Linkus SDK 手册

Yeastar P 系列软件版

版本: 1.0
日期: 2026年09月21日

400-661-8815
0592-5503307
www.yeastar.cn

Contents

Linkus SDK 介绍.....1
Yeastar Linkus SDK 介绍.....1
Linkus Android SDK.....4
概述.....4
更新记录.....6
接入 Linkus Android SDK.....6
启用 Linkus SDK 并绑定推送证书.....6
集成 Linkus Android SDK.....8
初始化 Linkus Android SDK.....10
获取 Linkus Android SDK 登录签名.....13
使用 Linkus Android SDK.....17
登录与连接状态.....17
通话功能.....21
多方通话.....25
会议室通话.....27
推送消息.....32
通话记录.....33
音频配置.....34
Linkus iOS SDK.....35
概述.....35
更新记录.....36
接入 Linkus iOS SDK.....36
启用 Linkus SDK 并绑定 APNs 证书.....36
集成 Linkus iOS SDK.....38
获取 Linkus iOS SDK 登录签名.....40
使用 Linkus iOS SDK.....44
配置.....44
登录及登出.....45
通话功能.....46
会议室通话.....47

通话信息 48
复杂通话场景 50
通话记录 51
Linkus macOS SDK 53
概述 53
更新记录 54
接入 Linkus macOS SDK 54
启用 Linkus SDK 54
集成 Linkus macOS SDK 55
获取 Linkus macOS SDK 登录签名 57
使用 Linkus macOS SDK 61
配置 61
登录及登出 62
通话功能 63
通话信息 64
复杂通话场景 66
通话记录 67
Linkus Windows SDK 68
概述 68
更新记录 69
启用 Linkus SDK 69
获取 Linkus Windows SDK 登录签名 70
Linkus Windows SDK Core 75
集成 Linkus Windows SDK Core 75
使用 Linkus Windows SDK Core 79
Linkus Windows SDK UI 102
集成 Linkus Windows SDK UI 102
Linkus Web SDK 109
概述 109
更新记录 110
启用 Linkus SDK 110
获取 Linkus Web SDK 登录签名 111
Linkus Web SDK Core 116

集成 Linkus Web SDK Core..... 116
使用 Linkus Web SDK Core..... 120
Linkus Web SDK UI.....146
集成 Linkus Web SDK UI..... 146

Linkus SDK 介绍

Yeastar Linkus SDK 介绍

Yeastar P 系列软件版 提供 Linkus 软件开发工具包 (SDK) 用于二次开发。开发者可以将第三方应用与 Linkus SDK 对接，从而快速实现在第三方应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。

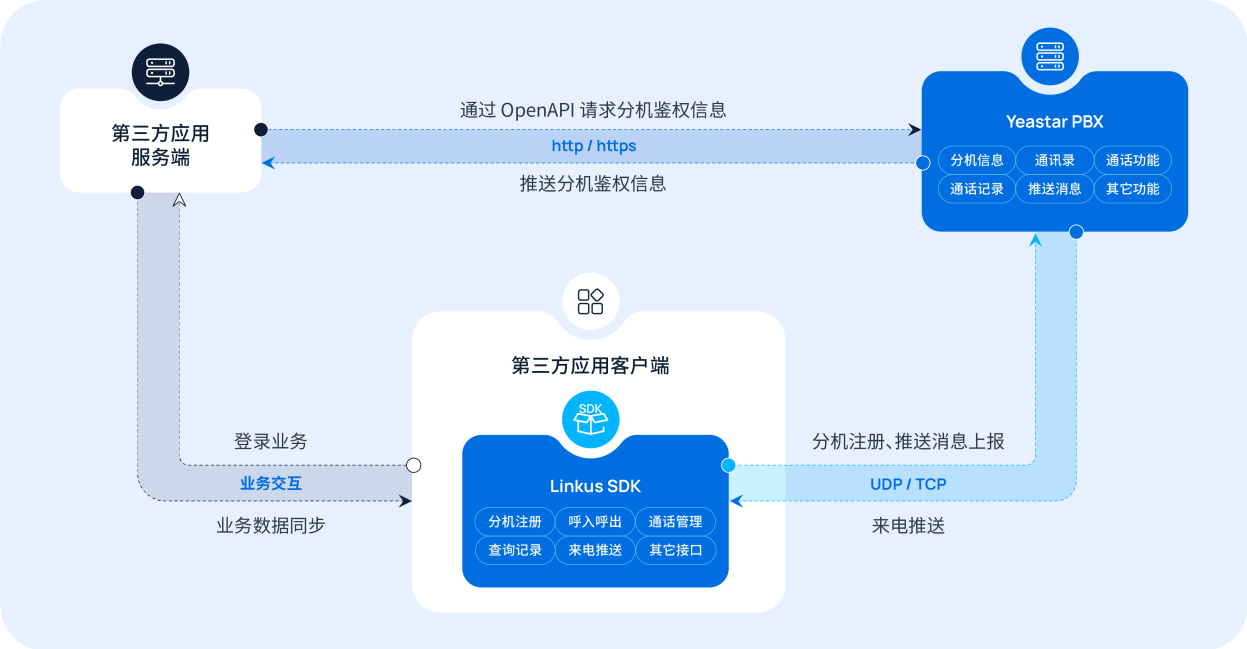
使用要求

确保 Yeastar P 系列软件版 满足以下要求：

- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

> **Note:**
> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

产品架构



支持平台

Linkus SDK 支持多平台开发。你可以将 Linkus SDK 与已有应用、网页、平台等进行集成，简化开发流程，快速实现语音通信功能。

平台	相关资源
Android 手机端	• 主要功能 • Demo & 源码 • Linkus SDK 开发手册
iOS 手机端	• 主要功能 • Demo & 源码 • Linkus SDK 开发手册
macOS 桌面端	• 主要功能 • Demo & 源码 • Linkus SDK 开发手册

Windows 桌面端	• 主要功能
	• Demo & 源码
	  ◦ 核心通话功能
	  ◦ UI 组件
	• Linkus SDK 开发手册

Web 端	• 主要功能
	• Demo & 源码
	  ◦ 核心通话功能
	  ◦ UI 组件
	• Linkus SDK 开发手册

主要功能

参见下表以了解不同平台的 Linkus SDK 所支持的主要功能。

		Android 手机端	iOS 手机端	macOS 桌面端	Windows 桌面端	Web 端
基础通话功能	1 对 1 语音通话	✓	✓	✓	✓	✓
	通话中发起新呼叫	✗	✗	✓	✓	✓
	通话转接 (盲转 & 咨询转)	✓	✓	✓	✓	✓
	通话保持 & 通话恢复	✓	✓	✓	✓	✓
	通话静音	✓	✓	✓	✓	✓

		Android 手机端	iOS 手机端	macOS 桌面端	Windows 桌面端	Web 端
	通话录音	✓	✓	✓	✓	✓
	呼叫等待	✓	✓	✓	✓	✓
高级通话功能	多方通话 (最多五方)	✓	✓	✗	✗	✗
	会议室通话 (最多九方)	✓	✓	✗	✗	✗
	选择语音编解码	✓	✓	✗	✗	✗
	选择音频设备	✗	✗	✓	✓	✓
	通话质量检测	✓	✓	✓	✓	✓
通话记录查询与管理	查询通话记录	✓	✓	✓	✓	✓
	删除通话记录	✓	✓	✗	✗	✗
	查询未接来电数量	✓	✓	✗	✗	✗
系统推送通知	来电推送	✓	✓	✗	✗	✗
	未接来电推送	✓	✓	✗	✗	✗

Linkus Android SDK

Linkus Android SDK 概述

Yeastar P 系列软件版 支持 Linkus SDK，可实现在第三方 Android 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。本文介绍 Linkus Android SDK 的使用要求、前提条件、Demo & 项目源码、接入流程及功能。

使用要求

平台 / 环境	要求
PBX 服务器	• **固件版本**：83.12.0.23 或更高 • **订阅服务**：**Linkus SDK 服务** **Note:** 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。
开发环境	• **Java 版本**：Java 11 • **Android 版本**：8.0 或更高 • **Android Studio 版本**：Arctic Fox 或更高 • **Gradle 版本**：6.5 • **Android Gradle Plugin 版本**：4.1.1 **Note:** • 如使用其它版本的 Gradle 及 Gradle 插件，可能需要额外的调试才能保证 Linkus SDK 的正常运行。 • SDK 兼容 32 位及 64 位操作系统。

前提条件

已向 Android 设备的运营商获取到推送证书。

Note:

- 推送证书用于保证在完成 Linkus SDK 与 Android 应用的集成后，可以在 Android 设备上接收到来电通知。

> - **Vivo 推送**、**荣耀推送**、**OPPO 推送** 需要 PBX 版本 83.15.0.22 以上才能使用。

不同 Android 设备所需的推送证书信息不同，参见下表以了解各个平台所需的证书信息。

推送平台	所需证书信息
安卓谷歌 Firebase 推送	完整证书
安卓个推推送	证书中的指定字段：AppId、AppKey、MasterSecret
华为推送	证书中的指定字段：AppID、AppSecret
小米推送	证书中的指定字段：App Secret, App Package Name, Channel ID, Custom Ringtone Channel ID, Custom Ringtone Path
Vivo 推送	证书中的指定字段：App ID、App Key、App Secret
荣耀推送	证书中的指定字段：App ID、Client ID、Client Secret
OPPO 推送	证书中的指定字段：App Key、Master Secret、 Channel ID

Demo & 源码

体验 Linkus Android SDK 的 Demo 并查阅项目源码，以了解 Linkus Android SDK 的整体运行框架及流程。

更多信息，前往 Linkus Android SDK 的 GitHub 仓库。

接入流程及功能

接入流程

1. 启用 Linkus SDK 并绑定推送证书
2. 集成 Linkus Android SDK
3. 初始化 Linkus Android SDK
4. 获取 Linkus Android SDK 登录签名

功能

- 登录与连接状态
- 通话功能
- 多方通话
- 会议室通话
- 推送消息
- 通话记录

- 音频配置

Linkus Android SDK 更新记录

版本 1.2.5

发布日期：2024年4月22日

- 新增兼容平板设备，支持在平板端及手机端同时使用 Linkus SDK。
- 新增支持推送证书：**Vivo 推送** 及 **荣耀推送**。

> **Note:**
> 该功能要求 PBX 版本为 83.14.0.24 或更高。

版本 1.1.2

发布日期：2023年10月11日

- 首次发布 Linkus Android SDK。通过集成 SDK 与 Android 项目，快速实现在第三方 Android 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。

接入 Linkus Android SDK

启用 Linkus SDK 并绑定推送证书

将 Linkus SDK 接入至 Android 项目之前，你需要先在 PBX 上启用 Linkus SDK 功能，并绑定设备运营商所提供的推送证书，从而保证 Android 设备能够在集成后接收到来电通知。

使用要求

确保 Yeastar P 系列软件版 满足以下要求：

- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

> **Note:**

> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

前提条件

已向 Android 设备的运营商获取到推送证书。

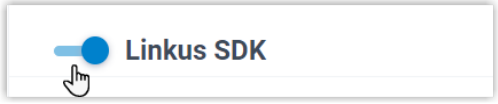
> **Note:**
>
> - 推送证书用于保证在完成 Linkus SDK 与 Android 应用的集成后，可以在 Android 设备上接收到来电通知。
> - **Vivo 推送**、**荣耀推送**、**OPPO 推送** 需要 PBX 版本 83.15.0.22 以上才能使用。

不同 Android 设备所需的推送证书信息不同，参见下表以了解各个平台所需的证书信息。

推送平台	所需证书信息
安卓谷歌 Firebase 推送	完整证书
安卓个推推送	证书中的指定字段：AppId、AppKey、MasterSecret
华为推送	证书中的指定字段：AppID、AppSecret
小米推送	证书中的指定字段：App Secret, App Package Name, Channel ID, Custom Ringtone Channel ID, Custom Ringtone Path
Vivo 推送	证书中的指定字段：App ID、App Key、App Secret
荣耀推送	证书中的指定字段：App ID、Client ID、Client Secret
OPPO 推送	证书中的指定字段：App Key、Master Secret、 Channel ID

操作步骤

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 打开 **Linkus SDK** 开关并在弹窗中点击 **确认**。



3. 绑定推送证书。

a. 在 **推送证书** 栏，点击 **添加**。
b. 在 **平台** 下拉列表中，选择推送证书所属的平台，然后上传证书或输入证书中的指定字段。
c. 点击 **保存**。

4. 点击 **保存**。

执行结果

你已启用 Linkus SDK 功能且完成证书绑定，可以 集成 Linkus Android SDK。

> **Important:**
> 完成 Linkus Android SDK 的集成后，你的 Android 应用将使用绑定的证书向设备发送通话相关的推送消息，而 Linkus 提供的推送证书 (Linkus 手机端的推送功能) 将不再生效。

集成 Linkus Android SDK

本文介绍如何通过手动导入 Linkus SDK AAR 文件的方式，将 Linkus SDK 与 Android 项目进行集成。

使用要求及前提条件

使用要求

确保你的开发环境满足以下要求：

开发环境	版本要求
Java	Java 11
Android	8.0 及以上
Android Studio	Arctic Fox 及以上

开发环境	版本要求
Gradle & Gradle 插件	• **Gradle**：6.5 • **Android Gradle Plugin**：4.1.1 **Note:** 如使用其它版本的 Gradle 及 Gradle 插件，可能需要额外的调试才能保证 Linkus SDK 的正常运行。

前提条件

你已 启用 Linkus SDK 并绑定推送证书。

Demo & 源码

在正式集成之前，我们推荐你体验 Linkus Android SDK 的 Demo 并查阅项目源码，以了解 Linkus Android SDK 的整体运行框架及流程。

更多信息，前往 Linkus Android SDK 的 GitHub 仓库。

操作步骤

1. 前往 Linkus SDK 的 GitHub 仓库，下载最新版本的 **linkus-sdk-x.x.x.aar** 文件。

   Note:
   x.x.x 代表 Linkus SDK 的版本号，请参见 Linkus Android SDK 更新记录 查看最新的版本号。

2. 将 **linkus-sdk-x.x.x.aar** 文件复制到 Android 项目的 **app > libs** 目录中。
3. 在项目中引入 Linkus SDK 的 AAR 文件。

   Note:
   - 开发环境版本不同，引入 AAR 文件的方式会存在差异。你可根据实际的开发环境选择对应的引入方式。
   - 如果你的工程文件采用的是 Kotlin 语法 (文件拓展名为 **.kts**)，你需要将以下代码转换为 Kotlin 语法。

   a. 在项目的根目录下，打开 **build.gradle** 文件，并在 **allprojects > repositories** 下添加以下代码。

   ```
   flatDir {
       dirs 'app/libs'
   ```

```
}
```

b. 在 **app** 目录下，打开 **build.gradle** 文件，并在 **dependencies** 下添加以下代码。

```
implementation(name: 'linkus-sdk-x.x.x', ext: 'aar')
```

> **Note:**
> **linkus-sdk-x.x.x** 表示 Linkus SDK 的 AAR 文件的文件名 (不包括扩展名)。

4. 同步项目。

执行结果

你已成功将 Linkus SDK 与你的 Android 应用进行集成，可 初始化 Linkus Android SDK。

> **Note:**
> AAR 文件中已包含混淆文件，你无需进行额外的设置以防止代码混淆。

初始化 Linkus Android SDK

使用 Linkus Android SDK 之前，你需要对其进行初始化以启动核心服务和组件。Linkus Android SDK 支持默认初始化及自定义初始化两种方法。

前提条件

- 你已 启用 Linkus SDK 并绑定推送证书。
- 你已 集成 Linkus Android SDK。

限制条件

初始化只能执行一次，且必须在主进程中执行。

背景信息

Linkus Android SDK 默认配置

下表展示了 Linkus Android SDK 的基础配置项及默认值。

配置项	默认值
语音编解码	iLBC
音频自动增益	禁用
回音消除	禁用
音质调试	禁用
主动降噪	启用
呼叫等待	启用

初始化选项

Linkus Android SDK 支持默认初始化及自定义初始化两种方式，选择任意一种方式初始化 Linkus Android SDK。

- **默认初始化**：使用默认的参数配置对 Linkus Android SDK 进行初始化，无需额外设置。

  更多信息，参见 默认初始化 Linkus Android SDK。
- **自定义初始化**：使用自定义的参数配置对 Linkus Android SDK 进行初始化。支持为 Linkus Android SDK 自定义以下参数及设置：
  - 音频自动增益
  - 回音消除
  - 主动降噪
  - 呼叫等待
  - Linkus SDK 的信息存储地址

  更多信息，参见 自定义初始化 Linkus Android SDK。

默认初始化 Linkus Android SDK

实现方法

在项目的 Application 类的 onCreate() 方法中，调用以下方法对 Linkus Android SDK 进行初始化并应用默认的参数配置。

```
YlsBaseManager.getInstance().initYlsSDK(this, null);
```

示例代码

```
public class LinkusDemo extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        // 在主进程的入口点调用初始化方法
        YlsBaseManager.getInstance().initYlsSDK( context: this,  ylsInitConfig: null);
    }
}
```

自定义初始化 Linkus Android SDK

实现方法

在项目的 Application 类的 onCreate() 方法中，调用以下方法自定义 Linkus SDK 的基础参数设置并对其进行初始化。

```
YlsInitConfig config = new
 YlsInitConfig.Builder(projectPath)//指定 Linkus SDK 信息
 （包括日志信息） 的保存地址

 .supportCallWaiting(true)//是否启用呼叫等待；true：启用，false：禁用
        .agc(true)//是否启用音频自动增益；true：启用，false：禁用
        .ec(true)//是否启用回音消除；true：启用，false：禁用
        .nc(true)//是否启用主动降噪；true：启用，false：禁用
        .key("")//可选：指定数据库的访问密码
        .build();//应用以上参数初始化 Linkus SDK
        YlsBaseManager.getInstance().initYlsSDK(this, config);
```

示例代码

```
public class LinkusDemo extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        // 在主进程的入口点调用初始化方法
        String projectPath = YlsBaseManager.getInstance().getProjectPath( context: this);//保持默认 SDK 信息存储路径
        YlsInitConfig config = new YlsInitConfig.Builder(projectPath)
                .supportCallWaiting( b: false)//禁用呼叫等待
                .agc( b: true)//启用音频自动增益
                .key( s: "12345")//设置数据库密码
                .build();
        YlsBaseManager.getInstance().initYlsSDK( context: this, config);
    }
}
```

后续步骤

向 PBX 服务器请求 SDK 登录签名，用于用户鉴权及登录 Linkus Android SDK。

更多信息，参见 获取 Linkus Android SDK 登录签名。

Related information

初始化会议室通话功能

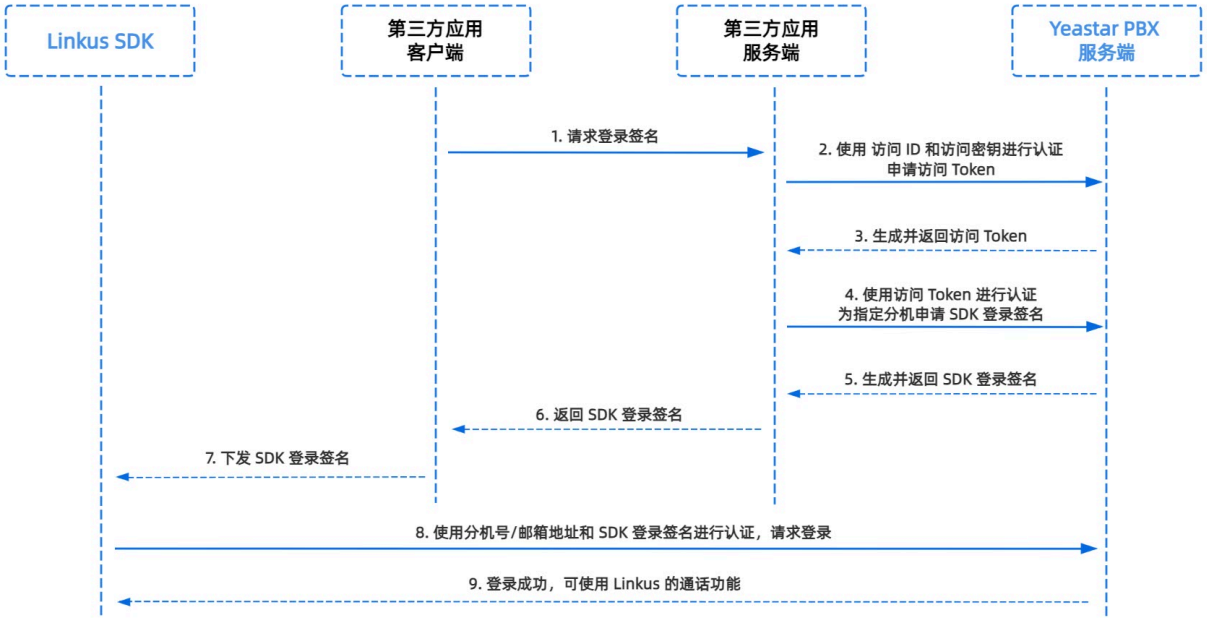
获取 Linkus Android SDK 登录签名

用户登录 Linkus SDK 时需要使用 SDK 登录签名进行鉴权，而不是登录密码。本文介绍如何通过 OpenAPI 向 PBX 服务器请求用户的 Linkus SDK 登录签名。

前提条件

- 你已 启用 Linkus SDK 并绑定推送证书。
- 你已 集成并初始化 Linkus Android SDK。

Linkus SDK 登录流程



步骤一、在 PBX 获取 Linkus SDK 的访问 ID 及密钥

从 Yeastar P 系列软件版 上获取 Linkus SDK 的访问 ID 及密钥，用于应用服务端向 PBX 进行认证并建立连接。

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 记录 **访问 ID** 及 **访问密钥**。

步骤二、在应用服务端向 PBX 申请访问 Token

通过 OpenAPI，使用 Linkus SDK 的访问 ID 及密钥向 PBX 申请一个访问 Token。访问 Token 会作为 PBX API 接口调用的凭证，用于为分机用户请求 Linkus SDK 的登录签名。

请求 URL

```
POST {base_url}/openapi/v1.0/get_token
```

> **Note:**
> 如需了解 PBX 的 API 请求结构，参见 **请求结构**。

请求参数

参数	是否必填	类型	说明
username	是	String	用户名。在步骤一中获取的 **访问 ID** 作为用户名。
password	是	String	密码。在步骤一中获取的 **访问密钥** 作为密码。

请求示例

```
POST /openapi/v1.0/get_token
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "VXKkvsR2w5H28JAWtBXuJHMTeaQO7Zmf",
 "password": "Yq6yVsBceOZLhnuaGeMUG4U4qXXXXXXX"
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。

参数	类型	说明
		• `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
access_token_expire_time	Integer	访问 token 有效时长 (**单位**：秒)。
access_token	String	访问 token，即 API 接口调用凭证。所有的请求都需要带一个访问 token。
refresh_token_expire_time	Integer	刷新 token 有效时长 (**单位**：秒)。
refresh_token	String	刷新 token。此刷新 token 可用于获取新的访问 token 和刷新 token。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "access_token_expire_time": 1800,
    "access_token": "EXZMpZAO86mbrKm6rFtgeb3rfcpC9uqS",
    "refresh_token_expire_time": 86400,
    "refresh_token": "SCduGecwbG9jIusiS8FxFUVn3kf0Q9R8"
}
```

步骤三、为分机请求登录签名

通过 OpenAPI， 使用访问 Token 向 PBX 服务器请求分机的登录签名。

请求 URL

```
POST /openapi/v1.0/sign/create?access_token={access_token}
```

请求参数

参数	是否必填	类型	说明
username	是	string	分机号或邮箱地址。
sign_type	是	string	登录签名的类型。 **允许填写的值**：`sdk`

