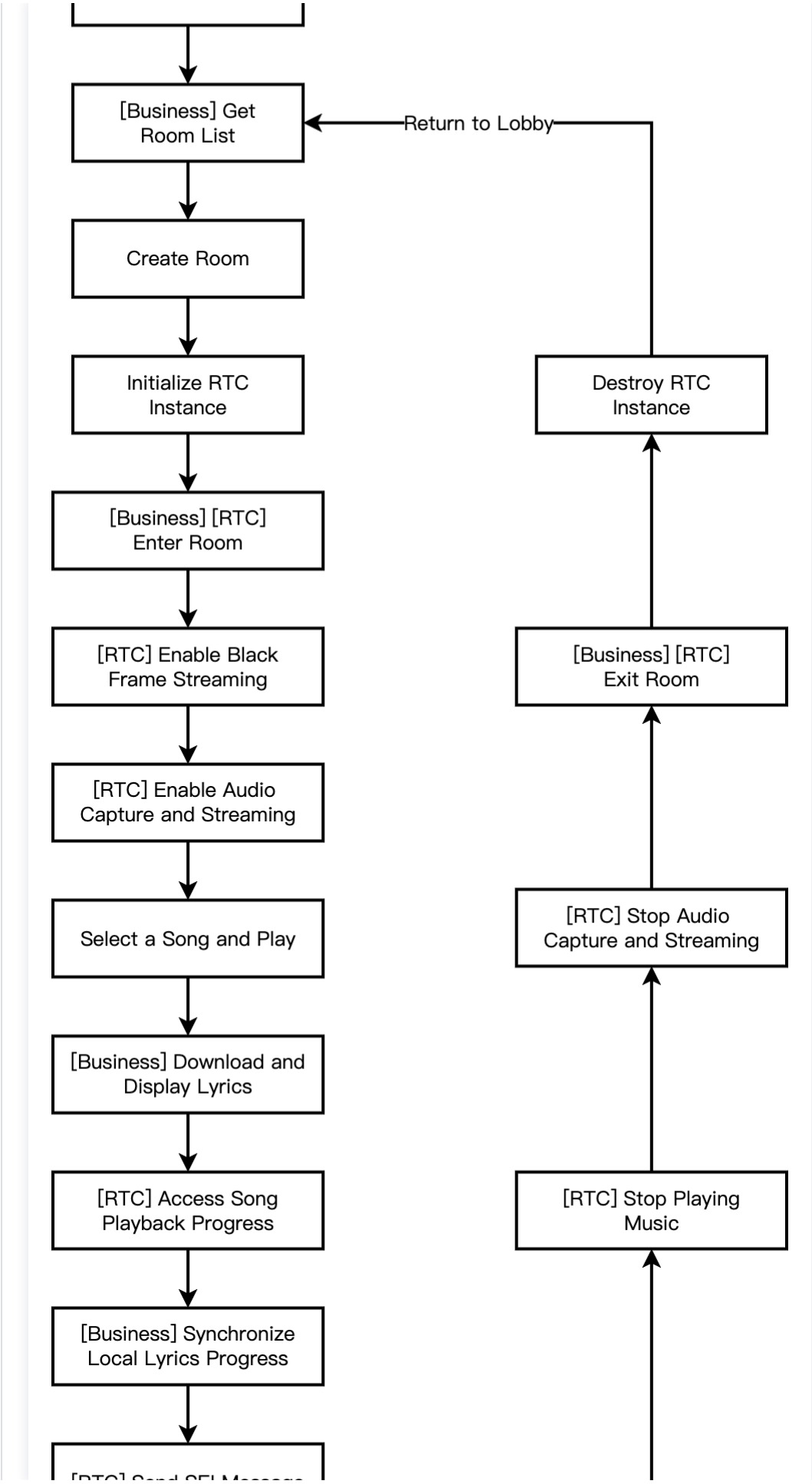
Song request process

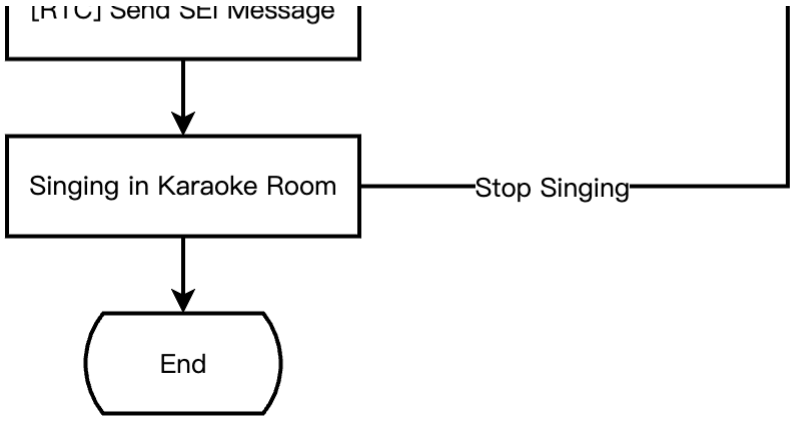
The following figure shows the process of requesting songs from a music repository on the business side and playing them using the TRTC SDK.



Solo singing process

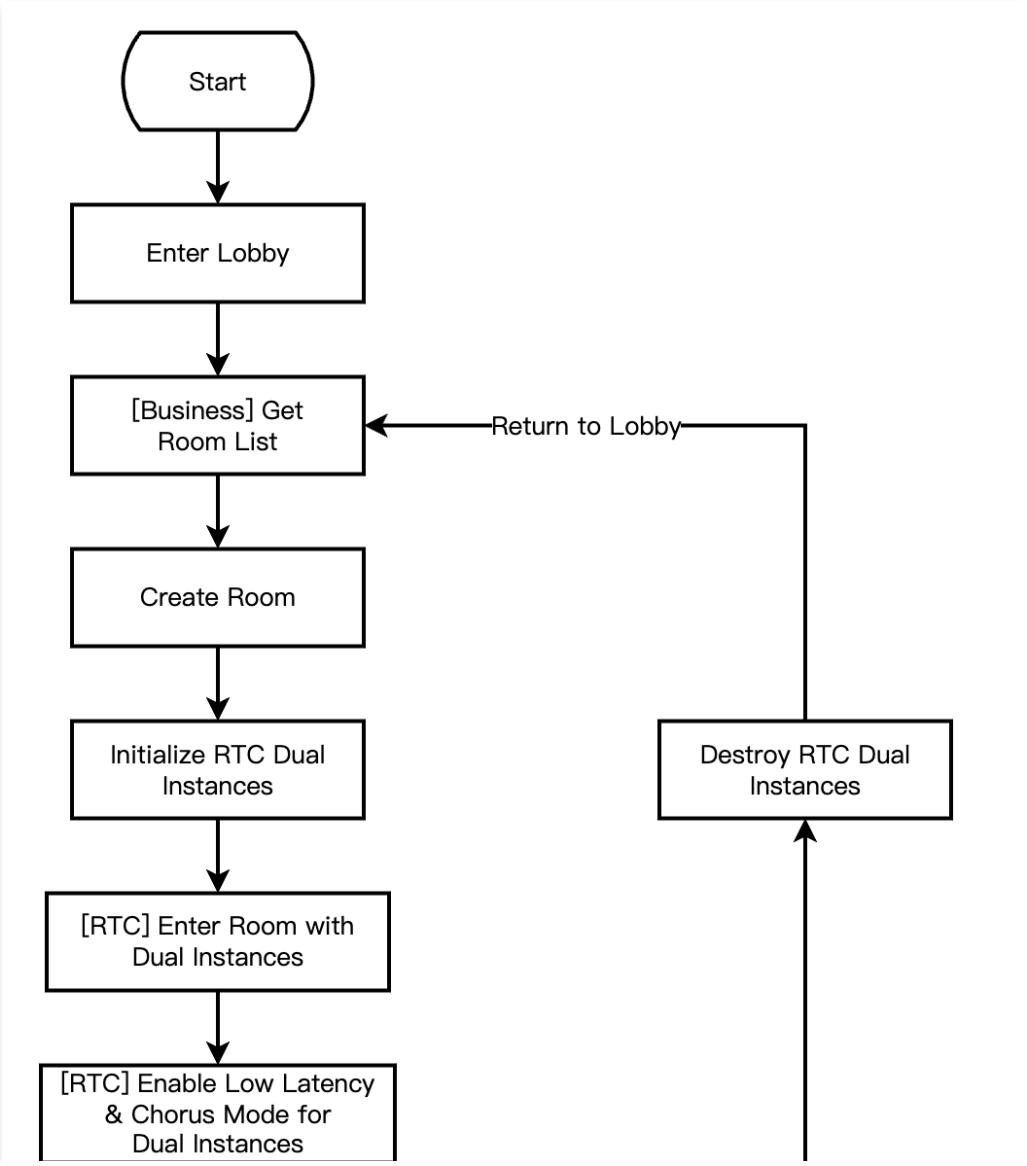
The following figure shows the process of a solo singing turn-taking game, that is, the performer enters a room to perform, stops performing, and exits the room.

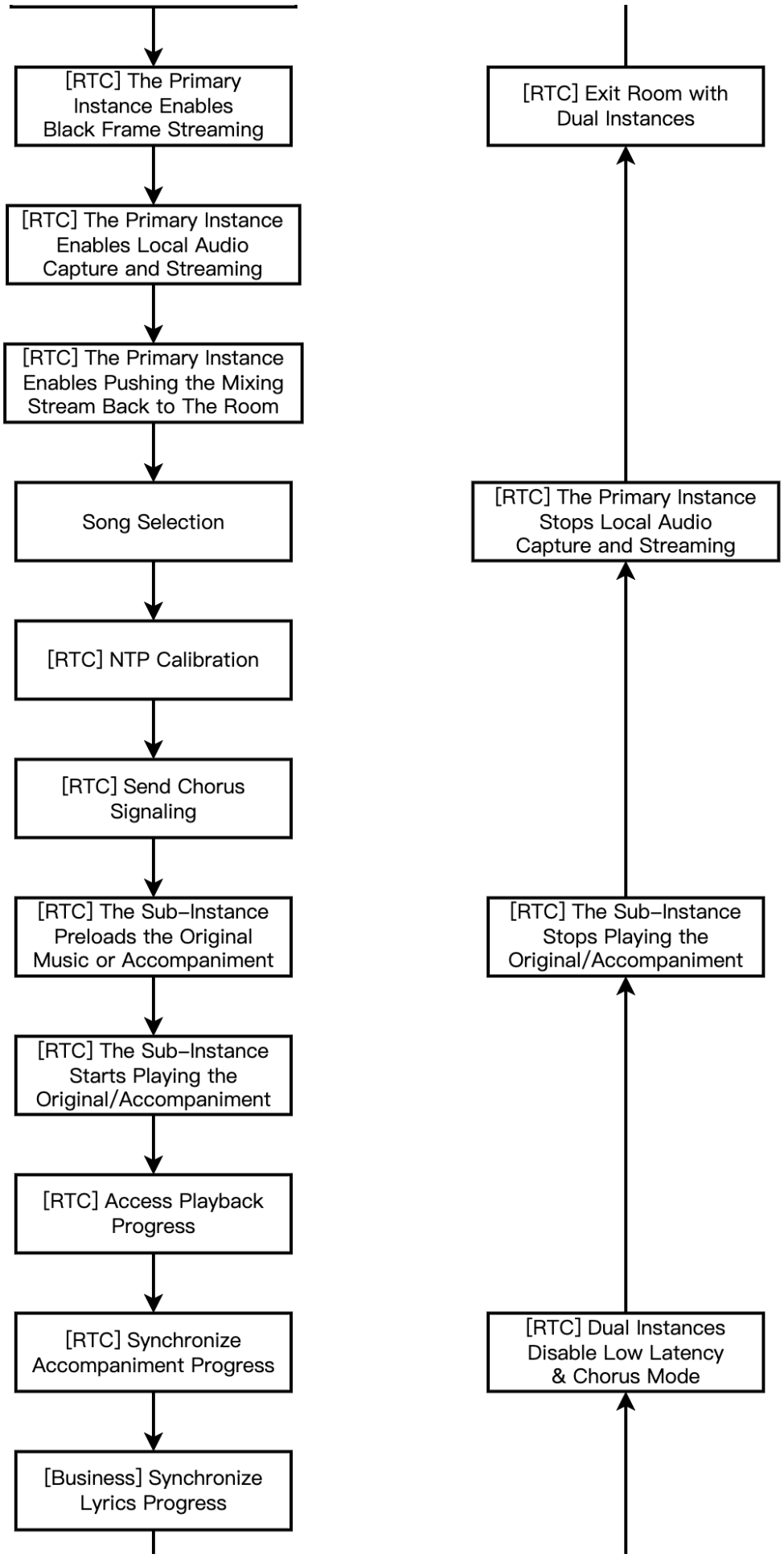


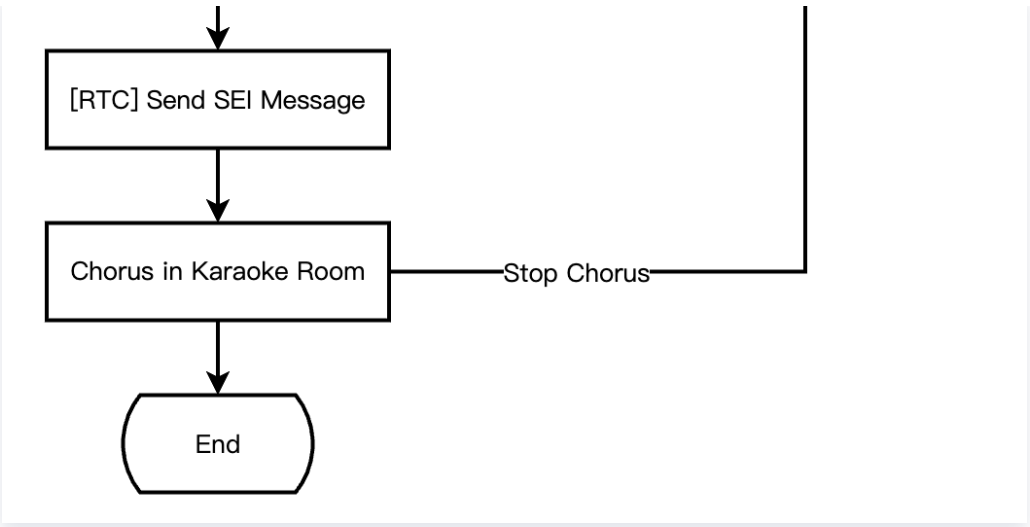


Lead singer process

The following figure shows the process of a real-time chorus game, that is, the lead singer initiates a chorus, stops the chorus, and exits the room.

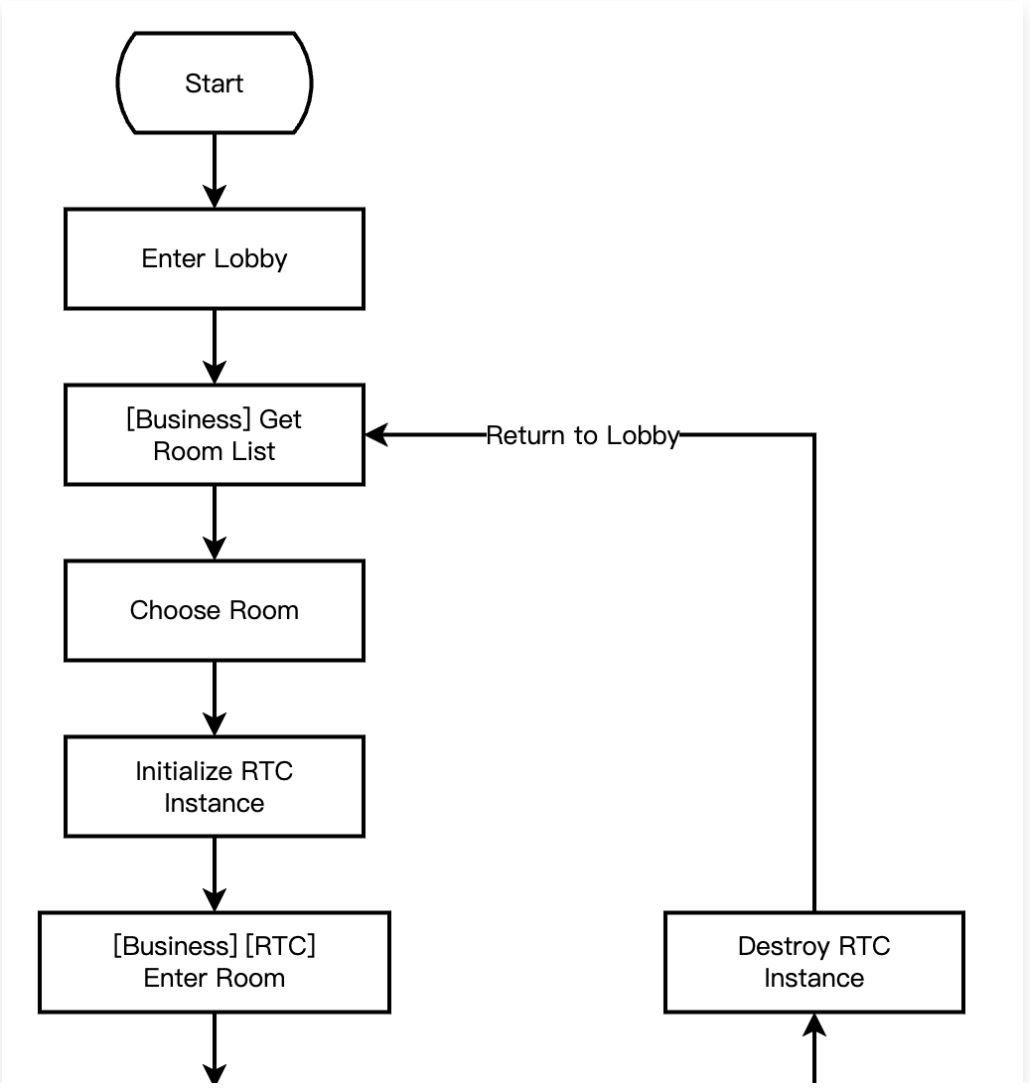


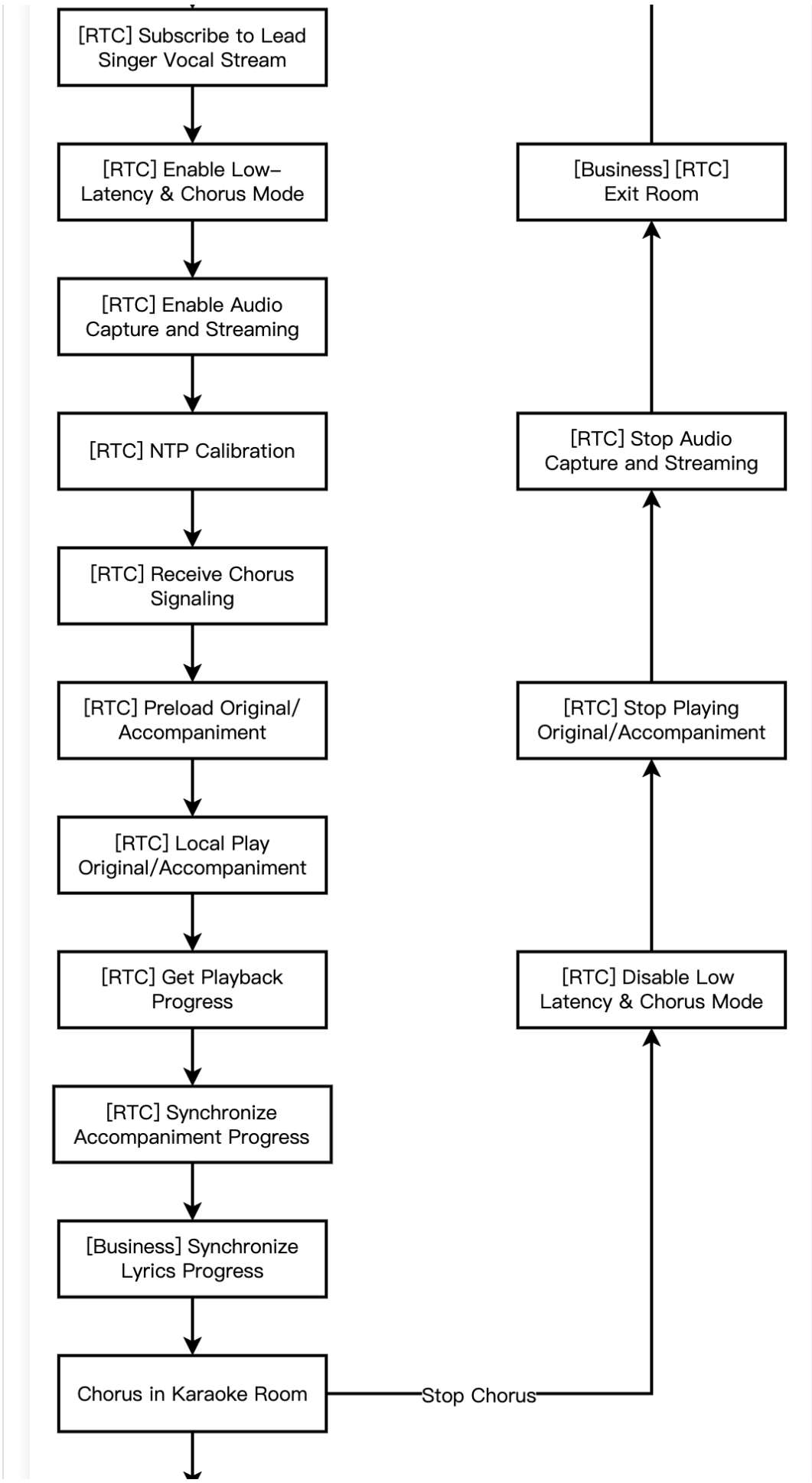


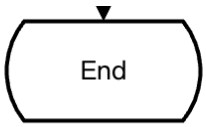


Chorus process

The following figure shows the process of a real-time chorus game, that is, the chorus members join the chorus, stop the chorus, and exit the room.

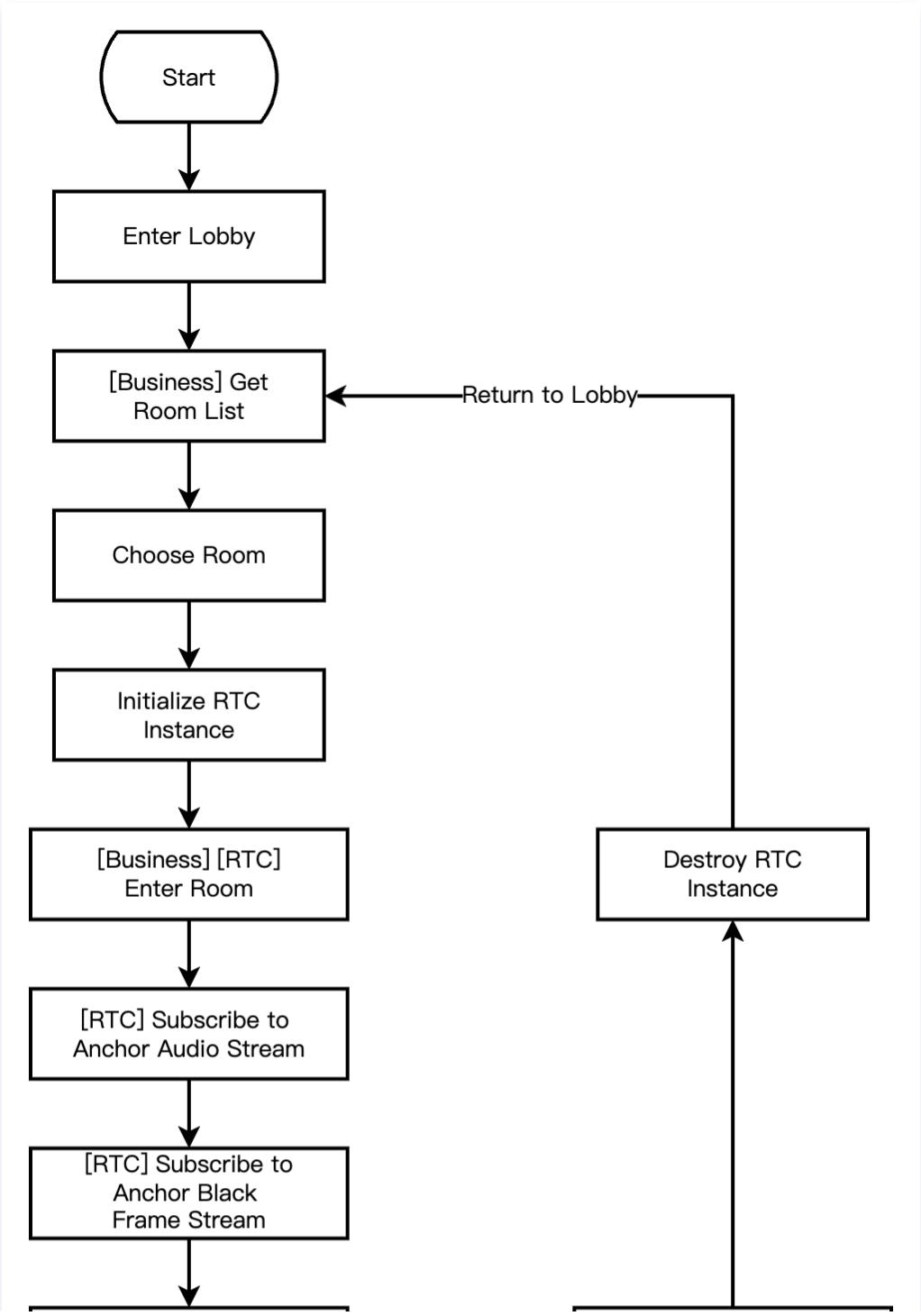


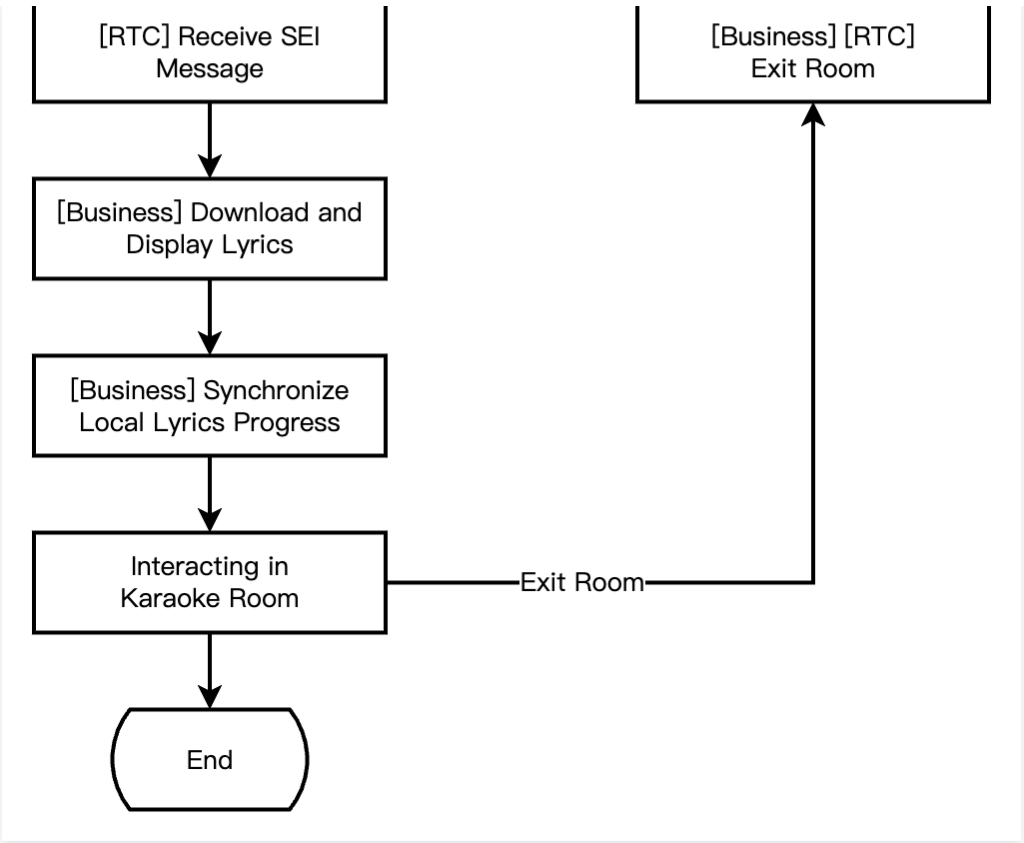




Audience process

The following figure shows the process of an online karaoke scenario, that is, the audience enters the room to listen to songs and synchronizes lyrics.





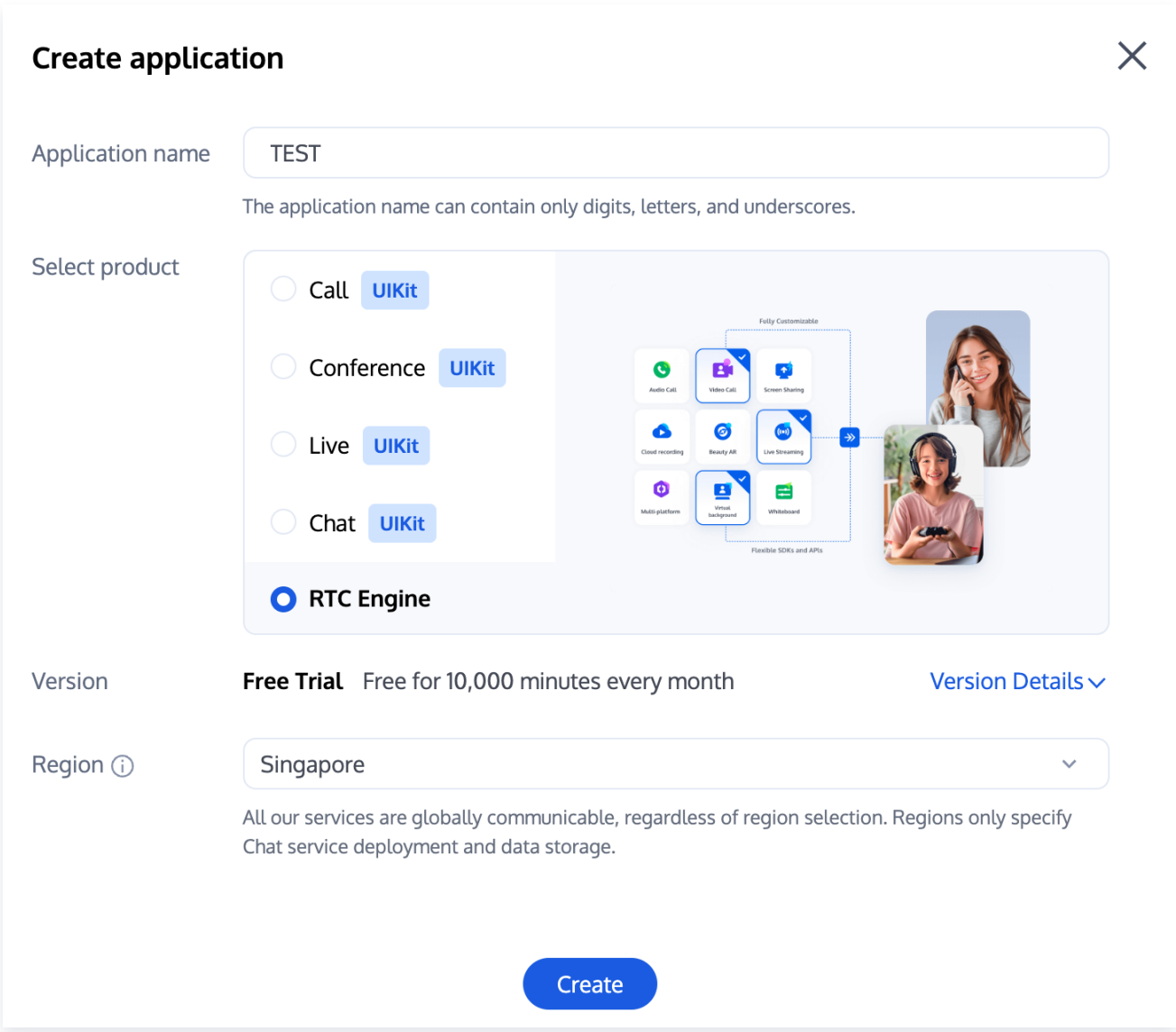
Integration Preparation

Step 1. Activating the service.

The online karaoke scenarios usually require two paid PaaS services from Tencent Cloud: Tencent Real-Time Communication (TRTC) and Intelligent Music Solution for construction. TRTC is responsible for providing real-time audio and video interaction capabilities. Intelligent Music Solution is responsible for providing lyric recognition, smart composition, music recognition, and music scoring capabilities.

Activate TRTC service.

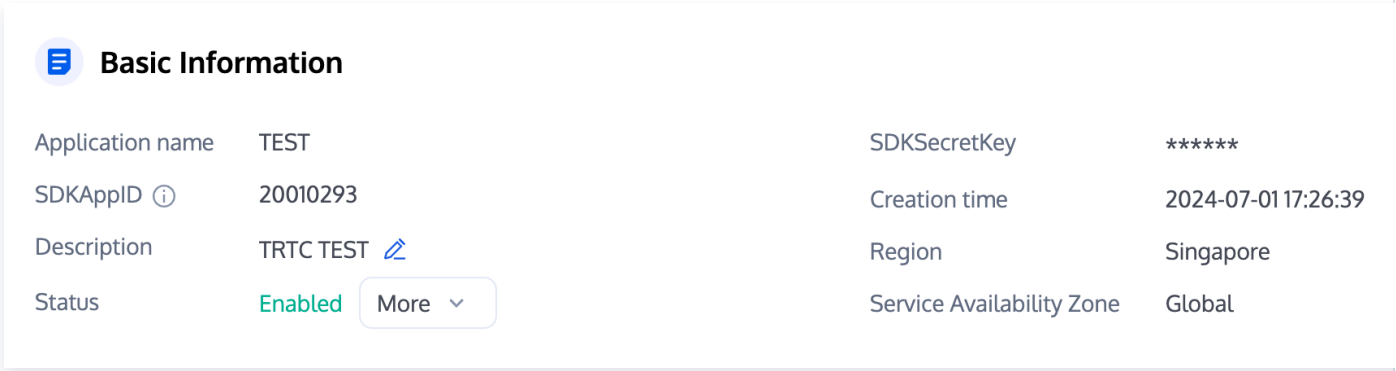
1. First, you need to log in to the Tencent Real-Time Communication (TRTC) console to create an application. You can choose to upgrade the TRTC application version according to your needs. For example, the professional edition unlocks more value-added feature services.



Note:

- It is recommended to create two applications for testing and production environments, respectively. Each Tencent Cloud account (UIN) is given 10,000 minutes of free duration every month for one year.
- TRTC offers monthly subscription plans including the experience edition (default), basic edition, and professional edition. Different value-added feature services can be unlocked. For details, see Version Features and Monthly Subscription Plan Instructions.

2. After an application is created, you can see the basic information of the application in the Application Management – Application Overview section. It is important to keep the **SDKAppID** and **SDKSecretKey** safe for later use and to avoid key leakage that could lead to traffic theft.



Activate the Intelligent Music service.

Preparation

1. Go to the Purchase Page to activate the music service, and choose the appropriate features such as music scoring to activate.
2. Create an AK/SK Key Pair in CAM (namely, a programmable access user that does not require log-in or any user permissions).
3. Create a COS Bucket, and in the COS Bucket Management interface, authorize the read and write permissions of the COS Bucket to the created programmable access user.
4. Prepare the parameters.

- operateUin: Tencent Cloud sub-user's account ID.
- cosConfig: COS related parameters.
  - secretId: Bucket's secretId.
  - secretKey: Bucket's secretKey.
  - bucket: Bucket's name.
  - region: Bucket's region, for example, ap-guangzhou.

Activation and registration.

After the preparation is completed, proceed with registration activation by initiating a request, with an estimated wait time of about 2 minutes.

Initiate request.

```
curl -X POST \
  http://service-mqk0mc83-
1257411467.bj.apigw.tencentcs.com/release/register \
  -H 'Content-Type: application/json' \
  -H 'Cache-control: no-cache' \
  -d '{
    "requestId": "test-regisiter-service",
    "action": "Register",
    "registerRequest": {
      "operateUin": <operateUin>,
      "userName": <customedName>,
      "cosConfig": {
        "secretId": <CosConfig.secretId>,
        "secretKey": <CosConfig.secretKey>,
        "bucket": <CosConfig.bucket>,
        "region": <CosConfig.region>
      }
    }
  }'
```

Request result:

```
{
    "requestId": "test-regisiter-service",
    "registerInfo": {
        "tmpContentId": <tmpContentId>,
        "tmpSecretId": <tmpSecretId>,
        "tmpSecretKey": <tmpSecretKey>,
        "apiGateSecretId": <apiGateSecretId>,
        "apiGateSecretKey": <apiGateSecretKey>,
        "demoCosPath": "UIN_demo/run_musicBeat.py",
        "usageDescription": "Download the python version demo file
[UIN_demo/run_musicBeat.py] from the COS bucket [CosConfig.bucket],
replace the input file in the demo, and then execute python
run_musicBeat.py",
```

```
        "message": "Registration successful, and thank you for
registering.",
        "createdAt": <createdAt>,
        "updatedAt": <updatedAt>
    }
}
```

Run verification.

After the above activation and registration service are completed, a python version executable demo example based on music beat recognition capability will be generated in the `demoCosPath` directory. Execute the command `python run_musicBeat.py` in a networked environment for verification.

> **Note:**
> For more detailed intelligent music solution integration instructions, see Integration Guide.

Step 2: Importing SDK.

The TRTC SDK has been released to the **mavenCentral** repository, and you can configure Gradle to download and update automatically.

1. Add the dependency for the appropriate version of the SDK in dependencies.

```
dependencies {
    // TRTC Lite SDK. It includes TRTC and live streaming playback
features and is compact in size.
    implementation 'com.tencent.liteav:LiteAVSDK_TRTC:latest.release'

    // TRTC Professional SDK. It also includes live streaming, short
video, video on demand, and other features, and is slightly larger in
size.
    // implementation
'com.tencent.liteav:LiteAVSDK_Professional:latest.release'
}
```

> **Note:**
> Besides the recommended automatic loading method, you can also choose to download the SDK and manually import it. For details, see Manually Integrating the TRTC SDK.

2. Specify the CPU architecture used by the app in defaultConfig.

```
defaultConfig {
    ndk {
        abiFilters "armeabi-v7a", "arm64-v8a"
    }
}
```

Note:
The TRTC SDK supports architectures including armeabi, armeabi-v7a and arm64-v8a. Additionally, it supports architectures for simulators including x86 and x86_64.

Step 3: Project configuration.

1. Configure permissions.

To configure app permissions in AndroidManifest.xml, for karaoke scenarios, the TRTC SDK requires the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"
/>
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Note:

- The TRTC SDK does not have built-in permission request logic. You need to declare the corresponding permissions and features yourself. Some permissions (such as storage and recording), also require runtime dynamic requests.
- If the Android project's `targetSdkVersion` is 31 or higher, or if the target device runs Android 12 or a newer version, the official requirement is to dynamically request `android.permission.BLUETOOTH_CONNECT` permission in the code to use the Bluetooth feature properly. For more information, see Bluetooth Permissions.

2. Obfuscation configuration.

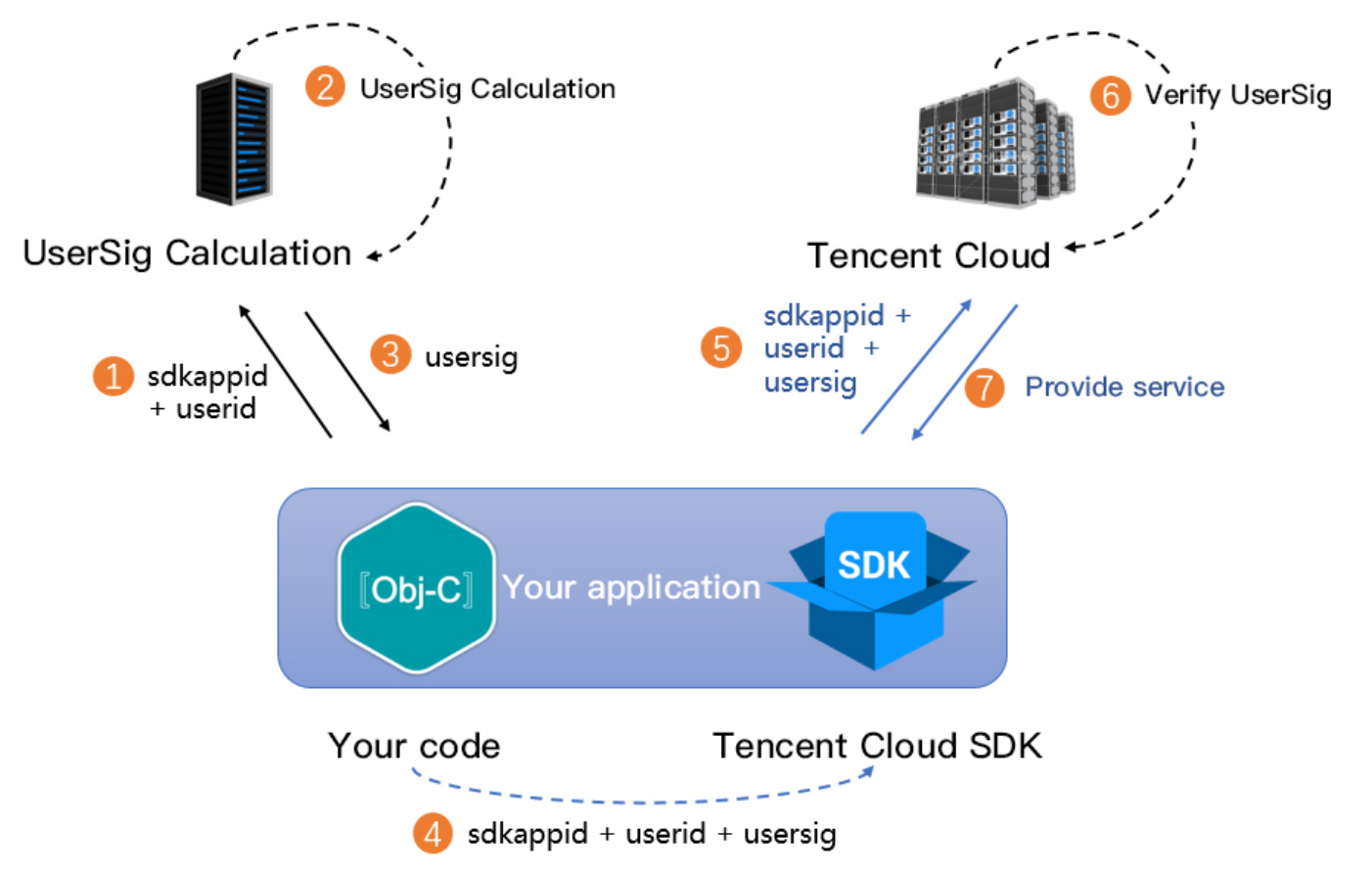
Since we use Java's reflection features inside the SDK, you need to add relevant SDK classes to the non-obfuscation list in the proguard-rules.pro file:

```
-keep class com.tencent.** { *; }
```

Step 4: Authentication and authorization.

UserSig is a security protection signature designed by Tencent Cloud to prevent malicious attackers from misappropriating your cloud service usage rights. TRTC validates this authentication credential when it enters the room.
Debugging Stage: UserSig can be generated through two methods for debugging and testing purposes only: client sample code and console access.
Formal Operation Stage: It is recommended to use a higher security level server computation for generating UserSig. This is to prevent key leakage due to client reverse engineering.
The specific implementation process is as follows:

1. Before calling the SDK's initialization function, your app must first request UserSig from your server.
2. Your server computes the UserSig based on the SDKAppID and UserID.
3. The server returns the computed UserSig to your app.
4. Your app passes the obtained UserSig into the SDK through a specific API.
5. The SDK submits the SDKAppID + UserID + UserSig to Tencent Cloud CVM for verification.
6. Tencent Cloud verifies the UserSig and confirms its validity.
7. After the verification is passed, real-time audio and video services will be provided to the TRTC SDK.



> **Note:**
>
> - The local computation method of UserSig during the debugging stage is not recommended for application in an online environment. It is prone to reverse engineering, leading to key leakage.
> - We provide server computation source code for UserSig in multiple programming languages (Java/GO/PHP/Nodejs/Python/C#/C++). For details, see Server Computation of UserSig.

Step 5: Initializing the SDK.

```
// Create TRTC SDK instance (Single Instance Pattern).
TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(context);
// Set event listeners.
mTRTCCloud.addListener(trtcSdkListener);
```

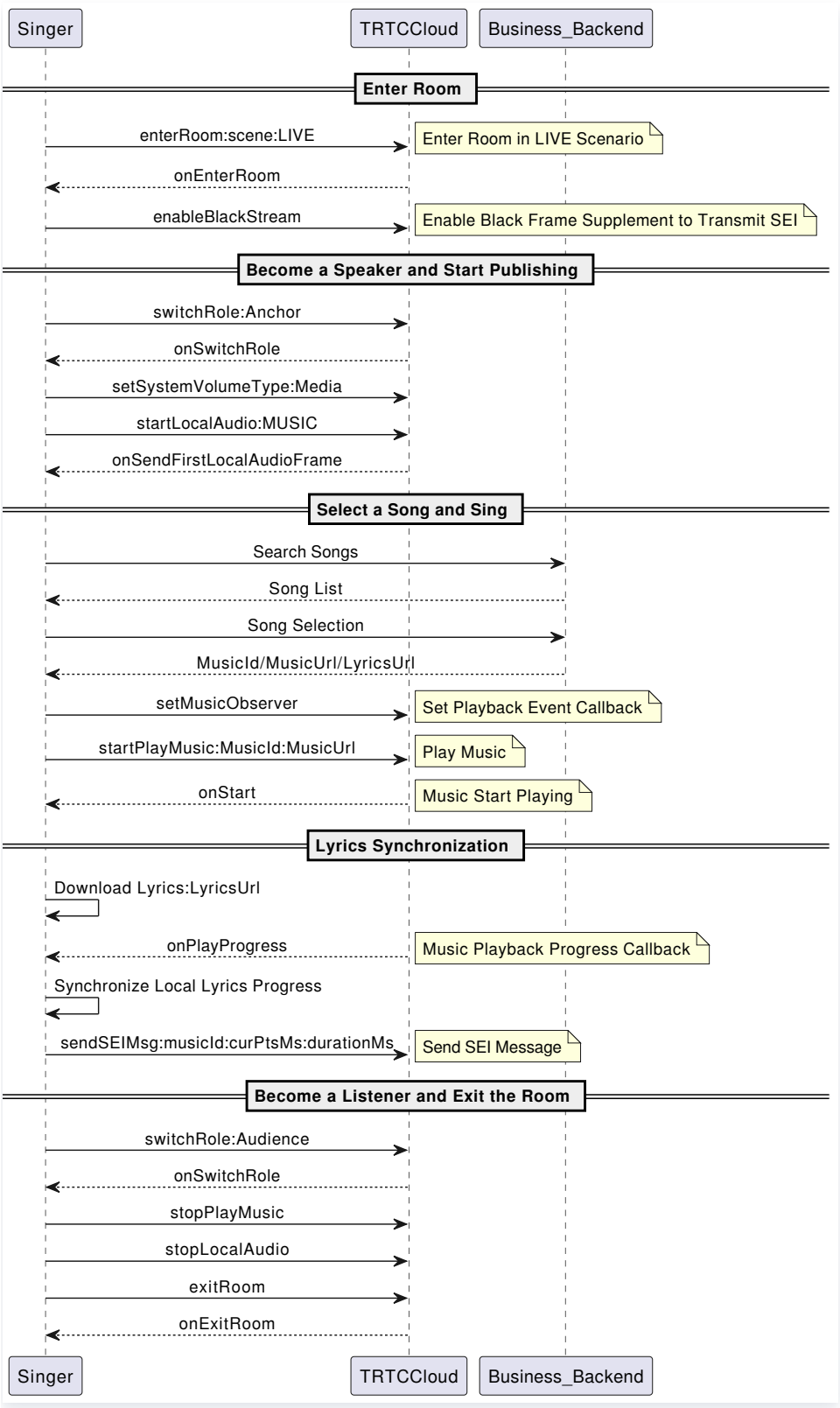
```
// Notifications from various SDK events (e.g., error codes, warning
codes, audio and video status parameters, etc.).
private TRTCCloudListener trtcSdkListener = new TRTCCloudListener() {
    @Override
    public void onError(int errCode, String errMsg, Bundle extraInfo) {
        Log.d(TAG, errCode + errMsg);
    }

    @Override
    public void onWarning(int warningCode, String warningMsg, Bundle
extraInfo) {
        Log.d(TAG, warningCode + warningMsg);
    }
};

// Remove event listener.
mTRTCCloud.removeListener(trtcSdkListener);
// Terminate TRTC SDK instance (Singleton Pattern).
TRTCCloud.destroySharedInstance();
```

Note:
It is recommended to listen to SDK event notifications. Perform log printing and handling for some common errors. For details, see Error Code Table.

Scenario 1: Solo singing turn-taking

Perspective 1: Performer actions

Sequence diagram



1. Enter the room.

```
public void enterRoom(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // Take the room ID string as an example.
```

```
    params.strRoomId = roomId;
    params.userId = userId;
    // UserSig obtained from the business backend.
    params.userSig = getUserSig(userId);
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
    // It is recommended to enter the room as an audience role.
    params.role = TRTCCloudDef.TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}
```

Note:

To better transmit SEI messages for lyrics synchronization, it is recommended to choose `TRTC_APP_SCENE_LIVE` for room-entry scenarios.

```
// Event callback for the result of entering the room.
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        Log.d(TAG, "Enter room succeed");
        // Enable the experimental API for black frame insertion.
        mTRTCCloud.callExperimentalAPI("
{\"api\":\"enableBlackStream\",\"params\": {\"enable\":true}}");
    } else {
        // result indicates the error code when you fail to enter the
room.
        Log.d(TAG, "Enter room failed");
    }
}
```

Note:

Under the pure audio mode, the performer needs to enable the insertion of black frames to carry SEI messages. This API should be called after successfully entering the room.

2. Go live on streams.

```
// Switched to the anchor role.
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);

// Event callback for switching the role.
@Override
public void onSwitchRole(int errCode, String errMsg) {
    if (errCode == TXLiteAVCode.ERR_NULL) {
        // Set media volume type.

mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);
        // Upstream local audio streams and set audio quality.

mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
    }
}
```

Note:
In karaoke scenarios, it is recommended to set the full-range media volume and music quality to achieve a high-fidelity listening experience.

3. Song selection and performance.

- Search for songs, and obtain music resources.

  Search for songs and acquire music resources through the business backend. Obtain identifiers such as the MusicId, the song's URL (MusicUrl), and the lyrics URL (LyricsUrl).
  It is recommended that the business side select an appropriate music repository production to provide licensed music resources.
- Play accompaniment and start singing.

```
// Obtain audio effects management.
TXAudioEffectManager mTXAudioEffectManager =
mTRTCCloud.getAudioEffectManager();

// originMusicId: Custom identifier for the original vocal music.
originMusicUrl: URL of the original vocal music resource.
TXAudioEffectManager.AudioMusicParam originMusicParam = new
TXAudioEffectManager.AudioMusicParam(originMusicId, originMusicUrl);
```

```
// Whether to publish the original vocal music to remote (otherwise play
locally only).
originMusicParam.publish = true;

// accompMusicId: Custom identifier for the accompaniment music.
accompMusicUrl: URL of the accompaniment music resource.
TXAudioEffectManager.AudioMusicParam accompMusicParam = new
TXAudioEffectManager.AudioMusicParam(accompMusicId, accompMusicUrl);
// Whether to publish the accompaniment to remote (otherwise play
locally only).
accompMusicParam.publish = true;

// Start playing the original vocal music.
mTXAudioEffectManager.startPlayMusic(originMusicParam);
// Start playing the accompaniment music.
mTXAudioEffectManager.startPlayMusic(accompMusicParam);

// Switch to the original vocal music.
mTXAudioEffectManager.setMusicPlayoutVolume(originMusicId, 100);
mTXAudioEffectManager.setMusicPlayoutVolume(accompMusicId, 0);
mTXAudioEffectManager.setMusicPublishVolume(originMusicId, 100);
mTXAudioEffectManager.setMusicPublishVolume(accompMusicId, 0);

// Switch to the accompaniment music.
mTXAudioEffectManager.setMusicPlayoutVolume(originMusicId, 0);
mTXAudioEffectManager.setMusicPlayoutVolume(accompMusicId, 100);
mTXAudioEffectManager.setMusicPublishVolume(originMusicId, 0);
mTXAudioEffectManager.setMusicPublishVolume(accompMusicId, 100);
```

Note:

- In karaoke scenarios, both the original vocal and accompaniment need to be played simultaneously (distinguished by MusicID). The switch between the original vocal and accompaniment is achieved by adjusting the local and remote playback volumes.
- If the music being played has dual audio tracks (including both the original vocal and accompaniment), switching between them can be achieved by specifying the music's playback track using setMusicTrack.

4. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Synchronize local lyrics, and transmit song progress via SEI.

```
mTXAudioEffectManager.setMusicObserver(musicId, new
TXAudioEffectManager.TXMusicPlayObserver() {
    @Override
    public void onStart(int id, int errCode) {
        // Start playing music.
    }

    @Override
    public void onPlayProgress(int id, long curPtsMs, long durationMs) {
        // Determine whether seek is needed based on the latest progress
and the local lyrics progress deviation.
        // Song progress is transmitted by sending an SEI message.
        try {
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("musicId", id);
            jsonObject.put("progress", curPtsMs);
            jsonObject.put("duration", durationMs);
            mTRTCCloud.sendSEIMsg(jsonObject.toString().getBytes(), 1);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onComplete(int id, int errCode) {
        // Music playback completed.
    }
});
```

Note:

- Ensure to set the playback event callback using this API before playing the background music. This allows to be aware of the background music's playback progress.
- The frequency of the SEI messages sent by the performer is determined by the event callback

frequency. Also, the playback progress can be actively synchronized on a schedule through getMusicCurrentPosInMS.

5. Become a listener and exit the room.

```
// Switched to the audience role.
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);

// Event callback for switching the role.
@Override
public void onSwitchRole(int errCode, String errMsg) {
    if (errCode == TXLiteAVCode.ERR_NULL) {
        // Stop playing accompaniment music.
        mTRTCCloud.getAudioEffectManager().stopPlayMusic(musicId);
        // Stop local audio capture and publishing.
        mTRTCCloud.stopLocalAudio();
    }
}

// Exit the room.
mTRTCCloud.exitRoom();

// Exit room event callback.
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "Actively call exitRoom to exit the room.");
    } else if (reason == 1) {
        Log.d(TAG, "Removed from the current room by the server.");
    } else if (reason == 2) {
        Log.d(TAG, "The current room has been dissolved.");
    }
}
```

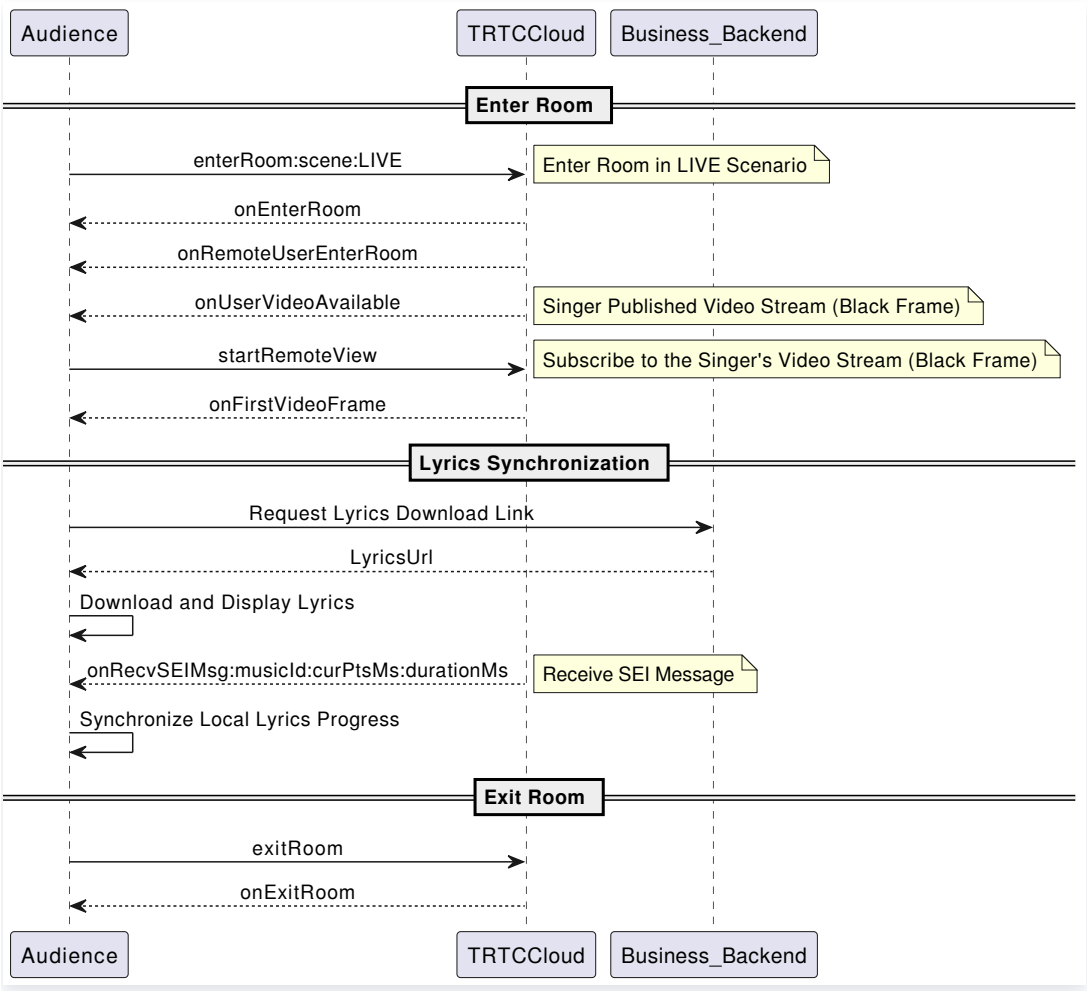
Note:

- After all resources occupied by the SDK are released, the SDK will throw the `onExitRoom` callback notification to inform you.
- If you want to call `enterRoom` again or switch to another audio and video SDK, wait for the

> `onExitRoom` callback before proceeding. Otherwise, you may encounter various exceptional issues such as the camera, microphone device being forcibly occupied.

Perspective 2: Listener actions

Sequence diagram



1. Enter the room.

```
public void enterRoom(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // Take the room ID string as an example.
    params.strRoomId = roomId;
    params.userId = userId;
    // UserSig obtained from the business backend.
    params.userSig = getUserSig(userId);
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
```

```
    // It is recommended to enter the room as an audience role.
    params.role = TRTCCloudDef.TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

// Event callback for the result of entering the room.
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result indicates the error code when you fail to enter the
room.
        Log.d(TAG, "Enter room failed");
    }
}
```

Note:

- To better transmit SEI messages for lyrics synchronization, it is recommended to choose `TRTC_APP_SCENE_LIVE` for room-entry scenarios.
- Under the automatic subscription mode (default), audiences automatically subscribe and play the on-mic anchor's audio and video streams upon entering the room.

2. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Listener end lyric synchronization

```
@Override
public void onUserVideoAvailable(String userId, boolean available) {
    if (available) {
        mTRTCCloud.startRemoteView(userId, null);
    } else {
        mTRTCCloud.stopRemoteView(userId);
```

```
    }
}

@Override
public void onRecvSEIMsg(String userId, byte[] data) {
    String result = new String(data);
    try {
        JSONObject jsonObject = new JSONObject(result);
        int musicId = jsonObject.getInt("musicId");
        long progress = jsonObject.getLong("progress");
        long duration = jsonObject.getLong("duration");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    ...
    // TODO: The logic of updating the lyric control.
    // Based on the received latest progress and the local lyrics
progress deviation, determine whether a lyric control seek is necessary.
    ...
}
```

Note:
Listeners need to actively subscribe to the performer's video streams in order to receive the SEI messages carried by black frames.

3. Exit the room.

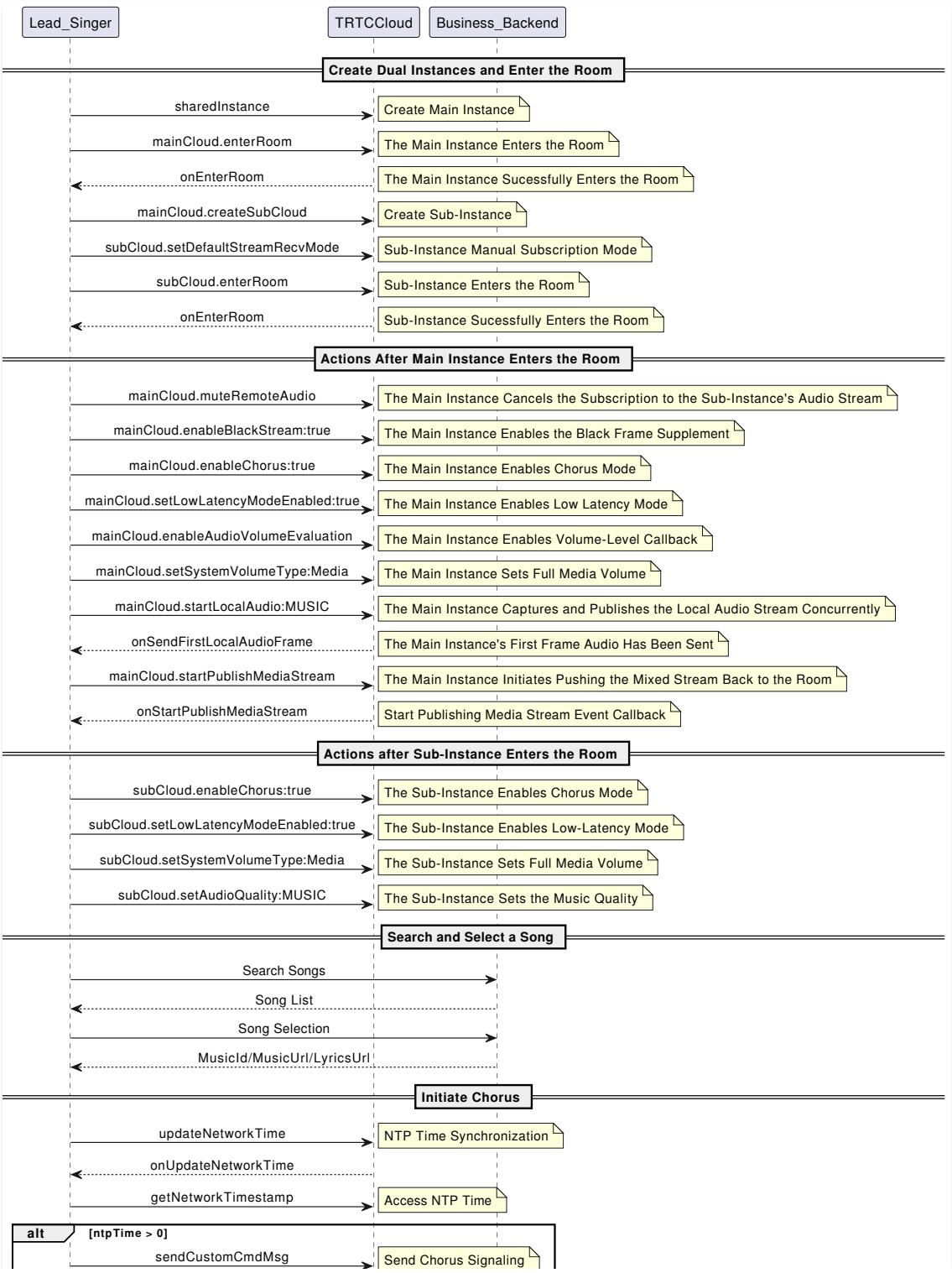
```
// Exit the room.
mTRTCCloud.exitRoom();

// Exit room event callback.
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "Actively call exitRoom to exit the room.");
    } else if (reason == 1) {
        Log.d(TAG, "Removed from the current room by the server.");
    } else if (reason == 2) {
        Log.d(TAG, "The current room has been dissolved.");
```

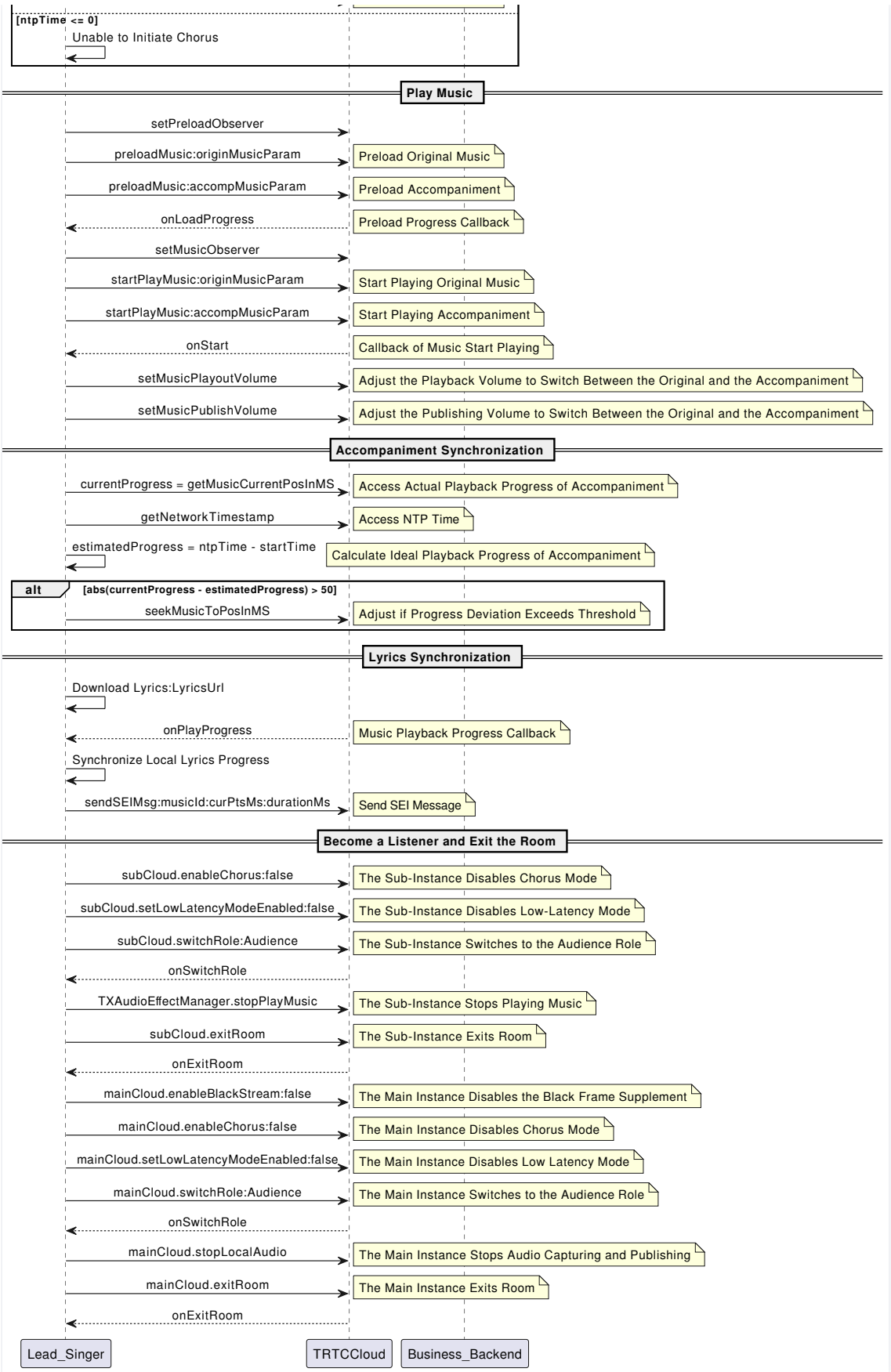
```
        }
    }
```

Scenario 2: Real-time chorus

Perspective 1: Lead singer actions

Sequence diagram





1. Dual instances enter the room.

```
// Create a TRTCCloud primary instance (vocal instance).
```

```
TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(context);
// Create a TRTCCloud sub-instance (music instance).
TRTCCloud subCloud = mTRTCCloud.createSubCloud();

// The primary instance (vocal instance) enters the room.
TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
params.sdkAppId = SDKAppId;
params.userId = UserId;
params.userSig = UserSig;
params.role = TRTCCloudDef.TRTCRoleAnchor;
params.strRoomId = RoomId;
mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);

// The sub-instance enables manual subscription mode. By default it does
not subscribe to remote streams.
subCloud.setDefaultStreamRecvMode(false, false);

// The sub-instance (music instance) enters the room.
TRTCCloudDef.TRTCParams bgmParams = new TRTCCloudDef.TRTCParams();
bgmParams.sdkAppId = SDKAppId;
// The sub-instance username must not duplicate with other users in the
room.
bgmParams.userId = UserId + "_bgm";
bgmParams.userSig = UserSig;
bgmParams.role = TRTCCloudDef.TRTCRoleAnchor;
bgmParams.strRoomId = RoomId;
subCloud.enterRoom(bgmParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
```

Note:

- In a real-time chorus solution, the lead singer end must create primary instance and sub-instance for upstream voice and accompaniment music, respectively.
- Sub-instances do not need to subscribe to other users' audio streams in the room. Therefore, it is recommended to enable manual subscription mode, and it must be activated before entering the room.

2. Set the settings after entering the room.

```
// Event callback for the result of primary instance entering the room.
```

```
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // The primary instance unsubscribe from music streams published
by sub-instances.
        mTRTCCloud.muteRemoteAudio(UserId + "_bgm", true);
        // The primary instance uses the experimental API to enable
black frame insertion.
        mTRTCCloud.callExperimentalAPI(
        "{\"api\":\"enableBlackStream\",\"params\":
{\"enable\":true}}");
        // The primary instance uses the experimental API to enable
chorus mode.
        mTRTCCloud.callExperimentalAPI(
        "{\"api\":\"enableChorus\",\"params\":
{\"enable\":true,\"audioSource\":0}}");
        // The primary instance uses the experimental API to enable low-
latency mode.
        mTRTCCloud.callExperimentalAPI(
        "{\"api\":\"setLowLatencyModeEnabled\",\"params\":
{\"enable\":true}}");
        // The primary instance enables volume level callback.
        mTRTCCloud.enableAudioVolumeEvaluation(300, false);
        // The primary instance sets the global media volume type.

mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);
        // The primary instance captures and publishes local audio, and
sets audio quality.

mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
    } else {
        // result indicates the error code when you fail to enter the
room.
        Log.d(TAG, "Enter room failed");
    }
}

// Event callback for the result of sub-instance entering the room.
@Override
public void onEnterRoom(long result) {
```

```
    if (result > 0) {
        // The sub-instance uses the experimental API to enable chorus
mode.
        subCloud.callExperimentalAPI(
        "{\"api\":\"enableChorus\",\"params\":
{\"enable\":true,\"audioSource\":1}}");
        // The sub-instance uses the experimental API to enable low-
latency mode.
        subCloud.callExperimentalAPI(
        "{\"api\":\"setLowLatencyModeEnabled\",\"params\":
{\"enable\":true}}");
        // The sub-instance sets global media volume type.

subCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);
        // The sub-instance sets audio quality.
        subCloud.setAudioQuality(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
    } else {
        // result indicates the error code when you fail to enter the
room.
        Log.d(TAG, "Enter room failed");
    }
}
```

> **Note:**
> Both the primary instance and sub-instance must use the experimental APIs to enable chorus mode and low-latency mode to optimize the chorus experience. Note the difference in the `audioSource` parameter.

3. Push the mixed stream back to the room.

```
private void startPublishMediaToRoom(String roomId, String userId) {
    // Create TRTCPublishTarget object.
    TRTCCloudDef.TRTCPublishTarget target = new
TRTCCloudDef.TRTCPublishTarget();
    // After mixing, the stream is relayed back to the room.
    target.mode = TRTCCloudDef.TRTC_PublishMixStream_ToRoom;
    target.mixStreamIdentity.strRoomId = roomId;
    // The mixing stream robot's username must not duplicate with other
users in the room.
```

```
    target.mixStreamIdentity.userId = userId + "_robot";

    // Set the encoding parameters of the transcoded audio stream (can
be customized).
    TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new
TRTCCloudDef.TRTCStreamEncoderParam();
    trtcStreamEncoderParam.audioEncodedChannelNum = 2;
    trtcStreamEncoderParam.audioEncodedKbps = 64;
    trtcStreamEncoderParam.audioEncodedCodecType = 2;
    trtcStreamEncoderParam.audioEncodedSampleRate = 48000;

    // Set the encoding parameters of the transcoded video stream (black
frame mixing required).
    trtcStreamEncoderParam.videoEncodedFPS = 15;
    trtcStreamEncoderParam.videoEncodedGOP = 3;
    trtcStreamEncoderParam.videoEncodedKbps = 30;
    trtcStreamEncoderParam.videoEncodedWidth = 64;
    trtcStreamEncoderParam.videoEncodedHeight = 64;

    // Set audio mixing parameters.
    TRTCCloudDef.TRTCStreamMixingConfig trtcStreamMixingConfig = new
TRTCCloudDef.TRTCStreamMixingConfig();
    // By default, leave this field empty. It indicates that all audio
in the room will be mixed.
    trtcStreamMixingConfig.audioMixUserList = null;

    // Configure video mixed-stream template (black frame mixing
required).
    TRTCCloudDef.TRTCVideoLayout videoLayout = new
TRTCCloudDef.TRTCVideoLayout();
    trtcStreamMixingConfig.videoLayoutList.add(videoLayout);

    // Start mixing and pushing back.
    mTRTCCloud.startPublishMediaStream(target, trtcStreamEncoderParam,
trtcStreamMixingConfig);
}
```

Note:

- To maintain alignment between chorus vocals and accompaniment music, it is recommended to

enable pushing the mixed stream back to the room. The on-mic chorus members mutually subscribe to single streams, and off-mic audiences by default only subscribe to mixed streams.

- The mixing stream robot, acting as an independent user, enters the room to pull, mix, and push streams. Its username must not duplicate with other usernames in the room. Otherwise, it may lead to mutual deletion from the room.

4. Search for and request songs.

   Search for songs and acquire music resources through the business backend. Obtain identifiers such as the MusicId, the song's URL (MusicUrl), and the lyrics URL (LyricsUrl).
   It is recommended that the business side select an appropriate music repository production to provide licensed music resources.
5. NTP synchronization.

```
TXLiveBase.setListener(new TXLiveBaseListener() {
    @Override
    public void onUpdateNetworkTime(int errCode, String errMsg) {
        super.onUpdateNetworkTime(errCode, errMsg);
        // errCode 0: Time synchronization successful and deviation
within 30 ms. 1: Time synchronization successful but deviation possibly
above 30 ms. -1: Time synchronization failed.
        if (errCode == 0) {
            // Time synchronization successful and NTP timestamp
obtained.
            long ntpTime = TXLiveBase.getNetworkTimestamp();
        } else {
            // If time synchronization fails, an attempt to
resynchronize can be made.
            TXLiveBase.updateNetworkTime();
        }
    }
});

TXLiveBase.updateNetworkTime();
```

Note:
NTP time synchronization results can reflect the current network quality of the application user. To ensure a good chorus experience, it is recommended not to allow users to initiate chorus if time synchronization fails.

6. Send chorus signaling.

```
Timer mTimer = new Timer();
mTimer.schedule(new TimerTask() {
    @Override
    public void run() {
        try {
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("cmd", "startChorus");
            // Agreed chorus start time: Current NTP time + delayed
playback time (for example, 3 seconds).
            jsonObject.put("startPlayMusicTS",
TXLiveBase.getNetworkTimestamp() + 3000);
            jsonObject.put("musicId", musicId);
            jsonObject.put("musicDuration",
subCloud.getAudioEffectManager().getMusicDurationInMS(originMusicUri));
            mTRTCCloud.sendCustomCmdMsg(1,
jsonObject.toString().getBytes(), false, false);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}, 0, 1000);
```

Note:
The lead singer needs to cyclically broadcast chorus signaling to the room at a fixed time interval (e.g., every 1 second), so that new users who join mid-session can also participate in the chorus.

7. Load and play accompaniment.

```
// Obtain audio effects management.
TXAudioEffectManager mTXAudioEffectManager =
subCloud.getAudioEffectManager();

// originMusicId: Custom identifier for the original vocal music.
originMusicUrl: URL of the original vocal music resource.
TXAudioEffectManager.AudioMusicParam originMusicParam = new
TXAudioEffectManager.AudioMusicParam(originMusicId, originMusicUrl);
// Publish original music to the remote.
```

```
originMusicParam.publish = true;
// Music start playing time point (in milliseconds).
originMusicParam.startTimeMS = 0;

// accompMusicId: Custom identifier for the accompaniment music.
accompMusicUrl: URL of the accompaniment music resource.
TXAudioEffectManager.AudioMusicParam accompMusicParam = new
TXAudioEffectManager.AudioMusicParam(accompMusicId, accompMusicUrl);
// Publish accompaniment music to the remote.
accompMusicParam.publish = true;
// Music start playing time point (in milliseconds).
accompMusicParam.startTimeMS = 0;

// Preload the original vocal music.
mTXAudioEffectManager.preloadMusic(originMusicParam);
// Preload the accompaniment music.
mTXAudioEffectManager.preloadMusic(accompMusicParam);

// Start playing the original vocal music after a delayed playback time
(for example, 3 seconds).
mTXAudioEffectManager.startPlayMusic(originMusicParam);
// Start playing the accompaniment music after a delayed playback time
(for example, 3 seconds).
mTXAudioEffectManager.startPlayMusic(accompMusicParam);

// Switch to the original vocal music.
mTXAudioEffectManager.setMusicPlayoutVolume(originMusicId, 100);
mTXAudioEffectManager.setMusicPlayoutVolume(accompMusicId, 0);
mTXAudioEffectManager.setMusicPublishVolume(originMusicId, 100);
mTXAudioEffectManager.setMusicPublishVolume(accompMusicId, 0);

// Switch to the accompaniment music.
mTXAudioEffectManager.setMusicPlayoutVolume(originMusicId, 0);
mTXAudioEffectManager.setMusicPlayoutVolume(accompMusicId, 100);
mTXAudioEffectManager.setMusicPublishVolume(originMusicId, 0);
mTXAudioEffectManager.setMusicPublishVolume(accompMusicId, 100);
```

Note:

- It is recommended to preload music before starting playback. By loading music resources into

memory in advance, you can effectively reduce the load delay of music playback.

- In karaoke scenarios, both the original vocal and accompaniment need to be played simultaneously (distinguished by MusicID). The switch between the original vocal and accompaniment is achieved by adjusting the local and remote playback volumes.
- If the music being played has dual audio tracks (including both the original vocal and accompaniment), switching between them can be achieved by specifying the music's playback track using setMusicTrack.

