直播 SDK

语聊房

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

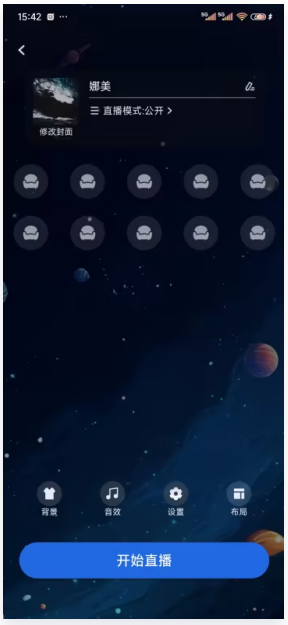
文档目录

语聊房
准备工作
准备工作（Android）

最近更新时间：2025-11-21 14:15:33

功能预览

TUILiveKit 是一个功能全面的语聊房组件，集成后可快速实现以下功能模块：

主播准备页	主播开播页	直播列表	观众观看页
			

准备工作

开通服务

在使用 TUILiveKit 前，请先参考 开通服务，领取 `TUILiveKit` 体验版或者开通付费版。

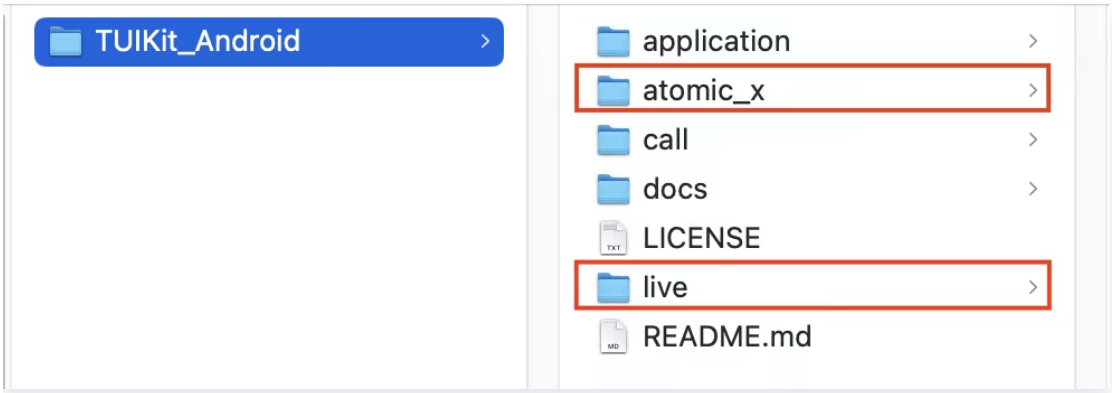
环境要求

- `Android Studio Arctic Fox (2020.3.1)` 及以上版本。
- `Gradle 7.0` 及以上的版本。
- `Android 5.0` 及以上的手机设备。

代码集成

步骤 1：下载 TUILiveKit 组件

在 GitHub 中克隆/下载代码，然后将 `live` 子目录和 `atomic_x` 子目录拷贝到您当前 `Android` 项目的 `app` 文件夹同级目录中。



步骤 2：工程配置

1. 配置 JitPack 仓库

在工程根目录下的 `settings.gradle.kts` 或 `settings.gradle` 文件中，添加 `JitPack` 仓库地址，项目中播放礼物 `SVG` 动画的第三方库托管在 `JitPack` 仓库中，需要从 `JitPack` 仓库中查找依赖项。

settings.gradle.kts

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()

        // 添加 JitPack 仓库地址
        maven { url = uri("https://jitpack.io") }
    }
}
```

settings.gradle

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
```

```
        // 添加 JitPack 仓库地址
        maven { url 'https://jitpack.io' }
    }
}
```

2. 导入 TUILiveKit 组件

在工程根目录下的 `settings.gradle.kts` 或 `settings.gradle` 文件中，增加以下代码以导入 `TUILiveKit` 组件。

settings.gradle.kts

```
include ':tuilivekit'
project(':tuilivekit').projectDir = new File(settingsDir,
"live/tuilivekit")

include(":atomic")
project(':atomic').projectDir = new File(settingsDir, "atomic_x")
```

settings.gradle

```
include(":tuilivekit")
project(":tuilivekit").projectDir = File(settingsDir, "live/tuilivekit")

include(":atomic")
project(":atomic").projectDir = File(settingsDir, "atomic_x")
```

3. 添加组件依赖

在 `app` 目录下找到 `build.gradle.kts`（或 `build.gradle`）文件，并在文件中增加以下代码，该步骤的作用是声明当前 `App` 对新加入的 `TUILiveKit` 组件的依赖。

build.gradle.kts

```
dependencies {
    // 添加 tuilivekit 依赖
    api(project(":tuilivekit"))
```

```
}
```

build.gradle

```
dependencies {
    // 添加 tuilivekit 依赖
    api project(':tuilivekit')
}
```

> **说明：**
> TUILiveKit 内部已默认依赖 TRTC SDK 、IM SDK 等公共库，您无需单独配置。

4. 配置混淆规则

由于 SDK 内部使用了 Java 反射，请在 proguard-rules.pro 文件中添加以下代码，将相关类加入不混淆名单，以确保功能正常运行。

```
-keep class com.tencent.** { *; }
-keep class com.tencent.beacon.** { *; }
-keep class com.tencent.cloud.iai.lib.** { *; }
-keep class com.tencent.qimei.** { *; }
-keep class com.trtc.uikit.livekit.component.gift.store.model.** { *;
}
-keep class com.trtc.uikit.livekit.livestreamcore.** { *; }
-keep class com.tcmediax.** { *; }

# Google 序列化/反序列化框架 Gson 库相关混淆
-keep class com.google.gson.** { *; }

# 播放礼物 SVG 动画相关混淆
-keep class com.opensource.svgaplayer.proto.** { *; }
-keep class com.squareup.wire.** { *; }
```

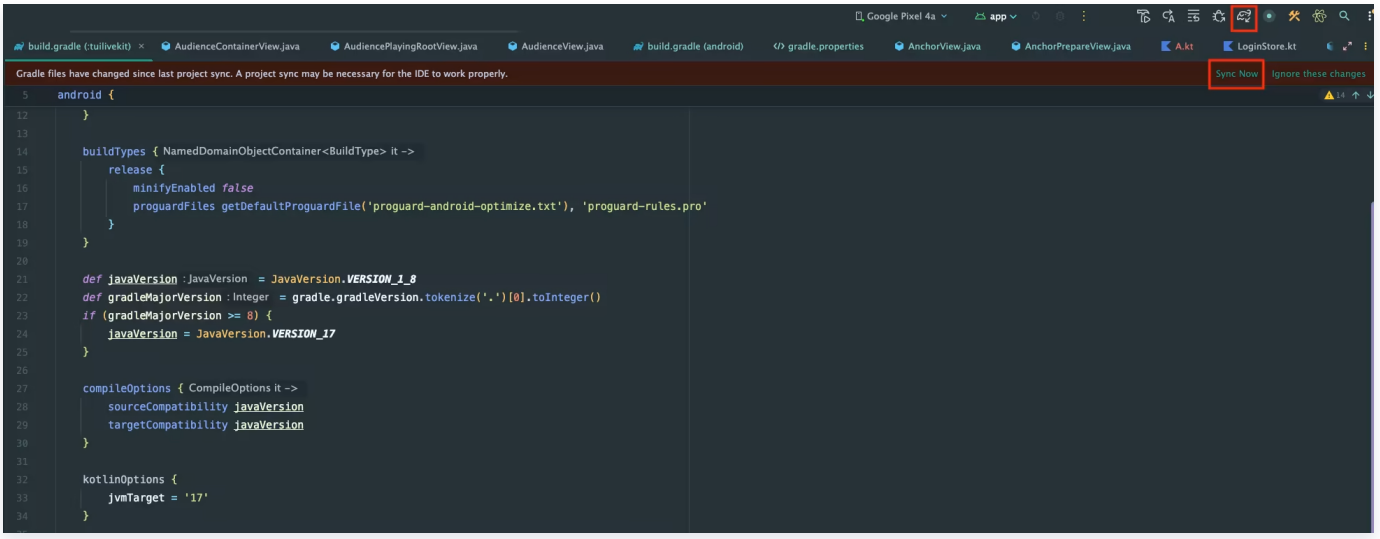
5. 修改 AndroidManifest.xml

为了防止编译时 AndroidManifest 合并过程中产生属性冲突，您需要在 app/src/main/AndroidManifest.xml 文件的 <application> 节点中，添加 tools:replace="android:allowBackup" 和 android:allowBackup="false"，用以覆盖组件内的设置。

```
<application
    ...

    // 添加如下配置覆盖 依赖的 sdk 中的配置
    android:allowBackup="false"
    tools:replace="android:allowBackup">
```

6. 完成项目同步

在您完成上述步骤后，通常情况下 `Android Studio` 会自动为您弹出 `Sync Now` 按钮，您需要点击该同步按钮完成项目的同步，若 `IDE` 没有自动弹出同步按钮，请您手动点击工具栏中的同步按钮进行项目的同步，同步后 `IDE` 会完成项目的构建配置和索引等工作，您就可以在您的项目中正常使用 `TUILiveKit` 组件了。



完成登录

代码集成完成后，您需要完成登录。这是**使用 `TUILiveKit` 的关键步骤**，因为只有在登录成功后才能正常使用 `TUILiveKit` 的各项功能，故请您耐心检查相关参数是否配置正确：

> **说明：**
> 在示例代码中，直接进行了登录接口的调用。但在实际项目场景下，强烈推荐 **您在完成自己的用户身份验证等相关登录操作后，再调用 `TUILiveKit` 的登录服务**。这样可以避免因过早调用登录服务，导致业务逻辑混乱或数据不一致的问题，同时也能更好地适配您项目中现有的用户管理和权限控制体系。

Kotlin

```
// 1.导入依赖
import com.tencent.qcloud.tuicore.TUILogin
```

```
// 2.调用登录接口，推荐您在自己登录业务完成后再调用 TUILogin 登录
TUILogin.login(applicationContext,
    1400000001,       // 请替换为开通服务控制台的 SDKAppID
    "denny",          // 请替换为您的 UserID
    "xxxxxxxxxxx",    // 您可以在控制台中计算一个 UserSig 并填在这个位置
    object : TUICallback() {
        override fun onSuccess() {
            Log.i(TAG, "login success")
        }

        override fun onError(errorCode: Int, errorMessage: String) {
            Log.e(TAG, "login failed, errorCode: $errorCode
msg:$errorMessage")
        }
    })
```

Java

```
// 1.导入依赖
import com.tencent.qcloud.tuicore.TUILogin

// 2.调用登录接口，推荐您在自己登录业务完成后再调用 TUILogin 登录
TUILogin.login(context,
    1400000001,      // 请替换为开通服务控制台的 SDKAppID
    "denny",         // 请替换为您的 UserID
    "xxxxxxxxxxx",   // 您可以在控制台中计算一个 UserSig 并填在这个位置
    new TUICallback() {
    @Override
    public void onSuccess() {
        Log.i(TAG, "login success");
    }

    @Override
    public void onError(int errorCode, String errorMessage) {
        Log.e(TAG, "login failed, errorCode: " + errorCode + " msg:" +
errorMessage);
    }
```

```
});
```

登录接口参数说明

参数	类型	说明
SDKAppID	Int	从 TRTC 控制台 > 应用管理 获取。
UserID	String	当前用户的唯一 ID，仅包含英文字母、数字、连字符和下划线。
userSig	String	用于腾讯云鉴权的票据。请注意： • 开发环境：您可以采用本地 `GenerateTestUserSig.genTestSig` 函数生成 UserSig 或者通过 UserSig 辅助工具 生成临时的 UserSig。 • 生产环境：为了防止密钥泄露，请务必采用服务端生成 UserSig 的方式。详细信息请参见 服务端生成 UserSig 。 更多信息请参见 如何计算及使用 UserSig 。

登录异常状态处理【可选】

`TUILogin` 提供了登录状态回调机制，方便您处理可能出现的登录异常情况，主要包括 “**被踢下线**” 和 “**签名过期**” 这两种异常状态的回调：

- **被踢下线**：用户在线情况下被踢，`IM SDK` 会通过 `onKickedOffline` 回调通知给您，此时可以在 `UI` 提示用户，并调用 `TUILogin.login` 重新登录。
- **签名过期**：用户在线期间收到 `onUserSigExpired` 回调，说明您之前给该用户签发的 `userSig` 已经过期了，这个时候如果当前用户在您后台的登录态依然有效，您可以让您的 `app` 向您的后台请求新的 `userSig`，并调用 `TUILogin.login` 续签登录态。

kotlin

```
class YourLoginService : TUILoginListener() {

    // 监听登录状态回调
    fun addObserver() {
        TUILogin.addLoginListener(this)
    }

    // 取消监听登录状态回调
    fun removeObserver() {
        TUILogin.removeLoginListener(this)
    }
```

```
    // 用户被踢下线回调
    override fun onKickedOffline() {
        // 您的业务代码：UI 交互提示用户，然后重新登录
    }

    // 用户签名过期回调
    override fun onUserSigExpired() {
        // 您的业务代码：如果当前用户在您后台的登录态依然有效，您可以让您的 app 向您的后台请求新的 userSig，并调用 TUILogin.login 续签登录态。
    }
}
```

java

```
class YourLoginService extends TUILoginListener {

    // 监听登录状态回调
    public void addObserver() {
        TUILogin.addLoginListener(this);
    }

    // 取消监听登录状态回调
    public void removeObserver() {
        TUILogin.removeLoginListener(this);
    }

    // 用户被踢下线回调
    @Override
    public void onKickedOffline() {
        // 您的业务代码：UI 交互提示用户，然后重新登录
    }

    // 用户签名过期回调
    @Override
    public void onUserSigExpired() {
        // 您的业务代码：如果当前用户在您后台的登录态依然有效，您可以让您的 app 向您的后台请求新的 userSig，并调用 TUILogin.login 续签登录态。
    }
```

```
}
```

