即时通信 IM

场景实践

产品文档

腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

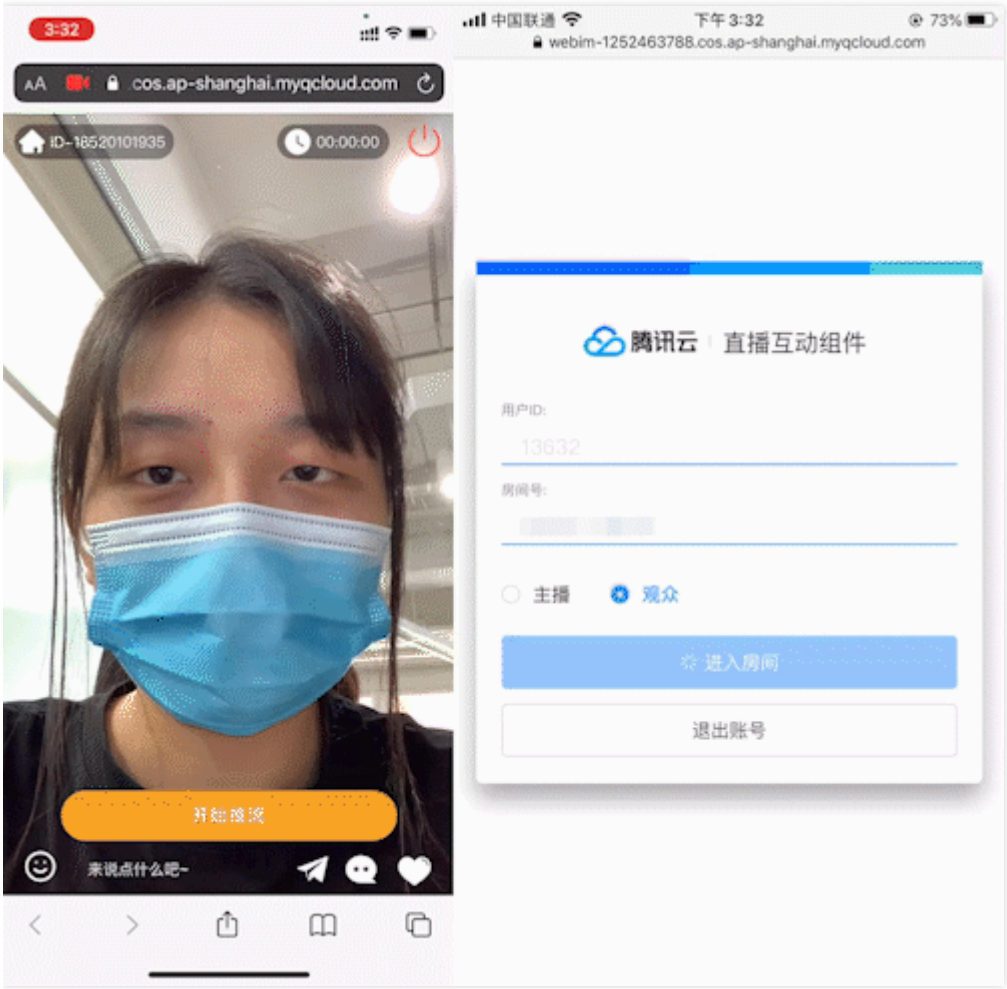
文档目录

场景实践
Web 直播互动组件

最近更新时间：2020-10-23 17:09:29

一、TWebLive 简介

TWebLive 即腾讯云 Web 直播互动组件，是腾讯云终端研发团队推出的一个新的 SDK，集成了 腾讯云实时音视频 TRTC、腾讯云即时通信 IM、腾讯云超级播放器 TCPlayer，覆盖了 Web 直播互动场景常见的功能（推流、开/关麦、开/关摄像头、微信分享观看、聊天点赞等），并封装了简单易用的 API，接入后可快速实现 Web 端推流、拉流以及实时聊天互动功能。



二、TWebLive 的优势

开发者接入此 SDK，可彻底替代 flash 推流方案，极大地降低 Web 推流、Web 低延时观看、CDN 观看以及实时聊天互动（或弹幕）的实现复杂度和时间成本，下面我们通过举例来进行说明。

1. 推流

当需要推流时，创建 Pusher（推流）对象，最简单的推流仅需3步：

```
<div id="pusherView" style="width:100%; height:auto;"></div>
<script>
// 1、创建 Pusher（推流）对象
let pusher = TWebLive.createPusher({ userID: 'your userID' });

// 2、设置渲染界面，且从麦克风采集音频，从摄像头采集视频（默认720p）
pusher.setRenderView({
elementID: 'pusherView',
audio: true,
video: true
}).then(() => {
// 3、填入 sdkappid roomid 等信息，推流
// url 必须以 `room://` 开头
let url = `room://sdkappid=${SDKAppID}&roomid=${roomID}&userid=${userID}&usersig=${userSig}
&livedomainname=${liveDomainName}&streamid=${streamID}`;
pusher.startPush(url).then(() => {
console.log('pusher | startPush | ok');
});
}).catch(error => {
console.error('pusher | setRenderView | failed', error);
});
</script>
```

