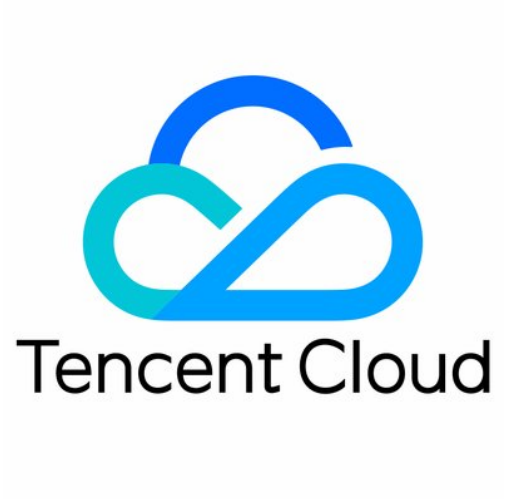
Instant Messaging

Scene Practice

Product Documentation



Contents

Scene Practice
TWebLive Components

Last updated : 2020-11-03 11:10:59

1. Introduction to TWebLive

TWebLive is a Tencent Cloud web ILVB component. As a new SDK developed by the Tencent Cloud terminal development team, it integrates Tencent Cloud TRTC, Tencent Cloud IM, and the Tencent Cloud superplayer TCPlayer. It provides common features in web interactive livestreaming scenarios, including push, enabling/disabling the microphone, enabling/disabling the camera, WeChat video sharing, chat, and likes. It also contains simple and easy-to-use APIs for implementing web push/pull, real-time chat interaction, and other features.

2. TWebLive Advantages

Developers can use this SDK to completely replace the flash push solution, which greatly reduces the complexity and time for implementing web push, low-latency web streaming, CDN streaming, and real-time chat interactions (or on-screen comments). For more information, see the following examples.

1. Push

To use the push feature, you must create a pusher object. The simplest push operation takes only three steps, as shown below.

```
<div id="pusherView" style="width:100%; height:auto;"></div>
<script>
// 1. Create a pusher object.
let pusher = TWebLive.createPusher({ userID: 'your userID' });

// 2. Configure the rendering view, use the microphone to capture audio, and use the camera to ca
pture video (720p by default).
pusher.setRenderView({
elementID: 'pusherView',
audio: true,
video: true
}).then(() => {.then(() => {
// 3. Enter the sdkappid, roomid, and other information for push.
```

```
// The URL must start with `room://`
let url = `room://sdkappid=${SDKAppID}&roomid=${roomID}&userid=${userID}&usersig=${userSig}&lived
omainname=${liveDomainName}&streamid=${streamID}`;
pusher.startPush(url).then(() => {
console.log('pusher | startPush | ok');
});
}).catch(error => {
console.error('pusher | setRenderView | failed', error);
});
</script>
```

2. Pull

To use the pull feature, you must create a player object. The simplest pull operation takes only three steps, as shown below.

```
<div id="playerView" style="width:100%; height:auto;"></div>
<script>
// 1. Create a player object.
let player = TWebLive.createPlayer();

// 2. Configure the rendering view.
player.setRenderView({ elementID: 'playerView' });

// 3. Enter the FLV and HLS addresses and other required information for pulling CDN streams for
playback. The URL must start with `https://`.
// Alternatively, enter the sdkappid, roomid, and other information for pulling WebRTC low-latenc
y streams for playback. The URL must start with `room://`.
let url = 'https://'
+ 'flv=https://200002949.vod.myqcloud.com/200002949_b6ffc.f0.flv' + '&' // Replace the URL with a
n actual playback address
+ 'hls=https://200002949.vod.myqcloud.com/200002949_b6ffc.f0.m3u8' // Replace the URL with an act
ual playback address

// let url = `room://sdkappid=${SDKAppID}&roomid=${roomID}&userid=${userID}&usersig=${userSig}`;
player.startPlay(url).then(() => {
console.log('player | startPlay | ok');
}).catch((error) => {
console.error('player | startPlay | failed', error);
});
</script>
```

3. Livestreaming interaction

To enable the host to chat with the audience, you must create an IM object. The simplest message receiving and sending process takes only three steps, as shown below.

```
// 1. Create an IM object and monitor events.
let im = TWebLive.createIM({
SDKAppID: 0 // Replace 0 with the SDKAppID of your IM instance
});
// Monitor IM_READY IM_TEXT_MESSAGE_RECEIVED and other events
let onIMReady = function(event) {
im.sendTextMessage({ roomID: 'your roomID', text: 'hello from TWebLive' });
};
let onTextMessageReceived = function(event) {
event.data.forEach(function(message) {
console.log((message.from || message.nick) + ' : ', message.payload.text);
});
};
// Monitor this event on the access side, and then you can call the SDK to send messages or perfo
rm other operations
im.on(TWebLive.EVENT.IM_READY, onIMReady);
// Receive a text message and display it on the screen
im.on(TWebLive.EVENT.IM_TEXT_MESSAGE_RECEIVED, onTextMessageReceived);

// 2. Log in to the IM console.
im.login({userID: 'your userID', userSig: 'your userSig'}).then((imResponse) => {
console.log(imResponse.data); // Logged in successfully
if (imResponse.data.repeatLogin === true) {
// This indicates that the account has been logged in and that the current login is a repeated lo
gin
console.log(imResponse.data.errorInfo);
}
}).catch((imError) => {
console.warn('im | login | failed', imError); // Information about the login failure
});

// 3. Enter the chat room.
im.enterRoom('your roomID').then((imResponse) => {
switch (imResponse.data.status) {
case TWebLive.TYPES.ENTER_ROOM_SUCCESS: // Entered the chat room successfully
break;
case TWebLive.TYPES.ALREADY_IN_ROOM: // You are already in the chat room
break;
default:
break;
}
}).catch((imError) => {
console.warn('im | enterRoom | failed', imError); // Information about the failure to enter the c
hat room
});
</script>
```

To further reduce developers' development and labor costs, in addition to the TWebLive SDK, we provide an open-source demo on GitHub, which is adaptable to both PC and mobile browsers. Developers can fork and clone a project to their local devices and make simple modifications to run the demo. Alternatively, they can integrate the demo into their own projects for deployment and launch.