下一步

恭喜您，现在您已经成功集成了**语聊房**组件并完成了登录。接下来，您可以实现**主播开播、观众观看、直播列表**等功能，可参考下表：

功能	描述	集成指引
主播开播	主播创建语聊房全流程功能，包括开播前的准备和开播后的各种互动。	主播开播
观众观看	观众进入语聊房后收听，实现上麦、弹幕显示等功能。	观众观看
直播列表	展示语聊房列表界面和功能，包含语聊房列表，房间信息展示功能。	直播列表

常见问题

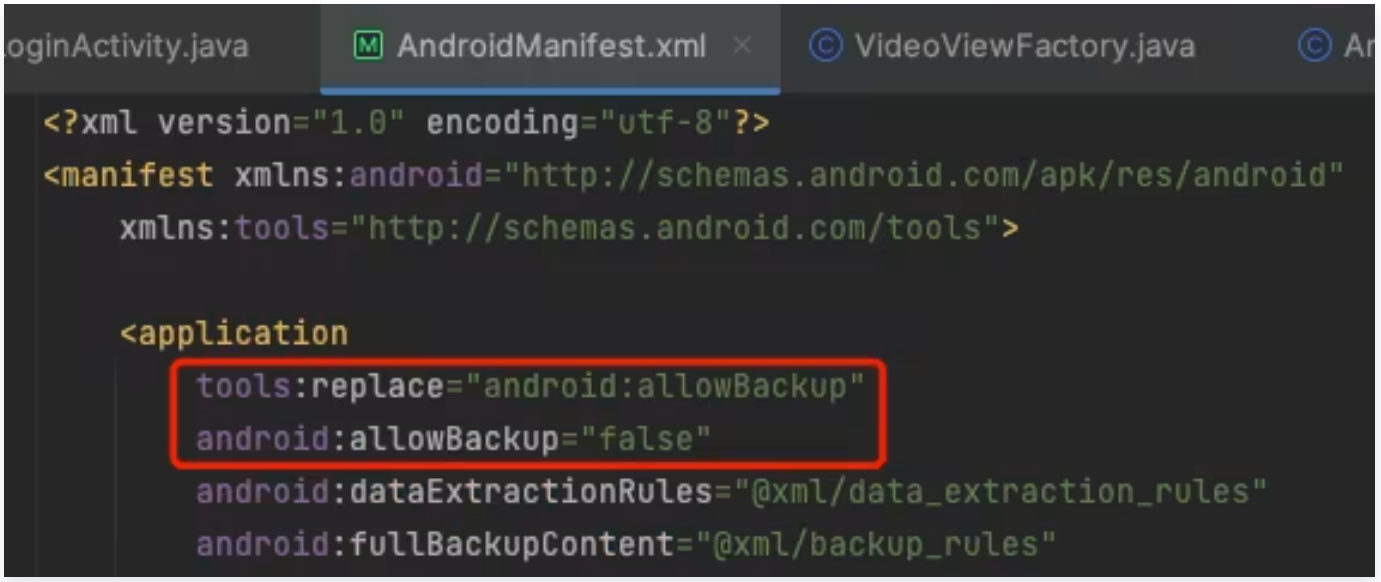
每次进房都需要调用登录吗?

不需要。通常您只需要完成一次 `TUILogin.login` 调用即可，我们建议您将 `TUILogin.login` 和 `TUILogin.logout` 与自己的登录业务关联。

集成代码后产生如下图所示编译报错 allowBackup 异常，如何处理?

```
Manifest merger failed : Attribute application@allowBackup value=(false) from AndroidManifest.xml:7:9-36
    is also present at [com.github.yyued:SVGAPlayer-Android:2.6.1] AndroidManifest.xml:12:9-35 value=(true).
    Suggestion: add 'tools:replace="android:allowBackup"' to <application> element at AndroidManifest.xml:5:5-53:19 to override.
```

- **问题原因**：多个模块的 `AndroidManifest.xml` 中都配置了 `allowBackup` 属性，造成冲突。
- **解决方法**：您可以在您工程的 `AndroidManifest.xml` 文件中删除 `allowBackup` 属性或将该属性改为 `false`，表示关闭备份和恢复功能；并在 `AndroidManifest.xml` 文件的 `application` 节点中添加 `tools:replace="android:allowBackup"` 表示覆盖其他模块的设置，使用您自己的设置。修复示例如图所示：



集成代码后还需要添加麦克风权限的声明吗?

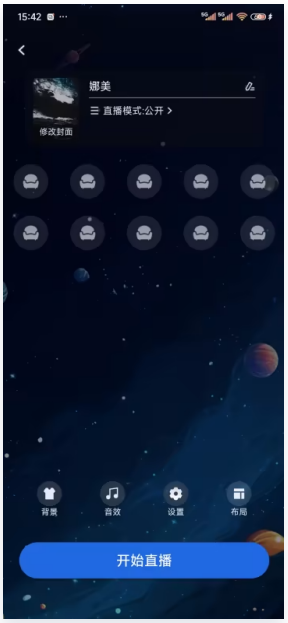
不需要，`TUILiveKit` 中已经内置了麦克风权限的声明，在接入过程中您无需再关心权限的声明。

准备工作（iOS）

最近更新时间：2025-11-21 14:15:33

功能预览

TUILiveKit 是一个功能全面的语聊房组件，集成后可快速实现以下功能模块：

主播准备页	主播开播页	直播列表	观众观看页
			

准备工作

开通服务

在使用 TUILiveKit 前，请先参考 开通服务，领取 TUILiveKit 体验版或者开通付费版。

环境要求

- **Xcode**：需使用 Xcode 15 或更高版本。
- **iOS 系统**：支持 iOS 13.0 或更高版本的设备。
- **CocoaPods 环境**：已安装 CocoaPods 环境。如果您尚未安装，请参见 CocoaPods官网安装，或按以下步骤操作：
  - **使用 gem 安装 CocoaPods**：在终端中执行 `sudo gem install cocoapods` 命令进行安装。

> **提示：**

> `sudo gem install cocoapods` 安装过程中可能需要输入电脑密码，按提示输入管理员密码即可。

代码集成

步骤 1：通过 CocoaPods 导入组件

1. 添加 Pod 依赖：
   - 若项目已有 Podfile 文件。

     在您项目的 `Podfile` 文件中添加 `pod 'TUILiveKit'` 依赖。例如：

```
target 'YourProjectTarget' do
  # 其他已有的 Pod 依赖...
  # 添加 pod 'TUILiveKit' 依赖
  pod 'TUILiveKit'

end
```

   - 若项目没有 Podfile 文件。

     在终端中通过 `cd` 命令切换到您的 `.xcodeproj` 目录下，然后执行 `pod init` 命令创建 `Podfile` 文件，创建完成后，在您的 `Podfile` 文件中添加 `pod 'TUILiveKit'` 依赖。例如：

```
// 如果您的项目目录是 /Users/yourusername/Projects/YourProject

// 1. cd 到您的.xcodeproj 工程目录下
cd /Users/yourusername/Projects/YourProject

// 2. 执行 pod init，此命令运行完后，会在您的工程目录下生成一个 Podfile 文件。
pod init

// 3. 在生成的 Podfile 文件中添加 pod 'TUILiveKit' 依赖
  target 'YourProjectTarget' do
    # 添加 pod 'TUILiveKit' 依赖
    pod 'TUILiveKit'

  end
```

2. 安装组件：

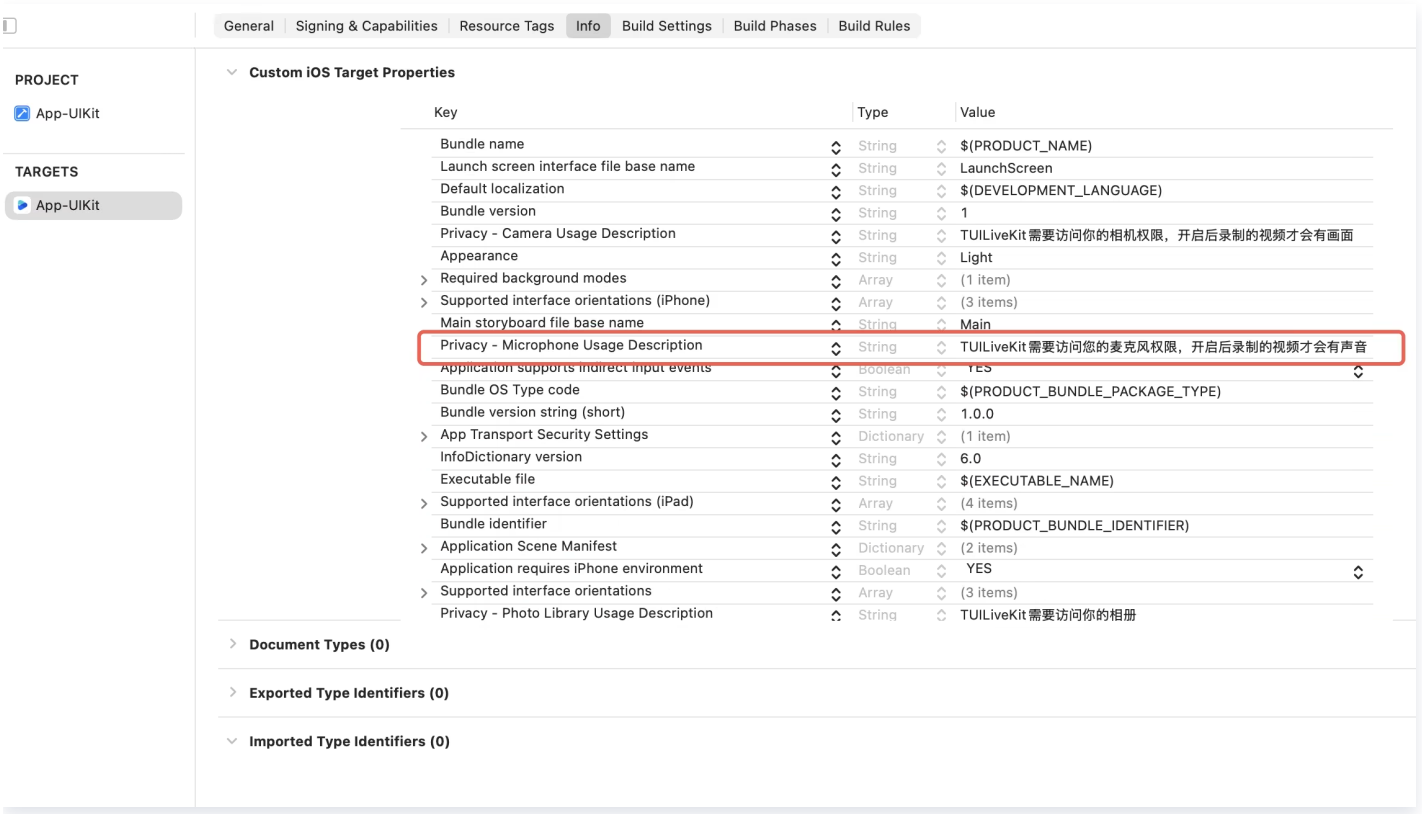
在终端中 `cd` 到 `Podfile` 文件所在的目录，然后执行以下命令安装组件。

```
pod install
```

步骤 2：工程配置

为了使用音视频功能，您的应用需要获取麦克风和摄像头的权限。请在应用的 `Info.plist` 文件中添加以下两项，并填写对应的使用说明，这些说明将在系统请求权限时向用户显示：

```
<key>NSMicrophoneUsageDescription</key>
<string>TUILiveKit需要访问您的麦克风权限，开启后录制的视频才会有声音</string>
```



完成登录

代码集成完成后，您需要完成登录，这是**使用 TUILiveKit 的关键步骤**，因为只有在登录成功后才能正常使用 TUILiveKit 的各项功能，故请您耐心检查相关参数是否配置正确：

> ⚠ **说明：**
>
> 在示例代码中，登录操作是在 `didFinishLaunchingWithOptions` 方法内完成的。但在实际项目场景下，强烈推荐**您在完成自己的用户身份验证等相关登录操作后，再调用 TUILiveKit 的登录服务**。这样可以避免因过早调用登录服务，导致业务逻辑混乱或数据不一致的问题，同时也能更好地适配您项目中现有的用户管理和权限控制体系。

