

Tencent Real-Time Communication

高度な機能

製品ドキュメント





Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice

🔗 Tencent Cloud

All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

カタログ:

高度な機能

オーディオビデオストリームをライブストリーミングCDNに公開 高度な権限制御の起動 ハードウェアデバイステスト Android&iOS&Windows&Mac Web ネットワーク品質テスト Android&iOS&Windows&Mac Web TRTCクラウドレコーディングの説明 カスタムキャプチャとレンダリング Android&iOS&Windows&Mac Web カスタムオーディオのキャプチャとレンダリング Android&iOS&Windows&Mac Web メッセージの送受信 サーバーイベントコールバックの監視 サーバーイベントコールバックの監視 アクセス管理 概要 承認されるリソースおよび操作 プリセットポリシー カスタムポリシー

高度な機能

オーディオビデオストリームをライブスト リーミングCDNに公開

最終更新日:::2024-07-19 15:29:07

本文書では、TRTCルームのオーディオビデオストリームをライブストリーミングCDNに公開(「転送/プッシュ」とも呼ばれる)して、互換性のある通常のライブプレーヤーで視聴する方法について説明します。

現在のユーザーのオーディオストリームとビデオストリームをラ イブストリーミングCDNに公開します



機能説明

TRTCCloudが提供するインターフェース**startPublishMediaStream**を使用することで、ルームにおける現在の ユーザーのオーディオビデオストリームをライブストリーミングCDNに公開できます(「CDNへの公開」は 「CDNへの転送/プッシュ」とも呼ばれます)。

このプロセスでは、TRTCのクラウドサーバーはオーディオビデオデータに対して二次トランスコーディング処理 を実行するのではなく、オーディオビデオデータをライブストリーミングCDNサーバーに直接インポートするた め、プロセス全体のコストが最も低くなります。

ただし、ルーム内の各オーディオビデオユーザーは、対応するCDNライブストリームを持っている必要がありま す。そのため、ルームに複数のユーザーのオーディオビデオストリームがある場合、視聴者は複数のライブプレー ヤーを使用して視聴する必要があります。また、再生の進捗状況は、CDNライブストリーム間で大きく異なる可 能性があります。

操作ガイド

以下のガイドに従って、ルームにいる現在のユーザーのオーディオビデオストリームをライブストリーミング CDNに公開することができます。

1. TRTCPublishTargetオブジェクトを作成し、TRTCPublishTargetオブジェクトのmodeパラメータ

に TRTCPublishBigStreamToCdn または TRTCPublishSubStreamToCdn を指定します。前者は現在の ユーザーのメインチャネル画面(通常はカメラ)の公開、後者は現在のユーザーのサブチャネル画面(通常は画 面共有)の公開を指します。

 2. TRTCPublishTargetオブジェクトのcdnUrlListパラメータに1つまたは複数のCDNプッシュアドレスを指定します(標準CDNプッシュアドレスは通常、 rtmp:// をURLプレフィックスとして使用します)。指定したurlが Tencent CloudのライブストリーミングCDNのプッシュアドレスである場合は、isInternalLineにtrueを設定する必要があります。そうでない場合は、falseを設定します。

3. このモードでは、トランスコーディングサービスが提供されていないため、インターフェースを呼び出す際 に、 TRTCStreamEncoderParam パラメータと TRTCStreamMixingConfig パラメータを指定しないでくだ さい。

4. startPublishMediaStreamインターフェースを呼び出し、onStartPublishMediaStreamを介してローカルAPIの呼び出しに成功したかどうかを監視します。成功した場合、onStartPublishMediaStreamのtaskIdは空白でない文字列を返します。

5. 公開を停止する必要がある場合は、stopPublishMediaStreamを呼び出し、onStartPublishMediaStreamを介して 取得したtaskIdを渡してください。

参照コード

以下のコードは、現在のユーザーのオーディオビデオストリームをライブストリーミングCDNに公開する機能を 実装します。

Java

ObjC

C++



• • •

// 現在のユーザーのオーディオビデオストリームをライブストリーミングCDNに公開します
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishBigStream_ToCdn;

```
TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);
```

mTRTCCloud.startPublishMediaStream(target, null, null);







. . .

```
// 現在のユーザーのオーディオビデオストリームをライブストリーミングCDNに公開します
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishBigStreamToCdn;
```

```
TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;
```



```
NSMutableArray* cdnUrlList = [NSMutableArray new];
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;
```

[_trtcCloud startPublishMediaStream:target encoderParam:nil mixingConfig:nil];



. . . // 現在のユーザーのオーディオビデオストリームをライブストリーミングCDNに公開します

🔗 Tencent Cloud

```
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishBigStreamToCdn;
TRTCPublishCdnUrl* cdn_url_list = new TRTCPublishCdnUrl[1];
cdn_url_list[0].rtmpUrl = "rtmp://tencent/live/bestnews";
cdn_url_list[0].isInternalLine = true;
target.cdnUrlList = cdn_url_list;
target.cdnUrlListSize = 1;
trtc->startPublishMediaStream(&target, nullptr, nullptr);
delete[] cdn_url_list;
....
```

合成済みのオーディオビデオストリームをライブストリーミング CDNに公開します



機能の説明

TRTCルーム内の複数のユーザーのオーディオビデオストリームを1つのチャネルに合成し、合成後のオーディオ ビデオストリームをライブストリーミングCDNに公開する場合は、startPublishMediaStreamインターフェース を呼び出して、ミクスストリーミングとトランスコーディングを制御する TRTCStreamEncoderParam パラ メータと TRTCStreamMixingConfig パラメータを指定します。

1つのオーディオビデオストリームを合成してライブストリーミングCDNに公開するには、TRTCの複数のチャネ ルのオーディオビデオストリームをクラウドでデコードしてから、指定したミクスストリーミングパラメータ (TRTCStreamMixingConfig)によって合成を実行し、最後に指定したパラメータ

(TRTCStreamEncoderParam)によって再エンコードする必要があるため、このモードでの転送/プッシュサー ビスはトランスコーディング料金が請求されます。

操作ガイド

以下のガイドに従って、ルームにいる複数のユーザーのオーディオビデオストリームを合成してライブストリー ミングCDNに公開することができます。

1.TRTCPublishTargetオブジェクトを作成し、TRTCPublishTargetオブジェクトのmodeパラメータ

に TRTCPublishMixStreamToCdn を指定します。

2.TRTCPublishTargetオブジェクトのcdnUrlListパラメータに1つまたは複数のCDNプッシュアドレスを指定します (標準CDNプッシュアドレスは通常、 rtmp:// をURLプレフィックスとして使用します)。指定したurlが

Tencent CloudのライブストリーミングCDNのプッシュアドレスである場合は、isInternalLineにtrueを設定してく ださい。そうでない場合は、falseを設定します。

3. パラメータ TRTCStreamEncoderParam を使用して、トランスコーディングされたオーディオビデオスト リームのコーデックパラメータを設定します。

ビデオコーデックパラメータ:合成後の画面の解像度、フレームレート、ビットレート、エンコードのGOPを指定してください。そのうち、GOPに3s、FPSに15を設定することをお勧めします。ビットレートと解像度は対応関係があります。下表によく使用される解像度とそれに対応する推奨ビットレートを示します。

オーディオコーデックパラメータ:合成後のオーディオのコーデックフォーマット、コーデックビットレート、サ ンプリングレートとサウンドチャンネル数を指定してください。このステップでは、startLocalAudioを呼び出すと きに2番目のパラメータAudioQualityに指定された音質タイプに応じて、パラメータを指定してください。

| videoEncodedWidth | videoEncodedHeight | videoEncodedFPS | videoEncodedGOP | videoEncodedKbp |
|-------------------|--------------------|-----------------|-----------------|-----------------|
| 640 | 360 | 15 | 3 | 800kbps |
| 960 | 540 | 15 | 3 | 1200kbps |
| 1280 | 720 | 15 | 3 | 1500kbps |
| 1920 | 1080 | 15 | 3 | 2500kbps |

| TRTCAudioQuality | audioEncodedSampleRate | audioEncodedChannelNum | audioEncodedKbps |
|------------------|------------------------|------------------------|------------------|
| | | | |



| TRTCAudioQualitySpeech | 48000 | 1 | 50 |
|-------------------------|-------|---|----|
| TRTCAudioQualityDefault | 48000 | 1 | 50 |
| TRTCAudioQualityMusic | 48000 | 2 | 60 |

1.パラメータ TRTCStreamMixingConfig を使用して、オーディオのミクスストリーミングパラメータと画面 レイアウトモードを設定します。

オーディオのミクスストリーミングパラメータ(audioMixUserList):デフォルトでは、空白のままでよいです。こ れは、ルーム内のすべてのオーディオが合成されることを意味します。ルーム内の一部のユーザーの音声を合成す る場合にのみ、このパラメーターを指定してください。

画面レイアウトパラメータ(videoLayoutList):画面のレイアウトは配列によって定義されます。配列における各 TRTCVideoLayoutオブジェクトは、1つのエリアの位置、サイズ、背景色などを表します。TRTCVideoLayout のfixedVideoUserフィールドを指定した場合、このlayoutオブジェクトで定義されたエリアに、常にあるユー ザーの画面が表示されます。fixedVideoUserにnullを指定した場合、このエリアに画面が表示されるが、画面に表 示されたユーザーは固定されておらず、TRTCミクスストリーミングサーバがルールに従って決めます。

例1:4名のユーザーの画面が1つの画面に合成して表示されています。1枚の画像を背景として使用しています

layout1:ユーザーjerryのカメラ画面のサイズと位置を定義します。画面サイズは640x480です。画面は中央 上部に配置されます。

、layout2、layout3、layout4:具体的なユーザーIDが指定されていないため、TRTCは、特定のルールに 従ってルームにおける3つのチャネルのユーザーの画面を選択し、これらの3つの位置に配置します。





例2:4名のユーザーのカメラ画面と1つのチャネルの画面共有の画面が1つの画面に合成して表示されています

layout1: ユーザーjerryが共有する画面のサイズと位置を定義します。画面サイズは1280x720で、埋め込み モードは黒枠が表示されることがあるFitモードで、背景の塗りつぶしは黒いです。画面は左側に配置されま す。

layout2: ユーザーjerryのカメラ画面のサイズと位置を定義します。画面サイズは300x200で、埋め込みモードはFillモードです。画面は右上に配置されます。

、layout3、layout4、layout5:具体的なユーザーIDが指定されていないため、TRTCは、特定のルールに 従ってルームにおける3つのチャネルのユーザーの画面を選択し、これらの3つの位置に配置します。





2. startPublishMediaStreamインターフェースを呼び出し、onStartPublishMediaStreamを介してローカルAPIの呼 び出しに成功したかどうかを監視します。成功した場合、onStartPublishMediaStreamのtaskIdは空白でない文字列 を返します。

6.ミクスストリーミングパラメータを変更する場合、たとえば、複数の画面のレイアウトモードを調整する場合 は、ステップ5のtaskldでupdatePublishMediaStream APIを呼び出し、新しいTRTCStreamMixingConfigパラメータ を渡してください。転送/プッシュ中にTRTCStreamEncoderParamを変更すれば、CDNプレーヤーの安定性に影響 しますので、変更しないことをお勧めします。

3. 公開アクションを停止する必要がある場合は、stopPublishMediaStreamを呼び出し、

onStartPublishMediaStreamを介して取得したtaskIdを渡してください。

参照コード

以下のコードは、ルーム内の複数のユーザーのオーディオビデオストリームを合成してライブストリーミング CDNに公開する機能を備えています。

- Java
- ObjC
- C++



• • •

// 公開モードをTRTC_PublishMixedStream_ToCdnに指定します
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishMixedStream_ToCdn;

```
// 公開するCDNのプッシュアドレスを指定します
TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);
```

```
// 合成後のオーディオビデオストリームの二次コーデックパラメータを設定します
TRTCCloudDef.TRTCStreamEncoderParam encoderParam
    = new TRTCCloudDef.TRTCStreamEncoderParam();
encoderParam.videoEncodedWidth = 1280;
encoderParam.videoEncodedHeight = 720;
encoderParam.videoEncodedFPS = 15;
encoderParam.videoEncodedGOP = 3;
encoderParam.videoEncodedKbps = 1000;
encoderParam.audioEncodedSampleRate = 48000;
encoderParam.audioEncodedChannelNum = 1;
encoderParam.audioEncodedKbps = 50;
encoderParam.audioEncodedCodecType = 0;
// 画面のレイアウトパラメータを設定します
TRTCCloudDef.TRTCStreamMixingConfig mixingConfig =
      new TRTCCloudDef.TRTCStreamMixingConfig();
TRTCCloudDef.TRTCVideoLayout layout1 = new TRTCCloudDef.TRTCVideoLayout();
layout1.zOrder = 0;
layout1.x = 0;
layout1.y = 0;
layout1.width = 720;
layout1.height = 1280;
layout1.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB;
layout1.fixedVideoUser.intRoomId = 1234;
layout1.fixedVideoUser.userId = "mike";
TRTCCloudDef.TRTCVideoLayout layout2 = new TRTCCloudDef.TRTCVideoLayout();
layout2.zOrder = 0;
layout2.x = 1300;
layout2.y = 0;
layout2.width = 300;
layout2.height = 200;
layout2.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG;
layout2.fixedVideoUser.intRoomId = 1234;
layout2.fixedVideoUser.userId = "mike";
TRTCCloudDef.TRTCVideoLayout layout3 = new TRTCCloudDef.TRTCVideoLayout();
layout3.zOrder = 0;
layout3.x = 1300;
layout3.y = 220;
layout3.width = 300;
layout3.height = 200;
layout3.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB;
layout3.fixedVideoUser = null;
mixingConfig.videoLayoutList.add(layout1);
```



. . .

```
mixingConfig.videoLayoutList.add(layout2);
mixingConfig.videoLayoutList.add(layout3);
mixingConfig.audioMixUserList = null;
// ミクスストリーミングをリクエストします
mTRTCCloud.startPublishMediaStream(target, encoderParam, mixingConfig);
```



// 公開モードにTRTCPublishMixStreamToCdnを指定します

```
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishMixStreamToCdn;
// 公開するCDNのプッシュアドレスを指定します
TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;
NSMutableArray* cdnUrlList = [NSMutableArray new];
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;
// 合成後のオーディオビデオストリームの二次コーデックパラメータを設定します
TRTCStreamEncoderParam* encoderParam = [[TRTCStreamEncoderParam alloc] init];
encoderParam.videoEncodedWidth = 1280;
encoderParam.videoEncodedHeight = 720;
encoderParam.videoEncodedFPS = 15;
encoderParam.videoEncodedGOP = 3;
encoderParam.videoEncodedKbps = 1000;
encoderParam.audioEncodedSampleRate = 48000;
encoderParam.audioEncodedChannelNum = 1;
encoderParam.audioEncodedKbps = 50;
encoderParam.audioEncodedCodecType = 0;
// 画面のレイアウトパラメータを設定します
TRTCStreamMixingConfig* config = [[TRTCStreamMixingConfig alloc] init];
NSMutableArray* videoLayoutList = [NSMutableArray new];
TRTCVideoLayout* layout1 = [[TRTCVideoLayout alloc] init];
layout1.zOrder = 0;
layout1.rect = CGRectMake(0, 0, 720, 1280);
layout1.fixedVideoStreamType = TRTCVideoStreamTypeSub;
layout1.fixedVideoUser.intRoomId = 1234;
layout1.fixedVideoUser.userId = @"mike";
TRTCVideoLayout* layout2 = [[TRTCVideoLayout alloc] init];
layout2.zOrder = 0;
layout2.rect = CGRectMake(1300, 0, 300, 200);
layout2.fixedVideoStreamType = TRTCVideoStreamTypeBig;
layout2.fixedVideoUser.intRoomId = 1234;
layout2.fixedVideoUser.userId = @"mike";
TRTCVideoLayout* layout3 = [[TRTCVideoLayout alloc] init];
layout3.zOrder = 0;
layout3.rect = CGRectMake(1300, 220, 300, 200);
layout3.fixedVideoStreamType = TRTCVideoStreamTypeSub;
layout3.fixedVideoUser = nil;
```

🕗 Tencent Cloud

```
[videoLayoutList addObject:layout1];
[videoLayoutList addObject:layout2];
[videoLayoutList addObject:layout3];
config.videoLayoutList = videoLayoutList;
config.audioMixUserList = nil;
```

// ミクスストリーミングをリクエストします

[_trtcCloud startPublishMediaStream:target encoderParam:encoderParam mixingConfig:c





```
// 公開モードにTRTCPublishMixStreamToCdnを指定します
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishMixStreamToCdn;
// 公開するCDNのプッシュアドレスを指定します
TRTCPublishCdnUrl* cdn url = new TRTCPublishCdnUrl[1];
cdn_url[0].rtmpUrl = "rtmp://tencent/live/bestnews";
cdn_url[0].isInternalLine = true;
target.cdnUrlList = cdn_url;
target.cdnUrlListSize = 1;
// 合成後のオーディオビデオストリームの二次コーデックパラメータを設定します
TRTCStreamEncoderParam encoder_param;
encoder_param.videoEncodedWidth = 1280;
encoder_param.videoEncodedHeight = 720;
encoder_param.videoEncodedFPS = 15;
encoder_param.videoEncodedGOP = 3;
encoder_param.videoEncodedKbps = 1000;
encoder_param.audioEncodedSampleRate = 48000;
encoder_param.audioEncodedChannelNum = 1;
encoder_param.audioEncodedKbps = 50;
encoder_param.audioEncodedCodecType = 0;
// 画面のレイアウトパラメータを設定します
TRTCStreamMixingConfig config;
TRTCVideoLayout* video_layout_list = new TRTCVideoLayout[3];
TRTCUser* fixedVideoUser0 = new TRTCUser();
fixedVideoUser0->intRoomId = 1234;
fixedVideoUser0->userId = "mike";
video_layout_list[0].zOrder = 0;
video_layout_list[0].rect.left = 0;
video_layout_list[0].rect.top = 0;
video_layout_list[0].rect.right = 720;
video_layout_list[0].rect.bottom = 1280;
video_layout_list[0].fixedVideoStreamType =
     TRTCVideoStreamType::TRTCVideoStreamTypeSub;
video_layout_list[0].fixedVideoUser = fixedVideoUser0;
TRTCUser* fixedVideoUser1 = new TRTCUser();
fixedVideoUser1->intRoomId = 1234;
fixedVideoUser1->userId = "mike";
video_layout_list[1].zOrder = 0;
video_layout_list[1].rect.left = 1300;
video_layout_list[1].rect.top = 0;
```

殓 Tencent Cloud

```
video_layout_list[1].rect.right = 300;
video_layout_list[1].rect.bottom = 200;
video_layout_list[1].fixedVideoStreamType =
      TRTCVideoStreamType::TRTCVideoStreamTypeBig;
video_layout_list[1].fixedVideoUser = fixedVideoUser1;
video_layout_list[2].zOrder = 0;
video layout list[2].rect.left = 1300;
video_layout_list[2].rect.top = 220;
video layout list[2].rect.right = 300;
video_layout_list[2].rect.bottom = 200;
video_layout_list[2].fixedVideoStreamType =
      TRTCVideoStreamType::TRTCVideoStreamTypeSub;
video_layout_list[2].fixedVideoUser = nullptr;
config.videoLayoutList = video_layout_list;
config.videoLayoutListSize = 3;
config.audioMixUserList = nullptr;
// ミクスストリーミングをリクエストします
trtc->startPublishMediaStream(&target, &encoder_param, &config);
delete fixedVideoUser0;
delete fixedVideoUser1;
delete[] video_layout_list;
```

よくある質問

 CDNストリームの現在のステータスを監視できますか?ステータスが異常な場合はどうすればよいですか?
 回答: onCdnStreamStateChanged コールバックを監視することで、最新のバックグラウンドタスクステー タスを取得してコールバックを更新できます。詳細については、APIドキュメントをご参照ください。

2. 単一チャネルの転送/プッシュからミクスストリーミングの転送/プッシュに切り替えるには、手動で停止してか ら転送/プッシュタスクを再作成する必要がありますか?

回答:単一チャネルによるプッシュタスクからミクスストリーミングタスクに直接切り替えることができます。単 ーチャネルによるプッシュタスク taskid に対して、 updatePublishMediaStream プッシュタスクの変更 を実行すればよいです。プッシュリンクの安定性を確保するため、単一チャネルによるプッシュからミクススト リーミングによるプッシュに切り替えると、オーディオのみ、またはビデオのみのモードに切り替えることはで きません。デフォルトでは、単一チャネルによるプッシュはオーディオビデオモードになっており、切り替え後 のミクスストリーミングもオーディオビデオモードになっている必要があります。



3. ビデオのみでのミクスストリーミングを実現するにはどうすればよいですか?

回答:ミクスストリーミングの設定を構成するときは、 TRTCStreamEncodeParam でオーディオ関連のパラ メータを設定せず、 TRTCStreamMixingConfig で audioMixUserList を空白に設定してください。

4. ミクスストリーミング画面にウォーターマークを追加できますか?

回答:はい、 TRTCStreamMixingConfig の watermarkList で設定できます。詳細については、APIド キュメントをご参照ください。

5. 教育関連の仕事をしていますので、教師のコンテンツ共有画面をミクスストリーミング、転送、プッシュする 必要があります。画面上の共有コンテンツをミクスストリーミングすることはできますか?