3. Using TWebLive

Create a TRTC instance in the Tencent Cloud TRTC console and save the SDKAPPID. At the same time, an IM instance with the same SDKAppID will be automatically created. Then, choose **App Management** > **Feature Configuration** and enable automatic relayed push. After relayed push is enabled, each video stream in the TRTC room is assigned a playback URL. (If CDN livestreaming is not required, you do not need to enable relayed push.)

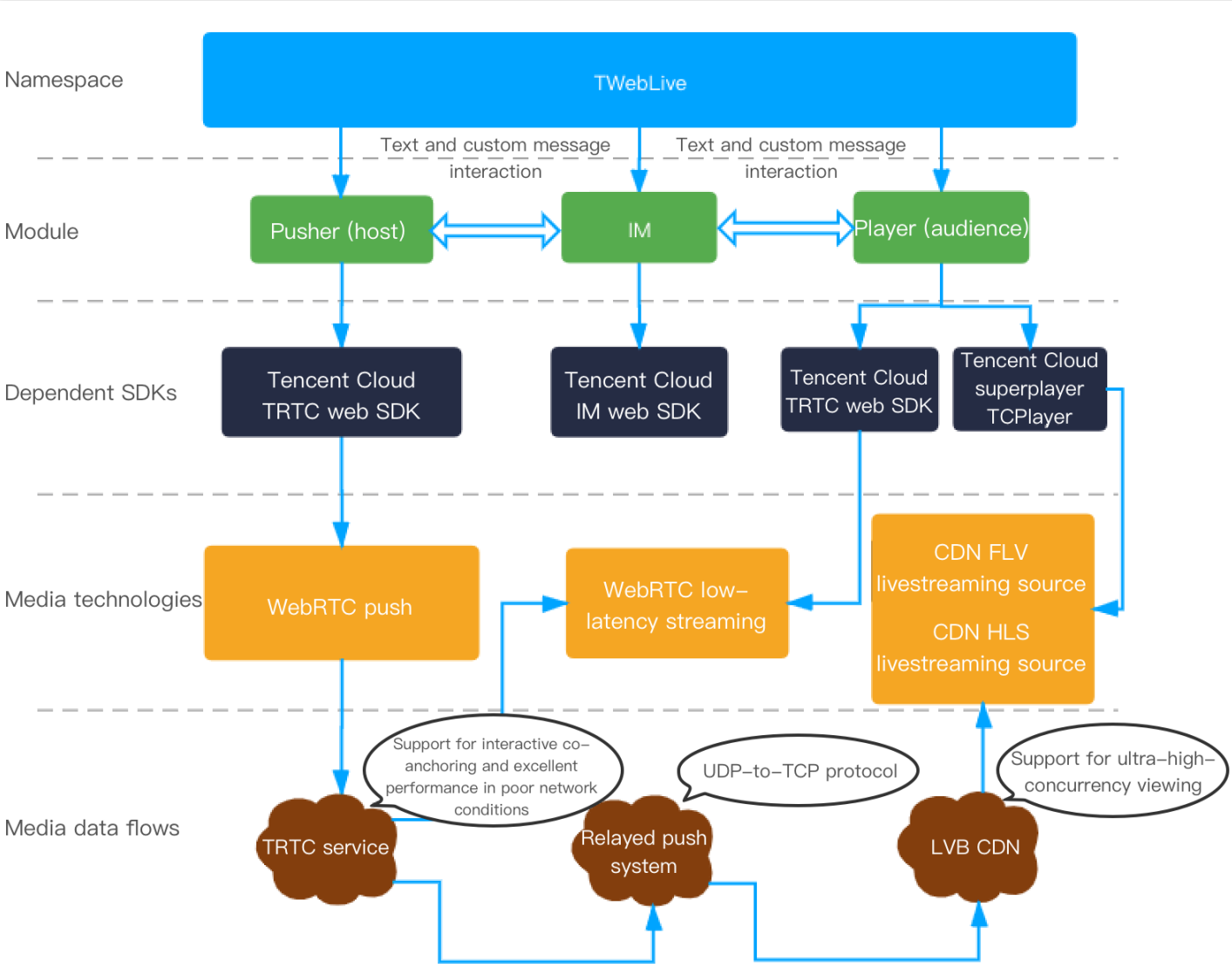
Configure the playback domain name and CNAME in the Tencent Cloud LVB console. For detailed instructions, see CDN Relayed Live Streaming. (If CDN livestreaming is not required, skip this step.)

Download TWebLive through NPM as follows.

```
npm i tweblive --save
```

4. Architecture and Platform Support

The following figure shows the TWebLive architecture design.