Swift

```
//
//  AppDelegate.swift
//


// 1. 导入 TUICore
import TUICore


// 2. 示例代码在 didFinishLaunchingWithOptions 完成登录，推荐您在自己登录业务完成后再调用登录服务
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    // 3. 调用登录接口
    TUILogin.login(1400000001,                // 请替换为开通服务控制台的 SDKAppID
            userID: "denny",                  // 请替换为您的 UserID
            userSig: "xxxxxxxxxxx") {         // 您可以在控制台中计算一个 UserSig 并填在这个位置
      print("login success")
    } fail: { (code, message) in
      print("login failed, code: \(code), error: \(message ?? "nil")")
    }

    return true
}
```

Objective-C

```
//
//  AppDelegate.m
//
```

```
// 1. 导入 TUICore
#import <TUICore/TUILogin.h>


// 2. 示例代码在 didFinishLaunchingWithOptions 完成登录，推荐您在自己登录业务完成后再调用登录服务
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // 3. 调用登录接口
    [TUILogin login:1400000001          // 请替换为开通服务控制台的 SDKAppID
             userID:@"denny"            // 请替换为您的 UserID
            userSig:@"xxxxxxxxxxx"      // 您可以在控制台中计算一个 UserSig 并填在这个位置
               succ:^{
        NSLog(@"login success");
    } fail:^(int code, NSString * _Nullable msg) {
        NSLog(@"login failed, code: %d, error: %@", code, msg);
    }];
    return YES;
}
```

登录接口参数说明：

参数	类型	说明
SDKAppID	Int	从 控制台 获取，国内站通常是以 `140` 或 `160` 开头的 10 位整数。
userID	String	当前用户的唯一 ID，仅包含英文字母、数字、连字符和下划线。为避免多端登录冲突，**请勿使用** `1`、`123` **等简单 ID**。
userSig	String	用于腾讯云鉴权的票据。请注意： • 开发环境：您可以采用本地 `GenerateTestUserSig.genTestSig` 函数生成 userSig 或者 通过 UserSig 辅助工具 生成临时的 UserSig。 • 生产环境：为了防止密钥泄露，请务必采用服务端生成UserSig 的方式。详细信息请参考 服务端生成 UserSig 。

		更多信息请参见 如何计算及使用 UserSig 。

登录异常状态处理【可选】

`TUILogin` 提供了登录状态回调机制，方便您处理可能出现的登录异常情况，主要包括 “被踢下线” 和 “签名过期” 这两种异常状态的回调：

- **被踢下线**：用户在线情况下被踢，IM SDK 会通过 `onKickedOffline` 回调通知给您，此时可以在 UI 提示用户，并调用 `TUILogin.login` 重新登录。
- **签名过期**：用户在线期间收到 onUserSigExpired 回调，说明您之前给该用户签发的 userSig 已经过期了，这个时候如果当前用户在您后台的登录态依然有效，您可以让您的 app 向您的后台请求新的 userSig，并调用 TUILogin.login 续签登录态。

Swift

```
// YourLoginService 代表您负责登录的业务模块
class YourLoginService: NSObject {

    // 监听登录状态回调
    func addLoginListener() {
        TUILogin.add(self)
    }
    // 取消监听登录状态回调
    func removeLoginListener() {
        TUILogin.remove(self)
    }
}

// 实现登录回调 TUILoginListener
extension YourLoginService: TUILoginListener {

    // 用户被踢下线回调
    func onKickedOffline() {
      // 您的业务代码：UI 交互提示用户，然后重新登录
    }


     // 用户签名过期回调
    func onUserSigExpired() {
      // 您的业务代码：如果当前用户在您后台的登录态依然有效，您可以让您的 app 向您的后台请求新的 userSig，并调用 TUILogin.login 续签登录态。
```

```
    }
}
```

Objective-C

```
@interface YourLoginService() <TUILoginListener>

// 监听登录状态回调
- (void)addLoginListener;

// 取消监听登录状态回调
- (void)removeLoginListener;

@end

@implementation YourLoginService

// 监听登录状态回调
- (void)addLoginListener {
    [TUILogin add:self];
}

// 取消监听登录状态回调
- (void)removeLoginListener {
    [TUILogin remove:self];
}

#pragma mark - TUILoginListener

// 用户被踢下线回调
- (void)onKickedOffline {
    // 您的业务代码：UI 交互提示用户，然后重新登录
}

// 用户签名过期回调
- (void)onUserSigExpired {
    // 您的业务代码：UI 交互提示用户，然后重新登录
}
```

```
@end
```

下一步

恭喜您，现在您已经成功集成了**语聊房**组件并完成了登录。接下来，您可以实现**主播开播**、**观众观看**、**直播列表**等功能，可参考下表：

功能	描述	集成指引
主播开播	主播创建语聊房全流程功能，包括开播前的准备和开播后的各种互动。	主播开播
观众观看	观众进入语聊房后收听，实现上麦、弹幕显示等功能。	观众观看
直播列表	展示语聊房列表界面和功能，包含语聊房列表，房间信息展示功能。	直播列表

常见问题

pod install 执行后本地安装找不到 TUILiveKit 最新版本?

如果无法安装 TUILiveKit 最新版本，请按以下步骤操作：

1. 在 Podfile 所在目录下删除 `Podfile.lock` 和 `Pods`，您可以选择手动删除或终端执行以下命令：

```
// cd 到 Podfile 所在目录下

rm -rf Pods/
rm Podfile.lock
```

2. 在 Podfile 所在目录下执行 `pod install --repo-update`，示例如下：

```
// cd 到 Podfile 所在目录下

pod install --repo-update
```

每次进房都需要调用登录吗?

不需要。通常您只需要完成一次 `TUILogin.login` 调用即可，我们建议您将 `TUILogin.login` 和 `TUILogin.logout` 与自己的登录业务关联。

Podfile 文件有没有示例配置可以参考?

您可以参考 GitHub TUILiveKit Example 工程 `Podfile` 示例文件。

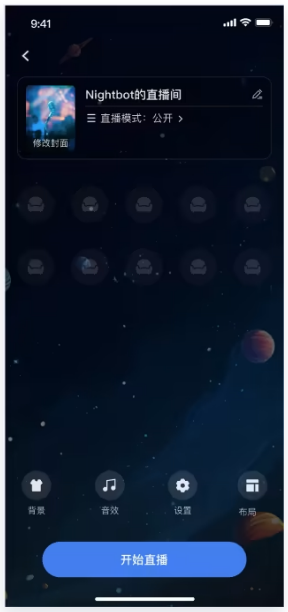
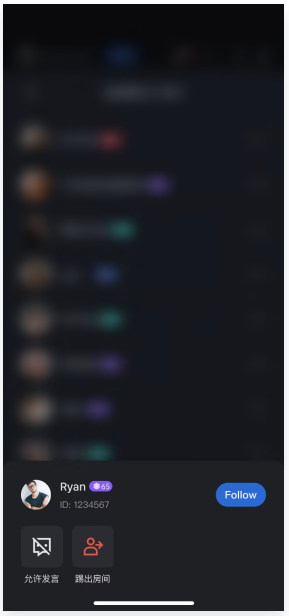
主播开播
主播开播（Android）

最近更新时间：2025-11-21 14:15:33

`TUILiveKit` 语聊房为纯音频直播场景提供了**开箱即用的全功能界面**。它支持快速搭建主播开播所需的核心能力，让您无需关注复杂的 UI 与麦位管理逻辑，即可高效集成语聊房开播流程。

功能概览

- **开播前预览：** 支持主播开播前设置房间名称、封面等个性化配置。
- **麦位管理：** 支持上麦、下麦、禁麦、锁麦等多种麦位管理操作。
- **观众互动：** 支持弹幕、礼物、点赞 等丰富的直播互动形式。
- **直播间管理：** 支持在线用户列表展示，以及直播间内的禁言、踢人 等多种管理操作。

开播前预览	麦位管理	观众互动	直播间管理
			

快速接入

步骤 1：开通服务

参考 开通服务 文档开通「体验版」或「大规模直播版」套餐。

步骤 2：代码集成

参考 准备工作 接入 `TUILiveKit`。

步骤 3：创建并展示语聊房视图控制器

`VoiceRoomActivity` 组件已内置了语聊房场景的主播端完整 `UI` 与业务逻辑。您只需启动该 `Activity`，即可快速实现主播开播功能。推荐在您 `App` 的“开始直播”按钮点击事件中，执行以下逻辑：

```
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import com.tencent.cloud.tuikit.engine.room.TUIRoomDefine.SeatMode
import
com.trtc.uikit.livekit.voiceroom.view.TUIVoiceRoomFragment.RoomBehavior.
PREPARE_CREATE
import
com.trtc.uikit.livekit.voiceroom.view.TUIVoiceRoomFragment.RoomParams
import com.trtc.uikit.livekit.voiceroom.view.VoiceRoomActivity

// YourActivity 代表您发起直播的页面
class YourActivity : AppCompatActivity() {

    // 响应“开始直播”按钮点击事件
    fun onStartVoiceRoomClicked() {

        // 1. 配置房间参数 (RoomParams)
        // RoomParams 需要实现 Parcelable 接口
        val params = RoomParams().apply {
            maxSeatCount = 10 // 最大麦位数量
            seatMode = SeatMode.APPLY_TO_TAKE // 上麦模式
        }

        // 2. 准备 Intent，并传入必要参数
        val roomId = "test_voice_room_id"
        val intent = Intent(this, VoiceRoomActivity::class.java).apply {
            putExtra(VoiceRoomActivity.INTENT_KEY_ROOM_ID, roomId)
            // behavior: PREPARE_CREATE 代表先进入开播前预览页
            putExtra(VoiceRoomActivity.INTENT_KEY_CREATE_ROOM_PARAMS,
params)
            putExtra(VoiceRoomActivity.INTENT_KEY_ROOM_BEHAVIOR,
PREPARE_CREATE.ordinal)
        }

        // 3. 跳转到语聊房页面
```

```
            startActivity(intent)
        }
    }
```

Intent Extra 参数说明：

参数名	类型	描述
`VoiceRoomActivity.INTENT_KEY_ROOM_ID`	`string`	全局唯一的直播间 `Id` 。
`VoiceRoomActivity.INTENT_KEY_ROOM_BEHAVIOR`	`Int`	进房行为： • `AUTO_CREATE`：自动创建直播间并进房。 • `PREPARE_CREATE`：先进入开播前预览页，用户点击“开始直播”后创建直播间并进房。 • `JOIN`：观众进房。
`roomParams`	`RoomParams`	主播开播参数，详见下一节。

RoomParams 参数说明：

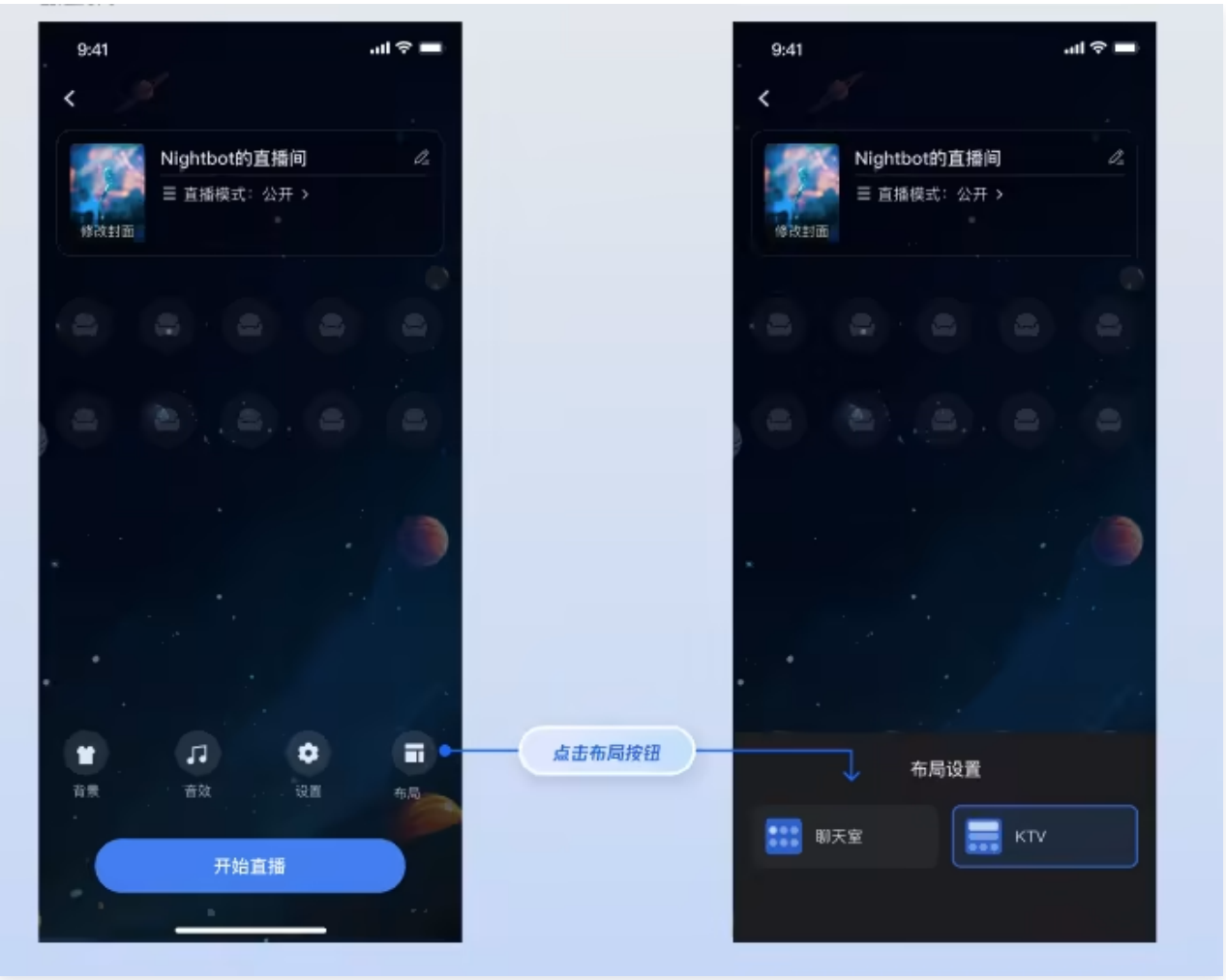
参数名	类型	描述
`maxSeatCount`	`Int`	直播间最大麦位数。
`seatMode`	`TUIRoomDefine.SeatMode`	观众的上麦模式： • `APPLY_TO_TAKE`：观众需要申请，主播同意后上麦 • `FREE_TO_TAKE`：观众自由上麦，无需主播同意。