2. 拉流

当需要拉流播放时，创建 Player（播放器）对象，最简单的拉流仅需3步：

```
<div id="playerView" style="width:100%; height:auto;"></div>
<script>
// 1、创建 Player（播放器）对象
let player = TWebLive.createPlayer();

// 2、设置渲染界面
player.setRenderView({ elementID: 'playerView' });

// 3、填入 flv hls 地址等信息，拉 CDN 流播放，此时 url 必须以 `https://` 开头
// 或 填入 sdkappid roomid 等信息，拉 WebRTC 低延时流播放，此时 url 必须以 `room://` 开头
let url = 'https://'
```

```
+ 'flv=https://200002949.vod.myqcloud.com/200002949_b6ffc.f0.flv' + '&' // 请替换成实际可用的播放地址
+ 'hls=https://200002949.vod.myqcloud.com/200002949_b6ffc.f0.m3u8' // 请替换成实际可用的播放地址

// 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. 直播互动

当主播和观众需要聊天互动时，创建 IM（即时通信）对象，最简单的消息收发仅需3步：

```
// 1、创建 IM（即时通信）对象并监听事件
let im = TWebLive.createIM({
SDKAppID: 0 // 接入时需要将0替换为您的即时通信应用的 SDKAppID
});
// 监听 IM_READY IM_TEXT_MESSAGE_RECEIVED 等事件
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);
});
};
// 接入侧监听此事件，然后可调用 SDK 发送消息等
im.on(TWebLive.EVENT.IM_READY, onIMReady);
// 收到文本消息，上屏
im.on(TWebLive.EVENT.IM_TEXT_MESSAGE_RECEIVED, onTextMessageReceived);

// 2、登录
im.login({userID: 'your userID', userSig: 'your userSig'}).then((imResponse) => {
console.log(imResponse.data); // 登录成功
if (imResponse.data.repeatLogin === true) {
// 标识账号已登录，本次登录操作为重复登录
console.log(imResponse.data.errorInfo);
}
}).catch((imError) => {
console.warn('im | login | failed', imError); // 登录失败的相关信息
});
```