Web push and low-latency web streaming adopt the WebRTC technology, which works well with Chrome (desktop version) and Safari (desktop and mobile versions) but poorly or not at all with other platforms (such as browsers on Android terminals).

Operating System	Browser	Minimum Browser Version Requirement	Receiving (Playback)	Sending (Mic-on)	Screen Sharing
MacOS	Safari (desktop)	V11 and later	Supported	Supported	Not supported
MacOS	Chrome (desktop)	V56 and later	Supported	Supported	Supported (for Chrome

					V72 and later)
Windows	Chrome (desktop)	V56 and later	Supported	Supported	Supported (for Chrome V72 and later)
Windows	QQ Browser (desktop)	V10.4	Supported	Supported	Not supported
iOS	Safari (mobile)	V11.1.2	Supported	Supported	Not supported
iOS	WeChat embedded browser	V12.1.4	Supported	Not supported	Not supported
Android	QQ Browser (mobile)	-	Not supported	Not supported	Not supported
Android	UC Browser (mobile)	-	Not supported	Not supported	Not supported
Android	WeChat embedded browser (with the TBS kernel)	-	Supported	Supported	Not supported

We recommend that you use the Mini Program solution on mobile devices, which is supported by both WeChat and Mobile QQ. This solution is built around native technologies of corresponding platforms and delivers excellent audio/video performance. It is specifically adapted to major mobile phone brands. If your application scenario mainly involves the education sector, we recommend that you use the Electron solution for the teacher end. This solution is more stable and supports dual-channel pictures, more flexible screen sharing schemes, and more powerful recovery capabilities in poor network conditions.

5. Notes

- TRTC and IM instances can share each other's accounts and authentication data only if they have the same SDKAppID.
- The IM instance provides the basic-edition content filtering capability for text messages. To use the sensitive word customization feature, click **Upgrade**.

- The local UserSig calculation method is used for local development and debugging only. Do not publish it to your online systems. Once your SECRETKEY is disclosed, attackers can use your Tencent Cloud traffic without authorization. To correctly distribute UserSigs, integrate the computing code of UserSigs into your server and provide an app-oriented API. When a UserSig is required, your app can send a request to the business server to obtain the dynamic UserSig. For more information, see "Generating a UserSig on the Server" in Generating UserSig.
- Due to H.264 copyright restrictions, Chrome and Chrome WebView-based browsers on Huawei devices do not support the TRTC SDK for desktop browsers.

6. Summary

This document describes the new Tencent Cloud web ILVB component TWebLive. With this SDK, developers can quickly implement web push, low-latency web streaming, CDN streaming, real-time chat interactions (or on-screen comments), and other features. In addition, this component can completely replace the traditional flash push solution.

This document also provides a detailed access solution and online demo. Currently, TWebLive works well with mainstream desktop browsers and provides Mini Program solutions on mobile devices.

In the future, we will provide more livestreaming services, such as screen sharing on the push end, image message interaction, multi-line display for viewers (over the WebRTC low-latency and CDN lines), and host-viewer co-anchoring.

References

- TWebLive
- Quick Demo of the Web ILVB Component

Mini Program Livestream Marketing

Last updated : 2020-06-28 15:45:36

Traditional sales methods are costly and the results are difficult to assess. Livestream marketing, however, can integrate search channels, delivery channels, order channels, assessment channels, and operation channels to prevent customer loss due to channel redirection. Therefore, livestream marketing is an integral part of your integrated marketing blueprint in the Internet era.
Data shows that the number of online livestream service users in China reached 504 million in 2019 and is expected to reach 526 million in 2020. Retail sales via livestream platforms will hit RMB 916 billion, accounting for 8.7% of online retail sales in China. With the current boom in livestream e-commerce, it might be a good idea to build your own livestream marketing platform instead of merely selling through mainstream platforms.

Background

The following features must be implemented for interactive scenarios:

- Chat room
- Announcement
- Notifications on user joining and exit of the group
- Notification on new products on the backend
- Notification on gift giving
- Notification on new gifts on the backend
- Like and like notifications
- Livstreaming
- Live room status control

As you can see, the core functionality of livestream marketing consists of Instant Messaging (IM) and livestreaming. Therefore, you can use Instant Messaging (IM) and Live Video Broadcasting (LVB) as the underlying services to meet your needs.

Scenario-based SDK

In livestreaming scenarios, you can enable like, gift, and follow notifications through custom messages, and enable notifications on new products and group status changes through group

custom fields. For this purpose, we encapsulate a scenario-based SDK for easy implementation of the related features.

You can also build your own livestream marketing platform by completing these steps:

Directions

Step 1: Create a live chat room by using AVChatroom in IM

Chat rooms are a crucial component of livestreaming. They allow users to send messages and receive messages from other users in the same room.

1. Log in to the IM console and click **Add Application**.
2. In the **Create Application** dialog box, enter your app name and click **OK**.
   After the app is created, you can view the status, version, SDKAppID, creation time, and expiry time of the new app on the overview page in the console.