参数	是否必填	类型	说明
expire_time	否	Integer	登录签名的到期时间戳 (秒)。 `0` 表示不限制登录签名的有效时长。

请求示例

```
POST /openapi/v1.0/sign/create?access_token=EXZMpZAO86mbrKm6rFtg
eb3rfcpC9uqS
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "1000",
 "sign_type": "sdk",
       "expire_time": 0
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
data	Array<Ext_Sign>	分机的登录签名。

Ext_Sign

参数	类型	说明
sign	String	分机的登录签名。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
```

```
    "data":{
        "sign": "ueb3rfcpsiS8FxFUZMpZAO86mbrKmVn3kf0Q9R8"
    }
}
```

执行结果

第三方应用服务端已获取到用户登录 Linkus SDK 所需的签名，且自动将其同步至应用客户端，并下发给 Linkus SDK。

后续操作

使用登录签名向 PBX 服务器进行鉴权并 登录 Linkus Android SDK。

使用 Linkus Android SDK

登录与连接状态

本文介绍与 Linkus Android SDK 登录及连接状态相关的功能及实现方法。

支持功能

功能	说明
首次登录 (手动登录)	初始化 Linkus SDK 后的首次登录。首次登录需要提供完整的登录凭证以及 PBX 服务器的地址，用于与 PBX 服务器建立连接并进行身份验证。
缓存登录	首次登录后，Linkus SDK 会将登录凭据及 PBX 服务器信息缓存在本地设备上，以便用户在后续的登录中可以直接使用，而无需再次输入。
查询用户登录状态	检查当前用户的 Linkus SDK 登录状态。
查询与 PBX 服务器的连接状态	判断 Linkus SDK 是否与 PBX 服务器相连接。
SDK 通知回调	监听及处理用户账号登出、与 PBX 重连成功及 CDR 变更等事件。

首次登录 (手动登录)

此方法会在用户首次登录 Linkus SDK 时使用。此方法要求用户输入自己的分机号或邮箱地址及登录签名作为登录凭证，以及 PBX 的 IP 地址用于与 PBX 服务器建立连接。

前提条件

已 获取 Linkus Android SDK 登录签名。

实现方法

```
/**
 * 手动登录
 *
 * @param context
 * @param userName：用户的分机号或邮箱地址
 * @param passWord：用户的 Linkus SDK 登录签名
 * @param localeIp：PBX 的本地 IP 地址
 * @param localePort：PBX 本地 IP 地址所使用的端口
 * @param remoteIp：PBX 的远程 IP 地址
 * @param remotePort：PBX 远程 IP 地址所使用的端口
 * 根据用户的实际使用环境，"localeIp,localePort" 和 "remoteIp,
 remotePort" 两组数据中至少有一组必须填写对应的值。
 * @param requestCallback 登录结果回调
 * @return
 登录接口返回值，表示登录失败或异常的原因。更多信息参见代码下方的表格。
 */
public void loginBlock(Context context, String userName, String
 passWord, String localeIp, int localePort,
        String remoteIp, int remotePort, RequestCallback<Boolean
> requestCallback)
//手动登录示例
        YlsLoginManager.getInstance().loginBlock(this, userName,
 password, localeIp,
        localePortI, remoteIp, remotePortI, new RequestCallback<
>() {

@Override // 登录成功回调
public void onSuccess(Boolean result) {
        closeProgressDialog();
        startActivity(new Intent(LoginActivity.this, DialPadActi
vity.class));
        }

@Override// 登录失败回调
public void onFailed(int code) {
        closeProgressDialog();
        Toast.makeText(LoginActivity.this, R.string.login_tip_lo
gin_failed, Toast.LENGTH_LONG).show();
        }

@Override// 登录异常回调
public void onException(Throwable exception) {
        closeProgressDialog();
```

```
        Toast.makeText(LoginActivity.this, R.string.login_tip_lo
gin_failed, Toast.LENGTH_LONG).show();
        }
        });
```

登录接口返回值说明

返回值	说明
1	无法连接 PBX 服务器。
-5	登录请求无响应。
403	用户名或登录签名错误。
405	Linkus 客户端被禁用。
407	该用户账号已被锁定。
416	禁止访问所请求的 IP 地址 (PBX 已启用 国家地区 IP 访问防御)。

缓存登录

首次登录后，Linkus SDK 会缓存登录信息。此方法会自动读取已有的缓存信息进行登录，无需用户再次输入。

```
int networkType = NetWorkUtil.getNetWorkType(this);
        YlsLoginManager.getInstance().cacheLogin(networkType);
```

查询用户登录状态

```
/**
 * 判断用户是否已登录 Linkus SDK
 * @return 返回值，表示用户的登录状态；true：用户已登录，false：用户未登录
 */
public boolean isLoginEd()
//调用示例
        boolean isLoginEd = YlsLoginManager.getInstance().isLoginEd();
```

查询与 PBX 服务器的连接状态

```
/**
 * 判断 Linkus SDK 是否与 PBX 服务器相连接
 * @return 返回值，表示连接状态；true：已连接，false：未连接
 */
public synchronized boolean isConnected()
//调用示例
```

```
        YlsLoginManager.getInstance().isConnected();
```

SDK 通知回调

SDK 通知回调可用于监听及处理用户账号登出、与 PBX 重连成功及通话记录变更等事件。

```
YlsBaseManager.getInstance().setSdkCallback(new SdkCallback() {
// 账号登出回调。账号登出事件类型参见代码下方表格
@Override
public void onLogout(int type) {
        context.startActivity(new Intent(context, LoginActivity.class));
        }

// Linus SDK 与 PBX 服务器重连成功回调
@Override
public void onReconnectSuccess() {}

// 通话记录(CDR)变更回调
@Override
public void onCdrChange(int syncResult) {
        EventBus.getDefault().post(new CallLogChangeEvent(syncResult));
        }
        });
```

用户账号登出事件类型

事件	事件码	说明
SdkEventCode.EVENT_USER_RELOGIN	1005	该帐号已在其他设备登录。
SdkEventCode.P_EVENT_DISABLE_LINKUS_APP	20014	Linkus 手机端被禁用。
SdkEventCode.EVENT_LOGIN_LOCKED	1009	该账号已被锁定。
SdkEventCode.EVENT_LOGIN_INFO_ILLEGAL	1010	缓存登录信息错误。
SdkEventCode.EVENT_CACHE_LOGIIN_USER_NOTFOUND	1011	缓存登录信息为空。
SdkEventCode.P_EVENT_LOGIN_MODE_CHANGE	20008	登录模式 (手动登录或缓存登录) 发生变更。
SdkEventCode.P_EVENT_COUNTRY_IP_LIMIT	20083	禁止访问所请求的 IP 地址 (PBX 已启用 国家地区 IP 访问防御)。
SdkEventCode.P_EVENT_LICENSE_EXPIRE	20093	Linkus SDK 服务不可用 (PBX

事件	事件码	说明
		未购买该服务或服务已过期)。
SdkEventCode.P_EVENT_SDK_STATUS_CHANGE	20153	Linkus SDK 的可用状态发生变更 (PBX 上禁用 Linkus SDK 功能)。
SdkEventCode.P_EVENT_SDK_ACCESSKEY_CHANGE	20154	Linkus SDK 的 **访问密钥** 发生变更 (PBX 上刷新了 Linkus SDK 的 **访问秘钥**)。

通话功能

本文介绍与 Linkus Android SDK 通话相关的功能及实现方法。

发起呼叫

```
/**
 *
 * @param callNumber：被叫号码
 * @param netWorkAvailable：网络是否可用；true:可用，false：不可用
 * @return
 */
public void makeNewCall(String callNumber, boolean netWorkAvailable)
//调用方式如下
        YlsCallManager.getInstance().makeNewCall(number, netWorkAvailable);
```

接听来电

```
YlsCallManager.getInstance().answerCall(callId);
```

拒接来电

```
YlsCallManager.getInstance().answerBusy(context, callId);
```

挂断通话

```
YlsCallManager.getInstance().hangUpCall(context, callId);
```

保持通话

```
YlsCallManager.getInstance().holdCall(inCallVo);
```

恢复通话

```
YlsCallManager.getInstance().unHoldCall(getContext(), inCallVo);
```

咨询转接通话

发起咨询转接

```
YlsCallManager.getInstance().makeTransferCall(context, calleeNam
e, number, trunkName, route, object);
```

确认咨询转接

```
YlsCallManager.getInstance().confirmTransfer(App.getInstance().g
etContext());
```

盲转接通话

```
YlsCallManager.getInstance().blindTransferCall(context, callOutNumber);
```

静音 / 取消静音通话

```
/**
 * @param inCallVo
 */
public void mute(InCallVo inCallVo)
```

通话录音

```
/**
 *
 * @param vo
 * @return
 */
public int record(InCallVo vo)
```

发送 DTMF

```
/**
 *
 * @param callId
 * @param recordCode
 * @return
 */
```

```
public int sendDtmf(int callId, String recordCode)
```

通话质量检测

```
CallQualityVo callQualityVo =
 YlsCallManager.getInstance().getCallQuality();
```

通话状态回调

```
YlsCallManager.getInstance().setCallStateCallback(new CallStateCallback() {
//通话状态变化回调
@Override
public void onCallStateChange(CallStateVo callStateVo) {
        EventBus.getDefault().post(new CallStateEvent(callStateVo));
        }

//通话质量等级变化回调
@Override
public void onNetWorkLevelChange(int callId, int networkLevel) {
        EventBus.getDefault().postSticky(new NetWorkLevelEvent(callId,
 networkLevel));
        }

//网络连接变化回调
@Override
public void onConnectChange() {
        EventBus.getDefault().postSticky(new ConnectionChangeEvent());
        }

//录音状态变化回调
@Override
public void onRecordChange(boolean isRecording) {
        EventBus.getDefault().post(new RecordEvent(isRecording));
        }

        });
```

通话 UI 界面回调

```
YlsCallManager.getInstance().setActionCallback(new ActionCallback() {

//通话结束回调
@Override
public void onFinishCall() {
```

```
        finishAllCall(context);
        }

//来电弹窗回调
@Override
public void onNewCall() {
        jump2CallActivity(context);
        }

//来电等待回调
@Override
public void onCallWaiting() {
        EventBus.getDefault().post(new CallWaitingEvent());
        SoundManager.getInstance().startPlay(context,
 YlsConstant.SOUND_CALL_WAITING_TYPE);
        }

//未接来电回调
@Override
public void onMissCallClick() {

        }

//通话结束后停止前台服务的回调 （Android 11 以上的通话在后台运行时，需要开启前台服务）
@Override
public void onStopMicroPhoneService() {

        }

//音频路由弹窗消失回调
@Override
public void onDismissPopupView() {
        dismissPopupView();
        }

//音频路由变更回调
@Override
public void onNotifyAudioChange() {
        notifyAudioChange();
        }
        });
```

多方通话

Linkus Android SDK 支持最多五方通话。本文介绍与多方通话相关的功能及实现方法。

发起多方通话

```
/**
 * 发起多方通话
 */
public void makeMultipartyCall(String number, String trunkName, String
 route, Activity activity, Object obj)
```

移除单个成员

```
/**
 * 将单个成员从当前的多方通话中移除
 *
 * @return
 */
public void hangUpSingleCall(Context context, int callId)
```

静音或取消静音单个成员

```
/**
 * 在多方通话中为单个成员进行静音或取消静音操作
 */
public void muteSingleMember(InCallVo inCallVo)
```

查询多方通话的相关信息

```
/**
 * 获取多方通话中所有通话的 callID 数组
 */
public int[] getCallIdArrays()

/**
 * 获取多方通话中所有被静音的通话的 callID 数组
 */
public int[] getMuteArrays()

/**
 * 获取多方通话中所有被保持通话的 callID 数组
 */
public int[] getHoldArrays()
```

```
/**
 * 判断当前通话是否处于多方通话中
 *
 * @return
 */
public boolean isInMultipartyCall()

/**
 * 设置当前通话是否处于多方通话中
 *
 * @param inMultipartyCall
 */
public void setInMultipartyCall(boolean inMultipartyCall)

/**
 * 查询多方通话内是否所有通话都处于保持状态
 *
 * @return
 */
public boolean isInMultipartyHold()

/**
 * 获取多方通话保持通话的开始时间
 *
 * @return
 */
public long getMultipartyHoldStartTime()

/**
 * 查询多方通话内是否全体静音
 *
 * @return
 */
public boolean isMultipartyMute()

/**
 * 是否要将多方通话中的所有成员静音
 *
 * @param multipartyMute
 */
public void setMultipartyMute(boolean multipartyMute)

/**
 * 获取多方通话开始时间
```

```
 *
 * @return
 */
public long getMultipartyCallStartTime()

/**
 * 查询是否达到多方通话上限（4 通）
 *
 * @return
 */
public boolean reachMultiPartyCallsLimit()

/**
 * 判断多方通话是否在录音
 *
 * @return
 */
public boolean isMultipartyCallRecord(LinkedList<InCallVo> list)

/**
 * 判断多方通话的录音是否可用
 *
 * @return
 */
public boolean isMultiPartyCallRecordAble()

/**
 * 判断多方通话的录音是否禁用
 *
 * @return
 */
public boolean isMultiPartyCallAlwaysRecordDisable()
```

会议室通话

本文介绍与 Linkus Android SDK 会议室通话相关的功能及实现方法。

初始化会议室通话功能

要使用会议室通话，你需要在项目的 Application 类的主进程调用以下方法初始化该功能。

```
YlsConferenceManager.getInstance().setConferenceCallback(context, new
 ConferenceCallback() {
@Override
public void onConferenceException(ConferenceVo conferenceVo) {
```

```
        //异常会议室通话回调
        EventBus.getDefault().postSticky(new
 ConferenceExceptionEvent(conferenceVo));
        }

@Override
public void onConferenceStatusChange(String conferenceId, String number,
 int status) {
        //会议室成员状态回调
        EventBus.getDefault().post(new ConferenceStatusEvent(conferenceId,
 number, status));
        }
        });
```

发起会议室通话

```
/**
 *
 * @param context
 * @param
 conferenceName：会议室名称。名称不能包
含  、!、$、(、)、/、#、;、,、[、]、″、=、<、>、&、\、’、```、^、%、@、{、}、|、空格，且字
符长度不能超过 63。
 * @param memberArray：会议室通话成员数组。
 * @param requestCallback
 */
public void startConference(Context context, String conferenceName,
 String[] memberArray, RequestCallback requestCallback)
```

管理会议室通话成员

```
/**
 * 会议室通话成员小于9个时，可调用此方法添加 ″新增成员″ 选项
 *
 * @param memberList
 */
public void addNullMember(List<ConferenceMemberVo> memberList)

/**
 * 此方法只针对会议室通话的主持人
 * 会议室通话成员小于9个时，调用此方法添加 ″新增成员″ 选项
 * 会议室通话成员大于2个时，调用此方法添加 ″删除成员″ 选项
 * @param memberList
 */
public void addNullMemberByAdmin(List<ConferenceMemberVo> memberList)
```

```
/**
  * 当会议室通话成员达到上限（9方）时，调用此方法移除 ″新增成员″ 选项
  *
  * @param memberList
  */
public void removeNullMember(List<ConferenceMemberVo> memberList)
```

管理会议室通话

```
/**
 *
 * 会议室通话中，主持人对所有成员执行 静音/取消静音 操作
 * @param conferenceId：会议室通话 ID
 * @param member：会议室通话成员
 * @param isMute：是否要将所有成员静音
 */
public ResultVo muteAllConferenceMemberBlock(String conferenceId, String
 member, boolean isMute)

/**
 *
 * 会议室通话中，主持人对指定成员执行 静音/取消静音 操作
 * @param conferenceId：会议室通话 ID
 * @param number：需要静音或取消静音的成员的号码
 * @param isMute：是否要将指定成员静音
 */
public ResultVo muteConferenceMemberBlock(String conferenceId, String
 number, boolean isMute)

/**
 *
 * 主持人将指定成员从当前的会议室通话中移除
 * @param conferenceId：会议室通话 ID
 * @param number：要从当前会议室通话中移除的成员的号码
 */
public ResultVo kickConferenceMemberBlock(String conferenceId, String
 number)

/**
 *
 * 邀请新成员加入当前的会议室通话
 * @param conferenceId：会议室通话 ID
 * @param number：要邀请的成员的号码
 */
```

```
public ResultVo inviteConferenceMemberBlock(String conferenceId, String
 number)

/**
 *
 * 之前已邀请的成员未加入会议室通话时，可调用此方法再次邀请成员
 * @param conferenceId：会议室通话 ID
 * @param number：要邀请的成员的号码
 */
public ResultVo reInviteConferenceMemberBlock(String conferenceId, String
 number)

/**
 *
 * 结束当前会议室通话
 * @param context
 * @param callId
 * @param conferenceVo
 * @param callback
 * @return
 */
public void endConferenceBlock(Context context, int callId, ConferenceVo
 conferenceVo, RequestCallback callback)

/**
 *
 * 重新连接至因异常情况而被中断的会议室通话
 * @param context
 * @param conferenceId：异常会议室通话的 ID
 * @param member：会议室通话的成员
 * @return
 */
public ResultVo returnConferenceBlock(Context context, String conferenceId,
 String member)
```

获取会议室通话记录

```
conferenceModelList =
 YlsConferenceManager.getInstance().getConferenceList();
```

删除会议室通话记录

```
/**
 * 删除指定的会议室通话记录
 * @param conferenceId：需删除记录的会议室通话的 ID
```

```
 */
public void deleteConferenceLog(String conferenceId)

/**
 * 删除所有的会议室通话记录
 */
public void deleteAllConferenceLog()
```

设置 / 查询会议室通话结束时间

```
/**
 * 设置会议室通话的结束时间
 * @param endConferenceTime
 */
public void setEndConferenceTime(long endConferenceTime)

/**
 * 查询会议室通话的倒计时时间
 * 只有倒计时为负数时可以发起新的会议室通话
 * @return
 */
public long getCountDownTime()
```

设置 / 查询会议室通话缓存信息

```
/**
 * 设置当前会议室通话缓存信息(名称、成员等信息)
 * @param conferenceVo
 */
public void setConferenceVo(ConferenceVo conferenceVo)

 /**
 * 查询当前会议室通话缓存
 * @return
 */
public ConferenceVo getConferenceVo()
```

重连至异常的会议室通话

```
/**
 * 由于网络不稳定导致当前会议室通话中断时，可调用此方法重新连接到该会议室通话
 * @param conferenceId：异常会议室通话的 ID
 * @param member：会议室通话的成员
 * @return
```

```
 */
public ResultVo returnConferenceBlock(String conferenceId, String member)
```

推送消息

本文介绍与 Linkus Android SDK 推送通知相关的功能及实现方法。

设置推送消息

```
/**
 *
 * @param mode：推送证书所属的平台; huawei, xiaomi, firebase,
 GETUI,vivo,honor,oppo
 * @param token：推送证书的鉴权信息
 * @param requestCallback
 * @return
 */
public void setPushInfo(String mode, String token, RequestCallback
 requestCallback)
//调用方法示例
YlsBaseManager.getInstance().setPushInfo("GETUI", clientid, new
 RequestCallback() {

// 设置成功回调
@Override
public void onSuccess(Object result) {
        }

// 设置失败回调
@Override
public void onFailed(int code) {
        }

// 设置异常回调
@Override
public void onException(Throwable exception) {
        }
        });
```

推送消息处理

```
String data = new String(payload);
JSONObject jsonObject = null;
try {
```

```
    jsonObject = new JSONObject(data);
} catch (JSONException e) {
    e.printStackTrace();
}
YlsCallManager.getInstance().handlerPushMessage(context, jsonObject);
```

通话记录

本文介绍与 Linkus Android SDK 通话记录相关的功能及实现方法。

获取通话记录

```
/**
 * 获取指定数量的通话记录（CDR）
 * @param limit：指定需要获取的通话记录数量
 * @return
 */
public List<CdrVo> getCdrList(int limit);
//调用示例
        List<CdrVo> cdrVoList =
 YlsCallLogManager.getInstance().getCdrList(1000);
```

删除指定通话记录

```
/**
 *
 * @param cdrIds：需要删除的通话记录的 ID，以","分隔多个通话记录 ID
 * @return
 */
public int deleteCdr(String cdrIds)
```

删除所有通话记录

```
/**
 *
 * @return
 */
public int deleteAllCdr()
//调用示例
        btnCdrClear.setOnClickListener(v ->
 YlsCallLogManager.getInstance().deleteAllCdr());
```

查询未接来电数量

```
/**
 *
 * @return
 */
public int getMissCallCdrCount();
```

标记所有未读通话记录为已读

```
/**
 *
 * @return
 */
public void readAllCdr()
```

音频配置

本文介绍与 Linkus Android SDK 音频配置相关的功能及实现方法。

音频自动增益开关

```
public void agcSetting(boolean isOpen)
```

回音消除开关

```
public void echoSetting(boolean isOpen)
```

主动降噪开关

```
public void ncSetting(boolean isOpen)
```

Linkus iOS SDK

Linkus iOS SDK 概述

Yeastar P 系列软件版 支持 Linkus SDK，可实现在第三方 iOS 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。本文介绍 Linkus iOS SDK 的使用要求、前提条件、Demo & 项目源码、接入流程、以及功能。

使用要求及前提条件

使用要求

平台 / 环境	要求
PBX 服务器	• **固件版本**：83.12.0.23 或更高 • **订阅服务**：**Linkus SDK 服务** **Note:** 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。
开发环境	• **iOS 版本**：11 及以上 • **Xcode 版本**：14.3 及以上

前提条件

已获取 APNs 证书 (iOS 推送通知服务证书)。

Demo & 源码

在正式集成之前，我们推荐你体验 Linkus iOS SDK 的 Demo 并查阅项目源码，以了解 Linkus iOS SDK 的整体运行框架及流程。

更多信息，前往 Linkus iOS SDK 的 GitHub 仓库。

接入流程及功能

接入流程

1. 启用 Linkus SDK 并绑定 APNs 证书
2. 集成 Linkus iOS SDK

3. 获取 Linkus iOS SDK 登录签名

功能

- 配置
- 登录及登出
- 通话功能
- 会议室通话
- 通话信息
- 复杂通话场景
- 通话记录

Linkus iOS SDK 更新记录

版本 1.2.0

发布日期：2024年4月22日

- 新增兼容 iPad 设备，支持在 iPhone 及 iPad 上同时使用 Linkus SDK。

版本 1.1.1

发布日期：2023年10月11日

- 首次发布 Linkus iOS SDK。通过集成 SDK 与 iOS 项目，快速实现在第三方 iOS 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。

接入 Linkus iOS SDK

启用 Linkus SDK 并绑定 APNs 证书

将 Linkus iOS SDK 接入至 iOS 项目之前，你需要先在 PBX 上启用 Linkus SDK 功能，并绑定 APNs (iOS 推送通知服务) 证书 ，从而保证 iOS 设备能够在集成后接收到来电通知。

使用要求及前提条件

使用要求

确保 PBX 服务器满足以下要求：

- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

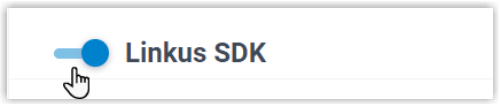
> **Note:**
> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