```
// 3、加入直播间
im.enterRoom('your roomID').then((imResponse) => {
switch (imResponse.data.status) {
case TWebLive.TYPES.ENTER_ROOM_SUCCESS: // 加入直播间成功
break;
case TWebLive.TYPES.ALREADY_IN_ROOM: // 已经在直播间内
break;
default:
break;
}
}).catch((imError) => {
console.warn('im | enterRoom | failed', imError); // 加入直播间失败的相关信息
});
</script>
```

为了进一步降低开发者的开发和人力成本，我们在 TWebLive SDK 的基础上，提供了同时适配 PC 和移动端浏览器的 Demo，并开源到了 Github。开发者 fork&clone 项目到本地，稍作修改即可把 Demo 跑起来，或者集成到自己的项目部署上线。

三、接入使用

在 腾讯云实时音视频 TRTC 控制台 中创建一个实时音视频应用（与此同时会自动创建一个 SDKAppID 相同的 IM 应用），保存 SDKAPPID 。然后在【应用管理】>【功能配置】中开启自动旁路推流。开启旁路推流功能后，TRTC 房间里的每一路画面都配备一路对应的播放地址（如果不需要 CDN 直播观看，可略过开启旁路推流的步骤）。

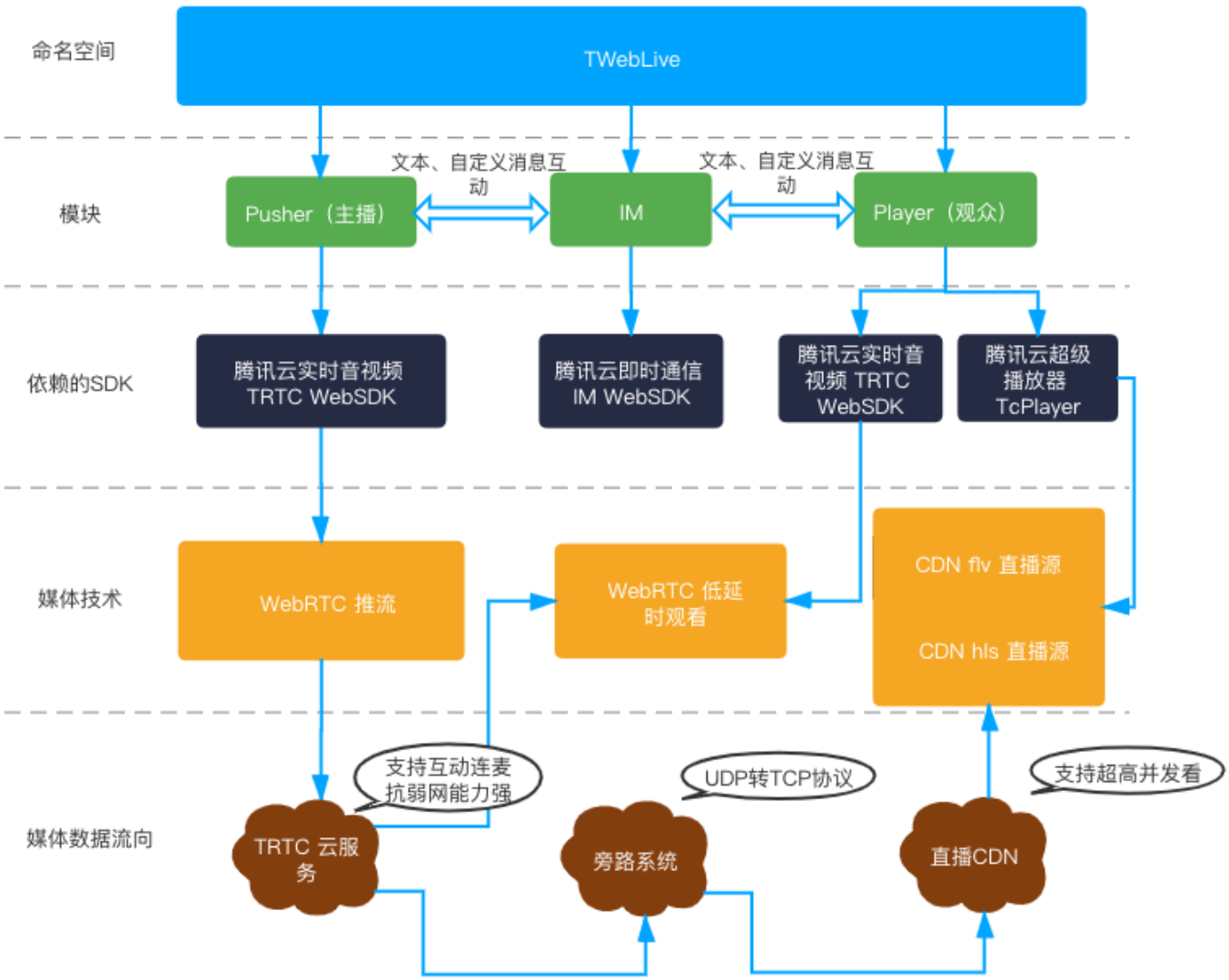
在 腾讯云直播控制台 配置播放域名并完成 CNAME 配置，详细操作指引请参见 实现 CDN 直播观看 文档（如果不需要 CDN 直播观看，此步骤可略过）。

通过 npm 下载 TWebLive：

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

四、架构与平台支持

TWebLive 架构设计如下图所示：



Web 推流和 Web 低延时观看用到了 WebRTC 技术。目前主要在桌面版 Chrome 浏览器、桌面版 Safari 浏览器以及移动版 Safari 浏览器上有较为完整的支持，其他平台（例如 Android 平台的浏览器）支持情况均比较差，具体如下：

操作系统	浏览器类型	浏览器最低版本要求	接收（播放）	发送（上麦）	屏幕分享
Mac OS	桌面版 Safari 浏览器	11+	支持	支持	不支持

操作系统	浏览器类型	浏览器最低版本要求	接收（播放）	发送（上麦）	屏幕分享
Mac OS	桌面版 Chrome 浏览器	56+	支持	支持	支持（需要 chrome72+ 版本）
Windows	桌面版 Chrome 浏览器	56+	支持	支持	支持（需要 chrome72+ 版本）
Windows	桌面版 QQ 浏览器	10.4	支持	支持	不支持