   > New apps are created as Trial Edition apps by default and can be upgraded to Pro Edition or Flagship Edition when needed.
   > A Tencent Cloud account supports a maximum of 100 IM apps. If you have reached that limit, disable and delete unwanted apps before creating a new one. **Once an app is deleted, all the data and services associated with the SDKAppID are removed and cannot be restored, so please proceed with caution.**

3. Click the target app card and select **Group Management** in the left sidebar.
4. On the **Group Management** page, click **Add Group**.
5. Configure the following parameters in the "Add Group" dialog box that appears.
   - Group Name: enter the name of the group. This parameter is required, and its maximum length is 30 bytes.
   - Group Owner ID: enter the group owner ID. This parameter is optional, and you must enter a registered user name.
   - Group Type: set the group type. **Audio-video chat room is recommended** according to Group Types.
6. Click **OK** to save the configuration.
   After the group is created, you can see the group ID, group name, group owner, type, and creation time in the group list.

Step 2: Enable like, gift, and purchase notifications through custom messages

In a livestream marketing scenario, you want to send lots of messages in real time to engage group members. For example, when a group member sends a gift to the host, and you want all group members to be notified of this with a special effect. This can be achieved by using custom messages in IM. Custom messages can carry additional information. For example, a gift message can have gift information and user information attached to it. Other user behaviors such as giving likes, purchasing products, and following the host can trigger notifications in the same way. Unlike text messages and rich-text messages, custom messages are a special kind of message that carry special signals.

The following shows the sample code for sending custom messages in the SDK:

```
public async sendCustomMsgAndEmitEvent(type: string, extension?: string) {
const message = this.tim.createCustomMessage({
to: this.roomID,
conversationType: this.TIM.TYPES.CONV_GROUP,
priority: this.TIM.TYPES.MSG_PRIORITY_HIGH,
payload: {
data: type,
description: '',
extension: extension
}
})
await this.tim.sendMessage(message)
const res = {
nick: this.userInfo.nick,
avatar: this.userInfo.avatar,
value: extension,
userID:this.userInfo.userID
}
this.eventBus.emit(type, res)
return res
}
```

Step 3: Enable notifications on new products and lives room status changes through group custom fields

In a livestream marketing scenario, when the host is introducing a product, the product area at the bottom of the screen should be updated with the current product, and all users in the chat room should be notified of the launch of the new product. Generally, new product messages are triggered by the assistant.
Technically, the assistant can trigger new product messages in two ways:

- The assistant sends custom messages:
  - This method requires that the assistant has already joined the current livestreaming group.

  - When there are too many group messages, the IM backend returns messages based on message priority. As custom messages have lower priority, the product displayed in the client app might not be promptly updated.
- The assistant modifies the group profile:
  - In this case, the assistant may or may not be a member of the current livestreaming group. For example, the assistant can be the group admin.
  - While there are defined default group profile fields, IM provides group-level custom fields for which you can specify definitions and read-write permissions.

We recommend that you enable notifications on new products by having the admin modify group custom fields. The steps are as follows:

Adding group custom fields

1. Log in to the IM console and click the target app card. In the left sidebar, choose **Feature Configuration** > **Group Custom Field**.
2. On the **Group Custom Field** page, click **Add Group-level Custom Field**.
3. In the "Group Custom Field" dialog that appears, enter a name for the custom field and set the group type and access permissions.

   - The field name can only contain letters, numbers, and underscores (_), but cannot begin with a number. The maximum length of the name is 16 characters.
   - A group custom field and a group member custom field cannot have the same name.

4. Select **I understand that after adding a group custom field and group type, the read-write permissions of the group type can be modified, but the custom field cannot be deleted.** and click **Confirm** to save the configuration.
   The group-level custom field configuration takes effect in about 10 minutes.

Using group custom fields

The assistant calls the RESTful API for modifying basic group profiles as the group admin to update the corresponding group custom field and send notification messages for new products or LVB status changes.

Step 4: Calculate user levels by using the callback triggered by group message sending

In addition to a profile photo and nickname, every livestreaming audience member has a level, which increases after sending messages or gifts. This feature can be enabled with callbacks. You can

use callbacks to be notified of user changes related to groups, relationship chains, one-to-one chat messages, and online statuses and then change a user's level depending on whether the user sends text messages or gift messages.
For callback-related operations, see Callback Configuration and Third-Party Callback Overview.

Step 5: Block sensitive words with content filtering

Content filtering capabilities are essential in fields like livestreaming, social networking, and consultation. In the live streaming scenario, sending sensitive content could have serious consequences. IM provides a feature to filter out sensitive words and supports filtering some national leader names by default. You can also purchase the value-added content filtering service for this purpose.

Step 6: Generate a push address and start streaming with LVB

Tencent Cloud Live Video Broadcasting (LVB) helps with the growth of e-commerce platforms by enabling merchants to display items in greater detail and assisting consumers in making informed decisions, ultimately reducing marketing costs and boosting sales.
Before starting a live stream, you need to generate a push address through the address generator and then begin LVB push.

Integrating IM Agent in WeChat Subscription Accounts

Last updated : 2020-06-02 11:15:48

This article describes the process of developing a customer service demo using Node.js in order to integrate Tencent Cloud IM with a WeChat subscription account.

> This demo is for reference only. There are many other factors to consider, such as load balancing, API rate limits and persistent data storage, before bringing this to the production environment. These are not covered by this article. You should implement them to suit your needs.

