直播 SDK

视频直播

腾讯云

文档目录

视频直播
开播方式
主播开播（Android）

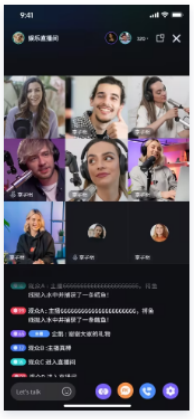
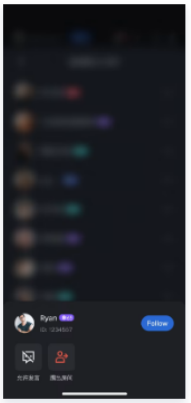
最近更新时间：2026-07-02 11:05:08

`TUILiveKit` 主播开播页为直播场景提供**开箱即用的全功能界面**，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 `UI` 与逻辑实现，即可高效集成直播开播流程。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的开播方式（含完整交互 UI）。若您需通过 `Core SDK` 自行搭建主播端界面，请参考 主播开播 (Core SDK)。

功能概览

- **开播前预览：**支持主播开播前的房间名称、背景、视频预览、美颜调试、音效调试、布局模板等多种个性化配置。
- **连麦互动：**支持直播过程中与观众 或 与其他直播间主播实时互动。
- **观众互动：**支持弹幕、礼物等丰富直播互动形式。
- **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	连麦互动	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. 添加开播前的准备页面

`AnchorPrepareView` 组件已内置了摄像头预览、音效设置、布局设置、以及其他功能设置，您需创建并加载 `AnchorPrepareView` ，具体示例代码如下：

Kotlin

```
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.trtc.uikit.livekit.features.anchorprepare.AnchorPrepareView
import
com.trtc.uikit.livekit.features.anchorprepare.AnchorPrepareViewListener

class AnchorActivity : AppCompatActivity() {
    private val TAG = "AnchorActivity"

    lateinit var anchorPrepareView: AnchorPrepareView
    private val anchorPrepareViewListener = object :
AnchorPrepareViewListener {
        override fun onClickStartButton() {
            // 点击预览界面开始直播按钮触发
        }

        override fun onClickBackButton() {
            // 点击预览界面返回按钮触发
            finishAndRemoveTask()
        }
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // 1.创建并初始化 AnchorPrepareView
        anchorPrepareView = AnchorPrepareView(this)
        anchorPrepareView.init("live_1236666", null)

anchorPrepareView.addAnchorPrepareViewListener(anchorPrepareViewListener
)

        // 2.将 AnchorPrepareView 添加到界面
```

```
        setContentView(anchorPrepareView)
    }
}
```

步骤 4. 添加主播用的推流页面

`AnchorView` 组件已内置了音视频推流、观众连麦、直播互动、直播管理等功能，您只需创建并加载 `AnchorView` ，具体示例代码如下：

> **配合服务端 (RESTFUL API) 开播：**
> 如果您的房间是先通过 服务端 API 创建的，在集成时无需修改任何代码逻辑，只需将示例代码中的 `live ID` 替换为服务端生成的 ID 即可。SDK 将自动接管该房间并开启直播。

Kotlin

```
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.trtc.uikit.livekit.features.anchorprepare.AnchorPrepareView
import
com.trtc.uikit.livekit.features.anchorprepare.AnchorPrepareViewListener
import
com.trtc.uikit.livekit.features.anchorprepare.LiveStreamPrivacyStatus
import com.trtc.uikit.livekit.features.anchorview.AnchorState
import com.trtc.uikit.livekit.features.anchorview.AnchorView
import com.trtc.uikit.livekit.features.anchorview.AnchorViewListener
import com.trtc.uikit.livekit.features.anchorview.RoomBehavior
import io.trtc.tuikit.atomicxcore.api.live.LiveInfo
import io.trtc.tuikit.atomicxcore.api.live.SeatLayoutTemplate

class AnchorActivity : AppCompatActivity() {
    private val TAG = "AnchorActivity"

    lateinit var anchorPrepareView: AnchorPrepareView
    private val anchorPrepareViewListener = object :
AnchorPrepareViewListener {
        override fun onClickStartButton() {
            // 完成从准备页到主播开播页的切换
            initAnchorView()
        }
```

```
        override fun onClickBackButton() {
            finishAndRemoveTask()
        }
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // 1.创建并初始化 AnchorPrepareView
        anchorPrepareView = AnchorPrepareView(this)
        anchorPrepareView.init("live_1236666", null)

anchorPrepareView.addAnchorPrepareViewListener(anchorPrepareViewListener
)

        // 2.将 AnchorPrepareView 添加到界面
        setContentView(anchorPrepareView)
    }

    val anchorViewListener = object : AnchorViewListener {
        override fun onEndLiving(state: AnchorState) {
            finishAndRemoveTask()
        }

        override fun onClickFloatWindow() {
        }
    }

    fun initAnchorView() {
        // 1.创建 AnchorView
        val anchorView = AnchorView(this)

        // 2.初始化 AnchorView，其中 liveInfo 为直播间信息，anchorPrepareView
为您的开播前的准备页面
        val liveInfo = LiveInfo()
        liveInfo.liveID = "live_1236666"
        liveInfo.liveName =
anchorPrepareView.getState()?.roomName?.value ?: ""
        liveInfo.isPublicVisible =
anchorPrepareView.getState()?.liveMode?.value ==
LiveStreamPrivacyStatus.PUBLIC
```

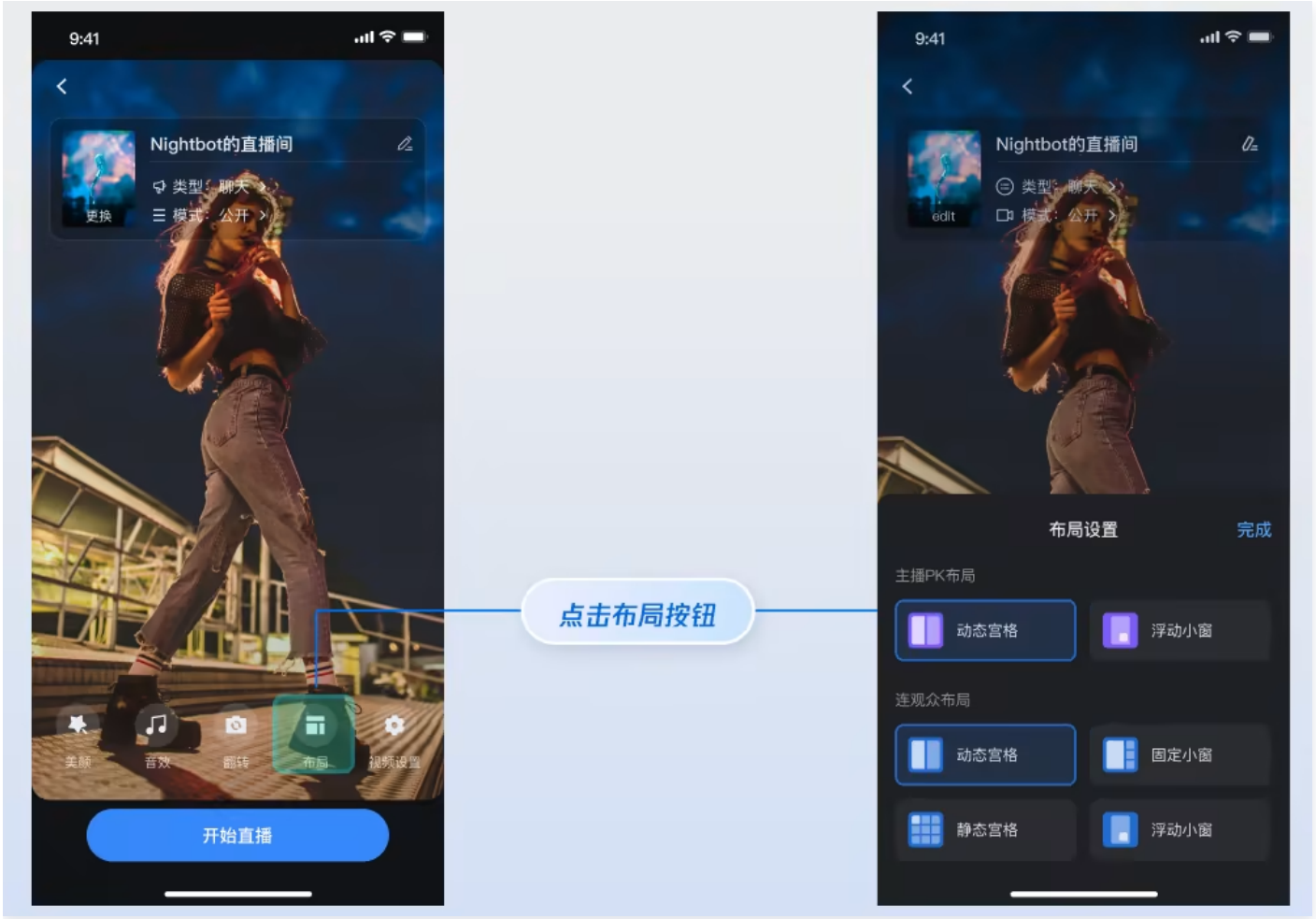
```
        liveInfo.coverURL =
anchorPrepareView.getState()?.coverURL?.value ?: ""
        liveInfo.seatTemplate =
SeatLayoutTemplate.VideoDynamicGrid9Seats
        // 进房行为：
        //     RoomBehavior.CREATE_ROOM: 主播创建房间
        //     RoomBehavior.ENTER_ROOM: 主播进入服务端 rest api 创建的直播间
        anchorView.init(liveInfo, anchorPrepareView.getCoreView(),
RoomBehavior.CREATE_ROOM, null)
        anchorView.addAnchorViewListener(anchorViewListener)

        // 3.将 AnchorView 添加到界面
        setContentView(anchorView)
    }
}
```

自定义您的界面布局

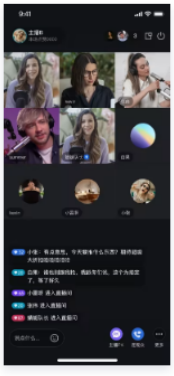
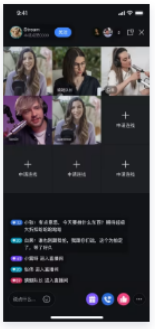
`TUILiveKit` 支持灵活定制开播页与直播页的功能和样式，您可根据业务需求调整布局、隐藏 / 显示功能模块。

直播布局模板选择

`TUILiveKit` 提供 4 种直播布局模板，您可在主播开播页的 UI 交互「布局」入口选择合适样式：



布局模板一览：

布局分类	动态宫格布局	浮动小窗布局	固定宫格布局	固定小窗布局
模板取值	`VideoDynamicGrid9Seats`	`VideoDynamicFloat7Seats`	`VideoFixedGrid9Seats`	`VideoFixedFloat7Seats`
描述	默认布局，可根据连麦人数动态调整宫格大小。	连麦嘉宾以浮动小窗形式显示。	连麦人数固定，每个嘉宾占据一个固定宫格。	连麦人数固定，嘉宾以固定小窗形式显示。
运行效果				

自定义开播页功能区

通过调用 步骤 3 创建的 `prepareView` 的 API，可自定义隐藏开播准备页的操作区或特定功能：

Kotlin

```kotlin
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.trtc.uikit.livekit.features.anchorprepare.AnchorPrepareView

class AnchorActivity : AppCompatActivity() {

    lateinit var anchorPrepareView: AnchorPrepareView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        anchorPrepareView = AnchorPrepareView(this)
        anchorPrepareView.init("live_1235858", null)

        // 1. 自定义功能区 - 例如：隐藏整个功能区
        anchorPrepareView.disableFeatureMenu(true)

        setContentView(anchorPrepareView)
    }
}
```

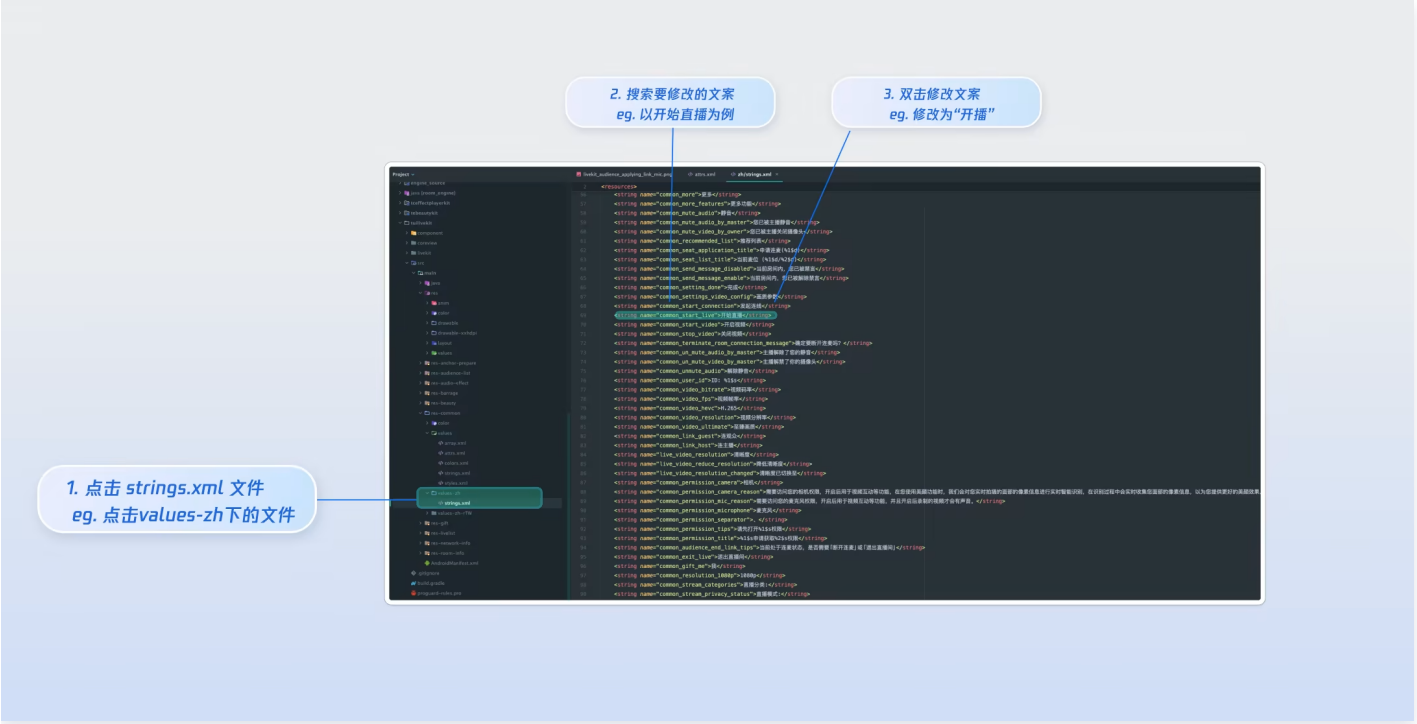
隐藏整个功能区	隐藏美颜按钮	隐藏音效按钮	隐藏摄像头切换按钮
`disableFeatureMenu(true)`	`disableMenuBeautyButton(true)`	`disableMenuAudioEffectButton(true)`	`disableMenuSwitchCameraButton(true)`
			

自定义主播直播页界面

- 参考 主播核心页面 进行主播界面定制，在 `AnchorView` 中修改 `UI` 样式，添加您的业务组件。
- 参考 调整视频直播挂件 进行视频位挂件定制，在 `AnchorView` 中修改视频位上的名称、头像挂件等 `UI` 。

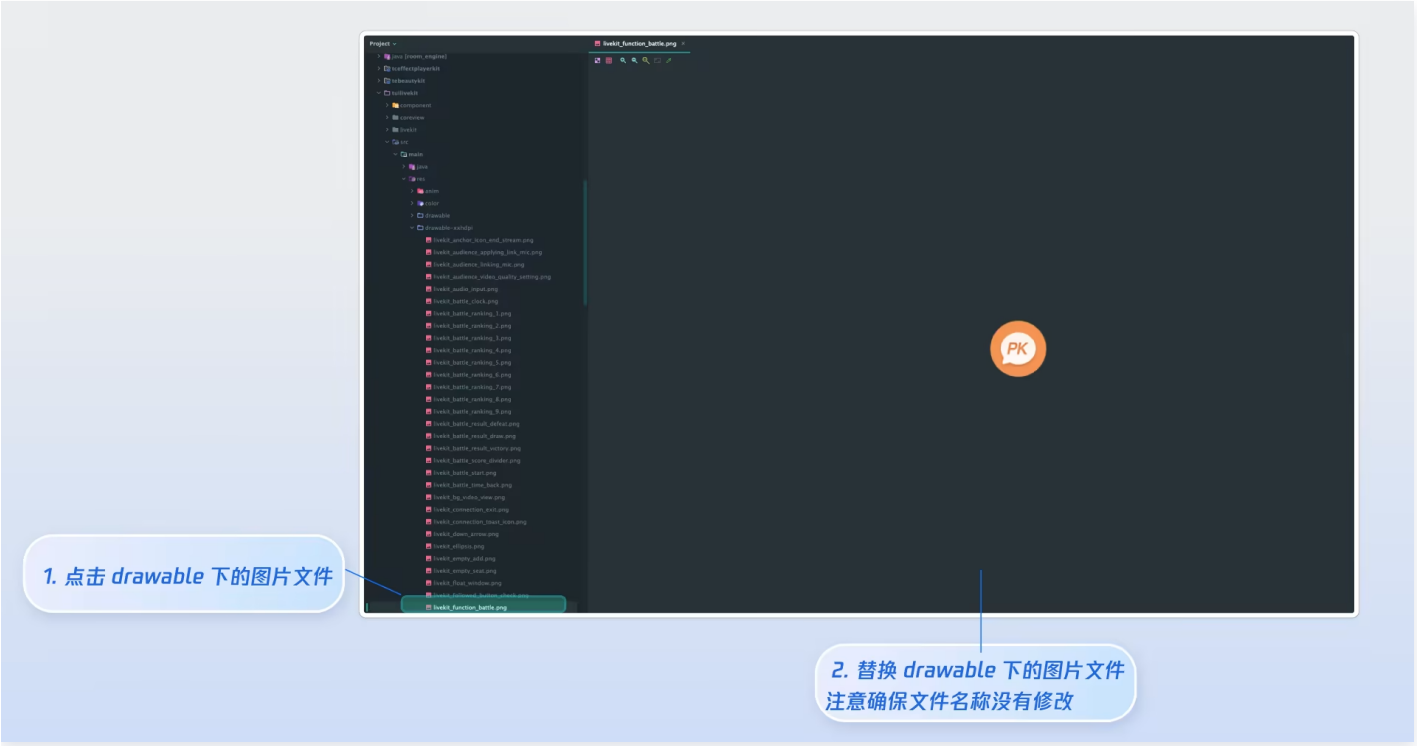
文案定制

`TUILiveKit` 使用 Android 通用的 `XML` 资源文件来管理 UI 所需的文案显示，您可以直接通过 `XML` 文件修改需要调整的文案：



图标定制

`TUILiveKit` 使用 Android 通用的 `drawable` 资源文件夹管理 UI 所需的图片资源，您可以通过替换资源文件快速修改自定义界面所需的图标，替换时请确保新的文件名称与原文件名称一致。



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直播间后观看直播，实现观众连麦 、直播间信息、在线观众、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
礼物系统	实现自定义礼物素材配置、计费系统对接、PK 场景互动等功能。	礼物系统

常见问题

开播后无画面?

请前往**应用信息 > 权限 > 相机**，检查摄像头权限是否开启，可参考下图：

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。

主播开播（iOS）

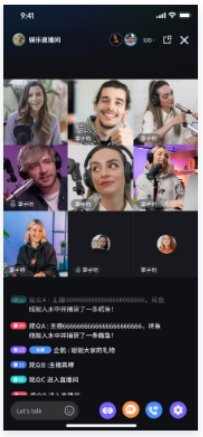
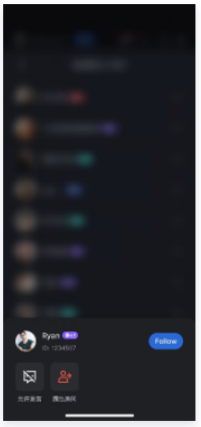
最近更新时间：2026-07-06 15:19:22

TUILiveKit 主播开播页为直播场景提供**开箱即用的全功能界面**，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 UI 与逻辑实现，即可高效集成直播开播流程。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的开播方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建主播端界面，请参考 主播开播 (Core SDK)。

功能概览

- **开播前预览：**支持主播开播前的房间名称、背景、视频预览、美颜调试、音效调试、布局模板等多种个性化配置。
- **连麦互动：**支持直播过程中与观众 或 与其他直播间主播实时互动。
- **观众互动：**支持弹幕、礼物等丰富直播互动形式。
- **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	连麦互动	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. 添加开播前的准备页面

`AnchorPrepareView` 组件已内置了摄像头预览、音效设置、布局设置、以及其它功能设置，您需创建并加载主播开播页视图，具体示例代码如下：

Swift

```
import UIKit
import Snapkit
import TUILiveKit
import AtomicXCore

// YourAnchorPrepareViewController 代表您加载主播开播页的视图控制器
class YourAnchorPrepareViewController: UIViewController {
    private let roomId = "testLiveId"

    // 2. 懒加载 AnchorPrepareView
    private lazy var prepareView: AnchorPrepareView = {
        // 3. AnchorPrepareView 初始化时传入roomId
        let view = AnchorPrepareView(roomId: roomId)
        return view
    }()

    public override func viewDidLoad() {
        super.viewDidLoad()
        // 4. 将 prepareView 添加到视图上
        view.addSubview(prepareView)
        prepareView.snp.makeConstraints { make in
            make.edges.equalToSuperview()
        }
    }
}
```

步骤 4. 添加主播用的推流页面

`AnchorView` 组件已内置了音视频推流、观众连麦、直播互动、直播管理等功能，您只需创建并加载 `AnchorView` ，具体示例代码如下：

swift

```
import UIKit
```

```
import Snapkit
import TUILiveKit
import AtomicXCore

// YourAnchorPrepareViewController 代表您加载主播开播页的视图控制器
class YourAnchorViewController: UIViewController {
    // 核心 view 组件
    private let coreView: LiveCoreView

    // 1. 声明 anchorView 实例
    private let anchorView: AnchorView

    public init(liveInfo: LiveInfo, coreView: LiveCoreView? = nil,
behavior: RoomBehavior = .createRoom) {
        if let coreView = coreView {
            self.coreView = coreView
        } else {
            self.coreView = LiveCoreView(viewType: .pushView)
        }
        // 3. 实例化主播开播页
        self.anchorView = AnchorView(liveInfo: liveInfo, coreView:
self.coreView, behavior: behavior)
        super.init(nibName: nil, bundle: nil)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    public override func viewDidLoad() {
        super.viewDidLoad()
        // 4. 将 anchorView 添加到视图上
        view.addSubview(anchorView)
        anchorView.snp.makeConstraints { make in
            make.edges.equalToSuperview()
        }
    }
}
```

步骤 5. 从准备页面跳转到推流页面

结合 步骤 3 实现 `AnchorPrepareView` 组件的代理事件，完成主播开播页的路由跳转，具体示例代码如下：

> **配合服务端 (RESTFUL API) 开播：**
> 如果您的房间是先通过 服务端 API 创建的，在集成时无需修改任何代码逻辑，只需将示例代码中的 `live ID` 替换为服务端生成的 ID 即可。SDK 将自动接管该房间并开启直播。

swift

```
import UIKit
import TUILiveKit
import AtomicXCore

// 1. 在 AnchorPrepareView 初始化时设置代理
class YourAnchorPrepareViewController: UIViewController {
    private lazy var prepareView: AnchorPrepareView = {
        let view = AnchorPrepareView(roomId: roomId)
        // 设置代理
        view.delegate = self
        return view
    }()
}

// 2. 实现 AnchorPrepareViewDelegate 回调代理
extension YourAnchorPrepareViewController : AnchorPrepareViewDelegate {
    // 响应开播按钮点击事件
    //   - state: PrepareState 封装了主播开播页的摄像头、音效等功能设置，您只需按以下示例代码设置即可
    public func onClickStartButton(state: PrepareState) {
        // 4. 跳转到主播开播页
        // 4.1. 初始化直播信息
        var liveInfo = LiveInfo(seatTemplate: .videoDynamicGrid9Seats)
        // 如果您使用了服务端 API 预创房，请在此处填入服务端返回的 RoomId。
        liveInfo.liveID = roomId
        liveInfo.liveName = state.roomName
        liveInfo.coverURL = state.coverUrl
        liveInfo.isPublicVisible = state.privacyMode == .public
        liveInfo.backgroundURL = state.coverUrl
```

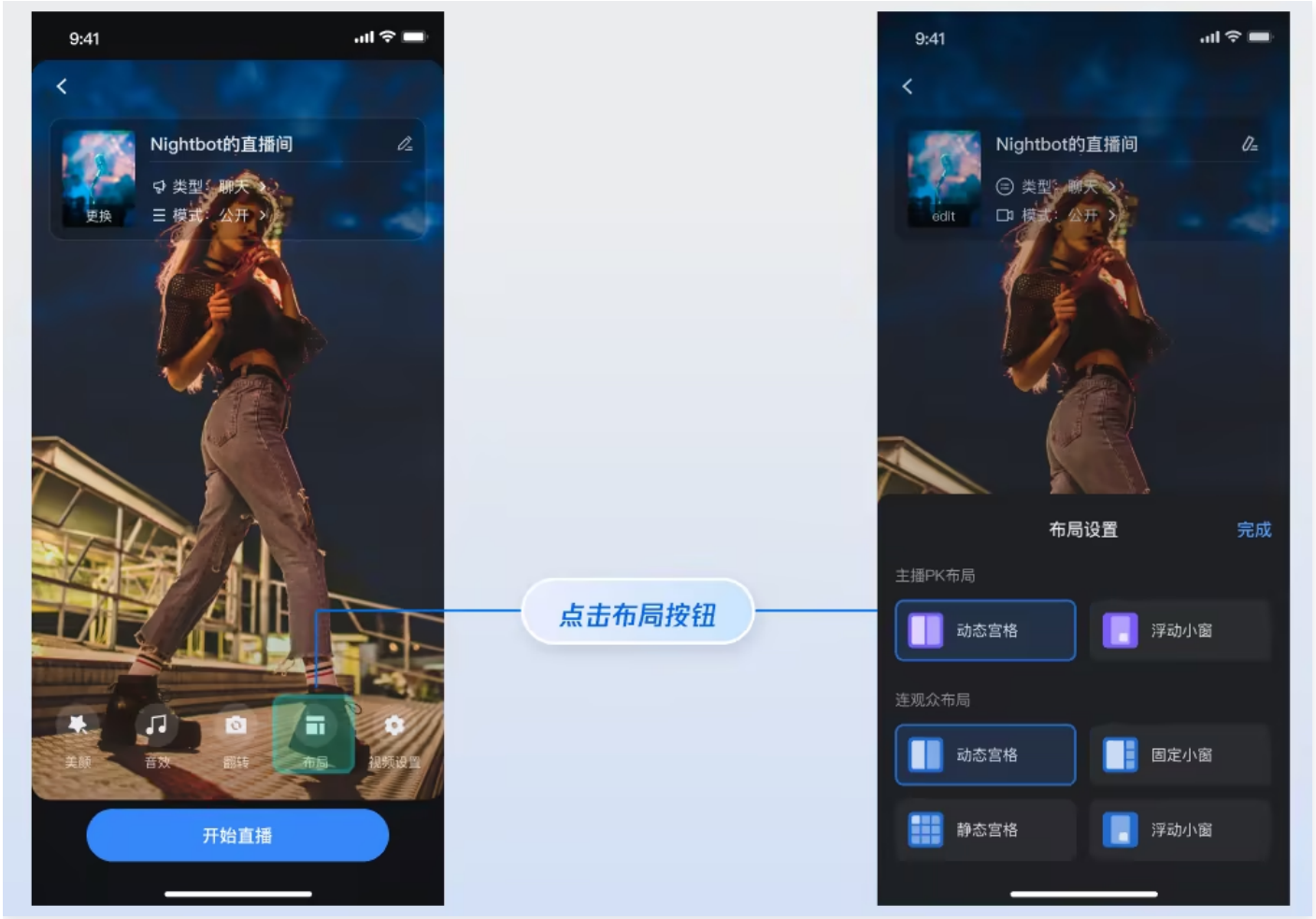
```
        // 4.2. 实例化您的直播开播视图控制器
        let anchorVC = YourAnchorViewController(liveInfo: liveInfo,
coreView: prepareView.getCoreView())
        anchorVC.modalPresentationStyle = .fullScreen
        // 4.3. 跳转到您的直播开播视图控制器
        present(anchorVC, animated: false)
    }

    // 响应返回按钮点击事件
    public func onClickBackButton() {
        if let nav = navigationController {
            nav.popViewController(animated: true)
        } else {
            dismiss(animated: true)
        }
    }
}
```

自定义您的界面布局

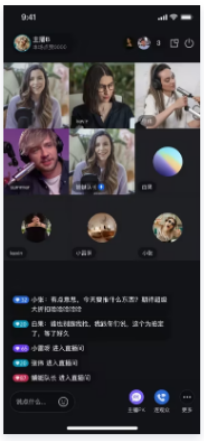
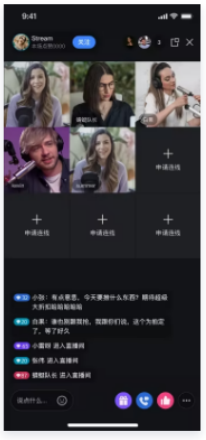
`TUILiveKit` 支持灵活定制开播页与直播页的功能和样式，您可根据业务需求调整布局、隐藏 / 显示功能模块。

直播布局模板选择

`TUILiveKit` 提供 4 种直播布局模板，您可在主播开播页的 UI 交互「布局」入口选择合适样式：



布局模板一览：

布局	动态宫格布局	浮动小窗布局	固定宫格布局	固定小窗布局
模板取值	`.videoDynamicGrid9Seats`	`.videoDynamicFloat7Seats`	`.videoFixedGrid9Seats`	`.videoFixedFloat7Seats`
描述	默认布局，可根据连麦人数动态调整宫格大小。	连麦嘉宾以浮动小窗形式显示。	连麦人数固定，每个嘉宾占据一个固定宫格。	连麦人数固定，嘉宾以固定小窗形式显示。
预览				

自定义开播页功能区

通过调用 步骤 3 创建的 `prepareView` 的 API，可自定义隐藏开播准备页的操作区或特定功能：

swift

```swift
import UIKit
import Snapkit
import TUILiveKit
import AtomicXCore

// YourAnchorPrepareViewController 代表您加载主播开播页的视图控制器
class YourAnchorPrepareViewController: UIViewController {
    private let roomId = "testLiveId"
    private lazy var prepareView: AnchorPrepareView = {
        let view = AnchorPrepareView(roomId: roomId)
        view.delegate = self
        return view
    }()

    public override func viewDidLoad() {
        super.viewDidLoad()
        // 1. 将 prepareView 添加到视图上
        view.addSubview(prepareView)
        prepareView.snp.makeConstraints { make in
            make.edges.equalToSuperview()
        }
        // 2. 自定义功能区 - 示例：隐藏美颜功能
        prepareView.disableMenuBeauty(true)
    }
}
```

隐藏整个功能区	隐藏美颜按钮	隐藏音效按钮	隐藏摄像头切换按钮
`disableFeatureMenu(true)`	`disableMenuBeauty(true)`	`disableMenuAudioEffect(true)`	`disableMenuSwitchCamera(true)`
			

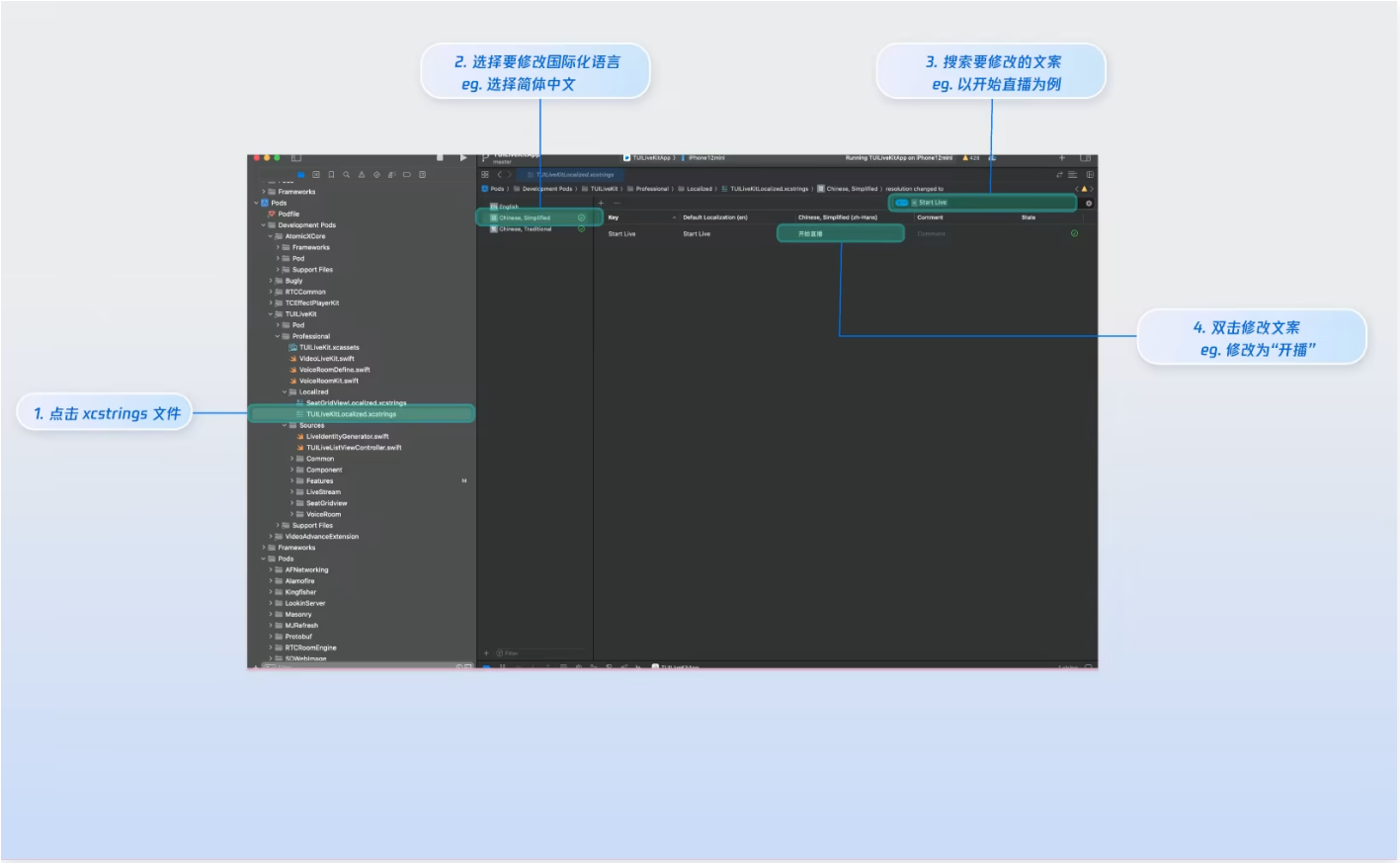
自定义主播直播页界面

- 参考 主播核心页面 进行主播界面定制，在 `AnchorView` 中修改 UI 样式，添加您的业务组件。
- 参考 调整视频直播挂件 进行视频位挂件定制，在 `AnchorView` 中修改视频位上的名称、头像挂件等 UI。

文案定制

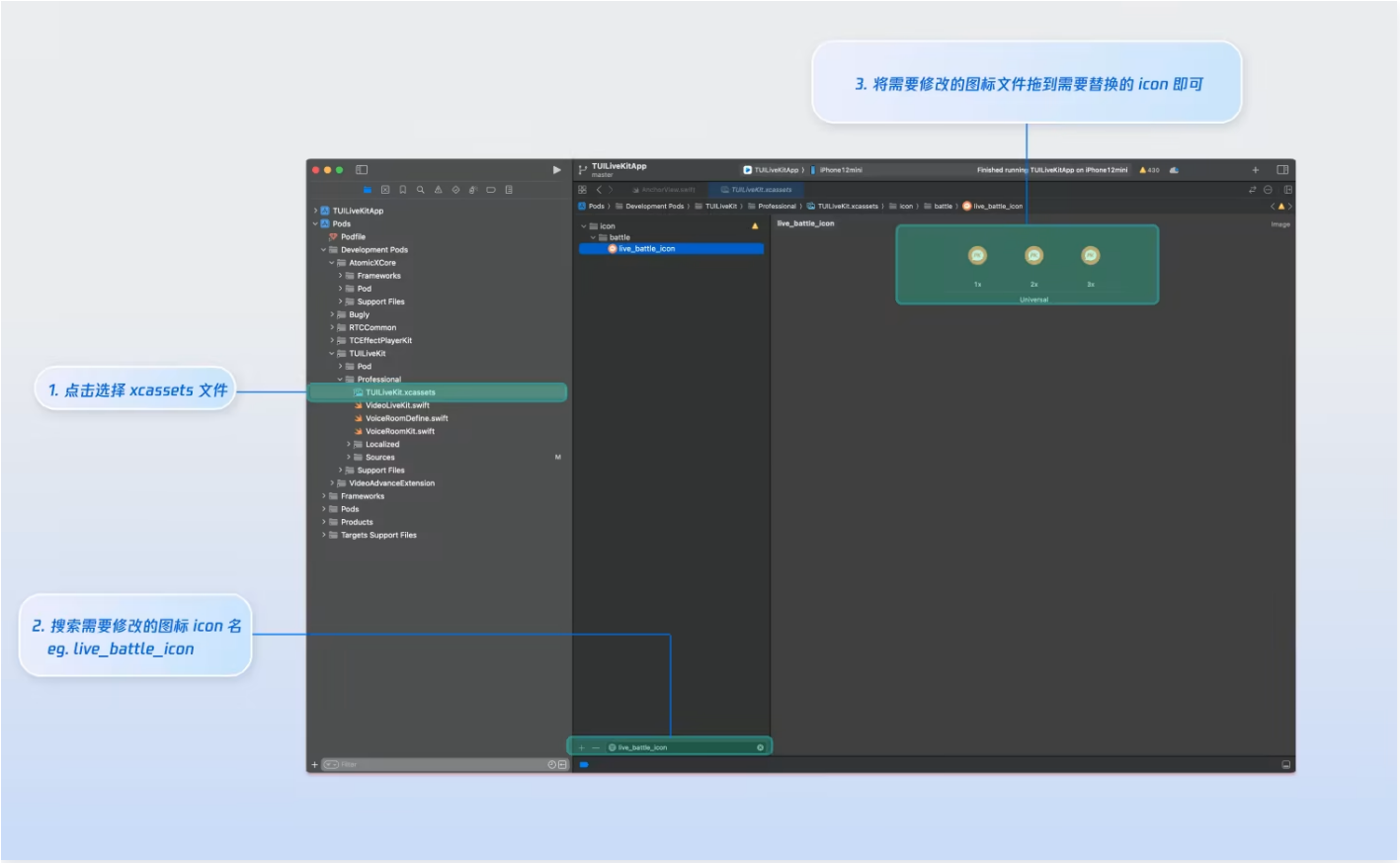
`TUILiveKit` 使用更为方便的 Apple Strings Catalog 工具来管理 UI 所需的文案显示，您可以很直观的通过 Xcode 视图化管理工具修改需要调整的文案：

> **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。



图标定制

`TUILiveKit` 使用 `TUILiveKit.xcassets` 管理 UI 所需的图片资源，您可以借助 Xcode 图形化工具快速修改自定义界面所需的图标。



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**、**礼物系统**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直播间后观看直播，实现观众连麦 、直播间信息、在线观众、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
礼物系统	实现自定义礼物素材配置、计费系统对接、PK 场景互动等功能。	礼物系统

常见问题

开播后无画面?