iOS	移动版 Safari 浏览器	11.1.2	支持	支持	不支持
iOS	微信内嵌网页	12.1.4	支持	不支持	不支持
Android	移动版 QQ 浏览器	-	不支持	不支持	不支持
Android	移动版 UC 浏览器	-	不支持	不支持	不支持
Android	微信内嵌网页（TBS 内核）	-	支持	支持	不支持

在移动端推荐使用 小程序 解决方案，微信和手机 QQ 小程序均已支持，都是由各平台的 Native 技术实现，音视频性能更好，且针对主流手机品牌进行了定向适配。如果您的应用场景主要为教育场景，那么教师端推荐使用稳定性更好的 Electron 解决方案，支持大小双路画面，更灵活的屏幕分享方案以及更强大的弱网络恢复能力。

五、注意事项

- 实时音视频应用与 IM 应用的 SDKAppID 一致，才能复用账号与鉴权。
- IM 应用针对文本消息，提供基础版本的安全打击能力，如果希望使用自定义不雅词功能，可以单击【升级】或在购买页 购买安全打击 - 专业版服务。
- 本地计算 UserSig 的方式仅用于本地开发调试，请勿直接发布到线上，一旦 SECRETKEY 泄露，攻击者就可以盗用您的腾讯云流量。正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。
- 由于 H.264 版权限制，华为系统的 Chrome 浏览器和以 Chrome WebView 为内核的浏览器均不支持 TRTC 桌面浏览器端 SDK 的正常运行。

六、结语

本文为您介绍了腾讯云新的 Web 直播互动组件：TWebLive，通过接入此 SDK，开发者可以快速轻便地实现 Web 推流、Web 低延时观看、CDN 观看以及实时聊天互动（或弹幕）等功能，能够很好替换传统的 flash 推流方案。

同时，提供详细的接入方案和 在线 Demo 供您体验。目前 TWebLive 在主流的桌面浏览器上也有较好的支持，在移动端支持小程序的解决方案。

后续，我们会提供更全方位的直播功能服务，例如：推流端支持屏幕分享、图片消息互动、观众端多线路观看（WebRTC 低延时线路和 CDN 线路）、主播观众连麦互动等功能。

参考资料：

- TWebLive 接口手册
- 一分钟跑通 Web 直播互动组件

相关文档

折扣活动

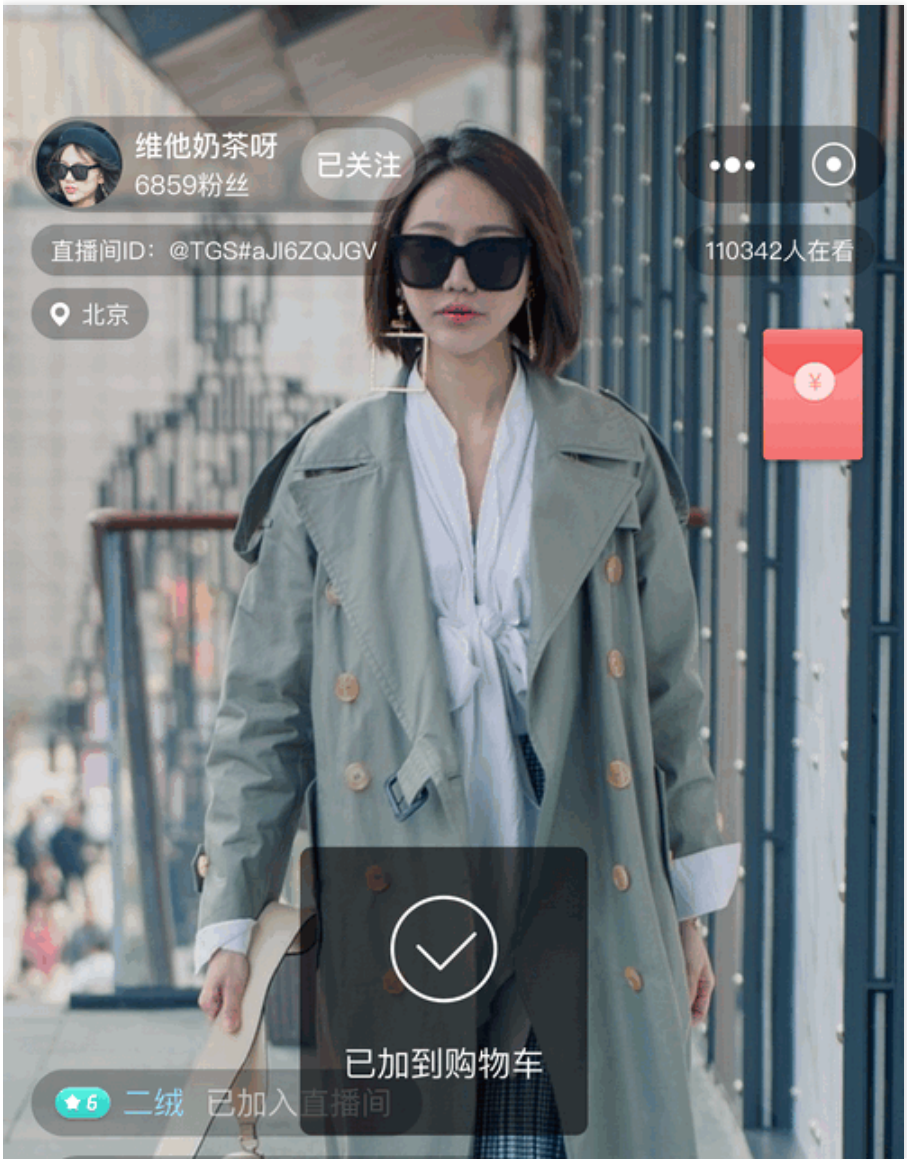
小程序直播带货

最近更新时间：2020-10-23 17:09:59

传统销售方式成本高，效果难以评估。直播带货无疑为各大商家突破了营销活动中的层层阻碍，将搜索渠道、触达渠道、下订渠道、评估渠道、运营渠道整合到了一起，整合后能有效避免因渠道跳转带来的用户流失。因此，直播带货成为了互联网时代整合营销的必备之选。
有数据显示，2019年全国在线直播用户5.04亿，2020年预计5.26亿，销售规模9160亿，占我国网络零售规模的8.7%。市场如此之大，直播带货的效果如此好，除了进驻各大直播平台进行直播以外，如何搭建自己的直播带货平台呢？

