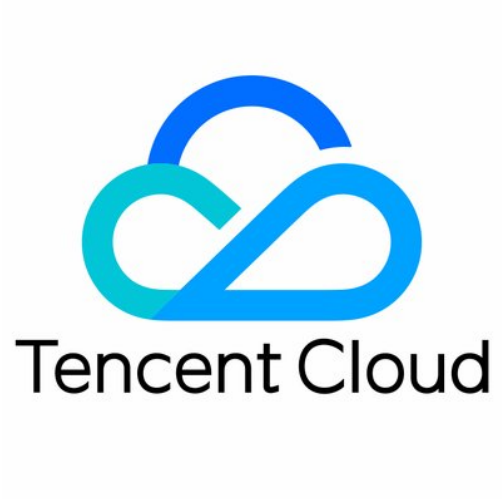
Cloud Streaming Services

SDK 사례

제품 문서



목록:

SDK 사례
0. SDK 통합 가이드

최종 업데이트 날짜: : 2022-12-23 10:41:21

본문은 라이브 푸시 및 스트림 풀 솔루션을 비즈니스 프로그램에 신속하게 통합하는 방법을 설명합니다. 다양한 비즈니스용 SDK는 다음과 같습니다.

기능 모듈		App	Web	문서
스트림 푸시		iOS & Android	Web	Stream Push
재생		iOS & Android	-	Playback
고급 기능	AV1 비디오 재생	AV1 인코딩		Overview
	고품질 업스트림	TMIO SDK	-	
	플레이어별 LEB 재생	Integrating libLebConnection SDK into a Player	-	
	화면 댓글 및 채팅	On-Screen Comment and Chat	-	
	뷰티 필터 및 특수 효과	iOS & Android	-	

2. 재생

최종 업데이트 날짜: : 2022-12-23 10:41:21

준비 작업

- Tencent Cloud CSS 서비스를 활성화해야 합니다.
- 도메인 관리를 선택하고 **도메인 추가**를 클릭한 다음 외부 도메인 추가에 설명된 대로 재생 도메인 이름을 추가합니다.
- CSS 콘솔에서 주소 생성기의 지침에 따라 **CSS 툴 박스** > 주소 생성기에서 재생 주소를 생성합니다. 그 다음 다음과 같이 비즈니스 시나리오에 따라 자신의 비즈니스에서 라이브 재생을 구현합니다.

App에 통합

iOS & Android 통합 가이드의 지침에 따라 MLVB SDK를 다운로드하고 통합합니다.

주의 :
스트림 가져오기 및 재생을 활성화합니다. 다음은 iOS 및 Android용 구성입니다.

- iOS

```
V2TXLivePlayer *_txLivePlayer = [[V2TXLivePlayer alloc] init];
>
```

- Android

```
V2TXLivePlayer mLivePlayer = new V2TXLivePlayerImpl(mContext);
```

더 보기

- MLVB SDK를 사용하면 요금이 부과됩니다. 자세한 내용은 Billing Overview를 참고하십시오.

3. 고급 기능

개요

최종 업데이트 날짜: : 2022-12-23 10:41:21

본문은 고급 라이브 스트리밍 기능을 프로그램으로 가져오는 방법을 설명합니다.

AV1 비디오 재생

AV1은 로열티가 없는 오픈 소스 비디오 압축 형식입니다. 이전 H.265[HEVC] 인코딩과 비교하여 동일한 수준의 화질을 유지하면서 비트레이트를 30% 이상 더 줄일 수 있습니다. 이는 동일한 대역폭에서 더 높은 이미지 품질을 제공할 수 있음을 의미합니다. 현재 CSS는 AV1 인코딩을 지원합니다. 그러나 AV1 비디오를 재생하려면 플레이어가 AV1 형식 디코딩을 지원해야 합니다.
다음과 같이 자체 플레이어에서 AV1 디코딩을 구현할 수 있습니다.

컨테이너 형식 및 배포 프로토콜

Tencent Cloud는 AV1 in FLV에 대한 독점 확장(in T-FFmpeg)을 구현했습니다. 플레이어를 수정하려면 T-FFmpeg의 Patch 파일을 기반으로 코드를 확장할 수 있습니다. 자세한 내용은 tencentyun/AV1을 참고하십시오.

디코딩

- **하드웨어 디코딩**
  PC에서 AMD, Intel 및 Nvidia GPU의 거의 모든 새 모델은 AV1 하드웨어 디코딩을 지원합니다.
  AV1 하드웨어 디코딩을 지원하는 장치 모델은 다음과 같습니다.

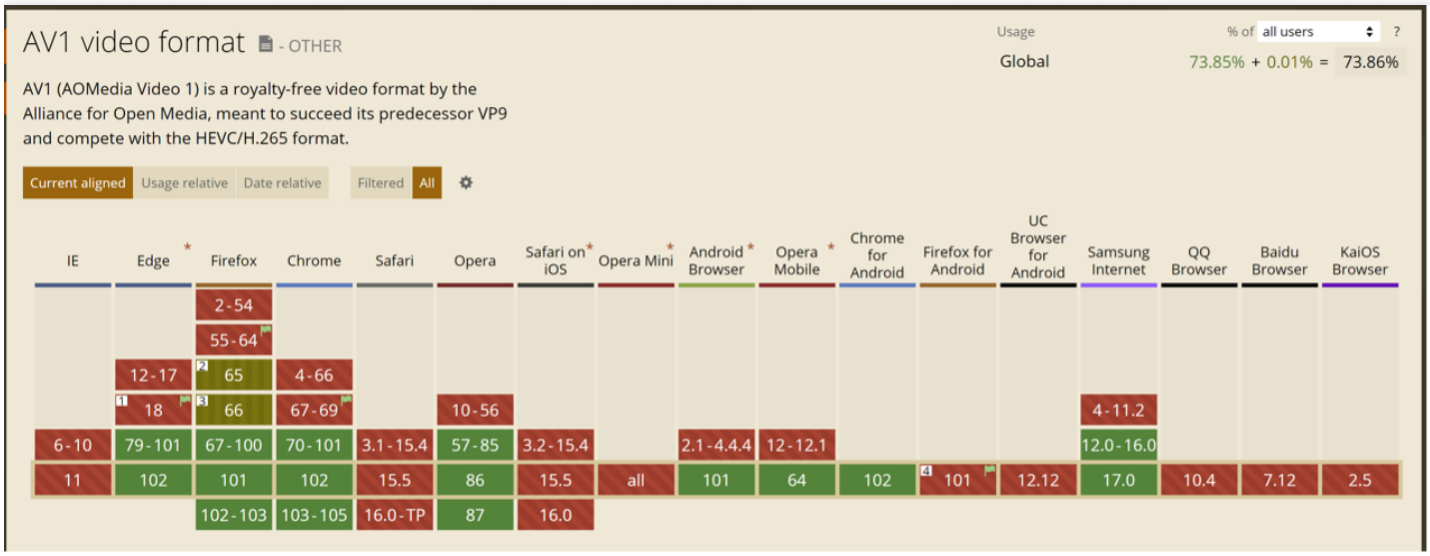
유형	브랜드	프로세서
휴대폰	realme X7 Pro	Dimensity 1000+
	oppo reno 5 pro	Dimensity 1000+
	HONOR v40	Dimensity 1000+
	Redmi k30 Ultra	Dimensity 1000+
	vivo iQOO Z1	Dimensity 1000+
	Redmi Note 10 Pro	Dimensity 1000+
	vivo S9	Dimensity 1100

유형	브랜드	프로세서
	realme Q3 Pro	Dimensity 1100
	vivo s10	Dimensity 1100
	vivo s12	Dimensity 1100
	vivo s12 pro	Dimensity 1200
	OPPO Reno6 Pro	Dimensity 1200
	OPPO Reno7 Pro	Dimensity 1200
	Redmi K40 pro	Dimensity 1200
	realme GT Neo	Dimensity 1200
	HONOR X20	Dimensity 1200
	OnePlus Nord 2	Dimensity 1200
	realme GT Neo2	Dimensity 1200
	OPPO K9 Pro	Dimensity 1200
	OPPO Find X5 Pro Dimensity 버전	Dimensity 9000
	Redmi K50	Dimensity 9000
	Galaxy S21(삼성 Exynos 버전)	Exynos 2100
	Galaxy S22(삼성 Exynos 버전)	Exynos 2200
TV	삼성 Q950TS QLED 8K TV	-

- **소프트웨어 디코딩**
  - av1d (Tencent Cloud의 최적화된 AV1 디코더로 dav1d를 능가하며 폐쇄 소스 라이브러리를 제공할 수 있습니다. av1d 통합 가이드의 지침에 따라 통합할 수 있습니다. T-FFmpeg는 통합할 FFmpeg Patch와 av1d 라이브러리를 제공합니다.
  - dav1d
  - libgav1
  - Android 10.0은 AV1 디코더를 통합합니다
  - Chrome 제품군은 AV1 디코딩을 지원합니다

브라우저 지원

Chrome 제품군은 AV1 디코딩을 지원하지만 iOS의 브라우저는 지원하지 않습니다.



주의 :
상기 정보는 2022년 7월에 AVI 웹사이트에서 수집한 것입니다. 최신 정보를 보려면 웹사이트를 방문하십시오.