请前往手机**设置** > **App** > **相机**，检查摄像头权限是否开启，可参考下图：

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。

主播开播（Web Vue3 桌面浏览器）

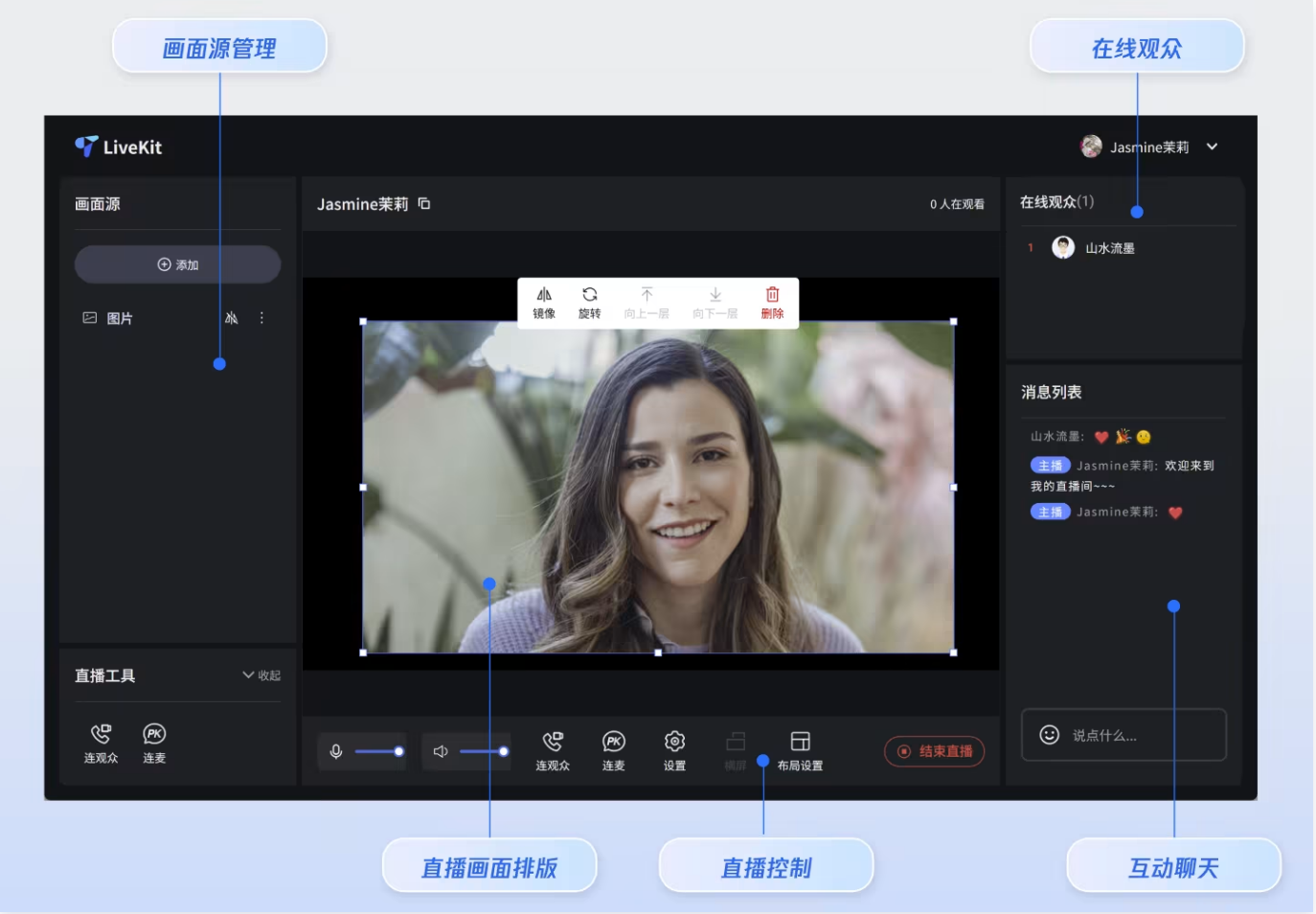
最近更新时间：2026-04-20 17:34:02

本文将介绍如何接入 TUILiveKit Demo 的**主播开播页**，指导您将我们提供的开播页面集成到您的项目中，同时介绍如何对直播页面的样式、布局、以及功能进行自定义修改。

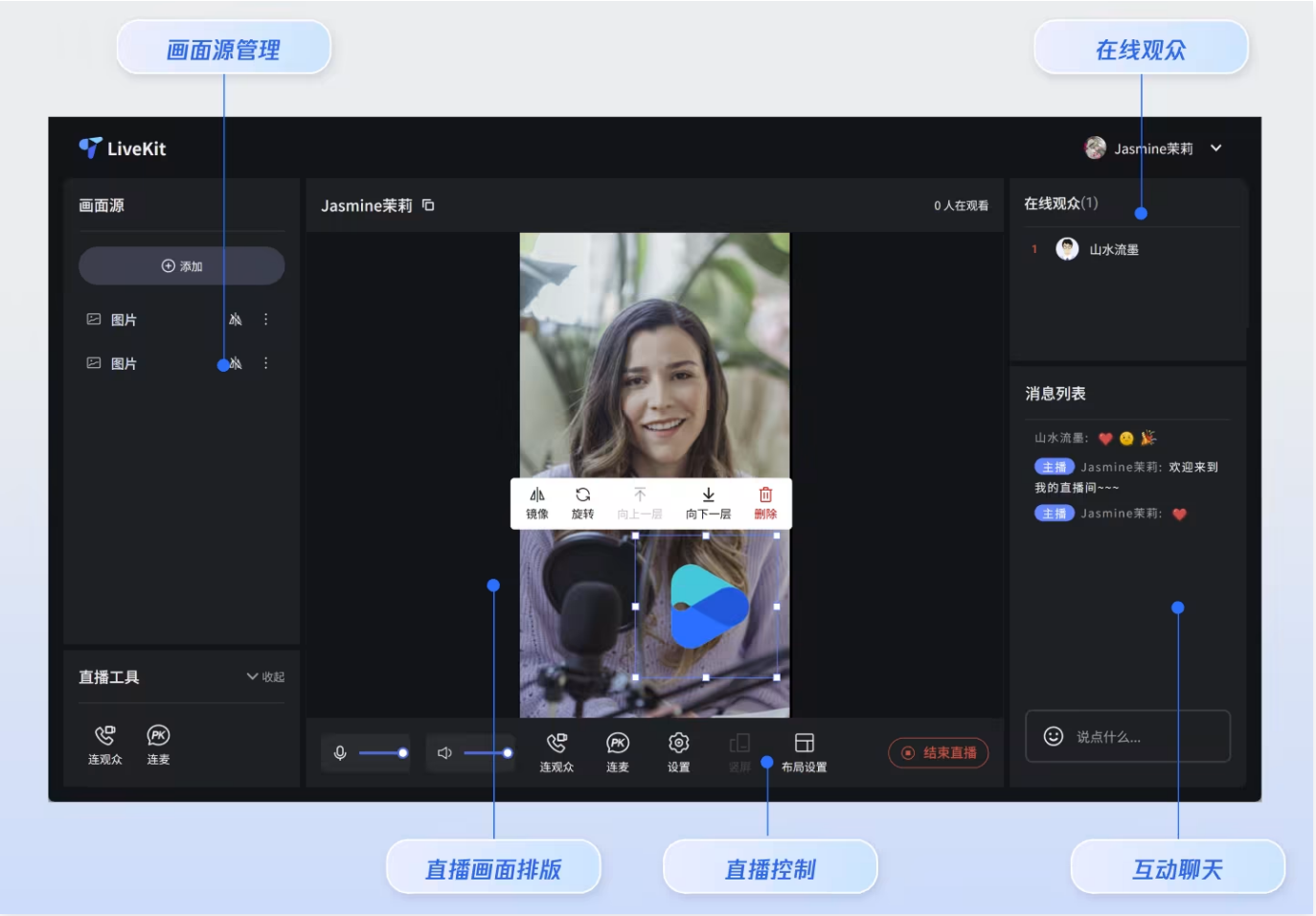
> **集成提示：**
> 本文档介绍基于 **UI 组件** 的开播方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建主播端界面，请参考 主播开播 (Core SDK)。

功能展示

- **横屏开播**



- **竖屏开播**



功能概览

- **画面源管理**：支持添加多个摄像头、屏幕分享、窗口分享和本地图片。
- **直播画面排版**：支持对画面源拖拽、缩放，支持通过快捷菜单对画面源设置镜像、旋转和层级调整。
- **观众连麦**：支持房间内观众连麦，内置多种连麦布局。
- **主播 PK**：支持主播跨房连线、PK。
- **互动聊天**：支持收发文字和表情消息，聊天互动。
- **观众管理**：支持对观众禁言、踢出直播间等操作。

快速接入

步骤1：环境配置及开通服务

在进行快速接入之前，请参考 准备工作 集成组件并实现登录。

步骤2：安装依赖

npm

```
npm install tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
--save
```

pnpm

```
pnpm add tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
```

yarn

```
yarn add tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
```

步骤3：主播开播页面接入

在您的项目下创建 `live-pusher.vue` 文件，通过复制如下代码至您新建的 `live-pusher.vue` 文件中集成完整的**主播开播页面**。

> **注意：**
> 您可以直接复制如下代码至您的工程中集成示例工程，也可以访问 开播页面 地址，查看更加详细的源码内容。

```
<template>
  <UIKitProvider language="zh-CN" theme="dark">
    <div class="custom-live-pusher">
      <!-- 主要内容区 -->
      <div class="main-content">
        <!-- 左侧：视频源和工具 -->
        <div class="left-panel">
          <div class="scene-section">
            <div class="panel-title">画面源</div>
            <LiveScenePanel />
          </div>
        </div>

        <!-- 中央：直播画面 -->
        <div class="center-panel">
          <div class="stream-section">
            <div class="stream-header">
```

```
              <div class="stream-title">
                <span>{{ liveName }}</span>
              </div>
              <div class="stream-audience">{{ audienceCount }} 人在观看
</div>
            </div>
            <StreamMixer />
          </div>
          <div class="live-controls">
            <div class="bottom-panel">
              <!-- <MediaDeviceSetting />// 媒体设置能力请参考本文高级功能集成
部分
              <CoGuestButton />           // 连麦观众能力请参考本文高级功能集成
部分
              <OrientationSwitch />       // 布局设置请参考本文高级功能集成部分
              <LayoutSwitch />            // 横竖屏推流能力请参考本文高级功能集
成部分 -->
            </div>
            <TUIButton type="primary" v-if="!currentLive?.liveId"
@click="handleStartLive">开始直播</TUIButton>
            <TUIButton color="red" v-else @click="handleStopLive">停止直播
</TUIButton>
          </div>
        </div>

        <!-- 右侧：观众互动 -->
        <div class="right-panel">
          <div class="audience-section">
            <div class="panel-title">在线观众({{ audienceCount }})</div>
            <LiveAudienceList />
          </div>
          <div class="message-section">
            <div class="panel-title">消息列表</div>
            <BarrageList />
            <BarrageInput />
          </div>
        </div>

      </div>
    </div>
```

```
  </UIKitProvider>
</template>

<script setup lang="ts">
import { onMounted } from 'vue';
import { UIKitProvider, TUIButton } from '@tencentcloud/uikit-base-
component-vue3';
import { LiveScenePanel, StreamMixer, LiveAudienceList, BarrageList,
BarrageInput, useLiveListState, useLiveAudienceState, useLoginState }
from 'tuikit-atomicx-vue3';

const { login, setSelfInfo } = useLoginState();
const { createLive, currentLive, endLive } = useLiveListState();
const { audienceCount } = useLiveAudienceState();
const liveName = '我的直播间';

const handleStartLive = async () => {
  // 创建直播间
  await createLive({
    liveId: 'my-live-room',        // 直播间 ID
    liveName: liveName,            // 直播间名称
    coverUrl: '',                  // 直播间封面 URL
    isPublicVisible: true,         // 是否公开可见
    seatLayoutTemplateId: 600,     // 麦位布局模板 ID
  });
  // 设置个人信息
  await setSelfInfo({
    userName: '我的名字/昵称',     // 用户名
    avatarUrl: '',                 // 头像 URL 地址
  });
};

const handleStopLive = async () => {
  await endLive();
};

async function initLogin() {
  try {
    await login({
      sdkAppId: 0,         // SDKAppID, 可以参考步骤 1 获取
```

```
      userId: '',          // UserID, 可以参考步骤 1 获取
      userSig: '',         // userSig, 可以参考步骤 1 获取
    });
  } catch (error) {
    console.error('登录失败:', error);
  }
}

onMounted(async () => {
  await initLogin();
});

</script>

<style>html,body,#app{height:100%;width:100%;margin:0;padding:0}</style>
<style scoped>.custom-live-pusher,.left-panel{flex-
direction:column;display:flex}.custom-live-pusher,.live-
title{color:var(--text-color-primary)}.live-controls,.tools-
section,.top-controls{backdrop-filter:blur(10px)}*{box-sizing:border-
box;margin:0;padding:0}:global(::before){box-sizing:border-
box;margin:0;padding:0}.layout-label,.template-options{margin-
bottom:16px}:global(body){line-height:1.6;color:var(--text-color-
primary);background:var(--bg-color-default)}.custom-live-
pusher{height:100vh;width:100vw;background:linear-gradient(135deg,var(--
bg-color-default) 0,var(--bg-color-function) 100%);overflow:hidden}.top-
controls{display:flex;justify-content:space-between;align-
items:center;padding:12px 20px;background:var(--bg-color-
operate);border-bottom:1px solid var(--stroke-color-primary);z-
index:100;min-height:60px}.live-title{font-size:18px;font-
weight:600;text-shadow:0 2px 4px var(--shadow-color)}.audience-
count{font-size:14px;color:var(--text-color-error);background:var(--
uikit-color-red-1);padding:6px 12px;border-radius:20px;border:1px solid
var(--uikit-color-red-3)}.live-controls,.tools-section{background:var(--
bg-color-operate)}.main-content{display:flex;flex:1;height:calc(100vh -
60px);gap:16px;padding:16px;overflow:hidden}.left-
panel{width:320px;gap:16px;flex-shrink:0}.panel-title{font-
size:16px;font-weight:600;color:var(--text-color-primary);margin-
bottom:12px;text-align:left}.audience-section,.message-section,.scene-
section{display:flex;flex-direction:column;background:var(--bg-color-
operate);border-radius:12px;border:1px solid var(--stroke-color-
```

```
primary);padding:16px;overflow:hidden}.scene-section{flex:1;min-
height:0}.message-section{flex:1;min-height:0}.message-section
:deep(input),.message-section :deep(textarea){text-align:left}.tools-
section{border-radius:12px;padding:16px;border:1px solid var(--stroke-
color-primary)}.center-panel{display:flex;flex-
direction:column;flex:1;gap:16px;min-width:0}.stream-
section{display:flex;flex-direction:column;flex:1;min-
height:0;background:var(--bg-color-operate);border-
radius:12px;border:1px solid var(--stroke-color-
primary);overflow:hidden}.stream-header{display:flex;justify-
content:space-between;align-items:center;padding:22px;border-bottom:1px
solid var(--stroke-color-primary)}.stream-title{display:flex;align-
items:center;gap:8px;font-size:18px;font-weight:500;color:var(--text-
color-primary)}.stream-audience{font-size:14px;color:var(--text-color-
primary)}.live-controls{display:flex;justify-content:space-
between;align-items:center;padding:16px;border-radius:12px;border:1px
solid var(--stroke-color-primary);gap:16px}.live-controls
button{padding:18px 20px}.right-panel{display:flex;flex-
direction:column;width:320px;gap:16px;flex-shrink:0}.center-panel>.live-
controls,.left-panel>*,.right-panel>*{background:var(--bg-color-
operate);border-radius:12px;border:1px solid var(--stroke-color-
primary);overflow:hidden}.left-panel>*{padding:16px}.left-panel>.panel-
title,.scene-section>.panel-title,.audience-section>.panel-
title,.message-section>.panel-
title{padding:0;background:transparent;border:none;border-
radius:0}.custom-icon-container,.device-setting{padding:8px
12px;background:var(--bg-color-function)}.stream-section :deep(>*:last-
child){flex:1;min-height:300px}.bottom-panel,.device-setting{align-
items:center;display:flex}.custom-icon-container:hover,.option-
card:hover{border-color:var(--stroke-color-primary);background:var(--
list-color-hover)}.bottom-panel{gap:16px;flex:1}.device-
setting{gap:8px;border-radius:8px;border:1px solid var(--stroke-color-
secondary)}.device-icon{cursor:pointer;color:var(--text-color-
primary);transition:color .2s}.device-icon:hover{color:var(--text-color-
link)}.device-slider{width:80px}.custom-icon-
container{display:flex;align-items:center;gap:6px;border-
radius:8px;border:1px solid var(--stroke-color-
secondary);cursor:pointer;transition:.2s;position:relative}.custom-icon-
container.disabled{opacity:.5;cursor:not-allowed}.custom-icon-
container.disabled:hover{background:var(--bg-color-function);border-
```

```
color:var(--stroke-color-secondary)}.custom-
icon{width:16px;height:16px;display:inline-block;background-
size:contain;background-repeat:no-repeat;background-
position:center}.custom-text{font-size:12px;color:var(--text-color-
secondary);white-space:nowrap}.unread-
count{position:absolute;top:-4px;right:-4px;background:var(--text-color-
error);color:var(--text-color-button);border-radius:10px;padding:2px
6px;font-size:10px;font-weight:600;min-width:16px;text-
align:center;line-height:1}.layout-label,.option-info h4{color:var(--
text-color-primary)}.layout-dialog{max-width:600px}.layout-label{font-
size:16px;font-weight:600}.options-grid{display:grid;grid-template-
columns:repeat(auto-fit,minmax(120px,1fr));gap:12px}.option-
card{padding:16px;background:var(--bg-color-function);border:2px solid
var(--stroke-color-secondary);border-
radius:8px;cursor:pointer;transition:.2s;text-align:center}.option-
card.active{border-color:var(--text-color-link);background:var(--bg-
color-operate)}.option-info h4{margin:8px 0 0;font-size:12px}.option-
icon{width:32px;height:32px;margin:0 auto;color:var(--text-color-
secondary)}.co-guest-dialog{max-width:500px}.co-guest-panel{min-
height:300px}</style>
```

核心 API 参数说明

createLive

创建一个新的直播间，并设置直播间的基础信息。

参数	类型	必填	说明
liveId	String	是	直播间 ID，需保证全局唯一。
liveName	String	是	直播间名称，用于界面展示。
coverUrl	String	否	直播间封面图片的 URL 地址。
isPublicVisible	Boolean	否	是否公开可见。默认为 true，表示公开。
seatLayoutTemplateId	Number	否	麦位布局模板 ID。600 代表竖屏动态宫格布局，更多模板 ID 请参考 高级功能 章节。

setSelfInfo

设置用户信息。

参数	类型	必填	说明
userName	String	是	用户昵称，将在直播间及聊天区域显示。
avatarUrl	String	否	用户头像的 URL 地址。

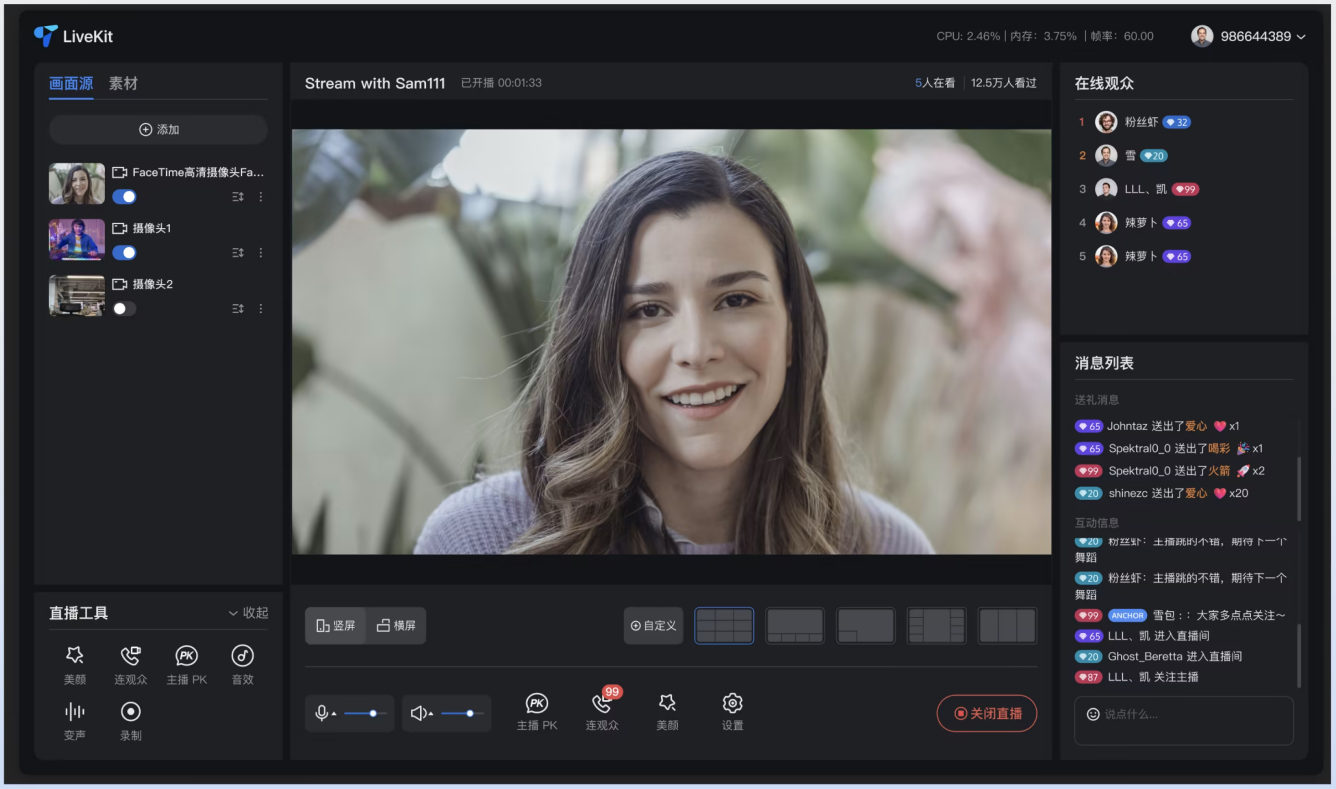
步骤4：开启直播

开启您的第一场直播。

```
npm run dev
```

说明：

上述命令执行成功后，请在浏览器地址栏输入本地访问地址（您可以根据项目参考，例如 `http://localhost:5173/live-pusher`，具体端口号可能因您的项目配置而有所不同），即可看到开播页面。



步骤5：观看直播

- **方式一：（推荐）**

  通过我们提供的 在线观看网站，立刻观看您开启的直播。

- **方式二：**

  集成我们提供的 观众观看（Web 桌面浏览器）或者 观众观看（H5 移动端浏览器） 页面，观看直播。

> ⚠ **重要提示：**
> 请使用不同的**用户 ID** 登录观看： 开播端和观看端必须使用**不同**的用户 ID，否则后登录的设备会强制先登录的设备下线（即“被踢下线”）。

高级功能集成

> ⚠ **注意：**
> 以下高级功能均是基于 步骤 3 中创建的 `live-pusher.vue` 示例文件进行的扩展。请根据您的业务需求，将下方的代码片段集成到 `live-pusher.vue` 的对应位置（ `template` 区域或 `script` 逻辑中），以启用相应的能力。

支持媒体设置能力

若您需要支持媒体设置能力，包括设置扬声器、麦克风音量大小等内容，请参考如下代码示例复制到 `live-pusher.vue` 文件中即可。

```
<!-- MediaDeviceSetting 媒体设备设置 -->
<template>
  <div class="media-device-container">
    <!-- MicVolumeSetting 麦克风设置 -->
    <div class="device-setting">
      <TUIIcon class="device-icon" :icon="microphoneIsOn ?
IconAudioOpen : IconAudioClose" :size="20"
@click="switchMicrophoneStatus" />
      <TUISlider v-model="microphoneVolume" class="device-slider"
:min="0" :max="100" :disabled="!microphoneIsOn"
@change="handleMicrophoneVolumeChange" />
    </div>
    <!-- SpeakerVolumeSetting 扬声器设置 -->
    <div class="device-setting">
      <TUIIcon class="device-icon" :icon="speakerIsOn ?
IconSpeakerOn : IconSpeakerOff" :size="20"
@click="switchSpeaker(!speakerIsOn)" />
      <TUISlider v-model="speakerVolume" class="device-slider"
:min="0" :max="100" :disabled="!speakerIsOn"
@change="handleSpeakerVolumeChange" />
    </div>
  </div>
```

```
</template>

<script lang="ts" setup>
import { ref, watch } from 'vue';
import { TUIIcon, TUISlider, IconSpeakerOn, IconSpeakerOff,
IconAudioOpen, IconAudioClose } from '@tencentcloud/uikit-base-
component-vue3';
import { useDeviceState } from 'tuikit-atomicx-vue3';

const {
  captureVolume,
  setCaptureVolume,
  openLocalMicrophone,
  closeLocalMicrophone,
} = useDeviceState();
const { outputVolume, setOutputVolume } = useDeviceState();
const microphoneVolume = ref(captureVolume.value);
const speakerVolume = ref(outputVolume.value);
const microphoneIsOn = ref(true);
const speakerIsOn = ref(true);
const templateSpeakerVolume = ref(outputVolume.value);
const templateMicrophoneVolume = ref(captureVolume.value);

const handleMicrophoneVolumeChange = (value: number) => {
  if (value !== captureVolume.value) {
    setCaptureVolume(value);
  }
};

const switchMicrophoneStatus = () => {
  microphoneIsOn.value = !microphoneIsOn.value;
  if (!microphoneIsOn.value) {
    templateMicrophoneVolume.value = captureVolume.value;
    closeLocalMicrophone();
  } else {
    openLocalMicrophone();
    setCaptureVolume(templateMicrophoneVolume.value);
  }
};
```

```
const switchSpeaker = (open: boolean) => {
  speakerIsOn.value = open;
  if (!open) {
    templateSpeakerVolume.value = outputVolume.value;
    setOutputVolume(0);
  } else {
    setOutputVolume(templateSpeakerVolume.value);
  }
};

const handleSpeakerVolumeChange = (value: number) => {
  if (value !== outputVolume.value) {
    setOutputVolume(value);
  }
};

watch(captureVolume, (newVal) => {
  microphoneVolume.value = newVal;
});

watch(outputVolume, (newVal) => {
  speakerVolume.value = newVal;
});
</script>

<style scoped>.media-device-container{display:flex;align-
items:center;gap:12px}.device-setting{display:flex;align-
items:center;gap:8px;padding:12px 16px;background:var(--bg-color-
function);border-radius:8px;border:1px solid var(--stroke-color-
secondary)}.device-slider{width:100px}.device-
icon{cursor:pointer;color:var(--text-color-primary);transition:color
.2s}.device-icon:hover{color:var(--text-color-link)}</style>
```

支持横竖屏推流能力

若您需要支持横竖屏推流能力，包括设置切换横竖屏等内容，请参考如下代码示例复制到 `live-pusher.vue` 文件中即可。

```
<!-- LayoutSwitch 横竖屏推流设置 -->
<template>
  <div
    class="custom-icon-container"
    :class="{ disabled: currentLive?.liveId }"
    @click="handleOrientationSwitch"
  >
    <IconHorizontalMode
      v-if="currentOrientation === LiveOrientation.Landscape"
      class="custom-icon"
    />
    <IconPortrait
      v-else
      class="custom-icon"
    />
    <span class="custom-text co-guest-text">{{
      currentOrientation === LiveOrientation.Portrait ?
t('Portrait') : t('Landscape')
    }}</span>
  </div>
</template>

<script setup lang="ts">
import { ref, watch } from 'vue';
import { useUIKit, TUIToast, TOAST_TYPE, IconPortrait,
IconHorizontalMode } from '@tencentcloud/uikit-base-component-vue3';
import { useLiveListState, LiveOrientation } from 'tuikit-atomicx-
vue3';

const { t } = useUIKit();

enum TUISeatLayoutTemplate {
  LandscapeDynamic_1v3 = 200,
  PortraitDynamic_Grid9 = 600,
  PortraitDynamic_1v6 = 601,
  PortraitFixed_Grid9 = 800,
  PortraitFixed_1v6 = 801,
  PortraitFixed_6v6 = 802,
}
```

```
const { currentLive, updateLiveInfo } = useLiveListState();
const currentOrientation = ref(LiveOrientation.Portrait);

watch(
  () => currentLive.value?.layoutTemplate,
  newVal => {
    if (newVal === TUISeatLayoutTemplate.LandscapeDynamic_1v3) {
      currentOrientation.value = LiveOrientation.Landscape;
    } else {
      currentOrientation.value = LiveOrientation.Portrait;
    }
  },
  { immediate: true }
);

const handleOrientationSwitch = () => {
  if (currentLive.value?.liveId) {
    TUIToast({
      message: t('Cannot switch orientation during live streaming'),
      type: TOAST_TYPE.ERROR,
    });
    return;
  }
  if (currentOrientation.value === LiveOrientation.Portrait) {
    updateLiveInfo({ layoutTemplate:
TUISeatLayoutTemplate.LandscapeDynamic_1v3 });
  } else {
    updateLiveInfo({ layoutTemplate:
TUISeatLayoutTemplate.PortraitDynamic_Grid9 });
  }
};
</script>

<style scoped>.custom-icon-container{display:flex;flex-
direction:column;align-items:center;justify-
content:center;gap:4px;width:56px;height:56px;cursor:pointer;color:v
ar(--text-color-primary);border-
radius:12px;position:relative}.custom-
icon{width:24px;height:24px;background:transparent}.custom-
```

```
text{font-size:12px}.custom-icon-container:not(.disabled):hover{box-
shadow:0 0 10px 0 var(--bg-color-mask)}.custom-icon-
container:not(.disabled):hover .custom-icon,.custom-icon-
container:not(.disabled):hover .custom-text{color:var(--text-color-
link)}.custom-icon-container.disabled{cursor:not-
allowed;opacity:.5;color:var(--text-color-secondary)}.custom-icon-
container.disabled .custom-icon,.custom-icon-container.disabled
.custom-text{color:var(--text-color-secondary);cursor:not-allowed}
</style>
```

支持连观众能力

若您需要支持观众连麦能力，包括设置连麦申请、连麦管理等内容，请参考如下代码示例复制到 `live-pusher.vue` 文件中即可。

```
<!-- CoGuestButton 观众连麦设置 -->
<template>
  <div class="custom-icon-container" :class="{ disabled: disabled }"
@click="handleCoGuest">
    <span v-if="applicants.length > 0" class="unread-count">{{
applicants.length }}</span>
    <IconCoGuest class="custom-icon" />
    <span class="custom-text co-guest-text">{{ t('CoGuest') }}
</span>
  </div>
  <TUIDialog :title="t('CoGuest')" :visible="coGuestPanelVisible"
:custom-classes="['co-guest-dialog']" @close="coGuestPanelVisible =
false" @confirm="coGuestPanelVisible = false"
@cancel="coGuestPanelVisible = false">
    <CoGuestPanel class="co-guest-panel" />
    <template #footer>
      <div />
    </template>
  </TUIDialog>
</template>

<script lang="ts" setup>
import { computed, ref, watch } from 'vue';
```

```
import { useUIKit, TUIDialog, TUIToast, TOAST_TYPE, IconCoGuest }
from '@tencentcloud/uikit-base-component-vue3';
import { CoGuestPanel, CoHostStatus, useCoGuestState,
useCoHostState, useLiveListState } from 'tuikit-atomicx-vue3';

const { t } = useUIKit();
const { applicants } = useCoGuestState();
const { currentLive } = useLiveListState();
const { coHostStatus } = useCoHostState();
const disabled = computed(() => !currentLive.value?.liveId ||
coHostStatus.value !== CoHostStatus.Disconnected);

const coGuestPanelVisible = ref(false);

const handleCoGuest = () => {
  if (disabled.value) {
    const message = !currentLive.value?.liveId
      ? t('Cannot use co-guest before live starts')
      : t('Cannot enable audience co-hosting while co-hosting with
other hosts');
    TUIToast({ type: TOAST_TYPE.ERROR, message });
    return;
  }
  coGuestPanelVisible.value = true;
};

watch(disabled, () => {
  if (disabled.value) {
    coGuestPanelVisible.value = false;
  }
});
</script>

<style scoped>.custom-icon-container{display:flex;flex-
direction:column;align-items:center;justify-
content:center;gap:4px;width:56px;height:56px;cursor:pointer;color:v
ar(--text-color-primary);border-
radius:12px;position:relative}.unread-
count{position:absolute;top:0;right:0;background-color:var(--text-
color-error);border-
```

```
radius:50%;width:16px;height:16px;display:flex;align-
items:center;justify-content:center;font-size:12px}.custom-
icon{width:24px;height:24px;background:transparent}.custom-
text{font-size:12px}.custom-icon-container:not(.disabled):hover{box-
shadow:0 0 10px 0 var(--bg-color-mask)}.custom-icon-
container:not(.disabled):hover .custom-icon,.custom-icon-
container:not(.disabled):hover .custom-text{color:var(--text-color-
link)}.custom-icon-container.disabled{cursor:not-
allowed;opacity:.5;color:var(--text-color-secondary)}.custom-icon-
container.disabled .custom-icon{cursor:not-allowed}.custom-icon-
container.disabled .custom-text{color:var(--text-color-
secondary)}.co-guest-panel{height:560px}:deep(.co-guest-dialog)
{width:520px}</style>
```