自定义您的界面布局

`TUILiveKit` 提供了界面定制能力，以满足多样化的业务需求。您不仅可以选择不同的布局模板，还可以轻松替换界面中的文案和图标。

直播布局模板选择

`TUILiveKit` 语聊提供 2 **种布局样式**，您可在主播开播前预览页的 **UI 交互布局按钮**选择合适样式：



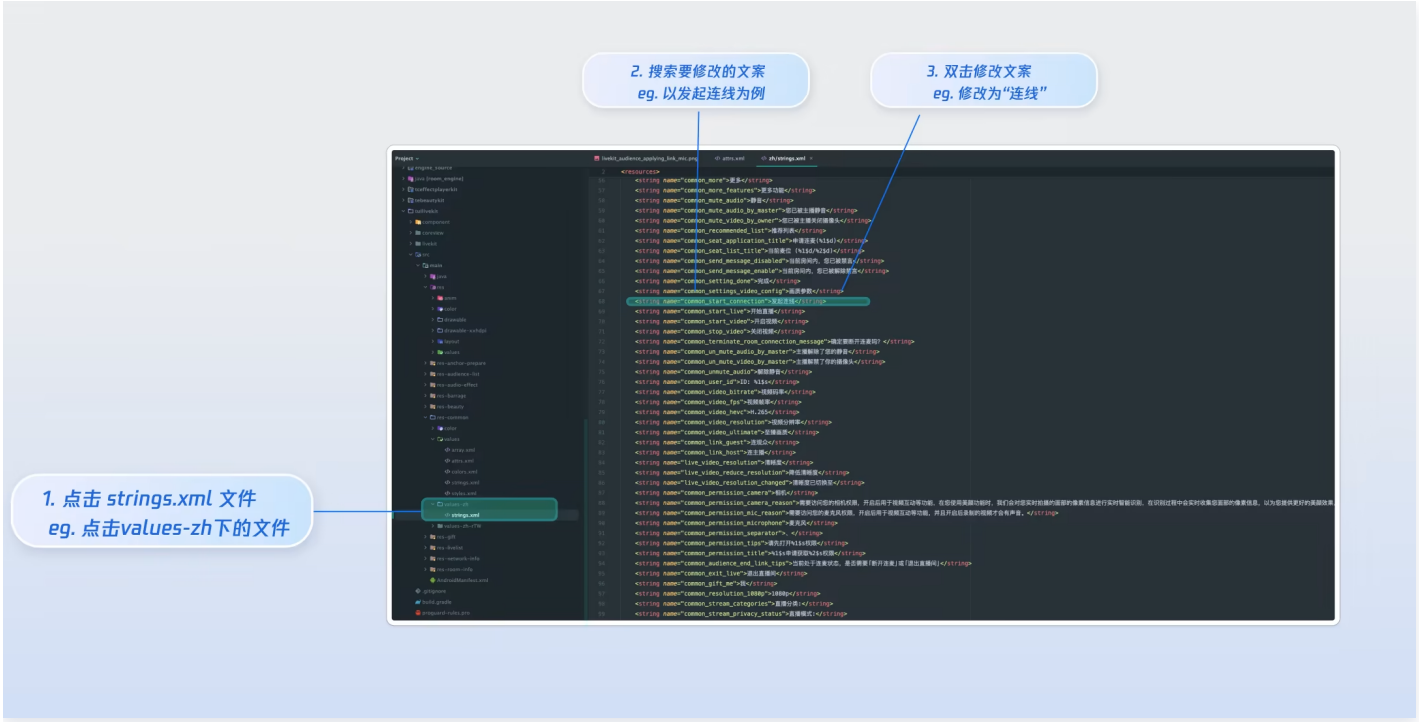
布局概览：

名称	聊天室	KTV
描述	默认布局，页面只显示麦位网格图。	麦位网格图上方显示KTV歌曲播放器。

预览		

文案修改

`TUILiveKit` 使用 `Android` 标准的 字符串资源 (`strings.xml`) 来管理 `UI` 所需的文案显示。您可以直接修改 `tuilivekit` 模块中的 `strings.xml` 文件，以实现文案自定义。



图标替换

`TUILiveKit` 使用 `drawable` 资源来管理 UI 所需的图片资源。您可以直接在 `tuilivekit` 模块的 `res/drawable` 目录下替换同名图片资源，以实现图标替换。

下一步

恭喜您，现在您已经成功集成了 **主播开播** 。接下来，您可以实现**观众观看**、**直播列表**等功能，可参考下表：

功能	描述	集成指引
观众观看	实现观众进入主播的语聊房后收听，实现上麦、弹幕显示等功能	观众观看
直播列表	展示直播列表界面和功能，包含直播列表，房间信息展示功能	直播列表

常见问题

开播后没有声音?

请检查是否已授予 `App` 麦克风权限。您可以前往手机 **应用信息 > 权限 > 麦克风**，检查“麦克风”权限是否已开启。

点击开播按钮无法开播，提示“未登录”？

参考 完成登录， 确认已完成登录功能接入。

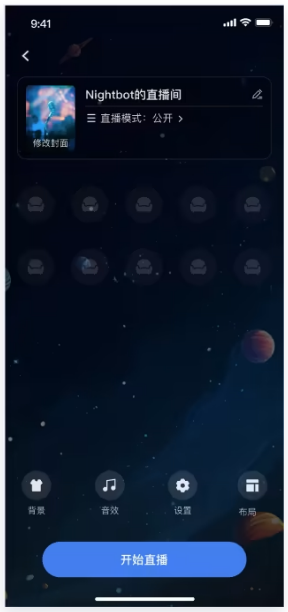
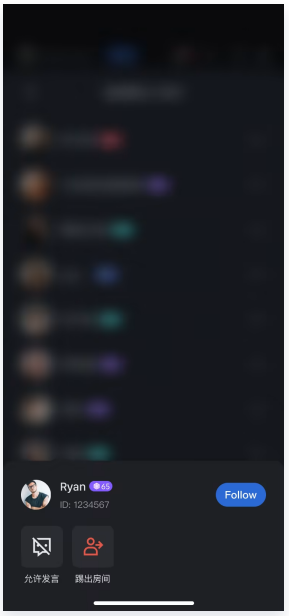
主播开播（iOS）

最近更新时间：2025-11-21 14:15:33

`TUILiveKit` 语聊房为纯音频直播场景提供了**开箱即用的全功能界面**。它支持快速搭建主播开播所需的核心能力，让您无需关注复杂的 UI 与麦位管理逻辑，即可高效集成语聊房开播流程。

功能概览

- **开播前预览：** 支持主播开播前设置房间名称、封面等个性化配置。
- **麦位管理：** 支持上麦、下麦、禁麦、锁麦等多种麦位管理操作。
- **观众互动：** 支持弹幕、礼物、点赞等丰富的直播互动形式。
- **直播间管理：** 支持在线用户列表展示，以及直播间内的禁言、踢人等多种管理操作。

开播前预览	麦位管理	观众互动	直播间管理
			

快速接入

步骤 1：开通服务

参考 开通服务 文档开通「体验版」或「大规模直播版」套餐。

步骤 2：代码集成

参考 准备工作 接入 TUILiveKit。

步骤 3：创建并展示语聊房视图控制器

`TUIVoiceRoomViewController` 组件已内置了语聊房场景的主播端完整 UI 与业务逻辑。您只需创建并展示该视图控制器，即可快速实现主播开播功能。推荐在您 App 的“开始直播”按钮点击事件中，执行以下逻辑：

```
import TUILiveKit
import UIKit

// YourViewController 代表您发起直播的视图控制器
class YourViewController: UIViewController {

    // 响应“开始直播”按钮点击事件
    @objc func onStartVoiceRoomClicked() {

        // 1. 配置房间参数 (RoomParams)
        var params = RoomParams()
        params.maxSeatCount = 10 // 最大麦位数量
        params.seatMode = .applyToTake // 上麦模式

        // 2. 实例化语聊房控制器
        let roomId = "test_voice_room_id"
        // 2. 实例化语聊房控制器
        let voiceRoomVC = TUIVoiceRoomViewController(roomId: roomId,
behavior: .prepareCreate, roomParams: params)
        voiceRoomVC.modalPresentationStyle = .fullScreen

        // 3. 跳转到语聊房页面
        present(voiceRoomVC, animated: true)
    }
}
```

TUIVoiceRoomViewController参数说明：

参数名	类型	描述
`roomId`	`string`	全局唯一的直播间 ID。
`behavior`	`RoomBehavior`	进房行为： • `autoCreate`：自动创建直播间并进房。 • `prepareCreate`：先进入开播前预览页，用户点击“开始直播”后创建直播间并进房。 • `join`：观众进房。

roomParams	RoomParams?	主播开播参数，详细参见下表。

RoomParams 参数说明：

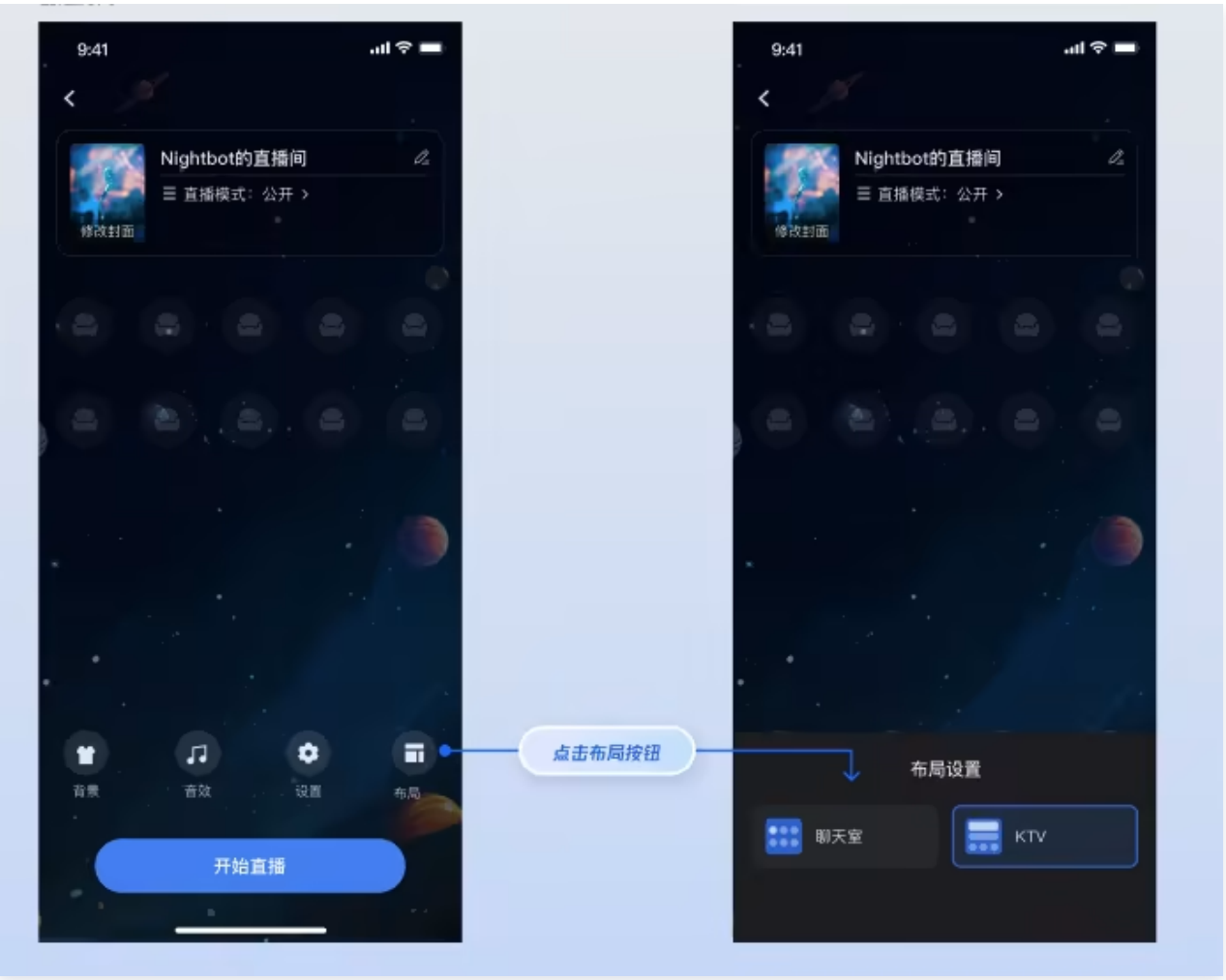
参数名	类型	描述
maxSeatCount	Int	直播间最大麦位数。
seatMode	TUISeatMode	观众的上麦模式： • applyToTake：观众需要申请，主播同意后上麦 • freeToTake：观众自由上麦，无需主播同意。