8. Accompaniment Synchronization

```
// Agreed chorus start time.
long mStartPlayMusicTs = jsonObject.getLong("startPlayMusicTS");
// Actual playback progress of the current accompaniment music.
long currentProgress =
subCloud.getAudioEffectManager().getMusicCurrentPosInMS(musicId);
// Ideal playback progress of the current accompaniment music.
long estimatedProgress = TXLiveBase.getNetworkTimestamp() -
mStartPlayMusicTs;
// When the progress difference exceeds 50 ms, corrections are made.
if (estimatedProgress >= 0 && Math.abs(currentProgress -
estimatedProgress) > 50) {
    subCloud.getAudioEffectManager().seekMusicToPosInMS(musicId, (int)
estimatedProgress);
}
```

9. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Synchronize local lyrics, and transmit song progress via SEI.

```
mTXAudioEffectManager.setMusicObserver(musicId, new
TXAudioEffectManager.TXMusicPlayObserver() {
    @Override
    public void onStart(int id, int errCode) {
        // Start playing music.
    }
```

```
    @Override
    public void onPlayProgress(int id, long curPtsMs, long durationMs) {
        // Determine whether seek is needed based on the latest progress
and the local lyrics progress deviation.
        // Song progress is transmitted by sending an SEI message.
        try {
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("musicId", id);
            jsonObject.put("progress", curPtsMs);
            jsonObject.put("duration", durationMs);
            mTRTCCloud.sendSEIMsg(jsonObject.toString().getBytes(), 1);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onComplete(int id, int errCode) {
        // Music playback completed.
    }
});
```

Note:

- Ensure to set the playback event callback using this API before playing the background music. This allows to be aware of the background music's playback progress.
- The frequency of the SEI messages sent by the performer is determined by the event callback frequency. Also, the playback progress can be actively synchronized on a schedule through getMusicCurrentPosInMS.

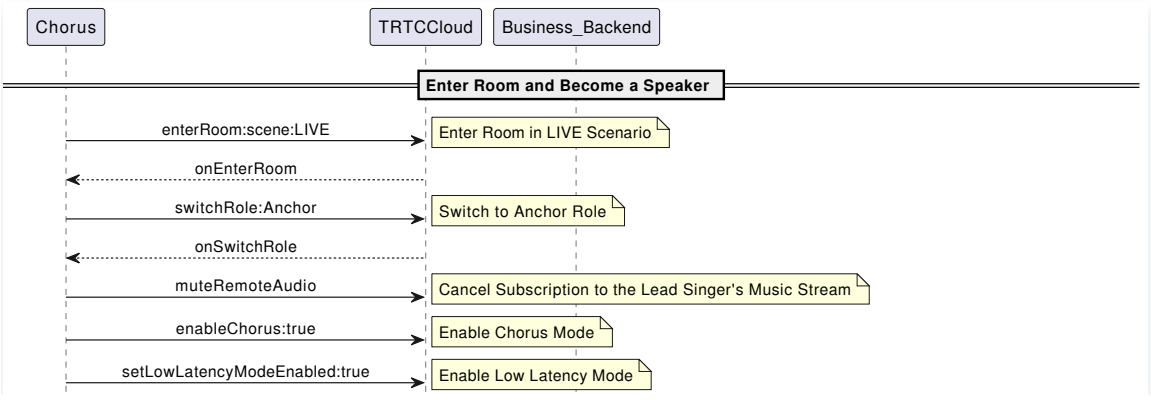
10. Become a listener and exit the room.

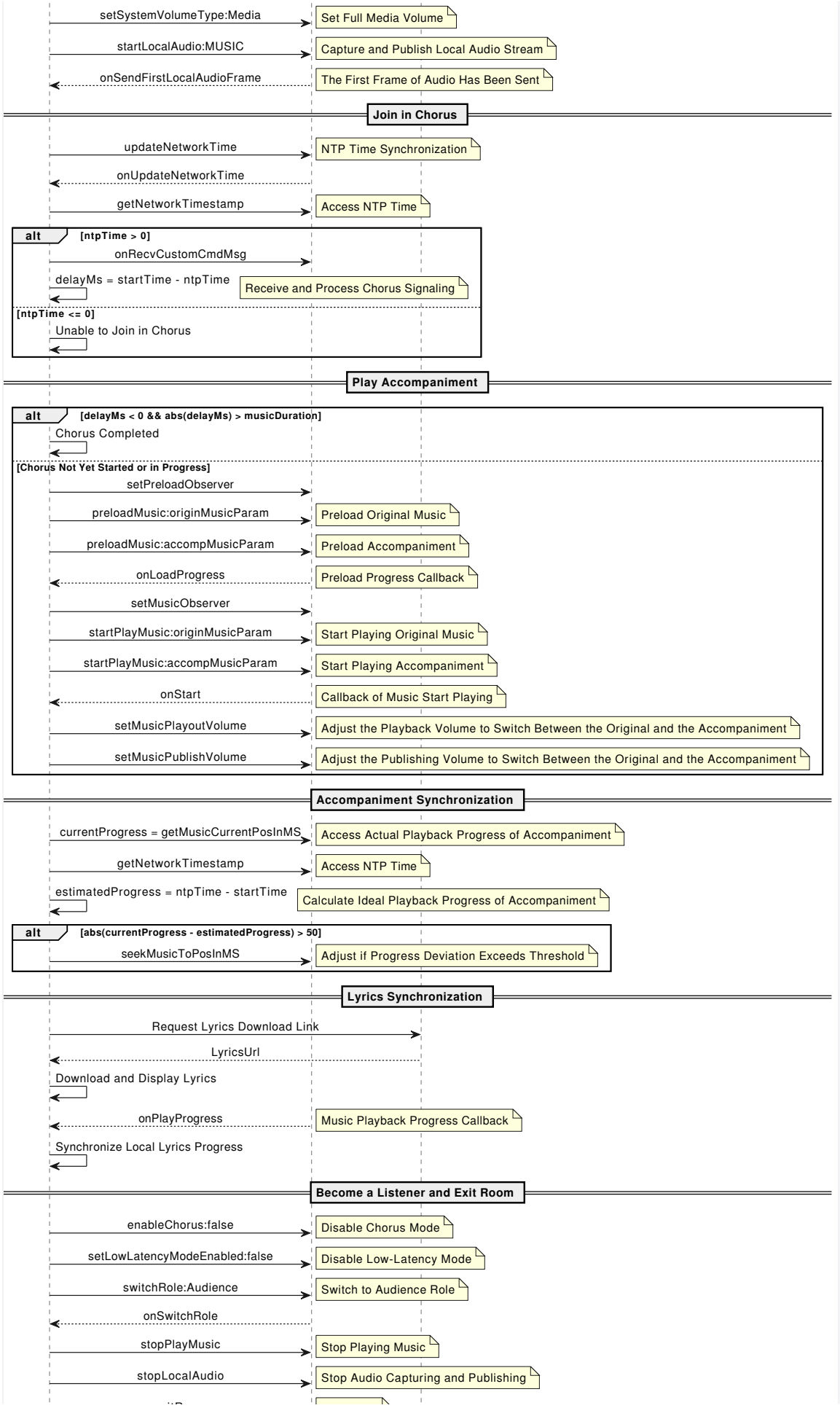
```
// The sub-instance uses the experimental API to disable chorus mode.
subCloud.callExperimentalAPI(
"{\"api\":\"enableChorus\",\"params\":
{\"enable\":false,\"audioSource\":1}}");
// The sub-instance uses the experimental API to disable low-latency
mode.
subCloud.callExperimentalAPI(
```

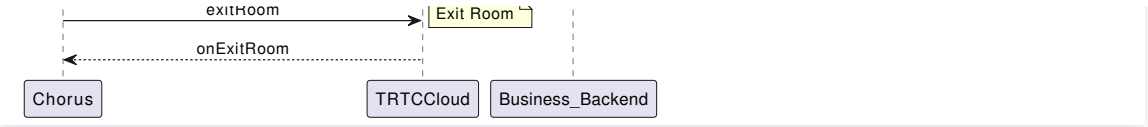
```
"{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":false}}");
// The sub-instance switches to the audience role.
subCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
// The sub-instance stops playing accompaniment music.
subCloud.getAudioEffectManager().stopPlayMusic(musicId);
// The sub-instance exits the room.
subCloud.exitRoom();

// The primary instance uses the experimental API to disable black frame
insertion.
mTRTCCloud.callExperimentalAPI(
"{\"api\":\"enableBlackStream\",\"params\": {\"enable\":false}}");
// The primary instance uses the experimental API to disable chorus
mode.
mTRTCCloud.callExperimentalAPI(
"{\"api\":\"enableChorus\",\"params\":
{\"enable\":false,\"audioSource\":0}}");
// The primary instance uses the experimental API to disable low-latency
mode.
mTRTCCloud.callExperimentalAPI(
"{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":false}}");
// The primary instance switches to the audience role.
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
// The primary instance stops local audio capture and publishing.
mTRTCCloud.stopLocalAudio();
// The primary instance exits the room.
mTRTCCloud.exitRoom();
```

Perspective 2: Chorus actions

Sequence diagram







1. Enter the room.

```
public void enterRoom(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // Take the room ID string as an example.
    params.strRoomId = roomId;
    params.userId = userId;
    // UserSig obtained from the business backend.
    params.userSig = getUserSig(userId);
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
    // Example of entering the room as an audience role.
    params.role = TRTCCloudDef.TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

// Event callback for the result of entering the room.
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result indicates the error code when you fail to enter the
room.
        Log.d(TAG, "Enter room failed");
    }
}
```

2. Go live on streams.

```
// Switched to the anchor role.
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);
```

```
// Event callback for switching the role.
@Override
public void onSwitchRole(int errCode, String errMsg) {
    if (errCode == TXLiteAVCode.ERR_NULL) {
        // Cancel subscription to music streams published by the lead
singer sub-instance.
        mTRTCCloud.muteRemoteAudio(mBgmUserId, true);
        // Use the experimental API to enable chorus mode.
        mTRTCCloud.callExperimentalAPI(
        "{\"api\":\"enableChorus\",\"params\":
{\"enable\":true,\"audioSource\":0}}");
        // Use the experimental API to enable low-latency mode.
        mTRTCCloud.callExperimentalAPI(
        "{\"api\":\"setLowLatencyModeEnabled\",\"params\":
{\"enable\":true}}");
        // Set media volume type.

mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);
        // Upstream local audio streams and set audio quality.

mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
    }
}
```

Note:

- To minimize delay, all chorus members play the accompaniment music locally. Therefore, it is necessary to cancel subscriptions to music streams published by the lead singer.
- Chorus members also need to use the experimental API to enable chorus mode and low-latency mode to optimize the chorus experience.
- In karaoke scenarios, it is recommended to set the full-range media volume and music quality to achieve a high-fidelity listening experience.

3. NTP synchronization.

```
TXLiveBase.setListener(new TXLiveBaseListener() {
    @Override
    public void onUpdateNetworkTime(int errCode, String errMsg) {
        super.onUpdateNetworkTime(errCode, errMsg);
```

```
        // errCode 0: Time synchronization successful and deviation
within 30 ms. 1: Time synchronization successful but deviation possibly
above 30 ms. -1: Time synchronization failed.
        if (errCode == 0) {
            // Time synchronization successful and NTP timestamp
obtained.
            long ntpTime = TXLiveBase.getNetworkTimestamp();
        } else {
            // If time synchronization fails, an attempt to
resynchronize can be made.
            TXLiveBase.updateNetworkTime();
        }
    }
});

TXLiveBase.updateNetworkTime();
```

> **Note:**
> NTP time synchronization results can reflect the current network quality of the application user. To ensure a good chorus experience, it is recommended not to allow users to participate in the chorus if time synchronization fails.

4. Receive chorus signaling.

```
@Override
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[]
message) {
    try {
        JSONObject json = new JSONObject(new String(message, "UTF-8"));
        // Match the chorus signaling.
        if (json.getString("cmd").equals("startChorus")) {
            long startPlayMusicTs = json.getLong("startPlayMusicTS");
            int musicId = json.getInt("musicId");
            long musicDuration = json.getLong("musicDuration");
            // Agree on the time difference between chorus time and
current time.
            long delayMs = startPlayMusicTs -
TXLiveBase.getNetworkTimestamp();
        }
```

```
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

> **Note:**
> Once the chorus members receive the chorus signaling and join in, the status should be changed to Chorus In Progress. Chorus signaling would not be responded to again before the end of this chorus round.

5. Play accompaniment, and start the chorus.

```
if (delayMs > 0) {    // The chorus has not started.
    // Begin to preload music.
    preloadMusic(musicId, 0L);
    // Play music after a delay of delayMs.
    startPlayMusic(musicId, 0L);
} else if (Math.abs(delayMs) < musicDuration) {    // The chorus is in
progress.
    // Play start time: Absolute value of the time difference + preload
delay (e.g., 400 ms).
    long startTimeMS = Math.abs(delayMs) + 400;
    // Begin to preload music.
    preloadMusic(musicId, startTimeMS);
    // Start playing music after a preload delay (e.g., 400 ms).
    startPlayMusic(musicId, startTimeMS);
} else {    // The chorus has ended.
    // Joining the chorus is not allowed.
}

// Preload music.
public void preloadMusic(int musicId, long startTimeMS) {
    // musicId: Obtained from chorus signaling. musicUrl: Corresponding
music resource URL.
    TXAudioEffectManager.AudioMusicParam musicParam = new
    TXAudioEffectManager.AudioMusicParam(musicId, musicUrl);
    // Only local music playback.
    musicParam.publish = false;
    // Music start playing time point (in milliseconds).
```

```
    musicParam.startTimeMS = startTimeMS;

    mTRTCCloud.getAudioEffectManager().preloadMusic(musicParam);
}

// Begin to play music.
public void startPlayMusic(int musicId, long startTimeMS) {
    // musicId: Obtained from chorus signaling. musicUrl: Corresponding
music resource URL.
    TXAudioEffectManager.AudioMusicParam musicParam = new
    TXAudioEffectManager.AudioMusicParam(musicId, musicUrl);
    // Only local music playback.
    musicParam.publish = false;
    // Music start playing time point (in milliseconds).
    musicParam.startTimeMS = startTimeMS;

    mTRTCCloud.getAudioEffectManager().startPlayMusic(musicParam);
}
```

Note:

- To minimize transmission delay as much as possible, chorus members perform along with the local playback of accompaniment music, and they do not need to publish or receive remote music.
- Based on `delayMs`, the current chorus status can be determined. Developers must implement the `startPlayMusic` delayed call for different statuses on their own.

6. Accompaniment Synchronization

```
// Agreed chorus start time.
long mStartPlayMusicTs = jsonObject.getLong("startPlayMusicTS");
// Actual playback progress of the current accompaniment music.
long currentProgress =
mTRTCCloud.getAudioEffectManager().getMusicCurrentPosInMS(musicId);
// Ideal playback progress of the current accompaniment music.
long estimatedProgress = TXLiveBase.getNetworkTimestamp() -
mStartPlayMusicTs;
// When the progress difference exceeds 50 ms, corrections are made.
```

```
if (estimatedProgress >= 0 && Math.abs(currentProgress -
estimatedProgress) > 50) {
    mTRTCCloud.getAudioEffectManager().seekMusicToPosInMS(musicId, (int)
estimatedProgress);
}
```

7. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Local lyric synchronization.

```
mTXAudioEffectManager.setMusicObserver(musicId, new
TXAudioEffectManager.TXMusicPlayObserver() {
    @Override
    public void onStart(int id, int errCode) {
        // Start playing music.
    }

    @Override
    public void onPlayProgress(int id, long curPtsMs, long durationMs) {
        // TODO: The logic of updating the lyric control.
        // Determine whether seek in the lyrics control is needed based
on the latest progress and the local lyrics progress deviation.

    }

    @Override
    public void onComplete(int id, int errCode) {
        // Music playback completed.
    }
});
```

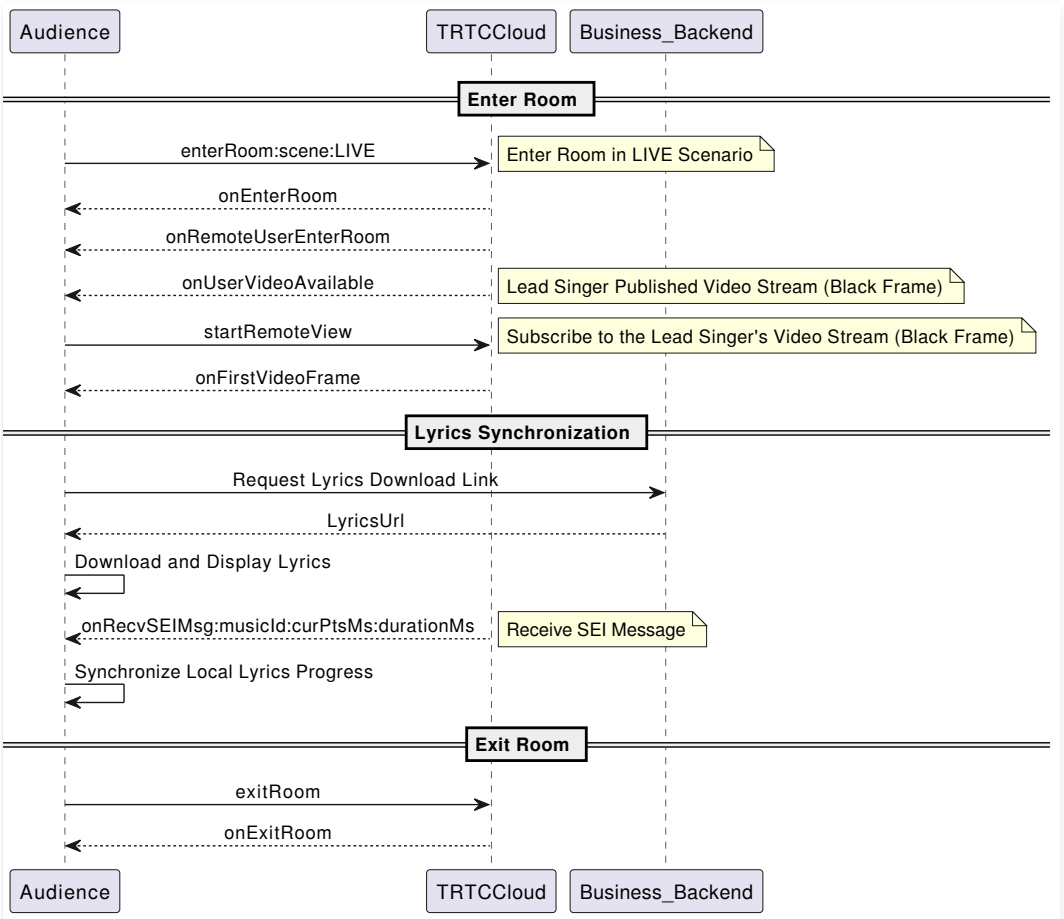
> **Note:**
> Ensure to set the playback event callback using this API before playing the background music. This allows to be aware of the background music's playback progress.

8. Become a listener and exit the room.

```
// Use the experimental API to disable chorus mode.
mTRTCCloud.callExperimentalAPI(
"{\"api\":\"enableChorus\",\"params\":
{\"enable\":false,\"audioSource\":0}}");
// Use the experimental API to disable low-latency mode.
mTRTCCloud.callExperimentalAPI(
"{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":false}}");
// Switched to the audience role.
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
// Stop playing accompaniment music.
mTRTCCloud.getAudioEffectManager().stopPlayMusic(musicId);
// Stop local audio capture and publishing.
mTRTCCloud.stopLocalAudio();
// Exit the room.
mTRTCCloud.exitRoom();
```

Perspective 3: Listener actions

Sequence diagram



1. Enter the room.

```
public void enterRoom(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // Take the room ID string as an example.
    params.strRoomId = roomId;
    params.userId = userId;
    // UserSig obtained from the business backend.
    params.userSig = getUserSig(userId);
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
    // It is recommended to enter the room as an audience role.
    params.role = TRTCCloudDef.TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

// Event callback for the result of entering the room.
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result indicates the error code when you fail to enter the
room.
        Log.d(TAG, "Enter room failed");
    }
}
```

Note:

- To better transmit SEI messages for lyrics synchronization, it is recommended to choose `TRTC_APP_SCENE_LIVE` for room-entry scenarios.
- Under the automatic subscription mode (default), audiences automatically subscribe and play the on-mic anchor's audio and video streams upon entering the room.

2. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Listener end lyric synchronization

```
@Override
public void onUserVideoAvailable(String userId, boolean available) {
    if (available) {
        mTRTCCloud.startRemoteView(userId, null);
    } else {
        mTRTCCloud.stopRemoteView(userId);
    }
}

@Override
public void onRecvSEIMsg(String userId, byte[] data) {
    String result = new String(data);
    try {
        JSONObject jsonObject = new JSONObject(result);
        int musicId = jsonObject.getInt("musicId");
        long progress = jsonObject.getLong("progress");
        long duration = jsonObject.getLong("duration");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    ...
    // TODO: The logic of updating the lyric control.
    // Based on the received latest progress and the local lyrics
progress deviation, determine whether a lyric control seek is necessary.
    ...
}
```

Note:

- Listeners need to actively subscribe to the lead singer's video streams in order to receive the SEI messages carried by black frames.
- If the lead singer's mixed stream also mixes in black frames, then only subscribing to the mixing stream robot's video stream is required.

3. Exit the room.

```
// Exit the room.
mTRTCCloud.exitRoom();

// Exit room event callback.
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "Actively call exitRoom to exit the room.");
    } else if (reason == 1) {
        Log.d(TAG, "Removed from the current room by the server.");
    } else if (reason == 2) {
        Log.d(TAG, "The current room has been dissolved.");
    }
}
```

Advanced Features

Music scoring module integration

Music scoring provides users with multi-dimensional singing scoring capabilities. Currently, supported scoring dimensions include intonation and rhythm.

1. Prepare scoring-related files.

   Prepare in advance the performance recording files to be scored, original music standard files, MIDI pitch files, and upload them to COS storage.
2. Create a music scoring task.
   - Request Method: POST(HTTP).
   - Request Address: http://service-mqk0mc83-1257411467.bj.apigw.tencentcs.com/release/job .
   - Request Header: Content-Type: application/json.

   A request sample is as follows:

Request sample:

```
{
  "action": "CreateJob",
  "secretId": "{secretId}",
  "secretKey": "{secretKey}",
  "createJobRequest": {
```

```
    "customId": "{customId}",
    "callback": "{callback}",
    "inputs": [{ "url": "{url}" }],
    "outputs": [
      {
        "contentId": "{contentId}",
        "destination": "{destination}",
        "inputSelectors": [0],
        "smartContentDescriptor": {
          "outputPrefix": "{outputPrefix}",
          "vocalScore": {
            "standardAudio": {
              "midi": {"url":"{url}"},
              "standardWav": {"url":"{url}"},
              "alignWav": {"url":"{url}"}
            }
          }
        }
      }
    ]
  }
}
```

Response sample:

```
{
    "requestId": "ac004192-110b-46e3-ade8-4e449df84d60",
    "createJobResponse": {
        "job": {
            "id": "13f342e4-6866-450e-b44e-3151431c578b",
            "state": 1,
            "customId": "{customId}",
            "callback": "{callback}",
            "inputs": [{ "url": "{url}" }],
            "outputs": [
                {
                    "contentId": "{contentId}",
                    "destination": "{destination}",
                    "inputSelectors": [0],
```

```
                    "smartContentDescriptor": {
                        "outputPrefix": "{outputPrefix}",
                        "vocalScore": {
                          "standardAudio": {
                            "midi": {"url":"{url}"},
                            "standardWav": {"url":"{url}"},
                            "alignWav": {"url":"{url}"}
                          }
                        }
                    }
                }
            ],
            "timing": {
                "createdAt": "1603432763000",
                "startedAt": "0",
                "completedAt": "0"
            }
        }
    }
}
```

3. Obtain music scoring results.

   Obtain Method: Divided into active acquisition and passive callback.

   - By querying with the ID obtained from the response packet after creating the task, if the queried task is successful (state=3), the task's Output will carry the smartContentResult structure, in which the vocalScore field stores the result JSON file name. Users can construct the output file's COS path based on the information in Output's COS and destination.

Request sample:

```
{
  "action": "GetJob",
  "secretId": "{secretId}",
  "secretKey": "{secretKey}",
  "getJobRequest": {
    "id": "{id}"
  }
}
```

Response sample:


```
{
  "requestId": "c9845a99-34e3-4b0f-80f5-f0a2a0ee8896",
  "getJobResponse": {
    "job": {
      "id": "a95e9d74-6602-4405-a3fc-6408a76bcc98",
      "state": 3,
      "customId": "{customId}",
      "callback": "{callback}",
      "timing": {
        "createdAt": "1610513575000",
        "startedAt": "1610513575000",
        "completedAt": "1610513618000"
      },
      "inputs": [{ "url": "{url}" }],
      "outputs": [
        {
          "contentId": "{contentId}",
          "destination": "{destination}",
          "inputSelectors": [0],
          "smartContentDescriptor": {
            "outputPrefix": "{outputPrefix}",
            "vocalScore": {
              "standardAudio": {
                "midi": {"url":"{url}"},
                "standardWav": {"url":"{url}"},
                "alignWav": {"url":"{url}"}
              }
            }
          },
          "smartContentResult": {
            "vocalScore": "out.json"
          }
        }
      ]
    }
  }
}
```

- Passive callbacks need to fill in the callback field when creating a task. The platform will send the entire Job structure to the address specified by the callback after the task reaches the Completed state (COMPLETED/ERROR). It is recommended to obtain task results using passive callbacks. The entire Job structure of tasks that have reached the Completed state (COMPLETED/ERROR) will be sent to the address corresponding to the callback field specified when the task was created. See the active query sample for the Job structure (under getJobResponse).

> **Note:**
> For more detailed intelligent music solution integration instructions for the music scoring module, see Music Scoring Integration.

Transparent transmission of single stream volume in mixed streams.

After the mixed streaming is enabled, the audience cannot directly obtain the on-mic anchor's single stream volume. In order to transparently transmit the single stream volume, the room owner may employ SEI to transmit the callback volume values of all on-mic anchors.

```
@Override
public void onUserVoiceVolume(ArrayList<TRTCCloudDef.TRTCVolumeInfo>
userVolumes, int totalVolume) {
    super.onUserVoiceVolume(userVolumes, totalVolume);
    if (userVolumes != null && userVolumes.size() > 0) {
        // For storing volume values corresponding to on-mic users.
        HashMap<String, Integer> volumesMap = new HashMap<>();
        for (TRTCCloudDef.TRTCVolumeInfo user : userVolumes) {
            // Can set an appropriate volume threshold.
            if (user.volume > 10) {
                volumesMap.put(user.userId, user.volume);
            }
        }
        Gson gson = new Gson();
        String body = gson.toJson(volumesMap);
        // Transmit a collection of on-mic users' volume via SEI
messages.
        mTRTCCloud.sendSEIMsg(body.getBytes(), 1);
    }
}

@Override
```

```
public void onRecvSEIMsg(String userId, byte[] data) {
    Gson gson = new Gson();
    HashMap<String, Integer> volumesMap = new HashMap<>();
    try {
        String message = new String(data, "UTF-8");
        volumesMap = gson.fromJson(message, volumesMap.getClass());
        for (String userId : volumesMap.keySet()) {
            // Print the volume levels of single streams of all on-mic
users.
            Log.i(userId, String.valueOf(volumesMap.get(userId)));
        }
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
```

> **Note:**
> The prerequisite for using SEI messages to transparently transmit single stream volume through a mixed stream is that the room owner must either be video streaming or have black frame insertion enabled and furthermore, the audiences must actively subscribe to the room owner's video stream.

Real-time network quality callback

You can listen to `onNetworkQuality` to real-time monitor the network quality of both local and remote users. This callback is thrown every 2 seconds.

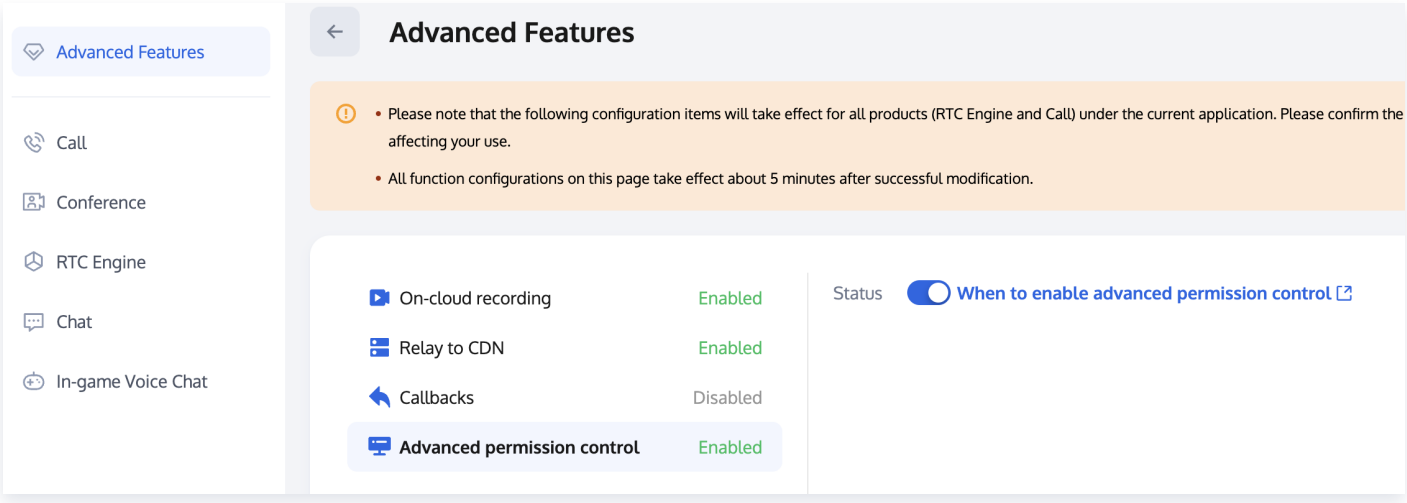
```
private class TRTCCloudImplListener extends TRTCCloudListener {
    @Override
    public void onNetworkQuality(TRTCCloudDef.TRTCQuality localQuality,
                                 ArrayList<TRTCCloudDef.TRTCQuality>
remoteQuality) {
        // localQuality userId is empty. It represents the local user's
network quality evaluation result.
        // remoteQuality represents the remote user's network quality
evaluation result. The result is affected by both remote and local
factors.
        switch (localQuality.quality) {
            case TRTCCloudDef.TRTC_QUALITY_Excellent:
                Log.i(TAG, "The current network is excellent.");
```

```
                break;
            case TRTCCloudDef.TRTC_QUALITY_Good:
                Log.i(TAG, "The current network is good.");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Poor:
                Log.i(TAG, "The current network is moderate.");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Bad:
                Log.i(TAG, "The current network is poor.");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Vbad:
                Log.i(TAG, "The current network is very poor.");
                break;
            case TRTCCloudDef.TRTC_QUALITY_Down:
                Log.i(TAG, "The current network does not meet the
minimum requirements of TRTC.");
                break;
            default:
                Log.i(TAG, "Undefined.");
                break;
        }
    }
}
```

Advanced permission control

TRTC advanced permission control can be used to set different entry permissions for different rooms, such as advanced VIP rooms. It can also be used to control the permission for the audience to speak, such as handling ghost microphones.

Step 1: Enable the Advanced Permission Control Switch in the TRTC console application's advanced features page.



Note:

Once advanced permission control is enabled for a certain SDKAppID, all users using that SDKAppID need to pass in the `privateMapKey` parameter in `TRTCParams` to successfully enter the room. Therefore, if you have users online using this SDKAppID, do not enable this feature.

Step 2: Generate privateMapKey on the backend. For sample code, see privateMapKey computation source code.

Step 3: Room entry verification & speaking permission verification with PrivateMapKey.

- Room entry verification

```
TRTCCloudDef.TRTCParams mTRTCParams = new TRTCCloudDef.TRTCParams();
mTRTCParams.sdkAppId = SDKAPPID;
mTRTCParams.userId = mUserId;
mTRTCParams.strRoomId = mRoomId;
// UserSig obtained from the business backend.
mTRTCParams.userSig = getUserSig();
// PrivateMapKey obtained from the backend.
mTRTCParams.privateMapKey = getPrivateMapKey();
mTRTCParams.role = TRTCCloudDef.TRTCRoleAudience;
mTRTCCloud.enterRoom(mTRTCParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
```

- Speaking permission verification

```
// Pass in the latest PrivateMapKey obtained from the backend into the
role switching API.
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor, getPrivateMapKey());
```

Exception Handling

Exception error handling

When the TRTC SDK encounters an unrecoverable error, the error will be thrown in the `onError` callback. For details, see Error Code Table.

1. UserSig related

UserSig verification failure will lead to room-entering failure. You can use the UserSig tool for verification.

Enumeration	Value	Description
ERR_TRTC_INVALID_USER_SIG	-3320	Room entry parameter userSig is incorrect. Check if `TRTCParams.userSig` is empty.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	UserSig verification failed. Check if the parameter `TRTCParams.userSig` is filled in correctly or has expired.

2. Room entry and exit related

If failed to enter the room, you should first verify the correctness of the room entry parameters. It is essential that the room entry and exit APIs are called in a paired manner. This means that, even in the event of a failed room entry, the room exit API must still be called.

Enumeration	Value	Description
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	Room entry request timed out. Check if your internet connection is lost or if a VPN is enabled. You may also attempt to switch to 4G for testing.
ERR_TRTC_INVALID_SDK_APPID	-3317	Room entry parameter sdkAppId is incorrect. Check if `TRTCParams.sdkAppId` is empty.
ERR_TRTC_INVALID_ROOM_ID	-3318	Room entry parameter roomId is incorrect. Check if `TRTCParams.roomId` or `TRTCParams.strRoomId` is empty. Note that roomId and strRoomId cannot be used interchangeably.

ERR_TRTC_INVALID_USER_ID	-3319	Room entry parameter userId is incorrect. Check if `TRTCParams.userId` is empty.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	Room entry request is denied. Check if `enterRoom` is called consecutively to enter rooms with the same ID.

3. Device related

Errors for relevant monitoring devices. Prompt the user via UI in case of relevant errors.

Enumeration	Value	Description
ERR_MIC_START_FAIL	-1302	Failed to open the mic. For example, if there is an exception for the mic's configuration program (driver) on a Windows or macOS device, you should try disabling then re-enabling the device, restarting the machine, or updating the configuration program.
ERR_SPEAKER_START_FAIL	-1321	Failed to open the speaker. For example, if there is an exception for the speaker's configuration program (driver) on a Windows or macOS device, you should try disabling then re-enabling the device, restarting the machine, or updating the configuration program.
ERR_MIC_OCCUPY	-1319	The mic is occupied. This occurs when, for example, the user is currently having a call on the mobile device.

Issues with IEMs

1. How to enable IEMs feature and set the volume?

```
// Enable IEMs.
mTRTCCloud.getAudioEffectManager().enableVoiceEarMonitor(true);
// Set the volume of IEMs.
mTRTCCloud.getAudioEffectManager().setVoiceEarMonitorVolume(int volume);
```

> **Note:**
> The IEMs can be set in advance without having to monitor audio routing changes. Once headphones are connected, the IEMs feature will automatically take effect.

2. The IEMs feature does not take effect after enabled.

Due to the high hardware delay of Bluetooth headphones, it is recommended to prompt the anchor to wear wired headphones on the user interface. Also, it should be noted that not all smartphones will achieve

excellent IEMs effect after this feature is enabled. TRTC SDK has already blocked this feature on some smartphones with poor effect.

3. High IEM delay

Check if Bluetooth headphones are in use. Due to the high hardware delay of Bluetooth headphones, wired headphones are recommended. Additionally, you can try improving the issue of high IEM delay by enabling hardware IEM through the experimental API `setSystemAudioKitEnabled`. Hardware IEMs have better performance and lower delay. Software IEMs have higher delay but better compatibility. Currently, for Huawei and VIVO devices, SDK defaults to hardware IEMs. Other devices default to software IEMs. If there are compatibility issues with hardware IEMs, contact us to configure forced use of software IEMs.

Issues with NTP sync

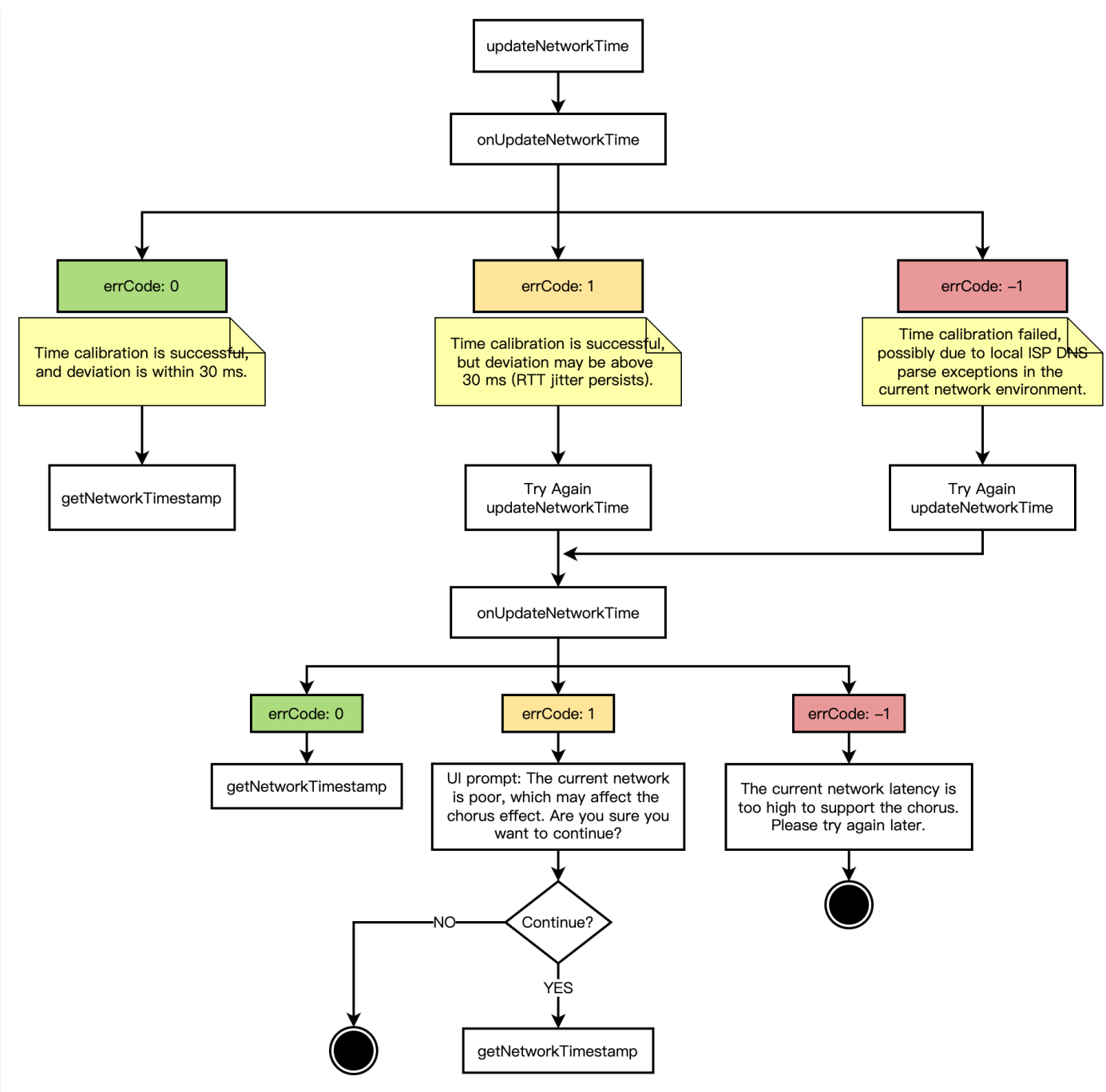
1. NTP time sync finished, but result maybe inaccurate

NTP sync is successful, but the deviation may still be more than 30 milliseconds. This indicates a poor client network environment with persistent RTT jitter.

2. Error in AddressResolver: No address associated with hostname

NTP sync has failed, possibly due to a temporary exception in local ISP DNS resolution under the current network environment. Try again later.

3. NTP service retry processing logic.



Issue with resource paths for playing music

In karaoke scenarios, when using TRTC SDK to play accompaniment music, you can choose to play from local or online music resources. Currently, playback paths only support URLs of online resources, absolute paths of music files in device external storage, and application private directories. It does not support paths in directories like assets in Android development.
You can work around this issue by copying resource files from the assets directory to either the device external storage or the application private directory beforehand. Sample code is as follows:

```
public static void copyAssetsToFile(Context context, String name) {
    // The files directory under the application's directory.
    String savePath = ContextCompat.getExternalFilesDirs(context, null)
[0].getAbsolutePath();
    // The cache directory under the application's directory.
    // String savePath =
getApplication().getExternalCacheDir().getAbsolutePath();
    // The files directory under the application's private storage
directory.
    // String savePath =
getApplication().getFilesDir().getAbsolutePath();
    String filename = savePath + "/" + name;
    File dir = new File(savePath);
    // Create the directory if it does not exist.
    if (!dir.exists()) {
        dir.mkdir();
    }
    try {
        if (!(new File(filename)).exists()) {
            InputStream is =
context.getResources().getAssets().open(name);
            FileOutputStream fos = new FileOutputStream(filename);
            byte[] buffer = new byte[1024];
            int count = 0;
            while ((count = is.read(buffer)) > 0) {
                fos.write(buffer, 0, count);
            }
            fos.close();
            is.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
```

```
    }
}
```

- Application External Storage Files Directory Path: /storage/emulated/0/Android/data/<package_name>/files/<file_name>.
- Application External Storage Cache Directory Path: /storage/emulated/0/Android/data/<package_name>/cache/file_name>.
- Application Private Storage Files Directory Path: /data/user/0/<package_name>/files/<file_name>.

Note:

- If you provide a path that is an external storage path outside of the application's specific directories, on Android 10 and above devices, you may face denial of access to the resource. This is due to Google introducing Partition Storage, a new storage management system. You can temporarily bypass this by adding the following code inside the <application> tag in the AndroidManifest.xml file: `android:requestLegacyExternalStorage="true"`. This attribute only takes effect on applications with targetSdkVersion 29 (Android 10), and applications with a higher version targetSdkVersion are still recommended to use the application's private or external storage paths.
- For TRTC SDK v11.5 and later, playback of local music resources on Android devices via Content URI from Content Provider components is supported.
- On Android 11 and HarmonyOS 3.0 or later, if you cannot access resource files in the external storage directory, you need to request the `MANAGE_EXTERNAL_STORAGE` permission:
  - First, you need to add the following entry in your application's AndroidManifest file.

```
<manifest ...>
    <!-- This is the permission itself -->
    <uses-permission
android:name="android.permission.MANAGE_EXTERNAL_STORAGE" />

    <application ...>
        ...
    </application>
</manifest>
```

  - Then, guide users to manually grant this permission at the point in your application where it is needed.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
```

```
    if (!Environment.isExternalStorageManager()) {
        Intent intent = new
Intent(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION);
        Uri uri = Uri.fromParts("package", getPackageName(),
null);
        intent.setData(uri);
        startActivity(intent);
    }
} else {
    // For Android versions less than Android 11, you can use
the old permissions model
    ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.WRITE_EXTERNAL_STORAGE}, REQUEST_CODE);
}
```

Issues with real-time chorus usage

1. **Why does the lead singer in real-time chorus scenarios need to use dual-instance streaming?**

   In real-time chorus scenarios, to minimize end-to-end delay and achieve sync between vocals and accompaniment, a common approach is to use dual instances at the lead singer's end to separately upload vocal and accompaniment streams, while other chorus participants only upload their vocal streams and locally play the accompaniment. In this case, each chorus participant needs to subscribe to the lead singer's vocal stream, while refraining from subscribing to the lead singer's music stream. This setup can only be achieved by implementing dual-instance separate streaming.
2. **Why is it recommended to enable mixing pushback in real-time chorus scenarios?**

   Having the audience pull multiple single streams at the same time is likely to result in misalignment between multiple vocal streams and accompaniment streams. Pulling a mixed stream can ensure absolute alignment of all streams and reduce downstream bandwidth.
3. **What are the uses of SEI in real-time chorus scenarios?**
   - Transmitting accompaniment music progress, for lyric sync on the audience's end.
   - Transparently transmitting single stream volume through a mixed stream, for display as sound waves on the listener's end.
4. **Loading accompaniment music takes a long duration, causing significant playback delay?**

   Loading network music resources via the SDK incurs a certain delay. It is recommended to initiate music pre-loading before starting playback.

```
mTRTCCloud.getAudioEffectManager().preloadMusic(musicParam);
```

5. **When singing along with accompaniment, the vocals are barely audible. Is the music overwhelming the vocals?**

   If the default volume settings result in the accompaniment overwhelming the vocals, it is recommended to adjust the volume balance between the music and vocals accordingly.

```
// Set the local playback volume of a piece of background music.
mTRTCCloud.getAudioEffectManager().setMusicPlayoutVolume(musicID,
volume);
// Set the remote playback volume of a specific background music.
mTRTCCloud.getAudioEffectManager().setMusicPublishVolume(musicID,
volume);
// Set the local and remote volume of all background music.
mTRTCCloud.getAudioEffectManager().setAllMusicVolume(volume);
// Set the volume of voice capture.
mTRTCCloud.getAudioEffectManager().setVoiceCaptureVolume(volume);
```

iOS

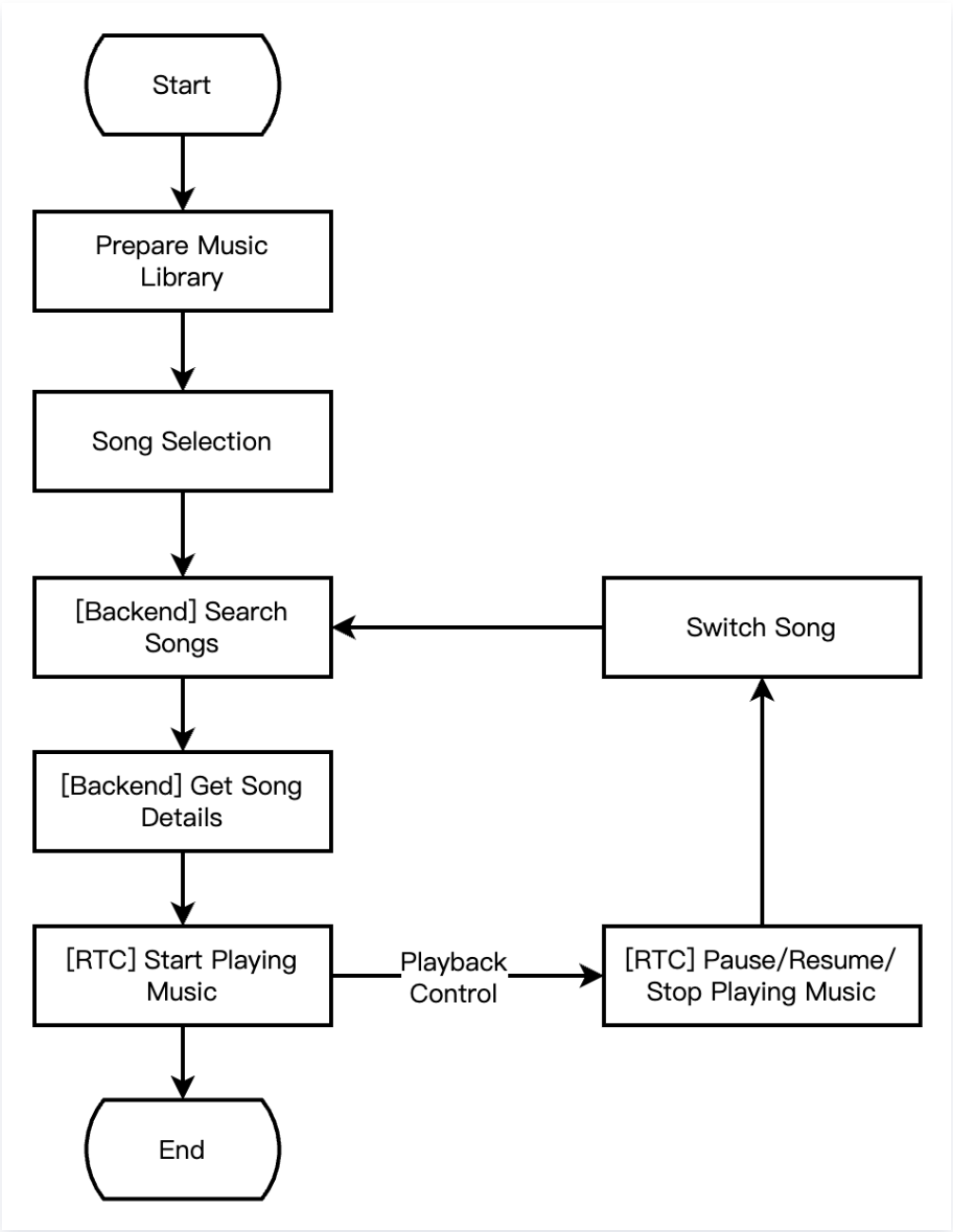
최종 업데이트 날짜:: 2024-07-18 14:26:14

Business Process

This section summarizes some common business processes in online karaoke, helping you better understand the implementation process of the entire scenario.

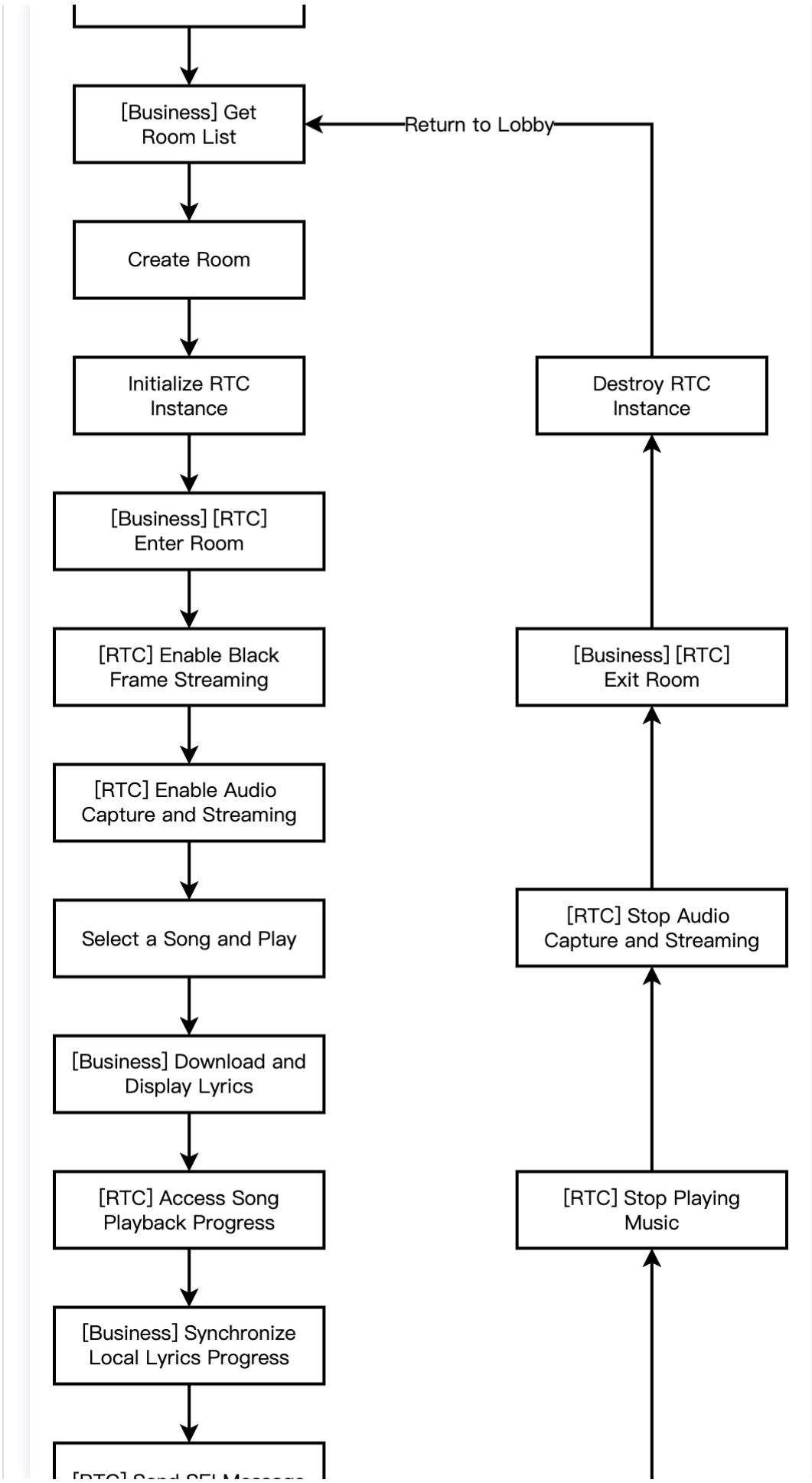
Song request process

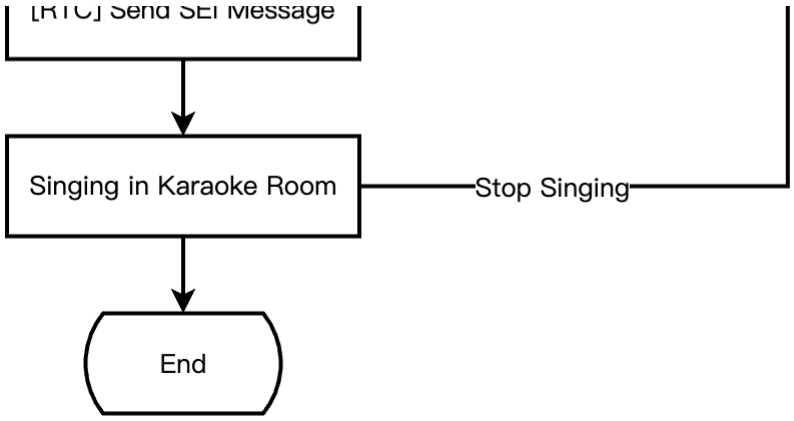
The following figure shows the process of requesting songs from a music repository on the business side and playing them using the TRTC SDK.



Solo singing process

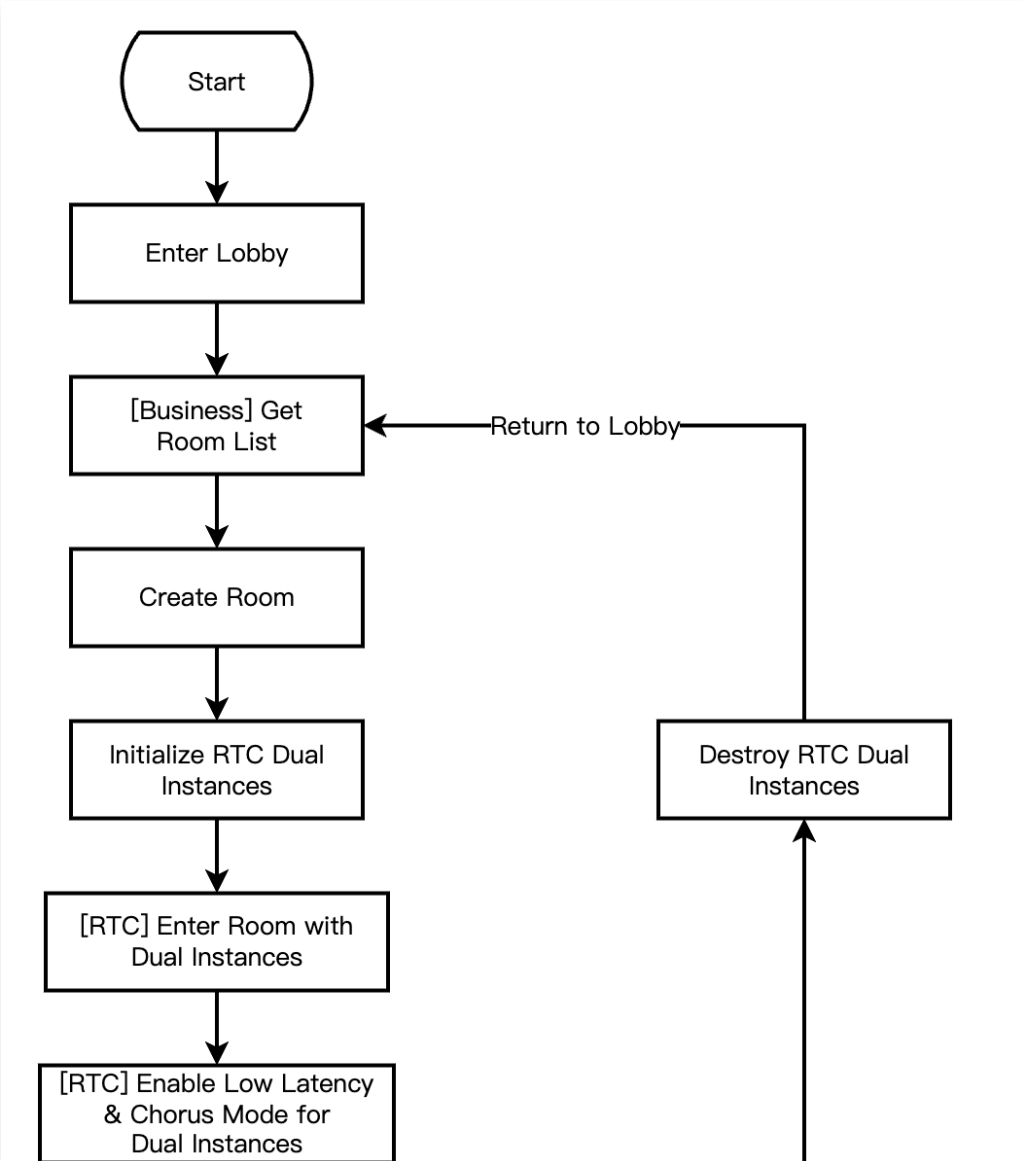
The following figure shows the process of a solo singing turn-taking game, that is, the performer enters a room to perform, stops performing, and exits the room.

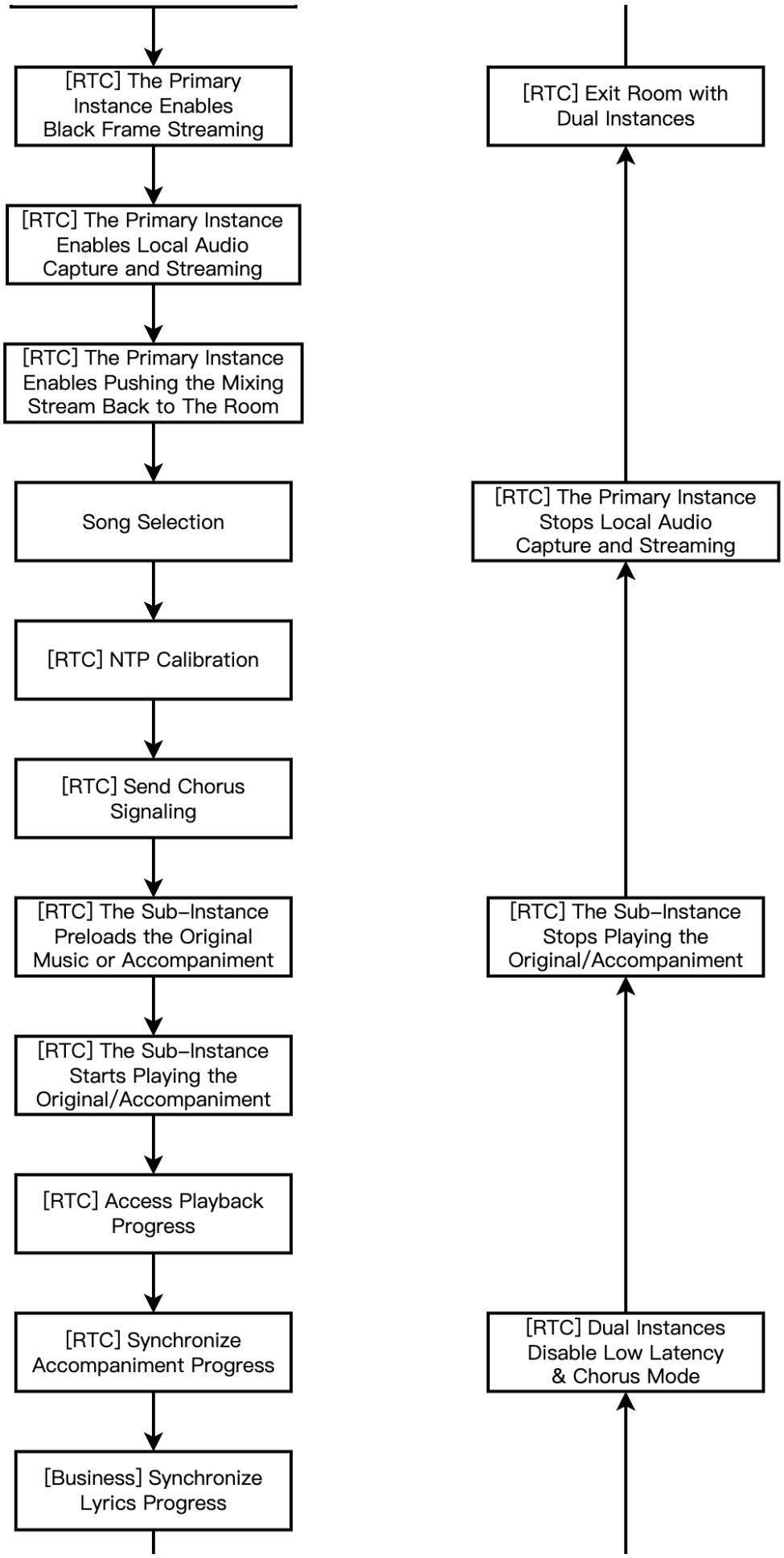


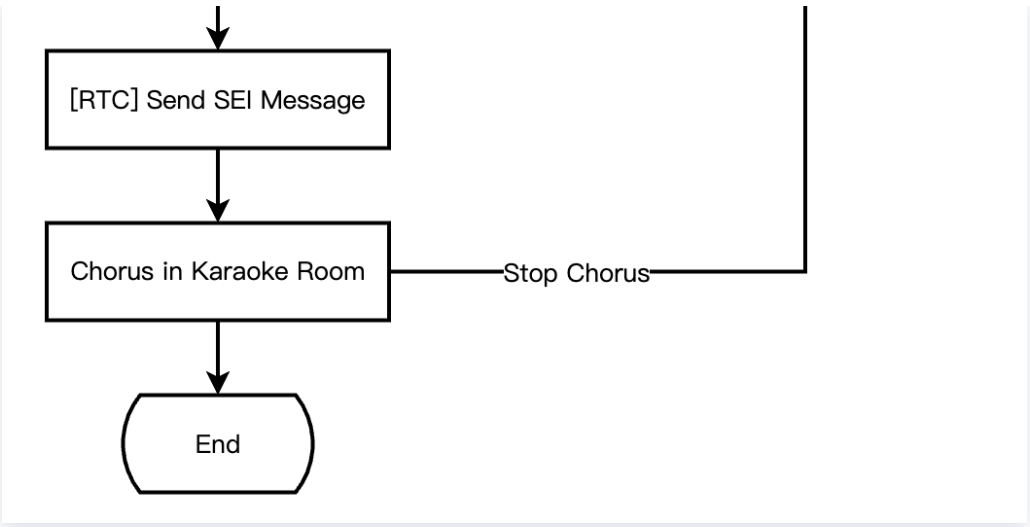


Lead singer process

The following figure shows the process of a real-time chorus game, that is, the lead singer initiates a chorus, stops the chorus, and exits the room.

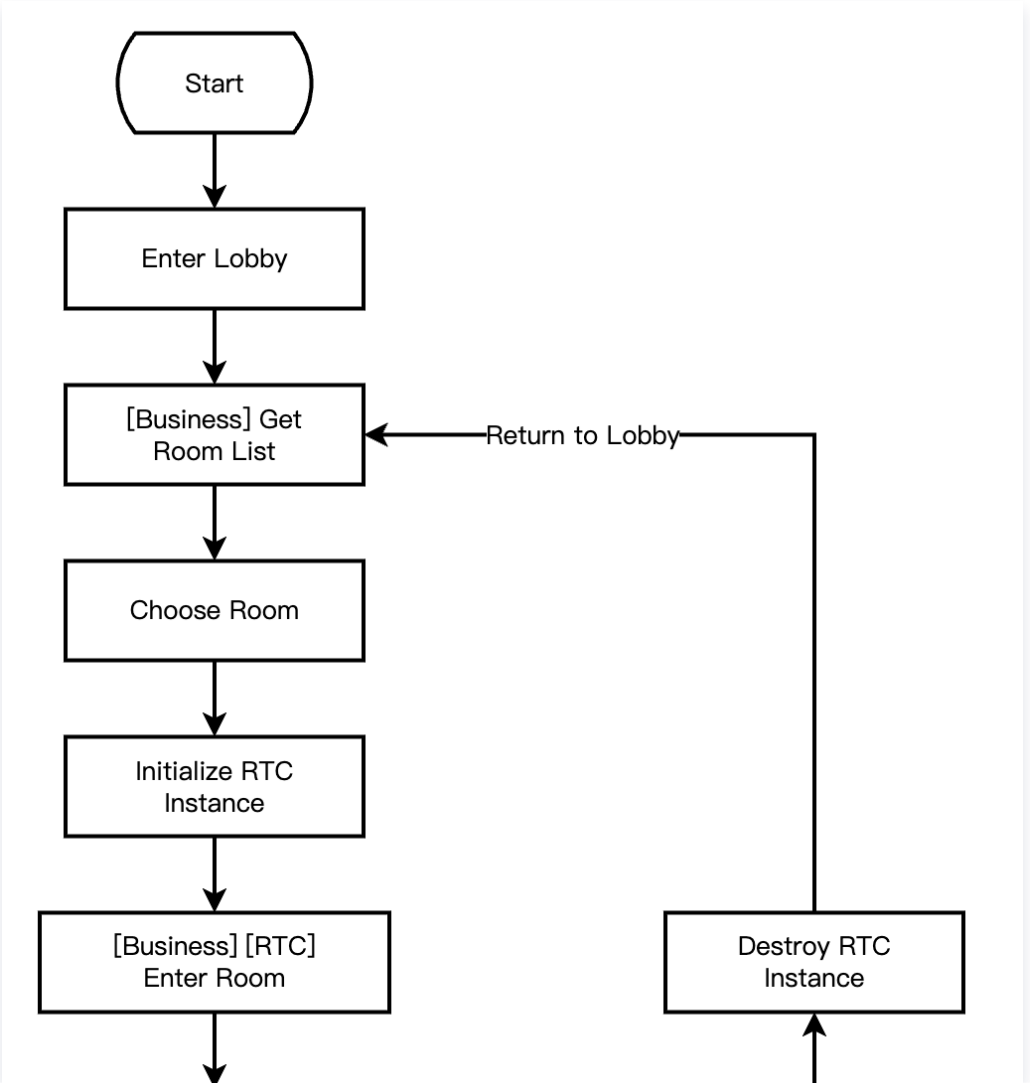


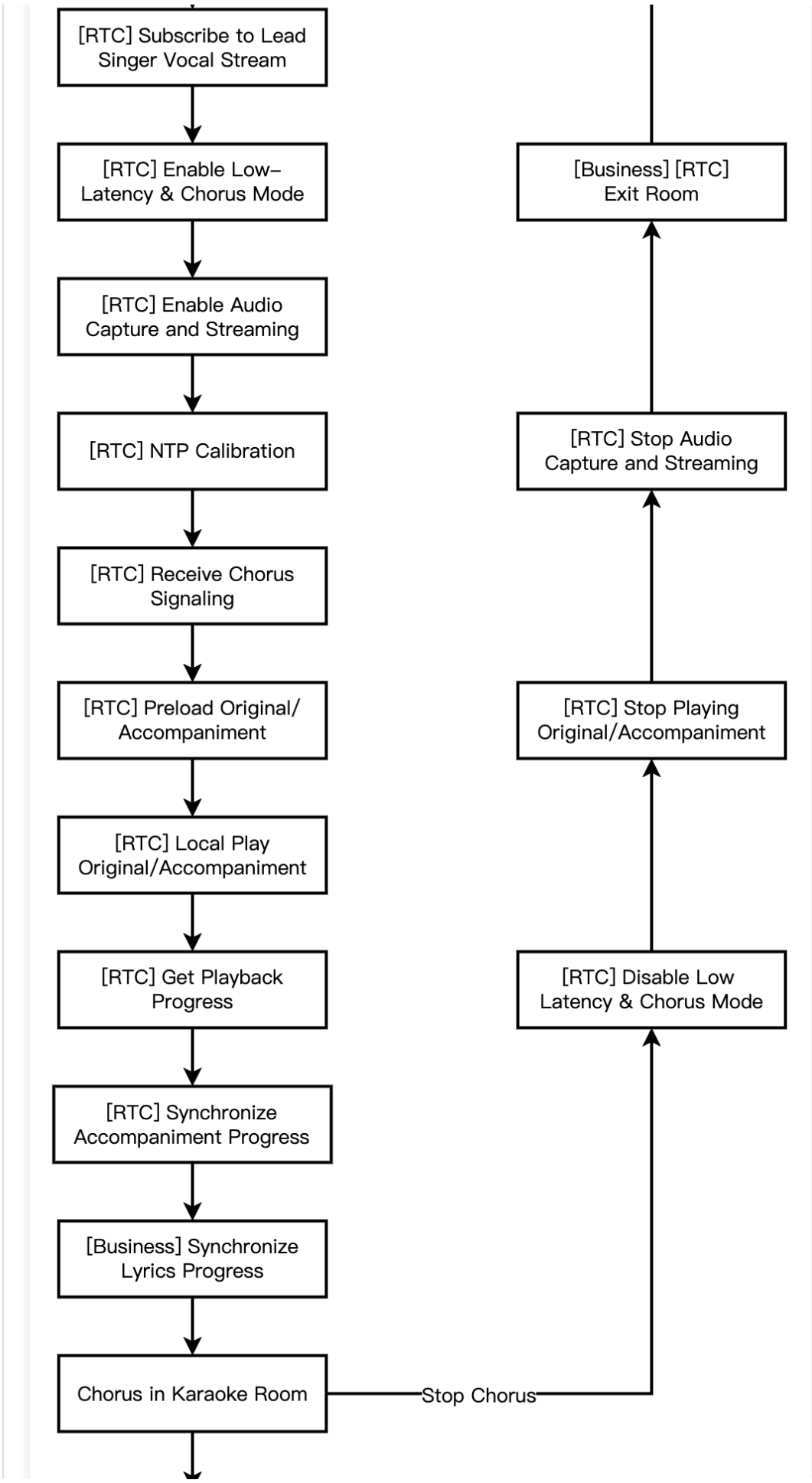


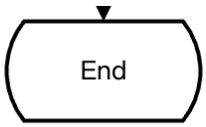


Chorus process

The following figure shows the process of a real-time chorus game, that is, the chorus members join the chorus, stop the chorus, and exit the room.

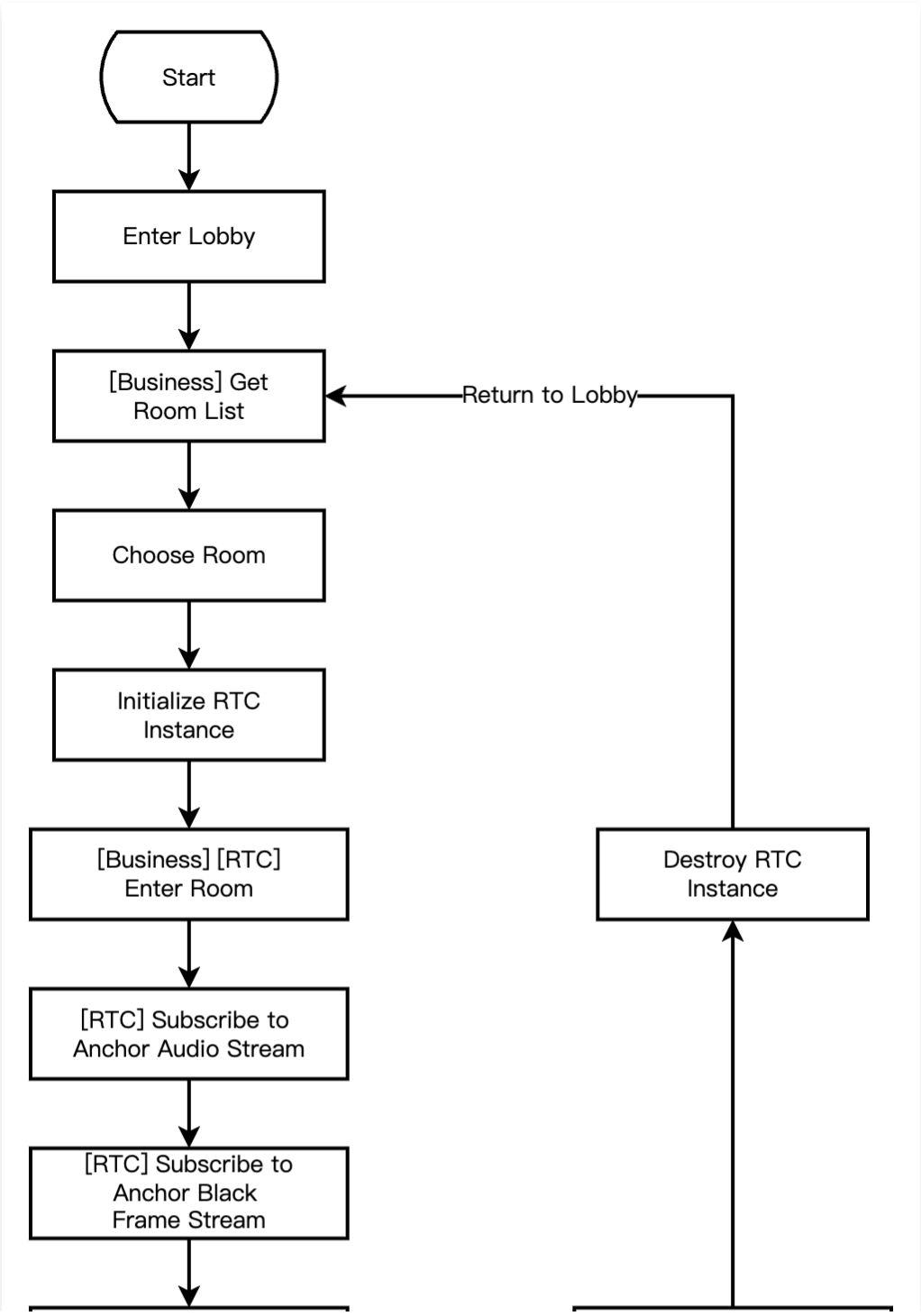


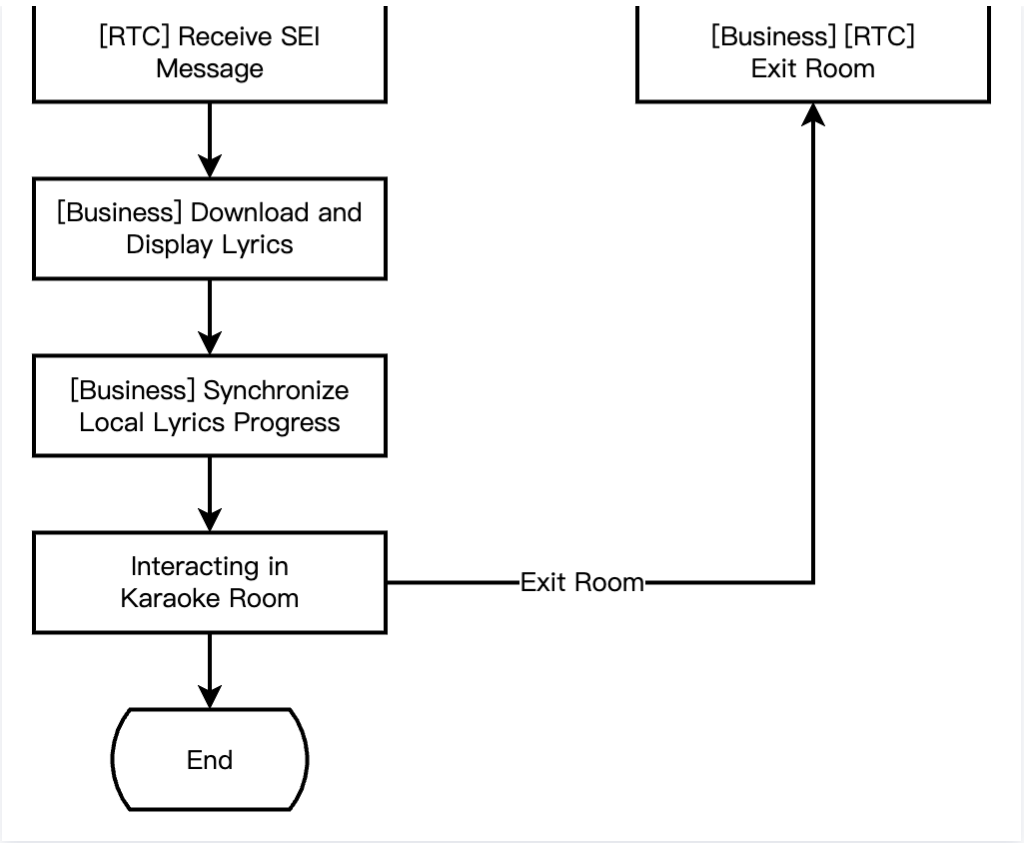




Audience process

The following figure shows the process of an online karaoke scenario, that is, the audience enters the room to listen to songs and synchronizes lyrics.





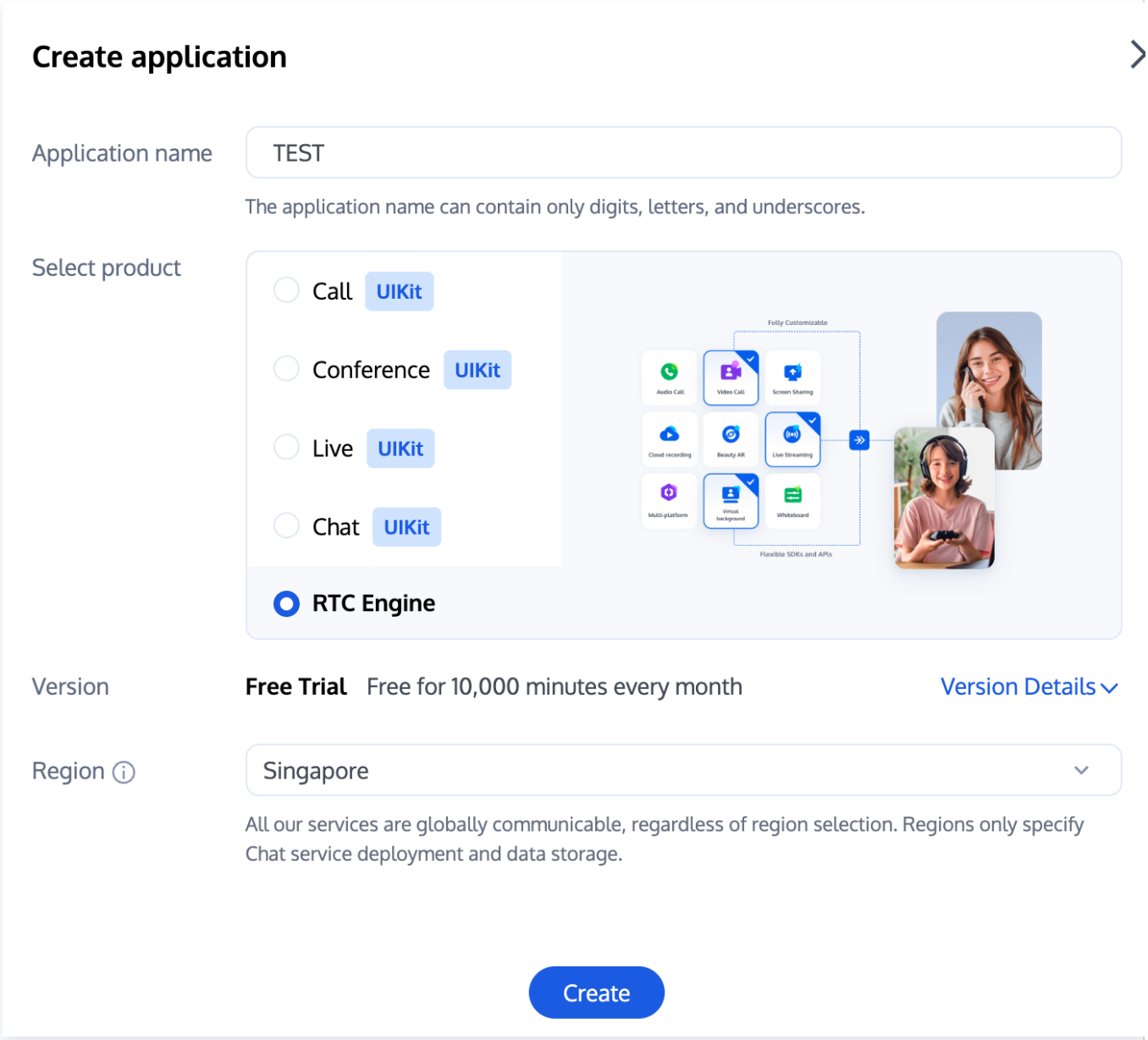
Integration Preparation

Step 1. Activating the service.