需求背景

根据场景分析，可将交互效果设计如下图所示：



交互场景中需要实现以下能力：

- 聊天室功能
- 公告功能
- 用户加入群聊、退出群里提示
- 后台上架新商品提示
- 送礼提示
- 后台上架新礼物提示
- 点赞&点赞提示
- 直播功能
- 直播间状态控制功能

由此可以看出，直播带货的核心功能分为即时通信 IM 能力和直播能力，您可以选用 即时通信 IM 与 云直播 作为需求实现的基础。

Demo 体验

场景化 SDK

在直播带货场景中，您可以通过自定义消息实现点赞、送礼、关注的提醒，通过群自定义字段实现商品上架通知、群状态改变通知。基于此，我们封装了一个 场景化 SDK，轻松实现直播带货的场景的相关功能，详细使用指南请参阅 小程序直播 SDK API。

您也可以根据以下步骤搭建自己的直播带货平台：

操作步骤

步骤1：使用即时通信 IM 的 AVChatroom 实现直播聊天室

聊天室是直播中很重要的一部分，用户可以在这里发送自己想说的话并且可以收到同一个聊天室中其他成员的消息。

1. 登录 即时通信 IM 控制台，单击【+添加新应用】。
2. 在【创建应用】对话框中输入您的应用名称，单击【确定】。
   创建完成后，可在控制台总览页查看新建应用的状态、业务版本、SDKAppID、创建时间以及到期时间。

> 说明：
> 新建应用的业务版本默认为体验版，后续可根据需求 升级 至专业版或旗舰版。
> 同一个腾讯云账号，最多可创建100个即时通信 IM 应用。若已有100个应用，您可以先 停用并删除 无需使用的应用后再创建新的应用。**应用删除后，该 SDKAppID 对应的所有数据和服务不可恢复，请谨慎操作。**

3. 单击目标应用卡片，在左侧导航栏选择【群组管理】。
4. 在【群组管理】页面，单击【添加群组】。
5. 在弹出的添加群组对话框中，配置以下参数：
   - 群名称：请输入群组的名称，必填参数，长度不超过30字节。
   - 群主 ID：请输入群主的 ID，选填参数，必须输入已注册的用户名。
   - 群类型：请设置群组类型，根据 群组类型介绍，**建议设置为直播群 (AVChatRoom)。**
6. 单击【确定】保存配置。
   群组创建完成后，在群组列表中可以查看群 ID、群名称、群主、类型和创建时间。

步骤2：使用自定义消息实现点赞、送礼、购买等行为的消息通知

在直播的场景中，为达到活跃群热度的效果需要将大量消息实时反馈给群成员，例如有人送礼给主播，需要全员通知，可能还需要一个特别炫酷的动效来展现。全员通知的送礼消息可以用 IM 的自定义消息来实现，并且自定义消息可以携带额外信息，用户在发送礼消息时，可以携带上礼物信息以及用户信息。同理，用户的点赞行为、购买商品行为、关注主播的行为等都可以用自定义消息来实现。自定义消息与文本消息、富文本消息有所不同，可以理解为用户发送的一种特殊消息，它仅仅是传递了一种特殊信号。

SDK 中发送自定义消息的示例代码如下：