Use Case and Demo Images

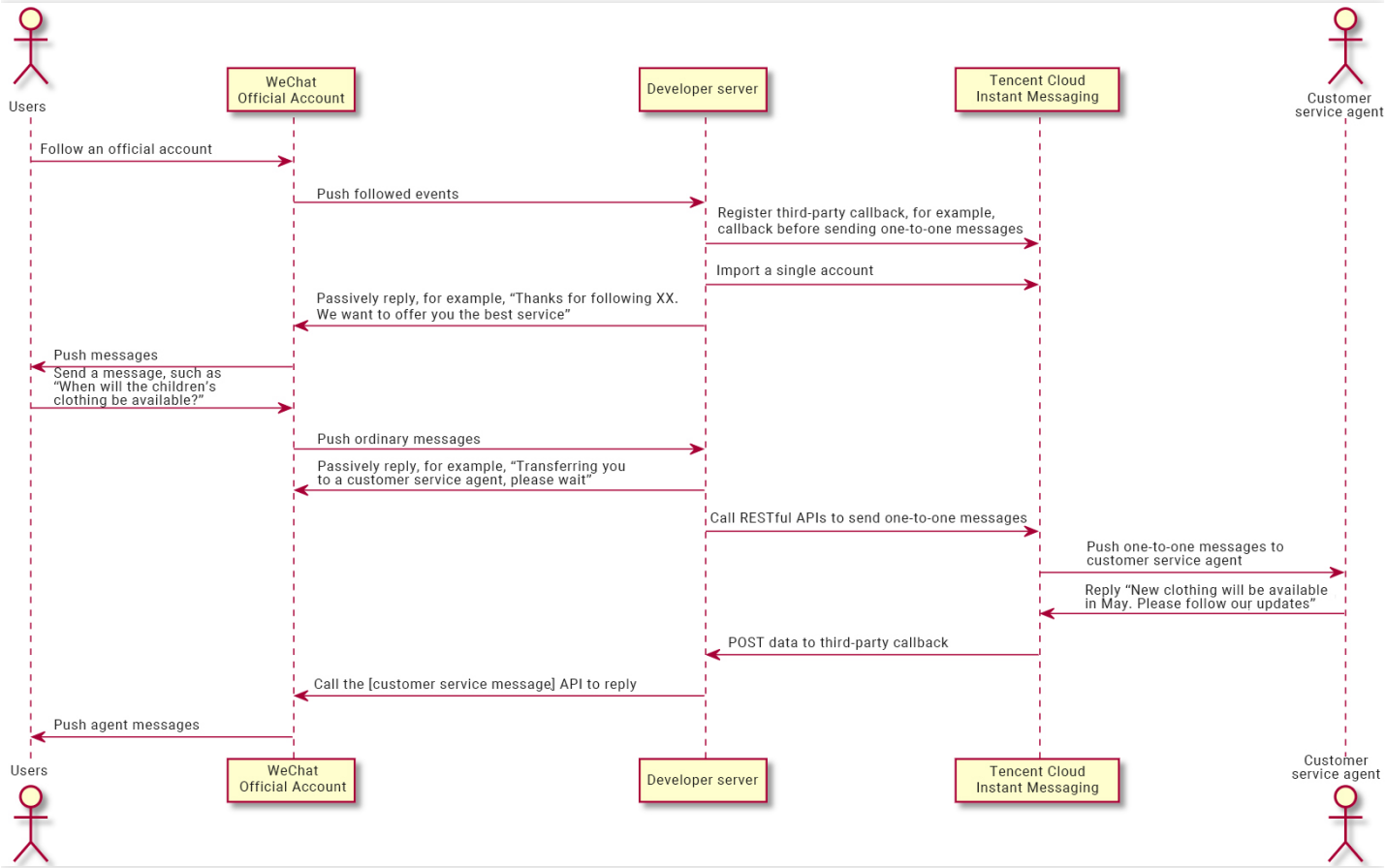
The use case is described as follows:

1. You are a online clothing seller and a customer use the WeChat subscription account to make an inquiry: “when are you getting new children clothing?”.
2. The inquiry is transferred to your customer service agent through Tencent Cloud IM.
3. The agent replies "We are getting new arrivals in May. Please follow our updates." The message is pushed to the customer through Tencent Cloud IM and WeChat.

The following figure illustrates what a customer sees

The following figure illustrates what a customer agent sees

The following figure is a flowchart of the use case;



Considerations

- If the message transfer link is long, it may take longer to send and receive messages.
- Subscription accounts registered by individuals cannot use the [customer service message] API of the WeChat Official Accounts Platform to push messages to subscribers.

Prerequisites

- You have prepared a public network development server or a CVM instance that is capable of running Node.js.
- You have registered a WeChat subscription account or service account.
- You have read the developer documentation for the WeChat Official Accounts Platform.
- You have created an IM app.
- You have imported IM user accounts, such as user0 and user1, through one-by-one import or batch import.

References

- API documentation
- Third-party callback
- Developer Guide for the WeChat Official Accounts Platform
- Express framework tutorial

Directions

Step 1: create a project and install dependencies

```
npm init -y

// express framework
npm i express@latest --save

// Encryption module
npm i crypto@latest --save

// xml parsing tool
npm i xml2js@latest --save

// Initiate an HTTP request.
npm i axios@latest --save

// Calculate userSig.
npm i tls-sig-api-v2@latest --save
```

Step 2: enter IM app information and calculate the UserSig

```
// ------------ IM ------------
const IMAxios = axios.create({
  timeout: 10000,
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8',
  },
});

// The user ID mapping table imported into the IM account system is not persistently stored and used
const importedAccountMap = new Map();
// IM app and app administrator information can be obtained from the IM console.
const SDKAppID = 0; // Enter the SDKAppID of the IM app.
const secrectKey = ''; // Enter the key of the IM app.
```

```
const AppAdmin = 'user0'; // Set user0 as the app admin account.
const kfAccount1 = 'user1'; // Set user1 as a customer service agent account.
// Calculate UserSig, which will be used when calling RESTful APIs. For more information, refer to c
const api = new TLSSigAPIv2.Api(SDKAppID, secrectKey);
const userSig = api.genSig(AppAdmin, 86400*180);
console.log('userSig:', userSig);
```

Step 3: configure the URL and token

> This guide document was written with reference to the Developer Guide for the WeChat Official Accounts Platform. In case of any change, the latest information can be found in the Access Guide.