自定义您的界面布局

TUILiveKit 提供了界面定制能力，以满足多样化的业务需求。您不仅可以选择不同的布局模板，还可以轻松替换界面中的文案和图标。

直播布局模板选择

TUILiveKit 语聊提供 2 种布局样式，您可在主播开播前预览页的 UI 交互「布局」入口 选择合适样式：



布局概览：

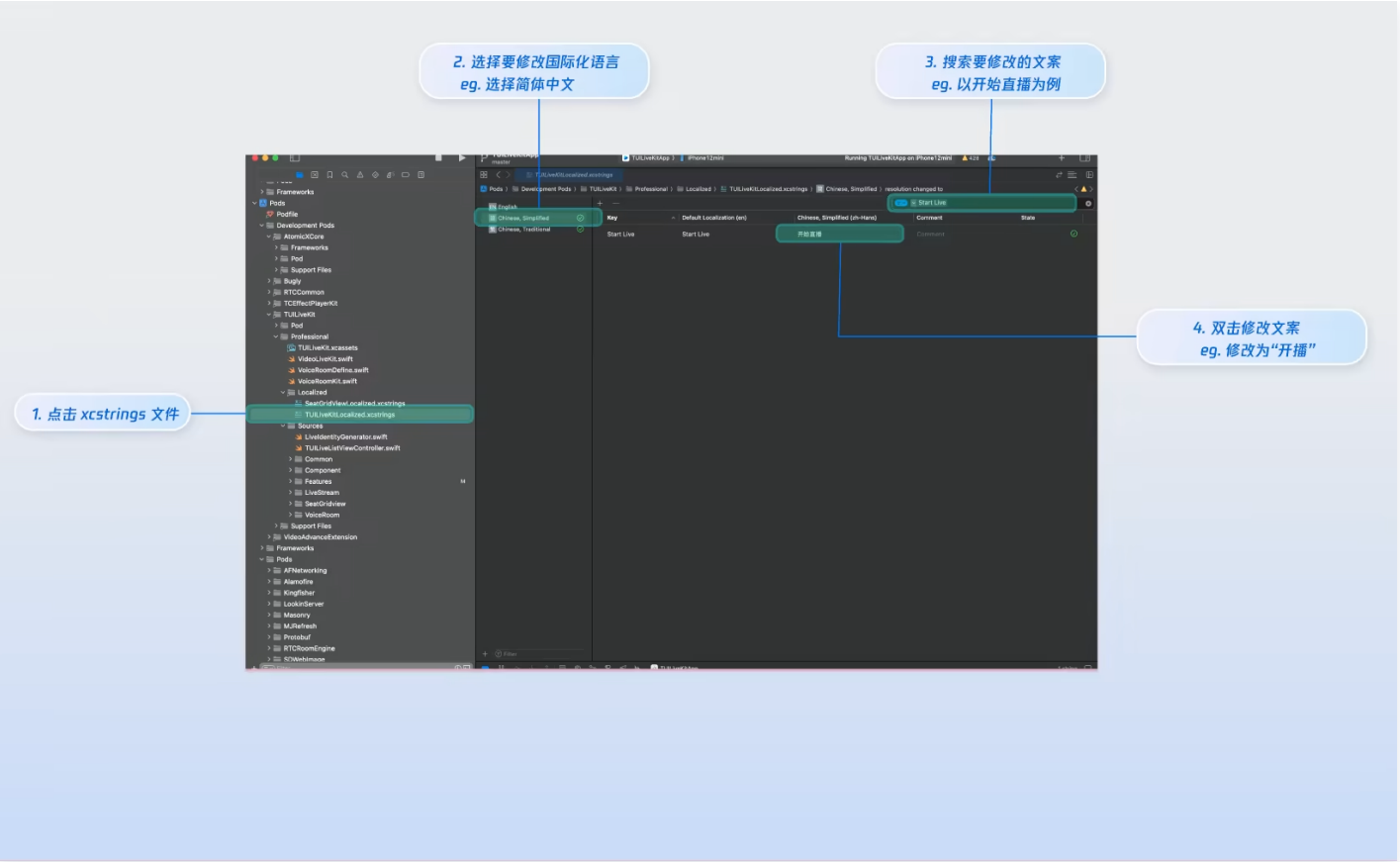
名称	聊天室	KTV
描述	默认布局，页面只显示麦位网格图。	麦位网格图上方显示 KTV 歌曲播放器。

预览		

文案修改

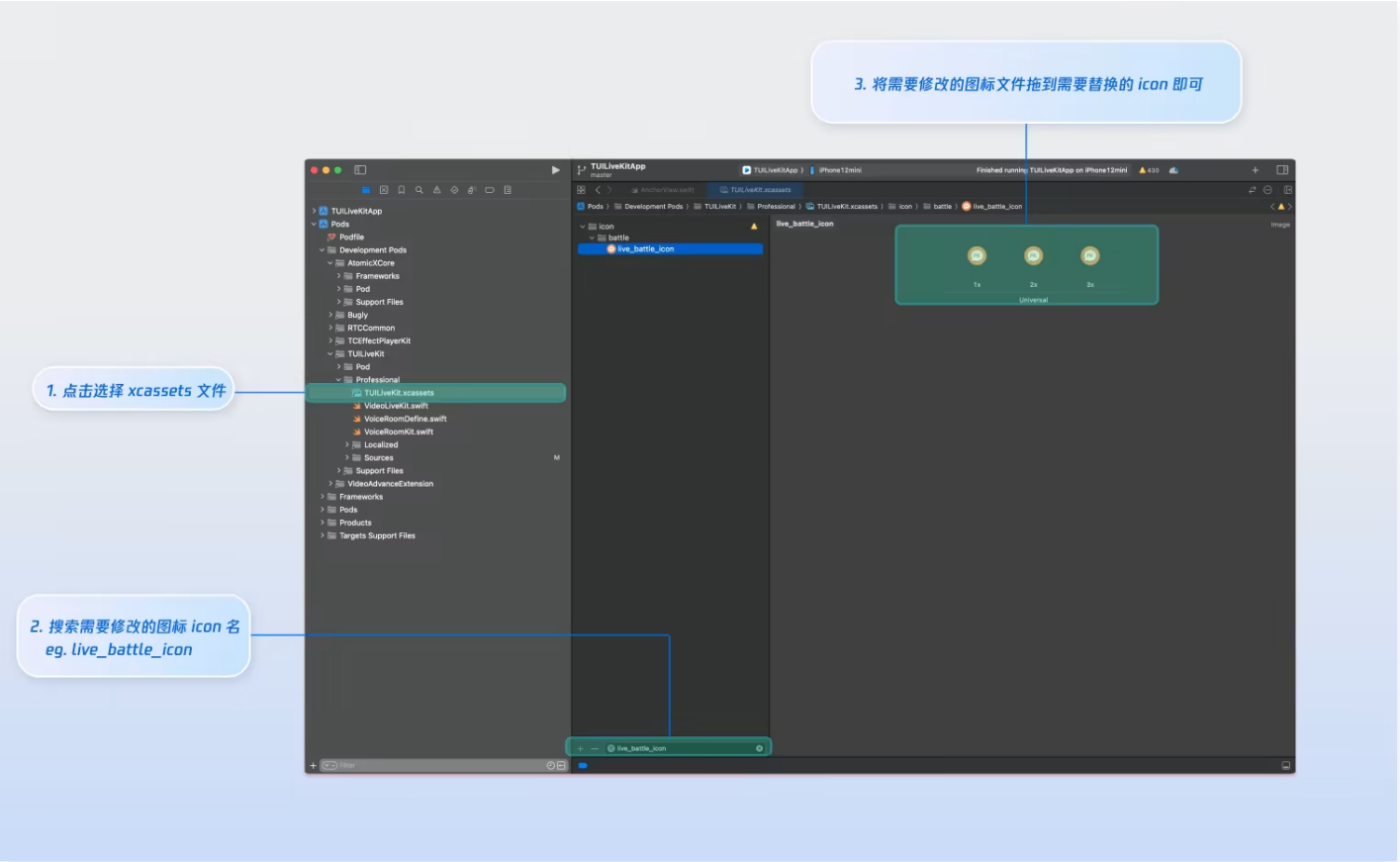
TUILiveKit 使用更为方便的 Apple Strings Catalog 工具来管理 UI 所需的文案显示，您可以很直观的通过 Xcode 视图化管理工具修改需要调整的文案：

> **说明：**
> Apple Strings Catalog（.xcstrings）是在 Xcode 15 中引入的本地化格式。它增强了开发者管理本地化字符串的方式，支持处理复数、设备特定变体等的结构化格式。这种格式正成为管理 iOS 和 macOS 应用程序本地化的推荐方法。



图标替换

TUILiveKit 使用 `TUILiveKit.xcassets` 管理 UI 所需的图片资源，您可以借助 Xcode 图形化工具快速替换图标。



下一步

恭喜您，现在您已经成功集成了**主播开播** 。接下来，您可以实现**观众观看**、**直播列表**等功能，可参考下表：

功能	描述	集成指引
观众观看	实现观众进入主播的语聊房后收听，实现上麦、弹幕显示等功能	观众观看
直播列表	展示直播列表界面和功能，包含直播列表，房间信息展示功能	直播列表

常见问题

开播后没有声音?

请前往手机的**设置** > **App** > **麦克风**，检查麦克风权限是否开启。

点击开播按钮无法开播，提示“未登录”？

参考 登录指引，确认已完成登录功能接入。

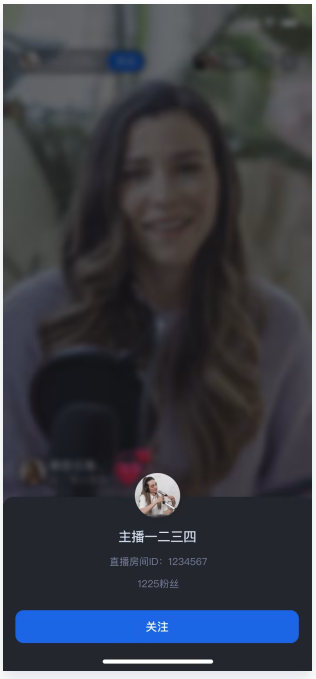
观众观看
观众观看（Android）

最近更新时间：2025-11-21 14:15:33

`TUILiveKit` 语聊房为纯音频直播场景提供了**开箱即用的全功能界面**。它支持快速搭建观众观看、上麦互动所需的核心能力，让您无需关注复杂的 `UI` 与麦位管理逻辑。

功能概览

- **收听直播：** 清晰、流畅地收听主播的实时音频直播。
- **上麦互动：** 申请上麦，与主播进行音频连麦互动。
- **直播间信息：** 查看直播间公告，及在线观众列表等信息。
- **观众互动：** 支持弹幕、礼物、点赞 等丰富的直播互动形式。

收听直播	上麦互动	直播间信息	观众互动
			

快速接入

步骤1：开通服务

参考 开通服务 文档开通「体验版」或「大规模直播版」套餐。

步骤2：代码集成

参考 准备工作 接入 TUILiveKit 。

步骤3：创建并展示语聊房视图控制器

VoiceRoomActivity 组件已内置了语聊房场景的观众端完整 UI 与业务逻辑。您只需启动该 Activity ，即可快速实现观众进入直播间功能。通常在 直播列表 点击房间时，您需要跳转到观众观看页，可参考如下代码示例：

```
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.trtc.uikit.livekit.voiceroom.VoiceRoomActivity
import com.trtc.uikit.livekit.voiceroom.VoiceRoomDefine

// YourLiveListActivity 代表您的直播列表页面
class YourLiveListActivity : AppCompatActivity() {

    // 响应“点击直播间”
    fun onJoinVoiceRoomClicked(roomId: String) {

        // 1. 准备 Intent，并传入必要参数
        //    - roomId: 您要加入的直播间 ID
        //    - behavior: VoiceRoomDefine.RoomBehavior.JOIN 代表以“观众”身份加入房间
        val intent = Intent(this, VoiceRoomActivity::class.java).apply {
            putExtra(VoiceRoomActivity.INTENT_KEY_ROOM_ID, roomId)
            putExtra(VoiceRoomActivity.INTENT_KEY_ROOM_BEHAVIOR, JOIN.ordinal)
        }

        // 2. 跳转到语聊房页面
        startActivity(intent)
    }
}
```