The online karaoke scenarios usually require two paid PaaS services from Tencent Cloud: Tencent Real-Time Communication (TRTC) and Intelligent Music Solution for construction. TRTC is responsible for providing real-time audio and video interaction capabilities. Intelligent Music Solution is responsible for providing lyric recognition, smart composition, music recognition, and music scoring capabilities.

Activate TRTC service.

1. First, you need to log in to the Tencent Real-Time Communication (TRTC) console to create an application. You can choose to upgrade the TRTC application version according to your needs. For example, the professional edition unlocks more value-added feature services.

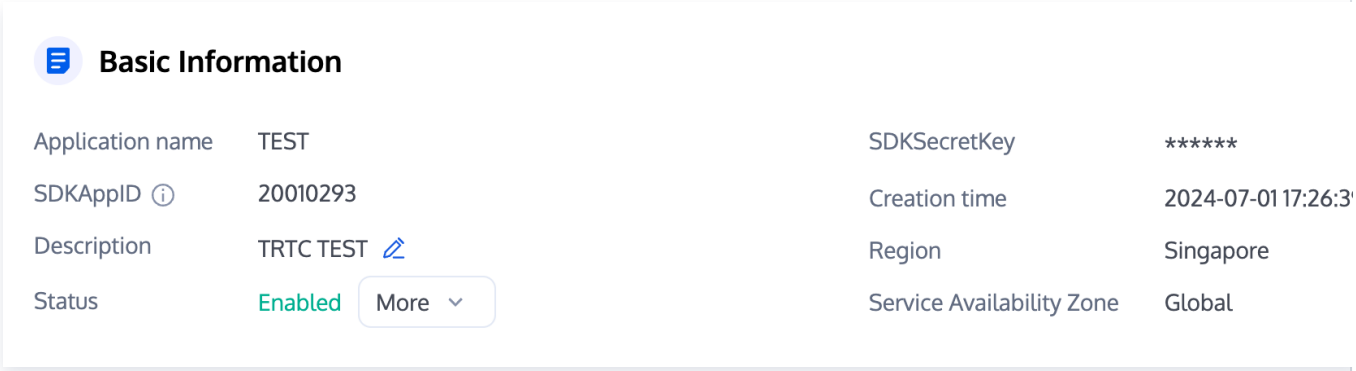


Note:

- It is recommended to create two applications for testing and production environments, respectively. Each Tencent Cloud account (UIN) is given 10,000 minutes of free duration every month for one year.
- TRTC offers monthly subscription plans including the experience edition (default), basic edition, and professional edition. Different value-added feature services can be unlocked. For details, see Version Features and Monthly Subscription Plan Instructions.

2. After an application is created, you can see the basic information of the application in the

Application Management – Application Overview section. It is important to keep the **SDKAppID** and **SDKSecretKey** safe for later use and to avoid key leakage that could lead to traffic theft.



Activate the Intelligent Music service.

Preparation

1. Go to the Purchase Page to activate the music service, and choose the appropriate features such as music scoring to activate.
2. Create an AK/SK Key Pair in CAM (namely, a programmable access user that does not require log-in or any user permissions).
3. Create a COS Bucket, and in the COS Bucket Management interface, authorize the read and write permissions of the COS Bucket to the created programmable access user.
4. Prepare the parameters.

- operateUin: Tencent Cloud sub-user's account ID.
- cosConfig: COS related parameters.
  - secretId: Bucket's secretId.
  - secretKey: Bucket's secretKey.
  - bucket: Bucket's name.
  - region: Bucket's region, for example, ap-guangzhou.

Activation and registration.

After the preparation is completed, proceed with registration activation by initiating a request, with an estimated wait time of about 2 minutes.

Initiate request.

```
curl -X POST \
  http://service-mqk0mc83-
1257411467.bj.apigw.tencentcs.com/release/register \
  -H 'Content-Type: application/json' \
  -H 'Cache-control: no-cache' \
  -d '{
    "requestId": "test-regisiter-service",
    "action": "Register",
    "registerRequest": {
      "operateUin": <operateUin>,
      "userName": <customedName>,
      "cosConfig": {
        "secretId": <CosConfig.secretId>,
        "secretKey": <CosConfig.secretKey>,
        "bucket": <CosConfig.bucket>,
        "region": <CosConfig.region>
      }
    }
  }'
```

Request result:

```
{
    "requestId": "test-regisiter-service",
    "registerInfo": {
        "tmpContentId": <tmpContentId>,
        "tmpSecretId": <tmpSecretId>,
        "tmpSecretKey": <tmpSecretKey>,
        "apiGateSecretId": <apiGateSecretId>,
        "apiGateSecretKey": <apiGateSecretKey>,
        "demoCosPath": "UIN_demo/run_musicBeat.py",
        "usageDescription": "Download the python version demo file
[UIN_demo/run_musicBeat.py] from the COS bucket [CosConfig.bucket],
replace the input file in the demo, and then execute python
run_musicBeat.py",
        "message": "Registration successful, and thank you for
registering.",
        "createdAt": <createdAt>,
```

```
        "updatedAt": <updatedAt>
    }
}
```

Run verification.

After the above activation and registration service are completed, a python version executable demo example based on music beat recognition capability will be generated in the `demoCosPath` directory. Execute the command `python run_musicBeat.py` in a networked environment for verification.

> **Note:**
> For more detailed intelligent music solution integration instructions, see Integration Guide.

Step 2: Importing SDK.

The TRTC SDK is now available on **CocoaPods**. We recommend integrating the SDK via CocoaPods.

1. Install CocoaPods.

   Enter the following command in a terminal window (you need to install Ruby on your Mac first):

```
sudo gem install cocoapods
```

2. Create a Podfile.

   Go to the project directory, and enter the following command. A Podfile file will then be created in the project directory.

```
pod init
```

3. Edit the Podfile.

   Choose the appropriate version for your project and edit the Podfile.

```
platform :ios, '8.0'
    target 'App' do

    # TRTC Lite Edition
    # The installation package has the minimum incremental size. But it
only supports two features of Real-Time Communication (TRTC) and
```

```
TXLivePlayer for live streaming playback.
    pod 'TXLiteAVSDK_TRTC', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_TRTC.podspe
c'

    # Pro Edition
    # Includes a wide range of features such as Real-Time Communication
(TRTC), TXLivePlayer for live streaming playback, TXLivePusher for RTMP
push streams, TXVodPlayer for on-demand playback, and UGSV for short
video recording and editing.
    # pod 'TXLiteAVSDK_Professional', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_Professiona
l.podspec'

end
```

4. Update and install the SDK.

   Enter the following command in a terminal window to update the local repository files and install the SDK.

```
pod install
```

   Or use the following command to update the local repository.

```
pod update
```

Upon the completion of pod command execution, an .xcworkspace project file integrated with the SDK will be generated. Double-click to open it.

Note:

- If the pod search fails, it is recommended to try updating the local repo cache of pod. The update command is as follows.

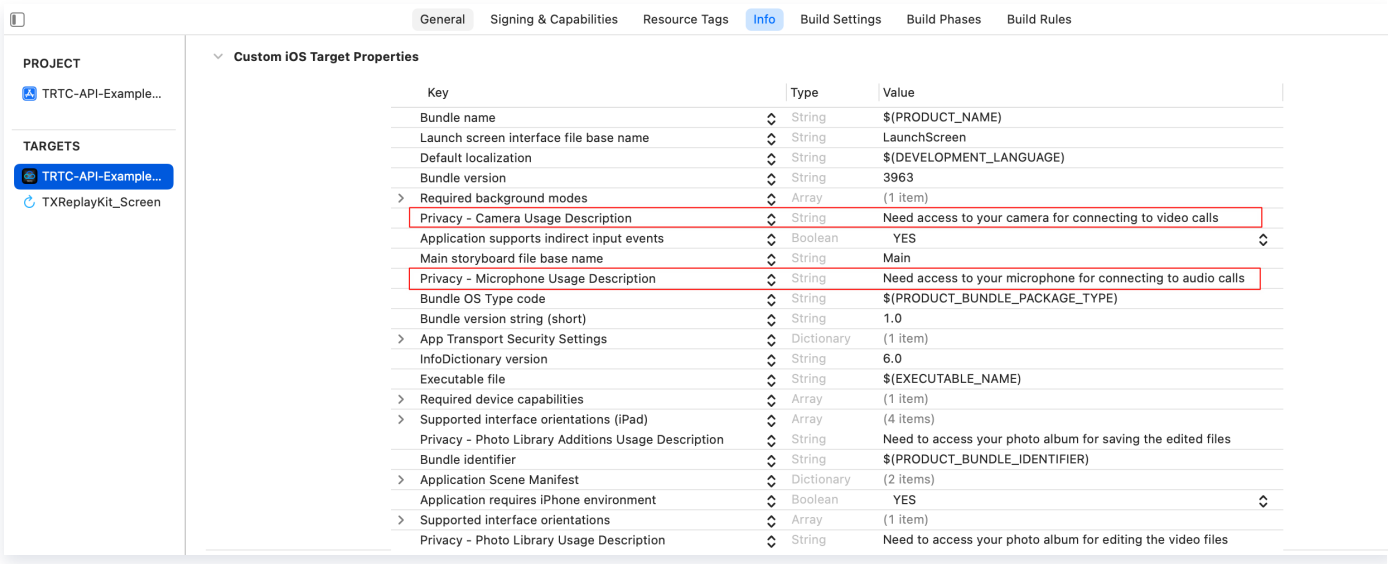
```
pod setup
pod repo update
rm ~/Library/Caches/CocoaPods/search_index.json
```

- Besides CocoaPods integration, you can also choose to download the SDK and manually import it. For details, see Manually Integrating the TRTC SDK.

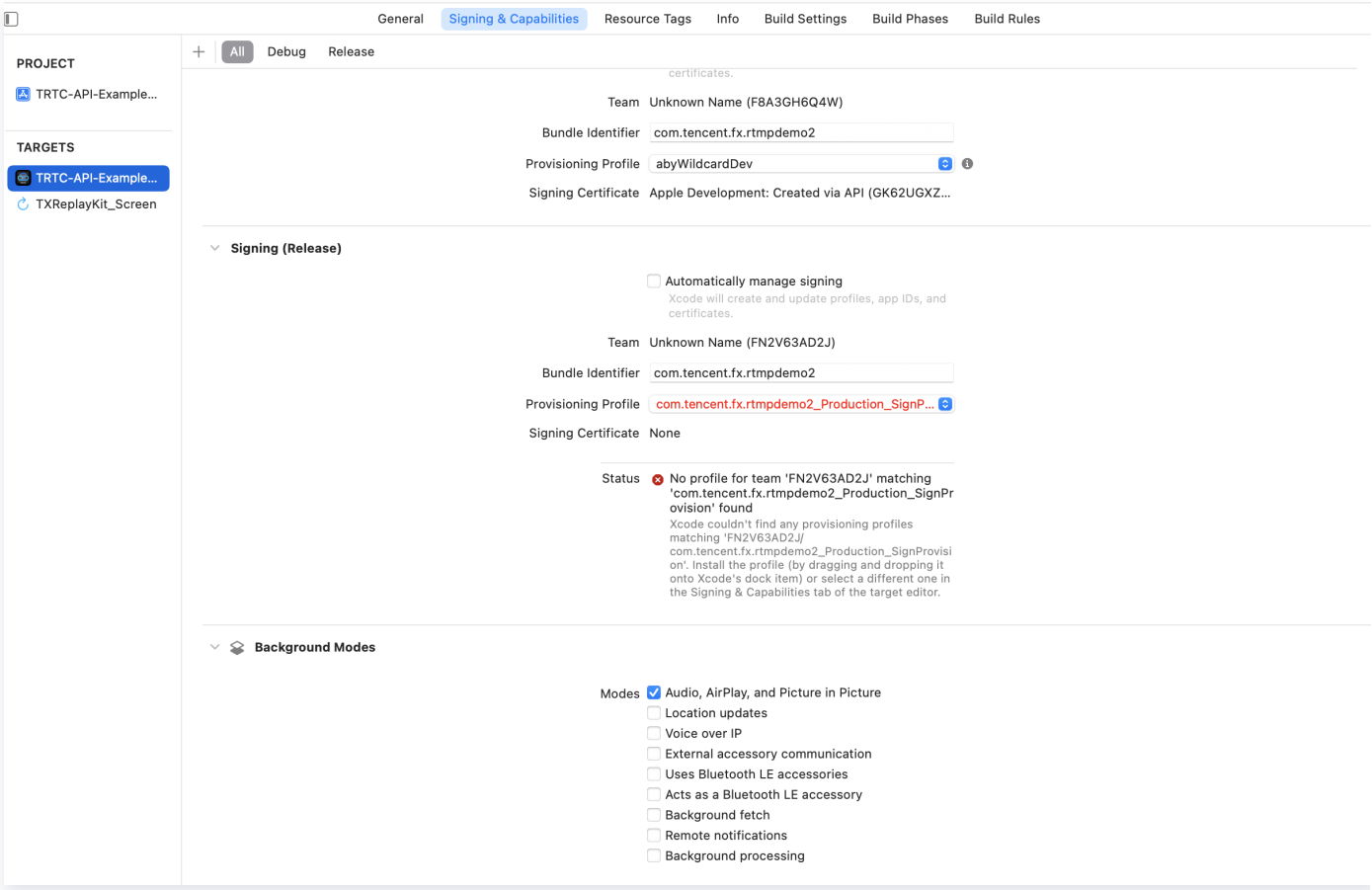
Step 3: Project configuration.

1. In karaoke scenarios, the TRTC SDK needs to be authorized for microphone permissions. Add the following content to your app's Info.plist. It corresponds to the system's prompt message when microphone permission is requested.

```
Privacy - Microphone Usage Description. Also enter a prompt specifying
the purpose of microphone use.
```



2. If you need your App to continue running certain features in the background, go to XCode. Choose your current project. Under Capabilities, set the settings for Background Modes to ON, and check Audio, AirPlay, and Picture in Picture, as shown below:



Step 4: Authentication and authorization.

UserSig is a security protection signature designed by Tencent Cloud to prevent malicious attackers from misappropriating your cloud service usage rights. TRTC validates this authentication credential when it enters the room.

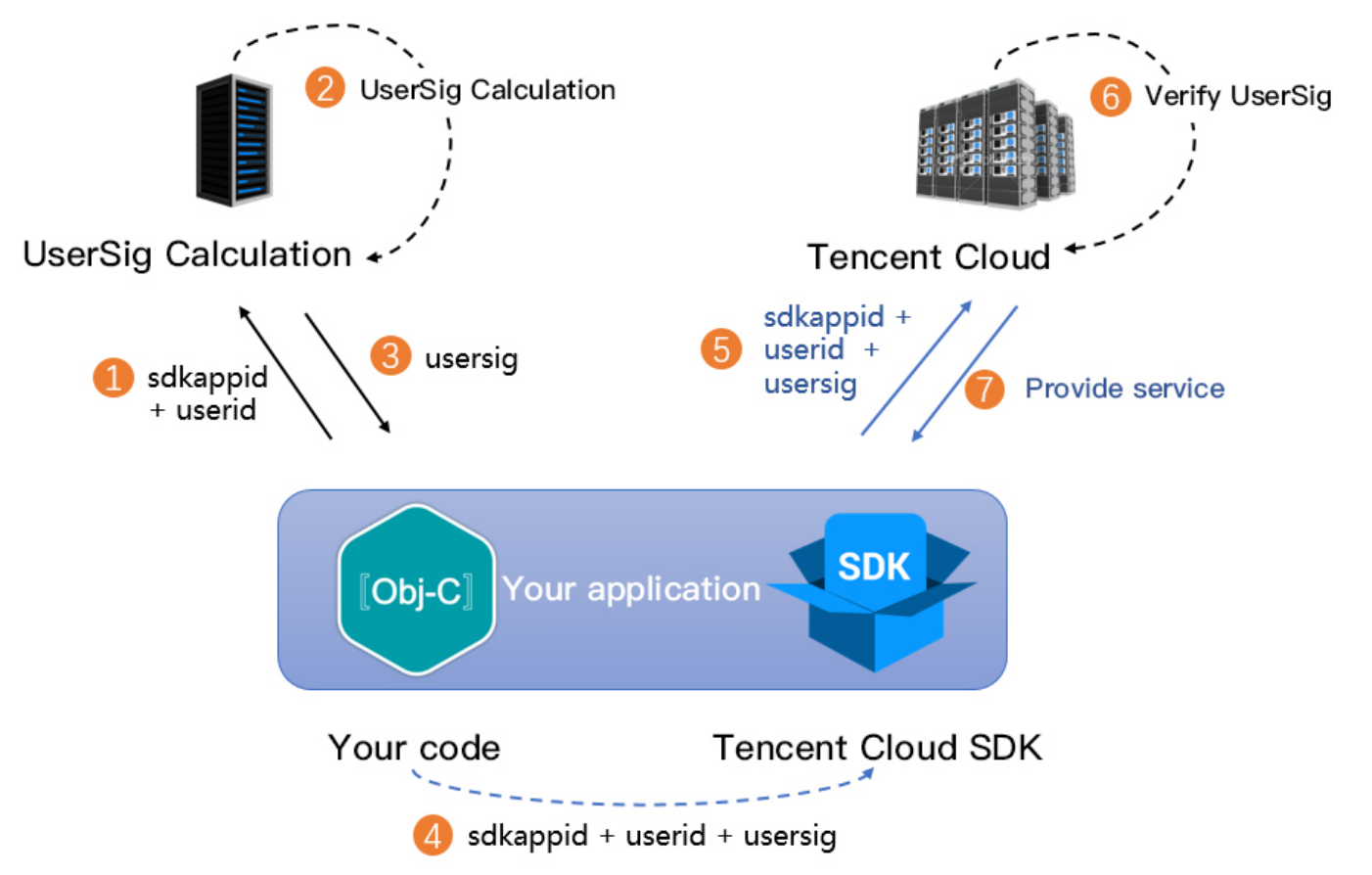
Debugging Stage: UserSig can be generated through two methods for debugging and testing purposes only: client sample code and console access.

Formal Operation Stage: It is recommended to use a higher security level server computation for generating UserSig. This is to prevent key leakage due to client reverse engineering.

The specific implementation process is as follows:

1. Before calling the SDK's initialization function, your app must first request UserSig from your server.
2. Your server computes the UserSig based on the SDKAppID and UserID.

3. The server returns the computed UserSig to your app.
4. Your app passes the obtained UserSig into the SDK through a specific API.
5. The SDK submits the SDKAppID + UserID + UserSig to Tencent Cloud CVM for verification.
6. Tencent Cloud verifies the UserSig and confirms its validity.
7. After the verification is passed, real-time audio and video services will be provided to the TRTC SDK.



> **Note:**
> - The local computation method of UserSig during the debugging stage is not recommended for application in an online environment. It is prone to reverse engineering, leading to key leakage.
> - We provide server computation source code for UserSig in multiple programming languages (Java/GO/PHP/Nodejs/Python/C#/C++). For details, see Server Computation of UserSig.

Step 5: Initializing the SDK.

```
// Create TRTC SDK instance (Single Instance Pattern).
```

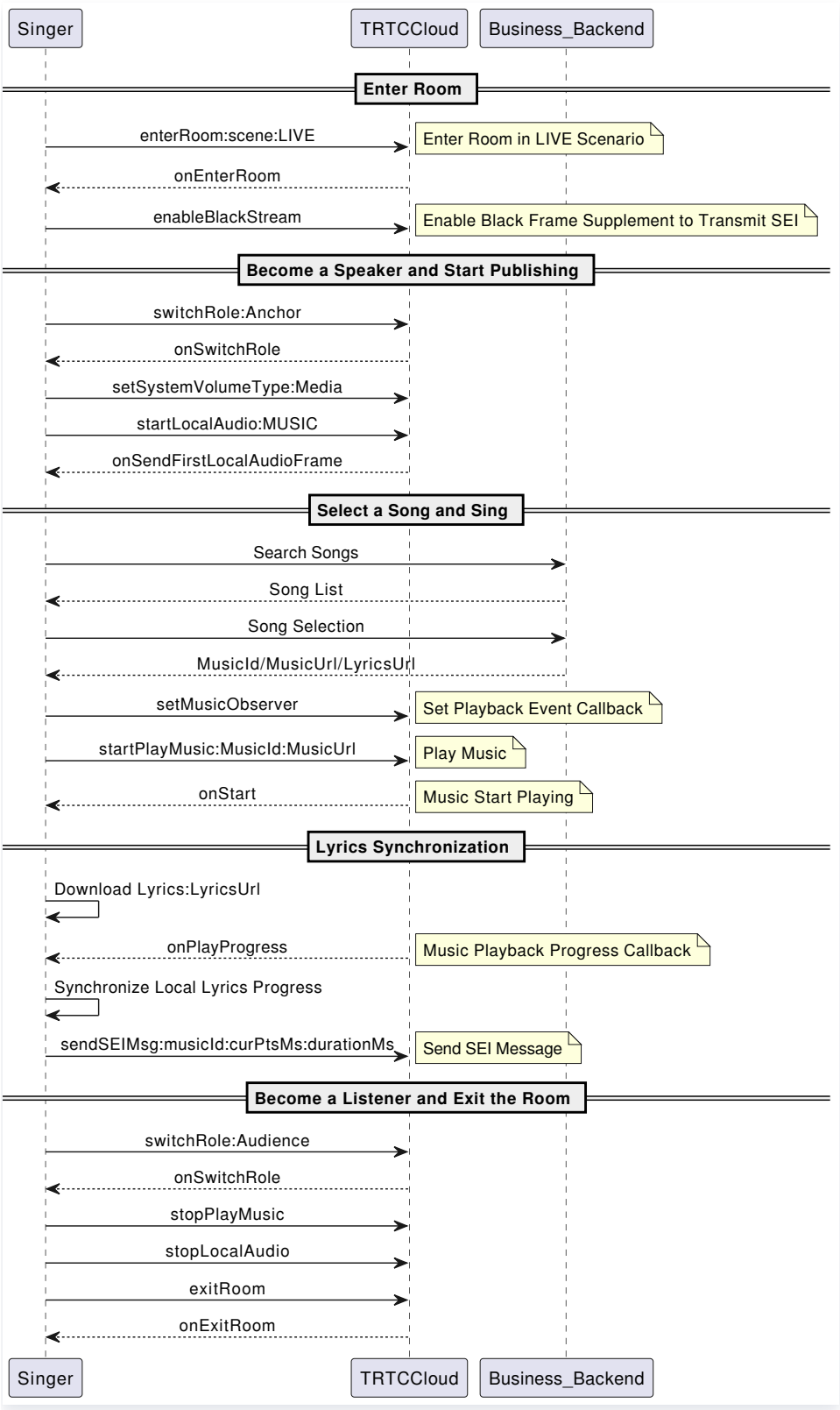
```
self.trtcCloud = [TRTCCloud sharedInstance];
// Set event listeners.
self.trtcCloud.delegate = self;

// Notifications from various SDK events (e.g., error codes, warning
codes, audio and video status parameters, etc.).
- (void)onError:(TXLiteAVError)errCode errMsg:(nullable NSString
*)errMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", errCode, errMsg);
}

- (void)onWarning:(TXLiteAVWarning)warningCode warningMsg:(nullable
NSString *)warningMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", warningCode, warningMsg);
}

// Remove event listener.
self.trtcCloud.delegate = nil;
// Terminate TRTC SDK instance (Singleton Pattern).
[TRTCCloud destroySharedIntance];
```

Note:
It is recommended to listen to SDK event notifications. Perform log printing and handling for some common errors. For details, see Error Code Table.

Scenario 1: Solo singing turn-taking

Perspective 1: Performer actions

Sequence diagram



1. Enter the room.

```
- (void)enterRoomWithRoomId:(NSString *)roomId userID:(NSString *)userId
{
    TRTCParams *params = [[TRTCParams alloc] init];
```

```
    // Take the room ID string as an example.
    params.strRoomId = roomId;
    params.userId = userId;
    // UserSig obtained from the business backend.
    params.userSig = [self generateUserSig:userId];
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
    // It is recommended to enter the room as an audience role.
    params.role = TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}
```

Note:

To better transmit SEI messages for lyric synchronization, it is recommended to choose `TRTCAppSceneLIVE` for room entry scenarios.

```
// Event callback for the result of entering the room.
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        NSLog(@"Enter room succeed!");
        // Enable the experimental API for black frame insertion.
        [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"enableBlackStream\",\"params\": {\"enable\":true}}"];
    } else {
        // result indicates the error code when you fail to enter the
room.
        NSLog(@"Enter room failed!");
    }
}
```

Note:

Under the pure audio mode, the performer needs to enable the insertion of black frames to carry SEI messages. This API should be called after successfully entering the room.

2. Go live on streams.

```
// Switched to the anchor role.
[self.trtcCloud switchRole:TRTCRoleAnchor];

// Event callback for switching the role.
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(NSString *)errMsg {
    if (errCode == ERR_NULL) {
        // Set media volume type.
        [self.trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
        // Upstream local audio streams and set audio quality.
        [self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
    }
}
```

> **Note:**
> In karaoke scenarios, it is recommended to set the full-range media volume and music quality to achieve a high-fidelity listening experience.

3. Song selection and performance.

- Search for songs, and obtain music resources.

  Search for songs and acquire music resources through the business backend. Obtain identifiers such as the MusicId, the song's URL (MusicUrl), and the lyrics URL (LyricsUrl).
  It is recommended that the business side select an appropriate music repository production to provide licensed music resources.
- Play accompaniment and start singing.

```
// Obtain audio effects management.
self.audioEffectManager = [self.trtcCloud getAudioEffectManager];

// originMusicId: Custom identifier for the original vocal music.
originMusicUrl: URL of the original vocal music resource.
TXAudioMusicParam *originMusicParam = [[TXAudioMusicParam alloc] init];
originMusicParam.ID = originMusicId;
originMusicParam.path = originMusicUrl;
// Whether to publish the original vocal music to remote (otherwise play
locally only).
originMusicParam.publish = YES;
```

```
// accompMusicId: Custom identifier for the accompaniment music.
accompMusicUrl: URL of the accompaniment music resource.
TXAudioMusicParam *accompMusicParam = [[TXAudioMusicParam alloc] init];
accompMusicParam.ID = accompMusicId;
accompMusicParam.path = accompMusicUrl;
// Whether to publish the accompaniment to remote (otherwise play
locally only).
accompMusicParam.publish = YES;

// Start playing the original vocal music.
[self.audioEffectManager startPlayMusic:originMusicParam
onStart:^(NSInteger errCode) {
    // onStart
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // onProgress
} onComplete:^(NSInteger errCode) {
    // onComplete
}];

// Start playing the accompaniment music.
[self.audioEffectManager startPlayMusic:originMusicParam
onStart:^(NSInteger errCode) {
    // onStart
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // onProgress
} onComplete:^(NSInteger errCode) {
    // onComplete
}];

// Switch to the original vocal music.
[self.audioEffectManager setMusicPlayoutVolume:originMusicId
volume:100];
[self.audioEffectManager setMusicPublishVolume:originMusicId
volume:100];
[self.audioEffectManager setMusicPlayoutVolume:accompMusicId volume:0];
[self.audioEffectManager setMusicPublishVolume:accompMusicId volume:0];

// Switch to the accompaniment music.
[self.audioEffectManager setMusicPlayoutVolume:originMusicId volume:0];
```

```
[self.audioEffectManager setMusicPublishVolume:originMusicId volume:0];
[self.audioEffectManager setMusicPlayoutVolume:accompMusicId
volume:100];
[self.audioEffectManager setMusicPublishVolume:accompMusicId
volume:100];
```

Note:

- In karaoke scenarios, both the original vocal and accompaniment need to be played simultaneously (distinguished by MusicID). The switch between the original vocal and accompaniment is achieved by adjusting the local and remote playback volumes.
- If the music being played has dual audio tracks (including both the original vocal and accompaniment), switching between them can be achieved by specifying the music's playback track using setMusicTrack.

4. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Synchronize local lyrics, and transmit song progress via SEI.

```
[self.audioEffectManager startPlayMusic:musicParam onStart:^(NSInteger
errCode) {
    // Start playing music.
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // Determine whether seek is needed based on the latest progress and
the local lyrics progress deviation.
    // Song progress is transmitted by sending an SEI message.
    NSDictionary *dic = @{
        @"musicId": @(self.musicId),
        @"progress": @(progressMs),
        @"duration": @(durationMs),
    };
    JSONModel *json = [[JSONModel alloc] initWithDictionary:dic
error:nil];
    [self.trtcCloud sendSEIMsg:json.toJSONData repeatCount:1];
} onComplete:^(NSInteger errCode) {
    // Music playback completed.
}];
```

> **Note:**
> The frequency of the SEI messages sent by the performer is determined by the event callback frequency. Also, the playback progress can be actively synchronized on a schedule through getMusicCurrentPosInMS.

5. Become a listener and exit the room.

```
// Switched to the audience role.
[self.trtcCloud switchRole:TRTCRoleAudience];

// Event callback for switching the role.
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(NSString *)errMsg {
    if (errCode == ERR_NULL) {
        // Stop playing accompaniment music.
        [[self.trtcCloud getAudioEffectManager]
stopPlayMusic:self.musicId];
        // Stop local audio capture and publishing.
        [self.trtcCloud stopLocalAudio];
    }
}

// Exit the room.
[self.trtcCloud exitRoom];

// Exit room event callback.
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"Proactively call exitRoom to exit the room.");
    } else if (reason == 1) {
        NSLog(@"Removed from the current room by the server.");
    } else if (reason == 2) {
        NSLog(@"The current room is dissolved.");
    }
}
```

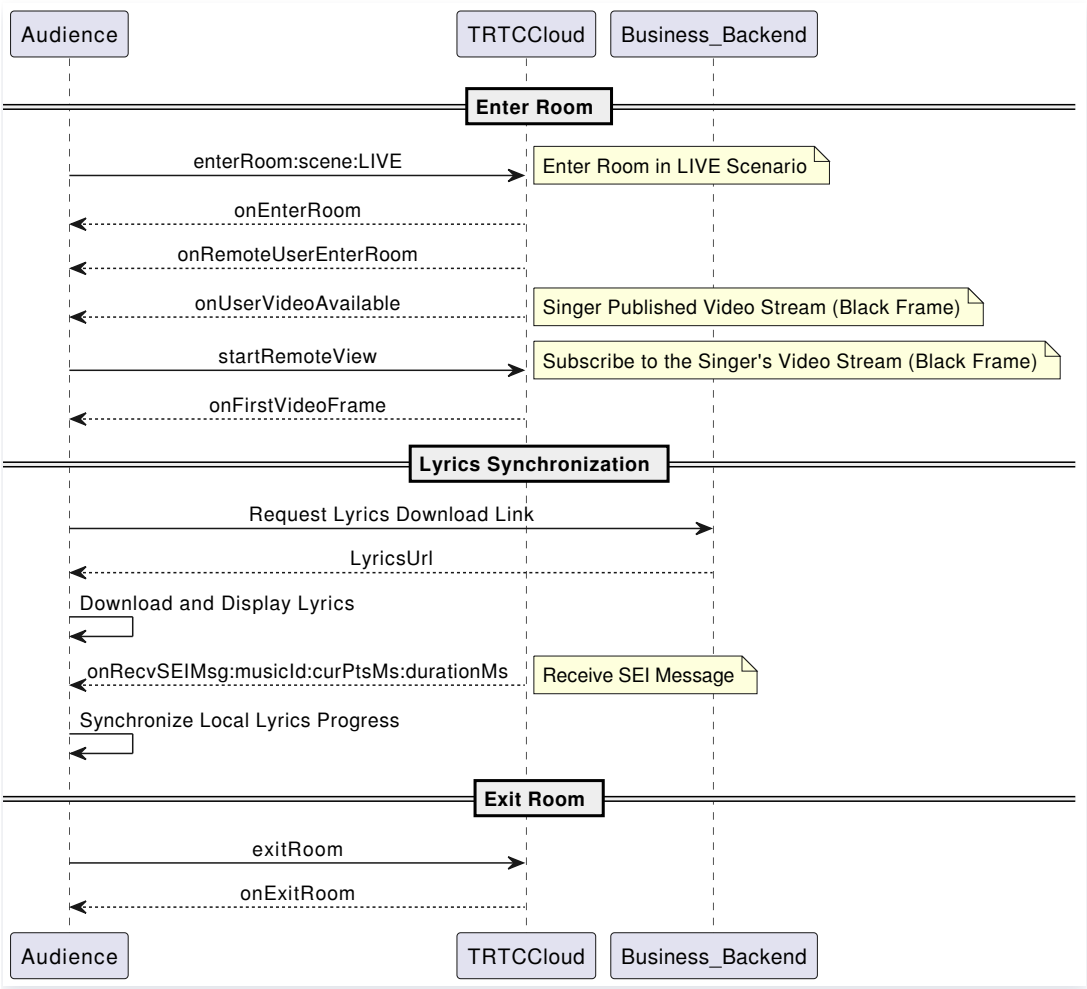
> **Note:**
> - After all resources occupied by the SDK are released, the SDK will throw the `onExitRoom`

callback notification to inform you.

- If you want to call `enterRoom` again or switch to another audio and video SDK, wait for the `onExitRoom` callback before proceeding. Otherwise, you may encounter various exceptional issues such as the camera, microphone device being forcibly occupied.

Perspective 2: Listener actions

Sequence diagram



1. Enter the room.

```
// Enter the room.
- (void)enterRoomWithRoomId:(NSString *)roomId userID:(NSString *)userId
{
    TRTCParams *params = [[TRTCParams alloc] init];
    // Take the room ID string as an example.
    params.strRoomId = roomId;
    params.userId = userId;
```

```
    // UserSig obtained from the business backend.
    params.userSig = [self generateUserSig:userId];
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
    // It is recommended to enter the room as an audience role.
    params.role = TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}

// Event callback for the result of entering the room.
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        NSLog(@"Enter room succeed!");
    } else {
        // result indicates the error code when you fail to enter the
room.
        NSLog(@"Enter room failed!");
    }
}
```

Note:

- To better transmit SEI messages for lyric synchronization, it is recommended to choose `TRTCAppSceneLIVE` for room entry scenarios.
- Under the automatic subscription mode (default), audiences automatically subscribe and play the on-mic anchor's audio and video streams upon entering the room.

2. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Listener end lyric synchronization

```
- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    if (available) {
```

```
        [self.trtcCloud startRemoteView:userId view:nil];
    } else {
        [self.trtcCloud stopRemoteView:userId];
    }
}

- (void)onRecvSEIMsg:(NSString *)userId message:(NSData *)message {
    JSONModel *json = [[JSONModel alloc] initWithData:message
error:nil];
    NSDictionary *dic = json.toDictionary;
    int32_t musicId = [dic[@"musicId"] intValue];
    NSInteger progress = [dic[@"progress"] integerValue];
    NSInteger duration = [dic[@"duration"] integerValue];
    // ......
    // TODO: The logic of updating the lyric control.
    // Based on the received latest progress and the local lyrics
progress deviation, determine whether a lyric control seek is necessary.
    // ......
}
```

Note:
Listeners need to actively subscribe to the performer's video streams in order to receive the SEI messages carried by black frames.

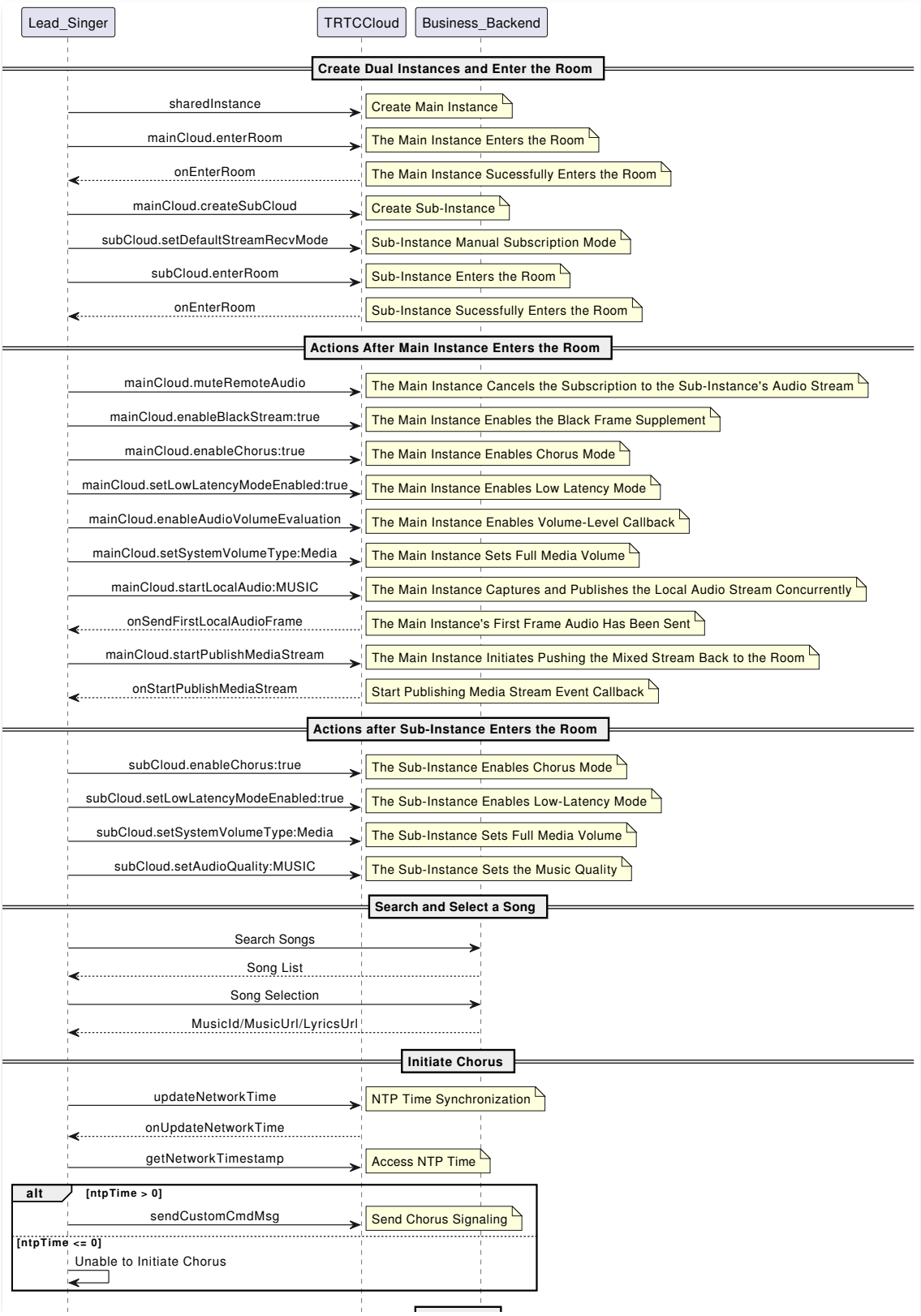
3. Exit the room.

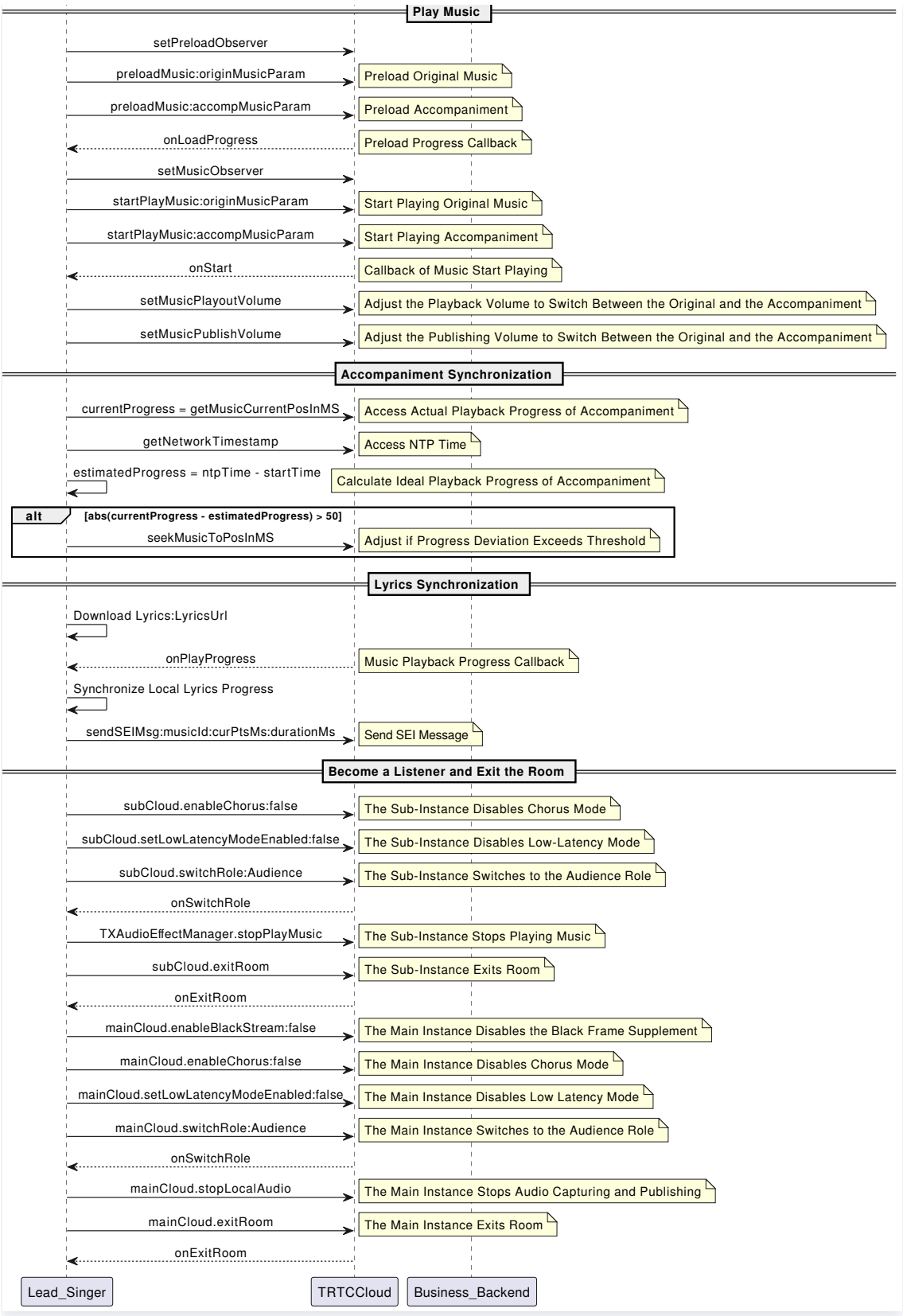
```
// Exit the room.
[self.trtcCloud exitRoom];

// Exit room event callback.
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"Proactively call exitRoom to exit the room.");
    } else if (reason == 1) {
        NSLog(@"Removed from the current room by the server.");
    } else if (reason == 2) {
        NSLog(@"The current room is dissolved.");
    }
}
```

Scenario 2: Real-time chorus

Perspective 1: Lead singer actions

Sequence diagram





1. Dual instances enter the room.

```
- (void)enterRoomWithRoomId:(NSString *)roomId userID:(NSString *)userId
{
    // Create a TRTCCloud primary instance (vocal instance).
```

```
    TRTCCloud *mainCloud = [TRTCCloud sharedInstance];
    // Create a TRTCCloud sub-instance (music instance).
    TRTCCloud *subCloud = [mainCloud createSubCloud];

    // The primary instance (vocal instance) enters the room.
    TRTCParams *params = [[TRTCParams alloc] init];
    params.strRoomId = roomId;
    params.userId = userId;
    params.userSig = userSig;
    params.sdkAppId = SDKAppID;
    params.role = TRTCRoleAnchor;
    [mainCloud enterRoom:params appScene:TRTCAppSceneLIVE];

    // The sub-instance enables manual subscription mode. By default it
does not subscribe to remote streams.
    [subCloud setDefaultStreamRecvMode:NO video:NO];
    // The sub-instance (music instance) enters the room.
    TRTCParams *bgmParams = [[TRTCParams alloc] init];
    bgmParams.strRoomId = roomId;
    // The sub-instance username must not duplicate with other users in
the room.
    bgmParams.userId = [userId stringByAppendingString:@"_bgm"];
    bgmParams.userSig = userSig;
    bgmParams.sdkAppId = SDKAppID;
    bgmParams.role = TRTCRoleAnchor;
    [subCloud enterRoom:bgmParams appScene:TRTCAppSceneLIVE];
}
```

Note:

- In a real-time chorus solution, the lead singer end must create primary instance and sub-instance for upstream voice and accompaniment music, respectively.
- Sub-instances do not need to subscribe to other users' audio streams in the room. Therefore, it is recommended to enable manual subscription mode, and it must be activated before entering the room.

2. Set the settings after entering the room.

```
// Event callback for the result of primary instance entering the room.
```

```
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // The primary instance unsubscribe from music streams published
by sub-instances.
        [self.trtcCloud muteRemoteAudio:[self.userId
stringByAppendingString:@"_bgm"] mute:YES];
        // The primary instance uses the experimental API to enable
black frame insertion.
        [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"enableBlackStream\",\"params\": {\"enable\":true}}"];
        // The primary instance uses the experimental API to enable
chorus mode.
        [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"enableChorus\",\"params\":
{\"enable\":true,\"audioSource\":0}}"];
        // The primary instance uses the experimental API to enable low-
latency mode.
        [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":true}}"];
        // The primary instance enables volume level callback.
        TRTCAudioVolumeEvaluateParams *aveParams =
[[TRTCAudioVolumeEvaluateParams alloc] init];
        aveParams.interval = 300;
        [self.trtcCloud enableAudioVolumeEvaluation:YES
withParams:aveParams];
        // The primary instance sets the global media volume type.
        [self.trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
        // The primary instance captures and publishes local audio, and
sets audio quality.
        [self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
    } else {
        // result indicates the error code when you fail to enter the
room.
        NSLog(@"Enter room failed");
    }
}

// Event callback for the result of sub-instance entering the room.
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
```

```
        // The sub-instance uses the experimental API to enable chorus
mode.
        [self.subCloud callExperimentalAPI:@"
{\"api\":\"enableChorus\",\"params\":
{\"enable\":true,\"audioSource\":1}}"];
        // The sub-instance uses the experimental API to enable low-
latency mode.
        [self.subCloud callExperimentalAPI:@"
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":true}}"];
        // The sub-instance sets global media volume type.
        [self.subCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
        // The sub-instance sets audio quality.
        [self.subCloud setAudioQuality:TRTCAudioQualityMusic];
    } else {
        // result indicates the error code when you fail to enter the
room.
        NSLog(@"Enter room failed");
    }
}
```

Note:

Both the primary instance and sub-instance must use the experimental APIs to enable chorus mode and low-latency mode to optimize the chorus experience. Note the difference in the `audioSource` parameter.

3. Push the mixed stream back to the room.

```
- (void)startPublishMediaToRoomWithRoomId:(NSString *)roomId userId:
(NSString *)userId {
    // Create TRTCPublishTarget object.
    TRTCPublishTarget *target = [[TRTCPublishTarget alloc] init];
    // After mixing, the stream is relayed back to the room.
    target.mode = TRTCPublishMixStreamToRoom;
    TRTCUser *mixStreamIdentity = [[TRTCUser alloc] init];
    mixStreamIdentity.strRoomId = roomId;
    // The mixing stream robot's username must not duplicate with other
users in the room.
    mixStreamIdentity.userId = [userId
stringByAppendingString:@"_robot"];
```

```
    target.mixStreamIdentity = mixStreamIdentity;

    // Set the encoding parameters of the transcoded audio stream (can
be customized).
    TRTCStreamEncoderParam *encoderParam = [[TRTCStreamEncoderParam
alloc] init];
    encoderParam.audioEncodedChannelNum = 2;
    encoderParam.audioEncodedKbps = 64;
    encoderParam.audioEncodedCodecType = 2;
    encoderParam.audioEncodedSampleRate = 48000;

    // Set the encoding parameters of the transcoded video stream (black
frame mixing required).
    encoderParam.videoEncodedFPS = 15;
    encoderParam.videoEncodedGOP = 3;
    encoderParam.videoEncodedKbps = 30;
    encoderParam.videoEncodedWidth = 64;
    encoderParam.videoEncodedHeight = 64;

    // Set audio mixing parameters.
    TRTCStreamMixingConfig *mixingConfig = [[TRTCStreamMixingConfig
alloc] init];
    // By default, leave this field empty. It indicates that all audio
in the room will be mixed.
    mixingConfig.audioMixUserList = nil;

    // Configure video mixed-stream template (black frame mixing
required).
    TRTCVideoLayout *layout = [[TRTCVideoLayout alloc] init];
    mixingConfig.videoLayoutList = @[layout];

    // Start mixing and pushing back.
    [self.trtcCloud startPublishMediaStream:target
encoderParam:encoderParam mixingConfig:mixingConfig];
}
```

Note:

- To maintain alignment between chorus vocals and accompaniment music, it is recommended to enable pushing the mixed stream back to the room. The on-mic chorus members mutually

subscribe to single streams, and off-mic audiences by default only subscribe to mixed streams.
- The mixing stream robot, acting as an independent user, enters the room to pull, mix, and push streams. Its username must not duplicate with other usernames in the room. Otherwise, it may lead to mutual deletion from the room.

4. Search for and request songs.

   Search for songs and acquire music resources through the business backend. Obtain identifiers such as the MusicId, the song's URL (MusicUrl), and the lyrics URL (LyricsUrl).
   It is recommended that the business side select an appropriate music repository production to provide licensed music resources.
5. NTP synchronization.

```
- (void)updateNetworkTimeExample {
    [TXLiveBase sharedInstance].delegate = self;
    [TXLiveBase updateNetworkTime];
}

- (void)onUpdateNetworkTime:(int)errCode message:(NSString *)errMsg {
    // errCode 0: Time synchronization successful and deviation within
30 ms. 1: Time synchronization successful but deviation possibly above
30 ms. -1: Time synchronization failed.
    if (errCode == 0) {
        // Time synchronization successful and NTP timestamp obtained.
        NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    } else {
        NSLog(@"Time synchronization failed, and you can try re-
synchronization.");
    }
}
```

Note:
NTP time synchronization results can reflect the current network quality of the application user. To ensure a good chorus experience, it is recommended not to allow users to initiate chorus if time synchronization fails.

6. Send chorus signaling.

```
- (void)sendChorusSignalExample {
```

```
    __weak typeof(self) weakSelf = self;
    NSTimer *timer = [NSTimer timerWithTimeInterval:1.0 repeats:YES
block:^(NSTimer * _Nonnull timer) {
        __strong typeof(weakSelf) strongSelf = weakSelf;
        NSDictionary *dic = @{
            @"cmd": @"startChorus",
            // Agreed chorus start time: Current NTP time + delayed
playback time (for example, 3 seconds).
            @"startPlayMusicTS": @([TXLiveBase getNetworkTimestamp] +
3000),
            @"musicId": @(self.musicId),
            @"musicDuration": @([[strongSelf.subCloud
getAudioEffectManager] getMusicDurationInMS:strongSelf.originMusicUri]),
        };
        JSONModel *json = [[JSONModel alloc] initWithDictionary:dic
error:nil];
        [strongSelf.trtcCloud sendCustomCmdMsg:1 data:json.toJSONData
reliable:NO ordered:NO];
    }];
    [[NSRunLoop currentRunLoop] addTimer:timer
forMode:NSRunLoopCommonModes];
}
```

Note:

The lead singer needs to cyclically broadcast chorus signaling to the room at a fixed time interval (e.g., every 1 second), so that new users who join mid-session can also participate in the chorus.

7. Load and play accompaniment.

```
// Obtain audio effects management.
TXAudioEffectManager *audioEffectManager = [self.subCloud
getAudioEffectManager];

// originMusicId: Custom identifier for the original vocal music.
originMusicUrl: URL of the original vocal music resource.
TXAudioMusicParam *originMusicParam = [[TXAudioMusicParam alloc] init];
originMusicParam.ID = originMusicId;
originMusicParam.path = originMusicUrl;
```

```
// Whether to publish the original vocal music to remote (otherwise play
locally only).
originMusicParam.publish = YES;
// Music start playing time point (in milliseconds).
originMusicParam.startTimeMS = 0;

// accompMusicId: Custom identifier for the accompaniment music.
accompMusicUrl: URL of the accompaniment music resource.
TXAudioMusicParam *accompMusicParam = [[TXAudioMusicParam alloc] init];
accompMusicParam.ID = accompMusicId;
accompMusicParam.path = accompMusicUrl;
// Whether to publish the accompaniment to remote (otherwise play
locally only).
accompMusicParam.publish = YES;
// Music start playing time point (in milliseconds).
accompMusicParam.startTimeMS = 0;

// Preload the original vocal music.
[audioEffectManager preloadMusic:originMusicParam onProgress:nil
onError:nil];
// Preload the accompaniment music.
[audioEffectManager preloadMusic:accompMusicParam onProgress:nil
onError:nil];

// Start playing the original vocal music after a delayed playback time
(for example, 3 seconds).
[self.audioEffectManager startPlayMusic:originMusicParam
onStart:^(NSInteger errCode) {
    // onStart
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // onProgress
} onComplete:^(NSInteger errCode) {
    // onComplete
}];

// Start playing the accompaniment music after a delayed playback time
(for example, 3 seconds).
[self.audioEffectManager startPlayMusic:originMusicParam
onStart:^(NSInteger errCode) {
    // onStart
```

```
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // onProgress
} onComplete:^(NSInteger errCode) {
    // onComplete
}];

// Switch to the original vocal music.
[self.audioEffectManager setMusicPlayoutVolume:originMusicId
volume:100];
[self.audioEffectManager setMusicPublishVolume:originMusicId
volume:100];
[self.audioEffectManager setMusicPlayoutVolume:accompMusicId volume:0];
[self.audioEffectManager setMusicPublishVolume:accompMusicId volume:0];

// Switch to the accompaniment music.
[self.audioEffectManager setMusicPlayoutVolume:originMusicId volume:0];
[self.audioEffectManager setMusicPublishVolume:originMusicId volume:0];
[self.audioEffectManager setMusicPlayoutVolume:accompMusicId
volume:100];
[self.audioEffectManager setMusicPublishVolume:accompMusicId
volume:100];
```

Note:

- It is recommended to preload music before starting playback. By loading music resources into memory in advance, you can effectively reduce the load delay of music playback.
- In karaoke scenarios, both the original vocal and accompaniment need to be played simultaneously (distinguished by MusicID). The switch between the original vocal and accompaniment is achieved by adjusting the local and remote playback volumes.
- If the music being played has dual audio tracks (including both the original vocal and accompaniment), switching between them can be achieved by specifying the music's playback track using setMusicTrack.

8. Accompaniment Synchronization

```
// Agreed chorus start time.
@property (nonatomic, assign) NSInteger startPlayMusicTS;

- (void)syncBgmExample {
```

```
    // Actual playback progress of the current accompaniment music.
    NSInteger currentProgress = [[self.subCloud getAudioEffectManager]
getMusicCurrentPosInMS:self.musicId];
    // Ideal playback progress of the current accompaniment music.
    NSInteger estimatedProgress = [TXLiveBase getNetworkTimestamp] -
self.startPlayMusicTS;
    // When the progress difference exceeds 50 ms, corrections are made.
    if (estimatedProgress >= 0 && labs(currentProgress -
estimatedProgress) > 50) {
        [[self.subCloud getAudioEffectManager]
seekMusicToPosInMS:self.musicId pts:estimatedProgress];
    }
}
```

9. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Synchronize local lyrics, and transmit song progress via SEI.

```
[[self.subCloud getAudioEffectManager] startPlayMusic:musicParam
onStart:^(NSInteger errCode) {
    // Start playing music.
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // Determine whether seek is needed based on the latest progress and
the local lyrics progress deviation.
    // Song progress is transmitted by sending an SEI message.
    NSDictionary *dic = @{
        @"musicId": @(self.musicId),
        @"progress": @(progressMs),
        @"duration": @(durationMs),
    };
    JSONModel *json = [[JSONModel alloc] initWithDictionary:dic
error:nil];
    [self.trtcCloud sendSEIMsg:json.toJSONData repeatCount:1];
} onComplete:^(NSInteger errCode) {
    // Music playback completed.
}];
```

> **Note:**
> The frequency of the SEI messages sent by the performer is determined by the event callback frequency. Also, the playback progress can be actively synchronized on a schedule through getMusicCurrentPosInMS.

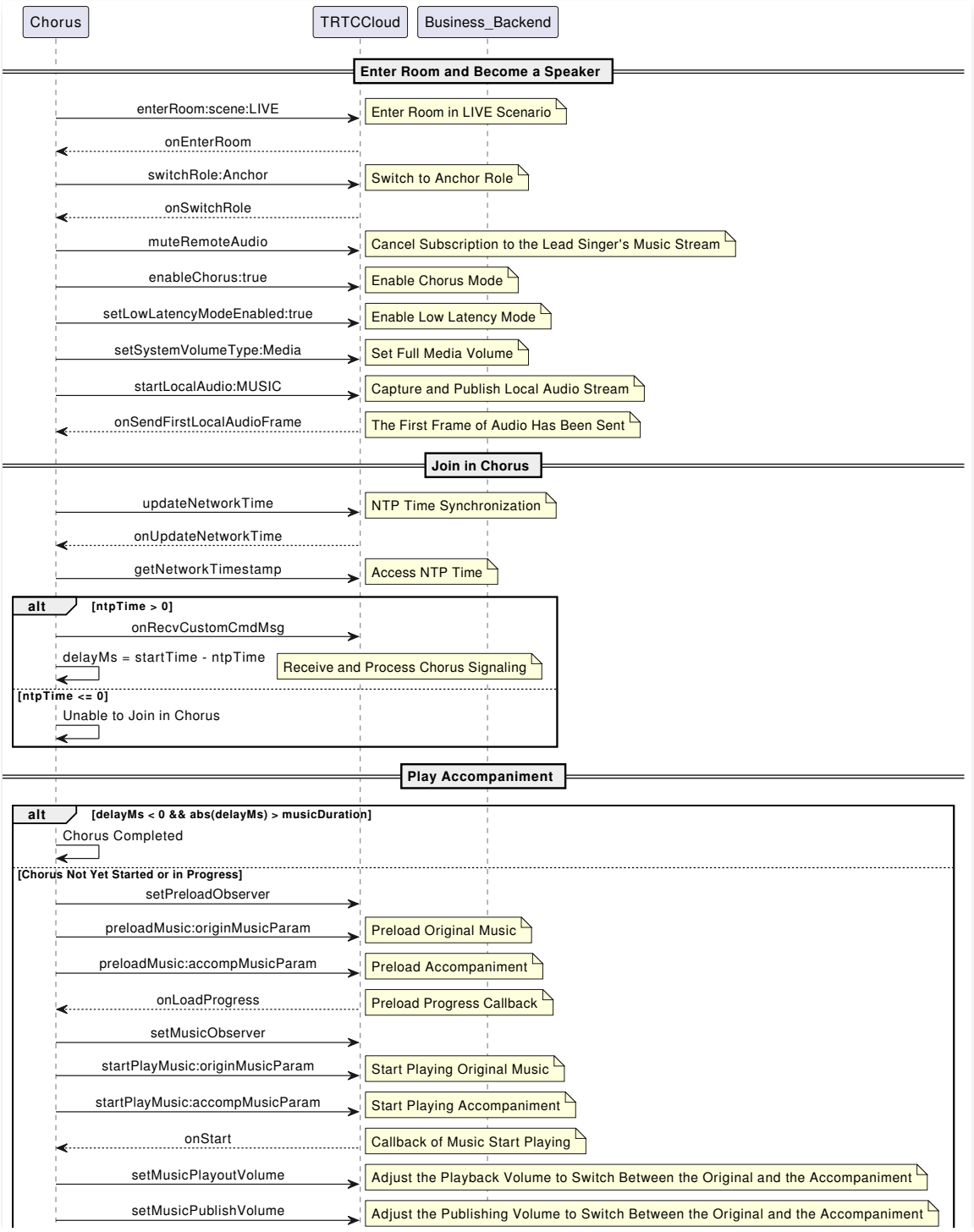
10. Become a listener and exit the room.

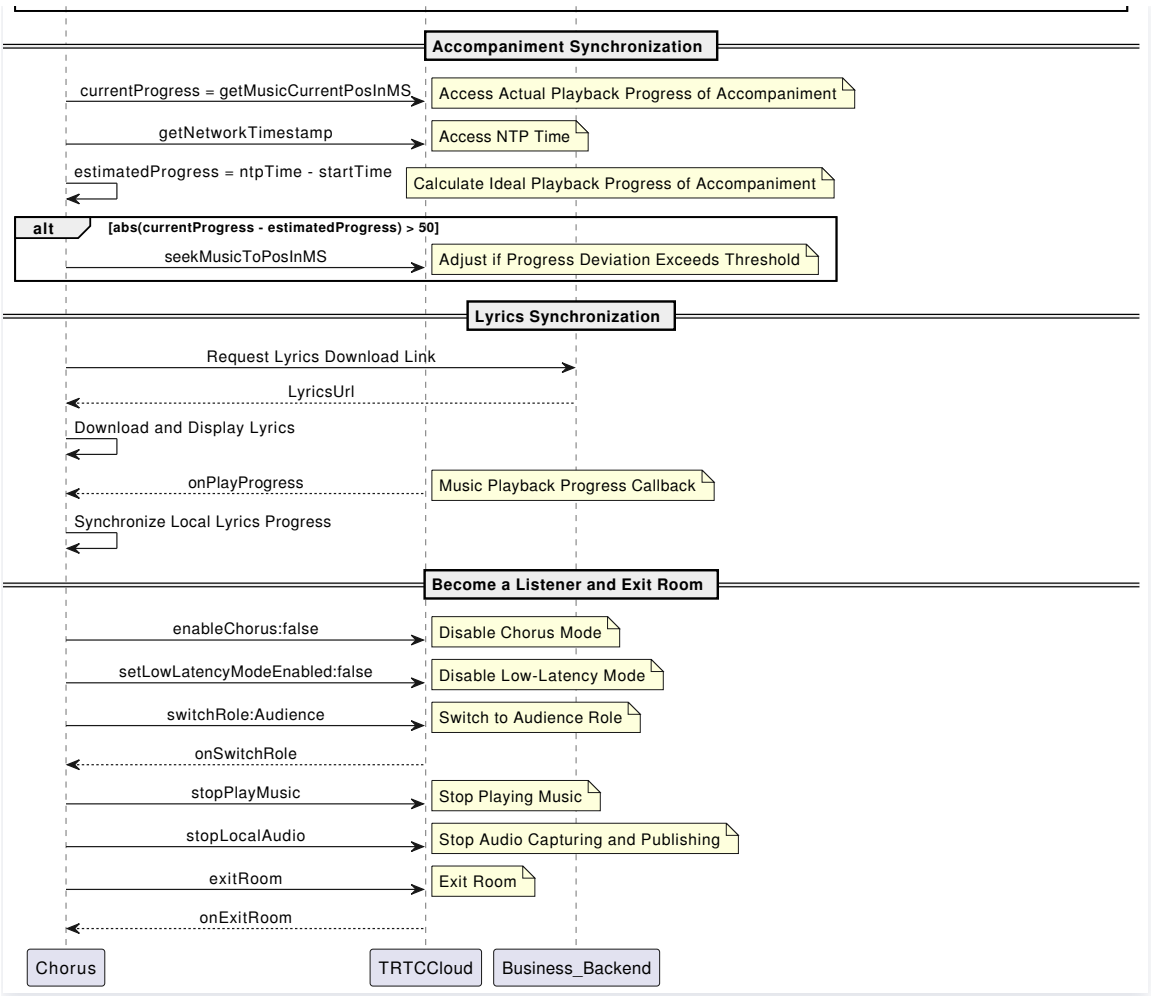
```
- (void)exitRoomExample {
    // The sub-instance uses the experimental API to disable chorus
mode.
    [self.subCloud callExperimentalAPI:@"
{\"api\":\"enableChorus\",\"params\":
{\"enable\":false,\"audioSource\":1}}"];
    // The sub-instance uses the experimental API to disable low-latency
mode.
    [self.subCloud callExperimentalAPI:@"
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":false}}"];
    // The sub-instance switches to the audience role.
    [self.subCloud switchRole:TRTCRoleAudience];
    // The sub-instance stops playing accompaniment music.
    [[self.subCloud getAudioEffectManager] stopPlayMusic:self.musicId];
    // The sub-instance exits the room.
    [self.subCloud exitRoom];

    // The primary instance uses the experimental API to disable black
frame insertion.
    [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"enableBlackStream\",\"params\": {\"enable\":false}}"];
    // The primary instance uses the experimental API to disable chorus
mode.
    [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"enableChorus\",\"params\":
{\"enable\":false,\"audioSource\":0}}"];
    // The primary instance uses the experimental API to disable low-
latency mode.
    [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":false}}"];
    // The primary instance switches to the audience role.
    [self.trtcCloud switchRole:TRTCRoleAudience];
```

```
    // The primary instance stops local audio capture and publishing.
    [self.trtcCloud stopLocalAudio];
    // The primary instance exits the room.
    [self.trtcCloud exitRoom];
}
```

Perspective 2: Chorus actions

Sequence diagram





1. Enter the room.

```
- (void)enterRoomWithRoomId:(NSString *)roomId userID:(NSString *)userId
{
    TRTCParams *params = [[TRTCParams alloc] init];
    // Take the room ID string as an example.
    params.strRoomId = roomId;
    params.userId = userId;
    // UserSig obtained from the business backend.
    params.userSig = [self generateUserSig:userId];
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
    // Example of entering the room as an audience role.
    params.role = TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}
```

```
// Event callback for the result of entering the room.
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        NSLog(@"Enter room succeed!");
    } else {
        // result indicates the error code when you fail to enter the
room.
        NSLog(@"Enter room failed!");
    }
}
```

2. Go live on streams.

```
// Switched to the anchor role.
[self.trtcCloud switchRole:TRTCRoleAnchor];

// Event callback for switching the role.
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(NSString *)errMsg {
    if (errCode == ERR_NULL) {
        // Cancel subscription to music streams published by the lead
singer sub-instance.
        [self.trtcCloud muteRemoteAudio:self.bgmUserId mute:YES];
        // Use the experimental API to enable chorus mode.
        [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"enableChorus\",\"params\":
{\"enable\":true,\"audioSource\":0}}"];
        // Use the experimental API to enable low-latency mode.
        [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":true}}"];
        // Set media volume type.
        [self.trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
        // Upstream local audio streams and set audio quality.
        [self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
    }
}
```

> ⚠ Note:

- To minimize delay, all chorus members play the accompaniment music locally. Therefore, it is necessary to cancel subscriptions to music streams published by the lead singer.
- Chorus members also need to use the experimental API to enable chorus mode and low-latency mode to optimize the chorus experience.
- In karaoke scenarios, it is recommended to set the full-range media volume and music quality to achieve a high-fidelity listening experience.

3. NTP synchronization.

```
- (void)updateNetworkTimeExample {
    [TXLiveBase sharedInstance].delegate = self;
    [TXLiveBase updateNetworkTime];
}

- (void)onUpdateNetworkTime:(int)errCode message:(NSString *)errMsg {
    // errCode 0: Time synchronization successful and deviation within
30 ms. 1: Time synchronization successful but deviation possibly above
30 ms. -1: Time synchronization failed.
    if (errCode == 0) {
        // Time synchronization successful and NTP timestamp obtained.
        NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    } else {
        NSLog(@"Time synchronization failed, and you can try re-
synchronization.");
    }
}
```

Note:
NTP time synchronization results can reflect the current network quality of the application user. To ensure a good chorus experience, it is recommended not to allow users to participate in the chorus if time synchronization fails.

4. Receive chorus signaling.

```
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:
(NSInteger)cmdID seq:(UInt32)seq message:(NSData *)message {
    JSONModel *json = [[JSONModel alloc] initWithData:message
error:nil];
```

```
    NSDictionary *dic = json.toDictionary;
    // Match the chorus signaling.
    if ([dic[@"cmd"] isEqualToString:@"startChorus"]) {
        self.startPlayMusicTS = [dic[@"startPlayMusicTS"] integerValue];
        self.musicId = [dic[@"musicId"] intValue];
        self.musicDuration = [dic[@"musicDuration"] intValue];
        // Agree on the time difference between chorus time and current
time.
        self.delayMs = self.startPlayMusicTS - [TXLiveBase
getNetworkTimestamp];
    }
}
```

Note:

Once the chorus members receive the chorus signaling and join in, the status should be changed to Chorus In Progress. Chorus signaling would not be responded to again before the end of this chorus round.

5. Play accompaniment, and start chorus.

```
- (void)playBmgExample {
    // Chorus has not started.
    if (self.delayMs > 0) {
        // Begin to preload music.
        [self preloadMusicWithStartTimeMS:0];
        // Play music after a delay of delayMs.
        [self startPlayMusicWithStartTimeMS:0];
    } else if (labs(self.delayMs) < self.musicDuration) {
        // Chorus is in progress.
        // Play start time: Absolute value of the time difference +
preload delay (e.g., 400 ms).
        NSInteger startTimeMS = labs(self.delayMs) + 400;
        // Begin to preload music.
        [self preloadMusicWithStartTimeMS:startTimeMS];
        // Start playing music after a preload delay (e.g., 400 ms).
        [self startPlayMusicWithStartTimeMS:startTimeMS];
    } else {
        // Chorus has ended.
        // Joining the chorus is not allowed.
```

```
    }
}

// Preload music.
- (void)preloadMusicWithStartTimeMS:(NSInteger)startTimeMS {
    // musicId: Obtained from chorus signaling. musicUrl: Corresponding
music resource URL.
    TXAudioMusicParam *musicParam = [[TXAudioMusicParam alloc] init];
    musicParam.ID = self.musicId;
    musicParam.path = self.musicUrl;
    // Only local music playback.
    musicParam.publish = NO;
    musicParam.startTimeMS = startTimeMS;
    [self.audioEffectManager preloadMusic:musicParam onProgress:nil
onError:nil];
}

// Begin to play music.
- (void)startPlayMusicWithStartTimeMS:(NSInteger)startTimeMS {
    // musicId: Obtained from chorus signaling. musicUrl: Corresponding
music resource URL.
    TXAudioMusicParam *musicParam = [[TXAudioMusicParam alloc] init];
    musicParam.ID = self.musicId;
    musicParam.path = self.musicUrl;
    // Only local music playback.
    musicParam.publish = NO;
    musicParam.startTimeMS = startTimeMS;
    [self.audioEffectManager startPlayMusic:musicParam onStart:nil
onProgress:nil onComplete:nil];
}
```

Note:

- To minimize transmission delay as much as possible, chorus members perform along with the local playback of accompaniment music, and they do not need to publish or receive remote music.
- Based on `delayMs`, the current chorus status can be determined. Developers must implement the `startPlayMusic` delayed call for different statuses on their own.

6. Accompaniment Synchronization

```
// Agreed chorus start time.
@property (nonatomic, assign) NSInteger startPlayMusicTS;

- (void)syncBgmExample {
    // Actual playback progress of the current accompaniment music.
    NSInteger currentProgress = [[self.trtcCloud getAudioEffectManager]
getMusicCurrentPosInMS:self.musicId];
    // Ideal playback progress of the current accompaniment music.
    NSInteger estimatedProgress = [TXLiveBase getNetworkTimestamp] -
self.startPlayMusicTS;
    // When the progress difference exceeds 50 ms, corrections are made.
    if (estimatedProgress >= 0 && labs(currentProgress -
estimatedProgress) > 50) {
        [[self.trtcCloud getAudioEffectManager]
seekMusicToPosInMS:self.musicId pts:estimatedProgress];
    }
}
```

7. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Local lyric synchronization.

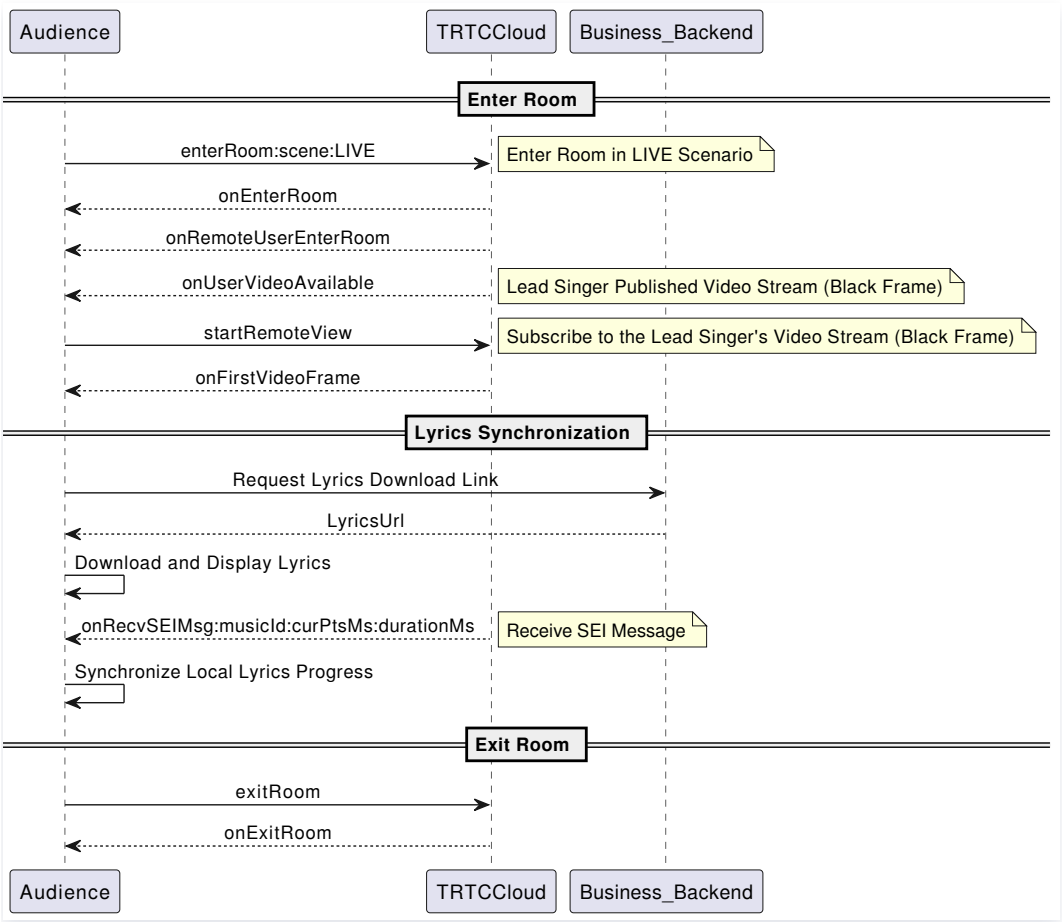
```
[self.audioEffectManager startPlayMusic:musicParam onStart:^(NSInteger
errCode) {
    // Start playing music.
} onProgress:^(NSInteger progressMs, NSInteger durationMs) {
    // TODO: The logic of updating the lyric control.
    // Determine whether seek in the lyrics control is needed based on
the latest progress and the local lyrics progress deviation.
} onComplete:^(NSInteger errCode) {
    // Music playback completed.
}];
```

8. Become a listener and exit the room.

```
- (void)exitRoomExample {
    // Use the experimental API to disable chorus mode.
    [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"enableChorus\",\"params\":
{\"enable\":false,\"audioSource\":0}}"];
    // Use the experimental API to disable low-latency mode.
    [self.trtcCloud callExperimentalAPI:@"
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":false}}"];
    // Switched to the audience role.
    [self.trtcCloud switchRole:TRTCRoleAudience];
    // Stop playing accompaniment music.
    [[self.trtcCloud getAudioEffectManager] stopPlayMusic:self.musicId];
    // Stop local audio capture and publishing.
    [self.trtcCloud stopLocalAudio];
    // Exit the room.
    [self.trtcCloud exitRoom];
}
```

Perspective 3: Listener actions

Sequence diagram



1. Enter the room.

```
- (void)enterRoomWithRoomId:(NSString *)roomId userID:(NSString *)userId
{
    TRTCParams *params = [[TRTCParams alloc] init];
    // Take the room ID string as an example.
    params.strRoomId = roomId;
    params.userId = userId;
    // UserSig obtained from the business backend.
    params.userSig = [self generateUserSig:userId];
    // Replace with your SDKAppID.
    params.sdkAppId = SDKAppID;
    // It is recommended to enter the room as an audience role.
    params.role = TRTCRoleAudience;
    // LIVE should be selected for the room entry scenario.
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}

// Event callback for the result of entering the room.
```

```
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result indicates the time taken (in milliseconds) to join the
room.
        NSLog(@"Enter room succeed!");
    } else {
        // result indicates the error code when you fail to enter the
room.
        NSLog(@"Enter room failed!");
    }
}
```

Note:

- To better transmit SEI messages for lyric synchronization, it is recommended to choose `TRTCAppSceneLIVE` for room entry scenarios.
- Under the automatic subscription mode (default), audiences automatically subscribe and play the on-mic anchor's audio and video streams upon entering the room.

2. Lyric synchronization

- Download lyrics.

  Obtain the target lyrics download link, LyricsUrl, from the business backend, and cache the target lyrics locally.
- Listener end lyric synchronization

```
- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    if (available) {
        [self.trtcCloud startRemoteView:userId view:nil];
    } else {
        [self.trtcCloud stopRemoteView:userId];
    }
}

- (void)onRecvSEIMsg:(NSString *)userId message:(NSData *)message {
    JSONModel *json = [[JSONModel alloc] initWithData:message
error:nil];
    NSDictionary *dic = json.toDictionary;
    int32_t musicId = [dic[@"musicId"] intValue];
```

```
    NSInteger progress = [dic[@"progress"] integerValue];
    NSInteger duration = [dic[@"duration"] integerValue];
    // ......
    // TODO: The logic of updating the lyric control.
    // Based on the received latest progress and the local lyrics
progress deviation, determine whether a lyric control seek is necessary.
    // ......
}
```

Note:

- Listeners need to actively subscribe to the lead singer's video streams in order to receive the SEI messages carried by black frames.
- If the lead singer's mixed stream also mixes in black frames, then only subscribing to the mixing stream robot's video stream is required.

3. Exit the room.

```
// Exit the room.
[self.trtcCloud exitRoom];

// Exit room event callback.
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"Proactively call exitRoom to exit the room.");
    } else if (reason == 1) {
        NSLog(@"Removed from the current room by the server.");
    } else if (reason == 2) {
        NSLog(@"The current room is dissolved.");
    }
}
```

Advanced Features

Music scoring module integration

Music scoring provides users with multi-dimensional singing scoring capabilities. Currently, supported scoring dimensions include intonation and rhythm.

1. Prepare scoring-related files.

Prepare in advance the performance recording files to be scored, original music standard files, MIDI pitch files, and upload them to COS storage.

2. Create a music scoring task.
   - Request Method: POST(HTTP).
   - Request Address: http://service-mqk0mc83-1257411467.bj.apigw.tencentcs.com/release/job.
   - Request Header: Content-Type: application/json.