1. Log in to the management backend of the subscription account.
2. Select **Basic Configuration** and select the checkbox to agree to become a developer.
3. Click **Modify Configuration** and enter the following:
   - URL: the API URL for receiving WeChat messages and events. Required.
   - Token: user-defined string for generating a signature. The token will be compared with the token contained in the API URL for authentication. Required.
   - EncodingAESKey: manually entered or randomly generated. This is used as the key for message encryption and decryption. Optional.

Step 4: enable the web service listening port and correctly respond to the token authentication request by WeChat

```
const express = require('express'); // express framework
const crypto =  require('crypto'); // Encryption module
const util = require('util');
const xml2js = require('xml2js'); // Parse xml.
const axios = require('axios'); // Initiate an HTTP request.
const TLSSigAPIv2 = require('tls-sig-api-v2'); // Calculate userSig.

// ------------ Web service ------------
var app = express();
// The token is set in **Subscription Account Management Backend** -> **Basic Configuration**.

// Process all GET requests from port 80.
app.get('/', function(req, res) {
  // ------------ Accessing the WeChat Official Accounts Platform ------------
  // For more information, see the WeChat official documentation.
  // Obtain signature, timestamp, nonce, and echostr from the WeChat server GET request.
```

```
  var signature = req.query.signature; // WeChat encrypted signature
  var timestamp = req.query.timestamp; // Timestamp
  var nonce = req.query.nonce; // Random number
  var echostr = req.query.echostr; // Random character string

  // Sort the token, timestamp, and nonce parameters in a lexicographical order.
  var array = [myToken, timestamp, nonce];
  array.sort();

  // Concatenate the character strings of the three parameters into one for sha1 encryption.
  var tempStr = array.join('');
  const hashCode = crypto.createHash('sha1'); // Create an encryption type.
  var resultCode = hashCode.update(tempStr,'utf8').digest('hex'); // Encrypt the character string.

  // After obtaining the encrypted character string, the developer can compare it with signature and
  if (resultCode === signature) {
    res.send(echostr);
  } else {
    res.send('404 not found');
  }
});

// Listen to port 80.
app.listen(80);
```