回答:はい、できます。サブチャネルを使用してコンテンツ共有画面をプッシュしてから、教師のカメラ画面とコ ンテンツ共有画面を同じミクスストリーミングタスクに配置することをお勧めします。ミクスストリーミングのサ ブチャネルを設定する場合

は、 TRTCVideoLayout の fixedVideoStreamType を TRTCVideoStreamTypeSub に設定すればよいで す。

6. プリセットのレイアウトモードで、オーディオストリームの合成を決定するにはどうすればよいですか? 回答:プリセットのレイアウトモードを使用すると、ミクスストリーミングのオーディオは、現在のルームのすべ てのアップストリームオーディオから最大16チャネルのオーディオを選択して合成します。

関連料金

料金の計算

Cloud MixTranscodingでは、MCUクラスタに入力されたオーディオビデオストリームをデコードしてから、再エ ンコードして出力する必要があるため、追加のサービス料金が発生します。TRTCは、MCUクラスタを使用して Cloud MixTranscodingを行ったユーザーに対し、追加の付加価値料金を請求します。Cloud MixTranscodingの料金 は、トランスコーディングで入力された解像度とトランスコーディング時間に応じて請求されます。CDNへの転 送/プッシュの料金は、毎月のピーク帯域幅に応じて請求されます。詳細については、ミクスストリーミングと転 送/プッシュの課金説明をご参照ください。

料金の節約

クライアントのSDK APIベースのミクスストリーミングスキームで、バックエンドのミクスストリーミングタスク を停止する場合、以下のいずれかの条件を満たす必要があります:

ミクスストリーミングタスクをリクエストした(つまり、 startPublishMediaStream を呼び出した)キャ スターがルームから退出したこと

stopPublishMediaStream を呼び出して、自らミクスストリーミングを停止したこと

その他の場合、TRTC Cloudは、ミクスストリーミングの状態を継続して維持するために最善を尽くします。従っ て、予期せぬミクスストリーミング料金が発生しないよう、ミクスストリーミングが不要な場合は、上記の方法 でクラウドミクスストリーミングを可能な限り速やかに終了してください。

高度な権限制御の起動

最終更新日:::2024-07-19 15:29:07

内容紹介

あるルーム内に入室制限やマイク・オン制限を追加したい場合、つまり指定のユーザーだけにアクセスやマイ ク・オンを許可し、かつクライアントの権限を判断するに際してクラッキング攻撃に極めて遭いやすいことを懸 念する場合は、**高度な権限制御を有効にする**を検討することができます。 次のケースでは、高度な権限制御を有効にする必要はありません。 ケース1:自身がより多くの人による視聴を希望し、入室権限の制御を必要としない場合。 ケース2:攻撃者のクラッキングに対してクライアントの防御ニーズが差し迫っていない場合。 次のケースでは、セキュリティを強化するため、高度な権限制御を有効にすることを推奨します。 ケース1:セキュリティに対する要求の高いビデオ通話または音声通話シーン。 ケース2:ルームごとに異なる入室権限を設定するケース。 ケース3:視聴者のマイク・オンに対して権限制御のあるケース。

サポートするプラットフォーム

| iOS | Android | Mac OS | Windows | Electron | Web端末 |
|-----|---------|--------|---------|----------|-------|
| 1 | 1 | 1 | 1 | 1 | 1 |

高度な権限制御の原理

高度な権限制御を有効にした後、TRTCのバックエンドサービスシステムはUserSigの「入室チケット」を検証す るのみならず、 PrivateMapKey と呼ばれる「権限チケット」を検証することができるようになります。権限チ ケットには、暗号化されたroomidと暗号化された「権限順位リスト」が含まれます。

PrivateMapKeyにroomidが含まれることから、ユーザーが UserSig のみを提供し、PrivateMapKeyを提供しない場 合は、指定のルームに入室することができません。

PrivateMapKeyに含まれる「権限順位リスト」には、1byte中の8ビットが使用され、それぞれこのチケットを所持 するユーザーが、このチケットで指定されたルーム内で持つ8種の特定の機能権限を表します。

| ビット数 | バイナリー表示 | 10進数 | 権限の定義 |
|-------|-----------|------|------------|
| 第1ビット | 0000 0001 | 1 | ルームを作成する権限 |
| | | | |

| 第 2 ビット | 0000 0010 | 2 | ルームに入室する権限 |
|----------------|-----------|-----|-------------------------|
| 第 3 ビット | 0000 0100 | 4 | 音声を送信する権限 |
| 第4ビット | 0000 1000 | 8 | 音声を受信する権限 |
| 第5ビット | 0001 0000 | 16 | ビデオを送信する権限 |
| 第6ビット | 0010 0000 | 32 | ビデオを受信する権限 |
| 第7ビット | 0100 0000 | 64 | サブストリーム(画面共有)ビデオを送信する権限 |
| 第8ビット | 1000 0000 | 128 | サブストリーム(画面共有)ビデオを受信する権限 |

高度な権限制御の起動

手順1:TRTCコンソールでの高度な権限制御の有効化

1. Tencent CloudのTRTCコンソールで左側のアプリケーション管理をクリックします。

2. 右側のアプリケーションリストから高度な権限制御を有効にしたいアプリケーションをクリックし、機能設定 ボタンをクリックします。

3. 「機能設定」画面の中で**高度な権限制御のオン**のボタンを押し、**OK**をクリックすれば、高度な権限制御を有効 化できます。

| Application Info | Function Configuration | Material Management | Quick Start |
|----------------------|--|---|-------------|
| | | | |
| Application Info | | | Edit |
| Application Name | | | |
| SDKAppID | G | | |
| | SDKAppID is the unique ide is required when calling the | entifier of the application. This e voice API. | parameter |
| Creation Time | 2020-06-08 17:17:36 | | |
| Description | | | |
| Start permission key | Permission key 🛽 | | |
| Application Source | Auto-created when activati | ng TRTC service in the IM cons | ole |

ご注意:

あるSDKAppidの高度な権限制御を有効にした後、そのSDKAppidを使用するすべてのユーザーが入室に成功(手順2で説明)するためには、TRTCParamsに privateMapKey パラメータを渡す必要があります。オンライン

でこのSDKAppidを使用するユーザーである場合は、この機能を不用意に有効にしないでください。

手順2:お客様のサーバー上でPrivateMapKeyを計算します

PrivateMapKeyの価値はクライアントが逆方向のクラッキングを受けることで「非会員が高レベルのルームに侵入 できる」クラッキングバージョンが出現することを防止する点にあるため、自身のサーバーで計算して自身のApp に返されることにのみ適用し、自身のAppで直接計算することは絶対にありません。

弊社は、Java、GO、PHP、Node.js、Python、C#およびC++バージョンのPrivateMapKey計算コードを提供しております。お客様ご自身で直接ダウンロードしていただき、サーバーに統合することができます。

| 言語バージョ ン | 主な関数 | ダウンロード リンク |
|-------------|---|---------------|
| Java | genPrivateMapKey およ び genPrivateMapKeyWithStringRoomID | Github |
| GO | GenPrivateMapKey およ び GenPrivateMapKeyWithStringRoomID | Github |
| PHP | genPrivateMapKey およ び genPrivateMapKeyWithStringRoomID | Github |
| Node.js | genPrivateMapKey およ び genPrivateMapKeyWithStringRoomID | Github |
| Python | genPrivateMapKey およ び genPrivateMapKeyWithStringRoomID | Github |
| C# | genPrivateMapKey およ び genPrivateMapKeyWithStringRoomID | Github |
| C++ | genPrivateMapKey と genPrivateMapKeyWithStringRoomID | GitHub |

手順3:お客様のサーバーからPrivateMapKeyをお客様のAppに転送します





上の図に示すとおり、お客様のサーバーでPrivateMapKeyを計算した後、お客様のAppに配布することができ、お 客様のAppは2種の方法でPrivateMapKeyをSDKに配信することができます。

方法一:enterRoom時にSDKに配信します

ユーザーが入室する権限を制御したい場合は、 TRTCCloud の enterRoom インターフェースを呼び出す時 に、TRTCParamsのprivateMapKeyパラメータを設定することで、権限の制御を実現することができます。 入室時にPrivateMapKeyを検証するこの方法は比較的簡単で、ユーザーが入室する前にユーザーの権限を確認し明 確にするシーンに適しています。

方法二:試験的インターフェースを介してSDKで更新します

ライブストリーミングシーンでは、視聴者のマイク・オンがキャスターのマイク接続に変更されるシーンがよく 見受けられます。視聴者がキャスターとなる場合、TRTCは入室時の入室パラメータ TRTCParams 中の PrivateMapKeyをもう一度検証し、PrivateMapKeyの有効期限が「5分間」など比較的短く設定されている場合 は、検証エラーがトリガーされやすく、ユーザーがルームから退出させられる事態を招くことがあります。 この問題は、有効期限を延長する(「5分間」を「6時間」に変更するなど)以外にも、視聴者 が switchRole を介して自身のIDをキャスターに切り替える前に、お客様のサーバーにprivateMapKeyを再申請 し、SDKの試験的インターフェース updatePrivateMapKey を呼び出して、SDKでその更新を行うことにより 解決することができます。サンプルコードは次のとおりです。

Android

iOS

C++

C#



```
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("api", "updatePrivateMapKey");
    JSONObject params = new JSONObject();
    params.put("privateMapKey", "xxxxx"); // 新しい privateMapKeyを記入
    jsonObject.put("params", params);
    mTRTCCloud.callExperimentalAPI(jsonObject.toString());
} catch (JSONException e) {
    e.printStackTrace();
}
```





NSMutableDictionary *params = [[NSMutableDictionary alloc] init]; [params setObject:@"xxxxx" forKey:@"privateMapKey"]; // 新しいprivateMapKeyを入力 NSDictionary *dic = @{@"api": @"updatePrivateMapKey", @"params": params}; NSData *jsonData = [NSJSONSerialization dataWithJSONObject:dic options:0 error:NULL NSString *jsonStr = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEn [WXTRTCCloud sharedInstance] callExperimentalAPI:jsonStr];



std::string api = "{\\"api\\":\\"updatePrivateMapKey\\",\\"params\\":{\\"privateMap TRTCCloudCore::GetInstance()->getTRTCCloud()->callExperimentalAPI(api.c_str());



std::string api = "{\\"api\\":\\"updatePrivateMapKey\\",\\"params\\":{\\"privateMap mTRTCCloud.callExperimentalAPI(api);

よくあるご質問

1. オンラインのルームに入れないのはなぜですか。

ルーム権限制御を有効にしてしまうと、現在のSDKAppidでルームに入室するためには TRTCParams において privateMapKeyを設定する必要がありますので、オンラインサービスが稼働中で、オンラインバージョンに privateMapKey関連ロジックが追加されていない場合は、この権限制御を有効にしないでください。

2. PrivateMapKeyとUserSigには、どのような違いがあるのですか。

UserSigはTRTCParamsの入力必須項目であり、攻撃者がお客様のSDKAppidアカウント内のトラフィックを盗用 することを防止するため、現在のユーザーがTRTC クラウドサービスを使用する権限を持っているかどうかを検証 するために使用されます。

PrivateMapKeyはTRTCParamsの非必須項目であり、現在のユーザーが指定されたroomidのルームに入室する権限 およびこのユーザーがこのルームで持つことができる権限を持っているかどうかを検証するために使用され、ビ ジネスでユーザーを識別する必要がある場合に限り、有効化する必要があります。

ハードウェアデバイステスト

Android&iOS&Windows&Mac

最終更新日:::2024-07-19 15:29:07

内容紹介

ビデオ通話の前に、カメラおよびマイクなどのデバイスのテストを先に行うことを推奨します。テストしないと、 ユーザーが実際にビデオ通話を行うときにデバイスの問題を見つけることが難しくなります。

この機能のプラットフォームのサポート

| iOS | Android | Mac OS | Windows | Electron | Web端末 |
|-----|---------|--------------|---------|--------------|-----------|
| × | × | \checkmark | 1 | \checkmark | ✔ (Web 端) |

カメラテスト

TRTCCloudの startCameraDeviceTestInView インターフェースを使用すればカメラテストが行えます。テ ストのプロセスでは setCurrentCameraDevice 関数をコールすることでカメラを切り替えられます。 MacプラットフォームObjective-C Windowsプラットフォーム (C++) Windowsプラットフォーム (C#)



```
// カメラテストインターフェースの表示(カメラのプレビュー、カメラの切り替えのサポート)
- (IBAction)startCameraTest:(id)sender {
    // カメラテストの開始。cameraPreviewをmacOSのNSViewまたはiOSプラットフォームのUIView
    [self.trtcCloud startCameraDeviceTestInView:self.cameraPreview];
}
//カメラテストインターフェースの停止
- (void)windowWillClose:(NSNotification *)notification{
    // カメラテストの終了
    [self.trtcCloud stopCameraDeviceTest];
}
```





```
// カメラテストの開始。レンダリングする必要のあるビデオの制御ハンドルを渡します。
void TRTCMainViewController::startTestCameraDevice(HWND hwnd)
{
    trtcCloud->startCameraDeviceTest(hwnd);
}
// カメラテストの停止
void TRTCMainViewController::stopTestCameraDevice()
{
```





```
// カメラテストの開始。レンダリングする必要のあるビデオの制御ハンドルを渡します。
private void startTestCameraDevice(Intptr hwnd)
{
    mTRTCCloud.startCameraDeviceTest(hwnd);
}
// カメラテストの停止
```

```
private void stopTestCameraDevice()
{
    mTRTCCloud.stopCameraDeviceTest();
}
```

マイクテスト

TRTCCloudの startMicDeviceTest 関数を使用すると、マイクの音量を測定でき、コールバック関数はリア ルタイムでマイク音量値を返します。 MacプラットフォームObjective-C Windowsプラットフォーム (C++) Windowsプラットフォーム (C#)


```
}];
btn.title = @"テストの停止";
}
else{
    //マイクテストの終了
    [self.trtcCloud stopMicDeviceTest];
    [self _updateInputVolume:0];
    btn.title = @"テスト開始";
}
```



```
// マイクテストサンプルコード
void TRTCMainViewController::startTestMicDevice()
{
    // 音量コールバック率の設定。ここでは500msに1回コールバック。 onTestMicVolumeコールバック
    uint32_t interval = 500;
    // マイクテストの開始
    trtcCloud->startMicDeviceTest(interval);
}
// マイクテストの終了
void TRTCMainViewController::stopTestMicDevice()
{
    trtcCloud->stopMicDeviceTest();
}
```



```
// マイクテストサンプルコード
private void startTestMicDevice()
{
    // 音量コールバック率の設定。ここでは500msに1回コールバック。 onTestMicVolumeコールバック・
    uint interval = 500;
    // マイクテストの開始
    mTRTCCloud.startMicDeviceTest(interval);
}
// マイクテストの終了
private void stopTestMicDevice()
```

{

```
mTRTCCloud.stopMicDeviceTest();
}
```

スピーカーテスト

TRTCCloudの startSpeakerDeviceTest 関数を使用し、デフォルトのmp3オーディオデータを再生すること で、スピーカーが正常に動作しているかテストします。 MacプラットフォームObjective-C Windowsプラットフォーム(C++)

Windowsプラットフォーム (C#)





```
[wself _updateOutputVolume:volume];
               if (playFinished) {
                  // 再生完了時にはボタンのステータスを「テスト開始」にします
                  sender.state = NSControlStateValueOff;
               }
           });
       }];
   } else {
       // 「テスト終了」をクリック
       [self.trtcEngine stopSpeakerDeviceTest];
       [self _updateOutputVolume:0];
   }
}
// スピーカー音量インジケータの更新
- (void)_updateOutputVolume:(NSInteger)volume {
   // speakerVolumeMeterはNSLevelIndicatorです
   self.speakerVolumeMeter.doubleValue = volume / 255.0 * 10;
}
```



```
// スピーカーテストサンプルコード
void TRTCMainViewController::startTestSpeakerDevice(std::string testAudioFilePath)
{
    // testAudioFilePathオーディオファイルの絶対パス。パス文字列にはUTF-8エンコードフォーマッ
    // onTestSpeakerVolumeコールバックインターフェースからスピーカーテスト音量値をモニタします。
    trtcCloud->startSpeakerDeviceTest(testAudioFilePath.c_str());
}
// スピーカーテストを終了
void TRTCMainViewController::stopTestSpeakerDevice() {
    trtcCloud->stopSpeakerDeviceTest();
```





```
// スピーカーテストサンプルコード
private void startTestSpeakerDevice(string testAudioFilePath)
{
    // testAudioFilePathオーディオファイルの絶対パス。パス文字列にはUTF-8エンコードフォーマッ
    // onTestSpeakerVolumeコールバックインターフェースからスピーカーテスト音量値をモニタします。
    mTRTCCloud.startSpeakerDeviceTest(testAudioFilePath);
}
```

```
// スピーカーテストを終了
private void stopTestSpeakerDevice() {
    mTRTCCloud.stopSpeakerDeviceTest();
}
```

Web

最終更新日:::2024-07-19 15:29:07

内容紹介

ビデオ通話の前に、ブラウザ環境の確認およびカメラとマイクなどのデバイスのテストを先に行うことをお勧め します。テストしない場合、ユーザーが実際に通話を行うときにデバイスの問題を見つけることが難しくなりま す。

ブラウザ環境の確認

SDKの通信機能を呼び出す前に、まず{@link TRTC.checkSystemRequirements checkSystemRequirements()}イン ターフェイスを使用して、SDKが現在のWebページをサポートしているかどうかを確認することをお勧めしま す。SDKが現在のブラウザをサポートしていない場合、ユーザーのデバイスタイプに応じて、SDKでサポートさ れているブラウザを使用するようにユーザーに提案してください。



```
TRTC.checkSystemRequirements().then(checkResult => {
    if (checkResult.result) {
        // 入室をサポートします
        if (checkResult.isH264DecodeSupported) {
            // プルをサポートします
        }
        if (checkResult.isH264EncodeSupported) {
            // プッシュをサポートします
        }
    })
```



ユーザーがSDKでサポートされているブラウザを使用し、 TRTC.checkSystemRequirements によって返さ れる検出結果がfalseの場合、次の理由が考えられます:

ケース1:リンクが次の3つの条件のいずれかを満たしているかどうかを確認してください

localhostドメイン(Firefoxブラウザはlocalhostとローカルipアクセスをサポートします)

HTTPSがオンになっているドメイン

file:///プロトコルで開いたローカルファイル

ケース2: Firefoxブラウザをインストールした後、H264コーデックを動的にロードする必要があるため、検出結果 はしばらくの間falseになります。しばらく待ってから再試行するか、先に別の推奨ブラウザを使用してリンクを 開いてください。

既知のブラウザの使用制限の説明

Firefox

Firefoxは30fpsのビデオフレームレートのみをサポートします。フレームレートを設定する必要がある場合は、 SDKでサポートされている他のブラウザを使用してください。

QQブラウザ

カメラとマイクが通常に動作する一部のWindowsデバイスは、localhost環境でlocalStream.initialize()を呼び出すと き、NotFoundErrorエラーをスローします。

オーディオビデオデバイスのテスト

ユーザーがTRTC-SDKを使用する過程でより良いユーザーエクスペリエンスを確実に得られるために、ユーザー がTRTCルームに参加する前に、ユーザーのデバイスとネットワーク状態を確認し、提案とガイダンスを提供する ことをお勧めします。

デバイス検出機能とネットワーク確認機能を迅速に統合できるために、参照するための以下の方法を提供しま す:

rtc-detectのJSライブラリ デバイスを検出するReactコンポーネント TRTC機能検出ページ

rtc-detectライブラリ

rtc-detectを使用してTRTC SDKに対する現在の環境のサポートおよび現在の環境の詳細を確認できます。

インストール





npm install rtc-detect

使用方法



import RTCDetect from 'rtc-detect';
// 監視モジュールを初期化します
const detect = new RTCDetect();
// 現在の環境の検視結果を取得します
const result = await detect.getReportAsync();
// resultは、現在の環境システムの情報、APIサポート、コーデックサポートおよびデバイス関連情報が含ま
console.log('result is: ' + result);

API

(async) isTRTCSupported()

現在の環境がTRTCをサポートしているかどうかを判断します。



```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();
if (data.result) {
    console.log('current browser supports TRTC.')
}else{
    console.log(`current browser does not support TRTC, reason: ${data.reason}.`)
}
```

getSystem()

現在のシステム環境パラメータを取得します。

| Item | Туре | Description |
|------------------------|--------|----------------------------|
| UA | string | ブラウザのua |
| OS | string | 現在のデバイスのシステムモデル |
| browser | object | 現在のブラウザ情報{ name, version } |
| displayResolution | object | 現在の解像度{ width, height } |
| getHardwareConcurrency | number | デバイスのCPUコアの現在の数 |



const detect = new RTCDetect(); const result = detect.getSystem();

getAPISupported()

現在の環境APIサポートを取得します。

| Item | Туре | Description |
|----------------------|---------|------------------------------------|
| isUserMediaSupported | boolean | ユーザーメディアデータストリームの取得をサポート するかどうか |



| isWebRTCSupported | boolean | WebRTCをサポートするかどうか |
|-----------------------------------|---------|--|
| isWebSocketSupported | boolean | WebSocketをサポートするかどうか |
| isWebAudioSupported | boolean | WebAudioをサポートするかどうか |
| isScreenCaptureAPISupported | boolean | 画面のストリームの取得をサポートするかどうか |
| isCanvasCapturingSupported | boolean | canvas からのデータストリームの取得をサポートする かどうか |
| isVideoCapturingSupported | boolean | video からのデータストリームの取得をサポートするか どうか |
| isRTPSenderReplaceTracksSupported | boolean | trackを置き換えるときにpeerConnectionと再ネゴシ エーションを行わないことをサポートするかどうか |
| isApplyConstraintsSupported | boolean | getUserMediaを再度呼び出さずにカメラの解像度の変 更をサポートするかどうか |





const detect = new RTCDetect(); const result = detect.getAPISupported();

(async) getDevicesAsync()

現在の環境で利用可能なデバイスを取得します。

| Item | Туре | Description |
|----------------------|---------|---------------------------|
| hasWebCamPermissions | boolean | ユーザーのカメラデータの取得をサポートするかどうか |
| | | |

| hasMicrophonePermission | boolean | ユーザーのマイクデータの取得をサポートするかどうか |
|-------------------------|---------|--|
| cameras | array | サポートされているビデオストリームの解像度情報、最大の幅と高 さ、および最大フレームレートを含むユーザーのカメラデバイスリ スト(最大フレームレートは一部のブラウザではサポートされてい ません) |
| microphones | array | ユーザーのマイクデバイスのリスト |
| speakers | array | ユーザーのスピーカーデバイスのリスト |

Cameraltem

| Item | Туре | Description |
|------------|--------|--|
| deviceId | string | デバイスIDは、通常は一意であり、デバイスの収集と識別に使用できます |
| groupId | string | グループの識別子です。 2 つのデバイスが同じ物理デバイスに属する場合、それらは 同じ識別子を持ちます |
| kind | string | カメラデバイスのタイプ:'videoinput' |
| label | string | デバイスを説明するタグ |
| resolution | object | カメラでサポートされている最大解像度の幅と高さ、フレームレート{maxWidth: 1280, maxHeight: 720, maxFrameRate: 30} |