MediaConnect SDK (TMIO SDK)

TMIO SDK는 SRT 및 QUIC와 같은 스트리밍 미디어 프로토콜을 사용자 지정, 캡슐화 및 최적화합니다. 업스트림을 보호하고 높은 패킷 손실 방지율, 다중 연결 전송 최적화 및 RTO(재전송 시간 초과) 메커니즘으로 지연 시간이 짧은 전송을 구현합니다. 높은 데이터 소스 안정성과 장거리 전송이 필요한 시나리오에서 폭넓은 적용을 약속합니다.

기능 소개

- TMIO SDK는 장거리 전송 및 라디오 및 TV 산업에 적합합니다.
- TMIO SDK는 Android, iOS, Windows, MacOS 및 Linux를 포함한 주요 플랫폼을 지원합니다.

통합 방식

TMIO SDK의 안내에 따라 SDK를 연동할 수 있습니다.

libLebConnection SDK

libLebConnection SDK는 기본 WebRTC를 기반으로 업그레이드된 전송 기능을 제공합니다. 몇 가지 간단한 수정만으로 기존 플레이어를 LEB에 연결할 수 있습니다. LVB 호환 스트림 푸시 및 클라우드 기반 미디어 처리 기능을 기반으로 높은 동시성에서도 저지연 라이브 스트리밍을 구현할 수 있으므로 표준 LVB 스트리밍에서 지연 시간이 짧은 LEB 스트리밍으로 원활하게 마이그레이션할 수 있습니다. 최신 RTC(실시간 통신) 시나리오의 대형 회의실의 경우 저렴한 비용과 낮은 대기 시간으로 릴레이된 라이브 스트리밍을 신속하게 구현하는 데 도움이 될 수도 있습니다.

기능 소개

- libLebConnection SDK는 약한 네트워크 조건에서도 짧은 대기 시간으로 오디오/비디오 스트림을 가져올 수 있습니다.
- B 프레임이 있는 H.264, H.265 및 AV1 비디오를 재생하고 각각 H.264/H.265 및 AV1 입력 비디오에 대한 AnnexB 및 OBU 파일과 같은 원시 비디오 프레임 데이터로 출력할 수 있습니다.
- AAC 및 OPUS 오디오를 재생하고 원시 오디오 프레임 데이터로 출력할 수 있습니다.
- Android, iOS, Windows, Linux 및 Mac을 지원합니다.

통합 방식

플레이어에 libLebConnection SDK 통합의 지침에 따라 libLebConnection SDK를 통합할 수 있습니다.

특수 효과 및 뷰티 필터

뷰티 필터와 특수 효과를 통합하고 라이브 스트리밍 중에 뷰티 필터, 이미지 필터 및 스티커를 가져오려면 Tencent Cloud · Tencent 특수 효과 엔진(Tencent Effect) SDK를 통합할 수 있습니다.

App에 통합

여기에서 Tencent 특수 효과 엔진(Tencent Effect) SDK를 다운로드하고 iOS & Android 통합 가이드의 지침에 따라 통합합니다.

더 보기

Tencent Effect SDK를 사용하면 요금이 발생합니다. 자세한 내용은 가격 리스트를 참고하십시오.

TMIO SDK

최종 업데이트 날짜: : 2023-06-14 15:24:26

개요

점점 더 많은 스트리밍 미디어 전송 프로토콜이 등장함에 따라 TMIO SDK(Tencent Media IO SDK)는 주요 프로토콜을 통합, 최적화 및 확장하여 안정적이고 사용 가능한 미디어 애플리케이션을 쉽게 개발할 수 있도록 지원하여 다양한 프로토콜의 힘든 개발 및 디버깅에서 자유로울 수 있게 합니다.
TMIO SDK는 SRT 및 QUIC 등 주류 스트리밍 미디어 프로토콜을 최적화 및 확장했으며, 자체 개발 전송 제어 프로토콜 ETC(Elastic Transmission Control)를 추가했습니다. 향후 더 많은 주류 스트리밍 미디어 프로토콜을 최적화하도록 계속 확장될 것입니다.

강점

- **멀티 플랫폼 지원**: Android, iOS, Linux, Mac 및 windows 지원.
- **유연한 통합 방법**:
  - 프록시 모드 선택에 설명된 대로 비간섭 프록시 모드 제공.
  - 간단한 API 설계를 기반으로 신속하게 통합하여 기존 전송 프로토콜 대체 가능. 내부 통합을 참고하십시오.
- **API 설계가 간단하고 호환성과 유연성이 높음**.
  - 사용하기 쉬운 API 제공.
  - 비즈니스 요구 사항 및 시나리오에 따라 적절한 모드와 정책 선택 가능.
  - 다른 프로토콜을 사용자 지정, 최적화, 확장 가능.
- **다양한 전송 프로토콜 확장, 최적화 및 통합**:
  - SRT, QUIC 등 차세대 주요 전송 프로토콜 및 자체 개발 전송 제어 프로토콜 ETC 지원
  - 다양한 비즈니스 시나리오에 적용 가능한 다양한 수요 옵션
  - UDP 기반의 저지연, 안전하고 안정적인 전송 설계
  - 보다 안정적이고 원활한 전송을 보장하는 다중 연결 가속 이점

효과 표시(TMIO SRT 예시)

- **TMIO는 업스트림 안정성과 다운스트림 원활성을 향상시키기 위해 약한 네트워크 및 장거리 전송 시나리오에서 사용할 수 있는 SRT 프로토콜 지원.**
  다음 테스트 시나리오에서, RTMP 스트리밍은 10% 패킷 손실률에서 이미 랙이 발생하였으나, SRT 스트리밍은 10%에서는 물론 30% 패킷 손실률에서도 안정적이고 낮은 딜레이 상태를 유지합니다.

기능 소개

- **SRT 기반 스트리밍 미디어 전송**
  - **높은 랜덤 패킷 손실 방지율 제공**.
  - **ARQ 및 타임아웃 정책에 기반한 재전송 메커니즘 구현**
  - **UDT 기반의 저지연, 안전하고 안정적인 전송 설계**.
  - **새로운 연결 취합 기능을 통한 다중 연결 전송**:
    다중 연결 전송 기능을 사용하면 여러 연결을 구성하여 데이터를 전송할 수 있습니다. 4G/5G 네트워크가 널리 사용되는 현 시대에 모바일 기기는 Wi-Fi와 데이터 네트워크 모두를 통해 데이터를 전송할 수 있으므로 갑자기 네트워크 연결이 끊어졌을 때 하나의 연결만 가능하더라도 연결 안정성을 보장할 수 있습니다.

기능 모드	설명
방송 모드	이 모드에서는 데이터 무결성과 연결 안정성을 보장하는 중복 데이터를 전송하도록 여러 링크를 구성할 수 있습니다.
프라이머리/세컨더리 모드	이 모드에서는 연결 안정성과 신뢰도에 따라 한 번에 하나의 링크만 활성화되며 실시간으로 최적의 연결을 선택하여 데이터를 전송합니다. 이는 연결 안정성과 신뢰성을 보장할 뿐만 아니라 중복 데이터의 대역폭 사용을 줄입니다.

기능 모드	설명
취합 모드	높은 비트 레이트와 대역폭이 필요한 시나리오에서 단일 연결의 대역폭이 요구 사항을 충족할 수 없는 경우 이 모드는 데이터를 분할하고 여러 연결을 통해 데이터를 보낸 다음 수신기 끝에서 데이터를 재결합하여 사용 가능한 대역폭을 늘릴 수 있습니다.

- **QUIC 기반 스트리밍 미디어 전송**
  - **어댑티브 혼잡 방지 알고리즘**
  - **네트워크 연결 마이그레이션 지원, 매끄럽고 감지 불가**
  - **차세대 HTTP3 기본 전송 프로토콜 지원**
  - **대역폭 제한 및 지터 환경에서는 중복 데이터 전송을 줄이고 대역폭 비용을 절약하여, 이점이 큼**