```
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
}
```

步骤3：使用群自定义字段实现直播间上新商品、直播状态改变的消息通知

在直播带货场景中，主播推荐某款产品时，屏幕下方的商品位需要立即更新为当前产品，并需要通知所有在直播中的用户有新的商品上线，商品上新消息一般由小助手触发。
从技术角度来看，小助手触发商品上新消息有两种方案：

- 小助手发送自定义消息
  - 该方案要求小助手已加入到当前直播群中。
  - 当群消息量太大时，IM 后台会根据消息优先级返回消息，自定义消息的优先级低，可能会出现用户端商品刷新不及时的情况。
- 小助手修改群资料
  - 采用该方案，小助手可以是当前直播群的群成员（例如管理员角色），也可以不是群成员。
  - 虽然默认的群资料字段已有既定含义，但 IM 还提供了群维度的自定义字段，您可以自定义指定字段含义和读/写权限等。

因此，推荐采用**管理员修改群自定义字段的方式**实现商品上新消息通知，具体操作步骤如下：

添加群自定义字段

1. 登录 即时通信 IM 控制台 ，单击目标应用卡片，在左侧导航栏选择【功能配置】>【群自定义字段】。
2. 在【群自定义字段】页面，单击【添加群维度自定义字段】。
3. 在弹出的群维度自定义字段对话框中，输入字段名称，设置群组形态及其对应的读写权限。

> 说明：
>
> - 字段名称只能由字母、数字以及下划线（_）组成，不能以数字开头，且长度不能超过16个字符。
> - 群自定义字段名称不允许与群成员自定义字段名称一致。

4. 勾选【我已经知道自定义字段和群组形态添加后，除了形态的读写权限可修改，无法删除。】，单击【确定】保存设置。
   群维度自定义字段配置后大约10分钟左右生效。

使用群自定义字段

小助手以群管理员的角色身份，适时调用 修改群组基础资料 REST API 更新对应的群自定义字段，从而实现直播间的商品上新通知和直播状态改变消息通知。

步骤4：使用群发消息后回调实现用户等级统计

在直播场景中，用户除了头像、昵称等一般还有等级信息，例如观众发消息、送礼之后用户的成长等级需要增加。您可以通过回调实现相关功能，业务方可通过回调获取用户在群组、关系链、单聊消息、在线状态等方面状态改变时的消息，根据在群聊中用户发普通文本消息以及送礼消息来改变用户的等级。
回调相关操作请参见 回调配置 和 第三方回调简介。

步骤5：通过安全打击来实现敏感词的拦截

安全打击能力一直是直播、社交、咨询等领域中必不可少的能力。在直播场景中，如果涉嫌传播敏感内容，后果非常严重。即时通信 IM 提供敏感词过滤功能，默认支持对部分国家领导人名进行敏感词过滤，您也可以增值购买 安全打击服务。

步骤6：使用云直播生成推流地址并开始直播

腾讯云标准直播助力电商平台发展，使商家更加全面地传递了商品信息，促进用户的有效决策，降低营销成本，增加成交量。
在开始直播前，您需要通过 地址生成器 生成推流地址，然后再进行 直播推流。

相关文档

折扣活动

微信订阅号客服系统

最近更新时间：2020-05-11 11:37:47

本文以使用 Node.js 开发一个简单常见的客服场景 Demo 为例，介绍微信订阅号集成腾讯云即时通信 IM 的基本流程。

> 说明：
> 示例仅供参考，正式上线前需要进一步完善，例如服务器负载均衡、接口并发控制、信息持久化存储等。此类优化操作不在本文介绍范围内，请开发者根据实际情况自行实现。

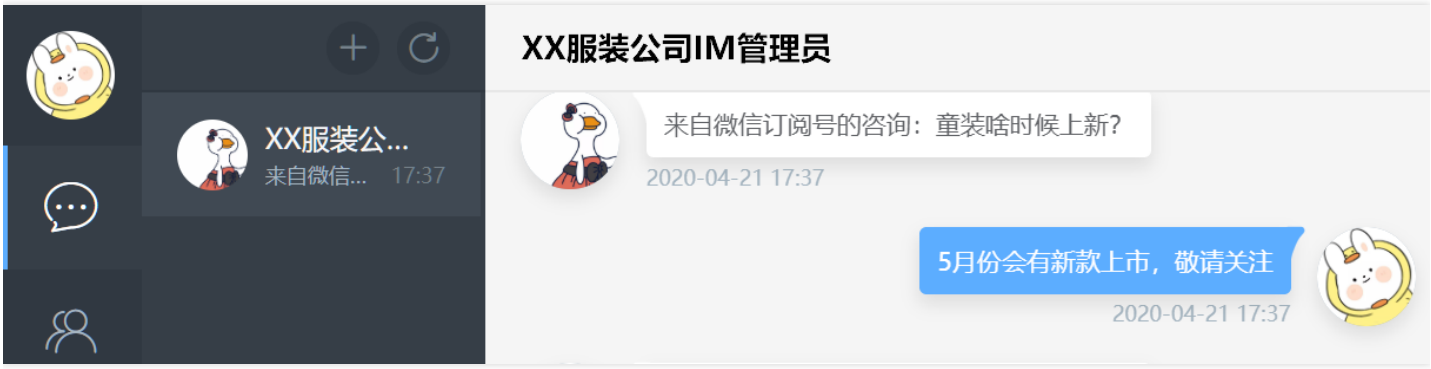
场景流程及效果图

本文 Demo 场景的基本流程如下：

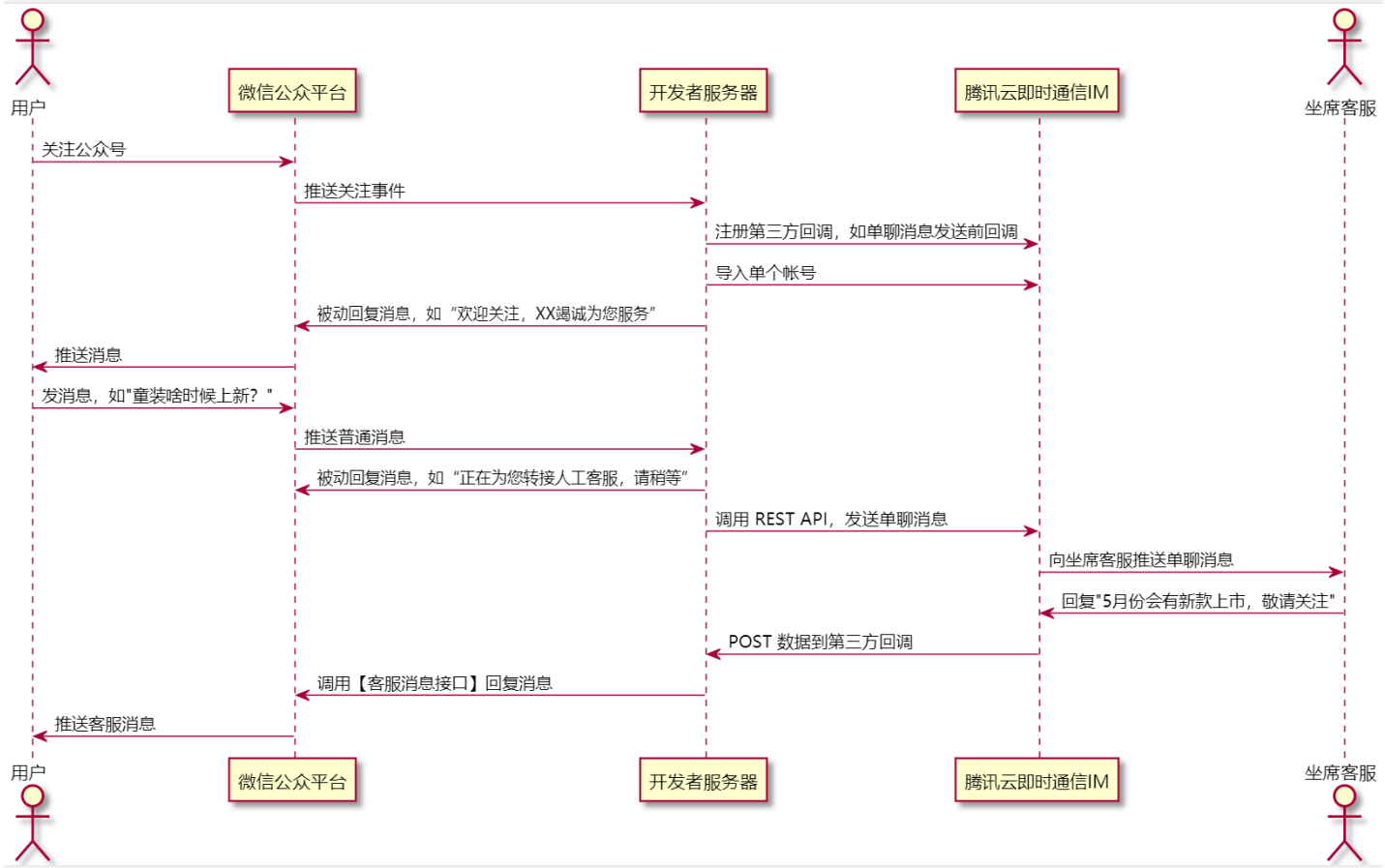
1. 客户通过某服装电商订阅号询问“童装啥时候上新？”。
2. 客户的咨询消息经过腾讯云 IM 系统传输至此服装电商的坐席客服。
3. 客服人员回复“5月份会上新，敬请关注！”，消息经过腾讯云 IM 系统和微信传输推送给客户。

客户侧效果图如下：

坐席客服侧效果图如下：



场景流程图如下：



注意事项

- 消息传输链路较长，可能会影响消息收发耗时。
- 个人注册的订阅号，不能使用微信公众平台的【客服消息】接口向订阅者主动推送消息。

前提条件

- 准备一台可以运行 Node.js 的公网开发服务器或云服务器。
- 注册 微信订阅号或服务号。
- 详细阅读 微信公众平台开发文档。
- 已 创建即时通信 IM 应用。
- 需提前 逐个导入 或 批量导入 即时通信 IM 用户帐号，例如 user0 和 user1。

参考文档

- API 文档
- 第三方回调
- 微信公众平台开发指南
- Express 框架教程

操作步骤

步骤1：创建开发项目并安装依赖