前提条件

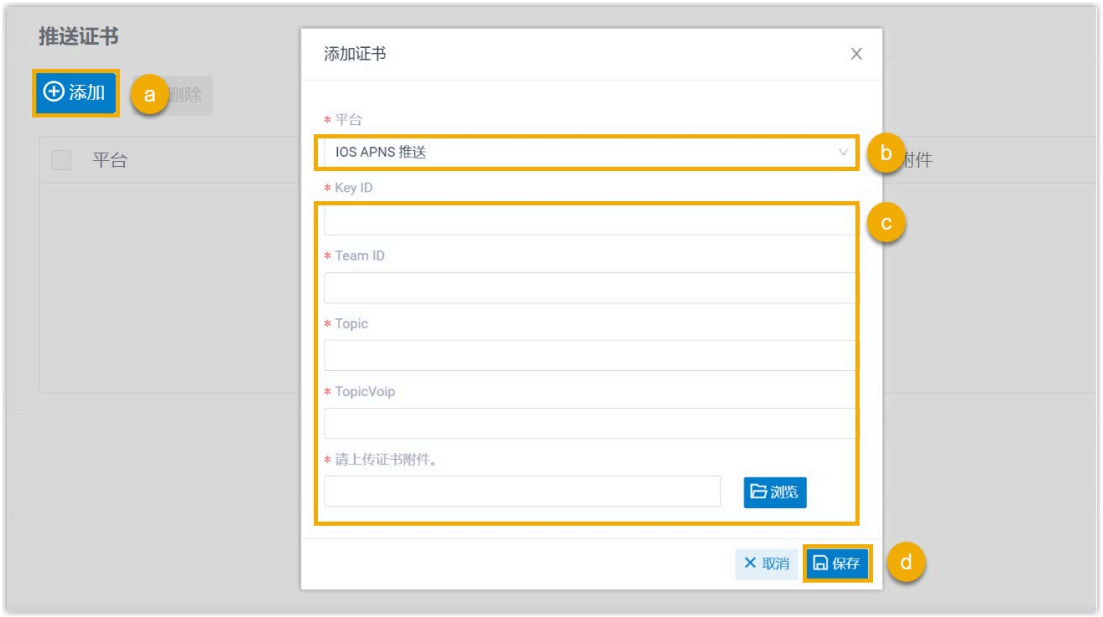
已获取 APNs 证书 (iOS 推送通知服务证书)。

操作步骤

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 打开 **Linkus SDK** 开关并在弹窗中点击 **确认**。



3. 绑定 APNs 证书。



a. 在 **推送证书** 栏，点击 **添加**。
b. 在 **平台** 下拉列表中，选择 **IOS APNS 推送**。

c. 填写证书信息并上传 APNs 证书。
d. 点击 **保存**。
4. 点击 **保存**。

执行结果
你已启用 Linkus SDK 功能且完成 APNs 证书绑定，可以 集成 Linkus iOS SDK。

> **Important:**
> 完成 Linkus iOS SDK 的集成后，你的 iOS 应用将使用 APNs 证书向设备发送通话相关的推送消息，而 Linkus 提供的推送证书 (Linkus 手机端的推送功能) 将不再生效。

集成 Linkus iOS SDK

要集成 Linkus iOS SDK，你需要将 Linkus iOS SDK 导入到 iOS 项目中并对其进行初始化。

使用要求及前提条件

使用要求

确保你的开发环境满足以下要求：

- **iOS 版本**：11 及以上
- **Xcode 版本**：14.3 及以上

前提条件

已 启用 Linkus SDK 并绑定 APNs 证书。

Demo & 源码
在正式集成之前，我们推荐你体验 Linkus iOS SDK 的 Demo 并查阅项目源码，以了解 Linkus iOS SDK 的整体运行框架及流程。

更多信息，前往 Linkus iOS SDK 的 GitHub 仓库。

步骤一、导入 Linkus SDK 至 iOS 工程
选择以下任意一种方法导入 Linkus SDK：

- 使用 CocoaPods 自动集成 Linkus iOS SDK
- 手动集成 Linkus iOS SDK

使用 CocoaPods 自动集成 Linkus iOS SDK

> **Note:**
> 使用此方法集成 Linkus iOS SDK 前，确保你已安装 CocoaPods。更多信息，请参见 CocoaPods 入门指南。

1. 打开 iOS 工程项目的 **Podfile** 文件，添加以下代码并保存：

   > **Note:**
   > 若无 **Podfile** 文件，可从终端进入项目根目录，并运行 `pod init` 命令生成该文件夹。

   ```
   pod 'linkus-sdk'
   ```

2. 在终端内，运行以下命令安装 Linkus iOS SDK。

   ```
   pod install
   ```

   安装完成后，终端会显示 **Pod installation complete!**，且项目文件夹下会生成一个后缀为 **.xcworkspace** 的文件。
3. 使用 Xcode 打开后缀为 **.xcworkspace** 的文件。

手动集成 Linkus iOS SDK

1. 前往 Linkus SDK 的 GitHub 仓库，下载 Linkus iOS SDK 项目文件。
2. 在 Xcode 中，将 **linkus_sdk_iOS.framework** 文件拖入对应的 **Target** 下，并在弹窗中勾选 **Copy items if needed**。
3. 在工程项目中，前往 **Build Phases > Link Binary With Libraries**，并添加以下依赖库：

   ```
   libz.dylib
   libc++.dylib
   libxml2.dylib
   libresolvdylib
   ```

步骤二、初始化 Linkus iOS SDK

1. 在 iOS 工程项目内，打开 **PrefixHeader.pch** 文件并引入头文件。

   ```
   #import <linkus_sdk_iOS/linkus_sdk.h>
   ```

2. 在工程项目 **AppDelegate.m** 的 **application:didFinishLaunchingWithOptions:** 方法中，使用以下代码初始化 Linkus iOS SDK。

```
- (BOOL)application:(UIApplication *)application
 didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [[YLSSDK sharedYLSSDK] initApp];
    return YES;
}
```

后续步骤

向 PBX 服务器请求 SDK 登录签名，用于用户鉴权及登录 Linkus iOS SDK。

更多信息，参见 获取 Linkus iOS SDK 登录签名。

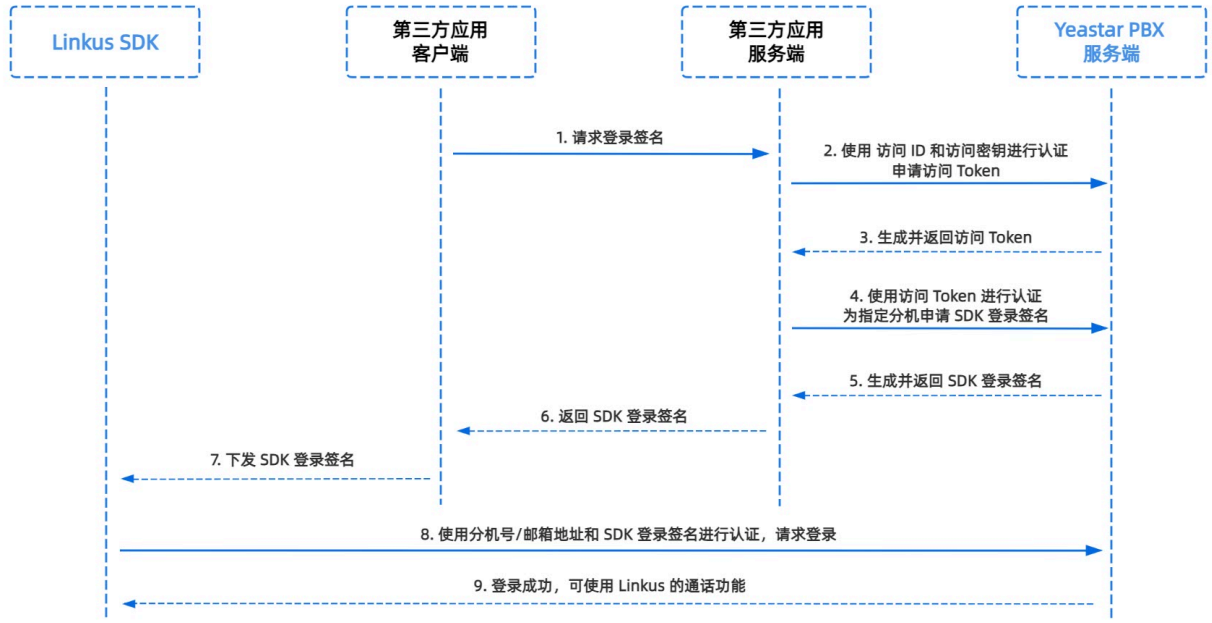
获取 Linkus iOS SDK 登录签名

用户登录 Linkus SDK 时需要使用 SDK 登录签名进行鉴权，而不是登录密码。本文介绍如何通过 OpenAPI 向 PBX 服务器请求用户的 Linkus SDK 登录签名。

前提条件

- 你已 启用 Linkus SDK 并绑定 APNs 证书。
- 你已 集成 Linkus iOS SDK。

Linkus SDK 登录流程



步骤一、在 PBX 获取 Linkus SDK 的访问 ID 及密钥

从 Yeastar P 系列软件版 上获取 Linkus SDK 的访问 ID 及密钥，用于应用服务端向 PBX 进行认证并建立连接。

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 记录 **访问 ID** 及 **访问密钥**。

步骤二、在应用服务端向 PBX 申请访问 Token

通过 OpenAPI，使用 Linkus SDK 的访问 ID 及密钥向 PBX 申请一个访问 Token。访问 Token 会作为 PBX API 接口调用的凭证，用于为分机用户请求 Linkus SDK 的登录签名。

请求 URL

```
POST {base_url}/openapi/v1.0/get_token
```

> **Note:**
> 如需了解 PBX 的 API 请求结构，参见 **请求结构**。

请求参数

参数	是否必填	类型	说明
username	是	String	用户名。在步骤一中获取的 **访问 ID** 作为用户名。
password	是	String	密码。在步骤一中获取的 **访问密钥** 作为密码。

请求示例

```
POST /openapi/v1.0/get_token
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "VXKkvsR2w5H28JAWtBXuJHMTeaQO7Zmf",
 "password": "Yq6yVsBceOZLhnuaGeMUG4U4qXXXXXXX"
```

```
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
access_token_expire_time	Integer	访问 token 有效时长 (**单位**：秒)。
access_token	String	访问 token，即 API 接口调用凭证。所有的请求都需要带一个访问 token。
refresh_token_expire_time	Integer	刷新 token 有效时长 (**单位**：秒)。
refresh_token	String	刷新 token。此刷新 token 可用于获取新的访问 token 和刷新 token。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "access_token_expire_time": 1800,
    "access_token": "EXZMpZAO86mbrKm6rFtgeb3rfcpC9uqS",
    "refresh_token_expire_time": 86400,
    "refresh_token": "SCduGecwbG9jIusiS8FxFUVn3kf0Q9R8"
}
```

步骤三、为分机请求登录签名

通过 OpenAPI， 使用访问 Token 向 PBX 服务器请求分机的登录签名。

请求 URL

```
POST /openapi/v1.0/sign/create?access_token={access_token}
```

请求参数

参数	是否必填	类型	说明
username	是	string	分机号或邮箱地址。
sign_type	是	string	登录签名的类型。 **允许填写的值**：`sdk`
expire_time	否	Integer	登录签名的到期时间戳 (秒)。 `0` 表示不限制登录签名的有效时长。

请求示例

```
POST /openapi/v1.0/sign/create?access_token=EXZMpZAO86mbrKm6rFtg
eb3rfcpC9uqS
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "1000",
 "sign_type": "sdk",
       "expire_time": 0
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
data	Array<Ext_Sign>	分机的登录签名。

Ext_Sign

参数	类型	说明
sign	String	分机的登录签名。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "data":{
        "sign": "ueb3rfcpsiS8FxFUZMpZAO86mbrKmVn3kf0Q9R8"
    }
}
```

执行结果

第三方应用服务端已获取到用户登录 Linkus SDK 所需的签名，且自动将其同步至应用客户端，并下发给 Linkus SDK。

后续操作

使用登录签名向 PBX 服务器进行鉴权并 登录 Linkus iOS SDK。

使用 Linkus iOS SDK

配置

本文介绍了 Linkus iOS SDK 配置相关的功能及实现方法。

```
/**
 *  配置项
 */
+ (instancetype)sharedConfig;

/// 设置应用的名称
@property (nonatomic,copy,nullable) NSString *localizedName;

/// 设置应用图标，尺寸为 40*40
@property (nonatomic,copy,nullable) NSData *iconTemplateImageData;

/// 设置日志文件路径,外层需创建文件夹
@property (nonatomic,copy) NSString *logPath;

/// 设置数据文件路径,外层需创建文件夹
@property (nonatomic,copy) NSString *dataPath;

/// 设置来电铃声
```

```
@property (nonatomic,copy) NSString *comeAudioFileName
 API_AVAILABLE(macos(10.13));

/// 设置通话结束铃声
@property (nonatomic,copy) NSString *hangupAudioFileName;

/// 设置来电等待铃声
@property (nonatomic,copy) NSString *alertAudioFileName;
```

登录及登出

本文介绍与 Linkus iOS SDK 登录及登出相关的功能及实现方法。

手动登录

```
/**
 *   登录接口返回值参见代码下方的表格
 */
- (void)login:(NSString *)account token:(NSString *)token localIP:(NSString
 *)localIP localPort:(NSString *)localPort
     remoteIP:(NSString *)remoteIP remotePort:(NSString *)remotePort
 completion:(void (^)(NSError * _Nullable error))completion;
```

登录接口返回值说明

返回值	说明
1	无法连接 PBX 服务器。
-5	登录请求无响应。
403	用户名或登录签名错误。
405	Linkus 客户端被禁用。
407	该用户账号已被锁定。
416	禁止访问所请求的 IP 地址 (PBX 已启用 国家地区 IP 访问防御)。

自动登录

```
- (void)autoLogin API_AVAILABLE(ios(11.0));
```

用户账号登出

```
- (void)logout:(void (^)(NSError * _Nullable error))completion;
```

SDK 通知回调

```
/**
 *  登录回调
 */
- (void)onLoginStep:(LoginStep)step;

/**
 *  账号强制登出回调
 */
- (void)onKickStep:(KickReason)code;
```

通话功能

本文介绍与 Linkus iOS SDK 通话相关的功能及实现方法。

发起呼叫

```
- (void)startCall:(YLSSipCall *)sipCall completion:(void (^)(NSError
 *error))completion;
```

挂断通话

```
- (void)endCall:(YLSSipCall *)sipCall;
```

保持/ 恢复通话

```
- (void)setHeld:(YLSSipCall *)sipCall;
```

静音/ 取消静音通话

```
- (void)setMute:(YLSSipCall *)sipCall;
```

通话录音

```
- (BOOL)setRecord:(YLSSipCall *)sipCall;
```

咨询转接通话

```
- (void)transferConsultation:(YLSSipCall *)sipCall;
```

盲转接通话

```
- (void)tranforBlind:(YLSSipCall *)sipCall;
```

通话质量检测

```
- (NSString *)callQuality;
```

会议室通话

本文介绍与 Linkus iOS SDK 会议室通话相关的功能及实现方法。

发起会议室通话

```
- (void)createConference:(YLSConfCall *)confCall
               complete:(void(^)(NSError * _Nullable error,NSString
 *confid))complete;
```

管理会议室通话成员

```
- (void)operationConferenceMember:(NSString *)member
                          confid:(NSString *)confid
                   operationType:(int)type
                        complete:(void(^)(NSError * _Nullable
 error))complete;
```

邀请成员加入当前会议室通话

```
- (void)inviteConferenceMembers:(NSArray<NSString *> *)contacts
                        confid:(NSString *)confid
                      complete:(void(^)(NSError * _Nullable
 error))complete;
```

接听会议室来电

```
- (void)conferenceManager:(YLSConfManager *)manager
              callStatus:(YLSSipCall *)sipCall
      reportIncomingCall:(void (^)(void (^controllerBlock)(void),void
 (^errorBlock)(NSError * _Nullable error)))completion;
```

查询会议室通话状态

```
- (void)conferenceManager:(YLSConfManager *)manager callStatus:(YLSSipCall
 *)sipCall;
```

查询当前会议室通话信息

```
- (YLSSipCall *)currentConfSipCall;
```

查询会议室通话成员的状态

```
- (void)conferenceManager:(YLSConfManager *)manager
 conferenceInfo:(YLSConfCall *)confCall;
```

重连至异常的会议室通话

此方法用于处理会议室通话的异常情况，例如由于网络不稳定导致当前会议室通话中断时，可调用此方法重新连接到该会议室通话。

```
- (void)conferenceManager:(YLSConfManager *)manager abnormal:(nullable
 YLSConfCall *)confCall;
```

会议室来电委托

```
- (void)setIncomingCallDelegate:(id<YLSConfManagerDelegate>)delegate;
```

添加会议室通话委托

```
- (void)addDelegate:(id<YLSConfManagerDelegate>)delegate;
```

移除会议室通话委托

```
- (void)removeDelegate:(id<YLSConfManagerDelegate>)delegate;
```

通话信息

本文介绍与 Linkus iOS SDK 通话信息相关的功能及实现方法。

处理 VoIP 来电推送

```
- (void)receiveIncomingPushWithPayload:(NSDictionary *)dictionaryPayload;
```

处理未接来电

```
- (BOOL)didReceiveRemoteNotification:(NSDictionary *)userInfo;
```

查询当前通话信息

```
- (YLSSipCall *)currentSipCall;
```

查询所有通话信息

```
- (NSArray<YLSSipCall *> *)currentSipCalls;
```

查询 SIP 的注册状态

```
- (BOOL)sipRegister;
```

查询录音功能是否可用

```
- (BOOL)enableRecord;
```

查询用户是否有录音权限

```
- (BOOL)adminRecord;
```

来电委托

```
- (void)setIncomingCallDelegate:(id<YLSCallManagerDelegate>)delegate;
```

添加委托

```
- (void)addDelegate:(id<YLSCallManagerDelegate>)delegate;
- (void)addDelegate:(id<YLSCallStatusManagerDelegate>)delegate;
```

移除委托

```
- (void)removeDelegate:(id<YLSCallManagerDelegate>)delegate;
```

SDK 通知回调

```
/**
 *  来电回调
 */
```

```
- (void)callManager:(YLSCallManager *)callManager contact:(void
 (^)(id<YLSContactProtocol> (^block)(NSString *number)))contact
 completion:(void (^)(void (^controllerBlock)(void),void
 (^errorBlock)(NSError *error)))completion;

/**
 *  通话状态变更回调
 */
- (void)callManager:(YLSCallManager *)callManager
 callInfoStatus:(NSMutableArray<YLSSipCall *> *)currenCallArr;

/**
 *  SIP 错误码回调
 */
- (void)callManager:(YLSCallManager *)callManager callFaild:(NSError
 *)error;

/**
 *  通话录音状态回调
 */
- (void)callManagerRecordType:(YLSCallManager *)callManager;

/**
 *  当前通话质量回调
 */
- (void)callManager:(YLSCallManager *)callManager
 callQuality:(BOOL)quality;

/**
 *  呼叫等待回调
 */
- (BOOL)callWaitingSupport;
```

复杂通话场景

本文介绍与 Linkus iOS SDK 的复杂通话场景 (呼叫等待、通话转接、多方通话等) 相关的功能及实现方法。

呼叫等待状态下切换通话

```
- (void)callChange:(YLSSipCall *)waitingCall;
```

添加委托

当存在通话转接、呼叫等待、多方通话等复杂通话场景时，调用此方法添加委托。

```
- (void)addDelegate:(id<YLSCallStatusManagerDelegate>)delegate;
```

移除委托

当存在通话转接、呼叫等待、多方通话等复杂通话场景时，调用此方法移除委托。

```
- (void)removeDelegate:(id<YLSCallStatusManagerDelegate>)delegate;
```

挂断所有通话

在多方通话中挂断所有通话。

```
- (void)callStatusManagerDissmiss:(YLSCallStatusManager
 *)callStatusManager;
```

处理通话中来电

通话过程中收到新来电时，调用此方法回调当前通话的状态 (接听、响铃、挂断、静音等)。

```
- (void)callStatusManager:(YLSCallStatusManager *)callStatusManager
 currentCall:(YLSSipCall *)currentCall;
```

回调复杂通话场景下的通话状态

当存在通话转接、呼叫等待、多方通话等复杂通话场景时，调用此方法回调当前通话的状态。

```
- (void)callStatusManager:(YLSCallStatusManager *)callStatusManager
 currentCall:(YLSSipCall *)currentCall
             callWaiting:(nullable YLSSipCall *)callWaitingCall
 transferCall:(nullable YLSSipCall *)transferCall;
```

通话记录

本文介绍与 Linkus iOS SDK 通话记录相关的功能及实现方法。

获取通话记录

```
- (NSArray<YLSHistory *> *)historys;
```

删除指定通话记录

```
- (void)historyManagerRemove:(NSArray<YLSHistory *> *)historys;
```

删除所有通话记录

```
- (void)historyManagerRemoveAll;
```

标记未读通话记录为已读

```
- (void)checkMissedCalls;
```

SDK 通知回调

```
/**
 *  通话记录变更回调
 */
- (void)historyReload:(NSMutableArray<YLSHistory *> *)historys;

/**
 *  未接来电数量变更回调
 */
- (void)historyMissCallCount:(NSInteger)count;
```

Linkus macOS SDK

Linkus macOS SDK 概述

Yeastar P 系列软件版 支持 Linkus SDK，可实现在第三方 macOS 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。本文介绍 Linkus macOS SDK 的使用要求、Demo & 项目源码、接入流程、以及功能。

使用要求

平台 / 环境	要求
PBX 服务器	• **固件版本**：83.12.0.23 或更高 • **订阅服务**：**Linkus SDK 服务** **Note:** 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。
开发环境	• **macOS 版本**：10.13 及以上 • **Xcode 版本**：14.3 及以上

Demo & 源码

体验 Linkus macOS SDK 的 Demo 并查阅项目源码，以了解 Linkus macOS SDK 的整体运行框架及流程。

更多信息，前往 Linkus macOS SDK 的 GitHub 仓库。

接入流程及功能

接入流程

1. 启用 Linkus SDK
2. 集成 Linkus macOS SDK
3. 获取 Linkus macOS SDK 登录签名

功能

- 配置
- 登录及登出

- 通话功能
- 通话信息
- 复杂通话场景
- 通话记录

Linkus macOS SDK 更新记录

版本 1.0.12
发布日期：2023年10月11日

- 首次发布 Linkus macOS SDK。通过集成 SDK 与 macOS 项目，快速实现在第三方 macOS 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。

接入 Linkus macOS SDK

启用 Linkus SDK

将 Linkus macOS SDK 接入至 macOS 项目之前，你需要先在 PBX 上启用 Linkus SDK 功能。

> **Note:**
> 启用 Linkus SDK 后，Linkus 手机端的推送功能将不再生效。如果你想要在手机上继续接收通话相关的推送消息，参见 Linkus Android SDK 概述 或 Linkus iOS SDK 概述。

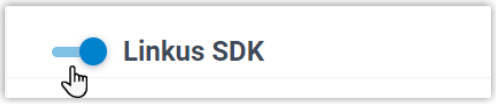
使用要求
确保 Yeastar P 系列软件版 满足以下要求：

- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

> **Note:**
> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

操作步骤

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 打开 **Linkus SDK** 开关并在弹窗中点击 **确认**。