支持布局设置能力

若您需要支持视频流切换布局能力，包括设置动态宫格布局、静态宫格布局、静态小窗布局、浮动小窗布局等内容，请参考如下代码示例复制到 `live-pusher.vue` 文件中即可。

```
<!-- OrientationSwitch 布局设置 -->
<template>
  <div
    class="custom-icon-container"
    :class="{ 'disabled': disabled }"
    @click="handleSwitchLayout"
  >
    <IconLayoutTemplate class="custom-icon" />
    <span class="custom-text setting-text">{{ t('Layout Settings')
}}</span>
  </div>
  <TUIDialog
    :customClasses="['layout-dialog']"
    :title="t('Layout Settings')"
    :visible="layoutSwitchVisible"
    @close="handleCancel"
    @confirm="handleConfirm"
    @cancel="handleCancel"
    appendTo="body"
```

```
  >
    <div class="layout-label">
      {{ t('Audience Layout') }}
    </div>
    <div class="template-options">
      <div class="options-grid">
        <template
          v-for="template in layoutOptions"
          :key="template.id"
        >
          <div
            class="option-card"
            :class="{ active: selectedTemplate ===
template.templateId }"
            @click="selectTemplate(template.templateId)"
          >
            <div class="option-info">
              <component
                :is="template.icon"
                v-if="template.icon"
                class="option-icon"
              />
              <h4>{{ template.label }}</h4>
            </div>
          </div>
        </template>
      </div>
    </div>
  </TUIDialog>
</template>

<script lang="ts" setup>
import { ref, computed, watch, h, defineComponent } from 'vue';
import { TUIErrorCode } from '@tencentcloud/tuiroom-engine-js';
import { useUIKit, TUIDialog, TUIToast, TOAST_TYPE,
IconLayoutTemplate } from '@tencentcloud/uikit-base-component-vue3';
import { useLiveListState, useCoHostState, CoHostStatus } from
'tuikit-atomicx-vue3';

/**
```

```
 * 直播间麦位排版模板
 */
enum TUISeatLayoutTemplate {
  LandscapeDynamic_1v3 = 200,
  PortraitDynamic_Grid9 = 600,
  PortraitDynamic_1v6 = 601,
  PortraitFixed_Grid9 = 800,
  PortraitFixed_1v6 = 801,
  PortraitFixed_6v6 = 802,
}

const createSvgIcon = (pathD: string) => defineComponent({
  render: () => h('svg', { xmlns: 'http://www.w3.org/2000/svg',
viewBox: '0 0 24 24', fill: 'currentColor', width: '24', height:
'24' }, [h('path', { d: pathD })])
});

const Dynamic1v6 = createSvgIcon('M3 3h7v7H3V3zm11 0h7v4h-7V3zm0
6h7v4h-7V9zm0 6h7v6h-7v-6zM3 12h7v9H3v-9z');
const DynamicGrid9 = createSvgIcon('M3 3h5v5H3V3zm7 0h4v5h-4V3zm6
0h5v5h-5V3zM3 10h5v4H3v-4zm7 0h4v4h-4v-4zm6 0h5v4h-5v-4zM3
16h5v5H3v-5zm7 0h4v5h-4v-5zm6 0h5v5h-5v-5z');
const Fixed1v6 = createSvgIcon('M2 2h9v9H2V2zm11 0h9v4h-9V2zm0
5h9v4h-9V7zm0 5h9v4h-9v-4zm0 5h9v5zM2 13h9v9H2v-9z');
const FixedGrid9 = createSvgIcon('M2 2h6v6H2V2zm7 0h6v6H9V2zm7
0h6v6h-6V2zM2 9h6v6H2V9zm7 0h6v6H9V9zm7 0h6v6h-6V9zM2 16h6v6H2v-6zm7
0h6v6H9v-6zm7 0h6v6h-6v-6z');
const HorizontalFloat = createSvgIcon('M2 4h20v12H2V4zm3 14h4v2H5v-
2zm6 0h4v2h-4v-2zm6 0h4v2h-4v-2z');

const { t } = useUIKit();
const { currentLive, updateLiveInfo } = useLiveListState();
const { coHostStatus } = useCoHostState();
const disabled = computed(() => coHostStatus.value ===
CoHostStatus.Connected);

watch(
  () => currentLive.value?.liveId,
  (liveId) => {
    if (!liveId) {
```

```
      updateLiveInfo({ layoutTemplate:
TUISeatLayoutTemplate.PortraitDynamic_Grid9 });
    }
  },
  { immediate: true },
);

const layoutSwitchVisible = ref(false);

const handleSwitchLayout = () => {
  if (disabled.value) {
    TUIToast({ type: TOAST_TYPE.ERROR, message: t('Layout switching
is not available during co-hosting') });
    return;
  }
  layoutSwitchVisible.value = true;
};

const portraitLayoutOptions = computed(() => [
  {
    id: 'PortraitDynamic_Grid9',
    icon: DynamicGrid9,
    templateId: TUISeatLayoutTemplate.PortraitDynamic_Grid9,
    label: t('Dynamic Grid9 Layout'),
  },
  {
    id: 'PortraitFixed_1v6',
    icon: Fixed1v6,
    templateId: TUISeatLayoutTemplate.PortraitFixed_1v6,
    label: t('Fixed 1v6 Layout'),
  },
  {
    id: 'PortraitFixed_Grid9',
    icon: FixedGrid9,
    templateId: TUISeatLayoutTemplate.PortraitFixed_Grid9,
    label: t('Fixed Grid9 Layout'),
  },
  {
    id: 'PortraitDynamic_1v6',
    icon: Dynamic1v6,
```

```
    templateId: TUISeatLayoutTemplate.PortraitDynamic_1v6,
    label: t('Dynamic 1v6 Layout'),
  },
]);

const horizontalLayoutOptions = computed(() => [
  {
    id: 'LandscapeDynamic_1v3',
    icon: HorizontalFloat,
    templateId: TUISeatLayoutTemplate.LandscapeDynamic_1v3,
    label: t('Landscape Template'),
  },
]);

const layoutOptions = computed(() => {
  if (currentLive.value && currentLive.value?.layoutTemplate >= 200
&& currentLive.value?.layoutTemplate <= 599) {
    return horizontalLayoutOptions.value;
  }
  return portraitLayoutOptions.value;
});

const selectedTemplate = ref<TUISeatLayoutTemplate | null>
(currentLive.value?.layoutTemplate ?? null);

function selectTemplate(template: TUISeatLayoutTemplate) {
  selectedTemplate.value = template;
}

watch(() => currentLive.value?.layoutTemplate, (newVal) => {
  if (newVal) {
    selectedTemplate.value = newVal;
  }
});

async function handleConfirm() {
  if (selectedTemplate.value) {
    try {
      await updateLiveInfo({ layoutTemplate: selectedTemplate.value
});
```

```
      layoutSwitchVisible.value = false;
    } catch (error: any) {
      let errorMessage = t('Layout switch failed');
      if (error.code === TUIErrorCode.ERR_FREQ_LIMIT) {
        errorMessage = t('Operation too frequent, please try again
later');
      }
      TUIToast({ type: TOAST_TYPE.ERROR, message: errorMessage });
    }
  } else {
    layoutSwitchVisible.value = false;
  }
}

function handleCancel() {
  selectedTemplate.value = currentLive.value?.layoutTemplate ??
null;
  layoutSwitchVisible.value = false;
}
</script>

<style scoped>
.custom-icon-container { display: flex; flex-direction: column;
align-items: center; justify-content: center; gap: 4px; width: 56px;
height: 56px; cursor: pointer; color: var(--text-color-primary);
border-radius: 12px; position: relative; } .custom-icon-container
.custom-icon { display: inline-block; width: 24px; height: 24px;
background: transparent; } .custom-icon-container .custom-text {
font-size: 12px; font-weight: 400; } .custom-icon-
container:not(.disabled):hover { box-shadow: 0 0 10px 0 var(--bg-
color-mask); } .custom-icon-container:not(.disabled):hover .custom-
icon { color: var(--text-color-link); } .custom-icon-
container:not(.disabled):hover .custom-text { color: var(--text-
color-link); } .custom-icon-container.disabled { cursor: not-
allowed; opacity: 0.5; color: var(--text-color-tertiary); } .custom-
icon-container.disabled .custom-icon { color: var(--text-color-
tertiary); cursor: not-allowed; } .custom-icon-container.disabled
.custom-text { color: var(--text-color-tertiary); } :deep(.layout-
dialog) { padding: 24px; width: 480px; } :deep(.layout-dialog)
.dialog-body { flex-wrap: wrap; } :deep(.layout-dialog) .dialog-
```

```
footer { padding-top: 32px; } .layout-label { font-size: 14px; font-
weight: 400; color: var(--text-color-primary, #ffffff); margin: 4px
0px 16px 0px; } .template-options { width: 100%; height: 100%;
overflow: auto; } .template-options .options-grid { display: flex;
flex-wrap: wrap; gap: 16px; justify-content: flex-start; }
.template-options .options-grid .option-card { box-sizing: border-
box; padding: 12px 13px; width: 208px; background: #3a3a3a; border:
2px solid transparent; border-radius: 12px; cursor: pointer;
transition: all 0.2s ease; text-align: center; } .template-options
.options-grid .option-card:hover { background: #4a4a4a; border-
color: #5a5a5a; } .template-options .options-grid .option-
card.active { border: 2px solid var(--text-color-link-hover,
#2B6AD6); background: var(--list-color-focused, #243047); }
.template-options .options-grid .option-card.active .option-info h4
{ color: #ffffff; } .template-options .options-grid .option-card
.option-info { display: flex; align-items: center; justify-content:
flex-start; gap: 8px; } .template-options .options-grid .option-card
.option-info .option-icon { width: 24px; height: 24px; } .template-
options .options-grid .option-card .option-info h4 { margin: 0;
font-size: 14px; font-weight: 600; color: #ffffff; transition: color
0.2s ease; }
</style>
```

自定义您的界面布局

`TUILiveKit` 支持灵活定制开播页与直播页的功能和样式，您可根据业务需求调整布局、隐藏 / 显示功能模块。

横竖屏推流设置

`TUILiveKit` 支持**横屏**和**竖屏**两种推流模式，您可根据直播场景选择合适的推流方向：

推流模式	适用场景	说明
竖屏推流	秀场直播、电商带货、聊天互动	默认模式，适合移动端观看，画面比例为 9:16。
横屏推流	游戏直播、在线教育、会议直播	适合 PC 端观看，画面比例为 16:9。

> ⚠ **注意：**

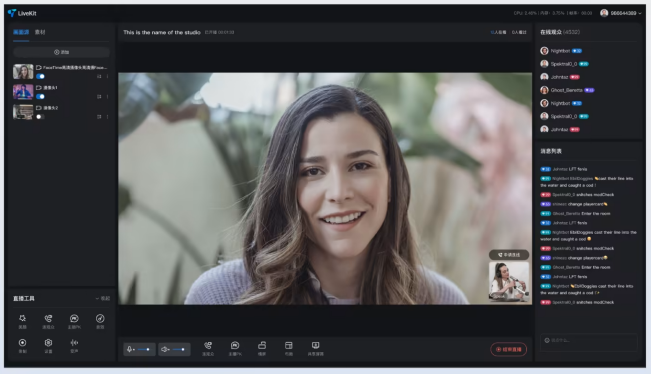
> 横竖屏切换必须在**开播前**进行设置，直播过程中无法切换推流方向。

通过 UI 交互切换

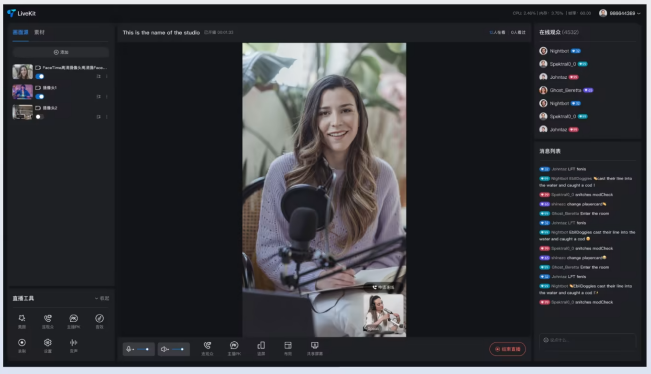
在主播开播页的底部控制栏，点击**横屏 / 竖屏**按钮即可切换推流方向：

- 显示**竖屏**时，当前为**竖屏推流模式**。
- 显示**横屏**时，当前为**横屏推流模式**

横屏布局



竖屏布局



通过代码设置

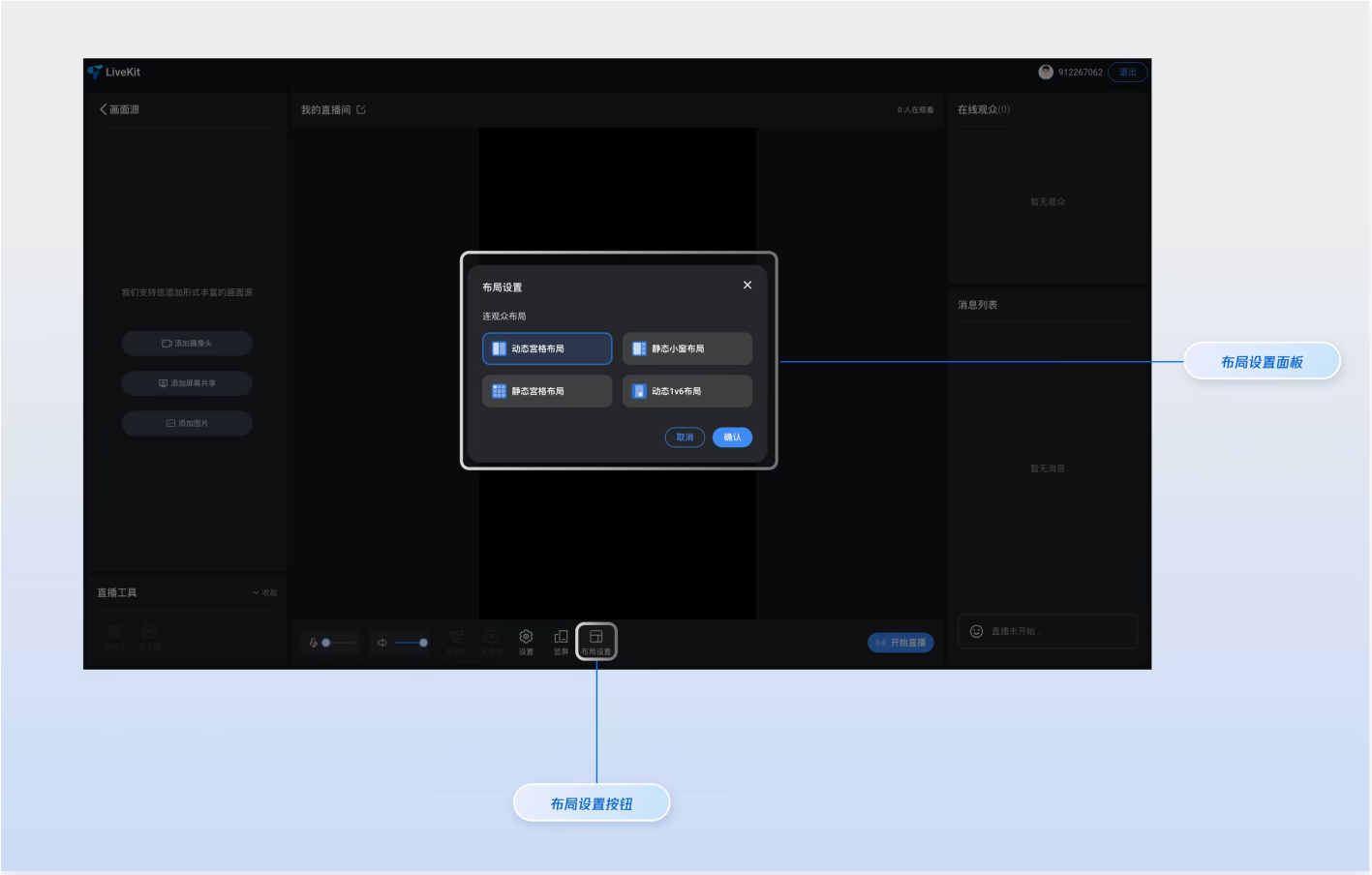
您也可以通过调用 `updateLiveInfo` 方法，在代码中设置推流方向：

```
import { useLiveListState } from 'tuikit-atomicx-vue3';

const { updateLiveInfo } = useLiveListState();

// 切换为横屏模式（模板 ID：200）
updateLiveInfo({ layoutTemplate: 200 });

// 切换为竖屏模式（模板 ID：600）
updateLiveInfo({ layoutTemplate: 600 });
```

> **说明：**
>
> - 横屏模式对应的布局模板 ID 范围为 200–599，默认使用 200（横屏浮动布局）。
> - 竖屏模式对应的布局模板 ID 范围为 600–899，默认使用 600（动态宫格布局）。
> - 切换横竖屏时，系统会自动切换到对应方向的默认布局模板。

直播布局模板选择

`TUILiveKit` 提供多种直播布局模板，用于控制主播与连麦嘉宾的画面排列方式。您可在主播开播页的**布局设置**选择合适样式：



竖屏模式布局模板

竖屏模式下提供 **4 种布局模板**，适用于秀场直播、电商带货等场景：

名称	动态宫格布局	动态 1V6 布局	静态宫格布局	静态小窗布局
模板 ID	600	601	800	801
描述	**默认布局**，根据连麦人数动态调整宫格大小，画面自适应填充。	主播画面居中大窗显示，连麦嘉宾以**浮动小窗**形式环绕在主播周围。	固定 9 宫格布局，每个嘉宾占据一个**固定大小**的宫格位置。	主播画面全屏显示，连麦嘉宾以**固定小窗**形式显示在画面边缘。
适用场景	通用场景，连麦人数不固定。	主播为主、嘉宾为辅的互动场景。	多人连麦、语音聊天室等固定人数场景。	主播全屏展示、嘉宾辅助的场景。

横屏模式布局模板

横屏模式下提供 **1 种布局模板**，适用于游戏直播、在线教育等场景：

名称	横屏浮动布局
模板 ID	200
描述	主播画面全屏显示，连麦嘉宾以**浮动小窗**形式显示在画面底部。
适用场景	游戏直播、屏幕分享、在线教育等需要横屏展示的场景。

通过代码设置布局模板

您可以通过调用 `updateLiveInfo` 方法，在代码中设置布局模板：

```
import { useLiveListState } from 'tuikit-atomicx-vue3';

const { updateLiveInfo } = useLiveListState();

// 设置竖屏动态宫格布局
updateLiveInfo({ layoutTemplate: 600 });

// 设置竖屏动态 1v6 布局
updateLiveInfo({ layoutTemplate: 601 });

// 设置竖屏静态宫格布局
updateLiveInfo({ layoutTemplate: 800 });

// 设置竖屏静态 1v6 布局
updateLiveInfo({ layoutTemplate: 801 });

// 设置横屏浮动布局
updateLiveInfo({ layoutTemplate: 200 });
```

注意：

- 布局模板可以在**开播前**和**直播中**进行切换。
- 在主播 PK 连线期间，**无法切换**布局模板。
- 切换横竖屏推流方向时，布局模板会自动切换为对应方向的默认模板。

自由定制

颜色主题及语言

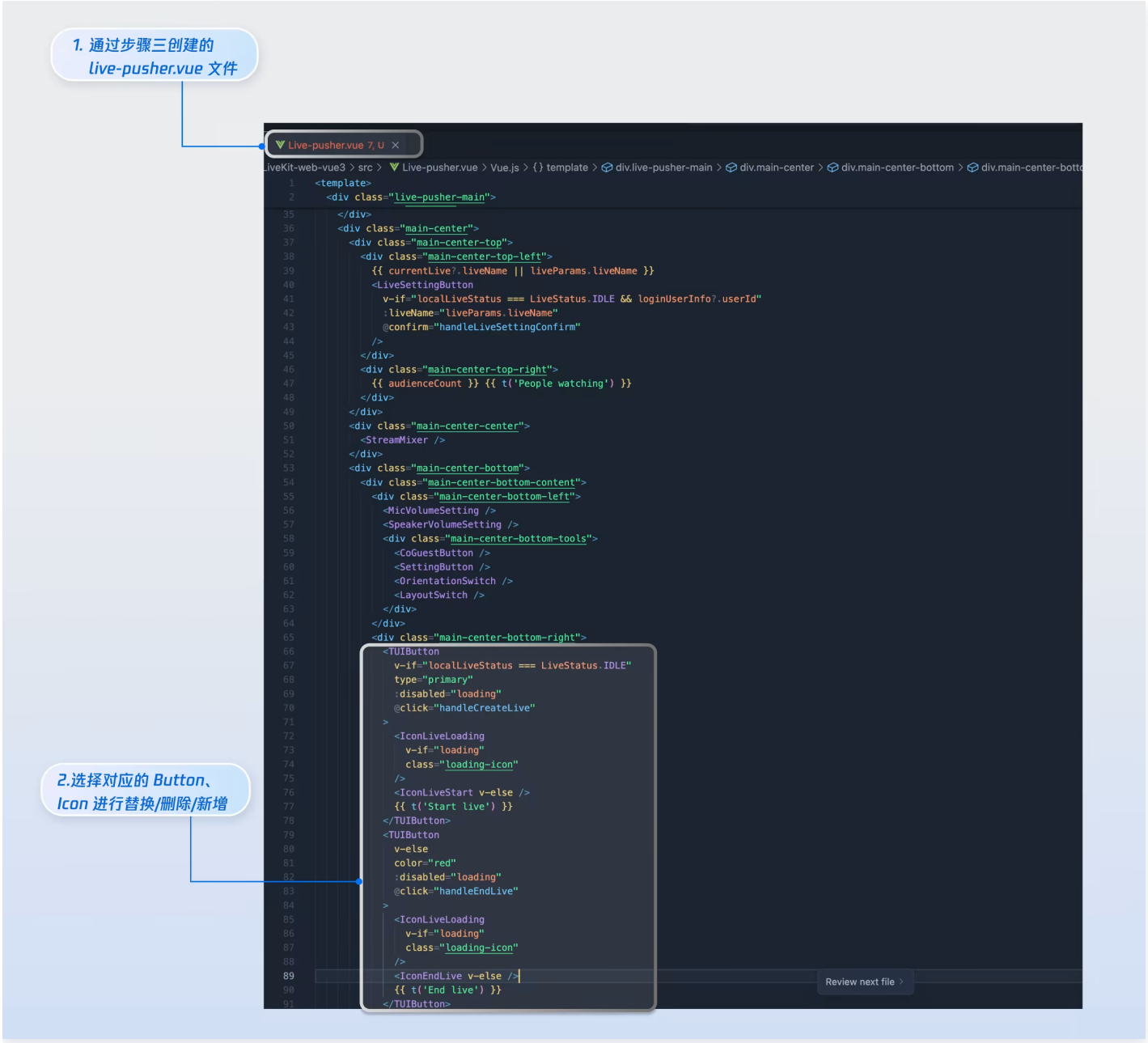
通过配置 `App.vue` 中 `UIKitProvider` 的入参，修改主题及语言的默认值。

UIKitProvider 参数	可选值	默认值
theme	"light" | "dark"	"light"
language	"zh-CN" | "en-US"	"en-US"

```
<UIKitProvider theme="light">
  <router-view />
</UIKitProvider>

<script setup lang="ts">
import { UIKitProvider } from '@tencentcloud/uikit-base-component-vue3';
```

按钮 Button 和图标 Icon

若您需要对按钮 Button 或图标 Icon 等其他控件进行新增或替换等 UI 定制，您可以通过如下方式实现，以 `live-player.vue` 文件中的按钮和图标为例，您可以参考下图找到对应按钮或图标的指定位置源码，对当前部分的控件进行**增加、删除、替换**等 UI 定制操作。



根据上述示例，我们同样支持您根据项目需求对**观众观看页面**进行 UI 定制的能力。除了页面 UI 布局调整，我们也支持您对**颜色主题、字体、圆角、按钮、图标、输入框、弹框等**内容进行**增加、删除、修改**等操作，满足您的 UI 定制需要。

类别	功能	描述	组件详细定制指引
素材管理	自定义素材管理区域展示	支持： • 调整展示 Icon 的大小、颜色或对 Icon 进行替换。	媒体源配置面板
在线观众	自定义观众信息展示	支持： • 展示/隐藏观众等级。	视频源编辑画板

		• 观众信息字体、颜色 UI 自定义设置。 • 替换为您需要的 Icon 风格。	
消息列表	自定义消息弹幕区域展示	支持： • 展示/隐藏聊天输入区域。 • 支持 UI 定制聊天气泡风格、定制观众等级等内容。	聊天弹幕组件

下一步

恭喜您，现在您已经成功集成了**主播开播页面** 。接下来，您可以实现**观众观看页**、**直播列表页**和 **UI 自定义**等内容，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直播间后观看直播，实现观众连麦 、直播间信息、在线观众、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

主播开播（Flutter）

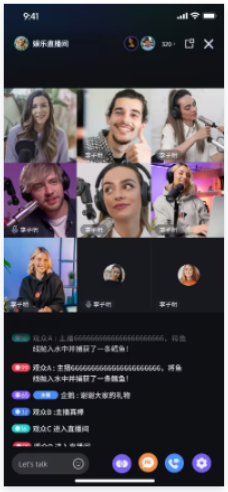
最近更新时间：2026-04-20 17:34:02

TUILiveKit 主播开播页为直播场景提供**开箱即用的全功能界面**，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 UI 与逻辑实现，即可高效集成直播开播流程。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的开播方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建主播端界面，请参考 主播开播 (Core SDK)。

功能概览

- **开播前预览：**支持主播开播前的房间名称、背景、视频预览、美颜调试、音效调试、布局模板等多种个性化配置。
- **连麦互动：**支持直播过程中与观众 或 与其它直播间主播实时互动。
- **观众互动：**支持弹幕、礼物等丰富直播互动形式。

开播前预览	连麦互动	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：添加主播页面

`TUILiveRoomAnchorWidget` 已内置了直播场景的主播端完整 UI 与业务逻辑，该组件不支持浮窗模式，如需浮窗功能请转到 添加浮窗版主播页面 。您只需要配置 `TUILiveRoomAnchorWidget` 的调用入口（具体由您的业务决定），执行如下操作，跳转到主播页面或将主播页面集成到自己的 Widget 树中：

直接跳转

```
import 'package:tencent_live_uikit/tencent_live_uikit.dart';

// 跳转到主播页面
Navigator.push(context, MaterialPageRoute(
  builder: (BuildContext context) {
      final roomId = "test_live_room_id";
      return TUILiveRoomAnchorWidget(roomId: roomId);
}));
```

集成到 Widget 树

```
// --- 根据您的Widget树结构，选择以下一种方式集成 ---

// [选项一] 作为唯一子Widget（单子树）
// 适用于Container、Padding等通常只包含一个子Widget的容器
Container(
    child: TUILiveRoomAnchorWidget(roomId: roomId) // 在此处集成主播页
)

// [选项二] 作为多个子Widget之一（多子树）
// 适用于Column、Row、Stack等可以包含多个子Widget的布局
Stack(
    children: [
      YourOtherWidget(), // 您的其他子Widget
      TUILiveRoomAnchorWidget(roomId: roomId), // 在此处集成主播页
      YourOtherWidget(), // 您的其他子Widget
```

```
  ])
```

集成后，调用代码即可拉起主播页面，效果如 功能概览 的开播预览图所示。

步骤4：（可选）添加浮窗版主播页面

`TUILiveRoomAnchorOverlay` 是支持浮窗模式的主播页面。在直播期间，可以切换到 **应用内** 和 **应用外（即画中画）** 2 种场景的浮窗模式。`TUILiveRoomAnchorOverlay` 是基于 Flutter 官方 API **Overlay** 和 原生画中画实现的，具体接入流程如下：

1. App 工程配置开启系统画中画特性

参考 工程配置，开启系统画中画特性。

2. 跳转浮窗版主播页面

在您需要主播开播的调用入口（具体由您的业务决定），执行如下操作，跳转到主播页面。

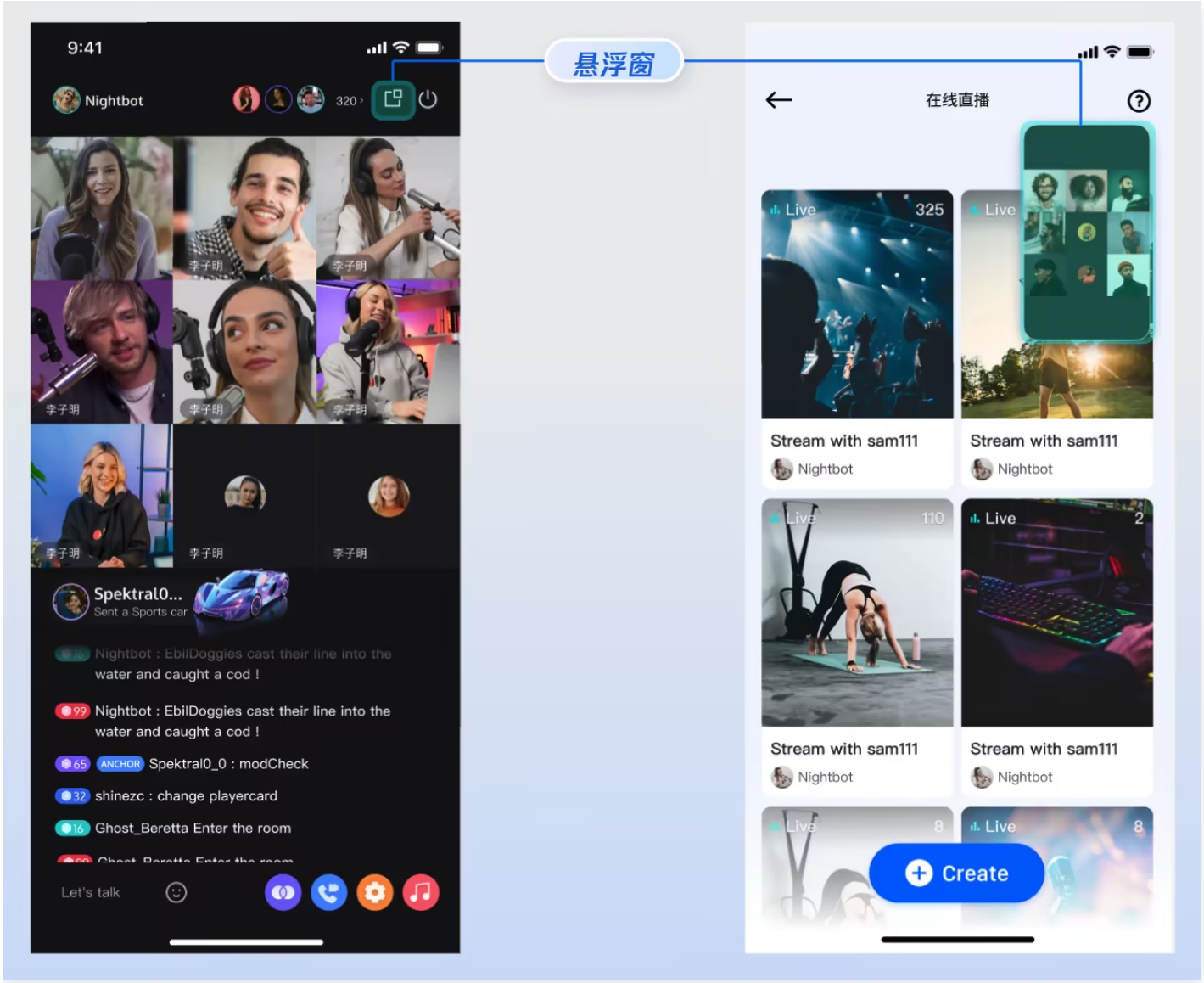
```
import 'package:tencent_live_uikit/tencent_live_uikit.dart';

// 跳转到主播页面
Navigator.push(context, MaterialPageRoute(
  builder: (BuildContext context) {
     final roomId = "test_live_room_id";
     return TUILiveRoomAnchorOverlay(roomId: roomId);
}));
```

说明：

1. `TUILiveRoomAnchorOverlay` 不支持作为子 widget 嵌入到容器类 widget（例如：Container、Stack 等），只能作为独立页面跳转。因为内部使用了 Overlay，LiveKit 需要操控整个 Overlay 页面来切换浮窗模式。
2. iOS 端仅支持普通观众进入应用外浮窗模式。

进入应用内浮窗模式

点击主播页面右上角的浮窗按钮，进入浮窗模式，效果如图所示：



进入应用外浮窗模式

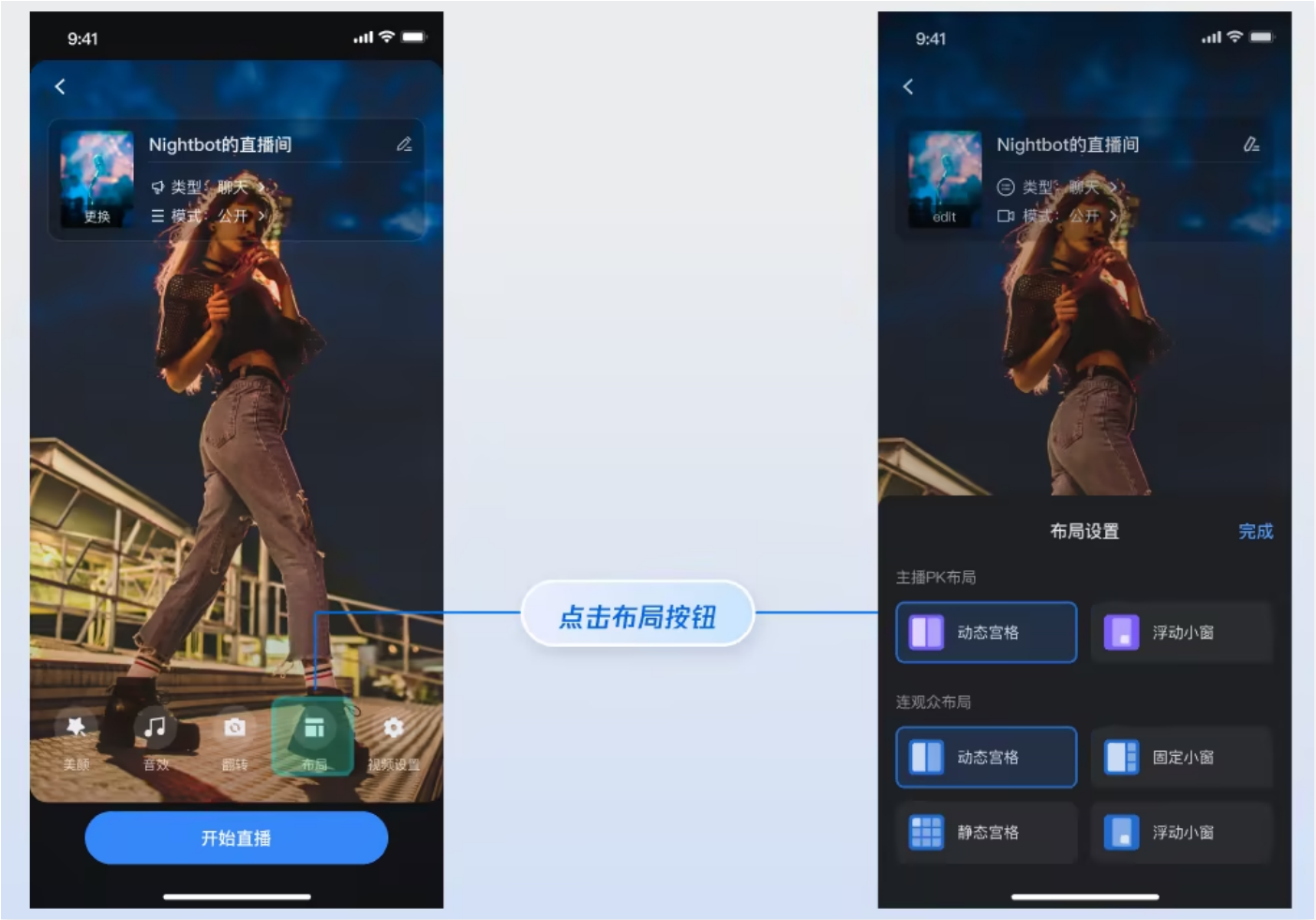
依次点击主播页面右下角的**更多 > 画中画**，打开画中画开关。然后，您的 App 退后台时，将自动进入到系统画中画模式，效果如图所示：

自定义您的界面布局

TUILiveKit 支持灵活定制开播页与直播页的功能和样式，您可根据业务需求调整布局。

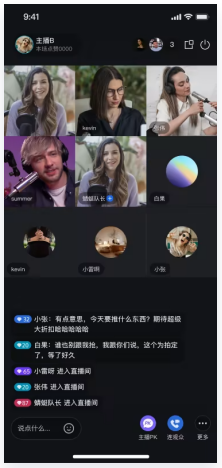
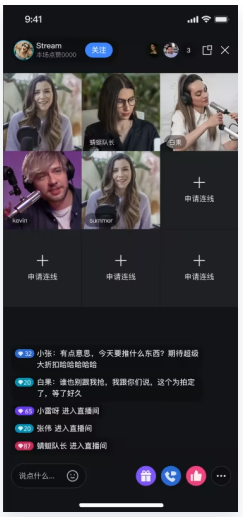
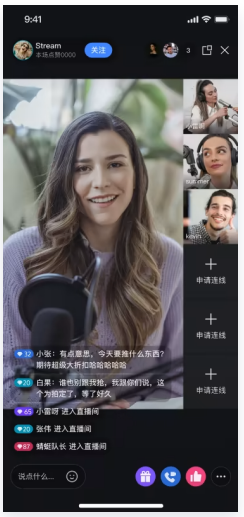
直播布局模板选择

TUILiveKit 提供 **4 种直播布局模板**，您可在主播开播页的**布局**入口选择合适样式：



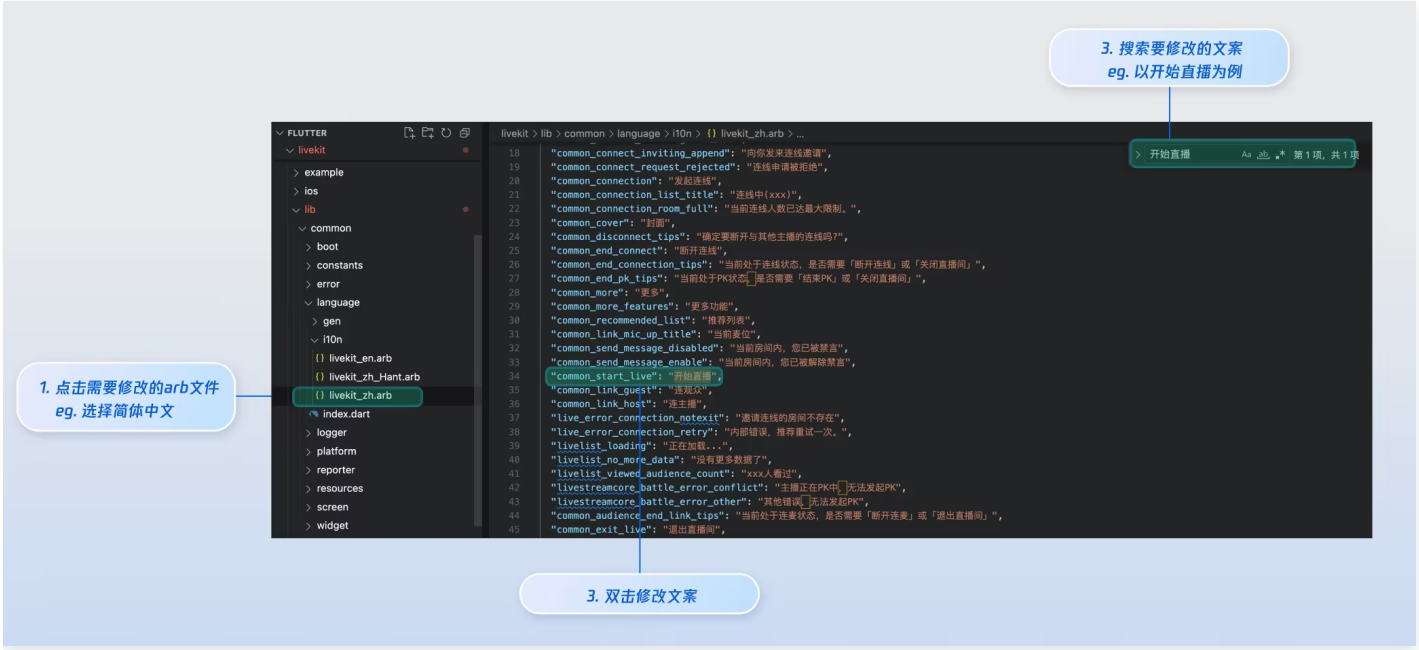
布局模板一览：

名称	动态宫格布局	浮动小窗布局	固定宫格布局	固定小窗布局
模板 ID	600	601	800	801
描述	默认布局，可根据连麦人数动态调整宫格大小。	连麦嘉宾以浮动小窗形式显示。	连麦人数固定，每个嘉宾占据一个固定宫格。	连麦人数固定，嘉宾以固定小窗形式显示。

预览				

文案定制

TUILiveKit 使用 ARB 文件和 Flutter 标准国际化方案来管理 UI 文案显示。您可以直接编辑 `livekit/lib/common/language/i10n/` 目录下的ARB文件来修改需要调整的文案：

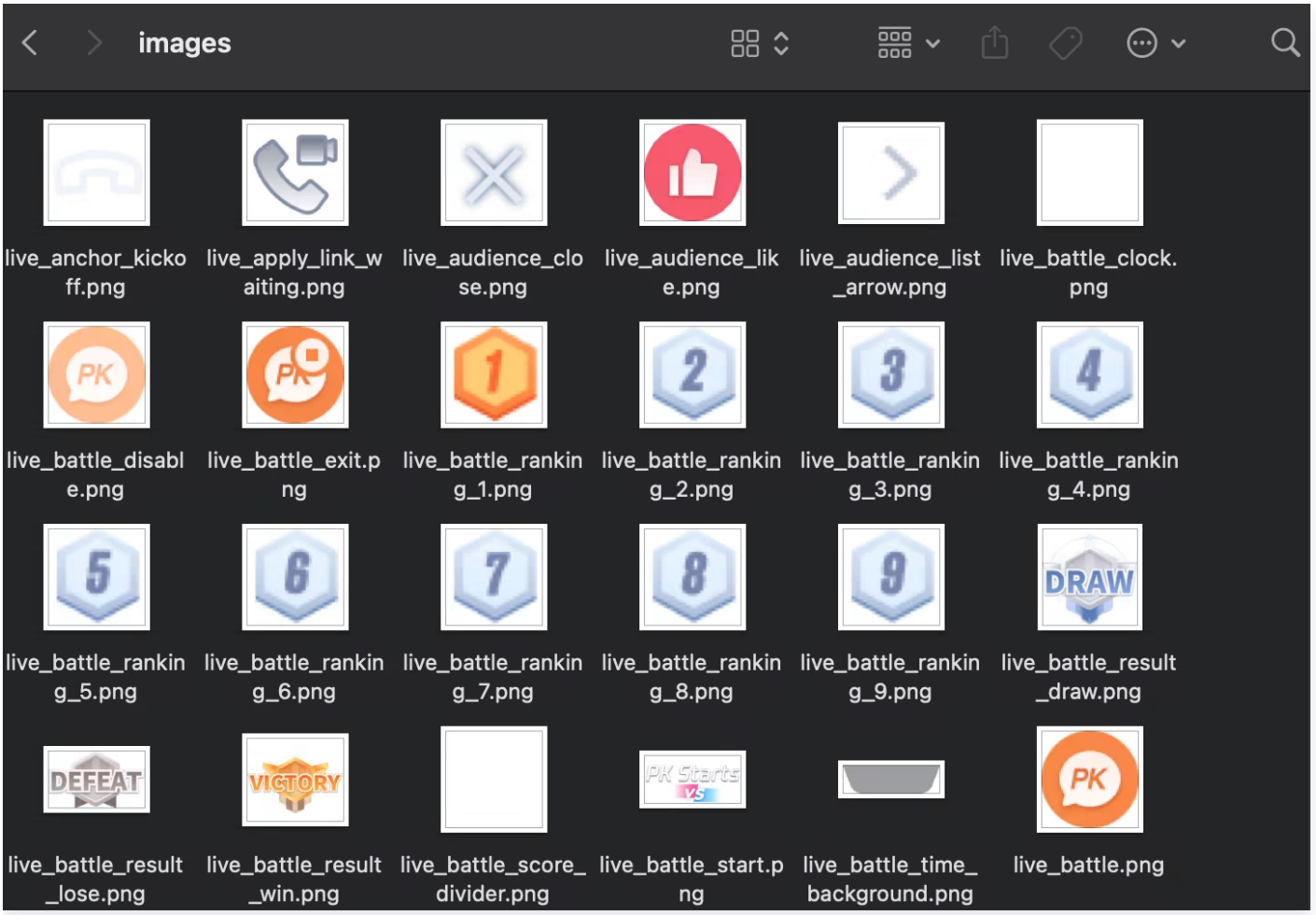


- `livekit_en.arb`：英文文案。
- `livekit_zh.arb`：简体中文文案。
- `livekit_zh_Hant.arb`：繁体中文文案。

修改后通过命令行执行 `flutter gen-l10n` 重新生成本地化代码即可。生成成功后，`livekit/lib/common/language/gen/` 目录里的代码文件将会刷新。

图标定制

TUILiveKit UI所需的图片资源在 `livekit/assets/images/` 目录下管理，您可以直接替换该目录下的 PNG 图片文件来快速修改自定义界面所需的图标。



重新构建并运行应用，即可看到更新的图标。

下一步

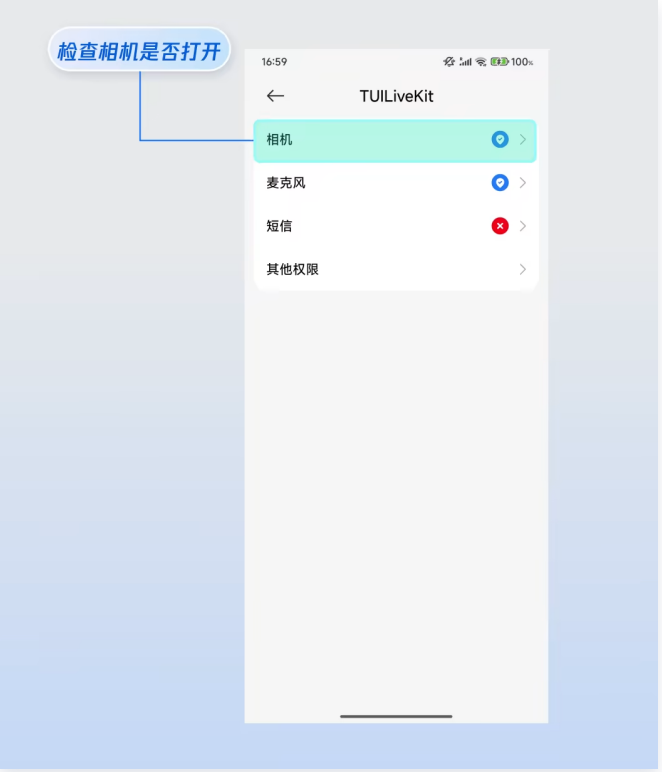
恭喜您，现在您已经成功集成了**主播开播**。接下来，您可以实现**观众观看**、**直播列表**等功能，可参考下表：

功能	描述	集成指引
观众观看	实现观众进入主播的直播间后观看直播，实现观众连麦 、直播间信息、在线观众、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，包含直播列表和房间信息展示功能	直播列表

常见问题

开播后无画面?

