语音通话

开发指南

文档版本 01

发布日期 2024-12-19

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

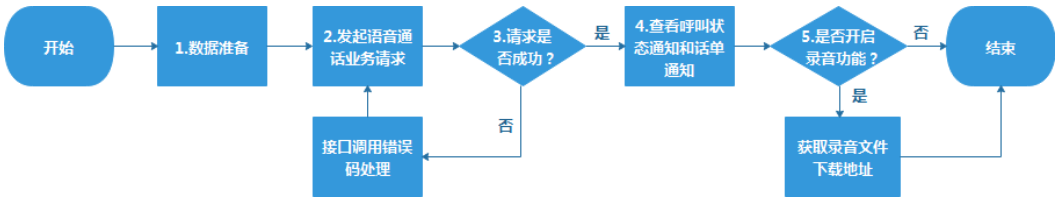
您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 新手必读

语音通话二次开发业务整体流程如下：



1. 参考**申请资源**，获取调用语音通话API的关联数据。
2. 参考**代码样例**调用语音通话API，发起语音回呼/语音通知/语音验证码请求。
3. 根据请求响应消息，判断请求是否成功。
   - 请求成功 => **4**
   - 请求失败 => 参考**API错误码**，修正后重新执行**2**
4. 参考**挂机原因值、Q850原因值、呼叫拆线点**查看语音通话平台推送的呼叫状态通知和话单通知。
5. （仅适用于语音回呼）根据添加应用时的“是否开启录音功能”和语音回呼请求中“recordFlag”参数，判断是否开启了录音功能。
   - 是 =>**获取录音文件下载地址**=>结束
   - 否 => 结束

说明

语音回呼支持录音功能，语音通知和语音验证码不支持录音功能。

步骤**5**仅对语音回呼有效，语音通知和语音验证码在第**4**步之后直接结束流程。

2 开发流程

流程	详细介绍
开发准备	客户在接入华为语音通话服务时，根据购买的业务类型需要提前准备资源，包括申请相关资源、获取代码样例、制作放音文件。
线下开发	客户根据业务能力的代码样例，在线下完成业务应用的代码开发以及调试。
商业发布	线下开发完成，确认业务应用代码可正常调用接口后，正式发布此应用在行业使用。

3 开发准备

3.1 申请资源

语音回呼

表 3-1 语音回呼需准备的资源

资源项	对应接口参数	用途	获取方式
APP_Key	X-AKSK	X-AKSK鉴权所需参数。	登录管理控制台，从“**应用管理**”页面获取。 创建应用，请参考**添加应用**。
APP_Secret			
APP接入地址	-	API调用的基地址。	
访问URI	-	语音回呼场景API的接口访问URI。	从**语音回呼场景API**获取。
主叫端显示的号码	displayNbr	主叫端接收到平台来电时的显示号码。可申请多个。该号码可以与displayCalleeNbr配置为同一个号码，也可以配置为不同号码。	从**订购号码**页面申请。 号码下发后在**号码管理**页面获取。
被叫端显示的号码	displayCalleeNbr	被叫端接收到平台来电时的显示号码。该号码可以与displayNbr配置为同一个号码，也可以配置为不同号码。	

资源项	对应接口参数	用途	获取方式
放音文件（可选）	lastMinVoice	若设置了最大通话时长，平台会于最后一分钟时进行放音提示。 • 若需使用个性化放音，可提交该资源。 • 若不提交，使用默认放音“本次通话时长还剩1分钟”。	登录管理控制台，从“**放音文件管理**”页面获取。 上传放音文件，请参考**添加放音文件**。 具体制作方法参见**制作放音文件**。
	waitVoice	主叫接听平台来电后的等待音。 • 若需使用个性化放音，可提交该资源。 • 若不提交，使用默认回铃音，例如：“嘟...嘟...”。	
呼叫状态通知URL（可选）	statusUrl	接收呼叫过程中状态信息（振铃、应答、挂机等）的服务器地址。若需订阅呼叫状态通知，可提交该资源。	提前准备可用的服务器地址，创建应用时填写，或调用接口时填写。 创建应用，请参考**添加应用**。
话单通知URL（可选）	feeUrl	接收呼叫结束后产生话单的服务器地址。若需订阅话单通知，可提交该资源。	

准备的数据与语音回呼场景API请求参数关联关系如下：

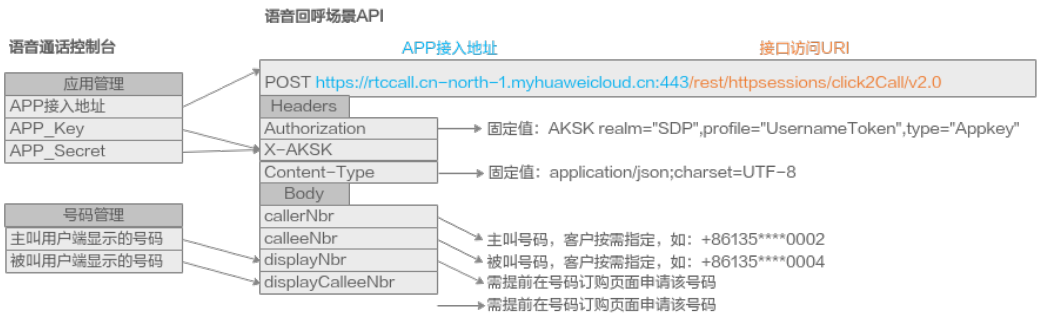


表 3-2 语音回呼录音功能需准备的资源

资源项	对应接口参数	用途	获取方式
放音文件（可选）	recordHintTone	平台会于录音前放音，提示通话用户。 • 若需使用个性化放音，可提交该资源。 • 若不提交，使用默认放音。	登录管理控制台，从“**放音文件管理**”页面获取。 上传放音文件，请参考**添加放音文件**。 具体制作方法参见**制作放音文件**。
获取录音文件			
录音文件名	fileName	下载录音文件时使用。	通过“话单通知API”的recordObjectName参数获取。
录音文件存储的服务器域名	recordDomain		通过“话单通知API”的recordDomain参数获取。

说明

- 使用录音功能，需在**添加应用**时开通。
- 使用录音功能必须订阅话单通知。不订阅话单通知，则无法获取下载录音文件的关键参数。

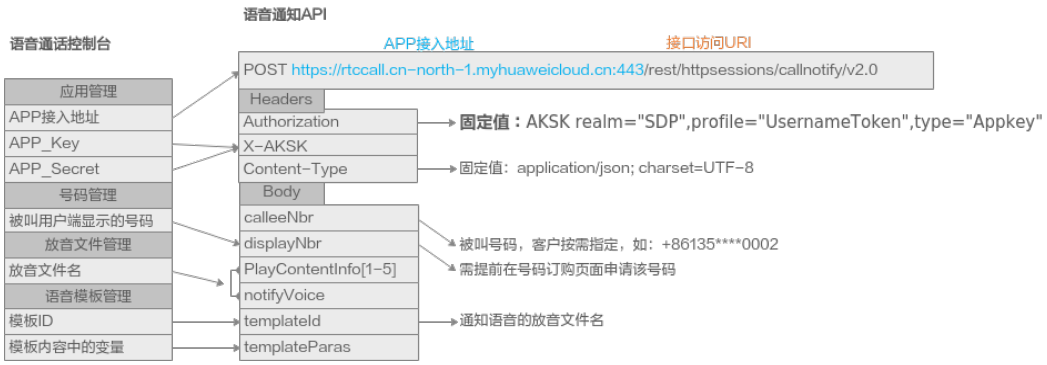
语音通知

表 3-3 语音通知需准备的资源

资源项	对应接口参数	用途	获取方式
APP_Key	X-AKSK	X-AKSK鉴权所需参数。	登录管理控制台，从“**应用管理**”页面获取。 创建应用，请参考**添加应用**。
APP_Secret			
APP接入地址	-	API调用的基地址。	
访问URI	-	语音通知API的接口访问URI。	从**语音通知API**获取。
被叫用户端显示的号码	displayNbr	用户接收到平台来电时的显示号码。可申请多个。	从**订购号码**页面申请。 号码下发后在**号码管理**页面获取。

资源项	对应接口参数	用途	获取方式
放音文件	notifyVoice	音频文件，用户接听平台来电后的语音通知内容。	登录管理控制台，从“**放音文件管理**”页面获取。 上传放音文件，请参考**添加放音文件**。 具体制作方法参见**制作放音文件**。
语音通知模板ID（可选）	templateId	• 仅接口版本为v2.0时需申请。 • 语音通知模板唯一标识。 • 语音通知模板为文本格式，平台将其转化为用户接听来电后的语音通知。可申请多个语音通知模板。	登录管理控制台，从“**语音模板管理**”页面获取。 添加语音通知模板，请参考**添加语音模板**。
呼叫状态通知URL（可选）	statusUrl	接收呼叫过程中状态信息（振铃、应答、挂机等）的服务器地址。若需订阅呼叫状态通知，可提交该资源。	提前准备可用的服务器地址，创建应用时填写，或调用接口时填写。 创建应用，请参考**添加应用**。
话单通知URL（可选）	feeUrl	接收呼叫结束后产生话单的服务器地址。若需订阅话单通知，可提交该资源。	

准备的数据与语音通知API请求参数关联关系如下：

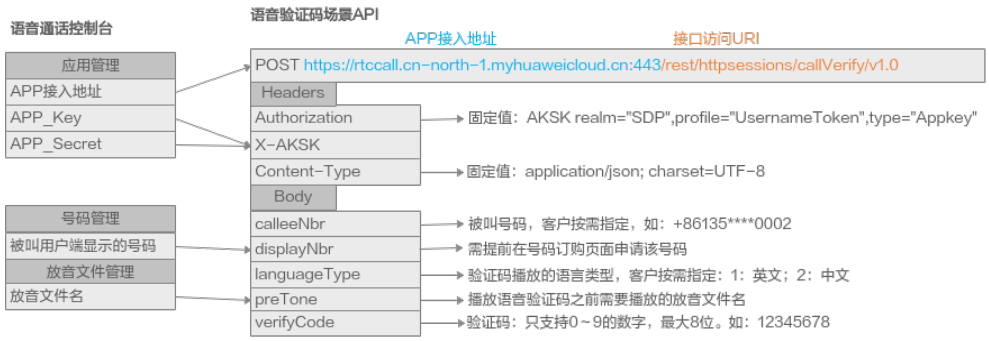


语音验证码

表 3-4 语音验证码需准备的资源

资源项	对应接口参数	用途	获取方式
APP_Key	X-AKSK	X-AKSK鉴权所需参数。	登录管理控制台，从“**应用管理**”页面获取。 创建应用，请参考**添加应用**。
APP_Secret			
APP接入地址	-	API调用的基地址。	
访问URI	-	语音验证码场景API的接口访问URI。	从**语音验证码场景API**获取。
被叫用户端显示的号码	displayNbr	用户接收到平台来电时的显示号码。可申请多个。	从**订购号码**页面申请。 号码下发后在**号码管理**页面获取。
放音文件	preTone	播放语音验证码之前播放的放音文件。	登录管理控制台，从“**放音文件管理**”页面获取。 上传放音文件，请参考**添加放音文件**。 具体制作方法参见**制作放音文件**。
放音文件（可选）	posTone	播放语音验证码之后播放的放音文件。 • 若使用个性化放音，可提交该资源。 • 若不申请，平台将在语音验证码播放完毕后结束通话。	
呼叫状态通知URL（可选）	statusUrl	接收呼叫过程中状态信息（振铃、应答、挂机等）的服务器地址。若需订阅呼叫状态通知，可提交该资源。	提前准备可用的服务器地址，创建应用时填写，或调用接口时填写。 创建应用，请参考**添加应用**。
话单通知URL（可选）	feeUrl	接收呼叫结束后产生话单的服务器地址。若需订阅话单通知，可提交该资源。	

准备的数据与语音验证码API请求参数关联关系如下：



3.2 获取代码样例

语音通话平台提供代码样例作为参考，可根据需求更改适配，快速开发，提高开发效率，节省开发时间。

语音回呼代码样例

Node.js：**点击查看**

Java：**点击查看**

Python：**点击查看**

PHP：**点击查看**

C#：**点击查看**

语音通知代码样例

Node.js：**点击查看**

Java：**点击查看**

Python：**点击查看**

PHP：**点击查看**

C#：**点击查看**

语音验证码代码样例

Node.js：**点击查看**

Java：**点击查看**

Python：**点击查看**

PHP：**点击查看**

C#：**点击查看**

3.3 制作放音文件

平台对放音文件的规格有约束，标准规格为A-Law、8000 Hz采样、单声道的Wave文件，当不满足规格时需要执行该任务转换放音文件的格式。

获取工具

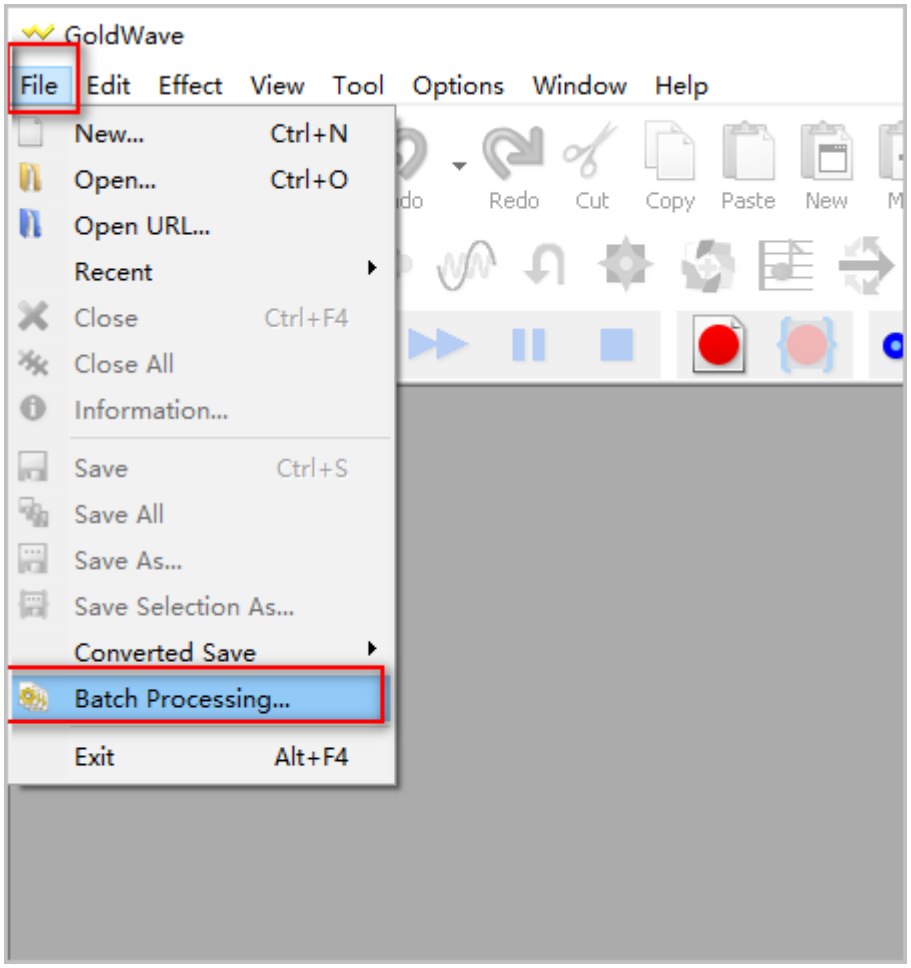
请访问GoldWave官网，根据网页提示获取与PC操作系统对应版本的GoldWave。

制作放音文件

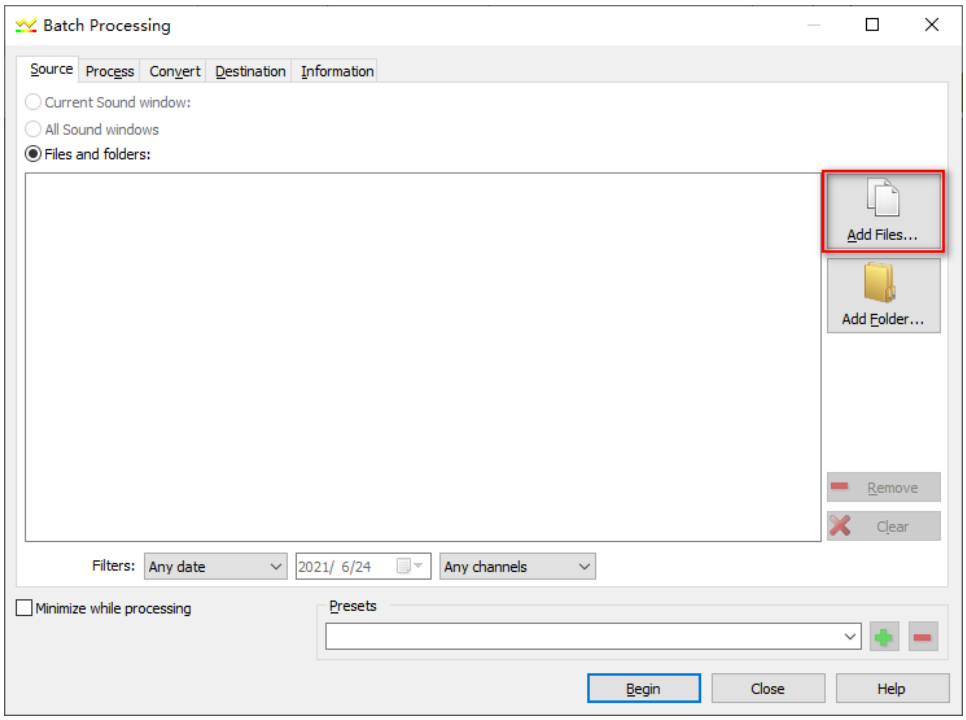
说明

- 本章节中的界面截图以GoldWave v6.55版本为例，实际操作时请以软件实际界面为准。
- 平台要求文件名称只能由数字、字母和特殊字符“-”、“_”、“.”、“@”组成，例如：wait_voice1.wav。若文件名称不符合要求，请更改文件名。

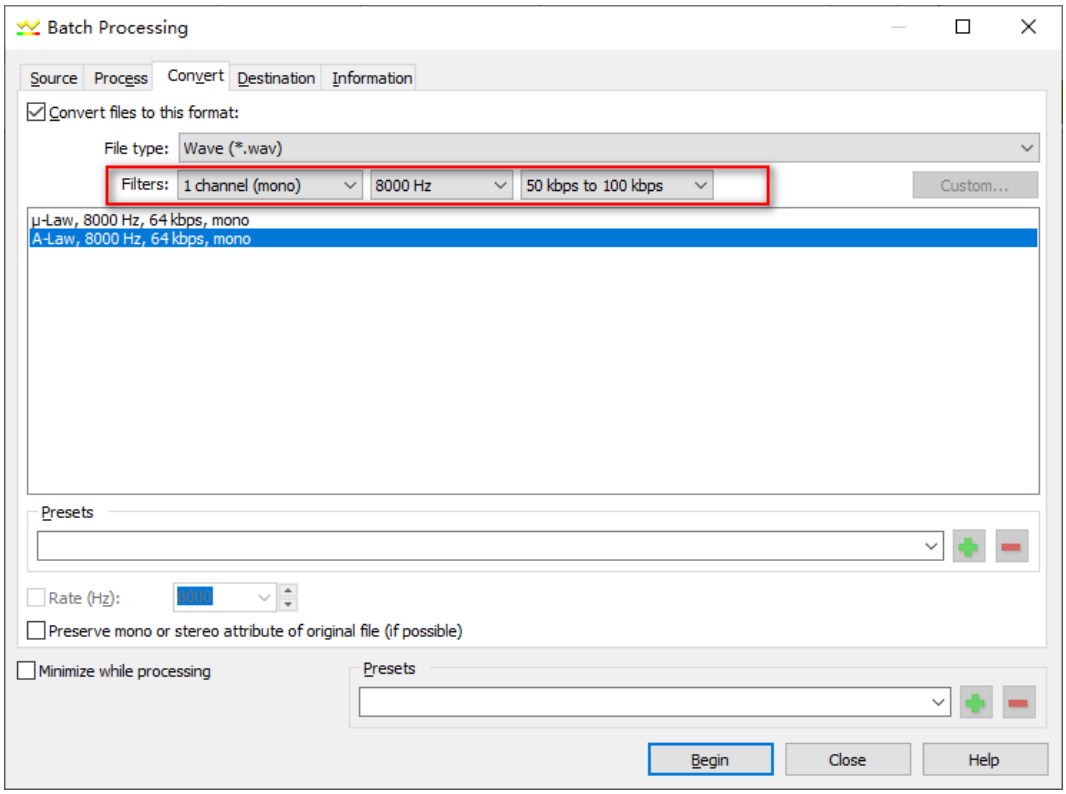
步骤1 创建批处理。



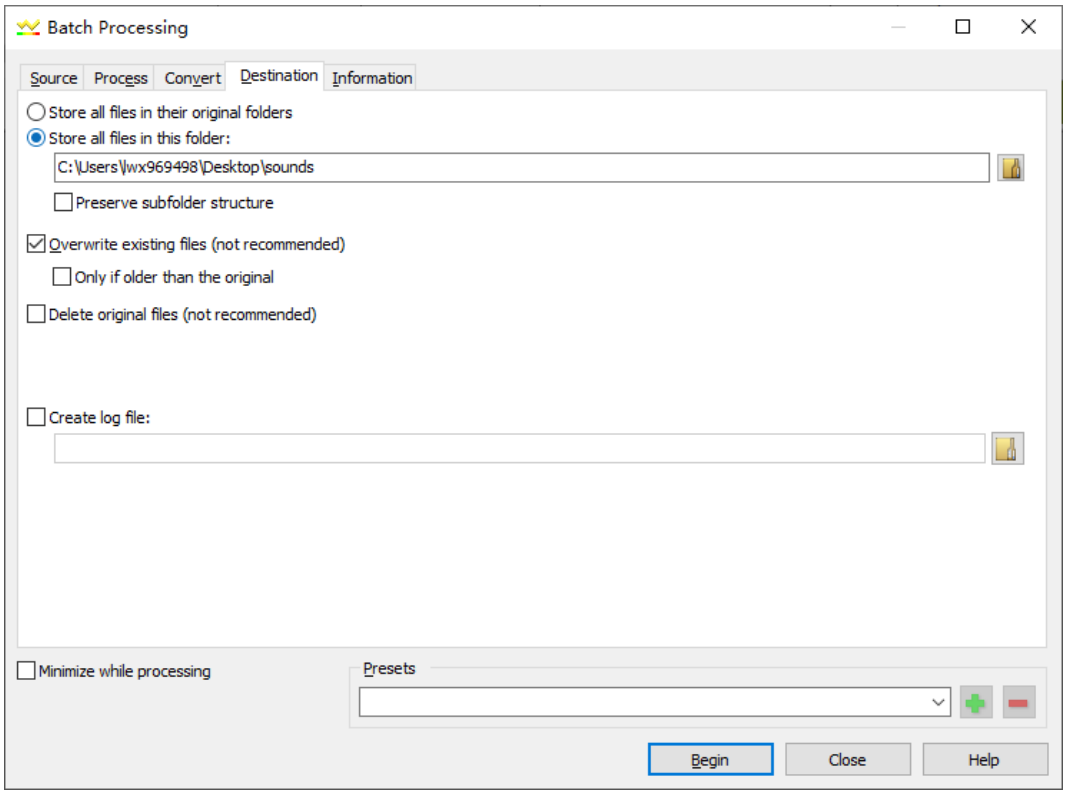
步骤2 添加需要处理的文件/文件夹，因最终生成的文件要求不大于2M，建议源文件不大于6M。



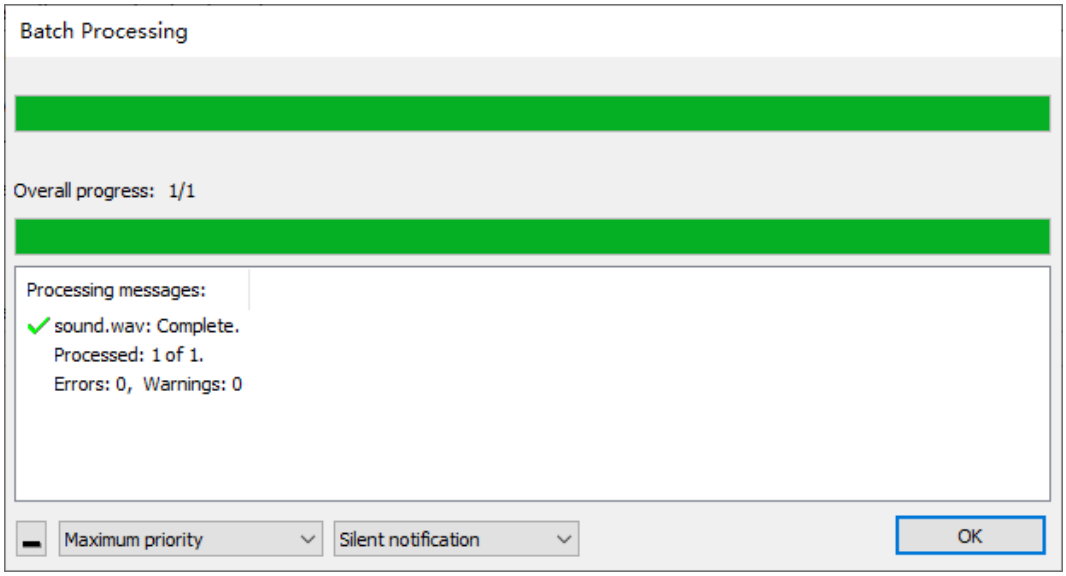
步骤3 设置转换文件的格式（A-Law、8000 Hz、单声道）。



步骤4 设置输出文件的路径，点击“Begin”,开始转换文件。



步骤5 转换完成后，点击“OK”，去目标文件夹获取转换后的Wave文件。



----结束

4 线下开发

4.1 语音回呼代码样例

4.1.1 Node.js

注：为节省开发时间，建议先使用Node.js代码样例进行调测，熟悉接口使用后，再参考Java、python、PHP或C#代码样例，结合**接口文档**进行功能开发。

样例	**语音回呼场景API**、**获取录音文件下载地址API**、**呼叫状态通知API**、**话单通知API**
环境要求	Node.js 4.4.4及以上版本。
引用库	-

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音回呼场景 API”代码样例

```
/*jshint esversion: 6 */
var https = require('https');
var data = require('./data.js');
var util = require('./reqUtil.js');
var record = require('./getRecordLink.js');
/**
 * voiceCallAPI
 * @param displayNbr
 * @param callerNbr
 * @param displayCalleeNbr
 * @param calleeNbr
```



```
 * @returns
 */
function voiceCallAPI(displayNbr, callerNbr, displayCalleeNbr, calleeNbr) {
    if(displayNbr === undefined || displayNbr === null || callerNbr === undefined || callerNbr === null) {
        return;
    }
    if(displayCalleeNbr === undefined || displayCalleeNbr === null || calleeNbr === undefined || calleeNbr
=== null) {
        return;
    }

    var method = 'POST';
    var uri = '/rest/httpsessions/click2Call/v2.0';
    var xaksk = util.buildAKSKHeader(data.data.click2call_appid, data.data.click2call_secret);

    var options = util.createOptions(method, uri, null, xaksk);

    var body = {
            /* 必填参数 */
            'displayNbr': displayNbr,//主叫用户手机终端的来电显示号码。
            'callerNbr': callerNbr,//发起呼叫时所使用的主叫号码。
            'displayCalleeNbr': displayCalleeNbr,//被叫用户终端的来电显示号码。
            'calleeNbr': calleeNbr//发起呼叫时所拨打的被叫号码。
            /* 选填参数 */
//            'bindNbr': '+86123456789', //CallEnabler业务号码,即绑定号码
//            'maxDuration': 0, //允许单次通话进行的最长时间
//            'lastMinVoice': 'lastmin_voice1.wav', //最后一分钟放音提示音
//            'lastMinToUE': 'both', //最后一分钟放音的播放对象
//            'playPreVoice': 'false', //设置主叫（callerNbr）应答语音回呼后,呼叫被叫（calleeNbr）前,是否向主叫
（callerNbr）播放提示音
//            'preVoice': 'pre_voice1.wav', //设置主叫（callerNbr）应答语音回呼后,呼叫被叫（calleeNbr）前向主
叫播放的提示音
//            'waitVoice': 'wait_voice1.wav', //设置主叫应答语音回呼后的等待音
//            'calleeMedia': 'all', //指定被叫的媒体音播放方式
//            'statusUrl': 'aHR0cDovLzIxOC40LjMzLjU1Ojg4ODgvdGVzdA==', //要获取通话状态需要在请求中加入
statusUrl
//            'feeUrl': 'aHR0cDovLzIxOC40LjMzLjU1Ojg4ODgvdGVzdA==', //要获取话单需要在请求中加入feeUrl
//            'recordFlag': 'false', //与调测信息中的recordFlag保持一致
//            'recordHintTone': 'recordhint_voice1.wav', //设置使用录音功能的提示音
//            'partyTypeRequiredInDisconnect': 'false', //disconnect状态是否需要携带通话主动挂机的用户类型
//            'returnIdlePort': 'false', //指示是否需要返回平台空闲呼叫端口数量
//            'userData': 'customerId123' //设置用户的附属信息
    };
    var req = https.request(options, function (res) {
        var resHeaders = JSON.stringify(res.headers);
        res.setEncoding('utf8');
        res.on('data', function (chunk) {
            console.log(chunk);
        });
    });

    req.on('error', function(e) {
        console.error('problem with request: ' + e);
    });
    console.log(JSON.stringify(body));
    req.write(JSON.stringify(body));
    req.end();
}
voiceCallAPI('+8653159511234', '+8613500000001', '+8653159511234', '+8613500000002');
var xaksk = util.buildAKSKHeader(data.data.click2call_appid, data.data.click2call_secret);
var location = record.getRecordLinkAPI('1200_366_0_20161228102743.wav', 'ostor.huawei.com', xaksk);
console.log('The record file download link is: ' + location);
```

“获取录音文件下载地址 API”代码样例

```
/*jshint esversion: 6 */
var https = require('https');
var util = require('./reqUtil.js');
var querystring = require('querystring');
```

```
/**
 * Get the download link of record file.
 * @param fileName  record file name
 * @param recordDomain  domain name of record file restore server
 * @returns
 */
function getRecordLinkAPI(fileName, recordDomain, xaksk) {
    if(fileName === undefined || fileName === null || recordDomain === undefined || recordDomain === null){
        return;
    }
    if(xaksk === undefined || xaksk === null){
        return;
    }

    var location;

    var method = 'GET';
    var uri = '/rest/provision/voice/record/v1.0';
    var queryParams = querystring.stringify({'fileName': fileName, 'recordDomain': recordDomain});

    var options = util.createOptions(method, uri, queryParams, xaksk);

    var req = https.request(options, function (res) {
        if(301 === res.statusCode){
            location = Object.getOwnPropertyDescriptor(res.headers, 'location').value;
        }
        var resHeaders = JSON.stringify(res.headers);
        res.setEncoding('utf8');
        res.on('data', function (chunk) {
            console.log('resp:', chunk); //打印响应数据
        });
    });

    req.on('error', function(e) {
        console.error('problem with request: ' + e); //打印错误信息
    });
    req.end(); //结束请求
    return location;
}
module.exports = {
        getRecordLinkAPI
};
```

“呼叫状态通知 API”代码样例

```
/**
 * 呼叫事件通知
 * 客户平台收到RTC业务平台的呼叫事件通知的接口通知
 */
//呼叫事件通知样例
var jsonBody = JSON.stringify({
    'eventType': 'callout',
    'statusInfo': {
        'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp': '2019-01-24 03:04:24',
        'caller': '+8613800000022',
        'called': '+8613800000021'
    }
});
console.log('jsonBody:', jsonBody);
/**
 * 呼叫事件通知
 * @brief 详细内容以接口文档为准
 * @param jsonBody
 */
function onCallEvent(jsonBody) {
    var jsonObj = JSON.parse(jsonBody); //将通知消息解析为jsonObj
    var eventType = jsonObj.eventType; //通知事件类型
```

```
if ('fee' === eventType) {
    console.log('EventType error:', eventType);
    return;
}

if (!jsonObj.hasOwnProperty('statusInfo')) {
    console.log('param error: no statusInfo.');
    return;
}
var statusInfo = jsonObj.statusInfo; //呼叫状态事件信息

console.log('eventType:', eventType); //打印通知事件类型

//callout：呼出事件
if ('callout' === eventType) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.hasOwnProperty('sessionId')) {
        console.log('sessionId:', statusInfo.sessionId);
    }
    return;
}
//alerting：振铃事件
if ('alerting' === eventType) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.hasOwnProperty('sessionId')) {
        console.log('sessionId:', statusInfo.sessionId);
    }
    return;
}
//answer：应答事件
if ('answer' === eventType) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.hasOwnProperty('sessionId')) {
        console.log('sessionId:', statusInfo.sessionId);
    }
    return;
}
//collectInfo：放音收号结果事件,仅应用于语音通知场景
if ('collectInfo' === eventType) {
    /**
     * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'sessionId': 通话链路的标识ID
     * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
```



```
        */
        if (statusInfo.hasOwnProperty('digitInfo')) {
            console.log('digitInfo:', statusInfo.digitInfo);
        }
        return;
    }
    //disconnect：挂机事件
    if ('disconnect' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
         * 'stateCode': 通话挂机的原因值
         * 'stateDesc': 通话挂机的原因值的描述
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
}
//呼叫事件处理
onCallEvent(jsonBody);
```