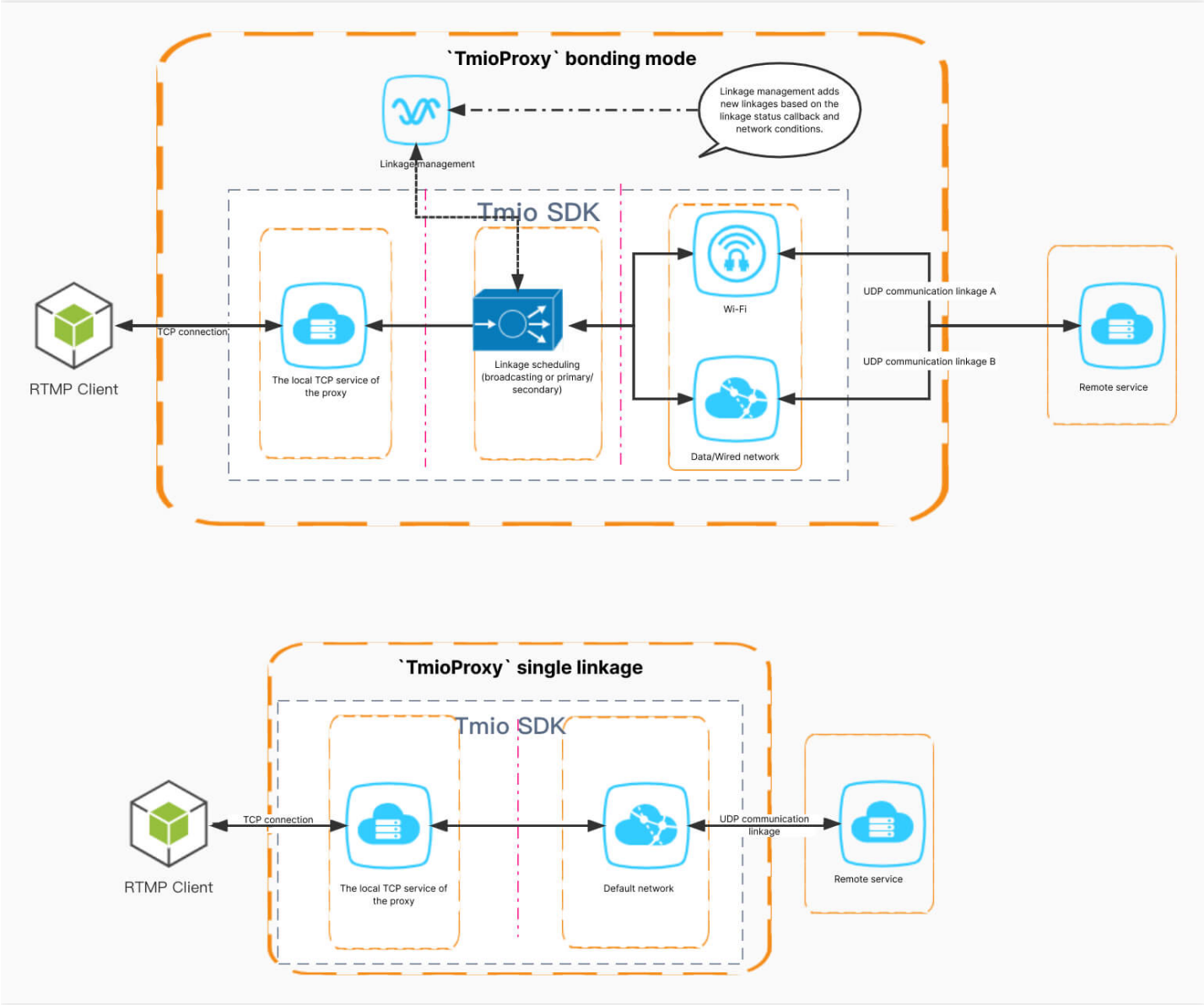
- **자체 개발 전송 제어 프로토콜 ETC**
  - **순수 자체 개발, 경량, 크로스 플랫폼**
  - **엔드 투 엔드 통신에 적합한 IoT 장치 지원**
  - **빠른 실행, 낮은 딜레이 시간 및 높은 대역폭 사용률**
  - **빠르고 정확하게 링크 상태 변화를 감지하고 적시에 최상의 전송 정책으로 조정**
  - **주요 전송 프로토콜과 공존할 경우, 대역폭을 보다 공정하고 안정적으로 사용할 수 있음**

통합 방법

RTMP over SRT 프로토콜을 예로 듭니다.

프록시 모드 선택

Tmio Proxy 모드에서 통합



작업 순서

1. **Tmio Proxy 인스턴스 생성**:

```
std::unique_ptr<tmio::TmioProxy> proxy_ = tmio::TmioProxy::createUnique();
```

2. **리스너 설정**:

```
void setListener(TmioProxyListener *listener);
```

TmioProxyListener 리스너 API는 아래와 같습니다.

- Tmio 구성에 대한 콜백
- TmioProxy 시작에 대한 콜백
- 오류 메시지에 대한 콜백

이 콜백에서 Tmio 매개변수를 구성할 수 있습니다. **간단한 구성을 위해 `tmio-preset.h` 에서 제공하는 보조 방법을 사용할 수 있습니다.**

```
/*
void onTmioConfig(Tmio *tmio);
*/
void onTmioConfig(tmio::Tmio *tmio) override {
auto protocol = tmio->getProtocol();
if (protocol == tmio::Protocol::SRT) {
tmio::SrtPreset::rtmp(tmio);
} else if (protocol == tmio::Protocol::RIST) {
tmio->setIntOption(tmio::base_options::RECV_SEND_FLAGS,
tmio::base_options::FLAG_SEND);
}
}
```

3. **프록시 시작**:

```
std::error_code start(const std::string &local_url, const std::string &remote_ur
l, void * config=nullptr)
```

- API 매개변수

매개변수	비고
local_url	TCP Scheme만 지원합니다. 형식은 `tcp://${ip}:${port}` 입니다. port는 0일 수 있으며 이는 임의의 포트를 바인딩함을 나타냅니다. 바인딩이 성공하면 onStart() 콜백을 통해 바인딩된 포트가 애플리케이션으로 반환됩니다. 포트 0을 사용하면 포트 점유 및 권한 없음과 같은 문제로 인한 바인딩 실패를 방지할 수 있습니다.
remote_url	원격 서버 URL
config	SRT bonding 기능 및 QUIC H3 프로토콜이 활성화된 경우에만 적용되는 구성 매개변수입니다. 구체적인 정의는 `tmio.h` 의 TmioFeatureConfig 구조 정의를 참고하십시오.

- 단일 연결(예시 코드)

```
proxy_->start(local_url, remote_url, NULL);
```

- 다중 연결 bonding(예시 코드)

```
tmio::TmioFeatureConfig option;
option_.protocol = tmio::Protocol::SRT;
option_.trans_mode = static_cast<int>(tmio::SrtTransMode::SRT_TRANS_BACKUP);
/*-------------------------------------------------------------*/
{
//필요에 따라 여러 연결을 추가할 수 있습니다
option_.addAvailableNet(net_name, local_addr, remote_url, 0, weight, -1);
}
/*-------------------------------------------------------------*/

proxy_->start(local_url, remote_url, &option_);
```

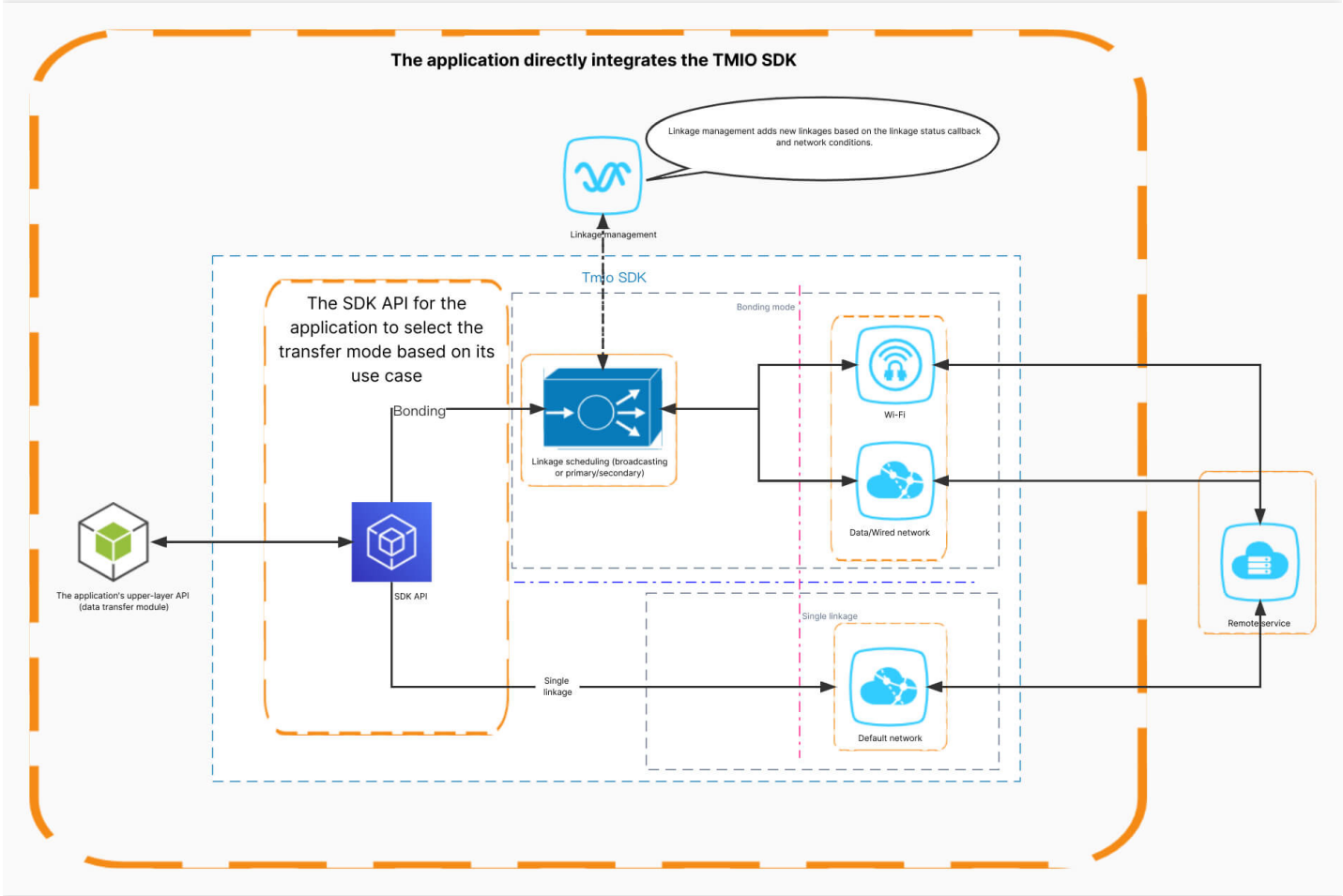
1. 중지:

```
/*
void stop();
```

```
*/
proxy_.stop();
```