DeviceItem

| Item | Туре | Description |
|----------|--------|---|
| deviceId | string | デバイスIDは、通常は一意であり、デバイスの収集と識別に使用できます |
| groupId | string | グループの識別子です。 2 つのデバイスが同じ物理デバイスに属する場合、それらは同 じ識別子を持ちます |
| kind | string | デバイスのタイプです。例えば:'audioinput', 'audiooutput' |
| label | string | デバイスを説明するタグ |





const detect = new RTCDetect(); const result = await detect.getDevicesAsync();

(async) getCodecAsync()

現在の環境パラメータのコーデックサポートを取得します。

| Item | Туре | Description |
|-----------------------|------|--------------------------|
| isH264EncodeSupported | | h264エンコーディン グをサポートするか |



| | | どうか |
|-----------------------|---------|--------------------------------|
| isH264DecodeSupported | boolean | h264デコーディング をサポートするかど うか |
| isVp8EncodeSupported | boolean | vp8エンコーディン グをサポートするか どうか |
| isVp8DecodeSupported | boolean | vp8デコーディング をサポートするかど うか |

エンコーディングのサポートとは、オーディオビデオのリリースをサポートすることを意味し、デコードのサ ポートとは、オーディオビデオ再生のプルをサポートすることを意味します





const detect = new RTCDetect(); const result = await detect.getCod

(async) getReportAsync()

現在の環境モニタリングレポートを取得します。

| Item | Туре | Description |
|--------|--------|-----------------------|
| system | object | getSystem()の戻り値と一致します |
| | | |

S Tencent Cloud

| APISupported | object | getAPISupported()の戻り値と一致します |
|-----------------|--------|-----------------------------|
| codecsSupported | object | getCodecAsync()の戻り値と一致します |
| devices | object | getDevicesAsync()の戻り値と一致します |



```
const detect = new RTCDetect();
const result = await detect.getReportAsync();
```

(async) isHardWareAccelerationEnabled()



Chromeブラウザでハードウェアアクセラレーションが有効になっているかどうかを確認します。 ご注意:

このインターフェースの実装は、WebRTCネイティブインターフェースに依存します。isTRTCSupportedがサ ポートを確認した後、このインターフェースを呼び出して検出することをお勧めします。最大検出時間は30sで す。テストにより、

ハードウェアアクセラレーションを有効にすると、このインターフェイスはWindowsで約2秒、Macで約10秒かか ります。

ハードウェアアクセラレーションを無効にすると、このインターフェイスはWindowsとMacの両方で約30秒かか ります。



🔗 Tencent Cloud

```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();
if (data.result) {
   const result = await detect.isHardWareAccelerationEnabled();
   console.log(`is hardware acceleration enabled: ${result}`);
}else{
   console.log(`current browser does not support TRTC, reason: ${data.reason}.`)
}
```

デバイス検出用のReactコンポーネント

デバイス検出UIコンポーネントの機能

デバイス接続とデバイス検出のロジックを処理します
 ネットワーク検出のロジックを処理します
 ネットワーク検出タブはオプションです
 中国語と英語の両方をサポートします

デバイス検出UIコンポーネントの関連リンク

コンポーネントnpmパッケージの使用手順については、rtc-device-detector-reactをご参照ください コンポーネントのソースコードのデバッグについては、github/rtc-device-detectorをご参照ください コンポーネントの参照例については、WebRTC API Exampleをご参照ください

デバイス検出UIコンポーネントのインターフェース











| <u>@</u> — | 👱 | ()» | |
|------------|--|--------------------------------------|-------|
| CAMERA | MICROPHONE | SPEAKER | NETWO |
| Speak | er selection | (Built-in) | ~ |
| F | Please turn up the volu click to play the | me of the device an e audio below | ıd |
| | ▶ 0:00 / 1:27 - | | |
| | Can you hear | the sound? | |
| | No | Yes | |







デバイスとネットワークの検出ロジック

1) デバイス接続

デバイス接続の目的は、ユーザーが使用する機器にカメラ、マイク、またはスピーカーデバイスがあるかどうか、 およびインターネットに接続されているかどうかを検出することです。カメラ、マイクデバイスがある場合は、 オーディオビデオストリームを取得し、カメラとマイクへのアクセスを許可するようにユーザーに提案します。 デバイスにカメラ、マイク、スピーカーデバイスがあるかどうかを判断します



```
import TRTC from 'trtc-js-sdk';
```

```
const cameraList = await TRTC.getCameras();
const micList = await TRTC.getMicrophones();
const speakerList = await TRTC.getSpeakers();
const hasCameraDevice = cameraList.length > 0;
const hasMicrophoneDevice = micList.length > 0;
const hasSpeakerDevice = speakerList.length > 0;
```

カメラとマイクへのアクセス権限を取得します



```
navigator.mediaDevices
.getUserMedia({ video: hasCameraDevice, audio: hasMicrophoneDevice })
.then((stream) => {
    // オーディオビデオストリームの取得に成功しました
    // ...
    // カメラとマイクデバイスをリリースします
    stream.getTracks().forEach(track => track.stop());
})
.catch((error) => {
    // オーディオビデオストリームの取得に失敗しました
});
```



デバイスがインターネットに接続されているかどうかを判断します



```
export function isOnline() {
  const url = 'https://web.sdk.qcloud.com/trtc/webrtc/assets/trtc-logo.png';
  return new Promise((resolve) => {
    try {
      const xhr = new XMLHttpRequest();
      xhr.onload = function () {
        resolve(true);
      };
      xhr.onerror = function () {
```
```
resolve(false);
};
xhr.open('GET', url, true);
xhr.send();
} catch (err) {
    // console.log(err);
}
});
}
const isOnline = await isOnline();
```

2) カメラ検出

カメラ検出は、選択したカメラによってキャプチャされたビデオストリームをユーザーにレンダリングし、ユー ザーがカメラが正常に使用できるかどうかを確認するのに役立ちます。 カメラリストを取得します。デフォルトでは、カメラリストの最初のデバイスを使用します



```
import TRTC from 'trtc-js-sdk';
```

```
let cameraList = await TRTC.getCameras();
let cameraId = cameraList[0].deviceId;
```

ビデオストリームを初期化し、idがcamera-videoのdom要素でストリームを再生します



```
const localStream = TRTC.createStream({
   video: true,
   audio: false,
   cameraId,
});
await localStream.initialize();
localStream.play('camera-video');
```

ユーザーがカメラデバイスを切り替えた後にストリームを更新します





localStream.switchDevice('video', cameraId);

デバイスの抜き差しを監視します



```
navigator.mediaDevices.addEventListener('devicechange', async () => {
  cameraList = await TRTC.getCameras();
  cameraId = cameraList[0].deviceId;
  localStream.switchDevice('video', cameraId);
})
```

検出が完了したら、カメラの占有をリリースします





localStream.close();

3) マイク検出

マイク検出は、選択したマイクによってキャプチャされたオーディオストリームのボリュームをユーザーにレン ダリングし、ユーザーがマイクが正常に使用できるかどうかを確認するのに役立ちます。 マイクリストを取得します。デフォルトでは、マイクリストの最初のデバイスを使用します



```
import TRTC from 'trtc-js-sdk';
```

let microphoneList = await TRTC.getMicrophones(); let microphoneId = microphoneList[0].deviceId;

オーディオストリームを初期化し、idがaudio-containerのdom要素でストリームを再生します



```
const localStream = TRTC.createStream({
   video: false,
   audio: true,
   microphoneId,
});
await localStream.initialize();
localStream.play('audio-container');
timer = setInterval(() => {
   const volume = localStream.getAudioLevel();
}, 100);
```



ユーザーがマイクデバイスを切り替えた後にストリームを更新します



// ユーザーが新しく選択したmicrophoneIdを取得します
localStream.switchDevice('audio', microphoneId);

デバイスの抜き差しを監視します



```
navigator.mediaDevices.addEventListener('devicechange', async () => {
    microphoneList = await TRTC.getMicrophones();
    microphoneId = microphoneList[0].deviceId;
    localStream.switchDevice('audio', microphoneId);
})
```

検出が完了したら、マイクの占有をリリースし、ボリュームの監視を停止します





localStream.close();
clearInterval(timer);

4) スピーカー検出

スピーカー検出はオーディオプレーヤーを提供し、ユーザーはオーディオの再生によって、選択したスピーカーが 正常に使用できるかどうかを確認できます。

mp3プレーヤーを提供し、デバイスの再生ボリュームを上げるようにユーザーに通知し、mp3を再生して、スピー カーデバイスが正常であるかどうかを確認します





<audio id="audio-player" src="xxxxx" controls></audio>

検出が終了したら、再生を停止します



```
const audioPlayer = document.getElementById('audio-player');
if (!audioPlayer.paused) {
    audioPlayer.pause();
}
audioPlayer.currentTime = 0;
```

5) ネットワーク検出

TRTC.createClientを呼び出して、それぞれuplinkClientおよびdownlinkClientという2つのClientを作成します。 この2つのClientは、同じルームに入ります。



uplinkClientを使用してプッシュし、NETWORK_QUALITYイベントを監視してアップリンクのネットワーク品質 を確認します。

downlinkClientを使用してプルし、NETWORK_QUALITYイベントを監視してダウンリンクのネットワーク品質を 確認します。

プロセス全体は約15秒間続く可能性があり、最後に平均ネットワーク品質を使用して、アップリンクとダウンリ ンクのネットワーク状態を大まかに判断します。

ご注意:

検出プロセスには少額の基本サービス料金が発生します。プッシュ解像度が指定されていない場合、デフォルトで 640*480の解像度でプッシュします。



```
let uplinkClient = null; // アップリンクネットワーク品質の確認に使用されます
let downlinkClient = null; //ダウンリンクネットワーク品質の確認に使用されます
let localStream = null; // テストされるストリームに使用されます
let testResult = {
 // アップリンクネットワーク品質のデータを記録します
 uplinkNetworkQualities: [],
 // ダウンリンクネットワーク品質のデータを記録します
 downlinkNetworkQualities: [],
 average: {
   uplinkNetworkQuality: 0,
   downlinkNetworkQuality: 0
 }
}
// 1. アップリンクネットワーク品質を確認します
async function testUplinkNetworkQuality() {
 uplinkClient = TRTC.createClient({
   sdkAppId: 0, // sdkAppIdを記入します
   userId: 'user_uplink_test',
   userSig: '', // uplink_testOuserSig
   mode: 'rtc'
 });
 localStream = TRTC.createStream({ audio: true, video: true });
 // 実際のサービスシーンに応じてvideo profileを設定します
 localStream.setVideoProfile('480p');
 await localStream.initialize();
 uplinkClient.on('network-quality', event => {
   const { uplinkNetworkQuality } = event;
   testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
 });
 // テスト用ルームを追加します。競合を避けるためにルーム番号はランダムである必要があります
 await uplinkClient.join({ roomId: 8080 });
 await uplinkClient.publish(localStream);
}
// 2. ダウンリンクネットワーク品質を確認します
async function testDownlinkNetworkQuality() {
 downlinkClient = TRTC.createClient({
   sdkAppId: 0, // sdkAppIdを記入します
   userId: 'user_downlink_test',
   userSig: '', // userSig
   mode: 'rtc'
 });
```

```
Stencent Cloud
```

```
downlinkClient.on('stream-added', async event => {
   await downlinkClient.subscribe(event.stream, { audio: true, video: true });
       // サブスクリプションに成功した後、ネットワーク品質イベントの監視を開始します
   downlinkClient.on('network-quality', event => {
     const { downlinkNetworkQuality } = event;
     testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
   });
 })
 // テスト用ルームを追加します。競合を避けるためにルーム番号はランダムである必要があります
 await downlinkClient.join({ roomId: 8080 });
}
// 3. 確認を開始します
testUplinkNetworkQuality();
testDownlinkNetworkQuality();
// 4.15秒後に確認を停止し、平均ネットワーク品質を計算します
setTimeout(() => {
 // アップリンクの平均ネットワーク品質を計算します
 if (testResult.uplinkNetworkQualities.length > 0) {
   testResult.average.uplinkNetworkQuality = Math.ceil(
     testResult.uplinkNetworkQualities.reduce((value, current) => value + current,
   );
 }
 if (testResult.downlinkNetworkQualities.length > 0) {
   // ダウンリンクの平均ネットワーク品質を計算します
   testResult.average.downlinkNetworkQuality = Math.ceil(
     testResult.downlinkNetworkQualities.reduce((value, current) => value + curren
   );
 }
 // 確認が終了し、関連する状態がクリアされます。
 uplinkClient.leave();
 downlinkClient.leave();
 localStream.close();
}, 15 * 1000);
```

TRTC機能検出ページ

現在環境の検出のために、現在、TRTCSDKを使用している場合にTRTC検出ページを使用できます。また、環境 検出またはトラブルシューティングのために、レポート生成ボタンをクリックして、現在の環境のレポートを取 得できます。

ネットワーク品質テスト

Android&iOS&Windows&Mac

最終更新日:::2024-07-19 15:29:07

一般のユーザーがネットワーク品質を評価することは困難ですので、ビデオ通話を行う前にネットワークテスト を実行することをお勧めします。速度測定はネットワーク品質をより直感的に評価することができます。

注意事項

通話品質への影響を避けるため、ビデオ通話中にテストしないでください。

速度測定は一定量のトラフィックを消費するため、ごくわずかな追加的トラフィック費用が発生します(ほぼ無視 できる程度です)。

サポートするプラットフォーム

| iOS | Android | Mac OS | Windows | Electron | Web端末 |
|-----|---------|--------|---------|----------|--------------------|
| 1 | 1 | 1 | 1 | 1 | √(参考:Web端末チュートリアル) |

速度測定の原理



速度測定の原理は次のとおりです。SDKが検出パケットのバッチをサーバーノードに送信した後、返送されたパ ケットの品質を集計し、コールバックインターフェースを介して速度測定の結果を通知します。

速度測定の結果は、SDKの次のサーバー選択戦略を最適化するために使用されます。したがって、ユーザーが最 初の通話を行う前に速度測定を実行することを推奨します。このことは最適なサーバーの選択に役立ちます。ま た極めて不満足なテスト結果であった場合は、目立つUIを使用し、より適切なネットワークを選択するよう、 ユーザーに促すことができます。

速度測定の結果(TRTCSpeedTestResult)には、次のいくつかのフィールドが含まれます。

| フィールド | 意味 | 意味の説明 | | |
|------------|---------------------|---|--|--|
| success | 成功したか どうか | 今回のテストが成功したかどうか | | |
| errMsg | エラー情報 | 帯域幅テストの詳細なエラー情報 | | |
| ip | サーバー | サーバー速度測定のIP | | |
| quality | ネットワー ク品質スコ ア | 評価アルゴリズムによって測定および計算されたネットワーク品 質。lossが低いほど、rttが小さいほど、スコアは高くなります | | |
| upLostRate | 上りパケッ ト損失率 | 範囲は[0 - 1.0]、例えば0.3はサーバーに送信される10データパ ケットごとに、途中で3パケットが失われる可能性があることを | | |



| | | 意味します |
|------------------------|---------------|--|
| downLostRate | 下りパケッ ト損失率 | 範囲は[0 - 1.0]、例えば0.2はサーバーから受信する10データパ ケットごとに、途中で2パケットが失われる可能性があることを 意味します |
| rtt | ネットワー ク遅延 | SDKとサーバー間の往復で消費される時間を表します。値が小さ いほど遅延が少なく、通常の値は10msから100msの間です |
| availableUpBandwidth | 上り帯域幅 | 予測される上り帯域幅。単位はkbps、 -1は無効値を表わします |
| availableDownBandwidth | 下り帯域幅 | 予測される下り帯域幅。単位はkbps。 -1は無効値を表わします |

速度測定方法

TRTCCloudの startSpeedTest 機能によって速度測定機能を起動できます。速度測定の結果はコールバック関数によって返されます。

Objective-C

Java

C++

C#



```
// ネットワーク速度測定を起動するためのサンプルコードとして、sdkAppIdとUserSigが必要です(取得方
// ここではログイン後の測定開始を例示します
- (void)onLogin:(NSString *)userId userSig:(NSString *)userSid
{
    TRTCSpeedTestParams *params;
    // sdkAppIDはコンソールから取得した実際のアプリケーションのAppIDです
    params.userID = userId;
    params.userSig = userSig;
    // 予想される上り帯域幅(kbps、値範囲: 10 ~ 5000、0の時はテストなし)
    params.expectedUpBandwidth = 5000;
```

```
// 予想される下り帯域幅(kbps、値範囲: 10 ~ 5000、0の時はテストなし)
params.expectedDownBandwidth = 5000;
[trtcCloud startSpeedTest:params];
}
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {
    // 速度測定完了後、測定結果がコールバックされます
}
```



//ネットワーク速度測定を起動するためのサンプルコードとして、sdkAppIdとUserSigが必要です(取得方法)// ここではログイン後の測定開始を例示します

```
public void onLogin(String userId, String userSig)
{
 TRTCCloudDef.TRTCSpeedTestParams params = new TRTCCloudDef.TRTCSpeedTestParams();
 params.sdkAppId = GenerateTestUserSig.SDKAPPID;
 params.userId = mEtUserId.getText().toString();
 params.userSig = GenerateTestUserSig.genTestUserSig(params.userId);
 params.expectedUpBandwidth = Integer.parseInt(expectUpBandwidthStr);
 params.expectedDownBandwidth = Integer.parseInt(expectDownBandwidthStr);
   // sdkAppIDはコンソールから取得した実際のアプリケーションのAppIDです
   trtcCloud.startSpeedTest(params);
}
// 速度測定の結果を監視し、TRTCCloudListenerを継承し、次のメソッドを実装します
void onSpeedTestResult(TRTCCloudDef.TRTCSpeedTestResult result)
{
   // 速度測定完了後、測定結果がコールバックされます
}
```



// ネットワーク速度測定を起動するためのサンプルコードとして、sdkAppIdとUserSigが必要です(取得方
// ここではログイン後の測定開始を例示します
void onLogin(const char* userId, const char* userSig)
{
 TRTCSpeedTestParams params;
 // sdkAppIDはコンソールから取得した実際のアプリケーションのAppIDです
 params.userId = userid;
 params.userId = userid;
 param.userSig = userSig;
 // 予想される上り帯域幅(kbps、値範囲: 10 ~ 5000、0の時はテストなし)
 param.expectedUpBandwidth = 5000;



```
// ネットワーク速度測定を起動するためのサンプルコードとして、sdkAppIdとUserSigが必要です(取得方
// ここではログイン後の測定開始を例示します
private void onLogin(string userId, string userSig)
{
    TRTCSpeedTestParams params;
    // sdkAppIDはコンソールから取得した実際のアプリケーションのAppIDです
    params.sdkAppID = sdkAppId;
    params.userId = userid;
    param.userSig = userSig;
    // 予想される上り帯域幅(kbps、値範囲: 10 ~ 5000、0の時はテストなし)
    param.expectedUpBandwidth = 5000;
```

```
// 予想される下り帯域幅(kbps、値範囲: 10 ~ 5000、0の時はテストなし)
param.expectedDownBandwidth = 5000;
mTRTCCloud.startSpeedTest(params);
}
// 速度測定結果を監視します
public void onSpeedTestResult(TRTCSpeedTestResult result)
{
    // 速度測定完了後、測定結果がコールバックされます
}
```

速度測定ツール

インターフェースを呼び出すことによるネットワーク速度測定を行いたくない場合、詳細なネットワーク品質情 報を迅速に得るために、TRTCはデスクトップ版のネットワーク速度測定ツールプログラムも提供しています。

ダウンロードリンク

Mac |Windows

テスト指標

| インジ ケータ | 意味 |
|-----------------|--|
| WiFi Quality | Wi-Fi 信号品質 |
| DNS RTT | Tencent Cloudのドメイン名解析時間の速度測定 |
| MTR | MTRはネットワークテストツールで、クライアントからTRTCノードまでのパケット損失率と レイテンシーを検出することができます。また、ルートの各ジャンプに関する具体的な情報も 確認できます |
| UDP Loss | クライアントからTRTCノードまでのUDPパケット損失率 |
| UDP RTT | クライアントからTRTCノードまでのUDPレイテンシー |
| Local RTT | クライアントからローカルゲートウェイまでのレイテンシー |
| Upload | 推定上り帯域幅 |
| Download | 推定下り帯域幅 |

ツールスクリーンキャプチャ

クイックテスト:

| ••• ••• (7) 1. | | | | | Start Test | |
|-------------------------|------------------|------------|------------|-------------|-----------------------|---------------|
| | WiFi Quality (!) | DNS RTT () | MTR Loss 🕥 | MTR RTT 🕥 | UDP Loss (!) | UDP RTT (!) |
| | EXCELLENT | 45ms | 0.0% | 76.0ms | 0.00% | 86ms |
| | | | | Lo | cal IP : 113.108.77.5 | 50 |
| ±. | | | | | | |
| \$ | | | | User: +8615 | 0 Ve | rsion: 1.1.19 |

継続的テスト:



Web

最終更新日:::2024-07-19 15:29:07

入室前や通話中に、ユーザーのネットワーク品質を検出し、ユーザーの現在のネットワーク品質を事前に判断す ることができます。ユーザーのネットワーク品質が低すぎる場合は、通常の通話品質を確保するためにネットワー ク環境を変更するようにユーザーにアドバイスする必要があります。

このドキュメントでは、主にNETWORK_QUALITYイベントに基づいて通話前のネットワーク品質検出を実装す る方法を紹介します。通話中にネットワーク品質を確認するには、NETWORK_QUALITYイベントを監視するのみ です。

実装のフロー

1.TRTC.createClientを呼び出して、それぞれuplinkClientおよびdownlinkClientという2つのClientを作成します。

2. これら2つのClientは、同じルームに入ります。

3. uplinkClientを使用してプッシュし、NETWORK_QUALITYイベントを監視してアップリンクのネットワーク品 質を確認します。

4. downlinkClientを使用してプルし、NETWORK_QUALITYイベントを監視してダウンリンクのネットワーク品質 を確認します。

5. プロセス全体は約15秒間続く可能性があり、最後に平均ネットワーク品質を使用して、アップリンクとダウン リンクのネットワーク状態を大まかに判断します。

ご注意:

検出プロセスには少額の基本サービス料金が発生します。プッシュ解像度が指定されていない場合、デフォルトで 640*480の解像度でプッシュします。

サンプルコード



```
let uplinkClient = null; // アップリンクネットワーク品質の確認に使用されます
let downlinkClient = null; //ダウンリンクネットワーク品質の確認に使用されます
let localStream = null; // テストされるストリームに使用されます
let testResult = {
    // アップリンクネットワーク品質のデータを記録します
    uplinkNetworkQualities: [],
    // ダウンリンクネットワーク品質のデータを記録します
    downlinkNetworkQualities: [],
    average: {
        uplinkNetworkQuality: 0,
        downlinkNetworkQuality: 0
```

```
}
// 1. アップリンクネットワーク品質を確認します
async function testUplinkNetworkQuality() {
 uplinkClient = TRTC.createClient({
   sdkAppId: 0, // sdkAppIdを記入します
   userId: 'user_uplink_test',
   userSig: '', // uplink_testOuserSig
   mode: 'rtc'
 });
 localStream = TRTC.createStream({ audio: true, video: true });
 // 実際のサービスシーンに応じてvideo profileを設定します
 localStream.setVideoProfile('480p');
 await localStream.initialize();
 uplinkClient.on('network-quality', event => {
   const { uplinkNetworkQuality } = event;
   testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
 });
 // テスト用ルームを追加します。競合を避けるためにルーム番号はランダムである必要があります
 await uplinkClient.join({ roomId: 8080 });
 await uplinkClient.publish(localStream);
}
// 2. ダウンリンクネットワーク品質を確認します
async function testDownlinkNetworkQuality() {
 downlinkClient = TRTC.createClient({
   sdkAppId: 0, // sdkAppIdを記入します
   userId: 'user_downlink_test',
   userSig: '', // userSig
   mode: 'rtc'
 });
 downlinkClient.on('stream-added', async event => {
   await downlinkClient.subscribe(event.stream, { audio: true, video: true });
       // サブスクリプションに成功した後、ネットワーク品質イベントの監視を開始します
   downlinkClient.on('network-quality', event => {
     const { downlinkNetworkQuality } = event;
     testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
   });
 })
 // テスト用ルームを追加します。競合を避けるためにルーム番号はランダムである必要があります
 await downlinkClient.join({ roomId: 8080 });
```

```
// 3. 確認を開始します
testUplinkNetworkQuality();
testDownlinkNetworkQuality();
// 4.15秒後に確認を停止し、平均ネットワーク品質を計算します
setTimeout(() => {
 // アップリンクの平均ネットワーク品質を計算します
 if (testResult.uplinkNetworkQualities.length > 0) {
   testResult.average.uplinkNetworkQuality = Math.ceil(
     testResult.uplinkNetworkQualities.reduce((value, current) => value + current,
   );
 }
 if (testResult.downlinkNetworkQualities.length > 0) {
   // ダウンリンクの平均ネットワーク品質を計算します
   testResult.average.downlinkNetworkQuality = Math.ceil(
     testResult.downlinkNetworkQualities.reduce((value, current) => value + curren
   );
 }
 // 確認が終了し、関連する状態がクリアされます。
 uplinkClient.leave();
 downlinkClient.leave();
 localStream.close();
}, 15 * 1000);
```

結果の分析

上記の手順により、アップリンクの平均ネットワーク品質とダウンリンクの平均ネットワーク品質を取得できま す。ネットワーク品質の列挙値は次のとおりです:

| 数值 | 意味 |
|----|---|
| 0 | ネットワーク状態は不明で、現在のclientインスタンスがまだアップリンク/ダウンリンク接続を確 立していないことを示しています |
| 1 | ネットワーク状態が優れています |
| 2 | ネットワーク状態が良好です |
| 3 | ネットワーク状態が一般です |
| 4 | ネットワーク状態が悪いです |
| | |

| 5 | ネットワーク状態が非常に悪いです |
|---|---|
| 6 | ネットワーク接続が切断されています。注:ダウンリンクネットワーク品質がこの値の場合、すべ てのダウンリンク接続が切断されていることを示します |

ご注意:

推奨事項:ネットワーク品質が3を超える場合は、ユーザーにネットワークを確認してネットワーク環境を変更す るように提案する必要があります。そうでない場合、通常のオーディオビデオ通話を確保することが困難になりま す。

次のポリシーを使用して、帯域幅の消費を削減することもできます:

アップリンクネットワーク品質が3より大きい場合、LocalStream.setVideoProfile()インターフェースを使用して ビットレートを低減したりLocalStream.muteVideo()を使用してビデオを閉じたりして、アップリンク帯域幅の消 費を削減することができます。

ダウンリンクネットワーク品質が3より大きい場合、低画質ストリームをサブスクリプションしたり(高・低画質 ストリーム伝送を有効にするを参照)オーディオのみをサブスクリプションしたりして、ダウンリンク帯域幅の 消費を削減することができます。

TRTCクラウドレコーディングの説明

最終更新日:::2024-07-19 15:29:07

eラーニング、ステージライブストリーミング、ビデオミーティング、オンライン医療相談、リモートバンキング などのユースケースでは、コンテンツ審査、録画アーカイブ、ビデオ再生などのニーズを考慮して、ビデオ通話や インタラクティブライブストリーミングのプロセス全体をレコーディングし保存する必要が度々出てきます。これ らはクラウドレコーディング機能によって実現することができます。

機能の説明

TRTCのクラウドレコーディング機能では、REST APIインターフェースを呼び出してクラウドレコーディングタ スクを開始し、レコーディングしたいオーディオビデオストリーミングをサブスクライブし、リアルタイムかつ フレキシブルに管理することができます。また、開発者が自らサーバーやレコーディング関連モジュールをデプロ イする必要がなく、より軽く使いやすくなっています。

レコーディングモード:シングルストリームレコーディングでは、ルームの各ユーザーのオーディオビデオスト リームをそれぞれ独立したファイルとしてレコーディングできます。ミクスストリームレコーディングでは、同一 のルームのオーディオビデオメディアストリームを1つのファイルとしてレコーディングすることができます ストリームサブスクリプション:サブスクリプションユーザーのブラックリスト/ホワイトリストを制定する方法

で、サブスクライブしたいユーザーのメディアストリームを指定することができます

トランスコードパラメータ:ミクスストリーミングのシーンで、コーデックのパラメータを設定することで、レ コーディングするビデオファイルの品質を指定することができます

ミクスストリームパラメータ:ミクスストリームのシーンで、複数のフレキシブルな自動マルチ画面レイアウト テンプレートやカスタムレイアウトテンプレートをサポートします

ファイルストレージ:レコーディングしたファイルの保存先にクラウドストレージ/VODを指定することができま す。現時点で、クラウドストレージベンダーはTencent CloudのCOSストレージをサポートし、VODベンダーは Tencent Cloud VODをサポートしています

(今後、その他のクラウドベンダーのストレージやVODサービスをサポートする場合があり、その際はお客様の クラウドサービスアカウントのご提供が必要です。クラウドストレージサービスにはストレージパラメータのご提 供が必要です)

コールバック通知:コールバック通知機能をサポートし、コールバックドメイン名を設定することで、クラウド レコーディングのイベントステータスがお客様のコールバックサーバーに通知されます

シングルストリームレコーディング



シングルストリームレコーディングのシーンを図にしたものです。ルーム1234でキャスター1とキャスター2がど ちらもオーディオビデオストリームを行っています。キャスター1とキャスター2のオーディオビデオストリーム をサブスクライブし、レコーディングモードをシングルストリームレコーディングに設定し、レコーディングバッ クエンドでキャスター1とキャスター2のオーディオビデオストリームをそれぞれプルし、それらを独立したメ ディアファイルとしてレコーディングすると仮定します。ファイルには次のものが含まれます。

- 1. キャスター1のビデオM3U8インデックスファイル1個。
- 2. キャスター1のいくつかのビデオTS分割ファイル。
- 3. キャスター1のオーディオM3U8インデックスファイル1個。
- 4. キャスター1のいくつかのオーディオTS分割ファイル。
- 5. キャスター2のビデオM3U8インデックスファイル1個。
- 6. キャスター2のいくつかのビデオTS分割ファイル。
- 7. キャスター2のオーディオM3U8インデックスファイル1個。
- 8. キャスター2のいくつかのオーディオTS分割ファイル。

レコーディングバックエンドはこれらのファイルを、お客様の指定するクラウドストレージサーバーにアップ ロードします。お客様の業務バックエンドはこれらのレコーディングファイルをプルし、業務ニーズに応じてファ イルの結合やトランスコード操作を行う必要があります。オーディオビデオの結合とトランスコードのスクリプト をご提供できます。

ミクスストリームレコーディング



ミクスストリームレコーディングのシーンを図にしたものです。ルーム1234でキャスター1とキャスター2がどち らもオーディオビデオストリームを行っています。キャスター1とキャスター2のオーディオビデオストリームを サブスクライブし、レコーディングモードをミクスストリームレコーディングに設定し、レコーディングバックエ ンドでキャスター1とキャスター2のオーディオビデオストリームをそれぞれプルし、それらのビデオストリーム を設定したマルチ画面テンプレートに従ってミクスストリームし、オーディオストリームをミキシングし、最後 にメディアストリームを1つのメディアファイルに統合すると仮定します。ファイルには次のものが含まれます。 1. ミクスストリーム後のビデオM3U8インデックスファイル1個。

2. ミクスストリーム後のいくつかのビデオTS分割ファイル。

レコーディングバックエンドはこれらのファイルを、お客様の指定するクラウドストレージサーバーにアップ ロードします。お客様の業務バックエンドはこれらのレコーディングファイルをプルし、業務ニーズに応じてファ イルの結合やトランスコード操作を行う必要があります。オーディオビデオの結合とトランスコードのスクリプト をご提供できます。

ご注意:

レコーディングインターフェースの呼び出し頻度は20qpsに制限されています

1つのインターフェースのタイムアウト時間は6秒です

デフォルトでサポートされている同時レコーディングチャネル数は100チャネルです。チャネル数を増やしたい場 合は、チケットを提出してご連絡ください。

1つのレコーディングタスクにつき、同時サブスクリプションをサポートする1ルーム内のオーディオビデオスト リームは最大で25チャネルです。

呼び出しフロー

1. レコーディング開始
お客様のバックエンドサービスからREST API (CreateCloudRecording)を呼び出してクラウドレコーディングを 開始します。特に重視すべきパラメータは次のとおりです。

タスクID (TaskId)

このパラメータはそのレコーディングタスクの固有識別子であり、後にこのレコーディングタスクインターフェースの操作を行うための入力パラメータとして、このタスクIDを保存しておく必要があります。

レコーディングモード (RecordMode)

シングルストリームレコーディングでは、ルーム内でお客様がサブスクライブしたキャスターのオーディオおよ びビデオファイルをそれぞれレコーディングし、レコーディングファイル(M3U8/TS)をクラウドストレージに アップロードします。

ミクスストリームレコーディングでは、ルーム内でお客様がサブスクライブしたすべてのキャスターのオーディ オビデオストリームを1つのオーディオビデオファイルにレコーディングし、レコーディングファイル[M3U8/TS] をクラウドストレージにアップロードします。

レコーディングユーザーのブラックリスト/ホワイトリスト(SubscribeStreamUserIds)

デフォルトでは、クラウドレコーディングはルーム内のすべてのメディアストリーム(最大25チャネル)をサブ スクライブできます。このパラメータによって、サブスクライブするキャスターユーザーのブラックリスト/ホワ イトリスト情報を指定することも可能です。もちろん、レコーディング中の更新操作もサポートしています。

クラウドストレージパラメータ (StorageParams)

クラウドストレージパラメータを指定すると、レコーディング後のファイルはお客様がアクティブ化した指定の クラウドストレージ/VODサービスにアップロードされます。クラウドストレージ/VODスペースのパラメータの有 効性と、未払いの料金がない状態を維持するようご注意ください。ここでは、レコーディングファイルの名前に 決まったルールがあることに注意が必要です

レコーディングファイル名の命名ルール

シングルストリームレコーディングのM3U8ファイル名ルール: <Prefix>/<Taskld>/<SdkAppld>_<RoomId>_UserId_s_<UserId>_UserId_e_<Mediald>_<Type>.m3u8 シングルストリームレコーディングのTSファイル名ルール: <Prefix>/<TaskId>/<SdkAppld>_<RoomId>_UserId_s_<UserId>_UserId_e_<Mediald>_<Type>_<UTC>.ts シングルストリームレコーディングのmp4ファイル名ルール: <Prefix>/<TaskId>/<SdkAppld>_<RoomId>_UserId_s_<UserId>_UserId_e_<Mediald>_<Index>.mp4 ミクスストリームレコーディングのM3U8ファイル名ルール: <Prefix>/<TaskId>/<SdkAppld>_<RoomId>.m3u8 ミクスストリームレコーディングのTSファイル名ルール: <Prefix>/<TaskId>/<SdkAppld>_<RoomId>.m3u8 ミクスストリームレコーディングのTSファイル名ルール: <Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<UTC>.ts ミクスストリームレコーディングのmp4ファイル名ルール: <Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<Index>.mp4 高可用性プル後のファイル名ルール

クラウドレコーディングサービス中にデータセンターに障害が発生した場合、高可用性ソリューションによって レコーディングタスクを復旧させることがあります。このような場合、元のレコーディングファイルを上書きしな いよう、プル後にプレフィックスha<1/2/3>を付けることで、高可用性の発生回数を表します。1つのレコーディ ングタスクにつき、プルの回数が最大で3回まで許容されます。

シングルストリームレコーディングのM3U8ファイル名ルール:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>.m3u8 シングルストリームレコーディングのTSファイル名ルール:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>_<UT C>.ts

ミクスストリームレコーディングのM3U8ファイル名ルール:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>.m3u8

ミクスストリームレコーディングのTSファイル名ルール:

<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>_<UTC>.ts

フィールドの意味の説明:

<Prefix>: レコーディングパラメータに設定するファイル名プレフィックスです。設定していなければ存在しません。

<TaskId>: レコーディングのタスクIDです。世界唯一のもので、レコーディング開始後に返されるパラメータ内に 付帯します。

<SdkAppId>: レコーディングタスクのSdkAppIdです。

<RoomId>: レコーディングルーム番号です。

<UserId>: 特殊なbase64処理後のレコーディングストリームのユーザーIDです。下記の注意事項をご参照ください。

<Mediald>:メイン/サブストリームを識別します。mainまたはauxです。

<Type>: レコーディングストリームのタイプです。audioまたはvideoです。

<UTC>: このファイルのUTCレコーディングの開始時間です。タイムゾーンはUTC+0で、年、月、日、時間、分、 秒およびミリ秒から成ります。

このフィールドはありません。トリガーされている場合は分割ファイルのインデックス番号であり、1から順に付 与されます。

ha<1/2/3>: 高可用性プルのプレフィックスです。例えば、最初のプルにより、

<Prefix>/<TaskId>/ha1_<SdkAppId>_<RoomId>.m3u8に変わります。

ご注意:

ここで<RoomId>が文字列のルームIDの場合、ルームIDに対してまずbase64操作を行ってから、base64後の文字 列中の記号'/を'-' (ダッシュ)に、記号'='を'.'にそれぞれ置き換えます。

レコーディングストリームの<UserId>についてはまずbase64操作を行ってから、base64後の文字列中の記号'/'を'-' (ダッシュ)に、記号'='を'.'にそれぞれ置き換えます。



レコーディング開始時間の取得

レコーディング開始時間の定義は、最初にキャスターのオーディオビデオデータを受信し、最初のファイルのレ コーディングを開始したサーバーunix時間です。

次の3つの方法で、レコーディング開始のタイムスタンプを取得することができます。

レコーディングファイルインターフェース(DescribeCloudRecording)のBeginTimeStampフィールドを照会する 例えば、次の照会インターフェースから返された情報からは、BeginTimeStampは1622186279144msであること がわかります。



M3U8ファイル中の、対応するタグの項目(#EXT-X-TRTC-START-REC-TIME)を読みとる

例えば、次のM3U8ファイルからは、レコーディング開始時間のunixタイムスタンプが1622425551884msである ことがわかります。





```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-TARGETDURATION:70
#EXT-X-TRTC-START-REC-TIME:1622425551884
#EXT-X-TRTC-VIDEO-METADATA:WIDTH:1920 HEIGHT:1080
#EXTINF:12.074
1400123456_12345__UserId_s_MTY4NjExOQ.._UserId_e_main_video_20330531094551825.ts
#EXTINF:11.901
```

#EXTINF:12.076
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094615764.ts
#EXT-X-ENDLIST

レコーディングコールバックイベントを監視する

サブスクリプションのコールバックにより、イベントタイプ**307**のBeginTimeStampフィールドで、レコーディン グファイルに対応するレコーディング開始時間のタイムスタンプを取得することができます。

例えば、次のコールバックイベントからは、このファイルのレコーディング開始時間のunixタイムスタンプが 1622186279144msであることが読み取れます。



```
"EventGroupId": 3,
       "EventType": 307,
       "CallbackTs": 1622186289148,
       "EventInfo":
                     {
               "RoomId":
                            "xx",
               "EventIs":
                             "1622186289",
               "UserId":
                             "xx",
               "TaskId":
                             "xx",
               "Payload":
                             {
                      "FileName":
                                    "xx.m3u8",
                                    "xx",
                      "UserId":
                      "TrackType":
                                    "audio",
                      "BeginTimeStamp": 1622186279144
               }
       }
}
```

ミクスストリームレコーディングのウォーターマークパラメータ (MixWatermark)

ミクスストリームレコーディング中の、画像ウォーターマーク追加をサポートしています。最大サポート数は25 個で、キャンバスの任意の位置にウォーターマークを追加することができます。

| フィールド名 | 説明 |
|--------|-----------------------|
| Тор | ウォーターマークの左上隅に対しての垂直移動 |
| Left | ウォーターマークの左上隅に対しての水平移動 |
| Width | ウォーターマークの表示幅 |
| Height | ウォーターマーク表示の高さ |
| url | ウォーターマークファイルの保存先url |

ミクスストリームレコーディングのレイアウトモードパラメータ(MixLayoutMode)

9グリッドレイアウト(デフォルト)、フロートレイアウト、画面共有レイアウト、カスタムレイアウトの、4種類 のレイアウトをサポートしています

9グリッドレイアウト:

キャスターの数に応じて各画面のサイズは自動調整されます。各キャスターの画面サイズは同一で、最大25画面 をサポートします。

サブ画面が1枚の場合:

各小画面の幅と高さはそれぞれキャンバス全体の幅と高さとなります。



サブ画面が2枚の場合:

各小画面の幅はキャンバス全体の幅の1/2となります。

各小画面の高さはキャンバス全体の高さとなります。

サブ画面が4枚以下の場合:

各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/2となります。

サブ画面が9枚以下の場合:

各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/3となります。

サブ画面が16枚以下の場合:

各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/4となります。

サブ画面が16枚を超える場合:

各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/5となります。

9グリッドレイアウトはサブスクリプションのサブ画面が増えるに従って、下図のように変化します。



フロートレイアウト:

デフォルトでは最初に入室したキャスター(キャスター1名を指定することもできます)のビデオ画面はスクリー ン全体となります。その他のキャスターのビデオ画面は左下隅から順に、入室の順序に従って水平に並び、小画面 として表示され、小画面が大画面の上にフロート表示されます。画面の数が17枚以下の場合は、1列に4画面(4 x 4列)となります。画面の数が17枚より多い場合は、小画面が1列に5画面(5 x 5列)となるよう再レイアウトされ ます。最大25画面をサポートし、ユーザーがオーディオのみを送信している場合でも、画面の位置を占有するこ とができます。

サブ画面が17枚以下の場合:

各小画面の幅と高さはそれぞれキャンバス全体の幅と高さ×0.235となります。

隣接する小画面の左右と上下の間隔はそれぞれキャンバス全体の幅と高さ×0.012となります。

小画面からキャンバスの縦枠および横枠までの余白も、それぞれキャンバス全体の幅と高さ×0.012となります。



サブ画面が17枚を超える場合:

各小画面の幅と高さはそれぞれキャンバス全体の幅と高さ×0.188となります。

隣接する小画面の左右と上下の間隔はそれぞれキャンバス全体の幅と高さ×0.01となります。

小画面からキャンバスの縦枠および横枠までの余白も、それぞれキャンバス全体の幅と高さ×0.01となります。 フロートレイアウトはサブスクリプションのサブ画面が増えるに従って、下図のように変化します。

| 14 | 15 | 16 | 17 |
|----|----|----|----|
| 10 | 11 | 12 | 13 |
| 6 | 7 | 8 | 9 |
| 2 | 3 | 4 | 5 |

| 21 | 22 | 23 | 24 | 25 |
|----|----|----|----|----|
| 16 | 17 | 18 | 19 | 20 |
| 12 | 12 | 13 | 14 | 15 |
| 7 | 8 | 9 | 10 | 11 |
| 2 | 3 | 4 | 5 | 6 |

画面共有レイアウト:

キャスター1名を画面左側の大画面位置に指定し(指定しない場合、大画面位置は背景色になります)、その他の キャスターは右側に上から下へ順に垂直に配置します。画面の数が17枚より少ない場合は、右側の各列に最大8 名、最大2列まで配置します。画面の数が17枚より多い場合は、17枚目以降のキャスターの画面を左下隅から順に 水平に配置します。最大24画面をサポートし、キャスターがオーディオのみを送信している場合でも、画面の位 置を占有することができます。

サブ画面が5枚以下の場合:

右側の小画面の幅はキャンバス全体の幅の1/5、右側の小画面の高さはキャンバス全体の高さの1/4となります。

左側の大画面の幅はキャンバス全体の幅の4/5、左側の大画面の高さはキャンバス全体の高さとなります。 サブ画面が5枚を超え、7枚以下の場合:

右側の小画面の幅はキャンバス全体の幅の1/7、右側の小画面の高さはキャンバス全体の高さの1/6となります。 左側の大画面の幅はキャンバス全体の幅の6/7、左側の大画面の高さはキャンバス全体の高さとなります。 サブ画面が7枚を超え、9枚以下の場合:

右側の小画面の幅はキャンバス全体の幅の1/9、右側の小画面の高さはキャンバス全体の高さの1/8となります。 左側の大画面の幅はキャンバス全体の幅の8/9、左側の大画面の高さはキャンバス全体の高さとなります。 サブ画面が9枚を超え、17枚以下の場合:

右側の小画面の幅はキャンバス全体の幅の1/10、右側の小画面の高さはキャンバス全体の高さの1/8となります。 左側の大画面の幅はキャンバス全体の幅の4/5、左側の大画面の高さはキャンバス全体の高さとなります。 サブ画面が17枚を超える場合:

右(下)側の小画面の幅はキャンバス全体の幅の1/10、右(下)側の小画面の高さはキャンバス全体の高さの1/8 となります。

左側の大画面の幅はキャンバス全体の幅の4/5、左側の大画面の高さはキャンバス全体の高さの7/8となります。 画面共有レイアウトはサブスクリプションのサブ画面が増えるに従って、下図のように変化します。









| | 1 | |
|----------------|---|--|
| | 2 | |
| | 3 | |
| Saraan Sharing | 4 | |
| Screen Sharing | 5 | |
| | 6 | |
| | 7 | |
| | 8 | |



| | | | | | | | | 1 |
|----|----------------|----|----|----|----|----|----|---|
| | | | | | | 2 | | |
| | | | | | | | | 3 |
| | Screen Sharing | | | | | 4 | | |
| | | | | | | 5 | | |
| | | | | | | | | 6 |
| | | | | | | | | 7 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 8 |

カスタムレイアウト:

業務の必要性に応じ、MixLayoutList内で各キャスター画面のレイアウト情報をカスタマイズできます。

2. レコーディング照会(DescribeCloudRecording)

必要に応じ、このインターフェースを呼び出してレコーディングサービスのステータスを照会することができま す。レコーディングタスクが存在する場合にのみ、情報の照会が可能なことにご注意ください。レコーディング タスクがすでに終了している場合はエラーが返されます。

レコーディングしたファイルのリスト(StorageFile)には、今回レコーディングしたすべてのM3U8インデックス ファイルおよびレコーディング開始時のUnixタイムスタンプ情報が含まれます。VODにアップロードしたタスク については、このインターフェースから返されるStorageFileは空となることにご注意ください。

3. レコーディング更新(ModifyCloudRecording)

必要に応じ、このインターフェースを呼び出してレコーディングサービスのパラメータを変更することができま す。例えば、サブスクリプションのブラックリスト/ホワイトリストSubscribeStreamUserIds(シングルストリー ミングとミクスストリーミングレコーディングで有効)、レコーディングテンプレートのパラメータ MixLayoutParams(ミクスストリーミングレコーディングで有効)などがあります。更新操作はすべて上書きの操 作であり、増分更新の操作ではないことにご注意ください。毎回の更新時に、テンプレートパラメータ MixLayoutParamsおよびブラックリスト/ホワイトリストSubscribeStreamUserIdsを含む、全情報を引き継ぐ必要 があるため、以前のレコーディング開始パラメータまたは再計算した完全なレコーディング関連パラメータを保存 する必要があります。

4. レコーディング停止(DeleteCloudRecording)

レコーディング終了後に、レコーディング停止(DeleteCloudRecording)インターフェースを呼び出してレコー ディングタスクを終了する必要があります。これを行わなかった場合、レコーディングタスクは設定したタイムア ウト時間MaxIdleTimeになると自動的に終了します。注意が必要な点は、MaxIdleTimeの定義が、ルーム内に継続 的にキャスターがいない状態がMaxIdleTimeの時間を超えることを指す点です。ルーム内にキャスターが存在して いるが、キャスターがデータをアップストリームしていないという場合は、タイムアウト時間のカウント状態に 入りません。レコーディング終了時にこのインターフェースを呼び出してレコーディングタスクを終了すること をお勧めします。

高度な機能

mp4ファイルのレコーディング

レコーディングの出力ファイル形式をmp4形式にしたい場合は、レコーディング開始パラメータ(OutputFormat)で レコーディングファイルの出力形式をhls+mp4に指定することができます。デフォルトではhlsファイルへのレコー ディングのままですが、hlsレコーディングの終了後、ただちにmp4への変換タスクがトリガーされ、hlsの存在す るcos内からプルされてmp4パッケージ化が行われ、お客様のcosにアップロードされます。

ここではCOSのファイルプル権限を提供する必要があることにご注意ください。これを行わなければ、mp4変換 タスクはhlsファイルのプル失敗によって中止されます。

mp4ファイルは次のような場合、強制的に分割されます。

1. レコーディングファイルの長さが24時間を超えた場合。

2.1個のmp4ファイルのサイズが2GBに達した場合。

具体的なフローは図のとおりです



VODへのレコーディング

レコーディングした出力ファイルをVODプラットフォームにアップロードしたい場合は、レコーディング開始の ストレージパラメータ(StorageParams)の中でCloudVodメンバーパラメータを指定することができます。レコー ディングバックエンドはレコーディング終了後、レコーディングしたmp4ファイルを指定された方式でVODプ ラットフォームにアップロードし、コールバックの形式で再生アドレスをお客様に送信します。レコーディング モードがシングルストリーミングレコーディングモードの場合は、各サブスクリプションのキャスターが1つずつ 対応する再生アドレスを持っています。レコーディングモードがミクスストリーミングレコーディングモードの場 合は、ミクスストリーミング後のメディアの再生アドレスは1つのみとなります。VODアップロードタスクには次 のいくつかの注意事項があります。

1. ストレージパラメータ中のCloudVodとCloudStorageは同時に1つしか指定できません。それ以上を指定した場合はレコーディングの開始に失敗します。

2. VODアップロードタスクの過程でDescribeCloudRecordingを使用して照会したタスクステータスには、レコー ディングファイルの情報が含まれません。

具体的なフローは図のとおりです。ここに記載したクラウドストレージはレコーディング内部のクラウドストレージであり、お客様が関連のパラメータを入力する必要はありません。



シングルストリームファイル結合スクリプト

シングルストリームオーディオビデオファイル結合スクリプトをご提供しています。シングルストリームレコー ディングのオーディオビデオファイルを結合してMP4ファイルとするために用います。

説明:

2つの分割ファイル間の時間間隔が15秒を超え、その間に何のオーディオ/ビデオ情報もない場合(サブストリームを無効にしている場合、サブストリーム情報は無視できます)、2つの分割ファイルはそれぞれ異なるセグメントであると判断されます。それらのうちレコーディング時間が早い方の分割ファイルが前のセグメントの終了分割ファイル、レコーディング時間が遅い方の分割ファイルが後のセグメントの開始分割ファイルとそれぞれ判断されます。

セグメントモード (-m0)

このモードでは、スクリプトは各Userld下のレコーディングファイルをセグメントごとに結合し、1つのUserld下 のセグメントがそれぞれのファイルに独立して結合されます。

結合モード (-m1)

同一のUserld下の全セグメントを1つのオーディオビデオファイルとして結合します。-sオプションを利用して、 各セグメント間の間隔を埋めるかどうかを選択できます。

依存環境

Python3

Centos: sudo yum install python3



Ubuntu: sudo apt-get install python3

Python3依存パッケージ



sortedcontainers:pip3 install sortedcontainers

使用方法

1.結合スクリプトを実行します: python3 TRTC_Merge.py [option]
 2.レコーディングファイルディレクトリ下で結合後のmp4ファイルを生成します
 例: python3 TRTC_Merge.py -f /xxx/file -m 0

オプションパラメータおよび対応する機能は下表のとおりです。

| パラ メー タ | 機能 |
|---------------|--|
| -f | 結合対象のファイルのストレージパスを指定します。複数のUserldを持つレコーディングファイルの 場合、スクリプトはそれぞれ結合操作を行います。 |
| -m | 0:セグメントモード(デフォルト設定)。このモードでは、スクリプトは各Userld下のレコーディングファイルをセグメントごとに結合します。各Userldにつき、複数のファイルが生成される可能性があります。 1:結合モード。1つのUserld下の全オーディオビデオファイルを1つのファイルとして結合します。 |
| -S | 保存モード。このパラメータを設定すると、結合モードにおけるセグメント間の空白部分が削除され ます。ファイルの実際の時間は物理時間より短くなります。 |
| -a | 0: メインストリーム結合(デフォルト設定)。同一のUserldのメインストリームをオーディオと結合させます。サブストリームはオーディオと結合しません。 1: 自動結合。メインストリームが存在する場合は、メインストリームをオーディオと結合し、メインストリームが存在せずサブストリームが存在する場合は、サブストリームをオーディオと結合します。 2: サブストリーム結合。同一のUserldのサブストリームをオーディオと結合させます。メインストリームはオーディオと結合しません |
| -р | 出力するビデオのfpsを指定します。デフォルトでは15fps、有効範囲は5~120fpsです。5fps未満は 5fpsとして、120fpsを超える場合は120fpsとしてカウントします。 |
| -r | 出力するビデオの解像度を指定します。-r 640 360であれば、出力するビデオの幅が640、高さが360 であることを表します。 |

ファイル名のルール

オーディオビデオファイル名のルール: UserId_timestamp_av.mp4

オーディオのみのファイル名のルール: Userld_timestamp.m4a

説明:

ここでのUserldはレコーディングストリームのユーザーIDの特殊なbase64値です。詳細についてはレコーディン グ開始の項のファイル名命名ルールに関する説明をご参照ください。timestampは各セグメントの最初のts分割 ファイルのレコーディング開始時間となっています。

コールバックインターフェース

コールバックを受信するHttp/Httpsサービスゲートウェイを設けてコールバックメッセージをサブスクライブする ことができます。関連するイベントが発生した際、クラウドレコーディングシステムによってイベント通知がお客 様のメッセージ受信サーバーにコールバックされます。

イベントコールバックメッセージ形式

イベントコールバックメッセージは、HTTP/HTTPS POSTリクエストとしてサーバーに送信されます。以下がその詳細となります。

文字エンコード形式:UTF-8。

リクエスト:bodyの形式はJSONです。

応答:HTTP STATUS CODE = 200、サーバーは応答パケットの具体的な内容を無視します。プロトコルとの親和 性のため、クライアントの応答内容に、JSON: {"code":0} を付けることをお勧めします。

パラメータの説明

イベントコールバックメッセージのheaderには、次のフィールドが含まれています。

| フィールド名 | 值 |
|--------------|--------------------|
| Content-Type | application/json |
| Sign | 署名值 |
| SdkAppId | sdk application id |

イベントコールバックメッセージのbodyには、次のフィールドが含まれています。

| フィールド名 | タイプ | 意味 |
|--------------|----------------|--|
| EventGroupId | Number | イベントグループID。クラウドレコーディングでは 3 に固定されています |
| EventType | Number | コールバック通知のイベントタイプ |
| CallbackTs | Number | イベントコールバックサーバーからお客様のサーバーに送信されるコールバッ クリクエストのUnixタイムスタンプ。単位はミリ秒 |
| EventInfo | JSON Object | イベント情報 |

イベントタイプの説明

| フィールド名 | タイプ | 意味 |
|---|-----|-----------------------------------|
| EVENT_TYPE_CLOUD_RECORDING_RECORDER_START | 301 | クラウドレコーディング レコーディングモジュール 起動 |
| | | |



| EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP | 302 | クラウドレコーディング レコーディングモジュール 退出 |
|---|-----|--|
| EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START | 303 | クラウドレコーディング アップロードモジュール起 動 |
| EVENT_TYPE_CLOUD_RECORDING_FILE_INFO | 304 | クラウドレコーディング m3u8インデックスファイ ルを生成し、初回生成と アップロードの成功後に コールバック |
| EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP | 305 | クラウドレコーディング アップロード終了 |
| EVENT_TYPE_CLOUD_RECORDING_FAILOVER | 306 | クラウドレコーディング マイグレーションが発生 し、元のレコーディングタ スクが新たなロードに移行 した際にトリガー |
| EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE | 307 | クラウドレコーディング M3U8ファイル(最初のts分 割ファイルの切り出し) 生 成後にコールバック |
| EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR | 308 | クラウドレコーディング アップロードモジュールに エラー発生 |
| EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR | 309 | クラウドレコーディング デコード画像ファイルのダ ウンロード時にエラー発生 |
| EVENT_TYPE_CLOUD_RECORDING_MP4_STOP | 310 | クラウドレコーディング mp4レコーディングタスク 終了、コールバックにはレ コーディングしたmp4ファ イル名を含める |
| EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT | 311 | クラウドレコーディング vodレコーディングタスク メディアリソースアップ ロード完了 |
| | | |

EVENT_TYPE_CLOUD_RECORDING_VOD_STOP 312 クラウドレコーディング vodレコーディングタスク 終了

イベント情報の説明

| フィールド名 | タイプ | 意味 |
|---------|---------------|----------------------------------|
| RoomId | String/Number | ルーム名(タイプはクライアントのルーム番号タイプと同様) |
| EventTs | Number | 時間で発生するUnixタイムスタンプ。単位は秒 |
| UserId | String | レコーディングボットのユーザーID |
| Taskld | String | レコーディングID。1回のクラウドレコーディングタスクの固有ID |
| Payload | JsonObject | イベントタイプごとに異なる定義 |

イベントタイプ301 EVENT_TYPE_CLOUD_RECORDING_RECORDER_START時のPayloadの定義

| フィールド 名 | タイプ | 意味 |
|------------|--------|--|
| Status | Number | 0:レコーディングモジュール起動成功。1:レコーディングモジュール起動失 敗。 |





```
{
```

```
"EventGroupId": 3,
"EventType": 301,
"CallbackTs": 1622186275913,
"EventInfo": {
    "RoomId": "xx",
    "EventTs": "1622186275",
    "UserId": "xx",
    "IaskId": "xx",
    "Payload": {
        Status": 0
```



} }

イベントタイプ302 EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP時のPayloadの定義

| フィールド 名 | タイプ | 意味 |
|------------|--------|---|
| LeaveCode | Number | 0:レコーディングモジュールが正常に呼び出し、レコーディングを停止して退出した。 1:レコーディングボットがクライアントから強制退室させられた。 2:クライアントがルームを解散した。 3:サーバーがレコーディングボットを強制退室させた。 4:サーバーがルームを解散した。 9:ルーム内にレコーディングボット以外のユーザーストリームがなく、指定の時間を超過したため退出した。 100:ルームのタイムアウトにより退出した。 101:同一のユーザーが同じルームに重複して入室したため、ボットを退出させた |





```
{
```



} }

イベントタイプ303 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START時のPayloadの定義

| フィールド名 | タイプ | 意味 |
|--------|--------|--|
| Status | Number | 0:アップロードモジュールが正常に起動 1:アップロードモジュール初期化失敗。 |



```
{
       "EventGroupId": 3,
       "EventType": 303,
       "CallbackTs": 1622191965320,
       "EventInfo": {
              "RoomId":
                           "20015",
              "EventTs":
                          1622191965,
              "UserId":
                           "xx",
              "TaskId":
                           "xx",
              "Payload": {
                     "Status": 0
              }
       }
}
```

イベントタイプ304 EVENT_TYPE_CLOUD_RECORDING_FILE_INFO時のPayloadの定義

| フィールド名 | タイプ | 意味 |
|----------|--------|---------------|
| FileList | String | 生成したM3U8ファイル名 |





```
{
```



}

イベントタイプ305 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP時のPayloadの定義

| フィー ルド名 | タイプ | 意味 |
|------------|--------|--|
| Status | Number | 0: 今回のレコーディングアップロードタスクは完了し、指定されたサードパーティク ラウドストレージに全ファイルがアップロードされた 1: 今回のレコーディングアップロードタスクは完了しているが、少なくとも1つ以上 のファイルがサーバーまたはバックアップストレージ上で滞留している 2: サーバーまたはバックアップストレージ上で滞留していたファイルが復元され、指 定されたサードパーティクラウドストレージにアップロードされた |





```
{
```



} }

イベントタイプ306 EVENT_TYPE_CLOUD_RECORDING_FAILOVER時のPayloadの定義

| フィールド名 | タイプ | 意味 |
|--------|--------|---------------------|
| Status | Number | 0:今回のマイグレーションはすでに完了 |





}

イベントタイプ307 EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE時のPayloadの定義

| フィールド名 | タイプ | 意味 |
|----------------|--------|---------------------------------|
| FileName | String | m3u8ファイル名 |
| Userld | String | このレコーディングファイルに対応するユーザーID |
| TrackType | String | audio/video/audio_video |
| BeginTimeStamp | Number | レコーディング開始時のサーバーUnixタイムスタンプ(ミリ秒) |





{

```
"EventGroupId": 3,
"EventType": 307,
"CallbackTs": 1622186289148,
"EventInfo": {
    "RoomId": "xx",
    "EventTs": "1622186289",
    "UserId": "xx",
    "TaskId": "xx",
    "TaskId": "xx",
    "Payload": {
```



}

```
"UserId": "xx",
"TrackType": "audio",
"BeginTimeStamp": 1622186279144
}
```

```
}
```

イベントタイプ308 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR時のPayloadの定義

| フィールド名 | タイプ | 意味 | |
|---------|--------|------------------------------------|--|
| Code | String | サードパーティクラウドストレージから返される code | |
| Message | String | サードパーティクラウドストレージから返されるメッセージの内容 | |



```
{
    "Code": "InvalidParameter",
    "Message": "AccessKey invalid"
}
{
    "EventGroupId": 3,
    "EventType": 308,
    "CallbackTs": 1622191989674,
    "EventInfo": {
        "RoomId": "20015",
    }
}
```



イベントタイプ309 EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR時のPayloadの定義

| フィールド名 | タイプ | 意味 |
|--------|--------|----------------|
| Url | String | ダウンロードに失敗したurl |





```
{
```


}

イベントタイプ310 EVENT_TYPE_CLOUD_RECORDING_MP4_STOP時のPayloadの定義

| フィー ルド名 | タイプ | 意味 |
|------------|--------|--|
| Status | Number | 0: 今回のmp4レコーディングタスクは正常に終了し、指定されたサードパーティクラ ウドストレージに全ファイルがアップロードされた 1: 今回のmp4レコーディングタスクは正常に終了しているが、少なくとも1つ以上の ファイルがサーバーまたはバックアップストレージ上で滞留している 2: 今回のmp4レコーディングタスクは正常に終了しなかった(cosのhlsファイルのプル に失敗したことが原因の可能性あり) |
| FileList | String | 生成した M3U8 ファイル名 |





```
{
```



}

}

```
"FileList": ["xxxx1.mp4", "xxxx2.mp4"]
```

```
}
```

イベントタイプ311 EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT時のPayloadの定義

| フィール ド名 | タイプ | 意味 | | | |
|------------|--------|---|--|--|--|
| Status | Number | 0: このレコーディングファイルはVODプラットフォームに正常にアップロードされた 1: このレコーディングファイルがサーバーまたはバックアップストレージ上で滞留している 2: このレコーディングファイルのVODアップロードタスクに異常が発生した | | | |
| Userld | String | このレコーディングファイルに対応するユーザーID(レコーディングモードがミ クスストリーミングモードの場合、このフィールドは空) | | | |
| TrackType | String | audio/video/audio_video | | | |
| Mediald | String | main/aux | | | |
| FileId | String | このレコーディングファイルのVODプラットフォームでの固有id | | | |
| VideoUrl | String | このレコーディングファイルのVODプラットフォームでの再生アドレス | | | |
| CacheFile | String | このレコーディングファイルに対応するmp4ファイル名(VODアップロード前) | | | |
| Errmsg | String | statueが0ではない場合、対応するエラー情報 | | | |

アップロード成功のコールバック:





```
{
```



| | | | "Те | encentVod": { | |
|---|---|---|-----|---------------|----------------|
| | | | | "UserId": | "xx", |
| | | | | "TrackType": | "audio_video", |
| | | | | "MediaId": | "main", |
| | | | | "FileId": | "xxxx", |
| | | | | "VideoUrl": | "http://xxxx" |
| | | | } | | |
| | | } | | | |
| | } | | | | |
| } | | | | | |
| | | | | | |

アップロード失敗のコールバック:





```
{
```

```
"Errmsg": "xxx",
"TencentVod": {
    "UserId": "123",
    "TrackType": "audio_video",
    "CacheFile": "xxx.mp4"
    }
}
```

イベントタイプ312 EVENT_TYPE_CLOUD_RECORDING_VOD_STOP時のPayloadの定義

| フィールド名 | タイプ | 意味 |
|--------|--------|---|
| Status | Number | 0: 今回のvodアップロードタスクは正常に終了した 1: 今回のvodアップロードタスクは正常に終了しなかった |





```
{
```



} }

ベストプラクティス

レコーディングの高可用性を保証するため、Restful APIを統合する際は、次のいくつかの点に注意するようお勧め します。

1. CreateCloudRecordingリクエストを呼び出した後は、http responseに注目してください。リクエストが失敗した場合は、具体的なステータスコードに応じて必要なリトライポリシーをとる必要があります。

エラーコードは「1次エラーコード」と「2次エラーコード」の組み合わせから成り、

「InvalidParameter.SdkAppId」のようになっています。

返されたCodeがInternalError.xxxxxであれば、サーバーエラーに遭遇したことを示しています。リターンが正常 となり、taskidが得られるまで、同じパラメータを使用して何度かリトライすることができます。例えば、1回目 は3sのリトライ、2回目は6sのリトライ、3回目は12sのリトライというように、バックオフを使用してポリシーを リトライすることをお勧めします。

返されたCodeがInValidParameter.xxxxであれば、入力したパラメータに誤りがあることを示しています。表示に 従ってパラメータをチェックしてください。

返されたCodeがFailedOperation.RestrictedConcurrencyであれば、お客様の同時レコーディングタスク数がバック エンドの予備リソースを超過したことを示しています(デフォルトでは100チャネル)。Tencent Cloudテクニカルサ ポートにご連絡いただき、最大同時チャネル数制限の調整を依頼してください。

2. CreateCloudRecordingを呼び出す際、指定するUserId/UserSigはレコーディングを行う単独のボットユーザーが ルームに入室する際のidですので、TRTCルーム内のその他のユーザーと重複しないようにしてください。また、