“话单通知 API”代码样例

```
/**
 * 话单通知
 * 客户平台收到RTC业务平台的话单通知的接口通知
 */
//话单通知样例
var jsonBody = JSON.stringify({
    'eventType': 'fee',
    'feeLst': [
        {
            'direction': 0,
            'spId': 'CaaS_Test_01',
            'appKey': 'ka4kESI5s3YyurL1wpx63s9YnEm2',
            'icid': 'CAE-20190124110424-12019775',
            'bindNum': '+8675512345678',
            'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
            'callerNum': '+8613800000022',
            'calleeNum': '+8613800000021',
            'fwdDisplayNum': '+8613800000022',
            'fwdDstNum': '+8613866887021',
            'fwdStartTime': '2019-01-24 03:04:31',
            'fwdAlertingTime': '2019-01-24 03:04:36',
            'fwdAnswerTime': '2019-01-24 03:04:38',
            'callEndTime': '2019-01-24 03:04:49',
            'fwdUnaswRsn': 0,
            'ulFailReason': 0,
            'sipStatusCode': 0,
            'callOutStartTime': '2019-01-24 03:04:24',
            'callOutAlertingTime': '2019-01-24 03:04:27',
            'callOutAnswerTime': '2019-01-24 03:04:31',
            'callOutUnaswRsn': 0,
            'recordFlag': 0,
            'ttsPlayTimes': 0,
            'ttsTransDuration': 0,
            'serviceType': '002',
            'hostName': 'callenabler245.huaweicaas.com'
        }
    ]
});
```

```
console.log('jsonBody:', jsonBody);
/**
 * 话单通知
 * @brief 详细内容以接口文档为准
 * @param jsonBody
 */
function onFeeEvent(jsonBody) {
    var jsonObj = JSON.parse(jsonBody); //将通知消息解析为jsonObj
    var eventType = jsonObj.eventType; //通知事件类型

    if ('fee' !== eventType) {
        console.log('EventType error:', eventType);
        return;
    }

    if (!jsonObj.hasOwnProperty('feeLst')) {
        console.log('param error: no feeLst.');
        return;
    }
    var feeLst = jsonObj.feeLst; //呼叫话单事件信息
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'direction': 通话的呼叫方向,以RTC业务平台为基准
     * 'spId': 客户的云服务账号
     * 'appKey': 应用的app_key
     * 'icid': 呼叫记录的唯一标识
     * 'bindNum': 发起此次呼叫的CallEnabler业务号码
     * 'sessionId': 通话链路的唯一标识
     * 'callerNum': 主叫号码
     * 'calleeNum': 被叫号码
     * 'fwdDisplayNum': 转接呼叫时的显示号码(仅语音回呼场景携带)
     * 'fwdDstNum': 转接呼叫时的转接号码(仅语音回呼场景携带)
     * 'fwdStartTime': 转接呼叫操作的开始时间(仅语音回呼场景携带)
     * 'fwdAlertingTime': 转接呼叫操作后的振铃时间(仅语音回呼场景携带)
     * 'fwdAnswerTime': 转接呼叫操作后的应答时间(仅语音回呼场景携带)
     * 'callEndTime': 呼叫结束时间
     * 'fwdUnaswRsn': 转接呼叫操作失败的Q850原因值
     * 'failTime': 呼入,呼出的失败时间
     * 'ulFailReason': 通话失败的拆线点
     * 'sipStatusCode': 呼入,呼出的失败SIP状态码
     * 'callOutStartTime': Initcall的呼出开始时间
     * 'callOutAlertingTime': Initcall的呼出振铃时间
     * 'callOutAnswerTime': Initcall的呼出应答时间
     * 'callOutUnaswRsn': Initcall的呼出失败的Q850原因值
     * 'dynIVRStartTime': 自定义动态IVR开始时间(仅语音通知场景携带)
     * 'dynIVRPath': 自定义动态IVR按键路径(仅语音通知场景携带)
     * 'recordFlag': 录音标识
     * 'recordStartTime': 录音开始时间(仅语音回呼场景携带)
     * 'recordObjectName': 录音文件名(仅语音回呼场景携带)
     * 'recordBucketName': 录音文件所在的目录名(仅语音回呼场景携带)
     * 'recordDomain': 存放录音文件的域名(仅语音回呼场景携带)
     * 'recordFileDownloadUrl': 录音文件下载地址(仅语音回呼场景携带)
     * 'ttsPlayTimes': 应用TTS功能时,使用TTS的总次数
     * 'ttsTransDuration': 应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
     * 'serviceType': 携带呼叫的业务类型信息
     * 'hostName': 话单生成的服务器设备对应的主机名
     * 'userData': 用户附属信息
     */
    //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
    if (feeLst.length > 1) {
        for ( var i=0; i<feeLst.length; i++) {
            if (feeLst[i].hasOwnProperty('sessionId')) {
                console.log('sessionId:', feeLst[i].sessionId);
            }
        }
    } else if (feeLst.length === 1) {
        if (feeLst[0].hasOwnProperty('sessionId')) {
            console.log('sessionId:', feeLst[0].sessionId);
```

```
        }
    } else {
        console.log('feeLst error: no element.');
    }
}
//话单处理
onFeeEvent(jsonBody);
```

4.1.2 Java

4.1.2.1 公共要求

注：使用前请务必先仔细阅读使用**注意事项**。

样例	语音回呼场景API、获取录音文件下载地址API、呼叫状态与话单通知API
环境要求	JDK 1.6及以上版本。
引用库	**httpclient**、**httpcore**、**httpmime**、**commons-codec**、**commons-logging**、**jackson-databind**、**jackson-annotations**、**jackson-core**、**fastjson**、**log4j**

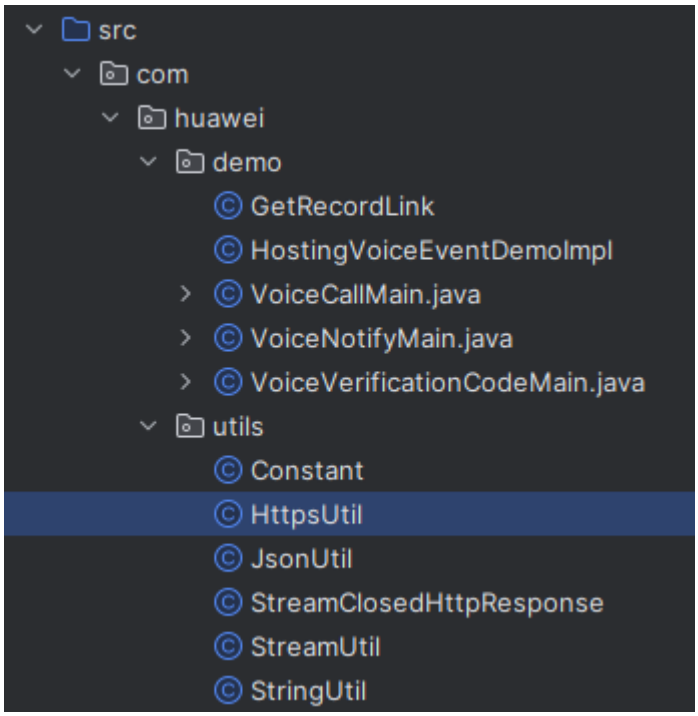
将下列代码样例复制到新建java文件中即可运行。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

公共依赖

使用Java样例时请参考以下工程目录结构：



Constant.java

```
/*
 * File Name: com.huawei.utils.Constant.java
 *
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

public class Constant {

    // purchase and get : base_url,appid,secret

    // please replace the IP and Port, when you use the demo.
    public static final String BASE_URL = "https://{domain}:{port}";

    // please replace the appId and secret, when you use the demo.
    public static final String CLICK2CALL_APPID = "*****";
    public static final String CLICK2CALL_SECRET = "******";
    public static final String CALLNOTIFY_APPID = "******";
    public static final String CALLNOTIFY_SECRET = "******";
    public static final String CALLVERIFY_APPID = "*****";
    public static final String CALLVERIFY_SECRET = "*****";

    // ************************** The following constants do not need to be
    // modified *********************************//

    /*
     * request header
     * 1. HEADER_APP_AUTH
     * 2. HEADER_APP_AKSK
     * 3. AUTH_HEADER_VALUE
     * 4. AKSK_HEADER_FORMAT
     */
```



```
    public static final String HEADER_APP_AUTH = "Authorization";
    public static final String HEADER_APP_AKSK = "X-AKSK";
    public static final String AUTH_HEADER_VALUE = "AKSK realm=\"SDP\",profile=\"UsernameToken\",type=
\"Appkey\"";
    public static final String AKSK_HEADER_FORMAT = "UsernameToken Username=\"%s\",PasswordDigest=
\"%s\",Nonce=\"%s\",Created=\"%s\"";

    /*
     * Voice Call:
     * 1. VOICE_CALL_COMERCIAL
     * 2. VOICE_VERIFICATION_COMERCIAL
     * 3. CALL_NOTIFY_COMERCIAL
     * 4. VOICE_FILE_DOWNLOAD
     */
    public static final String VOICE_CALL_COMERCIAL = BASE_URL + "/rest/httpsessions/click2Call/v2.0";
    public static final String VOICE_VERIFICATION_COMERCIAL = BASE_URL + "/rest/httpsessions/callVerify/
v1.0";
    public static final String CALL_NOTIFY_COMERCIAL = BASE_URL + "/rest/httpsessions/callnotify/v2.0";
    public static final String VOICE_FILE_DOWNLOAD = BASE_URL + "/rest/provision/voice/record/v1.0";
}
```

HttpsUtil.java

```
package com.huawei.utils;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.SSLContextBuilder;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.client.HttpClients;

import java.io.IOException;
import java.net.URISyntaxException;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.List;
import java.util.Map;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

@SuppressWarnings("deprecation")
public class HttpsUtil extends DefaultHttpClient {
    public final static String CONTENT_LENGTH = "Content-Length";

    private static CloseableHttpClient httpClient = getHttpClient();

    public static CloseableHttpClient getHttpClient() {
        SSLContext sslcontext = null;
        try {
            sslcontext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
```



```
            @Override
            public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                return true;
            }
        }).build();
    } catch (KeyManagementException e) {
        return null;
    } catch (NoSuchAlgorithmException e) {
        return null;
    } catch (KeyStoreException e) {
        return null;
    }

    // Allow TLSv1, TLSv1.1, TLSv1.2 protocol only
    SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslcontext,
        new String[] {"TLSv1", "TLSv1.1", "TLSv1.2"}, null,
        SSLConnectionSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);

    CloseableHttpClient httpclient = HttpClients.custom()
        .setDefaultRequestConfig(RequestConfig.custom().setRedirectsEnabled(false).build())
        .setSSLSocketFactory(sslsf).build();

    return httpclient;
}

public HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
        return true;
    }
};

public void trustAllHttpsCertificates() throws Exception {
    TrustManager[] trustAllCerts = new TrustManager[1];
    TrustManager tm = new miTM();
    trustAllCerts[0] = tm;
    SSLContext sc = SSLContext.getInstance("SSL");
    sc.init(null, trustAllCerts, null);
    HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
}

static class miTM implements TrustManager, X509TrustManager {
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    public boolean isServerTrusted(X509Certificate[] certs) {
        return true;
    }

    public boolean isClientTrusted(X509Certificate[] certs) {
        return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String authType)
        throws java.security.cert.CertificateException {
        return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String authType)
        throws java.security.cert.CertificateException {
        return;
    }
}

public StreamClosedHttpResponse doPostJsonGetStatusLine(String url, Map<String, String> headerMap,
String content) {
    HttpPost request = new HttpPost(url);
    addRequestHeader(request, headerMap);
```

```
        request.setEntity(new StringEntity(content, ContentType.APPLICATION_JSON));

        HttpResponse response = executeHttpRequest(request);
        if (null == response) {
            System.out.println("The response body is null.");
        }

        return (StreamClosedHttpResponse) response;
    }

    public HttpResponse doGetWithParas(String url, List<NameValuePair> queryParams, Map<String, String>
headerMap)
        throws Exception {
        HttpGet request = new HttpGet();
        addRequestHeader(request, headerMap);

        URIBuilder builder;
        try {
            builder = new URIBuilder(url);
        } catch (URISyntaxException e) {
            System.out.printf("URISyntaxException: {}", e);
            throw new Exception(e);
        }

        if (queryParams != null && !queryParams.isEmpty()) {
            builder.setParameters(queryParams);
        }
        request.setURI(builder.build());

        return executeHttpRequest(request);
    }

    public StreamClosedHttpResponse doGetWithParasGetStatusLine(String url, List<NameValuePair>
queryParams,
        Map<String, String> headerMap) throws Exception {
        HttpResponse response = doGetWithParas(url, queryParams, headerMap);
        if (null == response) {
            System.out.println("The response body is null.");
        }

        return (StreamClosedHttpResponse) response;
    }

    private static void addRequestHeader(HttpUriRequest request, Map<String, String> headerMap) {
        if (headerMap == null) {
            return;
        }

        for (String headerName : headerMap.keySet()) {
            if (CONTENT_LENGTH.equalsIgnoreCase(headerName)) {
                continue;
            }

            String headerValue = headerMap.get(headerName);
            request.addHeader(headerName, headerValue);
        }
    }

    private HttpResponse executeHttpRequest(HttpUriRequest request) {
        HttpResponse response = null;

        try {
            response = httpClient.execute(request);
        } catch (Exception e) {
            System.out.println("executeHttpRequest failed.");
            System.out.println(e.getStackTrace());
        } finally {
            try {
                response = new StreamClosedHttpResponse(response);
```

```
            } catch (IOException e) {
                System.out.println("IOException: " + e.getMessage());
            }
        }

        return response;
    }
}
```

JsonUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */

package com.huawei.utils;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;

import java.io.IOException;

public class JsonUtil {

    private static ObjectMapper objectMapper;

    static {
        objectMapper = new ObjectMapper();

        // 设置FAIL_ON_EMPTY_BEANS属性,当序列化空对象不要抛异常
        objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, false);

        // 设置FAIL_ON_UNKNOWN_PROPERTIES属性,当JSON字符串中存在Java对象没有的属性,忽略
        objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
    }

    /**
     * Convert Object to JsonString
     *
     * @param jsonObj
     * @return
     */
    public static String jsonObj2Sting(Object jsonObj) {
        String jsonString = null;

        try {
            jsonString = objectMapper.writeValueAsString(jsonObj);
        } catch (IOException e) {
            System.out.printf("pasre json Object[{}] to string failed.", jsonString);
        }

        return jsonString;
    }

    /**
     * Convert JsonString to Simple Object
     *
     * @param jsonString
     * @param cls
     * @return
     */
```

```
    public static <T> T jsonString2SimpleObj(String jsonString, Class<T> cls) {
        T jsonObj = null;

        try {
            jsonObj = objectMapper.readValue(jsonString, cls);
        } catch (IOException e) {
            System.out.printf("pasre json Object[{}] to string failed.", jsonString);
        }

        return jsonObj;
    }
}
```

StreamClosedHttpResponse.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

import java.io.IOException;
import java.util.Locale;

import org.apache.http.Header;
import org.apache.http.HeaderIterator;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.ProtocolVersion;
import org.apache.http.StatusLine;
import org.apache.http.params.HttpParams;

@SuppressWarnings("deprecation")
public class StreamClosedHttpResponse implements HttpResponse {

    private final HttpResponse original;

    private final String content;

    public StreamClosedHttpResponse(final HttpResponse original) throws UnsupportedOperationException,
IOException {
        this.original = original;

        HttpEntity entity = original.getEntity();
        if (entity != null && entity.isStreaming()) {
            String encoding = entity.getContentEncoding() != null ? entity.getContentEncoding().getValue() :
null;
            content = StreamUtil.inputStream2String(entity.getContent(), encoding);
        } else {
            content = null;
        }
    }

    @Override
    public StatusLine getStatusLine() {
        return original.getStatusLine();
    }

    @Override
    public void setStatusLine(final StatusLine statusline) {
        original.setStatusLine(statusline);
    }
```



```
@Override
public void setStatusLine(final ProtocolVersion ver, final int code) {
    original.setStatusLine(ver, code);
}

@Override
public void setStatusLine(final ProtocolVersion ver, final int code, final String reason) {
    original.setStatusLine(ver, code, reason);
}

@Override
public void setStatusCode(final int code) throws IllegalStateException {
    original.setStatusCode(code);
}

@Override
public void setReasonPhrase(final String reason) throws IllegalStateException {
    original.setReasonPhrase(reason);
}

@Override
public HttpEntity getEntity() {
    return original.getEntity();
}

@Override
public void setEntity(final HttpEntity entity) {
    original.setEntity(entity);
}

@Override
public Locale getLocale() {
    return original.getLocale();
}

@Override
public void setLocale(final Locale loc) {
    original.setLocale(loc);
}

@Override
public ProtocolVersion getProtocolVersion() {
    return original.getProtocolVersion();
}

@Override
public boolean containsHeader(final String name) {
    return original.containsHeader(name);
}

@Override
public Header[] getHeaders(final String name) {
    return original.getHeaders(name);
}

@Override
public Header getFirstHeader(final String name) {
    return original.getFirstHeader(name);
}

@Override
public Header getLastHeader(final String name) {
    return original.getLastHeader(name);
}

@Override
public Header[] getAllHeaders() {
    return original.getAllHeaders();
}
```

```
@Override
public void addHeader(final Header header) {
    original.addHeader(header);
}

@Override
public void addHeader(final String name, final String value) {
    original.addHeader(name, value);
}

@Override
public void setHeader(final Header header) {
    original.setHeader(header);
}

@Override
public void setHeader(final String name, final String value) {
    original.setHeader(name, value);
}

@Override
public void setHeaders(final Header[] headers) {
    original.setHeaders(headers);
}

@Override
public void removeHeader(final Header header) {
    original.removeHeader(header);
}

@Override
public void removeHeaders(final String name) {
    original.removeHeaders(name);
}

@Override
public HeaderIterator headerIterator() {
    return original.headerIterator();
}

@Override
public HeaderIterator headerIterator(final String name) {
    return original.headerIterator(name);
}

@Override
public String toString() {
    final StringBuilder sb = new StringBuilder("HttpResponseProxy{");
    sb.append(original);
    sb.append('}');
    return sb.toString();
}

@Override
@Deprecated
public HttpParams getParams() {
    return original.getParams();
}

@Override
@Deprecated
public void setParams(final HttpParams params) {
    original.setParams(params);
}

public String getContent() {
    return content;
```

```
    }
}
```

StreamUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

import java.io.Closeable;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class StreamUtil {

    private static final String DEFAULT_ENCODING = "utf-8";

    public static String inputStream2String(InputStream in, String charsetName) {
        if (in == null) {
            return null;
        }

        InputStreamReader inReader = null;

        try {
            if (StringUtil.strIsNullOrEmpty(charsetName)) {
                inReader = new InputStreamReader(in, DEFAULT_ENCODING);
            } else {
                inReader = new InputStreamReader(in, charsetName);
            }

            int readLen = 0;
            char[] buffer = new char[1024];
            StringBuilder strBuf = new StringBuilder();
            while ((readLen = inReader.read(buffer)) != -1) {
                strBuf.append(buffer, 0, readLen);
            }

            return strBuf.toString();
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            closeStream(inReader);
        }

        return null;
    }

    public static void closeStream(Closeable closeable) {
        if (closeable != null) {
            try {
                closeable.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}
```

StringUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

//import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.text.SimpleDateFormat;
import java.util.Base64; // JDK versionâ‰¥1.8
import java.util.Calendar;
import java.util.Locale;
import java.util.TimeZone;
import java.util.UUID;

//import org.apache.commons.codec.binary.Base64; //JDK version<1.8
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class StringUtil {

    public static boolean strIsNullOrEmpty(String s) {
        return (null == s || s.trim().length() < 1);
    }

    public static String buildAKSKHeader(String appKey, String appSecret) throws Exception {
        if (StringUtil.strIsNullOrEmpty(appKey) || StringUtil.strIsNullOrEmpty(appSecret)) {
            return null;
        }

        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");
        format.setTimeZone(TimeZone.getTimeZone("UTC"));
        Calendar calendar = Calendar.getInstance();
        String time = format.format(calendar.getTime());
        String stNonce = UUID.randomUUID().toString().replace("-", "").toUpperCase(Locale.ROOT);
        String str = stNonce + time;
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(appSecret.getBytes(StandardCharsets.UTF_8), "HmacSHA256"));
        byte[] authBytes = mac.doFinal(str.getBytes(StandardCharsets.UTF_8));
        String passwordDigestBase64Str = encodeBase64(authBytes);
        return String.format(Constant.AKSK_HEADER_FORMAT, appKey, passwordDigestBase64Str, stNonce,
time);
    }

    private static String encodeBase64(byte[] bytes) {
        if (bytes.length == 0) {
            return null;
        } else {
            return new String(org.apache.commons.codec.binary.Base64.encodeBase64(bytes),
StandardCharsets.UTF_8);
        }
    }
}
```

4.1.2.2 代码样例

“语音回呼场景 API”代码样例

```
package com.huawei.demo;

import java.util.HashMap;
import java.util.Map;
```

```
import javax.net.ssl.HttpsURLConnection;

import com.huawei.utils.Constant;
import com.huawei.utils.HttpsUtil;
import com.huawei.utils.JsonUtil;
import com.huawei.utils.StreamClosedHttpResponse;
import com.huawei.utils.StringUtil;

public class VoiceCallMain {
    // 接口通用返回值
    private static String status = "";

    private static String resultcode = "";

    private static String resultdesc = "";

    // 语音回呼接口返回值
    private static String sessionId = "";

    // 语音回呼业务类实体
    public static VoiceCall voiceCallAPI = new VoiceCall();

    // 获取录音文件下载地址实体
    public static GetRecordLink recordLinkAPI = new GetRecordLink();

    // 调用接口成功标识
    private static final String success = "200";

    public static void main(String args[]) throws Exception {

        // TODO 程序前端要求发起语音回呼,调用doVoiceCall方法.
        // 以下代码仅供调试使用,实际开发时请删除
        VoiceCallMain.doVoiceCall("+8653159511234", "+8613800000001", "+8653159511234",
"+8613800000002");
        if (status.indexOf(success) != -1) {
            System.out.println(status);
            System.out.println(resultcode + " " + resultdesc);
            System.out.println("The session id is: " + sessionId);
        }

        // TODO 需要接收状态和话单时,请参考"呼叫状态和话单通知API"接口实现状态通知和话单的接收和解析.
        // HostingVoiceEventDemoImpl

        // TODO 需要下载录音文件时,请参照"获取录音文件下载地址API"接口获取录音文件下载地址.
        String aksk = StringUtil.buildAKSKHeader(Constant.CLICK2CALL_APPID, Constant.CLICK2CALL_SECRET);
        String code = recordLinkAPI.getRecordLinkAPI("1200_366_0_20161228102743.wav",
"ostor.huawei.com", aksk);
        if (code.indexOf("301") != -1) {
            System.out.println("The record file download link is: " + recordLinkAPI.getLocation());
        } else {
            System.out.println("code: " + code);
            System.out.println("Failed: " + recordLinkAPI.getResponsebody().toString());
        }
    }

    /*
     * 前端需要发起语音回呼时,调用此方法 该示例只仅体现必选参数,可选参数根据接口文档和实际情况配置.
     */
    public static void doVoiceCall(String displayNbr, String callerNbr, String displayCalleeNbr, String
calleeNbr)
        throws Exception {

        Boolean retry = false;
        // 调用语音回呼接口,直至成功
        do {
            status = voiceCallAPI.voiceCallAPI(displayNbr, callerNbr, displayCalleeNbr, calleeNbr);
            if (status.indexOf(success) != -1) {
                retry = false;
```

```
            // 调用成功,记录返回的信息.
            resultcode = voiceCallAPI.getResponsePara("resultcode");
            resultdesc = voiceCallAPI.getResponsePara("resultdesc");
            sessionId = voiceCallAPI.getResponsePara("sessionId");
        } else {
            retry = true;
            // 调用失败,获取错误码和错误描述.
            resultcode = voiceCallAPI.getResponsePara("resultcode");
            resultdesc = voiceCallAPI.getResponsePara("resultdesc");
            // 处理错误
            VoiceCallMain.processError();
        }
    } while (retry);
  }

  // 当API的返回值不是200时,处理错误.
  private static void processError() throws InterruptedException {

    // TODO 根据错误码和错误码描述处理问题
    // 以下代码仅供调试使用,实际开发时请删除
    System.out.println(status);
    System.out.println(resultcode + " " + resultdesc);
    System.exit(-1);
  }
}

class VoiceCall {
  // 语音回呼API的URL
  private String urlVoiceCall;

  // 接口响应的消息体
  private Map<String, String> Responsebody;

  // Https实体
  private HttpsUtil httpsUtil;

  public VoiceCall() {
    // 商用地址
    urlVoiceCall = Constant.VOICE_CALL_COMERCIAL;
    Responsebody = new HashMap<>();
  }

  @SuppressWarnings("unchecked")
  /*
   * 该示例只仅体现必选参数,可选参数根据接口文档和实际情况配置. 该示例不体现参数校验,请根据各参数的格式要求自行实现校验功能.
   */
  public String voiceCallAPI(String displayNbr, String callerNbr, String displayCalleeNbr, String calleeNbr)
      throws Exception {

    httpsUtil = new HttpsUtil();

    // 忽略证书信任问题
    httpsUtil.trustAllHttpsCertificates();
    HttpsURLConnection.setDefaultHostnameVerifier(httpsUtil.hv);

    // 请求Headers
    Map<String, String> headerMap = new HashMap<>();
    headerMap.put(Constant.HEADER_APP_AUTH, Constant.AUTH_HEADER_VALUE);
    headerMap.put(Constant.HEADER_APP_AKSK,
        StringUtil.buildAKSKHeader(Constant.CLICK2CALL_APPID, Constant.CLICK2CALL_SECRET));

    // 构造消息体
    Map<String, String> bodys = new HashMap<String, String>();
    bodys.put("displayNbr", displayNbr);//主叫用户手机终端的来电显示号码。
    bodys.put("callerNbr", callerNbr);//发起呼叫时所使用的主叫号码。
    bodys.put("displayCalleeNbr", displayCalleeNbr);//被叫用户终端的来电显示号码。
    bodys.put("calleeNbr", calleeNbr);//发起呼叫时所拨打的被叫号码。
    String jsonRequest = JsonUtil.jsonObj2Sting(bodys);
```

```
        /*
         * Content-Type为application/json且请求方法为post时, 使用doPostJsonGetStatusLine方法构造http
         * request并获取响应.
         */
        StreamClosedHttpResponse responseVoiceCall = httpsUtil.doPostJsonGetStatusLine(urlVoiceCall,
headerMap,
            jsonRequest);

        // 响应的消息体写入Responsebody.
        Responsebody = JsonUtil.jsonString2SimpleObj(responseVoiceCall.getContent(),
Responsebody.getClass());

        // 返回响应的status.
        return responseVoiceCall.getStatusLine().toString();
    }

    // 获取整个响应消息体
    public Map<String, String> getResponsebody() {
        return this.Responsebody;
    }

    // 获取响应消息体中的单个参数
    public String getResponsePara(String ParaName) {
        return this.Responsebody.get(ParaName);
    }
}
```

“获取录音文件下载地址 API”代码样例

```
package com.huawei.demo;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.net.ssl.HttpsURLConnection;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;

import com.huawei.utils.Constant;
import com.huawei.utils.HttpsUtil;
import com.huawei.utils.JsonUtil;
import com.huawei.utils.StreamClosedHttpResponse;

public class GetRecordLink {
    // 获取录音文件下载地址API的URL
    private String urlRecordFile;
    // Https实体
    private HttpsUtil httpsUtil;
    // 接口响应的消息体
    private Map<String, String> Responsebody;
    // 录音文件下载地址
    private String Location;

    public GetRecordLink() {
        // 商用地址
        urlRecordFile = Constant.VOICE_FILE_DOWNLOAD;

        Responsebody = new HashMap<>();
    }

    @SuppressWarnings("unchecked")
    public String getRecordLinkAPI(String fileName, String recordDomain, String aksk) throws Exception {

        httpsUtil = new HttpsUtil();
```

```
        // 忽略证书信任问题
        httpsUtil.trustAllHttpsCertificates();
        HttpsURLConnection.setDefaultHostnameVerifier(httpsUtil.hv);

        // 构造URL
        List<NameValuePair> keyValues = new ArrayList<NameValuePair>();
        keyValues.add(new BasicNameValuePair("fileName", fileName));
        keyValues.add(new BasicNameValuePair("recordDomain", recordDomain));

        // 请求Headers
        Map<String, String> headerMap = new HashMap<>();
        headerMap.put(Constant.HEADER_APP_AUTH, Constant.AUTH_HEADER_VALUE);
        headerMap.put(Constant.HEADER_APP_AKSK, aksk);

        /*
         * Content-Type为application/json且请求方法为get时, 使用doGetWithParasGetStatusLine方法构造http
         * request并获取响应.
         */
        StreamClosedHttpResponse respRecordLink = httpsUtil.doGetWithParasGetStatusLine(urlRecordFile,
keyValues,
                headerMap);

        // 响应的消息体写入Responsebody.
        Responsebody = JsonUtil.jsonString2SimpleObj(respRecordLink.getContent(), Responsebody.getClass());

        // 从响应头域中获取location.
        String code = respRecordLink.getStatusLine().toString();
        if (code.indexOf("301") != -1) {
            Location = respRecordLink.getFirstHeader("Location").getValue();
        } else {
            Location = "";
        }

        // 返回响应的status.
        return code;
    }

    // 获取整个响应消息体
    public Map<String, String> getResponsebody() {
        return this.Responsebody;
    }

    // 获取响应消息体中的单个参数
    public String getResponsePara(String ParaName) {
        return this.Responsebody.get(ParaName);
    }

    // 获取location
    public String getLocation() {
        return this.Location;
    }
}
```

“呼叫状态通知 API”与“话单通知 API”代码样例

```
package com.huawei.demo;

import org.apache.log4j.Logger;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;

/**
 * 呼叫事件通知/话单通知
 * 客户平台收到RTC业务平台的呼叫事件通知/话单通知的接口通知
 */
public class HostingVoiceEventDemoImpl {
    private static Logger logger = Logger.getLogger(HostingVoiceEventDemoImpl.class);
```

```
/**
 * 呼叫事件 for 语音回呼/语音通知/语音验证码
 *
 * @param jsonBody
 * @breif 详细内容以接口文档为准
 */
public static void onCallEvent(String jsonBody) {
    // 封装JOSN请求
    JSONObject json = JSON.parseObject(jsonBody);
    String eventType = json.getString("eventType"); // 通知事件类型

    if ("fee".equalsIgnoreCase(eventType)) {
        logger.info("EventType error: " + eventType);
        return;
    }

    if (!(json.containsKey("statusInfo"))) {
        logger.info("param error: no statusInfo.");
        return;
    }
    JSONObject statusInfo = json.getJSONObject("statusInfo"); // 呼叫状态事件信息

    logger.info("eventType: " + eventType); // 打印通知事件类型

    //callout：呼出事件
    if ("callout".equalsIgnoreCase(eventType)) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.containsKey("sessionId")) {
            logger.info("sessionId: " + statusInfo.getString("sessionId"));
        }
        return;
    }
    //alerting：振铃事件
    if ("alerting".equalsIgnoreCase(eventType)) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.containsKey("sessionId")) {
            logger.info("sessionId: " + statusInfo.getString("sessionId"));
        }
        return;
    }
    //answer：应答事件
    if ("answer".equalsIgnoreCase(eventType)) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.containsKey("sessionId")) {
```

```
            logger.info("sessionId: " + statusInfo.getString("sessionId"));
        }
        return;
    }
    //collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ("collectInfo".equalsIgnoreCase(eventType)) {
        /**
         * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'sessionId': 通话链路的标识ID
         * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
         */
        if (statusInfo.containsKey("digitInfo")) {
            logger.info("digitInfo: " + statusInfo.getString("digitInfo"));
        }
        return;
    }
    //disconnect：挂机事件
    if ("disconnect".equalsIgnoreCase(eventType)) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
         * 'stateCode': 通话挂机的原因值
         * 'stateDesc': 通话挂机的原因值的描述
         */
        if (statusInfo.containsKey("sessionId")) {
            logger.info("sessionId: " + statusInfo.getString("sessionId"));
        }
        return;
    }
}

/**
 * 话单通知 for 语音回呼/语音通知/语音验证码
 *
 * @param jsonBody
 * @breif 详细内容以接口文档为准
 */
public static void onFeeEvent(String jsonBody) {
    // 封装JSON请求
    JSONObject json = JSON.parseObject(jsonBody);
    String eventType = json.getString("eventType"); // 通知事件类型

    if (!("fee".equalsIgnoreCase(eventType))) {
        logger.info("EventType error: " + eventType);
        return;
    }

    if (!(json.containsKey("feeLst"))) {
        logger.info("param error: no feeLst.");
        return;
    }
    JSONArray feeLst = json.getJSONArray("feeLst"); // 呼叫话单事件信息
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'direction': 通话的呼叫方向,以RTC业务平台为基准
     * 'spId': 客户的云服务账号
     * 'appKey': 应用的app_key
     * 'icid': 呼叫记录的唯一标识
     * 'bindNum': 发起此次呼叫的CallEnabler业务号码
     * 'sessionId': 通话链路的唯一标识
```

```
        * 'callerNum': 主叫号码
        * 'calleeNum': 被叫号码
        * 'fwdDisplayNum': 转接呼叫时的显示号码(仅语音回呼场景携带)
        * 'fwdDstNum': 转接呼叫时的转接号码(仅语音回呼场景携带)
        * 'fwdStartTime': 转接呼叫操作的开始时间(仅语音回呼场景携带)
        * 'fwdAlertingTime': 转接呼叫操作后的振铃时间(仅语音回呼场景携带)
        * 'fwdAnswerTime': 转接呼叫操作后的应答时间(仅语音回呼场景携带)
        * 'callEndTime': 呼叫结束时间
        * 'fwdUnaswRsn': 转接呼叫操作失败的Q850原因值
        * 'failTime': 呼入,呼出的失败时间
        * 'ulFailReason': 通话失败的拆线点
        * 'sipStatusCode': 呼入,呼出的失败SIP状态码
        * 'callOutStartTime': Initcall的呼出开始时间
        * 'callOutAlertingTime': Initcall的呼出振铃时间
        * 'callOutAnswerTime': Initcall的呼出应答时间
        * 'callOutUnaswRsn': Initcall的呼出失败的Q850原因值
        * 'dynIVRStartTime': 自定义动态IVR开始时间(仅语音通知场景携带)
        * 'dynIVRPath': 自定义动态IVR按键路径(仅语音通知场景携带)
        * 'recordFlag': 录音标识
        * 'recordStartTime': 录音开始时间(仅语音回呼场景携带)
        * 'recordObjectName': 录音文件名(仅语音回呼场景携带)
        * 'recordBucketName': 录音文件所在的目录名(仅语音回呼场景携带)
        * 'recordDomain': 存放录音文件的域名(仅语音回呼场景携带)
        * 'recordFileDownloadUrl': 录音文件下载地址(仅语音回呼场景携带)
        * 'ttsPlayTimes': 应用TTS功能时,使用TTS的总次数
        * 'ttsTransDuration': 应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
        * 'serviceType': 携带呼叫的业务类型信息
        * 'hostName': 话单生成的服务器设备对应的主机名
        * 'userData': 用户附属信息
        */
       //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
       if (feeLst.size() > 1) {
           for (Object loop : feeLst) {
               if (((JSONObject)loop).containsKey("sessionId")) {
                   logger.info("sessionId: " + ((JSONObject)loop).getString("sessionId"));
               }
           }
       } else if (feeLst.size() == 1) {
           if (feeLst.getJSONObject(0).containsKey("sessionId")) {
               logger.info("sessionId: " + feeLst.getJSONObject(0).getString("sessionId"));
           }
       } else {
           logger.info("feeLst error: no element.");
       }
   }
}
```