   A request sample is as follows:

Request sample:

```
{
  "action": "CreateJob",
  "secretId": "{secretId}",
  "secretKey": "{secretKey}",
  "createJobRequest": {
    "customId": "{customId}",
    "callback": "{callback}",
    "inputs": [{ "url": "{url}" }],
    "outputs": [
      {
        "contentId": "{contentId}",
        "destination": "{destination}",
        "inputSelectors": [0],
        "smartContentDescriptor": {
          "outputPrefix": "{outputPrefix}",
          "vocalScore": {
            "standardAudio": {
              "midi": {"url":"{url}"},
              "standardWav": {"url":"{url}"},
              "alignWav": {"url":"{url}"}
            }
          }
        }
      }
    ]
  }
}
```

Response sample:


```
{
    "requestId": "ac004192-110b-46e3-ade8-4e449df84d60",
    "createJobResponse": {
        "job": {
            "id": "13f342e4-6866-450e-b44e-3151431c578b",
            "state": 1,
            "customId": "{customId}",
            "callback": "{callback}",
            "inputs": [{ "url": "{url}" }],
            "outputs": [
                {
                    "contentId": "{contentId}",
                    "destination": "{destination}",
                    "inputSelectors": [0],
                    "smartContentDescriptor": {
                        "outputPrefix": "{outputPrefix}",
                        "vocalScore": {
                          "standardAudio": {
                            "midi": {"url":"{url}"},
                            "standardWav": {"url":"{url}"},
                            "alignWav": {"url":"{url}"}
                          }
                        }
                    }
                }
            ],
            "timing": {
                "createdAt": "1603432763000",
                "startedAt": "0",
                "completedAt": "0"
            }
        }
    }
}
```


3. Obtain music scoring results.

   Obtain Method: Divided into active acquisition and passive callback.

- By querying with the ID obtained from the response packet after creating the task, if the queried task is successful (state=3), the task's Output will carry the smartContentResult structure, in which the vocalScore field stores the result JSON file name. Users can construct the output file's COS path based on the information in Output's COS and destination.

Request sample:

```
{
  "action": "GetJob",
  "secretId": "{secretId}",
  "secretKey": "{secretKey}",
  "getJobRequest": {
    "id": "{id}"
  }
}
```

Response sample:

```
{
  "requestId": "c9845a99-34e3-4b0f-80f5-f0a2a0ee8896",
  "getJobResponse": {
    "job": {
      "id": "a95e9d74-6602-4405-a3fc-6408a76bcc98",
      "state": 3,
      "customId": "{customId}",
      "callback": "{callback}",
      "timing": {
        "createdAt": "1610513575000",
        "startedAt": "1610513575000",
        "completedAt": "1610513618000"
      },
      "inputs": [{ "url": "{url}" }],
      "outputs": [
        {
          "contentId": "{contentId}",
          "destination": "{destination}",
          "inputSelectors": [0],
          "smartContentDescriptor": {
            "outputPrefix": "{outputPrefix}",
```

```
            "vocalScore": {
              "standardAudio": {
                "midi": {"url":"{url}"},
                "standardWav": {"url":"{url}"},
                "alignWav": {"url":"{url}"}
              }
            }
          },
          "smartContentResult": {
            "vocalScore": "out.json"
          }
        }
      ]
    }
  }
}
```

- Passive callbacks need to fill in the callback field when creating a task. The platform will send the entire Job structure to the address specified by the callback after the task reaches the Completed state (COMPLETED/ERROR). It is recommended to obtain task results using passive callbacks. The entire Job structure of tasks that have reached the Completed state (COMPLETED/ERROR) will be sent to the address corresponding to the callback field specified when the task was created. See the active query sample for the Job structure (under getJobResponse).

> **Note:**
> For more detailed intelligent music solution integration instructions for the music scoring module, see Music Scoring Integration.

Transparent transmission of single stream volume in mixed streams.

After the mixed streaming is enabled, the audience cannot directly obtain the on-mic anchor's single stream volume. In order to transparently transmit the single stream volume, the room owner may employ SEI to transmit the callback volume values of all on-mic anchors.

```
- (void)onUserVoiceVolume:(NSArray<TRTCVolumeInfo *> *)userVolumes
totalVolume:(NSInteger)totalVolume {
    if (userVolumes.count) {
        // For storing volume values corresponding to on-mic users.
```

```
        NSMutableDictionary *volumesMap = [NSMutableDictionary
dictionary];
        for (TRTCVolumeInfo *user in userVolumes) {
            // Can set an appropriate volume threshold.
            if (user.volume > 10) {
                volumesMap[user.userId] = @(user.volume);
            }
        }
        JSONModel *json = [[JSONModel alloc]
initWithDictionary:volumesMap error:nil];
        // Transmit a collection of on-mic users' volume via SEI
messages.
        [self.trtcCloud sendSEIMsg:json.toJSONData repeatCount:1];
    }
}

- (void)onRecvSEIMsg:(NSString *)userId message:(NSData *)message {
    JSONModel *json = [[JSONModel alloc] initWithData:message
error:nil];
    NSDictionary *dic = json.toDictionary;
    for (NSString *userId in dic.allKeys) {
        // Print the volume levels of single streams of all on-mic
users.
        NSLog(@"%@: %@", userId, dic[userId]);
    }
}
```

> **Note:**
> The prerequisite for using SEI messages to transparently transmit single stream volume through a mixed stream is that the room owner must either be video streaming or have black frame insertion enabled and furthermore, the audiences must actively subscribe to the room owner's video stream.

Real-time network quality callback

You can listen to `onNetworkQuality` to real-time monitor the network quality of both local and remote users. This callback is thrown every 2 seconds.

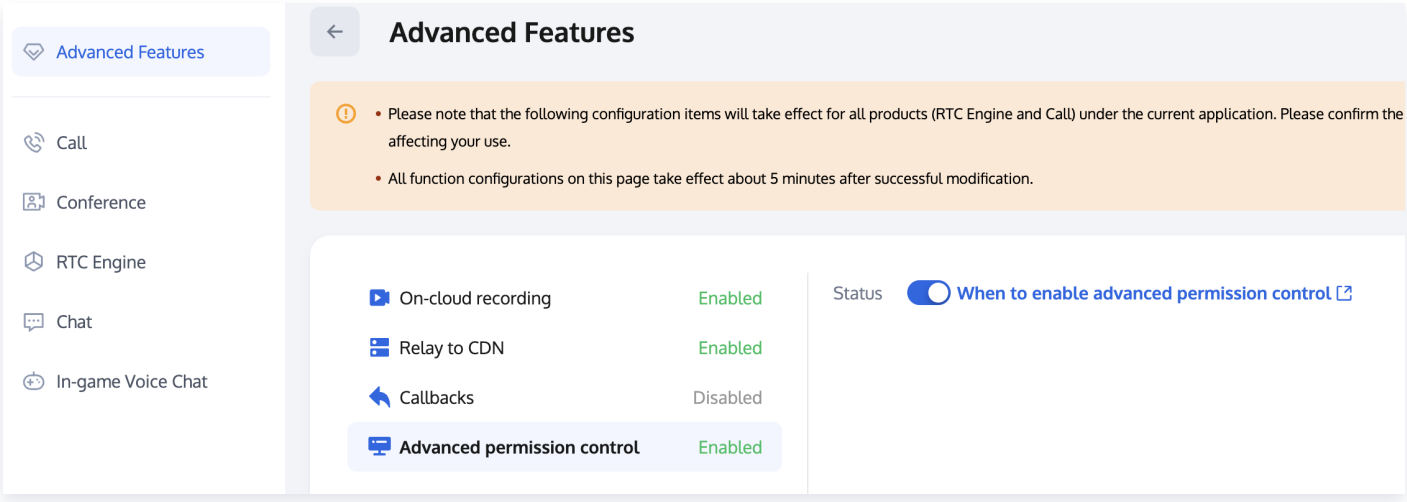
```
#pragma mark - TRTCCloudDelegate
```

```
- (void)onNetworkQuality:(TRTCQualityInfo *)localQuality remoteQuality:
(NSArray<TRTCQualityInfo *> *)remoteQuality {
    // localQuality represents the local user's network quality
evaluation result.
    // remoteQuality represents the remote user's network quality
evaluation result. The result is affected by both remote and local
factors.
    switch(localQuality.quality) {
        case TRTCQuality_Unknown:
            NSLog(@"Undefined.");
            break;
        case TRTCQuality_Excellent:
            NSLog(@"The current network is excellent.");
            break;
        case TRTCQuality_Good:
            NSLog(@"The current network is good.");
            break;
        case TRTCQuality_Poor:
            NSLog(@"The current network is moderate.");
            break;
        case TRTCQuality_Bad:
            NSLog(@"The current network is poor.");
            break;
        case TRTCQuality_Vbad:
            NSLog(@"The current network is very poor.");
            break;
        case TRTCQuality_Down:
            NSLog(@"The current network does not meet the minimum
requirements of TRTC.");
            break;
        default:
            break;
    }
}
```

Advanced permission control

TRTC advanced permission control can be used to set different entry permissions for different rooms, such as advanced VIP rooms. It can also be used to control the permission for the audience to speak, such as handling ghost microphones.

Step 1: Enable the Advanced Permission Control Switch in the TRTC console application's advanced features page.



Note:
Once advanced permission control is enabled for a certain SDKAppID, all users using that SDKAppID need to pass in the `privateMapKey` parameter in `TRTCParams` to successfully enter the room. Therefore, if you have users online using this SDKAppID, do not enable this feature.

Step 2: Generate privateMapKey on the backend. For sample code, see privateMapKey computation source code.

Step 3: Room entry verification & speaking permission verification with PrivateMapKey.

- Room entry verification

```
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = SDKAppID;
params.roomId = self.roomId;
params.userId = self.userId;
// UserSig obtained from the business backend.
params.userSig = [self getUserSig];
// PrivateMapKey obtained from the backend.
params.privateMapKey = [self getPrivateMapKey];
params.role = TRTCRoleAudience;
```

```
[self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
```

- Speaking permission verification

```
// Pass in the latest PrivateMapKey obtained from the backend into the
role switching API.
[self.trtcCloud switchRole:TRTCRoleAnchor privateMapKey:[self
getPrivateMapKey]];
```

Exception Handling

Exception error handling

When the TRTC SDK encounters an unrecoverable error, the error will be thrown in the `onError` callback. For details, see Error Code Table.

1. UserSig related

UserSig verification failure will lead to room-entering failure. You can use the UserSig tool for verification.

Enumeration	Value	Description
ERR_TRTC_INVALID_USER_SIG	-3320	Room entry parameter userSig is incorrect. Check if `TRTCParams.userSig` is empty.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	UserSig verification failed. Check if the parameter `TRTCParams.userSig` is filled in correctly or has expired.

2. Room entry and exit related

If failed to enter the room, you should first verify the correctness of the room entry parameters. It is essential that the room entry and exit APIs are called in a paired manner. This means that, even in the event of a failed room entry, the room exit API must still be called.

Enumeration	Value	Description
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	Room entry request timed out. Check if your internet connection is lost or if a VPN is enabled. You may also attempt to switch to 4G for testing.
ERR_TRTC_INVALID_SDK_APPID	-3317	Room entry parameter sdkAppId is incorrect. Check if `TRTCParams.sdkAppId` is empty.
ERR_TRTC_INVALID_ROOM_I	-331	Room entry parameter roomId is incorrect. Check if

D	8	`TRTCParams.roomId` or `TRTCParams.strRoomId` is empty. Note that roomId and strRoomId cannot be used interchangeably.
ERR_TRTC_INVALID_USER_ID	-3319	Room entry parameter userId is incorrect. Check if `TRTCParams.userId` is empty.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	Room entry request is denied. Check if `enterRoom` is called consecutively to enter rooms with the same ID.

3. Device related

Errors for relevant monitoring devices. Prompt the user via UI in case of relevant errors.

Enumeration	Value	Description
ERR_MIC_START_FAIL	-1302	Failed to open the mic. For example, if there is an exception for the mic's configuration program (driver) on a Windows or macOS device, you should try disabling then re-enabling the device, restarting the machine, or updating the configuration program.
ERR_SPEAKER_START_FAIL	-1321	Failed to open the speaker. For example, if there is an exception for the speaker's configuration program (driver) on a Windows or macOS device, you should try disabling then re-enabling the device, restarting the machine, or updating the configuration program.
ERR_MIC_OCCUPY	-1319	The mic is occupied. This occurs when, for example, the user is currently having a call on the mobile device.

Issues with IEMs

1. How to enable IEMs feature and set the volume?

```
// Enable IEMs.
[[self.trtcCloud getAudioEffectManager] enableVoiceEarMonitor:YES];
// Set the volume of IEMs.
[[self.trtcCloud getAudioEffectManager]
setVoiceEarMonitorVolume:volume];
```

> **Note:**
> The IEMs can be set in advance without having to monitor audio routing changes. Once headphones are connected, the IEMs feature will automatically take effect.

2. The IEMs feature does not take effect after enabled.

Due to the high hardware delay of Bluetooth headphones, it is recommended to prompt the anchor to wear wired headphones on the user interface. Also, it should be noted that not all smartphones will achieve excellent IEMs effect after this feature is enabled. TRTC SDK has already blocked this feature on some smartphones with poor effect.

3. High IEM delay

Check if Bluetooth headphones are in use. Due to the high hardware delay of Bluetooth headphones, wired headphones are recommended. Additionally, you can try improving the issue of high IEM delay by enabling hardware IEM through the experimental API `setSystemAudioKitEnabled`. Hardware IEMs have better performance and lower delay. Software IEMs have higher delay but better compatibility. Currently, for Huawei and VIVO devices, SDK defaults to hardware IEMs. Other devices default to software IEMs. If there are compatibility issues with hardware IEMs, contact us to configure forced use of software IEMs.

Issues with NTP sync

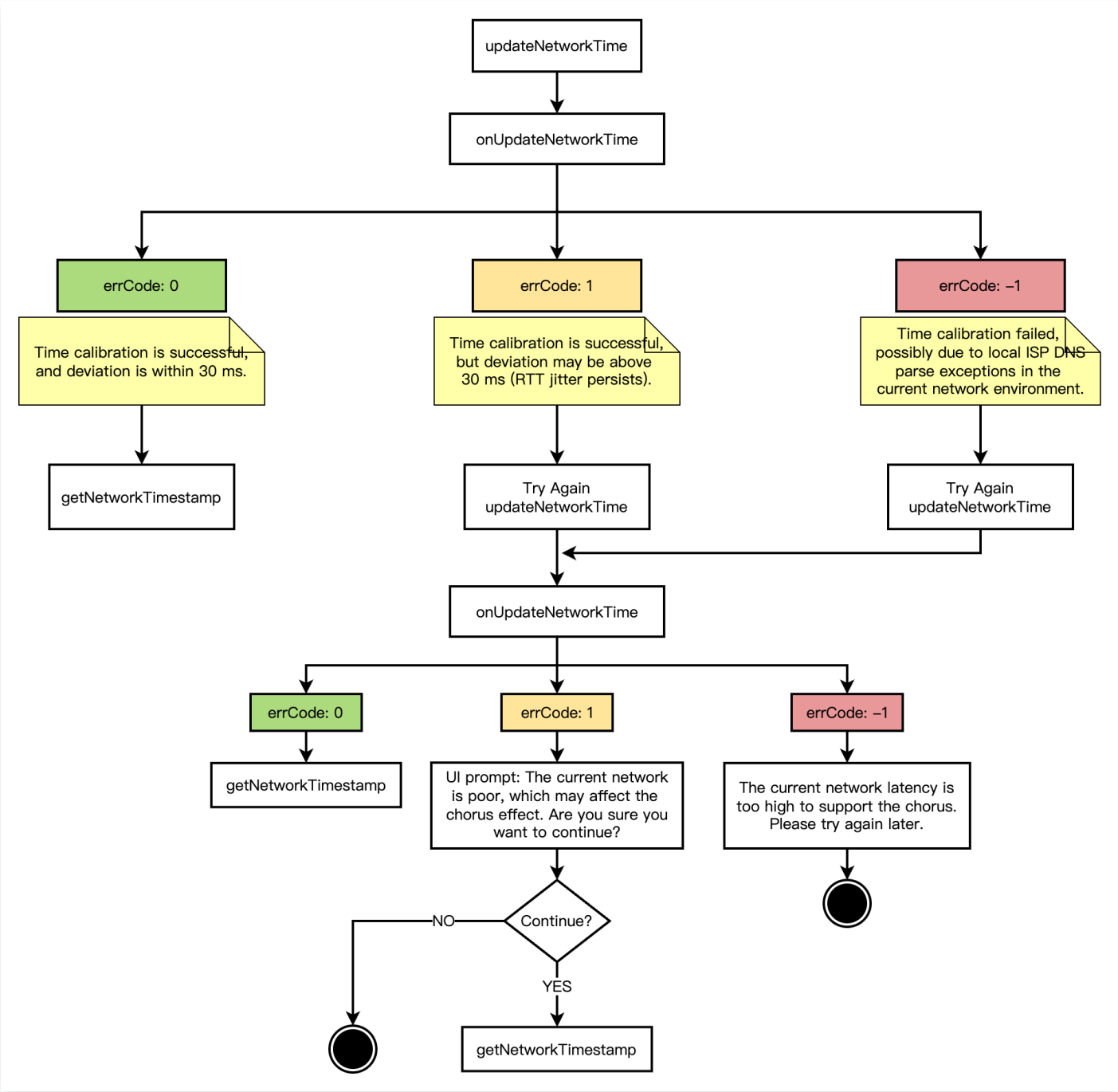
1. NTP time sync finished, but result maybe inaccurate.

NTP sync is successful, but the deviation may still be more than 30 milliseconds. This indicates a poor client network environment with persistent RTT jitter.

2. Error in AddressResolver: No address associated with hostname

NTP sync has failed, possibly due to a temporary exception in local ISP DNS resolution under the current network environment. Try again later.

3. NTP service retry processing logic.



Issues with real-time chorus usage

1. **Why does the lead singer in real-time chorus scenarios need to use dual-instance streaming?**

   In real-time chorus scenarios, to minimize end-to-end delay and achieve sync between vocals and accompaniment, a common approach is to use dual instances at the lead singer's end to separately upload vocal and accompaniment streams, while other chorus participants only upload their vocal streams and locally play the accompaniment. In this case, each chorus participant needs to subscribe to the lead singer's vocal stream, while refraining from subscribing to the lead singer's music stream. This setup can only be achieved by implementing dual-instance separate streaming.
2. **Why is it recommended to enable mixing pushback in real-time chorus scenarios?**

   Having the audience pull multiple single streams at the same time is likely to result in misalignment between multiple vocal streams and accompaniment streams. Pulling a mixed stream can ensure absolute alignment of all streams and reduce downstream bandwidth.
3. **What are the uses of SEI in real-time chorus scenarios?**
   - Transmitting accompaniment music progress, for lyric sync on the audience's end.
   - Transparently transmitting single stream volume through a mixed stream, for display as sound waves on the listener's end.
4. **Loading accompaniment music takes a long duration, causing significant playback delay?**

   Loading network music resources via the SDK incurs a certain delay. It is recommended to initiate music pre-loading before starting playback.

```
[[self.trtcCloud getAudioEffectManager] preloadMusic:musicParam
onProgress:nil onError:nil];
```

5. **When singing along with accompaniment, the vocals are barely audible. Is the music overwhelming the vocals?**

   If the default volume settings result in the accompaniment overwhelming the vocals, it is recommended to adjust the volume balance between the music and vocals accordingly.

```
// Set the local playback volume of a piece of background music.
[[self.trtcCloud getAudioEffectManager]
setMusicPlayoutVolume:self.musicId volume:volume];
// Set the remote playback volume of a specific background music.
[[self.trtcCloud getAudioEffectManager]
setMusicPublishVolume:self.musicId volume:volume];
// Set the local and remote volume of all background music.
[[self.trtcCloud getAudioEffectManager] setAllMusicVolume:volume];
// Set the volume of voice capture.
[[self.trtcCloud getAudioEffectManager] setVoiceVolume:volume];
```

쇼 라이브 방송 시나리오의 솔루션

최종 업데이트 날짜:: 2025-10-28 11:32:36

시나리오의 소개

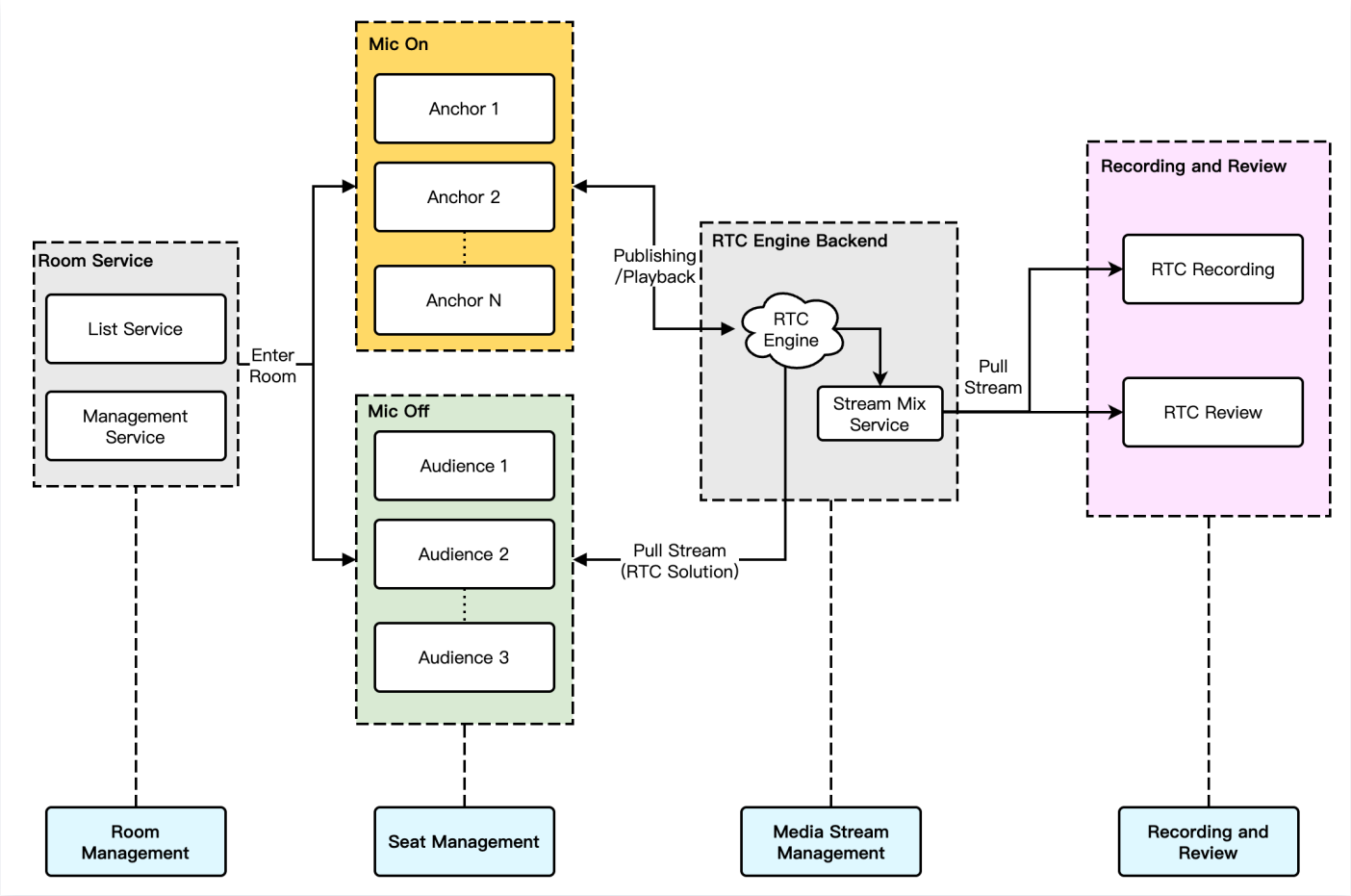
쇼 라이브방송 시나리오는 소셜 엔터테인먼트 모드의 비디오 인터랙티브 시나리오이며, 많은 사용자들이 같이 비디오 연결을 지원하여 사용자의 참여 유도와 소비 의향 및 충성도를 높이기 쉽습니다. 또한, 쇼 라이브방송은 다른 방의 스트리머 간의 크로스 룸 PK를 지원하여 라이브방송의 재미를 더욱 강화시킵니다. 크로스 룸 연결 PK의 지연 시간은 300ms 미만이며 동시에 시청자와 스트리머의 연결을 지원하고, 마이크 연결/해제 시 원활한 전환이 가능하여 쇼 라이브 시나리오의 고빈도 인터랙티브 요구를 충족합니다. 쇼 라이브방송의 스마트 뷰티 기능 또한 스트리머의 개인화 요구를 충족시켜 라이브 방송을 더욱 매력적으로 만듭니다.

구현 방안

일반적으로 완전한 쇼 라이브방송 시나리오를 구현하려면 여러 기능 모듈이 필요합니다. 룸 관리, 마이크 관리, 미디어 스트림 관리, 클라우드 레코딩 등 있습니다. 각 기능 모듈의 주요 작업 및 기능 포인트는 아래 표와 같습니다. 다음으로 각 기능 모듈을 하나씩 소개하여 쇼 라이브방송 구축에 필요한 기능에 대한 완전한 이해를 만들어주겠습니다.

기능 모듈	주요 작업 및 기능 포인트
룸 관리	룸 목록, 룸 생성, 룸 입장, 룸 나가기, 룸 파기
마이크 순위 관리	마이크 사용 신청, 마이크 사용 포기, 마이크 사용 초청, 강제로 마이크 사용 불가, 마이크 사용 금지
미디어 스트림 관리	RTC Engine 실시간 인터랙티브의 솔루션
클라우드 레코딩	RTC Engine 클라우드 레코딩

쇼 라이브 방송 시나리오의 전체 업무 아키텍처는 아래 그림과 같습니다. 방주인이 방을 생성하면 사용자는 관심 있는 방을 선택해서 입장할 수 있습니다. 방에 입장한 사용자는 마이크를 켜고 스트리머와 오디오/비디오로 인터랙티브할 수 있습니다. 일반적으로 규정 준수 요구 사항으로 인해 방 내 오디오/비디오 콘텐츠는 녹화되어 심사에 제출되어야 합니다.



설명:

- 레코딩 심사 단계에서는 실제 업무 요구 사항에 따라 타겟 단일 스트림 또는 혼합 스트림을 선택할 수 있습니다.
- 레코딩 및 심사 단계에서는 실제 업무 요구 사항에 따라 RTC Engine 레코딩 심사를 선택할 수 있습니다.

룸 관리

룸 관리 모듈은 주로 룸 목록 유지 관리를 담당하며, 주로 다음 기능을 포함합니다.

- 방 생성: 사용자가 업무 시스템에 로그인한 후 방을 생성할 수 있으며, 방 생성 후 방 목록에 추가 작업이 이루어집니다.
- 방 입장: 사용자는 기존 방에 입장할 수 있으며, 방 임장 후 현재 방 인원 목록에 추가 작업이 이루어집니다.
- 방 나가기: 사용자는 현재 방을 나갈 수 있으며, 방 나간 후 현재 방 인원 목록에 삭제 작업이 이루어집니다.

- 방 파기: 모든 사용자가 방을 나간 후 방을 파기해야 하며, 방 파기 후 방 목록에 삭제 작업이 이루어집니다.

> **설명:**
> 룸 관리는 쇼 라이브 방송을 구현하기 위한 필수 모듈이지만 주요 기능 모듈은 아닙니다. 구체적으로는 업무 시스템 및 Chat & RTC Engine SDK와 결합하여 구현할 수 있으며, 자세한 내용은 음성 채팅방-룸 관리를 참조하세요.

마이크 순위 관리

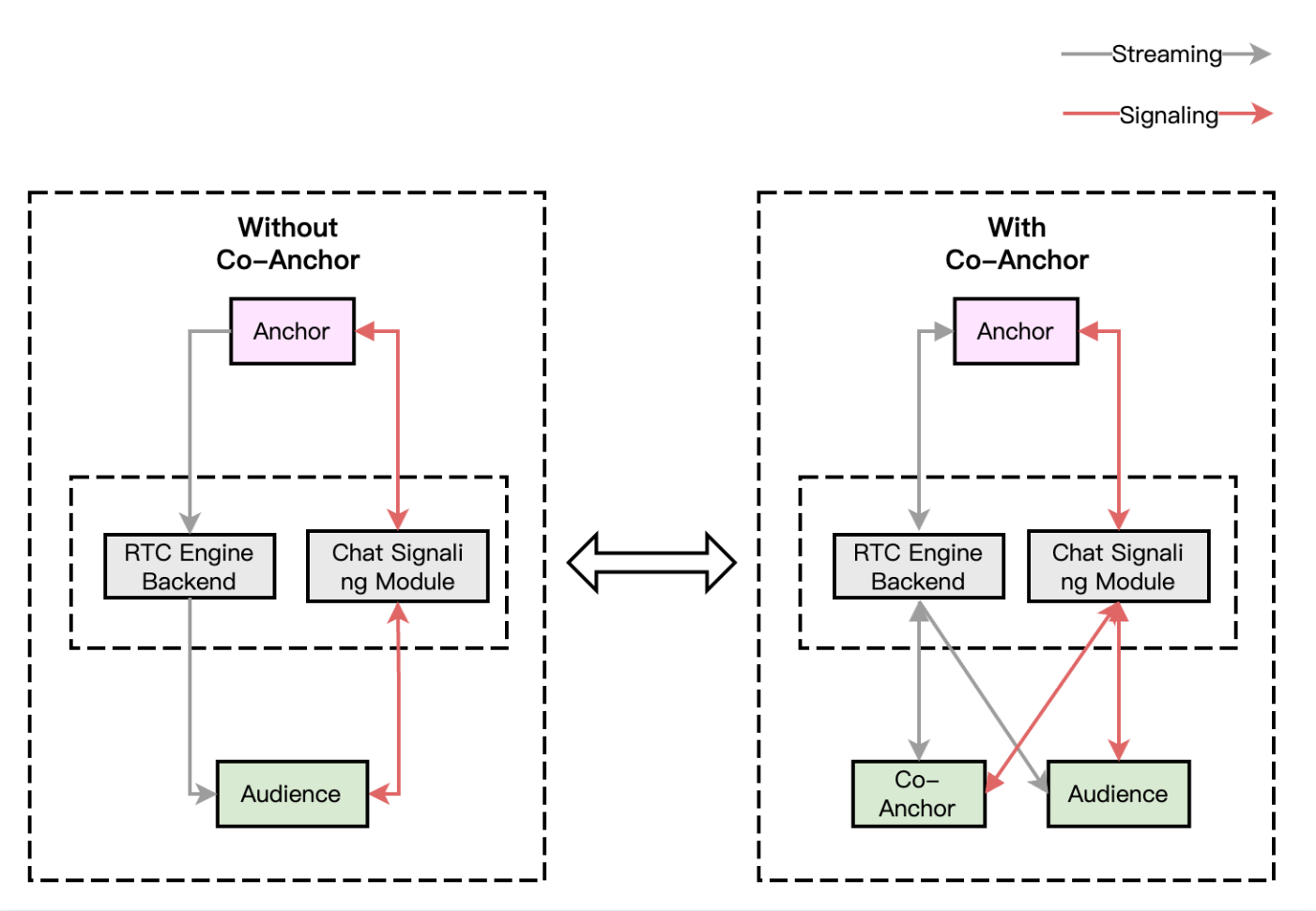
라이브 방송실 내의 마이크 순위는 일반적으로 순서가 있고 제한적입니다. 마이크 순위 관리는 주로 업무 시나리오에 따라 방 내의 마이크 수량을 정하고, 현재 방의 모든 마이크 순위 상태를 관리하는 역할을 합니다. 마이크 순위 관리에는 주로 다음이 포함됩니다. 마이크 사용 신청, 마이크 사용 자동 포기, 마이크 사용 초청, 강제로 마이크 사용 불가, 마이크 사용 금지등 있습니다.

- 사용자가 방에 입장한 후, 비어 있는 상태의 마이크만 사용을 신청할 수 있습니다.
- 방주인이 사용자의 마이크 신청을 수용한 후에는 마이크 순위 상태를 비어 있지 않음으로 수정해야 합니다.
- 사용자가 스트리밍을 중단하고 마이크 사용을 포기한 후 마이크 순위 상태를 재설정해야 합니다.
- 방 주인은 마이크 잠금, 마이크 사용 초청, 강제로 마이크 사용 불가, 마이크 사용 금지등의 권한이 있습니다.

> **설명:**
> 마이크 관리는 쇼 라이브 방송을 구현하기 위한 필수 모듈이지만 주요 기능 모듈은 아닙니다. 구체적으로는 업무 시스템 및 Chat & RTC Engine SDK와 결합하여 구현할 수 있으며, 자세한 내용은 음성 채팅방-마이크 관리를 참조하세요.

미디어 스트림 관리

일반적인 쇼 라이브 방송 시나리오에서는 RTC Engine 실시간 인터랙티브 솔루션을 권장합니다.스트리머와 시청자 모두 RTC 프로토콜을 사용하여 스트림을 푸시 및 풀하며, 이 솔루션은 최소한의 엔드투엔드 지연 시간을 제공합니다. 동시에 시청자의 마이크 연결/해제는 더 원활하고 화면 빠르기/되감기 등의 급격한 변화 현상이 없습니다. 다중 사용자 마이크 연결 라이브 인터랙티브를 예로 들면, 쇼 라이브 방송의 순수 RTC 푸시/풀 스트리밍 시나리오의 주요 아키텍처는 아래 그림과 같습니다.



해당 솔루션의 전체 프로세스는 다음과 같습니다.

1. 스트리머와 시청자는 모두 시그널링 모듈을 통해 연결됩니다.시그널링 모듈은 주로 라이브 방송의 프로세스 제어와 라이브 방송의 상태 동기화를 관리합니다.
2. 연결된 시청자 유무와 관계없이 스트리머와 시청자는 모두 RTC Engine 오디오/비디오 클라우드 서비스를 통해 스트림을 푸시 및 풀합니다.
3. 시청자가 스트리머와 연결을 요청하면, 시그널링 모듈이 스트리머에게 알리고 연결 요청자의 개인 정보를 동기화합니다.
4. 스트리머가 연결 요청을 수락하면 연결된 시청자가 스트리밍을 시작하고 방 내 모든 멤버는 스트림 업데이트 알림을 받으며 연결된 시청자의 오디오/비디오 스트림을 풀합니다.
5. 연결된 시청자가 마이크 나가기의 요청을 하면 스트리밍을 중지하고 방의 모든 멤버는 스트림 업데이트 알림을 받으며 해당 시청자의 오디오/비디오 스트림을 풀하지 않습니다.

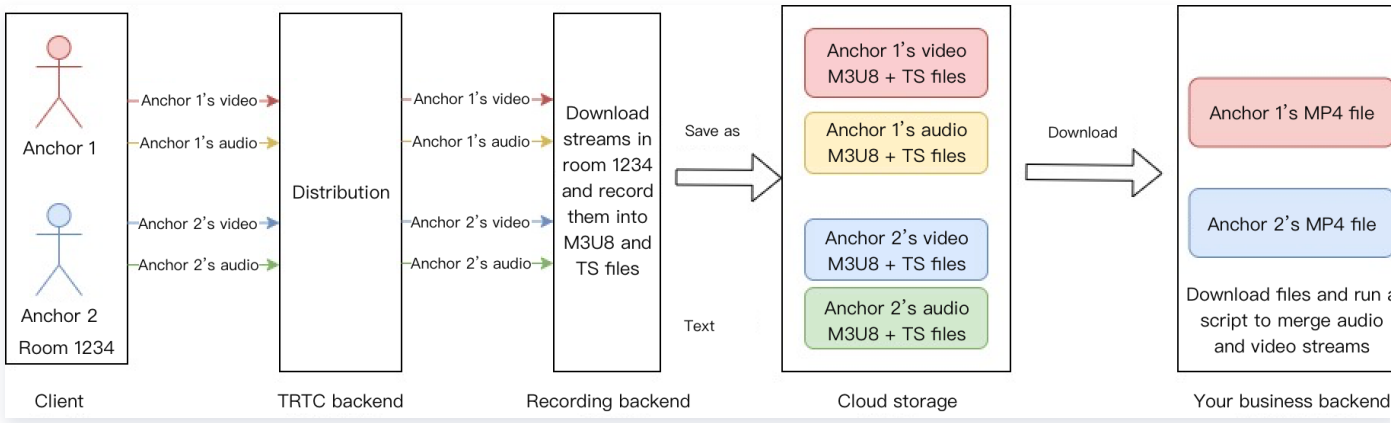
설명:

> 시그널링 모듈은 자체 개발 시그널링 채널일 수 있으며 동시에 텐센트 클라우드 Chat을 사용하여 시그널링 교체하는 것이 좋습니다.

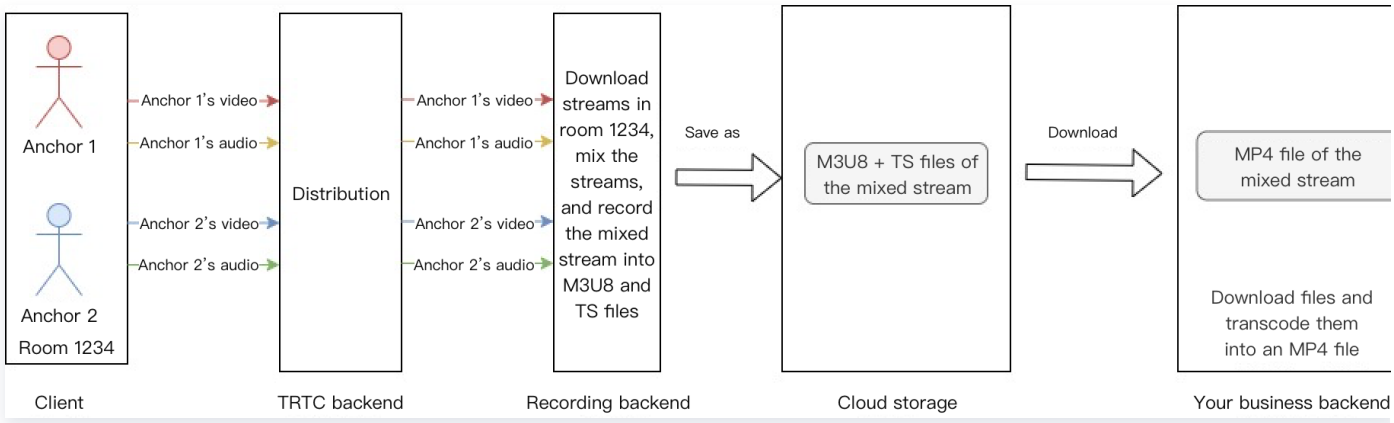
클라우드 레코딩

RTC Engine 최신 업그레이드된 클라우드 레코딩은 RTC Engine 내부의 실시간 레코딩 클러스터를 사용하여 오디오 및 비디오를 녹화하며, 더 완전하고 통일된 레코딩 경험을 제공합니다.

- 단일 스트림 레코딩: RTC Engine 클라우드 레코딩 기능을 통해 방 안의 각 사용자의 오디오 및 비디오 스트림을 독립적인 파일로 녹화할 수 있습니다.



- 혼합 스트림 레코딩: 동일한 방의 오디오 및 비디오등 미디어 스트림을 하나의 파일로 혼합하여 녹화합니다.



> **설명:**
> - RTC Engine 클라우드 레코딩에 대한 자세한 소개 및 개설 안내는 RTC Engine 클라우드 레코딩 설명을 참조하십시오.

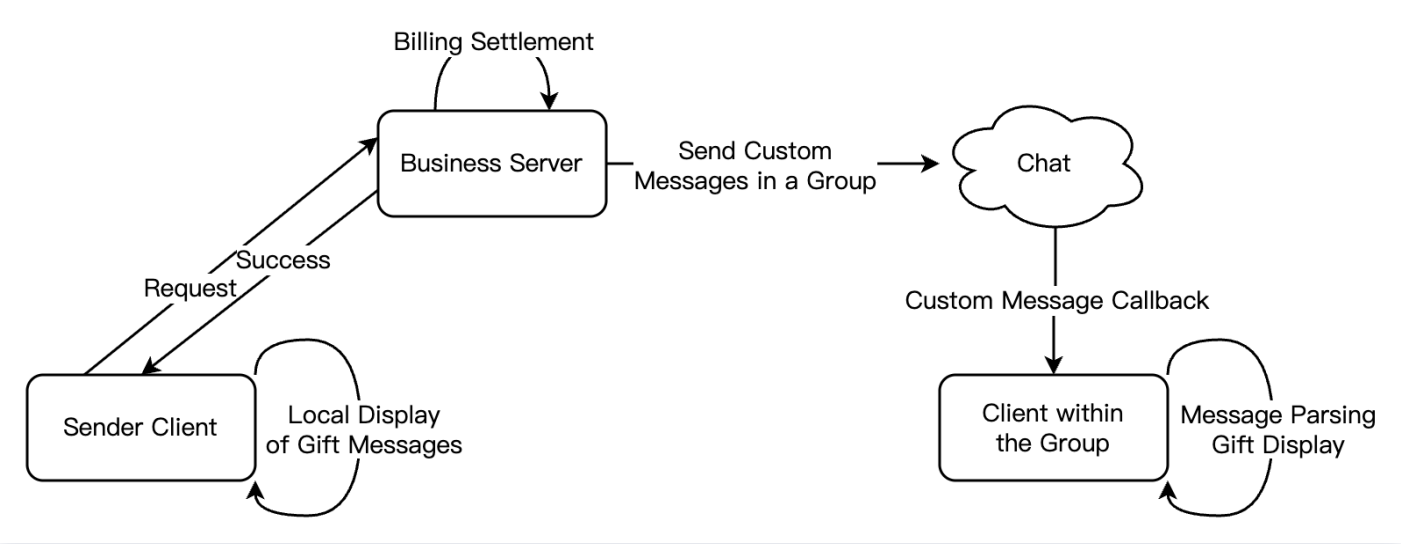
- 쇼 라이브 방송 시나리오에서는 일반적으로 혼합 스트림 레코딩 솔루션을 사용합니다.스트리머 또는 연결된 시청자의 단일 스트림 심사가 필요한 경우 단일 스트림 레코딩 솔루션을 선택할 수 있습니다.

핵심 업무의 로직

선물과 좋아요 메시지

쇼 라이브 방송 시나리오에서 선물과 좋아요는 일반적인 인터랙티브 방식이고 시청자는 선물을 보내고 좋아요를 눌러 스트리머에 대한 사랑과 지원을 표현할 수 있으며, 스트리머도 이로부터 수익을 얻을 수 있습니다. 아래에서는 텐센트 클라우드 Chat을 기반으로 한 선물 메시지 및 좋아요 메시지 구현 방식을 각각 소개하겠습니다.

선물 메시지

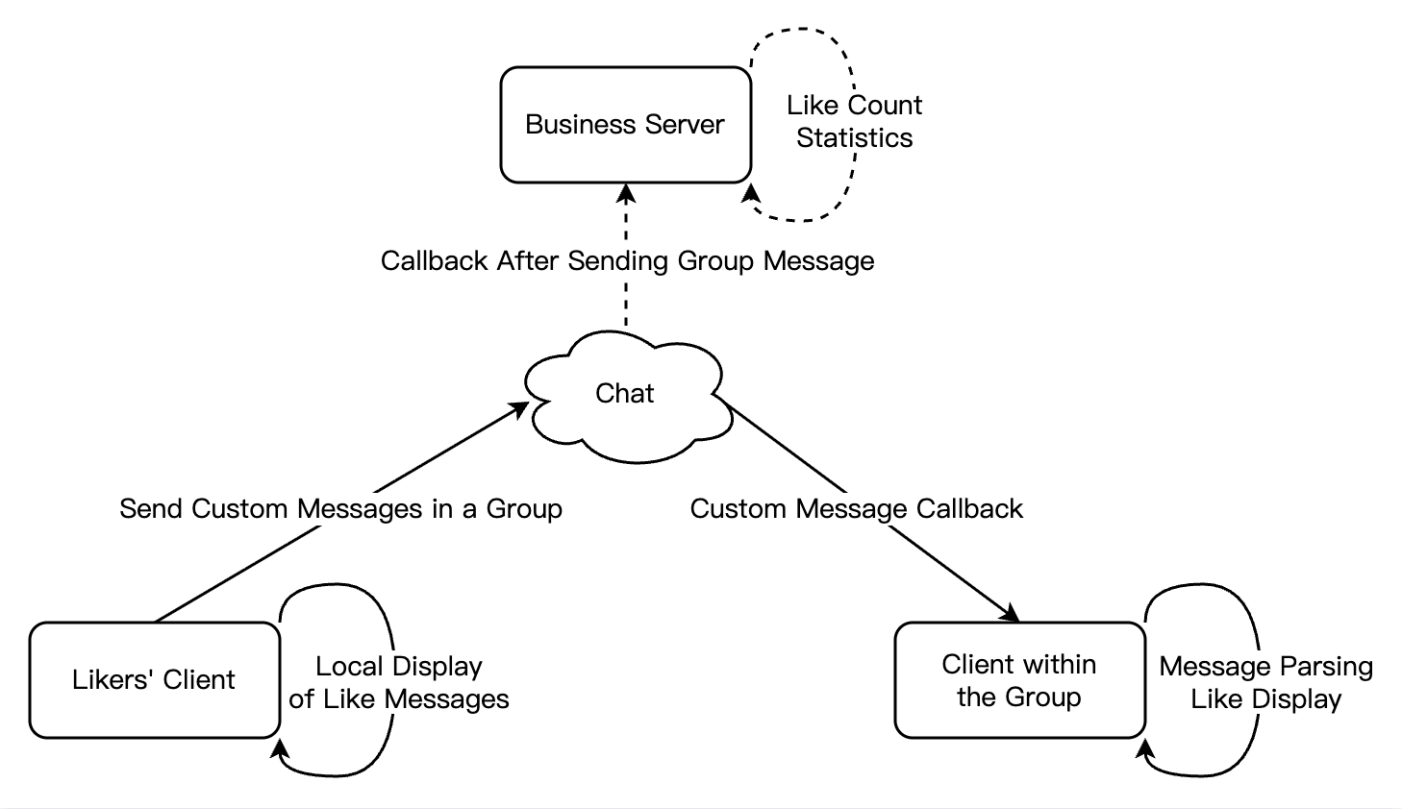


1. 클라이언트는 자체 업무 서버와 연결을 요청하면 선물 결제 로직이 관련됩니다.
2. 요금 결제 후 발송자는 XXX가 XXX 선물을 보낸 것을 직접 확인할 수 있습니다(발송자가 자신이 보낸 선물을 직접 볼 수 있도록 하기 위함이며, 메시지 수량이 많을 경우 포기 전략이 트리거될 수 있습니다).
3. 요금 결제 완료 후, 서버 인터페이스 를 호출하여 그룹에서 사용자 커스텀 메시지(선물)를 발송합니다.
4. 연속 선물 전송 시나리오에서는 메시지 병합이 필요합니다.
   - 선물 수량을 직접 선택하여 선물을 보내는 경우, 예를 들어 선물 99개를 직접 선택하고 1개의 메시지만으로 발송하면 매개변수에 선물 수량이 99로 나옵니다.
   - 연속 클릭 선물인 경우 멈추는 개수를 확정할 수 없으면 20개마다(수량은 자체 조정 가능) 또는 연속 1초를 초과할 때 하나로 병합하여 발송합니다. 위 로직에 따라 예를 들어 연속 99개 선물은 최적화 후 5개만 발송하면 됩니다.

서버 인터페이스 는 그룹에서 사용자 커스텀 메시지 를 발송하는 요청 패킷 예시는 다음과 같으며, 이 중에 MsgBody는 사용자 커스텀 메시지 요소를 지원합니다.

```
{
    "GroupId": "@TGS#12DEVUDHQ",
    "Random": 2784275388,
    "MsgPriority": "High",  // 메시지의 우선순위이고 선물 메시지는 높은 우선순
위로 설정해야 합니다
    "MsgBody": [
        {
            "MsgType": "TIMCustomElem",
            "MsgContent": {
                // type: 선물 유형; giftUrl: 선물 리소스 주소; giftName: 선
물 이름; giftCount: 선물 수량
                "Data": "{""cmd"": ""gift_msg", ""msg"": {""type"": 1,
""giftUrl"": ""xxx", ""giftName"": ""xxx", ""giftCount"": 1}}"
            }
        }
    ]
}
```

좋아요 메시지



1. 좋아요는 요금이 부과되지 않으며 일반적으로 클라이언트에서 직접 그룹 커스텀 메시지를 보내는 방식을 사용합니다.
2. 서버 측에서 카운팅이 필요한 좋아요 메시지의 경우, 클라이언트에서 쓰로틀링을 적용한 후 클라이언트의 좋아요 누르는 횟수를 집계하여 짧은 시간 내의 여러번으로 누른 좋아요 메시지를 병합하고 한 번만 메시지를 전송합니다. 업무 측 서버는 그룹 메시지 전송 후 를 콜백하여 좋아요 누르는 횟수를 받아 통계를 수행합니다.
3. 서버 측에서 카운팅이 필요하지 않은 좋아요 메시지의 경우, 단계 2의 로직과 동일하게 업무 측은 클라이언트에서 좋아요 메시지를 쓰로틀링한 후 전송하며, 그룹 메시지 전송 후 콜백에서 카운팅할 필요가 없습니다.

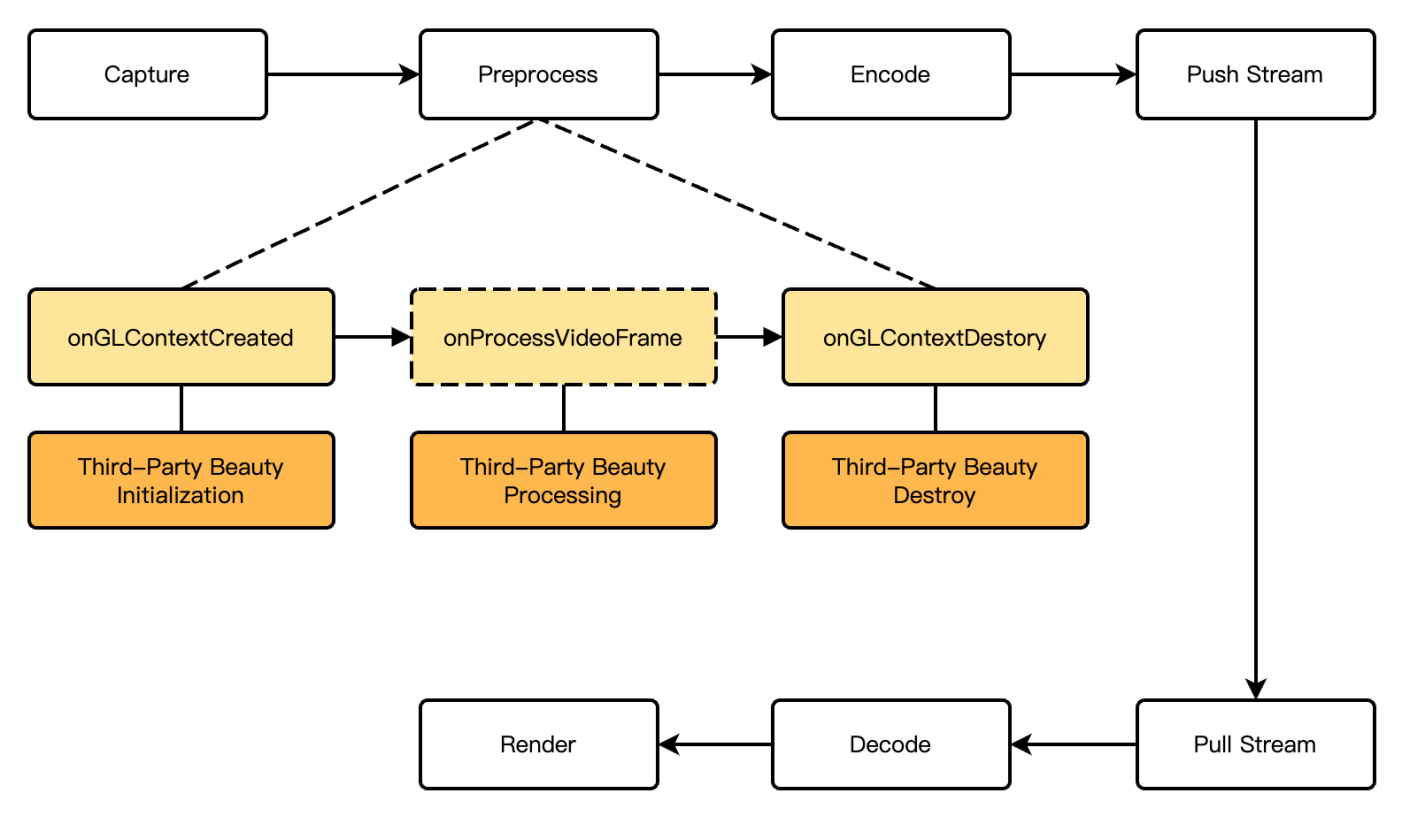
⚠ 주의:

- 중요한 메시지(예: 선물 메시지)는 높은 우선순위로 설정하고 빈도가 높지만 중요하지 않은 메시지(예: 좋아요 메시지)는 낮은 우선순위로 설정하세요.
- 라이브 방송 인터랙션 기능(좋아요, 선물, 탄막 발송 및 수신)의 구체적인 구현 단계는 빠른 접근 가이드-라이브 인터랙션 메시지를 참조하세요.

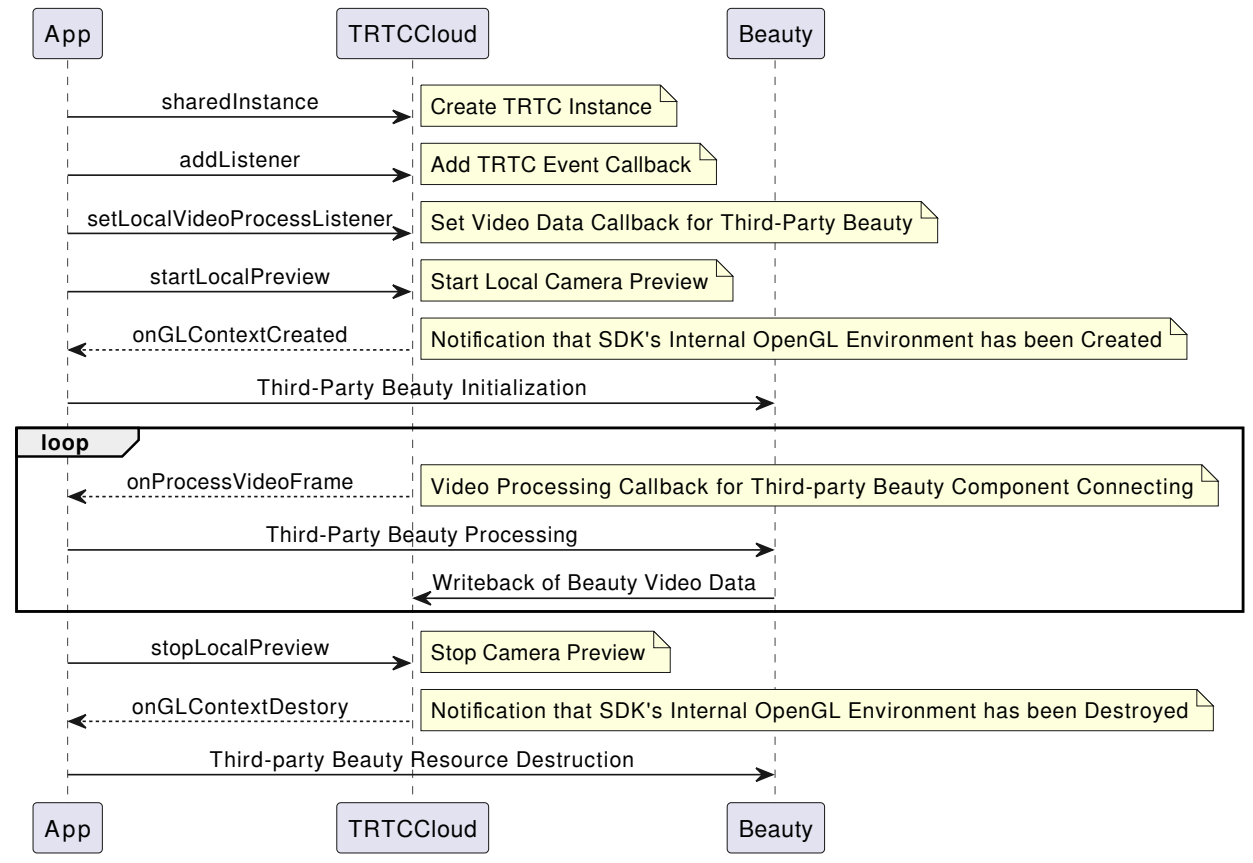
뷰티 효과 접근

쇼 라이브 방송 시나리오에서 뷰티 기능은 자주 사용되는 기능입니다. 뷰티 기능은스트리머의 외모를 개선할 뿐만 아니라 다양한 스티커 효과를 통해 라이브 방송의 재미를 더할 수 있습니다. RTC Engine 뷰티 AR SDK 통합을 지원하며 화산 뷰티, 얼굴 뷰티 등 시중의 주요 타사 뷰티 제품도 지원할 수 있습니다.

뷰티 효과 접근 절차



API 호출 타이밍



뷰티 제품 비교

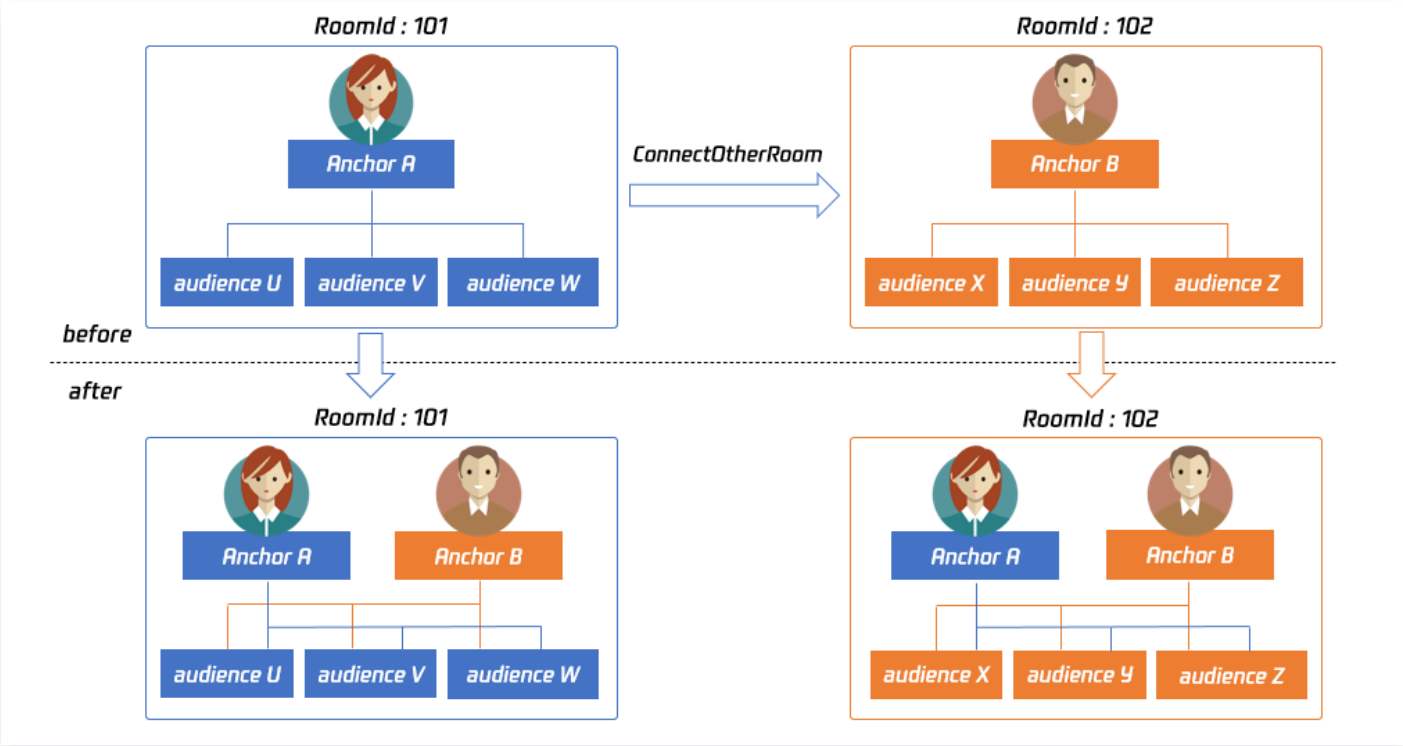
뷰티 유형	뷰티 효과	접속 비용	비용	가상 디지털 인간	단말기 지원
뷰티 AR SDK	기본 효과가 우수하며 고급 효과(큰 눈/얼굴 슬림 등)가 뛰어납니다.	낮음	적당함	지원	Android/iOS/PC/Flutter/Web/미니프로그램
얼굴 뷰티 효과 SDK	기본 효과가 우수하며 고급 효과(큰 눈/얼굴 슬림 등)가 일반적입니다.	높음	적당함	지원	Android/iOS/PC/Unity
화산 효과 SDK	기본 효과가 우수하며 고급 효과가(큰 눈/얼굴 슬림 등) 뛰어납니다.	높음	높은 편	지원	Android/iOS/PC/Linux

크로스 룸 연결 PK

스트리머 간의 크로스 룸 연결 PK는 쇼 라이브 방송에서 매우 일반적인 플레이 방식이며 인터랙티브 라이브 방송의 재미를 높이고 시청자들이 선물을 보내고 랭킹을 올리려는 욕구를 자극합니다. RTC Engine 여러 방과 여러 스트리머 간의 크로스 룸 연결 PK를 지원합니다.아래에서 구체적인 구현 방법을 소개합니다.

1. **구현 원리**

기본적으로 동일한 방 내의 사용자들 간에만 음성 및 영상 통화가 가능하며 서로 다른 방 간의 음성 및 영상 스트림은 분리되어 있습니다. 크로스 룸 연결을 통해 다른 방의 스트리머의 음성 및 영상 스트림을 자신이 있는 방에 게시할 수 있으며, 동시에 자신의 음성 및 영상 스트림을 대상 스트리머의 방에 게시할 수 있습니다. 이렇게 하면 서로 다른 두 방에 있는 스트리머들이 크로스 룸 음성 및 영상 스트림을 공유할 수 있어서 각 방의 시청자들이 두 스트리머의 음성 및 영상을 볼 수 있습니다.



위 그림은 크로스 룸 연결 PK의 주요 프로세스를 보여줍니다. 예를 들어, 101 방의 스트리머 A가 `ConnectOtherRoom()` 을 통해 102 방의 스트리머 B와 크로스 룸 통화를 합니다.

- 101 방의 사용자들은 모두 스트리머 B의 `onRemoteUserEnterRoom(B)` 및 `onUserVideoAvailable(B,true)` 이 두 가지 이벤트 콜백을 받게 되며, 이는 101방의 사용자들이 모두 스트리머 B의 음성 및 영상을 구독할 수 있음을 의미합니다.
- 102 방의 사용자들은 모두 스트리머 A의 `onRemoteUserEnterRoom(A)` 및 `onUserVideoAvailable(A,true)` 이 두 가지 이벤트 콜백을 받게 되며, 이는 102 방의 사용자들이 모

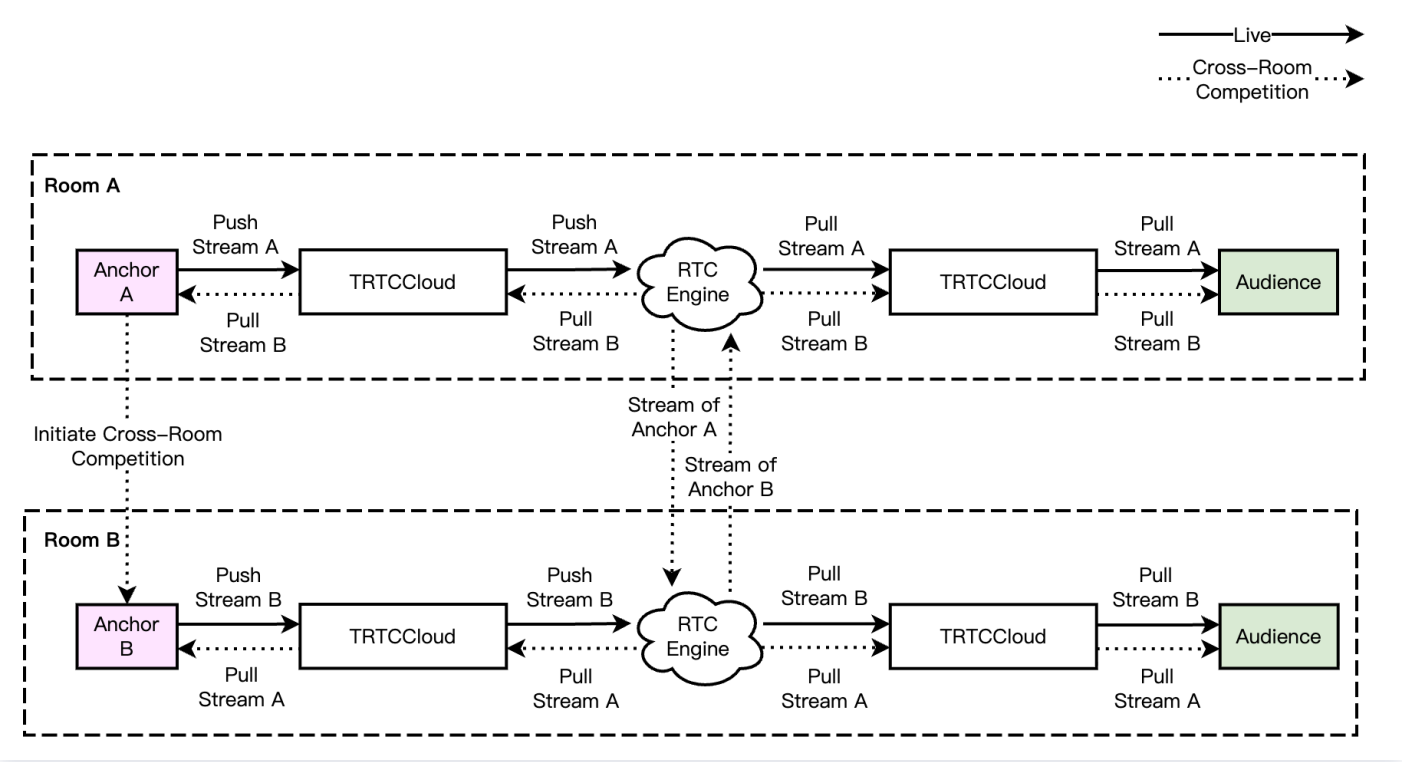
두 스트리머 A의 음성 및 영상을 구독할 수 있음을 의미합니다.

⚠ 주의:

- 크로스 룸 연결 PK의 로컬 사용자와 상대방 사용자는 모두 스트리머 역할로 되어야 하며, 오디오/비디오 업스트림이 모두 있어야 합니다.
- `ConnectOtherRoom()` 을 여러 번 호출하여 여러 방의 스트리머와 크로스 룸 연결을 구현할 수 있습니다. 현재에 한 방은 최대 세 개의 다른 방의 스트리머와의 크로스 룸 연결이 가능하며, 한 방 내에서 최대 10명의 스트리머가 다른 방의 스크리머와 크로스 룸 연결을 할 수 있습니다.
- RTC Engine 크로스 룸 연결 PK는 `createSubCloud()` 을 통해 서브 인스턴스를 생성하여 상대방 방에 가입해서 스트리밍을 주고받는 방식으로도 구현할 수 있으며, 현재 서브 인스턴스의 수량에는 제한이 없어서 향후 여러 방 간 PK 또는 여러 스트리머 간 PK의 업무 확장이 용이해집니다.

2. 실시간 인터랙티브 크로스 룸 연결 프로세스

순수 RTC 시나리오의 크로스 룸 연결 PK 프로세스는 전반적으로 간단합니다. 스트리머와 크로스 룸 연결 스트리머는 서로 RTC 단일 스트림을 풀하고 시청자는 동시에 스트리머와 크로스 룸 연결 스트리머의 RTC 단일 스트림을 풀하며, 시청자는 스트리머와 크로스 룸 연결 스트리머의 미디어 스트림 구독 로직을 독립적으로 제어할 수 있습니다. 실시간 인터랙티브 시나리오의 크로스 룸 연결 프로세스는 아래 그림과 같습니다.



⚠ 주의:
실시간 인터랙티브 크로스 룸 연결 시나리오에서 방 내 시청자는 크로스 룸 연결 스트리머의 미디어 스트림 구독 로직을 독립적으로 제어할 수 있으며 스트리머 가 크로스 룸 스트리머의 본 방 내의 업스트림 능력을 변경할 수도 있습니다.

시나리오 플레이

단일 스트리머 라이브 방송

라이브 방송에 스트리머가 한 명만 있는 시나리오를 단일 스트리머 라이브 방송이라고 합니다. 이 시나리오에서 라이브 방송의 방 주인은 유일한 스트리머입니다. 시청자는 라이브 방송실에 입장하여 방송을 시청하고 방 주인에게 메시지를 보내거나 선물을 할 수 있습니다.

다중 스트리머 라이브방송

다중스트리머 라이브 방송은 라이브 방송에 여러명의 스트리머가 실시간 음성 및 영상 인터랙티브하는 시나리오를 말합니다. 방 주인은 시청자를 마이크 사용 초대하고 마이크 순위를 제어할 수 있으며 시청자도 마이크의 사용을 신청하여 스트리머와 채팅할 수 있습니다.

크로스 룸 PK 라이브방송

라이브 방송실에서 방송 분위기를 조성하고 빠르게 팬을 모으기 위해 방 주인은 다른 라이브 방송실의 방주인을 초대해서 인터랙티브나 온라인 PK를 할 수 있습니다. 연결된 라이브 방송실의 시청자는 두 방 주인을 동시에 시청할 수 있으며 방 주인의 실시간 퍼포먼스에 따라 선물을 보내거나 빠르게 방송실을 전환해서 다른 방 주인에게 투표할 수 있습니다. 이는 전형적인 비디오 PK 연결 시나리오입니다.

이커머스 라이브 커머스

이커머스 라이브방송은 전자상거래와 비디오 인터랙티브 라이브방송가 결합된 시나리오입니다. 라이브 방송에서 방 주인은 시청자에게 상품을 소개하고 상품 목록과 링크를 제공합니다. 시청자는 마음에 드는 상품을 보면 링크를 클릭해서 빠르게 주문할 수 있습니다. 라이브 방송 중 시청자는 마이크를 켜고 방 주인과 실시간으로 채팅할 수 있으며 상품의 상세 정보를 문의하거나 가격을 협상하거나 사용 후기를 공유하는 등 다양한 활동을 할 수 있습니다. 방 주인은 또 다른 라이브 방송실의 방 주인과 PK 연결을 통해 각자의 상품을 보여주고 시청자의 구매 열기를 자극하며 라이브 방송의 재미를 더할 수 있습니다.

방안 부속 제품

시스템 계층	제품명	시나리오 용도
접속 계층	RTC Engine	저지연, 고품질의 다중 사용자 실시간 음성 및 영상 인터랙티브 라이브 방송 솔루션을 제공하며 쇼 라이브방송 시나리오의 기반 인프라 기능입니다.
접속 계층	Chat	기반으로 된 그룹 기능의 방 관리 및 마이크 관리 능력을 제공하며, 라이브 방송 참가자 전체 메시지, 공개 화면 메시지 등의 풍부한 미디어 메시지 송부/수신 및 사용자 자체 정의 시그널링 등 통신 요구를 구현합니다.
접속 계층	뷰티 AR SDK	뷰티, 필터, 메이크업, 재미있는 스티커, Moji 이모티콘, 가상 아바타 등 실시간 특수 효과 처리 기능을 제공합니다.
클라우드 서비스	클라우드 리퀘스트 VOD	음성, 영상, 이미지 등 미디어를 위해 제작 업로드, 저장, 트랜스코딩, 미디어 처리, 미디어 AI, 가속 분배 재생, 저작권 보호 등 일체형의 고품질 미디어 서비스를 제공합니다.
데이터 저장	상대 저장 COS	음성 및 영상 녹화 파일, 음성 및 영상 슬라이스 파일의 저장 서비스를 제공합니다.

고속 접속 가이드
Android

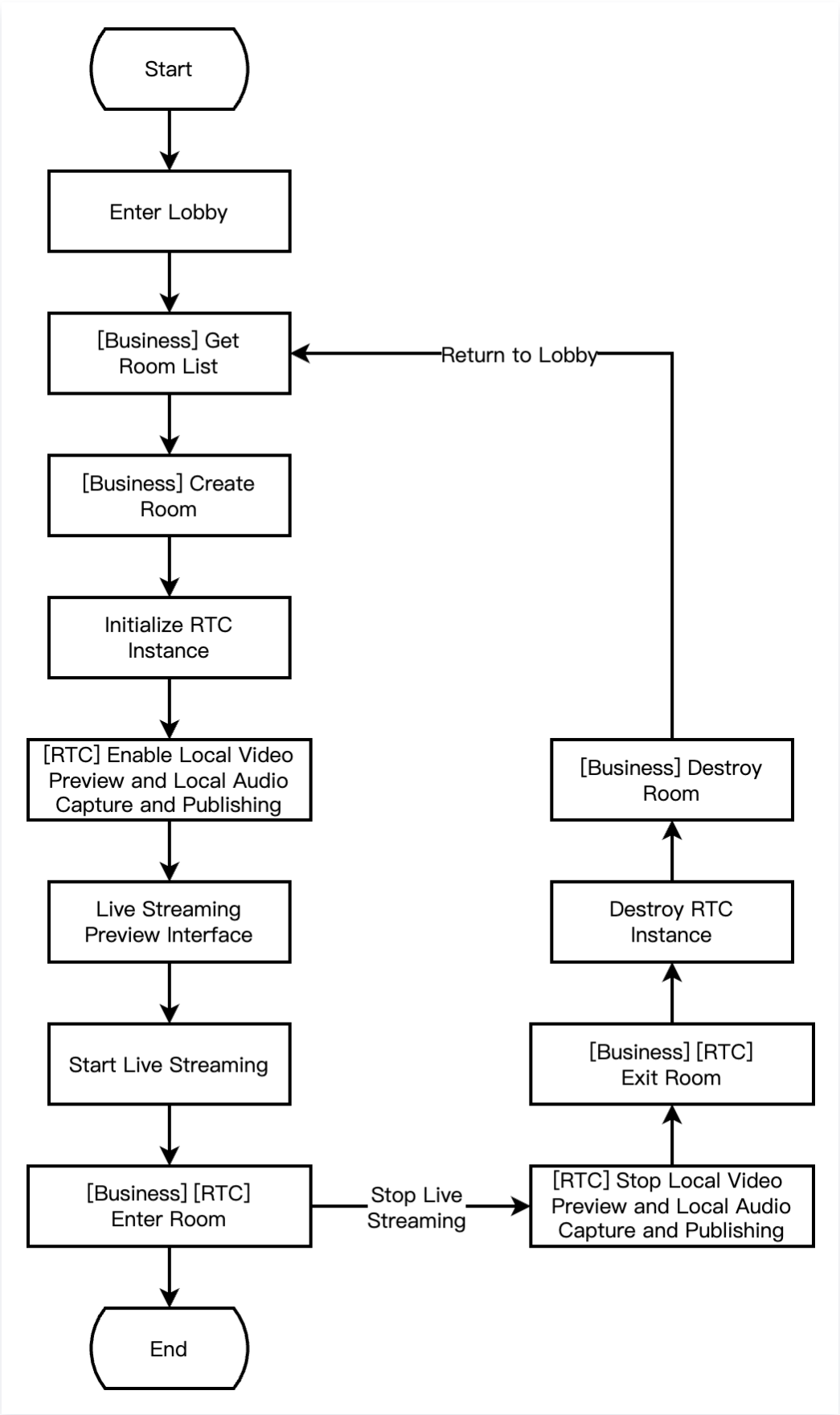
최종 업데이트 날짜:: 2025-10-28 11:32:37

업무 프로세스

본 섹션에서 쇼 라이브 방송에서 자주 사용되는 업무 프로세스를 종합하여 전체 시나리오의 구현 프로세스를 더욱 쉽게 이해할 수 있도록 도와줍니다.

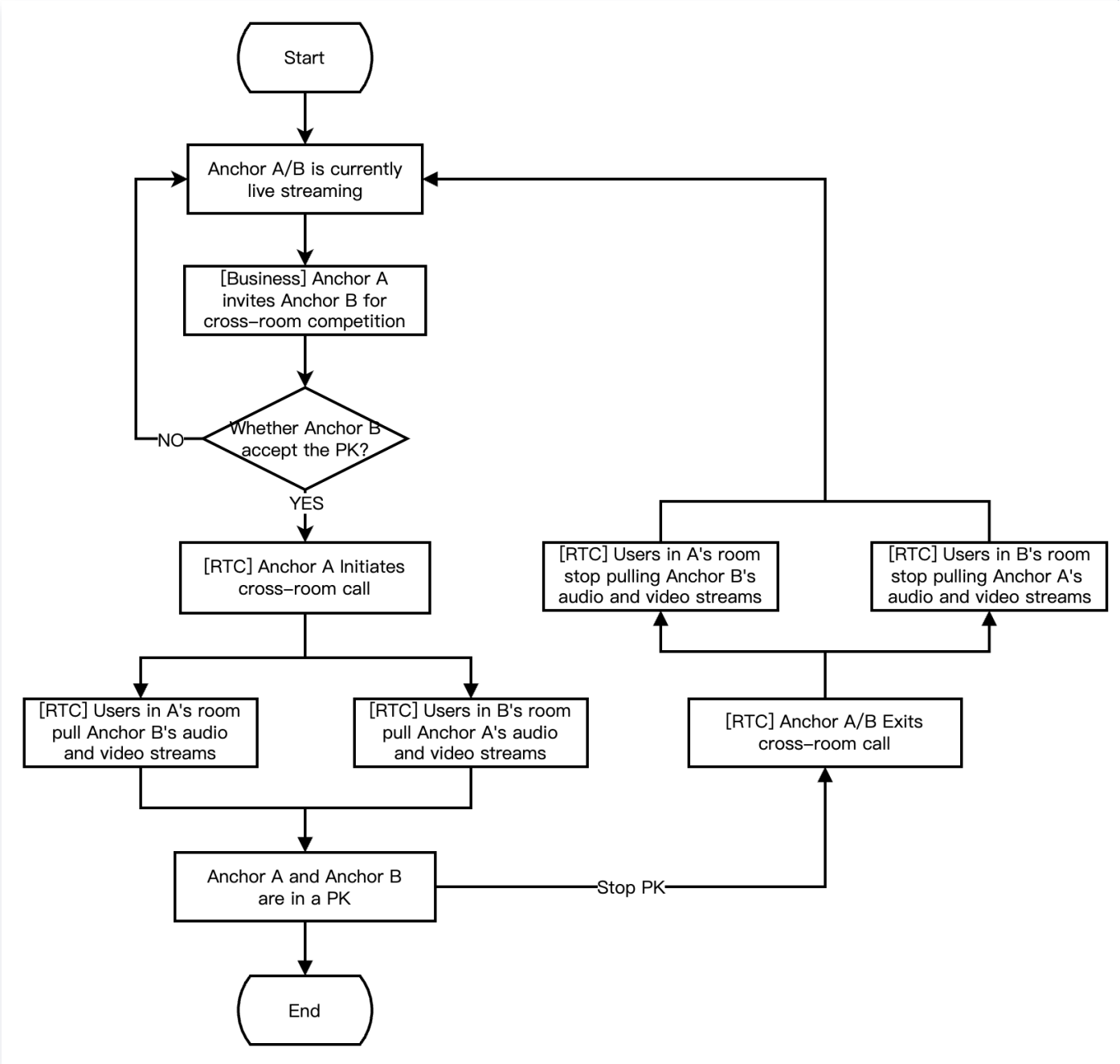
스트리머 방송의 시작 및 종료

아래 그림은 스트리머(방주인)의 로컬 미리보기, 방 생성, 방 입장 및 방송 시작, 방 퇴장 및 방송 종료에 대한 프로세스를 보여줍니다.