내부 통합

Tmio SDK의 내부 통합



통합 프로세스

1. **Tmio 인스턴스 생성&매개변수 구성**(예시 코드):

```
tmio_ = tmio::TmioFactory::createUnique(tmio::Protocol::SRT);
tmio::SrtPreset::mpegTsLossless(tmio_.get());
tmio_->setIntOption(tmio::srt_options::CONNECT_TIMEOUT, 4000);
tmio_->setBoolOption(tmio::base_options::THREAD_SAFE_CHECK, true);
```

- **Tmio 생성**: `TmioFactory` 를 사용하여 생성할 수 있습니다.
- **매개변수 구성**: 다른 API를 선택하여 매개변수를 구성합니다.
  - **매개변수**: `tmio-option.h` 를 참고하십시오.
  - **간단한 구성**: `tmio-preset.h` 를 참고하십시오.

```
//다양한 매개변수 속성을 기반으로 적절한 구성 선택
bool setBoolOption(const std::string &optname, bool value);
bool setIntOption(const std::string &optname, int64_t value);
bool setDoubleOption(const std::string &optname, double value);
```

```
bool setStrOption(const std::string &optname, const std::string &value);
...
```

2. **연결 시작**(예시 코드):

```
/**
```

- open the stream specified by url
-
- @param config protocol dependent
- /

```
virtual std::error_code open(const std::string &url,
```

```
<span class="hljs-keyword">void</span> *config = <span class="hljs-literal">nul
lptr</span>)</span> </span>= <span class="hljs-number">0</span>;
```

- 단일 연결(config는 NULL일 수 있음)

```
//기본적으로 단일 링크
auto err = tmio->open(TMIO_SRT_URL);
if (err) {
LOGE("open failed, %d, %s", err.value(), err.message().c_str());
}
```

- 다중 연결 bonding(현재 SRT 프로토콜만 지원)
  - SRT bonding 기능에 대한 config를 설정하려면 `tmio.h` 파일의 TmioFeatureConfig 구조 정의를 참고하십시오.

```
tmio::TmioFeatureConfig option_;
option_.protocol = tmio::Protocol::SRT;
option_.trans_mode = static_cast<int>(tmio::SrtTransMode::SRT_TRANS_BACKUP);
option_.addAvailableNet(net_name, local_addr, remote_url, 0, weight, -1);
```

```
//다중 연결 bonding
auto err = tmio_->open(TMIO_SRT_URL, &option_);
if (err) {
LOGE("open failed, %d, %s", err.value(), err.message().c_str());
}
```

- 다중 연결 bonding open API를 사용하여 데이터 전송을 위해 group에 새 연결을 추가할 수 있습니다.

3. **데이터 전송**:

```
int ret = tmio_->send(buf.data(), datalen, err);
if (ret < 0) {
LOGE("send failed, %d, %s", err.value(), err.message().c_str());
break;
}
```

4. **데이터 수신**:
   - 프로토콜에 RTMP와 같은 인터랙션이 필요한 경우 데이터 수신 API를 활성화하여 데이터를 읽고 프로토콜 인터랙션을 완료해야 합니다. 여기에 두 가지 API가 제공됩니다.

```
/**
```

- receive data
-
- @param err return error details
- @return number of bytes which were received, or < 0 to indicate error
- /

```
virtual int recv(uint8_t *buf, int len, std::error_code &err) = 0;

using RecvCallback = std::function<bool(const uint8_t *buf, int len, const
std::error_code &err)>;
/**
```

- receive data in event loop
-
- recvLoop() block current thread, receive data in a loop and pass the data to recvCallback
- @param recvCallback return true to continue the receive loop, false for break
-

```
/
virtual void recvLoop(const RecvCallback &recvCallback) = 0;
```

  - 상위 레이어 애플리케이션의 루프 읽기

```
while (true) {
ret = tmio_->recv(buf.data(), buf.size(), err);
if (ret < 0) {
LOGE("recv error: %d, %s", err.value(), err.message().c_str());
break;
}
...
}
```

  - 콜백을 통해 읽기

```
FILE *file = fopen(output_path, "w");
tmio_->recvLoop([file](const uint8_t *buf, int len,
const std::error_code &err) {
if (len < 0) {
fwrite(buf, 1, len, file);
} else if (len < 0) {
LOGE("recv error: %d, %s", err.value(), err.message().c_str());
}
return true;
});
```

5. **Tmio 인스턴스 종료**:

```
tmio_->close();
```

6. **기타**:
   이 API는 현재 연결 상태를 가져오는 데 사용됩니다(애플리케이션은 상태에 따라 스트림 푸시 정책을 조정할 수 있음).

```
tmio::PerfStats stats_;
tmio_->control(tmio::ControlCmd::GET_STATS, &stats_);
```

최신 API 및 Demo 세부 정보 설명

TMIO SDK에 액세스하기 위해서는 최신 API 및 Demo 설명을 참고할 수 있으며, 자세한 내용은 TMIO 액세스 세부 정보를 참고하십시오.

FAQ

모든 장치에서 다중 연결 SRT bonding 기능을 사용할 수 있습니까?

사용 가능한 네트워크 인터페이스가 여러 개인 기기만 다중 연결 본딩을 사용할 수 있으며 Android 기기는 Android 버전이 6.0(api level >=23) 이상인 경우에만 이 기능을 사용할 수 있습니다.

Wi-Fi에 연결된 Android 휴대폰에서 4G/5G 데이터 네트워크를 어떻게 활성화합니까?

Wi-Fi에 연결된 Android 휴대폰은 4G/5G 네트워크를 통해 직접 데이터를 전송할 수 없습니다. 데이터 네트워크를 활성화하려면 다음과 같이 데이터 네트워크 권한을 신청하십시오.

```
ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystem
Service(Context.CONNECTIVITY_SERVICE);
NetworkRequest request = new NetworkRequest.Builder().addTransportType(NetworkCap
abilities.TRANSPORT_CELLULAR)
.addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)
.build();

ConnectivityManager.NetworkCallback networkCallback = new ConnectivityManager.Net
workCallback(){
@Override
public void onAvailable(@NonNull Network network) {
Log.d(TAG, "모바일 데이터 네트워크 채널이 활성화되었습니다.");
super.onAvailable(network);
```

```
}
}
```

libLebConnection SDK를 플레이어에 통합

최종 업데이트 날짜: : 2022-12-23 10:56:49

개요

libLebConnection SDK는 네이티브 WebRTC를 기반으로 업그레이드된 전송 기능을 제공합니다. 몇 가지 간단한 수정만으로 기존 플레이어를 LEB에 연결할 수 있습니다. LVB 호환 스트림 푸시 및 클라우드 기반 미디어 처리 기능을 기반으로 높은 동시성에서도 저지연 라이브 스트리밍을 구현할 수 있으므로 표준 LVB 스트리밍에서 지연 시간이 짧은 LEB 스트리밍으로 원활하게 마이그레이션할 수 있습니다. 최신 RTC(실시간 통신) 시나리오의 대형 회의실의 경우 저렴한 비용과 낮은 대기 시간으로 릴레이된 라이브 스트리밍을 신속하게 구현하는 데 도움이 될 수도 있습니다.

기능 소개

- libLebConnection SDK는 약한 네트워크 조건에서도 짧은 대기 시간으로 오디오/비디오 스트림을 가져올 수 있습니다.
- B 프레임이 있는 H.264, H.265 및 AV1 비디오를 재생하고 각각 H.264/H.265 및 AV1 입력 비디오에 대한 AnnexB 및 OBU 파일과 같은 원시 비디오 프레임 데이터로 출력할 수 있습니다.
- AAC 및 OPUS 오디오를 재생하고 원시 오디오 프레임 데이터로 출력할 수 있습니다.
- Android, iOS, Windows, Linux 및 Mac을 지원합니다.

통합 방식

기본 API 설명

- LEB 연결을 생성합니다

```
LEB_EXPORT_API LebConnectionHandle* OpenLebConnection(void* context, LebLogLeve
l loglevel);
```

- 콜백 함수를 등록합니다

```
LEB_EXPORT_API void RegisterLebCallback(LebConnectionHandle* handle, const LebC
allback* callback);
```

- 스트림을 가져오기 위해 연결을 시작합니다

```
LEB_EXPORT_API void StartLebConnection(LebConnectionHandle* handle, LebConfig c
onfig);
```

- 연결을 중지합니다

```
LEB_EXPORT_API void StopLebConnection(LebConnectionHandle* handle);
```

- 연결을 닫습니다

```
LEB_EXPORT_API void CloseLebConnection(LebConnectionHandle* handle);
```

콜백 API 설명

```
typedef struct LebCallback {
// 로그 콜백
OnLogInfo onLogInfo;
// 비디오 정보 콜백
OnVideoInfo onVideoInfo;
// 오디오 정보 콜백
OnAudioInfo onAudioInfo;
// 비디오 데이터 콜백
OnEncodedVideo onEncodedVideo;
// 오디오 데이터 콜백
OnEncodedAudio onEncodedAudio;
// MetaData 콜백
OnMetaData onMetaData;
// 통계 콜백
OnStatsInfo onStatsInfo;
// 오류 콜백
OnError onError;
} LebCallback;
```

주의 :
데이터 구조의 자세한 정의는 `leb_connection_api.h` 를 참고하십시오.