4.1.3 Python

样例	**语音回呼场景API**、**获取录音文件下载地址API**、**呼叫状态通知API**、**话单通知API**
环境要求	Python 3.0及以上版本。
引用库	requests 2.18.1 1. 请自行下载安装Python 3.x，并完成环境配置。 2. 打开命令行窗口，执行pip install requests命令。 3. 执行pip list查看安装结果。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音回呼场景 API”代码样例

```
# -*- coding: utf-8 -*-
import base64
import hashlib
import requests #需要先使用pip install requests命令安装依赖
import time
import uuid
import hmac

from hashlib import sha256

#必填,请参考"开发准备-申请资源"获取如下数据,替换为实际值
base_url = 'https://{domain}:{port}'#APP接入地址,购买服务时下发,请替换为实际值
appKey = '***appKey***'#语音回呼应用的appKey,购买服务时下发,请替换为实际值
appSecret = '***appSecret***'#语音回呼应用的appSecret,购买服务时下发,请替换为实际值

def buildAKSKHeader(appKey, appSecret):
    now = time.strftime('%Y-%m-%dT%H:%M:%SZ') #Created
    nonce = str(uuid.uuid4()).replace('-','') #Nonce
    digist = hmac.new(appSecret.encode(), (nonce + now).encode(), digestmod=sha256).digest()
    digestBase64 = base64.b64encode(digist).decode() #PasswordDigest
    return 'UsernameToken Username="{}",PasswordDigest="{}",Nonce="{}",Created="{}"'.format(appKey,
digestBase64, nonce, now);

def voiceCallAPI(displayNbr, callerNbr, displayCalleeNbr, calleeNbr):
    if len(displayNbr) < 1 or len(callerNbr) < 1 or len(displayCalleeNbr) < 1 or len(calleeNbr) < 1:
        return

    apiUri = '/rest/httpsessions/click2Call/v2.0'
    requestUrl = base_url + apiUri

    header = {
        'Content-Type': 'application/json;charset=UTF-8',
        'Authorization': 'AKSK realm="SDP",profile="UsernameToken",type="Appkey"',
        'X-AKSK': buildAKSKHeader(appKey, appSecret)
    }

    jsonData = {
        # 必填参数
        'displayNbr': displayNbr,#主叫用户手机终端的来电显示号码。
        'callerNbr': callerNbr,#发起呼叫时所使用的主叫号码。
        'displayCalleeNbr': displayCalleeNbr,#被叫用户终端的来电显示号码。
        'calleeNbr': calleeNbr#发起呼叫时所拨打的被叫号码。
        # 选填参数
#         'maxDuration': 0, #允许单次通话进行的最长时间
#         'lastMinVoice': 'lastmin_voice1.wav', #最后一分钟放音提示音
#         'lastMinToUE': 'both', #最后一分钟放音的播放对象
#         'playPreVoice': 'false', #设置主叫（callerNbr）应答语音回呼后,呼叫被叫（calleeNbr）前,是否向主叫
（callerNbr）播放提示音
#         'preVoice': 'pre_voice1.wav', #设置主叫（callerNbr）应答语音回呼后,呼叫被叫（calleeNbr）前向主叫播
放的提示音
#         'waitVoice': 'wait_voice1.wav', #设置主叫应答语音回呼后的等待音
#         'calleeMedia': 'all' , #指定被叫的媒体音播放方式
#         'statusUrl': '', #设置SP接收状态上报的URL,要求使用BASE64编码
#         'feeUrl': '', #设置SP接收话单上报的URL,要求使用BASE64编码
#         'recordFlag': 'false', #设置语音回呼通话过程是否录音
#         'recordHintTone': 'recordhint_voice1.wav', #设置使用录音功能的提示音
```

```
#         'partyTypeRequiredInDisconnect': 'false', #disconnect状态是否需要携带通话主动挂机的用户类型
#         'returnIdlePort': 'false', #指示是否需要返回平台空闲呼叫端口数量
#         'userData': 'customerId123' #设置用户的附属信息
    }

    try:
        r = requests.post(requestUrl, json=jsonData, headers=header, verify=False)
        print(r.text) #打印响应结果
    except requests.exceptions.HTTPError as e:
        print(e.code)
        print(e.read().decode('utf-8')) #打印错误信息
    pass

if __name__ == '__main__':
    voiceCallAPI('+86531*******4', '+86135*******1', '+86531*******4', '+86135*******2')
```

“获取录音文件下载地址 API”代码样例

```
# -*- coding: utf-8 -*-
import base64
import hashlib
import requests #需要先使用pip install requests命令安装依赖
import time
import urllib
import uuid
import hmac

from hashlib import sha256

#必填,请参考"开发准备-申请资源"获取如下数据,替换为实际值
base_url = 'https://{domain}:{port}'#APP接入地址,购买服务时下发，请替换为实际值

def buildAKSKHeader(appKey, appSecret):
    now = time.strftime('%Y-%m-%dT%H:%M:%SZ') #Created
    nonce = str(uuid.uuid4()).replace('-','') #Nonce
    digist = hmac.new(appSecret.encode(), (nonce + now).encode(), digestmod=sha256).digest()
    digestBase64 = base64.b64encode(digist).decode() #PasswordDigest
    return 'UsernameToken Username="{}",PasswordDigest="{}",Nonce="{}",Created="{}"'.format(appKey,
digestBase64, nonce, now);

def getRecordLinkAPI(fileName, recordDomain, xaksk):
    if len(fileName) < 1 or len(recordDomain) < 1 or len(xaksk) < 1:
        return

    location = ''

    apiUri = '/rest/provision/voice/record/v1.0'
    queryParams = urllib.parse.urlencode({'fileName':fileName, 'recordDomain':recordDomain})
    requestUrl = base_url + apiUri + '?' + queryParams

    header = {
        'Content-Type': 'application/json;charset=UTF-8',
        'Authorization': 'AKSK realm="SDP",profile="UsernameToken",type="Appkey"',
        'X-AKSK': xaksk
    }

    try:
        r = requests.get(requestUrl, headers=header, allow_redirects=False, verify=False) #发送请求
        if(301 == r.status_code):
            print(r.status_code) #打印响应结果
            location = r.headers['Location']
        else:
            print(r.status_code)
            print(r.text)#打印响应结果
    except requests.exceptions.HTTPError as e:
        print(e.code)
        print(e.read().decode('utf-8')) #打印错误信息

    return location
```

```
if __name__ == '__main__':
    aksk = buildAKSKHeader('***appKey***', '***appSecret***') #语音回呼应用的appKey和appSecret
    location = getRecordLinkAPI('1200_366_0_20161228102743.wav', 'ostor.huawei.com', aksk)
    print('The record file download link is: ' + location)
```

“呼叫状态通知 API”代码样例

```
# -*- coding: utf-8 -*-
'''
呼叫事件通知
客户平台收到语音通话平台的呼叫事件通知的接口通知
'''
import json

#呼叫事件通知样例
jsonBody = json.dumps({
    'eventType': 'callout',
    'statusInfo': {
        'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp': '2019-01-24 03:04:24',
        'caller': '+86138*******2',
        'called': '+86138*******1',
        'userData': 'customerId123'
    }
}).encode('ascii')

print(jsonBody)

'''
呼叫事件通知
@see: 详细内容以接口文档为准
@param param: jsonBody
@return:
'''
def onCallEvent(jsonBody):
    jsonObj = json.loads(jsonBody) #将通知消息解析为jsonObj
    eventType = jsonObj['eventType'] #通知事件类型

    if ('fee' == eventType):
        print('EventType error: ' + eventType)
        return

    if ('statusInfo' not in jsonObj):
        print('param error: no statusInfo.')
        return
    statusInfo = jsonObj['statusInfo'] #呼叫状态事件信息

    print('eventType: ' + eventType) #打印通知事件类型

    #callout：呼出事件
    if ('callout' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #alerting：振铃事件
    if ('alerting' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
```

```
        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #answer：应答事件
    if ('answer' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ('collectInfo' == eventType):
        '''
        Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'sessionId': 通话链路的标识ID
        'digitInfo': 放音收号场景中,语音通话平台对开发者进行放音收号操作的结果描述
        '''
        if ('digitInfo' in statusInfo):
            print('digitInfo: ' + statusInfo['digitInfo'])
        return
    #disconnect：挂机事件
    if ('disconnect' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        'partyType': 挂机的用户类型,仅在语音回呼场景携带
        'stateCode': 通话挂机的原因值
        'stateDesc': 通话挂机的原因值的描述
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return

if __name__ == '__main__':
    onCallEvent(jsonBody) #呼叫事件处理
```

“话单通知 API”代码样例

```
# -*- coding: utf-8 -*-
'''
话单通知
客户平台收到语音通话平台的话单通知的接口通知
'''
import json

#话单通知样例
jsonBody = json.dumps({
    'eventType': 'fee',
    'feeLst': [
```

```
        {
            'direction': 0, #通话的呼叫方向,以语音通话平台为基准
            'spId': 'CaaS_Test_01', #客户的云服务账号
            'appKey': 'ka4k******9YnEm2', #应用的app_key
            'icid': 'CAE-20190124110424-12019775', #呼叫记录的唯一标识
            'bindNum': '+86755*******8', #发起此次呼叫的CallEnabler业务号码,即绑定号码
            'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com', #通话链路的
唯一标识
            'callerNum': '+86138*******2', #主叫号码
            'calleeNum': '+86138*******1', #被叫号码
            'fwdDisplayNum': '+86138*******2', #转接呼叫时的显示号码(仅语音回呼场景携带)
            'fwdDstNum': '+86138*******1', #转接呼叫时的转接号码(仅语音回呼场景携带)

            'fwdStartTime': '2019-01-24 03:04:31', #转接呼叫操作的开始时间(仅语音回呼场景携带)
            'fwdAlertingTime': '2019-01-24 03:04:36', #转接呼叫操作后的振铃时间(仅语音回呼场景携带)
            'fwdAnswerTime': '2019-01-24 03:04:38', #转接呼叫操作后的应答时间(仅语音回呼场景携带)
            'callEndTime': '2019-01-24 03:04:49', #呼叫结束时间
            'fwdUnaswRsn': 0, #转接呼叫操作失败的Q850原因值
            'failTime': '', #呼入,呼出的失败时间
            'ulFailReason': 0, #通话失败的拆线点
            'sipStatusCode': 0, #呼入,呼出的失败SIP状态码
            'callOutStartTime': '2019-01-24 03:04:24', #Initcall的呼出开始时间
            'callOutAlertingTime': '2019-01-24 03:04:27', #Initcall的呼出振铃时间
            'callOutAnswerTime': '2019-01-24 03:04:31', #Initcall的呼出应答时间
            'callOutUnaswRsn': 0, #Initcall的呼出失败的Q850原因值
            'dynIVRStartTime': '', #自定义动态IVR开始时间(仅语音通知场景携带)
            'dynIVRPath': '', #自定义动态IVR按键路径(仅语音通知场景携带)
            'recordFlag': 0, #录音标识
            'recordStartTime': '', #录音开始时间(仅语音回呼场景携带)
            'recordObjectName': '', #录音文件名(仅语音回呼场景携带)
            'recordBucketName': '', #录音文件所在的目录名(仅语音回呼场景携带)
            'recordDomain': '', #存放录音文件的域名(仅语音回呼场景携带)
            'recordFileDownloadUrl': '', #录音文件下载地址(仅语音回呼场景携带)
            'ttsPlayTimes': 0, #应用TTS功能时,使用TTS的总次数
            'ttsTransDuration': 0, #应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
            'serviceType': '002', #携带呼叫的业务类型信息
            'hostName': 'callenabler245.huaweicaas.com', #话单生成的服务器设备对应的主机名
            'userData': '' #用户附属信息
        }
    ]
}).encode('ascii')

print(jsonBody)

'''
话单通知
@see: 详细内容以接口文档为准
@param param: jsonBody
@return:
'''
def onFeeEvent(jsonBody):
    jsonObj = json.loads(jsonBody) #将通知消息解析为jsonObj
    eventType = jsonObj['eventType'] #通知事件类型

    if ('fee' != eventType):
        print('EventType error: ' + eventType)
        return

    if ('feeLst' not in jsonObj):
        print('param error: no feeLst.');
        return

    feeLst = jsonObj['feeLst']

    #Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
    #短时间内有多个通话结束时语音通话平台会将话单合并推送,每条消息最多携带50个话单
    if (len(feeLst) > 1):
        for loop in feeLst:
            if ('sessionId' in loop):
```



```
            print('sessionId: ' + loop['sessionId'])
    elif(len(feeLst) == 1):
        if ('sessionId' in feeLst[0]):
            print('sessionId: ' + feeLst[0]['sessionId'])
    else:
        print('feeLst error: no element.');

if __name__ == '__main__':
    onFeeEvent(jsonBody) #话单处理
```

4.1.4 PHP

样例	语音回呼场景API、获取录音文件下载地址API、呼叫状态通知API、话单通知API
环境要求	PHP 7.0及以上版本。
引用库	-

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音回呼场景 API”代码样例

```
<?php
include 'data.php';
include 'util.php';
include 'getRecordLink.php';
/**
 * voiceCallAPI
 * @param string $displayNbr
 * @param string $callerNbr
 * @param string $displayCalleeNbr
 * @param string $calleeNbr
 */
function voiceCallAPI($displayNbr, $callerNbr, $displayCalleeNbr, $calleeNbr) {
    if (empty($displayNbr) || empty($callerNbr) || empty($displayCalleeNbr) || empty($calleeNbr)) {
        return;
    }

    $method = 'POST';
    $apiUri = '/rest/httpsessions/click2Call/v2.0';
    $xaksk = buildAKSKHeader(getClick2Call_AppId(), getClick2Call_Secret());

    $content = json_encode([
        /* 必填参数*/
        'displayNbr' => $displayNbr,//主叫用户手机终端的来电显示号码。
        'callerNbr' => $callerNbr,//发起呼叫时所使用的主叫号码。
        'displayCalleeNbr' => $displayCalleeNbr,//被叫用户终端的来电显示号码。
        'calleeNbr' => $calleeNbr//发起呼叫时所拨打的被叫号码。
        /* 选填参数*/
//        'bindNbr' => '+86123456789', //CallEnabler业务号码,即绑定号码
//        'maxDuration' => 0, //允许单次通话进行的最长时间
//        'lastMinVoice' => 'lastmin_voice1.wav', //最后一分钟放音提示音
//        'lastMinToUE' => 'both', //最后一分钟放音的播放对象
```

```
//         'playPreVoice' => 'false', //设置主叫（callerNbr）应答语音回呼后,呼叫被叫（calleeNbr）前,是否向主叫（callerNbr）播放提示音
//         'preVoice' => 'pre_voice1.wav', //设置主叫（callerNbr）应答语音回呼后,呼叫被叫（calleeNbr）前向主叫播放的提示音
//         'waitVoice' => 'wait_voice1.wav', //设置主叫应答语音回呼后的等待音
//         'calleeMedia' => 'all', //指定被叫的媒体音播放方式
//         'statusUrl' => '', //设置SP接收状态上报的URL,要求使用BASE64编码
//         'feeUrl' => '', //设置SP接收话单上报的URL,要求使用BASE64编码
//         'recordFlag' => 'false', //设置语音回呼通话过程是否录音
//         'recordHintTone' => 'recordhint_voice1.wav', //设置使用录音功能的提示音
//         'partyTypeRequiredInDisconnect' => 'false', //disconnect状态是否需要携带通话主动挂机的用户类型
//         'returnIdlePort' => 'false', //指示是否需要返回平台空闲呼叫端口数量
//         'userData' => 'customerId123' //设置用户的附属信息
    ]);

    $requestUrl = getBaseUrl() . $apiUri;
    $context_options = createOptions($method, $xaksk, $content);

    try {
        $response = file_get_contents($requestUrl, false, stream_context_create($context_options)); // 发送请求
        print_r($response . PHP_EOL); // 打印响应结果
    } catch (Exception $e) {
        echo $e->getMessage(); //打印错误信息
    }
}
voiceCallAPI('+8653159511234', '+8613500000001', '+8653159511234', '+8613500000002');
$xaksk = buildAKSKHeader(getClick2Call_AppId(), getClick2Call_Secret());
$location = getRecordLinkAPI('1200_366_0_20161228102743.wav', 'ostor.huawei.com', $xaksk);
print_r('The record file download link is: ' . $location . PHP_EOL);
?>
```

“获取录音文件下载地址 API”代码样例

```
<?php
include 'data.php';
include 'util.php';
/**
 * getRecordLinkAPI
 * @param string $fileName
 * @param string $recordDomain
 * @param string $xaksk
 * @return void|string|mixed
 */
function getRecordLinkAPI($fileName, $recordDomain, $xaksk) {
    if (empty($fileName) || empty($recordDomain) || empty($xaksk)) {
        return;
    }

    $location = '';

    $method = 'GET';
    $apiUri = '/rest/provision/voice/record/v1.0';
    $queryParams = http_build_query(['fileName' => $fileName, 'recordDomain' => $recordDomain]);

    $requestUrl = getBaseUrl() . $apiUri . '?' . $queryParams;
    $context_options = createOptions($method, $xaksk, null);

    try {
        $http_response_header = get_headers($requestUrl, 1, stream_context_create($context_options)); //获取响应消息头域信息
        if(strpos($http_response_header[0], '301')){
            $location = $http_response_header['Location']; //获取录音文件下载地址
        }else{
            print_r($http_response_header[0] . PHP_EOL); //打印响应码400/401/403/500
//             $response = file_get_contents($requestUrl, false, stream_context_create($context_options)); //获取响应消息数据信息
//             print_r($response . PHP_EOL); // 打印响应结果
        }
    } catch (Exception $e) {
```

```
        echo $e->getMessage(); //打印错误信息
    }

    return $location;
}
?>
```

“呼叫状态通知 API”代码样例

```
<?php
/**
 * 呼叫事件通知
 * 客户平台收到RTC业务平台的呼叫事件通知的接口通知
 */
//呼叫事件通知样例
$jsonBody = json_encode([
    'eventType' => 'callout',
    'statusInfo' => [
        'sessionId' => '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp' => '2019-01-24 03:04:24',
        'caller' => '+8613800000022',
        'called' => '+8613800000021'
    ]
]);
print_r($jsonBody . PHP_EOL);
onCallEvent($jsonBody);//呼叫事件处理
/**
 * 呼叫事件通知
 * @desc 详细内容以接口文档为准
 * @param jsonBody
 */
function onCallEvent($jsonBody) {
    $jsonArr = json_decode($jsonBody, true); //将通知消息解析为关联数组
    $eventType = $jsonArr['eventType']; //通知事件类型

    if (strcasecmp($eventType, 'fee') == 0) {
        print_r('EventType error: ' . $eventType);
        return;
    }

    if (!array_key_exists('statusInfo', $jsonArr)) {
        print_r('param error: no statusInfo.');
        return;
    }
    $statusInfo = $jsonArr['statusInfo']; //呼叫状态事件信息

    print_r('eventType: ' . $eventType . PHP_EOL); //打印通知事件类型
    //callout：呼出事件
    if (strcasecmp($eventType, 'callout') == 0) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (array_key_exists('sessionId', $statusInfo)) {
            print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
        }
        return;
    }
    //alerting：振铃事件
    if (strcasecmp($eventType, 'alerting') == 0) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
```

```
        * 'userData': 用户附属信息
        * 'sessionId': 通话链路的标识ID
        * 'caller': 主叫号码
        * 'called': 被叫号码
        */
        if (array_key_exists('sessionId', $statusInfo)) {
            print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
        }
        return;
    }
    //answer：应答事件
    if (strcasecmp($eventType, 'answer') == 0) {
        /**
        * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
        *
        * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
        * 'userData': 用户附属信息
        * 'sessionId': 通话链路的标识ID
        * 'caller': 主叫号码
        * 'called': 被叫号码
        */
        if (array_key_exists('sessionId', $statusInfo)) {
            print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
        }
        return;
    }
    //collectInfo：放音收号结果事件,仅应用于语音通知场景
    if (strcasecmp($eventType, 'collectInfo') == 0) {
        /**
        * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
        *
        * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
        * 'sessionId': 通话链路的标识ID
        * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
        */
        if (array_key_exists('digitInfo', $statusInfo)) {
            print_r('digitInfo: ' . $statusInfo['digitInfo'] . PHP_EOL);
        }
        return;
    }
    //disconnect：挂机事件
    if (strcasecmp($eventType, 'disconnect') == 0) {
        /**
        * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
        *
        * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
        * 'userData': 用户附属信息
        * 'sessionId': 通话链路的标识ID
        * 'caller': 主叫号码
        * 'called': 被叫号码
        * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
        * 'stateCode': 通话挂机的原因值
        * 'stateDesc': 通话挂机的原因值的描述
        */
        if (array_key_exists('sessionId', $statusInfo)) {
            print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
        }
        return;
    }
}
?>
```

“话单通知 API”代码样例

```
<?php
/**
 * 话单通知
 * 客户平台收到RTC业务平台的话单通知的接口通知
 */
```

```
//话单通知样例
$jsonBody = json_encode([
    'eventType' => 'fee',
    'feeLst' => [
        [
            'direction' => 0, //通话的呼叫方向,以RTC业务平台为基准
            'spId' => 'CaaS_Test_01', //客户的云服务账号
            'appKey' => 'ka4kESI5s3YyurL1wpx63s9YnEm2', //应用的app_key
            'icid' => 'CAE-20190124110424-12019775', //呼叫记录的唯一标识
            'bindNum' => '+8675512345678', //发起此次呼叫的CallEnabler业务号码,即绑定号码
            'sessionId' => '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com', //通话链
路的唯一标识
            'callerNum' => '+8613800000022', //主叫号码
            'calleeNum' => '+8613800000021', //被叫号码
            'fwdDisplayNum' => '+8613800000022', //转接呼叫时的显示号码(仅语音回呼场景携带)
            'fwdDstNum' => '+8613866887021', //转接呼叫时的转接号码(仅语音回呼场景携带)
            'fwdStartTime' => '2019-01-24 03:04:31', //转接呼叫操作的开始时间(仅语音回呼场景携带)
            'fwdAlertingTime' => '2019-01-24 03:04:36', //转接呼叫操作后的振铃时间(仅语音回呼场景携带)
            'fwdAnswerTime' => '2019-01-24 03:04:38', //转接呼叫操作后的应答时间(仅语音回呼场景携带)
            'callEndTime' => '2019-01-24 03:04:49', //呼叫结束时间
            'fwdUnaswRsn' => 0, //转接呼叫操作失败的Q850原因值
            'failTime' => '', //呼入,呼出的失败时间
            'ulFailReason' => 0, //通话失败的拆线点
            'sipStatusCode' => 0, //呼入,呼出的失败SIP状态码
            'callOutStartTime' => '2019-01-24 03:04:24', //Initcall的呼出开始时间
            'callOutAlertingTime' => '2019-01-24 03:04:27', //Initcall的呼出振铃时间
            'callOutAnswerTime' => '2019-01-24 03:04:31', //Initcall的呼出应答时间
            'callOutUnaswRsn' => 0, //Initcall的呼出失败的Q850原因值
            'dynIVRStartTime' => '', #自定义动态IVR开始时间(仅语音通知场景携带)
            'dynIVRPath' => '', //自定义动态IVR按键路径(仅语音通知场景携带)
            'recordFlag' => 0, //录音标识
            'recordStartTime' => '', //录音开始时间(仅语音回呼场景携带)
            'recordObjectName' => '', //录音文件名(仅语音回呼场景携带)
            'recordBucketName' => '', //录音文件所在的目录名(仅语音回呼场景携带)
            'recordDomain' => '', //存放录音文件的域名(仅语音回呼场景携带)
            'recordFileDownloadUrl' => '', //录音文件下载地址(仅语音回呼场景携带)
            'ttsPlayTimes' => 0, //应用TTS功能时,使用TTS的总次数
            'ttsTransDuration' => 0, //应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
            'serviceType' => '002', //携带呼叫的业务类型信息
            'hostName' => 'callenabler245.huaweicaas.com', //话单生成的服务器设备对应的主机名
            'userData' => '' //用户附属信息
        ]
    ]
]);
print_r($jsonBody . PHP_EOL);
onFeeEvent($jsonBody);//话单处理
/**
 * 话单通知
 * @desc 详细内容以接口文档为准
 * @param jsonBody
 */
function onFeeEvent($jsonBody) {
    $jsonArr = json_decode($jsonBody, true); //将通知消息解析为关联数组
    $eventType = $jsonArr['eventType']; //通知事件类型

    if (strcasecmp($eventType, 'fee') != 0) {
        print_r('EventType error: ' . $eventType);
        return;
    }

    if (!array_key_exists('feeLst', $jsonArr)) {
        print_r('param error: no feeLst.');
        return;
    }

    $feeLst = $jsonArr['feeLst']; //呼叫话单事件信息

    //Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
    //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
```

```
    if (sizeof($feeLst) > 1) {
        foreach ($feeLst as $loop){
            if (array_key_exists('sessionId', $loop)) {
                print_r('sessionId: ' . $loop['sessionId'] . PHP_EOL);
            }
        }
    } else if(sizeof($feeLst) == 1) {
        if (array_key_exists('sessionId', $feeLst[0])) {
            print_r('sessionId: ' . $feeLst[0]['sessionId'] . PHP_EOL);
        }
    } else {
        print_r('feeLst error: no element.');
    }
}
?>
```

4.1.5 C#

样例	语音回呼场景API、获取录音文件下载地址API、呼叫状态通知API、话单通知API
环境要求	.NET Core 2.0及以上版本或.NET Framework 4.7.1及以上版本。
引用库	Newtonsoft.Json 11.0.2，请参考https://www.newtonsoft.com/json获取。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音回呼场景 API”代码样例

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Security.Cryptography;
using System.Text;

namespace voicecall_csharp_demo_x_aksk_
{
    class VoiceCall
    {
        string base_url = "https://{domain}:{port}"; //APP接入地址,购买服务时下发,请替换为实际值
        string appKey = "***appKey***"; //语音回呼应用的appKey,购买服务时下发,请替换为实际值
        string appSecret = "***appSecret***"; //语音回呼应用的appSecret,购买服务时下发,请替换为实际值

        static void Main(string[] args)
        {
            //主叫号码,被叫号码请替换为实际号码.固话号码从控制台号码管理页获取
            voiceCallAPI("+86531******4", "+86135******1", "+86531******4", "+86135******2");
```

```
    }

    static void voiceCallAPI(string displayNbr, string callerNbr, string displayCalleeNbr, string calleeNbr)
    {
        if (String.IsNullOrEmpty(displayNbr) || String.IsNullOrEmpty(callerNbr) ||
String.IsNullOrEmpty(displayCalleeNbr) || String.IsNullOrEmpty(calleeNbr))
        {
            return;
        }

        string apiURI = "/rest/httpsessions/click2Call/v2.0"; //接口URI
        string requestUrl = base_url + apiURI;

        try
        {
            //为防止因HTTPS证书认证失败造成API调用失败,需要先忽略证书信任问题
            //.NET Framework 4.7.1及以上版本,请采用如下代码
            var sslHandler = new HttpClientHandler()
            {
                ServerCertificateCustomValidationCallback = (message, cert, chain, err) => { return true; }
            };
            HttpClient client = new HttpClient(sslHandler, true);
            //低于.NET Framework 4.7.1版本,请采用如下代码
            //HttpClient client = new HttpClient();
            //ServicePointManager.ServerCertificateValidationCallback = new
RemoteCertificateValidationCallback(CheckValidationResult);

            //请求Headers
            client.DefaultRequestHeaders.Add("Authorization", "AKSK realm=\"SDP\",profile=
\"UsernameToken\",type=\"Appkey\"");
            client.DefaultRequestHeaders.Add("X-AKSK", buildAKSKHeader(appKey, appSecret));

            //请求Body
            var body = new Dictionary<string, object>() {
                /*必填参数*/
                {"displayNbr", displayNbr},//主叫用户手机终端的来电显示号码。
                {"callerNbr", callerNbr},//发起呼叫时所使用的主叫号码。
                {"displayCalleeNbr", displayCalleeNbr},//被叫用户终端的来电显示号码。
                {"calleeNbr", calleeNbr}//发起呼叫时所拨打的被叫号码。
                /*选填参数*/

                //{"maxDuration", 0}, //允许单次通话进行的最长时间
                //{"lastMinVoice", "lastmin_voice1.wav"}, //最后一分钟放音提示音
                //{"lastMinToUE", "both"}, //最后一分钟放音的播放对象
                //{"playPreVoice", "false"}, //设置主叫（callerNbr）应答语音回呼后,呼叫被叫（calleeNbr）前,是
否向主叫（callerNbr）播放提示音
                //{"preVoice", "pre_voice1.wav"}, //设置主叫（callerNbr）应答语音回呼后,呼叫被叫
（calleeNbr）前向主叫播放的提示音
                //{"waitVoice", "wait_voice1.wav"}, //设置主叫应答语音回呼后的等待音

                //{"calleeMedia", "all"}, //指定被叫的媒体音播放方式
                //{"statusUrl", ""}, //设置SP接收状态上报的URL,要求使用BASE64编码
                //{"feeUrl", ""}, //设置SP接收话单上报的URL,要求使用BASE64编码
                //{"recordFlag", "false"}, //设置语音回呼通话过程是否录音
                //{"recordHintTone", "recordhint_voice1.wav"}, //设置使用录音功能的提示音
                //{"partyTypeRequiredInDisconnect", "false"}, //disconnect状态是否需要携带通话主动挂机的用
户类型
                //{"returnIdlePort", "false"}, //指示是否需要返回平台空闲呼叫端口数量
                //{"userData", "customerId123"} //设置用户的附属信息
            };

            HttpContent content = new StringContent(JsonConvert.SerializeObject(body));
            //请求Headers中的Content-Type参数
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            var response = client.PostAsync(requestUrl, content).Result;
            Console.WriteLine(response.StatusCode);
            var res = response.Content.ReadAsStringAsync().Result;
            Console.WriteLine(res); //打印响应结果
```

```
        }
        catch (Exception e)
        {
            Console.WriteLine(e.StackTrace);
            Console.WriteLine(e.Message); //打印错误信息
        }
    }

    static string buildAKSKHeader(string appKey, string appSecret)
    {
        string now = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ"); //Created
        string nonce = Guid.NewGuid().ToString().Replace("-", ""); //Nonce
        String str = nonce + now;

        byte[] keyByte = Encoding.UTF8.GetBytes(appSecret);
        byte[] messageBytes = Encoding.UTF8.GetBytes(str);
        using (var hmacsha256 = new HMACSHA256(keyByte))
        {
            byte[] hashmessage = hmacsha256.ComputeHash(messageBytes);
            string base64 = Convert.ToBase64String(hashmessage);
            return String.Format("UsernameToken Username=\"{0}\",PasswordDigest=\"{1}\",Nonce=
\"{2}\",Created=\"{3}\"",
                appKey, base64, nonce, now);
        }
    }

    //低于.NET Framework 4.7.1版本,启用如下方法
    //static bool CheckValidationResult(object sender, X509Certificate certificate, X509Chain chain,
SslPolicyErrors errors)
    //{
    //    return true;
    //}
  }
}
```

“获取录音文件下载地址 API”代码样例

```
using System;
using System.Collections.Specialized;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Security.Cryptography;
using System.Text;

namespace voicecall_csharp_demo_x_aksk_
{
  class GetRecordLink
  {
    string base_url = "https://{domain}:{port}"; //APP接入地址,购买服务时下发,请替换为实际值

    static void Main(string[] args)
    {
        //语音回呼应用的appKey和appSecret,购买服务时下发,请替换为实际值
        string aksk = buildAKSKHeader("***appKey***", "***appSecret***");
        //录音文件名和录音存储服务器域名,从话单通知中获取
        string location = getRecordLinkAPI("1200_366_0_20161228102743.wav", "ostor.huawei.com", aksk);
        //打印录音文件下载地址
        Console.WriteLine($"The record file download link is: {location}");
    }

    static string getRecordLinkAPI(string fileName, string recordDomain, string xaksk)
    {
        if (String.IsNullOrEmpty(fileName) || String.IsNullOrEmpty(recordDomain) ||
String.IsNullOrEmpty(xaksk))
        {
            return;
        }
```