3. 点击 **保存**。

执行结果

你已启用 Linkus SDK 功能，可以 集成 Linkus macOS SDK。

集成 Linkus macOS SDK

要集成 Linkus macOS SDK，你需要将 Linkus macOS SDK 导入到 macOS 项目中并对其进行初始化。

使用要求及前提条件

使用要求

确保你的开发环境满足以下要求：

- **macOS 版本**：10.13 及以上
- **Xcode 版本**：14.3 及以上

前提条件

已 启用 Linkus SDK。

Demo & 源码

在正式集成之前，我们推荐你体验 Linkus macOS SDK 的 Demo 并查阅项目源码，以了解 Linkus macOS SDK 的整体运行框架及流程。

更多信息，前往 Linkus macOS SDK 的 GitHub 仓库。

步骤一、导入 Linkus macOS SDK 至 macOS 工程

选择以下任意一种方法导入 Linkus macOS SDK：

- 使用 CocoaPods 自动集成 Linkus macOS SDK
- 手动集成 Linkus macOS SDK

使用 CocoaPods 自动集成 Linkus macOS SDK

> **Note:**
> 使用此方法集成 Linkus macOS SDK 前，确保你已安装 CocoaPods。更多信息，请参阅 CocoaPods 入门指南。

1. 打开 macOS 工程项目的 **Podfile** 文件，添加以下代码并保存：

   > **Note:**
   > 若无 **Podfile** 文件，可从终端进入项目根目录，并运行 `pod init` 命令生成该文件夹。

   ```
   pod 'linkus-sdk-MacOS'
   ```

2. 在终端内，运行以下命令安装 Linkus SDK。

   ```
   pod install
   ```

   安装完成后，终端会显示 **Pod installation complete!**，且项目文件夹下会生成一个后缀为 **.xcworkspace** 的文件。
3. 使用 Xcode 打开后缀为 **.xcworkspace** 的文件。

手动集成 Linkus macOS SDK

1. 前往 Linkus SDK 的 GitHub 仓库，下载 Linkus macOS SDK 项目文件。
2. 在 Xcode 中，将 **linkus_sdk_MacOS.framework** 文件拖入对应的 **Target** 下，并在弹窗中勾选 **Copy items if needed**。
3. 在工程项目中，前往 **Build Phases > Link Binary With Libraries**，并添加以下依赖库：

   ```
   libcurl.dylib
   libxml2.dylib
   libc++.dylib
   ```

步骤二、初始化 Linkus macOS SDK

1. 在 macOS 工程项目内，打开 **PrefixHeader.pch** 文件并引入头文件。

   ```
   #import <linkus_sdk_MacOS/linkus_sdk.h>
   ```

2. 参考 Demo 源码，初始化 Linkus SDK。

后续步骤

向 PBX 服务器请求 SDK 登录签名，用于用户鉴权及登录 Linkus macOS SDK。

更多信息，参见 获取 Linkus macOS SDK 登录签名。

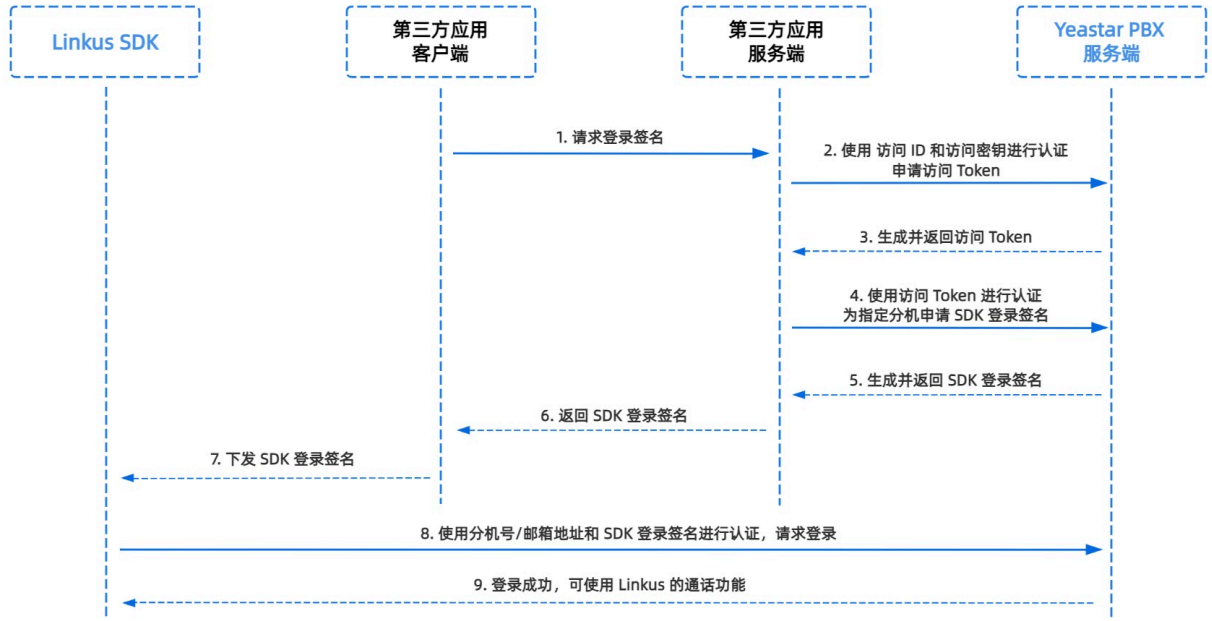
获取 Linkus macOS SDK 登录签名

用户登录 Linkus SDK 时需要使用 SDK 登录签名进行鉴权，而不是登录密码。本文介绍如何通过 OpenAPI 向 PBX 服务器请求用户的 Linkus SDK 登录签名。

前提条件

- 你已 启用 Linkus SDK。
- 你已 集成 Linkus macOS SDK。

Linkus SDK 登录流程



步骤一、在 PBX 获取 Linkus SDK 的访问 ID 及密钥

从 Yeastar P 系列软件版 上获取 Linkus SDK 的访问 ID 及密钥，用于应用服务端向 PBX 进行认证并建立连接。

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 记录 **访问 ID** 及 **访问密钥**。

步骤二、在应用服务端向 PBX 申请访问 Token

通过 OpenAPI，使用 Linkus SDK 的访问 ID 及密钥向 PBX 申请一个访问 Token。访问 Token 会作为 PBX API 接口调用的凭证，用于为分机用户请求 Linkus SDK 的登录签名。

请求 URL

```
POST {base_url}/openapi/v1.0/get_token
```

> **Note:**
> 如需了解 PBX 的 API 请求结构，参见 **请求结构**。

请求参数

参数	是否必填	类型	说明
username	是	String	用户名。在步骤一中获取的 **访问 ID** 作为用户名。
password	是	String	密码。在步骤一中获取的 **访问密钥** 作为密码。

请求示例

```
POST /openapi/v1.0/get_token
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "VXKkvsR2w5H28JAWtBXuJHMTeaQO7Zmf",
 "password": "Yq6yVsBceOZLhnuaGeMUG4U4qXXXXXXX"
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。

参数	类型	说明
		• `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
access_token_expire_time	Integer	访问 token 有效时长 (**单位**：秒)。
access_token	String	访问 token，即 API 接口调用凭证。所有的请求都需要带一个访问 token。
refresh_token_expire_time	Integer	刷新 token 有效时长 (**单位**：秒)。
refresh_token	String	刷新 token。此刷新 token 可用于获取新的访问 token 和刷新 token。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "access_token_expire_time": 1800,
    "access_token": "EXZMpZAO86mbrKm6rFtgeb3rfcpC9uqS",
    "refresh_token_expire_time": 86400,
    "refresh_token": "SCduGecwbG9jIusiS8FxFUVn3kf0Q9R8"
}
```

步骤三、为分机请求登录签名

通过 OpenAPI， 使用访问 Token 向 PBX 服务器请求分机的登录签名。

请求 URL

```
POST /openapi/v1.0/sign/create?access_token={access_token}
```

请求参数

参数	是否必填	类型	说明
username	是	string	分机号或邮箱地址。
sign_type	是	string	登录签名的类型。 **允许填写的值**：`sdk`

参数	是否必填	类型	说明
expire_time	否	Integer	登录签名的到期时间戳 (秒)。 `0` 表示不限制登录签名的有效时长。

请求示例

```
POST /openapi/v1.0/sign/create?access_token=EXZMpZAO86mbrKm6rFtg
eb3rfcpC9uqS
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "1000",
 "sign_type": "sdk",
       "expire_time": 0
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
data	Array<Ext_Sign>	分机的登录签名。

Ext_Sign

参数	类型	说明
sign	String	分机的登录签名。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
```

```
    "data":{
        "sign": "ueb3rfcpsiS8FxFUZMpZAO86mbrKmVn3kf0Q9R8"
    }
}
```

执行结果

第三方应用服务端已获取到用户登录 Linkus SDK 所需的签名，且自动将其同步至应用客户端，并下发给 Linkus SDK。

后续操作

使用登录签名向 PBX 服务器进行鉴权并 登录 Linkus macOS SDK。

使用 Linkus macOS SDK

配置

本文介绍了 Linkus macOS SDK 配置相关的功能及实现方法。

```
/**
 *  配置项
 */
+ (instancetype)sharedConfig;

/// 设置应用的名称
@property (nonatomic,copy,nullable) NSString *localizedName;

/// 设置应用图标，尺寸为 40*40
@property (nonatomic,copy,nullable) NSData *iconTemplateImageData;

/// 设置日志文件路径,外层需创建文件夹
@property (nonatomic,copy) NSString *logPath;

/// 设置数据文件路径,外层需创建文件夹
@property (nonatomic,copy) NSString *dataPath;

/// 设置来电铃声
@property (nonatomic,copy) NSString *comeAudioFileName
 API_AVAILABLE(macos(10.13));

/// 设置通话结束铃声
@property (nonatomic,copy) NSString *hangupAudioFileName;
```

```
/// 设置来电等待铃声
@property (nonatomic,copy) NSString *alertAudioFileName;
```

登录及登出

本文介绍与 Linkus macOS SDK 登录及登出相关的功能及实现方法。

手动登录

```
/**
 *   登录接口返回值参见代码下方的表格
 */
- (void)login:(NSString *)account token:(NSString *)token localIP:(NSString
 *)localIP localPort:(NSString *)localPort
     remoteIP:(NSString *)remoteIP remotePort:(NSString *)remotePort
 completion:(void (^)(NSError * _Nullable error))completion;
```

登录接口返回值说明

返回值	说明
1	无法连接 PBX 服务器。
-5	登录请求无响应。
403	用户名或登录签名错误。
405	Linkus 客户端被禁用。
407	该用户账号已被锁定。
416	禁止访问所请求的 IP 地址 (PBX 已启用 国家地区 IP 访问防御)。

自动登录

```
- (void)autoLogin API_AVAILABLE(ios(11.0));
```

用户账号登出

```
- (void)logout:(void (^)(NSError * _Nullable error))completion;
```

SDK 通知回调

```
/**
 *   登录回调
 */
```

```
- (void)onLoginStep:(LoginStep)step;

/**
 *  账号强制登出回调
 */
- (void)onKickStep:(KickReason)code;
```

通话功能

本文介绍与 Linkus macOS SDK 通话相关的功能及实现方法。

发起呼叫

```
- (void)startCall:(YLSSipCall *)sipCall completion:(void (^)(NSError
 *error))completion;
```

挂断通话

```
- (void)endCall:(YLSSipCall *)sipCall;
```

保持/ 恢复通话

```
- (void)setHeld:(YLSSipCall *)sipCall;
```

静音/ 取消静音通话

```
- (void)setMute:(YLSSipCall *)sipCall;
```

通话录音

```
- (BOOL)setRecord:(YLSSipCall *)sipCall;
```

咨询转接通话

```
- (void)transferConsultation:(YLSSipCall *)sipCall;
```

盲转接通话

```
- (void)tranforBlind:(YLSSipCall *)sipCall;
```

通话质量检测

```
- (NSString *)callQuality;
```

通话信息

本文介绍与 Linkus macOS SDK 通话信息相关的功能及实现方法。

处理未接来电

```
- (BOOL)didReceiveRemoteNotification:(NSDictionary *)userInfo;
```

查询当前通话信息

```
- (YLSSipCall *)currentSipCall;
```

查询所有通话信息

```
- (NSArray<YLSSipCall *> *)currentSipCalls;
```

获取麦克风与扬声器信息

```
- (NSArray<YLSCaptureDevice *> *)audioALLDevice
 API_AVAILABLE(macos(10.13));
```

设置麦克风与扬声器

```
- (void)audioSetDevice:(NSInteger)microphone speaker:(NSInteger)speaker
 API_AVAILABLE(macos(10.13));
```

查询 SIP 的注册状态

```
- (BOOL)sipRegister;
```

查询录音功能是否可用

```
- (BOOL)enableRecord;
```

查询用户是否有录音权限

```
- (BOOL)adminRecord;
```

来电委托

```
- (void)setIncomingCallDelegate:(id<YLSCallManagerDelegate>)delegate;
```

添加委托

```
- (void)addDelegate:(id<YLSCallManagerDelegate>)delegate;
- (void)addDelegate:(id<YLSCallStatusManagerDelegate>)delegate;
```

移除委托

```
- (void)removeDelegate:(id<YLSCallManagerDelegate>)delegate;
```

SDK 通知回调

```
/**
 *  来电回调
 */
- (void)callManager:(YLSCallManager *)callManager contact:(void
 (^)(id<YLSContactProtocol> (^block)(NSString *number)))contact
 completion:(void (^)(void (^controllerBlock)(void),void
 (^errorBlock)(NSError *error)))completion;

/**
 *  通话状态变更回调
 */
- (void)callManager:(YLSCallManager *)callManager
 callInfoStatus:(NSMutableArray<YLSSipCall *> *)currenCallArr;

/**
 *  SIP 错误码回调
 */
- (void)callManager:(YLSCallManager *)callManager callFaild:(NSError
 *)error;

/**
 *  通话录音状态回调
 */
- (void)callManagerRecordType:(YLSCallManager *)callManager;

/**
 *  当前通话质量回调
 */
```

```
- (void)callManager:(YLSCallManager *)callManager
 callQuality:(BOOL)quality;

/**
 *  呼叫等待回调
 */
- (BOOL)callWaitingSupport;
```

复杂通话场景

本文介绍与 Linkus macOS SDK 的复杂通话场景 (呼叫等待、通话转接等) 相关的功能及实现方法。

呼叫等待状态下切换通话

```
- (void)callChange:(YLSSipCall *)waitingCall;
```

添加委托

当存在通话转接、呼叫等待等复杂通话场景时，调用此方法添加委托。

```
- (void)addDelegate:(id<YLSCallStatusManagerDelegate>)delegate;
```

移除委托

当存在通话转接、呼叫等待等复杂通话场景时，调用此方法移除委托。

```
- (void)removeDelegate:(id<YLSCallStatusManagerDelegate>)delegate;
```

处理通话中来电

通话过程中收到新来电时，调用此方法回调当前通话的状态 (接听、响铃、挂断、静音等)。

```
- (void)callStatusManager:(YLSCallStatusManager *)callStatusManager
 currentCall:(YLSSipCall *)currentCall;
```

回调复杂通话场景下的通话状态

当存在通话转接、呼叫等待等复杂通话场景时，调用此方法回调当前通话的状态。

```
- (void)callStatusManager:(YLSCallStatusManager *)callStatusManager
 currentCall:(YLSSipCall *)currentCall
             callWaiting:(nullable YLSSipCall *)callWaitingCall
 transferCall:(nullable YLSSipCall *)transferCall;
```

通话记录

本文介绍与 Linkus macOS SDK 通话记录相关的功能及实现方法。

获取通话记录

```
- (NSArray<YLSHistory *> *)historys;
```

SDK 通知回调

```
/**
 *  通话记录变更回调
 */
- (void)historyReload:(NSMutableArray<YLSHistory *> *)historys;
```

Linkus Windows SDK

Linkus Windows SDK 概述

Yeastar P 系列软件版 支持 Linkus SDK，可实现在第三方 Windows 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。本文介绍 Linkus Windows SDK 的使用要求、可用模块、接入流程以及功能。

使用要求

确保 Yeastar P 系列软件版 满足以下要求：

- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

> **Note:**
> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

可用模块

Linkus Windows SDK 分离了 PBX Web 通话组件中的核心通话逻辑模块及 UI 界面模块。参见下表以了解可用模块的更多信息。

模块	说明	资源链接
Linkus Windows SDK Core	提供 Linkus 客户端的核心通话功能。	• React Demo • 项目源码
Linkus Windows SDK UI	提供预设好的 UI 组件。	• 项目源码

接入流程及功能

接入流程

1. 在 Yeastar P 系列软件版 上启用 Linkus SDK
2. 获取 Linkus Windows SDK 登录签名
3. 集成 Linkus Web SDK 的核心通话功能
4. 可选操作：集成 Linkus Web SDK 的 UI 组件

Linkus Windows SDK Core 功能

- Linkus Windows SDK Core 使用示例
- PBX 功能(PBXOperator)
- 通话功能 (PhoneOperator)
- 通话状态及通话功能 (Session)
- 返回结果 (Result)
- 其它对象的接口及类型
- 音频资源

Linkus Windows SDK 更新记录

版本 1.0.9

发布日期：2023年10月11日

- 首次发布 Linkus Windows SDK。通过集成 SDK 与 Windows 项目，快速实现在第三方 Windows 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。

启用 Linkus SDK

将 Linkus Windows SDK 接入至 Windows 项目之前，你需要先在 PBX 上启用 Linkus SDK 功能。

> **Note:**
> 启用 Linkus SDK 后，Linkus 手机端的推送功能将不再生效。如果你想要在手机上继续接收通话相关的推送消息，参见 Linkus Android SDK 概述 或 Linkus iOS SDK 概述。

使用要求

确保 Yeastar P 系列软件版 满足以下要求：

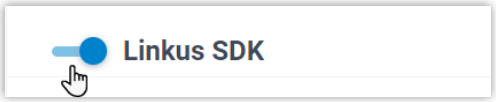
- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

> **Note:**

> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

操作步骤

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 打开 **Linkus SDK** 开关并在弹窗中点击 **确认**。



3. 点击 **保存**。

执行结果

你已启用 Linkus SDK 功能，可 向 PBX 服务器申请 Linkus SDK 登录签名 用于鉴权。

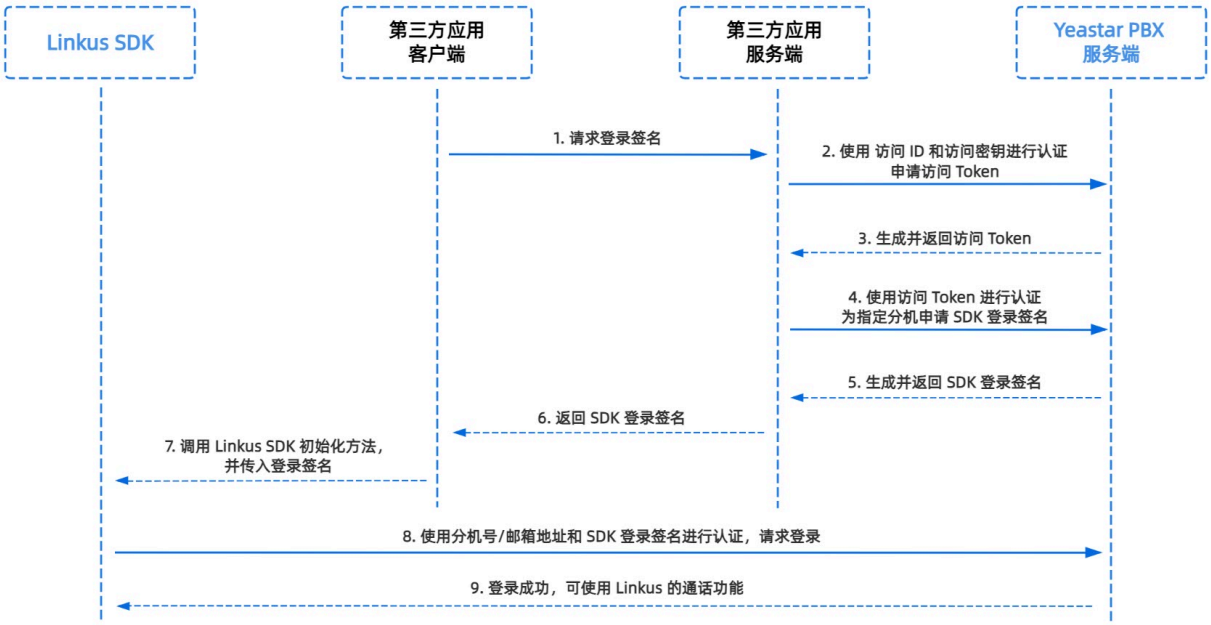
获取 Linkus Windows SDK 登录签名

在初始化 Linkus Windows SDK 时，用户需要使用 SDK 登录签名进行鉴权。本文描述如何通过 OpenAPI 向 PBX 服务器 请求用户的 Linkus SDK 登录签名。

前提条件

已 启用 Linkus SDK。

鉴权流程



步骤一、在 PBX 获取 Linkus SDK 的访问 ID 及密钥

从 Yeastar P 系列软件版 上获取 Linkus SDK 的访问 ID 及密钥，用于应用服务端向 PBX 进行认证并建立连接。

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 记录 **访问 ID** 及 **访问密钥**。

步骤二、在应用服务端向 PBX 申请访问 Token

通过 OpenAPI，使用 Linkus SDK 的访问 ID 及密钥向 PBX 申请一个访问 Token。访问 Token 会作为 PBX API 接口调用的凭证，用于为分机用户请求 Linkus SDK 的登录签名。

请求 URL

```
POST {base_url}/openapi/v1.0/get_token
```

> **Note:**
> 如需了解 PBX 的 API 请求结构，参见 **请求结构**。

请求参数

参数	是否必填	类型	说明
username	是	String	用户名。在步骤一中获取的 **访问 ID** 作为用户名。
password	是	String	密码。在步骤一中获取的 **访问密钥** 作为密码。

请求示例

```
POST /openapi/v1.0/get_token
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "VXKkvsR2w5H28JAWtBXuJHMTeaQO7Zmf",
 "password": "Yq6yVsBceOZLhnuaGeMUG4U4qXXXXXXX"
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
access_token_expire_time	Integer	访问 token 有效时长 (**单位**：秒)。
access_token	String	访问 token，即 API 接口调用凭证。所有的请求都需要带一个访问 token。
refresh_token_expire_time	Integer	刷新 token 有效时长 (**单位**：秒)。
refresh_token	String	刷新 token。此刷新 token 可用于获取新的访问 token 和刷新 token。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "access_token_expire_time": 1800,
    "access_token": "EXZMpZAO86mbrKm6rFtgeb3rfcpC9uqS",
    "refresh_token_expire_time": 86400,
    "refresh_token": "SCduGecwbG9jIusiS8FxFUVn3kf0Q9R8"
}
```

步骤三、为分机请求登录签名

通过 OpenAPI， 使用访问 Token 向 PBX 服务器请求分机的登录签名。

请求 URL

```
POST /openapi/v1.0/sign/create?access_token={access_token}
```

请求参数

参数	是否必填	类型	说明
username	是	string	分机号或邮箱地址。
sign_type	是	string	登录签名的类型。 **允许填写的值**：`sdk`
expire_time	否	Integer	登录签名的到期时间戳 (秒)。 `0` 表示不限制登录签名的有效时长。

请求示例

```
POST /openapi/v1.0/sign/create?access_token=EXZMpZAO86mbrKm6rFtg
eb3rfcpC9uqS
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "1000",
 "sign_type": "sdk",
       "expire_time": 0
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • 0：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • SUCCESS：请求成功。 • FAILURE：请求失败。
data	Array<Ext_Sign>	分机的登录签名。

Ext_Sign

参数	类型	说明
sign	String	分机的登录签名。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "data":{
        "sign": "ueb3rfcpsiS8FxFUZMpZAO86mbrKmVn3kf0Q9R8"
    }
}
```