API 호출 프로세스

1. **LEB 연결 생성**: OpenLebConnection()
2. **다양한 콜백 함수 등록**: RegisterXXXXCallback()
3. **스트림을 가져오기 위한 연결 시작**: StartLebConnection()
4. **원시 오디오/비디오 데이터를 콜백하고 출력**:

- OnEncodedVideo()
- OnEncodedAudio()

5. **연결 중지**: StopLebConnection()
6. **연결 닫기**: CloseLebConnection

통합 예시

이 예시는 Android 기기에서 널리 사용되는 일반적인 오픈 소스 플레이어 ijkplayer에 libLebConnection을 통합하는 방법을 설명합니다. 다른 플랫폼에 통합하는 방법은 예시 코드를 참고할 수도 있습니다.

최신 SDK 버전

여기에서 libLebConnection SDK를 다운로드할 수 있습니다.

통합에 대한 FAQ

랙 통계 기능 개발

버퍼링이 비활성화되었으므로 이제 비디오 렌더링 간격을 기반으로 랙 통계를 수집할 수 있습니다. 비디오 렌더링 간격이 특정 임계값을 초과하면 랙으로 간주되고 랙 기간이 총 랙 기간에 추가됩니다.
ijkplayer를 예로 들면 다음과 같이 랙 통계 기능을 개발할 수 있습니다.

1. **코드 수정**
   i. 랙 통계 수집에 필요한 변수를 VideoState 및 FFPlayer 구조에 추가합니다.

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay_def.h b/ijkmedia/ijkplayer/ff_ffplay_de
f.h
index 00f19f3c..f38a790c 100755
--- a/ijkmedia/ijkplayer/ff_ffplay_def.h
+++ b/ijkmedia/ijkplayer/ff_ffplay_def.h
@@ -418,6 +418,14 @@ typedef struct VideoState {
SDL_cond *audio_accurate_seek_cond;
volatile int initialized_decoder;
int seek_buffering;
+
+ int64_t stream_open_time;
```

```
+ int64_t first_frame_display_time;
+ int64_t last_display_time;
+ int64_t current_display_time;
+ int64_t frozen_time;
+ int frozen_count;
+ float frozen_rate;
} VideoState;
/* options specified by the user */
@@ -720,6 +728,14 @@ typedef struct FFPlayer {
char *mediacodec_default_name;
int ijkmeta_delay_init;
int render_wait_start;
int low_delay_playback;
+ int frozen_interval;
int high_level_ms;
int low_level_ms;
int64_t update_plabyback_rate_time;
int64_t update_plabyback_rate_time_prev;
} FFPlayer;
#define fftime_to_milliseconds(ts) (av_rescale(ts, 1000, AV_TIME_BASE))
@@ -844,6 +860,15 @@ inline static void ffp_reset_internal(FFPlayer *ffp)
ffp->pf_playback_volume = 1.0f;
ffp->pf_playback_volume_changed = 0;
ffp->low_delay_playback = 0;
ffp->high_level_ms = 500;
ffp->low_level_ms = 200;
+ ffp->frozen_interval = 200;
ffp->update_plabyback_rate_time = 0;
ffp->update_plabyback_rate_time_prev = 0;
av_application_closep(&ffp->app_ctx);
ijkio_manager_destroyp(&ffp->ijkio_manager_ctx);
```

ii. **랙 통계 로직 추가**

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay.c b/ijkmedia/ijkplayer/ff_ffplay.c
index 714a8c9d..c7368ff5 100755
--- a/ijkmedia/ijkplayer/ff_ffplay.c
+++ b/ijkmedia/ijkplayer/ff_ffplay.c
@@ -874,6 +874,25 @@ static void video_image_display2(FFPlayer *ffp)
VideoState *is = ffp->is;
Frame *vp;
Frame *sp = NULL;
+ int64_t display_interval = 0;
+
+ if (!is->first_frame_display_time){
+ is->first_frame_display_time = SDL_GetTickHR() - is->stream_open_time;
+ }
```

```
+
+ is->last_display_time = is->current_display_time;
+ is->current_display_time = SDL_GetTickHR() - is->stream_open_time;
+ display_interval = is->current_display_time - is->last_display_time;
+ av_log(NULL, AV_LOG_DEBUG, "last_display_time:%"PRId64" current_display_tim
e:%"PRId64" display_interval:%"PRId64"\n", is->last_display_time, is->current_dis
play_time, display_interval);
+
+ if (is->last_display_time > 0) {
+ if (display_interval > ffp->frozen_interval) {
+ is->frozen_count += 1;
+ is->frozen_time += display_interval;
+ }
+ }
+ is->frozen_rate = (float) is->frozen_time / is->current_display_time;
+ av_log(NULL, AV_LOG_DEBUG, "frozen_interval:%d frozen_count:%d frozen_time:%"PR
Id64" is->current_display_time:%"PRId64" frozen_rate: %f ", ffp->frozen_interval,
is->frozen_count, is->frozen_time, is->current_display_time, is->frozen_rate);
vp = frame_queue_peek_last(&is->pictq);
```

주의 :
이 예시에서 랙 임계값(frozen_interval)의 초기 값은 `200(ms)` 이며 필요에 따라 조정할 수 있습니다.

2. **테스트 랙 통계 수집**
   QNET을 사용하여 다음과 같이 약한 네트워크 조건을 시뮬레이션하여 테스트를 진행합니다.
   i. 여기에서 QNET을 다운로드합니다.
   ii. QNET을 열고 **추가 > 템플릿 유형 > 사용자 지정 템플릿**을 클릭하고 필요에 따라 약한 네트워크 조건에 대한 템플릿과 매개변수를 구성합니다(다음은 다운스트림 중 30% 임의 네트워크 패킷 손실).
   iii. 프로그램 목록에서 프로그램을 선택합니다.
   iv. 테스트를 위해 약한 네트워크 조건 구성을 활성화합니다.

주의 :
테스트를 용이하게 하기 위해 상기 랙 매개변수를 수정하고 데이터를 jni를 통해 Java 레이어에 전달하여 표시할 수 있습니다.

재생 중 노이즈 문제 해결(Android soundtouch 최적화)

buffer 워터마크를 기반으로 재생 속도를 조정하려면 soundtouch를 사용하여 오디오 속도를 조정해야 합니다. 그러나 네트워크 변동이 많은 경우 buffer 워터마크를 자주 조정하므로 여러 속도 조정이 필요하며, 이는 네이티브 ijkplayer가

soundtouch를 호출할 때 노이즈가 발생할 수 있습니다. 이 경우 최적화를 위해 다음 코드를 참고할 수 있습니다.

저지연 재생 모드에서 속도 조정을 위해 soundtouch가 호출되면 모든 버퍼가 soundtouch에 의해 translate됩니다.

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay.c b/ijkmedia/ijkplayer/ff_ffplay.c
index 714a8c9d..c7368ff5 100755
--- a/ijkmedia/ijkplayer/ff_ffplay.c
+++ b/ijkmedia/ijkplayer/ff_ffplay.c
@@ -2579,7 +2652,7 @@ reload:
int bytes_per_sample = av_get_bytes_per_sample(is->audio_tgt.fmt);
resampled_data_size = len2 * is->audio_tgt.channels * bytes_per_sample;
#if defined(__ANDROID__)
- if (ffp->soundtouch_enable && ffp->pf_playback_rate != 1.0f && !is->abort_reque
st) {
+ if (ffp->soundtouch_enable && (ffp->pf_playback_rate != 1.0f || ffp->low_delay_
playback) && !is->abort_request) {
av_fast_malloc(&is->audio_new_buf, &is->audio_new_buf_size, out_size * translate_
time);
for (int i = 0; i < (resampled_data_size / 2); i++)
{
```

MediaCodec이 활성화된 후 H.265 비디오를 디코딩할 수 없는 이유는 무엇입니까?

libLebConnection SDK는 H.265 비디오 스트림을 지원하지만 네이티브 ijkplayer의 **Settings**에서 `Using MediaCodec` 을 활성화하면 H.265 비디오 스트림이 MediaCodec에 의해 디코딩되지 않습니다. 이 경우 다음 코드를 참고하여 최적화할 수 있습니다.

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay_options.h b/ijkmedia/ijkplayer/ff_ffpla
y_options.h
index b021c26e..958b3bae 100644
--- a/ijkmedia/ijkplayer/ff_ffplay_options.h
+++ b/ijkmedia/ijkplayer/ff_ffplay_options.h
@@ -178,8 +178,8 @@ static const AVOption ffp_context_options[] = {
OPTION_OFFSET(vtb_handle_resolution_change), OPTION_INT(0, 0, 1) },

// Android only options
- { "mediacodec", "MediaCodec: enable H.264 (deprecated by 'mediacodec-avc')",
- OPTION_OFFSET(mediacodec_avc), OPTION_INT(0, 0, 1) },
+ { "mediacodec", "MediaCodec: enable all_videos (deprecated by 'mediacodec_all_v
ideos')",
+ OPTION_OFFSET(mediacodec_all_videos), OPTION_INT(0, 0, 1) },
{ "mediacodec-auto-rotate", "MediaCodec: auto rotate frame depending on meta",
OPTION_OFFSET(mediacodec_auto_rotate), OPTION_INT(0, 0, 1) },
{ "mediacodec-all-videos", "MediaCodec: enable all videos",
```

WebRTC 기반 로컬 스트림 혼합

최종 업데이트 날짜: : 2022-12-23 10:41:21

설명 :
스트림 푸시 SDK는 현재 베타 테스트 중입니다. 공식 버전의 과금 규칙이 적용됩니다.