```
        var location;

        string apiURI = "/rest/provision/voice/record/v1.0"; //接口URI

        var keyValues = new NameValueCollection();
        keyValues.Add("fileName", fileName);
        keyValues.Add("recordDomain", recordDomain);

        string requestUrl = base_url + apiURI + "?" + buildQueryString(keyValues);

        try
        {
            //为防止因HTTPS证书认证失败造成API调用失败,需要先忽略证书信任问题
            //.NET Framework 4.7.1及以上版本,请采用如下代码
            var sslHandler = new HttpClientHandler()
            {
                ServerCertificateCustomValidationCallback = (message, cert, chain, err) => { return true; }
            };
            sslHandler.AllowAutoRedirect = false; //关闭重定向
            HttpClient client = new HttpClient(sslHandler, true);
            //低于.NET Framework 4.7.1版本,请采用如下代码
            //var sslHandler = new HttpClientHandler();
            //sslHandler.AllowAutoRedirect = false;
            //HttpClient client = new HttpClient(sslHandler);
            //ServicePointManager.ServerCertificateValidationCallback = new
RemoteCertificateValidationCallback(CheckValidationResult);

            //请求Headers
            client.DefaultRequestHeaders.Add("Authorization", "AKSK realm=\"SDP\",profile=
\"UsernameToken\",type=\"Appkey\"");
            client.DefaultRequestHeaders.Add("X-AKSK", xaksk);

            var response = client.GetAsync(requestUrl).Result; //GET请求
            if (response.StatusCode.Equals(301))
            {
                location = Convert.ToString(response.Headers.GetValues("Location"));
            }
            else
            {
                var res = response.Content.ReadAsStringAsync().Result;
                Console.WriteLine(response.StatusCode);
                Console.WriteLine(res); //打印响应结果
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.StackTrace);
            Console.WriteLine(e.Message); //打印错误信息
        }

        return location;
    }

    static string buildAKSKHeader(string appKey, string appSecret)
    {
        string now = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ"); //Created
        string nonce = Guid.NewGuid().ToString().Replace("-", ""); //Nonce
        String str = nonce + now;

        byte[] keyByte = Encoding.UTF8.GetBytes(appSecret);
        byte[] messageBytes = Encoding.UTF8.GetBytes(str);
        using (var hmacsha256 = new HMACSHA256(keyByte))
        {
            byte[] hashmessage = hmacsha256.ComputeHash(messageBytes);
            string base64 = Convert.ToBase64String(hashmessage);
            return String.Format("UsernameToken Username=\"{0}\",PasswordDigest=\"{1}\",Nonce=
\"{2}\",Created=\"{3}\"",
                appKey, base64, nonce, now);
        }
```

```
    }

    static string buildQueryString(NameValueCollection keyValues)
    {
        StringBuilder temp = new StringBuilder();
        foreach (string item in keyValues.Keys)
        {

temp.Append(item).Append("=").Append(WebUtility.UrlEncode(keyValues.Get(item))).Append("&");
        }
        return temp.Remove(temp.Length - 1, 1).ToString();
    }

    //低于.NET Framework 4.7.1版本,启用如下方法
    //static bool CheckValidationResult(object sender, X509Certificate certificate, X509Chain chain,
SslPolicyErrors errors)
    //{
    //    return true;
    //}
  }
}
```

“呼叫状态通知 API”代码样例

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;

namespace voicecall_csharp_demo_x_aksk_
{
  class CallEventImpl
  {
    static void Main(string[] args)
    {
        //呼叫事件通知样例
        string jsonBody = JsonConvert.SerializeObject(new Dictionary<string, object>(){
            {"eventType", "callout"},
            {"statusInfo", new Dictionary<string, object>(){
                    {"sessionId", "1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com"},
                    {"timestamp", "2019-01-24 03:04:24"},
                    {"caller", "+86138*******2"},
                    {"called", "+86138*******1"},
                }
            }
        });

        Console.WriteLine("jsonBody:" + jsonBody);

        OnCallEvent(jsonBody); //呼叫事件处理
    }

    /// <summary>
    /// 呼叫事件通知,详细内容以接口文档为准
    /// </summary>
    /// <param name="jsonBody"></param>
    static void OnCallEvent(string jsonBody)
    {
        JObject jsonObj = (JObject)JsonConvert.DeserializeObject(jsonBody); //将通知消息解析为jsonObj
        string eventType = jsonObj["eventType"].ToString(); //通知事件类型

        if ("fee".Equals(eventType))
        {
            Console.WriteLine("EventType error:" + eventType);
            return;
        }

        if (!jsonObj.ContainsKey("statusInfo"))
        {
```

```
        Console.WriteLine("param error: no statusInfo.");
        return;
    }
    JObject statusInfo = (JObject)jsonObj["statusInfo"]; //呼叫状态事件信息

    Console.WriteLine("eventType:" + eventType); //打印通知事件类型

    //callout：呼出事件
    if ("callout".Equals(eventType))
    {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.ContainsKey("sessionId"))
        {
            Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
        }
        return;
    }
    //alerting：振铃事件
    if ("alerting".Equals(eventType))
    {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.ContainsKey("sessionId"))
        {
            Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
        }
        return;
    }
    //answer：应答事件
    if ("answer".Equals(eventType))
    {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.ContainsKey("sessionId"))
        {
            Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
        }
        return;
    }
    //collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ("collectInfo".Equals(eventType))
    {
        /**
         * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
         * 'sessionId': 通话链路的标识ID
```



```
            * 'digitInfo': 放音收号场景中,语音通话平台对开发者进行放音收号操作的结果描述
            */
            if (statusInfo.ContainsKey("digitInfo"))
            {
                Console.WriteLine("digitInfo:" + statusInfo["digitInfo"]);
            }
            return;
        }
        //disconnect：挂机事件
        if ("disconnect".Equals(eventType))
        {
            /**
            * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
            *
            * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
            * 'userData': 用户附属信息
            * 'sessionId': 通话链路的标识ID
            * 'caller': 主叫号码
            * 'called': 被叫号码
            * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
            * 'stateCode': 通话挂机的原因值
            * 'stateDesc': 通话挂机的原因值的描述
            */
            if (statusInfo.ContainsKey("sessionId"))
            {
                Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
            }
            return;
        }
    }
  }
}
```

“话单通知 API”代码样例

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;

namespace voicecall_csharp_demo_x_aksk_
{
   class FeeImpl
   {
      static void Main(string[] args)
      {
         //话单通知样例
         string jsonBody = JsonConvert.SerializeObject(new Dictionary<string, object>(){
            {"eventType", "fee"},
            {"feeLst",  new object[] {
               new Dictionary<string, object>(){
                     {"direction", 0}, //通话的呼叫方向,以语音通话平台为基准
                     {"spId", "CaaS_Test_01"}, //客户的云服务账号
                     {"appKey", "ka4k*****Em2"}, //应用的app_key，购买服务时下发,请替换为实际值
                     {"icid", "CAE-20190124110424-12019775"}, //呼叫记录的唯一标识
                     {"bindNum", "+8675512****78"}, //发起此次呼叫的CallEnabler业务号码,即绑定号码
                     {"sessionId",
"1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com"}, //通话链路的唯一标识
                     {"callerNum", "+86138****0022"}, //主叫号码
                     {"calleeNum", "+86138****0021"}, //被叫号码
                     {"fwdDisplayNum", "+86138****0022"}, //转接呼叫时的显示号码(仅语音回呼场景携带)
                     {"fwdDstNum", "+86138****7021"}, //转接呼叫时的转接号码(仅语音回呼场景携带)

                     {"fwdStartTime", "2019-01-24 03:04:31"}, //转接呼叫操作的开始时间(仅语音回呼场景携带)
                     {"fwdAlertingTime", "2019-01-24 03:04:36"}, //转接呼叫操作后的振铃时间(仅语音回呼场景
携带)
                     {"fwdAnswerTime", "2019-01-24 03:04:38"}, //转接呼叫操作后的应答时间(仅语音回呼场景
携带)
                     {"callEndTime", "2019-01-24 03:04:49"}, //呼叫结束时间
```

```
                {"fwdUnaswRsn", 0}, //转接呼叫操作失败的Q850原因值
                {"failTime", ""}, //呼入,呼出的失败时间
                {"ulFailReason", 0}, //通话失败的拆线点
                {"sipStatusCode", 0}, //呼入,呼出的失败SIP状态码
                {"callOutStartTime", "2019-01-24 03:04:24"}, //Initcall的呼出开始时间
                {"callOutAlertingTime", "2019-01-24 03:04:27"}, //Initcall的呼出振铃时间
                {"callOutAnswerTime", "2019-01-24 03:04:31"}, //Initcall的呼出应答时间
                {"callOutUnaswRsn", 0}, //Initcall的呼出失败的Q850原因值
                {"dynIVRStartTime", ""}, //自定义动态IVR开始时间(仅语音通知场景携带)
                {"dynIVRPath", ""}, //自定义动态IVR按键路径(仅语音通知场景携带)
                {"recordFlag", 0}, //录音标识
                {"recordStartTime", ""}, //录音开始时间(仅语音回呼场景携带)
                {"recordObjectName", ""}, //录音文件名(仅语音回呼场景携带)
                {"recordBucketName", ""}, //录音文件所在的目录名(仅语音回呼场景携带)
                {"recordDomain", ""}, //存放录音文件的域名(仅语音回呼场景携带)
                {"recordFileDownloadUrl", ""}, //录音文件下载地址(仅语音回呼场景携带)
                {"ttsPlayTimes", 0}, //应用TTS功能时,使用TTS的总次数
                {"ttsTransDuration", 0}, //应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
                {"serviceType", "002"}, //携带呼叫的业务类型信息
                {"hostName", "callenabler245.huaweicaas.com"}, //话单生成的服务器设备对应的主机名
                {"userData", "customerId123"} //用户附属信息
            }
        }
    }
});

    Console.WriteLine(jsonBody);

    OnFeeEvent(jsonBody); //话单处理
}

/// <summary>
/// 话单通知,详细内容以接口文档为准
/// </summary>
/// <param name="jsonBody"></param>
static void OnFeeEvent(string jsonBody)
{
    JObject jsonObj = (JObject)JsonConvert.DeserializeObject(jsonBody); //将通知消息解析为jsonObj
    string eventType = jsonObj["eventType"].ToString(); //通知事件类型

    if (!"fee".Equals(eventType))
    {
        Console.WriteLine("EventType error:" + eventType);
        return;
    }

    if (!jsonObj.ContainsKey("feeLst"))
    {
        Console.WriteLine("param error: no feeLst.");
        return;
    }
    JObject[] feeLst = jsonObj["feeLst"].ToObject<JObject[]>(); //呼叫话单事件信息

    //Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
    //短时间内有多个通话结束时语音通话平台会将话单合并推送,每条消息最多携带50个话单
    if (feeLst.Length > 1)
    {
        foreach (JObject loop in feeLst)
        {
            if (loop.ContainsKey("sessionId"))
            {
                Console.WriteLine("sessionId:" + loop["sessionId"]);
            }
        }
    }
    else if (feeLst.Length == 1)
    {
        if (feeLst[0].ContainsKey("sessionId"))
        {
```

```
                Console.WriteLine("sessionId:" + feeLst[0]["sessionId"]);
            }
        }
        else
        {
            Console.WriteLine("feeLst error: no element.");
        }
    }
  }
}
```

4.2 语音通知代码样例

4.2.1 Node.js

注：为节省开发时间，建议先使用Node.js代码样例进行调测，熟悉接口使用后，再参考Java、python、PHP或C#代码样例，结合**接口文档**进行功能开发。

样例	**语音通知场景API**、**呼叫状态通知API**、**话单通知API**
环境要求	Node.js 4.4.4及以上版本。
引用库	-

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音通知场景 API”代码样例

```
/*jshint esversion: 6 */
var https = require('https');
var data = require('./data.js');
var util = require('./reqUtil.js');

/**
 * voiceNotifyAPI
 * @param displayNbr
 * @param calleeNbr
 * @param playInfoList
 * @returns
 */
function voiceNotifyAPI(displayNbr, calleeNbr, playInfoList) {
    if(displayNbr === undefined || displayNbr === null || calleeNbr === undefined || calleeNbr === null) {
        return;
    }
    if(playInfoList === undefined || playInfoList === null) {
        return;
    }

    var method = 'POST';
    var uri = '/rest/httpsessions/callnotify/v2.0'; //v1.0 or v2.0
    var xaksk = util.buildAKSKHeader(data.data.callnotify_appid, data.data.callnotify_secret);
```



```
    var options = util.createOptions(method, uri, null, xaksk);

    var body = {
            /* 必填参数 */
            'displayNbr': displayNbr,//主叫用户手机终端的来电显示号码。
            'calleeNbr': calleeNbr,//被叫用户终端的来电显示号码。
            'playInfoList': playInfoList//播放信息列表，最大支持5个，每个播放信息携带的参数都可以不相同。
            /* 选填参数 */
//            'bindNbr': '+86123456789', //CallEnabler业务号码,即绑定号码
//            'statusUrl': 'aHR0cDovLzIxOC40LjMzLjU1Ojg4ODgvdGVzdA==', //要获取通话状态需要在请求中加入
statusUrl
//            'feeUrl': 'aHR0cDovLzIxOC40LjMzLjU1Ojg4ODgvdGVzdA==', //要获取话单需要在请求中加入feeUrl
//            'returnIdlePort': 'false', //指示是否需要返回平台空闲呼叫端口数量
//            'userData': 'customerId123' //设置用户的附属信息
    };
    var req = https.request(options, function (res) {
        var resHeaders = JSON.stringify(res.headers);
        res.setEncoding('utf8');
        res.on('data', function (chunk) {
            console.log(chunk);
        });
    });

    req.on('error', function(e) {
        console.error('problem with request: ' + e);
    });
    console.log(JSON.stringify(body));
    req.write(JSON.stringify(body));
    req.end();
}

/**
 * getPlayInfoList
 * @param notifyVoice
 * @param templateId
 * @param templateParas
 * @returns
 */
function getPlayInfoList(notifyVoice, templateId, templateParas) {
    var playInfoList = [{
        'notifyVoice': notifyVoice,//通知语音的放音文件名
        'templateId': templateId,//语音通知模板ID，用于唯一标识语音通知模板。
        'templateParas': templateParas,//语音通知模板的变量值列表，用于依次填充templateId参数指定的模板内
容中的变量。
//        'collectInd': 0, //是否进行收号
//        'replayAfterCollection': 'false', //设置是否在收号后重新播放notifyVoice或templateId指定的放音
//        'collectContentTriggerReplaying': '1' //设置触发重新放音的收号内容
    }];
    return playInfoList;
}

var playInfoList = getPlayInfoList('notifyvoice.wav', 'xxxxxx', ['3', '人民公园正门']);
voiceNotifyAPI('+8653159511234', '+8613500000001', playInfoList);
```

“呼叫状态通知 API”代码样例

```
/**
 * 呼叫事件通知
 * 客户平台收到RTC业务平台的呼叫事件通知的接口通知
 */
//呼叫事件通知样例
var jsonBody = JSON.stringify({
    'eventType': 'callout',
    'statusInfo': {
        'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp': '2019-01-24 03:04:24',
        'caller': '+8613800000022',
        'called': '+8613800000021'
```

```
    }
});
console.log('jsonBody:', jsonBody);
/**
 * 呼叫事件通知
 * @brief 详细内容以接口文档为准
 * @param jsonBody
 */
function onCallEvent(jsonBody) {
    var jsonObj = JSON.parse(jsonBody); //将通知消息解析为jsonObj
    var eventType = jsonObj.eventType; //通知事件类型

    if ('fee' === eventType) {
        console.log('EventType error:', eventType);
        return;
    }

    if (!jsonObj.hasOwnProperty('statusInfo')) {
        console.log('param error: no statusInfo.');
        return;
    }
    var statusInfo = jsonObj.statusInfo; //呼叫状态事件信息

    console.log('eventType:', eventType); //打印通知事件类型

    //callout：呼出事件
    if ('callout' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
    //alerting：振铃事件
    if ('alerting' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
    //answer：应答事件
    if ('answer' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
```

```
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
    //collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ('collectInfo' === eventType) {
        /**
         * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'sessionId': 通话链路的标识ID
         * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
         */
        if (statusInfo.hasOwnProperty('digitInfo')) {
            console.log('digitInfo:', statusInfo.digitInfo);
        }
        return;
    }
    //disconnect：挂机事件
    if ('disconnect' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
         * 'stateCode': 通话挂机的原因值
         * 'stateDesc': 通话挂机的原因值的描述
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
}
//呼叫事件处理
onCallEvent(jsonBody);
```

“话单通知 API”代码样例

```
/**
 * 话单通知
 * 客户平台收到RTC业务平台的话单通知的接口通知
 */
//话单通知样例
var jsonBody = JSON.stringify({
    'eventType': 'fee',
    'feeLst': [
        {
            'direction': 0,
            'spId': 'CaaS_Test_01',
            'appKey': 'ka4kESI5s3YyurL1wpx63s9YnEm2',
            'icid': 'CAE-20190124110424-12019775',
            'bindNum': '+8675512345678',
            'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
            'callerNum': '+8613800000022',
            'calleeNum': '+8613800000021',
            'fwdDisplayNum': '+8613800000022',
            'fwdDstNum': '+8613866887021',
            'fwdStartTime': '2019-01-24 03:04:31',
            'fwdAlertingTime': '2019-01-24 03:04:36',
            'fwdAnswerTime': '2019-01-24 03:04:38',
            'callEndTime': '2019-01-24 03:04:49',
            'fwdUnaswRsn': 0,
            'ulFailReason': 0,
            'sipStatusCode': 0,
```

```
            'callOutStartTime': '2019-01-24 03:04:24',
            'callOutAlertingTime': '2019-01-24 03:04:27',
            'callOutAnswerTime': '2019-01-24 03:04:31',
            'callOutUnaswRsn': 0,
            'recordFlag': 0,
            'ttsPlayTimes': 0,
            'ttsTransDuration': 0,
            'serviceType': '002',
            'hostName': 'callenabler245.huaweicaas.com'
        }
    ]
});
console.log('jsonBody:', jsonBody);
/**
 * 话单通知
 * @brief 详细内容以接口文档为准
 * @param jsonBody
 */
function onFeeEvent(jsonBody) {
    var jsonObj = JSON.parse(jsonBody); //将通知消息解析为jsonObj
    var eventType = jsonObj.eventType; //通知事件类型

    if ('fee' !== eventType) {
        console.log('EventType error:', eventType);
        return;
    }

    if (!jsonObj.hasOwnProperty('feeLst')) {
        console.log('param error: no feeLst.');
        return;
    }
    var feeLst = jsonObj.feeLst; //呼叫话单事件信息
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'direction': 通话的呼叫方向,以RTC业务平台为基准
     * 'spId': 客户的云服务账号
     * 'appKey': 应用的app_key
     * 'icid': 呼叫记录的唯一标识
     * 'bindNum': 发起此次呼叫的CallEnabler业务号码
     * 'sessionId': 通话链路的唯一标识
     * 'callerNum': 主叫号码
     * 'calleeNum': 被叫号码
     * 'fwdDisplayNum': 转接呼叫时的显示号码(仅语音回呼场景携带)
     * 'fwdDstNum': 转接呼叫时的转接号码(仅语音回呼场景携带)
     * 'fwdStartTime': 转接呼叫操作的开始时间(仅语音回呼场景携带)
     * 'fwdAlertingTime': 转接呼叫操作后的振铃时间(仅语音回呼场景携带)
     * 'fwdAnswerTime': 转接呼叫操作后的应答时间(仅语音回呼场景携带)
     * 'callEndTime': 呼叫结束时间
     * 'fwdUnaswRsn': 转接呼叫操作失败的Q850原因值
     * 'failTime': 呼入,呼出的失败时间
     * 'ulFailReason': 通话失败的拆线点
     * 'sipStatusCode': 呼入,呼出的失败SIP状态码
     * 'callOutStartTime': Initcall的呼出开始时间
     * 'callOutAlertingTime': Initcall的呼出振铃时间
     * 'callOutAnswerTime': Initcall的呼出应答时间
     * 'callOutUnaswRsn': Initcall的呼出失败的Q850原因值
     * 'dynIVRStartTime': 自定义动态IVR开始时间(仅语音通知场景携带)
     * 'dynIVRPath': 自定义动态IVR按键路径(仅语音通知场景携带)
     * 'recordFlag': 录音标识
     * 'recordStartTime': 录音开始时间(仅语音回呼场景携带)
     * 'recordObjectName': 录音文件名(仅语音回呼场景携带)
     * 'recordBucketName': 录音文件所在的目录名(仅语音回呼场景携带)
     * 'recordDomain': 存放录音文件的域名(仅语音回呼场景携带)
     * 'recordFileDownloadUrl': 录音文件下载地址(仅语音回呼场景携带)
     * 'ttsPlayTimes': 应用TTS功能时,使用TTS的总次数
     * 'ttsTransDuration': 应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
     * 'serviceType': 携带呼叫的业务类型信息
     * 'hostName': 话单生成的服务器设备对应的主机名
```

```
    * 'userData': 用户附属信息
    */
   //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
   if (feeLst.length > 1) {
       for ( var i=0; i<feeLst.length; i++) {
           if (feeLst[i].hasOwnProperty('sessionId')) {
               console.log('sessionId:', feeLst[i].sessionId);
           }
       }
   } else if (feeLst.length === 1) {
       if (feeLst[0].hasOwnProperty('sessionId')) {
           console.log('sessionId:', feeLst[0].sessionId);
       }
   } else {
       console.log('feeLst error: no element.');
   }
}
//话单处理
onFeeEvent(jsonBody);
```

4.2.2 Java

4.2.2.1 公共要求

注：使用前请务必先仔细阅读使用**注意事项**。

样例	语音通知场景API、获取录音文件下载地址API、呼叫状态与话单通知API
环境要求	JDK 1.6及以上版本。
引用库	**httpclient**、**httpcore**、**httpmime**、**commons-codec**、**commons-logging**、**jackson-databind**、**jackson-annotations**、**jackson-core**、**fastjson**、**log4j**

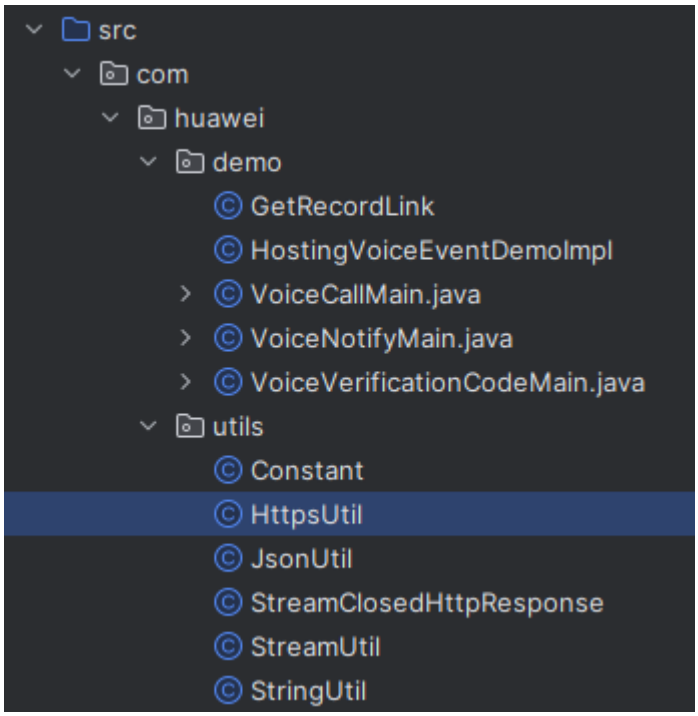
将下列代码样例复制到新建java文件中即可运行。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

公共依赖

使用Java样例时请参考以下工程目录结构：



Constant.java

```
/*
 * File Name: com.huawei.utils.Constant.java
 *
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

public class Constant {

    // purchase and get : base_url,appid,secret

    // please replace the IP and Port, when you use the demo.
    public static final String BASE_URL = "https://{domain}:{port}";

    // please replace the appId and secret, when you use the demo.
    public static final String CLICK2CALL_APPID = "******";
    public static final String CLICK2CALL_SECRET = "******";
    public static final String CALLNOTIFY_APPID = "******";
    public static final String CALLNOTIFY_SECRET = "******";
    public static final String CALLVERIFY_APPID = "******";
    public static final String CALLVERIFY_SECRET = "******";

    // *************************** The following constants do not need to be
    // modified *********************************//

    /*
     * request header
     * 1. HEADER_APP_AUTH
     * 2. HEADER_APP_AKSK
     * 3. AUTH_HEADER_VALUE
     * 4. AKSK_HEADER_FORMAT
     */
```

```
    public static final String HEADER_APP_AUTH = "Authorization";
    public static final String HEADER_APP_AKSK = "X-AKSK";
    public static final String AUTH_HEADER_VALUE = "AKSK realm=\"SDP\",profile=\"UsernameToken\",type=
\"Appkey\"";
    public static final String AKSK_HEADER_FORMAT = "UsernameToken Username=\"%s\",PasswordDigest=
\"%s\",Nonce=\"%s\",Created=\"%s\"";

    /*
     * Voice Call:
     * 1. VOICE_CALL_COMERCIAL
     * 2. VOICE_VERIFICATION_COMERCIAL
     * 3. CALL_NOTIFY_COMERCIAL
     * 4. VOICE_FILE_DOWNLOAD
     */
    public static final String VOICE_CALL_COMERCIAL = BASE_URL + "/rest/httpsessions/click2Call/v2.0";
    public static final String VOICE_VERIFICATION_COMERCIAL = BASE_URL + "/rest/httpsessions/callVerify/
v1.0";
    public static final String CALL_NOTIFY_COMERCIAL = BASE_URL + "/rest/httpsessions/callnotify/v2.0";
    public static final String VOICE_FILE_DOWNLOAD = BASE_URL + "/rest/provision/voice/record/v1.0";
}
```

HttpsUtil.java

```
package com.huawei.utils;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.SSLContextBuilder;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.client.HttpClients;

import java.io.IOException;
import java.net.URISyntaxException;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.List;
import java.util.Map;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

@SuppressWarnings("deprecation")
public class HttpsUtil extends DefaultHttpClient {
    public final static String CONTENT_LENGTH = "Content-Length";

    private static CloseableHttpClient httpClient = getHttpClient();

    public static CloseableHttpClient getHttpClient() {
        SSLContext sslcontext = null;
        try {
            sslcontext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
```



```
            @Override
            public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                return true;
            }
        }).build();
    } catch (KeyManagementException e) {
        return null;
    } catch (NoSuchAlgorithmException e) {
        return null;
    } catch (KeyStoreException e) {
        return null;
    }

    // Allow TLSv1, TLSv1.1, TLSv1.2 protocol only
    SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslcontext,
        new String[] {"TLSv1", "TLSv1.1", "TLSv1.2"}, null,
        SSLConnectionSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);

    CloseableHttpClient httpclient = HttpClients.custom()
        .setDefaultRequestConfig(RequestConfig.custom().setRedirectsEnabled(false).build())
        .setSSLSocketFactory(sslsf).build();

    return httpclient;
}

public HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
        return true;
    }
};

public void trustAllHttpsCertificates() throws Exception {
    TrustManager[] trustAllCerts = new TrustManager[1];
    TrustManager tm = new miTM();
    trustAllCerts[0] = tm;
    SSLContext sc = SSLContext.getInstance("SSL");
    sc.init(null, trustAllCerts, null);
    HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
}

static class miTM implements TrustManager, X509TrustManager {
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    public boolean isServerTrusted(X509Certificate[] certs) {
        return true;
    }

    public boolean isClientTrusted(X509Certificate[] certs) {
        return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String authType)
        throws java.security.cert.CertificateException {
        return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String authType)
        throws java.security.cert.CertificateException {
        return;
    }
}

public StreamClosedHttpResponse doPostJsonGetStatusLine(String url, Map<String, String> headerMap,
String content) {
    HttpPost request = new HttpPost(url);
    addRequestHeader(request, headerMap);
```

```
        request.setEntity(new StringEntity(content, ContentType.APPLICATION_JSON));

        HttpResponse response = executeHttpRequest(request);
        if (null == response) {
            System.out.println("The response body is null.");
        }

        return (StreamClosedHttpResponse) response;
    }

    public HttpResponse doGetWithParas(String url, List<NameValuePair> queryParams, Map<String, String>
headerMap)
        throws Exception {
        HttpGet request = new HttpGet();
        addRequestHeader(request, headerMap);

        URIBuilder builder;
        try {
            builder = new URIBuilder(url);
        } catch (URISyntaxException e) {
            System.out.printf("URISyntaxException: {}", e);
            throw new Exception(e);
        }

        if (queryParams != null && !queryParams.isEmpty()) {
            builder.setParameters(queryParams);
        }
        request.setURI(builder.build());

        return executeHttpRequest(request);
    }

    public StreamClosedHttpResponse doGetWithParasGetStatusLine(String url, List<NameValuePair>
queryParams,
        Map<String, String> headerMap) throws Exception {
        HttpResponse response = doGetWithParas(url, queryParams, headerMap);
        if (null == response) {
            System.out.println("The response body is null.");
        }

        return (StreamClosedHttpResponse) response;
    }

    private static void addRequestHeader(HttpUriRequest request, Map<String, String> headerMap) {
        if (headerMap == null) {
            return;
        }

        for (String headerName : headerMap.keySet()) {
            if (CONTENT_LENGTH.equalsIgnoreCase(headerName)) {
                continue;
            }

            String headerValue = headerMap.get(headerName);
            request.addHeader(headerName, headerValue);
        }
    }

    private HttpResponse executeHttpRequest(HttpUriRequest request) {
        HttpResponse response = null;

        try {
            response = httpClient.execute(request);
        } catch (Exception e) {
            System.out.println("executeHttpRequest failed.");
            System.out.println(e.getStackTrace());
        } finally {
            try {
                response = new StreamClosedHttpResponse(response);
```

```
            } catch (IOException e) {
                System.out.println("IOException: " + e.getMessage());
            }
        }

        return response;
    }
}
```

JsonUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */

package com.huawei.utils;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;

import java.io.IOException;

public class JsonUtil {

    private static ObjectMapper objectMapper;

    static {
        objectMapper = new ObjectMapper();

        // 设置FAIL_ON_EMPTY_BEANS属性,当序列化空对象不要抛异常
        objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, false);

        // 设置FAIL_ON_UNKNOWN_PROPERTIES属性,当JSON字符串中存在Java对象没有的属性,忽略
        objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
    }

    /**
     * Convert Object to JsonString
     *
     * @param jsonObj
     * @return
     */
    public static String jsonObj2Sting(Object jsonObj) {
        String jsonString = null;

        try {
            jsonString = objectMapper.writeValueAsString(jsonObj);
        } catch (IOException e) {
            System.out.printf("pasre json Object[{}] to string failed.", jsonString);
        }

        return jsonString;
    }

    /**
     * Convert JsonString to Simple Object
     *
     * @param jsonString
     * @param cls
     * @return
     */
```

```
    public static <T> T jsonString2SimpleObj(String jsonString, Class<T> cls) {
        T jsonObj = null;

        try {
            jsonObj = objectMapper.readValue(jsonString, cls);
        } catch (IOException e) {
            System.out.printf("pasre json Object[{}] to string failed.", jsonString);
        }

        return jsonObj;
    }
}
```

StreamClosedHttpResponse.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

import java.io.IOException;
import java.util.Locale;

import org.apache.http.Header;
import org.apache.http.HeaderIterator;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.ProtocolVersion;
import org.apache.http.StatusLine;
import org.apache.http.params.HttpParams;

@SuppressWarnings("deprecation")
public class StreamClosedHttpResponse implements HttpResponse {

    private final HttpResponse original;

    private final String content;

    public StreamClosedHttpResponse(final HttpResponse original) throws UnsupportedOperationException,
IOException {
        this.original = original;

        HttpEntity entity = original.getEntity();
        if (entity != null && entity.isStreaming()) {
            String encoding = entity.getContentEncoding() != null ? entity.getContentEncoding().getValue() :
null;
            content = StreamUtil.inputStream2String(entity.getContent(), encoding);
        } else {
            content = null;
        }
    }

    @Override
    public StatusLine getStatusLine() {
        return original.getStatusLine();
    }

    @Override
    public void setStatusLine(final StatusLine statusline) {
        original.setStatusLine(statusline);
    }
```

```
@Override
public void setStatusLine(final ProtocolVersion ver, final int code) {
    original.setStatusLine(ver, code);
}

@Override
public void setStatusLine(final ProtocolVersion ver, final int code, final String reason) {
    original.setStatusLine(ver, code, reason);
}

@Override
public void setStatusCode(final int code) throws IllegalStateException {
    original.setStatusCode(code);
}

@Override
public void setReasonPhrase(final String reason) throws IllegalStateException {
    original.setReasonPhrase(reason);
}

@Override
public HttpEntity getEntity() {
    return original.getEntity();
}

@Override
public void setEntity(final HttpEntity entity) {
    original.setEntity(entity);
}

@Override
public Locale getLocale() {
    return original.getLocale();
}

@Override
public void setLocale(final Locale loc) {
    original.setLocale(loc);
}

@Override
public ProtocolVersion getProtocolVersion() {
    return original.getProtocolVersion();
}

@Override
public boolean containsHeader(final String name) {
    return original.containsHeader(name);
}

@Override
public Header[] getHeaders(final String name) {
    return original.getHeaders(name);
}

@Override
public Header getFirstHeader(final String name) {
    return original.getFirstHeader(name);
}