执行结果

第三方应用服务端已获取初始化 Linkus Windows SDK 所需的签名，且自动将其同步至应用客户端，并下发给 Linkus Windows SDK。

后续操作

使用登录签名向 PBX 服务器鉴权并 集成并初始化 Linkus Windows SDK Core。

Linkus Windows SDK Core

集成 Linkus Windows SDK Core

Linkus Windows SDK Core 提供核心的通话功能 (不带 UI 组件)，本文介绍如何将 Linkus Windows SDK Core 集成到 Windows 项目。

前提条件

- 已 在 Yeastar P 系列软件版 上启用 Linkus SDK。
- 已 获取 Linkus Windows SDK 登录签名。

Demo & 源码

在正式集成之前，我们推荐你体验 Linkus Windows SDK Core 的 React Demo，并查阅 项目源码 以了解 Linkus Windows SDK Core 的整体运行框架及流程。

支持的模块化格式

Linkus Windows SDK Core 支持 **UMD** 、**CJS**、**ESM** 及 **IIFE** 四种模块化格式，你可以根据实际需求选择所需的格式。

> **Note:**
> 如果你想要较小的 SDK 体积，或现有项目支持 ESM，建议导入 **ESM** 使用。

步骤一、引入 Linkus Windows SDK Core

1. 前往 Linkus Windows SDK Core 的 GitHub 仓库，下载 Linkus Windows SDK Core。
2. 使用以下任意一种方式将 Linkus Windows SDK Core 引入至 Windows 项目。
   - 使用 npm 引入 Linkus Windows SDK Core

     ```
     npm install ys-webrtc-sdk-core
     ```

   - 使用 script 标签方式引入 Linkus Windows SDK Core

     ```
     <script src="./ys-webrtc.umd.js"></script>
     ```

步骤二、初始化 Linkus Windows SDK Core

使用 **init** 方法对 Linkus Windows SDK Core 进行初始化。初始化成功后会得到两个实例化后的 Operator 对象和一个方法。

名称	类型	说明
PBXOperator	对象	包含 PBX 相关的方法和参数，如通话记录查询、用户账号登出等。
PhoneOperator	对象	包含通话相关的方法及参数，如发起呼叫、接听通话、挂断通话等。
destroy	方法	用于销毁 Linkus Windows SDK Core。

方法

```
init({
    params//具体参数参见下表
})
```

参数

参数	类型	是否必填	说明
username	string	是	分机号或邮箱地址。
secret	string	是	用户的 Linkus SDK 登录签名。
pbxURL	URL | string	是	PBX 的访问地址，需要包含使用协议以及对应的端口。如：`https://192.168.1.1:8088` 或 `https://yeastar.example.com`。
enableLog	boolean	否	是否启用日志输出并将错误日志上报至 PBX。 **取值范围**： • `true`：启用 • `false`：禁用 **Note:** 此功能默认启用。
reRegistryPhoneTimes	number	否	指定可重新注册 SIP UA 的次数，默认无限制。
userAgent	WebPC" | "WebClient	否	Asterisk 系统中的用户代理 (User Agent)，用于标识正在使用该系统的客户端。 默认值为 **WebClient**。
deviceIds	{ cameraId?: string; microphoneId?: string;}	否	指定音视频输入设备的 ID，包含摄像头 ID 及麦克风 ID。

参数	类型	是否必填	说明
disableCallWaiting	boolean	否	是否禁用呼叫等待。 **取值范围**： • `true`：禁用呼叫等待，PBX 设置的呼叫等待时间不生效，且 PBX 只处理单路通话。 • `false`：启用呼叫等待。

示例代码

- 使用 npm 安装并初始化 Linkus SDK Core。

```
import { init } from 'ys-webrtc-sdk-core';

init({
    username: '1000',
    secret: 'sdkshajgllliiaggskjhf',
    pbxURL: 'https://192.168.1.1:8088'
})
    .then(operator => {
        // 获得 PhoneOperator实例、PBXOperator 实例和 destroy 方法
        const { phone, pbx, destroy } = operator;

        // 创建 RTC 实例
        phone.on('newRTCSession', ({callId,session})=>{
            const {status} = session

            // 监听事件
            session.on('confirmed', {callId,session})=>{
                // 通话成功建立，'session.status.callStatus' 变成'talking'
                // 通话界面开始进行计时
            })

        })
        // 监听'startSession'事件
        phone.on('startSession',({callId,session})=>{
            const {status} = session
            if(status.communicationType === 'outbound') {
                // 外线去电
                // 刷新界面显示 'Calling'，表示被叫正在响铃
            }else{
                // 外线来电
```

```
                // 刷新界面显示 'Connecting'
            }

        });

        // 监听来电事件
        phone.on('incoming', ({callId,session})=>{
            const {status} = session
            // 来电弹窗显示来电号码和联系人名称
            // ...
            // 点击接听按钮触发 'answer' 方法和 'startSession' 事件
            phone.answer(status.number);
        });

        // 订阅事件后，开始连接 SIP UA
        phone.start();

        // ...
        // 点击 Call 按钮呼叫 1001
        phone.call('1001')

    })
    .catch(error => {
        console.log(error);
    });
```

- 使用 script 标签方式导入并初始化 Linkus SDK Core。

```
<script src="./ys-webrtc.umd.js"></script>
<script>
  // 加载成功后通过YSWebRTC对象进行初始化
  YSWebRTC.init({
    username: '1000',
    secret: 'sdkshajgllliiaggskjhf',
    pbxURL: 'https://192.168.1.1:8088',
  })
    .then((operator) => {
      // 获得 PhoneOperator和PBXOperator实例和destroy方法
      const { phone, pbx, destroy } = operator;
    })
    .catch((error) => {
      console.log(error);
    });
    </script>
```

Related information

Linkus Windows SDK Core 使用示例

使用 Linkus Windows SDK Core

Linkus Windows SDK Core 使用示例

本文使用简化的代码提供一个示例，展示使用 Linkus Windows SDK Core 发起呼叫及接听来电的流程。

```
import { init, on } from 'ys-webrtc-sdk-core';

init({
    username: '1000',
    secret: 'sdkshajgllliiaggskjhf',
    pbxURL: 'https://192.168.1.1:8088'
})
    .then(operator => {
        // 获得 PhoneOperator 实例、PBXOperator 实例及 destroy 方法
       const { phone, pbx, destroy } = operator;

        // 创建 RTC 实例
        phone.on('newRTCSession', ({callId,session})=>{
            const {status} = session

            // 开始监听 session 中的事件
            session.on('confirmed', {callId,session})=>{
                // 通话成功接通，session.status.callStatus 进入 talking 状态
                // 界面更新，开始通话计时
            })

        })
        // 监听 startSession 事件
        phone.on('startSession',({callId,session})=>{
            const {status} = session
            if(status.communicationType === 'outbound') {
                // 去电
                // 界面更新，对方响铃中
            }else{
                // 来电
                // 界面更新，连接中
            }
```

```
        });

        // 监听来电事件
        phone.on('incoming', (callId,session)=>{
            const {status} = session
            // 界面弹出来电窗口，显示来电号码，来电者姓名
            // ...
            // 点击接听按钮，调用 answer 方法，随后触发 startSession 事件
            phone.answer(status.number);
        });

        // 监听事件后，开始创建 SIP UA 连接
        phone.start();

        // ...
        // 点击呼出按钮，呼叫1001
        phone.call('1001')

    })
    .catch(error => {
        console.log(error);
    });
```

PBX 功能(PBXOperator)

PBXOperator 对象用于实现与 PBX 相关的功能，如通话记录查询、用户账号登出等。本文介绍与 PBX (PBXOperator 对象) 相关功能的属性、方法及事件。

属性

属性	类型	说明
username	string	用户的登录名，即分机号或邮箱地址。
secret	string	用户的登录签名，可通过 OpenAPI 向 PBX 请求获取。
token	string	向 PBX 请求的访问 Token，可通过 OpenAPI 获取。 更多信息，请参见。
url	URL	PBX 的访问地址。
socket	WebSocket	用于实时监听 PBX 运行状态变更及接收消息通知的 socket 实例。
extensionNumber	string	分机号码。

属性	类型	说明
extensionId	string	分机 ID。
extensionName	string	分机姓名。

方法

方法概览

- 初始化
- 监听事件
- 查询通话记录 (CDR)
- 用户账号登出
- 销毁 PBXOperator 对象

初始化

Linkus SDK Core 内部会调用此方法进行初始化。

> **Note:**
> 初始化成功后，如果再次调用此方法，调用不生效且返回 `Promise.reject`。

方法	`init()`
方法参数	空。
返回值	`Promise<Result>`

监听事件

方法	`on(eventName, listener)`
方法参数	• `eventName`：事件名称。 • `listener`：回调函数。
返回值	空。

> **Note:**
> 关于可监听的事件，参见 事件。

查询通话记录 (CDR)

方法

```
cdrQuery(params)
```

方法参数

```
params: {
    page: number; //必填，定义显示第几个页面
    size: number; //必填，定义每页显示几项查询结果
    status?: number; // 可选，指定通话类型。0：所有；1：呼入；2：未接来电；3：呼出
    sortBy: 'time' | 'id'; //必填，定义排序字段
    orderBy?: 'desc' | 'asc'; //可选，定义显示顺序
    filter?: string | null; //可选，定义筛选条件
}
```

返回值

```
Promise
```

返回值字段示例

```
{
    errcode: 0,
    errmsg: "SUCCESS",
    personal_cdr_list: [ // 通话记录列表
        {
            id: 2546, // 通话记录的序号
            date: "Today", // 接听或拨打该通通话的日期
            time: "17:21:39", // 接听或拨打该通通话的时间
            timestamp: 1686561699, // 通话记录时间的时间戳
            number: "1011", // 主叫号码
            number_type: "extension",
            extension: { // 分机信息
                ext_id: 25, // 分机 ID
                ext_num: "1011", // 分机号码
                caller_id_name: "cwt1011", // 分机姓名
                photo: "", // 头像 ID
                status: 0, // 分机的终端在线状态。 0：离线； 1：在线
                presence_status: "available", // 分机的在线状态
                presence_information: "",
                first_name: "wt1011", // 名
                last_name: "c", // 姓
                email_addr: "", // 邮箱地址
                mobile_number: "", // 手机号码
                enb_vm: 0, // 是否启用语音信箱
                vm_total_count: 0, // 语音留言总数
                vm_unread_count: 0, // 未读的语音留言数
                visable: 0,
                im_id: "xxxxx",
                org_list_info: "",
                is_favorite: 0,
                title: "" // 职位
            },
            status: 3, //通话类型。0：所有；1：呼入；2：未接来电；3：呼出
            talk_duration: 23, // 通话被应答到通话结束的时间。
            call_type: "Internal", // 通讯类型。Internal：内线；External：外线
            uniqueid: "1686561699.137", // 通话记录唯一 ID
            disposition: "ANSWERED", // 通话状态。ANSWERED：已接；NO
ANSWER：未接；BUSY：忙；FAILED：失败；VOICEMAIL：语音留言
```

```
            dcontext: "DLPN_DialPlan1010",
            caller_id: "1011", // 主叫号码
            clid: "\"cwt1010\" <1010>"
        }
    ],
    total_number: 63, // 查询的通话记录的总数
}
```

用户账号登出

方法	`logout()`
方法参数	空。
返回值	`Promise`

销毁 PBXOperator 对象

方法	`destroy()`
方法参数	空。
返回值	`void`

事件

监听事件示例

```
pbx.on('eventName', (data) => {}) //eventName：事件名，data：数据类型
```

运行错误通知

事件名	`runtimeError`
数据类型	`PBXResult`
报告参数	`{ code: '', msg: '', }` **Note:** 关于事件的报告参数及说明，参见下表。

报告参数说明

code	msg	说明
-106	LINKUS_DISABLED	Linkus 客户端未启用。
-107	LOGGED_IN_ELSEWHERE	该分机已在其它应用登录。
-108	EXTENSION_DELETED	该分机已被删除。
-109	RE_LOGIN	分机需要重新登录。
-110	SDK_PLAN_DISABLED	未购买 **Linkus SDK 服务**。

通话记录变更通知

该事件触发时需要手动调用查询通话记录 (CDR) 的方法。

事件名	`cdrChange`
数据类型	`cdrNotifyData`
报告参数	`{ ext_num: '1001', personal_cdr: { date:"", //##### time:"", //##### timestamp: 1693201380, // ############## number: "1001", talk_duration:0 // #### } }`

通话功能 (PhoneOperator)

PhoneOperator 对象用于管理通话。每通通话都是通过 `sessions` 属性缓存，该属性本身是一个包含了不同通话实例的地图，即 `Map<string, Session>`，每个通话实例都被表示为一个 `Session`。本文介绍与通话功能 (PhoneOperator 对象) 相关的属性、方法及事件。

属性

属性名	类型	说明
currentSessionID	string	标识当前通话 (Session) 的通话 ID。
reRegistryPhoneTimes	number	指定可重新注册 SIP UA (User Agent) 的次数。

属性名	类型	说明
deviceIds	{ cameraId?: string; microphoneId?: string;}	指定音视频输入设备的 ID，包含摄像头 ID 及麦克风 ID。
sessions	Map<string, Session>	通话 Session 的缓存 Map 对象， key 为 callId。
currentSession	Session	当前通话 Session。
isRegistered	boolean	是否成功注册 SIP UA。
extraHeaders	string[]	额外的 SIP 请求头。
videoPlan	string	获取视频服务状态。
recordPermissions	number	是否有录音权限： • `0`：无权限 • `1`：有暂停 / 恢复录音权限 • `2`：有开始 / 暂停 / 恢复录音权限
incomingList	session[]	来电列表。
isNoneCamera	boolean	是否无摄像头。
isMaxCall	boolean	是否已达最大通话数量。

方法

方法概览

• 监听事件	• 咨询转接通话	• 结束通话
• 开始注册 SIP UA	• 保持通话	• 断开 SIP UA 注册
• 重新注册 SIP UA	• 恢复通话	• 获取全部通话 (sessions 属性)
• 发起呼叫	• 发送 DTMF	• 设置当前通话 (currentSession)
• 拒接来电	• 静音通话	• 获取当前通话 (currentSession)
• 接听来电	• 取消静音通话	• 更新 Session 的静态属性
• 挂断通话	• 开始录音	• 销毁 PhoneOperator 对象
• 盲转接通话	• 暂停录音	

监听事件

方法	`on(eventName:string,listener: (...args: any[]) => void)`
方法参数	• `eventName`：事件名称。 • `listener`：回调函数。
返回值	空。

> **Note:**
> 关于可监听的事件，参见 事件。

开始注册 SIP UA

SIP UA 注册成功后即可呼出和接听电话。

> **Note:**
> 调用此方法之前，确保你已监听需要监听的事件，否则可能会错失某些事件的通知。

方法	`start()`
方法参数	空。
返回值	`this`

重新注册 SIP UA

该方法会在 phone 实例内部调用。外部程序无需调用此方法，以免出现异常。

方法	`reRegister(authorizationUser: string, ha1: string)`
方法参数	• `authorizationUser`：用户名。 • `ha1`：登录密码。
返回值	`this`

发起呼叫

方法	`call(number: string, option?: CallOptions, transferId?: string)`
方法参数	• `number`：被叫号码。 • `option`：可选。指定通话选项。 • `transferId`：可选。咨询转接通话的 ID。

	Note: 当有 transferId 时则为咨询转接通话，异步函数。
返回值	`Promise<Result>`

拒接来电

方法	`reject(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

接听来电

方法	`answer(callId: string, option?: CallOptions, allowNoneCamera: boolean = true)`
方法参数	• `callId`：通话的唯一 ID。 • `option`：可选。指定通话选项。 • `allowNoneCamera`：是否允许无摄像头接听。
返回值	`Promise<Result>`

挂断通话

方法	`hangup(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

盲转接通话

方法	`blindTransfer(callId: string, number: string)`
方法参数	• `callId`：通话的唯一 ID。 • `number`：通话转接对象的号码。
返回值类型	`boolean`

咨询转接通话

方法	`attendedTransfer(callId: string, number: string)`

方法参数	• `callId`：通话的唯一 ID。 • `number`：通话转接对象的号码。
返回值类型	`boolean`

保持通话

方法	`hold(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

恢复通话

方法	`unhold(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

发送 DTMF

方法	`dtmf(callId: string, dtmf: string)`
方法参数	• `callId`：通话的唯一 ID。 • `dtmf`：字符串 (0123456789*#)
返回值类型	`boolean`

静音通话

方法	`mute(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

取消静音通话

方法	`unmute(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

开始录音

方法	`startRecord(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

暂停录音

方法	`pauseRecord(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

结束通话

方法	`terminate(callId: string, type: 'hangup' | 'reject' | 'terminate' = 'terminate')`
方法参数	• `callId`：通话的唯一 ID。 • `type`：通话结束类型。
返回值类型	`boolean`

断开 SIP UA 注册

方法	`disconnect()`
方法参数	空。
返回值类型	`boolean`

获取全部通话 (sessions 属性)

该方法将以数组的形式返回。

方法	`getSession()`
方法参数	空。
返回值类型	`boolean`

设置当前通话 (currentSession)

方法	`setCurrentSession(callId: string)`

方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

获取当前通话 (currentSession)

方法	`getCurrentSession()`
方法参数	空。
返回值	`Session`

更新 Session 的静态属性

方法	`setSessionStaticStatus(callId: string, staticStatus: Partial<StaticCallStatus>, startManualModel?: boolean)`
方法参数	• `callId`：通话的唯一 ID。 • `staticStatus`：Session 的静态属性包括 `name`、`avatar`、`company`。 • `startManualModel`：可选。是否启用手动模式，启用后将不再自动更新静态属性。
返回值类型	`boolean`

销毁 PhoneOperator 对象

此方法将解除所有订阅事件，停止 SIP UA 实例，并删除所有 session。

方法	`destroy()`
方法参数	空。
返回值	空。

事件

事件名称	事件说明	报告参数
connected	SIP 服务连接成功	空
disconnected	SIP 服务断开连接	空
registered	SIP UA 注册成功	空
registrationFailed	SIP UA 注册失败	• `code: number`：错误码 • `msg: string`：错误信息

事件名称	事件说明	报告参数
newRTCSession	创建新的通话实例 (Session)	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
incoming	来电	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
startSession	添加通话实例 (Session) 至存储通话的 Map (sessions) 中	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
recordPermissionsChange	录音权限发生改变	• `0`：无权限 • `1`：有暂停 / 恢复录音权限 • `2`：有开始 / 暂停 / 恢复录音权限
deleteSession	删除通话实例 (Session)	• `callId: string`：通话的唯一 ID • `cause: string`：删除通话实例的原因
isRegisteredChange	SIP UA 注册状态是否发生改变	• `true`：注册状态已改变 • `false`：注册状态未改变

通话状态及通话功能 (Session)

Session 对象为通话实例，用于实现通话状态管理及通话中的相关操作。本文介绍与通话状态及通话功能 (Session 对象) 相关的属性、方法及事件。

属性

属性	类型	说明
RTCSession	RTCSession	通话实例。
callReport	Report	通话质量报告。
agreeChangeVideo	(params: AgreeChangeVideoParamsType) => Promise<boolean>	是否同意切换成视频通话回调函数。如果没有提供该回调函数，将默认拒绝切换。
status	CallStatus	通话状态，包含名字、号码、头像、是否静音等属性。
incomingList	Session[]	来电数组。
timer	TimerType	通话计时器。
localStream	MediaStream	本地媒体流，仅含视频轨，不含音频。
remoteStream	MediaStream	远端媒体流，包含音视频轨。

方法

方法概览

• 停止通话计时	• 咨询转接通话	• 开始录音
• 监听事件	• 保持通话	• 暂停录音
• 拒接来电	• 恢复通话	• 结束通话
• 接听来电	• 发送 DTMF	• 更新通话状态
• 挂断通话	• 静音通话	• 更新 Session 的静态属性
• 盲转接通话	• 取消静音通话	• 销毁 Session 实例

停止通话计时

方法	`stopTimer()`
方法参数	空。
返回值	`this`

监听事件

方法	`on(eventName:string,listener: (...args: any[]) => void)`
方法参数	• `eventName`：事件名称。 • `listener`：回调函数。
返回值	空。

> **Note:**
> 关于可监听的事件，参见 事件。

拒接来电

该方法等价于 **PhoneOperator** 对象的拒接来电方法。

方法	`reject()`
方法参数	空。
返回值	`void`

接听来电

该方法等价于 **PhoneOperator** 对象的接听来电方法。

方法	`answer(option?: CallOptions, allowNoneCamera: boolean = true)`
方法参数	• `option`：指定通话参数。 • `allowNoneCamera`： 是否允许无摄像头接听。
返回值	`Promise<Result>`

挂断通话

该方法等价于 **PhoneOperator** 对象的挂断通话方法。

方法	`hangup()`
方法参数	`callId`：通话唯一标识。
返回值	`void`

盲转接通话

该方法等价于 **PhoneOperator** 对象的盲转接通话方法。

方法	`blindTransfer(number: string)`
方法参数	`number`：通话转接对象的号码。
返回值	`void`

咨询转接通话

该方法等价于 **PhoneOperator** 对象的咨询转接通话方法。

方法	`attendedTransfer(number: string)`
方法参数	`number`：通话转接对象的号码。
返回值	`void`

保持通话

该方法等价于 **PhoneOperator** 对象的保持通话方法。

方法	`hold()`

方法参数	空。
返回值	`void`

恢复通话

该方法等价于 **PhoneOperator** 对象的恢复通话方法。

方法	`unhold()`
方法参数	空。
返回值	`void`

发送 DTMF

该方法等价于 **PhoneOperator** 对象的发送 DTMF 方法。

方法	`dtmf(dtmf: string)`
方法参数	`dtmf`：字符串 (0123456789*#)
返回值	`void`

静音通话

该方法等价于 **PhoneOperator** 对象的静音通话方法。

方法	`mute()`
方法参数	空。
返回值	`void`

取消静音通话

该方法等价于 **PhoneOperator** 对象的取消静音通话方法。

方法	`unmute()`
方法参数	空。
返回值	`void`

开始录音

该方法等价于 **PhoneOperator** 对象的开始录音方法。

方法	`startRecord()`
方法参数	空。
返回值	`void`

暂停录音

该方法等价于 **PhoneOperator** 对象的暂停录音方法。

方法	`pauseRecord()`
方法参数	空。
返回值	`void`

结束通话

该方法等价于 **PhoneOperator** 对象的结束通话方法。

方法	`terminate(type: 'hangup' | 'reject' | 'terminate' = 'terminate')`
方法参数	`type`：通话结束类型。
返回值	`void`

更新通话状态

方法	`setStatus(status: Partial<CallStatus>)`
方法参数	`status`：通话状态对象。
返回值	`this`

更新 Session 的静态属性

方法	`setStaticStatus(staticStatus: Partial<StaticCallStatus>, startManualModel?: boolean)`
方法参数	• `staticStatus`：Session 的静态属性包括 `name`、`avatar`、`company`。 • `startManualModel`：可选。是否启用手动模式，启用后将不再自动更新静态属性。
返回值	`this`

销毁 Session 实例

此方法将解除所有订阅事件，停止 RTCSession。

方法	`destroy()`
方法参数	空。
返回值	空。

事件

事件名称	事件说明	报告参数
callReport	通话质量报告更新事件，每3秒更新一次	• `callId: string`：通话的唯一 ID • `callReport: Report`：通话质量报告
streamAdded	添加媒体流	• `callId: string`：通话的唯一 ID • `communicationType: "outbound" | "inbound"`：通话类型 • `stream: MediaStream`：媒体流
ended	通话结束	• `callId: string`：通话的唯一 ID • `cause: string`：通话结束的原因
failed	通话失败	• `callId: string`：通话的唯一 ID • `cause: string`：通话失败的原因 • `code: number`：错误码
clientError	客户端错误，导致无法发起通话	• `callId: string`：通话的唯一 ID • `cause: string`：客户端错误的原因
reinvite	重新邀请事件，该事件在对端执行咨询转接时触发	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
accepted	收到成功状态响应代码 **200 OK**	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
confirmed	通话正式建立（收到 **ack** 包）	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
statusChange	通话状态变更	• `newStatus`：新通话状态 • `oldStatus`：原通话状态
staticStatusChange	通话的静态属性变更	• `newStatus`：新通话静态属性 • `oldStatus`：原通话状态静态属性