Intent Extra 参数说明：

参数名	类型	描述
VoiceRoomActivity.INTENT_KEY	string	全局唯一的直播间 Id 。

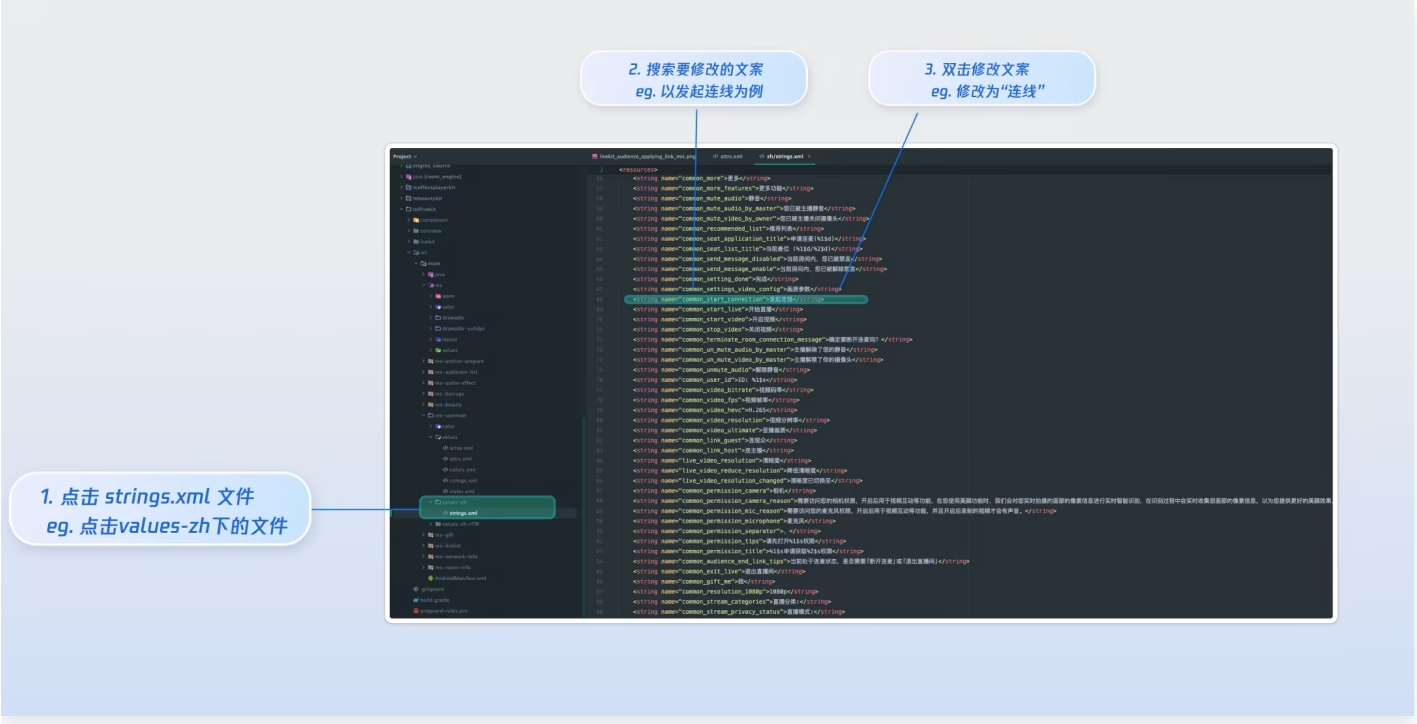
_ROOM_ID		
VoiceRoomActivity.INTENT_KEY_ROOM_BEHAVIOR	Int	进房行为： • AUTO_CREATE：自动创建直播间并进房。 • PREPARE_CREATE：先进入开播前预览页，用户点击“开始直播”后创建直播间并进房。 • JOIN：观众进房。

自定义您的界面

TUILiveKit 提供了界面定制能力，以满足多样化的业务需求。 您可以轻松自定义界面中的文案和图标。

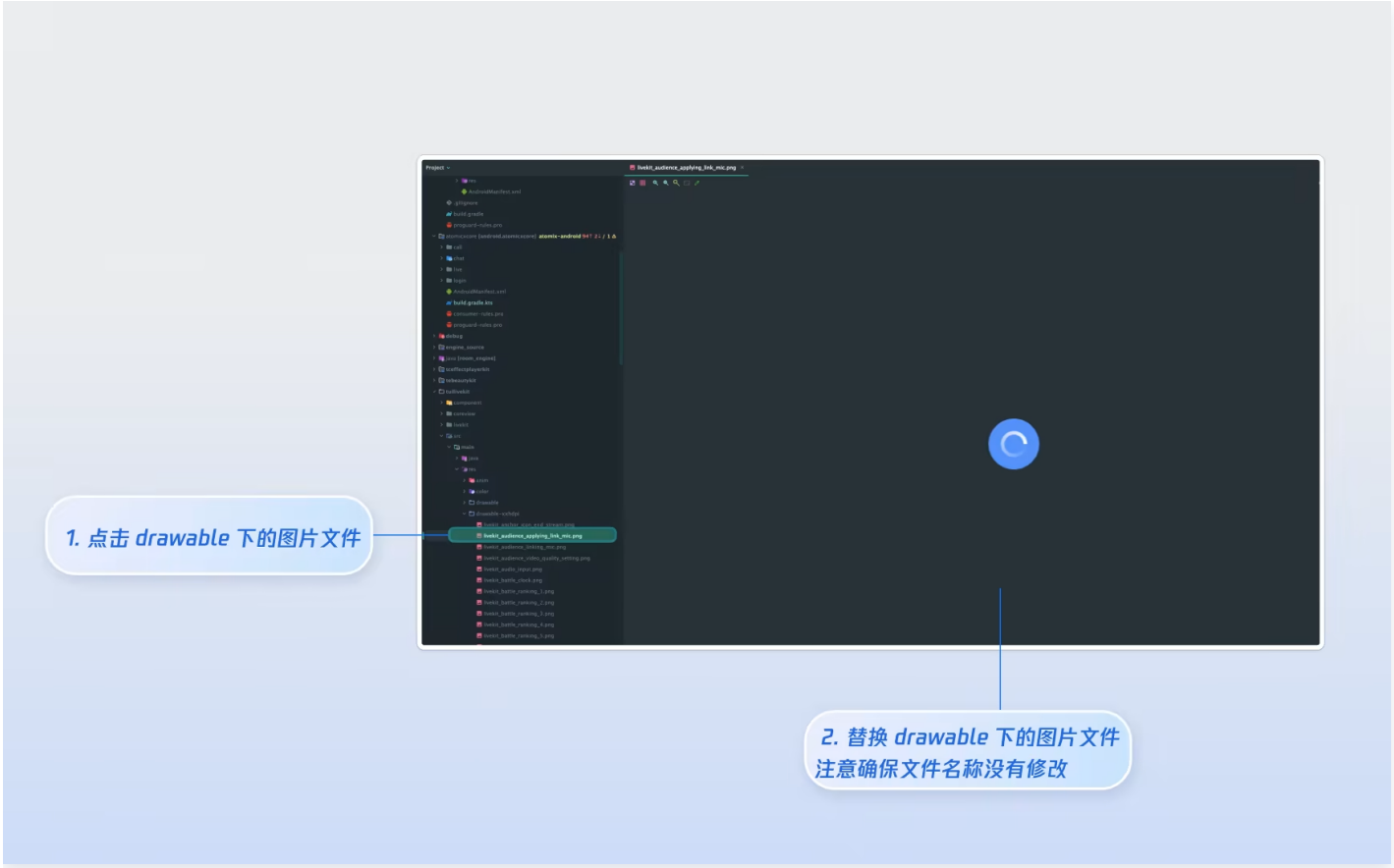
文案修改

TUILiveKit 使用 Android 标准的 字符串资源 (strings.xml) 来管理 UI 所需的文案显示。您可以直接修改 tuilivekit 模块中的 strings.xml 文件，以实现文案自定义。



图标替换

TUILiveKit 使用 drawable 资源来管理 UI 所需的图片资源。您可以直接在 tuilivekit 模块的 res/drawable 目录下替换同名图片资源，以实现图标替换。



下一步

恭喜您，现在您已经成功集成了观众观看。接下来，您可以实现**主播开播**、**直播列表**等功能，可参考下表：

功能	描述	集成指引
主播开播	主播开播全流程功能，包括开播前的准备和开播后的各种互动	主播开播
直播列表	展示直播列表界面和功能，包含直播列表，房间信息展示功能	直播列表

常见问题

观众上麦后，说话没有声音?

请检查是否已授予 `App` 麦克风权限。您可以前往手机 **应用信息 > 权限 > 麦克风**，检查“麦克风”权限是否已开启。

观众发送弹幕时，直播间内其他观众看不到弹幕内容?

- 原因 1： 先检查网络连接，确保观众设备网络正常。
- 原因 2： 该观众被主播禁言，无法发送弹幕。
- 原因 3： 观众的弹幕内容涉及关键词屏蔽，请确认观众发送的弹幕内容是否符合直播间规则。

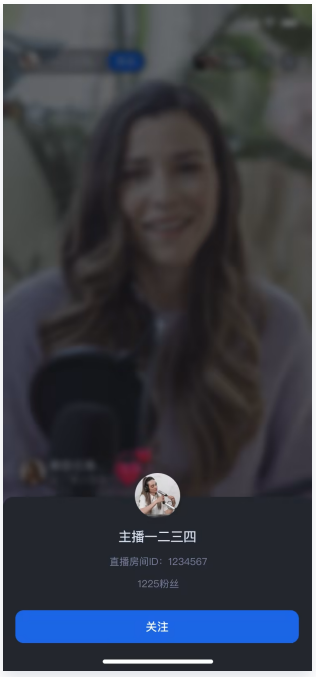
观众观看（iOS）

最近更新时间：2025-11-21 14:15:33

`TUILiveKit` 语聊房为纯音频直播场景提供了**开箱即用的全功能界面**。它支持快速搭建观众观看、上麦互动所需的核心能力，让您无需关注复杂的 UI 与麦位管理逻辑。

功能概览

- **收听直播：** 清晰、流畅地收听主播的实时音频直播。
- **上麦互动：** 申请上麦，与主播进行音频连麦互动。
- **直播间信息：** 查看直播间公告，及在线观众列表等信息。
- **观众互动：** 支持弹幕、礼物、点赞等丰富的直播互动形式。

收听直播	上麦互动	直播间信息	观众互动
			

快速接入

步骤 1：开通服务

参考 开通服务 文档开通「体验版」或「大规模直播版」套餐。

步骤 2：代码集成

参考 准备工作 接入 TUILiveKit。

步骤 3：创建并展示语聊房视图控制器

`TUIVoiceRoomViewController` 组件已内置了语聊房场景的观众端完整 UI 与业务逻辑。您只需创建并展示该视图控制器，即可快速实现观众进入直播间功能。通常在 直播列表 点击房间时，您需要导航跳转到观众观看页，可参考如下代码示例：

```
import TUILiveKit
import UIKit

// YourLiveListViewController 代表您的直播列表视图控制器
class YourLiveListViewController: UIViewController {

    // 响应“点击直播间”
    @objc func onJoinVoiceRoomClicked(roomId: String) {

        // 1. 实例化语聊房控制器
        //    - roomId: 您要加入的直播间 ID
        //    - behavior: .join 代表以“观众”身份加入房间
        let voiceRoomVC = TUIVoiceRoomViewController(roomId: roomId,
behavior: .join)
        voiceRoomVC.modalPresentationStyle = .fullScreen

        // 2. 跳转到语聊房页面
        present(voiceRoomVC, animated: true)
    }
}
```

TUIVoiceRoomViewController 参数说明：

参数名	类型	描述
`roomId`	`string`	全局唯一的直播间 ID。
`behavior`	`RoomBehavior`	进房行为： • `autoCreate`：自动创建直播间并进房。 • `prepareCreate`：先进入开播前预览页，用户点击“开始直播”后创建直播间并进房。 • `join`：观众进房。