@Override
public Header getLastHeader(final String name) {
    return original.getLastHeader(name);
}

@Override
public Header[] getAllHeaders() {
    return original.getAllHeaders();
}
```

```
@Override
public void addHeader(final Header header) {
    original.addHeader(header);
}

@Override
public void addHeader(final String name, final String value) {
    original.addHeader(name, value);
}

@Override
public void setHeader(final Header header) {
    original.setHeader(header);
}

@Override
public void setHeader(final String name, final String value) {
    original.setHeader(name, value);
}

@Override
public void setHeaders(final Header[] headers) {
    original.setHeaders(headers);
}

@Override
public void removeHeader(final Header header) {
    original.removeHeader(header);
}

@Override
public void removeHeaders(final String name) {
    original.removeHeaders(name);
}

@Override
public HeaderIterator headerIterator() {
    return original.headerIterator();
}

@Override
public HeaderIterator headerIterator(final String name) {
    return original.headerIterator(name);
}

@Override
public String toString() {
    final StringBuilder sb = new StringBuilder("HttpResponseProxy{");
    sb.append(original);
    sb.append('}');
    return sb.toString();
}

@Override
@Deprecated
public HttpParams getParams() {
    return original.getParams();
}

@Override
@Deprecated
public void setParams(final HttpParams params) {
    original.setParams(params);
}

public String getContent() {
    return content;
```

```
    }
}
```

StreamUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

import java.io.Closeable;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class StreamUtil {

    private static final String DEFAULT_ENCODING = "utf-8";

    public static String inputStream2String(InputStream in, String charsetName) {
        if (in == null) {
            return null;
        }

        InputStreamReader inReader = null;

        try {
            if (StringUtil.strIsNullOrEmpty(charsetName)) {
                inReader = new InputStreamReader(in, DEFAULT_ENCODING);
            } else {
                inReader = new InputStreamReader(in, charsetName);
            }

            int readLen = 0;
            char[] buffer = new char[1024];
            StringBuilder strBuf = new StringBuilder();
            while ((readLen = inReader.read(buffer)) != -1) {
                strBuf.append(buffer, 0, readLen);
            }

            return strBuf.toString();
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            closeStream(inReader);
        }

        return null;
    }

    public static void closeStream(Closeable closeable) {
        if (closeable != null) {
            try {
                closeable.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}
```

StringUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

//import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.text.SimpleDateFormat;
import java.util.Base64; // JDK versionâ‰¥1.8
import java.util.Calendar;
import java.util.Locale;
import java.util.TimeZone;
import java.util.UUID;

//import org.apache.commons.codec.binary.Base64; //JDK version<1.8
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class StringUtil {

    public static boolean strIsNullOrEmpty(String s) {
        return (null == s || s.trim().length() < 1);
    }

    public static String buildAKSKHeader(String appKey, String appSecret) throws Exception {
        if (StringUtil.strIsNullOrEmpty(appKey) || StringUtil.strIsNullOrEmpty(appSecret)) {
            return null;
        }

        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");
        format.setTimeZone(TimeZone.getTimeZone("UTC"));
        Calendar calendar = Calendar.getInstance();
        String time = format.format(calendar.getTime());
        String stNonce = UUID.randomUUID().toString().replace("-", "").toUpperCase(Locale.ROOT);
        String str = stNonce + time;
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(appSecret.getBytes(StandardCharsets.UTF_8), "HmacSHA256"));
        byte[] authBytes = mac.doFinal(str.getBytes(StandardCharsets.UTF_8));
        String passwordDigestBase64Str = encodeBase64(authBytes);
        return String.format(Constant.AKSK_HEADER_FORMAT, appKey, passwordDigestBase64Str, stNonce,
time);
    }

    private static String encodeBase64(byte[] bytes) {
        if (bytes.length == 0) {
            return null;
        } else {
            return new String(org.apache.commons.codec.binary.Base64.encodeBase64(bytes),
StandardCharsets.UTF_8);
        }
    }
}
```

4.2.2.2 代码样例

“语音通知场景 API”代码样例

```
package com.huawei.demo;

import java.util.*;
import javax.net.ssl.HttpsURLConnection;
```



```
import com.huawei.utils.Constant;
import com.huawei.utils.HttpsUtil;
import com.huawei.utils.JsonUtil;
import com.huawei.utils.StreamClosedHttpResponse;
import com.huawei.utils.StringUtil;

public class VoiceNotifyMain {
    // 接口返回值
    private static String status = "";
    private static String resultcode = "";
    private static String resultdesc = "";
    // 语音通知接口返回值
    private static String sessionId = "";
    // 语音通知业务类实体
    public static VoiceNotify callNotifyAPI = new VoiceNotify();

    // 调用接口成功标识
    private static final String success = "200";

    public static void main(String args[]) throws Exception {

        // TODO 程序前端要求发起语音通知呼叫,首先使用getplayInfo构造构造playInfoList参数,然后调用
doCallNotify方法.
        // 以下代码仅供调试使用,实际开发时请删除
        // 构造playInfoList参数
        List<Map<String, Object>> playInfoList = new ArrayList<Map<String, Object>>();
        // 使用音频文件作为第一段放音内容
        playInfoList.add(callNotifyAPI.getplayInfo("test.wav"));
        // 使用v2.0版本接口的语音通知模板作为第二段放音内容
        String templateId = "test";
        List<String> templateParas = new ArrayList<String>();
        templateParas.add("3");
        templateParas.add("1栋保安亭");
        playInfoList.add(callNotifyAPI.getplayInfo(templateId, templateParas));
        // 调用doCallNotify方法
        VoiceNotifyMain.doCallNotify("+8653159511234", "+8613800000001", playInfoList);
        if (status.indexOf(success) != -1) {
            System.out.println(status);
            System.out.println(resultcode + " " + resultdesc);
            System.out.println("The session id is: " + sessionId);
        }

        // TODO 需要接收状态和话单时,请参考"呼叫状态和话单通知API"接口实现状态通知和话单的接收和解析
        // HostingVoiceEventDemoImpl
    }

    /*
     * 前端需要发起语音通知呼叫时,调用此方法 该示例只仅体现必选参数,可选参数根据接口文档和实际情况配置.
     */
    public static void doCallNotify(String displayNbr, String calleeNbr, List<Map<String, Object>> playInfoList)
            throws Exception {

        Boolean retry = false;
        // 调用语音通知接口,直至成功
        do {
            status = callNotifyAPI.callNotifyAPI(displayNbr, calleeNbr, playInfoList);
            if (status.indexOf(success) != -1) {
                retry = false;
                // 调用成功,记录返回的信息.
                resultcode = callNotifyAPI.getResponsePara("resultcode");
                resultdesc = callNotifyAPI.getResponsePara("resultdesc");
                sessionId = callNotifyAPI.getResponsePara("sessionId");
            } else {
                retry = true;
                // 调用失败,获取错误码和错误描述.
                resultcode = callNotifyAPI.getResponsePara("resultcode");
                resultdesc = callNotifyAPI.getResponsePara("resultdesc");
                // 处理错误
```

```
                VoiceNotifyMain.processError();
            }
        } while (retry);
    }

    // 当API的返回值不是200时,处理错误.
    private static void processError() throws InterruptedException {

        // TODO 根据错误码和错误码描述处理问题
        // 以下代码仅供调试使用,实际开发时请删除
        System.out.println(status);
        System.out.println(resultcode + " " + resultdesc);
        System.exit(-1);
    }
}

class VoiceNotify {
    // 语音通知API的调用地址
    private String urlCallNotify;
    // 接口响应的消息体
    private Map<String, String> Responsebody;
    // Https实体
    private HttpsUtil httpsUtil;

    public VoiceNotify() {
        // 商用地址
        urlCallNotify = Constant.CALL_NOTIFY_COMERCIAL;
        Responsebody = new HashMap<>();
    }

    @SuppressWarnings("unchecked")
    /*
     * 该示例只仅体现必选参数,可选参数根据接口文档和实际情况配置. 该示例不体现参数校验,请根据各参数的格
式要求自行实现校验功能.
     * playInfoList为最大个数为5的放音内容参数列表,每个放音内容参数以Map<String,Object>格式存储,
     * 放音内容参数的构造方法请参考getplayInfo方法.
     */
    public String callNotifyAPI(String displayNbr, String calleeNbr, List<Map<String, Object>> playInfoList)
        throws Exception {

        httpsUtil = new HttpsUtil();

        // 忽略证书信任问题
        httpsUtil.trustAllHttpsCertificates();
        HttpsURLConnection.setDefaultHostnameVerifier(httpsUtil.hv);



        // 请求Headers
        Map<String, String> headerMap = new HashMap<>();
        headerMap.put(Constant.HEADER_APP_AUTH, Constant.AUTH_HEADER_VALUE);
        headerMap.put(Constant.HEADER_APP_AKSK,
            StringUtil.buildAKSKHeader(Constant.CALLNOTIFY_APPID, Constant.CALLNOTIFY_SECRET));

        // 构造消息体
        Map<String, Object> bodys = new HashMap<>();
        bodys.put("displayNbr", displayNbr);//主叫用户手机终端的来电显示号码。
        bodys.put("calleeNbr", calleeNbr);//发起呼叫时所拨打的被叫号码。
        bodys.put("playInfoList", playInfoList);//播放信息列表，最大支持5个，每个播放信息携带的参数都可以不
相同。
        String jsonRequest = JsonUtil.jsonObj2Sting(bodys);

        /*
         * Content-Type为application/json且请求方法为post时, 使用doPostJsonGetStatusLine方法构造http
         * request并获取响应.
         */
        StreamClosedHttpResponse responseCallNotify = httpsUtil.doPostJsonGetStatusLine(urlCallNotify,
headerMap,
            jsonRequest);
```

```
        // 响应的消息体写入Responsebody.
        Responsebody = JsonUtil.jsonString2SimpleObj(responseCallNotify.getContent(),
Responsebody.getClass());

        // 返回响应的status.
        return responseCallNotify.getStatusLine().toString();
    }

    /*
     * 构造playInfoList中携带的放音内容参数 使用语音文件或者v1.0版本接口的TTS文本作为放音内容
     */
    public Map<String, Object> getplayInfo(String fileorTTS) {
        Map<String, Object> body = new HashMap<String, Object>();
        // 音频文件只支持wave格式,文件名以.wav结尾
        if (fileorTTS.endsWith(".wav")) {
            body.put("notifyVoice", fileorTTS);
        } else {
            System.out.println("Only .wav file is supported.");
        }
        return body;
    }

    /*
     * 构造playInfoList中携带的放音内容参数 使用v2.0版本接口的TTS模板作为放音内容 重构getplayInfo方法
     */
    public Map<String, Object> getplayInfo(String templateId, List<String> templateParas) {
        Map<String, Object> bodys = new HashMap<String, Object>();
        bodys.put("templateId", templateId);
        bodys.put("templateParas", templateParas);
        return bodys;
    }

    // 获取整个响应消息体
    public Map<String, String> getResponsebody() {
        return this.Responsebody;
    }

    // 获取响应消息体中的单个参数
    public String getResponsePara(String ParaName) {
        return this.Responsebody.get(ParaName);
    }
}
```

“呼叫状态通知 API”与“话单通知 API”代码样例

```
package com.huawei.demo;

import org.apache.log4j.Logger;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;

/**
 * 呼叫事件通知/话单通知
 * 客户平台收到RTC业务平台的呼叫事件通知/话单通知的接口通知
 */
public class HostingVoiceEventDemoImpl {
    private static Logger logger = Logger.getLogger(HostingVoiceEventDemoImpl.class);

    /**
     * 呼叫事件 for 语音回呼/语音通知/语音验证码
     *
     * @param jsonBody
     * @breif 详细内容以接口文档为准
     */
    public static void onCallEvent(String jsonBody) {
        // 封装JOSN请求
```

```
JSONObject json = JSON.parseObject(jsonBody);
String eventType = json.getString("eventType"); // 通知事件类型

if ("fee".equalsIgnoreCase(eventType)) {
    logger.info("EventType error: " + eventType);
    return;
}

if (!(json.containsKey("statusInfo"))) {
    logger.info("param error: no statusInfo.");
    return;
}
JSONObject statusInfo = json.getJSONObject("statusInfo"); // 呼叫状态事件信息

logger.info("eventType: " + eventType); // 打印通知事件类型

//callout：呼出事件
if ("callout".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.containsKey("sessionId")) {
        logger.info("sessionId: " + statusInfo.getString("sessionId"));
    }
    return;
}
//alerting：振铃事件
if ("alerting".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.containsKey("sessionId")) {
        logger.info("sessionId: " + statusInfo.getString("sessionId"));
    }
    return;
}
//answer：应答事件
if ("answer".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.containsKey("sessionId")) {
        logger.info("sessionId: " + statusInfo.getString("sessionId"));
    }
    return;
}
//collectInfo：放音收号结果事件,仅应用于语音通知场景
if ("collectInfo".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
     *
```

```
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'sessionId': 通话链路的标识ID
         * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
         */
        if (statusInfo.containsKey("digitInfo")) {
            logger.info("digitInfo: " + statusInfo.getString("digitInfo"));
        }
        return;
    }
    //disconnect：挂机事件
    if ("disconnect".equalsIgnoreCase(eventType)) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
         * 'stateCode': 通话挂机的原因值
         * 'stateDesc': 通话挂机的原因值的描述
         */
        if (statusInfo.containsKey("sessionId")) {
            logger.info("sessionId: " + statusInfo.getString("sessionId"));
        }
        return;
    }
}

/**
 * 话单通知 for 语音回呼/语音通知/语音验证码
 *
 * @param jsonBody
 * @breif 详细内容以接口文档为准
 */
public static void onFeeEvent(String jsonBody) {
    // 封装JSON请求
    JSONObject json = JSON.parseObject(jsonBody);
    String eventType = json.getString("eventType"); // 通知事件类型

    if (!("fee".equalsIgnoreCase(eventType))) {
        logger.info("EventType error: " + eventType);
        return;
    }

    if (!(json.containsKey("feeLst"))) {
        logger.info("param error: no feeLst.");
        return;
    }
    JSONArray feeLst = json.getJSONArray("feeLst"); // 呼叫话单事件信息
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'direction': 通话的呼叫方向,以RTC业务平台为基准
     * 'spId': 客户的云服务账号
     * 'appKey': 应用的app_key
     * 'icid': 呼叫记录的唯一标识
     * 'bindNum': 发起此次呼叫的CallEnabler业务号码
     * 'sessionId': 通话链路的唯一标识
     * 'callerNum': 主叫号码
     * 'calleeNum': 被叫号码
     * 'fwdDisplayNum': 转接呼叫时的显示号码(仅语音回呼场景携带)
     * 'fwdDstNum': 转接呼叫时的转接号码(仅语音回呼场景携带)
     * 'fwdStartTime': 转接呼叫操作的开始时间(仅语音回呼场景携带)
     * 'fwdAlertingTime': 转接呼叫操作后的振铃时间(仅语音回呼场景携带)
     * 'fwdAnswerTime': 转接呼叫操作后的应答时间(仅语音回呼场景携带)
     * 'callEndTime': 呼叫结束时间
     * 'fwdUnaswRsn': 转接呼叫操作失败的Q850原因值
```



```
        * 'failTime': 呼入,呼出的失败时间
        * 'ulFailReason': 通话失败的拆线点
        * 'sipStatusCode': 呼入,呼出的失败SIP状态码
        * 'callOutStartTime': Initcall的呼出开始时间
        * 'callOutAlertingTime': Initcall的呼出振铃时间
        * 'callOutAnswerTime': Initcall的呼出应答时间
        * 'callOutUnaswRsn': Initcall的呼出失败的Q850原因值
        * 'dynIVRStartTime': 自定义动态IVR开始时间(仅语音通知场景携带)
        * 'dynIVRPath': 自定义动态IVR按键路径(仅语音通知场景携带)
        * 'recordFlag': 录音标识
        * 'recordStartTime': 录音开始时间(仅语音回呼场景携带)
        * 'recordObjectName': 录音文件名(仅语音回呼场景携带)
        * 'recordBucketName': 录音文件所在的目录名(仅语音回呼场景携带)
        * 'recordDomain': 存放录音文件的域名(仅语音回呼场景携带)
        * 'recordFileDownloadUrl': 录音文件下载地址(仅语音回呼场景携带)
        * 'ttsPlayTimes': 应用TTS功能时,使用TTS的总次数
        * 'ttsTransDuration': 应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
        * 'serviceType': 携带呼叫的业务类型信息
        * 'hostName': 话单生成的服务器设备对应的主机名
        * 'userData': 用户附属信息
        */
        //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
        if (feeLst.size() > 1) {
            for (Object loop : feeLst) {
                if (((JSONObject)loop).containsKey("sessionId")) {
                    logger.info("sessionId: " + ((JSONObject)loop).getString("sessionId"));
                }
            }
        } else if (feeLst.size() == 1) {
            if (feeLst.getJSONObject(0).containsKey("sessionId")) {
                logger.info("sessionId: " + feeLst.getJSONObject(0).getString("sessionId"));
            }
        } else {
            logger.info("feeLst error: no element.");
        }
    }
}
```

4.2.3 Python

样例	语音通知API、呼叫状态通知API、话单通知API
环境要求	Python 3.0及以上版本。
引用库	requests 2.18.1 1. 请自行下载安装Python 3.x，并完成环境配置。 2. 打开命令行窗口，执行pip install requests命令。 3. 执行pip list查看安装结果。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音通知 API”代码样例

```
# -*- coding: utf-8 -*-
import base64
import hashlib
import requests #需要先使用pip install requests命令安装依赖
import time
import uuid
import hmac

from hashlib import sha256

#必填,请参考"开发准备-申请资源"获取如下数据,替换为实际值
base_url = 'https://{domain}:{port}'#APP接入地址,购买服务时下发,请替换为实际值
appKey = '***appKey***'#语音通知应用的appKey,购买服务时下发,请替换为实际值
appSecret = '***appSecret***'#语音通知应用的appSecret,购买服务时下发,请替换为实际值

def buildAKSKHeader(appKey, appSecret):
    now = time.strftime('%Y-%m-%dT%H:%M:%SZ') #Created
    nonce = str(uuid.uuid4()).replace('-','') #Nonce
    digist = hmac.new(appSecret.encode(), (nonce + now).encode(), digestmod=sha256).digest()
    digestBase64 = base64.b64encode(digist).decode() #PasswordDigest
    return 'UsernameToken Username="{}",PasswordDigest="{}",Nonce="{}",Created="{}"'.format(appKey,
digestBase64, nonce, now);

def voiceNotifyAPI(displayNbr, calleeNbr, playInfoList):
    if len(displayNbr) < 1 or len(calleeNbr) < 1 or playInfoList is None:
        return

    apiUri = '/rest/httpsessions/callnotify/v2.0' #v1.0 or v2.0
    requestUrl = base_url + apiUri

    header = {
        'Content-Type': 'application/json;charset=UTF-8',
        'Authorization': 'AKSK realm="SDP",profile="UsernameToken",type="Appkey"',
        'X-AKSK': buildAKSKHeader(appKey, appSecret)
    }

    jsonData = {
        # 必填参数
        'displayNbr': displayNbr,#主叫用户手机终端的来电显示号码。
        'calleeNbr': calleeNbr,#发起呼叫时所拨打的被叫号码。
        'playInfoList': playInfoList#播放信息列表，最大支持5个，每个播放信息携带的参数都可以不相同。
        # 选填参数
#         'statusUrl': '', #设置SP接收状态上报的URL,要求使用BASE64编码
#         'feeUrl': '', #设置SP接收话单上报的URL,要求使用BASE64编码
#         'returnIdlePort': 'false', #指示是否需要返回平台空闲呼叫端口数量
#         'userData': 'customerId123' #设置用户的附属信息
    }

    try:
        r = requests.post(requestUrl, json=jsonData, headers=header, verify=False)
        print(r.text) #打印响应结果
    except requests.exceptions.HTTPError as e:
        print(e.code)
        print(e.read().decode('utf-8')) #打印错误信息
    pass

def getPlayInfoList(notifyVoice, templateId, templateParas):
    playInfoList = [{
        'notifyVoice': notifyVoice,
        'templateId': templateId,
        'templateParas': templateParas
#         'collectInd': 0, #是否进行收号
#         'replayAfterCollection': 'false', #设置是否在收号后重新播放notifyVoice或templateId指定的放音
#         'collectContentTriggerReplaying': '1' #设置触发重新放音的收号内容
    }]
    return playInfoList

if __name__ == '__main__':
```

```
playInfoList = getPlayInfoList('notifyvoice.wav', 'xxxxxx', ['3', '人民公园正门'])
voiceNotifyAPI('+86531*******4', '+86135*******1', playInfoList)
```

“呼叫状态通知 API”代码样例

```
# -*- coding: utf-8 -*-
'''
呼叫事件通知
客户平台收到语音通话平台的呼叫事件通知的接口通知
'''
import json

#呼叫事件通知样例
jsonBody = json.dumps({
    'eventType': 'callout',
    'statusInfo': {
        'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp': '2019-01-24 03:04:24',
        'caller': '+86138*******2',
        'called': '+86138*******1',
        'userData': 'customerId123'
    }
}).encode('ascii')

print(jsonBody)

'''
呼叫事件通知
@see: 详细内容以接口文档为准
@param param: jsonBody
@return:
'''
def onCallEvent(jsonBody):
    jsonObj = json.loads(jsonBody) #将通知消息解析为jsonObj
    eventType = jsonObj['eventType'] #通知事件类型

    if ('fee' == eventType):
        print('EventType error: ' + eventType)
        return

    if ('statusInfo' not in jsonObj):
        print('param error: no statusInfo.')
        return
    statusInfo = jsonObj['statusInfo'] #呼叫状态事件信息

    print('eventType: ' + eventType) #打印通知事件类型

    #callout：呼出事件
    if ('callout' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #alerting：振铃事件
    if ('alerting' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
```

```
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #answer：应答事件
    if ('answer' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ('collectInfo' == eventType):
        '''
        Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'sessionId': 通话链路的标识ID
        'digitInfo': 放音收号场景中,语音通话平台对开发者进行放音收号操作的结果描述
        '''
        if ('digitInfo' in statusInfo):
            print('digitInfo: ' + statusInfo['digitInfo'])
        return
    #disconnect：挂机事件
    if ('disconnect' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        'partyType': 挂机的用户类型,仅在语音回呼场景携带
        'stateCode': 通话挂机的原因值
        'stateDesc': 通话挂机的原因值的描述
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return

if __name__ == '__main__':
    onCallEvent(jsonBody) #呼叫事件处理
```

“话单通知 API”代码样例

```
# -*- coding: utf-8 -*-
'''
话单通知
客户平台收到语音通话平台的话单通知的接口通知
'''
import json

#话单通知样例
jsonBody = json.dumps({
    'eventType': 'fee',
    'feeLst': [
        {
            'direction': 0, #通话的呼叫方向,以语音通话平台为基准
            'spId': 'CaaS_Test_01', #客户的云服务账号
```

```
        'appKey': 'ka4k******9YnEm2', #应用的app_key
        'icid': 'CAE-20190124110424-12019775', #呼叫记录的唯一标识
        'bindNum': '+86755*******8', #发起此次呼叫的CallEnabler业务号码,即绑定号码
        'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com', #通话链路的
唯一标识
        'callerNum': '+86138*******2', #主叫号码
        'calleeNum': '+86138*******1', #被叫号码
        'fwdDisplayNum': '+86138*******2', #转接呼叫时的显示号码(仅语音回呼场景携带)
        'fwdDstNum': '+86138*******1', #转接呼叫时的转接号码(仅语音回呼场景携带)

        'fwdStartTime': '2019-01-24 03:04:31', #转接呼叫操作的开始时间(仅语音回呼场景携带)
        'fwdAlertingTime': '2019-01-24 03:04:36', #转接呼叫操作后的振铃时间(仅语音回呼场景携带)
        'fwdAnswerTime': '2019-01-24 03:04:38', #转接呼叫操作后的应答时间(仅语音回呼场景携带)
        'callEndTime': '2019-01-24 03:04:49', #呼叫结束时间
        'fwdUnaswRsn': 0, #转接呼叫操作失败的Q850原因值
        'failTime': '', #呼入,呼出的失败时间
        'ulFailReason': 0, #通话失败的拆线点
        'sipStatusCode': 0, #呼入,呼出的失败SIP状态码
        'callOutStartTime': '2019-01-24 03:04:24', #Initcall的呼出开始时间
        'callOutAlertingTime': '2019-01-24 03:04:27', #Initcall的呼出振铃时间
        'callOutAnswerTime': '2019-01-24 03:04:31', #Initcall的呼出应答时间
        'callOutUnaswRsn': 0, #Initcall的呼出失败的Q850原因值
        'dynIVRStartTime': '', #自定义动态IVR开始时间(仅语音通知场景携带)
        'dynIVRPath': '', #自定义动态IVR按键路径(仅语音通知场景携带)
        'recordFlag': 0, #录音标识
        'recordStartTime': '', #录音开始时间(仅语音回呼场景携带)
        'recordObjectName': '', #录音文件名(仅语音回呼场景携带)
        'recordBucketName': '', #录音文件所在的目录名(仅语音回呼场景携带)
        'recordDomain': '', #存放录音文件的域名(仅语音回呼场景携带)
        'recordFileDownloadUrl': '', #录音文件下载地址(仅语音回呼场景携带)
        'ttsPlayTimes': 0, #应用TTS功能时,使用TTS的总次数
        'ttsTransDuration': 0, #应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
        'serviceType': '002', #携带呼叫的业务类型信息
        'hostName': 'callenabler245.huaweicaas.com', #话单生成的服务器设备对应的主机名
        'userData': '' #用户附属信息
    }
  ]
}).encode('ascii')

print(jsonBody)

'''
话单通知
@see: 详细内容以接口文档为准
@param param: jsonBody
@return:
'''
def onFeeEvent(jsonBody):
  jsonObj = json.loads(jsonBody) #将通知消息解析为jsonObj
  eventType = jsonObj['eventType'] #通知事件类型

  if ('fee' != eventType):
    print('EventType error: ' + eventType)
    return

  if ('feeLst' not in jsonObj):
    print('param error: no feeLst.');
    return

  feeLst = jsonObj['feeLst']

  #Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
  #短时间内有多个通话结束时语音通话平台会将话单合并推送,每条消息最多携带50个话单
  if (len(feeLst) > 1):
    for loop in feeLst:
      if ('sessionId' in loop):
        print('sessionId: ' + loop['sessionId'])
  elif(len(feeLst) == 1):
    if ('sessionId' in feeLst[0]):
```

```
        print('sessionId: ' + feeLst[0]['sessionId'])
    else:
        print('feeLst error: no element.');

if __name__ == '__main__':
    onFeeEvent(jsonBody) #话单处理
```

4.2.4 PHP

样例	语音通知API、呼叫状态通知API、话单通知API
环境要求	PHP 7.0及以上版本。
引用库	-

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音通知 API”代码样例

```
<?php
include 'data.php';
include 'util.php';

/**
 * voiceNotifyAPI
 * @param string $displayNbr
 * @param string $calleeNbr
 * @param string $playInfoList
 */
function voiceNotifyAPI($displayNbr, $calleeNbr, $playInfoList) {
    if (empty($displayNbr) || empty($calleeNbr) || empty($playInfoList)) {
        return;
    }

    $method = 'POST';
    $apiUri = '/rest/httpsessions/callnotify/v2.0'; //v1.0 or v2.0
    $xaksk = buildAKSKHeader(getCallNotify_AppId(), getCallNotify_Secret());

    $content = json_encode([
        /* 必填参数*/
        'displayNbr' => $displayNbr,//主叫用户手机终端的来电显示号码。
        'calleeNbr' => $calleeNbr,//发起呼叫时所拨打的被叫号码。
        'playInfoList' => $playInfoList//放信息列表，最大支持5个，每个播放信息携带的参数都可以不相同。
        /* 选填参数*/
//        'bindNbr' => '+86123456789', //CallEnabler业务号码,即绑定号码
//        'statusUrl' => '', //设置SP接收状态上报的URL,要求使用BASE64编码
//        'feeUrl' => '', //设置SP接收话单上报的URL,要求使用BASE64编码
//        'returnIdlePort' => 'false', //指示是否需要返回平台空闲呼叫端口数量
//        'userData' => 'customerId123' //设置用户的附属信息
    ]);

    $requestUrl = getBaseUrl() . $apiUri;
    $context_options = createOptions($method, $xaksk, $content);
```

```
    try {
        $response = file_get_contents($requestUrl, false, stream_context_create($context_options)); // 发送请求
        print_r($response . PHP_EOL); // 打印响应结果
    } catch (Exception $e) {
        echo $e->getMessage(); //打印错误信息
    }
}

/**
 * getPlayInfoList
 * @param string $notifyVoice
 * @param string $templateId
 * @param string[] $templateParas
 * @return string[][]
 */
function getPlayInfoList($notifyVoice, $templateId, $templateParas) {
    $playInfoList = [
        [
            'notifyVoice' => $notifyVoice,//通知语音的放音文件名。
            'templateId' => $templateId,//语音通知模板ID，用于唯一标识语音通知模板。
            'templateParas' => $templateParas//语音通知模板的变量值列表，用于依次填充templateId参数指定的
模板内容中的变量。
//             'collectInd' => 0, // 是否进行收号
//             'replayAfterCollection' => 'false', // 设置是否在收号后重新播放notifyVoice或templateId指定的放音
//             'collectContentTriggerReplaying' => '1' // 设置触发重新放音的收号内容
        ]
    ];
    return $playInfoList;
}

$playInfoList = getPlayInfoList('notifyvoice.wav', 'xxxxxx', ['3', '人民公园正门']);
voiceNotifyAPI('+8653159511234', '+8613500000001', $playInfoList);
?>
```

“呼叫状态通知 API”代码样例

```
<?php
/**
 * 呼叫事件通知
 * 客户平台收到RTC业务平台的呼叫事件通知的接口通知
 */
//呼叫事件通知样例
$jsonBody = json_encode([
    'eventType' => 'callout',
    'statusInfo' => [
        'sessionId' => '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp' => '2019-01-24 03:04:24',
        'caller' => '+8613800000022',
        'called' => '+8613800000021'
    ]
]);
print_r($jsonBody . PHP_EOL);
onCallEvent($jsonBody);//呼叫事件处理
/**
 * 呼叫事件通知
 * @desc 详细内容以接口文档为准
 * @param jsonBody
 */
function onCallEvent($jsonBody) {
    $jsonArr = json_decode($jsonBody, true); //将通知消息解析为关联数组
    $eventType = $jsonArr['eventType']; //通知事件类型

    if (strcasecmp($eventType, 'fee') == 0) {
        print_r('EventType error: ' . $eventType);
        return;
    }

    if (!array_key_exists('statusInfo', $jsonArr)) {
        print_r('param error: no statusInfo.');
```

```
    return;
}
$statusInfo = $jsonArr['statusInfo']; //呼叫状态事件信息

print_r('eventType: ' . $eventType . PHP_EOL); //打印通知事件类型
//callout：呼出事件
if (strcasecmp($eventType, 'callout') == 0) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (array_key_exists('sessionId', $statusInfo)) {
        print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
    }
    return;
}
//alerting：振铃事件
if (strcasecmp($eventType, 'alerting') == 0) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (array_key_exists('sessionId', $statusInfo)) {
        print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
    }
    return;
}
//answer：应答事件
if (strcasecmp($eventType, 'answer') == 0) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (array_key_exists('sessionId', $statusInfo)) {
        print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
    }
    return;
}
//collectInfo：放音收号结果事件,仅应用于语音通知场景
if (strcasecmp($eventType, 'collectInfo') == 0) {
    /**
     * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'sessionId': 通话链路的标识ID
     * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
     */
    if (array_key_exists('digitInfo', $statusInfo)) {
        print_r('digitInfo: ' . $statusInfo['digitInfo'] . PHP_EOL);
    }
    return;
}
//disconnect：挂机事件
if (strcasecmp($eventType, 'disconnect') == 0) {
```



```
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
         * 'stateCode': 通话挂机的原因值
         * 'stateDesc': 通话挂机的原因值的描述
         */
        if (array_key_exists('sessionId', $statusInfo)) {
            print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
        }
        return;
    }
}
?>
```

“话单通知 API”代码样例