TRTCクライアントが入室するルームのタイプはレコーディングインターフェースが指定するルームのタイプと常に同じものにする必要があります。例えば、SDKがルームを新規作成する際に文字列のルームナンバーを使用した場合は、クラウドレコーディングのルームタイプもそれに合わせて文字列のルームナンバーに設定する必要があります。

3. レコーディングステータスの照会を行う場合、次のいくつかの方法でレコーディングに対応するファイル情報 を取得することができます。

CreateCloudRecordingタスクの開始に成功してから約15秒後に、DescribeCloudRecordingインターフェースを呼び出してレコーディングファイルに対応する情報を照会します。ステータスがidleであることがわかった場合、レコーディングがアップストリームされたオーディオビデオストリームをプルできていなかったことを示しています。ルーム内にアップストリーム中のキャスターがいるかどうかを確認してください。

CreateCloudRecordingの開始に成功した後、ルーム内にオーディオビデオのアップストリームが存在することが 確実な状況では、レコーディングファイル名の生成ルールに従ってレコーディングファイル名をスプライシング することができます。具体的なファイル名ルールについては上記をご参照ください(ここにファイル名ルールへの リンクを貼る)。 レコーディングファイルのステータスはコールバックによってお客様のサーバーに送信されます。関連するコール バックをサブスクライブすると、レコーディングファイルのステータス情報を受け取ることができます。(ここに コールバックページへのリンクを貼る)

Cosを通じてレコーディングファイルを照会します。クラウドレコーディングの開始時点でcosに保存されている ディレクトリを指定し、レコーディングタスクの終了後に、対応するディレクトリからレコーディングファイル を見つけることができます。

4. レコーディングユーザー(userid)のusersigの有効期限は、レコーディングタスクの有効期限よりも長く設定する 必要があります。レコーディングタスクマシンのネットワーク接続が切れた後、内部高可用性が有効になり、レ コーディングを再開しようするときに、usersigの期限切れによって失敗することを防止できます。

カスタムキャプチャとレンダリング

Android&iOS&Windows&Mac

最終更新日:::2024-07-19 15:29:07

このドキュメントでは、主にTRTC SDKを使用してカスタムビデオキャプチャとレンダリングを実装する方法を 紹介します。ビデオキャプチャとビデオレンダリングの2つの部分に分かれています。

ビデオキャプチャのカスタマイズ

TRTC SDKのカスタムビデオキャプチャ機能の有効化は、機能の有効化とSDKへのビデオフレームの送信の2つの 手順に分かれています。具体的なAPIの使用手順は次のとおりです。また、対応するプラットホームのAPI-Exampleも提供します:

Android

iOS

Windows

カスタムビデオキャプチャ機能の有効化

まず、TRTCCloudの enableCustomVideoCapture インターフェースを呼び出して、TRTC SDKのカスタムビ デオキャプチャ機能を有効にする必要があります。有効にすると、TRTC SDK独自のカメラ取得および画像処理ロ ジックがスキップされ、エンコードおよび送信機能のみが保持されます。サンプルコードは次のとおりです: Android

iOS&Mac

Windows





TRTCCloud mTRTCCloud = TRTCCloud.shareInstance(); mTRTCCloud.enableCustomVideoCapture(TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, true);





self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud enableCustomVideoCapture:TRTCVideoStreamTypeBig enable:YES];



liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance(); trtc_cloud->enableCustomVideoCapture(TRTCVideoStreamType::TRTCVideoStreamTypeBig, t

カスタムビデオフレームの送信

次に、TRTCCloudの sendCustomVideoData インターフェースを使用して、TRTC SDKに独自のビデオデータ を送信できます。サンプルコードは次のとおりです: 説明:



不要な性能の低下を回避するために、TRTC SDKに入力されるビデオデータには、プラットフォームごとに異なるフォーマット要件があります。詳細については、APIドキュメント:簡体字中国語、Englishを参照してください。

Android

iOS&Mac

Windows



// Androidプラットフォームには、BufferとTextureの2つのスキームがあります。ここでは、Textureス: TRTCCloudDef.TRTCVideoFrame videoFrame = new TRTCCloudDef.TRTCVideoFrame(); videoFrame.texture = new TRTCCloudDef.TRTCTexture();



```
videoFrame.texture.textureId = textureId;
videoFrame.texture.eglContext14 = eglContext;
videoFrame.width = width;
videoFrame.height = height;
videoFrame.timestamp = timestamp;
videoFrame.pixelFormat = TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D;
videoFrame.bufferType = TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE;
mTRTCCloud.sendCustomVideoData(TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, videoFrame)
```



// iOS/Macプラットフォームでは、カメラによってネイティブに取得されたビデオ形式はNV12であり、ネイジ



```
TRTCVideoFrame *videoFrame = [[TRTCVideoFrame alloc] init];
videoFrame.pixelFormat = TRTCVideoPixelFormat_NV12;
videoFrame.bufferType = TRTCVideoBufferType_PixelBuffer;
videoFrame.pixelBuffer = imageBuffer;
videoFrame.timestamp = timeStamp;
```

[[TRTCCloud sharedInstance] sendCustomVideoData:TRTCVideoStreamTypeBig frame:videoF



// Windowsプラットフォームは現在、Bufferスキームのみをサポートしており、この方法で機能を実装する liteav::TRTCVideoFrame frame;



```
frame.timestamp = getTRTCShareInstance()->generateCustomPTS();
frame.videoFormat = liteav::TRTCVideoPixelFormat_I420;
frame.bufferType = liteav::TRTCVideoBufferType_Buffer;
frame.length = buffer_size;
frame.data = array.data();
frame.width = YUV_WIDTH;
frame.height = YUV_HEIGHT;
getTRTCShareInstance()->sendCustomVideoData(&frame);
```

ビデオレンダリングのカスタマイズ

カスタムレンダリングは、主にローカルプレビュー画面のレンダリングとリモートユーザー画面のレンダリングに 分けられています。基本的な原則として、ローカル/リモートのカスタムレンダリングコールバックを設定し、 TRTC SDKは対応するビデオフレーム(つまり、TRTCVideoFrame)をコールバック関 数 onRenderVideoFrame に渡し、開発者は受信したビデオフレームに応じてレンダリングをカスタマイズで きます。このプロセスには、特定のOpenGL基盤が必要です。また、対応するプラットフォームのAPI-Exampleも 提供しています: Android:

iOS

Windows

ローカルプレビュー画面のレンダリングコールバックの設定

Android iOS&Mac

Windows



```
mTRTCCloud.setLocalVideoRenderListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture
@Override
public void onRenderVideoFrame(String suserId int streamType, TRTCCloudDef.TRTC
// 詳細については、TRTC-API-Exampleのカスタムレンダリングのツールクラスをご参照ください
}
});
```



self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud setLocalVideoRenderDelegate:self pixelFormat:TRTCVideoPixelFormat_N



```
// 具体的な実装については、TRTC-API-Example-Qtのtest_custom_render.cppを参照してください。
void TestCustomRender::onRenderVideoFrame(
    const char* userId,
    liteav::TRTCVideoStreamType streamType,
    liteav::TRTCVideoFrame* frame) {
    if (gl_yuv_widget_ == nullptr) {
        return;
    }
    if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
```



// レンダリングウィンドウを調整します
emit renderViewSize(frame->width, frame->height);
// ビデオフレームを描画します
gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
frame->width, frame->height);
}

リモートユーザー画面のレンダリングコールバックの設定

Android

iOS&Mac

Windows





```
videoView = [strongSelf.userVideoViews objectForKey:userId];
}
else{
    videoView = strongSelf.localVideoView;
}
videoView.image = [UIImage imageWithCIImage:[CIImage imageWithCVImageBuffer
videoView.contentMode = UIViewContentModeScaleAspectFit;
CFRelease(frame.pixelBuffer);
});
```



```
. . .
// 具体的な実装については、TRTC-API-Example-Qtのtest_custom_render.cppを参照してください。
void TestCustomRender::onRenderVideoFrame(
   const char* userId,
   liteav::TRTCVideoStreamType streamType,
   liteav::TRTCVideoFrame* frame) {
 if (gl_yuv_widget_ == nullptr) {
   return;
  }
 if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
   // レンダリングウィンドウを調整します
   emit renderViewSize(frame->width, frame->height);
   // ビデオフレームを描画します
   gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
                              frame->width, frame->height);
  }
}
. . .
```

Web

最終更新日:::2024-07-19 15:29:07

このドキュメントでは、主にローカルストリームのカスタムキャプチャおよびオーディオビデオストリームのカ スタム再生とレンダリングなどの高レベルの使用方法を紹介します。

キャプチャのカスタマイズ

{@link TRTC.createStream createStream()}によって、ローカルストリームを作成する場合、SDKを使用してデフォルトのキャプチャ方法を指定できます。

以下のように、カメラとマイクからオーディオビデオのデータをキャプチャします:



```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
    // local stream initialized success
});
```

または、画面共有ストリームをキャプチャします:



```
const localStream = TRTC.createStream({ userId, audio: false, screen: true });
localStream.initialize().then(() => {
    // local stream initialized success
});
```

ローカルストリームを作成する上記の2つの方法は、SDKのデフォルトのキャプチャ方法を使用します。開発者が オーディオビデオストリームを前処理できるようにするために、createStreamは、外部のオーディオビデオソース からのローカルストリームの作成をサポートします。この方法でローカルストリームを作成することにより、開発 者は次のようなカスタムキャプチャを実装できます:

getUserMediaを使用してカメラとマイクのオーディオビデオストリームをキャプチャできます。

getDisplayMediaを使用して画面共有ストリームをキャプチャします。 captureStreamを使用してページで再生されているオーディオビデオをキャプチャします。 captureStreamを使用してcanvasキャンバスでアニメーションをキャプチャします。

ページで再生されているビデオソースのキャプチャ



```
// 現在のブラウザがvideo要素からのstreamのキャプチャをサポートするかどうかを確認します
const isVideoCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
    if (item in document.createElement('video')) {
```

```
return true;
      }
   });
   return false;
};
// 現在のブラウザがvideo要素からのstreamのキャプチャをサポートするかどうかを確認します
if (!isVideoCapturingSupported()) {
   console.log('your browser does not support capturing stream from video element'
   return
}
// ページで再生されているビデオのvideoタグを取得します
const video = document.getElementByID('your-video-element-ID');
// 再生されているビデオからビデオストリームをキャプチャします
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];
const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSourc
// ビデオプロパティが外部から渡されたビデオソースと一致していることを確認してください。一致していな
localStream.setVideoProfile('480p');
localStream.initialize().then(() => {
   // local stream initialized success
});
```

canvasからの動画のキャプチャ



```
// 現在のブラウザがcanvas要素からのstreamのキャプチャをサポートするかどうかを確認します
const isCanvasCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
        if (item in document.createElement('canvas')) {
            return true;
            }
        });
        return false;
};
// 現在のブラウザがcanvas要素からのstreamのキャプチャをサポートするかどうかを確認します
```

```
if (!isCanvasCapturingSupported()) {
   console.log('your browser does not support capturing stream from canvas element
   return
}
// canvasタグを取得します
const canvas = document.getElementByID('your-canvas-element-ID');
// canvasから15 fpsのビデオストリームをキャプチャします
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];
const localStream = TRTC.createStream({ userId, videoSource: videoTrack });
// ビデオプロパティが外部から渡されたビデオソースと一致していることを確認してください。一致していな
localStream.setVideoProfile('480p');
localStream.initialize().then(() => {
   // local stream initialized success
});
```

再生とレンダリングのカスタマイズ

TRTC.createStream()によって作成および初期化されたローカルストリーム、またはClient.on('stream-added')に よって受信されたリモートストリームの場合、オーディオビデオストリームのオブジェクトの{@link Stream#play Stream.play()}方法を使用してオーディオとビデオを再生とレンダリングできます。Stream.play()の内部は、オー ディオプレーヤーとビデオプレーヤーを自動的に作成し、対応する

Appが独自のプレーヤーを使用する場合は、Stream.play()/stop()メソッドを使用せずに呼び出すことができます。 {@link Stream#getAudioTrack Stream.getAudioTrack()}/{@link Stream#getVideoTrack Stream.getVideoTrack()}メ ソッドを使用して対応するオーディオビデオトラックを取得し、独自のプレーヤーを使用してオーディオビデオ を再生およびレンダリングします。このようなカスタム再生とレンダリング方法を使用して、Stream.on('playerstate-changed')イベントはトリガーされず、AppはオーディオビデオトラックMediaStreamTrack の mute/unmute/ended などのイベントを自己監視して、現在のオーディオビデオデータストリームの状態を 判断してください。

また、App層は、 Client.on('stream-added') 、 Client.on('stream-updated') およ び Client.on('stream-removed') などのイベントを監視して、オーディオビデオストリームのライフサイ クルを処理してください。

ご注意:

'stream-added'と'stream-updated'の2つのイベントの処理コールバックでは、オーディオまたはビデオトラックが あるかどうかを確認してください。'stream-updated'イベントの処理では、オーディオまたはビデオトラックがあ る場合、必ずプレーヤーを更新し、最新のオーディオビデオtrackを使用して再生してください。 オンラインの例

カスタムオーディオのキャプチャとレンダリ ング

Android&iOS&Windows&Mac

最終更新日:::2024-07-19 15:29:07

このドキュメントでは、主にTRTC SDKを使用して、カスタムオーディオのキャプチャと取得を実装する方法を 紹介します。オーディオキャプチャとオーディオ取得の2つの部分に分かれています。

オーディオキャプチャのカスタマイズ

TRTC SDKのカスタムオーディオキャプチャ機能の有効化は、機能の有効化とSDKへのオーディオフレームの送 信の2つの手順に分かれています。具体的なAPIの使用手順は次のとおりです。また、対応するプラットホームの API-Exampleも提供します:

Android

iOS

Windows

カスタムオーディオキャプチャ機能の有効化

まず、**TRTCCloud**の enableCustomAudioCapture インターフェースを呼び出して、**TRTC SDK**のカスタム オーディオキャプチャ機能を有効にしてください。サンプルコードは次のとおりです:

Android

iOS&Mac

Windows





TRTCCloud mTRTCCloud = TRTCCloud.shareInstance(); mTRTCCloud.enableCustomAudioCapture(true);




self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud enableCustomAudioCapture:YES];



liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance(); trtc_cloud->enableCustomAudioCapture(true);

カスタムオーディオフレームの送信

次に、TRTCCloudの sendCustomAudioData インターフェースを使用して、TRTC SDKに自身の音声データを 送信できます。サンプルコードは次のとおりです: Android



iOS&Mac

Windows



TRTCCloudDef.TRTCAudioFrame trtcAudioFrame = new TRTCCloudDef.TRTCAudioFrame(); trtcAudioFrame.data = data; trtcAudioFrame.sampleRate = sampleRate; trtcAudioFrame.channel = channel; trtcAudioFrame.timestamp = timestamp; mTRTCCloud.sendCustomAudioData(trtcAudioFrame);



```
TRTCAudioFrame *audioFrame = [[TRTCAudioFrame alloc] init];
audioFrame.channels = audioChannels;
audioFrame.sampleRate = audioSampleRate;
audioFrame.data = pcmData;
```

[self.trtcCloud sendCustomAudioData:audioFrame];





```
liteav::TRTCAudioFrame frame;
frame.audioFormat = liteav::TRTCAudioFrameFormatPCM;
frame.length = buffer_size;
frame.data = array.data();
frame.sampleRate = 48000;
frame.channel = 1;
getTRTCShareInstance()->sendCustomAudioData(&frame);
```

ご注意:

endCustomAudioData`を使用すると、エコーキャンセル(AEC)の機能が無効になる場合があります。

オーディオ生データの取得

音声モジュールは非常に複雑なモジュールで、SDKは音声デバイスのキャプチャおよび再生のロジックを厳密に 制御する必要があります。一部のシーンにおいて、リモートユーザーのオーディオデータまたはローカルマイクに よってキャプチャされたオーディオデータを取得する必要がある場合は、TRTCCloudに対応する異なるプラット フォームのインターフェースを介して取得することができ、対応するプラットフォームのAPI-Exampleも提供しま す:

Android:

iOS

Windows

オーディオコールバック関数の設定

Android

iOS&Mac

Windows



```
mTRTCCloud.setAudioFrameListener(new TRTCCloudListener.TRTCAudioFrameListener() {
    @Override
    public void onCapturedRawAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFr
    }
    @Override
    public void onLocalProcessedAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudi
    }
}
```

| | ØOverride |
|-----|--|
| | public void onRemoteUserAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFra |
| | l |
| | J |
| | @Override |
| | public void onMixedPlayAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFram |
| | ł |
| | |
| | @Override |
| | public void onMixedAllAudioFrame (TRTCCloudDef.TRTCAudioFrame trtcAudioFrame |
| | // 註細については、TRTC-API-Exampleのカスタムレンタリンクのツールクラスをこ参照く |
| }); | 7 |
| ,,, | |
| | |





```
So Tencent Cloud
```

```
// オーディオデータのカスタムコールバックの設定
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->setAudioFrameCallback(callback)
// オーディオデータのカスタムコールバック
virtual void onCapturedRawAudioFrame(TRTCAudioFrame* frame) {
}
virtual void onLocalProcessedAudioFrame(TRTCAudioFrame* frame) {
}
virtual void onPlayAudioFrame(TRTCAudioFrame* frame, const char* userId) {
}
virtual void onMixedPlayAudioFrame(TRTCAudioFrame* frame) {
}
```

ご注意:

音声が途切れたりエコーキャンセル(AEC)が無効になったりするトラブルが発生するおそれがありますので、上述 のコールバック関数では時間のかかる操作を行わず、直接コピーして別のスレッドで処理することを推奨します。 各種の不確実な影響が生じるおそれがありますので、上述のコールバック関数でコールバックされたデータはい ずれも読み取りとコピーのみが許可され、修正することはできません。

Web

最終更新日:::2024-07-19 15:29:07

このドキュメントでは、主にローカルストリームのカスタムキャプチャおよびオーディオビデオストリームのカ スタム再生とレンダリングなどの高レベルの使用方法を紹介します。

キャプチャのカスタマイズ

{@link TRTC.createStream createStream()}によって、ローカルストリームを作成する場合、SDKを使用してデフォルトのキャプチャ方法を指定できます。

以下のように、カメラとマイクからオーディオビデオのデータをキャプチャします:



```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
    // local stream initialized success
});
```

または、画面共有ストリームをキャプチャします:



```
const localStream = TRTC.createStream({ userId, audio: false, screen: true });
localStream.initialize().then(() => {
    // local stream initialized success
});
```

ローカルストリームを作成する上記の2つの方法は、SDKのデフォルトのキャプチャ方法を使用します。開発者が オーディオビデオストリームを前処理できるようにするために、createStreamは、外部のオーディオビデオソース からのローカルストリームの作成をサポートします。この方法でローカルストリームを作成することにより、開発 者は次のようなカスタムキャプチャを実装できます:

getUserMediaを使用してカメラとマイクのオーディオビデオストリームをキャプチャできます。

getDisplayMediaを使用して画面共有ストリームをキャプチャします。 captureStreamを使用してページで再生されているオーディオビデオをキャプチャします。 captureStreamを使用してcanvasキャンバスでアニメーションをキャプチャします。

ページで再生されているビデオソースのキャプチャ



```
// 現在のブラウザがvideo要素からのstreamのキャプチャをサポートするかどうかを確認します
const isVideoCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
    if (item in document.createElement('video')) {
```

```
return true;
      }
   });
   return false;
};
// 現在のブラウザがvideo要素からのstreamのキャプチャをサポートするかどうかを確認します
if (!isVideoCapturingSupported()) {
   console.log('your browser does not support capturing stream from video element'
   return
}
// ページで再生されているビデオのvideoタグを取得します
const video = document.getElementByID('your-video-element-ID');
// 再生されているビデオからビデオストリームをキャプチャします
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];
const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSourc
// ビデオプロパティが外部から渡されたビデオソースと一致していることを確認してください。一致していな
localStream.setVideoProfile('480p');
localStream.initialize().then(() => {
   // local stream initialized success
});
```

canvasからの動画のキャプチャ



```
// 現在のブラウザがcanvas要素からのstreamのキャプチャをサポートするかどうかを確認します
const isCanvasCapturingSupported = () => {
    ['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) =>
        if (item in document.createElement('canvas')) {
            return true;
            }
        });
        return false;
};
// 現在のブラウザがcanvas要素からのstreamのキャプチャをサポートするかどうかを確認します
```

```
if (!isCanvasCapturingSupported()) {
   console.log('your browser does not support capturing stream from canvas element
   return
}
// canvasタグを取得します
const canvas = document.getElementByID('your-canvas-element-ID');
// canvasから15 fpsのビデオストリームをキャプチャします
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];
const localStream = TRTC.createStream({ userId, videoSource: videoTrack });
// ビデオプロパティが外部から渡されたビデオソースと一致していることを確認してください。一致していな
localStream.setVideoProfile('480p');
localStream.initialize().then(() => {
   // local stream initialized success
});
```

再生とレンダリングのカスタマイズ

TRTC.createStream()によって作成および初期化されたローカルストリーム、またはClient.on('stream-added')に よって受信されたリモートストリームの場合、オーディオビデオストリームのオブジェクトの{@link Stream#play Stream.play()}方法を使用してオーディオとビデオを再生とレンダリングできます。Stream.play()の内部は、オー ディオプレーヤーとビデオプレーヤーを自動的に作成し、対応する

```
Appが独自のプレーヤーを使用する場合は、Stream.play()/stop()メソッドを使用せずに呼び出すことができます。
{@link Stream#getAudioTrack Stream.getAudioTrack()}/{@link Stream#getVideoTrack Stream.getVideoTrack()}メ
ソッドを使用して対応するオーディオビデオトラックを取得し、独自のプレーヤーを使用してオーディオビデオ
を再生およびレンダリングします。このようなカスタム再生とレンダリング方法を使用して、Stream.on('player-
state-changed')イベントはトリガーされず、AppはオーディオビデオトラックMediaStreamTrack
の mute/unmute/ended などのイベントを自己監視して、現在のオーディオビデオデータストリームの状態を
判断する必要があります。
また、App層は、 Client.on('stream-added') 、 Client.on('stream-updated') およ
び Client.on('stream-removed') などのイベントを監視して、オーディオビデオストリームのライフサイ
```