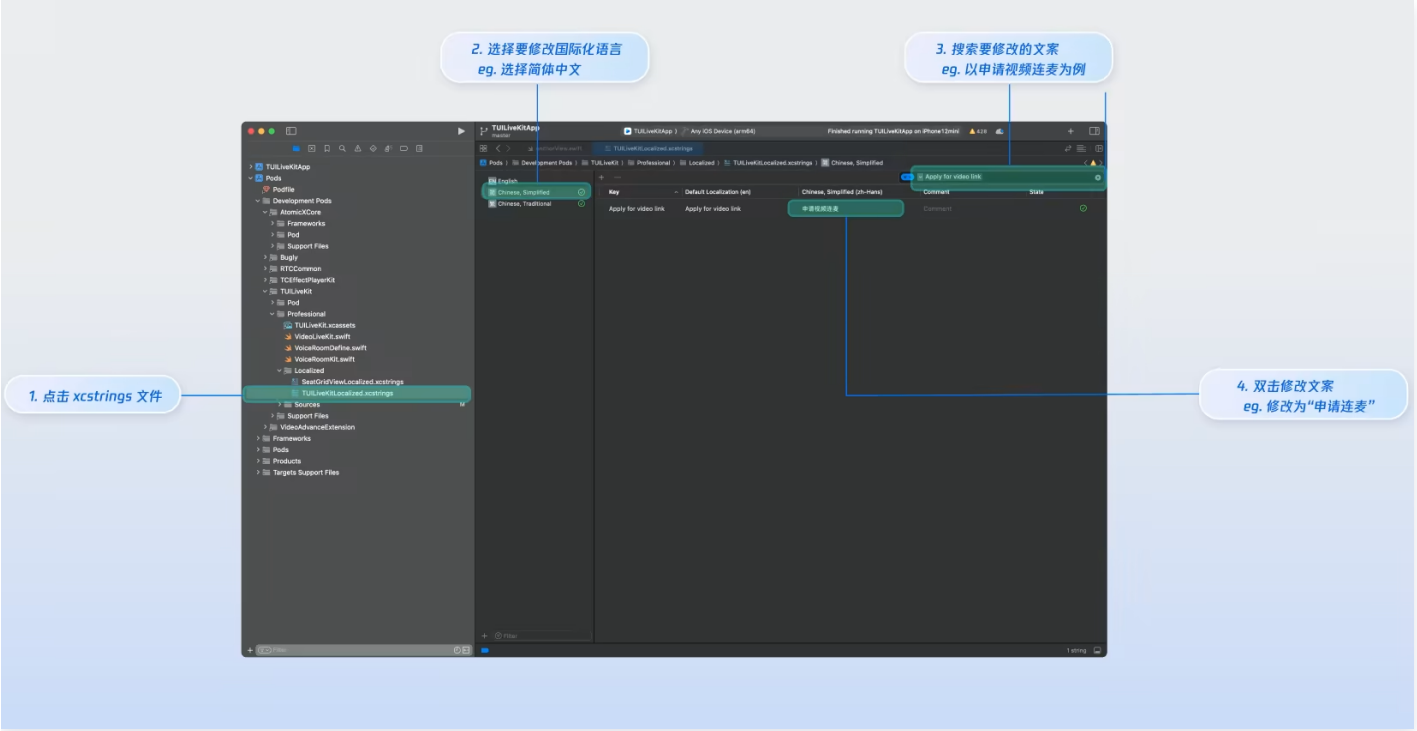
自定义您的界面

`TUILiveKit` 提供了界面定制能力，以满足多样化的业务需求。 您可以轻松自定义界面中的文案和图标。

文案修改

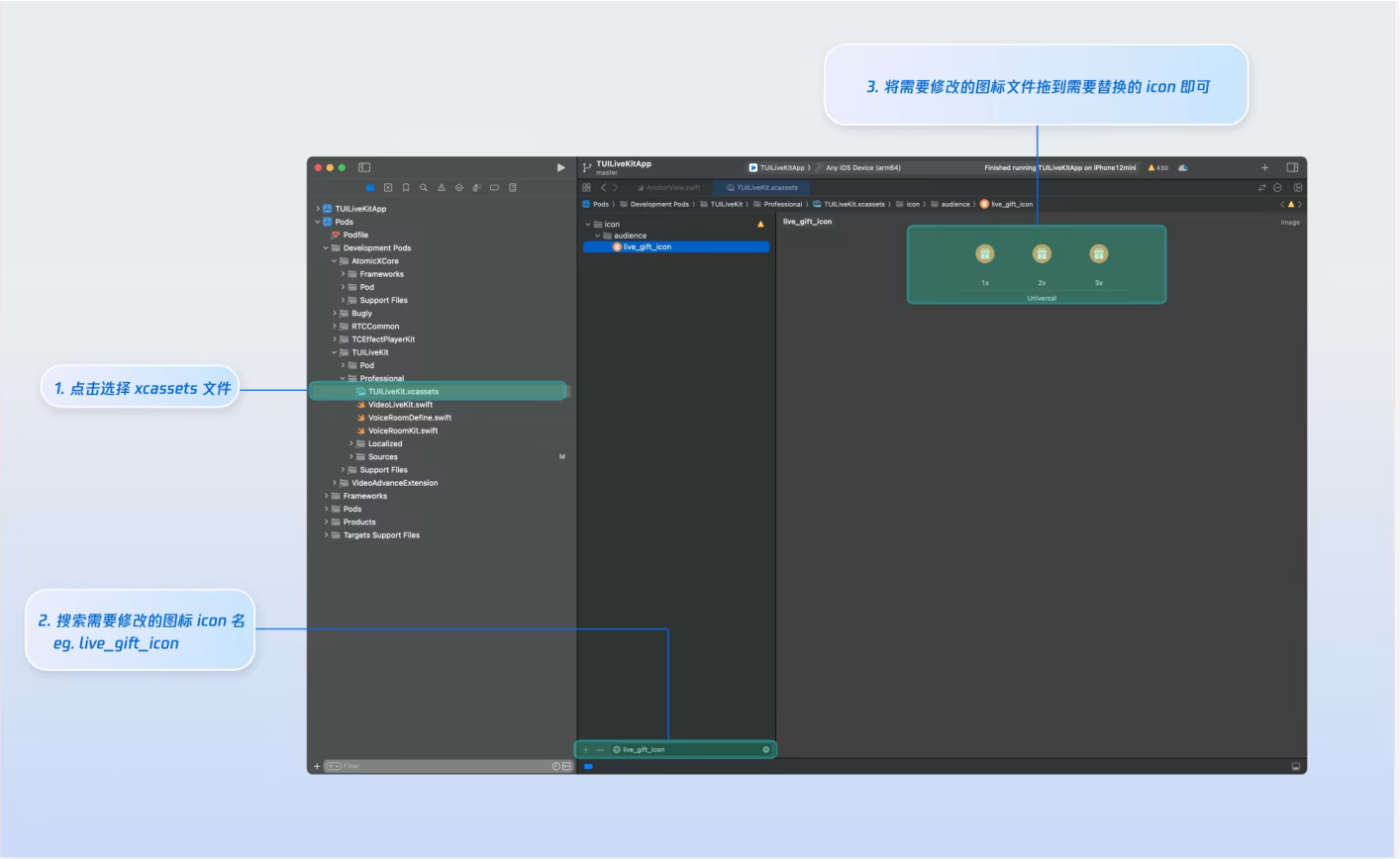
TUILiveKit 使用更为方便的 Apple Strings Catalog 工具来管理 UI 所需的文案显示，您可以很直观的通过 Xcode 视图化管理工具修改需要调整的文案：

> **说明：**
> Apple Strings Catalog（.xcstrings）是在 Xcode 15 中引入的本地化格式。它增强了开发者管理本地化字符串的方式，支持处理复数、设备特定变体等的结构化格式。这种格式正成为管理 iOS 和 macOS 应用程序本地化的推荐方法。



图标替换

TUILiveKit 使用 `TUILiveKit.xcassets` 管理 UI 所需的图片资源，您可以借助 Xcode 图形化工具快速替换图标。



下一步

恭喜您，现在您已经成功集成了观众观看功能。接下来，您可以实现**主播开播**、**直播列表**等功能，可参考下表：

功能	描述	集成指引
主播开播	主播开播全流程功能，包括开播前的准备和开播后的各种互动	主播开播
直播列表	展示直播列表界面和功能，包含直播列表，房间信息展示功能	直播列表

常见问题

观众上麦后，说话没有声音?

请前往手机**设置**>App>**麦克风**，检查麦克风权限是否开启。

观众发送弹幕时，直播间内其他观众看不到弹幕内容?

- 原因 1： 先检查网络连接，确保观众设备网络正常。
- 原因 2： 该观众被主播禁言，无法发送弹幕。
- 原因 3： 观众的弹幕内容涉及关键词屏蔽，请确认观众发送的弹幕内容是否符合直播间规则。

直播列表
直播列表（Android）

最近更新时间：2025-11-21 14:15:33

功能预览

本文对 TUILiveKit 中的**语聊房直播列表页面**进行了详细的介绍。您可以在已有项目中直接参考本文档集成我们开发好的直播列表页面，也可以根据您的需求按照文档中的内容对页面的样式，布局以及功能项进行深度的定制。

双列瀑布流：默认以双列卡片形式展示语聊房列表。

快速接入

步骤 1: 开通服务

参考 开通服务 文档开通「体验版」或「大规模直播版」套餐。

步骤 2: 代码集成

参考 准备工作 接入 `TUILiveKit` 。

步骤 3: 添加直播列表瀑布流视图

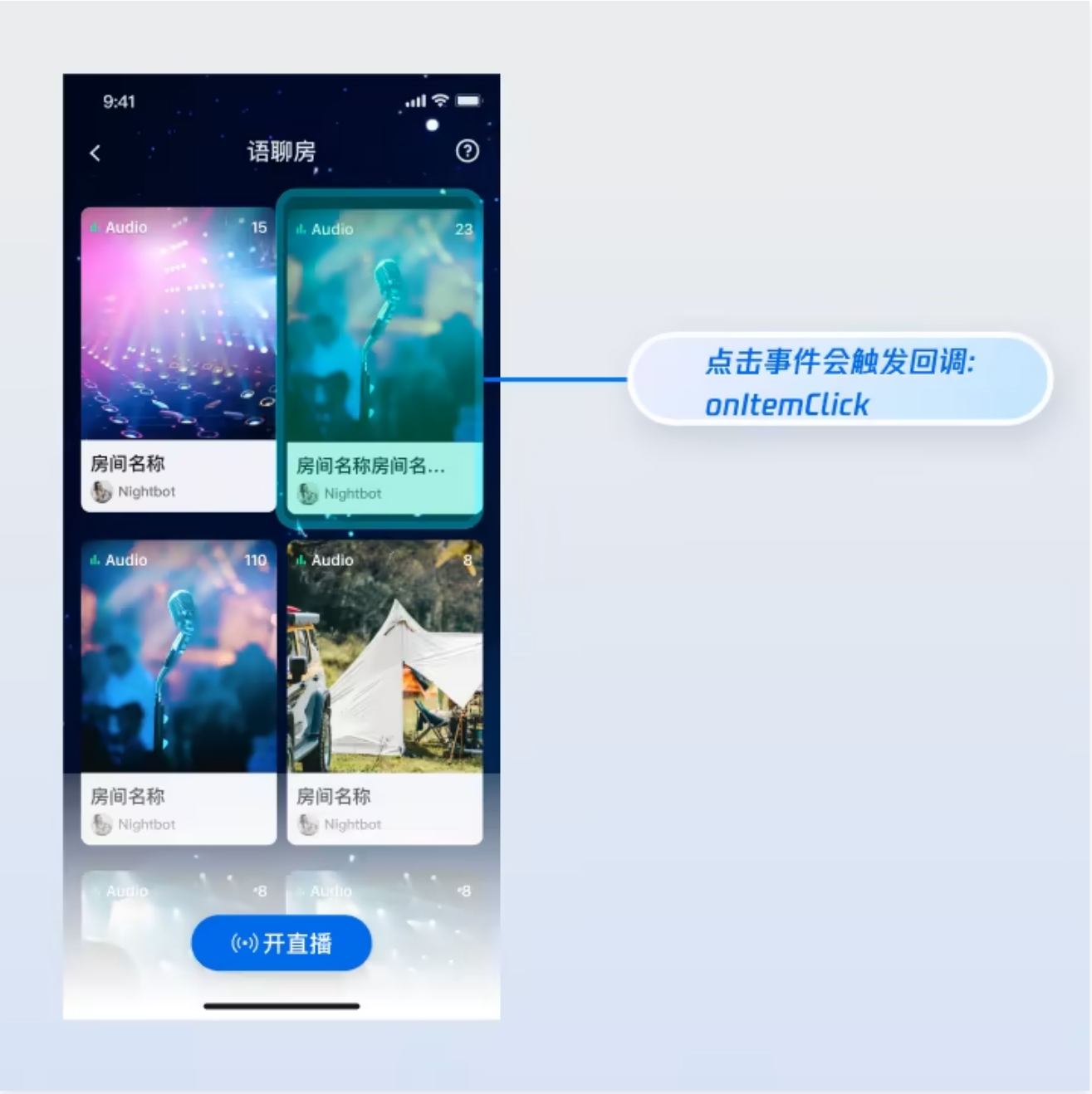
语聊房场景目前仅支持双列瀑布流样式。

```
import android.content.Context
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.trtc.uikit.livekit.features.livelist.LiveListView
import com.trtc.uikit.livekit.features.livelist.Style

class YourActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // 1.创建并初始化瀑布流视图
        val view = createLiveListView(this)
        // 2.将创建的 LiveListView 添加到您的 Activity 或 Fragment 中
        setContentView(view)
    }

    fun createLiveListView(context: Context): LiveListView {
        val liveListView = LiveListView(context)
        // 初始化为双列瀑布流
        liveListView.init(this, Style.DOUBLE_COLUMN)
        return liveListView
    }
}
```

步骤 4: 实现直播列表页到观众观看页的跳转

直播列表会通过 `OnItemClickListener` 回调处理点击事件，您只需在直播列表瀑布流视图中实现 `OnItemClickListener` 来响应用户的点击事件，并在 `onItemClick` 中实现跳转进入观众观看页的功能，观众观看页实现可参考 观众观看 。

交互示例：



代码示例：

```
fun createLiveListView(context: Context): LiveListView {
    val liveListView = LiveListView(context)
    // 初始化为双列瀑布流
    liveListView.init(this, Style.DOUBLE_COLUMN)

    liveListView.setOnItemClickListener { view, liveInfo ->
        // 点击直播列表项时，跳转进入到观众观看页面
        val intent = Intent(context, YourAudienceActivity::class.java)
        intent.putExtra("liveId", liveInfo.roomId)
```

```
            context.startActivity(intent)
        }
        return liveListView
    }
```