```
npm init -y

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

// 加密模块
npm i crypto@latest --save

// 解析 xml 的工具
npm i xml2js@latest --save

// 发起 http 请求
npm i axios@latest --save

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

步骤2：填入 IM 应用信息并计算 UserSig

```
// ------------ IM ------------
const IMAxios = axios.create({
timeout: 10000,
headers: {
```

```
'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8',
},
});

// 已导入 IM 帐号系统的用户 ID 映射表，非持久化存储，作 Demo 快速检索用，生产环境请用别的技术方案
const importedAccountMap = new Map();
// IM 应用及 App 管理员信息，请登录 即时通信 IM 控制台 获取
const SDKAppID = 0; // 填入 IM 应用的 SDKAppID
const secrectKey = ''; // 填入 IM 应用的密钥
const AppAdmin = 'user0'; // 设置 user0 为 App 管理员帐号
const kfAccount1 = 'user1'; // 设置 user1 为一个坐席客服帐号
// 计算 UserSig，调用 REST API 时需要用到，详细操作请参考 Github
const api = new TLSSigAPIv2.Api(SDKAppID, secrectKey);
const userSig = api.genSig(AppAdmin, 86400*180);
console.log('userSig:', userSig);
```

步骤3：配置 URL 和 Token

> 说明：
> 此指引文档是直接参考微信公众平台开发指南所写，若有变动，请以 接入指南 为准。

1. 登录订阅号管理后台。
2. 选择【基本配置】，勾选协议成为开发者。
3. 单击【修改配置】，填写相关信息：
   - URL：服务器地址，用作接收微信消息和事件的接口 URL，必填参数。
   - Token：可任意填写，用作生成签名，该 Token 会和接口 URL 中包含的 Token 进行比对，从而验证安全性，必填参数。
   - EncodingAESKey：手动填写或随机生成，用作消息体加解密密钥，选填参数。