クルを処理する必要があります。

ご注意

'stream-added'と'stream-updated'の2つのイベントの処理コールバックでは、オーディオまたはビデオトラックが あるかどうかを確認する必要があります。'stream-updated'イベントの処理では、オーディオまたはビデオトラッ クがある場合、必ずプレーヤーを更新し、最新のオーディオビデオtrackを使用して再生してください。 オンラインの例

メッセージの送受信

最終更新日:::2024-07-19 15:29:07

内容紹介

TRTC SDK はカスタムメッセージを送信する機能を提供します。この機能を介して、キャスターのロールを持つ ユーザーは同じビデオルーム内の他のユーザーに自身のカスタムメッセージをブロードキャストすることができま す。

サポートするプラットフォーム

| iOS | Android | Mac OS | Windows | Electron | web |
|-----|---------|--------|---------|----------|-----|
| 1 | 1 | 1 | ✓ | 1 | × |

送受信の原理

あるユーザーのカスタマイズメッセージは音声ビデオデータストリーム内にクリップされ、音声ビデオデータストリームと同時にルーム内の他のユーザーに送信されます。音声ビデオライン自体は100%信頼できるものではないため、信頼性を高めるため、TRTC SDK 内部に信頼性保護メカニズムを実装しています。



メッセージの送信

TRTCCloudの sendCustomCmdMsg インターフェースを呼び出して送信する場合は、送信時に4つのパラメー タを指定する必要があります:

| パラ メータ 名 | パラメータの説明 |
|----------------|--|
| cmdID | メッセージID。値の範囲は 1 ~ 10、異なるサービスタイプのメッセージには異なるcmdlDを使用 する必要があります。 |
| data | 送信待機メッセージ。最大 1KB(1000バイト)のデータサイズをサポートします。 |
| reliable | 信頼できる送信かどうか。受信者は再送信を待機するため一定時間データを一時的に保存する必 要があることから、信頼できる送信の代償として一定のタイムディレイを受け入れます。 |
| ordered | 順序要求の有無、つまり受信者が受信するデータの順序と送信者が送信する順序が一致すること が要求されるかどうかであり、このことによって、受信側はこれらのメッセージを一時的に保存 し並べ替える必要があるため、一定の受信タイムディレイが生じます。 |

reliable とordered を同時にYES または NOに設定してください。現在のところ、クロス設定はサポートされていません。

Objective-C



```
//カスタマイズメッセージを送信するためのサンプルコード
- (void)sendHello {
    // カスタマイズメッセージのコマンドワードであり、サービスに応じて一連のルールをカスタマイズする
    NSInteger cmdID = 0x1;
    NSData *data = [@"Hello" dataUsingEncoding:NSUTF8StringEncoding];
    // 現在のところ、reliable と ordered を一致させる必要があります。ここではメッセージの送信順
    [trtcCloud sendCustomCmdMsg:cmdID data:data reliable:YES ordered:YES];
}
```

Java



```
//カスタマイズメッセージを送信するためのサンプルコード
public void sendHello() {
   try {
      // カスタマイズメッセージのコマンドワードであり、サービスに応じて一連のルールをカスタマイン
      int cmdID = 0x1;
      String hello = "Hello";
      byte[] data = hello.getBytes("UTF-8");
      // 現在のところ、reliable と ordered を一致させる必要があります。ここではメッセージの
      trtcCloud.sendCustomCmdMsg(cmdID, data, true, true);
    } catch (UnsupportedEncodingException e) {
```







```
// カスタマイズメッセージを送信するためのサンプルコード
void sendHello()
{
    // カスタマイズメッセージのコマンドワードであり、サービスに応じて一連のルールをカスタマイズする
uint32_t cmdID = 0x1;
```

🕗 Tencent Cloud

```
uint8_t* data = { '1', '2', '3' };
uint32_t dataSize = 3; // dataの長さ
// 現在のところ、reliable と ordered を一致させる必要があります。ここではメッセージの送信順
trtcCloud->sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

C#



// カスタマイズメッセージを送信するためのサンプルコード
private void sendHello()

```
// カスタマイズメッセージのコマンドワードであり、サービスに応じて一連のルールをカスタマイズする
uint cmdID = 0x1;
byte[] data = { '1', '2', '3' };
uint dataSize = 3; // dataの長さ
// 現在のところ、reliable と ordered を一致させる必要があります。ここではメッセージの送信順
mTRTCCloud.sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

メッセージの受信

ルーム内の1人のユーザーが sendCustomCmdMsg を介してカスタムメッセージを送信した後、ルーム内の他の ユーザーは SDK コールバックの onRecvCustomCmdMsg インターフェースを介してこれらのメッセージを受 信することができます。

Objective-C



```
//ルーム内の他のユーザーが送信したメッセージの受信および処理
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:(UIn
{
    // userId が送信したメッセージの受信
    switch (cmdId) // 送信者と受信者が承諾済みのcmdId
    {
        case 0:
            // cmdId = 0メッセージを処理
            break;
        case 1:
            // cmdId = 1メッセージを処理
```

Java



```
//TRTCCloudListenerの継承、onRecvCustomCmdMsg メソッドの実装でルーム内の他のユーザーが送信し
public void onRecvCustomCmdMsg(String userId, int cmdId, int seq, byte[] message) {
   // userId が送信したメッセージを受信
   switch (cmdId) // 送信者と受信者が承諾済みのcmdId
   {
   case 0:
      // cmdId = 0メッセージを処理
      break;
   case 1:
      // cmdId = 1メッセージを処理
      break;
   case 2:
      // cmdId = 2メッセージを処理
      break;
   default:
      break;
}
```

C++



🔗 Tencent Cloud

```
// cmdId = 1メッセージを処理
break;
case 2:
    // cmdId = 2メッセージを処理
    break;
default:
    break;
}
```

C#



```
S Tencent Cloud
```

```
// ルーム内の他のユーザーが送信したメッセージの受信および処理
public void onRecvCustomCmdMsg(string userId, int cmdId, uint seq, byte[] msg, uint
{
   // userId が送信したメッセージを受信
   switch (cmdId) // 送信者と受信者が承諾済みのcmdId
   {
   case 0:
       // cmdId = 0メッセージを処理
      break;
   case 1:
       // cmdId = 1メッセージを処理
      break;
   case 2:
      // cmdId = 2メッセージを処理
      break;
   default:
     break;
   }
}
```

使用制限

カスタマイズメッセージは音声ビデオデータよりも送信優先度が高いため、カスタマイズデータの送信が多すぎる と、音声ビデオデータが干渉を受け、画像がフリーズしたりぼやけたりする可能性があります。以上のことから、 カスタマイズメッセージの送信について次の頻度制限を実施しています。

カスタマイズメッセージはクラウドによってルーム内のすべてのユーザーにブロードキャストされるため、送信可 能なメッセージは30通/秒を上限とします。

各メッセージパケット(dataのサイズ)は 1 KBを上限とし、これを超える場合は、中間ルーターまたはサーバー によって破棄される可能性が極めて高くなります。

各クライアントが送信可能なデータは最大8 KB /秒とします。つまり、各データパケットがいずれも1KBであれ ば、最大8個のデータパケット/秒しか送信できません。

サーバーイベントコールバックの監視 サーバーイベントコールバックの監視

最終更新日:::2024-07-19 15:29:07

イベントコールバックサービスは、HTTP/HTTPSリクエストという形でサーバーへのTRTC業務でのイベントの通 知をサポートします。イベントコールバックサービスは、ルームイベントグループ(Room Event)とメディアイ ベントグループ(Media Event)およびレコーディングイベントグループのいくつかのイベントを統合しています (クラウドレコーディング機能のコールバックイベントの説明については、クラウドレコーディングと再生の実 現をご参照ください)。お客様は、関連する設定情報をTencent Cloudに提供することで、このサービスをアク ティブにすることができます。

設定情報

TRTCコンソールでは、自身でのコールバック情報の設定をサポートしており、設定が完了した後、イベントコー ルバック通知を受信できます。詳細な操作ガイドについては、コールバック設定をご参照ください。

注意:

事前に次の情報を準備する必要があります。

必須項目:コールバック通知を受信するためのHTTP/HTTPSサーバーアドレス。

オプション項目:署名を計算するためのkeyで、大文字・小文字アルファベットと数字で構成される最大32文字の keyをカスタマイズできます。

タイムアウト・リトライ

イベントコールバックサーバーがメッセージ通知を送信してから5秒以内にお客様のサーバーから応答を受信しない場合、通知は失敗とみなされます。最初の通知が失敗すると、直ちにリトライが行われます。その後の失敗は、 メッセージがリトライせずに1分以上続くまで、** 10秒**の間隔でリトライし続けます。

イベントコールバックメッセージ形式

イベントコールバックメッセージは、HTTP/HTTPS POSTリクエストとしてサーバーに送信されます。以下がその詳細となります。

文字エンコード形式:UTF-8。

リクエスト: bodyの形式はJSONです。

応答:HTTP STATUS CODE = 200、サーバーは応答パケットの具体的な内容を無視します。プロトコルとの親和 性のため、クライアントの応答内容に、JSON: {"code":0} を付けることをお勧めします。 パケット例:以下の内容は「ルームイベントグループ-入室」イベントのパケット例です。



{

#ルームイベントグループ #入室イベント #コールバック時間。単位はミリ秒

#数字のルーム番号 #イベント発生時間。単位は秒

| | 「UserId」: 「test」、 | #ユーザーID |
|---|--------------------------|----------------------|
| | 「UniqueId」: 1615554 | 1922656、 #固有識別子 |
| | 「Role」: 20、 | #ユーザーロール。キャスター |
| | 「TerminalType」: 3、 | #端末のタイプ。IOS端末 |
| | 「UserType」: 3、 | #ユーザーのタイプ。Native SDK |
| | [[] Reason] : 1 | #入室の原因。正常な入室 |
| } | | |
| } | | |
| | | |

パラメータの説明

コールバックメッセージのパラメータ

イベントコールバックメッセージのheaderには、次のフィールドが含まれています。

| フィールド名 | 值 |
|--------------|--------------------|
| Content-Type | application/json |
| Sign | 署名值 |
| SdkAppId | sdk application id |

イベントコールバックメッセージのbodyには、次のフィールドが含まれています。

| フィールド名 | タイプ | 意味 |
|--------------|----------------|--|
| EventGroupId | Number | イベントグループID |
| EventType | Number | コールバック通知のイベントタイプ |
| CallbackTs | Number | イベントコールバックサーバーからお客様のサーバーに送信されるコールバッ クリクエストのUnixタイムスタンプ。単位はミリ秒 |
| EventInfo | JSON Object | イベント情報 |

イベントグループID

| フィールド名 | 値 | 意味 |
|-------------------|---|--------------|
| EVENT_GROUP_ROOM | 1 | ルームイベントグループ |
| EVENT_GROUP_MEDIA | 2 | メディアイベントグループ |

説明:

レコーディングイベントグループに関する説明については、クラウドレコーディングと再生の実現をご参照くだ さい。

イベントタイプ

| フィールド名 | 値 | 意味 |
|-------------------------|-----|--------------------|
| EVENT_TYPE_CREATE_ROOM | 101 | ルームの作成 |
| EVENT_TYPE_DISMISS_ROOM | 102 | ルームの解散 |
| EVENT_TYPE_ENTER_ROOM | 103 | ルームに参加 |
| EVENT_TYPE_EXIT_ROOM | 104 | ルームを退出 |
| EVENT_TYPE_CHANGE_ROLE | 105 | ロールの切り替え |
| EVENT_TYPE_START_VIDEO | 201 | ビデオデータのプッシュを開始 |
| EVENT_TYPE_STOP_VIDEO | 202 | ビデオデータのプッシュを停止 |
| EVENT_TYPE_START_AUDIO | 203 | オーディオデータのプッシュを開始 |
| EVENT_TYPE_STOP_AUDIO | 204 | オーディオデータのプッシュを停止 |
| EVENT_TYPE_START_ASSIT | 205 | サブストリームデータのプッシュを開始 |
| EVENT_TYPE_STOP_ASSIT | 206 | サブストリームデータのプッシュを停止 |

ご注意:

退室時は104イベントのみコールバックし、202と204イベントはコールバックしません。104イベントには202お よび204イベントが含まれることになります。ビデオ/オーディオを手動でオフにした場合のみ、202/204イベント がコールバックされます。

イベント情報

| フィールド名 | タイプ | 意味 |
|-----------|---------------|----------------------------------|
| RoomId | String/Number | ルーム名(タイプはクライアントのルーム番号タイプと同様) |
| EventTs | Number | イベントで発生するUnixタイムスタンプ。単位は秒(互換性保留) |
| EventMsTs | Number | イベントで発生するUnixタイムスタンプ。単位はミリ秒 |
| Userld | String | ユーザーID |
| | | |


| UniqueId | Number | 固有識別子(option:ルームイベントグループに付帯) クライアントでネットワークの切り替え、異常退出および再入室プロセ スなどの特殊な行為が発生すると、お客様のコールバックサーバーが同 じユーザーの複数回の入室および退室コールバックを受信する可能性が あり、Uniqueldを使用してユーザーの同一回の入退室を識別することが できます |
|--------------|--------|--|
| Role | Number | ロールタイプ(option:入退室時に付帯) |
| TerminalType | Number | 端末のタイプ(option:入室時に付帯) |
| UserType | Number | ユーザーのタイプ(option:入室時に付帯) |
| Reason | Number | 具体的な原因(option:入退室時に付帯) |

ご注意:

「クライアントの特殊な行為による重複コールバックのフィルタリング」ポリシーをリリース済みです。2021年 7月30日より後にコールバックサービスにアクセスした場合、デフォルトで新しいポリシーが適用され、ルームイ ベントグループにはUniqueld(固有識別子)が付加されません。

ロールタイプ

| フィールド名 | 値 | 意味 |
|--------------------|----|-------|
| MEMBER_TRTC_ANCHOR | 20 | キャスター |
| MEMBER_TRTC_VIEWER | 21 | 視聴者 |

端末のタイプ

| フィールド名 | 値 | 意味 |
|-----------------------|-----|-----------|
| TERMINAL_TYPE_WINDOWS | 1 | Windows端末 |
| TERMINAL_TYPE_ANDROID | 2 | Android端末 |
| TERMINAL_TYPE_IOS | 3 | iOS端末 |
| TERMINAL_TYPE_LINUX | 4 | Linux端末 |
| TERMINAL_TYPE_OTHER | 100 | その他 |

ユーザーのタイプ

| フィールド名 | 値 | 意味 |
|--------|---|----|
| | | |

STencent Cloud

| USER_TYPE_WEBRTC | 1 | webrtc |
|----------------------|---|------------|
| USER_TYPE_APPLET | 2 | ミニプログラム |
| USER_TYPE_NATIVE_SDK | 3 | Native SDK |

具体的な原因

| フィールド名 | 意味 |
|--------|--|
| 入室 | 1:正常な入室 2:Networkingの切り替え 3:タイムアウト・リトライ 4:ルーム間マイク接続による入室 |
| 退室 | 1:正常な退室 2:タイムアウトによる退室 3:ルームのユーザーが削除される 4:マイク接続のキャンセルによる退室 5:強制終了 注意:Androidシステムではプロセスの強制終了を捕捉できず、バックエンドのタイムア ウトによる退室を待つしかありません。この場合のコールバックreasonは2となります |

署名計算

署名はHMAC SHA256暗号化アルゴリズムによって算出されます。イベントコールバック受信サーバーがコール バックメッセージを受信した後、同じ方法で署名が算出されます。同様に、Tencent CloudのTRTCのイベントコー ルバックであり、偽造されていないことを意味します。署名の計算は次のとおりです。



//署名signの計算式のkeyは、署名signの計算に用いられる暗号化鍵です。 Sign = base64(hmacsha256(key, body))

ご注意:

bodyはお客様が受信したコールバックリクエストのオリジナルパケットです。いかなる変更も行わないでください。例:



body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback

アクセス管理

概要

最終更新日:::2024-07-19 15:29:07

ご注意:

ここでは主に**Tencent Real-Time Communication(TRTC)**のCloud Access Management(CAM)機能関連コンテン ツについて紹介します。他製品のCAM関連コンテンツについては、CAMサポート製品をご参照ください。 CAM (Cloud Access Management、CAM) は、Tencent Cloudが提供するWebサービスであり、主にお客様が Tencent Cloudアカウントのリソースへのアクセス権限を安全に管理するのに役立ちます。CAMを使用すると、 ユーザー(グループ)を作成、管理、および廃棄でき、ID管理とポリシー管理を介して、Tencent Cloudリソース へのアクセスと使用が許可されるユーザーを制御できます。

TRTCはCAMに接続済みであり、開発者は自身の必要に応じてサブアカウントに適切な TRTCアクセス権限を割り 当てることができます。

はじめに

TRTC CAMを使用する前に、 CAMとTRTCの基本コンセプトについて理解する必要があります。関連する主なコ ンセプトは次のとおりです。 CAM関連:ユーザー、ポリシー

TRTC関連:アプリケーション、SDKAppID

適用ケース

Tencent Cloudクラウドサービスのディメンションに対する権限隔離

ある企業内にTencent Cloudを使用する部門が複数存在し、その中のA部門がTRTCへの接続のみを担当する場合。 A部門のスタッフはTRTCにアクセスする権限を持っている必要がありますが、 他のTencent Cloud製品にアクセ スする権限を持つことはできません。この企業はルートアカウントを介してA部門のためにサブアカウントを作成 し、このサブアカウントにTRTC関連権限のみを付与し、その後、このサブアカウントをA部門が使用するよう提 供します。

TRTCアプリケーションのディメンションに対する権限隔離

ある企業内の複数業務においてTRTCを使用する場合、相互間の隔離を行う必要があります。隔離にはリソースの 隔離と権限の隔離という2つの面があり、前者はTRTCアプリケーションシステムによって提供され、後者は TRTC CAMによって実現されます。この企業は業務ごとにサブアカウントを作成し、関連のTRTCアプリケーション権限を付与し、それぞれの業務と関連のあるアプリケーションにしかアクセスできないようにします。

TRTC操作のディメンションに対する権限隔離

ある企業の1つの業務でTRTCを使用する場合、その業務の製品オペレーターは稼働統計データを取得するため、 TRTCコンソールにアクセスする必要がありますが、誤操作によって業務に影響を及ぼすことを回避するため、機 密性の高い操作(Relayed Pushの変更、クラウドレコーディングのコンフィグレーションなど)を行うことを許 可されていません。この場合は、まずTRTCコンソールのログインと稼働統計関連APIにアクセスする権限を持つ カスタムポリシーを作成し、その後、サブアカウントを作成し、上述のポリシーとのバインディングを行い、こ のサブアカウントを製品オペレーターに提供します。

権限の粒度

CAMのコア機能は、特定のアカウントが特定のリソースに対して何らかの操作を実行することを許可または禁止 すると表すことができます。TRTC CAMでは、リソースレベルの承認をサポートし、リソースの粒度はTRTCアプ リケーションです。操作の粒度はTencent Cloud APIであり、サーバーAPIとTRTCコンソールにアクセスするとき に使用する可能性のあるAPIを含みます。詳しい説明は、権限付与可能なリソースと操作をご参照ください。

機能の制限

TRTC CAMのリソース粒度は アプリケーションであり、より細かい粒度のリソース(アプリケーション情報、コ ンフィグレーション情報など)に対する承認をサポートしていません。

TRTC CAMではプロジェクトをサポートしていません。タグを用いて、クラウドサービスのリソースを管理する ことを推奨します。

承認されるリソースおよび操作

最終更新日:::2024-07-19 15:29:07

ご注意:

ここでは主に**Tencent Real-Time CommunicationTRTC** のCloud Access Management(CAM)機能関連コンテ ンツについて紹介します。他製品のCAM関連コンテンツについては、CAMサポート製品をご参照ください。 CAMのコア機能は、**特定のアカウントが特定のリソースに対して何らかの操作を実行することを許可または禁止 する**と表すことができます。TRTC CAMは、リソースレベルの承認をサポートし、リソースの粒度はTRTC アプ リケーション、操作の粒度はTencent Cloud APIであり、サーバーAPIとTRTC コンソールにアクセスするときに使 用する可能性のあるAPIを含みます。

TRTC CAMを管理する必要がある場合、Tencent Cloud ルートアカウントにログインし、プリセットポリシーまた はカスタムポリシーを使用して、具体的な認証操作を完了します。

承認可能なリソースタイプ

TRTC CAMが承認可能なリソースタイプは、アプリケーションです。

リソースレベルの承認をサポートするAPI

一部のリソースレベルの承認をサポートしないAPIを除いて、このセクションにリストアップされているAPI操作 は、リソースレベルの承認をサポートします。 アクセスポリシー構文のこれらのAPI操作の**リソース構文の記述** は、いずれも同じです。具体的には次のとおりです。

すべてのアプリケーションアクセス権限を承認する: qcs::trtc::uin/\${uin}:sdkappid/* 。 単一のアプリケーションアクセス権限を承認する: qcs::trtc::uin/\${uin}:sdkappid/\${SdkAppId} 。

サーバーAPIの操作

| インターフェース名 | インターフェースの分 類 | 機能の説明 |
|------------------------|-----------------|--------------------------|
| DismissRoom | ムール管理 | ムールの解散 |
| RemoveUser | ルーム管理 | ユーザーの削除 |
| RemoveUserByStrRoomId | ルーム管理 | ユーザーの削除(文字列のルームナン バー) |
| DismissRoomByStrRoomId | ルーム管理 | ルームの解散(文字列のルームナン バー) |

| StartMCUMixTranscode | ミクスストリーミング トランスコード | クラウドミクスストリーミングの起動 |
|---------------------------------|-----------------------|------------------------------------|
| StopMCUMixTranscode | ミクスストリーミング トランスコード | Cloud MixTranscodingの終了 |
| StartMCUMixTranscodeByStrRoomId | ミクスストリーミング トランスコード | クラウドミクスストリーミングの起動 (文字列のルームナンバー) |
| StopMCUMixTranscodeByStrRoomId | ミクスストリーミング トランスコード | クラウドミクスストリーミングの終了 (文字列のルームナンバー) |
| CreateTroubleInfo | 通話品質の監視 | 異常情報の作成 |
| DescribeAbnormalEvent | 通話品質の監視 | 異常体験イベントの確認 |
| DescribeCallDetail | 通話品質の監視 | ユーザーリストと通話インジケータの確 認 |
| DescribeHistoryScale | 通話品質の監視 | ルーム履歴とユーザー数の確認 |
| DescribeRoomInformation | 通話品質の監視 | ルームリストの確認 |
| DescribeUserInformation | 通話品質の監視 | 過去のユーザーリストの照会 |