SDK는 여러 비디오 스트림 혼합(예: PiP), 이미지 효과 처리(예: 미러링 및 필터), 기타 요소(예: 워터마크 및 텍스트) 추가를 포함하여 다양한 비디오 스트림 이미지 처리 기능을 제공합니다. 기본 프로세스는 다음과 같습니다. SDK는 먼저 여러 스트림을 수집하고 로컬에서 혼합하여 이미지를 결합하고 오디오를 혼합합니다. 그런 다음 다른 효과를 처리합니다. 모든 기능은 브라우저의 지원에 의존하므로 SDK에는 브라우저 성능에 대한 특정 요구 사항이 있습니다. 구체적인 API 프로토콜은 TXVideoEffectManager를 참고하십시오. 본문은 로컬 스트림 혼합의 기본 사용법을 설명합니다.

기본 사용

로컬 스트림 혼합 기능을 사용하려면 SDK를 초기화하고 SDK 인스턴스 livePusher를 가져와야 합니다. 초기화 코드는 WebRTC 프로토콜 푸시 스트리밍을 참고하십시오.

1단계: 비디오 효과 관리 인스턴스 가져오기

```
var videoEffectManager = livePusher.getVideoEffectManager();
```

2단계: 로컬 스트림 혼합 활성화

먼저 로컬 스트림 혼합 기능을 활성화해야 합니다. 기본적으로 SDK는 하나의 비디오 스트림과 오디오 스트림만 수집합니다. 이 기능이 활성화되면 SDK는 브라우저에서 로컬로 혼합되는 여러 스트림을 수집할 수 있습니다.

```
videoEffectManager.enableMixing(true);
```

3단계: 스트림 혼합 매개변수 설정

스트림 혼합 매개변수, 특히 스트림 혼합 후 비디오 출력의 해상도와 프레임 레이트를 설정합니다.

```
videoEffectManager.setMixingConfig({
videoWidth: 1280,
videoHeight: 720,
```

```
videoFramerate: 15
});
```

4단계: 여러 스트림 수집

로컬 스트림 혼합이 활성화되면 SDK는 카메라 및 공유 화면과 같은 여러 스트림을 수집하기 시작합니다. 후속 작업에 필요하므로 스트림 ID를 유지해야 합니다.

```
var cameraStreamId = null;
var screenStreamId = null;
livePusher.startCamera().then((streamId) => {
cameraStreamId = streamId;
}).catch((error) => {
console.log('카메라 켜기 실패:'+ error.toString());
});
livePusher.startScreenCapture().then((streamId) => {
screenStreamId = streamId;
}).catch((error) => {
console.log('화면 공유 실패:'+ error.toString());
});
```

5단계: 이미지 레이아웃 설정

수집된 두 스트림의 이미지 레이아웃을 설정합니다. 여기에서 공유 화면이 메인 이미지로 표시되며 카메라 이미지는 왼쪽 상단에 표시됩니다. 특정 매개변수 구성에 대해서는 TXLayoutConfig를 참고하십시오.

```
videoEffectManager.setLayout([{
streamId: screenStreamId,
x: 640,
y: 360,
width: 1280,
height: 720,
zOrder: 1
}, {
streamId: cameraStreamId,
x: 160,
y: 90,
width: 320,
height: 180,
zOrder: 2
}]);
```

6단계: 미러링 효과 설정

카메라에 의해 수집된 이미지가 실제로 반전되므로 카메라 이미지를 미러링합니다.

```
videoEffectManager.setMirror({
streamId: cameraStreamId,
mirrorType: 1
});
```

7단계: 워터마크 추가

이미지 객체를 준비하고 비디오 스트림 이미지에 워터마크로 추가합니다. 여기에서 워터마크 이미지는 오른쪽 상단 모서리에 배치됩니다.

```
var image = new Image();
image.src = './xxx.png'; // 이미지 주소는 다른 도메인에 속할 수 없음에 유의해야하며, 그렇지 않으면 도메인 간 문제가 발생합니다
videoEffectManager.setWatermark({
image: image,
x: 1230,
y: 50,
width: 100,
height: 100,
zOrder: 3
});
```

8단계: 스트림 푸시 시작

상기 단계를 거친 후 PiP 레이아웃, 미러링된 이미지 및 워터마크 출력이 포함된 비디오 스트림을 서버로 푸시합니다.

```
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx'
);
```

설명 :
WebRTC 기반 스트림 혼합 API에 대한 자세한 내용은 TXLivePusher를 참고하십시오.

댓글 자막 및 채팅 통합

최종 업데이트 날짜: : 2022-12-23 10:41:21

개요

라이브 스트리밍 비즈니스에서 호스트와 시청자는 일반적으로 화면 댓글 및 채팅과 같은 다양한 수단을 통해 실시간으로 상호 작용해야 합니다. 그러나 이러한 기능의 통합은 복잡합니다. 본문은 라이브 스트리밍 비즈니스 및 요구 사항을 이해할 수 있도록 IM을 예로 들어 라이브 스트리밍 중 화면 댓글, 채팅, 항목 추천과 같은 요구 사항을 구현하는 방법과 가능한 문제 및 고려 사항을 설명합니다.

주요 기능 설명

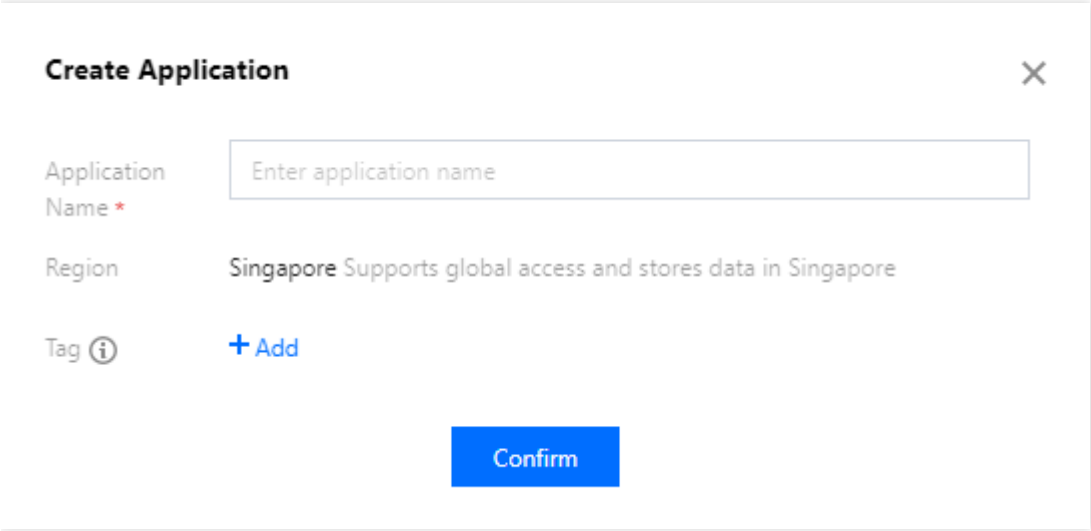
기능	설명
실시간 화면 댓글, 선물, 좋아요	친근한 인터랙션 경험을 구축하기 위해 수억 개의 메시지를 동시 발송할 수 있습니다.
일대일 채팅, 그룹 라이브 채팅 등 다양한 채팅 모드	사용자는 같은 채팅방에서 다른 구성원과 메시지를 주고받을 수 있습니다. 텍스트, 이미지, 오디오, 쇼트 비디오 등 다양한 메시지 유형을 실시간으로 푸시할 수 있어 더 많은 사용자 활동을 유도할 수 있습니다.

기능	설명
라이브 커머스에서 상품 푸시	라이브 쇼핑 시나리오에서 호스트가 제품을 추천하면 화면 하단의 아이템 슬롯에 즉시 표시되고 모든 채팅방 안의 사용자에게 알려야 합니다. 신상품에 대한 알림은 일반적으로 가상 어시스턴트에 의해 트리거됩니다.
라이브 룸에서 방송	방송 기능은 라이브 룸으로 전송되는 시스템 알림과 유사합니다. 시스템 관리자가 방송 메시지를 전달하면 SDKAppID 아래의 모든 라이브 룸이 이를 수신합니다.

액세스 방법

1단계: 애플리케이션 생성

Tencent Cloud에서 라이브 룸을 설정하려면 아래와 같이 콘솔에서 IM 애플리케이션을 생성해야 합니다.