请前往**手机设置** > **App** > **相机**，检查摄像头权限是否开启，可参考如下示例：

iOS	Android
	

开播提示“未登录”？

参考 登录指引， 确认已完成登录功能接入。

App 退后台无法进入应用外浮窗（画中画）模式?

参考 进入应用外浮窗模式，进入直播间后，开启了画中画开关。
如果是 Android 平台，在您打开画中画开关后，还会自动检测 App 系统画中画模式开关状态，如果没有打开，App 会自动跳转到设置页面，请您允许进入画中画模式：


16:57
画中画
app-uikit
版本 1.0.0
允许进入画中画模式
允许此应用在您打开或离开应用（如继续观看视频）时创建画中画窗口。该窗口会显示在您当前使用的其他应用的上层。
请确保打开画中画开关

主播开播（uni-app 客户端）

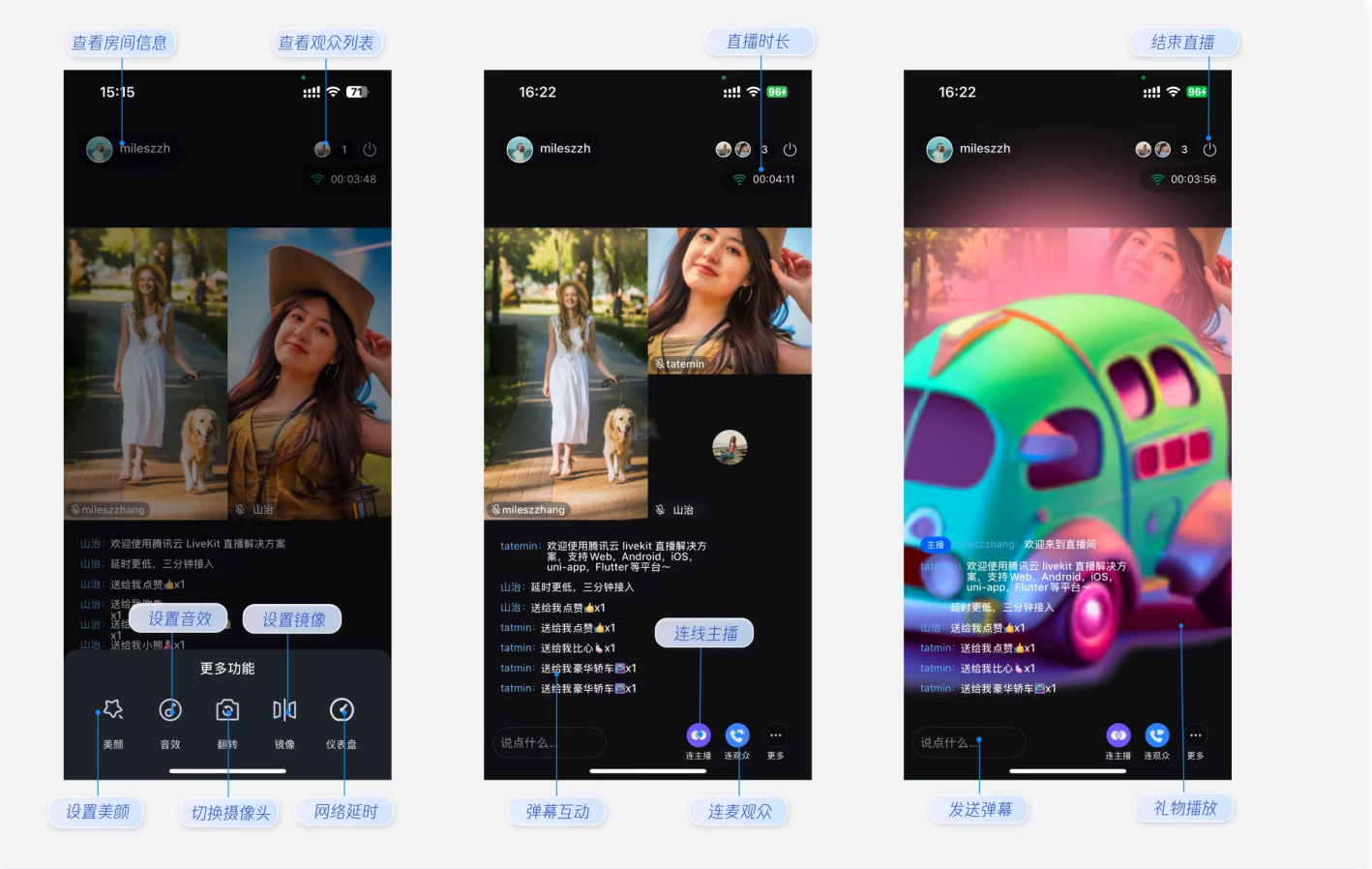
最近更新时间：2026-04-20 17:34:02

本文将指导您如何快速集成主播开播页，实现主播开播的能力，体验摄像头预览、美颜、音效、摄像头翻转、观众连麦、主播连线、直播间信息展示、观众列表、礼物播放和弹幕互动等功能。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的开播方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建主播端界面，请参考 主播开播 (Core SDK)。

功能预览

主播开播页提供默认的行为和样式，但如果默认行为和样式不能完全满足您的需求，您也可以对 UI 进行自定义。下图介绍了主播开播页的主要功能。



开始接入

步骤1：集成核心文件

已集成核心文件请忽略，已经按照 代码集成指引 将 TUILiveKit 的 `pages` 、 `uni_modules` 等核心文件拷贝到了您的项目中。

步骤2：完成登录

已登录请忽略该步骤，您的应用已经调用了 登录接口 并成功登录。这是使用所有功能的基础

步骤3：制作并使用自定义基座

已制作自定义基座请忽略，由于直播功能依赖原生插件，您必须在 HBuilderX 中为您的项目制作并使用自定义调试基座来运行。

步骤4：注册主播开播页面

现在，您需要在 `pages.json` 文件中告诉您的应用，这个新页面是存在的。
打开您项目根目录下的 `pages.json` 文件，在 `"pages"` 数组中，添加以下对象来注册主播开播页。

```
{
    "pages": [
        // ... 您项目已有的其他页面配置
        {
            "path": "pages/scenes/live/anchor/index",
            "style": {
                "navigationBarTitleText": "",
                "disableSwipeBack": true, // 禁止右滑返回，防止直播中误操作退出
                "app-plus": {
                    "titleNView": false // 隐藏原生导航栏，使用页面内的自定义导航
                }
            }
        }
    ]
    // ... 其他配置
}
```

步骤5：跳转主播开播页面

在您需要开启直播的地方（具体由您的业务决定，在其点击事件里执行），调用 `uni.navigateTo` 或 `uni.redirectTo` 方法，即可进入主播开播页面。

```
// 示例：在一个按钮的点击事件中
function startBroadcast() {
    // 跳转到主播开播页
  uni.redirectTo({
```

```
    url: '/pages/scenes/live/anchor/index' // URL 对应 pages.json 中配置的
path
    });
}
```

说明：

navigateTo 与 redirectTo

- `uni.navigateTo`：保留当前页面，跳转到新页面。主播可以从开播页返回到上一页。适用于开播前可以取消的场景。
- `uni.redirectTo`：关闭当前页面，跳转到新页面。主播无法返回。适用于点击后直接进入开播流程的场景。 您可以根据自己的业务需求选择使用。

定制您的 UI

替换主播开播页图标

TUILiveKit 用到的所有图标都存放于 `static/images` 目录下，部分示例如下，您可以根据您的诉求来替换目录下的图标。

图标路径	详细描述
/static/images/link-host.png	底部操作栏的“连主播”图标
/static/images/link-guest.png	底部操作栏的“连观众”图标
/static/images/live-more.png	底部操作栏的“更多”图标
/static/images/live-end.png	顶部操作栏的“结束直播”图标

设置字体的大小/颜色

uniapp 官方对 nvue 页面限制字体的大小和颜色只能在 `text` 标签上设置，对于您想修改的文案，直接修改其 `css` 样式即可，示例如下

以 “连主播” 文字为例：

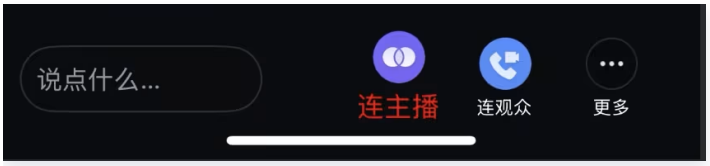
```
<text class="action-button-text">连主播</text>
  .action-button-text {
    color: #fff;
```

```
    font-size: 20rpx;
  }
```

修改其 css 样式：

```
<text class="action-button-text">连主播</text>
  .action-button-text {
    color: red;
    font-size: 40rpx;
  }
```

最终效果：



设置按钮的大小

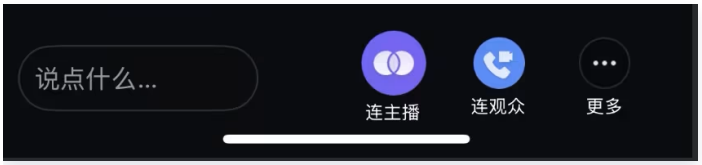
设置按钮的大小，也可以直接通过修改 `css` 属性来进行设置，示例如下
以 “连主播” 按钮为例：

```
<image class="action-button-icon" src="/static/images/link_host.png"
mode="aspectFit" />
  .action-button-icon {
    width: 56rpx;
    height: 56rpx;
    margin-bottom: 4rpx;
  }
```

设置对应的 `css` 属性：

```
<image class="action-button-icon" src="/static/images/link_host.png"
mode="aspectFit" />
   .action-button-icon {
    width: 70rpx;
    height: 70rpx;
    margin-bottom: 4rpx;
  }
```

最终效果：



隐藏按钮

隐藏按钮可以通过注释代码的方式来进行直接隐藏，示例如下
以“连主播”的按钮为例：

```
<!-- <view class="action-button-item" @tap="showCoHostPanel">
        <image class="action-button-icon" src="/static/images/link-
host.png" mode="aspectFit" />
        <text class="action-button-text">连主播</text>
      </view> -->
```

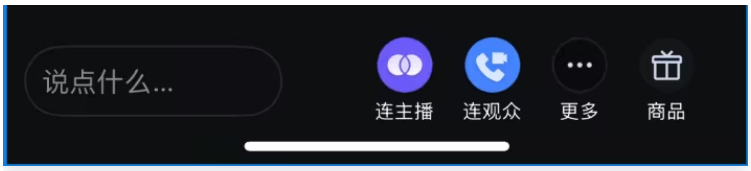
将其注释掉即可实现隐藏按钮。

新增按钮

在您需要新增的位置，插入对应的按钮实现。
以底部栏新增“商品”按钮为例：

```
<template>
        ......
    <!-- 底部操作栏 -->
    <view class="live-bottom-panel" >
        ......
        <view class="action-button-item" @tap="showProductPanel">
           <image class="action-button-icon"
src="/static/images/product.png" mode="aspectFit" />
           <text class="action-button-text">商品</text>
        </view>
     </view>
</template>
```

最终效果：



其他参考文档

- TUILiveKit uni-app 观众观看页
- TUILiveKit uni-app 直播列表页

主播开播（PC 推流助手）

最近更新时间：2026-04-20 17:34:02

概述

对 Electron 直播推流助手的核心功能以及如何进行快速接入进行详细的介绍。只需简单几步，您就能快速启动一个功能强大的桌面端高清视频直播软件。

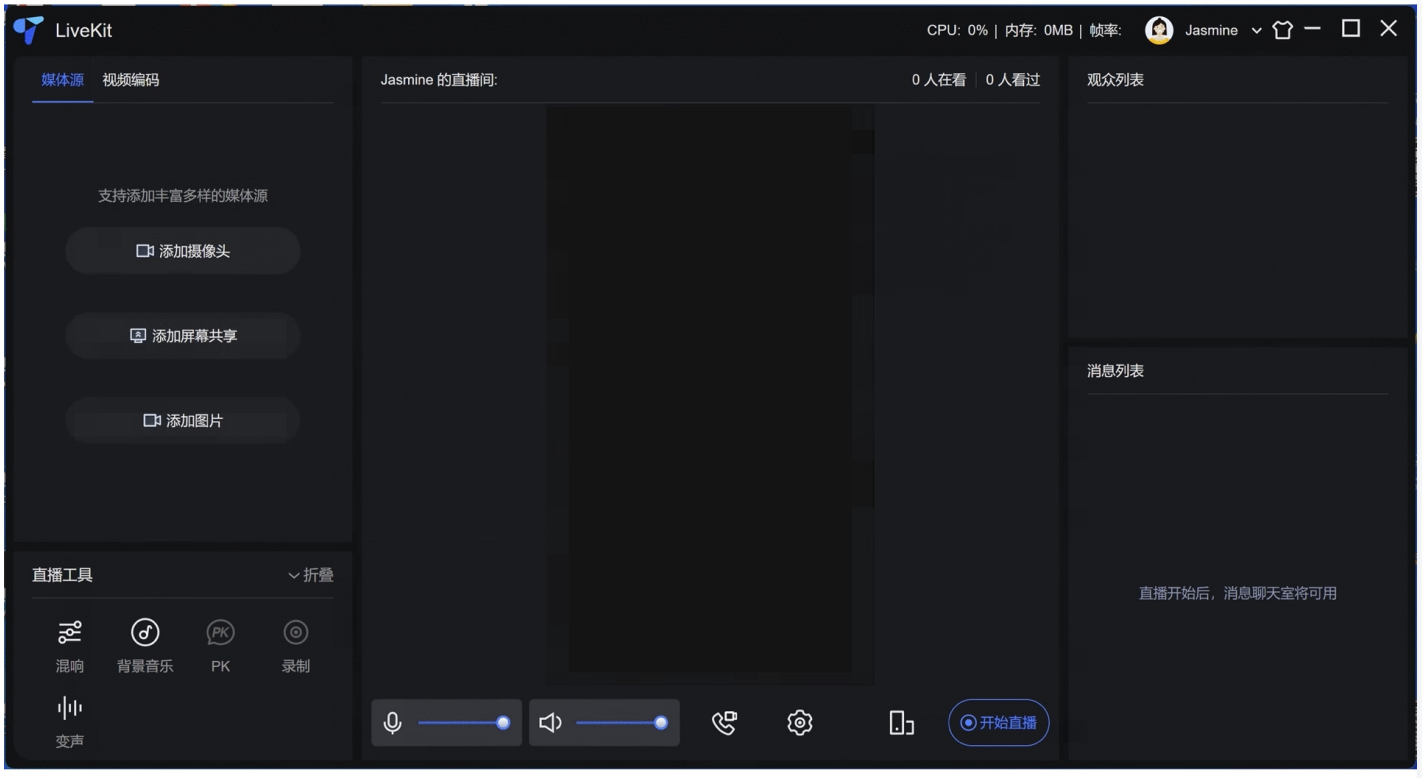
功能展示

本地混流

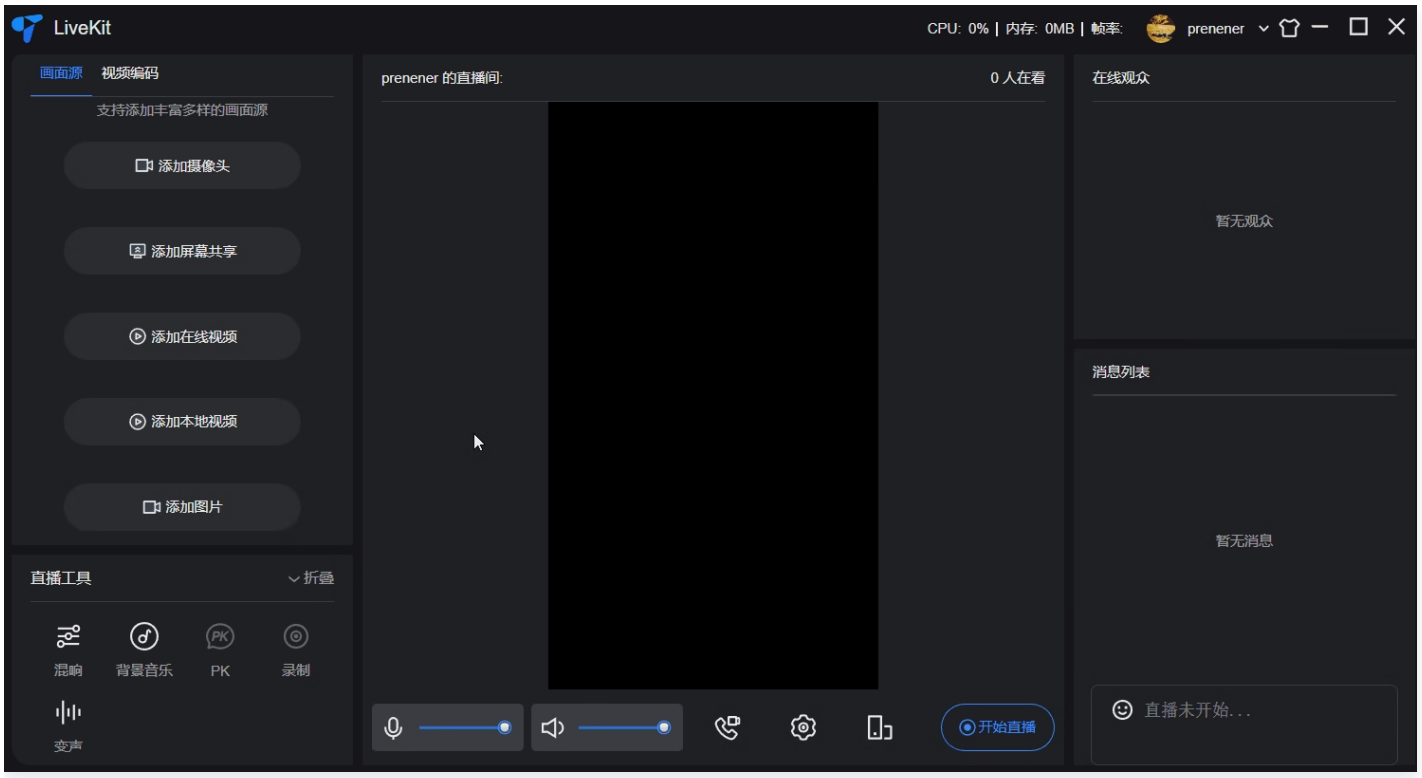
支持多种媒体源本地混流并自由排版（位置、大小、层级可编辑）：

- 摄像头。
- 屏幕/窗口分享。
- 本地图片。
- 本地视频：支持 mp4，mkv，mov 视频格式。
- 在线视频：支持 http，https，rtmp 协议在线视频流。

1. 摄像头、屏幕分享和图片混流

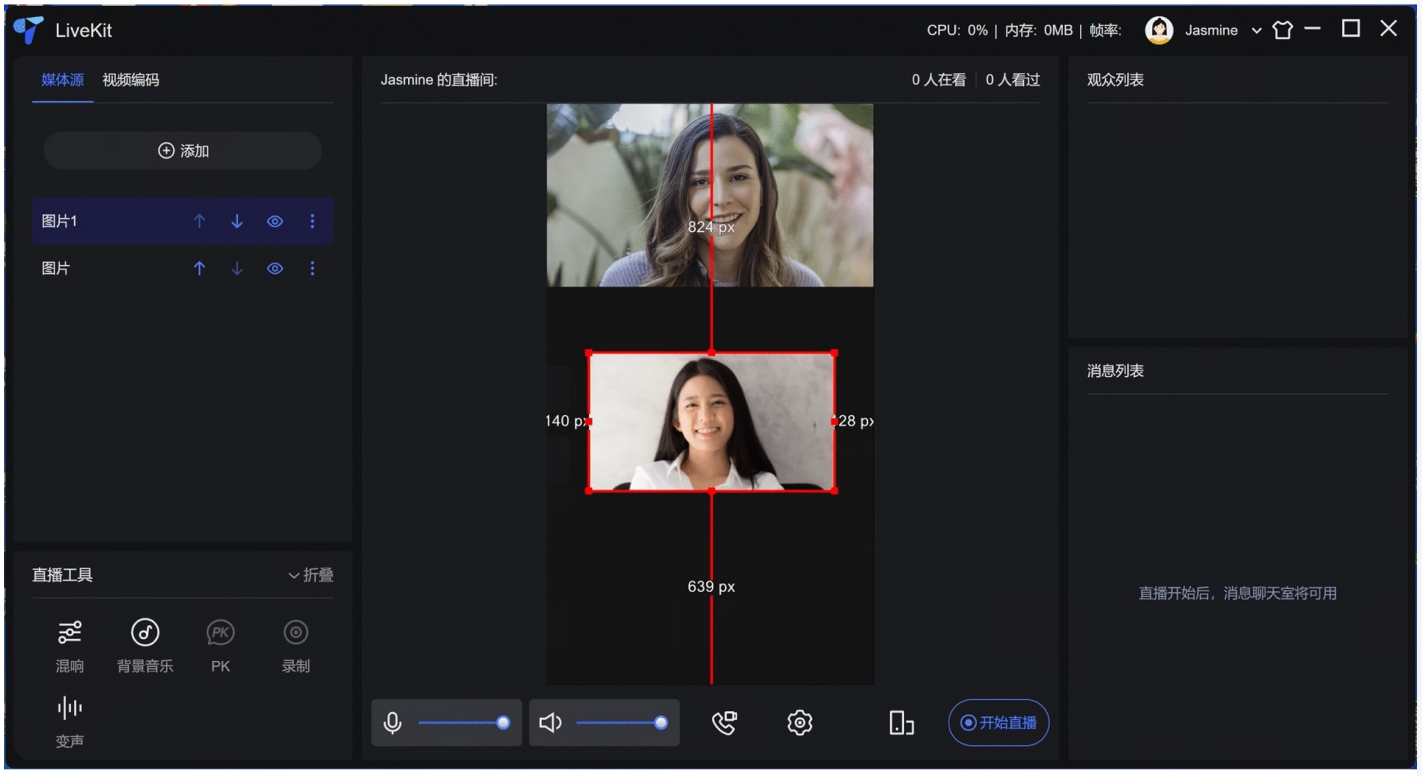


2. 本地视频和在线视频混流



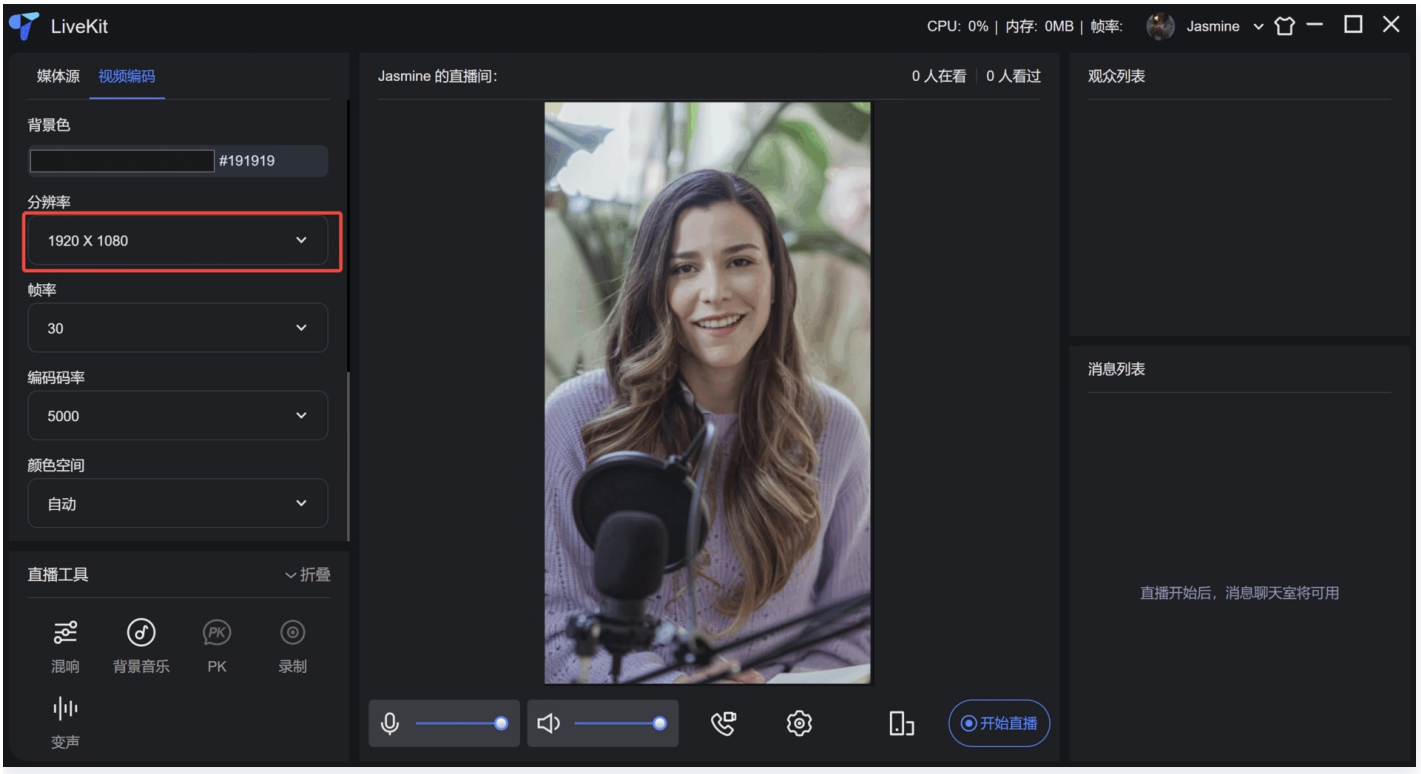
排版与编辑

功能说明：画面源可任意移动、缩放、吸附到边缘、调整显示层级、旋转与镜像。



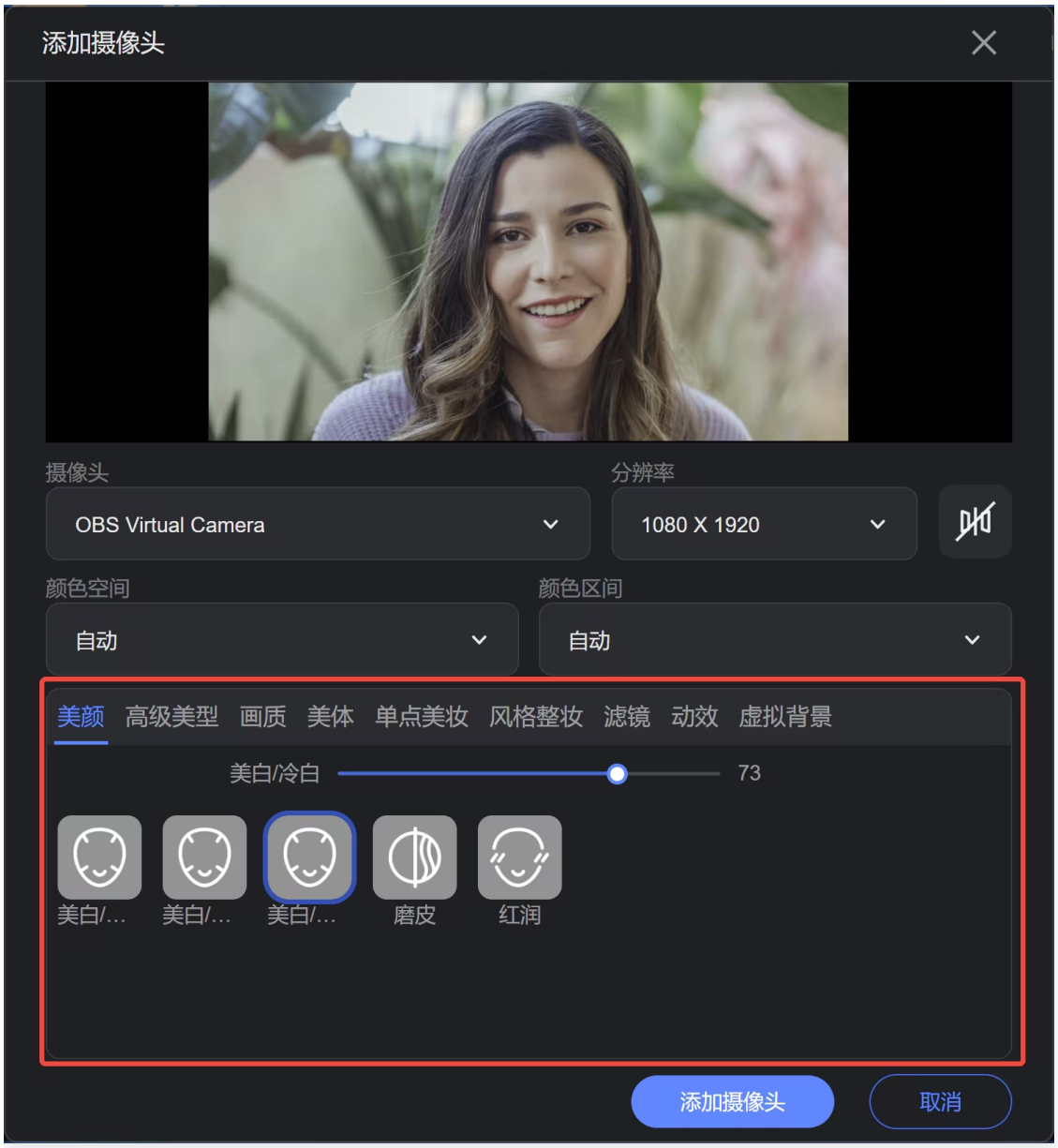
2K 高清直播

- 功能说明：支持 2560×1440 分辨率本地混流与推流，需在高编码档位与充足带宽/硬件下启用以获得最佳画质。
- 分辨率与帧率范围：支持 720p/1080p/2K 视频，最高 60 fps。



高级美颜

功能说明：提供丰富美颜与美妆项，可对每路视频流独立配置，支持虚拟背景替换与滤镜叠加。

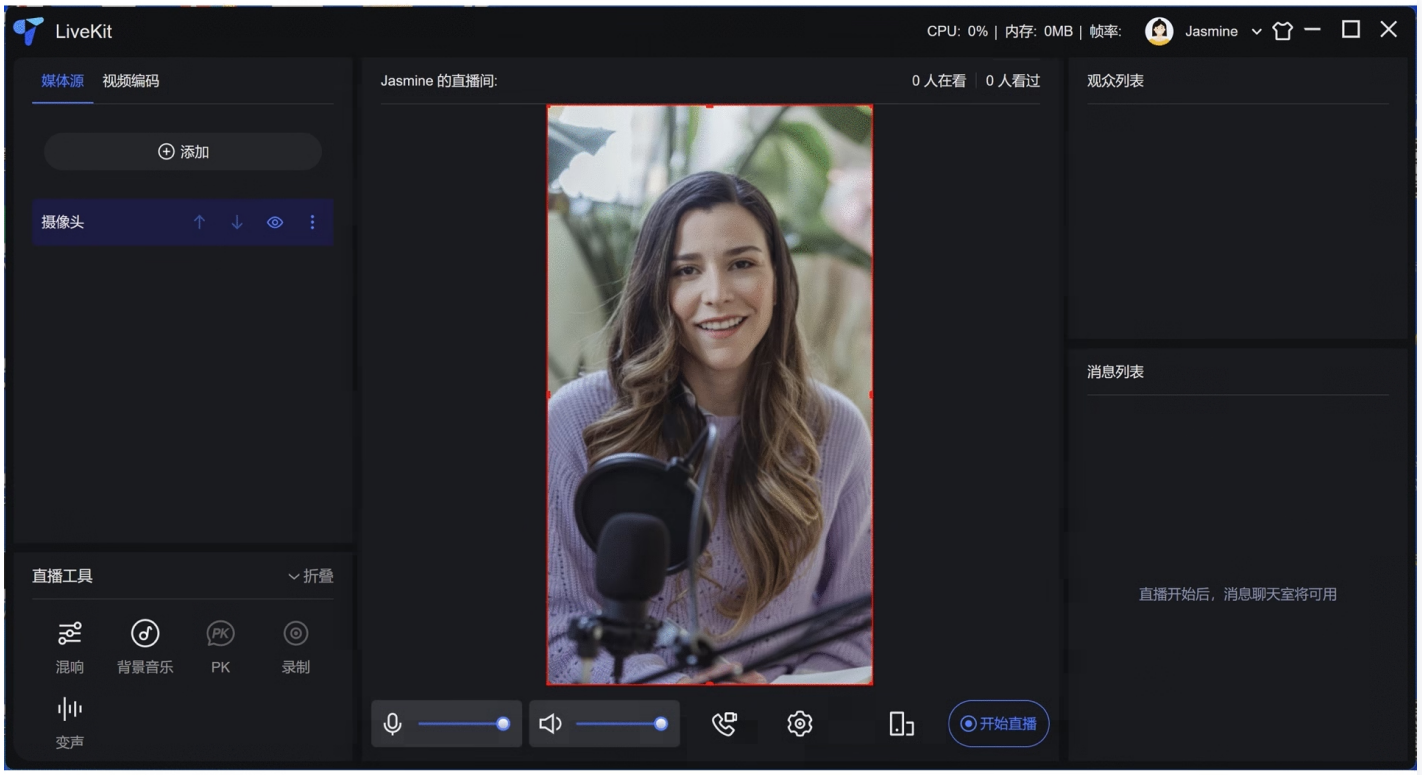


说明：

若您需要高级美颜能力，请单独购买腾讯特效 SDK 套餐：美颜特效移动端/PC 端计费说明 。

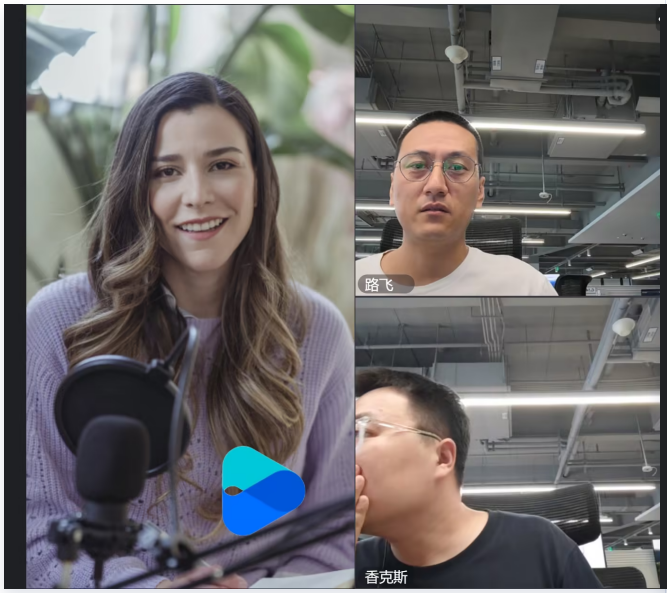
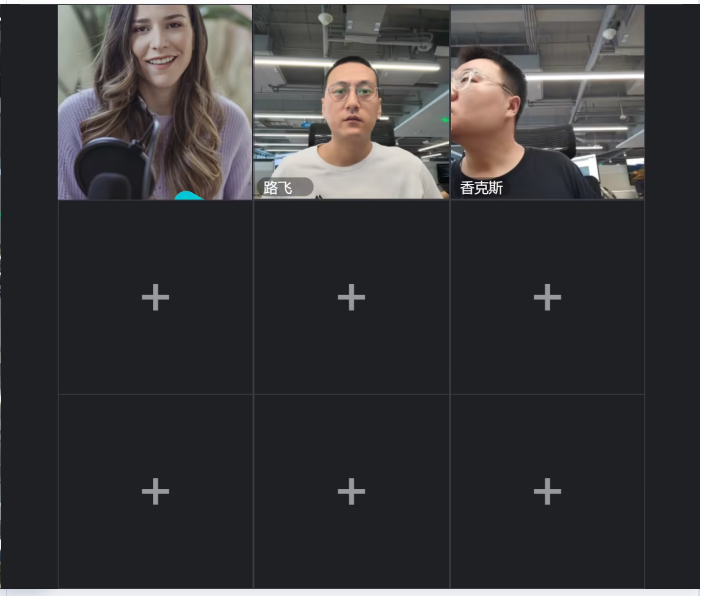
音频特效