```
<?php
/**
 * 话单通知
 * 客户平台收到RTC业务平台的话单通知的接口通知
 */
//话单通知样例
$jsonBody = json_encode([
    'eventType' => 'fee',
    'feeLst' => [
        [
            'direction' => 0, //通话的呼叫方向,以RTC业务平台为基准
            'spId' => 'CaaS_Test_01', //客户的云服务账号
            'appKey' => 'ka4kESI5s3YyurL1wpx63s9YnEm2', //应用的app_key
            'icid' => 'CAE-20190124110424-12019775', //呼叫记录的唯一标识
            'bindNum' => '+8675512345678', //发起此次呼叫的CallEnabler业务号码,即绑定号码
            'sessionId' => '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com', //通话链路的唯一标识
            'callerNum' => '+8613800000022', //主叫号码
            'calleeNum' => '+8613800000021', //被叫号码
            'fwdDisplayNum' => '+8613800000022', //转接呼叫时的显示号码(仅语音回呼场景携带)
            'fwdDstNum' => '+8613866887021', //转接呼叫时的转接号码(仅语音回呼场景携带)
            'fwdStartTime' => '2019-01-24 03:04:31', //转接呼叫操作的开始时间(仅语音回呼场景携带)
            'fwdAlertingTime' => '2019-01-24 03:04:36', //转接呼叫操作后的振铃时间(仅语音回呼场景携带)
            'fwdAnswerTime' => '2019-01-24 03:04:38', //转接呼叫操作后的应答时间(仅语音回呼场景携带)
            'callEndTime' => '2019-01-24 03:04:49', //呼叫结束时间
            'fwdUnaswRsn' => 0, //转接呼叫操作失败的Q850原因值
            'failTime' => '', //呼入,呼出的失败时间
            'ulFailReason' => 0, //通话失败的拆线点
            'sipStatusCode' => 0, //呼入,呼出的失败SIP状态码
            'callOutStartTime' => '2019-01-24 03:04:24', //Initcall的呼出开始时间
            'callOutAlertingTime' => '2019-01-24 03:04:27', //Initcall的呼出振铃时间
            'callOutAnswerTime' => '2019-01-24 03:04:31', //Initcall的呼出应答时间
            'callOutUnaswRsn' => 0, //Initcall的呼出失败的Q850原因值
            'dynIVRStartTime' => '', #自定义动态IVR开始时间(仅语音通知场景携带)
            'dynIVRPath' => '', //自定义动态IVR按键路径(仅语音通知场景携带)
            'recordFlag' => 0, //录音标识
            'recordStartTime' => '', //录音开始时间(仅语音回呼场景携带)
            'recordObjectName' => '', //录音文件名(仅语音回呼场景携带)
            'recordBucketName' => '', //录音文件所在的目录名(仅语音回呼场景携带)
            'recordDomain' => '', //存放录音文件的域名(仅语音回呼场景携带)
            'recordFileDownloadUrl' => '', //录音文件下载地址(仅语音回呼场景携带)
            'ttsPlayTimes' => 0, //应用TTS功能时,使用TTS的总次数
            'ttsTransDuration' => 0, //应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
            'serviceType' => '002', //携带呼叫的业务类型信息
            'hostName' => 'callenabler245.huaweicaas.com', //话单生成的服务器设备对应的主机名
            'userData' => '' //用户附属信息
        ]
```

```
    ]
]);
print_r($jsonBody . PHP_EOL);
onFeeEvent($jsonBody);//话单处理
/**
 * 话单通知
 * @desc 详细内容以接口文档为准
 * @param jsonBody
 */
function onFeeEvent($jsonBody) {
    $jsonArr = json_decode($jsonBody, true); //将通知消息解析为关联数组
    $eventType = $jsonArr['eventType']; //通知事件类型

    if (strcasecmp($eventType, 'fee') != 0) {
        print_r('EventType error: ' . $eventType);
        return;
    }

    if (!array_key_exists('feeLst', $jsonArr)) {
        print_r('param error: no feeLst.');
        return;
    }

    $feeLst = $jsonArr['feeLst']; //呼叫话单事件信息

    //Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
    //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
    if (sizeof($feeLst) > 1) {
        foreach ($feeLst as $loop){
            if (array_key_exists('sessionId', $loop)) {
                print_r('sessionId: ' . $loop['sessionId'] . PHP_EOL);
            }
        }
    } else if(sizeof($feeLst) == 1) {
        if (array_key_exists('sessionId', $feeLst[0])) {
            print_r('sessionId: ' . $feeLst[0]['sessionId'] . PHP_EOL);
        }
    } else {
        print_r('feeLst error: no element.');
    }
}
?>
```

4.2.5 C#

样例	**语音通知API**、**呼叫状态通知API**、**话单通知API**
环境要求	.NET Core 2.0及以上版本或.NET Framework 4.7.1及以上版本。
引用库	Newtonsoft.Json 11.0.2，请参考**https://www.newtonsoft.com/json**获取。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音通知 API”代码样例

```
using Newtonsoft.Json;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Security.Cryptography;
using System.Text;

namespace voicecall_csharp_demo_x_aksk_
{
    class VoiceNotify
    {
        string base_url = "https://{domain}:{port}"; //APP接入地址，购买服务时下发,，请替换为实际值
        string appKey = "***appKey***"; //语音通知应用的appKey,购买服务时下发,请替换为实际值
        string appSecret = "***appSecret***"; //语音通知应用的appSecret,购买服务时下发,请替换为实际值

        static void Main(string[] args)
        {
            //构造放音列表,此处取值仅为样例,请替换为实际值
            ArrayList playInfoList = getPlayInfoList("notifyvoice.wav", "xxxxxx", ["3", "人民公园正门"]);
            //固话号码从控制台号码管理页获取,被叫号码请替换为实际号码.
            voiceNotifyAPI("+86531*******4", "+86135*******1", playInfoList);
        }

        static void voiceNotifyAPI(string displayNbr, string calleeNbr, ArrayList playInfoList)
        {
            if (String.IsNullOrEmpty(displayNbr) || String.IsNullOrEmpty(calleeNbr) || Count(playInfoList) < 1)
            {
                return;
            }

            string apiURI = "/rest/httpsessions/callnotify/v2.0"; //接口URI, v1.0 or v2.0
            string requestUrl = base_url + apiURI;

            try
            {
                //为防止因HTTPS证书认证失败造成API调用失败,需要先忽略证书信任问题
                //.NET Framework 4.7.1及以上版本,请采用如下代码
                var sslHandler = new HttpClientHandler()
                {
                    ServerCertificateCustomValidationCallback = (message, cert, chain, err) => { return true; }
                };
                HttpClient client = new HttpClient(sslHandler, true);
                //低于.NET Framework 4.7.1版本,请采用如下代码
                //HttpClient client = new HttpClient();
                //ServicePointManager.ServerCertificateValidationCallback = new
RemoteCertificateValidationCallback(CheckValidationResult);

                //请求Headers
                client.DefaultRequestHeaders.Add("Authorization", "AKSK realm=\"SDP\",profile=
\"UsernameToken\",type=\"Appkey\"");
                client.DefaultRequestHeaders.Add("X-AKSK", buildAKSKHeader(appKey, appSecret));

                //请求Body
                var body = new Dictionary<string, object>() {
                    /*必填参数*/
                    {"displayNbr", displayNbr},//主叫用户手机终端的来电显示号码。
                    {"calleeNbr", calleeNbr},//发起呼叫时所拨打的被叫号码。
                    {"playInfoList", playInfoList}//播放信息列表，最大支持5个，每个播放信息携带的参数都可以不相
同。
                    /*选填参数*/

                    //{"statusUrl", ""}, //设置SP接收状态上报的URL,要求使用BASE64编码
                    //{"feeUrl", ""}, //设置SP接收话单上报的URL,要求使用BASE64编码
```

```
                //{"returnIdlePort", "false"}, //指示是否需要返回平台空闲呼叫端口数量
                //{"userData", "customerId123"} //设置用户的附属信息
            };

            HttpContent content = new StringContent(JsonConvert.SerializeObject(body));
            //请求Headers中的Content-Type参数
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            var response = client.PostAsync(requestUrl, content).Result;
            Console.WriteLine(response.StatusCode);
            var res = response.Content.ReadAsStringAsync().Result;
            Console.WriteLine(res); //打印响应结果
        }
        catch (Exception e)
        {
            Console.WriteLine(e.StackTrace);
            Console.WriteLine(e.Message); //打印错误信息
        }
    }

    static string buildAKSKHeader(string appKey, string appSecret)
    {
        string now = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ"); //Created
        string nonce = Guid.NewGuid().ToString().Replace("-", ""); //Nonce
        String str = nonce + now;

        byte[] keyByte = Encoding.UTF8.GetBytes(appSecret);
        byte[] messageBytes = Encoding.UTF8.GetBytes(str);
        using (var hmacsha256 = new HMACSHA256(keyByte))
        {
            byte[] hashmessage = hmacsha256.ComputeHash(messageBytes);
            string base64 = Convert.ToBase64String(hashmessage);
            return String.Format("UsernameToken Username=\"{0}\",PasswordDigest=\"{1}\",Nonce=
\"{2}\",Created=\"{3}\"",
                appKey, base64, nonce, now);
        }
    }

    static ArrayList getPlayInfoList(string notifyVoice, string templateId, string[] templateParas)
    {
        ArrayList playInfoList = new ArrayList(); //播放信息列表,最大支持5个

        /*此处以构建一个playInfoDic,添加到playInfoList为例,请按需构建多个*/
        Dictionary<string, object> playInfoDic = new Dictionary<string, object>();
        playInfoDic.Add("notifyVoice", notifyVoice);
        playInfoDic.Add("templateId", templateId);
        playInfoDic.Add("templateParas", templateParas);
        //playInfoDic.Add("collectInd", 0); //是否进行收号
        //playInfoDic.Add("replayAfterCollection", "false"); //设置是否在收号后重新播放notifyVoice或
templateId指定的放音
        //playInfoDic.Add("collectContentTriggerReplaying", "1"); //设置触发重新放音的收号内容
        playInfoList.Add(playInfoDic); //playInfoList添加元素

        return playInfoList;
    }

    //低于.NET Framework 4.7.1版本,启用如下方法
    //static bool CheckValidationResult(object sender, X509Certificate certificate, X509Chain chain,
SslPolicyErrors errors)
    //{
    //    return true;
    //}
  }
}
```

“呼叫状态通知 API”代码样例

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
```

```
using System;
using System.Collections.Generic;

namespace voicecall_csharp_demo_x_aksk_
{
    class CallEventImpl
    {
        static void Main(string[] args)
        {
            //呼叫事件通知样例
            string jsonBody = JsonConvert.SerializeObject(new Dictionary<string, object>(){
                {"eventType", "callout"},
                {"statusInfo", new Dictionary<string, object>(){
                        {"sessionId", "1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com"},
                        {"timestamp", "2019-01-24 03:04:24"},
                        {"caller", "+86138*******2"},
                        {"called", "+86138*******1"},
                    }
                }
            });

            Console.WriteLine("jsonBody:" + jsonBody);

            OnCallEvent(jsonBody); //呼叫事件处理
        }

        /// <summary>
        /// 呼叫事件通知,详细内容以接口文档为准
        /// </summary>
        /// <param name="jsonBody"></param>
        static void OnCallEvent(string jsonBody)
        {
            JObject jsonObj = (JObject)JsonConvert.DeserializeObject(jsonBody); //将通知消息解析为jsonObj
            string eventType = jsonObj["eventType"].ToString(); //通知事件类型

            if ("fee".Equals(eventType))
            {
                Console.WriteLine("EventType error:" + eventType);
                return;
            }

            if (!jsonObj.ContainsKey("statusInfo"))
            {
                Console.WriteLine("param error: no statusInfo.");
                return;
            }
            JObject statusInfo = (JObject)jsonObj["statusInfo"]; //呼叫状态事件信息

            Console.WriteLine("eventType:" + eventType); //打印通知事件类型

            //callout：呼出事件
            if ("callout".Equals(eventType))
            {
                /**
                 * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
                 *
                 * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
                 * 'userData': 用户附属信息
                 * 'sessionId': 通话链路的标识ID
                 * 'caller': 主叫号码
                 * 'called': 被叫号码
                 */
                if (statusInfo.ContainsKey("sessionId"))
                {
                    Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
                }
                return;
            }
            //alerting：振铃事件
```

```
if ("alerting".Equals(eventType))
{
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.ContainsKey("sessionId"))
    {
        Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
    }
    return;
}
//answer：应答事件
if ("answer".Equals(eventType))
{
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.ContainsKey("sessionId"))
    {
        Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
    }
    return;
}
//collectInfo：放音收号结果事件,仅应用于语音通知场景
if ("collectInfo".Equals(eventType))
{
    /**
     * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
     * 'sessionId': 通话链路的标识ID
     * 'digitInfo': 放音收号场景中,语音通话平台对开发者进行放音收号操作的结果描述
     */
    if (statusInfo.ContainsKey("digitInfo"))
    {
        Console.WriteLine("digitInfo:" + statusInfo["digitInfo"]);
    }
    return;
}
//disconnect：挂机事件
if ("disconnect".Equals(eventType))
{
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
     * 'stateCode': 通话挂机的原因值
     * 'stateDesc': 通话挂机的原因值的描述
     */
    if (statusInfo.ContainsKey("sessionId"))
    {
        Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
```

```
            }
            return;
        }
    }
  }
}
```

“话单通知 API”代码样例

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;

namespace voicecall_csharp_demo_x_aksk_
{
   class FeeImpl
   {
      static void Main(string[] args)
      {
         //话单通知样例
         string jsonBody = JsonConvert.SerializeObject(new Dictionary<string, object>(){
            {"eventType", "fee"},
            {"feeLst",  new object[] {
               new Dictionary<string, object>(){
                     {"direction", 0}, //通话的呼叫方向,以语音通话平台为基准
                     {"spId", "CaaS_Test_01"}, //客户的云服务账号
                     {"appKey", "ka4k*****Em2"}, //应用的app_key，购买服务时下发,请替换为实际值
                     {"icid", "CAE-20190124110424-12019775"}, //呼叫记录的唯一标识
                     {"bindNum", "+8675512****78"}, //发起此次呼叫的CallEnabler业务号码,即绑定号码
                     {"sessionId",
"1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com"}, //通话链路的唯一标识
                     {"callerNum", "+86138****0022"}, //主叫号码
                     {"calleeNum", "+86138****0021"}, //被叫号码
                     {"fwdDisplayNum", "+86138****0022"}, //转接呼叫时的显示号码(仅语音回呼场景携带)
                     {"fwdDstNum", "+86138****7021"}, //转接呼叫时的转接号码(仅语音回呼场景携带)

                     {"fwdStartTime", "2019-01-24 03:04:31"}, //转接呼叫操作的开始时间(仅语音回呼场景携带)
                     {"fwdAlertingTime", "2019-01-24 03:04:36"}, //转接呼叫操作后的振铃时间(仅语音回呼场景
携带)
                     {"fwdAnswerTime", "2019-01-24 03:04:38"}, //转接呼叫操作后的应答时间(仅语音回呼场景
携带)
                     {"callEndTime", "2019-01-24 03:04:49"}, //呼叫结束时间
                     {"fwdUnaswRsn", 0}, //转接呼叫操作失败的Q850原因值
                     {"failTime", ""}, //呼入,呼出的失败时间
                     {"ulFailReason", 0}, //通话失败的拆线点
                     {"sipStatusCode", 0}, //呼入,呼出的失败SIP状态码
                     {"callOutStartTime", "2019-01-24 03:04:24"}, //Initcall的呼出开始时间
                     {"callOutAlertingTime", "2019-01-24 03:04:27"}, //Initcall的呼出振铃时间
                     {"callOutAnswerTime", "2019-01-24 03:04:31"}, //Initcall的呼出应答时间
                     {"callOutUnaswRsn", 0}, //Initcall的呼出失败的Q850原因值
                     {"dynIVRStartTime", ""}, //自定义动态IVR开始时间(仅语音通知场景携带)
                     {"dynIVRPath", ""}, //自定义动态IVR按键路径(仅语音通知场景携带)
                     {"recordFlag", 0}, //录音标识
                     {"recordStartTime", ""}, //录音开始时间(仅语音回呼场景携带)
                     {"recordObjectName", ""}, //录音文件名(仅语音回呼场景携带)
                     {"recordBucketName", ""}, //录音文件所在的目录名(仅语音回呼场景携带)
                     {"recordDomain", ""}, //存放录音文件的域名(仅语音回呼场景携带)
                     {"recordFileDownloadUrl", ""}, //录音文件下载地址(仅语音回呼场景携带)
                     {"ttsPlayTimes", 0}, //应用TTS功能时,使用TTS的总次数
                     {"ttsTransDuration", 0}, //应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
                     {"serviceType", "002"}, //携带呼叫的业务类型信息
                     {"hostName", "callenabler245.huaweicaas.com"}, //话单生成的服务器设备对应的主机名
                     {"userData", "customerId123"} //用户附属信息
                  }
               }
            }
         });
```

```
            Console.WriteLine(jsonBody);

            OnFeeEvent(jsonBody); //话单处理
        }

        /// <summary>
        /// 话单通知,详细内容以接口文档为准
        /// </summary>
        /// <param name="jsonBody"></param>
        static void OnFeeEvent(string jsonBody)
        {
            JObject jsonObj = (JObject)JsonConvert.DeserializeObject(jsonBody); //将通知消息解析为jsonObj
            string eventType = jsonObj["eventType"].ToString(); //通知事件类型

            if (!"fee".Equals(eventType))
            {
                Console.WriteLine("EventType error:" + eventType);
                return;
            }

            if (!jsonObj.ContainsKey("feeLst"))
            {
                Console.WriteLine("param error: no feeLst.");
                return;
            }
            JObject[] feeLst = jsonObj["feeLst"].ToObject<JObject[]>(); //呼叫话单事件信息

            //Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
            //短时间内有多个通话结束时语音通话平台会将话单合并推送,每条消息最多携带50个话单
            if (feeLst.Length > 1)
            {
                foreach (JObject loop in feeLst)
                {
                    if (loop.ContainsKey("sessionId"))
                    {
                        Console.WriteLine("sessionId:" + loop["sessionId"]);
                    }
                }
            }
            else if (feeLst.Length == 1)
            {
                if (feeLst[0].ContainsKey("sessionId"))
                {
                    Console.WriteLine("sessionId:" + feeLst[0]["sessionId"]);
                }
            }
            else
            {
                Console.WriteLine("feeLst error: no element.");
            }
        }
    }
}
```

4.3 语音验证码代码样例

4.3.1 Node.js

注：为节省开发时间，建议先使用Node.js代码样例进行调测，熟悉接口使用后，再参考Java、python、PHP或C#代码样例，结合**接口文档**进行功能开发。

样例	**语音验证码场景API**、**呼叫状态通知API**、**话单通知API**
环境要求	Node.js 4.4.4及以上版本。

引用库	-

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音验证码场景 API”代码样例

```
/*jshint esversion: 6 */
var https = require('https');
var data = require('./data.js');
var util = require('./reqUtil.js');

/**
 * voiceVerificationCodeAPI
 * @param displayNbr
 * @param calleeNbr
 * @param languageType
 * @param preTone
 * @param verifyCode
 * @returns
 */
function voiceVerificationCodeAPI(displayNbr, calleeNbr, languageType, preTone, verifyCode) {
    if(displayNbr === undefined || displayNbr === null || calleeNbr === undefined || calleeNbr === null) {
        return;
    }
    if(languageType === undefined || languageType === null || preTone === undefined || preTone === null) {
        return;
    }
    if(languageType === verifyCode || languageType === verifyCode) {
        return;
    }

    var method = 'POST';
    var uri = '/rest/httpsessions/callVerify/v1.0';
    var xaksk = util.buildAKSKHeader(data.data.callverify_appid, data.data.callverify_secret);

    var options = util.createOptions(method, uri, null, xaksk);

    var body = {
            /* 必填参数 */
            'displayNbr': displayNbr,//主叫用户手机终端的来电显示号码。
            'calleeNbr': calleeNbr,//被叫用户终端的来电显示号码。
            'languageType': languageType,//验证码播放的语言类型。
            'preTone': preTone,//播放语音验证码之前需要播放的放音文件名
            'verifyCode': verifyCode//验证码：只支持0～9的数字，最大8位。
            /* 选填参数 */
//            'bindNbr': '+86123456789', //CallEnabler业务号码,即绑定号码
//            'posTone': 'postone.wav', //播放语音验证码之后需要播放的放音文件名
//            'times': 3, //播放次数:0～9
//            'statusUrl': 'aHR0cDovLzIxOC40LjMzLjU1Ojg4ODgvdGVzdA==', //要获取通话状态需要在请求中加入
statusUrl
//            'feeUrl': 'aHR0cDovLzIxOC40LjMzLjU1Ojg4ODgvdGVzdA==', //要获取话单需要在请求中加入feeUrl
//            'returnIdlePort': 'false', //指示是否需要返回平台空闲呼叫端口数量
//            'userData': 'customerId123' //设置用户的附属信息
    };
    var req = https.request(options, function (res) {
        var resHeaders = JSON.stringify(res.headers);
```



```
        res.setEncoding('utf8');
        res.on('data', function (chunk) {
            console.log(chunk);
        });
    });

    req.on('error', function(e) {
        console.error('problem with request: ' + e);
    });
    console.log(JSON.stringify(body));
    req.write(JSON.stringify(body));
    req.end();
}

voiceVerificationCodeAPI('+8653159511234', '+8613500000001', 2, 'test.wav', '1234');
```

“呼叫状态通知 API”代码样例

```
/**
 * 呼叫事件通知
 * 客户平台收到RTC业务平台的呼叫事件通知的接口通知
 */
//呼叫事件通知样例
var jsonBody = JSON.stringify({
    'eventType': 'callout',
    'statusInfo': {
        'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp': '2019-01-24 03:04:24',
        'caller': '+8613800000022',
        'called': '+8613800000021'
    }
});
console.log('jsonBody:', jsonBody);
/**
 * 呼叫事件通知
 * @brief 详细内容以接口文档为准
 * @param jsonBody
 */
function onCallEvent(jsonBody) {
    var jsonObj = JSON.parse(jsonBody); //将通知消息解析为jsonObj
    var eventType = jsonObj.eventType; //通知事件类型

    if ('fee' === eventType) {
        console.log('EventType error:', eventType);
        return;
    }

    if (!jsonObj.hasOwnProperty('statusInfo')) {
        console.log('param error: no statusInfo.');
        return;
    }
    var statusInfo = jsonObj.statusInfo; //呼叫状态事件信息

    console.log('eventType:', eventType); //打印通知事件类型

    //callout：呼出事件
    if ('callout' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
```



```
        return;
    }
    //alerting：振铃事件
    if ('alerting' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
    //answer：应答事件
    if ('answer' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
    //collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ('collectInfo' === eventType) {
        /**
         * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'sessionId': 通话链路的标识ID
         * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
         */
        if (statusInfo.hasOwnProperty('digitInfo')) {
            console.log('digitInfo:', statusInfo.digitInfo);
        }
        return;
    }
    //disconnect：挂机事件
    if ('disconnect' === eventType) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
         * 'stateCode': 通话挂机的原因值
         * 'stateDesc': 通话挂机的原因值的描述
         */
        if (statusInfo.hasOwnProperty('sessionId')) {
            console.log('sessionId:', statusInfo.sessionId);
        }
        return;
    }
}
```

```
//呼叫事件处理
onCallEvent(jsonBody);
```

“话单通知 API”代码样例

```
/**
 * 话单通知
 * 客户平台收到RTC业务平台的话单通知的接口通知
 */
//话单通知样例
var jsonBody = JSON.stringify({
    'eventType': 'fee',
    'feeLst': [
        {
            'direction': 0,
            'spId': 'CaaS_Test_01',
            'appKey': 'ka4kESl5s3YyurL1wpx63s9YnEm2',
            'icid': 'CAE-20190124110424-12019775',
            'bindNum': '+8675512345678',
            'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
            'callerNum': '+8613800000022',
            'calleeNum': '+8613800000021',
            'fwdDisplayNum': '+8613800000022',
            'fwdDstNum': '+8613866887021',
            'fwdStartTime': '2019-01-24 03:04:31',
            'fwdAlertingTime': '2019-01-24 03:04:36',
            'fwdAnswerTime': '2019-01-24 03:04:38',
            'callEndTime': '2019-01-24 03:04:49',
            'fwdUnaswRsn': 0,
            'ulFailReason': 0,
            'sipStatusCode': 0,
            'callOutStartTime': '2019-01-24 03:04:24',
            'callOutAlertingTime': '2019-01-24 03:04:27',
            'callOutAnswerTime': '2019-01-24 03:04:31',
            'callOutUnaswRsn': 0,
            'recordFlag': 0,
            'ttsPlayTimes': 0,
            'ttsTransDuration': 0,
            'serviceType': '002',
            'hostName': 'callenabler245.huaweicaas.com'
        }
    ]
});
console.log('jsonBody:', jsonBody);
/**
 * 话单通知
 * @brief 详细内容以接口文档为准
 * @param jsonBody
 */
function onFeeEvent(jsonBody) {
    var jsonObj = JSON.parse(jsonBody); //将通知消息解析为jsonObj
    var eventType = jsonObj.eventType; //通知事件类型

    if ('fee' !== eventType) {
        console.log('EventType error:', eventType);
        return;
    }

    if (!jsonObj.hasOwnProperty('feeLst')) {
        console.log('param error: no feeLst.');
        return;
    }
    var feeLst = jsonObj.feeLst; //呼叫话单事件信息
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'direction': 通话的呼叫方向,以RTC业务平台为基准
     * 'spId': 客户的云服务账号
     * 'appKey': 应用的app_key
```

```
    * 'icid': 呼叫记录的唯一标识
    * 'bindNum': 发起此次呼叫的CallEnabler业务号码
    * 'sessionId': 通话链路的唯一标识
    * 'callerNum': 主叫号码
    * 'calleeNum': 被叫号码
    * 'fwdDisplayNum': 转接呼叫时的显示号码(仅语音回呼场景携带)
    * 'fwdDstNum': 转接呼叫时的转接号码(仅语音回呼场景携带)
    * 'fwdStartTime': 转接呼叫操作的开始时间(仅语音回呼场景携带)
    * 'fwdAlertingTime': 转接呼叫操作后的振铃时间(仅语音回呼场景携带)
    * 'fwdAnswerTime': 转接呼叫操作后的应答时间(仅语音回呼场景携带)
    * 'callEndTime': 呼叫结束时间
    * 'fwdUnaswRsn': 转接呼叫操作失败的Q850原因值
    * 'failTime': 呼入,呼出的失败时间
    * 'ulFailReason': 通话失败的拆线点
    * 'sipStatusCode': 呼入,呼出的失败SIP状态码
    * 'callOutStartTime': Initcall的呼出开始时间
    * 'callOutAlertingTime': Initcall的呼出振铃时间
    * 'callOutAnswerTime': Initcall的呼出应答时间
    * 'callOutUnaswRsn': Initcall的呼出失败的Q850原因值
    * 'dynIVRStartTime': 自定义动态IVR开始时间(仅语音通知场景携带)
    * 'dynIVRPath': 自定义动态IVR按键路径(仅语音通知场景携带)
    * 'recordFlag': 录音标识
    * 'recordStartTime': 录音开始时间(仅语音回呼场景携带)
    * 'recordObjectName': 录音文件名(仅语音回呼场景携带)
    * 'recordBucketName': 录音文件所在的目录名(仅语音回呼场景携带)
    * 'recordDomain': 存放录音文件的域名(仅语音回呼场景携带)
    * 'recordFileDownloadUrl': 录音文件下载地址(仅语音回呼场景携带)
    * 'ttsPlayTimes': 应用TTS功能时,使用TTS的总次数
    * 'ttsTransDuration': 应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
    * 'serviceType': 携带呼叫的业务类型信息
    * 'hostName': 话单生成的服务器设备对应的主机名
    * 'userData': 用户附属信息
    */
   //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
   if (feeLst.length > 1) {
       for ( var i=0; i<feeLst.length; i++) {
           if (feeLst[i].hasOwnProperty('sessionId')) {
               console.log('sessionId:', feeLst[i].sessionId);
           }
       }
   } else if (feeLst.length === 1) {
       if (feeLst[0].hasOwnProperty('sessionId')) {
           console.log('sessionId:', feeLst[0].sessionId);
       }
   } else {
       console.log('feeLst error: no element.');
   }
}
//话单处理
onFeeEvent(jsonBody);
```

4.3.2 Java

4.3.2.1 公共要求

注：使用前请务必先仔细阅读使用**注意事项**。

样例	语音验证码场景API、获取录音文件下载地址API、呼叫状态与话单通知AP
环境要求	JDK 1.6及以上版本。

引用库	httpclient、httpcore、httpmime、commons-codec、commons-logging、jackson-databind、jackson-annotations、jackson-core、fastjson、log4j

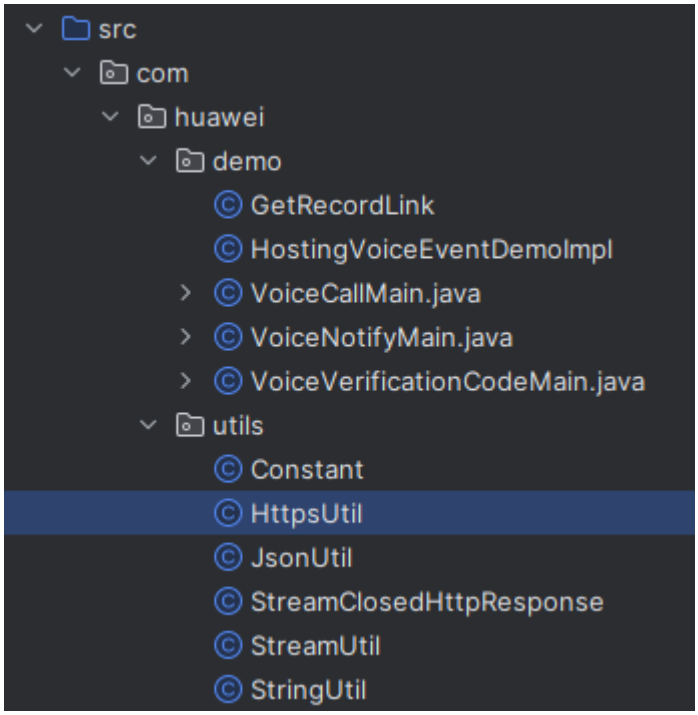
将下列代码样例复制到新建java文件中即可运行。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

公共依赖

使用Java样例时请参考以下工程目录结构：



Constant.java

```
/*
* File Name: com.huawei.utils.Constant.java
*
* Copyright Notice:
*      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
*
*      Warning: This computer software sourcecode is protected by copyright law
*      and international treaties. Unauthorized reproduction or distribution
*      of this sourcecode, or any portion of it, may result in severe civil and
*      criminal penalties, and will be prosecuted to the maximum extent
```



```
 *      possible under the law.
 */
package com.huawei.utils;

public class Constant {

    // purchase and get : base_url,appid,secret

    // please replace the IP and Port, when you use the demo.
    public static final String BASE_URL = "https://{domain}:{port}";

    // please replace the appId and secret, when you use the demo.
    public static final String CLICK2CALL_APPID = "*****";
    public static final String CLICK2CALL_SECRET = "*****";
    public static final String CALLNOTIFY_APPID = "*****";
    public static final String CALLNOTIFY_SECRET = "*****";
    public static final String CALLVERIFY_APPID = "*****";
    public static final String CALLVERIFY_SECRET = "*****";

    // ************************** The following constants do not need to be
    // modified *******************************//

    /*
     * request header
     * 1. HEADER_APP_AUTH
     * 2. HEADER_APP_AKSK
     * 3. AUTH_HEADER_VALUE
     * 4. AKSK_HEADER_FORMAT
     */
    public static final String HEADER_APP_AUTH = "Authorization";
    public static final String HEADER_APP_AKSK = "X-AKSK";
    public static final String AUTH_HEADER_VALUE = "AKSK realm=\"SDP\",profile=\"UsernameToken\",type=
\"Appkey\"";
    public static final String AKSK_HEADER_FORMAT = "UsernameToken Username=\"%s\",PasswordDigest=
\"%s\",Nonce=\"%s\",Created=\"%s\"";

    /*
     * Voice Call:
     * 1. VOICE_CALL_COMERCIAL
     * 2. VOICE_VERIFICATION_COMERCIAL
     * 3. CALL_NOTIFY_COMERCIAL
     * 4. VOICE_FILE_DOWNLOAD
     */
    public static final String VOICE_CALL_COMERCIAL = BASE_URL + "/rest/httpsessions/click2Call/v2.0";
    public static final String VOICE_VERIFICATION_COMERCIAL = BASE_URL + "/rest/httpsessions/callVerify/
v1.0";
    public static final String CALL_NOTIFY_COMERCIAL = BASE_URL + "/rest/httpsessions/callnotify/v2.0";
    public static final String VOICE_FILE_DOWNLOAD = BASE_URL + "/rest/provision/voice/record/v1.0";
}
```

HttpsUtil.java

```
package com.huawei.utils;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.SSLContextBuilder;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.client.HttpClients;
```



```
import java.io.IOException;
import java.net.URISyntaxException;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.List;
import java.util.Map;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

@SuppressWarnings("deprecation")
public class HttpsUtil extends DefaultHttpClient {
    public final static String CONTENT_LENGTH = "Content-Length";

    private static CloseableHttpClient httpClient = getHttpClient();

    public static CloseableHttpClient getHttpClient() {
        SSLContext sslcontext = null;
        try {
            sslcontext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {

                @Override
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (KeyManagementException e) {
            return null;
        } catch (NoSuchAlgorithmException e) {
            return null;
        } catch (KeyStoreException e) {
            return null;
        }

        // Allow TLSv1, TLSv1.1, TLSv1.2 protocol only
        SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslcontext,
            new String[] {"TLSv1", "TLSv1.1", "TLSv1.2"}, null,
            SSLConnectionSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);

        CloseableHttpClient httpclient = HttpClients.custom()
            .setDefaultRequestConfig(RequestConfig.custom().setRedirectsEnabled(false).build())
            .setSSLSocketFactory(sslsf).build();

        return httpclient;
    }

    public HostnameVerifier hv = new HostnameVerifier() {
        public boolean verify(String urlHostName, SSLSession session) {
            return true;
        }
    };

    public void trustAllHttpsCertificates() throws Exception {
        TrustManager[] trustAllCerts = new TrustManager[1];
        TrustManager tm = new miTM();
        trustAllCerts[0] = tm;
        SSLContext sc = SSLContext.getInstance("SSL");
        sc.init(null, trustAllCerts, null);
        HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
    }
```

```
    static class miTM implements TrustManager, X509TrustManager {
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
        }

        public boolean isServerTrusted(X509Certificate[] certs) {
            return true;
        }

        public boolean isClientTrusted(X509Certificate[] certs) {
            return true;
        }

        public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String authType)
            throws java.security.cert.CertificateException {
            return;
        }

        public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String authType)
            throws java.security.cert.CertificateException {
            return;
        }
    }

    public StreamClosedHttpResponse doPostJsonGetStatusLine(String url, Map<String, String> headerMap,
String content) {
        HttpPost request = new HttpPost(url);
        addRequestHeader(request, headerMap);

        request.setEntity(new StringEntity(content, ContentType.APPLICATION_JSON));