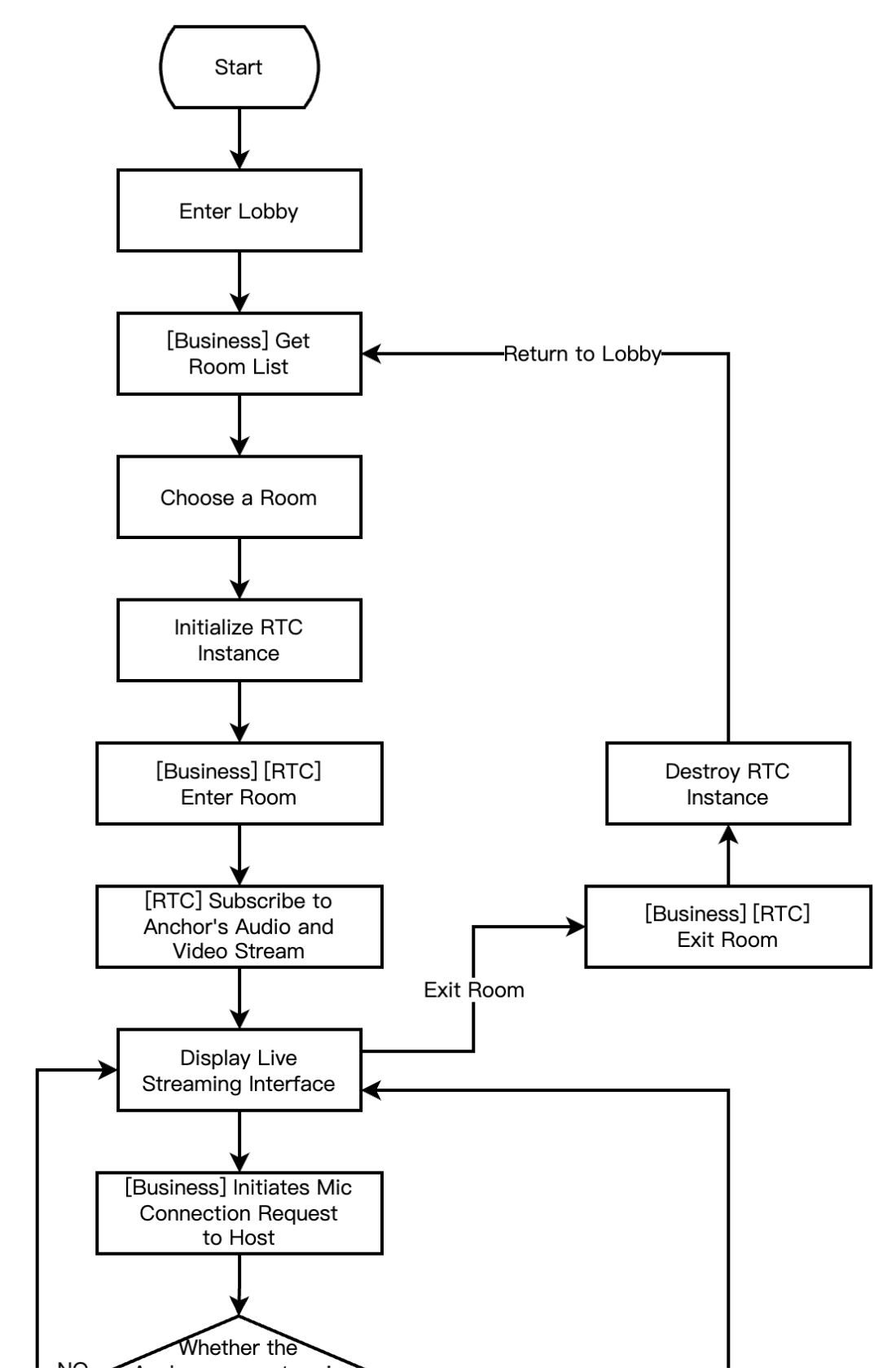
스트리머 간 크로스 룸 PK 연결

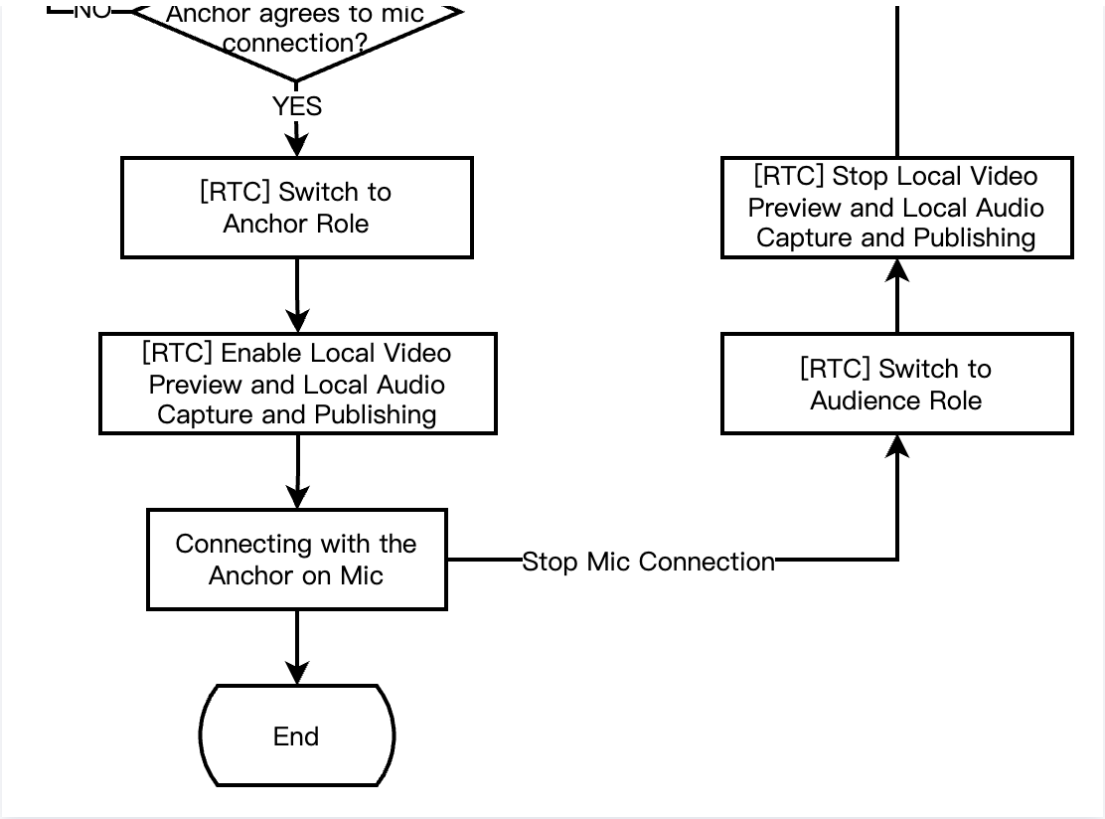
아래 그림으로 스트리머 A가 스트리머 B를 초청하여 크로스 룸 PK 연결을 진행하는 프로세스를 보여줍니다. 크로스 룸 PK 과정에서 두 방의 시청자 모두 두 방주인의 PK 연결 라이브 방송 화면을 볼 수 있습니다.



RTC 시청자 방 입장 후 마이크 연결합니다

아래 그림으로 RTC 실시간 인터랙션 라이브 방송실에서 시청자의 방 입장, 마이크 연결 신청, 연결 종료, 방 퇴장 프로세스를 보여줍니다.





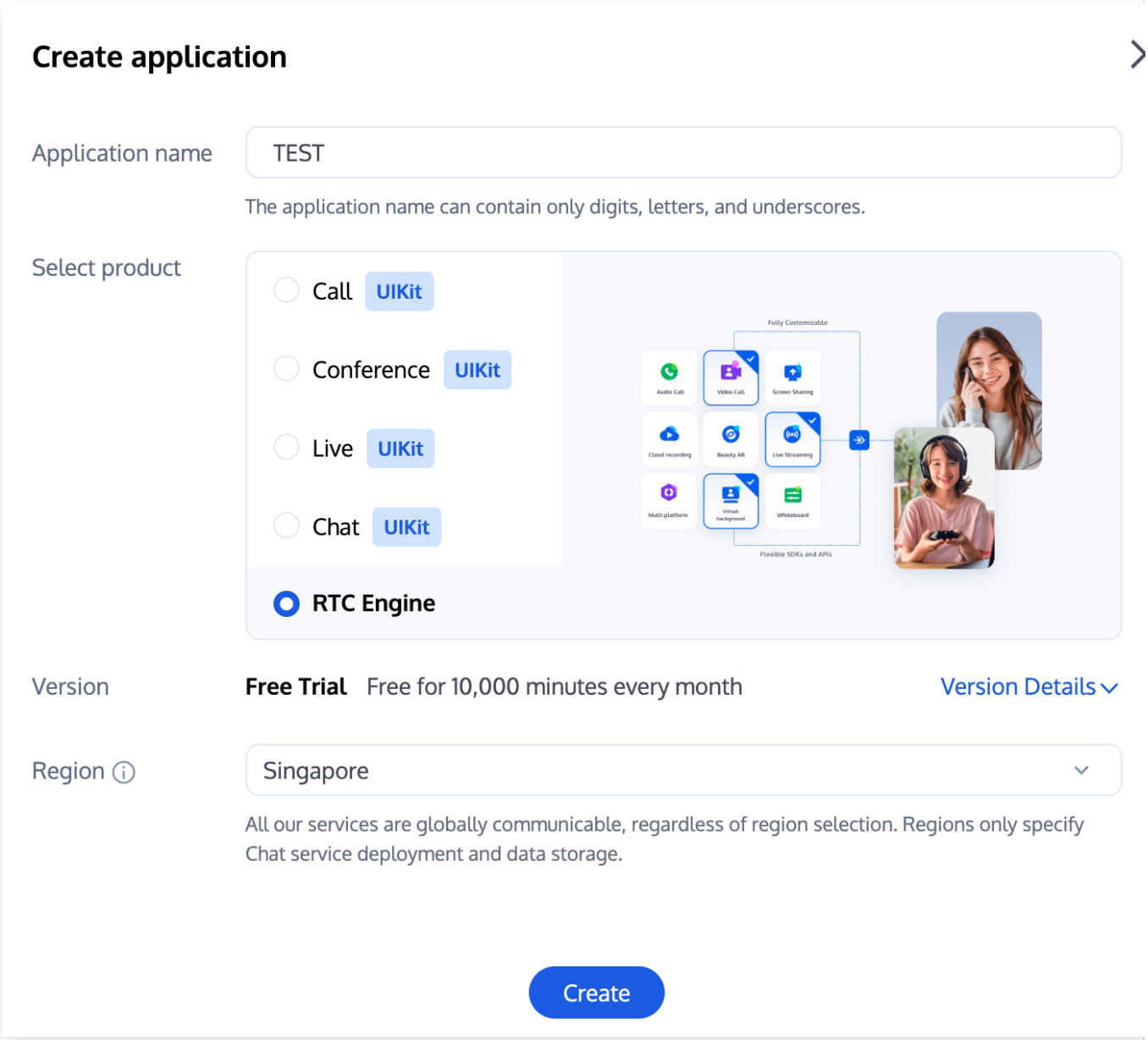
접근 준비

단계1: 서비스 개통

쇼 라이브방송 시나리오는 일반적으로 RTC Engine과 뷰티 AR 두 가지 유료 PaaS 서비스를 활용하여 구축됩니다. 이 중에 RTC Engine 실시간 음성 및 비디오 상호 작용 기능을 제공하고, 뷰티 AR은 뷰티 효과 기능을 제공합니다. 타사 뷰티 제품을 사용하는 경우 뷰티 AR 통합 부분은 무시해도 됩니다.

RTC Engine 서비스의 개통

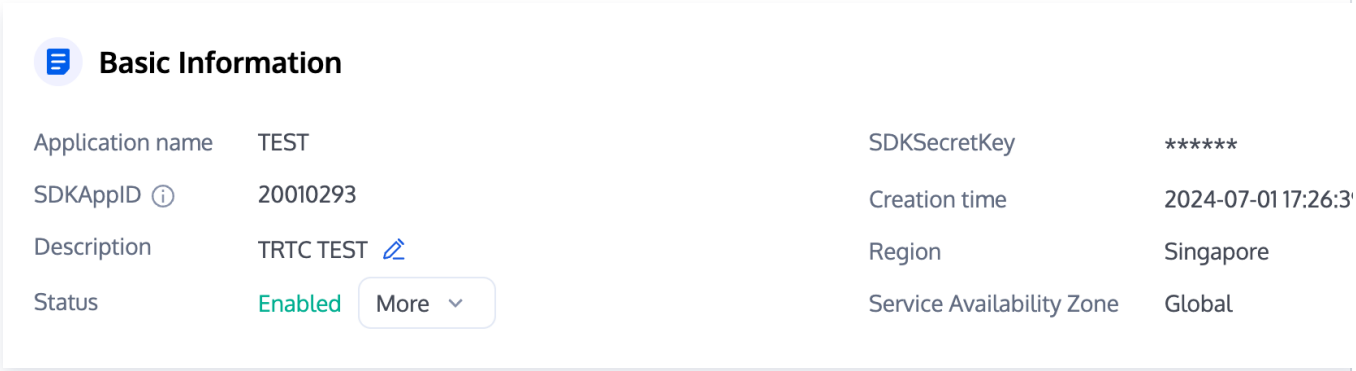
1. 먼저 RTC Engine 콘솔에 로그인하여 애플리케이션을 생성해야 합니다. 필요에 따라 RTC Engine 애플리케이션 버전을 업그레이드할 수 있으며, 예를 들어 프로페셔널 버전은 더 많은 부가 기능 서비스를 이용할 수 있습니다.



설명:

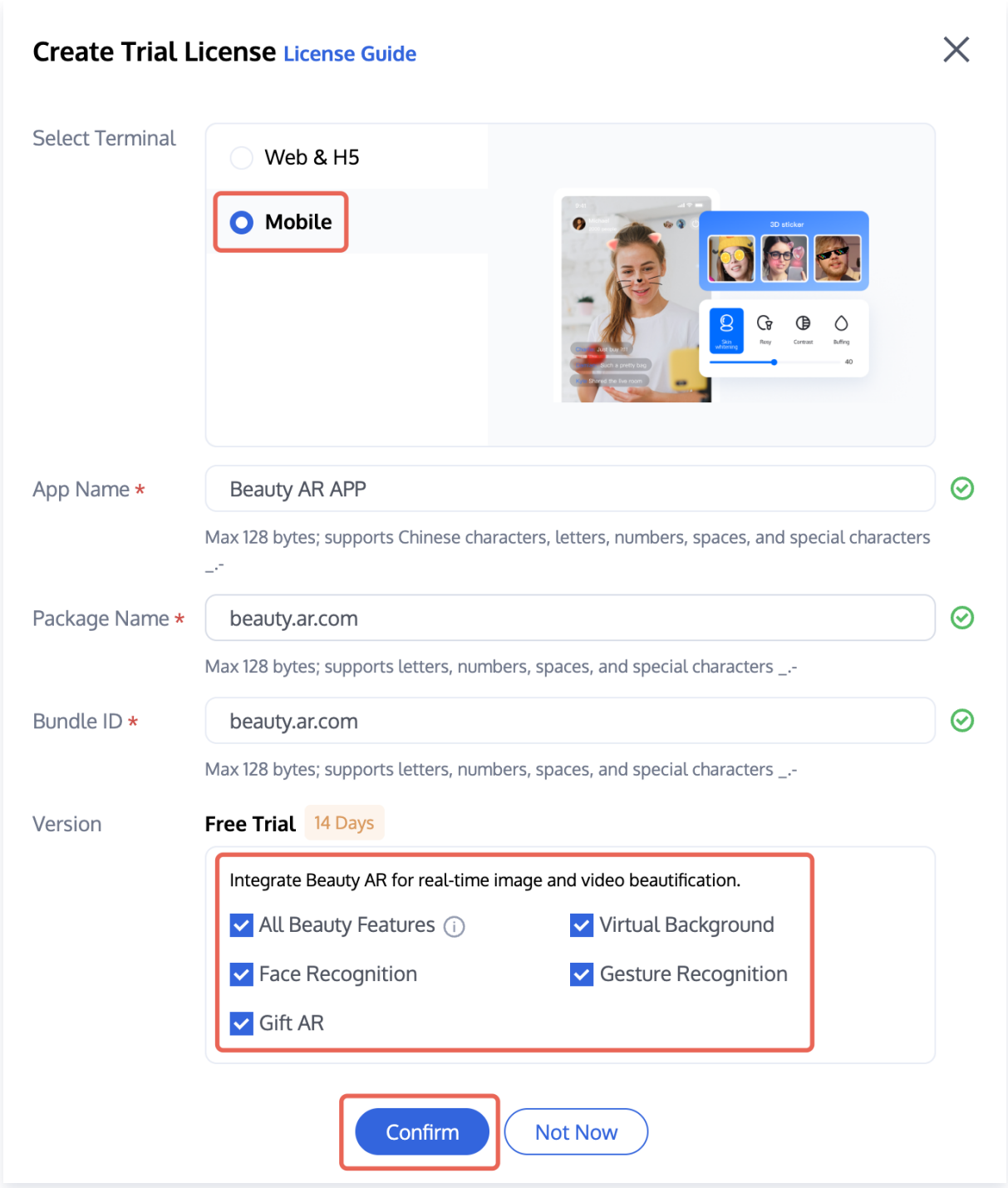
- 두 개의 애플리케이션을 생성하고 각각 테스트 환경과 프로덕션 환경에 적용하는 것을 권장하며, 1년 동안 각 텐센트 클라우드 계정(UIN)에 매월 10,000분의 무료 사용 시간이 제공됩니다.
- RTC Engine 월정액 요금제는 체험판(기본), 라이트, 스탠다드, 프로페셔널로 구분되며, 각각 다른 부가 기능 서비스를 이용할 수 있습니다. 자세한 내용은 버전 기능 및 월정액 요금제 설명을 참조하세요.

2. 애플리케이션 생성이 완료되면, 앱 관리 – 앱 개요 섹션에서 해당 애플리케이션의 기본 정보를 확인할 수 있습니다. 향후 사용을 위해 **SDKAppID**와 **SDKSecretKey**를 안전하게 보관해야 하며, 키 유출로 인한 트래픽 도용을 방지해야 합니다.

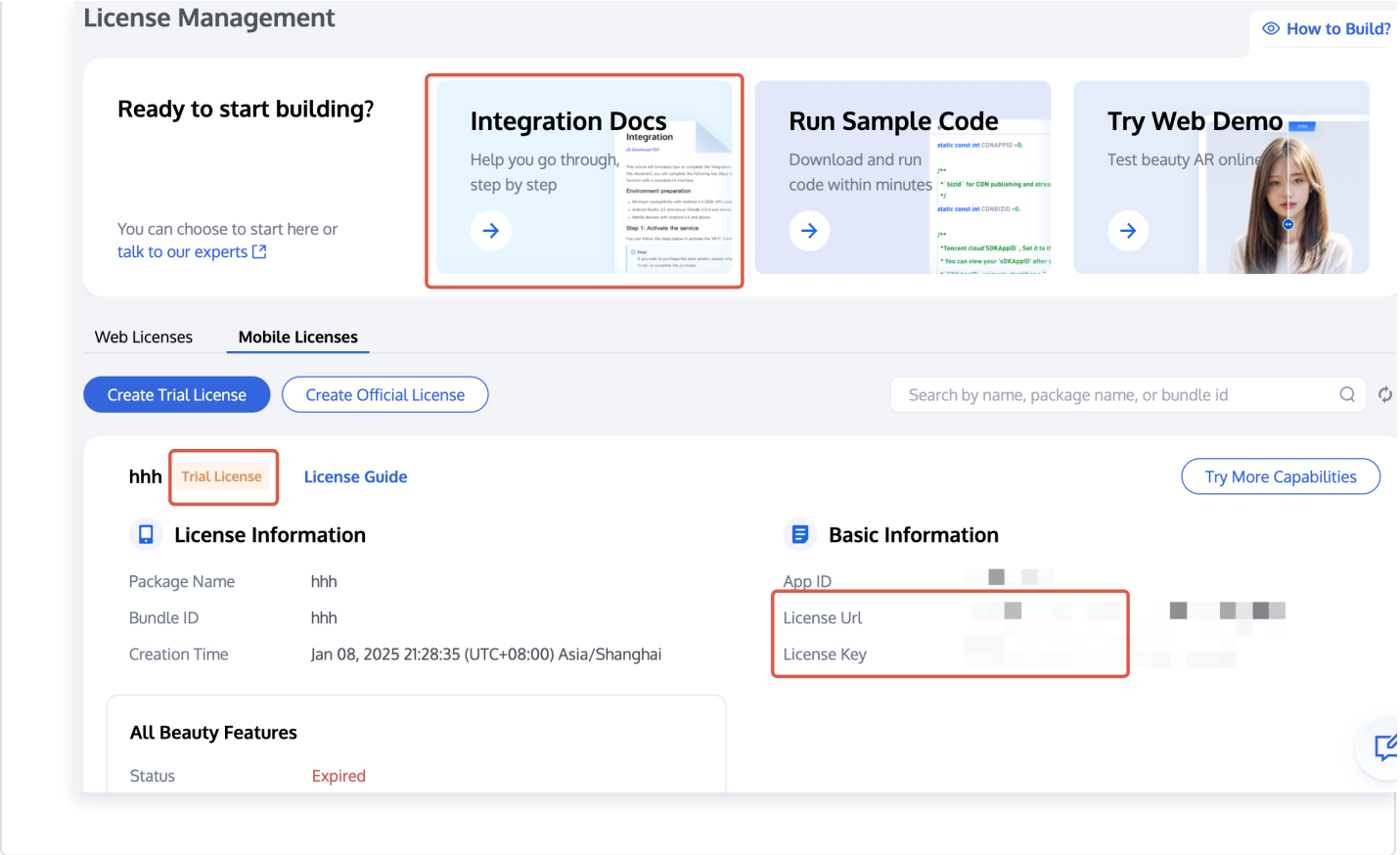


뷰티 AR 서비스의 개통

1. 뷰티 AR 콘솔 > 모바일 라이선스에 로그인한 후 **을 클릭하여 테스트 라이선스** 을 생성하세요(테스트비전 라이선스는 무료로 14일간 유효하며, 1회 연장 가능하여 총 28일 사용할 수 있습니다). 모바일을 선택하고 실제 필요에 따라 앱 이름, 패키지 이름, 번들 ID를 입력하세요. 사용해 보고 싶은 기능을 선택하세요: **모든 뷰티 기능**, **가상 배경**, **얼굴 인식**, **제스처 인식**, **선물 애니메이션 효과**, 그런 다음에 **확인**을 클릭하세요.



2. 활성화 후 현재 페이지에서 본인의 정보를 확인하고 상단의 통합 가이드를 참조하여 통합시킬 수 있습니다. 통합 가이드에서 License Key와 License URL 사용 방법을 확인할 수 있습니다.



단계2: SDK 임포트하기

RTC Engine SDK와 뷰티 AR SDK가 **mavenCentral** 라이브러리에 송부되었으며, gradle을 구성하여 자동으로 다운로드 및 업데이트할 수 있습니다.

1. dependencies에 적합한 버전의 SDK 종속성을 추가합니다.

```
dependencies {
    // RTC Engine 경량판 SDK에 RTC 엔진과 라이브 방송 재생 두 가지 기능을 포함하며, 크기가 작습니다.
    implementation 'com.tencent.liteav:LiteAVSDK_TRTC:latest.release'

    // RTC Engine 전체 기능판 SDK에 라이브 방송, 숏 비디오, 주문형 비디오 등 다양한 기능을 추가로 포함하며, 크기가 약간 큽니다.
    // implementation 'com.tencent.liteav:LiteAVSDK_Professional:latest.release'

    // 뷰티  AR SDK, 예: S1-07 패키지는 다음과 같음
    implementation 'com.tencent.mediacloud:TencentEffect_S1-07:latest.release'
}
```

> **설명:**
> 추천하는 자동 로드 방식 외에도 SDK를 다운로드하여 수동으로 도입할 수도 있습니다. 자세한 내용은 RTC Engine SDK 수동 통합 및 뷰티 AR SDK 수동 통합을 참조하세요.

2. defaultConfig에서 앱이 사용하는 CPU 아키텍처를 지정합니다.

```
defaultConfig {
      ndk {
            abiFilters "armeabi-v7a", "arm64-v8a"
      }
}
```

> **설명:**
> - RTC Engine SDK는 armeabi-v7a/arm64-v8a 아키텍처를 지원하며, 시뮬레이터 전용 x86/x86_64 아키텍처도 추가로 지원합니다.
> - 뷰티AR SDK는 현재 armeabi-v7a/arm64-v8a 아키텍처만 지원합니다.

3. **Sync Now**를 클릭하면 SDK가 자동으로 다운로드되어 프로젝트에 통합됩니다. 뷰티 AR 패키지에 동적 효과와 필터 기능이 포함된 경우 SDK 다운로드 페이지에서 해당 패키지를 다운로드하고 무료 필터 소재(./assets/lut)와 스티커 동적 효과(./MotionRes)를 압축 해제한 후 프로젝트의 다음 디렉터리에 배치해야 합니다
   - 모션 효과: `../assets/MotionRes`
   - 필터 효과: `../assets/lut`

단계 3: 엔지니어링 구성

1. 권한의 설정.

   AndroidManifest.xml에서 앱 권한을 구성하고, 쇼 라이브방송 시나리오에서 RTC Engine SDK 및 뷰티 AR SDK에는 다음 권한이 필요합니다

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission
android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

> ⚠ 주의:
> - `android:hardwareAccelerated="false"` 를 설정하지 마십시오. 하드웨어 가속을 끄면 상대방의 비디오 스트림이 렌더링되지 않습니다.
> - RTC Engine SDK에는 내장된 권한 요청 로직이 없으므로 해당 권한과 기능을 직접 선언해야 합니다. 일부 권한(예: 저장, 녹음, 카메라 등)은 런타임에 동적으로 요청해야 합니다.
> - Android 프로젝트의 `targetSdkVersion` 이 31이거나 대상 기기가 Android 12및 그 이상 시스템 버전과 관련된 경우, 블루투스 기능을 정상적으로 사용하기 위해 코드에서 `android.permission.BLUETOOTH_CONNECT` 권한을 동적으로 신청해야 합니다. 자세한 내용은 블루투스 권한를 참조하십시오.

2. 혼동 구성.

SDK 내부에서 Java의 리플렉션 기능을 사용하기 때문에 proguard-rules.pro 파일에서 SDK 관련 클래스를 혼동 제외 목록에 추가해야 합니다.

```
-keep class com.tencent.** { *; }
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class androidx.exifinterface.** { *;}
```

단계4: 인증 및 허가

RTC Engine 인증 자격 증명

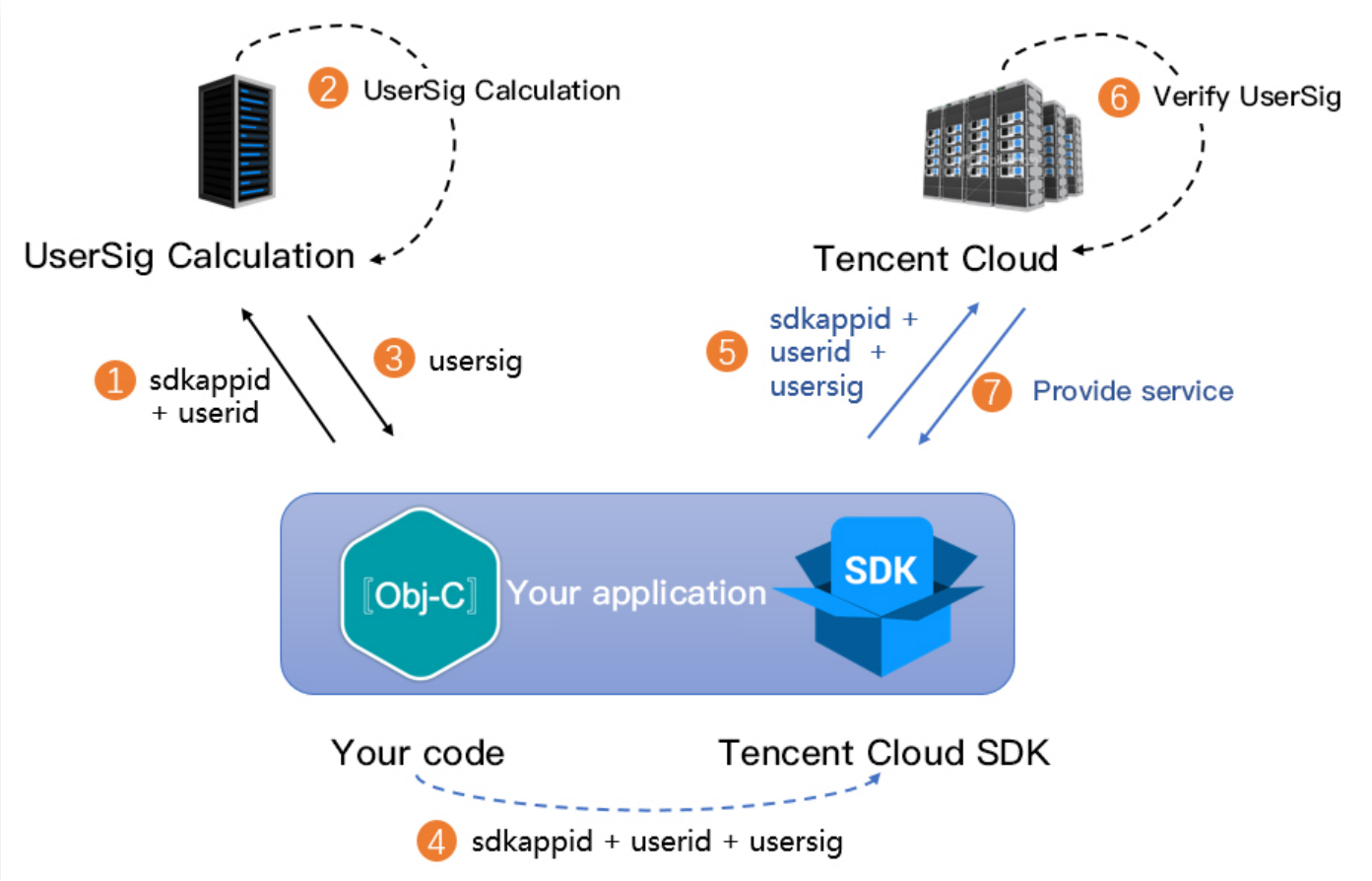
UserSig는 텐센트 클라우드에서 설계한 보안 서명으로, 악의적인 공격자가 클라우드 서비스 사용 권한을 도용하는 것을 방지하기 위한 것입니다. RTC Engine은 방 입장 시 인증에 사용됩니다.

- 디버깅 및 테스트 단계: 클라이언트 예시 코드와 콘솔에서 가져오기 두 가지 방법으로 UserSig를 계산 및 생성할 수 있으며, 디버깅 및 테스트 용도로만 사용됩니다.
- 정식 운영 단계: 클라이언트가 역공학으로 키가 유출되는 것을 방지하기 위해 보안 등급이 더 높은 서버 UserSig 계산 방식의 사용을 권장합니다.

구체적인 구현 프로세스는 다음과 같습니다.

1. 손님의 App은 SDK 초기화 함수를 호출하기 전에 먼저 서버에 UserSig를 요청해야 합니다.
2. 손님의 서버는 SDKAppID와 UserID에 따라 UserSig를 계산합니다.

3. 서버는 계산된 UserSig를 손님의 App에 반환합니다.
4. 손님의 App은 획득한 UserSig를 특정 API를 통해 SDK에 전달합니다.
5. SDK는 SDKAppID + UserID + UserSig를 텐센트 클라우드 서버에 제출하여 검증합니다.
6. 텐센트 클라우드는 UserSig를 검증하여 합법성을 확인합니다.
7. 검증이 통과되면 RTC Engine SDK에 실시간 음성 및 영상 서비스를 제공합니다.



⚠ 주의:

- 디버깅 단계의 로컬 UserSig 계산 방식은 온라인 환경에 적용하는 것을 권장하지 않으며, 역공학으로 인해 키가 유출될 수 있기 때문입니다.
- 여러 언어 버전(Java/GO/PHP/Node.js/Python/C#/C++)의 UserSig 서버 계산원 코드를 제공하며, 자세한 내용은 서버 계산 UserSig 을 참조하세요.

뷰티 AR 인증 허가

뷰티 AR 효과를 사용하기 전에 텐센트 클라우드에서 허가 증명에 대한 검증 받아야 합니다. License 에 사용되는 License Key와 License URL를 설정하며, 예시 코드는 다음과 같습니다.

```
import com.tencent.xmagic.telicense.TELicenseCheck;

// 라이선스dml 다운로드 또는 업데이트만을 목적으로 하고 인증 결과에 관심이 없는 경우, 네 번째 매개변수에 null을 전달합니다.
TELicenseCheck.getInstance().setTELicense(context, URL, KEY, new TELicenseCheck.TELicenseCheckListener() {
    @Override
    public void onLicenseCheckFinish(int errorCode, String msg) {
        // 주의: 이 콜백은 호출 스레드에서 실행되지 않을 수 있습니다
        if (errorCode == TELicenseCheck.ERROR_OK) {
            // 인증 성공
        } else {
            // 인증 실패
        }

    }
});
```

⚠ 주의:

- 관련 업무 모듈의 초기화 코드에서 인증 허가를 트리거하고 사용하기 전에 임시로 라이선스를 다운로드하는 것을 피하며, 인증 시에는 네트워크 권한이 있어야 합니다.
- 실제 적용된 패키지 이름은 라이선스 생성 시 바인딩된 Package Name과 완전히 일치해야 하며, 그렇지 않으면 라이선스 검증이 실패할 수 있습니다. 자세한 내용은 인증 오류 코드을 참조하세요.

단계5: SDK 초기화

RTC Engine SDK의 초기화

```
// RTC Engine SDK 인스턴스의 생성(싱글톤 모드)
TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(context);
// 이벤트 리스너의 설정
```

```
mTRTCCloud.addListener(trtcSdkListener);

// SDK의 다양한 이벤트 알림(예: 오류 코드, 경고 코드, 오디오/비디오 상태 매개
변수 등)
private TRTCCloudListener trtcSdkListener = new TRTCCloudListener()
{
    @Override
    public void onError(int errCode, String errMsg, Bundle
extraInfo) {
        Log.d(TAG, errCode + errMsg);
    }

    @Override
    public void onWarning(int warningCode, String warningMsg, Bundle
extraInfo) {
        Log.d(TAG, warningCode + warningMsg);
    }
};

// 이벤트 리스너의 제거
mTRTCCloud.removeListener(trtcSdkListener);
// RTC Engine SDK 인스턴스(싱글톤 모드)를 파기합니다.
TRTCCloud.destroySharedInstance();
```

설명:
SDK 이벤트 알림을 모니터링하고 일반적인 오류에 대한 로그 출력 및 처리를 권장합니다. 자세한 내용은 오류 코드 테이블을 참조하세요.

뷰티 AR SDK의 초기화

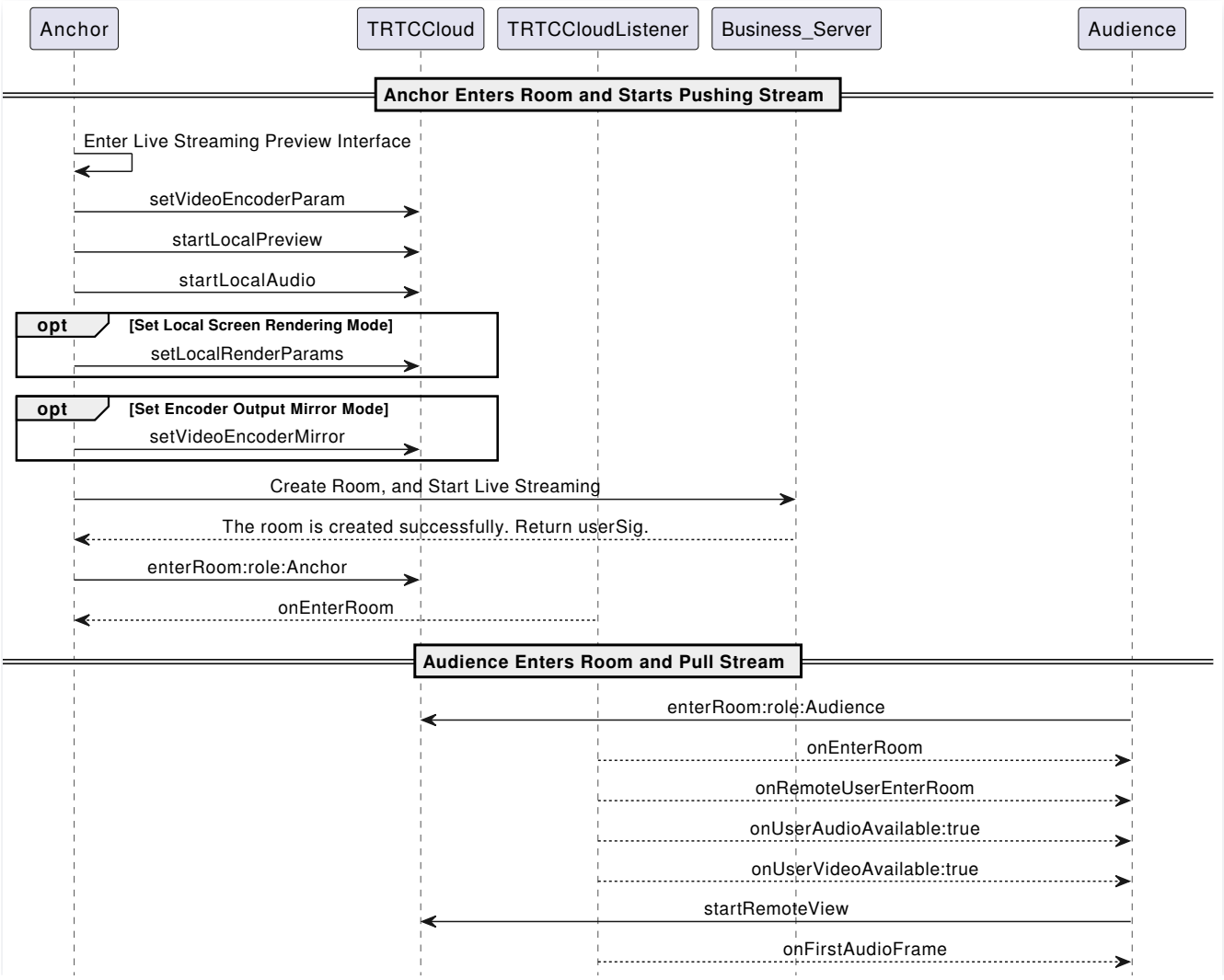
```
import com.tencent.xmagic.XmagicApi;

// 뷰티 SDK 초기화
XmagicApi mXmagicApi = new XmagicApi(context,
XmagicResParser.getResPath(), new
XmagicApi.OnXmagicPropertyErrorListener());
```

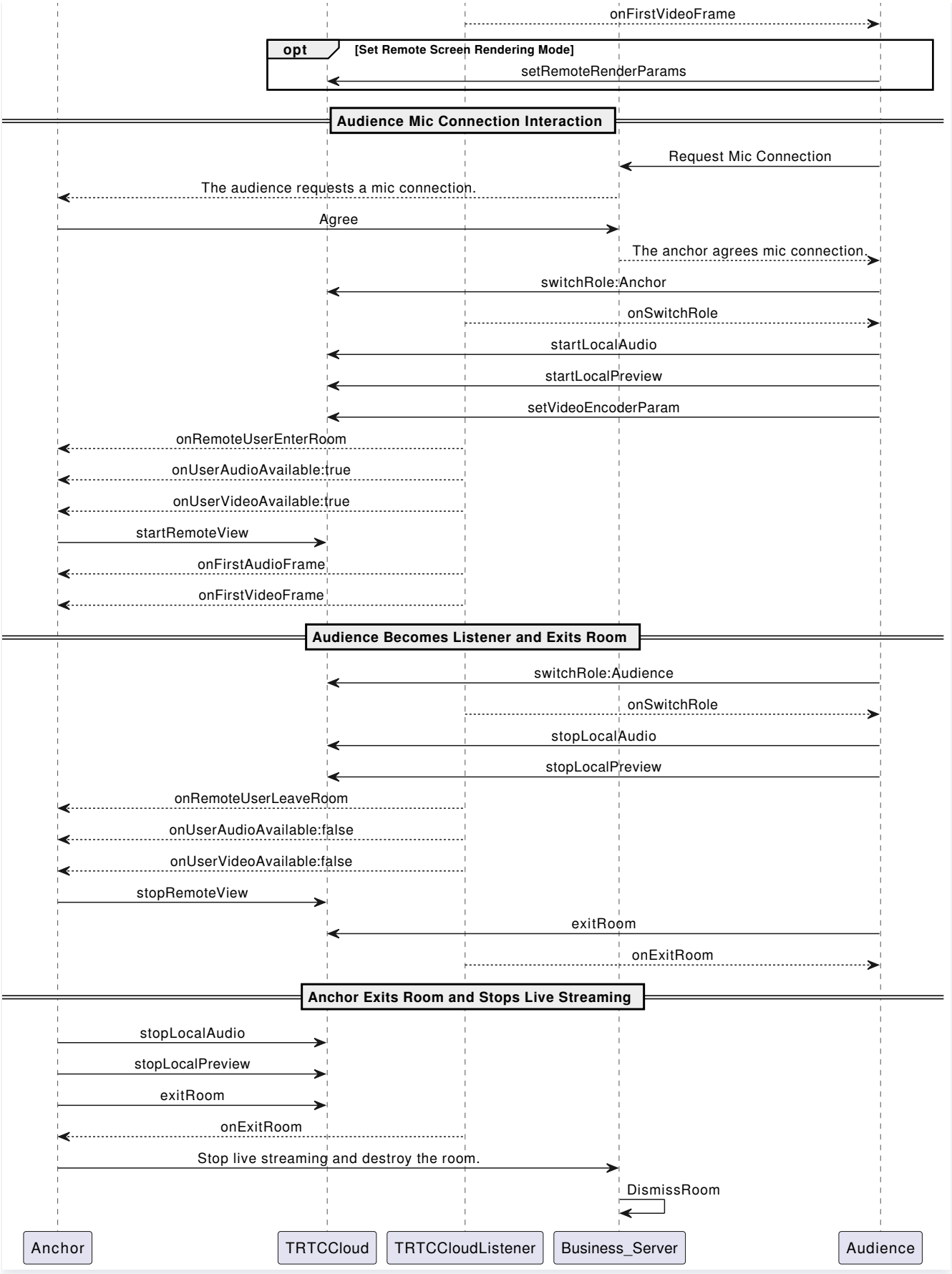
```
// 개발 및 디버깅 시 로그 등급을 DEBUG로 설정하며 릴리스 패키지는 WARN으로 설정하세요. 그렇지 않으면 성능에 영향을 미칠 수 있습니다.
mXmagicApi.setXmagicLogLevel(Log.WARN);

// 뷰티 SDK를 릴리스하는 것은  GL 스레드에서 호출해야 합니다
mXmagicApi.onDestroy();
```

설명:
뷰티 AR SDK를 초기화하기 전에 리소스 복사 등의 준비가 필요하며, 자세한 단계는 뷰티 AR SDK 사용 절차 를 참조하십시오.

접속 과정

API 시퀀스 다이어그램





단계1: 스트리머가 방에 입장하여 스트리밍을 시작합니다

RTC Engine SDK는 비디오 화면을 표시하는 컨트롤이고 `TXCloudVideoView` 유형만 전달할 수 있으므로, 먼저 레이아웃 파일에 뷰 렌더링 컨트롤을 정의해야 합니다.

```
<com.tencent.rtmp.ui.TXCloudVideoView
    android:id="@+id/live_cloud_view_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

> ⚠ 주의:
> `TextureView` 또는 `SurfaceView`를 뷰 렌더링 컨트롤로 지정하려면 고급 기능–뷰 렌더링 컨트롤을 참조하십시오.

1. 스트리머가 방에 들어가기 전에 로컬 비디오 미리보기 및 오디오 수집을 시작합니다.

```
// 스트리머의 로컬 화면 미리보기를 표시하는 데 사용되는 비디오 렌더링 컨트롤 가져
옵니다
TXCloudVideoView mTxcvvAnchorPreviewView =
findViewById(R.id.live_cloud_view_main);

// 비디오 인코딩 매개변수 설정은 원격 사용자가 보는 화면의 화질을 결정합니다
TRTCCloudDef.TRTCVideoEncParam encParam = new
TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_960_540;
encParam.videoFps = 15;
encParam.videoBitrate = 1300;
encParam.videoResolutionMode =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
mTRTCCloud.setVideoEncoderParam(encParam);

// boolean mIsFrontCamera는 전면/후면 카메라의 사용을 지정하여 비디오를 수집
합니다.
mTRTCCloud.startLocalPreview(mIsFrontCamera,
mTxcvvAnchorPreviewView);

// 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
```

⚠ 주의:
- 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam를 직접 설정할 수 있으며, 각 등급별 최적 해상도 및 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.
- `enterRoom` 전에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집만 시작하며, `enterRoom` 을 호출한 후에야 스트리밍을 시작합니다.
- `enterRoom` 후에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집을 시작하고 자동으로 스트리밍을 시작합니다.

2. 스트리머가 로컬 화면의 렌더링 매개변수 및 인코더 출력 화면 모드(선택 사항)를 설정합니다.

```
TRTCCloudDef.TRTCRenderParams params = new
TRTCCloudDef.TRTCRenderParams();
params.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_AUTO;    // 화
면 미러 모드
params.fillMode = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;      // 화
면 채우기 모드
params.rotation = TRTCCloudDef.TRTC_VIDEO_ROTATION_0;            // 화
면 회전 각도
// 로컬 화면의 렌더링 매개변수를 설정합니다
mTRTCCloud.setLocalRenderParams(params);

// 인코더 출력 화면 미러 모드를 설정합니다
mTRTCCloud.setVideoEncoderMirror(boolean mirror);
// 비디오 인코더 출력 화면의 방향을 설정합니다
mTRTCCloud.setVideoEncoderRotation(int rotation);
```

⚠ 주의:
- 로컬 화면 렌더링 파라미터의 설정은 로컬 화면의 렌더링 효과에만 영향을 줍니다.
- 인코더 출력 모드의 설정은 방의 다른 사용자들이 보는 (및 클라우드 레코딩 파일) 화면 효과에 영향을 줍니다.

3. 스트리머가 정식으로 라이브 방송을 시작하고 방에 입장하여 스트리밍 시작합니다.

```
public void enterRoomByAnchor(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
```

```
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = getUserSig(userId);
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = SDKAppID;
    // 스트리머 역할 지정
    params.role = TRTCCloudDef.TRTCRoleAnchor;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

// 방 입장 결과 이벤트의 콜백
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result는 방에 임장하는 데 소요된 시간(밀리초)을 나타냅니다.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result는 방 입장 실패의 오류 코드를 나타냅니다.
        Log.d(TAG, "Enter room failed");
    }
}
```

⚠ 주의:

- RTC Engine 방 번호는 정수형 `roomId` 와 문자열 형식 `strRoomId` 로 나뉘며, 두 유형의 방은 서로 통하지 않으므로 방 번호 유형을 통일하는 것이 좋습니다.
- RTC Engine 사용자 역할은 스트리머와 청취자로 구분되며, 스트리머만 스트리밍 권한을 가집니다. 방 입장 시 사용자 역할을 지정해야 하며, 지정하지 않으면 기본적으로 스트리머 역할로 설정됩니다.
- 쇼 라이브방송 시나리오에서는 방 입장 모드를 `TRTC_APP_SCENE_LIVE` 로 선택하는 것이 좋습니다.

단계2: 시청자가 방에 들어가서 스트리밍을 시작합니다

1. 시청자가 RTC Engine 방에 입장합니다.

```
public void enterRoomByAudience(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
```

```
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = getUserSig(userId);
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = SDKAppID;
    // 시청자 역할의 지정
    params.role = TRTCCloudDef.TRTCRoleAudience;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

// 방 입장 결과 이벤트의 콜백
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result는 방에 임장하는 데 소요된 시간(밀리초)을 나타냅니다.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result는 방 입장 실패의 오류 코드를 나타냅니다.
        Log.d(TAG, "Enter room failed");
    }
}
```

2. 시청자가 스트리머의 오디오 및 비디오 스트림을 구독합니다.

```
@Override
public void onUserAudioAvailable(String userId, boolean available) {
    // 원격 사용자가 자신의 오디오를 게시/취소합니다
    // 자동 구독 모드에서는 사용자가 아무런 작업을 하지 않아도 SDK가 원격 사용자
의 오디오를 자동으로 재생합니다.
}

@Override
public void onUserVideoAvailable(String userId, boolean available) {
    // 원격 사용자가 메인 비디오 화면을 게시/취소합니다
    if (available) {
```

```
        // 원격 사용자의 비디오 스트림을 구독하고 비디오 렌더링 컨트롤을 바인딩
합니다
        mTRTCCloud.startRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, TXCloudVideoView view);
    } else {
        // 원격 사용자의 비디오 스트림 구독을 중지하고 렌더링 컨트롤을 릴리스합
니다
        mTRTCCloud.stopRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
    }
}
```

3. 시청자가 원격 화면의 렌더링 모드를 설정합니다(선택 사항).

```
TRTCCloudDef.TRTCRenderParams params = new
TRTCCloudDef.TRTCRenderParams();
params.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_AUTO;    // 화
면 미러 모드
params.fillMode = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;      // 화
면 채우기 모드
params.rotation = TRTCCloudDef.TRTC_VIDEO_ROTATION_0;            // 화
면 회전 각도
// 원격 화면의 렌더링 모드를 설정합니다
mTRTCCloud.setRemoteRenderParams(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, params)
```

단계3: 시청자와의 실시간 인터랙션

1. 시청자가 스트리머 역할로 전환됩니다.

```
// 스트리머 역할로 전환됩니다.
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);

// 역할 전환 이벤트의 콜백
@Override
public void onSwitchRole(int errCode, String errMsg) {
    if (errCode == TXLiteAVCode.ERR_NULL) {
        // 역할 전환 성공
    }
```

```
}
```

2. 시청자가 로컬 오디오/비디오 수집 및 스트리밍을 시작합니다.

```
// 연결된 시청자의 로컬 화면 미리보기를 표시하기 위한 비디오 렌더링 컨트롤을 가져
옵니다.
TXCloudVideoView mTxcvvAudiencePreviewView =
findViewById(R.id.live_cloud_view_sub);

// 비디오 인코딩 매개변수 설정은 원격 사용자가 보는 화면의 화질을 결정합니다
TRTCCloudDef.TRTCVideoEncParam encParam = new
TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_480_270;
encParam.videoFps = 15;
encParam.videoBitrate = 550;
encParam.videoResolutionMode =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
mTRTCCloud.setVideoEncoderParam(encParam);

// boolean mIsFrontCamera는 전면/후면 카메라의 사용을 지정하여 비디오를 수집
합니다.
mTRTCCloud.startLocalPreview(mIsFrontCamera,
mTxcvvAudiencePreviewView);

// 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
```

> ⚠ **주의:**
> 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam을 직접 설정할 수 있으며, 각 등급별 최적 해상도 및 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.

3. 시청자가 마이크를 끄고 스트리밍을 중지합니다.

```
// 시청자 역할로 전환됩니다
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
```

```
// 역할 전환 이벤트의 콜백
@Override
public void onSwitchRole(int errCode, String errMsg) {
    if (errCode == TXLiteAVCode.ERR_NULL) {
        // 카메라 수집 및 스트리밍 중지합니다
        mTRTCCloud.stopLocalPreview();
        // 마이크 수집 및 스트리밍 중지합니다
        mTRTCCloud.stopLocalAudio();
    }
}
```

단계4: 방 나가기 및 해산하기

1. 방에서 나가기.

```
public void exitRoom() {
    mTRTCCloud.stopLocalAudio();
    mTRTCCloud.stopLocalPreview();
    mTRTCCloud.exitRoom();
}

// 방 나가기 이벤트의 콜백
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "exitRoom을 호출하여 방에서 나가기");
    } else if (reason == 1) {
        Log.d(TAG, "서버에 의해 현재 방에서 나가게 됩니다");
    } else if (reason == 2) {
        Log.d(TAG, "현재 방 전체가 해체됩니다");
    }
}
```

⚠ 주의:

- SDK가 점유한 모든 리소스가 릴리스된 후, SDK는 `onExitRoom` 콜백을 통해 알려줍니다.
- `enterRoom` 을 다시 호출하거나 다른 음성/영상 SDK로 전환하려면 `onExitRoom` 콜백이 발생한 후 관련 작업을 수행하십시오. 그렇지 않으면 카메라, 마이크등 장치가 강제로 점유된 것과 같은 다양한 이상 현상이 발생할 수 있습니다.

2. 방 해산
   - **서버 측 해산:** RTC Engine 서버 측 방 해산 API `DismissRoom` (숫자 방 ID와 문자열 방 ID 구분)을 제공하며, 이 인터페이스를 호출하여 방의 모든 사용자를 방에서 내보내고 방을 해산할 수 있습니다.
   - **클라이언트 측 해산:** 각 클라이언트의 방 나가기 `exitRoom` 인터페이스를 통해 방 내 모든 스트리머와 청취자를 방에서 내보낼 수 있습니다.퇴장 후 RTC Engine 방 생명주기 규칙에 따라 방이 자동으로 해산됩니다. 자세한 내용은 방 나가기를 참조하세요.

> ⚠ 주의:
> 라이브방송이 종료된 후 서버 측에서 방 해산 API를 호출하여 방이 해산되도록 하는 것이 좋습니다. 이는 시청자가 의도치 않게 방에 들어와 예상치 못한 비용이 발생하는 것을 방지하기 위함입니다.

고급 기능

스트리머 간 크로스 룸 PK 연결

1. 어느 한쪽이 크로스 룸 PK 연결을 시작합니다.

```
public void connectOtherRoom(String roomId, String userId) {
    try {
        JSONObject jsonObj = new JSONObject();
        // 숫자 방 번호는 roomId입니다
        jsonObj.put("strRoomId", roomId);
        jsonObj.put("userId", userId);
        mTRTCCloud.ConnectOtherRoom(jsonObj.toString());
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

// 크로스 룸 연결 결과의 콜백
@Override
public void onConnectOtherRoom(String userId, int errCode, String
errMsg) {
    // 크로스 룸 연결하려는 다른 방의 스트리머의 사용자 ID
    // 에러 코드, ERR_NULL은 요청 성공을 나타냅니다
    // 에러 정보
}
```

> ⚠ 주의:

> 크로스 룸 PK 연결의 로컬 사용자와 상대 사용자는 모두 스트리머 역할이어야 하며, 오디오 또는 비디오 업스트림이 모두 있어야 합니다.

2. 두 방의 모든 사용자는 다른 방의 스크리머로부터 오디오/비디오 스트림 사용 가능 콜백을 받게 됩니다.

```
@Override
public void onUserAudioAvailable(String userId, boolean available) {
    // 원격 사용자가 자신의 오디오를 게시/취소합니다
    // 자동 구독 모드에서는 사용자가 아무런 작업을 하지 않아도 SDK가 원격 사용자
의 오디오를 자동으로 재생합니다.
}

@Override
public void onUserVideoAvailable(String userId, boolean available) {
    // 원격 사용자가 메인 비디오 화면을 게시/취소합니다
    if (available) {
        // 원격 사용자의 비디오 스트림을 구독하고 비디오 렌더링 컨트롤을 바인딩
합니다
        mTRTCCloud.startRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, TXCloudVideoView view);
    } else {
        // 원격 사용자의 비디오 스트림 구독을 중지하고 렌더링 컨트롤을 릴리스합
니다
        mTRTCCloud.stopRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
    }
}
```

3. 어느 한쪽이 크로스 룸 PK 연결을 종료합니다.

```
// 크로스 룸 채팅 연결 종료
mTRTCCloud.DisconnectOtherRoom();

// 크로스 룸 연결 종료 결과의 콜백
@Override
public void onDisConnectOtherRoom(int errCode, String errMsg) {
    super.onDisConnectOtherRoom(errCode, errMsg);
}
```

⚠ 주의:

- `DisconnectOtherRoom()` 을 호출하면 모든 다른 방의 스트리머와의 크로스 룸 PK 연결이 종료됩니다.
- 크로스 룸 PK 연결의 발신 측 또는 수신 측 중 어느 한쪽에서 `DisconnectOtherRoom()` 을 호출하여 크로스 룸 PK 연결을 종료할 수 있습니다.

제3자 뷰티 제품의 사용

RTC Engine은 서드파티 뷰티 효과 제품의 접근을 지원하며, 아래에서는 뷰티 AR을 예로 들어 서드파티 뷰티 접근 절차를 설명합니다.

1. 뷰티 AR SDK 통합 및 라이선스 권한의 신청에 대한 자세한 내용은 접속 준비 단계를 참조하세요.
2. 리소스 복사(있는 경우). 리소스 파일이 assets 디렉터리에 내장된 경우 사용 전에 App의 개인 디렉터리로 복사해야 합니다.

```
XmagicResParser.setResPath(new File(getFilesDir(),
"xmagic").getAbsolutePath());
//loading

//개인 디렉터리로 리소스 파일을 복사하며, 한 번만 수행하면 됩니다.
XmagicResParser.copyRes(getApplicationContext());
```

리소스 파일이 네트워크 동적 다운로드에서 가져온 경우 다운로드가 완료된 후 리소스 파일 경로를 설정해야 합니다.

```
XmagicResParser.setResPath(다운로드한 리소스 파일의 로컬 경로);
```

3. 제3자 뷰티의 비디오 데이터 콜백을 설정하고, 뷰티 SDK가 처리한 프레임별 데이터 결과를 RTC Engine SDK 내부로 전달하여 렌더링 처리를 수행합니다.

```
mTRTCCloud.setLocalVideoProcessListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_
FORMAT_Texture_2D, TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new
TRTCCloudListener.TRTCVideoFrameListener() {
    @Override
    public void onGLContextCreated() {
        // SDK 내부 OpenGL 환경이 생성되었으며, 이때 제3자 뷰티의 초기화 작업
을 진행할 수 있습니다.
        if (mXmagicApi == null) {
```

```
            XmagicApi mXmagicApi = new XmagicApi(context,
XmagicResParser.getResPath(), new
XmagicApi.OnXmagicPropertyErrorListener());
        } else {
            mXmagicApi.onResume();
        }
    }

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame
srcFrame, TRTCCloudDef.TRTCVideoFrame dstFrame) {
        // 제3자 뷰티 컴포넌트와 연동하기 위한 비디오 처리의 콜백
        if (mXmagicApi != null) {
            dstFrame.texture.textureId =
mXmagicApi.process(srcFrame.texture.textureId, srcFrame.width,
srcFrame.height);
        }
        return 0;
    }

    @Override
    public void onGLContextDestory() {
        // SDK 내부 OpenGL 환경이 파기되었으며, 이때 제3자 뷰티 리소스 파기
작업을 진행할 수 있습니다.
        mXmagicApi.onDestroy();
    }
});
```

> ⚠ 주의:
> 단계1 및 단계2는 제3자 뷰티 제품의 구현 방식에 따라 다르며, **단계3**은 RTC Engine이 제3자 뷰티를 통합하는 **공통적이고 중요한 단계**입니다.

듀얼 인코딩 모드

듀얼 인코딩 모드를 활성화하면 현재 사용자의 인코더가 [고화질 대형 화면]과 [저화질 소형 화면] 두 가지 비디오 스트림을 동시에 출력합니다(단, 오디오 스트림은 하나만 존재함). 이렇게 하면 방의 다른 사용자가 자신의 네트워크 상황이나 화면 크기에 따라 [고화질 대형 화면] 또는 [저화질 소형 화면]을 선택하여 구독할 수 있습니다.

1. 대형및 소형 화면 듀얼 인코딩 모드를 활성화합니다.

```
public void enableDualStreamMode(boolean enable) {
    // 소형 스트림의 비디오 인코딩 매개변수(자체 정의 가능)
    TRTCCloudDef.TRTCVideoEncParam smallVideoEncParam = new
TRTCCloudDef.TRTCVideoEncParam();
    smallVideoEncParam.videoResolution =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_480_270;
    smallVideoEncParam.videoFps = 15;
    smallVideoEncParam.videoBitrate = 550;
    smallVideoEncParam.videoResolutionMode =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
    mTRTCCloud.enableEncSmallVideoStream(enable, smallVideoEncParam);
}
```

> ⚠ 주의:
> 듀얼 인코딩을 활성화하면 더 많은 CPU 및 네트워크 대역폭이 소모되므로 Mac, Windows 또는 고성능 Pad에서는 활성화를 고려할 수 있지만 모바일에서는 활성화하지 않는 것이 좋습니다.

2. 원격 사용자 비디오 스트림의 유형을 선택합니다.

```
// 원격 사용자 비디오 스트림을 구독할 때 선택 가능한 비디오 스트림의 유형
mTRTCCloud.startRemoteView(userId, streamType, videoView);

// 원격 사용자의 대/소 화면을 언제든지 전환할 수 있습니다
mTRTCCloud.setRemoteVideoStreamType(userId, streamType);
```

> ⚠ 주의:
> 듀얼 스트림 인코딩 활성화 후, `streamType` 비디오 스트림 유형을 `TRTC_VIDEO_STREAM_TYPE_SMALL` 으로 지정하여 저화질 소형 화면을 가져올 수 있습니다.

뷰 렌더링 컨트롤

RTC Engine에는 비디오 화면을 관리하는 인터페이스가 많으며, 이러한 인터페이스는 모두 비디오 렌더링 컨트롤을 지정해야 합니다. Android 플랫폼에서는 `TXCloudVideoView` 를 비디오 렌더링 컨트롤로 사용하며, `SurfaceView` 와 `TextureView` 두 가지 렌더링 방식을 지원합니다. 아래에서는 렌더링 컨트롤 유형을 지정하는 방법과 비디오 렌더링 컨트롤을 업데이트하는 방법을 소개합니다.

1. 특정 방식을 강제로 사용하거나 로컬 비디오 렌더링 컨트롤을 `TXCloudVideoView` 으로 변환하려면 다음과 같이 코딩할 수 있습니다.

```
// TextureView를 강제로 사용합니다
TextureView textureView = findViewById(R.id.texture_view);
TXCloudVideoView cloudVideoView = new TXCloudVideoView(context);
cloudVideoView.addVideoView(textureView);

// SurfaceView를 강제로 사용합니다
SurfaceView surfaceView = findViewById(R.id.surface_view);
TXCloudVideoView cloudVideoView = new TXCloudVideoView(surfaceView);
```

2. 업무에 표시 영역 전환과 관련된 인터랙티브 시나리오가 포함된 경우 RTC Engine SDK를 사용하여 로컬 미리보기 화면을 업데이트하고 원격 사용자 비디오 렌더링 컨트롤 기능을 구현할 수 있습니다.

```
// 로컬 미리보기 화면 렌더링 컨트롤의 업데이트
mTRTCCloud.updateLocalView(videoView);

// 원격 사용자 비디오 렌더링 컨트롤의 업데이트
mTRTCCloud.updateRemoteView(userId, streamType, videoView);
```

> ⚠ 주의:
> 매개변수 `videoView` 를 대상 비디오 렌더링 컨트롤로 전달하고, `streamType` 은 `TRTC_VIDEO_STREAM_TYPE_BIG` 및 `TRTC_VIDEO_STREAM_TYPE_SUB` 만 지원됩니다.

라이브 방송의 인터랙티브 메시지

라이브 방송의 인터랙티브는 라이브 방송 시나리오에서 특히 중요하며, 사용자는 좋아요 메시지, 선물 메시지, 탄막 메시지 등을 통해 스트리머와 인터랙티브 할 수 있습니다 . 라이브방송 인터랙티브 기능을 구현하기 위해서는 Chat 서비스를 활성화하고 Chat SDK를 임포트해야 합니다. 자세한 안내는 음성 채팅방 접속 가이드–접속 준비를 참조하십시오.

좋아요 메시지

1. 좋아요를 누른 사용자는 클라이언트에서 좋아요 관련 그룹 커스텀 메시지를 전송하며, 전송 성공 후 업무 측에서 로컬에 좋아요 효과를 렌더링합니다.

```
// 좋아요 메시지 본문 구성
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("cmd", "like_msg");
    JSONObject msgJsonObject = new JSONObject();
```

```
    msgJsonObject.put("type", 1);         // 좋아요 유형
    msgJsonObject.put("likeCount", 10);   // 좋아요 수량
    jsonObject.put("msg", msgJsonObject);
} catch (JSONException e) {
    e.printStackTrace();
}
String data = jsonObject.toString();

// 그룹 커스텀 메시지 전송 (좋아요 메시지는 낮은 우선순위로 설정 권장드립니다)
V2TIMManager.getInstance().sendGroupCustomMessage(data.getBytes(),
mRoomId,
        V2TIMMessage.V2TIM_PRIORITY_LOW, new
V2TIMValueCallback<V2TIMMessage>() {
            @Override
            public void onError(int i, String s) {
                // 좋아요 메시지 전송 실패
            }

            @Override
            public void onSuccess(V2TIMMessage v2TIMMessage) {
                // 좋아요 메시지 전송 성공
                // 로컬에서  좋아요 효과 렌더링합니다
            }
        });
```

2. 방 내의 다른 사용자의 클라이언트는 그룹 커스텀 메시지 콜백을 수신한 후 메시지 파싱 및 좋아요 효과 렌더링을 수행합니다.

```
// 그룹 커스텀 메시지의 수신
V2TIMManager.getInstance().addSimpleMsgListener(new
V2TIMSimpleMsgListener() {
    @Override
    public void onRecvGroupCustomMessage(String msgID, String
groupID, V2TIMGroupMemberInfo sender, byte[] customData) {
        String customStr = new String(customData);
        if (!customStr.isEmpty()) {
            try {
                JSONObject jsonObject = new JSONObject(customStr);
                String command = jsonObject.getString("cmd");
```

```
                JSONObject messageJsonObject =
jsonObject.getJSONObject("msg");
                if (command.equals("like_msg")) {
                    int type = messageJsonObject.getInt("type");
// 좋아요 유형
                    int likeCount =
messageJsonObject.getInt("likeCount");  // 좋아요 수량
                    // 좋아요 유형과 수량에 따라 좋아요 효과 렌더링합니다
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
});
```

선물 메시지

1. 선물 보내는 사람이 업무 서버에 요청을 보내면 업무 서버가 결제 정산을 완료한 후 REST API를 호출하여 그룹에 커스텀 메시지를 전송합니다.

   1.1 URL 요청의 예시.

```
https://xxxxxx/v4/group_open_http_svc/send_group_msg?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&conten
ttype=json
```

   1.2 패킷 요청의 예시.

```
{
    "GroupId": "@TGS#12DEVUDHQ",
    "Random": 2784275388,
    "MsgPriority": "High",  // 메시지의 우선순위이고 선물 메시지는 높은 우선
순위로 설정해야 합니다
    "MsgBody": [
        {
            "MsgType": "TIMCustomElem",
            "MsgContent": {
                // type: 선물 유형; giftUrl: 선물 리소스 주소; giftName:
선물 이름; giftCount: 선물 수량
```

```
                "Data": "{""cmd"": ""gift_msg"", ""msg"": {""type"":
1, ""giftUrl"": ""xxx"", ""giftName"": ""xxx"", ""giftCount"": 1}}"
            }
        }
    ]
}
```

2. 방 내의 다른 사용자 클라이언트는 그룹 커스텀 메시지 콜백을 수신한 후 메시지 파싱 및 선물 효과 렌더링을 수행합니다.

```
// 그룹 커스텀 메시지의 수신
V2TIMManager.getInstance().addSimpleMsgListener(new
V2TIMSimpleMsgListener() {
    @Override
    public void onRecvGroupCustomMessage(String msgID, String
groupID, V2TIMGroupMemberInfo sender, byte[] customData) {
        String customStr = new String(customData);
        if (!customStr.isEmpty()) {
            try {
                JSONObject jsonObject = new JSONObject(customStr);
                String command = jsonObject.getString("cmd");
                JSONObject messageJsonObject =
jsonObject.getJSONObject("msg");
                if (command.equals("gift_msg")) {
                    int type = messageJsonObject.getInt("type");
// 선물 유형
                    int giftCount =
messageJsonObject.getInt("giftCount");       // 선물 수량
                    String giftUrl =
messageJsonObject.getString("giftUrl");     // 선물 리소스 주소
                    String giftName =
messageJsonObject.getString("giftName");   // 선물 이름
                    // 선물 유형, 선물 수량, 선물 리소스 주소, 선물 이름에 따
라 선물 효과를 렌더링합니다.
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
```

```
    }
});
```

탄막 메시지

쇼 라이브 방송에는 일반적으로 텍스트 형식의 탄막 메시지가 나오며, 여기서는 Chat의 전송 및 그룹 채팅 일반 텍스트 메시지 수신을 통해 구현할 수 있습니다.

```
// 공개 탄막 메시지의 전송
V2TIMManager.getInstance().sendGroupTextMessage(text, groupID,
V2TIMMessage.V2TIM_PRIORITY_NORMAL, new V2TIMValueCallback<V2TIMMessage>
() {
    @Override
    public void onError(int i, String s) {
        // 탄막 메시지의 전송이 실패됩니다
    }

    @Override
    public void onSuccess(V2TIMMessage v2TIMMessage) {
        // 탄막 메시지의 전송이 성공됩니다
        // 로컬에서 메시지 텍스트가 표시됨
    }
});

// 공개 탄막 메시지의 수신
V2TIMManager.getInstance().addSimpleMsgListener(new
V2TIMSimpleMsgListener() {
    @Override
    public void onRecvGroupTextMessage(String msgID, String groupID,
V2TIMGroupMemberInfo sender, String text) {
        // 발신자 정보 sender와 메시지 텍스트 text에 따라 탄막 메시지 렌더링합니
다
    }
});
```

⚠ 주의:

- 선물 메시지는 높은 우선순위로 설정하는 것을 권장합니다. 탄막 메시지는 일반 우선순위로 설정하며, 좋아요 메시지는 낮은 우선순위로 설정하는 것을 권장합니다.
- 본인이 클라이언트에서 그룹 채팅 메시지를 보내면 메시지 수신 콜백이 트리거되지 않으며 그룹 내의

다른 사용자만 수신할 수 있습니다.

이상 처리

고장및 오류 처리

RTC Engine SDK에서 복구할 수 없는 오류가 발생하면 `onError` 콜백에서 나오며, 자세한 내용은 오류 코드표을 참조하십시오.

1. UserSig 관련. UserSig 검증 실패로 인해 방 입장에 실패할 수 있으며 UserSig 도구를 사용하여 검증할 수 있습니다.

열거형	값	설명
ERR_TRTC_INVALID_USER_SIG	-3320	입장 매개변수 UserSig가 올바르지 않습니다. `TRTCParams.userSig` 이 비어 있는지 확인하세요.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	UserSig 검증에 실패했습니다. 매개변수 `TRTCParams.userSig` 이 올바르게 입력되었거나 만료되지 않았는지 확인하세요.

2. 방 입장 실패 시 먼저 입장 매개변수가 올바른지 확인하고 입장 및 퇴장 인터페이스는 반드시 쌍으로 호출해야 합니다. 입장에 실패한 경우에도 퇴장 인터페이스를 호출해야 합니다.

열거형	값	설명
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	입장 요청이 시간 초과했습니다, 네트워크 연결이 끊겼는지 또는 VPN이 켜져 있는지 확인하세요. 4G로 전환하여 테스트할 수도 있습니다.
ERR_TRTC_INVALID_SDK_APPID	-3317	입장 매개변수 sdkAppId가 잘못되었습니다. `TRTCParams.sdkAppId` 이 비어 있는지 확인하세요.
ERR_TRTC_INVALID_ROOM_ID	-3318	입장 매개변수 roomId가 올바르지 않습니다. `TRTCParams.roomId` 또는 `TRTCParams.strRoomId` 이 비어 있는지 확인하세요. roomId와 strRoomId는 혼용할 수 없습니다.
ERR_TRTC_INVALID_USER_ID	-3319	입장 매개변수 userId가 올바르지 않습니다. `TRTCParams.userId` 이 비어 있는지 확인하세요.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	입장 요청이 거부되었습니다. `enterRoom` 을 연속적으로 호출하여 동일한 ID의 방에 입장했는지 확인하세요.

3. 장치 관련. 장치 관련 오류를 감지할 수 있으며, 관련 오류 발생 시 사용자에게 UI로 알립니다.

열거형	값	설명
ERR_CAMERA_START_FAIL	-1301	Windows 또는 Mac 장치에서 카메라 구성 프로그램(드라이버)이 비정상일 경우, 카메라 열 수가 없습니다. 장치를 비활성화한 후 다시 활성화하거나, 기기를 재시작하거나, 구성 프로그램을 업데이트하세요.
ERR_MIC_START_FAIL	-1302	Windows 또는 Mac 장치에서 마이크 구성 프로그램(드라이버)에 오류가 발생하여 마이크를 열지 못한 경우, 장치를 비활성화한 후 다시 활성화하거나 기기를 재시작하거나 구성 프로그램을 업데이트하세요.
ERR_CAMERA_NOT_AUTHORIZED	-1314	카메라 장치에 권한이 없습니다. 일반적으로 모바일 장치에서 발생하며, 사용자가 권한을 거부했을 수 있습니다.
ERR_MIC_NOT_AUTHORIZED	-1317	마이크 장치에 권한이 없습니다. 일반적으로 모바일 장치에서 발생하며, 사용자가 권한을 거부했을 수 있습니다.
ERR_CAMERA_OCCUPY	-1316	카메라가 사용 중입니다. 다른 카메라를 열어 볼 수 있습니다.
ERR_MIC_OCCUPY	-1319	마이크가 사용 중입니다. 예를 들어 모바일 장치에서 통화 중일 때 마이크를 열 수가 없습니다.

원격 미러 모드의 무효 문제

RTC Engine 설정 화면 미러는 로컬 미리보기 미러 `setLocalRenderParams` 과 비디오 인코더 미러 `setVideoEncoderMirror` 으로 나뉘며, 각각 로컬 미리보기 화면의 미러 효과와 비디오 인코딩 출력 화면의 미러 효과(원격 시청자 및 클라우드 레코딩의 미러링 모드)에 영향을 줍니다. 로컬 미리보기의 미러 효과가 원격 시청자 측에도 동시에 적용되도록 하려면 다음과 같이 코딩하십시오.

```
// 로컬 화면의 렌더링 매개변수를 설정합니다
TRTCCloudDef.TRTCRenderParams params = new
TRTCCloudDef.TRTCRenderParams();
params.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_ENABLE; // 화면
미러 모드
params.fillMode = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;      // 화면
채우기 모드
params.rotation = TRTCCloudDef.TRTC_VIDEO_ROTATION_0;            // 화면
회전 각도
mTRTCCloud.setLocalRenderParams(params);
```

```
// 인코더 출력 화면 미러 모드를 설정합니다
mTRTCCloud.setVideoEncoderMirror(true);
```

카메라 줌/초점/전환 문제

쇼 라이브 방송 시나리오에서 스트리머는 카메라에 대한 사용자 커스텀 필요할 수 있으며, RTC Engine SDK 장치 관리 클래스에는 이러한 요구 사항을 해결하기 위한 관련 인터페이스가 있습니다.

1. 카메라 줌 배율의 조회 및 설정.

```
// 카메라 최대 줌 배율 가져오기(모바일 전용)
float zoomRatio =
mTRTCCloud.getDeviceManager().getCameraZoomMaxRatio();
// 카메라 줌 배율의 설정(모바일 전용)
// 범위는 1-5이며, 1은 가장 먼 시야(일반 렌즈)이고 5는 가장 가까운 시야(확대 렌즈)를 나타냅니다. 최대값은 5를 권장하며 5를 초과하면 비디오 데이터가 흐릿해질 수 있습니다
mTRTCCloud.getDeviceManager().setCameraZoomRatio(zoomRatio);
```

2. 카메라 초점 기능 및 위치의 설정.

```
// 카메라 자동 초점 기능 켜기 또는 끄기(모바일 전용)
mTRTCCloud.getDeviceManager().enableCameraAutoFocus(false);
// 카메라 초점 위치의 설정(모바일 전용)
// 해당 인터페이스를 사용하려면 먼저 enableCameraAutoFocus를 통해 자동 초점 기능을 꺼야 합니다
mTRTCCloud.getDeviceManager().setCameraFocusPosition(int x, int y);
```

3. 전면 또는 후면 카메라의 판단 및 전환.