返回结果 (Result)

本文介绍 Linkus Windows SDK Core 返回结果 (Result 对象) 的结构及各个子类的状态码说明。

背景信息

Result 结构

```
{
    code: '',
    msg: '',
}
```

Result 子类

子类	说明	代码范围
CommonResult	通用的返回结果	• 正确代码：0 到 99 • 错误代码：-1 到 -99
PBXResult	与 PBXOperator 对象相关的返回结果	• 正确代码：100 到 199 • 错误代码：-100 到 -199
PhoneResult	与 PhoneOperator 对象相关的返回结果	• 正确代码：200 到 299 • 错误代码：-200 到 -299

CommonResult

错误代码 (COMMON_ERROR)

code	msg	说明
-1	UNKNOWN_ERROR	未知错误。
-2	INVALID_PBX_URL	无效的 PBX 访问地址。
-3	PBX_URL_NOT_HTTPS	PBX 访问地址的传输协议不是 HTTPS。
-4	GET_PRODUCT_FAILED	获取 PBX 的 PRODUCT 接口失败。

成功代码 (COMMON_SUCCESS)

code	msg	说明
0	SUCCESS	成功。

PBXResult

错误代码 (PBX_ERROR)

code	msg	说明
-100	UNKNOWN_ERROR	未知错误。
-101	REGISTRY_FAILED	SIP UA 注册失败。
-102	PBX_NETWORK_ERROR	PBX 接口请求错误 (客户端网络异常)。
-103	PBX_API_ERROR	PBX 接口请求错误 (服务端异常)。
-104	GET_PERSONAL_NOT_FOUND_DATA	PBX 未返回分机信息。
-105	PBX_ALREADY_INITIALIZED	PBX 对象已初始化过，不允许重复初始化。
-106	LINKUS_DISABLED	Linkus 客户端未启用。
-107	LOGGED_IN_ELSEWHERE	该分机已在其它应用登录。
-108	EXTENSION_DELETED	该分机已被删除。
-109	RE_LOGIN	分机需要重新登录。
-110	SDK_PLAN_DISABLED	未购买 **Linkus SDK 服务**。

成功代码 (PBX_SUCCESS)

code	msg	说明
100	SUCCESS	成功。

PhoneResult

错误代码 (PHONE_ERROR)

code	msg	说明
-200	UNKNOWN_ERROR	未知错误。
-201	REGISTRY_FAILED	SIP UA 注册失败。
-202	GET_AGREE_CHROME_USER_MEDIA_ROLE_ERROR	获取媒体流失败 (未获得浏览器授权)。

code	msg	说明
-203	GET_LOCAL_STREAM_ERROR	获取本地媒体流失败。
-204	RE_REGISTRY_MAX_LIMIT_TIMES	已达最大 SIP UA 重新注册次数。
-205	MAX_LIMIT_CALL	已达最大通话数。
-206	GET_LOCAL_MEDIA_INFO_ERROR	获取本地媒体设备错误。
-207	ATTENDED_PARENT_NOT_FOUND	咨询转接通话的父节点未找到。
-208	CALL_TOO_MANY_TIMES	到达一秒钟内的呼出次数上限。
-209	INVALID_NUMBER	无效号码。
-210	CURRENT_CALL_HAS_NOT_CONNECTED	当前有未接通的通话。
-211	NOT_FOUND_CALL_ID	未找到通话 ID。
-290	NOT_FOUND_AUDIO_INPUT_DEVICE	未找到音频输入设备。
-291	NOT_FOUND_VIDEO_INPUT_DEVICE	未找到视频输入设备。

成功代码

code	msg	说明
200	SUCCESS	成功。

其它对象的接口及类型

本文介绍 Linkus Windows SDK Core 所使用的其它对象的接口及类型。

Report 对象

通话质量报告对象。

```
interface Report {
    lksTimestamp: string; // 接听来电或发起呼叫时的时间戳字符串
    lksDate: string; // 接听来电或发起呼叫时的时间。时间格式 YYYY-MM-DD hh:mm:ss
    callId: string; // 通话的唯一 ID
    extension: string; // 分机号码
    iceResult: string; // ice 类型：local（对应host）、
 public（对应srflx或prflx）、 turn（对应relay）
```

```
    lksNetworkType: string; // 网络类型
    lksDeviceType: string; // 客户端类型：目前只有web
    lksDuration: string; // 对应 totalSamplesDuration 属性
    lksAudioCodec: string; // 音频编码
    lksAudioRx: string; // 收到的音频包，对应 packetsReceived 属性
    lksAudioRxLost: string; // 音频丢包率，对应
 packetsLost属性。计算公式：lksAudioRxLost=(packetsLost / (packetsLost +
 packetsReceived)).toFixed(6)
    lksAudioRxMaxjitter: string; // 音频接收端的最大抖动
    lksAudioRxAvgjitter: string; // 音频平均抖动，对应 jitter*1000
    lksAudioTx: string; // 发送的音频包
    lksAudioTxLost: string; // 发送端的音频丢包率
    lksAudioTxMaxjitter: string; // 音频发送端的最大抖动
    lksAudioTxAvgjitter: string; // 音频发送端的平均抖动
    lksVideoCodec: string; // 视频编码
    lksVideoRx: string; // 收到的视频包
    lksVideoRxLost: string; // 视频丢包率， 对应 packetsLost
 属性，计算公式lksVideoRxLost=(packetsLost / (packetsLost +
 packetsReceived)).toFixed(6)
    lksVideoRxMaxjitter: string; // 视频接收端最大抖动
    lksVideoRxAvgjitter: string; // 视频接收端平均抖动
    lksVideoTx: string; // 发送的视频包
    lksVideoTxLost: string; // 发送端的视频丢包率
    lksVideoTxMaxjitter: string; // 视频发送端的最大抖动
    lksVideoTxAvgjitter: string; // 视频发送端的平均抖动
}
```

StaticCallStatus 对象

通话的静态属性对象。

```
interface StaticCallStatus {
    name: string; // 来电者姓名
    avatar: string; // 头像
    company: string; // 公司名
}
```

CallStatus 对象

通话状态对象。

```
interface CallStatus extends StaticCallStatus{
    number: string; // 电话号码或分机号
    callId: string; // 通话的唯一 ID
    communicationType: 'outbound' | 'inbound'; //
 通话类型：outbound：去电，inbound：来电
```

```
    isVideo: boolean; //是否为视频通话
    isRing: boolean; //是否正在响铃中
    isHold: boolean; //该通话是否处于保持状态
    isMute: boolean; //该通话是否处于静音状态
    callStatus: 'ringing' | 'calling' | 'talking' | 'connecting';// 通话状态
    callStartTime: number; // 通话开始时间
    recordStatus: 'stop' | 'recording' |
 'pause'; //通话录音状态。stop：未在录音；recording：正在录音；pause：暂停录音
    isTransfer: boolean; // 是否为咨询转接通话
    transferParent?:
 TransferParentType; //当是咨询转接通话时，指定咨询转接的父通话
    isConference: boolean; // 是否为语音会议室通话
}

type TransferParentType = {
    // 咨询转接的父通话
    callId: string; //通话的唯一 ID
    avatar: string; // 头像
    name: string; // 通话转接对象的名称
    number: string; // 通话转接对象的号码
    callDuration: number; // 通话被接起到挂断的时长
    holdDuration: number; // 通话被保持的时长
}
```

TimerType 类型

通话计时类型。

```
type TimerType = {
    ringDuration: number; // 响铃时长
    callingDuration: number; // 呼叫时长：拨号开始到通话被应答的时间。
    callDuration: number; // 通话被接起到挂断的时长
    holdDuration: number; // 通话被保持的时长
}
```

音频资源

本文介绍 Linkus Windows SDK Core 所提供的音频资源及使用方法。

背景信息

Linkus Windows SDK Core 项目文件的 **assets** 目录包含了可用音频的源文件及 base 64 编码后的字符串。

- **音频源文件**：存储于 **sounds** 文件夹内。

- **base64 编码字符串**：存储于 **sound.js** 文件内。

参见下表以了解所提供的音频资源。

名称	说明	名称	说明
Ring	来电铃声	DTMF04	按键 4 提示音
Callend	通话结束提示音	DTMF05	按键 5 提示音
Callwaiting	呼叫等待提示音	DTMF06	按键 6 提示音
ringback	回铃音	DTMF07	按键 7 提示音
DTMF00	按键 0 提示音	DTMF08	按键 8 提示音
DTMF01	按键 1 提示音	DTMF09	按键 9 提示音
DTMF02	按键 2 提示音	DTMFStar	按键 * 提示音
DTMF03	按键 3 提示音	DTMFPound	按键 # 提示音

使用音频资源

Linkus Windows SDK Core 支持通过以下方式引入并使用音频资源。

- 直接引用音频源文件。
- 通过 base64 编码引用音频资源。更多信息，参见下方示例代码。

```
import sounds from '/assets/sounds';
const { Ring, Callend } = sounds;
const audio = new Audio(Ring);
audio.play();

// 改变 audio.src 的值，播放不同的音频
audio.src = Callend;
audio.play();
```

Linkus Windows SDK UI

集成 Linkus Windows SDK UI

Linkus Windows SDK 抽离了 PBX Web 通话组件中的 UI 界面及通话逻辑。Linkus Windows SDK UI 是预设好的 UI 组件，你可以将其与 Windows 项目集成，并搭配 Linkus Windows SDK Core 实现带有 UI 界面的通话功能。

前提条件

- 已 获取 Linkus Windows SDK 登录签名。
- 已 集成 Linkus Windows SDK Core。

步骤一、导入 Linkus Windows SDK UI

1. 前往 Linkus Windows SDK UI 的 GitHub 仓库，下载 Linkus Windows SDK UI。
2. 使用以下任意一种方式将 Linkus Windows SDK UI 引入至你的项目。
   - 使用 npm 安装 Linkus Windows SDK UI

     ```
     npm install ys-webrtc-sdk-ui --save
     ```

   - 使用 script 标签方式引入 Linkus Windows SDK UI

     ```
     <script src="./ys-webrtc-sdk-ui.js"></script>
     ```

步骤一、初始化 Linkus Windows SDK UI

使用 `init` 方法初始化 Linkus Windows SDK UI，渲染 UI 组件。

方法

```
init (container,{rtcOption})
```

方法参数

- `container`：用于渲染 UI 组件，类型为 `HTMLElement`。
- `rtcOption`：初始化选项，具体参数参见下表。

属性	类型	是否必填	说明
username	string	是	分机号或邮箱地址。
secret	string	是	用户的 Linkus SDK 登录签名。
pbxURL	URL | string	是	PBX 的访问地址，需要包含使用的协议以及对应的端口。 如：`https://192.168.1.1:8088`。
enableLog	boolean	否	是否启用日志输出并将错误日志上报至 PBX。 **取值范围**： ◦ `true`：启用

属性	类型	是否必填	说明
			◦ `false`：禁用 **Note:** 此功能默认启用。
reRegistryPhoneTimes	number	否	指定可重新注册 SIP UA (User Agent) 的次数。
deviceIds	{ cameraId?: string; microphoneId?: string; speakerId:string; volume:number }	否	指定音视频输入设备的 ID (包含摄像头 ID、麦克风 ID 和扬声器 ID)，并设置音量 (包含通话、来电铃声和拨号盘)。音量默认值为 0.6，取值范围为 0 到 1。
incomingOption	{ style?: React.CSSProperties; class?: string; }	否	调整 Incoming 组件样式。
dialPanelOption	{ style?: React.CSSProperties; class?: string; hideVideoBtn?: boolean }	否	调整拨号盘组件样式，并设置是否隐藏视频按钮。 **Note:** `hideVideoBtn` 参数默认值为 `false`。如需隐藏视频按钮，可将其设置为 `true`。
sessionOption	SessionOption	否	调整通话窗口组件位置和尺寸。
hiddenIncomingComponent	boolean	否	隐藏来电弹屏组件。
hiddenDialPanelComponent	boolean	否	隐藏拨号盘组件。
disableCallWaiting	boolean	否	是否禁用呼叫等待。 **取值范围**： ◦ `true`：禁用呼叫等待，PBX 设置的呼叫等待时间不生效，且 PBX 只处理单路通话。 ◦ `false`：启用呼叫等待。

属性	类型	是否必填	说明
intl	{ local: string; messages: Record<string, string>}	否	国际化 (多语言) 设置。

SessionOption 类型说明

```
type SessionOption = {
    sessionSetting?: {
        width?: number;
        height?: number;
        miniWidth?: number;
        miniHeight?: number;
        x?: number;
        y?: number;
    };
    style?: React.CSSProperties;
    class?: string;
}
```

intl (国际化设置) 类型说明

- `local`：当前使用语言的名称，用短横线连接，如 `en-US`、`zh-CN`。
- `message`：可配置的文案内容，以键值对形式呈现。参见以下内容以了解具体的配置项及默认值。

```
{
    "common.cancel": "Cancel", // ########
    "common.confirm": "Confirm", //
 ########

  "dial_panel.input.placeholder": "Please
 input number", // "####" ####

  "dial_panel.tip.connect_failed": "Failed
 to connect to server, you cannot initiate
 or answer a call. Trying to reconnect to
 the server.", //############
    "incoming.btn.hang_up": "Hang up", //
 "####" ####
    "incoming.btn.video": "Video", // ####
 "####" ####
```

```
    "incoming.btn.audio": "Audio", // ##
 "####" ####
    "session.calling": "Calling...", //
 "###" ##
    "session.ringing": "Ringing...", //
 "#####" ##
    "session.talking": "Talking...", //
 "###" ##

  "session.connecting": "Connecting...", /
/ "###" ##
    "session.hang_up": "End Call", //
 "####" ##
    "session.new_call": "New call", //
 "###" ##
    "session.record": "Record", // "##" ##
    "session.mute": "Mute", // "####" ##
    "session.video": "Video", // "####" ##
    "session.hold": "Hold", // "####" ##
    "session.resume": "Resume", // "####"
 ##
    "session.dialpad": "Dialpad", //
 "####" ##
    "session.transfer": "Transfer", //
 "####" ##
    "session.attended_transfer": "Attended
 Transfer", // "####" ##
    "session.blind_transfer": "Blind
 Transfer", // "###" ##

  "session.modal.change_to_video.title": "
Request", // "##" ###########

  "session.modal.change_to_video.content":
 "{0} invites you switch to video call. Do
 you accept?", // "#########" #######"{0}"
 #############
    "session.error.client_error": "Client
 Error: {0}", // "#####" ###"{0}" ####
    "session.tip.recording": "Recording
 the Audio...", // "#####" ##
    "session.tip.pause": "The recording is
 paused.", // "#######" ##

  "session.tip.can_no_use_video": "Unlock
```

```
  this feature with Ultimate Plan.", //
  "##########" ##
     "error.code_200": "Unknown Error.", //
  "####" ####
     "error.code_202": "No available
  communication device found.(no
  permissions)", // "########" ####
     "error.code_205": "Call failed. Cannot
  process more new calls now.", // "####
  (#######)" ####
     "error.code_206": "No available
  communication device found.", //
  "########" ####
     "error.code_207": "Attended Transfer
  Failed.", // "######" ####
     "error.code_208": "Call failed. Called
  too many times.", // "#### (########)"
  ####
     "error.code_209": "Call failed.
  Invalid Number.", // "#### (####)" ####
     "error.code_210": "Operation failed
  with pending calls.", // "###########"
  ####
     "error.code_211": "Answer failed", //
  "######" ####
     "error.code_290": "No available
  microphone found.", // "#######" ####
     "error.code_291": "No available camera
  found." // "#######" ####
}
```

示例代码

使用 npm 安装并初始化 Linkus Windows SDK UI

```
import { init } from 'ys-webrtc-sdk-ui';
import 'ys-webrtc-sdk-ui/lib/ys-webrtc-sdk-ui.css';
const container =
 document.getElementById('container');
// 初始化
init(container, {
    username: '1000',
    secret: 'sdkshajgllliiaggskjhf',
    pbxURL: 'https://192.168.1.1:8088'
}).then(data => {
// 可以在这里获取暴露出的实例，处理更多业务
```

```
const { phone, pbx, destroy, on } = data;
    // ...
}).catch(err=>{
    console.log(err)
})
```

使用 script 标签方式导入并初始化 Linkus Windows SDK UI

```
<!-- 加载样式 -->
<link rel="stylesheet" href="ys-webrtc-sdk-ui.css">

<div id="test"></div>
<!-- 加载UI集成 SDK -->
<script src="./ys-webrtc-sdk-ui.js"></script>
<script>
    const test = document.getElementById('test');
    // 加载成功后通过YSWebRTCUI对象进行初始化
    window.YSWebRTCUI.init(test, {
        username: '1000',
        secret: 'sdkshajgllliiaggskjhf',
        pbxURL: 'https://192.168.1.1:8088'
    })
        .then(data => {
            const { phone, pbx } = data;
        })
        .catch(error => {
            console.log(error);
        });
</script>
```

执行结果

你已完成 Linkus Windows SDK UI 的集成及初始化，且得到了两个实例化后的 Operator 对象：

- PBXOperator：包含 PBX 相关的方法和参数，如通话记录查询、用户账号登出等。
- PhoneOperator：包含通话相关的方法及参数，如发起呼叫、接听通话及挂断通话等。

Linkus Web SDK

Linkus Web SDK 概述

Yeastar P 系列软件版 支持 Linkus SDK，可实现在第三方 Web 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。本文介绍 Linkus Web SDK 的使用要求、可用模块、接入流程以及功能。

使用要求

确保 Yeastar P 系列软件版 满足以下要求：

- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

> **Note:**
> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

可用模块

Linkus Web SDK 分离了 PBX Web 通话组件中的核心通话逻辑模块及 UI 界面模块。参见下表以了解可用模块的更多信息。

模块	说明	资源链接
Linkus	提供 Linkus 客户端的核心通话功能。	• React Demo • 项目源码
Linkus Web SDK UI	提供预设好的 UI 组件。	• 项目源码

接入流程及功能

接入流程

1. 在 Yeastar P 系列软件版 上启用 Linkus SDK
2. 获取 Linkus Web SDK 登录签名
3. 集成 Linkus Web SDK 的核心通话功能
4. (可选操作) 集成 Linkus Web SDK 的 UI 组件

Linkus Web SDK Core 功能

- Linkus Web SDK Core 使用示例
- PBX 功能(PBXOperator)
- 通话功能 (PhoneOperator)
- 通话状态及通话功能 (Session)
- 返回结果 (Result)
- 其它对象的接口及类型
- 音频资源

Linkus Web SDK 更新记录

版本 1.1.0

发布日期：2025年8月5日

- 支持视频通话。

版本 1.0.9

发布日期：2023年10月11日

- 首次发布 Linkus Web SDK。通过集成 SDK 与 Web 项目，快速实现在第三方 Web 应用内集成呼叫、通话记录、及其他 Linkus 客户端功能。

启用 Linkus SDK

将 Linkus SDK 接入至 Web 项目之前，你需要先在 PBX 上启用 Linkus SDK 功能。

> **Note:**
> 启用 Linkus SDK 后，Linkus 手机端的推送功能将不再生效。如果你想要在手机上继续接收通话相关的推送消息，参见 Linkus Android SDK 概述 或 Linkus iOS SDK 概述。

使用要求

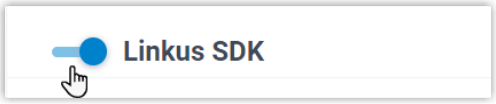
确保 Yeastar P 系列软件版 满足以下要求：

- **固件版本**：83.12.0.23 或更高
- **订阅服务**：**Linkus SDK 服务**

> **Note:**
> 订阅服务前，你可以免费试用 30 天。更多信息，请参见 启用 Yeastar P 系列软件版服务的免费试用。

操作步骤

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 打开 **Linkus SDK** 开关并在弹窗中点击 **确认**。



3. 点击 **保存**。

执行结果

已启用 Linkus SDK 功能，可 向 PBX 服务器申请 Linkus SDK 登录签名 用于鉴权。

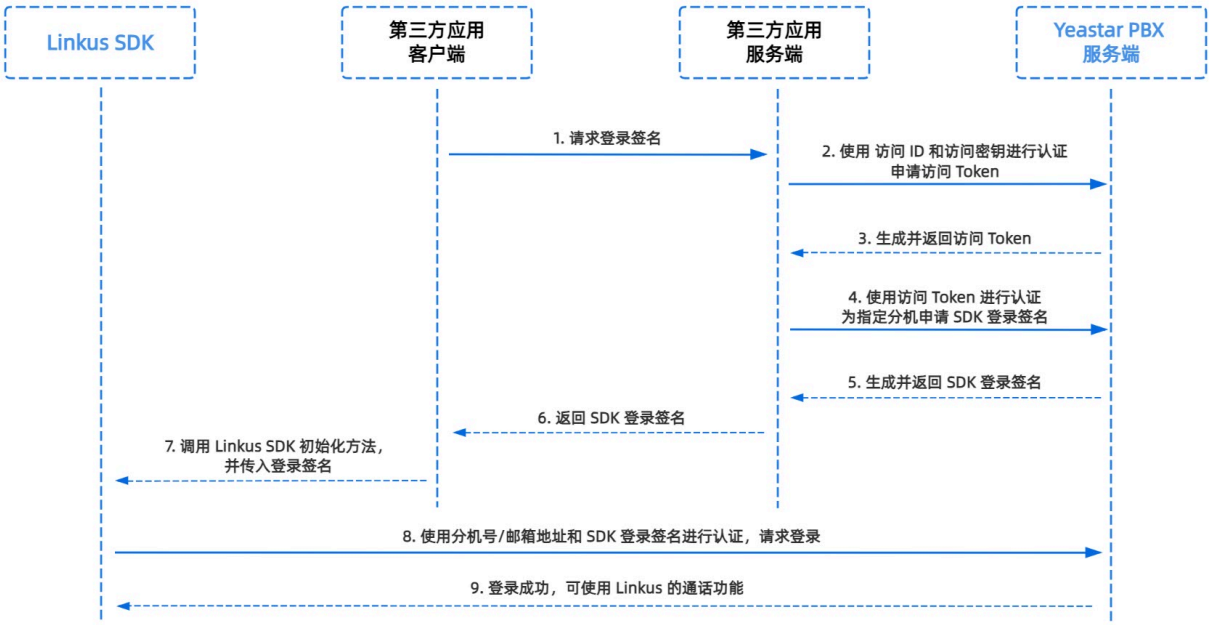
获取 Linkus Web SDK 登录签名

初始化 Linkus Web SDK 时，用户需要使用 SDK 登录签名进行鉴权。本文介绍如何通过 OpenAPI 向 PBX 服务器请求用户的 Linkus SDK 登录签名。

前提条件

已 启用 Linkus SDK。

鉴权流程



步骤一、在 PBX 获取 Linkus SDK 的访问 ID 及密钥

从 Yeastar P 系列软件版 上获取 Linkus SDK 的访问 ID 及密钥，用于应用服务端向 PBX 进行认证并建立连接。

1. 登录 PBX 管理网页，进入 **应用对接 > Linkus SDK**。
2. 记录 **访问 ID** 及 **访问密钥**。

步骤二、在应用服务端向 PBX 申请访问 Token

通过 OpenAPI，使用 Linkus SDK 的访问 ID 及密钥向 PBX 申请一个访问 Token。访问 Token 会作为 PBX API 接口调用的凭证，用于为分机用户请求 Linkus SDK 的登录签名。