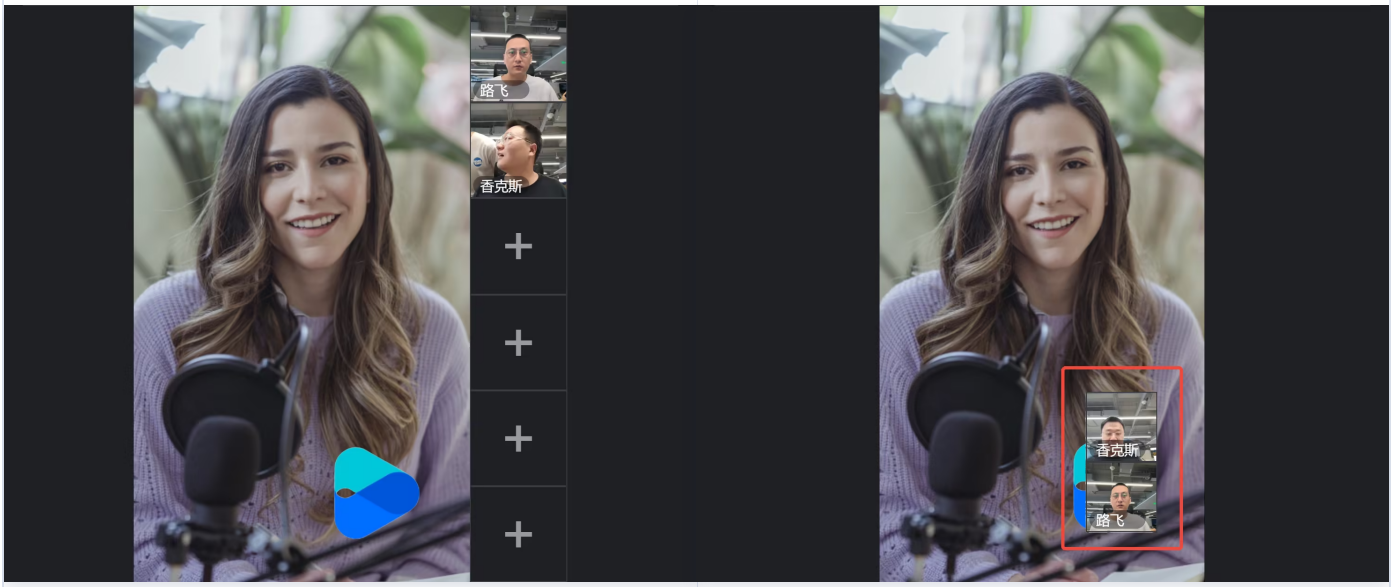
功能说明：支持 13 种变声与 11 种混响效果，并支持本地或在线背景音乐播放（请确保音乐版权合规）。



直播连线与布局

- 竖屏模式：支持动态九宫格、固定九宫格、动态浮窗、固定浮窗四种布局，并支持自定义布局。

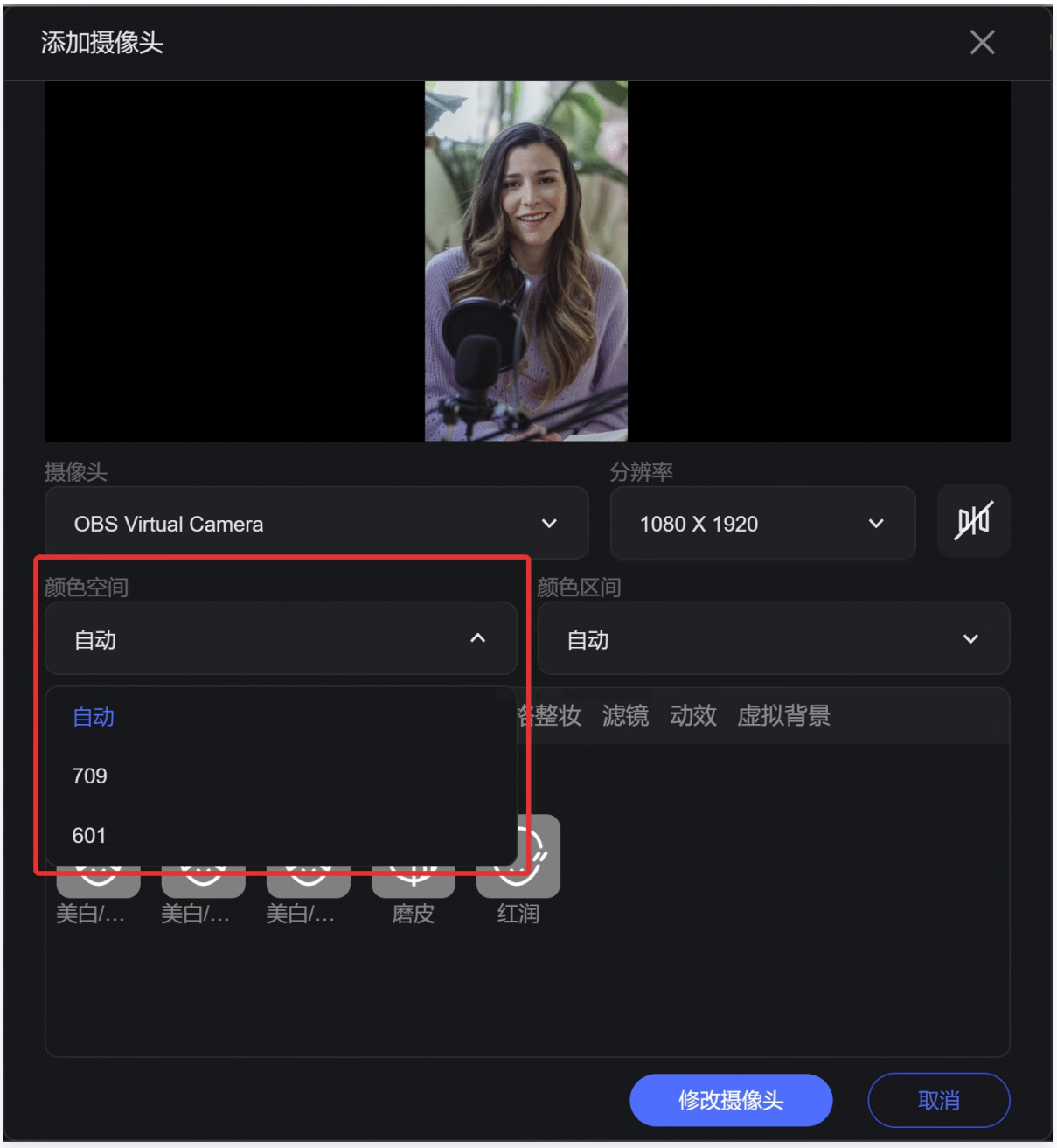
动态九宫格	固定九宫格
	
固定浮窗	动态浮窗



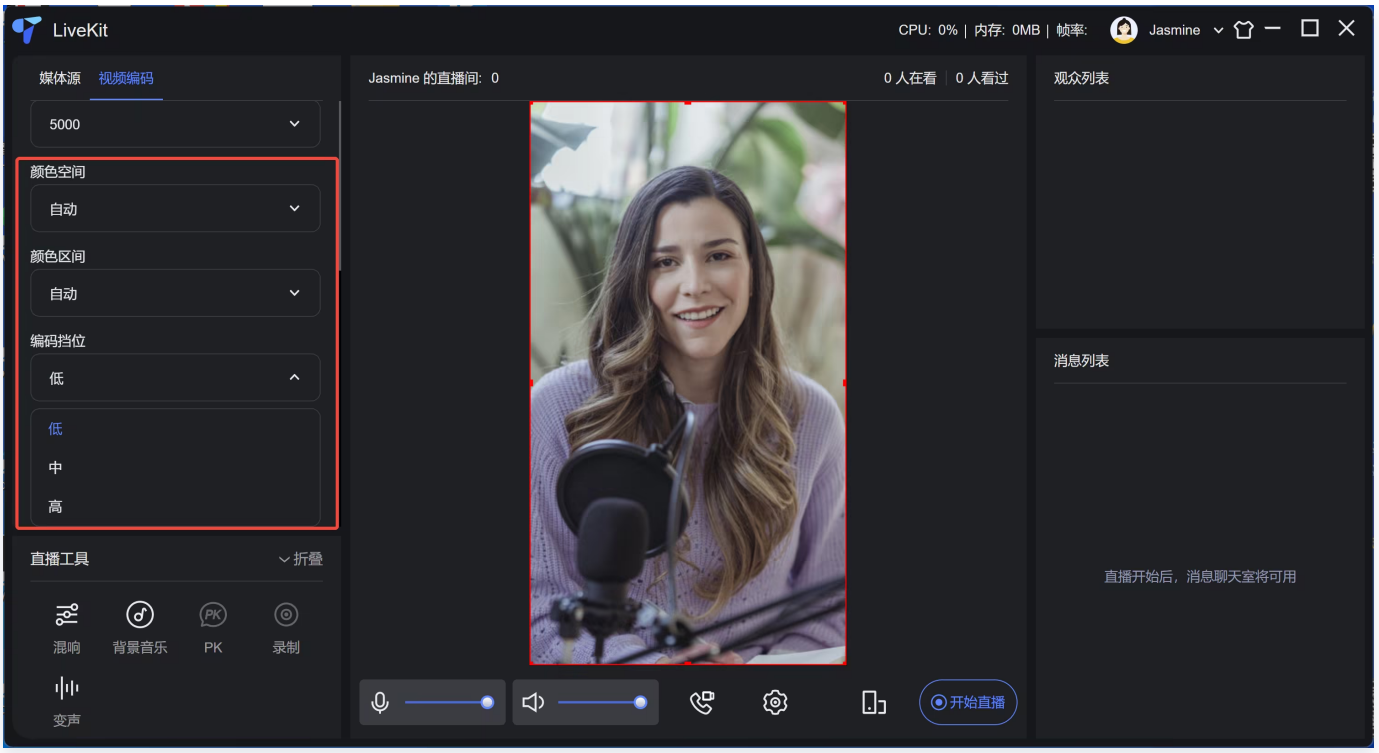
- 横屏模式：
  - 1v3 连麦：主播可开启摄像头、麦克风。上麦观众仅支持开启麦克风，不支持开启摄像头。

视频高级设置

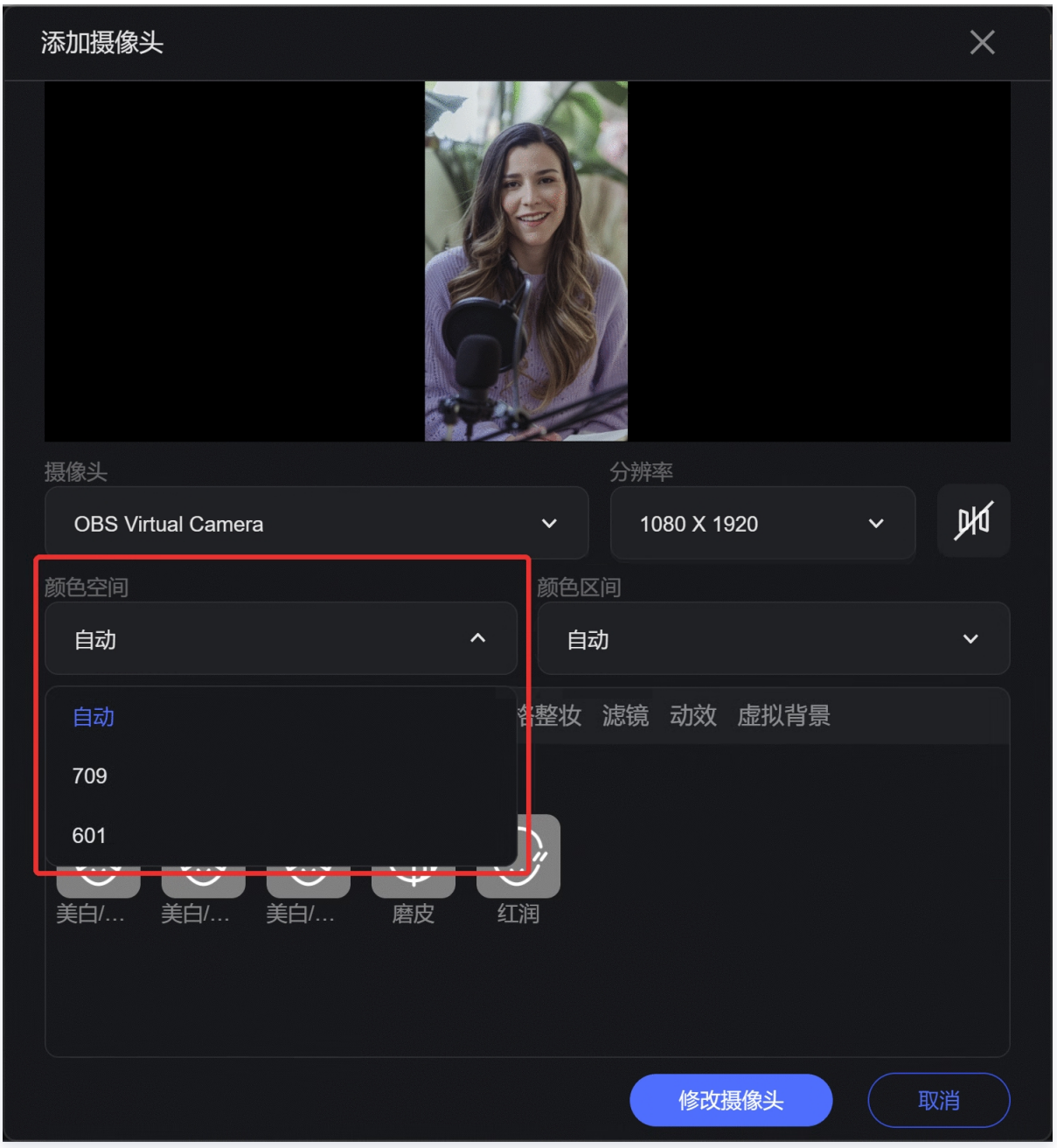
- 摄像头采集参数：支持设置采集的颜色空间与颜色区间（YUV 类），以适配不同编码/显示需求。



- **视频编码设置：支持设置编码颜色空间、颜色区间与编码档位。高编码档位可提升画质，但会增加 CPU/GPU 负载和网络带宽需求。**



- **视频编码设置：支持设置编码颜色空间、颜色区间与编码档位。高编码档位可提升画质，但会增加 CPU/GPU 负载和网络带宽需求。**



可选：对接封面上传服务（upload-server）

如需直播封面上传能力，可按以下步骤配置：

1. **准备环境文件**

```
cp upload-server/.env.example upload-server/.env
```

2. **配置存储服务：**在 `upload-server/.env` 中配置存储服务（ COS 或自定义对象存储）。
   - **STORAGE_PROVIDER：**可配置 cos | custom | oss。
   - **COS_SECRET_ID：**开发者拥有的项目身份识别 ID，用于身份认证，可在 API 密钥管理 页面获取。
   - **COS_SECRET_KEY：**开发者拥有的项目身份密钥，可在 API 密钥管理 页面获取。

- **COS_BUCKET**：存储桶，COS 中用于存储数据的容器。有关存储桶的进一步说明，请参见 存储桶概述 文档。
- **COS_REGION**：地域信息，枚举值可参见 可用地域 文档，例如：ap-beijing、ap-hongkong、eu-frankfurt 等。

3. **安装并启动 upload-server：**

```
npm run upload-server:bootstrap
npm run upload-server:standalone
```

4. **验证服务可用性：**
   - `http://127.0.0.1:3071/api/test`
   - `http://127.0.0.1:3071/api/upload/config`

> **说明：**
> 如 upload-server 不可用，界面会自动回退为手动输入封面 URL，渲染进程默认请求 `http://127.0.0.1:3071` 。如果图片上传服务部署在远端，请参考下一步设置 `VUE_APP_UPLOAD_SERVER_BASE_URL` 。

5. **配置上传服务地址：**

```
# 本次启动临时生效
VUE_APP_UPLOAD_SERVER_BASE_URL=https://your-upload-domain npm run start
# 本次构建临时生效
VUE_APP_UPLOAD_SERVER_BASE_URL=https://your-upload-domain npm run build
```

快速接入

步骤1：环境配置及开通服务

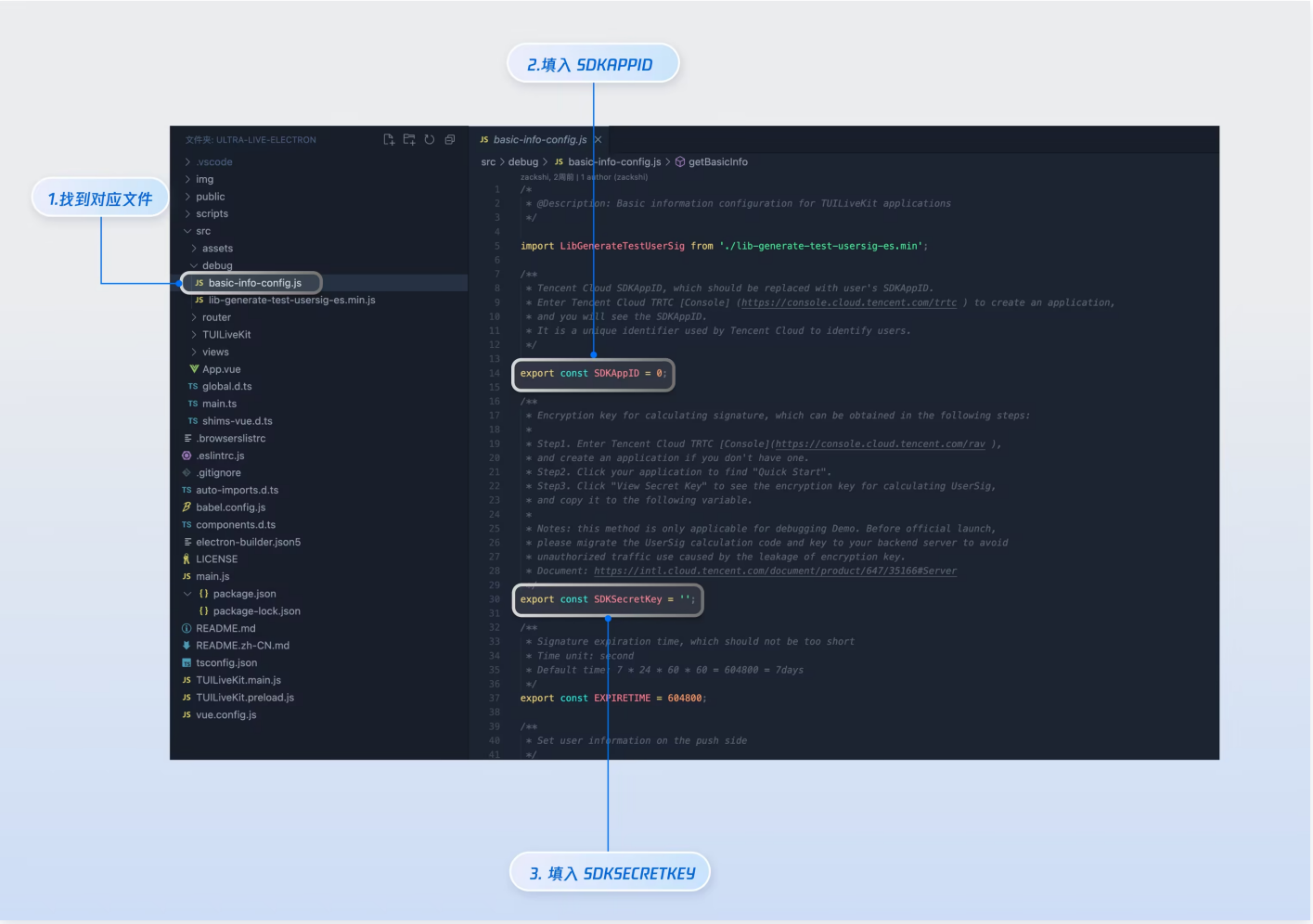
在进行快速接入之前，您需要参考 准备工作（Electron），满足相关环境配置及开通对应服务。

步骤2：下载项目

```
git clone https://github.com/Tencent-RTC/ultra-live-electron.git
cd ultra-live-electron
npm install
```

步骤3：配置相关服务信息

您首先需要参考 准备工作 开通对应的服务，接着在 Demo 中打开 `ultra-live-electron/src/debug/basic-info-config.js` 文件，输入准备工作中激活服务时获得的 SDKAppID 和 SDKSecretKey。



步骤4：运行项目并进行开播

通过如下命令运行当前项目：

```
npm run start
```

运行项目后，您可以通过如下步骤开启第一场直播：

1. **添加视频源：**点击界面左侧的添加按钮，选择您需要添加的视频源（摄像头、屏幕分享、图片等）。
2. **调整画面布局：**鼠标拖拽移动和调整视频源大小，选中视频源（出现黄色边框）后右键单击，可进行旋转或层级调整。
3. **开始推流：**点击界面上的**开始直播**按钮。按钮变为**结束直播**时说明已成功开播。

> **说明：**

至此，您已完成了完整的直播推流助手跑通流程，可以随时开启您的直播。另外，该桌面端软件仅支持推流，若您需要观看直播，请使用手机端 App 或者 Web 浏览器，您需要运行 iOS、Android 或者 Web 端的 TUILiveKit，在直播列表中找到并进入您创建的直播间即可。您可以参考如下文档链接至观看页面进行集成或通过 Web Player 在线体验 立即验证观看效果。

平台	文档链接
Web	观看页面
iOS	观看页面
Android	观看页面

打包构建

构建完成后，安装包位于 `release` 目录下。

Windows:

```
npm run pack:win64
```

Mac:

```
npm run pack:mac-x64
```

自由定制

您可以在现有代码基础上对页面样式、布局、功能进行修改。所有页面代码位于 `src` 目录中，可参考源码结构进行扩展。

替换 logo

1. 顶部主 Logo（主界面左上角）
   - 直接修改 `LogoIcon.vue` 的 SVG path 为您的品牌 SVG。
   - 如果要修改右侧文字 LiveKit，您需要到 `LiveHeader/index.vue` 修改。
2. 应用图标
   - 替换 `public/favicon.ico`（网页图标）。
   - Windows 安装包图标， 替换 `img/tuiroom.ico`， 配置位置 `electron-builder.json5` 里 `win.icon`， `nsis.installerIcon`， `nsis.uninstallerIcon`。
   - Mac 安装包图标，electron-builder.json5 下配置中设置 `mac.icon`，建议使用 `.icns` 文件（例如 `img/tuiroom.icns`），并将 `mac.icon` 指向该文件。

3. 验证步骤
   - `npm run start`（查看登录页和主界面 Logo 是否正确）。
   - `npm run pack:win64`（查看安装包图标是否正确）。
   - `npm run pack:mac-x64`（查看安装包图标是否正确）。

常见问题

运行时提示“未购买套餐包或者已欠费”？

请前往 购买套餐包 或 开通体验版 。首次开通后可能存在服务生效延迟，建议等待约 5 分钟后重试。

运行时提示“套餐包正在配置中暂未生效”？

该提示通常表示服务正在生效中。请等待约 5 分钟后再次尝试创建直播间。

创建直播间时提示“直播间 ID 只能包含字母、数字、下划线”？

请修改直播间 ID，仅使用字母、数字、下划线，然后重新开始直播。

登录失败 `code: 2025`（重复登录）？

请先退出其他终端的同账号登录，或更换用户 ID 后重试。

`npm install` 失败?

请先执行 `node -v` 检查 Node.js 版本是否为 18.0.0 或更高版本，然后重新执行安装命令。

登录失败或无法完成鉴权?

请检查 `SDKAppID` 与 `SDKSecretKey` 是否填写正确，并确认二者来自同一个应用，且未包含多余空格或其他字符。

主播开播（OBS Studio）

最近更新时间：2026-05-11 16:56:11

本文介绍如何通过 OBS Studio 将外部媒体流推送到 LiveKit 直播间。该方案适用于游戏直播、电竞解说或播放本地视频文件等场景。

> **说明：**
> Live OBS 开播能力已全面升级，使用该功能需开通 Live 包月套餐 **体验版、多人连麦版**或**大规模直播版**套餐。为了获得更稳定的推流表现和更低的延迟体验，建议在正式开播前，联系 技术支持 进行 **OBS 专项配置优化**。

获取 OBS 推流地址

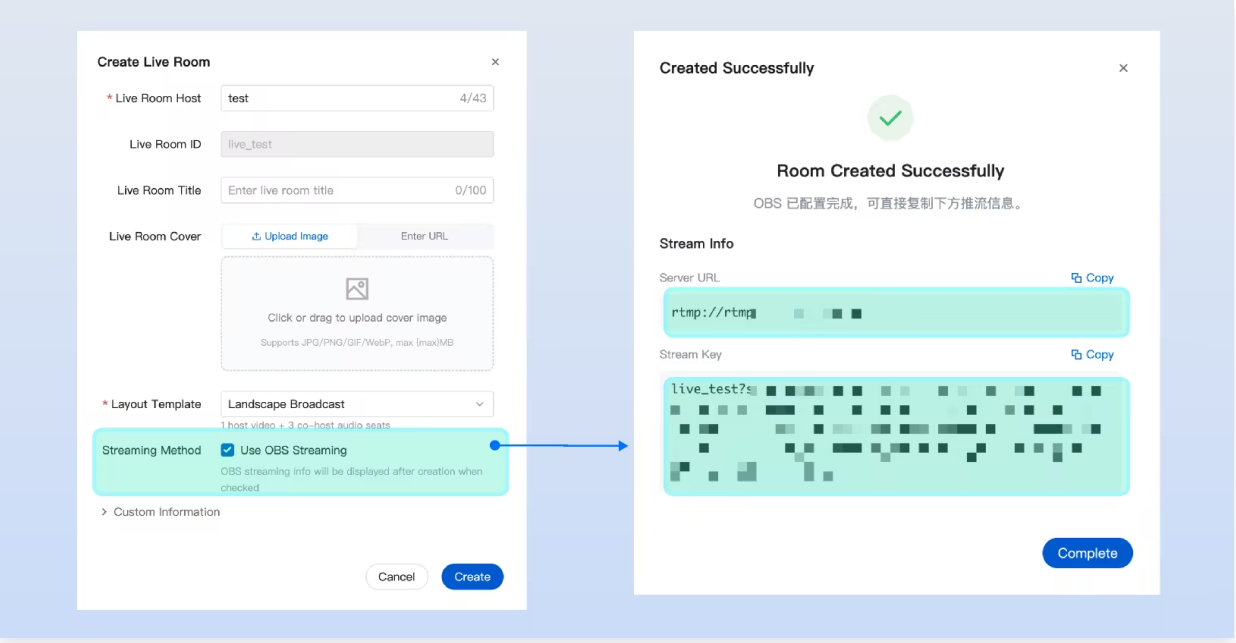
为了满足不同的开发需求，Live 提供了两种获取 OBS 推流地址的方案。

方案一：直播管理后台

适用于绝大多数生产环境和开发测试。通过可视化界面管理，能够降低开发维护成本，快速上线。

操作步骤

使用 直播管理系统（Vue3）点击新建直播间，在**推流方式**中勾选“**使用 OBS 推流**”。创建成功后将自动展示 **服务器地址 (Server URL)** 和 **推流码 (Stream Key)**。



方案二：服务端 API 搭建

适用于对直播业务有高度定制化需求、希望在自有系统中实现完全闭环逻辑的场景。

步骤 1：创建房间与机器人

- 调用 RESTful API 创建直播间 。
- 调用 RESTful API 创建机器人，推荐将房主 ID 设为机器人 UserID。

步骤 2： 实现机器人上麦

Live 观众端默认拉取麦上主播流，需调用 RESTful API 机器人上麦 接口模拟主播上麦。

步骤 3：生成 RTMP 推流地址

无需调用额外接口，直接在您的服务端按照下述规则拼接推流地址：

```
rtmp://rtmp.rtc.qq.com/push/{房间号}?sdkappid={AppID}&userid={用户ID}&usersig={签名}
```

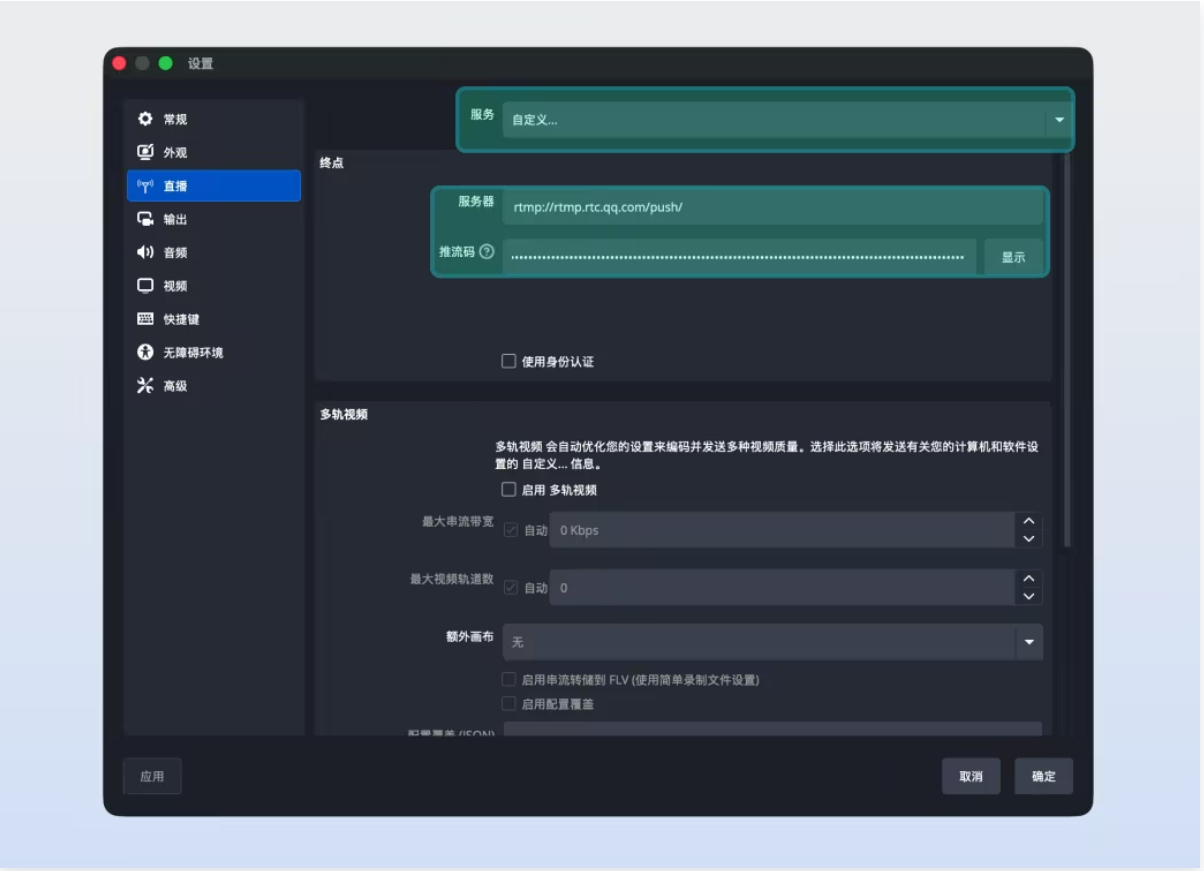
- **主域名：** `rtmp.rtc.qq.com` ，备用域名为 `rtmp.cloud-rtc.com` ，如果主域名解析有问题，可使用备用域名。
- **房间号：**对应步骤1创建的房间号 `RoomID` 。
- **用户 ID：**对应步骤2创建的机器人 `UserID` 。
- **userSig：**需在有效期内，规则请参考 UserSig 相关文档 。

配置 OBS 软件

获取到推流地址后，请按照以下步骤配置 OBS Studio。

步骤 1：设置服务器信息

- 通过底部工具栏的**控件按钮** > **设置**按钮进入设置界面。
- 点击**直播**进入服务器信息配置。
  - **服务：**选择 **自定义**。
  - **服务器和推流码配置：**从 获取 OBS 推流地址 步骤中获取。



步骤 2：优化参数配置

为了确保观众端能获得低延迟、高清晰度的观看体验，请务必完成以下两个维度的配置。

1. 设置视频参数

进入**设置** > **视频**，通常可参考以下配置，游戏直播建议将帧率提升至 30 fps 以保证画面流畅：

分辨率	帧率 (fps)	推荐码率
1920x1080	24 fps	3000 kbps
1280x720	24 fps	2500 kbps

2. 设置输出参数

进入**设置** > **输出**，将**输出模式**切换为**高级**。RTMP 后台不支持传输 B 帧，请按以下参数调整：

参数项	推荐值	配置描述
编码器	`x264` 或 `NVIDIA NVENC`	优先选硬件编码以降低 CPU 压力。
关键帧间隔	`1` 或 `2`	**严禁设为 0**，否则拉流极慢。
CPU 使用预设	`ultrafast`	降低编码耗时，减少卡顿。

配置 (Profile)	`baseline`	确保去除 B 帧以兼容 RTMP 后台。
微调 (Tune)	`zerolatency`	针对流媒体直播优化，大幅降低端到端延迟。
x264 选项	`threads=1`	限制编码线程数，提升流稳定性。

步骤 3：设置高级选项

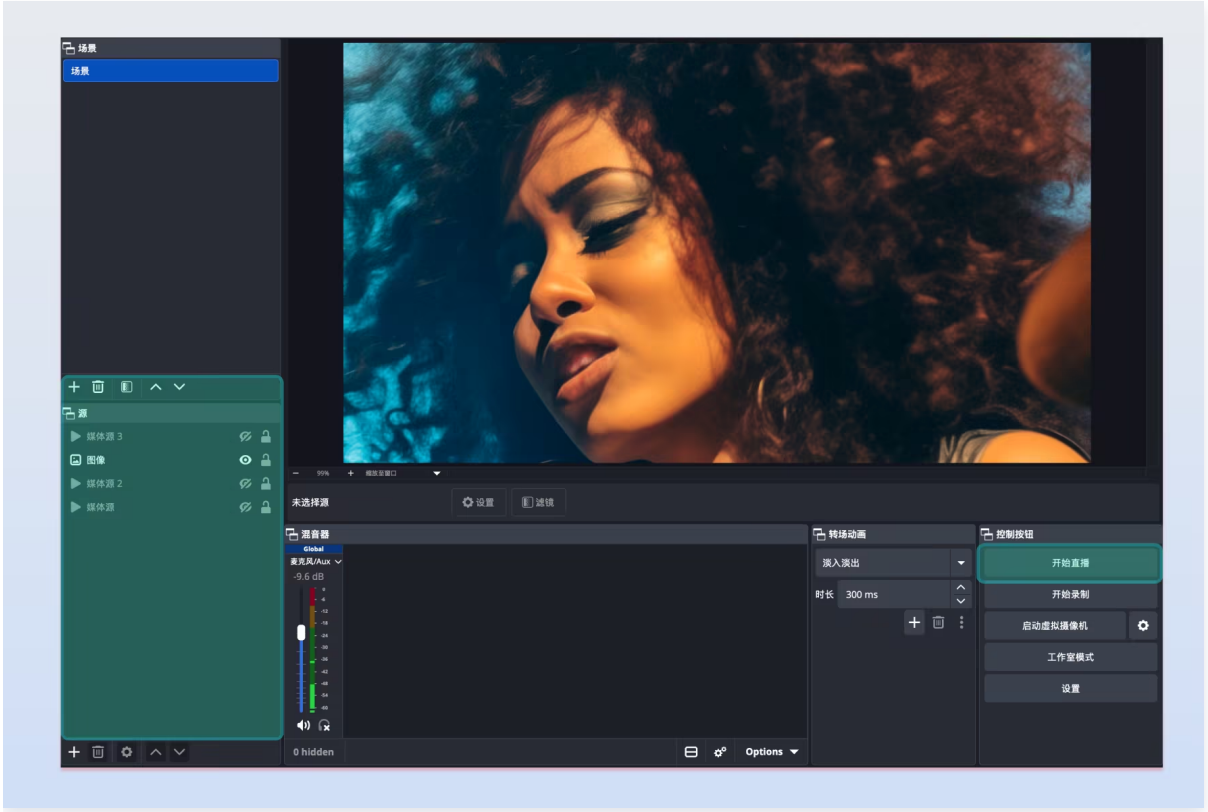
进入**设置 > 高级**，确保网络波动时能快速恢复。

1. **串流延迟**：**不启用**。启用该项会显著增加端到端延迟。
2. **自动重连**：启用。
   - **重试延迟时长**：建议设为 2秒。
   - **最大重试次数**：建议设为 20次。

开始 OBS 直播

开始直播

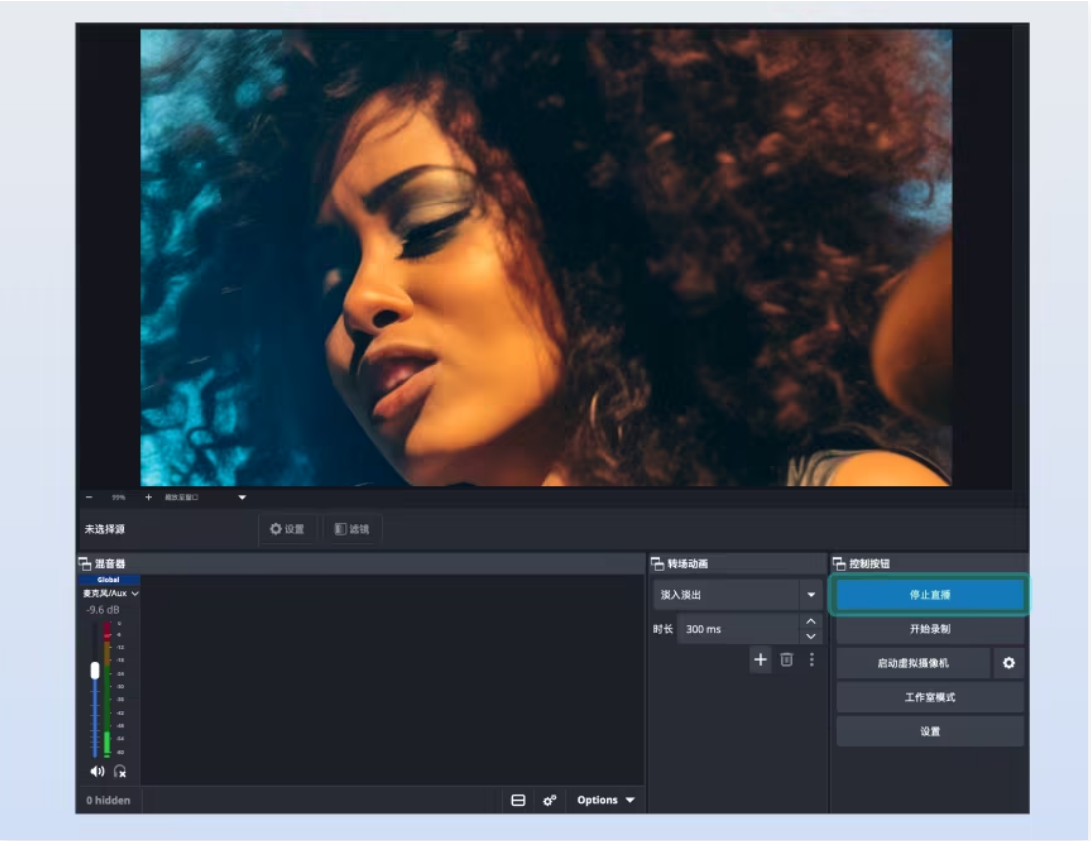
在 OBS 来源栏添加“窗口采集”或“视频采集设备”，预览确认无误后，点击**开始直播**。



结束直播

步骤 1：停止 OBS 推流

在 OBS Studio 主界面，点击右下角的 **停止直播**。



步骤 2：关闭直播间

为了确保房间资源及时释放并停止计费，请根据**获取 OBS 推流地址**的方式选择对应的操作：

- **直播管理后台**：在管理后台的直播列表页，点击对应的直播间并选择**结束直播**。
- **服务端 API 搭建**：通过服务端 RESTful API 解散直播间 。

常见问题

当主播使用 OBS 第三方工具进行 RTMP 推流时，若发生网络波动导致断流，观众端画面是否会自动恢复?