```
// 현재 전면 카메라인지 판단하세요(모바일만 적용)
boolean isFrontCamera =
mTRTCCloud.getDeviceManager().isFrontCamera();
// 전면 또는 후면 카메라의 전환(모바일 전용)
// true 전달: 전면으로 전환; false 전달: 후면으로 전환
mTRTCCloud.getDeviceManager().switchCamera(!isFrontCamera);
```

iOS

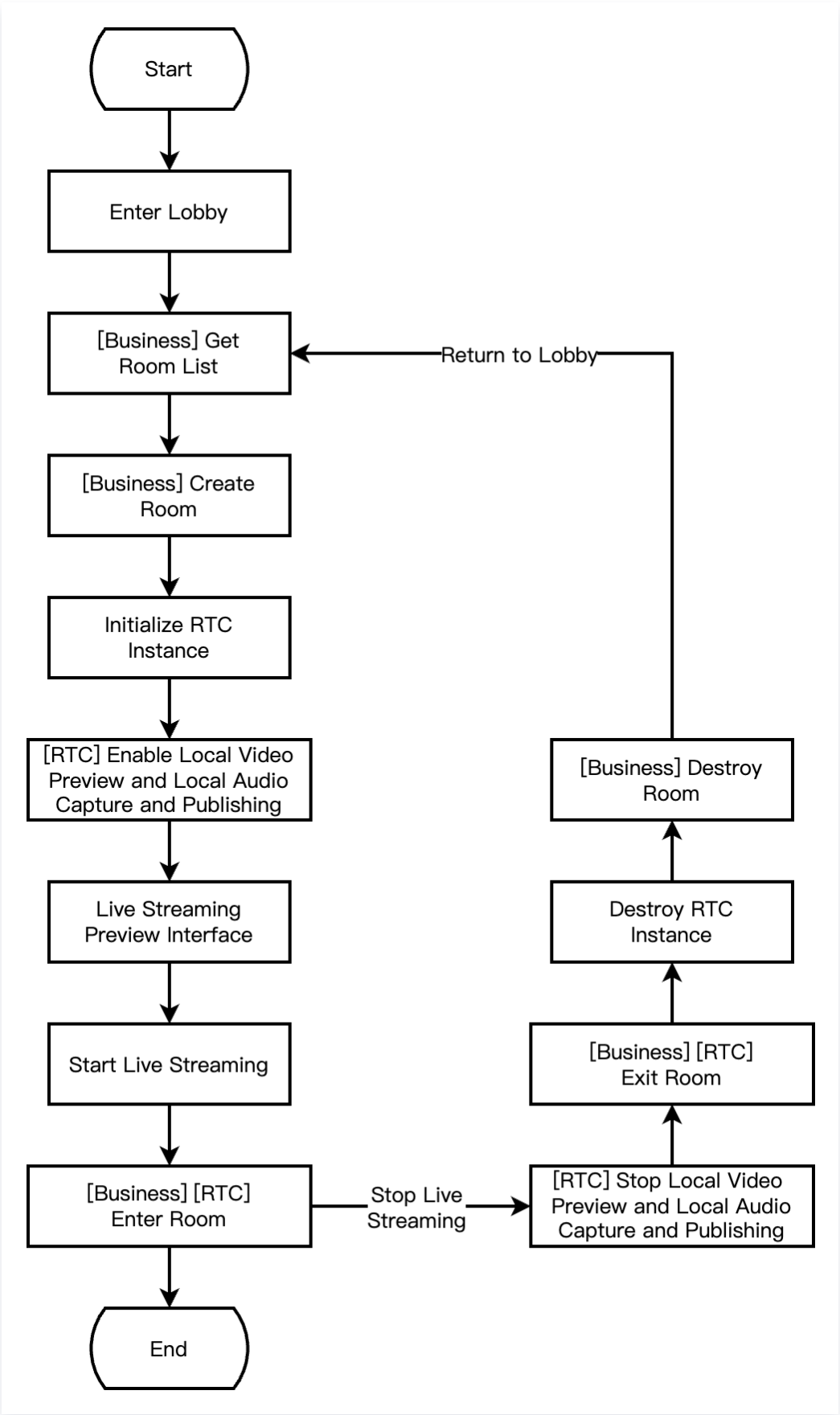
최종 업데이트 날짜:: 2025-10-28 11:32:37

업무 프로세스

본 섹션에서 쇼 라이브 방송에서 자주 사용되는 업무 프로세스를 종합하여 전체 시나리오의 구현 프로세스를 더욱 쉽게 이해할 수 있도록 도와줍니다.

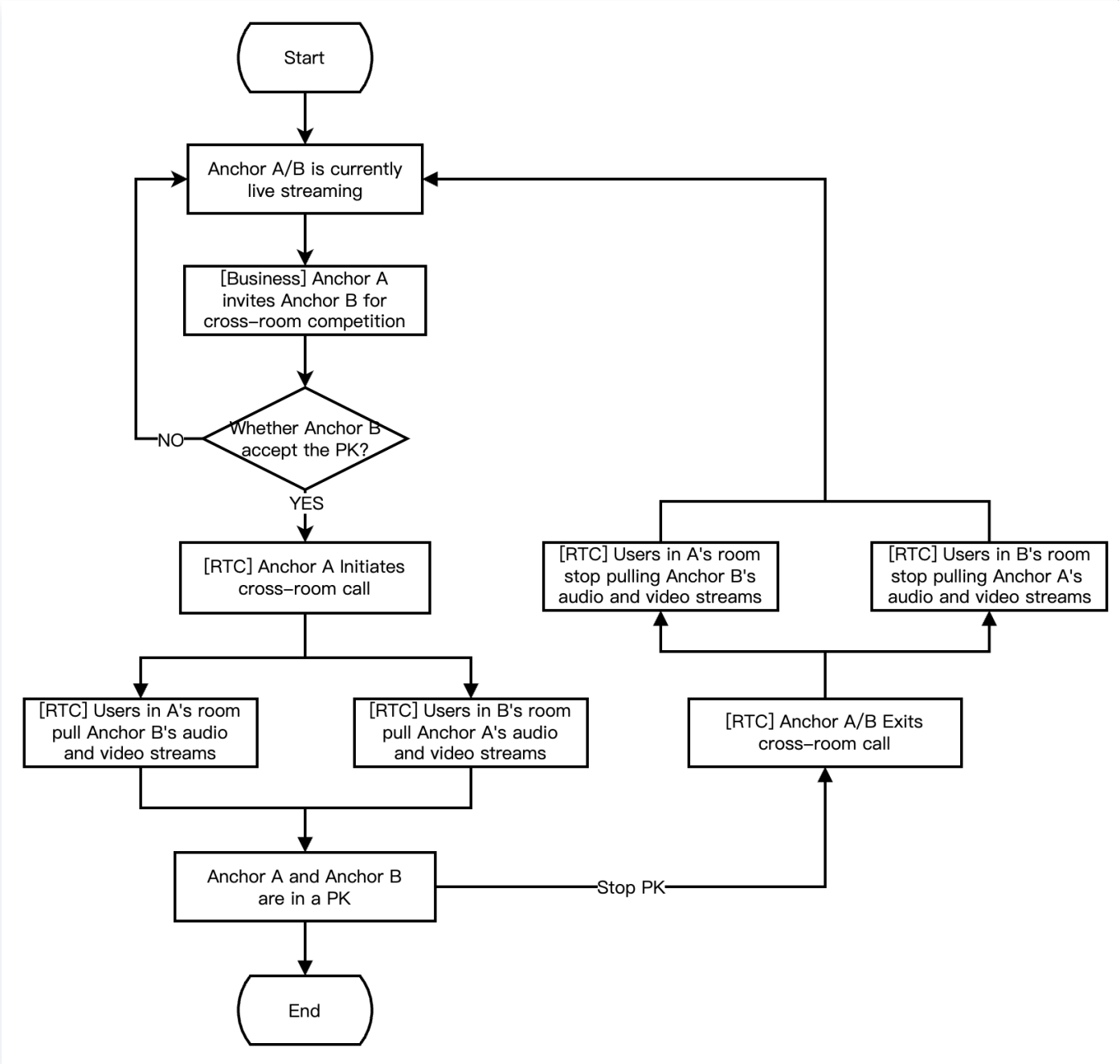
스트리머 방송의 시작 및 종료

아래 그림은 스트리머(방주인)의 로컬 미리보기, 방 생성, 방 입장 및 방송 시작, 방 퇴장 후 방송 종료의 프로세스를 보여줍니다.



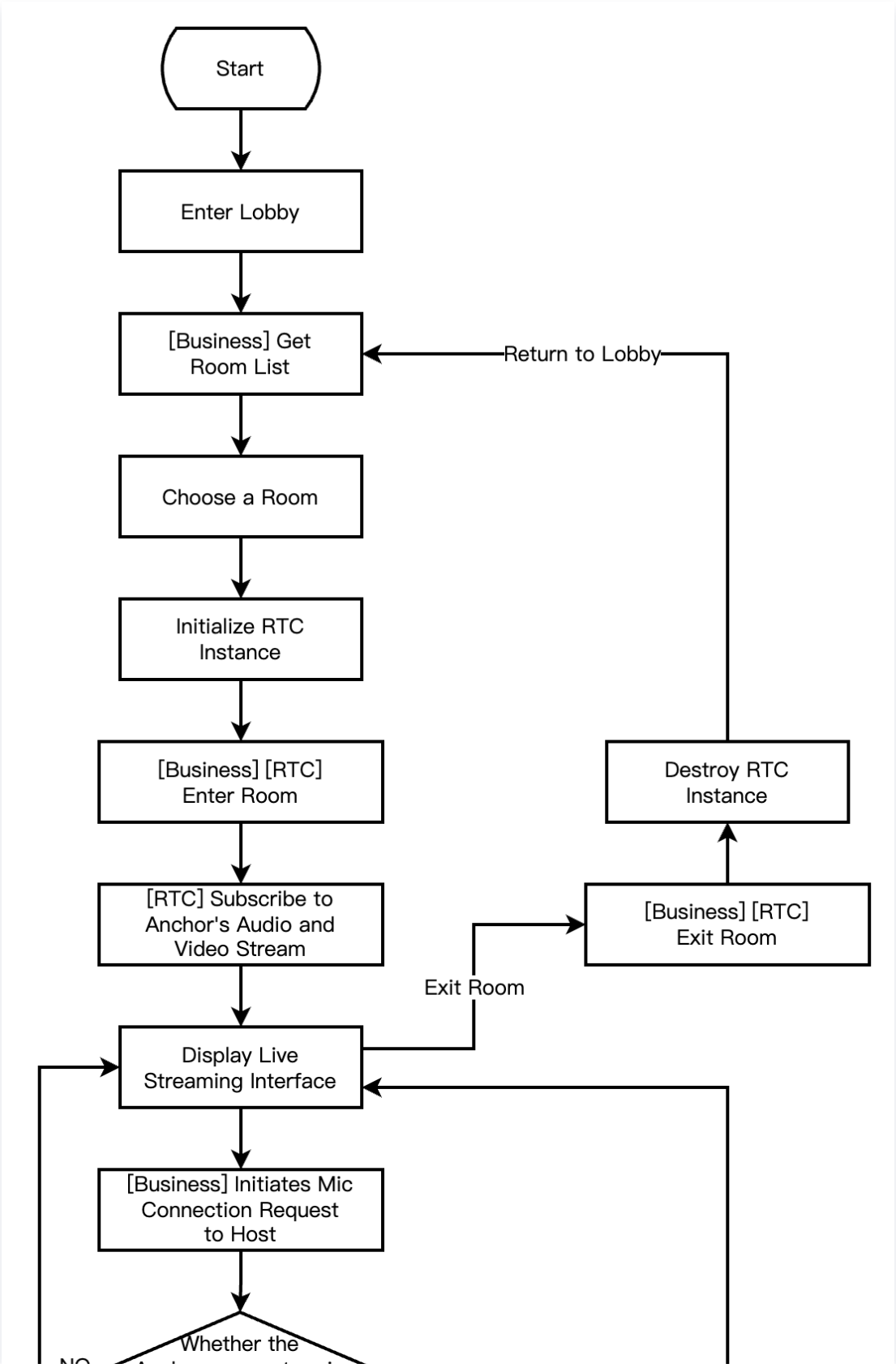
스트리머 간 크로스 룸 PK 연결

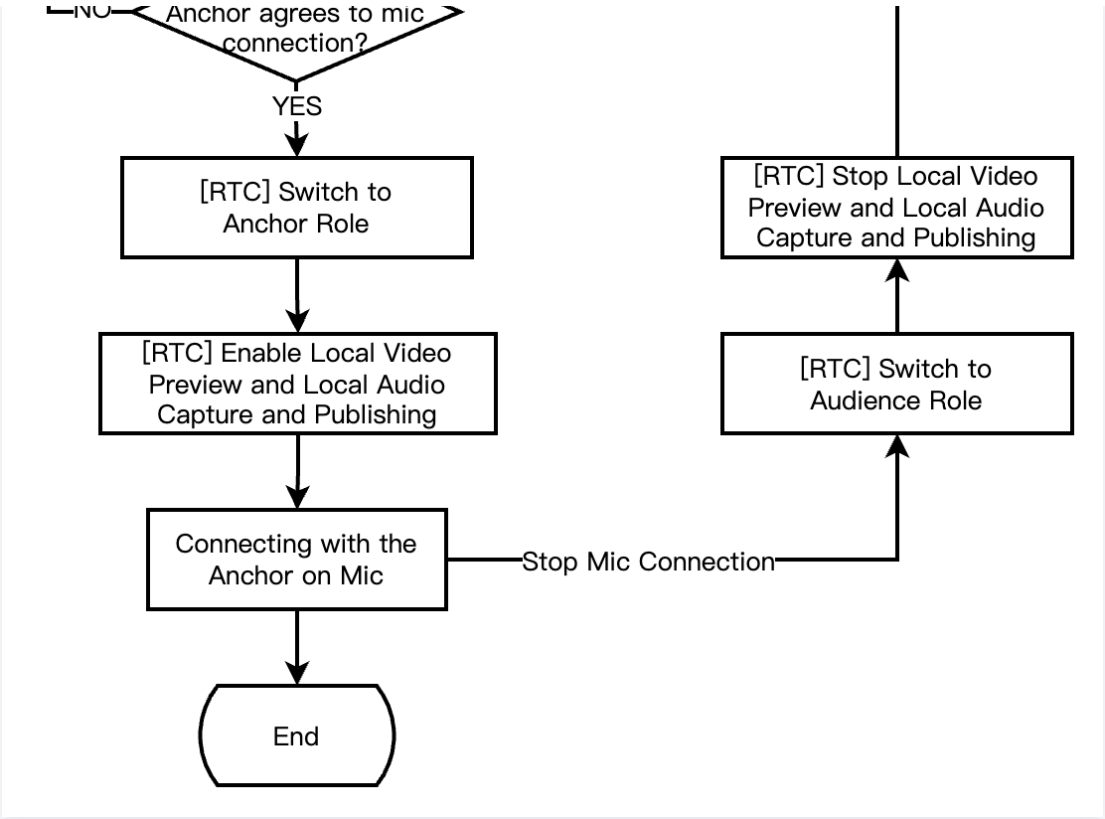
아래 그림으로 스트리머 A가 스트리머 B를 초청하여 크로스 룸 PK 연결을 진행하는 프로세스를 보여줍니다. 크로스 룸 PK 과정에서 두 방의 시청자 모두 두 방주인의 PK 연결 라이브 방송 화면을 볼 수 있습니다.



RTC 시청자 방 입장 후 마이크 연결합니다

아래 그림으로 RTC 실시간 인터랙션 라이브 방송실에서 시청자의 방 입장, 마이크 연결 신청, 연결 종료, 방 퇴장 프로세스를 보여줍니다.





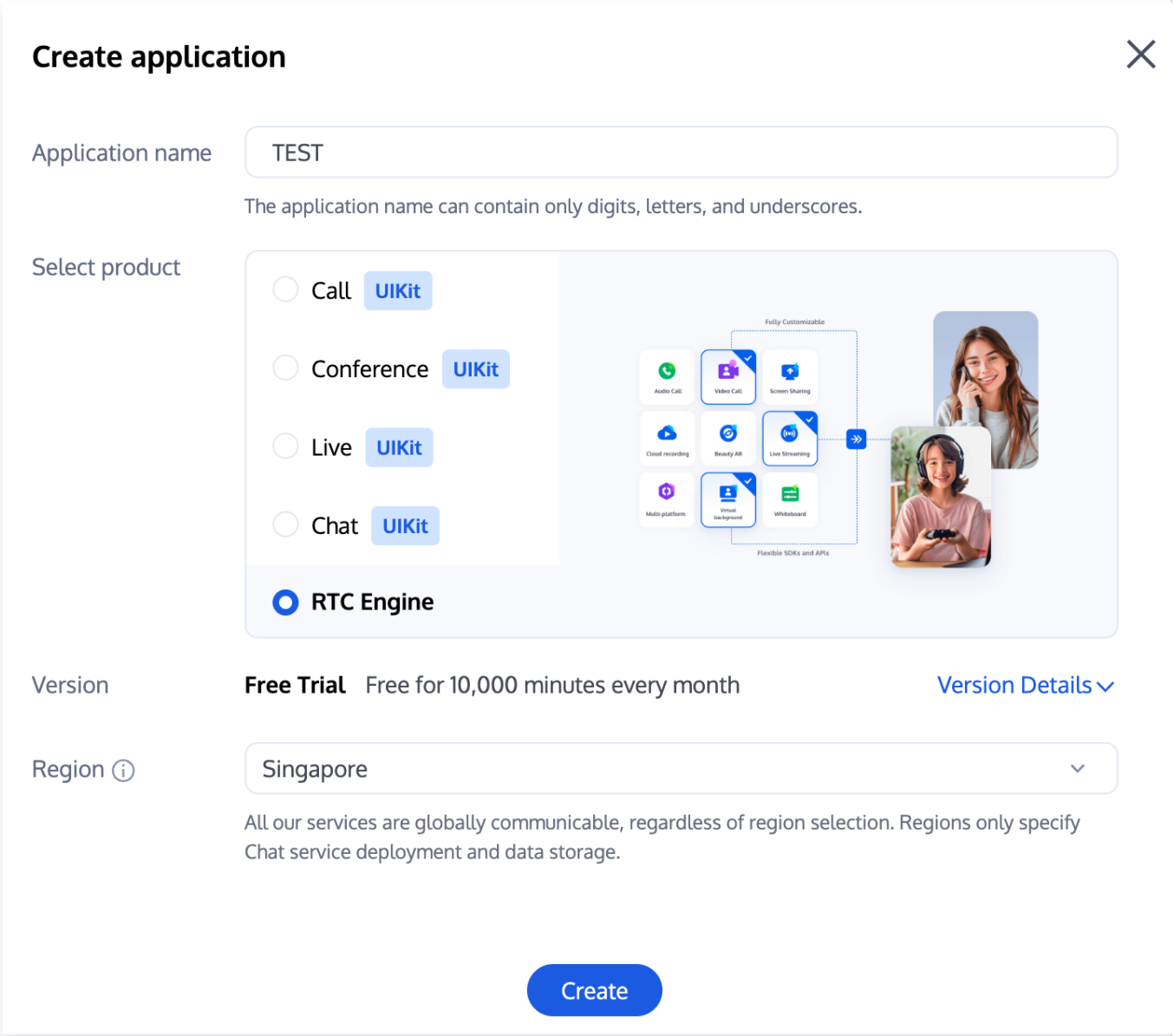
접수 준비

단계1: 서비스 개통

쇼 라이브 방송 시나리오는 일반적으로 RTC Engine와 뷰티 AR 두 가지 유료 PaaS 서비스에 의존하여 구축됩니다. 여기서 RTC Engine은 실시간 음성 및 영상 인터랙티브 기능을 제공하고, 뷰티 AR은 뷰티 효과 기능을 제공합니다. 타사 뷰티 제품을 사용하는 경우 뷰티 AR 통합 부분을 무시할 수 있습니다.

RTC Engine 서비스의 개통

1. 먼저, RTC Engine 콘솔에 로그인하여 애플리케이션을 생성해야 합니다. 필요에 따라 RTC Engine 애플리케이션 버전을 업그레이드할 수 있으며, 예를 들어 프로페셔널 버전은 더 많은 부가 기능 서비스를 사용할 수 있습니다.

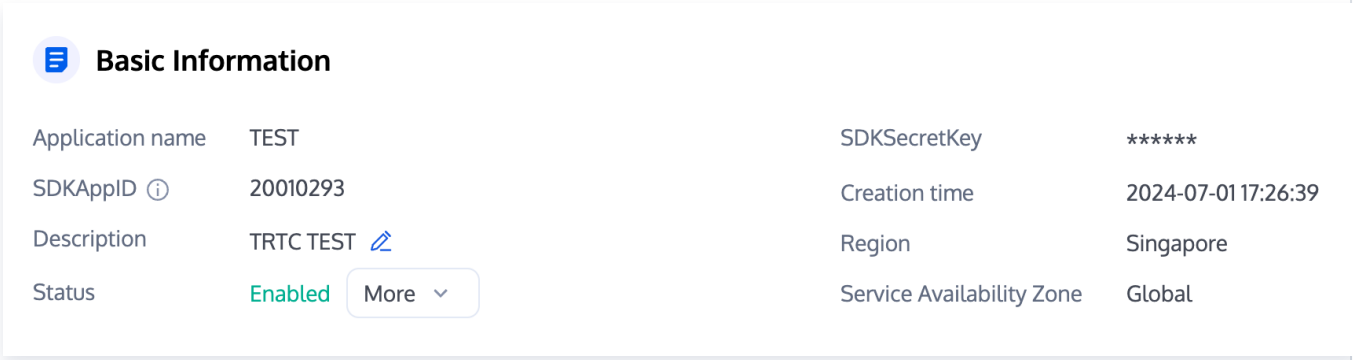


설명:

- 두 개의 애플리케이션을 생성하고 각각 테스트 환경과 프로덕션 환경에 적용하는 것을 권장하며, 1년 동안 각 텐센트 클라우드 계정(UIN)에 매월 10,000분의 무료 사용 시간이 제공됩니다.
- RTC Engine 월정액 요금제는 체험판(기본), 라이트, 스탠다드, 프로페셔널로 구분되며, 각각 다른 부가 기능 서비스를 사용할 수 있습니다. 자세한 내용은 버전 기능 및 월정액 요금제 설명 을 참조하세요.

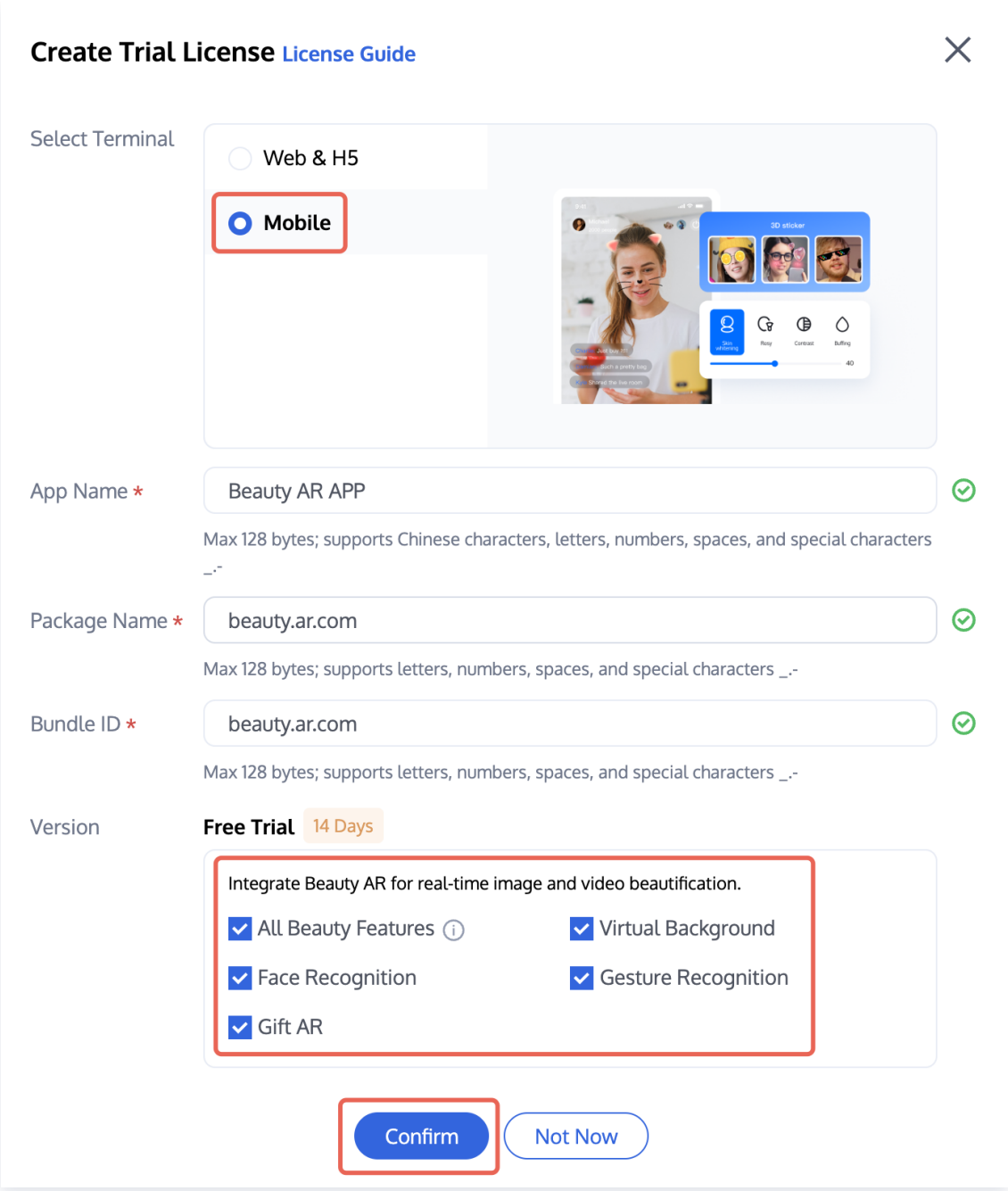
2. 애플리케이션 생성이 완료되면, 앱 관리 – 앱 개요 섹션에서 해당 애플리케이션의 기본 정보를 확인할 수 있습니다. 향후 사용을 위해 **SDKAppID**와 **SDKSecretKey**를 안전하게 보관해야 하며, 키 유출로 인한 트

래픽 도용을 방지해야 합니다.

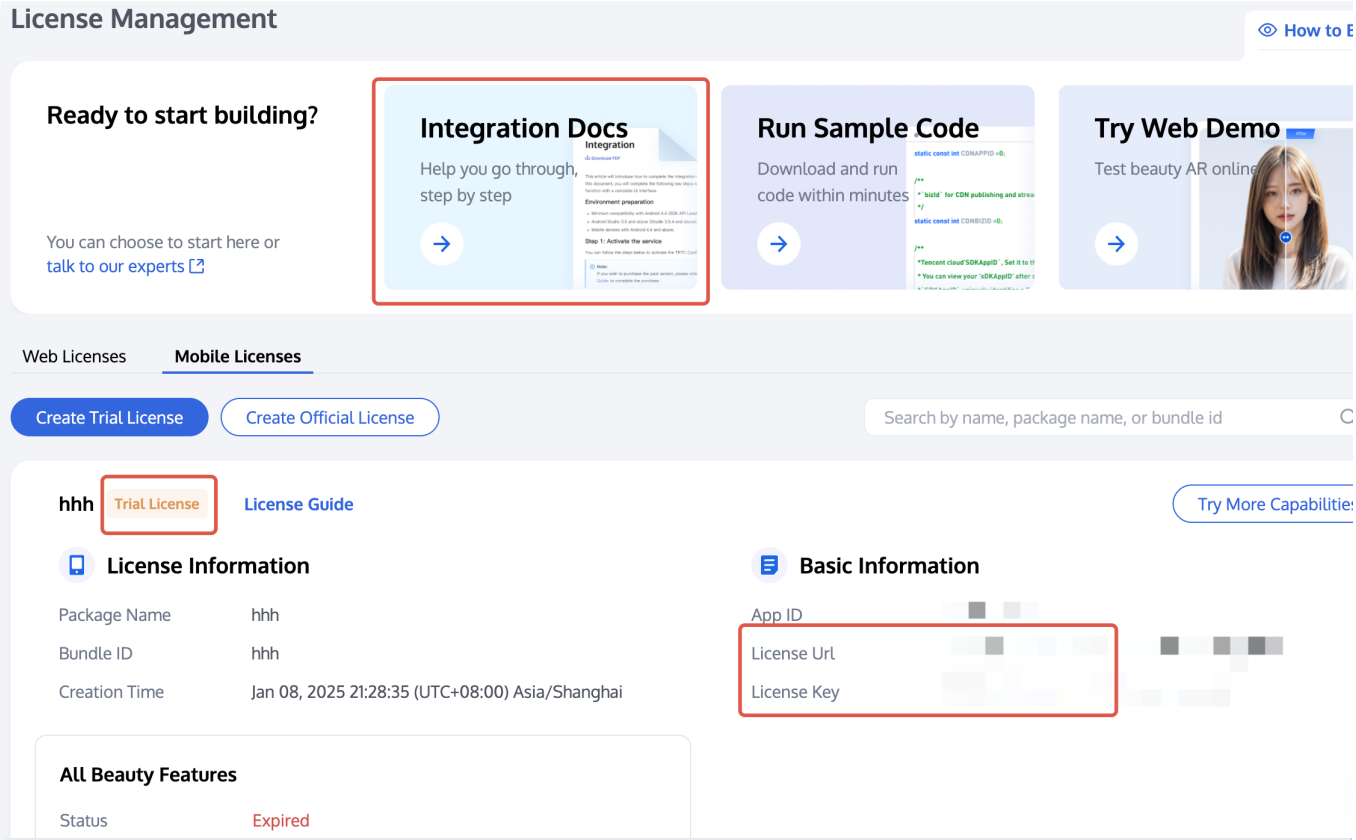


뷰티 AR 서비스의 개통

1. 뷰티 AR 콘솔 > 모바일 라이선스에 로그인하여 **테스트 라이선스 생성**을 클릭하세요(테스트 버전 라이선스는 무료로 14일간 유효하며, 1회 연장 가능하여 총 28일 사용 가능). 모바일을 선택하고 실제 필요에 따라 앱 이름, 패키지 이름 및 번들 ID를 입력하세요. 사용해 보고 싶은 기능을 선택하세요. 예를 들어 **모든 뷰티 기능**, **가상 배경**, **얼굴 인식**, **제스처 인식**, **선물 애니메이션 효과** 등, 그런 다음 **확인**을 클릭하세요.



2. 활성화 후, 현재 페이지에서 손님의 정보를 확인하고 상단의 통합 가이드를 참조하여 통합할 수 있습니다. 통합 가이드에서 라이선스 Key와 라이선스 URL 사용 방법을 확인할 수 있습니다.



단계2: SDK 임포트하기

RTC Engine SDK와 뷰티 AR SDK는 **CocoaPods** 라이브러리에 릴리스되었으며 CocoaPods를 통해 통합할 수 있습니다.

1. CocoaPods를 설치합니다. 터미널 창구에 다음 명령을 입력하세요(Mac에 Ruby 환경을 미리 설치해야 함).

```
sudo gem install cocoapods
```

2. Podfile 파일을 생성합니다. 프로젝트 경로로 이동한 후 다음 명령줄을 입력하면 프로젝트 경로에 Podfile 파일이 생성됩니다.

```
pod init
```

3. Podfile 파일을 편집합니다. 프로젝트 요구 사항에 따라 적절한 버전을 선택하고 Podfile 파일을 편집하세요.

```
platform :ios, '8.0'
    target 'App' do
```

```
    # RTC Engine 라이트 버전
    # 설치 패키지 크기 증가가 최소화되었지만 RTC Engine와 라이브 방송 플레이어
(TXLivePlayer) 두 가지 기능만 지원됩니다.
    pod 'TXLiteAVSDK_TRTC', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_TRTC.pods
pec'

    # Professional 프로페셔널 버전
    # Real-Time Communication Engine (RTC Engine), 라이브방송 플레이어
(TXLivePlayer), RTMP 푸시 스트리밍(TXLivePusher), VOD 플레이어
(TXVodPlayer), 짧은 비디오 레코딩 및 편집(UGSV) 등 다양한 기능을 포함합니다.
    # pod 'TXLiteAVSDK_Professional', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_Professio
nal.podspec'

    # 뷰티 AR SDK 예: S1-07 패키지
    pod 'TencentEffect_S1-07'

end
```

4. SDK를 업데이트하고 설치합니다.

   터미널 창구에 다음 명령을 입력하여 로컬 라이브러리 파일을 업데이트하고 SDK를 설치하세요.

   ```
   pod install
   ```

   또는 다음 명령을 사용하여 로컬 라이브러리 버전을 업데이트합니다.

   ```
   pod update
   ```

   pod 명령 실행 후 SDK가 통합된 .xcworkspace 확장자의 프로젝트 파일이 생성되며, 더블 클릭하여 열 수 있습니다.

   설명:

   - pod 검색이 실패할 경우, pod의 로컬 repo 캐시를 업데이트해 보는 것이 좋습니다. 업데이트 명령은 다음과 같습니다.

     ```
     pod setup
     ```

```
pod repo update
rm ~/Library/Caches/CocoaPods/search_index.json
```

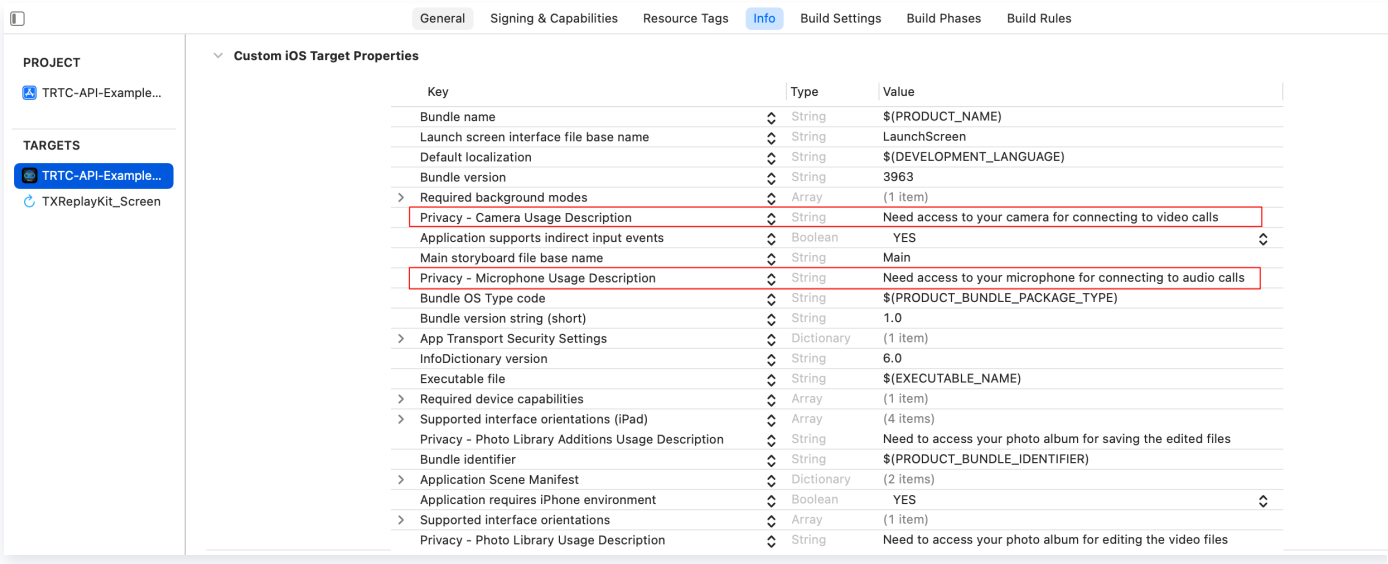
- 추천하는 자동 로드 방식 외에도, SDK를 다운로드하여 수동으로 임포트할 수도 있으며, 자세한 내용은 RTC Engine SDK 수동 통합 및 뷰티 AR SDK 수동 통합를 참조하세요.

5. 뷰티 리소스를 실제 프로젝트에 추가합니다.
   5.1 해당 패키지의 SDK 및 뷰티 리소스를 다운로드하고 압축을 해제한 후, resources/motionRes 폴더의 bundle 리소스를 실제 프로젝트에 추가합니다.
   5.2 Build Settings의 Other Linker Flags에 `-ObjC` 을 추가합니다.
6. Bundle Identifier를 신청한 테스트 권한과 일치하도록 수정합니다.

단계 3: 엔지니어링 구성

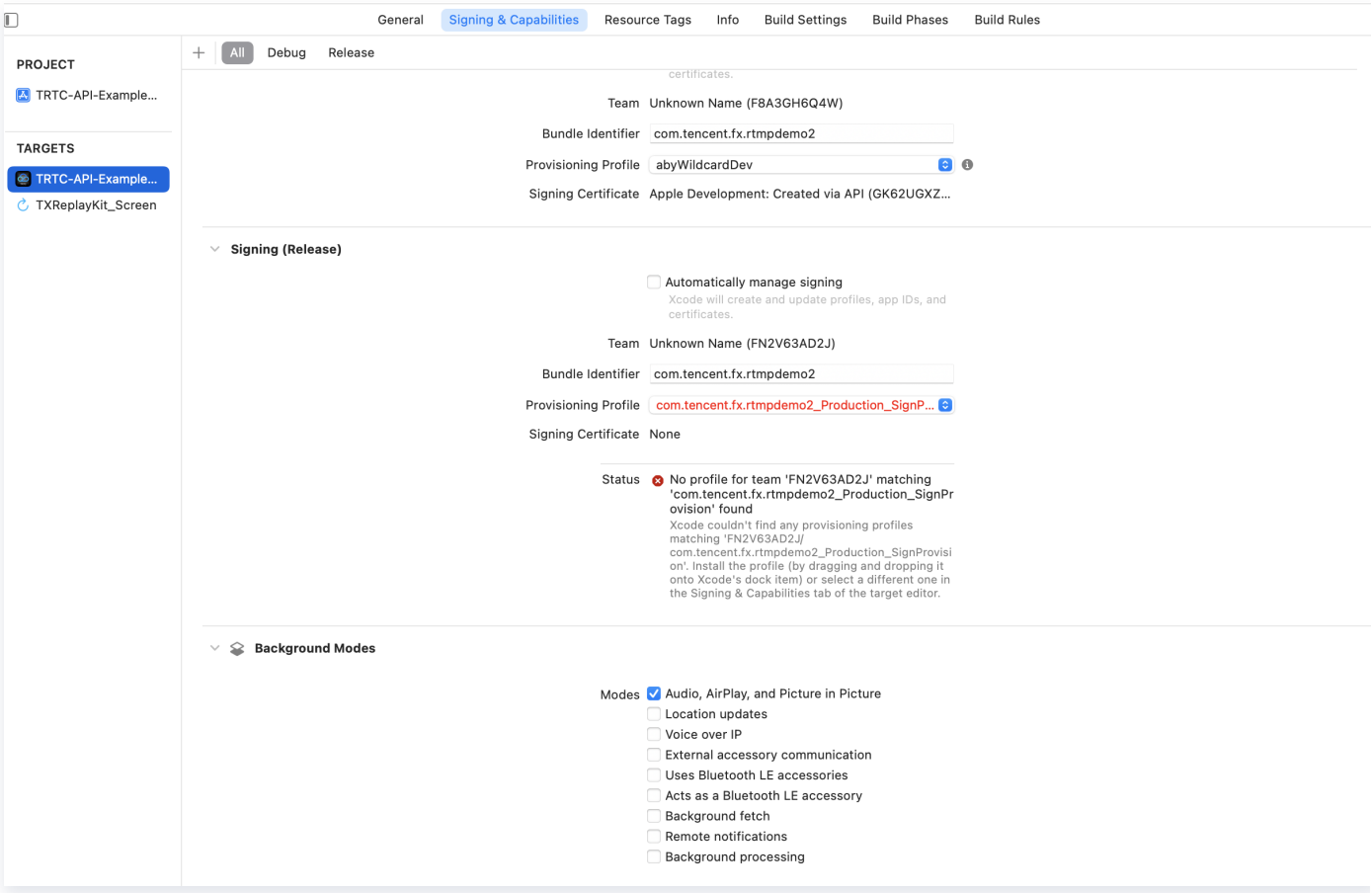
1. 권한의 설정.

   쇼 라이브 방송 시나리오에서 RTC Engine SDK 및 뷰티 AR SDK는 다음 권한이 필요합니다. App의 Info.plist에 다음 두 항목을 추가하세요. 각각 마이크와 카메라에 대한 시스템 권한 다이얼로그 박스의 안내 메시지에 해당합니다.
   - **Privacy – Microphone Usage Description**, 마이크 사용 목적에 대한 안내문을 입력하세요.
   - **Privacy – Camera Usage Description**, 카메라 사용 목적 안내문을 입력하세요.



2. 앱이 백그라운드에서도 관련 기능을 계속 실행하려면 XCode에서 현재 프로젝트를 선택하고 Capabilities에 있

는 Background Modes 설정을 ON으로 설정한 후 **Audio, AirPlay and Picture in Picture**을 클릭하세요. 아래 그림과 같습니다.



단계4: 인증 및 허가

RTC Engine 인증 자격 증명

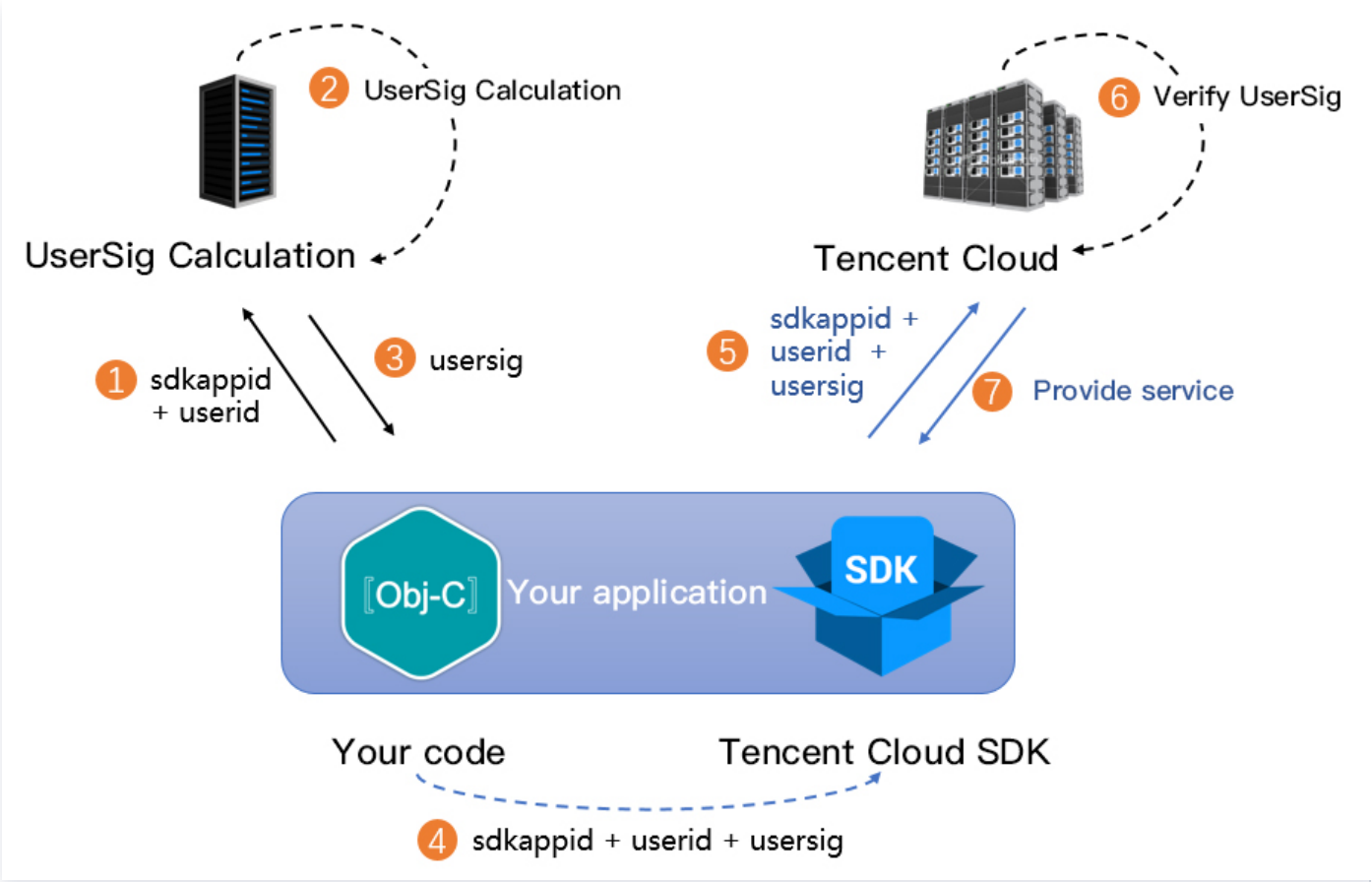
UserSig는 텐센트 클라우드가 설계한 보안 서명이고 악의적인 공격자가 손님의 클라우드 서비스 사용 권한을 도용하는 것을 방지하기 위한 것입니다. RTC Engine은 입장 시 이 인증 자격 증명을 검증합니다.

- 디버깅 및 테스트 단계: 클라이언트 예시 코드와 콘솔에서 가져오기 두 가지 방법으로 UserSig를 계산 및 생성할 수 있으며, 디버깅 및 테스트 용도로만 사용됩니다.
- 정식 운영 단계: 클라이언트가 역공학으로 키가 유출되는 것을 방지하기 위해 보안 등급이 더 높은 서버 UserSig 계산 방식의 사용을 권장합니다.

구체적인 구현 프로세스는 다음과 같습니다.

1. 손님의 App은 SDK 초기화 함수를 호출하기 전에 먼저 서버에 UserSig를 요청해야 합니다.
2. 손님의 서버는 SDKAppID와 UserID에 따라 UserSig를 계산합니다.

3. 서버는 계산된 UserSig를 손님의 App에 반환합니다.
4. 손님의 App은 획득한 UserSig를 특정 API를 통해 SDK에 전달합니다.
5. SDK는 SDKAppID + UserID + UserSig를 텐센트 클라우드 서버에 제출하여 검증합니다.
6. 텐센트 클라우드는 UserSig를 검증하여 합법성을 확인합니다.
7. 검증이 완료되면 RTC Engine SDK에 실시간 음성 및 영상 서비스를 제공합니다.



⚠ 주의:

- 디버깅 단계의 로컬 UserSig 계산 방식은 온라인 환경에 적용하는 것을 권장하지 않으며, 역공학으로 인해 키가 유출될 수 있기 때문입니다.
- 여러 언어 버전(Java/Go/PHP/Node.js/Python/C#/C++)의 UserSig 서버 계산 소스 코드를 제공하며, 자세한 내용은 서버에서 UserSig 계산을 참조하세요.

뷰티 AR 인증 허가

뷰티 AR 사용 전에 텐센트 클라우드에 라이선스 증명을 검증해야 합니다. 라이선스 설정에는 라이선스 Key와 라이선스 URL이 필요하며 예시 코드는 다음과 같습니다.

```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey
completion:^(NSInteger authresult, NSString * _Nonnull errorMsg) {
 if (authresult == TELicenseCheckOk) {
     NSLog(@"인증 성공");
 } else {
     NSLog(@"인증 실패");
 }
}];
```

⚠ 주의:

- 관련 업무 모듈의 초기화 코드에서 인증 허가를 트리거하고 사용하기 전에 임시로 라이선스를 다운로드하는 것을 피하며, 인증 시에는 네트워크 권한이 있어야 합니다.
- 실제 애플리케이션의 Bundle ID는 라이선스 생성 시 바인딩된 Bundle ID와 완전히 일치해야 하며, 그렇지 않으면 라이선스 검증이 실패합니다. 자세한 내용은 인증 오류 코드을 참조하세요.

단계5: SDK 초기화

RTC Engine SDK의 초기화

```
// RTC Engine SDK 인스턴스의 생성(싱글톤 모드)
self.trtcCloud = [TRTCCloud sharedInstance];
// 이벤트 리스너의 설정
self.trtcCloud.delegate = self;

// SDK의 다양한 이벤트 알림(예: 오류 코드, 경고 코드, 오디오/비디오 상태 매개변수 등)
- (void)onError:(TXLiteAVError)errCode errMsg:(nullable NSString *)errMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", errCode, errMsg);
}
```

```
- (void)onWarning:(TXLiteAVWarning)warningCode warningMsg:(nullable
NSString *)warningMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", warningCode, warningMsg);
}

// 이벤트 리스너의 제거
self.trtcCloud.delegate = nil;
// RTC Engine SDK 인스턴스(싱글톤 모드)를 파기합니다.
[TRTCCloud destroySharedIntance];
```

설명:
SDK 이벤트 알림을 모니터링하고 일반적인 오류에 대한 로그 출력 및 처리를 권장합니다. 자세한 내용은 오류 코드 표을 참조하세요.

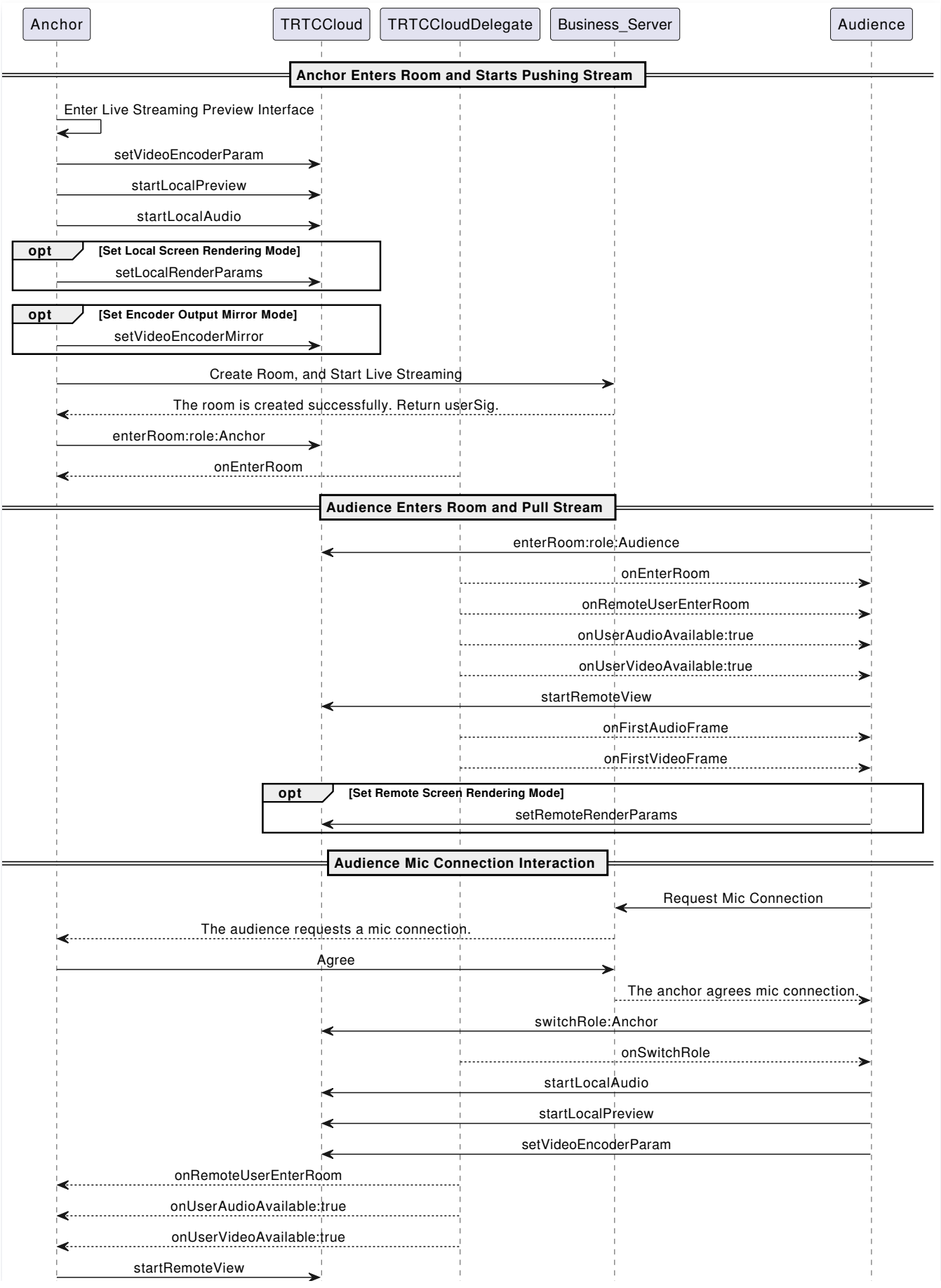
뷰티 AR SDK의 초기화

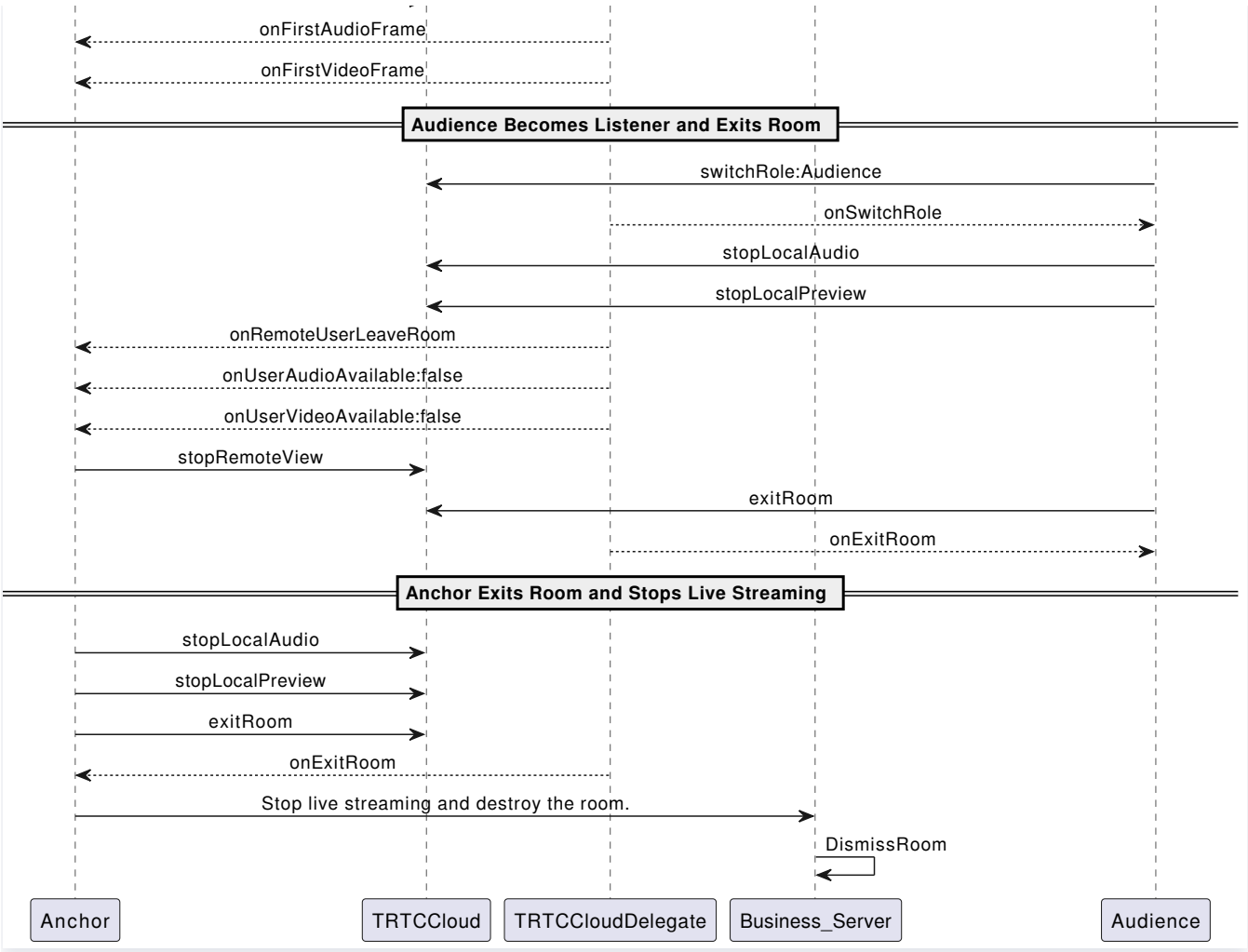
```
// 뷰티 관련 리소스의 로드
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
 @"root_path":[[NSBundle mainBundle] bundlePath]
};

// 텐센트 이펙트 SDK의 초기화
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize
assetsDict:assetsDict];

// 텐센트 이펙트 SDK의 릴리스
[self.beautyKit deinit]
```

설명:
뷰티 AR SDK 초기화 전에 리소스 복사 등 준비 작업이 필요하며 자세한 단계는 뷰티 AR SDK 통합 단계을 참조하세요.

접속 과정

API 시퀀스 다이어그램





단계1: 스트리머가 방에 입장하여 스트리밍을 시작합니다

1. 스트리머가 방에 들어가기 전에 로컬 비디오 미리보기 및 오디오 수집을 시작합니다.

```
// 스트리머의 로컬 화면 미리보기를 표시하는 데 사용되는 비디오 렌더링 컨트롤 가져
옵니다
@property (nonatomic, strong) UIView *anchorPreviewView;
@property (nonatomic, strong) TRTCCloud *trtcCloud;

- (void)setupTRTC {
    self.trtcCloud = [TRTCCloud sharedInstance];
    self.trtcCloud.delegate = self;
    // 원격 사용자가 보는 화면 품질을 결정하는 비디오 인코딩 매개변수를 설정합니
다.
    TRTCVideoEncParam *encParam = [[TRTCVideoEncParam alloc] init];
    encParam.videoResolution = TRTCVideoResolution_960_540;
    encParam.videoFps = 15;
    encParam.videoBitrate = 1300;
```

```
    encParam.resMode = TRTCVideoResolutionModePortrait;
    [self.trtcCloud setVideoEncoderParam:encParam];

    // isFrontCamera는 전면/후면 카메라를 사용하여 비디오를 캡처하도록 지정할
수 있습니다
    [self.trtcCloud startLocalPreview:self.isFrontCamera
view:self.anchorPreviewView];

    // 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
    [self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];
}
```

⚠ 주의:

- 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam을 직접 설정할 수 있으며, 각 등급별 최적의 해상도와 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.
- `enterRoom` 전에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집만 시작하며, `enterRoom` 을 호출한 후에야 스트리밍을 시작합니다.
- `enterRoom` 후에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집을 시작하고 자동으로 스트리밍을 시작합니다.

2. 스트리머가 로컬 화면의 렌더링 매개변수 및 인코더 출력 화면 모드(선택 사항)를 설정합니다.

```
- (void)setupRenderParams {
    TRTCRenderParams *params = [[TRTCRenderParams alloc] init];
    // 화면 미러 모드
    params.mirrorType = TRTCVideoMirrorTypeAuto;
    // 화면 채우기 모드
    params.fillMode = TRTCVideoFillMode_Fill;
    // 화면 회전 각도
    params.rotation = TRTCVideoRotation_0;
    // 로컬 화면의 렌더링 매개변수를 설정합니다
    [self.trtcCloud setLocalRenderParams:params];

    // 인코더 출력 화면 미러 모드를 설정합니다
    [self.trtcCloud setVideoEncoderMirror:YES];
    // 비디오 인코더 출력 화면의 방향을 설정합니다
    [self.trtcCloud setVideoEncoderRotation:TRTCVideoRotation_0];
```

```
}
```

> ⚠ 주의:
> - 로컬 화면 렌더링 파라미터의 설정은 로컬 화면의 렌더링 효과에만 영향을 줍니다.
> - 인코더 출력 모드의 설정은 방의 다른 사용자들이 보는 (및 클라우드 레코딩 파일) 화면 효과에 영향을 줍니다.

3. 스트리머가 정식으로 라이브 방송을 시작하고 방에 입장하여 스트리밍 시작합니다.

```
- (void)enterRoomByAnchorWithUserId:(NSString *)userId roomId:
(NSString *)roomId {
    TRTCParams *params = [[TRTCParams alloc] init];
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = @"userSig";
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = 0;
    // 스트리머 역할의 지정
    params.role = TRTCRoleAnchor;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}

// 방 입장 결과 이벤트의 콜백
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result는 방에 입장하는 데 소요된 시간(밀리초)을 나타냅니다.
        NSLog(@"Enter room succeed!");
    } else {
        // result는 방 입장 실패의 오류 코드를 나타냅니다.
        NSLog(@"Enter room failed!");
    }
}
```

> ⚠ 주의:
> - RTC Engine 방 번호는 정수형 `roomId` 와 문자열 형식 `strRoomId` 로 나뉘며, 두 유형의 방

은 서로 통하지 않으므로 방 번호 유형을 통일하는 것이 좋습니다.
- RTC Engine 사용자 역할은 스트리머와 청취자로 구분되며, 스트리머만 스트리밍 권한을 가집니다. 방 입장 시 사용자 역할을 지정해야 하며, 지정하지 않으면 기본적으로 스트리머 역할로 설정됩니다.
- 쇼 라이브방송 시나리오에서는 방 입장 모드를 `TRTCAppSceneLIVE` 으로 선택하는 것이 좋습니다.

단계2: 시청자가 방에 들어가서 스트리밍을 시작합니다

1. 시청자가 RTC Engine 방에 입장합니다.

```
- (void)enterRoomByAudienceWithUserId:(NSString *)userId roomId:
(NSString *)roomId {
    TRTCParams *params = [[TRTCParams alloc] init];
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = @"userSig";
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = 0;
    // 시청자 역할의 지정
    params.role = TRTCRoleAudience;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}

// 방 입장 결과 이벤트의 콜백
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result는 방에 임장하는 데 소요된 시간(밀리초)을 나타냅니다.
        NSLog(@"Enter room succeed!");
    } else {
        // result는 방 입장 실패의 오류 코드를 나타냅니다.
        NSLog(@"Enter room failed!");
    }
}
```

2. 시청자가 스트리머의 오디오 및 비디오 스트림을 구독합니다.

```
- (void)onUserAudioAvailable:(NSString *)userId available:
(BOOL)available {
    // 원격 사용자가 자신의 오디오를 게시/취소합니다
    // 자동 구독 모드에서는 사용자가 아무런 작업을 하지 않아도 SDK가 원격 사용자
의 오디오를 자동으로 재생합니다.
}

- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    // 원격 사용자가 메인 비디오 화면을 게시/취소합니다
    if (available) {
        // 원격 사용자의 비디오 스트림을 구독하고 비디오 렌더링 컨트롤을 바인딩
합니다
        [self.trtcCloud startRemoteView:userId
streamType:TRTCVideoStreamTypeBig view:self.remoteView];
    } else {
        // 원격 사용자의 비디오 스트림 구독을 중지하고 렌더링 컨트롤을 릴리스합
니다
        [self.trtcCloud stopRemoteView:userId
streamType:TRTCVideoStreamTypeBig];
    }
}
```

3. 시청자가 원격 화면의 렌더링 모드를 설정합니다(선택 사항).

```
- (void)setupRemoteRenderParams {
    TRTCRenderParams *params = [[TRTCRenderParams alloc] init];
    // 화면 미러 모드
    params.mirrorType = TRTCVideoMirrorTypeAuto;
    // 화면 채우기 모드
    params.fillMode = TRTCVideoFillMode_Fill;
    // 화면 회전 각도
    params.rotation = TRTCVideoRotation_0;
    // 원격 화면의 렌더링 모드를 설정합니다
    [self.trtcCloud setRemoteRenderParams:@"userId"
streamType:TRTCVideoStreamTypeBig params:params];
}
```

단계3: 시청자가 마이크 켜고 인터랙션 합니다

1. 시청자가 스트리머 역할로 전환됩니다.

```
- (void)switchToAnchor {
    // 스트리머 역할로 전환됩니다
    [self.trtcCloud switchRole:TRTCRoleAnchor];
}

// 역할 전환 이벤트의 콜백
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(NSString *)errMsg
{
    if (errCode == ERR_NULL) {
        // 역할 전환 성공
    }
}
```

2. 시청자가 로컬 오디오/비디오 수집 및 스트리밍을 시작합니다.

```
- (void)setupTRTC {
    // 원격 사용자가 보는 화면 품질을 결정하는 비디오 인코딩 매개변수를 설정합니
다.
    TRTCVideoEncParam *encParam = [[TRTCVideoEncParam alloc] init];
    encParam.videoResolution = TRTCVideoResolution_480_270;
    encParam.videoFps = 15;
    encParam.videoBitrate = 550;
    encParam.resMode = TRTCVideoResolutionModePortrait;
    [self.trtcCloud setVideoEncoderParam:encParam];

    // isFrontCamera는 전면/후면 카메라를 사용하여 비디오를 캡처하도록 지정할
수 있습니다
    [self.trtcCloud startLocalPreview:self.isFrontCamera
view:self.audiencePreviewView];
    // 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
    [self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];
}
```

⚠ 주의:

> 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam을 직접 설정할 수 있으며, 각 등급별 최적의 해상도와 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.

3. 시청자가 마이크를 끄고 스트리밍을 중지합니다.

```
- (void)switchToAudience {
    // 시청자 역할로 전환됩니다
    [self.trtcCloud switchRole:TRTCRoleAudience];
}

// 역할 전환 이벤트의 콜백
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(NSString *)errMsg
{
    if (errCode == ERR_NULL) {
        // 카메라 수집 및 스트리밍 중지합니다
        [self.trtcCloud stopLocalPreview];
        // 마이크 수집 및 스트리밍 중지합니다
        [self.trtcCloud stopLocalAudio];
    }
}
```

단계4: 방 나가기 및 해산하기

1. 방에서 나가기.

```
- (void)exitRoom {
    [self.trtcCloud stopLocalAudio];
    [self.trtcCloud stopLocalPreview];
    [self.trtcCloud exitRoom];
}

// 방 나가기 이벤트의 콜백
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"exitRoom을 호출하여 방에서 나가기");
    } else if (reason == 1) {
        NSLog(@"서버에 의해 현재 방에서 나가게 됩니다");
    } else if (reason == 2) {
        NSLog(@"현재 방 전체가 해체됩니다");
```

```
    }
}
```

> ⚠ 주의:
> - SDK가 점유한 모든 리소스가 릴리스된 후, SDK는 `onExitRoom` 콜백을 통해 알려줍니다.
> - `enterRoom` 을 다시 호출하거나 다른 음성/영상 SDK로 전환하려면 `onExitRoom` 콜백이 발생한 후 관련 작업을 수행하십시오. 그렇지 않으면 카메라, 마이크 장치에 강제로 점유됨 등 다양한 이상 현상이 발생할 수 있습니다.

2. 방 해산
   - **서버 측 해산:** RTC Engine은 서버 측 방 해산 API `DismissRoom` (숫자 방 ID와 문자열 방 ID 구분)을 제공합니다. 이 인터페이스를 호출하여 방의 모든 사용자를 방에서 나가게 시키하고 방을 해산할 수 있습니다.
   - **클라이언트 측 해산:** 각 클라이언트의 방에서 나가기 `exitRoom` 인터페이스를 통해 방 내 모든 스트리머와 청취자의 퇴장을 완료시키고, 퇴장 후 RTC Engine 방 라이프사이클 규칙에 따라 방이 자동으로 해산됩니다. 자세한 내용은 방에서 나가기 를 참조하세요.

> ⚠ 주의:
> 라이브 방송이 종료된 후 서버 측에서 방 해산 API를 호출하여 방이 해산되도록 하는 것이 좋습니다. 이렇게 하면 시청자가 의도치 않게 방에 들어가 예상치 못한 비용이 발생하는 것을 방지할 수 있습니다.

고급 기능

스트리머 간 크로스 룸 PK 연결

1. 어느 한쪽이 크로스 룸 PK 연결을 신청합니다.

```
- (void)connectOtherRoom:(NSString *)roomId {
    NSMutableDictionary *jsonDict = [[NSMutableDictionary alloc]
init];
    // 숫자 방 번호는 roomId입니다
    [jsonDict setObject:roomId forKey:@"strRoomId"];
    [jsonDict setObject:self.userId forKey:@"userId"];
    NSData *jsonData = [NSJSONSerialization
dataWithJSONObject:jsonDict options:NSJSONWritingPrettyPrinted
error:nil];
```

```
    NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
    [self.trtcCloud connectOtherRoom:jsonString];
}

// 크로스 룸 연결 결과의 콜백
- (void)onConnectOtherRoom:(NSString *)userId errCode:
(TXLiteAVError)errCode errMsg:(NSString *)errMsg {
    // 크로스 룸 연결을 하려는 다른 방의 스트리머의 사용자 ID
    // 에러 코드, ERR_NULL은 요청 성공을 나타냅니다
    // 에러 정보
}
```

> ⚠ 주의:
> 크로스 룸 PK 연결의 로컬 사용자와 상대 사용자는 모두 스트리머 역할이어야 하며, 오디오 또는 비디오 업스트림이 모두 있어야 합니다.

2. 두 방의 모든 사용자는 다른 방의 스크리머로부터 오디오/비디오 스트림 사용 가능 콜백을 받게 됩니다.

```
- (void)onUserAudioAvailable:(NSString *)userId available:
(BOOL)available {
    // 원격 사용자가 자신의 오디오를 게시/취소합니다
    // 자동 구독 모드에서는 사용자가 아무런 작업을 하지 않아도 SDK가 원격 사용자
의 오디오를 자동으로 재생합니다.
}

- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    // 원격 사용자가 메인 비디오 화면을 게시/취소합니다
    if (available) {
        // 원격 사용자의 비디오 스트림을 구독하고 비디오 렌더링 컨트롤에 바인딩
합니다
        [self.trtcCloud startRemoteView:userId
streamType:TRTCVideoStreamTypeBig view:self.remoteView];
    } else {
        // 원격 사용자의 비디오 스트림 구독을 중지하고 렌더링 컨트롤을 릴리스합
니다
        [self.trtcCloud stopRemoteView:userId
streamType:TRTCVideoStreamTypeBig];
```

```
    }
}
```

3. 어느 한쪽이 크로스 룸 PK 연결을 종료합니다.

```
// 크로스 룸 채팅 연결 종료
[self.trtcCloud disconnectOtherRoom];

// 크로스 룸 연결 종료 결과의 콜백
- (void)onDisconnectOtherRoom:(TXLiteAVError)errCode errMsg:(NSString
*)errMsg {
}
```

> ⚠ 주의:
> - `DisconnectOtherRoom()` 을 호출하면 모든 다른 방의 스트리머와의 크로스 룸 PK 연결이 종료됩니다.
> - 크로스 룸 PK 연결의 발신 측 또는 수신 측 중 어느 한쪽에서 `DisconnectOtherRoom()` 을 호출하여 크로스 룸 PK 연결을 종료할 수 있습니다.

제3자 뷰티 제품의 사용

RTC Engine은 서드파티 뷰티 효과 제품의 접근을 지원하며, 아래에서는 뷰티 AR을 예로 들어 서드파티 뷰티 접근 절차를 설명합니다.

1. 뷰티 AR SDK 통합 및 라이선스 권한의 신청에 대한 자세한 내용은 접속 준비 단계를 참조하세요.
2. SDK 소재 리소스 경로의 설정(해당하는 경우).

```
NSString *beautyConfigPath =
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) lastObject];
beautyConfigPath = [beautyConfigPath
stringByAppendingPathComponent:@"beauty_config.json"];
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath
isDirectory:&isDir] && !isDir) {
    NSString *beautyConfigJsonStr = [NSString
stringWithContentsOfFile:beautyConfigPath
```

```
encoding:NSUTF8StringEncoding error:nil];
    NSError *jsonError;
    NSData *objectData = [beautyConfigJsonStr
dataUsingEncoding:NSUTF8StringEncoding];
    beautyConfigJson = [NSJSONSerialization
JSONObjectWithData:objectData

options:NSJSONReadingMutableContainers
                                          error:&jsonError];
}
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle]
bundlePath],
                             @"tnn_"
                             @"beauty_config":beautyConfigJson
};
// SDK 초기화: width와 height는 각각 texture의 너비와 높이입니다.
self.xMagicKit = [[XMagic alloc]
initWithRenderSize:CGSizeMake(width,height) assetsDict:assetsDict];
```

3. 제3자 뷰티 제품의 비디오 데이터 콜백을 설정하여 뷰티 SDK가 처리한 각 프레임 데이터 결과를 RTC Engine SDK 내부로 전달하고 렌더링 처리합니다.

```
// RTC Engine SDK가 서드파티 뷰티의 비디오 데이터 콜백을 설정합니다
[self.trtcCloud setLocalVideoProcessDelegete:self
pixelFormat:TRTCVideoPixelFormat_Texture_2D
bufferType:TRTCVideoBufferType_Texture];

#pragma mark - TRTCVideoFrameDelegate

// YTProcessInput을 구성하여 SDK 내부로 전달하여 렌더링 처리를 합니다
- (uint32_t)onProcessVideoFrame:(TRTCVideoFrame *_Nonnull)srcFrame
dstFrame:(TRTCVideoFrame *_Nonnull)dstFrame {
    if (!self.xMagicKit) {
        [self buildBeautySDK:srcFrame.width and:srcFrame.height
texture:srcFrame.textureId];//XMagic SDK의 초기화
        self.heightF = srcFrame.height;
        self.widthF = srcFrame.width;
    }
```

```
    if(self.xMagicKit!=nil && (self.heightF!=srcFrame.height ||
self.widthF!=srcFrame.width)){
        self.heightF = srcFrame.height;
        self.widthF = srcFrame.width;
        [self.xMagicKit setRenderSize:CGSizeMake(srcFrame.width,
srcFrame.height)];
    }
    YTProcessInput *input = [[YTProcessInput alloc] init];
    input.textureData = [[YTTextureData alloc] init];
    input.textureData.texture = srcFrame.textureId;
    input.textureData.textureWidth = srcFrame.width;
    input.textureData.textureHeight = srcFrame.height;
    input.dataType = kYTTextureData;
    YTProcessOutput *output = [self.xMagicKit process:input
withOrigin:YtLightImageOriginTopLeft
withOrientation:YtLightCameraRotation0];
    dstFrame.textureId = output.textureData.texture;
    return 0;
}
```

> ⚠ 주의:
> 단계1 및 단계2는 제3자 뷰티 제품의 구현 방식에 따라 다르며, **단계3**은 RTC Engine이 제3자 뷰티를 통합하는 **공통적이고 중요한 단계**입니다.

듀얼 인코딩 모드

듀얼 인코딩 모드를 활성화하면 현재 사용자의 인코더가 [고화질 대형 화면]과 [저화질 소형 화면] 두 가지 비디오 스트림을 동시에 출력합니다(단, 오디오 스트림은 하나만 존재함). 이렇게 하면 방의 다른 사용자가 자신의 네트워크 상황이나 화면 크기에 따라 [고화질 대형 화면] 또는 [저화질 소형 화면]을 선택하여 구독할 수 있습니다.

1. 대형및 소형 화면 듀얼 인코딩 모드를 활성화합니다.

```
- (void)enableDualStreamMode:(BOOL)enable {
    // 소형 비디오 스트림의 인코딩 매개변수 (자체 정의 가능)
    TRTCVideoEncParam *smallVideoEncParam = [[TRTCVideoEncParam
alloc] init];
    smallVideoEncParam.videoResolution = TRTCVideoResolution_480_270;
    smallVideoEncParam.videoFps = 15;
    smallVideoEncParam.videoBitrate = 550;
    smallVideoEncParam.resMode = TRTCVideoResolutionModePortrait;
```

```
    [self.trtcCloud enableEncSmallVideoStream:enable
withQuality:smallVideoEncParam];
}
```

> ⚠ 주의:
> 듀얼 인코딩을 활성화하면 더 많은 CPU 및 네트워크 대역폭이 소모되므로 Mac, Windows 또는 고성능 Pad에서는 활성화를 고려할 수 있지만 모바일에서는 활성화하지 않는 것이 좋습니다.

2. 원격 사용자 비디오 스트림의 유형을 선택합니다.

```
// 원격 사용자 비디오 스트림을 구독할 때 선택 가능한 비디오 스트림의 유형
[self.trtcCloud startRemoteView:userId
streamType:TRTCVideoStreamTypeBig view:view];

// 원격 사용자의 화면 크기를 언제든지 전환할 수 있습니다
[self.trtcCloud setRemoteVideoStreamType:userId
type:TRTCVideoStreamTypeSmall];
```

> ⚠ 주의:
> 듀얼 스트림 인코딩 활성화 후, `streamType` 비디오 스트림 유형을 `TRTCVideoStreamTypeSmall` 로 지정하여 저화질 소형 화면을 가져올 수 있습니다.

뷰 렌더링 컨트롤

업무에 표시 영역 전환과 관련된 인터랙티브 시나리오가 포함된 경우 RTC Engine SDK를 사용하여 로컬 미리보기 화면을 업데이트하고 원격 사용자 비디오 렌더링 컨트롤 기능을 구현할 수 있습니다.

```
// 로컬 미리보기 화면 렌더링 컨트롤의 업데이트
[self.trtcCloud updateLocalView:view];

// 원격 사용자 비디오 렌더링 컨트롤의 업데이트
[self.trtcCloud updateRemoteView:view streamType:TRTCVideoStreamTypeBig
forUser:userId];
```

> ⚠ 주의:
> 전달 매개변수인 `view` 를 대상 비디오 렌더링 컨트롤로 전달합니다. `streamType` 은 `TRTCVideoStreamTypeBig` 및 `TRTCVideoStreamTypeSub` 만 지원합니다.

라이브 방송의 인터랙티브 메시지

라이브 인터랙션은 라이브 방송 시나리오에서 특히 중요합니다.사용자는 좋아요 메시지, 선물 메시지, 탄막 메시지 등을 통해 스트리머와 인터랙티브합니다. 라이브 인터랙션 기능을 구현하기 위해서는 Chat 서비스를 활성화하고 Chat SDK를 임포트해야 하며, 자세한 내용은 음성 채팅방 접근 가이드-접근 준비 을 참조하십시오.

좋아요 메시지

1. 좋아요를 누른 사용자는 클라이언트에서 좋아요 관련 그룹 커스텀 메시지를 전송하며, 전송 성공 후 업무 측에서 로컬에 좋아요 효과를 렌더링합니다.

```
// 좋아요 메시지 본문 구성
NSDictionary *msgDict = @{
    @"type": @1,         // 좋아요 메시지의 유형
    @"likeCount": @10    // 좋아요 메시지의 수량
};
NSDictionary *dataDict = @{
    @"cmd": @"like_msg",
    @"msg": msgDict
};
NSError *error;
NSData *data = [NSJSONSerialization dataWithJSONObject:dataDict
options:0 error:&error];

// 그룹 커스텀 메시지 전송 (좋아요 메시지는 낮은 우선순위로 설정 권장드립니다)
[[V2TIMManager sharedInstance] sendGroupCustomMessage:data to:groupID
priority:V2TIM_PRIORITY_LOW succ:^{
    // 좋아요 메시지 전송 성공
    // 로컬에서  좋아요 효과 렌더링합니다
} fail:^(int code, NSString *desc) {
    // 좋아요 메시지 전송 실패
}];
```

2. 방 내 다른 사용자의 클라이언트에서는 그룹 사용자의 커스텀 메시지 콜백을 수신한 후 메시지 파싱 및 좋아요 효과의 렌더링을 수행합니다.

```
// 그룹 커스텀 메시지의 수신
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];
- (void)onRecvGroupCustomMessage:(NSString *)msgID groupID:(NSString
*)groupID sender:(V2TIMGroupMemberInfo *)info customData:(NSData
```

```
*)data {
    if (data.length > 0) {
        NSError *error;
        NSDictionary *dataDict = [NSJSONSerialization
JSONObjectWithData:data options:0 error:&error];
        if (!error) {
            NSString *command = dataDict[@"cmd"];
            NSDictionary *msgDict = dataDict[@"msg"];
            if ([command isEqualToString:@"like_msg"]) {
                NSNumber *type = msgDict[@"type"];            // 좋아
요 유형
                NSNumber *likeCount = msgDict[@"likeCount"];  // 좋아
요 수
                // 좋아요 메시지의 유형과 수량에 따라 좋아요 효과 렌더링합니다
            }
        } else {
            NSLog(@"파싱 오류: %@", error.localizedDescription);
        }
    }
}
```

선물 메시지

1. 선물 보내려고 하는 사용자가 업무 서버에 요청을 발송하면 업무 서버가 결제 정산을 완료한 후 REST API을 호출하여 그룹에 커스텀 메시지를 전송합니다.

   1.1 URL 요청의 예시.

```
https://xxxxxx/v4/group_open_http_svc/send_group_msg?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&conten
ttype=json
```

   1.2 패킷 요청의 예시.

```
{
    "GroupId": "@TGS#12DEVUDHQ",
    "Random": 2784275388,
    "MsgPriority": "High",  // 메시지의 우선순위이고 선물 메시지는 높은 우선
순위로 설정해야 합니다
    "MsgBody": [
```

```
        {
            "MsgType": "TIMCustomElem",
            "MsgContent": {
                // type: 선물 유형; giftUrl: 선물 리소스 주소; giftName:
선물 이름; giftCount: 선물 수량
                "Data": "{""cmd"": ""gift_msg", ""msg"": {""type"":
1, ""giftUrl"": ""xxx", ""giftName"": ""xxx", ""giftCount"": 1}}"
            }
        }
    ]
}
```

2. 방 내의 다른 사용자 클라이언트는 그룹 커스텀 메시지 콜백을 수신한 후 메시지 파싱 및 선물 효과 렌더링을 수행합니다.

```
// 그룹 커스텀 메시지의 수신
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];
- (void)onRecvGroupCustomMessage:(NSString *)msgID groupID:(NSString
*)groupID sender:(V2TIMGroupMemberInfo *)info customData:(NSData
*)data {
    if (data.length > 0) {
        NSError *error;
        NSDictionary *dataDict = [NSJSONSerialization
JSONObjectWithData:data options:0 error:&error];
        if (!error) {
            NSString *command = dataDict[@"cmd"];
            NSDictionary *msgDict = dataDict[@"msg"];
            if ([command isEqualToString:@"gift_msg"]) {
                NSNumber *type = msgDict[@"type"];            // 선물
유형
                NSNumber *giftCount = msgDict[@"giftCount"];  // 선물
수량
                NSString *giftUrl = msgDict[@"giftUrl"];      // 선물
리소스 주소
                NSString *giftName = msgDict[@"giftName"];    // 선물
이름
                // 선물 유형, 선물 수량, 선물 리소스 주소, 선물 이름에 따라
선물 효과를 렌더링합니다.
            }
```

```
        } else {
            NSLog(@"파싱 오류: %@", error.localizedDescription);
        }
    }
}
```

탄막 메시지

쇼 라이브 방송에는 일반적으로 텍스트 형식의 탄막 메시지가 나오며, 여기서는 Chat의 전송 및 그룹 채팅 일반 텍스트 메시지 수신을 통해 구현할 수 있습니다.