步骤4：启动 Web 服务监听端口，并正确响应微信发送的 Token 验证

```
const express = require('express'); // express 框架
const crypto = require('crypto'); // 加密模块
const util = require('util');
const xml2js = require('xml2js'); // 解析 xml
const axios = require('axios'); // 发起 http 请求
const TLSSigAPIv2 = require('tls-sig-api-v2'); // 计算 userSig

// ------------ Web 服务 ------------
var app = express();
// Token 需在【订阅号管理后台】>【基本配置】设置
```

```
// 处理所有进入80端口的 get 请求
app.get('/', function(req, res) {
// ------------ 接入微信公众平台 ------------
// 详细请参考 微信官方文档
// 获取微信服务器 Get 请求的参数 signature、timestamp、nonce、echostr
var signature = req.query.signature; // 微信加密签名
var timestamp = req.query.timestamp; // 时间戳
var nonce = req.query.nonce; // 随机数
var echostr = req.query.echostr; // 随机字符串

// 将 token、timestamp、nonce 三个参数进行字典序排序
var array = [myToken, timestamp, nonce];
array.sort();

// 将三个参数字符串拼接成一个字符串进行 sha1 加密
var tempStr = array.join('');
const hashCode = crypto.createHash('sha1'); // 创建加密类型
var resultCode = hashCode.update(tempStr,'utf8').digest('hex'); // 对传入的字符串进行加密

// 开发者获得加密后的字符串可与 signature 对比，标识该请求来源于微信
if (resultCode === signature) {
res.send(echostr);
} else {
res.send('404 not found');
}
});

// 监听80端口
app.listen(80);
```

步骤5：实现开发者服务器侧业务逻辑

- 收到微信推送的关注事件时，调用 导入单个帐号 或 导入多个帐号 API 向帐号系统导入帐号。
- 收到微信推送的关注事件时，被动回复消息。
- 收到微信推送的取消关注事件时，调用 删除帐号 API 将该帐号从帐号系统删除。
- 收到微信推送的普通消息时，调用 单发单聊消息 API 向客服帐号发单聊消息。

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

// 生成 wx 文本回复的 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;
}

/**
* 向 IM 帐号系统导入用户
* @param {String} userID 要导入的用户 ID
*/
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);
})
});
}

/**
* 从 IM 帐号系统删除用户
* @param {String} userID 要删除的用户 ID
*/
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);
})
});
}

/**
* 单发单聊消息
*/
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&
usersig=%s&random=%s&contenttype=json',
SDKAppID, AppAdmin, userSig, genRandom());
console.log('sendC2CTextMessage url:', url);
IMAxios({
url: url,
data: {
"SyncOtherMachine": 2, // 消息不同步至发送方。若希望将消息同步至 From_Account，则 SyncOtherMac
hine 填写1。
```

```
"To_Account": userID,
"MsgLifeTime":60, // 消息保存60秒
"MsgRandom": 1287657,
"MsgTimeStamp": Math.floor(Date.now() / 1000), // 单位为秒，且必须是整数
"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);
});
});
}

// 处理微信的 post 请求
app.post('/', function(req, res) {
var buffer = [];
// 监听 data 事件，用于接收数据
req.on('data', function(data) {
buffer.push(data);
});
// 监听 end 事件，用于处理接收完成的数据
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; // 接收方微信
var fromUser = wxXMLData.FromUserName;// 发送仿微信
if (wxXMLData.Event) { // 处理事件类型
switch (wxXMLData.Event) {
case "subscribe": // 关注订阅号
res.send(genWxTextReplyXML(fromUser, toUser, '欢迎关注，XX竭诚为您服务！'));
importAccount(fromUser).then(() => {
// 记录已导入用户的 ID
importedAccountMap.set(fromUser, 1);
});
break;
case "unsubscribe": // 取消关注
deleteAccount(fromUser).then(() => {
importedAccountMap.delete(fromUser);
});
res.send("success");
break;
}
} else { // 处理消息类型
switch (wxXMLData.MsgType) {
case "text":
// 处理文本消息
sendC2CTextMessage(kfAccount1, '来自微信订阅号的咨询：' + wxXMLData.Content).then(() => {
console.log('发送C2C消息成功');
}).catch((error) => {
console.log('发送C2C消息失败');
});
break;
case "image":
// 处理图片消息
break;
case "voice":
// 处理语音消息
break;
case "video":
// 处理视频消息
break;
case "shortvideo":
// 处理小视频消息
break;
case "location":
```

```
// 处理发送地理位置
break;
case "link":
// 处理点击链接消息
break;
default:
break;
}
res.send(genWxTextReplyXML(fromUser, toUser, '正在为您转接人工客服，请稍等'));
}
}
})
});
});
```

步骤6：注册并处理 IM 第三方回调

```
// 处理 IM 第三方回调的 post 请求
app.post('/imcallback', function(req, res) {
var buffer = [];
// 监听 data 事件 用于接收数据
req.on('data', function(data) {
buffer.push(data);
});
// 监听 end 事件 用于处理接收完成的数据
req.on('end', function() {
const tmpStr = Buffer.concat(buffer).toString('utf-8');
console.log('imcallback', tmpStr);
const imData = JSON.parse(tmpStr);
// kfAccount1 发的消息推送给客户
if (imData.From_Account === kfAccount1) {
// 组包消息，并通过微信的【客服消息】接口，向指定的用户推送消息
// 注意！个人注册的订阅号不支持使用此接口，详情请参见 客服消息
}

res.send({
"ActionStatus": "OK",
"ErrorInfo": "",
"ErrorCode": 0 // 0表示允许发言，1表示拒绝发言
});
});
});
```