LiveKit 现已全面升级流媒体调度方案，针对推流中断场景提供了更智能的保障机制，请务必在正式开播前联系 技术支持 进行 **OBS 专项配置优化**：

- **自动恢复机制**：在网络波动导致短时间断流时，系统将自动保持混流通道。只要主播在规定时间内恢复推流，观众端即可实现无感自动重连，无需开发者调用任何后端接口。
- **稳定性增强**：新版方案大幅延长了混流节点的等待阈值，有效解决了旧版本中因断流超过 2 分钟导致混流中断、观众端持续黑屏的问题。

观看方式
观众观看（Android）

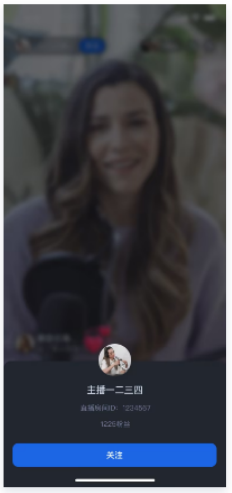
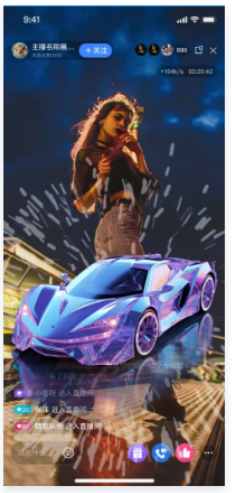
最近更新时间：2026-07-02 11:05:36

`TUILiveKit` 的观众观看页为用户提供了丰富且便捷的直播观看与互动功能，能快速集成到您的应用中，满足观众多样化的需求。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的观众观看方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建观众端界面，请参考 观众观看 (Core SDK)。

功能概览

- **直播观看：**清晰、流畅观看主播实时直播画面。
- **互动连麦：**申请连麦，与主播进行音视频互动。
- **直播信息：**查看直播间标题、简介，及在线观众列表等信息。
- **直播互动：**送礼物（有动画特效，主播接收提示）、点赞（有动画及实时统计）、发弹幕互动。

直播观看	互动连麦	直播信息	直播互动
			

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. 添加观众观看页面

在您的观众观看页 `Activity` 中 初始化添加 `AudienceView` 观众观看视图：

Kotlin

```
import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import com.trtc.uikit.livekit.features.audienceview.AudienceView
import
com.trtc.uikit.livekit.features.audienceview.AudienceViewDefine.Audience
ViewListener
import io.trtc.tuikit.atomicxcore.api.live.LiveInfo

class AudienceActivity : AppCompatActivity() {
    lateinit var audienceView: AudienceView
    lateinit var listener: AudienceViewListener

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // 1. 创建 AudienceView
        audienceView = AudienceView(this)
        val roomId = "live_1236666"
        // 2. 初始化 AudienceView
        audienceView.init(this, roomId)

        // 3. （可选）添加 AudienceContainerView 事件监听
        listener = object : AudienceViewListener {

            override fun onLiveEnded(roomId: String, ownerName: String,
ownerAvatarUrl: String) {
                finishAndRemoveTask()
            }

            override fun onClickFloatWindow() {
                // 画中画按钮点击事件，您可以按照您的业务需求处理该事件，此处代码示
例：点击画中画按钮后，将当前页面转换为画中画模式
```

```
                enterPictureInPictureMode()
            }

            override fun onClickCloseButton(liveInfo: LiveInfo) {
                Log.i("TAG", "onClickCloseButton liveInfo: $liveInfo")
                finishAndRemoveTask()
            }
        }
        audienceView.addListener(listener)
        // 4. 将 AudienceContainerView 添加到界面
        setContentView(audienceView)
    }

    override fun onDestroy() {
        // 5. （可选）若您添加了事件监听器，则需要在页面销毁时移除监听器
        audienceView.removeListener(listener)
        super.onDestroy()
    }
}
```

步骤 4. 跳转到观众观看页面

在您需要点击跳转到观众观看页时，可参考如下代码示例：

Kotlin

```
fun startActivity(context: Context) {
    val intent = Intent(context, AudienceActivity::class.java)
    context.startActivity(intent)
}
```

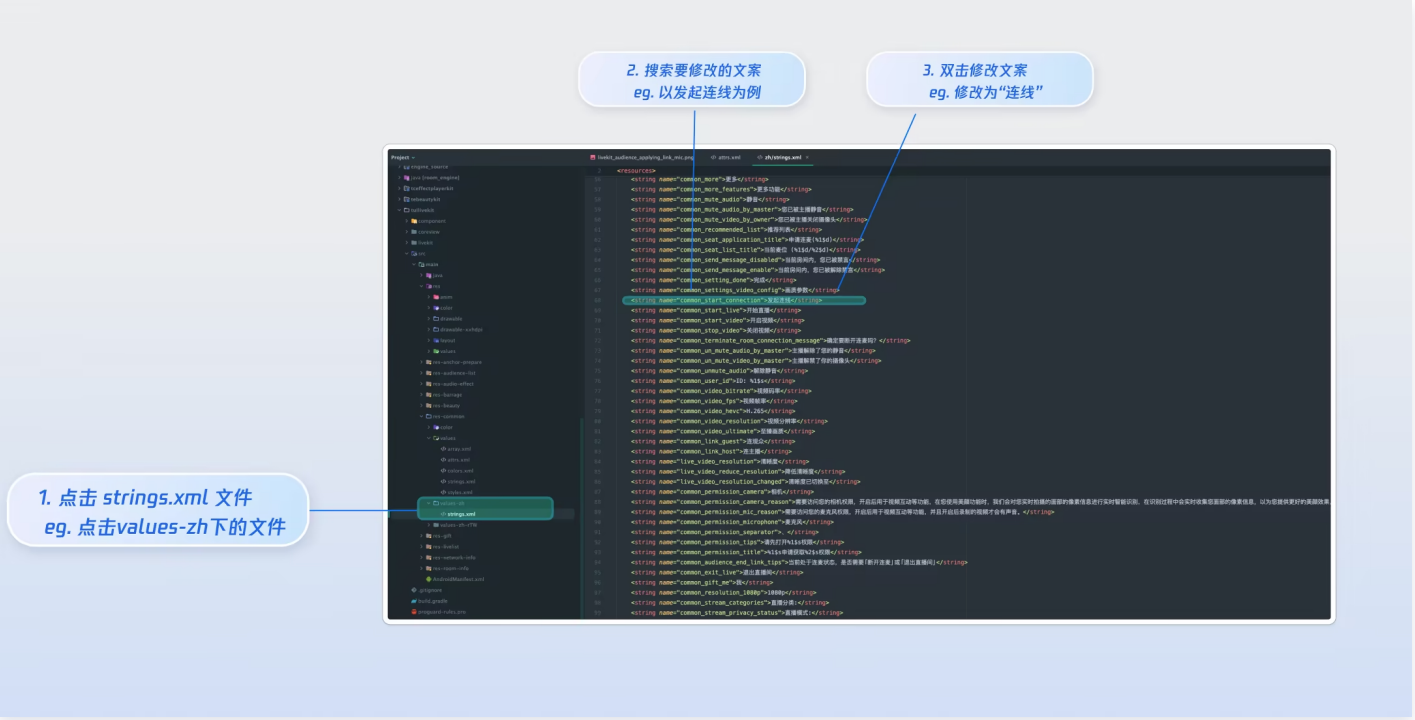
自定义您的界面布局

自定义观众观看页界面

- 参考 观众核心页面 进行观众界面定制，在 `AudienceView` 中修改 UI 样式，添加您的业务组件。
- 参考 调整视频直播挂件 进行视频位挂件定制，在 `AudienceView` 中修改视频位上的名称、头像挂件等 UI。

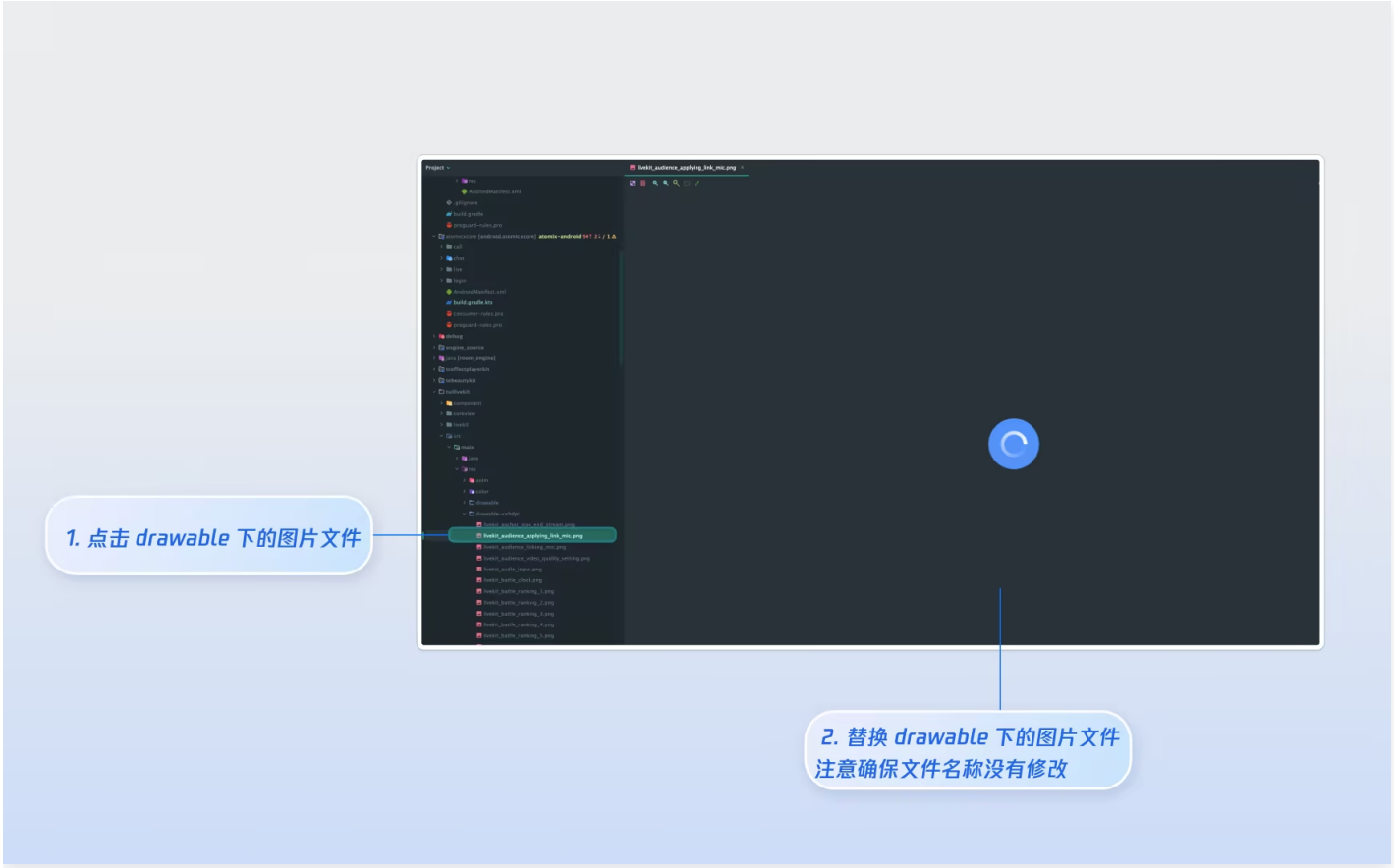
文案定制

`TUILiveKit` 使用 `Android` 通用的 `XML` 资源文件来管理 `UI` 所需的文案显示，您可以直接通过 `XML` 文件修改需要调整的文案：



图标定制

`TUILiveKit` 使用 `Android` 通用的 `drawable` 资源文件夹管理 `UI` 所需的图片资源，您可以通过替换资源文件快速修改自定义界面所需的图标，替换时请确保新的文件名称与原文件名称一致。



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
礼物系统	实现自定义礼物素材配置、计费系统对接、PK 场景互动等功能。	礼物系统

常见问题

观众选择视频连麦后，视频画面显示黑屏?

请前往**应用信息 > 权限 > 相机**，检查摄像头权限是否开启，可参考下图：

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）

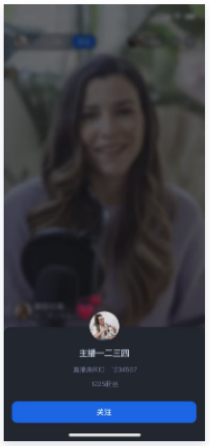
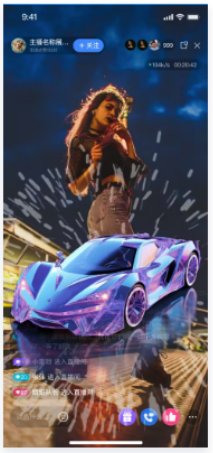
最近更新时间：2026-07-06 10:18:48

TUILiveKit 的观众观看页为用户提供了丰富且便捷的直播观看与互动功能，能快速集成到您的应用中，满足观众多样化的需求。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的观众观看方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建观众端界面，请参考 观众观看 (Core SDK)。

功能概览

- **直播观看：**清晰、流畅观看主播实时直播画面。
- **互动连麦：**申请连麦，与主播进行音视频互动。
- **直播信息：**查看直播间标题、简介，及在线观众列表等信息。
- **直播互动：**送礼物（有动画特效，主播接收提示）、点赞（有动画及实时统计）、发弹幕互动。

直播观看	互动连麦	直播信息	直播互动
			

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. 添加观众拉流页面

在您的观众视图控制器中 `YourAudienceViewController` 初始化添加 `AudienceView` 观众拉流视图：

Swift

```
import UIKit
import SnapKit
import TUILiveKit

// YourAudienceViewController 代表您观众观看页的视图控制器
class YourAudienceViewController: UIViewController {

    // 1.  声明 audienceView 作为成员变量
    private let audienceView: AudienceView

     // 2. 新增便利构造函数：
     //        - roomId: 直播房间id
    public init(roomId: String) {
        // 3. 初始化 AudienceView 组件
        self.audienceView = AudienceView(roomId: roomId)
        super.init(nibName: nil, bundle: nil)
    }

    @available(*, unavailable)
    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    public override func viewDidLoad() {
        super.viewDidLoad()
        // 4. 将 audienceView 添加到视图上
        view.addSubview(audienceView)
        audienceView.snp.makeConstraints { make in
            make.edges.equalToSuperview()
        }
    }
}
```

步骤 4. 导航路由到观众拉流页面

通常在 直播列表 或者 **点击主播开播推送通知** 时，您需要导航跳转到观众观看页，可参考如下代码示例：

Swift

```
// 1. 实例化您的观众观看视图控制器
let audienceVC = YourAudienceViewController(roomId: "您的直播房间 id")
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

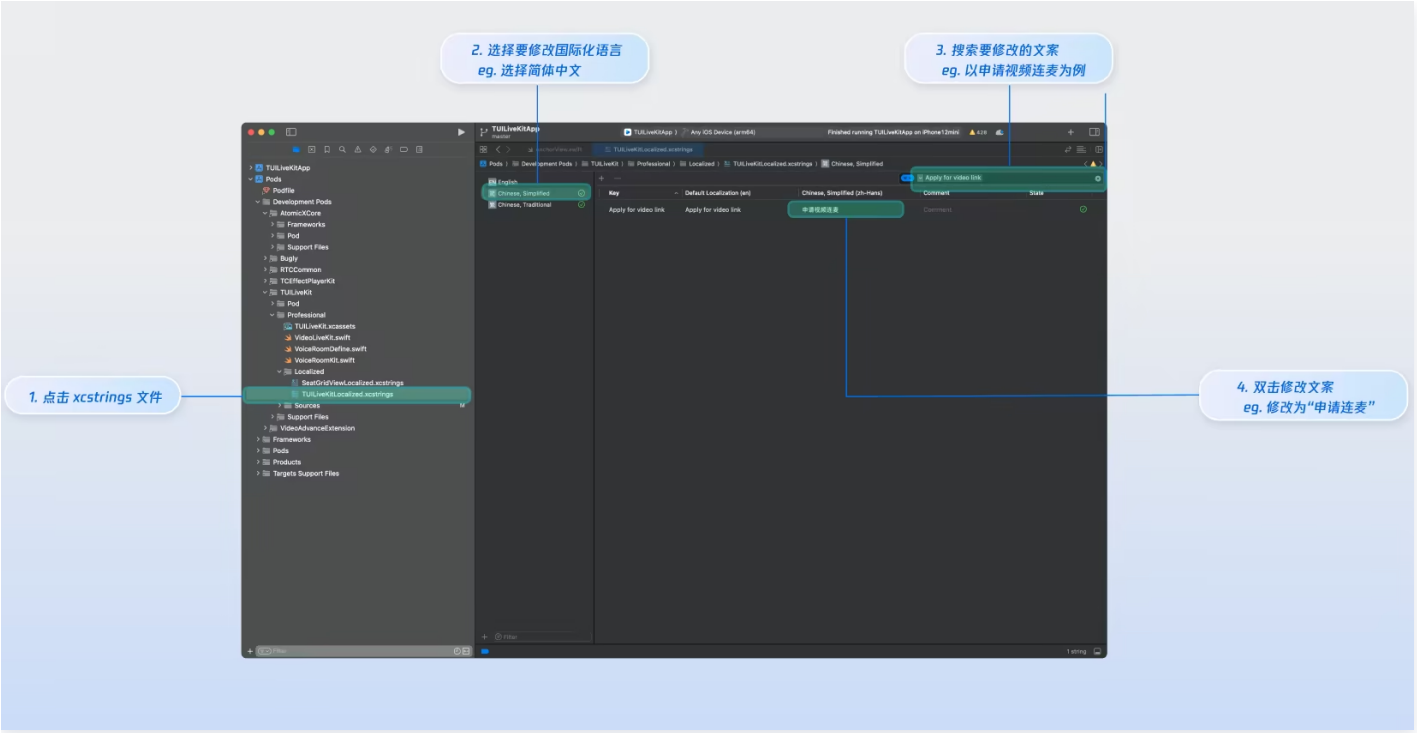
自定义观众观看页界面

- 参考 观众核心页面 进行观众界面定制，在 `AudienceView` 中修改 UI 样式，添加您的业务组件。
- 参考 调整视频直播挂件 进行视频位挂件定制，在 `AudienceView` 中修改视频位上的名称、头像挂件等 UI。

文案定制

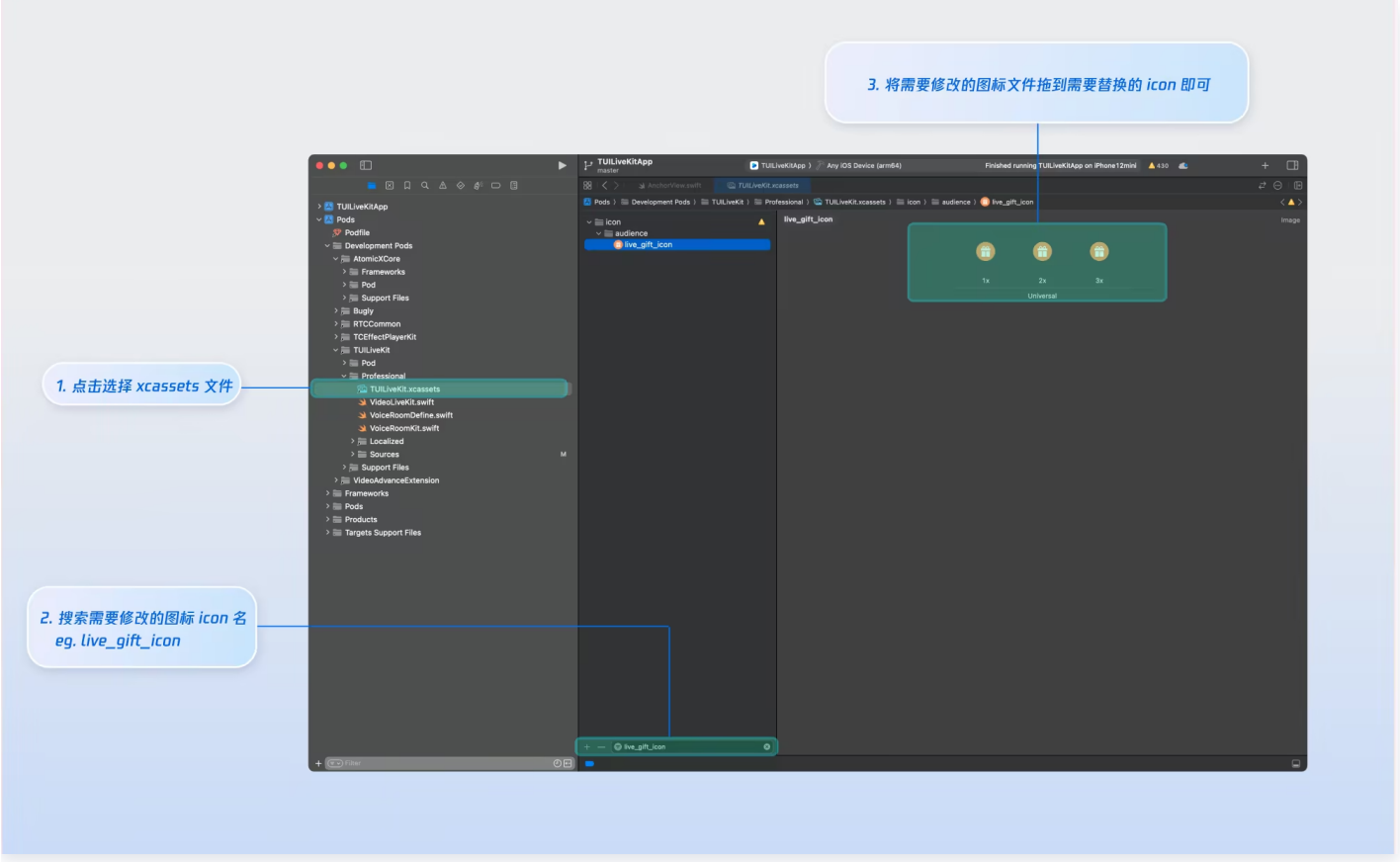
TUILiveKit 使用更为方便的 Apple Strings Catalog 工具来管理 UI 所需的文案显示，您可以很直观的通过 Xcode 视图化管理工具修改需要调整的文案：

> **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。



图标定制

TUILiveKit 使用 `TUILiveKit.xcassets` 管理 UI 所需的图片资源，您可以借助 Xcode 图形化工具快速修改自定义界面所需的图标。



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
礼物系统	实现自定义礼物素材配置、计费系统对接、PK 场景互动等功能	礼物系统

常见问题

观众选择视频连麦后，视频画面显示黑屏？

请前往**手机设置 > App > 相机**，检查摄像头权限是否开启，可参考下图：

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（Web Vue3 桌面浏览器）

最近更新时间：2026-04-20 17:34:02

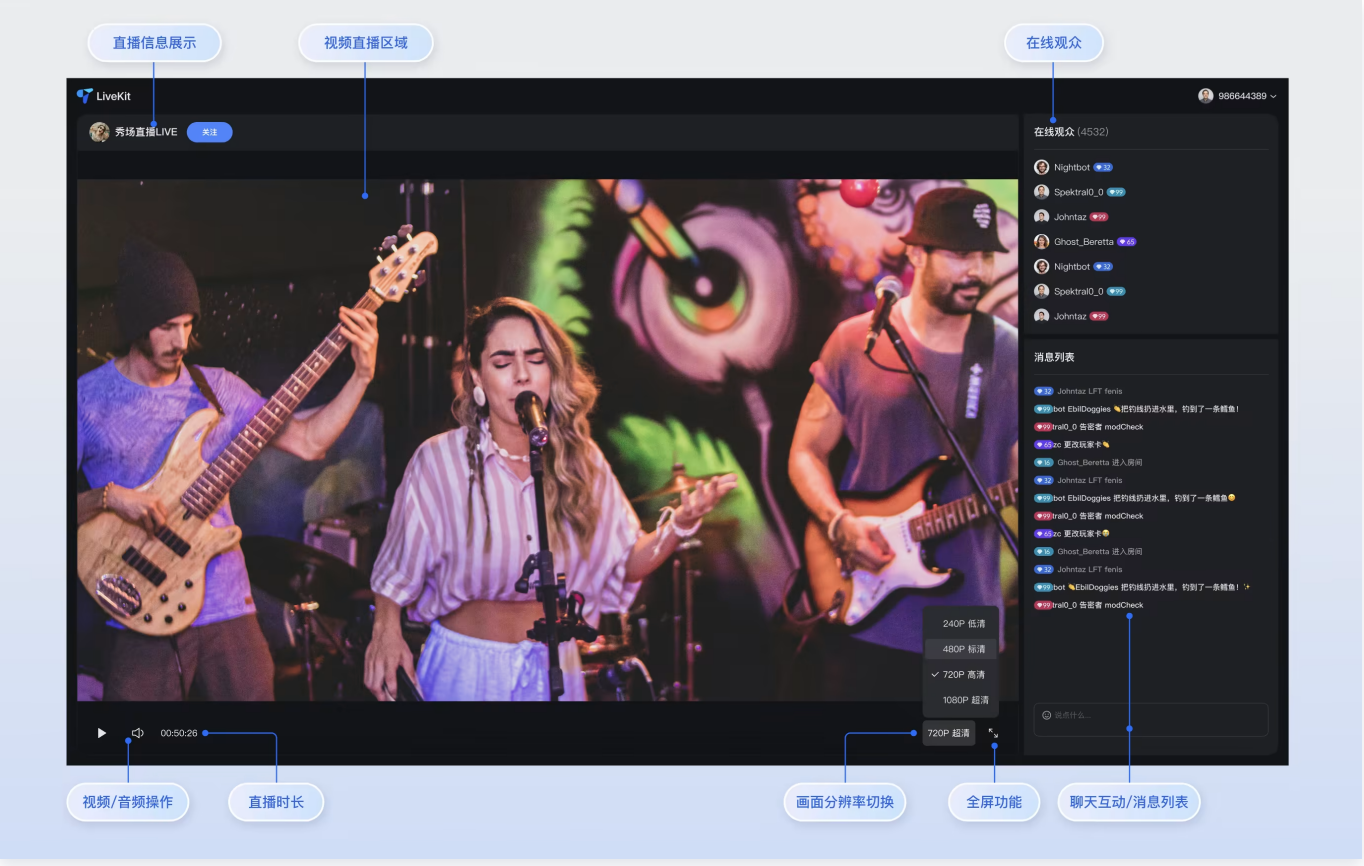
本文对 TUILiveKit Demo 中的**观看页面**进行了详细的介绍，您可以在已有项目中直接参考本文档集成我们开发好的观看页面，也可以根据您的需求按照文档中的内容对页面的样式，布局以及功能项进行深度的定制。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的观众观看方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建观众端界面，请参考 观众观看 (Core SDK)。

功能展示

观看页面提供默认行为和风格，但如果默认行为和样式不能完全满足您的需求，您也可以对 UI 进行自定义。图中显示的数字与特定功能列表中的类别相对应。其中主要包含直播**信息展示、视频直播区域、在线观众、音视频操作、直播时长、画面分辨率切换、全屏功能、聊天互动、消息列表等。**

- **直播视频播放**：高清流畅的直播体验。
- **弹幕互动**：实时弹幕交流。
- **观众列表**：查看在线观众信息。
- **全屏播放**：沉浸式观看体验。
- **H5 移动平台支持**：跨平台兼容。



> **说明：**
> 本文档主要介绍观众端的核心功能逻辑。如果您需要修改观众端的界面样式（如修改某些默认控件、修改图标和颜色、调整弹窗风格），请直接前往 调整界面风格 指南进行配置。

快速接入

步骤1：环境配置及开通服务

在进行快速接入之前，请参考 准备工作 集成组件并实现登录。

步骤2：安装依赖

您可以选择以下任一方式安装依赖：

npm

```
npm install tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
--save
```

pnpm

```
pnpm add tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
```

yarn

```
yarn add tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
```

步骤3：观看页面接入

在您的项目下创建 `live-player.vue` 文件，可直接复制如下代码至您的项目中集成**观看页面**。

> **注意：**
> 您可以直接复制如下代码至您的工程中集成示例工程，也可以访问 观众观看 地址，查看更加详细的源码内容。

```
<template>
  <UIKitProvider theme="dark">
    <div class="container">
```

```
      <!-- 直播核心区域 -->
      <section class="live">
        <header class="header">
          <IconArrowStrokeBack class="back-btn" size="20" />
          <Avatar :src="currentLive?.liveOwner.avatarUrl" :size="32"
class="avatar" />
          <span class="user-name">{{ currentLive?.liveOwner.userName ||
currentLive?.liveOwner.userId }}</span>
        </header>
        <LiveCoreView class="player" />
      </section>

      <div class="sidebar">
        <!-- 在线观众列表 -->
        <section class="audience">
          <header class="section-header">
            <h3> 在线观众 <span>({{ audienceList.length }})</span></h3>
          </header>
          <LiveAudienceList class="list" />
        </section>

        <!-- 消息列表 & 消息输入框 -->
        <section class="barrage">
          <header class="section-header">
            <h3>消息列表</h3>
          </header>
          <BarrageList class="list" />
          <BarrageInput class="input" height="48px" />
        </section>
      </div>
    </div>
  </UIKitProvider>
</template>

<script setup lang="ts">
import { onMounted } from 'vue';
import { LiveAudienceList, BarrageList, BarrageInput,
useLiveAudienceState, LiveCoreView, useLiveListState, Avatar,
useLoginState } from 'tuikit-atomicx-vue3';
```

```
import { UIKitProvider, IconArrowStrokeBack } from '@tencentcloud/uikit-
base-component-vue3';

const { audienceList } = useLiveAudienceState();
const { currentLive, joinLive } = useLiveListState();
const { login, setSelfInfo } = useLoginState();

const liveId = 'live_xxxx'  // 填入您要观看的直播间的ID

async function initLogin() {
  try {
    await login({
      sdkAppId: 0,      // SDKAppID, 可以参考步骤 1 获取
      userId: '',       // UserID, 可以参考步骤 1 获取
      userSig: '',      // userSig, 可以参考步骤 1 获取
    });
  } catch (error) {
    console.error('登录失败:', error);
  }
}

onMounted(async () => {
  await initLogin();
  await setSelfInfo({
    userName: '我的名字/昵称',     // 用户名
    avatarUrl: '',                // 头像 URL 地址
  });
  await joinLive({ liveId });
});

</script>

<style>html,body,#app{height:100%;width:100%;margin:0;padding:0;overflow
:hidden}:global(body){font-size:15px;line-height:1.6;text-
rendering:optimizeLegibility;}:global(*),:global(*::before),:global(*::a
fter){box-sizing:border-box;margin:0;}.container{display:grid;grid-
template-columns:1fr
320px;height:100%;width:100%;gap:16px;padding:16px;background:var(--bg-
color-default);box-sizing:border-
box;overflow:hidden;}.live{display:flex;flex-
```

```
direction:column;background:var(--bg-color-operate);border-
radius:12px;overflow:hidden;box-shadow:0 2px 8px var(--shadow-
color);}.header{display:flex;align-
items:center;gap:12px;padding:16px;border-bottom:1px solid var(--stroke-
color-primary);}.back-btn{cursor:pointer;color:var(--text-color-
tertiary);transition:color 0.2s;}.back-btn:hover{color:var(--text-color-
link-hover);}.avatar{border:1px solid var(--uikit-color-white-7);}.user-
name{color:var(--text-color-primary);font-
weight:500;}.player{flex:1;background:var(--uikit-color-black-1);min-
height:0;}.sidebar{display:flex;flex-
direction:column;gap:16px;height:100%;overflow:hidden;}.audience{display
:flex;flex-direction:column;background:var(--bg-color-operate);border-
radius:12px;overflow:hidden;box-shadow:0 2px 8px var(--shadow-
color);flex:1;min-height:0;}.barrage{display:flex;flex-
direction:column;background:var(--bg-color-operate);border-
radius:12px;overflow:hidden;box-shadow:0 2px 8px var(--shadow-
color);flex:1;min-height:0;}.section-header{padding:16px;border-
bottom:1px solid var(--stroke-color-primary);background:var(--bg-color-
operate);}.section-header h3{margin:0;font-size:16px;font-
weight:600;color:var(--text-color-primary);}.section-header span{font-
weight:400;color:var(--text-color-secondary);font-
size:14px;}.list{flex:1;min-height:0;overflow-y:auto;}.input{border-
top:1px solid var(--stroke-color-primary);flex-
shrink:0;height:48px;}@media (max-width:1200px){.container{grid-
template-columns:1fr;grid-template-rows:60% 20%
20%;gap:12px;}.sidebar{gap:12px;}.audience,.barrage{min-
height:200px;}}@media (max-width:768px)
{.container{padding:8px;gap:8px;grid-template-rows:50% 25%
25%;}.header,.section-header{padding:12px;}.sidebar{gap:8px;}}</style>
```

核心 API 参数说明

setSelfInfo

设置用户信息。

参数	类型	必填	说明
userName	String	是	用户昵称，将在直播间及聊天区域显示。

avatarUrl	String	否	用户头像的 URL 地址。

步骤4：启动项目

打开终端，进入项目目录，执行以下命令启动项目后，您就可以观看直播了。

> **说明：**
> 上述命令执行成功后，请在浏览器地址栏输入本地访问地址（您可以根据项目参考，例如 `http://localhost:5173/live-player`，具体端口号可能因您的项目配置而有所不同），即可看到观看页面。

```
npm run dev
```

播放直播流

步骤1：开启一场直播

- **方式一（推荐）：**
  使用我们提供的 在线开播网站 ，开启一场直播来观看，开启后获取直播间 ID
- **方式二：**
  集成我们提供的 主播开播（Web 桌面浏览器），开启一场直播来观看，开启后获取直播间 ID。

  > **重要提示：**
  > 请使用不同的**用户 ID** 开播和观看，否则后登录的设备会强制先登录的设备下线（即**被踢下线**）。

步骤2：观看直播

参考上述快速接入 步骤 3 的示例代码，输入对应 `liveId` 后即可进入直播间观看直播，观看效果如下图所示：

步骤3：路由配置

由于涉及到直播列表(或首页)到直播间的跳转逻辑，您需要配置 Vue Router。在项目 **src** 目录下新建 **router** 文件夹，并创建 **index.ts** 文件。然后，在您的主文件（例如 **main.ts** 或 **index.ts**）中引入并使用路由。可参见 GitHub 代码示例，如果需要直播列表，可参见文档 直播列表 。

> ⚠ **注意：**
> 如下代码示例中的 `live-player.vue` 文件即代表 步骤3 中创建的示例文件。

```
// src/router/index.ts
import { createRouter, createWebHistory } from 'vue-router';

const routes = [
  {
    path: '/live-player',
    component: () => import('../live-player.vue'),
  },
  // 如果需要直播列表功能，可添加如下路由，注意以下路径为您项目实际路径
  // {
    //   path: '/live-list',
    //   component: () => import('../live-list.vue'),
```

```
  // },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});
export default router;

// src/main.ts
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

const app = createApp(App);
app.use(router);
app.mount('#app');
```

自由定制

颜色主题及语言

通过配置 `App.vue` 中 `UIKitProvider` 的入参，修改主题及语言的默认值。

UIKitProvider 参数	可选值	默认值
theme	"light" | "dark"	"light"
language	"zh-CN" | "en-US"	-

```
<UIKitProvider theme="light">
  <router-view />
</UIKitProvider>

<script setup lang="ts">
import { UIKitProvider } from '@tencentcloud/uikit-base-component-vue3';
```

按钮 Button 和图标 Icon

若您需要对按钮 Button 或图标 Icon 等其他控件进行新增或替换等 UI 定制，您可以通过如下方式实现，以 `live-player.vue` 文件中的按钮和图标为例，您可以参考下图找到对应按钮或图标的指定位置源码，对当前部分的控

件进行**增加、删除、替换**等 UI 定制操作。

live-player.vue

```vue
<template>
  <UIKitProvider language="zh-CN" theme="dark">
    <div class="container">
      <!-- 直播核心区域 -->
      <section class="live">
        <header class="header">
          <IconArrowStrokeBack class="back-btn" size="20" />
          <Avatar :src="currentLive?.liveOwner.avatarUrl" :size="32" class="avatar" />
          <span class="user-name">{{ currentLive?.liveOwner.userName || currentLive?.liveOwner.userId }}</span>
        </header>
        <LiveCoreView class="player" />
      </section>

      <div class="sidebar">
        <!-- 在线观众列表 -->
        <section class="audience">
          <header class="section-header">
            <h3> 在线观众 <span>({{ audienceList.length }})</span></h3>
          </header>
          <LiveAudienceList class="list" />
        </section>

        <!-- 消息列表 & 消息输入框 -->
        <section class="barrage">
          <header class="section-header">
            <h3>消息列表</h3>
          </header>
          <BarrageList class="list" />
          <BarrageInput class="input" height="48px" />
        </section>
      </div>
    </div>
  </UIKitProvider>
</template>