```
// 공개 탄막 메시지의 전송
[[V2TIMManager sharedInstance] sendGroupTextMessage:text to:groupID
priority:V2TIM_PRIORITY_NORMAL succ:^{
    // 탄막 메시지의 전송이 성공됩니다
    // 로컬에서 메시지 텍스트가 표시됨
} fail:^(int code, NSString *desc) {
    // 탄막 메시지의 전송이 실패됩니다
}];

// 공개 탄막 메시지의 수신
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];
- (void)onRecvGroupTextMessage:(NSString *)msgID groupID:(NSString
*)groupID sender:(V2TIMGroupMemberInfo *)info text:(NSString *)text {
    // 발신자 정보 info 및 메시지 텍스트 text에 따라 탄막 메시지 렌더링합니다
}
```

⚠ 주의:

- 선물 메시지는 높은 우선순위로 설정하는 것을 권장합니다. 탄막 메시지는 일반 우선순위로 설정하며, 좋아요 메시지는 낮은 우선순위로 설정하는 것을 권장합니다.
- 본인이 클라이언트에서 그룹 채팅 메시지를 보내면 메시지 수신 콜백이 트리거되지 않으며 그룹 내의 다른 사용자만 수신할 수 있습니다.

이상 처리

고장및 오류 처리

RTC Engine SDK에서 복구할 수 없는 오류가 발생하면 `onError` 콜백으로 전달되며, 자세한 내용은 오류 코드 표를 참조하세요.

1. UserSig 관련. UserSig 검증 실패로 인해 방 입장에 실패할 수 있으며, UserSig 도구를 사용하여 검증할 수 있습니다.

열거형	값	설명
ERR_TRTC_INVALID_USER_SIG	-3320	방 입장 매개변수 userSig가 올바르지 않습니다. `TRTCParams.userSig` 이 비어 있는지 확인하세요.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	UserSig 검증에 실패했습니다. 매개변수 `TRTCParams.userSig` 이 올바르게 입력되었는지 또는 만료되지 않았는지 확인하세요.

2. 입장및 퇴장 관련. 입장 실패 시 먼저 입장 매개변수가 올바른지 확인하고, 입장및 퇴장 인터페이스는 반드시 쌍으로 호출해야 합니다. 입장에 실패한 경우에도 퇴장 인터페이스를 호출해야 합니다.

열거형	값	설명
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	입장 요청 시간 초과, 네트워크 연결이 끊겼는지 또는 VPN이 켜져 있는지 확인하세요. 4G로 전환하여 테스트할 수도 있습니다.
ERR_TRTC_INVALID_SDK_APPID	-3317	입장 매개변수 sdkAppId가 잘못되었습니다. `TRTCParams.sdkAppId` 이 비어 있는지 확인하세요.
ERR_TRTC_INVALID_ROOM_ID	-3318	입장 매개변수 roomId가 잘못되었습니다. `TRTCParams.roomId` 또는 `TRTCParams.strRoomId` 이 비어 있는지 확인하세요. roomId와 strRoomId는 혼용할 수 없습니다.
ERR_TRTC_INVALID_USER_ID	-3319	입장 매개변수 userId가 올바르지 않습니다. `TRTCParams.userId` 이 비어 있는지 확인하세요.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	입장 요청이 거부되었습니다. `enterRoom` 을 연속적으로 호출하여 동일한 ID의 방에 입장했는지 확인하세요.

3. 장치 관련. 장치 관련 오류를 감지할 수 있으며, 관련 오류 발생 시 UI에서 사용자에게 알립니다.

열거형	값	설명
ERR_CAMERA_START_FAIL	-1301	Windows 또는 Mac 장치에서 카메라 구성 프로그램(드라이버)이 비정상일 경우, 카메라 열 수가 없습니다. 장치를 비활성화한 후 다시 활성화하거나, 기기를 재시작하거나, 구성 프로그램을 업데이트하세요.
ERR_MIC_START_FAIL	-1302	Windows 또는 Mac 장치에서 마이크 구성 프로그램(드라이버)에 이상이 있는 경우 마이크를 열 수 없습니다. 장치를 비활성화한 후

		다시 활성화하거나, 기기를 재시작하거나, 구성 프로그램을 업데이트하세요.
ERR_CAMERA_NOT_AUTHORIZED	-1314	카메라 장치에 권한이 없습니다. 일반적으로 모바일 장치에서 발생하며, 사용자가 권한을 거부했을 수 있습니다.
ERR_MIC_NOT_AUTHORIZED	-1317	마이크 장치에 권한이 없습니다. 일반적으로 모바일 장치에서 발생하며, 사용자가 권한을 거부했을 수 있습니다.
ERR_CAMERA_OCCUPY	-1316	카메라가 사용 중입니다. 다른 카메라를 열어 볼 수 있습니다.
ERR_MIC_OCCUPY	-1319	마이크가 사용 중입니다. 예를 들어 모바일 장치에서 통화 중일 때 마이크를 열 수가 없습니다.

원격 미러 모드의 무효 문제

RTC Engine 설정 화면 미러는 로컬 미리보기 미러 `setLocalRenderParams` 와 비디오 인코더 미러 `setVideoEncoderMirror` 로 나뉘며, 각각 로컬 미리보기 화면 미러 효과와 비디오 인코딩 출력 화면의 미러 효과(원격 시청자 및 클라우드 레코딩의 미러 모드)에 영향을 미칩니다. 로컬 미리보기의 미러 효과가 원격 시청자 측에서도 동시에 적용되도록 하려면 다음과 같이 코딩하십시오.

```
// 로컬 화면의 렌더링 매개변수를 설정합니다
TRTCRenderParams *params = [[TRTCRenderParams alloc] init];
params.mirrorType = TRTCVideoMirrorTypeEnable; // 화면 미러 모드
params.fillMode = TRTCVideoFillMode_Fill; // 화면 채우기 모드
params.rotation = TRTCVideoRotation_0; // 화면 회전 각도
[self.trtcCloud setLocalRenderParams:params];
// 인코더 출력 화면 미러 모드를 설정합니다
[self.trtcCloud setVideoEncoderMirror:YES];
```

카메라 줌/초점/전환 문제

쇼 라이브 방송 시나리오에서 스트리머는 카메라에 대한 커스텀 요구가 있을 수 있으며, RTC Engine SDK 장치 관리 클래스에는 이러한 요구를 해결하기 위한 관련 인터페이스가 있습니다.

1. 카메라 줌 배율의 조회 및 설정.

```
// 카메라 최대 줌 배율 가져오기(모바일 전용)
CGFloat zoomRatio = [[self.trtcCloud getDeviceManager]
getCameraZoomMaxRatio];
// 카메라 줌 배율의 설정(모바일 전용)
```

```
// 범위는 1-5이며, 1은 가장 먼 시야(일반 렌즈)이고 5는 가장 가까운 시야(확대 렌
즈)를 나타냅니다. 최대값은 5를 권장하며 5를 초과하면 비디오 데이터가 흐릿해질 수
있습니다
[[self.trtcCloud getDeviceManager] setCameraZoomRatio:zoomRatio];
```

2. 카메라 초점 기능 및 위치의 설정.

```
// 카메라 자동 초점 기능의 켜기 또는 끄기(모바일만 적용)
[[self.trtcCloud getDeviceManager] enableCameraAutoFocus:NO];
// 카메라 초점 위치의 설정(모바일 전용)
// 해당 인터페이스를 사용하려면 먼저 enableCameraAutoFocus로 자동 초점 기능을
꺼야 합니다
[[self.trtcCloud getDeviceManager]
setCameraFocusPosition:CGPointMake(x, y)];
```

3. 전면 또는 후면 카메라의 판단 및 전환.

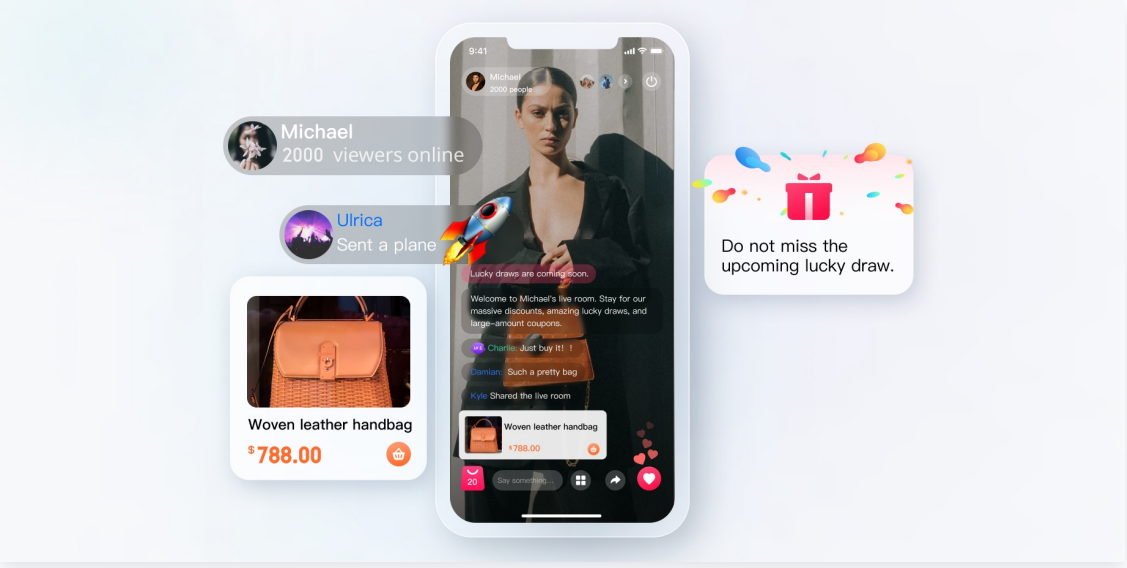
```
// 현재 전면 카메라인지 판단하세요(모바일만 적용)
BOOL isFrontCamera = [[self.trtcCloud getDeviceManager]
isFrontCamera];
// 전면 또는 후면 카메라의 전환(모바일만 적용)
// true 전달: 전면으로 전환; false 전달: 후면으로 전환
[[self.trtcCloud getDeviceManager] switchCamera:!isFrontCamera];
```

이커머스 라이브 방송 라이브 커머스 시나리오의 솔루션

최종 업데이트 날짜:: 2025-10-28 11:32:36

시나리오의 설명

라이브 커머스는 라이브 방송 플랫폼에서 스트리머와 시청자가 실시간으로 인터랙티브하며 상품의 소개및 홍보를 통해서 판매가 이루어지는 전자상거래의 새로운 방식입니다. 이 시나리오에서 스트리머는 일반적으로 의류, 화장품, 가정용품 등 다양한 상품을 라이브 방송실에서 보여주며 제품 특징, 할인 정보 및 사용 방법 등을 시청자에게 설명해줍니다. 시청자는 라이브 방송실에서 문의하고 댓글을 달며 상품을 구매할 수 있어 즉각적인 소통과 거래가 가능합니다. RTC Engine과 Chat 등의 제품을 활용하면 전자상거래 라이브 방송실의 구축을 쉽게 완료할 수 있습니다.



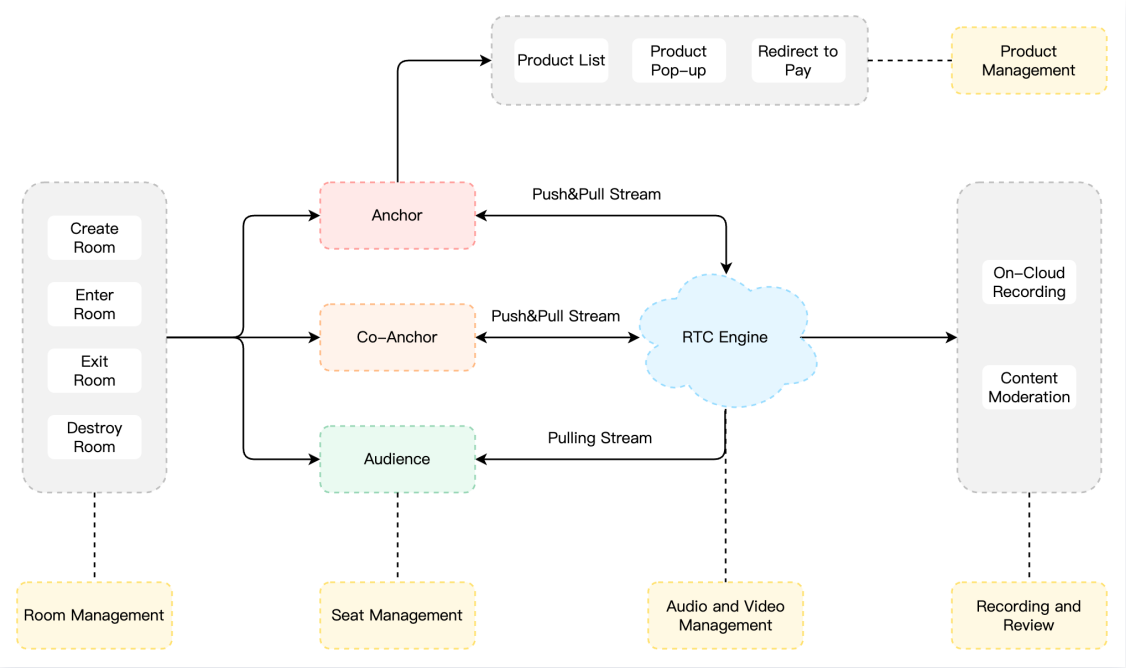
구현 방안

일반적으로 완전한 라이브 커머스 시나리오를 구현하려면 여러 기능 모듈이 필요합니다. 예를 들어 방 관리, 마이크 관리, 상품 관리, 음성및 영상 관리, 클라우드 레코딩 등 있습니다. 각 기능 모듈의 주요 작업 및 기능 포인트는 아래 표와 같습니다. 다음으로 각 기능 모듈을 하나씩 소개하며 전자상거래 라이브 방송 구축에 필요한 기능에 대해 비교적 완전한 인식을 갖출 수 있도록 하겠습니다.

기능 모듈	주요 작업 및 기능 포인트
방 관리	방 생성, 방 입장, 방 나가기, 방 파기
마이크 관리	마이크의 사용 신청, 자동으로 마이크 끄기, 마이크 사용 초청, 강제로 아미

	크 끄기, 마이크 금언
상품 관리	상품 목록 관리, 상품 팝업 관리, 이동 및 결제
음성 및 영상 관리	로컬 스트리밍, 원격 스트리밍, 시청자의 마이크 연결
클라우드 레코딩	RTC Engine 클라우드 레코딩

라이브 커머스 시나리오의 전체 업무 아키텍처는 아래 그림과 같습니다. 스트리머가 방을 생성하면 다른 사용자는 관심 있는 방을 선택해 입장할 수 있습니다. 방에 입장한 후 사용자는 마이크를 켜고 스트리머와 음성/영상으로 인터랙티브할 수 있습니다. 스트리머는 상품 목록과 상품의 설명 및 업데이트를 관리합니다. 일반적으로 규정 준수를 위해 라이브 방송실 내의 음성/영상 콘텐츠는 녹화되어 심사 받아야 합니다.



방 관리

방 관리 모듈은 주로 방 목록의 유지를 관리하며, 주요 기능은 다음과 같습니다.

- 방 생성: 사용자가 업무 시스템에 로그인한 후 방을 생성할 수 있으며 방 생성 후 방 목록에 추가 작업이 이루어집니다.
- 방 입장: 사용자는 기존 방에 입장할 수 있으며, 방에 입장한 후 현재 방의 인원 목록에 추가 작업이 이루어집니다.
- 방 퇴장: 사용자는 현재 방에서 나갈 수 있으며, 방을 나간 후 현재 방의 인원 목록에서 삭제 작업이 이루어집니다.
- 방 파기: 모든 사용자가 방을 나간 후 방을 파기해야 하며, 방 파기 후 방 목록에서 삭제 작업이 이루어집니다.

> **설명:**
> 방 관리는 전자상거래 라이브 방송을 구현하는 데에 필요한 모듈이지만 주요 기능 모듈은 아닙니다. 구체적으로는 업무 시스템 및 Chat & RTC Engine SDK와 결합하여 구현할 수 있으며, 자세한 내용은 음성 채

팅방-방 관리을 참조하세요.

마이크 관리

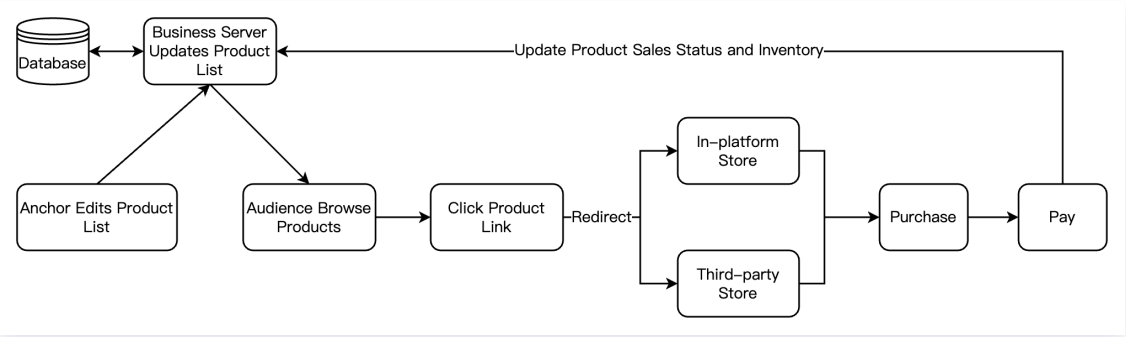
라이브 방송실 내의 마이크 위치는 일반적으로 순서가 정해져 있고 제한적입니다. 마이크 관리는 주로 업무 시나리오에 따라 방 내의 마이크 수량을 정의하고, 현재 방의 모든 마이크 상태를 관리하는 역할을 합니다. 마이크 관리에는 주로 마이크 사용의 신청, 자동으로 마이크 끄기, 아미크 사용 초청, 강제로 아미크 끄기, 마이크 금언등이 포함됩니다.

- 사용자가 방에 입장한 후 비어 있는 상태의 마이크만에 대해서 사용을 신청할 수 있습니다.
- 스트리머가 사용자의 마이크 사용 신청을 허락한 후, 마이크 상태를 비어 있지 않음으로 수정해야 합니다.
- 연결된 사용자가 아미크 끈 후에 로컬 스트리밍을 중지하고 마이크 상태를 재설정해야 합니다.
- 스트리머는 마이크 잠금, 마이크 사용 초청, 강제로 아미크 끄기, 마이크 금언 등 권한이 있습니다.

설명:
마이크 관리는 전자상거래 라이브 방송을 구현하는 데에 필요한 모듈이지만 주요 기능 모듈은 아닙니다. 구체적으로는 업무 시스템 및 Chat & RTC Engine SDK와 결합하여 구현할 수 있으며, 자세한 내용은 음성 채팅방-마이크 관리을 참조하세요.

상품 관리

상품 관리 모듈은 전자상거래 라이브 방송 시나리오에 특화된 것이고 일반적으로 상품 목록 관리, 상품 팝업 관리, 상품 링크 이동 및 결제 등을 포함합니다. 아래 그림으로 상품 관리의 기본 프로세스를 보여줍니다.



- **상품 목록 관리**

  상품 목록 관리는 상품 관리의 기본 기능이고 주로 상품 추가, 삭제, 수정 및 조회 등 기능을 포함합니다. 일반적으로 백엔드 데이터베이스에 상품명, 설명, 가격, 재고, 이미지 등 다양한 상품 정보를 저장합니다. 프론트엔드에서는 API 인터페이스를 통해 이러한 정보를 가져와서 사용자에게 목록 형태로 보여줄 수 있습니다.
- **상품 팝업 관리**

  라이브 커머스 진행 시 스트리머가 상품을 설명하고 링크 올릴 때 시청자 측에 해당 상품 정보를 팝업으로 표시하여 시청자가 쉽게 찾아보고 구매할 수 있도록 안내해줘야 합니다. 상품 정보 팝업 기능은 다음 두 가지 방법으로 구현할 수 있으며, 업무 요구에 따라 선택하여 사용할 수 있습니다.
  - 자체 정의 메시지

일반적으로 상품 정보 팝업은 라이브 방송실에 자체 정의 메시지를 전송하는 방식으로 구현할 수 있으며, 라이브 방송실의 시청자는 이 자체 정의 메시지를 수신한 후 분석하여 표시합니다. 자체 정의 메시지의 송부/수신은 업무 측에서 자체적으로 구현하거나 Chat 그룹 메시지를 사용하여 구현할 수 있습니다. 구체적인 구현 방법은 상품 정보 팝업-자체 정의 메시지을 참조하세요.

- SEI 정보

SEI(Supplemental Enhancement Information)는 보완 강화 정보로서 비디오 스트림에 별도의 정보를 추가하는 방법을 제공합니다. RTC Engine SEI 정보 전송을 통해 지정된 상품 정보를 스트리머의 비디오 스트림에 삽입할 수 있으며, 라이브 방송실의 시청자가 스트림을 받아 SEI 메시지를 수신한 후 이를 분석하여 표시할 수 있습니다. SEI의 특성에 따라 이 방식은 상품 정보 팝업과 스트리머의 라이브 화면을 정확하게 동기화할 수 있습니다. 구체적인 구현 방법은 상품 정보 팝업-SEI 정보을 참조하세요.

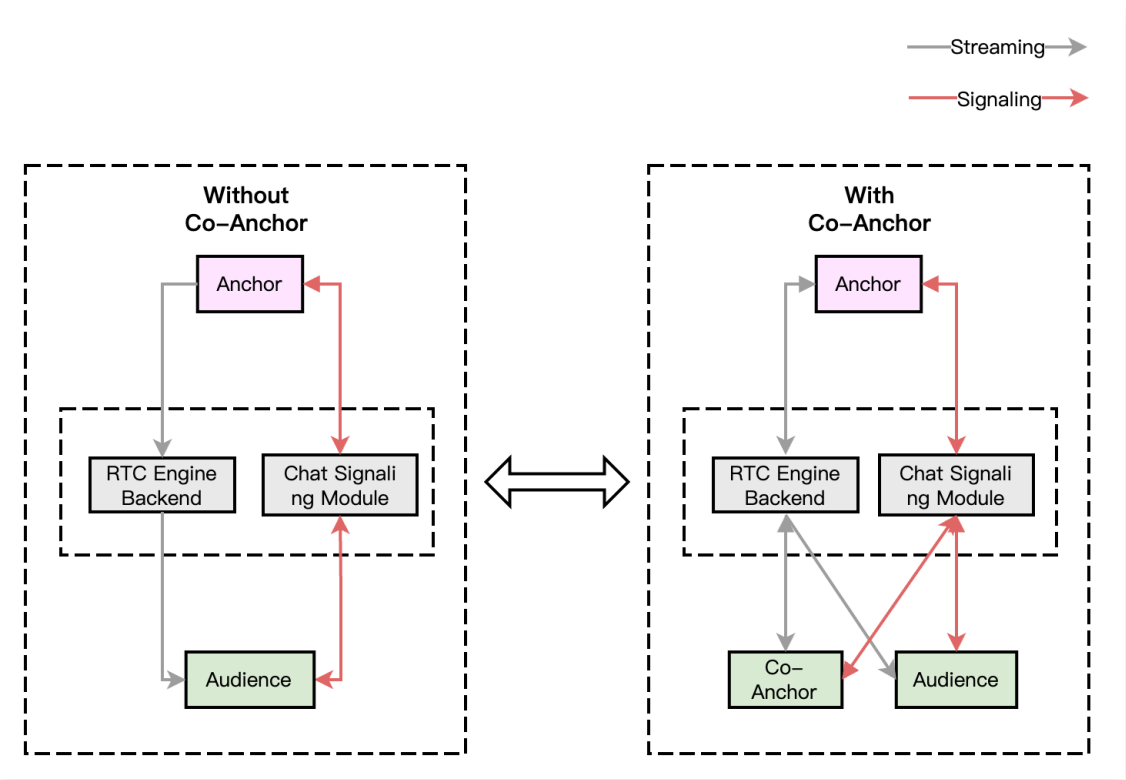
- **상품 이동 및 결제**

라이브 방송실의 시청자가 상품 선택을 완료한 후, 상품 링크를 클릭하여 해당 전자상거래 매장으로 이동하고 관련 오더 확인하여 결제합니다. 여기서 전자상거래 매장은 플랫폼 내의 매장이거나 통합된 제3자 플랫폼 매장일 수 있습니다. 사용자 결제가 완료되면 상품 판매 상태 및 재고 정보 등을 업데이트하기 위해 결제 결과를 확인해야 합니다.

> **설명:**
> 상기 상품 관리 모듈은 참고용으로만 제공되며, 실제 적용 시에는 업무 요구사항에 따라 자체적으로 설계 및 배포해야 합니다.

음성 및 영상 관리

일반적인 전자상거래 라이브방송 시나리오(시청자 인원수가 10만 명 이하)의 경우 **RTC 실시간 인터랙티브 솔루션**을 추천합니다.스트리머와 시청자 모두 RTC 프로토콜을 사용하여 스트림을 푸시 및 풀하는 이 방식은 엔드투엔드 지연 시간이 가장 짧으며 시청자의 마이크 켜고 끄는 체험이 더 원활하고 화면 빠른 진행/되감기 등의 급격한 변화 현상이 없습니다. 다인원 마이크 연결 인터랙티브 라이브 방송을 예로 들면, 전자상거래 라이브 방송 시나리오에서 순수 RTC 스트림 푸시/풀의 주요 아키텍처는 아래 그림과 같습니다.



해당 솔루션 아키텍처의 전체 프로세스는 다음과 같습니다.

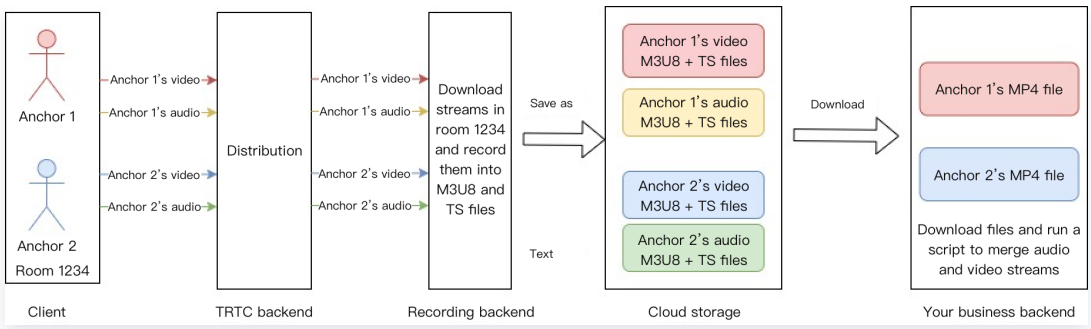
1. 스트리머와 시청자는 모두 시그널링 모듈을 통해 연결되며, 시그널링 모듈은 주로 라이브 방송 프로세스 관리 및 라이브 방송 상태의 동기화를 합니다.
2. 연결된 시청자가 있든 없든, 스트리머와 시청자는 모두 RTC Engine 오디오/비디오 클라우드 서비스를 통해 스트림을 푸시 및 풀합니다.
3. 시청자가 스트리머와의 연결을 요청하면 신시그널링 모듈이 스트리머에게 알리고 연결자의 개인 정보를 동기화합니다.
4. 스트리머가 연결 요청을 허락하면 연결된 시청자가 스트리밍을 시작하고 방의 모든 멤버는 스트림 업데이트 알림을 받으며 연결된 시청자의 오디오/비디오 스트림을 풀합니다.
5. 연결된 시청자가 연결 해제 요청을 하면 스트리밍을 중지하고 방의 모든 멤버는 스트림 업데이트 알림을 받으며 해당 시청자의 오디오/비디오 스트림 풀링을 중단합니다.

> **설명:**
> 시그널링 모듈은 업무 자체 개발 시그널링 채널일 수 있으며, 동시에 Chat을 사용하여 시그널링을 교체하는 것을 권장합니다.

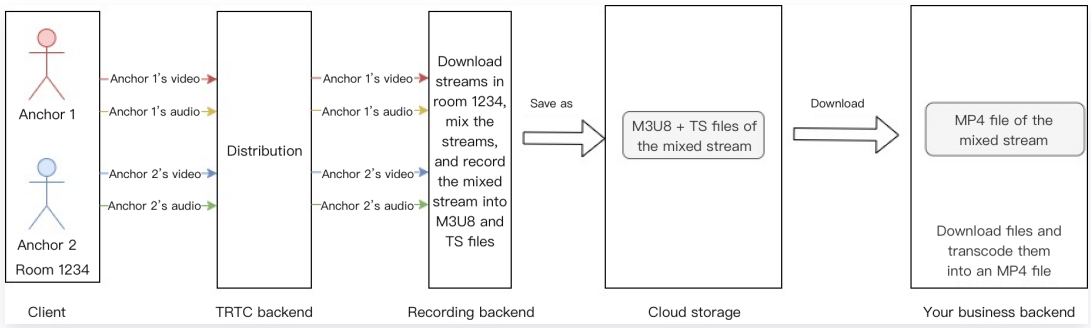
클라우드 레코딩

RTC Engine 최신 업그레이드된 클라우드 레코딩은 RTC Engine 내부의 실시간 레코딩 클러스터를 사용하여 오디오/비디오를 레코딩하며 더 완전하고 통일된 레코딩 체험을 제공합니다.

- 단일 스트림 레코딩: RTC Engine의 클라우드 레코딩 기능을 통해 방 안의 각 사용자의 오디오/비디오 스트림을 독립적인 파일로 레코딩할 수 있습니다.



- 혼합 스트림 레코딩: 동일한 방의 오디오/비디오 미디어 스트림을 혼합하여 하나의 파일로 레코딩합니다.



> **설명:**
> RTC Engine 클라우드 레코딩의 상세 소개 및 개설 안내는 RTC Engine 클라우드 레코딩 설명 을 참조하십시오.

핵심 업무 로직

상품 설명의 재방송

상품 설명의 재방송은 전자상거래 라이브 방송 시나리오의 필수 기능이며 일반적으로 라이브 방송 중의 재방송과 라이브 방송 후 재방송으로 나눕니다. 상품 설명의 재방송은 방송살에 늦게 입장하거나 라이브 방송을 놓친 사용자가 스트리머의 상품 설명을 자체적으로 다시 볼 수 있도록 도와줘서 판매량을 증가시키고 전환율을 높일 수 있습니다. 상품 설명 재방송 기능은 다음 두 가지 방식으로 구현할 수 있습니다.

레코딩 재방송

레코딩 재방송은 비교적 흔히 사용되는 상품 설명 재방송 기능의 구현 방식이며 구현이 간단하고 재방송 시점에 제한이 없습니다. 다음은 레코딩 재방송을 사용하여 상품 설명 재방송을 구현하는 기본 프로세스입니다.

1. **소개 자료 레코딩**
   - 레코딩 모드를 지정합니다(RecordMode)
     단일 스트림 레코딩: 방 안의 각 스트리머의 오디오 및 비디오 스트림을 개별적인 오디오/비디오 파일로 레코딩하여 클라우드 스토리지 플랫폼에 업로드합니다.
     합성 스트림 레코딩: 방 안에서 구독한 모든 스트리머의 오디오 및 비디오 스트림을 혼합하여 하나의 오디오/비디오 파일로 레코딩하여 클라우드 스토리지 플랫폼에 업로드합니다.
   - 저장 위치 및 레코딩 형식을 지정합니다(StorageParams)

저장 위치: 클라우드 VOD 또는 객체 저장소 COS에 저장할 수 있으며, AWS storage도 지원됩니다. StorageParams 매개변수를 통해 CloudStorage(COS 저장 매개변수) 또는 CloudVod(VOD 저장 매개변수)를 지정할 수 있습니다. VOD와 객체 저장소를 동시에 설정하는 것은 지원되지 않습니다.
레코딩 형식: COS에 저장할 때 기본 레코딩 형식은 HLS이며, RecordParams의 OutputFormat 매개변수를 통해 레코딩 파일 형식을 수정할 수 있습니다. VOD에 저장할 때 기본 레코딩 형식은 MP4이며, TencentVod의 MediaType 매개변수를 통해 레코딩 파일 형식을 수정할 수 있습니다.

- 레코딩 작업의 시작 (CreateCloudRecording)

  손님의 백엔드 서비스를 통해 REST API(CreateCloudRecording)을 호출하여 클라우드 레코딩을 시작할 때, **작업 ID(TaskId)** 매개변수에 특히 주의해야 합니다. 이 매개변수는 해당 레코딩 작업의 유일한 식별자이며 이후 이 레코딩 작업에 대한 인터페이스 조작을 위한 입력 매개변수로 이 작업 ID를 저장해야 합니다.

  > ⚠ **주의:**
  > 클라우드 레코딩 작업을 시작하는 인터페이스 CreateCloudRecording에서는 레코딩 로봇의 입장 매개변수 UserId와 UserSig(UserSig 획득 방법)을 지정해야 합니다. 해당 UserId는 방 내의 일반 스트리머 또는 시청자가 사용하는 UserId와 중복되어서는 안 되며, 현재 레코딩 중인 방에 지정된 레코딩 로봇의 UserId와도 일치해서는 안 됩니다. 그렇지 않으면 레코딩 작업이 실패할 수 있습니다.

- 레코딩 작업의 중지 (DeleteCloudRecording)

  REST API(DeleteCloudRecording)를 통해 백엔드 서비스를 호출하여 클라우드 레코딩 작업을 적절한 시기에 중지할 수 있습니다. 이때 레코딩 작업 시작 시 반환된 작업 ID(TaskId) 매개변수를 전달해야 합니다.

2. **재생 주소 가져오기**

- 방법 1: 수동으로 찾기

  레코딩 작업 종료 후, RTC Engine 레코딩 시스템에서 레코딩된 파일은 지정한 클라우드 스토리지 플랫폼(클라우드 VOD 또는 객체 저장소 COS 또는 AWS storage)에 업로드됩니다. 클라우드 VOD 콘솔 또는 객체 저장소 COS 콘솔에서 직접 대상 레코딩 미디어 파일을 찾고 수동으로 재생 주소를 가져올 수 있습니다.

- 방법 2: 콜백 수신

  또한 콘솔 에서 레코딩 콜백에서 주소를 구성할 수도 있습니다. 텐센트 클라우드가 새 레코딩 파일의 메시지를 손님의 서버로 자동으로 푸시하도록 할 수 있습니다. 레코딩 파일이 전송 완료되면 RTC Engine은 콘솔에서 설정한 콜백 주소(HTTP/HTTPS)를 통해 손님의 서버에 알림을 보냅니다. RTC Engine은 레코딩 및 레코딩 관련 이벤트를 모두 설정한 콜백 주소를 통해 손님의 서버에 푸시합니다. **이벤트 유형이 311**이고 업로드가 성공된 콜백을 수신하여 레코딩 파일의 재생 주소 **VideoUrl**을 얻을 수 있습니다. 콜백 예시 정보는 다음과 같습니다.

```
{
  "EventGroupId": 3,
  "EventType": 311,
  "CallbackTs": 1622191965320,
```

```
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "TencentVod": {
        "UserId": "xx",
        "TrackType": "audio_video",
        "MediaId": "main",
        "FileId": "xxxx",
        "VideoUrl": "http://xxxx",
        "CacheFile": "xxxx.mp4",
        "StartTimeStamp": xxxx,
        "EndTimeStamp": xxxx
      }
    }
  }
}
```

3. **레코딩된 영상의 재생**

   사전 준비 작업을 완료한 후 TXVodPlayer의 `startVodPlay` 을 호출하여 레코딩된 상품 설명의 동영상을 재생할 수 있습니다. TXVodPlayer는 내부적으로 재생 프로토콜을 자동으로 인식하므로 재생 URL을 `startVodPlay` 함수에 전달하기만 하면 됩니다. 더 많은 코드 예제는 빠른 시작 가이드-상품 설명 재생를 참조하십시오.

Android

```
// URL 비디오 리소스의 재생
String url = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
mVodPlayer.startVodPlay(url);
```

iOS

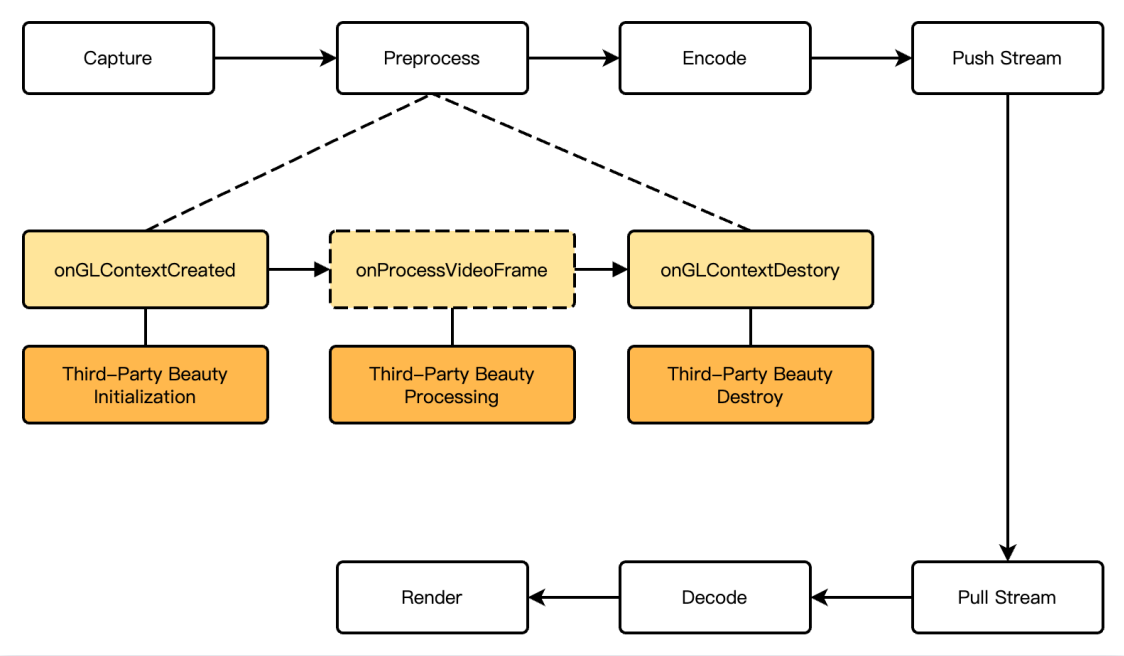
```
// URL 비디오 리소스의 재생
NSString* url = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
[_txVodPlayer startVodPlay:url];
```

> **설명:**
> 주문형 재생에는 텐센트 클라우드 플레이어 SDK의 사용이 필요하지만, 별도로 통합할 필요 없이 전체 기능 버전의 LiteAVSDK만 통합하면 됩니다. 자세한 내용은 신속 시작 가이드–SDK 임포트을 참조하세요.

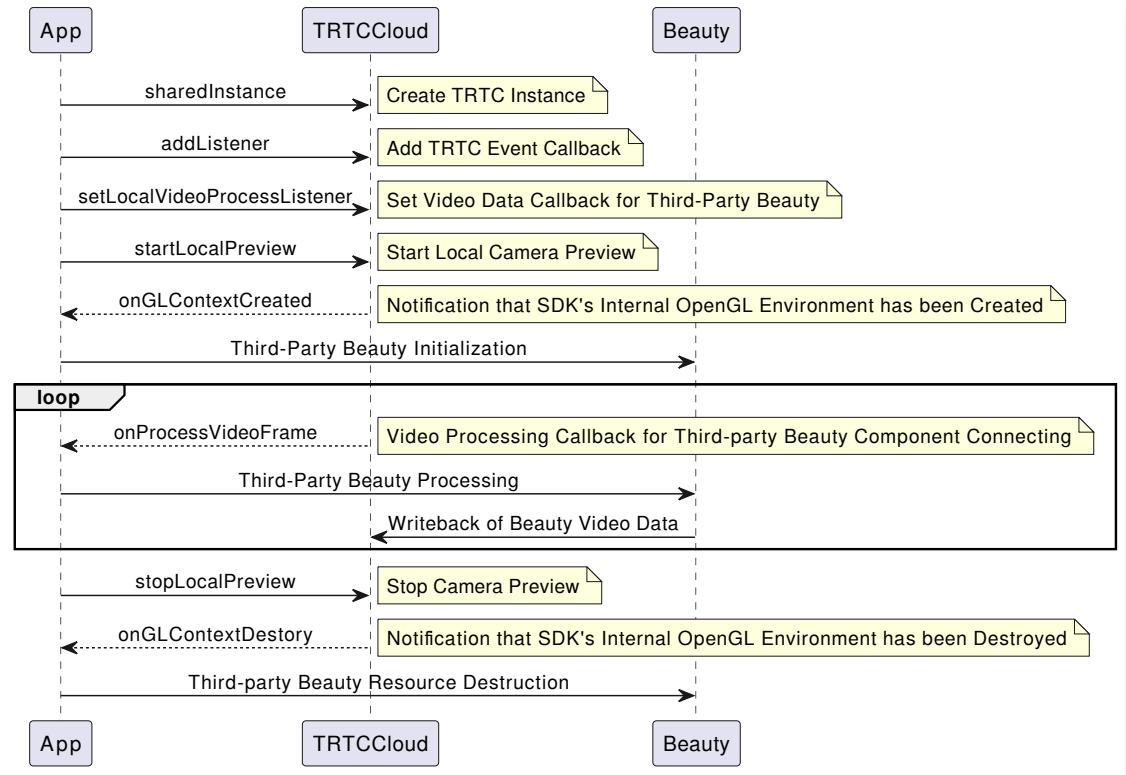
뷰티 효과 접근

전자상거래 라이브방송 시나리오에서 뷰티는 자주 사용되는 기능 중의 하나입니다. 뷰티는 스트리머의 외모를 개선할 뿐만 아니라 다양한 스티커 효과를 통해 라이브 방송 인터랙티브의 재미를 더할 수 있습니다. RTC Engine은 뷰티 AR SDK의 통합을 지원하며, 화산 뷰티, 얼굴 이펙트와 같은 시장의 주요 타사 뷰티 제품도 지원합니다.

뷰티 접근 절차



API 호출 타이밍



뷰티 제품의 비교

뷰티 유형	뷰티 효과	접속 비용	비용	가상 디지털 휴먼	단말기 지원
뷰티 AR SDK	기본 효과가 우수하며 고급 효과인 큰 눈/얼굴 슬림 등 효과가 뛰어납니다.	낮음	적당함	지원	Android/iOS/PC/Flutter/Web/미니프로그램
얼굴 이펙트 SDK	기본 효과가 우수하며 고급 효과인 큰 눈/얼굴 슬림 등 효과가 보통입니다.	높음	적당함	지원	Android/iOS/PC/Unity
화산 이펙트 SDK	기본 효과가 우수하며 고급 효과인 큰 눈/얼굴 슬림 등 효과가 좋습니다.	높음	높은 편	지원	Android/iOS/PC/Linux

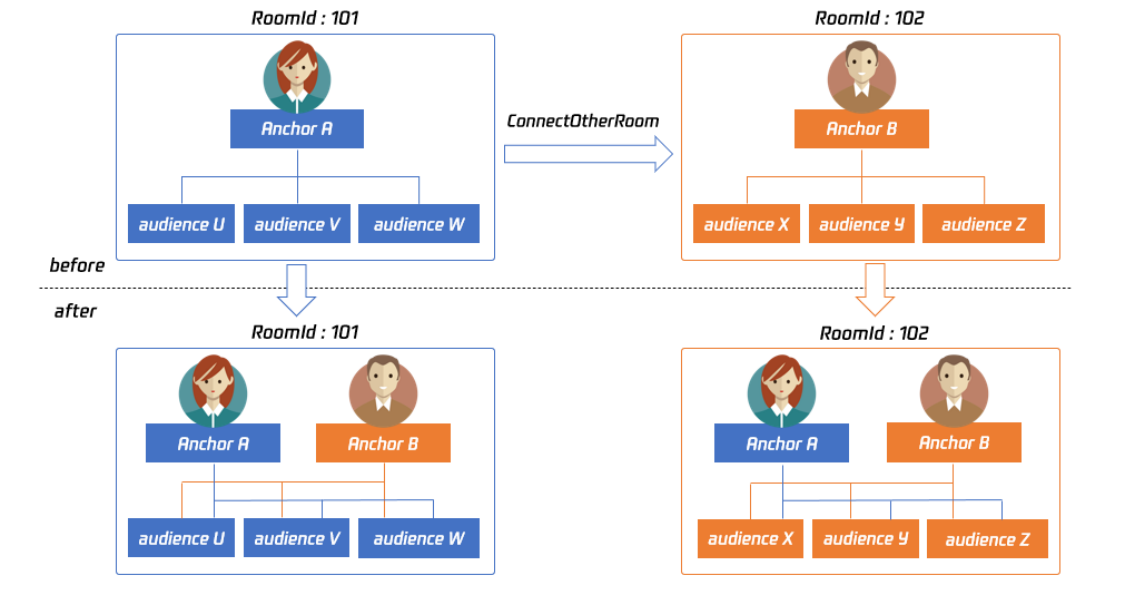
스트리머 간 크로스 룸 PK 연결

스트리머 간의 크로스룸 PK 연결은 전자상거래 라이브 방송 시나리오에서 새로운 플레이 방식이고 인터랙티브 PK는 라이브 방송의 재미를 높일 뿐만 아니라 어느 정도 시청자의 구매 욕구를 자극할 수 있습니다. RTC Engine

은 여러 방과 여러 스트리머 간의 크로스룸 PK 연결을 지원하며 구체적인 구현 방법은 아래와 같습니다.

1. **구현 원리**

기본적으로 동일한 방의 사용자들 간에만 음성 및 영상 통화가 가능하며 다른 방 간의 음성 및 영상 스트림은 서로 격리되어 있습니다. 크로스 룸 연결을 통해 다른 방의 특정 스트리머의 음성 및 영상 스트림을 자신이 있는 방에 게시할 수 있으며, 동시에 자신의 음성 및 영상 스트림을 대상 스트리머의 방에 게시할 수 있습니다. 이렇게 하여 서로 다른 두 방에 있는 스트리머들이 방을 넘어 음성 및 영상 스트림을 공유할 수 있게 되어 각 방의 시청자들이 두 스트리머의 음성 및 영상을 같이 볼 수 있습니다.



위 그림은 크로스 룸 PK 연결의 주요 프로세스를 보여줍니다. 예를 들어, 방 '101'의 스트리머 A가 `ConnectOtherRoom()` 을 통해 방 '102'의 스트리머 B와 크로스 룸 통화를 진행한 후,

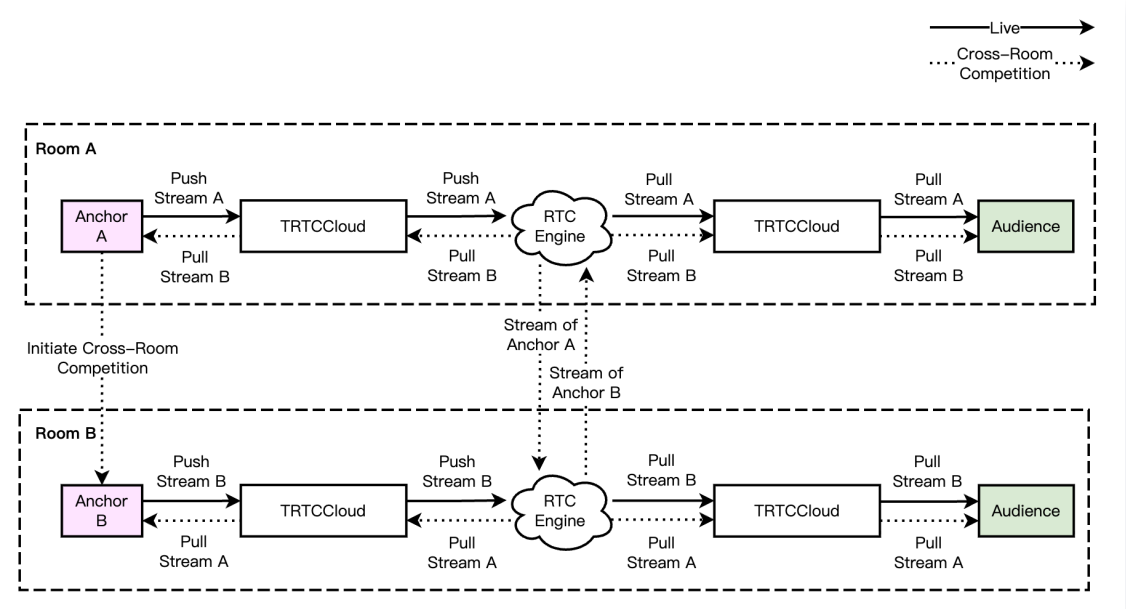
- 방 "101"의 사용자들은 모두 스트리머 B의 `onRemoteUserEnterRoom(B)` 및 `onUserVideoAvailable(B,true)` 이 두 가지 이벤트 콜백을 받게 되며, 이는 방 "101"의 사용자들이 모두 스트리머 B의 음성 및 영상을 구독할 수 있음을 의미합니다.
- 방 "102"의 사용자들은 모두 스트리머 A의 `onRemoteUserEnterRoom(A)` 및 `onUserVideoAvailable(A,true)` 이 두 가지 이벤트 콜백을 받게 되며, 이는 방 "102"의 사용자들이 모두 스트리머 A의 음성 및 영상을 구독할 수 있음을 의미합니다.

⚠ 주의:

- 크로스 룸 연결 PK의 로컬 사용자와 상대방 사용자는 모두 스트리머 역할로 되어야 하며, 오디오/비디오 업스트림이 모두 있어야 합니다.
- `ConnectOtherRoom()` 을 여러 번으로 호출하여 여러 방의 스트리머와 크로스 룸 연결을 구현할 수 있으며, 현재 한 방은 최대 다른 세 개 방의 스트리머와 크로스 룸 연결이 가능하고, 한 방에서 최대 10명의 스트리머가 다른 방의 스트리머와 크로스 룸 연결을 할 수 있습니다.
- RTC Engine 크로스 룸 연결 PK는 `createSubCloud()` 을 통해 서브 인스턴스를 생성하여 상대방 방에 가입해서 스트리밍을 주고받는 방식으로도 구현할 수 있으며, 현재 서브 인스턴스의 수량에는 제한이 없어서 향후 여러 방 간 PK 또는 여러 스트리머 간 PK의 업무 확장이 용이해집니다.

2. **실시간 인터랙티브 크로스 룸 연결 프로세스**

RTC 실시간 인터랙티브 솔루션 크로스 룸 PK 연결 프로세스는 전반적으로 간단하며, 스트리머와 크로스 룸 연결된 스트리머가 서로 RTC 단일 스트림을 풀하고, 시청자는 동시에 스트리머와 크로스 룸 연결된 스트리머의 RTC 단일 스트림을 풀하며, 시청자는 스트리머와 크로스 룸 연결된 스트리머의 미디어 스트림 구독 로직을 독립적으로 관리할 수 있습니다. RTC 실시간 인터랙티브 크로스 룸 연결 프로세스는 아래 그림과 같습니다.



> **주의:**
> 실시간 인터랙티브 크로스 룸 연결 시나리오에서 방 내 시청자는 크로스 룸 연결 스트리머의 미디어 스트림 구독 로직을 독립적으로 제어할 수 있으며 스트리머 가 크로스 룸 스트리머의 본 방 내의 업스트림 능력을 변경 할 수도 있습니다.

시나리오 플레이

단일 스트리머 라이브 커머스

단일 스트리머 라이브 커머스는 전자상거래 라이브 방송 시나리오에서 가장 흔하고 기본적인 플레이 방식입니다. 이 시나리오에서는 라이브 방송실에 하나의 스트리머만 존재하며, 마이크 연결 스트리머와 같은 역할은 없습니다. 시청자는 라이브 방송실에 들어가서 방송을 시청하고 방송실에서 인터랙티브 및 쇼핑 등을 할 수 있습니다.

다인원 아미크 연결 인터랙티브 라이브 커머스

다인원 마이크 연결 인터랙티브 라이브 커머스는 단일 스트리머 라이브 커머스 시나리오를 기반으로 하여 시청자가 마이크 켜고 사용하며 스트리머와 인터랙티브하는 플레이 방식을 추가한 것입니다. 스트리머는 시청자에게 마이크 사용의 초청을 할 수 있으며 마이크를 관리합니다. 시청자도 스스로 마이크의 연결을 신청해서 스트리머와 인터랙티브할 수 있습니다. 다인원 마이크 연결 인터랙티브 라이브 커머스는 시청자의 참여감을 높이고 시청자의 적극성을 이끌어낼 수 있습니다.

크로스 룸 PK 연결 라이브 커머스

기존의 단일 방 스트리머 라이브 커머스 외에도, 스트리머는 다른 라이브 방송의 스트리머와 크로스 룸 PK 연결을 통해 각자 판매하는 상품을 선보이고 시청자의 구매 열기를 자극하며 라이브의 재미를 더할 수 있습니다.

방안 관련 제품

시스템 계층	제품명	시나리오 용도
접근 계층	RTC Engine	시간 저지연 및 고품질의 다인원 사용자 실시간 음성/영상 인터랙션 라이브 방송 솔루션을 제공하며 전자상거래 라이브 방송 시나리오의 기반 인프라 기능입니다.
접근 계층	Chat	기반으로 된 그룹 기능의 방 관리 및 마이크 관리 능력을 제공하며, 라이브 방송 참가자 전체 메시지, 공개 화면 메시지 등의 풍부한 미디어 메시지 송부/수신 및 사용자 자체 정의 시그널링 등 통신 요구를 구현합니다.
접근 계층	뷰티 AR SDK	뷰티, 필터, 메이크업, 재미있는 스티커, Moji 이모티콘, 가상 아바타 등 실시간 특수 효과 처리 기능을 제공합니다.
클라우드 서비스	클라우드 VOD	음성, 영상, 이미지 등 미디어를 위해 제작 업로드, 저장, 트랜스코딩, 미디어 처리, 미디어 AI, 가속 분배 재생, 저작권 보호 등 일체형의 고품질 미디어 서비스를 제공합니다.
데이터 저장	객체 저장 COS	음성 및 영상 녹화 파일, 음성 및 영상 슬라이스 파일의 저장 서비스를 제공합니다.

고속 접속 가이드
Android

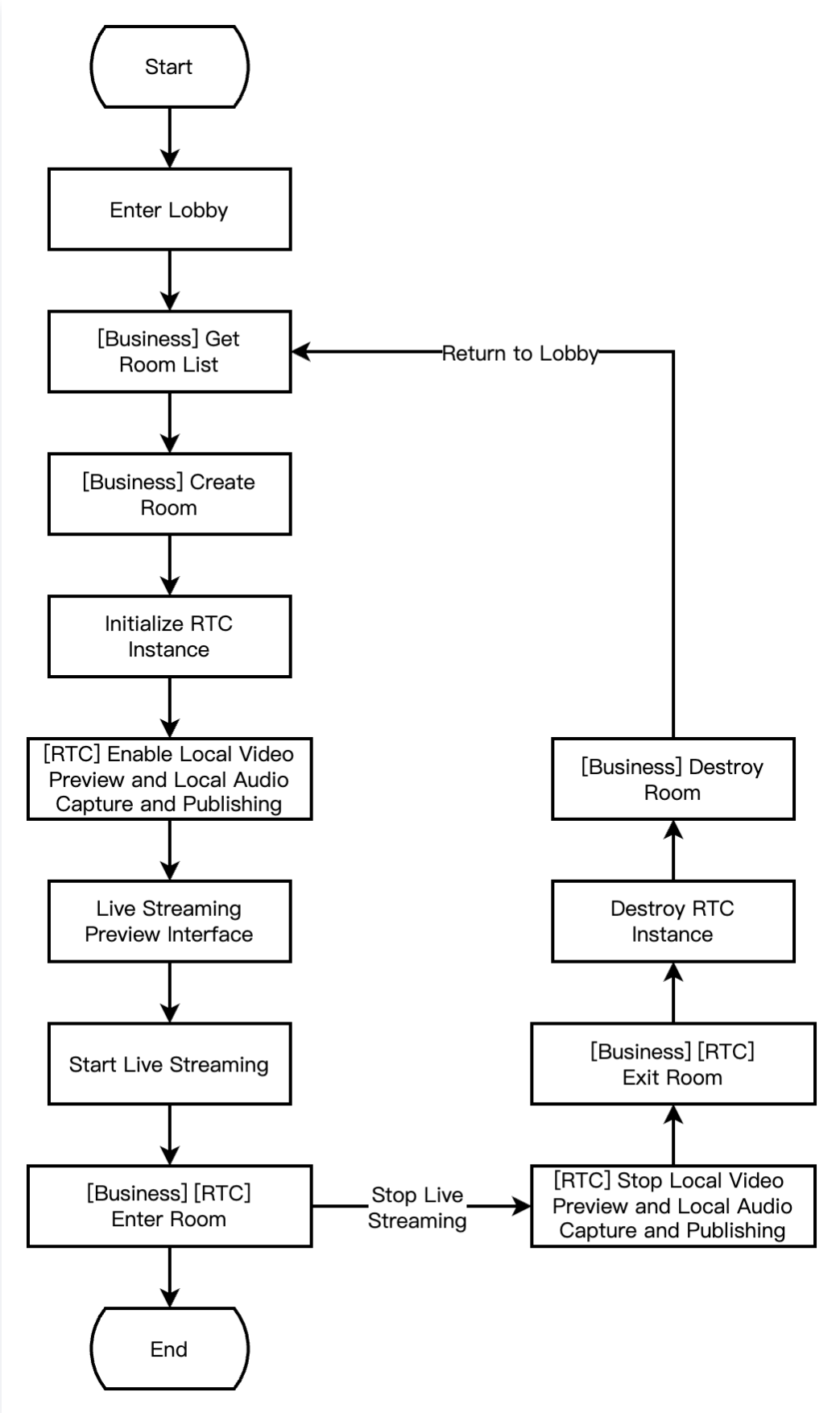
최종 업데이트 날짜:: 2025-10-28 11:32:37

업무 프로세스

본 섹션은 전자상거래 라이브 방송 업무 시나리오에서 일반적으로 사용되는 업무 프로세스를 종합하여 전체 시나리오 구현 프로세스를 더 잘 이해할 수 있도록 도와줍니다.

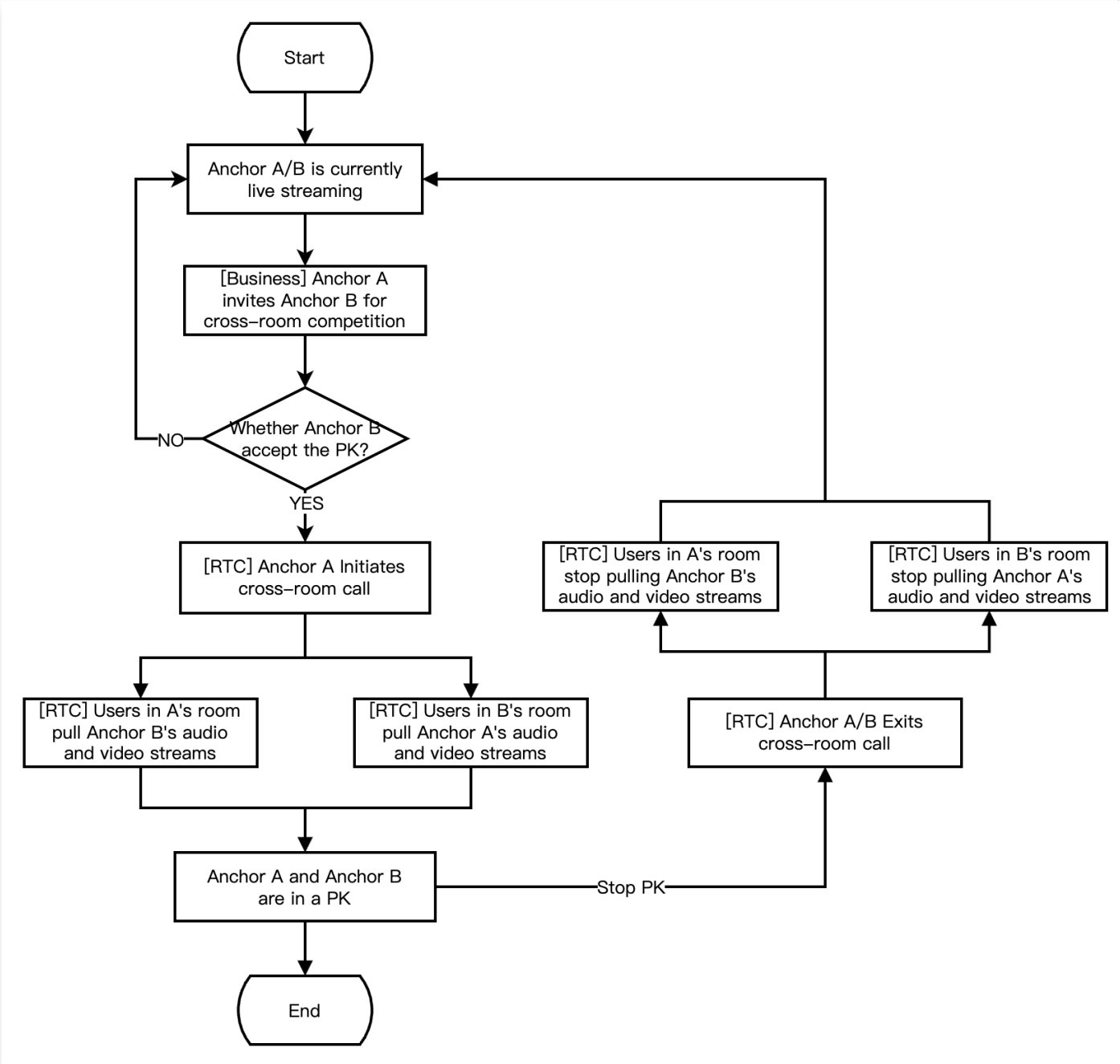
스트리머의 방송 시작 및 종료

아래 그림으로 스트리머(방주인)의 로컬 미리보기, 방 생성, 방 입장하여 방송 시작, 방 퇴장 및 방송 종료 프로세스를 보여줍니다.



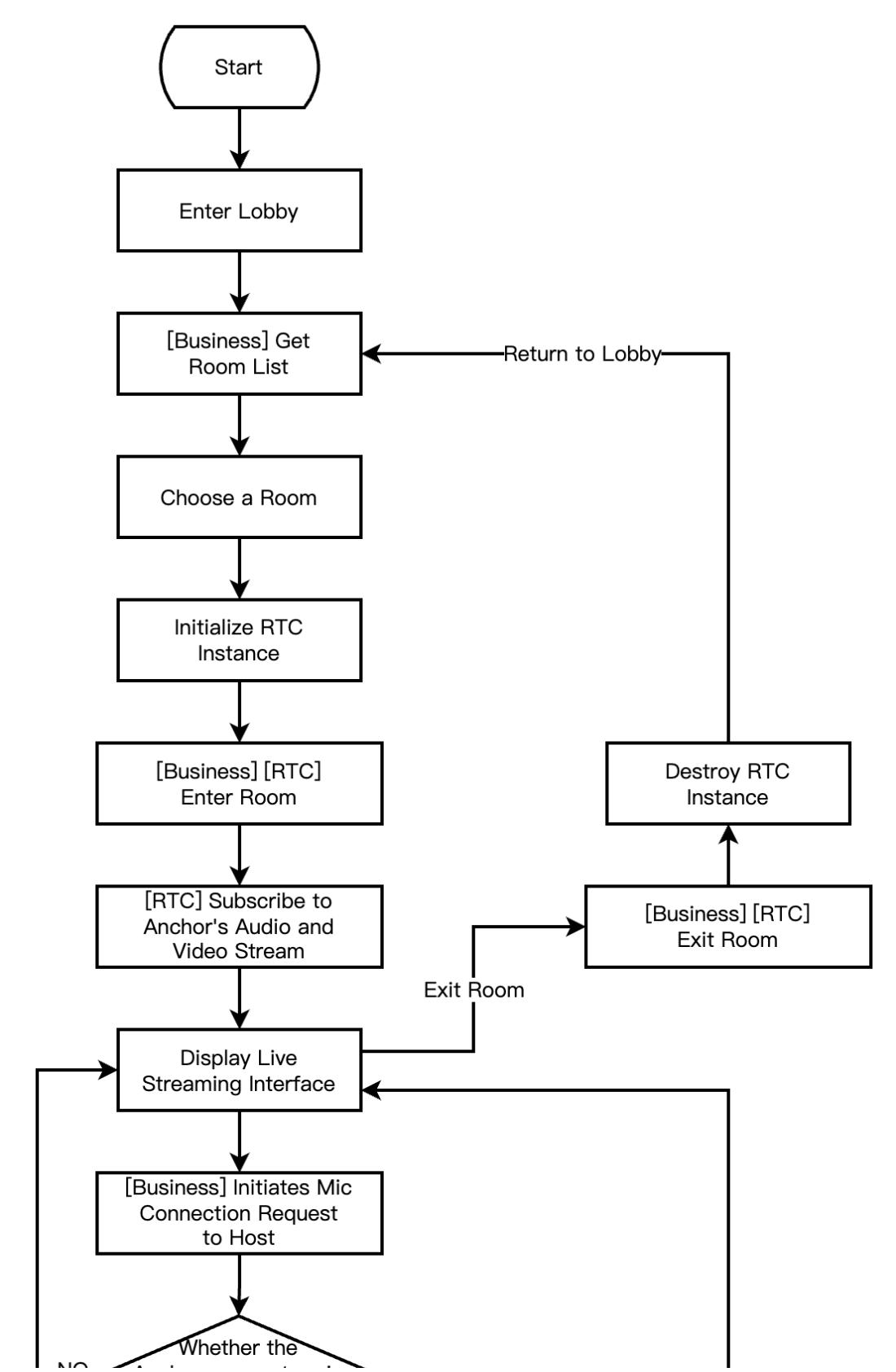
스트리머 간 크로스 룸 PK 연결

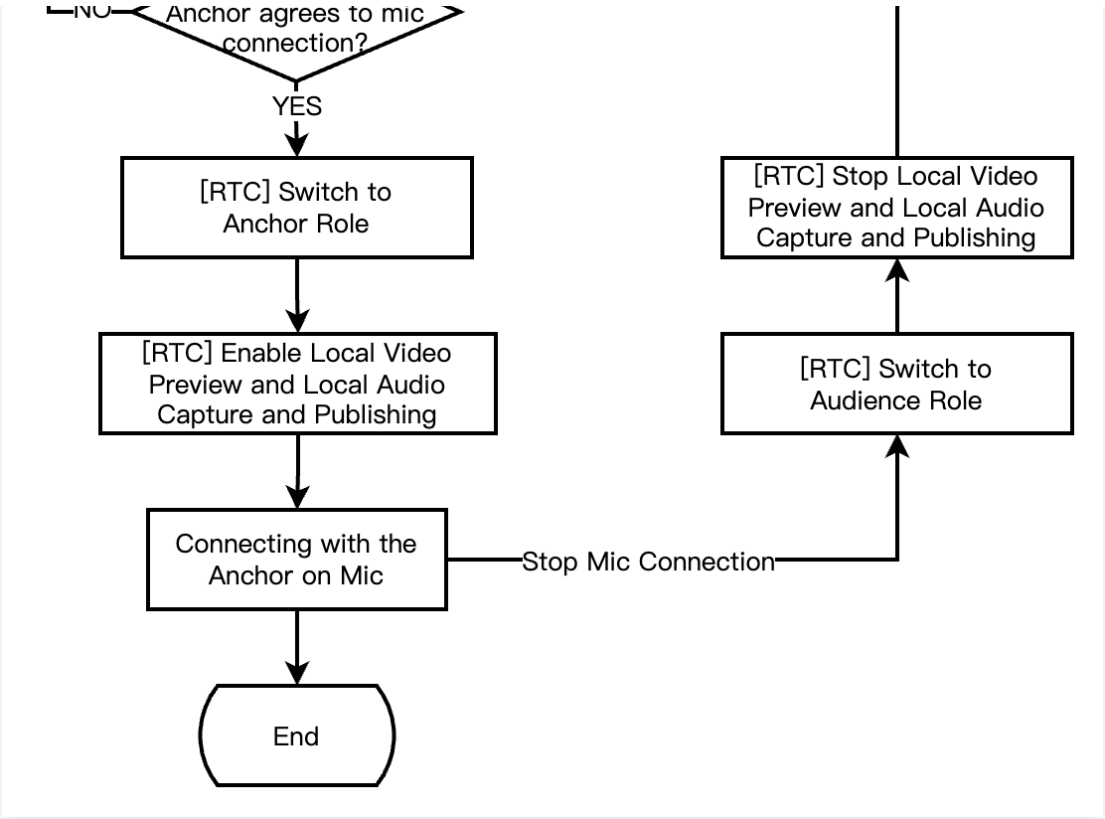
아래 그림으로 스트리머 A가 스트리머 B를 초청하여 크로스 룸 PK 연결을 진행하는 프로세스를 보여줍니다. 크로스 룸 PK 과정에서 두 방의 시청자 모두 두 방주인의 PK 연결 라이브 방송 화면을 볼 수 있습니다.



RTC 시청자 방 입장 후 마이크 연결합니다

아래 그림으로 RTC 실시간 인터랙션 라이브 방송실에서 시청자의 방 입장, 마이크 연결 신청, 연결 종료, 방 퇴장 프로세스를 보여줍니다.





판매 상품의 관리

아래 그림으로 라이브 커머스 시나리오에서 스트리머가 상품을 편집하고 상품 링크 올리며, 시청자가 상품을 보고 구매하는 프로세스를 보여줍니다.

접근 준비

단계1: 서비스 개통

전자상거래 라이브 방송 시나리오는 일반적으로 RTC Engine, 뷰티 AR, 플레이어 SDK와 같은 유료 PaaS 서비스를 활용하여 구축됩니다. 이 중 RTC Engine은 실시간 음성 및 영상 인터랙티브 기능을 제공하고 뷰티 AR은 뷰

티 효과 기능을 제공하며, 플레이어는 라이브 및 VOD 재생 기능을 제공합니다. 실제 업무 요구에 따라 위 서비스를 자유롭게 선택하여 개통할 수 있습니다.

RTC Engine 서비스의 개통

1. 먼저 RTC Engine 콘솔에 로그인하여 애플리케이션을 생성해야 합니다. 필요에 따라 RTC Engine 애플리케이션 버전을 업그레이드할 수 있으며, 예를 들어 프로페셔널 버전은 더 많은 부가 기능 서비스를 이용할 수 있습니다.

설명:

- 두 개의 애플리케이션을 생성하고 각각 테스트 환경과 프로덕션 환경에 적용하는 것을 권장하며, 1년 동안 각 텐센트 클라우드 계정(UIN)에 매월 10,000분의 무료 사용 시간이 제공됩니다.
- RTC Engine 월정액 요금제는 체험판(기본), 라이트 버전, 스탠다드 버전, 프로페셔널 버전으

> 로 구분되며, 각각 다른 부가 기능 서비스를 이용할 수 있습니다. 자세한 내용은 버전 기능 및 월정액 요금제 설명을 참조하세요.

2. 애플리케이션 생성이 완료되면, 앱 관리 – 앱 개요 섹션에서 해당 애플리케이션의 기본 정보를 확인할 수 있습니다. 향후 사용을 위해 **SDKAppID**와 **SDKSecretKey**를 안전하게 보관해야 하며, 키 유출로 인한 트래픽 도용을 방지해야 합니다.

뷰티 AR 서비스의 개통

1. 뷰티 AR 콘솔 > 모바일 라이선스에 로그인한 후 **테스트 라이선스 생성**을 클릭하세요(테스트판 라이선스는 무료로 14일간 유효하며, 1회 연장 가능하여 총 28일 사용할 수 있습니다). 모바일을 선택하고 실제 필요에 따라 앱 이름, 패키지 이름 및 번들 ID를 입력하세요. 사용해 보고 싶은 기능을 선택하세요.예를 들어 **모든 뷰티 기능**, **가상 배경**, **얼굴 인식**, **제스처 인식**, **선물 애니메이션 효과**등, 그런 다음 **확인**를 클릭하세요.

2. 활성화 후 현재 페이지에서 손님의 정보를 확인하고 상단의 통합 가이드를 참조하여 통합시킬 수 있습니다. 통합 가이드에서 License Key와 License URL 사용 방법을 확인할 수 있습니다.

플레이어 서비스의 개통

1. VOD 콘솔 또는 라이브 콘솔 > **라이선스 관리** > **모바일 라이선스**에 로그인한 후 **테스트 라이선스 생성**를 클릭하세요.

2. 실제 요구 사항에 따라 `App Name` , `Package Name` , `Bundle Id` 를 입력하고 **플레이어 고급판**을 선택한 후 **생성**를 클릭하세요.

3. 테스트판 라이선스가 성공적으로 생성되면 페이지에 생성된 라이선스 정보가 표시됩니다. **SDK 초기화 설정 시 License Key와 License URL 두 가지 매개변수를 전달해야 하므로 다음 정보를 안전하게 보관하**

세요.

> ⚠ 주의:
> 동일한 앱의 라이선스 URL과 Key는 고유하며, 테스트판 라이선스가 정식판으로 업그레이드된 후에도라이선스 URL과 Key는 변경되지 않습니다.

단계2: SDK 임포트하기

RTC Engine SDK, 뷰티 AR SDK, 플레이어 SDK는 모두 **mavenCentral** 라이브러리에 릴리스되었으며, gradle을 구성하여 자동으로 다운로드 및 업데이트할 수 있습니다.

1. dependencies에 적합한 버전의 SDK 종속성을 추가합니다.

```
dependencies {
    // 전체 기능판 SDK에 RTC Engine, 라이브 방송, 쇼트 비디오, 플레이어 등 다
양한 기능 포함됩니다
    implementation
'com.tencent.liteav:LiteAVSDK_Professional:latest.release'

    // 뷰티 AR SDK 예: S1-07 패키지는 다음과 같습니다
    implementation 'com.tencent.mediacloud:TencentEffect_S1-
07:latest.release'
}
```

> ⓘ 설명:

- 추천하는 자동 로드 방식 외에도 SDK를 다운로드하여 수동으로 임포트할 수도 있습니다. 자세한 내용은 RTC Engine SDK 수동 통합 및 뷰티 AR SDK 수동 통합을 참조하세요.
- 전자상거래 라이브 방송 시나리오의 구현은 일반적으로 RTC Engine, 플레이어 등 여러 기능의 조합이 필요하며, **개별 통합 시 발생할 수 있는 신호 충돌 문제를 방지하기 위해 전체 기능판 SDK 통합을 권장합니다.**

2. defaultConfig에서 앱이 사용하는 CPU 아키텍처를 지정합니다.

```
defaultConfig {
      ndk {
              abiFilters "armeabi-v7a", "arm64-v8a"
      }
}
```

설명:
LiteAVSDK 전체 기능판은 armeabi-v7a/arm64-v8a/x86/x86_64 아키텍처를 지원하며, 뷰티 AR SDK는 armeabi-v7a/arm64-v8a 아키텍처만 지원합니다.

3. **Sync Now**을 클릭하면 SDK가 자동으로 다운로드되어 프로젝트에 통합됩니다. 뷰티 AR 패키지에 모션 효과와 필터 기능이 포함된 경우 SDK 다운로드 페이지에서 해당 패키지를 다운로드하고, 무료 필터 소스(./assets/lut)와 스티커 모션(./MotionRes)을 압축 해제한 후 프로젝트의 다음 디렉터리에 배치해야 합니다.
   - 모션 효과: `../assets/MotionRes`
   - 필터 효과: `../assets/lut`

단계 3: 엔지니어링 구성

1. 권한의 구성. AndroidManifest.xml에서 앱 권한을 구성하며 전자상거래 라이브 방송 시나리오에서 RTC Engine 및 뷰티 AR SDK에는 다음 권한이 필요합니다:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission
android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
```

```
<uses-feature android:name="android.hardware.camera.autofocus" />
```

> ⚠ 주의:
> - `android:hardwareAccelerated="false"` 를 설정하지 마십시오. 하드웨어 가속을 끄면 상대방의 비디오 스트림이 렌더링되지 않습니다.
> - RTC Engine SDK에는 내장된 권한 요청 로직이 없으므로 해당 권한을 직접 선언해야 하며, 일부 권한(예: 저장소, 레코딩, 카메라 등)은 런타임에 동적으로 신청해야 합니다.
> - Android 프로젝트의 `targetSdkVersion` 이 31이거나 대상 기기가 Android 12및 그 이상 시스템 버전과 관련된 경우, 블루투스 기능을 정상적으로 사용하기 위해 코드에서 `android.permission.BLUETOOTH_CONNECT` 권한을 동적으로 신청해야 합니다. 자세한 내용은 블루투스 권한를 참조하십시오.

2. 혼합 스트림의 구성. SDK 내부에서 Java의 리플렉션 기능을 사용하므로 proguard-rules.pro 파일에서 SDK 관련 클래스를 난독화 제외 목록에 추가해야 합니다.

```
-keep class com.tencent.** { *; }
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class androidx.exifinterface.** { *;}
```

단계4: 인증 및 허가

RTC Engine 인증 자격 증명

UserSig는 텐센트 클라우드에서 설계한 보안 서명이고 악의적인 공격자가 손님의 클라우드 서비스 사용 권한을 도용하는 것을 방지하기 위한 것입니다. RTC Engine은 방 입장 시 이를 인증합니다.

- 디버깅 및 테스트 단계: 클라이언트 예시 코드와 콘솔에서 가져오기 두 가지 방법으로 UserSig를 계산 및 생성할 수 있으며, 디버깅 및 테스트 용도로만 사용됩니다.
- 정식 운영 단계: 클라이언트가 역공학으로 키가 유출되는 것을 방지하기 위해 보안 등급이 더 높은 서버 UserSig 계산 방식의 사용을 권장합니다.

구체적인 구현 프로세스는 다음과 같습니다.

1. 손님의 App은 SDK 초기화 함수를 호출하기 전에 먼저 서버에 UserSig를 요청해야 합니다.
2. 손님의 서버는 SDKAppID와 UserID에 따라 UserSig를 계산합니다.
3. 서버는 계산된 UserSig를 손님의 App에 반환합니다.
4. 손님의 App은 가져온 UserSig를 특정 API를 통해 SDK에 전달합니다.

5. SDK는 SDKAppID + UserID + UserSig를 텐센트 클라우드 서버에 제출하여 검증합니다.
6. 텐센트 클라우드는 UserSig를 검증하여 합법성을 확인합니다.
7. 검증이 통과되면 RTC Engine SDK에 실시간 음성 및 영상 서비스를 제공합니다.

⚠ 주의:

- 디버깅 단계의 로컬 UserSig 계산 방식은 온라인 환경에 적용하는 것을 권장하지 않으며, 역공학으로 인해 키가 유출될 수 있기 때문입니다.
- 여러 언어 버전(Java/GO/PHP/Node.js/Python/C#/C++)의 UserSig 서버 계산 소스 코드를 제공하며, 자세한 내용은 서버에서 UserSig 계산을 참조하세요.

뷰티 AR 인증 허가

뷰티 AR 사용 전에 텐센트 클라우드에서 라이선스 자격 증명을 검증해야 합니다. 라이선스 설정에는 License Key와 License URL이 필요하며, 예시 코드는 다음과 같습니다.

```
import com.tencent.xmagic.telicense.TELicenseCheck;

// 라이선스dml 다운로드 또는 업데이트만을 목적으로 하고 인증 결과에 관심이 없는
경우, 네 번째 매개변수에 null을 전달합니다.
TELicenseCheck.getInstance().setTELicense(context, URL, KEY, new
TELicenseCheck.TELicenseCheckListener() {
    @Override
    public void onLicenseCheckFinish(int errorCode, String msg) {
        // 주의: 이 콜백은 호출 스레드에서 실행되지 않을 수 있습니다
        if (errorCode == TELicenseCheck.ERROR_OK) {
            // 인증 성공
        } else {
            // 인증 실패
        }

    }
});
```

⚠ 주의:

- 관련 업무 모듈의 초기화 코드에서 인증 허가를 트리거하고 사용하기 전에 임시로 라이선스를 다운로드하는 것을 피하며, 인증 시에는 네트워크 권한이 있어야 합니다.
- 실제 적용된 패키지 이름은 라이선스 생성 시 바인딩된 Package Name과 완전히 일치해야 하며, 그렇지 않으면 라이선스 검증이 실패할 수 있습니다. 자세한 내용은 인증 오류 코드을 참조하세요.