Step 5: implement the service logic on the developer server

- When receiving subscription events from WeChat, call the Importing a Single Account or Batch Importing Accounts API to import accounts into the system.
- When receiving subscription events from WeChat, passively reply to the message.
- When receiving unsubscription events from WeChat, call the Deleting Accounts API to delete the account from the system.
- When receiving a common message pushed by WeChat, call the Sending a One-to-One Message API to send a one-to-one message to the agent account.

```
const genRandom = function() {
return Math.floor(Math.random() * 10000000);
}

// Generate the wx text reply xml.
const genWxTextReplyXML = function(to, from, content) {
let xmlContent = '<xml><ToUserName><![CDATA[' + to + ']]></ToUserName>'
xmlContent += '<FromUserName><![CDATA[' + from + ']]></FromUserName>'
xmlContent += '<CreateTime>' + new Date().getTime() + '</CreateTime>'
xmlContent += '<MsgType><![CDATA[text]]></MsgType>'
```

```
xmlContent += '<Content><![CDATA[' + content + ']]></Content></xml>';

return xmlContent;
}

/**
* Import users into the IM account system.
* @param {String} userID ID of the user to be imported
*/
const importAccount = function(userID) {
console.log('importAccount:', userID);
return new Promise(function(resolve, reject) {
var url = util.format('https://console.tim.qq.com/v4/im_open_login_svc/account_import?sdkappid=%s
&identifier=%s&usersig=%s&random=%s&contenttype=json',
SDKAppID, AppAdmin, userSig, genRandom());
console.log('importAccount url:', url);
IMAxios({
url: url,
data: {
"Identifier": userID
},
method: 'POST'
}).then((res) => {
if (res.data.ErrorCode === 0) {
console.log('importAccount ok.', res.data);
resolve();
} else {
reject(res.data);
}
}).catch((error) => {
console.log('importAccount failed.', error);
reject(error);
})
});
}

/**
* Delete users from the IM account system.
* @param {String} userID ID of the user to be deleted
*/
const deleteAccount = function(userID) {
console.log('deleteAccount', userID);
return new Promise(function(resolve, reject) {
var url = util.format('https://console.tim.qq.com/v4/im_open_login_svc/account_delete?sdkappid=%s
&identifier=%s&usersig=%s&random=%s&contenttype=json',
SDKAppID, AppAdmin, userSig, genRandom());
console.log('deleteAccount url:', url);
IMAxios({
```

```
url: url,
data: {
"DeleteItem": [
{
"UserID": userID,
},
]
},
method: 'POST'
}).then((res) => {
if (res.data.ErrorCode === 0) {
console.log('deleteAccount ok.', res.data);
resolve();
} else {
reject(res.data);
}
}).catch((error) => {
console.log('deleteAccount failed.', error);
reject(error);
})
});
}

/**
* Send a one-to-one message.
*/
const sendC2CTextMessage = function(userID, content) {
console.log('sendC2CTextMessage:', userID, content);
return new Promise(function(resolve, reject) {
var url = util.format('https://console.tim.qq.com/v4/openim/sendmsg?sdkappid=%s&identifier=%s&use
rsig=%s&random=%s&contenttype=json',
SDKAppID, AppAdmin, userSig, genRandom());
console.log('sendC2CTextMessage url:', url);
IMAxios({
url: url,
data: {
"SyncOtherMachine": 2, // The message will not be synchronized to the sender. If you want to sync
hronize the message to the From_Account, set SyncOtherMachine to 1.
"To_Account": userID,
"MsgLifeTime":60, // Messages are retained for 60 seconds.
"MsgRandom": 1287657,
"MsgTimeStamp": Math.floor(Date.now() / 1000), // The unit is second and must be an integer.
"MsgBody": [
{
"MsgType": "TIMTextElem",
"MsgContent": {
"Text": content
}
```

```
}
]
},
method: 'POST'
}).then((res) => {
if (res.data.ErrorCode === 0) {
console.log('sendC2CTextMessage ok.', res.data);
resolve();
} else {
reject(res.data);
}
}).catch((error) => {
console.log('sendC2CTextMessage failed.', error);
reject(error);
});
});
}

// Process WeChat post requests.
app.post('/', function(req, res) {
var buffer = [];
// Listen for data events in order to receive data.
req.on('data', function(data) {
buffer.push(data);
});
// Listen for end events in order to process received data.
req.on('end', function() {
const tmpStr = Buffer.concat(buffer).toString('utf-8');
xml2js.parseString(tmpStr, { explicitArray: false }, function(err, result) {
if (err) {
console.log(err);
res.send("success");
} else {
if (!result) {
res.send("success");
return;
}
console.log('wx post data:', result.xml);
var wxXMLData = result.xml;
var toUser = wxXMLData.ToUserName; // Recipient's WeChat
var fromUser = wxXMLData.FromUserName;// Sender's WeChat
if (wxXMLData.Event) { // Process the event type.
switch (wxXMLData.Event) {
case "subscribe": // Subscribe.
res.send(genWxTextReplyXML(fromUser, toUser, 'Thanks for following XX. We want to offer you the b
est service.'));
importAccount(fromUser).then(() => {
// Record the ID of the imported user.
```

```
importedAccountMap.set(fromUser, 1);
});
break;
case "unsubscribe": // Unsubscribe.
deleteAccount(fromUser).then(() => {
importedAccountMap.delete(fromUser);
});
res.send("success");
break;
}
} else { // Process the message type.
switch (wxXMLData.MsgType) {
case "text":
// Process text messages.
sendC2CTextMessage(kfAccount1, 'Inquiry from a WeChat subscription account: ' + wxXMLData.Conten
t).then(() => {
console.log('C2C message sent successfully');
}).catch((error) => {
console.log('C2C message failed to sent');
});
break;
case "image":
// Process image messages.
break;
case "voice":
// Process voice messages.
break;
case "video":
// Process video messages.
break;
case "shortvideo":
// Process short video messages.
break;
case "location":
// Process locations.
break;
case "link":
// Process link messages.
break;
default:
break;
}
res.send(genWxTextReplyXML(fromUser, toUser, 'Transferring you to a customer service agent, pleas
e wait.'));
}
}
})
```

```
});
});
```

Step 6: register and process IM third-party callbacks

```
// Process POST requests from IM third-party callbacks.
app.post('/imcallback', function(req, res) {
  var buffer = [];
  // Listen for data events in order to to receive data.
  req.on('data', function(data) {
    buffer.push(data);
  });
  // Listen to end events in order to process received data.
  req.on('end', function() {
    const tmpStr = Buffer.concat(buffer).toString('utf-8');
    console.log('imcallback', tmpStr);
    const imData = JSON.parse(tmpStr);
    // Push the message sent by kfAccount1 to the customer.
    if (imData.From_Account === kfAccount1) {
      // Package the message and push it using the **customer service message** API of WeChat to the
      // Note: subscription accounts registered by individuals cannot use this API. For more informa
    }

    res.send({
      "ActionStatus":"OK",
      "ErrorInfo":"",
      "ErrorCode": 0 // 0: not muted. 1: muted.
    });
  });
});
```

WeChat HTML5 Livestreaming Interaction

Last updated : 2020-07-16 13:46:05

Based on the web (HTML5) player and Instant Messaging (IM), the Web ILVB component encapsulates easy-to-use APIs and provides a free and open-source demo, allowing developers to quickly integrate and use this component.
This component is suitable for web livestreaming scenarios, such as large conferences, events, classes, and webinars as well as WeChat HTML5 livestream marketing.

Trying It Online

Click here to try it out

References

- TWebLive API
- Tencent Real-Time Communication
- TRTC Electron API
- Web (HTML5) Player
- Instant Messaging (IM)
- WebIM API

Benefits of the Web ILVB Component

Reduce development time and allow developers to focus on business logic

When using the Web ILVB component, developers only need to specify several parameters after downloading the SDK to implement projects that involve common features such as LVB videos, chats, likes, and gifts.