<script setup lang="ts">
import { onMounted } from 'vue';
import { LiveAudienceList, BarrageList, BarrageInput, useLiveAudienceState, LiveCoreView, useLiveState, Avatar, useLoginState } from 'tuikit-atomicx-vue3';
import { UIKitProvider, IconArrowStrokeBack } from '@tencentcloud/uikit-base-component-vue3';

const { audienceList } = useLiveAudienceState();
const { currentLive } = useLiveState();
const { login } = useLoginState();

async function initLogin() {
  try {
    await login({
      sdkAppId: 0,        // SDKAppID, 可以参考步骤 1 获取
      userId: 'xxx',      // UserID, 可以参考步骤 1 获取
      userSig: 'xxx',     // userSig, 可以参考步骤 1 获取
    });
  } catch (error) {
    console.error('登录失败:', error);
  }
}

onMounted(async () => {
  await initLogin();
});
```

根据上述示例，我们同样支持您根据项目需求对**观众观看页面**进行 UI 定制的能力。除了页面 UI 布局调整，我们也支持您对**颜色主题、字体、圆角、按钮、图标、输入框、弹框等**内容进行**增加、删除、修改**等操作，满足您的 UI 定制需要。

类别	功能	描述
素材管理	自定义素材管理区域展示	支持调整 Icon 的大小、颜色或对 Icon 进行替换。
直播工具	自定义直播工具信息展示	支持调整 Icon 的大小、颜色或对 Icon 进行替换。
在线观众	自定义观众信息展示	**支持：** • 展示/隐藏观众等级。 • 观众信息字体、颜色 UI 自定义设置。

		• 替换为您需要的 Icon 风格。
消息列表	自定义消息弹幕区域展示	**支持：** • 展示/隐藏聊天输入区域。 • 支持 UI 定制聊天气泡风格、定制观众等级等内容。

常见问题

浏览器自动播放限制

出于用户体验考虑，现代浏览器对网页自动播放 (Autoplay) 功能实施了限制性策略：所有带声音的媒体内容必须经过用户主动交互（如点击或触摸）后才能播放。这项限制主要是为了防止网站在用户未明确同意的情况下突然播放音频，避免对用户造成干扰。大多数浏览器都不限制无声视频的自动播放，但是在低电量模式下的 iOS Safari 浏览器中以及开启了自定义自动播放限制的 iOS WKWebView 中（例如 iOS 微信浏览器），无声视频的自动播放也会受到限制。

自动播放失败表现

当用户对该站点的媒体互动指数 (MEI, Media Engagement Index) 未达到阈值时，企图自动播放有声视频会失败。在 SDK 默认情况下，当检测到自动播放失败时，会弹窗引导用户与页面产生交互（如点击**确认**按钮）。产生交互后，浏览器策略即被满足，有声视频即可正常播放。

解决方案：自定义处理自动播放失败

如果您希望自定义自动播放失败时的交互体验（例如替换默认的弹窗 UI），可以通过监听 SDK 抛出的 `onAutoplayFailed` 回调来实现。以下是在 Vue3 项目中，通过监听事件并弹出自定义对话框的实现代码示例：

```
import TUIRoomEngine, { TUIRoomEvents } from '@tencentcloud/tuiroom-
engine-js';
import { useUIKit, TUIMessageBox } from '@tencentcloud/uikit-base-
component-vue3';
import { useRoomEngine } from 'tuikit-atomicx-vue3';

const roomEngine = useRoomEngine();

let isShowAutoPlayDialog = false;

export default function useCustomizedAutoPlayDialog() {
  const { t } = useUIKit();
  TUIRoomEngine.once('ready', () => {
    roomEngine.instance?.on(TUIRoomEvents.onAutoPlayFailed, () => {
      if (!isShowAutoPlayDialog) {
```

```
          isShowAutoPlayDialog = true;
          TUIMessageBox.alert({
            title: t('RoomCommon.Attention'),
            content: t('RoomNotifications.AudioPlaybackFailed'),
            showClose: false,
            modal: false,
            confirmText: t('Confirm'),
            callback: () => {
              isShowAutoPlayDialog = false;
            },
          });
        }
      });
    });
}

export { useCustomizedAutoPlayDialog };
```

说明：
您需要安装对应的 `@tencentcloud/tuiroom-engine-js`。

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，您可以继续接入**直播列表**、**UI 自定义**和**监播**等功能：

功能	描述	集成指引
直播列表	展示直播列表界面和功能，包含直播列表和房间信息展示功能。	直播列表
UI 自定义	更详细的 UI 组件自定义指引。	概述
直播管理系统	运营平台，支持直播间管控。	直播管理系统（Vue3）

观众观看（Web Vue3 移动端浏览器）

最近更新时间：2026-06-04 17:38:20

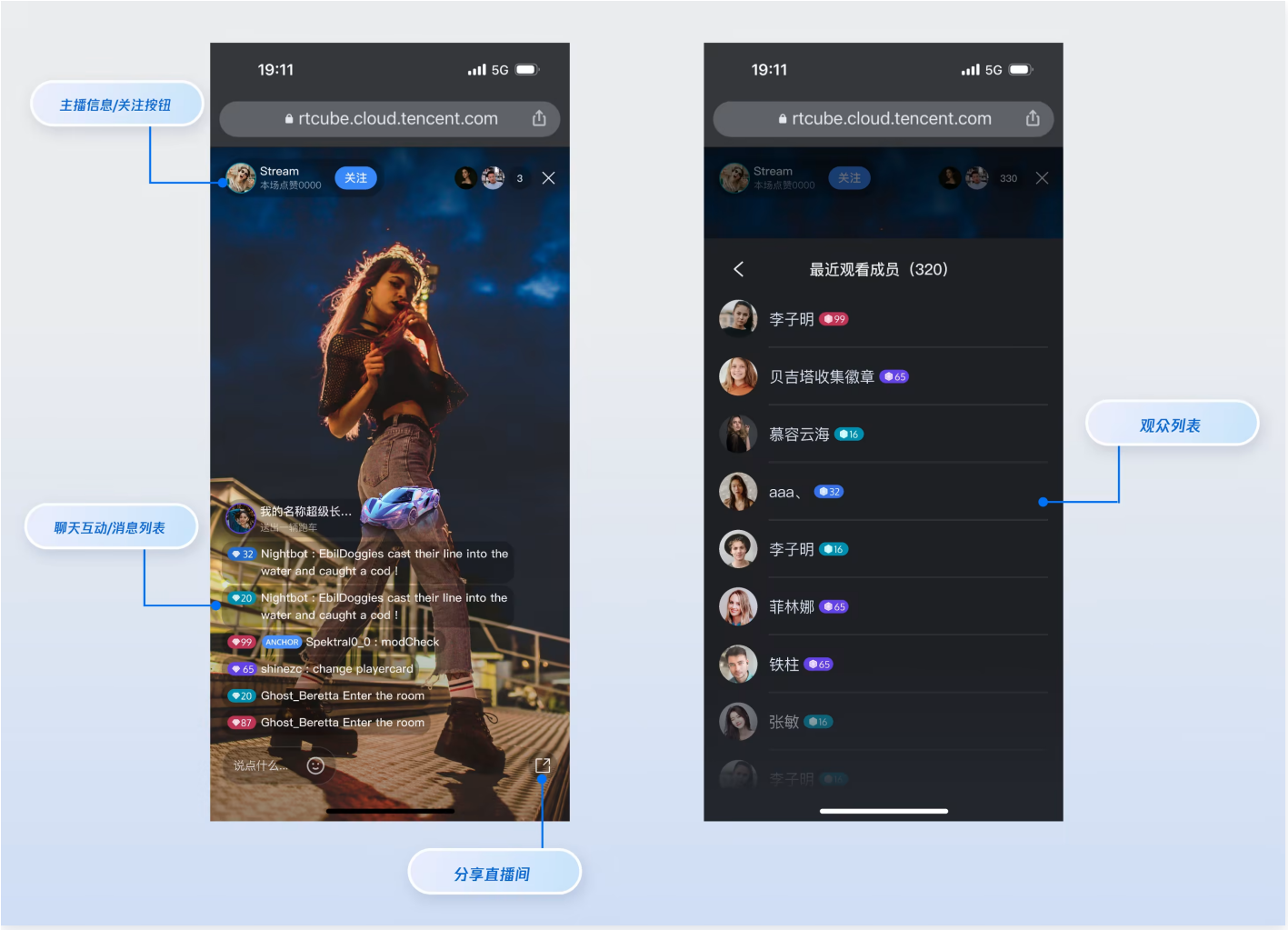
本文对 TUILiveKit Demo 中的**观看页面**进行了详细的介绍，您可以在已有项目中直接参考本文档集成我们开发好的观看页面，也可以根据您的需求按照文档中的内容对页面的样式，布局以及功能项进行深度的定制。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的观众观看方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建观众端界面，请参考 观众观看 (Core SDK)。

功能展示

观看页面提供默认行为和风格，但如果默认行为和样式不能完全满足您的需求，您也可以对 UI 进行自定义。图中显示的数字与特定功能列表中的类别相对应。其中主要包含直播**信息展示、视频直播区域、在线观众、音视频操作、直播时长、画面分辨率切换、全屏功能、聊天互动、消息列表等。**

- **直播视频播放**：高清流畅的直播体验。
- **弹幕互动**：实时弹幕交流。
- **观众列表**：查看在线观众信息。
- **全屏播放**：沉浸式观看体验。
- **H5 移动平台支持**：跨平台兼容。



快速接入

步骤1：环境配置及开通服务

在进行快速接入之前，请参考 准备工作 集成组件并实现登录。

步骤2：安装依赖

您可以选择以下任一方式安装依赖：

npm

```
npm install tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
--save
```

pnpm

```
pnpm add tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
```

yarn

```
yarn add tuikit-atomicx-vue3 @tencentcloud/uikit-base-component-vue3
```

步骤3：观看页面接入

在您的项目下创建 **live-player.vue** 文件，可直接复制如下代码至您的项目中集成**观看页面**。

> ⚠ **注意：**
> 您可以直接复制如下代码至您的工程中集成示例工程，也可以访问 观众观看 地址，查看更加详细的源码内容。

```
<template>
  <UIKitProvider theme="dark">
    <div class="container">
      <!-- 直播核心区域 -->
      <section class="live">
        <header class="header">
          <IconArrowStrokeBack class="back-btn" size="20" />
          <Avatar :src="currentLive?.liveOwner.avatarUrl" :size="32"
class="avatar" />
          <span class="user-name">{{ currentLive?.liveOwner.userName ||
currentLive?.liveOwner.userId }}</span>
        </header>
        <LiveCoreView class="player" />
      </section>

      <div class="sidebar">
        <!-- 在线观众列表 -->
        <section class="audience">
          <header class="section-header">
            <h3> 在线观众 <span>({{ audienceList.length }})</span></h3>
          </header>
          <LiveAudienceList class="list" />
        </section>

        <!-- 消息列表 & 消息输入框 -->
        <section class="barrage">
```

```
          <header class="section-header">
            <h3>消息列表</h3>
          </header>
          <BarrageList class="list" />
          <BarrageInput class="input" height="48px" />
        </section>
      </div>
    </div>
  </UIKitProvider>
</template>

<script setup lang="ts">
import { onMounted } from 'vue';
import { LiveAudienceList, BarrageList, BarrageInput,
useLiveAudienceState, LiveCoreView, useLiveListState, Avatar,
useLoginState } from 'tuikit-atomicx-vue3';
import { UIKitProvider, IconArrowStrokeBack } from '@tencentcloud/uikit-
base-component-vue3';

const { audienceList } = useLiveAudienceState();
const { currentLive } = useLiveListState();
const { login, setSelfInfo } = useLoginState();

const liveId = 'live_xxxx'  // 填入您要观看的直播间的ID

async function initLogin() {
  try {
    await login({
      sdkAppId: 0,      // SDKAppID, 可以参考步骤 1 获取
      userId: '',       // UserID, 可以参考步骤 1 获取
      userSig: '',      // userSig, 可以参考步骤 1 获取
    });
  } catch (error) {
    console.error('登录失败:', error);
  }
}

onMounted(async () => {
  await initLogin();
  await setSelfInfo({
```

```
    userName: '我的名字/昵称',     // 用户名
    avatarUrl: '',                // 头像 URL 地址
  });
  await joinLive({ liveId });
});

</script>

<style>html,body,#app{height:100%;width:100%;margin:0;padding:0;overflow
:hidden}:global(body){font-size:15px;line-height:1.6;text-
rendering:optimizeLegibility;}:global(*),:global(*::before),:global(*::a
fter){box-sizing:border-box;margin:0;}.container{display:grid;grid-
template-columns:1fr
320px;height:100%;width:100%;gap:16px;padding:16px;background:var(--bg-
color-default);box-sizing:border-
box;overflow:hidden;}.live{display:flex;flex-
direction:column;background:var(--bg-color-operate);border-
radius:12px;overflow:hidden;box-shadow:0 2px 8px var(--shadow-
color);}.header{display:flex;align-
items:center;gap:12px;padding:16px;border-bottom:1px solid var(--stroke-
color-primary);}.back-btn{cursor:pointer;color:var(--text-color-
tertiary);transition:color 0.2s;}.back-btn:hover{color:var(--text-color-
link-hover);}.avatar{border:1px solid var(--uikit-color-white-7);}.user-
name{color:var(--text-color-primary);font-
weight:500;}.player{flex:1;background:var(--uikit-color-black-1);min-
height:0;}.sidebar{display:flex;flex-
direction:column;gap:16px;height:100%;overflow:hidden;}.audience{display
:flex;flex-direction:column;background:var(--bg-color-operate);border-
radius:12px;overflow:hidden;box-shadow:0 2px 8px var(--shadow-
color);flex:1;min-height:0;}.barrage{display:flex;flex-
direction:column;background:var(--bg-color-operate);border-
radius:12px;overflow:hidden;box-shadow:0 2px 8px var(--shadow-
color);flex:1;min-height:0;}.section-header{padding:16px;border-
bottom:1px solid var(--stroke-color-primary);background:var(--bg-color-
operate);}.section-header h3{margin:0;font-size:16px;font-
weight:600;color:var(--text-color-primary);}.section-header span{font-
weight:400;color:var(--text-color-secondary);font-
size:14px;}.list{flex:1;min-height:0;overflow-y:auto;}.input{border-
top:1px solid var(--stroke-color-primary);flex-
shrink:0;height:48px;}@media (max-width:1200px){.container{grid-
```

```
template-columns:1fr;grid-template-rows:60% 20%
20%;gap:12px;}.sidebar{gap:12px;}.audience,.barrage{min-
height:200px;}}@media (max-width:768px)
{.container{padding:8px;gap:8px;grid-template-rows:50% 25%
25%;}.header,.section-header{padding:12px;}.sidebar{gap:8px;}}</style>
```

核心 API 参数说明

setSelfInfo

设置用户信息。

参数	类型	必填	说明
userName	String	是	用户昵称，将在直播间及聊天区域显示。
avatarUrl	String	否	用户头像的 URL 地址。

步骤4：启动项目

打开终端，进入项目目录，执行以下命令启动项目后，您就可以观看直播了。

> **说明：**
> 上述命令执行成功后，请在浏览器地址栏输入本地访问地址（您可以根据项目配置参考，例如 `http://localhost:5173/live-player`，具体端口号可能因您的项目配置而有所不同），即可看到观看页面。

```
npm run dev
```

播放直播流

步骤1：开启一场直播

- **方式一（推荐）：**
  使用我们提供的 在线开播网站，开启一场直播来观看，开启后获取直播间 ID
- **方式二：**
  集成我们提供的 主播开播（Web 桌面浏览器），开启一场直播来观看，开启后获取直播间 ID。

> **重要提示：**
> 请使用不同的**用户 ID** 开播和观看，否则后登录的设备会强制先登录的设备下线（即“被踢下线”）。

步骤2：观看直播

参考上述快速接入中的 步骤 3 的示例代码，输入对应 `liveId` 后即可进入直播间观看直播：

> **说明：**
> 另外，若您需要修改个人信息，请参考 步骤 3 示例代码中的 `setSelfInfo` 进行个人信息的设置。

步骤3：路由配置

由于涉及到直播列表(或首页)到直播间的跳转逻辑，您需要配置 Vue Router。在项目 **src** 目录下新建 **router** 文件夹，并创建 **index.ts** 文件。然后，在您的主文件（例如 **main.ts** 或 **index.ts**）中引入并使用路由。可参见 GitHub 上的代码示例，如果需要直播列表，可参见文档 直播列表 。

> **注意：**
> 如下代码示例中的 `live-player.vue` 文件即代表 步骤3 中创建的示例文件。

```
// src/router/index.ts
import { createRouter, createWebHistory } from 'vue-router';

const routes = [
  {
    path: '/live-player',
    component: () => import('../live-player.vue'),
  },
  // 如果需要直播列表功能，可添加如下路由，注意以下路径为您项目实际路径
  // {
    //   path: '/live-list',
    //   component: () => import('../live-list.vue'),
  // },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});
export default router;

// src/main.ts
import { createApp } from 'vue';
import App from './App.vue';
```

```
import router from './router';

const app = createApp(App);
app.use(router);
app.mount('#app');
```

自由定制

颜色主题及语言

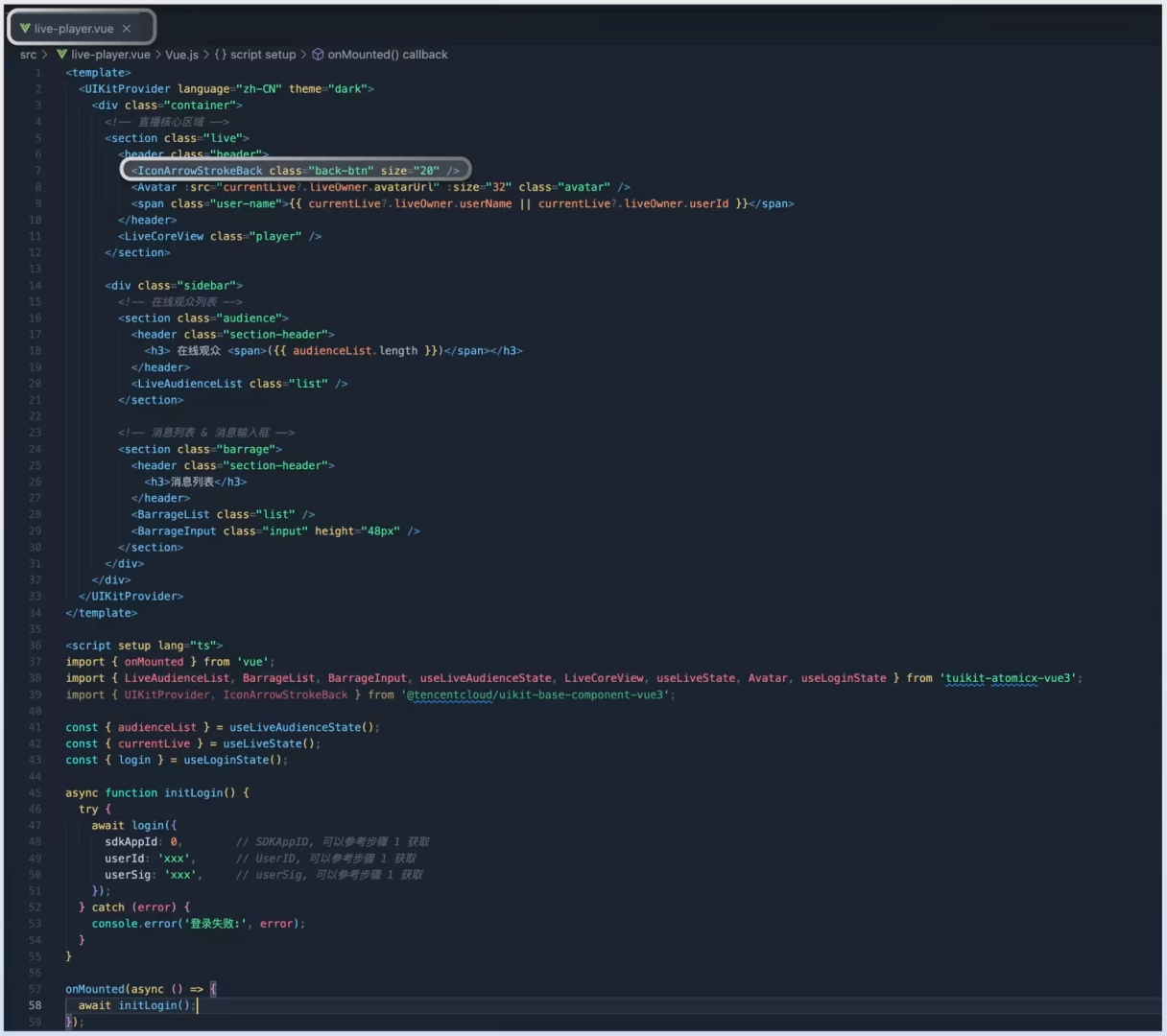
通过配置 `App.vue` 中 `UIKitProvider` 的入参，修改主题及语言的默认值。

UIKitProvider 参数	可选值	默认值
theme	"light" | "dark"	"light"
language	"zh-CN" | "en-US"	-

```
<UIKitProvider theme="light">
  <router-view />
</UIKitProvider>

<script setup lang="ts">
import { UIKitProvider } from '@tencentcloud/uikit-base-component-vue3';
```

按钮 Button 和图标 Icon

若您需要对按钮 Button 或图标 Icon 等其他控件进行新增或替换等 UI 定制，您可以通过如下方式实现，以 `live-player.vue` 文件中的按钮和图标为例，您可以参考下图找到对应按钮或图标的指定位置源码，对当前部分的控件进行增加、删除、替换等 UI 定制操作。



根据上述示例，我们同样支持您根据项目需求对**观众观看页面**进行 UI 定制的能力。除了页面 UI 布局调整，我们也支持您对**颜色主题、字体、圆角、按钮、图标、输入框、弹框等**内容进行**增加、删除、修改**等操作，满足您的 UI 定制需要。

类别	功能	描述
素材管理	自定义素材管理区域展示	支持调整 Icon 的大小、颜色或对 Icon 进行替换。
直播工具	自定义直播工具信息展示	支持调整 Icon 的大小、颜色或对 Icon 进行替换。
在线观众	自定义观众信息展示	**支持：** • 展示/隐藏观众等级。 • 观众信息字体、颜色 UI 自定义设置。

		• 替换为您需要的 Icon 风格。
消息列表	自定义消息弹幕区域展示	**支持：** • 展示/隐藏聊天输入区域。 • 支持 UI 定制聊天气泡风格、定制观众等级等内容。

常见问题

浏览器自动播放限制

出于用户体验考虑，现代浏览器对网页自动播放（autoplay）功能实施了限制性策略：所有带声音的媒体内容必须经过用户主动交互（例如点击或触摸）后才能播放。这项限制主要是为了防止网站在用户未明确同意的情况下突然播放音频，避免对用户造成干扰。大多数浏览器都不限制无声视频的自动播放，但是在低电量模式下的 iOS Safari 浏览器中以及开启了自定义自动播放限制的 iOS WKWebView 中（例如 iOS 微信浏览器），无声视频的自动播放也会受到限制。

自动播放失败表现

当用户对该站点的媒体互动指数（MEI, Media Engagement Index）未达到阈值时，企图自动播放有声视频会失败。在 SDK 默认情况下，当检测到自动播放失败时，会弹窗引导用户与页面产生交互（例如点击**确认**按钮）。产生交互后，浏览器策略即被满足，有声视频即可正常播放。

解决方案：自定义处理自动播放失败

如果您希望自定义自动播放失败时的交互体验（例如替换默认的弹窗 UI），可以通过监听 SDK 抛出的 `onAutoplayFailed` 回调来实现。以下是在 Vue3 项目中，通过监听事件并弹出自定义对话框的实现代码示例：

```
import TUIRoomEngine, { TUIRoomEvents } from '@tencentcloud/tuiroom-
engine-js';
import { useUIKit, TUIMessageBox } from '@tencentcloud/uikit-base-
component-vue3';
import { useRoomEngine } from 'tuikit-atomicx-vue3';

const roomEngine = useRoomEngine();

let isShowAutoPlayDialog = false;

export default function useCustomizedAutoPlayDialog() {
  const { t } = useUIKit();
  TUIRoomEngine.once('ready', () => {
    roomEngine.instance?.on(TUIRoomEvents.onAutoPlayFailed, () => {
      if (!isShowAutoPlayDialog) {
```

```
          isShowAutoPlayDialog = true;
          TUIMessageBox.alert({
            title: t('RoomCommon.Attention'),
            content: t('RoomNotifications.AudioPlaybackFailed'),
            showClose: false,
            modal: false,
            confirmText: t('Confirm'),
            callback: () => {
              isShowAutoPlayDialog = false;
            },
          });
        }
      });
    });
}

export { useCustomizedAutoPlayDialog };
```

说明：
您需要安装对应的 `@tencentcloud/tuiroom-engine-js`。

下一步

恭喜您，现在您已经成功集成了观众观看。接下来，您可以继续接入 **直播列表**、**UI 自定义**和**监播**等功能：

功能	描述	集成指引
直播列表	展示直播列表界面和功能，包含直播列表和房间信息展示功能。	直播列表
UI 自定义	更详细的 UI 组件自定义指引。	概述
直播管理系统	运营平台，支持直播间管控。	直播管理系统（Vue3）

观众观看（Web React 桌面端浏览器）

最近更新时间：2026-04-20 17:34:02

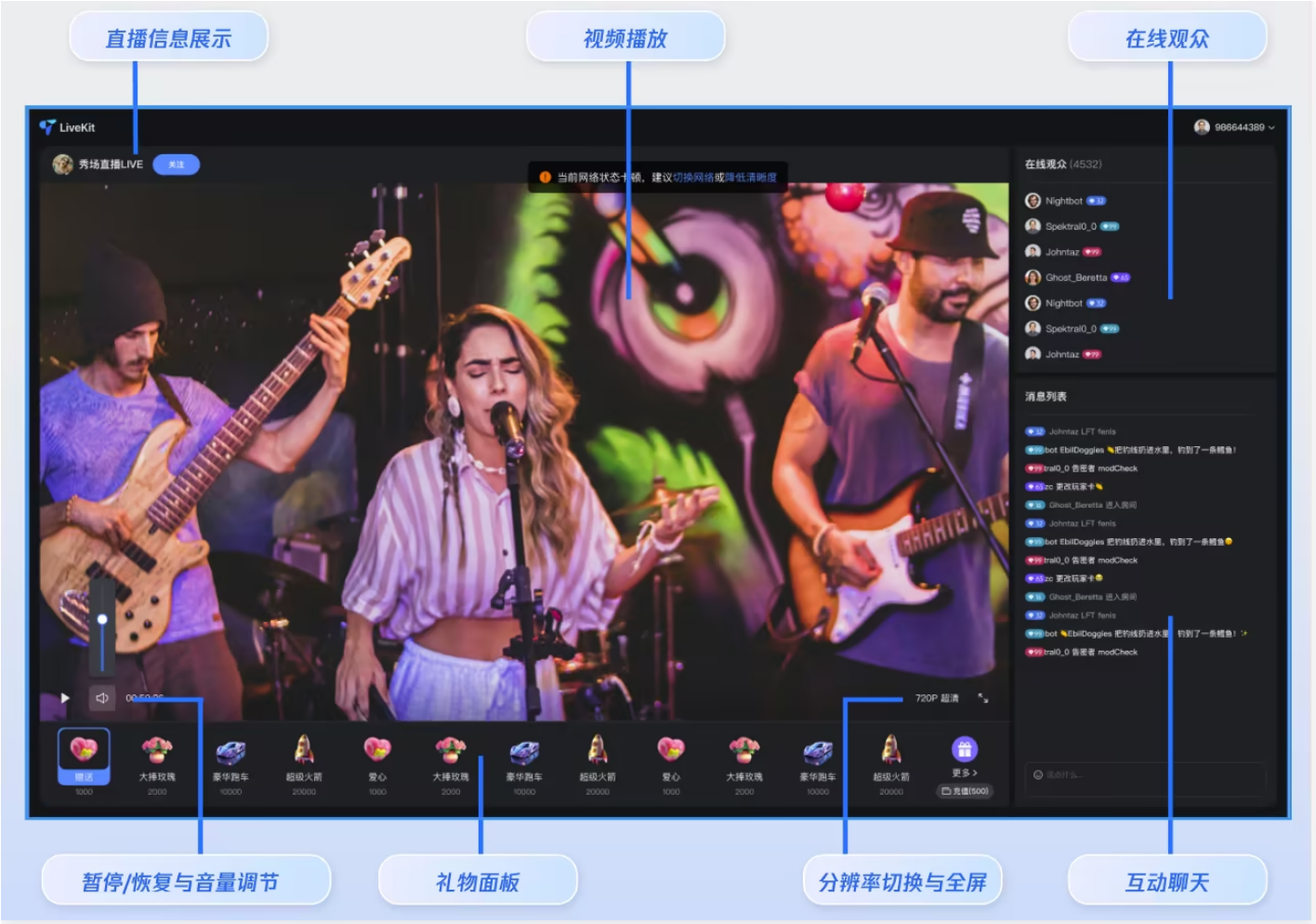
本文将详细介绍 TUILiveKit React Demo 中的观看页面，引导您在自己的项目中集成 React 观看页面，并对页面的样式、功能及布局进行深度定制。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的观众观看方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建观众端界面，请参考 观众观看 (Core SDK)。

功能展示

默认的观看页面效果如下图所示，主要包括：**直播信息展示、视频播放、礼物面板、在线观众、互动聊天、播放控制**等功能。

- **视频播放**：高清流畅的直播体验，同时支持多分辨率切换。
- **互动聊天**：支持实时弹幕消息，发送文字、表情消息。
- **礼物面板**：支持发送礼物，打赏主播。
- **播放控制**：支持暂停、恢复、分辨率切换、音量调节、画中画播放、全屏播放。



快速接入

步骤1：环境配置及开通服务

在进行快速接入之前，请参考 准备工作（Web React） 集成组件并实现登录。

步骤2：安装依赖

您可以选择以下任一方式安装依赖：

npm

```
npm install tuikit-atomicx-react @tencentcloud/uikit-base-component-react --save
npm install sass --save-dev
```

pnpm

```
pnpm add tuikit-atomicx-react @tencentcloud/uikit-base-component-react
```

```
pnpm add sass --dev
```

yarn

```
yarn add tuikit-atomicx-react @tencentcloud/uikit-base-component-react
yarn add sass --dev
```

步骤3：观看页面接入

在您的项目下创建 **LivePlayerView.tsx** 和 **LivePlayerView.module.scss** 文件，可直接复制如下代码至您的项目中集成**观看页面**。

> ⚠ **注意：**
> 您可以直接复制如下代码至您的工程中集成示例工程，也可以访问 观众观看 地址，查看更加详细的源码内容。

LivePlayerView.tsx

```
import React, { useEffect, useCallback } from 'react';
import { useSearchParams, useNavigate } from 'react-router-dom';
import TUIRoomEngine, { TUIRoomEvents } from "@tencentcloud/tuiroom-
engine-js";
import { Avatar, LiveView, LiveGift, LiveListEvent, BarrageList,
BarrageInput, LiveAudienceList, useLiveListState, useLiveAudienceState,
useLoginState, useRoomEngine } from 'tuikit-atomicx-react';
import { UIKitProvider, IconChevronLeft, MessageBox, Dialog, useUIKit }
from '@tencentcloud/uikit-base-component-react';
import styles from './LivePlayerView.module.scss';

interface LivePlayerProps {
  className?: string;
}

const LivePlayer: React.FC<LivePlayerProps> = ({ className }) => {
  const { t } = useUIKit();
  const navigate = useNavigate();
  const roomEngine = useRoomEngine();
```

```
  const { currentLive, leaveLive, subscribeEvent, unsubscribeEvent } =
useLiveListState();
  const { audienceCount } = useLiveAudienceState();

  const handleAutoPlayFailed = useCallback(() => {
    MessageBox.alert({
      content: '内容已准备就绪，点击【播放】按钮开始播放',
      confirmText: '播放',
      showClose: false,
      modal: false,
    });
  }, []);

 const handleKickedOutOfLive = useCallback(() => {
    Dialog.open({
      content: '你已被踢出直播间',
      confirmText: '确认',
      className: styles.livePlayer__liveDialog,
      showCancel: false,
      showClose: false,
      modal: true,
      center: true,
      onConfirm: () => {
        Dialog.close();
        // 这里可以添加您自己的业务逻辑，如跳转到首页或直播列表页
      },
      onClose: () => {
        // 这里可以添加您自己的业务逻辑，如跳转到首页或直播列表页
      },
    });
  }, [navigate]);


  const handleLiveEnded = useCallback(() => {
    Dialog.open({
      content: '直播已结束',
      confirmText: '确认',
      className: styles.livePlayer__liveDialog,
      showCancel: false,
      showClose: false,
```

```
      modal: true,
      center: true,
      onConfirm: () => {
        Dialog.close();
        // 这里可以添加您自己的业务逻辑，如跳转到首页或直播列表页
      },
      onClose: () => {
        // 这里可以添加您自己的业务逻辑，如跳转到首页或直播列表页
      },
    });
  }, [navigate]);

  const handleLeaveLive = useCallback(async () => {
    try {
      await leaveLive();
      navigate('/live-list');
    } catch (error) {
      console.error('Failed to leave live:', error);
      MessageBox.alert({
        content: '离开直播间失败，请重试',
        confirmText: '确认',
        showClose: false,
        modal: true,
      });
    }
  }, [leaveLive, navigate]);

  // 设置事件监听
  useEffect(() => {
    // 监听自动播放失败事件监听，浏览器默认禁止音频播放，自动播放失败时，增加一次 UI
交互触发音频播放
    if (roomEngine.instance) {
      roomEngine.instance.on(TUIRoomEvents.onAutoPlayFailed,
handleAutoPlayFailed);
    } else {
      TUIRoomEngine.once("ready", () => {
        roomEngine.instance?.on(TUIRoomEvents.onAutoPlayFailed,
handleAutoPlayFailed);
      });
    }
```

```
    // 监听直播结束事件
    subscribeEvent(LiveListEvent.ON_LIVE_ENDED, handleLiveEnded);
    // 监听被主播踢出直播间事件
    subscribeEvent(LiveListEvent.ON_KICKED_OUT_OF_LIVE,
handleKickedOutOfLive);

    return () => {
      roomEngine.instance?.off(TUIRoomEvents.onAutoPlayFailed,
handleAutoPlayFailed);
      unsubscribeEvent(LiveListEvent.ON_LIVE_ENDED, handleLiveEnded);
      unsubscribeEvent(LiveListEvent.ON_KICKED_OUT_OF_LIVE,
handleKickedOutOfLive);
    };
  }, [handleAutoPlayFailed, handleLiveEnded, handleKickedOutOfLive,
roomEngine.instance, subscribeEvent, unsubscribeEvent]);

  return (
    <div className={`${styles.livePlayer} ${className || ''}`}>
      <div className={styles.livePlayer__left}>
        <div className={styles.livePlayer__header}>
          <div className={styles.livePlayer__headerContent}>
            <IconChevronLeft
              className={styles.livePlayer__headerChevronLeft}
              size="32"
              onClick={handleLeaveLive}
            />
            <Avatar
              className={styles.livePlayer__headerAvatar}
              src={currentLive?.liveOwner?.avatarUrl}
              size={32}
            />
            <span>{currentLive?.liveOwner?.userName ||
currentLive?.liveOwner?.userId}</span>
          </div>
        </div>
        <div className={styles.livePlayer__player}>
          <LiveView />
        </div>
        <div className={styles.livePlayer__giftContainer}>
```

```
          <LiveGift />
        </div>
      </div>
      <div className={styles.livePlayer__right}>
        <div className={styles.livePlayer__audienceList}>
          <div className={styles.livePlayer__audienceListTitle}>
            <span>{t('观众列表')} </span>
            <span className={styles.livePlayer__audienceCount}>
({audienceCount})</span>
          </div>
          <div className={styles.livePlayer__audienceListContent}>
            <LiveAudienceList height="100%" />
          </div>
        </div>
        <div className={styles.livePlayer__messageList}>
          <div className={styles.livePlayer__messageListTitle}>
            <span>{t('消息列表')}</span>
          </div>
          <div className={styles.livePlayer__messageListContent}>
            <BarrageList />
            <BarrageInput />
          </div>
        </div>
      </div>
    </div>
  );
};

const LivePlayerView: React.FC = () => {
  const [searchParams] = useSearchParams();
  const { loginUserInfo, login, setSelfInfo } = useLoginState();
  const { joinLive } = useLiveListState();

  useEffect(() => {
    // 方式1：从 URL 参数获取（推荐用于页面跳转场景）
    const liveId = searchParams.get('liveId') || '';

    // 方式2：从组件 Props 获取（如果将 LivePlayerView 作为子组件使用）
    // const liveId = props.liveId || '';
```

```
    // 方式3：硬编码用于测试（请替换为实际的直播间 ID）
    // const liveId = 'your_live_room_id';

    if (liveId) {
      joinLive({ liveId });
    }
  }, [searchParams, joinLive]);

  const initLogin = useCallback(async () => {
    try {
      await login({
        SDKAppID: 0,     // 请替换为您的 SDKAppID（服务开通时获取）
        userID: '',      // 请替换为您的用户 ID
        userSig: '',     // 请替换为您的用户签名（详细获取方式请参阅【步骤1：环境
配置及开通服务】文档）
      });
      await setSelfInfo({
        userName: '',    // 用户昵称，会显示在成员列表、聊天消息中。不设置昵称时，
将显示用户 ID
        avatarUrl: '',   // 用户头像，必须为完整的 URL 图片地址，例如：
https://your.domain.com/avatar-default.png
      });
    } catch (error) {
      console.error('登录失败:', error);
    }
  }, [login, setSelfInfo]);

  useEffect(() => {
    async function init() {
      await initLogin();
    }

    if (!loginUserInfo?.userId) {
      init();
    } else {
      console.log('[LiveList]用户已登录:', loginUserInfo.userId);
    }
  }, [initLogin, loginUserInfo?.userId]);

  return (
```

```
    <UIKitProvider theme="dark" language='zh-CN'>
      <div className={styles.livePlayerView}>
        <div className={styles.livePlayerView__body}>
          <LivePlayer />
        </div>
      </div>
    </UIKitProvider>
  );
};

export default LivePlayerView
```

LivePlayerView.module.scss

```
@mixin text-size-16 {
  font-size: 16px;
  font-weight: 600;
}

@mixin text-size-12 {
  font-size: 12px;
  font-weight: 400;
}

@mixin text-size-14 {
  font-size: 14px;
  font-weight: 400;
}

@mixin text-size-24 {
  font-size: 24px;
  font-weight: 500;
}

@mixin scrollbar {
  &::-webkit-scrollbar {
    width: 6px;
    background: transparent;
  }
```

```
  &::-webkit-scrollbar-track {
    background: transparent;
  }

  &::-webkit-scrollbar-thumb {
    background: var(--uikit-color-gray-3);
    border-radius: 3px;
    border: 2px solid transparent;
    background-clip: padding-box;

    &:hover {
      background: var(--uikit-color-gray-3);
    }
  }
}

.livePlayerView {
  display: flex;
  flex-direction: column;
  width: 100%;
  height: 100%;
  padding: 16px;
  background-color: var(--uikit-bg-color-topbar);
  color: var(--uikit-text-color-primary);

  .livePlayerView__header {
    width: 100%;
    padding-bottom: 16px;
  }

  .livePlayerView__body {
    flex: 1;
    display: flex;
    flex-direction: column;
    width: 100%;
    overflow: auto;
    align-items: center;
  }
}
```

```
.livePlayer {
  display: flex;
  width: 100%;
  height: 100%;
  border-radius: 8px;
  overflow: hidden;
  @include scrollbar;