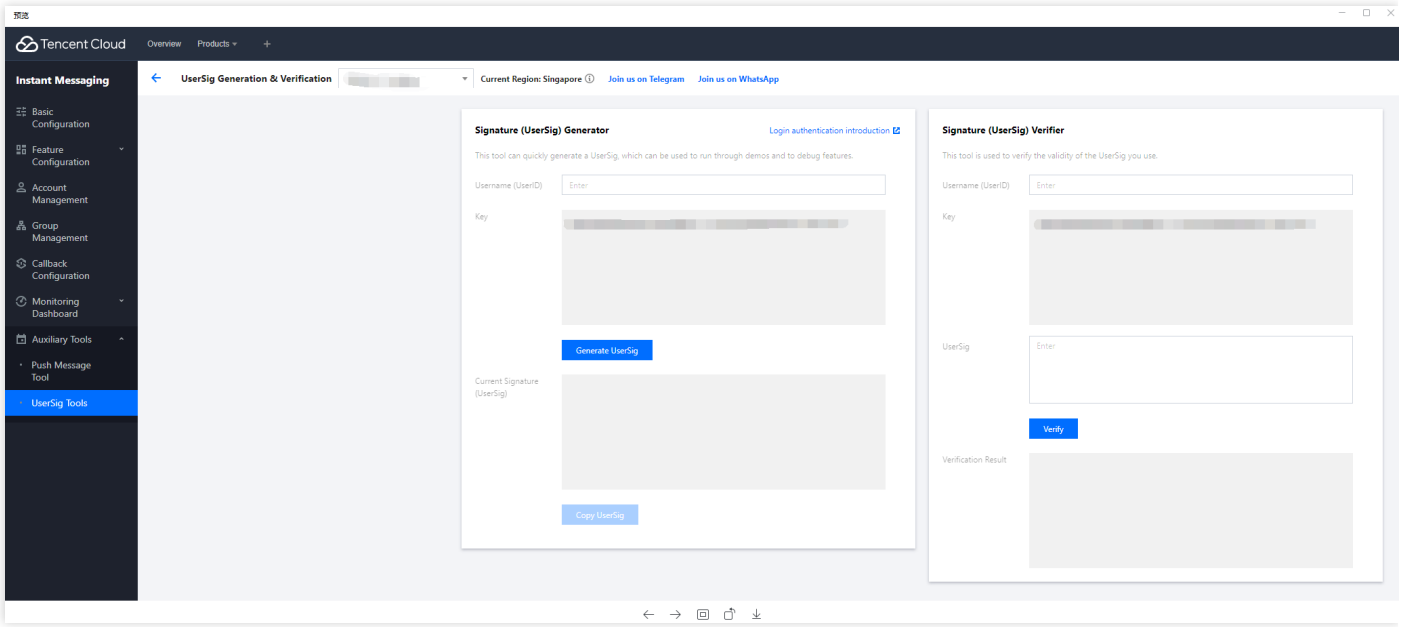


2단계: 관련 구성 완료

준비 작업에서 생성된 애플리케이션은 개발에만 적용되는 무료 버전입니다. 프로덕션 환경에서는 필요에 따라 프로 또는 플래그십 에디션을 활성화해야 합니다. 버전별 차이점에 대한 자세한 내용은 요금 안내를 참고하십시오.
라이브 스트리밍 시나리오에서는 애플리케이션을 생성한 후 몇 가지 추가 구성이 필요합니다.

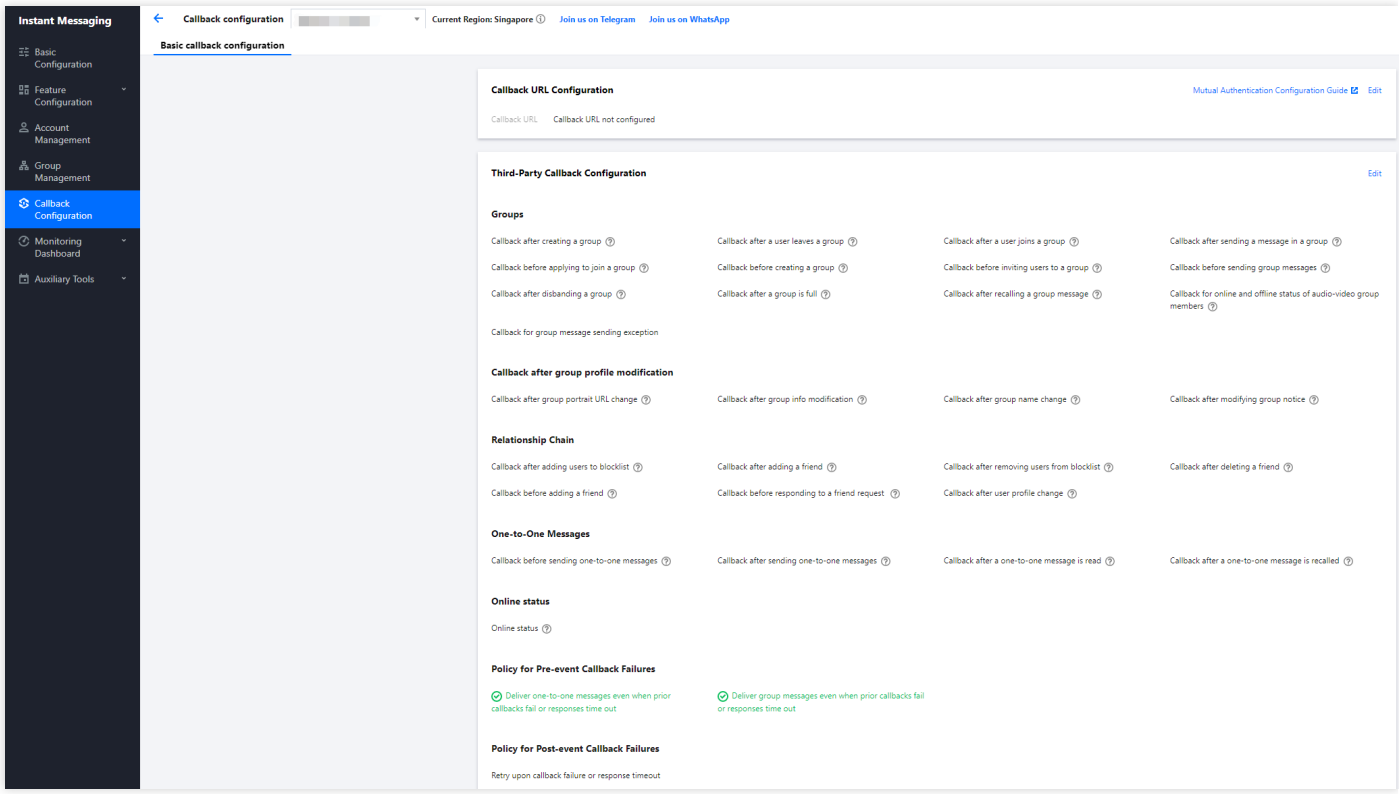
- **Key로 UserSig 계산하기**
  IM 계정 시스템에서 사용자 로그인에 필요한 암호는 IM에서 제공한 키를 사용하여 서버에서 계산합니다. 자세한 내용은 Generating UserSig를 참고하십시오. 개발 단계에서 클라이언트의 개발 지연을 방지하기 위해 아래와 같이

콘솔에서 UserSig 계산할 수도 있습니다.



- **관리자 계정 구성**
  라이브 스트리밍 중에 관리자는 라이브 룸에 메시지를 보내거나 정책을 준수하지 않는 사용자를 음소거(강제 퇴장)해야 할 수 있습니다. 이 작업은 RESTful APIs를 통해 수행할 수 있습니다. 이러한 API를 호출하려면 IM 관리자 계정을 생성해야 합니다. 기본적으로 IM은 UserID가 administrator인 계정을 제공합니다. 필요에 따라 여러 관리자 계정을 만들 수도 있으며, 최대 5개의 관리자 계정을 만들 수 있습니다.
- **콜백 주소 구성 및 콜백 활성화**
  라이브 룸에서 화면 댓글을 기반으로한 경품 추첨, 메시지 통계 수집, 민감한 콘텐츠 감지 등 요구 사항을 구현하려면 IM 백엔드가 특정 시나리오에서 비즈니스 백엔드를 다시 호출하는 IM 콜백 모듈을 사용해야 합니다. HTTP API

를 제공하고 아래와 같이 콘솔 > 콜백 구성 모듈에서 구성하기만 하면 됩니다.



3단계: 클라이언트 SDK 통합

준비가 끝나면 IM 및 TRTC 클라이언트 SDK를 프로젝트에 통합해야 합니다. 필요에 따라 다른 통합 옵션을 선택할 수 있습니다. 자세한 지침은 시작하기를 참고하십시오.
다음은 라이브 룸의 일반적인 기능을 설명하고 구현 코드와 함께 모범 사례를 제공합니다.

4단계: 주요 라이브 룸 기능 개발

1. **그룹 유형 선택**
   라이브 스트리밍 시나리오의 사용자 채팅 섹션에는 다음과 같은 특징이 있습니다.
   - 사용자가 자주 그룹에 가입하고 탈퇴하며 그룹 대화 정보(읽지 않은 횟수 및 lastMessage)를 관리할 필요가 없습니다.
   - 사용자는 승인 없이 그룹에 가입할 수 있습니다.
   - 사용자는 채팅 기록에 신경 쓰지 않고 메시지를 보냅니다.
   - 일반적으로 많은 수의 그룹 구성원이 있습니다.
   - 그룹 구성원 정보는 저장할 필요가 없습니다.