请求 URL

```
POST {base_url}/openapi/v1.0/get_token
```

> **Note:**
> 如需了解 PBX 的 API 请求结构，参见 **请求结构**。

请求参数

参数	是否必填	类型	说明
username	是	String	用户名。在步骤一中获取的 **访问 ID** 作为用户名。
password	是	String	密码。在步骤一中获取的 **访问密钥** 作为密码。

请求示例

```
POST /openapi/v1.0/get_token
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "VXKkvsR2w5H28JAWtBXuJHMTeaQO7Zmf",
 "password": "Yq6yVsBceOZLhnuaGeMUG4U4qXXXXXXX"
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • `0`：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • `SUCCESS`：请求成功。 • `FAILURE`：请求失败。
access_token_expire_time	Integer	访问 token 有效时长 (**单位**：秒)。
access_token	String	访问 token，即 API 接口调用凭证。所有的请求都需要带一个访问 token。
refresh_token_expire_time	Integer	刷新 token 有效时长 (**单位**：秒)。
refresh_token	String	刷新 token。此刷新 token 可用于获取新的访问 token 和刷新 token。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "access_token_expire_time": 1800,
    "access_token": "EXZMpZAO86mbrKm6rFtgeb3rfcpC9uqS",
    "refresh_token_expire_time": 86400,
    "refresh_token": "SCduGecwbG9jIusiS8FxFUVn3kf0Q9R8"
}
```

步骤三、为分机请求登录签名

通过 OpenAPI， 使用访问 Token 向 PBX 服务器请求分机的登录签名。

请求 URL

```
POST /openapi/v1.0/sign/create?access_token={access_token}
```

请求参数

参数	是否必填	类型	说明
username	是	string	分机号或邮箱地址。
sign_type	是	string	登录签名的类型。 **允许填写的值**：`sdk`
expire_time	否	Integer	登录签名的到期时间戳 (秒)。 `0` 表示不限制登录签名的有效时长。

请求示例

```
POST /openapi/v1.0/sign/create?access_token=EXZMpZAO86mbrKm6rFtg
eb3rfcpC9uqS
Host: 192.168.5.150:8088
Content-Type: application/json
{
 "username": "1000",
 "sign_type": "sdk",
       "expire_time": 0
}
```

响应参数

参数	类型	说明
errcode	Integer	返回错误码。 • 0：请求成功。 • 非零值：请求失败。
errmsg	String	返回信息。 • SUCCESS：请求成功。 • FAILURE：请求失败。
data	Array<Ext_Sign>	分机的登录签名。

Ext_Sign

参数	类型	说明
sign	String	分机的登录签名。

响应示例

```
HTTP/1.1 200 OK
{
    "errcode": 0,
    "errmsg": "SUCCESS",
    "data":{
        "sign": "ueb3rfcpsiS8FxFUZMpZAO86mbrKmVn3kf0Q9R8"
    }
}
```

执行结果

第三方应用服务端已获取初始化 Linkus Web SDK 所需的签名，且自动将其同步至应用客户端，并下发给 Linkus Web SDK。

后续操作

使用登录签名向 PBX 服务器鉴权并 集成并初始化 Linkus Web SDK Core。

Linkus Web SDK Core

集成 Linkus Web SDK Core

Linkus Web SDK Core 提供核心的通话功能 (不带 UI 组件)，本文介绍如何将 Linkus Web SDK Core 集成到 Web 项目。

前提条件

- 已 在 Yeastar P 系列软件版 上启用 Linkus SDK
- 已 获取 Linkus Web SDK 登录签名。

Demo & 源码

在正式集成之前，我们推荐你体验 Linkus Web SDK Core 的 React Demo，并查阅 项目源码 以了解 Linkus Web SDK Core 的整体运行框架及流程。

支持的模块化格式

Linkus Web SDK Core 支持 **UMD** 、**CJS**、**ESM** 及 **IIFE** 四种模块化格式，你可以根据实际需求选择所需的格式。

> **Note:**
> 如果你想要较小的 SDK 体积，或现有项目支持 ESM，建议导入 **ESM** 使用。

步骤一、引入 Linkus Web SDK Core

1. 前往 Linkus Web SDK Core 的 GitHub 仓库，下载 Linkus Web SDK Core。
2. 使用以下任意一种方式将 Linkus Web SDK Core 引入至 Web 项目。
   - 使用 npm 安装 Linkus Web SDK Core

     ```
     npm install ys-webrtc-sdk-core
     ```

   - 使用 script 标签方式引入 Linkus Web SDK Core

     ```
     <script src="./ys-webrtc.umd.js"></script>
     <script>
       // 加载成功后通过YSWebRTC对象进行初始化
       YSWebRTC.init({
         username: '1000',
         secret: 'sdkshajgllliiaggskjhf',
         pbxURL: 'https://192.168.1.1:8088',
     ```

```
  })
    .then((operator) => {
      // 获得 PhoneOperator和PBXOperator实例和destroy方法
      const { phone, pbx, destroy } = operator;
    })
    .catch((error) => {
      console.log(error);
    });
</script>
```

步骤二、初始化 Linkus Web SDK Core

使用 **init** 方法对 Linkus Web SDK Core 进行初始化。初始化成功后会得到两个实例化后的 Operator 对象和一个方法。

名称	类型	说明
PBXOperator	对象	包含 PBX 相关的方法和参数，如通话记录查询、用户账号登出等。
PhoneOperator	对象	包含通话相关的方法及参数，如发起呼叫、接听通话、结束通话等。
destroy	方法	用于销毁 Linkus Web SDK Core。

方法

```
init({
    params//具体参数参见下表
})
```

参数

参数	类型	是否必填	说明
username	string	是	分机号或邮箱地址。
secret	string	是	用户的 Linkus SDK 登录签名。
pbxURL	URL | string	是	PBX 的访问地址，需要包含使用协议以及对应的端口。如：`https://192.168.1.1:8088` 或 `https://yeastar.example.com`。
enableLog	boolean	否	是否启用日志输出并将错误日志上报至 PBX。 **取值范围**： • `true`：启用 • `false`：禁用 **Note:**

参数	类型	是否必填	说明
			此功能默认启用。
reRegistryPhoneTimes	number	否	指定可重新注册 SIP UA 的次数，默认无限制。
userAgent	WebPC" | "WebClient	否	Asterisk 系统中的用户代理 (User Agent)，用于标识正在使用该系统的客户端。 默认值为 **WebClient**。
deviceIds	{ cameraId?: string; microphoneId?: string;}	否	指定音视频输入设备的 ID，包含摄像头 ID 及麦克风 ID。
disableCallWaiting	boolean	否	是否禁用呼叫等待。 **取值范围**： • `true`：禁用呼叫等待，PBX 设置的呼叫等待时间不生效，且 PBX 只处理单路通话。 • `false`：启用呼叫等待。

示例代码

- 使用 npm 安装并初始化 Linkus SDK Core。

```
import { init } from 'ys-webrtc-sdk-core';

init({
    username: '1000',
    secret: 'sdkshajgllliiaggskjhf',
    pbxURL: 'https://192.168.1.1:8088'
})
    .then(operator => {
        // 获得 PhoneOperator实例、PBXOperator 实例和 destroy 方法
        const { phone, pbx, destroy } = operator;

        // 创建 RTC 实例
        phone.on('newRTCSession', ({callId,session})=>{
            const {status} = session

            // 监听事件
            session.on('confirmed', {callId,session})=>{
```

```
                // 通话成功建立，'session.status.callStatus'
变成'talking'
                // 通话界面开始进行计时
            })

        })
        // 监听'startSession'事件
        phone.on('startSession',({callId,session})=>{
            const {status} = session
            if(status.communicationType === 'outbound') {
                // 外线去电
                // 刷新界面显示 'Calling'，表示被叫正在响铃
            }else{
                // 外线来电
                // 刷新界面显示 'Connecting'
            }

        });

        // 监听来电事件
        phone.on('incoming', ({callId,session})=>{
            const {status} = session
            // 来电弹窗显示来电号码和联系人名称
            // ...
            // 点击接听按钮触发 'answer' 方法和 'startSession'
事件
            phone.answer(status.number);
        });

        // 订阅事件后，开始连接 SIP UA
        phone.start();

        // ...
        // 点击 Call 按钮呼叫 1001
        phone.call('1001')

    })
    .catch(error => {
        console.log(error);
    });
```

- 使用 script 标签方式导入并初始化 Linkus SDK Core。

```
<script src="./ys-webrtc.umd.js"></script>
<script>
  // 加载成功后通过YSWebRTC对象进行初始化
```

```
YSWebRTC.init({
  username: '1000',
  secret: 'sdkshajgllliiaggskjhf',
  pbxURL: 'https://192.168.1.1:8088',
})
  .then((operator) => {
    // 获得 PhoneOperator和PBXOperator实例和destroy方法
    const { phone, pbx, destroy } = operator;
  })
  .catch((error) => {
    console.log(error);
  });
  </script>
```

Related information

Linkus Web SDK Core 使用示例

使用 Linkus Web SDK Core

Linkus Web SDK Core 使用示例

本文使用简化的代码提供一个示例，展示使用 Linkus Web SDK Core 发起呼叫及接听来电的流程。

```
import { init, on } from 'ys-webrtc-sdk-core';

init({
    username: '1000',
    secret: 'sdkshajgllliiaggskjhf',
    pbxURL: 'https://192.168.1.1:8088'
})
    .then(operator => {
        // 获得 PhoneOperator 实例、PBXOperator 实例及 destroy 方法
       const { phone, pbx, destroy } = operator;

        // 创建 RTC 实例
        phone.on('newRTCSession', ({callId,session})=>{
            const {status} = session

            // 开始监听 session 中的事件
            session.on('confirmed', {callId,session})=>{
                // 通话成功接通，session.status.callStatus 进入 talking 状态
                // 界面更新，开始通话计时
```

```
            })

        })
        // 监听 startSession 事件
        phone.on('startSession',({callId,session})=>{
            const {status} = session
            if(status.communicationType === 'outbound') {
                // 去电
                // 界面更新，对方响铃中
            }else{
                // 来电
                // 界面更新，连接中
            }

        });

        // 监听来电事件
        phone.on('incoming', (callId,session)=>{
            const {status} = session
            // 界面弹出来电窗口，显示来电号码，来电者姓名
            // ...
            // 点击接听按钮，调用 answer 方法，随后触发 startSession 事件
            phone.answer(status.number);
        });

        // 监听事件后，开始创建 SIP UA 连接
        phone.start();

        // ...
        // 点击呼出按钮，呼叫1001
        phone.call('1001')

    })
    .catch(error => {
        console.log(error);
    });
```

PBX 功能(PBXOperator)

PBXOperator 对象用于实现与 PBX 相关的功能，如通话记录查询、用户账号登出等。本文介绍与 PBX (PBXOperator 对象) 相关功能的属性、方法及事件。

属性

属性	类型	说明
username	string	用户的登录名，即分机号或邮箱地址。
secret	string	用户的登录签名，可通过 OpenAPI 向 PBX 请求获取。
token	string	向 PBX 请求的访问 Token，可通过 OpenAPI 获取。 更多信息，请参见。
url	URL	PBX 的访问地址。
socket	WebSocket	用于实时监听 PBX 运行状态变更及接收消息通知的 socket 实例。
extensionNumber	string	分机号码。
extensionId	string	分机 ID。
extensionName	string	分机姓名。

方法

方法概览

- 初始化
- 监听事件
- 查询通话记录 (CDR)
- 用户账号登出
- 销毁 PBXOperator 对象

初始化

Linkus SDK Core 内部会调用此方法进行初始化。

> **Note:**
> 初始化成功后，如果再次调用此方法，调用不生效且返回 `Promise.reject`。

方法	`init()`
方法参数	空。

返回值	`Promise<Result>`

监听事件

方法	`on(eventName, listener)`
方法参数	• `eventName`：事件名称。 • `listener`：回调函数。
返回值	空。

> **Note:**
> 关于可监听的事件，参见 事件。

查询通话记录 (CDR)

方法

```
cdrQuery(params)
```

方法参数

```
params: {
    page: number; //必填，定义显示第几个页面
    size: number; //必填，定义每页显示几项查询结果
    status?: number; // 可选，指定通话类型。0：所有；1：呼入；2：未接来电；3：呼出
    sortBy: 'time' | 'id'; //必填，定义排序字段
    orderBy?: 'desc' | 'asc'; //可选，定义显示顺序
    filter?: string | null; //可选，定义筛选条件
}
```

返回值

```
Promise
```

返回值字段示例

```
{
    errcode: 0,
    errmsg: "SUCCESS",
    personal_cdr_list: [ // 通话记录列表
        {
            id: 2546, // 通话记录的序号
            date: "Today", // 接听或拨打该通通话的日期
            time: "17:21:39", // 接听或拨打该通通话的时间
            timestamp: 1686561699, // 通话记录时间的时间戳
            number: "1011", // 主叫号码
            number_type: "extension",
            extension: { // 分机信息
                ext_id: 25, // 分机 ID
                ext_num: "1011", // 分机号码
                caller_id_name: "cwt1011", // 分机姓名
                photo: "", // 头像 ID
                status: 0, // 分机的终端在线状态。 0：离线； 1：在线
                presence_status: "available", // 分机的在线状态
                presence_information: "",
```

```
                first_name: "wt1011", // 名
                last_name: "c", // 姓
                email_addr: "", // 邮箱地址
                mobile_number: "", // 手机号码
                enb_vm: 0, // 是否启用语音信箱
                vm_total_count: 0, // 语音留言总数
                vm_unread_count: 0, // 未读的语音留言数
                visable: 0,
                im_id: "xxxxx",
                org_list_info: "",
                is_favorite: 0,
                title: "" // 职位
            },
            status: 3, //通话类型。0：所有；1：呼入；2：未接来电；3：呼出
            talk_duration: 23, // 通话被应答到通话结束的时间。
            call_type: "Internal", // 通讯类型。Internal：内线；External：外线
            uniqueid: "1686561699.137", // 通话记录唯一 ID
            disposition: "ANSWERED", // 通话状态。ANSWERED：已接；NO
 ANSWER：未接；BUSY：忙；FAILED：失败；VOICEMAIL：语音留言
            dcontext: "DLPN_DialPlan1010",
            caller_id: "1011", // 主叫号码
            clid: "\"cwt1010\" <1010>"
        }
    ],
    total_number: 63, // 查询的通话记录的总数
}
```

用户账号登出

方法	`logout()`
方法参数	空。
返回值	`Promise`

销毁 PBXOperator 对象

方法	`destroy()`
方法参数	空。
返回值	`void`

事件

监听事件示例

```
pbx.on('eventName', (data) => {}) //eventName：事件名，data：数据类型
```

运行错误通知

事件名	`runtimeError`
数据类型	`PBXResult`
报告参数	`{ code: '', msg: '', }` **Note:** 关于事件的报告参数及说明，参见下表。

报告参数说明

code	msg	说明
-106	LINKUS_DISABLED	Linkus 客户端未启用。
-107	LOGGED_IN_ELSEWHERE	该分机已在其它应用登录。
-108	EXTENSION_DELETED	该分机已被删除。
-109	RE_LOGIN	分机需要重新登录。
-110	SDK_PLAN_DISABLED	未购买 **Linkus SDK 服务**。

通话记录变更通知

该事件触发时需要手动调用查询通话记录 (CDR) 的方法。

事件名	`cdrChange`
数据类型	`cdrNotifyData`
报告参数	`{ ext_num: '1001', personal_cdr: { date:"", //##### time:"", //##### timestamp: 1693201380, // ############## number: "1001", talk_duration:0 // #### } }`

```
}
```

通话功能 (PhoneOperator)

PhoneOperator 对象用于管理通话。每通通话都是通过 `sessions` 属性缓存，该属性本身是一个包含了不同通话实例的地图，即 `Map<string, Session>`，每个通话实例都被表示为一个 `Session`。本文介绍与通话功能 (PhoneOperator 对象) 相关的属性、方法及事件。

属性

属性名	类型	说明
currentSessionID	string	标识当前通话 (Session) 的通话 ID。
reRegistryPhoneTimes	number	指定可重新注册 SIP UA (User Agent) 的次数。
deviceIds	{ cameraId?: string; microphoneId?: string;}	指定音视频输入设备的 ID，包含摄像头 ID 及麦克风 ID。
sessions	Map<string, Session>	通话 Session 的缓存 Map 对象， key 为 callId。
currentSession	Session	当前通话 Session。
isRegistered	boolean	是否成功注册 SIP UA。
extraHeaders	string[]	额外的 SIP 请求头。
videoPlan	string	获取视频服务状态。
recordPermissions	number	是否有录音权限： • `0`：无权限 • `1`：有暂停 / 恢复录音权限 • `2`：有开始 / 暂停 / 恢复录音权限
incomingList	session[]	来电列表。
isNoneCamera	boolean	是否无摄像头。
isMaxCall	boolean	是否已达最大通话数量。

方法

方法概览

监听事件	盲转接通话	开始录音	获取全部通话 (sessions 属性)

开始注册 SIP UA	咨询转接通话	暂停录音	设置当前通话 (currentSession)
重新注册 SIP UA	保持通话	结束通话	获取当前通话 (currentSession)
发起呼叫	恢复通话	重新协商通话	更新 Session 的静态属性
拒接来电	发送 DTMF	音频转视频	销毁 PhoneOperator 对象
接听来电	静音通话	视频转音频	
挂断通话	取消静音通话	断开 SIP UA 注册	

监听事件

方法	`on(eventName:string,listener: (...args: any[]) => void)`
方法参数	• `eventName`：事件名称。 • `listener`：回调函数。
返回值	`this`

> **Note:**
> 关于可监听的事件，参见 事件。

开始注册 SIP UA

SIP UA 注册成功后即可呼出和接听电话。

> **Note:**
> 调用此方法之前，确保你已监听需要监听的事件，否则可能会错失某些事件的通知。

方法	`start()`
方法参数	空。
返回值	`this`

重新注册 SIP UA

该方法会在 phone 实例内部调用。外部程序无需调用此方法，以免出现异常。

方法	`reRegister(authorizationUser: string, ha1: string)`

方法参数	• `authorizationUser`：用户名。 • `ha1`：登录密码。
返回值	`this`

发起呼叫

方法	`call(number: string, option?: CallOptions, transferId?: string)`
方法参数	• `number`：被叫号码。 • `option`：可选。指定通话选项。 • `transferId`：可选。咨询转接通话的 ID。 **Note:** 当有 transferId 时则为咨询转接通话，异步函数。
返回值	`Promise<Result>`

拒接来电

方法	`reject(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

接听来电

方法	`answer(callId: string, option?: CallOptions, allowNoneCamera: boolean = true)`
方法参数	• `callId`：通话的唯一 ID。 • `option`：可选。指定通话选项。 • `allowNoneCamera`：是否允许无摄像头接听。
返回值	`Promise<Result>`

挂断通话

方法	`hangup(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

盲转接通话

方法	`blindTransfer(callId: string, number: string)`
方法参数	• `callId`：通话的唯一 ID。 • `number`：通话转接对象的号码。
返回值类型	`boolean`

咨询转接通话

方法	`attendedTransfer(callId: string, number: string)`
方法参数	• `callId`：通话的唯一 ID。 • `number`：通话转接对象的号码。
返回值类型	`boolean`

保持通话

方法	`hold(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

恢复通话

方法	`unhold(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

发送 DTMF

方法	`dtmf(callId: string, dtmf: string)`
方法参数	• `callId`：通话的唯一 ID。 • `dtmf`：字符串 (0123456789*#)
返回值类型	`boolean`

静音通话

方法	`mute(callId: string)`

方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

取消静音通话

方法	`unmute(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

开始录音

方法	`startRecord(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

暂停录音

方法	`pauseRecord(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

结束通话

方法	`terminate(callId: string, type: 'hangup' | 'reject' | 'terminate' = 'terminate')`
方法参数	• `callId`：通话的唯一 ID。 • `type`：通话结束类型。
返回值类型	`boolean`

重新协商通话

方法	`renegotiate(callId: string, offerToReceiveVideo?: boolean)`
方法参数	• `callId`：通话唯一标识。 • `offerToReceiveVideo`：是否接收视频。

返回值类型	`boolean`

音频转视频

方法	`audioToVideo(callId: string, allowNoneCamera: boolean = true)`
方法参数	• `callId`：通话唯一标识。 • `allowNoneCamera`：是否允许无摄像头。
返回值	`Promise<Result>`

视频转音频

方法	`videoToAudio(callId: string)`
方法参数	`callId`：通话唯一标识。
返回值类型	`boolean`

断开 SIP UA 注册

方法	`disconnect()`
方法参数	空。
返回值类型	`boolean`

获取全部通话 (sessions 属性)

该方法将以数组的形式返回。

方法	`getSession()`
方法参数	空。
返回值类型	`boolean`

设置当前通话 (currentSession)

方法	`setCurrentSession(callId: string)`
方法参数	`callId`：通话的唯一 ID。
返回值类型	`boolean`

获取当前通话 (currentSession)

方法	`getCurrentSession()`
方法参数	空。
返回值	`Session`

更新 Session 的静态属性

方法	`setSessionStaticStatus(callId: string, staticStatus: Partial<StaticCallStatus>, startManualModel?: boolean)`
方法参数	• `callId`：通话的唯一 ID。 • `staticStatus`：Session 的静态属性包括 `name`、`avatar`、`company`。 • `startManualModel`：可选。是否启用手动模式，启用后将不再自动更新静态属性。
返回值类型	`boolean`

销毁 PhoneOperator 对象

此方法将解除所有订阅事件，停止 SIP UA 实例，并删除所有 session。

方法	`destroy()`
方法参数	空。
返回值	空。

事件

事件名称	事件说明	报告参数
connected	SIP 服务连接成功	空
disconnected	SIP 服务断开连接	空
registered	SIP UA 注册成功	空
registrationFailed	SIP UA 注册失败	• `code: number`：错误码 • `msg: string`：错误信息
newRTCSession	创建新的通话实例 (Session)	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
incoming	来电	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例

事件名称	事件说明	报告参数
startSession	添加通话实例 (Session) 至存储通话的 Map (sessions) 中	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
recordPermissionsChange	录音权限发生改变	• `0`：无权限 • `1`：有暂停 / 恢复录音权限 • `2`：有开始 / 暂停 / 恢复录音权限
deleteSession	删除通话实例 (Session)	• `callId: string`：通话的唯一 ID • `cause: string`：删除通话实例的原因
isRegisteredChange	SIP UA 注册状态是否发生改变	• `true`：注册状态已改变 • `false`：注册状态未改变

通话状态及通话功能 (Session)

Session 对象为通话实例，用于实现通话状态管理及通话中的相关操作。本文介绍与通话状态及通话功能 (Session 对象) 相关的属性、方法及事件。

属性

属性	类型	说明
RTCSession	RTCSession	通话实例。
callReport	Report	通话质量报告。
agreeChangeVideo	(params: AgreeChangeVideoParamsType) => Promise<boolean>	是否同意切换成视频通话回调函数。如果没有提供该回调函数，将默认拒绝切换。
status	CallStatus	通话状态，包含名字、号码、头像、是否静音等属性。
incomingList	Session[]	来电数组。
timer	TimerType	通话计时器。
localStream	MediaStream	本地媒体流，仅含视频轨，不含音频。
remoteStream	MediaStream	远端媒体流，包含音视频轨。