自定义您的界面布局

`TUILiveKit` 提供了灵活的接口定制直播列表瀑布流组件，您可以根据业务需求自定义数据源和列表项样式。

自定义数据源

如果您的后台有单独的直播列表数据，可以通过实现 `LiveListDataSource` 接口来自定义数据源，而不使用组件默认的列表数据。

```
// 1.导入依赖
import com.trtc.uikit.livekit.features.livelist.LiveListDataSource;

// 2.通过实现 LiveListDataSource 自定义数据源
val dataSource = object : LiveListDataSource {
    override fun fetchLiveList(param: FetchLiveListParam,callback:
LiveListCallback) {
        // 对接自己的业务后台，按照下面的格式返回数据给UI组件
        val liveInfoList = mutableListOf<TUILiveListManager.LiveInfo>()
        val liveInfo = TUILiveListManager.LiveInfo().apply {
            roomInfo = TUIRoomDefine.RoomInfo().apply {
                roomId = "live_123456"
                name = "live_123456"
            }
        }
        liveInfoList.add(liveInfo)
        val cursor = "aabbccdd"
        callback.onSuccess(cursor, liveInfoList)
    }
}

// 3.初始化时传入自定义的 dataSource
liveListView.init(this, Style.DOUBLE_COLUMN, dataSource = dataSource)
```

自定义挂件

瀑布流列表项默认显示房间封面。如果您需要自定义列表项顶部的 `UI` 元素（例如主播头像、直播标题等），可以通过实现 `LiveListViewAdapter` 接口来完成。

```
// 1.导入依赖
import com.trtc.uikit.livekit.features.livelist.LiveListViewAdapter;

// 2.通过实现 LiveListViewAdapter 自定义挂件
val liveListViewAdapter = object : LiveListViewAdapter {
    override fun createLiveInfoView(liveInfo:
TUILiveListManager.LiveInfo): View {
        // 自定义挂件 view
        val widgetView = YourView(context)
        widgetView.init(liveInfo)
        return widgetView
    }

    override fun updateLiveInfoView(view: View, liveInfo:
TUILiveListManager.LiveInfo) {
        // 更新挂件 view 中绑定的数据
        val widgetView = view as YourView
        widgetView.updateLiveInfoView(liveInfo)
    }
}

// 3.初始化时传入自定义的 liveListViewAdapter
liveListView.init(this, Style.DOUBLE_COLUMN, adapter =
liveListViewAdapter)
```

下一步

恭喜您，现在您已经成功集成了**直播列表**功能。接下来，您可以实现**主播开播、观众观看**等功能，可参考下表：

功能	描述	集成指引
主播开播	实现主播开播语聊房全流程功能，包括开播前的准备和开播后的各种互动。	主播开播
观众观看	实现观众进入主播的语聊房后进行互动，如上麦、收发弹幕等功能。	观众观看

常见问题

集成直播列表功能后页面没有任何直播怎么办?

如果您看到空白页面，需要检查您是否已完成 登录步骤 。为了测试该功能，您可以使用两台设备：一台设备用于开播，另一台设备在直播列表页面，就能拉取到已开播的直播间。

我使用了自定义数据源 (LiveListDataSource)，但列表不显示/不刷新怎么办?

请确保您正确实现了 `LiveListDataSource` 接口。重点检查以下几点：

1. 检查 `fetchLiveList` 方法是否被正确调用。
2. 确保在获取数据后（无论成功或失败）都调用了 `callback.onSuccess` 或 `callback.onFailure` 。
3. 检查您的业务后台接口是否返回了正确的数据格式。

直播列表（iOS）

最近更新时间：2025-11-21 14:15:33

功能预览

本文对 TUILiveKit 中的**语聊房直播列表页面**进行了详细的介绍。您可以在已有项目中直接参考本文档集成我们开发好的直播列表页面，也可以根据您的需求按照文档中的内容对页面的样式，布局以及功能项进行深度的定制。

双列瀑布流：默认以双列卡片形式展示语聊房列表。

快速接入

步骤1：开通服务

参考 开通服务 文档开通「体验版」或「大规模直播版」套餐。

步骤2：代码集成

参考 准备工作 接入 `TUILiveKit` 。

步骤3：添加直播列表瀑布流视图

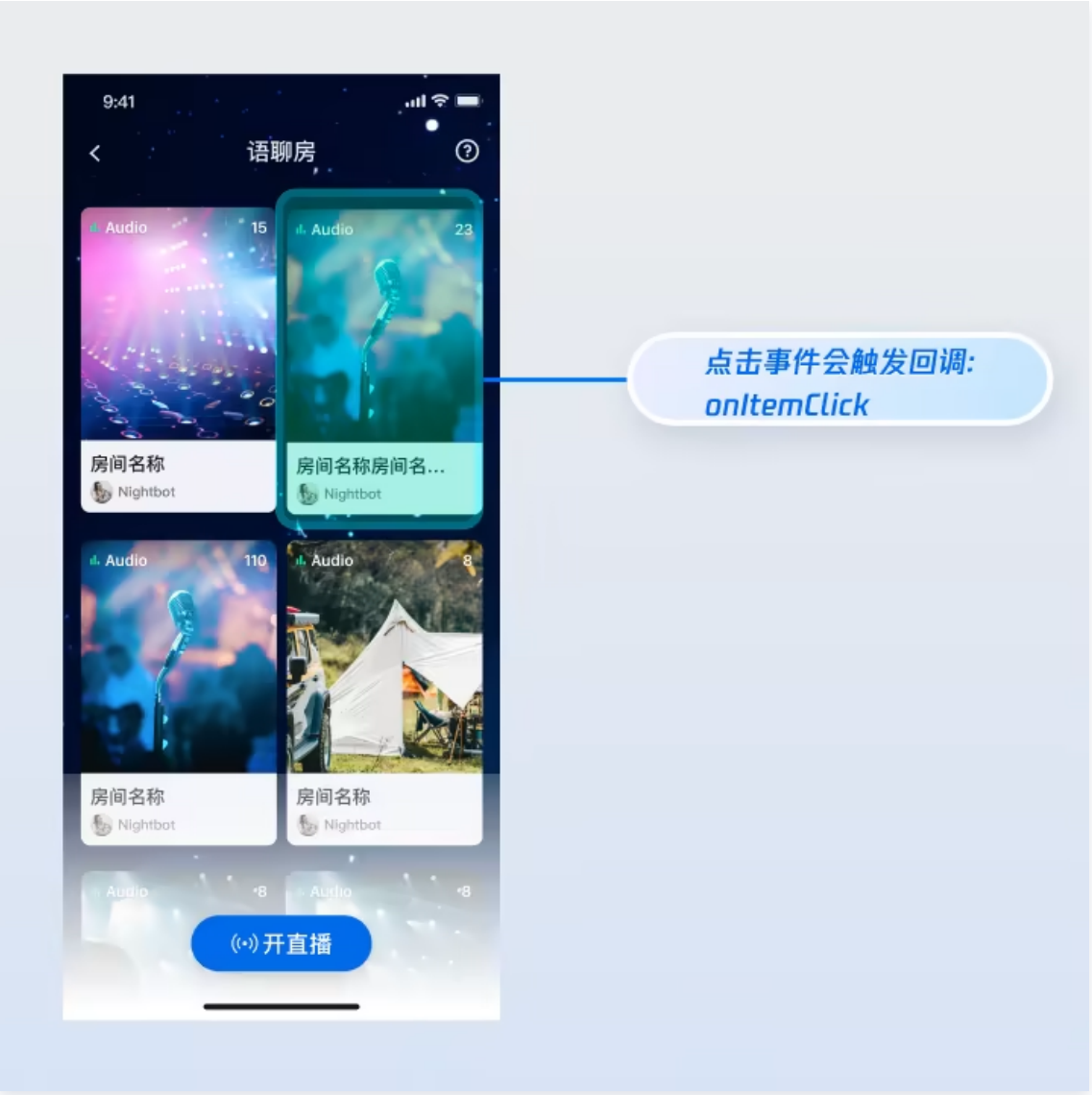
语聊房场景目前仅支持双列瀑布流样式。

```
// 示例： YourLiveListViewController 代表您直播列表瀑布流的视图控制器
class YourLiveListViewController: UIViewController {

    // 1. 声明 liveListView 作为成员变量
    private let liveListView = LiveListView(style: .doubleColumn)

    public override func viewDidLoad() {
        super.viewDidLoad()
        // 2. 将 liveListView 添加到视图上
        view.addSubview(liveListView)
        liveListView.snp.makeConstraints { make in
            make.edges.equalToSuperview()
        }
        // 3. 设置列表的点击事件代理
        liveListView.itemClickDelegate = self
    }
}
```

步骤4：实现直播列表页到观众观看页的跳转

直播列表会通过 `OnItemClickListener` 回调处理点击事件，您只需在直播列表瀑布流视图中实现 `OnItemClickListener` 来响应用户的点击事件，并在 `onItemClick` 中实现跳转进入观众观看页的功能，观众观看页实现可参考 观众观看 。

交互示例：



代码示例：

```
extension YourLiveListViewController: OnItemClickDelegate {

    func onItemClick(liveInfo: LiveInfo, frame: CGRect) {
        // 1. 实例化您的观众观看视图控制器
        let audienceVC = YourAudienceViewController(roomId:
liveInfo.roomId)
        audienceVC.modalPresentationStyle = .fullScreen
        // 2. 跳转到您的观众观看视图控制器
        present(audienceVC, animated: false)
    }

}
```

自定义您的界面布局

`TUILiveKit` 提供了灵活的接口定制直播列表瀑布流组件，您可以根据业务需求自定义数据源和列表项样式。

自定义数据源

如果您的后台有单独的直播列表数据，可以通过实现 `LiveListDataSource` 接口来自定义数据源，而不使用组件默认的列表数据。

```
// 示例：YourLiveListViewController 代表您直播列表瀑布流的视图控制器
class YourLiveListViewController: UIViewController {

    private let liveListView: LiveListView = LiveListView(style:
.doubleColumn)

    public override func viewDidLoad() {
        super.viewDidLoad()
        view.addSubview(liveListView)
        liveListView.snp.makeConstraints { make in
            make.edges.equalToSuperview()
        }
        liveListView.itemClickDelegate = self
        // 1. 设置自定义数据源代理
        liveListView.dataSource = self
    }
}

// 2. 实现自定义数据源代理：LiveListDataSource
extension YourLiveListViewController: LiveListDataSource {

    public func fetchLiveList(cursor: String, onSuccess: @escaping
LiveListBlock, onError: @escaping TUIErrorBlock) {
        // 3. 对接自己的业务后台，按照下面的格式返回数据给UI组件
        var liveInfoList: [LiveInfo] = []
        var liveInfo = LiveInfo()
        liveInfo.roomId = "live_123456"
        liveInfo.name = "live_123456"
        liveInfoList.append(liveInfo)
        let cursor = "aabbccdd"
        onSuccess(cursor, liveInfoList)
    }
```

```
}
```

自定义挂件

瀑布流列表项默认显示房间封面。如果您需要自定义列表项顶部的 `UI` 元素（例如主播头像、直播标题等），可以通过实现 `LiveListViewAdapter` 接口来完成。

```
// 示例：YourLiveListViewController 代表您列表瀑布流的视图控制器
class YourLiveListViewController: UIViewController {

    private let liveListView: LiveListView = LiveListView(style:
.doubleColumn)

    public override func viewDidLoad() {
        super.viewDidLoad()
        view.addSubview(liveListView)
        liveListView.snp.makeConstraints { make in
            make.edges.equalToSuperview()
        }
        liveListView.itemClickDelegate = self
        liveListView.dataSource = self
        // 1. 设置自定义挂件代理
        liveListView.adapter = self
    }
}

// 2. 实现自定义挂件代理
extension YourLiveListViewController: LiveListViewAdapter {
    public func createLiveInfoView(info: LiveInfo) -> UIView {
        // 自定义挂件view
        return YourCustomView(liveInfo: info)
    }

    public func updateLiveInfoView(view: UIView, info: LiveInfo) {
        if let infoView = view as? YourCustomView {
            // 更新挂件view中绑定的数据
            infoView.updateView(liveInfo: info)
        }
    }
```

```
}
```

下一步

恭喜您，现在您已经成功集成了**直播列表**功能。接下来，您可以实现**主播开播**、**观众观看**等功能，可参考下表：

功能	描述	集成指引
主播开播	实现主播开播语聊房全流程功能，包括开播前的准备和开播后的各种互动。	主播开播
观众观看	实现观众进入主播的语聊房后进行互动，如上麦、收发弹幕等功能。	观众观看

常见问题

集成直播列表功能后页面没有任何直播怎么办？

如果您看到空白页面，需要检查您是否已完成 登录步骤 。为了测试该功能，您可以使用两台设备：一台设备用于开播，另一台设备在直播列表页面，就能拉取到已开播的直播间。

我使用了自定义数据源 (LiveListDataSource)，但列表不显示/不刷新怎么办？

请确保您正确实现了 `LiveListDataSource` 接口。重点检查以下几点：

- 检查 `fetchLiveList` 方法是否被正确调用。
- 确保在获取数据后（无论成功或失败）都调用了 `callback.onSuccess` 或 `callback.onFailure` 。
- 检查您的业务后台接口是否返回了正确的数据格式。