        HttpResponse response = executeHttpRequest(request);
        if (null == response) {
            System.out.println("The response body is null.");
        }

        return (StreamClosedHttpResponse) response;
    }

    public HttpResponse doGetWithParas(String url, List<NameValuePair> queryParams, Map<String, String>
headerMap)
        throws Exception {
        HttpGet request = new HttpGet();
        addRequestHeader(request, headerMap);

        URIBuilder builder;
        try {
            builder = new URIBuilder(url);
        } catch (URISyntaxException e) {
            System.out.printf("URISyntaxException: {}", e);
            throw new Exception(e);
        }

        if (queryParams != null && !queryParams.isEmpty()) {
            builder.setParameters(queryParams);
        }
        request.setURI(builder.build());

        return executeHttpRequest(request);
    }

    public StreamClosedHttpResponse doGetWithParasGetStatusLine(String url, List<NameValuePair>
queryParams,
        Map<String, String> headerMap) throws Exception {
        HttpResponse response = doGetWithParas(url, queryParams, headerMap);
        if (null == response) {
            System.out.println("The response body is null.");
        }
```

```
        return (StreamClosedHttpResponse) response;
    }

    private static void addRequestHeader(HttpUriRequest request, Map<String, String> headerMap) {
        if (headerMap == null) {
            return;
        }

        for (String headerName : headerMap.keySet()) {
            if (CONTENT_LENGTH.equalsIgnoreCase(headerName)) {
                continue;
            }

            String headerValue = headerMap.get(headerName);
            request.addHeader(headerName, headerValue);
        }
    }

    private HttpResponse executeHttpRequest(HttpUriRequest request) {
        HttpResponse response = null;

        try {
            response = httpClient.execute(request);
        } catch (Exception e) {
            System.out.println("executeHttpRequest failed.");
            System.out.println(e.getStackTrace());
        } finally {
            try {
                response = new StreamClosedHttpResponse(response);
            } catch (IOException e) {
                System.out.println("IOException: " + e.getMessage());
            }
        }

        return response;
    }
}
```

JsonUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */

package com.huawei.utils;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;

import java.io.IOException;

public class JsonUtil {

    private static ObjectMapper objectMapper;

    static {
        objectMapper = new ObjectMapper();

        // 设置FAIL_ON_EMPTY_BEANS属性,当序列化空对象不要抛异常
        objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, false);
```



```
        // 设置FAIL_ON_UNKNOWN_PROPERTIES属性,当JSON字符串中存在Java对象没有的属性,忽略
        objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
    }

    /**
     * Convert Object to JsonString
     *
     * @param jsonObj
     * @return
     */
    public static String jsonObj2Sting(Object jsonObj) {
        String jsonString = null;

        try {
            jsonString = objectMapper.writeValueAsString(jsonObj);
        } catch (IOException e) {
            System.out.printf("pasre json Object[{}] to string failed.", jsonString);
        }

        return jsonString;
    }

    /**
     * Convert JsonString to Simple Object
     *
     * @param jsonString
     * @param cls
     * @return
     */
    public static <T> T jsonString2SimpleObj(String jsonString, Class<T> cls) {
        T jsonObj = null;

        try {
            jsonObj = objectMapper.readValue(jsonString, cls);
        } catch (IOException e) {
            System.out.printf("pasre json Object[{}] to string failed.", jsonString);
        }

        return jsonObj;
    }
}
```

StreamClosedHttpResponse.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

import java.io.IOException;
import java.util.Locale;

import org.apache.http.Header;
import org.apache.http.HeaderIterator;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.ProtocolVersion;
import org.apache.http.StatusLine;
import org.apache.http.params.HttpParams;

@SuppressWarnings("deprecation")
public class StreamClosedHttpResponse implements HttpResponse {
```



```
    private final HttpResponse original;

    private final String content;

    public StreamClosedHttpResponse(final HttpResponse original) throws UnsupportedOperationException,
IOException {
        this.original = original;

        HttpEntity entity = original.getEntity();
        if (entity != null && entity.isStreaming()) {
            String encoding = entity.getContentEncoding() != null ? entity.getContentEncoding().getValue() :
null;
            content = StreamUtil.inputStream2String(entity.getContent(), encoding);
        } else {
            content = null;
        }
    }

    @Override
    public StatusLine getStatusLine() {
        return original.getStatusLine();
    }

    @Override
    public void setStatusLine(final StatusLine statusline) {
        original.setStatusLine(statusline);
    }

    @Override
    public void setStatusLine(final ProtocolVersion ver, final int code) {
        original.setStatusLine(ver, code);
    }

    @Override
    public void setStatusLine(final ProtocolVersion ver, final int code, final String reason) {
        original.setStatusLine(ver, code, reason);
    }

    @Override
    public void setStatusCode(final int code) throws IllegalStateException {
        original.setStatusCode(code);
    }

    @Override
    public void setReasonPhrase(final String reason) throws IllegalStateException {
        original.setReasonPhrase(reason);
    }

    @Override
    public HttpEntity getEntity() {
        return original.getEntity();
    }

    @Override
    public void setEntity(final HttpEntity entity) {
        original.setEntity(entity);
    }

    @Override
    public Locale getLocale() {
        return original.getLocale();
    }

    @Override
    public void setLocale(final Locale loc) {
        original.setLocale(loc);
    }
```

```
@Override
public ProtocolVersion getProtocolVersion() {
    return original.getProtocolVersion();
}

@Override
public boolean containsHeader(final String name) {
    return original.containsHeader(name);
}

@Override
public Header[] getHeaders(final String name) {
    return original.getHeaders(name);
}

@Override
public Header getFirstHeader(final String name) {
    return original.getFirstHeader(name);
}

@Override
public Header getLastHeader(final String name) {
    return original.getLastHeader(name);
}

@Override
public Header[] getAllHeaders() {
    return original.getAllHeaders();
}

@Override
public void addHeader(final Header header) {
    original.addHeader(header);
}

@Override
public void addHeader(final String name, final String value) {
    original.addHeader(name, value);
}

@Override
public void setHeader(final Header header) {
    original.setHeader(header);
}

@Override
public void setHeader(final String name, final String value) {
    original.setHeader(name, value);
}

@Override
public void setHeaders(final Header[] headers) {
    original.setHeaders(headers);
}

@Override
public void removeHeader(final Header header) {
    original.removeHeader(header);
}

@Override
public void removeHeaders(final String name) {
    original.removeHeaders(name);
}

@Override
public HeaderIterator headerIterator() {
    return original.headerIterator();
}
```

```
    @Override
    public HeaderIterator headerIterator(final String name) {
        return original.headerIterator(name);
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("HttpResponseProxy{");
        sb.append(original);
        sb.append('}');
        return sb.toString();
    }

    @Override
    @Deprecated
    public HttpParams getParams() {
        return original.getParams();
    }

    @Override
    @Deprecated
    public void setParams(final HttpParams params) {
        original.setParams(params);
    }

    public String getContent() {
        return content;
    }
}
```

StreamUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

import java.io.Closeable;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class StreamUtil {

    private static final String DEFAULT_ENCODING = "utf-8";

    public static String inputStream2String(InputStream in, String charsetName) {
        if (in == null) {
            return null;
        }

        InputStreamReader inReader = null;

        try {
            if (StringUtil.strIsNullOrEmpty(charsetName)) {
                inReader = new InputStreamReader(in, DEFAULT_ENCODING);
            } else {
                inReader = new InputStreamReader(in, charsetName);
            }

            int readLen = 0;
```



```
            char[] buffer = new char[1024];
            StringBuilder strBuf = new StringBuilder();
            while ((readLen = inReader.read(buffer)) != -1) {
                strBuf.append(buffer, 0, readLen);
            }

            return strBuf.toString();
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            closeStream(inReader);
        }

        return null;
    }

    public static void closeStream(Closeable closeable) {
        if (closeable != null) {
            try {
                closeable.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}
```

StringUtil.java

```
/*
 * Copyright Notice:
 *      Copyright  1998-2008, Huawei Technologies Co., Ltd.  ALL Rights Reserved.
 *
 *      Warning: This computer software sourcecode is protected by copyright law
 *      and international treaties. Unauthorized reproduction or distribution
 *      of this sourcecode, or any portion of it, may result in severe civil and
 *      criminal penalties, and will be prosecuted to the maximum extent
 *      possible under the law.
 */
package com.huawei.utils;

//import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.text.SimpleDateFormat;
import java.util.Base64; // JDK versionâ‰¥1.8
import java.util.Calendar;
import java.util.Locale;
import java.util.TimeZone;
import java.util.UUID;

//import org.apache.commons.codec.binary.Base64; //JDK version<1.8
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class StringUtil {

    public static boolean strIsNullOrEmpty(String s) {
        return (null == s || s.trim().length() < 1);
    }

    public static String buildAKSKHeader(String appKey, String appSecret) throws Exception {
        if (StringUtil.strIsNullOrEmpty(appKey) || StringUtil.strIsNullOrEmpty(appSecret)) {
            return null;
        }

        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");
        format.setTimeZone(TimeZone.getTimeZone("UTC"));
        Calendar calendar = Calendar.getInstance();
        String time = format.format(calendar.getTime());
```

```
        String stNonce = UUID.randomUUID().toString().replace("-", "").toUpperCase(Locale.ROOT);
        String str = stNonce + time;
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(appSecret.getBytes(StandardCharsets.UTF_8), "HmacSHA256"));
        byte[] authBytes = mac.doFinal(str.getBytes(StandardCharsets.UTF_8));
        String passwordDigestBase64Str = encodeBase64(authBytes);
        return String.format(Constant.AKSK_HEADER_FORMAT, appKey, passwordDigestBase64Str, stNonce,
time);
    }

    private static String encodeBase64(byte[] bytes) {
        if (bytes.length == 0) {
            return null;
        } else {
            return new String(org.apache.commons.codec.binary.Base64.encodeBase64(bytes),
StandardCharsets.UTF_8);
        }
    }
}
```

4.3.2.2 代码样例

“语音验证码场景 API”代码样例

```
package com.huawei.demo;

import java.util.HashMap;
import java.util.Map;

import javax.net.ssl.HttpsURLConnection;

import com.huawei.utils.Constant;
import com.huawei.utils.HttpsUtil;
import com.huawei.utils.JsonUtil;
import com.huawei.utils.StreamClosedHttpResponse;
import com.huawei.utils.StringUtil;

public class VoiceVerificationCodeMain {
    // 接口通用返回值
    private static String status = "";
    private static String resultcode = "";
    private static String resultdesc = "";
    // 语音验证码接口返回值
    private static String sessionId = "";

    // 语音验证码业务类实体
    public static VoiceVerificationCode voiceVerificationCodeAPI = new VoiceVerificationCode();

    // 调用接口成功标识
    private static final String success = "200";

    public static void main(String args[]) throws Exception {

        // TODO 程序前端要求发起语音验证码呼叫,调用doVoiceVerificationCode方法.
        // 以下代码仅供调试使用,实际开发时请删除
        VoiceVerificationCodeMain.doVoiceVerificationCode("+8653159511234", "+8613800000001", 2,
"test.wav", "1234");
        if (status.indexOf(success) != -1) {
            System.out.println(status);
            System.out.println(resultcode + " " + resultdesc);
            System.out.println("The session id is: " + sessionId);
        }

        // TODO 需要接收状态和话单时,请参考"呼叫状态和话单通知API"接口实现状态通知和话单的接收和解析
        // HostingVoiceEventDemoImpl
    }

    /*
```

```
    * 前端需要发起语音验证码呼叫时,调用此方法 该示例只仅体现必选参数,可选参数根据接口文档和实际情况配置.
     */
    public static void doVoiceVerificationCode(String displayNbr, String calleeNbr, int languageType, String preTone,
            String verifyCode) throws Exception {

        Boolean retry = false;
        // 调用语音验证码接口,直至成功
        do {
            status = voiceVerificationCodeAPI.voiceVerificationCodeAPI(displayNbr, calleeNbr, languageType, preTone,
                    verifyCode);
            if (status.indexOf(success) != -1) {
                retry = false;
                // 调用成功,记录返回的信息.
                resultcode = voiceVerificationCodeAPI.getResponsePara("resultcode");
                resultdesc = voiceVerificationCodeAPI.getResponsePara("resultdesc");
                sessionId = voiceVerificationCodeAPI.getResponsePara("sessionId");
            } else {
                retry = true;
                // 调用失败,获取错误码和错误描述.
                resultcode = voiceVerificationCodeAPI.getResponsePara("resultcode");
                resultdesc = voiceVerificationCodeAPI.getResponsePara("resultdesc");
                // 处理错误
                VoiceVerificationCodeMain.processError();
            }
        } while (retry);
    }

    // 当API的返回值不是200时,处理错误.
    private static void processError() throws InterruptedException {

        // TODO 根据错误码和错误码描述处理问题
        // 以下代码仅供调试使用,实际开发时请删除
        System.out.println(status);
        System.out.println(resultcode + " " + resultdesc);
        System.exit(-1);
    }
}

class VoiceVerificationCode {
    // 语音验证码API的URL
    private String urlVoiceVerificationCode;
    // 接口响应的消息体
    private Map<String, String> responsebody;
    // Https实体
    private HttpsUtil httpsUtil;

    public VoiceVerificationCode() {
        // 商用地址
        urlVoiceVerificationCode = Constant.VOICE_VERIFICATION_COMERCIAL;
        responsebody = new HashMap<>();
    }

    @SuppressWarnings("unchecked")
    /*
     * 该示例只仅体现必选参数,可选参数根据接口文档和实际情况配置. 该示例不体现参数校验,请根据各参数的格式要求自行实现校验功能.
     */
    public String voiceVerificationCodeAPI(String displayNbr, String calleeNbr, int languageType, String preTone,
        String verifyCode) throws Exception {

        httpsUtil = new HttpsUtil();

        // 忽略证书信任问题
        httpsUtil.trustAllHttpsCertificates();
        HttpsURLConnection.setDefaultHostnameVerifier(httpsUtil.hv);
```

```
        // 请求Headers
        Map<String, String> headerMap = new HashMap<>();
        headerMap.put(Constant.HEADER_APP_AUTH, Constant.AUTH_HEADER_VALUE);
        headerMap.put(Constant.HEADER_APP_AKSK,
            StringUtil.buildAKSKHeader(Constant.CALLVERIFY_APPID, Constant.CALLVERIFY_SECRET));

        // 构造消息体
        Map<String, Object> bodys = new HashMap<String, Object>();
        bodys.put("displayNbr", displayNbr);//主叫用户手机终端的来电显示号码。
        bodys.put("calleeNbr", calleeNbr);//发起呼叫时所拨打的被叫号码。
        bodys.put("languageType", languageType);//验证码播放的语言类型。
        bodys.put("preTone", preTone);//播放语音验证码之前需要播放的放音文件名
        bodys.put("verifyCode", verifyCode);//验证码：只支持0～9的数字，最大8位。
        String jsonRequest = JsonUtil.jsonObj2Sting(bodys);

        /*
         * Content-Type为application/json且请求方法为post时, 使用doPostJsonGetStatusLine方法构造http
         * request并获取响应.
         */
        StreamClosedHttpResponse responseVoiceVerificationCode = httpsUtil
            .doPostJsonGetStatusLine(urlVoiceVerificationCode, headerMap, jsonRequest);

        // 响应的消息体写入responsebody.
        responsebody = JsonUtil.jsonString2SimpleObj(responseVoiceVerificationCode.getContent(),
            responsebody.getClass());

        // 返回响应的status.
        return responseVoiceVerificationCode.getStatusLine().toString();
    }

    // 获取整个响应消息体
    public Map<String, String> getResponsebody() {
        return this.responsebody;
    }

    // 获取响应消息体中的单个参数
    public String getResponsePara(String paraName) {
        return this.responsebody.get(paraName);
    }
}
```

“呼叫状态通知 API”与“话单通知 API”代码样例

```
package com.huawei.demo;

import org.apache.log4j.Logger;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;

/**
 * 呼叫事件通知/话单通知
 * 客户平台收到RTC业务平台的呼叫事件通知/话单通知的接口通知
 */
public class HostingVoiceEventDemoImpl {
    private static Logger logger = Logger.getLogger(HostingVoiceEventDemoImpl.class);

    /**
     * 呼叫事件 for 语音回呼/语音通知/语音验证码
     *
     * @param jsonBody
     * @breif 详细内容以接口文档为准
     */
    public static void onCallEvent(String jsonBody) {
        // 封装JOSN请求
        JSONObject json = JSON.parseObject(jsonBody);
        String eventType = json.getString("eventType"); // 通知事件类型
```

```
if ("fee".equalsIgnoreCase(eventType)) {
    logger.info("EventType error: " + eventType);
    return;
}

if (!(json.containsKey("statusInfo"))) {
    logger.info("param error: no statusInfo.");
    return;
}
JSONObject statusInfo = json.getJSONObject("statusInfo"); // 呼叫状态事件信息

logger.info("eventType: " + eventType); // 打印通知事件类型

//callout：呼出事件
if ("callout".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.containsKey("sessionId")) {
        logger.info("sessionId: " + statusInfo.getString("sessionId"));
    }
    return;
}
//alerting：振铃事件
if ("alerting".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.containsKey("sessionId")) {
        logger.info("sessionId: " + statusInfo.getString("sessionId"));
    }
    return;
}
//answer：应答事件
if ("answer".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'userData': 用户附属信息
     * 'sessionId': 通话链路的标识ID
     * 'caller': 主叫号码
     * 'called': 被叫号码
     */
    if (statusInfo.containsKey("sessionId")) {
        logger.info("sessionId: " + statusInfo.getString("sessionId"));
    }
    return;
}
//collectInfo：放音收号结果事件,仅应用于语音通知场景
if ("collectInfo".equalsIgnoreCase(eventType)) {
    /**
     * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
     *
     * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
     * 'sessionId': 通话链路的标识ID
```



```
         * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
         */
        if (statusInfo.containsKey("digitInfo")) {
            logger.info("digitInfo: " + statusInfo.getString("digitInfo"));
        }
        return;
    }
    //disconnect：挂机事件
    if ("disconnect".equalsIgnoreCase(eventType)) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
         * 'stateCode': 通话挂机的原因值
         * 'stateDesc': 通话挂机的原因值的描述
         */
        if (statusInfo.containsKey("sessionId")) {
            logger.info("sessionId: " + statusInfo.getString("sessionId"));
        }
        return;
    }
}

/**
 * 话单通知 for 语音回呼/语音通知/语音验证码
 *
 * @param jsonBody
 * @breif 详细内容以接口文档为准
 */
public static void onFeeEvent(String jsonBody) {
    // 封装JSON请求
    JSONObject json = JSON.parseObject(jsonBody);
    String eventType = json.getString("eventType"); // 通知事件类型

    if (!("fee".equalsIgnoreCase(eventType))) {
        logger.info("EventType error: " + eventType);
        return;
    }

    if (!(json.containsKey("feeLst"))) {
        logger.info("param error: no feeLst.");
        return;
    }
    JSONArray feeLst = json.getJSONArray("feeLst"); // 呼叫话单事件信息
    /**
     * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
     *
     * 'direction': 通话的呼叫方向,以RTC业务平台为基准
     * 'spId': 客户的云服务账号
     * 'appKey': 应用的app_key
     * 'icid': 呼叫记录的唯一标识
     * 'bindNum': 发起此次呼叫的CallEnabler业务号码
     * 'sessionId': 通话链路的唯一标识
     * 'callerNum': 主叫号码
     * 'calleeNum': 被叫号码
     * 'fwdDisplayNum': 转接呼叫时的显示号码(仅语音回呼场景携带)
     * 'fwdDstNum': 转接呼叫时的转接号码(仅语音回呼场景携带)
     * 'fwdStartTime': 转接呼叫操作的开始时间(仅语音回呼场景携带)
     * 'fwdAlertingTime': 转接呼叫操作后的振铃时间(仅语音回呼场景携带)
     * 'fwdAnswerTime': 转接呼叫操作后的应答时间(仅语音回呼场景携带)
     * 'callEndTime': 呼叫结束时间
     * 'fwdUnaswRsn': 转接呼叫操作失败的Q850原因值
     * 'failTime': 呼入,呼出的失败时间
     * 'ulFailReason': 通话失败的拆线点
```

```
        * 'sipStatusCode': 呼入,呼出的失败SIP状态码
        * 'callOutStartTime': Initcall的呼出开始时间
        * 'callOutAlertingTime': Initcall的呼出振铃时间
        * 'callOutAnswerTime': Initcall的呼出应答时间
        * 'callOutUnaswRsn': Initcall的呼出失败的Q850原因值
        * 'dynIVRStartTime': 自定义动态IVR开始时间(仅语音通知场景携带)
        * 'dynIVRPath': 自定义动态IVR按键路径(仅语音通知场景携带)
        * 'recordFlag': 录音标识
        * 'recordStartTime': 录音开始时间(仅语音回呼场景携带)
        * 'recordObjectName': 录音文件名(仅语音回呼场景携带)
        * 'recordBucketName': 录音文件所在的目录名(仅语音回呼场景携带)
        * 'recordDomain': 存放录音文件的域名(仅语音回呼场景携带)
        * 'recordFileDownloadUrl': 录音文件下载地址(仅语音回呼场景携带)
        * 'ttsPlayTimes': 应用TTS功能时,使用TTS的总次数
        * 'ttsTransDuration': 应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
        * 'serviceType': 携带呼叫的业务类型信息
        * 'hostName': 话单生成的服务器设备对应的主机名
        * 'userData': 用户附属信息
        */
       //短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
       if (feeLst.size() > 1) {
           for (Object loop : feeLst) {
               if (((JSONObject)loop).containsKey("sessionId")) {
                   logger.info("sessionId: " + ((JSONObject)loop).getString("sessionId"));
               }
           }
       } else if (feeLst.size() == 1) {
           if (feeLst.getJSONObject(0).containsKey("sessionId")) {
               logger.info("sessionId: " + feeLst.getJSONObject(0).getString("sessionId"));
           }
       } else {
           logger.info("feeLst error: no element.");
       }
   }
}
```

4.3.3 Python

样例	语音验证码场景API、呼叫状态通知API、话单通知API
环境要求	Python 3.0及以上版本。
引用库	requests 2.18.1 1. 请自行下载安装Python 3.x，并完成环境配置。 2. 打开命令行窗口，执行pip install requests命令。 3. 执行pip list查看安装结果。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音验证码场景 API”代码样例

```
# -*- coding: utf-8 -*-
import base64
```

```
import hashlib
import requests #需要先使用pip install requests命令安装依赖
import time
import uuid
import hmac

from hashlib import sha256

#必填,请参考"开发准备-申请资源"获取如下数据,替换为实际值
base_url = 'https://{domain}:{port}'#APP接入地址,购买服务时下发,请替换为实际值
appKey = '***appKey***'#语音回呼应用的appKey,购买服务时下发,请替换为实际值请替换为实际值
appSecret = '***appSecret***'#语音回呼应用的appSecret,购买服务时下发,请替换为实际值请替换为实际值

def buildAKSKHeader(appKey, appSecret):
    now = time.strftime('%Y-%m-%dT%H:%M:%SZ') #Created
    nonce = str(uuid.uuid4()).replace('-','') #Nonce
    digist = hmac.new(appSecret.encode(), (nonce + now).encode(), digestmod=sha256).digest()
    digestBase64 = base64.b64encode(digist).decode() #PasswordDigest
    return 'UsernameToken Username="{}",PasswordDigest="{}",Nonce="{}",Created="{}"'.format(appKey,
digestBase64, nonce, now);

def voiceVerificationCodeAPI(displayNbr, calleeNbr, languageType, preTone, verifyCode):
    if len(displayNbr) < 1 or len(calleeNbr) < 1 or len(languageType) < 1 or len(preTone) < 1 or
len(verifyCode) < 1:
        return

    apiUri = '/rest/httpsessions/callVerify/v1.0'
    requestUrl = base_url + apiUri

    header = {
        'Content-Type': 'application/json;charset=UTF-8',
        'Authorization': 'AKSK realm="SDP",profile="UsernameToken",type="Appkey"',
        'X-AKSK': buildAKSKHeader(appKey, appSecret)
    }

    jsonData = {
        # 必填参数
        'displayNbr': displayNbr,#主叫用户手机终端的来电显示号码。
        'calleeNbr': calleeNbr,#发起呼叫时所拨打的被叫号码。
        'languageType': languageType,#验证码播放的语言类型。
        'preTone': preTone,#播放语音验证码之前需要播放的放音文件名
        'verifyCode': verifyCode#验证码：只支持0～9的数字，最大8位。
        # 选填参数
#         'posTone': 'postone.wav', #播放语音验证码之后需要播放的放音文件名
#         'times': 3, #播放次数:0～9
#         'statusUrl': '', #设置SP接收状态上报的URL,要求使用BASE64编码
#         'feeUrl': '', #设置SP接收话单上报的URL,要求使用BASE64编码
#         'returnIdlePort': 'false', #指示是否需要返回平台空闲呼叫端口数量
#         'userData': 'customerId123' #设置用户的附属信息
    }

    try:
        r = requests.post(requestUrl, json=jsonData, headers=header, verify=False)
        print(r.text) #打印响应结果
    except requests.exceptions.HTTPError as e:
        print(e.code)
        print(e.read().decode('utf-8')) #打印错误信息
    pass

if __name__ == '__main__':
    voiceVerificationCodeAPI('+86531*******4', '+86135*******1', 2, 'test.wav', '1234');
```

“呼叫状态通知 API”代码样例

```
# -*- coding: utf-8 -*-
'''
呼叫事件通知
客户平台收到语音通话平台的呼叫事件通知的接口通知
'''
```

```
import json

#呼叫事件通知样例
jsonBody = json.dumps({
    'eventType': 'callout',
    'statusInfo': {
        'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp': '2019-01-24 03:04:24',
        'caller': '+86138*******2',
        'called': '+86138*******1',
        'userData': 'customerId123'
    }
}).encode('ascii')

print(jsonBody)

'''
呼叫事件通知
@see: 详细内容以接口文档为准
@param param: jsonBody
@return:
'''
def onCallEvent(jsonBody):
    jsonObj = json.loads(jsonBody) #将通知消息解析为jsonObj
    eventType = jsonObj['eventType'] #通知事件类型

    if ('fee' == eventType):
        print('EventType error: ' + eventType)
        return

    if ('statusInfo' not in jsonObj):
        print('param error: no statusInfo.')
        return
    statusInfo = jsonObj['statusInfo'] #呼叫状态事件信息

    print('eventType: ' + eventType) #打印通知事件类型

    #callout：呼出事件
    if ('callout' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #alerting：振铃事件
    if ('alerting' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #answer：应答事件
    if ('answer' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
```

```
        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return
    #collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ('collectInfo' == eventType):
        '''
        Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'sessionId': 通话链路的标识ID
        'digitInfo': 放音收号场景中,语音通话平台对开发者进行放音收号操作的结果描述
        '''
        if ('digitInfo' in statusInfo):
            print('digitInfo: ' + statusInfo['digitInfo'])
        return
    #disconnect：挂机事件
    if ('disconnect' == eventType):
        '''
        Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理

        'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
        'userData': 用户附属信息
        'sessionId': 通话链路的标识ID
        'caller': 主叫号码
        'called': 被叫号码
        'partyType': 挂机的用户类型,仅在语音回呼场景携带
        'stateCode': 通话挂机的原因值
        'stateDesc': 通话挂机的原因值的描述
        '''
        if ('sessionId' in statusInfo):
            print('sessionId: ' + statusInfo['sessionId'])
        return

if __name__ == '__main__':
    onCallEvent(jsonBody) #呼叫事件处理
```

“话单通知 API”代码样例

```
# -*- coding: utf-8 -*-
'''
话单通知
客户平台收到语音通话平台的话单通知的接口通知
'''
import json

#话单通知样例
jsonBody = json.dumps({
    'eventType': 'fee',
    'feeLst': [
        {
            'direction': 0, #通话的呼叫方向,以语音通话平台为基准
            'spId': 'CaaS_Test_01', #客户的云服务账号
            'appKey': 'ka4k******9YnEm2', #应用的app_key
            'icid': 'CAE-20190124110424-12019775', #呼叫记录的唯一标识
            'bindNum': '+86755*******8', #发起此次呼叫的CallEnabler业务号码,即绑定号码
            'sessionId': '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com', #通话链路的
唯一标识
            'callerNum': '+86138*******2', #主叫号码
            'calleeNum': '+86138*******1', #被叫号码
            'fwdDisplayNum': '+86138*******2', #转接呼叫时的显示号码(仅语音回呼场景携带)
            'fwdDstNum': '+86138*******1', #转接呼叫时的转接号码(仅语音回呼场景携带)
```

```
            'fwdStartTime': '2019-01-24 03:04:31', #转接呼叫操作的开始时间(仅语音回呼场景携带)
            'fwdAlertingTime': '2019-01-24 03:04:36', #转接呼叫操作后的振铃时间(仅语音回呼场景携带)
            'fwdAnswerTime': '2019-01-24 03:04:38', #转接呼叫操作后的应答时间(仅语音回呼场景携带)
            'callEndTime': '2019-01-24 03:04:49', #呼叫结束时间
            'fwdUnaswRsn': 0, #转接呼叫操作失败的Q850原因值
            'failTime': '', #呼入,呼出的失败时间
            'ulFailReason': 0, #通话失败的拆线点
            'sipStatusCode': 0, #呼入,呼出的失败SIP状态码
            'callOutStartTime': '2019-01-24 03:04:24', #Initcall的呼出开始时间
            'callOutAlertingTime': '2019-01-24 03:04:27', #Initcall的呼出振铃时间
            'callOutAnswerTime': '2019-01-24 03:04:31', #Initcall的呼出应答时间
            'callOutUnaswRsn': 0, #Initcall的呼出失败的Q850原因值
            'dynIVRStartTime': '', #自定义动态IVR开始时间(仅语音通知场景携带)
            'dynIVRPath': '', #自定义动态IVR按键路径(仅语音通知场景携带)
            'recordFlag': 0, #录音标识
            'recordStartTime': '', #录音开始时间(仅语音回呼场景携带)
            'recordObjectName': '', #录音文件名(仅语音回呼场景携带)
            'recordBucketName': '', #录音文件所在的目录名(仅语音回呼场景携带)
            'recordDomain': '', #存放录音文件的域名(仅语音回呼场景携带)
            'recordFileDownloadUrl': '', #录音文件下载地址(仅语音回呼场景携带)
            'ttsPlayTimes': 0, #应用TTS功能时,使用TTS的总次数
            'ttsTransDuration': 0, #应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
            'serviceType': '002', #携带呼叫的业务类型信息
            'hostName': 'callenabler245.huaweicaas.com', #话单生成的服务器设备对应的主机名
            'userData': '' #用户附属信息
        }
    ]
}).encode('ascii')

print(jsonBody)

'''
话单通知
@see: 详细内容以接口文档为准
@param param: jsonBody
@return:
'''
def onFeeEvent(jsonBody):
    jsonObj = json.loads(jsonBody) #将通知消息解析为jsonObj
    eventType = jsonObj['eventType'] #通知事件类型

    if ('fee' != eventType):
        print('EventType error: ' + eventType)
        return

    if ('feeLst' not in jsonObj):
        print('param error: no feeLst.');
        return

    feeLst = jsonObj['feeLst']

    #Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
    #短时间内有多个通话结束时语音通话平台会将话单合并推送,每条消息最多携带50个话单
    if (len(feeLst) > 1):
        for loop in feeLst:
            if ('sessionId' in loop):
                print('sessionId: ' + loop['sessionId'])
    elif(len(feeLst) == 1):
        if ('sessionId' in feeLst[0]):
            print('sessionId: ' + feeLst[0]['sessionId'])
    else:
        print('feeLst error: no element.');

if __name__ == '__main__':
    onFeeEvent(jsonBody) #话单处理
```


4.3.4 PHP

样例	语音验证码场景API、呼叫状态通知API、话单通知API
环境要求	PHP 7.0及以上版本。
引用库	-

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

"语音验证码场景 API”代码样例

```
<?php
include 'data.php';
include 'util.php';

/**
 * voiceVerificationCodeAPI
 * @param string $displayNbr
 * @param string $calleeNbr
 * @param integer $languageType
 * @param string $preTone
 * @param string $verifyCode
 */
function voiceVerificationCodeAPI($displayNbr, $calleeNbr, $languageType, $preTone, $verifyCode) {
    if (empty($displayNbr) || empty($calleeNbr) || empty($languageType) || empty($preTone) ||
empty($verifyCode)) {
        return;
    }

    $method = 'POST';
    $apiUri = '/rest/httpsessions/callVerify/v1.0';
    $xaksk = buildAKSKHeader(getCallverify_AppId(), getCallverify_Secret());

    $content = json_encode([
        /* 必填参数*/
        'displayNbr' => $displayNbr,//主叫用户手机终端的来电显示号码。
        'calleeNbr' => $calleeNbr,//发起呼叫时所拨打的被叫号码。
        'languageType' => $languageType,//验证码播放的语言类型。
        'preTone' => $preTone,//播放语音验证码之前需要播放的放音文件名。
        'verifyCode' => $verifyCode//验证码：只支持0～9的数字，最大8位。
        /* 选填参数*/
//        'bindNbr' => '+86123456789', //CallEnabler业务号码,即绑定号码
//        'posTone' => 'postone.wav', //播放语音验证码之后需要播放的放音文件名
//        'times' => 3, //播放次数:0～9
//        'statusUrl' => '', //设置SP接收状态上报的URL,要求使用BASE64编码
//        'feeUrl' => '', //设置SP接收话单上报的URL,要求使用BASE64编码
//        'returnIdlePort' => 'false', //指示是否需要返回平台空闲呼叫端口数量
//        'userData' => 'customerId123' //设置用户的附属信息
    ]);

    $requestUrl = getBaseUrl() . $apiUri;
    $context_options = createOptions($method, $xaksk, $content);

    try {
```

```
        $response = file_get_contents($requestUrl, false, stream_context_create($context_options)); // 发送请求
        print_r($response . PHP_EOL); // 打印响应结果
    } catch (Exception $e) {
        echo $e->getMessage(); //打印错误信息
    }
}

voiceVerificationCodeAPI('+8653159511234', '+8613500000001', 2, 'test.wav', '1234');
?>
```