  .livePlayer__left {
    display: flex;
    flex-direction: column;
    flex: 1;
    min-width: 0;
    margin-right: 8px;
    overflow: hidden;

    .livePlayer__header {
      width: 100%;
      height: 56px;
      flex-shrink: 0;
      padding: 0 16px;
      background: var(--uikit-bg-color-operate);

      .livePlayer__headerContent {
        display: flex;
        align-items: center;
        width: 100%;
        height: 100%;
        border-bottom: 1px solid var(--uikit-stroke-color-primary);

        span {
          @include text-size-16;
        }
      }

      .livePlayer__headerChevronLeft {
        cursor: pointer;
      }
```

```
      .livePlayer__headerAvatar {
        margin: 0 8px;
        border: 1px solid var(--uikit-color-white-7);
      }
    }

    .livePlayer__player {
      width: 100%;
      flex: 1;
      min-height: 0;
      background: var(--uikit-bg-color-topbar);
    }

    .livePlayer__giftContainer {
      width: 100%;
      height: 130px;
      flex-shrink: 0;
      border-top: 1px solid var(--uikit-stroke-color-primary);
      background: var(--uikit-bg-color-operate);
    }
  }

  .livePlayer__right {
    display: flex;
    flex-direction: column;
    height: 100%;
    width: 20%;
    min-width: 160px;
    max-width: 360px;

    .livePlayer__audienceList {
      display: flex;
      flex-direction: column;
      flex-shrink: 0;
      height: 30%;
      padding: 8px;
      background: var(--uikit-bg-color-operate);

      .livePlayer__audienceListTitle {
        padding: 12px 0;
```

```
        border-bottom: 1px solid var(--uikit-stroke-color-primary);
        @include text-size-16;
      }

      .livePlayer__audienceCount {
        font-weight: 400;
        color: var(--uikit-text-color-secondary);
      }

      .livePlayer__audienceListContent {
        flex: 1;
        overflow: hidden;
      }
    }

    .livePlayer__messageList {
      display: flex;
      flex-direction: column;
      flex: 1 0 auto;
      margin-top: 8px;
      padding: 8px;
      background: var(--uikit-bg-color-operate);

      .livePlayer__messageListTitle {
        padding: 12px 0;
        border-bottom: 1px solid var(--uikit-stroke-color-primary);
        @include text-size-16;
      }

      .livePlayer__messageListContent {
        display: flex;
        flex: 1;
        flex-direction: column;
      }
    }
  }
}

.livePlayer__liveDialog {
  text-align: center;
```

```
}
```

步骤4：路由配置

如果您是通过首页或者直播列表页面，跳转到直播观看页面，您需要配置 React Router 页面路由。在项目中新建或修改 `src/router/index.tsx` 文件。然后，在您的主文件（例如 main.tsx 或 App.tsx）中引入并使用路由。可参见 GitHub 代码示例，如果需要直播列表，可参见文档 直播列表（Web React）。

```
// src/router/index.tsx
import { createHashRouter } from 'react-router-dom';
import { LiveListView } from '../views/LiveList';
import { LivePlayerView } from '../views/LivePlayer';

// 路由保护组件
const ProtectedRoute = ({ children }: { children: React.ReactNode; }) =>
{
  return (
    <>{children}</>
  );
};

const routes = [
  {
    path: '/live-player',
    element: <LivePlayerView />,
  },
  // // 如果需要直播列表功能，可添加如下路由，接入直播列表页面
  // // 接入文档，请参阅【直播列表 -> 直播列表（Web React）】
  // {
  //  path: '/live-list',
  //  element: <LiveListView />,
  //}
];

export const router = createHashRouter(
  routes.map(route => ({
    ...route,
    element: <ProtectedRoute>{route.element}</ProtectedRoute>,
  }))
```

```
);

// 在 src/App.tsx 中使用路由组件 src/router/index.tsx
import { RouterProvider } from 'react-router-dom'
import { router } from './router'
import './App.css'

function App() {
  return (
    <RouterProvider router={router} />
  )
}

export default App
```

步骤5：启动项目

打开终端，进入项目目录，执行以下命令启动项目。

```
npm run dev
```

播放直播流

步骤1：开启一场直播

- **方式一（推荐）：**
  使用我们提供的 在线开播网站 ，开启一场直播来观看，开启后获取直播间 ID
- **方式二：**
  跑通其它端的 Demo 工程，开启一场直播，例如： 跑通 Web Vue3 推流与观看 。

> ⚠ **重要提示：**
> 请使用不同的**用户 ID** 开播和观看，否则后登录的设备会强制先登录的设备下线（即**被踢下线**）。

步骤2：观看直播

参考上述 快速接入 步骤3 的示例代码，输入**登录账号**和**直播间 ID** 后，即可进入直播间观看直播，观看效果如下图所示：

自由定制

颜色主题及语言

你可以使用 `UIKitProvider` 组件，修改默认的主题和语言。

UIKitProvider 参数	可选值	默认值
theme	"light" | "dark"	"light"
language	"zh-CN" | "en-US"	-

1. 在 App.tsx 中全局设置

在 App.tsx 中，以 `UIKitProvider` 作为根元素。

```
import { RouterProvider } from 'react-router-dom'
import { UIKitProvider } from '@tencentcloud/uikit-base-component-
react'
import { router } from './router'
import './App.css'

function App() {
```

```
  return (
    <UIKitProvider theme="dark" language='zh-CN'>
      <RouterProvider router={router} />
    </UIKitProvider>
  );
}

export default App
```

2. **在单个页面或组件中设置**

在 React 组件中，以 `UIKitProvider` 作为根节点元素。以下示例代码截取自 快速接入 步骤3 的代码片段。

```
const LivePlayerView: React.FC = () => {
  return (
    <UIKitProvider theme="dark" language='zh-CN'>
      <div className={styles.livePlayerView}>
        <div className={styles.livePlayerView__body}>
          <LivePlayer />
        </div>
      </div>
    </UIKitProvider>
  );
};

export default LivePlayerView;
```

设置昵称和头像

上文中，快速接入：步骤3 已包含昵称和头像设置代码，如下面代码所示，设置的头像和昵称，将显示在自己和其它直播间成员的成员列表、聊天消息中。如果不主动设置昵称和头像，昵称将显示为登录时的用户 ID、头像将显示为默认头像。

```
const initLogin = useCallback(async () => {
  try {
    await login({
      SDKAppID: 0,     // 请替换为您的 SDKAppID（服务开通时获取）
      userID: '',      // 请替换为您的用户 ID
```

```
        userSig: '',     // 请替换为您的用户签名（详细获取方式请参阅【步骤1：环境
配置及开通服务】文档）
      });
      await setSelfInfo({
        userName: '',    // 用户昵称，会显示在成员列表、聊天消息中。不设置昵称时，
将显示用户 ID
        avatarUrl: '',   // 用户头像，必须为完整的 URL 图片地址，例如：
https://your.domain.com/avatar-default.png
      });
    } catch (error) {
      console.error('登录失败:', error);
    }
  }, [login, setSelfInfo]);
```

常见问题

浏览器自动播放限制

出于用户体验考虑，现代浏览器对网页自动播放 (Autoplay) 功能实施了限制性策略：所有带声音的媒体内容必须经过用户主动交互（如点击或触摸）后才能播放。这项限制主要是为了防止网站在用户未明确同意的情况下突然播放音频，避免对用户造成干扰。大多数浏览器都不限制无声视频的自动播放，但是在低电量模式下的 iOS Safari 浏览器中以及开启了自定义自动播放限制的 iOS WKWebView 中（如 iOS 微信浏览器），无声视频的自动播放也会受到限制。

自动播放失败表现

当用户对该站点的媒体互动指数 (MEI, Media Engagement Index) 未达到阈值时，企图自动播放有声视频会失败。在 SDK 默认情况下，当检测到自动播放失败时，会弹窗引导用户与页面产生交互（如点击**确认**按钮）。产生交互后，浏览器策略即被满足，有声视频即可正常播放。

解决方案：自定义处理自动播放失败

如果您希望自定义自动播放失败时的交互方式（例如替换默认的弹窗 UI），可以通过监听 SDK 抛出的 `onAutoplayFailed` 回调来实现。以下是 **快速接入 步骤3** 中的代码片段，通过监听事件并弹出自定义对话框提示用户。

```
import React, from 'react';
import TUIRoomEngine, { TUIRoomEvents } from "@tencentcloud/tuiroom-
engine-js";
import { useLiveListState, useRoomEngine } from 'tuikit-atomicx-react';
import { MessageBox } from '@tencentcloud/uikit-base-component-react';
import styles from './LivePlayerView.module.scss';
```

```
const LivePlayer: React.FC<LivePlayerProps> = ({ className }) => {
  const roomEngine = useRoomEngine();
  const { currentLive, leaveLive, subscribeEvent, unsubscribeEvent } =
useLiveListState();

  // 自动播放失败事件处理函数
  const handleAutoPlayFailed = useCallback(() => {
    MessageBox.alert({
      content: '内容已准备就绪，点击【播放】按钮开始播放',
      confirmText: '播放',
      showClose: false,
      modal: false,
    });
  }, []);

  // 设置事件监听
  useEffect(() => {
    // 监听自动播放失败事件监听，浏览器默认禁止音频播放，自动播放失败时，增加一次 UI
交互触发音频播放
    if (roomEngine.instance) {
      roomEngine.instance.on(TUIRoomEvents.onAutoPlayFailed,
handleAutoPlayFailed);
    } else {
      TUIRoomEngine.once("ready", () => {
        roomEngine.instance?.on(TUIRoomEvents.onAutoPlayFailed,
handleAutoPlayFailed);
      });
    }

    // ... 省略其它代码

    return () => {
      roomEngine.instance?.off(TUIRoomEvents.onAutoPlayFailed,
handleAutoPlayFailed);
      // ... 省略其它代码
    };
  }, [handleAutoPlayFailed, handleLiveEnded, handleKickedOutOfLive,
roomEngine.instance, subscribeEvent, unsubscribeEvent]);

  //... 省略其它代码
```

```
}
```

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，您可以继续接入**直播列表**、**UI 自定义**和**监播**等功能：

功能	描述	集成指引
直播列表	展示直播列表界面和功能，包含直播列表和房间信息展示功能。	直播列表（Web React）
UI 自定义	更详细的 UI 组件自定义指引。	UI 自定义
直播管理系统	运营平台，支持直播间管控。	直播管理系统（React）

观众观看（Flutter）

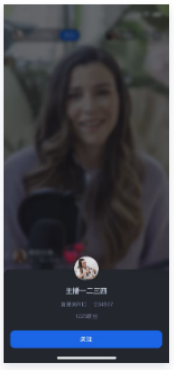
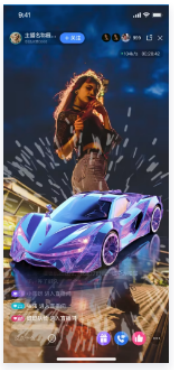
最近更新时间：2026-04-20 17:34:02

TUILiveKit 的观众观看页为用户提供了丰富且便捷的直播观看与互动功能，能快速集成到您的应用中，满足观众多样化的需求。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的观众观看方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建观众端界面，请参考 观众观看 (Core SDK)。

功能概览

- **直播观看：**清晰、流畅观看主播实时直播画面。
- **互动连麦：**申请连麦，与主播进行音视频互动。
- **直播信息：**查看直播间标题、简介，及在线观众列表等信息。
- **直播互动：**送礼物（有动画特效，主播接收提示）、点赞（有动画及实时统计）、发弹幕互动。

直播观看	互动连麦	直播信息	直播互动
			

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：添加观众观看页面

`TUILiveRoomAudienceWidget` 已内置了直播场景的观众端完整 UI 与业务逻辑，该组件不支持浮窗模式，如需浮窗功能请转到 添加浮窗版观众观看页面。您只需要配置 `TUILiveRoomAudienceWidget` 的调用入口（具体由

您的业务决定），执行如下操作，拉起观众观看页面或将观众观看页面集成到自己的 Widget 树中：

直接跳转

```
import 'package:tencent_live_uikit/tencent_live_uikit.dart';

// 跳转到观众观看页面
Navigator.push(context, MaterialPageRoute(
  builder: (BuildContext context) {
      final roomId = "test_live_room_id";
      return TUILiveRoomAudienceWidget(roomId: roomId);
}));
```

集成到 Widget 树

```
// --- 根据您的Widget树结构，选择以下一种方式集成 ---

// [选项一] 作为唯一子Widget（单子树）
// 适用于Container、Padding等通常只包含一个子Widget的容器
Container(
    child: TUILiveRoomAudienceWidget(roomId: roomId) // 在此处集成观众观看页
)

// [选项二] 作为多个子Widget之一（多子树）
// 适用于Column、Row、Stack等可以包含多个子Widget的布局
Stack(
    children: [
      YourOtherWidget(), // 您的其他子Widget
      TUILiveRoomAudienceWidget(roomId: roomId), // 在此处集成观众观看页
      YourOtherWidget(), // 您的其他子Widget
])
```

集成后，调用代码即可拉起观众页面并开始播放直播内容，效果如 功能概览 的观看直播图所示。

步骤4：（可选）添加浮窗版观看页面

`TUILiveRoomAudienceOverlay` 是支持浮窗模式的观看页面。在观看直播期间，可以切换到 **应用内** 和 **应用外（即画中画）** 2 种场景的浮窗模式。`TUILiveRoomAudienceOverlay` 是基于 Flutter 官方 API **Overlay** 和原生画中画实现的，具体接入流程如下：

1. App 工程配置开启系统画中画特性

参考 工程配置，开启系统画中画特性。

2. 跳转浮窗版观看页面

在您需要观众进房的调用入口（具体由您的业务决定），执行如下操作，跳转到观看页面。

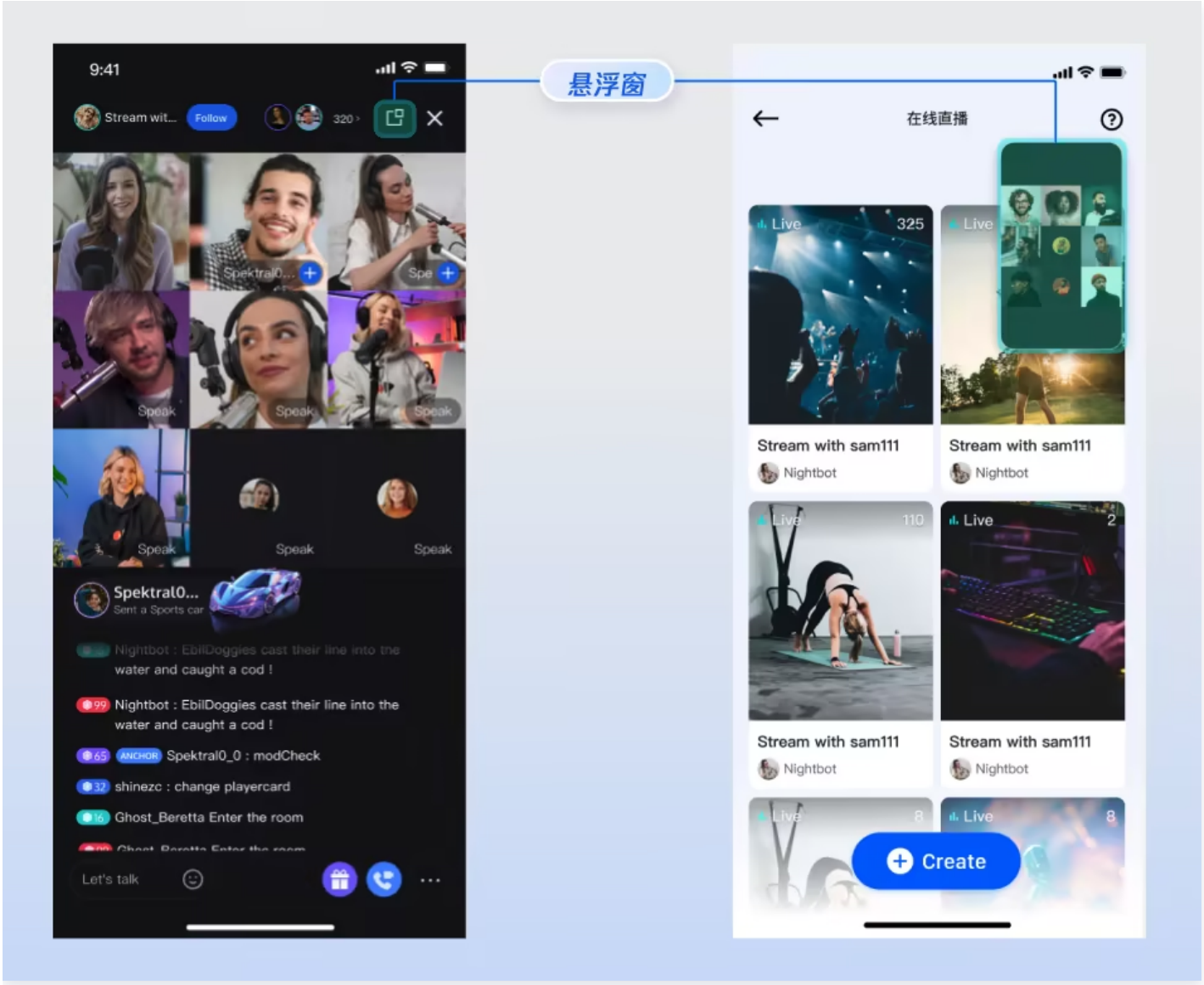
```
import 'package:tencent_live_uikit/tencent_live_uikit.dart';

// 跳转到观看页面
Navigator.push(context, MaterialPageRoute(
  builder: (BuildContext context) {
     final roomId = "test_live_room_id";
     return TUILiveRoomAudienceOverlay(roomId: roomId);
}));
```

说明：

1. `TUILiveRoomAudienceOverlay` 不支持作为子 widget 嵌入到容器类 widget（例如：Container、Stack 等），只能作为独立页面跳转。因为内部使用了 Overlay，LiveKit 需要操控整个 Overlay 页面来切换浮窗模式。
2. iOS 端仅支持普通观众进入应用外浮窗模式。

进入应用内浮窗模式

点击观看页面右上角的浮窗按钮，进入浮窗模式，效果如图所示：



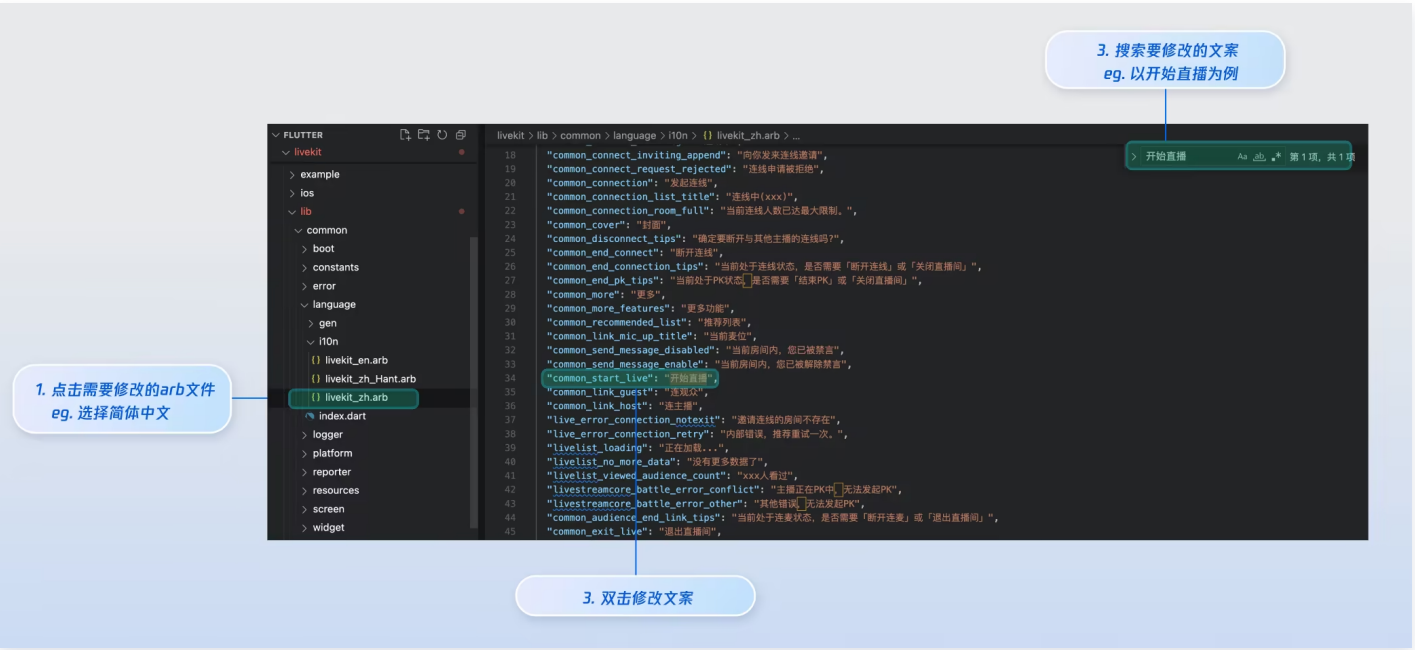
进入应用外浮窗模式

依次点击观看页面右下角的**更多 > 画中画**，打开画中画开关。然后，您的 App 退到后台时，将自动进入到系统画中画模式，效果如图所示：

自定义您的界面布局

文案定制

TUILiveKit 使用 ARB 文件和 Flutter 标准国际化方案来管理 UI 文案的显示。您可以直接编辑 `livekit/lib/common/language/i10n/` 目录下的 ARB 文件来修改需要调整的文案：

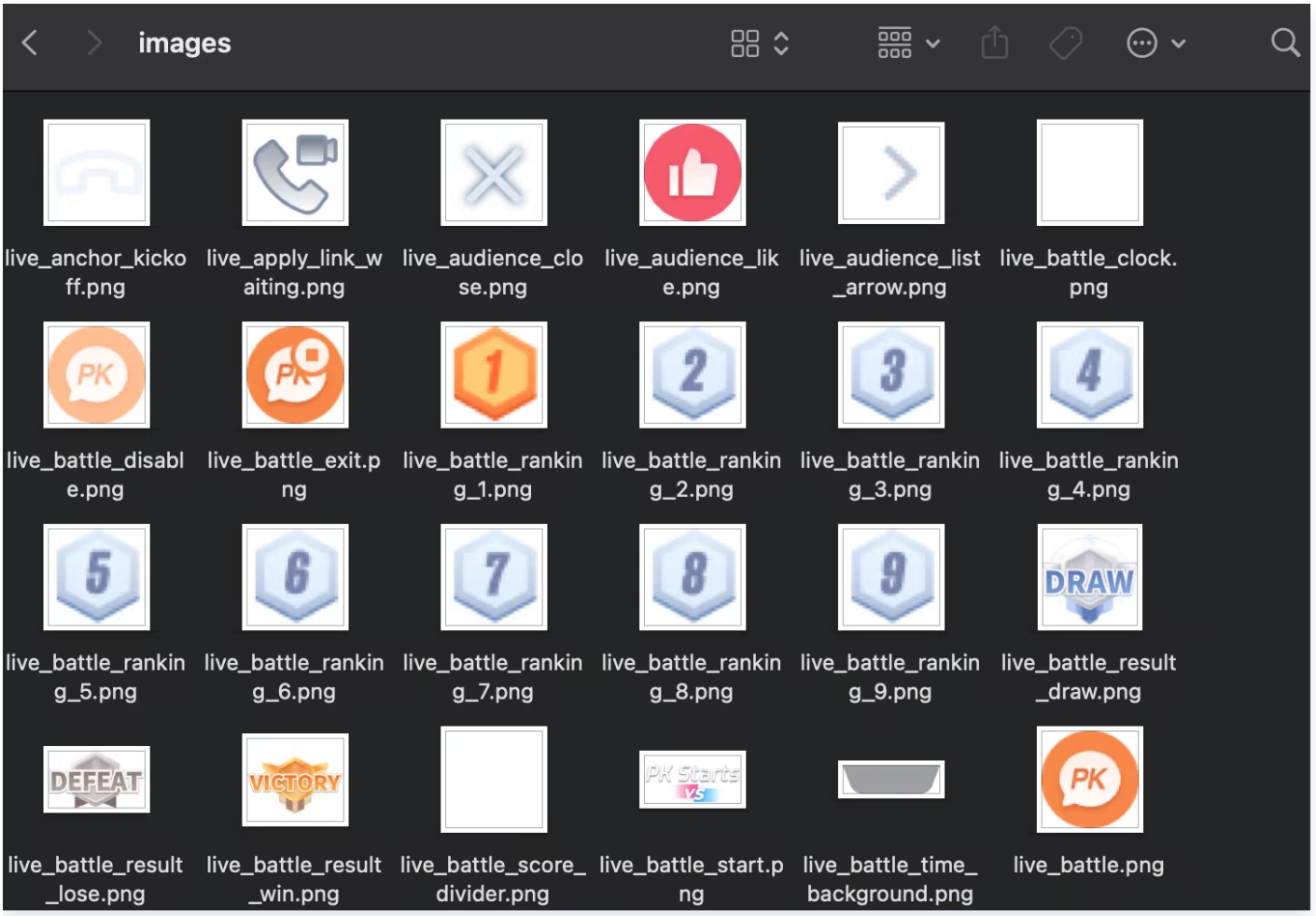


- `livekit_en.arb`：英文文案。
- `livekit_zh.arb`：简体中文文案。
- `livekit_zh_Hant.arb`：繁体中文文案。

修改后通过命令行执行 `flutter gen-l10n` 重新生成本地化代码即可。生成成功后，`livekit/lib/common/language/gen/` 目录里的代码文件将会刷新。

图标定制

TUILiveKit UI 所需的图片资源在 `livekit/assets/images/` 目录下管理，您可以直接替换该目录下的 PNG 图片文件来快速修改自定义界面所需的图标。



重新构建并运行应用，即可看到更新的图标。

下一步

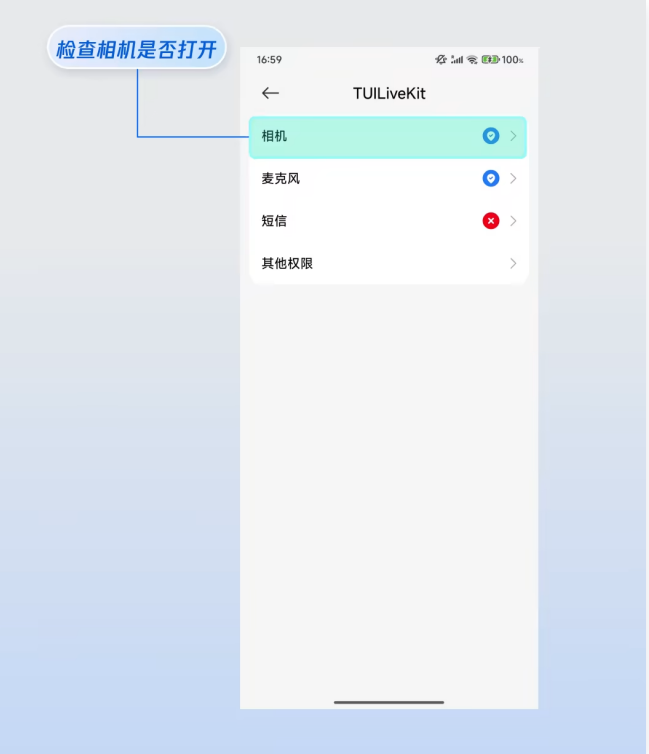
恭喜您，现在您已经成功集成了**观众观看**功能。接下来，您可以实现**主播开播**、**直播列表**等功能，可参考下表：

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

观众选择视频连麦后，视频画面显示黑屏?

请前往**手机设置** > **App** > **相机**，检查摄像头权限是否开启，可参考下图：

iOS	Android
	

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。

App 退到后台无法进入应用外浮窗（画中画）模式?

参考 进入应用外浮窗模式，进入直播间后，开启了画中画开关。
如果是 Android 平台，在您打开画中画开关后，还会自动检测 App 的系统画中画模式开关状态，如果没有打开，App 会自动跳转到设置页面，请您允许进入画中画模式：


16:57
画中画
app-uikit
版本 1.0.0
允许进入画中画模式
允许此应用在您打开或离开应用（如继续观看视频）时创建画中画窗口。该窗口会显示在您当前使用的其他应用的上层。
请确保打开画中画开关

观众观看（uni-app 客户端）

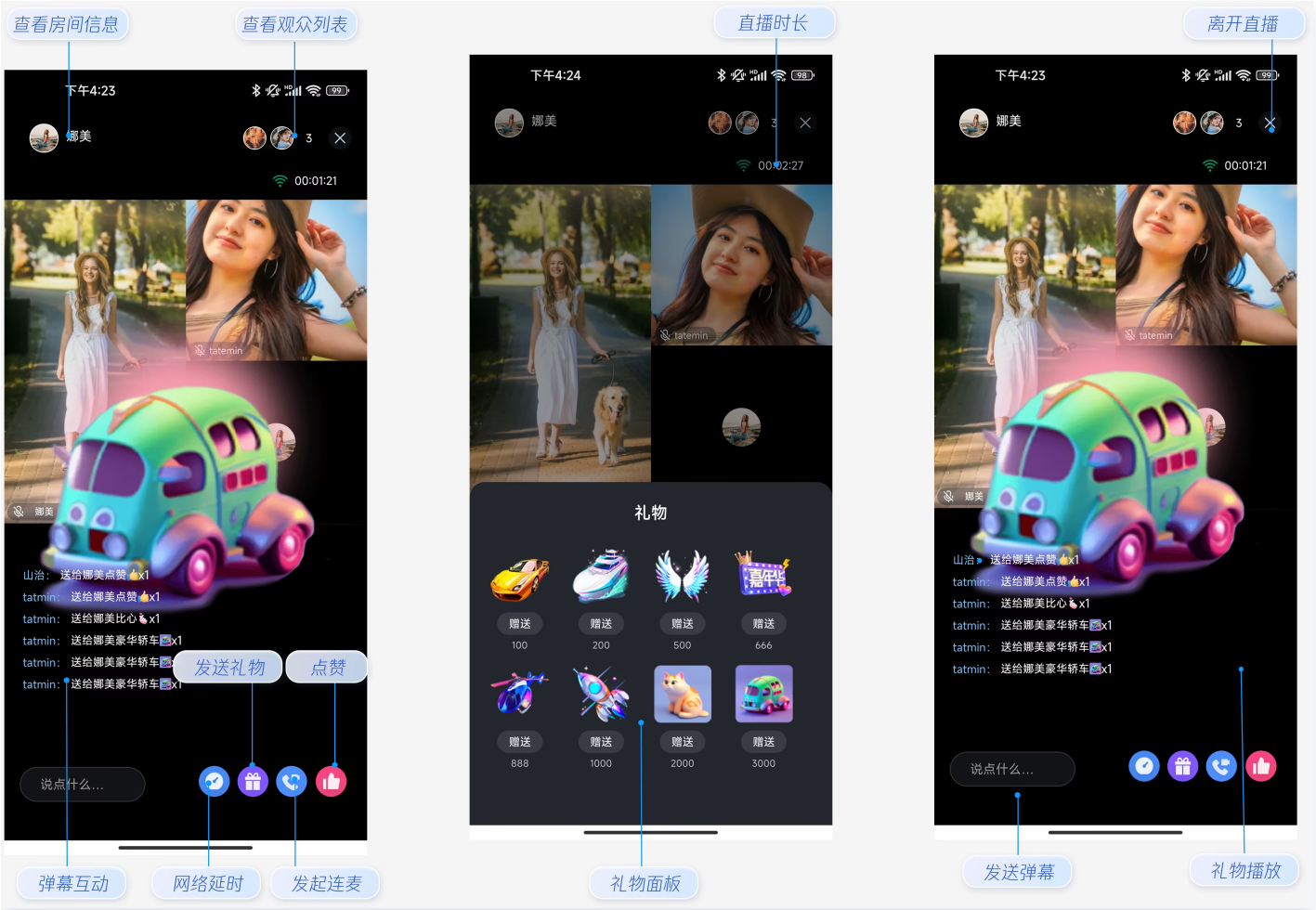
最近更新时间：2026-04-20 17:34:02

本文将指导您如何快速集成观众观看页，体验观众进入主播的直播间后观看直播，观众连麦 、查看直播间信息、查看在线观众、发送礼物、点赞，弹幕互动等功能。

> **集成提示：**
> 本文档介绍基于 **UI 组件** 的观众观看方式（含完整交互 UI）。若您需通过 **Core SDK** 自行搭建观众端界面，请参考 观众观看 (Core SDK)。

功能预览

观众观看页提供默认的行为和样式，但如果默认行为和样式不能完全满足您的需求，您也可以对 UI 进行自定义。



开始接入

步骤1：集成核心文件

已集成核心文件请忽略，已经按照 代码集成指引 将 TUILiveKit 的 `pages` 、`uni_modules` 等核心文件拷贝到了您的项目中。

步骤2：完成登录

已登录请忽略该步骤，您的应用已经调用了 登录接口 并成功登录。这是使用所有功能的基础

步骤3：制作并使用自定义基座

已制作自定义基座请忽略，由于直播功能依赖原生插件，您必须在 HBuilderX 中为您的项目制作并使用自定义调试基座来运行。

步骤4：注册观众观看页面

现在，您需要在 `pages.json` 文件中告诉您的应用，这个新页面是存在的。
打开您项目根目录下的 `pages.json` 文件，在 `"pages"` 数组中，添加以下对象来注册观众观看页。

```
{
    "pages": [
        // ... 您项目已有的其他页面配置
        {
           "path": "pages/scenes/live/audience/index",
            "style": {
                "navigationBarTextStyle": "white",
                "app-plus": {
                    "titleNView":false
                 }
            }
        },
    ]
    // ... 其他配置
}
```

步骤5：跳转到观众观看页面

在您需要观众观看的地方（具体由您的业务决定，在其点击事件里执行），调用 `uni.navigateTo` 或 `uni.redirectTo` 方法，即可进入观众观看页面。

```
// 示例：在一个按钮的点击事件中
function startBroadcast() {
  uni.$liveID = liveID //您要加入的直播间 Id;
```

```
  joinLive({
    liveID: uni.$liveID,
    success: () => {
      uni.redirectTo({
        url: '/pages/scenes/live/audience/index', // URL 对应 pages.json
中配置的 path
      });
    }
  });
}
```

定制您的 UI

替换观众观看页图标

TUILiveKit 用到的所有图标都存放于 `static/images` 目录下, 部分示例如下，您可以根据您的诉求来替换目录下的图标

图标路径	详细描述
/static/images/dashboard.png	底部操作栏的“仪表盘”图标
/static/images/link-guest.png	底部操作栏的“申请连麦”图标
/static/images/live-gift.png	底部操作栏的“礼物”图标
/static/images/live-like.png	底部操作栏的“点赞”图标
/static/images/close.png	顶部操作栏的“离开直播间”图标

设置字体的大小/颜色

uniapp 官方对 nvue 页面限制字体的大小和颜色只能在 `text` 标签上设置，对于您想修改的文案，直接修改其 `css` 样式即可，下方以“主播昵称”文字为例：

```
<text class="stream-title" :numberOfLines="1">{{
currentLive?.liveOwner?.userName || currentLive?.liveOwner?.userId}}
</text>
  .stream-title {
    color: #ffffff;
    font-size: 28rpx;
    font-weight: 500;
    margin-bottom: 4rpx;
```

```
    width: 120rpx;
    height: 40rpx;
    lines: 1;
  }
```

修改其 `css` 样式：

```
<text class="stream-title" :numberOfLines="1">{{
currentLive?.liveOwner?.userName || currentLive?.liveOwner?.userId}}
</text>
  .stream-title {
    color: rgba(155, 108, 255, 1);
    font-size: 40rpx;
    font-weight: 500;
    margin-bottom: 4rpx;
    width: 120rpx;
    height: 40rpx;
    lines: 1;
  }
```

最终效果：

设置按钮的大小

设置按钮的大小，也可以直接通过修改 `css` 属性来进行设置，下方以 “礼物” 按钮为例：

```
<image class="action-btn" @click="showGiftPicker()"
src="/static/images/live-gift.png" />
  .action-btn {
    width: 64rpx;
    height: 64rpx;
    margin-left: 16rpx;
  }
```

修改其 `css` 样式：

```
<image class="action-btn" @click="showGiftPicker()"
src="/static/images/live-gift.png" />
  .action-btn {
    width: 80rpx;
    height: 80rpx;
    margin-left: 16rpx;
  }
```

最终效果：

隐藏按钮

隐藏按钮可以通过注释代码的方式来进行直接隐藏，下方以“礼物”的按钮为例：

```
<!-- <image class="action-btn" @click="showGiftPicker()"
src="/static/images/live-gift.png" /> -->
```

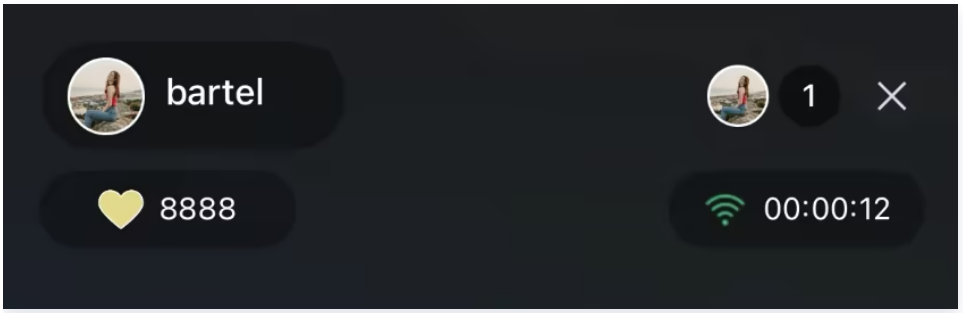
新增按钮

在您需要新增的位置，插入对应的按钮实现，下方以视频区域左上角新增“点赞数量”按钮为例：

```
<view class="live-content">
    ......
    <view class="live-like-container" >
      <image style="width: 36rpx; height: 36rpx;"
src="/static/images/gift_heart.png" alt="" />
      <text style="color: #fff; font-size: 24rpx;">8888</text>
    </view>
</view>
  .live-like-container {
    position: fixed;
    top: 200rpx;
    left: 30rpx;
    width: 200rpx;
    height: 60rpx;
    background-color: rgba(0, 0, 0, 0.3);
```

```
    border-radius: 45rpx;
    flex-direction: row;
    display: flex;
    justify-content: center;
    align-items: center;
  }
```

最终效果：



常见问题

如何展示观众等级?

您的业务中若是有需要展示观众等级的需求，可以在目标位置插入对应的元素用于展示观众等级，这里以观众列表展示观众等级为例。在 `TUIKit_uni-app/components/atomic-x/LiveAudienceList.nvue` 组件中找到观众信息展示的代码，并在其中插入观众等级展示代码。

```
<view class="audience-info">
   <text>{{audience.level}}</text>// 新增观众等级展示代码
   <view class="audience-avatar-container">// 观众头像展示代码
      <image class="audience-avatar" :src="audience.avatarUrl ||
defaultAvatarUrl" mode="aspectFill" />
   </view>
</view>
```

其他参考文档

- TUILiveKit uni-app 主播开播页
- TUILiveKit uni-app 直播列表页