플레이어 라이선스의 인증

라이브 방송의 재생 및 VOD 재생 기능은 플레이어 라이선스 권한을 구성한 후에야 성공적으로 재생할 수 있으며, 그렇지 않으면 재생에 실패합니다(검은 화면). 전역적으로 한 번만 설정하면 됩니다. 아직 라이선스를 획득하지 않으셨다면 무료 테스트 버전 라이선스 신청을 통해 정상적으로 재생할 수 있으며, 정식 버전 라이선스는 구매가 필요합니다. 라이선스 신청이 성공하면 **라이선스 URL**와 **라이선스 key** 두 개의 문자열을 받게 됩니다.
앱에서 SDK 관련 기능을 호출하기 전에 다음과 같이 구성해야 합니다(Application 클래스에서 구성하는 것이 좋습니다).

```
public class MApplication extends Application {
```

```
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // 가져온 licence url
        String licenceKey = ""; // 가져온 licence Key
        TXLiveBase.getInstance().setLicence(appContext, licenceURL,
licenceKey);
        TXLiveBase.setListener(new TXLiveBaseListener() {
            @Override
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ",
reason:" + reason);
                if (result != 0) {
                    // result가 0이 아니면 설정 실패를 의미하며, 재시도가
필요합니다.
                    TXLiveBase.getInstance().setLicence(appContext,
licenceURL, licenceKey);
                }
            }
        });
    }
}
```

라이선스 설정이 완료되면(네트워크 상황에 따라 조금 시간이 소요될 수 있음) 다음과 같은 방법으로 라이선스 정보를 확인할 수 있습니다.

```
TXLiveBase.getInstance().getLicenceInfo();
```

⚠ 주의:

- 실제 적용된 패키지 이름은 라이선스 생성 시 바인딩된 Package Name과 완전히 일치해야 하며, 그렇지 않으면 라이선스 검증이 실패할 수 있습니다.
- 라이선스는 강력한 온라인 검증 로직이고 앱이 처음 시작된 후 TXLiveBase#setLicence를 호출할 때 네트워크 사용 가능한 상태여야 합니다. 앱이 처음 시작될 때 아직 네트워크 권한이 부여되지 않았을 수 있으므로, 네트워크 권한이 부여된 후 다시 TXLiveBase#setLicence를 호출해야 합니다.
- TXLiveBase#setLicence 로드 결과의 모니터링: onLicenceLoaded 인터페이스가 실패 시 실제 상황에 따라 재시도 및 안내를 수행하고, 여러 번 실패한 경우 빈도 제한을 적용하고 제품 팝업 등의 안내를 통해 사용자가 네트워크 상태를 확인하도록 유도할 수 있습니다.
- TXLiveBase#setLicence는 여러 번 호출할 수 있으며, App 메인 화면 진입 시

TXLiveBase#setLicence를 호출하여 로드가 성공적으로 완료되도록 하는 것이 좋습니다.

- 멀티 프로세스 앱의 경우, 플레이어를 사용하는 각 프로세스가 시작될 때 TXLiveBase#setLicence가 호출되도록 해야 합니다. 예를 들어,Android에서 독립 프로세스로 비디오를 재생하는 앱의 경우, 백그라운드 재생 중 프로세스가 시스템에 의해 종료되고 다시 시작될 때도 TXLiveBase#setLicence를 호출해야 합니다.

단계5: SDK 초기화

RTC Engine SDK의 초기화

```
// RTC Engine SDK 인스턴스의 생성(싱글톤 모드)
TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(context);
// 이벤트 리스너의 설정
mTRTCCloud.addListener(trtcSdkListener);

// SDK의 다양한 이벤트 알림(예: 오류 코드, 경고 코드, 오디오/비디오 상태 매개
변수 등)
private TRTCCloudListener trtcSdkListener = new TRTCCloudListener()
{
    @Override
    public void onError(int errCode, String errMsg, Bundle
extraInfo) {
        Log.d(TAG, errCode + errMsg);
    }

    @Override
    public void onWarning(int warningCode, String warningMsg, Bundle
extraInfo) {
        Log.d(TAG, warningCode + warningMsg);
    }
};

// 이벤트 리스너의 제거
mTRTCCloud.removeListener(trtcSdkListener);
// RTC Engine SDK 인스턴스(싱글톤 모드)를 파기합니다.
TRTCCloud.destroySharedInstance();
```

설명:
SDK 이벤트 알림을 모니터링하고 일반적인 오류에 대한 로그 출력 및 처리를 권장합니다. 자세한 내용은 오류 코드 테이블을 참조하세요.

뷰티 AR SDK의 초기화

```
import com.tencent.xmagic.XmagicApi;

// 뷰티 AR SDK의 초기화
XmagicApi mXmagicApi = new XmagicApi(context,
XmagicResParser.getResPath(), new
XmagicApi.OnXmagicPropertyErrorListener());

// 개발 및 디버깅 시 로그 등급을 DEBUG로 설정하며 릴리스 패키지는 WARN으로 설
정하세요. 그렇지 않으면 성능에 영향을 미칠 수 있습니다.
mXmagicApi.setXmagicLogLevel(Log.WARN);

// 뷰티 AR SDK를 릴리스합니다. 이 방법은 GL 스레드에서 호출되어야 합니다.
mXmagicApi.onDestroy();
```

설명:
뷰티 AR SDK 초기화 전에 리소스 복사 등의 준비 작업이 필요하며, 자세한 단계는 뷰티 AR SDK 사용 절차을 참조하세요.

플레이어 SDK의 초기화

- VOD 재생 시나리오 SDK의 초기화.

```
// SDK 접근 환경을 설정합니다(글로벌 사용자를 대상으로 하는 경우 SDK 접근
환경을 글로벌 접근 환경으로 구성하세요)
TXLiveBase.setGlobalEnv("GDPR");

// Player 객체의 생성
```

```
TXVodPlayer mVodPlayer = new TXVodPlayer(mContext);

// 비디오 렌더링을 위한 View 컨트롤을 추가합니다
TXCloudVideoView mPlayerView = findViewById(R.id.video_view);
// Player 객체와 View 컨트롤의 연결
mVodPlayer.setPlayerView(mPlayerView);

// 플레이어 매개변수의 구성
TXVodPlayConfig config = new TXVodPlayConfig();
config.setEnableAccurateSeek(true);  // seek의 설정이 정확한지?,기본
값이 true입니다
config.setMaxCacheItems(5);          // 캐시 파일 개수를 5로 설정합니
다
config.setProgressInterval(200);     // 진행 콜백 간격을 설정합니다.
단위는 밀리초입니다
config.setMaxBufferSize(50);         // 최대 사전 로드 크기의 단위는
Mb입니다
mVodPlayer.setConfig(config);        // config를 mVodPlayer에 전달
합니다

// 플레이어 이벤트의 리스너
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle
param) {
        // 이벤트 알림
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
        // 상태 피드백
    }
});
```

- 라이브방송 재생 시나리오 SDK의 초기화.

```
// 비디오 렌더링을 위한 TXCloudVideoView를 미리서 추가해야 합니다
TXCloudVideoView mRenderView = findViewById(R.id.video_view);
// Player 객체의 생성
```

```
V2TXLivePlayer mLivePlayer = new V2TXLivePlayerImpl(mContext);
// Player 객체및 비디오 렌더링 view와 관련
mLivePlayer.setRenderView(mRenderView);

// 플레이어 이벤트의 리스너
mLivePlayer.setObserver(new V2TXLivePlayerObserver() {
    @Override
    public void onVideoLoading(V2TXLivePlayer player, Bundle
extraInfo) {
        // 비디오 로드 이벤트
    }

    @Override
    public void onVideoPlaying(V2TXLivePlayer player, boolean
firstPlay, Bundle extraInfo) {
        // 비디오 재생 이벤트
    }
});
```

접속 과정

API 시퀀스 다이어그램

단계1: 스트리머가 방에 입장하여 스트리밍을 시작합니다

RTC Engine SDK는 비디오 화면을 표시하는 컨트롤이고 `TXCloudVideoView` 유형만 전달할 수 있으므로, 먼저 레이아웃 파일에 뷰 렌더링 컨트롤을 정의해야 합니다.

```
<com.tencent.rtmp.ui.TXCloudVideoView
    android:id="@+id/live_cloud_view_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

> ⚠ 주의:
> `TextureView` 또는 `SurfaceView`를 뷰 렌더링 컨트롤로 지정하려면 고급 기능–뷰 렌더링 컨트롤를 참조하십시오.

1. 스트리머가 방에 들어가기 전에 로컬 비디오 미리보기 및 오디오 수집을 시작합니다.

```
// 스트리머의 로컬 화면 미리보기를 표시하기 위한 비디오 렌더링 컨트롤을 가져옵니다
TXCloudVideoView mTxcvvAnchorPreviewView = findViewById(R.id.live_cloud_view_main);

// 원격 사용자가 보는 화면 품질을 결정하는 비디오 인코딩 매개변수를 설정합니다.
TRTCCloudDef.TRTCVideoEncParam encParam = new TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_960_540;
encParam.videoFps = 15;
encParam.videoBitrate = 1300;
encParam.videoResolutionMode = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
mTRTCCloud.setVideoEncoderParam(encParam);

// boolean mIsFrontCamera는 전면/후면 카메라의 사용을 지정하여 비디오를 수집합니다.
```

```
mTRTCCloud.startLocalPreview(mIsFrontCamera,
mTxcvvAnchorPreviewView);

// 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
```

⚠ 주의:

- 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam을 직접 설정할 수 있으며, 각 등급에 대한 최적의 해상도 및 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.
- `enterRoom` 전에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집만 시작하며, `enterRoom` 을 호출한 후에야 스트리밍을 시작합니다.
- `enterRoom` 후에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집을 시작하고 자동으로 스트리밍을 시작합니다.

2. 스트리머가 로컬 화면의 렌더링 매개변수 및 인코더 출력 화면 모드(선택 사항)를 설정합니다.

```
TRTCCloudDef.TRTCRenderParams params = new
TRTCCloudDef.TRTCRenderParams();
params.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_AUTO;    // 화
면 미러 모드
params.fillMode = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;      // 화
면 채우기 모드
params.rotation = TRTCCloudDef.TRTC_VIDEO_ROTATION_0;            // 화
면 회전 각도
// 로컬 화면의 렌더링 매개변수를 설정합니다
mTRTCCloud.setLocalRenderParams(params);

// 인코더 출력 화면 미러 모드를 설정합니다
mTRTCCloud.setVideoEncoderMirror(boolean mirror);
// 비디오 인코더 출력 화면의 방향을 설정합니다
mTRTCCloud.setVideoEncoderRotation(int rotation);
```

⚠ 주의:

- 로컬 화면 렌더링 파라미터의 설정은 로컬 화면의 렌더링 효과에만 영향을 줍니다.
- 인코더 출력 모드의 설정은 방의 다른 사용자들이 보는 (및 클라우드 레코딩 파일) 화면 효과에 영

향을 줍니다.

3. 앵커가 정식으로 라이브 방송을 시작하고 방에 입장하여 스트리밍을 시작합니다.

```
public void enterRoomByAnchor(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = getUserSig(userId);
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = SDKAppID;
    // 스트리머 역할의 지정
    params.role = TRTCCloudDef.TRTCRoleAnchor;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

// 방 입장 결과 이벤트의 콜백
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result는 방에 임장하는 데 소요된 시간(밀리초)을 나타냅니다.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result는 방 입장 실패의 오류 코드를 나타냅니다.
        Log.d(TAG, "Enter room failed");
    }
}
```

⚠ 주의:

- RTC Engine 방 번호는 숫자 유형 `roomId` 과 문자열 유형 `strRoomId` 으로 구분되며, 두 유형의 방은 서로 통하지 않으므로 방 번호 유형을 통일기키는 것이 좋습니다.
- RTC Engine 사용자 역할은 스트리머와 청취자로 구분되며, 스트리머만 스트리밍 권한을 가집니다. 방 입장 시 사용자 역할을 지정해야 하며, 지정하지 않으면 기본적으로 스트리머 역할로 설정됩니다.
- 전자상거래 라이브 방송 시나리오에서는 방 입장 모드를 `TRTC_APP_SCENE_LIVE` 으로 선택하

는 것이 좋습니다.

단계2: 시청자가 방에 들어가서 스트리밍을 시작합니다

1. 시청자가 RTC Engine 방에 입장합니다.

```
public void enterRoomByAudience(String roomId, String userId) {
    TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = getUserSig(userId);
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = SDKAppID;
    // 시청자 역할의 지정
    params.role = TRTCCloudDef.TRTCRoleAudience;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}

// 방 입장 결과 이벤트의 콜백
@Override
public void onEnterRoom(long result) {
    if (result > 0) {
        // result는 방에 입장하는 데 소요된 시간(밀리초)을 나타냅니다.
        Log.d(TAG, "Enter room succeed");
    } else {
        // result는 방 입장 실패의 오류 코드를 나타냅니다.
        Log.d(TAG, "Enter room failed");
    }
}
```

2. 시청자가 스트리머의 오디오 및 비디오 스트림을 구독합니다.

```
@Override
public void onUserAudioAvailable(String userId, boolean available) {
    // 원격 사용자가 자신의 오디오를 게시/취소합니다
```

```
    // 자동 구독 모드에서 사용자가 아무런 작업을 하지 않아도 SDK가 원격 사용자의
오디오를 자동으로 재생합니다.
}

@Override
public void onUserVideoAvailable(String userId, boolean available) {
    // 원격 사용자가 메인 비디오 화면을 게시/취소합니다
    if (available) {
        // 원격 사용자의 비디오 스트림을 구독하고 비디오 렌더링 컨트롤을 바인딩
합니다
        mTRTCCloud.startRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, TXCloudVideoView view);
    } else {
        // 원격 사용자의 비디오 스트림 구독을 중지하고 렌더링 컨트롤을 릴리스합
니다
        mTRTCCloud.stopRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
    }
}
```

3. 시청자가 원격 화면의 렌더링 모드를 설정합니다(선택 사항).

```
TRTCCloudDef.TRTCRenderParams params = new
TRTCCloudDef.TRTCRenderParams();
params.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_AUTO;    // 화
면 미러 모드
params.fillMode = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;      // 화
면 채우기 모드
params.rotation = TRTCCloudDef.TRTC_VIDEO_ROTATION_0;            // 화
면 회전 각도
// 원격 화면의 렌더링 모드를 설정합니다
mTRTCCloud.setRemoteRenderParams(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, params)
```

단계3: 시청자가 마이크 켜고 인터랙션 합니다

1. 시청자가 스트리머 역할로 전환됩니다.

```
// 스트리머 역할로 전환됩니다.
```

```
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);

// 역할 전환 이벤트의 콜백
@Override
public void onSwitchRole(int errCode, String errMsg) {
    if (errCode == TXLiteAVCode.ERR_NULL) {
        // 역할 전환 성공
    }
}
```

2. 시청자가 로컬 오디오/비디오 수집 및 스트리밍을 시작합니다.

```
// 연결된 시청자의 로컬 화면 미리보기를 표시하기 위한 비디오 렌더링 컨트롤을 가져
옵니다.
TXCloudVideoView mTxcvvAudiencePreviewView =
findViewById(R.id.live_cloud_view_sub);

// 원격 사용자가 보는 화면 품질을 결정하는 비디오 인코딩 매개변수를 설정합니다.
TRTCCloudDef.TRTCVideoEncParam encParam = new
TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_480_270;
encParam.videoFps = 15;
encParam.videoBitrate = 550;
encParam.videoResolutionMode =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
mTRTCCloud.setVideoEncoderParam(encParam);

// boolean mIsFrontCamera는 전면/후면 카메라의 사용을 지정하여 비디오를 수집
합니다.
mTRTCCloud.startLocalPreview(mIsFrontCamera,
mTxcvvAudiencePreviewView);

// 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
```

> ⚠ 주의:

> 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam을 직접 설정할 수 있으며, 각 등급에 대한 최적의 해상도 및 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.

3. 시청자가 마이크를 끄고 스트리밍을 중지합니다.

```
// 시청자 역할로 전환됩니다
mTRTCCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);

// 역할 전환 이벤트의 콜백
@Override
public void onSwitchRole(int errCode, String errMsg) {
    if (errCode == TXLiteAVCode.ERR_NULL) {
        // 카메라 수집 및 스트리밍 중지합니다
        mTRTCCloud.stopLocalPreview();
        // 마이크 수집 및 스트리밍 중지합니다
        mTRTCCloud.stopLocalAudio();
    }
}
```

단계4: 방 나가기 및 해산하기

1. 방에서 나가기.

```
public void exitRoom() {
    mTRTCCloud.stopLocalAudio();
    mTRTCCloud.stopLocalPreview();
    mTRTCCloud.exitRoom();
}

// 방 나가기 이벤트의 콜백
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "exitRoom을 호출하여 방에서 나가기");
    } else if (reason == 1) {
        Log.d(TAG, "서버에 의해 현재 방에서 나가게 됩니다");
    } else if (reason == 2) {
        Log.d(TAG, "현재 방 전체가 해체됩니다");
    }
```

```
}
```

> ⚠ 주의:
> - SDK가 점유한 모든 리소스가 릴리스된 후, SDK는 `onExitRoom` 콜백을 통해 알려줍니다.
> - `enterRoom` 을 다시 호출하거나 다른 음성/영상 SDK로 전환하려면 `onExitRoom` 콜백이 발생한 후 관련 작업을 수행하십시오. 그렇지 않으면 카메라나 마이크 장치에서 점유됨 등 다양한 오류 문제가 발생할 수 있습니다.

2. 방 해산하기.
   - **서버 측에서 방 해산합니다**

     RTC Engine은 서버 측에서 숫자 유형 방을 해산하는 API `DismissRoom` 과 문자열 유형 방을 해산하는 API DismissRoomByStrRoomId를 제공합니다. 서버 측의 방 해산 인터페이스를 호출하여 방 내 모든 사용자를 방에서 나가게 시키고 방을 해산할 수 있습니다.
   - **클라이언트 측에서 방 해산합니다**

     클라이언트에는 방을 직접 해산하는 API가 없으며, 각 클라이언트가 exitRoom을 호출하여 방을 나가야 합니다. 방 내 모든 스트리머와 시청자가 방을 나가면 RTC Engine 방 라이프사이클 규칙에 따라 방이 자동으로 해산됩니다. 자세한 내용은 RTC Engine 방 나가기 을 참조하십시오.

> ⚠ 주의:
> 라이브 방송이 종료된 후에는 서버 측의 방 해산 API를 호출하여 방을 해산시키는 것이 좋습니다. 이는 일부 사용자가 예정대로 방을 나가지 않아 방이 유지되고 예상치 못한 비용이 발생하는 것을 방지하기 위함입니다.

고급 기능

상품 정보의 팝업

상품 정보 팝업 기능은 Chat 자체 정의 메시지을 통해 구현할 수 있으며, SEI 정보을 통해서도 구현할 수 있습니다. 아래에서는 이 두 가지 구현 방식을 각각 소개하겠습니다.

자체 정의 메시지

자체 정의 메시지는 텐센트 클라우드 Chat의 기능에 의존하며, 서비스를 사전에 개통하고 Chat SDK를 임포트해야 합니다. 자세한 안내는 음성 채팅방 접속 가이드-접속 준비 을 참조하십시오.

1. 자체 정의 메시지의 전송.
   - 방식 1: 스트리머가 클라이언트에서 상품 팝업과 관련된 그룹 커스텀 메시지를 전송합니다.

```
// 상품 팝업 메시지 본문의 구성
JSONObject jsonObject = new JSONObject();
```

```
try {
    jsonObject.put("cmd", "item_popup_msg");
    JSONObject msgJsonObject = new JSONObject();
    msgJsonObject.put("itemNumber", 1);      // 상품 번호
    msgJsonObject.put("itemPrice", 199.0);   // 상품 가격
    msgJsonObject.put("itemTitle", "xxx");   // 상품 제목
    msgJsonObject.put("itemUrl", "xxx");     // 상품 이미지 주소
    jsonObject.put("msg", msgJsonObject);
} catch (JSONException e) {
    e.printStackTrace();
}
String data = jsonObject.toString();

// 그룹 커스텀 메시지의 전송 (상품 팝업 메시지는 높은 우선순위로 설정하는 것
이 좋음)
V2TIMManager.getInstance().sendGroupCustomMessage(data.getBytes(),
mRoomId,
        V2TIMMessage.V2TIM_PRIORITY_HIGH, new
V2TIMValueCallback<V2TIMMessage>() {
            @Override
            public void onError(int i, String s) {
                // 상품 팝업 메시지의 전송이 실패됩니다
            }

            @Override
            public void onSuccess(V2TIMMessage v2TIMMessage) {
                // 상품 팝업 메시지의 전송이 성공됩니다
                // 로컬에서 상품 팝업 효과를 렌더링합니다
            }
        });
```

- 방식 2: 백엔드 운영은 서버 측에서 상품 팝업과 관련된 그룹 커스텀 메시지를 전송합니다.

  URL 요청 예시:

```
https://xxxxxx/v4/group_open_http_svc/send_group_msg?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&con
tenttype=json
```

  패킷 요청 예시:

```
{
    "GroupId": "@TGS#12DEVUDHQ",
    "Random": 2784275388,
    “MsgPriority": "High",  // 메시지의 우선순위에서 상품 팝업 메시지는
높은 우선순위로 설정하는 것을 권장합니다
    "MsgBody": [
        {
            "MsgType": "TIMCustomElem",
            "MsgContent": {
                // itemNumber: 상품 번호; itemPrice: 상품 가격;
itemTitel: 상품 제목; itemUrl: 상품 이미지 주소
                "Data": "{\"cmd\": \"item_popup_msg\", \"msg\":
{\"itemNumber\": 1, \"itemPrice\": 199.0, \"itemTitle\": \"xxx\",
\"itemUrl\": \"xxx\"}}"
            }
        }
    ]
}
```

2. 자체 정의 메시지의 수신.

   방 내 다른 사용자 클라이언트가 그룹 커스텀 메시지 콜백을 수신한 후 메시지 분파싱 및 상품 팝업 효과의 렌더링을 수행합니다.

```
// 그룹 커스텀 메시지의 수신
V2TIMManager.getInstance().addSimpleMsgListener(new
V2TIMSimpleMsgListener() {
    @Override
    public void onRecvGroupCustomMessage(String msgID, String
groupID, V2TIMGroupMemberInfo sender, byte[] customData) {
        String customStr = new String(customData);
        if (!customStr.isEmpty()) {
            try {
                JSONObject jsonObject = new JSONObject(customStr);
                String command = jsonObject.getString("cmd");
                JSONObject messageJsonObject =
jsonObject.getJSONObject("msg");
                if (command.equals("item_popup_msg")) {
```

```
                    int itemNumber =
messageJsonObject.getInt("itemNumber");       // 상품 번호
                    double itemPrice =
messageJsonObject.getDouble("itemPrice");  // 상품 가격
                    String itemTitle =
messageJsonObject.getString("itemTitle");  // 상품 제목
                    String itemUrl =
messageJsonObject.getString("itemUrl");       // 상품 이미지 주소
                    // 상품 번호, 상품 가격, 상품 제목, 상품 이미지 주소에 따
라 상품 팝업 효과를 렌더링합니다
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
});
```

SEI 정보

SEI 정보는 스트리머의 비디오 스트림에 삽입되어 전송되며, 상품 정보 팝업과 스트리머의 라이브 방송 화면을 정확하게 동기화할 수 있습니다.

1. SEI 정보 전송.

   스트리머는 RTC Engine 클라이언트에서 상품 팝업 관련 SEI 메시지를 전송합니다.

```
// 상품 팝업 메시지 본문의 구성
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("cmd", "item_popup_msg");
    JSONObject msgJsonObject = new JSONObject();
    msgJsonObject.put("itemNumber", 1);      // 상품 번호
    msgJsonObject.put("itemPrice", 199.0);  // 상품 가격
    msgJsonObject.put("itemTitle", "xxx");  // 상품 제목
    msgJsonObject.put("itemUrl", "xxx");     // 상품 이미지 주소
    jsonObject.put("msg", msgJsonObject);
} catch (JSONException e) {
    e.printStackTrace();
}
String data = jsonObject.toString();
```

```
// SEI 정보의 전송
mTRTCCloud.sendSEIMsg(data.getBytes(), 1);
```

2. SEI 정보의 수신.

   시청자는 RTC Engine 클라이언트에서 SEI 메시지를 수신한 후 메시지 파싱 및 상품 팝업 효과의 렌더링을 수행합니다.

```
mTRTCCloud.setListener(new TRTCCloudListener() {
    @Override
    public void onRecvSEIMsg(String userId, byte[] data) {
        String dataStr = new String(data);
        if (!dataStr.isEmpty()) {
            try {
                JSONObject jsonObject = new JSONObject(dataStr);
                String command = jsonObject.getString("cmd");
                JSONObject messageJsonObject =
jsonObject.getJSONObject("msg");
                if (command.equals("item_popup_msg")) {
                    int itemNumber =
messageJsonObject.getInt("itemNumber");       // 상품 번호
                    double itemPrice =
messageJsonObject.getDouble("itemPrice");  // 상품 가격
                    String itemTitle =
messageJsonObject.getString("itemTitle");  // 상품 제목
                    String itemUrl =
messageJsonObject.getString("itemUrl");       // 상품 이미지 주소
                    // 상품 번호, 상품 가격, 상품 제목, 상품 이미지 주소에 따
라 상품 팝업 효과를 렌더링합니다
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
});
```

상품 설명의 재방송

미리 레코딩된 상품 설명 동영상을 재생하여 상품 설명 재생 기능을 구현합니다.

먼저 플레이어 초기화을 진행한 후 레코딩된 영상을 재생하기 시작합니다. TXVodPlayer는 두 가지 재생 모드를 지원하며 필요에 따라 선택할 수 있습니다.

URL 방식으로

```
// URL 비디오 리소스의 재생
String url = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
mVodPlayer.startVodPlay(url);

// 로컬 비디오 리소스의 재생
String localFile = "/sdcard/video.mp4";
mVodPlayer.startVodPlay(localFile);
```

FileId 방식으로

```
// 아래의 새 인터페이스 사용을 권장합니다
// psign은 플레이어 서명이고 서명 소개 및 생성 방법은 다음 링크를 참조하세요.
https://cloud.tencent.com/document/product/266/42436
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, //
텐센트 클라우드 계정의 appId
    "4564972819220421305", // 비디오의 fileId
    "psignxxxxxxx"); // 플레이어 서명
mVodPlayer.startVodPlay(playInfoParam);

// 구 인터페이스이고 사용을 권장하지 않습니다
TXPlayerAuthBuilder authBuilder = new TXPlayerAuthBuilder();
authBuilder.setAppId(1252463788);
authBuilder.setFileId("4564972819220421305");
mVodPlayer.startVodPlay(authBuilder);
```

재생 관리: 진행률 조정, 재생 일시 정지, 재생 재개, 재생 종료.

```
// 진행률 조정(초)
```

```
mVodPlayer.seek(time);

// 재생 일시 정지
mVodPlayer.pause();

// 재생 재개
mVodPlayer.resume();

// 재생 종료(마지막 프레임 화면를 제거합니다)
mVodPlayer.stopPlay(true);
```

⚠ 주의:
재생 종료 시, 특히 다음 `startVodPlay` 전에 View 컨트롤을 반드시 파기해야 합니다. 그렇지 않으면 메모리 누수 및 화면 깜빡임 문제가 발생할 수 있습니다.
동시에, 재생 화면을 종료할 때 렌더링 View의 `onDestroy()` 함수를 반드시 호출해야 합니다. 그렇지 않으면 메모리 누출 및 "Receiver not registered" 경고가 발생할 수 있습니다.

```
@Override
public void onDestroy() {
    super.onDestroy();
    mVodPlayer.stopPlay(true); // true는 마지막 프레임을 제거하는 것을
의미합니다
    mPlayerView.onDestroy();
}
```

크로스 룸 PK 연결

1. 어느 한쪽이 크로스 룸 PK 연결을 발기합니다.

```
public void connectOtherRoom(String roomId, String userId) {
    try {
        JSONObject jsonObj = new JSONObject();
        // 숫자 방 번호는 roomId입니다
        jsonObj.put("strRoomId", roomId);
        jsonObj.put("userId", userId);
        mTRTCCloud.ConnectOtherRoom(jsonObj.toString());
    } catch (JSONException e) {
        e.printStackTrace();
```

```
    }
}

// 크로스 룸 연결 결과의 콜백
@Override
public void onConnectOtherRoom(String userId, int errCode, String
errMsg) {
    // 크로스 룸 연결을 하려는 다른 방 스트리머의 사용자 ID
    // 에러 코드, ERR_NULL은 요청 성공을 나타냅니다
    // 에러 정보
}
```

⚠ 주의:

- 크로스 룸 PK 연결의 로컬 사용자와 상대 사용자는 모두 스트리머 역할이어야 하며, 오디오/비디오 업스트림이 모두 있어야 합니다.
- `ConnectOtherRoom()` 을 여러 번 호출하여 여러 방의 스트리머와 크로스 룸 연결을 구현할 수 있습니다. 현재에 한 방은 최대 세 개의 다른 방의 스트리머와의 크로스 룸 연결이 가능하며, 한 방 내에서 최대 10명의 스트리머가 다른 방의 스크리머와 크로스 룸 연결을 할 수 있습니다.

2. 두 방의 모든 사용자는 다른 방의 스크리머로부터 오디오/비디오 스트림 사용 가능 콜백을 받게 됩니다.

```
@Override
public void onUserAudioAvailable(String userId, boolean available) {
    // 원격 사용자가 자신의 오디오를 게시/취소합니다
    // 자동 구독 모드에서는 사용자가 아무런 작업을 하지 않아도 SDK가 원격 사용자
의 오디오를 자동으로 재생합니다.
}

@Override
public void onUserVideoAvailable(String userId, boolean available) {
    // 원격 사용자가 메인 비디오 화면을 게시/취소합니다
    if (available) {
        // 원격 사용자의 비디오 스트림을 구독하고 비디오 렌더링 컨트롤을 바인딩
합니다
        mTRTCCloud.startRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, TXCloudVideoView view);
    } else {
```

```
        // 원격 사용자의 비디오 스트림 구독을 중지하고 렌더링 컨트롤을 릴리스합
니다
        mTRTCCloud.stopRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
    }
}
```

3. 어느 한쪽이 크로스 룸 PK 연결을 종료합니다.

```
// 크로스 룸 연결의 종료
mTRTCCloud.DisconnectOtherRoom();

// 크로스 룸 연결 종료 결과의 콜백
@Override
public void onDisConnectOtherRoom(int errCode, String errMsg) {
    super.onDisConnectOtherRoom(errCode, errMsg);
}
```

⚠ 주의:

- `DisconnectOtherRoom()` 을 호출하면 모든 다른 방의 스트리머와의 크로스 룸 PK 연결이 종료됩니다.
- 크로스 룸 PK 연결의 발신 측 또는 수신 측 중 어느 한쪽에서 `DisconnectOtherRoom()` 을 호출하여 크로스 룸 PK 연결을 종료할 수 있습니다.

제3자 뷰티 제품의 사용

RTC Engine은 제3자 뷰티 제품의 사용을 지원하며, 아래는 뷰티 AR을 예로 들어 제3자 뷰티 제품의 사용 절차를 보여줍니다.

1. 뷰티 AR SDK 통합 및 라이선스 권한의 신청에 대한 자세한 내용은 접속 준비 단계를 참조하세요.
2. 리소스 복사(있는 경우). 리소스 파일이 assets 디렉터리에 내장된 경우 사용 전에 App의 개인 디렉터리로 복사해야 합니다.

```
XmagicResParser.setResPath(new File(getFilesDir(),
"xmagic").getAbsolutePath());
//loading

//개인 디렉터리로 리소스 파일을 복사하며, 한 번만 수행하면 됩니다.
XmagicResParser.copyRes(getApplicationContext());
```

리소스 파일이 네트워크 동적 다운로드에서 가져온 경우 다운로드가 완료된 후 리소스 파일 경로를 설정해야 합니다.

```
XmagicResParser.setResPath(다운로드한 리소스 파일의 로컬 경로);
```

3. 제3자 뷰티 기능의 비디오 데이터 콜백을 설정하여 부티 AR SDK가 처리한 프레임별 데이터 결과를 RTC Engine SDK 내부로 전달해서 렌더링 처리를 수행합니다.

```
mTRTCCloud.setLocalVideoProcessListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_
FORMAT_Texture_2D, TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new
TRTCCloudListener.TRTCVideoFrameListener() {
    @Override
    public void onGLContextCreated() {
        // SDK 내부 OpenGL 환경이 생성되었으며, 이때 제3자 뷰티의 초기화 작업
을 진행할 수 있습니다.
        if (mXmagicApi == null) {
            XmagicApi mXmagicApi = new XmagicApi(context,
XmagicResParser.getResPath(), new
XmagicApi.OnXmagicPropertyErrorListener());
        } else {
            mXmagicApi.onResume();
        }
    }

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame
srcFrame, TRTCCloudDef.TRTCVideoFrame dstFrame) {
        // 제3자 뷰티 컴포넌트와 연동하기 위한 비디오 처리의 콜백
        if (mXmagicApi != null) {
            dstFrame.texture.textureId =
mXmagicApi.process(srcFrame.texture.textureId, srcFrame.width,
srcFrame.height);
        }
        return 0;
    }

    @Override
    public void onGLContextDestory() {
```

```
        // SDK 내부 OpenGL 환경이 파기되었으며, 이때 제3자 뷰티 리소스 파기
작업을 진행할 수 있습니다.
        mXmagicApi.onDestroy();
    }
});
```

> ⚠ 주의:
> 단계1 및 단계2는 제3자 뷰티 제품의 구현 방식에 따라 다르며, **단계3**은 RTC Engine이 제3자 뷰티를 통합하는 **공통적이고 중요한 단계**입니다.

듀얼 인코딩 모드

듀얼 인코딩 모드를 활성화하면 현재 사용자의 인코더가 [고화질 대형 화면]과 [저화질 소형 화면] 두 가지 비디오 스트림을 동시에 출력합니다(단, 오디오 스트림은 하나만 존재함). 이렇게 하면 방의 다른 사용자가 자신의 네트워크 상황이나 화면 크기에 따라 [고화질 대형 화면] 또는 [저화질 소형 화면]을 선택하여 구독할 수 있습니다.

1. 대형및 소형 화면 듀얼 인코딩 모드를 활성화합니다.

```
public void enableDualStreamMode(boolean enable) {
    // 소형 스트림의 비디오 인코딩 매개변수(자체 정의 가능)
    TRTCCloudDef.TRTCVideoEncParam smallVideoEncParam = new
TRTCCloudDef.TRTCVideoEncParam();
    smallVideoEncParam.videoResolution =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_480_270;
    smallVideoEncParam.videoFps = 15;
    smallVideoEncParam.videoBitrate = 550;
    smallVideoEncParam.videoResolutionMode =
TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
    mTRTCCloud.enableEncSmallVideoStream(enable, smallVideoEncParam);
}
```

> ⚠ 주의:
> 듀얼 인코딩을 활성화하면 더 많은 CPU 및 네트워크 대역폭이 소모되므로 Mac, Windows 또는 고성능 Pad에서는 활성화를 고려할 수 있지만 모바일에서는 활성화하지 않는 것이 좋습니다.

2. 원격 사용자 비디오 스트림의 유형을 선택합니다.

```
// 원격 사용자 비디오 스트림을 구독할 때 선택 가능한 비디오 스트림의 유형
mTRTCCloud.startRemoteView(userId, streamType, videoView);
```

```
// 원격 사용자의 화면 크기를 언제든지 전환할 수 있습니다
mTRTCCloud.setRemoteVideoStreamType(userId, streamType);
```

> ⚠ 주의:
> 듀얼 스트림 인코딩 활성화 후, `streamType` 비디오 스트림 유형을 `TRTC_VIDEO_STREAM_TYPE_SMALL` 으로 지정하여 저화질 소형 화면을 가져올 수 있습니다.

뷰 렌더링 컨트롤

RTC Engine에는 비디오 화면을 관리하는 인터페이스가 많으며, 이러한 인터페이스는 모두 비디오 렌더링 컨트롤을 지정해야 합니다. Android 플랫폼에서는 `TXCloudVideoView` 를 비디오 렌더링 컨트롤로 사용하며, `SurfaceView` 와 `TextureView` 두 가지 렌더링 방식을 지원합니다. 아래에서는 렌더링 컨트롤 유형을 지정하는 방법과 비디오 렌더링 컨트롤을 업데이트하는 방법을 소개합니다.

1. 특정 방식을 강제로 사용하거나 로컬 비디오 렌더링 컨트롤을 `TXCloudVideoView` 으로 변환하려면 다음과 같이 코딩할 수 있습니다.

```
// TextureView를 강제로 사용합니다
TextureView textureView = findViewById(R.id.texture_view);
TXCloudVideoView cloudVideoView = new TXCloudVideoView(context);
cloudVideoView.addVideoView(textureView);

// SurfaceView를 강제로 사용합니다
SurfaceView surfaceView = findViewById(R.id.surface_view);
TXCloudVideoView cloudVideoView = new TXCloudVideoView(surfaceView);
```

2. 업무에 표시 영역 전환과 관련된 인터랙티브 시나리오가 포함된 경우 RTC Engine SDK를 사용하여 로컬 미리보기 화면을 업데이트하고 원격 사용자 비디오 렌더링 컨트롤 기능을 구현할 수 있습니다.

```
// 로컬 미리보기 화면 렌더링 컨트롤의 업데이트
mTRTCCloud.updateLocalView(videoView);

// 원격 사용자 비디오 렌더링 컨트롤의 업데이트
mTRTCCloud.updateRemoteView(userId, streamType, videoView);
```

> ⚠ 주의:
> 매개변수 `videoView` 를 대상 비디오 렌더링 컨트롤로 전달하고, `streamType` 은 `TRTC_VIDEO_STREAM_TYPE_BIG` 및 `TRTC_VIDEO_STREAM_TYPE_SUB` 만 지원됩니다.

이상 처리

이상및 오류 처리

RTC Engine SDK에서 복구할 수 없는 오류가 발생하면 `onError` 콜백에서 나오며, 자세한 내용은 오류 코드표 을 참조하십시오.

1. UserSig 관련. UserSig 검증 실패로 인해 방 입장에 실패할 수 있으며 UserSig 도구 를 사용하여 검증할 수 있습니다.

열거형	값	설명
ERR_TRTC_INVALID_USER_SIG	-3320	방 입장 매개변수 userSig가 올바르지 않습니다. `TRTCParams.userSig` 이 비어 있는지 확인하세요.
ERR_TRTC_USER_SIG_CHECK_FAILED	-100018	UserSig 검증에 실패했습니다. 매개변수 `TRTCParams.userSig` 이 올바르게 입력되었거나 만료되지 않았는지 확인하세요.

2. 입장및 퇴장 관련. 입장 실패 시 먼저 입장 매개변수가 올바른지 확인하고, 입장및 퇴장 인터페이스는 반드시 쌍으로 호출해야 합니다. 입장에 실패한 경우에도 퇴장 인터페이스를 호출해야 합니다.

열거형	값	설명
ERR_TRTC_CONNECT_SERVER_TIMEOUT	-3308	입장 요청이 시간 초과했습니다, 네트워크 연결이 끊겼는지 또는 VPN이 켜져 있는지 확인하세요. 4G로 전환하여 테스트할 수도 있습니다.
ERR_TRTC_INVALID_SDK_APPID	-3317	입장 매개변수 sdkAppId가 잘못되었습니다. `TRTCParams.sdkAppId` 이 비어 있는지 확인하세요.
ERR_TRTC_INVALID_ROOM_ID	-3318	입장 매개변수 roomId가 잘못되었습니다. `TRTCParams.roomId` 또는 `TRTCParams.strRoomId` 이 비어 있는지 확인하세요. roomId와 strRoomId는 혼용할 수 없습니다.
ERR_TRTC_INVALID_USER_ID	-3319	입장 매개변수 userId가 올바르지 않습니다. `TRTCParams.userId` 이 비어 있는지 확인하세요.
ERR_TRTC_ENTER_ROOM_REFUSED	-3340	입장 요청이 거부되었습니다. `enterRoom` 을 연속적으로 호출하여 동일한 ID의 방에 입장했는지 확인하세요.

3. 장치 관련. 장치 관련 오류를 감지할 수 있으며, 관련 오류 발생 시 UI에서 사용자에게 알립니다.

열거형	값	설명
ERR_CAMERA_START_FAIL	-1301	Windows 또는 Mac 장치에서 카메라 구성 프로그램(드라이버)이 비정상일 경우, 카메라 열 수가 없습니다. 장치를 비활성화한 후 다시 활성화하거나, 기기를 재시작하거나, 구성 프로그램을 업데이트하세요.
ERR_MIC_START_FAIL	-1302	Windows 또는 Mac 장치에서 마이크 구성 프로그램(드라이버)이 비정상일 경우, 마이크 열 수가 없습니다. 장치를 비활성화한 후 다시 활성화하거나, 기기를 재시작하거나, 구성 프로그램을 업데이트하세요.
ERR_CAMERA_NOT_AUTHORIZED	-1314	카메라 장치에 권한이 없습니다. 일반적으로 모바일 장치에서 발생하며, 사용자가 권한을 거부했을 수 있습니다.
ERR_MIC_NOT_AUTHORIZED	-1317	마이크 장치에 권한이 없습니다. 일반적으로 모바일 장치에서 발생하며, 사용자가 권한을 거부했을 수 있습니다.
ERR_CAMERA_OCCUPY	-1316	카메라가 사용 중입니다. 다른 카메라를 열어 볼 수 있습니다.
ERR_MIC_OCCUPY	-1319	마이크가 사용 중입니다. 예를 들어 모바일 장치에서 통화 중일 때 마이크를 열 수가 없습니다.

원격 미러 모드의 무효 문제

RTC Engine 설정 화면 미러는 로컬 미리보기 미러 `setLocalRenderParams` 과 비디오 인코더 미러 `setVideoEncoderMirror` 으로 나뉘며, 각각 로컬 미리보기 화면의 미러 효과와 비디오 인코딩 출력 화면의 미러 효과(원격 시청자 및 클라우드 레코딩의 미러링 모드)에 영향을 줍니다. 로컬 미리보기의 미러 효과가 원격 시청자 측에도 동시에 적용되도록 하려면 다음과 같이 코딩하십시오.

```
// 로컬 화면의 렌더링 매개변수를 설정합니다.
TRTCCloudDef.TRTCRenderParams params = new
TRTCCloudDef.TRTCRenderParams();
params.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_ENABLE; // 화면
미러 모드
params.fillMode = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL;      // 화면
채우기 모드
params.rotation = TRTCCloudDef.TRTC_VIDEO_ROTATION_0;            // 화면
회전 각도
mTRTCCloud.setLocalRenderParams(params);
```

```
// 인코더 출력 화면 미러 모드를 설정합니다
mTRTCCloud.setVideoEncoderMirror(true);
```

카메라 줌/초점/전환 문제

전자상거래 라이브 방송 시나리오에서 스트리머는 카메라에 대한 자체 정의 조정이 필요할 수 있으며, RTC Engine SDK 장치 관리 클래스에는 이러한 요구 사항을 해결하기 위한 관련 인터페이스가 있습니다.

1. 카메라 줌 배율의 조회 및 설정.

```
// 카메라 최대 줌 배율 가져오기(모바일 전용)
float zoomRatio =
mTRTCCloud.getDeviceManager().getCameraZoomMaxRatio();
// 카메라 줌 배율의 설정(모바일 전용)
// 범위는 1-5이며, 1은 가장 먼 시야(일반 렌즈)이고 5는 가장 가까운 시야(확대 렌즈)를 나타냅니다. 최대값은 5를 권장하며, 5를 초과하면 비디오 데이터가 흐릿해질 수 있습니다
mTRTCCloud.getDeviceManager().setCameraZoomRatio(zoomRatio);
```

2. 카메라 초점 기능 및 위치의 설정.

```
// 카메라 자동 초점 기능 켜기 또는 끄기(모바일 전용)
mTRTCCloud.getDeviceManager().enableCameraAutoFocus(false);
// 카메라 초점 위치의 설정(모바일 전용)
// 해당 인터페이스를 사용하려면 먼저 enableCameraAutoFocus를 통해 자동 초점 기능을 꺼야 합니다
mTRTCCloud.getDeviceManager().setCameraFocusPosition(int x, int y);
```

3. 전면 또는 후면 카메라의 판단 및 전환.

```
// 현재 전면 카메라인지 판단합니다(모바일 전용)
boolean isFrontCamera =
mTRTCCloud.getDeviceManager().isFrontCamera();
// 전면 또는 후면 카메라의 전환(모바일 전용)
// true 전달: 전면으로 전환; false 전달: 후면으로 전환
mTRTCCloud.getDeviceManager().switchCamera(!isFrontCamera);
```

iOS

최종 업데이트 날짜:: 2025-10-28 11:32:37

업무 프로세스

본 섹션은 전자상거래 라이브 방송 업무 시나리오에서 일반적으로 사용되는 업무 프로세스를 종합하여 전체 시나리오 구현 프로세스를 더 잘 이해할 수 있도록 도와줍니다.

스트리머의 방송 시작 및 종료

아래 그림으로 스트리머가(방주인)의 로컬 미리보기, 방 생성, 방 입장 및 방송 시작, 방 퇴장 및 방송 종료 프로세스를 보여줍니다.

스트리머 간 크로스 룸 PK 연결

아래 그림으로 스트리머 A가 스트리머 B를 초청하여 크로스 룸 PK 연결을 진행하는 프로세스를 보여줍니다. 크로스 룸 PK 과정에서 두 방의 시청자 모두 두 방주인의 PK 연결 라이브 방송 화면을 볼 수 있습니다.

RTC 시청자 방 입장 후 마이크 연결합니다

아래 그림으로 RTC 실시간 인터랙션 라이브 방송실에서 시청자의 방 입장, 마이크 연결 신청, 연결 종료, 방 퇴장 프로세스를 보여줍니다.

판매 상품의 관리

아래 그림으로 라이브 커머스 시나리오에서 스트리머가 상품을 편집하고 상품 링크 올리며, 시청자가 상품을 보고 구매하는 프로세스를 보여줍니다.

접수 준비

단계1: 서비스 개통

전자상거래 라이브 방송 시나리오는 일반적으로 RTC Engine, 뷰티 AR, 플레이어 SDK와 같은 유료 PaaS 서비스를 활용하여 구축됩니다. 이 중 RTC Engine은 실시간 음성 및 영상 인터랙티브 기능을 제공하고 뷰티 AR은 뷰

티 효과 기능을 제공하며, 플레이어는 라이브 및 VOD 재생 기능을 제공합니다. 실제 업무 요구에 따라 위 서비스를 자유롭게 선택하여 개통할 수 있습니다.

RTC Engine 서비스의 개통

1. 먼저 RTC Engine 콘솔에 로그인하여 애플리케이션을 생성해야 합니다. 필요에 따라 RTC Engine 애플리케이션 버전을 업그레이드할 수 있으며, 예를 들어 프로페셔널 버전은 더 많은 부가 기능 서비스를 사용할 수 있습니다.

설명:

- 두 개의 애플리케이션을 생성하고 각각 테스트 환경과 프로덕션 환경에 적용하는 것을 권장하며, 1년 동안 각 텐센트 클라우드 계정(UIN)에 매월 10,000분의 무료 사용 시간이 제공됩니다.
- RTC Engine 월정액 요금제는 체험판(기본), 라이트 버전, 스탠다드 버전, 프로페셔널 버전으

> 로 구분되며, 각각 다른 부가 기능 서비스를 이용할 수 있습니다. 자세한 내용은 버전 기능 및 월정액 요금제 설명을 참조하세요.

2. 애플리케이션 생성이 완료되면, 앱 관리 – 앱 개요 섹션에서 해당 애플리케이션의 기본 정보를 확인할 수 있습니다. 향후 사용을 위해 **SDKAppID**와 **SDKSecretKey**를 안전하게 보관해야 하며, 키 유출로 인한 트래픽 도용을 방지해야 합니다.

뷰티 AR 서비스의 개통

1. 뷰티 AR 콘솔 > 모바일 라이선스에 로그인한 후 **테스트 라이선스 생성**을 클릭하세요(테스트판 라이선스는 무료로 14일간 유효하며, 1회 갱신 가능하여 총 28일 사용할 수 있습니다). 모바일을 선택하고 실제 필요에 따라 앱 이름, 패키지 이름, 번들 ID를 입력하세요. 사용해 보고 싶은 기능을 선택하세요.예를 들어 **모든 뷰티 기능**, **가상 배경**, **얼굴 인식**, **제스처 인식**, **선물 애니메이션 효과**등, 그런 다음 **확인**을 클릭하세요.

2. 활성화 후 현재 페이지에서 손님의 정보를 확인하고 상단의 통합 가이드를 참조하여 통합시킬 수 있습니다. 통합 가이드에서 License Key와 License URL 사용 방법을 확인할 수 있습니다.

플레이어 서비스의 개통

1. VOD 콘솔 또는 라이브 콘솔 > **라이선스 관리** > **모바일 라이선스**에 로그인한 후 **테스트 라이선스 생성**를 클릭하세요.

2. 실제 요구 사항에 따라 `App Name` , `Package Name` , `Bundle Id` 를 입력하고 **플레이어 고급판**을 선택한 후 **생성**를 클릭하세요.

3. 테스트판 라이선스가 성공적으로 생성되면 페이지에 생성된 라이선스 정보가 표시됩니다. **SDK 초기화 설정 시 License Key와 License URL 두 가지 매개변수를 전달해야 하므로 다음 정보를 안전하게 보관하**

세요.

> ⚠ 주의:
> 동일한 앱의 라이선스 URL과 Key는 고유하며, 테스트판 라이선스가 정식판으로 업그레이드된 후에도라이선스 URL과 Key는 변경되지 않습니다.

단계2: SDK 임포트하기

RTC Engine SDK와 뷰티 AR SDK는 **CocoaPods** 라이브러리에 릴리스되었으며 CocoaPods를 통해 통합할 수 있습니다.

1. CocoaPods를 설치합니다. 터미널 창구에 다음 명령을 입력하세요(Mac에 Ruby 환경이 미리 설치되어 있어야 함):

```
sudo gem install cocoapods
```

2. Podfile 파일을 생성합니다. 프로젝트 경로로 이동한 후 다음 명령줄을 입력하면 프로젝트 경로에 Podfile 파일이 생성됩니다.

```
pod init
```

3. Podfile 파일을 편집합니다. 프로젝트 요구 사항에 따라 적절한 버전을 선택하고 Podfile 파일을 편집하세요.

```
platform :ios, '8.0'
    target 'App' do
```

```
    # 전체 기능 버전 SDK
    # Real-Time Communication Engine (RTC Engine), 라이브방송 플레이어
(TXLivePlayer), RTMP 푸시 스트리밍(TXLivePusher), VOD 플레이어
(TXVodPlayer), 짧은 비디오 레코딩 및 편집(UGSV) 등 다양한 기능을 포함합니다.
    pod 'TXLiteAVSDK_Professional', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_Professio
nal.podspec'

    #텐센트 이펙트 SDK 예: S1-07 패키지는 다음과 같습니다
    pod 'TencentEffect_S1-07'

end
```

> ⚠ 주의:
> 전자상거래 라이브 방송 시나리오의 구현은 일반적으로 RTC Engine, 플레이어 등 여러 기능의 조합이 필요하며, **개별 통합 시 발생할 수 있는 신호 충돌 문제를 방지하기 위해 전체 기능판 SDK 통합을 권장합니다**.

4. SDK를 업데이트하고 설치합니다.
   터미널 창구에서 다음 명령을 입력하여 로컬 라이브러리 파일을 업데이트하고 SDK를 설치합니다.

```
pod install
```

   또는 다음 명령을 사용하여 로컬 라이브러리 버전을 업데이트합니다.

```
pod update
```

   pod 명령 실행 후 SDK가 통합된 .xcworkspace 확장자의 프로젝트 파일이 생성되며, 더블 클릭하여 열 수 있습니다.

> ⓘ **설명:**
> - pod 검색이 실패할 경우, pod의 로컬 repo 캐시를 업데이트해 볼 것을 권장하며, 업데이트 명령은 다음과 같습니다.

```
pod setup
pod repo update
rm ~/Library/Caches/CocoaPods/search_index.json
```

- 추천하는 자동 로드 방식 외에도, SDK를 다운로드하여 수동으로 임포트할 수도 있으며, 자세한 내용은 RTC Engine SDK 수동 통합 및 뷰티 AR SDK 수동 통합를 참조하세요.

5. 뷰티 리소스를 실제 프로젝트에 추가합니다.
   5.1 해당 패키지의 SDK 및 뷰티 리소스를 다운로드하고 압축을 해제한 후, resources/motionRes 폴더의 bundle 리소스를 실제 프로젝트에 추가합니다.
   5.2 Build Settings의 Other Linker Flags에 `-ObjC` 을 추가합니다.
6. Bundle Identifier를 신청한 테스트 권한과 일치하도록 수정합니다.

단계 3: 엔지니어링 구성

1. 권한의 설정.

   전자상거래 라이브 방송 시나리오에서 LiteAVSDK 및 텐센트 이펙트 SDK는 다음 권한이 필요합니다. 앱의 Info.plist에 다음 두 항목을 추가하세요. 각각 마이크와 카메라에 대한 시스템 권한 요청 다이얼로그 박스의 메시지에 해당합니다.
   - **Privacy – Microphone Usage Description**, 마이크 사용 목적 안내문을 입력하세요.
   - **Privacy – Camera Usage Description**, 카메라 사용 목적 안내문을 입력하세요.

2. 앱이 백그라운드에서도 관련 기능을 계속 실행하려면 XCode에서 현재 프로젝트를 선택하고 Capabilities에 있는 Background Modes 설정을 ON으로 설정한 후 **Audio, AirPlay and Picture in Picture**을 클릭하세요. 아래 그림과 같습니다.

단계4: 인증 및 허가

RTC Engine 인증 자격 증명

UserSig는 텐센트 클라우드가 설계한 보안 서명이고 악의적인 공격자가 손님의 클라우드 서비스 사용 권한을 도용하는 것을 방지하기 위한 것입니다. RTC Engine은 입장 시 이 인증 자격 증명을 검증합니다.

- 디버깅 및 테스트 단계: 클라이언트 예시 코드와 콘솔에서 가져오기 두 가지 방법으로 UserSig를 계산 및 생성할 수 있으며, 디버깅 및 테스트 용도로만 사용됩니다.
- 정식 운영 단계: 클라이언트가 역공학으로 키가 유출되는 것을 방지하기 위해 보안 등급이 더 높은 서버 UserSig 계산 방식의 사용을 권장합니다.

구체적인 구현 프로세스는 다음과 같습니다.

1. 손님의 App은 SDK의 초기화 함수를 호출하기 전에 먼저 서버에 UserSig를 요청해야 합니다.

2. 손님의 서버는 SDKAppID와 UserID에 따라 UserSig를 계산합니다.
3. 서버는 계산된 UserSig를 손님의 App에 반환합니다.
4. 손님의 App은 가져온 UserSig를 특정 API를 통해 SDK에 전달합니다.
5. SDK는 SDKAppID + UserID + UserSig를 텐센트 클라우드 서버에 제출하여 검증합니다.
6. 텐센트 클라우드는 UserSig를 검증하여 합법성을 확인합니다.
7. 검증이 통과되면 RTC Engine SDK에 실시간 음성 및 영상 서비스를 제공합니다.

⚠ 주의:

- 디버깅 단계의 로컬 UserSig 계산 방식은 온라인 환경에 적용하는 것을 권장하지 않으며, 역공학으로 인해 키가 유출될 수 있기 때문입니다.
- 여러 언어 버전(Java/Go/PHP/Node.js/Python/C#/C++)의 UserSig 서버 계산 소스 코드를 제공하며, 자세한 내용은 서버에서 UserSig 계산을 참조하세요.

뷰티 AR 인증 허가

뷰티 AR 사용 전에 텐센트 클라우드에서 라이선스 자격 증명을 검증해야 합니다. 라이선스 설정에는 License Key와 License URL이 필요하며, 예시 코드는 다음과 같습니다.

```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey
completion:^(NSInteger authresult, NSString * _Nonnull errorMsg) {
 if (authresult == TELicenseCheckOk) {
     NSLog(@"인증 성공");
 } else {
     NSLog(@"인증 실패");
 }
}];
```

⚠ 주의:

- 관련 업무 모듈의 초기화 코드에서 인증 허가를 트리거하고 사용하기 전에 임시로 라이선스를 다운로드하는 것을 피하며, 인증 시에는 네트워크 권한이 있어야 합니다.
- 실제 애플리케이션의 Bundle Id는 라이선스 생성 시 바인딩된 Bundle ID와 완전히 일치해야 하며, 그렇지 않으면 라이선스 검증이 실패합니다. 자세한 내용은 인증 오류 코드을 참조하세요.

플레이어 라이선스의 인증

라이브 방송의 재생 및 VOD 재생 기능은 플레이어 라이선스 권한을 구성한 후에야 성공적으로 재생할 수 있으며, 그렇지 않으면 재생에 실패합니다(검은 화면). 전역적으로 한 번만 설정하면 됩니다. 아직 라이선스를 획득하지 않으셨다면 무료 테스트 버전 라이선스 신청을 통해 정상적으로 재생할 수 있으며, 정식 버전 라이선스는 구매가 필요합니다. 라이선스 신청이 성공하면 **라이선스 URL**와 **라이선스 key** 두 개의 문자열을 받게 됩니다.
앱에서 SDK 관련 기능을 호출하기 전에(권장: `- [AppDelegate application:didFinishLaunchingWithOptions:]` 에서) 다음과 같이 설정하세요.

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    NSString * const licenceURL = @"<가져온 licenseUrl>";
    NSString * const licenceKey = @"<가져온 key>";

    // TXLiveBase는 "TXLiveBase.h" 헤더 파일에 위치합니다
    [TXLiveBase setLicence:licenceURL key:licenceKey];
```

```
    [TXLiveBase setObserver:self];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
    return YES;
}

#pragma mark - TXLiveBaseDelegate
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
    // result가 0이 아니면 설정 실패를 의미하며, 재시도가 필요합니다.
    if (result != 0) {
        [TXLiveBase setLicence:licenceURL key:licenceKey];
    }
}
@end
```

라이선스 설정이 완료되면(네트워크 상황에 따라 조금 시간이 소요될 수 있음) 다음과 같은 방법으로 라이선스 정보를 확인할 수 있습니다.

```
NSLog(@"%@", [TXLiveBase getLicenceInfo]);
```

⚠ 주의:

- 실제 애플리케이션의 Bundle ID는 라이선스 생성 시 바인딩된 Bundle ID와 완전히 일치해야 하며, 그렇지 않으면 라이선스 검증이 실패합니다.
- 라이선스는 강력한 온라인 검증 로직이고 앱이 처음 시작된 후 TXLiveBase#setLicence를 호출할 때 네트워크 사용 가능한 상태여야 합니다. 앱이 처음 시작될 때 아직 네트워크 권한이 부여되지 않았을 수 있으므로, 네트워크 권한이 부여된 후 다시 TXLiveBase#setLicence를 호출해야 합니다.
- TXLiveBase#setLicence 로드 결과의 모니터링: onLicenceLoaded 인터페이스가 실패 시 실제 상황에 따라 재시도 및 안내를 수행하고, 여러 번 실패한 경우 빈도 제한을 적용하고 제품 팝업 등의 안내를 통해 사용자가 네트워크 상태를 확인하도록 유도할 수 있습니다.
- TXLiveBase#setLicence는 여러 번 호출할 수 있으며 App 메인 화면 진입 시 TXLiveBase#setLicence를 호출하여 로딩이 성공적으로 완료되도록 하는 것이 좋습니다.
- 멀티 프로세스 앱의 경우, 플레이어를 사용하는 각 프로세스가 시작될 때 TXLiveBase#setLicence가 호출되도록 해야 합니다. 예를 들어,Android에서 독립 프로세스로 비디오를 재생하는 앱의 경우, 백그라운드 재생 중 프로세스가 시스템에 의해 종료되고 다시 시작될 때도 TXLiveBase#setLicence를 호출해야 합니다.

단계5: SDK 초기화

RTC Engine SDK의 초기화

```
// RTC Engine SDK 인스턴스의 생성(싱글톤 모드)
self.trtcCloud = [TRTCCloud sharedInstance];
// 이벤트 리스너의 설정
self.trtcCloud.delegate = self;

// SDK의 다양한 이벤트 알림(예: 오류 코드, 경고 코드, 오디오/비디오 상태 매개
변수 등)
- (void)onError:(TXLiteAVError)errCode errMsg:(nullable NSString
*)errMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", errCode, errMsg);
}

- (void)onWarning:(TXLiteAVWarning)warningCode warningMsg:(nullable
NSString *)warningMsg extInfo:(nullable NSDictionary *)extInfo {
    NSLog(@"%d: %@", warningCode, warningMsg);
}

// 이벤트 리스너의 제거
self.trtcCloud.delegate = nil;
// RTC Engine SDK 인스턴스(싱글톤 모드)를 파기합니다.
[TRTCCloud destroySharedIntance];
```

설명:
SDK 이벤트 알림을 모니터링하고 일반적인 오류에 대한 로그 출력 및 처리를 권장합니다. 자세한 내용은 오류 코드 표을 참조하세요.

뷰티 AR SDK의 초기화

```
// 뷰티 관련 리소스의 로드
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
```

```
 @"root_path":[[NSBundle mainBundle] bundlePath]
};

// 텐센트 이펙트 SDK의 초기화
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize
assetsDict:assetsDict];

// 텐센트 이펙트 SDK의 릴리스
[self.beautyKit deinit];
```

설명:
뷰티 AR SDK 초기화 전에 리소스 복사 등 준비 작업이 필요하며, 자세한 단계는 뷰티 AR SDK 통합 단계을 참조하세요.

플레이어 SDK의 초기화

- VOD 재생 시나리오 SDK의 초기화.

```
// 1. SDK 접속 환경의 설정
// 전 세계 사용자를 대상으로 하는 경우 SDK 접속 환경을 글로벌 접속 환경으로
구성하세요.
[TXLiveBase setGlobalEnv:"GDPR"];

// 2. Player 생성
TXVodPlayer *_txVodPlayer = [[TXVodPlayer alloc] init];

// 3. View 렌더링
[_txVodPlayer setupVideoWidget:_myView insertIndex:0];

// 4. 플레이어 매개변수의 구성
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setEnableAccurateSeek:true];    // seek의 설정이 정확한지?
기본값 true입니다
[_config setMaxCacheItems:5];            // 캐시 파일 개수를 5로 설정
합니다
[_config setProgressInterval:200];       // 진행률 콜백 간격을 설정합
니다, 단위는 밀리초입니다
```

```
[_config setMaxBufferSize:50];           // 최대 사전 로드의 크기, 단
위 MB
[_txVodPlayer setConfig:_config];        // config를 _txVodPlayer
에 전달합니다

// 5. 플레이어 이벤트의 리스닝
- (void)onPlayEvent:(TXVodPlayer *)player event:(int)EvtID
withParam:(NSDictionary*)param {
    if (EvtID == PLAY_EVT_VOD_PLAY_PREPARED) {
        //플레이어 준비 완료 이벤트를 수신하고,이 시점에서 pause,
resume, getWidth, getSupportedBitrates 등의 인터페이스 호출할 수 있습
니다.
    } else if (EvtID == PLAY_EVT_PLAY_BEGIN) {
        //재생 시작 이벤트 수신합니다
    } else if (EvtID == PLAY_EVT_PLAY_END) {
        //재생 종료 이벤트 수신합니다
    }
}
```

- 라이브방송 재생 시나리오 SDK의 초기화.

```
// 1. Player 생성
V2TXLivePlayer *_txLivePlayer = [[V2TXLivePlayer alloc] init];

// 2. View 렌더링
[_txLivePlayer setRenderView:_myView];

// 3. 플레이어 이벤트의 리스닝
[_txLivePlayer setObserver:self];

- (void)onVideoLoading:(id<V2TXLivePlayer>)player extraInfo:
(NSDictionary *)extraInfo {
    // 비디오 로드 이벤트
}

- (void)onVideoPlaying:(id<V2TXLivePlayer>)player firstPlay:
(BOOL)firstPlay extraInfo:(NSDictionary *)extraInfo {
    // 비디오 재생 이벤트
}
```

접속 과정

API 시퀀스 다이어그램

단계1: 스트리머가 방에 입장하여 스트리밍을 시작합니다

1. 스트리머가 방에 들어가기 전에 로컬 비디오 미리보기 및 오디오 수집을 시작합니다.

```
// 스트리머의 로컬 화면 미리보기를 표시하는 데 사용되는 비디오 렌더링 컨트롤 가져
옵니다
@property (nonatomic, strong) UIView *anchorPreviewView;
@property (nonatomic, strong) TRTCCloud *trtcCloud;

- (void)setupTRTC {
    self.trtcCloud = [TRTCCloud sharedInstance];
    self.trtcCloud.delegate = self;
    // 원격 사용자가 보는 화면 품질을 결정하는 비디오 인코딩 매개변수를 설정합니
다.
```

```
    TRTCVideoEncParam *encParam = [[TRTCVideoEncParam alloc] init];
    encParam.videoResolution = TRTCVideoResolution_960_540;
    encParam.videoFps = 15;
    encParam.videoBitrate = 1300;
    encParam.resMode = TRTCVideoResolutionModePortrait;
    [self.trtcCloud setVideoEncoderParam:encParam];

    // isFrontCamera는 전면/후면 카메라를 사용하여 비디오를 캡처하도록 지정할
수 있습니다.
    [self.trtcCloud startLocalPreview:self.isFrontCamera
view:self.anchorPreviewView];

    // 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
    [self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];
}
```

⚠ 주의:

- 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam을 직접 설정할 수 있으며, 각 등급에 대한 최적의 해상도 및 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.
- `enterRoom` 전에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집만 시작하며, `enterRoom` 을 호출한 후에야 스트리밍을 시작합니다.
- `enterRoom` 후에 위 인터페이스를 호출하면 SDK는 카메라 미리보기와 오디오 수집을 시작하고 자동으로 스트리밍을 시작합니다.

2. 스트리머가 로컬 화면의 렌더링 매개변수 및 인코더 출력 화면 모드(선택 사항)를 설정합니다.

```
- (void)setupRenderParams {
    TRTCRenderParams *params = [[TRTCRenderParams alloc] init];
    // 화면 미러 모드
    params.mirrorType = TRTCVideoMirrorTypeAuto;
    // 화면 채우기 모드
    params.fillMode = TRTCVideoFillMode_Fill;
    // 화면 회전 각도
    params.rotation = TRTCVideoRotation_0;
    // 로컬 화면의 렌더링 매개변수를 설정합니다
    [self.trtcCloud setLocalRenderParams:params];
```

```
    // 인코더 출력 화면 미러 모드를 설정합니다
    [self.trtcCloud setVideoEncoderMirror:YES];
    // 비디오 인코더 출력 화면의 방향을 설정합니다
    [self.trtcCloud setVideoEncoderRotation:TRTCVideoRotation_0];
}
```

⚠ 주의:

- 로컬 화면 렌더링 파라미터의 설정은 로컬 화면의 렌더링 효과에만 영향을 줍니다.
- 인코더 출력 모드의 설정은 방의 다른 사용자들이 보는 (및 클라우드 레코딩 파일) 화면 효과에 영향을 줍니다.

3. 스트리머가 정식으로 라이브 방송을 시작하고 방에 들어가서 스트리밍을 합니다.

```
- (void)enterRoomByAnchorWithUserId:(NSString *)userId roomId:
(NSString *)roomId {
    TRTCParams *params = [[TRTCParams alloc] init];
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = @"userSig";
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = 0;
    // 스트리머 역할의 지정
    params.role = TRTCRoleAnchor;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}

// 방 입장 결과 이벤트의 콜백
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result는 방에 임장하는 데 소요된 시간(밀리초)을 나타냅니다.
        NSLog(@"Enter room succeed!");
    } else {
        // result 는 방 입장 실패의 오류 코드를 나타냅니다
        NSLog(@"Enter room failed!");
```

```
    }
}
```

⚠ 주의:

- RTC Engine 방 번호는 정수형 `roomId` 와 문자열 형식 `strRoomId` 로 나뉘며 두 유형의 방은 서로 통하지 않으므로 방 번호 유형을 통일시키는 것이 좋습니다.
- RTC Engine 사용자 역할은 스트리머와 청취자로 구분되며, 스트리머만 스트리밍 권한을 가집니다. 방 입장 시 사용자 역할을 지정해야 하며, 지정하지 않으면 기본적으로 스트리머 역할로 설정됩니다.
- 전자상거래 라이브 스트리밍 시나리오에서는 방 입장 모드로 `TRTCAppSceneLIVE` 를 선택하는 것이 좋습니다.

단계2: 시청자가 방에 들어가서 스트리밍을 시작합니다

1. 시청자가 RTC Engine 방에 입장합니다.

```
- (void)enterRoomByAudienceWithUserId:(NSString *)userId roomId:
(NSString *)roomId {
    TRTCParams *params = [[TRTCParams alloc] init];
    // 문자열 방 번호를 예로 들면
    params.strRoomId = roomId;
    params.userId = userId;
    // 업무 백엔드에서 가져온 UserSig
    params.userSig = @"userSig";
    // 손님의 SDKAppID로 교체합니다
    params.sdkAppId = 0;
    // 시청자 역할의 지정
    params.role = TRTCRoleAudience;
    // 인터랙티브 라이브 방송 시나리오로 방 입장하기
    [self.trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
}

// 방 입장 결과 이벤트의 콜백
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result는 방에 입장하는 데 소요된 시간(밀리초)을 나타냅니다.
        NSLog(@"Enter room succeed!");
    } else {
```

```
        // result 는 방 입장 실패의 오류 코드를 나타냅니다
        NSLog(@"Enter room failed!");
    }
}
```

2. 시청자가 스트리머의 오디오 및 비디오 스트림을 구독합니다.

```
- (void)onUserAudioAvailable:(NSString *)userId available:
(BOOL)available {
    // 원격 사용자가 자신의 오디오를 게시/취소합니다
    // 자동 구독 모드에서는 사용자가 아무런 작업을 하지 않아도 SDK가 원격 사용자
의 오디오를 자동으로 재생합니다.
}

- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    // 원격 사용자가 메인 비디오 화면을 게시/취소합니다
    if (available) {
        // 원격 사용자의 비디오 스트림을 구독하고 비디오 렌더링 컨트롤을 바인딩
합니다
        [self.trtcCloud startRemoteView:userId
streamType:TRTCVideoStreamTypeBig view:self.remoteView];
    } else {
        // 원격 사용자의 비디오 스트림 구독을 중지하고 렌더링 컨트롤을 릴리스합
니다
        [self.trtcCloud stopRemoteView:userId
streamType:TRTCVideoStreamTypeBig];
    }
}
```

3. 시청자가 원격 화면의 렌더링 모드를 설정합니다(선택 사항).

```
- (void)setupRemoteRenderParams {
    TRTCRenderParams *params = [[TRTCRenderParams alloc] init];
    // 화면 미러 모드
    params.mirrorType = TRTCVideoMirrorTypeAuto;
    // 화면 채우기 모드
    params.fillMode = TRTCVideoFillMode_Fill;
    // 화면 회전 각도
```

```
    params.rotation = TRTCVideoRotation_0;
    // 원격 화면의 렌더링 모드를 설정합니다
    [self.trtcCloud setRemoteRenderParams:@"userId"
streamType:TRTCVideoStreamTypeBig params:params];
}
```

단계3: 시청자가 마이크 켜고 인터랙션 합니다

1. 시청자가 스트리머 역할로 전환됩니다.

```
- (void)switchToAnchor {
    // 스트리머 역할로 전환됩니다
    [self.trtcCloud switchRole:TRTCRoleAnchor];
}

// 역할 전환 이벤트의 콜백
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(NSString *)errMsg
{
    if (errCode == ERR_NULL) {
        // 역할 전환 성공
    }
}
```

2. 시청자가 로컬 오디오/비디오 수집 및 스트리밍을 시작합니다.

```
- (void)setupTRTC {
    // 원격 사용자가 보는 화면 품질을 결정하는 비디오 인코딩 매개변수를 설정합니
다.
    TRTCVideoEncParam *encParam = [[TRTCVideoEncParam alloc] init];
    encParam.videoResolution = TRTCVideoResolution_480_270;
    encParam.videoFps = 15;
    encParam.videoBitrate = 550;
    encParam.resMode = TRTCVideoResolutionModePortrait;
    [self.trtcCloud setVideoEncoderParam:encParam];

    // isFrontCamera는 전면/후면 카메라를 사용하여 비디오를 캡처하도록 지정할
수 있습니다.
    [self.trtcCloud startLocalPreview:self.isFrontCamera
view:self.audiencePreviewView];
```

```
    // 여기서는 음질을 지정할 수 있으며, 낮음에서 높음 순으로
SPEECH/DEFAULT/MUSIC입니다.
    [self.trtcCloud startLocalAudio:TRTCAudioQualityDefault];
}
```

> ⚠ 주의:
> 업무 요구에 따라 비디오 인코딩 매개변수 TRTCVideoEncParam을 직접 설정할 수 있으며, 각 등급에 대한 최적의 해상도 및 비트레이트 조합은 해상도 비트레이트 참조표에서 확인할 수 있습니다.

3. 시청자가 마이크를 끄고 스트리밍을 중지합니다.

```
- (void)switchToAudience {
    // 시청자 역할로 전환됩니다
    [self.trtcCloud switchRole:TRTCRoleAudience];
}

// 역할 전환 이벤트의 콜백
- (void)onSwitchRole:(TXLiteAVError)errCode errMsg:(NSString *)errMsg
{
    if (errCode == ERR_NULL) {
        // 카메라 수집 및 스트리밍 중지합니다
        [self.trtcCloud stopLocalPreview];
        // 마이크 수집 및 스트리밍 중지합니다
        [self.trtcCloud stopLocalAudio];
    }
}
```

단계4: 방 나가기 및 해산하기

1. 방에서 나가기.

```
- (void)exitRoom {
    [self.trtcCloud stopLocalAudio];
    [self.trtcCloud stopLocalPreview];
    [self.trtcCloud exitRoom];
}

// 방 나가기 이벤트의 콜백
- (void)onExitRoom:(NSInteger)reason {
```

```
    if (reason == 0) {
        NSLog(@"exitRoom을 호출하여 방에서 나가기");
    } else if (reason == 1) {
        NSLog(@"서버에 의해 현재 방에서 나가게 됩니다");
    } else if (reason == 2) {
        NSLog(@"현재 방 전체가 해체됩니다");
    }
}
```

> ⚠ 주의:
> - SDK가 점유한 모든 리소스가 릴리스된 후, SDK는 `onExitRoom` 콜백을 통해 알려줍니다.
> - `enterRoom` 을 다시 호출하거나 다른 음성/영상 SDK로 전환하려면 `onExitRoom` 콜백이 발생한 후야만 관련 작업을 수행하십시오. 그렇지 않으면 카메라, 마이크 장치에 점유됨 등 다양한 오류 문제가 발생할 수 있습니다.

2. 방 해산
   - **서버 측에서 방 해산합니다**

     RTC Engine은 서버 측에서 숫자 유형 방을 해산하는 API 과 문자열 유형 방을 해산하는 API `DismissRoomByStrRoomId` 를 제공합니다. 서버 측의 방 해산 인터페이스를 호출하여 방 내 모든 사용자를 방에서 나가게 시키고 방을 해산할 수 있습니다.
   - **클라이언트 측에서 방 해산합니다**

     클라이언트에는 방을 직접 해산하는 API가 없으며, 각 클라이언트가 exitRoom 을 호출하여 방을 나가야 합니다. 방 내 모든 스트리머와 시청자가 방을 나가면 RTC Engine 방 라이프사이클 규칙에 따라 방이 자동으로 해산됩니다. 자세한 내용은 RTC Engine 방 나가기 을 참조하십시오.

> ⚠ 주의:
> 라이브 방송이 종료된 후에는 서버 측의 방 해산 API를 호출하여 방을 해산시키는 것이 좋습니다. 이는 일부 사용자가 예정대로 방을 나가지 않아 방이 유지되고 예상치 못한 비용이 발생하는 것을 방지하기 위함입니다.

고급 기능

상품 정보의 팝업

상품 정보 팝업 기능은 Chat 자체 정의 메시지 을 통해 구현할 수 있으며, SEI 정보 을 통해서도 구현할 수 있습니다. 아래에서는 이 두 가지 구현 방식을 각각 소개하겠습니다.

자체 정의 메시지

자체 정의 메시지는 텐센트 클라우드 Chat의 기능에 의존하며, 서비스를 사전에 개통하고 Chat SDK를 임포트해야 합니다. 자세한 안내는 음성 채팅방 접속 가이드-접속 준비을 참조하십시오.

1. 자체 정의 메시지의 전송.
   - 방식 1: 스트리머가 클라이언트에서 상품 팝업과 관련된 그룹 커스텀 메시지를 전송합니다.

```
// 상품 팝업 메시지 본문의 구성
NSDictionary *msgDict = @{
    @"itemNumber": @1,        // 상품 번호
    @"itemPrice": @199.0,     // 상품 가격
    @"itemTitle": @"xxx",     // 상품 제목
    @"itemUrl": @"xxx"        // 상품 이미지 주소
};
NSDictionary *dataDict = @{
    @"cmd": @"item_popup_msg",
    @"msg": msgDict
};
NSError *error;
NSData *data = [NSJSONSerialization dataWithJSONObject:dataDict
options:0 error:&error];

// 그룹 커스텀 메시지의 전송 (상품 팝업 메시지는 높은 우선순위로 설정하는 것이 좋음)
[[V2TIMManager sharedInstance] sendGroupCustomMessage:data
to:groupID priority:V2TIM_PRIORITY_HIGH succ:^{
    // 상품 팝업 메시지의 전송이 성공됩니다
    // 로컬에서 상품 팝업 효과를 렌더링합니다
} fail:^(int code, NSString *desc) {
    // 상품 팝업 메시지의 전송이 실패됩니다
}];
```

   - 방식 2: 백엔드 운영은 서버 측에서 상품 팝업과 관련된 그룹 커스텀 메시지를 전송합니다.

     URL 요청 예시:

```
https://xxxxxx/v4/group_open_http_svc/send_group_msg?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&con
tenttype=json
```

     패킷 요청 예시:

```
{
    "GroupId": "@TGS#12DEVUDHQ",
    "Random": 2784275388,
    “MsgPriority": "High",  // 메시지의 우선순위에서 상품 팝업 메시지는
높은 우선순위로 설정하는 것을 권장합니다
    "MsgBody": [
        {
            "MsgType": "TIMCustomElem",
            "MsgContent": {
                // itemNumber: 상품 번호; itemPrice: 상품 가격;
itemTitel: 상품 제목; itemUrl: 상품 이미지 주소
                "Data": "{\"cmd\": \"item_popup_msg\", \"msg\":
{\"itemNumber\": 1, \"itemPrice\": 199.0, \"itemTitle\": \"xxx\",
\"itemUrl\": \"xxx\"}}"
            }
        }
    ]
}
```

2. 자체 정의 메시지의 수신.

   방 내 다른 사용자 클라이언트가 그룹 커스텀 메시지 콜백을 수신한 후 메시지 분파싱 및 상품 팝업 효과의 렌더링을 수행합니다.

```
// 그룹 커스텀 메시지의 수신
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];
- (void)onRecvGroupCustomMessage:(NSString *)msgID groupID:(NSString
*)groupID sender:(V2TIMGroupMemberInfo *)info customData:(NSData
*)data {
    if (data.length > 0) {
        NSError *error;
        NSDictionary *dataDict = [NSJSONSerialization
JSONObjectWithData:data options:0 error:
```