“呼叫状态通知 API”代码样例

```
<?php
/**
 * 呼叫事件通知
 * 客户平台收到RTC业务平台的呼叫事件通知的接口通知
 */
//呼叫事件通知样例
$jsonBody = json_encode([
    'eventType' => 'callout',
    'statusInfo' => [
        'sessionId' => '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com',
        'timestamp' => '2019-01-24 03:04:24',
        'caller' => '+8613800000022',
        'called' => '+8613800000021'
    ]
]);
print_r($jsonBody . PHP_EOL);
onCallEvent($jsonBody);//呼叫事件处理
/**
 * 呼叫事件通知
 * @desc 详细内容以接口文档为准
 * @param jsonBody
 */
function onCallEvent($jsonBody) {
    $jsonArr = json_decode($jsonBody, true); //将通知消息解析为关联数组
    $eventType = $jsonArr['eventType']; //通知事件类型

    if (strcasecmp($eventType, 'fee') == 0) {
        print_r('EventType error: ' . $eventType);
        return;
    }

    if (!array_key_exists('statusInfo', $jsonArr)) {
        print_r('param error: no statusInfo.');
        return;
    }
    $statusInfo = $jsonArr['statusInfo']; //呼叫状态事件信息

    print_r('eventType: ' . $eventType . PHP_EOL); //打印通知事件类型
    //callout：呼出事件
    if (strcasecmp($eventType, 'callout') == 0) {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (array_key_exists('sessionId', $statusInfo)) {
            print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
        }
        return;
    }
    //alerting：振铃事件
    if (strcasecmp($eventType, 'alerting') == 0) {
        /**
```

```
        * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
        *
        * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
        * 'userData': 用户附属信息
        * 'sessionId': 通话链路的标识ID
        * 'caller': 主叫号码
        * 'called': 被叫号码
        */
       if (array_key_exists('sessionId', $statusInfo)) {
           print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
       }
       return;
   }
   //answer：应答事件
   if (strcasecmp($eventType, 'answer') == 0) {
       /**
        * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
        *
        * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
        * 'userData': 用户附属信息
        * 'sessionId': 通话链路的标识ID
        * 'caller': 主叫号码
        * 'called': 被叫号码
        */
       if (array_key_exists('sessionId', $statusInfo)) {
           print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
       }
       return;
   }
   //collectInfo：放音收号结果事件,仅应用于语音通知场景
   if (strcasecmp($eventType, 'collectInfo') == 0) {
       /**
        * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
        *
        * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
        * 'sessionId': 通话链路的标识ID
        * 'digitInfo': 放音收号场景中,RTC业务平台对开发者进行放音收号操作的结果描述
        */
       if (array_key_exists('digitInfo', $statusInfo)) {
           print_r('digitInfo: ' . $statusInfo['digitInfo'] . PHP_EOL);
       }
       return;
   }
   //disconnect：挂机事件
   if (strcasecmp($eventType, 'disconnect') == 0) {
       /**
        * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
        *
        * 'timestamp': 该呼叫事件发生时RTC业务平台的UNIX时间戳
        * 'userData': 用户附属信息
        * 'sessionId': 通话链路的标识ID
        * 'caller': 主叫号码
        * 'called': 被叫号码
        * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
        * 'stateCode': 通话挂机的原因值
        * 'stateDesc': 通话挂机的原因值的描述
        */
       if (array_key_exists('sessionId', $statusInfo)) {
           print_r('sessionId: ' . $statusInfo['sessionId'] . PHP_EOL);
       }
       return;
   }
}
?>
```

“话单通知 API”代码样例

```
<?php
/**
```

```
 * 话单通知
 * 客户平台收到RTC业务平台的话单通知的接口通知
 */
//话单通知样例
$jsonBody = json_encode([
    'eventType' => 'fee',
    'feeLst' => [
        [
            'direction' => 0, //通话的呼叫方向,以RTC业务平台为基准
            'spId' => 'CaaS_Test_01', //客户的云服务账号
            'appKey' => 'ka4kESI5s3YyurL1wpx63s9YnEm2', //应用的app_key
            'icid' => 'CAE-20190124110424-12019775', //呼叫记录的唯一标识
            'bindNum' => '+8675512345678', //发起此次呼叫的CallEnabler业务号码,即绑定号码
            'sessionId' => '1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com', //通话链
路的唯一标识
            'callerNum' => '+8613800000022', //主叫号码
            'calleeNum' => '+8613800000021', //被叫号码
            'fwdDisplayNum' => '+8613800000022', //转接呼叫时的显示号码(仅语音回呼场景携带)
            'fwdDstNum' => '+8613866887021', //转接呼叫时的转接号码(仅语音回呼场景携带)
            'fwdStartTime' => '2019-01-24 03:04:31', //转接呼叫操作的开始时间(仅语音回呼场景携带)
            'fwdAlertingTime' => '2019-01-24 03:04:36', //转接呼叫操作后的振铃时间(仅语音回呼场景携带)
            'fwdAnswerTime' => '2019-01-24 03:04:38', //转接呼叫操作后的应答时间(仅语音回呼场景携带)
            'callEndTime' => '2019-01-24 03:04:49', //呼叫结束时间
            'fwdUnaswRsn' => 0, //转接呼叫操作失败的Q850原因值
            'failTime' => '', //呼入,呼出的失败时间
            'ulFailReason' => 0, //通话失败的拆线点
            'sipStatusCode' => 0, //呼入,呼出的失败SIP状态码
            'callOutStartTime' => '2019-01-24 03:04:24', //Initcall的呼出开始时间
            'callOutAlertingTime' => '2019-01-24 03:04:27', //Initcall的呼出振铃时间
            'callOutAnswerTime' => '2019-01-24 03:04:31', //Initcall的呼出应答时间
            'callOutUnaswRsn' => 0, //Initcall的呼出失败的Q850原因值
            'dynIVRStartTime' => '', #自定义动态IVR开始时间(仅语音通知场景携带)
            'dynIVRPath' => '', //自定义动态IVR按键路径(仅语音通知场景携带)
            'recordFlag' => 0, //录音标识
            'recordStartTime' => '', //录音开始时间(仅语音回呼场景携带)
            'recordObjectName' => '', //录音文件名(仅语音回呼场景携带)
            'recordBucketName' => '', //录音文件所在的目录名(仅语音回呼场景携带)
            'recordDomain' => '', //存放录音文件的域名(仅语音回呼场景携带)
            'recordFileDownloadUrl' => '', //录音文件下载地址(仅语音回呼场景携带)
            'ttsPlayTimes' => 0, //应用TTS功能时,使用TTS的总次数
            'ttsTransDuration' => 0, //应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
            'serviceType' => '002', //携带呼叫的业务类型信息
            'hostName' => 'callenabler245.huaweicaas.com', //话单生成的服务器设备对应的主机名
            'userData' => '' //用户附属信息
        ]
    ]
]);
print_r($jsonBody . PHP_EOL);
onFeeEvent($jsonBody);//话单处理
/**
 * 话单通知
 * @desc 详细内容以接口文档为准
 * @param jsonBody
 */
function onFeeEvent($jsonBody) {
    $jsonArr = json_decode($jsonBody, true); //将通知消息解析为关联数组
    $eventType = $jsonArr['eventType']; //通知事件类型

    if (strcasecmp($eventType, 'fee') != 0) {
        print_r('EventType error: ' . $eventType);
        return;
    }

    if (!array_key_exists('feeLst', $jsonArr)) {
        print_r('param error: no feeLst.');
        return;
    }

    $feeLst = $jsonArr['feeLst']; //呼叫话单事件信息
```

```
//Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
//短时间内有多个通话结束时RTC业务平台会将话单合并推送,每条消息最多携带50个话单
if (sizeof($feeLst) > 1) {
    foreach ($feeLst as $loop){
        if (array_key_exists('sessionId', $loop)) {
            print_r('sessionId: ' . $loop['sessionId'] . PHP_EOL);
        }
    }
} else if(sizeof($feeLst) == 1) {
    if (array_key_exists('sessionId', $feeLst[0])) {
        print_r('sessionId: ' . $feeLst[0]['sessionId'] . PHP_EOL);
    }
} else {
    print_r('feeLst error: no element.');
}
}
?>
```

4.3.5 C#

样例	语音验证码场景API、呼叫状态通知API、话单通知API
环境要求	.NET Core 2.0及以上版本或.NET Framework 4.7.1及以上版本。
引用库	Newtonsoft.Json 11.0.2，请参考https://www.newtonsoft.com/json获取。

须知

- 本文档所述Demo在提供服务的过程中，可能会涉及个人数据的使用，建议您遵从国家的相关法律采取足够的措施，以确保用户的个人数据受到充分的保护。
- 本文档所述Demo仅用于功能演示，不允许客户直接进行商业使用。
- 本文档信息仅供参考，不构成任何要约或承诺。

“语音验证码场景 API”代码样例

```
using Newtonsoft.Json;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Security.Cryptography;
using System.Text;

namespace voicecall_csharp_demo_x_aksk_
{
    class VoiceVerificationCode
    {
        string base_url = "https://{domain}:{port}"; //APP接入地址,购买服务时下发,请替换为实际值
        string appKey = "***appKey***"; //语音验证码应用的appKey,购买服务时下发,请替换为实际值
        string appSecret = "***appSecret***"; //语音验证码应用的appSecret,购买服务时下发,请替换为实际值

        static void Main(string[] args)
```

```
    {
        //固话号码从控制台号码管理页获取.此处取值仅为样例,请替换为实际值
        voiceVerificationCodeAPI("+86531*******4", "+86135*******1", 2, "test.wav", "1234");
    }

    static void voiceVerificationCodeAPI(string displayNbr, string calleeNbr, int languageType, string
preTone, string verifyCode)
    {
        if (String.IsNullOrEmpty(displayNbr) || String.IsNullOrEmpty(calleeNbr) ||
String.IsNullOrEmpty(languageType) || String.IsNullOrEmpty(preTone) || String.IsNullOrEmpty(verifyCode))
        {
            return;
        }

        string apiURI = "/rest/httpsessions/callVerify/v1.0"; //接口URI
        string requestUrl = base_url + apiURI;

        try
        {
            //为防止因HTTPS证书认证失败造成API调用失败,需要先忽略证书信任问题
            //.NET Framework 4.7.1及以上版本,请采用如下代码
            var sslHandler = new HttpClientHandler()
            {
                ServerCertificateCustomValidationCallback = (message, cert, chain, err) => { return true; }
            };
            HttpClient client = new HttpClient(sslHandler, true);
            //低于.NET Framework 4.7.1版本,请采用如下代码
            //HttpClient client = new HttpClient();
            //ServicePointManager.ServerCertificateValidationCallback = new
RemoteCertificateValidationCallback(CheckValidationResult);

            //请求Headers
            client.DefaultRequestHeaders.Add("Authorization", "AKSK realm=\"SDP\",profile=
\"UsernameToken\",type=\"Appkey\"");
            client.DefaultRequestHeaders.Add("X-AKSK", buildAKSKHeader(appKey, appSecret));

            //请求Body
            var body = new Dictionary<string, object>() {
                /*必填参数*/
                {"displayNbr", displayNbr},//主叫用户手机终端的来电显示号码
                {"calleeNbr", calleeNbr},//发起呼叫时所拨打的被叫号码。
                {"languageType", languageType},//验证码播放的语言类型。
                {"preTone", preTone},//播放语音验证码之前需要播放的放音文件名
                {"verifyCode", verifyCode}//验证码：只支持0～9的数字，最大8位。
                /*选填参数*/

                //{"posTone", "postone.wav"}, //播放语音验证码之后需要播放的放音文件名
                //{"times", 3}, //播放次数:0～9
                //{"statusUrl", ""}, //设置SP接收状态上报的URL,要求使用BASE64编码
                //{"feeUrl", ""}, //设置SP接收话单上报的URL,要求使用BASE64编码
                //{"returnIdlePort", "false"}, //指示是否需要返回平台空闲呼叫端口数量
                //{"userData", "customerId123"} //设置用户的附属信息
            };

            HttpContent content = new StringContent(JsonConvert.SerializeObject(body));
            //请求Headers中的Content-Type参数
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            var response = client.PostAsync(requestUrl, content).Result;
            Console.WriteLine(response.StatusCode);
            var res = response.Content.ReadAsStringAsync().Result;
            Console.WriteLine(res); //打印响应结果
        }
        catch (Exception e)
        {
            Console.WriteLine(e.StackTrace);
            Console.WriteLine(e.Message); //打印错误信息
        }
    }
```

```
        static string buildAKSKHeader(string appKey, string appSecret)
        {
            string now = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ"); //Created
            string nonce = Guid.NewGuid().ToString().Replace("-", ""); //Nonce
            String str = nonce + now;

            byte[] keyByte = Encoding.UTF8.GetBytes(appSecret);
            byte[] messageBytes = Encoding.UTF8.GetBytes(str);
            using (var hmacsha256 = new HMACSHA256(keyByte))
            {
                byte[] hashmessage = hmacsha256.ComputeHash(messageBytes);
                string base64 = Convert.ToBase64String(hashmessage);
                return String.Format("UsernameToken Username=\"{0}\",PasswordDigest=\"{1}\",Nonce=
\"{2}\",Created=\"{3}\"",
                    appKey, base64, nonce, now);
            }
        }

        //低于.NET Framework 4.7.1版本,启用如下方法
        //static bool CheckValidationResult(object sender, X509Certificate certificate, X509Chain chain,
SslPolicyErrors errors)
        //{
        //    return true;
        //}
    }
}
```

“呼叫状态通知 API”代码样例

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;

namespace voicecall_csharp_demo_x_aksk_
{
    class CallEventImpl
    {
        static void Main(string[] args)
        {
            //呼叫事件通知样例
            string jsonBody = JsonConvert.SerializeObject(new Dictionary<string, object>(){
                {"eventType", "callout"},
                {"statusInfo", new Dictionary<string, object>(){
                        {"sessionId", "1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com"},
                        {"timestamp", "2019-01-24 03:04:24"},
                        {"caller", "+86138*******2"},
                        {"called", "+86138*******1"},
                    }
                }
            });

            Console.WriteLine("jsonBody:" + jsonBody);

            OnCallEvent(jsonBody); //呼叫事件处理
        }

        /// <summary>
        /// 呼叫事件通知,详细内容以接口文档为准
        /// </summary>
        /// <param name="jsonBody"></param>
        static void OnCallEvent(string jsonBody)
        {
            JObject jsonObj = (JObject)JsonConvert.DeserializeObject(jsonBody); //将通知消息解析为jsonObj
            string eventType = jsonObj["eventType"].ToString(); //通知事件类型

            if ("fee".Equals(eventType))
            {
```

```
        Console.WriteLine("EventType error:" + eventType);
        return;
    }

    if (!jsonObj.ContainsKey("statusInfo"))
    {
        Console.WriteLine("param error: no statusInfo.");
        return;
    }
    JObject statusInfo = (JObject)jsonObj["statusInfo"]; //呼叫状态事件信息

    Console.WriteLine("eventType:" + eventType); //打印通知事件类型

    //callout：呼出事件
    if ("callout".Equals(eventType))
    {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.ContainsKey("sessionId"))
        {
            Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
        }
        return;
    }
    //alerting：振铃事件
    if ("alerting".Equals(eventType))
    {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.ContainsKey("sessionId"))
        {
            Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
        }
        return;
    }
    //answer：应答事件
    if ("answer".Equals(eventType))
    {
        /**
         * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
         *
         * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
         * 'userData': 用户附属信息
         * 'sessionId': 通话链路的标识ID
         * 'caller': 主叫号码
         * 'called': 被叫号码
         */
        if (statusInfo.ContainsKey("sessionId"))
        {
            Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
        }
        return;
    }
    //collectInfo：放音收号结果事件,仅应用于语音通知场景
    if ("collectInfo".Equals(eventType))
```

```
        {
            /**
             * Example: 此处以解析digitInfo为例,请按需解析所需参数并自行实现相关处理
             *
             * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
             * 'sessionId': 通话链路的标识ID
             * 'digitInfo': 放音收号场景中,语音通话平台对开发者进行放音收号操作的结果描述
             */
            if (statusInfo.ContainsKey("digitInfo"))
            {
                Console.WriteLine("digitInfo:" + statusInfo["digitInfo"]);
            }
            return;
        }
        //disconnect：挂机事件
        if ("disconnect".Equals(eventType))
        {
            /**
             * Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
             *
             * 'timestamp': 该呼叫事件发生时语音通话平台的UNIX时间戳
             * 'userData': 用户附属信息
             * 'sessionId': 通话链路的标识ID
             * 'caller': 主叫号码
             * 'called': 被叫号码
             * 'partyType': 挂机的用户类型,仅在语音回呼场景携带
             * 'stateCode': 通话挂机的原因值
             * 'stateDesc': 通话挂机的原因值的描述
             */
            if (statusInfo.ContainsKey("sessionId"))
            {
                Console.WriteLine("sessionId:" + statusInfo["sessionId"]);
            }
            return;
        }
    }
  }
}
```

“话单通知 API”代码样例

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;

namespace voicecall_csharp_demo_x_aksk_
{
   class FeeImpl
   {
      static void Main(string[] args)
      {
         //话单通知样例
         string jsonBody = JsonConvert.SerializeObject(new Dictionary<string, object>(){
            {"eventType", "fee"},
            {"feeLst",  new object[] {
               new Dictionary<string, object>(){
                     {"direction", 0}, //通话的呼叫方向,以语音通话平台为基准
                     {"spId", "CaaS_Test_01"}, //客户的云服务账号
                     {"appKey", "ka4k*****Em2"}, //应用的app_key，购买服务时下发,请替换为实际值
                     {"icid", "CAE-20190124110424-12019775"}, //呼叫记录的唯一标识
                     {"bindNum", "+8675512****78"}, //发起此次呼叫的CallEnabler业务号码,即绑定号码
                     {"sessionId",
"1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com"}, //通话链路的唯一标识
                     {"callerNum", "+86138****0022"}, //主叫号码
                     {"calleeNum", "+86138****0021"}, //被叫号码
                     {"fwdDisplayNum", "+86138****0022"}, //转接呼叫时的显示号码(仅语音回呼场景携带)
                     {"fwdDstNum", "+86138****7021"}, //转接呼叫时的转接号码(仅语音回呼场景携带)
```

```
                    {"fwdStartTime", "2019-01-24 03:04:31"}, //转接呼叫操作的开始时间(仅语音回呼场景携带)
                    {"fwdAlertingTime", "2019-01-24 03:04:36"}, //转接呼叫操作后的振铃时间(仅语音回呼场景
携带)
                    {"fwdAnswerTime", "2019-01-24 03:04:38"}, //转接呼叫操作后的应答时间(仅语音回呼场景
携带)
                    {"callEndTime", "2019-01-24 03:04:49"}, //呼叫结束时间
                    {"fwdUnaswRsn", 0}, //转接呼叫操作失败的Q850原因值
                    {"failTime", ""}, //呼入,呼出的失败时间
                    {"ulFailReason", 0}, //通话失败的拆线点
                    {"sipStatusCode", 0}, //呼入,呼出的失败SIP状态码
                    {"callOutStartTime", "2019-01-24 03:04:24"}, //Initcall的呼出开始时间
                    {"callOutAlertingTime", "2019-01-24 03:04:27"}, //Initcall的呼出振铃时间
                    {"callOutAnswerTime", "2019-01-24 03:04:31"}, //Initcall的呼出应答时间
                    {"callOutUnaswRsn", 0}, //Initcall的呼出失败的Q850原因值
                    {"dynIVRStartTime", ""}, //自定义动态IVR开始时间(仅语音通知场景携带)
                    {"dynIVRPath", ""}, //自定义动态IVR按键路径(仅语音通知场景携带)
                    {"recordFlag", 0}, //录音标识
                    {"recordStartTime", ""}, //录音开始时间(仅语音回呼场景携带)
                    {"recordObjectName", ""}, //录音文件名(仅语音回呼场景携带)
                    {"recordBucketName", ""}, //录音文件所在的目录名(仅语音回呼场景携带)
                    {"recordDomain", ""}, //存放录音文件的域名(仅语音回呼场景携带)
                    {"recordFileDownloadUrl", ""}, //录音文件下载地址(仅语音回呼场景携带)
                    {"ttsPlayTimes", 0}, //应用TTS功能时,使用TTS的总次数
                    {"ttsTransDuration", 0}, //应用TTS功能时,TTS Server进行TTS转换的总时长(单位为秒)
                    {"serviceType", "002"}, //携带呼叫的业务类型信息
                    {"hostName", "callenabler245.huaweicaas.com"}, //话单生成的服务器设备对应的主机名
                    {"userData", "customerId123"} //用户附属信息
                }
            }
        }
    });

    Console.WriteLine(jsonBody);

    OnFeeEvent(jsonBody); //话单处理
}

/// <summary>
/// 话单通知,详细内容以接口文档为准
/// </summary>
/// <param name="jsonBody"></param>
static void OnFeeEvent(string jsonBody)
{
    JObject jsonObj = (JObject)JsonConvert.DeserializeObject(jsonBody); //将通知消息解析为jsonObj
    string eventType = jsonObj["eventType"].ToString(); //通知事件类型

    if (!"fee".Equals(eventType))
    {
        Console.WriteLine("EventType error:" + eventType);
        return;
    }

    if (!jsonObj.ContainsKey("feeLst"))
    {
        Console.WriteLine("param error: no feeLst.");
        return;
    }
    JObject[] feeLst = jsonObj["feeLst"].ToObject<JObject[]>(); //呼叫话单事件信息

    //Example: 此处以解析sessionId为例,请按需解析所需参数并自行实现相关处理
    //短时间内有多个通话结束时语音通话平台会将话单合并推送,每条消息最多携带50个话单
    if (feeLst.Length > 1)
    {
        foreach (JObject loop in feeLst)
        {
            if (loop.ContainsKey("sessionId"))
            {
                Console.WriteLine("sessionId:" + loop["sessionId"]);
            }
```

```
            }
        }
        else if (feeLst.Length == 1)
        {
            if (feeLst[0].ContainsKey("sessionId"))
            {
                Console.WriteLine("sessionId:" + feeLst[0]["sessionId"]);
            }
        }
        else
        {
            Console.WriteLine("feeLst error: no element.");
        }
    }
  }
}
```

5 API 错误码

说明

本章节响应码数量较多，建议您使用快捷键**Ctrl+F**在界面进行搜索，找到您需要的错误码处理建议。

404 问题处理

若调用接口时返回了404响应，请检查APP接入地址和访问URI（详见**申请资源**）是否都填写正确，且拼接成了完整的请求URL，如“https://rtccall.myhuaweicloud.cn:443/rest/httpsessions/click2Call/v2.0”。

错误码处理

调用语音通话相关接口会产生接口调用错误码，响应示例如下：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
"resultcode":"0",
"resultdesc":"Success",
"sessionId":"1202_566_0_20161228102743@callenabler.home1.com"
}
```

resultcode参数处理

参数取值	英文描述	中文描述	处理建议
0	Success.	请求成功	无需处理。
1010001	Internal system error.	系统错误。	联系华为云客服处理。
1010002	Invalid request.	非法请求。	检查请求携带的参数格式是否都合法。

参数取值	英文描述	中文描述	处理建议
1010003	Invalid app_key.	无效的app_key。	检查请求携带的app_key是否填写正确，app_key从**应用管理**页面获取，若填写正确，请在**应用管理**页面检查请求携带的app_key所属应用状态是否正常。
1010006	Invalid Rest API.	无效的Rest API。	请参照**接口文档**中的请求方法检查对应接口的请求方法填写是否正确。
1010007	The ativeState of User is not ACTIVATING.	用户状态未激活。	请检查app_key所属的华为云账户是否处于欠费状态，若处于欠费状态，请参考**华为云账户充值**完成充值，若没有处于欠费状态，请联系华为云客服处理。
1010008	The status of the app_key is unavailable.	app_key被暂停使用。	请在**应用管理**页面检查请求携带的app_key所属应用状态是否正常。
1010009	No more APIs can be invoked.	• 刷新授权API：app_secret无效。 • 其他API：API达到调用上限。	• 刷新授权API：输入正确的app_secret，app_secret从“**应用管理**”页面获取。 • 其他API：请稍等一分钟后再试，并联系华为云客服申请更高的应用使用配额。
1010010	The flow control upper limit is reached on the platform.	平台达到系统流控上限。	请稍等一分钟后再试。
1010011	The app is not allowed to access a commercial address.	APP没有访问商用地址的权限。	请在**应用管理**页面检查请求携带的app_key所属应用状态是否正常。
1010013	Time out limit.	时间超出限制。	请确认X-AKSK鉴权时，生成随机数的时间与发送请求时的本地时间不能相差太大（具体差值请与管理员确认）。
1010021	Application unavailable.	应用不可用。	请在**应用管理**页面检查请求携带的app_key所属应用状态是否正常。
1010022	Invalid verification code.	验证码不合法。	请检查verifyCode参数的填写是否合法。

参数取值	英文描述	中文描述	处理建议
1010023	Invalid display number.	固话号码不合法。	检查displayNbr和displayCalleeNbr参数的填写是否合法，与**号码管理**页面的“固话号码”保持一致。若合法，请确认该号码是否已申请并下发。申请号码在**号码订购**页面申请，号码下发后可在**号码管理**页面查看。
1010024	Invalid caller number.	主叫号码不合法。	检查callerNbr参数的填写是否合法，若合法，请联系管理员确认。
1010028	The API is not allowed to be invoked.	此API已禁止调用。	请将请求中的version参数改为v2.0并修改其余相关参数，再重新调用该API。
1010040	The app_key is not allowed to invoke the API.	app_key没有调用本API的权限。	请联系华为云客服确认该app_key对应的应用是否具有语音通话（语音回呼、语音通知、语音验证码）能力。
1012001	Resource of number is not to be applied.	资源未申请。	app_key和业务号码未绑定。
1012002	The template ID is not approved.	模板ID审核未通过。	请在控制台**语音模板管理**页面确认该模板ID是否已审核通过。
1012003	The template ID does not exist.	模板ID不存在。	请在控制台**语音模板管理**页面确认该模板ID是否已添加。
1012004	The template ID does not exist.	templateParas的参数个数与模板的变量个数不一致。	请检查templateParas携带的变量值个数和templateId对应的模板内容中变量的个数是否一致。
1012005	%s of templateParas does not meet template requirements.	参数templateParas中的%s不符合模板定义的要求。	请检查templateParas携带的变量值格式与长度是否符合templateId对应的模板内容中变量的定义。
1012006	The service number is not applied.	业务号码未申请。	请确认是否申请业务号码。

参数取值	英文描述	中文描述	处理建议
1012012	Application does not open recording function.	应用未开启录音功能。	请在**应用管理**页面确认请求携带的app_key是否开启了录音功能。
1013001	Calls exceed the SP limit.	请求次数超过SP配置上限。	请联系华为云客服确认开发者呼叫数量限制。
1013002	Calls exceed the APP limit.	呼叫数超过APP的阈值，app_key是{}，策略名是{}。	请联系华为云客服确认应用呼叫数量限制。
1013003	Calls exceed the display number limit.	呼叫数超过号码{}的阈值，策略名是{}。	请联系华为云客服确认显示号码呼叫数量限制。 注：若是全局呼叫频次策略组，则不返回具体号码。
1013004	Callee in blacklist.	用户{}在黑名单里面，策略名是{}。	请联系华为云客服确认被叫黑名单限制。
1013011	Callee is not on the whitelist.	被叫用户不在白名单中。	请联系华为云客服确认被叫号码白名单限制。
1016001	The record does not exist.	记录不存在。	检查请求携带的callerNbr和app_key是否填写正确。app_key从**应用管理**页面获取。并确保使用该callerNbr和app_key调用过“语音回呼场景API”。
1020001	Parameter error.	参数错误。	检查请求携带的参数格式是否都合法。
1020002	Internal error.	内部错误。	请联系华为云客服处理。
1020003	Parameter error.	参数错误。	根据API接口文档的参数描述和要求，排查已开发的代码中参数设置是否有效。
1020150	Invalid app_key.	app_key无效。	检查请求携带的app_key是否填写正确，app_key从**应用管理**页面获取，若填写正确，请在**应用管理**页面检查请求携带的app_key所属应用状态是否正常。
1020151	The bindNum is invalid.	业务号码无效。	业务号码无效，请联系管理员处理。

参数取值	英文描述	中文描述	处理建议
1020152	Invalid sessionId.	sessionId无效。	检查请求携带的sessionId是否填写正确。语音回呼的sessionId是调用“语音回呼场景API”的成功响应消息的sessionId参数值，也可通过呼叫状态和话单通知API获取。
1020154	Insufficient voice ports.	语音端口不足。	请稍等一分钟后再试，并联系华为云客服申请扩容语音端口。
1020165	The number of app_key voice call ports exceeds the upper limit.	超出app_key语音呼叫端口数限制。	请稍等一分钟后再试，并联系华为云客服为该app_key对应的应用申请更多的端口配额。
1020166	The app client ip is not in ip white list.	对端app IP不在白名单列表中。	联系华为云客服检查IP白名单是否配置正确。
1020176	Authentication failed, try again later.	鉴权失败，稍后重试。	IP因鉴权失败次数过多导致被拉黑，请30分钟后重试，或联系华为云客服申请放通该IP。
1023001	Internal error.	内部错误。	请联系管理员处理。
1023002	Response timeout.	响应超时。	重新发送一次请求，若依然返回响应超时，请联系华为云客服处理。
1023006	Authorization not contained in the HTTP header.	HTTP消息头未找到Authorization字段。	请检查HTTP消息头中是否携带了Authorization字段。
1023007	realm not contained in Authorization.	Authorization字段中未找到realm属性。	请检查Authorization字段中的是否携带了realm属性。
1023008	profile not contained in Authorization.	Authorization字段中未找到profile属性。	请检查Authorization字段中的是否携带了profile属性。
1023009	The value of realm in Authorization must be SDP.	Authorization中realm属性值应该为“SDP”。	请检查Authorization字段中的realm属性值是否为“SDP”。

参数取值	英文描述	中文描述	处理建议
1023010	The value of profile in Authorization must be UsernameToken.	Authorization中profile属性值应该为“UsernameToken”。	请检查Authorization字段中的profile属性值是否为“UsernameToken”。
1023011	The value of type in Authorization must be app_key.	Authorization中type属性值应该为“Appkey”。	请检查Authorization字段中的type属性值是否为“Appkey”。
1023012	type not contained in Authorization.	Authorization字段中未找到type属性。	请检查Authorization字段中是否携带了type属性。
1023033	HTTP header not found X-AKSK field.	HTTP头未找到X-AKSK字段。	请检查HTTP消息头中是否携带了X-AKSK字段。
1023034	UserName not contained in X-AKSK.	X-AKSK字段中未找到UserName属性。	请检查X-AKSK字段中的是否携带了Username属性。
1023035	Nonce not contained in X-AKSK.	X-AKSK字段中未找到Nonce属性。	请检查X-AKSK字段中的是否携带了Nonce属性。
1023036	Created not contained in X-AKSK.	X-AKSK字段中未找到Created属性。	请检查X-AKSK字段中的是否携带了Created属性。
1023037	PasswordDigest not contained in X-AKSK.	X-AKSK字段中未找到PasswordDigest属性。	请检查X-AKSK字段中的是否携带了PasswordDigest属性。
1023038	UsernameToken not contained in X-AKSK.	X-AKSK中没有携带UsernameToken。	请检查X-AKSK字段中的是否携带了UsernameToken属性。
-		-	获取录音文件下载地址API：成功响应，请从Location头域中获取录音文件下载地址。

6 挂机原因值、Q850 原因值、呼叫拆线点

调用语音通话相关接口会产生接口调用错误码，详见**API错误码**。

调用接口成功后，如果“statusUrl”和“feeUrl”参数指定了客户接收状态上报的URL和接收话单上报的URL，或在**添加应用**时指定了呼叫状态接收地址和呼叫话单接收地址，则语音通话平台在接收到南向网元返回的呼叫状态通知和话单通知时，会主动将呼叫状态通知和话单通知推送给客户。

消息示例如下：

```
POST /status HTTP/1.1

Content-Length: xx

{"eventType":"fee","feeLst":
[{"direction":0,"spId":"CaaS_Test_01","appKey":"ka4k******YnEm2","icid":"CAE-20190124110424-12019775","bindNum":"+8675512****78","sessionId":"1201_612_4294967295_20190124030424@callenabler245.huaweicaas.com","callerNum":"+86138****0022","calleeNum":"+86138****0021","fwdDisplayNum":"+86138****0022","fwdDstNum":"+86138****7021","fwdStartTime":"2019-01-24 03:04:31","fwdAlertingTime":"2019-01-24 03:04:36","fwdAnswerTime":"2019-01-24 03:04:38","callEndTime":"2019-01-24 03:04:49","fwdUnaswRsn":0,"ulFailReason":0,"sipStatusCode":0,"callOutStartTime":"2019-01-24 03:04:24","callOutAlertingTime":"2019-01-24 03:04:27","callOutAnswerTime":"2019-01-24 03:04:31","callOutUnaswRsn":0,"recordFlag":0,"ttsPlayTimes":0,"ttsTransDuration":0,"serviceType":"002","hostName":"callenabler245.huaweicaas.com"}]}
```

挂机原因值、Q850原因值、呼叫拆线点请参考**通话挂机原因值说明**。

7 商业发布

确认获取所有资源，完成线下开发后，正式发布此应用在行业使用。