Reduce the time required to identify and resolve issues

- In the live streaming scenario, when there are many participants and messages, in-house services can encounter performance bottlenecks (for example, they might not be able to handle highly concurrent workloads), resulting in difficult problems, including a low request success rate,

message backlog, message loss, and high message latency. The Web ILVB component ensures high-concurrency and highly available IM capabilities by integrating Instant Messaging (IM), which is built on QQ's wealth of experience in the IM field. It also provides statistics, logging, and troubleshooting features to help you quickly identify and resolve issues.

- The Web ILVB component allows you to set message priorities. For example, the priority of host messages and gift messages can be set to high, and the priority of like messages can be set to low. When the frequency exceeds 40 messages per second in a live room, the IM backend limits the message frequency, but ensures that high-priority messages are delivered.

Reduce project development and OPS costs

- The Web ILVB component is free and open-source, and the accompanied Web (HTML5) player is also free. IM is the only value-added service required for this solution. You can use the free Trial Edition or upgrade IM to Enterprise Edition or Flagship Edition. For billing details, see Billing Overview.
- AVChatRoom groups in IM support unlimited group members.
- IM server nodes provide extensive coverage, allowing users to access the service from nearby without worrying about server scaling.
- Content filtering can identify harmful content such as pornography, politics, and profanity in order to meet regulatory requirements.
- Our IM service team quickly responds to protect your projects.

Instructions

Step 1: Create an IM app and a group

1. Create an app in the IM console and obtain the SDKAppID.
2. Generate a UserSig.
3. Import a single account (or import multiple accounts) to IM.
4. Create an live streaming group (AVChatRoom group) through the RESTful API or console.

Step 2: Run the Web ILVB component

```
// npm i tweblive
import TWebLive from 'tweblive';

let options = {
SDKAppID: 0, // Replace 0 with the SDKAppID of your IM app during integration.
domID: "id_test_video", // ID of the web player container, such as <div id="id_test_video" style
="width:100%; height:auto;"></div>
```

```
m3u8: "http://200002949.vod.myqcloud.com/200002949_b6ffc.f0.m3u8", // Replace it with the actual playback URL.
flv: "http://200002949.vod.myqcloud.com/200002949_b6ffc.f0.flv" // Replace it with the actual playback URL.
};
// Create an instance.
let tweblive = new TWebLive(options);

// This is triggered when the SDK is in the ready state. The accessing side listens to this event so that it can call SDK APIs to send messages or perform other actions and use different features of the SDK.
let onIMReady = function() {
tweblive.sendTextMessage({
roomID: 'AV1', // Replace it with the ID of the joined live room.
text: 'hello from TWebLive'
}).then(function(res) {
console.log('demo sendTextMessage OK', res);
}).catch(function(err) {
console.log('demo sendTextMessage failed', err);
});
}
tweblive.on(TWebLive.EVENT.IM_READY, onIMReady);

// A text message from another user in the live room was received.
let onTextMessageReceived = function(event) {
event.data.forEach(function(message) {
// Use the nickname if any. Otherwise, use the UserID.
console.log('demo ' + (message.nick || message.from) + ' says: ', message.payload.text);
});
}
tweblive.on(TWebLive.EVENT.TEXT_MESSAGE_RECEIVED, onTextMessageReceived);

// A custom message (such as a gift or like message) from another user in the live room was received.
let onCustomMessageReceived = function(event) {
event.data.forEach(function(message) {
console.log('demo ' + 'data:' + message.payload.data + ' description:' + message.payload.description + ' extension:' + message.payload.extension);
});
}
tweblive.on(TWebLive.EVENT.CUSTOM_MESSAGE_RECEIVED, onCustomMessageReceived);

// A notification of a user joining the live room was received.
let onRemoteUserJoin = function(event) {
event.data.forEach(function(message) {
// Use the nickname if any. Otherwise, use the userID.
console.log('demo ' + (message.nick || message.payload.userIDList[0]) + ' joined the group');
});
```

```
}
tweblive.on(TWebLive.EVENT.REMOTE_USER_JOIN, onRemoteUserJoin);

// For more events, see EVENT.

// Anonymously join a live room when not logged in. In this case, you can only receive messages and cannot send messages.
tweblive.enterRoom(""); // Enter the ID of the live room to join. It corresponds to the groupID of the AVChatRoom group in IM.
```

Step 3: Provide the live streaming source

We recommend that you use the relayed push service of TRTC.
For compatibility with both computers and mobile devices, we recommend that you provide your live streaming source in both flv and hls formats.

- On browsers that support MSEs, the component will prefer flv sources as they enable lower latency and better performance.
- On browsers that do not support MSEs, the component will prefer hls sources as they work better with mobile devices, even though they increase latency somewhat.

For LVB push on Windows or Mac platforms, we strongly recommend that you use TRTC Electron.
Relayed push generates both flv and hls streams and integrates with IM to provide stable, reliable, and comprehensive services.
For instructions on how to quickly enable LVB and relayed push, see Electron.

Step 4: Perform secondary development based on the Web ILVB component

You can further develop other business logic and features based on the Web ILVB component.

FAQs

1. How can I display the nickname (nick) instead of `userID` in the notification and chat messages after joining a live room?

To display the nickname in a live room, set the nickname (skip this step if it is already set) and then join the live room.

```
// Only logged-in users can change their nicknames.
tweblive.setMyProfile({ nick: "Indiana Jones" }).then(() => {
tweblive.enterRoom(""); // Enter the ID of the live room to join. It corresponds to the groupID of the AVChatRoom group in IM.
});
```