コンソールAPIの操作

| インターフェース名 | 使用モジュール | 機能の説明 |
|---------------------|---|----------------|
| DescribeAppStatList | TRTCコンソール 概要 使用量の統計 監視ダッシュボード 開発支援 > UserSig 生成 &検証 アプリケーション管理 | アプリケーションリストの取得 |
| DescribeSdkAppInfo | TRTCコンソールアプリ ケーション管理 > アプリ ケーション情報 | アプリケーション情報の取得 |
| ModifyAppInfo | TRTCコンソールアプリ ケーション管理 > アプリ ケーション情報 | アプリケーション情報の編集 |
| ChangeSecretKeyFlag | TRTCコンソールアプリ | 権限キーのステータスの変更 |



| | ケーション管理 > アプリ ケーション情報 | |
|---|--|--|
| CreateWatermark | TRTCコンソールアプリ ケーション管理 > 素材管 理 | 画像のアップロード |
| DeleteWatermark | TRTCコンソールアプリ ケーション管理 > 素材管 理 | 画像の削除 |
| ModifyWatermark | TRTCコンソールアプリ ケーション管理 > 素材管 理 | 画像の編集 |
| DescribeWatermark | TRTCコンソールアプリ ケーション管理 > 素材管 理 | 画像の検索 |
| CreateSecret | TRTCコンソール アプリ ケーション管理 > クイッ クマスター | 対称暗号化鍵の作成 |
| ToggleSecretVersionTRTCコンソールアプリ ケーション管理 > クイッ クマスター | | キーバージョンの切り替え(公開鍵・秘密 鍵/対称暗号化鍵) |
| DescribeSecret | <pre>TRTCコンソール 開発支援 > Demoクイッ クスタート 開発支援 > UserSig 生成 &検証 アプリケーション管理 > クイックマスター</pre> | 対称暗号化鍵の取得 |
| DescribeTrtcAppAndAccountInfo | TRTCコンソール開発支 援 > UserSig 生成&検証 | アプリケーションとアカウント情報、公開 鍵・秘密鍵の取得 |
| CreateSecretUserSig | TRTCコンソール開発支 援 > UserSig 生成&検証 | 対称暗号化鍵を使用してUserSigを生成 |
| DescribeSig | TRTCコンソール 開発支援 > UserSig 生成 &検証 | 旧バージョンの公開鍵・秘密鍵を使用して 生成されたUserSigを取得 |



| | アプリケーション管理 > クイックマスター | |
|---------------------|------------------------------------|--|
| VerifySecretUserSig | TRTCコンソール開発支 援 > UserSig 生成&検証 | 対称暗号化鍵生成の UserSig 検証 |
| VerifySig | TRTCコンソール開発支 援 > UserSig 生成&検証 | 公開鍵・秘密鍵で生成されたUserSig検証 |
| CreateSpearConf | TRTCコンソールアプリ ケーション管理 > 画面設 定 | 画面設定の構成を追加しました。 この機 能設定カードは、iLiveSDK1.9.6以前の バージョンでのみ表示されます。TRTC SDK 6.0以降のバージョンについては、画 質設定 をご参照ください |
| DeleteSpearConf | TRTCコンソールアプリ ケーション管理 > 画面設 定 | 画面設定の構成を削除しました。 この機 能設定カードは、iLiveSDK1.9.6以前の バージョンでのみ表示されます。TRTC SDK 6.0以降のバージョンについては、画 質設定 をご参照ください |
| ModifySpearConf | TRTCコンソールアプリ ケーション管理 > 画面設 定 | 画面設定の構成を変更しました。 この機 能設定カードは、iLiveSDK1.9.6以前の バージョンでのみ表示されます。TRTC SDK 6.0以降のバージョンについては、画 質設定 をご参照ください |
| DescribeSpearConf | TRTCコンソールアプリ ケーション管理 > 画面設 定 | 画面設定の構成を取得しました。 この機 能設定カードは、iLiveSDK1.9.6以前の バージョンでのみ表示されます。TRTC SDK 6.0以降のバージョンについては、画 質設定 をご参照ください |
| ToggleSpearScheme | TRTCコンソールアプリ ケーション管理 > 画面設 定 | 画面設定のシナリオを切り替えました。 この機能設定カードは、iLiveSDK1.9.6以 前のバージョンでのみ表示されます。 TRTC SDK 6.0以降のバージョンについて は、画質設定をご参照ください |

リソースレベルの承認をサポートしないAPI

特殊な制限により、次のAPIはリソースレベルの承認をサポートしていません。

サーバーAPIの操作

| インターフェース名 | インター フェース の分類 | 機能の説明 | 特殊な制限の説明 |
|------------------------------|---------------------|----------------|--|
| DescribeDetailEvent | 通話品質 の監視 | 詳細なイベントを 取得 | 入力パラメータに SDKApplD がな いため、リソースレベルの認証を 実行できません。 |
| DescribeRecordStatistic | その他イ | クラウドレコー | 業務上の理由により、現在のとこ |
| | ンター | ディング課金時間 | ろ、リソースレベルの承認をサ |
| | フェース | の照会 | ポートしていません |
| DescribeTrtcInteractiveTime | その他イ | インタラクティブ | 業務上の理由により、現在のとこ |
| | ンター | オーディオビデオ | ろ、リソースレベルの承認をサ |
| | フェース | 課金時間の照会 | ポートしていません |
| DescribeTrtcMcuTranscodeTime | その他イ | バイパストランス | 業務上の理由により、現在のとこ |
| | ンター | コード課金時間の | ろ、リソースレベルの承認をサ |
| | フェース | 照会 | ポートしていません |

コンソールAPIの操作

| インターフェース名 | 使用モ ジュール | 機能の説明 | 特殊な制限の説明 |
|--------------------------|------------------------------------|-------------------------------|--|
| DescribeTrtcStatistic | TRTCコン ソール 概要 使用量の統 計 | 課金期間中 の使用量統 計データの 取得 | このインターフェースには、完全な SDKAppIDを返す統計データが含まれていま す。不完全なSDKAppIDを制限すると、エ ラーが返されます。必要に応じて、 DescribeAppStatListインターフェースを介し てクエリーできるアプリケーションのリスト を制限できます |
| DescribeDurationPackages | TRTCコン ソール 概要 パッケージ 管理 | 前払いパッ ケージリス トの取得 | 前払いパッケージは単一のTencent Cloudア カウントですべてのTRTCアプリによって共 有されます。パッケージ情報にSDKAppIDパ ラメータがないため、リソースレベルの認証 を実行できません |
| GetUserList | TRTCコン ソール 監視ダッ シュボード | ユーザーリ ストの取得 | 入力パラメータにSDKApplDがないため、リ ソースレベルの認証を実行できません。必 要に応じて、DescribeAppStatListインター フェースを介してクエリーできるアプリケー ションのリストを制限できます |
| GetUserInfo | TRTCコン | ユーザー情 | 入力パラメータにSDKAppIDがないため、リ |



| | ソール 監視ダッ シュボード | 報の取得 | ソースレベルの認証を実行できません。必 要に応じて、DescribeAppStatListインター フェースを介してクエリーできるアプリケー ションのリストを制限できます |
|----------------------|--|---|---|
| GetCommState | TRTCコン ソール 監視ダッ シュボード | 通話情報の 取得 | 入力パラメータにSDKAppIDがないため、リ ソースレベルの認証を実行できません。必 要に応じて、DescribeAppStatListインター フェースを介してクエリーできるアプリケー ションのリストを制限できます |
| GetElasticSearchData | TRTCコン ソール 監視ダッ シュボード | ESデータの クエリー | 入力パラメータにSDKApplDがないため、リ ソースレベルの認証を実行できません。必 要に応じて、DescribeAppStatListインター フェースを介してクエリーできるアプリケー ションのリストを制限できます |
| CreateTrtcApp | <pre>TRTCコン ソール 開発支援> Demoクイッ クスタート アプリケー ション管理</pre> | TRTCアプ リケーショ ンの作成 | 入力パラメータにSDKApplDがないため、リ ソースレベルの認証を実行できません。 SDKApplDはTRTCアプリケーションの一意 の識別子であり、SDKApplD情報はアプリ ケーションの作成後にのみ利用できます |
| HardDescribeMixConf | <pre>TRTCコン ソール アプリケー ション管理> 機能設定</pre> | Auto- Relayed Pushのス テータスの クエリー | 入力パラメータにSDKApplDがないため、リ ソースレベルの認証を実行できません。必 要に応じて、DescribeAppStatListインター フェースを介してクエリーできるアプリケー ションのリストを制限できます |
| ModifyMixConf | TRTC コン ソール アプリケー ション管理 > 機能設定 | Auto- Relayed Pushのオ ン/オフ | 入力パラメータにSDKApplDがないため、リ ソースレベルの認証を実行できません。必 要に応じて、DescribeAppStatListインター フェースを介してクエリーできるアプリケー ションのリストを制限できます |
| RemindBalance | TRTCコン ソール パッケージ 管理 | 前払いパッ ケージ残高 アラート情 報の取得 | 前払いパッケージは単一のTencent Cloudア カウントですべてのTRTCアプリによって共 有されます。パッケージ情報にSDKAppIDパ ラメータがないため、リソースレベルの認証 を実行できません |

ご注意:

リソースレベルの承認をサポートしないAPI操作の場合でも、カスタムポリシーを介してユーザーに操作を使用す る権限を付与できます。ただし、ポリシーステートメントのリソース要素は、必ず * と指定する必要がありま す。

プリセットポリシー

最終更新日:::2024-07-19 15:29:07

ご注意:

ここでは主に**Tencent Real-Time CommunicationTRTC**のCloud Access Management(CAM)機能関連コンテン ツについて紹介します。他製品のCAM関連コンテンツについては、CAMサポート製品をご参照ください。

TRTCのCAMは、実質的にはサブアカウントをポリシーにバインドするか、またはポリシーをサブアカウントに付 与します。開発者は、コンソールのプリセットポリシーを直接使用して、いくつかの簡単な承認操作を行うことが できます。複雑な権限付与の操作については、カスタムポリシーをご参照ください。

TRTCは現在、次のようなプリセットポリシーを提供しています。

| ポリシー名 | ポリシーの説明 |
|--------------------------|--------------------------|
| QcloudTRTCFullAccess | TRTCへの完全な読み取り/書き込みアクセス権限 |
| QcloudTRTCReadonlyAccess | TRTCへの読み取り専用アクセス権限 |

プリセットポリシー使用例

TRTCへの完全な読み取り/書き込みアクセス権限を有するサブアカウントの作成

1. Tencent Cloudルートアカウントとして、CAMコンソールの【ユーザーリスト】にアクセスし、【ユーザーの作 成】をクリックします。

2. 「ユーザーの作成」ページで【カスタム作成】を選択し、「サブユーザーの作成」ページに進みます。

説明:

CAMサブユーザーのカスタム作成の操作ガイドに従って、「ユーザー権限の設定」までの手順を完了してください。

3. 「ユーザー権限の設定」ページにおいて、次の事項を実施します。

3.1 プリセットポリシー QcloudTRTCFullAccess を検索してチェックを入れます。

3.2 【次のステップ】をクリックします。

4. 「情報と権限のチェック」フィールドの下にある【完了】をクリックして、サブユーザーの作成を完了しま

す。成功ページで、サブユーザーのログインリンクとセキュリティ証明書をダウンロードして保管します。そこに は、次のような情報が含まれます。

| 情報 | ソース | 機能 | 保存が必須かど うか |
|-------------|---------|---|---------------|
| ログイン リンク | ページでコピー | コンソールにログインするのに有用、ルートアカウ ントを入力する手順を省略 | いいえ |
| | | | |

| ユーザー 名 | セキュリティ証明 書CSVファイル | コンソールにログインする時に入力 | はい |
|-----------|----------------------|---|----|
| パスワー ド | セキュリティ証明 書CSVファイル | コンソールへのログイン時に入力 | はい |
| SecretId | セキュリティ証明 書CSVファイル | サーバーAPI呼び出し時に使用します。詳細はアク セスキー をご参照ください | はい |
| SecretKey | セキュリティ証明 書CSVファイル | サーバーAPI呼び出し時に使用します。詳細はアク セスキー をご参照ください | はい |

5. 前述のログインリンクとセキュリティ証明書を許可された当事者に提供します。許可された当事者は、サブ ユーザーを使用して、TRTCコンソールへのアクセスやTRTCサーバーAPIのリクエストなど、TRTCでのすべての 操作を実行できます。

TRTCへの完全な読み取り/書き込みアクセス権限を既存のサブアカウントに付与

1. Tencent Cloudルートアカウントとして、CAMコンソールの【ユーザーリスト】にアクセスし、権限を付与した いサブアカウントをクリックします。

2.「ユーザー詳細」ページの権限フィールドで、【ポリシーの追加】をクリックします。サブアカウントの権限 が空でない場合は、【ポリシーの関連付け】をクリックします。

3. 【ポリシーリストからポリシーの関連付けを選択】を選択し、プリセットポリ

シー QcloudTRTCFullAccess を検索しチェックを入れます。その後、ページのプロンプトに従って承認プロ セスを完了してください。

サブアカウントのTRTCへの完全な読み取り/書き込みアクセス権限を解除

1. Tencent Cloudルートアカウントとして、CAMコンソールの【ユーザーリスト】にアクセスし、権限を解除した いサブアカウントをクリックします。

2. 「ユーザー詳細」ページの権限タブでプリセットポリシー QcloudTRTCFullAccess を検索し、右側の【解除】をクリックします。プロンプトに従って承認解除プロセスを完了してください。

カスタムポリシー

最終更新日:::2024-07-19 15:29:07

ご注意:

ここでは主に**Tencent Real-Time CommunicationTRTC**のCloud Access Management(CAM)機能関連コンテン ツについて紹介します。他製品のCAM関連コンテンツについては、CAMサポート製品をご参照ください。 TRTCのCAMにおいてプリセットポリシーを使用して権限付与を行うのは簡便ではありますが、プリセットポリ シーは権限制御の粒度が比較的粗く、TRTCアプリケーションやTencent Cloud APIの粒度まで細分化することがで きません。きめ細かい権限制御機能を必要とする場合は、カスタムポリシーを作成する必要があります。

カスタムポリシーの作成方法

カスタムポリシーを作成するには、さまざまな方法があります。次の表に、それぞれの方法の比較を示します。具 体的な操作手順については、以下の文章をご参照ください。

| エントリー の作成 | 作成方法 | 効果 (Effect) | リソース (Resource) | 操作 (Action) | 柔軟 性 | 難易度 |
|----------------|-----------------|----------------|--------------------|----------------|---------|-----|
| CAMコン ソール | ポリシージェネ レーター | 手動選択 | 構文の記述 | 手動選択 | 中 | 中 |
| CAMコン ソール | ポリシー構文 | 構文の記 述 | 構文の記述 | 構文の記述 | 高 | 高 |
| CAMサー バーAPI | CreatePolicy | 構文の記 述 | 構文の記述 | 構文の記述 | 高 | 高 |

説明:

TRTCは、製品の機能またはアイテムに応じたカスタムポリシーの作成は**サポートしていません**。 **手動選択**とは、ユーザーがコンソールに表示される候補リストからオブジェクトを選択することを意味します。 **構文の記述**とは、アクセスポリシー構文によってオブジェクトを記述することを意味します。

アクセスポリシー構文

リソース構文の記述

上記のように、TRTC権限管理のリソース粒度はアプリケーションです。アプリケーションのポリシー構文の記述 法は、CAMリソース記述法に従います。以下の例では、開発者のルートアカウントIDは12345678で、開発者は、 SDKAppIDがそれぞれ1400000000、1400000001および1400000002のアプリケーションを3つ作成しています。



TRTCのすべてのアプリケーションのポリシー構文の記述



```
"resource": [
   "qcs::trtc::uin/12345678:sdkappid/*"
]
```

単一アプリケーションのポリシー構文の記述



```
"resource": [
   "qcs::trtc::uin/12345678:sdkappid/140000001"
]
```

複数アプリケーションのポリシー構文の記述



```
"resource": [
   "qcs::trtc::uin/12345678:sdkappid/140000000",
   "qcs::trtc::uin/12345678:sdkappid/1400000001"
]
```

操作構文の記述

上記のとおり、TRTCの権限管理の操作粒度はTencent Cloud APIです。詳細については、権限付与可能なリソース と操作をご参照ください。次の例では、 DescribeAppStatList (アプリケーションリストの取



得)、 DescribeSdkAppInfo (アプリケーション情報の取得)などのTencent Cloud APIを例として取り上げ ています。

TRTCのすべてのTencent Cloud APIのポリシー構文の記述



```
"action": [
   "name/trtc:*"
]
```

単一のTencent Cloud APIを操作する際のポリシー構文の記述





```
"action": [
    "name/trtc:DescribeAppStatList"
]
```

複数のTencent Cloud APIを操作する際のポリシー構文の記述





```
"action": [
    "name/trtc:DescribeAppStatList",
    "name/trtc:DescribeTrtcAppAndAccountInfo"
]
```

カスタムポリシー使用例

ポリシージェネレーターの使用

- 以下の例では、カスタムポリシーを作成します。このポリシーはサーバーAPI RemoveUser を除くすべての操作 を、TRTCアプリケーション1400000001で実行できます。
- 1. Tencent Cloudルートアカウントとして、CAMコンソールの【ポリシー】にアクセスし、【カスタムポリシーの 新規作成】をクリックします。
- 2. 【ポリシージェネレーターで作成】を選択して、ポリシー作成ページに進みます。
- 3. サービスと操作を選択します。
- 【効果(Effect)】設定項目は、【許可】を選択します。
- 【サービス(Service)】設定項目は、【TRTC】を選択します。
- 【操作(Action)】設定項目は、すべての項目にチェックを入れます。
- 【リソース(Resource)】設定項目には、リソース構文の記述の説明に従っ
- て qcs::trtc::uin/12345678:sdkappid/1400000001 を入力します。
- 【条件(Condition)】設定項目は設定不要です。

【ステートメントの追加】をクリックすると、ページの一番下に「TRTCアプリケーション1400000001に対する あらゆる操作を許可する」というステートメントが表示されます。

- 4. 同じページに別のステートメントを続けて追加します。
- 【効果(Effect)】設定項目は、【拒否】を選択します。
- 【サービス(Service)】設定項目は、【TRTC】を選択します。

【操作(Action)】設定項目は、 RemoveUser (検索機能で速やかに見つけられます)にチェックを入れます。

- 【リソース(Resource)】設定項目には、リソース構文の記述の説明に従っ
- て qcs::trtc::uin/12345678:sdkappid/1400000001 を入力します。

【条件(Condition)】設定項目は設定不要です。

【ステートメントの追加】をクリックすると、ページの一番下に「TRTCアプリケーション1400000001に対する RemoveUser 操作を拒否する」というステートメントが表示されます。

- 5.【次のステップ】をクリックし、必要に応じてポリシー名を変更します(または変更しなくてもかまいません)。
- 6. 【完了】をクリックし、カスタムポリシーの作成を完了します。

その後、このポリシーを他のサブアカウントに付与する方法は、TRTCの完全な読み取り/書き込みアクセス権限 を既存のサブアカウントに付与すると同様です。

ポリシー構文の使用

以下の例では、カスタムポリシーを作成します。このポリシーは、1400000001と1400000002という2つのTRTC サブアプリケーションですべての操作を実行できますが、1400000001の PRemoveUser 操作は拒否します。

1. Tencent Cloudルートアカウントとして、CAMコンソールの【ポリシー】にアクセスし、【カスタムポリシーの 新規作成】をクリックします。

- 2. 【ポリシー構文で作成】を選択し、ポリシー作成ページに進みます。
- 3. 【テンプレートタイプの選択】ボックスで【空白テンプレート】を選択します。

説明:

ポリシーテンプレートは、新しいポリシーが既存のポリシー(プリセットポリシーまたはカスタムポリシー)を コピーしてから、それをベースとして調整が行われることを意味します。実際の使用においては、開発者は状況に 応じて適切なポリシーテンプレートを選択することで、ポリシー内容の難しい入力と作業の負荷を軽減すること ができます。

4. 【次のステップ】をクリックし、必要に応じてポリシー名を変更します(または変更しなくてもかまいません)。

5. 【ポリシー内容の編集】ボックスにポリシー内容を入力します。この例のポリシーの内容は次のとおりです。



"version": "2.0",



```
"statement": [
     {
         "effect": "allow",
         "action": [
             "name/trtc:*"
         1,
         "resource": [
             "qcs::trtc::uin/12345678:sdkappid/1400000001",
             "qcs::trtc::uin/12345678:sdkappid/140000002"
         1
     },
     {
         "effect": "deny",
         "action": [
             "name/trtc:RemoveUser"
         ],
         "resource": [
             "qcs::trtc::uin/12345678:sdkappid/1400000001"
         1
     }
]
}
```

説明:

ポリシーの内容は、CAMポリシー構文ロジックに従う必要があります。リソースと操作という2つの要素の構文 は、それぞれ上述のリソース構文の記述と操作構文の記述のとおりです。

6. 【ポリシーの作成】をクリックして、カスタムポリシーの作成を完了します。

その後、このポリシーを他のサブアカウントに付与する方法は、TRTCの完全な読み取り/書き込みアクセス権限 を既存のサブアカウントに付与すると同様です。

CAMが提供するサーバーAPIの使用

ほとんどの開発者にとって、コンソールで権限管理操作が完了すれば、ビジネスニーズが満たされたことになり ます。ただし、権限管理機能を自動化・システム化する必要がある場合は、サーバーAPIを使用することができま す。

ポリシー関連のサーバーAPIはCAMに属します。詳細については、CAM公式ウェブサイトドキュメントをご参照 ください。ここには、いくつかの主なインターフェースだけをリストアップしています。

- ポリシーの作成
- ポリシーの削除
- ユーザーへのポリシーのバインド
- ユーザーにバインドしたポリシーの解除