方法

方法概览

停止通话计时	保持通话	结束通话

监听事件	恢复通话	重新协商通话
拒接来电	发送 DTMF	音频转视频
接听来电	静音通话	视频转音频
挂断通话	取消静音通话	更新通话状态
盲转接通话	开始录音	更新 Session 的静态属性
咨询转接通话	暂停录音	销毁 Session 实例

停止通话计时

方法	`stopTimer()`
方法参数	空。
返回值	`this`

监听事件

方法	`on(eventName:string,listener: (...args: any[]) => void)`
方法参数	• `eventName`：事件名称。 • `listener`：回调函数。
返回值	`this`

Note:
关于可监听的事件，参见 事件。

拒接来电

该方法等价于 **PhoneOperator** 对象的拒接来电方法。

方法	`reject()`
方法参数	空。
返回值	`void`

接听来电

该方法等价于 **PhoneOperator** 对象的接听来电方法。

方法	`answer(option?: CallOptions, allowNoneCamera: boolean = true)`
方法参数	• `option`：指定通话参数。 • `allowNoneCamera`： 是否允许无摄像头接听。
返回值	`Promise<Result>`

挂断通话

该方法等价于 **PhoneOperator** 对象的挂断通话方法。

方法	`hangup()`
方法参数	`callId`：通话唯一标识。
返回值	`void`

盲转接通话

该方法等价于 **PhoneOperator** 对象的盲转接通话方法。

方法	`blindTransfer(number: string)`
方法参数	`number`：通话转接对象的号码。
返回值	`void`

咨询转接通话

该方法等价于 **PhoneOperator** 对象的咨询转接通话方法。

方法	`attendedTransfer(number: string)`
方法参数	`number`：通话转接对象的号码。
返回值	`void`

保持通话

该方法等价于 **PhoneOperator** 对象的保持通话方法。

方法	`hold()`
方法参数	空。
返回值	`void`

恢复通话

该方法等价于 **PhoneOperator** 对象的恢复通话方法。

方法	`unhold()`
方法参数	空。
返回值	`void`

发送 DTMF

该方法等价于 **PhoneOperator** 对象的发送 DTMF 方法。

方法	`dtmf(dtmf: string)`
方法参数	`dtmf`：字符串 (0123456789*#)
返回值	`void`

静音通话

该方法等价于 **PhoneOperator** 对象的静音通话方法。

方法	`mute()`
方法参数	空。
返回值	`void`

取消静音通话

该方法等价于 **PhoneOperator** 对象的取消静音通话方法。

方法	`unmute()`
方法参数	空。
返回值	`void`

开始录音

该方法等价于 **PhoneOperator** 对象的开始录音方法。

方法	`startRecord()`
方法参数	空。

返回值	`void`

暂停录音

该方法等价于 **PhoneOperator** 对象的暂停录音方法。

方法	`pauseRecord()`
方法参数	空。
返回值	`void`

结束通话

该方法等价于 **PhoneOperator** 对象的结束通话方法。

方法	`terminate(type: 'hangup' | 'reject' | 'terminate' = 'terminate')`
方法参数	`type`：通话结束类型。
返回值	`void`

重新协商通话

方法	`renegotiate(offerToReceiveVideo?: boolean)`
方法参数	`offerToReceiveVideo`：是否接收视频。
返回值类型	`boolean`

音频转视频

方法	`audioToVideo(allowNoneCamera: boolean = true)`
方法参数	`allowNoneCamera`：是否允许无摄像头。
返回值	`Promise<Result>`

视频转音频

方法	`videoToAudio()`
方法参数	空。
返回值类型	`boolean`

更新通话状态

方法	`setStatus(status: Partial<CallStatus>)`
方法参数	`status`：通话状态对象。
返回值	`this`

更新 Session 的静态属性

方法	`setStaticStatus(staticStatus: Partial<StaticCallStatus>, startManualModel?: boolean)`
方法参数	• `staticStatus`：Session 的静态属性包括 `name`、`avatar`、`company`。 • `startManualModel`：可选。是否启用手动模式，启用后将不再自动更新静态属性。
返回值	`this`

销毁 Session 实例

此方法将解除所有订阅事件，停止 RTCSession。

方法	`destroy()`
方法参数	空。
返回值	空。

事件

事件名称	事件说明	报告参数
callReport	通话质量报告更新事件，每3秒更新一次	• `callId: string`：通话的唯一 ID • `callReport: Report`：通话质量报告
streamAdded	添加媒体流	• `callId: string`：通话的唯一 ID • `communicationType: "outbound" | "inbound"`：通话类型 • `stream: MediaStream`：媒体流
ended	通话结束	• `callId: string`：通话的唯一 ID • `cause: string`：通话结束的原因
failed	通话失败	• `callId: string`：通话的唯一 ID • `cause: string`：通话失败的原因 • `code: number`：错误码

事件名称	事件说明	报告参数
clientError	客户端错误，导致无法发起通话	• `callId: string`：通话的唯一 ID • `cause: string`：客户端错误的原因
reinvite	重新邀请事件，该事件在对端执行咨询转接时触发	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
accepted	收到成功状态响应代码 **200 OK**	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
confirmed	通话正式建立 (收到 **ack** 包)	• `callId: string`：通话的唯一 ID • `session: Session`：当前通话实例
statusChange	通话状态变更	• `newStatus`：新通话状态 • `oldStatus`：原通话状态
staticStatusChange	通话的静态属性变更	• `newStatus`：新通话静态属性 • `oldStatus`：原通话状态静态属性

返回结果 (Result)

本文介绍 Linkus Web SDK Core 返回结果 (Result 对象) 的结构及各个子类的状态码说明。

背景信息

Result 结构

```
{
    code: '',
    msg: '',
}
```

Result 子类

参见下表以了解 Result 包含的子类及其代码范围。

子类	说明	代码范围
CommonResult	通用的返回结果	• 正确代码：0 到 99 • 错误代码：-1 到 -99
PBXResult	与 PBXOperator 对象相关的返回结果	• 正确代码：100 到 199 • 错误代码：-100 到 -199
PhoneResult	与 PhoneOperator 对象相关的返回结果	• 正确代码：200 到 299

子类	说明	代码范围
		• 错误代码：-200 到 -299

CommonResult

错误代码 (COMMON_ERROR)

code	msg	说明
-1	UNKNOWN_ERROR	未知错误。
-2	INVALID_PBX_URL	无效的 PBX 访问地址。
-3	PBX_URL_NOT_HTTPS	PBX 访问地址的传输协议不是 HTTPS。
-4	GET_PRODUCT_FAILED	获取 PBX 的 PRODUCT 接口失败。

成功代码 (COMMON_SUCCESS)

code	msg	说明
0	SUCCESS	成功。

PBXResult

错误代码 (PBX_ERROR)

code	msg	说明
-100	UNKNOWN_ERROR	未知错误。
-101	REGISTRY_FAILED	SIP UA 注册失败。
-102	PBX_NETWORK_ERROR	PBX 接口请求错误 (客户端网络异常)。
-103	PBX_API_ERROR	PBX 接口请求错误 (服务端异常)。
-104	GET_PERSONAL_NOT_FOUND_DATA	PBX 未返回分机信息。
-105	PBX_ALREADY_INITIALIZED	PBX 对象已初始化过，不允许重复初始化。
-106	LINKUS_DISABLED	Linkus 客户端未启用。
-107	LOGGED_IN_ELSEWHERE	该分机已在其它应用登录。
-108	EXTENSION_DELETED	该分机已被删除。

code	msg	说明
-109	RE_LOGIN	分机需要重新登录。
-110	SDK_PLAN_DISABLED	未购买 **Linkus SDK 服务**。

成功代码 (PBX_SUCCESS)

code	msg	说明
100	SUCCESS	成功。

PhoneResult

错误代码 (PHONE_ERROR)

code	msg	说明
-200	UNKNOWN_ERROR	未知错误。
-201	REGISTRY_FAILED	SIP UA 注册失败。
-202	GET_AGREE_CHROME_USER_MEDIA_ROLE_ERROR	获取媒体流失败 (未获得浏览器授权)。
-203	GET_LOCAL_STREAM_ERROR	获取本地媒体流失败。
-204	RE_REGISTRY_MAX_LIMIT_TIMES	已达最大 SIP UA 重新注册次数。
-205	MAX_LIMIT_CALL	已达最大通话数。
-206	GET_LOCAL_MEDIA_INFO_ERROR	获取本地媒体设备错误。
-207	ATTENDED_PARENT_NOT_FOUND	咨询转接通话的父节点未找到。
-208	CALL_TOO_MANY_TIMES	到达一秒钟内的呼出次数上限。
-209	INVALID_NUMBER	无效号码。
-210	CURRENT_CALL_HAS_NOT_CONNECTED	当前有未接通的通话。
-211	NOT_FOUND_CALL_ID	未找到通话 ID。
-290	NOT_FOUND_AUDIO_INPUT_DEVICE	未找到音频输入设备。
-291	NOT_FOUND_VIDEO_INPUT_DEVICE	未找到视频输入设备。

成功代码

code	msg	说明
200	SUCCESS	成功。

其它对象的接口及类型

本文介绍 Linkus Web SDK Core 所使用的其它对象的接口及类型。

CallOptions 类型

通话选项。

```
type CallOptions = {
    video?: boolean; // 是否为视频通话
    offerToReceiveVideo?: boolean; // 是否接收对端视频
    extraHeaders?: string[]; // 额外的sip请求头
};
```

Report 对象

通话质量报告对象。

```
interface Report {
    lksTimestamp: string; // 接听来电或发起呼叫时的时间戳字符串
    lksDate: string; // 接听来电或发起呼叫时的时间。时间格式 YYYY-MM-DD hh:mm:ss
    callId: string; // 通话的唯一 ID
    extension: string; // 分机号码
    iceResult: string; // ice 类型：local（对应host）、
 public（对应srflx或prflx）、 turn（对应relay）
    lksNetworkType: string; // 网络类型
    lksDeviceType: string; // 客户端类型：目前只有web
    lksDuration: string; // 对应 totalSamplesDuration 属性
    lksAudioCodec: string; // 音频编码
    lksAudioRx: string; // 收到的音频包，对应 packetsReceived 属性
    lksAudioRxLost: string; // 音频丢包率，对应
 packetsLost属性。计算公式：lksAudioRxLost=(packetsLost / (packetsLost +
 packetsReceived)).toFixed(6)
    lksAudioRxMaxjitter: string; // 音频接收端的最大抖动
    lksAudioRxAvgjitter: string; // 音频平均抖动，对应 jitter*1000
    lksAudioTx: string; // 发送的音频包
    lksAudioTxLost: string; // 发送端的音频丢包率
    lksAudioTxMaxjitter: string; // 音频发送端的最大抖动
    lksAudioTxAvgjitter: string; // 音频发送端的平均抖动
    lksVideoCodec: string; // 视频编码
```

```
    lksVideoRx: string; // 收到的视频包
    lksVideoRxLost: string; // 视频丢包率， 对应 packetsLost
 属性，计算公式lksVideoRxLost=(packetsLost / (packetsLost +
 packetsReceived)).toFixed(6)
    lksVideoRxMaxjitter: string; // 视频接收端最大抖动
    lksVideoRxAvgjitter: string; // 视频接收端平均抖动
    lksVideoTx: string; // 发送的视频包
    lksVideoTxLost: string; // 发送端的视频丢包率
    lksVideoTxMaxjitter: string; // 视频发送端的最大抖动
    lksVideoTxAvgjitter: string; // 视频发送端的平均抖动
}
```

StaticCallStatus 对象

通话的静态属性对象。

```
interface StaticCallStatus {
    name: string; // 来电者姓名
    avatar: string; // 头像
    company: string; // 公司名
}
```

CallStatus 对象

通话状态对象。

```
interface CallStatus extends StaticCallStatus{
    number: string; // 电话号码或分机号
    callId: string; // 通话的唯一 ID
    communicationType: 'outbound' | 'inbound'; //
 通话类型：outbound：去电，inbound：来电
    isVideo: boolean; //是否为视频通话
    isRing: boolean; //是否正在响铃中
    isHold: boolean; //该通话是否处于保持状态
    isMute: boolean; //该通话是否处于静音状态
    callStatus: 'ringing' | 'calling' | 'talking' | 'connecting';// 通话状态
    callStartTime: number; // 通话开始时间
    recordStatus: 'stop' | 'recording' |
 'pause'; //通话录音状态。stop：未在录音；recording：正在录音；pause：暂停录音
    isTransfer: boolean; // 是否为咨询转接通话
    transferParent?:
 TransferParentType; //当是咨询转接通话时，指定咨询转接的父通话
    isConference: boolean; // 是否为语音会议室通话
}

type TransferParentType = {
```

```
    // 咨询转接的父通话
    callId: string; //通话的唯一 ID
    avatar: string; // 头像
    name: string; // 通话转接对象的名称
    number: string; // 通话转接对象的号码
    callDuration: number; // 通话被接起到挂断的时长
    holdDuration: number; // 通话被保持的时长
}
```

TimerType 类型

通话计时类型。

```
type TimerType = {
    ringDuration: number; // 响铃时长
    callingDuration: number; // 呼叫时长：拨号开始到通话被应答的时间。
    callDuration: number; // 通话被接起到挂断的时长
    holdDuration: number; // 通话被保持的时长
}
```

AgreeChangeVideoParamsType 类型

```
type AgreeChangeVideoParamsType = { callId: string; name: string };
```

agreeChangeVideo Usage

如果需要在通话中同意切换成视频通话，可以使用 `agreeChangeVideo` 方法。

```
session.agreeChangeVideo = (inviteInfo: AgreeChangeVideoParamsType) => {
    return new Promise<boolean>((resolve) => {
        // 这里可以添加逻辑来决定是否同意切换成视频通话
        const userAgrees = Modal.confirm({
            title: '切换成视频通话',
            content: `${inviteInfo.name}邀请您切换成视频通话，是否同意？  ,
            onOk: () => {
                // 用户同意切换成视频通话
                resolve(true);
            },
            onCancel: () => {
                // 用户拒绝切换成视频通话
                resolve(false);
            }
        });
        setTimeout(() => {
            // 关闭 modal
            userAgrees.close();
```

```
            // 如果用户在10秒内没有做出选择，则默认拒绝
            resolve(false);
        }, 10 * 1000);
    });
}
```

音频资源

本文介绍 Linkus Web SDK Core 所提供的音频资源及使用方法。

背景信息

Linkus Web SDK Core 项目文件的 **assets** 目录包含了可用音频的源文件及 base 64 编码后的字符串。

- **音频源文件**：存储于 **sounds** 文件夹内。
- **base64 编码字符串**：存储于 **sound.js** 文件内。

参见下表以了解所提供的音频资源。

名称	说明	名称	说明
Ring	来电铃声	DTMF04	按键 4 提示音
Callend	通话结束提示音	DTMF05	按键 5 提示音
Callwaiting	呼叫等待提示音	DTMF06	按键 6 提示音
ringback	回铃音	DTMF07	按键 7 提示音
DTMF00	按键 0 提示音	DTMF08	按键 8 提示音
DTMF01	按键 1 提示音	DTMF09	按键 9 提示音
DTMF02	按键 2 提示音	DTMFStar	按键 * 提示音
DTMF03	按键 3 提示音	DTMFPound	按键 # 提示音

使用音频资源

Linkus Web SDK Core 支持通过以下方式引入并使用音频资源。

- 直接引用音频源文件。
- 通过 base64 编码引用音频资源。更多信息，参见下方示例代码。

```
import sounds from '/assets/sounds';
const { Ring, Callend } = sounds;
const audio = new Audio(Ring);
audio.play();
```

```
// 改变 audio.src 的值，播放不同的音频
audio.src = Callend;
audio.play();
```

Linkus Web SDK UI

集成 Linkus Web SDK UI

Linkus Web SDK UI 提供预设好的 UI 组件，你可以将其与 Web 项目集成，在 Linkus Web SDK Core 集成的基础上实现带有 UI 界面的通话功能。

前提条件

- 已 获取 Linkus Web SDK 登录签名。
- 已 集成 Linkus Web SDK Core。

步骤一、导入 Linkus Web SDK UI

1. 前往 Linkus SDK UI 的 GitHub 仓库，下载 Linkus SDK UI。
2. 使用以下任意一种方式将 Linkus Web SDK UI 引入至你的 Web 项目。
    - 使用 npm 安装 Linkus Web SDK UI

      ```
      npm install ys-webrtc-sdk-ui --save
      ```

    - 使用 script 标签方式引入 Linkus Web SDK UI

      ```
      <script src="./ys-webrtc-sdk-ui.js"></script>
      ```

步骤二、初始化 Linkus Web SDK UI

使用 `init` 方法初始化 Linkus Web SDK UI，渲染 UI 组件。

方法

```
init (container,{rtcOption})
```

方法参数

- `container`：用于渲染 UI 组件，类型为 `HTMLElement`。
- `rtcOption`：初始化选项，具体参数参见下表。

属性	类型	是否必填	说明
username	string	是	分机号或邮箱地址。
secret	string	是	用户的 Linkus SDK 登录签名。
pbxURL	URL | string	是	PBX 的访问地址，需要包含使用的协议以及对应的端口。 如：`https://192.168.1.1:8088`。
enableLog	boolean	否	是否启用日志输出并将错误日志上报至 PBX。 **取值范围**： ◦ `true`：启用 ◦ `false`：禁用 **Note:** 此功能默认启用。
reRegistryPhoneTimes	number	否	指定可重新注册 SIP UA (User Agent) 的次数。
deviceIds	{ cameraId?: string; microphoneId?: string; speakerId:string; volume:number }	否	指定音视频输入设备的 ID (包含摄像头 ID、麦克风 ID 和扬声器 ID)，并设置音量 (包含通话、来电铃声和拨号盘)。音量默认值为 0.6，取值范围为 0 到 1。
incomingOption	{ style?: React.CSSProperties; class?: string; }	否	调整 Incoming 组件样式。
dialPanelOption	{ style?: React.CSSProperties; class?: string; hideVideoBtn?: boolean }	否	调整拨号盘组件样式，并设置是否隐藏视频按钮。 **Note:** `hideVideoBtn` 参数默认值为 `false`。如需隐藏视频按

属性	类型	是否必填	说明
			钮，可将其设置为 `true`。
sessionOption	SessionOption	否	调整通话窗口组件位置和尺寸。
hiddenIncomingComponent	boolean	否	隐藏来电弹屏组件。
hiddenDialPanelComponent	boolean	否	隐藏拨号盘组件。
disableCallWaiting	boolean	否	是否禁用呼叫等待。 **取值范围**： ◦ `true`：禁用呼叫等待，PBX 设置的呼叫等待时间不生效，且 PBX 只处理单路通话。 ◦ `false`：启用呼叫等待。
intl	{ local: string; messages: Record<string, string>}	否	国际化 (多语言) 设置。

SessionOption 类型说明

```
type SessionOption = {
    sessionSetting?: {
        width?: number;
        height?: number;
        miniWidth?: number;
        miniHeight?: number;
        x?: number;
        y?: number;
    };
    style?: React.CSSProperties;
    class?: string;
}
```

intl (国际化设置) 类型说明

◦ `local`：当前使用语言的名称，用短横线连接，如 `en-US`、`zh-CN`。

- `message`：可配置的文案内容，以键值对形式呈现。参见以下内容以了解具体的配置项及默认值。

```
{
    "common.cancel": "Cancel", // ########
    "common.confirm": "Confirm", //
 ########

  "dial_panel.input.placeholder": "Please
 input number", // "####" ####

  "dial_panel.tip.connect_failed": "Failed
 to connect to server, you cannot initiate
 or answer a call. Trying to reconnect to
 the server.", //############
    "incoming.btn.hang_up": "Hang up", //
 "####" ####
    "incoming.btn.video": "Video", // ####
 "####" ####
    "incoming.btn.audio": "Audio", // ##
 "####" ####
    "session.calling": "Calling...", //
 "###" ##
    "session.ringing": "Ringing...", //
 "#####" ##
    "session.talking": "Talking...", //
 "###" ##

  "session.connecting": "Connecting...", /
/ "###" ##
    "session.hang_up": "End Call", //
 "####" ##
    "session.new_call": "New call", //
 "###" ##
    "session.record": "Record", // "##" ##
    "session.mute": "Mute", // "####" ##
    "session.video": "Video", // "####" ##
    "session.hold": "Hold", // "####" ##
    "session.resume": "Resume", // "####"
 ##
    "session.dialpad": "Dialpad", //
 "####" ##
    "session.transfer": "Transfer", //
 "####" ##
```

```
    "session.attended_transfer": "Attended
 Transfer", // "####" ##
    "session.blind_transfer": "Blind
 Transfer", // "###" ##

  "session.modal.change_to_video.title": "
Request", // "##" ###########

  "session.modal.change_to_video.content":
 "{0} invites you switch to video call. Do
 you accept?", // "#########" #######"{0}"
 #############
    "session.error.client_error": "Client
 Error: {0}", // "#####" ###"{0}" ####
    "session.tip.recording": "Recording
 the Audio...", // "#####" ##
    "session.tip.pause": "The recording is
 paused.", // "#######" ##

  "session.tip.can_no_use_video": "Unlock
 this feature with Ultimate Plan.", //
 "##########" ##
    "error.code_200": "Unknown Error.", //
 "####" ####
    "error.code_202": "No available
 communication device found.(no
 permissions)", // "########" ####
    "error.code_205": "Call failed. Cannot
 process more new calls now.", // "####
 (#######)" ####
    "error.code_206": "No available
 communication device found.", //
 "########" ####
    "error.code_207": "Attended Transfer
 Failed.", // "######" ####
    "error.code_208": "Call failed. Called
 too many times.", // "#### (########)"
 ####
    "error.code_209": "Call failed.
 Invalid Number.", // "#### (####)" ####
    "error.code_210": "Operation failed
 with pending calls.", // "###########"
 ####
    "error.code_211": "Answer failed", //
 "######" ####
```

```
    "error.code_290": "No available
 microphone found.", // "#######" ####
    "error.code_291": "No available camera
 found." // "#######" ####
}
```

示例代码

使用 npm 安装并初始化 Linkus Web SDK UI

```
import { init } from 'ys-webrtc-sdk-ui';
import 'ys-webrtc-sdk-ui/lib/ys-webrtc-sdk-ui.css';
const container =
 document.getElementById('container');
// 初始化
init(container, {
    username: '1000',
    secret: 'sdkshajgllliiaggskjhf',
    pbxURL: 'https://192.168.1.1:8088'
}).then(data => {
// 可以在这里获取暴露出的实例，处理更多业务
const { phone, pbx, destroy, on } = data;
    // ...
}).catch(err=>{
    console.log(err)
})
```

使用 script 标签方式导入并初始化 Linkus Web SDK UI

```
<!-- 加载样式 -->
<link rel="stylesheet" href="ys-webrtc-sdk-ui.css">

<div id="test"></div>
<!-- 加载UI集成 SDK -->
<script src="./ys-webrtc-sdk-ui.js"></script>
<script>
    const test = document.getElementById('test');
    // 加载成功后通过YSWebRTCUI对象进行初始化
    window.YSWebRTCUI.init(test, {
        username: '1000',
        secret: 'sdkshajgllliiaggskjhf',
        pbxURL: 'https://192.168.1.1:8088'
    })
        .then(data => {
            const { phone, pbx } = data;
        })
```

```
        .catch(error => {
            console.log(error);
        });
</script>
```

执行结果

你已完成 Linkus Web SDK UI 的集成及初始化，且得到了两个实例化后的 Operator 对象：

- PBXOperator：包含 PBX 相关的方法和参数，如通话记录查询、用户账号登出等。
- PhoneOperator：包含通话相关的方法及参数，如发起呼叫、接听通话及挂断通话等。