따라서 그룹 시스템에서 설명한 대로 IM의 그룹 시스템에 따라 라이브 룸의 그룹 유형으로 AVChatRoom을 선택할 수 있습니다. IM 라이브 방송 그룹(AVChatRoom)에는 다음과 같은 특성이 있습니다.

- **인원 제한이 없으며, 천만 규모의 인터랙티브 라이브 방송 시나리오를 구현할 수 있습니다.**

- 모든 온라인 사용자 대상 푸시 메시지(그룹 시스템 알림)를 지원합니다.
- 그룹 참여 신청 후 관리자의 승인 없이 바로 참여할 수 있습니다.

> 설명 :
> Web용 IM SDK를 사용하면 사용자가 한 번에 하나의 오디오/비디오 그룹(AVChatRoom)에만 가입할 수 있습니다. 사용자가 클라이언트에 로그인하여 라이브 룸 A에 입장하고 콘솔에서 멀티 클라이언트 로그인이 활성화된 상태에서 다른 클라이언트에 로그인하여 라이브 룸 B에 입장하면 해당 사용자는 라이브 룸 A에서 제거됩니다.

2. 라이브 룸 화면 댓글, 선물하기, 좋아요 구성

- **화면 댓글**
  AVChatRoom(오디오/비디오 그룹)은 친근한 인터랙션 경험을 구축하기 위해 화면 댓글, 선물하기 및 좋아요 등 다양한 메시지를 지원합니다.
  화면 댓글을 작성하려면 IM API를 사용하여 텍스트 또는 사용자 정의 메시지를 작성할 수 있습니다. 메시지가 성공적으로 전송된 후 OnRecvNewMessage() 콜백을 수신하여 라이브 룸에서 텍스트 또는 사용자 지정 속성을 가져온 다음 원하는 UI에 표시합니다.
- **선물하기**
  - 클라이언트의 비영구적 연결 요청은 과금 로직과 관련된 비즈니스 서버로 전송됩니다.
  - 요금이 발생한 후 발신자는 XXX가 XXX 선물을 보낸 것을 볼 수 있습니다. (보내는 사람이 자신이 보낸 선물을 볼 수 있도록, 메시지가 많은 경우 메시지 폐기 정책이 트리거될 수 있음)
  - 요금 정산 후 서버 API를 호출하여 사용자 정의 메시지(선물)를 보낼 수 있습니다.
  - 여러 개의 선물을 연속으로 보낼 경우 메시지를 병합해야 합니다.
    - 99와 같이 미리 선물 개수를 선택하면 매개변수에 99가 포함된 메시지를 보낼 수 있습니다.
      - 선물을 여러 번 보내고 총 개수가 불확실한 경우 선물 20개마다 메시지를 보내거나(값 조정 가능) 1초 이내에 클릭할 수 있습니다. 예를 들어 99개의 선물을 연속으로 클릭하면 최적화 후 5개의 메시지만 보내면 됩니다.
- **좋아요**
  - 선물 메시지와 달리 좋아요 메시지는 청구되지 않으며 고객에게 직접 전송됩니다.
  - 서버에서 집계해야 하는 좋아요 메시지의 경우 클라이언트에서 트래픽 스로틀링이 수행된 후 클라이언트에서 좋아요가 집계되고 짧은 시간 동안의 좋아요 메시지가 하나로 병합됩니다. 비즈니스 서버는 메시지를 보내기 전에 콜백에서 좋아요 수를 얻습니다.
  - 계산할 필요가 없는 유사 메시지의 경우 2단계의 로직이 사용됩니다. 여기서 비즈니스 서버는 클라이언트에서 트래픽 스로틀링이 수행된 후 메시지를 보내고 메시지를 보내기 전에 콜백에서 카운트를 가져올 필요가 없습니다.

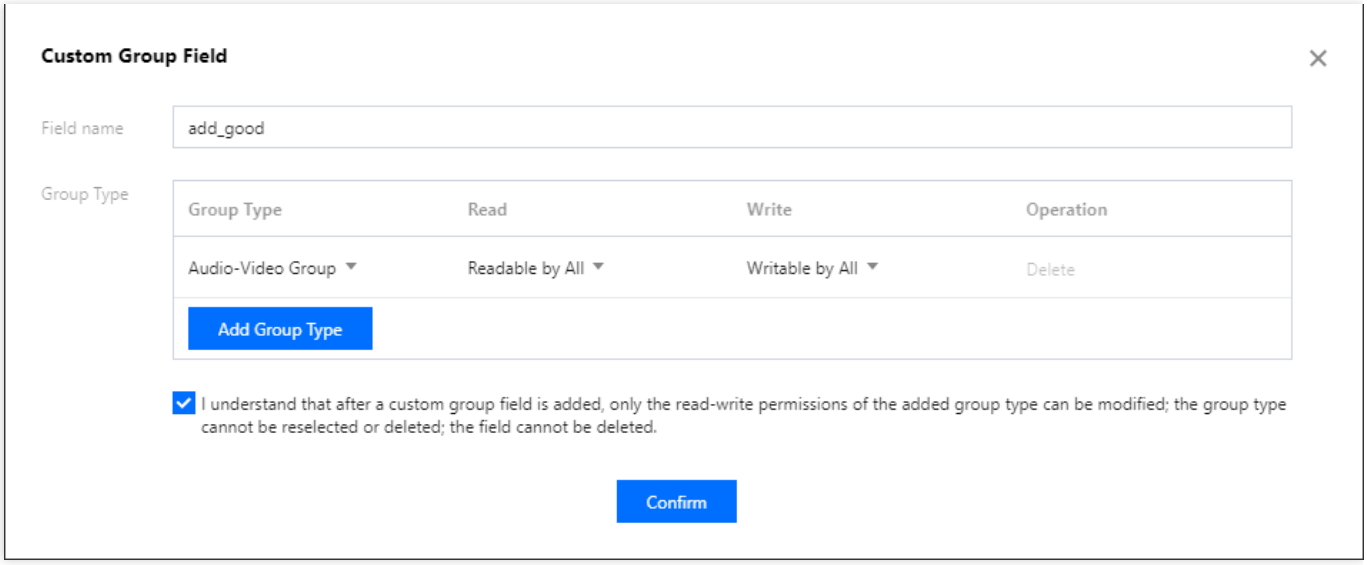
3. **라이브 커머스에서 상품 푸시**

호스트가 제품을 추천하면 화면 하단의 아이템 슬롯에 즉시 표시되고 모든 채팅방 안의 사용자에게 알려야 합니다. 신상품 알림은 일반적으로 가상 어시스턴트에 의해 트리거됩니다. 관리자가 다음과 같이 사용자 정의 그룹 필드를 수정할 수 있도록 하여 새 제품에 대한 알림을 구현하는 것이 좋습니다.

i. **사용자 정의 그룹 필드 추가**

a. IM 콘솔에 로그인하여 대상 애플리케이션 카드를 클릭하고 왼쪽 사이드바에서 **기능 구성** > **그룹 구성**을 선택합니다.
b. **사용자 지정 그룹 필드** 페이지에서 ** 오른쪽 상단 모서리에 있는 **사용자 지정 그룹 필드 추가**를 클릭합니다.
c. 그룹 레벨 사용자 정의 필드 팝업 창에서 필드 이름을 입력하고 그룹 유형과 해당 읽기 및 쓰기 권한을 설정합니다.

> 설명 :
>
> - 필드 이름은 문자, 숫자, 언더바(_)로만 구성할 수 있으며, 숫자로 시작할 수 없고, 길이는 16자를 초과할 수 없습니다.
> - 그룹 사용자 정의 필드 이름은 그룹 구성원 사용자 정의 필드 이름과 같을 수 없습니다.



d. **"사용자 지정 그룹 필드"를 추가한 후에는 추가된 그룹 유형의 읽기/쓰기 권한만 수정할 수 있으며 그룹 유형을 다시 선택하거나 삭제할 수 없음을 알고 있습니다**를 선택합니다. 이 필드는 삭제할 수 없습니다. **확인**을 클릭합니다.

> 설명 :

사용자 지정 그룹 필드는 구성 후 약 10분 후에 적용됩니다.

ii. **사용자 지정 그룹 필드 사용**

어시스턴트가 그룹 관리자 신분으로 Modifying the Profile of a Group REST API를 호출하여 그룹 사용자 정의 필드를 업데이트하고, 라이브 룸의 신상품 알림 및 라이브 방송 상태 변경 알림을 구현할 수 있습니다.

4. **라이브 룸에서 방송**

방송 기능은 라이브 룸의 시스템 알림 기능과 유사하지만 메시징에 속한다는 점에서 후자와 다릅니다. 시스템 관리자가 방송 메시지를 전달하면 SDKAppID 아래의 모든 라이브 룸이 이를 수신합니다.
방송 기능은 현재 플래그십 버전에서만 사용할 수 있으며 콘솔에서 활성화해야 합니다. 비즈니스 백엔드에서 방송 메시지를 보내는 방법에 대한 자세한 내용은 Broadcast Message of Audio-Video Group을 참고하십시오.

설명：
플래그십 버전 사용자가 아닌 경우 서버에서 사용자 정의 그룹 메시지를 보내 기능을 구현할 수 있습니다.

관련 문서

사용자 ID, 사용자 레벨, 기록 메시지, 온라인 사용자 수 표시와 같은 더 많은 라이브 룸 기능을 구현하려면 라이브 룸 구축 가이드를 참고하십시오.